

## Table des matières

3- Algorithmique.....	1
3.1 Définition : algorithmique .....	1
Critère algorithmique élémentaire.....	1
3.2 De l' algorithme au programme.....	2
Définition : programme :.....	2
3.3 Importance des algorithmes :.....	3
3.3.1 Méthodologie simple.....	3
3.4 Langage algorithmique .....	4
3.4.1 Données :.....	4
3.4.2 Constantes :.....	5
3.4.3 Variable: .....	5
3.4.4 Entrées / Sorties.....	5
3.4.5 Opérateurs.....	5
3.4.6 Structures.....	7

## 3- Algorithmique

### 3.1 Définition : algorithmique

#### *Critère algorithmique élémentaire*

Une application courante ou un problème est automatisable (traitable par informatique) si

- il est possible de définir et décrire parfaitement les données et les résultats de sortie
- il est possible de décomposer le passage de ces données vers ces résultats en une suite finie d'opérations élémentaires dont chacune peut être exécutée par une machine

L'algorithmique est la transformation de la connaissance que l'humain possède sur un problème en actions élémentaires exécutées par l'ordinateur

*“Ensemble de règles opératoires dont l'application permet de résoudre un problème au moyen d'un nombre fini d'opérations (ou actions)”*

**Exemple 1:** Fabrication d'un pain la « machine » réalisant ces actions élémentaires n'est bien sur pas un ordinateur !

Entrées : farine, eau, sel, levure      Sortie : pain cuit

Opérations séquentielles

- Battre ensemble les ingrédients
- Faire monter la pâte 1h à 25°C
- Faire cuire 20mn à 200°C

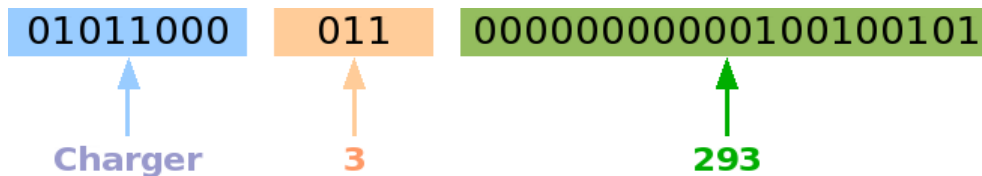
## 3.2 De l' algorithme au programme

### **Définition : programme :**

Codage d'un algorithme afin que l'ordinateur puisse exécuter les actions décrites dans l'algorithme. Ce dernier doit être écrit dans un langage « **compréhensible** » par l'ordinateur dit langage de programmation (Assembleur (micropro), C, Fortran, Pascal, Cobol ...).

A la conception d'un ordinateur, est défini l'ensemble des opérations élémentaires qu'il peut réaliser. Ces opérations doivent être les plus simples possible pour diminuer la complexité des circuits électroniques. L'ensemble des opérations élémentaires est appelé **langage machine (ou assembleur)**.

Un programme en "code-machine" est une suite d'instructions élémentaires, composées uniquement de 0 et de 1, exprimant les opérations de base que la machine peut physiquement exécuter: instructions de calcul (addition, ...) ou de



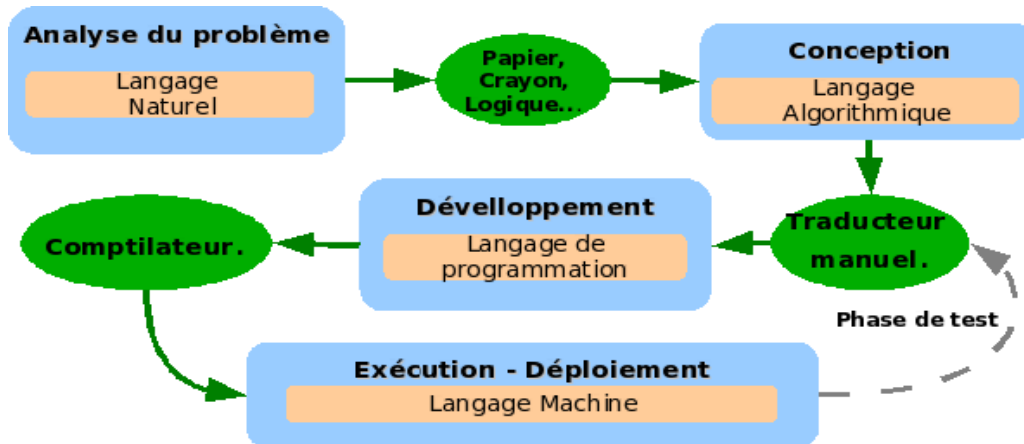
**Schéma** : Exemple d'instruction en langage machine  
(Charger le nombre 293 dans le registre 3)

traitement ("et" logique, ...), instructions d'échanges entre la mémoire principale et l'unité de calcul ou entre la mémoire principale et une mémoire externe, des instructions de test qui permettent par exemple de décider de la prochaine instruction à effectuer.

Voici en code machine de l'IBM 370 l'ordre de charger la valeur 293 dans le *registre* "3":

Ce type de code binaire est le seul que la machine puisse directement comprendre et donc réellement exécuter.

Tout programme écrit dans un langage évolué devra par conséquent être d'abord traduit en code-machine avant d'être exécuté.



**Schéma 3.2** : Cycle de vie d'un logiciel

### 3.3 Importance des algorithmes :

Permettent de formaliser le problème et de déterminer et de perfectionner les mécanismes nécessaires à sa résolution.

Plusieurs propriétés sont inhérentes à un algorithme donné et permettent de définir son utilité :

- - **La calculabilité des algorithmes** (convergence de l'algorithme)
- - **La complexité des algorithmes** (nombre d'opérations nécessaires)
- - **L'efficacité des algorithmes** (vitesse des algo: raisonnable) : temps d'exécution et taux d'occupation de la mémoire centrale

#### 3.3.1 Méthodologie simple

Suivre la démarche suivante :

1. *Définir* clairement le problème (Analyse de l'énoncé, associer aux éléments de cet énoncé des actions simples)

2. Chercher une *méthode de résolution* (formules...)
3. Définir les *entrées* nécessaires et les *résultats* obtenus
4. Écrire l'algorithme (**langage algorithmique**)

Si le problème est trop complexe, réaliser une **analyse méthodique descendante** :

- **décomposer** en sous-problèmes,
- appliquer, pour chacun, la méthodologie simple

Décomposer un problème revient à en fournir une *description de + en + détaillée* jusqu'à obtenir un *ensemble d'opérations élémentaires* traductibles en langage de programmation

Exemple: Ecrire un algorithme calculant la circonférence d'un cercle de rayon 12

- 1) Définir clairement l'énoncé : relever les actions de cet énoncé (=> calculant) ainsi que les attributs associés à ces actions (=> **circonférence**, **cercle**, **r=12**)
- 2) Chercher une méthode de résolution : à partir des éléments de l'énoncé, définir la marche à suivre pour résoudre le problème ( rappel de la formule de calcul de la circonférence d'un cercle  $C = 2*PI*R$ )
- 3) Définir les entrées nécessaires et les résultats obtenus (préciser la nature de ces entrées et sorties)

<b>Entrées</b>	<b>Sortie</b>
<ul style="list-style-type: none"> <li>● PI (réel) (&lt;!=&gt; <i>facultatif</i> &lt;!=&gt;)</li> <li>● R rayon du cercle (entier ou réel)</li> </ul>	Circonférence du cercle (réel)

Reste l'écriture de l'algorithme : nous allons aborder les éléments et formalismes que vous devez posséder pour réaliser cette écriture.

### 3.4 Langage algorithmique

Peut se définir comme une boîte à outil avec un mode d'emploi permettant d'ajouter de la rigueur et de la lisibilité à un traitement donné.

On distingue plusieurs catégories d'éléments :

#### 3.4.1 Données :

Éléments introduits dans la méthode par l'utilisateur (au clavier ou dans un fichier). Ils sont désignés par un **identificateur** et possèdent un **type** clairement définies ((facultatif) peuvent également être des méthodes )

#### 3.4.2 Constantes :

Element dont la valeur, utilisée dans le programme, ne change pas lors de l'exécution (par ex. PI). Tout comme la donnée, cet élément est désigné par un identificateur qui lui est propre et possède un type clairement défini qui ne varie pas tout au long du programme.

### 3.4.3 Variable:

Element dont la valeur, utilisée dans le programme, peut changer lors de l'exécution. Cette variable, elle aussi désignée par un identificateur, se réfère à un emplacement précis en mémoire centrale

De façon plus générale, il est indispensable de:

- Définir la nature des données, constantes et variables
- Choisir les identificateurs les plus appropriés
- Pouvoir référencer tous les genres de valeurs (entier, reel, caractère...)

### 3.4.4 Entrées / Sorties

\* L'échange de données entre programme et utilisateur

(On peut éventuellement s'aider d'un schéma)

**Lire** : pour recevoir de l'information de l'extérieur

- **Lire (x)** où x est une variable qui va prendre la valeur donnée par l'utilisateur au clavier

**Écrire** : pour fournir de l'information à l'extérieur

- **Écrire (x)** : la valeur contenue dans la variable x sera affichée

Exemple : Écrire ('Bonjour') : Bonjour sera écrit à l'écran

### 3.4.5 Opérateurs

\* L'affectation :

Opération qui consiste à attribuer une valeur donnée à une variable.donnée. Cette valeur peut être une donnée simple ou le résultat d'une opération ou d'une méthode.

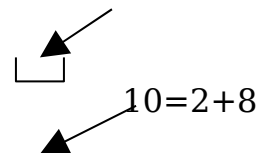
Se note : variable <- valeur

Exemples:

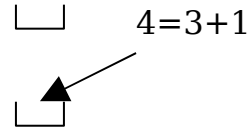
x <- 1.2

(x prend la valeur 1.2)

x



n <- 2+8      (n prend la valeur 10)      n



i <- i+1      (i est incrémenté de 1)      i

*Représenter les variables par ces petites boîtes avec leur nom, leur type pour tous les algos et programmes qui suivent et expliquer que chaque boîte occupe un espace mémoire dont la taille dépend du type de la variable.*

2 classes d'opérateurs : arithmétiques et logiques

\* Opérateurs arithmétiques

Opérateurs classiques que l'on retrouve sur l'espace des réels :

- addition « + »
- soustraction « - »
- multiplication « \* »
- division « / »
- modulo (reste de la div euclidienne) « % »

Attention toutefois aux types : (division : (entier)/(entier) => (entier) (3/2 => 1))

\* Opérateurs logiques

Opérateurs logiques normaux : ET, OU, NON + opérateurs de comparaison ( =, >, <, >=, <= ).

### 3.4.6 Structures

\* Structure globale d'un algorithme

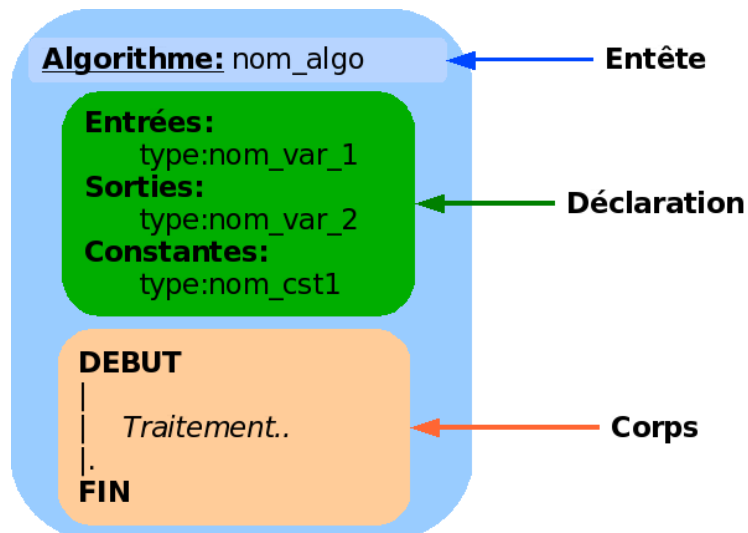


Schéma 3.3 : Formalisme d'écriture d'un algorithme

Exemple :

Retour sur le calcul de la circonférence d'un cercle de rayon 12

Rappel des entrées et sorties :

<b>Entrées</b>	<b>Sortie</b>
<ul style="list-style-type: none"> <li>● Pi (réel) (&lt;!-- facultatif --&gt;)</li> <li>● R rayon du cercle (entier ou réel)</li> </ul>	Circonférence du cercle (réel)

Solution 1: solution simple

algorithme	Programme C
DEBUT Écrire (2*3.14159*12) FIN	<pre>#include &lt;stdio.h&gt; main () {     printf(" %0.2f\n ", 2*3.14159*12) ; }</pre>

## Solution 2: Notion de variable x

algorithme	Programme C
Variables : Réel : x  DEBUT Ecrire ('tapez la valeur du rayon') Lire (x) Écrire (2*3.14159*x) FIN	<pre>#include &lt;stdio.h&gt; main () {     float x ;     printf("tapez la valeur du rayon\n") ;     scanf(" %f ",&amp;x) ;     printf(" %0.2f\n ", 2*3.14159*x) ; }</pre>

## Solution 3: Utilisation de trois variables réelles

algorithme	Programme C
Variables : Réel : rayon, Pi, Perimetre DEBUT Pi <- 3.14159 Écrire ('Entrer la valeur du rayon:') Lire (rayon) Perimetre <- 2*Pi*rayon Écrire ('la circonférence du cercle est', Perimetre) FIN	<pre>#include &lt;stdio.h&gt; main () {     float Rayon, Pi , Perimetre ;     /* lecture du rayon */     printf(" tapez la valeur du rayon\n") ;     scanf(" %f ",&amp;Rayon) ;     /* initialisation de Pi */     Pi=3.14159 ;     /* calcul du perimetre et écriture */     Perimetre=2*Pi*Rayon ;     printf("%0.2f\n", Perimetre) ; }</pre>

### \* La structure de séquence

suite d'opérations (instructions) exécutées dans l'ordre...Le paquet peut être précédé de DEBUT et suivi de FIN

### \* La structure de sélection simple

**SI** condition **ALORS**

expression1

SINON

expression2

FINSI

Exemple : Recherche du Max de A et B:

algorithme	Programme C
Entrées : réel: A, B Sortie : réel : max  DEBUT  Ecrire('donnez A et B') lire(A,B)  SI A>B ALORS max <- A SINON max <- B FINSI Ecrire('Le maximum est',max)  FIN	<pre>#include &lt;stdio.h&gt; main () {   double A,B,max ;   /* lecture données */   printf(" tapez les valeurs de A et B\n") ;   scanf(" %lf %lf " , &amp;A , &amp;B ) ;   /* calcul du maximum */   if(A &gt; B)     max = A;   else     max = B;   /* écriture*/   printf(" le max de %lf et %lf est   %lf\n " ,A,B,max); }</pre>

**Exercice :** Dérouler et comprendre l'algorithme et le programme en langage C suivants :

Pour les entrées 100.4 , 56 puis - 24.2 , 2.12 puis 3 , 3

Algorithme	Programme C
Entrées : réel: A, B Sortie : réel : max  DEBUT Ecrire('donnez A et B') lire(A,B)  SI A>B ALORS max <- A SINON max <- B FINSI Ecrire('Le maximum est',max)  FIN	<pre>#include &lt;stdio.h&gt; main () {   double A,B,max ;   /* lecture données */   printf(" tapez les valeurs de A et B\n") ;   scanf(" %lf,%lf " , &amp;A , &amp;B ) ;   /* calcul du maximum */   if (A &gt; B)     max = A;   else     max = B;   /* écriture*/   printf(" le max de %lf et %lf est   %lf\n " ,A,B,max); }</pre>

**Exercice :** écrire un algorithme permettant de faire le calcul suivant :  
 En entrée on a deux **entiers**  $x$  et  $y$  , et le programme produit la sortie suivante :  
 $x+y$  si  $x \geq y$  et  $2*x + 2*y$  si  $x < y$

\* *Structure de répétition (nombre connu)*

**POUR** variable **DE** valeur1 **À** valeur2 **FAIRE**  
 Expression  
 FINPOUR

Pour calculer  $X^n$  ( $n > 0$ ) et stocker le résultat dans Y:

Algorithme	Programme C
Entrées : réel : X entier : n Sorties : réel : Y Variables : entier : i  DEBUT Lire (X,n) Y <- 1 POUR i DE 1 À n FAIRE Y <- Y * X FINPOUR  Écrire (X, 'puissance', n, 'vaut', Y) FIN	<pre>#include &lt;stdio.h&gt; main () {   double X,Y; int n , i ;   /* lecture de X et n */   printf(" tapez les valeurs de X et n \n") ;   scanf("%lf %d", &amp;X , &amp;n) ;   /* calcul de X à la puissance n */   Y=1.0 ;   for ( i=1; i&lt;n ;i=i+1)     Y = Y * X;   /* resultat */   printf(" %lf puissance %d vaut %lf \n ",   X, n, Y ); }</pre>

Pour calculer la moyenne de 16 notes:

algorithme	Programme C
<p>Entrées : réel : Note</p> <p>Sorties : réel : Moyenne</p> <p>Variables : réel : Somme entier : i</p> <p>DEBUT</p> <p>Somme &lt;- 0 POUR i DE 1 À 16 FAIRE   Lire (Note)   Somme &lt;- Somme + Note FINPOUR Moyenne &lt;- Somme / 16 Écrire ('La moyenne des 16 notes vaut', Moyenne)</p> <p>FIN</p>	<pre>#include &lt;stdio.h&gt; main () {   double note , moyenne , somme ;   int i ;   /* initialisation de la somme */   somme = 0.0 ;    /* calcul de la somme */   for ( i=1; i&lt;16 ; i++)   {     /* lecture de la note courante */     printf(" donnez la note numéro %d \n ",i) ;     scanf("%lf",&amp;note) ;     /* ajout de la note à somme */     somme = somme + note;   }    /* resultat et sortie du resultat */   moyenne = somme /16 ;   printf(" la moyenne est %lf \n",moyenne) ; }</pre>

\* Structures de répétitions (tant que condition choisie vraie)

<!> Attention aux boucles infinies <!>

**TANT-QUE** condition **FAIRE**  
  expression  
FINTANTQUE

La boucle REPETER...TANTQUE : structure répétitive, nombre de répétition inconnu

**REPETER**  
Expression  
**TANTQUE** condition

peut s'écrire

**REPETER**  
Expression  
**JUSQUA** condition opposée

## Exemple

Pour calculer  $X^n$  ( $n > 0$ ) et stocker le résultat dans Y:

Algorithme	Programme C
<p>Entrées : réel : X entier : n</p> <p>Sorties : réel : Y</p> <p>Variables : entier : i</p> <p>DEBUT Lire (X , n) Y &lt;- 1 i &lt;- 1 TANTQUE i &lt;= n FAIRE   Y &lt;- Y * X   i &lt;- i+1 FINTANTQUE Écrire (X, 'puissance', n, 'vaut', Y)</p> <p>FIN</p>	<pre>#include &lt;stdio.h&gt; main () {   float X,Y;   int n , i ;   /* lecture de X et n */   printf(" tapez les valeurs de X et n \n");   scanf("%lf,%d",&amp;X, &amp;n) ;    /* calcul de X à la puissance n */   Y=1.0 ;   i=1;   while (i &lt;= n)   {     Y = Y * X;     i=i+1;   }   /* resultat */   printf(" %lf puissance %d vaut %lf \n ", X ,n , Y ); }</pre>

Algorithme		Programme C
Entrées : réel : X entier : n Sorties : réel : Y Variables : entier : i		<pre>#include &lt;stdio.h&gt; main () {   float X,Y;   int n , i ;   /* lecture de X et n */   printf(" tapez les valeurs de X et n \n" ) ;   scanf("%lf,%d",&amp;X, &amp;n) ;</pre>
DEBUT Lire (X , n) Y <- 1 i <- 1 REPETER Y <- Y * X i <- i+1 TANTQUE i <= n	DEBUT Lire (X , n) Y <- 1 i <- 1 REPETER Y <- Y * X i <- i+1 JUSQUA i > n	<pre>/* calcul de X à la puissance n */ Y=1.0 ; i=1; do {   Y = Y * X;   i=i+1; } while(i &lt;= n);</pre>
Écrire(X,'puissance', n, 'vaut', Y)	Écrire(X,'puissance', n, 'vaut', Y)	<pre>/* resultat */ printf(" %lf puissance %d vaut %lf \n ", X ,n , Y ) ;</pre>
FIN	FIN	}

## Exercice

Ecrire un algorithme permettant de faire le calcul suivant :

En entrée on a six entiers **sexe, an\_nais, mois\_nais, dept\_nais, x et y** représentant le numéro de sécurité sociale de l'utilisateur ( par exemple 1, 46, 04, 97, 129, 132) , et le programme produit les sorties suivantes :

« Bonjour monsieur » si c'est un homme et

« Bonjour madame » si c'est une femme

« Tu es bien petit » si son age est inférieur ou égal à 6 ans

« que fais tu la a ton age » si il a entre 6 et 16 ans

« bonjour pépé » si il a plus de 16 ans

« bonjour la Guadeloupe » si dept\_ nais vaut 97 et que x est un nombre commençant par 1

Ecrire ensuite le programme C correspondant

Rappel: entier / entier donne le quotient de la division entière

### Exercice

Faire un algorithme et le programme en C correspondant qui lit 20 notes et calcule deux moyennes, la moyenne des notes supérieures ou égales à 10 et la moyenne des notes inférieures à 10

### Exercice

Ecrire un algorithme et le programme C correspondant qui lit un entier entre 1 et 10. Puis qui demande à un autre utilisateur ne connaissant pas le nombre qui a été rentré précédemment de donner un nombre entre 1 et 10. Dès que l'utilisateur tape le bon nombre on quitte le programme et on félicite le joueur. Sinon on lui indique si ce qu'il a donné est trop bas ou trop haut et on lui demande de redonner un nombre.