Algorithme

Table des matières

1 Codage

1.1 Système binaire

Lorsqu'un ordinateur traite du texte, du son, de l'image, de la vidéo, il traite en réalité des nombres. Mais ces nombres sont exclusivement des informations binaires.

Mais qu'est-ce qu'une information binaire ? C'est une information qui ne peut avoir que deux états : par exemple, ouvert - fermé, libre - occupé, vrai - faux ...

La mémoire vive (la « RAM ») est formée de millions de composants électroniques qui peuvent retenir ou relâcher une charge électrique. La surface d'un disque dur, d'une bande ou d'une disquette est recouverte de particules métalliques qui peuvent, grâce à un aimant, être orientées dans un sens ou dans l'autre. Et sur un CD-ROM, on trouve un long sillon étroit irrégulièrement percé de trous.

Toutefois, la coutume veut qu'on symbolise une information binaire, quel que soit son support physique, sous la forme de 1 et de 0. Il faut bien comprendre que ce n'est là qu'une représentation, une image commode, que l'on utilise pour parler de toute information binaire.

Lorsque nous disons que $4 + 3 = 7$, nous manions de pures abstractions, représentées par de non moins purs symboles !

Le concept de nombre, de quantité pure, a donc constitué un immense progrès (que les ordinateurs n'ont quant à eux, toujours pas accompli). Mais si concevoir les nombres, c'est bien, posséder un système de notation performant de ces nombres, c'est encore mieux.

1.2 La numérotation de position en base décimale

Notre système de numération est un système décimal de position. Il est constitué de 10 chiffres dont la position indique le nombre d'unités de la puissance de 10 indiquée par le rang.

$$3\,405 = 3 \times 1000 + 4 \times 100 + 0 \times 10^1 + 5 \times 1$$
$$3\,405 = 3 \times 10^3 + 4 \times 10^2 + 0 \times 10^1 + 5 \times 10^0$$

1.3 La numérotation de position en base binaire

Les ordinateurs, eux, comme on l'a vu, ont un dispositif physique fait pour stocker (de multiples façons) des informations binaires. Alors, lorsqu'on représente une information stockée par un ordinateur, le plus simple est d'utiliser un système de représentation à deux chiffres : les fameux 0 et 1.

Dans un ordinateur, le dispositif qui permet de stocker de l'information est donc rudimentaire, bien plus rudimentaire que les mains humaines. Avec un emplacement d'information d'ordinateur, on est limité à deux choses différentes seulement. Avec une telle information binaire, on ne va pas loin. Voilà pourquoi, dès leur invention, les ordinateurs ont été conçus pour manier ces informations par paquets de 0 et de 1. Et la taille de ces paquets a été fixée à 8 informations binaires.

Une information binaire (symbolisée couramment par 0 ou 1) s'appelle un bit. Un groupe de huit bits s'appelle un octet (en anglais, byte)

Combien d'états différents un octet possède-t-il ? Chaque bit de l'octet peut occuper deux états. Il y a donc dans un octet :

$$2^8 = 256 \text{ possibilités}$$

Cela signifie qu'un octet peut servir à coder 256 nombres différents : ce peut être la série des nombres entiers de 1 à 256, ou de 0 à 255, ou de -127 à $+128$. C'est une pure affaire de convention, de choix de codage.

Si l'on veut coder des nombres plus grands que 256, ou des nombres négatifs, ou des nombres décimaux, on va donc être contraint de mobiliser plus d'un octet. Ce n'est pas un problème, et c'est très souvent que les ordinateurs procèdent ainsi.

En effet, avec deux octets, on a $256 \times 256 = 65\,536$ possibilités.

En utilisant trois octets, on passe à $256^3 = 16\,777\,216$ possibilités.

Cela implique également qu'un octet peut servir à coder autre chose qu'un nombre : l'octet est très souvent employé pour coder du texte. Il y a 26 lettres dans l'alphabet. Même en comptant différemment les minuscules et les majuscules, et même en y ajoutant les chiffres et les signes de ponctuation, on arrive à un total inférieur à 256. Cela veut dire que pour coder convenablement un texte, le choix d'un caractère par octet est un choix pertinent.

Se pose alors le problème de savoir quel caractère doit être représenté par quel état de l'octet. Il existe un standard international de codage des caractères et des signes de ponctuation. Ce standard stipule quel état de l'octet correspond à quel signe du clavier. Il s'appelle l'ASCII (pour American Standard Code for Information Interchange). Et fort heureusement, l'ASCII est un standard universellement reconnu et appliqué par les fabricants d'ordinateurs et de logiciels. Bien sûr, se pose le problème des signes propres à telle ou telle langue (comme les lettres accentuées en français, par exemple). L'ASCII a paré le problème en réservant certains codes d'octets pour ces caractères spéciaux à chaque langue. En ce qui concerne les langues utilisant un alphabet non latin, un standard particulier de codage a été mis au point. Quant aux langues non alphabétiques (comme le chinois), elles payent un lourd tribut à l'informatique pour n'avoir pas su évoluer vers le système alphabétique…

Revenons-en au codage des nombres sur un octet. Nous avons vu qu'un octet pouvait coder 256 nombres différents, par exemple (c'est le choix le plus spontané) la série des entiers de 0 à 255. Comment faire pour, à partir d'un octet, reconstituer le nombre dans la base décimale qui nous est plus familière ? Il suffit d'appliquer les principes de la numérotation de position, en gardant à l'esprit que là, la base n'est pas décimale, mais binaire. Prenons un octet au hasard :

$$11\,010\,011$$

On a alors

$$\begin{aligned}11\,010\,011 &= 1\times 2^7 + 1\times 2^6 + 0\times 2^5 + 1\times 2^4 + 0\times 2^3 + 0\times 2^2 + 1\times 2^1 + 1\times 2^0\\ &= 1\times 128 + 1\times 64 + 1\times 16 + 1\times 2 + 1\times 1\\ &= 128 + 64 + 16 + 2 + 1\\ &= 211\end{aligned}$$

Inversement, comment traduire un nombre décimal en codage binaire ? Il suffit de rechercher dans notre nombre les puissances successives de deux. Prenons, par exemple, 186.

Il est bon de connaître les premières puissances de 2 :

2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
1	2	4	8	16	32	64	128	256	512	1024

On cherche la plus grande puissance de 2 contenu dans le nombre, on le soustrait puis à ce nouveau nombre on cherche la plus grande puissance de 2 et ainsi de suite.

Dans 186, on trouve 128, soit 1×2^7. Je retranche 128 de 186 et j'obtiens 58.

Dans 58, on trouve 32, soit 1×2^5. Je retranche 32 de 58 et j'obtiens 26.

Dans 26, on trouve 16, soit 1×2^4. Je retranche 16 de 26 et j'obtiens 10.

Dans 10, on trouve 8, soit 1×2^3. Je retranche 8 de 10 et j'obtiens 2.

Dans 2, on trouve 2, soit 1×2^1. Je retranche 2 de 2 et j'obtiens 0.

Il ne me reste plus qu'à reporter ces différents résultats (dans l'ordre !) pour reconstituer l'octet. J'écris alors qu'en binaire, 186 est représenté par :

$$186 = 10\,111\,010$$

1.4 Codage hexadécimal

Pour finir, on va évoquer un dernier type de codage, qui constitue une alternative pratique au codage binaire. Il s'agit du codage hexadécimal, autrement dit en base seize.

Pourquoi ? Tout d'abord, parce que le codage binaire, ce n'est tout de même pas très économique, ni très lisible. Pas très économique : pour représenter un nombre entre 1 et 256, il faut utiliser systématiquement huit chiffres. Pas très lisible.

Alors, une alternative toute naturelle, était de représenter l'octet non comme huit bits (ce que nous avons fait jusque là), mais comme deux paquets de 4 bits (les quatre de gauche, et les quatre de droite).

Avec 4 bits, nous pouvons coder $2^4 = 16$ nombres différents. En base seize, 16 nombres différents se représentent avec un seul chiffre (de même qu'en base 10, dix nombres se représentent avec un seul chiffre).

Quels symboles choisir pour les chiffres ? Pour les dix premiers, on prend les dix chiffres de la base décimale. Les dix premiers nombres de la base seize s'écrivent donc 0, 1, 2, 3, 4, 5, 6, 7, 8, et 9. Il nous manque encore 6 chiffres, on

prend les premières lettres de l'alphabet. Ainsi, par convention, A vaut 10, B vaut 11, etc. jusqu'à F qui vaut 15.

Prenons un octet au hasard :

$$10\ 011\ 110$$

Pour convertir ce nombre en hexadécimal, la méthode consiste à passer du binaire vers l'hexadécimal.

Divisons 10 011 110 en 1 001 (partie gauche) et 1 110 (partie droite).

1 001, c'est $8 + 1 = 9$

1 110, c'est $8 + 4 + 2 = 14$

Le nombre s'écrit donc en hexadécimal : $9E$.

Le codage hexadécimal est très souvent utilisé quand on a besoin de représenter les octets individuellement, car dans ce codage, tout octet correspond à seulement deux signes.

2 Introduction

2.1 Algorithme

Définition 1 : Un algorithme est une suite d'instructions, qui une fois exécutée correctement, conduit à un résultat donné.
Pour fonctionner, un algorithme doit donc contenir uniquement des instructions compréhensibles par celui qui devra l'exécuter.

La maîtrise de l'algorithmique requiert deux qualités :

- il faut avoir une certaine **intuition**, car aucune recette ne permet de savoir à priori quelles instructions permettront d'obtenir le résultat voulu.
- il faut être **méthodique et rigoureux**. En effet, chaque fois qu'on écrit une série d'instructions qu'on croit justes, il faut systématiquement se mettre à la place de la machine qui va les exécuter, pour vérifier si le résultat obtenu est bien celui que l'on voulait.

Les ordinateurs, quels qu'ils soient, ne sont fondamentalement capables de comprendre que quatre catégories d'ordres (en programmation, on n'emploiera pas le terme d'ordre, mais plutôt celui d'instructions). Ces quatre familles d'instructions sont :

- l'affectation de variables
- la lecture / écriture
- les tests
- les boucles

Un algorithmique exprime les instructions résolvant un problème donné indépendamment des particularités de tel ou tel langage de programmation.

Apprendre un algorithmique, c'est apprendre à manier la structure logique d'un programme informatique.

« Un langage de programmation est une convention pour donner des ordres à un ordinateur.»

2.2 Conventions pour écrire un algorithme

Historiquement, plusieurs types de notations ont représenté des algorithmes.

- Il y a eu notamment **une représentation graphique**, avec des carrés, des losanges, etc. qu'on appelait des organigrammes. Cependant dès que l'algorithme commence à grossir un peu, ce n'est plus pratique.
- On utilise généralement une série de conventions appelée « **pseudo-code** », qui ressemble à un langage de programmation authentique dont on aurait évacué la plupart des problèmes de syntaxe. Ce pseudo-code est susceptible de varier légèrement d'un livre (ou d'un enseignant) à un autre. C'est bien normal : le pseudo-code, encore une fois, est purement conventionnel ; aucune machine n'est censée le reconnaître.

3 Les variables

3.1 Définition

Définition 2 : Dès que l'on a besoin de stocker une information au cours d'un programme, on utilise une **variable**.

Pour employer une image, une variable est une boîte, que le programme (l'ordinateur) va repérer par une étiquette. Pour avoir accès au contenu de la boîte, il suffit de la désigner par son étiquette.

3.2 Déclaration des variables

La première chose à faire avant de pouvoir utiliser une variable est de créer la boîte et de lui coller une étiquette. C'est ce qu'on appelle la déclaration des variables.

Le nom de la variable (l'étiquette de la boîte) obéit à des impératifs changeant selon les langages. Toutefois, une règle absolue est qu'un nom de variable peut comporter des lettres et des chiffres, mais qu'il exclut la plupart des signes de ponctuation, en particulier les espaces. Un nom de variable correct commence également impérativement par une lettre. Quant au nombre maximal de signes pour un nom de variable, il dépend du langage utilisé.

Une fois le nom choisi, il faut déterminer le type de la variable. On peut distinguer 6 types de variable :

- Type numérique : un nombre (entier, décimal, réel).
- Type monétaire : un nombre avec deux décimales.
- Type date : jour / mois / année.
- Type alphanumérique : du texte.
- Type booléen : qui ne peut prendre que deux valeurs VRAI ou FAUX.

3.3 Affectation d'une variable

La seule chose qu'on puisse faire avec une variable, c'est l'affecter, c'est-à-dire lui attribuer une valeur.

$\boxed{a \leftarrow 24}$ Attribue la valeur 24 à la variable a.

$\boxed{a \leftarrow b}$ Attribut la valeur de a à la variable b.

une instruction d'affectation ne modifie que ce qui est situé à gauche de la flèche.

On peut aussi affecter une variable à l'aide d'une opération :

$\boxed{c \leftarrow a + 4}$ Attribut la valeur $a + 4$ à la variable c.

On peut changer la valeur d'une variable avec elle-même :

$\boxed{b \leftarrow b + 1}$ Augmente de 1 la variable b.

Opérateur alpha numérique : concaniser &

$\boxed{\begin{array}{l} A \leftarrow \text{Paul} \\ B \leftarrow \text{Milan} \\ C \leftarrow A\&B \end{array}}$ Attribue à la valeur C PaulMilan.

Les opérateurs numériques sont :

- L'addition $+$
- La soustraction $-$
- La multiplication $*$
- La division $/$
- La puissance $5^2 = 25$

Les opérateurs logiques sont :

- ET intersection de deux ensembles
- OU (non exclusif) union de deux ensembles
- NON complémentaire d'un ensemble

Deux remarques :

- En informatique une variable ne peut prendre à un moment donné qu'une valeur et une seule, contrairement à une équation qui peut éventuellement avoir plusieurs solutions
- Souvent l'affectation d'une variable se note : $B = A$ qui veut dire que B prend la valeur A, qui est alors différent de $A = B$ où A prend la valeur de B

4 Lecture et écriture d'une variable

Les mots lecture et écriture se situe au niveau du programme

4.1 Définition

Définition 3 : **Lire** une variable signifie que l'utilisateur doit rentrer une valeur pour que le programme la lise

Ecrire une variable signifie que le programme renvoie la valeur de la variable que le programme a trouvé.

4.2 Exemples

Un programme tout simple pour écrire votre nom

Instruction 1 : Ecrire "Entrez votre nom de famille"
Instruction 2 : Lire NomFamille

4.3 Exercice

On considère l'algorithme suivant :

Choisir un nombre. Lui ajouter 1. Multiplier le résultat par 2. Soustraire 3 au résultat. Afficher le résultat.

1) Appliquer cet algorithme à : 3, -4, 0, $\frac{1}{3}$.
2) Ecrire cet algorithme avec le logiciel Algobox. Vérifier les résultats obtenus.
3) Comment choisir un nombre pour que s'afficher le nombre 0 ? le nombre -5 ?
4) Ecrire à l'aide du logiciel Algobox un programme permettant en partant du nombre affiché, de retrouver le nombre choisi initialement.

1) On trouve aisement les résultats suivant : 5, -9, -1, $-\frac{1}{3}$
2) Le langage d'Algobox est très proche du pseudo code. On trouve ainsi :

```
1    VARIABLES
2      x EST_DU_TYPE NOMBRE
3    DEBUT_ALGORITHME
4      LIRE x
5      x PREND_LA_VALEUR x+1
6      x PREND_LA_VALEUR x*2
7      x PREND_LA_VALEUR x-3
8      AFFICHER x
9    FIN_ALGORITHME
```

3) Pour pouvoir trouver le nombre de départ, il nous faut remonter l'algorithme :

$$0 \xrightarrow{+3} 3 \xrightarrow{\div 2} \frac{3}{2} \xrightarrow{-1} \frac{1}{2}$$
$$-5 \xrightarrow{+3} -2 \xrightarrow{\div 2} -1 \xrightarrow{-1} -2$$

4) On obtient donc l'agorithme avec Algobox :

```
1   VARIABLES
2     x EST_DU_TYPE NOMBRE
3   DEBUT_ALGORITHME
4     LIRE x
5     x PREND_LA_VALEUR x+3
6     x PREND_LA_VALEUR x/2
7     x PREND_LA_VALEUR x-1
8     AFFICHER x
9   FIN_ALGORITHME
```

5 Les tests

Il y a deux formes pour un test : soit la forme complète, soit la forme simplifiée :

Forme complète	**Forme simplifiée**
Si condition **alors** Instructions 1 **Sinon** Instructions 2 **Finsi**	**Si** condition **alors** Instructions **Finsi** *Si la condition n'est pas vérifiée le programme ne fait rien.*

La condition portant sur une variable peut être :

- Une valeur à atteindre.
- Une comparaison avec une autre variable (égalité, inégalité, non égalité)
- Autre valeur

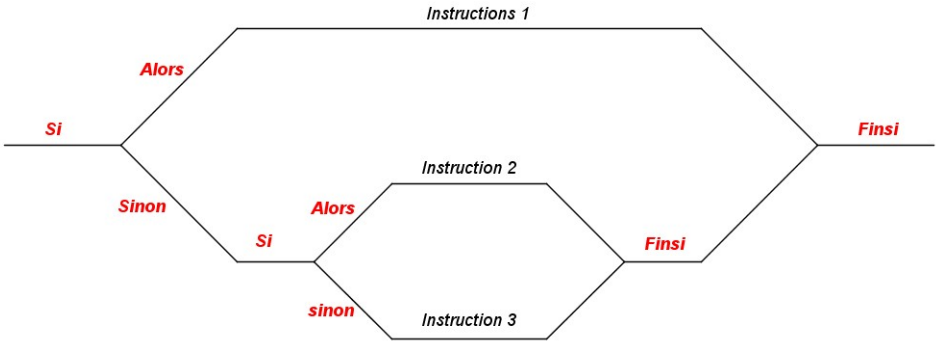
On peut aussi mettre un test qui se décompose en plusieurs conditions reliées par un opérateur logique :

- condition 1 ET condition 2 : les deux conditions doivent être vérifiées en même temps.
- condition 1 OU condition 2 : l'une au moins des deux conditions doivent être vérifiées.

On peut aussi imbriquer un ou plusieurs autres tests à l'intérieur d'un test. On a alors le schéma suivant :

```
Si condition 1 alors
    Instructions 1
Sinon                    Si condition 2 alors
                             Instructions 2
                         Sinon
                             Instruction 3
                         Finsi
Finsi
```

On pourrait schématiser cette situation par :



5.1 Exercice

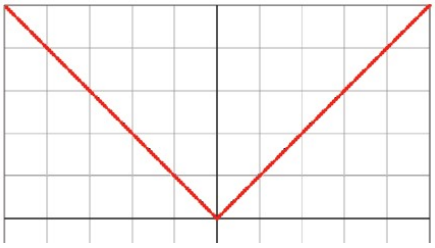
On donne ci-dessous l'algorithme associée à la valeur absolue, "$abs()$" de votre calculatrice.

```
1   VARIABLES
2     x EST_DU_TYPE NOMBRE
3     A EST_DU_TYPE NOMBRE
4   DEBUT_ALGORITHME
5     LIRE x
6     SI (x>=0) ALORS
7       DEBUT_SI
8       A PREND_LA_VALEUR x
9       FIN_SI
10      SINON
11        DEBUT_SINON
12        A PREND_LA_VALEUR -x
13        FIN_SINON
14    AFFICHER A
15  FIN_ALGORITHME
```

1) Tester cet algorithme pout -2, 3 et 0.
2) Représenter graphiquement cette fonction à l'aide de votre calculatrice.

1) On trouve facilement : 2,3 et 0.

2) Voici la courbe grâce à un algorithme d'Algobox



6 Les boucles

6.1 Définition

Définition 4 : Une **boucle** est une structure répétitive ou itérative, c'est à dire que la boucle effectue n fois un calcul sous le contrôle d'une condition d'arrêt.

6.2 La boucle simple

La boucle simple obéit au schéma suivant :

```
Tant que   condition
      Instructions
FinTantque
```

Dans le cas où la condition est toujours vérifiée, l'ordinateur tourne alors dans la boucle et n'en sort plus. La « **boucle infinie** » est une des hantises les plus redoutées des programmeurs.

Cette faute de programmation est courante chez tout apprenti programmeur.

6.3 Exercice

Soit un algorithme permettant de trouver la partie entière d'un nombre positif. On rappelle que la partie entière n d'un nombre x est définie comme suit :

$$n \leqslant x < n+1$$

```
1    VARIABLES
2      x EST_DU_TYPE NOMBRE
3      N EST_DU_TYPE NOMBRE
4    DEBUT_ALGORITHME
5      AFFICHER "Saisir un nombre réel positif :"
6      LIRE x
7      AFFICHER x
8      N PREND_LA_VALEUR 0
9      TANT_QUE (N+1<=x) FAIRE
10       DEBUT_TANT_QUE
11       N PREND_LA_VALEUR N+1
12       FIN_TANT_QUE
13     AFFICHER N
14   FIN_ALGORITHME
```

1) Tester cet algorithme avec le nombre $x = 4,3$, en écrivant tous les résultats par boucle.

2) Trouver un algorithme qui permette de calculer la partie entière d'un nombre quelconque (positif ou négatif).

1) La valeur de N au début vaut 0 donc $N + 1 = 1$

   1^{er} test $4,3 \geqslant 1$ donc $N \longleftarrow 1$

   2^{e} test $4,3 \geqslant 2$ donc $N \longleftarrow 2$

   3^{e} test $4,3 \geqslant 3$ donc $N \longleftarrow 3$

   4^{e} test $4,3 \geqslant 4$ donc $N \longleftarrow 4$

   5^{e} test $4,3 \leqslant 5$ donc on affiche 4

2) La définition de la partie entière est la même pour un nombre négatif. Il ne faut pas donc confondre partie entière et troncature en effet : la partie entière de $-2,5$ est -3 alors que sa troncature est -2.

   Pour écrire un algorithme à partir de l'ancien est intéressant de passer par la valeur absolue $abx()$ (vue plus haut). Il faut donc introduire une nouvelle variable A qui correspond à la valeur absolue de x. Pour l'affichage, comme la partie entière est différente de la troncature, pour $x < 0$, il faut afficher $-N - 1$. On obtient alors l'algorithme suivant :

```
1   VARIABLES
2     x EST_DU_TYPE NOMBRE
3     N EST_DU_TYPE NOMBRE
4     A EST_DU_TYPE NOMBRE
5   DEBUT_ALGORITHME
6     AFFICHER "Saisir un nombre réel :"
7     LIRE x
8     AFFICHER x
9     N PREND_LA_VALEUR 0
10    A PREND_LA_VALEUR abs(x)
11    TANT_QUE (N+1<=A) FAIRE
12      DEBUT_TANT_QUE
13      N PREND_LA_VALEUR N+1
14      FIN_TANT_QUE
15    SI (x<0) ALORS
16      DEBUT_SI
17      N PREND_LA_VALEUR -N-1
18      AFFICHER N
19      FIN_SI
20      SINON
21        DEBUT_SINON
22        AFFICHER N
23        FIN_SINON
24  FIN_ALGORITHME
```

6.4 Boucler en comptant

Si l'on connaît à l'avance le nombre de d'incrémantation, on a alors la structure suivante :

Pour compteur = initial **à** final **par** valeur du pas
    Instructions
FinPour

Si l'on ne connaît pas à l'avance le nombre d'itération, on ajoute la structure avzec tant que :

Pour compteur = initial **par** valeur du pas **Tant que** condition
    Instructions
FinPour

6.5 Exercice

On considère l'algorithme suivant :

Variables
N, i, S
Algorithme
Afficher « Saisisser un nombre entier N : »
Saisir N
S reçoit la valeur 1
Pour i de 1 jusqu'à N
S reçoit $S \times i$
FinPour
Afficher S

1) Tester cette algorithme pour $N = 5$ en donnant les résultats à chaque itération.
2) Pourquoi l'initialisation est-elle importante.
3) Ecrire cet algorithme avec le logiciel Algobox ou votre calculatrice.

1) On trouve comme résutat :

i	0	1	2	3	4	5
S	1	1	2	6	24	120

2) L'initialisation est important ($S = 1$) car si l'on oublit cette ligne la valeur par défaut de S est 0, ce qui donnera un résultat nul à chaque itération.
3) Voici l'alorithme avec Algobox :

```
1   VARIABLES
2     N EST_DU_TYPE NOMBRE
3     i EST_DU_TYPE NOMBRE
4     S EST_DU_TYPE NOMBRE
5   DEBUT_ALGORITHME
6     LIRE N
7     S PREND_LA_VALEUR 1
8     POUR i ALLANT_DE 1 A N
9       DEBUT_POUR
10      S PREND_LA_VALEUR S*i
11      FIN_POUR
12    AFFICHER S
13  FIN_ALGORITHME
```