Algorithmique et Programmation : 1re Année / 1er Trimestre Programmation Fonctionnelle

Serge Stinckwich ©1997
e-mail : Serge.Stinckwich@info.unicaen.fr
GREYC - UPRESA CNRS 6072
Esplanade de la Paix - 14032 Caen Cedex
http://www.info.unicaen.fr/~serge

—

IUT de Caen - Département informatique
Rue Anton Tchekhov - BP53 14123 Ifs Cedex
http://www.iutc3.unicaen.fr/~stincs
e-course : http://www.iutc3.unicaen.fr/~stincs/iut/1a/

22 septembre 1997

I - Introduction

> Commençons par le commencement, mais pas nécessairement dans cet ordre.
> – Dr. Who, Meglos

L'informatique est omni-présente dans la vie quotidienne. Elle permet le traitement automatique de données ou de certaines tâches. Pour cela, on doit indiquer aux machines comment réaliser ces tâches au moyen de «recettes» ou *algorithmes*. C'est leur étude qui motive ce cours.

Quelques exemples d'utilisation :

- informatique de gestion : système d'information, comptabilité, facturation, banque, gestion du personnel, ...
- gestion de gros volumes d'informations : bases de données (SGBD, CD-ROM), recherche d'informations, bibliothèque numériques, ...
- calcul scientifique : traitement d'images (médecine), calcul de déformation, d'efforts (aviation), supervision de procédés industriels (centrales nucléaires, usines chimiques), traitements statistiques, ...
- loisirs : applications multi-médias, éducation, jeux, minitel, services on-line, ...

Quelques définitions du dictionnaire

- *Informatique* : science du traitement rationnel automatique, notamment par machines informatiques, de l'information considérée comme le support des connaissances humaines et des communications.
- *Ordinateur* : machine à traiter automatiquement de l'information.
- *Information* : élément de connaissance pouvant être transmis par un signal ou un ensemble de signaux.

Deux notions importantes : connaissance et transmission.

La transmission s'effectue au moyen d'un codage de l'information. *Remarque* : L'ordinateur n'est pas la seule machine à pouvoir effectuer des traitements automatiques. Autres exemples : programmateur de machine à laver, orgue de barbarie, machines à tisser, à tricoter, ...

Programme : description précise des actions à effectuer par un ordinateur pour réaliser une tâche.

À partir d'informations connues, les *données*, il s'agit d'élaborer des information appellées *résultats*.

Définitions d'Algorithme

Définition de la notion d'algorithme (Al Kowaritzmi) : Un algorithme est la description, pour un exécutant donné (humain ou machinique), d'une méthode de résolution d'un problème, c'est-à-dire la suite d'opérations fournissant le résultat cherché.

*Autre définition (*ISO / AFNOR*)* : Ensemble fini de règles déterminées servant à résoudre un problème au moyen d'un nombre fini d'opérations.

De manière informelle, un algorithme est une procédure de calcul bien définie, qui prend en entrée une valeur, ou un ensemble de valeurs, et qui produit, en sortie, une valeur ou un ensemble de valeurs. Un algorithme est donc une séquence d'étapes de calcul permettant de passer de la valeur d'entrées à la valeur de sortie.

On peut également voir un algorithme comme un outil de résolution d'un problème calculatoire bien défini. L'énoncé du problème spécifie en termes généraux la relation désirée entre l'entrée et la sortie. L'algorithme décrit une procédure de calcul spécifique permettant d'établir la relation (Cormen).

Définition de Programme

On appelle *programme* un ensemble d'instructions qui introduites dans les cases de la mémoire, commanderont l'action de la machine et lui feront exécuter un traitement donné. (Arsac, 1970)

Un algorithme traduit dans un langage compréhensible par l'ordinateur (ou *langage de programmation*) donne un *programme*.

Un programme est d'abord écrit (généralement par un humain, mais c'est également possible par une machine), puis est ensuite exécuté sur un ordinateur. D'une manière similaire, un recette de cuisine est d'abord écrite dans un livre de recette, avant d'être réalisée par une personne.

Spécification : Description, dans un langage formel, du problème à résoudre. Cette description doit être complète et rigoureuse.

Utilisation vs programmation

Il est tout à fait possible d'utiliser un ordinateur sans savoir le programmer. L'utilisation même de haut niveau d'un ordinateur, ne fait pas de vous un informaticien pour autant ! !

La programmation est une activité humaine qui demande du temps, des compétences (formation) et qui coute cher !

On appelle «informatique appliquée» ou *bureautique*, l'étude de logiciels existants afin d'en faire le meilleur usage : logiciels de dessin, traitements de textes, tableurs, ...

On s'intéresse ici à «l'informatique fondamentale», c'est-à-dire à l'ensemble des outils et méthodes qui permettent l'écriture de logiciels fiables, attractifs, performants et adaptés aux problèmes à résoudre.

Contrairement à une idée reçue, l'informatique ne se réduit pas uniquement à des outils de calcul, aussi puissants soient-ils, mais elle est en fait une science à part entière. Une science qui dispose de ses *modèles*, ses *outils*, ses *limites*. Comme toute science, son utilisation qui peut conduire à des erreurs très graves, est sujette à de nombreux débats.

Le développement spectaculaire de l'informatique dans toutes les activités humaines repose sur deux avancées technologiques : celles

des composants électroniques (hardware) et celles des composants logiciels (software).

Le progrès dans les composants se mesure plutôt de manière quantitative : miniaturisation, amélioration des performances, nouvelles technologies, ... alors que le progrès dans le logiciel est plus qualitatif (on parle de *qualité logiciel*) : facilité d'utilisation, convivialité, ...

La notion d'algorithme n'est pas une spécificité de l'informatique. Par exemple une recette de cuisine, une partition musicale, une bande perforée d'orgue de barbarie, des instructions de montage pour un meuble en kit ou encore un patron pour faire un vêtement sont autant de formes différentes d'algorithmes.

Exemple d'algorithmique culinaire : comment faire une omelette ?

- Casser 4 œufs dans un bol,
- Battre les œufs avec un fouet,
- Mettre de l'huile dans une poêle,
- Si votre cuisinière est au gaz alors ouvrir l'arrivée de gaz et faire une étincelle sinon mettre en route la résistance,
- Poser la poêle sur la cuisinière,
- Verser les œufs battus dans la poêle,
- Tant que l'omelette ne correspond pas à votre goût faire chauffer les œufs,
- Éteindre la cuisinière,
- Servir l'omelette.

Forme et propriétés d'un algorithme

Un algorithme comprend un certain nombre *d'actions* (ou encore dans le jargon informatique des «instructions»). Ces actions suivent un certain ordre et sont exécutés séquentiellement, c'est-à-dire que l'on effectue l'action suivante dès que l'on a finit la précédente.

On parle également de processus, méthodes, techniques, procédures ou routines.

Pour que ce mécanisme se réalise avec succès, un algorithme doit possèder un certains nombres de propriétés :

- la forme la plus courante d'algorithme transforme des valeurs fournies en entrées, pour obtenir après quelques manipulations un ensemble de valeurs de sorties. Un algorithme doit clairement établir le nombre et le type des valeurs d'entrées et les conditions que doivent vérifier ces entrées pour que le résultat soit celui désiré.
- Définition précise de chacune des instructions : Chaque étape d'un algorithme doit être précisemment défini.
- Correction : Un algorithme est supposé résoudre un problème. Il faut donc pouvoir «prouver» qu'un algorithme va bien résoudre le problème demandé. Généralement, on utilise une méthode mathématique ou logique.

- Terminaison : Un algorithme est censé résoudre un problème dans un temps fini, c'est-à-dire qu'il doit se terminer en un temps fini.
- Description du résultat d'un algorithme.

À qui est destiné un programme ?

Un programme est destiné principalement à deux publics :

- l'ordinateur
- le lecteur humain

Mais que fait ce programme ?

```
float s=1944,x[5],y[5],z[5],r[5],j,h,a,b,d,e;
int i=33,c,l,f=1;
int g(){return f=(f*6478+1)%65346;}
m(){x[i]=g()-l;y[i]=(g()-l)/4;r[i]=g()>>4;}
main()
{char t[1948]=
" `MYmtw%FFlj%Jqig~%jqig~Etsqnsj3stb",*p=t+3,
*k="3tjlq9TX";
l=s*20;
while(i<s) p[i++]='\n'+5;
for(i=0;i<5;i++)
z[i]=(i?z[i-1]:0)+l/3+!m();
while(1)
{for(c=33;c<s;c++)
{c+=!((c+1)%81);j=c/s-.5;
h=c%81/40.0-1;p[c]=37;
```

```
for(i=4;i+1;i--)
if((b=(a=h*x[i]+j*y[i]+z[i])*a-(d=1+j*j+h*h)*
(-r[i]*r[i]+x[i]*x[i]+y[i]*y[i]+z[i]*z[i]))>0)
{for(e=b;e*e>b*1.01||e*e<b*.99;e-=.5*(e*e-b)/e);
p[c]=k[(int)(8*e/d/r[i])];}}
for(i=4;i+1;z[i]-=s/2,i--)
z[i]=z[i]<0?l*2+!m():z[i];
while(i<s)putchar(t[i++]-5);}}
```

Problèmes liés à la programmation

Voici quelques bonnes qualités que doit posséder un algorithme :

Un algorithme doit être *correct*, c'est-à-dire qu'il doit réaliser ce qu'il doit faire et que cela.

Un algorithme doit être efficace.

Un algorithme doit être lisible : il doit pouvoir être lu par l'auteur(!) et par d'autres personnes.

Équilibre entre lisibilité du programme et efficacité ?

Voici quelques difficultés qui surgissent naturellement lors de l'activité de programmation :

- Comment spécifier ce que l'on attend d'un programme ?
- Comment associer à un programme une description précise de ce qu'il calcule ?
- Comment s'assurer qu'un programme est correct vis-à-vis d'une spécification donnée ?

Un algorithme représente une *connaissance universelle* pour résoudre un problème. L'algorithme est plus fondamental qu'un langage de programmation qui n'est qu'un moyen d'exécution. La conception d'algorithmes est une activité intellectuelle difficile.

Elle requiert à la fois des connaissances et une certaine créativité. Malheureusement, il n'existe pas d'algorithmes (de méthodes) pour écrire des algorithmes ! ! !

L'algorithmique : une technique, une science ou un art ?

Langages de programmation

> «Ce qui fait la valeur d'un langage de programmation, ce n'est pas qu'il autorise, c'est ce qu'il interdit.» (Rosen).

Les langages de programmation connaissent depuis leur création une évolution qui tend à les *abstraire* des spécificités et des particularismes de telles ou telles architectures matérielles. Pour cela, ils utilisent des structures et des concepts issus des **mathématiques** et de la **logique**.

- **Langages fonctionnels** : Les langages fonctionnels sont comme leur nom l'indique basés sur la notion de fonction. L'écriture d'un programme consiste alors à construire des fonctions qui ont des arguments et qui rendent un résultat. On utilise ici une notion proche de la fonction en mathématique. L'ancêtre des langages fonctionnels est LISP (1960) (Voir module deuxième année)
- **Langages impératifs** : Cette famille de langage est basée sur la notion d' *état de la mémoire*. L'exécution d'une action ou instruction d'un langage impératif provoque un changement de cet état.

  Ecrire un programme au moyen d'un langage impératif consiste donc à décrire la suite des états par lesquels doit passer la

machine, qui à partir d'un état initial doit atteindre un état final qui fournit la solution au problème.

Les premiers langages impératifs (FORTRAN, COBOL, C) sont encore utilisés mais restent encore très proche de la machine.

Les années 70 ont vu se développer une famille de langage (ALGOL ou PASCAL) qui offrent au programmeur des facilités de structuration de son programme en blocs, d'assurer la communication entre ses blocs et de définir ses propres types de données. Ces langages permettent plus facilement d'écrire des spécifications.

– Autres familles de langages : Langages logiques, langages orientés-objets (seront abordés par la suite).

Remarque : Il n'existe pas de langages qui soient meilleurs qu'un autre ! ! ! Tout dépend de l'application, du problème que l'on désire résoudre.

Langages utilisés dans ce cours

Deux langages de programmations servent de support pour ce cours :

- CAML, langage fonctionnel typé, développé à l' INRIA (Institut National de la Recherche en Informatique et Automatique) depuis 1984.

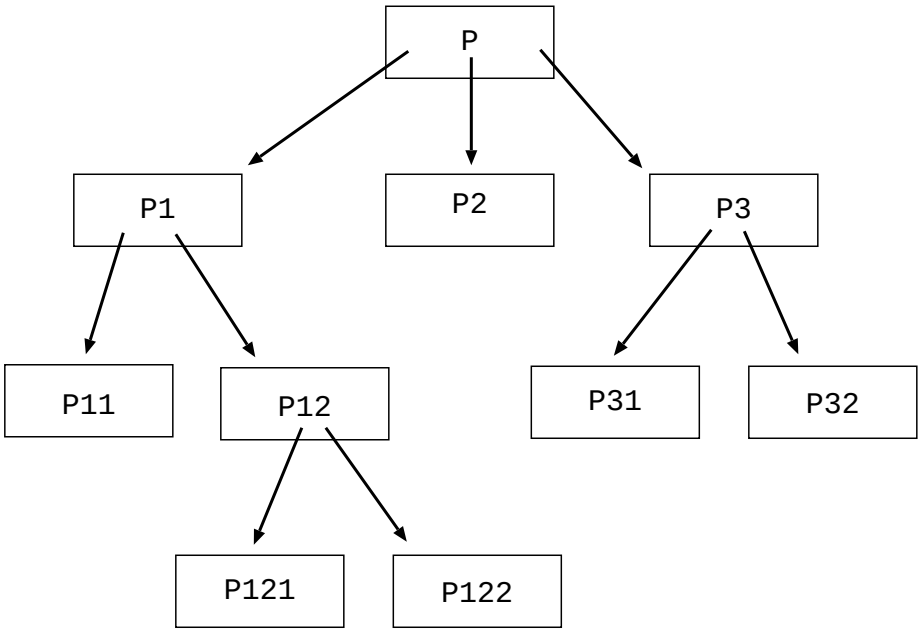
  Disponible pour UNIX, WIN * et MACOS. CAML permet de s'abstraire et donne de la programmation une vision moins embrouillée par des aspects techniques que d'autres.

  Langage rigoureux et didactique, utilisé en classes préparatoires.
- JAVA (Version 1.1.1). Langage objets. Disponible pour UNIX, WIN * et MACOS.

Conception d'un algorithme par analyse descendante

La conception d'un algorithme est une tâche difficile et très peu formalisée. De nombreuses méthodes existent. On choisit une méthode dite de *l'analyse descendante*.

Cette méthode consiste à décomposer un problème complexe en N sous-problèmes de complexité inférieure. Si un ou plusieurs problèmes demeurent trop complexes, on peut également les décomposer en sous-problèmes.



Exemple d'analyse descendante : conception d'un algorithme de détermination du plus petit de trois nombres

Algorithme Plus_Petit

Entrées :

a, b, c : trois nombres

Sorties :

p : le plus petit nombre

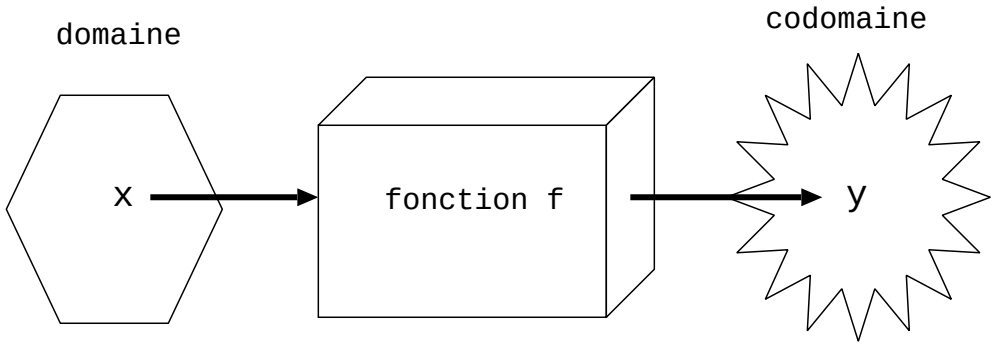
Actions :

- Phase 1 : Introduction des nombres dans la machine.
- Phase 2 : Déterminer le plus petit des trois.
  - Phase 21 : Comparer a avec b et mettre le plus petit des deux dans p.
  - Phase 22 : Comparer p avec c et mettre le plus petit des deux dans p.
- Phase 3 : Présentation du plus nombre p

Fonctions

Dans un style *fonctionnel*, la résolution d'un problème s'effectue en identifiant une ou plusieurs fonctions dont l'application à des valeurs données fournira le résultat escompté.

Une *fonction* est une association entre deux types, qui à chaque élément du type de départ, appellé *domaine* de la fonction fait correspondre un élément du type d'arrivé, appellé *codomaine*. Le domaine et le codomaine peuvent être des types de bases ou composés de plusieurs types.



Programmation fonctionnelle et analyse descendante

En programmation fonctionnelle, l'analyse descendante se traduit par la décomposition d'une fonction en sous-fonctions.

On distingue dans le processus de construction d'une fonction 3 étapes :

- *Spécifier une fonction* : nommer la fonction, définir son domaine et son codomaine et en fixer la signification. Par exemple, on introduit une fonction nommé `Cube` en indiquant que son domaine et sont codomaine sont de type `entier` et dont la signification est de calculer le cube d'un nombre entier.
- *Réaliser une fonction* : c'est associer à une fonction spécifiée une expression paramétrée. Par exemple, on associe au nom de fonction `Cube`, l'expression `x*x*x` ou x^3, où x est un nom formel. On parle de *paramètres formels* de la fonction : il s'agit de noms choisis pour décrire la fonction en faisant abstraction de la valeur particulière dont on veut calculer le cube.
- *Utiliser une fonction* : on décrit l'application aux arguments désirés au sein d'une expression, par exemple : `Cube(3)+Cube(4)`. Les arguments sont appellés *paramètres effectifs*.

Exemple de fonction

Moyenne Olympique :
Décrire une fonction qui associe à 4 entiers positifs, la moyenne des deux nombres parmi les 4 qui ne sont ni le plus grand, ni le plus petit. Par exemple, la moyenne olympique de 11, 9, 13, 17 est 12.

Spécification de la fonction :
On la nomme `moyenneOlympique`, son domaine est 4 entiers positifs et son codomaine un réel :

`moyenneOlympique : un entier` $\geq$ `0, un entier` $\geq$ `0, un entier` $\geq$ `0, un entier` $\geq$ `0` $\rightarrow$ `un réel`
{ moyenneOlympique(w,x,y,z) désigne la moyenne olympique des nombres w,x,y et z }

Exemple d'utilisation :
L'expression `moyenneOlympique(11,9,13,17)` donne la valeur 12, `(moyenneOlympique(2,3,5,5)+ moyenneOlympique(11,9,13,17))/2` calcule la moyenne de deux moyennes olympiques.

La *signification* de la fonction est indiquée ici par un commentaire.

Réalisation de la fonction :
On associe au nom de la fonction une expression qui peut être

composée en termes d'opérations primitives ou à partir de fonctions intermédiaires. On spécifie alors ces fonctions et on réalise la fonction initiale en les composant. Dans notre exemple, la fonction `moyenneOlympique` peut être réalisée de plusieurs façons :

- on classe les 4 entiers, on enlève le premier et le dernier et on fait la moyenne des deux restants.
- on enlève le plus petit des 4 entiers, puis on enlève le plus grand des trois restants et on fait la moyenne des deux restants.
- on additionne les 4 entiers, on soustrait le plus grand et le plus petit et on divise par 2.

On utilise la dernière idée, en introduisant deux fonctions intermédiaires :

`maxQuatre : quatre entiers` $\geq 0 \rightarrow$ `un entier`
{ maxQuatre(a,b,c,d) désigne le maximum des 4 entiers }

`minQuatre : quatre entiers` $\geq 0 \rightarrow$ `un entier`
{minQuatre(a,b,c,d) désigne le minimum des 4 entiers}

`moyenneOlympique` s'écrit alors :

```
moyenneOlympique (w,x,y,z) = (w+x+y+z - minQuatre(w,x,y,z)
- maxQuatre(w,x,y,z))/2
```

Réalisation des fonctions intermédiaires :

On décrit maintenant `minQuatre` et `maxQuatre` en termes de deux fonctions plus élémentaires :

`minDeux, maxDeux : deux entiers` $\geq 0 \rightarrow$ `un entier` désignent respectivement le minimum et le maximum de deux entiers.

```
minQuatre(i,j,k,l) = mindeux(mindeux(mindeux(i,j),k),l)

maxQuatre(i,j,k,l) = maxdeux(maxdeux(maxdeux(i,j),k),l)
```

et on réalise `minDeux` et `maxDeux` de la façon suivante :

```
minDeux (a,b) = (a+b-abs(a-b))/2

maxDeux (a,b) = (a+b+abs(a-b))/2
```