Chapitre 2

Algorithme séquentiel simple

Sommaire

Objectif

Dans ce chapitre, on présente les fondamentaux qui vont guider l'étudiant pour concevoir et analyser des algorithmes.

L'objectif de ce chapitre est de présenter la structure d'un algorithme séquentiel simple avec ses composants et ses opérations de base.

Une traduction en langage C est donnée à la fin de ce chapitre pour permettre à l'étudiant d'aborder le domaine de développement en réalisant son premier programme.

I. Introduction

On a noté dans le premier chapitre qu'un ordinateur sert à faire des traitements automatiques sur les informations. Autrement dit, l'activité de programmation permet à un ordinateur de donner des solutions à des problèmes réels. De façon générale, la programmation consiste, à partir d'un problème donné, à réaliser un programme dont l'exécution apporte une solution satisfaisante au problème posé. Le cœur du processus de résolution d'un problème est la conception d'algorithmes.

Un algorithme permet d'expliciter clairement les idées de la solution indépendamment d'un langage de programmation. L'intérêt d'un algorithme est la possibilité d'être codé dans un langage algorithmique et pouvoir être traduit en n'importe quel langage de programmation, en un programme exécutable par un ordinateur.

II. Notion de langage et langage algorithmique

Définition 2.1 : Langage algorithmique

Le langage algorithmique est un pseudo-langage qui tient en compte des caractéristiques de la machine, tout en étant plus souple qu'un langage de programmation. C'est, donc, un compromis entre le langage naturel et un langage de programmation.

Ce langage utilise un ensemble de **mots clés** et de **structures** permettant de décrire de manière complète et claire, les objets manipulés par l'algorithme ainsi que l'ensemble des **instructions** à exécuter sur ces objets pour résoudre un problème.

Définition 2.2 : Mots clés

Un mot clé est un identificateur (mot) qui a une signification particulière au langage, comme : *Début, Fin, Si, Pour, Répéter, Tantque, …etc*

Définition 2.3 : Programme

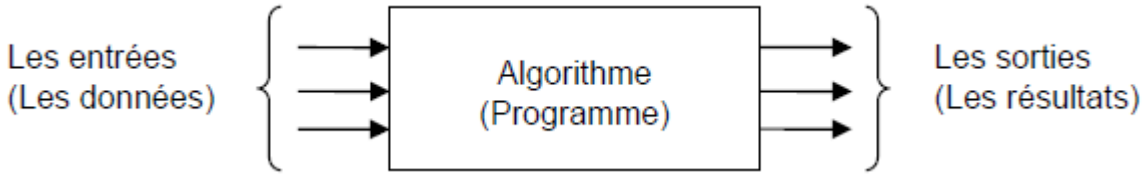
Un programme est la traduction d'un algorithme dans un langage de programmation compréhensible par la machine (en langage C, par exemple).

Un programme informatique, aussi bien qu'un algorithme, réalise en général trois choses :

1- **Introduire les données nécessaires (lire des données en entrée) :** Le programme doit en effet savoir à partir de quoi travailler. Par exemple, pour utiliser une calculatrice, on doit lui donner des nombres et lui dire quelles opérations effectuer. Pour cela, on utilise souvent un *clavier*, mais le programme peut aussi tirer les données d'un *disque dur* ou encore d'un *autre ordinateur via* un réseau ou autre.

2. **Le traitement** : exécuter séquentiellement des instructions sur ces données. À partir des données en entrée, le programme va appliquer automatiquement des méthodes pour traiter ces données et produire un résultat. Par exemple, une calculatrice va appliquer l'opération d'addition ou de multiplication.

3. **Afficher les résultats obtenus (écrire les résultats en sortie)** : Lorsque le programme a obtenu un résultat, il doit écrire ce résultat quelque part pour qu'on puisse l'utiliser. Par exemple, une calculatrice va afficher un résultat à l'*écran* ou stocker le résultat en *mémoire*.



Le travail d'un programmeur consiste à créer des programmes informatiques. Le programmeur doit pour cela expliquer à l'ordinateur dans un certain langage, appelé *langage de programmation*, quelles sont les données et quelles sont les méthodes à appliquer pour traiter ces données.

Définition 2.4 : Langage de programmation

Un langage de programmation est une notation conventionnelle destinée à formuler des algorithmes et produire des programmes informatiques qui les appliquent. D'une manière similaire à une langue naturelle, un langage de programmation est composé d'un alphabet (lettres, chiffres, symboles, caractère spéciaux…), d'un vocabulaire, d'un ensemble de règles (syntaxe ou grammaire) et de significations qui permettent de réunir les éléments du vocabulaire pour former des phrases (ligne de programme) correctes.

Le langage de programmation est l'intermédiaire entre l'humain et la machine, il permet d'écrire dans un langage proche de la machine mais intelligible par l'humain les opérations que l'ordinateur doit effectuer. Ainsi, étant donné que le langage de programmation est destiné à l'ordinateur, il doit donc respecter une syntaxe stricte.

III. Cycle de développement d'un programme

L'utilisation d'un ordinateur pour la résolution d'un problème consiste à réaliser un programme informatique qui reçoit les données en entrée et fournit les résultats en sortie. L'élaboration d'un programme passe généralement par les étapes suivantes :

Problème --> Analyse --> Algorithme --> Programme --> Compilation --> Exécution

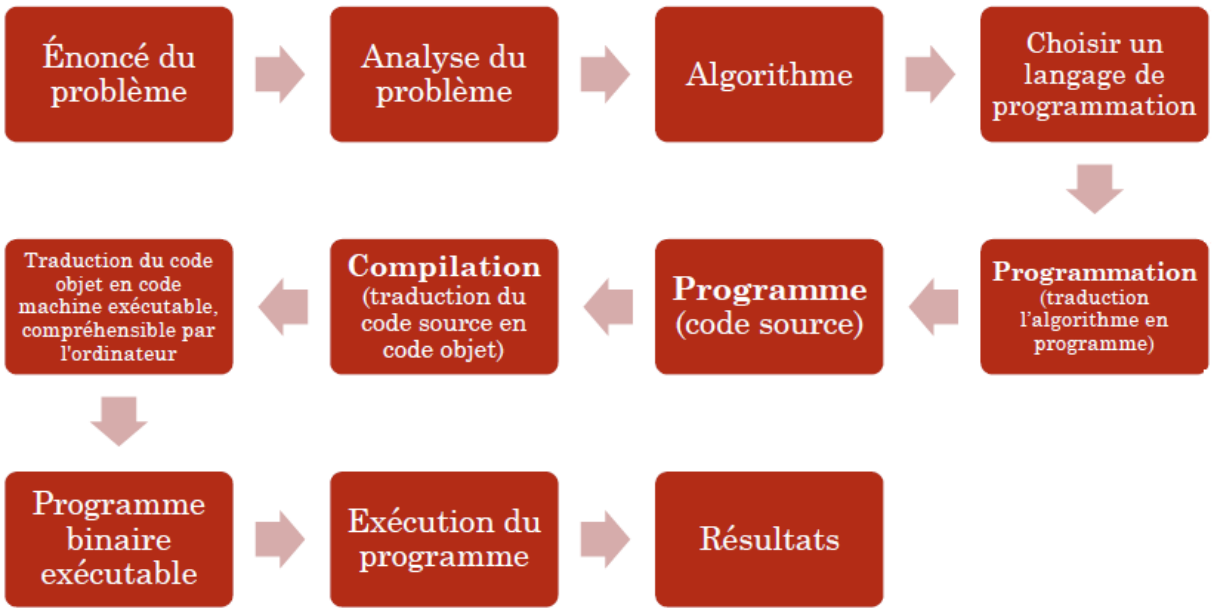


Figure 2.1. Résolution d'un problème informatique

Etape 1 : Définition du problème

Il s'agit de déterminer toutes les informations disponibles et la forme des résultats désirés.

Etape 2 : Analyse du problème

C'est une phase de réflexion qui permet d'identifier les caractéristiques du problème à traiter. Elle consiste à trouver le moyen de passer des données aux résultats.

Selon la complexité du problème P à résoudre, cette étape peut être décomposée en trois parties :

1) Décomposer P en N sous-problèmes (P_1, P_2,..., P_N) de complexité inférieure. Si un ou plusieurs sous-problèmes apparaissent encore trop complexes on les décompose à leurs tours.

2) Identifier les objets nécessaires à la formulation du modèle qui permet de définir le processus de résolution.
   - Objets relatifs aux données : les entrées.
   - Objets relatifs aux résultats : les sorties.
   - Objets intermédiaires qui résultent de la phase de décomposition.

3) Définir les relations qui existent entre ces objets en termes de règles, de formule, d'équations mathématiques et de méthodes de traitements.

Etape 3 : Algorithme

Une fois qu'on trouve le moyen de passer des données aux résultats, il faut être capable de rédiger une solution claire et non ambiguë.

Il s'agit d'une description compréhensible par un être humain de la suite des opérations à effectuer pour résoudre le problème analysé en respectant un formalisme (un ensemble de règles d'écriture) bien déterminé.

Les étapes 1, 2 et 3 se font sans le recours à la machine. Si on veut rendre l'algorithme concret ou pratique, il faudrait le traduire dans un langage de programmation.

Etape 4 : Programme

Cette étape consiste à traduire les instructions de l'algorithme, développé antérieurement, dans un langage de programmation choisi par l'utilisateur. Pour notre cours, nous traduirions en langage C.

Une fois le programme écrit, il va falloir le vérifier et le corriger en lançant la compilation.

Etape 5 : Compilation

Chaque langage de programmation contient un compilateur intégré. Le compilateur est un logiciel qui détecte les erreurs de syntaxe du programme, mais ne détecte pas les erreurs de logique.

- Si le programme est syntaxiquement correct, le compilateur crée ce qu'on appelle programme objet, c'est un programme prêt à être exécuté.

- Si le programme contient des erreurs de syntaxe, le compilateur affiche la liste des erreurs à l'écran.

Etape 6 : Exécution

L'ordinateur exécute les instructions d'un programme en langage "binaire". Ainsi, l'utilisateur "lance" l'exécution de programme compilé pour savoir quel est le résultat.

IV. Structure générale d'un algorithme

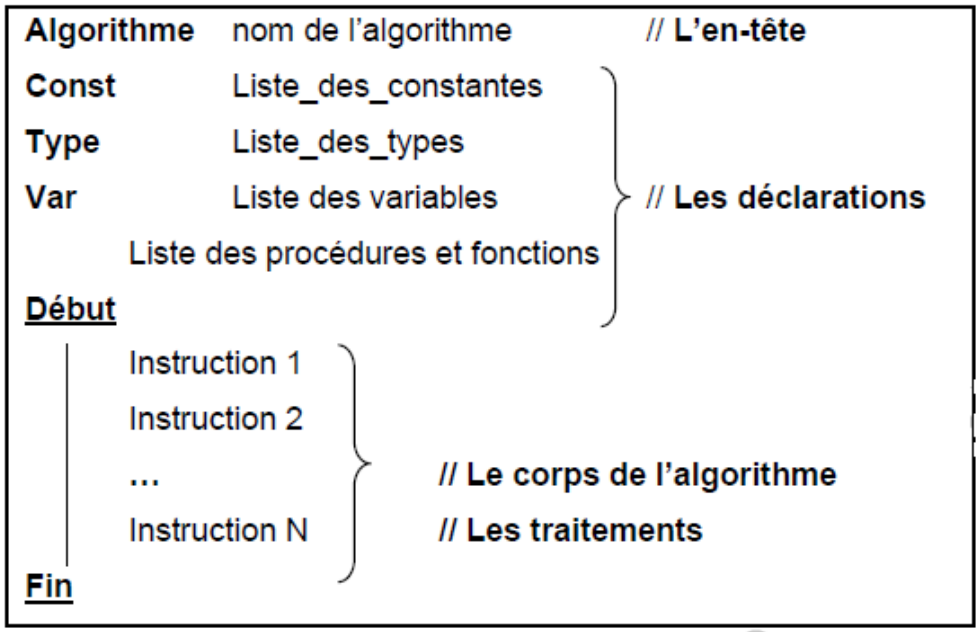
Un algorithme est composé de trois parties indissociables :

1) **Entête :** permet d'identifier l'algorithme. On spécifie le nom de l'algorithme par exemple : Algorithme tri ;

2) **Déclaration :** contient la déclaration de de tous les objets (constantes, variables, …) avec leurs types utilisés et manipulés dans le corps de l'algorithme.

3) **Corps :** contient la séquence d'instructions entre les deux mots clés : Début et Fin

La structure générale d'un algorithme est la suivante :



Exemple : Calculer la surface d'un carré

```
ALGORITHME TEST ;
 VAR
   LONGUEUR, SURFACE : RÉEL ;
DÉBUT
      ECRIRE ("ENTREZ SVP LA LONGUEUR DU CARRÉ") ;
       LIRE (LONGUEUR) ;
       SURFACE ← LONGUEUR*LONGUEUR ;
      ECRIRE ("LA SURFACE DU CARRÉ EST ", SURFACE) ;
FIN.
```

La succession d'instructions n'est pas toujours détaillée explicitement au sein de l'algorithme, mais parfois dans une autre partie, à travers ce que l'on appelle des fonctions ou des procédures. Cela permet de découper l'algorithme en plusieurs sous-algorithmes et de le rendre plus facile à comprendre.

V. Les données : variables et constantes

Lors de la préparation d'un algorithme, on a toujours besoin de stocker provisoirement des valeurs. Il peut s'agir des données fournies par l'utilisateur, comme il peut s'agir des résultats obtenus par le programme (intermédiaires ou définitifs). Pour cela, on utilise des objets.

Un objet est de nature ***variable***, si sa valeur peut changer pendant l'exécution des actions de l'algorithme. Comme il est de nature ***constante*** si sa valeur est invariable.

Tous les objets qui deviennent partie intégrante de l'exécution d'un algorithme doivent être déclarés avant leurs utilisation; ceci constitue la partie déclaration d'un algorithme. Un objet peut être décrit par un **nom (identificateur**), un **type** et une **valeur**.

- **Nom (identificateur**) : est utilisé pour donner un nom à un objet, afin de l'identifier et de le distinguer des autres objets dans l'algorithme

  Remarques :
  - Un identificateur doit être significatif (représentatif) pour faciliter la compréhension de l'algorithme.
  - Un identificateur est une suite de lettres non accentuées et/ou de chiffres, commençant par une lettre.
  - En l'algorithmique, il n'y a pas de différence entre minuscules et majuscules.
  - Le caractère « _ » peut être utilisé pour construire des noms d'identificateurs composés de plusieurs mots.
  - Un identificateur doit être différent des mots clés (algorithme, début, fin, ….)

- **Type :** caractérise l'intervalle ou l'ensemble des valeurs que peut prendre cet objet ainsi que les opérations qui lui sont autorisées tout au long de l'algorithme.

  Une fois qu'un type de données est associé à une variable le contenu de cette variable doit **obligatoirement** être du même type.

- **Valeur :** La valeur d'un objet est celle qui lui a été assignée pour représenter la grandeur que contient cet objet.

  Dans la partie déclaration, les variables n'ont pas de valeur. Attribuer une valeur à une variable se fait par le biais d'une instruction. Initialiser une variable, c'est lui donner une première valeur.

Définition 2.5 : Variables

Une variable est une case mémoire qui sert à stocker l'information traitée par un algorithme. Cette variable associe un nom à une valeur qui peut varier au cours du temps de l'exécution d'un algorithme.

En effet, la variable est représentée dans la mémoire par un nombre de mots mémoire correspond à leur type (Figure 2.2).

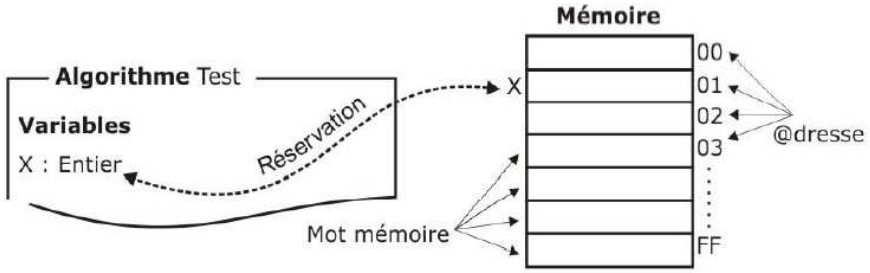


Figure 2.2. Représentation d'une variable dans la mémoire

Un objet de nature variable est déclarée à l'aide du mot **VAR** (ou Variables). L'expression de déclaration de variables s'écrit suivant la règle suivante :

```
VAR
        Nom_variable : Type_variable ;
```

Exemple :

```
VAR
    X : Entier ;          // déclaration d'une variable X de type Entier
    Y, Z : réel ;         // déclaration d'une variable Y et Z de type réel
```

Définition 2.6 : Constantes

Une constante est objet informatique similaire à une variable en nom et type. Elle se diffère d'une variable seulement dans sa valeur qui ne varie pas au cours de l'exécution de l'algorithme, fixée en début de l'algorithme.

Une constante est déclarée à l'aide du mot **CONST** (ou Constante). Un objet de nature constante suit la déclaration suivante :

```
CONST
        Nom_constante = Valeur_constante ;
```

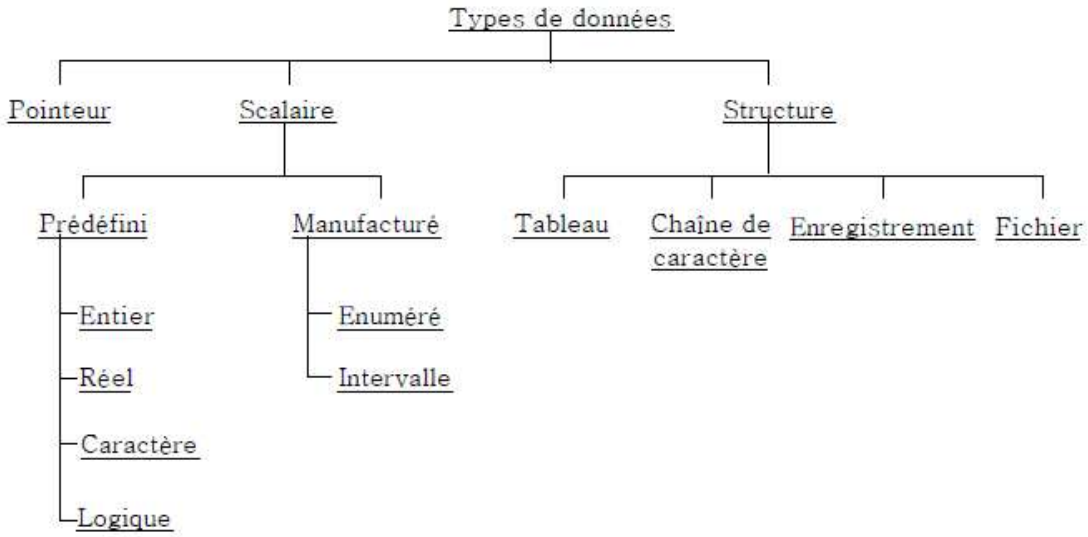
Exemple :

```
CONST
    Pi = 3.14              // déclaration d'une constante Y et sa valeur = 3.14
    ville = 'BISKRA'       // déclaration d'une constante ville de type chaine de
                              caractère et sa valeur = 3.14.
```

VI. Types de données

Un type définit l'ensemble des **valeurs** que peut prendre un objet. Il définit également les **opérations**, généralement appelées opérateurs, qui pourront être appliquées sur les données de ce type.

Les types de données sont résumés dans le diagramme suivant :



Dans cette partie du cours, on s'intéresse au type prédéfini. Ces types sont des types de base fréquemment utilisés, ils n'ont pas besoin d'être définis par l'utilisateur. Ils sont :

- **Type entier :** Ce type représente le domaine des nombres entiers (Z).
- **Type réel :** Ce type représente le domaine des nombres réels (R).
- **Type logique (booléenne):** Ce type représente le domaine logique qui contient deux valeurs logiques : {FAUX, VRAI},
- **Type caractère :** Ce type représente le domaine des caractères qui contient les lettres alphabétiques, les caractères numériques, les caractères spéciaux (., ?, !, <, >, =, *, +, ...etc), et le caractère espace (ou blanc).
- **Type chaîne de caractères** permet de représenter des mots ou des phrases.

Remarques :

- Une variable de type réel peut avoir une valeur de type entier, car le type entier est inclus dans le type réel, mais l'inverse est totalement faux.
- Une variable de type chaîne de caractères peut avoir une valeur de type caractère, car le type caractère est inclus dans le type chaîne de caractères, mais l'inverse est totalement faux.

VII. Opérations de base

Pour pouvoir comprendre et utiliser correctement les opérations de base en algorithmique, on doit tout d'abord aborder les notions d'opérateur, d'opérande et d'expression.

Définition 2.7 : Opérateur

Un opérateur est un symbole d'opération qui permet d'agir sur des variables ou de faire des "calculs". Il existe plusieurs types d'opérateurs :

- **Les opérateurs arithmétiques** qui permettent d'effectuer des opérations arithmétiques entre des opérandes numériques :
  Les opérateurs élémentaires sont : «+», «-», «x», «/», «[/]» (division entière) et l'opérateur de changement de signe «-» (qui est similaire à l'opérateur de soustraction).

- **Les opérateurs de comparaison** («<», «>», «≤», «≥», «=» et «≠») qui permettent de comparer deux opérandes et produisent une valeur booléenne, en s'appuyant sur des relations d'ordre :
  - Ordre naturel pour les entiers et les réels
  - Ordre lexicographique ASCII pour les caractères

- **Les opérateurs logiques** qui combinent des opérandes booléens pour former des expressions logiques plus complexes :
  - Unaire : «non» (négation)
  - Opérateurs binaires : «et» (conjonction), «ou» (disjonction), «ou_exclusif»

Définition 2.8 : Opérande

Un opérande est une entité (variable, constante ou expression) utilisée par un opérateur

Définition 2.9 : Expression

Une expression est une combinaison d'opérateur(s) (arithmétiques, logiques) et d'opérandes (constantes, variables), elle est évaluée durant l'exécution de l'algorithme, et possède une valeur (son interprétation) et un type

Exemple :

Dans a + b

a est l'opérande gauche.

+ est l'opérateur.

b est l'opérande droit.

a + b est appelé une expression.

par exemple a vaut 2 et b 3, l'expression a + b vaut 5.

Si par exemple a et b sont des entiers, l'expression a + b est un entier.

L'évaluation d'une expression faisant intervenir plus d'un opérateur repose sur des règles de priorité entre opérateurs. Pour lever toute ambiguïté lors de l'évaluation d'une expression, il est nécessaire d'associer une priorité à chaque opérateur pour définir son ordre d'exécution. Comme il est possible de modifier cet ordre par l'utilisation des parenthèses.

- **Ordre de priorité décroissante des opérateurs arithmétiques**

Symbole	Opération
«-»	changement de signe
«x», «/», «[/]»	Multiplication, division et division entière
«+», «-»	Addition et Soustraction

- **Ordre de priorité décroissante des opérateurs logiques**

Symbole	Opération
«non»	négation logique (fonction inverse)
«et»	pour réaliser la conjonction logique entre deux valeurs booléennes
«ou»	pour réaliser la disjonction logique entre deux valeurs booléennes
«Xou»	pour réaliser le ou exclusif entre deux valeurs booléennes

Les opérations élémentaires

- **Sur les entiers** : Les opérations utilisables sur les entiers sont :
  - Les opérateurs arithmétiques classiques.
  - La division entière, notée DIV, telle que n DIV p donne la partie entière du quotient de la division entière de n par p.
  - Le MODulo, noté MOD, telle que n MOD p donne le reste de la division entière de n par p.
  - Les opérateurs de comparaison classiques.

- **Sur les réels :** Les opérations utilisables sur les réels sont :
  - Les opérations arithmétiques classiques.
  - Les opérateurs de comparaison classiques.

- **Sur les booléens :** Les opérations utilisables sur les booléens sont réalisées à l'aide des opérateurs logiques

- **Sur le caractère :** Les opérations élémentaires réalisables sont les opérateurs de comparaisons selon l'ordre lexicographique

- **Sur une chaîne de caractères :** Les opérations utilisables sur chaîne de caractères sont :
  - Les opérateurs de comparaison classiques selon l'ordre lexicographique.
  - La Concaténation représenter par un + .

VIII. Instructions de base

Les instructions de base permettent le transfert d'informations entre objets, ou une communication entre un algorithme et l'un des périphériques d'entrées ou de sorties. On distingue trois genres d'instructions élémentaires : l'affectation, la lecture et l'écriture.

A. Instruction d'affectation

C'est l'action par laquelle nous pouvons charger une valeur dans une variable X. Cette valeur peut elle-même être une variable, une constante ou le résultat de l'évaluation d'une expression arithmétique/logique.

L'affectation s'effectue en utilisant l'opérateur d'affectation représenté par le symbole ←

L'affectation est faite selon la syntaxe suivante :

- Affectation d'une valeur à une variable : **Variable ← valeur ;**
- Affectation d'une variable à une variable : **Variable1 ← Variable2**
- Affectation du résultat de calcul à une variable : **Variable← Variable1 opérateur Variable2 ;**
  Ou encore l'affectation du résultat de plusieurs calculs à une variable :
  Variable←Variable1 opérateur1 Variable2 … opérateur-N VariableN ;

Le type de la partie droite de l'instruction doit être compatible avec le type de la variable X.

Exemple :

```
Age ← 24 ;          // Initialiser (attribuer) la valeur 24 à la variable Age.
A   ← ((A+B)*2) ;
```

Remarque :

L'instruction d'affectation permet de :

- Forcer le contenu d'une variable (initialisation de la variable).
- D'écraser la valeur contenue précédemment dans la variable.

B. Instruction de lecture

Cette instruction permet d'introduire des valeurs par l'utilisateur via un périphérique d'entrée dans des cases mémoire bien définies. La lecture est faite selon la syntaxe suivante :

Lire (variable) ; ou **Lire (variable1, variable2,… variableN) ;**

L'exécution de cette instruction consiste à affecter une valeur à la variable en prenant cette valeur sur le périphérique d'entrée. Avant l'exécution de cette instruction, la variable avait ou n'avait pas de valeur. Après, elle a la valeur prise sur le périphérique d'entrée. La valeur à introduire doit être de même type que la variable réceptrice.

Exemple : Supposons x une variable de type entier alors :

Lire(x) ; // ***Affectera une valeur de type entier taper au clavier à la variable X.***

La fin de la valeur est reconnue en tapant la touche ENTREE

Remarque :

L'instruction de lecture bloquera l'exécution de programme jusqu'à ce que la touche ENTREE soit tapée.

C. Instruction d'écriture

L'écriture permet d'afficher un résultat ou un message à l'utilisateur sur un périphérique de sortie. L'écriture est faite selon la syntaxe suivante :

Ecrire (variable);

Ecrire (var1, var2,… varN) ***;*** ***// pour afficher les valeurs des variables var1, var2,… varN***

Ecrire ('Bonjour'**)** ; ***// pour afficher un message à l'utilisateur sur l'écran.***
// dans ce cas, le message doit être écrit entre simples quottes.

Remarque :

Il est possible de regrouper aussi un message et une variable dans la même instruction d'écriture comme suit :

Ecrire ('Résultat = ' **,** x**1)**

Exemple : supposons x=0, y=0 alors:

Ecrire (x, y+2, "bonjour") ; //***Affiche sur l'écran : 0 2 bonjour***

IX. Construction d'un algorithme simple

Un algorithme se présente en général sous la forme suivante :

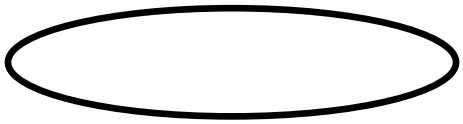
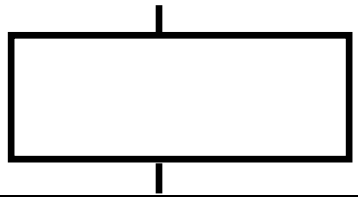
- ***Déclaration des variables :*** On décrit dans le détail les éléments que l'on va utiliser dans l'algorithme (variables, constantes et structures),
- ***Initialisation ou Entrée des données :*** On récupère les données et/ou on les initialise (par lecture ou par affectation),
- ***Traitement des données :*** On effectue les opérations nécessaires pour répondre au problème posé,
- ***Sortie des résultats:*** On affiche les résultats (par l'écriture).

Exemple : Écrire un algorithme qui permet de calculer la somme de deux entiers.

```
ALGORITHME CALCUL_SOMME ;
VAR
        A, B, C: ENTIER ;
DÉBUT
    ECRIRE ("ENTREZ VOTRE PREMIERE VALEUR ") ;
    LIRE (A) ;
    ECRIRE ("ENTREZ VOTRE DEUXIEME VALEUR ") ;
    LIRE (B) ;
    C ← A+B;
    ECRIRE ("LA SOMME = ", C ) ;
FIN.
```

X. Représentation d'un algorithme par un organigramme

En général, on peut représenter un algorithme sous forme structurée ou sous forme graphique. Un **organigramme** est une représentation graphique d'un algorithme. Pour cela, on utilise des symboles normalisés dont les principaux sont les suivants :

Début et fin 	On l'utilise pour commencer et pour terminer un algorithme.
Entrée / Sortie 	On l'utilise pour lire ou écrire des données.
Symbole de traitement 	On l'utilise pour le reste des opérations hors la lecture et l'écriture.
Symbole de liaison	On l'utilise pour connecter les autres symboles

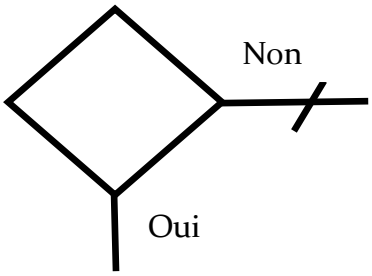
Symbole de test (branchement)	
	Choix avec condition : on l'utilise pour l'exploitation de conditions variables impliquant le choix d'une voie parmi plusieurs.

Tableau 2.1. Symboles d'un organigramme

Exemple : L'algorithme de l'exemple précédent (somme de deux entiers) peut être représenté par l'organigramme suivant :

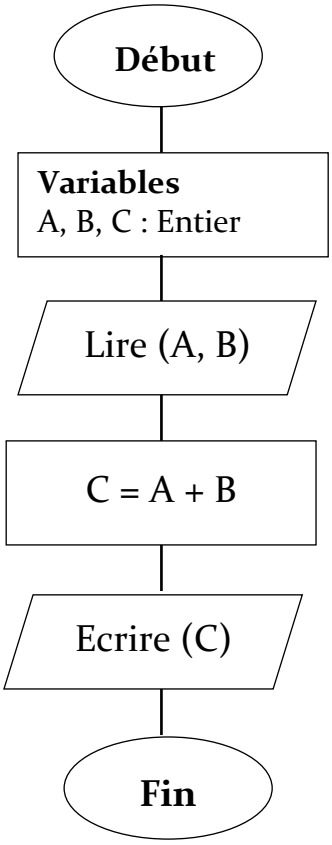


Figure 2.3 : Structure d'un organigramme

XI. Traduction en langage C

Dans notre ouvrage nous présentons la traduction des algorithmes en langage C.

Le langage C est un langage évolué et structuré. C'est un langage souple et puissant, assez proche du langage machine utilisé pour des projets tels que les systèmes d'exploitation, des applications de contrôle de processus (gestion d'entrées/sorties, applications temps réel …), des applications de traitements de textes, des graphiques ou même des compilateurs pour d'autres langages. Inventé au début des années 1970, C est devenu un des langages les plus utilisés.

Le langage C possède assez peu d'instructions, il fait par contre appel à des bibliothèques, fournies en plus ou moins grand nombre avec le compilateur.

Exemples:

- **math.h** : bibliothèque de fonctions mathématiques de base.
- **stdio.h** : bibliothèque d'entrées/sorties standard utilisée avec les unités classiques d'entrées-sorties, qui sont respectivement le clavier et l'écran.

La programmation par le langage C est basée sur :

1. La rédaction du texte du programme (codage) en respectant la syntaxe précise du langage, à l'aide d'un éditeur de texte.
2. La compilation : transformation du code C en langage machine par un compilateur C;
3. Exécution du fichier binaire obtenu.

Premier programme :

```
#include <stdio.h>
main()
{
int a,b,c;      /* Déclaration de variables */
printf (" Bonjour \n");
a = 1;          /* initialisation */
b = 2;
c = a + b;     /* affectation */
printf (" Résultat = %d \n ",c);
printf (" Au revoir !\n");
}
```

La fonction ***main()*** est la fonction principale du programme. Elle est la seule à être exécutée au lancement du programme ; elle fait appel éventuellement à d'autres fonctions. Dans l'exemple ci-dessus, elle appelle la fonction ***printf ().***

La fonction ***printf ()*** est une fonction standard définie dans le fichier ***stdio.h*** inclus en début de programme. Elle affiche une chaıne de caractères à l'´écran ainsi que certains types de variables prédéfinis.

Le corps des fonctions est toujours délimité par des **{…}**, et chaque instruction doit se terminer par un **" ; ".**

Les commentaires sont mis entre **/* … */,** ainsi le compilateur les ignore.

XI.1 Variables et Constantes en langage C

Le C est un langage typé. Cela signifie en particulier que toute variable, constante ou Fonction est d'un type précis.

En langage C, les variables doivent être déclarées en début de programme et leurs types doivent être explicités. Le tableau ci-dessous englobe les types les plus utilisés en langage C avec leurs spécificités :

Nom du Type	Description	Nombre d'octets	Domaine Min	Domaine Max
char	Caractère	*1*	-128	127
int	Entier standard signé	2	-32 768	32 767
short	entier court	2	-32 768	-32 768
long	entier long	4	-2147483648	2147483647
unsigned int	entier positif	2	0	65535
float	Réel à virgule flottante simple précision	4	$3.4 * 10^{-38}$	$3.4 * 10^{38}$
double	Réel à virgule flottante double précision	8	$1.7 * 10^{-308}$	$1.7 * 10^{308}$

Tableau 2.2. Principaux types de variables en langage C

En langage C, le type char est un cas particulier du type entier : ***Un caractère est un entier de 8 bits***

La déclaration en langage C se fait à l'aide du mot-clé correspondant au type souhaité, suivi d'un nom de variable

Exemple :

- int Var1 ; // ***Var1 est une variable de type entier.***
- float Y ; // ***Y est une variable de type réel.***
- char W ; // ***W est une variable de type caractère.***

En langage C, l'opérateur d'affectation est "=". Le langage C permet l'initialisation des variables dans la zone des déclarations:

char c; est équivalent à ***char*** c = 'A';
c = 'A';

int i; est équivalent à ***int*** i = 50;
i = 50;

Le langage C distingue les minuscules des majuscules (Exemple : a est différente de A). Les mots réservés du langage C doivent être écrits en minuscules.

La déclaration et l'initialisation d'une variable peuvent commencer par le qualificatif ***const*** qui indique que la valeur de cette variable ne pourra plus être modifiée.

```
const int N = 30;
```

Il est également possible de déclarer des constantes à l'aide de la commande du préprocesseur ***#define*** :

```
#define NMAX 10
```

Avant la compilation, le préprocesseur (une partie du compilateur) remplacera toutes les occurrences de NMAX par le chiffre 10.

XI.2 Les opérateurs en langage C

- **Les opérateurs arithmétiques**

  x + y addition
  x - y soustraction
  x * y multiplication
  x / y division
  x % y reste de la division entière (modulo)

- **Les opérateurs de comparaison**

  Les opérateurs de comparaison sont : **> , >= , < , <=**
  Les opérateurs d'égalité : == , **!=** .
  Attention, == est différent de l'opérateur d'affectation **=.**

- **Les opérateurs logiques**

  Les opérateurs logiques **&&** et || signifient "et" et "ou" .
  L'opérateur de négation : **! .**

Ces opérateurs donnent un résultat "***Vrai***" ou "***Faux***" selon l'expression où ils figurent. En réalité ils retournent la valeur **0** pour "Faux" et **1** pour "Vrai".

En général, toute valeur non nulle est ′évaluée comme "Vrai".

```
int a,b;
a = 2;
b = 3;
a > b;                    /* vaut 0 */
a < b;                    /* vaut 1 */
!b;                       /* vaut 0 */
a == b;                   /* vaut 0 */
b != a;                   /* vaut 1 */
a > b && a < b;           /* vaut 0 */
a > b || a < b;           /* vaut 1 */
```

- **Les opérateurs d'incrémentation et de décrémentation**

```
int i = 0;
int a;
i++;          /* equivalent à  i = i + 1 */
i--;          /*equivalent à   i = i - 1 */
```

- **Les opérateurs d'affectation**

```
a = i;          /* a prend la valeur de i */
a += i;         /* equivalent à   a = a + i */
a -= i;         /* equivalent à   a = a - i */
a *= i;         /* equivalent à   a = a * i */
a /= i;         /* equivalent à   a = a / i */
```

XI.3 Instruction de lecture en langage C

La lecture en langage C s'effectue avec l'utilisation de la fonction ***scanf***. Il s'agit d'une fonction de la librairie standard ***stdio.h*** .

La fonction ***scanf*** permet de saisir des données au clavier et de les stocker aux adresses spécifiées par les arguments de la fonction. Les variables à saisir sont formatées, le nom de la variable est précédé du symbole & désignant l'adresse de la variable. La syntaxe de scanf est la suivante :

Scanf ("%format ", & nom_de_variable)

Le tableau ci-dessous indique les Formats de saisie les plus utilisés .

Format	Type attendu
%c	Caractère
%d	int
%hd	short
% ld	long
%u	unsigned int
%f	float
% lf	double

Tableau 2.3. Principaux formats de saisie

Exemples:
```
char alpha;
int i;   float r;
scanf("%c",&alpha);          // saisie d'un caractère .
scanf("%d",&i);              // saisie d'un nombre entier en décimal .
scanf("%f",&r);              // saisie d'un nombre réel .
```

XI.4 Instruction d'écriture en langage C

L'écriture en langage C s'effectue avec l'utilisation de la fonction ***printf***. Il s'agit d'une fonction d'affichage formatée de la librairie standard ***stdio.h***, ce qui signifie que les données sont converties selon un format particulier choisi.

On utilise ***printf*** de la même manière que pour afficher un texte, sauf que l'on rajoute un symbole spécial à l'endroit où l'on veut afficher la valeur de la variable. La fonction ***printf*** exige l'utilisation de formats de sortie, avec la structure suivante:

printf ("%format",nom_de_variable);

Pour un affichage multiple, on utilise la structure suivante:

printf ("format1 format2 .... formatN",variable1,variable2, ...,variableN);

Exemples: affichage d'un texte:

printf("BONJOUR"); // **pas de retour à la ligne du curseur après l'affichage.**

printf("BONJOUR\n"); // **affichage du texte, puis retour à la ligne du curseur.**

On peut éventuellement préciser certains paramètres du format d'impression, qui sont spécifies entre le % et le caractère de conversion dans l'ordre suivant :

- ***Largeur minimale du champ d'impression*** : %10d spécifie qu'au moins 10 caractères seront réservés pour imprimer l'entier.

- ***Précision :*** %.2f signifie qu'un flottant sera imprimé avec 2 chiffres après la virgule. Lorsque la précision n'est pas spécifiée, elle correspond par défaut à 6 chiffres après la virgule. Pour une chaîne de caractères, la précision correspond au nombre de caractères imprimés : %30.4s signifie que l'on réserve un champ de 30 caractères pour imprimer la chaîne mais que seulement les 4 premiers caractères seront imprimés (suivis de 26 blancs).

Trace d'exécution :

L'exécution d'un algorithme (programme) se fait instruction par instruction selon l'ordre indiqué par l'algorithme. Elle commence à partir de la première instruction qui suit le mot clé DEBUT et se termine à la dernière instruction qui précède juste le mot clé FIN.

Au début les valeurs des variables sont inconnues « ? ». Quand une variable reçoit une valeur, cette valeur est sauvegardée jusqu'à ce qu'une autre instruction change cette valeur.

Exemple : donnez la Trace d'exécution de l'algorithme suivant

```
Algorithme trace ;
   x : entier ;
   y : entier ;
Début
      x ← 12 ;
      y ← x + 4 ;
      x ← 3 ;
Fin.
```

Instructions	x	y
(1)	12	?
(2)	12	16
(3)	3	16

Tableau 2.4. Trace d'exécution d'un algorithme

XII. Conclusion

Nous avons présenté à travers ce chapitre les premiers pas pour construire un algorithme séquentiel simple. Par conséquent, les notions de variables, d'affectation, de lecture et d'écriture sont présentées algorithmiquement et techniquement avec le langage de programmation C.