Dans ce chapitre, nous présentons des notions de base sur l'algorithmique, c-à-d comment écrire un algorithme qui résout un problème donné tout, à savoir : utiliser les actions de base (affectation, lecture, écriture), définir les notions de variable et constante, définir le types d'une variable et déclarer une variable, …

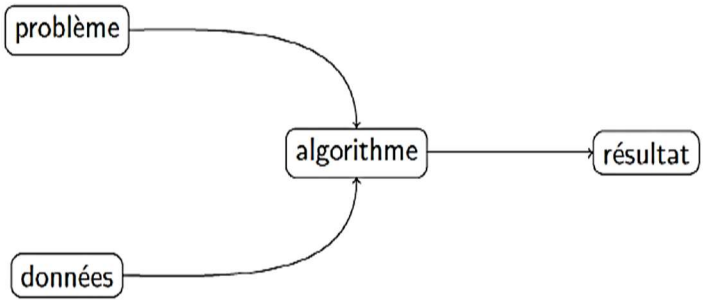
---

1. Notion d'algorithme

Un programme informatique permet à l'ordinateur de résoudre un problème, mais avant de communiquer à l'ordinateur comment résoudre ce problème, il faut en premier lieu pouvoir le résoudre nous même → **faire un algorithme**.

1.1. **L'algorithmique** désigne la discipline qui étudie les algorithmes et leurs applications en informatique

1.2. **Un algorithme** est une méthode permettant de résoudre un problème donné en un temps fini; le mot algorithme vient du nom du célèbre mathématicien arabe Al Khawarizmi (Abu Ja'far Mohammed Ben Mussa Al-Khwarismi)



1.3. **Le programme** ne sera que la traduction de l'algorithme dans un langage de programmation, c'est-à-dire, un langage plus simple que le français dans sa syntaxe, sans ambiguïtés, que la machine peut utiliser et transformer pour exécuter les actions qu'il peut décrire. Pascal, C, Java et Visual Basic sont des noms de langages de programmation.

1.4. Algorithme et programme

- L'élaboration d'un algorithme précède l'étape de programmation
- Un programme est un algorithme
- Un langage de programmation est un langage compris par l'ordinateur
- L'élaboration d'un algorithme est une démarche exigeante de résolution de problème
- La rédaction d'un algorithme est un exercice de réflexion qui se fait sur papier
- L'algorithme est indépendant du langage de programmation, par exemple, on utilisera le même algorithme pour une implantation en Java, ou bien en C++ ou en Visual Basic
- L'algorithme est la résolution brute d'un problème informatique.

1.5. Représentation d'un algorithme

Historiquement, deux façons pour représenter un algorithme :
L'Organigramme : représentation graphique avec des symboles (carrés, losanges, etc.)

- Offre une vue d'ensemble de l'algorithme
- Représentation quasiment abandonnée aujourd'hui

Le pseudo-code : représentation textuelle avec une série de conventions ressemblant à un langage de programmation (il s'agit d'un langage informel proche du langage naturel et indépendant de tout langage de programmation).

- Il est plus pratique pour écrire un algorithme
- Sa représentation est largement utilisée

1.6. Etapes de résolution d'un problème (Algorithme)

1. Comprendre l'énoncé du problème
2. Décomposer le problème en sous-problèmes plus simple à résoudre
3. Associer à chaque sous problème, une spécification :
   - Les données nécessaires
   - Les données résultantes
1. La démarche à suivre pour arriver au résultat en partant d'un ensemble de données.
2. Elaboration d'un algorithme

1.7. Réalisation d'un programme

La résolution d'un problème donné passe par une succession d'étapes à savoir :
Problème → Enoncé explicite → Algorithme → Programme
Durant l'écriture d'un programme, on peut être confronté à 2 types d'erreur :

- **Les erreurs syntaxiques** : elles se remarquent à la compilation et sont le résultat d'une mauvaise écriture dans le langage de programmation.
- **Les erreurs sémantiques** : elles se remarquent à l'exécution et sont le résultat d'une mauvaise analyse. Ces erreurs sont beaucoup plus graves car elles peuvent se déclencher en cours d'exploitation du programme.

1.8. Langages de programmation (langages informatiques)

Un langage de programmation est un code de communication, permettant à un être humain de dialoguer avec une machine en lui soumettant des instructions et en analysant les données matérielles fournies par le système. Le langage de programmation est l'intermédiaire entre le programmeur et la machine. Il permet d'écrire des programmes (suite consécutive d'instructions) destinés à effectuer une tache donnée.
On distingue deux types de langages :

- Langages procéduraux : Fortran, Cobol, Pascal, C, …
- Langages orientés objets : C++, Java, C#,…

Le choix d'un langage de programmation n'est pas facile, chacun a ses spécificités et correspond mieux à certains types d'utilisations.

1.9. Bases d'un langage algorithmique

1.9.1. Structure de Base

En pseudo-code la structure générale d'un algorithme est la suivante :

Algorithme nom_de_l'algorithme

CONST {Définition des constantes}
TYPE {Définition de types}
VAR {Déclaration de variables}

} ils sont facultatifs (s'il n'y a pas de var ou const ou type on ne les écrit pas)

DEBUT

{Suite d'instructions}

FIN.

- **Partie en-tête** : c'est la première ligne de l'algorithme, elle commence par le mot algorithme suit d'un espace puit le nom de l'algorithme.
- **Partie déclarations** : dans cette partie, toutes objets utilisés dans la résolution du problème doivent êtres déclarés.
- **Partie traitement** : cette partie commence par le mot Début et se termine par le mot Fin, elle contient les actions utilisées dans la résolution du problème.

1.9.2. Instructions de base

Un algorithme est formé de quatre types d'instructions considérées comme des petites briques de base :

1. L'affectation de variables
2. La lecture et/ou l'écriture
3. Les tests
4. Les boucles

Avant de décrire ces instructions, nous présentons d'abord la notion de variable et constante.

1.9.3. Constantes et variables

On sépare les objets en deux classes : les constantes et les variables.

1.9.3.1. Notion de variable

Les données manipulées dans un algorithme sont appelées des variables. Une variable sert à stocker la valeur d'une donnée dans un langage de programmation. Elle désigne un emplacement mémoire dont le contenu peut changer au cours d'un programme (d'où le nom de variable).
Chaque emplacement mémoire a un numéro qui permet d'y faire référence de façon unique : c'est l'adresse mémoire de cette cellule.
Règle : La variable doit être déclarée avant d'être utilisée, elle doit être caractérisée par :

- Un nom (Identificateur)
- Un type qui indique l'ensemble des valeurs que peut prendre la variable (entier, réel, booléen, caractère, chaîne de caractères, …)
- Une valeur

Identificateurs : règles
Le choix du nom d'une variable est soumis à quelques règles qui varient selon le langage, mais en général :

- Un nom doit commencer par une lettre alphabétique. Exemple : E1 (1E n'est pas valide)

- Doit être constitué uniquement de lettres, de chiffres et du soulignement (« _ ») (Éviter les caractères de ponctuation et les espaces). Exemples : SMI2008, SMI_2008. (SMP 2008, SMP-2008, SMP;2008 : sont non valides)
- Doit être différent des mots réservés du langage (par exemple en C: int, float, double, switch, case, for, main, return, …)
- La longueur du nom doit être inférieure à la taille maximale spécifiée par le langage utilisé (en c, le nom doit faire au plus 32 caractères).
- En c, Les majuscules sont distinguées des minuscules : a et A sont deux noms différents.

Conseil : pour la lisibilité du code choisir des noms significatifs qui décrivent les données manipulées. Exemples : NoteEtudiant, Prix_TTC, Prix_HT

Remarque : en pseudo-code algorithmique, on va respecter les règles citées, même si on est libre dans la syntaxe

Types des variables

Le type d'une variable détermine l'ensemble des valeurs qu'elle peut prendre. Les types offerts par la plupart des langages sont :

- Type numérique (entier ou réel)
- Byte (codé sur 1octet): de $[-2^7,2^7[$ ou $[0, 2^8[$
- Entier court (codé sur 2 octets) : $[-2^{15},2^{15}[$
- Entier long (codé sur 4 octets): $[-2^{31},2^{31}[$
- Réel simple précision (codé sur 4 octets) : précision d'ordre 10^{-7}
- Réel double précision (codé sur 8 octets) : précision d'ordre 10^{-14}
- Type logique ou booléen : deux valeurs VRAI ou FAUX
- Type caractère : lettres majuscules, minuscules, chiffres, symboles, ... Exemples : 'A', 'b', '1', '?', …
- Type chaîne de caractère : toute suite de caractères. Exemples : " " , " Nom, Prénom", "code postale: 1000", …

A) Entier : Pour représenter les nombres entiers, les opérations utilisables sur les entiers sont :

- Toutes les opérations élémentaires de bases sont permises : + , - , * , /
- Les opérateurs de comparaison classiques : <, >, =, ...
- La division **/** est la division euclidienne (ou entière). Ex : 11 / 4 = 2 et non pas 2.75 !)
- Il existe l'opérateur modulo **%**, qui donne le reste de la division euclidienne. Ex : 11%4 = 3

B) Réel : Pour représenter les nombres réels, les opérations utilisables sur les réels sont :

- Les opérations arithmétiques classiques : + (addition), - (soustraction), * (produit), / (division)
- Les opérateurs de comparaison classiques : <, >, =, ...
- La division / donne un résultat décimal
- L'opérateur modulo % n'existe pas

C) Booléen : Une variable de type logique (booléen) peut prendre deux valeurs : VRAI ou FAUX. Les opérations principales les plus utilisées sont :

• Les opérateurs logiques : NON, ET, OU

Tables de vérité :

A	B	A ET B	A OU B
FAUX	FAUX	FAUX	FAUX
FAUX	VRAI	FAUX	VRAI
VRAI	FAUX	FAUX	VRAI
VRAI	VRAI	VRAI	VRAI

A	NON A
FAUX	VRAI
VRAI	FAUX

• Opérateurs de comparaison : =, ≤, ≥, ≠

D) Caractère

Il s'agit du domaine constitué des caractères alphabétiques et numériques. Une variable de ce type ne peut contenir qu'un seul et unique caractère. Les opérations élémentaires réalisables sont les comparaisons : <, >, =, ...

E) Chaine de caractère

Une chaine de caractère est un objet qui peut contenir plusieurs caractères de manière ordonnée.

Déclaration des variables

Rappel : toute variable utilisée dans un programme doit avoir fait l'objet d'une déclaration préalable
En pseudo-code, la déclaration de variables est effectuée par la forme suivante :

Variables liste d'identificateurs : type

Exemple :
Variables i, j, k : entier
x, y : réel
OK : booléen
Ch1, ch2 : chaîne de caractères

En C : types de base :
- char (caractères) Par exemple : 'a'...'z', 'A'...'Z',...
- int (entiers) Par exemple : 129, -45, 0, ...
- float (réels) Par exemple : 3.14, -0.005, 67.0, ...

De façon générale leur déclaration est comme suit :
<type> <nom> ;
<type> <nom1>,<nom2>,<nom3> ;
Exemples :

```
int a;
float valeur, res;
char a,b,c ;
```

Déclarer une variable revient à lui réserver un emplacement mémoire. On ne connait pas sa valeur initiale !

1.9.3.2. Notion de constante

Une constante est une variable dont la valeur ne change pas au cours de l'exécution du programme, elle peut être un nombre, un caractère, ou une chaine de caractères. En pseudo-code,
Constante identificateur=valeur (par convention, les noms de constantes sont en majuscules)
Exemple : pour calculer la surface des cercles, la valeur de pi est une constante mais le rayon est une variable.

- Constante PI=3.14, MAXI=32
- Une constante doit toujours recevoir une valeur dès sa déclaration.

En C :

1. Les constantes littérale : Exp. Int a=10 ; float tax_rate = 0.28;
2. Les constantes symboliques
   2.1. #define PI 3.14159 (#define peut se trouver n'importe où dans le code source, mais son effet est limité à la partie de code qui la suit. En général, les programmeurs groupent tous les #define ensemble, au début du fichier, avant la fonction main()).
   2.2. const float pi = 3.14159;

1.9.4. Opérations de base

Dans ce qui suit nous décrivons la liste des opérations de base pouvant composer un algorithme. Elles sont décrites en pseudocode (pseudolangage).

1.9.4.1. Affectation

L'affectation consiste à attribuer une valeur à une variable (c'est-à-dire remplir ou modifier le contenu d'une zone mémoire). En pseudo-code, l'affectation est notée par le signe ←

- Var← e : attribue la valeur de e à la variable Var
- e peut être une valeur, une autre variable ou une expression
- Var et e doivent être de même type ou de types compatibles

L'affectation ne modifie que ce qui est à gauche de la flèche

Exemples 1 : i ←1, j ←i , k ←i+j , x ←10.3 , OK ←FAUX , ch1 ←"SMI" , ch2 ←ch1 , x ←4 , x ←j
(avec i, j, k : entier; x :réel; ok :booléen; ch1,ch2 :chaine de caractères)
Exemples non valides : i ←10.3 , OK ←"SMI" , j ←x

Exemples 2
a ←10 a reçoit la constante 10
a ← (a*b)+c a reçoit le résultat de (a*b)+c
d ←'m' d reçoit la lettre m

Remarques

- Le langage de programmation C utilisent le signe égal = pour l'affectation ←.
- Lors d'une affectation, l'expression de droite est évaluée et la valeur trouvée est affectée à la variable de gauche. Ainsi, A←B est différente de B←A
- L'affectation est différente d'une équation mathématique :
- Les opérations x ← x+1 et x ← x-1 ont un sens en programmation et se nomment respectivement incrémentation et décrémentation.
- A+1 ← 3 n'est pas possible en langages de programmation et n'est pas équivalente à A ← 2
- Certains langages donnent des valeurs par défaut aux variables déclarées. Pour éviter tout problème il est préférable d'initialiser les variables déclarées.

Exemple en C

```
int n;
int p;
n = 10;
p = 2*n-3;
```

Quelques opérateurs bien pratiques

- i + + : opérateur permettant d'ajouter une unité à la variable i (de type int ou char)
- i - - : idem mais pour retirer une unité
- x* = y, x/ = y, x- = y, x+ = y : opérateurs permettant de multiplier (diviser, soustraire ou ajouter) x par y (pas de restriction de type)

1.9.4.2. La lecture

Lire(variable)

Cette opération permet d'attribuer à une variable une valeur introduite au moyen d'un organe d'entrée (généralement le clavier).

Exemples de lecture
Lire(a) On demande à l'utilisateur d'introduire une valeur pour a
Lire(a,b,c) On demande à l'utilisateur d'introduire 3 valeurs pour a, b et c respectivement

En C :
Saisie avec scanf.

- Permet de lire sur l'entrée standard (le clavier)
- scanf(''<code format>'', &<variable>); avec <code format> = %d, %f ou %c pour lire un entier, un flottant ou un caractère.

Exemples

- int n; scanf(''%d'', &n);
- float x; scanf(''%f'', &x);
- char a ; scanf((''%c'', &a);

Remarque : Le programme s'arrête lorsqu'il rencontre une instruction Lire et ne se poursuit qu'après la saisie de l'entrée attendue par le clavier et de la touche Entrée (cette touche signale la fin de l'entrée)
Conseil : Avant de lire une variable, il est fortement conseillé d'écrire des messages à l'écran, afin de prévenir l'utilisateur de ce qu'il doit frapper

1.9.4.3. L'écriture

Ecrire(expression)
Elle communique une valeur donnée ou un résultat d'une expression à l'organe de sortie.
Exemples d'écriture

Ecrire('bonjour')	Affiche le message bonjour (constante)
Ecrire(12)	Affiche la valeur 12
Ecrire(a,b,c)	Affiche les valeurs de a, b et c
Ecrire(a+b)	Affiche la valeur de a+b
Ecrire(a, b+2, "Message")	Affiche la valeur de a, puis la valeur de l'expression b+2 et enfin le mot ''message'' .

Exemples en C

- printf(''hello nn '');
- int x=10; printf(''valeur de x = %dnn '',x);
- int x=10; float y = 3.4; printf(''valeur de x = %d et de y+x = %fnn '',x,y+x);

Principaux codes de format

- %d pour le type int
- %c pour le type char
- %f pour le type float

Caractères de mise en forme

- /n : retour à la ligne
- /r : retour en début de ligne courante
- /t : tabulation

1.9.5. Syntaxe générale de l'algorithme

Algorithme exemple
Constantes var1=20 , var2="bonjour!"
Variables
var3, var4 : réels
var5 : caractère
Début // corps de l'algorithme
Instructions 1
Instructions 2
…….
Instructions n
Fin

Notes :

- Les instructions d'un algorithme sont habituellement exécutées une à la suite de l'autre, en séquence (de haut en bas et de gauche à droite).
- L'ordre d'exécution est important
- On ne peut pas changer cette séquence de façon arbitraire

1.9.6. Expressions et opérateurs

Une expression peut être une valeur, une variable ou une opération constituée de variables reliées par des opérateurs
Exemples : 1, b, a*2, a+ 3*b-c, …
L'évaluation de l'expression fournit une valeur unique qui est le résultat de l'opération
Les opérateurs dépendent du type de l'opération, ils peuvent être :

- Des opérateurs arithmétiques : +, -, *, /, % (modulo), ^(puissance)
- Des opérateurs logiques : NON(!), OU(| |), ET (&&)
- Des opérateurs relationnels : =, <, >, <=, >=
- Des opérateurs sur les chaînes : & (concaténation)
- Une expression est évaluée de gauche à droite mais en tenant compte des priorités des opérateurs.

Remarque :

- On ne peut pas additionner un entier et un caractère
- Toutefois dans certains langages on peut utiliser un opérateur avec deux opérandes de types différents, c'est par exemple le cas avec les types arithmétiques (4 + 5.5)
- La signification d'un opérateur peut changer en fonction du type des opérandes, par exemple
  - L'opérateur + avec des entiers effectue l'addition, 3+6 vaut 9
  - Avec des chaînes de caractères il effectue la concaténation, "bonjour" + " tout le monde" vaut "bonjour tout le monde"

1.9.7. Priorité des opérateurs

Pour les opérateurs arithmétiques donnés ci-dessus, l'ordre de priorité est le suivant (du plus prioritaire au moins prioritaire) :

- () : les parenthèses
- ^ : (élévation à la puissance)
- *, / (multiplication, division)
- % (modulo)
- +, - (addition, soustraction)

Exemple : 9 + 3 * 4 vaut 21

- En cas de besoin, on utilise les parenthèses pour indiquer les opérations à effectuer en priorité

Exemple : (9 + 3) * 4 vaut 48

- À priorité égale, l'évaluation de l'expression se fait de gauche à droite

1.9.8. Les opérateurs booléens

Associativité des opérateurs et et ou a et (b et c) = (a et b) et c
Commutativité des opérateurs et et ou a et b = b et a a ou b = b ou a
Distributivité des opérateurs et et ou a ou (b et c) = (a ou b) et (a ou c) a et (b ou c) = (a et b) ou (a et c)
Involution (homographie réciproque) : non non a = a _ Loi de Morgan : non (a ou b) = non a et non b non (a et b) = non a ou non b

Exemple : soient a, b, c et d quatre entiers quelconques :
(a<b)| |((a>=b)&&(c==d)) ⇔ (a<b)| |(c==d) car (a<b)| |(!(a<b)) est toujours vraie

Exercice 1 :

Quel est le type de chaque variable :

A=1, B=vrai, test= 12.23, specialite='m',

Exercice 2 :

Soient A, B deux variables de types entiers, C, D deux variables de type réel, E, F deux variables de type booléen.

Quel est le type des variables suivants : A1, B1, C1, A2, B2, C2, D2, A3, B3, C3, D3

A1 ← A+B ; B1←A*B; C1←A/B;

A2←C+D; B2←C*D; C2←C/D; D2←A*C;

A3←E ou F; B3←E et F; C3←(A>B) ; D3←Faux ;

Exercice 3 :

Quel sont les identificateurs valides et ceux qui ne sont pas valides :

A, cA, 12, 1A, A1, A12m, bougie, test?, SI .

Exercice 4 :

Donnez les valeurs des variables A, B et C après exécution des instructions suivantes ?

Variables A, B, C : Entier

Début

A ← 7
B ← 17
A ← B
B ← A+5
C ← A + B
C ← B – A

Fin

Exercice 5 :

Donnez les valeurs des variables A et B après exécution des instructions suivantes ?

Variables A, B : Entier

Début

A ← 6
B ← 2
A ← B
B ← A

Fin

Les deux dernières instructions permettent-elles d'échanger les valeurs de A et B ?

Exercice 6 :

Écrire un algorithme permettant d'échanger les valeurs de deux variables A et B ?

Exercice 7 :

Écrire un algorithme qui permet d'effectuer la saisie d'un nom, d'un prénom et affiche ensuite le nom complet.

Exercice 8 :

Écrire un algorithme qui demande un nombre entier à l'utilisateur, puis qui calcule et affiche le carré de ce nombre.

Exercice 9

On dispose de trois variables A, B et C. Ecrivez un algorithme transférant à B la valeur de A, à C la valeur de B et à A la valeur de C (quels que soient les contenus préalables de ces variables).

Exercice 10 :

Ecrire un programme qui permet d'introduire un nombre et d'afficher son double ainsi que sa moitié.