un

Initiation au Langage

C++

Elaboré par : Mr Mustapha AITMAHJOUB

SOMMAIRE

Page

INTRODUCTION AU LANGAGE C++

1. ELEMENTS DU LANGAGE ……... 2
   1.1 Alphabet du langage……...……2
   1.2 Mots réservés……...……..2
   1.3 Identificateurs……..…2
   1.4 Constantes……...3
   1.5 Commentaire……...3
   1.6 Les séparateurs……3
2. STRUCTURE D'UN PROGRAMME C++……4
   2.1 Structure de base……5
3. TYPES DE DONNEES FONDAMENTAUX……6
   3.1 Données de type entier.....6
   3.2 Données de type réel……..6
4. VARIABLES ET CONSTANTES……...7
   4.1 Variables ……7
   4.2 Constantes……..7
5. ENTREE/SORTIE DES DONNEES……...8
   5.1 Fonction printf ()……8
      5.1.1 Sortie formatée……..…8
      5.1.2 Largeur du champ……9
      5.1.3 Séquences d'échappement……..10
   5.2 Fonction scanf ()……..10
6. LES EXPRESSIONS……11
   6.1 Opérateurs arithmétiques……12
   6.2 Opérateurs de manipulation……12
   6.3 Opérateurs d'incrémentation/decrémentation……12
   6.4 Opérateurs sizeof ()……12
7. EXPRESSION BOOLEENNE……13
   7.1 Opérateurs de comparaison……13
   7.2 Opérateurs logiques……13
   7.3 Opérateurs conditionnels ternaire……14
8. INSTRUCTIUONS D'AFFECTATION……..14
9. PRIORITE DES OPERATEURS……..17

INSTRUCTION DE CONTROLE

1. STRUCTURE SEQUENTIELLE……17
2. STRUCTURE SELECTIVE……18
   2.1 Structure if……19
   2.2 Structure if,,else……21
   2.3 Structures if imbriquées……24
   2.4 Structure switch……25
3. STRUCTURE REPETITIVE……26
   3.1 Structure while……28

3.2 Structure do..while……...29
3.3 Structure for……30

4. INSTRUCTUION DE BRANHEMENT……..……30
4.1 Instruction break……...31
4.2 Instruction continue……31
4.3 Instruction goto……34

TYPES DE DONNEES COMPLEXES

1. LES TABLEAUX……34
1.1 Tableaux unidimensionnels ……34
1.1.1 Déclaration d'un tableau……34
1.1.2 Initialisation d'un tableau……36
1.2 Tableaux multidimensionnels……..36
1.2.1 Déclaration d'un tableau……...36
1.2.2 Initialisation d'un tableau……..38
2. LES CHAINES DE CARACTERES……38
2.1 Fonctions d'entrée/sortie des chaînes……..39
2.2 Fonctions de traitement des chaînes……40
3. LES STRUCTURES……41
3.1 Déclarations d'une structure……41
3.2 Déclaration de variables d'une structur……41
3.3 Accès aux membres de la structure……..42
4. LES POINTUERS……43
4.1 Déclaration des pointeurs……45
4.2 Tableaux et pointeurs……47
4.3 Pointeurs et chaînes de caractères ……47
4.3.1 Initialisation d'une chaîne……47
4.3.2 Traitement des chaînes……49
4.4 Pointeurs et structures……..50
4.5 Allocation dynamique de la mémoire……..52
5. ENUMERATION……..52
5.1 Utilisation des énumérations ……53
6. DEFINITION DE NOUVEAUX TYPES……..54

LES FONCTIONS

1. FONCTION NE RETOURNANT PAS DE VALEUR……57
1.1 Fonction ne prenant pas de paramètres……57
1.2 Fonction prenant des paramètres……...59
2. FONCTION RETOURNANT UNE VALEUR……61
3. PASSAGE DE PARAMETRES……63
3.1 Passage par valeur……63
3.2 Passage par adresse……..64
3.3 Passage par référence……64

4. PASSAGE D'UN TABLEAU à UNE FONCTION ……66
5. PASSAGE D'UNE STRUCTURE à NE FONCTION……..69
6. CLASSE DE STOKAGE……72
6.1 Durée de vie……..72
6.1.1 Variables automatiques……73
6.1.2 Variables statiques et externes……73

6.2 Portée des variables……..74
6.2.1 Portée bloc……74
6.2.2 Portée fonction……74
6.2.3 Portée fichier……75

LES FICHIERS

1. DEFINITION……77
2. PRINCIPE D'ACCES AUX FICHIERS……..……77
3. ENTREE/SORTIE STANDARD……..78
3.1 E/S des caractères……79
3.1.1 Ouverture d'un fichier……80
3.1.2 Ecriture vers un fichier……83
3.1.3 Fermer un fichier……83
3.1.4 Lecture d'un fichier……83
3.1.5 Fin de fichier……84
3.1.6 Erreur d'ouverture d'un fichier……84
3.1.7 Application : compter le nombre de caractères……86
3.2 E/S des chaînes……..87
3.3 E/S formatées……90
3.4 Mode binaire et mode texte……..93
3.5 E/S des enregistrements……95
3.5.1 Ecriture des structures avec fwrite()……95
3.5.2 Lecture des structures avec fread()……..97
3.6 Accès aléatoire……98
4. E/S SYSTEME……..101
4.1 Lecture des fichiers……..102
4.1.1 Création d'un tampon……..103
4.1.2 Ouverture d'un fichier……103
4.1.3 Identificateur de fichier……...104
4.1.4 Lecture du fichier dans le tampon……...104
4.1.5 Fermeture du fichier……..……..105
4.1.6 Message d'erreur……105
4.2 Opérations sur le tampon……105
4.3 Ecrire dans un fichier……..109

Introduction

OFPPT : ISTA HAY RIAD

INTRODUCTION AU LANGAGE C

1. ELEMENTS DU LANGAGE C

1.1 Alphabet du langage

Le langage C utilise les lettres A à Z (majuscule et minuscule), les chiffres 0..9 et certains symboles spéciaux comme éléments de construction pour former les éléments de base d'un programme.

Certains symboles spéciaux se composent de deux caractères consécutifs, par exemple ++ >= /* !=

1.2 Mots réservés

Les mots réservés ou mots clés ont en langage C et pour le compilateur une signification standard prédéfini. Les mots clés ne peuvent pas être redéfinis par l'utilisateur.

Asm	_asm	__asm	auto	break	case
cdecl	_cdecl	__cdecl	char	class	const
continue	_cs	__cs	default	delet	do
double	_ds	__ds	else	enum	_es
__es	_export	__export	extern	far	_far
__far	_fastcall	__fastcall	float	for	friend
goto	huge	__huge	if	inline	int
interrupt	_interupt	__interupt	_loadds	__loadds	long
near	_near	__near	new	operator	pascal
_pascal	__pascal	private	protected	public	register
return	_saveregs	__savaregs	_seg	__seg	short
signed	sizeof	_ss	__ss	static	struct
switch	template	this	typedef	union	unsigned
virtual	void	volatile	while		

TABLEAU 1 : Mots réservés

1.3 Identificateurs

Un identificateur se définit comme un nom donné à un élément du programme tels que les variables, les types, fonctions et les labels.

L'identificateur est crée en le spécifiant dans la déclaration d'une variable, type ou d'une fonction. Dans l'exemple suivant, Valeur1, Valeur 2 et Somme sont des identificateurs de variables entières, et main() et printf() sont des identificateurs de fonctions.

```
#include<stdio.h>
void main()
{
        int Valeur 1 , Valeur 2, Somme ;
        Valeur 1 = 10
        Valeur 2 = 20
        Somme = Valeur 1 + Valeur  2 ;
        Printf (“ Somme = %d”, Somme) ;
}
```

Une fois déclaré, on peut utiliser l’identificateur, plus tard, dans une instruction de programme pour se référer à la valeur associé à cet identificateur.

Les identificateurs se composent de lettres, de chiffres ou du caractère souligné dans n’importe quel ordre, pourvu que le premier caractère soit une lettre ou le caractère souligné. Le compilateur C fait une distinction entre les majuscules et les minuscules, par exemple valeur et Valeur sont deux identificateurs différents.

J, nombre_jour, ENTIER sont des identificateurs valides
Valeur 1 : invalide contient la caractère espace
7 somme : invalide car le premier caractère est un chiffre

1.4 Constantes

Il est souvent commode d’associer un élément de donnée simple, tel qu’une valeur numérique, un caractère ou une chaîne de caractère, à un identificateur, attribuant ainsi un nom à l’élément de donnée. On appelle l’identifcateur une constante si on lui attribue l’élément de donnée de manière définitive (la valeur reste inchangée tout au long du programme)

1.5 Le commentaire

Les commentaires sont des textes servant à annoter un programme. Ils ne présentent d’utilité que pour le programmeur, et ils sont supprimés logiquement du fichier source avant la compilation.
Il y’ a deux façons de délimiter les commentaires, soit par la paire ** lorsque le commentaire est écrit sur plusieurs lignes, soit // pour débuter une ligne de commentaire jusqu’à la fin de la ligne.

1.6 Les séparateurs

Crochets [] : les crochets ouvrants et fermants *[ ]* servant à spécifier un indice d’accès à un tableau mono ou multi-dimensios.

```
Char Ch, Str[]= “ Bonjour ” ;
Int Mat[3] [4] ; //matrice 3x4
Ch=Str[3] ;     //ch=’n’
```

Parenthèses () : Les parenthèses permettent de grouper des expressions, d'isoler des expressions conditionnelles et d'indiquer des appels de fonctions et des paramètres :

```
X = (a+b)/c ; // remplace la priorité normal
if (a= =0) ; // utilisation dans une instruction conditionnelle
Fonc() ; // appel d'une fonction
```

Accolades {} : Le couple d'accolades permet d'indiquer le début et la fin d'un bloc d'instruction.

```
if (x !=5)
        {
        x++ ;
        fonct(x) ;
        }
```

Virgule : La virgule sépare les éléments d'une liste d'arguments d'une fonction :

```
Int somme (intx, int y) ;
```

Point-virgule ; Le point-virgule est le terminateur d'instruction. Toute expression légale en C (y compris l'expression vide) qui est suivie par le signe ; est interprétée en tant qu'instruction.

Deux-points : Le signe deux-points permet une instruction avec label :

```
Debut : x=0 ;
.
.
goto Debut ;
```

2. STRUCTURE D'UN PROGRAMME C++

Les deux principales parties des programmes C++ sont les fichiers sources et les fichiers en-tête. Les fichiers sources contiennent les paries principales du programme. C'est dans les fichiers sources qu'on définit les routines, les données et les instructions.

Parfois, les fichiers sources empruntent les routines et fonctions d'autres fichiers sources ou de bibliothèque. Pour que ces routines soient reconnues par le compilateur, on doit déclarer dans le fichier principal des fichiers d'en-tête. Le fichier d'en-tête indique au compilateur les noms et caractéristiques des routines. Par exemple pour calculer 5^{10} , on doit faire appel à la fonction pow qui est déclarée dans le fichier à inclure math.h

2.1 Structure de base des programmes C++

L'exemple suivant est un programme C++

```
/* ----------------------------------------------------------------------------------------------------------
*PROGRAMME 1
*Fichier : affichev.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date : 10/09/1999
*Description : Ce programme affiche un message sur l'écran
----------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
void main()
{
// Affichage d'une chaîne
printf(''Ce ci est un programme C++/n'') ;
}
```

Regardons sa structure. Le texte entre /* et */ est un commentaire. Main() est une fonction. Tout programme C++ est un ensemble de fonctions dont une et une seule doit s'appeler main(). Le mot clé void indique au compilateur que la fonction main ne renvoie aucune valeur.

Si la programme doit renvoyer une valeur au système d'exploitation. Le programme précédent s'écrira :

```
/* ----------------------------------------------------------------------------------------------------------
 *PROGRAMME 2
 *Fichier : affichei.cpp
 *Auteur :  Mustapha AIT MAHJOUB
 *Date : 10/09/1999
 *Description : Ce programme affiche un message sur l'écran et retourne une valeur au dos
----------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
int main()
{
// Affichage d'une chaîne
printf(''Ce ci est un programme C++/n'') ;
return(0) ;
}
```

En effet, il se peut que le système d'exploitation s'intéresse à la valeur renvoyée par main(). Par convention, 0 signifie succès, et toute valeur de retour non nulle désigne une erreur rencontrée durant l'execution du programme.

Les accolades { } marquent le début et la fin de la fonction main (). Les chaînes de caractères comme ''Ce ci est un programme C++\n '' sont délimitées par des guillemets. \n est le caractère retour à la ligne. La fonction printf ( )affichera la chaîne de caractères sur l'écran. // débute une zone de commentaire qui s'étendra jusqu'à la fin de la ligne. En fin, #include <stdio.h> donnera au

préprocesseur l'ordre d'inclure les lignes présentes dans le fichier d'en-tête stdio.h à l'intention du compilateur. Ce fichier contient, entre beaucoup d'autre, la déclaration de la fonction printf ().

3. TYPES DE DONNEES FONDAMENTAUX

Le type de données caractérisent des variables et des constantes en définissant leurs plages de valeurs et les opérations que l'on peut faire dessus. Le type indique en plus le nombre d'octets consécutifs sur les quelle s'étale la valeur. Une variable de type int (entier), par exemple occupe un mot de 2 octets sous DOS.
Les types de données se décomposent en deux catégories : les types prédéfinis et les types personnalisés. Les principaux types de C++ sont :

- type entier
- type réel
- type structuré
- type énumératifs

3.1 Données de type entier

Les données de type entier sont des objets avec un nombre complet (entier).Ces catégories incluent les constantes de type entier, les variables, les fonctions et expression.

Le tableau suivant donne les types entiers prédéfinis de C++, avec leurs tailles et leurs plages de valeurs relatives à l'environnement 16 bits.

Type	**Plage de valeurs**	**Taille**
char	-128 à127	1 octet
unsigned char	0 à 255	1 octet
short	-2^{15} à $+2^{15}-1$	2 octets
unsigned short	0 à2^{16}	2octets
int	-2^{15} à2^{15} –1	2 octets
unsigned int	0 à 2^{16} –1	2 octets
long	-2^{31} à $+2^{31}$	4 octets

TABLEAU 2 : Types entiers

C++ permet de spécifier des valeurs octales ou hexadécimales.

- Constante octale : elle commence par '0'' (Zéro)
- Constante Hexadécimale : elle commence par '0x''

```
I = 20        // 20 en décimal
I = 024       // 20 en octal
I = 0x14      // 20 en hexadécimal
```

3.2 Données de type réel

Les données de type réel se rapportent aux éléments de données qui représentent des objets numériques réels. Ceci inclut les constantes de type réel. Les variables, fonctions et expressions. Les types réels standards de C++ sont les suivants :

Type	Taille
Float	4 octets
Double	8 octets
Long double	10 octets

TABLEAU 3 : Types réels

Les nombres réels peuvent s'écrire soit sous la forme classique (par exemple 3.14), soit sous la forme scientifique (par exemple 2.45 E-5).

4. VARIABLES et CONSTANTES

4.1 Variables

Une variable est un identificateur qui permet de représenter une valeur modifiable. Une variable possède un nom qui doit être unique dans un certain domaine (portée). Outre le nom, une variable doit posséder un type de donnée précisant au compilateur, le nombre d'octets à allouer à la variable.

Une variable doit être déclarée, n'importe où dans le code, avant son utilisation dans le programme.

```
Int Somme ;              //déclaration d'une variable qui a pour nom somme et
                         //comme type, le type entier
float x, Delta;          // déclaration de deux variables de type réel.
Char c ;                 // déclaration d'une variable de type caractère
```

L'écriture des noms de variable doit respecter les règles d'écriture des identificateurs. Lorsqqu'on définit une variable, on peut lui fournir en même temps une valeur initiale. C'est cette valeur que la variable aura lors de sa première utilisation.

```
Int Somme =0 ;           // déclaration et initialisation des variables
Float x=10, Delta=12.3 ;
Char c=''0'' ;
```

4.2 Constantes

Il est souvent commode d'associer un élément de donnée simple, tel qu'une valeur numérique, à un identificateur, attribuant ainsi un nom à l'élément de donnée. On appelle l'identificateur une constante si on lui attribue un élément de donnée de manière définitive (la valeur reste inchangée tout au long du programme).

On pourrait considérer une constante comme une variable qui, une fois, initialisée, peut être lue mais non modifiée.

Les déclarations des variables et des constantes se ressemblent. La seule différence consiste en la présence du mot clé const.

```
Const int NombreMax= 100 ;
Const float SalaireMin= 1500.00 ;
```

```
/* ----------------------------------------------------------------------------------------------------------
 *PROGRAMME 3
*Fichier : surface.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date : 10/09/1999
*Description : Ce programme calcule et  affiche la surface d'un cercle
----------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
void main()
{
const Pi = 3.14 ; //Pi est une constante
float Rayon = 10.0 ;    //Rayon est une variable initialisée à 10
printf (''surface = %.2f'', Pi*Rayon) ; //affiche la surface du cercle
}
```

5. ENTREE / SORTIE DES DONNEES

5.1 Fonction printf ()

On utilise la fonction printf() pour afficher des données sur l'écran. Les éléments de données de sortie peuvent être des chaînes de caractères, des constantes numériques ou les valeurs de variables et d'expressions.

5.1.1 Sortie formatée

Le spécificateur de format ou formateur (comme %d ou %c) sont utilisés pour contrôler le format qui sera utilisé par printf() pour afficher une variable particulière. En général, le formateur doit correspondre au type de la variable à afficher. Nous utilisons, par exemple, le formateur %d pour afficher un entier et %c pour afficher un caractère. La liste suivante fournit les formateurs pour printf() et scanf() ;

Formateur	Description	Printf()	Scan()
%c	Un seul caractère	X	X
%s	Chaîne de caractères	X	X
%d	Entier signé	X	X
%f	Réel (notation décimale)	X	X
%e	Réel (notation exponentielle)	X	
%g	Réel (notation décimale)	X	
%u	Entier non signé (décimal)	X	
%x	Entier non signé (hexadécimal)	X	
%o	Entier non signé (octal)	X	
1	Préfixe utilisé avec %d, %u, %x, %o pour spécifier un entier long	X	

TABLEAU 4 : Caractères formateurs

Le programme suivant affiche l'entier i et le réel x avec les différents formats

```
/* ------------------------------------------------------------------------------------------------------------
 *PROGRAMME 4
*Fichier : format.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date : 10/09/1999
*Description : Ce programme affiche un entier et un réel sous différents formats
------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
void main()
{
int i = 255 ;
float x = 10.0 ;
printf (''%d %u   %x   %o\n'',i,i,i,i) ; //affichage d'un entier sous différents formats
 printf (''%f  %e   %g  \n'',x,x,x) ;   //affichage d'un réel sous différents formats
}
```

5.1.2 Largeur du champ

Considérons le programme suivant :

```
/* ------------------------------------------------------------------------------------------------------------
 *PROGRAMME 5
*Fichier : champ.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date : 10/09/1999
*Description : Ce programme affiche un réel sous différents longueur de champs
------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
void main()

{
float x = 10.56 ;
printf (''x = %f\n'' ,x) ;              //affiche 10.560000
printf (''x = %5.2f\n'',x) ;            //affiche 10.56
printf (''x = %8.3f\n'' ,x) ;           //affiche 10.560
}
```

Nous remarquons que la variable x est affichée, avec la première fonction printf (), avec 6 chiffres après la virgule alors que deux seulement de ces chiffres sont significatifs. Pour supprimer les zéros insignifiants, nous avons inséré la chaîne 5.2 entre le caractère % et f (seconde fonction printf()). Le nombre après le point décimal précise le nombre de caractères qui seront affichés après le point décimal. Le nombre qui précède la point spécifie l'espace à utiliser pour contenir le nombre lorsqu'il sera affiché.

5.1.3 Séquences d'échappement

Dans les exemples précédents, nous avons utilisé le caractère “/n” dans la chaîne de format pour insérer un retour à la ligne suivante. Le caractère nouvelle ligne /n est un exemple de séquence d’échappement. La liste suivante montre les principales séquences d’échapppement :

Séquence	Description
\n	Nouvelle ligne
\t	Tabulation
\b	Retour en arrière avec effacement
\r	Retour chariot
\’	Caractère apostrophe
\’’	Caractère guillemet
\\	Caractère anti-slash
\xdd	Code ASCII Hexadécimal
\ddd	Code ASCII décimal

TABLEAU 5 : Séquences d’échappement

5.2 Fonction scanf()

On utilise cette fonction pour lire des données, de différents types, à partir du clavier. Voici un exemple utilisant la fonction scanf() :

```
/* ----------------------------------------------------------------------------------------------------------------
 *PROGRAMME 6
*Fichier : var_con.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date : 10/09/1999
*Description : Ce programme lit une valeur et calcule une expression
----------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr() ;
const int JoursAn = 365 ;
float Annees, NombreJours ;
printff(‘’Entrez votre age :’’) ;

scanf(‘’%f’’ ,&Annees) ;
NombreJours = Annees*JoursAn ;
printf(‘’Votre age en nombre de jours = %.1f jours\n’’ ,Nombre Jours) ;


getch() ; //attend que l’utilisateur appuie sur une touche
}
```

Nous remarquons que le format de scanf () ressemble beaucoup à celui de la fonction printf () (voir tableau 4). Comme pour printf () l’argument à gauche est une chaîne qui contient le formateur ‘’%f’’ et à droite le nom de la variable &-annee. La fonction scanf () reçoit comme paramètre l’adresse de la variable annees d’où la présence de l’opérateur d’adresse &.

6. LES EXPRESSIONS

Une expression est un groupe d'opérandes liés par certains opérateurs pour former un terme algébrique qui représente une valeur. Il y' a deux sortes d'expression : numérique et logique. Une expression numérique donne comme résultat une valeur numérique, alors qu'une expression logique représente une condition qu'est vraie ou fausse.

6.1 Opérateurs arithmétiques

Ils sont utilisés pour effectuer des opérations de type numérique. Le tableau suivant décrit cinq opérateurs très courants qu'on peut utiliser pour créer des expressions numériques

Opérateurs	Signification
+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Reste de la division

TABLEAU 6 : Opérateurs arithmétiques

L'opérateur de division/ appliqué à des entiers donne un résultat entre de sorte que plusieurs divisions différentes peuvent donner le même résultat.

```
int resultat ;
resultat=3/3 ; //resultat = 1
resultat = 4/3 ; //resultat = 1
```

Si l'on ne désire pas perdre le reste, on peut employer l'opérateur %. Ce dernier, dont l'utilisation est réservée aux entiers, calcule le reste de la division de deux nombres.

```
int reste ;
reste = 3 % 3 ; // reste = 0
reste = 4 % 3 ; // reste = 1
```

Le programme 7 calcule le polynôme $ax^2 + bx + c$

```
/* ------------------------------------------------------------------------------------------------------------
 *PROGRAMME 7
*Fichier : polyncpp
*Auteur :  Mustapha AIT MAHJOUB
*Date : 10/09/1999
*Description : Ce programme calcule l'expression ax2 + bx + c
```

```
#include<stdio.h>
#include<conio.h> //pour clrscr() et getch()
void main()
{
clrscr() ;
float a,b,c,x,y ;
scanf(''%f %f  %f %f'' , &a, &b, &c, &x) ;
y = a* x* x + b* x + c ;
printf (''ax*x + bx + c = %g\n'',y) ;
getch() ;
}
```

6.2 Opérateurs de manipulation

Les opérateurs de bits agissent au niveau du bit. Les opérandes de ces opérateurs ne sont plus des entiers ou des caractères, mains tous simplement une suite de bits. Un opérateur de bits permet de manipuler les bits individuellement en les positionnant soit à 1 soit à 0. Les opérandes doivent être de type entier. Les opérateurs du tableau 7 suivant permettent d'agir sur des bits particulièrement à l'intérieur d'une variable.

Opérateur	Signification	Exemple x = 15 ou x = 1111 ou x = 0xF
~	Non(complément)	y = x y = 0xfff0
<<	Décalage à gauche	Y = x <<2 y = 0x3c
>>	Décalage à droite	y = x >> 2 y = 3
&	ET binaire	y = x & 0xa y =0xa
|	OU binaire	y = x | 0xa y = 0xf
^	OU-Exclusif	y = x ^ 0xa y = 5

TABLEAU 7 : Opérateurs de manipulation

6.3 Opérateurs d'incrémentation / decrémentation

L'expression n++ utilise l'opérateur d'incrémentation ++ qui a pour effet d'ajouter 1 à n. Ces opérateurs regroupent deux fonctionnalités : l'effet de bords qui est l'incrémentation ou la decrémentation, et la valeur qu'ils renvoient lorsqu'ils sont utilisés dans des expressions. Cette dernière dépend de la position relative du ++ ou du – par rapport à son opérande, par exemple si a = 5

```
b = 2*a++          // b =10, a = 6
b = 2*++a          // a =7, b =14
b = ++a-2          // a = 8, b = 6
```

b = 10 –(- -a) // a = 7, b =3

6.4 Opérateur sizeof ()

Cet opérateur renvoie la taille , en nombre d'octets, de l'expression au du type syntaxe :

seizeof< expression> Ou Sizeof(<type>)

Exemple : Sizeof (int) renvoie le nombre d'octet qu'occupe un entier

7. EXPRESSION BOOLEEENNE

Les expressions booléennes prennent des valeurs de vérité vraies ou fausses. Bien que nous parlions de valeurs logiques « vraie » et « fausse » n'oubliant pas qu'en fait ce ne sont que des valeurs numériques entières. Si le résultat de l'expression booléenne est 0, l'expression est fausse. Si le résultat est différent de zéro, l'expression est vrai.

7.1 Opérateurs de comparaison

Les opérateurs de comparaison sont tous des opérateurs binaires : ils relient deux opérandes. Le résultat de comparaison soit vrai(non nul), soit faux(nul).

Le tableau suivant décrit les opérateurs de comparaison.

Opérateur	**Signification**
<	Inférieur
>	Supérieur
<=	Inférieur ou égal
>=	Supérieur ou égal
= =	Egal
!=	Différent

TABLEAU 8 : Opérateur de comparaison

7.2 Opérateurs logiques

Le langage C contient trois opérateurs logiques. A l'instar des opérateurs des comparaisons , de résultat d'une expression utilisant les opérateurs logiques soit vrai (non nul), soit faux (nul). Le type de résultat est toujours int. Les opérateurs logiques sont toujours évalués de la gauche vers la droite.

Le tableau suivant décrit les opérateurs logiques.

Opérateur	Signification	Exemple a = 10, b =3
& &	ET logique	(a>0) && (b<10) est vraie
||	OU logique	(a = = 4) || (b= = 3) est vraie
!	NON logique	! (b = =3) est fausse

TABLEAU 9 : Opérateurs logiques

7.3 Opérateur conditionnel ternaire

Jusqu'à présent nous avons rencontré des opérateurs unaires ((NON binaire, incrémentation, etc.) qui manipulent un seul opérande et des opérateurs binaires (addition, division, etc.) qui manipulent deux opérandes. Il existe un opérateur ternaire qui manipule trois opérandes : l'opérateur arithmétique conditionnel. Voici sa syntaxe :

EXPRESSION 1 ;EXPRESSION 2 ; EXPRESSION 3 ;

On commence par évaluer Expression 1. Si elle est vrai (non nulle), alors on évalue Expression 2, si non on évalue Expression 3

Par exemple, pour calculer le maximum de deux nombres, on écrit,

```
Int a, b ; max ;
a = 5 ;
b = 10 ;
Max = (a>b) ; a :b ; // max = 10 car a n'est pas supérieur à b
```

8. INSTRUCTIONS D 'AFFECTATION

Nous avons déjà vu, dans les exemples précédents, que l'instruction d'assignation (ou affectation) sert à assigner un élément de données à une variable.
Considérons l'expression

```
X= 10 ;
```

Une fois l'affectation effectuée, la variable x vaut 10. Mais l'expression entière x = 10 possède aussi la valeur 10 de sorte qu'on peut écrire en C ++ l'expression

```
Y = x = 10 ;
```

Après traitement de l'instruction, x et y valent 10 tous les deux.

En C++, outre l'opérateur =, il existe plusieurs opérateurs d'affectation combinés comme le montre le tableau suivant :

Opérateur	**Signification**	**Exemple**	**Equivalent**
=	Affectation simple	x = 2	
*=	Affectation avec multiplication	x * = 2	x = x *2
/=	Affectation avec division	x /= 2	x = x /2
%=	Affectation avec modulo	x %= 2	x = x %2
+=	Affectation avec addition	x += 2	x = x +2
-=	Affectation avec soustraction	x - = 2	x = x -2
<<=	Affectation avec décalage à gauche	x<< = 2	x = x <<2
>>=	Affectation avec décalage à droite	x >>= 2	x = x>> 2

&=	Affectation avec ET binaire	x &= 2	x = x &2
|=	Affectation avec OU binaire	x |= 2	x = x |2
^=	Affectation avec OU exclusif binaire	x ^= 2	x = x ^2

TABLEAU 10 : Opérateurs d'affectation

9. PRIORITE DES OPERATEURS ET ORDRE D'EVALUATION.

Le tableau suivant résume à priorité de tous les opérateurs C++, incluant ceux non encore vus. Les opérateurs apparaissent dans un ordre de priorité décroisante

Les opérateurs sur une même ligne ont la même priorité et leurs opérandes sont évalués dans l'ordre indiqué dans la colonne ordre.

Opérateur	Description	Ordre
( )	Expression	gauche
Sizeof ~ !	Opérateurs unaire	droite
* / %	Opérateurs multiplicatifs	gauche
+ -	Opérateur additif	gauche
<< >>	Décalage binaire	gauche
< > <= >=	Opérateurs rationnels	gauche
= = ! =	Opérateurs d'égalité	gauche
&	ET binaire	gauche
^	OU binaire exclusif	gauche
|	OU binaire	gauche
&&	ET logique	gauche
||	OU logique	gauche
?=	Opérateur ternaire	droite
= * = / = % + = -= <<= >>= & = ^= |=	Affectations combinées	droite

TABLEAU 11 : Priorité des opérateurs

Quant on 'est pas certain de la priorité d'un opérateur, on doit utiliser des parenthèses pour assurer le bon ordre d'évaluation des opérateurs. Les parenthèses facilitent la compréhension des expressions.

Structures de contrôle

STRUCTURES DE CONTRÔLE

Le programme doit être conçu de telle sorte que les commandes s'exécutent dans un ordre logique et progressif. C'est ce qu'on appelle le déroulement d'un programme. Toutefois, si on veut que le programme répète plusieurs fois les mêmes instructions ou bien qu'il exécute une instruction que sous certaines conditions, l'approche séquentielle n'est pas pratique.

On utilise des mécanismes de contrôle permettant au programme de se brancher à des sous-programmes qui prendront en charge des tâches répétitives, répéteront une série d'instructions tant q' une condition reste vraie ou déclencher une action en fonction d'une condition. Ces trois types de mécanismes de contrôle sont connus respectivement comme branchement, boucles et condition (sélection). Les branchements, boucles et sélection sont tous des déviations temporaire du déroulement linéaire du programme. Ils vous permettent de créer des programmes avec souplesse et puissance.

1. STRUCTURE SEQUENTIELLE

Dans une structure séquentielle, chaque instruction est exécutée une fois et seulement une fois, dans le même ordre qu'elle apparaît dans le programme. Les instructions sont exécutées séquentiellement. Les programmes vus aux chapitres précédents sont de type séquentiel.

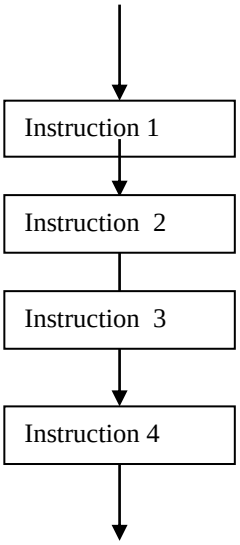
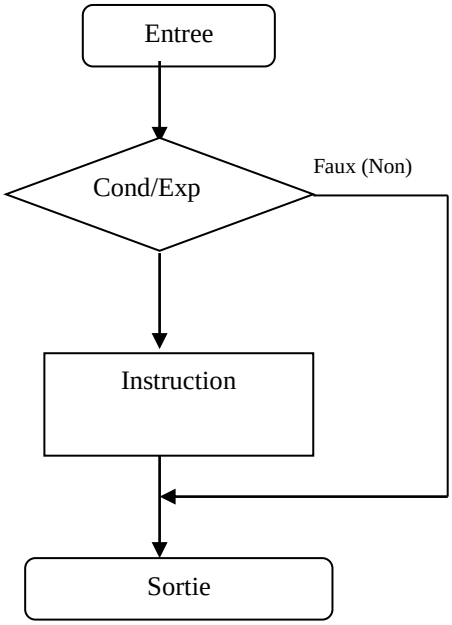


Figure 1 : Ordinogramme d'une structure séquentielle

2. STRUCTURE SELECTIVE (ALTERNATIVE)

Une structure de décision est une structure de contrôle conditionnelle qui autorise une action seulement si une condition donnée à une valeur non nulle (Vrai).

2.1 Structure if



If <condition/expression>
Instruction

Figure 2 : Ordinogramme et syntaxe de la structure if.

Quand le programme atteint la condition ou l'expression, il l'évalue. Si le résultat est vrai (non nul), le programme exécute Instruction, sinon (résultat nul) Instruction est pas exécutée.

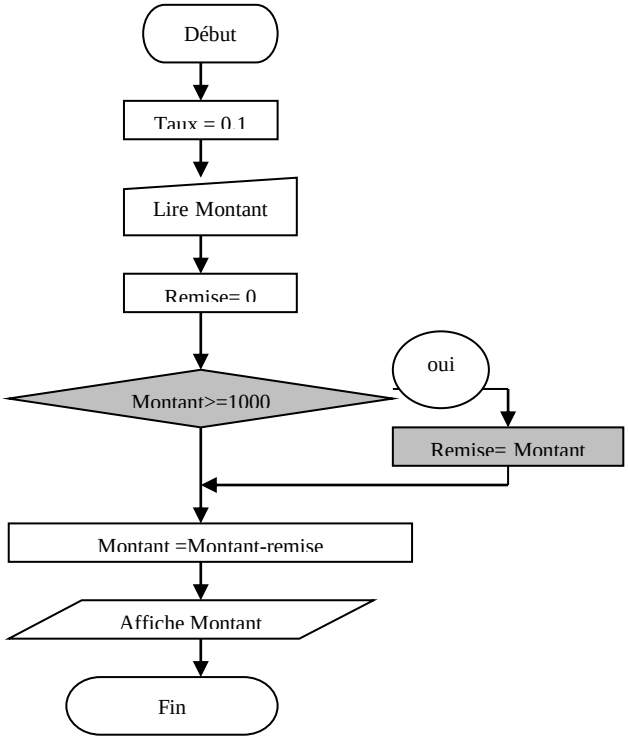


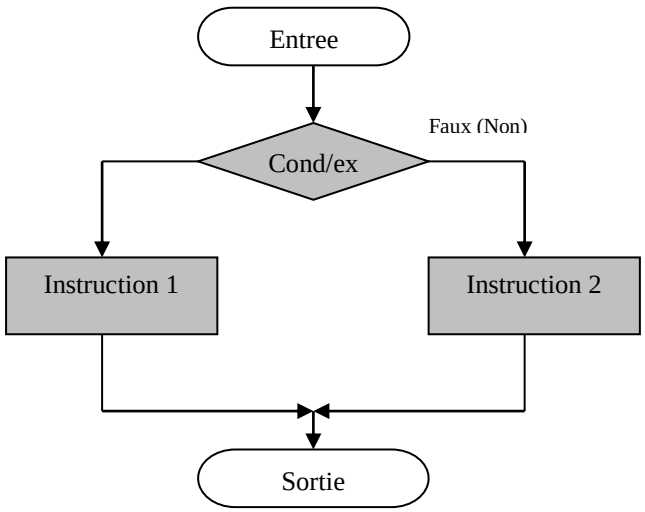
Figure 3 : Exemple d'ordinogramme pour if

```
/* ------------------------------------------------------------------------------------------------------------
*PROGRAMME 1
*Fichier : if.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :03/02/99
*Description : Calcule de la remise avec un seul taux
------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
void main()
{
        const float Taux=0.1 ;
        float Remise, Montant
        Remise=0.0 ;
        printf(''Enter montant ;'') ;
        scanf(''%f'', &Montant) ;
        if (Montant>=1000.0)
                Remise = Montant*Taux ;
        Montant =Montant-Remise ;
        printf(''Montant..=%.2f\n'', Montant) ;
}
```

L'exemple du programme 1 montre comment calculer le montant à payer par un client en lui accordant une remise. L'utilisateur saisit le montant. Si ce montant est supérieur à 1000, une remise avec un taux de 10% est faite (Remise Montant*taux), sinon pas de remise (Remise = 0). Après, le montant est calculé (Montant = Montant – Remise) puis afficher.

2.2 Structure if..else

L'ordinogramme et la forme de cette structure sont



If (<Condition/expression>)

Instruction 1

Else

Instruction 2

Figure 4 : Ordinogramme et syntaxe de la structure if..else

Quand le programme atteint la condition, il l'évalue. Si la condition est vraie (expression non nulle), instruction 1 est exécutée, sinon Instruction 2 est exécutée. L'une des instructions s'exécute toujours (mais jamais les deux).

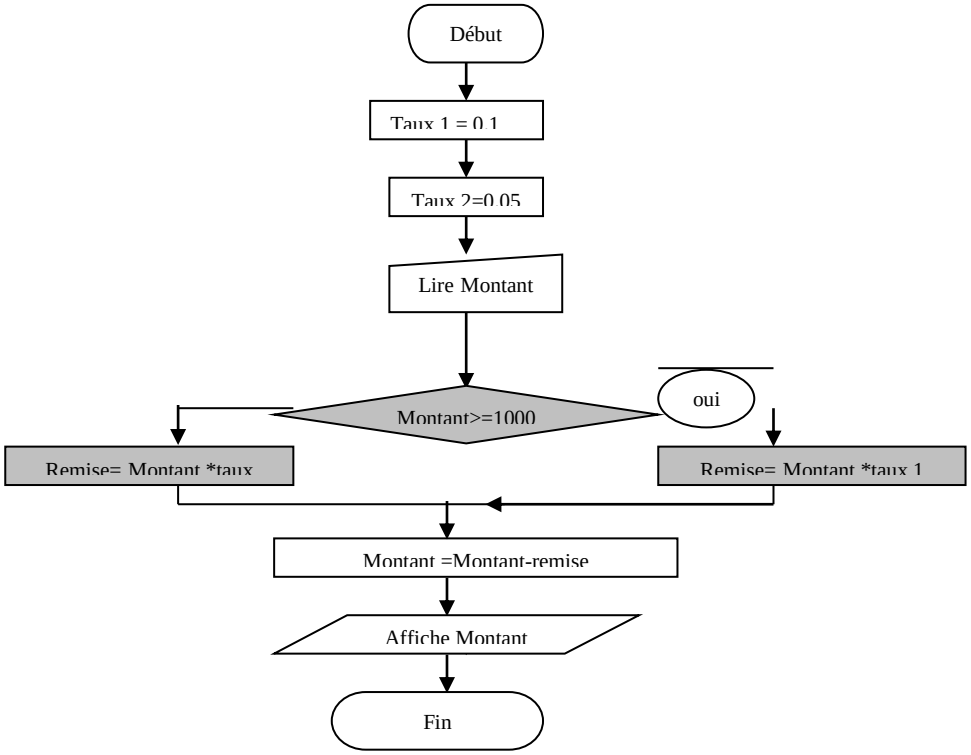


Figure 5: Exemple d'ordinogramme pour if..else

```
/* ----------------------------------------------------------------------------------------------------------------
*PROGRAMME 2
*Fichier : ifelse.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :03/02/99
*Description : Calcule de la remise avec deux taux
----------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include <conio.h>
void main()
{
        const float Taux 1=0.1, Taux 2=0,05 ;
        float Remise, Montant

        printf(''Enter montant ;'') ;
        scan(''%f'' ,&Montant :'') ;
        if (Montant>=1000.0)
                Remise = Montant*Taux 1;
```

```
        else
                Remise = Montant*Taux 2;
        Montant =Montant-Remise ;
        printf(‘’Montant..=%.2f\n’’, Montant) ;
        getch() ;
}
```

L’exemple du programme 2 montre comment calculer le montant à payer par un client en lui accordant une remise. L’utilisateur saisit le montant. Si ce montant est supérieur à 1000, une remise avec un taux de 10% (Taux 1) est faite. Sinon on accorde une remise avec un taux de 5% (Taux2). Après, on calcule le montant (Montant = Montant – Remise) puis on l’affiche.

2.3 Structures if imbriquées

Les structures if peuvent s’imbriquer l’une dans l’autre, tout comme n’importe quelle autre structure de contrôle. L’imbrication des structures if doit être manipuler avec beaucoup d’attention afin d’éviter toute ambiguïté possible.

L’exemple suivant montre la forme générale d’imbrication à deux niveaux que peuvent prendre les structures if imbriquées.

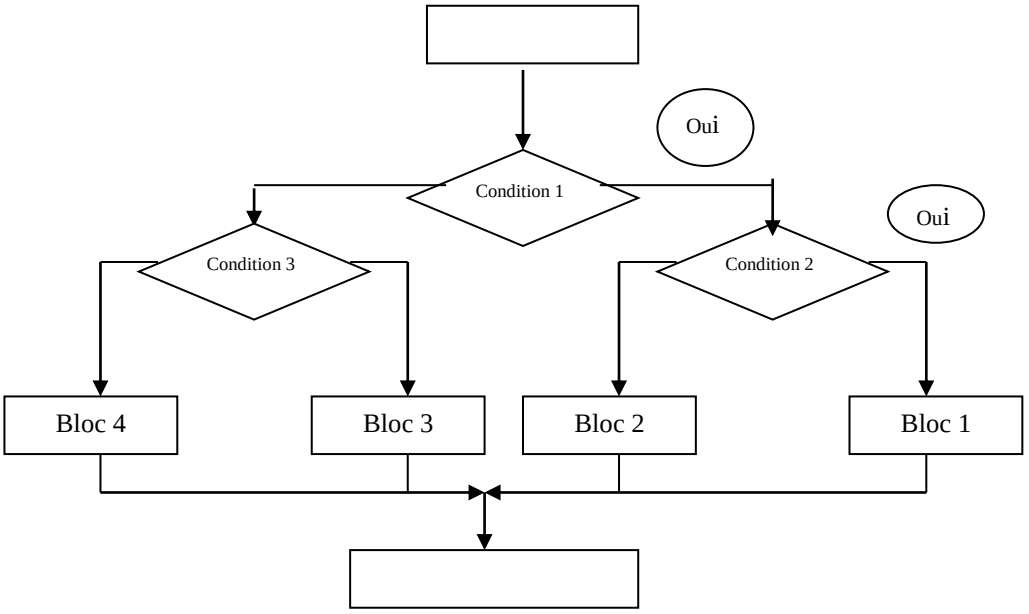


Figure 6 : Organigramme et syntaxe de la forme générale de structures if imbriquées

Dans cette forme d’imbrication à deux niveaux, Condition 1, Condition 2 et Condition 3 représentent des expressions, et Bloc 1, Bloc 2, Bloc 3 et Bloc 4 représentent des blocs d’instructions. Dans ce cas, une structure complète if..else sera exécutée si Condition 1 est vrai, et une autre si condition 1 est fausse. IL est évidemment possible que Bloc 1, Bloc 2, Bloc 3 et Bloc 4 contiennent d’autres structures if..else. Nous aurions alors une imbrication à plusieurs niveaux.

Programme 3 : Ecrivons un programme C qui résout une équation du deuxième degré

$Ax^2 + Bx + C = 0$

La résolution de l'équation se déroule de la manière suivante :

1. Lecture des coefficients A, B et C
2. Si la valeur attribuée à A est nulle (A = 0), il s'agit d'une équation de premier degré
   - 2.1 Si la valeur de B et zéro, deux cas se présentent :
     - 2.1.1 Si C = 0 :
       L'équation $Ax^2 + Bx + C = 0$ quelle que soit la valeur de x
       Dans ce cas la solution est l'ensemble des valeurs réelles.
     - 2.1.2 Sinon (C <> 0) : exemple C = 2
       L'équation $Ax^2 + Bx + C = 0 \Rightarrow 2 = 0$ ce qui est impossible.
       Donc l'équation n'a pas de solution.
   - 2.2 Sinon (B <> 0) : Exemple B = 5 ; C = 10
     L'équation à une seule solution qui est X = -C/B = -2

3. Sinon (A<> 0) :
   - 3.1 Calculer le déterminant delta = B^2 –4*A*C
   - 3.2 Si delta est inférieur ou égale à zéro, l'équation n'a pas de solution
   - 3.3 Sinon (delta supérieur ou égale à zéro : delta >= 0), deux cas possible
     - 3.3.1 Si delta = 0, dans ce cas on a une solution double
       X1 = X2 = -B/2*A
     - 3.3.2 Sinon (Delta> 0), dans ce cas on a deux solutions distinctes :
       X1 = (-B- SQR (delta))/2*A
       X2 = (-B+ SQR (delta))/2*A

4. Fin

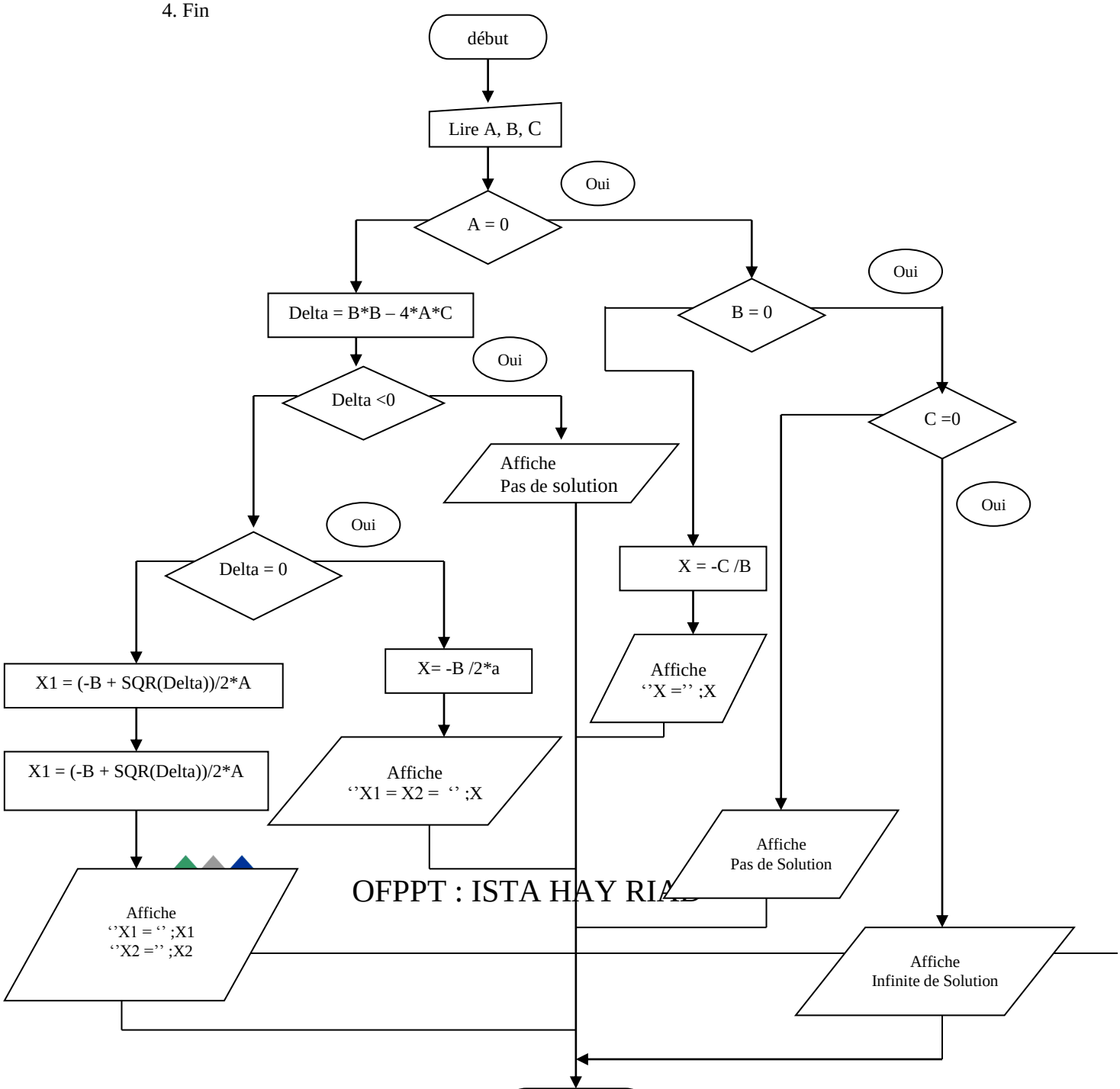


Figure 7 : Ordinogramme de résolution d'une équation de deuxième ordre

```
* ---------------------------------------------------------------------------------------------------------------
*PROGRAMME 3
*Fichier : resoudre.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :03/02/99
*Description : résolution d'une équation de deuxième degré
---------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<math.h> //prototype de sqrt
#include<conio.h>
void main()
{
        float A,B,C,Delta,X1,X2 ;
        clrscr() ;
        scan(''%f'' ,&A) ;
        scan(''%f'' ,&B) ;
        scan(''%f'' ,&C) ;
        IF ( !A)
        {
                if ( !B)
                {
                        if ( !C)
                                printf(''Infinité de solutions'') ;
                        else
                                printf(''Pas de solutions'') ;
                }
                else
                {
                        X1 = -C/B ;
                        Printf(''X = %g'' ,X1) ;
                }
        }
        else
```

```
        {
                Delta = B*B – 4*A*C ;
                if (Delta < 0)
                  Printf(‘’Pas de solutions relles’’) ;
                else
                  if ( !Delta)
                        Printf(‘’Solution double X1 = X2 = %g’’ ,-B/(2*A)) ;
                  else
                   {
                        X1 =(-B –sqrt(Delta))/(2*A) ;
                        X2 =(-B +sqrt(Delta))/(2*A) ;
                        Printf(‘’X1 = %g\nX2 = %g’’ ,X1,X2) ;
                   }
        }
}
```

2.4 Structure switch

La structure se définit comme une structure conditionnelle de contrôle qui permet de sélectionner un groupe particulier d'instructions parmi plusieurs groupes possible. La sélection s'effectue sur la base d'une expression, appelé sélecteur.

Voici la forme générale de la structure swich

```
Switch <expression>

{

        case <const-exp-1> :
                                <bloc1>
        case <const-exp-2> :
                                <bloc2>

        case <const-exp-3> :
                                <bloc3>


        case <const-exp-N> :
                                <blocN>
        default
                                <bloc>
        }
```

Forme de la structure switch

Le sélecteur <expression> peut appartenir à n'importe quel type entier. Il prend souvent la forme d'une seule variable de type entier.

Chacune des références de cas (cas <const-exp-1,case const-exp-2 ,…) représente une des valeurs constante autorisées du sélecteur. Donc si l'expression du sélecteur est de type entier, les références de cas représentent des valeurs entières se situant dans la gamme des possibilités. Les références de cas n'ont pas besoin de figurer dans un ordre particulier, mais chacune d'entre elles demeure cependant unique.

En cas de non-correspondance, et il existe un label default, le contrôle passe au bloc d'instruction <Bloc>.

Dans le programme suivant, le sélecteur Choix est une variable de type char. Suivant la valeur fournie par l'utilisateur à cette variable, le traitement sera aiguillé à l'instruction dans l'étiquette correspond à celle fournie. Si par exemple, l'utilisateur a tapé le caractère + l'instruction Resultat = Valeur 1 + Valeur 2 sera exécuté puis le contrôle passe à l'instruction qui suie le bloc le switch.

L'instruction break permet de sortir du blos de l'instruction switch.

```
* ---------------------------------------------------------------------------------------------------------------
*PROGRAMME 4
*Fichier : case.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :04/02/99
*Description : Calcul la somme, la différence , le produit ou le quotion de deux nombres
---------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
void main()
{
        char Choix ;
        float Valeur 1 , Valeur 2, Resultat ;
        clrscr() ;
        printf(''Addition……(+)/n'') ;
        printf(''Soustraction……(-)/n'') ;
        printf(''Multiplication ……(*)/n'') ;
        printf(''Division……(/)/n'') ;
        printf(''Entrer votre choix : '') ;
        Choix = getche() ;
        printf(''\nValeur 1 = '') ;
        scanf(''%f'',&Valeur 1) ;
        printf(''\nValeur 2 = '') ;
        scanf(''%f'',&Valeur 2) ;
        switch(Choix)
        {
                case'+' :{Resultat = Valeur 1 +Valeur 2 ; break ;}
                case'-' :{Resultat = Valeur 1 -Valeur 2 ; break ;}
                case'*' :{Resultat = Valeur 1 *Valeur 2 ; break ;}
                case'/' :
                        if (Valeur 2 = = 0)
                                printf(''\nDivision par zéro'') ;
                        else
                                Resultat = Valeur 1/Valeur 2 ;
                                Break ;
                Default :
                        printf(''\n Opérateur incorrect'') ;
        }
        printf(''\n Resultat = %g'', Resultat) ;
        getch() ;
}
```

3. STRUCTURES ET REPETITIVE S (BOUCLES)

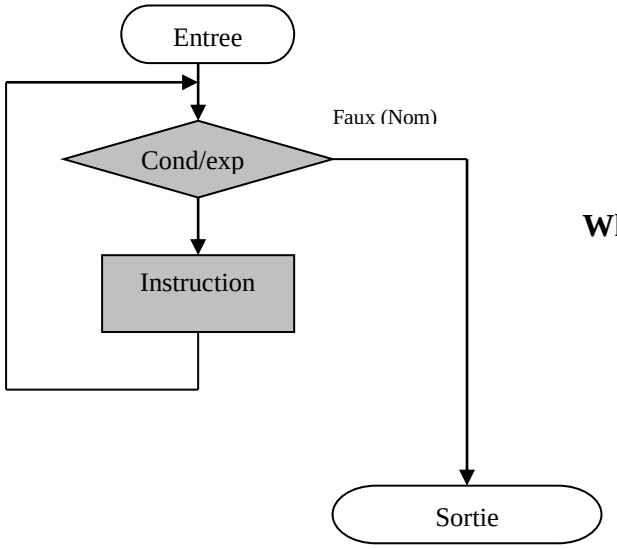
Vous pouvez désirer que votre programme exécute les mêmes instructions de façon répétitive tant qu'une certaine condition reste vraie.

En programmation, ceci s'appelle une boucle. Le programme répète les mêmes instructions dans une boucle jusqu'au moment où la condition n'est plus vraie (ou n'est plus fausse).

C possède plusieurs formes de boucles qui seront présentés dans la suite.

3.1 Structure while

La structure while correspond à une instruction de contrôle répétitive utilisée pour la réalisation de boucles conditionnelles. Voici l'organigramme et le forme générale de cette structure :



While (<Condition/Expression>)

instruction

Figure 8 :Organigramme est formé de la structure while

Instruction s'exécute continuellement aussi longtems que l'expression booléenne <Condition/Expression demeure vraie (non nulle). Dès que <condition/Expression> devienne fausse, on sort de la boucle.

Servons nous de la structure while pour obtenir la moyenne d'une liste de N nombres. Nous utilisons une variable Somme initialement mise à zéro et mise à jour à chaque lecture d'un nombre nouveau dans la mémoire.

Supposons que l'utilisateur a donné la valeur 3 à la variable N. En entrant dans la boucle la variable Compte a pour valeur 1 donc inférieur à N. La condition « Compte = N » est vraie et les instructions continuent dans le bloc de while seront exécutées. On lit un nombre(Nombre = 10) puis on l'ajoute à l'ancienne somme (Compte = 0 + Nombre = 10) ensuite, la variable compte est incrémentée de 1 (Compte = Compte + 1 = 1 + 1 = 2). La variable compte est comparé à une deuxième fois avec la variable N. Le résultat de la comparaison est vrai(Compte<= N ou 2 <= 3) donc la boucle est exécutée une deuxième fois en lisant un nouveau nombre(Nombre = 1) suivi du calcul de la somme (Somme = Somme + Nombre = 10 + 20 = 30) et l'incrémentation de compte (Compte = Compte + 1= 2 + 1 = 3). La condition est toujours vraie. La boucle sera exécutée une troisième fois. Un nombre est lu(Nombre = 60) puis ajouté à l'ancienne somme (Somme =30 + 60 = 90)suivi de l'incrémentation de compte (Compte = Compte + 1 = 3 + 1 = 4). La variable compte est comparée avec N (4<= 3). Le résultat de la

comparaison est faux . On sort de la boucle pour calculer la moyenne et afficher la somme et la moyenne.

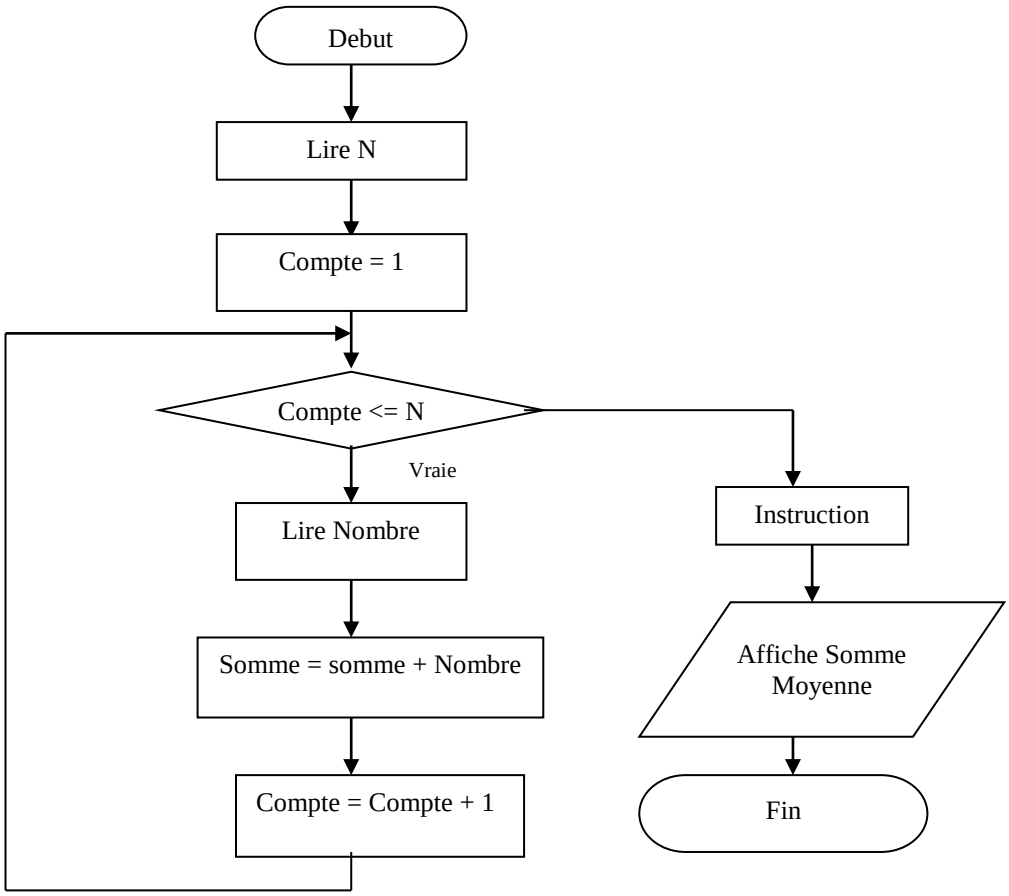


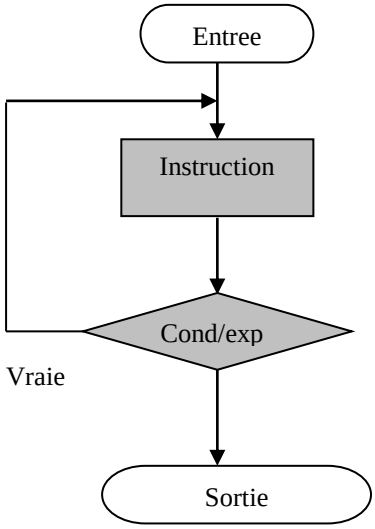
Figure 9 : Organigramme de calcul de la moyenne des nombres

```
* ---------------------------------------------------------------------------------------------------------------
*PROGRAMME 5
*Fichier : while.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :03/02/99
*Description : calcul de la moyenne en utilisant la boucle while
---------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
void main()
{
        float Somme = 0.0, Nombre, Moyenne ;
        int Compte = 1,N ;
        printf(''Nombre d'éléments :'') ;
        scanf(''%d'',&N) ;
        while (Compte <= N)
        {
```

```
            printf(''Un nombre :'') ;
            scanf(''%f'',&Nombre) ;



            Somme +=Nombre ;
            Compte++ ;
        }
        Moyenne = Somme /N ;
        printf(''Somme =%g\n'' ,Somme) ;
        printf(''Moyenne = %g'' ,Moyenne) ;
        getch() ;
}
```

3.1 Structure do..while

L'organigramme et la forme générale de cette structure sont



do

instruction

While (<Condition/Expression)

Figure 10 : Organigramme est forme de la structure do..while

Instruction est exécuté continuellement jusqu'à ce que la condition <Condition/Expression> soit égale à zéro (fausse). La différence avec while est que <Condition/Express> est testée après, et non avant, chaque exécution de la boucle. Le système suppose qu'il y a au moins une exécution du bloc d'instructions.

Le programme suivant utilise la structure do..while pour calculer la moyenne des notes.

```
* ------------------------------------------------------------------------------------------------------------
*PROGRAMME 6
*Fichier : do.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :03/02/99
*Description : calcul de la remise en  utilisant la boucle do
```

----------*/

```
#include<stdio.h>
#include<conio.h>
void main()
{
        float Somme = 0.0, Nombre, Moyenne ;
        in Compte = 1,N ;


        printf(‘’Nombre d’éléments :’’) ;
        scanf(‘’%d’’,&N) ;
        do
        {
                printf(‘’Un nombre :’’) ;
                scanf(‘’%f’’,&Nombre) ;
                Somme +=Nombre ;
                Compte++ ;
        }
                while (Compte <= N) ;
                Moyenne = Somme /N ;
                printf(‘’Somme =%g\n’’ ,Somme) ;
                printf(‘’Moyenne = %g’’ ,Moyenne) ;
                getch() ;
}
```

3.2 Structure for

La format de cette structure est

for(<Init/Exp> ; <Test/Exp> ;<Incr/Exp>)
<Bloc d’instruction>

La séquence d’événement est la suivante :

1. L’expression d’initialisation <Init/exp> est exécutée. Comme son non l’indique, il s’agit d’une instruction qui initialise un ou plusieurs compteurs de boucle mais la Syntaxe permet une expression d’un quelconque degré de complexité(y compris des déclarations). D’où l’affirmation que tout programme C peut être écrit sous la forme d’une seule boucle for.

2. L’expression <Test/Exp est évaluée suivant les règles de la boucle while. Si <Test/Exp> ets différente de zéro (vraie), la boucle est exécutée. Une expression vide est prise ici comme while (1 ou vrai) . Si <Test/Exp> est égal à zéro (fausse).L a boucle prend fin.

<Incr/Exp>incrémente les compteurs

Le Bloc d’instruction est exécuté et le contrôle revient à 2.

Le programme suivant utilise la structure for pour calculer la moyenne des notes.

```
 *-------------------------------------------------------------------------------------------------------------
*PROGRAMME 7
*Fichier : for.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :03/02/99
*Description : calcul de la moyenne en  utilisant la boucle for
-------------------------------------------------------------------------------------------------------------*/



#include<stdio.h>
#include<conio.h>
void main()
{
        float Somme = 0.0, Nombre, Moyenne ;
        in Compte = 1,N ;
        clrscr() ;
        printf(‘’Nombre d’éléments :’’) ;
        scanf(‘’%d’’,&N) ;
        for(Compte = 1 ; Compte <= N ; Compte ++)
        {
                printf(‘’Un nombre :’’) ;
                scanf(‘’%f’’,&Nombre) ;
                Somme +=Nombre ;
        }

        Moyenne = Somme /N ;
        Printf(‘’Somme =%g\n’’ ,Somme) ;
        Printf(‘’Moyenne = %g’’ ,Moyenne) ;
        getch() ;
}
```

Si dans cet exemple, l’utilisateur a donné la valeur 3 à la variable N. Compteur prendra successivement les valeurs 1,2 et 3, ce qui cause trois exécutions de la boucle à chaque fois, un nombre est lu puis ajouter à Somme. La sortie de la boucle for est faite lorsque la variable compte aura atteint la valeur de N.

4. INSTRUCTIONS DE BRANCHEMENT

Une instruction de branchement, lorsqu’elle est exécutée, passe le contrôle à la suivante. Il existe quatre instructions de branchement :break, continue, goto et return.

4.1 Instruction break

Une instruction break ne peut s'utiliser que dans itération (while, do..while et for) ou un switch. Elle marque la fin d'une instruction d'itération ou de switch.

Le programme suivant determine la valeur de N pour laquelle la somme 0 + 1+ 2 +……+ N est supérieur ou égale à une valeur donnée.

```
* ---------------------------------------------------------------------------------------------------------------
*PROGRAMME 8
*Fichier : break.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :05/02/99
*Description : utilisation de break dans un boucle while
---------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>




void main()
{
        int Valeur, Somme, N ;
        Somme = N = 0 ;
        clrscr() ;
        scanf(‘’%d’’,&Valeur ;
        while(1) //condition est toujours vraie
        {
                if (Somme >= Valeur) break ;//sortie forcée de la boucle
                Somme +=N++ ;
        }
        printf(‘’%d\n ‘’,N-1) ;
        getch() ;
}
```

4.2 Instruction continue

La syntaxe de cette instruction est

Continue ;

Elle n'est utilisable que dans une instruction d'itération.. Elle passe le contrôle à la condition de test pour les boucles while et do, et à l'expression d'incrémentation dans une boucle for.

Dans le cas de boucles d'itération imbriquées, une instruction continue est considérée comme appartenant à l'itération la plus proche la contenant.

```
* ---------------------------------------------------------------------------------------------------------------
*PROGRAMME 9
*Fichier : continue.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :05/02/99
```

```
*Description : utilisation de mot clé continue dans un boucle while
-----------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
void main()
{
        int Valeur, Somme, i, Nombre ; ;
        Somme = i = 0 ;
        Clrscr() ;

        While (i+ + <5)
        {
                Scanf(‘’%d’’,&Nombre;
                If (Nombre< 0) continue ;
                Somme +=Nombre;
        }
        printf(‘’%d\n ‘’,Somme) ;
        Getch() ;
}
```

4.3 Instruction goto

La syntaxe est

Goto label ;

L’instruction goto passe le contrôle à l’instruction label qui doit être dans la même fonction.

Il est illégal de sauter une instruction de déclaration ayant une valeur d’initialisation implicite ou explicite à moins que cette déclaration ne soit dans un bloc interne déjà ignoré.

Types de données complexes

TYPE DE DONNEES COMPLEXES

1. LES TABLEAUX

1.1 Tableaux unidimensionnels

Un tableau unidimensionnel est une liste de données de même type auxquels on se refaire collectivement par le même le nom. On peut accéder à chaque élément individuel en spécifiant le nom du tableau, suivi par un indexe (ou indice) cerné par des crochets. Donc si une liste de note d'un tableau unidimensionnel contient N éléments, les éléments individuels du tableau sont Note [0], Note[1]……Note [N-1]

1.1.1 Déclaration d'un tableau

La déclaration d'un tableau comprend trois parties :
- L'indication de type
- Le nom du tableau.
- Le nombre d'éléments (entre crochet) du tableau (ou taille).

<type> <non> [<nombre d'élément>] ;

L'exemple suivant est une déclaration d'un tableau de 25 notes :

```
Float Note [24] ;
```

Ou

```
Const int N = 24 ;
Float Note [N] ;
```

La taille du tableau doit être une expression constante (connu au moment de la compilation).

1.1.2 Initialisation des tableaux

On peut initialiser un tableau lors de sa déclaration. Il suffit de fournir la liste des valeurs des divers éléments placées entre accolades et séparées entre elles par des virgules.

```
Int JoursMois [11] = {31,30,31,30,31,30,31,31,30,31,30,31} ;
```

Quand on initialise un tableau dans sa déclaration, on peut omettre la taille. Celle ci est déterminée par le compilateur par le nombre de valeurs d'initialisation. Le tableau JoursMois peut être déclaré par

```
Int JoursMois [] = {31,30,31,30,31,30,31,31,30,31,30,31} ;
```

Les points suivants doivent être respecter lors de l'utilisation des tableaux :
- On ne peut pas initialiser un tableau à partir d'un autre tableau.
- Les affectations entre tableaux dont interdites

```
Float Vec1[10], Vec2[10] ;

Vec 1 = Vec 2 ; // erreur
```

- Toute expression donnant un résultat entier peut être utilisée comme indice

```
Int i,j ;
Float x = 1.0 ;
Float Vect[10] ;
Vec[5] = Vec[i + j] ; //correct i + j est un entier
Vec[4] = Vec[x] ; //correct x n'est pas un entier
```

- Le compilateur ne vérifie pas si l'indice d notes et en calculer la moyenne et l'écart de chacune de ces notes par rapport à la moyenne à l'aide de la formule.

```
Ecart[i] = Note[i] – Moyenne.
```

```
* ---------------------------------------------------------------------------------------------------------------
*PROGRAMME 1
*Fichier : tab_moy.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :02/02/99
*Description : utilisation des tableaux pour calculer les écarts simples
---------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
void main()
{
        float Somme,Moyenne ;
        float Note[MAX_ELEMENT] ;
                Ecart[MAX_ELEMENT] ;
        Int i,N ;
        // Lecture des notes
        clrscr() ;
        printf(''Nombre d'éléments :'') ;
        scanf(''%d'',&N) ;
        for (i=0 ;i<N ; i++)
        {
                printf(''Note[%d]'' i) ;
                scan(''%f'',&Note[i]) ;
        }
        //calcul de la somme des notes
        for (i=0, Somme =0 ; i<N i++)
                Somme +=Note[i] ;
        //calcul de la moyenne
        Moyenne = Somme /N ;
        //calcul des écarts simples
```

```
        for (i=0 ; i<N ; ++)
                Ecart[i] = Note[i] – Moyenne ;
        //Affichage des résultats
        clrscr() ;



        printf(‘’Moyenne =%2f\n’’, Moyenne) ;
        printf(‘’Note\t\tEcart\n’’) ;
        for (i=0 ; i<N ; i++)
                printf(‘’%.2f\t\t%.2f n’’,Note[i],Ecart[i]) ;
        getch() ;
}
```

1.2 Tableaux multidimensionnels

Nous avons déjà envisager un tableau unidimensionnel sous la forme d'une liste (sur une seule colonne) de données de même type . Les tableaux multidimensionnels constituent une extension de ce concept. On peut ainsi imaginer un tableau à deux dimensions comme un tableau de données composé de lignes et de colonnes. Supposons, par exemple, que Matrice est tableau à deux dimensions contenant M lignes et N colonnes. Les éléments individuels du tableau sont Matrice [0] [0],..,Matrice[0] [N-1],…,Matrice[M-1] [0],…,Matrice[M-1] [N-1].

Le tableau se visualise de la manière suivante :

```
                Matrice[0] [0]……Matrice[0] [N-1]
                    .                   .
                    .                   .
Matrice =           .                   .
                    .                   .
                    .                   .
                Matrice [M-1][0]    Matrice [M-1][N-1]
```

1.2.1 Déclaration d'un tableau multidimensionnel

Quelle que soit sa dimension, un tableau comprend toujours une collection de données de même type.

On peut définir un tableau multidimensionnel dans une déclaration de variable, de la même façon que pour chacune des dimensions du tableau

<type> <nom> [nombre de ligne>][<nombre de colonne>] ;

Par exemple,

```
Float Matrice[3][3] ;
```

Matrice est tableau à deux dimensions. Il peut stocker 3x3=9 éléments de type float.

L'accès aux différents éléments se fait comme avec les tableaux à une dimension, à ceci près que l'on a besoin de deux indices. Par exemple, pour calculer la somme des éléments de la première ligne

Somme = Matrice[0][0] + Matrice[0][1] + Matrice[0][2] ;

1.2.2 Initialisation d'un tableau multidimensionnel

On peut initialiser un tableau multidimensionnel dès sa déclaration. Par exemple, pour initialiser Matrice à zéro

```
float Matrice [][] = {
                        {0,0,0},
                        {0,0,0},
                        {0,0,0}
                    }

* ---------------------------------------------------------------------------------------------------------------
*PROGRAMME 2
*Fichier : tab_mat.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :02/02/99
*Description : calcul la somme et le produit de deux matrices
---------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
#definie MAX_LIGNE 10
#definie MAX_COLONNE 10
// Définnition d'un nouveau type MATRICE
type de f float MATRICE[MAX_LIGNE][MAX_COLONNE] ;
void main ()
{
        int ligne,colonne ;
        MATRICE Mat1,Mat2,MS,MP ;
        Clrscr() ;
        printf(''Matrice1\n '') ;
        For (inti=o ;i<ligne ;i++) // boucle de lignes
        For (intj=o ;j<colonne ;i++) // boucle de colonnes
        {
                printf(''Mat1 ![%d],[%d]='',i,j) ;
                scanff(''%f',&Mat1[i][j] ;
        }
        //lecture de la deuxième matrice
        printf(''Matrice2\n'') ;
        for(i=0 ;i<ligne ;i++)
                for (int j=0 ;j<colonne ;j++)
                        {
                                printf(''Mat2[%d],[%d]='',i,j) ;
```

```
                        scanf(''%f'',&Mat2[i][j] ;
                }
        //somme de deux matrices
        for(i = 0 ;i<ligne ;i++)
                for (int j=0 ;j<colonne ;j++)
                        MS[i][j] = Mat 1[i][j] + Mat2[i][j] ;
        //Produit de deux matrices
        for (i=0 ; i<ligne ; i++)
                for (int j=0 ; j<colonne ; j++)
                        {
                                MP [i][j] = 0.0 ;
                                //produit d'une ligne par colonne
                                for (int k=0 ; k<ligne ; k++)


                                {
                                        MP[i][j] =MP[i][j] + Mat1[i][k]*Mat2[k][j] ;
                                }
                        }
        //affichage matrice somme
        printf(''Matrice Somme\n'') ;
        for (i=0 ; i<ligne ; i++)
                for (int j=0 ; j<colonne ; j++)
                {
                        printf(''MS[%d][%d]=%g\n'',i,j,ms[i][j] ;
                }
        //affiche matrice produit
        printf(''Matrice Produit\n'') ;
        for(i=0 ; i<ligne ;i++)
                for(int j=0 ; j<colonne ; j++)
                {
                        printf (''MP[%d][%d]=%g\n'',i,j,MP[i][j]) ;
                }
        getch() ;
}
```

2. LES CHAINES DE CARACTERES

Les chaînes de caractères sont des formes de données utilisées pour stocker et manipuler le texte. En C++, il n'existe pas de type de données spécifique aux chaînes de caractères. Pour gérer des chaînes de caractères, il faut donc passer par des tableaux de chair. L'exemple suivant définit une chaîne de caractères.

```
Char Nom[] = ''Nacer'' ;
```

Ou

```
Char Nom[6] = ''Nacer'' ;
```

On reserve ici, de l'espace mémoire pour stocker 6caractères. Les 5 premiers sont initialisés avec les caractères 'N', 'a', 'c', 'e', 'r'. Le dernier est initialisé avec le caractère terminateur nul '\o' marquant la fin de la chaîne.

On peut, aussi, initialiser une chaîne avec une liste de caractères

Char Nom[] = { 'N', 'a', 'c' , 'e', 'r', '-0'} ;

2.1 Fonction d' E / S des chaînes de caractères

La fonction, de la librairie C++, gets() permet de saisir une chaîne à partir du clavier . Elle n'est pas universelle comme scanf() ; elle est écrite seulement pour lire une chaîne. Le caractère retour chariot indique la fin de la lecture d'une chaîne. On peut ainsi inclure des espaces et des tabulations comme caractères de la chaîne.

La fonction puts(), qui va en pair avec gets(), permet d'afficher une chaîne sur l'écran. L'exemple suivant lit une chaîne puis l'affiche sur l'écran :

```
Char Nom[81] ;

gets(Nom) ;// le nombre de caractère inférieur à 81
puts(Nom) ;
```

2.2 Fonctions de traitement des chaînes

La bibliothèque C++ fournit un ensemble de fonctions de traitement des chaînes. Pour les utiliser, il suffit d'inclure le fichier d'en tête string h dans votre programme, voici quelques unes de ces fonctions (pour plus de détailles consulter l'aide) :

strlen(chaîne) :calcul la longueur d'une chaîne
strcpy(destination,source) :copie une chaîne dans une autre
strcpy(destination,source,N) :copie N caractères de la chaîne source dans la chaîne destination.
Sttrcat(destination,source) :concatène destination et source.
Strncat(destination,source,N) :concatène Ncaractère de la source avec destination
Strcmp(destination,source) :compare destination et source.

Les programmes 3 et 4 suivants montrent l'utilisation de quelques fonctions de manipulation des chaînes.

```
* ---------------------------------------------------------------------------------------------------------------
*PROGRAMME 3
*Fichier : chaînes.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :02/02/99
*Description : manipulation des chaînes
---------------------------------------------------------------------------------------------------------------*/
#include<conio.h>
#include<stdio.h>
#include<string.h>
void main ()
{
```

```
        char ch1[20]=''Bon'',ch2[20]=''Jour'' ;
        int i ;
        printf(''Longueur de chaîne 1 :%d'',strlen(ch1)) ;
        strcpy(ch1,che) ;
        puts(ch1) ; //affiche ''Jour''
        strcpy(ch1,''Bon'') ;
        strncpy(ch1,ch2,2) ;//les 2 caractères ''JO'' de ch2 remplace ''BO''
        puts(ch1) ;
        strcpy(ch1,''Bon'') ;
        strcat(ch1,ch2) ;
        puts(ch1) ;//affiche ''BonJour''
        strcpy(ch1,''Bon'') ;
        strncat(ch1,ch2,1) ;
        puts(ch1) ;//affiche ''BonJ''
        strcpy(ch1,''Jour'') ;
        i= strcmp(ch1,ch2) ;
        printf(''%d\n'',i) ;//affiche 0 puisque ch 1 = ch 2
        getch()
}



* ---------------------------------------------------------------------------------------------------------------
*PROGRAMME 4
*Fichier : inverse.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :02/02/99
*Description : inverse une chaîne, par exemple NACER devient RECNA
---------------------------------------------------------------------------------------------------------------*/
#include<conio.h>
#include<stdio.h>
#include<string.h>
void main ()
{
        char ch1[20],ch2[20] ;
        int L ;
        clrscr()
        printf(''Enter une chaîne ;'') ;
        gets(Ch1) ;//saisir abonjour
        L = strlen(Ch1) ;//L = 7
        Strcpy(Ch2,Ch1) ;//Ch2 = ''BONJOUR''
        For (int i=0 ; i<L ; i++)
        Ch2i = Ch1{L-i-1] ;

puts(Ch2) ;//affiche « RUOJNOB »

Getch()

}
```

3. LES STRUCTURES

Nous avons déjà vu comment une variable simple peut contenir une seule information d'un type donné et comment un tableau peut contenir une collection d'information de même type . Ces deux mécanismes de stockage des données peuvent résoudre une grande variété de problèmes. Mais, il arrive souvent qu'on désire utiliser des ensembles d'éléments de données de différents types comme une seule entêtée dans ce cas, l'utilisation des variables et des tableaux est inadéquate.

Supposons qu'on désire stocker le nom d'un employer, son salaire et le nombre de ses enfants. On peut stocker ces informations dans trois variables séparées : une pour le nom, une autre pour le salaire et une autre pour le nombre d'enfants. Cette façon de faire n'est pas pratique : dans la vie, il est naturel, de regrouper les éléments (Nom, Salaire et Nombre Enfants) en une seule entité appelée structure.

Les structures sont une caractéristiques très puissante de C++, et facilite l'organisation et le traitement des informations.

3.2 Déclaration d'une structure

Pour déclarer une structure, on utilise le mot clé struct, puis le nom de la structure. En suite, entre accolades, les variables qui constituent la structure, par exemple :

```
Struct employe] ;]
{
        char Nom[81]
        float Slaire ;
        int Nombre Enfants ;
} ;
```

Cette instruction définit un nouveau type de données appelé struct employer. Caque variable de ce type est constituée de 3 éléments (ou membres) : une variable chaîne de caractères appelée Nom, une variable de type réel Salaire, et une variable de type entier Nombre Enfants.

Il faut noter que cette instruction ne déclare aucune variable donc il n'y a pas de réservation mémoire. Il indique, simplement au compilateur les différents éléments de la structure.

3.3 déclaration de variables d'une structure

Une fois, on a définit la structure, on peut déclarer une ou plusieurs variables qui ont pour type structure on tapant le nom de la structure, puis le nom de la variable.

Pour définir une variable de la structure employée , on écrit

```
        employe Salaire ;
```

Cette instruction réserve de l'espace mémoire, pour la variable Salaire pour contenir tous les éléments de la structure : 81 octets pour Nom, 4 octets pour Salaire, et 2 octets pour Nombre Enfants.

3.3 Accès aux membres de la structure

Après avoir créé une structure, puis une variable de cette structure, on accède aux membres en tapant le nom de la variable, puis un point, puis le nom du membre.

Par exemple, pour affecter des valeurs aux différents membres de la variable Salaire, on écrit :

```
Salaire.Nom[] = ''Hassan'' ;
Salaire.Salaire = 3000.0 ;
Salaire.NombreEnfants = 2 ;
```

L'opération point, connecte le nom de la variable structure avec les membres de cette structure.

Le programme suivant utilise la structure employe

```
* ---------------------------------------------------------------------------------------------------------------
*PROGRAMME 5
*Fichier : structur.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :02/02/99
*Description : manipulation des structures
---------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
void main ()
{
        const float Allocation = 150.00 ;
        struct employe//declaration de la structure
                                {       char Nom[30]
                                        float Salaire Base ;


                                        int Nombre Enfants ;
                                } ;
        float Salaire ;
        employe Agent ;//déclaration d'une variable de la structure
        clrscr() ;
        printf(''Enter le nom'') ;
        gest(Agent.Nom) ;
        printf(''Enter salaire de base :'') ;
        scanf(''%f',&Agent.Salaire Base) ;
        printf(''Enter Nombre d'enfants : )
        Slaire = Agent.Salaire Base + Agent.Nombre Enfants*Allocation.
        printf(''Salaire : %.2f', Salaire);
getch();
```

4 LES POINTEURS

Le pointeur fournit un autre moyen pour accéder à une variable sans se référer directement à celle ci. Plus les données deviennent complexes, pour gérer à l'aide des variables des informations dont la taille varie.

La différence entre une variable et un pointeur est que la variable pointe toujours sur le même emplacement mémore, alors que le pointeur peut être modifier de façon qu'il pointe sur un autre emplacement en mémore.

4.1 Déclaration des pointeurs

Pour créer un pointeur vers un type de données, créer une variable de ce type de données mais placer un astérisque devant son nom. La déclaration suivante:

```
Int       *p;
```

Alloue une variable p pouvant contenir l'adresse d'un entier. Maintenant, *p est l'entier dont l'adresse est contenue dans p, par exemple:

```
Int       n= 5;
          P = &n;//stocke l'adresse de n dans p
          *p + = 2;//incrémente n de 2
```

Dans l'instruction p = &n, & est l'opérateur d'adresse renvoyant de la variable n.

Considérons le programme suivant :

```
---------------------------------------------------------------------------------------------------------------------------------- *
*PROGRAMME 6
*Fichier : pointeur.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :02/02/99
*Description : manipulation des structures
----------------------------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
void main ()
{
        int x = 4

        float y = 10.0 ;

        int *px ;//pointeur sur un entier
        float *py ;//pointeur sur un réel
        clrscr() ;
        printf(''x = %d\t,y = %g\n'',x,y) ;
        px = &x ;//px contient l'adresse de x
        py = &y ;//py contient l'adresse de y
        *px = *px + 10 ;//ajoute 10 … la valeur contenue dans x
        *py = *py / 5 ;//divise par 5 le contenu de y
        printf(''x = %d\t,y = %g'' ,x,y) ;
        getch() ;
}
```

Après avoir afficher la valeur de x et de y, les adresses des variables x et y sont affectées aux pointeurs px et py.

```
Px = &x ;//px contient l'adresse de x
Py =&y ;//py contient l'adresse de y
```

En suite, l'opérateur d'instruction * est utilsé dans les instructions d'assignation

```
*px = *px +10 ;
*py = *py / 5 ;
```

Dans l'instruction *px = *py + 10, nous prenons le contenu de la variable pointée par px (x = 4), lui ajoutons 10. Le résultat est retourné dans la variable pointée par px (x = x + 10 = 14). Le contenu de la variable y est divisé par 5 dans l'instruction *py = *py / 5.

Remarques :

La déclaration suivante :

```
Int *p, q ;
```

ne déclare pas deux pointeurs p et q, mais un pointeur p et entier q. Pour déclarer p et q comme pointeur, il faut écrire

```
Int *p, *q ;
```

Ou

```
Int*p ;
Int *q ;
```

Le mot clé const interagit de façon étrange avec la déclaration des pointeurs :

```
Const  int*p ;
```

déclare un pointeur sur une constante entière, et non pas un pointeur constant sur un entier. Cela veut dire que l'on peut regarder la valeur de *p. mais que l'on ne peut pas la modifiée. Ecrire

```
*p = 0 ;//erreur modification d'une constante
```

générera une erreur alors que

```
p++ ;//p n'est pas constant don peut être modifié
```

est autorisée.

Pour déclarer un pointeur constant, il faut placer le mot clé const à côté du non du pointeur

```
Int*  const q = 10 ;
```

q ne peut pas être modifié, mais on peut modifier le contenu de la variable sur laquelle il pointe ;

```
*q = 10 ;// est correct
q++ ; //erreur
```

4.2 Tableaux et pointeurs

Les tableaux et les pointeurs sont intimement liés en C++. Le compilateur translate la notation d'un tableau en notation pointeur, puisque l'architecture interne d'un microprocesseur comprend les pointeurs mais pas les tableaux. Lorsqu' on déclare un tableau, comme par exemple :

```
int v[10] ;
```

Le nom du tableau v représente un pointeur constant pointant sur l'élément d'indice 0 du tableau. De même vecteur +1 représente l'adresse de l'élément suivant et *(v+1) l'entier stocké à cette adresse. Dans le cas général ou p, un pointeur et n, un entier on a :

```
p[n] = ù(p+n) ;
```

Si un pointeur pointe sur un tableau, on peut le faire se déplacer sur les éléments en lui ajoutant ou retranchant des valeurs entières. Ce pointeur sur un entier ajoute en fait sizeof(int) à l'adresse contenue dans le pointeur.

Pour voir la relation entre les tableaux et les pointeurs. Considérons le programme suivant . Le premier utilise la notation des tableaux et le second la notation des pointeurs.

```
* ----------------------------------------------------------------------------------------------------------------
*PROGRAMME 7
*Fichier : tab_tab.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :02/02/99
*Description : utilisation des tableaux
----------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
void main ()
{
        int Nombre[] = {10,20,30,40,50} ;
        clrscr()


        for (int i=0 ; i<5 ; ++)
        {
                Nombre[i] = Nombre[i]+5 ;
                Printf(''Nombre[%d]=%d\n'',i,Nombre[i] ;
        }
```

```
        getch()
}
```

L'expression Nombre [i] est utilisée pour accéder à chaque élément du tableau.

```
---------------------------------------------------------------------------------------------------------------------------- *
*PROGRAMME 8
*Fichier : tab_ptr.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :02/02/99
*Description : utilisation des pointeurs sur des tableaux
----------------------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
void main ()
{
        int Nombre[] = {10,20,30,40,50} ;
        clrscr() ;
        for (int i=0 ; i<5 ; i++)
        {
                *(Nombre+i) = *(Nombre +i)+5 ;
                printf(''Nombre[%d]=%d\n'',i,*(Nombre+i)) ;
        }
        getch(),
}
```

Cette version de programme est identique au précédente, sauf pour l'expression *(Nombre +i)

```
---------------------------------------------------------------------------------------------------------------------------- *
*PROGRAMME 9
*Fichier : ptr_moy.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :02/02/99
*Description : Saisir des températures par jour, puis en
                        calculer la moyenne. La saisie du nombre >=100
                        indique la fin de saisie
----------------------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
void main ()
{
        float Tempr[40],
        float Somme = 0.0, Moyenne ;
        int Nombre Jours, Jour =0 ;
        float *p ;


        clrscr() ;
```

```
        p = Tempr ; //initialise le pointeur avec l'adresse du tableau
        do
        {
                printf(''Température[%d] :'', Jour ++) ;
                scanf(''%f',p) ;
        }
        while (*(p++) <100) ; //r,p,ter tant température<100
        p= Tempr ; //initialise le pointeur avec l'adresse du tableau.
        Nombre Jours= Jour –1 ; //nombre de temp,ratures lues
        if (Nombre Jours >=1)
        {
                for(Jour = 0 ; Jour <Nombre Jours, Jour ++)
                        Somme += *(p++) ;//calcul de la somme cumule des temp,rature
                Moyenne = Somme/Nombre Jours ;
                Printf(''Moyenne des températures =%g''.Moyenne) ;
}
        getch() ;
}
```

4.3 Pointeurs et chaînes de caractères

Les chaînes de caractères sont des tableaux de caractères, terminées par le caractère nul '\0'. Les deux instructions suivantes déclarent une chaîne s, et un tampon buf capable de contenir une chaîne de 10 caractères.

```
Char s[ ] = ''Bonjour'' ;
Char buf[10] ;
```

4.3.1 Initialisation d'une chaîne comme pointeur

L'instruction

```
Char *p = ''Bonjour'' ;
```

déclare une variable de type pointeur, puis alloue 8 caractères contigus en mémoire, puis initialise p avec l'adresse de cette chaîne de caractères.

4.3.2 Traitement des chaînes de caractères

Le programme suivant calcul la longueur d'une chaîne, c'est à dire le nombre de caractères non nul.

```
---------------------------------------------------------------------------------------------------------------------- *
*PROGRAMME 10
*Fichier : Cha_ptr.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :02/02/99
*Description : manipulation des structures
----------------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
```

```
void main ()
{
        char Texte[30] ;
        char *p ;
        int Longueur ;
        clrscr() ;
        p= Texte ;
        Longueur = 0 ;
        Puts(‘’Enter un texte :’’) ;
        Gets(texte) ;
        while (*p++ !=’\0’)
        Longueur++ ;
        Printf(‘’la chaîne contient %d caractères’’, Longueur) ;
        getch() ;
}
```

Ce programme renvoie le nombre de caractères de la chaîne Texte saisie. Le pointeur p est initialisé avec p est initialisé avec l'adresse de début de la chaîne

```
P = Texte ;
```

Dans le texte de continuation de la boucle, l'expression *p++, l'opérateur est exécuté avant l'opérateur *. P++ est exécuté en premier, ayant pour effet d'incrémenter p pour passer au caractère suivant de la chaîne. Ce pendant la valeur renvoyée par p++ est la valeur du caractère se trouvant à l'adresse p avant incrémentation.

Considérons la déclaration suivante

```
Char Source [ ] = ‘’Bonjour’’ ;
Char Destination[ 10] ;
```

Peut-on écrire Destination = Source pour copier la chaîne ‘’Bonjour’’ dans Destination ?

Nous avons vu précédemment que Destination et Source qui sont écrites sans crochets, sont des pointeurs, pointant sur l'adresse de début du tableau. En particulier, Destination est une constante (le tableau est fixé en mémoire) et ne peut être modifier donc l'écriture Destination = Source est interdite

Le programme suivant utilise les pointeurs pour copier une chaîne dans une autre chaîne.

```
---------------------------------------------------------------------------------------------------------------------------------- *
*PROGRAMME 11
*Fichier : cha_copy.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :02/02/99
*Description : copie une chaîne dans une autre
----------------------------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
```

```
#include<conio.h>
void main ()
{
        char Source [] =''Chaîne source'' ;
        char Destination[81] ='''' ; //chaîne vide


        char *ps ;
        char *pd ;
        clrscr() ;
        ps= Source ;
        pd = Destination ;
        printf(''Source=%s\nDestination avant =%s\n'', Source,Destination) ;
        while ((*pd++ = *ps++) !='\0')
                                        ;//instruction vide
        printf(''Destination après%s'', Destination) ;
        getch() ;
}
```

Les pointeurs pd et ps parcourent les tableaux de caractères. L'instruction

pd++ =ps++

incrémente pd et ps, mais * s'applique aux valeurs des pointeurs avant incrémentation, et le contenu de l'adresse pointée par l'ancien ps est copié à l'adresse de l'ancien pd.

```
---------------------------------------------------------------------------------------------------------------------------------- *
*PROGRAMME 12
*Fichier : tab_ch.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :02/02/99
*Description : utilisation des tableaux de pointeurs vers des chaînes
----------------------------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
#define MAX 12
void main ()
{
        int i, drapeau=0 ;
        char Nom[10] ;
        static char * ListeNom[MAX]=
                                        {       ''Janvier'', ''Fevrier'', ''Mars'',
                                                ''Avril'', ''Mai'', ''Juin'',
                                                ''Juillet'',''Aout'',''Septembre'',
                                                ''Octobre'',''Novembre'',''Decembre''
                                        } ;
        static int List Nombre[] =
                                        {       31 30 31,
```

```
                                        30,31,30,
                                        31,31,30,
                                        31,30,31
                                    } ;
        clrscr() ;
        printf(‘’Enter le num,ro du mois :’’) ;
        scanf(‘’%d’’ ,&i) ;
        i -= 1 ;



        if ( (i>=0) && (i<=11) )
         printf(‘’%s\t%d’’, ListeNom[i],ListeNombre[i]) ;
        getch() ;
}
```

4.4 Pointeurs et structures

Lorsqu’on dépose d’un pointeur sur une structure, on a la possibilité de modifier la valeur d’un membre de cette structure. Dans le programme suivant. ARTICLE est une structure et pArt est un pointeur sur cette structure. On peut utiliser *pArt à tout endroit où on peut employer la structure.

```
---------------------------------------------------------------------------------------------------------------------------------- *
*PROGRAMME 13
*Fichier : p_struct.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :03/02/99
*Description : utilisation des pointeurs vers structure
----------------------------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
#include <string.h>
void main()
{
        typedef struct
        {
                char Designation[81] ;
                unsigned int Code ;
                float Prix ;
                float Stock ;
        }ARTICLE ;
        clrscr() ;
        ARTICLE Article ;  //variable structure
        Strcpy(Aricle.Designation, ‘’Résistance 10K 1/2w’’) ; //initialisation des membres
        Article.Code = 1234 ;  //modification des membres
        Article.Prix = 80 ;
        Article.Stock = 200 ;
        ARTICLE*pArt ;  //pointeur sur la structure
```

```
        PArt = &Article ;  //fait pointer pArt sur la structure
        (*pArt).Prix = 60 ;  //modification des membres
        (*pArt).Stock = 400 ;
        printf(''Designation   ;%s\n'', Article.Designation) ;
        printf(''Code          :%u\n'' ,ARTICLE.Code) ;
         printf(''Prix         :%2f\n'' ,ARTICLE.Prix) ;
        printf(''Stock         :%2f\n'' ,ARTICLE.Stock) ;
getch() ;
}
```

Comme l'emploi de la syntaxe

(*pointeur).membre

n'est pas pratique. C++ possède le raccourci équivalent

pointeur-> membre

Dans le programme précédent, on peut remplacer

(*pArt).Prix ;

par

pArt -> Prix ;

et

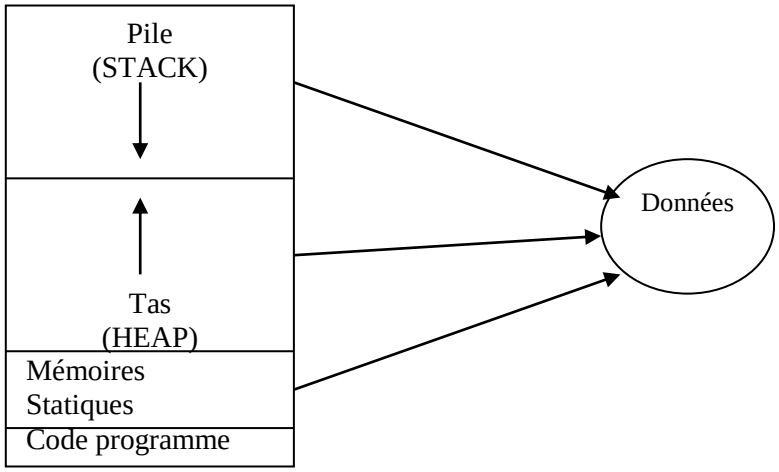
(*pArt).stock

par

Part -> Stock ;

4.5 Allocation dynamique de la mémoire

Quand on définit une variable, ordinaire ou pointeur, on alloue une certaine quantité mémoire bien définie située à une adresse équivalente bien définie (connu du compilateur). En effet, un programme C++ compilé utilise quatre régions mémoires distinctes. Leur modèle d'organisation est présenté par la figure suivante :



Organisation mémoire d'un programme

Zone de code : Elle contient les instructions machine pour toutes les fonctions.

Zone de données statiques : L'allocation statique de la mémoire est faite dès le début du programme(si la variable est globale). Ou de la fonction (si la variable est locale). La mémoire allouée ne sera libérée qu'à la fin du programme ou de la fonction.

Zone pile : Elle contient toutes les données locales non statiques, y compris les paramètres des fonctions.
Zone d'allocation dynamique : Il existe une autre technique appelée allocation dynamique de la mémoire qui consiste à utiliser la mémoire disponible, en fonction des besoins des programmes. Il faut,

Pour cela, passer par un pointeur qui pointe vers un bloc de mémoire alloué dynamiquement par l'opérateur new propre à C. Par exemple :

```
int*pEntier=new int ;
```

L'opérateur new prend une partie (ici la taille d'un entier) de la mémoire libre allouée au programme (mémoire nommée Tas).
Un pointeur, qu'il soit initialisé statiquement ou dynamiquement, peut être ensuite déréférencé :

```
*pEntier) 100 ;
```

Il faut explicitement libérer la mémoire allouée dynamiquement qu'on en 'a plus besoin, pour cela, on fait appel à l'opérateur delete :

```
Delete pEntier ;
```

Pour une structure par exemple, l'allocation dynamique se fait de la façon suivante :

```
Typedef struct
{
        char Designation[81] ;
        unsigned int code ;
        float Prix ;
        float Stock ;
} ARTICLE ;
ARTICLE *pArt ; //pointeur sur structure
PArt = new ARTICLE ; // allocation dynamique
```

5 ENUMERATION

Un type de données d'énumération permet d'obtenir des identifications mnémoniques pour un ensemble de valeurs entières. Par exemple, la déclaration suivante :

enum categorie {ingénieur, technicien, ouvrier} Catégorie ;

Les identificateurs utilisés dans une liste d'énumération sont implicitement de type signed char, unsigned char ou int, selon la valeur de ces énumérations

5.1 Initialisation des constantes d'énumération

Les énumérateurs placés entre accolades sont appelés constantes d'énumération. Chacun reçoit une valeur fixe entière. En l'absence de valeurs d'initialisation explicites, le premier énumérateur (ingénieur) est égal à zéro. Et chaque énumération qui suit est incrémenté de un (technicien 1, ouvrier2).

Lorsqu'il existe des valeurs d'initialisation, on peut attribuer d's valeurs spécifiques aux énumérateurs. Ceux qui apparaîtront ensuite sans valeur d'initialisation seront incrémentés de un . Par exemple, dans la déclaration suivante

enum categorie {ingénieur =100, technicien =ingénieur/2, ouvrier =technicien/2} Catégorie :
La valeur d'initialisation est une expression ayant pour résultat une valeur entière

5.2 Utilisation des énumérations

Les variables de type énumération sont souvent utilisées pour clarifier les opérations du programme. Si on a besoin, par exemple, d'utiliser les catégories d'un groupe d'employé dans un programme source si on utilise des valeurs telle que ingénieur, technicien ou lieu des valeurs entières. Voici un exemple d'utilisation des variables énumérations

```
---------------------------------------------------------------------------------------------------------------------------- *
*PROGRAMME 14
*Fichier : enum.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :02/02/99
*Description : utilisation des variables énumération
----------------------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
void main()
{
 enum categorie {ingenieur =0, technicien, ouvrier} ;
 struct employe
 {
                char Nom[35] ;
                float Slaire ;
                catégorie Grade ;
 } ;
```

```
employe Slaire ;
float Prime ;
clrscr() ;
printf(''Nom……..'') ;
scanf(''%s'' ,Salaire.Nom) ;
printf(''Salaire…..) ;
scanf(''%f'' ,&Salaire.Salaire) ;
printf(''Grade.(0/1/2) :'') ;
scanf(''%d'',&Salaire.Grade) ;
//Salaire.Grade = ingénieur ;
switch(Salaire.Grade)
 {
case 0 : Prime = 1000.0 ;break
 case 1 : Prime = 600.0 ;break ;
case 2 : Prime = 300.0 ;break ;
}
Salaire.Salaire + + Prime ;

Printf(''NOM…….. :%s\n'' , Salaire.Nom ) ;
Printf(''Salaire…….. :%g\n'' , Salaire.Salaire) ;
Printf(''Catégorie…….. :%u\n'' , Salaire.Grade ) ;
Return
}
```

6 DEFINITION DE NOUVEAUX TYPES

Le mot réservé typedef indique la définition d'un nouveau spécificateur de type de données. Comme exemple, considérons l'instruction suivante dans laquelle le type unsigned char est redéfini pour être de type OCTET :

```
Typedef unsigned char OCTET ;
```

Maintenant on peut déclarer des variables de type unsigned char en écrivant

```
OCTET var1, var2 ;
```

Ou lieu de

```
Unsigned var1, var2 ;
```

On peut, par exemple dans le programme 5, définir un nouveau type

```
Typedef struct categorie
{
    char Nom[35] ;
    float Salaire ;
    categorie Grade ;
```

```
}AGENT ;
```

Maintenant on peut déclarer des variables qui ont pour type AGENT

```
AGENT Salaire1, Salaire2 ;//variables de type AGENT
AGENT Salaire[50] ;//Tableau de structure de type AGENT
AGENT* Salaire ;//pointeur vers une structure
```

LES FONCTIONS

Les programmes sont souvent très complexes et très longs. Pour les créer, il est bon de les découper en sections facilement appréhendables.

Dans cette optique, C++ autorise la création de groupes d'instructions et l'attribution de nom à ces croupes. On appelle ces groupes des fonctions. Une fonction se présente comme une espèce d'opérateur personnalisé, identifiée par un nom et non par un symbole opératoire. Les opérandes d'une fonction, appelés arguments, sont spécifiés par les biais d'une liste dans laquelle les arguments sont séparés les uns des autres par les virgules.

Les déclarations des fonctions se présentent typiquement de la façon suivante.

<type> <nom de fonction> (<type> <arg1>, <type> arg2> ,…) ;

1. FONCTION NE RETOURNANT PAS DE VALEUR

1.1 Fonction ne prenant pas des paramètres

Une fonction qui ne retourne pas de valeur, donc qui se contente d'exécuter une certaine action, a le type void. Le corps de la fonction, encadré d'accolades contient les instructions exécutées par la fonction. Voici un exemple simple de fonction de type void.

```
/* FONCTION  1*/
void cadre()
{
        int X0 = 1, Y0 = 80, Y1 = 24 ;
        int i ;
        gotoxy(X0, Y0) ;printf(''\xDA'') ;
                for (i = X0+1 ; i<X1, i++)
                {gotoxy(i,Y0) ; printf(''\xC4'') ;}
        gotoxy(X1, Y0) ;printf(''\xBF'') ;
        for (i = X0+1 ; i<Y1, i++)
                {gotoxy(X1,i);printf(''\xB3'') ;}
        gotoxy(X1, Y1) ;printf(''\xD9'') ;

        for (i = X0+1 ; i<X1, i++)
                {gotoxy(i,Y1) ; printf(''\xC4'') ;}
        gotoxy(X0, Y1) ;printf(''\XC0'') ;
        for (i = Y0+1 ; i<Y1, i++)
                {gotoxy(X0,i);printf(''\xB3'') ;}
}
```

La première ligne constitue l'en-tête de la fonction, void indiquant ici que la fonction ne renvoie pas de valeur de retour.

Le type de la fonction est suivi du nom, cadre, de celle ci. Vient ensuite, la liste des arguments, encadrés de parenthèses. Ici, elle vide, ce qui signifie que la fonction n'attend pas de paramètres.

L'en-tête de la fonction et suivi du corps de celle ci, composé d'instructions spécifiant ce que doit faire la fonction, instructions encadrées par des accolades.

Cette fonction affiche un cadre.

Une fois, la fonction définie, on peut l'appeler à partir d'une autre fonction. La fonction main(), de l'exemple suivant appelle la fonction cadre(), sans lui transmettre de paramètres, pour dessiner un cadre.

```
---------------------------------------------------------------------------------------------------------------------------- *
*PROGRAMME 1
*Fichier : fct_void.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :04/02/99
*Description : utilisation des variables énumération
----------------------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
void cadre() ;//prototype de la fonction cadre() ;
void main()
{
        clrscr() ;
        cadre() ;//appelle la fonction pour dessiner uncadre
        getch() ;
}
/* ---------------------------------------------------------------------------------------------------------------------------- *
*FONCTION 1
*Description : Cette fonction dessine un cadre dont les coordonnées
                1,1 :représente le coin haut …gauche et
                80,24 : représentent le coin bas à droite.
*Entree :aucune
*Sortie :aucune
----------------------------------------------------------------------------------------------------------------------------*/
void cadre()//aucun paramètre reçu

{
        int X0= 1, Y0= 1 , X1 =80, Y1 =24 ;
        int i ;
        gotoxy(X0, Y0) ;printf(''\xDA'') ;
                for (i = X0+1 ; i<X1, i++)
                {gotoxy(i,Y0) ; printf(''\xC4'') ;}
        gotoxy(X1, Y0) ;printf(''\xBF'') ;
        for (i = X0+1 ; i<Y1, i++)
                {gotoxy(X1,i);printf(''\xB3'') ;}
        gotoxy(X1, Y1) ;printf(''\xD9'') ;
        for (i = X0+1 ; i<X1, i++)
                {gotoxy(i,Y1) ; printf(''\xC4'') ;}
        gotoxy(X0, Y1) ;printf(''\XC0'') ;
        for (i = Y0+1 ; i<Y1, i++)
```

```
                {gotoxy(X0,i);printf(''\xB3'') ;}
}
```

Pour appeler une fonction ne prenant pas de paramètres impose de laisser les parenthèses après le nom de la fonction. Un nom de fonction sans parenthèses symbolise l'adresse de début de la fonction, qui peut alors être passée à d'autres fonctions.

L'emploi des fonctions présente plusieurs avantages :

- On peut regrouper en une entité structurelle homogène des actions connexes, que l'on désignera par un simple identificateur. On améliore ainsi l'organisation et la lisibilité du code source. Lire cadre() est plus clair que de lire les instructions constituant la fonction.
- Une fonction permet de répéter une certaine séquence d'action fréquemment exécutée, sans devoir recopier tout le code source.
- Le découpage du code en fonction facilite la modularité du code source.

1.1.2 Fonction prenant des paramètres

Nous allons maintenant regarder la fonction cadre() qui reçoit des paramètres de la fonction appelante. Au lieu de dessiner un cadre fixe, la nouvelle version de la fonction cadre() à une liste d'arguments X0, Y0, X1, et Y1 de type int. Les arguments X0 et Y0 représentent le coin supérieur gauche du cadre, alors que X1, Y1 représentent le coin bas à droite du cadre.

```
/*FONCTION 2*/
void cadre(int X0,int Y0,int X1, int Y1)
{
        int i ;
        gotoxy(X0, Y0) ;printf(''\xDA'') ;
                for (i = X0+1 ; i<X1, i++)
                {gotoxy(i,Y0) ; printf(''\xC4'') ;}
        gotoxy(X1, Y0) ;printf(''\xBF'') ;
        for (i = X0+1 ; i<Y1, i++)
                {gotoxy(X1,i);printf(''\xB3'') ;}
        gotoxy(X1, Y1) ;printf(''\xD9'') ;
        for (i = X0+1 ; i<X1, i++)
                {gotoxy(i,Y1) ; printf(''\xC4'') ;}
        gotoxy(X0, Y1) ;printf(''\xC0'') ;
        for (i = Y0+1 ; i<Y1, i++)
                {gotoxy(X0,i);printf(''\xB3'') ;}
}
```

Le programme suivant dessine deux cadres en appelant deux fois la fonction cadre() avec des paramètres différents.

```
/* ------------------------------------------------------------------------------------------------------------------------
*PROGRAMME 2
*Fichier : cadvar.cpp
*Auteur :  Mustapha AIT MAHJOUB
```

```
*Date :04/02/99
*Description : dessine deux cadres concentriques en faisant appel deux fois … la fonction cadre.
-------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
void cadre(int X0,int Y0, int Y1) ;




void main()
{
        int C0,L0,C1,L1 ;
        clrscr() ;
        C0= 1 ;

        L0 = 1 ;
        C1 = 80 ;
        L1 = 24 ;
        Cadre (C0, L0,C1,L1) ;//dessine le cadre extérieur
        C0 +=5 ;
        L0 +=5 ;
        C1 - =5 ;
        L1 - =5 ;
        Cadre(C0,L0,C01,L1) ;//dessine le cadre inférieur
        getch() ;
}
/* -------------------------------------------------------------------------------------------------------------
*FONCTION 2
*Description : Cette fonction dessine un cadre dont les coordonnées
X0, Y0 :représentent le coin haut …gauche et X1, Y1 :représentent le coin bas à droite.
*Entree :Les coordonnéees X0, Y0, X1, Y1 de type entier
*Sortie :aucune
-------------------------------------------------------------------------------------------------------------*/
void cadre(int X0,int Y0,int X1, int Y1)
{
        int i ;
        gotoxy(X0, Y0) ;printf(''\xDA'') ;
                for (i = X0+1 ; i<X1, i++)
                {gotoxy(i,Y0) ; printf(''\xC4'') ;}
        gotoxy(X1, Y0) ;printf(''\xBF'') ;
        for (i = X0+1 ; i<Y1, i++)
                {gotoxy(X1,i);printf(''\xB3'') ;}
        gotoxy(X1, Y1) ;printf(''\xD9'') ;
        for (i = X0+1 ; i<X1, i++)
                {gotoxy(i,Y1) ; printf(''\xC4'') ;}
        gotoxy(X0, Y1) ;printf(''\xC0'') ;
        for (i = Y0+1 ; i<Y1, i++)
```

```
        {gotoxy(X0,i);printf(''\xB3'') ;}
}
```

2. FONCTION RETOURNAT UNE VALEUR

Une fonction, qui retourne une valeur, restitue à la fonction appelante un résultat d'un type donné. Nous allons voir maintenant une fonction munie d'une liste d'arguments et d'une valeur de retour, à savoir une fonction qui calcul la puissance. Notre fonction puissance() aura deux arguments, l'un de type double et l'autre de type int et retourne une valeur de type double.

```
/* FONCTION 3 */
double puissance(double x, int n)
{
        double r= 1 ;
        if (n= = 0) return 1 ;
        if (n< 0)
        {
                if (x= =0) return 0 ;//mathématiquement incorrect
                x = 1/x ;
                n = -n ;
        }
        while (n-- >0)
                r*= x ;
        return( r) ;
        }
```

Le mot clé return termine l'exécution de la fonction et redonne le contrôle à la fonction appelante. L'exemple suivant montre l'appel de la fonction puissance() dans une instruction d'affectation.

```
/* ----------------------------------------------------------------------------------------------------------------------
*PROGRAMME 3
*Fichier : fct_ret.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :04/02/99
*Description : calcul n2^n à l'aide de la fonction puissance()
----------------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
double puissance (double x, int n) ;
void main()
{
        int n ;
        double y, x = 2.0 ;
clrscr() ;
        n = 3 ;
        y = n* puissance (x,n) ;
```

```
        printf(''n2^n = %g'' ,y) ;
        getch() ;
}
/* ----------------------------------------------------------------------------------------------------------------------------
*FONCTION 3
*Description : calcul de la puissance par multiplication successive.
*Entrée :l'argument x de type double et l'argument n de type int
*Sortie :x puissance n
----------------------------------------------------------------------------------------------------------------------------*/
double puissance(double x, int n)
double r= 1 ;
        if (n= = 0) return 1 ;




        if (n< 0)
        {
                if (x= =0) return 0 ;//mathématiquement incorrect
                x = 1/x ;
                n = -n ;
        }
        while (n-- >0)
                r*= x ;
        return( r) ;
}
```

Le type de la valeur de retour peut être un type standaard (int, float, etc), un type dérivé (int*, float*,etc), un type personnalisé, ou bien void. Quand on ne précise pas le type de la fonction, c'est à un int qui est pris par défaut.

3. PASSAGE DE PARAMETRES

3.1 Passage par valeur

Par défaut, les paramètres sont passés par valeur. La fonction alloue une variable locale sur la pile pour chaque paramètre. Cette variable est initialisée avec la valeur passée à la fonction lors de l'appel.

Une fois initialisée, la variable se comporte comme toute autre variable locale de la fonction. Les valeurs des variables allouées pour les paramètres peuvent être modifiées (comme n-dans la fonction puissance()). Leur contenu est perdu lors de la sortie de la fonction, et elles ne sont en aucune façon recopiées dans l'environnement de la fonction appelante. Dans le programme 2.1, la fonction puissance est appelée avec comme paramètres x et n

```
Y = n*puissance(x,n) ;
```

Le n dans l'environnement d'appel n'est pas affecté, alors que le n local à la fonction est décrémentée jusqu'à 0.

3.2 Passage par adresse

Quand on veut qu'une fonction puisse modifier les paramètres, transmis, dans l'environnement appelant, il faut lui passer des pointeurs (passage par adresse) et non des variables ordinaires (passage par valeur).

```
/*FONCTION 4*/
void echange (int *x, int *y)
{
        int temp ;//variable temporaire
        temp = *x ;
        *x = *y ;
        *y = temp ;
}
```

Cette fonction devrait appelée avec échange (&x, &y). Le mécanisme de gestion de référence s'occupe de placer les opérateurs * et & à votre place.

3.3 Passage par référence

Cette technique consiste à déclarer les arguments formels en tant que référence. Passer des références à une fonction revient à lui passer les adresses des arguments, et non pas des copies de leurs contenus. Pour cette raison on utilise souvent les références pour retourner des résultats : voici la fonction puissance avec un argument de style référence pour retourner le résultat.

```
/* FONCTION 5 */
void puissance(double x, int n, double&resultat)
{
        resultat= 1 ;
        if (n= =0) return ;
        if (n <0)
        {
                if (x= =0) {
                                resultat=0 ; ;//mathématiquement incorrect
                                return
                        }
                x = 1/x ;
                n = -n ;
        }
        while (n- - > 0)
                resultat *= x ;
}
```

Quand on accède à la fonction puissance qui contient un paramètre de style référence (variable resultat), on substitue un paramètre réel dans la référence de la fonction au paramètre formel à l'intérieur de la fonction elle-même. C'est donc le paramètre réel lui-même qui sera utilisé pendant l'exécution de la procédure.

Les paramètres réels qui remplacent les paramètres de style référence doivent être eux-mêmes des variables ; ils ne peuvent pas prendre la fforme de constantes ou expressions comme pour le style de passage par valeur.

L'appel de la fonction puissance, instruction puissance(x,n,y), dans le programme suivant entraîne la substitution du paramètre réel y à l'argument formel resultat de la fonction puissance. Si la valeur de resultat change à l'intérieur de la fonction, ce qui est le cas, un changement correspondant dans la valeur de y se produira dans la fonction main()

```
/* ------------------------------------------------------------------------------------------------------------------------
*PROGRAMME 4
*Fichier : pass_ref.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :04/02/99
*Description : calcul la puissance
------------------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
void puissance(double x, int n, double&resultat) ;




void main()
{
        double y, x = 2.0 ;
        clrscr() ;
        // les variables x et n sont passées par valeur
        // alors que y est passées par référence
        puissance(x, 3, y) ;
        printf(''%g'' , y ) ;
getch() ;
}
/* ------------------------------------------------------------------------------------------------------------------------
*FONCTION 5
*Description : calcul de la puissance par multiplication successive.
*Entrée :l'argument x de type double et l'argument n de type int sont passés par valeur et l'argument
resultat est passé par référence.
*Sortie :x puissance n dans resultat
------------------------------------------------------------------------------------------------------------------------*/
void puissance(double x, int n, double& resultat)
{
                resultat =1 ;
                if (n=0) return ;
                if (n < 0)
                {
                        if  (x= =0){
                                        resultat=0 ; ;//mathématiquement incorrect
                                        return
```

```
                    }
                x= 1/x ;
                n = -n ;
        }
        while (n--> 0)
                resultat *= x ;
}
```

Au niveau de la transmission des arguments, les références combinent les avantages des pointeurs (possibilité de modifier les arguments) et ceux des variables ordinaires (simplicité de la syntaxe d'appel). Autre avantage des références (et aussi des pointeurs) : quand les arguments sont gros (structure par exemple), le passage par valeur est pénalisant car il fait recopier les arguments dans la pile (prend du temps).

4. PASSAGE D'UN TABLEAU A UNE FONCTION

Les paramètres par valeur ne peuvent jamais être modifiés par une fonction, cependant les tableaux passés enparamètres peuvent être modifiés.

La fonction suivante, par exemple, lit un vecteur

```
/* FONCTION 6 */
void Lire Tab(float t[],int n)
{


        for (int i=o ; i<n ; i++)
        {
                printf(''t[%d] :'',i)
                scanf(''%f'',&t[i] ;//le vecteur t est modifié par la fonction
        }
}
```

La fonction Lire Tab() peut lire des tableaux de n'importe quelle longueur. Le nombre d'éléments à Lire est passé explicitement dans le second paramètre.

Le tableau n'est pas copié dans la fonction avant d'être recopié lors du retour de la fonction, comme pour les variables simples, mais la fonction alloue un pointeur local initialisé avec l'adresse de début du tableau. Du fait que c'est l'adresse du tableau qui est passé, et non pas le contenu, les éléments du tableau peuvent être modifiés.

Si l'on veut protéger un tableau passé en paramètre de toute modification , par la fonction appelée, il doit être déclaré const, par exemple

```
/* FONCTION 7 */
float Produit Slaire (const t[],int n)
{
        float r= 1.0 ;
        for (int i=0 ; i<n ; i++)
                r* t[i] ;
```

```
                    return r ;
                }
```

La fonction ProduitSalaire ne pourra pas modifier t puisqu'il est déclaré constant.

```
/* ------------------------------------------------------------------------------------------------------------------------
*PROGRAMME 5
*Fichier : pass_vec.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :08/02/99
*Description : utilisation des vecteurs.
------------------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
void Lire Tab(float[],int) ;
void Efface Tab(float[],int) ;
void Affiche Tab(float[] , int) ;
float Produit Scalaire(const float[] ,int) ;
void main()
{
        float Produit, Vecteur[40] ;
        int n ;
        clrscr() ;
        printf(''Dimension des tableaux :'') ;
        scanf(''%d'' ,&n) ;
        Lire Tab(Vecteur,n) ;
        Produit =ProduitScalaire(Vecteur,n) ;



        Affiche Tab(Vecteur,n) ;
        Printf(''Produit scalaire =%g\n'' ,Produit) ;
        Getch() ;
}
/* ------------------------------------------------------------------------------------------------------------------------
*FONCTION 6
*Description : lecture des éléments d'un tableau
*Entrée :le tableau t et sa dimension n
*Sortie :valeur des élément du tableau
------------------------------------------------------------------------------------------------------------------------*/
void Lire Tab(float t[],int n)
        {
                for (int i=o ; i<n ; i++)
        {
                printf(''t[%d] :'',i)
                scanf(''%f'',&t[i] ;
        }
}
void Efface Tab(float t[],int n)
{
```

```
        int i=0 ;
        while(i<n)
                t[i++] =0 ;
}
void Affiche Tab(float t[],int n)
{
        int i=0
        while (i<n)
                printf(‘’t[%d]=%g\n’’ ,i,t[i++] ;
}
/* ----------------------------------------------------------------------------------------------------------------------------
*FONCTION 7
*Description : calcul du produit scalaire par la formule
                p = t[0]*t[1}* …t[n-1]
*Entrée : tableau t et sa dimension, le tableau t n’est pas modifié
*Retour :produit scalaire de  type float
----------------------------------------------------------------------------------------------------------------------------*/
float Produit scalaire( const float t[],int n)
{
        float r= 1.0 ;
        for (int i=0 ; i<n ; i++)
                r *= t[i] ;
        return ;
}
```

5. PASSAGE D’UNE STRUCTURE A UNE FONCTION

Les structures peuvent être passées en paramètres à des fonctions, ainsi que renvoées par des fonctions, par exemple, si on définit la structure

```
Struct Article
{
        unsigned int Code ;
        char Designation[81] ;
        float PrixUnitaire ;
        float Stock ;
} ;
```

on peut écrire des fonctions telles que

```
Article LireFiche() ; //renvoie une structure de type Article
Void AfficheFiche(Article x) ;// accepte comme paramètre une structure
```

Lorsque l'on passe une structure à une fonction, le paramètre est une variable locale de la fonction, initialisée par une copie membre à membre de la variable passée. Lorsqu'une fonction renvoie une structure par une instruction return (ou r) est une variable de type structure locale à la fonction, une copie membre à membre de r est effectuée. Ces copies membre à membre prennent du temps si la structure est importante. Il est donc préférable de passer une référence constante, par exemple

float Calcul|Montant(const Article& x, float Quantite)

du fait de const, la fonction CalculMontant () ne peut modifier la structure x. Elle reçoit un pointeur vers cette structure et non la valeur elle- même. La copie membre à membre des paramètres de la fonction ets évitée.

On peut aussi écrire une fonction pour qu'elle reçoive explicitement l'adresse de la structure :

Void ModifieStock(Article* x)

On doit l'appeler ainsi :

ModifieStock(&Comp) ;//reçoit l'adresse de la structure

```
/* ------------------------------------------------------------------------------------------------------------------------
*PROGRAMME 6
*Fichier : pass_str.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :08/02/99
*Description : utilisation des structures.
------------------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>

struct Article //définition de la structure


        unsigned int Code ;
        char Designation[81] ;
        float Prix Unitaire ;
        float Stock ;
} ;
//prototype des fonctions
Article Lire Fiche() ;
Void Afficher Fiche(Article x) ;
Float CalculMontant(const Article& x,float Quantite ) ;
Void ModifieStock(Article* x) ;

//programme principale
void main()
{
```

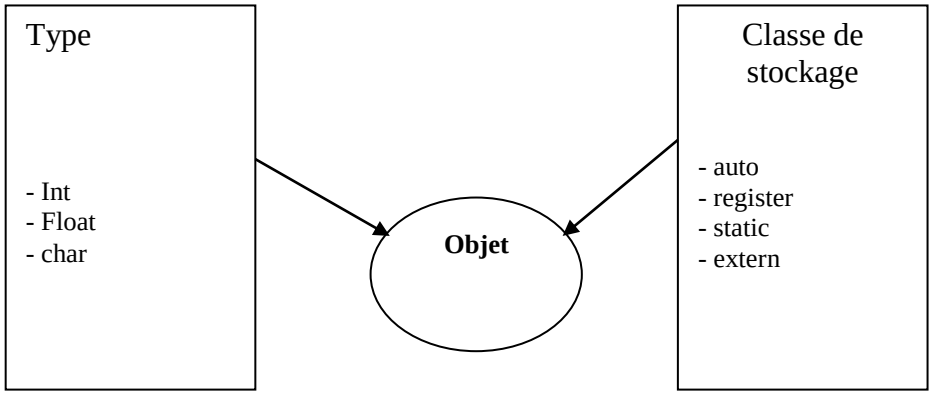
```
        Article Comp ;
        Float M,Q ;
        Comp = LireFiche() ;
Printf(‘’\Enter quantité :’’) ;
        Scanf(‘’%f’’ ,&Q) ;
        M= CalaculMontant(Comp, Q) ;
        AfficheFiche(Comp) ;
        Printf(‘’\ n ----------> Montant =%.2f’’ ,M) ;
        ModifieStock(&Comp) ;
        AfficheFiche(Comp) ;
        Getch() ;
}
/* ----------------------------------------------------------------------------------------------------------------------
*Cette fonction reçoit comme paramètre l’adresse de l ‘article pour modifier le membre Stock.
*Aucune valeur n’est retournée
----------------------------------------------------------------------------------------------------------------------*/
.void ModifieStock(Article* x)
{
        printf(‘’\nEntrer le nouveau stock :’’) ;

        scanf(‘’%f’’ ,&x->Stock).
}
/* ----------------------------------------------------------------------------------------------------------------------
*Cette fonction ne reçoit aucun paramètre et renvoie une structure Article
----------------------------------------------------------------------------------------------------------------------*/
Article Lire Fiche()
{
        Article x ;
        Clrscr() ;
        Printf(‘’Lecture d’une fiche\n’’) ;
        Printf(‘’Code……..’’) ;
        Scanf(‘’%d’’ ,&x.Code) ;
Printf(‘’Désignation…..’’) ;
        Scanf(‘’%s’’ ,x.Designation) ;
        Printf(‘’Prix unitaire…’’) ;



        Scanf(‘’%f’’ ,&x.PrixUnitaire) ;
        Printf(‘’Stock…….’’) ;
        Scanf(‘’%f’’ ,&x.Stock) ;
        Return x ;
}

/* ----------------------------------------------------------------------------------------------------------------------
*Cette fonction  reçoit comme paramètre une structure Article puis affiche ses membres
----------------------------------------------------------------------------------------------------------------------*/
void AfficheFiche(Article x)
{
```

```
        clrscr() ;
        printf(''Affichage d'une fiche\n'' ) ;
        printf(''Code……..%u\n'' , x.Code) ;
        printf(''Designation…..%s\n'' ,x.Designation) ;
        printf(''Prix unitaire…..%2f\n'',x.PrixUnitaire) ;
        printf(''Stock……%.4f'' ,x.Stock) :
}
/* ----------------------------------------------------------------------------------------------------------------------
*Cette fonction  reçoit la référence d'une structure et un réel puis renvoie le montant
----------------------------------------------------------------------------------------------------------------------*/
float CalculMontant(const Article& x,float Quantité)
{
        float m ;
        m = x.PrixUnitaire *Quantité ;
        return m ;
}
```

6. CLASSE DE STOCKAGE

L'association des identificateurs avec des objets nécessite que chaque identificateur possède au moins deux attributs : une classe de stockage et un type.

La classe de stockage détermine l'emplacement d'un objet (segment de données, registre, pile) et sa durée de vie



6.1 Durée de vie

La durée de vie, en relation étroite avec la classe de stockage, définit la période pendant la quelle les identificateurs déclarés ont des objets physiques réels affectés en mémoire.

6.1.1 Durée de vie des variables automatiques

Le spécificateur de classe de stockage auto n'est utilisé que dans des déclarations de variables à portée locale, pour une durée du vie locale (automatique). Auto est rarement employé puisque la valeur par défaut de ces déclarations est justement auto .

Le mot clé register est similaire au mot clé auto. Il demande au compilateur de maintenir l4objet dans un registre de microprocesseur s'il 'y' a un de disponible. Si aucun registre n'est disponible, le compilateur traite l'objet comme un objet automatique.

Les variables auto et register sont crées (allocation mémoire) lorsque la fonction auquel elles appartiennent est appelée, et détruites (désallocation de la mémoire lorsque la fonction est terminée. Dans la fonction suivante :

```
Void fonction1 ()
{
        int a ; //automatique par défaut
        auto int b ; //automatique
        register int c ;
}
```

Les variables a, b et c sont crées à l'appel de fonction 1 () et d étruites lorsque le contrôle est redu à la fonction appelante.

6.1.2 Durée de vie des variables statiques et externes

Le spécificateur de classe de stockage static est utilisable avec les déclarations de fonctions et de variables à portée locale et à portée fichier. Static signale aussi que la variable a une durée de vie statique. Dans la fonction suivante :

```
Void fonction 2()
{
        static float delta ;
}
```

La variable static delta est connue seulement dans la fonction ou' elle est définie, fonction2 (), mais, contrairement aux variables automatiques, elle ne disparaît pas lorsque la fonction est terminée.

Le spécificateur externe est utilisable avec les déclarations de fonctions et de variables à portée locale et à portée fichier, pour indiquer un lien externe. Avec les variables à portée fichier, le spécificateur par défaut est externe. Utilisé avec des variables, externe indique que la variable a une durée de vie static

6.2 Portée des variables (visibilité)

Chaque variable possède une certaine portée bien définie, domaine de validité au sein duquel on peut accéder à la variable. Si on essaye d'accéder à une variable hors de sa portée, le compilateur ne l'accepte pas.

6.2.1 Portée Bloc

La portée bloc, ou portée local, d'un identificateur commence au point de déclaration et se termine à la fin du bloc contenant la déclaration. Voici un exemple de bloc à l'intérieur d'une fonction :

```
Void main()
{//bloc 1
        int x= 10 ;

        {//bloc 2
                int y= 20 ;
                printf(‘’ %d  %d\n’’ , x, y) ;//coorect
                        }
        y = y+ 10 ; //erreur
}
```

L'instruction y = y+ 10 générera un message d'erreur puisque la variable y est définie dans le bloc 2, donc ne peut pas être utilisée dans le bloc 1.

6.2.2 Portée fonction

Les seuls identicateurs qui ont une portée de fonction sont les labels d'instructions.

```
Void fonction()
{
        int x ;
```

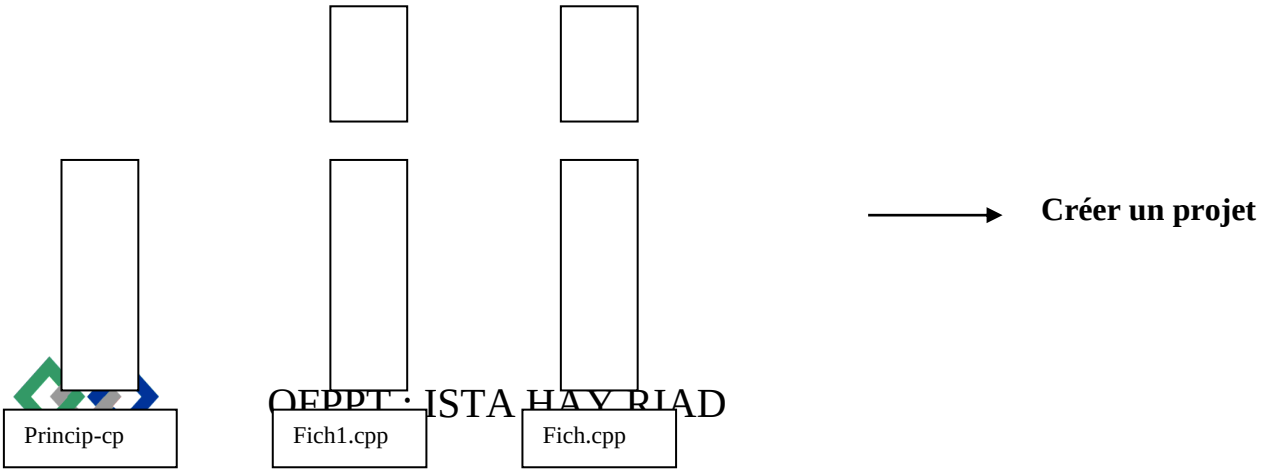
```
while(1)
        {
        scan(‘’ %d’’ ,&x) ;


        if (x <0)
                goto fin ;
        }
fin : printf(‘’le label fin est accesssible dans toute la fonction’’) ;
}
```

6.2.3 Portée fichier

Les identificateurs ayant une portée fichier, appel és identificateurs globaux, sont déclarés en dehors des blocs. Leur portée part du point de leur déclaration et se termine à la fin du fichier source. Voici un exemple :

```
int X ;
Void main()
{
        x = x+ 20 ;//correct, x est déjà définie
        y = y +30 ;//errreur, y n’est pas encore définie
}
int y ;
void fonct()
{
        x = x+20 ;//correct
        y = y+ 30 ;//correct
}
```

La variable x est visible aux fonctions main() et fonct(), alors que y n’est visible qu’à la fonction fonct().



Les fichiers

LES FICHIERS

1. DEFINITION

Un fichier est une structure de donnée. Tout comme un tableau, un fichier est constitué d'une collection de données de même type. A la différence d'un tableau, il est possible d'emmagasiner un fichier sur un disque pour conserver l'information d'une façon permanente.

2. PRINCIPE D'ACCES AUX FICHIERS SUR DISQUE

L'accès au disque est une procédure complexes et relativement long puisque le lecteur du disque contient une parie méxanique. L'envoie des données caractère par caractère est impraticables. Le système d'exploitation qui s'occupe des transferts utilise une mémoire tampon où les données a enregistré sur le disque seront stockées temporairement en attendant que le tampon soit plein ou lorsqu'on ferme un fichier.

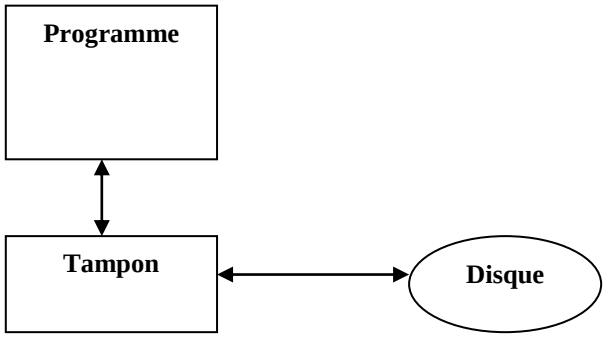


Figure 1 : mécanisme de transfert

La gestion de la mémoire tampon implique plusieurs taches :

- Resever l'espace mémoire dans la RAM
- Gérer cette mémoire à l'aide d'un pointeur sur celle ci
- Libérer cette mémoire lorsque l'on n'a pas besoin d'elle.

Le langage C nous fournit deus alternatives : soit gérer le tampon nous même en utilisant la méthode d'entrée/sortie de bas niveau (ou système) soit confier la tache au système d'exploitation en utilisant la méthode d'entrée sortie standard.

Chacune des deux méthodes est plus au moins complète pour effectuer des E/S sur disque. Ils ont toutes les deux des fonctions pour lire et écrire un fichier ou réaliser d'autres taches. Mais il y'a des différences importantes entre les deux méthodes.

La méthode d' E/S standard comme son nom l'indique, et la méthode la plus simple et utilisée pour effectuer des E/S dans un programme C. Il possède une large gamme de commande assez facile à utiliser et évite au programmeur les détails des opérations élémentaires d' E/S tel que la gestion du tampon et la conversion automatique des données.

La méthode d'E/S de bas niveau fournit moins de choix que la méthode standard, il peut être considérer comme un système primitif. Les techniques utilisés sont proches de celle du système d'exploitation. Le programmmeur doit créer et gérer le tampon utilisé pour le transfert des données. La méthode d'E/S système et plus difficile à utiliser que la méthode standard mais plus efficace en terme de vitesse et d'occupation mémoire.

3. ENTREE/SORTIE STANDARD

La méthode d'E/S standard est utilisée de quatre façons pour lire et écrire des données. Trois parmi ses quatres façons pour transférer les données correspondent aux méthodes, déjà vues, pour lire les données à partir du clavier et leur affichage sur l'écran.

Premièrement, les données peuvent être lues ou écrites un caractère à la fois. Ceci est analogue au travail réalisé par la fonction getche() pour lire un caractère à partir du clavier et à putchar() pour afficher un caractère

Deuxièmement, les données peuvent être lues ou écrites sous forme d'une chaîne, comme pour les fonctions gets() et puts().

Troisièmement, des données peuvent être lues et écrites dans un format donné analogue à celui généré par les fonctions printf() et scan() : comme une collection de valeurs mixtes de type caractère, chaîne, réel ou entier.

Quatrièmement, les données peuvent être lues et écrites dans un nouveau format appelé enregistrement ou bloc. L'enregistrement est un groupe de données de longueur fixe et généralement utilisé, par exemple, poue stocker les tableaux et las structures.

Le tableau suivant montre les fonctions utilisées pour lire et écrire des données dans un fichier pour la méthode d'E/S système et la méthode standard avec ses quatre possibilités.

	Caractère	Chaîne	Formaté	Enregistrement
E/S standard	getc() putc()	fgets() fputs()	fscan() fprint()	fread() fwrite()
E/S système				read() write()

3.1 E/S des caractères

Notre premier exemple prend les caractères tapés au clavier, et un à la fois, sont envoyés sur le disque (tampon). Le programme est le suivant et essayant de voir ce qu'il fait.

```
/* ----------------------------------------------------------------------------------------------------
*PROGRAMME 1
```

```
*Fichier : ecri_car.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :11/02/99
*Description : écrit des caractères tapés au clavier, vers un fichier
-----------------------------------------------------------------------------------------------------------------*/


#include<stdio.h>
#include<conio.h>
void main()
{
        FILE *fptr ;//pointeur vers fichier
        char car ;
        char fichier[81] ;
        puts(''Nom du fichier :'') ;
        gets(fichier) ;
        fptr =fopen(fichier, ''w'') ;//ouvrir fichier en écriture
        while ( (car=getche()) != '\r')//saisie des caractères
                Putc(car,fptr) ;//ecrire caractère vers fichier
        fclose(fptr) ; //fermer fichier
}
```

Lors de l'exécution du programme, celui ci demande à l'utilisateur de taper le chemin et le nom du fichier (par exemple C: \TEXTE.txt) puis attend qu'une ligne de texte soit saisie (par exemple, ENTREE SORTIE DES CARACTERS C'EST FACILE.). Le programme s'arrête lorsque la touche [ENTREE] est appuyée.

On peut utiliser la commande TYPE du DOS pour voir le contenu du fichier TEXTE.TXT.

```
C:\> TYPE TEXTE.TXT
ENTREE SORTIE DES CARACTERES C'EST FACILE
```

3.1.1 Ouverture d'un fichier

Avant d'écrire un fichier vers un disque, ou le lire, on doit l'ouvrir. L'ouverture d'un fichier établit une communication entre le programme et le système d'exploitation pour préciser le fichier auquel on veut accéder et comment on va l'utiliser. On doit fournir, au système d'exploitation, le nom du fichier et d'autres informations (lire, écrire, tec…). La communication entre le système d 'exploitation et le programme se fait à travers une structure C qui contient les informations concernant le fichier .

Cette structure, qui est définie par struct FILE dans le fichier stdio.h. est le point de contact . Lorsque on demande d'ouvrir un fichier, on reçoit au retour, c'est la demande est satisfaite, un pointeur vers une structure d type File. Chaque fichier ouvert doit avoir sa propre structure FILE, avec un pointeur vers elle. La figure 2 montre ce processus

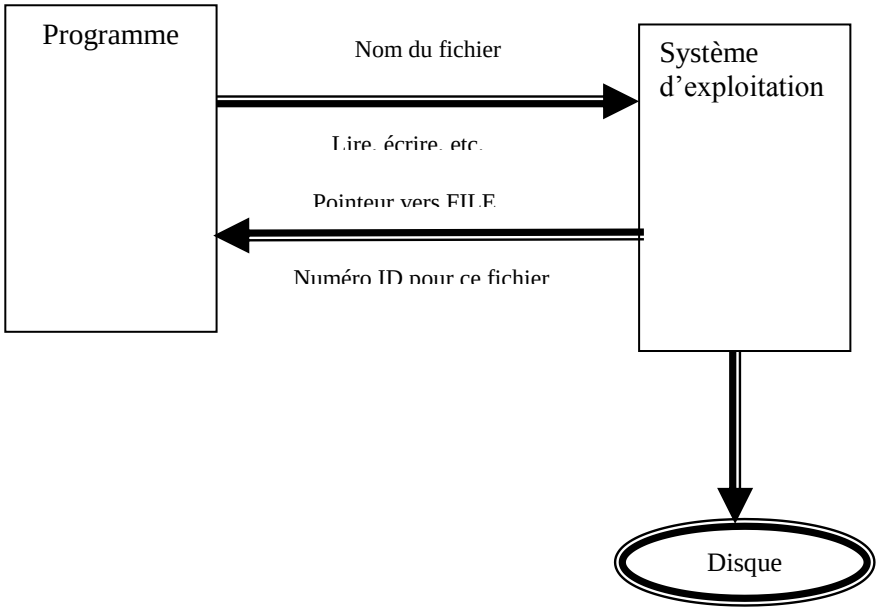


Figure 2 : Ouverture d'un fichier

La structure FILE contient les informations sur le fichier qui vient d'être ouvert, comme exemple sa taille courante et l'adresse de son tampon. Cette structure est définie dans stdio.h de la façon suivante :

```
Typedef struct {
        Short           level ;
        Unsigned                flags ;
        Char            fd ;
        Unsigned char  hold ;
        Short           bsize ;
        Unsigned char *buffer, *curp ;
        Unsigned        istemp ;
        Short    token ;
} FILE ;
```

En examinant le détail de cette structure, on remarque l'existence de deux pointeurs : buffer et curp. Le rôle de buffer est de contenir l'adresse de la première case de la mémoire tampon. Celui de curp est de se déplacer dans la mémoire tampon pour lire ou écrire les caractères un par un. Un autre membre de cette structure est bsize qui contient la taille du tampon qui est de 512 octets.

Dans le programme 1, nous avons déclaré, en premier, une variable de type pointeur vers FILE, dans l'instruction :

```
FILE *fptr ;
```

Puis, on a ouvert le fichier avec l'instruction

Fptr =fopen(fichier, ‘’w’’) ;

Cette instruction demande au système d’exploitation d’ouvrir un fichier dont le nom est saisi par l’instruction gets(fichier) ;

Le caractère ‘’w’’ indique qu’on désire écrire vers le fichier. La fonction fopen() retourne un pointeur vers la structure FILE, pour le fichier, et qu’on stocke dans la variable fptr.

Le prototype de la fonction fopen() est déclaré dans stdio.h comme suit :

FILE *fopen(const char *filename, const char *mode) ;

La chaîne de mode prend l’une des valeurs ci-dessous :

Chaîne de mode	Desvription
‘’r’’	Ouverture en lecture seule
‘’w’’	Création pour écriture
‘’a’’	Ajout :ouvert en écriture en fin de fichier ou création pour écriture si inexistant
‘’r+’’	Ouverture de fichier préexistant pour mise à jour (lecture/écriture).
‘’w+’’	Création pour mise à jour (lecture/écriture). Si un tel fichier préexiste, le contenu est perdu
‘’a+’’	Ouverture pour ajout ; ouverture pour mise à jour en fin de fichier ou création si inexistant.

Pour indiquer qu’un fichier doit être ouvert ou crée en mode texte, on ajoute t à la chaîne de mode (par exemple, ‘’rt , ‘’w + t’’).

Pour le mode binaire, on ajoute b (par exemple, ‘’wb’’ , ‘’a +b’’).

3.1.2 Ecriture vers un fichier

Une fois, on a établit une ligne de communication avec le fichier en l’ouvrant, on peut écrire dans ce fichier. Dans le programme 1 nous écrivons les caractères, un à la fois, en utilisant l’instruction :

Putc(car, fptr) ;

Cette instruction écrit le caractère car dans le fichier pointé par la variable fptr.

Le processus d’écriture, vers le fichier, continue dans la boucle while jusqu’à ce que l’utilisateur tape la touche ENTREE.

3.1.3 Fermer un fichier

Lorsqu'on a terminé l'écriture vers le fichier, on doit fermer celui ci. La fermeture du fichier se fait par l'instruction

fclose(fptr) ;

Normalement, les caractères envoyés par putc() sont stockés temporairement dans le tampon. Lorsque le tampon est plein (512 octets), sont contenu est alors envoyé automatiquement au disque. La fermeture du fichier par la fonction fclose() fait en sorte que les caractères du tampon sont envoyés vers le disque même si celui ci n'est pas plein.

Une autre raison pour fermer le fichier est de libérer l'espace de communication utilisé par un fichier pour être utiliser par d'autres fichiers. Cet espace contient la structure FILE et le tampon lui-même.

3.1.4 Lecture d'un fichier

Si on peut écrire un fichier, on doit être capable de le lire. Voici un programme qui lit un fichier en utilisant la fonction getc() :

```
/* ------------------------------------------------------------------------------------------------------------
*PROGRAMME 2
*Fichier : lire_car.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :11/02/99
*Description : lit des caractères d'un fichier puis les affiche
------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
void main()
{
        FILE* fptr ;//pointeur vers fichier
        int car ;
        char fichier[81] ;
        puts(''Nom du fichier :'') ;
        gets(fichier) ;
        fptr =fopen(fichier,''r'') ;//ouvrir fichier en lecture
        while ( (car=getc(fptr)) !=EOF)//lecture des caract7res du fichier
                printf(''%c'' ,car):://affichage du caractère lu
        fclose(fptr) ; //fermer fichier
}
```

Ce programme est similaire au programme 1. Le pointeur vers FILE est déclaré de la même façon, et la fichier est ouvert et fermé de la même façon. La fonction getc() lit un caractère à partir du fichier (par exemple , TEXTE.TXT) ; cette fonction est complémentaire à la fonction putc().

3.1.5 Fin de fichier

La différence majeur entre ce programme et PROGRAMME 1 est que PROGRAMME 2 doit savoir la consultation de l'indicateur de fin de fichier EOF qui est envoyé par le système d'exploitation. L'indicateur EOF n'est pas un caractère mais un entier de valeur 1.

3.1.6 Erreur d'ouverture d'un fichier

Dans le programme 1 ou 2 si le fichier spécifié dans la fonction fopen() n'a pas pu être ouvert , le programme ne pourra pas être exécuté. En effet, la fonction fopen() renvoie une valeur nulle si par exemple :

- Le disque est plein pour écriture
- Fichier inexistant
- Disque ou fichier protégé contre l'écriture.

Le programme suivant est une version améliorée du programme2.

```
/* ---------------------------------------------------------------------------------------------------------------
*PROGRAMME 3
*Fichier : lir_car2.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :11/02/99
*Description : lit des caractères d'un fichier puis les affiche s'il n'pas d'erreurs d'ouverture de fichier.
---------------------------------------------------------------------------------------------------------------*/
void AfficheErreur (FILE*fptr, char* ptr) ;
#include<stdio.h>
#include<conio.h>
#include<stdib.h>
void main()
{
        FILE* fptr ;//pointeur vers fichier
        int car ;
        char fichier[81] ;
        puts(''Nom du fichier :'') ;
        gets(fichier) ;
        fptr =fopen(fichier,''r'') ;//ouvrir fichier en lecture
        if (fptr = = NULL)
        {
printf(''Impossible d'ouvrir le fichier %s en lecture\n'' , fichier) ;
                exit(1) ; //sortie du programme
        }
        while ( (car =getc(fptr)) ! = EOF)//lecture des caractères du fichier
        {
                if (ferror(fptr))
```

```
                {

                        AfficheErreur(fptr,fichier) ;
                        Fclose(fptr) ;
                        Exit(1) ;
                }

                        printf(''%c'' ,car) ;//affichage du caractère lu
                }
                fclose(fptr) ; //fermer fichier
        }


        void AfficheErreur(FILE* fptr, char* ptr)
        {

                perror(ptr) ;
                clearerr(fptr) ;
}
```

Même si le fichier a été ouvert correctement, il se peut que d'autres erreurs de lecture ou d'écriture surviennent lors d'un accès au disque. Un secteur pourrait être défectueux, par exemple. La fonction ferror() détecte ce genre d'erreurs, perror() affiche un message correspondant à cette erreur et clearer() remet à zéro un indicateur d'erreur pour permettre de continuer les opérations sur le fichier.

3.1.7 Application : Compter le nombre de caractères dans un fichier

La possibilité de lire et d'écrire ves un fichier caractère par caractère peut avoir beaucoup d'applications. Comme exemple, voici un programme qui compte le nombre de caractères dans un fichier :

```
/* ------------------------------------------------------------------------------------------------------------
*PROGRAMME 4
*Fichier : compteC.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :12/02/99
*Description : compte le nombre de caract7res dans un fichier
------------------------------------------------------------------------------------------------------------*/

#include<stdio.h>
#include<conio.h>
#include<stdib.h>
#include<dos.h>
void main(int_argc,char*_argv[]))
{
        FILE* fptr ;//pointeur vers fichier
        int car ;
```

```
        char fichier[81] ;
        int compte=0 ;
        clrsc() ;
        if (_argc != 2)
        {
                printf(''Format : >comptec NomFichier'') ;
                exit(0) ;
                }
                fptr =fopen(_argv[1],''r'') ;//ouvrir fichier en lecture
                if (fptr = = NULL)
                {
                        printf(''Fichier %s ne peut pas être ouvert\n''_argv[1] ;
                        exit(0) ;//fin du programme
                }


                while ( (getc(fptr) ) != EOF)//lire caractère du fichier
                        compte++ ;
                fclose(fptr) ; //ferme fichier
                printf(''le fichier %s contient %d caractères.'',_argv[1],compte) ;
        getch()
}
```

Dans de programme nous avons utilisés une ligne de commande avec arguments pour obtenir le nom du fichier. On commence par chaercher le nombre d'arguments ; s'il y'en a deux, alors le deuxième, arg[1], est le nom du fichier.

Le programme ouvre le fichier en lecture tout en s'assurant de son existence, puis entre dans une boucle de lecture, pour lire les caractères un à la fois. A chaque lecture d'un caractère la variable compte est incrémentée de 1.

Le programme compte peut être exécuté à partir du DOS, par exemple :

```
C :>compte TEXTEE.txt
```

3.2 E/S des chaînes

Lire et écrire des chaînes de caractères à partir ou vers un fichier est aussi facile que lire et écrire des caractères individuels. Voici un programme qui écrit des chaînes vers un fichier en utilisant la fonction d'E/S de chaîne fputs().

```
/* ----------------------------------------------------------------------------------------------------------
*PROGRAMME 5
*Fichier : ecri_ch.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :05/03/99
*Description : écrit des chaînes de caractères, tapées au clavier vers un fichier
----------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
```

```
#include<conio.h>
#include<string.h>
#include<process.h>
void main()
{
        FILE* fptr ;//pointeur vers fichier
        char car ;
        char fichier[81] ;
        char chaîne[81] ;
        puts(''Nom du fichier :'') ;
        gets(fichier) ;
        fptr =fopen(fichier,''w'') ;//ouvrir fichier en ecriture
        if (fptr = = NULL)
        {
                printf(''Fichier %s ne peut pas être ouvert\n'' , fichier) ;
                exit(0) ;//fin du programme
        }


        while ( strlen(gets(chaîne)) > 0)
        {
                fputs(chaine,fptr) ;
                fputs(''\n'' ,fptr) ;
        }
        fclose(fptr) ; //ferme fichier
}
```

L'utilisateur tape une série de chaînes en terminant chacune d'elles par[ENTREE]. Pour terminer le programme, l'utilisateur tape la touche [ENTREE] au début d4une ligne pour créer une chaîne de longueur nulle (condition de sortie de la boucle).

Nous avons déclaré un tableau de caractères pour stocker une chaîne fputs() écrit le contenu de ce tableau vers le disque, mais elle n'ajoute pas automatiquement le caractère nouvelle ligne \n à la fin de la chaîne, on doit le faire explicitement (fputs('\n' ,fptr) pour rendre facile la lecture des chaînes à partir du fichier.

Voici le programme qui lit des chaînes à partir d'un fichier :

```
/* ----------------------------------------------------------------------------------------------------------
*PROGRAMME 6
*Fichier : lit_ch.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :05/03/99
*Description : lit des chaînes de caractères, … partir d'un fichier puis les affiche
----------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
```

```
#include<string.h>
#include<process.h>
void main()
{
        FILE* fptr ;//pointeur vers fichier
        char car ;
        char fichier[81] ;
        char chaîne[81] ;
        puts(‘’Nom du fichier :’’) ;
        gets(fichier) ;
        fptr =fopen(fichier,’’w’’) ;//ouvrir fichier en ecriture
        if (fptr = = NULL)
        {
                printf(‘’Fichier %s ne peut pas être ouvert\n’’ , fichier) ;
                exit(0) ;//fin du programme
        }
        while (fgets(chaine,80,fptr) != NULL)
                printf(‘’%s’’,chaine) ;
        fclose(fptr) ; //ferme fichier
}
```

La fonction fgets() est déclarée dans le stdio.h par :

```
Char *fgets(char *s, int n, FILE *stream)
```

Nous remarquons que cette fonction prends trois paramètres. Le premier est l'adresse où la chaîne est stockée, le second est la longueur maximale de la chaîne. La fonction cesse la lecture soit lorsque n-1 caractères ont été lus soit suite à la lecture de saut de ligne (‘\n’) (ce caractère est copié dans s). la fin de la chaîne de caractères est marquée par l'ajout d'un caractère nul (‘\0’). Le troisième est le pointeur sur la structure FILE pour ce fichier.

3.3 E/S Formatées

Les programmes précédents nous ont permet de lire et écrire des caractères et du texte. Nous allons voir maintenant comment manipuler les nombres. Le programme 7 lit les données à partir du clavier, puis les envoie vers un fichier.

```
/* ----------------------------------------------------------------------------------------------------
*PROGRAMME 7
*Fichier : ecrifo_ch.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :05/03/99
*Description : écrit des données formatées, tapées au clavier vers un fichier
----------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<process.h>
```

```
void main()
{
        FILE* fptr ;//pointeur vers fichier
        char car ;
        char fichier[81] ;
        char Designation[41] ;
        int Code ;
        float Prix ;
        float Stock ;
        clrscr() ;
        puts(''Nom du fichier :'') ;
        gets(fichier) ;
        fptr =fopen(fichier,''w'') ;//ouvrir fichier en ecriture
        if (fptr = = NULL)
        {
                printf(''Fichier %s ne peut pas être ouvert\n'' , fichier) ;
                exit(0) ;//fin du programme
        }
        do
        {
                printf (''Code :'') ;
                scanf(''%d'' ,&Code) ;
                if (Code < = 0) break ;
                printf(''Désignation :'') ;
                scanf(''%s'' ,Designation) ;

                printf(''Prix Unitaire :'') ;
                scanf(''%f'' ,&Prix) ;
                printf(''Stock :'') ;
                scanf(''%f'' ,&Stock) ;
                fprintf (fptr, ''%s %d %f %f' , Designation, Code, Prix, Stock) ;
        }
        while ( 1) ;

        fclose(fptr) ; //ferme fichier
}
```

La clé de ce programme est la fonction fprintf(), qui écrit les valeurs des quatre variables vers le fichier . Cette fonction est similaire à printf(), sauf que le pointeur FILE est inclus comme premier paramètre. Comme pour printf(), nous pouvons formater une donnée de plusieurs façon ; en d'autre terme, les conventions d'écriture avec printf() reste valable pour fprintf()

Dans la boucle do, l'utilisateur saisie les données Code, Désignation, Prix et Stock tant que code est supérieur à 0.

Le programme suivant permet de lire les données enregistrées avec la programme 7.

```
/* ----------------------------------------------------------------------------------------------------------
*PROGRAMME 8
```

```
*Fichier : litfo_ch.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :05/03/99
*Description : lecturedes données formatées, à partir du disque, et leurs affichage
-----------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<process.h>
void main()
{
        FILE* fptr ;//pointeur vers fichier
        char car ;
        char fichier[81] ;
        char Designation[41] ;
        int Code ;
        float Prix ;
        float Stock ;

        clrscr() ;
        puts(''Nom du fichier :'') ;
        gets(fichier) ;
        fptr =fopen(fichier,''w'') ;//ouvrir fichier en lecture
        if (fptr = = NULL)
        {
                printf(''Fichier %s ne peut pas être ouvert\n'' , fichier) ;
                exit(0) ;//fin du programme
        }


        while (fscanf(fptr, ''%s %d  %f  %f'' , Designation,&Code,&Prix,&Stock) !=EOF)
        {
                printf (''Code :%d\n'',Code) ;
                printf(''Désignation : %s\n'' ,Designation) ;
                printf(''Prix Unitaire : %.2f\n'' Prix) ;
                printf(''Stock : %2f\n'' ,Stock) ;) ;
        }
        fclose(fptr) ; //ferme fichier
        getch() ;
}
```

Le programme 8 utilise la fonction fscan() pour lire les données à partir d'un fichier. Cette fonction est similaire à scanf(), sauf que, comme pour fprintf() , le pointeur vers FILE est inclus comme premier paramètre.

Le texte et les caractères sont stockés un caractère par octet. Les entiers sont stockés, en mémoire, sous forme de deux octets et les réels sur quatre octets alors que sur le disque, ils sont enregistrés sous forme d'un texte. Ainsi, l'entier 999 est stocké en mémoire sur deux octets, mais il a

besoin de trois octets pour être stocké sur disque sous la forme ‘’999’’. Le nombre réel 45.36 occupe 4 octets en mémoire et 5 octets sur disque ; un octet pour cahaque chiffre et un octet pour le point décimal.

3.4 Mode binaire et mode texte

Le langage C a été développé initialement pour le système d’exploitation UNIX , en utilisant évidemment les conventions de traitement de fichier correspondantes à UNIX. Lorsque le langage C a été transporté sur MSDOS , on a ddécidé que C suppoterait les conventions des deux systèmes d’exploitation. Ainsi le mode texte est conforme au système UNIX et le mode binaire, au système MSDOS.

La distinction entre les deux modes réside dans l’interprétation du caractère ‘\n’ de code x0A (ligne suivante) et du caractère ctrl-Z, de code x/A

La figure suivante décrit l’interaction entre un programme C et le DOS pour un fichier ouvert en mode texte.

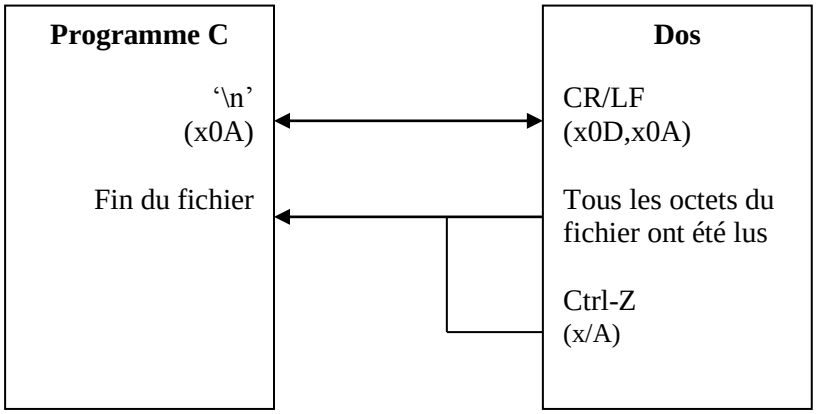


Figure 3 : Interaction C et DOS en mode texte

Pour un fichier ouvert en mode texte et en écriture, le caractère ‘n’ est converti et sauvé par le DOS sous forme de deux octets : CR/LF (x0D/x0A). lors d’une opération de lecture, le DOS transmet au programme C le code x0A lorsqu’il rencontre la séquence x0D et x0A

Normalement, la fin du fichier est reconnue lorsque tous les octets lus. Le dos connaît le nombre d’octets de chaque fichier.

Si le caractère ctrl-Z se trouve dans le fichier ouvert en mode texte, le DOS l’interprète comme fin de fichier. Le reste du fichier sera ignoré.

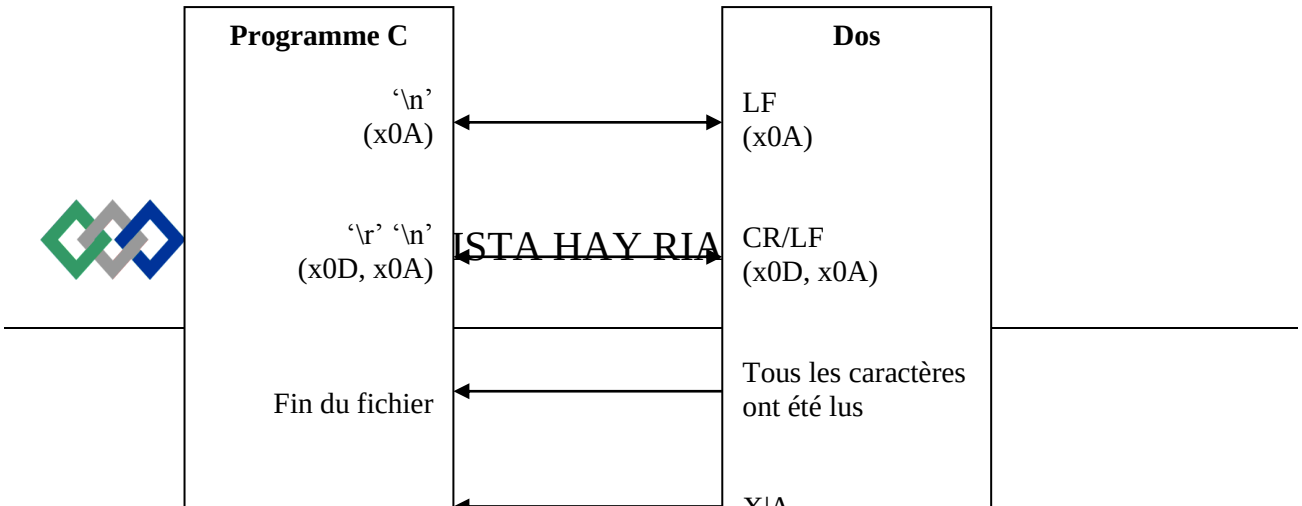


Figure 5 : Interaction Cet DOS en mode binaire

Pour un fichier ouvert en mode binaire :

- Les caractères ‘\n’ et ‘\r’ sont enregistrés et relus tels quels par le DOS
- La fin du fichier est reconnue lorsque tous les octets ont été lus.
- Le code ctrl-Z est un code binaire comme les autres. Il n’est donc pas interprété par le DOS comme fin de fichier.

3.5 E/S des enregistrements

Nous avons vu, précédemment, comment les nombres peuvent être stockés en utilisant les fonctions d’E/S formatées fscan() et fprintf(). Nous avons aussi vu que la sauvegarde des nombres occupe beaucoup d’espace disque, car chaque chiffre est stocké comme caractère. L’E/S des données formatées présente un autre problème : il n’y a pas de moyen direct pour stocker des données de type complexe tels que les tableaux et les structures. La solution à ces problèmes est l’es des enregistrements (ou block). L’E/S des enregistrements écrit, vers le disque, les nombres dans un format binaire ce que réduit l’espace occupé sur le disque.

3.5.1 Ecriture des structures avec fwrite()

Nous allons récrire l’exemple du programme 7 en utilisant la structure ARTICLE. Le programme suivant accepte les entrées concernant, le nombre de la structure ARTICLE, Désignation, Code, Prix et stock puis écrit ces données vers le fichier disque.

Le fichier est ouvert en mode binaire et les informations obtenues de l’utilisateur sont placées dans la structureart. Puis, l’instruction suivante écrit la structure vers le fichier :

Fwrite(&art, sizof(art), 1, fptr ;

Le premier argument de cette fonction est l’adresse de la structure qui va être écrite. Le second argument est la taille de la structure en nombre d’octets. Le troisième argument est le nombre de structure à écrire en même temps. Le dernier argument est le pointeur vers le fichier dans lequel en va écrire.

```
/* ------------------------------------------------------------------------------------------------------------
*PROGRAMME 9
```

```
*Fichier : ecristru_ch.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :09/03/99

*Description : écrit des structures vers un fichier
-------------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<process.h>
typedef struct
        {
                char Designation[41] ;
                int Code ;
                float Prix ;
                float Stock ;
        }ARTICLE ;
void main()
{
        FILE * fptr ;//pointeur vers fichier
        Char  car ;
        Char fichier[81] ;
        ARTICLE art ;
        clrscr() ;
        puts(''Nom du fichier :'') ;
        gets(fichier) ;
        fptr =fopen(fichier,''wb'') ;//ouvrir fichier en écriture,mode binaire
        if (fptr = = NULL)
        {
                printf('Impossible d'ouvrir le fichier %s en écriture\n'' ,fichier) ;) ;
                exit(1) ;//fin du programme
}
do
{
                printf (''Code :'') ;
                scanf(''%d'' ,&art.code) ;
                if (aer.Code <0) break ;
                printf(''Désignation : '') ;
                scanf(''%s'',art.Designation) ;


                printf(''Prix Unitaire :'') ;
                scanf(''%f'' ,&art.Prix) ;
                printf(''Stock : '') ;
                scanf(''%f'' ,&art.Stock) ;

                fwrite (&art,sizeof(art),1,fptr) ;
        }
```

```
        while(1) ;

        fclose(fptr) ; //ferme fichier
}
```

3.5.2 Lecture des structures avec fread()

Le programme suivant permet de lire la structure écrite avec la programme 9 :

```
/* ----------------------------------------------------------------------------------------------------------------
*PROGRAMME 10
*Fichier : litstru_ch.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :09/03/99
*Description : lit des enregistrements à partir d'un fichier
----------------------------------------------------------------------------------------------------------------*/

#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<process.h>
typedef struct
        {
                char Designation[41] ;
                int Code ;
                float Prix ;
                float Stock ;
        }ARTICLE ;
void main()
{
                FILE * fptr ;//pointeur vers fichier
                char  car ;
                char fichier[81] ;
                ARTICLE art ;
                clrscr() ;
                puts(''Nom du fichier :'') ;
                gets(fichier) ;
                fptr =fopen(fichier,''rb'') ;//ouvrir fichier en lecture,mode binaire
        if (fptr = = NULL)
        {
                printf('Impossible d'ouvrir le fichier %s en lecture\n'',fichier) ;) ;
                exit(0) ;//fin du programme
        }



        while (fread(&art,sizeof(art),fptr) = =1=
        {
                printf (''Code :%d\n'' ,art.Code) ;
```

```
                printf(''Désignation : %s\n'' ,art.Designatio) ;
                printf(''Prix Unitaire :%.2f\n'' ,art.Prix) ;
                printf(''Stock : %.2f\n'' ,art.Stock) ;
        }
        fclose(fptr) ; //ferme fichier
        getch() ;
}
```

La principale instruction du programme 10 est l'expression

```
Fread(&art, sizeof(art), 1, fptr) ;
```

Celle ci lit la donnée à partir du disque pour la placer dans la structure art : le format est similaire à fwrite(). La fonction fread() retourne le nombre d'éléments lu qui correspond normalement au troisième argument de la fonction. La lecture des enregistrements continue jusqu'à que fread() retourne une valeur inférieure 1.

3.6 Accès aléatoire

Tous les fichiers peuvent être organisés d'une ou de deux manières différentes, séquentielle ou aléatoire (on appelle aussi cette dernière fichier à accès direct). Dans un fichier séquentiel, toutes les composantes d'un fichier sont stockées de façon séquentielle, l'une après l'autre. Pour accéder à une composante particulière, il faut commencer au début du fichier et chercher à travers le fichier entier pour trouver la composante qui nous intéresse. Ce type d'accès peut prendre beaucoup de temps, particulièrement si on travaille avec des fichiers volumineux. Cependant, les fichiers séquentiels sont relativement faciles à créer.

Dans un fichier aléatoire, on accède directement à n'importe quelle composante, sans traiter le fichier en entier à partir du début. Ce type d'accès est plus rapide mais difficile à créer et à maintenir. L'accès aux données, dans les programmes précédents, se faisait d'une manière séquentielle. Le programme suivant autorise l'accès aléatoire aux données du fichier crée par le programme 9.

```
/* ----------------------------------------------------------------------------------------------------------
*PROGRAMME 11
*Fichier : alea.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :05/03/99
*Description : lit l'enregistrement d'un article, sélectionné par l'utilisateur
----------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<process.h>
typedef struct
{
        char Designation[41] ;
        int Code ;
```

```
        float Prix ;
        float Stock ;
        }ARTICLE ;
void main()
{
        FILE * fptr ;//pointeur vers fichier
        char fichier[81] ;
        int NumeroEnreg ;
        long int Deplacement ;
        ARTICLE art ;
        clrscr() ;
        puts(''Nom du fichier :'') ;
        gets(fichier) ;
        fptr =fopen(fichier,''rb'') ;//ouvrir fichier en lecture,mode binaire
        if (fptr = = NULL)
        {
                printf('Impossible d'ouvrir le fichier %s en lecture\n'',fichier) ;) ;
                exit(0) ;//fin du programme
        }
        printf(''Enregistrement actuel :%d\n'', ftell(fptr)/sizeof(art)) ;
        printf(''Enter le numéro d'enregistrement :'') ;
        scanf(''%d ,&NumeroEnreg) ;//lit le numéro d'enregistrement
        Deplacement = NumeroEnrg * sizeof(art) ;//calcul son déplacement
        if (fseek(fptr,Deplacement,0) !=0) //par rapport au début du fichier
        {
                printf(''Le pointeur du fichier ne peut être déplacé :'') ;
                exit(0) ;
        }
        fread(&art,sizeof(art), 1 ,fptr) ;

                printf (''Code : %d'' ,&art.code) ;
                printf(''Désignation : %s\n'' ,art.Designation) ;
                printf(''Prix Unitaire : %.2f\n'' ,art.Prix) ;
                printf(''Stock : %.2f\n'' ,art.Stock) ;

        fclose(fptr) ; //ferme fichier
        getch() ;
}
```

Le pointeur de lecture du fichier est un pointeur vers des octets particuliers du fichier. Les programmes qu'on a vus font usage du pointeur. Chaque fois qu'on écrit des données vers un fichier, le pointeur est déplacé vers la fin. Lorsqu'on ouvre un fichier en lecture, le pointeur est positionné au début du fichier. Si le fichier est ouvert pour ajout, en utilisant l'option ''a'', le pointeur est placé à la fin du fichier.

La fonction ftell() permet de connaître la position courante du pointeur :

```
Long ftell(FILE *fptr) ;
```

La fonction fseek() permet de déplacer le pointeur sur une donnée particulière en lui fournissant un déplacement par rapport à l'emplacement spécifié. Cette fonction est déclarée par

nt fssek(FILE *fptr, long Deplacement , int origine)

Fptr : fichier pointé,
Deplacement : Différence en octets entre origine (position pointeur fichier) et la nouvelle position. En mode texte, offset doit être 0 ou une valeur envoyée par ftell.
Origine : Une des trois origines de pointeurs fichier (0, 1, ou 2)

0 : début du fichier
1 : position courante du pointeur
2 : fin de fichier
Valeur Renvoyée par fseek()
Si succès, (pointeur repositionné), 0.
Si échec, non-zéro. fseek renvoie un code erreur uniquement sur un fichier non ouvert ou un périphérique.

4 ENTRE/SORTIE SYSTEME

L'E/S système (ou E/S de bas niveau) est semblable à celle utilisée par MS DOS pour lire et écrire un fichier. Avec l'E/S système, les données ne peuvent pas être écrites comme des caractères individuels ou comme chaînes de caractères ou encore comme données formatées comme pour les E//S standards. Il y a un seul moyen pour écrire les données : comme un tampon plein de caractères.

Ecrire un tampon de caractères ressemble aux E/S des enregistrements avec les E/S standards. Contrairement aux E/S standards, le programmeur doit créer le tampon pour ces données, le remplir avec des valeurs appropriées avant l'écriture. La figure suivante montre que le tampon pour une E/S système est une partie du programme, alors que celui ci était invisible pour une E/S standard.

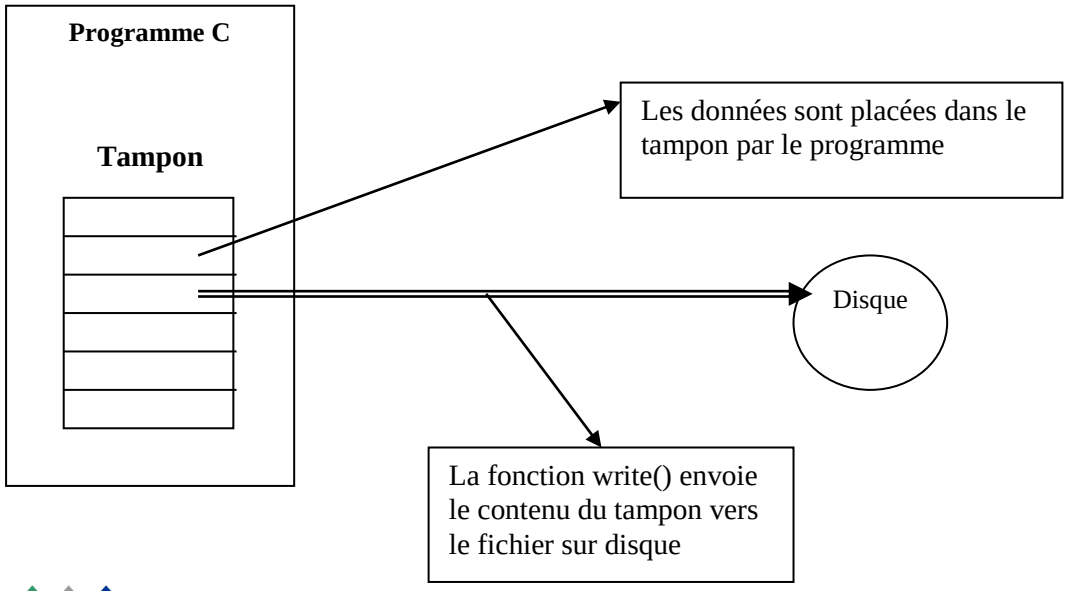


Figure3 : E/S système

L'E/S de bas niveau a plusieurs avantages :

- Le code utilisé par les routines de la bibliothèque est plus court que celui de l'E/S standard
- Plus rapide, car il utilise moins de routines.

4.1 Lecture des fichiers

Le programme suivant lit un fichier disque et affiche son contenu sur l'écran.

```
/* ------------------------------------------------------------------------------------------------------------
*PROGRAMME 12
*Fichier : litsys.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :16/03/99
*Description : lit un fichier avec la méthode d'E/S système
------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
#include<fcntl.h> //constantes symboliques pour open
#include<process.h>
#include<io.h>
#define Long Tampon 512 //taille du tampon en nombre d'octets
char Tampon[Long Tampon] ; // déclaration tampon
void main(int argc,char *argv[])
{
        int Handle, Octets, i ;

        if(argc !=2)
        {
                printf(''Format : c:\>liresys fichier.xxx'') ;
                exit(0) ;
        }
        if ( (Handle =open(argv[1], O_RDONLY | O_BINARY)) < 0)
        {
        printf(''Impossible d'ouvrir le fichier %s'' ,argv[1]) ;
                exit(0) ;
        }
        while( (Octets = read(Handle, Tampon, Long Tampon)) >0)
                for( j=0 ; j < octets ; j++)
                        putch(Tampon[j]) ;
        close (Handle) ;
}
```

4.1.1 Création d'un tampon

La première étape consiste en la création d'un tampon avec les instructions :

```
#define Long Tampon 512
char Tampon[Long Tampon] ;
```

Le tampon est la zone mémoire où les données lues à partir du disque seront placées. La taille du tampon est importante pour des opérations efficaces. Suivant le système d'exploitation, les tampons avec une certaine taille seront gérés plus efficacement que d'autres. Avec MSDOS, la valeur optimale est de 512 octets. Dans d'autre cas un tampon plus grand peut être utilisé en choisissant des multiples de 512 octets.

4.1.2 Ouverture d'un fichier

Comme pour les E/S standards, on doit ouvrir un fichier avant accéder. Ceci est réalisé par l'instruction

```
Handle = open(argv[1], O_RDONLY | O_BINARY)
```

L'ouverture d'un fichier consiste à établir une communication avec le système d'exploitation . On fournit au système à travers la fonction open()

- le nom du fichier qu'on veut ouvrir.
- accès en lecture ou en écriture.
- le fichier est ouvert en mode binaire ou en mode texte.

La méthode utilisée pour indiquer ces caractéristiques est différente de celle utilisée par la méthode d'E/S standard. Chaque caractéristique est indiquée par une constante. La liste de ces constantes symboliques est fournie dans le tableau suivant :

Constantes	**Signification**
O_APPEND	Place le pointeur à la fin du fichier
O_CREAT	Crée un nouveau fichier pou écriture (n'a pas d'effet si le fichier existe déjà)
O_RDONLY	Ouvre le fichier en lecture seulement
O_RDWR	Ouvre le fichier en lecture et en écriture
O_TRUNC	Ouvre le fichier avec troncature. Si le fichier sa longueur est ramenée à zéro. les attributs restent identiques

O_WRONLY	Ouvre le fichier pour écriture seulement
O_BINARY	Ouvre le fichier en mode binaire
O_TEXT	Ouvre le fichier en mode texte

Dans l'exemple du programme 10, on a ouvert le fichier avec les constantes O_RDONLY et O_BINARY c'est à dire en lecture avec le mode binaire. Lorsque plusieurs constantes sont utilisées ensembles, elles sont combinées avec l'opérateur de bits OU (\).

4.1.3 Identification de fichier (Handle)

Au lieu de retourner un pointeur, comme pour la fonction fopen () le fait, la fonction open() retourne une valeur entière appelé identificateur (handle). Ce nombre qui est assigné à un fichier sera utilisé pour se référer à celui ci.

Si open() retourne la valeur – 1, elle signale une erreur d'ouverture

4.1.4 Lecture du fichier dans le tampon

L'instruction suivante lit le contenu du fichier :

```
Octets = read(Handle, Tampon, Long Tampon) ;
```

La fonction read() prend trois paramètres. Le premier est l'indicateur du fichier. Le second est l'adresse du tampon. Le troisième est le nombre maximum d'octets qu'on veut lire(dans l'exemple, la totalité du tampon).

La fonction read() retourne le nombre d'octets mis dans le tampon. Si ce nombre égale zéro, elle indique la fin de fichier.

4.1.5 Fermeture du fichier

Un fichier doit fermé après utilisation. On utilise la fonction close() pour fermer un fichier. La fonction close() prend comme paramètre l'identificateur du fichier à fermer.

```
Close(Handle) ;
```

4.1.6 Messages d'erreur

Comme pour l'E/S standard, pour déterminer l'origine d'une erreur lorsqu'on ouvre un fichier ou lorsqu'on lui accède, on peut utiliser la fonction perror(). Cette fonction prend comme paramètre une chaîne de caractères. Par exemple, on peut remplacer les instructions suivantes dans le programme 12 :

```
Printf(''Impossible d'ouvrir le fichier %s'' ,argv[1] ;
```

Exit(0) ;

par l'instruction

perror(''Impossible d'ouvrir le fichier :'') ;

4.2 Opération sur le tampon

Mettre le contenu d'un fichier dans un tampon a beaucoup d'avantage ; on peut faire différentes opérations sur le contenu du tampon sans le besoin d'accéder à nouveau au fichier. Le programme suivant cherche, dans le fichier, un texte ou une phrase saisie par l'utilisateur.

```
/* ----------------------------------------------------------------------------------------------------------
*PROGRAMME 13
*Fichier : fcherche.cpp
*Auteur :  Mustapha AIT MAHJOUB

*Date :5/03/99


*Description : cherche une phrase dans un fichier
----------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
#include<fcntl.h>
#include<process.h>
#include<io.h>
#include<string.h>

#define Long Tampon 1024
void cherche(char* phrase, int Ltampon) ;
char Tampon[Long Tampon] ;
void main(int argc,char *argv[])
{
        int Handle, Octets, j;
        if(argc !=3)
        {
                printf(''Format : c:\>liresys fichier.xxx phrase'') ;
                exit(0) ;
        }
        if ( (Handle =open(argv[1], O_RDONLY | O_BINARY)) < 0)
        {
                printf(''Impossible d'ouvrir le fichier %s'',argv[1]) ;
                exit(0) ;
        }
        while( (Octets = read(Handle, Tampon, Long Tampon)) >0)
                cherche(argv[2], Octets) ;
        close (Handle) ;
}
void cherche(char* phrase,int Ltampon)
```

```
{
        char* ptr, *p ;
        ptr = Tampon ;
        while ( (ptr =(char*) memchr(ptr, phrase[0], Ltampon )) != NULL)
        if (memcmp(ptr,phrase, strlen(phrase)) = = 0)
        {
                printf(‘’Première occurrencede la phrase:\n’’) ;
                for (p =ptr – 10; p<ptr + 10;p++)
                        putchar(*p) ;
                exit(0) ;
        }
        else ptr++ ;
}
```

Le programme suggère de l'utilisateur de taper deux arguments dans la ligne de commande : la nom du fichier et la phrase à chercher, par exemple

C:\>fcherche fcherche.cpp main

La fonction cherche(), appelée par le programme principale, utilise plusieurs fonctions de manipulation du tampon, la première est la fonction memchr(). Cette fonction cherche, dans le tampon, un caractère spécifique. Dans le programme 13, l'expression

Ptr = (char*) memchr(ptr, phrase[0], Ltampon)

montre trois arguments nécessaires pour memchr(). Le premier est l'adresse du tampon. Le second est le caractère à chercher (le premier caractère de la phrase à chercher). Le troisième est la plage de recherche (taille du tampon). La fonction memchr() renvoie NULL si le caractère n'est pas trouvé dans le tampon. Sinon, elle retourne un pointeur vers le caractère trouvé. La fonction cherche() entre dans une instruction if pour vérifier si la phrase débutant par le caractère trouv é est identique à la phrase recherchée. Cette comparaison est faite avec la fonction memcmp() dans l'expression

if (memcmp(ptr,phrase, strlen(phrase)) = = 0)

La fonction memcmp() prend trois paramètres : un pointeur vers un emplacement du tampon où la comparaison doit débuter, l'adresse de la phrase à chercher et la longueur de la phrase. Si la phrase existe, la fonction memcmp() retourne la valeur zéro.

4.3 Ecrire dans un fichier

Comme exemple, essayons de voir le programme suivant qui copie un fichier vers un autre ; il limite le commande COPY du DOS.

Pour utiliser cette fonction, l'utilisateur tape le nom du fichier source (qui existe déjà) et le fichier de destination (qui va être crée) dans la ligne de commande.

C:\>fcopie source.txt desti.txt

```
/* ----------------------------------------------------------------------------------------------------------------
*PROGRAMME 14
*Fichier : fcopie.cpp
*Auteur :  Mustapha AIT MAHJOUB
*Date :5/03/99

*Description : Copie un fichier
----------------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
#include<fcntl.h>
#include ‘’c : \borlandc\include\sys\stat.h’’
#include<process.h>
#include<io.h>
#include<string.h>
#define Long Tampon 4096
char Tampon[Long Tampon] ;

void main(int argc,char *argv[])
{
        int E_ Handle, S_Handle , Octets;



        if(argc !=3)
        {
                printf(‘’Format : c:\>fcopie source.xxx desti.xxx’’) ;
                exit(0) ;
        }
        if ( (E_Handle =open(argv[1], O_RDWR | O_BINARY)) < 0)
        {
                printf(‘’Impossible d’ouvrir le fichier %s’’,argv[1]) ;
                exit(0) ;
        }
        if ((S_Handle =open(argv[2], O_CREAT | O_WRONLY|
                                        O_BINARY, S_IWRITE )) < 0)
        {
                printf(‘’Impossible d’ouvrir le fichier %s’’ ,argv[2]) ;
                exit(0) ;
        }
        while( (Octets = read(E_Handle, Tampon, Long Tampon)) >0)
                write   (S_Handle, Tampon,Octets) ;
        close (E_Handle) ;
        close (S_Handle) ;
}
```

Les deux fichiers sont ouverts. L'un est la source à qui on assigne l'identificateur E_Handle. L'autre est le fichier de destination dont l'identificateur est S_Handle. L'expression qui ouvre le fichier de destination est

```
If ( (S_Handle =open(argv[2], O_CREAT | O_WRONLY|
                                    O_BINARY, S_IWRITE  )) < 0 )
```

Pour créer un nouveau fichier, on utilise la constante symbolique O_CREAT. Nous voulons écrire ce fichier et ne pas le lire, pour le faire, nous utilisons la constante O_WRONLY . La constante O_BINARY indique que le fichier est ouvert en mode binaire.

Lorsque la constante O_CREATest utilisée, une autre variable doit être ajouter à la fonction open() pour indiquer l'état de la lecture/Ecriture du fichier qui va être crée. Ces options sont appelées arguments de permission (droit d'accès). Il existe trois possibilités :

S_IWRITE	: Ecriture permise
S_IREAD	: Lecture permise
S_IREAD | S_IWRITE	: Lecture et écriture permises

Pour utiliser ces possibilités, il faut inclure le fichier ‘’sys-stat.h dans le fichier source du programme.

La fonction write() est similaire à read(). Elle prend trois arguments : l'identificateur du fichier, l'adresse du tampon et le nombre d'octets à écrire.