**Module** : Langage C

**Objectifs** : Proposer une solution logicielle conforme à un cahier des charges simple

**Compétences visées** :

Plus précisément, les étudiants, au terme de ce cours, devraient être capables de :

- Concevoir un algorithme à partir d'un cahier des charges simple
- Transcrire un algorithme en C
- Compiler, corriger et tester un programme

**Contenu** :

Chapitre 1 : Structure d'un programme en C

Chapitre 2 : Les structures des contrôles

Chapitre 3 : Les tableaux

Chapitre 4 : Les pointeurs

Chapitre 5 : Les fonctions

Chapitre 6 : Les chaînes des caractères

Chapitre 7 : Les structures et les fichiers

Chapitre 8 : Allocation dynamique de mémoire

**Mots-clefs** : Algorithmes, Langage de programmation

**Intervenant** : TATI LAMBERT

**Contact** : lamb_tati@yahoo.fr (00242) 06 916 09 69

## Chapitre 1 : Structure d'un programme en C

### 1- Présentation du langage C

### 1-1 Historique

Langage de programmation développé en 1970 par Dennie RITCHIE aux laboratoires Bell d'AT&T.

Il fut limité à l'usage interne de Bell jusqu'en 1978 date à laquelle Brian KEKERNIGAN et Dennie RITCHIE publièrent les spécialisations du langage : « **The C programming Language** »

Au milieu des années 1980 la popularité du langage était établie.

De nombreux compilateurs ont été écrits, mais comportant quelques incompatibilités portant atteinte à l'objectif de portabilité.

**1-2 Intérêts du langage**

- Langage polyvalent permettant le développement de système d'exploitation, de programmes applications scientifiques et de gestion

- Langage structuré

- Langage évolué qui permet néanmoins d'effectuer des opérations de bas niveau

- Portabilité due à l'emploi de bibliothèques dans lesquelles sont reléguées les fonctionnalités liées à la machine

- Grande efficacité et puissance

1-3- **Qualités attendues d'un programme**

-Clarté

-Simplicité

-Modularité

-Extensibilité

1-4- **Jeu de caractères**

**-**26 caractères de l'alphabet (minuscules, majuscules)

-Chiffres 0 à 9

- Caractères spéciaux :

| | | | | | |
|---|---|---|---|---|---|
| ! | * | + | \ | '' | < |
| # | ( | = | \| | { | > |
| % | ) | ~ | ; | ] | / |
| ^ | - | [ | : | , | ? |
| & | - | } | ' | . | (espace) |

-Séquences d'échappement telles :

- passage à la ligne (\n)

- tabulation horizontale (\t)

- backspace (\b)

**1-5- Identificateurs et mots-clés**

**Identificateur :**

Nom donné aux diverses composantes d'un programme ; variables, tableaux, fonctions.

- Formé de lettres et de chiffres ainsi que du caractère_ permettant une grande lisibilité.

Le 1er caractère doit obligatoirement être une lettre ou bien le caractère_

- Peut contenir jusqu'à 31 caractères minuscules et majuscules.
- Il est d'usage de réserver les identificateurs entièrement en majuscules aux variables du préprocesseur.

**Exemples :**

- Identificateurs valides :
  x y12 somme_1 _temperature fin_de_fichier
- Identificateurs invalides :
  4eme commence par un chiffre
  x #y caractère non autorisé (#)
  taux change caractère non autorisé (espace)

**Mots réservés (mots clés)**

Auto extern sizeof

break float static

case for struct

char goto switch

const if typedef

continue int union

default long unsigned

do register void

double return volatile

else short while

enum signed

2-**Structure d'un programme en C**

**2.1- Structure provisoire**

Inclusion des bibliothèques

Void main ()

{

Corps du programme principal

}

**Exemple** : Ecrire un programme qui affiche le message « Bonjour le monde » à l'écran

```
#include <stdio.h> //inclusion de la bibliothèque standard

Void main () // en-tête du programme principal

{

Printf(‘’ Bonjour le monde’’) ; // affiche bonjour le monde à l’écran

}
```

**Remarques :**

- Toute instruction en C se termine toujours par un point virgule

- // commentaire sur une ligne

- /* commentaire sur plusieurs lignes */

- instruction préprocesseur commence par #

2.2- **Compilation et édition des liens**

Le fichier source d’une application écrite en langage C a une extension « .c »

La compilation de ce fichier s’effectue à l’aide de la commande cc. Sans autre spécification, cette commande enchaîne 3 étapes :

- Appel au pré-processeur
- Appel au compilateur
- Appel à l’éditeur de liens

**3- Les types de données**

**3-1- Les réels**

-Les réels simple précision en C -> float (4 octets)

-Les réels à double précision en C -> double (8 octets)

**3-2- Les entiers**

-Entier court en C -> short int (2 octets)

-Entier par défaut en C -> int (2 octets )

-Entier long en C -> long int (4 octets)

**3-3- Les caractères**

-Les caractères en C -> char (1 octet)

Un caractère est considéré comme un entier qu’on pourra donc utiliser dans une expression arithmétique

**Remarque** :

Il n'y a pas le type chaîne de caractère en C, pas de type booléen en C

**4-Déclaration des variables**

**4-1-Déclaration d'une variable**

**Syntaxe** : type nom_variable ;

**Exemple** : int x ; //déclaration d'une variable entière

**4-2-Déclaration de plusieurs variables de même type**

**Syntaxe** : type nomvar1,nomvar2,…, nomvarn ;

**Exemple** : float x,y ;//déclaration de deux variables de type réel

char x,y,t ;

4-3-**Déclaration d'une constance en C**

**Syntaxe** : const type nom_constante = valeur ;

**Exemple** : const float pi = 3.14 ;

Constantes symboliques

#define pi 3.14 // directive de compilation

**4-4- Déclaration d'un type constante énumérée**

**Syntaxe** : enum nom_enumeration {liste des constantes} ;

**Exemples** :

1) Enum bool {false, true} ;
2) Enum couleur {noir, bleu, vert, jaune} ;

**Utilisation** : enum bool v ; /* v est une constante énumérée de type bool qui peut prendre deux valeurs : false ou true */

**5-Les opérateurs**

**5-1-Les opérateurs arithmétiques**

addition ->+

division ->/

multiplication ->*

soustraction ->-

modulo (reste de la division entière ) ->%

**5-2 Les opérateurs logiques**

OU logique (inclusif) -> | |

OU logique (exclusif) -> |

**ET logique ->&&**

NON logique ->!

Egalité logique ->==

**5-3-Les opérateurs de comparaison**

Supérieur ->>

Inférieur -><

Supérieur ou égal ->>=

Inférieur ou égal -><=

Différent de ->!=

Egal à ->==

**5-4-L'opérateur d'affectation**

Affectation en C -> =

**Exemples** :

Int x ;

X=3 ; // affecte la valeur 3 à x

float y = 0 ; // déclaration et initialisation

char Y ='a', y='u' ;

**Remarque** : le langage C respecte la casse (différence entre majuscule et minuscule)

**Les formats de données utilisées avec printf**

%d -> entier

%ld -> entier long

%f -> flottant ou réel

%lf -> réel long

%e -> écriture exponentielle d'un réel

%c -> caractère

**Exemples :**

int a ;float ;

printf(''%3d '',a) ; // entier avec 3 caractères minimum

printf(''%10.3f'',b) ; // notation décimale sur 10 caractères ayant 3 chiffres après la virgule

**Syntaxe de printf** : printf(format, liste_d'expression) ;

**Application 1** :

Ecrire un programme qui calcule et affiche la surface d'un rectangle de longueur L = 10 m et de largeur l = 5 m

```
#include <stdio.h>

#include<conio.h>

Void main ()

{

clrscr(); // efface l'écran

Int  L = 10, l = 5 , s;

S= L*l;

Printf(''la surface du rectangle est : %d'',s);

getch() ; // attente d'une saisie au clavier

}
```

**Application 2** :

Ecrire un programme qui calcule et affiche la somme de deux réels

```
#include<stdio.h>

#include<conio.h>

Void main()

{

clrscr(); //efface l'écran

float x,y;
```

```
printf(''Entrer le premier reel:\n'') ;
scanf(''%f'',&x) ;
printf(''Entrer le deuxième reel:\n'') ;
scanf(''%f'',&y) ;
printf(''la somme de %f et de %f est : %f'',x,y,x+y) ;
getch() ;
}
```

**Application 3** :

Ecrire un programme qui calcule et affiche la surface d'un cercle de rayon quelconque

```
#include<stdio.h>
#include<conio.h>
#define pi 3.14
Void main()
{
clrscr();
float r;
printf(''Entrer le rayon :\n'');
scanf(''%f'',&r) ;
printf(''la surface du cercle est :%f'',pi*r*r) ;
getch() ;
}
```

**5-5-Les opérateurs conditionnels**

**La clause : if (si)**

**Syntaxe** : if(condition)

Instruction ;

Si la condition est vraie alors on exécute l'instruction

**Remarques** : - L'instruction peut être un bloc d'instruction

- Chaque bloc d'instruction doit être délimité par { }

- Chaque condition doit être délimité par ( )

**Application** :

Ecrire un programme qui permet de résoudre l'équation de type ax = 0

```
#include<stdio.h>
#include<conio.h>
Void main()
{
clrscr();
float a ;
printf(''Entrer le coefficient de l'équation :\n'');
scanf(''%f'',&a) ;
If(a !=0)
printf(''l'équation admet une solution unique x = 0'');
If(a==0)
printf(''Infinité des solutions'') ;
getch() ;
}
```

La clause : if ……. else (si ……. sinon)

**Syntaxe** : if(condition)

Instruction 1 ;

Else instruction 2 ;

Si la condition est vraie on exécute l'instruction 1 sinon on exécute l'instruction 2

**Application 1**

Ecrire un programme qui permet de résoudre l'équation de type ax + b = 0

```
#include<stdio.h>
#include<conio.h>
```

```
Void main()
{
clrscr();
float a, b, x;
printf(''Entrer les coefficients de l'équation :\n'');
scanf(''%f%f'',&a,&b) ;
If(a !=0)
{
x=-b/a;
Printf(''solution unique = %f'',x);
}
else //a = 0
{
If(b!=0)
printf(''pas de solution'');
else printf("Infinité des solutions'') ;
}
getch();
}
```

**Application 2**

Ecrire un programme qui lit un entier au clavier et qui teste si cet entier est positif ou négative

```
#include<stdio.h>
#include<conio.h>
Enum bool {false, true};
Void main()
{
clrscr();
```

```
int n;
Enum bool test;
Printf(''Entrer un entier:'');
Scanf(''%d'',&n) ;
If(n>=0)
{  test = true;
 Printf(''%d est un entier positif'',n);
}
else
{ test = false ;
 Printf(''%d est un entier négatif'', n) ;
}
getch() ;
}
```

La clause : else ……. if (sinon …… si)

**Syntaxe** : if(cond 1)

```
            Instruction 1 ;
           else if (cond 2)
             Instruction 2;
          else if (cond 3)
           Instruction 3;
         .
         .
         .
        else if (cond n)
         Instruction n;
        else instruction;
```

**Application**

Ecrire un programme qui lit un caractère au clavier et qui teste si c'est une voyelle

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
char x;
printf(''Entrer un caractère:\n'');
scanf(''%c'',&x) ;
If(x=='a') printf(''C'est une voyelle '') ;
else if(x=='i') printf(''c'est une voyelle'') ;
else if(x=='o') printf(''c'est une voyelle'') ;
else if(x=='u') printf("c'est une voyelle") ;
else if(x=='e') printf(''c'est une voyelle'') ;
else if(x==yi') printf(''c'est une voyelle'') ;
else printf("c'est une consonne'') ;
getch() ;
}
```

**5-6-Les opérateurs d'incrémentation et de décrémentation**

**Incrémentation**

```
x++ ;//x=x+1
++x ; //x=x+1
```

**Décrémentation**

```
x-- ; //x=x-1
--x ; // x=x-1
```

**Pré-incrémentation**

x=++a ;// ++a ; puis x=a ;

**Remarques** :

- L'opérateur de pré-incrémentation est prioritaire par rapport à l'affectation

- ++ est un opérateur de pré-incrémentation lorsqu'il est placé à gauche de la lvalue sur laquelle il porte

**Post-incrémentation**

x=a++ ;// x=a ; puis a++ ;

**Remarques** :

- L'affectation est prioritaire par rapport à l'opérateur de post-incrémentation

- ++ est un opérateur de post-incrémentation lorsqu'il est placé à droite de la lvalue sur laquelle il porte

**5-7-Les opérateurs d'affectation élargie**

**Syntaxe** : lvalue = lvalue opérateur expression ; < = > lvalue opérateur = expression ;

Exemples :

X+ = a ; // x = x + a ;

X- = a ; // x = x - a ;

X* = a ; // x = x * a ;

X/ = a ; // x = x / a ;

X% = a ; // x = x % a ;

**5-8-L'opérateur séquentiel**

**Intérêt :** permet d'exprimer plusieurs calculs successifs au sein d'une même expression

Par exemple : a*b ; i+j est une expression qui évalue d'abord a*b, puis i+j et qui prend comme valeur la dernière calculée (donc ici celle de i+j). Certes, dans ce cas, le calcul de a*b est inutile puisqu'il n'intervient pas dans la valeur de l'expression globale et qu'il ne réalise aucune action.

En revanche, une expression telle que :

i++ ; a+b peut présenter un intérêt puisque la première expression (dont la valeur ne sera pas utilisée ) réalise en fait une incrémentation de la variable i

**5-9- L'opérateur sizeof**

L'opérateur sizeof, dont l'emploi ressemble à celui d'une fonction, fournit la taille en octets

**Exemple :**

Supposons le type int est représenté sur 2 octets et le type double sur 8 octets

int n ;

double z ;

sizeof(n) vaudra 2

sizeof(z) vaudra 8

**Intérêt** : lorsque l'on souhaite écrire des programmes portables dans lesquels il est nécessaire de connaître la taille exacte de certains objets

**6-Exercices**

**Exercice 1**

Ecrire un programme qui permet de résoudre l'équation de type $ax^2 + bX + c = 0$

**Exercice 2**

Ecrire un programme qui retourne le maximum et le minimum de trois nombres entiers lus au clavier

**Exercice 3**

Ecrire un programme qui affiche le message « reçu », « admis aux épreuves orales » ou « ajourné » selon que la note obtenue est supérieure ou égale à 10, compris entre 8 et 10, ou inférieure à 8

**Chapitre 2 : Les structures de contrôle**

**1-La structure switch … case**

La structure switch permet de faire des choix multiples (uniquement) sur une liste de valeurs constantes. Elle correspond à une cascade d'instruction if … else .

La syntaxe est la suivante :

```
Switch(expression)
{
 case constante1 : instruction1 ; break ;
case constante2: instruction2 ; break;
.
.
.
case constante n: instruction n; break;
default : instructions;
}
```

Si la valeur de l'expression est égale à l'une des constantes, les instructions correspondantes sont exécutées. Sinon les instructions correspondantes à **default** sont exécutées

**Remarque** : l'instruction du default n'est pas obligatoire

**Exemple 1 :**

Simulation d'un feu de route

```
#include<stdio.h>
#include<conio.h>
Void main()
{
clrscr();
int feu;
const int Vert = 1, Rouge =2, Orange =3;
printf("Quel feu:");
```

```
scanf(''%d'',&feu) ;

switch(feu)

{

case Vert : printf(''Passe''); break;

case Rouge : printf(''Arrêt ''); break;

case Orange : printf(''Ralentis''); break;

default: printf(''Valeur feu inconnu'');

}

getch() ;

}
```

**Exemple 2 :**

Ecrire un programme qui demande à l'utilisateur de fournir un nombre compris entre 0 et 3 et qui l'affiche à l'écran.

```
#include<stdio.h>

#include<conio.h>

Void main()

{

Clrscr();

Int n;

Printf("saisir le nombre:");

Scanf(''%d '',&n) ;

Switch(n)

{

Case 0 : printf(''Zero\n''); break;

Case 1: printf(''Un\n''); break;

Case 2: printf(''deux\n''); break;

Default: printf("Error: le nombre doit être compris entre 0 et 3\n'') ;

}
```

```
Printf(''fin'') ;
getch() ;
}
```

**Exemple 3 :**

Simuler une calculatrice à 4 opérateurs (+, - , *, /)

```
#include<stdio.h>
#include<conio.h>
Void main()
{
Clrscr();
Char oper;
Float a,b;
Printf(''Entrer les deux reels:'');
Scanf(''%f%f'',&a,&b) ;
Printf(''Entrer un operateur :'') ;
Scanf(''%c'',&oper) ;
Switch(oper)
{
case '+' : printf(''La somme est : %f'', a+b) ; break ;
case '-' : printf(''La soustraction est : %f'', a-b) ; break ;
case '*' : printf(''La multiplication est : %f'', a*b) ; break ;
case '/' : { if(b==0)
              Printf(''Error'') ;
               Else printf(''La division est : %f'', a/b);
            } break ;
default : printf
}
```

```
getch() ;
}
```

**2-La boucle while (tant que)**

La syntaxe de while est la suivante :

**while (expression)**

**instruction ;**

Tant que expression est vérifiée, l'instruction est exécutée. Si expression est non vérifiée au départ, l'instruction ne sera jamais exécutée. L'instruction peut évidemment être une instruction composée.

**Exemple 1 :**

Ecrire un programme qui affiche les 100 premiers entiers non nuls

```
#include<stdio.h>
#include<conio.h>
Void main()
{
Clrscr();
Int i =1;
While (i<=100)
{
Printf("\n i=%d",i);
I++;
}
Getch();
}
```

**Exemple 2 :**

Analyser le programme suivant:

```
#include<stdio.h>
#include<conio.h>
Void main()
```

```
{
Clrscr();
Int n, som=0;
While(som<100)
{
Printf("Donnez un nombre:");
Scanf(''%d",&n);
Som+=n ;
}
printf('' La somme obtenue est : %d'', som) ;
getch() ;
}
```

**3-La boucle do … while**

Il peut arriver que l'on veuille effectuer le test d'arrêt qu'après avoir exécuté l'instruction.

Dans ce cas, on utilise la boucle do … while.

Sa syntaxe est la suivante :

**do**

**Instruction ;**

**While (expression) ;**

Cela signifie répéter l'instruction tant que l'expression est vraie. Ce qui veut dire également que l'instruction est exécutée au moins une fois

**Exemple 1 :**

Ecrire un programme qui force l'utilisateur à entrer le nombre 482

```
#include<stdio.h>
#include<conio.h>
Void main()
{
Clrscr();
```

```
Int a;
Do
{
Puts(“Veuillez entrer le nombre 482”); // affiche une chaîne à l’écran
Scanf(‘’%d ‘’,&a) ;
}
while (a !=482);
Puts(“c’est gentil de m’avoir obéi”);
getch();
}
```

**Exemple 2:**

Ecrire un programme qui calculi la somme des N premiers entiers non nulls

```
#include<stdio.h>
#include<conio.h>
Void main()
{
Clrscr();
Int som=0;
Int n, i=1;
Printf(“Saisir N”);
Scanf(“%d”,&n);
do
{
Som+=i ;
i++ ;
}
While(i<=n) ;
```

```
printf("La somme des %d premiers entiers non nulls vaut %d\n", n, som);
getch() ;
}
```

**Exemple 3 :**

Ecrire un programme qui affiche les 100 premiers entiers par pas de 3

```
#include<stdio.h>
#include<conio.h>
Void main()
{
Clrscr();
Int i=0, compt=0;
do
{
Printf("%d\n",i);
Compt++;
I+=3;
}
While(compt<=100);
Getch();
}
```

**Exemple 4:**

Afficher toutes les puissances de 2 inferieurs à 100000

```
#include<stdio.h>
#include<conio.h>
Void main()
{
Clrscr();
```

```
Const long int M=100000;
Long int p = 1;
do
{
Printf("%ld\n",p);
p=p*2;
}
while(p<=M);
getch();
}
```

**4-La boucle for**

La syntaxe de for est :

**For (expr1 ; expr2 ;expr3)**

**Instruction ;**

Expr1 : représente les instructions d'initialisation de la boucle

Expr2 : représente la condition d'arrêt de la boucle

Expr3 : l'incrémentation de la boucle

**Exemple :**

Afficher tous les nombres de 0 à 9

```
#include<stdio.h>
#include<conio.h>
Void main()
{
Clrscr();
```

```
Int I;
For(i=0; i<10;i++)
Printf("\n i=%d",i);
Getch();
}
```

**Remarque :**

Une boucle for est un cas particulier de la boucle while et donc une version équivalente plus intuitive de la boucle for est:

```
Expr1 ;
while(expr2)
{
instruction ;
Expr3 ;
}
```

**Exemple :**

Afficher tous les nombres de 0 à 9

```
#include<stdio.h>
#include<conio.h>
Void main()
{
Clrscr();
Int I = 0;
While (i<=9)
{
Printf("\n I = %d",i);
I++;
}
getch();
```

}

**Remarque :**

Les trios expressions utilisées dans une boucle for peuvent être constituées de plusieurs expressions séparées par des virgules. Cela permet par exemple de faire plusieurs initialisations à la fois ;

**Exemple 1 :**

Calculer le factorielle d'un entier n lit au clavier

```
#include<stdio.h>

#include<conio.h>

Void main()

{

clrscr();

int n,i, fact;

Printf("Saisir l'entier :");

Scanf("%d",&n);

For(i=1 , fact=1 ;i<=n ;i++)

Fact*=I;

Printf("%d!=%d\n",n,fact);

Getch();

}
```

**Exemple 2:**

Afficher les 100 premiers entiers par pas de 4

```
#include<stdio.h>

#include<conio.h>

Void main()

{

Clrscr();

Int I, cpt;

For(cpt=0,i=0;cpt<=100;i+=4,cpt++)
```

```
Printf("%d\n",i);
Getch();
}
```

**5-L'instruction break**

**Break** permet de sortir de la boucle la plus interne

Exemple :

```
For(i=0;i<5 ;i++)
{
Printf(''Boucle for :'');
If(i==2) break;
Printf("%d fois \n",i+1);
}
```

**Donne comme résultat :**

**Boucle for : 1 fois**

**Boucle for : 2 fois**

**Boucle for :**

**6-L'instruction continue**

"**continue**" permet de passer prématurément au tour de boucle suivant.

« **continue** » s'applique à la boucle la plus interne

**Exemple :**

```
For(i=0;i<5 ;i++)
{ printf("Boucle for :");
If(i==2) continue;
Printf("%d fois\n",i+1);
}
```

**Donne comme résultat:**

**Boucle for : 1 fois**

**Boucle for : 2 fois**

**Boucle for : 3 fois**

**Boucle for : 4 fois**

**Boucle for : 5 fois**

**7-L'instruction goto**

**Goto** permet de se brancher à un emplacement quelconque du programme dans la même fonction

```
For(i=0;i<5 ;i++)
{
printf(''Boucle for :'') ;
If(i==2) goto sortie;
printf(''%d fois \n'',i+1);
}
sortie: printf('' \n fin\n'');
```

**donne comme résultat :**

**boucle for : 1 fois**

**boucle for : 2 fois**

**boucle for :**

**fin**

L'instruction **goto** est généralement utilisée pour sortir d'une encapsulation de boucle

**8-Exercices**

**Exercice 1**

Calculer la somme des 100 premiers entiers par pas de 3

**Exercice 2**

Ecrire un programme qui affiche les 10 premiers nombres magiques.

Un nombre est dit magique lorsqu'il est égal à la somme de ses diviseurs sauf lui-même par exemple :
6 = 1 + 2 + 3

**Exercice 3**

Ecrire un programme qui affiche les 100 premiers nombres premiers (nombre ayant 2 diviseurs 1 et lui-même)

**Exercice 4**

Saisir un entier positif n et un réel x puis calculer et affiche $x^n$

**Exercice 5**

Ecrire un programme qui affiche toutes les tables de multiplication.

Pour réfléchir au problème commencer par écrire une table de multiplication

**Exercice 6**

Afficher le minimum, le maximum et la moyenne des notes d'une classe de N élèves. N à saisir au clavier

**Exercice 7**

Evaluer la somme suivante pour un N fixé : 1+ 1/1 ! + 1/2 ! +… + 1/N !

**Exercice 8**

Ecrire un programme qui lit une série de notes au clavier et qui affiche leur moyenne.

La fin de la saisie se fera en tapant la note -1

**NB** : une note est comprise entre 0 et 20

**Exercice 9**

Ecrire un programme de jeu demandant de deviner un nombre entre 0 et 10 choisi par l'ordinateur. On ne donnera pas d'indications avant la découverte de la solution, où l'on indiquera le nombre d'essais. La solution sera choisie par l'ordinateur par la fonction rand() qui rend un entier aléatoire (déclarée dans stdlib.h)

```
#include<stdio.h>

#include<conio.h>

#include<stdlib.h>// pour rand()

#include<time.h>// pour trouver l'heure pour srand

Void main()

{

Clrscr() ;

int solution, reponse, nb_essais =0 ;
```

```
{ time_t ; srand((unsigned)time(&t)) ; /* initialiser le générateur à partir du compilateur de temps,
                                        pour qu'il soit plus aléatoire*/
}
solution = rand()%11 ; // reste sera toujours entre 0 et 10
do
{
nb_essais++ ;
puts(''proposez votre nombre entre 0 et 10'') ;
scanf(''%d '',&reponse) ;
}
while(reponse !=solution) ;
Printf(''trouvé en %d essais\n",nb_essais) ;
getch() ;
}
```

**Exercice 10**

La fonction **kbhit** appartient à la bibliothèque **conio.h**.

La fonction kbhit teste si un caractère a été frappé au clavier. Tant que ce n'est pas vrai **kbhit** renvoie 0

**Exemple d'utilisation** :

```
While (kbhit()==0) // tant que aucun caractère tapé
{
Bloc
}
```

Ou encore

```
While ( !kbhit()) // ici c'est pour dire tant que kbhit n'est pas vraie ie renvoie 0
{
Bloc
```

}

**Exercice 11**

Ecrire un programme qui affiche le carré des nombres 1, 2, 3,…, toutes les 500 ms tant que aucun caractère n' a été frappé au clavier

```
#include<stdio.h>

#include<conio.h>

Void main()

{

Clrscr();

int I,n;

float note, som=0,moy;

puts("Nombres des notes?");

scanf("%d",&n);

for(i=0;i<n ;i++)

{

printf(''Entrez votre %d ième note",i+1) ;

scanf(''%f'',&note) ;

Som+=note ;

}

moy = som/n ;

printf(''moyenne calculée :%5.2f\n'',moy);

getch() ;

}
```

**Chapitre 3** : **Les Tableaux**

**1-Définition**

Un tableau est une structure de données permettant de stocker plusieurs informations de même type. Un tableau est caractérisé par sa taille et par ses éléments

**2-Déclaration d'un tableau à une dimension**

**Syntaxe** : type nom_tableau[dim] ;

Cette déclaration signifie que le compilateur réserve **dim** places en mémoire pour ranger les éléments du tableau

**Exemples** :

Int tab[10] ;

Float nombre[20] ;

Char matrice[30] ;

**Remarque** : **dim** est nécessairement une valeur numérique

**Utilisation** :

- Un élément du tableau est repéré par indice
- Les éléments du tableau sont indicés de **0** à **dim -1**

**Exemple** : tab[2]= 5 ;

nombre[i] =6.789 ;

printf(''%d'', tab[i]) ;

scanf(''%f'',&nombre[i]) ;

**Application 1**

Remplir un tableau de 10 entiers de la manière suivante : tab[i] <- $2^i$

```
#include<stdio.h>

#include<conio.h>

Void main()

{

clrscr();

int I, tab[10];

tab[0] = 1;
```

```
for(i=1;i<10;i++)

tab[i]=2*tab[i-1];

// affichage des elements du tableau

for(i=0;i<10;i++)

printf(''%d\t'',tab[i]) ;

getch();

}
```

**Application 2**

Saisir 10 réels, les ranger dans un tableau. Calculer et afficher la moyenne et l'écart entre note et moyenne

```
#include<stdio.h>

#include<conio.h>

#define n 10

Void main()

{

Clrscr();

int I;

float som, moy, tab[n];

//saisie des elements du tableau

for(i=0;i<n;i++)

{

printf(''l'élément %d numéro : \n'',i) ;

Scanf(''%f'',&tab[i]) ;

}

for(i=0,som=0;i<n;i++)

Som+=tab[i];

moy=som/n;

printf(''la moyenne des %d notes est : %5.2f\n'', n, moy);
```

```
for(i=0;i<n ;i++)

Printf(''l'écart de la note %d est : %5.2f\n'', i, tab[i] – moy) ;

getch() ;

}
```

**3-Initialisation d'un tableau à une dimension**

```
Int tab[10]= {0, 1, 0, 0} ;
```

Donc :

```
Tab[0]=0 ;

Tab[1]=1 ;

Tab[2]=0 ;

Tab[3]=0 ;
```

**Remarques** :

- On ne pas obligé d'initialiser tous les éléments d'un tableau
- Les valeurs manquantes seront, suivant la classe d'allocation du tableau, initialisées à 0 (statique) ou aléatoire (automatique)

Autres exemples :

```
float tab[20]={ 0.1, 0.2, 11.3, 4.5} ;

char voy[6]={'a', 'i', 'u', 'e', 'o', 'y'} ;
```

**4-Déclaration d'un tableau à deux dimensions**

Syntaxe : type nom_tableau[dim1][dim2] ;

Dim1 représente le nombre de ligne

Dim2 représente le nombre de colonne

**Exemple** :

```
Int mat[20][10] ;//déclaration d'un tableau de 20 lignes pour 10 colonnes
```

Chaque élément du tableau est repéré par 2 indices i et j

I=ligne

J=colonne

mat[i][j] représente l'élément du tableau mat se trouvant à l'intersection entre la ligne i et la colonne j

**Application 1**

Ecrire un programme qui affiche la matrice magique d'ordre 4

Mat[i][j] <- i*j  // matrice logique

```
#include<stdio.h>
#include<conio.h>
Void main()
‘
Clrscr();
Int mat[4][4], I, j;
For(i=0;i<4;i++)
{
For(j=0;j<4;j++)
Mat[i][j]=i*j;
}
puts(‘’Affichage de la matrice  magique d’ordre 4’’) ;
for(i=0 ;i<4 ;i++)
{
for(j=0 ;j<4 ;j++)
printf(‘’%d\t’’, mat[i][j]);
printf(‘’\n”);
}
getch();
}
```

**Application 2**

Saisir une matrice d'entiers 2x2, calculer et afficher son déterminant

**5-Initialisation d'un tableau à deux dimensions**

L'initialisation se fait ligne par ligne

**Exemple** :

Int mat[4][4] ={{0, 0, 0, 0},{0, 1, 2, 3},{0, 1, 4, 6},{0, 3, 6, 9}} ;

**Application**

Ecrire un programme qui affiche

**L M M J V S D**

**U A E E E A I**

**N R R U N M M**

**D D C D D E A**

**I I R I R D N**

**E E I C**

**D D H**

**I I E**

```
#include<stdio.h>
#include<conio.h>
Void main()
{
Clrscr();
int i,j;
char semaine [8][7] ={{'L', 'M','M','J','S','D'},…, {' ', ' ', 'I',' ', 'I',' ','E'}};
puts("Affichage du tableau semaine'');
for(i=0;i<8 ;i++)
{
for(j=0;j<7 ;j++)
printf(''%c\t'', semaine[i][j]);
printf(''\n'');
}
getch();
```

}

**6-Tableaux à plus de deux dimensions**

On procède de la même manière en ajoutant les éléments de dimensionnement ou les indices nécessaires

**Exercice**

Ecrire un programme qui affiche le triangle de pascal d'ordre 10

```
#include<stdio.h>
#include<conio.h>
Void main()
{
Clrscr();
int  mat[10][10], I, j;
for(i=0 ;i<10 ;i++)
{
if((j==0)||(j==i))
mat[i][j]=1;
else
mat[i][j]=mat[i-1][j-1]+mat[i-1][j];
}
for(i=0 ;i<10 ;i++)
{
For(j=0;j<=i;j++)
printf(''%d\t'', mat[i][j]);
printf(''\n'');
}
getch();
}
```

**Chapitre 4 : L es Pointeurs**

**1- Définition**

Un pointeur est une variable destinée à contenir l'adresse mémoire d'une autre variable

**2- Déclaration d'un pointeur**

**Syntaxe** : type *nom_pointeur ;

**Exemples** : int *p ; // p est un pointeur sur un entier

float *x, *y ; // x, y pointeurs de réels

**3- Comment stocker une adresse dans un pointeur ?**

**Syntaxe** : nom_pointeur = &nom_variable ;

**Exemple** : int *p, x= 10 ; // p est un pointeur sur un entier

P=&x ; // p pointe sur x

**Remarques** :

- Le nom du pointeur précédé de * représente la variable pointée
- Le nom du pointeur sans * représente l'adresse de la variable pointée ou l'adresse de la zone pointée

**Exemple 1**

Analyser le programme

```
Void main()
{
 int  i, j, *p ;
i=5 ;
p=&i ;
j=*p ;
*p=j + 2;
}
```

**Exemple 2**

```
#include <stdio.h>
#include<conio.h>
Void main()
```

```
{


Clrscr();

int  x, *p; // p est un pointeur d'entier

x=4 ;

p=&x ; // p pointe sur x

*p = 1 ; // x = 1

Printf(‘‘ le resultat est : %d’’, *p) ;

getch() ;

}
```

**Exemple 3**

```
int  *p1, *p2, x = 10, y = 20 ;

p1=&x ; // p1 pointe sur x

p2=&y ; // p2 pointe sur y

*p1=*p1 + 10 ; // x = 20

*p2 = *p1 + 7 // y =27
```

**4- intérêts**

- Le principal intérêt des pointeurs est qu'avec eux, on peut créer des nouvelles variables indépendantes à tout moment
- Ils permettent de manipuler de façon simple des données pouvant être importantes ( au lieu de passer à une fonction un élément très grand (taille), on pourra par exemple lui fournir un pointeur vers cet élément)
- Les tableaux ne permettent de stocker qu'un nombre fixé d'éléments de même type. En stockant des pointeurs dans les cases d'un tableau, il sera possible de stocker des éléments de taille diverse, et même de rajouter des éléments au tableau en cours d'utilisation (c'est la notion de tableau dynamique qui est très étroitement liée à celle de pointeur)
- Il est possible de créer les structures chaînées, c'est-à-dire comportant des maillons

**5-Opérations sur les pointeurs**

- **Incrémentation**

```
float *p , x=3.12 ;

p=&x ;
```

p++ ; // incrémentation

**Remarque** : l'incrémentation d'un pointeur n'est autorisée que lorsque le pointeur pointe sur un tableau

**Exemple**

Int tab[5]={0,0,0,0,0}, *p ;

P=&tab[0] ; // p pointe sur le tableau

*p = 1 ; // tab[0]=1

P++ ; // p pointe sur tab[1]

*p=2 ;

P=p+1 ; // p pointe sur tab[2]

*p = 10 ; // tab[2]=10

- **Décrémentation**

p-- ; // décrémentation

p=&tab[0] ;// p pointe sur le tableau

**E xemple 1**

int *p1 , *p2 , x=10, y=20 ;

P1=&x ; // p1 pointe sur x

P2=&y ; // p2 pointe sur y

*p1=*p2 + 3 ; // x=23, y=20

*p1 = *p1 + 1 ; // x=24, y=20

P1=p2 ; // p1 et p2 pointent sur y

*p2=12 ; // x=24, y=12

P2++ ; // x=24, y=12 ;

**Exemple 2**

Analyser le programme suivant

Int a[20]={12, 23, 34, 45, 56, 67, 78, 89, 90}, *p ;

P=&a[0] ;

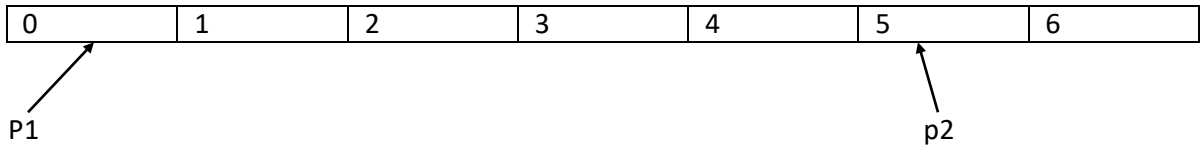
Quelles valeurs ou adresses fournissent les expressions :

1) *p + 2 // a[0] + 2
2) *(p + 2) // a[2]=34
3) &p+1 // case suivante après l'adresse du pointeur
4) &a[4]-3 // &a[1]
5) P+3 // &a[3]
6) &a[7]-p // 7-0=7
7) P + (*p – 10) // &a[2]
8) *[p + *(p + 8) – a[7] ] // a[1]

- **Comparaison de deux pointeurs**

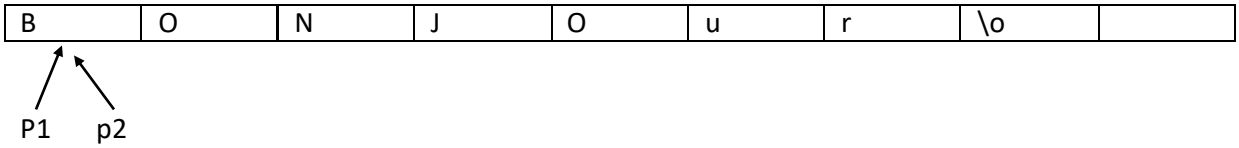
```
Int tab[10] = {0, 1, 2, 3, 0, 0, 0, 1, 1}, *p1, *p2 ;

P1=&tab[0] ;

P2=&tab[9] ;

While(p1<p2)

{

*p1=1; // remplir tout le tableau de 1

P1++ ; // incrémentation

}
```

- **Différence de deux pointeurs pointant sur un même tableau**




P2 – p1 = nombre d'élément du type en question situés entre les deux adresses correspondantes

P2 – p1 représen te le nombre d'élément compris entre p1 et p2 + 1




```
Char ch[10] = {'b', 'o', 'n', 'j', 'o', 'u', 'r', '\o'}, *p1, *p2 ;

P1 =&ch[0] ; // p1 pointe sur le tableau

P2=&ch[0] ; // p2 pointe sur le tableau

While(*p2 !='\o')

P2++ ; // incrémentation
```

Printf(''La longueur de la chaîne est : %p'', p2 – p1) ;

**6-Tableau et pointeur**

On peut optimiser l'utilisation des tableaux et pointeurs en utilisant le fait que le nom d'un tableau est l'adresse de son premier élément.

Exemple

Int t[7], *p ;

P=t ; // p pointe sur le tableau

t ⇔ &t[0] *t ⇔t[0]

t + 1 ⇔ &t[1] *(t + 1) ⇔t[1]

t + i ⇔&t[i] *(t + i) ⇔t[i]

(t + i) représente l'adresse de (i +1) ème élément du tableau

**Application 1**

Placer la valeur 1 dans les cinq éléments d'un tableau

```
#include<stdio.h >
#include<conio.h>
Void main()
{
clrscr();
int I, t[5];
for(i=0;i<5;i++)
*(t + i) = 1;
for(i=0;i<5;i++)
printf(''%d\t'',t[i]);
printf(''\n'');
getch();
}
```

**Application 2**

Ecrire, de deux façons différentes, un programme qui lit 10 nombres entiers dans un tableau avant d'en rechercher le plus grand et le plus petit :

En utilisant uniquement « le formalisme tableau »

En utilisant « le formalisme pointeur », chaque fois que cela est possible

**Résolution**

**a) le formalisme tableau**

```
#include<stdio.h>
#include<conio.h>
#define m 10
Void main()
{
Clrscr() ;
int i, min, max, t[m];
printf(''Donnez %d valeurs \n', m) ;
for(i=0 ;i<m ;i++)
scanf(''%d'',&t[i]) ;
max=min=t[0];
for(i=1;i<m;i++)
{
if(t[i]>max)
max=t[i];
if(t[i]<min)
min=t[i];
}
Printf('' valeur maximale : %d\n'', max);
Printf('' valeur minimale : %d\n'', min) ;
getch() ;
}
```

**b) le formalisme pointeur**

```
#include<stdio.h>
#include<conio.h>
#define m 10
Void main()
{
Clrscr() ;
int i, min, max, t[m];
printf(''Donnez %d valeurs \n', m) ;
for(i=0;i<m ;i++)
scanf(''%d'', t + i) ;
max=min=*t;
for(i=1;i<m;i++)
{
if(*(t + i)>max)
max=*(t + i);
if(*(t + i)<min)
min=*(t + i);
}
Printf('' valeur maximale : %d\n'', max);
Printf('' valeur minimale : %d\n'', min) ;
getch() ;
}
```

**Chapitre 5 : Les fonctions**

**1) Définition**

Une fonction est un sous programme qui réalise une action et qui a pour résultat de fournir une valeur

Toute fonction possède :

- Un nom
- Un type
- Paramètres

**Syntaxe** : type nom_fonction (paramètres)

{

---- // corps de la fonction

}

**2) Quelques fonctions sans paramètres ni valeur de retour**

```
Void titre()
{
printf(''Bonjour'') ;
}
```

**Remarque** : le type void est utilisé si la fonction ne retourne aucune valeur. La fonction titre est appelée par n'importe quelle fonction

**Exemple 1 :**

```
#include<stdio.h>
#include<conio.h>
Void bonjour() // declaration de la fonction
{
Printf('' Bonjour\n'') ;
}
Void main()
{
bonjour() ;// appel de la fonction
```

```
getch() ;
}
```

**Exemple 2** :

```
#include<stdio.h>
#include<conio.h>
Void bonjour()  // declaration de la function
{
Printf(''Bonjour\n '') ;
}
Void coucou()
{
bonjour() ;// appel d'une fonction dans une fonction
printf(''Coucou\n'');
}
Void main()  // programme principal
{
coucou() ; // appel de la fonction
  getch() ;
}
```

**3) Fonction renvoyant une valeur au programme et sans passage d'arguments**

**Exemple :**

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int lance()
{
int test; // variable locale
```

```
test=random(7);
return(test);
}
Void main()

{
int resultat;
randomize(); // initialise avec une valeur aléatoire le générateur interne de nombres aléatoires
resultat = lance() ;
printf(''vous avez obtenu le nombre :%d\n'', resultat) ;
getch() ;
}
```

**4) Fonction avec paramètres sans valeur de retour**

```
void multiplication (int a, int b)
{
Printf('' le resultat de %d et de %d est %d'', a, b, a*b) ;
}
```

**5) Fonction avec paramètres et avec valeur de retour**

```
int multiplication (int a, int b)
{
int c ;
c=a*b ;
return© ;
}
float f (float x)
{
return(x*x + 3*x +1);
}
```

**Exemple :**

```
#include<stdio.h>
#include<conio.h>
int carre(int x) // declaration de la fonction
{
```

```
int r ; // variable locale
r=x*x ;
return (r);
}
Void main ()
{
int n, resultat; // variables locales
printf(''Entrer un nombre :'');
scanf(''%d'',&n) ;
resultat = carre(n) ;
printf(''le carré de %d est :%d'', n, resultat) ;
getch() ;
}
```

**6) Déclaration d'une fonction**

Type_valeur_renvoyée nom_fonction (liste paramètres)

{

----//----

return(expression) ;

}

**7) Structure d'un programme en C**

inclusion des bibliothèques

constantes globales et variables globales

déclaration des en-têtes des fonctions

Void main()

```
{
------//---------
}
```

déclaration des corps des fonctions

**Exemple :**

#include<stdio .h>

#include<conio.h>

int somme (int , int) ;

int produit (int, int) ;

```
void main()
{
clrscr() ;
int x, y ;
printf(''Entrer deux entiers :'') ;
scanf(''%d%d'',&x,&y) ;
printf(''La somme de %d et de %d = %d'', x, y, somme(x, y)) ;
printf('' le produit de %d et de %d = %d'', x,y, produit(x, y)) ;
getch() ;
}
int somme (int x, int y)
{
return(x + y) ;
}
int produit (int x, int y)
{
return(x*y) ;
}
```

**8) Résumé sur les variables et les fonctions**

- Une variable **globale** est déclarée au début du programme et qu'elle est connue de tout le programme. Les variables globales sont initialisées à 0 au début de l'exécution du programme, sauf si on les initialise à une autre valeur
- Une variable **locale** (déclarée au début d'une fonction ou de main() ) n'est connue que de cette fonction ou de main(). Une variable locale est encore appelée **automatique**

Les variables locales ne sont pas initialisées (sauf si on le fait dans le programme) et elles perdent leur valeur à chaque appel à la fonction. On peut allonger la durée de vie d'une variable locale en la déclarant **static** . Lors d'un nouvel appel à la fonction, la variable garde la valeur obtenue à la fin de l'exécution précédente. Une variable **static** est initialisée à 0 lors du premier appel à la fonction

**Exemple :**

int i ; devient static int i ;

**Exercice**

Quelle sera la valeur finale de n si I est déclarée comme variable static, puis comme variable automatique?

```
#include<stdio.h>

#include<conio.h>

int n; // variable globale initialisée à 0

Void  calcul () // declaration de la fonction

{

static int i ; // initialisée à 0

i++ ;

printf(''i=%d\n'',i);

n=n + i;

}

Void main()  // programme principal

{

clrscr();

printf(''n=%d\n'', n);

calcul();

printf('' n=%d\n'', n);

getch();

}
```

## 9) Le passage des paramètres entre fonctions ou entre fonctions et programme principal

En langage C, le passage de paramètres se fait uniquement par adresse. Autrement dit, une fonction ne peut pas modifier la valeur des variables locales à main() ou à une fonction. Elle ne peut modifier que le contenu de l'adresse de cette variable.

### 9-1-Passage des paramètres par valeur

La fonction reçoit des valeurs par des noms des variables et la modification de ces variables n'est pas répercutée à l'extérieur de la fonction

**Exemple : syntaxe incorrect**

```
#include<stdio.h>
#include<conio.h>
void echange (int x, int y)
{
int tampon ;
tampon=x ;
x=y ;
y=tampon ;
}
void main()
{
int a = 5, b=8 ;
echange(a, b) ;
printf(''a=%d\n'',a) ;
printf(''b=%d\n'',b);
getch();
}
```

**9-2-Passage des paramètres par adresse**

Le passage par adresse signifie que l'on passe en argument l'adresse de la variable. La fonction manipulera donc l'adresse de la variable passée en argument. Si la valeur stockée à cette adresse est modifiée par la fonction, la valeur de cette variable sera aussi modifiée à l'extérieur de la fonction. Le passage par adresse se déclare en utilisant l'opérateur *

**Exemple : syntaxe correct**

```
#include<stdio.h>
#include<conio.h>
void echange (int  *x, int *y)
{
```

```
int tampon ;
tampon=*x ;
*x=*y ;
*y=tampon ;
}
void main()
{
Int a = 5, b=8 ;
echange(&a, &b) ;
printf(''a=%d\n'',a) ;
printf(''b=%d\n'',b);
getch();
}
```

**10) Fonctions récursives**

Une fonction récursive est une fonction qui s'appelle elle-même.

**Exemple** : calculer le factorielle d'un nombre entier n

```
#include<stdio.h>
#include<conio.h>
Int fact(int n)
{
if(n==0)
return 1;
else return (n*fact(n-1));
}
Void main ()
{
Clrscr();
```

```
int a = 4;
printf(''%d=%d'', a, fact(a));
getch();
}
```

**11) Fonction retournant un pointeur et pointeur de fonction**

- **Syntaxe** : type *nom_fonction(arguments) // est une fonction qui renvoie un pointeur

**Exemple** :

```
int  *max(int tab[], int taille)
{
Int  i, *grand ;
for(grand=tab ,i=1 ;i<taille ;i++)
If(tab[i]>*grand)
grand=tab + i;
return(grand);
}
```

Cette fonction rend l'adresse du plus grand entier du tableau

**Application :**

```
#include<stdio.h>
#include<conio.h>
int  *max(int tab[], int taille)
{
Int  i, *grand ;
for(grand=tab ,i=1 ;i<taille ;i++)
If(tab[i]>*grand)
grand=tab + i;
return(grand);
}
```

```
Void main()
{
clrscr();
Int tab[10], i;
printf(''Entrez les éléments du tableau :\n'');
for(i=0 ;i<10 ;i++)
scanf(''%d'',&tab[i]) ;
printf(''l'adresse du plus grand élément du tableau est: %p\n'', max(tab, 10)) ;
getch() ;
}
```

- **Syntaxe** : type (*nom_fonction)(arguments) // est un pointeur sur une fonction

**Exemple** :

Evaluer la valeur de l'expression : E = 1 + 1/1 ! + ½ ! + … + 1/n !

**Mini projet : Analyser le projet suivant**

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define n 10
int tab[n];
// fonction initialization
Void  initialization(int tab[])
{
for(int i =0;i<n;i++)
tab[i]=0;
}
// fonction initrandom
Void initrandom(int tab[])
```

```
{
randomize();
for(int i=0;i<n;i++)
tab[i]=random(101);
}
//fonction saisie
void saisie (int tab[])
{
        for (int I = 0;i<n;i++)
            {
                printf(''Element numéro %d :\n'',i);
                scanf(''%d'',&tab[i]);
            }
}
//fonction affiche
Void affiche (int tab[])
{
for (int i =0;i<n;i++)
printf(''%d\t'',tab[i]);
}
//fonction menu
Void menu()
{
int choix;
do
    {
      clrscr() ;
```

```
printf(''---MENU GENERAL---\n'');
printf(''1: INITIALISATION\n'');
printf(''2: INITRANDOM\n'');
printf(''3: SAISIE\n'');
printf(''Entrez votre choix:\n'');
scanf(''%d'',&choix);
switch(choix)
   {
    case 1 : {  clrscr();
                initialisation (tab);
                printf(''MATRICE NULLE\n'');
                affiche (tab);
             }
            break;
    case 2 : {  clrscr();
                initrandom (tab);
                printf(''MATRICE RANDOM\n");
                affiche (tab);
             }
            break;
      case 3 :{   clrscr();
                saisie (tab);
                printf(''MATRICE SAISIE\n'');
                affiche(tab);
             }
            break;
getch();
```

```
                }
            }
while(choix !=4) ;
getch() ;
} // fin de la fonction menu
Void main() // programme principal
{
menu() ;
}
```

**Chapitre 6 : LES CHAINES DE CARACTERES**

En langage C, les chaînes de caractères sont des **tableaux de caractères**. Leur manipulation est donc analogue à celle d'un tableau à une dimension:

**6-1- Déclaration***:* **char nom[dim];** ou bien **char *nom;**

**nom = (char*)malloc(dim);**

Exemple: **char texte[dim];** ou bien **char *texte;**

**texte = (char*)malloc(10);**

Le compilateur réserve (dim-1) places en mémoire pour la chaîne de caractères: En effet, il ajoute toujours le caractère **NUL ('\0')** à la fin de la chaîne en mémoire.

**6-2 -Affichage à l'écran***:*

On peut utiliser la fonction **printf** et le format %s:

**char texte[10] = « BONJOUR »;**

**printf("VOICI LE TEXTE: %s\n",texte);**

On utilisera si possible la fonction **puts** non formatée:

**puts(texte);** est équivalent à **printf("%s\n",texte);**

**Saisie** *:* On peut utiliser la fonction **scanf** et le format %s. Une chaîne étant un pointeur, on n'écrit pas le symbole &. On utilisera de préférence la fonction **gets** non formatée.

La fin de la saisie d'une chaîne par la touche Entrée du clavier

**char texte[10];**

**printf("ENTRER UN TEXTE: ");**

**scanf("%s",texte);** est équivalent à **gets(texte);**

**Remarque** : Pour saisir une chaîne de type "il fait beau", il faut utiliser gets. En effet scanf ne prend pas en compte les espaces par contre gets prend en compte les espaces

A l'issue de la saisie d'une chaîne de caractères, le compilateur ajoute '\0' en mémoire après le dernier caractère.

**Application 1**

Analyser le programme suivant :

```
#include<stdio.h>

#include<conio.h>

#include<string.h>

Int main()

{

char nom[20], prenom[20], ville[25] ;

printf(‘’ Quelle est votre ville ?’’) ;

gets(ville) ;

printf(‘’Donnez votre nom et votre prenom :’’) ;

scanf(‘’%s%s’’, nom, prenom) ;

printf(‘’Bonjour cher %s %s qui habitez à ‘’, prenom, nom) ;

puts(ville) ;

getch() ;

return 0 ;

}
```

**Application 2**

```
#include<stdio.h>

#include<conio.h>
```

```
#include<string.h>
int main()
{
char tab[20] ;
int i ;
printf(‘’ Entrez une chaîne :’’) ;
gets(tab) ;
puts(‘’La chaîne à l’envers est:\n”) ;
for (i=strlen(tab)-1 ;i>=0 ;i--)
printf(‘’%c”, tab[i]) ;
getch() ;
return 0 ;
}
```

Application 3

Analyser le programme suivant

```
#include<stdio.h>
#include<conio.h>
Void main()
{
char *adr ;
adr=’’bonjour’’ ;
while(*adr)
{
Printf(‘’%c’’,*adr) ;
Adr++ ;
}
Getch() ;
```

}

Commentaire :

Char *adr ; /* reserve simplement l'emplacement pour un pointeur sur un caractère ( ou sur une suite de caractères) */

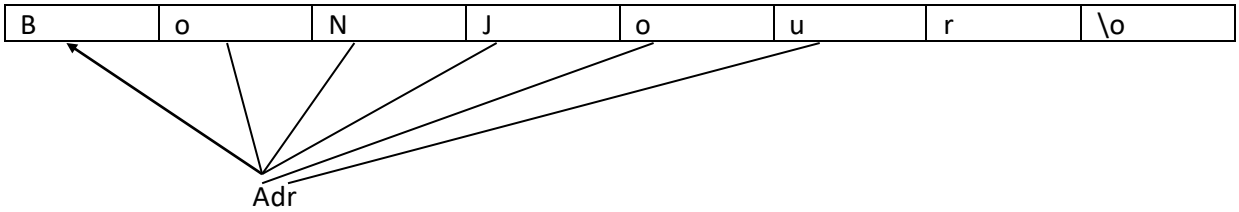
En ce qui concerne la constante :

''bonjour'' ; /* le compilateur a crée en mémoire la suite d'octets correspondants */

Mais dans l'affectation :

Adr=''bonjour'' ; /* la notation bonjour a comme valeur, non pas la valeur de la chaîne elle-même, mais son adresse */

Voici un schéma illustrant ce phénomène ; la flèche en trait plein correspondant à la situation après l'exécution de l'affectation : adr = ''bonjour'' ; les autres flèches correspondent à l'évolution de la valeur de adr , au cours de la boucle




**Initialisation de tableaux de caractères**

Char tab[20] = ''bonjour'' ; // char tab[20]={ 'b', 'o', 'n', 'j', 'o', 'u', 'r', '\0'} ;

**Initialisation de tableaux de pointeurs sur des chaînes**

Nous avons vu qu'une chaîne constante était traduite par le compilateur en une adresse que l'on pouvait, par exemple, affecter à un pointeur sur une chaîne . Cela peut se généraliser à un tableau de pointeur, comme suit :

Char *jour[7]={''lundi'', ''mardi'', ''mercredi'', ''jeudi'', ''vendredi'' , ''samedi'', ''dimanche''} ;

/* cette déclaration réalise à a fois la création de 7 chaînes constantes correspondant aux 7 jours de la semaine et à l'initialisation du tableau jour avec les 7 adresses de ces 7 chaînes */

Attention :

Printf(''%s'', jour[0]) ; //affiche lundi

Printf(''%c'', *jour[0]) ; //affiche l

**Exemple** :

#include<stdio.h>

#include<conio.h>

```
#include<string.h>

Int main()

{

Char *jour[7]={''lundi'', ''mardi'', ''mercredi'', ''jeudi'', ''vendredi" , ''samedi'', ''dimanche"} ;

Int i ;

Printf(''Donnez un entier entre 1 et 7 :'') ;

Scanf(''%d'',&i) ;

Printf(''le jour numéro %d de la semaine est : %s'', i, jour[i-1]) ;

Getch() ;

Return 0 ;

}
```

**Application 4**

Ecrire un programme qui lit un numéro de téléphone sous la forme (242)066438523 et extrait l'indicatif international dans une chaîne et qui l'affiche à l'écran

**Résolution**

```
#include<stdio.h>

#include<conio.h>

#include<string.h>

Void main()

{

Char tel[20], indicatif[5] ;

Int i=1, j=0 ;

Printf(''Entrez le numéro de téléphone:\n") ;

Gets(tel) ;

While(tel[i] !=')')

{
```

```
indicatif[j]=tel[i] ;

i++ ;

j++ :

}

indicatif[j]='\o' ; // très important

printf('' Voici l'indicatif du pays : %s'',indicatif) ;

getch() ;

}
```

**Exercice 1**

Saisir une chaîne de caractères, afficher les éléments de la chaîne et leur adresse (y compris le dernier caractère '\0').

```
#include<stdio.h>

#include<conio.h>

#define n 25

Void main()

{

Char texte[n] ;

Int i ;

Printf(''Entrez une chaîne:\n'') ;

Gets(texte) ;

For(i=0 ;i<=n ;i++)  // y compris le caractère de fin de chaîne

{

Printf(''le caractère %d est %c'', i, texte[i]) ;

Printf(''Son adresse est %p'', texte +i) ;

}

Getch() ;

}
```

**Exercice 2**

Saisir une chaîne de caractères. Afficher le nombre de e et d'espaces de cette chaîne.

**6-3- Fonctions permettant la manipulation des chaînes**

Les bibliothèques fournies avec les compilateurs contiennent de nombreuses fonctions de traitement des chaînes de caractères. En BORLAND C++, elles appartiennent aux bibliothèques **string.h ou stdlib.h.** En voici quelques exemples:

**Générales (string.h)**

void *strcat(char *chaine1,char *chaine2) concatène les 2 chaînes, résultat dans chaine1,

renvoie l'adresse de chaine1.

Exemple :

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
Void main()
{
Char ch1[100], ch2[20] ;
Printf(''Entrez la première chaîne :'') ;
Gets(ch1) ;
Printf(''Entrez la deuxième chaîne :'') ;
Gets(ch2) ;
Strcat(ch1,ch2) ;
Printf('' la chaîne concatenée est : %s'', ch1) ;
Getch() ;
}
```

**Remarque** :

Strncat(ch1, ch2, lmax) // lmax= nombre de caractère de ch2 à concaténer

int strlen(char *chaine) renvoie la longueur de la chaine ('\0' non comptabilisé).

Exemple : strlen(‘’bonjour’’) vaudra 7

void *strrev(char *chaine) inverse la chaîne et, renvoie l'adresse de la chaine inversée.

**Application** :

Ecrire un programme qui affiche une chaîne à l’envers

**Comparaison (string.h)**

int strcmp(char *chaine1,char *chaine2) renvoie un nombre:

- positif si la chaîne1 est supérieure à la chaine2 (au sens de l'ordre alphabétique c.à.d. chaîne1 arrive avant chaîne2 )

- négatif si la chaîne1 est inférieure à la chaîne2

- nul si les chaînes sont identiques.

Exemple :

Char *ch1, *ch2 ;

Ch1=’’Bonjour’’ ;

Ch2=’’bonj’’ ;

codeB =66 et codeb=99

ch2>ch1 donc strcmp(ch1,ch2)<0 /* la chaîne ch2 arrive après la chaîne ch1 au sens de l’ordre défini par le code de caractère)

**NB** : il n’y a pas de type chaîne de caractère en C. donc on ne peut jamais transmettre la valeur d’une chaîne en argument d’une fonction mais seulement son adresse, ou plus précisement un pointeur sur son premier caractère.

**Copie (string.h)**

void *strcpy(char *chaine1,char *chaine2)

recopie chaine2 dans chaine1 et renvoie l'adresse de chaîne1.

**Remarque** : la fonction strcnpy limite la recopie au nombre de caractère précisés dans le prototype suivant :

Strcnpy(ch1, ch2, lmax)

Exemple : Analyser le programme suivant

#include<stdio.h>

```
#include<conio.h>
#include<string.h>
Void main()
{
Char ch1[10]=''xxxxxxxxxx'' ;
ch2[10] ;
Printf(''donnez un mot :'') ;
Gets(ch2) ;
Strncpy(ch1,ch2,5) ;
Printf('' :%s'', ch1) ;
Getch() ;
}
```

**Recopie (string.h)**

Ces fonctions renvoient l'adresse de l'information recherchée en cas de succès, sinon le pointeur NULL (c'est à dire le pointeur de valeur 0 ou encore le pointeur faux)

void *strchr(chaine,caractère) recherche le caractère dans la chaîne.

void *strrchr(chaine,caractère) idem en commençant par la fin.

void *strstr(chaîne,sous-chaîne) recherche la sous-chaine dans la chaîne.

**Exercice 2**

Saisir un texte. Afficher le nombre de e de ce texte.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#define lettre 'e'
Void main()
{
Char texte[120], *adr ;
```

```
Int nlettre = 0 ;
Printf(''Donnez un texte \n'') ;
Gets(texte) ;
adr=texte ; // adr pointe sur le tableau
while(adr=strchr((adr,lettre))
{
nlettre++ ;
adr++ ;
}
Printf(''Le texte comporte %d fois le caractère %c'', nlettre,lettre) ;
Getch() ;
}
```

**Conversions (stdlib.h)**

int atoi(char *chaîne) convertit la chaîne en entier

float atof(char *chaine)convertit la chaîne en réel

exemple:
```
printf("ENTRER UN TEXTE: ");
gets(texte);
n = atoi(texte) ;
printf("%d",n); /* affiche 123 si texte vaut "123" */
                /* affiche 0 si texte vaut "bonjour" */
```

void *itoa(int n,char *chaîne,int base) convertit un entier en chaîne:

base: base dans laquelle est exprimé le nombre,

cette fonction renvoie l'adresse de la chaîne.

exemple: itoa(12,texte,10);  texte vaut "12"

Pour tous ces exemples, la notation void* signifie que la fonction renvoie un pointeur (l'adresse de l'information recherchée), mais que ce pointeur **n'est pas typé.** On peut ensuite le typer à l'aide de l'opérateur cast .

Exemple*:*  int *adr;

char texte[10] = "BONJOUR";

adr = (int*)strchr(texte,'O');

**chapitre 7 : LES STRUCTURES**

**7-1- Définition**

Ce sont des collectes d'info rmation regroupées en une entité logique. Il faut définir le type de la structure

**Syntaxe :**

```
struct nom_de_structure {
                          type_de_donnee nom1;
                          type_de_donnee nom2;
                         }; /* ne pas oublier le point-virgule ! */
```

ou bien

```
typedef struct {
    type_de_donnee nom1;
    type_de_donnee nom2;
} type_de_structure; /* ne pas oublier le point-virgule ! */
```

**Exemple 1** : créer la structure point

```
struct point
{
float abs ;
float ord ;
} ;
```

**Exemple 2 :** Créer la structure personne

```
{
char nom[30] ;
char adresse[300] ;
long matricule ;
int age ;
} ;
```

**Exemple 3 :** Créer la structure date

```
struct date
{
int jour ;
int mois ;
```

int année ;
} ;

**7-2- Déclaration**

struct nom_de_structure nom_de_variable ;

ou bien

type_de_structure nom_de_variable ;

**Exemple :** créer la structure point

```
struct point
{
float abs ;
float ord ;
} ;
Point v ; // v est une variable de type point
Ou bien
typedef struct
{
float abs ;
float ord ;
} point ;
Point v ; // v est une variable de type point
```

**7-3- Utilisation**

pour accéder à un champ d'une structure, c'est l'opérateur '.' qui est utilisé ou l'opérateur '->' dans le cas d'une variable pointeur sur une structure.

**Exemple 1**

Calculer la norme à l'origine d'un point M quelconque

```
#include<stdio.h>

#include<conio.h>

#include<math.h>

Struct point

{

Float x ;

Float y ;
```

```
} ;

Int main()

{

struct point  V ;

float x,y ;

printf(‘’ Entrez les coordonnées du point :’’) ;

scanf(‘’%f%f’’,&x,&y) ;

V.x=x ;

V.y=y ;

Printf(‘’Norme = %f’’,sqrt((v.x)*(v.x) + (v.y)*(v.y))) ;

getch() ;

}
```

**Exemple 2**

```
#include<stdio.h>

#include<conio.h>

struct date
{
int jour ;
int mois ;
int année ;
} ;
Int main()
{
struct date hier ;

hier.jour = 30;
 hier.mois = 5 ;
hier.année  = 1986;
printf(‘’Hier le %d %d %d \n’’, hier.jour , hier.mois, hier.année) ;
getch() ;
}
```

**Exemple 3**

```
#include<stdio.h>

#include<conio.h>

struct date
{
```

```
int jour ;
int mois ;
int année ;
} ;
Int main()
{
struct date hier , *x;
x=&hier ;
hier->jour = 30 ;
hier->mois = 5 ;
hier->année = 1986 ;
printf(''Hier le %d %d %d \n'', hier->jour , hier->mois, hier->année) ;
getch() ;
}
```

**Exemple 4**

```
#include <stdio.h>
#include <string.h>
struct salarie {
  char nom[30] ;
  float salaire ;
} ;
salarie dIOP, *NDIAYE, FATOU ;
int main(void) {
  durant = new salarie ;
  strcpy(DIOP.nom,"DIOP Emile") ;
  ...
  DIOP.salaire = NDIAYE->salaire+1000 ;
  Printf("le salaire de %s est %f" , diop.nom ,diop.salaire ) ;
  }
```

**7-4- Tableaux des structures**

Exemple :

```
Struct point

{

Char nom ;

Int x ;

Int y ;

} ;
```

Struct point courbe[50] ;// courbe est un tableau de 50 éléments de type point

**Initiation Partielle de notre variable courbe**

```
Struct point courbe[50]={{'A',10,15},{'M',12,5}} ;
```

**7-5- Structures comportant d'autres structures**

```
struct date
{
int jour ;
int mois ;
int année ;
} ;
Struct personne
{
char nom[30] ;
char prenom[30] ;
float heure[31] ;
struct date date_embauche ;
}employe ;
```

**Exemple :**

Employe.date_embauche.année représente l'année d'embauche correspondant à la structure employe. Il s'agit d'une valeur de type int

**7-6- Transmission d'une structure en argument d'une fonction**

- **Transmission de la valeur d'une structure**

Exemple :

```
#include<stdio.h>

#include<conio.h>

Struct enreg

{ int a ;

Float b ;

} ;

Void fct(struct enreg s)

{

s.a = 0 ; s.b = 1 ;

printf(''\n dans fct : %d %e'', s.a , s.b) ;

}

Int main()

{
```

```
Struct enreg x ;

x.a=1 ;

x.b=12.5 ;

printf(''\n avant appel fct : %d %e'', x.a, x.b) ;

fct(x) ;

printf(''\n au retour dans main : %d %e'', x.a, x.b) ;

getch() ;

}
```

- **Transmission de l'adresse d'une structure : l'operateur ->**
- Exemple :

```
#include<stdio.h>

#include<conio.h>

Struct enreg

{ int a ;

Float b ;

} ;

Void fct(struct enreg  *s)

{

s->a = 0 ; s->b = 1 ;

printf(''\n dans fct : %d %e'', s->a , s->b) ;

}

Int main()

{

Struct enreg x ;

x.a=1 ;

x.b=12.5 ;

printf(''\n avant appel fct : %d %e'', x.a, x.b) ;

fct(&x) ;
```

```
printf(''\n au retour dans main : %d %e'', x.a, x.b) ;

getch() ;

}
```

**chapitre 8 : MANIPULATION DES FICHIERS C**

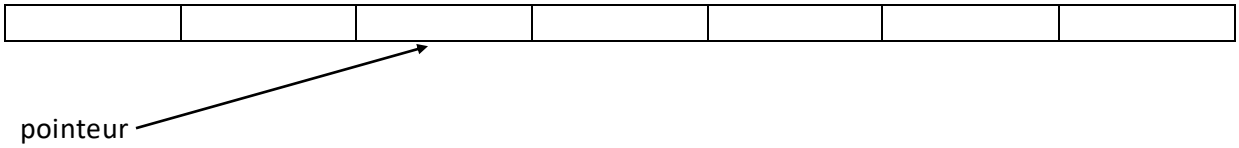
Opérations possibles avec les fichiers: Créer - Ouvrir - Fermer - Lire - Ecrire - Détruire - Renommer. La plupart des fonctions permettant la manipulation des fichiers sont rangées dans la bibliothèque standard STDIO.H, certaines dans la bibliothèque IO.H pour le BORLAND C++.

**1- Définition**

Un fichier est un ensemble des données situées sur les mémoires de masse(disque dur, disquette, clé USB)

En langage C, un fichier est une suite d'octets. Les informations contenues dans le fichier ne sont pas forcément de même type (un char, un int , une structure ---)

Un pointeur fournit l'adresse d'une information quelconque




on distingue généralement deux types d'accès :

- Accès séquentiel
  On accède à une cellule quelconque en se déplaçant (via un pointeur ) depuis la cellule de départ
- Accès direct
  On peut directement accéder à une cellule

Il existe d'autre part deux façons de coder les informations stockées dans un fichier :

**Fichiers binaires** :

Les informations sont codées telles que ce sont en général des fichiers . ils ne sont pas listables

**Fichiers texte** :

Les informations sont codées en ASCII. Ces fichiers sont listables. Le dernier octet de ces fichiers est EOF (caractère ASCII spécifique)

2- **Déclaration d'un fichier**

```
FILE *fichier;   /* majuscules obligatoires pour FILE */
```

On définit un pointeur. Il s'agit du pointeur représenté sur la figure du début de chapitre. Ce pointeur fournit l'adresse d'une cellule donnée.

La déclaration des fichiers doit figurer AVANT la déclaration des autres variables.

3- **Les opérations sur les fichiers**

*- Ouverture:* **FILE *fopen(char *nom, char *mode);**

On passe donc 2 chaînes de caractères

nom: celui figurant sur le disque, exemple: « a :\toto.dat »

mode (pour les fichiers TEXTES) :

« r » lecture seule

« w » écriture seule (destruction de l'ancienne version si elle existe)

« w+ » lecture/écriture (destruction ancienne version si elle existe)

« r+ » lecture/écriture d'un fichier existant (mise à jour), pas de création d'une nouvelle version.

« a+ » lecture/écriture d'un fichier existant (mise à jour), pas de création d'une nouvelle version, le pointeur est positionné à la fin du fichier.

mode (pour les fichiers BINAIRES) :

« rb » lecture seule

« wb » écriture seule (destruction de l'ancienne version si elle existe)

« wb+ »lecture/écriture (destruction ancienne version si elle existe)

« rb+ » lecture/écriture d'un fichier existant (mise à jour), pas de création d'une nouvelle version.

« ab+ »lecture/écriture d'un fichier existant (mise à jour), pas de création d'une nouvelle version, le pointeur est positionné à la fin du fichier.

**A l'ouverture, le pointeur est positionné au début du fichier (sauf « a+ » et « ab+ »)**

Exemple : **FILE *fichier ;**

**fichier = fopen(« a :\toto.dat », « rb ») ;**

*- Fermeture:* **int fclose(FILE *fichier);**

Retourne 0 si la fermeture s'est bien passée, EOF en cas d'erreur.

Il faut toujours fermer un fichier à la fin d'une session. mode (pour les fichiers TEXTE) :

Exemple : **FILE *fichier ;**

**fichier = fopen(« a :\toto.dat », « rb ») ;**

**/* Ici instructions de traitement */**

**fclose(fichier) ;**

*- Destruction:* **int remove(char *nom);**

Retourne 0 si la fermeture s'est bien passée.

Exemple : **remove(« a :\toto.dat ») ;**

*- Renommer:* **int rename(char *oldname, char *newname);**

Retourne 0 si la fermeture s'est bien passée.

*- Positionnement du pointeur au début du fichier:* **void rewind(FILE *fichier);**

*- Ecriture dans le fichier:*

**int putc(char c, FILE *fichier);**

Ecrit la valeur de c à la position courante du pointeur , le pointeur avance d'une case mémoire.

Retourne EOF en cas d'erreur.

Exemple : **putc('A', fichier) ;**

**int putw(int n, FILE *fichier);**

Idem, n de type int, le pointeur avance du nombre de cases correspondant à la taille d'un entier (4 cases en C standard).

Retourne n si l'écriture s'est bien passée.

**int fputs(char *chaîne, FILE *fichier);** idem avec une chaîne de caractères, le pointeur avance de la longueur de la chaine ('\0' n'est pas rangé dans le fichier).

Retourne EOF en cas d'erreur.

Exemple : **fputs(« BONJOUR ! », fichier) ;**

**int fwrite(void *p,int taille_bloc,int nb_bloc,FILE *fichier);** p de type pointeur, écrit à partir de la position courante du pointeur fichier nb_bloc X taille_bloc octets lus à partir de l'adresse p. Le pointeur fichier avance d'autant.

Le pointeur p est vu comme une adresse, son type est sans importance.

Retourne le nombre de blocs écrits.

Exemple: taille_bloc=4 (taille d'un entier en C), nb_bloc=3, écriture de 3 octets.

```
int tab[10] ;
fwrite(tab,4,3,fichier) ;
```

**int fprintf(FILE *fichier, char *format, liste d'expressions);** réservée plutôt aux fichiers ASCII.

Retourne EOF en cas d'erreur.

Exemples: **fprintf(fichier,"%s","il fait beau");**

```
fprintf(fichier,%d,n);
fprintf(fichier,"%s%d","il fait beau",n);
```

Le pointeur avance d'autant.

*- Lecture du fichier:*

**int getc(FILE *fichier);** lit 1 caractère, mais retourne un entier n; retourne EOF si erreur ou fin de fichier; le pointeur avance d'une case.

Exemple: **char c ;**

**c = (char)getc(fichier) ;**

**int getw(FILE *fichier);** idem avec un entier; le pointeur avance de la taille d'un entier.

Exemple: **int n ;**

**n = getw(fichier) ;**

**char *fgets(char *chaine,int n,FILE *fichier);** lit n-1 caractères à partir de la position du pointeur et les range dans chaine en ajoutant '\0'.

**int fread(void *p,int taille_bloc,int nb_bloc,FILE *fichier);** analogue à fwrite en lecture.

Retourne le nombre de blocs lus, et 0 à la fin du fichier.

**int fscanf(FILE *fichier, char *format, liste d'adresses);** analogue à fprintf en lecture.

*- Gestion des erreurs:*

**fopen** retourne le pointeur NULL si erreur (Exemple: impossibilité d'ouvrir le fichier).

**fgets** retourne le pointeur NULL en cas d'erreur ou si la fin du fichier est atteinte.

la fonction **int feof(FILE *fichier)** retourne 0 tant que la fin du fichier n'est pas atteinte.

la fonction **int ferror(FILE *fichier)** retourne 1 si une erreur est apparue lors d'une manipulation de fichier, 0 dans le cas contraire.

*Fonction particulière aux fichiers à acces direct:*

**int fseek(FILE *fichier,int offset,int direction)** déplace le pointeur de offset cases à partir de direction.

Valeurs possibles pour direction:

0 -> à partir du début du fichier.

1 -> à partir de la position courante du pointeur.

2 -> en arrière, à partir de la fin du fichier.

Retourne 0 si le pointeur a pu être déplacé;

**Exercice 1**

Programme qui permet de créer un fichier d'entier et d'afficher ces entiers.

**Exercice 2**
Programme qui permet de créer un fichier d'étudiant (numéro ,nom ,prénom, age) et d'afficher ces étudiants.

**Exercice 3 : Gestion des comptes épargnes**
- Créer un fichier pour les clients en mode ajout
  Client(nom, adresse, tel, codecompte, solde)
- Créer un index de recherche par code
- Fournir les fonctions créditer et débiter + mise à jour du fichier client
- Fournir une fonction affiche qui permet d'afficher le solde

*SEANCE 2*

---

## *Chapitre 9 : Allocation de mémoire dynamique : Liste chaînée*

### Réservation et libération

L'utilisation des pointeurs permet d'allouer de la mémoire sans connaître la taille nécessaire au moment de la compilation. L'instruction malloc permet de réserver une certaine quantité de mémoire. Cette zone mémoire est ensuite accessible par un pointeur. L'exemple suivant permet d'allouer une zone de 5 entiers

**Exemple**

```
int * ptr;
```

ptr = (int*) malloc(5*sizeof(int));

On peut accéder à cette zone en utilisant la notation de tableau :

ptr[0] = 4;

ptr[4] = 1;

ou la notation de pointeur :

*ptr = 4;

*(ptr+4) = 1;

L'exemple suivant propose une fonction AllouerPersonne() qui renvoie un pointeur sur un enregistrement de type TPersonne qui a été alloué.

**Exemple**

```
typedef
{
  char nom[30];
  int age;
} TPersonne;

TPersonne * AllouerPersonne(char * nom, int age)
{
  TPersonne * ptr;
  ptr = (TPersonne*)malloc(sizeof(TPersonne));
  return ptr;
}
```

La libération de la mémoire allouée dynamiquement se fait par la commande free comme illustré dans l'exemple suivant :

**Exemple**

```
voi main()
```

```
{
  TPersonne * p;
  p = AllouerPersonne("John Doe",33);
  if (p==NULL)
    printf("Erreur de memoire !\n");
  else
  {
    printf("%s (%d) a ete cree.\n",p->nom, p->age);
    free(p); /* liberation de la memeoire */
  }
}
```

**Remarques** .Si la réservation mémoire n'a pas pu se faire, malloc renvoie la valeur NULL.

**Structures Dynamiques**

L'exemple suivant montre la création dynamique d'une structure.

**Exemple**

```
struct livre
{
  int disponible;
  int numero;
};


void main()
{
  int i;


  struct livre * original;
  struct livre * copies;
```

```
 original = (struct livre*)malloc(sizeof(struct livre));
 copies = (struct livre *)malloc(10*sizeof(struct livre));

 original->numero = 100;
 original->disponible = 1;
 for (i=0;i<10;i++)
 {
  copies[i].numero = 1000+i;
  copies[i].disponible = 1;
 }

 /* ... */

 free(original);
 free(copies);
}
```

**Remarques**

La variable pointeur original pointe sur une zone mémoire de type struct livre .

La variable pointeur copies pointe sur une zone mémoire de 10 structures de type struct livre .
L'accés à chaque enregistrement de copie se fait grâce à la notation de type tableau ( copies[i] ).

**Remarques**

La fonction calloc permet d'allouer de la mémoire et d'en initialiser le contenu.

Pour créer une liste chaînée il faudra ajouter dans la structure un champs qui indique le pointeur suivant .

**EXERCICES RECAPITULATIFS TP N°5**

- Programme qui permet de créer une liste d'entier et d'afficher ces entiers.
- Programme qui permet de créer une liste de livre (numéro ,titre ,disponible) et d'afficher ces livres.

**Exemple**

```
struct livre

{char titre[30] ;

 int disponible;

 int numero;

struct livre * suivant ;

};


typedef struct livre  * Liste;
```

liste L ;

L=NULLl ; liste initialisée à vide

Aie , on donne pas tous aux enfants voir si possible cours , Livre , recherche sur Internet ce n'est plus l'initialisation au langage. Merci bonne continuation dans l'approfondissement de vos connaissances en programmation avancées et courage