Introduction

INTRODUCTION 1

QU'ALLONS-NOUS FAIRE CETTE ANNEE ? 1
QU'EST-CE QUE FAIRE DE L'ALGORITHMIQUE ? 1
QU'EST-CE QUE PROGRAMMER ? 3

L'ORDINATEUR EST-IL INTELLIGENT ? 5

LES LANGAGES 5

LE LANGAGE C : 5
LES LANGAGES : UNE HISTOIRE DE NIVEAU 6
LE NIVEAU 0 : LE MICROPROCESSEUR 7
LE NIVEAU 1 : L'ASSEMBLEUR 8
LE NIVEAU 2 : FORTRAN, PASCAL, C VS LISP ET PROLOG 10
LE NIVEAU 3 : C++, JAVA 11
LE NIVEAU 4 : LES GENERATEURS DE CODE 11
LE NIVEAU 5 : LES SOLVEURS DE PROBLEME 12
LE NIVEAU 6 : L'INTELLIGENCE ARTIFICIELLE 13

LES SYSTEMES D'EXPLOITATION : UNE PYRAMIDE DE NIVEAUX 14

Introduction

Avant de nous lancer à la découverte de l'algorithmique et de la programmation en langage C, qui sont deux termes couvrant un vaste domaine, il est important de situer le contexte dans lequel nous allons évoluer.

Nous allons donc dans un premier temps définir les termes d'algorithmique et de langage de programmation, en nous basant sur un exemple, puis nous décrirons la manière dont est organisé l'enseignement de l'algorithmique et du langage C, notamment en précisant le rôle :

- Des différents supports qui vous seront attribués
- Des différents intervenants avec qui vous travaillerez
- Et enfin, le dernier mais non le moindre, le vôtre, car c'est déjà votre vie d'ingénieur que vous préparez.

Qu'allons-nous faire cette année ?

Nous allons essentiellement faire deux choses : de l'algorithmique et de la programmation en langage C. Pour que les choses soient claires dès maintenant, allons un peu plus loin dans le détail :

Qu'est-ce que faire de l'algorithmique ?

algorithme : Ensemble des règles opératoires qui permettent la résolution d'un problème par l'application d'un nombre fini d'opérations de calcul à exécuter en séquence.

Faire de l'algorithmique, c'est trouver, en réponse à un problème simple, une solution que l'on peut traduire très facilement en un programme. Il s'agit, la plupart du temps, de décrire la solution sous la forme d'actions élémentaires qui se déroulent selon un certain ordre. Il n'y a absolument pas besoin de connaître un langage de programmation précis pour arriver à une solution algorithmique, dite aussi algorithme.

Cela a pour avantage que l'on peut traduire un algorithme dans le langage que l'on choisit, et que l'algorithme est indépendant du langage de programmation.

Illustration avec le problème des récipients :

Vous devez mesurer exactement 4 litres d'eau Pour cela, vous ne disposez que de deux récipients : un récipient de 5 litres et un deuxième de 3 litres, et d'un robinet. Les actions que vous pouvez utiliser pour obtenir une solution sont :

Remplir un récipient au robinet, vider un récipient (à l'égout), et transvaser un récipient dans un autre (lorsque l'on transvase, on s'arrête lorsque le récipient qui reçoit l'eau est plein, et on garde le restant dans le récipient à partir duquel on verse).

La solution algorithmique est la suivante, on y ajoute des commentaires pour expliquer plus clairement la solution. Les commentaires sont notés entre parenthèses.

```
Remplir le récipient de 5 litres

Transvaser le récipient de 5 litres dans celui de 3 litres

(il reste donc 2 litres dans le récipient de 5 litres)

Vider le récipient de 3 litres

Transvaser le récipient de 5 litres dans le récipient de 3 litres

(le récipient de 5 litres est vide, celui de 3 litres contient 2
litres)

Remplir le récipient de 5 litres

Transvaser le récipient de 5 litres dans le récipient de 3 litres

(on ne met donc qu'1 litre dans le récipient de 3 litres, il en
reste donc 4 dans le récipient de 5 litres, c'est bien ce que l'on
voulait).
```

La présence de commentaire ne change rien à l'algorithme en lui-même, mais permet de mieux comprendre la manière dont il fonctionne.

Qu'est-ce que programmer ?

Programmer : concevoir, écrire, modifier et tester un programme à l'aide d'un langage de programmation.

Programmer, c'est avant tout traduire un algorithme en un langage de programmation, dont les règles d'écriture (ou règles de syntaxe) sont assez strictes, mais qui permet d'obtenir un résultat concret par un ordinateur (ou un système).

On écrit, à l'aide d'un éditeur de texte le plus souvent, l'ensemble des actions élémentaires en respectant ces règles syntaxiques, et l'on obtient ce qu'on appelle le code source. Le code source doit en général être traduit en un langage plus proche de la machine (le langage machine) à l'aide d'un compilateur. Cet outil de traduction transforme le code source en un programme exécutable.

Pour écrire le code source, il est indispensable de bien connaître les règles de la syntaxe du langage qui est utilisé. On trouve l'ensemble de ces règles dans des documentations à caractère essentiellement technique. Il ne faut donc pas chercher dans ce type de documentation comment on va résoudre un problème, puisque cette résolution appartient plutôt au domaine de l'algorithmique. Par contre, lorsque l'on voudra traduire l'algorithme en langage C par exemple, ce document servira de référence.

Je vous propose de continuer l'exemple traité lors du paragraphe précédent, le problème des récipients. On veut maintenant programmer un bras-robot (voir ci-contre) afin qu'il réalise les différentes opérations. Le langage (imaginaire) utilisé est le LMR (Langage de Manipulation de Robot).

Figure 1 : bras robot articulé

Voici, toujours à titre d'exemple, une documentation de ce langage, à partir de laquelle on va écrire le programme à partir de l'algorithme.

LMR – © 2004 EFREI

Syntaxe d'un programme LMR :

Tout programme doit débuter par : #PROG et se terminer par #END

#PROG doit être suivi d'un nom que l'on donne au programme, ce nom ne doit comporter que des lettres en minuscule.

Tout objet manipulé par le bras doit porter un nom et doit être défini par la syntaxe :

.object : , suivi du nom de l'objet. le nom d'un objet ne doit comporter que des lettres minuscules. Tout objet doit être défini dans le programme avant les commandes.

commandes du langage :

REMPLIR, suivi du nom d'un objet entre parenthèses, permet de remplir un objet

VIDER, suivi du nom d'un objet entre parenthèses, permet de vider un objet

TRANS, suivi, entre parenthèses, du nom de deux objets séparés par un %, permet de transvaser le contenu du premier objet listé dans le deuxième.

Toute commande du langage doit être suivie du caractère ':'.

A partir de ces règles et de l'algorithme, on peut donc écrire le programme de manipulation du robot :

```
#PROG quatrelitres
.object : recipcinq
.object : reciptrois
REMPLIR(recipcinq):
TRANS(recipcinq % reciptrois):
VIDER(reciptrois):
TRANS(recipcinq % reciptrois):
REMPLIR(recipcinq):
TRANS(recipcinq % reciptrois):
#END
```

L'ordinateur est-il intelligent ?

Sans ambiguïté, nous pouvons répondre NON ! à cette question. Le programme traité en exemple dans le paragraphe précédent le montre d'ailleurs bien : le problème est résolu par un algorithme, et l'ordinateur est incapable de mettre au point tout seul cet algorithme, on doit donc lui expliquer ce qu'il faut faire.

De plus, même à partir de l'algorithme, le problème n'est pas totalement traité : il faut encore traduire cet algorithme dans un langage que l'ordinateur comprend : on doit lui expliquer comment faire dans les moindres détails.

Un ordinateur n'a aucune connaissance, tout ce qu'il sait faire, c'est calculer avec des valeurs numériques (un ordinateur ne sait pas résoudre des équations, il faut une fois de plus lui fournir une méthode), et de ce point de vue, il est imbattable. Mais le fait qu'il calcule vite ne veut pas dire qu'il calcule bien…En fait, il ne comprend que des 0 et des 1. Il ne sait compter que sur…un seul doigt !

Les langages

Les premiers ordinateurs, datant des années 40, étaient à l'époque conçus pour montrer la faisabilité d'une telle machine à des fins de calcul, et non pour les performances, ou pour une grande diffusion. Son emploi était réservé à quelques spécialistes, qui se concentraient alors sur la tâche essentielle : faire des calculs. Il n'y avait à l'époque ni clavier ni écran sophistiqués. Les entrées / sorties se faisaient alors manuellement, avec des supports assez variés : des cartes perforées, des fiches de relais téléphoniques; on est tout de même arrivé assez rapidement à l'utilisation d'un clavier avec lequel on entrait les programmes et les données sous formes de chiffres.

Le langage C :

Dérivé du langage B (et oui, parfois il ne faut pas chercher des explications trop compliquées !), le langage C est lié à la conception du système UNIX par les laboratoires Bell. Les langages ayant influencé son développement sont :

- le langage BCPL de M. Richards (1967) ;
- le langage B développé aux laboratoires Bell (1970).

Ces deux langages partagent avec le langage C quelques points communs, mais étaient loin d'être aussi aboutis. Notamment, ces langages étaient basés sur des considérations techniques très proche de la machine : on ne pouvait pas avoir de programmes portables, c'est à dire que l'on peut les exécuter sur des machines différentes : il fallait réécrire le programme pour l'adapter à chaque famille d'ordinateurs. Les auteurs du langage C ont, entre autres, pris cela en compte.

Les dates marquantes de l'histoire du langage C sont les suivantes :

- 1970 diffusion de la famille PDP 11.
- 1971 début du travail sur le langage C, car le PDP 11 peut manipuler un octet alors que son mot mémoire est de 2 octets, il est nécessaire pour utiliser les fonctionnalités du PDP11 introduire un type de donnée char et un type int. Ces notions de type n'étant pas prises en compte par le langage B, elles le seront dans son successeur le C.
- 1972 la première version de C est écrite en assembleur par Brian W. Kernighan et Dennis M. Ritchie.
- 1973 Alan Snyder écrit un compilateur C portable (thèse MIT).
- 1975 Steve C. Johnson écrit et présente le PCC (Portable C Compiler). C'était à l'époque le compilateur le plus répandu. Le PCC a longtemps assuré sa propre norme puisqu'il était plus simple de porter le PCC que de réécrire un compilateur C (25 % du code du PCC à modifier).
- 1987 début de la normalisation du langage par l'IEEE avec constitution d'un comité appelé : X3 J-11.
- 1989 sortie du premier document normalisé appelé norme ANSI X3-159.

Compilateur ? octets ? type char ? Ces mots ne font pas encore partie de notre vocabulaire, mais nous apprendrons assez rapidement ce qu'ils signifient.

Les langages : une histoire de niveau

Pour bien comprendre ce que l'on peut faire et ce que l'on ne peut pas faire en informatique avec tel ou tel langage, il nous faut repartir du niveau 0 de ce qu'est un ordinateur. Un langage est avant tout un moyen de communiquer, et selon ce que l'on peut décrire avec ce langage, la communication sera plus ou moins riche et porteuse

d'informations. Les langages les plus riches sont les langages dits naturels, ceux que nous utilisons pour communiquer, tous les jours ou presque. La richesse des informations transmises par un langage dépend de l'étendue du vocabulaire proposé mais aussi de la manière dont on peut structurer les éléments du langage. Ainsi, le français par exemple, dispose d'un vocabulaire très étendu qui permet de parler d'abstractions complexes (du genre de celles que l'on traite pendant les cours de philosophie), et d'une structuration également complexe.

En informatique, le vocabulaire et les éléments de structuration dépendent du niveau auquel on se place, par rapport au matériel qui exécute toutes les tâches demandées

Le niveau matériel

Le vocabulaire de base d'un ordinateur est composé de deux mots : 0 et 1. Cela correspond en réalité à deux niveaux électriques, car les ordinateurs sont basés sur des transistors qui agissent comme des interrupteurs : ils sont ouverts ou fermés, il n'existe pas d'état intermédiaire.

Le niveau 0 : le microprocesseur

Le microprocesseur (ou unité centrale) est souvent comparé au cerveau de l'ordinateur. Cette comparaison n'est pas très pertinente, car nous avons vu qu'un ordinateur n'était pas intelligent, et nous ne sommes pas près de voir une machine avoir des idées ! On peut plutôt le considérer comme un chef d'orchestre, qui utilise les autres éléments de l'ordinateur pour mener à bien les tâches qui lui sont confiées. Cependant, son vocabulaire est une fois de plus très limité, il ne comprend des ordres que sous la forme de suites de 0 et de 1, et pilote les autres éléments à l'aide d'ordre identiques. Le langage que comprend le microprocesseur est le **langage machine**. Un petit exemple de langage machine :

```
01001101010110101001000000000000000000001100000000000000000000000000
01000000000000000000000000001111111111111110000000000000000010111000
00000000000000000000000000000000000000000000000000000001000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00001000000000000000000000000000000000001110000111111011101000001110
00000000101101000000100111001101001000011011100000000000101001100110
11010010000101010100011010000110100101110011001000000011100000111001
01101111011001110111001001100001011011010010000001100011011000010110
11100110111001101111011101000010000000110001001100101001000000111001
01110101011011100010000001101001011011100010000000100010001001111010
00110010000000110110101101111011001000110010100101110000001101000010
```

```
001001000000000000000000000000000000000000000000000000000000000000101
000001000101000000000000000001001100000000010000001100000000000111001
110100101110000100110101000000000000000000000000000000000000000000000
000000000000000000001110000000000000000011110000000110000101100000001
000000110000101000000000000101000100111000000000000000000111000000000
001000000000000000000000001000000000000000000000000000001000000000000
000000000000000000010000000000000000000000000000001110000010010110000
000000000000000000000000000000110000000000000001000000000000000000000
000000000001000000000000000000000000010000000000000000000000000000000
100000000000000000000000000000000100000000000000000000000000000000000
000000000000000000000000000000000101000001010010000000000000000000000
010000000000000000001110000101011001010100100000000000000001000000000
000000000000000000000000000000000001000000000000000000000000100000000
000000000000000000000000000000010000000000000000000000001000000000000
000000000000000000000000000000000000000000010000000000000000000000000
0000001100000001100101001111..................
```

Ceci est le début du programme WORD utilisé pour la réalisation de ce document. Avec ce langage, il faudrait plusieurs milliers de pages remplies de 0 et de 1 pour décrire le programme complet. Alors, pensez-vous que WORD a été programmé en langage machine ?

Le niveau 1 : l'assembleur

La programmation à l'aide de langage machine est une tâche quasiment impossible, même pour les tous premiers ordinateurs. Afin de simplifier les programmes, la première astuce utilisée a été d'utiliser une autre notation : le langage machine est en fait issu de la notation en base 2, ou base binaire. Or, une suite de 0 et de 1, notée en base 2, peut être comprise comme un nombre. Imaginons un langage à base d'un vocabulaire de 10 mots, et dont les mots correspondraient aux chiffres décimaux que nous utilisons couramment : 0, 1, 2, 3, 4, 5, 6, 7, 8 et 9. Un programme écrit dans un tel langage serait simplement un nombre. On peut donc considérer qu'un programme en langage machine est un nombre écrit en base 2, il suffit donc de le réécrire dans une autre base pour avoir une écriture plus courte. Pour que cette transformation soit facile à faire, la base 16, aussi nommée hexadécimale, a été choisie. Mais on se trouvait toujours en présence d'un programme décrit par une suite de chiffres, ce qui n'aide guère à sa compréhension.

Mais, heureusement, puisque le langage machine permet de manipuler des composants matériels, et que ces derniers ne sont finalement pas très évolués, les ordres qui leur sont donnés ne sont pas tellement diversifiés. Ils savent calculer et déplacer des valeurs. Pourquoi ne pas trouver des noms pour ces opérations, plutôt que de leur associer des nombres ? Il suffirait alors de traduire automatiquement ces noms en chiffres, chose que sait très bien faire une machine (puisque c'est une tâche répétitive et facile). On obtient alors des instructions en assembleur, qui est le langage utilisant des noms au lieu des chiffres pour que l'ordinateur

réalise les opérations les plus élémentaires. Bien entendu, on gagnera du temps au niveau de la rédaction du programme, mais ce langage reste néanmoins assez ardu à utiliser, car c'est toujours au programmeur de faire la démarche consistant à traduire ses problèmes en des instructions qui restent basiques. Faire de l'algorithmique en assembleur nécessite énormément de patience et d'aspirine !

De plus, le langage assembleur étant conçu pour s'adresser directement au microprocesseur, chaque famille de microprocesseur possède un langage assembleur particulier. Ainsi, la portion de programme ci-dessous est écrite en assembleur Intel, c'est à dire compréhensible uniquement par un PC. Pour programmer un autre processeur, il faudrait apprendre de A à Z un nouveau langage assembleur, ce qui n'est pas évident.

```
7C581C4C   jl          7C5A67E8
7C581C52   mov         eax,dword ptr [ebp-1Ch]
7C581C55   mov         ecx,dword ptr ds:[7C5D047Ch]
7C581C5B   mov         dword ptr [ecx+8],eax
7C581C5E   push        1
7C581C60   add         esi,70h
7C581C63   push        esi
7C581C64   lea         eax,[ebp-20h]
7C581C67   push        eax
7C581C68   call        edi
7C581C6A   mov         dword ptr [ebp-28h],eax
7C581C6D   test        eax,eax
7C581C6F   jl          7C5A67F0
7C581C75   mov         eax,dword ptr [ebp-1Ch]
7C581C78   mov         ecx,dword ptr ds:[7C5D047Ch]
7C581C7E   mov         dword ptr [ecx+0Ch],eax
7C581C81   jmp         7C57FC26
7C581C86   mov         esi,7C5D03A0h
7C581C8B   push        esi
7C581C8C   call        dword ptr ds:[7C571180h]
```

Ce programme est beaucoup plus compréhensible que du langage machine, n'est-ce pas ?

On se heurte donc toujours au même problème : il faut connaître intimement le fonctionnement d'un processeur pour programmer en assembleur, qui est situé un niveau au dessus du langage machine. Or, ce n'est pas forcément le travail qu'on attend d'un programmeur…

Le niveau 2 : Fortran, Pascal, C vs LISP et Prolog

Les principaux soucis des langages assembleur sont donc : le manque de portabilité (il faut réécrire les programmes lorsqu'on change de processeur) et les connaissances pointues des instructions de base du microprocesseur nécessaires à leur écriture. Pour pallier à ces problèmes, de nouveaux langages sont apparus entre les années 60 et 75, chacun d'eux avec une philosophie différente. Cependant, ils partagent tous le point commun de pouvoir être traduits dans plusieurs langages machine, et non dans un seul. Ainsi, il suffit de copier un programme en langage C ou Pascal d'une machine sur une autre, puis de le traduire (cette opération de traduction est la compilation ou l'interprétation, nous reviendrons plus tard sur la distinction entre ces deux méthodes) pour avoir son équivalent en langage machine. Plus besoin de faire ce travail ! En conséquence, ces langages ne doivent plus manipuler des données trop proches de la machine, car leur traduction deviendrait difficile; ils permettent une programmation plus aisée des algorithmes, même s'ils restent assez pauvres au niveau syntaxique et au niveau du vocabulaire. Le langage C par exemple, ne reconnaît que 32 mots !

Il existe déjà une distinction à ce stade entre plusieurs familles de langage :

- les langages dits impératifs (objets ou procéduraux) comme le C, le Pascal ou le Fortran et leur successeurs C++ et Java entre autres. Ces langages indiquent instruction après instruction ce qu'il faut faire en détail pour arriver à la solution cherchée;
- les langages fonctionnels comme LISP, Scheme ou CamL, qui considèrent que toute donnée définie est une fonction, et qui se manipulent de manière particulière, nous n'en parlerons plus par la suite. Cependant, se frotter à LISP ou Scheme peut être une expérience enrichissante pour votre vie d'informaticien, on apprend notamment à regarder les problèmes sous un angle différent ! Il existe de nos jours des versions orientées objet de LISP (langage CLOS par exemple);
- les langages logiques comme PROLOG, qui indiquent des ensembles de contraintes logiques que doivent respecter les données, et qui calcule les solutions d'un problème à partir de ces contraintes.

Le langage pris en exemple pour la manipulation d'un bras robot fait partie des langages impératifs procéduraux : il est relativement simple à aborder, mais possède une syntaxe très lourde. De plus, il ne s'approche que très lointainement du langage naturel, et risque d'être limité pour traiter des problèmes plus complexes.

Le niveau 3 : C++, Java

Ces langages sont les archétypes des langage orientés objet, dont le principal apport est de faciliter la mise en algorithme d'un problème en lui permettant de travailler avec des données complexes. Les objets informatiques, dont nous parlerons vers la fin de l'année scolaire, sont en fait des représentations des objets (au sens très large du terme) réels que l'on veut traiter par informatique. Par exemple, un programme de gestion de stocks d'automobile d'occasion chez un concessionnaire utilisera des objets 'voiture' qui auront certaines propriétés, mais ne pourront pas décrire exactement ce qu'est une voiture, car dans la vie réelle, une voiture est déjà quelque chose d'extrêmement compliqué !

Ces langages n'apportent par ailleurs pas de réelle nouveauté au niveau syntaxique, c'est à dire au niveau de la facilité de traduire l'algorithme en un programme écrit dans un langage facilement traduisible en langage machine.

Le niveau 4 : les générateurs de code

La plupart des programmes réalisés de nos jours intègrent une interface graphique (ou GUI), pour deux raisons principales : le système d'exploitation (windows ou unix/linux, pour schématiser) offre de manière standard une telle interface à base de fenêtres, de menus et de boutons, et l'interface des programmes est rendue plus conviviale (en théorie). Il faut donc, lors de la conception des programmes, prévoir ces facilités. Pour ce type de programmes, la programmation est dite évènementielle, c'est à dire que l'on indique comment doivent être traités les évènements générés par l'utilisation du programme : que faire lorsque la souris se déplace ? lorsqu'il y a un clic droit ou gauche à tel ou tel endroit ? lorsque l'on ferme une fenêtre ? La définition d'une "simple" fenêtre avec tous les évènements associés nécessite plusieurs dizaines d'instructions, car il faut spécifier tout un ensemble de dépendances entre les différents composants de l'interface graphique. Par exemple, une fenêtre peut être redimensionnable, et comporter des boutons et des zones de saisie de texte : à chaque redimensionnement, les boutons et zones de texte doivent être correctement replacés : il faut donc indiquer leur nouvel emplacement.

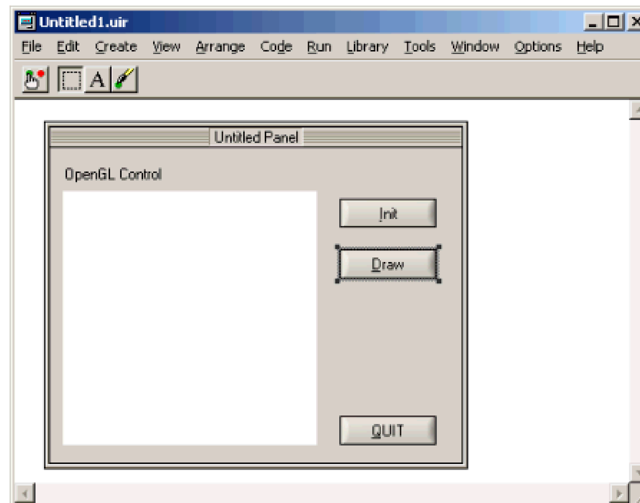

Figure 2 : Exemple de fenêtre conçue par un générateur d'interfaces

Tout cette partie de programme peut être réalisée de manière plus simple, tout simplement en réalisant cette interface graphique de manière graphique : le placement des boutons, des menus, des fenêtres est fait par l'exemple, comme sur la Figure 2. Il ne reste plus alors qu'à écrire les traitements correspondant aux évènements susceptibles de se manifester.

Le niveau 5 : les solveurs de problème

Ce sont des programmes spécialisés et le plus souvent appliqués à un domaine particulier, pour lequel ils sont particulièrement efficaces. Ces programmes fournissent un nouveau langage conçu spécifiquement pour ces problèmes, et qui nécessitent un nouvel apprentissage. L'avantage de ces langages est qu'ils sont de plus en plus détachés des aspects informatiques de base (quel processeur, quelle mémoire, quelles instructions de base) et se rapprochent de la formulation d'un problème en termes simples. Par exemple, les logiciels spécialisés dans la résolution de problèmes mathématiques, tels que matlab, octave, mathemica et maple sont les plus simples solveurs de problèmes. Un logiciel comme Deep Fritz, spécialisé dans le domaine des échecs, est également un solveur de problèmes : il est incapable de jouer aux dames, bien qu'il soit capable de faire match nul contre le champion du monde d'échecs. Seul un expert au jeu d'échecs est d'ailleurs capable de faire fonctionner ce genre de programmes pour en obtenir le meilleur. Il existe de nombreux domaines dans lesquels ces programmes sont utilisés, et de multiples formes de ces solveurs, apparentés à l'intelligence artificielle : systèmes experts, programmes auto-apprenants, optimisation sous contraintes.

Voici un exemple d'un tel solveur appliqué aux mathématiques : le logiciel MuPAD est capable de faire du calcul symbolique. On cherche à lui faire calculer l'intégrale de $\cos^2(x)$

- `int(cos(x)*cos(x),x)`

```
x    sin(2 x)
- + --------
2       4
```

L'instruction est : int(cos(x)*cos(x),x) (on indique la fonction à intégrer et la variable par rapport à laquelle n doit effectuer l'intégration). il s'agit d'une syntaxe plus simple à utiliser que la notation mathématique que l'on aurait du employer : $\int \cos^2(x).dx$?

La réponse est donnée sous une forme simple et compréhensible. Une version gratuite de MuPAD (nommée MuPAD Light) est disponible sur le site www.sciface.com.

Le niveau 6 : l'Intelligence Artificielle

L'intelligence artificielle est en fait déjà à l'œuvre dans certains solveurs de problèmes. Que recouvre exactement ce terme ? Il s'agit encore pour l'instant de programmes ou de systèmes censés imiter l'intelligence humaine, mais cette dernière est tellement complexe que l'on ne sait pas encore comment la décrire. Certains programmes essaient de construire des raisonnements logiques, d'autres sont bâtis sur des réseaux neuro-mimétiques (aussi appelés réseaux de neurones), d'autres encore sur la logique floue, et ces approches peuvent parfois être mixées. Mais il reste énormément de chemin à parcourir pour arriver à une véritable intelligence artificielle qui intégrerait autant d'aspects que notre intelligence peut le faire, car nous sommes capables d'élaborer des stratégies parfois très complexes, de manipuler des abstractions, et aussi d'imaginer ou de faire des actions qui n'ont pas de sens particulier. Un célèbre chercheur et pionnier de l'informatique, Alan Turing, avait trouvé un moyen simple selon lui de tester un programme intelligent : une personne, discutant par l'intermédiaire d'un clavier et d'un écran avec deux interlocuteurs (dont il ne connaît pas l'identité) et dont l'un peut être une machine, ne devait pas être capable de déterminer si l'un des deux est une machine au bout de 5 minutes d'échanges.

Voir à ce sujet la page web suivante : http://perso.enst.fr/~rigouste/tci/turing.htm, où il est notamment écrit : *Finalement, Turing lui-même semblait avoir déjà cerné, à l'époque, les points-clé à remplir. Pour la plupart, ils sont encore d'actualité : la machine doit pouvoir «être aimable, ingénieuse, amicale, avoir le sens de l'humour, distinguer le bien du mal, faire des erreurs, tomber amoureuse, être l'objet de sa propre réflexion, faire quelque chose de vraiment nouveau».*

Les systèmes d'exploitation : une pyramide de niveaux

Un programme tel que ceux que nous serons amené à étudier ou à écrire, sous sa forme en langage machine, fait appel à de nombreuses ressources de l'ordinateur, qui ne sont pas forcément d'un accès évident. L'ordinateur est un ensemble d'éléments inter communicants, et ces éléments se rendent service entre eux : clavier, disque dur, carte graphique. On dit qu'ils offrent des ressources de la machine pour qu'elles soient exploitées. Les programmes manipulent ces ressources, et il faut que ces manipulations soient les plus simples possibles. Par exemple, le simple affichage d'une valeur ou d'une lettre nécessite d'utiliser les ressources de la carte graphique. Cependant, cette carte n'est pas du tout aisée à programmer, car il faudrait le faire en langage machine, et donc tout le bénéfice de l'emploi d'un langage comme le C serait perdu ! Plutôt que de demander directement des ressources aux éléments, le programme va se tourner vers le système d'exploitation, qui permet d'accéder plus simplement aux ressources.

Le système d'exploitation est un programme dont le rôle principal est de permettre à d'autres programmes de fonctionner. Il gère, entre autres, les pilotes de périphériques, qui sont des programmes (encore ? et oui, dans un ordinateur, il y a beaucoup de programmes) permettant de faciliter l'accès aux ressources. Le système contient donc des programmes très proches du matériel, écrits en assembleur ou en C à un niveau assez bas. Ce que le système offre, plus globalement, consiste en :

Un système de gestion de fichiers : les programmes, ainsi que l'utilisateur, doivent accéder facilement aux fichiers par l'utilisation d'une structure arborescente et hiérarchique en répertoires

Un système de gestion des processus : n'importe quel ordinateur est aujourd'hui capable de faire fonctionner plusieurs programmes en même temps (c'est du moins l'illusion qu'il nous donne !), mais en réalité, il ne peut traiter qu'une tâche à la fois. Le principe est d'allouer des tranches de temps courtes (quelques millisecondes) à une tâche (encore appelée processus), puis de passer à la tâche suivante, pour quelques millisecondes également. Ces changements sont invisibles pour l'utilisateur, qui a l'impression que tous les programmes s'exécutent en même temps;

Un système de gestion de mémoire, pour s'assurer que tous les programmes auront assez de mémoire;

Un système de gestion des entrées/sorties, pour optimiser les accès à des ressources telles que les disques, qui présentent la particularité d'être beaucoup plus lents que la mémoire.

TABLE DES FIGURES

Figure 1 : bras robot articulé 3
Figure 2 : Exemple de fenêtre conçue par un générateur d'interfaces 12