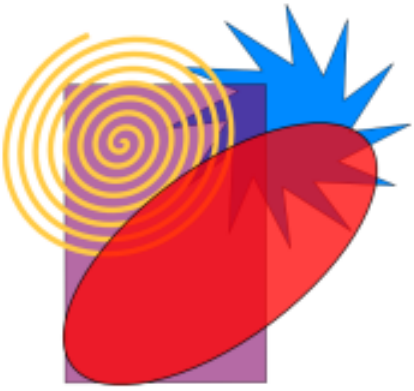
Edition 2010

Jacques THOORENS

principes et méthodes de programmation



École de Commerce et d'Informatique
Liège

Première partie

Premiers pas

Chapitre 1

L'homme et l'ordinateur

1.1 L'ordinateur est la création de l'être humain

Les premiers ordinateurs ont été créés au cours de la deuxième guerre mondiale, par des chercheurs américains. Malgré le côté révolutionnaire de cette invention, elle n'en reste pas moins profondément marquée par le caractère humain de ses concepteurs. Si les premiers ordinateurs étaient particulièrement ardus à programmer, dès le début des années 70, des ordinateurs plus conviviaux ont commencé à être produits. Les programmes de type « conversationnels » sont devenus monnaie courante. Au début des années 90, les interfaces graphiques, style Windows, ont quelque peu modifié la donne, mais on continue à simuler une interaction : action/réaction.

1.1.1 Analyse d'une conversation humaine

Nous allons prendre un exemple très simple d'un échange d'informations entre deux personnes, sur un sujet mathématique. L'une des deux personnes veut obtenir le résultat d'une addition et la seconde va effectuer l'addition et en donner le résultat.

- Dites moi combien font 4 et 5.
- 4 + 5 font 9.

En examinant le comportement de la personne qui calcule, nous pouvons discerner une série d'actions.

Écouter

L'homme dispose d'une paire d'oreilles reliées à son cerveau. Il va percevoir les vibrations sonores produites par son interlocuteur et les informations vont pénétrer dans son système nerveux central.

Retenir

Le calcul est simple, mais il faut néanmoins retenir les données. Si au lieu de 4 + 5, j'avais voulu additionner 187.899 et 147.942, cette phase de mémorisation paraîtrait plus naturelle.

Calculer

Pour effectuer son calcul, notre calculateur a besoin de connaissances. Encore une fois, cela paraît couler de source, mais on peut facilement constater que si on s'adresse à un enfant ou si on choisit des nombres plus grands, cette phase de calcul cesse d'être triviale. Il est probable que l'être humain, à moins d'être un phénomène de foire, va effectuer un calcul écrit. C'est ce que j'ai fait personnellement pour dire que notre deuxième exemple produisait 335.841. Nous reviendrons sur la portée exacte du calcul écrit.

Parler

Le calculateur doit, pour nous transmettre le résultat de son calcul, disposer d'une série d'organes, qu'un phonéticien appellera *appareil phonateur* pour émettre des sons perceptibles par l'interlocuteur.

1.1.2 Imitation de l'homme dans l'ordinateur

Les quatre types de comportements que nous avons isolés dans notre dialogue humain, vont réapparaître dans la structure d'un ordinateur. Il faut néanmoins savoir que la nature binaire des processus physiques qui fonctionnent dans l'ordinateur va fortement compliquer ces comportements.

Écouter - Les périphériques d'entrée

Au début, l'ordinateur possédait des dispositifs d'entrée très sommaires pour écouter ce que les hommes avaient à lui dire : des commutateurs ouverts ou fermés. Assez rapidement, on s'est mis à utiliser les **cartes perforées**, où les trous représentent l'ouverture et la fermeture de centaines de commutateurs. Les **claviers** sont venus beaucoup plus tard. Actuellement,

d'autres périphériques sont disponibles : souris, tablettes graphiques, scanner, claviers midi, micros et des capteurs physiques mesurant différents paramètres du réel (intensité de courants, température...) [1].

L'analogie entre l'être humain et les ordinateurs a ses limites. Les données introduites dans l'oreille d'un individu sous la forme d'ondes sonores ou par les touches tapées sur un clavier ne permettent pas de réaliser les calculs directement. Il faudra d'abord qu'elles soient respectivement transformées en représentations mentales ou binaires avant d'être traitées. Si nous connaissons parfaitement en quoi consiste une représentation binaire, la représentation « mentale » risque de rester longtemps encore un profond mystère. Il est d'ailleurs possible que cette représentation mentale soit différente dans le cerveau d'un épicier ou d'un lauréat du prix Nobel de mathématiques. Nous verrons qu'il n'est pas indispensable, du moins au niveau élémentaire, de comprendre comment se réalise ce transcodage de l'information.

Retenir - La mémoire

La mémoire est probablement la partie la plus complexe de l'ordinateur. Elle se compose de cellules élémentaires qui fonctionnent à la manière de mini-accumulateurs, capables de conserver de l'énergie électrique. La présence ou l'absence d'une infime différence de potentiel permet de savoir si on a mémorisé une information 1 ou une information 0. Notons que, dans la plupart des mémoires actuellement utilisées, il est nécessaire de disposer d'une source d'énergie pour régénérer constamment l'information mémorisée. Un ordinateur qui s'éteint perd donc sa mémoire.

L'élément fondamental de la mémoire est le *bit* (en anglais un morceau ou la condensation de *binary digit*, chiffre binaire). Un bit peut prendre deux valeurs, conventionnellement notées 1 et 0. En pratique, les bits sont groupés par mots, de 8 bits (un *octet*), de 16, de 32 voire de 64 bits. Les ordinateurs courants travaillent avec des mots de 32 bits, les ordinateurs 64 bits restent encore plus coûteux. Certaines consoles de jeux travaillent en 128 bits.

En pratique, chaque octet est identifiable par une adresse (dont la taille peut varier selon la puissance de la machine). En général, l'adresse désigne le premier octet d'une série de 2, 4 ou 8. Une information stockée dans un mot, quelle que soit sa largeur, n'a aucun sens pour l'ordinateur. Ainsi, un octet pourra représenter selon le cas un caractère, une note de musique, huit points d'une image en noir et blanc, un point d'une parmi 256 couleurs dans une image en couleur. Un mot de 32 bits pourra représenter un nombre entier compris dans l'intervalle -2 milliards + 2 milliards ou un nombre décimal avec 6 chiffres significatifs.

Calculer - Le processeur

Les aptitudes au calcul montrent une supériorité écrasante de l'ordinateur sur l'être humain. Les processeurs se montrent capable de réaliser un nombre sans cesse croissant d'opérations mathématiques. Les processeurs les plus simples sont capables de réaliser des additions et soustractions de nombres entiers. Les processeurs les plus performants effectuent les quatre opérations fondamentales directement sur des nombres pseudo-réels. En outre, les processeurs sont capables de réaliser des opérations de type logique. Nous n'expliquerons pas en détails toutes les possibilités des processeurs (lecture/écriture dans la mémoire et les canaux d'entrées/sorties, chargement et exécution des instructions, rupture dans la séquence de ces instructions, comparaisons et adaptation du comportement en fonction de leur résultat).

1. L'étude détaillée de l'ordinateur et de ses périphériques fait l'objet d'un cours à part entière. Nous ne détaillerons donc pas ces caractéristiques plus qu'il n'est nécessaire pour la compréhension de l'exposé.

Parler - Les périphériques de sortie

Historiquement, les ordinateurs ont d'abord produit des textes écrits sur des imprimantes. Actuellement, l'imprimante ne sert plus qu'à garder trace de traitement terminés et utiles. On dispose de l'écran pour l'affichage de la majorité des messages. Il existe également des périphériques de sortie permettant à l'ordinateur de commander l'environnement au moyen de signaux électriques.

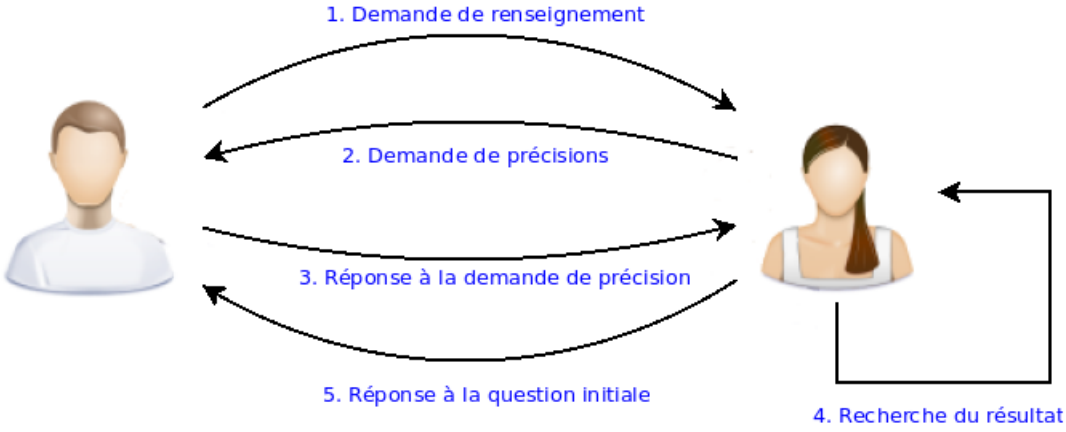
Périphériques complexes

D'autres périphériques sont d'un emploi quotidien : souris, disques durs, lecteurs de disques, de CD et de cartouches... Leur interaction avec des programmes se situe à un niveau de complexité sans commune mesure avec ce cours d'introduction. Je laisserai donc leur examen à des collègues d'autres cours.

1.2 Simulation du comportement humain

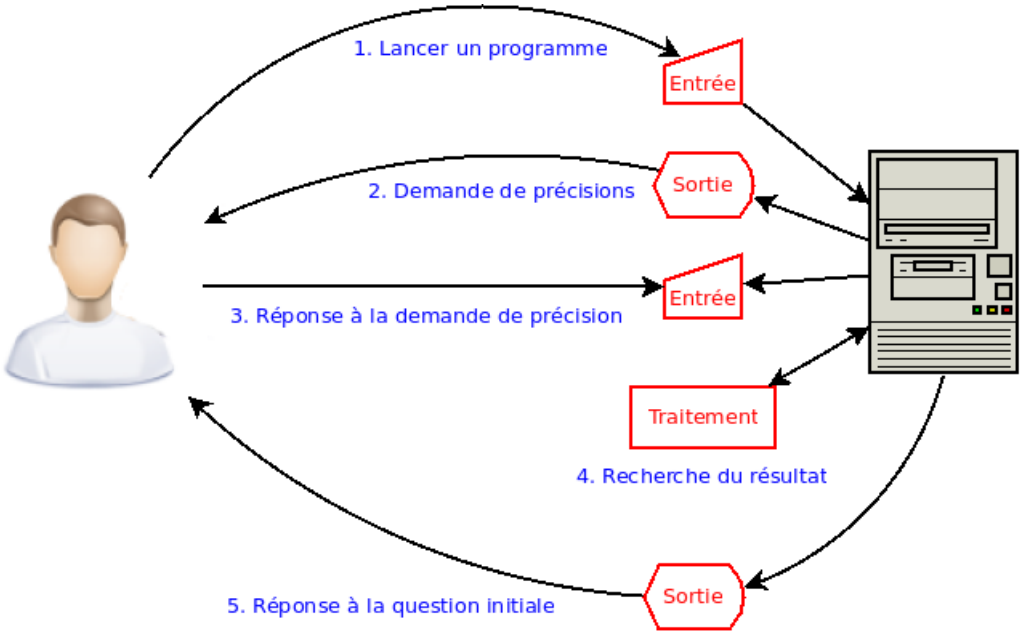
1.2.1 Un ordinateur n'est pas un être humain

Une première différence fondamentale distingue notre ordinateur de son père créateur : au contraire de l'homme, l'ordinateur est totalement incapable de s'adapter aux circonstances. Si parfois, il semble réagir avec énormément d'à propos, c'est parce qu'un programmeur habile a énormément travaillé pour y parvenir.



Le programmeur va donc devoir, dans la mesure du possible, anticiper toutes les circonstances qui peuvent se produire et préparer des réponses appropriées. Nous verrons plus tard qu'un vrai programme peut supposer des traitements extrêmement complexes. Si nous voulons en rester à notre petit exercice simple, nous allons devoir nous limiter à quelques points bien précis :

- écouter les nombres fournis par l'utilisateur
- calculer le résultat
- afficher le résultat



Cette première version du programme sera totalement incapable de réagir à des réponses non appropriées.

1.2.2 Le programmeur n'est pas l'utilisateur

Pour bien comprendre la programmation, il est indispensable de bien distinguer le programmeur de l'utilisateur. Le **programmeur** est celui qui écrit, qui *programme* l'ordinateur pour le rendre capable d'une série de comportements, dans notre cas, le rendre apte à additionner des nombres tapés au clavier. L'**utilisateur** est celui qui va utiliser le programme pour obtenir une réponse à ses besoins (ici l'addition des nombres 187.899 et 147.942). Il est important de bien comprendre que le programmeur ne sait absolument pas quels nombres l'utilisateur va proposer. Si le programme s'utilise plusieurs fois, le programme devra calculer la somme de paires de nombres différents à chaque fois. Si le programmeur connaissait les nombres à l'avance, le programme se réduirait à afficher 335.841. Notons qu'un tel programme ne pourrait être utile qu'à ceux qui doivent additionner ces deux nombres : un marché plutôt réduit !

C'est précisément l'ignorance des nombres qui seront effectivement utilisés, qui va m'obliger à concevoir mon programme de manière à ce qu'il fonctionne avec n'importe quelle paire de nombres, ou du moins un grand nombre de paires (en l'occurrence, nous nous contenteront de 16 milliards de milliards de paires).

1.2.3 Nécessité des variables

Pour pouvoir mémoriser les données fournies par l'utilisateur, nous allons utiliser la mémoire de l'ordinateur. Les premiers programmes devaient spécifier l'adresse de chacune des cellules utilisées. Les langages modernes disposent du mécanisme de la *variable*, qui est beaucoup plus facile à manipuler. Une variable possède les caractéristiques suivantes :

son identificateur : qui est en fait le *nom* de la variable. La forme de l'identificateur obéit à des règles différentes d'un langage à l'autre. En général, seules les lettres non accentuées et les chiffres sont autorisés. On peut aussi utiliser le blanc souligné (*underscore* en anglais). Les compilateurs modernes autorisent parfois un nombre de lettres supérieur à 24 (la taille de « anticonstitutionnellement »), ce qui est généralement suffisant ! Remarquons que le langage C, qui sera utilisé au cours de cette année, peut distinguer deux

identificateurs au moyen de la casse : `PrixTotal`, `PRIXTOTAL` et `prixtotal` sont trois identificateurs distincts en C. Le choix du l'identificateur est fondamental pour la relecture du programme et sa compréhension par le programmeur ou d'autres personnes. Il doit être long assez pour décrire la valeur qu'il représente, mais pas trop pour ne pas allonger inutilement les instructions.

sa taille : selon la précision recherchée, on utilise 1, 2, 4 ou plus d'octets pour représenter un nombre. D'autres variables plus complexes peuvent nécessiter plus d'espace encore.

son type : un même emplacement mémoire peut contenir un caractère, un nombre, une note de musique ou une portion de graphique. Il est donc indispensable de préciser le type d'information qu'on va placer à l'endroit désigné par la variable. Dans la plupart des langages, le type spécifie également la taille (il y a donc des sous-types en fonction de la taille).

sa valeur : Celle-ci va changer au cours du déroulement du programme. Une variable a toujours une valeur, parfois aléatoire lorsqu'aucune instruction n'a modifié le contenu de la variable. Il est en principe imprudent de tenter d'utiliser le contenu d'une variable qui n'a pas reçu de valeur initiale. C'est une bonne pratique de programmation de toujours *initialiser* les variables. Bien que beaucoup de langages permettent d'initialiser une variable au moment de sa déclaration, je conseille de toujours initialiser une variable à l'aide d'une instruction explicite (voir plus loin).

Dans notre programme, nous aurons besoin de deux variables pour mémoriser les deux nombres choisis par l'utilisateur. Nous verrons que le programme sera plus clair si nous choisissons une troisième variable pour mémoriser le résultat.

1.2.4 Première ébauche du programme : commentaires

Notre travail de programmation se définit à présent de la manière suivante :
- définir les variables
- définir le traitement à effectuer
  - écouter les nombres fournis par l'utilisateur et les mémoriser dans les variables
  - calculer le résultat et le mémoriser
  - afficher le résultat

Le programme que nous avons défini peut parfaitement s'écrire sur une feuille de papier ou dans un fichier d'ordinateur à l'aide de phrases françaises. Aucun ordinateur ne sera capable de comprendre cette première version, mais nous allons l'écrire quand même. Cette première ébauche peut parfaitement rester dans la version finale du programme à condition d'indiquer à la machine que c'est du blabla destiné aux humains. On appelle ce blabla des commentaires. Voici ce que pourrait devenir notre ébauche :

```
/* Définition des variables */
/* Traitement */
   /* Saisie des données à traiter */
   /* Calcul du résultat */
   /* Affichage du résultat */
```

J'utilise le marqueur `/*` pour indiquer le début d'un commentaire et `*/` pour en indiquer la fin. La plupart des professeurs de programmation insistent sur l'importance des commentaires dans un programme. Ils assurent la documentation du programme et facilitent sa maintenance. Un bon commentaire doit expliquer la signification d'une opération et non la décrire. Voici deux exemples de commentaires :

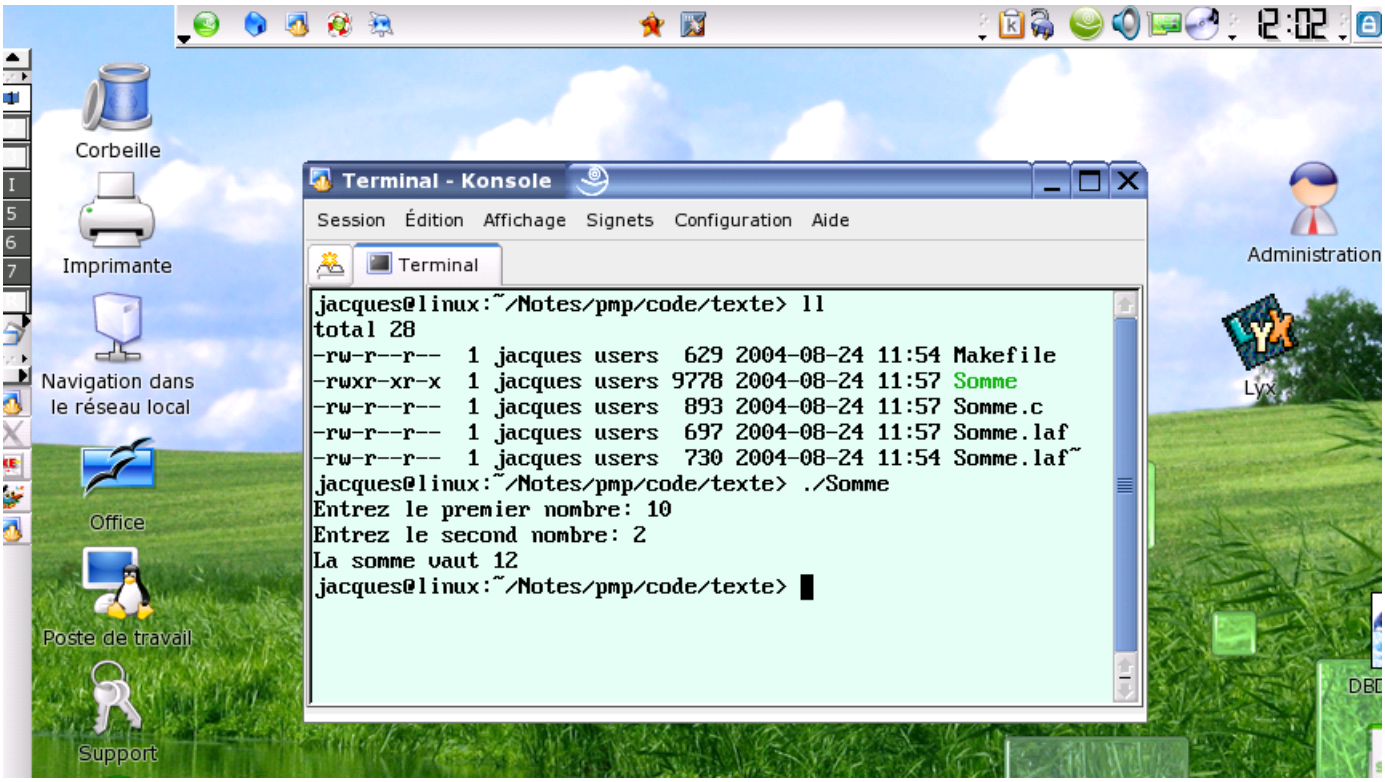
– commentaire inutile

```
/* 90% du prix total*/
fPrixTotal <- fPrixTotal * 0.90
```

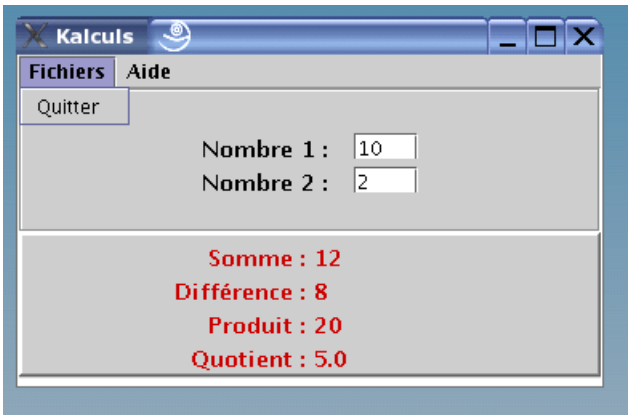
– commentaire explicatif

```
/* Calcul de la remise de 10% */
fPrixTotal <- fPrixTotal*0.90
```

À titre d'exemple, voici ce que pourrait donner notre futur programme lors de son exécution. On notera que le caractère champêtre du fond d'écran n'arrive pas à compenser la rugosité de l'interface du programme.



Avant de voir plus avant comment réaliser notre programme, il faut signaler qu'une autre approche pourrait être suivie. Le programme suivant utilise une interface en mode graphique. Nous pouvons grâce à elle calculer plusieurs opérations en tapant des nombres dans les zones de saisie ou mettre fin au programme en choisissant l'option *Quitter* du menu *Fichiers*. Un tel programme est évidemment beaucoup plus complexe à écrire, pas seulement à cause des quatre opérations.



1.3 Décrire un programme

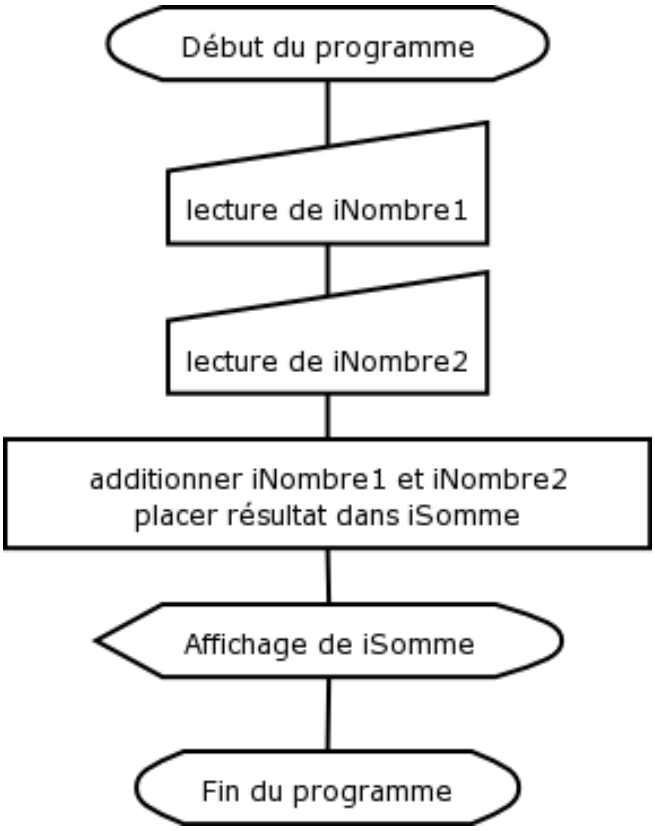
Avant d'encoder un programme dans l'ordinateur, il convient de spécifier clairement ce qu'il va faire et comment. Dans un cas simple comme celui de notre addition, il serait possible d'écrire directement le programme, mais dans les cas de la vie réelle, le programme est tellement complexe qu'il existe de nombreuses manières de l'écrire. Certaines sont excellentes, d'autres correctes, mais maladroites, parce que lentes ou difficiles à comprendre, d'autres encore hasardeuses et enfin, certaines produisent des résultats systématiquement faux. L'expérience démontre qu'un programme rédigé directement sur un ordinateur appartient dans 99% des cas à cette dernière catégorie. Cela s'explique par la complexité des langages et au nombre de détails auxquels il faut penser.

Plusieurs techniques sont disponibles pour décrire un programme :

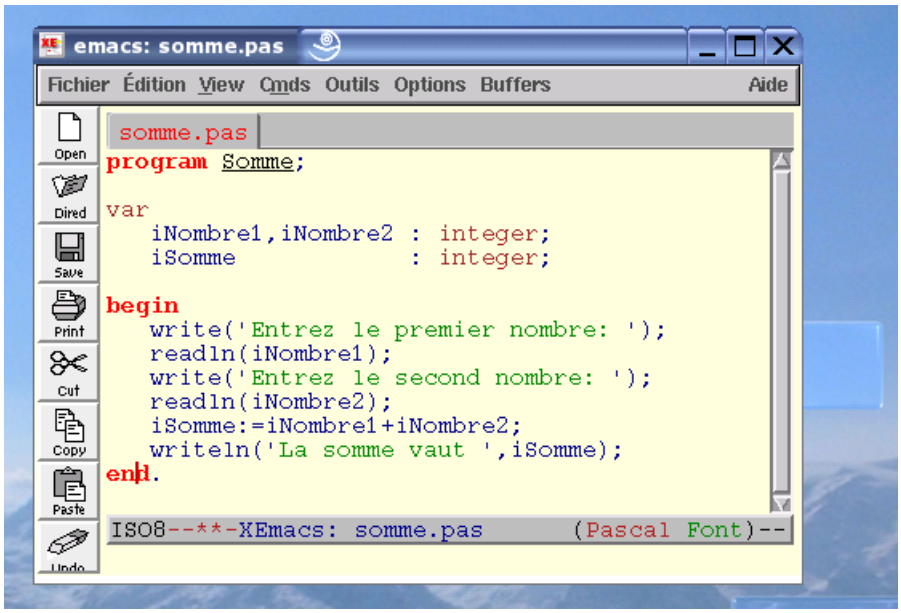
1. une méthode graphique, à la mode dans les années 60 et 70, porte le nom d'*ordinogramme*. Elle est fastidieuse pour les programmes simples et peut amener à des erreurs de programmation pour la description des boucles. L'un de ses avantages est sa capacité à permettre des effets de zoom. On peut donc faire un organigramme qui montre une vue générale du programme et plusieurs sous-ordinogrammes qui montrent des détails [2]. Son usage tend à se perdre [3].

2. Nous verrons qu'une bonne programmation procédurale permet de récupérer cet avantage sans devoir recourir au dessin d'ordinogrammes.

3. Pour les esprits curieux, on trouve un résumé intéressant de la notion d'ordinogramme à l'adresse suivante : *http ://www.efii.com/ANFH/cadre/goweb/Cadre_GO_Ordinogramme.htm.*



2. une description en français des différentes actions à entreprendre permet l'effet d'échelle propre à l'ordinogramme (voir notre description du programme esquissée dans les paragraphes précédents). Cependant, une même action peut se décrire de cinquante manières différentes. Il peut également résulter beaucoup d'ambiguïté dans les termes employés, ce qui fera que différentes programmeurs comprendront la description de manières différentes.
3. l'emploi d'un langage prévu pour décrire des programmes, comme le langage Pascal par exemple, présente l'avantage d'une plus grande précision, mais oblige l'utilisateur à se soucier de détails fastidieux comme une ponctuation omniprésente et peu intuitive.



4. l'emploi d'un pseudo-code, qui est proche du langage pédagogique, mais moins tatillon dans ses formes. Le pseudo-code est généralement écrit dans la langue maternelle des

utilisateurs, ce qui améliore sa lisibilité. Il doit néanmoins employer des mots précis pour éviter l'ambiguïté mentionnée plus haut. Il fournit à l'utilisateur un certain nombre de commandes correspondant à des opérations nécessaires à la réalisation des programmes. Il se distingue d'un langage (ou code) par le fait que ces opérations ne correspondent pas à des opérations réellement exécutables sur un ordinateur parce qu'elles sont trop complexes, ou trop peu explicites. Un pseudo-code ne se soucie donc pas toujours de tous les détails, mais permet d'écrire une esquisse facile à comprendre et à évaluer.

Dans notre pratique, nous allons utiliser un mélange des solutions 2, 3 et 4. Nous allons employer un pseudo-code (solution 4), proche du langage Pascal (solution 3) dans lequel nous ajouterons de nombreux commentaires en français (solution 2). En outre, ce pseudo-code maison offre l'avantage de pouvoir être traité par un programme et traduit en langage C pour permettre l'exécution du programme. Cette dernière facilité nécessite quelques précautions formelles qui brident un peu la liberté. Il est également possible d'opérer cette traduction vers le langage C ou tout autre langage manuellement. Notre pseudo-code s'appelle LAF (pour Langage Algorithmique Francisé, avec un clin d'œil aux anglophones, puisque *laugh* se prononce de la même façon et signifie rire).

1.3.1 Définition des variables

En général, les variables des langages typés [4] appartiennent à un type précis. Pour les besoins de ce cours, nous n'avons pas l'utilité de raffiner nos types à l'extrême et nous nous contenterons de cinq types. Par souci de clarté, nous prendrons l'habitude de préfixer chacun de nos identificateurs de variables par une lettre rappelant le type.

- un type `Entier` correspondra à des nombres entiers, c'est à dire les nombres naturels et leurs équivalent négatifs (ce que les mathématiciens appellent l'ensemble Z). En réalité, ce type ne correspond qu'à un petit sous-ensemble de Z puisque Z est infini et que pour distinguer une infinité de nombres, il nous faudrait une mémoire infinie. Dans les langages actuels, on va considérer les nombres compris entre -2.000.000.000 et +2.000.000.000 (j'arrondis), ce qui peut suffire pour nos besoins. Nous utiliserons la lettre minuscule `i` pour préfixer nos variables entières : `iTaux`, `iJour`, `iNombreDEtudiants`. Pour toutes nos variables traitées par le compilateur, nous prendrons soin de toujours faire suivre l'initiale d'une majuscule.
- un type `Flottant` correspond à un sous-ensemble des nombres réels [5]. Ici l'approximation est beaucoup plus grande et chaque programmeur doit être conscient que les erreurs de représentation peuvent induire des erreurs de calcul graves. Dans un calcul de budget familial, le problème n'est pas très important. Le budget d'un état peut déjà poser des problèmes plus sérieux, sans parler des calculs astronomiques où les erreurs peuvent provoquer l'atterrissage de la sonde martienne sur Jupiter. Pour des calculs spécifiques, il faut avoir recours à des modules de calculs spécialisés qui sortent de nos préoccupations

4. Il existe des langages non-typés dans lesquels les variables peuvent contenir n'importe quel type de valeur (par exemple, les cellules du tableur Excel, qui sont en fait des variables nommées A1, A2,...., peuvent contenir du texte, des nombres, des dates selon ce que l'utilisateur met dedans). Le danger est grand quand on fait des mélanges.

Les langages fortement typés imposent des contraintes énormes sur l'utilisateur. Ils refusent plus ou moins catégoriquement les mélanges de types. Pascal interdit d'ajouter 1 à une variable qui contient un caractère. Java refuse de copier le contenu d'une variable flottante dans une variable entière, parce qu'une partie de son contenu est perdue.

Les langages à typage faible (comme C) sont parfois très tolérants. Si on ajoute 1 à une variable caractère qui contient 'A', elle prend la valeur 'B'. Ce n'est pas nécessairement désirable.

5. Les nombres rationnels sont rarement implémentés dans les langages.

actuelles. Nous utiliserons la lettre minuscule `f` pour préfixer nos variables flottantes : `fPrixHorsTVA`, `fTVA`, `fPrixTotal`.
- un type `Caractère` permet de représenter des caractères isolés, par exemple pour tester une réponse au clavier. Nos ordinateurs reconnaissent tous les caractères employés par le français. Les Arabes ou les Chinois n'ont pas notre chance. Nous utiliserons la lettre minuscule `c` pour préfixer nos variables caractères : `cOption`, `cInitiale`.
- un type `Chaîne` permet de contenir des mots ou des messages entiers. En pratique, les chaînes sont difficiles à modifier : `sNationalite`, `sNomClient`. Nous préfixerons les chaînes à l'aide d'un `s` (comme dans le mot anglais *string*).
- un type `Logique` qui pourra contenir la valeur `VRAI` ou la valeur `FAUX`. Son utilité n'apparaîtra que plus tard : `lTermine`. Noter le préfixe `l`.
- un type utilisateur qui sera défini par la suite.

Pour définir nos variables, en tête du programme, nous utiliserons la notation suivante :

`Variable` ***Type Identificateur***

Voici quelques exemples de déclarations :

```
Variable entière iNombreEtudiants
Variable flottante fTauxTVA
Variables caractères cReponse, cInitiale
Variables chaînes sNom, sPrenom
Variable chaîne # 5 sCourteReponse
```

Nous pourrons grouper plusieurs déclarations sur une seule ligne, en séparant les identificateurs par des virgules. Notons aussi la dimension explicite d'une chaîne de caractères (on peut ignorer le problème, le compilateur se chargeant de donner une taille suffisante, par défaut 256 caractères).

Les trois variables de notre premier programme correspondent donc à ceci :

```
Variable Entière iNombre1
Variable Entière iNombre2
Variable Entière iSomme
```

Dans un souci de rapidité, mais aussi pour grouper les variables similaires, nous pouvons les regrouper par « familles ». On notera également l'utilisation des commentaires.

```
/* Déclaration des variables
   ------------------------ */

/* Données de l'utilisateur */
Variables Entières iNombre1, iNombre2
/* Résultat du calcul */
Variable Entière iSomme
```

1.3.2 Opérations disponibles et primitives

Commande n° 1 : affichage d'un message

Dans le but de faciliter le dialogue, nous allons doter notre programme de la possibilité de parler à l'utilisateur, notamment pour expliquer ce qu'il attend et ce qu'il affiche.

`Afficher` ***"Texte"***

```
Afficher "Veuillez entrer un nombre"
```

Notons que ces messages ne sont jamais le résultat d'un calcul. Si le texte à afficher doit varier, il faudra employer une variable de chaîne. Pour clarifier les choses, le texte à afficher sera placé entre des guillemets droits. Si le texte doit contenir un guillemet, ce dernier sera précédé d'un barre inverse :

```
Afficher "Il a utilisé le mot \"programme\" dans son texte"
```

Commande n° 2 : lecture d'une donnée

Notre programme doit disposer d'une commande permettant la lecture d'une donnée au clavier et son rangement dans une cellule mémoire (une variable). Nous disposerons de la commande `Lire` pour réaliser cette opération.

`Lire` ***Identificateur***

Remarquons que selon le type de la variable, l'opération de conversion des données sera différente. Un nombre entier sera converti en un entier, un nombre flottant en flottant. La conversion dépend en fait du type de la variable qui suit. Voici quelques quelques exemples de lecture [6] :

```
Lire iTauxTVA
Lire fTotalHTVA
Lire cOption
Lire sNomClient
```

Commande n° 3 : affichage d'une donnée

L'opération d'affichage est exactement inverse de celle d'une lecture. On utilisera à nouveau la commande `Afficher`.

`Afficher` ***Identificateur***

Quelques exemples :

```
Afficher iTauxTVA
Afficher fTotalHTVA
Afficher cOption
Afficher sNomClient
```

Commande n° 4 : affectation d'une valeur dans une variable

Il arrive fréquemment que nous devions placer une valeur dans une variable. La méthode vue en premier lieu consistait à lire cette valeur au clavier, mais ce n'est pas toujours possible et il y a des cas où cette valeur provient d'un calcul réalisé par l'ordinateur. La commande `Devient` également notée `<-` place à sa gauche le nom de la variable et à sa droite la valeur à mémoriser.

6. Il n'est pas possible de lire une valeur logique, parce que cela n'a pas d'intérêt. De la même façon, les variables logiques ne pourront pas être écrite.

Identificateur* `Devient` *Valeur

Identificateur* `<-` *Valeur

Selon les cas, cette valeur sera :
– *une autre variable*, on peut parler de copie

```
iCopie Devient iNombre1
```

– *une constante*, on peut parler de (ré)initialisation.

```
iTauxTVA Devient 6
```

– *une expression complexe*. Dans un premier temps, nous utiliserons les quatre opérateurs de l'arithmétique et les parenthèses, mais nous serons amenés à enrichir peu à peu la notion d'expression. Comme dans un tableur, nous utiliserons les signes '`*`' et '`/`' pour la multiplication et l'addition. Nous pourrons également utiliser l'opérateur *modulo*, noté `%`, qui permet de calculer le reste de la division entière.

```
fPrixTotal <- fPrixHTVA *(1+TauxTVA/100)
```

Primitives du langage : nombres, caractères et texte

Les nombres se représentent sur un ordinateur comme dans la vie réelle (du moins la vie américaine). On emploie donc des chiffres, un point décimal et parfois, comme sur la calculatrice scientifique, les notations dites « ingénieur ». Les caractères sont entourés d'apostrophes simples et les chaînes de guillemets doubles. Il existe deux valeurs logiques : `VRAI` et `FAUX`.

Type	Exemple
Entier	`1, 0, -7, 192421`
Flottant	`1., -3.7, -1.456E-7`
Caractère	`'a', '#', 'C'`
Chaîne	`"bonjour", "Pierre"`
Logique	`VRAI, FAUX`

1.3.3 Description de l'algorithme

Afin de rendre les choses claires, nous marquerons le traitement proprement dit au moyen des expressions `Début du Traitement` et `Fin du Traitement`. Dans un premier temps, nous pouvons reprendre les différentes parties de ce traitement sous forme de commentaires.

```
Début du Traitement
        /* Lecture des données */
        /* Calcul du résultat */
        /* Affichage du résultat */
Fin du Traitement
```

Nous allons ensuite nous pencher sur les instructions qui nous permettront de réaliser ces différentes tâches.

Lecture des données

Pour chacune des variables entières intervenant dans la somme, nous allons utiliser l'instruction `Lire`.

```
/* Lecture des données */
Lire iNombre1
Lire iNombre2
```

Comme l'ordinateur exécutera chacune des instructions en effectuant une lecture des touches du clavier, il serait plus efficace d'informer l'utilisateur de ce qu'on attend qu'il fasse. L'instruction `Afficher` a précisément été définie à cette fin.

```
/* Lecture des données */
Afficher "Entrez le premier nombre: "
Lire iNombre1
Afficher "Entrez le second nombre: "
Lire iNombre2
```

Calcul du résultat

Une simple addition suffit pour obtenir la somme demandée.

```
iNombre1 + iNombre2
```

Il convient néanmoins de mémoriser ce résultat dans la troisième variable, `iSomme`. Nous avons pour cela l'instruction `Devient`.

```
/* Calcul du résultat */
iSomme Devient iNombre1 + iNombre2
```

Affichage du résultat

Pour afficher notre résultat, contenu dans la variable entière `iSomme`, il nous suffit d'utiliser l'instruction `Afficher Entier`.

```
/* Affichage du résultat */
Afficher iSomme
```

Néanmoins, un petit message sera le bienvenu.

```
/* Affichage du résultat */
Afficher "La somme vaut "
Afficher iSomme
```

1.3.4 Notre premier programme complet

Notre programme complet comprendra donc les déclarations de variables et le traitement [7].

```
/* == Calcul de la somme de deux nombres == */
/* Définition des variables */
Variables Entières iNombre1, iNombre2
```

7. Les commentaires en italiques constituent ici un rappel pédagogique de notre démarche en deux temps : définir les variables puis appliquer les traitements. Ils n'apparaîtront plus dans les prochains programmes que nous écrirons. Les commandes de pseudo-code `variable` et `Début du traitement` sont suffisamment explicites.

```
Variable Entière iSomme
/* Traitement */
Début du Traitement
         /* Lecture des données */
   Afficher "Entrez le premier nombre: "
   Lire iNombre1
   Afficher "Entrez le second nombre: "
   Lire iNombre2
        /* Calcul du résultat */
   iSomme Devient iNombre1 + iNombre2
        /* Affichage du résultat */
   Afficher "La somme vaut "
   Afficher iSomme
Fin du Traitement
```

Voici deux exemples d'exécutions du programme, où nous retrouvons le dialogue initial [8] :

```
jacques@naxos: ~ > ./Addition
Entrez le premier nombre: 10
Entrez le second nombre: 8
La somme vaut 18
jacques@naxos: ~ > ./Addition
Entrez le premier nombre: 17
Entrez le second nombre: 3
La somme vaut 20
jacques@naxos: ~ >
```

1.4 Enrichissement des instructions de base

Dans le but d'améliorer la lisibilité de l'écran, il est parfois utile de disposer de commandes pour spécifier sa gestion. Nous utiliserons les instructions suivantes pour y parvenir [9].

Commande	Utilité
`Effacer Écran`	Effacer tout l'écran
`Fin de Ligne`	Renvoyer le curseur à la ligne
`Tab`	Sauter à la prochaine tabulation
`Attente`	Attente d'une frappe de *Enter* avant de continuer
`&`	Séparer plusieurs affichages successifs

Voici un exemple d'une zone d'affichage de plusieurs lignes :

```
/* Version normale */
Afficher "Ce programme va effectuer l'addition de deux nombres "
Fin de ligne
Afficher "entrés au clavier. "
```

On peut également simplifier l'affichage de plusieurs données

8. Avec une petite tricherie, j'ai ajouté l'instruction `Fin de Ligne` à la fin du programme pour ne pas que le message de la console (`jacques@naxos:~ >`) vienne se coller sur le dernier caractère du résultat.

9. A la demande des étudiants, la commande `Long Texte` incommode a été remplacée par l'usage de `&` comme séparateur

```
/* Version normale */
Afficher "La somme de "
Afficher iNombre1
Afficher " et de "
Afficher iNombre2
Afficher " vaut "
Afficher iSomme

/* Version courte */
Afficher "La somme de " & iNombre1 & " et de " & iNombre2
Afficher " vaut " & iSomme
```

1.5 Autre exemple de dialogue

Comme autre exemple, nous allons imaginer un dialogue entre l'ordinateur et l'utilisateur, limité à des échanges de politesses. Voici deux exemples d'exécution du programme :

```
Bonjour !
Comment vous appelez-vous ? Michel
Ravi de vous connaître, Michel
```

```
Bonjour !
Comment vous appelez-vous ? Sarah
Ravi de vous connaître, Sarah
```

Cet exemple sera réalisé en classe. En voici la version terminée :

```
/* == Petit dialogue avec la machine == */
/* Déclaration de la seule variable */
Variable Chaîne sNomUtilisateur

Début du Traitement
  /* Saisie de la donnée */
  Afficher "Bonjour !"
  Fin de ligne
  Afficher "Comment vous appelez-vous ? "
  Lire sNomUtilisateur
  /* Affichage de la réaction */
  Afficher "Ravi de vous connaître "
  Afficher sNomUtilisateur
Fin du Traitement
```

Nous faisons ici usage d'une seule variable, de type chaîne de caractères, pour contenir le nom de l'utilisateur. Nous constatons qu'il n'y a pas de partie calcul dans ce programme, puisque la donnée est réaffichée telle quelle.

Vocabulaire

Je donne ici une série de termes dont il faut comprendre le sens (et même parfois donner une définition). La constitution d'un lexique écrit de termes nouveaux pourrait constituer une activité très rentable pour un étudiant de l'enseignement supérieur. À titre d'exemple, voici

la définition d'algorithme, terme expliqué au cours. Les définitions sont reprises dans deux dictionnaires différents. La première définition [10] est plus concrète, la seconde [11] plus précise.

Algorithme :

1. séquence définie d'étapes pour résoudre un problème logique ou mathématique.
2. description du schéma de réalisation d'un événement à l'aide d'un répertoire fini d'actions élémentaires nommées, réalisables *a priori* et à durée limitée dans le temps.

ALGORITHME, AFFECTATION, BINAIRE, BIT, CASSE, CHAÎNE DE CARACTÈRES, DIGITAL (anglicisme pour *binaire*), IDENTIFICATEUR, INITIALISER, INSTRUCTION, MÉMOIRE, MÉMOIRE DE MASSE, MOT, NOMBRE ENTIER, NOMBRE RÉEL, OCTET, ORDINOGRAMME, PÉRIPHÉRIQUE, PROCESSEUR, PSEUDO-CODE, VARIABLE

Exercices

Les exercices suivants seront réalisés. Une solution sera donnée en annexe (avec un temps de délai, parce qu'il est nécessaire de faire l'exercice soi-même avant de lire la solution).

I-1. Écrire un programme qui s'inspire du programme de calcul d'addition, mais calcule la différence de deux nombres

I-2. Écrire un programme qui demande deux nombres et en fournit la somme, la différence, le produit et le quotient. Expérience : que se passera-t-il quand on fournira un diviseur nul ? Que remarquez-vous à propos du quotient ?

I-3. Écrire un programme qui calcule la somme de quatre nombres entrés au clavier.

I-4. Rédiger un programme poli qui dit bonjour à l'utilisateur, lui demande gentiment son nom, son prénom et son adresse, avant de réafficher les données encodées avec une demande de confirmation. Nous ne disposons pas, pour le moment, des outils nécessaires à une la programmation d'une quelconque réaction en cas d'erreur. Patience ! Voici un exemple de dialogue :

```
Bonjour cher utilisateur.
Pouvez-vous me donner votre nom ? Dupont
Pouvez-vous donner votre prénom ? Michel
Donnez-moi votre adresse complète : rue de l'Église 5 - 4000 Liège
Merci. Veuillez contrôler les informations :
Michel Dupont
rue de l'Église 5 - 4000 Liège
Est-ce correct ? (O ou N) O
```

I-5. Écrire un programme de facturation qui demande le prix de deux articles (prix HTVA), affiche le total, le montant de la TVA (6%) et le prix total.

10. *Dictionnaire encyclopédique bilingue de la micro-informatique*. Microsoft Press, 1999.
11. *Dictionnaire de l'informatique*. Larousse, 1981.