Mourad LOUKAM

INTRODUCTION A LA PROGRAMMATION EN LANGAGE C

TABLE DES MATIERES

CHAPITRE I : GENERALITES

1.1 Introduction :

Le langage C est un langage de programmation connu pour son efficacité, économie et portabilité. C'est un langage qui se situe entre les langages dits évolués (comme PASCAL) qui utilise un vocabulaire proche du langage naturel, et le langage machine. Ses caractéristiques le rendent très indiqué pour le développement des applications « orientées systèmes ».

En réalité il existe plusieurs versions du langage C (Turbo C, Quick C, C++, Visual C++, ...). Dans le cadre de ce document, nous développerons en priorité les objets du langage C standard (norme ANSI), en faisant référence, toutefois, le cas échéant aux nouvelles possibilités du langage C++.

1.2 Mise en œuvre d'un programme C :

La mise en œuvre d'un programme C, quelque soit l'environnement choisi, passe par quatre étapes importantes :

- La saisie
- La compilation
- L'édition de lien
- L'exécution

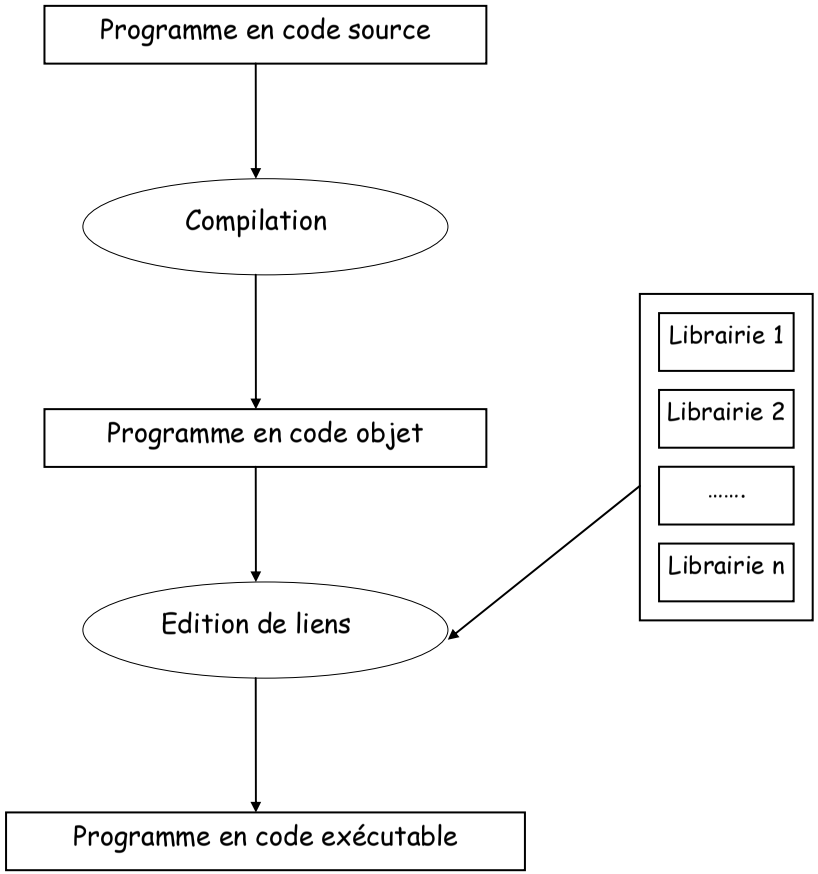
La saisie consiste à saisir le code du programme en utilisant un éditeur , généralement fourni par l'environnement du langage.

La **compilation** consiste à lancer une certaine « vérification » du programme. Le compilateur passe au peigne fin le programme et recherche les différents types d'erreurs : les erreurs lexicales, les erreurs syntaxiques et les erreurs sémantiques. Si le programme comporte une quelconque erreur, un message d'erreur et de diagnostic est affiché au programmeur. Celui-ci doit alors apporter les corrections nécessaires et doit refaire la compilation.

Lorsque le programme est épuré de toutes ses erreurs, le compilateur génère alors le **code objet** du programme.

Les programmes utilisent généralement des références externes fournies par le langage de programmation , qui sont habituellement contenues dans des librairies. **L'édition des liens** permet ainsi de résoudre ces références, et produit en cas de succès le code exécutable du programme.

Ces différentes étapes sont résumées par la figure suivante :



MISE EN ŒUVRE D'UN PROGRAMME

CHAPITRE II : LES OBJETS DE BASE DU LANGAGE C

Ce chapitre a pour objet de passer en revue les objets de base de C : les identificateurs, les types, les variables, les expressions et les opérateurs.

2.1 Les identificateurs :

Un identificateur est un « nom » qui désigne un objet du langage C. Exemples :

var1
montant_General
x2
...

A noter l'obligation imposée par la plupart des langages de programmation de commencer les identificateurs par une lettre alphabétique.

2.2 Les types de données :

Un type de données est un domaine de données particulier , comme le type des entiers, le type des réels, le type des caractères, ... etc. Pour le langage C, les types de base sont :

Type	*Domaine couvert*
Int	Domaine des entiers (..., -5, -4, ..., 0, +1, +2, +3, ...)
Float	Domaine des nombres réels (3.5, 2.55, ...)
Double	Domaine des nombres réels en double précision
Char	Domaine des caractères ('a', 'b', ..., '1', '2', ..., '*', '+', ...)
Long	Domaine des entiers longs. Ils sont représentés par des mots de 32 bits (04 octets)
Short	Domaine des entiers courts. Ils sont représentés par des mots de 16 bits (02 octets).
Unsigned	Domaine des entiers non signés

A partir de ces types de base, il est possible de construire des types de données personnalisés aussi complexes que l'on veut.

2.3 Les constantes :

Une constante est un objet dont la valeur est fixe et connue. Une constante peut être entière , flottante ou de type caractères.

Exemples : 3.14, 25, "Langage C", ...

A remarquer que les constantes de type caractère est toujours délimitée par le symbole double-quote (").

2.4 Les variables :

Une variable en informatique désigne une zone mémoire référencée par un identificateur. Cette zone en mémoire peut prendre une valeur variable.

On peut attribuer une valeur à une variable grâce à deux méthodes différentes : l'affectation et la lecture.

L'affectation consiste à attribuer directement une valeur à la variable. Elle est notée par le symbole (=). Exemple :

x=15 , attribue la valeur 15 à x.
x=y+z , attribue le résultat de l'expression y+z à x. Si x contenait une valeur avant cette affectation, cette valeur serait perdue.

x=x+15 , affecte à x l'ancienne valeur de x ajoutée à la valeur 15.

Avant de pouvoir utiliser une variable, il est obligatoire de ***la déclarer*** au préalable.

« ***Déclarer une variable*** » consiste à lui préciser son type de données. Exemples de déclarations :

int x, y, z , déclare trois variables entières de type entier.
float a1, a2 , déclare deux variables A1 et A2 de type réel.
char c1 , déclare une variable C1 de type caractère.

Il est possible de donner une valeur initiale aux variables déclarées. Exemples :

```
int  i=54 ;
int  j=34 ; k=12 ;
```

2.5 Les opérateurs :

C utilise essentiellement trois types d'opérateurs qui sont : *les opérateurs mathématiques*, *les opérateurs relationnels* et *les opérateurs logiques.*

Les opérateurs mathématiques :

Opérateur	*Sens*
+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Reste de la division
++	Incrémentation
- -	Décrémentation

Les opérateurs relationnels :

Opérateur	*Sens*
<	Inférieur à
>	Supérieur à
= =	Egal à
!=	Différent de
<=	Inférieur ou égal
>=	Supérieur ou égal

Les opérateurs logiques :

Opérateur	*Sens*
&&	Et
||	Ou
!	Non

Les opérateurs relationnels fournissent des résultats pouvant avoir deux valeurs possibles :

1 (Vrai) ou 0 (Faux).

2.6 Les expressions :

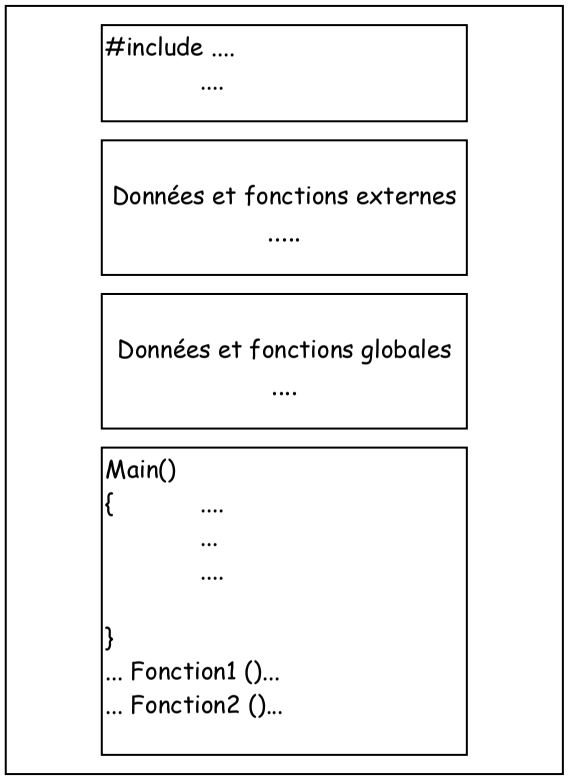
Une expression est constituée par la combinaison de champs, variables mémoires, constantes, fonctions et d'opérateurs.

Exemples :

nom
x = (y +z) /t
prix *(1 + taxe)

2.7 Structure générale d'un programme C :

En Général, un programme C est structuré en quatre parties essentielles :



STRUCTURE GENERALE D'UN PROGRAMME C

La partie ‹include› : permet de déclarer tous les fichiers à inclure dans le programme en cours, lors de la compilation.

La partie « variables et fonctions externes » permet d'informer le compilateur sur les variables et les fonctions externes manipulées par le programme. Ces variables et fonctions sont déclarés dans d'autres programmes.

La partie « variables et fonctions globales » permet de déclarer les fonction et les fonctions qui peuvent être utilisées à n'importe quel endroit du programme (par opposition aux variables et fonctions locales).

La quatrième partie définit les blocs de code du programme. Le nombre de bloc est au moins égal à un (01) : le bloc principal (main). D'autres blocs peuvent être définis.

Notez qu'il est obligatoire de terminer chaque instruction d'un programme C par un point virgule.

Exercices :

Exercice 1 : Qu'est ce qu'un programme ? .

Exercice 2 : Que signifie "langage évolué" ?

Exercice 3 : L'ordinateur peut-il exécuter directement un programme écrit en langage évolué ? Expliquez .

Exercice 4 : Expliquez pourquoi il est impératif de déclarer une variable avant de pouvoir l'utiliser.

Exercice 5 : Quelle différente y'a-t-il un entier court et un entier long.

CHAPITRE III : LES INSTRUCTIONS D'ENTREES/SORTIES

Ce chapitre a pour objet de présenter les instructions d'entrée/sortie les plus importantes.

3.1 L'affichage des données : printf

Syntaxe : ***printf (<chaine_de_formats>, <liste_d'expressions>)***

L'instruction **printf** permet d'afficher sur l'écran une liste d'expressions, selon un format d'affichage souhaité (double, entier, réel, caractère, ...).

Exemple : Le programme suivant affiche « Bonjour » à l'écran.

```
#Include <stdio.h>

void main()
{
        printf(''Bonjour'') ;
}
```

La chaîne de formats est composée de séquences commençant par le caractère % suivi d'un caractère qui indique le format d'impression. Il existe, entre autres :

%c pour un caractère,
%d pour un entier à afficher en décimal
%x pour un entier à afficher en hexadécimal.

Le programme suivant affiche le carré d'un nombre n.

```
#Include <stdio.h>

void main()
{
        int n ;
        n=4 ;
       n=n*n

              printf(''Le carré est égal à %d'', n) ;
}
```

La première ligne du programme fait l'inclusion du fichier <stdio.h> qui définit les instructions d'entrées/sorties.

Après la déclaration d'une variable n de type entier, le programme affecte la valeur 4 à n. Il affecte ensuite le carré de la valeur à n, qui prend ainsi la valeur 16.

La ligne `printf(''Le carré est égal à %d'', n)` affiche à l'écran le message suivant :

Le carré est égal à 16

3.2 La lecture des données : scanf

Syntaxe : ***scanf(<chaine_de_format>, <liste_de_variables>)***

Cette instruction permet d'affecter une valeur tapée au clavier à une variable.

Exemple : Le programme suivant fait l'entrée de deux valeurs qui seront affectées respectivement à X et à Y. Il affiche après le résultat de la somme :

```
#include <stdio.h>

void main()
{
        int x, y ;

          printf(''Entrez la première valeur : '') ;
          scanf(''%i'', &x) ;

          printf(''Entrez la seconde valeur : '') ;
          scanf(''%i'', &y) ;

          printf(''La somme est égale à %i'', x+y) ;
}
```

A noter la nécessité de faire précéder le nom de la variable cible par le symbole & (« le et commercial »).

Les deux fonctions « printf » et « scanf » sont définies dans le fichier des instructions standards d'entrée/sorties « stdio.h ». Il est nécessaire de faire appel à l'inclusion de ce fichier grâce à l'instruction include au début du programme.

3.3 Les formats de données :

Nous avons que , aussi bien pour l'instruction **printf** que **scanf** , il est impératif de préciser le format des expressions à afficher ou à lire ; les différents format supportés sont :

Format	***Signification***
%c	caractère
%i	Entier court
%f	réel (float)
%d	Entier double
%s	Chaîne de caractères

EXERCICES :

Exercice 1 : Faire un programme qui lit deux nombres X et Y , calcule puis affiche le produit de X par Y.

Exercice 2 : Faire un programme qui lit deux nombres X et Y , et permute leurs valeurs .C'est à dire si X=15 et Y=10 , on doit avoir : Y=15 et X=10 .

CHAPITRE IV : LES INSTRUCTIONS DE CONTROLE

Ce chapitre a pour objet de présenter les instructions de contrôle de C : if, while, do-while et for.

4.1 L'instruction if :

Syntaxe : ***if (<condition>)***
{ Bloc 1}
else
{ Bloc 2}

Cette instruction permet la réalisation conditionnelle de différents traitements. Si la condition évaluée est vraie, le C réalise la ou les instructions contenues dans la partie <Bloc1>. Dans le cas contraire, il réalise la ou les instructions contenues dans la partie <Bloc2>.

Exemple : Le programme suivant permet de faire introduire une valeur au clavier et de tester si oui ou non elle est nulle.

```
#include <stdio.h>
void main()
{
        int x ;

          printf(''Entrez votre nombre : '') ;
          scanf(''%i'', &x) ;

          if (x>0)
                  printf(''Nombre positif'') ;
              else
                  printf(''Nombre négatif'') ;

}
```

Ce programme fait la lecture, au clavier, d'une donnée numérique qui sera affectée à la variable X. La instruction IF évalue la condition 'est-ce que X est plus grand que zéro?'. Si la condition est vraie, le C affiche 'Nombre positif', dans le cas contraire, il affiche 'Nombre négatif ou nul'.

Remarquez que dans le cas où il y a plus d'une instruction à exécuter , en cas de valeur vraie ou fausse, il y a lieu de les délimiter par les deux symboles accolades { }.

4.2 L' instruction while :

Syntaxe : ***while (<Condition>)***
{ <action1>
<action2>
...
<action-n>
}

Cette instruction permet de répéter l'exécution d'un ensemble d'actions tant que une certaine condition est vraie.

Exemple : Ce programme affiche les (10) premiers nombres entiers.

```
#include <stdio.h>

void main()
{
        int i ;

      i=1 ;
        while (i<=10)
                {
              printf(''%i'', i) ;
                i++ ;
                }

        printf(''Traitement terminé'') ;
}
```

Ce programme commence par initialiser la valeur de la variable mémoire I à 1. Il répète ensuite, les instructions : ? I et I=I+1 tantque la valeur de I est inférieure ou égale à 10. Dès que la valeur de I dépasse 10, le fonctionnement de la boucle s'arrête, et l'exécution poursuit à l'instruction qui suit immédiatement le bloc de répétition. C'est à dire que C affichera à la fin 'Traitement terminé'.

Il est possible de rompre brusquement l'exécution de l'instruction while en faisant appel à l'instruction **break**..

4.3 L' instruction do...while :

Syntaxe : ***do***

{ <action1>

<action2>

...

<action-n>

}

while (<condition>)

Cette instruction, comme la précédente, permet de répéter un ensemble d'actions tant que une certaine condition est vraie ; à la différence que cette fois l'évaluation de la condition intervient après avoir exécuté le bloc d'actions et non avant. Cela veut dire qu'il est certain que le corps de la boucle s'exécutera au moins une fois.

Voici une autre variante du programme précédent avec l'instruction do…while.

```
#include <stdio.h>

void main()
{
        int i ;

      i=1 ;
        do {
              printf(''%i'', i) ;
                i++ ;
            }
      while (i<=10)

        printf(''Traitement terminé'') ;
}
```

4.4 L' instruction : for

Syntaxe : ***for (<Variable>=<valeur1> ; <condition d'arrêt> ; <instruction-de-fin-boucle>)***

```
{ <action1>
  <action2>
  ...
  <action-n>
}
```

Cette instruction permet de répéter l'exécution d'un ensemble d'actions pour une variable allant de valeur1 jusqu'à ce que une certaine condition d'arrêt soit atteinte.

Exemple : Ce programme affiche les (10) premiers nombres entiers.

```
#Include <stdio.h>

void main()
{
        int i ;

        for (i=1 ; i<=10 ; i++)
                { printf(''%i'', i) ;  }

        printf(''Traitement terminé'') ;
}
```

4.5 L' instruction switch :

Syntaxe : **switch (<expression>)**

```
{
    case <constante1> : <bloc_instructions>
                        break ;
    case <constante2> : <bloc_instructions>
                        break ;
    case <constante3> : <bloc_instructions>
                        break ;
    ...
    default           : <bloc_instructions>
}
```

Cette instruction permet d'éviter les imbrications de tests qui concernent une même expression.

Par exemple, le programme suivant fait la lecture, à partir du clavier, d'un nombre entier et vérifie s'il est égal à 0, 1 au autre.

```
#Include <stdio.h>

void main()
{
        Int I ;

        printf(''Entrez un nombre SVP :'') ;
      scanf(''%i'', &i) ;

      switch (i)
        {
          case 0 : printf(''c'est un nombre nul'') ;
                   break ;
           case 1 : printf(''c'est un nombre égal à un'') ;
                   break ;
           default : printf(''c'est un autre nombre'') ;
                   break ;
         }

}
```

Cette instruction est bien adaptée pour la programmation d'un menu de choix (interface à partir de laquelle l'utilisateur choisit une fonction).

EXERCICES :

Exercice 1 : Faire un programme C qui affiche tous les multiples de 5 inférieurs à 100 , en utilisant l'instruction *for* .

Exercice 2 : Même problème mais en utilisant l'instruction *while* .

Exercice 3 : Faire un programme qui calcule puis affiche la somme des 10 premiers entiers (de 1 à 10) .

Exercice 4 : Faire un programme qui calcule puis affiche la somme de tous les multiples de 5 inférieurs à 100 .

Exercice 5 : Faire un programme qui étant donné un nombre entier, vérifie s'il est *premier* ou non . Un nombre entier est dit premier s'il n'est divisible que par lui même et par 1 .

Exercice 6 : Faire un programme qui lit une suite de nombres entiers (on ne connaît pas leur nombre , mais le dernier est égal à 0) , calcule puis affiche :

. Le nombre total des entiers qui ont été lus.
. La valeur du plus grand entier lu .

Exercice 7 : Même problème que l'exercice précédent mais en calculant :
. Le nombre total des entiers qui ont été lus.
. Le nombre des entiers positifs qui ont été lus.
. Le nombre des entiers négatifs qui ont été lus.

Exercice 8 : Faire un programme qui lit une suite de nombres entiers (on ne connaît pas leur nombre , mais le dernier est égal à 0) , puis affiche le plus grand des nombres pairs lu et le plus petit des nombres impairs lu .

Exercice 9 : Faire un programme qui lit un nombre entier et vérifie s'il est parfait.Un nombre entier est dit parfait s'il est égal à la somme de tous ses diviseurs (excepté lui-même). Exemple 6=1+2+3.

Exercice 10 : Deux nombres entiers m et n sont dits amis si la somme des diviseurs de m (excepté lui-même) et égale à n, et inversement la somme des diviseurs de n (excepté lui-même) est égale à m. Exemple : 220 et 284. Faire un programme qui lit deux nombres entiers puis vérifie s'ils sont amis.

Exercice 11 : Faire un programme qui calcule le PPMC de deux nombres entiers.

Exercice 12 : Faire un programme qui calcule le PGCD de deux nombres entiers.

Exercice 13 : Faire un programme qui calcule le nombre PI de la manière suivante :

PI= 4*(1- 1/3 +1/5 -1/7 +1/9 -1/11 +...)

On s'arrêtera lorsque la valeur absolue de l'un des termes de la suite devient inférieur à 10^{-4} .

Exercice 14 : Un résultat connu en mathématiques : Tout nombre entier peut être écrit sous forme d'un produit de puissances de nombres premiers . Exemples :

$24=2^3 * 3$, 70=2*5*7 , $350=2*5^2 *7$

Faire un programme qui lit un nombre entier et l'affiche sous forme d'un produit de puissances de nombres premiers.

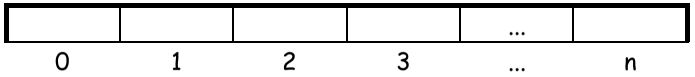
CHAPITRE V : LES TABLEAUX

Ce chapitre a pour objet de présenter la structure de données tableau.

5.1 Qu'est ce qu'un tableau ? :

Jusqu'à présent nous n'avons traité que des variables simples (entières, caractères, réelles, ...). Ces structures de données simples ne suffisent pour répondre à tous les cas de modélisation informatique ; d'autres structures de données plus complexes sont nécessaires. Parmi ces structures, nous trouvons les tableaux.

Un tableau peut être définit schématiquement comme une suite de variables simples. L'accès à un élément quelconque du tableau se fait grâce à son indice.



T[3] signifie, par exemple, l'élément de rang 3 du tableau T. Dans le langage C, la numérotation des tableaux commence par l'indice 0.

5.2 Déclaration des Tableaux :

La déclaration d'un tableau doit préciser son type de base et le nombre de ses éléments.

Exemples :

```
int    t[10]                  /* déclare un tableau t de 10 éléments  entiers.
long int  t1[10], t2[20]      /* déclare deux tableaux t1 et t2 d'entiers longs. Le premier
                              /* a 10 éléments et le second 20.
```

L'utilisation des tableaux recquiert la compréhension des remarques suivantes :

- Les index des éléments d'un tableau vont de 0 à n-1.
- La taille d'un tableau doit être connue statiquement par le compilateur. C'est à dire qu'il est impossible d'écrire :

int t[n], où n serait une variable.

Le programme suivant fait la saisie de 5 valeurs qui seront affectées à un tableau T. Ces valeurs seront alors affichées en sens inverse (de la dernière à la première).

```
#Include <stdio.h>

void main()
{
        int   T[5] ;
      int k, v ;

        for (k=0 ; k<5 ; k++)
      { scanf(''%i'', &v) ;
```

```
            T[k]=v
           }

            for (k=4 ; k>-1 ; k- -)
             printf(''%i'', T[k]) ;
}
```

5.3 Les tableaux à plusieurs dimensions :

Il est possible de déclarer des tableaux à plusieurs dimensions. Par exemple :

```
int   T[5][10]          /* Tableau à deux dimensions (matrice de 5 lignes
                        /*et 10 colonnes).
```

5.4 Les tableaux de caractères :

Comme pour les autres types de base, il est possible de manipuler des tableaux de caractères. Il existe toutefois une particularité pour ces derniers qui consiste en la possibilité d'affecter toute une chaîne au tableau .

Exemple :

```
char t[20] ;            /*déclare un tableau de caractères de 20 éléments
t= "Langage C";         /* affecte la chaine de caractère "Langage C" au
                        /*tableau t.
```

Exercices.

Exercice 1 : Faire un programme qui range N valeurs numériques dans un tableau, puis recherche si une valeur donnée X existe dans ce tableau.

Exercice 2 : Faire un programme qui lit les N éléments d'un tableau numérique puis affiche le maximum .

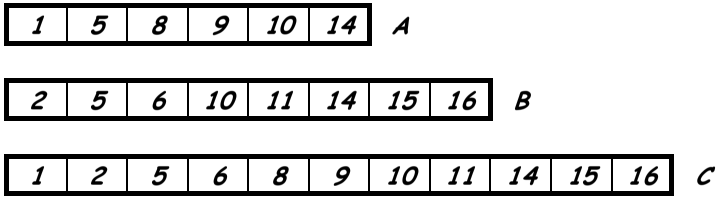
Exercice 3 : Faire un programme qui lit les N éléments d'un tableau numérique , puis le trie par ordre croissant .

Exercice 4 : Soit une suite mathématique définie de la manière suivante :

$$U_n = U_{n-1} + 2*U_{n-2}$$
$$U_0 = 1 \quad , \quad U_1 = 3$$

Faire un programme qui étant donné un nombre entier N range tous les éléments de cette suite dans un tableau .
Faire un autre programme qui consulte la valeur d'un élément quelconque U_i .

Exercice 5 : Faire un programme qui étant donnés deux tableaux numériques triés A et B (dont le nombre d'éléments sont respectivement m et n) en tire un nouveau tableau numérique trié C . (Faire attention de ne pas reproduire un élément plus d'une fois) . Exemple :



Exercice 6 : Faire un programme qui lit les N éléments d'un tableau de caractères , et vérifie s'il s'agit d'une chaîne **palindrome** .

Une chaîne de caractères est dite palindrome si elle est identique à son image miroir (c'est à dire qu'on peut la lire indifféremment de droite à gauche ou de gauche à droite) . Exemples : radar, elle, sos , ... etc.

Exercice 7 : Faire un programme qui affiche tous les entiers premiers inférieurs à un nombre donné N .

Indication : Représenter dans un tableau tous les nombres entiers compris entre 1 et N, puis au fur et à mesure afficher un nombre premier et éliminer tous ses multiples.

Exercice 8 : Etant donné un tableau numérique trié par ordre croissant. Faire une recherche dichotomique pour vérifier si un nombre X existe dans ce tableau.

La recherche dichotomique se fait de la manière suivante:

- On compare le nombre X avec l'élément du milieu du tableau.
- Si les deux valeurs sont égales on arrête la recherche , sinon on ignore la moitié du tableau dans laquelle on n'a aucune chance de trouver la valeur X et on considère l'autre moitié. Le nombre X est alors comparé au milieu de cette partie , ... et ainsi de suite.

Exercice 9 : Faire un programme qui étant donnée une matrice de nombres , recherche puis affiche le plus grand élément .

Exercice 10 : Faire un programme qui étant donnée une matrice de nombres , affiche le maximum de chaque colonne.

Exercice 11 : Le point de selle d'une matrice est l'élément qui est minimal sur sa ligne et maximal sur sa colonne . Par exemple, le nombre 7 de la matrice suivante est un point de selle :

6	5	2
11	7	8
14	3	1
4	4	9

Le point de selle s'il existe est unique. Faire un programme qui cherche le point de selle d'une matrice.

Exercice 12 : Faire un algorithme qui fait la somme de deux matrices numériques .

Exercice 13 : Faire un algorithme qui fait le produit de deux matrices numériques.

CHAPITRE VI : LES STRUCTURES & LES UNIONS

Ce chapitre a pour objet de présenter les structures du langage C.

6.1 Les Structures :

Une structure en langage C est une structure de données qui regroupe des variables hétérogènes, c'est à dire qui peuvent avoir des types différents.

C'est le cas , par exemple, des informations concernant une personne qui peuvent être regroupées au sein d'une même structure :

```
Struct personne
   {
        int age ;
        float taille, poids ;
        char nom[20] ;
   }
```

Cette déclaration précise que l'identificateur personne est une structure composée des quatre champs : le premier est un entier nommé age, le second est un réel nommé taille, le troisième est un réel nommé poids et le dernier est un tableau de 20 caractères représentant le nom de la personne.
Une structure est construite selon le schéma général suivant :

```
Struct <nom_structure>
        {   type1       membre1 ;
            type2       membre2 ;
            type3       memebre3 ;
            ....
            typen       membren
        }
```

L'exemple suivant déclare deux variables **p1** et **p2** , qui ont toutes les deux la même structure que personne.

```
Struct personne
   {
        int age ;
        float taille, poids ;
        char nom[20] ;
   }  p1, p2 ;
```

6.2 Accès aux champs de la structure :

Pour désigner un champs d'une structure, il faut utiliser l'opérateur de sélection de champ qui se note . (point).

Par exemple, si **p1** et **p2** sont deux variables de type **struct personne**, on désignera le champs **nom** de **p1** par **p1.nom**, et on désignera le champ **taille** de **p2** par **p2.taille**.

Le programme suivant montre un exemple de manipulation d'une structure **personne**. Il permet la saisie des renseignemens d'une personne donnée.

```
#Include <stdio.h>
```

```
void main()
{
            struct personne
            {  int age ;
               float    taille, poids ;
               char     nom[20] ;
            }

          int a ;
          float t, p ;
          char n[20] ;
          personne homme ;
           printf(''Introduisez l'age '') ;
           scanf(''%i'', &a) ;
           homme.age=a ;

           printf(''Introduisez le poids '') ;
           scanf(''%f'', &p) ;
           homme.poids=p ;

           printf(''Introduisez la taille '') ;
           scanf(''%f'', &t) ;
           homme.taille=t ;

           printf(''Introduisez le nom '') ;
           scanf(''%c'', &n) ;
           homme.nom=n ;
```

6.3 Tableaux de structures :

Une déclaration de tableau de structures se fait selon le même modèle que la déclaration d'un tableau dont les éléments sont de type simple.

Supposons que l'on ait déjà déclaré la struct personne, si on veut déclarer un tableau de 100 strcutures de ce type, on écrira :

Struct personne t[100] ;

Pour référencer le nom de la personne qui a l'index I dans t , on écrira :

T[I].nom ;

6.4 Les Unions :

La structure de données « union » est une structure qui permet de manipuler un champs en utilisant plusieurs types de données. En clair cela veut dire que la donnée peut être « vue » comme amorphe (ayant plusieurs formes).

L'exemple suivant déclare une union décrivant une zone mémoire qui peut être vue, selon le cas, comme une valeur entière ou un caractère.

```
Union zone
        {   int  i ;
            char c ;
        }
```

l'emploi des différents éléments d'une union se fait comme pour les structures. Ainsi si z est une variable de type zone, on peut alors écrire :

```
z.i = 15
z.c = 't'
```

D'une manière générale , la déclaration d'une union se fait selon le schéma suivant :

```
Union <nom_union>
        {   type1       membre1 ;
            type2       membre2 ;
            type3       memebre3 ;
            ....
```

```
        typen       membren
    }
```

La différence sémantique entre les **struct** et les **unions** est la suivante : alors que pour une variable de type structure tous les champs peuvent avoir en même temps une valeur, une variable de type union ne peut avoir , à un instant donné , qu'un seul champs ayant une valeur.

Dans l'exemple précédent, l'union z pourra posséder soit une valeur entière, soit une valeur réelle, amis pas les deux à la fois.

6.5 Accès aux champs de l'union :

L'accès au champs de données de l'union se fait avec le même opérateur sélection (noté .) que celui qui sert à accéder aux champs des structures.

Exercices.

Exercice 1 : Faire un programme qui saisit dans un tableau les informations d'un ensemble d'employés . Chaque employé est défini par : son matricule, son nom, son prénom, et son salaire .(Utilisez une structure d'enregistrement) .

Exercice 2 : Faire un programme qui affiche les informations de tous les employés de l'exercice précédent dont le salaire est supérieur à un montant donné .

Exercice 3 : Faire un programme qui trie par ordre croissant du salaire les employés de l'exercice 1 .

Exercice 4 : Etant donné un tableau contenant les informations de N personnes. Chaque personne est représentée par un enregistrement formé des champs suivants : Nom, Prénom et Date de naissance. Le champ date de naissance est aussi un enregistrement formé des 3 champs : jour, mois et année.

Faire les programmes suivants :

- Editer toutes les personnes nées au mois de Décembre.
- Editer toutes les personnes par année de naissance.

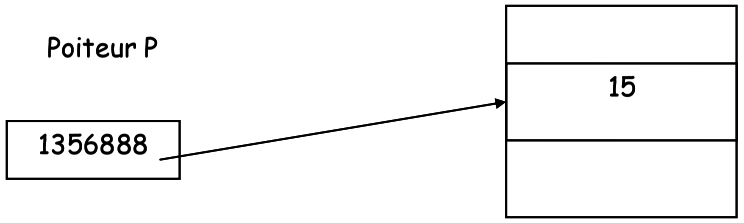
CHAPITRE VII : LES POINTEURS

Ce chapitre a pour objet de présenter les pointeurs en langage C.

7.3 Les Pointeurs :

Les pointeurs sont des outils fondamentaux du langage C. Toutefois leur maîtrise nécessite un minimum d'expérience.

Sommairement, un pointeur peut être défini comme une variable qui contient l'adresse d'une zone mémoire (et non le contenu de cette zone), comme le montre le schéma suivant :



7.2 Déclaration des pointeurs :

La déclaration d'une variable pointeur vers un type de base a une structure très proche de celle de la déclaration d'une variabe ayant un type de base. La seule différence consiste à faire précéder le nom de la variable du signe *.

Exemples de déclarations :

```
float  *pi ;        //  pi est un pointeur vers float
char   *pc ;        // pc est pointeur vers un caractère.
int    *s ;         // s est un pointeur vers un entier
```

Remarque importante :

En utilisant les pointeurs , il est important de faire attention à la notation . Ainsi par exemple, si p est un pointeur vers un entier, il faut bien comprendre que :

P contient l'adresse de l'entier (par exemple 1356888)
*p contient la valeur de l'entier lui-même (par exemple 15).

7.3 L'opérateur adresse de : &

Les programmes C utilisent souvent l'opérateur & (appelé « de » ou « et commercial »). Cet opérateur appliqué à une variable délivre l'adresse de celle-ci, adresse qui pourra donc être affectée à une variable de type pointeur.

Exemple :

```
int i ;
int *pi ;

pi=&i ;              //pi   pointera vers la variable I ;
```

Autre exemple :

```
int i,  j  ;
int *pi ;

pi=&i ;              //initialisation du pointeur pi
*pi=2 ;              //initialisation de la valeur pointée par pi
j=*pi+1 ;            // la variable j aura la valeur 3
```

7.4 Pointeur vers une structure :

Supposons que l'on ait défini la structure personne à l'aide de la déclaration :

```
Struct personne
   {
        int age ;
        float taille, poids ;
        char nom[20] ;
   }
```

On déclarera une variable de type pointeur vers une telle structure de la manière suivante :

```
Struct personne   *p ;
```

On pourra alors affecter à p des adresses de **struct personne**, par exemple :

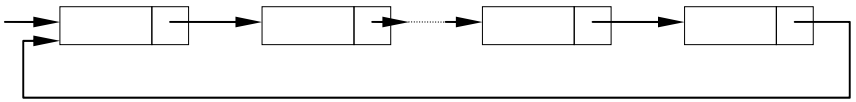
```
Struct personne
   {
        int age ;
        float taille, poids ;
        char nom[20] ;
   }


void main()
{
        struct personne pers ;      //pers est une variable de type
                                    //struct personne
        struct personne *p ;        //p est un pointeur vers une
                                    //struct personne

         p=&pers ;
}
```

7.5 Structure dont un champs pointe vers une structure du même type :

Une des utilisations fréquentes des structures, est de créer des listes de structures chaînées (voir figure). Pour cela, il faut que chaque structure contienne un champ qui soit de type pointeur vers une structure



Exemple de liste chainée.

En reprenant l'exemple de la structure personne, la liste précédente peut être construite à partir de la déclaration suivante :

```
Struct personne
   {
         int age ;
         float taille, poids ;
         char nom[20] ;

         struct personne   *suivant ;
   }
```

Le champs de nom **suivant** est déclaré comme étant du type pointeur vers une struct personne.

7.6 Accès aux éléments d'une structure pointée :

Supposons que nous ayons déclaré p comme étant de type pointeur vers une **struct personne** , comment faire référence à un champ de la structure pointée par p ?.

Pour référencer le champ **nom** de la structure en question, on peut utiliser la notation suivante :

```
(*p).nom
```

Il en va de même pour les autres champs : age, taille et poids, qui seront représentés respectivement ainsi :

```
(*p).age
(*p).poids
(*p).taille
```

Etant donné que cette écriture est assez lourde, le langage C a prévu un nouvel opérateur noté **->** qui permet de référencer facilement les champs d'une structure pointée.

Ainsi au lieu d'écrire **(*p).nom** , on écrira **p->nom**. De même que les autres champs pourront être référencés comme suit :

```
p->age
p->poids
p->taille
```

7.7 Détermination de la taille allouée à un type :

Pour connaître la taille en octets de l'espace mémoire nécessaire pour une variable, on dispose de l'opérateur **sizeof**.

Cet opérateur peut être utilisé de deux manières différentes :

sizeof expression
sizeof (nom-de-type)

Exemple :

```
int  i, taille ;

taille=sizeof  i ;
taille=sizeof (int) ;
taille=sizeof(struct personne) ;
```

la première instruction **taille=sizeof i** permet de calculer la taille en octets occupée par la variable mémoire i. Les deux suivantes permettent respectivement de calculer la taille occupée par une variable de type entier et le type structure.

7.8 Allocation et libération d'espace :

Les pointeurs permettent d'allouer et de libérer dynamiquement de l'espace mémoire. Cela se passe généralement lors de la manipulation des listes chaînées (dont le nombre d'éléments n'est pas connu à l'avance et peut varier pendant l'exécution).

Pour l'allocation, on dispose de deux fonctions **malloc** et **calloc** :

La fonction **malloc** admet un paramètre qui est la taille en octets de l'élément désiré, et elle rend un pointeur vers l'espace alloué.

Exemple :

```
struct personne  *p ;
```

```
p=malloc(sizeof(struct personne)) ;
```

La fonction **calloc** admet deux paramètres :

1. Le premier est le nombre d'éléments désirés.
2. Le second est la taille en octets d'un élément.

Son but est d'allouer un espace suffisant pour contenir les éléments demandés et de rendre un pointeur vers cet espace.

Exemple d'utilisation :

```
struct personne *p ;
int nb_elem ;

nb_elem=10 ;

/* allocation d'un espace de 10 éléments de type struct personne
p=calloc(nb_elem, sizeof(struct personne)) ;
```

On pourra alors utiliser les éléments p[0], p[1], … p[nb_elem-1].

La **libération** de l'espace alloué par malloc ou calloc se fait au moyen de la procédure **free** qui admet un seul paramètre : le pointeur précédemment rendu par malloc ou calloc.

Exemple :

```
struct personne *p ;
int nb_elem ;

nb_elem=10 ;

/* allocation d'un espace de 10 éléments de type struct personne
p=calloc(nb_elem, sizeof(struct personne)) ;
```

```
free(p) ;
```

EXERCICES :

Exercice 1 : Soit une liste chaînée de nombres triée par ordre croissant . Ecrire les programmes suivante :

a. Recherche si un nombre donné X existe dans cette liste.
b. Insère un nombre X au bon endroit de la liste de manière à préserver l'ordre .
c. Supprime le premier élément rencontré dont la valeur est X.

Exercice 2 : Refaire l'exercice précédent avec une liste bidirectionnelle puis une liste circulaire.

Exercice 3 : Soit une liste linéaire d'éléments Employé , dont chacun est formé des champs suivants :

. Numéro de l'employé .
. Nom de l'employé .
. Prénom de l'employé .
. Salaire de l'employé .

Ecrire les programmes suivants :

. Insérer un nouvel employé en fin de liste .
. Supprimer un employé de numéro donné .
. Afficher tous les employés dont le salaire est supérieur à un montant donné .
. Trier la liste par ordre croissant des salaires .

Exercice 9.4 : Soit une suite mathématique définie de la manière suivante :

$$U_n = U_{n-1} + 2*U_{n-2}$$
$$U_0 = 1 , U_1 = 3$$

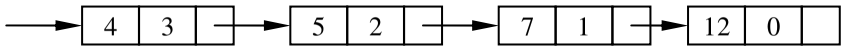
Faire un programme qui étant donné un nombre entier N range tous les éléments générés de cette suite dans une liste linéaire.

Faire un autre programme qui consulte la valeur d'un élément quelconque U_i .

Exercice 9.5 : Pour représenter les polynômes on a pensé à utiliser les listes. Chaque monôme est ainsi représenté par un élément de liste à trois champs : le coefficient du monôme , son exposant et le lien vers le monôme suivant. Exemple :

Le polynôme : $4*x^3 + 5*x^2 + 7*x + 12$

sera représenté ainsi :



Faire un programme qui étant données deux listes représentant deux polynômes (dont les adresses des premiers éléments sont respectivement Tête1 et Tête2) , construit une nouvelle liste représentant le polynôme résultat de la somme.

CHAPITRE VIII : LES FONCTIONS

Ce chapitre a pour objet de présenter la structuration des fonctions (sous-programmes) dans les programmes C.

8.1 Structure des fonctions :

La structure générale des fonctions en C obéit au schéma suivant :

```
<type_fonction> <nom_fonction> (type1 argument1, type2 argument2,
...,
typen   argumentn)
        {
                Corps de la fonction

        }
```

le type de la fonction permet de préciser le type du résultat rendu par la fonction. Si la fonction ne renvoit aucun résultat, sa déclaration doit être précédée par le mot réservé **void**.

Exemple : la fonction suivante réalise la somme de deux nombres réels x et y.

```
Float   Somme (float x, float y)
        {
                return(x+y)
        }
```

Le programme suivant utilise la fonction somme précédente pour faire la somme de deux nombres a et b introduits par l'utilisateur :

```
#include <stdio.h>

float   somme (float x, float y) ;
void main()
{
      float    a, b ;

      printf(''Introduisez le premier nombre '') ;
      scanf(''%f'', &a) ;

      printf(''Introduisez le second nombre '') ;
      scanf(''%f'', &b) ;

      printf(''La somme est %f = '', somme(a, b)) ;
}

float   Somme (float x, float y)
{
                return(x+y)
        }
```

Au moment de l'appel de la fonction somme, les paramètres x et y (appelés paramètres formels) sont remplacés respectivement par a et b (appelés paramètres effectifs).

Notez qu'on doit informer le compilateur du nom de la fonction avant le bloc (main).

8.2 Portée des variables :

Il existe en réalité deux sortes de variables en C : les variables globales et les variables locales.

Les variables globales sont des variables qui peuvent êtres « vues » et utilisées par tous les blocs d'instructions (programme principal main et les toutes les autres fonctions). Leur durée de vie est donc égale à la durée de vie du programme principal.

Les variables locales, par contre, sont des variables déclarée localement à l'intérieur d'un bloc. Leur durée de vie est égale à la durée de vie du bloc qui les a crées. A la sortie de ce bloc, ces variables sont détruites.

Le programme suivant résume la différence entre les deux types de variables :

```
#include <stdio.h>

int a,b ;                                   /* déclaration de variables
globales */
int fonc1(void);
int fonc2(void);

void main()
{
        int c;                              /* déclaration d'une variable
locale */
        a=5; b=10; c=15;
        printf("%i", a+b+c);
        printf("%i", fonc1());
        printf("%i", fonc2());
};

int fonc1(void)
{
        int d;                              /* déclaration d'une
variable locale */
        d=20;
        return(a+b+d);
}

int fonc2(void)
{
        int e;                              /* déclaration d'une
variable locale */

        e=20;
        return(a+c+e);                      /* il y a une erreur dans
cette ligne */
}
```

Les variables a et b sont des variables globales. Elles peuvent utilisées dans le programme principal (main) et tous les autres blocs (exemple : fonc1 & fonc2).

Cependant, la variable c est une variable locale. Elle n'est vue que dans le bloc correspondant au programme principal (main). De ce fait, une erreur sera signalée à la compilation : lors du passage sur le bloc fonc2, le compilateur conclura que la variable c n'est pas définie !.

EXERCICES :

Exercice 1 : Ecrire une fonction **ABS** qui calcule la valeur absolue d'un nombre .

Exercice 2 : Ecrire une procédure **Permute** qui permute le contenu de deux variables numériques. Utilisez cette procédure pour faire la permutation circulaire du contenu de trois variables.

Exercice 3 : Ecrire une fonction qui étant donné un rayon R, calcule la surface du cercle correspondant .

Exercice 4 : Ecrire une fonction qui calcule la nième puissance d'un nombre X .

Exercice 5 : Faire une fonction **Fact** qui calcule la factorielle d'un nombre entier N . Rappelons que la factorielle d'un nombre notée n! est : n! = 1*2*3*4*...*(n-1)*n

Exercice 6 : Utiliser la fonction de l'exercice précédent pour faire un programme qui lit 10 nombres entiers et calcule la somme de leurs factorielles .

CHAPITRE IX : LES FICHIERS

Le présent chapitre fait une introduction à la manipulation des fichiers comme supports d'entrée/sortie.

9.1 Déclaration des fichiers :

La déclaration d'un fichier se fait au moyen de l'instruction FILE qui est contenue dans le fichier <stdio.h>.

L'exemple suivant déclare un fichier **fp** :

```
#include <stdio.h>

void main()
{
        FILE   *fp ;
……
……
}
```

9.2 Ouverture des fichiers : fopen

Lorsqu'on désire accéder à un fichier, il est nécessaire avant tout d'ouvrir le fichier à l'aide de la fonction **fopen** dont la syntaxe est :

fopen(<nom_physique>, <mode>)

Le nom physique est une chaîne de caractères représentant le nom réel du fichier auquel on veut accéder sur un disque, exemple : "resultat.dat".

Le mode précise le mode d'utilisation du fichier. Il peut être choisi parmi les trois cas suivants :

- **"r"** : Ouverture en lecture (read). Dans ce cas le fichier ouvert ne peut être que lu et ne peut être modifié.
- **"w"** : Ouverture en écriture (write). Dans ce cas le fichier est remis à zéro avant de recevoir de nouvelles données.
- **"a"** : Ouverture en ajout (append). Dans ce cas le contenu du fichier ouvert est conservé, et les nouvelles données seront rajoutées à la fin de celles qui existent déjà.

La fonction fopen retourne une valeur de type pointeur vers FILE. Si l'ouverture a réussi, la valeur retournée permet de repérer le fichier, et devra être passée en paramètre à toutes les procédures d'entrées/sorties sur le fichier. Si l'ouverture s'est avéré impossible (cas d'un fichier inexistant qu'on veut lire, par exemple) fopen rend la valeur NULL.

Exemple : Cet exemple tente d'ouvrir le fichier "resultat.dat". Si l'ouverture ne réussit pas, le programme affiche un message d'alerte à l'utilisateur.

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

```
void main()
{
        FILE   *fp ;

fp=fopen(''resultat.dat'', ''r'') ;
if (fp==NULL)
     printf(''Impossible d'ouvrir ce fichier'') ;
……
……
}
```

9.3 Ouverture des fichiers : fclose

Après avoir terminé les entrées/sorties sur un fichier, il est fortement recommandé de le fermer en utilisant la fonction **fclose**, dont la syntaxe est :

fclose(<nom_fichier>)

Exemple : Cet exemple complète le programme précédent en faisant une ouverture d'un fichier en lecture, puis en le fermant grâce à l'instruction fclose.

```
#include <stdio.h>

void main()
{
        FILE   *fp ;

fp=fopen(''resultat.dat'', ''r'') ;
if (fp==NULL)
     printf(''Impossible d'ouvrir ce fichier'') ;

fclose(fp) ;
}
```

9.4 Ecriture sur un fichier : fprintf

L'instruction fprintf est analogue à printf à l'execption que fprintf s'applique pour un fichier sur disque alors que printf s'applique pour la sortie standard (écran). Les formats admissibles sont rigoureusement les mêmes pour les deux instructions.

L'exemple suivant écrit dans le fichier "nombre" les 100 premiers entiers :

```
#include <stdio.h>

void main()
{
        FILE   *fp ;
        int i ;

fp=fopen(''nombre'', ''w'') ;
if (fp==NULL)
     printf(''Impossible d'ouvrir ce fichier'') ;

for(i=1 ;i<=100 ; i++)
  {  fprintf(fp, ''%d'', i) ;
  }

fclose(fp) ;
}
```

9.5 Lecture à partir d'un fichier : fscanf

Comme pour scanf, l'instruction fscanf permet de lire des données formatées à partir d'un fichier . Le programme suivant complète le programme précédent en ouvrant le fichier "nombre" en lecture après l'écriture des 100 premiers entiers.

```
#include <stdio.h>

void main()
{
        FILE   *fp ;
        int i , n;

fp=fopen(''nombre'', ''w'') ;
if (fp==NULL)
     printf(''Impossible d'ouvrir ce fichier'') ;

for(i=1 ;i<=100 ; i++)
  {  fprintf(fp, ''%d'', i) ;
  }

close(fp) ;
fp=fopen(''nombre'', ''r'') ;
while (fscanf(fp, ''%d'', &n) !=EOF)
     printf(''%d'', n) ;

fclose(fp) ;
}
```

EXERCICES :

Exercice 1 : Soit un fichier destiné à contenir les informations des étudiants. Chaque étudiant est définit par les rubriques suivantes : matricule, nom et moyenne.

Ecrire un programme C qui appelle les fonctions suivantes :

- Saisie d'un nouvel étudiant
- Affichage des données du fichier.
- Suppression d'un nouvel étudiant
- Tri du fichier

Exercice 2 : Ecrire un programme C qui recopie le contenu d'un fichier sur un autre.

ANNEXE : PRINCIPALES FONCTIONS DE C/C++

Fonction	*Rôle*	*Fichiers entêtes nécessaires*
int abs(int *n* **);**	Calcule la valeur absolue d'un nombre entier n.	math.h
double acos(double *x* **);**	calcule l'arc-cosinus d'un nombre réel x.	math.h
double asin(double *x* **);**	calcule l'arc-sinus d'un nombre x.	math.h
double atan(double *x* **);**	calcule l'arc-tangente d'un nombre x.	math.h
double atof(const char * *string* **);**	convertit une chaine de caractère en un nombre réel.	Math.h et stdlib.h
int atoi(const char * *string* **);**	convertit une chaine de caractère en un nombre entier.	Math.h et stdlib.h
long atol(const char * *string* **);**	convertit une chaine de caractère en un entier long.	Math.h et stdlib.h
double ceil(double *x* **);**	donne l'entier supérieur ou égal à x.	math.h
char *_cgets(char * *buffer* **);**	lit à partir du clavier une chaine de caractères.	Conio.h
double cos(double *x* **);**	calcule le cosinus d'un nombre x	math.h
double cosh(double *x* **);**	calcule le cosinus hyperbolique d'un nombre x	math.h
char *_ecvt(double *value*, **int** *count*, **int *** *dec*, **int *** *sign* **);**	convertit un nombre x en une chaîne de caractères. Value : valeur à convertir. count : le nombre de positions souhaité. dec : nombre de positions décimales. Sign : signe du nombre.	Stdlib.h
double exp(double *x* **);**	calcule l'exponentielle d'un nombre x	math.h
double fabs(double *x* **);**	calcule la valeur absolue d'un nombre réel x.	math.h
char *_fcvt(double *value*, **int** *count*, **int *** *dec*, **int *** *sign* **);**	convertit un nombre réel x en une chaîne de caractères.	Stdlib.h
double floor(double *x* **);**	calcule le nombre entier supérieur ou égal à x.	math.h
double fmod(double *x*, **double** *y* **);**	calcule le reste de la division de deux nombres x et y.	math.h
double _hypot(double *x*, **double** *y* **);**	calcule l'hypoténuse d'un triangle droit : racine carré de $x^2 + y^2$.	Math.h
int _inp(unsigned short *port* **);** **unsigned short _inpw(unsigned short** *port* **);** **unsigned long _inpd(unsigned short** *port* **);**	lit un octet (byte) à partir d'un port.	Conio.h
unsigned short _inpw(unsigned short *port* **);**	lit un mot (word) à partir d'un port.	Conio.h
unsigned long _inpd(	lit un double-mot à partir d'un port.	Conio.h

unsigned short *port* **);**		
islower (int c)	teste si un caractère est miniscule. Dans l'affirmatif, la fonction renvoie une valeur non nulle.	Ctype.h
isprint (int c)	teste si un caractère est imprimable.	Ctype.h
ispunct (int c)	teste s'il s'agit d'un caractère de ponctuation.	Ctype.h
isspace (int c)	teste s'il s'agit d'un caractère espace.	Ctype.h
isupper (int c)	teste s'il s'agit d'un caractère majuscule.	Ctype.h
isxdigit (int c)	teste s'il s'agit d'un caractère hexadécimal.	Ctype.h
char *_itoa(int *value*, **char *** *string*, **int** *radix* **);**	convertit un entier en une chaîne de caractères. Value : valeur. Radix : base d'énumération choisie.	Stdlib.h
int _kbhit(void);	renvoie une valeur non nulle si une touche du clavier a été pressée, 0 sinon.	Conio.h
double log(double *x* **);**	calcule le logarithme d'un nombre x.	math.h
double log10(double *x* **);**	calcule le logarithme de base 10 d'un nombre x.	math.h
unsigned long _lrotl(unsigned long *value*, **int** *shift* **);**	fait une rotation à gauche d'un mot de bits. Shift : est le nombre de positions de la rotation.	Stdlib.h
unsigned long _lrotr(unsigned long *value*, **int** *shift* **);**	fait une rotation à droite d'un mot de bits. Shift : est le nombre de positions de la rotation.	Stdlib.h

type **__max(** *type a*, *type b* **);**	calcule le maximum de deux valeur a et b.	stdlib.h
type **__min(** *type a*, *type b* **);**	calcule le minimum de deux valeurs a et b.	Stdlib.h
int _outp(unsigned short *port*, **int** *databyte* **);**	envoie un octet (byte) vers un port.	conio.h
unsigned short _outpw(unsigned short *port*, **unsigned short** *dataword* **);**	envoie un mot vers un port.	conio.h
unsigned long _outpd(unsigned short *port*, **unsigned long** *dataword* **);**	envoie un mot double vers un port.	conio.h
double pow(double *x*, **double** *y* **);**	calcule la puissance x^y.	Conio.h
int printf(const char * *format* [, *argument*]... **);**	affiche une liste d'expressions sur l'écran.	Stdio.h
int _putch(int *c* **);**	imprime un caractère sur l'écran	conio.h
int rand(void);	renvoie un nombre aléatoire	stdlib.h
int scanf(const char * *format* [, *argument*]... **);**	permet de lire une liste de variables au clavier	stdio.h
double sin(double *x* **);**	calcule le sinus d'un nombre x.	math.h
double sinh(double *x* **);**	calcule le sinus hyperbolique d'un nombre x.	math.h
double sqrt(double *x* **);**	calcule la racine carrée d'un nombre x	math.h
char *strcat(char	Fait la concaténation de la chaine	string.h

***string1*, const char *_string2_);**	de caractères string2 à string1	
char *strchr(const char *_string_, int _c_);	Recherche la position d'un caractère dans une chaîne. La valeur renvoyée est un pointeur vers la position trouvée du caractère, NULL sinon.	String.h
int strcmp(const char *_string1_, const char *_string2_);	compare deux chaînes de caractères string1 et string2. La valeur renvoyée est : <0 si string1 est inférieure à string2 0 si string1 est identique à string2 >0 si string1 est supérieure à string2	string.h
char *strcpy(char *_string1_, const char *_string2_);	copie la chaîne de caractères string2 dans string1.	String.h
size_t strcspn(const char *_string1_, const char *_string2_);	recherche si une sous-chaîne de caractères string2 est incluse dans une chaîne string1. Dans l'affirmatif, elle renvoie la valeur de l'indice de string2 dans string1	String.h
size_t strlen(const char *_string_);	Renvoie la longueur d'une chaîne de caractères.	String.h
char *_strlwr(char *_string_);	convertit une chaîne de caractères en minuscules.	String.h
char *strncat(char *_string1_, const char *_string2_, size_t _count_);	concatène les n caractères de string2 à string1.	String.h
char *strncpy(char *_string1_, const char *_string2_, size_t _count_);	copie les n caractères de string2 dans string1.	String.h
char *strstr(const char *_string1_, const char *_string2_);	recherche si la chaîne string2 est incluse dans string1. Elle renvoie, dans l'affirmatif, l'indice de string2.	String.h
char *_strupr(char *_string_);	convertit une chaîne de caractères en majuscules.	String.h
double tan(double _x_);	calcule la tangente d'un nombre x.	math.h
double tanh(double _x_);	calcule la tangente hyperbolique d'un nombre x.	math.h
int tolower(int _c_);	convertit un caractère en minuscule.	string.h et ctype.h
int toupper(int _c_);	convertit un caractère en majuscule.	String.h et ctype.h