UNIVERSITÉ DE MONTRÉAL

Conception et développement d'un environnement favorisant l'apprentissage des concepts fondamentaux de la programmation

par

Yves Boudreault

Département de didactique

Faculté des sciences de l'éducation

Thèse présentée à la Faculté des études supérieures

En vue de l'obtention du grade de

Philosophiae Doctor (Ph.D.)

En didactique des sciences

Novembre 2003

UNIVERSITÉ DE MONTRÉAL

Faculté des études supérieures

Cette thèse intitulée :

Conception et développement d'un environnement favorisant l'apprentissage des concepts fondamentaux de la programmation

présentée par :

Yves Boudreault

a été évaluée par un jury composé des personnes suivantes :

France Caron
présidente rapporteur

Pierre Nonnon
Directeur de recherche

Aude Dufresne
Membre du jury

Martin Gagnon

Examinateur extérieur

Jacques Bélair

Représentant du doyen de la FES

RÉSUMÉ

L'Environnement Informatique pour l'Apprentissage Humain (EIAH) réalisé dans le cadre de cette recherche de développement se compose d'exercices de programmation à l'aide d'un montage électronique robotisé et d'un tutoriel. Notre principale motivation était la recherche de nouveaux moyens qui permettraient de rejoindre l'ensemble de notre clientèle étudiante fortement hétérogène. Pour ce faire, nous avons conçu notre EIAH de manière à proposer des moyens différents et diversifiés favorisant l'apprentissage de la programmation. Notre étude nous a dévoilé que ces derniers moyens permettent de créer de nouveaux rapports aux savoirs et au savoir-faire puisque l'environnement englobe différents concepts liés à l'apprentissage tels, l'apprentissage individualisé, l'approche socioconstructiviste, la pédagogie de projets, l'apprentissage collaboratif et l'apprentissage par analogie. Les deux composantes de cet EIAH peuvent être utilisées conjointement, simultanément ou séparément en temps opportun. Cela nous permet d'envisager la composition de plusieurs situations d'apprentissage d'ordre didactique ou a-didactique permettant des rapports aux savoirs variés. Les prototypes matériels et logiciels utilisés ont été évalués puis modifiés au niveau fonctionnel avant de les proposer à nos étudiants.

Mots clés : didactique, programmation informatique, EIAH, tutoriel, sciences et technologie,

SUMMARY

The Intelligent Educative System (IES) realized in this development research content programming exercises with electronic automated material and a tutorial. Our principal motivation was to search the new possibilities that would allow rejoining all students in class whom strongly heterogeneous. For this, we conceived our IES to propose different and diversified facilities favouring programming learning. Our study unveiled us that the facilities allow creating new reports to the knowledge and to the know-how since environment includes different concepts linked to learning such, individualized learning, socio constructivist approach, projects pedagogy, collaborative learning and learning by analogy. The two parts of IES can be used jointly, simultaneously or separately in timely time. That allows us to propose the composition of several teaching Brousseau's situations such "*didactique*" and "*a-didactique*" allowing reports to the varied knowledge. The material and software prototypes were evaluated then modified at the functional level before propose them to our students.

Words keys: educational, computer programming, IES, tutoriel, sciences and technology,

TABLE DES MATIÈRES

Listes des figures

LISTE DES TABLEAUX

DÉDICACE

À mon amour,
Jacynthe

À mes enfants,
Charles, Francis et Mathieu

À mes sœurs et mes frères,
Luc, Gilles, Hélène, Anne, Marc, Rémi, Martin et Lilie

À mes parents, disparus trop jeunes,
Marie-France et Léonard

Avec vous je partage ma fierté d'avoir réalisé ce rêve.

REMERCIEMENTS

Entreprendre des études doctorales signifie s'engager pour plusieurs années. Entreprendre des études doctorales à quarante ans nécessite, en plus des études, de concilier à la fois vie familiale et travail.

Mes premiers remerciements s'adressent à ma famille. À Jacynthe, merveilleuse Jacynthe, sans ton appui ce travail n'aurait pu se concrétiser. J'apprécie et savoure tous les jours de la vie en ta présence. À Charles, Francis et Mathieu je vous dis merci, merci pour votre patience, votre compréhension envers un père parfois impatient. Par votre seule présence, vous me ramenez à l'essentiel… Vive les repas houleux en famille !

J'ai bénéficié de conditions particulièrement avantageuses tout au long de mes études. En ce sens, je tiens à remercier chaleureusement l'École Polytechnique de Montréal qui a accepté de m'octroyer une période de perfectionnement de deux ans et demi. J'ai également une pensée toute spéciale pour mes collègues professeurs qui ont prévu et obtenu ces conditions.

Je m'en voudrais, et probablement lui aussi m'en voudrait, de ne pas mentionner l'appui indéfectible de mon directeur de recherche. J'ai beaucoup appris en le côtoyant. D'une rigueur scientifique exemplaire, il est à la fois profondément humain. Comme il fut agréable de travailler avec toi, Pierre. Je te remercie mille fois et te souhaite le titulariat avant la retraite...

J'en profite également pour remercier tous les étudiants du LRP et plus particulièrement mon libanais préféré, Georges. Leurs encouragements ont été grandement appréciés.

Finalement, merci à Christian et Wacef qui m'ont grandement motivé à entreprendre ce doctorat. Les « boys », un gros merci pour toutes les expériences partagées.

AVANT PROPOS

Enseignant la programmation depuis plus de dix ans, nous sommes constamment à la recherche de moyens et de méthodes qui permettront non pas à un sous-ensemble d'étudiants d'acquérir ses savoirs et ses savoir-faire mais bien à l'ensemble des étudiants. Souvent perçu comme une utopie, nous aimerions bien un jour voir ce rêve se réaliser. Nous avons donc décidé de retrousser nos manches et rechercher de nouveaux moyens qui permettront une évolution ou une amélioration de notre enseignement actuel afin qu'il réponde mieux aux besoins de tous nos étudiants.

Puisque l'objectif de nos travaux de recherche est principalement de nature ontologique, nous en profiterons pour explorer à travers des considérations théoriques plusieurs disciplines sous l'angle de l'apprentissage et de l'enseignement. C'est ainsi que dans notre recherche de développement nous examinerons dans des champs qui pourraient, à première vue, sembler disparates tels le génie logiciel, l'analogie, l'enseignement par projet et la didactique des théories pouvant nous mener à des pistes de solution.

Cette recherche se situe dans le champ disciplinaire de la didactique de l'informatique où les chercheurs s'affairent à déterminer la méthode de l'enseignement de l'informatique. Nos travaux nous conduirons dans le domaine de la robotique pédagogique prônant traditionnellement deux approches. Dans la première approche l'ordinateur peut être employé comme outil d'apprentissage, par exemple l'Expérimentation Assistée par Ordinateur (ExAO). Dans la deuxième approche, l'ordinateur est un objet d'apprentissage en technologie, soit par la conception, la réalisation de robots pédagogiques ou la programmation. C'est dans cette deuxième approche que nous situons notre idée d'un montage électronique robotisé que nous exploiterons comme un objet pour penser permettant de rendre concret certains concepts abstraits de la programmation. Le montage devrait permettre d'amenuiser les difficultés dues à la compréhension du domaine de l'application et d'offrir des indicateurs d'erreurs supplémentaires. En effet, lors de son utilisation par l'étudiant, nous avons prévu des manipulations permettant de découvrir et de déterminer les spécifications du programme

à rédiger. L'étudiant initiera de cette façon sa solution en identifiant les opérations propres à l'algorithme qui sera traduit ensuite en programme. À l'exécution du programme, l'étudiant pourra facilement vérifier son adéquation avec le comportement attendu sur le montage. Les anomalies seront alors des indicateurs d'erreurs probables.

Les technologies de l'information et des communications (TIC) sont aujourd'hui des incontournables pour les enseignants désirant renouveler leur enseignement. Afin d'en tirer le plus grand profit possible, nous avons pensé rendre disponible sur l'Internet un tutoriel d'aide à l'apprentissage de la programmation. La disponibilité du tutoriel, en tout temps, et son accès, de n'importe où, permettront une utilisation maximale. Le traitement multiple d'une même notion dans le tutoriel sera également un atout majeur permettant de rejoindre les différentes approches privilégiées par les étudiants.

Les activités de programmation utilisant un montage électronique robotisé et le tutoriel formeront le produit résultant de nos travaux. Ce produit correspondra à un environnement informatisé pour l'apprentissage humain (EIAH) offrant de nouveaux rapports aux savoirs et aux savoir-faire qui permettront d'améliorer l'apprentissage de la programmation par nos étudiants..

Pour réaliser nos objectifs, nous entreprendrons une recherche de développement en éducation conformément au modèle de Nonnon (1993). C'est ainsi qu'à partir de notre idée de recherche, confrontée préalablement à des considérations d'ordre théorique, nous développerons un prototype qui subira plusieurs types d'évaluation.

Chapitre 1

PROBLÉMATIQUE ET IDÉE DE RECHERCHE

1.1 Difficultés liées à l'enseignement et à l'apprentissage de la programmation

L'apprentissage de la programmation se retrouve dans tous les programmes universitaires en sciences et génies. Discipline incontournable, voire obligatoire, la programmation est généralement prévue comme un préalable à tout début de parcours académique. Notre recherche portera spécifiquement sur l'apprentissage de la conception de programmes informatiques, communément appelé l'apprentissage de la programmation, dans le cadre d'une formation de base en génie.

En 1988 naissait l'Association Francophone pour la Didactique de l'Informatique (AFDI) qui prévoyait entre autre, l'organisation d'une rencontre à tous les deux ans. Cinq rencontres furent organisées sous l'égide de cette association. Duchâteau (2002) rapporte que les deux premiers portèrent presque exclusivement sur la programmation. Pour lui, la didactique de l'informatique se consacre alors à des interrogations, des relations d'expériences, des analyses à propos de l'enseignement et de l'apprentissage de l'algorithme et de la programmation. Les trois dernières rencontres s'intéressent plus particulièrement à une informatique des utilisateurs. Duchâteau précise que lors de ces rencontres, la didactique s'oriente cette fois vers la maîtrise des instruments logiciels et des environnements informatisés. Il résume par la question suivante un des problèmes centraux de la didactique des usages des TIC: *« utiliser un système informatisé, cela peut s'apprendre, mais cela s'enseigne-t-il? ».*

Le retrait de la discipline informatique de l'ensemble des matières enseignées au secondaire en France sonna le glas de l'AFDI. Au Québec, la formation en didactique de l'informatique s'intéresse presque exclusivement à la clientèle du niveau secondaire. La récente décision du gouvernement québécois de retirer du niveau secondaire I le cours spécifiquement informatique pour répartir ses notions à travers différents cours ne contribuera certes pas à l'essor de la didactique de l'informatique. Conséquemment, l'Université de Montréal a déjà pris la décision d'éliminer les deux cours de didactique de l'informatique.

La didactique s'est associée initialement avec les mathématiques, puis les sciences et le français. Dans ces domaines, la contribution de la didactique dans les nouvelles approches pédagogiques est omniprésente. La situation est toute autre avec l'informatique où les spécialistes du domaine de l'enseignement semblent prononcer du bout des lèvres l'expression « didactique de l'informatique ». À notre avis, un volet de la problématique avec l'informatique réside dans l'utilisation d'outils informatiques pour l'enseignement de la théorie informatique. Les enseignants en informatique sont des maîtres de l'utilisation de l'ordinateur et de grands connaisseurs de logiciels. Il leur apparaît tout a fait naturel d'utiliser ces outils dans leur enseignement, qui plus est, dans certain cas peuvent correspondre directement à la matière à enseigner. Comble de bonheur, l'ordinateur et ses applications composent le matériel privilégié du didacticien. En effet, l'heure est aux technologies de formation et d'apprentissage, à l'utilisation didactique de l'ordinateur, aux technologies de l'information pour l'éducation, aux technologies éducationnelles, etc. Nos enseignants en informatique sont-ils pour autant des didacticiens de l'informatique?

Pour l'enseignant et l'apprenant, la problématique de l'apprentissage de la programmation revêt plusieurs dimensions. Nous aborderons dans ce chapitre les quatre difficultés suivantes:

- l'hétérogénéité de la clientèle étudiante;

- l'abstraction en vue de la transposition de la solution dans le langage informatique;
- la détection, l'interprétation et la correction d'erreurs;
- la nécessité d'intégrer les connaissances du domaine d'application en plus des connaissances du domaine informatique.

1.1.1 Hétérogénéité de la clientèle étudiante

Pour l'ensemble de la classe, le premier problème s'exprime par l'hétérogénéité des connaissances initiales en informatique des étudiants. L'enseignant devra alors tenir compte de cette hétérogénéité dans la préparation de ses interventions devant le groupe de sorte à préserver la motivation de l'ensemble des étudiants.

À leur première année d'études, tous les étudiants en génie et cela indépendamment de la discipline choisie[1] doivent compléter les cours du tronc commun, c'est-à-dire les cours d'enseignement commun à l'ensemble des programmes, incluant l'apprentissage de la programmation. Ces cours sont extrêmement populeux puisqu'ils s'adressent à la totalité des étudiants de première année. Typiquement, pour un cours donné, les étudiants sont répartis en une douzaine de groupes d'environ soixante étudiants. L'enseignement et l'évaluation s'effectuent de la même façon dans chacun des groupes afin de s'assurer de l'homogénéité de la formation.

Pour différentes raisons, il est reconnu que la clientèle étudiante du tronc commun est des plus hétérogènes. D'un côté, il y a les étudiants provenant de l'étranger qui ont des formations académiques différentes même si elles sont jugées équivalentes. De l'autre côté, il y a les étudiants d'ici avec des formations très variées, les collèges jouissant d'une certaine latitude dans l'élaboration de leurs cours ainsi que dans leurs méthodes d'évaluation. La formation dans certains collèges peut être poussée et pour d'autres plutôt faible. De plus, cette distinction de la formation étudiante est d'autant plus vraie

[1] L'École Polytechnique de Montréal offre présentement quatorze programmes en génie soit: biomédical, chimique, civil, électrique, géologique, industriel, informatique, logiciel, des matériaux, mécanique, métallurgique, des mines, nucléaire et physique.

dans le cas de l'informatique puisque les programmes collégiaux tels que définis par le ministère de l'éducation ne contiennent aucun cours obligatoire en informatique. Le cours d'informatique étant classé complémentaire dans les programmes menant aux études universitaires, certains étudiants décident de l'inclure à leur curriculum et d'autres non. Conséquemment nous pouvons distinguer, selon le niveau de connaissance initiale en informatique, trois types d'étudiants. Tout d'abord il y a ceux qui ont peu ou pas de connaissance des environnements informatiques, ensuite ceux qui connaissent et savent utiliser certains outils informatiques et finalement ceux qui ont certaines notions de la programmation en informatique.

Avec le premier type d'étudiant, celui possédant peu de connaissance des environnements informatiques, on doit s'assurer préalablement et le plus rapidement possible d'une maîtrise suffisante de ces environnements afin d'asseoir adéquatement les notions et concepts propres à la programmation. Cela se réalisera par une utilisation assidue de l'ordinateur avec des manipulations ciblées. Avec le deuxième type d'étudiant, celui capable d'utiliser adéquatement certains outils informatiques, on tentera de profiter au maximum de ses connaissances des interfaces et fonctionnalités normalisées pour écourter cette étape préalable et l'engager le plus rapidement possible dans la programmation. Le troisième type d'étudiant, celui pour qui la programmation n'est pas une nouveauté, semblerait avantagé au premier abord. Par contre, cet avantage peut dégénérer en une réticence envers l'enseignement prodigué. En effet, il se peut qu'ayant reçu un enseignement non conforme aux règles de l'art de la programmation structurée, l'étudiant doive réapprendre à programmer correctement même s'il prétend savoir programmer. Habituellement, la réticence est grande. On peut comparer cela à enseigner la natation à une personne qui sait déjà nager. La correction d'une mauvaise technique de nage passe par un effort supplémentaire de désapprentissage avant le réapprentissage qui peut sembler inutile à celui pouvant accumuler les longueurs de piscine. À quoi bon s'efforcer à corriger ses erreurs lorsque les bénéfices ne sont pas évidents, à première vue.

La quantité de savoirs à acquérir du premier type d'étudiant par rapport au troisième type est évidemment plus grande, mais d'un point de vue pratique pour la programmation nous les considérons tous sous la même épithète de débutant. Selon notre expertise nous qualifions chaque type selon leur niveau d'expérience qui s'échelonne d'aucune à significative. Le tableau 1.1 présente les caractéristiques de chaque type d'étudiants.

Tableau 1.1 Caractéristiques de chaque type d'étudiants

	Type 1	**Type 2**	**Type 3**
Connaissance initiale	Utilisation d'un ordinateur	Outils informatiques	Notions de programmation
Expérience de programmation	Aucune	Peu	Significative
Attitude face à l'apprentissage	Anxieux	Contrôlé	Confiant Parfois trop (réticence)
Intérêt/motivation	Moyen à Élevé		

Les professeurs doivent composer avec cette hétérogénéité lors de l'élaboration de leur cours. Ils doivent également proposer des défis intéressants qui permettront d'augmenter ou du moins conserver la motivation de chaque étudiant, facteur qui favorisera un bon apprentissage.

1.1.2 Abstraction et transposition dans le langage informatique

Détienne (1998) précise que du point de vue psychologique la structure d'un programme se distingue à travers différents types de relation et niveaux d'abstraction: la structure linéaire, le flux de contrôle, le flux de données, les plans et l'organisation hiérarchique des buts et sous-buts (Kant & Newell, 1995 ; Pennington, 1987 ; Pennington & Grabowski, 1990). La structure linéaire d'un programme s'observe dans sa version textuelle. Le flux de contrôle représentant l'ordonnancement des opérations à l'exécution s'exprime à l'aide de trois constructions: l'itération, la séquence et le test. Le flux de données représente les données et leur transformation au cours de l'exécution du

traitement. Un plan est un ensemble d'actions qui ordonnées de façon adéquate, permettent de réaliser un sous-but. Rist (1991) affirme qu'un programme peut être représenté par une combinaison de plans, chaque plan étant constitué d'un enchaînement d'actions permettant d'atteindre un but.

Le caractère abstrait du langage informatique est pour l'apprenant le premier rempart à franchir. Avant d'être en mesure de réaliser un programme informatique tout étudiant doit assimiler différents concepts abstraits tels que décrits précédemment par Détienne et Rist. Conséquemment, l'étudiant doit connaître certaines notions élémentaires de programmation telles les structures répétitives, les structures décisionnelles, les structures de décomposition (sous-programme) et les structures de données. Ensuite, l'activité de programmation en informatique nécessite une abstraction des entités du problème afin de permettre une transposition dans le langage de programmation. Sur ce point, certaines études ont montré que les experts forment des représentations abstraites, conceptuelles des problèmes alors que les novices représentent les problèmes selon des traits de surface, par exemple la structure syntaxique (Shneiderman, 1976 ; Adelson 1981 ;1985) .

La transposition de la solution du problème dans le langage de programmation nécessite l'assimilation de notions et concepts qui sont trop rapidement survolés en classe voire même tout simplement escamotés. Le plan de cours est abondant et le temps presse. Tous les efforts sont orientés vers la production d'un programme fonctionnel. Cependant pour arriver à cette fin, l'étudiant doit préalablement acquérir une méthode de penser qui l'achemine vers une description de la tâche sous forme de séquences d'opérations. La décomposition de la tâche en opérations n'est pas une démarche simple. Dans cette situation, l'étudiant est rapidement confronté à la difficulté du choix des opérations ainsi qu'au niveau de détail nécessaire pour décrire la tâche. A titre d'exemple, le tableau 1.2 présente trois descriptions différentes pour décrire le remplacement d'un pneu crevé d'une automobile.

Tableau 1.2 Remplacement d'un pneu crevé

Description 1	Description 2	Description 3
• Surélever l'automobile • Enlever le pneu crevé • Installer le pneu d'urgence • Abaisser l'automobile	• Placer le cric sous l'automobile • Enlever l'enjoliveur • Débloquer chaque vis • Tourner la manivelle du cric jusqu'à la hauteur désirée • Dévisser chaque vis • Retirer le pneu crevé • Déposer le pneu crevé • Prendre le pneu d'urgence • Insérer le pneu d'urgence • Visser chaque vis • Replacer l'enjoliveur • Tourner la manivelle pour descendre l'automobile • Enlever le cric	• Ouvrir le coffre arrière • Prendre le cric • Prendre le pneu d'urgence • Surélever l'automobile avec le cric • Enlever le pneu crevé • Placer le pneu d'urgence • Abaisser l'automobile • Mettre le pneu crevé et le cric dans le coffre arrière • Fermer le coffre

Ces trois descriptions informent le conducteur des opérations ou étapes à suivre pour installer son pneu d'urgence. Par contre, la clarté ou la complétude de chaque description peuvent être perçues différemment d'une personne à l'autre. La description 1 suppose que les termes surélever, enlever, installer et abaisser sont suffisants. Or, chacune de ces actions incorporent d'autres actions. Par exemple, pour surélever la voiture, il faut utiliser un cric et lever la voiture à une hauteur particulière permettant de dégager le pneu. La description 2, la plus longue, précise différentes actions à compléter pour installer le pneu d'urgence en omettant toutefois certaines précisions se retrouvant dans la description 3. Comme le cric ne tombera pas du ciel, il apparaît opportun d'indiquer comment le quérir! Cet exemple illustre bien la difficulté du choix des termes et du niveau de détail. Cette recherche de terminologie et du niveau de détail est la problématique initiale de l'apprenant. La description idéale sera celle conforme à la solution et que l'étudiant pourra aisément transposer dans le langage de programmation. Pour l'enseignant cette problématique peut se traduire par le dilemme de la poule et de l'œuf. En effet, en informatique pour décrire convenablement une tâche, l'apprenant doit

avoir une bonne idée de la programmation mais cette description est préalable à la programmation et il doit préalablement en faire abstraction. Bref, savoir décrire sans savoir programmer mais dans le but de programmer. Savoir bien décrire implique avoir une idée de la finalité du programme donc c'est un processus qui va s'acquérir progressivement de manière à faire passer l'apprenant d'une description exhaustive à une description efficace, juste nécessaire.

1.1.3 La détection, l'interprétation et la correction de l'erreur

Pendant la rédaction de son programme informatique, l'étudiant commet inévitablement des erreurs qu'il doit interpréter et corriger parce que l'environnement de développement ne détecte que certains types d'erreurs. En cas de détection d'erreur, l'étudiant doit tenter de comprendre et interpréter l'information obtenue afin d'apporter les bons correctifs. L'alternative où l'erreur n'est pas détectée automatiquement laisse l'étudiant à lui-même. Cette activité de correction d'erreurs installe l'étudiant en situation active de résolution de problèmes.

Lors de la réalisation d'exercices, l'étudiant recherche la solution au problème les yeux rivés à son écran d'ordinateur à l'affût des informations affichées par l'environnement de développement. Il tente d'interpréter correctement cette information mais son inexpérience peut lui jouer des tours. En effet, il peut être incapable de décoder l'information ou en déformer le sens. Dans un cas comme dans l'autre, l'étudiant risque de converger difficilement vers la solution ou au pire de s'en éloigner. De plus, les environnements de développement produisent une information, habituellement précise et pertinente dans le cas d'une erreur de compilation, mais de piètre qualité voire absente dans le cas d'une erreur d'exécution. L'étudiant embourbé réduit alors de plus en plus son champ d'action en se concentrant sur son écran où est affichée la parcelle d'information. Il faudrait le sortir de cette emprise et lui offrir une alternative lui permettant de réorienter son raisonnement.

Il nous apparaît important à ce moment ci d'apporter certaines précisions concernant l'erreur de programmation que nous divisons en deux classes: l'erreur de compilation et l'erreur d'exécution. L'erreur de compilation est automatiquement détectée par les environnements de programmation lors de la procédure de compilation. Elle comprend l'erreur syntaxique et l'erreur sémantique statique. L'erreur d'exécution comprend tous les autres types d'erreur qui ne sont pas détectés automatiquement par le compilateur et qui entraînent un fonctionnement erroné du programme. L'erreur d'exécution comprend une panoplie de types d'erreur tels: l'erreur sémantique dynamique, l'erreur de logique, l'erreur algorithmique, etc. Afin d'éviter de tomber dans un discours trop technique nous nous limiterons à parler d'erreur de compilation et d'erreur d'exécution. Le tableau 1.3 présente quelques exemples pour chaque type d'erreur : compilation et exécution.

Tableau 1.3 Quelques exemples de chaque type d'erreur : compilation et exécution

	Exemples
Erreur de compilation	▪ Omettre un `;` à la fin d'une instruction ▪ Identificateur mal orthographié ▪ Utiliser un énoncé `else` sans l'utilisation préalable d'un `if` ▪ Effectuer une affectation dans une expression booléenne ▪ Utiliser une variable sans valeur ▪ Appel erroné à une fonction
Erreur d'exécution	▪ Expression booléenne inexacte : o Utiliser l'opérateur relationnel > plutôt que < o Utiliser l'opérateur booléen OU plutôt que ET ▪ Formulation mathématique inexacte ▪ Mauvaise séquence de structures de contrôle o Trois énoncés `if` successifs plutôt qu'une imbrication de if-else ▪ Valeur erronée de l'indice d'un tableau

1.1.4 Intégration des connaissances de plusieurs domaines

Trouver la solution d'un problème consiste à comprendre et résoudre ce problème. Dans le cas précis de la réalisation d'un programme informatique, l'étudiant est centré sur l'élaboration d'une solution correspondant à une procédure. Pour ce faire, il doit se construire une représentation de la structure de cette procédure et l'exprimer dans un langage de programmation.

La programmation exige l'appropriation de connaissances concernant la syntaxe du langage, la sémantique des structures du langage (composition de structures de contrôle, de fonctions, etc.), la rédaction d'algorithmes permettant de réaliser la tâche et cela sans oublier les connaissances propres au domaine de l'application (Brooks, 1977 ; Rist, 1986 ; Soloway, Erhlich, Bonar & Greenspan, 1982). Cet amalgame de savoirs peut causer plusieurs ennuis à l'étudiant. Par exemple, l'étudiant n'étant pas familier avec le sujet de l'application à développer, il peut réaliser une mauvaise interprétation de l'énoncé du problème qui exigera d'apporter plusieurs modifications à sa réalisation.

Nous avons énoncé dans les paragraphes précédents plusieurs facteurs pouvant influencer l'apprentissage de la programmation. Parmi ceux-ci soulignons:

1.1.4.1 la formation initiale hétérogène, à savoir s'il a acquis ou non des connaissances en informatique personnellement ou par l'entremise d'un cours au collégial;

1.1.4.2 la nature abstraite de la conception d'un programme informatique;

1.1.4.3 les difficultés liées à la détection, l'interprétation et la correction de l'erreur qui ne peut être détectée automatiquement par l'environnement de développement de programme;

1.1.4.4 l'appropriation de connaissances concernant le domaine de l'application et le domaine de l'informatique;

1.1.4.5 la transposition de la solution du domaine de l'application au domaine de l'informatique et inversement pour valider la bonne exécution du programme.

Ces facteurs sont d'ordre motivationnel, conceptuel, administratif et pédagogique. Dans le cadre de notre recherche nous nous attaquerons principalement aux facteurs d'ordre didactique conceptuel, c'est-à-dire ceux où notre enseignement pourra avoir une influence.

1.2 Idée de recherche et développement

1.2.1 Aides à l'apprentissage de la programmation : approches courantes

Plusieurs approches ont été développées pour faciliter l'intégration de différents domaines cognitifs inhérents à la programmation. McGill et Hobbs (1996) prétendent que l'apprenant peut acquérir la syntaxe d'un langage informatique en consultant le texte d'un volume, tandis que les habiletés de conception et de résolution de problèmes nécessitent d'autres ressources. Greening (2000) croit que les systèmes ou outils de visualisation sont les ressources les plus prometteuses pour acquérir ces habiletés. Une panoplie de systèmes d'aide à l'apprentissage utilisant la visualisation à l'écran sous forme d'illustration, d'animation ou de simulation sont proposés pour différents sujets : l'architecture de l'ordinateur (Connelly et al., 1996), les structures de données (Shaffer et al., 1996) , les algorithmes (Price et al., 1994 ; Dershem et al., 1996) et les systèmes d'exploitation (Greening, 1996). Les étudiants en génie sont friands de nouvelles technologies, principalement de l'Internet. Greening (2000) affirme que les étudiants s'attendent à utiliser ces technologies et que l'inclusion de support sur Internet pour un cours d'introduction à l'informatique accroîtra la motivation des étudiants. De leur côté, Lawrence et al. (1994) s'intéressent à l'enseignement des algorithmes, ils affirment que la participation active de l'étudiant est la clé qui le mènera à l'apprentissage. Nous aurons recours à ces résultats de recherche pour développer un environnement informatique d'apprentissage.

Comme nous venons de le préciser, plusieurs chercheurs se sont penchés sur la problématique d'outils ou d'environnements qui permettraient non seulement d'enseigner mais surtout d'apprendre efficacement la programmation en informatique.

Pour ce faire, ils ont tenté de comprendre les habiletés cognitives et les connaissances nécessaires à la réalisation des tâches inhérentes au développement de logiciel (Weinberg, 1971 ; Papert, 1980 ; Shneiderman, 1980 ; Soloway, Ehrlich, Bonar & Greenspan, 1982 ; Mayer, 1988 ; Hoc, Green, Samurcay & Gilmore, 1990 ; Lemut, du Boulay & Dettori, 1993 ; Shackelford & Badre, 1993 ; Ebrahimi, 1994 ; Ennis, 1994 ; Deek, Kimmel & McHugh, 1998). Ces différentes tâches sont : l'apprentissage du langage, la rédaction de nouveau programme, la compréhension de programme existant, la vérification et le débogage, la rédaction de la documentation et la maintenance. Certaines tâches requièrent à la base plus de connaissances de la syntaxe et de la sémantique du langage de programmation utilisé (Shneiderman, 1980 ; Rogalski & Samurcay, 1990, 1993), alors que d'autres tâches requièrent principalement des connaissances reliées à la résolution de problème telles la compréhension du problème, l'analyse et le design de la solution, les exigences du domaine, les connaissances stratégiques et tactiques (Wirth 1971 ; Pennington & Grabowski, 1990).

Des systèmes et des méthodologies ont été développés afin d'améliorer l'apprentissage et la pratique de la programmation. Deek & McHugh (1999) les distinguent en quatre classes: environnements de programmation, aides au débogage, tuteur intelligent et environnement de programmation intelligent. Outre les fonctionnalités habituelles d'un environnement de programmation intégré, celui développé spécifiquement pour l'enseignement peut inclure des fonctionnalités supplémentaires permettant à l'étudiant d'expérimenter certaines particularités du langage ainsi que d'autres fonctionnalités permettant une forme limitée de tutorat (Carbonell, 1970 ; Brown, Burgon & Bell, 1974 ; Barr, Beard & Atkinson, 1975). L'aide au débogage est utilisé par le programmeur pour tester le programme, pour observer le comportement du programme lors de son exécution, pour détecter et corriger les erreurs. Les outils communément compris dans les débogueurs exploitent la visualisation et la simulation. Le système de tuteur intelligent offre l'accès au tutorat et la possibilité d'expérimenter la syntaxe du langage. Il peut proposer des leçons adaptées aux besoins de l'apprenant. Il peut aussi analyser la réponse de l'étudiant et la corriger, le guider et même interagir avec lui en fournissant des informations et/ou des rétroactions. Typiquement, le tuteur propose à

l'étudiant un problème à résoudre et vérifie le cheminement de l'étudiant selon une solution connue. L'environnement de programmation intelligent combine les propriétés du tuteur intelligent avec les outils de l'environnement de développement traditionnel.

Deek (1999) énumère quelques difficultés rencontrées par ces systèmes :

- *l'absence d'un cadre considérant à la fois la résolution de problèmes et le génie logiciel ; aucun outil ne favorise la formulation du problème, ni la planification et le design de la solution ;*
- *la séparation de l'activité de codage du reste des tâches de la résolution de problèmes crée une emphase exagérée sur la syntaxe du langage ;*
- *les étudiants peuvent être insatisfaits de l'interface usager;*
- *l'impossibilité d'intégrer l'outil dans le curriculum; l'outil est mal perçu et son impact sur les cours d'introduction à la programmation et le curriculum n'est pas clair ;*
- *la créativité de l'étudiant et le développement d'habilités de raisonnement seraient amoindris par la résolution rigide d'un problème.*

1.2.2 Notre idée de développement

Notre problématique a considéré certaines difficultés de l'apprentissage des concepts informatique. L'idée de recherche qui en découle vise, en partant de notre propre modèle d'enseignement et de cette problématique, à développer un environnement informatisé pour l'apprentissage humain (EIAH). Nous désirons permettre à l'étudiant de s'investir plus intensément afin de compenser ses lacunes. Dans cet esprit, notre EIAH proposera à l'étudiant des travaux pratiques encadrés (TPE) qui intégreront des exercices de programmation utilisant un matériel électronique robotisé jouant le rôle d'un objet réel pour penser de manière à rendre signifiante l'activité de programmation et faciliter la

manipulation des structures informatiques abstraites. Cette activité devrait favoriser l'assimilation d'une méthode d'analyse menant à la rédaction d'un algorithme exprimant la solution informatisée. Le comportement d'un montage offrant une représentation concrète de l'exécution d'un programme aidera à l'acquisition des structures de programmation par l'étudiant, ce montage robotisé fonctionnant conjointement avec le déroulement du programme. Notre EIAH offrira aussi un deuxième outil d'aide à l'apprentissage, un tutoriel informatisé que tout étudiant pourra consulter pour trouver des réponses à ses questions. La figure 1.2 précise la complémentarité des dispositifs inscrits dans notre EIAH.

Face à un nouveau concept à apprendre, l'étudiant peut être invité à réaliser un exercice de programmation par l'entremise du montage ou à utiliser directement le tutoriel.

Dans l'éventualité où l'étudiant doit réaliser un exercice de programmation à l'aide du montage, il pourra rencontrer certaines difficultés. Dans cette situation, il pourra au besoin recourir au tutoriel pour obtenir certaines informations. Pour ce faire, il consulte la ou les rubriques traitant du concept. Si l'information recherchée obtenue est suffisante et satisfaisante, l'étudiant pourra poursuivre son programme ou accéder à des analogies ou des exercices ou des simulations. Lors de cette activité l'étudiant pourra accéder au tutoriel autant de fois qu'il en ressent le besoin. À la rigueur, il pourra même explorer un concept à l'aide de ce tutoriel avant même de débuter son exercice de programmation. Il est important de préciser que l'activité de programmation s'effectuera dans le cadre d'une pédagogie de projet où l'apprentissage collaboratif sera mis de l'avant (Boutinet, 1993; Dubois 1997; Perrenoud 1999 ; Terenzi et al., 2001).

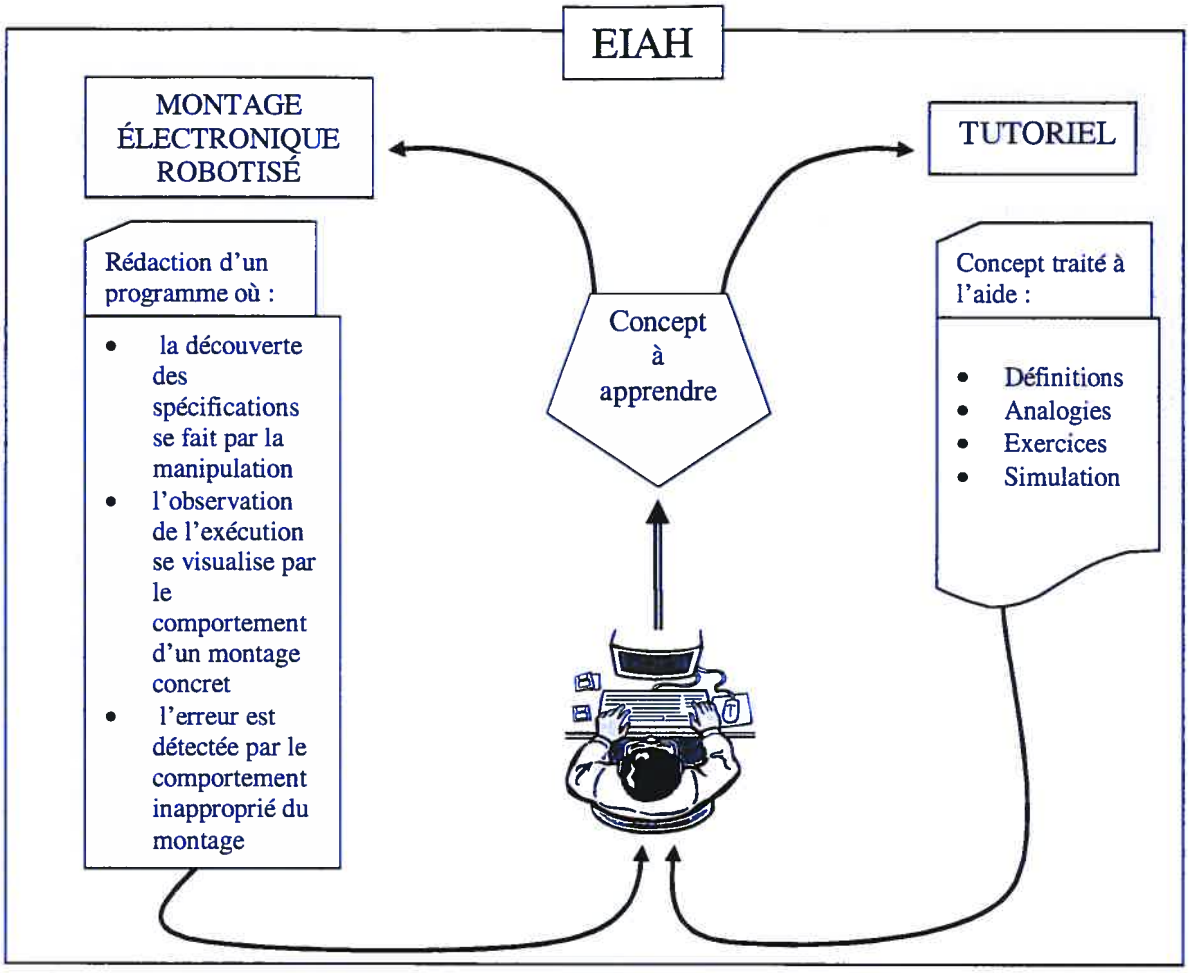


Figure 1.1 Un environnement informatique pour l'apprentissage humain se composant d'exercices utilisant un montage électronique robotisé et un tutoriel.

Dans le second mode d'utilisation de l'EIAH, l'étudiant pourra utiliser le tutoriel seul pour explorer et assimiler un nouveau concept sans réaliser un exercice de programmation par l'entremise du montage. L'étudiant pourra naviguer dans le tutoriel à son propre rythme, il l'utilisera autant de fois que désirée et il accèdera aux différents concepts selon ses besoins dans l'ordre de son choix. En ce sens, le tutoriel devra servir de support à un apprentissage autodidacte. Ici, le professeur présentera initialement un concept et l'étudiant pourra le revoir et l'expérimenter ensuite à l'aide du tutoriel. Nous tenterons de construire notre tutoriel de manière à répondre aux principales difficultés énoncées dans notre problématique. La caractéristique la plus originale de cet EIAH repose sur la cohésion entre l'exercice de programmation des comportements du montage électronique robotisé et d'un tutoriel d'aide en ligne que l'étudiant pourra

consulter à tout moment pour s'aider dans cet exercice de programmation. Notons ici que les utilisations du tutoriel ne seront pas limitées à cet exercice de programmation puisqu'il devra couvrir plusieurs concepts tels, les tableaux, les enregistrements, les fichiers binaires, etc.

Afin d'éviter toute confusion entre les exercices inscrites dans le tutoriel et les exercices de programmation avec le montage robotisé, nous utiliserons le terme « atelier de programmation » pour désigner les exercices de programmation avec le matériel robotisé.

Toujours à la recherche de moyens permettant d'attiser l'intérêt et la motivation des étudiants nous nous inscrirons donc dans le mouvement de la pédagogie active (Dirand et al., 2002 ; de Theux, Jacqmot, Wouters, 2002). Notre ligne maîtresse de pensée nous a conduit à transformer dans ce projet de R&D un enseignement centré sur le professeur vers un enseignement centré sur l'étudiant. L'étudiant à besoin de voir mais il réclame aussi de faire. Soucieux de son niveau de rétention, il ne veut plus crouler sous une masse imposante d'informations sans valider sa compréhension. Nous voulons également orienter nos interventions de sorte à favoriser un apprentissage par compétences (Perrenoud, 2000) en intégrant des savoirs faire et des savoirs en informatique. L'activité de programmation nécessitera alors la réalisation de plusieurs tâches telles que la rédaction du code source, la détection et la correction d'erreurs, la vérification et la modification du programme. Ces diverses tâches devront s'intégrer dans la compétence en programmation que l'apprenant aura à acquérir. Ces apprentissages par compétence ne saurait être complet sans intégrer un savoir être que nous introduirons par la mise en situations d'un apprentissage collaboratif. En effet, nous emprunterons de la pédagogie par projet (Dubois,1997 ; Perrenoud, 1999) les bénéfices de la collaboration de manière à stimuler la contribution des pairs dans l'apprentissage. Finalement, nous tenterons d'exploiter une approche d'apprentissage basée sur le raisonnement analogique de sorte à arrimer des nouvelles connaissances plus abstraites de la programmation à des connaissances plus concrètes, plus sensibles ou déjà assimilées.

Pour optimiser au maximum les séances de cours de manière à augmenter la compréhension des étudiants, nous préconiserons un enseignement favorisant un apprentissage actif. Pour ce faire, nous permettrons aux étudiants de manipuler le montage électronique en le programmant, agissant ainsi simultanément sur le concret et sur les concepts abstraits de la programmation.

En résumé notre recherche de développement consistera à proposer un environnement informatisé pour l'apprentissage humain des concepts de la programmation composé:

- d'exercices de programmation exploitant un objet concret pour penser, soit un montage réel, sur lequel l'étudiant interagira par l'entremise de son programme et obtiendra des indicateurs d'erreur plus précis;
- un tutoriel dédié qui agira comme support pédagogique à l'apprentissage de la programmation offrant une information en temps opportun ou juste à temps.

Cette nouvelle manière de faire apprendre la programmation est basée sur:

- un apprentissage actif de l'étudiant selon les principes de la pédagogie active (Lebrun 2002, Actes du colloque de l'AIPU, 2002);
- la pédagogie par problèmes et par projets (Norman et Schmidt, 1992 ; Woods, 2000, Barg et al., 2000)
- le recours au raisonnement analogique utilisant des notions connues pour l'assimilation de nouveaux concepts informatique (Gentner, Holyoak, Kokinov, 2000);
- l'exploitation des technologies et de l'Internet afin d'intensifier, de diversifier et de rapprocher les rapports aux savoirs et au savoir-faire;
- de nouveaux indicateurs d'erreurs obtenus par le montage électronique robotisé.

Le tableau 1.4 reprend chaque difficulté liée à l'apprentissage de la programmation que nous avons précédemment décrite et associe à chacune les interventions proposées dans notre idée de recherche.

Tableau 1.4 Interventions réalisées en fonction de la difficulté vécue

Difficulté	Interventions
Hétérogénéité de la clientèle	▪ Tutoriel permettant un apprentissage individualisé ▪ Pédagogie par problèmes et par projets incitant à l'apprentissage collaboratif et favorisant le partage et l'homogénéisation
Abstraction et transposition dans le langage informatique	▪ Manipulation du montage réel et observation de son comportement en notant les actions posées ▪ Recours à l'analogie pour représenter les concepts
Détection, l'interprétation et la correction de l'erreur	▪ Repérage de l'erreur basé sur la quasi-bijectivité entre l'exécution du programme et le comportement du montage robotisé ▪ L'observation du montage robotisé offre des indicateurs sur le type d'erreur commise permettant une correction appropriée
Intégration de connaissances de plusieurs domaines	▪ L'exercice de programmation avec le montage robotisé où la description du travail à réaliser se précise lors de la manipulation initiale ▪ La décomposition des concepts ainsi que les différents traitements de l'information dans le tutoriel

Dans ce chapitre, nous avons identifié le caractère abstrait du langage informatique, la transposition de la solution dans le langage informatique, l'interprétation-correction d'erreurs, la nécessité d'intégrer les connaissances du domaine d'application et domaine informatique en plus de remédier à l'hétérogénéité des connaissances initiales en informatique en tant que problème en organisant un environnement coopératif. Nous avons proposé une solution constituée d'un raisonnement analogique axé sur la transposition du comportement d'un objet concret vers l'exécution d'un programme informatique, l'approche de décomposition par raffinement graduel, la pédagogie par problèmes et par projets. Nous avons justifié cette idée par des considérations relatant les faiblesses du modèle actuel de l'enseignement de la programmation. Nous allons dans le prochain chapitre présenter la méthodologie de recherche utilisée pour réaliser nos travaux.

Chapitre 2

MÉTHODOLOGIE DE LA RECHERCHE

Nous allons dans ce chapitre situer, caractériser la démarche de notre recherche parmi l'éventail des méthodologies existantes. Pour ce faire, nous aborderons et analyserons quelques propositions de classification.

En général, une recherche consiste à l'ensemble des travaux consacrés à l'étude d'une question afin d'obtenir de nouvelles connaissances. Elle s'inscrit dans certains domaines d'intérêts suivant une méthodologie spécifique et se déroule dans différents champs disciplinaires. Selon Van der Maren (1996 p.112) une méthode de recherche est

un ensemble d'opérations systématiquement et rationnellement enchaînées afin
- *de relier avec consistance*
  - *l'intention, le but, l'objectif de la recherche*
  - *la manière de poser le problème*
  - *les techniques de constitution du matériel et leur validation*
  - *les techniques de traitement transformant les données en résultats*
  - *les procédures d'interprétation des résultats et de leur vérification*
  - *la justification des différents choix*
- *de répondre aux critères formel et opérationnels auxquels elles doivent s'astreindre pour se voir accorder la crédibilité recherchée.*

2.1 Une classification des recherches

Il n'est pas simple de recenser ni de classifier les différents types de recherche. Par exemple, les recherches visées par le Fonds Nature et Technologies du gouvernement du Québec couvrent un large spectre allant des mathématiques pures jusqu'au développement de nouvelles technologies. Les thèmes se regroupent en dix grands domaines[1]: **structures abstraites**, nature et transformation de la matière, **études du**

[1] L'alternance des caractères réguliers et des caractères gras est utilisée pour faciliter la distinction de chaque domaine.

vivant, environnement, **ressources naturelles**, matériaux, **technologies de l'information et des communications**, systèmes et procédés, **conception, mécanique et structures**, méthodologies, techniques et instrumentation. La classification québécoise totalise cent onze domaines de recherche parmi lesquels se trouve le domaine intitulé « éducation » qui se subdivise en 33 sous domaines.

Une classification correspond à une distribution logique de certains éléments selon un certain ordre. Les fondements de cette classification s'appuient sur les critères de qualification des classes retenues et par le degré ou le niveau d'intensité des liens entre les classes. La structure résultante de ce tri peut se présenter sous la forme d'une liste linéaire ou séquentielle, d'une arborescence où les classes sont strictement disjointes ou d'un graphe où les classes ne sont pas nécessairement disjointes.

Les critères les plus usités sont : les buts visés, les sources de données, la démarche empruntée et les résultats obtenus. Van der Maren (1996) se distingue en utilisant, comme critère de classification, les enjeux de la recherche. À partir de ce critère il identifie :

> l'enjeu nomothétique, ce type de recherche vise le développement et le raffinement de connaissances théoriques en proclamant des lois, des principes généraux, des théories;
>
> l'enjeu politique : « *le but principal des recherches aux enjeux politiques est de changer les valeurs ou les besoins afin de modifier les conduites* »;
>
> l'enjeu pragmatique : « *il s'agit avant tout de trouver des solutions fonctionnelles aux problèmes de la pratique pédagogique quels que soient les fondements théoriques de ces solutions* »;
>
> l'enjeu ontologique vise le perfectionnement du praticien en favorisant le développement de ses connaissances et de ses habiletés à travers une recherche liée à la pratique.

De ces descriptions, nous estimons que l'enjeu de notre recherche est plutôt ontologique avec une préoccupation pragmatique puisqu'elle se déroule précisément dans le cadre de notre pratique et vise à améliorer une pratique pédagogique.

La structure résultante d'une classification peut, à notre avis, être un excellent indicateur de la vue de son ou ses auteurs sur la recherche. Cette structure nous révèle clairement l'absence, la présence, l'ordre ou l'intensité des liens qui unissent les domaines. En effet, la liste séquentielle des domaines de recherche propose qu'un certain type de recherche se nourrisse d'un autre type de recherche, imposant alors qu'un type précède ou suive un autre type étant donné les résultats préalables. L'arborescence précise qu'un type de recherche n'appartient qu'à un domaine unique, imposant ainsi une distinction précise et rigoureuse de chacun. À l'inverse, le graphe permet une grande souplesse en permettant qu'un type puisse s'inscrire dans plus d'un domaine. Que les liens entre les différents domaines soient forts ou inexistants, qu'ils soient orientés ou non, la classification doit permettre de situer son travail de recherche parmi l'ensemble des recherches réalisées. C'est dans cette optique que nous abordons notre étude de la classification des domaines de recherche.

Désireux de bien situer notre recherche, nous optons pour une structure hiérarchique qui, à partir du critère le plus englobant à la racine, permet la descente dans les branches de l'arbre définies selon des critères de plus en plus spécifiques. Tel que le proposait Martin Riopel (2001), les critères retenus et associés au niveau de l'arbre sont dans l'ordre, le domaine scientifique, le champ disciplinaire, le secteur d'intérêt, la démarche et le type de production. Nous tenterons de qualifier notre recherche selon ces critères.

2.2 Le domaine de recherche

Comme point de départ classifions les recherches selon le domaine scientifique concerné. À partir de ce critère, on peut fragmenter l'ensemble des recherches en différentes parties telles : les sciences de la nature, les sciences de l'homme, les sciences pures, les sciences de la santé, etc. Dans notre cas, notre travail s'inscrit dans le domaine des sciences de l'homme qui, selon Granger (1995), s'intéresse à la production de connaissances des faits humains.

2.3 Champ disciplinaire

Notre interprétation d'un champ disciplinaire d'un domaine de recherche est une portion de l'ensemble des connaissances le concernant. Conséquemment, le domaine des sciences de l'homme peut se subdiviser en plusieurs champs disciplinaires dont le nombre et les définitions varient d'un auteur à l'autre. Quant à notre travail, nous le situons dans le champ disciplinaire de la didactique de l'informatique.

Les chercheurs en didactique de l'informatique s'affairent à déterminer le contenu et la méthode de l'enseignement de l'informatique comme discipline. Patrick Mendelsohn (1998), professeur de la Faculté de Psychologie et des Sciences de l'Education des Facultés universitaires Notre-Dame de la Paix Namur écrit

> *La didactique de l'informatique a un statut à part dans la mesure où elle est concernée à deux titres par les NTI. D'abord, en tant que discipline qui réfléchit aux modes de transmission des savoirs et savoir-faire de la science informatique (enseignement de la programmation), ensuite en tant que concepteur de formalismes informatiques et de logiciels compatibles avec cet enseignement (développement et génie logiciel). [...] Si l'informatique y poursuit une carrière de discipline d'enseignement supérieur, au niveau scolaire l'intérêt institutionnel s'est focalisé sur l'intégration des logiciels dans les disciplines existantes. L'essentiel des recherches de ce domaine porte sur les méthodes et la pertinence de l'enseignement de la programmation, ainsi que sur l'évolution des langages de programmation et des logiciels professionnels.*

C'est autour de l'étude des conceptions que se font les élèves des contenus scientifiques enseignés que s'est développée, dans un premier temps, la recherche en didactique des sciences. Voyons comment on peut transposer ou adapter cela à l'informatique en tentant de préciser la place et le rôle de la didactique de l'informatique par rapport à la didactique.

2.4 Secteur d'intérêt de la didactique

Il apparaît pertinent de situer la didactique de l'informatique dans ce qui est convenu d'appeler, selon Astolfi et al. (1997), le triangle didactique : Savoir – Professeur –

Étudiant. La didactique s'intéresse aux liens les unissant. La figure 2.1 illustre le triangle didactique et situe sur sa surface les quatre secteurs d'intérêt : construction des situations, élaboration des contenus, interactions didactiques et stratégies d'appropriation.

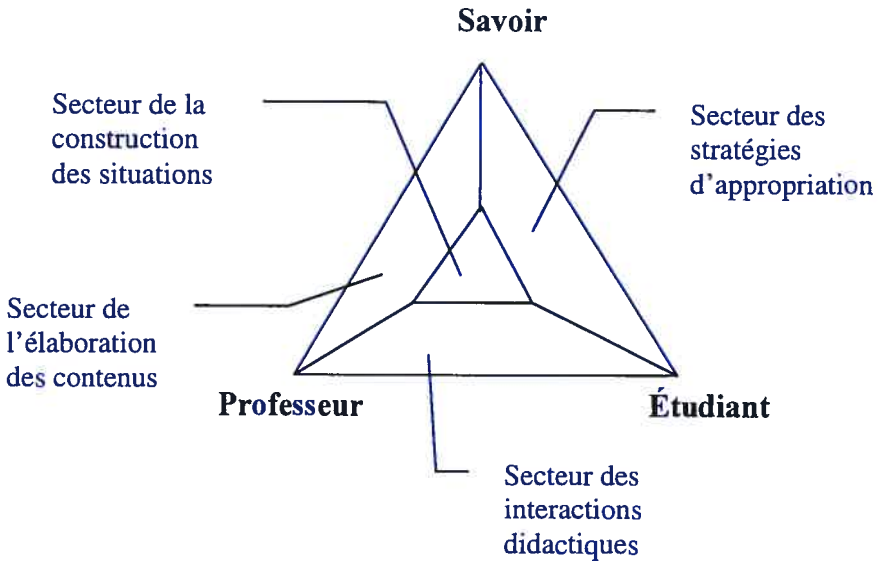


Figure 2.1 Les secteurs d'intérêt de la didactique (Astolfi et al. 1997)

Astolfi et al. (1997 p.72) mettent chaque secteur en relation préférentielle avec certains concepts de la didactique, soit :

> *« -un secteur de l'élaboration des contenus (transposition didactique, trame conceptuelle, niveau de formulation d'un concept, pratique sociale de référence...);*
>
> *-un secteur des stratégies d'appropriation (représentations, obstacles, erreur, résolution de problèmes...);*
>
> *-un secteur des interactions didactiques (coutume didactique, aide didactique...);*
>
> *- un secteur (central) de la construction des situations didactiques (objectif-obstacle, contrat didactique, dévolution, situation-problème, structuration...) »*

Notre recherche vise le développement d'un environnement d'apprentissage ainsi que la création d'ateliers ou de projets. Nous considérons que l'environnement d'apprentissage

appartient au secteur de l'élaboration des contenus puisqu'il traite les notions à différents niveaux et que sa conception utilise abondamment la transposition didactique. Quant aux ateliers, situations concrètes d'apprentissage sous forme d'une situation-problème, nous les situons dans le secteur central de la construction des situations didactiques.

L'un des aspects importants de la recherche est la démarche employée pour obtenir des résultats. Dans la section suivante nous exposerons quatre types généraux de recherche en soulignant le rôle de la démarche dans la méthodologie de la recherche.

2.5 Quatre types généraux de recherche

Dans le cadre de recherches en sciences humaines et dans le but de recenser et de classer les travaux de recherche en didactique du français langue maternelle, Gagné et al. (1989) dégagent quatre types généraux de recherche fondés sur le double critère de l'objectif principal de la recherche et de la démarche centrale d'investigation de la recherche. L'objectif principal d'une recherche est l'action la plus importante qu'elle tente d'accomplir. Ces deux critères permettent selon Gagné et al. (1989) d'englober la diversité des types de recherche tout en ciblant leur objectif respectif.

Gagné et al. (1989) identifient quatre objectifs principaux possibles pour la recherche scientifique que nous résumons ainsi :

décrire : *obtenir empiriquement des faits, des objets, des événements, des comportements en utilisant différentes stratégies d'observation;*

théoriser : *analyser rationnellement des concepts dans le but de construire un modèle théorique et opérationnel;*

expliquer : *utiliser un modèle pour déduire une relation entre au moins une cause et au moins un effet et mettre expérimentalement à l'épreuve cette relation;*

transformer : *mettre en œuvre une idée, une situation, une démarche ou un outil susceptible de produire un changement de la réalité.*

Selon ces descriptions, nous reconnaissons que l'objectif principal de notre recherche concerne la transformation d'une réalité, soit des concepts ou notions informatiques, inscrite à l'intérieur d'une démarche comprenant le développement d'un outil spécifique.

Décrire, théoriser, expliquer et transformer sont les quatre objectifs principaux identifiés par Gagné et al (1989) auxquels ils associent à chacun une démarche centrale d'investigation permettant de caractériser le type de recherche considéré. Ces démarches sont : l'observation pour décrire, l'analyse conceptuelle pour théoriser, l'expérimentation pour expliquer et l'intervention pour transformer. Selon ces deux critères, objectif et démarche, nous obtenons quatre types généraux de recherche : la recherche descriptive, la recherche théorique, la recherche expérimentale et la recherche-action. Les sections suivantes décrivent ces quatre types de recherche.

2.5.1 La recherche descriptive

La recherche descriptive vise principalement, dans le contexte de la didactique de l'informatique, à décrire empiriquement un ou plusieurs éléments d'une situation d'appropriation de savoirs informatiques. Pour atteindre cet objectif, elle peut utiliser une ou plusieurs stratégies d'observation en fonction du contexte de la recherche. Gagné et al (1989) identifient neuf stratégies d'observation possibles, soit

> ***l'enquête**, qui recueille des données concernant un échantillon à l'aide de questionnaires ou d'entrevues;*
>
> ***l'étude de cas**, qui tente d'analyser en profondeur un cas circonscrit;*
>
> ***l'analyse de contenu**, qui étudie objectivement, systématiquement et quantitativement un ensemble de documents;*
>
> ***l'analyse de données langagières**, qui se distingue de l'analyse de contenu par la prise en compte du corpus linguistiques produits par les sujets plutôt que des documents;*
>
> ***l'étude comparative**, qui compare les performances de groupes qui se différencient par des variables invoquées, sans que le pouvoir prédictif d'un modèle soit évalué;*

***l'étude corrélative**, qui étudie les relations entre des variables non manipulées;*

***l'étude historique**, qui vise à comprendre des événements en s'appuyant sur des témoignages ou des documents;*

***l'étude évaluative**, qui permet de juger de l'efficacité des enseignements en utilisant des tests standardisés, sans que le pouvoir prédictif d'un modèle soit évalué;*

***l'observation**, qui est effectué par un observateur non-impliqué utilisant des grilles et des échelles et qui ne peut être associée à aucun autre type de recherche descriptive.*

Cette dernière liste ne peut être absolue, certains éléments pourraient être regroupés ou subdivisés. La recherche descriptive doit esquisser un portrait d'une partie de la réalité didactique par une démarche typiquement inductive. Pour notre cas, notre recherche ne peut être une recherche descriptive puisque l'objet de celle-ci est l'environnement informatisé pour l'apprentissage humain qui est à concevoir.

2.5.2 La recherche théorique

La recherche théorique dans le contexte de la didactique de l'informatique devrait permettre à théoriser à propos d'une situation d'appropriation de savoirs. Elle utilise l'analyse conceptuelle qui tente de construire, d'analyser ou de faire la synthèse de concepts associés à un secteur d'intérêt de la didactique de l'informatique afin d'obtenir un modèle théorique opérationnel capable de produire au moins une prédiction nouvelle concernant un aspect de la réalité didactique. Gagné et al. (1989) subdivisent ce type de recherche en huit catégories, soit

***le développement de modèle théorique**, qui tente d'expliquer les relations entre les différentes variables d'un problème;*

***l'application théorique d'éléments disciplinaires**, qui tente d'appliquer des modèles disciplinaires existants à des situations didactiques;*

***l'étude théorique**, qui étudie une situation didactique dans le but de la cerner à l'aide d'une base de concepts;*

> ***l'étude critique**, qui se différencie de l'étude théorique par une argumentation concernant des forces et les faiblesses de la situation étudiée;*
>
> ***l'étude prospective**, qui, suite à une étude, effectue des projections ou des propositions concernant les orientations futures de la situation;*
>
> ***la synthèse des résultats de recherche**, qui opère une intégration descriptive d'un ensemble de recherches par une revue de littérature;*
>
> ***la synthèse critique de résultats de recherche**, qui se distingue de la synthèse de résultats de recherche par le développement d'une critique de la problématique d'ensemble des recherches;*
>
> ***la théorisation de l'action**, qui tente de construire des concepts ou des modèles d'intervention didactique.*

Les réalisations pratiques qui résulteront de notre recherche nous éloignent des objectifs de la recherche théorique. Nous ne sommes pas à la recherche d'une théorie ni désireux de théoriser une situation ou un modèle mais plutôt de concevoir du matériel qui améliorera notre enseignement. Cependant, l'aspect théorique de notre recherche sera développé sous forme de considérations théoriques qui permettront de supporter notre idée de recherche ou même de la rejeter.

2.5.3 La recherche expérimentale

La recherche expérimentale, dans le contexte de la didactique de l'informatique, devrait permettre d'expliquer un ou plusieurs éléments d'une situation d'appropriation de savoirs informatiques. Elle utilise une expérimentation capable de corroborer ou de falsifier une prédiction obtenue par un raisonnement déductif à partir d'un modèle théorique explicite. Gagné et al. subdivisent la recherche expérimentale en trois catégories, soit

> ***la recherche à plan expérimental**, qui compare les performances de groupes formés aléatoirement qui se différencient par des*

variables provoquées et dont les variables parasites sont strictement contrôlées;

***la recherche à plan quasi-expérimental**, qui se différencie de la recherche à plan expérimental par l'utilisation de groupes déjà formés ou dont les variables parasites ne sont pas strictement contrôlées;*

***l'étude évaluative**, qui permet de juger de l'efficacité des enseignements en utilisant des tests standardisés et impliquant une manipulation contrôlée de la situation.*

Nous considérons que notre recherche ne peut s'inscrire dans la catégorie de recherche expérimentale puisque nous ne cherchons pas principalement à produire des connaissances. Encore un fois, les réalisations visées par notre travail ne peuvent être obtenues par ce type de recherche.

2.5.4 La recherche action

Selon Van der Maren (1996) la recherche-action vise le changement plus ou moins radical d'une situation d'éducation avec la participation relative des populations concernées et des intervenants impliqués. Il ajoute que son principe reçoit des colorations différentes selon l'idéologie des auteurs et des acteurs. Gagné et al. (1989) précisent que ce type de recherche vise à transformer un ou plusieurs éléments d'une situation d'appropriation de savoirs scientifiques. Pour arriver à ces fins, la recherche-action utilise au moins une intervention dont elle tente d'évaluer ou d'optimiser les effets dans une démarche cohérente. Elle peut être subdivisée en sept catégories, soit

***L'innovation contrôlée,** qui s'intéresse à une modification spécifique du contenu de l'enseignement ou de l'aménagement pédagogique dont elle tente d'évaluer les effets quantitativement et de façon contrôlée;*

l'innovation structurée, *qui se distingue de l'innovation contrôlée par une évaluation des effets qualitative et non contrôlée;*

l'application contrôlée en classe d'éléments théoriques, *qui s'intéresse à une transformation pédagogique, basée sur des considérations théoriques, dont elle tente d'évaluer les effets quantitativement et de façon contrôlée;*

l'application structurée en classe d'éléments théoriques, *qui se distingue de l'application contrôlée en classe d'éléments théoriques par une évaluation des effets qualitative et non contrôlée;*

le développement contrôlé d'outils pédagogiques, *qui s'intéresse au développement d'outils dont il tente d'évaluer les effets quantitativement et de façon contrôlée;*

le développement structuré d'outils pédagogiques, *qui se distingue du développement contrôlé d'outils pédagogiques par une évaluation des effets qualitative et non contrôlée;*

la théorisation de l'action, *qui tente de comprendre ou d'expliquer à posteriori une intervention en construisant des concepts ou des modèles d'intervention didactique.*

Dans le cadre de notre travail, nous désirons, proposer des ateliers basés sur la pédagogie par projet favorisant la collaboration et développer un environnement d'aide à l'apprentissage de la programmation. Lors de l'évaluation des ces dispositifs nous emprunterons quelques principes de la recherche-action. Toutefois, nous n'inscrirons pas notre recherche dans le cadre de la recherche-action telle que décrite précédemment puisque l'emphase sera mise sur la construction de productions tangibles.

2.6 Le type de production

Il nous faut maintenant classer notre recherche en fonction de son type de production. De ce critère, De Landsheere (1985) propose une subdivision séquentielle de trois types de recherche :

la recherche expérimentale, *qui porte principalement sur la production de connaissances nouvelles;*

la recherche appliquée, *qui porte principalement sur la production d'applications pratiques des connaissances;*

la recherche de développement, *qui porte principalement sur la production de méthodes ou d'instruments.*

Rappelons que le qualificatif séquentiel est utilisé pour signifier la dépendance ou la préséance d'une recherche par rapport à une autre. La figure 2.2 présente la séquence d'ordonnancement de ces trois types de recherche qui se justifie par la production obtenue de chacune. En effet, des connaissances nouvelles obtenues de la recherche expérimentale nous pouvons développer des applications pratiques dans le cadre de la recherche appliquée qui nous inciteront à innover en suggérant des méthodes ou en créant des instruments les intégrant, résultats de la recherche de développement.

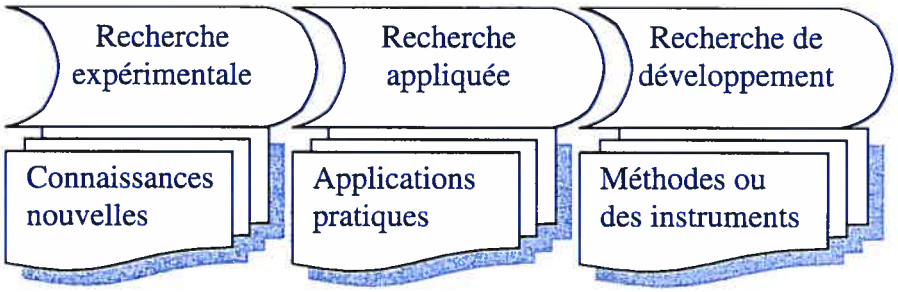


Figure 2.2 Séquence de trois types de recherche ordonnés selon leur production.

Exposer ainsi cette séquence est tout a fait sensé. Par contre, elle ne tient aucunement compte des découvertes historiques où on peut citer plusieurs exemples d'applications pratiques qui ont précédé l'arrivée de nouvelles connaissances. Par exemple, la construction du bateau à moteur a précédé de plusieurs années la théorie formelle de la thermodynamique et l'avion de Clément Ader, celui à qui on doit le terme avion pour désigner cet appareil, effectue en 1890 son premier vol bien avant la théorie formelle de l'aérodynamique.

Conséquemment à cette observation, Nonnon (1993) a opté pour un modèle où la recherche expérimentale et la recherche de développement se déroulent en parallèle en se complétant l'une l'autre. Des précisions s'imposent sur la recherche de développement.

2.7 La recherche de développement

Puisque notre travail se situe dans la recherche de développement, nous présentons dans la prochaine section deux modèles de ce type de recherche : le modèle proposé par Van der Maren et celui proposé par Nonnon.

2.7.1 La recherche de développement de Van der Maren

Van der Maren (1996) distingue trois formes de recherche de développement: le développement de concept, le développement d'objet ou d'outil et le développement ou perfectionnement d'habiletés personnelles en tant qu'outils professionnels.

À partir de certains énoncés théoriques, le développement de concept cherche à créer de nouvelles applications ou à développer de nouveaux outils. L'objectif de ces réalisations est de démontrer la pertinence des énoncés théoriques. Cette forme de recherche de développement englobe plusieurs étapes : la concrétisation de l'usage de l'idée, le développement ou la conceptualisation de l'usage de la théorie, la modélisation, la simulation, les essais de prototypes, les corrections multiples, la recherche d'un marché et la création du besoin.

Le développement d'un objet pédagogique est la réponse à certains problèmes de la pratique quotidienne et utilise diverses théories obtenues de la recherche nomothétique. Cette forme de recherche intéresse particulièrement la didactique et la technologie de l'éducation. Les différentes étapes inspirées de la résolution de problème sont : l'analyse, la conception, l'élaboration de plusieurs stratégies, l'évaluation, la construction d'un prototype, mise au point du prototype (se compose de plusieurs itérations des opérations : essai, évaluation, adaptation, modification) et la mise en marché. Le didacticien adaptera à ces fins cette forme de recherche en ajoutant un volet

collaboratif par l'analyse de l'objet et sa préparation en collaboration avec les enseignants de sorte à tenir compte de facto de leurs contraintes et de leurs priorités. La construction et la mise à l'essai du prototype s'effectuera alors en contexte scolaire plutôt qu'en situation expérimentale contrôlée.

La troisième forme de recherche de développement exposée par Van der Maren est le perfectionnement des habiletés personnelles comme développement d'outils. Distinctement, l'étude porte sur des habiletés ou des connaissances qu'il faut s'approprier selon diverses possibilités plutôt que sur un objet externe. Cette forme de recherche exige la constitution d'une trace primaire de la démarche lors de phase évaluative et le recours à un tiers-témoin analyste lors du développement.

2.7.2 Un modèle de recherche développement technologique en éducation

Nonnon (1993) présente un modèle de recherche qui mise sur la conception et la réalisation de systèmes d'enseignement sans pour autant négliger les considérations théoriques qui ont présidé à leur élaboration et les évaluations successives qui vont permettre principalement de les améliorer. Plusieurs thèses ont été développées en utilisant ce modèle: Henri Boudreault (2002), Fredéric Fournier (2001), Daniel Cervera (1998), Françoise Crevier (1998), ... La particularité de son modèle est de considérer deux points de départ qui commandent des démarches de recherche initialement distinctes selon que l'on part des résultats de la recherche expérimentale ou appliquée, ou que l'on part directement d'une idée de développement. Nonnon justifie son approche en écrivant :

> *Une recherche-développement en éducation pourrait par exemple s'amorcer sur une simple idée, ou sur un problème à résoudre, et elle se garderait alors libre de toutes considérations théoriques à l'étape de la conception, tout comme en recherche industrielle.*

Il est conscient de la nécessité d'appuyer ces travaux sur les théories existantes en éducation et notamment les processus d'apprentissage. Par contre, ces théories ne doivent pas freiner l'élan de créativité du chercheur mais contribuer à une approche systémique. Dans ce sens il ajoute :

> *Il faut imaginer des modèles de recherche-développement qui empruntent à l'approche expérimentale les fondements théoriques et les modalités de contrôle applicables à son objet, sans s'asservir pour autant au modèle expérimental et risquer ainsi de perdre la créativité et l'innovation qui lui sont propres. Il faut envisager ces modèles dans une perspective plus globale, plus systémique.*
>
> *[...] Le modèle de recherche développement que nous proposons doit permettre au chercheur de procéder du général au spécifique, de privilégier, d'intuitionner l'aspect fonctionnel du système en incluant son insertion dans une situation active d'apprentissage.*

Revenons à ce modèle qui considère deux situations de départ de manière à préciser ces étapes. Si le chercheur part des résultats de la recherche expérimentale ou appliquée sous forme d'un problème à résoudre on procède alors à une analyse de type déductive de ce problème de sorte à élaborer son idée. Si, par contre, le chercheur part d'une idée qui semble intéressante à explorer, il confronte alors cette idée aux théories existantes à travers des considérations susceptibles de l'appuyer, la rejeter, l'enrichir de sorte à jauger son originalité. Viennent ensuite pour les deux approches les étapes suivante : élaboration de l'idée, conception d'un modèle d'action pour préciser et opérationnaliser l'idée, réalisation d'un prototype, mise à l'essai fonctionnelle, mise à l'essai empirique et la mise à l'essai systématique. La mise à l'essai fonctionnelle et la mise à l'essai empirique sont destinées à améliorer le prototype tandis que la mise à l'essai systématique sert principalement à valider et à améliorer le modèle d'action. La figure 2.3 présente le modèle et précise les interactions entre les différentes étapes.

En ce qui concerne le point saillant de ce type de recherche, Nonnon (1993) affirme :

> *La démarche la plus importante de ce type de recherche est la conception et la construction du système technologique lui-même. Elle se reflétera par la description de cette démarche de conception, explicitant les caractéristiques et les fonctionnalités du modèle d'action en parallèle avec celles du prototype.*

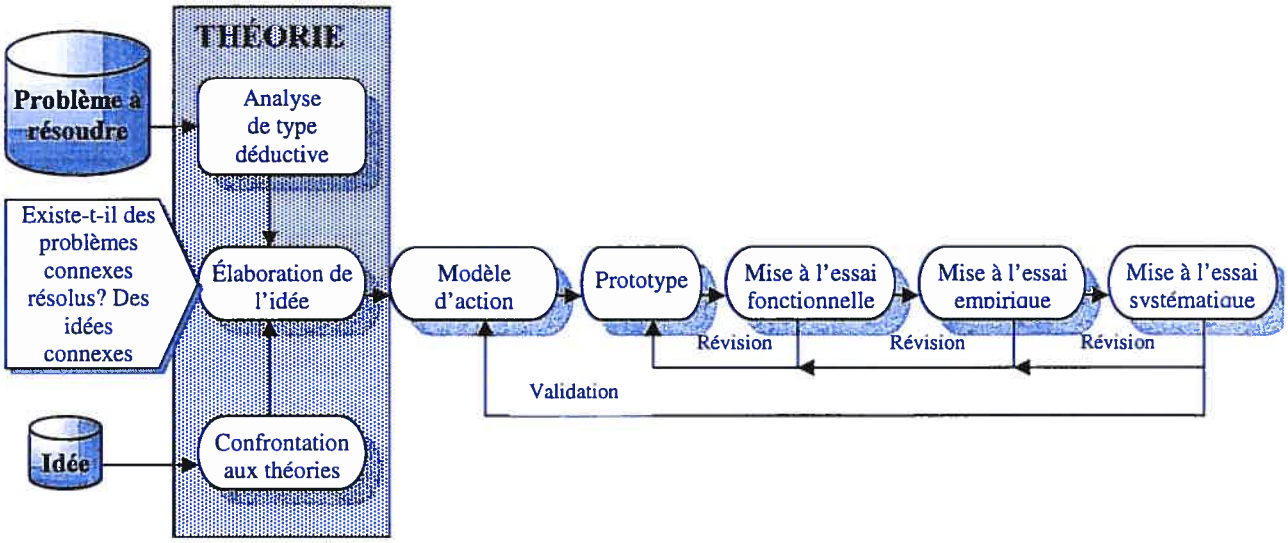


Figure 2.3 Modèle de recherche développement technologique (Nonnon 1993)

Comme nous l'avons précisé au départ, notre travail se situe dans la recherche de développement. Désireux d'offrir à nos étudiants un environnement informatique d'apprentissage humain (EIAH) composé de moyens différents et concrets favorisant l'apprentissage de la programmation, nous recherchions une méthodologie de conception efficace. De plus, la réflexion de notre idée de départ s'étalant sur plusieurs années d'enseignement de la programmation, nous sentions le besoin de confronter cette idée à des théories existantes. Nous désirions également employer une méthode de développement systématique et reconnue par la communauté des chercheurs. Pour ce faire, nous adopterons le modèle de Nonnon qui décrit clairement les étapes à suivre pour arriver à nos fins.

Notre choix de ce type de recherche en éducation peut s'avérer courageux puisque certains chercheurs estiment que les objectifs visés sont moins nobles que ceux de la recherche expérimentale. Cependant, comme le souligne Fournier (2001 :22) :

> *« En éducation l'un des objectifs de la recherche-développement est d'intégrer les nouvelles technologies (NT) dans un environnement d'apprentissage pour le rendre plus performant à un niveau technologique et didactique et de contribuer ainsi à l'avancement des connaissances. C'est grâce à l'arrivée des NT et à cette contribution que*

> *la recherche de développement possède une place toute légitime dans la recherche de type universitaire en éducation. [...] Pour faire de la recherche-développement, il est nécessaire de connaître les possibilités techniques qui s'offrent à nous, pour identifier ce qui est dans le domaine du possible et du réalisable, et avoir des connaissances en apprentissage pour intégrer ce développement dans un cadre théorique de manière à distinguer une recherche développement de type industrielle d'une recherche développement de type académique. »*

La lecture de cette thèse doit se faire en ne perdant jamais de vue que notre recherche ne part pas d'une question de recherche mais plutôt d'une idée de recherche. De même, notre objectif n'est pas l'élaboration d'une théorie sur la connaissance mais plutôt la conception d'un EIAH offrant de nouveaux moyens pour l'apprentissage de la programmation.

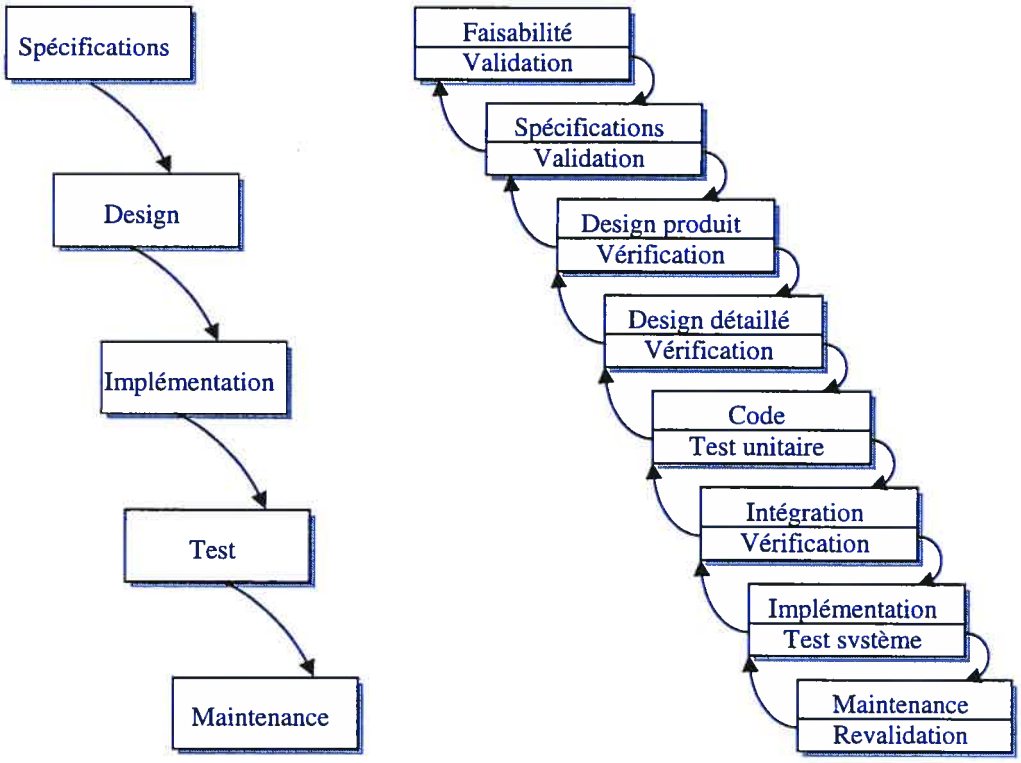
Dans le modèle de Nonnon, la conception du système technologique est la démarche la plus importante. Pour nous ce système se concrétise entre autre par un logiciel. Pour cette raison, nous exposons dans la section suivante quelques modèles classiques de développement de logiciel que l'on retrouve dans la grande majorité des volumes portant sur le sujet.

2.8 Modèle de développement de logiciel

La discipline qui s'intéresse à la modélisation du développement de logiciel est le génie logiciel. À la fin des années 1960, le développement de logiciel vécut une période noire que les journaux ont qualifié de « crise du logiciel ». En effet, certaines études révélaient que les logiciels conçus étaient de piètre qualité, que les développeurs ne respectaient pas les échéanciers ni les coûts.

Il devint impératif pour le génie logiciel de proposer des méthodes de développement qui assurent prioritairement un logiciel de qualité dont la conception respecte l'échéance et les coûts. Un des premiers modèles proposés est le modèle en cascade connu sous le vocable « waterfall model » dont la paternité est attribuée à Royce

(Royce 1970). Ce modèle a le mérite d'identifier dans l'ordre les étapes à suivre pour le développement d'un logiciel. Ce précurseur présente plusieurs lacunes dont la principale est l'absence de vérification et de validation à chaque étape. Un second problème est l'imposition d'un développement séquentiel sans possibilité de retour arrière qui exige de tout prévoir à l'étape du démarrage. Les étapes comprises dans ce modèle débutent par la détermination des spécifications, suit la réalisation du design conceptuel, l'implémentation du design dans le langage de programmation, le test du logiciel et se termine par la maintenance du logiciel opérationnel à l'aide de différents types de corrections de nature perfective ou évolutive. N'incorporant pas de validation à chaque étape, le modèle en cascade est fortement déconseillé. Par contre, d'autres versions modifiées incorporent la vérification et la validation de même que la possibilité de revenir en arrière permettant d'évaluer l'adéquation entre étapes adjacentes. La figure 2.4 présente deux versions du modèle en cascade. La version modifiée contient des étapes supplémentaires correspondant à des raffinements des étapes du modèle initial.



a)Modèle en cascade classique b)Modèle en cascade modifiée

Figure 2.4 Modèle en cascade classique et modifiée

En plus des lacunes citées précédemment, le modèle en cascade classique induit un pourcentage très élevé de l'effort sur les activités de codage et de test du code, révélant par le fait même une déficience au niveau des spécifications et de la conception (design). Toutefois, Pressman (2001) précise que le modèle classique demeure une approche raisonnable lorsque les spécifications sont bien définies. Les ingénieurs logiciels s'activent dès lors à proposer des modèles qui réduisent l'effort de codage de manière à concentrer l'effort de développement sur les spécifications et l'étape de conception.

L'intégration du prototypage dans le modèle de développement permet d'intensifier la tâche de spécification du logiciel. Le prototype est un outil favorisant et facilitant l'interaction entre le développeur et l'usager. Les résultats obtenus à l'aide des outils conceptuels du développeur ne peuvent être déchiffrés ni validés par l'usager. Par contre, l'utilisation du prototype, correspondant grosso modo à l'interface du logiciel, permettra à l'usager de valider aisément l'efficience de ses spécifications. Le prototype est l'outil tout désigné dans le cas d'un logiciel possédant une interface usager importante. Cependant, il doit être utilisé avec parcimonie puisqu'il peut mener à des fonctionnalités superflues, un design de qualité moindre et à un logiciel difficile à maintenir qui offrira de piètres performances à l'exécution. Un prototype peut être évolutif de sorte à devenir le logiciel cible comme il peut n'être qu'un outil éphémère jetable une fois les spécifications clairement établies.

Tel que proposé par le modèle en cascade, le développement ne peut s'effectuer efficacement en procédant successivement de la première à la dernière étape. Nous avons insisté sur l'importance de la validation des résultats de l'étape précédente par l'étape suivante. Cette idée nous conduit inévitablement vers des modèles où le développement est un processus itératif. De cette manière, le produit s'accroît et évolue d'un cycle à l'autre. Boehm (1988) propose un modèle en spirale, qui inclut les étapes des modèles précédents, pour lequel chaque convolution de la spirale s'applique à identifier le sous problème possédant le risque le plus élevé et lui trouver une solution. Le risque d'un sous problème correspond à l'importance de la résolution de ce problème pour la réussite du produit final. L'itération ne consiste pas à répéter les

étapes de la première à la dernière. En effet, d'une itération à l'autre, la convolution de la spirale correspondante comprend un ensemble d'étapes communes à toutes les itérations et un ensemble d'étapes particulières. La figure 2.5 présente le modèle en spirale de Boehm.

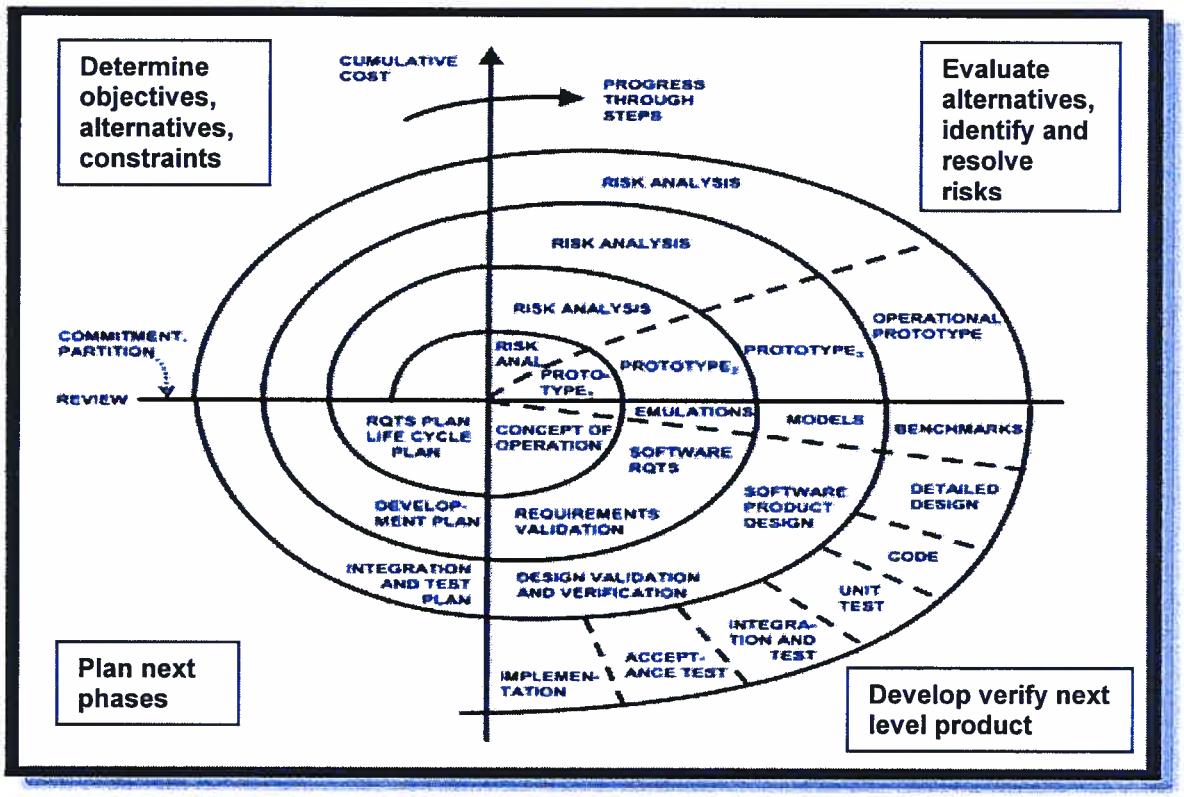


Figure 2.5 Le modèle en spirale (source B.W. Boehm, A spiral model of software development and enhancement, IEEE Computer 21, 5 (1988) © 1988 IEEE)

Le modèle en spirale conçoit un produit basé sur un prototype évolutif. L'emphase du développement est mise sur la détermination des spécifications et la conception au sens du design. En fin de parcours, on retrouve différents types de test, étapes nécessaires avant l'opérationnalisation du produit.

Le modèle en spirale offre plusieurs avantages mais il impose un temps de développement important. La compétition étant des plus vives dans le domaine des produits informatique et comme le dit si bien l'adage populaire, « le temps c'est de l'argent », les développeurs se penchent vers des processus permettant un

développement rapide. Le modèle RAD (Rapid Application Development) est un processus de développement logiciel incrémental misant sur la réutilisation de composants logiciels existants et la construction de composants réutilisables. Il correspond à une adaptation des modèles séquentiels linéaires de manière à permettre un développement à grande vitesse. Les phases de développement du modèle RAD sont: modélisation de l'entreprise, modélisation des données, modélisation des processus et test. D'autres modèles permettent des développements rapides, parmi ceux-ci citons : XP pour eXtreme Programming (Beck, 2000) et FDD pour Feature-Driven Development (Coad et al, 1999).

Les modèles les plus récents s'intéressent plus particulièrement au développement orienté objet (Graham, 2001). C'est ainsi qu'après avoir développé un langage de modélisation unifié appelé UML (Unified Modeling Language) permettant de décrire les modèles objets, Grady Booch, Ivar Jacobson and James Rumbaugh consacrèrent leurs efforts au processus de développement logiciel et créèrent RUP (the Rational Unified Process) lequel est un cadre de travail général utilisé pour décrire un processus de développement spécifique (Krutchen,1999 ;Jacobson et al, 1999). L'essence de RUP est l'itération qui se termine à chaque fois par un livrable, préférablement sous forme d'un exécutable. La production d'une itération peut aussi être un diagramme UML ou de toute autre forme. Les quatre phases du cycle de développement RUP sont : choix du concept (inception en anglais), élaboration (design), construction et transition (transfert à l'usager). Une version adaptée de RUP, appelée UPEDU (the Unified Proces for EDUcation), a été créée pour faciliter son apprentissage (Robillard et al, 2002).

Nous nous sommes inspirés des modèles itératifs lors du développement de notre environnement informatique pour l'apprentissage humain. Plus spécifiquement, nous avons conçu notre application comme un prototype évolutif. Nous avons mis l'emphase du développement sur la détermination des spécifications. Nous avons également tenté de tirer profit des composants logiciels existants pour l'élaboration de l'interface.

2.9 Démarche spécifique à notre recherche

Notre problématique vise l'apprentissage de la programmation chez une clientèle universitaire en génie. Nous avons identifié dans notre problématique certains facteurs pouvant ralentir l'apprentissage de l'étudiant. Ces facteurs, au nombre de quatre, se résument à l'abstraction en vue de la transposition de la solution dans le langage informatique, la détection-interprétation-correction d'erreurs, la nécessité d'intégrer les connaissances du domaine d'application en plus des connaissances du domaine informatique et à un autre niveau l'hétérogénéité de la clientèle étudiante. L'essence de notre projet s'exprime tout simplement par la question :

> Quelles sont les interventions qui permettront d'interagir sur ces facteurs de manière à améliorer l'apprentissage de la programmation?

Notre solution s'inspire de principes liés à la pédagogie active. Nous désirons permettre à l'étudiant de s'investir plus intensément dans son apprentissage. Nous lui proposerons des petits projets qui intégreront des exercices de programmation avec des objets pour penser de manière à rendre signifiante l'activité de programmation et à concrétiser les structures de programmation. Afin d'offrir en tout temps et juste à temps l'information nécessaire à la réalisation de son projet, nous développerons un outil complémentaire d'aide à l'apprentissage où l'étudiant pourra trouver réponses à ses questions. Ces deux éléments composeront un environnement informatisé pour l'apprentissage humain de la programmation.

Nous allons, dans un premier temps, confronter notre idée aux théories existantes. Le caractère multidisciplinaire de notre étude nous emmènera à faire des incursions dans des domaines variés tels la didactique, le génie logiciel et la pédagogie.

L'analyse de plusieurs études sur l'apprentissage de la programmation réalisées dans le domaine du génie logiciel nous permettra de mieux cerner la problématique de l'apprentissage d'une programmation utile et actuelle et nous sera d'une grande utilité afin d'identifier différentes stratégies de programmation employées par les étudiants. La

méthode pédagogique à envisager devra favoriser l'emploi adéquat et efficace de ces stratégies.

Nous tenterons d'appuyer notre travail sur des principes reconnus de la didactique. Nous savons que les didactiques des mathématiques et des sciences ont grandement contribué à l'apprentissage de leur domaine respectif. Il sera alors intéressant d'examiner la possibilité de transposer cette expertise développée dans ces autres domaines au domaine de l'informatique et plus spécifiquement dans le cadre de cette recherche à l'apprentissage de la programmation.

Une autre facette sur laquelle nous miserons concerne l'apprentissage collaboratif inscrit dans une pédagogie par projet. Nous appuierons l'élaboration des activités de programmation sur les théories du domaine de l'apprentissage collaboratif afin d'en soutirer le plus grand bénéfice possible. La littérature sur l'apprentissage collaboratif est abondante et concerne tous les domaines et tous les niveaux académiques, du primaire à l'université.

Lors de la présentation d'un concept abstrait, les informaticiens tentent régulièrement de le comparer à un objet concret connu. Cette comparaison a comme principal objectif d'aider à l'apprentissage d'une nouvelle notion à partir d'une ancienne notion plus connue et maîtrisée. La notion connue peut être appelée un analogue et cette méthode correspond à un enseignement par analogie. Dans cet esprit, nous examinerons les possibilités d'utilisation de l'analogie pour favoriser l'apprentissage des concepts de la programmation.

Une fois les considérations théoriques réalisées, nous passerons à l'élaboration de notre idée. Rappelons que nous désirons développer un environnement informatique pour l'apprentissage humain (EIAH) pour l'apprentissage de la programmation. Cet environnement comprend un tutoriel et des ateliers correspondant à des séances de travaux pratiques de programmation qui feront intervenir du matériel concret pour soutenir le raisonnement des étudiants. Dans cette recherche de développement technologique nous construirons un montage électronique commandé par l'entremise

d'un microcontrôleur qui recevra ses commandes de l'ordinateur. La création du tutoriel et du montage électronique s'effectuera en parallèle mais inévitablement notre énergie se concentrera tantôt sur l'un des dispositifs, tantôt sur l'autre.

Nous effectuerons, à travers plusieurs activités, l'opérationnalisation de notre modèle d'action développant et améliorant progressivement un prototype matériel et logiciel. Ces activités sont : la fabrication d'un prototype, la mise à l'essai fonctionnelle, la mise à l'essai empirique et la mise à l'essai systématique. Les phases suivantes décrivent séquentiellement et spécifiquement notre démarche.

Phase 1

Le développement d'un environnement informatique pour l'apprentissage humain est un travail de longue haleine. Dans un premier temps, nous établirons les spécifications d'un premier prototype. Pour ce faire, nous soumettrons un questionnaire électronique aux étudiants afin de nous assurer que notre application répondra adéquatement aux besoins de la clientèle étudiante. Une fois les spécifications complétées, nous passerons à l'étape de programmation.

Phase 2

Le prototype obtenu sera expérimenté en classe et les étudiants seront invités à l'utiliser hors classe. Il sera également présenté dans le cadre d'un colloque maison.

Phase 3

Une phase exploratoire sera lancée afin d'identifier le matériel à incorporer dans la construction des ateliers. Un choix crucial sera celui du micro contrôleur. Nous profiterons pour cela de l'expérience du laboratoire de robotique pédagogique. Un premier atelier sera conçu et se prêtera à notre propre expérimentation.

Phase 4

Les essais du premier prototype EIAH nous ayant dévoilé ses forces et ses faiblesses, nous pourrons dès lors travailler à son évolution. Encore une fois, un cahier de spécifications sera rédigé initialement. Par la suite, l'activité de programmation du second prototype sera effectuée s'inscrivant dans un cycle de test, de validation et de modification.

Phase 5

Le prototype et les ateliers seront mis à l'essai dans le cadre d'un cours. Un questionnaire sera présenté aux étudiants afin de recueillir leurs impressions.

Ce chapitre nous a permis de situer notre recherche selon différents critères tels son enjeu : ontologique; son domaine de recherche : les sciences de l'homme; son champ disciplinaire : la didactique de l'informatique; son secteur d'intérêt : la construction de situations didactiques; son objectif principal de recherche : transformer; sa démarche centrale d'investigation : le développement structuré d'un outil pédagogique.

Chapitre 3

CONSIDÉRATIONS PRATIQUES ET THÉORIQUES

3.1 Considérations sur l'enseignement actuel et la clientèle étudiante

Dans cette première section, les considérations porteront essentiellement sur les pratiques d'enseignement typique des rudiments de la programmation dans quelques cours universitaires et sur l'analyse du groupe qui participera à notre étude.

3.1.1 Pratiques d'enseignement universitaire

Nous exposons ici l'enseignement le plus couramment utilisé dans le cadre d'un premier cours universitaire portant sur la programmation. Nos informations, obtenues des sites électroniques officiels de chaque institution, concernent les universités: de Montréal, du Québec à Montréal, McGill, Concordia, l'École des technologies supérieures et l'École Polytechnique de Montréal. Le paradigme de programmation abordé est le paradigme procédural ou le paradigme objet. Les principaux langages enseignés sont : C, C++, Ada et Java.

La majorité des cours voués à l'apprentissage de la programmation ou d'un langage de programmation utilise un modèle d'enseignement constitué de trois heures de présentation théorique, auxquelles s'ajoutent une à trois heures d'exercices pratiques par semaine. La portion théorique s'exprime sous la forme d'un exposé oral présentant principalement la syntaxe du langage et l'utilisation typique de structures de programmation. La période de pratique en laboratoire d'informatique permet d'appliquer

la matière vue en classe à l'aide de quelques exercices de programmation. Le travail s'échelonne sur une période ou deux de laboratoire et est réalisé sous la supervision du professeur et/ou d'un assistant. La proximité hebdomadaire théorie-pratique permet de donner les bases et les structures de la programmation avant de les faire pratiquer en laboratoire en favorisant une rétroaction immédiate de l'enseignant. Dans certains cas, ce cycle théorie-pratique est enrichi d'un projet intégrateur exigeant l'élaboration d'un programme de plus grande envergure nécessitant plusieurs périodes de laboratoire et inversant le cycle théorie-pratique de manière à permettre à l'étudiant de concevoir, explorer et expérimenter.

La présentation théorique place l'étudiant en situation d'absorption de la matière. L'enseignant a, dans ce cas, la principale responsabilité de l'apprentissage. Il est le spécialiste et le propriétaire de la connaissance à transmettre. De son côté l'étudiant doit s'efforcer d'assimiler les différentes notions enseignées et s'assurer de leur compréhension par une validation s'exprimant sous la forme de questions adressées au professeur lorsque cela est possible.

Plus riche, plus percutante du point de vue acquisition de connaissance, la réalisation d'exercices plonge l'étudiant en situation d'exploration et de validation de ses apprentissages. Cette activité lui permet d'appliquer explicitement les notions vues en classe. Les exercices sont bien ciblés de manière à découper la matière notion par notion afin d'être ordonnés par niveau de difficulté. La finalité de ces exercices est de s'assurer que l'étudiant soit capable d'utiliser l'ensemble des structures vues en classe. Il est opportun, voire indispensable en bout de ligne de proposer un problème intégrateur qui sollicite plusieurs notions. Une fois les notions élémentaires de programmation enseignées telles les structures répétitives, les structures décisionnelles, les structures de décomposition (sous-programme) et les structures de données, l'étudiant doit réaliser un programme d'envergure étalé sur plusieurs semaines. Ce problème nommé généralement mini projet ou projet aura une grande répercussion sur l'assimilation des notions par l'étudiant. Nous parlons ici de problème plutôt que d'exercices puisqu'il doit être résolu en respectant une méthode de résolution de problèmes adaptée au développement

logiciel. La résolution du problème s'exécute alors à travers les étapes suivantes: définition, analyse, conception, codage et test. Cependant, l'ensemble des exercices ainsi que le projet nécessitent une abstraction des entités du domaine de l'application afin de permettre une transposition dans le langage de programmation. Réaliser un programme qui permettra de connaître la moyenne et l'écart type des notes des étudiants d'une classe exige de mémoriser les notes dans un fichier ou dans un tableau et de traduire les formules du calcul de la moyenne et de l'écart type dans le langage de programmation. Même s'il s'agit d'automatiser une procédure de calcul que l'étudiant a pu réaliser maintes fois, les entités à manipuler ici sont abstraites. Une autre lacune que l'on peut attribuer à cette dernière activité est que l'ensemble du travail est planifié explicitement ou implicitement par le professeur, diminuant par le fait même la responsabilité de l'étudiant envers son propre apprentissage en plus de lui tracer un parcours restreint vers la solution. On se contente alors de voir ici étape par étape s'il est capable de transférer les structures de programmation apprises au cours magistral avec l'exemple du professeur sur d'autres exemples.

Pour notre part, à l'École Polytechnique de Montréal, outre un volume de référence ou des notes de cours dédiées spécifiquement au cours, l'étudiant a accès à un site électronique offrant du matériel divers: anciens examens avec leur corrigé, matière sous forme textuelle ou d'acétates, exerciseur de type QCM et des outils d'interaction usuels. Le principal avantage du site est l'accès à l'information désirée en tout temps et de n'importe où.

L'évaluation de l'apprentissage s'effectue généralement à l'aide d'un contrôle périodique de mi-session, d'un examen final, des travaux en laboratoire. Le contrôle périodique et les travaux en laboratoire sont extrêmement utiles vu leur richesse qui sera basée sur la qualité et la disponibilité de rétroaction. Le projet incite à l'intégration des concepts enseignés et l'examen final vérifie le niveau de cette intégration. La figure 3.1 illustre le modèle d'enseignement décrit sous la forme d'une spirale qui tourne autour de quatre activités : la théorie, l'exercice, l'évaluation et la rétroaction.

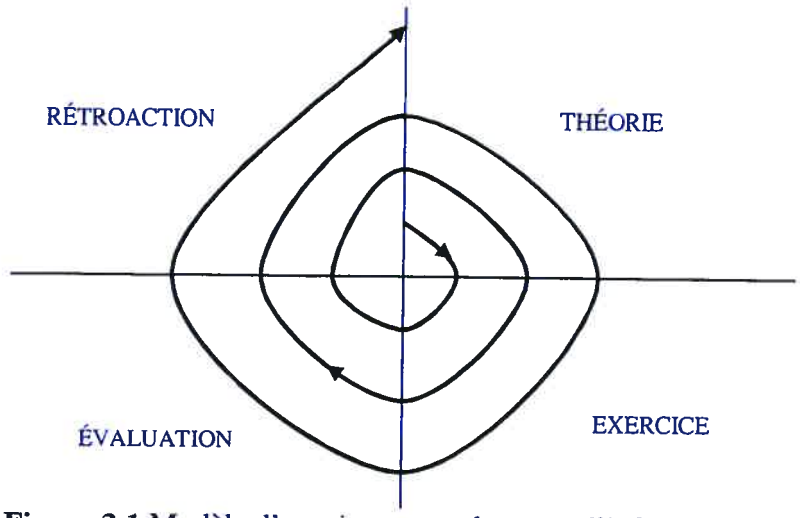


Figure 3.1 Modèle d'enseignement de cours d'informatique

Du point de vue apprentissage, la faiblesse de ce modèle réside dans la passivité de l'étudiant lors du cours théorique. Pendant ces périodes de cours théoriques, l'étudiant assiste à la prestation de l'enseignant bien assis sur sa chaise. De temps à autre, il peut être sollicité pour répondre à une question. La passivité de l'étudiant n'est pas très propice à l'apprentissage, il est connu que dans cette situation le taux de rétention est typiquement faible. Le niveau d'attention de l'étudiant est directement lié à son intérêt pour le sujet et à l'intensité fournie par le professeur lors de sa présentation. Comme l'indiquent les résultats du professeur Robert Carpick du département de génie et physique de l'Université du Wisconsin, les étudiants ne peuvent maintenir leur attention pendant les 50 minutes d'un exposé de cours théorique. D'autres études montrent que l'attention des étudiants tend à diminuer après 10 minutes environ. Une étude a déduit que les étudiants retiennent environ 70% des propos des 10 premières minutes mais seulement 20 pourcent des 10 dernières minutes (Daniel, 2002).

Ces dernières constatations orientent prioritairement notre intervention vers l'amélioration des périodes de cours théoriques. Durant ces périodes, il faut permettre à l'étudiant de ne pas être passif, mais de se mettre en action, de faire plutôt que d'écouter de sorte à l'inciter à prendre le plus rapidement possible son apprentissage en main.

3.1.2 Étude de la clientèle étudiante

Nous présentons dans cette section les résultats obtenus d'un questionnaire électronique (Annexe D) servant à mieux connaître le groupe participant à notre étude dans le double but de concevoir un environnement informatisé pour l'apprentissage humain adéquat et de proposer des améliorations à notre enseignement qui recevront l'aval des étudiants. Nous avons soumis ce questionnaire aux étudiants afin de connaître leur intérêt pour la programmation, leur connaissance initiale en informatique, leur expérience en travail collaboratif afin d'identifier certaines attentes et certaines préférences. Il devrait nous permettre de nuancer notre programme d'apprentissage en tenant compte des réponses des étudiants qui s'écartent du standard typique de l'étudiant de première année de polytechnique afin de mieux répondre à l'hétérogénéité de notre clientèle. Dans un deuxième temps, nous présenterons au chapitre 5 les résultats d'un questionnaire soumis après la réalisation de l'atelier et nous analyserons les programmes produits.

Nous réaliserons notre expérimentation dans le cadre du cours ING1025 Informatique de l'École Polytechnique de Montréal à l'automne 2002. Ce cours enseigne les rudiments de la programmation basés sur une méthode de résolution de problèmes. Près de six cent étudiants y sont inscrits et répartis sur douze groupes. Nous avons choisi, non pas au hasard, mais bien intentionnellement le seul de ces groupes participant au projet Essaim[1] qui exploite l'informatique mobile. L'organisation de nouvelles activités est beaucoup plus simple pour ce groupe puisque chaque étudiant possède son propre ordinateur portable auquel il est possible de connecter du matériel, de modifier la configuration d'un logiciel ou d'ajouter des logiciels. Il est également important de préciser que tous les étudiants, soit quarante-quatre, sont inscrits au programme de génie logiciel.

---

[1] En 2001-2002, le nouveau programme de génie logiciel du Département de génie informatique a implanté une nouvelle approche pédagogique qu'il a appelée «formule ESSAIM» (Enseignement supérieur des sciences appliquées avec l'informatique mobile). Cette formule favorise la meilleure intégration pédagogique possible de l'ordinateur portable dans l'enseignement.

Trente-cinq étudiants ont volontairement complété notre questionnaire, soit six femmes et vingt-neuf hommes. Les résultats des questionnaires dévoilent qu'un seul étudiant possède une moyenne globale au collégial inférieure à 70% tandis que huit étudiants ont conservé une excellente moyenne globale supérieure à 90%. Tous les autres, soit vingt-six, ont une moyenne comprise entre 70 et 89%.

3.1.2.1 Intérêt pour l'informatique, intérêt pour le cours

Du point de vue de l'intérêt porté au cours, 29 étudiants s'avouent grandement intéressés (première priorité ou priorité élevée) tandis que 6 étudiants ont un intérêt mitigé. Un peu surprenant pour des étudiants inscrits en génie logiciel. À la recherche d'une meilleure précision nous leur avons proposé de jauger trois affirmations selon l'échelle: fortement en désaccord, en désaccord, d'accord et fortement d'accord. Ces trois affirmations sont:

- Je considère que l'informatique est un sujet d'étude fascinant.
- J'anticipe découvrir mes intérêts concernant certains défis de l'informatique dans cette classe.
- J'anticipe à être stimulé par les nouvelles idées proposées dans cette classe.

Les résultats obtenus sont présentés à l'aide d'un graphique à la figure 3.2. Ces données confirment le grand intérêt de la majorité des étudiants du groupe envers l'informatique et plus précisément envers le cours. Les données montrent clairement qu'une grande majorité d'étudiants est en accord avec chacune des trois affirmations. A la lumière de ces résultats, nous croyons que l'attitude des étudiants est très positive autant à l'égard du cours que dans l'expérimentation de nouvelles idées.

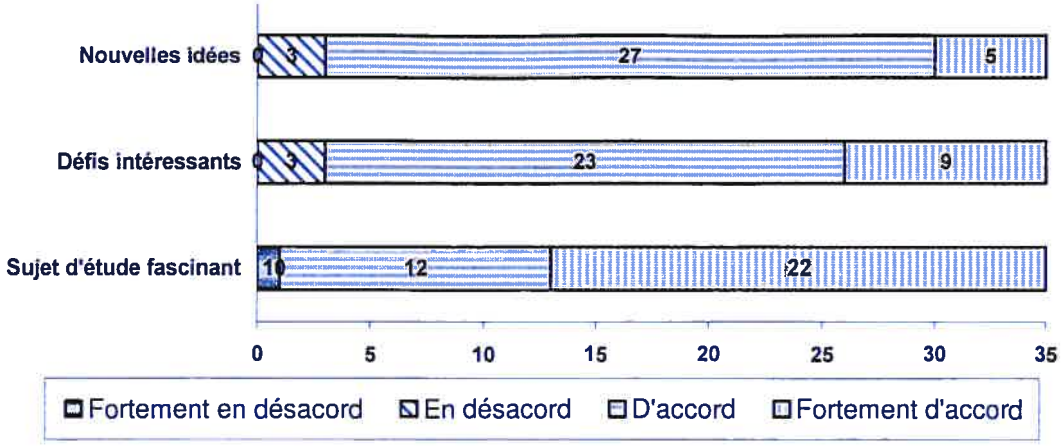


Figure 3.2 Intérêt des étudiant envers l'informatique et le cours

3.1.2.2 Connaissance initiale en informatique

Nous désirons connaître ensuite les connaissances préalables en informatique des étudiants. Pour ce faire, nous avons vérifié l'enseignement concernant l'informatique au secondaire et au collégial. Les résultats présentés à la figure 3.3 nous révèlent que la majorité des étudiants n'ont suivi qu'un seul ou aucun cours d'informatique dans les deux niveaux académiques. De plus, dans notre groupe cible l'étudiant du secondaire semble mieux servi en formation informatique que celui du collégial. Nous n'avons pas vérifié la teneur des cours en question tant au secondaire qu'au collégial. Il ne faut pas oublier également que l'étudiant passe 5 ans au secondaire contre 2 ans au collégial.

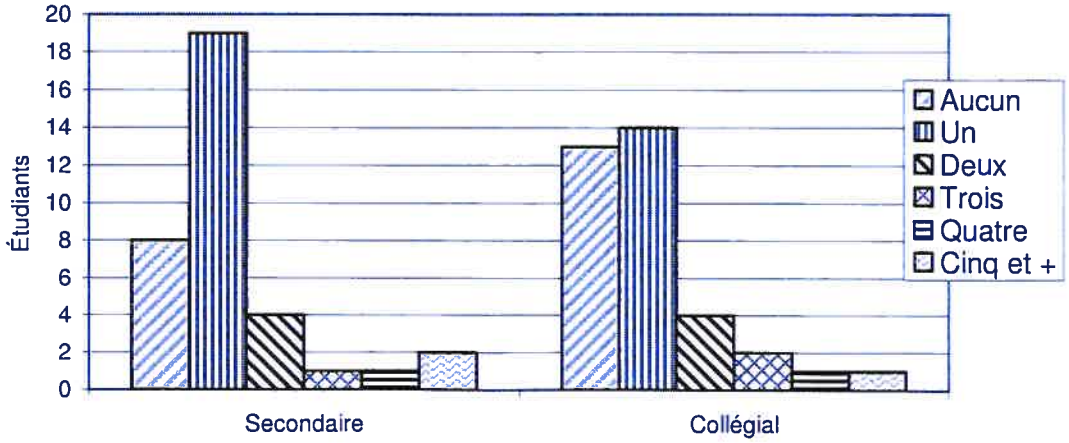


Figure 3.3 Nombre de cours suivis concernant l'informatique

Dans le but, de vérifier leur niveau de connaissance en programmation, nous avons essayé d'identifier les langages appris. Nous aimerions porter un bémol à ces résultats. En effet, un étudiant peut prétexter connaître un langage informatique sans pour autant en démontrer une certaine maîtrise. Malheureusement, il n'existe pas de moyen simple et sûr de vérifier cette information. Ici, nous avons prévu la possibilité qu'un étudiant puisse avoir été confronté à plus d'un langage informatique, cela expliquant le total de réponses supérieur à 35. De cette question nous obtenons les informations suivantes: 12 étudiants disent ne connaître aucun langage de programmation, 13 étudiants disent connaître un seul langage, 5 étudiants disent connaître deux langages, 3 étudiants disent connaître trois langages alors que 2 étudiants disent connaître les quatre langages proposés dans le questionnaire. Le tableau 3.1 résume ces informations pour chaque langage.

Tableau 3.1 Langage de programmation connu

Langage	Nombre d'étudiants
Aucun	**12**
BASIC	**15**
PASCAL	**5**
C	**10**
C++	**10**

3.1.2.3 Attentes et préférences

Nous désirions par la suite tracer un profil de l'apprenant, un profil nous indiquant leurs attentes et leurs préférences. Pour ce faire, nous leurs avons demandé de jauger selon la même échelle de valeurs que précédemment, les affirmations suivantes: attentes et préférences

- Je m'attends à travailler ardemment dans ce cours.
- Je m'attends à rencontrer certaines difficultés dans ce cours.
- Je préfère les problèmes qui ont seulement une seule solution.
- Je préfère les principes généraux plutôt que les notions pointues.

- Je suis excellent dans la mémorisation des faits.

De prime abord, 22 étudiants s'attendent à travailler ardemment tandis que 13 ne le pensent pas. Ce résultat nous suggère que 13 étudiants seraient prêts, confiants et que 22 appréhenderaient les nouvelles notions enseignées. Les étudiants sont presque également partagés en ce qui concerne la rencontre de difficultés, 16 entrevoient un cours sans difficultés et 19 avec difficultés. Du point de vue, des problèmes à solution unique, les étudiants préfèrent nettement ceux qui ont plusieurs solutions possibles, 26 contre 9. Nos répondants préfèrent, à notre grande surprise 20 contre 15, les notions pointues plutôt que les principes généraux. Finalement, 25 estiment être excellents dans la mémorisation des faits.

En résumé, nos étudiants s'attendent à devoir travailler, à rencontrer plus ou moins de difficultés, préfèrent les problèmes dont la solution n'est pas unique et les notions pointues plutôt que les principes généraux. Le graphique de la figure 3.4 présente les résultats obtenus.

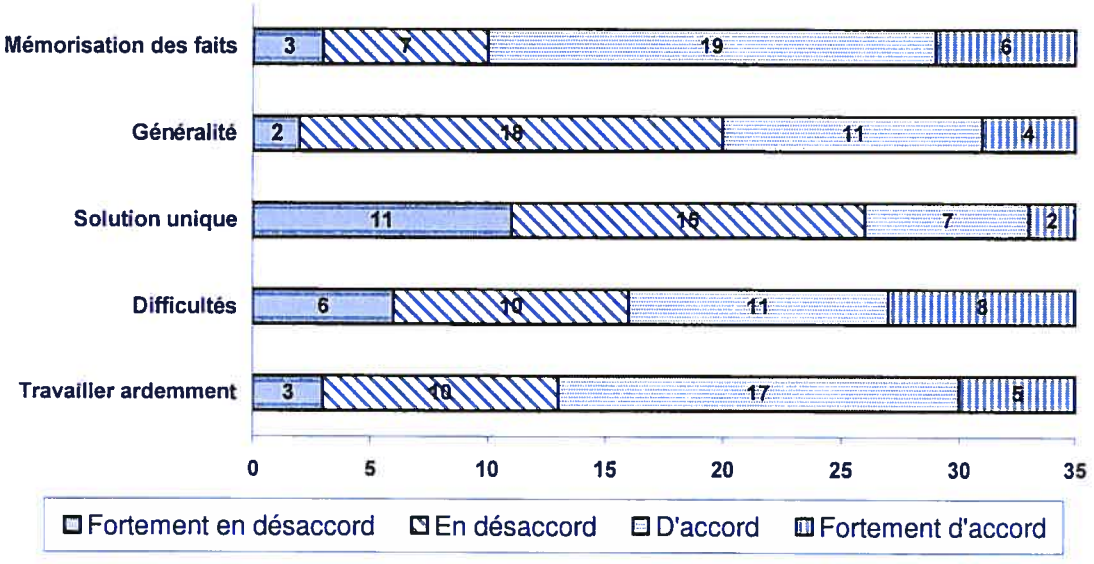


Figure 3.4 Profil de l'apprenant

3.1.2.4 Expérience en travail collaboratif

Le travail collaboratif est à la base d'un enseignement renouvelé axé sur la pédagogie active. Il apparaissait important de connaître l'expérience en travail collaboratif de nos étudiants. Pour ce faire, nous leur avons demandé de préciser le nombre de cours suivis au secondaire et au collégial dont la méthode d'enseignement se basait principalement sur le travail en petit groupe.

Les résultats sont présentés à l'aide du graphique de la figure 3.5. Ce graphique nous révèle qu'un peu plus de 50% des étudiants n'ont pas eu d'enseignement basé sur un apprentissage collaboratif au secondaire ou au collégial. De plus, 20% des étudiants n'ont reçu ce type d'enseignement ni au secondaire ni au collégial. Finalement, tant au secondaire qu'au collégial on peut conclure, à partir des résultats obtenus, que les étudiants ont rarement la chance d'apprendre par une collaboration avec les pairs.

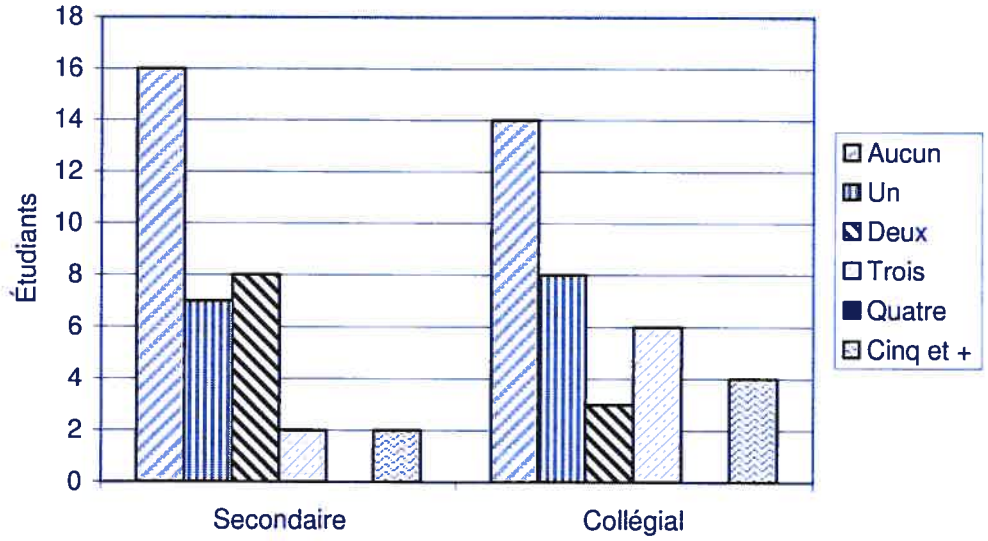


Figure 3.5 Nombre de cours permettant le travail en équipe

La question qui se pose maintenant est de savoir si les étudiants apprécient ou non le travail en équipe. Pour ce faire, nous leur avons demandé d'évaluer selon la même échelle de valeurs l'affirmation suivante:

- Je préfère résoudre les travaux à la maison individuellement plutôt qu'avec d'autres étudiants.

Sur le graphique présenté à la figure 3.6, nous voyons que 12 étudiants sur 35 préfèrent réaliser leurs travaux individuellement à la maison plutôt qu'en équipe avec d'autres.

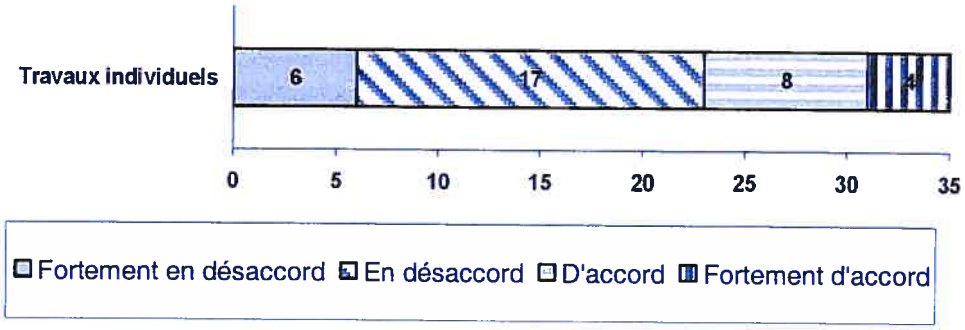


Figure 3.6 Travaux individuels ou en équipe

Par la suite, nous avons scruté leur perception et leur motivation à travailler en équipe. Les deux affirmations à évaluer selon l'échelle de valeurs sont :

- Les meilleurs travaux en informatique sont réalisés en équipe
- J'aime l'idée de travailler en équipe sur un projet

Les résultats apparaissent à la figure 3.7. Le graphique révèle des étudiants en accord quasi unanime avec ces deux affirmations. Nous estimons que les étudiants sont enthousiastes et motivés à travailler en équipe.

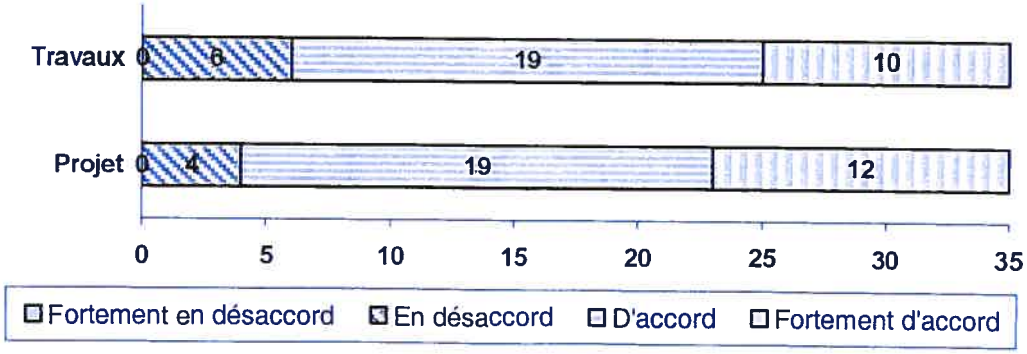


Figure 3.7 Le travail en équipe

Les étudiants étant régulièrement surchargés de travail, notre expérience nous a montré qu'ils apprécient et même exigent des travaux d'équipe afin de diviser la charge de travail entre les différents coéquipiers. Malheureusement, il ne s'agit pas dans ce cas d'une réelle collaboration. La collaboration doit susciter une interaction entre les participants de sorte à permettre une entraide. En effet, selon Brigitte Cord (2000),

> *« nous parlerons de travail collaboratif lorsque la cible commune du travail d'une équipe consiste, outre le travail en groupe, en la réalisation d'un produit final. Par travail collaboratif, nous désignons donc, d'une part, la coopération entre les membres d'une équipe et, d'autre part, la réalisation d'un produit fini.»*

Dans cette définition, la coopération est un élément clé. Ainsi, Brigitte Cord clarifie la coopération à l'aide de Lopriore (1999),

> *« L'apprentissage coopératif est une activité d'apprentissage en groupe, organisée de façon à ce que l'apprentissage soit dépendant de l'échange d'informations socialement structuré qui s'effectue entre les apprenants du groupe. C'est également une activité dans laquelle l'apprenant est responsable de son propre apprentissage et motivé pour participer à l'apprentissage des autres. »*

Ajoutons finalement que Cord distingue la coopération de la collaboration en précisant qu'une coopération n'engendre pas nécessairement la création d'une œuvre commune tandis que la collaboration nécessite la réalisation d'un produit final.

Afin de vérifier l'intention d'une bonne collaboration de l'étudiant nous lui avons demandé de préciser le rôle qu'il entend jouer dans la réalisation d'un projet de programmation en équipe. Par la suite, nous tenterons d'estimer sa capacité de travail en équipe par sa disposition à venir en aide aux autres.

En espérant que le leader actif ne fera pas qu'organiser les activités du groupe, on peut croire que les étudiants veulent contribuer activement au travail d'équipe. 14 étudiants se voient comme leader du groupe, 12 envisagent une contribution réelle et autonome, 8 une participation supervisée. Des répondants, un seul étudiant avoue son désintéressement du travail en groupe. Les résultats sont consignés au tableau 3.2.

Tableau 3.2 Rôle de l'étudiant dans la réalisation d'un projet de programmation

Lors d'un travail de groupe d'étudiants sur un projet de programmation, j'anticipe que mon RÔLE sera:	**Nombre d'étudiants**
Leader actif aidant à organiser les activités du groupe	14
Participant réalisant sa part du travail	12
Participant réalisant sa part du travail avec l'aide des coéquipiers	8
Désintéressé du groupe, mais réalise sa part du travail	1

En ce qui concerne la disposition des étudiants à venir en aide aux autres, les résultats du tableau 3.3 sont révélateurs. En effet, uniquement deux étudiants avouent que l'idée ne leur plait pas. Tous les autres, l'ont déjà fait ou sont disposés à le faire.

Tableau 3.3 Disposition à venir en aide

Quel sentiment éprouvez vous lorsqu'on sollicite votre aide pour un travail de programmation?	**Nombre d'étudiants**
Je l'ai déjà fait et généralement j'aime bien	20
On n'a pas encore sollicité mon aide, mais je pense que j'aimerais cela	4
J'essaie, mais je ne suis pas toujours d'une bonne aide	9
L'idée ne me plait pas et j'évite cela	2

Ces deux derniers résultats sont de bon augure pour le travail en équipe. Ainsi, dans notre proposition d'améliorations du modèle d'enseignement, nous examinerons la possibilité de miser sur la pédagogie par projet et par le fait même la collaboration entre équipiers.

Dans cette section nous avons exposé l'enseignement le plus couramment utilisé dans le cadre d'un premier cours portant sur la programmation et nous avons obtenu le profil de notre clientèle étudiante. Il nous apparaissait important, dans un premier temps, d'examiner les pratiques d'enseignement de la programmation dans le double but de nous assurer d'une certaine originalité et de vérifier la possibilité d'intégration de notre idée. Dans un deuxième temps, il nous apparaissait essentiel de bien connaître notre clientèle étudiante avant de leur proposer de nouvelles activités. Soulignons, entre autres, l'hétérogénéité de nos étudiants du point de vue des connaissances préalables en informatique, par exemple, les langages informatiques connus ou le nombre de cours suivis en informatique. De plus, il nous aurait été très difficile de décrire adéquatement la performance et la perception des étudiants concernant nos propositions sans connaître leurs connaissances préalables en informatique et leurs attitudes face au travail collaboratif.

Les enseignants multiplient les efforts pour favoriser l'apprentissage de la programmation. Cependant, une proportion importante d'étudiants ne réussit pas le cours. Ce constat nous suggère de repenser le modèle d'enseignement d'un cours de programmation en tenant compte prioritairement de l'hétérogénéité de la clientèle et de la diversité des stratégies d'apprentissage. Pour ce faire, il nous faudra offrir de nouveaux moyens, de nouvelles situations d'apprentissage qui permettront de nouveaux rapports aux savoirs et savoir faire à acquérir. Il nous faudra également briser un paradoxe de taille : « un enseignement uniforme à une classe hétérogène ». Ainsi, d'un enseignement uniforme dont l'origine est le professeur, il nous faudra concevoir un enseignement non uniforme ayant comme acteur central l'étudiant. Auparavant, dans la

prochaine section nous examinerons l'apprentissage de la programmation par le billet de recherches en génie logiciel.

3.2 Considérations sur l'apprentissage de la programmation et le génie logiciel

La finalité de nos travaux est d'offrir à nos étudiants de nouveaux outils favorisant, ou au mieux, améliorant l'apprentissage de la programmation. Pour ce faire, nous explorerons différentes études traitant de cet apprentissage afin de connaître les mécanismes en cause de manière à mieux cibler les difficultés inhérentes. Comme vous le lirez, ces études nous ont permis d'apprécier la discipline du génie logiciel.

Le génie logiciel est la discipline informatique qui vise le développement de logiciels sans erreur en proposant des méthodes et des outils adéquats. La recherche de méthodes systématiques et reproductibles a amené les chercheurs à se questionner sur l'apprentissage de la programmation et le raisonnement du programmeur. Dans un premier temps, nous présenterons chronologiquement les principales études et soulignerons les sujets traités. Dans un deuxième temps, nous exposerons différentes stratégies de production de programme en empruntant l'angle de la résolution de problème.

3.2.1 Études en génie logiciel

Dans son volume, Génie logiciel et psychologie de la programmation (1998), Détienne distingue deux périodes de l'étude de la programmation. La première période, s'étalant du début des années 70 jusqu'au milieu des années 80, s'intéressait à tester de nouveaux outils de programmation. Le développement de méthodes, langages et outils de visualisation ont contribué à l'évolution de la programmation. Cette période, centrée principalement sur la validation et la mise au point de nouveaux outils, a été suivie d'une seconde période orientée principalement sur l'analyse d'activités liées au développement

informatique et à l'élaboration de modèles cognitifs. Les psychologues et ergonomes ont alors été sollicités dans ces études.

La validation d'outils s'est orientée vers des études bi-factorielles et des tests statistiques, de type analyse de variance (montrer quelle est la probabilité de ne pas trouver ces résultats lors de répliques sur des groupes ou échantillons similaires) ou recherche de corrélation entre deux variables propres aux sujets (e. g. années d'expérience versus facilité à comprendre un programme). Les chercheurs sont alors confrontés à plusieurs critiques étant donné la difficulté d'isoler proprement l'interaction de deux variables sur un spectre complexe de plusieurs variables. Le phénomène s'intensifie dans les années 1980 où plusieurs critiques sévères paraissent (Brooks, 1980; Curtis, 1984; Hoc 1982; Laughery, 1985; Moher & Schneider, 1982; Sheil 1981). Entre autre, Hoc souligne qu'il s'agit d'une analyse superficielle de la tâche aux dépends d'une analyse psychologique de l'activité de programmation. De plus, l'objectif de ces études semblait avoir comme finalité le développement des tests de compétence à des fins de sélection.

Comme nous l'avons souligné en introduction, la deuxième période vise l'élaboration de modèles cognitifs de la programmation en empruntant les cadres théoriques de la psychologie cognitive pour l'analyse de tâches réelles. Un objectif ergonomique est d'améliorer l'activité en adaptant l'outil informatique à son utilisateur (Détienne 1998).

Des équipes pluridisciplinaires (psychologues/ergonomes, informaticiens) travaillent à la modélisation des activités de l'homme faite en terme de représentations et de traitement de l'information. Comme le souligne Richard (1990, p.9) «*Ce qui caractérise les activités mentales c'est qu'elles construisent des représentations et opèrent sur ces représentations. Les représentations sont [...] essentiellement des interprétations qui consistent à utiliser des connaissances pour attribuer une signification d'ensemble aux éléments issus de l'analyse perceptive et ceci dans le contexte d'une situation et d'une tâche particulière.*»

Dans son volume, Françoise Détienne (1998) définit la représentation, le traitement et la connaissance de la façon suivante:

> *«Plusieurs auteurs soulignent la distinction entre les notions de représentation, traitement et connaissance (Richard, Bonnet et Ghiglione (1990)). La représentation est le contenu cognitif sur lequel s'exerce le traitement; une nouvelle représentation est le produit du traitement et peut ainsi devenir l'objet d'un nouveau traitement. Les représentations sont transitoires. En effet, les traitements sont liés à des tâches et une nouvelle tâche engendre de nouvelles représentations. Certaines représentations peuvent cependant s'intégrer à la mémoire à long terme sous la forme de connaissances et certains traitements peuvent être stockés sous la forme de procédures.»*

Détienne (1998) nous apprend que du point de vue de l'ergonomie cognitive, une distinction est faite entre les concepts de tâche et d'activité. L'activité vise la réalisation d'une tâche. Les tâches de l'informaticien peuvent être: la détection et correction d'erreurs, la modification de programme, la vérification de programme, le test de programme, la réutilisation de programme, la documentation de programme. L'activité que l'apprenant doit mettre en œuvre pour atteindre un objectif défini pour une tâche particulière est caractérisée par des actions observables et par des processus cognitifs sous-jacents.

Plusieurs articles provenant de la psychologie cognitive et de façon moindre de l'intelligence artificielle tentent de cerner et d'expliquer comment se réalise l'activité de programmation. Détienne (1998) identifie et répertorie les différentes approches utilisées dans ces articles de la façon suivante:

- la catégorisation: l'approche de Rosch et ses collègues (Rosch, Mervis, Gray, Johnson et Boyes-Braem,1976; Rosch, 1978) est appliquée à la catégorisation de problèmes de programmation (Adelson, 1981);
- la compréhension de texte en langage naturel: la théorie de Kintsch et van Dijk (1978) est appliquée à la compréhension de programme (Atwood et Ramsey, 1978);

- l'apprentissage: des modèles cognitifs et éducationnels sont empruntés par Mayer (1981) pour rendre compte de l'apprentissage de la programmation;
- la modélisation des connaissances: la théorie des schémas de Schank et Abelson (1977) est empruntée par Soloway et son équipe pour rendre compte de l'organisation des connaissances des experts en programmation (Soloway et Erhlich, 1984) et la théorie du traitement de l'information sur l'expertise dans des domaines complexes, comme les échecs (Chase et Simon, 1973), est appliquée au domaine de la programmation (McKeithen, Reitman, Reuter et Hirtle, 1981);
- la résolution de problème: la théorie du traitement de l'information de Newell et Simon (1972) est utilisée par Brooks (1977) pour rendre compte des mécanismes cognitifs mis en œuvre dans la conception de programme.

La méthodologie utilisée dans les études récentes est davantage de type clinique, avec des analyses fines de l'activité selon le paradigme de l'interprétation de protocoles verbaux. Les activités mentales peuvent être inférées à partir de comportements et de verbalisations et peuvent être simulées par des modèles de traitement de l'information. Cela a permis la construction de modèles descriptifs de l'activité. La thèse de Patrick D'Astous (1999) traite spécifiquement du sujet en proposant un modèle qui utilise la nomenclature du paradigme de développement orienté objet.

Détienne affirme que l'évolution thématique récente s'attaque au problème de la généralisation des connaissances obtenues par l'étude de diverses activités liées à la programmation. Dans cette optique, elle énumère les recherches suivantes:

- l'activité de programmeurs professionnels;
- les aspects collectifs/collaboratifs dans le développement de logiciel;
- les activités de spécification et de conception;
- l'effet des langages/paradigmes de programmation sur les activités de programmation.

La psychologie cognitive et le génie logiciel ont travaillé de pair pour comprendre et tenter d'expliquer différentes activités inhérentes au développement de logiciel. Dans la prochaine section nous nous concentrerons sur l'activité de conception de logiciel dans le but d'aborder la facette de l'apprentissage de la programmation.

3.2.2 Conception de logiciel

Trouver la solution d'un problème consiste à comprendre et résoudre ce problème. Pour notre cas, la solution consiste à produire un programme informatique qui répondra aux besoins de l'usager. Dans des situations de production de programme, le sujet est centré sur l'élaboration d'une solution correspondant ici à une procédure. Dans ce cas, il va construire une représentation de la structure de cette procédure et l'exprimer dans un langage de programmation. Pour arriver à cette fin, le processus de développement logiciel peut s'inscrire dans une méthode de résolution de problèmes pour faciliter la tâche de l'étudiant débutant en programmation (Deek, 1997).

Dans son article intitulé « The software process : a parallel approach through problem solving and program development », Fadi P. Deek (1999) présente deux tableaux comparant une méthode de résolution de problèmes avec le processus de développement logiciel. Nous reproduisons ici ces tableaux. Le tableau 3.4 concerne la méthode de Polya (1945) et le tableau 3.5 concerne la méthode de Rubinstein (1975). Dans les deux cas, chaque phase énoncée dans la méthode de résolution de problème correspond à une ou plusieurs phases du processus de développement logiciel.

Tableau 3.4 Processus de développement logiciel et la méthode de résolution de problèmes de Polya

Phases du processus du développement logiciel	**Phases de la méthode de résolution de problème de Polya**
Identification du problème Étude de faisabilité	Compréhension du problème
Analyse et requis Conception et spécifications	Élaborer un plan
Implantation Intégration	Réaliser le plan
Test Déploiement Maintenance Retrait	Réviser

Tableau 3.5 Processus de développement logiciel et la méthode de résolution de problèmes de Rubinstein

Phases du processus du développement logiciel	**Phases de la méthode de résolution de problèmes de Rubinstein**
Identification du problème Étude de faisabilité	Rédiger le problème sous une forme primitive
Analyse et requis	Transformer le problème en langage simple Exprimer sous forme mathématique, si possible
Conception et spécification	Représenter le problème en utilisant des diagrammes, des tableaux et des graphes
Implantation Intégration Test Déploiement Maintenance Retrait	

Deek (1999) prétend que l'enseignement d'une méthode de résolution de problème intégrée ou adaptée au processus de développement logiciel est requise pour s'attaquer aux difficultés rencontrées par l'étudiant en apprentissage de la programmation. Il affirme qu'il est essentiel de joindre les habiletés de résolution de problème et les tâches

liées à la programmation dans un processus cohérent qui servira de cadre de travail pour l'étudiant débutant. Ainsi, il proposa son propre processus basé sur une étude approfondie des méthodologies de résolution de problème (Deek, 1997). Les étapes de ce processus sont : la formulation du problème, la planification de la solution, la conception de la solution, la transposition de la solution, les tests de la solution et la livraison de la solution. Voici une brève description de ces étapes.

Formulation du problème	Décrire le problème et organiser une représentation de toutes ses informations pertinentes telles les buts, les précisions, les inconnus, les conditions, les contraintes. La compréhension et l'interprétation du problème requièrent la rédaction d'une description précise obtenue par un raffinement progressif et l'élaboration des requis.
Planification de la solution	Identifier les solutions alternatives et préparer un plan soulignant la solution potentielle sans préciser ses étapes. La génération de la solution et sa planification requièrent l'accès à une connaissance pertinente et bien organisée, à une connaissance adéquate du domaine spécifique et à des stratégies ou heuristiques pour résoudre le problème.
Conception de la solution	Décrire la stratégie de solution planifiée à l'étape précédente. L'étudiant organise et raffine les composantes de la stratégie de solution. Il développe et rédige la solution sous forme algorithmique.
Transposition de la solution	Développer la solution en la transcrivant dans le langage de programmation cible. L'étudiant utilise un environnement de développement de programme qui générera l'exécutable.
Tests de la solution	Valider la solution et s'assurer que les requis sont satisfaits. L'étudiant doit savoir comment détecter et corriger les erreurs.
Livraison de la solution	Présenter les différentes parties de la solution en une forme organisée et compréhensible. Marque la fin du processus.

Pour un étudiant en génie, l'acquisition d'une méthode de résolution de problèmes est extrêmement bénéfique puisqu'elle peut lui venir en aide pour résoudre des problèmes dans différents domaines. Conscient de la proximité d'une méthode de résolution de problèmes avec le processus de développement logiciel et de son intérêt pour l'étudiant

en génie, nous enseignons dans notre propre cours l'apprentissage de la programmation en nous appuyant sur une méthode de résolution de problème (Boudreault et al, 1996 et 2001). Les différentes étapes de notre méthode sont : définition du problème, analyse du problème, conception des algorithmes, rédaction des programmes et mises au point des programmes. Voici une brève description de ces étapes.

Définition du problème	Qui? Que? Quoi? Identifier les principales composantes du problème : les entités en cause, les traitements à réaliser et les résultats attendus. L'étudiant doit réécrire en ses propres mots l'énoncé du problème en s'assurant de bien le comprendre et d'y incorporer tous les requis.
Analyse du problème	Comment? Identifier différentes solutions possibles pour réaliser chacun des traitements répertoriés à l'étape précédente. L'isolement de chacun des traitements correspond à une décomposition du problème. L'étudiant compare les solutions et tente d'identifier la meilleure.
Conception des algorithmes	La solution est décrite sous forme algorithmique pour chacun des traitements.
Rédaction des programmes	Le programme est obtenu en transposant la solution algorithmique dans le langage de programmation.
Mise au point des programmes	La validation du programme en s'assurant qu'il réponde adéquatement aux requis et qu'il ne contienne pas d'erreur.

L'étude de l'activité de conception de programme a été menée principalement dans le cadre des études sur l'activité de résolution de problèmes. Les difficultés de l'apprentissage de la programmation ont été étudiées par des éducateurs et des chercheurs (Mayer, 1981; Shackelford & Badre, 1993; Soloway et al. 1982). Plus spécifiquement, dans la conception de programme, les principales difficultés rencontrées par l'étudiant sont:

- une mauvaise interprétation de l'énoncé du problème[2] due principalement à un problème mal défini (Falzon, Bisseret, Bonnardel, Darses, Détienne & Visser, 1990; Newel & Simon, 1972; Visser & Hoc, 1990);
- existence de plusieurs solutions possibles ou acceptables pour le même problème;
- l'étudiant présente une déficience dans sa stratégie de résolution de problèmes et néglige sa planification (Anjaneyulu, 1994; Scholtz & Weidenbeck, 1992, 1993);
- le concepteur doit utiliser des connaissances dans au moins deux domaines (Brooks, 1977), le domaine de l'application (ou domaine du problème) et le domaine informatique dans lesquels il fait une mise en correspondance (Mayer, 1981; Perkins et al., 1986).

Détienne (1998) énumère trois approches théoriques utilisées dans les études sur l'activité de conception de logiciel: l'approche centrée sur les connaissances, l'approche centrée sur les stratégies et l'approche centrée sur l'organisation de l'activité.

3.2.2.1 Approche centrée sur les connaissances

En conception de programme, les auteurs s'accordent pour distinguer trois types de connaissances pour lesquelles il est possible de différencier les experts et les novices (Rist, 1986; Soloway, Erhlich, Bonar & Greenspans 1982):

- des connaissances syntaxiques définissant les éléments syntaxiques et lexicaux d'un langage de programmation;
- des connaissances sémantiques référant les concepts, e.g. la notion de variable ou de fonction;
- des connaissances schématiques, schémas de programmation, qui représentent des structures génériques de solutions.

[2] Il faut alors nous assurer que l'énoncé ne contient pas de spécifications imprécises ou manquantes;

La possession de ces trois types de connaissance permettrait de distinguer l'expert du novice.

La théorie des schémas cible l'organisation des connaissances en mémoire et les processus de mise en œuvre de ces connaissances. Un schéma correspond à une structure de données qui représente des concepts génériques stockés en mémoire. Ces concepts développés surtout en intelligence artificielle servent principalement à modéliser l'expert.
D'une certaine façon l'appropriation d'un schéma d'expert est l'objectif ultime d'un débutant. Les types de schémas connus de l'informaticien expert sont:

- schéma élémentaire de programmation représentant des connaissances sur la structure de contrôle et les variables (e.g. schéma de compteur avec boucle);
- schéma algorithmique ou schéma complexe de programmation (e.g. algorithme de tri, de recherche, etc.);
- schéma tactique (« tactical plan ») et stratégique (« strategic plan ») indépendant du langage de programmation (e.g. patrons de conception);
- et schéma d'implémentation qui est dépendant du langage (e.g. déclaration des classes abstraites dans la hiérarchie de classes en C++).

Il existe différents types de liens entre les schémas:

- liens de composition: un schéma de recherche d'information est composé de plusieurs schémas élémentaires;
- liens de spécialisation: un schéma de recherche linéaire est un type de schéma de recherche;
- et liens de mise en œuvre: par exemple un schéma de boucle de sommation spécifié par un « for » avec un critère d'arrêt.

La partie du schéma qui réalise directement le but du problème, la partie la plus importante, est appelée la ligne focale ou « focus » (Rist, 1986,1991). Dans le cas d'une boucle de comptage cette partie correspondrait à la mise à jour du compteur. Pour le débutant, cette partie peut être manquante et dévoiler une mauvaise compréhension de la structure ou considérée automatique au sens où c'est à l'ordinateur de le faire, c'est évident!

3.2.2.2 Approche centrée sur les stratégies

Détienne (1998) suppose que les experts choisiraient leurs stratégies de conception en fonction de facteurs comme la familiarité de la situation, les caractéristiques de la tâche, les caractéristiques notionnelles des langages. En revanche, elle ajoute que les novices éprouvent souvent des difficultés non seulement par manque de connaissances adéquates mais aussi par manque de stratégies adéquates pour répondre à une situation particulière: ils peuvent ainsi dans certains cas posséder les connaissances nécessaires mais ne pas être capables d'y accéder ou de les utiliser.

Les stratégies de conception peuvent être caractérisées selon plusieurs dimensions: la direction ascendante versus descendante du développement de la solution, le développement prospectif versus rétrospectif de la solution, le développement en largeur d'abord versus en profondeur d'abord de la solution, le guidage procédural versus déclaratif de la résolution. De nombreuses études ont porté sur les stratégies de conception de programme (Brooks, 1977; Chatel et Détienne, 1996; Détienne, 1995; Guindon, 1990; Rist, 1995; 1996; Visser, 1987; Visser et Hoc, 1990).

Pour bien comprendre ces différentes stratégies, il est nécessaire de rappeler qu'une étape préalable de décomposition du problème en sous problèmes doit être réalisé. Telle qu'illustrée à la figure 3.8, cette décomposition se présente habituellement sous forme d'une arborescence. Les stratégies consistent alors à préciser l'ordre dans lequel les lettres de l'arborescence, correspondant à une partie du problème, seront traitées.

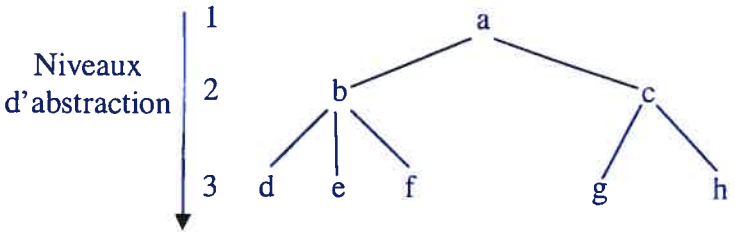


Figure 3.8 Arborescence de la décomposition du problème en sous problèmes

Les descriptions suivantes résument les propos de Détienne (1998 p. 46 à 48).

Direction ascendante versus descendante

En direction descendante, du haut vers le bas, le programmeur développe en partant à un niveau abstrait puis raffine graduellement en ajoutant de plus en plus de détail. Selon l'arbre de la figure 3.8, le développement s'exécute du niveau 1 vers le niveau 3. Cette méthode est privilégiée dans l'enseignement du paradigme procédural. En direction ascendante, la solution est d'abord développée à un niveau détaillé avant que sa structure abstraite soit identifiée. Malheureusement, à l'affût de résultats rapides, un novice aura tendance à coder directement au niveau du langage de programmation avant de construire la structure plus abstraite de la solution. Boudreault (1992) a montré que dans la réalisation d'un projet de programmation les étudiants codaient leur programme sans rédiger d'algorithme ni même effectuer d'analyse. La direction ascendante, du bas vers le haut, est celle empruntée par le paradigme par objets. L'identification des objets ainsi que leur comportement placent le développeur au bas de l'arborescence. L'assemblage des ces objets lui permettra d'avancer vers le haut de manière à obtenir un système fonctionnel.

Direction prospective versus rétrospective

La stratégie de conception est prospective (ou « forward ») quand la solution est développée dans le sens d'exécution de la procédure. La stratégie est rétrospective (ou « backward ») quand la solution est développée dans le sens inverse de l'exécution de la procédure. Une stratégie prospective reflète souvent chez les débutants une exécution mentale de la solution (Hoc, 1987). Dans ce cas, le développement de la solution s'appuie sur des connaissances du domaine du problème et non du domaine informatique. Le sens peut être rétrospectif quand aucun schéma de solution ou procédure connue n'est disponible (Rist, 1991). Sur la figure 3.8, une stratégie prospective développera le niveau 2 selon l'ordre b-c et le niveau 3 selon l'ordre d-e-f-g-h.

Développement en largeur d'abord versus en profondeur d'abord

Le développement de la solution est en largeur d'abord quand tous les éléments de solution sont développés à un même niveau d'abstraction avant d'être développés à un autre niveau d'abstraction (plus détaillé). Sur la figure 3.8, cela correspond à développer complètement le niveau 2 (b-c ou c-b) puis complètement le niveau 3 dans n'importe quel ordre. Le développement de la solution est en profondeur d'abord quand un élément de la solution est développé à tous les niveaux d'abstraction (il est raffiné) avant qu'un autre élément de la solution ne soit développé. Toujours sur l'arborescence de la figure 3.8, un développement en profondeur consiste à développer a-b-d, puis e, puis f, puis a-c-g, puis h.

La stratégie par raffinements successifs (« stepwise refinement ») (Wirth, 1974) se caractérise par sa direction descendante et le développement de la solution en largeur d'abord. Le problème est décomposé en sous problèmes, et à un niveau d'abstraction donné, la solution est explorée et développée complètement.

Guidage déclaratif versus procédural

Le guidage de la résolution est dit procédural quand c'est la structure de la procédure qui guide la résolution: la résolution de problème est alors basée sur les buts ou procédures. Le guidage de la résolution est dit déclaratif quand ce sont des propriétés statiques qui guident la résolution: la résolution de problème est alors basée sur des propriétés statiques des objets ou sur des rôles. Par exemple, le développement d'un jeu de billes selon le guidage procédural s'attaquera au déplacement des billes en recherchant la ou les équations satisfaisant le positionnement de chacune a un temps donné, tandis que le guidage déclaratif ciblera la description d'une bille (volume, masse, couleur, état: mouvement ou immobile, position), la description de l'espace de jeu (dimension de la table de jeu, présence et emplacement des bordures), etc.

Stratégie de la simulation mentale

Les activités de simulation mentale peuvent être mises en œuvre dans un objectif d'évaluation de la solution. (Visser et Hoc, 1990). La simulation mentale est également utilisée à des fins de compréhension du problème.

3.2.2.2.1 Facteurs de déclenchement des stratégies

Détienne (1998) énumère quatre facteurs influençant le déclenchement de stratégies: la structure des langages de programmation, l'environnement de développement, le type de problème et l'expertise du sujet dans le domaine informatique.

La structure des langages de programmation peut être un facteur influençant la stratégie. Le programmeur en langage C utilisera une stratégie plutôt descendante par raffinements successifs puisqu'il s'agit d'un langage procédural. Le programmeur en SQL (pour les

bases de données) utilisera une stratégie basée sur le guidage déclaratif étant donné les tables et leurs liens.

L'environnement de développement peut également être un facteur. L'environnement Jgrasp développé à l'université Auburn de l'Alabama offre une interface dirigeant le développeur vers la stratégie descendante. L'outil Schémacode dont le prototype a été développé à l'École Polytechnique de Montréal oriente le développeur spécifiquement vers le raffinement graduel. Les environnements spécialisés pour le développement orienté objet exige la déclaration de classe imposant ainsi une stratégie de guidage déclaratif.

Le type de problème est un autre facteur influençant le choix de la stratégie. La stratégie utilisée pour une application dédiée à une base de données sera différente de la stratégie utilisée pour une application vouée au temps réel. Dans le premier cas, la modélisation des données est prioritaire, dans le deuxième cas, la procédure est prioritaire.

L'expertise du sujet dans le domaine informatique et la disponibilité de schémas de programmation construits en mémoire peuvent influencer le choix de stratégie (Rist, 1991). L'expert peut suivre une stratégie descendante et prospective pour un problème qui lui est familier et pour lequel il dispose déjà d'un schéma alors que le novice suivra plutôt une stratégie ascendante et rétrospective

3.2.2.3 Approche centrée sur l'organisation de l'activité

Deux types de modèles s'opposent pour décrire comment s'organise l'activité de programmation: le modèle hiérarchique basé sur des modèles normatifs inspirés de méthode de programmation et les modèles opportunistes basés sur les résultats d'études empiriques qui mettent l'accent sur les déviations de l'activité réelle par rapport à un modèle strictement hiérarchique.

Le modèle hiérarchique a été fortement inspiré par la programmation structurée. Selon ce modèle (Adelson et Soloway, 1985), le processus de décomposition du problème en une solution est essentiellement descendant et une recherche de solution en largeur d'abord (stratégie par raffinements successifs ou « balanced development ») est privilégiée.

Les modèles opportunistes rendent compte des déviations observées par rapport à un plan hiérarchique. Des études (Guindon, Krasner et Curtis, 1987; Guindon, 1990; Visser, 1987) mettent l'accent sur les déviations de l'activité des informaticiens par rapport à un modèle hiérarchique. Elles déduisent que l'activité réelle est organisée de façon opportuniste: par exemple, la recherche de solution peut se faire aussi bien de façon descendante qu'ascendante et la solution est recherchée en privilégiant aussi bien la largeur que la profondeur selon la situation.

Alors que les approches présentées mettent en avant des modèles de résolution de problème certains auteurs ont fait le parallèle avec la production de textes en langage naturel. Un des modèles cognitifs de la production de texte est celui de Hayes et Flower (1980). Ces auteurs ont identifié trois phases:

- la planification de la structure du texte: l'organisation des idées est fonction des connaissances dans le domaine et l'établissement de buts dépend des objectifs de communication;
- la traduction ou mise en texte du plan du texte en une représentation linguistique
- la révision du texte qui est fonction de l'évaluation du scripteur.

La particularité de ce modèle est que le processus est cyclique plutôt que strictement linéaire.

Si l'on transpose à l'activité de programmation, on retrouve alors:

- la planification qui nécessite de récupérer des connaissances pertinentes pour la résolution du problème et de construire une solution abstraite;
- le processus de production est ici l'implémentation d'une solution dans un langage de programmation;
- le processus de révision peut impliquer de réviser l'implémentation, la solution abstraite ou la représentation du problème.

3.2.3 Quelques comparaisons entre l'expert et le novice

L'un est sans expérience et l'autre est expérimenté, le novice et l'expert sont deux personnes différentes et pour s'adresser adéquatement à l'un comme à l'autre il faut connaître ces différences. Plusieurs études comparatives nous informent de leur profil respectif.

Dans cette veine, des études (Chase et Simon, 1973; Chi, Feltovitch et Glaser, 1981) montrent que les experts forment des représentations abstraites, conceptuelles des problèmes alors que les novices représentent les problèmes selon des traits de surface.

Deux paradigmes expérimentaux ont permis de comparer l'organisation des connaissances des experts et des novices dans le domaine de la programmation: le paradigme compréhension/rappel et le paradigme de catégorisation.

Une étude (Shneiderman, 1980) a utilisé le paradigme de compréhension/rappel. Une tâche de rappel était consécutive à la présentation de programmes dont les lignes de code étaient soit dans l'ordre, soit dans le désordre. Le rappel des experts est alors supérieur à celui des novices dans la première condition mais pas dans la seconde. Dans la seconde condition, puisque l'expert ne peut identifier de structure significative, nous pensons qu'il n'a aucun outil cognitif pour organiser sa mémoire et de ce fait, ne possède aucun avantage par rapport à un programmeur novice.

Le paradigme de catégorisation consiste à présenter à des programmeurs un certain nombre de programmes ou bouts de programmes en leur demandant de les catégoriser et d'expliciter leurs critères de catégorisation. Adelson (1981;1985) montre que les catégories proposées par les experts sont différentes de celles des novices. Celles des novices dépendent de traits de surface, e.g. la structure syntaxique, alors que les catégories formées par les experts révèlent des structures plus profondes telles que la similarité fonctionnelle ou procédurale.

Un autre critère pouvant permettre de distinguer l'expert et le novice est la stratégie de développement utilisée que nous avons présenté à la section 3.2.2.2. Nous y avons souligné, en citant Détienne, que les experts choisiraient leurs stratégies de conception en fonction de différents facteurs tandis que les novices éprouvent souvent des difficultés par manque de connaissances adéquates ou par manque de stratégies adéquates pour répondre à une situation particulière. Les novices peuvent dans certains cas posséder les connaissances nécessaires mais ne pas être capables d'y accéder ou de les utiliser. Dans cette même section, Hoc (1987), nous apprend que le débutant peut s'engager dans une stratégie prospective par une exécution mentale de la solution en s'appuyant sur des connaissances du domaine du problème et non du domaine informatique.

3.2.4 Le développement orienté objet

Nous nous permettons un éloignement de notre propos initial pour discuter d'une question portant sur le choix du premier langage à apprendre. Comme nous l'avons vu auparavant, ce choix est capital puisque le langage de programmation est un facteur influençant la stratégie de développement. Cette fameuse question est:

> Quel langage de programmation est-il préférable d'enseigner en premier, un langage procédural tel le langage C ou un langage orienté objet tel le langage C++?

Au premier abord, les avis sont partagés. Nous savons que certaines institutions offrent un cours présentant initialement un langage orienté objet et que d'autres institutions offrent un cours présentant un langage procédural. Plusieurs paramètres, autres que pédagogique, sont pris en compte dans le choix du langage, par exemple, la popularité, les besoins de l'industrie, l'évolution du domaine, l'utilisation dans les autres cours, etc. Cependant, des constats intéressants nous parviennent d'études de la conception et la réutilisation de programme orienté-objet.

Les méthodes de conception par objet conduisent vers une approche hiérarchique spécifique, préconisant en premier lieu l'identification des classes et leurs relations (aspects déclaratifs) puis le codage des méthodes (aspects procéduraux). Des études empiriques (Lee & Pennington, 1994; Pennington, Lee & Rehder 1995; Détienne, 1990a et d, 1993 et 1995) montrent que les novices ont plus de facilité à suivre rigoureusement cette méthode mais révèlent une grande difficulté dans l'identification des classes et l'adéquation du code des méthodes. Les études montrent que les aspects déclaratifs et procéduraux de la solution ne sont pas intégrés même très tard dans la phase de conception. Il y a un processus de création et d'abandon de classes très intense.

Détienne (1995) observe que les novices décrivent les objets et les actions ou procédures séparément dans leur première version de solution. Plutôt que de décomposer la procédure, les novices l'associent parfois comme un tout à une seule classe. Ils révèlent ainsi un problème de décomposition et d'association d'aspects déclaratifs et procéduraux. Les novices raffinent certains aspects des classes à un niveau plus abstrait lors du codage en ajoutant des éléments. Les novices ont besoin de construire une représentation des aspects procéduraux de la solution afin de raffiner, évaluer et réviser leur décomposition en classes (avec actions associées) (Détienne 1998).

La littérature sur les novices en conception par objet suggère que la construction d'une représentation de certains aspects procéduraux de la solution précède la construction d'aspects plus déclaratifs de la solution. Cela va à l'encontre du caractère naturel de

l'approche objet, par contre elle va dans le sens des hypothèses faites en ergonomie cognitive qui nous disent que les connaissances sont organisées à partir des buts et procédure et non à partir des connaissances relationnelles sur les objets. De plus, les procédures sont des propriétés des objets et elles constituent les bases de la catégorisation des objets (Richard, 1996).

À la section 3.2.2.1 nous avons énoncé les trois types de connaissances permettant de différencier les experts des novices: les connaissances syntaxiques, les connaissances sémantiques et les connaissances schématiques. L'introduction de la programmation orientée objet se traduit par un plus grand volume de connaissances à acquérir puisque la matière propre au développement procédural est préalable à l'implantation de la méthode de l'objet correspondant à une procédure ou une fonction. Cela nécessite par le fait même une maîtrise en accéléré vu l'intégration du procédural dans l'objet. En revanche, la présentation d'un langage procédural initialement, voire dans un premier cours, permettra à l'étudiant d'explorer et d'acquérir les concepts de base de la programmation tels les structures décisionnelles, les structures répétitives, les fonctions et les différentes modes de transmission de paramètres, etc. Une fois bien maîtrisés, ces structures et concepts seront des outils à incorporer à l'objet qui devrait être introduit dans un second cours.

Parfois en parallèle mais de plus en plus en concertation, le génie logiciel, la psychologie cognitive et l'intelligence artificielle contribuent à l'avancement des recherches sur l'apprentissage de la programmation.

Sans être des certitudes, certains modèles et théories sont incontournables. Les enseignements les plus utiles se retrouvent dans les trois approches théoriques utilisées dans les études sur l'activité de conception de logiciel: l'approche centrée sur les connaissances, l'approche centrée sur les stratégies, et l'approche centrée sur l'organisation de l'activité. Nous retiendrons également la méthodologie amplement utilisée qui est celle de comparer la démarche du novice, telle que nous l'avons analysé à

celle de l'expert, puisque celle-ci nous oriente vers la finalité de l'apprentissage du novice qui est la maîtrise de la programmation.

Un modèle d'apprentissage qui met en valeur les processus d'acquisition de schémas de connaissance, comme dans l'approche développée par Soloway a donné lieu au développement d'outils tutoriels basés sur la notion de schémas de programmation. L'hypothèse sous-jacente est qu'enseigner consiste à fournir explicitement des schémas aux débutants, ce qui faciliterait le développement de leur expertise (Anderson, Boyle, Farrell et Reisner, 1987; Bonar et Cunningham, 1988). Nous croyons qu'une façon qui permettrait d'acquérir ces schémas est le recours à l'analogie. En effet, l'appariement structurel permettant d'identifier un sous-ensemble d'attributs et de comportements communs à l'analogue et à la notion à apprendre peut se comparer à l'extraction d'un schéma. Pour cette raison, nous étudierons au chapitre suivant le concept d'analogie et de raisonnement analogique.

De façon complémentaire, selon un modèle d'acquisition en termes de restructuration interne de schémas, on peut penser que la pratique de la programmation structurée en incluant la formation à la conception, faciliterait la construction de l'expertise (Davies, 1994): elle focaliserait les débutants sur les parties importantes des schémas (focus) et faciliterait ainsi la restructuration de schémas.

Des outils tutoriels ont été développés, ils utilisent des techniques en intelligence artificielle qui permettent de comparer les plans développés par les étudiants avec une base de connaissance de plans corrects dans le langage cible et de proposer des plans alternatifs. Cette stratégie automatisée d'apprentissage de la programmation devrait nous donner un outil complémentaire à la formation de nos étudiants.

3.3 Le recours à l'analogie

> *... elle « mesure » la distance de l'être à l'apparaître, du modèle à son image, de l'invisible au visible, de l'archétype intelligible à sa présence sensible, et cette « mesure », c'est précisément l'analogie qui, dit Platon dans le Timée, est « le plus beau des liens ».*

En enseignement, l'analogie s'avère régulièrement une constituante indispensable à la confection d'une explication. Elle offre un point d'ancrage à la nouvelle notion à apprendre étant composée d'éléments concrets et bien connus. Dans le domaine de l'informatique comme dans le domaine des sciences en général, l'analogie est une voie régulièrement empruntée par l'enseignant pour la présentation de notions de plus en plus nombreuses et de plus en plus complexes.

Dans le présent chapitre, nous désirons démontrer les bienfaits du recours à l'analogie dans l'enseignement. Une fois le terme analogie défini, nous exposerons les mécanismes en jeu dans la compréhension d'une analogie, principalement l'appariement structurel et la transposition. Nous poursuivrons par la présentation de quelques exemples d'utilisations variées de l'analogie dans le domaine de l'informatique. Finalement, nous terminerons par la présentation des utilisations envisagées de l'analogie dans le cadre de nos travaux.

3.3.1 L'analogie

Le Petit Larousse (1999) définit l'analogie comme le rapport de ressemblance que présentent deux ou plusieurs choses ou personnes. Le degré associé à ce rapport de ressemblance et sa détermination justifient, selon certains auteurs, une terminologie distinctive telle l'image, la métaphore, la similarité, etc. En outre, le degré de ressemblance est spécifiquement nommé la mesure de similarité dans le domaine de la psychométrie.

Keane et Costello (2001) croient que le rapport de ressemblance de l'analogie, identifié selon un mécanisme d'appariement structurel, s'étend à des phénomènes tels que la métaphore, la similarité et la combinaison conceptuelle[1]. La figure 3.9 montre l'importance du rôle joué par le mécanisme analogique de l'appariement structurel permettant d'expliquer la métaphore, la similarité et la combinaison de concepts. Inspiré de ces deux chercheurs, nous utiliserons le terme analogie pour identifier tout phénomène s'expliquant à l'aide du mécanisme d'appariement structurel.

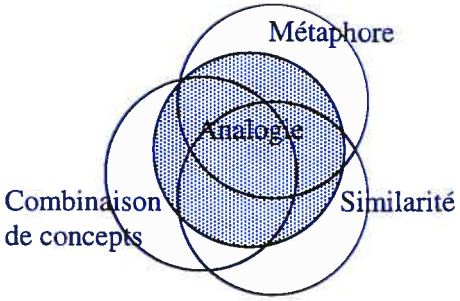


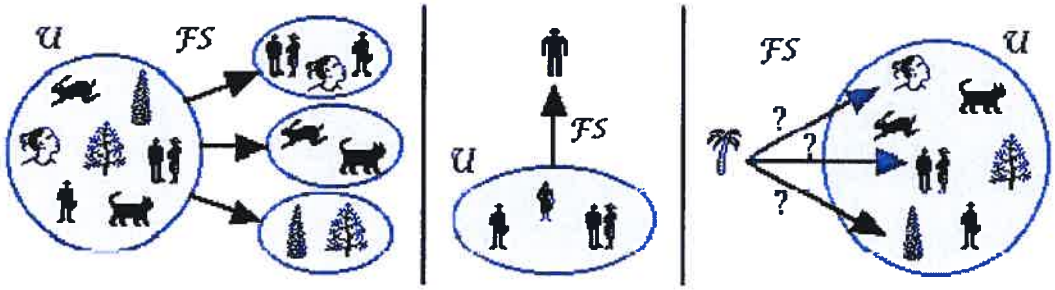
Figure 3.9 Enchevêtrement de trois phénomènes expliqué par un mécanisme analogique unifiant tel l'appariement structurel (tirée de Kean et Costello p.288)

Pour construire une analogie, les seules prémisses requises sont deux domaines nettement distincts et un prédicat bien défini. De plus, l'analogie peut être utilisée pour transférer des propriétés ou pour transférer des relations. L'analogie de la vidéo-cassette et du fichier texte remplit ces deux prémisses. Il s'agit de deux domaines bien distincts permettant d'expliquer le prédicat de la non simultanéité de la lecture et de l'écriture. En effet, il est impossible d'effectuer un enregistrement (écrire) sur une vidéo-cassette qu'on utilise présentement pour visionner (lire) son contenu. Il en est de même pour un fichier puisqu'il doit être initialement ouvert soit en écriture, soit en lecture. Impossible de lire et écrire en même temps. Ce prédicat d'exclusivité constitue un transfert de propriété, tandis que la séquence ouverture (du fichier vs insérer la cassette dans le magnétoscope), lecture (du fichier vs de la vidéo-cassette) et fermeture (du fichier vs arrêt de lecture de la vidéo-cassette) constitue un transfert de relation. La section 3.3.2 théorise les processus en cause lors de l'utilisation de l'analogie.

[1] La combinaison conceptuelle correspond à lier deux termes ou plus de nature différente, par exemple, « le poumon de la terre, le poteau de vieillesse ».

3.3.1.1 Rôle conceptuel

Comme nous l'avons écrit précédemment, l'analogie se définit à partir d'une certaine ressemblance qui peut être quantifiée par une fonction de similarité. Cette dernière fonction peut résulter en trois types de traitement : la classification, la caractérisation et l'identification. Le processus de classification vise à structurer les données, en fonction de leurs ressemblances, sous la forme d'un ensemble de classes à la fois homogènes et contrastées. On peut parler également de catégorisation puisqu'il s'agit d'un processus central dans la cognition humaine. Le processus de caractérisation consiste à construire une représentation explicite des informations qui sont communes à un ensemble de données. Il s'agit en fait d'un processus de généralisation. Finalement, le processus d'identification recherche la classe à laquelle un objet inconnu est susceptible d'appartenir ou détermine quel(s) objet(s) est le plus ressemblant. La figure 3.10 tirée de Bisson (2000) illustre les trois types de traitement réalisés par la fonction de similarité, où *U* représente l'ensemble des objets et *FS* la fonction de similarité.



Classification Caractérisation Identification

Figure 3.10 Trois types de traitement de la fonction de similarité (tirée de Bisson 2000)

Certains chercheurs proposent d'évaluer la qualité d'une similarité en proposant des méthodes astucieuses de calcul. Par exemple, Tversky (1977) suggère de calculer la similarité S(x,y) entre deux domaines x et y décrits respectivement par les ensembles d'attributs A et B, selon la formule :

$$S(x,y) = \frac{f(A \cap B)}{f(A \cup B) + \alpha \cdot f(A - B) + \beta \cdot f(B - A)} \qquad avec\ \alpha, \beta \geq 0$$

Les opérateurs ensemblistes d'intersection, d'union et de différence jouent un rôle primordial en réalisant de différentes façons la comparaison des attributs des deux domaines. Au numérateur, on retrouve l'opérateur d'intersection qui identifie les attributs communs aux deux domaines. Au dénominateur la formule tient compte de l'ensemble des attributs sans dédoublement et des différences de l'un par rapport à l'autre et de l'autre par rapport à l'un. Les paramètres *f,* α, et β permettent de considérer différents modèles cognitifs et mathématiques de la similarité.

La détermination de la similarité est sans contredit un traitement majeur dans la compréhension d'une analogie. La section suivante propose une théorie tentant d'expliquer comment s'effectue l'appariement des éléments semblables.

3.3.2 La théorie de la mise en correspondance de structure de Gentner

Dedre Gentner (1983, 1989) propose une théorie tentant d'expliquer le rôle de l'analogie dans l'apprentissage d'une nouvelle notion. Il s'agit de la théorie de la mise en correspondance de structure. Grosso modo, il s'agit d'identifier une structure commune aux deux domaines afin de réaliser un appariement structurel et de le transposer du domaine connu, celui de l'analogue, vers le domaine de la notion à apprendre. Habituellement, le domaine connu est appelé « la source » et le domaine de la notion à apprendre est appelé « la cible ».

Pour effectuer une analogie, Gentner propose quatre principes d'importance croissante, qui doivent guider la mise en correspondance des objets et des relations.

- Mise en correspondance des objets de la source et de la cible
- Pas de prise en compte des objets isolés décrits par quelques attributs

- On essaye d'apparier les objets liés par des relations communes
- Préservation des systèmes de relations connexes (principe de systématicité)

Plus formellement, le transfert de propriétés attribue à un terme x de X un prédicat bien établi pour un terme y de Y:

Y != X,	(X et Y deux domaines distincts)
P(y)	(Le prédicat bien établi de l'analogue)
d'où, P(x)	(Le prédicat à assimiler de X)

et le transfert de relations impose dans le domaine X un réseau de places et de dépendances hiérarchiques bien établi dans le domaine Y:

Y != X,	(X et Y deux domaines distincts)
$R(y_1,y_2,\ldots, y_n)$	(Le réseau de dépendances bien établi de l'analogue)
$R(x_1,x_2,\ldots, x_n)$	(Le réseau à assimiler de X)

En d'autres mots, Gentner (1989) définit une représentation des différents objets présents dans l'analogie par une série d'attributs et de relations. Les attributs désignent des caractéristiques physiques comme la pression de l'eau. Les relations désignent des propriétés reliant plusieurs attributs. Par exemple, une relation de grandeur entre la pression observée sur deux objets différents.

Gentner (1989) décrit dans son ouvrage les concepts à l'aide d'un langage structuré dans lequel l'importance sémantique des informations (leur *ordre*) est indiquée par la syntaxe. Sur cette base, Gentner propose une classification des méthodes d'appariement en fonction du type d'information mise en œuvre. Le premier niveau est celui des *similarités apparentes* dans lequel on ne prend en compte que les attributs directement vérifiés par les individus. Ensuite, on trouve le niveau des *similarités littérales* dans lequel on tient compte de l'ensemble des attributs et des relations, mais sans privilégier les unes ou les autres. Dans les expériences effectuées par Gentner, c'est très souvent la similarité littérale qui a le comportement le plus proche de celui des sujets humains. Au

niveau des *analogies* on favorise l'appariement des relations qui sont d'ordre le plus élevé; de ce fait, les appariements entre les objets de la source et de la cible sont déterminées par le rôle que ces objets jouent dans la structure relationnelle plus que par leurs similarités intrinsèques. Finalement, le dernier niveau est celui des *abstractions* qui est également basé sur l'appariement des relations d'ordre supérieur et dans lequel les propriétés des objets ne sont quasiment plus prises en compte.

Tableau 3.6 Les différents types d'appariements

	Attributs	**Relations**	**Exemple**
Similarités apparentes	Beaucoup	peu	Une luciole ressemble à une lampe
Similarités littérales	Beaucoup	Beaucoup	Le lait ressemble à de l'eau
Analogie	Peu	Beaucoup	L'atome et le système solaire
Abstraction	Très peu	Beaucoup	La chaleur s'écoule par diffusion

Gentner utilise l'analogie de Rutherford: « The atom is like the solar system » présentée à la figure 3.11 pour illustrer la transposition partielle de la représentation des connaissances du système solaire sur la structure de l'atome d'hydrogène.

Lors de la transposition, l'analogie sélectionne certains aspects de la connaissance acquise qui peut être structurellement caractérisée. L'apprenant tire des affirmations analogiques en appliquant des attributs valables dans le domaine du système solaire (domaine connu) vers le domaine d'apprentissage en substituant les nœuds du système solaire vers les nœuds de la structure de l'atome. Ainsi, les nœuds objets du domaine de la source, le soleil et les planètes du système solaire, correspondent aux nœuds objets du domaine cible, le noyau et les électrons. Étant donné cette correspondance, l'analogie transfert uniquement les relations satisfaites à la fois dans le système solaire et dans l'atome. Par exemple, il existe une force d'attraction des objets périphériques vers le centre; les objets périphériques tournent autour de l'objet central; l'objet central est plus massif que les objets périphériques; etc.

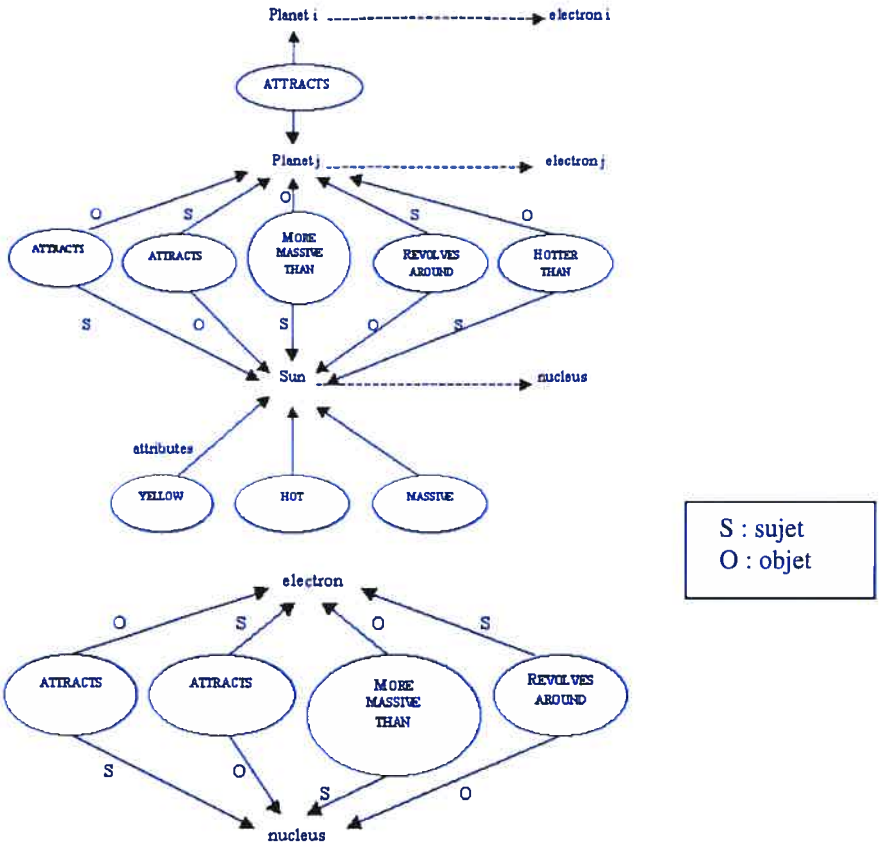


Figure 3.11 Transposition de la structure de l'analogie de Rutheford: « The atom is like the solar system ». D'après Gentner, 1983.

Nous avons précisé auparavant que l'analogie doit exploiter deux domaines distincts. Le transfert réalisé par cette analogie est dépendant de la relation entre ces domaines. Ainsi, pour des domaines semblables, le transfert sera identique; pour des domaines dissemblables ou inégaux, le transfert sera proportionnel; pour des domaines opposés, le transfert sera inverse. Un bémol à ces propos provient d'un article intitulé « analogie et logique naturelle » de Jean Charconnet-Méliès (1998) qui conteste la théorie de la projection de structure de Gentner. Il affirme que le domaine le moins connu structure le domaine le mieux connu. En d'autres mots, le transfert s'effectue dans le sens inverse. Il nomme ce phénomène, séquence analogie contre-projective. Sans rejeter complètement cette idée, nous pensons plutôt que l'apprenant construit la nouvelle connaissance en la validant dans le domaine bien connu lors de la sélection de la sous-structure transposée. Affirmer que l'apprenant construit son domaine connu à partir du domaine à acquérir représente un pas que nous ne franchirons pas.

Keane et Costello (2001) affirme que l'analogie, ou plus précisément l'appariement structurel, est actuellement perçu comme un mécanisme fondamental au cœur de différents processus cognitifs. L'appariement structurel est un processus qui associe la structure relationnelle de deux domaines de connaissance guidé par le principe de systématicité (Genter 1983). Ces trois composantes principales sont :

- *la correspondance bijective* (one-to-one correspondance), laquelle propose que le processus de comparaison maintienne un isomorphisme entre les éléments des deux domaines ;
- *la connectivité parallèle,* où l'appariement de deux énoncés impose la mise en correspondance de leurs arguments;
- *la systématicité,* qui privilégie le système d'appariements partagés (relations de causalités) plutôt qu'un autre système ayant le même nombre d'appariements mais cette fois fragmentés ou isolés.

3.3.3 Trois modèles de processus réalisant l'appariement structurel

Selon le modèle du processus de transposition, l'analogie s'explique par une transposition structurelle de la source vers la cible débutant par un processus d'appariement structurel symétrique où les représentations de la source et de la cible sont placées en correspondance afin de trouver l'appariement structurel le plus large et le plus profond possible. Tout naturellement, on imagine cette transposition de la source vers la cible. Gentner, Bowdle, Wolf et Boronat (2001) proposent trois modèles tentant d'expliquer ce dernier processus.

Le premier modèle dont le processus débute temporairement avec la source est explicitement ou implicitement supporté par le modèle de projection de schèmas[2]. Ce processus est également inhérent au modèle d'inclusion de classe où les analogies sont

[2] Le modèle NLAG de Greiner (1988) et le modèle IAM de Keane et Brayshaw (1988) supportent cette projection de schèmes.

interprétées en trouvant ou en créant la catégorie pour laquelle la source est le prototype qui est appliqué à la cible. Ce modèle privilégie les similarités d'ordre élevé puisque ces éléments seront les premiers prélevés de la source étant donné leur évidence. La figure 3.12 montre ce processus débutant à la source où l'information est extraite et projetée ensuite dans la cible.

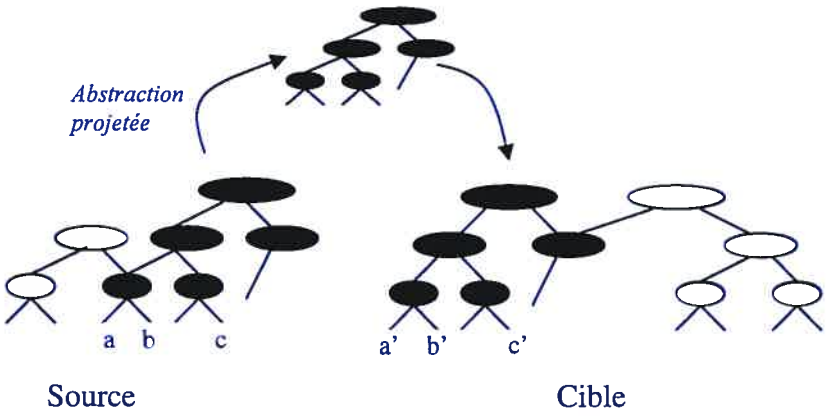


Figure 3.12 Modèle 1: projection initiale avec asymétrie temporelle

Illustrons ce modèle à l'aide de l'analogie, citée précédemment, à savoir celle de la vidéocassette et du fichier texte permettant d'expliquer la règle de non simultanéité des opérations de lecture et d'écriture. Pour ce premier modèle, le processus débute par l'identification de la règle sur le magnétoscope: une fois le magnétoscope en mode de visionnement, il est impossible de s'en servir pour enregistrer. Cette règle est ensuite projetée sur la cible: un fichier ouvert en mode lecture ne permet pas l'écriture des données.

Le deuxième modèle, attribué à Glucksberg, McGlone et Manfredi (1997), est plus élaboré. Ces chercheurs considèrent que les analogies sont des énoncés implicites d'inclusion de classes. Le modèle propose un processus asymétrique entre les termes de l'analogie plutôt qu'une asymétrie temporelle. Dans leur modèle d'attribution de catégorie, le processus débute simultanément sur les deux termes mais il se différencie dès le départ selon leur rôle spécifique. Le terme de la source fournit une catégorie abstraite qui peut être utilisée pour caractériser la cible et la cible fournit les dimensions selon lesquels il peut être caractérisé. Ensuite, l'abstraction de la source rencontre les dimensions dérivées de la cible pour former une interprétation. La figure 3.13 illustre ce

modèle qui se différencie du premier par un processus débutant simultanément dans les deux domaines.

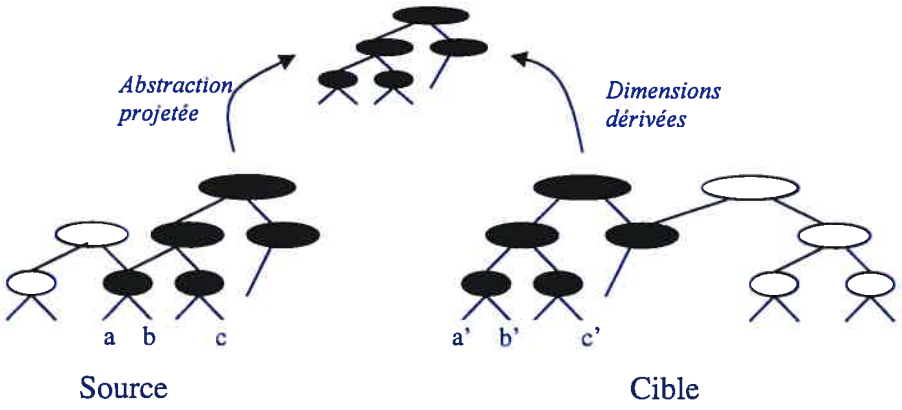


Figure 3.13 Modèle 2: abstraction initiale avec traitement asymétrique

Reprenons notre analogie, vidéocassette et fichier texte, dans le cadre de ce deuxième modèle. Le processus peut débuter ici par l'identification de la cassette dans la source et du fichier dans la cible. L'opération de lecture dans la cible peut ensuite être associée à l'opération d'enregistrement dans la source ou vice-versa. L'interdiction de lire et d'écrire à la fois dans un fichier est vérifiée par l'impossibilité de visionner et d'enregistrer à la fois une cassette ou vice et versa.

Le troisième modèle débute par un processus d'appariement symétrique. Une fois les représentations alignées, il est possible d'extraire une structure abstraite commune pour une projection par inférence de la source vers la cible. Ce modèle, illustré à la figure 3.14, reflète fidèlement la théorie de la mise en correspondance de structure de Gentner.

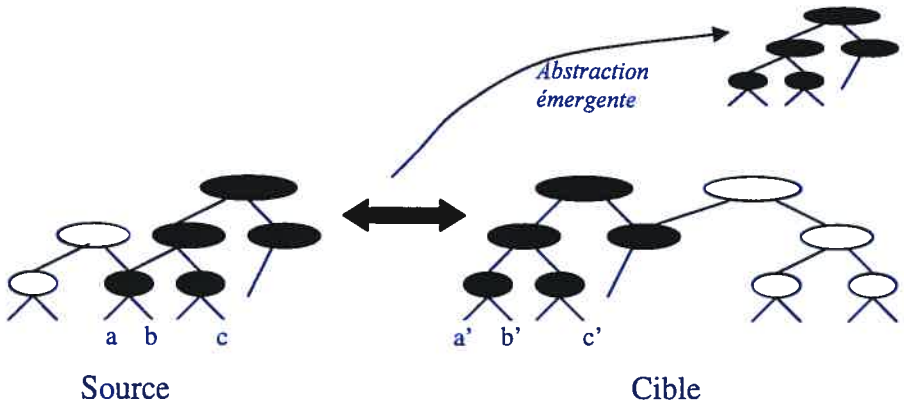


Figure 3.14 Modèle 3 : Appariement initial.

Pour ce dernier modèle, nul besoin d'utiliser notre analogie magnétoscope et fichier texte, il suffit de se référer à la figure 3.11 qui présente l'exemple de Rutheford.

Certains concepts de ces modèles sont implantés dans des environnements informatisés. Par exemple, la théorie MAC/FAC ("many are called but few are chosen") de Forbus, Gentner et Law (1995) basée sur la similarité a été traduite sous forme d'algorithmes. L'étape MAC utilise un appareilleur simple non structurel pour filtrer et obtenir les candidats les plus prometteurs d'une immense mémoire de descriptions structurées. L'étape FAC évalue minutieusement les candidats sélectionnés en utilisant un module nommé SME (Structure-mapping Engine). Ce module recherche un appariement structurel révélant les correspondances et les inférences qui précisent comment l'information rappelée peut être appropriée à la situation actuelle.

Voici quelques résultats obtenus par des recherches utilisant le modèle MAC/FAC et le module SME.

- La systématicité et la cohérence structurelle influence l'interprétation de l'analogie (Clement et Gentner 1991).
- La cohérence structurelle influence l'inférence en raisonnement analogique (Clement et Gentner 1991; Keane 1996; Spellman and Holyoak 1992, 1996; Markman 1997) et l'induction basée sur la catégorie (Lassaline 1996; Wu et Gentner 1998).

- La systématicité influence l'inférence dans le raisonnement analogique et l'induction basée sur la catégorie (Clement et Gentner 1991; Wu et Gentner 1998).
- La récupération basée sur la similarité est réalisée en surface, tandis que le raisonnement basé sur la similarité est réalisé selon la structure (Gentner, Rattermann et Forbus 1993; voir aussi Holyoak et Koh 1987; Ross 1989).

Un certain nombre de prédictions issues de SME et ayant trait a la psychologie ont été confirmées par expérimentation sur des humains.

- Le traitement en ligne de similarité et d'analogie est influencé par la richesse de l'objet et la profondeur de la relation (Gentner et Ratermann 1991; Markman et Gentner 1993; Rattermann et Gentner 1998).
- Plus tard dans le développement du modèle, l'appariement d'objet l'emporte sur l'appariement relationnel dû à une connaissance relationnelle inadéquate (Gentner et Toupin 1986; Gentner 1998; Rattermann et Gentner 1998; Gentner et Ratermann 1991).

Intuitivement on peut imaginer un modèle mental qui recherche une solution selon la simulation mentale plutôt que le raisonnement logique. Plus spécifiquement cette simulation est dite qualitative (de Kleer et Brown 1984; Forbus 1984; Bredeweg et Schut 1991; White et Frederiksen 1990). Les résultats de la simulation qualitative impliquent à un haut niveau, la description conceptuelle de valeurs physiques et leur changement, signe typique (e.g. un paramètre est croissant, décroissant ou constant) et la relation ordinale entre les paramètres (e.g. taux relatif du flux, température relative pour la transition de phase).

Forbus (2000) croit que trois facteurs favorisent le raisonnement basé sur la mémoire. Premièrement, la représentation qualitative réduit les différences. Deux personnes qui mémorisent et utilisent les représentations qualitatives de situations et de comportements, déduiront que deux situations se distinguant uniquement par des détails

quantitatifs sont identiques en regard de l'aspect qualitatif de leur comportement. Deuxièmement, le raisonnement analogique peut générer des prédictions pour de nouvelles situations. Prévoir les conséquences de la chute d'un vase s'appuie sur les expériences passées. Il est difficile dans un raisonnement de distinguer nettement ces différents processus de raisonnement par analogie. Cette distinction est utile au plan théorique. Dans la réalité, plusieurs analogies sont généralement utilisées pour assembler des modèles pour des systèmes complexes.

En acceptant qu'il utilise une approche constructiviste (Vygotsky, 1978; Piaget, 1970) , nous savons que l'étudiant doit intégrer la nouvelle connaissance à sa structure mentale existante. Cette intégration peut utiliser comme point d'ancrage la transposition d'une structure analogique telle que décrite par Gentner (1983). Lors de la transposition, l'étudiant est à la recherche des attributs et des relations satisfaites dans les deux domaines. Cette recherche s'inscrit dans le raisonnement de l'apprenant et s'accompagne d'une évaluation, voire d'une validation, de la nouvelle connaissance en comparaison avec la structure existante. La rétroaction de la comparaison lui permettra de réaliser des ajustements et devrait lui permettre une meilleure rétention. Une fois les ajustements réalisés, l'étudiant éprouve sa nouvelle conception pour vérifier son adéquation. Ce processus est itératif puisque l'étudiant peut ou plutôt doit éprouver sa nouvelle conception tant qu'il détecte des anomalies nécessitant des ajustements. La figure 3.15 résume ce processus d'apprentissage à l'aide de l'analogie.

Ainsi, le processus de raisonnement par analogie décrit précédemment est analogue à la démarche d'investigation scientifique où l'analogue a le rôle implicite de l'hypothèse ou du modèle à valider.

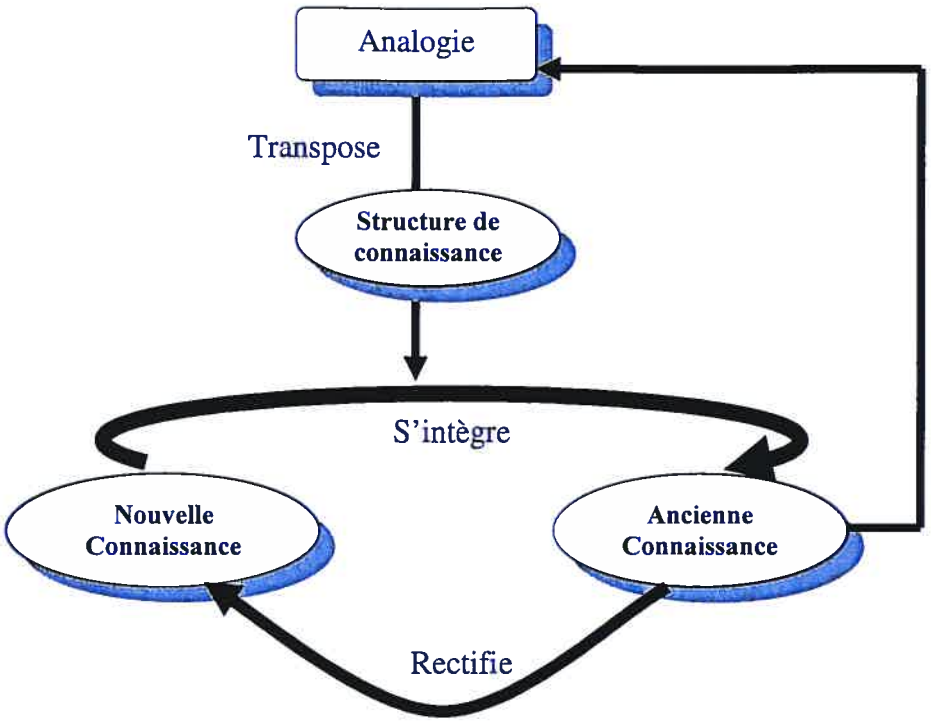


Figure 3.15 Apprentissage à l'aide de l'analogique

3.3.3.1 Rôle conceptuel de la transposition

Gentner, Bowdle, Wolf et Boronat (2001) énumèrent quatre facettes du rôle conceptuel de la transposition. Les constats rapportés par ces auteurs corroborent la théorie expliquant les trois modèles de processus d'appariement structurel. Ces quatre facettes, décrites ci-après, sont : la projection, la liaison entre deux domaines de représentations parallèles, l'extraction d'un système abstrait et les relations lexicales locales.

- **Projection.** *L'analogie crée le contexte. Dans ce cas il s'agit purement d'une projection. Le domaine cible est organisé et structuré selon le système conceptuel emprunté au domaine de la source plus facilement observable. (Lakoff 1990)*

- **Liaison entre deux domaines de représentations parallèles.** Murphy (1996) décrit le parallélisme structurel dans le domaine des représentations. *Étant donné les similarités dans les deux domaines de référence, des systèmes parallèles de relations sémantiques pourraient se développer indépendamment dans les deux domaines. La liaison entre les deux domaines*

reflète l'appariement structurel entre deux domaines de représentations parallèles. (Gentner et Markman 1997; Medin, Goldstone et Gentner 1993). Dans ce cas les deux domaines pourraient partager un système conceptuel sans que l'un soit dérivé de l'autre.

- **Extraction d'un système abstrait (Archéologie cognitive)**. *L'analogie représente un système conceptuel transposé d'un domaine de base à un domaine cible, mais lequel existe maintenant comme un système abstrait pouvant s'appliquer aux deux domaines. Des reliques métaphoriques témoigneraient de l'importance d'une configuration analogique donnée pour l'interprétation du domaine cible dans l'histoire culturelle, ils n'entraîneraient pas de configurations en ligne du domaine concret pendant le raisonnement.*

- **Relations lexicales locales**. *Il n'y a pas dans ce cas de transposition systématique, l'analogie consiste simplement en des polysémies et/ou homophonies. Ici le phénomène est purement lexical et il n'y a pas de structuration ni de transposition.*

3.3.4 Mise en garde sur l'utilisation de l'analogie

Nous avons tenté auparavant de démontrer les bénéfices du recours à l'analogie, nous discutons maintenant des inconvénients ou des précautions à prendre pour éviter une utilisation inappropriée.

Nous empruntons une formulation populaire pour souligner que l'analogie a plusieurs visages. Cette constatation nous incite à construire avec parcimonie l'analogie afin d'obtenir une formulation cognitive adéquate. Pour ce faire, notre attention devra se concentrer sur les objets et les relations qui constitueront la structure analogique à transposer. Nous avons soutenu précédemment que l'appariement structurel est au cœur du processus du raisonnement analogique. La profondeur ou la qualité de cet appariement est influencée par le degré de similarité de la source par rapport à la cible, sans oublier que l'analogie doit se formuler à partir de deux domaines distincts. Ainsi, il est pensable d'utiliser pour un même domaine cible plusieurs analogues de proximité variable de sorte à favoriser l'emploi de différents processus d'appariement. À première vue, on ne perçoit que des avantages à la possibilité de formuler une panoplie d'analogies, malheureusement des résultats insidieux peuvent se produire. En effet, la

recherche sur la mémoire en psychologie, plus précisément sur la récupération d'éléments mémorisés, s'intéresse à deux types de similarité: la similarité littérale basée sur les rapprochements structurels et la similarité superficielle basée sur les rapprochements de surface. Ces deux types de similarité donnent parfois naissance à un paradoxe puisqu'il est actuellement reconnu en psychologie que le processus de transfert est très sensible à la justesse structurelle afin que la connaissance soit exportée d'une description à une autre. Résultante de ce paradoxe : notre mémoire peut souvent nous donner une information non désirée.

Quant à eux, Gentner et Gentner (1983), soulignent que le degré de similarité ou d'appariement des domaines favorise d'autant le transfert. Il s'agit d'une observation réalisée dans le cadre d'une expérimentation du domaine de l'électricité en utilisant des analogies dans le domaine de l'écoulement de l'eau et du déplacement de foule. La problématique consistait à comprendre la distinction entre la liaison en série ou en parallèle de résistances et de piles. Leur étude a démontré que l'assimilation des concepts était dépendante de l'analogie utilisée. Les étudiants avec l'analogie de l'eau s'écoulant de réservoirs en parallèle ou en série différenciaient mieux le concept électrique appliqué à des piles qu'à des résistances. Inversement, les étudiants utilisant l'analogie du déplacement d'une foule sur des ponts différenciaient mieux le concept électrique appliqué aux résistances qu'aux piles. Ce phénomène s'explique par le degré d'appariement des domaines. Ainsi, le transfert s'effectue mieux avec l'analogie de réservoirs d'eau pour les piles puisque la différence de pression entre des réservoirs en parallèle ou en série peut se comprendre plus facilement en terme de hauteur du liquide. D'un autre coté, l'analogie du déplacement de foule sert mieux les résistances. En parallèle les résistances sont comme des ponts placés l'un contre l'autre permettant un plus large flux.

Joshua et Dupin (1989) décrivent un type d'analogie qu'ils qualifient de « modélisante ». Selon eux, ce type d'analogie permet d'éviter la problématique de l'étude pratique, expérimentale de l'objet analogique. Ils expriment cette problématique en précisant qu'aux difficultés de transfert des relations pertinentes du modèle

analogique au modèle-cible, s'ajoutent alors les difficultés de la modélisation de l'objet analogique lui-même et les multiples non-correspondances pratiques entre la phénoménologie des deux objets (Kircher, 1984 ; Tenney et Gentner, 1984). Pour contrer ces difficultés, l'analogie modélisante doit répondre explicitement à certains critères. Tout d'abord, elle doit remplir une fonction figurative, au sens de la présentation d'une image. On peut parler ici, d'un certain aspect métaphorique de l'analogie où la métaphore doit contenir des relations isomorphes à celles du domaine étudié. L'analogie modélisante doit également avoir une fonction descriptive qui permettra à l'apprenant d'admettre la plausibilité d'un système explicatif du domaine traité. La fonction descriptive correspond à l'utilisation de la phrase : « Tout se passe comme si... ». Une caractéristique primordiale de l'analogie modélisante est qu'elle doit correspondre à une situation simplifiée, voire même épurée, qui dans bien des cas ne peut se retrouver dans la réalité. Le recours à ce type d'analogie nécessite habituellement la présentation de systèmes fictifs favorisant les expériences par la pensée. Finalement, l'analogie modélisante doit permettre que l'investissement initial de l'apprenant en vue de la clarification du système analogique soit minime et surtout didactiquement rentable. Joshua et Dupin (1989) ont expérimenté ce type d'analogie à l'aide d'une analogie qui est aujourd'hui connue comme l'analogie du train. Cette analogie tente d'expliquer certaines notions propres à l'électrocinétique à l'aide d'un train composé uniquement de wagons sans locomotive circulant sur une voie ferrée formant un parcours en boucle fermé. N'ayant pas de locomotive, le déplacement du train s'effectue grâce à la poussée permanente des ouvriers sur les wagons qui défilent devant eux avec une force constante.

L'analogie est un excellent moyen heuristique lorsqu'elle est utilisée sous la responsabilité de celui qui en fait usage. Par contre, son utilisation dans la relation didactique en fait « un redoutable moyen de produire des effets Topaze » qui se traduisent par un retard dans l'acquisition de la connaissance de base (Brousseau, 1998). Rappelons que l'effet Topaze survient lorsque la réponse que doit donner l'étudiant est artificiellement favorisée par une négociation à la baisse du contrat didactique, par des consignes si explicites que la bonne réponse peut être produite sans passer par la

connaissance. Toujours dans le contexte de la relation didactique, l'étudiant peut croire que l'objet d'enseignement ne présente aucune difficulté puisqu'il maîtrise bien l'analogue. On parlera dans ce cas de l'effet Jourdain (Brousseau, 1998). Cette notion se trouve souvent présente dans les relations professeur–élèves où l'enseignant utilise un objet familier pour expliquer des savoirs scientifiques. Dans la même veine, lorsque l'enseignant s'aperçoit que son analogie est mal interprétée, il peut désirer se justifier et malencontreusement continuer son action et perdre l'étudiant en prenant ses propres explications et ses moyens heuristiques comme objets d'étude à la place de la véritable connaissance. L'acquisition de la connaissance à apprendre est alors mise en veille et par le fait même retardée.

Certaines considérations découlent de l'orientation de la transposition de l'appariement structurel qui s'effectue de la source vers la cible. En effet, une interprétation différente peut s'effectuer, si la source et la cible sont interverties. Par exemple, « l'acrobate est un hippopotame », suggère un acrobate lourd et maladroit, tandis que « l'hippopotame est un acrobate » suggère un hippopotame gracieux. Dans certains cas, l'analogie ne peut pas être inversée. Par exemple, « une rumeur est un virus », suggère la contagion et l'usage de règles d'hygiène associés à la source, le virus, transposé à la cible, la rumeur. L'inverse, « un virus est une rumeur », n'a pas vraiment de sens. Ces exemples sont tirés de Genter et France (1988).

Voyons maintenant quelles sont les utilisations de l'analogie dans le domaine de l'informatique.

3.3.5 L'utilisation de l'analogie en informatique

L'informatique et plus précisément le développement de programmes est un domaine fertile en analogie. De nombreuses notions abstraites sont comparées à des situations ou des objets familiers pour en faciliter l'apprentissage. La présente section répertorie quelques utilisations de l'analogie dans différentes facettes du développement de programmes.

3.3.5.1 Les systèmes d'exploitation et les icônes

L'interface principale des systèmes d'exploitation est basée sur l'analogie du bureau. À l'image de notre bureau où se trouve pêle-mêle différents objets, l'écran initial du système d'exploitation propose différentes applications de même que plusieurs objets d'usage familier tels la poubelle et l'horloge. L'accès à un dossier, habituellement nommé « mes documents », est équivalent à l'ouverture d'un tiroir permettant l'accès à son contenu.

Les systèmes d'exploitation et la majorité des logiciels utilisent efficacement des icônes pour représenter un objet ou une opération. Ces icônes sont ni plus ni moins que des analogies. Leur utilisation dans l'interface des systèmes d'exploitation a grandement simplifié leur utilisation et favoriser leur intégration dans le grand public. La figure 3.16 présente quelques unes des icônes les plus connues.

Figure 3.16 Quelques icônes d'utilisation standard

À l'instar des systèmes d'exploitation, plusieurs logiciels se basent sur une métaphore pour présenter et faciliter l'accès à leur contenu. Les logiciels pédagogiques utilisent régulièrement des situations bien connues de l'apprenant : déplacement en autobus, salle de classe, bibliothèque, laboratoire, etc.

3.3.5.2 Comparaison de l'activité de programmer avec celle d'écrire

La recherche sur la compréhension de programme informatique a amené certains chercheurs à faire le parallèle entre la production de programmes et la production de textes. L'analogie s'exprime en tout premier lieu par l'existence des trois mêmes activités: la planification, le codage et la révision. Le tableau 3.7 présente chacune de ces activités dans la production d'un texte et leur correspondance dans la production d'un programme.

Tableau 3.7 La production d'un texte est analogue à la production d'un programme

PRODUCTION D'UN TEXTE	PRODUCTION D'UN PROGRAMME
La planification de la structure du texte	La récupération des connaissances pertinentes pour la résolution du problème et construction d'une solution abstraite
La traduction ou mise en texte du plan du texte	L'implémentation d'une solution dans un langage de programmation
La révision du texte	La révision de l'implémentation ou la solution abstraite ou la représentation du problème.

3.3.5.3 Comparaison de structures de données avec des objets courants

Les structures de données sont utilisées en programmation pour mémoriser une quantité importante de données de façon efficace. Outre des considérations d'espace mémoire, les structures sont conçues afin de faciliter les manipulations de données tels: l'accès, la recherche, l'insertion et le retrait. La présentation de ces structures s'inspire souvent d'analogies. Le tableau 3.8 suivant présente quelques structures de données classiques et la ou les comparaisons les plus courantes. Précisons que même si leur nom nous est familier, ces structures sont dites abstraites (Abstract Data Type) en programmation.

Tableau 3.8 Des objets usuels et leur structure de données correspondante

OBJET	STRUCTURE DE DONNÉES
Pile de livre – Distributeur d'assiettes	Pile
File d'attente à la banque	File
Anneaux enchaînés (communément une chaîne)	Liste chaînée
Réseau du métro	Graphe

3.3.5.4 Comparaison de l'exécution d'un programme

Dans un article intitulé « Observation de programmes par la combinaison d'analogies », Damien Ploix (1997) propose une méthode de composition dans une même image des représentations de la structure, des données et leurs valeurs ainsi que du comportement du programme. Cette méthode doit permettre d'obtenir une vision globale des différents aspects d'un programme. Le tableau 3.9 résume les correspondances entre les caractéristiques d'un programme et certaines structures urbaines.

Tableau 3.9 L'analogie entre des structures urbaines et des caractéristiques d'un programme

REPRÉSENTATION	CARACTÉRISTIQUE REPRÉSENTÉE
Routes	Flux de contrôle
Habitations (immeubles ou maisons individuelles)	Fonctions
Etages et pièces	Instructions
Objets présents dans les jardins	Données
Animation d'un personnage	Evolution de l'exécution
Transformations et déplacements des objets présents dans les jardins	Evolution des données

La figure 3.17 montre la représentation d'un programme par une structure urbaine composée d'immeubles et de maisons individuelles qui représentent le corps des fonctions du programme. Ploix ajoute que ces dernières sont elles-mêmes composées de pièces liées à chaque instruction des fonctions. De plus, les habitations sont entourées d'un jardin contenant la représentation des données manipulées, et de routes, représentant les liaisons entre les fonctions, qui seront également utilisées au cours des traces de l'exécution du programme par un personnage se déplaçant de fonction en fonction et de pièce en pièce selon la progression de l'exécution.

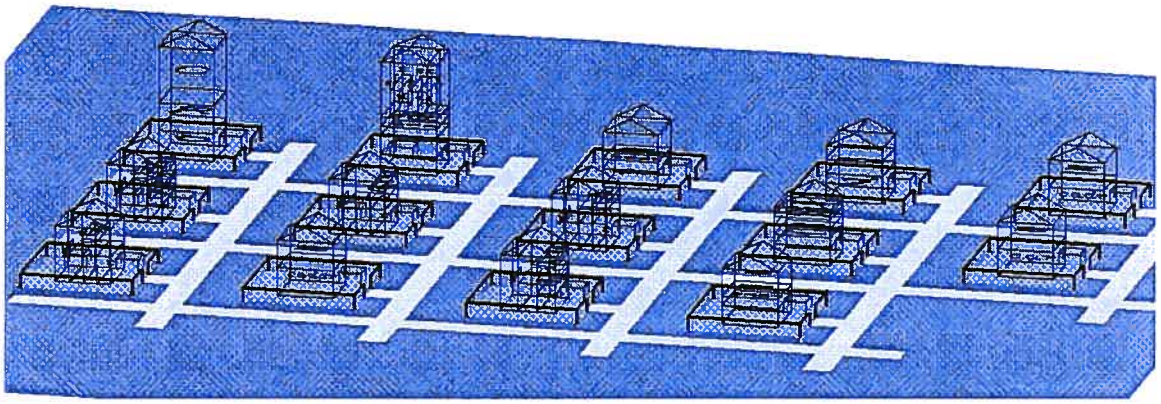


Figure 3.17 La représentation d'un programme par une structure urbaine
(Tirée de l'article de Damien Ploix)

3.3.5.5 Création de tutoriel d'apprentissage de résolution de problèmes

Notre dernier champ d'application de l'analogie est celui du tutoriel informatique. Certains tutoriels offrent à l'apprenant un module de résolution de problème. Ce module est construit de façon à guider ou diriger le raisonnement de l'étudiant vers la solution. Pour ce faire, l'approche classique en intelligence artificielle consiste à comparer le problème à résoudre avec certaines solutions de problèmes similaires ou analogues. La résolution du problème consiste alors à l'application d'opérateurs sur la structure d'une solution existante pour obtenir un nouvel état menant de l'état initial à l'état visé.

Les différentes techniques de résolution automatisée se distinguent par la structure de données utilisée pour la mémorisation des solutions et par la méthode de recherche de la solution. Prenons comme exemple celui des réseaux neuronaux.

Dans ce cas, la mémoire des solutions ou mémoire à long terme est un réseau où les nœuds représentant des concepts sont reliés par un ensemble de liens représentant des relations. Le réseau est utilisé pour rechercher l'analogie. La solution du problème s'exprime sous forme d'un réseau où chaque nœud a un niveau d'activation signifiant son importance et chaque lien est étiqueté d'une valeur représentant le poids de la relation entre les concepts.

La sélection de l'opérateur s'effectue en trois étapes:

- recherche d'un ensemble de buts précédents;
- évaluation et sélection de l'analogie appropriée;
- sélection d'un opérateur selon l'analogie sélectionnée.

L'évaluation de l'analogie est basée sur le degré d'appariement, les objectifs et l'historique de succès ou d'échecs des recherches précédentes. La recherche de l'opérateur nécessite une procédure de propagation d'activations qui consiste à augmenter le poids de certains liens.

3.3.6 Utilisation envisagée de l'analogie

Afin d'en faire un usage des plus appropriés, nous avons initialement étudié l'état de la recherche sur le domaine. Les différents modèles présentés à la section 3.3.3 qui tentent d'expliquer le processus d'appariement structurel nous dévoilent une grande variabilité pouvant exister entre les utilisateurs d'une analogie. Lors d'études, plusieurs chercheurs ont relevé l'influence de la systémacité et la cohésion structurelle dans l'interprétation de l'analogie (Clement et Gentner 1991; Keane 1996; Spellman and Holyoak 1992, 1996; Markman 1997 ; Lassaline 1996; Wu et Gentner 1998; Gentner, Rattermann et Forbus 1993; Holyoak et Koh 1987; Ross 1989). Une caractéristique découlant de la systémacité et la cohésion structurelle est le degré de similarité entre l'analogue et la cible. Fort de l'éclairage de ces considérations, nous envisageons d'utiliser l'analogie principalement sous deux formes. La première forme consiste à comparer un concept informatique à une situation ou objet familier et bien connu. À la recherche du meilleur déclic analogique possible nous utilisons une à trois analogies différentes pour le même concept avec des degrés de similarité distincts. Les analogies sont ordonnées selon notre perception de la proximité structurelle de l'objet analogue, du plus éloigné ou plus près, eu égard à la facilité d'identifier les attributs et les relations. Par exemple, l'écriture dans un fichier texte est comparée à la construction d'une voie ferroviaire, à l'enregistrement d'une vidéo-cassette et à l'affichage à l'écran. La deuxième forme d'analogie est la

similitude algorithmique de la résolution de deux problèmes différents. Déterminer le plus haut salarié d'une entreprise ou déterminer la température la plus élevée d'un mois nécessitent des opérations similaires. La résolution de l'un devrait favoriser la résolution de l'autre.

L'étudiant en situation d'apprentissage, plus particulièrement en ingénierie, peut facilement crouler sous l'abondance des nouvelles notions. Il a besoin de point d'ancrage, de contexte et d'utiliser sa connaissance. Une approche structurante lui permet d'accrocher la nouvelle connaissance à d'autres notions apprises. L'analogie permet de favoriser la compréhension par la comparaison de nouvelles notions avec des notions déjà acquises.

Le rôle de l'analogie est celui des formes utilisées pour couler le ciment d'une fondation. Les formes retiennent le ciment pour qu'il se solidifie comme l'analogie soutient la nouvelle connaissance pour empêcher qu'elle se dissipe. Lorsque le ciment est bien solide, les formes peuvent être retirées comme l'analogie se dissocie de la nouvelle connaissance une fois bien assimilée.

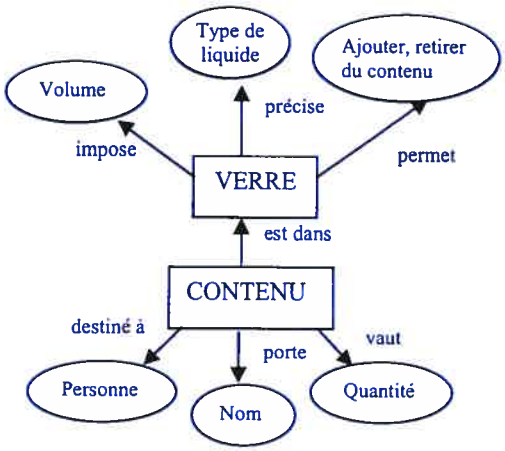
Nous espérons que cette pratique d'utiliser l'analogie se traduira par une véritable appropriation du concept d'analogie par l'étudiant qui pourra alors s'en servir comme d'un outil cognitif au même titre que le langage pour s'expliquer des phénomènes inconnus en développant des analogies spontanées. De cette façon, l'étudiant pourra non seulement utiliser les analogies que nous lui proposons mais construira sa propre analogie qui devrait nous révéler une partie de sa construction mentale de la notion à apprendre lorsqu'il la verbalisera. Finalement, la cible ainsi acquise pourra même devenir un analogue. Par exemple, l'ordinateur est devenu une source dans les théories de l'information qui sont construites et structurées à partir des fondements et des structures de l'ordinateur.

3.3.6.1 Exemples d'analogies

L'analogie peut être une alliée extrêmement utile dans l'apprentissage de la programmation. Quelques exemples d'analogies nous permettront d'illustrer clairement le type et l'utilisation envisagés. La première analogie compare une variable à un verre. La deuxième analogie compare un fichier texte à une vidéo-cassette. Finalement, la lecture dans un fichier est comparée au déplacement du train sur le rail.

Prenons la première analogie sur la notion de variable et de type. Conformément à la théorie de la transposition analogique de Gentner (1989) nous présentons à la figure 3.18 les relations transposables d'un domaine à l'autre.

Le verre et son contenu



La variable et son type

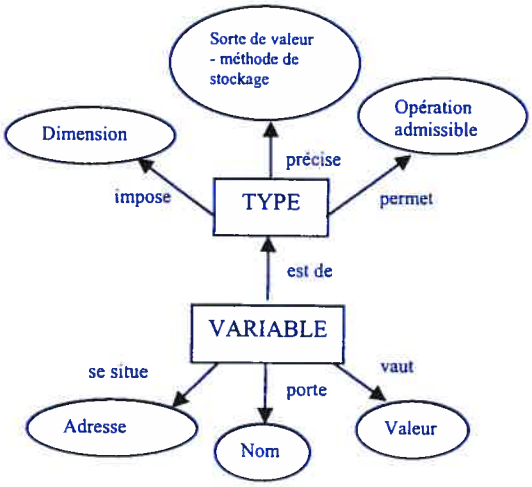


Figure 3.18 Analogie du verre et son contenu pour la relation de la variable et son type

Tout d'abord, rappelons qu'une variable correspond à un emplacement mémoire où est inscrite une donnée qui peut être modifiée. La dimension de l'emplacement, la nature de la donnée et les manipulations possibles sont prescrits par le type. L'analogie est celle d'un contenant et son contenu précisé par un liquide dans un verre.

D'un côté, le contenu est identifié par un nom, du vinaigre balsamique, une certaine quantité, soit 150 ml est dans le verre qui est une tasse à mesurer pouvant contenir de 0 à 250 ml. De l'autre, la variable est identifiée par un nom, par exemple Age, et correspond à une valeur qui doit être à l'intérieur des valeurs minimale et maximale de son type, par exemple `unsigned char` en C permet des valeurs entières comprises entre 0 et 255. Cette analogie transpose la relation d'un contenant et son contenu à celle d'une variable et son type.

Les fichiers textes en informatique sont des fichiers séquentiels ne contenant que des caractères. Le terme séquentiel signifie que les données sont mémorisées en continu l'une après l'autre. Pour ces fichiers, les opérations d'écriture et de lecture sont exclusives au sens où une seule des opérations est possible à la fois. Pour manipuler adéquatement ces fichiers l'étudiant doit acquérir des connaissances concernant une séquence d'opérations à réaliser, leur transposition selon la syntaxe du langage de programmation et la structure physique de la composition du fichier. Pas évident, tout cela pour l'étudiant! Mais avec l'aide de l'analogie le concept peut devenir concret et familier. En effet, il suffit de comparer le fichier texte à une vidéo-cassette. Le tableau 3.10 présente les correspondances entre les caractéristiques d'un fichier et d'une vidéo-cassette.

Tableau 3.10 L'analogie du fichier et de la vidéo-cassette

FICHIER	VIDÉO-CASSETTE
Information séquentielle	Enregistrement séquentiel sur le ruban
Exclusivité des opérations de lecture et d'écriture	Impossibilité de visionner un enregistrement lorsqu'on enregistre une émission
Ouverture Lecture Fermeture du fichier	Sélectionner la cassette Visionner la cassette Enlever la cassette

Comme la plupart des analogies, celle-ci n'est pas parfaite. Entre autres, l'ouverture d'un fichier en mode standard d'écriture (enregistrement) effacera tout le contenu présent dans le fichier; ce phénomène n'existe pas sur la vidéo-cassette.

Une deuxième analogie nous permet de comparer la lecture dans un fichier au déplacement d'un train sur le rail. À première vue, l'analogie est surprenante puisqu'on pourrait tout naturellement comparer la lecture dans un fichier à la lecture d'un texte. En effet, le texte est lu phrase après phrase, paragraphe après paragraphe, page après page. Par contre, rien n'empêche le lecteur de tricher un peu et de sauter un paragraphe ou de revenir en arrière pour relire la dernière phrase. Cette dernière constatation est malencontreusement transposée à la lecture d'un fichier et résulte en une conception erronée. Obligatoirement, la lecture dans un fichier texte doit se réaliser sans saut. Il nous faut une image plus « forte » pour souligner cette restriction. L'analogie du train, figure 3.19, répond à cette requête. L'impossibilité de sauter des données se compare à l'impossibilité de lever le train ou de le faire voler au-dessus du rail. Le train parcourt le rail sans jamais le quitter tronçon après tronçon comme les données doivent être lues dans le fichier une après l'autre.

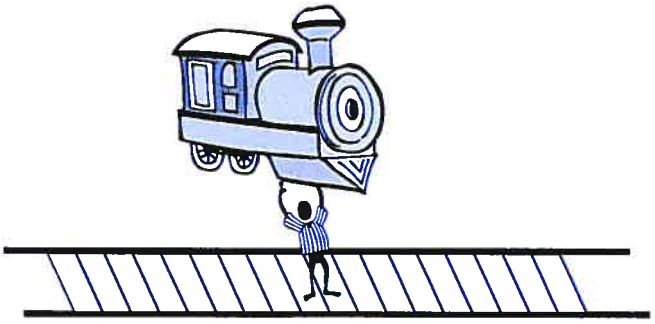


Figure 3.19 L'analogie du train

Le troisième et dernier exemple d'analogie compare l'écriture dans un fichier à l'affichage à l'écran dans une fenêtre de texte. Dans ce cas, les connaissances fraîchement acquises des opérations d'affichage à l'écran sont transposées à l'écriture d'un fichier puisqu'elles se comportent de façon identique. L'inscription invisible des données dans un fichier doit être réalisée exactement de la même manière que si nous les

affichions à l'écran. De même, l'étudiant saisira l'importance de séparer les données par des espaces ou des sauts de ligne.

Ces trois exemples d'analogie démontrent l'étendue des possibilités. La première, la vidéo cassette et le magnétoscope, bénéficie de sa similitude avec les fichiers texte et de leur familiarité pour faciliter l'acquisition des concepts propres à la manipulation de ces fichiers. La deuxième analogie, celle du train, est utilisée principalement pour corriger une perception erronée et prévenir une opération impossible. La troisième analogie, très proche de la notion à acquérir, permet de consolider l'apprentissage de notions antérieures en encourageant leur transposition.

L'analogie peut être accrocheuse comme elle peut être « boiteuse ». Il faut s'assurer d'avoir une bonne analogie en tenant compte des mises en garde exposées précédemment (voir à ce sujet la section 3.3.4). Une autre forme d'analogie que nous désirons exploiter est celle liant l'exécution d'un programme et le fonctionnement d'un matériel.

L'analogie n'est pas l'objet principal de notre recherche. Dans notre projet de recherche nous aurons recours à l'analogie pour expliquer certains concepts présentés dans le tutoriel et pour réifier l'abstrait à partir d'un comportement concret.

Dans les chapitres précédents nous avons répondu à plusieurs questions :

- Quel est l'intérêt de l'étudiant pour la programmation? Quelle est sa connaissance initiale en informatique? Quelle est son expérience en travail collaboratif? Quelle méthode d'apprentissage semble-t-il privilégier?

- Quelles sont les stratégies utilisées par un apprenant (novice) pour concevoir un programme informatique? Quels sont les mécanismes cognitifs mis en œuvre dans la conception de programme informatique?

- Quel rôle l'analogie peut-elle jouer dans l'apprentissage de la programmation? Comment peut s'effectuer l'appariement entre la notion à apprendre et

l'analogue? Quelle est l'utilisation envisagée de l'analogie dans notre enseignement?

En tant qu'enseignant, les réponses obtenues nous incitent maintenant à répondre à la question suivante :

- Comment incorporer les informations acquises des questions précédentes afin d'en soutirer le plus grand bénéfice possible?

Nous croyons qu'une réponse possible à cette question pourrait nous provenir de la pédagogie de projet. Pour ce faire, nous revenons, dans le prochain chapitre, à la source même de l'acte pédagogique afin de situer le plus précisément possible l'enseignement par projet.

3.4 Considérations sur l'enseignement par projets

> *L'une de mes principales convictions mathématiques est que la construction réalisée « dans sa tête » est souvent réussie lorsqu'une construction « dans le monde réel » lui fait écho – un château de sable, un gâteau, une maison en Légos, une entreprise, ... Une production dans le monde réel signifie, montrée, critiquée, explorée et admirée.*
>
> Seymour Papert
> Père du langage LOGO

> *La technologie ne doit pas être un conducteur mais un outil. Elle ne peut pas être le marteau qui enfoncera le clou de la connaissance dans la tête de l'apprenant.*
>
> Herbert A. Simon (1998)

> *On apprend en faisant.*
>
> Maman, papa, grand-maman, grand-papa, etc.

Principalement pour des raisons de logistique nous réaliserons notre expérience dans une approche par problèmes. Par contre, nous pensons que le genre d'activité proposée pourrait éventuellement s'inscrire dans une pédagogie de projet. La rédaction de ce chapitre constitue pour nous une exploration dans le domaine de la pédagogie de projet afin d'identifier ses principaux enjeux. Nous serons par la suite mieux outillé pour appliquer adéquatement ce paradigme.

Seymour Papert (1994) décrit le constructionnisme comme sa reformulation personnelle du constructivisme. Pour lui, le constructionnisme se distingue par l'importance et le rôle particulier des constructions réelles comme supports des constructions mentales. Papert, que nous avons cité au départ, plaide explicitement pour la réalisation d'un objet concret permettant de révéler la construction mentale adéquate d'une connaissance, d'un concept. Cette réalisation serait d'une grande richesse dans le processus d'apprentissage. En effet, puisqu'elle peut être vue, critiquée ou explorée, par son concepteur et/ou ses pairs, elle peut favoriser la détection d'erreurs de compréhension et motiver un approfondissement. De ce besoin d'approfondissement, associé à l'intervention des

pairs facilitée par le caractère visible et donc partagé de l'objet, émergera un véritable apprentissage collaboratif.

Dans un article intitulé "What we know about learning", Hebert A. Simon (1998) professeur de psychologie et d'informatique à l'université Carnegie Mellon, aborde l'apprentissage par l'étude de la conception de tutoriels proches des systèmes experts. Il associe le processus de conception d'un tutoriel à la résolution de problèmes. Simon précise que le processus de conception s'effectue selon un cycle continuel de génération d'alternatives et d'épreuves d'évaluation. Le concepteur choisit une alternative parmi un ensemble donné et éprouve son choix. Par contre, l'idée de connaître au départ tout l'ensemble des alternatives possibles est irréaliste. En effet, pour la conception en ingénierie ou en architecture, les buts ne sont jamais complètement définis tant que la conception n'est pas terminée. De plus, le plus grand effort et la majorité du temps sont consacrés à la génération de ces alternatives. L'apprentissage prend place dans la pensée de l'étudiant, nulle part ailleurs et le rôle de l'enseignant réside à l'inciter à le faire. Il doit trouver des activités qui engageront l'étudiant dans l'organisation des informations de développement de savoir-faire et l'acquisition de savoirs.

Simon (1998) affirme que l'une des meilleures techniques favorisant un bon apprentissage est de montrer des solutions de problèmes étape par étape. Ces solutions confrontent l'étudiant à déduire le passage d'une étape à l'autre. De cette façon l'étudiant développe un raisonnement se composant de l'opérateur conditionnel SI-ALORS. Par exemple, SI état_initial ALORS *action* pour obtenir état_visé.

Les propos de Papert et Simon prônent l'organisation d'une activité bâtie à partir d'un objet réel nécessitant des manipulations. Cette activité doit confronter l'étudiant à une situation concrète, réelle, qu'il doit analyser afin de déduire l'opération à réaliser pour obtenir une nouvelle situation. Il devra de nouveau, analyser cette nouvelle situation et réaliser une opération pour obtenir la prochaine situation. Cette procédure se poursuit tant que l'état final n'est pas atteint. Dans notre recherche nous favoriserons ce type de raisonnement tel que décrit par Simon. De plus, l'étudiant devra traduire ce

raisonnement dans le langage informatique cible à l'aide de structures conditionnelles ou de structures répétitives que l'on retrouve aujourd'hui dans les langages de programmation.

Les sections suivantes énoncent les composantes et les caractéristiques de la pédagogie par projets. La première section discute brièvement de l'utilisation de l'approche constructivisme dans l'enseignement de l'informatique. La deuxième section présente spécifiquement la pédagogie de projet. Finalement, la troisième section discute des changements d'attitudes qui favoriseront l'implantation optimale de la pédagogie de projet.

3.4.1 Le constructivisme

L'enseignement et l'apprentissage des concepts informatiques dans les différents programmes liés à l'informatique sont généralement basés sur deux modèles fondamentaux : l'objectivisme et le constructivisme (Van Gorp & Grissom, 2001). L'objectivisme est basé sur le modèle comportemental en psychologie, communément appelé le béhaviorisme. Pour les béhavioristes, l'apprentissage peut se résumer à l'acquisition d'un nouveau comportement. Plus précisément, les objectivistes croient que l'apprentissage s'effectue lorsque l'étudiant écoute les explications d'un enseignant, s'engage dans une pratique soutenue tout en répondant à des stimuli externes (Fosnot, 1996; Skinner, 1953). L'évaluation de l'apprentissage s'effectue par l'observation de comportements quantifiables sur une tâche prédéfinie (Fosnot, 1996). L'objectivisme correspond à un enseignement centré-professeur. De son coté, le constructivisme se concentre, plus particulièrement, sur les expériences de l'étudiant. Ce modèle est basé sur la psychologie cognitive (Atkinson, R., & Shiffrin, R., 1968). L'apprentissage selon le constructivisme est une construction personnelle de notre propre compréhension du monde dans lequel nous vivons à partir des expériences vécues. Ainsi, l'apprentissage exige de l'étudiant de construire activement sa compréhension avec ses propres repères significatifs en rapport avec ses expériences précédentes tout en considérant les perspectives d'alternatives apportées par les autres (Bednar, Cunningham, Duffy, &

Perry, 1992). Chaque individu génère ses propres règles et modèles mentaux qui donneront un sens à ses expériences. L'évaluation mesurera dans ce cas le fonctionnement adéquat de l'apprenant dans une discipline cible et sa capacité à défendre et expliquer ses décisions à l'aide des habiletés métacognitives développés (Bednar et al. 1992). Pour le constructivisme, on peut résumer que l'apprentissage est simplement un processus d'ajustement de notre modèle mental pour accommoder de nouvelles expériences. Koyanagi (1999) inscrit dans un tableau comparatif, tableau 3.11, les principales caractéristiques des deux modèles.

Tableau 3.11 Caractéristiques des modèles objectivisme et constructivisme

Psychologie comportementale	**Psychologie Cognitive:**
• Psychologie basée sur un comportement observable • Comportement est déterminé par dénouements/conséquences Connaissance se manifeste dans le comportement (réponse correcte) **Objectivisme:** • Méthode: o Présentation de contenu o Question posée à l'étudiant o Si la réponse est bonne, l'étudiant en est informé o Renforcement positif pour les bonnes réponses o Cycle répété pour les mauvaises réponses • Les réalités externes et la connaissance sont à mémoriser pour l'apprenant • Contrôle enseignant Étudiant apprend le contexte	• L'apprenant est comme un processeur traitant des informations (modèle basé ordinateur) • Emphase sur les états mentaux internes Considère la perspective et la connaissance de l'étudiant **Constructivisme :** • Diverses méthodes: o Encourage la formation de la connaissance o Processus différent pour chaque étudiant o Exploration auto-dirigée o Apprentissage par la découverte o Construction de concepts, schéma et modèles mentaux • Réalité et connaissance sont construites par l'étudiant en se basant sur la perspective et l'expérience • Enseignant observe, supervise et facilite Étudiant crée le contexte

Van Gorp et Grissom (2001) nous apprennent qu'en classe le constructivisme est souvent exploité sous la forme d'un environnement de résolution de problèmes basé sur la règle des trois C : Contexte, Construction et Collaboration. Le premier C, le contexte, précise que le problème soumis à l'étudiant doit se situer dans un contexte authentique et habituellement simplifié. Cette préoccupation mise sur la motivation personnelle de l'étudiant par rapport à la motivation externe encouragé par l'objectivisme. Le deuxième C, la construction, indique que l'étudiant construira sa connaissance à partir d'activités signifiantes. Finalement, le troisième C, la collaboration, tente de tirer avantage de la collaboration avec les pairs. La collaboration aidera le processus de construction de la connaissance puisque l'étudiant pourra examiner des perspectives alternatives à sa solution, ce qui pourrait l'amener à reconstruire ses propres perspectives et conséquemment ses solutions.

Dans le domaine de l'enseignement de l'informatique il y a relativement peu d'articles empiriques traitant du constructivisme si on le compare, par exemple, à l'enseignement des mathématiques (Van Gorp & Grissom, 2001). En effet, certaines études ont démontré les bienfaits du constructivisme dans la réforme américaine de l'enseignement des mathématiques (Kim, Sharp et Thompson, 1998). Par contre, d'autres articles du domaine de l'éducation de l'informatique ont noté l'apport positif de la collaboration et la coopération. Par exemple, Ramsey, Rada et Acquah (1994) écrivent que l'apprentissage collaboratif des étudiants en informatique peut être efficace lorsqu'une méthode de travail bien formulée existe. Keeler et Anson (1995) ont démontré pour leur part que l'apprentissage coopératif améliore significativement la performance d'apprentissage ainsi que la rétention lors des cours théoriques. De plus, d'autres auteurs tels Gorriz et Medina (2000) et McGrath (1990) ont noté que les filles en particulier excellent dans les environnements informatiques lorsque l'apprentissage est collaboratif.

Wills et al. (1999) utilisent le terme « apprentissage par les pairs » pour englober l'apprentissage collaboratif et l'apprentissage coopératif. Les résultats de leur recherche proviennent de deux ateliers (workshops juin 96 et juin 97) regroupant divers éducateurs en informatique réunis au campus du Worcester Polytechnic Institute (WPI). L'objectif

était alors d'étudier l'utilisation de l'apprentissage par les pairs dans le cadre de cours d'introduction à l'informatique. Les participants provenaient ainsi de différentes institutions qui ont utilisé différents types d'activités d'apprentissage par les pairs dans différents types de cours d'introduction à l'informatique. Les auteurs regroupent les activités en trois types :

Acquisition d'informations en classe	▪ Discussion entre les membres de l'équipe portant sur un exercice nécessitant la construction d'un dispositif permettant de résoudre une problématique quelconque. ▪ Cet exercice permet aux membres de l'équipe de se connaître.
Tâche réalisée en classe ou en laboratoire constituant une petite portion de l'évaluation	▪ Un enseignement de groupe où les étudiants sont responsables de l'enseignement d'une notion particulière aux autres membres du groupe. ▪ Un examen de groupe où les étudiants travaillent sur un exercice en groupe et dévoilent aux autres leurs solutions. ▪ Un exercice de débogage où les étudiants sont couplés pour déboguer les programmes des autres ou un programme imposé par l'enseignant. ▪ Un exercice d'écriture où les étudiants travaillent en équipe pour énumérer les concepts importants du jour ou de la semaine et préciser comment ces concepts s'appliquent à leur carrière.
Projet de programmation hors classe avec une évaluation individuelle et de groupe	▪ Les étudiants d'un groupe peuvent travailler collectivement ou individuellement à la conception de différents aspects du programme.

Voici quelques résultats de l'étude de Wills et al (1999) qui ont retenu notre attention. Le premier type d'activité est celui où les participants ont rapporté le plus d'expériences

et en ressentent des avantages dans leur enseignement, sans toutefois les nommer. Le deuxième type d'activité est, selon les participants, celui offrant le plus grand impact du point de vue apprentissage par les pairs. Il permet aux étudiants d'apprécier et de mieux comprendre la collaboration. À leur avis, ce type de situation permet aux étudiants d'être plus actifs et plus engagés dans leur apprentissage. Il permet à l'étudiant d'accroître sa confiance en lui lors de discussions en formulant sa réponse à l'aide du « Nous » plutôt que le « Je ». Le dernier type d'activité est le moins utilisé par les participants. Il est celui pouvant présenter plusieurs difficultés organisationnelles. Par exemple, la composition de l'équipe, en considérant, la capacité et la motivation de chaque étudiant. Ou encore, la problématique liée à l'évaluation de manière à tenir compte de la contribution personnelle de chaque membre de l'équipe.

Wills et al. (1999) soutiennent que l'activité du projet de programmation est celle présentant les plus grands avantages. Les auteurs justifient cette affirmation comme suit. En premier lieu, la réalisation d'un projet en équipe peut accommoder un grand nombre d'étudiants tout en nécessitant peu de ressources à l'école. Les projets en équipe sont naturellement utilisés dans les cours d'introduction à l'informatique où les étudiants commencent à utiliser les structures de données et font de la programmation. De plus, un groupe d'étudiants travaillant sur un exercice costaud accomplira plus de boulot qu'un étudiant seul. Finalement, à l'occurrence d'un problème, les étudiants travaillant en équipe développeront une meilleure compréhension des notions en cause qu'un étudiant seul.

Un enseignant ne peut pas simplement demander à ses étudiants de travailler en équipe. Il doit en premier lieu leur enseigner à travailler efficacement en coopération. Il peut assigner des rôles spécifiques à chaque membre (e.g. leader, lecteur, secrétaire) afin d'éviter la dominance d'un membre. Il doit également préciser comment il évaluera l'équipe et chacun de ses membres. Johnson et al. (1991) précisent que : « *si un étudiant ne peut pas voir les bénéfices du travail en équipe, il pourra préférer travailler individuellement plutôt que comme membre d'une équipe; une bonne tâche coopérative nécessite d'avoir une* ***interdépendance positive*** *entre tous les membres du groupe* ».

Puisque nous désirons proposer des exercices s'inscrivant dans une pédagogie par projet, nous présentons à la section suivante différentes notions qui nous permettront d'utiliser adéquatement cette pédagogie.

3.4.2 Le projet: une situation d'apprentissage

Laurent Dubois (1999) affirme que pour pratiquer la pédagogie de projet, il faut d'abord adhérer aux récentes théories de l'apprentissage qui ont amené à poser une distinction entre enseignement et apprentissage. Cette distinction modifie singulièrement les relations au sein du triangle didactique " professeur - étudiant - savoir ". Dans cette optique, le professeur n'est plus celui qui transmet des savoirs, l'étudiant n'est plus le sujet plus ou moins passif de ses apprentissages, l'accès à la connaissance ne se fait plus par placages successifs de notions.

Philippe Perrenoud (1999) s'est questionné sur la pédagogie par projet, à savoir son utilité, son application, etc. Il propose la définition suivante qu'il associe à une démarche de projet. Ainsi, pour Perrenoud une démarche de projet

- *est une entreprise collective gérée par le groupe-classe (l'enseignant anime, mais ne décide pas de tout) ;*
- *s'oriente vers une production concrète (au sens large: texte, journal, spectacle, exposition, maquette, carte, expérience scientifique, etc.) ;*
- *induit un ensemble de tâches dans lesquelles tous les élèves peuvent s'impliquer et jouer un rôle actif, qui peut varier en fonction de leurs moyens et intérêts ;*
- *suscite l'apprentissage de savoirs et de savoir-faire de gestion de projet (décider, planifier, coordonner, etc.) ;*
- *favorise en même temps des apprentissages identifiables (au moins après-coup) figurant au programme d'une ou plusieurs disciplines (français, musique, éducation physique, géographie, etc.).*

Perrenoud souligne l'importance d'avoir une définition commune et acceptée de tous afin de mieux cerner les attentes à la fois des étudiants et de l'enseignant. En acceptant cette définition, Perrenoud soutient qu'une démarche de projet, dans le cadre scolaire, peut viser un ou plusieurs des objectifs suivants:

- *Entraîner la mobilisation de savoirs et savoir-faire acquis, construire des compétences.*
- *Donner à voir des pratiques sociales qui accroissent le sens des savoirs et des apprentissages scolaires.*
- *Découvrir de nouveaux savoirs, de nouveaux mondes, dans une perspective de sensibilisation ou de « motivation ».*
- *Placer devant des obstacles qui ne peuvent être surmontés qu'au prix de nouveaux apprentissages, à mener hors du projet.*
- *Provoquer de nouveaux apprentissages dans le cadre même du projet.*
- *Permettre d'identifier des acquis et des manques dans une perspective d'auto évaluation et d'évaluation bilan.*
- *Développer la coopération et l'intelligence collective.*
- *Aider chaque étudiant à prendre confiance en soi, renforcer l'identité personnelle et collective à travers une forme d'empowerment, de prise d'un pouvoir d'acteur.*
- *Développer l'autonomie et la capacité de faire des choix et de les négocier.*
- *Former à la conception et à la conduite de projets.*
- *Impliquer un groupe dans une expérience « authentique », forte et commune, pour y revenir sur un mode réflexif et analytique et y ancrer des savoirs nouveaux.*
- *Stimuler la pratique réflexive et les interrogations sur les savoirs et les apprentissages.*

Le projet doit être mobilisateur pour les étudiants en les orientant vers une réalisation concrète. Il doit solliciter les ressources cognitives et favoriser leur intégration de manière à produire un contexte d'action nécessaire au développement des compétences. Le projet peut permettre de justifier des apprentissages scolaires en sollicitant des pratiques et des compétences qui utilisent des savoirs. Cette justification se traduira par un accroissement de la motivation de l'étudiant qui peut s'exprimer par une plus grande disponibilité à apprendre.

Perrenoud ajoute que le projet est un outil majeur d'observation formative. En effet, l'enseignant peut se servir des projets pour mieux connaître et comprendre ses étudiants et pour mieux identifier, parce qu'il les voit à l'œuvre dans des tâches multiples et complexes, leurs acquis et leurs difficultés.

Le projet doit permettre à l'étudiant d'être le principal artisan de son apprentissage. Par contre, les contraintes de réalisation, les normes à respecter et les objectifs de maîtrise à

atteindre, inévitables dans un contexte scolaire, ne doivent pas empêcher un cheminement personnel de la part de l'étudiant.

L'apprentissage visé par un projet peut porter sur un ensemble précis de connaissances à maîtriser, mais, le plus souvent, ces connaissances entretiennent un lien avec le développement d'une habileté, d'une attitude ou d'un comportement ou la réalisation d'une activité concrète. Cette habileté, cette attitude, ce comportement ou cette activité sont même fréquemment présentés comme l'objectif réel. Toutefois, on ne peut atteindre cet objectif que si on a d'abord maîtrisé un certain corpus de connaissances. Selon Dubois (1999) l'approche par projet se prête aisément à un dépassement des matières traditionnelles et appelle l'établissement de nouvelles "connexions"; le corpus de connaissances évoqué est donc lui-même influencé par l'évolution du projet.

L'utilisation de la pédagogie de projet se traduira par un incessant va et vient entre une tâche originale, le « projet », et des activités qui permettent aux étudiants de se questionner par rapport à cette tâche, de structurer leurs connaissances, d'étendre et de stabiliser des savoirs et savoir-faire.

3.4.2.1 Le rôle du professeur

Dans une pédagogie de projet, le professeur n'est plus le détenteur du savoir. Il organise les activités et tente d'y apporter un éclairage didactique dans le but d'enclencher des apprentissages. Il peut proposer des idées pour relancer les groupes « en panne » et encourager l'ensemble de la classe. Il doit s'assurer que le lien entre le projet à proprement parler et les différentes notions à acquérir est correctement réalisé.

Le professeur ne doit pas être le « souffleur de solutions ». Il doit plutôt jouer le rôle d'un aiguilleur. Il doit analyser l'état du projet, chercher à inférer le raisonnement de l'étudiant et intervenir de sorte à provoquer l'émergence des idées propres à suggérer l'étape suivante. Son défi est de s'ajuster aux différentes procédures envisagées par les étudiants et de les acheminer vers la solution. Il ne doit pas succomber à la tentation de court-circuiter la démarche de l'étudiant et de tout défaire… afin d'imposer la solution idéale.

3.4.2.2 Donner du sens aux apprentissages

La pédagogie de projet devrait donner du sens aux apprentissages des étudiants. Le codage erroné d'une structure de répétition peut se traduire par l'absence de résultat ou l'affichage continuel du même résultat. La conséquence est banale, un écran noir ou un écran tapissé de caractères en mouvement. Par contre, l'ajout d'un objet concret pour penser dans le processus de programmation confronte directement l'étudiant avec le résultat obtenu et sa conséquence, par exemple le moteur ne démarre pas ou le moteur ne s'arrête jamais. La motivation, voire, le plaisir d'entendre le moteur démarrer et s'arrêter au bon moment engage l'étudiant vers un apprentissage plus signifiant.

Un autre aspect important est la contribution de l'entourage facilitée par le caractère visible de l'objet. En effet, si le montage est à la vue et à la disposition de tous, il sollicite l'intervention d'autres étudiants, renvoie à tous et simultanément une rétroaction directe et sensorielle consécutive aux actions posées. Cela favorisant ainsi, par la prise en compte d'un objectif commun et d'un résultat facilement identifiable, la collaboration.

3.4.2.3 Effets et méfaits du projet

L'idée d'incorporer un projet dans la méthode d'enseignement est loin d'être nouvelle. On reconnaît au projet, certaines vertus, comme on déplore certains mythes ou inconvénients.

Le projet est idéal pour manipuler un ensemble de concepts et dans certains cas à agir concrètement sur la matière. Dans la majorité des cas, il permet à l'étudiant de quitter le mode passif pour passer au mode actif. Il est habituellement un lieu de créativité et d'échanges qui devra acheminer l'apprenant vers l'autonomie et la maîtrise des concepts.

Jean Pierre Boutinet (1993) répertorie, pour sa part, les effets émancipateurs et les méfaits du projet. Le tableau 3.12 est tiré de sa publication dans *Sciences Humaines no 39 mai 1993*. Voici quelques méfaits qu'il a observés lors de l'élaboration d'un projet. La difficulté de se concentrer sur le moment présent ou l'étape actuelle étant donné

l'anticipation des étapes subséquentes. L'attrait des résultats aux dépens de la procédure à suivre. La désillusion provoquée par l'idéalisation de l'objet visé aux dépens de l'objet réel peut se traduire par une démotivation. Les contraintes académiques, par exemple l'unicité des spécifications et l'objectivité de l'évaluation risque d'éliminer toute créativité. Les coûts engendrés par l'achat de nouveaux matériels peuvent apparaître injustifiés en regard des bénéfices du projet.

Tableau 3.12 Effets émancipateurs et méfaits du projet (tirée de *Sciences Humaines no 39 mai 1993)*

Effets émancipateurs	Méfaits
• le projet doit tenir compte de données contradictoires, ce qui amène à gérer l'incertitude et la complexité des situations. • le projet incite les personnes concernées à développer leurs capacités, à devenir des acteurs-auteurs. • le projet conduit l'acteur à penser la situation en termes d'innovation et de création. • le projet se fonde sur une logique de l'interaction et de la négociation; en ce sens, il s'inscrit dans une logique de recomposition du lien social. • le projet favorise une action plus efficace grâce à un temps d'anticipation et de conception. • le projet permet d'expliciter les intentions et donc de clarifier la pensée en l'aidant à concrétiser les possibilités qu'elle recèle. • le projet conduit à se poser des problèmes de sens au regard de l'action à entreprendre: sens-direction à prendre, sens signification à dégager, sens-sensation à manifester. Ces questions favorisent le confort existentiel de l'acteur lui-même.	• la fuite en avant dans le futur provoque une certaine incapacité à gérer les exigences du moment présent. • par le volontarisme soucieux d'une maîtrise totale sur les choses et les individus, l'acteur s'illusionne sur ses capacités et ne tient pas compte des effets pervers éventuels. L'objet à conquérir prend le pas sur le processus à conduire. • l'idéalisation de l'objet visé, transformé en "bon objet", risque d'engendrer une désillusion à l'épreuve des faits. • la dénégation conduit les institutions et professionnels à se substituer aux individus, transformant ainsi leur projet de en projet pour. • le leurre risque de créer un fossé entre l'intention originelle, ambitieuse, et la concrétisation, très modeste. • les coûts risquent d'être disproportionnés au regard des avantages que procure le projet. • la lourdeur des dispositifs du projet, étouffant la créativité.

Perrenoud (1998) présente la problématique de la pédagogie de projet sous la forme de dilemmes :

> *Réussir ou comprendre ? Mieux vaudrait réussir et comprendre, mais justement, une démarche de projet oblige à un exercice acrobatique d'équilibre entre deux logiques : le projet n'est pas une fin en soi, c'est un détour pour confronter les élèves à des obstacles et provoquer des situations d'apprentissage. En même temps, s'il devient un vrai projet, sa réussite devient un enjeu fort, et tous les acteurs, maîtres et élèves, sont tentés de viser l'efficacité, parfois au détriment des occasions d'apprendre. [...] La logique d'une représentation réussie contredit la logique de formation, pour une raison assez évidente : pour apprendre, il faut que chacun soit mobilisé dans sa zone de proche développement, zone où, par définition, il peut apprendre, mais n'a pas déjà appris, zone où il hésite, va lentement, revient sur ses pas, commet des erreurs, demande de l'aide. Lorsque la réussite de l'entreprise est en jeu, c'est prendre un risque que de confier des tâches à ceux qui ne les maîtrisent pas. Du coup, on les prive de la possibilité, selon la formule de Meirieu, " d'apprendre, en le faisant, à faire ce qu'ils ne savent pas faire ".*

Pour Perrenoud, ce dilemme et tous ceux découlant se traduisent par un conflit entre un projet de formation et un projet d'action. Face à cette contradiction, il suggère deux pistes de solution : 1) accepter la contradiction, la travailler, l'anticiper; 2) la faire partager aux étudiants, ne pas la considérer comme l'affaire de l'enseignant. Finalement, Perrenoud inscrit dans un tableau plusieurs exemples de questions classiques et proposent des réponses dans une logique d'action et dans une logique de formation. Celui-ci est reproduit au tableau 3.13. Les réponses proposées font ressortir clairement l'aspect distinctif de la logique d'action par rapport à la logique de formation. Dans le premier cas, la responsabilité première du projet incombe à l'enseignant tandis que dans le deuxième cas, la responsabilité est partagé entre les étudiants et l'enseignant.

Tableau 3.13 Questions classiques dans une démarche de projet et leurs réponses dans une logique d'action et une logique de formation (tirée de Perrenoud 1998)

	Réponses dans une logique d'action	**Réponses dans une logique de formation**
Qui prend l'initiative ?	L'enseignant, si rien ne se passe spontanément ou pas dans le sens des apprentissages visés.	Les élèves, un vrai projet part d'eux
Qui exerce le leadership ?	Il est exercé "spontanément" par l'enseignant et les élèves qui en ont déjà le goût et les moyens.	On encourage à le prendre, au moins par moments, les élèves qui ont besoin de s'affirmer et de construire des compétences.
Qui fait quoi ?	La division du travail vise à utiliser au mieux les compétences existantes.	La division du travail vise à placer chacun dans sa zone de proche développement.
Que faire lorsqu'on rencontre un obstacle cognitif ?	On le contourne, autant que possible, pour ne pas perdre de temps.	Il est bienvenu et on prend le temps de l'affronter.
Que faire lorsque la confrontation à un obstacle exige des concepts ou des connaissances difficiles à construire sur le vif ?	On se débrouille avec les moyens du bord pour ne pas ralentir l'avancement et tenir le calendrier.	On suspend l'avancement du projet, on se forme pour revenir au projet avec de meilleurs outils.
Que faire lorsque l'évolution du projet marginalise certains élèves ?	On le regrette, on fait un geste symbolique, mais le souci d'avancer écarte toute véritable solution.	On s'arrête pour analyser la situation et on propose des aménagements redonnant une place à ces élèves.
Que faire en cas de conflit sur les options à prendre ?	On cherche à dégager une majorité, on vote et on continue.	On cherche un compromis, sans perdants ni gagnants, pour n'exclure personne.
Que faire si la dynamique s'essouffle, si une partie de la classe décroche ?	Ceux qui y croient prennent en charge l'ensemble des tâches, sous le regard indifférent ou ironique des autres.	On renégocie avec l'ensemble, et le cas échéant, on renonce à poursuivre ou on redéfinit le projet.
Que faire si l'évolution du projet éloigne des objectifs d'apprentissage initiaux ?	On se dit qu'il faut continuer à tout prix, que la réussite prime sur les acquis.	On rappelle à une contrainte didactique, on aménage le projet en conséquence.
Que fait l'enseignant ?	Il est au centre de la démarche, tout s'organise autour de lui.	Il observe, conseille, joue le rôle de médiateur ou de personne ressource.
Que se passe-t-il si le produit final n'est pas à la hauteur des attentes présumées des destinataires ?	L'enseignant passe des heures à colmater les brèches, corriger les textes, suppléer aux manques.	Le groupe assume le risque ou met les bouchées doubles pour achever et parfaire le travail.
Comment vit-on les problèmes rencontrés ?	Comme des obstacles dont on ferait volontiers l'économie.	Comme des occasions bienvenues d'apprendre.
Quel type de bilan fait-on à la fin ?	On évalue la réussite, l'accueil du public, la satisfaction des acteurs.	On analyse la démarche, on cherche à expliciter ce que chacun a appris.

3.4.3 Intégrer la pédagogie de projet à son enseignement

La venue des nouvelles technologies dans le monde de l'enseignement universitaire a considérablement modifié les moyens et les techniques utilisés pour transmettre les contenus. Reconnus plutôt conservateurs, classiques ou récalcitrants, les professeurs universitaires ont dû remettre en question leur méthode d'enseignement pour s'approprier les nouvelles technologies informatiques. La réussite scolaire est au premier plan, il faut accroître la motivation et l'autonomie de l'étudiant. Nous sommes à l'heure de l'innovation pédagogique, de l'informatisation de l'apprentissage, à la recherche de méthodes d'apprentissage rendant l'étudiant actif vis à vis l'objet d'apprentissage.

Inspiré d'un article intitulé: « Transformation: from teacher-centered to student-centered engineering education » de George D. Catalano et Karen Catalano paru dans Journal of Engineering Education en janvier 1999, nous présentons un modèle d'apprentissage dans lequel nous désirons inscrire la pédagogie de projet.

3.4.3.1 À la recherche d'un enseignement renouvelé

Nous désirons offrir à l'étudiant des moyens lui permettant un apprentissage actif. Le modèle de l'apprentissage actif est dans l'air depuis au moins une décennie et se concrétise de plus en plus avec l'avènement des technologies de l'information. Sous la pression de la communauté, l'enseignant doit quitter le cadre sécuritaire et contrôlé de la classe d'étudiants assis à s'abreuver de ses connaissances afin de déléguer la responsabilité de l'apprentissage à ses étudiants. Catalano et Catalano (1999) soulignent selon les psychologues cogniticiens et les éducateurs que l'enseignement est plus efficace lorsque les étudiants sont encouragés et s'attendent même à devenir impliqués dans leur propre apprentissage, en changeant le centre du modèle de ce que l'enseignant fait par ce que l'étudiant fait. Pour sa part, King (1994) affirme que la clé du processus d'apprentissage est de formuler un problème motivant et signifiant pour placer ses étudiants dans une démarche active de résolution de problème. Cette voie s'avère

salutaire lorsque le concept ne peut pas être décrit précisément mais traité à l'aide d'exemples et d'analogies (Catalano et Catalano 1999).

En enseignement de l'ingénierie, Felder (1997) a réalisé des travaux sur l'apprentissage actif et a répertorié six principes que l'enseignant doit respecter:

- écrire clairement les objectifs visés par un travail;
- spécifier les stratégies d'apprentissage et les habiletés utiles pour les étudiants;
- maximiser l'apprentissage résultant de l'expérience et minimiser les séances de cours;
- utiliser grandement l'apprentissage par équipe;
- éviter que la vitesse soit un facteur lors des tests;
- renforcer positivement la réussite.

À notre avis, la réalisation d'un projet est une activité qui favorisera principalement l'atteinte des principes trois et quatre. D'un autre côté, il est intéressant de noter que les principes de Felder peuvent s'inscrire dans trois phases. La première, la planification, regroupe l'écriture d'objectifs et la modélisation de stratégies. La deuxième, la réalisation, concerne l'expérimentation et le recours à la théorie. La troisième, la validation, regroupe l'utilisation de test et la rétroaction positive. L'enseignant ingénieur sera des plus confortables avec ces trois phases.

Catalano et Catalano (1999) renchérissent en énumérant sept tâches que le professeur doit réaliser dans un enseignement dirigé vers l'étudiant:

- modéliser les habiletés de raisonnement et de traitement;
- connaître le niveau cognitif actuel et désiré de l'activité et des étudiants;
- développer des questions qui facilitent l'exploration et l'avancement de l'étudiant;
- utiliser des outils visuels pour établir des connections et encourager leur appropriation par l'étudiant;

- fournir des moyens d'apprentissage de groupe;
- utiliser l'analogie et la métaphore;
- fournir un mécanisme de dialogue indirect entre le professeur et les étudiants.

Nous reprenons ces différents points afin de bien comprendre la tâche qui attend le professeur.

Modéliser les habiletés de raisonnement et de traitement

Pour orienter son enseignement vers l'étudiant, le professeur doit fournir un modèle exploitant le « remue-méninges » ou la résolution de problème. Le professeur doit s'efforcer d'étaler sa pensée en l'extériorisant principalement par des interrogations adressées à l'étudiant. Plutôt qu'entendre des monologues, l'étudiant est mieux servi par de fréquentes et signifiantes prises de conscience avec la possibilité d'interaction entre le professeur et l'étudiant.

Connaître le niveau cognitif actuel et désiré de l'activité et des étudiants

Deux modèles de pensée peuvent être utilisés pour estimer le niveau cognitif: la taxonomie de Bloom et la structure de l'intellect de Guilford. Bloom (1956) décrit les niveaux d'apprentissage de la mémorisation (bas niveau) jusqu'à la synthèse et l'évaluation (haut niveau). La structure de Guilford, tel que décrite par Aschner et Gallagher (1965) divise la pensée entre mémorisation ou simple rappel, pensée convergente laquelle requiert l'utilisation de données pour atteindre la réponse, et pensée divergente laquelle nécessite la génération d'alternatives et d'évaluation exigeant du jugement et la prise de décision.

Il peut être pertinent de présenter une adaptation de ces échelles. L'étudiant pourra de cette façon établir le niveau d'apprentissage requis et pourrait, dans le meilleur des cas, identifier les causes de la difficulté rencontrée. Voici une adaptation de la taxonomie de Bloom proposée par Catalano et Catalano (1999):

1 Reconnaissance
2 Mémorisation
3 Interprétation
4 Réalisation des connections
5 Résolution de problèmes
6 Destruction des barrières
7 Assemblage des pièces
8 Déduction des conclusions
9 Évaluation du pour et du contre

Il ne faut surtout pas perdre de vue que l'objectif des différentes activités d'évaluation, examen, travail pratique, projet, est de permettre à l'étudiant de jauger en interaction avec les autres son niveau d'apprentissage à un instant précis.

Développer des questions qui faciliteront l'exploration et l'avancement de l'étudiant

Des techniques d'interrogation et leurs importances ont été étudiées par Hansen (1994), Dantonio (1990), Taba (1978) et Ehrenburg et Ehrenburg (1978). De ces études Catalano et Catalano déduisent quatre catégories de questions. La première catégorie se compose de questions qui permettent de recueillir de l'information. La deuxième catégorie contient les questions qui permettent de classer l'information. La troisième catégorie concerne les questions qui permettent d'organiser l'information. La dernière catégorie regroupe les questions qui permettent d'interpréter l'information.

Une fois les catégories bien définies et bien distinctes, une façon de procéder serait d'inciter les étudiant à composer en collaboration des questions

appartenant à chacune de ces catégories. Cette activité peut s'avérer très formatrice autant pour le professeur que pour les étudiants.

Utiliser des outils visuels pour établir des connections et encourager leur appropriation par l'étudiant

La recherche sur le cerveau suggère que l'hémisphère gauche sert à la pensée linéaire et analytique tandis que l'hémisphère droit sert à la pensée spatiale et intégrative. L'enseignement centré sur le professeur semble négliger un hémisphère. Pour remédier à cette lacune, l'enseignement centré étudiant peut proposer un outil visuel de représentation de la connaissance. Cet outil servirait alors de langage de codage connu facilitant la communication et la collaboration. Par exemple, sous forme de tableau ou d'arbre, l'étudiant doit s'efforcer à trier et classifier ses connaissances en identifiant les sujets et les liens.

Fournir des moyens d'apprentissage de groupe

Catalano et Catalano (1998) citent deux activités efficaces pour l'enseignement en génie: la composition d'équipes de deux ou trois personnes pour travailler sur des problèmes et présenter leur résultat au reste de la classe; organiser des séances d'échange où un étudiant expose son problème et reçoit les suggestions des autres, le tout supervisé par le professeur.

Utiliser l'analogie et la métaphore

Pour éviter d'être redondant nous vous référons à notre propos présenté à la section 3.2.

Pour renouveler l'enseignement, il faudra rendre les notions plus concrètes en les inscrivant dans une activité signifiante et collaborative. La pédagogie de projet bouleverse les habitudes des professeurs et des étudiants. Les relations au sein du triangle didactique « professeur - étudiants – savoir » seront profondément modifiées. Le professeur qui décide de proposer un projet à ses étudiants doit éviter toute improvisation et doit être conscient des forces et faiblesses de cette approche puisqu'il ne s'adresse plus à une seul étudiant mais à un groupe qui doit collaborer pour accéder aux savoirs faire et à la connaissance.

Nous avons situé la pédagogie de projet ainsi que l'outil dont nous envisageons l'utilisation selon l'acte d'enseignement. Pour nous, l'un ne va pas sans l'autre. En effet, les projets consisteront essentiellement à concrétiser la théorie en l'appliquant à notre outil. L'utilisation de la pédagogie de projet n'exclut pas la présentation de matière théorique ou d'autres activités connexes. En effet, cette technique nécessite un incessant va et vient entre une tâche originale, le « projet », et des activités qui permettent aux étudiants de se questionner par rapport à cette tâche, d'étendre et de structurer leurs connaissances et de stabiliser des savoirs et savoir-faire.

Afin d'optimiser notre approche, nous désirons inscrire la pédagogie de projet dans une philosophie d'enseignement centré sur l'étudiant ou disons plutôt ici sur les étudiants. Nous avons décrit sept tâches proposées par Catalano et Catalano (1999) que le professeur doit réaliser dans un enseignement dirigé vers l'étudiant qui favoriseront, voire optimiseront à notre avis, l'appropriation des savoirs par l'étudiant en orientant ces tâches vers un projet concret et signifiant. Au contraire de Perrenoud, nous ne distinguerons pas la pédagogie de projet de l'action, nous l'insérons directement dans l'action.

La didactique s'intéresse spécifiquement aux rapports entretenus par l'apprenant sur les savoirs de référence. Puisque nous recherchons à améliorer la qualité des différents rapports à ce savoir à apprendre, nous présentons à la section suivante quelques notions intéressantes de la didactique qui pourront assurément nous être utiles.

3.5 Autres considérations d'ordre didactique

> *« Dans un contexte scolaire, la connaissance de l'étudiant se développe dans le temps à travers une série d'interactions adaptatives avec les situations préparées, en connaissance de cause, pour lui, par l'enseignant. Au départ, l'étudiant maîtrise peu ou pas du tout, ces situations. Par la suite, ces dernières sont de plus en plus sous son propre contrôle. [...] Le premier temps est court, il correspond à la relation didactique. Lors du premier temps, le temps court, l'étudiant a un faible rapport au savoir. Il s'agit du moment de tous les risques, celui d'une évolution plus ou moins rapide des pratiques et des conceptions ou, au contraire, celui des blocages plus ou moins durables en face de cette situation parfois difficilement maîtrisée par l'étudiant.[...] Le temps long s'étale sur des années, il est celui de la psychogenèse de l'acquisition de la connaissance qui se développe bien au-delà de la relation didactique. » (Raisky et Caillot, 1996)*

3.5.1 Relation didactique

Une relation didactique s'articule autour d'un savoir de référence et des rapports entretenus par l'enseignant et les étudiants sur ce savoir. Ces rapports évoluent, se modifient au cours de la relation didactique. Raisky et Caillot (1996) précisent que

> *«Le premier rapport au savoir auquel l'élève est nécessairement confronté lorsqu'il entre dans la dynamique d'une relation didactique, c'est d'abord et avant tout le sien. L'enseignant ne devrait pas pouvoir imaginer, d'entrée de jeu, imposer à tous ses élèves un rapport unique à l'objet d'étude qu'il se propose d'envisager durant une séquence d'enseignement et d'apprentissage. Il n'y pas, d'une part, un « élève-type » et d'autre part, une « matière scolaire » univoque. Il n'existe pas une « discipline à enseigner » unique et universelle vers laquelle il faudrait, de l'extérieur, diriger l'élève, tout élève, pour qu'il n'existe qu'une façon de connaître ce savoir. »*

Pour sa part, l'enseignement de la discipline du génie, s'inspire de la théorie constructiviste en misant sur la résolution de problèmes pour l'acquisition de nouvelles connaissances. Cependant l'éloignement entre la théorie et la pratique demeure marqué dans la formation des ingénieurs qui procède beaucoup par la résolution de problèmes techniques concrets en s'appuyant sur les théories plutôt que d'amener l'étudiant à reconstruire la théorie à partir de ces problèmes. Dans cet esprit, Larochelle et Bednarz (1994 :5) écrivent :

> *« De façon générale, considérer le savoir des étudiants et des étudiantes, comme le promeut la thèse constructiviste, ne semble guère avoir modifié le protocole d'enseignement habituel, quel que soit l'ordre d'enseignement en cause. Certes, le point de vue de l'étudiant ou de l'étudiante sera davantage sollicité (c'est d'ailleurs là l'effet majeur actuel du « constructivisme » sur la pédagogie). Mais, le plus souvent, tout se passe comme si cette sollicitation n'avait d'autre finalité que de repérer « ce qui ne va pas » dans ce point de vue, et ce, bien sûr, en référence au savoir à enseigner, sans égard pour la nature et la portée, éventuellement fort distinctes, qui caractérisent celui-ci en regard du savoir développé par l'élève. Dans cette perspective, ce n'est donc pas la complexification du savoir de l'élève qui prime mais plutôt l'amenuisement de l'écart entre ce qu'il sait et le savoir à transmettre. »*

La principale leçon que nous tirons des remarques de Larochelle et Bednarz est de permettre à l'étudiant de construire son propre rapport au savoir. Surtout que Jonnaert et Laveault (1994) nous rappellent que *« le résultat de ces rapports privés et individuels au savoir est l'existence, dans une même classe, d'un grand nombre d'écarts entre les rapports que les élèves entretiennent au savoir et celui, souvent unique, que l'enseignant veut imposer à chacun d'entre eux. »*

3.5.2 Contrat didactique

Lors de la relation didactique, l'enseignant et l'étudiant exercent tour à tour différents rôles. Le concept de contrat didactique, décrit par Brousseau (1986), permet de comprendre le dynamisme de la relation didactique. L'ensemble des rapports établis par

l'enseignant et l'étudiant autour du savoir détermine les ruptures et les changements de rôles successifs à l'intérieur de la relation didactique. Ces rôles se définissent selon certaines règles implicites ou explicites. Par exemple, en jouant la règle de la dévolution, l'enseignant propose à l'étudiant d'utiliser sa propre démarche d'apprentissage (Brousseau, 1983,1986). Pour qu'il y ait dévolution, avec effet sur l'apprentissage, il faut que l'élève en ait préalablement accepté le principe (Margolinas, 1993). Si l'étudiant est dans l'impossibilité de compléter l'activité proposée par l'enseignant, ne pouvant plus avancer dans sa démarche, il doit alors demander à l'enseignant de reprendre ses fonctions en appliquant une autre règle. Dans ce cas, on dira que l'étudiant exerce son droit de « contre-dévoluer » (Brousseau, 1986). À l'extrême, il y aura rupture didactique quand l'étudiant n'est plus certain que l'enseignant est en mesure de le guider vers une bonne démarche de ses apprentissages scolaires. L'étudiant a perdu confiance en son enseignant qu'il ne croit plus capable de lui venir en aide dans les problèmes à résoudre (Brousseau, 1986). Le rôle de chacun, étudiants et enseignant, est fondamental dans l'organisation de l'évolution des savoirs à l'intérieur de la relation didactique (Raisky et Caillot, 1996).

C'est en considérant cette dynamique où les intervenants s'échangent des rôles selon certaines règles que se construit un contrat didactique. Brousseau définit le contrat didactique comme suit :

> *« il s'agit d'une relation qui détermine, explicitement pour une petite part, mais surtout implicitement, ce que chaque partenaire, l'enseignant et l'enseigné, a la responsabilité de gérer et dont il sera responsable, d'une manière ou d'une autre, devant l'autre. Ce système d'obligations réciproques ressemble à un contrat. Ce qui nous intéresse est le contrat didactique, c'est-à-dire la part de ce contrat qui est spécifique au contenu. » (Brousseau, 1986 :51)*

Raisky et Caillot, précisent à notre avis, très justement le véritable enjeu du contrat didactique en ces termes :

> *« Ainsi, un véritable contrat didactique sera composé de règles, implicites et explicites. Parmi ces règles, l'une d'entre elles acceptera la dévolution (versus la contre-dévolution); ce contrat*

> *fonctionnera parce que l'élève a effectivement le projet d'apprendre ce que l'enseignant souhaite lui faire apprendre. Il ne suffit donc pas de dire qu'une relation didactique existe parce que quelqu'un a le projet d'enseigner quelque chose à quelqu'un d'autre. C'est une approche unilatérale. Pour que cette relation didactique fonctionne optimalement, il faudra que le projet d'enseigner rencontre un projet d'apprendre. »*

Les mêmes auteurs soulignent l'importance de l'interaction en terme d'un espace de dialogue.

> *« Un contrat didactique [...] crée un espace de dialogue entre ces derniers (élève-enseignant-savoir) tout en respectant chacun d'entre eux. La fonction d'un contrat didactique n'est pas de transformer tout l'implicite en explicite, mais bien d'équilibrer les deux afin de créer une zone d'échanges entre les partenaires : un espace de dialogue. En ce sens, le contrat didactique ne peut se vivre qu'au sein d'une relation didactique, à l'intérieur même de la classe. » (Raisky et Caillot, 1996)*

3.5.3 Types de situation didactique

Brousseau (1986) traduit l'évolution de la relation didactique selon trois types de situation didactique : la situation didactique, la situation a-didactique et la situation non didactique. Dans la situation didactique initiale, l'enseignant a une relation privilégiée au savoir par rapport à l'étudiant qui a une relation inexistante ou inadéquate. Une situation est dite a-didactique lorsque l'étudiant peut utiliser adéquatement ses acquis pour la traiter en dehors de toute intention d'enseigner de l'enseignant. *« L'état a-didactique constitue un état intermédiaire où le maître est présent, mais dans lequel l'élève agit de son propre mouvement. »* (Margolinas, 1993 :229). Dans une situation non didactique, la relation de l'étudiant au savoir est indépendante de la relation de l'enseignant au savoir. Ce type de situation, a-didactique, n'est pas organisé dans le but d'apprentissage précis. Dans le cadre des activités réalisées en classe, en fonction du degré de contrôle exercé par l'enseignant, l'étudiant vit des situations didactiques et a-didactiques.

La réalisation des exercices de programmation sur le montage robotisé placera l'étudiant en situation a-didactique, le professeur étant présent en classe et l'étudiant exerçant sa propre démarche pour concevoir et réaliser un programme afin de résoudre le problème qui lui est soumis.

3.5.4 EIAH et situations didactiques

Les situations didactiques occupent une place de plus en plus importante dans le domaine. Ainsi Brousseau (1998) propose que, dans la didactique moderne, l'enseignement soit la dévolution à l'étudiant d'une situation adidactique et que l'apprentissage soit une adaptation à cette situation. Il ajoute: « *une des hypothèses de la didactique consiste à affirmer que seule l'étude globale des situations qui président au manifestations d'un savoir, permet de choisir et d'articuler les connaissances d'origines différentes, nécessaires pour comprendre les activités cognitives du sujet, ainsi que la connaissance qu'il utilise et la façon dont il la modifie* ». À cette hypothèse, il en ajoute une deuxième, plus forte, qui affirme que l'étude première des situations didactiques devrait finalement permettre de dériver ou de modifier les concepts nécessaires actuellement importés des autres champs scientifiques.

La théorie des situations didactique, développée par Guy Brousseau (1998), est construite autour du problème de l'organisation des interactions entre l'étudiant et un *milieu* spécifique permettant l'apprentissage d'une connaissance déterminée. Selon cette théorie, l'apprentissage est le résultat d'un processus d'adaptation de l'apprenant à son environnement, lequel environnement peut être façonné en un milieu permettant des apprentissages spécifiques. Ainsi, la connaissance est caractérisée par un certain état d'équilibre du système sujet-milieu en interaction.

Les EIAH, dans une problématique didactique, se présentent comme le milieu, ou encore l'emplacement d'une réalisation effective d'un milieu propice à un apprentissage donné. Selon Balacheff (1999) : « *la notion de milieu étant un construit théorique, son existence effective dépend d'une part des caractéristiques de l'environnement matériel*

dans lequel on veut l'implanter, et d'autre part des possibilités de l'apprenant de le "reconnaître" ».

L'interaction entre l'apprenant et le milieu s'exécute généralement sous des contraintes sur lesquelles il est possible de jouer pour obtenir des effets didactiques, c'est à dire des apprentissages ayant des caractéristiques particulières. Par exemple, on pourrait différer l'utilisation de l'ordinateur, ou définir des rôles de concepteur et d'opérateur relativement à un problème, pour provoquer l'apparition de formulations ou de processus de preuve nécessaires à l'apprentissage.

Dans sa théorie des situations didactiques, Brousseau décrit trois situations particulières: l'action, la formulation et la validation. L'étudiant est en situation d'action lorsqu'il prend une décision et agit sur la situation. La suite des situations d'actions constitue le processus par lequel l'étudiant va initier des stratégies, les mettre à l'essai, s'approprier des savoirs et savoir faire dans un environnement de résolution de problème. Pour notre cas de EIAH, l'étudiant sera en situation d'action lors d'exécution d'instructions de programmation sur le montage. L'étudiant sera en situation de formulation lorsqu'il exprimera une stratégie de solution dans un langage compris par tous. Le langage doit prendre en compte les objets et les relations pertinentes de la situation de façon adéquate. Ici, l'étudiant sera en situation de formulation en proposant des instructions ou une structure de programmation particulière. L'étudiant est en situation de validation lorsqu'une solution proposée est testée concrètement ou est jugée par ses pairs. Il peut y avoir une acceptation ou un refus sous forme de rétroaction. La situation didactique doit permettre aux étudiants d'évoluer, de réviser leur opinion, de remplacer leur théorie fausse par une théorie vraie. Dans le cas de notre EIAH, la validation s'effectuera par l'observation du comportement du montage comme elle pourra se faire au cours d'une discussion entre les étudiants. Finalement, dans l'institutionnalisation, l'enseignant définit les rapports que peuvent avoir les comportements ou les productions « libres » de l'étudiant avec le savoir et avec le projet didactique. Le tutoriel pourra jouer ce rôle dans notre EIAH. Ces différents éléments en interaction sont présentés à la figure 3.20 qui est une adaptation tirée à partir d'une figure du volume de Brousseau (1998).

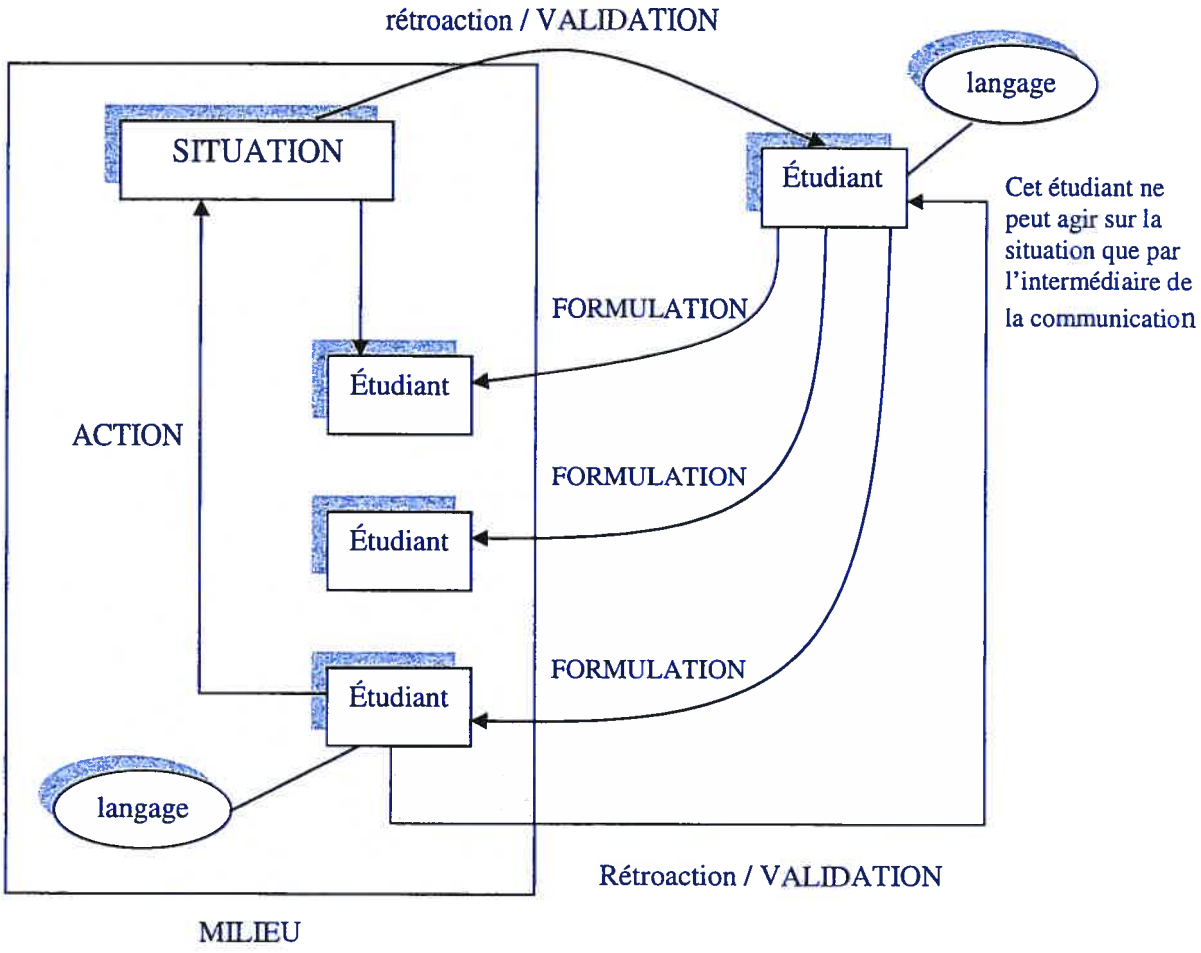


Figure 3.20 Situation de l'action, situation de la formulation, situation de la validation (Brousseau 1998 p. 35)

D'autre part, Balacheff (1999), précise qu'un EIAH constitue une instance très particulière d'un milieu au sens didactique car il crée par sa seule existence trois univers distincts qui sont autant de lieux de représentation de la connaissance :

- *L'**univers externe** dans lequel se trouve l'utilisateur et dans lequel il a accès, non seulement au dispositif informatique, mais encore à d'autres moyens de représentation de la connaissance qu'il peut mettre en relation avec les moyens que la technologie offre. Ainsi, outre les fonctions de dessin d'un cercle à l'écran de l'ordinateur, l'utilisateur peut recourir à un compas et à un support papier, et*

confronter ainsi différents outils de représentation en relation avec un même objet.

- *L'***interface** *qui est le lieu où sont matérialisées les interactions entre l'utilisateur et le dispositif informatique. L'interface est le média dans les contraintes duquel sont représentées les connaissances, il s'agit actuellement le plus souvent du dispositif écran-souris-clavier.* Nous avons ajouté ici une interface particulière qui réalise l'interaction entre un objet robotisé et l'ordinateur comme le contrôle de procédés (voir fig. 1.1 p. 13).
- *L'***univers interne** *dans lequel est implanté l'EIAH. Cet univers est d'une grande complexité, malgré sa petite taille apparente. Pour le comprendre il est intéressant d'imaginer qu'il est constitué de plusieurs strates : strate physique, strate électronique, strate logique, strate machine (combinaison des constituants logiques élémentaires), strate d'assemblage (ensemble des commandes de la machine sur la strate précédente), strate symbolique (celle des langages de programmation). La connaissance a effectivement une représentation dans ces diverses strates.*

Les relations entre ces univers sont complexes. Nécessairement, l'interface et l'univers interne se contraignent mutuellement et interagissent fortement. De plus, le développement d'un EIAH doit prioritairement se concentrer sur les besoins et les caractéristiques des utilisateurs (user driven) sans négliger cette forte interaction entre caractéristiques de l'univers externe et caractéristiques de l'univers interne.

La problématique informatique des EIAH s'organise autour de deux pôles principaux: les connaissances et les utilisateurs. Les connaissances ou ici les savoirs incluant savoir faire et savoirs être sont ceux qu'il faut représenter et acquérir puisque le principal enjeu est leur utilisation. Les utilisateurs enseignants et apprenants ont la particularité d'évoluer du point de vue de leurs connaissances lors de l'interaction avec l'environnement informatique.

Ballachef (1999) affirme que les principales compétences sur lesquelles s'appuiera la conception d'un EIAH seront la résolution de problèmes, l'explication et l'apprentissage. On ne trouvera pas nécessairement toutes ces compétences réunies en même temps dans un même logiciel, mais toujours au moins l'une d'entre elles, même sous une forme limitée.

3.5.4.1 Conception d'un EIAH

La conception des EIAH se divise en deux grandes catégories. La première catégorie se compose des environnements construits selon un schéma de la communication pour lesquels la connaissance est le contenu intentionnel d'un échange de dialogues. Les logiciels tuteurs offrant des dialogues avec l'apprenant où l'enseignant peut jouer un rôle explicite sont de cette catégorie. La deuxième catégorie se compose des environnements construits selon un schéma de l'action pour lesquels la connaissance est construite dans le cours de la résolution d'un problème. Les logiciels de micromondes, permettant une exploration libre et approfondie où l'enseignant est volontairement ou temporairement absent, sont de cette catégorie. Ces deux catégories se distinguent ainsi par le degré d'autonomie ou de liberté de l'apprenant.

Qu'il s'agisse d'une catégorie ou l'autre, la conception, la réalisation et les usages d'un EIAH ne peuvent se réaliser sans rencontrer inévitablement certains problèmes. Ces problèmes ciblent spécifiquement les constituants d'un EIAH dans l'architecture classique: modélisation des objets d'enseignement (les informaticiens parlent ici de modélisation de la connaissance), prise en compte de l'apprenant et de l'enseignant, modélisation des décisions didactiques. Dans un vocabulaire classique pour décrire l'architecture des systèmes, on parlera de module expert, module apprenant et module pédagogique. Balacheff (1999) présente à l'aide de trois postulats, ces différents problèmes :

> *Postulat 1: « **les problèmes sont les critères et la source des connaissances** », cela signifie que lorsque nous aborderons un EIAH*

une question que nous nous poserons est celle de savoir de quelle façon il engage la connaissance, quelle signification il permet de construire comme propriété de l'interaction entre la machine et l'apprenant.

*Postulat 2: « **l'apprentissage est le résultat d'un processus d'adaptation et de construction** », il s'agit là d'une formulation de ce que l'on appelle aussi l'hypothèse constructiviste. Ce postulat affirme que l'apprentissage n'est pas que le résultat d'un processus de mémorisation, mais qu'il implique une activité complexe de construction du coté de l'apprenant qui peut conduire à une réorganisation de la connaissance qu'il possède déjà, voire à une "révolution" lors du passage d'une conception à une autre lié à une notion particulière.*

*Postulat 3: « **toute représentation transforme le représenté** », ce postulat est le plus souvent ignoré tant est fort le désir de réduire la distance entre le monde "réel" et les mondes "artificiels". C'est pourtant un résultat classique de la sémiotique, science des signes et de leurs usages et de leurs significations. Nous verrons en étudiant la transposition informatique la portée de ce postulat et son importance dans une problématique d'apprentissage.*

La conception d'un EIAH, selon Balacheff (1999), peut être décrit par les étapes suivantes :

- choisir un domaine de connaissance de référence; ce domaine doit pouvoir être modélisé, représenté dans l'univers externe;
- produire un modèle dans une formalisation informatisable, c'est à dire susceptible d'être accessible via un dispositif informatique; ce modèle doit fournir de façon détaillée et complète la description des entités constitutives du modèle de référence et de leurs relations;
- implémenter le modèle dans un dispositif informatique en considérant les contraintes du langage de programmation.

Le logiciel obtenu constitue alors le modèle formel implémenté dans l'univers interne. Cette implémentation prend en compte également les contraintes des interfaces et, encore plus important, leurs liens avec le savoir de référence à apprendre. Cette implémentation prend également en compte les contraintes d'interaction avec un utilisateur. Contraintes qui ne sont pas a priori de l'ordre des savoirs en jeu dans le modèle de référence. Enfin c'est dans l'univers externe, dans l'interaction avec l'interface que l'utilisateur (apprenant ou non) élaborera une conception des savoirs en jeu.

L'implémentation d'un modèle de référence dans un dispositif informatique correspond à un processus de transposition semblable à celui de la transposition didactique (Chevallard 1985). Pour le cas spécifique d'un EIAH, Balacheff (1999) parle de transposition informatique. Il précise trois difficultés liées à cette transposition. Premièrement, les effets non voulus, qui sont dus en général à la complexité de l'implémentation ou aux spécifications de l'EIAH qui ne sont habituellement pas suffisantes pour prévoir complètement ses comportements. Deuxièmement, un problème de preuve permettant d'assurer que ce qui est produit est conforme aux résultats attendus pour tous les usages. On relève souvent cette problématique de preuve en génie logiciel. Troisièmement, il existe une étroite dépendance entre la représentation des données et le traitement de ces données, cette dépendance va notamment avoir un impact sur les résultats ou l'organisation de l'interface étant donné les actions de l'utilisateur.

Dans le cadre de nos travaux, un aspect que nous ne désirions pas négliger, était celui de la didactique. En effet, nous ne désirions pas concevoir un outil sans nous assurer qu'il réponde aux premières préoccupations de la didactique. Ainsi, une fois l'analyse didactique réalisée, nous pensons à la lumière des leçons tirées de ce chapitre, que l'environnement informatique pour l'apprentissage humain est d'un intérêt certain pour la didactique.

Nous avons exposé qu'il fallait permettre à l'étudiant de construire ses propres rapports au savoir même si cette approche va engendrer une plus grande diversité dans les rapports qu'entretiendront les étudiants avec le savoir puisque nous éclatons

l'enseignement uniforme des cours magistraux traditionnels. Nous avons précisé que le rôle, des étudiants et de l'enseignant dans la relation didactique, est fondamental dans l'organisation de l'évolution des savoirs à l'intérieur de celle-ci. Il nous faut, à partir de ces rôles, créer une zone d'échanges, de dialogues qui installera un climat d'apprentissage approprié. Nous avons également souligné la fonction de la situation didactique dans la relation didactique qui consiste à établir le degré du rapport avec le savoir entretenu par l'enseignant et l'étudiant. Ce degré de rapport s'échelonne de la possession exclusive du professeur jusqu'à l'appropriation progressive du savoir par l'apprenant à travers différents types de situation didactique.

En examinant la théorie des situations didactiques de Brousseau, nous avons déduit que les EIAH constituent un milieu propice pour engager l'étudiant dans une réalisation effective d'où devrait émerger sa compétence. Cet environnement permet la dévolution à l'étudiant d'une situation adidactique s'exprimant par la résolution d'un exercice de programmation sur un montage électronique robotisé. De plus, les composants de notre EIAH, le tutoriel et l'activité de programmation permettent l'établissement de rapports personnels aux savoirs. L'activité de programmation est effectuée en équipe tandis que le tutoriel favorise plutôt un apprentissage individuel. Le tutoriel sera l'outil idéal pour le débutant qui pourra prendre tout le temps nécessaire pour assimiler une notion alors que « le connaisseur » pourra à son gré valider la justesse de ses notions en explorer de nouvelles ou compléter les exercices les plus corsés. Les exercices de programmation permettront un échange soutenu entre les étudiants de différents niveaux. Cette activité nous incite à ajouter le groupe d'étudiants en réalisation d'exercices comme acteur d'impact au triangle didactique. En effet, le groupe favorisera une multitude d'échanges correspondant à autant de rapports différents avec le savoir. Les rôles jouées par chacun, étudiant, étudiants et professeur s'inscriront dans une dynamique des plus fertiles. Le tutoriel offre, de son côté, d'autres rapports au savoir. Les différentes rubriques disponibles pour une même notion consistent en différentes vues fournies à l'apprenant. L'encadrement, la rétroaction et la variation dans les niveaux de difficulté des exercices présentent plusieurs occasions de camper différents rôles dans la situation didactique et adidactique.

Les bénéfices didactiques de notre EIAH ayant été confrontés aux principaux modèles et théories en usage, nous allons poursuivre notre démarche de développement. Pour ce faire, nous présentons au prochain chapitre le modèle d'action de sorte à préciser et opérationnaliser notre idée et décrire comment l'étudiant devra interagir dans cet environnement informatique d'apprentissage.

Chapitre 4

MODÈLE D'ACTION

Le modèle d'action, à travers les choix techniques et les décisions que doit prendre le chercheur, précise l'idée en s'appuyant sur les considérations théoriques et l'opérationnalise. Dans cette optique, nous décrirons dans ce chapitre les interactions de l'apprenant avec l'environnement d'apprentissage. Nous aborderons les interventions proposées pour favoriser son apprentissage de la programmation sous l'angle de la didactique et plus particulièrement de la relation didactique. Nous préciserons les intentions didactiques découlant des anomalies relevées ou des conseils prodigués dans nos considérations. Du point de vue de l'opérationnalisation de l'idée, nous décrirons les différents prototypes matériels et logiciels que nous allons concevoir. Nous décrirons le déroulement des exercices de programmation avec le montage électronique robotisé ainsi que la mise à l'essai avec des experts du domaine avant de décrire notre EIAH.

4.1 Description de l'interaction attendue entre l'apprenant et l'environnement d'apprentissage

Notre environnement informatisé pour l'apprentissage humain (EIAH) sera constitué d'exercices de programmation exploitant un montage concret et un tutoriel présentant sous plusieurs formes les notions importantes. Ces composantes de l'EIAH devraient permettre de multiplier les rapports aux différents savoirs. Les exercices devraient offrir la possibilité à chaque étudiant de contribuer à la solution selon son niveau d'apprentissage et de compréhension du problème. Dans un esprit de travail collaboratif, un rapport plus fructueux devrait se produire lorsque l'étudiant confrontera sa solution à la collectivité, ici à son équipe ou lorsqu'il visualisera l'adéquation de l'algorithme implanté avec le comportement réel du montage. Cette relation didactique devrait favoriser les échanges entre étudiant-étudiants-savoir et étudiant- savoir.

Examinons maintenant notre EIAH sous l'angle du contrat didactique. La réalisation de l'atelier de programmation installe l'étudiant en situation de dévolution. Dans cette relation didactique, l'enseignant organise une activité où l'étudiant doit s'investir pleinement. L'enseignant fournit à l'étudiant, par l'entremise d'un montage électronique robotisé, un cadre favorisant sa propre démarche d'apprentissage et stimulant les dialogues entre étudiants. Ces dialogues sont importants puisqu'ils constitueront des interactions fertiles sur le savoir à acquérir. Le milieu, tel que l'entend Brousseau (1998), favorisera les situations d'action, de formulation et de validation.

Le tutoriel présente à l'étudiant un environnement d'apprentissage individualisé offrant une présentation hiérarchisée de notions, l'ancrage de notions à l'aide de l'analogie, la réalisation d'exercices et l'exécution de simulation. Ce composant de notre EIAH est principalement développé selon le schéma de la communication pour lesquels la connaissance est le contenu intentionnel d'un échange de dialogues (Balacheff, 1999). À travers toutes ces activités l'étudiant peut être constamment en dévolution, en plein contrôle de son apprentissage, comme il peut recourir à une certaine forme de contre-dévolution au besoin. Par exemple, lors de la réalisation d'exercices l'étudiant bénéficie d'une forme de guidage telle que le ferait un professeur et il pourra obtenir directement la solution lorsqu'il ne peut résoudre le problème.

La construction du tutoriel a comme principal objectif d'offrir aux étudiants un support à l'enseignement traditionnel en classe. Prolongement de cet enseignement, il constituera un milieu où l'étudiant pourra construire ses propres rapports au savoir.

Du point de vue didactique, il nous faut encore répondre à une question d'importance. Dans quels types de situation didactique les étudiants sont-ils plongés lors de l'utilisation de notre l'EIAH? À notre avis, la réalisation de l'atelier de programmation placera clairement l'étudiant en situation a-didactique, le professeur étant présent en classe et l'étudiant exerçant sa propre démarche de résolution. Le tutoriel, quant à lui, placera l'étudiant en situation didactique même si l'enseignant est concrètement absent.

Lorsqu'il utilise le tutoriel, l'étudiant peut compléter des exercices portant spécifiquement sur la syntaxe et la sémantique du langage de programmation. Pour résoudre ces exercices l'étudiant doit suivre une séquence d'opérations. Le tutoriel impose ainsi à l'étudiant un cheminement de résolution pensé par le professeur. La présentation des notions, des analogies et des simulations correspond à une médiatisation de l'information habituellement transmise par l'enseignant.

Nous venons d'identifier et qualifier les différentes interactions de l'étudiant avec l'EIAH selon différentes facettes de la didactique. Dans la suite de notre démarche, nous désirons ajouter des précisions sur le développement concret de l'environnement en fonction des considérations présentées. Ces précisions sont énoncées sous la forme d'intention.

4.2 Intentions didactiques

Nous désirons dans cette section préciser les intentions didactiques découlant des difficultés observées ou des recommandations issues de la prise en compte des différentes considérations. De cette façon, nous voulons montrer la consistance de notre travail et la légitimité de nos actions.

Attendu que l'apprentissage prend place dans la pensée de l'étudiant et nulle part ailleurs et que le rôle de l'enseignant réside à l'inciter à le faire, nous devons trouver des activités qui engageront l'étudiant dans l'organisation des informations, le développement de savoir-faire et l'acquisition de savoirs.

Considérant d'abord notre modèle générique d'enseignement, nous devons offrir de nouveaux moyens qui favoriseront un meilleur apprentissage. Notre première résolution consiste à proposer des moyens qui tiendront compte de la diversité de notre clientèle étudiante.

Intention 1. Offrir un enseignement adapté à l'ensemble de la clientèle étudiante.

Dans nos considérations d'ordre social nous avons relevé l'hétérogénéité de la clientèle s'exprimant par des connaissances initiales minimales jusqu'à des connaissances élevées de l'informatique. Il nous faut donc offrir un enseignement qui tiendra compte de cette hétérogénéité. Comme nous l'avons souligné la pédagogie par projets est assurément dans cette situation la plus prometteuse. Nous misons principalement sur les objectifs suivants de cette pédagogie cités par Perrenoud (1999):

> *impliquer un groupe dans une expérience «authentique», forte et commune, pour y revenir sur un mode réflexif et analytique et y ancrer des savoirs nouveaux; découvrir de nouveaux savoirs, de nouveaux mondes, dans une perspective de sensibilisation ou de «motivation»; permettre d'identifier des acquis et des manques dans une perspective d'auto évaluation et d'évaluation bilan; développer la collaboration et l'intelligence collective.*

Nous suggérons ici la pédagogie de projets. Cette pédagogie consiste à provoquer la collaboration d'un groupe d'étudiants sur un projet spécifique menant à une réalisation concrète. Ce projet est une activité offrant des défis qui obligeront l'acquisition de connaissances menant l'étudiant vers un apprentissage contextuel. Pour l'atteinte de ce but un choix est primordial, celui de l'activité à réaliser. Nous obtenons ainsi notre deuxième intention.

Intention 2. Fournir une activité qui sera accessible à la majorité des étudiants et plus particulièrement aux étudiants habituellement en difficulté.

Dans notre problématique et de nos considérations provenant d'études en génie logiciel, nous avons relevé la difficulté d'intégrer à la fois les notions informatiques et les notions du domaine de l'application. Idéalement, nous aimerions proposer une activité nécessitant peu ou pas de connaissances du domaine de l'application.

Une façon d'obtenir ce type d'activité est par l'observation du fonctionnement d'un système qui permettra, par la suite, de décrire les spécifications du programme à réaliser. Cette activité pourrait se centrer autour de la robotique qui constitue un environnement

permettant à l'étudiant de décrire et de prédire les comportements d'un objet robotisé. La robotique permet à l'étudiant d'appréhender directement, d'une manière quasi-sensorielle, l'ensemble des comportements de son objet. L'étudiant peut ainsi déduire concrètement la séquence des opérations réalisées par son objet. Cette séquence d'opérations correspondra au comportement du programme à réaliser par la suite.

Une fois la description du travail à réaliser terminée, nous devons offrir à l'étudiant un terrain propice à l'abstraction de la procédure. Ce qui nous amène à formuler une troisième intention.

Intention 3. Offrir une activité qui oriente l'étudiant vers des stratégies de développement appropriées

Dans nos considérations touchant la conception de programmes, nous avons énoncé deux modèles centrés sur l'organisation de l'activité: le modèle hiérarchique basé sur des modèles normatifs et les modèles opportunistes basés sur les déviations de l'activité. Le modèle hiérarchique privilégie un processus descendant de décomposition hiérarchique du problème avec une recherche de solution en largeur d'abord (stratégie par raffinements successifs). Les modèles opportunistes mettent l'accent sur les déviations de l'activité en admettant que l'activité réelle est organisée de façon opportuniste. Ici, au contraire du modèle hiérarchique, la recherche de solution peut se faire aussi bien de façon descendante qu'ascendante et la solution est recherchée en privilégiant aussi bien la largeur que la profondeur selon la situation. D'autre part, nous avons soutenu que la structure du langage de programmation, l'environnement de développement, le type de problème et l'expertise de l'étudiant dans le domaine informatique influencent le déclenchement d'une stratégie.

Un montage composé d'un microcontrôleur relié à un ordinateur exécutant les instructions d'un programme qui transmet des commandes au microcontrôleur peut répondre à nos besoins. En effet, ce type d'activité qui consiste à commander un microcontrôleur nécessite l'élaboration d'une séquence d'opérations (de commandes)

facilement identifiables qui sera le point de départ d'une recherche de solution par raffinements successifs ou d'une recherche opportuniste de solution voguant en largeur ou en profondeur guidé par le niveau d'évidence de l'opération à raffiner.

En processus d'abstraction de sa solution, l'étudiant devra identifier, outre les opérations automatisables, l'agencement de ces opérations sous forme de flux de contrôle. L'identification des structures exprimant ce flux de contrôle du système est loin d'être trivial pour l'apprenant et nous conduit à une quatrième intention.

Intention 4. Offrir des moyens et des outils facilitant l'abstraction de la solution d'un problème en vue de son informatisation.

Dans nos considérations portant sur l'organisation de l'activité de programmation, nous avons discuté d'un modèle d'apprentissage qui met en valeur les processus d'acquisition de schémas de connaissance en s'appuyant sur la thèse constructiviste. Ce modèle vise à fournir explicitement des schémas aux débutants de manière à faciliter le développement de leur expertise. De plus, selon un modèle d'acquisition en termes de restructuration interne de schémas, on peut penser que la pratique de la programmation structurée en incluant la formation à la conception, faciliterait la construction de l'expertise et ciblerait les parties importantes des schémas. Les deux types de schémas qui nous intéressent sont: le schéma élémentaire de programmation représentant des connaissances sur la structure de contrôle ainsi que les variables et le schéma algorithmique ou schéma complexe de programmation représentant une stratégie menant à la solution.

Un schéma est une structure générique de solution qui se distingue de la connaissance syntaxique et de la connaissance sémantique d'un langage. Pour l'acquisition de ces schémas nous exploiterons un outil familier au domaine de l'informatique soit, l'analogie.

De l'observation du comportement de l'environnement robotisé, l'étudiant visualise un algorithme concret qui lui permettra ensuite de déduire un algorithme abstrait comportant des notions propres à l'informatique. Ainsi, nous croyons que la logique inhérente à l'ordre d'ouverture de plusieurs luminodiodes suggèrera à l'étudiant l'utilisation d'une structure conditionnelle et que le tournoiement d'un moteur lui suggèrera l'utilisation d'une structure de répétition. La formulation analogique d'un schéma dans un domaine connu ou à l'aide d'une notion connue peut grandement faciliter son acquisition. Décrire une structure de répétition à l'aide d'une action familière et signifiante fournira à l'étudiant un schème d'appariement qui lui facilitera le transfert dans le domaine de l'application. Nous aimerions que ce transfert ou cette transposition puisse s'effectuer simplement sous la forme d'une projection ou d'une liaison entre deux domaines de représentations parallèles. De plus, la finalité de l'activité, l'objet robotisé à programmer devient aussi l'analogue qui guidera la démarche algorithmique. Ce qui théorise la pensée de Martial Vivet (2000) qui y voit un objet pour penser.

Nous avons spécifié, d'entrée de jeu, que les étudiants possèdent des niveaux de connaissance de la programmation variés. Les savoirs acquis et à acquérir étant par le fait même différents dans l'espace temporel d'une session de cours, les étudiants sont à la recherche d'informations inévitablement variées. Pour s'assurer de préserver leur motivation et satisfaire leur demande nous devons offrir un support facilement accessible qui permettra un apprentissage individualisé, le tutoriel.

Intention 5 Offrir un tutoriel, un outil qui permettra un apprentissage individualisé.

Dans nos considérations sur la pédagogie de projet nous précisions qu'elle favorise le développement de l'autonomie de l'étudiant tout en stimulant les interrogations sur les savoirs et les apprentissages. Pour ce faire, il faut rendre disponible efficacement l'ensemble des savoirs. L'accessibilité à ces connaissances, en tout temps et de n'importe où, est un atout majeur. Il faut semer lorsque le terrain est fertile!

Le recours aux technologies de l'information et des communications est alors opportun. Le dépôt sur Internet des notions à apprendre leur consentira une grande accessibilité tout en favorisant un traitement médiatisé. De cette façon, nous désirons présenter la matière dans un format épuré, facilitant une navigation logique comportant des interactions multiples. Un tutoriel disponible sur Internet, construit selon la philosophie des applets, est l'outil tout désigné qui répondra à ces attentes.

Dans nos considérations traitant de la pédagogie active centrée sur l'étudiant, nous prônions l'utilité des outils visuels pour établir des connexions, de même que le développement de questions qui facilitent l'exploration et l'avancement de l'étudiant. Ces considérations nous incitent à traiter astucieusement l'information sous forme visuelle. Dans cet esprit, nous proposerons dans le logiciel une rubrique offrant plusieurs vues graphiques lors de l'exécution d'un programme: une vue des instructions, une vue du contenu mémoire, un vue du contenu d'un fichier et une vue de l'affichage écran. Nous incorporerons également dans le logiciel une rubrique d'exercices où l'étudiant pourra découvrir l'utilité des notions, explorer leur utilisation et vérifier son apprentissage.

En considérant ces intentions, nous sommes maintenant en mesure d'esquisser nos premiers prototypes.

4.3 Prototypage

Notre EIAH se compose d'exercices de programmation à partir d'un montage électronique et d'un tutoriel présentant les notions importantes sous plusieurs formes. Dans le premier cas, il nous faut expérimenter du matériel afin d'obtenir celui regroupant les meilleures qualités possibles. Dans le deuxième cas, il nous faut identifier clairement les spécifications du logiciel à développer. Nous relaterons dans la section suivante, le cheminement qui nous a mené à notre réalisation matérielle et logicielle.

4.3.1 Prototype matériel : montage électronique

Nous avons dès le départ une bonne idée du montage à réaliser. Nous désirons, par l'entremise d'un microcontrôleur, permettre d'allumer et d'éteindre des luminodiodes. Nous désirons également permettre de démarrer et d'arrêter un moteur, en plus de commander sa vitesse. Pour ce faire, nous envisageons d'utiliser une plaque d'essai sur laquelle nous installerons un microcontrôleur PIC. En sortie du microcontrôleur, nous brancherons les luminodiodes et un transistor (monté en émetteur commun) pour déclencher/arrêter et varier la vitesse du moteur. En entrée du microcontrôleur, nous brancherons un capteur opto électronique (composé d'une luminodiode et d'une cellule photoélectrique) qui permettra à l'apprenant de détecter des événements extérieurs, de les compter ou de mesurer leurs écarts, pour par exemple calculer la vitesse du moteur.

Prototype 1

Notre premier prototype se consacre exclusivement au fonctionnement d'un moteur. Le montage est réalisé sur une plaque d'essai qui permet de commander le déplacement d'un curseur telle la tête d'écriture d'une « vieille imprimante » à points. Un microcontrôleur de la famille PIC16C7X est installé sur la plaque d'essai qui offre alors quatre connexions possibles. Dans ce montage, on peut commander le moteur DC et lire la position du curseur correspondant à la valeur de la résistance sur une entrée analogique. Le circuit électronique du microcontrôleur incorporant une sortie DB9, le montage est relié à l'ordinateur par son port série. Le microcontrôleur de ce montage travaille essentiellement avec une entrée analogique et un jeu de commandes trop restrictif pour programmer efficacement la variation de vitesse du moteur. La figure 4.1 montre le montage réalisé.

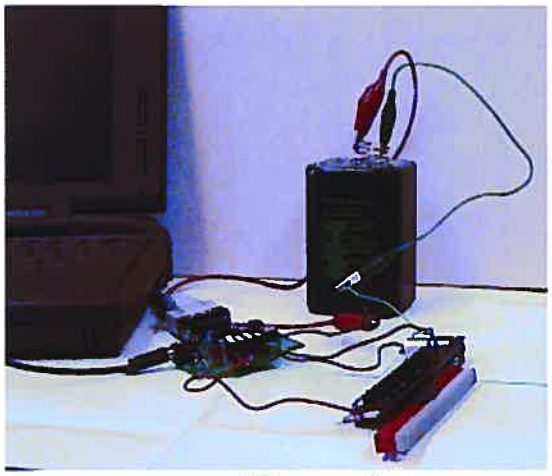


Figure 4.1 Prototype 1 : Montage électronique robotisé

Ce premier prototype nous a permis, par le fait même, d'explorer la communication sérielle. Notre objectif était de trouver un moyen simple qui pourrait s'intégrer facilement au programme de l'étudiant. Notre recherche a abouti sur trois entités de programmation appelées classe. En programmation par objet, l'instanciation d'une classe est un objet dont le comportement est autonome. Ainsi, il suffit d'incorporer une classe à un programme, d'instancier un objet de cette classe et finalement de l'utiliser. Précisons également qu'une classe contient des attributs et des méthodes. Les attributs sont en quelque sorte des données permettant de connaître l'état de la classe et les méthodes sont des fonctions ou des opérations propres à cette classe. La description des trois classes se trouve à l'annexe A.

La première classe nommée `CSerial` comprend 9 méthodes et 4 attributs. Les méthodes permettent d'ouvrir et de fermer le port sélectionné ainsi que d'y écrire ou d'y lire des caractères. La deuxième classe nommée `port` comprend 27 méthodes et 6 attributs. Outre des méthodes offrant des fonctionnalités similaires à la classe précédente, cette classe inclut des méthodes permettant de sélectionner le mode de transmission ou le type des données manipulées. La troisième classe nommée `CSerialPort` comprend 54 méthodes et 2 attributs. Beaucoup plus complexe que les deux autres classes, la classe `CSerialPort` offre plusieurs méthodes permettant de réaliser des opérations de bas niveau : gestion des interruptions, gestion des événements, gestion d'une file, etc.

Notre choix, dans le cadre de cette recherche, pour évaluer notre prototype, s'est arrêté à la première classe. Elle permet de réaliser efficacement toutes les opérations nécessaires à notre programmation tout en étant la classe la plus simple des trois.

Prototype 2

Notre deuxième prototype s'attaque à l'amélioration du fonctionnement du moteur. Le microcontrôleur est de nouveau installé sur une plaque d'essai. Par contre, nous utilisons un nouveau microcontrôleur identifié PIC16F876. Ce dernier a l'immense avantage de posséder une mémoire flash qui permet la mémorisation et la modification du programme. Le programme, rédigé sur un ordinateur en langage de programmation C, peut aisément, après avoir été compilé, être transféré en mémoire flash du microcontrôleur par l'entremise du port série. Puisque nous pouvons rédiger nos propres programmes, nous avons développé nos propres commandes et nous en avons profité pour y incorporer une procédure de variation de vitesse. En entrée du microcontrôleur, nous avons branché un capteur opto électronique (composé du luminodiode et d'une cellule photoélectrique). En sortie du microcontrôleur, nous avons branché un transistor (monté en émetteur commun) pour déclencher/arrêter et varier la vitesse du moteur. Le moteur était fixé à l'aide d'une pince sur un support de laboratoire juste au dessus du capteur opto électronique. Une pile de 6 volts a servi d'alimentation pour le moteur. Ainsi, le moteur possédait sa propre alimentation de manière à isoler le microcontrôleur des bruits de celui-ci. La figure 4.2 présente un croquis du montage obtenu.

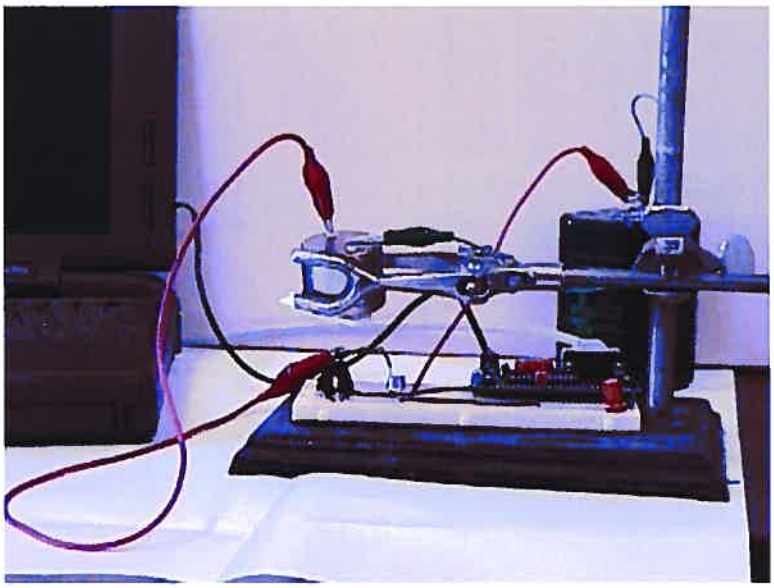


Figure 4.2 Prototype 2 : montage électronique robotisé

Ce prototype nous permet de commander, à notre guise, le fonctionnement du moteur. Par contre, outre la grande taille du montage, l'étudiant peut rencontrer certaines difficultés lorsqu'il devra ajuster la hauteur du moteur et connecter correctement les fils. Il nous faudra sûrement remédier à ces problèmes essentiellement techniques.

Prototype 3

Notre dernier prototype est une évolution du précédent qui intègre tous les composants sur la plaque d'essai. Un condensateur est ajouté à ce montage pour réguler son alimentation. Ce condensateur, en éliminant le parasitage en sus du moteur, permet d'utiliser la même alimentation que celle du microcontrôleur et l'éliminer ainsi une pile sur notre prototype. Le moteur, quant à lui, est placé debout près du capteur opto électronique de manière à permettre à l'hélice de passer au dessus. Ce stratagème nous permet d'enlever cette fois le support du montage. La figure 4.3 présente le montage obtenu.

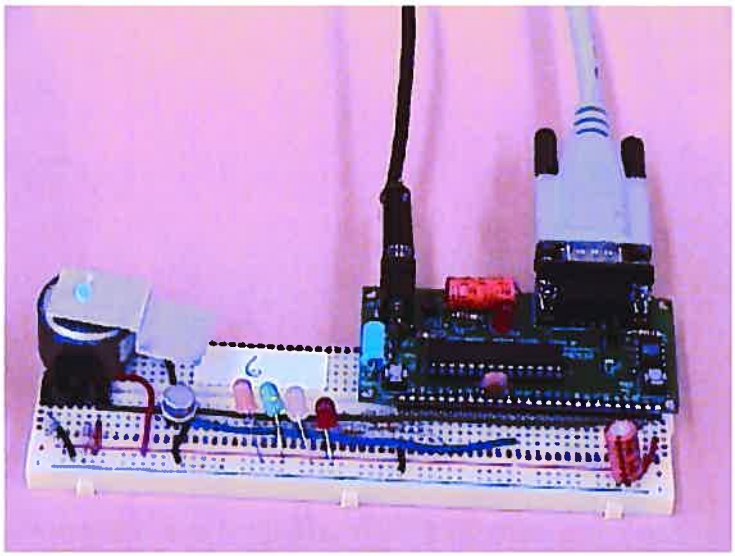


Figure 4.3 Prototype 3 : montage électronique robotisé

La facilité de programmation du microcontrôleur est un atout majeur puisqu'elle nous permet une programmation distincte pour chacun des montages. Ceci évitera qu'une équipe puisse utiliser le travail d'une autre équipe.

Le montage obtenu est simple, petit et robuste. La seule manipulation que l'étudiant doit effectuer sur le montage est le branchement de l'alimentation et du câble série. Du point de vue utilisation, le montage permet une grande diversité d'exercices. Entre autre, la commande d'un moteur nous ouvre la porte à une panoplie d'applications qui plairont à plusieurs étudiants futurs ingénieurs. Les exercices peuvent être conçus de manière à cibler une notion ou une difficulté particulière. On peut même imaginer la composition d'un exercice par notion à enseigner. Le potentiel didactique de cet outil n'est limité que par l'esprit créatif de l'enseignant et celui aussi de l'étudiant.

4.3.1.1 Mise à l'essai fonctionnelle

La mise à l'essai fonctionnelle a été réalisée auprès des membres du laboratoire de robotique pédagogique (LRP). Les experts du laboratoire nous sont venus en aide à plusieurs reprises lorsque des problèmes électroniques survenaient. Leur connaissance et leur expérience des technologies nous ont été d'un apport inestimable.

Les essais effectués sur le premier prototype nous ont permis de confirmer la faisabilité de l'idée tout en nous dévoilant certaines difficultés. La difficulté la plus importante concerne la variation de vitesse du moteur. Les commandes offertes pas le microcontrôleur ne permettent pas de faire varier convenablement la vitesse du moteur. Le comportement du moteur étant imprévisible nous devons, soit changer de type de moteur, soit changer de microcontrôleur. Nous avons décidé de regarder du côté d'un nouveau microcontrôleur puisque le jeu de commandes nous apparaît trop restrictif et par le fait même trop contraignant. Il nous faut un microcontrôleur plus souple et mieux adapté à nos besoins.

Les essais réalisés sur le deuxième prototype nous ont démontré les immenses possibilités de notre nouveau microcontrôleur. Pouvant inscrire notre propre programme en mémoire du microcontrôleur nous obtenons alors la solution à notre problème. Ce microcontrôleur permet même au développeur de modifier au besoin son programme et de le retranscrire en mémoire. Ces avantages nous ont permis de progresser plus rapidement. Les différents essais nous ont démontré la nécessité d'ajouter certains composants principalement pour obtenir de l'information du microcontrôleur. Par contre, les déplacements du montage nous ont également dévoilé la nécessité de nous départir du support et de la pile qu'on ne retrouve pas dans un laboratoire d'informatique.

Les essais sur le troisième prototype sont pratiquement d'ordre électronique. Lors de ces essais, nous avons grandement apprécié l'aide des experts du LRP. Leurs conseils et leurs suggestions nous ont conduit vers un montage compact, simple et pratique. L'outil obtenu est maintenant prêt pour la mise à l'essai empirique.

4.3.2 Prototype logiciel : le tutoriel

Le tutoriel que nous désirons développer doit prioritairement être accessible en tout temps. Pour ce faire, nous avons pensé le rendre disponible sur Internet. Puisque le

langage de programmation tout désigné pour le développement d'application s'exécutant sur l'Internet est le langage Java, nous décidons que le tutoriel sera conçu à l'aide de ce langage. Une fois cette décision prise, nous sommes prêt à débuter notre travail de conception.

Notre développement logiciel s'appuiera sur le modèle en spirale. Rappelons que ce modèle conçoit un produit basé sur le prototype évolutif. L'emphase du développement est mise sur la détermination des spécifications et la conception au sens du design.

Prototype 1

Plutôt que de couvrir au départ beaucoup de matières, nous avons décidé d'articuler notre premier prototype autour d'une seule notion, les modes de transmissions de paramètres d'une fonction. Cette notion fut traitée sous les quatre rubriques : théorie, simulation, exercices et simulations. Du point de vue logiciel nous avons regroupé ces rubriques dans une première fenêtre présentant cinq boutons, soit un par rubrique et un pour terminer l'application. La sélection d'un bouton engendre une nouvelle fenêtre, indépendante de la première, qui présente l'information sous la forme annoncée. Lorsque l'étudiant ferme cette dernière fenêtre, la fenêtre initiale est réaffichée. Il peut ensuite effectuer une nouvelle sélection. En quelques mots, le déroulement de l'application consiste à une superposition de fenêtres. La figure 4.4 illustre l'enchaînement de ces fenêtres.

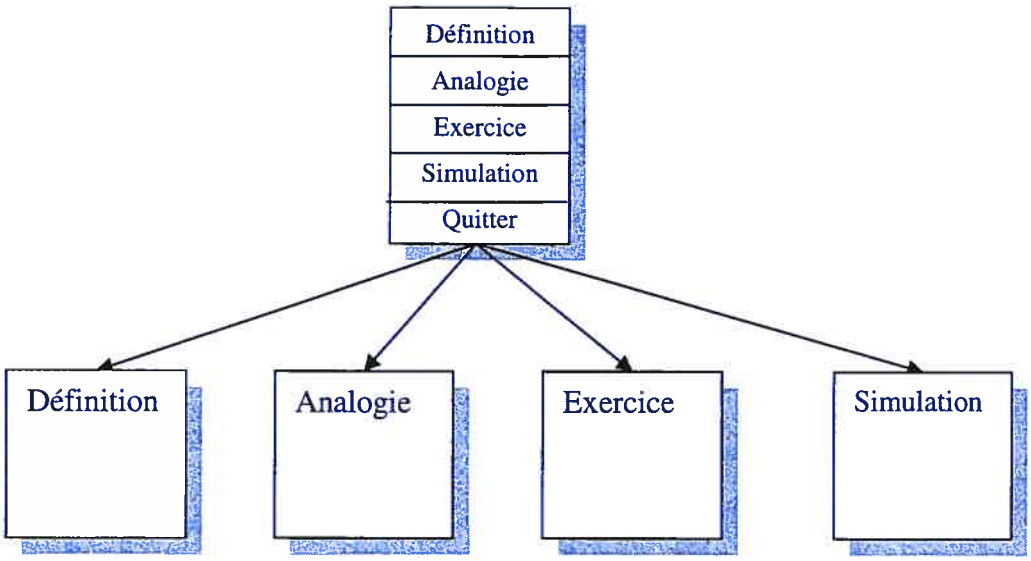


Figure 4.4 Enchaînement des fenêtres permettant l'accès à l'information

Ce premier prototype nous a permis d'expérimenter et de découvrir plusieurs possibilités d'organisation et de présentation de l'information. Il nous a surtout convaincu de la faisabilité du logiciel ainsi que de sa pertinence pédagogique.

Prototype 2

En se basant sur les bons résultats du premier prototype, nous avons décidé de conserver l'idée des quatre rubriques et de traiter toutes les notions de la même façon. Par contre, la stratégie des superpositions de fenêtres devient beaucoup trop lourde pour le traitement de plusieurs notions. Nous avons donc opté pour une facture plus classique qui utilise une fenêtre initiale divisée en trois zones : une barre de menus déroulants, une zone d'affichage de l'information et une barre d'état. Il s'agit du type d'écran offert par une grande majorité de logiciel. L'étudiant doit alors sélectionner la notion à étudier par l'entremise du menu « unité » et choisir ensuite la rubrique désirée. Il n'y aura plus ici de superposition de fenêtres comme pour le premier prototype puisque l'affichage s'effectue dans la même zone. Par contre, nous pourrons superposer une fenêtre au besoin pour une même rubrique. La figure 4.5 montre la nouvelle organisation de la fenêtre principale.

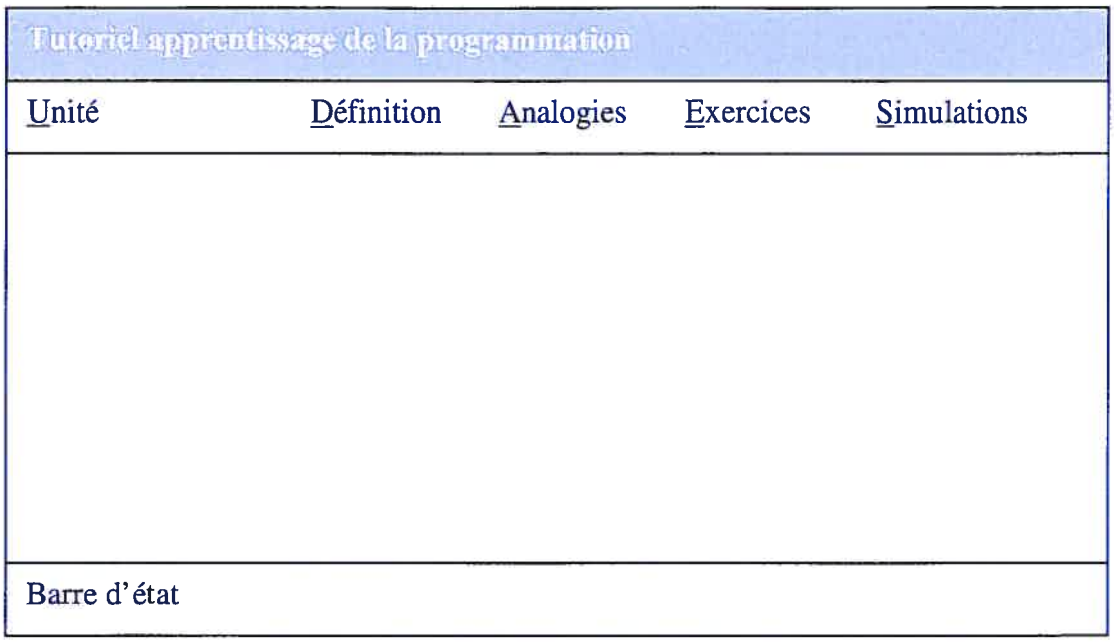


Figure 4.5 Fenêtre principale permettant l'accès à l'information

Dans la terminologie du logiciel, nous parlons d'une unité au sens de l'unité d'apprentissage d'une notion. Lorsqu'une unité est sélectionnée, les quatre rubriques de la notion correspondante sont aussitôt chargées dans la mémoire de l'ordinateur de l'étudiant.

Il est important de souligner que nous avons préalablement rédigé les spécifications de chacune des unités associées à une notion avant de réaliser la programmation du tutoriel. De plus, outre une seule unité, la programmation a été effectuée avec l'aide de deux étudiants stagiaires, d'où l'importance de bien spécifier chaque unité. Voici la description de la première des dix unités portant sur les variables et les types qui permettra de réaliser sa programmation. La description des autres unités est présentée à l'annexe C.

Unité 1 : Les variables et les types simples

a) Définition

Qu'est-ce qu'un identificateur?
Un identificateur est le nom d'une entité. Le nom d'une variable, d'une constante ou d'une fonction correspond à un identificateur.

Un identificateur est une chaîne de caractères composée uniquement de caractères alphanumériques et du caractère de soulignement, débutant obligatoirement par une lettre.
Les lettres minuscules et majuscules sont différenciées. Ainsi TP1, Tp1, tP1, tp1 sont quatre identificateurs distincts.

Qu'est-ce qu'une variable?
Il s'agit d'une donnée située à un emplacement mémoire dont la valeur peut être modifiée
Détails sur l'initialisation d'une variable

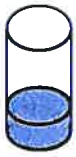
Qu'est-ce qu'une constante?
Il s'agit d'une donnée située à un emplacement mémoire dont la valeur ne peut être modifiée

Qu'est-ce qu'un type?
Le type correspond à la sorte de variable
Le type précise pour la variable:
- l'espace mémoire occupé
- la représentation mémoire; la façon dont elle est mémorisée
- les opérations admissibles

Faire ressortir le rôle de chacun.

b) Analogies

Contenu vs Contenant (liquide dans un verre)

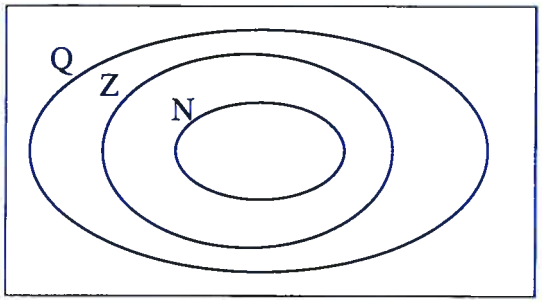


Une variable peut être comparée à ce verre. On peut ajouter et retirer du liquide selon sa bonne volonté.

Un contenant et son contenu. La variable se distingue également selon deux aspects : son emplacement mémoire et sa valeur mémorisée Le contenu est associé à la valeur de la variable et le contenant à l'emplacement mémoire (physique) de la variable.

Station de télé vs Écran de télé (changement de station à la télé)

Elément vs Ensemble
(x ∈ N, x ∈ Z, etc.)



c) Exercices

Identification des variables d'un problème et de leur type

« On peut reprendre ici le même style d'interface que celui des fonctions pour la détermination des paramètres et leur type »

1. Mémorisation de valeurs
    1. Le nombre de pages d'un livre
    2. L'age du chat de Marie
    3. Le numéro d'un étage d'un édifice
    4. La couleur du cheval blanc de Napoléon
    5. La longueur d'un câble
    6. L'ampérage d'un moteur électrique
    7. Le nom du fabricant d'un ordinateur
    8. Une lettre lue du clavier
    9. L'état d'une lampe (allumée ou éteinte)

Variable	
Identificateur	Type
NoEtage	int

```
void main()
{
   int NbPages;
   int AgeChat;

}
```

2. Identifier les variables d'un programme permettant de :
    1. ordonner alphabétiquement les trois noms de fleurs lus du clavier;
    2. calculer le nombre de secondes comprises dans un certain nombre d'heures lues du clavier;
    3. compter le nombre d'occurrences de la lettre «a» dans une phrase lue du clavier;
    4. ajouter la taxe de vente et la taxe de service au coût d'un produit;

5. calculer la résistance équivalente à trois résistances reliées en série; la valeur de chacune des trois résistances est lue du clavier.

Variable		Opérateur
Identificateur	Type	

```
void main()
{
   int NbPages;
   int AgeChat;

}
```

d) Simulations

Modification du contenu d'une variable par des affectations
Modification du contenu d'une variable par des opérations
Les trois opérations de la permutation
Exemple par type, exemple avec plusieurs types

Simulation 1

```
#include <iostream>
using namespace std;

void main()
{
   int Entier1, Entier2, Tampon;

   // L'operateur d'affectation precise pour la variable la valeur
   // a memoriser

   Entier1 = 0;
   Entier1 = 24;
   Entier1 = -65;

   Entier2 = 7;

   // Le terme a droite de l'operateur d'affectation est une expression.
   // Une expression peut etre une constante, une variable ou composee
   // d'operandes
   // lies par des operateurs

   Entier1 = 34 / 5;  // Division entiere
   Entier2 = ( 12 - 5)*24 % 2; // L'evaluation se fait selon l'ordre
                               // de priorite des operateurs

   // La permutation de deux valeurs

   // Manipulation erronee
   Entier1 = 52;
   Entier2 = 38;

   Entier1 = Entier2;
   Entier2 = Entier1;

   // Manipulation correcte
   Entier1 = 52;
   Entier2 = 38;

   Tampon = Entier1;
   Entier1 = Entier2;
   Entier2 = Tampon;

}
```

Simulation 2

```
#include <iostream>
using namespace std;

void main()
{
   char Lettre = 'K';
   int Entier1 = 37,
       Entier2 = -82,
       Entier3 = 5;
   bool Booleen = true;
   double Reel_X = 63.73,
          Reel_Y = -3.14e25;
   char Phrase[250] = "Le telephone sonne. Vite, repondez!";

   // Conversion implicite

   Reel_X = Entier1;    // Conversion sans probleme
   Entier2 = Reel_X;    // Conversion possible mais tres dangereuse
   Entier3 = Lettre;    // Conversion sans probleme
   Booleen = Reel_Y;    // Conversion, si Reel_Y=0 donne faux !=0 donne vrai
   Booleen = Lettre;    // Meme raisonnement que precedemment
   Entier1 = Booleen;   // Conversion sans probleme

   // Conversion explicite

  Reel_X = Entier1 / Entier2; // Division entiere puisque deux
                              // operandes entiers
  Reel_Y = Entier1 / double(Entier2);  // Division reelle puisque un operande
                                       // entier et un reele

  /* Instructions erronees */
   //Phrase = " Salut la compagnie ";
   //Phrase = Reel_X;    => et vice-versa
   //Phrase = Entier1;   => et vice-versa
   //Phrase = Booleen;   => et vice-versa
   //Phrase = Lettre;    => et vice-versa

   // Toutes les manipulations sur les chaines de caracteres doit se
   // faire a l'aide d'une ou des fonctions
   strcpy(Phrase, "Mon pays c'est l'hiver");

}
```

Simulation 3

```
#include <iostream>
#include <cstring>
using namespace std;

void main()
{
   char Phrase[120];
   char Ligne[80]= " Le telephone sonne. Vite, repondez!";
   int OrdreAlpha;

   //Affectation d'une valeur a une chaine
   strcpy(Phrase, "Dring! Dring!");
   // Affichage d'une chaine de caracteres
   cout << "Phrase =  " << Phrase << endl;

   // Affichage du nombre de caracteres presents dans la chaine
   cout << "Phrase contient " << strlen(Phrase) << " caracteres."
        << endl;

   // Concatenation de deux chaines de caracteres
   strcat(Phrase, Ligne);
   cout << "Phrase =  " << Phrase << endl;

   // Comparaison de deux chaines de caracteres
   OrdreAlpha = strcmp(Ligne, Phrase);
   cout << "OrdreAlpha =  " << OrdreAlpha << endl;
   OrdreAlpha = strcmp(Phrase, Phrase);
   cout << "OrdreAlpha =  " << OrdreAlpha << endl;
   OrdreAlpha = strcmp(Phrase,"Drelin! Drelin!");
   cout << "OrdreAlpha =  " << OrdreAlpha << endl;

   //Modification du 3 ieme caractere de la chaine Phrase
   strcpy(Phrase, "Dring! Dring!");   Phrase[2] = 'o';
   cout << "Phrase =  " << Phrase << endl;
}
```

Tout au long du développement du logiciel, il y a une vérification et une validation du travail accompli. Des modifications sont apportées au fur et à mesure de manière à produire un prototype évolutif fonctionnel.

4.3.2.1 Mise à l'essai fonctionnelle

La mise à l'essai fonctionnelle de notre tutoriel a été réalisée auprès d'étudiants développeurs et de deux membres du bureau d'appui pédagogique de l'École Polytechnique de Montréal. Soumis à un comité de sélection dans le cadre d'un concours[1] de cette institution, le tutoriel a décroché deux subventions sur deux années successives. Ce comité de sélection se compose de trois professeurs et d'un conseiller pédagogique. Cet appui nous confirmait la pertinence de développer notre logiciel.

Conformément au modèle de développement de logiciel dit en spirale, le prototype est soumis à plusieurs tests de validation. Chaque portion du programme est précisément décrite, puis transposée dans le langage de programmation et examinée par les évaluateurs. Les corrections et les modifications sont aussitôt effectuées. Le logiciel est de nouveau évalué avant d'entamer la portion suivante. Il faut préciser ici que les évaluations réalisées concernent autant le contenu que sa résultante informatique. Finalement, le prototype est examiné par les conseillers pédagogiques qui nous ont proposé quelques améliorations possibles tout en nous demandant de le présenter à des professeurs. Lors de cette présentation, plusieurs professeurs ont souligné l'apport indéniable du traitement graphique de l'information pour l'apprenant.

Le premier prototype logiciel nous a permis d'explorer différentes interfaces usagers. Lien direct avec l'étudiant, ces interfaces ont une importance capitale dans la présentation des savoirs et savoir-faire. Dépendamment des objectifs visés par une rubrique, il fallait trouver l'interface qui intéresserait et captiverait l'étudiant. En ce sens, notre ligne de pensée était de rendre l'étudiant actif lors de l'utilisation du logiciel et d'éviter de le submerger dans une mer de textes. Les commentaires reçus sont favorables à l'interface développer.

---

[1] Il s'agit du concours du Fonds d'aide au TIC (FATIC) qui vise à stimuler la création de sites Web de cours de bon niveau, ainsi que la réalisation d'applications exploitant d'autres formes de TIC.

Le deuxième prototype logiciel est une évolution du premier. En d'autres termes, nous avons complètement intégré le premier prototype dans le deuxième. Ayant adopté la même méthode de développement logiciel, le deuxième prototype est soumis aux mêmes essais ou validations que le premier. Par contre, la validation touche majoritairement le contenu puisque le design informatique concerne des adaptations du premier prototype. Le développement s'est poursuivi sur plus d'un an afin d'incorporer neuf nouvelles unités. Chaque unité est soumise au cycle de description, programmation et validation. La vérification des conseillers pédagogiques nous sera extrêmement utile. Encore une fois, on nous invite à présenter notre prototype. C'est dans le cadre du colloque Formation en ingénierie et TIC (FITIC) au printemps 2003 à l'École Polytechnique de Montréal qu'il est présenté.

4.4 Description de l'environnement informatique pour l'apprentissage humain

L'environnement informatisé pour l'apprentissage humain (EIAH) comprend d'une part un atelier de programmation correspondant à des exercices de programmation utilisant un montage électronique robotisé et d'autre part un tutoriel pour assister l'étudiant dans l'apprentissage de la programmation. Les exercices, de type travaux pratiques encadrés (TPE), intègrent des objets pour penser de manière à rendre signifiante l'activité de programmation et à concrétiser les structures de programmation. Le tutoriel renferme différents concepts informatiques inscrits dans des unités distinctes. Chaque unité présente et traite le concept à l'aide de quatre rubriques: définition, analogie, exercice et simulation. Le grand intérêt de ce système est qu'il peut être utilisé en classe et à l'extérieur de la classe. Le tutoriel est une application accessible sur Internet développée en langage JAVA. De cette façon, l'environnement est accessible en tout temps et de presque partout.

4.4.1 L'atelier de programmation

À la lecture des ACTES du 19e Colloque AIPU 2002 dont le thème central était *Les méthodes actives dans l'enseignement supérieur*, un consensus ressort à travers plusieurs articles: l'apprentissage s'accroît lorsque l'étudiant est sollicité et actif. Fini le temps du martelage théorique, du professeur monologuiste, de l'étudiant entonnoir qui tente d'ingurgiter le plus d'informations possibles. L'heure est à la recherche d'un enseignement significatif et « contextualisé » qui réponde au besoin de chacun. Pour ce faire, nous proposons deux exercices de programmation utilisant un montage électronique robotisé.

4.4.1.1 Exercice: Expression booléenne et structures de décision

Ce premier exercice vise à faciliter l'apprentissage de l'expression booléenne et des structures de décision. L'exercice consiste à allumer une luminodiode en fonction de l'état des autres luminodiodes. Par exemple,

- allumer la luminodiode blanche uniquement si la luminodiode rouge est allumée
- allumer la luminodiode verte si les luminodiodes jaune et rouge sont allumées
- etc.

Nous avons simplifié ce problème en l'énonçant de la façon suivante :

> Découvrir l'ordre dans lequel les quatre luminodiodes doivent être allumées dans une séquence préétablie.

Par contre, puisqu'il s'agit d'un premier exercice nécessitant l'utilisation d'un matériel électronique, nous avons prévu dans un premier temps une manipulation par l'entremise d'un programme. Ce programme jouera un double rôle. Primo, il permet de conscientiser l'étudiant aux contraintes matérielles et temporelles de la liaison entre l'ordinateur et le microcontrôleur. Secundo, il permet de clarifier le comportement du programme à écrire.

La vitesse du processeur de l'ordinateur (1.8GHz) quantifiée en milliards d'instructions par seconde est trop rapide pour le microcontrôleur (20MHz) qui ne peut réaliser que quelques millions d'instructions par seconde. Il ne faut pas oublier également la vitesse du port série, environ 19200 bauds[2] ou plus de 19200 bits par seconde constituant, dans notre cas, une commande transmise au microcontrôleur ou une information transmise par le microcontrôleur. Dans cette première activité, l'étudiant devra prendre conscience de cette différence de vitesse et la prendre en compte dans sa programmation. La problématique de l'interaction entre ces trois composants peut s'exprimer sous la forme d'une commande ignorée ou une lecture erronée provoquées par deux commandes successives envoyées quasi simultanément au microcontrôleur ou par une lecture trop rapide sans que le microcontrôleur ait pu envoyer complètement l'information.

Le travail consiste à déterminer l'ordre dans lequel quatre luminodiodes doivent être allumées. Aussi banal que cela puisse paraître, l'étudiant exigera inévitablement des précisions à cet énoncé. La solution est-elle unique? Si une luminodiode nécessite que deux autres luminodiodes soient allumées, pourra-t-on l'allumer si trois luminodiodes sont allumées? Si la première luminodiode à allumer est la rouge, est-il possible de l'allumer même si la première tentative concerne la luminodiode verte? L'utilisation initiale de notre programme permettra à l'étudiant d'explorer ses interrogations et de transposer ses observations dans le langage de programmation.

Cette étape initiale permet à l'équipe d'échanger tout en s'amusant. Le défi de trouver l'ordre d'allumage des luminodiodes est propice à la recherche d'une stratégie gagnante au cœur de laquelle on retrouve le savoir en jeu : expressions booléennes et structures de décision. Quel plaisir de trouver la bonne séquence! Cette réponse est le signal de départ et de surcroît la clé manquante du programme à réaliser.

---

[2] Le terme « baud » fait référence au nombre de changements de fréquence par seconde, tandis que « bits par seconde » représente réellement le nombre de bits émis. Chaque modulation porte généralement un bit à la fois, mais il arrive qu'elle en porte davantage. Il n'y a donc pas toujours d'équivalence entre le nombre de bits transmis par seconde et le nombre de bauds.

4.4.1.1.1 L'étape de programmation

Les instructions doivent s'inscrire dans un programme existant contenant les fonctions nécessaires à l'interaction avec le micro contrôleur. Les différentes commandes adressables au micro sont fournies à l'équipe sous forme de fonctions. Ces fonctions sont : initialiser le port série, démarrer le moteur, obtenir la vitesse du moteur, activer le moteur selon une vitesse donnée, allumer une luminodiode sans condition, éteindre une luminodiode sans condition, allumer une luminodiode sous condition, éteindre une luminodiode sous condition et attendre un certain délai.

La figure 4.6 montre le fichier initial dans lequel l'étudiant doit ajouter ses propres instructions. L'emplacement exact des instructions ajoutées correspond à l'endroit où apparaissent les fonctions `DemarrerArreterMoteur()` et `AllumerEteindreLed()`. Notons que l'exécution de ces deux fonctions permet à l'étudiant de réaliser la phase initiale de familiarisation et de recherche « manuelle » de solution. La fonction `AllumerEteindreLed()` sert le premier exercice que nous venons de décrire et la fonction `DemarrerArreterMoteur()` servira notre prochain exercice.

```
/*-------------------------------------------------------------------------*/
/* FICHIER:     Micro.cpp                                                  */
/* AUTEUR:      YVES BOUDREAULT                                            */
/* DATE  :      14 aout 2002                                               */
/* DESCRIPTION: Ce fichier contient le programme permettant de décou-      */
/*              vrir l'ordre d'ouverture des luminodiodes ainsi que        */
/*              l'intensité électrique nécessaire pour démarrer et         */
/*              arrêter le moteur.                                         */
/*-------------------------------------------------------------------------*/

#include "stdafx.h"
#include "serial.h"

#include <utilitaires.h>
#include <iostream>
#include <cstring>
using namespace std;

/*-------------------------------------------------------------------------*/
/* DESCRIPTION: Fonction principale                                        */
/*              Fonction offrant une interface permettant de démarrer      */
/*              et arrêter le moteur ainsi que d'allumer et d'éteindre     */
/*              les leds.                                                  */
/* FONCTIONS :  DemarrerArreterMoteur() pour manipuler le moteur           */
/*              AllumerEteindreLed() pour manipuler les leds               */
/* VALEUR DE RETOUR: Aucune                                                */
/* REMARQUE:         Aucune                                                */
/*-------------------------------------------------------------------------*/
void main()
{
   CSerial Port;

   //Attendre avant de transmettre une commande
   ATTENDRE(100000000);

   if (Port.Open(1,19200))
   {
       DemarrerArreterMoteur(Port);
      AllumerEteindreLed(Port);
   }
   else
      cout << "Probleme d'ouverture du port!";
}
```

Figure 4.6 Fichier où doivent être ajoutées les instructions de l'étudiant

4.4.1.1.2 Particularités

L'atelier 1 permet à l'étudiant de prendre conscience qu'un ordinateur est un automate qui exécute les commandes les unes à la suite des autres. La séquence des instructions du programme se concrétise par une séquence d'actions visualisables sur le montage.

La mémorisation de l'état d'une entité correspond à une action, par conséquence si aucune instruction ne précise de mémoriser une donnée, celle-ci sera oubliée. Notre montage plonge l'étudiant dans cette réalité par l'entremise de l'état d'une luminodiode. Il lui est impossible de savoir, à partir du programme rédigé, si la luminodiode est allumée ou éteinte. Cette constatation devrait le sensibiliser à l'utilité de la variable. Ce principe qui apparaît simple à première vue peut être à l'origine de plusieurs incompréhensions du débutant.

L'objectif principal de cet exercice est le choix de la structure conditionnelle qui satisfera l'ordre d'ouverture des luminodiodes. Cette structure se compose d'une ou plusieurs expressions booléennes dont la formulation sera également révélatrice de la logique et de la compréhension de l'étudiant.

4.4.1.2 Exercice : Structures de répétition

L'exercice consiste à déterminer l'impulsion minimale nécessaire pour démarrer un moteur et à l'inverse, l'impulsion qui arrêtera le moteur. Pour ce faire, nous avons inscrit sur le processeur du micro contrôleur une fonction agissant sur la vitesse du moteur. Cette fonction s'appelle une régulation en PWM (Pulse Width Modulation) ou modulation de largeur d'impulsion en français. Il s'agit d'un moyen simple de réguler le courant que l'on envoie dans un moteur unipolaire, en découpant le courant (continu) en périodes très fines, et de ne maintenir la tension que pendant une fraction de ce période de temps. À l'intérieur de cette période, par exemple, si on maintient la tension pendant 33% de la période, le moteur se comportera comme s'il était alimenté avec une tension trois fois inférieure. Il faut évidemment que les périodes soient suffisamment fines, pour permettre aux inductances du moteur d'intégrer celles-ci de manière à ce que le moteur perçoive ces impulsions comme un courant continu.

Comme dans l'exercice précédent, l'étudiant est invité à découvrir manuellement l'intensité nécessaire pour déclencher le moteur. À l'exécution du programme il doit

tenter successivement plusieurs valeurs afin de découvrir celle qui démarrera le moteur. L'équipe pourra utiliser différentes stratégies telles que débuter à 0 et incrémenter par pas unitaire jusqu'au déclenchement du moteur ou envisager une recherche dichotomique qui exigera nécessairement plusieurs essais. Une fois, le moteur en marche, l'étudiant recherchera de façon similaire le point d'arrêt en diminuant successivement l'intensité. Indépendamment des stratégies utilisées pour trouver ces deux valeurs, nous nous attendons à ce que l'étudiant raffine sa programmation en déduisant l'utilité d'une structure de répétition.

Pour réaliser son programme, l'équipe possède tous les outils décrits dans l'exercice précédent. Le démarrage et l'arrêt du moteur sont automatiquement perçus par un capteur qui transmet l'information au microcontrôleur. En comparant l'information donnée par le microcontrôleur et l'observation visuelle du moteur, l'étudiant validera le bon fonctionnement du montage.

Lors de ces exercices, l'étudiant peut trouver certaines réponses à ses questions en recourant au tutoriel. La section suivante précise les différentes composantes du tutoriel.

4.4.2 Le tutoriel

Le tutoriel offre en tout temps et juste à temps l'information nécessaire à la progression du travail de l'étudiant. Il est un outil complémentaire d'aide à l'apprentissage que tout étudiant pourra consulter pour trouver des réponses à ses questions. Permettant un apprentissage autodidacte, le tutoriel traite de différents concepts informatiques à l'aide de quatre rubriques : définitions, analogies, exercices et simulations.

4.4.2.1 Rubrique Définition

Cette rubrique contient la description des éléments propres au sujet de l'unité. Un élément peut être une entité informatique ou un terme syntaxique. Nous avons eu recours aux onglets pour organiser l'information sur ces éléments. La particularité de cet

ensemble d'onglets est qu'il permet d'afficher l'information d'un seul élément à la fois. Cette information se compose d'une brève description à laquelle s'ajoute dans certains cas un exemple d'utilisation correcte ou un exemple d'utilisation erronée. La figure 4.7 présente l'affichage de l'ensemble d'onglets de la première unité portant sur les variables et les types.

Sans parler de vulgarisation, l'information devra être simple, claire et précise. Elle est hiérarchisée selon un principe d'utilisation ou d'inclusion et de spécialisation. Le principe d'inclusion permet une abstraction croissante. Par exemple, dans l'ensemble des onglets de la figure 4.7 nous lisons la suite d'onglets :

Variable → Constante → Type → Identificateur

Cet ordre satisfait le principe d'inclusion. La variable est une donnée en mémoire qui peut être modifiée, une constante est une donnée en mémoire qui ne peut être modifiée, le type est une sorte de variable ou de constante et l'identificateur est le nom donné à une variable, une constante ou un type. Inversée la séquence nous imposerait de décrire un identificateur à l'aide de termes non maîtrisés ou d'un terme général. Par exemple la description d'un identificateur devient : un identificateur est le nom donné à une entité. Cette dernière description risque d'être vide de sens pour le débutant.

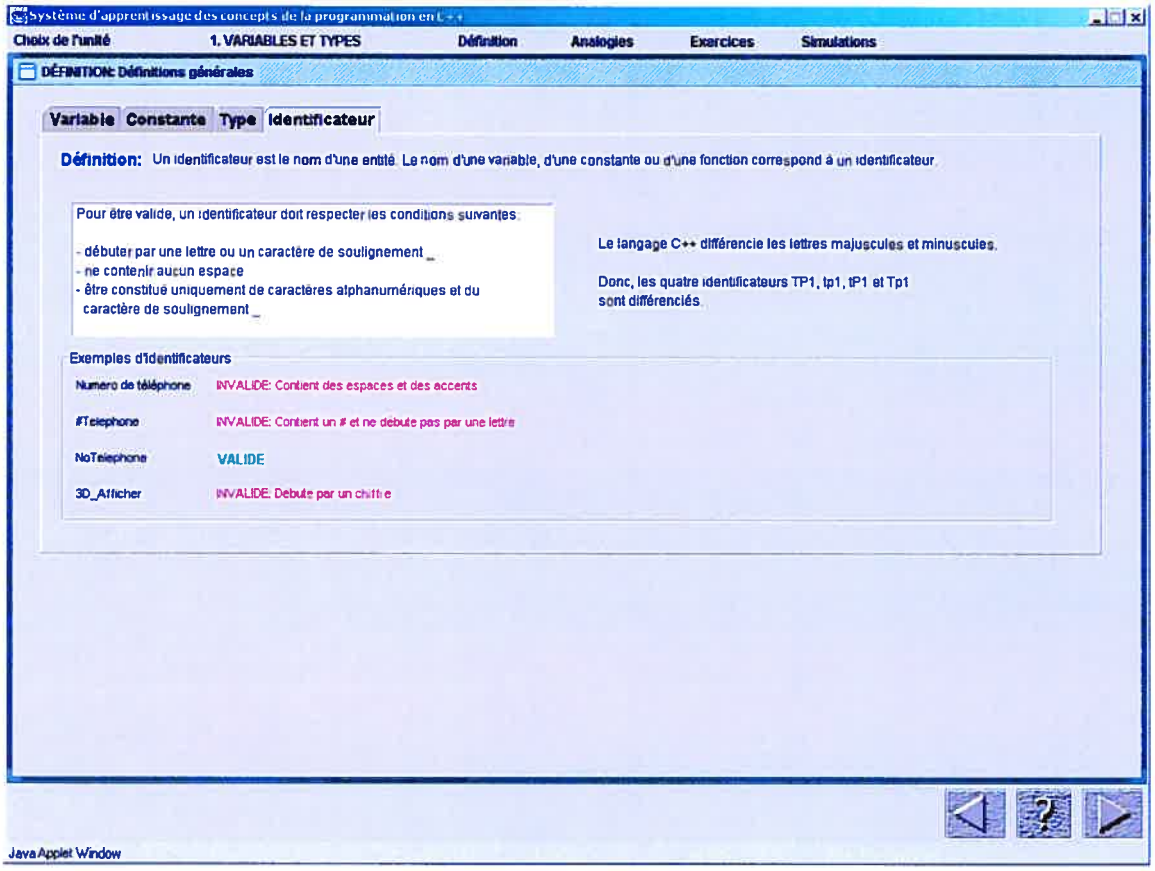


Figure 4.7 Ensemble d'onglets de la rubrique Définition

Lorsqu'il est nécessaire d'ajouter d'autres détails nous proposons un bouton à bascule dont l'activation produit un affichage superposé ou non à la zone de l'onglet. L'information supplémentaire apparaît et disparaît d'un simple clic de l'étudiant. Idéalement, l'étudiant devrait afficher et lire cette information supplémentaire uniquement après avoir lu l'information précédente essentielle à sa compréhension. L'idée d'information supplémentaire procède aussi d'une structure et d'une logique top-down. La figure 4.8 montre l'affichage obtenu lors de l'activation du bouton Afficher les types standard en C++. Dans ce cas, un deuxième ensemble d'onglets contenant les différents types est affiché sous le premier. Nous appliquons notre règle de simplicité de nouveau en présentant la description d'un seul type à la fois.

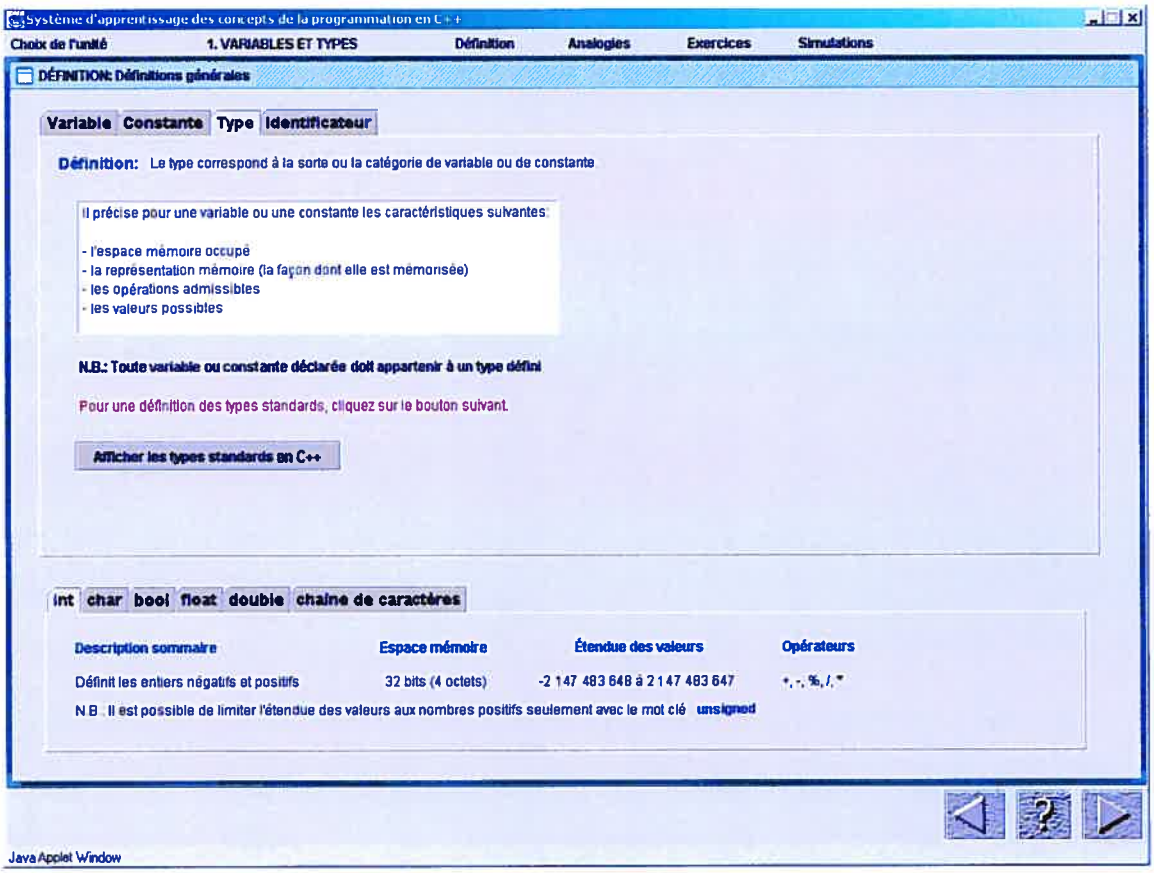


Figure 4.8 Affichage d'une information supplémentaire

La figure 4.9 présente une autre façon d'afficher une information supplémentaire sans alourdir la fenêtre principale. Ici, une deuxième fenêtre est créée et superposée à la fenêtre principale en masquant une portion de moindre importance pour l'instant.

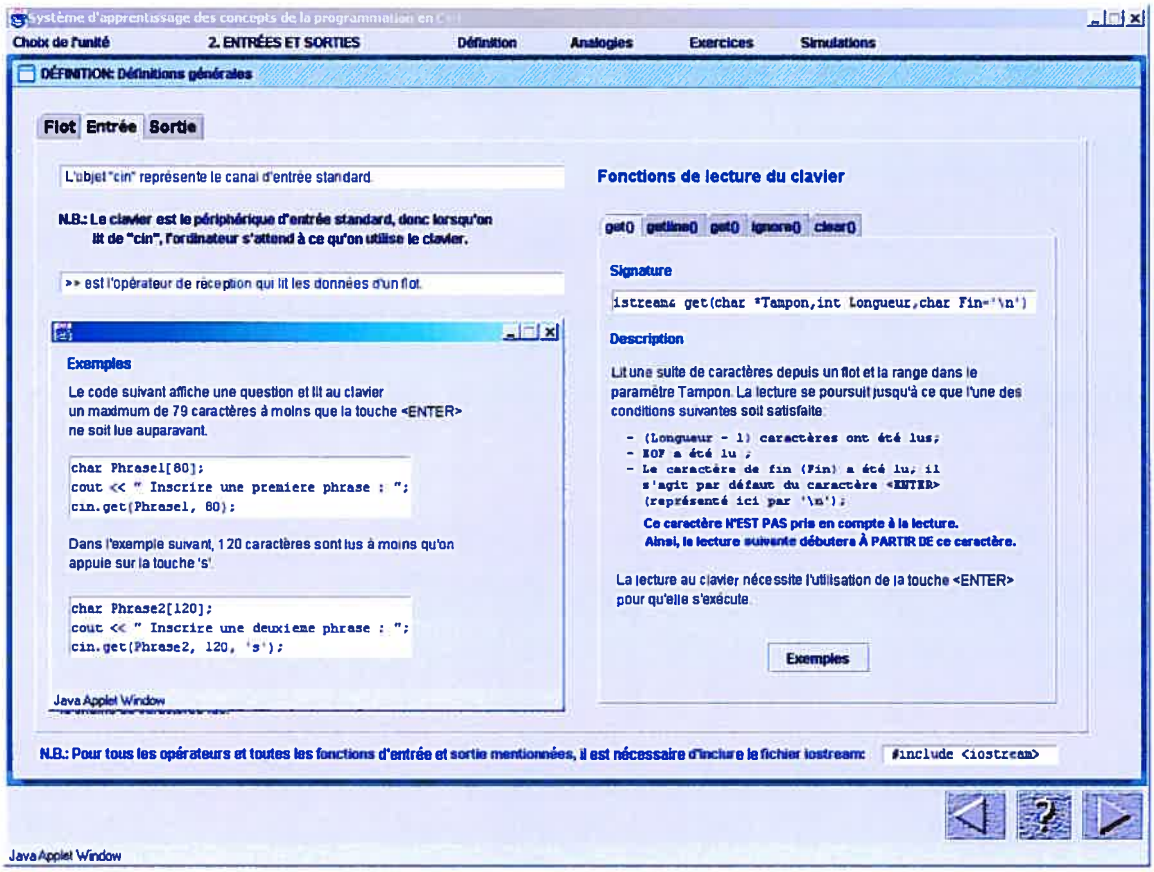


Figure 4.9 Affichage d'une information supplémentaire à l'aide d'une fenêtre

4.4.2.2 Rubrique Analogie

L'analogie est utilisée comme lien signifiant entre la connaissance acquise et la connaissance à acquérir. Pour chaque unité nous avons choisi de présenter trois analogies différentes qui ciblent certaines propriétés nécessitant une attention particulière vu leur importance ou leur difficulté d'appropriation.

Les analogies sont accessibles, une seule à la fois, à l'aide du menu déroulant ANALOGIE. Nous avons incorporé quelques interactions agissant directement sur l'objet analogique. L'étudiant peut de cette façon expérimenter à sa guise l'effet de certains paramètres sur cet analogue.

Prenons comme première analogie, celle du verre et son contenu pour expliquer la relation de la variable et son type, présentée à la figure 3.17 de la page 99. Tout d'abord, rappelons qu'une variable correspond à un emplacement mémoire où est inscrite une donnée qui peut être modifiée. La dimension de l'emplacement, la nature de la donnée et les manipulations possibles sont prescris par le type. L'analogie est celle d'un contenant et son contenu précisée par un liquide dans un verre. Conformément à la théorie de la transposition analogique de Gentner (1989) notre figure illustre les relations transposables d'un domaine à l'autre.

D'un côté, le contenu est identifié par un nom, une certaine quantité est dans le verre qui peut être à l'extrême vide ou plein. De l'autre, la variable est identifiée par un nom et correspond à une valeur qui doit être à l'intérieur des valeurs minimale et maximale de son type. Cette analogie transpose en partie la relation d'un contenant et son contenu à celle d'une variable et son type.

Une deuxième analogie, présentée à la figure 4.10, est utilisée pour expliquer le même concept. Cette fois la relation variable-type est expliquée à l'aide de l'analogie d'un téléviseur et son affichage à l'écran. Les dimensions de l'écran et son emplacement dans le téléviseur sont fixés par le modèle du téléviseur. L'affichage à l'écran est le seul élément variable. Le modèle du téléviseur est associé au type de la variable tandis que l'écran est associé à la variable dont l'affichage (la valeur) change continuellement. L'utilisation d'un objet familier comme le téléviseur nous permet de bien distinguer le rôle respectif du type et de la variable dans leur relation. La similarité entre les deux images peut favoriser la compréhension de l'étudiant. En effet, le cadre du téléviseur et le cadre noir de la variable sont tous deux définis ou imposés par leur type respectif; l'intérieur des cadres sert à inscrire des contenus qui peuvent varier.

0 1 1 0 1 1 0 1 1 0 0 1 0 1 0 1

Figure 4.10 L'analogie téléviseur/affichage avec type/variable

La troisième et dernière analogie de cette unité consiste en une transposition du diagramme classique illustrant la relation entre les ensembles de nombres à la relation entre les différents types numériques en informatique. Cette dernière analogie se distingue des deux premières par la grande proximité des deux domaines. Conscient que la proximité des domaines est un facteur important dans l'impact d'une analogie, nous avons choisi des analogies de proximité différente pour chaque unité.

4.4.2.3 Rubrique Exercices

Les exercices sont conçus de manière à exécuter adéquatement l'activité inhérente au sujet étudié en fonction de la tâche de programmation. Pour ce faire, nous proposons une procédure visant à encadrer l'étudiant dans la résolution de son exercice. Du point de vue de l'interface, une flèche précède l'étape à compléter par l'étudiant et les caractères du texte sont mis en évidence. Cette procédure s'exprime en quelques étapes où le passage à l'étape suivante nécessite la réussite de l'étape précédente. Par exemple, notre premier exercice qui consiste à déclarer une variable à partir d'une brève description s'effectue à l'aide des étapes :

1. Identifier le nom de la variable
2. Identifier le type de la variable

Une fois l'information obtenue, une évaluation est réalisée en appuyant sur le bouton correspondant. Par la suite, une fenêtre de message informe l'étudiant qu'il a répondu correctement ou non à l'exercice. Dans le cas d'une erreur, un message suffisamment précis oriente l'étudiant vers la correction à réaliser. À l'inverse, lorsque la solution est correcte, la déclaration est ajoutée à une portion de programme déjà présente à l'écran.

Cet affichage est important puisqu'il montre l'instruction résultante des étapes proposées. La figure 4.11 présente l'affichage de la fenêtre des exercices simples de l'unité Variables et Types.

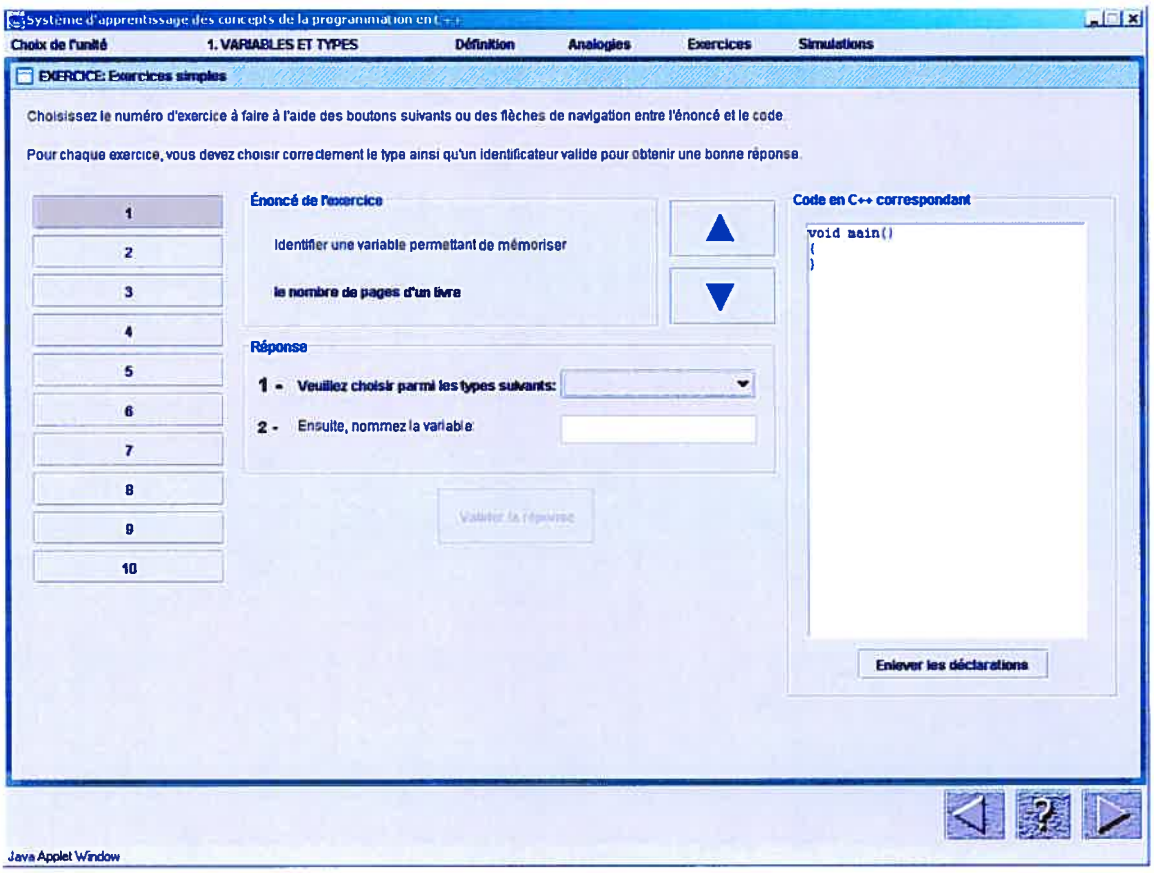


Figure 4.11 Exercices simples de l'unité Variables et Types.

Pour cette unité, un deuxième niveau d'exercices est proposé. Il s'agit d'identifier les variables nécessaires à la résolution d'un problème. L'identification des variables ne se limite pas ici à extraire certains termes de l'énoncé mais exige un commencement d'analyse afin de déduire celles qui seront nécessaires à la procédure de résolution. Le cheminement à suivre pour résoudre le problème est:

1. Déterminer le nombre de variables

Pour chacune des variables

2. Déterminer le nom de la variable et son type

La figure 4.12 présente l'interface utilisée pour implanter cette procédure. La fenêtre présente initialement trois cadres : l'énoncé, la réponse et le programme. L'énoncé est bref et précis. La réponse s'exécute selon les deux étapes décrites précédemment. Finalement, le cadre du programme affiche le résultat de la déclaration réalisée.

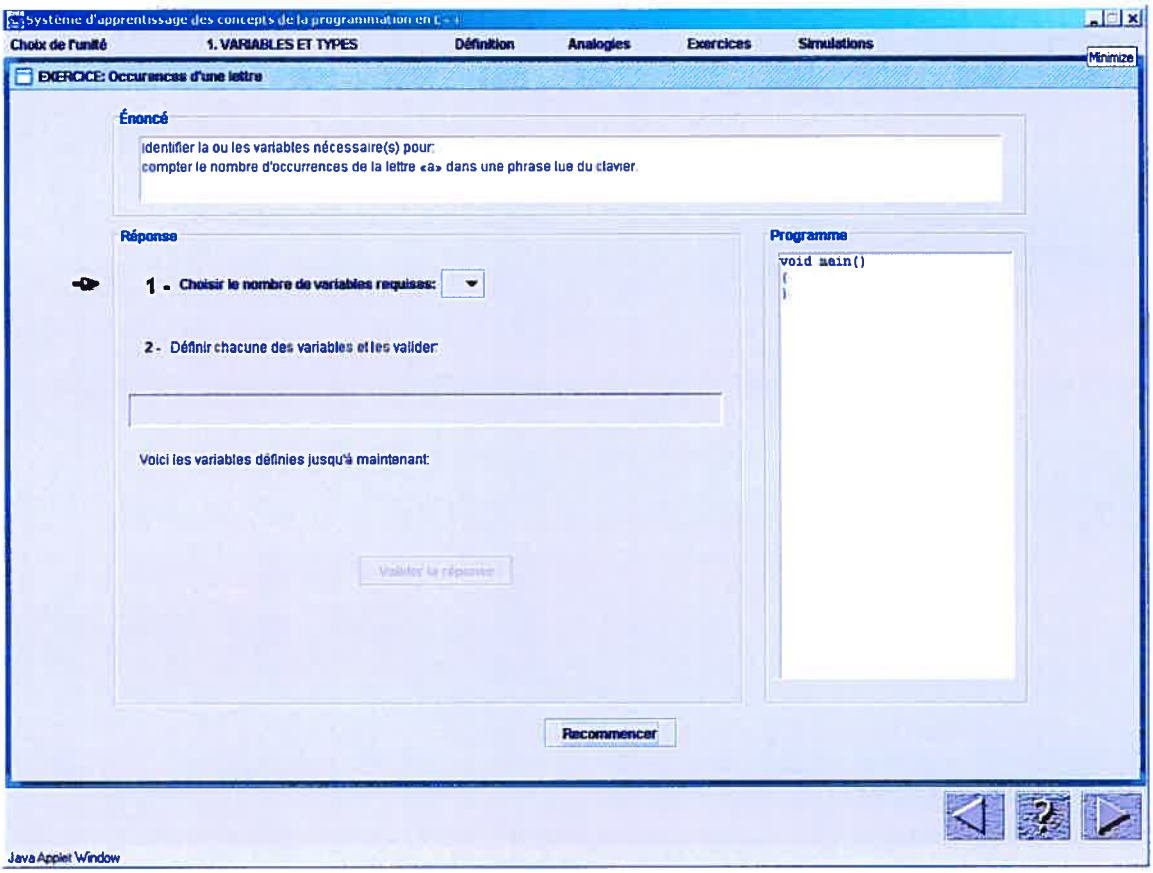


Figure 4.12 Fenêtre des exercices

Une fois le nombre de variables précisé à l'étape 1, un nombre égal de boutons portant l'inscription indéterminée est affiché dans l'encadré de l'étape 2. L'activation de chaque bouton génère une nouvelle fenêtre offrant un dialogue permettant de préciser le nom et le type de la variable. La figure 4.13 montre l'affichage de la déclaration d'une variable. Cette dernière fenêtre vérifie l'exactitude de la déclaration indépendamment du problème.

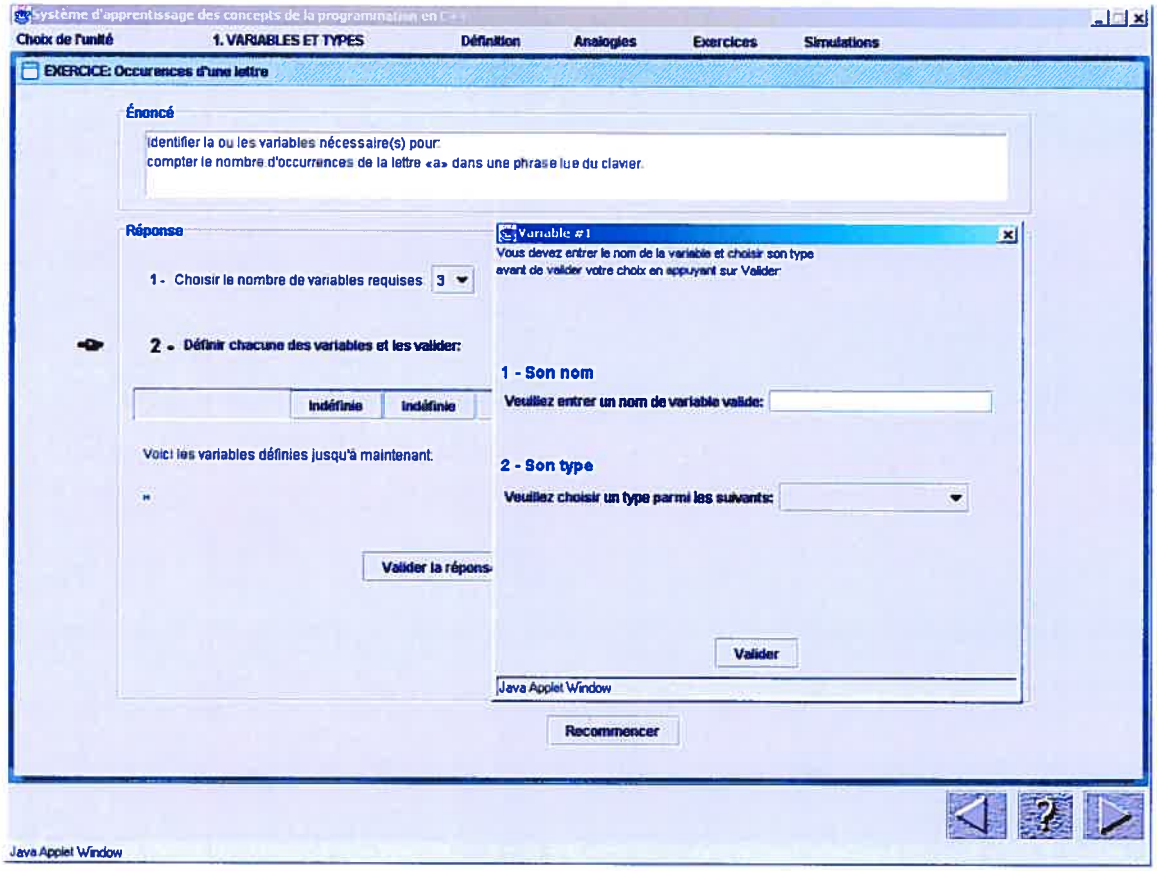


Figure 4.13 Superposition de la fenêtre de déclaration d'une variable

Lorsque toutes les variables sont déclarées, l'étudiant doit valider sa solution en appuyant sur le bouton correspondant. La validation vérifie le nombre et l'adéquation des déclarations avec l'énoncé du problème. Une fenêtre de message précise l'erreur en nombre ou en type de la déclaration et le bouton associé à la variable mal déclarée est affiché en rouge. L'étudiant possède alors plusieurs éléments d'aide à la correction. Il peut ultimement afficher la solution à l'aide d'un bouton prévu à cette fin. La figure 4.14 présente l'interface de l'environnement après l'activation du bouton « Valider la réponse ».

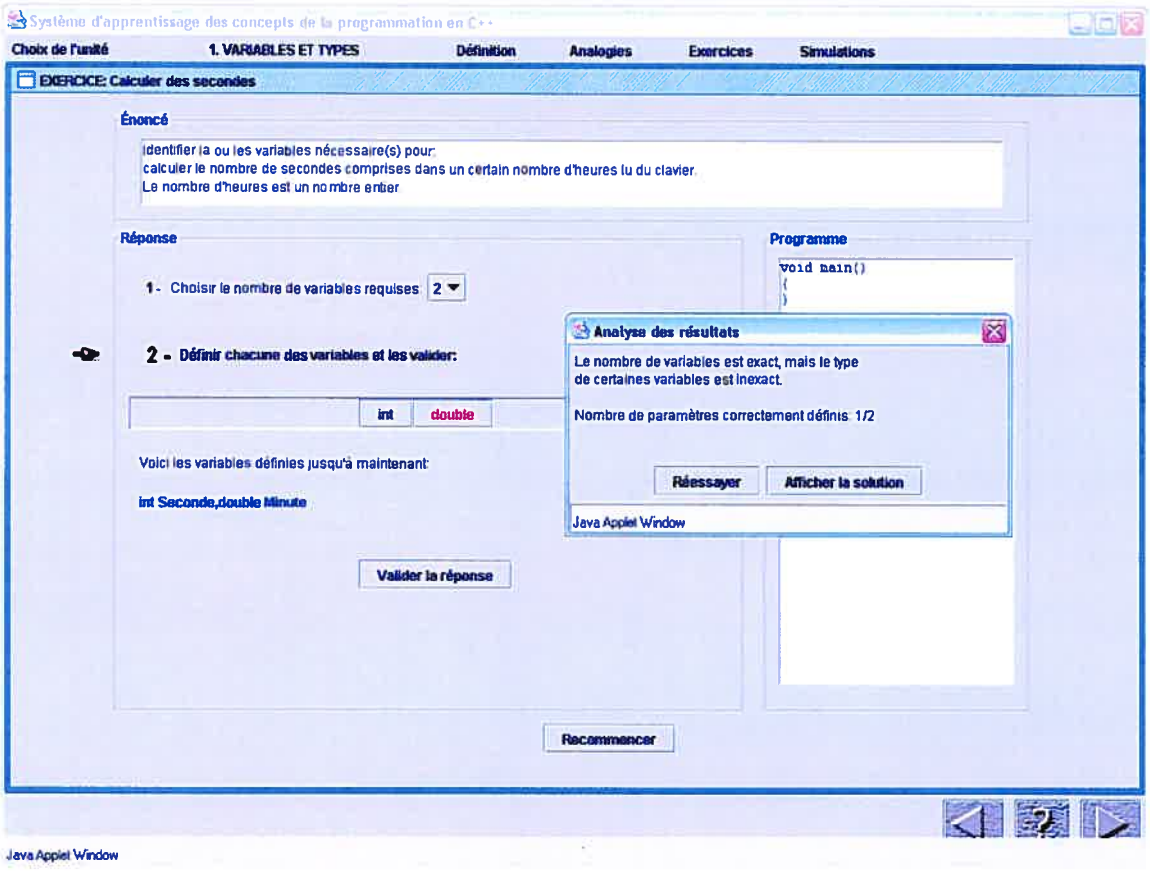


Figure 4.14 Valider la réponse

Pour l'ensemble des exercices, l'environnement offre un encadrement serré afin d'éviter la poursuite d'une solution erronée. Pour ce faire, à la fin de chaque étape, une procédure de validation est imposée et doit être réussie pour permettre le passage à l'étape subséquente. Conscient de l'importance de cette rétroaction dans le processus d'apprentissage de l'étudiant, notre stratégie a été développée de manière à faciliter le repérage d'erreur et nous permettre d'adresser un message extrêmement précis. La figure 4.15 présente un échantillon de ces messages d'erreur.

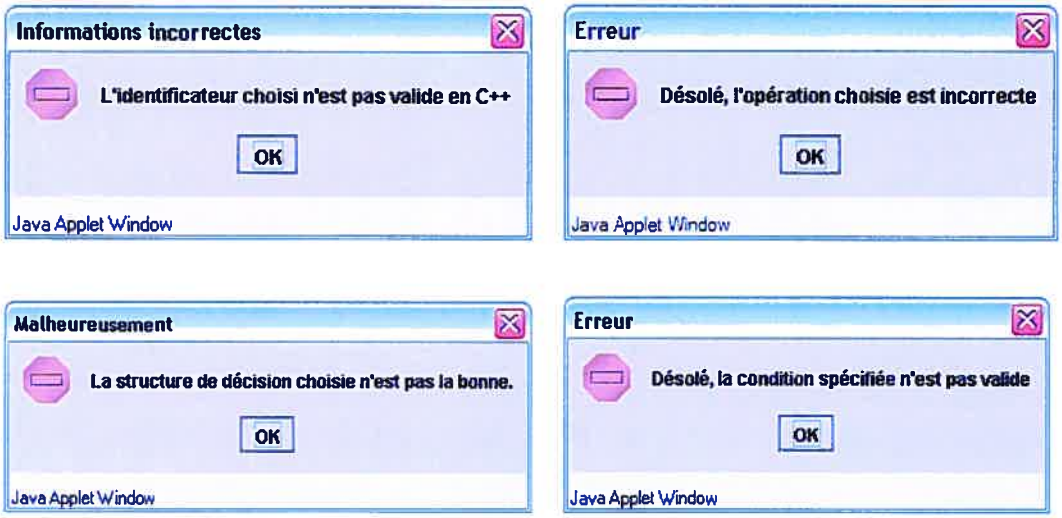


Figure 4.15 Messages d'erreur

4.4.2.4 Rubrique Simulations

La simulation permet d'expliquer le fonctionnement d'un programme instruction par instruction. Pour ce faire, la fenêtre présente simultanément trois vues distinctes auxquelles s'ajoutent un cadre où sont décrits les actions réalisées par l'instruction en cours d'exécution. La figure 4.16 présente la fenêtre de la simulation.

La première vue présente le programme. L'étudiant peut l'exécuter selon deux modes différents soit pas à pas ou automatiquement. Dans les deux modes l'exécution s'effectue en respectant l'ordre des instructions, la différence réside dans le délai d'attente avant l'exécution d'une instruction. Le mode pas à pas permet à l'étudiant d'enclencher, au moment voulu, l'exécution d'une instruction. Il peut regarder tant qu'il le désire l'affichage résultant d'une instruction avant d'enclencher l'exécution de la prochaine. Le mode automatique exige de préciser un temps en milliseconde compris entre 100 et 5000. Ce délai sera respecté avant l'exécution de chacune des instructions. Le choix du délai détermine, dans une certaine mesure, la vitesse d'exécution du programme et impose par le fait même la vitesse de lecture de l'information. Pour les deux modes, l'instruction exécutée est mise en évidence par l'ajout d'une barre bleu à la ligne correspondante.

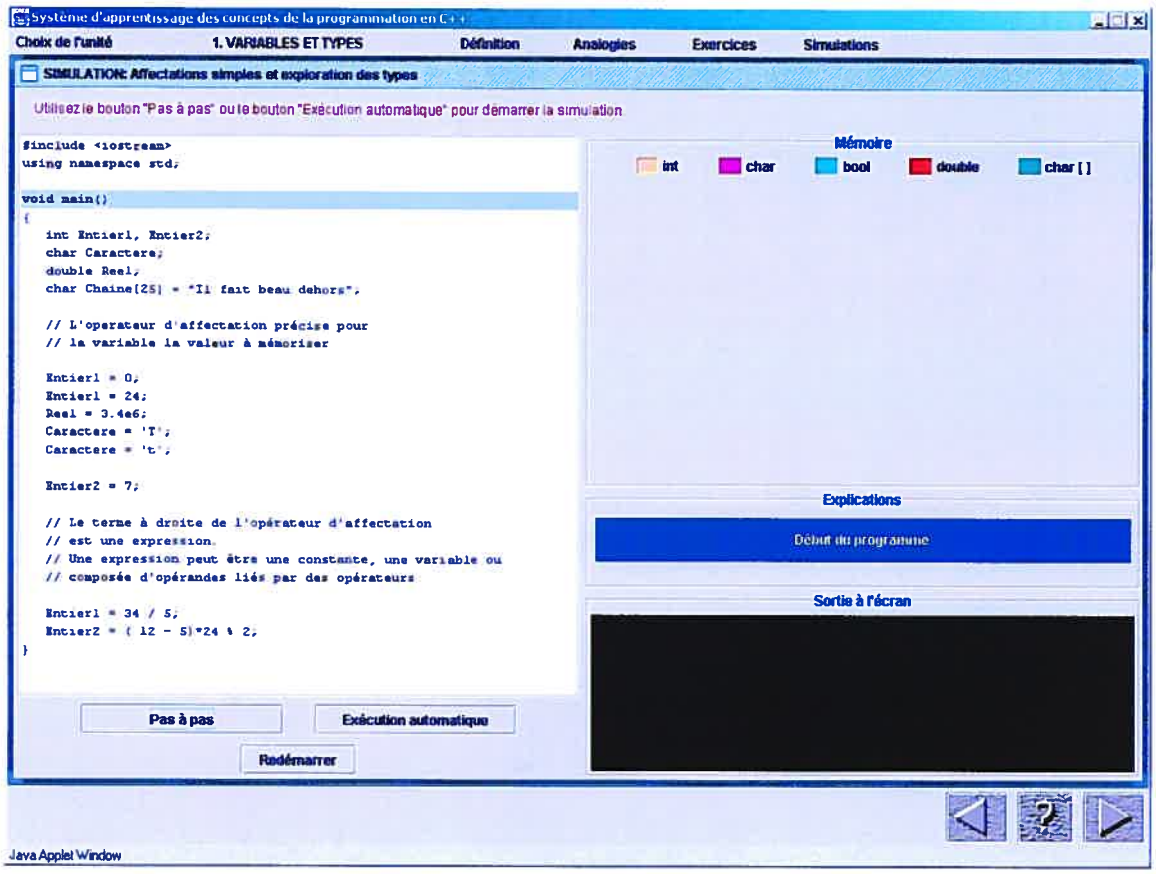


Figure 4.16 Fenêtre de simulation

La deuxième vue présente les différentes variables utilisées par le programme. Cette vue est associée à la mémoire puisque les valeurs des variables s'y retrouvent. Nous avons prévu un code de couleur afin de distinguer aisément les différents types de variable. Pour chacune des variables apparaîtra dans cette vue, une case portant le nom de celle-ci avec l'arrière plan de la couleur correspondant au type. Lorsqu'une instruction modifie la valeur d'une variable nous soulignons cette modification en changeant l'arrière plan du contenu de cette variable, de sorte à attirer l'attention. Cette vue est d'une grande importance pour l'apprenant puisqu'elle révèle une face cachée de l'exécution d'un programme. La figure 4.17 présente l'état de la vue mémoire après l'exécution des instructions de déclaration.

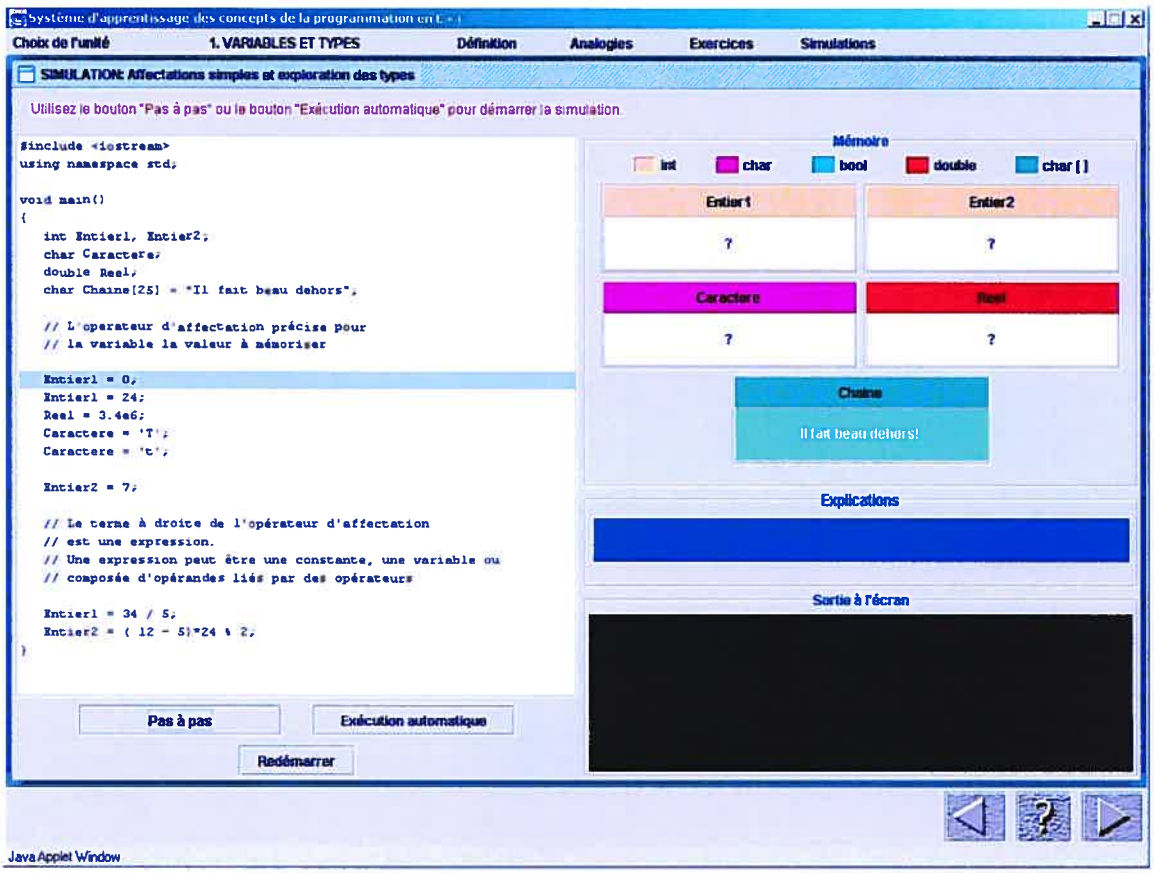


Figure 4.17 Ajout des cases associées aux variables dans la vue mémoire

La troisième vue correspond à l'affichage obtenu lors de l'exécution du programme. Lorsque le programme contient des instructions de lecture, les valeurs entrées au clavier y seront également affichées. Cette vue constitue le principal objectif de la programmation de l'étudiant, la concrétisation du résultat attendu.

4.4.2.5 Contenu

La matière est subdivisée en dix unités dans l'environnement d'apprentissage. Chaque unité traite d'une notion spécifique. Ces dix unités se nomment respectivement :

Variables et types

Entrées et sorties

Fichiers textes
Structure de décision
Structure de répétition
Énumération et tableau
Transmission de paramètres
Enregistrements
Fichiers binaires
Allocation dynamique

Nous avons colligé toute l'information propre à chacune des unités en examinant les contenus des quatre rubriques. Le tableau 4.1 présente une synthèse du contenu de l'environnement. Chaque ligne du tableau concerne précisément une unité. La colonne définition énumère les différentes notions présentées, la colonne analogie décrit brièvement le domaine source, la colonne exercice précise la quantité et décrit les opérations exigées, la colonne simulation identifie la ou les notions abordées dans les programmes.

Tableau 4.1 Contenu de l'environnement informatisé

Module	Définitions	Analogies	Exercices	Simulations
Variables et types	• Variable • Constante • Type • Identificateur	• Un verre et son contenu • Téléviseur et son écran • Ensembles de nombres et les types numériques	10 exercices • Déclaration d'une seule variable 10 exercices • Déclaration de plusieurs variables	• Affectation simple • Permutation • Conversion de type • Manipulation de chaîne de caractères
Entrées et sorties	• Flot • Fonctions d'entrée • Fonctions de sortie	• Partition musicale • Mot à plusieurs sens • Edition de texte	4 exercices • Déclaration de variables • Instructions de lecture	• Lectures simples • Lectures avancées • Format d'affichage
Fichiers textes	• Description • Spécification • Utilité • Structure • Variable fichier • Fonctions propres au fichier	• Déplacement d'un train sur le rail • Construction du rail • Affichage a l'écran • Contrôle d'un magnéto.	5 exercices • Ouverture • Lecture ou écriture	• Lecture d'un fichier • Lecture d'un fichier et écriture dans un autre • Fonctions d'E/S appliquées à un fichier
Structure de décision	• Description • Instruction simple • Instruction composée • Expression booléenne • Opérateurs booléens • if, if-else, switch	• Répondre au téléphone • Préparation à l'examen • Podium d'une épreuve sportive	5 exercices • Choisir la structure adéquate • Compléter l'expr. booléenne de la structure	• Utilisation successive des trois structures • Imbrications if-else • Structure switch possédant des énoncés if-else
Structure de répétition	• while • do-while • for • Comparaison des structures	• Faire le plein d'essence • Demander la permission de décollage • Distribuer des copies d'examen	6 exercices • Choisir la structure adéquate • Compléter l'expr. booléenne de la structure	• Utilisation successive des trois structures • Lecture complète d'un fichier • Imbrications de structures

Module	Définitions	Analogies	Exercices	Simulations
Énumération et tableau	Énumération • Énoncé enum • Valeur ordinale • Exemple • Lecture et affichage • Opérateur relationnel • Dans une structure de décision • Dans une structure de répétition Tableau • Déclaration • Dimension • Opérations • Accès aux éléments • Indice de type énumération	• Grille de pointage d'une partie de quilles • Un classeur • Les boutons radio	9 exercices • Choisir entre énumération ou tableau • Compléter l'expression	• Utilisation d'une énumération • Utilisation d'un tableau • Opérations entre deux tableaux
Transmission de paramètres	• Transmission par valeur • Transmission par adresse	• Photocopie • Vacancier vs messager • Fonction math.	10 exercices • Choisir le nombre de paramètres • Déclarer chaque paramètre	• Permutation transmission par valeur • Permutation transmission par adresse • Fonction avec les deux modes de transmission

Module	Définitions	Analogies	Exercices	Simulations
Enregistrements	• Description • Déclaration • Initialisation • Accès aux champs • Opérations • Exemples	• Adresse postale • Observation météo • Chanson préférée	4 exercices • Déterminer le nombre de champs • Déclarer chaque champ 5 exercices • Sélectionner l'opération nécessaire (nature et nombre) • Compléter l'opération	• Comparaison des coordonnées de deux points • Tableau d'enregistre-ments • Utilisation d'une fonction avec un paramètre enregistre-ment
Fichiers binaires	• Description • Déclaration • Fonctions • Lecture • Écriture • Positionnement • Position	• Lecteur CD • Remplacement d'une brique • Recherche d'une étoile dans le ciel	10 exercices • Définir la var. fichier • Choisir et définir les var. nécessaires au traitement du fichier • Accès au fichier • Lecture ou écriture • Fermeture	• Création-lecture-mise à jour d'un fichier binaire • Manipulation d'un fichier binaire d'enregistre-ments • Recherche dans un fichier binaire
Allocation dynamique	• Variable pointeur • Déclaration • Opérateur & • Opérateur new • Opérateur delete	• Association du nom et du surnom • Chasse au trésor • Ajouter ou retirer un anneau d'une chaîne en or	5 exercices • Déclaration et initialisation • Attribution de la mémoire 4 exercices • Manipulation d'une liste chaînée	• Pointeurs simples • Permutation des adresses de deux pointeurs et permutation des contenus référés par deux pointeurs • Création et affichage d'une liste • Retrait d'un élément d'une liste

Chapitre 5

MISE À L'ESSAI ET RÉSULTATS

Rappelons que nous avons réalisé notre expérimentation dans le cadre du cours ING1025 Informatique de l'École Polytechnique de Montréal à l'automne 2002. Ce cours enseigne les rudiments de la programmation basés sur une méthode de résolution de problèmes. Nous avons choisi le groupe d'étudiants en génie logiciel principalement parce que chaque étudiant possède un ordinateur portable. L'organisation de l'atelier est ainsi grandement simplifiée. Dix-sept étudiants sur une possibilité de quarante-quatre ont accepté volontairement de réaliser l'atelier à la fin de la session.

Nous présentons dans ce chapitre les résultats obtenus de la réalisation des deux exercices de l'atelier. Dans un premier temps, nous analyserons les programmes produits lors de l'atelier. Dans un deuxième temps, nous présenterons les résultats d'un questionnaire soumis après la réalisation de l'atelier.

5.1 Évaluation des programmes produits lors de l'atelier

Les étudiants sont invités, à titre de volontaire, à participer à une activité dans une salle de l'école qui consistera à réaliser des programmes basés sur un montage robotisé. Ils sont informés que cet exercice vise à déterminer la faisabilité de tels exercices dans le cadre du cours et correspond à un sujet de recherche du professeur.

Un rappel est transmis par courriel à l'ensemble des étudiants du cours. À ce message est attaché un fichier compressé nommé `Atelier.zip` contenant les fichiers:

Serial.h	Déclaration de la classe CSerial
Serial.cpp	Source des fonctions de la classe CSerial
StdAfx.h	Déclaration utilisée par la classe CSerial
StdAfx.cpp	Source du fichier StdAfx.h
Utilitaires.h	En-têtes des fonctions utilitaires
Utilitaires.cpp	Source des fonctions utilitaires
dllCSerial.lib	Bibliothèque des fonctions utilitaires
dllCSerial.dll	Exécutable des fonctions utilitaires

Un deuxième fichier nommé `EspaceTravail.doc` est attaché au message. Ce fichier contient l'information pour organiser l'espace de travail du projet à créer dans l'environnement de développement Visual C++. Le contenu du fichier est reproduit à l'annexe B.

Une période de deux heures est prévue pour réaliser les exercices de l'atelier. Les étudiants sont invités à former des équipes de trois individus. Après s'être assuré que l'environnement de développement était correctement organisé, le professeur distribue à chaque équipe une plaque d'essai, un bloc d'alimentation et un câble série.

Les étudiants sont alors prêts pour l'activité de programmation. Pour le premier exercice, ils doivent déterminer l'ordre d'ouverture des luminodiodes en exécutant le programme prévu à cet effet. Une fois, l'ordre déterminé ils doivent rédiger un programme qui reproduira ce fonctionnement à l'aide des fonctions mises à leur disponibilité. Une fois ce premier programme terminé, les étudiants s'attaquent au deuxième exercice. Toujours en exécutant le programme fourni, les étudiants doivent déterminer le courant nécessaire pour démarrer le moteur ainsi que celui qui l'arrêtera. Après que les observations effectuées sur le montage eurent permis d'obtenir les valeurs désirées, les étudiants rédigent un deuxième programme qui permettra de retrouver ces valeurs. Les programmes sont reproduits à l'annexe G.

A la fin de la séance de travail, nous avons demandé aux étudiants de nous remettre le programme rédigé. Nous avons évalué les documents reçus selon certaines caractéristiques et certaines métriques statiques du logiciel. Les métriques utilisées sont : le nombre de lignes de code (LOC), le niveau d'imbrication des structures de

contrôle (NISC), le nombre de structures de décision et le nombre de structures de répétition. Le niveau d'imbrication des structures de contrôle sera à un si le programme ne contient qu'une seule structure de contrôle. Il sera à deux, si une structure de contrôle en contient une autre, par exemple, une structure if dans une structure while. Une précision sur la formulation de l'expression booléenne s'ajoute à ces métriques. Nous qualifions une expression booléenne de simple lorsqu'elle se compose de deux opérandes et d'un opérateur relationnel (=, <, >, etc.) ou d'une variable booléenne. Nous qualifions une expression booléenne de composée lorsqu'elle contient un opérateur booléen (ET, OU, NOT, etc.). Notons que l'expression composée incorporera nécessairement deux expressions simples. À cette information objective, nous ajoutons quelques remarques subjectives concernant l'exactitude du travail par rapport au travail demandé et les anomalies de programmation.

5.1.1 Évaluation du premier exercice : ouverture des luminodiodes

Les résultats de l'analyse des programmes concernant le premier exercice sont présentés au tableau 5.3. Cet exercice qui consiste à imposer un ordre d'ouverture des luminodiodes se résout par la transposition de la logique déduite par le groupe dans le langage informatique. Rappelons que les étudiants sont dans un premier temps à la recherche de cette logique en manipulant le système. Le raisonnement s'exprimera ici tout naturellement sous la forme d'énoncés SI-ALORS. Le défi est d'agencer ces énoncés de manière à définir ou imposer un ordre d'ouverture des luminodiodes. Nous retrouvons dans les programmes réalisés par les étudiants, à l'exception d'un seul, cette logique conditionnelle. Deux programmes présentent une structure imbriquée sous forme d'un énoncé SI-ALORS contenant des instructions correspondant à des énoncés SI-ALORS. Un programme présente une structure séquentielle de plusieurs énoncés SI-ALORS. Deux autres programmes utilisent une abstraction sous forme d'un tableau contenant l'ordre d'ouverture de luminodiodes qui est vérifié par un énoncé SI-ALORS.

Groupe 1

L'ordre d'allumage des luminodiodes est imposé par une structure imbriquée composée d'énoncés SI-ALORS. La composition de la structure montre un raisonnement basé sur l'énumération d'une séquence de situations pouvant survenir. Par exemple, si l'usager n'allume pas initialement la lumière blanche que se passera-t-il ? Qu'arrivera-t-il si l'usager n'allume pas en deuxième la lumière rouge ? Pourra-t-il ensuite allumer la lumière verte ? Nous utilisons dans notre exemple la négation puisque les expressions booléennes comprises dans les énoncés SI-ALORS sont toutes formulées à l'aide de l'opérateur d'inégalité. Il nous apparaît évident qu'une séquence similaire a pu être expérimenté par l'entremise du montage et par le fait même orienter le groupe vers la solution proposée. La structure imbriquée composant l'ordre d'allumage des luminodiodes est inscrite dans une structure de répétition de sorte à permettre à l'usager plusieurs essais.

Le programme comprend 68 lignes de code, soit le plus volumineux, et est celui ayant le niveau d'imbrication le plus élevé avec 5. Le premier niveau prend en charge les essais de l'usager tandis que les quatre autres traitent l'allumage des quatre luminodiodes. Le grand nombre de lignes de code s'explique par une fonctionnalité non implantée par les autres groupes et la répétition des mêmes instructions quatre fois. La fonctionnalité supplémentaire consiste à éteindre les trois premières luminodiodes lorsque le choix de la quatrième est erroné. Cette fonctionnalité, quasi inutile, nécessite sept instructions.

Nous attribuons à ce programme une note d'appréciation de 15 sur 20.

Groupe 2

Le programme du groupe 2 est similaire à celui du groupe 1. La structure imbriquée est identique sauf que les expressions booléennes sont formulées plutôt pour tester l'égalité. Curieusement, ce groupe ne permet pas à l'usager d'effectuer plusieurs essais. Le programme se termine dès que l'usager commet une erreur. Pour un second essai,

l'usager doit exécuter de nouveau le programme. Ce manquement ou cette carence dans le programme se traduit par un raisonnement basé sur un scénario unique.

Le programme compte 45 lignes de code et possède un niveau d'imbrication de 4. Les quatre niveaux sont utilisés pour l'allumage des quatre luminodiodes. Les expressions booléennes sont simplement composées à l'aide de l'opérateur relationnel d'égalité.

De ces constatations, nous attribuons à ce programme une note d'appréciation de 10 sur 20.

Groupe 3

Le groupe 3 a rédigé un programme qui allume et éteint successivement les quatre luminodiodes à 10 reprises. L'effet est similaire à une guirlande de lumières clignotantes. Cela a sûrement amusé les membres du groupe mais ne correspond malheureusement pas à l'exercice.

Nous accordons une note de 5 sur 20 principalement pour l'effort.

Groupe 4

Le programme du groupe 4 présente une structure séquentielle constituée de quatre énoncés IF (sans ELSE) successifs incluse dans une structure de répétition DO-WHILE. Cette formulation est plus simple et plus souple que la structure imbriquée utilisée par les deux premiers groupes. La condition d'allumage d'une luminodiode se compose d'une expression booléenne vérifiant l'allumage de la luminodiode précédente selon l'ordre imposé. Pour ce faire, l'état d'une luminodiode, soit allumée ou éteinte, est mémorisé à l'aide d'une variable booléenne. L'abstraction de cette information facilite la transposition de la solution dans le langage informatique.

Le programme compte 37 lignes de code et possède un niveau d'imbrication de 2. Le premier niveau gère les essais de l'usager et le deuxième vérifie l'ordre d'allumage des luminodiodes. Le programme contient des expressions booléennes simples et composées. Les expressions simples sont formulées à l'aide de l'opérateur relationnel d'égalité. Les expressions composées sont formulées à l'aide de l'opérateur booléen ET avec comme opérandes une variable booléenne et une expression simple.

Nous attribuons à ce programme la respectable note de 17 sur 20.

Groupe 5

Le groupe 5 nous présente un programme typique d'étudiants ayant une certaine expérience en programmation. La stratégie de solution est basée sur l'abstraction à l'aide d'une chaîne de caractères et sa manipulation. Le choix de cette entité par le groupe peut s'expliquer par l'existence de plusieurs fonctions prédéfinies agissant sur les chaînes de caractères. Le programme exploite les fonctions : `strlen()`, `strcpy()`, `strstr()`, `strlwr()` et `strcat()`. Par contre, cette emphase sur la rédaction du programme s'effectue au dépend d'une bonne compréhension de l'exercice. En effet, le programme vérifie si le choix de luminodiode est correct et s'assure d'allumer au plus quatre luminodiodes sans imposer un ordre d'ouverture! Le programme contient, à notre grand désespoir, une structure de répétition DO-WHILE dont le critère d'arrêt est la valeur booléenne « true ». En d'autres mots, il s'agit d'une boucle infinie dont l'interruption est prévue par une sortie en catastrophe correspondant à un énoncé « break » exécuté lorsqu'une certaine situation se produit. Cette structure est une entrave aux règles de la programmation structurée. Nous retrouvons ici, le cas typique de l'étudiant qui sait déjà programmer et qui doit apprendre à bien programmer.

Le programme compte 40 lignes de code et possède un niveau d'imbrication de 3. Le premier niveau permet de réaliser plusieurs essais, le deuxième niveau vérifie le nombre de luminodiodes allumées et le troisième sert à vérifier la condition d'interruption des essais. Le programme contient trois expressions booléennes utilisant un opérateur

relationnel et une expression booléenne composée formulée avec l'opérateur booléen ET possédant deux opérandes qui sont des expressions simples.

Nous attribuons à ce programme la note d'appréciation de 10 sur 20.

Groupe 6

Dans le programme du groupe 6, la logique imposant l'ordre d'ouverture des luminodiodes utilise efficacement une abstraction sous la forme d'un tableau. La première entrée du tableau contient le numéro de la première luminodiode à allumer, la deuxième entrée contient le numéro de la deuxième luminodiode à allumer, etc. Un compteur du nombre de luminodiodes allumées sert astucieusement d'indice au tableau. De cette façon, le choix de l'usager sera comparé à la première entrée dans le tableau. Si ce choix est correct, le nombre de luminodiodes est augmenté de un de manière à permettre la comparaison avec la deuxième luminodiode au prochain essai, sinon il demeurera le même. Plus simplement, cette stratégie peut s'énoncer: tant que le choix de l'usager n'est pas correct, lui demander un nouveau choix. On peut facilement associer cette dernière phrase aux manipulations réalisées par le groupe pour deviner l'ordre d'ouverture des luminodiodes.

Le programme compte 39 lignes de code et un niveau d'imbrication de 3. Le premier niveau gère les essais de l'usager, le deuxième est tout à fait inutile puisqu'il gère également les essais et le troisième sert à tester le choix de luminodiode à allumer. En réalité, ce programme devrait nécessiter un niveau d'imbrication de 2. Les expressions booléennes, au nombre de quatre, sont bien formulées. Trois sont des expressions simples et une est composée à l'aide de l'opérateur booléen ET. Ce programme se distingue de tous les autres par l'utilisation d'une fonction dans laquelle sont confinées les instructions pour l'allumage des luminodiodes.

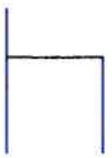
Nous attribuons une note d'appréciation de 17 sur 20 à ce programme.

Le tableau 5.1 présente un résumé des données déduites de l'analyse du premier exercice de l'atelier.

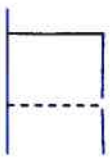
Tableau 5.1 Programme réalisant l'ouverture des luminodiodes selon un ordre précis

Groupe	LOC	NISC	Structure de décision	Structure de répétition	Expression booléenne	Remarque	Note/20
1	68	5	4 if-else	1 do-while	Simple	Répétition des mêmes instructions	15
2	45	4	4 if-else	Aucune	Simple	Exécution séquentielle sans répétition	10
3	22	1	0	1	Simple	Ouvre et ferme successivement les luminodiodes	5
4	37	2	4 if	1 do-while	Simple Composée		17
5	40	3	2 if 2 if-else	1 do-while	Simple Composée	Utilisation abusive de l'énoncé break Manipulation de chaînes de caractères.	10
6	39	3	1 if 1 if-else	1 while 1 do-while	Simple Composée	Str. do-while est inutile	17

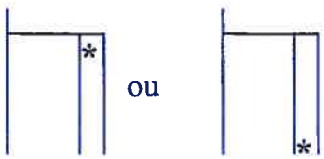
La figure 5.1 présente un diagramme correspondant au flux de contrôle de chaque programme. Cet outil visuel montre astucieusement l'agencement et l'imbrication des structures dans le programme. Dans ce diagramme les structures de contrôle de répétition ou de décision sont illustrées par les schémas respectifs suivants :



Structure de décision IF



Structure de décision IF-ELSE



Structures de répétition

Notons que l'astérisque « * » sert à préciser l'emplacement de l'expression booléenne, communément appelée la condition de sortie ou le critère d'arrêt, dans la structure de répétition.

Les graphiques de la figure 5.1 déduits des programmes réalisés par chaque groupe montrent des conceptions différentes. Ces graphiques nous révèlent clairement l'importance du niveau d'imbrication. En effet, plus la structure schématique s'étend en largeur, plus la logique se complexifie et plus le danger d'erreur est grand. La structure schématique nous révèle également l'habileté à transposer la solution dans le langage de programmation. Une structure simple s'associe à une logique simple qui dévoile généralement un meilleur niveau de raisonnement et de programmation informatique.

Les groupes 1 et 2 ont opté pour des stratégies similaires mais différentes. Outre l'omission pour le groupe 2 d'une structure répétitive, nous observons dans les deux cas une imbrication de quatre structures conditionnelles IF-ELSE. Par contre, le groupe 1 emboîte les structures selon les énoncés ELSE tandis que le groupe 2 emboîte les structures selon les énoncés IF. De cette façon le groupe 1 raisonne en vérifiant si une condition est satisfaite et à l'opposé le groupe 1 raisonne en vérifiant si une condition n'est pas satisfaite. Les groupes 4 et 5 ont implanté des stratégies très proches l'une de l'autre selon la structure schématique obtenue mais en réalité fort différentes selon la solution implantée. Nous observons pour le groupe 4 l'utilisation de quatre structures décisionnelles successives et trois structures décisionnelles successives pour le groupe 5. Le groupe 4 se limite à la structure simple IF pour vérifier l'ouverture correcte de chacune des quatre luminodiodes. Le programme du groupe 5 n'impose pas d'ordre d'ouverture des luminodiodes. Ce groupe utilise en plus de deux structures IF, une structure IF-ELSE imbriquée dans une structure IF-ELSE servant de critère d'arrêt à la structure répétitive englobante. Nous obtenons du groupe 6 un diagramme présentant deux structures de répétition dont une est redondante. En ignorant cette anomalie, ce groupe traite efficacement la vérification de l'ordre d'allumage des luminodiodes, principalement parce qu'ils ont utilisé une structure abstraite adéquate dans l'élaboration de leur solution.

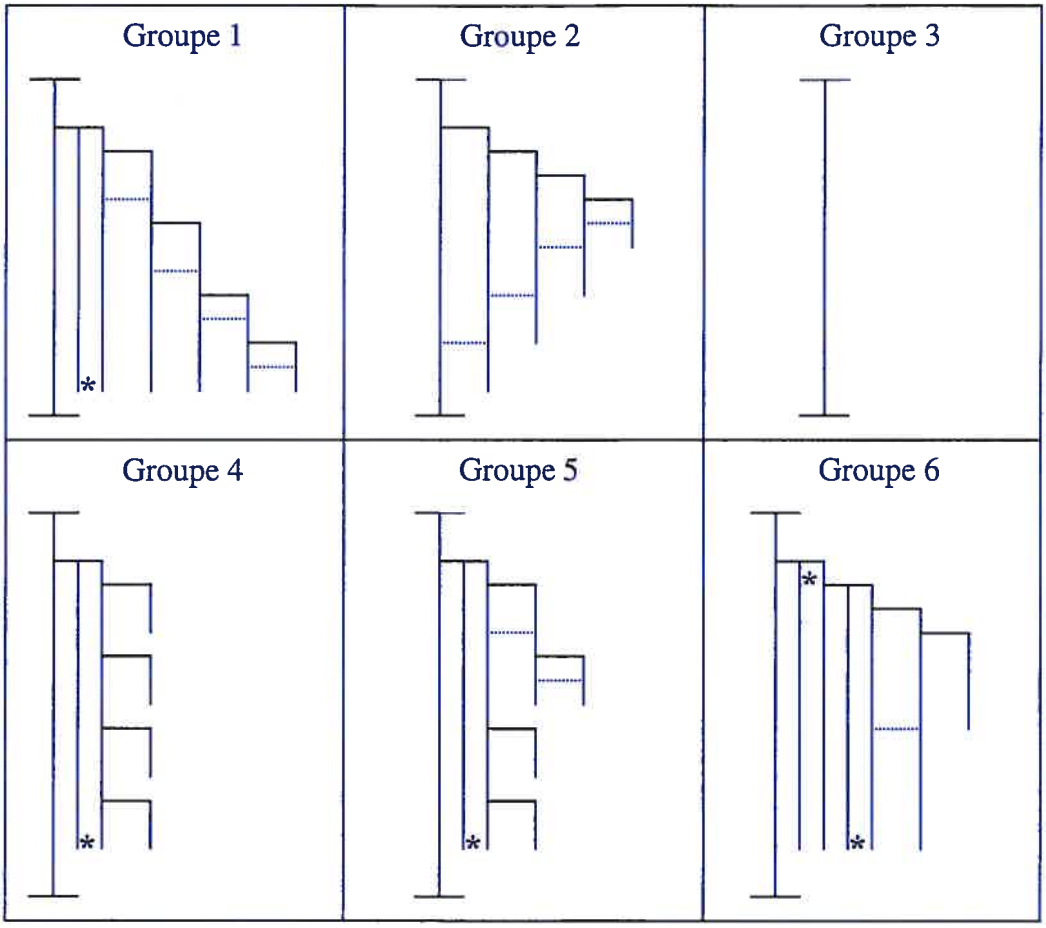


Figure 5.1 Diagramme de flux de contrôle des programmes

5.1.2 Deuxième exercice : démarrage et arrêt d'un moteur

L'exercice consistait à trouver le voltage nécessaire pour démarrer le moteur et son opposé, le voltage nécessaire pour arrêter le moteur. Notons que le voltage nécessaire au démarrage est supérieur au voltage d'arrêt étant donné l'inertie du moteur. Les étudiants manipulent initialement le système de façon à découvrir ce phénomène ainsi que les opérations à automatiser.

Groupes 1 et 2

Malheureusement, ces deux groupes n'ont pas remis de programme concernant cet exercice.

Groupe 3

Le programme permet le démarrage et l'arrêt du moteur. Il utilise successivement deux structures de répétition. La première structure de répétition impose des voltages augmentés à chaque tour d'un pas unitaire jusqu'au démarrage du moteur. La deuxième diminue le voltage d'un pas unitaire à chaque tour jusqu'à l'arrêt du moteur. Cette stratégie est en tout point similaire à la manipulation effectuée pour trouver manuellement ces deux valeurs.

Le programme compte 34 lignes de code et un niveau d'imbrication de 1. Les deux expressions booléennes incluses dans les structures de répétition sont formulées à l'aide de l'opérateur booléen OU dans un cas et ET dans l'autre. Malheureusement, les résultats obtenus de ce programme ne seront pas exacts (en excès de un) puisque l'instruction de lecture de la vitesse précède celle d'activation du moteur. En clair, il aurait fallu plutôt inscrire l'instruction d'activation du moteur et la faire suivre de l'instruction de lecture de la vitesse. Cette erreur se retrouve dans chaque structure.

Nous attribuons à ce programme une note d'appréciation de 18 sur 20.

Groupe 4

Le programme du groupe 4 ne réalise que le démarrage du moteur. La structure de répétition utilisée est bien conçue et comprend une expression booléenne formulée à l'aide de l'opérateur booléen OU. La structure impose des intensités augmentées par pas unitaire à chaque répétition.

Le programme compte 19 lignes de code et un niveau d'imbrication de 1. L'ordre des instructions dans le corps de la structure de répétition est correct. La simplicité et l'exactitude de cette structure nous incitent à penser que la seconde fonctionnalité est absente du programme uniquement pour une question de temps.

Nous attribuons une note d'appréciation de 10 sur 20.

Groupe 5

Notre groupe de programmeurs « avertis » nous propose un programme contenant trois structures de répétition FOR successives. La première structure de répétition recherche le voltage de démarrage en utilisant un pas d'incrément de 5. Une fois le moteur démarré, le voltage nécessaire est mémorisé. Comme la réponse obtenue risque d'être excédentaire, le moteur est arrêté à la sortie de la structure de répétition en lui transmettant un voltage nul. La deuxième structure de répétition suit et débute avec un voltage inférieur de cinq à celle mémorisée dans la première. Les instructions de la deuxième structure se répète en utilisant cette fois un pas unitaire. Lorsque le moteur démarre une seconde fois, le voltage de démarrage est obtenue et le passage à la troisième structure de répétition s'effectue. Cette dernière recherche le voltage d'arrêt en utilisant aussi un pas unitaire.

Le programme compte 39 lignes de code et un niveau d'imbrication de 2. Comme pour le premier exercice, le programme commet une entrave aux bonnes règles de la programmation structurée par l'utilisation d'une sortie en catastrophe. Le niveau d'imbrication pourrait facilement se réduire à 1 en éliminant cette mauvaise sortie. Le programme présente deux autres anomalies. La première consiste à l'omission d'initialiser une variable et la deuxième anomalie est l'instruction inutile i=i.

Nous attribuons à ce programme une note d'appréciation de 16 sur 20.

Groupe 6

Ce groupe est le seul ayant inscrit dans une fonction la recherche du voltage de démarrage et d'arrêt du moteur. Deux structures de répétition successives servent efficacement à la détermination du démarrage et l'arrêt du moteur par pas de voltage unitaire. Chacune de ces structures de répétition incluent une structure de décision

servant à mémoriser l'information désirée. Les instructions de la fonction décrivent aisément la stratégie de recherche des étudiants pour trouver manuellement ces valeurs. Du programme on lit : tant que le moteur n'est pas démarré, augmenter le voltage ; noter le voltage de démarrage ; tant que le moteur n'est pas arrêté, diminuer le voltage ; noter le voltage d'arrêt.

Le programme compte 53 lignes de code et un niveau d'imbrication de 2. Les deux structures de répétition FOR sont bien construites et sollicitent des expressions booléennes composées. L'utilisation d'une variable booléenne nommée, DepartTrouver, rend l'expression booléenne de la structure FOR extrêmement significative : (... PAS DepartTrouver). Cependant, le deuxième niveau d'imbrication pourrait être enlevé en ajoutant sa propre expression booléenne à l'expression booléenne de la structure de répétition. Cette manipulation simplifierait le code et diminuerait la valeur LOC d'environ 12.

Nous attribuons à ce programme une note d'appréciation de 18 sur 20. Les données de l'analyse des programmes du deuxième exercice sont résumées au tableau 5.2.

Tableau 5.2 Programme cherchant le voltage de démarrage et d'arrêt d'un moteur

Groupe	LOC	NISC	Structure de décision	Structure de répétition	Expression booléenne	Remarque	Note/20
1	0	0	0	0	0	NIL	0
2	0	0	0	0	0	NIL	0
3	34	1	0	2 do-while	Composée		18
4	19	1	0	1 do-while	Composée	Traite uniquement le démarrage	10
5	39	2	3 if	3 for	Simple	Sortie du for avec break Instruction i=i ;	16
6	53	2	2 if	2 for	Simple Composée	Les deux structures if peuvent être enlevées	18

Les diagrammes de flux de contrôle présentés à la figure 5.2 sont plus simples que ceux du premier exercice. Les diagrammes utilisent presque exclusivement les structures de répétition qui se distinguent en nombre et en emplacement de l'expression booléenne. Les groupes 5 et 6 ajoutent à leur programme des structures de décision IF ce qui augmentent le niveau d'imbrication. Rappelons que cet ajout est redondant et hautement déplorable dans le cas du groupe 5.

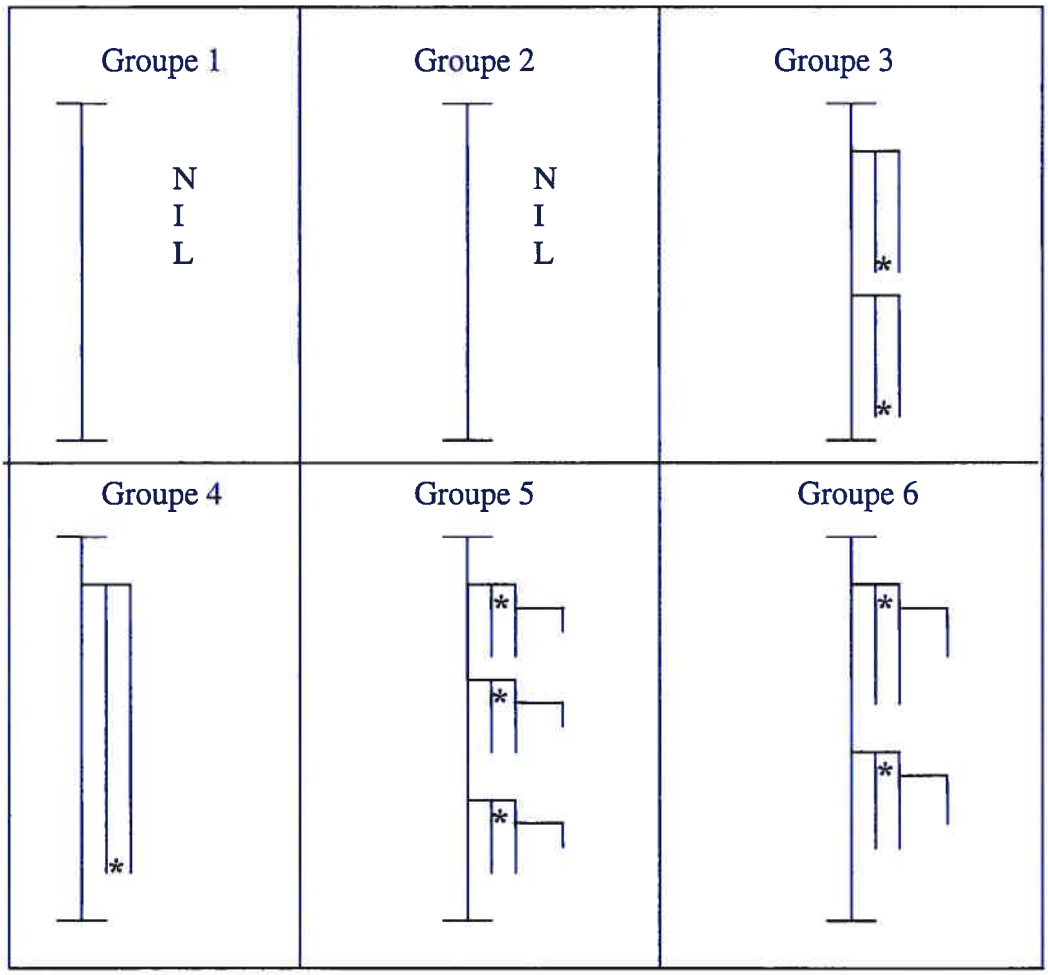


Figure 5.2 Diagramme de flux de contrôle des programmes

Il nous apparaît évident que la manipulation initiale du montage a influencé la stratégie de la majorité des solutions proposées. L'analyse des programmes réalisés par les différents groupes nous révèle une transposition quasi explicite de ces manipulations menant à une solution.

Les groupes 1 et 2 ont tenté de transposer directement leur méthode de recherche de solution de l'ordre d'allumage des luminodiodes. Dans les deux cas nous avons une structure de décision imbriquée qui teste l'état de chacune des quatre luminodiodes. Le programme du groupe 2 semble plus révélateur puisque l'absence d'une structure de répétition ne permet de vérifier qu'un seul scénario possible. Sans aucun doute celui exploré. Le programme du groupe 4 suggère aussi une transposition des manipulations même si le niveau d'abstraction est un peu plus élevé. Finalement, les groupes 5 et 6 utilisent une structure abstraite qui démontre une meilleure connaissance de la programmation permettant un certain éloignement des opérations de manipulation du système.

La transposition des manipulations de l'exercice concernant le démarrage et de l'arrêt d'un moteur apparaît encore plus significatif. En effet, tous les groupes qui ont complété cet exercice ont implanté explicitement les opérations effectuées lors de la manipulation du système. Dans un premier temps, une structure de répétition pour la recherche du seuil de départ suivie dans un deuxième temps d'un structure de répétition pour obtenir le seuil d'arrêt du moteur. La nécessité d'essayer plusieurs valeurs, de répéter la même opération, a sûrement facilité l'identification de ces structures.

5.2 Évaluation de l'atelier

Après la réalisation de l'atelier, nous avons soumis un questionnaire (annexe F) aux étudiants afin de connaître leur appréciation. Cette section présente les résultats de ce questionnaire.

Selon l'échelle de valeurs : fortement en désaccord, en désaccord, d'accord et fortement d'accord ; nous avons demandé aux étudiants de juger les affirmations suivantes :

- Les consignes sont claires et précises.
- Les objectifs sont bien identifiés.
- La période de temps disponible est suffisante pour la réalisation du travail.
- J'ai trouvé l'atelier très fascinant et intéressant.

Les résultats sont présentés à l'aide du graphique de la figure 5.3. Outre l'affirmation sur les consignes, les étudiants sont majoritairement d'accord avec la teneur des affirmations.15 étudiants indiquent leur fascination et leur intérêt pour l'exercice, 12 étudiants précisent que les objectifs sont bien identifiés, de même 12 étudiants prétendent que le temps disponible est suffisant pour la réalisation du travail. Or, 4 équipes sur 6 ont complété entièrement ou partiellement les deux exercices. Le caractère facultatif de l'atelier et sa nouveauté ont probablement contribué au biais de cette dernière affirmation.

Les étudiants sont partagés sur la clarté et la précision des consignes. En effet, 8 étudiants sur 15 soit la moitié, déplorent la clarté et la précision des consignes. Ce résultat ne nous surprend guère puisque nous avons construit notre activité selon une approche par découverte. Habitués à un énoncé de problème sans ambiguïté, l'étudiant est quelque peu désorienté lorsqu'il doit découvrir par lui-même les précisions du travail à réaliser. D'une certaine manière, ce résultat montre le niveau d'adaptation ou la réticence au changement des étudiants.

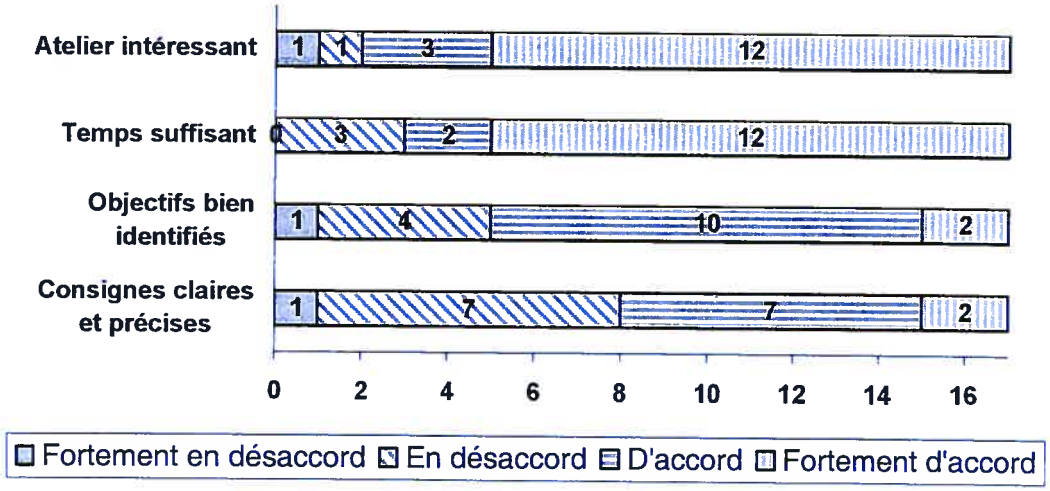


Figure 5.3 Appréciation de l'atelier

Nous avons ensuite tenté de vérifier les contributions « sociales » dans l'apprentissage réalisé lors des ateliers. Les contributions possibles proviennent soit du professeur, de

collègue(s) ou de l'apprenant. Toujours selon la même échelle de valeurs nous avons demandé aux étudiants de juger les affirmations suivantes :

- La principale contribution de mon apprentissage provient de mon professeur
- La principale contribution de mon apprentissage provient d'un ou d'une collègues
- La principale contribution de mon apprentissage provient de moi-même

En plus de leurs propres efforts, les étudiants affirment que leur apprentissage bénéficie également de l'aide du professeur et de collègues. Ces résultats sont présentés dans le graphique de la figure 5.4.

Nous désirons apporter une précision sur l'apport du professeur. Ayant préparé et œuvré lors de l'atelier, la contribution du professeur s'est limitée presque exclusivement à la présentation et l'installation du matériel physique et du logiciel. Le professeur n'est intervenu d'aucune façon dans l'élaboration des programmes. Les seules questions qui ont obtenu certaines réponses concernaient l'environnement de programmation.

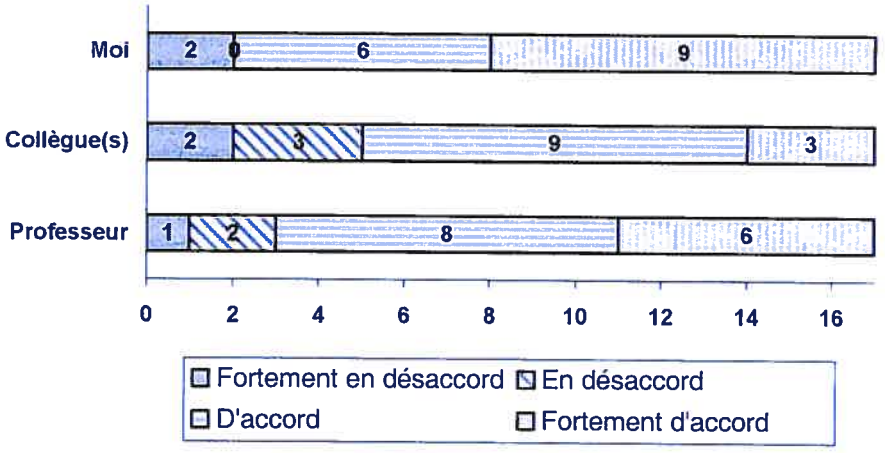


Figure 5.4 Contribution dans l'apprentissage

Quel apprentissage l'étudiant fera-t-il pendant cet atelier? Pour répondre à cette question nous avons demandé aux étudiants de préciser les notions explorées ou abordées lors de la réalisation des exercices. Cette question ouverte n'offre aucun choix à l'usager afin

d'éviter les biais de réponses suggérées. Les affirmations obtenues sont énumérées au tableau 5.3. Le chiffre entre parenthèses précise le nombre d'occurrences de la notion. Nous avons classé les réponses selon trois aspects: notion propre au langage de programmation, notion liée au matériel et notion relative à toute autre considération.

Tableau 5.3 Cet atelier m'a permis d'explorer les notions suivantes

Langage de programmation	Matériel	Autre
▪ Structure de répétition (4) ▪ Structure de décision (3) ▪ Fonction (3)	▪ Port, communication ordinateur-microcontrôleur (5) ▪ Imperfection de l'ordinateur ▪ Utilisation de matériel (2) ▪ Programmation de commandes à du matériel externe à l'ordinateur (2)	▪ Recherche de solutions réelles ▪ Ne pas perdre patience ▪ Utilisation d'un fichier dll

Les réponses obtenues soulignent l'intérêt des étudiants pour le montage proposé. Les mentions inscrites dans la colonne **Matériel** sont révélatrices. En effet, la possibilité de pouvoir interagir sur un montage électronique par l'entremise de l'exécution de leur propre programme a sans contredit piqué leur curiosité et par le fait même augmenter leur motivation. Ils sont également conscients des structures de programmation nécessaires pour réaliser le programme. Le premier exercice impose l'utilisation de structures de décision tandis que le deuxième exercice requiert l'utilisation de structures de répétition et chaque exercice fait nécessairement appel à un ensemble de fonctions préalablement fournies. Ces notions se retrouvent intégralement dans la colonne **Langage de programmation**. Nous attendions un nombre beaucoup plus élevé d'occurrences de ces notions vu l'évidence de leur utilisation dans chaque programme. À preuve, tous les programmes du premier exercice utilisent la structure de décision et tous les programmes du deuxième exercice utilisent la structure de répétition. À notre avis, soit qu'ils n'ont pas jugé pertinent de mentionner ces structures étant donné leur pertinence, soit qu'ils les utilisent sans pouvoir les nommer. Dans la dernière colonne, **Autre**, une mention souligne la recherche de solutions réelles, une autre invite à ne pas perdre patience et la dernière précise l'utilisation d'un fichier dll. Comme le fichier dll

est un sujet non abordé dans le cours, il n'est pas étonnant qu'il soit cité. Nous avons utilisé ce fichier pour dissimuler notre programmation aux étudiants qui auraient pu s'inspirer ou copier certaines portions de ces instructions. La recherche de solutions réelles est un élément intéressant puisqu'il traduit précisément l'idée de l'utilisation d'un montage réel. Nous désirons que l'étudiant raisonne à partir du réel pour obtenir sa solution abstraite. Finalement, ne pas perdre patience, suggère une réflexion de groupe qui invite à l'entraide ou à la collaboration.

En dernier lieu, nous avons encouragé les étudiants à commenter librement les ateliers. Ces commentaires sont regroupés au tableau 5.4. La majorité des commentaires, sauf deux, sont élogieux et soulignent le caractère intéressant de l'atelier. Les deux premiers commentaires présentés dans le tableau se distinguent des autres. Le premier commentaire souligne le recours à des notions inconnues par l'utilisation du port série et l'utilisation de nouvelles fonctions. Nous savons que certaines personnes doivent étudier en profondeur toutes les composantes d'une notion pour s'assurer d'un bon apprentissage. Cependant, comme la programmation du port série aurait exigé la présentation d'un volume important de théories qui sont au programme d'un cours plus avancé, nous avons jugé plus pertinent de fournir des fonctions simples réalisant la communication. Nous avons simplifié au maximum les interactions avec le port en offrant aux étudiants des fonctions qui le rendent transparent telles `AllumerLed()`, `EteindreLed()`, `AvancerMoteur()`, etc. De plus, sachant qu'une des facettes de la problématique de l'apprentissage de la programmation concerne la surcharge de l'acquisition de connaissances du domaine d'application, il nous apparaissait essentiel d'éviter cette difficulté à l'étudiant. Nous avons observé que l'exercice proposé possédait cette qualité puisqu'il n'a pas exigé à l'étudiant une compréhension particulière du système électronique. L'étudiant s'attaquant dès le départ au développement de la solution informatique. Le deuxième commentaire souhaite une feuille de directives. Nous avons jugé qu'il serait intéressant de permettre à l'étudiant de découvrir le travail à réaliser par la manipulation du montage. Cette nouvelle façon de procéder s'inscrit dans la philosophie d'un apprentissage par projet où chaque groupe d'étudiants peut réaliser un travail différent.

Tableau 5.4 Commentaires en vrac

▪ L'utilisation de ports requiert des notions que nous ne maîtrisons point. Je préfère des travaux où toutes les fonctions que nous utilisons peuvent être comprises. ▪ Y aurait pas pu avoir une feuille de directives ?
▪ Très intéressant. Cette approche pourrait certainement intéresser des étudiants qui ne s'intéresseraient pas normalement à la programmation. ▪ Atelier intéressant, permet de toucher un peu aux choses. ▪ Atelier très intéressant. Bravo pour l'initiative. ▪ C'est franchement une bonne idée. ▪ Cool ! ▪ Intéressant ▪ Intéressant. C'est une idée intéressante d'intégrer des notions plus « low-level » dans le cours. ▪ Très amusant. Mets en pratique plusieurs techniques en un projet. Très instructif et je recommencerais volontiers !

5.3 Évaluation du tutoriel

Puisque à notre avis, l'étudiant n'est pas le meilleur juge pour évaluer notre tutoriel, nous avons plutôt proposé cette tâche à un groupe d'experts. Nous avons alors sollicité vingt enseignants qui ont enseigné ou enseignent un cours touchant l'apprentissage de la programmation. De ce groupe, quinze enseignants n'ont pas pu résister au plaisir d'évaluer notre tutoriel en répondant à un questionnaire d'évaluation inspirée d'un document intitulé, « Grille d'évaluation d'un logiciel » de madame Anne Gras de l'Institut Universitaire de Formation des Maîtres (IUFM) de La Réunion. Le questionnaire explore de fond en comble le tutoriel par le biais de 71 affirmations (annexe E) réparties entre les huit facettes suivantes:

- l'interface;
- l'interactivité;
- la convivialité;
- le contenu;
- la rubrique Définition;
- la rubrique Analogies;
- la rubrique Exercices;
- la rubrique Simulations.

L'évaluateur devait juger ces affirmations selon l'échelle de valeurs: fortement en désaccord, en désaccord, d'accord et fortement d'accord (avec des cotes correspondantes s'étalant de 1 à 4). Ainsi, la moyenne de chacune des affirmations peut être calculée.

Comme chacune des facettes regroupe plusieurs questions, la moyenne générale de toutes les questions peut-être utilisée si les profils de réponse de chacune des questions ne sont pas significativement différents (Sokal et al., 1995). Cette évaluation de l'identité des profils de réponse peut se faire à l'aide d'un simple test Khi^2 portant sur la distribution de fréquences des questions de chaque facette. Si le test n'est pas significatif aux seuils alpha habituels de 0,05 et 0,01, la moyenne générale peut-être utilisée. Dès lors les résultats obtenus à une facette s'interprètent en comparant la moyenne générale à un désaccord total impliquant une moyenne de 1, une évaluation neutre avec une moyenne de 2,5 ou un accord complet générant une moyenne de 4.

Nous présentons dans les neuf tableaux suivants les distributions de fréquences aux questions associées aux huit facettes étudiées. Chaque tableau comprend pour chacune des questions: le nombre de répondants (N), la fréquence obtenue pour chaque valeur de l'échelle d'évaluation (1=fortement en désaccord, 2=en désaccord, 3=en accord, 4=fortement d'accord) la moyenne de la question considérée ainsi que la moyenne générale de la facette.

La première facette, nommée interface, englobe les attributs de l'interface concernant son esthétique, la justesse de la disposition de l'information et la conformité à l'usage.

Le test Chi^2 initial portant sur les 11 questions de cette facette est significatif (54,91, dl=30, p=0,0036) mais l'examen des profils révèlent que la question 16 portant sur la surcharge de l'écran est à la source de cette différence de profils probablement parce que relativement aux autres questions elle est trop générale et inversée dans son sens général; cet examen détaillé est nécessaire car le libellé des questions affecte le jugement des évaluateurs (Guilford, 1954). Les termes **souvent** et **surchargé** dans l'affirmation de la question 16, « L'écran est souvent surchargé », peuvent être interprétés de façons différentes d'une personne à l'autre. Lorsque l'on retire cette question, le test Chi^2 sur les dix questions résiduelles devient non-significatif (28,46, dl=27, p=0,3875) et la moyenne générale de la facette peut être interprétée.

Le tableau 5.5 présente donc les résultats aux questions retenues pour l'évaluation. La moyenne générale de 3,47 se rapproche d'un accord total du point de vue de la qualité de l'interface. Ce résultat est encourageant puisqu'il concerne autant la rapidité d'exécution des commandes que la présence ou la présentation de certaines fonctions ainsi que l'adéquation de la complexité du contenu avec le calibre des usagers.

Tableau 5.5 L'interface

Interface	N	Fréquence				Note
		1	2	3	4	
Les fonctions sont intuitives	15	0	2	8	5	3.20
Facilité à quitter	15	0	0	2	13	3.87
Peu quitter en tout temps	15	0	0	2	13	3.87
Complexité appropriée aux usagers	15	0	1	10	4	3.20
Utilisation d'illustrations, de son, etc.	15	0	0	9	6	3.40
Repérage et utilisation de l'information	15	0	0	8	7	3.47
Espace-écran balancé	15	0	0	9	6	3.40
Distinction entre l'espace information et l'espace fonctionnement logiciel	15	0	1	8	6	3.33
Déplacement dans le tutoriel	15	0	1	8	6	3.33
Rapidité d'exécution des commandes	15	0	1	4	10	3.60
					Moyenne générale	3.47

Les deux prochaines facettes s'intéressent aux outils ou procédés inscrits dans le tutoriel qui soutiendront l'étudiant dans son apprentissage et qui l'inciteront à l'utiliser. Ces deux facettes sont l'interactivité et la convivialité du tutoriel. L'évaluateur doit ici considérer l'étudiant en pleine action de manière à juger convenablement les différents attributs. Le test Chi^2 portant sur les 6 questions de la facette Interactivité est non-significatif (8,45, dl=15, p=0,9045) permettant l'interprétation de la moyenne générale de la facette. Le tableau 5.6 présente les résultats aux questions retenues pour l'évaluation. La moyenne générale de 3,33 se rapproche d'un accord total du point de vue de la qualité de l'interactivité. Les attributs liés à ce résultat concernent les capacités à soutenir l'attention, à nourrir l'intérêt et à favoriser les apprentissages. Pour nous, ces attributs sont indispensables voire fondamentaux à toute bonne application informatique. Le test Chi^2 portant sur les 8 questions de la facette Convivialité est non-significatif (27,37, dl=24, p=0,2874) permettant l'interprétation de la moyenne générale de la facette. Le tableau 5.7 présente donc les résultats aux questions retenues pour l'évaluation. La moyenne générale de 3,40 est près d'un accord total du point de vue de la qualité de la convivialité. Pour la convivialité, nous avons tenté d'uniformiser les séquences tout en permettant à l'étudiant de travailler à son rythme.

Tableau 5.6 L'interactivité

Interactivité	**N**	**Fréquence**				**Note**
		1	**2**	**3**	**4**	
Types d'interactions possibles	15	0	0	10	5	3.33
Utilisation appropriée des interactions	14	0	0	10	4	3.29
L'intensité des interactions est bien dosée	14	0	1	11	2	3.07
Capacité à soutenir l'attention	15	0	1	6	8	3.47
Capacité à nourrir l'intérêt	15	0	1	8	6	3.33
Capacité à favoriser les apprentissages	15	0	1	6	8	3.47
					Moyenne générale	3.33

Tableau 5.7 La convivialité

Convivialité	N	Fréquence				Note
		1	2	3	4	
Rétention à long terme des commandes	14	0	0	9	5	3.36
Sentiment d'une réalisation	14	0	1	8	5	3.29
Uniformité des séquences d'action	15	0	0	8	7	3.47
Présence de raccourcis	15	2	2	4	7	3.07
Rétroaction appropriée à l'action	15	0	2	7	6	3.27
Commande de renversement « undo »	15	1	1	4	9	3.40
Travail au rythme de l'étudiant	15	0	1	1	13	3.80
Convivial	15	0	0	7	8	3.53
Temps nécessaire à une tâche est adéquat	15	0	1	6	8	3.47
					Moyenne générale	3.40

Pour avoir une idée plus précise de l'appréciation de l'interactivité présente dans le tutoriel, nous avons demandé à nos évaluateurs de répondre à la question: « En quoi ces interactions sont-elles spéciales, remarquables ou déficientes? ». Les réponses obtenues sont regroupées au tableau 5.8. Elles nous informent des forces et des faiblesses de l'interactivité ainsi que des préoccupations des évaluateurs. Les commentaires mentionnent les interactions principalement dans trois rubriques: analogies, exercices et simulations. En majorité, ils soulignent la capacité de l'interaction à interpeller l'étudiant de manière à maintenir son intérêt ou le guider vers la solution d'un exercice ou l'inciter à un raisonnement analogique.

Tableau 5.8 En quoi ces interactions sont-elles spéciales, remarquables ou déficientes?

No.	Réponse
1	Il faut s'assurer de garder l'étudiant intéressé à apprendre.
2	je les trouve bien comme ça...
3	Faciles à comprendre, intuitives
5	xxx
6	Le fait d'avoir des interactions pour chacun des exercices permet à l'étudiant de vraiment s'interroger sur les réponses, et donc de pratiquer, contrairement à un solutionnaire seul.
7	Les interventions de l'étudiant sont minimales car choix de menus
9	Elles sont intuitives, permettent de garder l'étudiant éveillé et donnent une rétroaction immédiate, ce qui favorise l'apprentissage. Le danger est que l'étudiant peut court-circuiter la réflexion/recherche de sa réponse et sauter directement à la

	solution. Ce problème est cependant presque impossible à éviter... Certains boutons ne semblent pas fonctionnels (par exemple le ? pour informations additionnelles) dans plusieurs écrans. Suggestion: utiliser un effet visuel pour signaler s'ils sont actifs ou non selon l'écran.
10	Spéciales, car elles permettent de voir l'effet directement à l'écran (affectation de variables par exemple).
11	Quelques interactions, notamment celles utilisées lors des analogies, bien qu'elles peuvent être jugées superflues peut-être au niveau de l'enseignement, sont très bien pensées et permettent à l'étudiant de sortir du cadre formel du cours pour s'amuser un peu.. Très pédagogique, libérateur et humoristique. Les possibilités de retour en arrière (analogie, définition et simulation) sont aussi très bien pensées. L'accès omniprésent au code est aussi très bien fait, que ce soit dans les exercices et les simulations. Finalement, l'exécution pas à pas est remarquable. Il faudrait seulement présenter les outils de debug une fois ce concept introduit. Je pense que cela aiderait les étudiants dans leur lab et le saut entre un outil de debug et l'exécution pas à pas simulée ne leur demandera pas grand effort.
12	Les interactions sont très riches et aideront beaucoup les étudiants. Il faudrait peut-être revoir certains concepts comme la vitesse de déroulement des simulations (100 ms, c'est trop rapide). Le mode pas à pas est par contre excellent.
13	Remarquables: les interactions sont partout, et il y en a toujours, ce qui augmente l'intérêt pour explorer le tutoriel. Déficientes: des fois le background couvre le texte lorsqu'on clique sur un bouton ou rubrique, ça dérange beaucoup.
15	Permet d'apprendre par essais et erreurs sans craindre de faire des erreurs.

La prochaine facette se préoccupe du contenu de l'information inséré dans le tutoriel ainsi que son traitement. Les résultats sont présentés aux tableaux 5.9 et 5.10. Le tableau 5.9 regroupe des affirmations où l'évaluateur devait préciser un niveau d'accord tandis que le tableau 5.10 regroupe des affirmations où l'évaluateur devait préciser un niveau d'importance. Le test Chi^2 portant sur les 5 questions de la facette Contenu – jugements d'accord est non-significatif (13,52, dl=12, p=0,3325) permettant l'interprétation de la moyenne générale de la facette. La moyenne générale de 3,49 se rapproche d'un accord total du point de vue de la qualité du contenu. Le test Chi^2 portant sur les 8 questions de la facette Contenu – jugements d'importance est non-significatif (23,40, dl=21, p=0,3229) et la moyenne générale de la facette peut être interprétée. La moyenne générale de 3,32 est près d'un accord total du point de vue de la qualité du contenu.

L'adéquation du contenu du tutoriel avec le contenu du cours, la présentation du contenu sous plusieurs formes à l'aide de quatre rubriques, la division du contenu et son accessibilité sont des caractéristiques sur lesquelles nous avons porté une attention particulière tout au long du développement. De plus, nous avons tenté d'obtenir une information supplémentaire sur la pertinence des rubriques à l'aide des jugements d'importance.

Tableau 5.9 Le contenu – jugements d'accord

Contenu – jugements d'accord	N	Fréquence				Note
		1	2	3	4	
Contenu conforme au cours	14	0	1	6	7	3.43
Contenu présenté sous plusieurs formes	15	0	5	5	5	3.00
Division du contenu	15	0	1	2	12	3.73
Contenu accessible aux étudiants	15	0	1	5	9	3.53
Traitement à l'aide des quatre rubriques	15	0	0	4	11	3.73
					Moyenne générale	3.49

Tableau 5.10 Le contenu – jugements d'importance

Contenu – jugements d'importance	N	Fréquence				Note
		1	2	3	4	
Importance accordée à Définition	15	0	2	10	3	3.07
Importance que j'accorderais à Définition	15	0	2	8	5	3.20
Importance accordée à Analogie	15	0	1	11	3	3.13
Importance que j'accorderais à Analogie	15	0	3	9	3	3.00
Importance accordée à Exercice	15	0	0	6	9	3.60
Importance que j'accorderais à Exercice	15	0	0	3	12	3.80
Importance accordée à Simulation	15	0	3	5	7	3.27
Importance que j'accorderais à Simulation	15	0	0	8	7	3.47
					Moyenne générale	3.32

Les quatre prochains tableaux 5.11 à 5.14 présentent les attributs retenus pour évaluer plus précisément chacune des quatre rubriques. La rubrique Définition est conçue pour présenter un résumé de la matière du cours. Le tableau 5.11 présente les résultats aux questions retenues pour l'évaluation. Le test Chi^2 portant sur les 5 questions de la facette Définition est non-significatif (4,63, dl=12, p=0,9692) permettant l'interprétation de la

moyenne générale de la facette. La moyenne générale de 3,45 est tout près d'un accord total du point de vue de la qualité de la facette Définition.

Tableau 5.11 La rubrique Définition

Rubrique Définition	N	Fréquence				Note
		1	2	3	4	
Découpage des notions	13	0	0	8	5	3.38
Repérage	13	0	1	6	6	3.38
Utilisation appropriée des onglets	14	0	0	6	8	3.57
Choix des exemples	14	0	0	7	7	3.50
Outils facilitant l'apprentissage	14	0	0	8	6	3.43
					Moyenne générale	3.45

La rubrique Analogies constitue la principale originalité du tutoriel et représente un véritable moyen didactique mis à la disposition de l'étudiant. Nous sommes conscients qu'il y a place à plusieurs améliorations. Certaines analogies sont plutôt « faibles » et l'interaction n'est pas toujours évidente à intégrer. Voila des suggestions d'améliorations possibles. Le tableau 5.12 présente les résultats aux questions retenues pour l'évaluation. Le test Chi^2 portant sur les 6 questions de la facette Analogies est non-significatif (4,05, dl=15, p=0,9976) permettant l'interprétation de la moyenne générale de la facette. La moyenne générale de 3,28 se rapproche d'un accord total du point de vue de la qualité de la facette Analogies.

Tableau 5.12 La rubrique Analogies

Rubrique Analogies	N	Fréquence				Note
		1	2	3	4	
Directives et consignes claires	15	0	1	10	4	3.20
Interactions adéquates	15	0	2	6	7	3.33
Similitude avec la notion enseignée	15	0	1	8	6	3.33
Évidence de la relation à transférer	15	0	2	9	4	3.13
Les explications sont claires	15	0	1	7	7	3.40
Outils facilitant l'apprentissage	15	0	1	9	5	3.27
					Moyenne générale	3.28

L'apprentissage par problèmes est fortement usité en ingénierie. Les résultats obtenus par la rubrique Exercices vont dans ce sens. Le test Chi^2 initial appliqué sur les 8

questions de cette facette est significatif (35,21, dl=21, p=0,0268) mais l'examen des profils révèlent que la question 60 portant sur les outils facilitant l'apprentissage est à la source de cette différence de profils probablement parce que relativement aux autres questions elle est trop générale. L'affirmation de la question 60, « La rubrique EXERCICES offre des outils qui seront utiles à l'apprentissage de l'étudiant », peut être interprétée différemment selon la perception de l'utilité ou l'apport des exercices à l'apprentissage de l'étudiant. Lorsque l'on retire cette question, le test Chi^2 sur les dix questions résiduelles devient non-significatif (21,08, dl=18, p=0,2755) et la moyenne générale de la facette peut être interprétée. Le tableau 5.13 présente les résultats aux questions retenues pour l'évaluation. La moyenne générale de 3,28 se rapproche d'un accord total du point de vue de la qualité de la facette Exercices. Par contre, nous croyons qu'il y a sûrement place à de l'amélioration du côté de la correction.

Tableau 5.13 La rubrique Exercices

Rubrique Exercices	N	Fréquence				Note
		1	2	3	4	
Directives et consignes claires	15	0	0	11	4	3.27
Directives et consignes au moment opportun	15	0	1	8	6	3.33
Possibilité de commettre plusieurs erreurs	15	0	0	8	7	3.47
Considère plusieurs types d'erreurs	15	0	0	8	7	3.47
La correction	14	0	2	6	6	3.29
Résultat de la correction	15	1	1	6	7	3.27
La correction oriente l'étudiant	15	0	5	7	3	2.87
					Moyenne générale	3.28

Le tableau 5.14 présente les résultats aux questions retenues pour l'évaluation de la facette Simulations. Le test Chi^2 portant sur les 7 questions de la facette Simulations est non-significatif (12,54, dl=18, p=0,8180) et la moyenne générale de la facette peut être interprétée. La moyenne générale de 3,56 se rapproche d'un accord total du point de vue de la qualité de la facette Simulations. Il est nécessaire de préciser que l'interface présentée est très proche de celle offerte par les débogueurs compris dans les environnements intégrés de développement informatique. Les développeurs se

retrouvent ainsi à utiliser une interface familière et peuvent constater aisément les avantages comme les inconvénients de l'interface.

Tableau 5.14 La rubrique Simulations

Rubrique Simulations	N	Fréquence				Note
		1	2	3	4	
Utilisation des différentes vues	15	0	1	6	8	3.47
Longueur des programmes	14	0	0	4	10	3.71
Difficulté des programmes	15	0	1	7	7	3.40
Enchaînement des actions	15	0	1	2	12	3.73
Utilité de l'enchaînement des actions	15	0	1	7	7	3.40
L'interaction assure l'intérêt de l'étudiant	15	0	1	8	6	3.33
Outils facilitant l'apprentissage	15	0	0	2	13	3.87
	Moyenne générale					3.56

Sachant que le nombre de répondants ayant affirmé pouvoir utiliser un certain pourcentage du tutoriel est respectivement : 2 (1 à 25%), 4 (26 à 50%), 6 (51 à 75%) et 3 (76 et plus), nous leur avons demandé de répondre aux deux questions suivantes : « Si vous pensez pouvoir utiliser certains éléments du tutoriel dans votre enseignement en classe, quels sont-ils et à quelle fin les utiliserez-vous? » et en complément «Pour quelles raisons n'envisagez vous pas d'utiliser le tutoriel ou certains éléments du tutoriel dans votre enseignement en classe? ». Les réponses à ces questions sont présentées aux tableaux 5.15 et 5.16. Il est important de préciser ici que le tutoriel n'est pas dédié à l'enseignant comme il n'a pas été conçu en tant qu'outil d'aide à son enseignement. Par contre, ces questions peuvent nous informer grandement sur les contributions possibles du tutoriel du point de vue d'enseignement. Dans l'ensemble, les différentes rubriques sont énumérées par les évaluateurs dans certains cas comme élément qu'ils intégreraient à leur enseignement et en d'autres cas comme élément qu'ils conseilleraient à leurs étudiants. Un certain consensus ressort de ces commentaires : le tutoriel serait un complément utile à l'enseignement. La seconde question a permis à nos évaluateurs de souligner la vocation plutôt autodidacte du tutoriel. En effet, on peut déceler plusieurs mentions précisant ou conseillant une utilisation personnelle du tutoriel par l'étudiant.

Tableau 5.15 Si vous pensez pouvoir utiliser certains éléments du tutoriel dans votre enseignement en classe, quels sont-ils et à quelle fin les utiliserez-vous?

No.	Réponse
1	Je crois que ceci est un bon pas. Je suis persuadé avec quelques ajustements, l'étudiant pourrait suivre ce cours de façon interactive avec l'aide d'un tuteur virtuel.
2	les simulations, surtout... le reste, je les encouragerais très fortement a y aller par eux mêmes... je leur montrerai juste un peu au début, pour les convaincre que c'est simple et utile...
3	Analogies pour s'en inspirer Simulation pour remplacer les exécutions pas à pas de programmes que je montrais avant
4	Je pense utiliser la partie définitions et exercices. Je crois que la partie simulation est plus pour l'étudiant qui peut donc assimiler calmement certains concepts et faire la part des choses.
6	Aucun en particulier, mais le tutoriel pourrait surtout servir aux élèves qui désirent apprendre la matière à leur rythme mais qui ont de la difficulté à lire un manuel, ou encore le tutoriel pourrait me servir dans le cours pour démontrer une notion mal comprise par les étudiants, ou encore pour revenir sur des notions difficiles avec un ou des élèves en difficulté.
7	Surtout la partie Analogie. Faire les exercices en classe
8	Les définitions présentées dans le tutoriel sont bien choisies et pourraient être utilisées en classe. Un nombre d'exercices aussi (et laisser les autres à faire par les étudiants, de plus s'ils n'ont pas suivi dans le cours il peuvent les refaire) Je donnerais les analogies comme lecture personnelle à faire à l'extérieur du cours. Même chose pour les simulations.
9	J'utiliserais certainement les définitions, qui remplaceront et/ou complèteront les transparents actuels. Il serait peut-être bon, dans un tel cas, qu'une version "printer-friendly" des écrans du tutoriel puisse être accessible et imprimable par les étudiants afin de suivre en classe et de prendre des notes manuscrites. J'emploierais certainement aussi les analogies, qui sont appropriées et extrêmement utiles pour faire comprendre des points plus subtils.
10	Interaction avec les étudiants qui désirent améliorer leurs connaissances principalement. Aussi, un peu de démonstrations en classe, avec surtout les simulations, ce qui présente un outil d'enseignement additionnel très, très, très intéressant.
11	Certains exemples et analogies sont bien pensés et gagneraient à être utilisés en classe. Je recommanderais aussi aux étudiants les exercices du tutoriel en fonction de la matière vue en cours et les inciteraient à les compléter. Finalement, le tutoriel est un bon outil de complément post ou pré-cours.

12	Les définitions des sections se rapprochent beaucoup de ce que j'enseigne. Comme j'utilise peu les acétates car cela ennui les étudiants, le tutoriel, qui est beaucoup plus interactif que l'acétate va me donner un médium qui convient beaucoup mieux au type d'enseignement que je pratique. Pour les autres sections, je pourrais utiliser certains exercices et laisser aux étudiants le loisir de consulter les analogies et les simulations en les encourageant à faire les exercices.
13	Exercices et Simulations
14	Je crois que tout le contenu du tutoriel peut être utile aux étudiants lors de la préparation d'un cours ou encore lors de la révision des notions vues en classe. Il n'est même pas exclu que le tutoriel devienne un support essentiel pour l'apprentissage en classe ou encore l'apprentissage autonome des étudiants.
15	Simulation pour des démonstrations et Définitions en remplacement des acétates.

Tableau 5.16 Pour quelles raisons n'envisagez vous pas d'utiliser le tutoriel ou certains éléments du tutoriel dans votre enseignement en classe?

No.	Réponse
1	Je n'ai pas trouvé les sections très claires (où sont-elles ?) : simulation, exercice.
2	je ne leur montrerai pas trop les exercices, par exemple, car ce n est pas a moi de leur faire... mais je leur dirai que c'est très important d'aller les faire...
3	Le logiciel me semble plus destiné à un apprentissage individuel (i.e chez soi) qu'à un apprentissage en groupe (i.e. en classe)
4	Je vais utiliser le tutoriel
6	Aucune raison.
7	Aucune
8	La créativité de l'enseignant est un atout important et stimulant pour l'enseignant et pour les étudiants. Il est donc bien de conserver un certain pourcentage de temps pour des exemples personnels et des méthodes d'explication différentes. Si les étudiants voient quelque chose en classe expliquée d'une façon et ensuite dans le tutoriel expliqué d'une autre façon, c'est bien, car ce ne sont pas tous les étudiants qui apprennent de la même façon.
9	Je ne présenterais probablement pas les exemples, puisque je considère que l'étudiant peut tirer profit de les faire seul et non en classe. C'est un outil plus personnel et je présenterais d'autres exemples en classe afin d'exposer l'étudiant au plus grand nombre possible d'exemples. Je laisserais aussi les simulations pour le travail personnel des étudiants pour les mêmes raisons. Il faudrait vérifier que les écrans sont "présentables" en classe, i.e. qu'ils ne sont pas surchargés ou que les caractères ne sont pas trop petits. Les écrans sont parfaits pour une lecture seul devant son écran, mais ce ne sera peut-être pas le cas pour une présentation en classe. Cela est à vérifier... Si c'est le cas, il faudrait peut-être une version "classroom" du tutoriel...
10	Non, définitivement, je l'utiliserais (ne serait-ce que pour les simulations).

11	Les exercices et l'exécution pas à pas constituent un travail personnel à mon avis qui est bien plus formateur s'il est fait de façon volontaire et individuelle par l'étudiant.
12	La dernière section englobe pointeurs, allocation dynamique et listes chaînées. C'est énorme pour une section. La matière sur les pointeurs et les listes chaînées n'est pas assez élaborée pour le calibre du cours. Il n'y a pas de section sur l'OO.
13	J'ai ma propre méthode, et je préfère utiliser le tutoriel pour certains sujets seulement.
14	Le risque que je vois c'est que certains étudiants qui auraient déjà fait un tour bref sur le tutoriel (ou qui planifient en faire un éventuellement la veille de l'examen), auront l'impression de déjà tout savoir (ou savoir ou regarder quant la question se pose) et se désintéressent du cours en classe (risque d'absentéisme important). C'est là ou l'enseignement en classe pourrait être négativement affecté par un usage abusif du tutoriel.
15	Les analogies reprennent la même notion plusieurs fois. C'est nécessaire si quelqu'un ne comprend pas, mais de manière personnelle, pas pour un groupe. Les exercices devraient servir aux étudiants à s'auto évaluer.

Nous leur avons demandé également quel pourcentage des éléments du tutoriel ils utiliseraient en classe. Deux répondants ont précisé entre 26 et 50 %, quatre entre 51 et 75 %, deux entre 76 et 100 %. Finalement, tous les répondants affirment apprécier l'utilité du turoriel ainsi que sa disponibilité sur l'Internet.

Chapitre 6

CONCLUSIONS ET PERSPECTIVES

Dans le cadre de cette recherche de développement nous avons conçu et réalisé un environnement informatique destiné à l'initiation de la programmation pour permettre à des étudiants d'acquérir une structure de pensée algorithmique. Cet environnement est composé d'un atelier de programmation destinée à contrôler un montage électronique robotisé et d'un tutoriel pour superviser et encadrer les activités d'apprentissage réalisées dans cet atelier ou à l'extérieur de celui-ci. Nous voyons cet environnement comme une boite à outils didactiques destinée aux enseignants et aux étudiants afin de palier aux difficultés inhérentes à l'enseignement et à l'apprentissage de la programmation (voir chapitre 1). Nous avons soumis un prototype de cet environnement à deux types d'évaluation : 1) une évaluation fonctionnelle avec des experts afin de l'améliorer sur le plan technologique (matériel et logiciel) et didactique; 2) une mise à l'essai empirique avec un groupe d'étudiants et de professeurs cibles afin d'identifier et d'évaluer principalement les bénéfices de cet environnement au plan didactique. Au plan théorique et pratique, cet environnement s'intègre dans le champ de recherche constitué par les environnements informatiques pour l'apprentissage humain (EIAH) (Balacheff, 1999).

Cet environnement a été conçu afin de faciliter l'acquisition de savoirs incluant des savoir-faire et savoir-être dans le but ultime de permettre leurs formulations, leurs appropriations et leurs utilisations par l'apprenant. Il se veut propice à l'action de l'apprenant en offrant plusieurs rapports aux savoirs et savoir-faire, en constituant un milieu propice à l'élaboration de situations didactiques au sens de Brousseau telles la situation de la formulation, la situation de l'action et la situation de la validation. Dans notre environnement, l'étudiant est engagé dans une activité de programmation à l'aide

du montage robotisé. Cet activité place l'étudiant en situation a-didactique, puisque le professeur joue ici la règle de dévolution, c'est-à-dire que l'étudiant doit exercer sa propre démarche de résolution de problème pour concevoir son programme.

Nous avons précisé que la conception des EIAH se divise en deux grandes catégories: les environnements construits selon un schéma de la communication pour lesquels la connaissance est le contenu intentionnel d'un échange de dialogues et les environnements construits selon un schéma de l'action pour lesquels la connaissance est construite au cours de la résolution d'un problème (Balacheff, 1999). Très clairement, notre environnement se situe dans les deux catégories. En effet, le tutoriel appartient à la première catégorie tandis que la composante atelier de programmation appartient à la deuxième catégorie. La complexité de la conception de notre EIAH est par le fait même accru par rapport à ceux n'appartenant qu'à une seule catégorie. Entre autres, nous avons du recourir à des experts de deux domaines différents lors de la mise à l'essai fonctionnelle.

L'étape de mise à l'essai fonctionnelle portant sur l'utilisation du matériel robotisé fut cruciale pour l'amélioration successive de ce prototype. À travers ces essais et les résultats obtenus avec les experts du laboratoire de robotique pédagogique (LRP) nous avons développé et expérimenté trois versions successives de ce prototype matériel. À chaque évolution du prototype, nous avons constaté une amélioration de performance qui convergeait progressivement vers nos intentions pédagogiques.

La mise à l'essai fonctionnelle du tutoriel nous a permis initialement de valider sa pertinence. Les experts pédagogues consultés ont bien accueilli le premier prototype du tutoriel et se sont empressés à nous proposer certaines améliorations. Une fois le deuxième prototype du tutoriel réalisé, nous étions prêt à passer à l'étape suivante, soit la mise à l'essai empirique de notre atelier de programmation avec le montage robotisé et le tutoriel.

La mise à l'essai empirique de l'atelier de programmation s'est effectuée avec des étudiants. Les résultats obtenus nous ont permis d'évaluer son utilité pédagogique et sa facilité d'appréhension. Le comportement du montage suggère à l'apprenant les opérations à inclure dans son programme et lui facilite la détection de ses erreurs. Par contre, dans ce type d'approche où les apprenants sont placés dans une situation de résolution de problème, nous avons observé la présence de plusieurs instructions inutiles ou redondantes dans les programmes réalisés par nos sujets. Ce comportement est typique dans tout apprentissage de la programmation où l'étudiant est centré, voire obnubilé par la syntaxe du langage de programmation. À partir des programmes rédigés, nous devons toutefois préciser, que l'observation du comportement du montage a permis à l'étudiant d'effectuer un choix judicieux des structures de contrôle ainsi que de réaliser correctement l'ordonnancement des opérations.

La mise à l'essai empirique du tutoriel s'est effectuée avec 15 professeurs d'informatique ayant enseigné au moins un cours d'introduction à la programmation informatique. Le questionnaire d'évaluation soumis à ces professeurs nous montre leurs préférences que nous avons ordonnées comme suit : la rubrique Simulation est la plus prisée suivie par les rubriques Exercice, Définition et Analogie. Globalement, l'interface du tutoriel a été bien appréciée. Certains commentaires précisent: « elles sont intuitives, permettent de garder l'étudiant éveillé et donnent une rétroaction immédiate, ce qui favorise l'apprentissage; les interactions sont très riches et aideront beaucoup les étudiants; remarquables, les interactions sont partout, et il y en a toujours, ce qui augmente l'intérêt pour explorer le tutoriel ». Par contre, au plan ergonomique les évaluateurs déplorent l'information trop abondante sur certains écrans. Pour ce qui est de l'interactivité offerte par le tutoriel, les commentaires montrent sa capacité à interpeller l'étudiant de manière à maintenir son intérêt et à le guider vers la solution d'un exercice ou à l'inciter à un raisonnement analogique. À leur avis, l'ajout de raccourcis serait grandement utile et apprécié par l'usager. Pour la rubrique définition, les évaluateurs ont bien apprécié son contenu. Un commentaire précise : « les définitions présentées dans le tutoriel sont bien choisies et pourraient être utilisées en classe; j'utiliserais certainement les définitions, qui remplaceront et/ou complèteront les transparents actuels ». Pour la

rubrique Analogie, les évaluateurs ont fait ressortir la difficulté d'identifier certaines relations analogiques. Pourtant, nous avions utilisé des analogues accessibles et surtout bien connus de tous. Voici d'autres commentaires concernant la rubrique Analogie : « les analogies reprennent la même notion plusieurs fois, c'est nécessaire si quelqu'un ne comprend pas, mais de manière personnelle, pas pour un groupe; certains exemples et analogies sont bien pensés et gagneraient à être utilisés en classe; j'emploierais certainement aussi les analogies, qui sont appropriées et extrêmement utiles pour faire comprendre des points plus subtils.» Pour la rubrique Exercice, les évaluateurs ont principalement mentionné sa pertinence mais ils sont partagés sur la capacité du résultat de la correction à orienter l'étudiant vers la solution. Voici quelques commentaires obtenus concernant la rubrique Exercice : « permet d'apprendre par essais et erreurs sans craindre de faire des erreurs; le fait d'avoir des interactions pour chacun des exercices permet à l'étudiant de vraiment s'interroger sur les réponses, et donc de pratiquer, contrairement à un solutionnaire seul; le danger de l'accès direct à la solution est que l'étudiant peut court-circuiter la réflexion/recherche de sa réponse et sauter directement à la solution - ce problème est cependant presque impossible à éviter... ». Pour la rubrique Simulation, la plus prisée, les commentaires dévoilent l'appréciation des évaluateurs : « je crois que la partie simulation est plus pour l'étudiant qui peut donc assimiler calmement certains concepts et faire la part des choses; présente un outil d'enseignement additionnel très, très, très intéressant; l'exécution pas à pas est remarquable ». Dans l'ensemble, le tutoriel a été perçu par les répondants comme un complément qui pourrait être utile à leur enseignement et soulignent son potentiel d'engager l'étudiant dans un apprentissage autodidacte. Voici quelques commentaires obtenus à cet effet : « le logiciel me semble plus destiné à un apprentissage individuel (i.e chez soi) qu'à un apprentissage en groupe (i.e. en classe); les exercices et l'exécution pas à pas constituent un travail personnel à mon avis qui est bien plus formateur s'il est fait de façon volontaire et individuelle par l'étudiant.; le tutoriel pourrait surtout servir aux élèves qui désirent apprendre la matière à leur rythme; je crois que tout le contenu du tutoriel peut être utile aux étudiants lors de la préparation d'un cours ou encore lors de la révision des notions vues en classe; il n'est même pas exclu que le tutoriel devienne un support essentiel pour l'apprentissage en classe ou encore l'apprentissage autonome des

étudiants; le tutoriel pourrait servir à l'enseignant dans le cours pour démontrer une notion mal comprise par les étudiants, ou encore pour revenir sur des notions difficiles avec un ou des élèves en difficulté. »

La spécificité de notre environnement d'apprentissage réside dans le fait que notre objet d'apprentissage est informatique et que l'outil d'apprentissage qui le supporte est aussi informatique. En effet, nous avons développé ici un modèle d'action qui intègre l'apprentissage des structures de base de l'informatique dans un processus pédagogique lié à l'apprentissage par compétence où l'action du sujet et la résolution de problème sont largement sollicitées. Ces deux domaines que nous avons intégrés dans un modèle d'action nous ont conduits à proposer un outil informatique d'apprentissage composé d'un tutoriel et d'un atelier de programmation informatique avec un montage robotisé.

Nous avons brièvement abordé le paradigme de programmation par objets (voir chapitre 3). Nous avions à ce moment défendu la pertinence de l'apprentissage du paradigme procédural comme préalable à l'apprentissage du paradigme objet. Nous avons montré par cette recherche, l'importance du montage robotisé associé au tutoriel pour permettre aux étudiants de s'approprier les concepts de base de la programmation procédurale. A notre avis, cette étape est préalable à la programmation objet. Par contre, il ne faut pas nier l'importance et la place grandissante du paradigme objet en programmation. Par exemple, dans l'activité de programmation, le montage robotisé se présente comme « un objet permettant de penser objet ». En effet, à la base, la conception par objet nécessite à la fois l'identification des propriétés de l'objet et ses comportements. Les méthodes de conception orientées objets nous enseignent de rechercher les propriétés de l'objet en examinant les différents états possibles de l'objet, tandis que les comportements de cet objet sont principalement identifiés en recherchant les changements d'états de l'objet. Ainsi, le montage robotisé que nous avons développé, comme un objet concret pour penser objet, se présentera dès lors comme un facilitateur pour l'identification de l'information recherchée dans le cadre d'une conception par objets.

L'intérêt principal de cette R&D est d'avoir permis à l'étudiant de confronter l'exécution de son propre algorithme à la réalité, c'est-à-dire d'évaluer le fonctionnement d'un programme abstrait par la visualisation du comportement d'un objet concret. Puisque dans cet environnement la rétroaction effectuée par ce comportement sur un algorithme abstrait s'effectue de manière sensorielle, le processus d'apprentissage que nous avons développé ici permet ainsi à l'étudiant de travailler de manière plus autonome. La puissance de cette rétroaction se situe surtout au niveau de la facilité de détection d'une erreur se manifestant d'abord par un mauvais fonctionnement du montage robotisé ce qui facilitera ensuite le repérage par l'étudiant du code fautif correspondant dans le programme.

Notre préoccupation à l'égard de l'enseignement et de l'apprentissage de la programmation s'exprimait à travers quatre difficultés: l'hétérogénéité de la clientèle étudiante; l'abstraction en vue de la transposition de la solution dans le langage informatique; la détection, l'interprétation et la correction d'erreurs; la nécessité d'intégrer les connaissances du domaine d'application en plus des connaissances du domaine informatique. L'environnement développé dans le cadre de cette recherche se veut un élément de solution ou une solution à ces difficultés. La diversité des moyens offerts s'est imposée par une diversité de formations initiales et une multitude de méthodes d'apprentissages. Les exercices réalisés à l'aide du montage électronique placent l'étudiant dans une situation d'apprentissage plus concrète. La visualisation du comportement du montage facilite l'identification de la séquence des opérations à réaliser et par le fait même la transposition de la solution. Les différences perçues entre le comportement observé et le comportement attendu sont des indicateurs facilitant la manipulation d'erreurs. De plus, les étudiants ont pu réaliser rapidement les exercices proposées puisqu'ils n'exigeaient pas ou très peu de connaissances du domaine d'application.

Le potentiel évident d'un environnement comme le nôtre pourra amener à revoir l'enseignement traditionnel en consacrant plutôt les périodes de cours à la résolution de problèmes comme ceux présentés lors de l'atelier de programmation, avec le tutoriel

comme ressource. Aussi, en admettant le caractère évolutif de cet environnement qui, en combinant tutoriel élaboré et ateliers de robotique, permettra de passer progressivement d'un enseignement traditionnel, à une approche par problèmes et éventuellement à une pédagogie par projets.

Deek (1999) précisait qu'une lacune des environnements d'aide à l'apprentissage de la programmation est la faiblesse à engager l'étudiant dans une méthode de résolution de problème. Les environnements sont rigides au sens où ils conduisent l'étudiant vers une solution cible. À cet égard, notre environnement se distingue en laissant toute la place à la créativité de l'étudiant en lui proposant un problème clair et bien défini. L'étudiant peut réaliser en toute quiétude son problème en appliquant sa méthode de résolution de problème. La résolution du problème est d'une grande importance pour nous puisqu'il offre à l'étudiant un cadre idéal pour développer ses compétences en génie logiciel. L'étudiant est ainsi placé dans une situation idéale lui permettant de réaliser plusieurs tâches associées au développement logiciel: l'identification du problème précisant les exigences et les spécifications, l'analyse et le design permettant d'exprimer le problème à l'aide de divers diagrammes, l'implantation de la solution à l'aide d'un environnement de programmation et son langage. Pour le génie logiciel, il s'agira d'une situation d'apprentissage exceptionnelle lorsque le problème s'inscrira dans un véritable projet de développement logiciel, sachant que la réalisation et la gestion de projets constitueront les principales tâches qu'auront à accomplir nos futurs ingénieurs logiciels.

Dans ce type d'environnement, le comportement de l'objet robotisé peut être considéré comme un analogue concret de la structure abstraite du programme. Cette situation, où l'étudiant peut visualiser en même temps le concret et l'abstrait s'apparente à la situation expérimentale en sciences décrite par le concept métaphorique de lunette cognitive (Nonnon, 1986).

Comme le résultat de cette recherche de développement visait l'élaboration d'un environnement support à l'apprentissage de la programmation, la suite logique sera d'évaluer l'impact réel de celui-ci sur l'apprentissage de l'étudiant et sur la méthode d'enseignement du professeur. Nous aimerions également poursuivre notre recherche en

proposant de nouveaux exercices à l'aide du montage robotisé. Les composants du montage offrent une panoplie de possibilités, comme la construction d'un ascenseur qui se déplacerait par l'entremise du moteur commandé par le microcontrôleur constituerait une activité accessible et intéressante à l'étudiant. Par exemple, la réponse à une requête qui devrait tenir compte des requêtes précédentes devrait offrir un défi nécessitant une réflexion sérieuse pour l'apprenant. La visualisation du déplacement de l'ascenseur lors de l'exécution de son programme offrirait alors à l'étudiant une validation de la logique implantée. Finalement, nous pourrions mettre à contribution l'expertise développée dans cette recherche pour proposer un modèle de recherche et développement spécifique aux EIAH. En effet, la démarche que nous avons effectuée au cours de cette recherche, notamment l'atelier de programmation est une contribution originale à l'ingénierie des EIAH.

Nous aimerions rappeler également que l'enjeu de cette recherche était principalement de nature ontologique. À cet égard, nous croyons que les savoirs et savoir-faire didactiques acquis au cours de cette recherche permettront d'améliorer nos cours d'initiation à l'informatique. Ainsi, le constat d'un enseignement basé sur l'objectivisme plutôt que sur le constructivisme nous incite à privilégier un enseignement centré étudiant plutôt qu'un enseignement centré professeur. Dès lors, la porte s'ouvre sur une panoplie de méthodes pédagogiques à concevoir, à développer et à expérimenter.

Cette recherche, qui se déclare à enjeu ontologique, avait aussi un enjeu bien pragmatique, celui de développer un environnement propice à l'apprentissage de la programmation. Une recherche-action visant l'appropriation progressive d'un tel environnement et l'étude des transformations qui en découlent au niveau de l'enseignement et de l'apprentissage de la programmation serait une suite logique à cette recherche de développement didactique.

RÉFÉRENCES

ACTES du 19e Colloque AIPU 2002 : *Les méthodes actives dans l'enseignement supérieur.* Louvain-la-Neuve, Belgique, CD-ROM.

ADELSON, B.(1981) *Problem solving and the development of abstract categories in programming languages.* Memory and cognition, 9 (4), p.422- 433.

ADELSON , B.(1985) *Comparing natural and abstract categories: a case study from computer science.* Cognitive Science, 9, 1985, p.417- 430.

ADELSON, B., SOLOWAY, E. (1985) *The role of domain experience in software design.* IEEE Transactions on Software Enfineering, SE-11, pp 1351-1360.

ANDERSON, J. R., BOYLE, C.F., FARRELL, R., REISNER, B. J. (1987) *Cognitive Principles in the design of computer tutors.* In P. Morris: Modelling cognition, Chichester: John Wiley.

ANJANEYULU, K.S.R. (1994) *Bug analysis of Pascal programs.* ACM SIGPLAN Notices, 29 (4), pp 15-22.

ARSAC, J. (1988) *La didactique de l'informatique: un problème ouvert?* in Actes du Colloque francophone sur la didactique de l'informatique, Université René Descartes, Paris: Association E.P.I.

ASCHNER, M., GALLAGHER, J.(1965) *A system for classifying thought processes in the context of verbal interaction.* University of Illinois Institute for research on exceptional children, Urbana, 1965.

ASTOLFI, J.P., DAROT, É., GINSBURGER-VOGEL, Y., TOUSSAINT, J. (1997) *Mots-clés de la didactique des sciences.* Paris, De Boeck Université.

ATKINSON, R. C. & SHIFFRIN, R. M. (1968) *Human memory: A proposed system and its control processes.* In K.W.Spence and J. T. Spence (Eds.), *The Psychology of Learning and Motivation: Advances in Research and Theor y*(Vol. 2, pp. 89-195). New York: Academic Press.

ATWOOD, M. E. et RAMSEY, H. R. (1978) *Cognitive structures in the comprehension and memory of computer programs: an investigation of computer program debugging.* US Army Research Institute for the Behavioural and Social Sciences, Technical report (TR-78-A21), Va: Alexandra.

BALACHEFF, N. (1999) *Cours de DEA : Environnements informatiques d'apprentissage humain (EIAH)* http://www-didactique.imag.fr/CoursDEA1/

BARG, M. et al. (2000) *Problem-Based learning for foundation computer science courses*. Computer Science Education, Vol. 10, No.2, pp.109-128.

BARR, A., BEARD, M., ATKINSON, R.C. (1975) *A rationale and description of a CAI program to teach the BASIC programming language*. Instructional Science, 4, pp. 1-31.

BECK, K. (2000) *Extreme Programming Explained : embrace change*. Reading MA: Addisson-Wesley

BEDNAR, A.K., CUNNIGHAM, D., DUFFY, T.M., PERRY,J.D. (1992) *Theory into practice: how do we link?* In T.M. Duffy & D.H. Jonassen (Eds.) , Constructivism and the technology of instruction: a conversation, Hillsdale, NJ:Lawrence Erlbaum Associates, pp 17-34.

BISSON G. (2000) *La similarité: une notion symbolique/numérique*. Apprentissage symbolique-numérique (tome 2). Eds Moulet, Brito. Editions CEPADUES. 2000. (preprint PDF :
http://www-leibniz.imag.fr/Apprentissage/Publications/Bisson-SN-preprint.PDF)

BLOOM, B. (1956) *Taxonomy of educational objectives*. vol. 1, McKay, New York.

BOEHM, B.W. (1988). *A spiral model of software development and enhancement*. IEEE Compter, 21(5) p.61-72.

BONAR, J. et CUNNINGHAM, R. (1988) *Bridge: an intelligent tutor for thinking about programming.* In J. SELF Ed., Artificial intelligence and human learning, LONDON, Chapman et Hall.

BORELLA, J. (2000) *Penser L'analogie*. Éditions Ad Solem, Genève, 221 p.

BOUDREAULT, H. (2002) *Développement d'un outil informatique par le design des situations d'enseignement en formation professionnel.* Thèse de doctorat, Université de Montréal.

BOUDREAULT, Y. (1992) *Les algorithmes au banc des accusés*. Compte rendu du huitième congrès canadien de l'éducation en ingénierie, pp. 567-574.

BOUDREAULT, Y., GUERFALI, W. (1996) *C et un peu + : programmation et résolution de problèmes.* Éditions de l'École Polytechnique de Montréal, Montréal, 609 p.

BOUDREAULT, Y., GUERFALI, W. (2001) *C et un peu + : programmation et résolution de problèmes.* 2e édition, Éditions de l'École Polytechnique de Montréal, Montréal.

BOUTINET, J.P. (1993) *Effets et méfaits du projet*. in Sciences Humaines no 39 mai 1993. http://tecfa.unige.ch/~lombardf/ped_projet/effets_mefaits_projet.html

BREDEWEG, B. et SCHUT, C. (1991) *Cognitive plausibility of a conceptual framework for modeling problem solving expertise.* Proceedings of the thirteent annual conference of te Cognitive Science Society, pp. 473-479. Hillsdale, NJ:Erlbaum.

BROWN, J.S., BURGON, R.R., BELL, A. (1974) *A intelligent CAI System that Reasons and Understands*. Cambridge, Massachusetts: Bolt Beranek and Newman.

BROOKS, R. (1977) *Towards a theory of the cognitive processes in computer programming*. International Journal of Man-Machine Studies, 9, p. 737-751.

BROOKS, R. (1980) *Studying programmers behavior experimentally: the problems of proper methodology*. Communications of the ACM, 23 (4), p. 207-213.

BROUSSEAU, G. (1983) *Les obstacles épistémologiques et les problèmes en mathématiques*. In revue Recherches en Didactique des Mathématiques, La pensée sauvage, Grenoble,Vol 4, n°2, pp 165-198.

BROUSSEAU, G. (1986) *Fondements et méthodes de la didactique des mathématiques*. Recherche en didactique des mathématiques, 7(2), 33-115.

BROUSSEAU, G. (1998) *Théorie des situations didactiques : didactique des mathématiques 1970-1990* ; textes rassemblés et préparés par Nicolas Balacheff ... [et al.], Ed. La pensée sauvage, Grenoble, 395 p.

CARBONELL, J.R. (1970) *AI in CAI: An artificial approach to computer-aided instruction.* IEEE Transactions on Man-Machine Systems, MMS-11, pp. 190-202.

CATALANO, G.D. et CATALANO, K.(1999) *Transformation: from teacher-centered to student-centered engineering education*. Journal of engineering education, janvier, pp. 59-64.

CERVERA, D. (1998) *Élaboration d'un environnement d'expérimentation en simulation incluant un cadre théorique pour l'apprentissage de l'énergie des fluides*. Thèse de doctorat, Université de Montréal.

CHARCONNET-MÉLIÈS, J. (1998) *Analogie et logique naturelle*, Université Paris VIII, CALDE. http://www.hatt.nom.fr/rhetorique/articl19.htm

CHASE, W. G. et SIMON, H. A. (1973) *Perception in chess*. Cognitive Psychology, 4, p.55-81.

CHATEL, S. et DÉTIENNE, F. (1996) *Strategies in object-oriented design.* Acta Psychologica, 91, p. 245-269.

CHEVALLARD, Y. (1985) *La transposition didactique.* La pensée Sauvage, Grenoble.

CHI, M. T. H., FELTOVITCH, P.J. et GLASER, R. (1981) *Categorization and representation of physics problems by experts and novices.* Cognitive Science, 5, p. 1221-1252.

CLEMENT, C.A. et GENTNER, D. (1991) *Systematicity as a selection constraint in analogical mapping.* Cognitive Science **15**:89-132.

COAD, P., DE LUCA,J., LEFEBVRE, E. (1999) *Java Modeling in Color with UML,* Prentice Hall, 221 p.

CONNELLY, C., BIERMANN, A.W., PENNOCK, D. et WU, P. (1996) *Home-study software: flexible, interactive and distributed software for independent study.* in Proceedings of the 27th SIGCSE technical symposium on computer science education, New York, New York: ACM Press, pp. 63-67.

CÔTÉ, R.L. (1986) *Psychologie de l'apprentissage et enseignement. Une approche modulaire d'auto-formation.* Boucherville, Gaëtan Morin, p. 180.

CREVIER, F. (1998) *Conception et validation d'une méthode d'ingénierie didactique (AGD).* Thèse de doctorat, Université de Montréal.

CURTIS B. (1984) *Fifteen Years of psychology in Software engineering: individual differences and cognitive science.* Proceeding of the Seventh Conference on Software Engineering, Orlando, Florida USA, 26-29 mars.

DANIEL, A. (2002) *Staging your classes.* in ASEE prism nov., pp 41-42.

DANTONIO, M. (1990) *How can we create thinkers?* National Education Service: Bloomington.

D'ASTOUS, P. (1999) *Approche de mesure d'analyse des réunions de révision technique du processus de génie logiciel.* Thèse de doctorat, École Polytechnique de Montréal, 219 p.

DAVIES, S.P. (1994) *Knowledge restructuring and the acquisition of programming expertise.* International Journal of Human-Computer Studies, 40, p. 703-726.

DEEK, F.P. (1997) *An integrated problem solving and program development environment,* Ph. D. dissertation, New Jersey Institute of Technology.

DEEK, F.P. (1999) *A framework for an automated problem solving and program development environment.* Society for Design and Process Science, vol. 3, no. 3 pp. 1-13.

DEEK, F.P. (1999) *The Software Process: A Parallel Approach through Problem Solving and Program Development.* Computer Science Education, Vol.9, No.1, pp 43-70.

DEEK, F.P., KIMMEL, H., McHUGH, J.A. (1998) *Pedagogical changes in the delivery of the first course in computer science: problem solving then programming.* Journal of Engineering Education, 87 (3), pp. 313-320.

de KLEER, J. et BROWN, J.S. (1984) A qualitative physics based on confluences. Artificial Intelligence **24** :7-83.

de LANDSHEERE, G. (1985) *Introduction à la recherche en éducation.* Liège, Thone.

DERSHEM, H.L., BARTH, W.L., BOURSHER, C.J. et BROWN, D.P. (1996) *Data structures with Ada packages, laboratories and animations.* Proceedings of the first Australian conference on computer science education, New York, New York :ACM Press, pp. 32-40.

DÉTIENNE, F. (1990) *Difficulties in Designing with an object-oriented language: an empirical study.* In D. diaper, D. Gilmore, G. Cockton et B. Schaker Eds., Human Computer Interaction, Proceedings of INTERACT'90, North Holland, pp 971-976.

DÉTIENNE, F. (1990) *Program Understanding and Knowledge Organization: the Influence of Acquired Schemas.* In P. FALZON Ed., Cognitive Ergonomics: understanding, learning and designing Human-Computer Interaction, Academic Press, London, p. 245-256.

DÉTIENNE, F. (1990) *Expert Programming Knowledge: A Schema-Based Approach.* In J-M HOC, T.R.G. GREEN, R. SAMURÇAY, D. GILMORE Eds., Psychology of programming, Academic Press, People and Computer Series, p. 205-222.

DÉTIENNE, F. (1990) *Un exemple d'évaluation ergonomique d'un système de programmation orienté-objet, le système O2.* Actes du congrès ERGO IA'90, Biarritz, 19-21 septembre.

DÉTIENNE, F. (1993) *Acquiring experience in object-oriented programming: effects on design strategies.* In E. Lemut, B. du Boulay et G. Dettori Eds., Cognitive Models and Intelligent Environments for Learning Programming, Springer-Verlag, NATO ASI Series.

DÉTIENNE, F. (1995) *Design strategies and knowledge in object-oriented programming: effects of experience.* Human-Computer Interaction, 10 (2 et 3), pp 129-170.

DÉTIENNE, F. (1998) *Génie logiciel et psychologie de la programmation.* Éditions Hermès, Paris, 184 p.

DÉTIENNE, F., et SOLOWAY, E. (1990) *An Empirically-Derived Control Structure for the Process of Program Understanding.* In R. BROOKS Ed., special issue: what programmers know, International Journal of Man-Machine Studies, 33 (3), p. 323-342.

Dictionnaire Le Petit Larousse Illustré 1999, Librairie Larousse, p. 63.

DOUAY, F. (1987) *LA CONTRE-ANALOGIE: réflexion sur la récusation de certaines analogies pourtant bien formées cognitivement.* Polycopié du G.T.A. Recueil de textes ndeg.10, Paris, janvier.
http://www.revue-texto.net/nouveautes/Contre-Analogie/Contre-analogie.html

DUBOIS, L. (1999) *La pédagogie de projet.* enseignant à l'école primaire Onex-Bosson, Suisse.
http://www.edunet.ch/classes/c9/dubois/didact/projet.htm

DUCHATEAU, C. (2002) *Mais qu'est la didactique de l'informatique devenue?* in Les technologies en éducation : perspectives de recherche et questions vives, Actes du symposium international francophone, Paris, Maison des sciences de l'homme, p.33-42.

EBRAHIMI, A. (1994) *Novice programmer error: language constructs and plan composition.* International Journal of Human-Computer Studies, 41, pp. 457-480.

EHRENBURG, S. et L. EHRENBUR, G. (1978) *Building and applying strategies for intellectual competencies in students (basics).* Institute for curriculum and instruction, Miami.

ENNIS, D. (1994) *Combining problem solving and programming instruction to increase the problem solving abilities in high school students.* Journal of Research on Computing in Education, 26 (4), pp. 488-496.

Étude no. 9 : l'enseignement de l'informatique
http://tecfa.unige.ch/tecfa/research/pnr33/french/pnrweb-67.html

FALZON, P., BISSERET, A., BONNARDEL, N., DARSES F., DÉTIENNE, F., VISSER, W. (1990) *Les activités de conceptions : l'approche de l'ergonomie cognitive.* Actes du Colloque Recherches sur le design, incitations, implications, interactions, Compiègne, 17-19 octobre.

FELDER, R. (1997) *Who needs these headaches?* Success 101, No.4, Fall, 1997.

FORBUS, K.D. (1984) *Qualitative process theory*. Artificial Intelligence 24(1):85-168.

FORBUS, K.D. (2000) *Exploring Analogy in the Large.* The analogical mind: perspectives from cognitive science ed. Par Dedre Gentner, Keith J. Holyoak et Boicho N. Kokinov, The MIT Press, Massachusets, pp 23-58.

FORBUS, K, GENTNER, D. et LAW, K. (1995) *MAC/FAC: a model of similarity-based retrieval*. Cognitive Science 19(2): 141-205

FOSNOT, C.T. (1996) *Constructivism: theory, perspectives, and practice.* New-York, NY: Teachers College Press.

FOURNIER F. (2001) *Un environnement d'apprentissage technologique pour la compréhension du concept de mesure en sciences expérimentales.* Thèse Ph.D, Université de Montréal.

GAGNÉ, G., LASURE, R., SPRENGER-CHAROLLES L. et ROPÉ, F. (1989), *Recherche en didactique et acquisition du français langue maternelle.* Montréal, De Boeck-Université.

GENTNER, D. (1983) *Structure-mapping: A theoretical framework for analogy.* Cognitive Science, 7(2):155--170, 1983.

GENTNER D. (1988) *Metaphor as structure mapping: The relational shift.* Child Development **59**:47-59.

GENTNER, D. (1989) *The mechanisms of analogical learning.* Édition Vosniadou, S., & Ortony, A., Similarity and Analogical Reasoning. NewYork: CUP.

GENTNER D., BOWDLE B.F., P. WOLF et BORONAT C. (2001) *Metaphor is like analogy.* in The analogical mind: perspectives from cognitive science ed. Par Dedre Gentner, Keith J. Holyoak et Boicho N. Kokinov, The MIT Press, Massachusets, pp 199-253.

GENTNER, D. et GENTNER, D.R. (1983) *Flowing waters of teeming crowds: mental models of electricit.*, dans Mental Models, edited by Dedre GENTNER et Albert L. STEVENS, Lawrence Erlbaum associates publishers, Hillsdale New Jersey, p.99-129.

GENTNER, D. et FRANCE, I. M. (1988) *The verb mutability effect : Studies of the combinatorial semantics of nouns and verbs*. In S. L. Smal, G.W. Cottrel, and M.D. Tanenaus, Eds., Lexical ambiguity resolution: Perspectives from psycholinguistics, neuropsychology, and artificial intelligence, pp. 343-382. San Mateo, CA:Kaufmann.

GENTNER, D. et MARKMAN, A.B. (1997) *Structure mapping in analogy and similarity*. American Psychologist 52:45-56.

GENTNER, D. et RATERMANN, M.J. (1991) Language and the career of similarity. In S.A. Gelman and J.P. Brynes, Eds., Persperctives on thought and language: Interrelations in development, pp. 225-227. London:Cambridge University Press.

GENTNER, D., RATTERMANN, M.J. et FORBUS, K.D.(1993) *The roles of similarity in transfer: Separating retrieval from inferential soundness.* Cognitive Psychology **25**:524-575.

GENTNER , D. et TOUPIN, C. (1986) *Systematicity and surface similarity in the development of analogy*. Cognitive Science **10**:277-300.

GLUCKSBERB, S., McGLONE et MANFREDI, D. (1997) *Property attribution in metaphor comprehension*. Journal of memory and language 36:50-67.

GOGUELIN, P. (1991) La *formation-animation une vocation*. 2e édition, ESF éditeur, Paris, 277 p.

GORRIZ, C.M., MEDINA, C. (2000) *Engaging girls with computers through software games,* Communication of the ACM, 43, pp 42-49.

GRAHAM, I. (2001) Object-oriented methods : principles & practice. 2e edition, Addison-Wesley, Harlow, 832 p.

GRANGER, G.-G. (1995) *La science et les sciences.* Paris, Presses Universitaires de France.

GREENING, T. (1996) *Teaching and learning essential computer science skills: the Unix example.* SIGCSE Bulletin, 28(2): 21-24- 30.

GREENING, T. (2000) *Emerging Constructivist Forces in Computer Science Education: Shaping a New Future?* in Computer science education in the 21 st Century, Springer-Verlag, New-York, pp.47-80.

GREINER, R. (1988). *Learning by understanding analogies*. Artificial Intellignece 35 : 81-125

GUILFORD, J.P. (1954) *Psychometric methods.* 2e edition, New York: Mc Graw-Hill, chap. 12.

GUINDON, R. (1990) *Designing the design process: exploiting opportunistic thoughts.* Human-Computer Interaction, 5, p. 305-344.

GUINDON, R., KRASNER, H. et CURTIS, B. (1987) *Breakdowns and Processes during the Early Activities of Software Design by Professionals.* In G. M. OLSON, S. SHEPPARD and E. SOLOWAY, Eds., Empirical Studies of Programmers, Second Workshop, Ablex.

HANSEN, C.(1994) *Questioning techniques for active classroom.*, in Halperin, D., ed. Changing college classrooms, Jossey-Bass, San Francisco, pp. 93-106.

HAYES, J.R. et FLOWER, L (1980). *Identifying the organization of writing processes.* In L. W. GREGG and EW. STEINBERG Eds., Cognitive processes in writing, Hillsdale, NJ, Werlbaum, p. 3-30.

HOC, J.-M. (1981) *Planning and Direction of Problem-Solving in Structured Programming: an Empirical Comparison between two methods.* International Journal of Man-Machine Studies, 15 (4), p. 363-383.

HOC, J.-M. (1982) *L'étude psychologique de l'activité de programmation: une revue de question.* Technique et Sciences Informatiques, 1 (5), p. 383-392.

HOC, J.-M. (1987) *Psychologie cognitive de la planification.* PUG, Collection Sciences et Technologies de la connaissance.

HOC, J.M., GREEN, T.R.G., SAMURCAY, R., GILMORE, D.J. (Eds.) (1990) *Psychology of Programming.* London: Academic Press.

HOLYOAK, K.J. et KOH, K. (1987) *Surface and structural similarity in analogical transfer.* Memory and Cognition **15**:332-340.

JACOBSON, I., BOOCH, G., RUMBAUGH, J. (1999) *The Unified Software Development Process*, Reading MA: Addison-Wesley.

JOHNSON, D.W., JOHNSON, R.T., SMITH, K.A. (1991) *Active learning: Cooperation in the college classroom.* Interaction Book Company.

JONNAERT, Ph. et LAVEAULT, D.(1994) *Évaluation de la familiarité de la tâche: quelle confiance accorder à la perception de l'élève.* Revue des sciences de l'éducation, 20(2), pp 271-291.

JOSHUA, S. et DUPIN, J.J. (1989) *Représentations et modélisations : le « débat scientifique » dans la classe et l'apprentissage de la physique.* Editions Peter Lang, Berne, 220 p.

KANT, E. et NEWELL, A. (1984) *Problem solving techniques for the design of algorithms,* Information Processing and Management, 20, p. 97-118.

KEANE, M.T. (1996) *On adaptation in analogy: Tests of pragmatic importance and adaptability in analogical problem solving*. The Quarterly Journal of Experimental Psychology **49**/A(4):1062-1085.

KEANE, M.T., et BRAYSHAW, M. (1988) *The incremental analogical machine : a computational model of analogy.* in D. Sleeman, Ed., Third european working session on machine learning, San Mateo, CA:Kaufmann, pp. 53-62.

KEANE, T.K. et COSTELLO, F. (2001) *Setting limits on analogy :why conceptual combination is not structural alignment.* in The analogical mind: perspectives from cognitive science ed. Par Dedre Gentner, Keith J. Holyoak et Boicho N. Kokinov, The MIT Press, Massachusets, pp 287-312.

KEELER, C.M., ANSON, R. (1995) *An assessment of cooperative learning used for basic computer skills instruction in the college classroom.* Journal of educational computing research, 12, pp 379-393.

KIM, M.K., SHARP, J.M., THOMPSON, A.D. (1998) *Effects of integrating problem solving, interactive multimedia, and constructivism in teacher education.* Journal of Educational Computing Research, 19, pp 83-108.

KING, A.(1994) *Inquiry as a tool for critical thinking*. in Halperin, D., ed. Changing college classrooms, Jossey-Bass, San Francisco, pp 13-38.

KINTSCH, W. et VAN DIJK, T. (1978) *Toward a model of text comprehension and production*. Psychological review, 85, p. 363-394.

KIRCHER, E. (1984) *Analogies for the electric circuit?* In R. Duit, W. Jung & C.V. Phöneck (Eds): Aspects of Understanding Electricity, pp. 299-310, Kiel, West Germany:IPN.

KOYANAGI, M. (1999) *Theory Wars: Objectivism(Behavioral Psychology) versus Constructivism(Cognitive Psychology)*
http://www.ils.unc.edu/disted/cmi/final2.html

KRUTCHEN, P. (1999) *The Rational Unified Process*, Reading MA: Addison-Wesley.

LAKOFF, G. (1990) *The invariance hypothesis: Is abstract reason based on image-schemas*? Cognigive Linguistics 1(1):39-74.

LAROCHELLE, M. et BEDNARZ, N. (1994) *À propos du constructivisme et de l'Éducation.* Revue des sciences de l'éducation. (20) 1, pp 5-20.

LASSALINE, M.E. (1996) *Structural alignment in induction and similarity*. Journal of Experimental Psychology:Learning, Memory, and Cognition **22**(3):754-770.

LAUGHERY, K.R. (1985) *Human Factors in Software Engineering: A review of the Literature.* The journal of Systems and Software, 5, 1985, p. 3-14.

LAWRENCE, A.W., BADRE, A.N. et STASKO, S.T. (1994) *Empirically evaluating the use of animations to teach algorithms.* Georgia Institute of Technology Technical, Report GIT-GVU-94-07. ftp:/ftp.cc.gatech.edu/pub/gvu/tech-reports/94-07.ps.Z

LEBRUN, M. (2002) *Des méthodes actives pour une utilisation effective des technologies.* http://www.ipm.ucl.ac.be/Marcell/TECHPED/MethTech.html

LEE, A., PENNINGTON, N. (1994) *The effects of programming on cognitive activities in design.* International Journal of Human-Computer Studies, 40, pp 577-601.

LEMUT, E.B., du BOULAY, G., DETTORI, (Eds) (1993) *Cognitive Models and Intelligent environments for Learning programming.* Berlin: Springer-Verlag.

MARGOLINAS, Cl. (1993) *De l'importance du vrai et du faux dans la classe de mathématiques*. Grenobles : La Pensée sauvage, p. 229.

MARKMAN, A.B. (1997) *Constraints on analogical inference*. Cognitive Science **21**(4):373-418.

MARKMAN, A.B. et GENTNER, D.(1993) *Stuctural alignment during similarity comparisons.* Cognitive Psycology 25:431-467.

MAYER, R. (1981) *The psychology of how novice learn computer programming. ACM* Computing Surveys, 13 (1), pp. 121-141.

MAYER, R.E. (1988) *Teaching and Learning Computer Programming.* Hilldale, New Jersey: Lawrence Erlbaum.

McGILL, T.M. et HOBBS, V. (1996) *A supplementary package for distance education students studying introductory programming.* Proceedings of the 27th SIGCSE Technical Symposium on Computer Science Education, New York, New York: ACM Press, pp. 73-77.

McGARTH, D. (1990) *Eight ways to get beginners involved in programming.* The computing teacher, 18, pp 19-21.

McKEITHEN, K.B., REITMAN, J.S., REUTER, H.H. et HIRTLE, S.C. (1981) *Knowledge organization and skill differences in computer programmers.* Cognitive Psychology, 13, pp. 307-325.

MEDIN, D.L. , GOLDSTONE, R.L. et GENTNER, D. (1993) *Respects for similarity.* Psychological Review 100(2):234-278.

MENDELSOHN, P. (1998). *La didactique de l'informatique : présentation.* http://tecfa.unige.ch/themes/sa2/ter-did-dos2-didac-info.html

MOHER, T. et SCHNEIDER, G.M. (1982) *Methodology and experimental research in software engineering*. International Journal of Man-Machine Studies, 16, p. 65-87.

MURPHY, G.L. (1996) *On metaphoric representation.* Cognition 60(2):173-204.

NEWELL, A. et SIMON, H.(1972) A *Human Problem Solving.* New York, Prentice-Hall.

NONNON, P. (1986) *Laboratoire d'initiation aux sciences assistée par ordinateur*. Faculté des Sciences de l'Éducation, UdM. Thèse de doctorat, Université de Montréal.

NONNON, P. (1993) *Proposition d'un modèle de recherche-développement technologique en éducation.* in Regard sur la robotique pédagogique. Acte du quatrième colloque international de robotique pédagogique, Liège, Université de Liège.

NONNON, P. (1998) *Intégration du réel et du virtuel en sciences expérimentales*. Actes des 8e Journées Informatique et Pédagogique des Sciences Physiques de Montpellier, Montpellier, France 12-14 mars.

NORMAN, G.R., SCHMIDT, H.G. (1992) *The psychological basis of problem-based learning: a review of the evidence*. Acad. Med. 67: pp. 557-565

PAPERT, S. (1980) *Mindstorms: Children, Computers and Powerful Ideas.* New York: Basic Books.

PAPERT, S. (1994) *L'enfant et la machine à connaître: repenser l'école à l'ère de l'ordinateur*. Dunod, Paris, 225 p.

PENNINGTON, N. (1987) *Stimulus structures and mental representation in expert comprehension of computer programs.* Cognitive Psychology, 19, p. 295-341, 1987.

PENNINGTON, N. et GRABOWSKI, B. (1990) The *tasks of programming*. in J.-M. Hoc, T.R.G. Green, R. Samurcay, and D. Gilmore Eds., Psychology of programming, Academic Press, p.145-162.

PENNINGTON, N., LEE, A., REHDER, B. (1995) *Cognitive activities and levels of abstraction in procedural and object-oriented design.* Human-Computer Interaction, 10 (2 et 3) pp 171-226

PERKINS, D.N., HANCOCK, C., HOBBS, R., MARTIN, F., SIMMONS, R. (1986) *Conditions of learning in novice programmers.* Journal of Educational Computing Research, 2 (1), pp 37-56.

PERRRENOUD, P.(1998) *Réussir ou comprendre ?Les dilemmes classiques d'une démarche de projet.* Faculté de psychologie et des sciences de l'éducation, Université de Genève.
http://www.unige.ch/fapse/SSE/teachers/perrenoud/php_main/php_1998/1998_39.html

PERRRENOUD, P.(1999), *Apprendre à l'école à travers des projets: pourquoi ? comment ?* Faculté de psychologie et des sciences de l'éducation, U. de Genève.
http://www.unige.ch/fapse/SSE/teachers/perrenoud/php_main/Textes_1999.html

PIAGET, J. (1970) *Pour une théorie de la connaissance*, Psychologie et épistémologie. Paris : Gonthier.

PLOIX, D. (1999) *Élaboration, Réalisation et Évaluation d'un Environnement de Programmation Analogique*. Thèse de doctorat, Université Paris 8.
http://www.ai.univ-paris8.fr/~damien/papiers/iac97/iac97.html

POLYA, G. (1945) *How to solve it*. Princeton, NJ: Princeton University Press.

PRICE, B.A., BAECKER, R.M. et SMALL, I.S. (1996) *A principled taxonomy of software visualization.* Journal of visual languages and computing, 4(3): pp.211-266.

RAISKY, C. et CAILLOT M. (1996) *Au-delà des didactiques, le didactique : débats autour de concepts fédérateurs*. De Boeck & Larcier S.A., Bruxelles, 278 p.

RAMSEY, P., RADA, R., ACQUAH, S. (1994) *Collaborative learning for computer science students.* Journal of computers in mathematics and science teaching, 13, pp 377-389.

RATTERMANN, M.J. et GENTNER, D.(1998) *The effect of language on similarity: The use of relational labels improves young children's performance in mapping task.* In K. Holyoak, D. Gentner, and B. Kokinov, Eds., Advances in analogy research: Integration of theory and data from the cognitive, computational, and neural sciences, pp. 274-282. Sofia: New Bulgarian University Press.

REY, M.T. (1992) *Où en est-on ? Où va-t-on ? en Suisse*. Actes de la troisième rencontre francophones de didactique de l'informatique. Paris: Association EPI.

RICHARD, J.-F. (1996) *La représentation mentale d'un dispositif du point de vue de son utilisation et de son fonctionnement*". Workshop Les Sciences Cognitives et la Conception des Systèmes Informatiques, Florianopolis, Brasil, Fev. 26-27.

RICHARD, J-F, BONNET, C. et GHIGLIONE, R. (1990) *Traité de Psychologie Cognitive 2, Le traitement de l'information symbolique*, Paris, Dunod.

RIOPEL, M. (2001) *Conception et mise a l'essai d'un système laboratoire intégrant l'expérimentation assistée par ordinateur et la simulation en mécanique.* Examen de synthèse du doctorat, Université de Montréal.

RIST, R.S. (1986) *Plans in Programming: Definition, Demonstration, and Development.* In E. SOLOWAY and S. IYENGAR, Eds., Empirical Studies of Programmers, First Workshop, Norwood, NJ, Ablex Publishing Corporation.

RIST ,R.S. (1989) *Schema creation in programming.* Cognitive Science, 13, p. 389-414.

RIST, R.S. (1991) *Knowledge creation and retrieval in program design: a comparison of novice and experienced programmers*, Human-Computer Interaction, 6, p. 1-46.

RIST, R.S. (1995) *Program structure and design.* Cognitive Science, 19, p. 507-56.

RIST, R.S. (1996) *System structure and design.* In W.D. GRAY and D.A. BOEHM-DAVIS, Eds., Empirical Studies of Programmers, Sixth Workshop, Ablex Publishing Corporation, Norwood, NF, p. 163-194.

ROBERTSON, S.P. (1990) *Knowledge Representations used by Computer Programmers.* Journal of the Washington Academy of Sciences, 80 (3), p. 116-137.

ROBERTSON, S.P et YU, C-C.(1990) *Common cognitive representations of program code across tasks and languages.* International Journal of Man-Machine Studies, 33, p. 343-360.

ROBILLARD, P.N. (1986) *Schematic pseudocode for program constructs and its computer automation by Schemacode.* Communications of the ACM, no. 11, p.1072-1089

ROBILLARD, P.N., KRUCHTEN, P., D'ASTOUS, P. (2002) *Software engineering process with the UPEDU.* Pearson Education, Boston, 346 p.

ROGALSKI, J., SAMURCAY, R. (1990) *Acquisition of programming knowledge and skills.* Psychology of Programming, in J.-M. Hoc, T.R.G. Green, R. Samurcay, D. Gilmore (Eds), London: Academic Press, pp. 157-174.

ROGALSKI, J., SAMURCAY, R. (1993) *Task analysis and cognitive model as a framework to analyze environments for learning programming.* In E. Lemut, B. du Boulay, G. Dettori (Eds), Cognitive Models and Intelligent environments for learning programming, Berlin: Springer-Verlag, pp. 6-19.

ROSCH, E. (1978) *Principles of categorization.* In E. Rosch and B. Lloyd, Eds., Coginition and categorization, Hilsdale, NJ, Laurence Erlbaum Associates, p. 28-49.

ROSCH ,E., MERVIS, C.B., GRAY, W., JOHNSON, D. et BOYES-BRAEM, P. (1975) *Basic objects in natural categories*, Cognitive Psychology, 8, p. 133-156.

ROSS, B.H. (1989) *Distinguishing types of superficial similarities: Different effects on the access and use earlier problems.* Journal of Experimental Psychology: Learning, Memory and Cognition **15**(3):456-468.

ROYCE, W.W. (1970) *Managing the development of large software systems : concepts and techniques.* In proceedings IEEE WESCON, p. 1-9, IEEE.

RUBINSTEIN, M. (1975) *Patterns of problem solving. Englewood Cliffs, NJ: Prentice-Hall.*

SCHANK, R. et ABELSON R.(1977) *Scripts-Plans-Goals and Understanding.* Hillsdale, NJ, Lawrence Erlbaum Associates.

SCHACKELFORD, R., BADRE, A. (1993) *Why can't smart students solve simple programming problems?* International Journal of Man-Machine Studies, 38, pp 985-997.

SCHOLTZ, J., WIEDENBECK, S. (1992) *The role of planning in learning a new programming language.* International Journal of Man-Machine Studies, 37, pp 191-214.

SCHOLTZ, J., WIEDENBECK, S. (1993) *An analysis of novice programmers learning a second language.* In Proceedings of the Fifth Workshop on Empirical Studies of Programmers, Palo Alto, CA., pp 187-205.

SHAFFER, C.A., HEATH, L.S. et YANG, J. (1996) *Using the Swan data structure visualization system for computer science education.* in Proceedings of the 27th SIGCSE technical symposium on computer science education, New York, New York: ACM Press, pp.140-144.

SHEIL, B.(1981) *The psychological study of programming.* ACM Computing Surveys, 13 (1), p. 101-120.

SHNEIDERMAN, B. (1980) *Software Psychology: Human Factors in Computer and Information Systems.* Boston, Massachusetts: Little, Brown and Company.

SIMON, H. A.(1998) *What we know about learning*, Journal of engineering education, octobre, pp.343-348.

SKINNER, B.F. (1953) *Science and human behavior.* New York, NY: Free P

SOKAL, R.R., ROHLF, F.J. (1995) *Biometry: the principles and practice of statistics in biological research.* 3e éd., New York:W.H. Freeman.

SOLOWAY, E. et ERHLICH, K. (1984) *Empirical Studies of Programming Knowledge.* IEEE Transactions on Software Engineering, SE-10 5, p. 595-609.

SOLOWAY, E., EHRLICH, K., BONAR, J., GREENSPAN, J. (1982) *What the novices know about programming?* In A.N. Badre and B. Shneiderman (Eds), Directions in Human-Computer Interaction, New York: Ablex Publishing corporation.

SPELLMAN, B.A. et HOLYOAK, K.J. (1992) *If Saddam is Hitler then who is George Bush ? Analogical mapping between systems of social roles.* Journal of Personality and Social Psychology **62**(6):913-933.

SPELLMAN, B.A. et HOLYOAK, K.J. (1996) *Pragmatics in analogical mapping.* Cognitive Psychology **31**:307-346.

TABA, H. (1978) *Hilda Taba teaching strategies program.* Institute for development, Miami.

TENNEY, Y. et GENTNER, D. (1984) *What makes water analogies accessible: Experiments on the water flow-analogy for electricity.* In R. Duit, W. Jung & C.V. Phöneck (Eds): Aspects of Understanding Electricity, pp. 311-318, Kiel, West Germany:IPN.

TERENZI, P.T., CABRERA, A.F., COLBECK, C.L., PARENTE, J.M. et BJORKLUND, S.A. (2001) *Collaborative learning vs. lecture/discussion: strudents' reported learning gains*, Journal of Engineering Education, Janvier, pp. 123-130.

TVERSKY ,A. (1977) *Features of similarity.* In Readings in Cognitive Science, Morgan Kaufmann 1988. (from Psychological Review 84, 327-352, 1977).

VAN DER MARREN, J.M. (1996) *Méthodes de recherche pour l'éducation.* 2e éd. Les presses de l'Université de Montréal, De Boeck Université.

VAN GORP, M.J., GRISSOM, S. (2001) *An empirical evaluation of using constructive classroom activities to teach introductory programming.* Computer Science Education, Vol.11, No. 3, pp. 247-260.

VISSER, W. *(1987) Strategies in Programming Programmable Controllers: A Field Study on a Professional Programmer.* In G.M. OLSON, S. SHEPPARD an E. SOLOWAY, Eds., Empirical Studies of Programmers, Second Workshop, Ablex, p. 217-230.

VISSER, W. et HOC, J-M. (1990) *Expert software design strategies*. In J-M Hoc, T.R.G. GREEN, R. SAMURÇAY and D. GILMORE, Eds., Psychology of Programming, Academic Press, p. 235-250.

VIVET, M. (2000) *Des robots pour apprendre*. Revue Sciences et Techniques Éducative, Vol.7 no.1, pp. 17-60.

VYGOTSKY, L. S. (1978) *Mind in society: The development of higher psy-chological processes*. Editions M. Cole, V. John-Steiner, S. Scribner & E. Souberman, MA: Harvard University Press.

WEINBERG, G.M. (1971) *The psychology of Computer Programming.* New York: Van Nostrand Reinhold.

WHITE, B. et FREDERIKSEN, J. (1990) *Causal model progressions as a foundation for intelligent learning environments*. Artificial Intelligence 42:99-157.

WILLS, C.E., DEREMER, D., McCAULEY, R.A., NULL, L. (1999) *Studying the Use of Peer Learning in the Introductory Computer Science Curriculum.* Computer Science Education, Vol.9, No.2, pp 71-88.

WIRTH, N. (1971) *Program development by stepwinse refinement.* Communications of the ACM, 14(4), pp. 221-227.

WIRTH, N. (1974) *On the composition of well-structured programs.* Computing Surveys, 6, p. 247-259.

WOODS, D.R. (2000) An Evidence-Based Strategy for Problem Solving. Journal of Engineering Education, Octobre, pp 443-459.

WU, M. et GENTNER, D. (1998) *Structure in category-based induction*. Proceedings of twentieth annual conference of the Cognitive Science Society, pp. 1154-1158, Hillsdale, NJ: Erlbaum.

ANNEXES

ANNEXE A

Les classes pour la communication avec le port série

Déclaration de trois classes permettant la communication par l'entremise du port série.

Classe 1 : CSerial

FILE : Serial.h
Auteur : Inconnu
Provenance : Article titré CSerial – A C++ Class for Serial Communications
De Tom Archer et Rick Leinecker
Site : http://www.codeguru.com/network/serial.shtml

```
class CSerial
{

public:
   CSerial();
   ~CSerial();

   BOOL Open( int nPort = 2, int nBaud = 9600 );
   BOOL Close( void );

   int ReadData( void *, int );
   int SendData( const char *, int );
   int ReadDataWaiting( void );

   BOOL IsOpened( void ) { return( m_bOpened ); }

protected:
   BOOL WriteCommByte(unsigned char );

   HANDLE m_hIDComDev;
   OVERLAPPED    m_OverlappedRead, m_OverlappedWrite;
   BOOL m_bOpened;

};
```

Classe 2 : port

FILE: hsPortClass.hpp
PROGRAM: hsPLAYPORT
AUTHOR: Harpreet Singh Juneja
Date: 04/04/2001


```
class port
{
public:
   port();
   ~port();

   void setistty(int);
   int getistty(void);
   void setport(u_int);
   u_int getport(void);

   void setbyte(u_int);
   u_int getbyte(void);

   void setbytemode(int);
   int getbytemode(void);

   void setentrymode(int);
   int getentrymode(void);

   void setecho(int);
   int getecho(void);
   void getinput(int,char **);
   void author(void);
   void usage(char *);
   void copyright(void);
   void prompt(void);
   char * token(char *,char **);
   void print_it(void);
   char * to_bin(u_int);
   void rdport(void);
   void wrtport(void);
   void rep_io(int);
   void toggle_bit(u_int);
   void showit(int,int,int);

private:
   int istty;
   u_int myport;
   u_int byte;
   int byte_mode;
   int entry_mode;
   int echo_command;
};
```

Classe 3 : CSerialPort

Module : SERIALPORT.H Purpose: Declaration for an MFC wrapper class for serial ports
Created: PJN / 31-05-1999


```
class CSerialPort : public CObject
{
public:

//Constructors / Destructors
 CSerialPort();
 ~CSerialPort();

//General Methods
 void Open(int nPort, DWORD dwBaud = 9600, Parity parity = NoParity, BYTE DataBits = 8,
        StopBits stopbits = OneStopBit, FlowControl fc = NoFlowControl, BOOL bOverlapped = FALSE);
 void Close();
 void Attach(HANDLE hComm);
 HANDLE Detach();
 operator HANDLE() const { return m_hComm; };
 BOOL IsOpen() const { return m_hComm != INVALID_HANDLE_VALUE; };
#ifdef _DEBUG
 void CSerialPort::Dump(CDumpContext& dc) const;
#endif

//Reading / Writing Methods
 DWORD Read(void* lpBuf, DWORD dwCount);
 BOOL Read(void* lpBuf, DWORD dwCount, OVERLAPPED& overlapped);
 void ReadEx(void* lpBuf, DWORD dwCount);
 DWORD Write(const void* lpBuf, DWORD dwCount);
 BOOL Write(const void* lpBuf, DWORD dwCount, OVERLAPPED& overlapped);
 void WriteEx(const void* lpBuf, DWORD dwCount);
 void TransmitChar(char cChar);
 void GetOverlappedResult(OVERLAPPED& overlapped, DWORD& dwBytesTransferred, BOOL bWait);
 void CancelIo();

//Configuration Methods
 void GetConfig(COMMCONFIG& config);
 static void GetDefaultConfig(int nPort, COMMCONFIG& config);
 void SetConfig(COMMCONFIG& Config);
 static void SetDefaultConfig(int nPort, COMMCONFIG& config);

//Misc RS232 Methods
 void ClearBreak();
 void SetBreak();
 void ClearError(DWORD& dwErrors);
 void GetStatus(COMSTAT& stat);
 void GetState(DCB& dcb);
 void SetState(DCB& dcb);
 void Escape(DWORD dwFunc);
 void ClearDTR();
 void ClearRTS();
 void SetDTR();
 void SetRTS();
 void SetXOFF();
 void SetXON();
 void GetProperties(COMMPROP& properties);
 void GetModemStatus(DWORD& dwModemStatus);

//Timeouts
 void SetTimeouts(COMMTIMEOUTS& timeouts);
 void GetTimeouts(COMMTIMEOUTS& timeouts);
 void Set0Timeout();
 void Set0WriteTimeout();
 void Set0ReadTimeout();

//Event Methods
 void SetMask(DWORD dwMask);
 void GetMask(DWORD& dwMask);
 void WaitEvent(DWORD& dwMask);
 void WaitEvent(DWORD& dwMask, OVERLAPPED& overlapped);

//Queue Methods
 void Flush();
 void Purge(DWORD dwFlags);
 void TerminateOutstandingWrites();
 void TerminateOutstandingReads();
 void ClearWriteBuffer();
 void ClearReadBuffer();
 void Setup(DWORD dwInQueue, DWORD dwOutQueue);

//Overridables
 virtual void OnCompletion(DWORD dwErrorCode, DWORD dwCount, LPOVERLAPPED lpOverlapped);

protected:
 HANDLE m_hComm;        //Handle to the comms port
 BOOL m_bOverlapped; //Is the port open in overlapped IO

 static void WINAPI _OnCompletion(DWORD dwErrorCode, DWORD dwCount, LPOVERLAPPED lpOverlapped);

DECLARE_DYNAMIC(CSerialPort)
};
```

ANNEXE B

Le fichier EspaceTravail.doc

Le fichier `EspaceTravail.doc` contient plusieurs informations utiles pour la réalisation des exercices de programmation avec le montage électronique robotisé. Ce fichier contient l'information pour organiser l'espace de travail du projet à créer dans l'environnement de développement Visual C++.

Fichier : EspaceTravail.doc

Procédure d'installation de l'espace de travail

1. Copier le fichier Utilitaires.h dans le répertoire INCLUDE du compilateur : C:\Program Files\Microsoft Visual Studio\VC98\Include

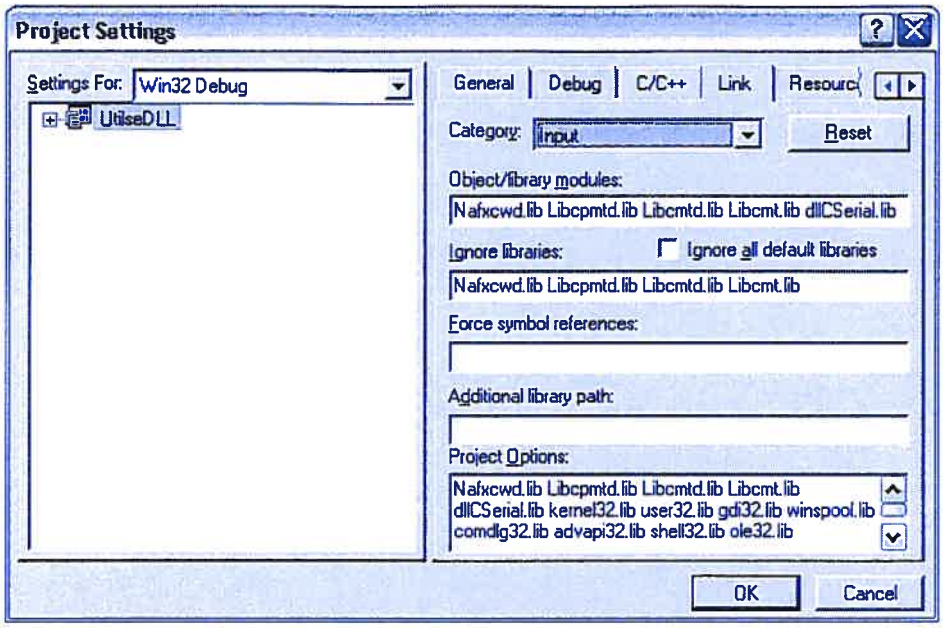
2. Copier le fichier dllCSerial.lib dans le répertoire LIB du compilateur : C:\Program Files\Microsoft Visual Studio\VC98\Lib

3. Placer le fichier dllCSerial.dll dans le répertoire Debug de l'espace de travail courant, i.e. celui que vous avez créer pour l'exercice

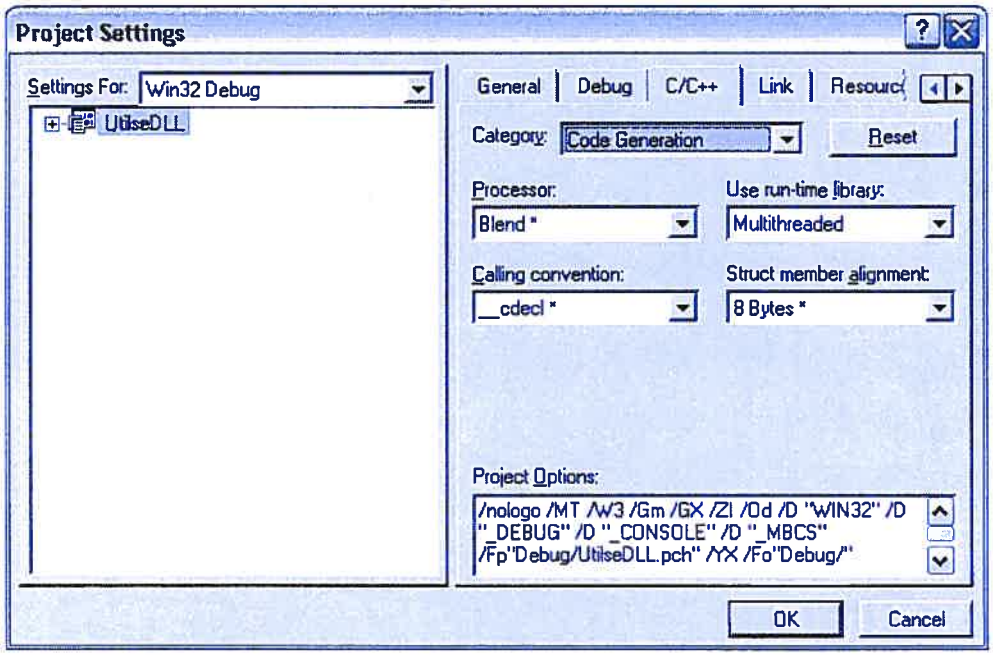
4. Ouvrir la boite de dialogue « Project settings » du menu « Project ». Sélectionner l'onglet « Link ». Sélectionner l'item « Input » du bouton déroulant « Category ».

   Ajouter dans la fenêtre de texte « Object/library » :
   Nafxcwd.lib Libcpmtd.lib Libcmtd.lib Libcmt.lib dllCSerial.lib

   Ajouter dans la fenêtre de texte « Ignore libraries » :
   Nafxcwd.lib Libcpmtd.lib Libcmtd.lib Libcmt.lib



5. Sélectionner l'onglet « C/C++ ». Sélectionner l'item « code generation » du bouton déroulant « Category » . Sélectionner l'item « Multithreaded » du bouton déroulant « Use run-time library »



6. Compiler et lier le projet.

ANNEXE C

Description du prototype du tutoriel de l'EIAH

1. Les variables et les types simples

a) Définition

Qu'est-ce qu'un identificateur?
Un identificateur est le nom d'une entité. Les noms d'une variable, d'une constante et d'une fonction correspondent à des identificateurs.
Un identificateur est une chaîne de caractères composée uniquement de caractères alphanumériques et du caractère de soulignement, débutant obligatoirement par une lettre.
Les lettres minuscules et majuscules sont différenciées. Ainsi TP1, Tp1, tP1, tp1 sont quatre identificateurs distincts.

Qu'est-ce qu'une variable?
Il s'agit d'une donnée située à un emplacement mémoire dont la valeur peut être modifiée
Initialisation

Qu'est-ce qu'une constante?
Il s'agit d'une donnée située à un emplacement mémoire dont la valeur ne peut être modifiée

Qu'est-ce qu'un type?
Le type correspond à la sorte de variable
Le type précise pour la variable:
- l'espace mémoire occupé
- la représentation mémoire; la façon dont elle est mémorisée
- les opérations admissibles

Rôle de chacun?

b) Analogies

Contenu vs Contenant (liquide dans un verre)

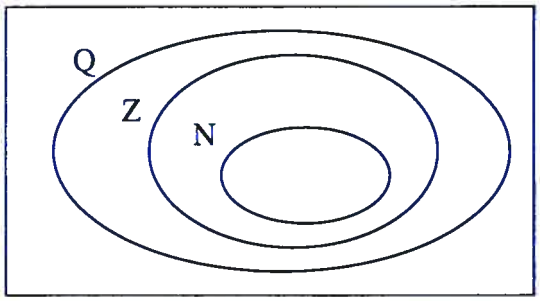
Une variable peut être comparée à ce verre. On peut ajouter et retirer du liquide selon sa bonne volonté.

Un contenant et son contenu. La variable se distingue également selon deux aspects : son emplacement mémoire et sa valeur mémorisée. Le contenu est associé à la valeur de la variable et le contenant à l'emplacement mémoire (physique) de la variable.

Station de télé vs Écran de télé (changement de station à la télé)

Elément vs Ensemble (x elt N, x elt de Z, etc.)

R



c) Exercices

Identification des variables d'un problème et de leur type

<< On peut reprendre ici le même style d'interface que celui des fonctions pour la détermination des paramètres et leur type>>

1. Mémorisation de valeurs
    1. Le nombre de pages d'un livre
    2. L'age du chat de Marie
    3. Le numéro d'un étage d'un édifice
    4. La couleur du cheval blanc de Napoléon
    5. La longueur d'un câble
    6. L'ampérage d'un moteur électrique
    7. Le nom du fabricant d'un ordinateur
    8. Une lettre lue du clavier
    9. L'état d'une lampe (allumée ou éteinte)

Variable	
Identificateur	Type
NoEtage	int

```
void main()
{
   int NbPages;
   int AgeChat;

}
```

2. Identifier les variables d'un programme permettant de :
    1. ordonner alphabétiquement les trois noms de fleurs lus du clavier;
    2. calculer le nombre de secondes comprises dans un certain nombre d'heures lu du clavier;
    3. compter le nombre d'occurrences de la lettre «a» dans une phrase lue du clavier;
    4. ajouter la taxe de vente et la taxe de service au coût d'un produit;
    5. calculer la résistance équivalente à trois résistances reliées en série; la valeur de chacune des trois résistances est lue du clavier.

Variable		Opérateur
Identificateur	Type	

```
void main()
{
   int NbPages;
   int AgeChat;

}
```

Opérateurs et opérations

d) Exemples

Modification du contenu d'une variable par des affectations
Modification du contenu d'une variable par des opérations
Les trois opérations de la permutation
Exemple par type, exemple avec plusieurs types

```
#include <iostream>
using namespace std;

void main()
{
   int Entier1, Entier2, Tampon;

   // L'operateur d'affectation precise pour la variable la valeur a memoriser

   Entier1 = 0;
   Entier1 = 24;
   Entier1 = -65;

   Entier2 = 7;

   // Le terme a droite de l'operateur d'affectation est une expression.
   // Une expression peut etre une constante, une variable ou composee d'operandes
   // lies par des operateurs

   Entier1 = 34 / 5;  // Division entiere
   Entier2 = ( 12 - 5)*24 % 2; // L'evaluation se fait selon l'ordre de
                               // priorite des operateurs

   // La permutation de deux valeurs

   // Manipulation erronee
   Entier1 = 52;
   Entier2 = 38;

   Entier1 = Entier2;
   Entier2 = Entier1;

   // Manipulation correcte
   Entier1 = 52;
   Entier2 = 38;

   Tampon = Entier1;
   Entier1 = Entier2;
   Entier2 = Tampon;


}
```

```
#include <iostream>
using namespace std;

void main()
{
   char Lettre = 'K';
   int Entier1 = 37,
       Entier2 = -82,
       Entier3 = 5;
   bool Booleen = true;
   double Reel_X = 63.73,
          Reel_Y = -3.14e25;
   char Phrase[250] = "Le telephone sonne. Vite, repondez!";

   // Conversion implicite

   Reel_X = Entier1;    // Conversion sans probleme
   Entier2 = Reel_X;    // Conversion possible mais tres dangereuse
   Entier3 = Lettre;    // Conversion sans probleme
   Booleen = Reel_Y;    // Conversion, si Reel_Y=0 donne faux !=0 donne vrai
   Booleen = Lettre;    // Meme raisonnement que precedemment
   Entier1 = Booleen;   // Conversion sans probleme

   // Conversion explicite

  Reel_X = Entier1 / Entier2; // Division entiere puisque deux
                              // operandes entiers
  Reel_Y = Entier1 / double(Entier2);  // Division reelle puisque un operande
                                       // entier et un reele

  /* Instructions erronees */
   //Phrase = " Salut la compagnie ";
   //Phrase = Reel_X;    => et vice-versa
   //Phrase = Entier1;   => et vice-versa
   //Phrase = Booleen;   => et vice-versa
   //Phrase = Lettre;    => et vice-versa

   // Toutes les manipulations sur les chaines de caracteres doivent se
   // faire a l'aide d'une ou des fonctions
   strcpy(Phrase, "Mon pays c'est l'hiver");

}
```

```
#include <iostream>
#include <cstring>
using namespace std;

void main()
{
   char Phrase[120];
   char Ligne[80]= " Le telephone sonne. Vite, repondez!";
   int OrdreAlpha;

   //Affectation d'une valeur a une chaine
   strcpy(Phrase, "Dring! Dring!");
   // Affichage d'une chaine de caracteres
   cout << "Phrase =  " << Phrase << endl;

   // Affichage du nombre de caracteres presents dans la chaine
   cout << "Phrase contient " << strlen(Phrase) << " caracteres." << endl;

   // Concatenation de deux chaines de caracteres
   strcat(Phrase, Ligne);
   cout << "Phrase =  " << Phrase << endl;

   // Comparaison de deux chaines de caracteres
   OrdreAlpha = strcmp(Ligne, Phrase);
   cout << "OrdreAlpha =  " << OrdreAlpha << endl;
   OrdreAlpha = strcmp(Phrase, Phrase);
   cout << "OrdreAlpha =  " << OrdreAlpha << endl;
   OrdreAlpha = strcmp(Phrase,"Drelin! Drelin!");
   cout << "OrdreAlpha =  " << OrdreAlpha << endl;

   //Modification du 3 ieme caractere de la chaine Phrase
   strcpy(Phrase, "Dring! Dring!");   Phrase[2] = 'o';
   cout << "Phrase =  " << Phrase << endl;
}
```

2. Entrées et Sorties

a) Définition

Opérations de lectures

- Les entrées/sorties s'appuient sur des flots(stream).
- Un flot est un flux de données abstrait qui part d'une source et qui va vers une cible.
- Source et cible peuvent être des fichiers usuels, des périphériques ou des emplacement mémoires.
- L'objet *cout* représente le canal de sortie standard
- << est l'opérateur de transmission pour envoyer des données vers un flot
- L'objet *cin* représente le canal d'entrée standard
- >> est l'opérateur de réception qui lit les données d'un flot

Opérations d'affichage

Règles régissant

la lecture

- ❑Les séparateurs permettant de distinguer la valeur à lire sont: l'espace, le tabulateur, la fin de ligne et la fin de fichier.
- ❑Les séparateurs précédant la valeur à lire sont ignorés (ou sautés).
- ❑La lecture s'effectue jusqu'à la rencontre du premier séparateur suivant la valeur à lire.

l'affichage

- ❑L'affichage s'effectue à l'écran de la position courante du curseur qui est initialement situé au coin supérieur gauche.
- ❑Les espaces et les sauts de ligne doivent être ajoutés explicitement dans l'énoncé d'affichage. La lecture s'effectue jusqu'à la rencontre du premier séparateur suivant la valeur à lire.

Opérateurs et fonctions

Fonctions d'entrée

```
istream& get(char *Tampon,int Longueur,char Fin='\n')
```

Lit une suite de caractères dans un flot et la mémorise dans le paramètre Tampon. La lecture se poursuit jusqu'à ce que:

–(Longueur - 1) caractères ont été lus ;
–EOF a été lu ;
–le caractère de fin (Fin) a été lu; il s'agit par défaut du caractère <ENTER> (représenté ici par '\n') ; ce caractère n'est pas pris en compte par la lecture. i.e. que la prochaine lecture débute à partir de ce caractère.

La lecture au clavier nécessite l'utilisation du caractère <Enter> pour exécuter la lecture.

EXEMPLE

```
//Deux appels sont possibles :
          char Phrase1[80], Phrase2[120];
          cout << " Inscrire une premiere phrase : ";
          cin.get(Phrase1, 80); // Lit 79 caractères ou arrête
                                // à la rencontre du ENTER
          cout << " Inscrire une premiere phrase : ";
          cin.get(Phrase2, 120, 's'); // Lit 119 caractères ou
                                // arrête à la rencontre d'un s
```

```
istream& getline(char *Tampon,int Longueur,char Fin='\n')
```

Lit une suite de caractères dans un flot et la mémorise dans le paramètre Tampon. La lecture se poursuit jusqu'à ce que:

–(Longueur - 1) caractères ont été lus ;
–EOF a été lu ;
–le caractère de fin (Fin) a été lu; il s'agit par défaut du caractère <ENTER> ; ce caractère est pris en compte par la lecture, i.e. que la prochaine lecture débute après ce caractère.

La lecture au clavier nécessite l'utilisation du caractère <Enter> pour exécuter la lecture.

EXEMPLE

```
//Deux appels sont possibles:
          char Phrase1[80], Phrase2[120];
          cout << " Inscrire une première phrase : ";
          cin.getline(Phrase1, 80); // Lit 79 caractères ou
                                // arrête à la rencontre du ENTER
```

```
cout << " Inscrire une première phrase : ";
cin.getline(Phrase2, 120, 's'); // Lit 119 caractères
                        // ou arrête à la rencontre d'un s
```

```
istream& get(char& CarLu)
```

La fonction lit un caractère depuis un flot et le range dans la variable CarLu. La fonction permet de lire n'importe quel caractère, ainsi <espace>, <tab>, etc. peuvent être lus. La lecture au clavier nécessite l'utilisation du caractère <Enter> pour exécuter la lecture.

EXEMPLE

```
//Deux appels sont possibles :
char Car;
        cout << "Inscrire un caractère : ";
cin.get(Car);  // Permet de lire le caractère entré
Car = cin.get(); // Permet de lire le <ENTER>
```

```
istream& ignore(int Nbre = 1, int Fin = EOF);
```

Saute une suite de caractères du flot d'entrée jusqu'à ce que :

–Nbre caractères ont été sautés ;

–le caractère Fin est rencontré; lorsqu'il est rencontré le caractère Fin est sauté également.

EXEMPLE

```
cin.ignore(120) ; // Saute 120 caractères ou arrête à la rencontre de EOF
cin.ignore(25, 's') ; // Saute 25 caractères ou arrête le saut à la rencontre du caractère s
```

```
void clear(int n = 0);
```

Affecte à l'état d'erreur du flot d'entrée la valeur précisée (par défaut 0).

```
cin.clear();
```

Cette fonction est utilisée lorsqu'une erreur de lecture est commise puisque toute instruction de lecture subséquente à une erreur de lecture est ignorée par le flot d'entrée concerné.

Fonctions de sortie

```
ostream& put(char Caractere)
```

Le caractère reçu en paramètre est ajouté au flot de sortie.

EXEMPLE

```
char Car = 'R';
cout << "Le caractère est : ";
cout.put(Car) ; // affichage du caractère R
```

```
setf(ios::mode) ou setiosflags(ios ::mode)
```

Permet au flot d'utiliser de nouvelles spécifications reçues en paramètre en précisant un ou des modes d'affichage. Ce mode demeure actif tant qu'un nouveau mode n'est pas précisé.

Mode :

fixed	Mode d'affichage en point flottant ex:-12.345	cout.setf(ios::fixed); ou cout<< setiosflags(ios ::fixed) ;
scientific	Mode d 'affichage en notation scientifique ex:1.234500876+E03	cout.setf(ios::scientific); ou cout<< setiosflags(ios ::scientific) ;
showpoint	Assure l'affichage du point en notation point flottant et ajoute des zéros s'il y a lieu.	cout.setf(ios::showpoint); ou cout<< setiosflags(ios ::showpoint) ;

Il est possible de préciser plusieurs modes en utilisant l'opérateur binaire OU soit |

Par exemple,

```
cout.setf(ios::fixed|ios::showpoint);
```

Réalise un affichage en notation point flottant en assurant l'affichage du point et des zéros.

```
setw(Nb_col)
```
[nécessite l'inclusion du fichier <iomanip>]

Précise le nombre de colonnes à utiliser pour afficher une valeur. La justification est à droite, i.e. que des colonnes vides précéderont, s'il y a lieu, la valeur à afficher. Actif uniquement pour la prochaine donnée affichée.

EXEMPLE

```
cout<<setw(5)<<12 ; // Affiche le nombre 12 sur 5 colonnes.
```

`setprecision(Nb_dec)` [nécessite l'inclusion du fichier <iomanip>]

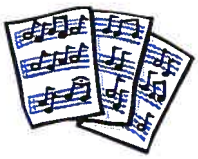
Précise le nombre de décimales à afficher pour une donnée réelle. Demeure actif jusqu'au prochain appel à setprecision().

EXEMPLE

```
cout.setf(ios::fixed);
cout<< setprecision(2);       // Colonne:12345678
cout << setw(8) << 12.57489; // Affiche:   12.57
cout << setw(8) << -45.2478; // Affiche:  -45.25
```

b) Analogies

Lecture d'une partition



Le musicien doit lire chaque note l'une à la suite de l'autre. Autrement, la mélodie risque d'être cacophonique.

Lecture de l'adresse sur une enveloppe

L'adresse inscrite sur une enveloppe contient des lettres et des chiffres. Tout naturellement nous lisons séquentiellement les mots et les nombres. Il faudra prévoir dans notre programme la distinction de lecture de chaînes de caractères ou de nombre.

Identification d'un terme

Tout naturellement, lors de la lecture d'un texte nous associons la signification d'un terme selon le contexte.

SI (Français et italien ou espagnol)
IF (anglais et français)
SON (français plusieurs sens et anglais):

- Adj. Poss. : Qui appartient, est relatif à la personne ou la chose dont il est question.

- Alimentation: Partie périphérique des céréales, qui est séparée de la farine par la mouture dans les blutoirs ou dans les tamis représentant en général 20 à 25 % des céréales elles-mêmes.
- Physique : Sensation produite dans l'appareil auditif par une vibration acoustique
- Anglais : fils

IL (français et nombre romain 49)

- Pronom personnel de la troisième personne du singulier.
- Nombre romain 49

Pour un programme, la lecture d'une donnée exige de préciser son type. En d'autres mots, il faut préalablement déclarer une variable qui servira à lire la donnée. Par exemple, le nombre 345 peut être lu comme un entier ou une chaîne de trois caractères.

Rappelons que la méthode de mémorisation de la donnée et les opérations possibles sur cette donnée sont précisées par le type de la variable.

Les différents formats de date

Il existe plusieurs conventions d'écriture d'une date sur des documents.

Canada :	AAAA-MM-JJ
États-Unis:	MM/JJ/AAAA
France :	JJ.MM.AAAA
Royaume-Uni :	JJ/MM/AAAA

Sainte-Misère, le 7 mai 2020

Madame Sophie Labelle
Labelle & Associés inc.
1234, rue des Beautés
Beaugrand (Québec) G1B 1A1

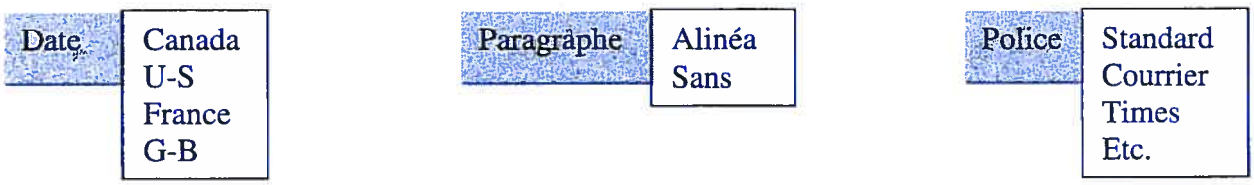
Madame,

Pour faire suite à l'offre d'emploi publiée dans Le Flatteur de Beaugrand du 15 avril dernier, je désire poser ma candidature au poste d'ingénieure à pourvoir dans votre entreprise.

Je vous remets ci-joint mon curriculum vitae. Vous pourrez ainsi constater que je détiens un baccalauréat en ingénierie et je possède une expérience pertinente dans le domaine ciblé. De plus, je suis responsable, minutieuse et capable de m'adapter rapidement.

Je vous remercie, à l'avance, de l'attention que vous accorderez à ma demande et vous prie d'accepter, Madame, mes salutations distinguées.

Rafaella Diablo-Tremblay
9, boulevard Pied-à-Terre
Ferland (Québec) G5C 5A5
Téléphone : (514) 123-4567



c) Exercices

Préciser la déclaration et l'instruction de lecture

Nous désirons lire :

- La température du jour en degré Celsius
- La hauteur maximale pouvant atteindre un projectile
- Le titre d'un livre de science-fiction
- Le numéro de version d'un logiciel
- La lettre associée à une rangée de sièges dans une salle de spectacle
- Le nom du père du gendre de l'oncle Hubert de ma mère
- La largeur et l'épaisseur d'une planche de bois
- Le prénom, le nom et l'âge d'une personne
- L'espace offert par la mémoire principale et par le disque rigide d'un ordinateur
- Le nom et la couleur d'une pièce du jeu d'échec

Donner le contenu à lire en fonction de l'instruction de lecture

```
char Lettre;
char Phrase[120], Prenom[80], Nom[80];
int  Age, NbLivres,
double Longueur, Largeur;
```

- cin >> Age >> Lettre;
- cin >> Nom >> Prenom;
- cin >> Phrase >> Longueur >> Largeur;
- cin.getline(Phrase,120);
- cin.get(lettre);
- cin >> Prenom >> Age >> Lettre >> Longueur >> Largeur;

Uniquement ce type d'exercices

Pour le prochain type d'exercices il faudrait utiliser une interface qui oriente l'étudiant vers la séquence :

Déclaration de variables
Message de sollicitation
Lecture
<Affichage du contenu des variables lues>
avec format d'affichage

- Déterminer la plus grande lettre selon l'ordre alphabétique de trois lettres lues du clavier.
- Déterminer le volume d'une boite dont la hauteur, la longueur et la largeur sont lues du clavier.

- Calculer l'âge de l'individu dont la date de naissance est lue du clavier. Le jour, le mois et l'année sont précisés à l'aide de trois nombres entiers.
- Calculer la force résultante du déplacement d'un chariot dont la masse et l'accélération sont lus du clavier.
- Déterminer le nombre d'occurrences d'une lettre dans une phrase. La lettre et la phrase sont lues du clavier.

d) Exemples

Montrer le contenu d'une variable en fonction d'instructions de lecture
Montrer l'affichage obtenu à l'aide de format utilisé.

```
#include <iostream>
#include <cstring>
using namespace std;

void main()
{
   char CarLu;
   char Mot[50];
   int Entier;
   double Reel;

   cout << "Entrez un caractere =  " ;
   cin >> CarLu;
   cout << "Le caractere lu est: " << CarLu << endl;


   cout << "Entrez un mot =  " ;
   cin >> Mot;
   cout << "Le mot lu est: " << Mot << endl;

   cout << "Entrez un nombre entier =  " ;
   cin >> Entier;
   cout << "Le nombre entier lu est: " << Entier << endl;

   cout << "Entrez un nombre reel =  " ;
   cin >> Reel;
   cout << "Le nombre reel lu est: " << Reel << endl;
}
```

```
#include <iostream>
#include <cstring>
using namespace std;

void main()
{
   char CarLu;
   char Phrase[120];
   int Entier;
   double Reel;

   cout << "Entrez un caractere =  " ;
   cin.get(CarLu);
   cout << "Le caractere lu est: ";
   cout.put(CarLu);
   cin.get(CarLu);   // Lecture du caractere ENTER toujours
                     // present dans le tampon
   cout << endl << "Le caractere lu est: ";
   cout << int(CarLu) << endl;

   cout << "Entrez une phrase =  " ;
   cin.get(Phrase,120);
   cout << "La phrase lu est: " << Phrase << endl;

   cin.ignore(); // Pour sauter le ENTER toujours present dans le tampon

   cout << "Entrez une phrase =  " ;
   cin.getline(Phrase,120);
   cout << "La phrase lu est: " << Phrase << endl;
}
```

```
#include <iostream>
#include <iomanip>
using namespace std;

void main()
{
   char CarLu = 'A';
   char Mot[50] = "strategie";
   int Entier = 763;
   double Reel = -754.236529;

   cout << "         1         2         3" << endl;
   cout << "12345678901234567890123456789O" << endl;

   cout << setw(5) << CarLu << endl;

   cout << setw(20) << Mot << endl;

   cout << setw(10) << Entier << endl;

   cout.setf(ios::fixed);
   cout << setprecision(2);
   cout << setw(15) << Reel << endl;

   cout.unsetf(ios::fixed);
   cout.setf(ios::scientific);
   cout << setprecision(5);
   cout << setw(30) << Reel << endl;
}
```

3. Fichier texte
   a. Définition

Qu'est-ce qu'un fichier?

✓Un fichier est une structure homogène (données de même nature) servant à mémoriser de l'information.
✓Les fichiers textes sont des fichiers séquentiels, pour accéder à une information il faut lire toutes les données précédentes.
Les opérations de lecture et d'écriture sont exclusives, on peut lire OU écrire et non les deux à la fois.
✓Pour utiliser un fichier texte on inclut préalablement le fichier d'en-tête *fstream.h.*
✓Les opérateurs et fonctions de lecture ou écriture (clavier ou écran) s'appliquent intégralement à un fichier texte.

Utilité

L'utilisation d'un fichier permet d'accéder à des données nécessaires à l'exécution d'un programme sans devoir les demander au clavier. Il permet également de conserver les résultats de l'exécution d'un programme afin de pouvoir les manipuler ultérieurement au besoin.

Spécification

Lecteur : chemin d'accès \ Nom Fichier. Ext

Structure d'un fichier (fin de ligne, fin de fichier)

Les modes d'ouverture

MODE	Signification
`ios::app`	Ouvrir un fichier texte pour écriture uniquement à partir de la fin du fichier.
`ios::ate`	Ouvrir un fichier texte (lecture ou écriture) et se positionner à la fin du fichier.
`ios::nocreate`	Ouvrir un fichier texte seulement si le fichier existe.
`ios::noreplace`	Ouvrir un fichier texte uniquement si le fichier n'existe pas, à moins que l'option *ios::ate* ou *ios::app* sont utilisés conjointement.

Les fonctions propres à la manipulation de fichier

Ouverture open()

L'ouverture correspond à assigner à la variable fichier un fichier physique. Pour ce faire, il suffit de préciser la spécification complète du fichier, tel que reconnu par le système d'exploitation, en argument à la fonction open().

De plus, la fonction open() permet de préciser les manipulations possibles dans le fichier ouvert. Nous parlons communément de mode d'ouverture du fichier.

Par exemple,

```
ifstream FichierLu;
FichierLu.open("C:\\Cours\\Devoir3.txt");
```

Entraîne que toutes manipulations réalisées sur la variable FichierLu s'effectueront dans le fichier Devoir3.txt du répertoire Cours du disque C :.

Lecture et écriture dans le fichier

La lecture est réalisée **uniquement** dans les fichiers de type ifstream (input file stream).

Toutes les fonctions décrites pour la lecture au clavier sont utilisables pour la lecture dans un fichier en remplaçant cin par l'identificateur de la variable fichier de type ifstream. Leur comportement est tout a fait identique à la différence que l'instruction est exécutée sans l'attente du ENTER au clavier. C'est logique! On ne lit pas du clavier.

Exemple,

```
ifstream FichierLu;
int Entier;
FichierLu.open("C:\\Cours\\Devoir3.txt");

Fichier >> Entier;  // Lecture d'un entier dans le fichier
```

Toutes les fonctions décrites pour l'affichage à l'écran sont utilisables pour l'écriture dans un fichier en remplaçant cout par l'identificateur de la variable fichier de type ofstream. Leur comportement est tout a fait identique. Dans cet esprit, il ne faut pas oublier d'ajouter les espaces et les sauts de ligne, autrement toutes les données seront « collées ».

Exemple,

```
ofstream FichierEcrit;
int Entier = 100;
FichierEcrit.open("C:\\Cours\\Devoir5.txt");

FichierEcrit << Entier << endl;  // Écriture de 100 suivi d'un saut de ligne
FichierEcrit << 'K';             // Écriture de K dans le fichier
```

Fermeture de fichier close()

La fonction close() permet de fermer proprement un fichier. Cette fonction est essentielle lors de l'écriture dans un fichier afin de s'assurer que tout le contenu à écrire soit bien transférer dans celui-ci. La fonction close() est préfixée de l'identificateur du fichier concerné.

Par exemple,

```
FichieEcrit.close();
```

Fin de fichier

À la mémorisation d'un fichier certains caractères « spéciaux » sont ajoutés, par exemple les deux caractères spécifiant un saut de ligne. Un cas semblable est la fin de fichier qui est marquée par le caractère spécial « CTRL-Z ». Dans certaines situations et surtout pour éviter de lire outre la fin du fichier on doit savoir que le caractère lu est la fin de fichier. C'est le rôle de la fonction booléenne eof() qui est préfixé de l'identificateur du fichier concerné. La fonction retourne la valeur true si la fin du fichier est atteinte ou false autrement.

Par exemple,

```
bool  Fini;
 Fini = FichierLu.eof()
```

Détection d'erreur

La fonction booléenne fail() est utilisée pour vérifier si une erreur est commise sur une variable fichier. Plus précisément, la fonction fail() retourne true si la variable fichier n'est plus utilisable puisque une erreur a été commise ou false autrement.

Nous utiliserons cette fonction principalement pour vérifier si l'ouverture du fichier c'est faite correctement. En effet, il arrive régulièrement qu'on associe un nom de fichier inexistant à une variable fichier de lecture.

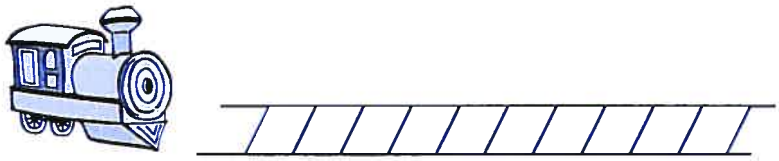
Par exemple,

```
ifstream FicLu;
bool Correct;

FicLu.open("C:\Cours\Devoir1.txt") ; // Nom erroné puisqu'un seul oblique
Correct = FicLu.fail();  // Correct vaut true
```

b. Analogies

Le train

[Lecture] Le train se déplace d'une ville à l'autre en empruntant le rail. Il ne peut sauter ou voler vers une ville.



[Écriture] La construction d'un tronçon de rail se fait séquentiellement. On dépose les rails les uns à la suite des autres.

L'affichage à l'écran

L'affichage à l'écran est le résultat d'un fichier texte.
Taper du texte au clavier et le faire afficher à l'écran. Ajouter le bouton : Recommencer qui efface l'écran et recommence.

Enregistrement et lecture sur ruban (video)

Offrir une interface classique avec les boutons : Play et Record. Le déroulement est perçu à l'aide de l'affichage numérique du temps ou de la longueur du ruban.

c. Exercices

La lecture d'un fichier est similaire à la lecture du clavier. Il s'agit des mêmes exercices qu'au module précédent sauf que la lecture est faite dans un fichier.

Pour le prochain type d'exercices il faudrait utiliser une interface qui oriente l'étudiant vers la séquence :

Déclaration de variables
Ouverture du fichier
Message de sollicitation
Lecture
<Affichage du contenu des variables lues>
avec format d'affichage
Fermeture du fichier

- Déterminer la plus grande lettre selon l'ordre alphabétique de trois lettres lues du fichier Lettres.txt.
- Déterminer le volume d'une boite dont la hauteur, la longueur et la largeur sont lues du fichier Dimension.txt.
- Calculer l'âge de l'individu dont la date de naissance est lue du fichier Date.txt. Le jour, le mois et l'année sont précisés respectivement à l'aide d'un nombre.
- Calculer la force résultante du déplacement d'un chariot dont la masse et l'accélération sont lus du fichier Chariot.txt.
- Déterminer le nombre d'occurrences d'une lettre dans une phrase. La lettre et la phrase sont lues du fichier Texte.txt.

Déclaration de variables
Ouverture du fichier
Message de sollicitation
Lecture
Écriture du contenu des variables lues dans le fichier
Fermeture du fichier

- Écrire dans le fichier Prenoms.txt les prénoms de deux personnes lus du clavier.
- Écrire dans le fichier Quilles.txt le meilleur pointage pour une partie de quilles de votre cousin Toutun Abas que vous devez lire du clavier.
- Calculer l'équivalent en dollar américain d'un montant en dollar canadien pour un taux d'échange donné. Inscrire dans le fichier CanAme.txt le montant canadien et le montant américain. Le montant en dollar canadien et le taux d'échange sont lus du clavier.
- Calculer la puissance d'un moteur électrique selon un voltage et une intensité électrique donnés lus du clavier. Le numéro du modèle du moteur (chiffres+lettres) lu du clavier et sa puissance sont transcrits dans le fichier Moteur.txt.
- Déterminer la couleur obtenue du mélange de deux couleurs primaires lues du clavier. La couleur obtenue et les deux couleurs primaires sont inscrites dans le fichier Couleur.txt à raison d'une couleur par ligne.

Exercices sélectionnés

- Calculer l'âge de l'individu dont la date de naissance est lue du fichier Date.txt. Le jour, le mois et l'année sont précisés respectivement à l'aide d'un nombre.
  - Instructions d'affichage et de lecture au clavier
  - AUCUNE
- Écrire dans le fichier Quilles.txt le meilleur pointage pour une partie de quilles de votre cousin Toutun Abas que vous devez lire du clavier.
  - Instructions d'affichage et de lecture au clavier
  - cout << "Meilleur Pointage : ";

- Calculer la puissance d'un moteur électrique selon un voltage et une intensité électrique donnés lus du clavier. Le numéro du modèle du moteur (chiffres+lettres) lu du clavier et sa puissance sont transcrits dans le fichier Moteur.txt.
  - Instructions d'affichage et de lecture au clavier
  - cout << "Numéro du modèle : ";
  - cin >> NoModele;
  - cout << "Intensité : ";
  - cin >> Intensite;
  - cout << "Voltage : ";
  - cin >> Voltage;
- Déterminer la couleur obtenue du mélange de deux couleurs primaires lues du clavier. La couleur obtenue et les deux couleurs primaires sont inscrites dans le fichier Couleur.txt à raison d'une couleur par ligne.
  - Instructions d'affichage et de lecture au clavier
  - cout << "Première couleur : ";
  - cin >> Premiere_Couleur;

- cout << "Deuxième couleur : ";
- cin >> Deuxieme_Couleur;

➢ Calculer la force résultante du déplacement d'un chariot dont la masse et l'accélération sont lus du fichier Chariot.txt.
- Instructions d'affichage et de lecture au clavier
- AUCUNE

d. Exemples

Montrer le contenu d'un fichier et la répercussion des instructions de lecture et d'écriture sur celui-ci.
Montrer le déplacement du « pointeur de lecture/écriture » dans le fichier.

```
#include <fstream>
#include <iostream>
using namespace std;

void main()
{
   int Entier;
   double Reel;
   char Lettre;
   char Mot[25];
   ifstream Fic_Entree;

   // Ouverture du fichier
   Fic_Entree.open("Essai-A.txt");

  // Lecture d'une lettre
   Fic_Entree >> Lettre;
   cout << "Lettre= " << Lettre << endl;
   // Lecture d'un entier
   Fic_Entree >> Entier;
   cout <<  "Entier= " << Entier << endl;
   // Lecture d'un reel
   Fic_Entree >> Reel;
   cout << "Reel = " << Reel << endl;
   // Lecture d'une chaine de caractères
   Fic_Entree >> Mot;
   cout << "Mot = " << Mot;

   Fic_Entree.close();
}
```

Fichier Essa-A.txt

```
K

12  -5.1
Logique
```

```
#include <fstream>
#include <iostream>
using namespace std;

void main()
{
   int Entier;
   double Reel;
   char Lettre;
   char Mot[25];
   ifstream Fic_Entree;
   ofstream Fic_Sortie;

   // Ouverture du fichier
   Fic_Entree.open("Essai-B.txt");
   Fic_Sortie.open("Essai-B2.txt");


  // Lecture d'une lettre
   Fic_Entree >> Lettre;
   Fic_Sortie << Lettre << endl;
   // Lecture d'un entier et d'un reel
   Fic_Entree >> Entier >> Reel;
   Fic_Sortie << Entier << ' ' << Reel << endl;
   // Lecture d'une chaine de caractère
   Fic_Entree >> Mot;
   Fic_Sortie << Mot;

   Fic_Entree.close();
   Fic_Sortie.close();

}
```

Fichier Essai-B.txt

```
H

12    -5.1
Logique
```

```
#include <fstream>
#include <iostream>
#include <iomanip>
using namespace std;

void main()
{
   int Entier;
   double Reel;
   char Lettre;
   char Phrase[25];
   ifstream Fic_Entree;
   ofstream Fic_Sortie;

   // Ouverture du fichier
   Fic_Entree.open("Essai-C.txt");
   Fic_Sortie.open("Essai-C2.txt");

   Fic_Sortie.setf(ios::fixed);

  // Lecture d'une lettre
   Fic_Entree.get(Lettre);
   Fic_Sortie.put(Lettre);
   Fic_Sortie << endl;
   // Lecture d'un entier et d'un reel
   Fic_Entree >> Entier >> Reel;
   Fic_Sortie << Entier << ' ' << setprecision(2) << Reel << endl;
   Fic_Entree.ignore(); // Pour enlever le ENTER
   // Lecture d'une chaine de caractère
   Fic_Entree.getline(Phrase,25);
   Fic_Sortie << Phrase;

   Fic_Entree.close();
   Fic_Sortie.close();
```

Fichier Essai-C.txt

```
v

78   -35.15687
La mer est calme.
```

4. Structure de décision

a. Définition

Les structures de décision permettent de contrôler l'exécution du programme en associant un test ou expressions booléenne à une instruction. L'instruction est exécutée si le test est vrai, autrement l'instruction n'est pas exécutée. L'instruction peut être simple ou composée.

Une Instruction simple correspond à un seul énoncé au sens du langage.
Par exemple,
Entier = 10;

L'instruction composée est une suite d'instructions débutant par une accolade ouvrante et se terminant par une accolade fermante.

Exemple,

```
{
    Entier = 10;
    Reel = 12.5;
    Strcpy(Mot, " Bonbon ");
}
```

Logique : table de vérité

Propositions		**Opérations**			
P	**Q**	**P et Q**	**P ou Q**	**P ou bien Q**	**Non P**
Vrai	Vrai	Vrai	Vrai	Faux	Faux
Vrai	Faux	Faux	Vrai	Vrai	Faux
Faux	Vrai	Faux	Vrai	Vrai	Vrai
Faux	Faux	Faux	Faux	Faux	Vrai

Trois structures de décision

L'instruction doit être exécutée si l'expression booléenne est vraie, autrement il n'y a rien à faire.

```
if (expression_booléenne)
   instruction;
```

Exemple :

```
int Entier ;
cout << " Entrer un entier : " ;
```

```
cin >> Entier ;
if (Entier <0)
   Entier = -1*Entier ;
cout << " Sa valeur absolu est "  << Entier ;
```

Une instruction est exécutée si l'expression booléenne est vraie. Une autre instruction est exécutée si l'expression booléenne est fausse.

```
if (expression_booléenne)
   instruction_if;
else
   instruction_else;
```

Exemple:

```
int AgeJean;
cout << " Age de Jean :  " ;
cin >> AgeJean ;
if (AgeJean >=18)
   cout << "Jean est un adulte";
else
   cout << "Jean n'est pas un adulte";
```

Selon la valeur d'une expression, une instruction doit être exécutée.

```
switch (expression)
{
   case constante_1 : instruction_1;
                      break;
   case constante_2 :
   case constante_3 : instruction_2_3;
                      break;
          ...
   case constante_x : instruction_x;
                      break;
           ...
   default          : instruction;
}
```

Dans ce cas,
si la valeur de Expression est égale à constante_1, Instruction_1 est exécutée
Si la valeur de Expression est égale à constante2 ou constante_3, instruction_2_3 est exécutée. Etc.

Si la valeur de Expression n'est égale à aucune des constantes précédentes, l'instruction default est exécutée.

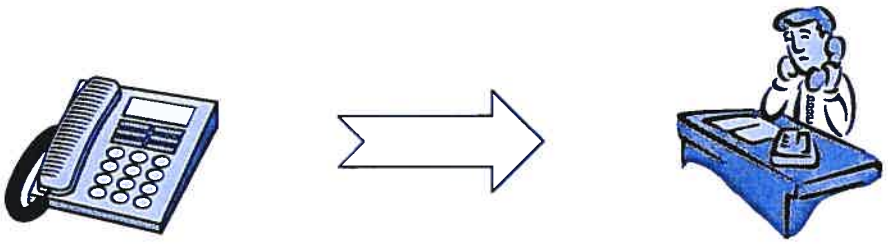
Exemple,

```
char Lettre;

cout << "Afficher un mot debutant par la lettre: ";
cin >> Lettre;

switch (toupper(Lettre)) // Conversion en majuscule
{
   case 'A': cout << "Armoire";
             break;
   case 'B': cout << "Botanique";
             break;
   case 'F': cout << "Fable";
             break;
   case 'S': cout << "Systeme";
             break;
   default : cout << "Desole! Aucun mot prevu pour cette lettre.";
}
```

b. Analogie

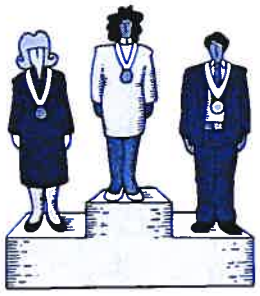
< Il faut penser interaction pour rendre cela attrayant >

[SI]
Si le téléphone sonne (je réponds)



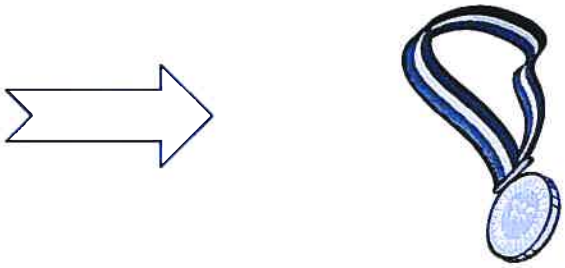
[SI-SINON]
Si je réussis mon cours (passe au suivant) sinon (reprise)

[SI + SÉLECTION]

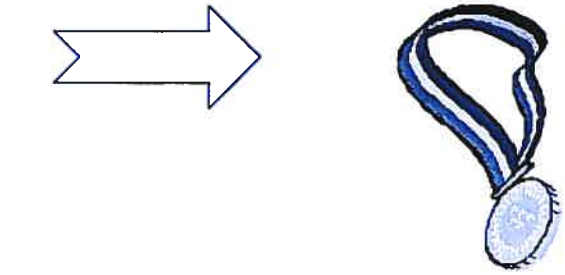


Si je suis parmi les trois premiers de la course (remise des médailles)

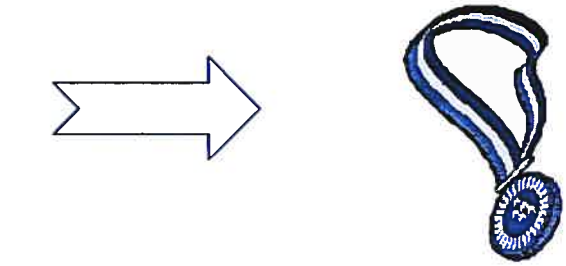
Si premier (Or)



Si deuxième (Argent)



Si troisième (Bronze)



[Composition d'expression booléenne]

<ET>
je possède 200$ ET la bicyclette vaut 150$ (j'achète)

<OU>
Il reste quatre place dans le voilier
Pierre embarque OU Jean embarque OU Jacques embarque

Note : couramment en français la conjonction OU est utilisée pour marquer l'alternative; ce qui correspond en logique au OU EXCLUSIF.

<OU EXCLUSIF>
Vous gagnez à un tirage
Vous devez choisir entre le montant d'argent OU le voyage

c. Exercices

Liste d'exercices résolus en suivant les étapes :

Composition du test
Écriture des instructions
Écriture de l'alternative (s'il y a lieu)

- Le prix d'un billet de spectacle est de 20 Douleurs pour un adulte (18 ans et plus) ou de 10 Douleurs pour un enfant. Afficher le prix d'un billet selon l'âge de la personne.
- Afficher si un nombre entier est pair ou impair.
- Le système d'air climatisé est en fonction uniquement lorsque la température ambiante est supérieure à 23 degrés Celsius. Afficher l'état de la thermopompe (marche-arrêt) selon la température ambiante.
- Un ascenseur ne peut pas transporter plus de cinq personnes à la fois. Afficher que l'ascenseur ne peut se déplacer s'il y a plus de cinq personnes.
- On distingue plusieurs types de bicyclettes: la bicyclette de tourisme, la bicyclette de ville, le tandem, la bicyclette de compétition, la bicyclette de montagne, la bicyclette de trial et le vélo cross, ces trois derniers types étant considérés comme des bicyclettes tout terrain. Afficher si la bicyclette d'une personne est une bicyclette tout terrain.
- Un programme présente un menu comportant quatre choix. 1- Lire la donnée; 2- Afficher la donnée; 3- Modifier la donnée; 4- Terminer le programme. Afficher l'opération correspondante au nombre choisi par l'usager.
- Afficher la nature d'une lettre, i.e. une voyelle ou une consonne.
- Un camp de vacances forme des groupes de jeunes selon leur âge : Grenouilles: 6-7 ans; Castors: 8-9 ans; Belettes: 10-11 ans; Loups : 12-13 ans; Coyotes: 14-16 ans. Afficher le nom du groupe d'un enfant selon son âge.

L'instruction finale est construite et affichée.

d. Exemple

Démontrer le fonctionnement (contrôle) de la structure
Exemple sans imbrication
Exemple avec imbrication
Démontrer l'importance du break

```
#include <iostream>
#include <cctype>
using namespace std;

void main()
{
   int Entier, NbRoues;
   double ReelA, ReelB;
   char Caractere;

  //Valeur absolue d'un entier
   cout << "Entrez un entier: ";
   cin >> Entier;
   cout << "La valeur absolue de " << Entier << " est ";
   if (Entier < 0)
     Entier = -1*Entier;
   cout << Entier << endl;

   //Deux reels ordonnees
   cout << "Entrez deux nombres réels: ";
   cin >> ReelA >> ReelB;
   cout << "En ordre croissant les deux nombres sont " ;
   if (ReelA < ReelB)
      cout << ReelA << ' ' << ReelB << endl;
   else
      cout  << ReelB << ' ' << ReelA << endl;

   //Le nom d'un vehicule
   cout << "Precisez le nombre de roues du vehicule (1 a 4): ";
   cin >> NbRoues;
   cout << "Il s'agit d'un(e) ";
   switch (NbRoues)
   {
      case 1: cout << "unicycle." << endl; break;
      case 2: cout << "bicycle." << endl; break;
      case 3: cout << "tricycle." << endl; break;
      case 4: cout << "automobile." << endl;
      default:
         cout << "engin.";
   }
}
```

```
#include <iostream>
#include <cctype>
using namespace std;

void main()
{
   char Caractere;

  // Lecture d'un caractere
   cout << "Entrer un caractere: ";
   cin >> Caractere;

   // Verifier la nature du caractere
   if (isalnum(Caractere))
   {
      if (isdigit(Caractere))
         cout << Caractere << " est un chiffre.";
      else
      {
         cout << Caractere << " est une lettre";
         if (isupper(Caractere))
            cout << " majuscule";
         else
            cout << " minuscule";
         Caractere = toupper(Caractere);
         if (Caractere =='A' || Caractere =='E' || Caractere=='I' ||
             Caractere =='O' || Caractere =='U' || Caractere=='Y')
            cout << " et une voyelle. ";
         else
            cout << " et une consonne.";
      }
   }
   else
     cout << Caractere << " n'est ni un chiffe ni une lettre.";
}
```

```
#include <iostream>  // Pour l'utilisation de cin et cout
#include <cstring>   // Pour l'utilisation de strcmp(), strcpy(),
                     // strupr() et strlwr()
using namespace std;

void main()
{
   char Mois[10];

   // Lecture d'un caractere
   cout << "Entrer le nom d'un mois: ";
   cin.getline(Mois,10);
   strcpy(Mois, strupr(Mois));   // Conversion en majuscules

   // Afficher une information sur le mois
   switch (Mois[0])
   {
      case 'J' : if (Mois[1] == 'U')
                   if (strcmp(Mois, "JUIN")==0)
                      cout << " juin est le sixième mois";
                   else
                      cout << " juillet est le septième mois";
                 else
                    cout << " janvier est le premier mois";
                 break;
      case 'M' : if (strcmp(Mois, "MARS")==0)
                    cout << " mars est le troisième mois.";
                 else
                    cout << " mai est le cinquième mois.";
                 break;
      case 'A' : if (strcmp(Mois, "AVRIL")==0)
                    cout << " avril est le quatrième mois de l'année.";
                 else
                    cout << " août est le huitième mois de l'année.";
                 break;
      case 'F' : if (strcmp(Mois,"FEVRIER")==0)
                    cout << " février est le deuxième mois de l'année.";
                 break;
      case 'S' :
      case 'O' :
      case 'N' :
      case 'D' : strcpy(Mois, strlwr(Mois));
                 cout << Mois << " est un mois appartenant à l'automne.";
                 break;
      default  : cout << " il ne s'agit pas d'un mois!";
   }
}
```

5. Structure de répétition

a. Définition

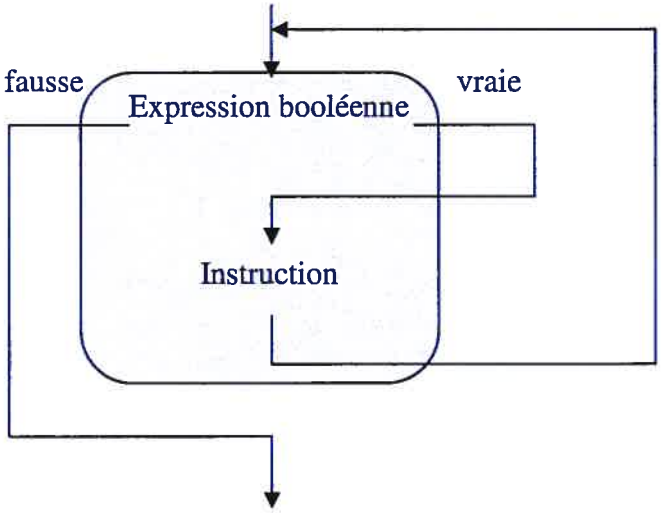
Trois structures de répétition :

```
while (expression_booléenne)
  instruction;
```

Tant que l'expression booléenne est vraie exécuter l'instruction qui peut être simple ou composée.

Points clés: -Prévoir une initialisation
-S'assurer que l'expr. Booléenne devienne fausse (critère d'arrêt)
-Dans le corps de la boucle, inclure une ou des instructions affectant l'expression booléenne

Fonctionnement :

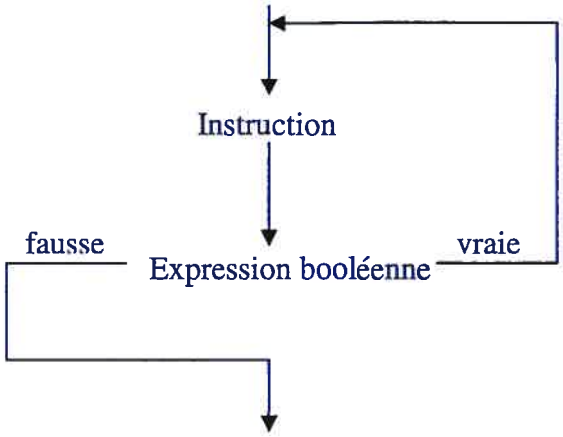


```
do
   instruction;
while (expression_booléenne);
```

Faire l'exécution de (exécuter) l'instruction qui peut être simple ou composée tant que l'expression booléenne est vraie.

Points clés : -S'assurer que l'expr. booléenne devienne fausse (critère d'arrêt)
-Dans le corps de la boucle, inclure une ou des instructions affectant l'expression booléenne
-Le corps de la boucle est exécuté au moins une fois

Fonctionnement



```
for  (initialisation; expression_booléenne; actualisation)
                instruction;
```

L'instruction for( ) prévoit dans sa syntaxe une initialisation, une expression booléenne et une actualisation. L'actualisation correspond à une ou des instructions affectant l'expression booléenne.

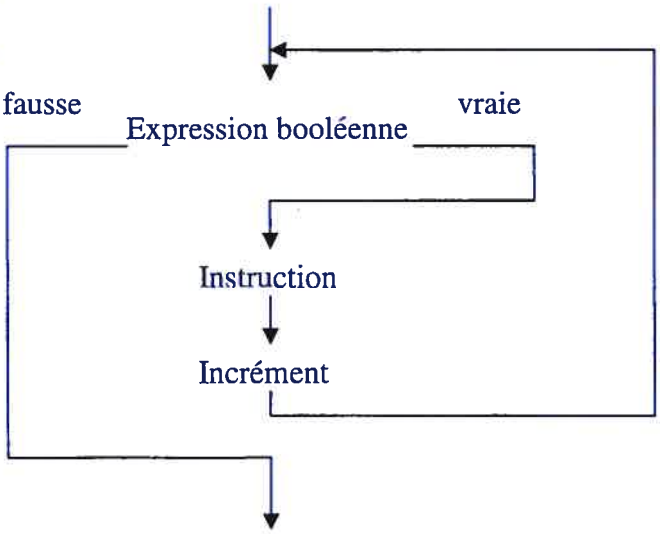
L'ordre d'exécution des trois composantes est décrit dans l'algorithme ci-dessous.

```
          1                    2                    4
for  (initialisation;  expression_booléenne;   actualisation)
        instruction;     3
```

1 l'initialisation est effectuée qu'une seule fois au départ
2 l'expression booléenne est vérifiée, si elle est vraie l'instruction 3 est exécutée, autrement la boucle est terminée
3 l'instruction à répéter; une fois terminée on passe à l'actualisation 4
4 l'actualisation doit permettre la modification de l'expr.booléenne,

on passe à 2 ensuite à 3 ensuite à 4....bref on boucle

Fonctionnement :



Distinction et « cadre » d'utilisation de chacune

while	do-while	for
Il faut tester l'expression booléenne avant l'exécution de l'instruction.	Il faut exécuter l'instruction au moins une fois avant que l'expression booléenne soit testée.	Comme le while sauf que l'instruction ne devrait pas affecter l'expression booléenne.
Le nombre de répétitions de l'instruction n'est pas connu d'avance	Le nombre d'exécutions du bloc d'instructions n'est pas connu d'avance.	Le nombre de répétitions de l'instruction est généralement connu d'avance.

Retour ou reprise des expressions booléennes

b. Analogie

Remplir le réservoir d'essence (TANT QU'il n'est pas plein – Ajouter de l'essence)



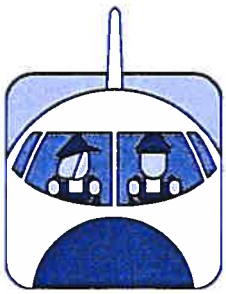
Montrer une interface usuelle d'une pompe d'essence
L'étudiant choisit le type d'essence : ordinaire, super, ultra (INITIALISATION)
Appuyer pour remplir – on voit la jauge d'essence qui augmente

Distribuer les copies d'examen (POUR chaque étudiant de la classe…)

Afficher des pupitres
L'étudiant clique sur chaque chapitre pour faire apparaître une feuille (deux images)



Valider une information (RÉPÉTER –Demander l'autorisation de décollage …)



En cliquant sur l'icône des pilotes on voit apparaître la demande de décollage ensuite on clique sur l'icône de la tour de contrôle pour connaître la réponse.

L'odomètre d'une automobile
L'horloge
Etc.

c. Exercice

Construction de chacune des structures
Instructions simples
Instructions imbriquées
Deviner le comportement d'une « boucle »

Pour les exercices suivants, l'étudiant doit choisir la structure de répétition et on doit l'encadrer pour composer l'expression booléenne. Idéalement, j'aimerais bien les étapes : choix de la structure, préciser l'initialisation, l'expression booléenne et l'actualisation. Un défi…

- Demander l'âge d'une personne tant que la réponse n'est pas entre 7 et 77 ans. [do-while]

```
int  Age;
do
{
  cout << "Age : ";
  cin  >> Age;
}
while (Age<7 || Age>77);
```

- Lire du clavier dix valeurs entières. [for]

```
int Entier, Compteur;
for (Compteur=1; Compteur<=10; Compteur++)
 {
    cout << " Valeur "   << Compteur << " : " ;
  cin >> Entier;
}
```

- Sommer tous les nombres positifs entrés par l'usager au clavier. La lecture d'un nombre nul ou négatif interrompt la sommation et le résultat est affiché. [while]

```
int Entier,
    Somme =0;
cout << " Valeur = ";
cin >> Valeur;
while (Entier>0)
{
   Somme + =Entier;
  cout << " Valeur = ";
  cin >> Valeur;
}
cout << " La somme des entiers positifs est " << Somme;
```

- Afficher le cosinus des angles compris entre 0 et 359 degré sur des lignes distinctes. Prévoir une interruption à toutes les 10 lignes pour en permettre une bonne lecture. [do-while]

```
const double PI=3.14159;
int Degre=1;
double Radian;  // Conversion pour le cosinus
 do
 {
    Radian = Degre * PI / 180;
    cout << cos(Radian) << endl;
    if (Degre % 10 ==0 )
    {
      cout << " Appuyer sur ENTER pour continuer";
      cin.get() :
   }
    Degre++;
}
while (Degre< 360)
```

- Afficher toutes les lettres de l'alphabet. [for]

```
char Lettre;
for (Lettre ='a'; Lettre<='z'; Lettre++)
  cout << Lettre << endl;
```

- Afficher le double du carré des nombres entiers tant que le résultat est inférieur à 150 000.

```
int Entier ,
    Expression = 1;
Expression = 2*pow(Entier,2);
while ( Expression < 150000)
{
   cout << Expression << endl;
   Entier++;
```

Expression = 2*pow(Entier,2);
}

d. Exemple

```
#include <iostream>
#include <cctype>
#include <cmath>

using namespace std;

void main()
{

  int Entier, Somme;
  char Lettre;

  // Additionner les entiers tant que leur somme est inférieure à 50
  Somme = 0;
  Entier = 0;

  while (Somme < 25)
  {
     cout << "Entier= " << Entier << " Somme= " << Somme << endl;
     Entier++;
       Somme += Entier;
  }
   cout << endl << endl;

  // Vérifier que l'usager à bien entrer une lettre, autrement
recommencer.
   do
   {
         cout << "Entrer une lettre: ";
         cin >> Lettre;
   }
   while (!isalnum(Lettre) || isdigit(Lettre));

     cout << endl << endl;

   // Afficher le carré et la racine carrée des nombres entiers de 10
à 15
   cout << "Entier  Racine    Carre" << endl;
   cout << "        carree" << endl;
   cout.setf(ios::showpoint);
   for (Entier=10; Entier<=15; Entier++)
        cout << Entier << "      " << sqrt(Entier) << "    "
             << (Entier*Entier) << endl;
}
```

```
#include <iostream>
#include <fstream>

using namespace std;

void main()
{

   int Entier, Compteur;
   ifstream Fichier;


   //Ouverture du fichier
   Fichier.open("Entiers.txt");

  // Lecture initiale
   Fichier >> Entier;
  // Boucle de lecture du fichier
   while (!Fichier.eof())
   {
      cout << Entier << ' ';
      for (Compteur=1;Compteur<10;Compteur++)
      {
         Fichier >> Entier;
         cout << Entier << ' ';
      }
      cout << endl;
      Fichier >> Entier  ;
   }
  Fichier.close();
}
```

```
#include <iostream>
#include <cctype>

using namespace std;

void main()
{
   int Ligne, Colonne;
   char Lettre, Reponse;

   // Afficher 3 lignes de XX
   Ligne = 1;
   while (Ligne < 4)
   {
      Colonne = 1;
      while (Colonne <3)
     {
        cout << "X";
        Colonne++;
       }
     cout << endl;
     Ligne++;
   }
   cout << endl << endl;

   // Tant que l'usager le désire, vérifier que l'usager a
   // bien entre une lettre, autrement recommencer.
   do
   {
      do
      {
            cout << "Entrer une lettre: ";
            cin >> Lettre;
      }
      while (!isalnum(Lettre) || isdigit(Lettre));
      cout << Lettre << " est bien une lettre." << endl;

      cout << "Recommencer (O/N)? ";
      cin >> Reponse;
   }
   while (toupper(Reponse)=='O');
   cout << endl << endl;

   // Afficher un astérisque sur la première ligne,deux sur
   // la deuxième ligne et trois sur la troisième ligne

   for (Ligne=1; Ligne<4; Ligne++)
   {
      for (Colonne=1;Colonne<=Ligne;Colonne++)
            cout << '*';
      cout << endl;
   }
}
```

6. Type énumération et type tableau

Le type énumération

a)Définition

Ce type permet d'expliciter les valeurs que pourra prendre une variable dans le but d'éviter de commettre des erreurs et d'augmenter par le fait même la lisibilité du programme par l'utilisation d'identificateur significatif

Le mot réservé enum est utilisé pour déclarer un type énumération suivi de l'identificateur du type et de l'énumération des valeurs inscrites entre accolades.

Exemple,
enum type_direction {NORD, SUD, EST, OUEST};
type_direction Direction;

L'identificateur type_direction précise un type énumération défini par l'usager et la variable Direction (de ce type) peut prendre l'une des valeurs énumérées soit NORD, SUD, EST ou OUEST.

Les identificateurs de l'énumération, ici NORD, SUD, EST ou OUEST, sont des identificateurs de constante. Par convention, ils sont en majuscules.

Le compilateur attribue une valeur ordinale à chaque identificateur de l'énumération. Pour notre type, type_direction :

- NORD vaut 0
- SUD vaut 1
- EST vaut 2
- OUEST vaut 3

Cependant, il est possible de préciser la valeur entière associée à un identificateur de l'énumération

Par exemple, pour la déclaration

```
enum type_couleur {BLEU=1, VERT, ROUGE=4, JAUNE=14};
```

- BLEU vaut 1
- VERT vaut 2
- ROUGEvaut 4
- JAUNE vaut 14

b) Opérateur

Puisqu'il existe une relation d'ordre, les opérateurs relationnels

= =, !=, >, <, >= et <=

s'appliquent sur des variables de type énumération.

Par exemple, pour la déclaration du type

enum type_direction {NORD, SUD, EST, OUEST};
type_direction Direction ;

On déduit que

NORD < SUD < EST < OUEST

c) Utilisation

- Lecture et affichage

Il n'est pas possible de lire du clavier ou d'un fichier texte une valeur appartenant à un type énumération.

L'affichage correspond à la valeur entière associée à l'identificateur de l'énumération

---

EXERCICE

1- Quelle instruction permettra de connaître et d'afficher « on a trouve le nord » si la direction est NORD sinon afficher « on a perdu le nord »

```
if  (Direction == NORD)
     cout << " on a trouve le nord "  ;
else
     cout << " on a perdu le nord "  ;
```

Une expression de type énumération peut être utilisée dans une structure conditionnelle switch(). Par exemple,

2- En utilisant une structure conditionnelle switch() afficher la valeur de la variable direction.

```
switch (Direction)
{
   case NORD  : cout << " La direction est nord "  ;
                break ;
   case SUD   : cout << "  La direction est sud "  ;
                break ;
   case EST   : cout << "  La direction est est "  ;
                break ;
   case OUEST : cout << "  La direction est ouest "  ;
}
```

3- Déclarer dans une énumération chaque jour de la semaine. Utiliser une structure de répétition avec cette énumération pour compter le nombre de jours dans une semaine.

```
NbreJour = 0 ;
For (Jour=LUNDI ; Jour<=DIMANCHE; Jour=type_Jour(Jour+1))
    NbreJour++ ;
```

Les tableaux

a) Définition

- Un tableau est une structure homogène constituée d'un nombre déterminé d'éléments de même type. En fait, un tableau est une variable qui permet de mémoriser plusieurs données de même type.

- Déclaration

  En premier lieu le type unique des données est précisé, suivi de l'identificateur de variable et du nombre de données que pourra mémoriser le tableau inscrit entre crochets.

  Par exemple,

  ```
  int  UnTableau [10] ;
  ```

  est un tableau pouvant mémoriser 10 valeurs entières de type int.

  Un tableau peut s'exprimer à l'aide de plusieurs dimensions. Pour ce faire, on doit inscrire entre crochet le nombre d'éléments de chaque dimension.

  Par exemple,

  ```
  double Date[31] [12] ;
  ```

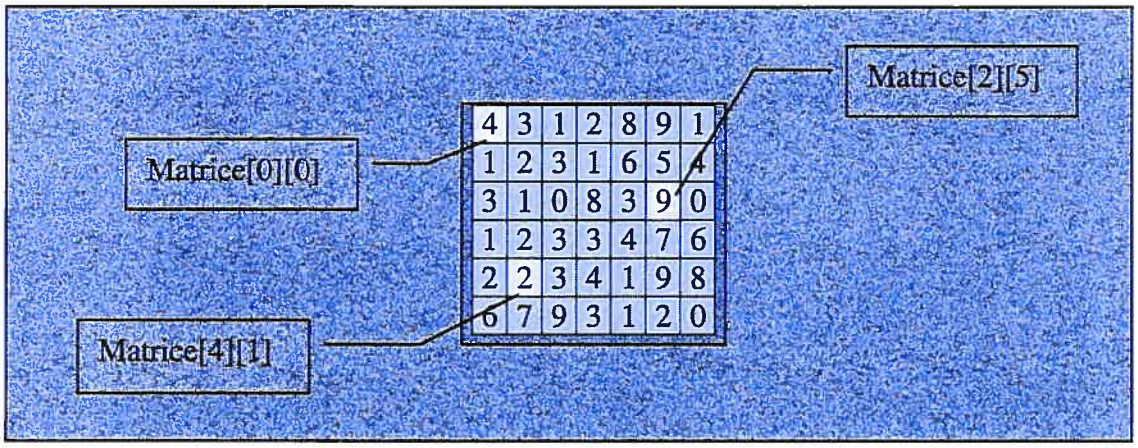
  est un tableau pouvant mémoriser 31 X 12 soit 372 données réelles de type double. La première dimension permet de traiter chaque jour d'un mois et la deuxième dimension permet de préciser le mois de l'année. Il est beaucoup plus simple de préciser un jour de l'année par le numéro du jour d'un mois plutôt que le numéro du jour selon l'année.

  Il est permis de déclarer un tableau du nombre de dimensions désirées. La seule contrainte est l'espace maximal que le compilateur accepte pour l'ensemble des données de votre programme.

b) Accès aux éléments d'un tableau

- On peut repérer chaque élément mémorisé dans le tableau à l'aide d'un indice qui sert à indiquer sa position
- Le premier élément de chaque dimension d'un tableau est toujours à la position 0

int Matrice[6][7] ;



- Pour ce tableau, l'affectation d'une valeur s'effectue à l'aide de l'instruction :
  int Matrice[4][1] = 2 ;

ATTENTION ! MISE EN GARDE ! ATTENTION !

- Il est important de noter que les valeurs possibles pour la première dimension vont de 0 à 5 et que les valeurs possibles pour la deuxième dimension vont de 0 à 6.

  Les affectations : Matrice[5][2] et Matrice[3][7] sont erronées.

- L'utilisation d'un indice de type énumération peut rendre très explicite l'accès à un élément du tableau.

```
enum type_grade{BING,MSCA, PHD};
int NbreGradues[PHD+1];
// L'instruction suivante permettra de mémoriser qu'il y a
// 653 gradués au baccalauréat en ingénierie :
NbreGradues[BING] = 653;
```

c) Manipulations

Il n'est pas possible de manipuler le tableau dans son entité. En d'autres mots, on ne peut pas effectuer de lecture, d'affichage, de comparaison ni d'autres opérations sur des tableaux complets (sauf les chaînes de caractères).

EXERCICE

Étant donné les déclarations suivantes

```
enum type_dept{MECA ,ELEC, INFO, MATH};
enum type_grade{BING,MSCA, PHD};
enum type_sexe{FEMME,HOMME} ;
int GradueDept[MATH+1][PHD+1][HOMME+1];
```

- 1- Donner l'instruction qui calculera le nombre de Ph.D. décernés par le département de génie électrique:

```
int NbrePhD;
NbrePhD = GradueDept[ELEC][PHD][FEMME] +
          GradueDept[ELEC][PHD][HOMME];
```

- 2- Donner la structure de répétition qui permettra de calculer le nombre de baccalauréat décerné aux femmes pour chacun des départements.

```
type_dept Dept;
int NbreBIng = 0;
for (Dept=MECA ; Dept<=MATH ; Dept=type_dept(Dept+1))
   NbreBIng + = GradueDept[Dept][BING][FEMME];
```

Les déclarations suivantes vont nous servir à illustrer quelques opérations le plus couramment effectuées sur les tableaux.

```
double VecteurA[10], VecteurB[10];
int Indice;
```

3- Donner la structure de répétition qui permettra d'initialiser chaque valeur de VecteurA à 0.

Initialisation d'un tableau

```
for (Indice=0;Indice<10;Indice++)
  VecteurA[Indice]=0;
```

4- Donner la structure de répétition qui permettra de copier VecteurA dans VecteurB

Affectation d'un tableau à un autre

```
for (Indice=0;Indice<10;Indice++)
  VecteurB[Indice]=VecteurA[Indice];
```

5- Utiliser une structure de répétition pour vérifier si le VecteurA est identique au VecteurB, i.e. qu'ils contiennent les mêmes valeurs au même emplacement.

Comparaison de deux tableaux

```
Pareil=true;
for (Indice=0;Indice<10 && Pareil; Indice++)
  if(VecteurA[Indice]!=VecteurB[Indice])
     Pareil = false;
```

6- Lire du clavier 10 valeurs et les mémoriser dans la variable VecteurA

Lire et afficher un tableau

```
// Lecture
for (Indice=0;Indice<10;Indice++)
{
  cout << "Valeur " << Indice << " = ";
  cin >> VecteurA[Indice];
}
```

7- Afficher les valeurs contenues dans la variable VecteurA

```
// Affichage
for (Indice=0;Indice<10;Indice++)
  cout << VecteurA[Indice] << endl;
```

Analogie tableau

Un vecteur
Une liste d'épicerie : insister sur l'homogénéité des valeurs

Lait
Pain
Salade
Tomate
Pomme

Lait	Pain	Salade	Tomate	Pomme

Une matrice

Un joueur de quille réalise quatre parties. Nous désirons mémoriser le résultat de chaque partie à chacun des dix carreaux.

9	18	36	45	64	84	102	110	116	141
20	40	58	67	93	113	131	139	159	179
30	60	90	120	150	180	210	240	270	300
17	35	37	43	58	67	72	80	82	91

La première ligne représente les résultats de la première partie. La première case correspond au résultat obtenu au premier carreau. La deuxième case correspond au résultat obtenu au deuxième carreau. Ainsi de suite.

Pour faciliter le repérage nous avons ajouté le numéro du carreau et le numéro de la partie.

	1	2	3	4	5	6	7	8	9	10
Partie 1	9	18	36	45	64	84	102	110	116	141
Partie 2	20	40	58	67	93	113	131	139	159	179
Partie 3	30	60	90	120	150	180	210	240	270	300
Partie 4	17	35	37	43	58	67	72	80	82	91

Un cube

Utiliser 3 signets pour un dialogue

Entraîneur de natation vous désirez conserver les trois meilleurs temps de vos quatre protégés à l'épreuve quatre cent mètres quatre nages pour chacune des trois dernières semaines.

4 :35.00	4.37.12	4.38.15
4 :27.87	4 :28.23	4 :28.89
4 :32.98	4 :35.65	4 :36.01
4 :28.11	4 :29.23	4 :29.56

4 :33.23	4 :34.67	4 :34.85
4 :25.98	4 :27.12	4 :28.01
4 :32.57	4 :34.77	4 :34:82
4 :29.34	4 :31.57	4 :33.89

4 :35.89	4 :36.21	4 :37.75
4 :27.87	4 :28.23	4 :28.89
4 :32.50	4 :33.62	4 :34:81
4 :30.00	4 :32.25	4 :33.27

Analogie énumération

Les jours de la semaine
Insister sur l'idée qu'un jour ne peut pas s'appeler jeumedi. Il doit appartenir à l'énumération.

Les couleurs de l'arc-en-ciel

Les états civils possibles

Les journaux montréalais

---

d) Exemples

```
#include <iostream>          // Pour l'utilisation de cin et cout
#include <cstring>           // Pour l'utilisation de strcpy()
#include <cstdlib>           // Pour l'utilisation de rand() et srand()
#include <ctime>             // Pour l'utilisation de time()

using namespace std;

void main (void)
{
   enum  valeur_carte { VALET, DAME, ROI, AS };
   enum  sorte_carte { PIQUE, TREFLE, CARREAU, COEUR };

   srand( (unsigned)time( NULL ) );   //Un germe pour le generateur
   cout << "Choix aleatoire d'une carte: ";
   switch (rand() % 4)              // Choix aleatoire de la valeur
   {
      case  VALET     : cout << " Valet de";
                        break;
      case  DAME: cout << " Dame de";
                        break;
      case  ROI       : cout << " Roi de";
                        break;
      case  AS   : cout << " As de";
   }

   switch (rand() % 4)              // Choix aleatoire de la sorte
   {
      case  PIQUE     : cout << " Pique";
            break;
      case  TREFLE    : cout << " Trefle";
            break;
      case  CARREAU   : cout << " Carreau";
            break;
      case  COEUR     : cout << " Coeur";
   }
   cout << endl;
}
```

```
#include <iostream>

using namespace std;

#define NBVALEURS 7

void main()
{

   int Vecteur[NBVALEURS];
   int i, Somme, PlusPetite;

   // Initialisation du tableau
   for (i=0; i<NBVALEURS; i++)
      Vecteur[i] = 0;

   // Lecture du clavier des 7 valeurs
   for (i=0; i<NBVALEURS; i++)
   {
      cout << "Vecteur[" << i << "]= ";
       cin >> Vecteur[i];
   }
   cout << endl;

   // Sommation des valeurs lues
   Somme = 0;
   for (i=0; i< NBVALEURS; i++)
     Somme += Vecteur[i];

   cout  << "La somme de ces valeurs est: " << Somme << endl;

   // Recherche de la plus petite valeur
   for (PlusPetite=Vecteur[0], i=1; i<NBVALEURS; i++)
      if (Vecteur[i] < PlusPetite)
          PlusPetite = Vecteur[i];
   cout << "La plus petite valeur est " << PlusPetite << endl;
}
```

7.

```
#include <iostream>

using namespace std;

#define NBVALEURS 5

void main()
{

      int TabA[NBVALEURS]={3,2,7,5,1},
          TabB[NBVALEURS];
   int i, Tampon;
   bool Identique;

   // Copie du tableau TabA dans le tableau TabB
   for (i=0; i<NBVALEURS; i++)
      TabB[i] = TabA[i];

   // Modification de la deuxième et la troisième valeur
   // de TabA
   cout << "Donner deux valeurs entieres: ";
   cin >> TabA[1] >> TabA[2];
   cout << endl;

   // Vérification de l'égalité des deux tableaux
   Identique = true;
   for (i=0; i< NBVALEURS && Identique; i++)
     if (TabA[i] != TabB[i])
           Identique = false;
   if (Identique)
       cout << "Les deux tableaux sont encore identiques"
            << endl;
   else
       cout << "Les deux tableaux ne sont plus identiques"
            << endl;


   // Inversion des valeurs du tableau TabB
   for (i=0; i<NBVALEURS/2; i++)
   {
        Tampon = TabB[i];
        TabB[i] = TabB[NBVALEURS-1-i];
       TabB[NBVALEURS-1-i] = Tampon;
   }

   cout << endl;
}
```

7. Fonctions

Première unité développée à l'été 2000.

Les spécifications ont été précisées au fur et à mesure du développement.

8. Les enregistrements

a. Définition

Un enregistrement est un type qui permet d'intégrer sous une seule variable toutes les caractéristiques d'une même entité. Par exemple le titre, le nom de l'auteur, l'éditeur, l'année de parution et le nombre de pages d'un livre peuvent être mémorisés à l'aide d'une seule variable de type enregistrement.

- Un enregistrement est une structure de données formée d'un certain nombre de champs portant chacun un nom et pouvant être de types différents.
- L'enregistrement permet de regrouper, sous un même identificateur, des données diverses mais logiquement interreliées
- Un type enregistrement est déclaré à l'aide du mot réservé struct.

```
struct type_struct
{
      déclaration des champs;
};
```

```
struct type_livre
{
  char  Titre[120];
  char  Auteur[80];
  char  Editeur[120];
  int   Annee;
  int   NbPages;
};
type_livre Livre;
```

- Initialisation d'un enregistrement

```
type_Livre = {"Tu dis tu soda!", "Par Di", "La lune nette",
              1898, 47};
```

- Accès aux champs d'un enregistrement

On accède à un champ à l'aide de l'identificateur de la variable suivi d'un point et du champ désiré : Variable.Champ

Par exemple, les instructions

```
cout << Livre.Titre << endl;
cout << Livre.Annee << end ;
```

permettent d'afficher le titre et l'année de parution du livre.

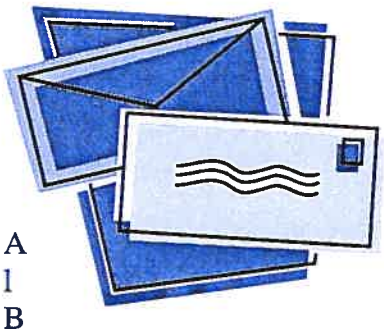
- La seule opération possible sur une entité enregistrement (dans sa globalité) est l'affectation à un autre enregistrement de même type. Toutes les autres opérations doivent être définies à l'aide des champs. Par exemple, il est

impossible de comparer deux enregistrement autrement qu'en vérifiant l'égalité entre chacun des champs correspondants.

- Peuvent être utiles pour transmettre plusieurs informations en paramètre.

b. Analogies

Adresse postale (Une grande chaîne de caractères ou plusieurs champs)



A
l
B

```
struct type_adresse
{
    char Prenom[80],
         Nom[80];
    int  NoCivique;
    char Rue[50];
    char Ville[50];
    char Province[20];
    char CodePostale[7];
};
```

En cliquant sur l'enveloppe on voit apparaître la description de l'enregistrement. OU on clique sur l'enveloppe et une fenêtre s'affiche avec une adresse précise, on doit cliquer sur chaque élément pour le voir apparaître dans l'enregistrement.

Travail de recherche sur la météorologie

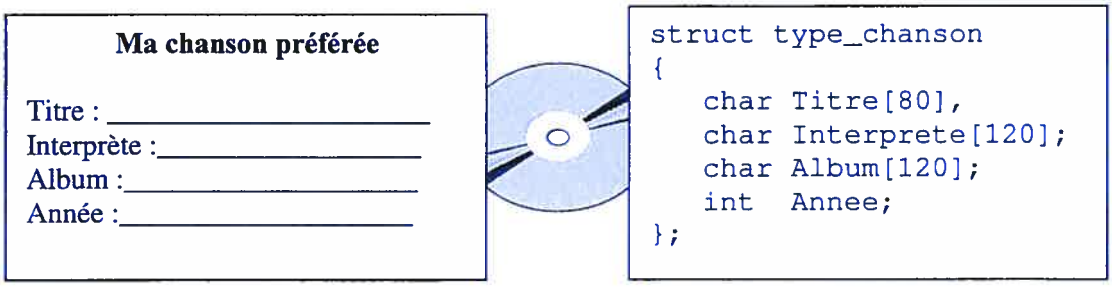
Votre professeur vous demande de réaliser un travail qui consiste à comparer des observations météorologiques de vingt villes québécoises. Pour ce faire, vous devez choisir cinq types d'observations parmi les différentes observations suivantes : température minimale, température maximale, humidité relative, vitesse des vents, direction des vents, pression, point de rosée, précipitation de pluie, précipitation de neige, hauteur du plafond, visibilité horizontale.

Observations météo retenues

```
struct type_meteo
{
    char Ville[80];
    ...     Obs1;
    ...     Obs2;
    ...     Obs3;
    ...     Obs4;
    ...     Obs5;
};
```

L'usager peut choisir chaque observation à l'aide d'un « combo box ». La description équivalente apparaît dans l'enregistrement.

Ma chanson préférée



En cliquant sur le formulaire, l'usager voit apparaître l'enregistrement nécessaire ou utiliser pour mémoriser l'information. À la saisie de chaque champ l'instruction équivalente est affichée dans une fenêtre ou un « panneau ».

c. Exercices

Déclarer un enregistrement qui permettra de mémoriser la description

- d'une voiture : le nom du fabricant, le nom du modèle, le type de transmission (automatique ou manuel), le nombre de passagers et le nombre de kilomètres par litre.
- d'un micro contrôleur : dimension ROM, dimension RAM, nombre de broches, nombre de ports E/S et la présence ou l'absence d'un TIMER.
- d'un logiciel : le nom du logiciel, le nom du fabricant, le prix.
- d'une bouteille de vin : appellation, couleur, cépage, année de mise en bouteille, pourcentage d'alcool, volume.

Mise à jour de certains champs

- Étant donné la déclaration suivante

```
struct type_feuille
{
   double Longueur,
          Largeur;
};
type_feuille Feuille = {8.5, 11};
```

Donnez la ou les instructions qui permettront de mémoriser à l'aide de la variable Feuille les dimensions 8.5 par 14.

- Tableau d'enregistrements

- Étant donné les déclarations suivantes

```
#define NBJOUEURS 5
```

```
struct type_joueur
{
   char Nom[30];
   int   NoDossard;
};
type_joueur Equipe[NBJOUEURS] = {{"Pierre",12},{"Jean",7},
{"Jacques",23}, {"Marie",15}, {"Helene", 5}};
int i;
```

Donnez la ou les instructions qui permettront de trouver le numéro de dossard de Jacques.

[On peut y arriver avec les trois répétitives]

```
i=0;
while(strcmp(Equipe[i].Nom, "Jacques")!=0)
i++;

for (i=0; strcmp(Equipe[i].Nom, "Jacques")!=0;i++) ;

i = -1;
do
{
   i++;
}
while(strcmp(Equipe[i].Nom, "Jacques")!=0);
```

- Comparaison de deux enregistrements de même type

Étant donné les déclarations suivantes :

```
struct type_produit
{
   char Nom[80];
   unsigned int Quantite;
} ;

ProduitA, ProduitB ;
```

Donner l'instruction qui permettra de vérifier si les deux produits sont identiques.

```
if (strcmp(ProduitA.Nom,ProduitB.Nom)==0 &&
   ProduitA.Quantite == ProduitB.Quantite)
```

- Comparaison de deux enregistrements

```
struct type_lumiere
{
```

```
        int Voltage,
            Amperage;
};

type_lumiere Lampe, Plafond;
```

Donner la ou les instructions qui permettront de vérifier si la lumière au plafond et de la lampe ont le même voltage et le même ampérage.

```
if (Lampe.Voltage == Plafond.Voltage && Lampe.Amperage ==
Plafond.Amperage)
  Cout << « Les deux lumieres sont identiques »;
else
  Cout << « Les deux lumieres sont differentes »;
```

- Modification

```
Soit les déclarations suivantes
Struct type_planche
{
   int Epaisseur,
       Largeur,
       Longueur;
};
type_planche Planche = { 2, 4, 10};
```

Donner la ou les instructions qui permettront de modifier la planche au dimension : Epaisseur (4), Largeur (6), Longueur (12).

d. Exemples

Déclaration de deux enregistrements avec initialisation
Les deux enregistrements ont les mêmes valeurs
Les deux enregistrements n'ont pas les mêmes valeurs
Vérifier le comportement de chaque opération
Faire l'égalité entre les deux (pas champ à champ)
Utilisation d'un enregistrement en paramètre
Par valeur et par adresse

```
#include <iostream>
#include <cmath>
#include <iomanip>

using namespace std;

struct type_point
{
   double x, y, z;
};

void main()
{

   type_point Origine = {0,0,0},
              Point;
   double Distance;

   cout.setf(ios::fixed);
   cout<< setprecision(2);

   // Affectation d'un enregistrement à un autre
   Point = Origine;

   // Lecture du point
   cout << "Coordonnee x du point: ";
   cin >> Point.x;
   cout << "Coordonnee y du point: ";
   cin >> Point.y;
   cout << "Coordonnee z du point: ";
   cin >> Point.z;
   cout << endl;

   //Vérification si les deux points sont identiques
   if (Origine.x == Point.x && Origine.y == Point.y &&
       Origine.z == Point.z)
            cout << "Les deux points sont identiques." << endl;
   else
            cout << "Les deux points sont distincts." << endl;
      cout << endl;

   //Calcul de la distance du point selon l'origine
   Distance = sqrt( pow(Point.x-Origine.x,2) + pow(Point.y-Origine.y,2) +
                    pow(Point.z-Origine.z,2));

   cout << "La distance du point a l'origine est " << Distance << endl;

}
```

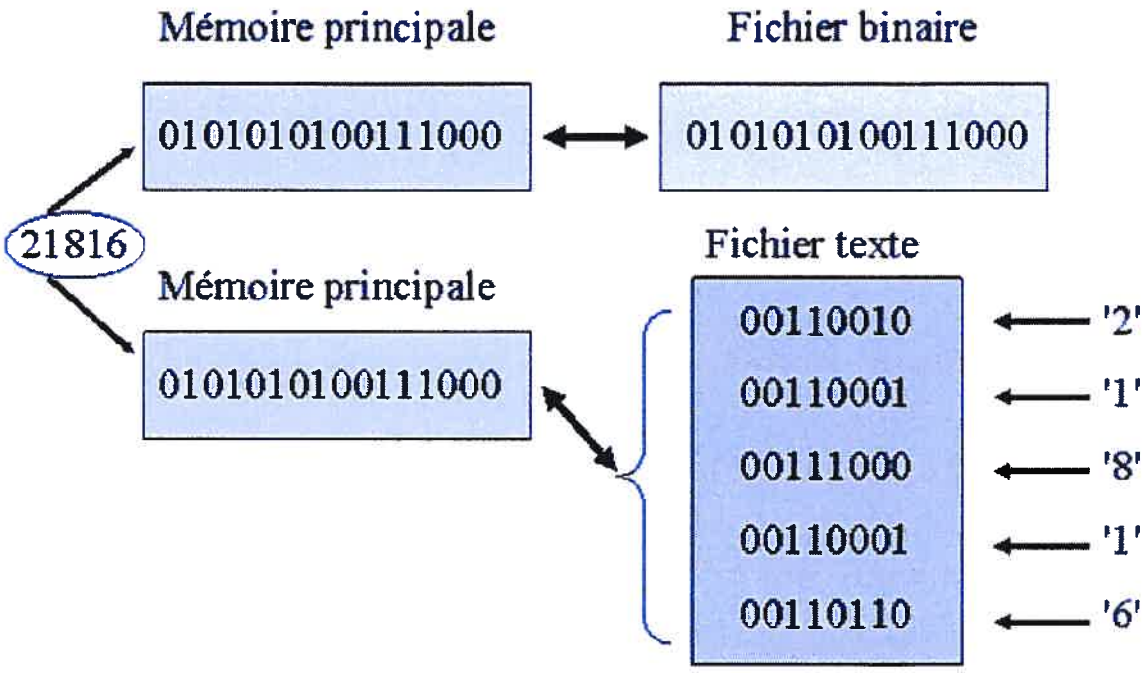
```
#include <iostream>
#include <cmath>
#include <iomanip>
#include <cstring>
using namespace std;

struct type_employe
{
   char Prenom[50], Nom[50];
   char Tache[120];
   unsigned long No_Social;
};

void main()
{

   type_employe TabEmploye[5],
                Tampon;
   int  i, j;

   // Lecture des employés
   for (i=0; i<5; i++)
   {
      cout << "Prenom: ";
      cin.getline(TabEmploye[i].Prenom,50);
      cout << "Nom: ";
      cin.getline(TabEmploye[i].Nom, 50);
      cout << "Tache: ";
      cin.getline(TabEmploye[i].Tache,120);
      cout << "Numéro social: ";
      cin >> TabEmploye[i].No_Social;
      cin.ignore(); // pour enlever le ENTER
   }

   // Tri des employes selon le nom et le prénom
   for (i=0;i<5; i++)
      for (j=i+1;j<5;j++)
         if ((strcmp(TabEmploye[i].Nom,TabEmploye[j].Nom)>0) ||
             (strcmp(TabEmploye[i].Nom,TabEmploye[j].Nom)==0 &&
              strcmp(TabEmploye[i].Prenom,TabEmploye[j].Prenom)>0))
         {
            Tampon = TabEmploye[i];
            TabEmploye[i] = TabEmploye[j];
            TabEmploye[j] = Tampon;
         }

   //Affichage des employes dans l'ordre alphabétique croissant
   cout << endl << "Employes tries" << endl;
   for (i=0; i<5; i++)
   {
      cout << "Prenom: " << TabEmploye[i].Prenom << endl;
      cout << "Nom: " << TabEmploye[i].Nom << endl;
      cout << "Tache: " << TabEmploye[i].Tache << endl;
      cout << "Numéro social: " << TabEmploye[i].No_Social << endl;
      cout << endl;
   }
}
```

```
#include <iostream>
#include <cmath>
#include <iomanip>
using namespace std;

struct type_employe
{
      char Prenom[50], Nom[50];
      char Tache[120];
      unsigned long No_Social;
};

void Permuter(type_employe &Un, type_employe &Deux)
{
      type_employe Tampon;

      Tampon = Un;
      Un = Deux;
      Deux = Tampon;
}

void main()
{

   type_employe TabEmploye[5],
                Tampon;
   int i, j;

   // Lecture des employés
   for (i=0; i<5; i++)
   {
      cout << "Prenom: ";
      cin.getline(TabEmploye[i].Prenom,50);
      cout << "Nom: ";
      cin.getline(TabEmploye[i].Nom, 50);
      cout << "Tache: ";
      cin.getline(TabEmploye[i].Tache,120);
      cout << "Numéro social: ";
      cin >> TabEmploye[i].No_Social;
      cin.ignore();
   }

   // Tri des employes selon le no. social
   for (i=0;i<5; i++)
      for (j=i+1;j<5;j++)
         if (TabEmploye[i].No_Social > TabEmploye[j].No_Social)
            Permuter (TabEmploye[i],TabEmploye[j]);

   //Affichage des employes dans l'ordre alphabétique croissant
   cout << endl << "Employes tries" << endl;
   for (i=0; i<5; i++)
   {
      cout << "Prenom: " << TabEmploye[i].Prenom << endl;
      cout << "Nom: " << TabEmploye[i].Nom << endl;
      cout << "Tache: " << TabEmploye[i].Tache << endl;
      cout << "Numéro social: " << TabEmploye[i].No_Social << endl;
      cout << endl;
   }
}
```

9. Fichier binaire

a. Définition

On appelle fichier binaire un fichier dans lequel les éléments sont représentés de la même façon que les données en mémoire. La figure suivante distingue l'écriture de la valeur 21816 dans un fichier binaire et dans un fichier texte.



- L'accès aux données d'un fichier binaire peut être séquentiel comme pour le fichier texte ou peut être direct en se positionnant à l'emplacement désiré du fichier.
- Le concept de ligne se terminant par un caractère de fin de ligne n'existe pas dans un fichier binaire.
- L'information est sous forme codée et généralement inintelligible lorsqu'elle est affichée avec un éditeur de texte.
- Un fichier binaire utilise en moyenne moins d'espace qu'un fichier texte pour la même information.
- Il est possible d'ouvrir un fichier binaire en mode lecture et écriture.

Organisation des données dans le fichier
Accès direct (repérage par adresse)

Déclaration
Les trois types de fichiers : ifstream, ofstream et fstream peuvent être utilisés pour déclarer un fichier binaire.

Par exemple,

```
ifstream Fic_Entree;  // pour un fichier qui sera uniquement lu
ofstream Fic_Sortie; // pour un fichier qui sera uniquement écrit
fstream  Fic_InOut; // pour un fichier qui sera lu et écrit
```

Ouverture

C'est à l'ouverture qu'on doit préciser que le fichier est binaire. En effet, l'ajout de l'énoncé ios::binary dans l'appel de la fonction open() annonce que le fichier est binaire.

Par exemple,

un fichier binaire qu'on désire lire

```
ifstream Fic_Entree;
Fic_Entree.open("Lit.dat",ios::binary);
```

un fichier binaire qu'on désire écrire

```
ofstream Fic_Sortie;
Fic_Sortie.open("Ecrit.dat",ios::binary);
```

Un fichier binaire qu'on désire lire et écrire

Dans ce cas il faut ajouter les énoncés ios::in et ios::out

```
fstream  Fic_InOut ;
Fic_InOut.open("LitEcrit.dat",ios::binarylios::inlios::out);
```

Lecture

La fonction read() préfixé de la variable fichier permet de lire un fichier binaire. Elle exige deux paramètres, le premier correspond à l'adresse de la variable où la donnée lue est copiée, le deuxième est la dimension en nombre d'octets de la donnée à lire du fichier.

Par exemple,

Fichier.read((char*)&Variable, 50);

Lit 50 octets à partir de la position courante dans Fichier et affecte (ou copie) ces 50 octets à Variable.

Généralement l'opérateur sizeof() est utilisé pour obtenir le nombre d'octets d'un type particulier. En effet, sizeof(int) retourne le nombre d'octets utilisés pour mémoriser un int. La forme générale d'utilisation de la fonction read() devient :

```
F_entree.read((char*)&Var, sizeof(type_var));
```

Écriture

La fonction write() préfixé de la variable fichier permet d'écrire une donnée dans un fichier binaire. Sa syntaxe est similaire à la fonction read(), puisqu'elle exige deux paramètres, le premier correspond à l'adresse de la

variable dont le contenu est à écrire dans le fichier, le deuxième est la dimension en nombre d'octets de la donnée à écrire dans le fichier.

Par exemple,
Fichier.write(char*)&Variable, 50);
Prendre les 50 octets du contenu de Variable et les écrire dans Fichier à partir de la position courante dans Fichier.

Généralement l'opérateur sizeof() est utilisé pour obtenir le nombre d'octets d'un type particulier. En effet, sizeof(int) retourne le nombre d'octets utilisés pour mémoriser un int. La forme générale d'utilisation de la fonction read() devient :

```
F_sortie.write((char*)&Variable, sizeof(type_variable));
```

Positionnement
La fonction seekg() préfixé de la variable fichier permet de se positionner dans le fichier ouvert en lecture. La fonction seekg() utilise deux paramètres. Le premier est un entier qui correspond à un décalage (un déplacement) selon une position de repère. Le deuxième paramètre correspond au repère à considérer qui peut être ios::beg (début du fichier), ios::cur (position courante dans le fichier) ou ios::end (fin du fichier).

Par exemple, l'instruction

```
F_Entree.seekg(0,ios::beg);  // positionne au début du fichier
Fic_Entree.seekg(0,ios::beg) ; // positionne 25 octets après le début du fichier
```

La fonction seekp() réalise exactement la même opération mais cette fois pour un fichier ouvert en écriture.

Position
Il est possible de connaître la position actuelle dans le fichier à l'aide de la fonction tellg() préfixé de la variable fichier. La fonction retourne un entier correspondant au nombre d'octets compris entre le début du fichier et la position actuelle.

Par exemple,

```
int Courant = Fic_Entree.tellg();
```

Un astuce souvent utilisé est de diviser le nombre obtenu par la dimension de chaque donné dans le fichier. Cette façon de faire simplifie le traitement en identifiant le numéro de la donnée plutôt que le nombre d'octets entre le début du fichier et la donnée.

Par exemple,

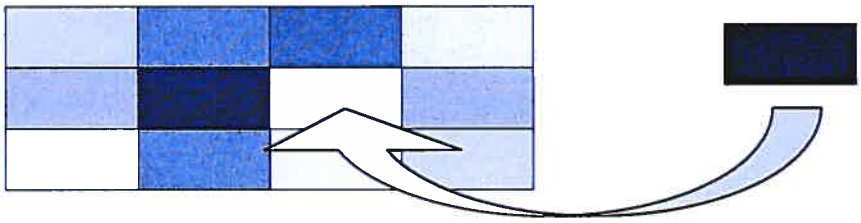
```
int Courant = Fic_Entree.tellg()/sizeof(type_variable);
```

b. Analogies

Disque compact

Positionnement à la chanson désiré
Présentation d'une interface axée sur le positionnement.

Un tableau
Repérage des éléments comme pour un tableau. Insister sur l'importance de modifier une donnée par une donnée de même dimension.



Recherche de la Grande Ourse dans le ciel

c. Exercices

Identification de la structure de données à utiliser
Choix du type du fichier : ifstream, ofstream ou fstream
Acquisition des étapes à suivre pour la manipulation d'un fichier :
Déclaration des variables fichier et données, ouverture du fichier, lecture et/ou écriture, fermeture du fichier.
Déplacement dans le fichier : début, courant, fin + sizeof()
Lecture, Écriture et Mise à jour

Pour le prochain type d'exercices il faudrait utiliser une interface qui oriente l'étudiant vers la séquence :

Déclaration de variables
Ouverture du fichier
Message de sollicitation
Lecture
<Affichage de l'information désirée>
Fermeture du fichier

- Déterminer la marque de café la moins cher parmi celle contenue dans le fichier Cafe.bin. L'enregistrement utilisé pour mémoriser l'information dans le fichier est :

```
struct type_cafe
{
   char Marque[50];
   double Prix;
};
```

- Afficher tous les livres de la maison d'édition « FlamaLion » présents dans le fichier Inventaire.Dat. L'enregistrement utilisé pour mémoriser l'information sur chaque livre dans le fichier est :

```
struct type_livre
{
   char Auteur[50];
   char Titre[120];
   char Editeur[80];
   int  Annee;
};
```

- Rechercher et afficher le numéro de téléphone de la personne nommée « Barbiere di Siviglia » à partir du fichier Annuaire.Dat. L'information est inscrite dans le fichier selon l'enregistrement déclaré :

```
struct type_personne
{
   char Nom[80];
   char Adresse[50];
   char  NoTel[9];
```

};

Déclaration de variables
Ouverture du fichier
Message de sollicitation
<Lecture au clavier>
Écriture du contenu des variables lues dans le fichier
Fermeture du fichier

- Écrire dans le fichier binaire Commande.Dat les articles précisés par le client. Les articles sont mémorisés selon le type :

```
struct type_article
{
   char Nom[30];
   int Quantite;
};.
```

- Écrire dans le fichier binaire Partie.Dat l'état de chaque case d'un échiquier lors d'une partie d'échec. L'information conservée pour chaque case est réalisée à l'aide du type :

```
struct type_case
{
  bool PiecePresent;  // vrai si une pièce est présente sinon faux
  char NomPiece[20]; // Nom de la pièce présente sur la case.
};
```

- Grande Vente 50% sur tous les articles en magasin. Réaliser la mise à jour du fichier Magasin.Dat en réduisant le coût de chaque article de 50%. Chaque article est mémorisé dans le fichier en respectant le type :

```
struct type_article
{
  char Nom[80];
  char Fournisseur[50];
  double Prix;
};
```

- Ajouter au fichier Piscine.Dat les données du dernier échantillon d'eau prélevé. Pour chaque échantillon les données sont inscrites dans le fichier selon le type :

```
struct type_echantillon
{
   float Temperature,
         Ph;
   char Clarte[15] ;
} ;
```

d. Exemple

```
#include <iostream>    // Pour l'utilisation de cin et cout
#include <fstream>     // Pour l'utilisation des fichiers
using namespace std;

void main (void)
{
   fstream Fichier;    // Fichier permettant de lire et ecrire
   int i, Pos, Valeur;


   // Creation du fichier.
   Fichier.open("ENTIER.BIN",ios::binary|ios::out);

   // Inscription des valeurs 10, 20, 30, 40 et 50.
   for (i=1; i<=5; i++)
   {
      Valeur = i*10;
      Fichier.write( (char*) &Valeur, sizeof(int));
   }

   Fichier.close();
   Fichier.open("ENTIER.BIN",ios::binary|ios::in);

   // Affichage de chaque valeur dans le fichier ainsi que sa position.
   cout << "Position       Valeur" << endl;
   cout << "----------------------" << endl;

   // Lecture du fichier avec boucle while
   Fichier.seekg(0,ios::beg);
   Pos = Fichier.tellg()/sizeof(int);
   Fichier.read((char *) &Valeur, sizeof(int));
   while (!Fichier.eof())
   {
      cout << Pos << "              " << Valeur << endl;
      Pos = Fichier.tellg()/sizeof(int);
      Fichier.read((char *) &Valeur, sizeof(int));
   }

   Fichier.close();
}
```

```
#include <iostream>   // Pour l'utilisation de cin et cout
#include <fstream>    // Pour l'utilisation des fichiers
using namespace std;

struct type_article
{
      char Nom[80];
      double Prix;
};

void Saisir(type_article &Article)
{
   cout << "Article: ";
   cin.getline(Article.Nom,80);
   cout << "Prix: ";
   cin >> Article.Prix;
   cin.ignore(); // Pour enlever le ENTER
   cout << endl;
}

void main (void)
{
   fstream Fichier;   // Fichier permettant de lire et écrire
   type_article Article;
   int i,Pos,Nb_Articles;


   // Création du fichier.
   Fichier.open("ARTICLE.BIN",ios::binary|ios::out);

   // Inscription de quatre articles dans le fichier
   for (i=0; i<4; i++)
   {
      Saisir(Article);
      Fichier.write( (char*) &Article, sizeof(type_article));
   }

   Fichier.close();
   // Ouverture du fichier en lecture et écriture
   Fichier.open("ARTICLE.BIN",ios::binary|ios::in|ios::out);

   // Se positionner au 3e article (position 2)
   Fichier.seekg(2*sizeof(type_article),ios::beg);

   // Affichage de la position courante
   cout << "Position dans le fichier = ";
   cout << Fichier.tellg()/sizeof(type_article) << endl;

   // Lecture
   Fichier.read((char *) &Article,sizeof(type_article));

   // Affichage de la position courante
   cout << "Position dans le fichier apres la lecture = ";
   cout << Fichier.tellg()/sizeof(type_article) << endl;
```

```
    // Affichage de l'article lu a la 3e position
    cout << "Le 3e article du fichier est " << Article.Nom << endl;
    cout << endl;

    // Se positionner a la fin du fichier.
    Fichier.seekp(0,ios::end);

    // Ajouter une valeur dans le fichier.
    Saisir(Article);
    Fichier.write((char *) &Article, sizeof(type_article));

    // Determiner le nombre d'articles dans le fichier.
    Fichier.seekg(0,ios::end);
    Nb_Articles = Fichier.tellg() / sizeof(type_article);

    // Se positionner au début et afficher le contenu du fichier
    cout << "Position       Valeur" << endl;
    cout << "----------------------" << endl;
    Fichier.seekg(0,ios::beg);

    for (i=1; i<= Nb_Articles; i++)  // Lecture du fichier
    {
          Pos = Fichier.tellg()/sizeof(type_article);
          Fichier.read((char *) &Article, sizeof(type_article));
          cout << Pos << "              " << Article.Nom
                << ' ' << Article.Prix << endl;
       }
     Fichier.close();
}
```

```
#include <iostream>    // Pour l'utilisation de cin et cout
#include <fstream>     // Pour l'utilisation des fichiers
#include <cstring>
using namespace std;

struct type_article
{
      char Nom[80];
      double Prix;
};

void Saisir(type_article &Article)
{
      cout << "Article: ";
      cin.getline(Article.Nom,80);
      cout << "Prix: ";
      cin >> Article.Prix;
      cin.ignore(); // Pour enlever le ENTER
      cout << endl;
}

void main (void)
{
   fstream Fichier;   // Fichier permettant de lire et ecrire
   type_article Article;
   char Nom[80];
   int i, Pos;

   // Creation du fichier.
   Fichier.open("ARTICLE.BIN",ios::binary|ios::out);

   // Inscription de quatre articles dans le fichier
   for (i=0; i<4; i++)
   {
         Saisir(Article);
         Fichier.write( (char*) &Article, sizeof(type_article));
   }

   Fichier.close();
   Fichier.open("ARTICLE.BIN",ios::binary|ios::in|ios::out);

   cout << "Nom de l'article dont le prix est a modifier: ";
   cin.getline(Nom,80);

   // Rechercher dans le fichier l'article à modifier
   Fichier.read((char *) &Article, sizeof(type_article));
   while (strcmp(Nom,Article.Nom)!=0)
         Fichier.read((char *) &Article, sizeof(type_article));

   cout << "Le prix de " << Article.Nom << " est " << Article.Prix;
   cout << endl << "Nouveau prix: ";
   cin  >> Article.Prix;

   // Positionnement dans le fichier
   Pos = Fichier.tellg()/sizeof(type_article);
   // Reculer d'une position
   Fichier.seekp((Pos-1)*sizeof(type_article),ios::beg);
   Fichier.write((char *) &Article, sizeof(type_article));
```

```
    // Se positionner au debut et afficher le contenu du fichier
    cout << "Contenu du fichier" << endl;
    Fichier.seekg(0,ios::beg);
    Fichier.read((char *) &Article, sizeof(type_article));
    while (!Fichier.eof())  // Autre boucle permettant de lire le fichier
    {
       cout << Article.Nom << ' ' << Article.Prix << endl;
       Fichier.read((char *) &Article, sizeof(type_article));
    }
    Fichier.close();
}
```

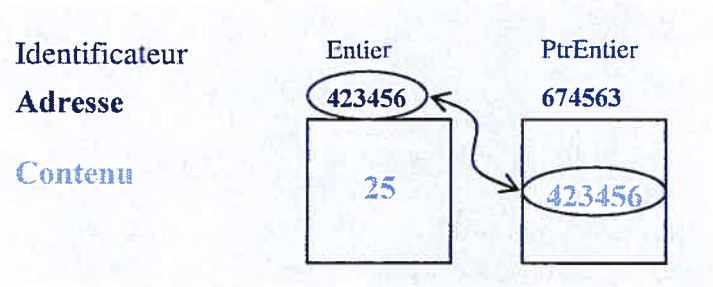
10. Allocation dynamique et Pointeur

a. Définition

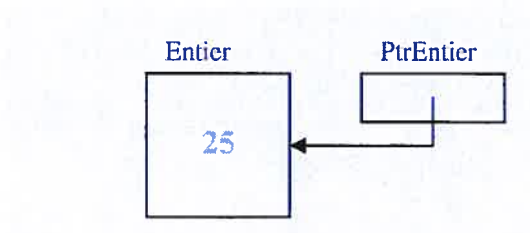
- Une variable entière est une variable qui contient un entier. Une variable caractère est une variable qui contient un caractère. Une variable pointeur est une variable qui contient l'adresse mémoire d'une donnée qui peut être le contenu d'une autre variable.
- Utile lorsqu'une grande quantité de données est nécessaire pour un programme et que la quantité de données à garder simultanément en mémoire fluctue continuellement durant l'exécution.
- Déclaration:
  À la déclaration, la variable pointeur se distingue des autres variables par l'ajout d'un astérisque devant l'identificateur. Par exemple,
  *int *PtrEntier;*
- L'opérateur & «adresse de» est utilisé pour connaître l'adresse d'une donnée.

```
int Entier=25;
int *PtrEntier ;
PtrEntier = &Entier;
```

En supposant que la variable Entier est située à l'adresse 423456 en mémoire, l'affectation *PtrEntier = &Entier;* attribue cette valeur à la variable PtrEntier. La figure suivante illustre cette affectation.
[on peut rendre accessible la figure via un bouton]



Généralement l'illustration utilisée pour représenter cette dernière affectation est :



Distinction entre contenu et adresse

Similarité entre la syntaxe et l'illustration
Opérateur new et delete

new
L'opérateur new est utilisé pour obtenir un espace mémoire lors de l'exécution du programme. Par exemple,

```
int *PtrEntier;

PtrEntier = new int;
```

pour stocker une donnée à l'emplacement obtenu on utilise l'instruction :
`*PtrEntier = 5;`

Dans le cas de l'enregistrement souvent utilisé dans le contexte de l'allocation dynamique l'opérateur -> est utilisé. Par exemple,

```
struct type_breuvage
{
   char Nom[80];
   int Volume;
   double Cout;
};
```

b. Analogie

Surnom

En cliquant sur le nom ou le surnom la même image est affichée

Garou -  - Pierre Garand

Whoopie Goldberg –  - Caryn Johnson

Yves Montand –  - Ivo Livi

Lewis Caroll –  - Charles Lutwidge Dodgson

Bono –  -Paul David Hewson

David Bowie –  - David Robert Jones

Enya –  - Eithne Ni Bhraonian

Elton John –  - Reginald Kenneth Dwight

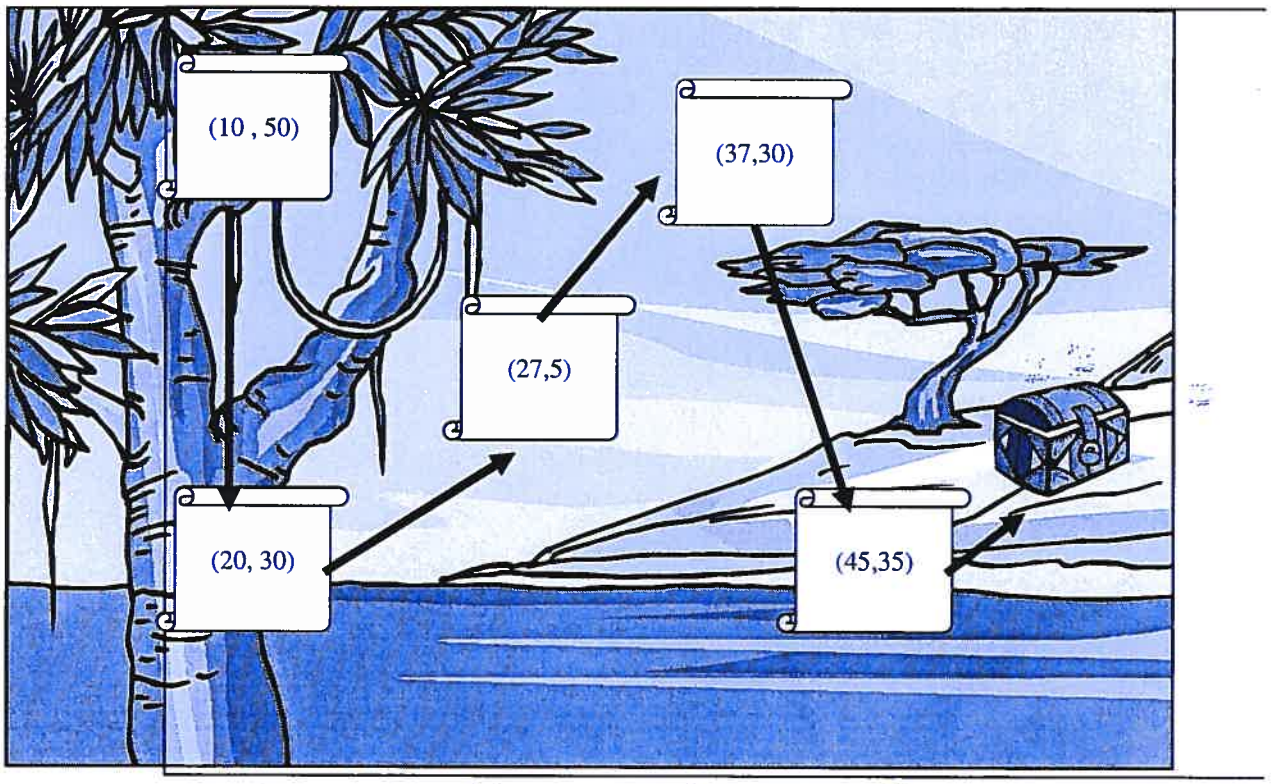
Satchmo –  - Louis Armstrong

Sting –  Gordon Summer

Adresse civique

Rallye automobile
Course au trésor

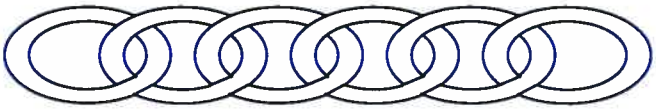
Une course au trésor se déroule de la façon suivante : un premier indice est donné au participant afin de lui permettre de trouver le deuxième indice, ce deuxième indice indique l'emplacement du troisième indice, une fois trouvé le troisième indice dévoile l'emplacement du quatrième indice, le quatrième indice indique l'emplacement du cinquième indice qui finalement précise l'emplacement du trésor.



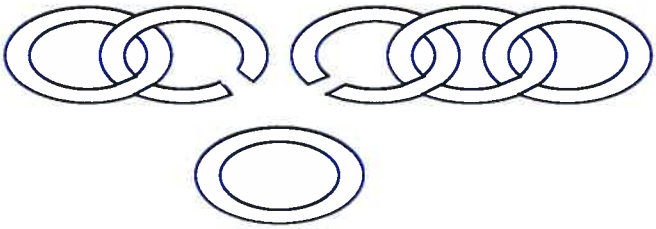
[Au départ le seul indice visible est le premier. Pour voir afficher le prochain indice il faut qu'il tape ces coordonnées ou qu'il clique sur l'icône de l'indice.]

Chaîne (collier)

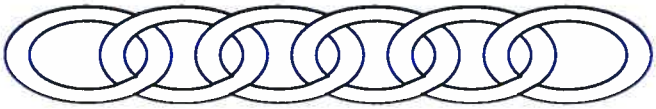
Manipulation pour ajouter ou retirer un anneau du collier



Retirer un anneau de la chaîne



Ajouter un anneau à la chaîne



c. Exercices

Déclaration et initialisation

- Donner la déclaration et les instructions qui permettront d'attribuer dynamiquement un espace mémoire
  - à un entier qui vaudra initialement 23257
  - à un enregistrement déclaré type_poste {char Titre[50]; double Salaire; double NbHresSem; }; initialisé à « Jongleur », 450.89$, 55 heures
  - à une chaîne de caractères initialisée à « débordement de la pile »
  - à un nombre réel initialisé à 97.7
  - à un enregistrement déclaré type_clavier {int NbTouches; char Compagnie [45]; bool Qwerty; } ; initialisé à 121 touches, IBM, true

[COMPLÉTER LE PROGRAMME]

- Donner les instructions qui permettront de réaliser l'insertion d'un nouvel élément, appelé chocolat de 45 g, à la tête d'une liste non vide selon les déclarations :

```
struct type_objet
{
  char Nom[50];
  double Poids;
  type_objet * PtrSuivant;
};
type_objet *PtrTete, *PtrNouveau;

Reponse :
PtrNouveau = new type_objet;
strcpy(PtrNouveau->Nom, « Chocolat »);
PtrNouveau->Poids = 45;
PtrNouveau->PtrSuivant = PtrTete->PtrSuivant;
PtrTete=PtrNouveau;
```

- Donner les instructions qui permettront de compter le nombre d'éléments de la liste construite à partir des déclarations suivantes :

```
struct type_piece
{
   char Titre[80];
   double Auteur;
   type_piece * PtrSuivant;
};
type_piece *PtrTete , // Pointeur au premier élément de la liste
            *PtrTampon; //Pointeur à déplacer dans la liste
```

- Donner les instructions qui permettront de retirer le dernier élément d'une liste non vide construite à partir des déclarations suivantes :

```
{
   char Auteur[50];
   char Titre[80];
   double Prix;
   type_livre *PtrSuivant;
};
type_livre *PtrTete,        //Pointeur au premier élément de la liste
           *PtrCourant,     //Pointeur courant lors du parcours
           *PtrPrecedent;   //Pointeur précédant le courant lors du parcours
```

Solution :

```
for (  PtrCourant=PtrTete;
       PtrCourant!=NULL;
       PtrCourant=PtrCourant->PtrSuivant)
PtrPrecedent =PtrCourant;

delete PtrPrecedent;
PtrPrecedent  = NULL;
// une boucle while peut aussi faire l'affaire
```

- Donner les instructions qui permettront d'afficher les articles de la liste dont le prix est inférieur à 10 $. La liste est construite selon les déclarations :

```
struct type_article
{
   char Nom[50];
   char Fournisseur[80];
   double Prix;
   type_article *PtrSuivant;
};
type_article *PtrTete,      //Pointeur au premier élément de la liste
             *PtrCourant;   //Pointeur courant lors du parcours
```

Reponse :

```
for (PtrCourant=PtrTete; PtrCourant!=NULL;
          PtrCourant = PtrCourant->PtrSuivant)
   if(PtrCourant->Prix < 10)
     cout << PtrCourant->Nom << endl;
```

d. Exemple

```
#include <iostream>
using namespace std;

void main()
{
    int *PtrEntier;
    float *PtrReel;
    int Entier = 7;
    float Reel = 1.2345;
    char *Phrase;

    // PtrEntier est une référence à la variable Entier
    PtrEntier = &Entier;
   *PtrEntier += 10;
   cout << "Entier vaut " <<  Entier << endl;
   cout << "*PtrEntier vaut " << *PtrEntier  << endl << endl;

   // PtrEntier réfère à une adresse mémoire distincte
   PtrEntier = new int;
   *PtrEntier = 25;
   cout << "Entier vaut " <<  Entier << endl;
   cout << "*PtrEntier vaut " << *PtrEntier  << endl << endl;

   // PtrReel est une référence à la variable Reel
   PtrReel = &Reel;
   Reel += 5 * (*PtrReel);
   cout << "Reel vaut " << Reel << endl;
   cout << "*PtrReel vaut " << *PtrReel << endl << endl;

   // PtrReel réfère à une adresse mémoire distincte
   PtrReel = new float;
   *PtrReel = 17.25;
   cout << "Reel vaut " << Reel << endl;
   cout << "*PtrReel vaut " << *PtrReel << endl << endl;

   // Phrase nécessite l'attribution d'un espace mémoire pour les caractères
   // à mémoriser
   Phrase = new char[50];
   strcpy(Phrase, "Un tien vaut mieux que deux tu l'auras");
   cout << "Phrase vaut " << Phrase << endl;

}
```

```
#include <iostream>
using namespace std;

void main()
{
   char *PtrUn, *PtrDeux, *PtrTampon;
   char Lettre;


   PtrUn = new char;
   *PtrUn = 'A';

   PtrDeux = new char;
   *PtrDeux = 'Z';

   // Permutation des adresses de PtrUn et PtrDeux
   cout << "Adresse de PtrUn: " <<  PtrUn  << " de PtrDeux: " <<  PtrDeux
        << endl;
   PtrTampon = PtrUn;
   PtrUn = PtrDeux;
   PtrDeux = PtrTampon;
   cout << "Adresse de PtrUn: " << PtrUn  << " de PtrDeux: " <<  PtrDeux
        << endl;

   // Permutation des contenus référés par PtrUn et PtrDeux
   cout << "Adresse de PtrUn: " << PtrUn << " de PtrDeux: " << PtrDeux
        << endl;
   Lettre = *PtrUn;
   *PtrUn = *PtrDeux;
   *PtrDeux = Lettre;
    cout << "Adresse de PtrUn: " << PtrUn << " de PtrDeux: " << PtrDeux
         << endl;

     delete PtrUn;
     delete PtrDeux;


}
```

```
#include <iostream> // Pour cin et cout
#include <cstdlib> // Pour NULL
#include <cctype> // Pour toupper()
using namespace std;

struct type_vehicule
{
   char Nom[51];
   type_vehicule *PtrSuivant;
};

void main()
{
   type_vehicule *PtrTete;
   type_vehicule *PtrNouveau;
   type_vehicule *PtrCourant, *PtrPrec;
   char Reponse;

   // Creation de la liste
   PtrTete=NULL;
   do
   {
      PtrNouveau = new type_vehicule;
      cout << "Nom: ";
      cin.getline(PtrNouveau->Nom,51);
      PtrNouveau->PtrSuivant= PtrTete; // instructions
      PtrTete = PtrNouveau; // de liaison
      cout << "Un autre vehicule (O/N) : ";
      cin >> Reponse;
      cin.ignore();
   }
   while (toupper(Reponse)=='O');

   //Affichage du contenu de la liste
   PtrCourant = PtrTete;
   while (PtrCourant != NULL)
   {
      cout << PtrCourant->Nom << endl;
      PtrCourant = PtrCourant->PtrSuivant;
   }

   //Destruction de la liste
   PtrCourant = PtrTete;
   while (PtrCourant!=NULL)
   {
      PtrPrec = PtrCourant;
      PtrCourant = PtrCourant->PtrSuivant;
      delete PtrPrec;
      PtrPrec = NULL;
   }
   PtrTete=NULL;
}
```

```
#include <iostream> // Pour cin et cout
#include <cstdlib> // Pour NULL
#include <cctype> // Pour toupper()
using namespace std;

struct type_vehicule
{
   char Nom[51];
   type_vehicule *PtrSuivant;
};

void CreerListe( type_vehicule * & PtrPremier)
{
   type_vehicule *PtrNouveau;
   char Reponse;

   PtrPremier=NULL;
   do
   {
      PtrNouveau = new type_vehicule;
      cout << "Nom: ";
      cin.getline(PtrNouveau->Nom,51);
      PtrNouveau->PtrSuivant= PtrPremier; // instructions
      PtrPremier = PtrNouveau; // de liaison
      cout << "Un autre vehicule (O/N) : ";
      cin >> Reponse;
      cin.ignore();
   }
   while (toupper(Reponse)=='O');
}

void AfficherListe(type_vehicule *PtrPremier)
{
   type_vehicule *PtrCourant;

   cout << endl << "Le contenu de la liste est:" << endl;
   PtrCourant = PtrPremier;
   while (PtrCourant != NULL)
   {
         cout << PtrCourant->Nom << endl;
         PtrCourant = PtrCourant->PtrSuivant;
   }
   cout << endl;
}
```

```
bool RetirerVehicule(type_vehicule *&PtrPremier)
{
      type_vehicule *PtrCourant=PtrPremier,
                    *PtrPrec=NULL;
      char NomCherche[51];
      bool Reussi= true;

      cout << "Le vehicule a retirer: ";
    cin >> NomCherche;
    //Rechercher le véhicule dans la liste
    PtrCourant = PtrPremier;
    PtrPrec=NULL;
    while (PtrCourant!=NULL &&
             strcmp(PtrCourant->Nom,NomCherche)!=0)
      {
       PtrPrec=PtrCourant;
       PtrCourant = PtrCourant->PtrSuivant;
      }
      if ( PtrCourant!=NULL )
      {
       if (PtrPrec!=NULL) // n'est pas à la tête
          PtrPrec->PtrSuivant=PtrCourant->PtrSuivant;
       else // est à la tête
          PtrPremier = PtrPremier->PtrSuivant;
       delete PtrCourant;
       PtrCourant = NULL;
      }
    else
         Reussi =false;

      return Reussi;
}

void DetruireListe(type_vehicule * &PtrPremier)
{
   type_vehicule *PtrCourant, *PtrPrec;
   PtrCourant = PtrPremier;
   while (PtrCourant!=NULL)
   {
      PtrPrec = PtrCourant;
      PtrCourant = PtrCourant->PtrSuivant;
      delete PtrPrec;
      PtrPrec = NULL;
   }
   PtrPremier=NULL;
}
```

```
void main()
{
   type_vehicule *PtrTete;


   // Creation de la liste
   CreerListe(PtrTete);

   //Affichage du contenu de la liste
   AfficherListe(PtrTete);

   if (RetirerVehicule(PtrTete))
         cout << "Retrait reussi" << endl;
   else
          cout << "Le vehicule a retirer est absent de la liste"
             << endl;

   AfficherListe(PtrTete);


   //Destruction de la liste
   DetruireListe(PtrTete);

}
```

ANNEXE D

Le questionnaire PROFIL INITIAL

Question 1
Quel est le niveau d'importance que vous accordez au cours ING1025 dans votre formation?

- ☐ a. Première priorité
- ☐ b. Priorité élevée
- ☐ c. Priorité milieu
- ☐ d. Basse priorité

Enregistrer la réponse

Question 2
Avant mon inscription au cours ING1025 je connaissais le langage de programmation

- ☐ a. Aucun
- ☐ b. BASIC
- ☐ c. C
- ☐ d. C++
- ☐ e. PASCAL

Enregistrer la réponse

Question 3
Quel est le langage le plus connu, le mieux maîtrisé?

- ☐ a. Aucun
- ☐ b. BASIC
- ☐ c. C
- ☐ d. C++
- ☐ e. PASCAL

Enregistrer la réponse

Question 4
Quel est votre niveau de maîtrise du langage le plus connu?

- a. Aucun
- b. Excellent programmeur
- c. Programmeur au dessus de la moyenne
- d. Programmeur dans la moyenne
- e. Programmeur en dessous de la moyenne

Enregistrer la réponse

Question 5
Je préfère résoudre les travaux individuellement plutôt qu'avec d'autres étudiants en équipe.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse
Question 5
Je m'attends à rencontrer certaines difficultés dans ce cours.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse
Question 7
Je considère que l'informatique est un sujet d'étude fascinant.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord

- [ ] d. Fortement d'accord

Enregistrer la réponse

Question 8

Je suis excellent dans la mémorisation des faits.

- [ ] a. Fortement en désaccord
- [ ] b. En désaccord
- [ ] c. D'accord
- [ ] d. Fortement d'accord

Enregistrer la réponse

Question 9

J'aime l'idée de travailler en équipe sur un projet.

- [ ] a. Fortement en désaccord
- [ ] b. En désaccord
- [ ] c. D'accord
- [ ] d. Fortement d'accord

Enregistrer la réponse

Question 10

J'anticipe à être stimulé par les nouvelles idées proposées dans cette classe.

- [ ] a. Fortement en désaccord
- [ ] b. En désaccord
- [ ] c. D'accord
- [ ] d. Fortement d'accord

Enregistrer la réponse

Question 11

J'anticipe de découvrir mes intérêts concernant certains défis de l'informatique dans cette classe.

- [ ] a. Fortement en désaccord
- [ ] b. En désaccord
- [ ] c. D'accord

- d. Fortement d'accord

Enregistrer la réponse

Question 12

Je m'attends à travailler ardemment dans ce cours.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 13

Lors d'un travail de groupe d'étudiants sur un projet de programmation, j'anticipe que mon rôle sera:

- a. Leader actif aidant à organiser les activités du groupe
- b. Participant réalisant sa part du travail
- c. Participant réalisant sa part du travail avec l'aide des coéquipiers
- d. Désintéressé du groupe, mais réalise sa part du travail

Enregistrer la réponse

Question 14

Quel sentiment éprouvez vous lorsqu'on sollicite votre en aide pour un travail de programmation?

- a. Je l'ai déjà fait et généralement j'aime bien
- b. On n'a pas encore sollicité mon aide, mais je pense que j'aimerai cela
- c. J'essaie, mais je ne suis pas toujours d'une bonne aide
- d. L'idée ne me plait pas et j'évite cela

Enregistrer la réponse

Question 15

Je suis:

- a. Femme
- b. Homme

Enregistrer la réponse

Question 16

Le nombre de cours concernant l'informatique que j'ai suivis et réussis au collège est :

- a. Aucun
- b. Un
- c. Deux
- d. Trois
- e. Quatre
- f. Cinq et +

Enregistrer la réponse

Question 17

Le nombre de cours concernant l'informatique que j'ai suivis et réussis au secondaire est :

- a. Aucun
- b. Un
- c. Deux
- d. Trois
- e. Quatre
- f. Cinq et +

Enregistrer la réponse

Question 18

Le nombre de cours au collégial dont la méthode d'enseignement se basait principalement sur le travail en petit groupe est

- a. Aucun
- b. Un
- c. Deux
- d. Trois
- e. Quatre
- f. Cinq et +

Enregistrer la réponse

Question 19

Le nombre de cours au secondaire dont la méthode d'enseignement se basait principalement sur le travail en petit groupe est

- a. Aucun
- b. Un
- c. Deux
- d. Trois
- e. Quatre
- f. Cinq et +

Enregistrer la réponse

Question 20

Ma moyenne globale au collégial se situe

- a. - de 50 %
- b. 50 et 69 %
- c. 70 et 89 %
- d. 90 et + %

Enregistrer la réponse

ANNEXE E

Le questionnaire ÉVALUATION DU TUTORIEL

Question 1
L'étudiant se rappellera facilement les commandes qu'il a utilisées depuis la dernière fois où il a travaillé avec le tutoriel.

- ☐ a. Fortement en désaccord
- ☐ b. En désaccord
- ☐ c. D'accord
- ☐ d. Fortement d'accord

Enregistrer la réponse

Question 2
Le tutoriel permet d'arriver à un produit, une réalisation, une réussite, une action qui assure à l'utilisateur qu'il est arrivé à quelque chose.

- ☐ a. Fortement en désaccord
- ☐ b. En désaccord
- ☐ c. D'accord
- ☐ d. Fortement d'accord

Enregistrer la réponse

Question 3
Les séquences d'actions sont uniformisées et constantes dans l'ensemble du tutoriel.

- ☐ a. Fortement en désaccord
- ☐ b. En désaccord
- ☐ c. D'accord
- ☐ d. Fortement d'accord

Enregistrer la réponse

Question 4
Le tutoriel prévoit des raccourcis pour les habitués.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 5
Toutes les rétroactions sont appropriées à l'action immédiate.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Question 6
Les étudiants ont accès aux affichages précédents. Présence d'une commande de renversement «undo» (retour).

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse
Question 7
Le tutoriel permet aux étudiants de travailler à leur rythme.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse
Question 8

Le tutoriel est convivial.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 9

Le temps nécessaire à l'étudiant pour accomplir une tâche est adéquat.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 10

La fonction de chacune des commandes présentes dans la fenêtre initiale est intuitive.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 11

Il est facile de quitter le tutoriel.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 12
Les étudiants peuvent quitter le tutoriel à n'importe quel moment.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 13
La complexité de l'affichage est appropriée au public visé: lisibilité, repérage aisé de l'information.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 14
Les effets sonores et les illustrations sont pertinents et ne sont pas superflus ni inappropriés.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 15
L'information est structurée de façon à faciliter la navigation et l'utilisation.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 16

L'écran est souvent surchargé.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 17

L'espace-écran est balancé, c'est-à-dire que les icones de gestion du logiciel et les menus déroulants prennent une place adéquate, ni trop embarrassante, ni trop absente.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 18

La distinction entre l'espace réservé à l'information spécifique et l'espace réservé au fonctionnement du tutoriel (icônes de menu, aide, etc.) est claire

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 19
La navigation dans le tutoriel est simple. On entend ici les déplacements dans le tutoriel par l'utilisation de la souris et du clavier, boutons (images, symboles, barre de menu, menus déroulants, case à cocher, ascenseur, clarté des boutons et menus).

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse
Question 20
La rapidité d'exécution des commandes est raisonnable (pas d'attente excessive qui incite à délaisser l'application).

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse
Question 21
La disponibilité du tutoriel sur Internet (site web) est un atout certain.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse
Question 22
Le contenu du tutoriel est conforme à celui du cours ING1025.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 23

Le contenu est présenté sous différentes formes. On retrouve différents médias: photographies, vidéos, animations, sons, diaporamas, etc.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 24

La division du contenu en plusieurs unités est justifiée et adéquate.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 25

Le contenu est approprié au public cible.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 26
Le traitement d'une notion à l'aide des quatre rubriques (définition, analogie, exercice et simulation) est approprié.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 27
L'importance accordée dans ce tutoriel à la rubrique DÉFINITION est

- a. pas du tout important
- b. peu important
- c. important
- d. très important

Enregistrer la réponse

Question 28
L'importance que moi j'accorderais à la rubrique DÉFINITION est

- a. pas du tout important
- b. peu important
- c. important
- d. très important

Enregistrer la réponse

Question 29
L'importance accordée à la rubrique ANALOGIE dans ce tutoriel est

- a. pas du tout important
- b. peu important
- c. important
- d. très important

Enregistrer la réponse

Question 30
L'importance que moi j'accorderais à la rubrique ANALOGIE est

- a. pas du tout important
- b. peu important
- c. important
- d. très important

Enregistrer la réponse
Question 31
L'importance attribuée à la rubrique EXERCICES dans le tutoriel est

- a. pas du tout important
- b. peu important
- c. important
- d. très important

Enregistrer la réponse
Question 32
L'importance que moi j'attribuerais à la rubrique EXERCICES est

- a. pas du tout important
- b. peu important
- c. important
- d. très important

Enregistrer la réponse
Question 33
L'importance attribuée à la rubrique SIMULATION dans ce tutoriel est

- a. pas du tout important
- b. peu important
- c. important
- d. très important

Enregistrer la réponse

Question 34
L'importance que moi j'attribuerais à la rubrique SIMULATION est

- a. pas du tout important
- b. peu important
- c. important
- d. très important

Enregistrer la réponse

Question 35
Les types d'interactions utilisés dans le tutoriel sont bien choisis.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 36
L'utilisation faite de chaque type d'interaction est appropriée.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 37
L'intensité des différentes interactions est bien dosée.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 38
Les interactions permettent de soutenir l'attention de l'étudiant.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 39
Les interactions contribuent à nourrir l'intérêt de l'étudiant.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 40
Les interactions contribuent à favoriser les apprentissages de l'étudiant.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 41
En quoi ces interactions sont-elles spéciales, remarquables ou déficientes?

Enregistrer la réponse

Question 42

Rubrique DEFINITION: Le découpage de la notion est logique et adéquat.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 43

Rubrique DEFINITION: La hiérarchisation du contenu facilite le repérage de l'information cherchée.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 44

Rubrique DEFINITION: L'utilisation des onglets pour diviser le contenu est appropriée.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 45

Rubrique DEFINITION: Les exemples complètent bien le texte de la définition.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 46

La Rubrique DEFINITION offre des outils qui seront utiles à l'apprentissage de l'étudiant.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 47

Rubrique ANALOGIE: Les directives ou consignes sont claires et précises.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 48

Rubrique ANALOGIE: Les interactions facilitent l'étude de l'analogie (incitent à la réflexion).

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 49

Rubrique ANALOGIE: L'analogie utilisée présente une bonne similitude de la notion à assimiler.

- a. Fortement en désaccord
- b. En désaccord

- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 50

Rubrique ANALOGIE: La relation à transférer de l'analogue vers la notion à apprendre est évidente.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 51

Rubrique ANALOGIE: Le texte explicatif est clair et précis.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 52

La rubrique ANALOGIE offre des outils qui seront utiles à l'apprentissage de l'étudiant.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 53

Rubrique EXERCICES: Les directives ou consignes sont toujours claires et précises.

- a. Fortement en désaccord
- b. En désaccord

- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 54

Rubrique EXERCICES: Les directives ou consignes apparaissent au moment opportun.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 55

Rubrique EXERCICES: Lors de la réalisation des exercices l'étudiant pourra commettre plusieurs erreurs.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 56

Rubrique EXERCICES: Lors de la réalisation des exercices, l'étudiant pourra commettre plusieurs sortes d'erreurs.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 57
Rubrique EXERCICES: La méthode utilisée pour corriger les exercices est pertinente et efficace.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse
Question 58
Rubrique EXERCICES: Le résultat de la correction est clair.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse
Question 59
Rubrique EXERCICES: Le résultat de la correction oriente l'étudiant vers la correction a effectuée.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse
Question 60
La rubrique EXERCICES offre des outils qui seront utiles à l'apprentissage de l'étudiant.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 61

Rubrique SIMULATIONS: Les différentes vues (programme source, variables en mémoire, explications, écran, etc) offertes sont pertinentes et suffisantes.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 62

Rubrique SIMULATIONS: La longueur des programmes informatiques exécutés lors de la simulation est adéquate.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 63

Rubrique SIMULATIONS: La difficulté des programmes informatiques exécutés lors de la simulation est bien dosée (accessible à l'apprenant).

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 64
Rubrique SIMULATIONS: L'enchaînement des actions dans les différentes vues lors de l'exécution d'une instruction aide à la compréhension du fonctionnement du programme à l'étude.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 65
Rubrique SIMULATIONS: L'enchaînement des actions dans les différentes vues lors de l'exécution d'une instruction du programme est utile et facile à suivre.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 66
Rubrique SIMULATIONS: L'interaction offerte permettra de conserver l'intérêt de l'étudiant.

- a. Fortement en désaccord
- b. En désaccord
- c. D'accord
- d. Fortement d'accord

Enregistrer la réponse

Question 67
La Rubrique SIMULATIONS: offre des outils qui seront utiles à l'apprentissage de l'étudiant.

- ☐ a. Fortement en désaccord
- ☐ b. En désaccord
- ☐ c. D'accord
- ☐ d. Fortement d'accord

Enregistrer la réponse

Question 68
Je crois que le tutoriel sera utile aux étudiants pour l'apprentissage des notions du cours ING1025.

- ☐ a. Fortement en désaccord
- ☐ b. En désaccord
- ☐ c. D'accord
- ☐ d. Fortement d'accord

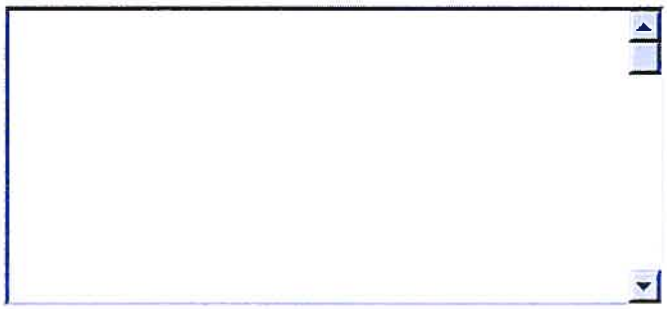
Enregistrer la réponse

Question 69
En tant qu'enseignant du cours ING1025, je pourrais utiliser un certain pourcentage des éléments du tutoriel lors de mon enseignement en classe.

- ☐ a. 0 %
- ☐ b. 1 à 25%
- ☐ c. 26 à 50 %
- ☐ d. 51 et 75 %
- ☐ e. 76 % et plus

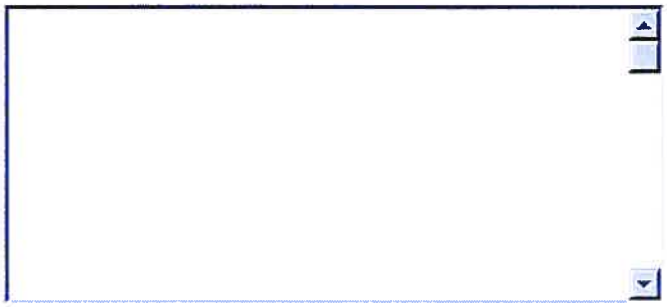
Enregistrer la réponse

Question 70
Si vous pensez pouvoir utiliser certains éléments du tutoriel dans votre enseignement en classe, quels sont-ils et à quel fin les utiliserez-vous?



Enregistrer la réponse

Question 71
Pour quels raisons n'envisagez vous pas d'utiliser le tutoriel ou certains éléments du tutoriel dans votre enseignement en classe?



ANNEXE F

Évaluation de la pertinence de l'atelier

Selon l'échelle de valeur :
(1-fortement en désaccord, 2= en désaccord, 3= d'accord, 4 = fortement d'accord)

1. Les consignes sont claires et précises	
2. Les objectifs sont bien identifiés	
3. La période de temps disponible est suffisante pour la réalisation du travail	
4. J'ai trouvé l'atelier très fascinant et intéressant	
5. J'aurais pu facilement me passer de cet atelier	
6. La principale contribution de mon apprentissage provient de a) mon professur b) un ou une de mes collègues c) moi-même	a) b) c)
7. Cet atelier m'a permis d'explorer les notions suivantes :	
8. Commentaires :	

ANNEXE G

Programmes réalisés lors de l'atelier

```
/*-------------------------------------------------------------------------*/
/* FICHIER:     Programme 1                                                */
/*                                                                         */
/*-------------------------------------------------------------------------*/


#include "stdafx.h"
#include "serial.h"
#include <utilitaires.h>

#include <iostream>
#include <cstring>

using namespace std;

void main()
{
     CSerial Port;

    //Attendre avant de transmettre une commande
     ATTENDRE(100000000);

     if (Port.Open(1,19200))
     {
       //DemarrerArreterMoteur(Port);
       //AllumerEteindreLed(Port);

     // EXERCICE 1
       char sSequence[5]="1234", sTemp[3], sPresente[ 5 ]="", *pdest;

       do
       {
          cin.getline(sTemp,3);

          if (strlen(sPresente)<4)
              strcat(sPresente, sTemp);
          else
          {
              cout<<"Maximum de diode ouvert, reinitialiser?\n";
              cin.getline(sTemp,3);
              strlwr(sTemp);
              if (sTemp[0]=='o')
                  sTemp[0]='-';    //pour reinitialiser sTemp, plus bas
              else
                  break;
              //eteindre tous les lumieres
              EteindreLed(Port,"1");
              Sleep(1000);
              EteindreLed(Port,"2");
              Sleep(1000);
```

```
                EteindreLed(Port,"3");
                 Sleep(1000);
                 EteindreLed(Port,"4");
                 Sleep(1000);
            }

            if (sTemp[0]=='-')              //"-1"
               strcpy(sPresente,"");

            pdest = strstr( sSequence, sPresente );
            if( pdest != NULL && strlen(sPresente)>0)
                  AllumerLed(Port, sTemp);
        }
        while (true);


// EXERCICE 2

        for (int i; i<125; i+=5)
        {
                Activer(Port, i);
                Sleep(1000);
                if (ObtenirVitesse(Port)>1)
                        break;
                Sleep(1000);
        }
        Sleep(1000);
        Activer(Port,0);
        Sleep(1000);
        i-=5;
        for (i=i;i<125;i++)
        {
                Activer(Port, i);
                Sleep(1000);
                if (ObtenirVitesse(Port)>1)
                        break;
                Sleep(1000);
        }
        cout<<" i = "<<i<<endl;
        Activer(Port,i+25);
        for (i=i;i>0;i--)
        {
                Activer(Port, i);
                Sleep(1000);
                if (ObtenirVitesse(Port)<=1)
                        break;
                Sleep(1000);
        }
        cout<<" i = "<<i<<endl;

        }
    else
       cout << "Probleme d'ouverture du port!";
}
```

```
/*-------------------------------------------------------------------------*/
/* FICHIER:     Programme 2                                                */
/*-------------------------------------------------------------------------*/


#include "stdafx.h"
#include "serial.h"
#include <utilitaires.h>

#include <iostream>
#include <cstring>
using namespace std;


void main()
{
      int Reponse;
      bool Valide;


      CSerial Port;

    //Attendre avant de transmettre une commande
      ATTENDRE(100000000);

      if (Port.Open(1,19200))
      {
         //DemarrerArreterMoteur(Port);
         //AllumerEteindreLed(Port);
            do
            {
                  Valide = false;
                  cout << "Quel led voulez-vous ouvrir?" << endl;
                  cin >> Reponse;
                  if (Reponse != 0)
                  {
                        cout << "Desole, mauvais led! Reassayez du
                                debut!" << endl;
                        Valide = true;
                  }
                  else
                  {
                        AllumerLed(Port, "0");
                        cout << "Quel led voulez-vous ouvrir en
                                 deuxième?" << endl;
                        cin >> Reponse;
                        if (Reponse != 2)
                        {
                              cout << "Desole, mauvais led! Reassayez
                                       du debut!" << endl;
                              EteindreLed(Port, "0");
                              Valide = true;
                        }
                        else
                        {
                              AllumerLed(Port, "2");
                              cout << "Quel led voulez-vous ouvrir en
```

```
                troisième?" << endl;
            cin >> Reponse;
            if (Reponse != 3)
            {
                cout << "Desole, mauvais led!
                        Reassayez du debut!"
                     << endl;
                EteindreLed(Port, "0");
                ATTENDRE(100000000);
                EteindreLed(Port, "2");
                Valide = true;
            }
            else
            {
                AllumerLed(Port, "3");
                cout << "Quel led voulez-vous
                        ouvrir en dernier?"
                     << endl;
                cin >> Reponse;
                if (Reponse != 1)
                {
                    cout << "Desole, mauvais led!
                            Reassayez du debut!"
                         << endl;
                    EteindreLed(Port, "0");
                    ATTENDRE(100000000);
                    EteindreLed(Port, "2");
                    ATTENDRE(100000000);
                    EteindreLed(Port, "3");
                    cin >> Reponse;
                }
                else
                    AllumerLed(Port, "1");
            }
                }
            }

        }
        while (Valide);

    }
  else
     cout << "Probleme d'ouverture du port!";
}
```

```
/*-------------------------------------------------------------------------*/
/* FICHIER:     Programme 3                                                */
/*-------------------------------------------------------------------------*/

#include "stdafx.h"
#include "serial.h"
#include <utilitaires.h>

#include <iostream>
#include <cstring>
using namespace std;


void main()
{
      CSerial Port;

    //Attendre avant de transmettre une commande
      ATTENDRE(100000000);
      int Rep;
      char Rep2;
      if (Port.Open(1,19200))
      {

            cout << "Entrer une vitesse...";
            cin >> Rep;


            Activer(Port, Rep);
            ObtenirVitesse(Port);
            ObtenirVitesse(Port);
            ObtenirVitesse(Port);

       //DemarrerArreterMoteur(Port);
       //AllumerEteindreLed(Port);

            int Rep;
            cout << "Quelle LED??";
            cin >> Rep;
            if (Rep == 2)
            {
                  AllumerLed(Port,"2");
                  cout << endl << "Quelle LED??";
                  cin >> Rep;
                  if (Rep == 3)

                  {
                        AllumerLed(Port,"3");
                        cout << endl << "Quelle LED??";
                        cin >> Rep;

                        if (Rep == 0)
                        {
                              AllumerLed(Port,"0");
                              cout << endl << "Quelle LED??";
                              cin >> Rep;
```

```
                                    if (Rep == 1)
                                    {
                                          AllumerLed(Port,"1");
                                          cout << endl << "Quelle LED??";
                                          cin >> Rep;
                                    }
                                    else
                                          cout << "NON!  RECOMMENCE! PAS
                                                  BON!" << endl << endl;

                              }
                              else
                                    cout << "NON!  RECOMMENCE! PAS BON!"
                                         << endl << endl;
                        }
                        else
                              cout << "NON!  RECOMMENCE! PAS BON!"
                                   << endl << endl;
                  }
                  else
                        cout << "NON!  RECOMMENCE! PAS BON!" << endl << endl;


            }

      else
         cout << "Probleme d'ouverture du port!";
}
```

```
/*-------------------------------------------------------------------------*/
/* FICHIER:     Programme 4                                                */
/*-------------------------------------------------------------------------*/

#include "stdafx.h"
#include "serial.h"
#include <utilitaires.h>
#include <ctime>
#include <iomanip>
#include <cctype>
#include <cstdio>
#include <cstdlib>
#include <iostream>
#include <cstring>
using namespace std;


void main()
{
      CSerial Port;

      int sequence[4][1];
      int i = 0;
      char scrap[2];
      char temp;
      int vitesse;

    //Attendre avant de transmettre une commande
      ATTENDRE(100000000);

      if (Port.Open(1,19200))
      {
            //DemarrerArreterMoteur(Port);
            //AllumerEteindreLed(Port);

            //cout << ObtenirVitesse(Port) << endl;;

            i = 40;
            do
            {
                  vitesse = ObtenirVitesse(Port);
                  cout << vitesse << "\t";
                  ATTENDRE(100000000)
                  Activer(Port,i);
                  cout << i << endl;
                  i++;
                  ATTENDRE(100000000);
            }
            while ((vitesse == 0) || (vitesse == 1));


            do
            {
                  vitesse = ObtenirVitesse(Port);
                  cout << vitesse << "\t";
                  ATTENDRE(100000000)
                  Activer(Port,i);
```

```
                cout << i << endl;
                i--;
                ATTENDRE(100000000);
            }
            while ((vitesse != 0) && (vitesse != 1));


        //STAR TREK!!!!!!!!!!
        for (int i = 0; i<10; i++)
        {       AllumerLed(Port,"0");
                ATTENDRE(100000000);
                EteindreLed(Port,"0");
                ATTENDRE(100000000);
                AllumerLed(Port,"1");
                ATTENDRE(100000000);
                EteindreLed(Port,"1");
                ATTENDRE(100000000);
                AllumerLed(Port,"2");
                ATTENDRE(100000000);
                EteindreLed(Port,"2");
                ATTENDRE(100000000);
                AllumerLed(Port,"3");
                ATTENDRE(100000000);
                EteindreLed(Port,"3");
                ATTENDRE(100000000);}
        }
    else
       cout << "Probleme d'ouverture du port!";
}
```

```
/*-------------------------------------------------------------------------*/
/* FICHIER:     Programme 5                                                */
/*-------------------------------------------------------------------------*/



#include "stdafx.h"
#include "serial.h"
#include <utilitaires.h>

#include <iostream>
#include <cstring>
using namespace std;

void FonctionLED(CSerial &Port)
{
      char Led[4][3], i=0, entree[3];
      bool rate=false;
      bool trouver = false;

      strcpy(Led[0],"1");
      strcpy(Led[1],"3");
      strcpy(Led[2],"0");
      strcpy(Led[3],"2");
      while(!trouver)
      {
         do
         {
            cout << "Entrer le numero de la led : (0,1,2,3)" << endl;
            cin >> entree;
            if (strcmp(entree,Led[i])==0)
            {
               cout << "BRAVO, plus que " << 3-i << " led a allumer!"
                    << endl;
               i++;
               AllumerLed(Port, entree);
               if(i==4) trouver=true;
            }
            else
            {
               rate=true;
               cout << "essayez encore" << endl;
            }

         }
         while(i<4 && !rate);
      }
      cout << "Vous avez trouve l'ordre d'ouverture: "
          <<Led[0]<<Led[1]<<Led[2]<<Led[3];
}

void TrouverVitesse(CSerial &Port)
{

      int i;
      int ImpulsionDepart, ImpulsionArret;
      int VDepart,VArret,VTemp;
```

```
	bool DepartTrouver=false, ArretTrouver=false;
	int TempAttente = 100000000;
	ATTENDRE(TempAttente);
	for(i=40;i<128 && !DepartTrouver;i++)
	{
		cout << "Recherche Impulsion Depart" << endl;
		cout << "Impulsion " << i << endl;

		Activer(Port,i);
		ATTENDRE(TempAttente);
		VTemp = ObtenirVitesse(Port);
		ATTENDRE(TempAttente);

		if(VTemp>1)
		{
			VDepart = VTemp;
			ImpulsionDepart = i;
			DepartTrouver = true;

		}
	}

	ATTENDRE(TempAttente);

	for(i=ImpulsionDepart;i>=0 && !ArretTrouver;i--)
	{
		cout << "Recherche Impulsion Arret" << endl;
		cout << "Impulsion " << i << endl;
		Activer(Port,i);
		ATTENDRE(TempAttente);
		VTemp = ObtenirVitesse(Port);
		ATTENDRE(TempAttente);

		if(VTemp<=1)
		{
			VArret = VTemp;
			ImpulsionArret = i;
			ArretTrouver = true;
		}
	}

	cout << "Impulsion de depart: " << ImpulsionDepart << endl;
	cout << "Impulsion de arret: " << ImpulsionArret << endl;



}

void main()
{
	CSerial Port;


    //Attendre avant de transmettre une commande
	ATTENDRE(100000000);
```

```
        if (Port.Open(1,19200))
        {
              TrouverVitesse(Port);
              //FonctionLED(Port);
             //DemarrerArreterMoteur(Port);
             //AllumerEteindreLed(Port);
        }
        else
          cout << "Probleme d'ouverture du port!";
}
```

```
/*-------------------------------------------------------------------------*/
/* FICHIER:     Programme 6                                                */
/*-------------------------------------------------------------------------*/

#include "stdafx.h"
#include "serial.h"
#include <utilitaires.h>

#include <iostream>
#include <cstring>
using namespace std;

void main()
{
      CSerial Port;
      //bool zero,un,deux,trois;
      //int reponse;


    //Attendre avant de transmettre une commande
      ATTENDRE(100000000);

      if (Port.Open(1,19200))
      {
         //DemarrerArreterMoteur(Port);
         //AllumerEteindreLed(Port);

         //créer ordre

         zero=false;
         un=false;
         deux=false;
         trois=false;

         do
         {
            //ordre 3102

            cout <<"Entrez le led a allumer:    ";
            cin >> reponse;

            if(reponse==3)
            {
                  AllumerLed(Port,"3");
                  un=true;
            }
            if(reponse==1 && un)
            {
                  AllumerLed(Port,"1");
                  zero=true;
            }
            if(reponse==0 && zero)
            {
                  AllumerLed(Port,"0");
                  deux=true;
            }
            if(reponse==2 && deux)
```

```
                {
                        AllumerLed(Port,"2");
                }

        }
        while(reponse!=-1);



//moteur

                int k=1,vitesse;
                do
                {
                        k++;
                        Activer(Port,k);
                        cout <<k;
                        ATTENDRE(200000000);
                        vitesse=ObtenirVitesse(Port);
                        cout <<" "<<vitesse <<endl;

                        ATTENDRE(200000000);

                }while(vitesse==1 || vitesse==0);

                cout <<"Depart a " <<k;


      }
    else
       cout << "Probleme d'ouverture du port!";
}
```