LFM

Cours -TP de C

Cours -TP de C

Ce cours est une introduction au langage C. Il est recommandé d'avoir sous la main un ouvrage de référence des commandes C. D'autre part, il n'aborde que les notions de base.

LYCEE FRANCO-MEXICAIN
HOMERO 1521
COLONIA POLANCO
11560 MEXICO ; D.F.

Sommaire

ELEMENTS DE LANGAGE C

Une disquette contenant les programmes (développés sous Borland C++) de ce polycopié est disponible, en évitant la saisie.
Le corrigé des exercices et le listing de ces programmes se trouvent à la fin de chaque chapitre.

Introduction

Le langage C est un langage évolué et structuré, assez proche du langage machine destiné à des applications de contrôle de processus (gestion d'entrées/sorties, applications temps réel ...). Les compilateurs C possèdent les taux d'expansion les plus faibles de tous les langages évolués (rapport entre la quantité de codes machine générée par le compilateur et la quantité de codes machine générée par l'assembleur et ce pour une même application);

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

Exemples : math.h : bibliothèque de fonctions mathématiques
stdio.h : bibliothèque d'entrées/sorties standard

On ne saurait développer un programme en C sans se munir de la documentation concernant ces bibliothèques.
Les compilateurs C sont remplacés petit à petit par des compilateurs C++.
Un programme écrit en C est en principe (mais pas toujours) compris par un compilateur C++.
Le cours qui suit est un cours ce langage C écrit dans un contexte C++ (évitant donc les problèmes de compabilité).

Edition, mise au point et exécution d'un programme

*1- **Edition du programme source***, à l'aide d'un éditeur (traitement de textes). Le nom du fichier contient l'extension .CPP, exemple: EXI_1.CPP (menu « edit »).

*2- **Compilation du programme source,*** c'est à dire création des codes machine destinés au microprocesseur utilisé. Le compilateur indique les erreurs de syntaxe mais ignore les fonctions-bibliothèque appelées par le programme.
Le compilateur génère un fichier binaire, non listable, appelé fichier objet: EXI_1.OBJ (commande « compile »).

*3- **Editions de liens***: Le code machine des fonctions-bibliothèque est chargé, création d'un fichier binaire, non listable, appelé fichier executable: EXI_1.EXE (commande « build all »).

*4- **Exécution du programme*** (commande « flèche jaune »).

Les compilateurs permettent en général de construire des programmes composés de plusieurs fichiers sources, d'ajouter à un programme des unités déjà compilées ...

Exercice I-1 : Editer (EXI_1.CPP), compiler et exécuter le programme suivant:

```
#include <stdio.h>              /* bibliotheque d'entrees-sorties standard */
#include <conio.h>
void main()
{
        puts("BONJOUR"); /* utilisation d'une fonction-bibliotheque */
        puts("Pour continuer frapper une touche...");
        getch();  /* Attente d'une saisie clavier */
}
```

Le langage C distingue les minuscules, des majuscules. Les mots réservés du langage C doivent être écrits **en minuscules.**

On a introduit dans ce programme la notion d'interface homme/machine (IHM) :

- L'utilisateur visualise une information sur l'écran,
- L'utilisateur, par une action sur le clavier, fournit une information au programme.

Modifier le programme comme ci-dessous, puis le tester :

```
#include <stdio.h>              /* bibliotheque d'entrees-sorties standard */
#include <conio.h>
void main()
{
        int a, b, somme ;  /* déclaration de 3 variables */
        puts("BONJOUR"); /* utilisation d'une fonction-bibliotheque */
        a = 10 ; /* affectation* /
        b = 50 ;  /* affectation */
        somme = (a + b)*2 ;  /* affectation et opérateurs   */
        printf(« Voici le resultat : %d\n », somme) ;
        puts("Pour continuer frapper une touche...");
        getch();  /* Attente d'une saisie clavier */
}
```

Dans ce programme, on introduit 3 nouveaux concepts :

- La notion de déclaration de variables : les variables sont les données que manipulera le programme lors de son exécution. Ces variables sont rangées dans la mémoire vive de l'ordinateur. Elle doivent être déclarées au début du programme.
- La notion d'affectation, symbolisée par le signe =.
- La notion d'opération.

Les différents types de variables

Les entiers

Le langage C distingue plusieurs types d'entiers :

TYPE	DESCRIPTION	TAILLE MEMOIRE
int	entier standard signé	4 octets: $-2^{31} \leq n \leq 2^{31}-1$
unsigned int	entier standard positif	4 octets: $0 \leq n \leq 2^{32}$
short	entier court signé	2 octets: $-2^{15} \leq n \leq 2^{15}-1$
unsigned short	entier court positif	2 octets: $0 \leq\leq n \leq 2^{16}$
char	caractère signé	1 octet : $-2^{7} \leq\leq n \leq 2^{7}-1$
unsigned char	caractère positif	1 octet : $0 \leq n \leq 2^{8}$

Numération : En décimal les nombres s'écrivent tels que, précédés de 0x en hexadécimal.
exemple: 127 en décimal s'écrit 0x7f en hexadécimal.

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

Exemples :
Les caractères alphanumériques s'écrivent entre ' '

Le **caractère** 'b' a pour valeur 98 (son code ASCII).
Le **caractère** 22 a pour valeur 22.
Le **caractère** 127 a pour valeur 127.
Le **caractère** 257 a pour valeur 1 (ce nombre s'écrit sur 9 bits, le bit de poids fort est perdu).

Quelques constantes caractères :

CARACTERE		VALEUR (code ASCII)	NOM ASCII
'\n'	interligne	0x0a	LF
'\t'	tabulation horizontale	0x09	HT
'\v'	tabulation verticale	0x0b	VT
'\r'	retour charriot	0x0d	CR
'\f'	saut de page	0x0c	FF
'\\'	backslash	0x5c	\
'\''	cote	0x2c	'
'\"'	guillemets	0x22	"

Modifier ainsi le programme et le tester :

```
#include <stdio.h>              /* bibliotheque d'entrees-sorties standard */
#include <conio.h>
void main()
{
        int a, b, calcul ;  /* déclaration de 3 variables */
        char u, v;
        puts("BONJOUR"); /* utilisation d'une fonction-bibliotheque */
        a = 10 ; /* affectation* /
        b = 50 ;  /* affectation */
        u = 65 ;
        v = 'A' ;
        calcul = (a + b)*2 ;  /* affectation et opérateurs   */
        printf(« Voici le resultat : %d\n », calcul) ;
        printf(« 1er affichage de u : %d\n »,u) ;
        printf(« 2ème affichage de v : %c\n »,u) ;
         printf(« 1er affichage de u: %d\n »,v) ;
        printf(« 2éme  affichage de v: %c\n »,v) ;
        puts("Pour continuer frapper une touche...");
        getch();  /* Attente d'une saisie clavier */
}
```

Les réels

Un réel est composé - d'un signe - d'une mantisse - d'un exposant
Un nombre de bits est réservé en mémoire pour chaque élément.

Le langage C distingue 2 types de réels :

TYPE	DESCRIPTION	TAILLE MEMOIRE
float	réel standard	4 octets
double	réel double précision	8 octets

Les initialisations

Le langage C permet l'initialisation des variables dans la zône des déclarations :

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

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

Cette règle s'applique à tous les nombres, char, int, float ...

Sortie de nombres ou de textes à l'écran : la fonction printf

Ce n'est pas une instruction du langage C, mais une fonction de la bibliothèque stdio.h.

Exemple: affichage d'un texte:

```
printf("BONJOUR");          /* pas de retour à la ligne du curseur apres l'affichage, */
printf("BONJOUR\n");        /* affichage du texte, puis retour à la ligne du curseur.  */
```

Exercice I-2 : Tester le programme suivant et conclure.

```
#include <stdio.h>
#include <conio.h>
void main()
{
        printf("BONJOUR ");
        printf("IL FAIT BEAU\n");  /* equivalent à puts("BONJOUR";  */
        printf("BONNES VACANCES");
        puts("Pour continuer frapper une touche...");
        getch();  /* Attente d'une saisie clavier */
}
```

La fonction printf exige l'utilisation de formats de sortie, avec la structure suivante:

printf("%format",nom_de_variable);

Exercice I-3 : Affichage d'une variable de type **char:**

Tester le programe suivant et conclure.
Dans un deuxième temps, le modifier ce programme pour améliorer l'interface utilisateur.

```
#include <stdio.h>
#include <conio.h>
void main()
{
        char c;
        c =66;              /* c est le caractere alphanumerique A */
        printf("%d\n",c);   /* affichage du code ASCII en decimal */
                            /* et retour … à la ligne */
        printf("%o\n",c);   /* affichage du code ASCII en base huit
```

```
                                /* et retour … à la ligne */
        printf("%x\n",c);       /* affichage du code ASCII en hexadecimal
                                /* et retour … à la ligne */
        printf("%c\n",c);       /* affichage du caractère */
                                /* et retour à la ligne */
        puts("Pour continuer frapper une touche...");
        getch();  /* Attente d'une saisie clavier */
}
```

Exercice I-4 : Affichage multiple de structure:

printf("format1 format2 .... formatn",variable1,variable2, ...,variablen);

Tester le programe suivant et conclure:

```
#include <stdio.h>
#include <conio.h>
void main()
{
        char c;
        c ='A';     /* c est le caractere alphanumerique A */
        printf("decimal = %d  ASCII = %c\n",c,c);
        puts("Pour continuer frapper une touche...");
        getch();  /* Attente d'une saisie clavier */
}
```

Formats de sortie pour les entiers:

%d	affichage en décimal (entiers de type int),
%x	affichage en hexadécimal (entiers de type int),
%u	affichage en décimal (entiers de type unsigned int),

D'autres formats existent, consulter une documentation constructeur.

Exercice I-5 :

a et b sont des entiers, a = -21430 b = 4782, calculer et afficher a+b, a-b, a*b, a/b, a%b en format décimal, et en soignant l'interface homme/machine.

a/b donne le quotient de la division, a%b donne le reste de la division.

Exercice I-6 :

Que va-t-il se produire, à l'affichage, lors de l'exécution du programme suivant ?

```
#include <stdio.h>  /* ex I_6 */
#include <conio.h>
void main()
{
        char a = 0x80;
        unsigned char b = 0x80;
        clrscr();
        printf("a en decimal vaut: %d\n",a);
        printf("b en decimal vaut: %d\n",b);
        puts("Pour continuer frapper une touche...");
        getch();  /* Attente d'une saisie clavier */
}
```

Exercice I-7 :

En C standard, la taille des entiers est de 32 bits;
Que va-t-il se passer, à l'affichage, lors de l'exécution du programme suivant ?

```
#include <stdio.h>  /* ex I_7.C */
#include <conio.h>
void main()
{
        int a = 12345000, b = 60000000, somme;
        somme=a*b;
        printf("a*b = %d\n",somme);
        printf("a*b (en hexa) = %x\n",somme);
        puts("Pour continuer frapper une touche...");
        getch();  /* Attente d'une saisie clavier */
}
```

Format de sortie pour les réels: %f

Exercice I-8 :

a et b sont des réels, a = -21,43 b = 4,782, calculer et afficher a+b, a-b, a*b, a/b, en soignant l'interface homme/machine.

Autres fonctions de sorties

Affichage d'un caractère : La fonction **putchar** permet d'afficher un caractère:
c étant une variable de type **char**, l'écriture **putchar(c);** est équivalente à **printf("%c\n",c);**

Affichage d'un texte : La fonction **puts** permet d'afficher un texte:
l'écriture **puts("bonjour");** est équivalente à **printf("bonjour\n");**

Il vaut mieux utiliser puts et putchar si cela est possible, ces fonctions, non formatées, sont d'exécution plus rapide, et nécessitent moins de place en mémoire lors de leur chargement.

Les opérateurs

Opérateurs arithmétiques sur les réels : + - * / avec la hiérarchie habituelle.

Opérateurs arithmétiques sur les entiers : + - * / (quotient de la division) % (reste de la division) avec la hiérarchie habituelle.

Exemple particulier :

```
char c,d;
c = 'G';
d = c+'a'-'A';
```

Les caractères sont des entiers sur 8 bits, on peut donc effectuer des opérations. Sur cet exemple, on transforme la lettre majuscule G en la lettre minuscule g.

Opérateurs logiques sur les entiers :
& ET | OU ^ OU EXCLUSIF ~ COMPLEMENT A UN « DECALAGE A GAUCHE » DECALAGE A DROITE.

Exemples :

```
p = n « 3;          /* p est égale à n décalé de 3 bits à gauche */
p = n » 3;          /* p est égale à n décalé de 3 bits à droite */
```

L'opérateur **sizeof(type)** renvoie le nombre d'octets réservés en mémoire pour chaque type d'objet.

Exemple:

```
n = sizeof(char);   /* n vaut 1 */
```

Exercice I-9 : n est un entier (n = 0x1234567a), p est un entier (p = 4). Ecrire un programme qui met à 0 les p bits de poids faibles de n.

Exercice I-10 : Quels nombres va renvoyer le programme suivant ?

```
#include <stdio.h>
#include <conio.h>
void main()
{
        printf("TAILLE D'UN CARACTERE:%d\n",sizeof(char));
        printf("TAILLE D'UN ENTIER:%d\n",sizeof(int));
        printf("TAILLE D'UN REEL:%d\n",sizeof(float));
        printf("TAILLE D'UN DOUBLE:%d\n",sizeof(double));
        puts("Pour continuer frapper une touche...");
        getch();  /* Attente d'une saisie clavier */
}
```

Incrémentation, décrémentation

Le langage C autorise des écritures simplifiées pour l'incrémentation et la décrémentation de variables :

```
i =  i+1;  est équivalent à i++;
i =  i-1;  est équivalent à i--;
```

Opérateurs combinés

Le langage C autorise des écritures simplifiées lorsqu'une même variable est utilisée de chaque côté du signe = d'une affectation. Ces écritures sont à éviter lorsque l'on débute l'étude du langage C car elles nuisent à la lisibilité du programme.

```
a =  a+b;          est équivalent à    a+= b;
a =  a-b;          est équivalent à    a-= b;
a =  a & b;        est équivalent à    a&= b;
```

Les déclarations de constantes

Le langage C autorise 2 méthodes pour définir des constantes.

1ere méthode : **déclaration** d'une variable, dont la valeur sera constante pour tout le programme:

Exemple:

```
void main()
{
        const float PI = 3.14159;
        float perimetre,rayon = 8.7;
        perimetre =  2*rayon*PI;
        ....
}
```

Dans ce cas, le compilateur réserve de la place en mémoire (ici 4 octets), pour la variable pi, mais dont on ne peut changer la valeur.

2eme méthode : **définition d'un symbole** à l'aide de la directive de compilation **#define.**

Exemple:

```
#define PI = 3.14159;
void main()
{
        float perimetre,rayon = 8.7;
        perimetre =  2*rayon*PI;
        ....
}
```

Le compilateur ne réserve pas de place en mémoire. Les constantes déclarées par #define s'écrivent traditionnellement en majuscules, mais ce n'est pas une obligation.

Les conversions de types

Le langage C permet d'effectuer des opérations de conversion de type: On utilise pour cela l'opérateur de "cast" ().

Exemple et exercice I-11 :

```
#include <stdio.h>
#include <conio.h>
void main()
{
        int i=0x1234,j;
        char d,e;
        float r=89.67,s;
        j = (int)r;
        s = (float)i;
        d = (char)i;
        e = (char)r;
        printf("Conversion float -> int: %5.2f -> %d\n",r,j);
        printf("Conversion int -> float: %d -> %5.2f\n",i,s);
        printf("Conversion int -> char: %x -> %x\n",i,d);
        printf("Conversion float -> char: %5.2f -> %d\n",r,e);
        printf("Pour sortir frapper une touche ");getch();
}
```

Corrigés des exercices

Exercice I-5 :

```
#include <stdio.h>
#include <conio.h>
void main()
{
        int a,b;
        a= -21430;
        b= 4782;
        printf("A + B = %d\n",a+b);
        printf("A - B = %d\n",a-b);
        printf("A x B = %d\n",a*b);
        printf("A sur B = %d\n",a/b);
        printf("A mod B = %d\n",a%b);
        puts("Pour continuer frapper une touche...");
        getch();  /* Attente d'une saisie clavier */
}
```

Exercice I-6 :

a en décimal vaut -128 b en décimal vaut 128

En C, le type **char** désigne un entier codé sur 8 bits, cela donne -128 <= char <=+127 et 0<= unsigned char <= 255

Exercice I-8 :

```
#include <stdio.h> /* EXI_8*/
#include <conio.h>
void main()
{
        float a,b;
        a= -21.43;
        b= 4.782;
        printf("A + B = %f\n",a+b);
        printf("A - B = %f\n",a-b);
        printf("A x B = %f\n",a*b);
        printf("A sur B = %f\n",a/b);
        puts("Pour continuer frapper une touche...");
        getch();  /* Attente d'une saisie clavier */
}
```

Exercice I-9 :

```
#include <stdio.h>
#include <conio.h>
main()
{
        int n,p,masque;
        clrscr();
        n = 45;
        p = 4;
        printf("valeur de n avant modification:%x\n",n);
        masque = ~0;        /* que des 1 */
        masque = masque « p;
        n = n & masque;
        printf("n modifié vaut:%x\n",n);
}
```

Exercice I-10 :

En C standard: sizeof(char) vaut 1 sizeof(int) vaut 4 sizeof(float) vaut 4 sizeof(double) vaut 8.

SAISIE DE NOMBRES ET DE CARACTERES AU CLAVIER

La fonction getch

La fonction getch, appartenant à la bibliothèque conio.h permet la saisie clavier d' un caractère alphanumérique, **sans écho écran**. La saisie s'arrête dès que le caractère a été frappé.

La fonction getch n'est pas définie dans la norme ANSI mais elle peut exister dans la bibliothèque d'autres compilateurs.

On peut utiliser getch de deux façons:

- **sans retour de variable au programme :**

  Exemple :

```
printf("POUR CONTINUER FRAPPER UNE TOUCHE ");
getch();
```

- **avec retour de variable au programme :**

  Exemple :

```
char alpha;
printf("ENTRER UN CARACTERE (ATTENTION PAS DE RETURN) ");
alpha = getch();
printf("\nVOICI CE CARACTERE: %c",alpha);
```

Les parenthèses vides de getch() signifient qu'aucun paramètre n'est passé à cette fonction par le programme.

La fonction scanf

La fonction scanf, appartenant à la bibliothèque stdio.h, permet la saisie clavier de n'importe quel type de variable.

Les variables à saisir sont formatées, le nom de la variable est précédé du symbole & désignant l'adresse de la variable (On reverra ce symbole dans le chapitre sur les pointeurs).
La saisie s'arrête avec "RETURN" (c'est à dire LF), les éléments saisis s'affichent à l'écran (**saisie avec écho écran**).

Tous les éléments saisis après un **caractère d'espacement** (espace, tabulation) sont ignorés.

Exemples :

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

Remarque : Si l'utilisateur ne respecte pas le format indiqué dans scanf, la saisie est ignorée. Aucune erreur n'est générée.

Exemple :

```
char alpha;
scanf("%d",&alpha);
```

Si l'utilisateur saisie 97 tout va bien, alpha devient le caractère dont le code ASCII vaut 97.
Si l'utilisateur saisie a, sa saisie est ignorée.

Exercice II_1 :
Saisir un caractère au clavier, afficher son code ASCII à l'écran. Soigner l'affichage.

Exercice II_2 :
Saisir un nombre entier en décimal au clavier, l'afficher en hexadécimal à l'écran. Soigner l'affichage.

Exercice II_3 : Que va-t-il se passer lors de l'exécution du programme suivant, si l'utilisateur saisit 67?

```
#include <stdio.h>
#include <conio.h>
void main()
{
        char c;
        printf("ENTRER UN CARACTERE: ");
        scanf("%c",&c);
        printf("VOICI SON CODE ASCII: %d\n",c);
        printf("POUR CONTINUER FRAPPER UNE TOUCHE ");
        getch();
}
```

Notion de flux d'entrée

Lorsque l'on saisit au clavier une suite de caractères terminés par "RETURN" ces caractères sont rangés dans un tampon (ou buffer) de type FIFO (First In/First Out), le dernier caractère rangé dans le tampon est LF (code ASCII 0x0A).
Cette suite de caractères est appelée **flux d'entrée.**
La taille du tampon dépend de la machine et du compilateur utilisés. Sur un PC et en TURBOC, la taille du tampon est de 127 caractères.

Une compilation du programme vide le tampon.

La fonction scanf – deuxième approche

La fonction scanf ne se comporte pas tout à fait comme décrit plus haut. Si le tampon est vide, tout se passe comme précédemment décrit.
Au contraire, si le tampon n'est pas vide, la fonction scanf en teste le premier élément, s'il correspond au format de la variable invoquée, le tampon perd cet élément et la variable en prend la valeur.

Tout caractère ou nombre saisi au clavier et non pris en compte par la fonction scanf est rangé dans le tampon.

Exemple et Exercice II-4 :

```
#include <stdio.h>
#include <conio.h>
void main()
{
        char c1,c2;
        printf("ENTRER UN CARACTERE: ");
        scanf("%c",&c1);
        printf("VOICI SON CODE ASCII EN HEXADECIMAL: %x\n",c1);
        printf("ENTRER UN AUTRE CARACTERE: ");
        scanf("%c",&c2);
        printf("VOICI SON CODE ASCII EN HEXADECIMAL: %x\n",c2);
        printf("POUR CONTINUER FRAPPER UNE TOUCHE ");
        getch();
}
```

Si l'utilisateur saisit K pour c1, le programme donnera l'écran d'exécution suivant :

ENTRER UN CARACTERE: K
VOICI SON CODE ASCII EN HEXADECIMAL: 4b
ENTRER UN AUTRE CARACTERE: VOICI SON CODE ASCII EN HEXADECIMAL: a

Lors de la saisie de K, le caractere LF est rangé dans le tampon. Lors du deuxième appel à scanf, le tampon n'est pas vide, l'utilisateur ne peut effectuer sa saisie clavier, le code ascii de LF est affiché à l'écran.
A l'issue de l'exécution, le tampon est vide.

Exercice II_5 : Le programme suivant s'exécute-t-il "correctement" ? Que contient le tampon à l'issue de l'exécution ?

```
#include <stdio.h>
#include <conio.h>
void main()
{
        char c;
        int i;
        printf("ENTRER UN CARACTERE: ");
        scanf("%c",&c);
        printf("VOICI SON CODE ASCII EN HEXADECIMAL: %x\n",c);
        printf("ENTRER UN NOMBRE: ");
        scanf("%d",&i);
        printf("VOICI CE NOMBRE EN HEXADECIMAL: %x\n",i);
        printf("POUR CONTINUER FRAPPER UNE TOUCHE ");
        getch();
}
```

Exercice II_6 : Le programme suivant s'exécute-t-il "correctement" ? Que contient le tampon à l'issue de l'exécution ?

```
#include <stdio.h>
#include <conio.h>
void main()
{
        char c;
        int i;
        printf("ENTRER UN NOMBRE: ");
        scanf("%d",&i);
        printf("VOICI CE NOMBRE EN HEXADECIMAL: %x\n",i);
        printf("ENTRER UN CARACTERE: ");
        scanf("%c",&c);
        printf("VOICI SON CODE ASCII EN HEXADECIMAL: %x\n",c);
        printf("POUR CONTINUER FRAPPER UNE TOUCHE ");
        getch();
}
```

Exercice II_7 : Dans l'exercice II_4 que se passe-t-il si, lors de la première saisie, l'utilisateur tape 67 ?

Remarque : En TURBOC la fonction **flushall()** permet de vider le tampon d'entrée. En l'invoquant après un appel à scanf, on se débarrasse des problèmes de flux d'entrée.

La fonction getchar

La fonction getchar pemet la saisie d'un caractère (char). Elle appartient à la bibliothèque stdio.h. Les 2 écritures suivantes sont équivalentes :

```
char c;
printf("ENTRER UN CARACTERE: ");
scanf("%c",&c);
```

```
char c;
printf("ENTRER UN CARACTERE: ");
c = getchar();
```

Non formatée, la fonction getchar est moins gourmande en place mémoire que scanf. Il vaut mieux l'utiliser quand cela est possible; getchar utilise le flux d'entrée exactement comme scanf.

Corrigés des exercices

Exercice II_1 :

```
#include <stdio.h>
#include <conio.h>
void main()
{
        char c;
        printf("ENTRER UN CARACTERE: ");
        scanf("%c",&c);
        printf("VOICI SON CODE ASCII EN DECIMAL: %d\n",c);
        puts("Pour continuer frapper une touche...");
        getch();
}
```

Exercice II_2 :

```
#include <stdio.h>
#include <conio.h>
void main()
{
        int nombre;
        printf("ENTRER UN NOMBRE ENTIER: ");
        scanf("%d",&nombre);
        printf("VOICI CE NOMBRE EN HEXADECIMAL: %x\n",nombre);
        printf("POUR CONTINUER FRAPPER UNE TOUCHE ");
        getch();
}
```

Exercice II_3 : Seul le **caractère 6** est pris en compte. L'affichage suivant la saisie donnera 54, c'est à dire le code ASCII de 6.

Exercice II_5 : Oui car lors du deuxième appel à scanf, le programme attend un entier(int), alors que le tampon ne contient qu'un caractère (char).
A l'issue de l'exécution le tampon contient les deux caractères LF.

Exercice II_6 : Non car à l'issue de la première saisie, le tampon contient le caractère LF qui sera lu lors du deuxième appel à scanf. Apres exécution du programme, le tampon est vide.

Exercice II_7 : L'affichage de c1 en hexadécimal donne 36 c'est à dire le code ASCII de 6, l'utilisateur ne peut saisir c2, l'affichage de c2 en hexadécimal donne 37 c'est à dire le code ASCII de 7.

LES BOUCLES

Les opérateurs logiques

test d'égalité :

```
if (a==b)            /*" si a égal b"*/
```

test de non égalité :

```
if (a!=b)            /*" si a différent de b"*/
```

tests de relation d'ordre :

```
if (a<b)
if (a<=b)
if (a>b)
if (a>=b)
```

test de ET LOGIQUE :

```
if ((expression1) && (expression2))     /*" si l'expression1 ET l'expression2 sont vraies "*/
```

test de OU LOGIQUE :

```
if ((expression1) || (expression2))     /*" si l'expression1 OU l'expression2 est vraie "*/
```

test de NON LOGIQUE :

```
if (!(expression1))                     /*" si l'expression1 est fausse "*/
```

Toutes les combinaisons sont possibles entre ces tests.

Exercice III-1: L'utilisateur saisit un caractère, le programme teste s'il s'agit d'une lettre majuscule, si oui il renvoie cette lettre en minuscule, sinon il renvoie un message d'erreur.

Le langage C admet des écritures contractées dans les expressions de test:

```
char reponse;
printf("Voulez-vous jouer ?");
reponse = getchar();
if(reponse == 'o')
printf("BONJOUR\n");
else printf("TANT-PIS"\n);
```

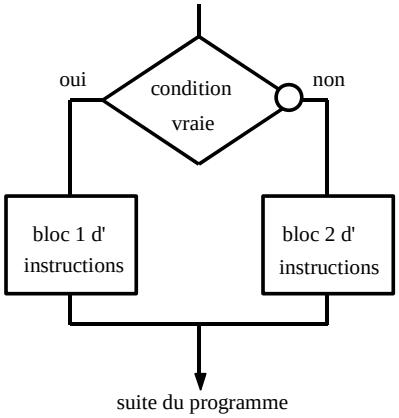
est équivalent à

```
char reponse;
printf("Voulez-vous jouer ?");

if((reponse = getchar()) =='o')
printf("BONJOUR\n");
else printf("TANT-PIS"\n);
```

L'instruction SI…ALORS…SINON

Il s'agit de l'instruction : **si (expression conditionnelle vraie)**
alors {BLOC 1 D'INSTRUCTIONS}
sinon {BLOC 2 D'INSTRUCTIONS}

Organigramme :

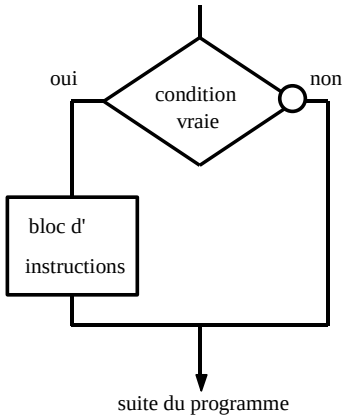


Syntaxe en C :

```
if (expression)
{
        .....;          /* bloc 1 d'instructions */
        .....;
        .....;
}
else
{
        .....;          /* bloc 2 d'instructions */
        .....;
        .....;
}
```

Le bloc "sinon" est optionnel :**si (expression vraie)**
alors {BLOC D'INSTRUCTIONS}

Organigramme :



Syntaxe en C :

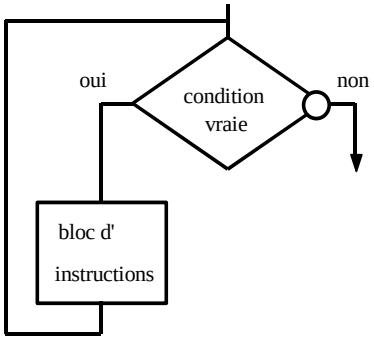
```
if (expression)
{
        .....;          /* bloc d'instructions */
        .....;
        .....;
}
```

Remarque : les {} ne sont pas nécessaires lorsque les blocs ne comportent qu'une seule instruction.

La boucle TANT QUE ... FAIRE ...

Il s'agit de l'instruction : **tant que (expression vraie)**
faire{BLOC D'INSTRUCTIONS}

Organigramme :



Syntaxe en C :

```
while (expression)
{
        .....;          /* bloc d'instructions */
        .....;
        .....;
}
```

Le test se fait **d'abord,** le bloc d'instructions n'est pas forcément exécuté.

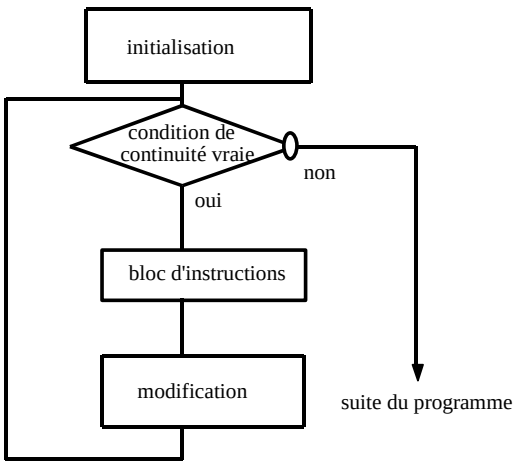
Remarque : les {} ne sont pas nécessaires lorsque le bloc ne comporte qu'une seule instruction.

Remarque : On peut rencontrer la construction suivante: **while (expression);** terminée par un ; et sans la présence du bloc d'instructions. Cette construction signifie: **"tant que l'expression est vraie attendre".**

L'instruction POUR ...

Il s'agit de l'instruction : **pour (initialisation; condition de continuité vraie;modification)**
{BLOC D'INSTRUCTIONS}

Organigramme:



Syntaxe en C :

```
for(initialisation ; condition de continuité ; modification)
{
        .....;          /* bloc d'instructions */
        .....;
        .....;
}
```

Remarques :

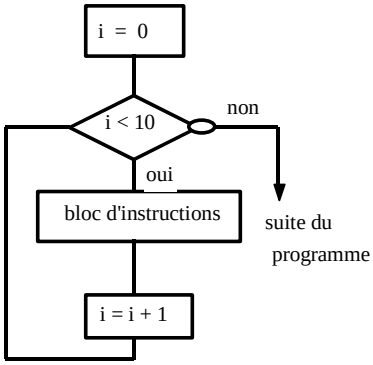
- Les {} ne sont pas nécessaires lorsque le bloc ne comporte qu'une seule instruction.
- Les 3 instructions du for ne portent pas forcément sur la même variable.
- Une instruction peut être omise, mais pas les ;

Exemples :

- Le programme :

```
for(i = 0 ; i<10 ; i++)
{
        .....;          /* bloc d'instructions */
        .....;
        .....;
}
```

correspond à l'organigramme suivant :



- La boucle :

```
for(;;)
{
        .....;          /* bloc d'instructions */
        .....;
        .....;
}
```

est une boucle infinie (répétition infinie du bloc d'instructions).

- Utilisation de variables différentes :

```
resultat = 0;
for(i = 0 ; resultat<30 ; i++)
{
        .....;          /* bloc d'instructions */
        .....;
        .....;
        resultat = resultat + 2*i;
}
```

Exercice III-2 :

Saisir un entier, calculer n!
Utiliser une boucle while puis une boucle for.
Quelle est la plus grande valeur possible de n, si n est déclaré int, puis unsigned ?

L'instruction AU CAS OU ... FAIRE ...

L'instruction switch permet des choix multiples **uniquement sur des entiers (int) ou des caractères (char).**

Syntaxe :

```
switch(variable de type char ou int)    /*au cas où la variable vaut:*/
{
case  valeur1:                          /*cette valeur1: executer ce bloc d'instructions.*/
            .......;
            break;
case valeur2:                           /*cette valeur2:  executer ce bloc d'instructions.*/
             ........;
             break;
.
.                                       /*etc ...*/
.
case default:                           /*aucune des val. précédentes: executer ce bloc*/
            ........;                   /*d'instructions, pas de "break" ici.*/
}
```

le bloc "default" n'est pas obligatoire.

L'instruction switch correspond à une cascade d'instructions if ...else

Exemple :

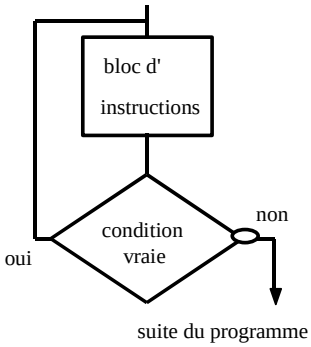
Cette instruction est commode pour fabriquer des "menus":

```
char choix;
printf("LISTE PAR GROUPE TAPER 1\n");
printf("LISTE ALPHABETIQUE TAPER 2\n");
printf("POUR SORTIR TAPER S\n");
printf("\nVOTRE CHOIX: ");
choix = getchar();
switch(choix)
{
case '1':  .......;
            .......;
            break;

case '2':  ......;
            ......;
            break;

case 'S':  printf("\nFIN DU PROGRAMME ....");
            break;

default;  printf("\nCE CHOIX N'EST PAS PREVU ");  /* pas de break ici */
}
```

L'instruction REPETER ... TANT QUE ...

Il s'agit de l'instruction : **répéter{BLOC D'INSTRUCTIONS}**
tant que (expression vraie)

Organigramme :



Syntaxe en C:

```
do
{
        .....;          /* bloc d'instructions */
        .....;
        .....;
}
while (expression);
```

Le test se faisant **après,** le bloc est exécuté au moins une fois.

Remarque: les {} ne sont pas nécessaires lorsque le bloc ne comporte qu'une seule instruction.

Compléments sur les tests

En langage C, **une expression nulle de type entier (int) est fausse, une expression non nulle de type entier (int) est vraie.**

Exemples :

```
int a,b,c,delta;                    est équivalent à    int a,b,c,delta;
delta = b*b-4*a*c;                                      delta = b*b-4*a*c;
if(delta != 0)                                          if(delta)
{ ..... }                                               { ..... }
```

```
int a,b,c,delta;                    est équivalent à    int a,b,c,delta;
delta = b*b-4*a*c;                                      delta = b*b-4*a*c;
if(delta == 0)                                          if(!delta)
{ ..... }                                               {.....}
```

Exercices récapitulatifs

Exercice III_3 : résoudre $ax^2 + bx + c = 0$.

Exercice III_4 : Saisir une suite de caractères, compter et afficher le nombre de lettres e et d'espaces. Utiliser les propriétés du tampon.

Exercice III_5 : La fonction kbhit appartient à la bibiothèque conio.h. Une fonction équivalente peut exister avec d'autres compilateurs. La fonction kbhit teste si un caractère a été frappé au clavier. Tant que ce n'est pas vrai kbhit renvoie 0 (ceci signifie que la valeur de la fonction kbhit est 0).

Exemple :

```
while(kbhit() == 0)          /*tant qu'aucun caractère n'a été frappé exécuter la boucle*/
        { ..... }
```

Cette écriture est équivalente à :

```
while(!kbhit());             /* tant que kbhit est faux,  exécuter la boucle */
        { ..... }
```

Ecrire un programme qui affiche le carré des entiers 1, 2, 3 ....., toutes les 500 ms tant qu'aucun caractère n'a été frappé au clavier. Générer la temporisation à l'aide d'une boucle for et d'un décompteur.

Corrigés des exercices

Exercice III-1 :

```
#include <stdio.h>
#include<conio.h>
void main()
{
        char c;
        printf("ENTRER UNE LETTRE:");
        c = getchar();
        if((c>='A') && (c<='Z')) printf("CETTE LETTRE EN MINUSCULE: %c\n",c);
        else printf("CE N'EST PAS UNE LETTRE MAJUSCULE\n");
        printf("POUR CONTINUER FRAPPER UNE TOUCHE ");
        getch();
}
```

Exercice III-2 :

```
#include <stdio.h>
#include <conio.h>
void main()
{
        int n,i,fac= 1;
        printf("ENTRER UN ENTIER: ");scanf("%d",&n);
        for (i=1;i<=n;i++) fac= fac * i;
        printf("\nn = %d  n! = %d",n,fac);
        printf("\nPOUR CONTINUER FRAPPER UNE TOUCHE");
        getch();
}
```

Les entiers sont des nombres de 32 bits:
n int: n! maximum= 12!
n unsigned: n! maximum = 12!

Exercice III_3 :

```
#include <stdio.h>
#include <conio.h>
#include <math.h> /* contient la fonction racine */
void main()
{
        float a,b,c,delta,x1,x2;

        /* saisie de A,B,C */
        printf("\t\t\tRESOLUTION DE L'EQUATION DU SECOND DEGRE\n");
        printf("\t\t\t              2\n");
        printf("\t\t\t           AX +BX+C=0\n\n\n");
        printf("SAISIR A B C SEPARES PAR RETURN\n");
        printf("A = ");scanf("%f",&a);
        printf("B = ");scanf("%f",&b);
        printf("C = ");scanf("%f",&c);

        /* debut du calcul */

        /* cas particuliers */
        if((a==0)&&(b==0)&&(c==0))printf("INFINITE DE SOLUTIONS\n");
        if((a==0)&&(b==0)&&(c!=0))printf("PAS DE SOLUTIONS\n");
        if((a==0)&&(b!=0))printf("UNE SOLUTION: X= %f\n",-c/b);

        /*cas general */
        if(a!=0)
        {
```

```
                delta = b*b-4*a*c;
                printf("DELTA= %f\n",delta);
                if(delta<0) printf("DELTA NEGATIF PAS DE SOLUTION\n");
                else
                {
                        if(delta==0)printf("DELTA NUL, UNE SOLUTION X= %f\n",-b/2/a);
                        else
                        {
                                x1= (-b+sqrt(delta))/2/a;
                                x2= (-b-sqrt(delta))/2/a;
                                printf("DELTA POSITIF DEUX SOLUTIONS\n");
                                printf("X1= %f  X2= %f\n",x1,x2);
                        }
                }
        }

        /* calculs termines */
        printf("\n\nPOUR CONTINUER FRAPPER UNE TOUCHE");
        getch();
}
```

Exercice III_4 :

```
#include <stdio.h>
#include <conio.h>
void main()
{
        char c,compt_espace= 0,compt_e= 0;
        printf("ENTRER UNE PHRASE:\n");     /* l'utilisateur saisit la totalite de sa phrase */

        while((c=getchar())!='\n')                  /* lors du 1er passage, getchar ne prend */
                                                    /* en compte que le 1er caractere */
        {                                           /* les autres sont ranges dans le tampon */
                if(c=='e')compt_e++;                /* et recuperes par getchar lors */
                                                    /* des autres passages */
                if(c==' ')compt_espace++;
        }

        printf("NOMBRE DE e: %d\n",compt_e);
        printf("NOMBRE D'ESPACE: %d\n",compt_espace);
        printf("POUR SORTIR FRAPPER UNE TOUCHE ");
        getch();
}
```

Exercice III_5 :

```
#include <stdio.h>
#include <conio.h>
void main()
{
        int i = 0;
        float x,tempo=5000000;
        printf("POUR SORTIR DE CE PROGRAMME FRAPPER UNE TOUCHE ...\n");
        do
        {
                printf("i = %d i*i = %d\n",i,i*i);
                for(x=tempo;x>0;x--);
                i++;
        }
        while(kbhit()==0);      /* on peut aussi écrire while(!kbhit()); */
}
```

UTILISATION D'UNE BIBLIOTHEQUE

Ce petit chapitre vise à expliquer comment se servir d'une bibliothèque de fonctions. On prendra quelques exemples dans la bibliothèque de BORLAND C++.

Notion de prototype

Les fichiers de type ".h" (conio.h, dos.h stdio.h etc...), appelés ***fichiers d'en tête*** contiennent la définition des ***prototypes*** des fonctions utilisées dans le programme. Le prototype précise la syntaxe de la fonction: son nom, les paramètres éventuels à passer, la valeur éventuelle retournée au programme.
Grâce aux lignes "#include", le compilateur lit les fichiers de type ".h" et vérifie que la syntaxe de l'appel à la fonction est correcte.

Fonction ne renvoyant rien au programme

Ce sont les fonctions de type **void.**
Exemple :

```
clrscr() ;
```

fonction	Efface l'écran de la fenêtre dos.
prototype	void clrscr();
prototype dans	conio.h et donc bibliothèque à charger.

Une fonction ne renvoyant rien (de type void) s'écrit telle que. Ici pas de passage d'arguments.
Exemple :

```
clrscr();              /* efface l'écran */
printf("BONJOUR\n");
printf("AU REVOIR\n");
```

Fonction renvoyant une valeur au programme

Ce sont les fonctions de type **autre que void.** Elles renvoient au programme une valeur dont le type est précisé dans la documentation.
Exemple :

```
Kbhit() ;
```

fonction	Teste une éventuelle frappe au clavier
prototype	int kbhit();
prototype dans	conio.h et donc bibliothèque à charger.

La fonction kbhit renvoie un entier au programme. Cet entier vaut 0, tant qu'il n'y a pas eu de frappe clavier. On ne peut donc pas écrire la fonction telle que. Il faut la traiter comme une variable de type int.
ex: while(kbhit ()== 0); /* tend que kbhit vaut 0, attendre */

Fonction avec passage de paramètre

Exemple :

```
log(…)
```

fonction	Fonction logarithme népérien..
prototype	double log(double);
prototype dans	math.h et donc bibliothèque à charger.

La fonction log, renvoie au programme un réel. On traite la fonction comme une variable de type double. Il faut lui passer un paramètre de type double.

Exemple :

```
double x,y;
printf("SAISIR x: ");
scanf("%f",&x);
y = log(x);
printf("log(x) = %f\n",y);
```

Exercice IV_1 : En utilisant randomize et random jouer au 421.

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <conio.h>
void main()
{
        char c;
        int n1,n2,n3;
        printf("JEU DU 421\n");
        randomize();
        do
        {
                clrscr() ;
                /* LANCEMENT DES DES */
                printf("LANCER LES DES EN FRAPPANT UNE TOUCHE: ");
                getch();
                n1 = random(6) + 1;
                n2 = random(6) + 1;
                n3 = random(6) + 1;
                printf("\n VOICI LES DES: %1d %1d %1d\n",n1,n2,n3);

                /* TEST */
                if(((n1==4) && (n2==2) && (n3 ==1))||
                ((n1==4) && (n2==1) && (n3 ==2))||
                ((n1==2) && (n2==4) && (n3 ==1))||
                ((n1==2) && (n2==1) && (n3 ==4))||
                ((n1==1) && (n2==2) && (n3 ==4))||
                ((n1==1) && (n2==4) && (n3 ==2))) printf("GAGNE !\n");
                else printf("PERDU !\n");

                printf("\nPOUR REJOUER FRAPPER O SINON UNE TOUCHE
                QUELCONQUE\n");
                c = getch();
        }
        while((c=='O')||(c=='o'));
}
```

LES POINTEURS

L'étude des pointeurs montre l'adaptation du langage C à la conduite de processus. On verra dans ce chapitre et les suivants la puissance de cette notion par ailleurs de concept simple pour un informaticien industriel.

L'opérateur adresse &

L'opérateur adresse & retourne l'adresse d'une variable en mémoire.

Exemple :

```
int i = 8;
printf("VOICI i: %d\n",i);
printf("VOICI SON ADRESSE EN HEXADECIMAL: %p\n",&i);
```

On remarque que le format d'une adresse est **%p** (hexadécimal) ou **%d** (décimal) dans printf.

Exercice V_1 : Exécuter l'exemple précédent, et indiquer les cases-mémoire occupées par la variable i.

Les pointeurs

Définition : Un pointeur est une adresse mémoire. On dit que le pointeur pointe sur cette adresse.

Déclaration des pointeurs

Une variable de type pointeur se déclare à l'aide de l'objet pointé précédé du symbole *** (opérateur d'indirection).**

Exemple :

```
char *pc;          /*pc est un pointeur pointant sur un objet de type char*/
int *pi;           /*pi est un pointeur pointant sur un objet de type int*/
float *pr;         /*pr est un pointeur pointant sur un objet de type float*/
```

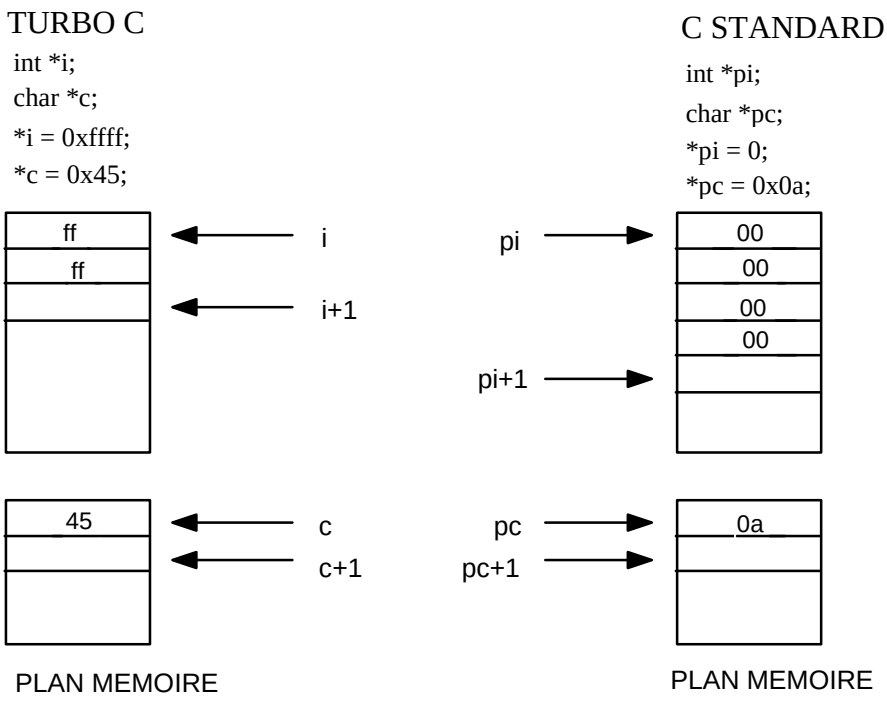
L'opérateur * désigne en fait le contenu de l'adresse.

Exemple :

```
char *pc;
*pc = 34 ;
printf("CONTENU DE LA CASE MEMOIRE: %c\n",*pc);
printf("VALEUR DE L'ADRESSE EN HEXADECIMAL: %p\n",pc);
```

Arithmétique des pointeurs

On peut essentiellemet **déplacer** un pointeur dans un plan mémoire à l'aide des opérateurs d'addition, de soustraction, d'incrémentation, de décrémentation. On ne peut le déplacer que **d'un nombre de cases mémoire multiple du nombre de cases réservées en mémoire pour la variable sur laquelle il pointe.**



Exemples :

```
int *pi;                    /*  pi pointe sur un objet de type entier */
float *pr;                  /*  pr pointe sur un objet de type réel */
char *pc;                   /*  pc pointe sur un objet de type caractère */

*pi = 421;                  /*  421 est le contenu de la case mémoire p et des 3 suivantes */
*(pi+1) = 53;               /* on range 53  4 cases mémoire plus loin   */
*(pi+2) = 0xabcd;           /* on range 0xabcd  8 cases mémoire plus loin */
*pr = 45.7;                 /* 45,7 est rangé dans la case mémoire r et les 3 suivantes  */
pr++;                       /* incrémente la valeur du pointeur r (de 4 cases mémoire) */
printf("L'ADRESSE r VAUT: %p\n",pr);          /* affichage de la valeur de l'adresse r */

*pc = 'j';                  /* le contenu de la case mémoire c est le code ASCII de 'j' */
pc--;                       /* décrémente la valeur du pointeur c (d'une case mémoire) */
```

Allocation dynamique

Lorsque l'on déclare une variable char, int, float .... un nombre de cases mémoire bien défini est **réservé** pour cette variable. Il n'en est pas de même avec les pointeurs.

Exemple :

```
char *pc;
*pc = 'a';          /* le code ASCII de a est rangé dans la case mémoire pointée par c */
*(pc+1) = 'b';      /* le code ASCII de b  est rangé une case mémoire plus loin */
*(pc+2) = 'c';      /* le code ASCII de c est rangé une case mémoire plus loin */
*(pc+3) = 'd';      /* le code ASCII de d est rangé une case mémoire plus loin */
```

Dans cet exemple, le compilateur a attribué une valeur au pointeur c, les adresses suivantes sont donc bien définies; mais le compilateur n'a pas **réservé** ces places: il pourra très bien les attribuer un peu plus tard à d'autres variables. Le contenu des cases mémoires c c+1 c+2 c+3 sera donc perdu.
Ceci peut provoquer un blocage du système sous WINDOWS.

Il existe en langage C, des fonctions permettant **d'allouer de la place en mémoire** à un pointeur.
Il faut absolument les utiliser dès que l'on travaille avec les pointeurs.

Exemple : la fonction **malloc**

```
char *pc;
int *pi,*pj,*pk;
float *pr;
pc = (char*)malloc(10);        /* on réserve 10 cases mémoire, soit la place pour 10 caractères */
pi = (int*)malloc(16);                /* on réserve 16 cases mémoire, soit la place pour 4 entiers
*/
 pr = (float*)malloc(24);      /* on réserve 24 places en mémoire, soit la place pour 6 réels */
pj = (int*)malloc(sizeof(int)); /* on réserve la taille d'un entier en mémoire */
pk = (int*)malloc(3*sizeof(int));          /* on réserve la place en mémoire pour 3 entiers */
```

Libération de la mémoire : la fonction **free**

```
free(pi);                      /* on libère la place précédemment réservée pour i */
free(pr);                      /* on libère la place précédemment réservée pour r */
```

Exercice V_2 :

adr1 et adr2 sont des pointeurs pointant sur des réels. Le contenu de adr1 vaut -45,78; le contenu de adr2 vaut 678,89. Ecrire un programme qui affiche les valeurs de adr1, adr2 et de leur contenu.

Affectation d'une valeur à un pointeur

Dans les exemples précédents, l'utilisateur **ne choisit pas** la valeur des adresses mémoire **attribuées par le compilateur à chaque variable.** L'utilisateur se contente de **lire** la valeur de ces adresses en l' affichant sur l'écran.

On ne peut pas affecter directement une valeur à un pointeur:

l'écriture suivante est interdite:

```
char *pc;
pc = 0xfffe;
```

On peut cependant être amené à **définir par programmation** la valeur d'une adresse. On utilise pour cela l'opérateur de "cast", jeu de deux parenthèses.

- Par exemple pour adresser un périphérique (adressage physique),
- Par exemple pour contrôler la zône DATA dans un plan mémoire.

Exemples :

```
char *pc;
pc = (char*)0x1000;            /* p est l'adresse 0x1000 et pointe sur un caractère */

int *pi;
pi = (int*)0xfffa;             /* i est l'adresse 0xfffa et pointe sur un entier */
```

Lorsqu'on utilise une fonction d'allocation dynamique on ne peut affecter de valeur au pointeur à l'aide de l'opérateur de cast.

Ces 2 premiers exemples sont à proscrire sous WINDOWS.

Exercice V_3 :

pi est un pointeur sur un entier; pi vaut 0x5000 et son contenu vaut 300. Ecrire le programme correspondant (programme dangereux sous WONDOWS).

L'opérateur de "cast", permet d'autre part, à des pointeurs de types différent de pointer sur la même adresse.

Exemple:
```
char *pc; /* pc pointe sur un objet de type caractère */
int *pi;          /* pi pointe sur un objet de type entier */
pi = (int*)malloc(4) ;  /* allocation dynamique pour i */
pc = (char*)i;    /* c et i pointent sur la même adresse, c sur un caractère  */
```

Exercice V_4 :

adr_i est un pointeur de type entier; son contenu i vaut 0x12345678. A l'aide d'une conversion de type de pointeur, écrire un programme montrant le rangement des 4 octets en mémoire.

Exercice V_5 :

Saisir un texte. Ranger les caractères en mémoire. Lire le contenu de la mémoire et y compter le nombre d'espaces et de lettres e .

Exercice V_6 :

Saisir 6 entiers et les ranger à partir de l'adresse adr_deb. Rechercher le maximum, l'afficher ainsi que son adresse.

Petit retour à la fonction scanf

On a vu au chapitre 2 que pour saisir un caractère ou un nombre, on donne en fait l'adresse de ce caractère:

Exemple :
```
char c;
printf("TAPER UNE LETTRE: ");
scanf("%c",&c);
```

On saisit ici le contenu de l'adresse &c c'est à dire le caractère c lui-même.

On peut donc aussi procéder ainsi :
```
char *adr;
printf("TAPER UNE LETTRE: ");
scanf("%c",adr);
```

On saisit ici le contenu de l'adresse adr.

Cette méthode s'applique aux caractères (char), aux entiers (int), aux réels (float).

Corrigé des exercices

Exercice V_2 :

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
void main()
{
        float *adr1,*adr2;
        adr1 = (float*)malloc(4);
        adr2 = (float*)malloc(4);
        *adr1 = -45.78;
        *adr2 = 678.89;
        printf("adr1 = %p adr2 = %p r1 = %f r2 = %f\n",adr1,adr2,*adr1,*adr2);
        free(adr1);
        free(adr2);
        printf("\nPOUR CONTINUER FRAPPER UNE TOUCHE ");
        getch();
}
```

Exercice V_3 :

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
void main()
{
        int *i;
        i = (int*)malloc(4);
        *i = 300;
        printf(" adresse = %p   variable = %d\n",i,*i);
        free(i);
        printf("\nPOUR CONTINUER FRAPPER UNE TOUCHE ");
        getch();
}
```

Exercice V_4 :

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
void main()
{
        char *adr_c;
        int *adr_i,i=0x12345678;
        adr_i = &i;
        adr_c = (char*)adr_i;
        printf("ADRESSE: %p CONTENU: %x\n",adr_c,*adr_c);
        printf("ADRESSE: %p CONTENU: %x\n",adr_c+1,*(adr_c+1));
        printf("ADRESSE: %p CONTENU: %x\n",adr_c+2,*(adr_c+2));
        printf("ADRESSE: %p CONTENU: %x\n",adr_c+3,*(adr_c+3));
        getch();
}
```

L'analyse de l'exécution en BORLANC C++, montre que les microprocesseurs INTEL rangent en mémoire d'abord les poids faibles d'une variable.

Exercice V_5:

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

```
void main()
{
	char *adr_deb,c;
	int i,imax,compt_e = 0,compt_sp = 0;
	adr_deb = (char*)malloc(30); /* texte d'au plus 30 caracteres */

	/* saisie et rangement du texte */
	printf("\nADRESSE DU TEXTE: %p (ATTRIBUEE PAR LE COMPILATEUR)",adr_deb);
	printf("\nENTRER UN TEXTE: ");
	for (i=0;((c=getchar())!='\n');i++) *(adr_deb + i) = c;
	imax = i;  /* borne superieure */
	/* lecture de la memoire et tri */
	for (i=0;i<imax;i++)
	{
		c = *(adr_deb+i);
		printf("\nCARACTERE: %c ADRESSE: %p",c,adr_deb+i);
		if (c=='e') compt_e++;
		if (c==' ') compt_sp++;
	}
	/* resultats */
	printf("\nNOMBRE DE e: %2d NOMBRE d'espaces: %2d\n",compt_e,compt_sp);
	free(adr_deb);
	printf("\nPOUR CONTINUER FRAPPER UNE TOUCHE ");
	getch();
}
```

Exercice V_6 :

```
#include  <stdio.h>
#include <conio.h>
#include <alloc.h>
void main()
{
	int *adr_deb,*adr_max,i,imax=6,max;
	adr_deb=(int*)malloc(4*6);
	printf("\nADRESSE DE BASE: %p (CHOISIE PAR LE PROGRAMMEUR)\n",adr_deb);
	/* saisie des nombres */
	printf("SAISIE DES NOMBRES: \n");
	for(i=0;i<imax;i++)
	{
		printf("ENTRER UN NOMBRE: ");
		scanf("%d",adr_deb+i);
	}
	/* tri */
	max = *adr_deb;
	adr_max = (int*)adr_deb;
	for(i=0;i<imax;i++)
	{
		if(*(adr_deb+i)>max)
		{
			max = *(adr_deb+i);
			adr_max =  adr_deb+i;
		}
	}
	/* resultats */
	printf("LE MAXIMUM:%d  SON ADRESSE:%p\n",max,adr_max);
	free(adr_deb);
	printf("\nPOUR CONTINUER FRAPPER UNE TOUCHE");
	getch();
}
```

LES TABLEAUX ET LES CHAINES DE CARACTERES

Les tableaux de nombres (int ou float)

Les tableaux correspondent aux matrices en mathématiques. Un tableau est caractérisé par sa taille et par ses éléments.

Les tableaux à une dimension:

Déclaration :

```
type nom[dim];
```

Exemples:

```
int compteur[10];
float nombre[20];
```

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

Exemples :

```
int compteur[10];          /*le compilateur réserve des places pour 10 entiers, soit 20 octets en
TURBOC et 40 octets en C standard.*/
float nombre[20];          /*le compilateur réserve des places pour 20 réels, soit 80 octets.*/
```

Remarque: dim est nécessairement une VALEUR NUMERIQUE. Ce ne peut être en aucun cas une combinaison des variables du programme.

Utilisation : Un élément du tableau est repéré par son indice. En langage C les tableaux commencent à l'indice 0. L'indice maximum est donc dim-1.

Appel :

```
nom[indice]
```

Exemples:

```
compteur[2] = 5;
nombre[i] = 6.789;
printf("%d",compteur[i]);
scanf("%f",&nombre[i]);
```

Tous les éléments d'un tableau (correspondant au dimensionnement maximum) ne sont pas forcément définis.

D'une façon générale, les tableaux consomment beaucoup de place en mémoire. On a donc intérêt à les dimensionner au plus juste.

Exercice VI_1 : Saisir 10 réels, les ranger dans un tableau. Calculer et afficher la moyenne et l'écart-type.

Les tableaux à plusieurs dimensions:

❖ Tableaux à deux dimensions : déclaration:

```
type nom[dim1][dim2];
```

Exemples :

```
int compteur[4][5];
float nombre[2][10];
```

Utilisation : Un élément du tableau est repéré par ses indices. En langage C les tableaux commencent aux indices 0. Les indices maximum sont donc dim1-1, dim2-1.

Appel :

```
nom[indice1][indice2]
```

Exemples :

```
compteur[2][4] = 5;
nombre[i][j] = 6.789;
printf("%d",compteur[i][j]);
scanf("%f",&nombre[i][j]);
```

Tous les éléments d'un tableau (correspondant au dimensionnement maximum) ne sont pas forcément définis.

Exercice VI_2 : Saisir une matrice d'entiers 2x2, calculer et afficher son déterminant.

❖ Tableaux à plus de deux dimensions : On procède de la même façon en ajoutant les éléments de dimensionnement ou les indices nécessaires.

Initialisation des tableaux

On peut initialiser les tableaux au moment de leur déclaration.
Exemples :

```
int liste[10] = {1,2,4,8,16,32,64,128,256,528};
float nombre[4] = {2.67,5.98,-8,0.09};
int x[2][3] = {{1,5,7},{8,4,3}};  /* 2 lignes et 3 colonnes */
```

Tableaux et pointeurs

En déclarant un tableau, on définit automatiquement un pointeur (on définit en fait l'adresse du premier élément du tableau).

Les tableaux à une dimension

Les écritures suivantes sont équivalentes :

```
int *tableau;                          int tableau[10];     /*déclaration*/
tableau = (int*)malloc(sizeof(int)*10); /*idem*/

*tableau                               tableau[0]           /*le 1er élément*/
*(tableau+i)                           tableau[i]           /*un autre élément*/
tableau                                &tableau[0]          /*adresse du 1er élément*/
(tableau + i)                          &(tableau[i])        /*adresse d'un autre élément*/
```

Il en va de même avec un tableau de réels (float).

Remarque: La déclaration d'un tableau entraine automatiquement la réservation de places en mémoire. Ce n'est pas le cas lorsque l'on déclare un pointeur. Il faut alors utiliser une fonction d'allocation dynamique comme malloc.

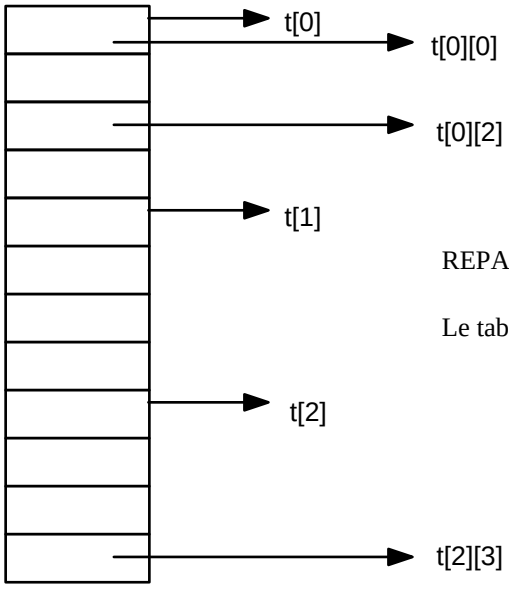
Les tableaux à plusieurs dimensions:

Un tableau à plusieurs dimensions est un pointeur de pointeur.
Exemple :

```
int t[3][4];  /*t est un pointeur de 3 tableaux de 4 éléments ou bien de 3 lignes à 4 éléments.*/
```

Les écritures suivantes sont équivalentes :

t[0]	&t[0][0]	adresse du 1er élément
t[1]	&t[1][0]	adresse du 1er élément de la 2e ligne
t[i]	&t[i][0]	adresse du 1er élément de la ième ligne
t[i]+1	&(t[i][0])+1	adresse du 1er élément de la ième +1 ligne



REPARTITION DES ELEMENTS EN MEMOIRE

Le tableau a été déclaréainsi: int t[3][4];

Exercice VI_3 :

Un programme contient la déclaration suivante:
int tab[10] = {4,12,53,19,11,60,24,12,89,19};
Compléter ce programme de sorte d'afficher les adresses des éléments du tableau.

Exercice VI_4 :

Un programme contient la déclaration suivante:
int tab[20] = {4,-2,-23,4,34,-67,8,9,-10,11, 4,12,-53,19,11,-60,24,12,89,19};
Compléter ce programme de sorte d'afficher les éléments du tableau avec la présentation suivante:

4	-2	-23	4	34
-67	8	9	-10	11
4	12	-53	19	11
-60	24	12	89	19

Les chaînes de caractères

Opérations simples

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:

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.

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.

```
char texte[10];
printf("ENTRER UN TEXTE: ");
scanf("%s",texte);  est équivalent à    gets(texte);
```

Remarque: scanf ne permet pas la saisie d'une chaîne comportant des espaces: les caractères saisis à partir de l'espace ne sont pas pris en compte (l'espace est un délimiteur au même titre que LF) mais rangés dans le tampon d'entrée. Pour saisir une chaîne de type "il fait beau", il faut utiliser gets.
A l'issue de la saisie d'une chaîne de caractères, le compilateur ajoute '\0' en mémoire après le dernier caractère.

Comme expliqué au chapitre 2, gets et scanf utilisent le flux d'entrée.

Exercice VI_5 :

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

Exercice VI_6 :

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

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*/
int strlen(char  *chaine)                      /*renvoie la longueur de la chaine ('\0' non comptabilisé)*/
void *strrev(char *chaine)                     /*inverse la chaîne et, renvoie l'adr. de la chaine inversée*/
```

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)
- négatif si la chaîne1 est inférieure à la chaîne2
- nul si les chaînes sont identiques.

Copie (string.h) :

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

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

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*/
```

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');
```

Exercice VI_7:

L'utilisateur saisit le nom d'un fichier. Le programme vérifie que celui-ci possède l'extension .PAS

Exercice VI_8:

Un oscilloscope à mémoire programmable connecté à un PC renvoie l'information suivante sous forme d'une chaîne de caractères terminée par '\0'au PC:
"CHANNELA 0 10 20 30 40 30 20 10 0 -10 -20 -30 -40 -30 -20 -10 -0"
Afficher sur l'écran la valeur des points vus comme des entiers. On simulera la présence de l'oscilloscope en initialisant une chaîne de caractères char mesures[100].

Corrigés des exercices

Exercice VI_1 :

```
#include <stdio.h>
#include <math.h>
#include <conio.h>
void main()
{
        float nombre[10],moyenne = 0,ecart_type = 0;
        int i;

        /* saisie des nombres */
        printf("SAISIR 10 NOMBRES SEPARES PAR RETURN: \n");
        for(i=0;i<10;i++)
        {
                printf("nombre[%1d] = ",i);
                scanf("%f",&nombre[i]);
        }
        /* calculs */
        for(i=0;i<10;i++)
        {
                moyenne = moyenne + nombre[i];
                ecart_type = ecart_type + nombre[i]*nombre[i];
        }
        moyenne = moyenne/10;
        ecart_type = ecart_type/10;
        ecart_type = ecart_type - moyenne*moyenne;
        ecart_type = sqrt(ecart_type);          /* racine */
        printf("MOYENNE = %f  ECART_TYPE = %f\n",moyenne,ecart_type);
        printf("POUR CONTINUER FRAPPER UNE TOUCHE: ");
        getch();
}
```

Exercice VI_2 :

```
#include <stdio.h>
#include <conio.h>
void main()
{
        int mat[2][2],det;
        /* saisie */
        printf("ENTRER SUCCESSIVEMENT LES VALEURS DEMANDEES: \n");
```

```
        printf("mat[0][0] = ");
        scanf("%d",&mat[0][0]);
        printf("mat[1][0] = ");
        scanf("%d",&mat[1][0]);
        printf("mat[0][1] = ");
        scanf("%d",&mat[0][1]);
        printf("mat[1][1] = ");
        scanf("%d",&mat[1][1]);

        /* calcul */
        det = mat[0][0]*mat[1][1]-mat[1][0]*mat[0][1];

        /* affichage */
        printf("DETERMINANT = %d\n",det);
        printf("POUR CONTINUER FRAPPER  UNE TOUCHE: ");
        getch();
}
```

Exercice VI_3 :

```
#include <stdio.h>
#include <conio.h>
void main()
{
        int i,tab[10]={4,12,53,19,11,60,24,12,89,19};
        printf("VOICI LES ELEMENTS DU TABLEAU ET LEURS ADRESSES:\n");
        for(i=0;i<10;i++)
        printf("ELEMENT N¯%1d: %2d    ADRESSE: %p\n",i,tab[i],tab+i);
        printf("POUR SORTIR FRAPPER UNE TOUCHE: ");
        getch();
}
```

Exercice VI_4 :

```
#include <stdio.h>
#include <conio.h>
void main()
{
        int i,tab[20] = {4,-2,-23,4,34,-67,8,9,-10,11, 4,12,-53,19,11,-60,24,12,89,19};
        printf("VOICI LE TABLEAU:\n\n");
        for(i=0;i<20;i++)
                if (((i+1)%5)==0) printf("\t%d \n",tab[i]);
                else printf("\t%d ",tab[i]);
        printf("\nPOUR SORTIR FRAPPER UNE TOUCHE: ");
        getch();
}
```

Exercice VI_5 :

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
void main()
{
        char i=0,*phrase;
        phrase = (char*)malloc(20);   /* reserve 20 places */
```

```
        printf("ENTRER UNE PHRASE: ");
        gets(phrase);       /* saisie */

        printf("VOICI LES ELEMENTS DE LA CHAINE ET LEUR ADRESSE\n");
        do
        {
                printf("LETTRE: %c CODE ASCII: %x ADRESSE:
                        %p\n",phrase[i],phrase[i],phrase+i);
                i++;
        }
        while(phrase[i-1]!='\0');
        free(phrase);
        printf("\nPOUR CONTINUER FRAPPER UNE TOUCHE: ");
        getch();
}
```

Exercice VI_6 :

```
 #include <stdio.h>
#include <conio.h>
#include <alloc.h>
void main()
{
        char *phrase,compt_espace = 0,compt_e = 0,i;
        phrase=(char*)malloc(20);    /* reserve 20 places */
        printf("ENTRER UNE PHRASE:");
        gets(phrase);  /* saisie */

        for(i=0;phrase[i]!='\0';i++)
        {
                if(phrase[i]=='e')compt_e++;
                if(phrase[i]==' ')compt_espace++;
        }
        printf("NOMBRE DE e: %d\n",compt_e);
        printf("NOMBRE D'ESPACES : %d\n",compt_espace);
        free(phrase);
        printf("POUR SORTIR FRAPPER UNE TOUCHE ");
        getch();
}
```

Exercice VI_7 :

```
 #include <conio.h>
#include <alloc.h>
#include <string.h>
void main()
{
        char *nom,*copie;
        int n;
        nom = (char*)malloc(30);
        copie = (char*)malloc(30);
        printf("\nNOM DU FICHIER (.PAS):");
        gets(nom);
        strcpy(copie,nom);
        strrev(copie);       /* chaine inversee */
        n = strnicmp("SAP.",copie,4);/* n vaut 0 si ,galite */
        if(n!=0)printf("\nLE FICHIER N'EST PAS DE TYPE .PAS\n");
        else printf("\nBRAVO CA MARCHE\n");
        free(nom);
```

```
        free(copie);
        printf("\nPOUR CONTINUER FRAPPER UNE TOUCHE ");
        getch();
}
```

Exercice VI_8 :

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void main()
{
        char mesures[100] =
                "CHANNELA 0 10 20 30 40 30 20 10 0 -10 -20 -30 -40 -30 -20 -10 0";
        int i,j,val[20],nombre_val=0;
        char temp[4];   /* chaine temporaire */

        /* recherche des nombres */
        for(i=9;mesures[i]!='\0';i++)
        {
                for(j=0;(mesures[i]!=' ')&&(mesures[i]!='\0');j++)
                {
                        temp[j]=mesures[i];
                        i++;
                }
                temp[j] = '\0';     /* On borne la chaine */
                val[nombre_val] = atoi(temp); /* Conversion de la chaine temporaire en nbre   */
                nombre_val++;
        }

        /* Affichage du resultat */
        clrscr();
        for(i=0;i<nombre_val;i++)printf("val[%d] = %d\n",i,val[i]);
        printf("POUR SORTIR FRAPPER UNE TOUCHE:");
        getch();
}
```

LES FONCTIONS

En langage C les sous-programmes s'appellent des **fonctions.**

L'imbrication de fonctions n'est pas autorisée en C: une fonction ne peut pas être déclarée à l'intérieur d'une autre fonction. Par contre, une fonction peut **appeler** une autre fonction. Cette dernière doit être déclarée **avant** celle qui l'appelle.

Une fonction possède un et un seul point d'entrée, mais éventuellement plusieurs points de sortie (à l'aide du mot **return**).

Une variable connue uniquement d'une fonction ou de main() est une variable locale.
Une variable connue de tout le programme est une variable globale.

Fonctions sans passage d'argument et ne renvoyant rien au programme

Une fonction ne renvoyant rien au programme est une fonction de type **void**.

Exemples et Exercices :

Exercice VII_1 :

```
#include <stdio.h>
#include <conio.h>
#include "c:\ bc5\courc_C\teach_c\chap7\chap7.h"   /* fichier d'en-tete */

void bonjour()       /* declaration de la fonction */
{
        printf("bonjour\n");
}

void main()             /* programme principal */
{
        bonjour();   /* appel de la fonction */
        printf("POUR CONTINUER FRAPPER UNE TOUCHE: ");
        getch();
}
```

Le fichier d'en-tête est un fichier texte contenant uniquement la ligne suivante :

```
void bonjour(void) ;
```

C'est à dire la déclaration de la fonction bonjour, sous la forme de son prototype.
Ce fichier est lu par le compilateur, qui vérifie si l'on appelle correctement la fonction.

Exercice VII_2 :

```
#include <stdio.h>
#include <conio.h>
#include "c:\bc5\courc_C\teach_c\chap7\chap7.h"   /* fichier d'en-tete */

void bonjour()       /* declaration de la fonction */
{
        printf("bonjour\n");
}
```

```
void coucou()        /* declaration de la fonction */
{
        bonjour();/* appel d'une fonction dans une fonction */
        printf("coucou\n");
}

void main()          /* programme principal */
{
        coucou();   /* appel de la fonction */
        printf("POUR CONTINUER FRAPPER UNE TOUCHE: ");
        getch();
}
```

Le fichier d'en-tête contient maintenant les lignes suivantes :

```
void bonjour(void) ;
void coucou(void) ;
```

Exercice VII_3 :

```
#include <stdio.h>
#include <conio.h>
#include "c:\bc5\cours_C\teach_c\chap7\chap7.h"   /* fichier d'en-tete */

void carre()       /* declaration de la fonction */
{
        int n,n2;           /* variables locales à carre */
        printf("ENTRER UN NOMBRE: ");
        scanf("%d",&n);
        n2 = n*n ;
        printf("VOICI SON CARRE: %d\n",n2);
}

void main()          /* programme principal */
{
        carre();   /* appel de la fonction */
        printf("POUR CONTINUER FRAPPER UNE TOUCHE: ");
        getch();
}
```

Les variables n et n2 ne sont connues que de la fonction carré.

Le fichier d'en-tête contient maintenant les lignes suivantes :

```
void bonjour(void) ;
void coucou(void) ;
void carre(void) ;
```

Les déclarations inutiles sont ignorées.

Exercice VII_4 :

```
#include <stdio.h>
#include <conio.h>
#include "c:\bc5\cours_C\teach_c\chap7\chap7.h"   /* fichier d'en-tete */

void carre()       /* declaration de la fonction */
{
        int n, n2;          /* variables locales a carre */
        printf("ENTRER UN NOMBRE: ");
```

```
        scanf("%d",&n);
        n2 = n*n;
        printf("VOICI SON CARRE: %d\n",n2);
}

void cube()     /* declaration de la fonction */
{
        int n, n3;          /* variables locales a cube */
        printf("ENTRER UN NOMBRE: ");
        scanf("%d",&n);
        n3 = n*n*n;
        printf("VOICI SON CUBE: %d\n",n3);
}

void main()         /* programme principal */
{
        char choix;                 /* variable locale a main() */
        printf("CALCUL DU CARRE TAPER 2\n");
        printf("CALCUL DU CUBE TAPER 3\n");
        printf("\nVOTRE CHOIX: ");
        scanf("%c",&choix);

        switch(choix)
        {
                case '2':carre();break;
                case '3':cube();break;
        }
        printf("\nPOUR CONTINUER FRAPPER UNE TOUCHE: ");
        getch();
}
```

Les 2 variables locales n sont indépendantes l'une de l'autre.
La variable locale choix n'est connue que de main().

Le fichier d'en-tête contient maintenant les lignes suivantes :

```
void bonjour(void) ;
void coucou(void) ;
void carre(void) ;
void cube(void) ;
```

Exercice VII_5 :

```
#include <stdio.h>
#include <conio.h>
#include "c:\bc5\courc_C\teach_c\chap7\chap7.h"   /* fichier d'en-tete */

int n;              /* variable globale, connue de tout le programme */

void carre()    /* declaration de la fonction */
{
        int n2;   /* variable locale */
        printf("ENTRER UN NOMBRE: ");
        scanf("%d",&n);
        n2 = n*n;
        printf("VOICI SON CARRE: %d\n",n2);
}

void cube()     /* declaration de la fonction */
{
        int n3;  /* variable locale */
        printf("ENTRER UN NOMBRE: ");
```

```
        scanf("%d",&n);
        n3 = n*n*n;
        printf("VOICI SON CUBE: %d\n",n3);
}

void main()         /* programme principal */
{
        char choix;                     /* variable locale a main() */
        printf("CALCUL DU CARRE TAPER 2\n");
        printf("CALCUL DU CUBE TAPER 3\n");
        printf("\nVOTRE CHOIX: ");
        scanf("%c",&choix);
        switch(choix)
        {
                case '2':carre();break;
                case '3':cube();break;
        }
        printf("\nPOUR CONTINUER FRAPPER UNE TOUCHE: ");
        getch();
}
```

La variable globale n est connue de tout le programme (fonctions et main()).
La variable locale choix n'est connue que de main().
Le fichier d'en-tête n'a pas changé.
Un programme bien construit possède peu de variables globales.

Exercice VII_6 :

Un programme contient la déclaration suivante:

```
int tab[10] = {1,2,4,8,16,32,64,128,256,512};  /* variable globale */
```

Ecrire une fonction de prototype **void affiche(void)** qui affiche les éléments du tableau, et leur adresse; la mettre en oeuvre dans main().
Actualiser le fichier d'en-tête en conséquence.

Fonctions renvoyant une valeur au programme et sans passage d'argument

Dans ce cas, la fonction, après exécution, renvoie une valeur. Le type de cette valeur est déclaré avec la fonction. La valeur retournée est spécifiée à l'aide du mot réservé ***return***.

Exemple et Exercice VII_7 :

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>
#include "c:\bc5\courc_C\teach_c\chap7\chap7.h"   /* fichier d'en-tete */

int lance_de()  /* declaration de la fonction */
{
        int test;  /* variable locale */
        test = random(6) + 1;
        return(test);
}

void main()
{
```

```
        int resultat;
        randomize();
        resultat = lance_de();
        /* resultat prend la valeur retournee par le sous-programme */
        printf("Vous avez obtenu le nombre: %d\n",resultat);
        printf("POUR SORTIR FRAPPER UNE TOUCHE ");
        getch();
}
```

Il faut ajouter au fichier d'en-tête la ligne suivante :

```
int lance_de(void) ;
```

Exercice VII 8 :

Un programme contient la déclaration suivante:

```
float liste[8] = {1.6,-6,9.67,5.90,345,-23.6,78,34.6};          /* variable globale */
```

Ecrire une fonction de prototype **float min(void)** qui renvoie le minimum de la liste.
Ecrire une fonction de prototype **float max(void)** qui renvoie le maximum de la liste.
Les mettre en oeuvre dans main().
Actualiser le fichier d'en-tête en conséquence.

Fonction avec passage d'arguments

Ce paragraphe traite directement du cas général: fonctions de type void ou non (renvoyant une valeur ou non).
Ces fonctions utilisent les valeurs de certaines variables du programme les ayant appelé: on passe ces valeurs au moyen d'arguments déclarés avec la fonction.

Exemple et Exercice VII 9 :

```
#include <stdio.h>
#include <conio.h>
#include "c:\bc5\courc_C\teach_c\chap7\chap7.h"   /* fichier d'en-tete */

int carre(int x)  /* declaration de la fonction */
{       /* x est un parametre*/
        int x2;  /* variable locale */
        x2 = x*x;
        return(x2);
}

void main()
{
        int n1, n2, res1, res2;                 /* variables locales */
        printf("ENTRER UN NOMBRE: ");
        scanf("%d",&n1);
        res1 = carre(n1);
        printf("ENTRER UN AUTRE NOMBRE: ");
        scanf("%d",&n2);
        res2 = carre(n2);
        printf("VOICI LEURS CARRES: %d   %d\n\n",res1, res2);
        printf("POUR SORTIR FRAPPER UNE TOUCHE: ");
        getch();
}
```

Dans le fichier d'en-tête, le prototype de la fonction carre devient **int carre(int) ;**

L'expression ***int carre(int)*** est appelée « prototype réduit » de la fonction.
L'expression ***int carre(int x)*** est appelée « prototype complet » de la fonction.

On peut ainsi appeler la fonction carre autant de fois que l'on veut avec des variables différentes.
x est un paramètre, ou argument: ce n'est pas une variable du programme.

S'il y a plusieurs arguments à passer, il faut respecter la syntaxe suivante:

Exemples :void fonction1(int x,int y) void fonction2(int a,float b,char c)

Exercice VII_10 :

Ecrire une fonction de prototype **int puissance(int a, int b)** qui calcule a^b, a et b sont des entiers (cette fonction n'existe pas en bibliothèque). La mettre en oeuvre dans main(). Actualiser le fichier d'en-tête en conséquence.

Exercice VII_11 :

tab1 et tab2 sont des variables locales à main :

```
int tab1[10] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19};
int tab2[10] = {-19,18,-17,16,-15,14,-13,12,-11,10,-9,8,-7,6,-5,4,-3,2,-1,0};
```

Ecrire une fonction de prototype **void affiche(int *tab)** qui permet d'afficher les 20 nombres suivant un tableau 4x5.
La mettre en oeuvre dans main() pour afficher tab1 et tab2.
Il faut ici ajouter la ligne **void affiche(int *) ;** dans le fichier d'en-tête.
On dit, dans ce cas, que l'on a passé le paramètre PAR ADRESSE.

Résumé sur variables et fonctions

On a donc vu qu'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.**

On a vu aussi qu'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 VII_12 :

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

```
#include <stdio.h>
#include <conio.h>
#include "c:\bc5\courc_C\teach_c\chap7\chap7.h"   /* fichier d'en-tete */

int n;      /* initialisee … 0 */

void calcul()
{
```

```
        static int i;           /* initialisee ... 0 */
        i++;
        printf("i=%d\n",i);
        n = n+i;
}

void main()
{
        calcul();
        printf("n= %d\n",n);
        calcul();
        printf("n= %d\n",n);
        printf("\nPOUR SORTIR FRAPPER UNE TOUCHE ");
        getch();
}
```

Le passage de paramètre entre fcts ou entre fcts et programme principal

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

Exemple: Fonction permettant d'échanger la valeur de 2 variables :

Syntaxe **qui conduit à une erreur :**

```
#include <stdio.h>
#include "c:\chap7\chap7.h"
void ech(int x,int y)
{
        int tampon;
        tampon = x;
        x = y;
        y = tampon;
}

void main()
{
        int a = 5 , b = 8;
        ech(a,b);
        printf(« a=%d\n », a) ;
        printf(« b=%d\n », b) ;
}
```

PASSAGE DES PARAMETRES PAR VALEUR

Syntaxe **correcte :**

```
#include <stdio.h>
#include "c:\chap7\chap7.h"
void ech(int *x,int *y)
{
        int tampon;
        tampon = *x;
        *x = *y;
        *y = tampon;
}

void main()
{
        int a = 5 , b = 8 ;
        ech(&a,&b);
        printf(« a=%d\n », a) ;
        printf(« b=%d\n », b) ;
}
```

PASSAGE DES PARAMETRES PAR ADRESSE

a et b sont des variables locales à main(). La fonction ech ne peut donc pas modifier leur valeur. On le fait donc en passant par l'adresse de ces variables.

Expérimenter ces deux exemples, noter les résultats obtenus.

Dans un deuxième temps, afficher dans main() les adresses de a et de b et dans ech les adresses **de** x et **de** y (programme de gauche), les adresses x et y (programme de droite).

Le problème ne se pose pas lorsque le paramètre est un pointeur ou un tableau.

Exercice VII_13 :

Saisir les 3 couleurs d'une résistance, afficher sa valeur.
Une fonction de prototype **float conversion(char *couleur)** calcule le nombre associé à chaque couleur.

Exercice VII_14 :

Calculer et afficher les racines de $ax^2+bx+c=0$.
Une fonction de prototype **void saisie(float *aa,float *bb,float *cc)** permet de saisir a,b,c.
Une fonction de prototype **void calcul(float aa,float bb,float cc)** exécute les calculs et affiche les résultats.
a, b, c sont des variables locales à main();
main() se contente d'appeler **saisie(&a,&b,&c) et calcul(a,b,c).**

Exercice VII_15 :

Ecrire une fonction de prototype **void saisie(int *tx)** qui saisie des entiers (au maximum 20), le dernier est 13. Cette fonction renvoie au programme principal **adr_debut et adr_fin,** les adresses respectives du 1er nombre et du dernier nombre saisis.
- **adr_debut** et **adr_fin** sont des variables **globales** à tout le programme.
- Le programme principal appelle **saisie(tab)** et affiche la valeur de **adr_debut** et **adr_fin** en hexadécimal; tab est une variable locale à main().

Exercice VII_16 :

Modifier la fonction de prototype **void affiche(int *tx)** de l'exercice VII_11 de sorte d'afficher les nombres en tableau 4x5 mais en s'arrêtant à **adr_fin.** Compléter le programme principal de l'exercice VII_15 par un appel à **affiche(tab).**

Corrigés des exercices

Exercice VII_6 :

```
#include <stdio.h>
#include <conio.h>
#include "c:\bc5\courc_C\teach_c\chap7\chap7.h"   /* fichier d'en-tete */

tab[10]={1,2,4,8,16,32,64,128,256,512};

void affiche()
{
        int i;
        printf("VOICI LES ELEMENTS DU TABLEAU ET LEURS ADRESSES:\n\n");
        for(i=0;i<10;i++)
                printf("ELEMENT Nø%1d: %3d    ADRESSE: %p\n",i,tab[i],tab+i);
}

void main()
{
        affiche();
        printf("\nPOUR SORTIR FRAPPER UNE TOUCHE: ");
        getch();
}
```

Exercice VII_8 :

```
#include <stdio.h>
#include <conio.h>
#include "c:\bc5\courc_C\teach_c\chap7\chap7.h"   /* fichier d'en-tete */

float liste[8] = {1.6,-6,9.67,5.90,345,-23.6,78,34.6};

float max()
{
        float maxi;
        int i;
        maxi = *liste;
        for(i=0;i<8;i++)
                if(*(liste+i)>maxi) maxi = *(liste+i);
        return(maxi);
}

float min()
{
        float mini;
        int i;
        mini = *liste;
        for(i=0;i<8;i++)
                if(*(liste+i)<mini) mini = *(liste+i);
        return(mini);
}

void main()
{
        float rmin, rmax;
        rmax = max();
        rmin = min();
        printf("LE MAXIMUM VAUT: %f\n",rmax);
        printf("LE MINIMUM VAUT: %f\n",rmin);
        printf("\nPOUR CONTINUER FRAPPER UNE TOUCHE");
        getch();
}
```

Exercice VII_10 :

```
#include <stdio.h>
#include <conio.h>
#include "c:\bc5\courc_C\teach_c\chap7\chap7.h"   /* fichier d'en-tete */

int puissance(int x,int y)
{
        int i,p=1;
        for(i=1;i<=y;i++) p=x*p;
        return(p);
}

void main()
{
        int a,b,res;
        printf("\nENTRER A: ");scanf("%d",&a);
        printf("\nENTRER B: ");scanf("%d",&b);
        res = puissance(a,b);
        printf("\A PUISS B = %d\n",res);
        printf("\n POUR CONTINUER FRAPPER UNE TOUCHE ");
        getch();
}
```

Exercice VII_11 :

```
 #include <stdio.h>
#include <conio.h>
#include "c:\bc5\courc_C\teach_c\chap7\chap7.h"   /* fichier d'en-tete */

void affiche(int *tab)
{
        int i;
        for(i=0;i<20;i++)
                if((i+1)%5==0) printf("%3d\n",tab[i]);
                else printf("%3d ",tab[i]);
        printf("\n\n");
}

void main()
{
        int tab1[20]={0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19};
        int tab2[20]={-19,18,-17,16,-15,14,-13,12,-11,10,-9,8,-7,6,-5,4,-3,2,-1,0};
        affiche(tab1);
        affiche(tab2);
        printf("\nPOUR CONTINUER FRAPPER UNE TOUCHE ");
        getch();
}
```

Exercice VII_13 :

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <alloc.h>
#include <conio.h>
#include "c:\bc5\courc_C\teach_c\chap7\chap7.h"   /* fichier d'en-tete */

float conversion(char *couleur)
{
        float x=10;
        couleur = strupr(couleur);  /* convertit en majuscules */
        /* stricmp permet d'eviter l'utilisation de strupr */
        if (strcmp("NOIR",couleur)==0) x=0;
        if (strcmp("MARRON",couleur)==0) x=1;
        if (strcmp("ROUGE",couleur)==0) x=2;
        if (strcmp("ORANGE",couleur)==0) x=3;
        if (strcmp("JAUNE",couleur)==0) x=4;
        if (strcmp("VERT",couleur)==0) x=5;
        if (strcmp("BLEU",couleur)==0) x=6;
        return(x); /* x permet d'ajouter ,ventuellement un contr“le d'erreur */
}

void main()
{
        float r,c1,c2,c3;
        char *coul1,*coul2,*coul3;
        coul1 = (char*)malloc(8);
        coul2 = (char*)malloc(8);
        coul3 = (char*)malloc(8);
        printf("\nENTRER LES 3 COULEURS DE LA RESISTANCE:\n");
        printf("COULEUR1: ");gets(coul1);
        printf("COULEUR2: ");gets(coul2);
        printf("COULEUR3: ");gets(coul3);
        c1=conversion(coul1);
        c2=conversion(coul2);
        c3=conversion(coul3);
```

```
		r = (c1*10 + c2)*pow(10,c3);
		if(r<1000) printf("\nVALEUR DE R: %4.0f KOHM\n",r);
		if((r>=1000)&&(r<999999))
		{
			r=r/1000;
			printf("\nVALEUR DE R: %7.0f KOHM\n",r);
		}
		if(r>=999999)
		{
			r=r/1e6;
			printf("\nVALEUR DE R: %4.0f MOHM\n",r);
		}
		free(coul1);
		free(coul2);
		free(coul3);
		printf("\nPOUR SORTIR FRAPPER UNE TOUCHE ");
		getch();
}
```

Exercice VII_14 :

```
#include <stdio.h>
#include <math.h>
#include <conio.h>
#include "c:\bc5\courc_C\teach_c\chap7\chap7.h"   /* fichier d'en-tete */

void saisie(float *aa,float *bb,float *cc)
{
		printf("\nENTRER A: ");scanf("%f",aa);
		printf("\nENTRER B: ");scanf("%f",bb);
		printf("\nENTRER C: ");scanf("%f",cc);
		printf("\n");
}

void calcul(float aa,float bb,float cc)
{
		float delta,x1,x2;
		printf("\nA= %f B= %f C=%f\n",aa,bb,cc);
		delta = bb*bb-4*cc*aa;
		printf("\nDELTA = %f\n",delta);
		if (delta<0) printf("\nPAS DE SOLUTION");
		if (delta == 0)
		{
			x1=-(bb/aa/2);
			printf("\UNE SOLUTION: X= %f\n",x1);
		}
		if (delta > 0)
		{
			x1=(-bb+sqrt(delta))/2/aa;
			x2=(-bb-sqrt(delta))/2/aa;
			printf("\DEUX SOLUTIONS: X1 = %f  X2 = %f\n",x1,x2);
		}
}

void main()
{
		float a,b,c;
		saisie(&a,&b,&c);
		calcul(a,b,c);
		printf("\n\nPOUR SORTIR FRAPPER UNE TOUCHE ");
		getch();
}
```

Exercice VII_15 :

```
#include <stdio.h>
#include <conio.h>
#include "c:\bc5\courc_C\teach_c\chap7\chap7.h"   /* fichier d'en-tete */

int *adr_deb,*adr_fin;

void saisie(int *tx)
{
        int i=0;
        printf("SAISIE DES NOMBRES SEPARES PAR RETURN (dernier =13)\n");
        do
        {
                printf("NOMBRE: ");
                scanf("%d",tx+i);
                i++;
        }
        while(*(tx-1+i)!=13);
        adr_fin= (int*)(tx+i-1);
        adr_deb=(int*)tx;
}

void main()
{
        int tab[20];
        saisie(tab);
        printf("\nADR_DEB = %p  ADR_FIN = %p\n",adr_deb,adr_fin);
        printf("\nPOUR SORTIR FRAPPER UNE TOUCHE ");
        getch();
}
```

Exercice VII_16 :

```
#include <stdio.h>
#include <conio.h>
#include "c:\bc5\courc_C\teach_c\chap7\chap7.h"   /* fichier d'en-tete */

int *adr_deb,*adr_fin;

void saisie(int *tx)
{
        int i=0;
        printf("SAISIE DES NOMBRES SEPARES PAR RETURN (dernier =13)\n");
        do
        {
                printf("NOMBRE: ");
                scanf("%d",tx+i);
                i++;
        }
        while(*(tx-1+i)!=13);
        adr_fin= (int*)(tx+i-1);
        adr_deb= (int*)tx;
}

void affiche(int *tx)
{
        int i;
        printf("\n");
        for(i=0;(tx+i)!=(adr_fin+1);i++)
        if((i+1)%5==0) printf("%5d\n",tx[i]);
        else printf("%5d  ",tx[i]);
}
```

```
void main()
{
        int tab[20];
        saisie(tab);
        affiche(tab);
        printf("\n\nADR_DEB = %p  ADR_FIN = %p\n",adr_deb,adr_fin);
        printf("\nPOUR SORTIR FRAPPER UNE TOUCHE ");
        getch();
}
```

LES TYPES DE VARIABLES COMPLEXES

Les déclarations de types synonymes : typedef

On a vu les types de variables utilisés par le langage C: char, int, float, pointeur; le chapitre 9 traitera des fichiers (type FILE).

Le programmeur a la possibilité de créer ses propres types: **Il suffit de les déclarer en début de programme** (après les déclarations des bibliothèques et les « define ») avec la syntaxe suivante:

Exemples :

```
typedef int entier;             /* on définit un nouveau type "entier" synonyme de "int" */

typedef int vecteur[3];         /* on définit un nouveau type "vecteur" synonyme */
                                /* de "tableau de 3 entiers"    */

typedef float *fpointeur;       /* on définit un nouveau type "fpointeur" synonyme */
                                /* de "pointeur sur un réel */
```

Utilisation : La portée de la déclaration de type dépend de l'endroit où elle est déclarée: dans main(), le type n'est connu que de main(); en début de programme, le type est reconnu dans tout le programme.

```
#include <stdio.h>

typedef  int entier;
typedef float point[2];

void main()
{
        entier n = 6;
        point xy;
        xy[0] = 8.6;
        xy[1] = -9.45;

        etc ...

}
```

Exercice VIII_1 : Afficher la taille mémoire d'un point à l'aide de l'opérateur sizeof.

Exercice VIII_2 : Définir le type

```
typedef char ligne[80];
```

Déclarer dans main() un pointeur de ligne; lui attribuer de la place en mémoire (pour 5 lignes). Ecrire une fonction qui effectue la saisie de 5 lignes puis une autre qui les affiche. Les mettre en oeuvre dans main().
Attention, le fichier d'en-tête doit contenir la déclaration du nouveau type.

Les structures

Le langage C autorise la déclaration de types particuliers: les structures . Une structure est constituée de plusieurs éléments de même type ou non.

Exemple de déclaration :

```
typedef struct                  /* On définit un type struct */
{
        char nom[10];
        char prenom[10];
        int age;
        float note;
}
fiche;
```

Utilisation :

On déclare des variables par exemple :

```
fiche f1,f2;
```

puis, par exemple :

```
strcpy(f1.nom,"DUPONT");
strcpy(f1.prenom,"JEAN");
f1.age = 20;
f1.note = 11.5;
```

L'affectation globale est possible avec les structures: on peut écrire: **f2 = f1;**

Exercice VIII_3 : Créer la structure ci-dessus, saisir une fiche, l'afficher.

Structures et tableaux

On peut définir un tableau de structures (mais ceci est assez peu utilisé) :

Exemple de déclaration : (à partir de la structure définie précédemment)

```
fiche f[10];          /* on déclare un tableau de 10 fiches */
```

Utilisation :

```
strcpy(f[i].nom,"DUPONT")   /* pour un indice i quelconque */
strcpy(f[i].prenom,"JEAN");
f[i].age = 20;
f[i].note = 11.5;
```

Exercice VIII_4 :

Créer une structure **point{int num;float x;float y;}**
Saisir 4 points, les ranger dans un tableau puis les afficher.

Structures et pointeurs

On peut déclarer des pointeurs sur des structures. Cette syntaxe est très utilisée en langage C, **elle est notamment nécessaire lorsque la structure est un paramètres modifiable dans la fonction.**

Un symbole spécial a été créé pour les pointeurs de structures, il s'agit du symbole **->**

Exemple de déclaration : (à partir de la structure définie précédemment)

```
fiche *f;   /* on déclare un pointeur de fiche */
```

Utilisation :

```
f = (fiche*)malloc(sizeof(fiche)); /* réserve de la place */
strcpy(f->nom,"DUPONT");
strcpy(f->prenom,"JEAN");
f->age = 20;
f->note = 11.5;
```

Exercice VIII_5 : Reprendre l'exercice VIII_2: Une fonction saisie permet de saisir un point, une fonction affiche permet d'afficher un point.

Corrigés des exercices

Exercice VIII_1 :

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

typedef float point[2];

void main()
{
        printf("TAILLE D'UN POINT: %2d OCTETS\n",sizeof(point));
        printf("\nPOUR SORTIR FRAPPER UNE TOUCHE ");
        getch();
}
```

Exercice VIII_2 :

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#include "c:\bc5\courc_C\teach_c\chap8\chap8.h"   /* fichier d'en-tete */

/* cette declaration est normalement dans le fichier µ/
/*d'en-tete typedef char ligne[80]; */

void saisie (ligne *tx)
{
        int i;
        printf("\n             SAISIE DU TEXTE\n\n");
        for (i=0;i<5;i++)
        {
                printf("LIGNE Num %d ",i);
```

```
                scanf("%s",tx+i); /* saisie de la ligne nøi */
        }
}

void affiche(ligne *tx)
{
        int i;
        printf("\n\n\n          AFFICHAGE DU TEXTE\n\n");
        for(i=0;i<5;i++)
                printf("%s\n",tx+i);
}

void main()
{
        ligne *texte;
        texte = malloc(sizeof(ligne)*5);/* reserve de la place pour 5 lignes */
        saisie(texte);
        affiche(texte);
        free(texte);
        printf("\nPOUR SORTIR FRAPPER UNE TOUCHE ");
        getch();
}
```

Exercice VIII_3 :

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

typedef struct
{
        char nom[10];
        char prenom[10];
        int age;
        float note;
}
fiche;

void main()
{
        fiche f;
        printf("SAISIE D'UNE FICHE \n");
        printf("NOM: ");gets(f.nom);
        printf("PRENOM: ");gets(f.prenom);
        printf("AGE: ");scanf("%d",&f.age);
        printf("NOTE: ");scanf("%f",&f.note);
        printf("\n\nLECTURE DE LA FICHE:\n");
        printf("NOM: %s PRENOM: %s AGE: %2d NOTE: %4.1f",f.nom,f.prenom,f.age,f.note);
        printf("\n\nPOUR SORTIR FRAPPER UNE TOUCHE ");
        getch();
}
```

Exercice VIII_4 :

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

typedef struct {int num;float x;float y;} point;

void main()
{
        point p[4];             /* tableau de points */
        int i;
        float xx,yy;
```

```
	/* saisie */
	printf("SAISIE DES POINTS\n\n");
	for(i=0;i<4;i++)
	{
		printf("\nRELEVE N¯%1d\n",i);
		p[i].num = i;
		printf("X= ");scanf("%f",&xx);
		printf("Y= ");scanf("%f",&yy);
		p[i].x = xx;p[i].y = yy;
	}

	/* relecture */
	printf("\n\nRELECTURE\n\n");
	for(i=0;i<4;i++)
	{
		printf("\nRELEVE N¯%1d",p[i].num);
		printf("\nX= %f",p[i].x);
		printf("\nY= %f\n",p[i].y);
	}
	printf("\n\nPOUR SORTIR FRAPPER UNE TOUCHE ");
	getch();
}
```

Exercice VIII_5 :

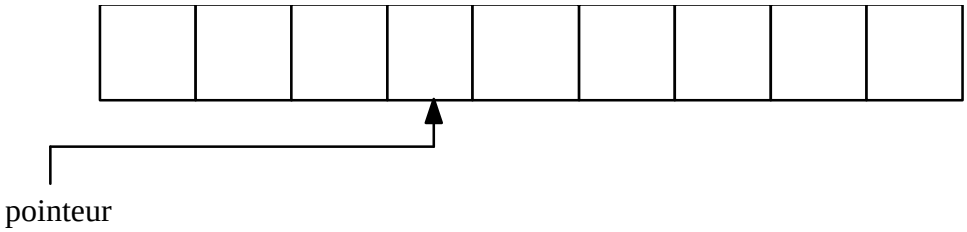
```
 #include <stdio.h>
#include <conio.h>
#include "c:\bc5\courc_C\teach_c\chap8\chap8.h"   /* fichier d'en-tete */

void saisie(point *pp,int i)
{
	float xx,yy;
	printf("\nRELEVE Nø%1d\n",i);
	printf("X= ");scanf("%f",&xx);
	printf("Y= ");scanf("%f",&yy);
	pp->num = i;pp->x = xx;pp->y = yy;
}

void affiche(point *pp)
{
	printf("\nRELEVE Nø%1d",pp->num);
	printf("\nX= %f",pp->x);
	printf("\nY= %f\n",pp->y);
}

void main()
{
	point p[4];		/* tableau de points */
	int i;
	printf("SAISIE:\n\n");
	for(i=0;i<4;i++)saisie(&p[i],i);
	printf("\n\nRELECTURE:\n\n");
	for(i=0;i<4;i++)affiche(&p[i]);
	printf("\n\nPOUR SORTIR FRAPPER UNE TOUCHE ");
	getch();
}
```

LES FICHIERS

Généralités

Un fichier est un ensemble d'informations stockées sur une mémoire de masse (disque dur, disquette, bande magnétique, CD-ROM).

En 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 :

1- Accès séquentiel (possible sur tout support, mais seul possible sur bande magnétique) :

- Pas de cellule vide.
- On accède à une cellule quelconque en se déplaçant (via un pointeur), depuis la cellule de départ.
- On ne peut pas détruire une cellule.
- On peut par contre tronquer la fin du fichier.
- On peut ajouter une cellule à la fin.

2- Accès direct (RANDOM I/O) (Utilisé sur disques, disquettes, CD-ROM où l'accès séquentiel est possible aussi).

- Cellule vide possible.
- On peut directement accéder à une cellule.
- On peut modifier (voir détruire) n'importe quelle cellule.

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

1- En binaire :

Fichier dit « binaire », les informations sont codées telles que. Ce sont en général des fichiers de nombres. Ils ne sont pas listables.

2- en ASCII :

Fichier dit « texte », les informations sont codées en ASCII. Ces fichiers sont listables. Le dernier octet de ces fichiers est EOF (caractère ASCII spécifique).

Manipulation des fichiers

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++.

Le langage C ne distingue pas les fichiers à accès séquentiel des fichiers à accès direct, certaines fonctions de la bibliothèque livrée avec le compilateur permettent l'accès direct. Les fonctions standards sont des fonctions d'accès séquentiel.

Déclaration : *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.

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;

Exercices

Exercice IX_1 : Copier un fichier texte dans un autre: créer un fichier "essai.dat" sous éditeur. Tester le programme en vérifiant après exécution la présence du fichier copié dans le directory.

Exercice IX_2 : Calculer et afficher le nombres de caractères d'un fichier ASCII (Utiliser n'importe quel fichier du répertoire).

Exercice IX_3 : Créer et relire un fichier binaire de 10 entiers.

Exercice IX_4 : Lire un fichier texte, avec un contrôle d'erreur: L'utilisateur saisit le nom du fichier, la machine retourne le listing du fichier s'il existe et un message d'erreur s'il n'existe pas.

Exercice IX_5 : Créer et relire un fichier texte de 5 chaînes de 3 caractères.

Exercice IX_6 : Ajouter une fiche (c'est à dire une chaîne de 3 caractères) au fichier précédent et relire le fichier.

Exercice IX_7 : Rechercher une fiche dans le fichier précédent.

Exercice IX_8 : Exercice récapitulatif: Créer une structure nom, prénom, âge. Ecrire un programme de gestion de fichier (texte) avec menu d'accueil: possibilité de créer le fichier, de le lire, d'y ajouter une fiche, d'en rechercher une.

Corrigé des exercices

Exercice IX_1 :

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

void main()
{
        FILE *fichier1,*fichier2;
        char c;

        printf("COPIE EN COURS ...\n");
        fichier1 = fopen("c:\\bc5\\Courc_C\\Teach_C\\CHAP9\\essai.dat","r");
        fichier2 = fopen("copie.dat","w");
        while((c=(char)getc(fichier1))!=EOF)putc(c,fichier2);
        fclose(fichier1);
        fclose(fichier2);
        printf("C'EST FINI !\n");
        printf("\nPOUR SORTIR FRAPPER UNE TOUCHE ");
        getch();
}
```

Exercice IX_2 :

```
 #include <stdio.h>
#include <conio.h>
void main()
{
        FILE *fichier;
        int compteur=0;
        fichier = fopen("c:\\bc5\\Courc_C\\teach_C\\chap9\\copie.dat","r");
        while(getc(fichier)!=EOF)compteur++;
        fclose(fichier);
        printf("TAILLE DU FICHIER: %d OCTETS\n",compteur);
        printf("\nPOUR SORTIR FRAPPER UNE TOUCHE ");
        getch();
}
```

Exercice IX_3 :

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

void main()
{
        FILE *fichier;
        int i,n;
        fichier = fopen("nombre.dat","wb+");
        for(i=0;i<10;i++)
        {
                printf("N = ");
                scanf("%d",&n);
                putw(n,fichier);
        }
        rewind(fichier);
```

```
        while(!feof(fichier))   /* essayer avec une boucle for */
        {
                n=getw(fichier);
                printf("%d ",n);
        }

        fclose(fichier);
        printf("\nPOUR SORTIR FRAPPER UNE TOUCHE ");
        getch();
}
```

Exercice IX_4 :

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

void main()
{
        FILE *fichier;
        char c,nom[10];

        printf("NOM DU FICHIER A LISTER: ");
        gets(nom);
        if((fichier = fopen(nom,"r"))==NULL)
                printf("\nERREUR A L'OUVERTURE, CE FICHIER N'EXISTE PAS\n");
        else
        {
                printf("\n\t\t\tLISTING DU FICHIER\n");
                printf("\t\t\t----------\n\n");
                while((c=getc(fichier))!=EOF)printf("%c",c);
        }
        fclose(fichier);
        printf("\n\nPOUR SORTIR FRAPPER UNE TOUCHE ");
        getch();
}
```

Exercice IX_5 :

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

void main()
{
        FILE *fichier;
        int i;
        char p[4];

        /* saisie du fichier */
        fichier = fopen("chaine.dat","w+");
        printf("\nENTRER 5 CHAINES DE 3 CARACTERES\n");
        for(i=0;i<5;i++)
        {
                gets(p);
                fputs(p,fichier);
        }
        rewind(fichier);   /* pointeur au debut */

        /* relecture du fichier */
        printf("\n\nLECTURE DU FICHIER\n\n");
```

```
        while((fgets(p,4,fichier))!=NULL)printf("%s ",p);
        fclose(fichier);

        printf("\n\nPOUR SORTIR FRAPPER UNE TOUCHE ");
        getch();
}
```

Exercice IX_6 :

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

void main()
{
        FILE *fichier;
        char p[4];

        /* ajout d'une chaine */
        fichier = fopen("chaine.dat","a+"); /* pointeur a la fin */
        printf("\nENTRER LA CHAINE DE 3 CARACTERES A AJOUTER\n");
        gets(p);
        fputs(p,fichier);

        /*lecture du fichier pour verification */
        rewind(fichier);
        printf("\n\nLECTURE DU FICHIER\n");
        while((fgets(p,4,fichier))!=NULL)printf("%s ",p);
        fclose(fichier);

        printf("\n\nPOUR SORTIR FRAPPER UNE TOUCHE ");
        getch();
}
```

Exercice IX_7 :

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void main()
{
        FILE *fichier;
        char p[4],q[4],trouve=0;

        /* recherche d'une chaine */
        fichier = fopen("chaine.dat","r"); /* lecture seule */
        printf("\nENTRER LA CHAINE DE 3 CARACTERES RECHERCHEE\n");
        gets(p);
        printf("\n\nRECHERCHE ...\n\n");
        while(((fgets(q,4,fichier))!=NULL)&&(trouve==0))
                if(strcmp(p,q)==0)trouve = 1;   /* compare les chaines */
        if (trouve ==0) printf("CETTE CHAINE N'EXISTE PAS DANS LE FICHIER\n");
        else printf("CHAINE TROUVEE DANS LE FICHIER\n");

        /*lecture du fichier pour verification */
        rewind(fichier);
        printf("\n\nLECTURE DU FICHIER\n");
        while((fgets(q,4,fichier))!=NULL)printf("%s ",q);
        fclose(fichier);
```

```
        printf("\n\nPOUR SORTIR FRAPPER UNE TOUCHE ");
        getch();
}
```

Exercice IX_8 :

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

typedef struct
{
        char nom[10];
        char prenom[10];
        int age;
}
carte;          /* creation d'un type carte */

void creer_fichier(FILE *f,char *n)
{
        char choix;
        carte fiche;
        clrscr();
        printf("CREATION DU FICHIER \n\n");
        printf("NOM DU FICHIER A CREER: ");
        gets(n);
        flushall();
        f = fopen(n,"w");
        do
        {
                printf("\nSAISIE D'UNE FICHE ?(o/n) ");
                choix = (char)getchar();
                flushall();
                if ((choix=='o')||(choix=='O'))
                {
                        printf("\nNOM: ");gets(fiche.nom);
                        printf("PRENOM: ");gets(fiche.prenom);
                        printf("AGE: ");scanf("%d",&fiche.age);
                        flushall();
                        fwrite(&fiche,sizeof(carte),1,f);
                }
        }
        while((choix=='o')||(choix=='O'));
        fclose(f);
}

void lire_fichier(FILE *f,char *n)
{
        carte fiche;
        int compteur=0;
        clrscr();
        printf("LECTURE DU FICHIER\n\n");
        printf("NOM DU FICHIER A LIRE: ");gets(n);
        flushall();
        f = fopen(n,"r");
        if (f == NULL) printf("\nERREUR, CE FICHIER N'EXISTE PAS\n\n");
        else
        {
                printf("\nLISTING DU FICHIER\n\n");
                while(fread(&fiche,sizeof(carte),1,f)!=0)
```

```
                {
                        printf("fiche nø%d: \n",compteur);
                        compteur++;
                        printf("%s %s %d an(s)\n\n",fiche.nom,fiche.prenom,fiche.age);
                }
                fclose(f);
        }
        printf("POUR CONTINUER FRAPPER UNE TOUCHE ");
        getch();
}

void ajout(FILE *f,char *n)
{
        carte fiche;
        char choix;
        clrscr();
        printf("AJOUT D'UNE FICHE \n\n");
        printf("NOM DU FICHIER A MODIFIER: ");
        gets(n);
        flushall();
        f = fopen(n,"a");
        do
        {
                printf("\nSAISIE D'UNE FICHE ?(o/n) ");
                choix = (char)getchar();
                flushall();
                if ((choix=='o')||(choix=='O'))
                {
                        printf("\nNOM: ");gets(fiche.nom);
                        printf("PRENOM: ");gets(fiche.prenom);
                        printf("AGE: ");scanf("%d",&fiche.age);
                        flushall();
                        fwrite(&fiche,sizeof(carte),1,f);
                }
        }
        while((choix=='o')||(choix=='O'));
        fclose(f);
}

void recherche(FILE *f,char *n)
{
        carte fiche;
        int compteur=0;
        char trouve = 0,nn[10],pp[10];
        clrscr();
        printf("RECHERCHE DE FICHE\n\n");
        printf("NOM DU FICHIER: ");
        gets(n);
        flushall();
        f = fopen(n,"r");
        printf("\nFICHE A RETROUVER:\n");
        printf("NOM: ");gets(nn);
        printf("PRENOM: ");gets(pp);
        flushall();
        while((fread(&fiche,sizeof(carte),1,f)!=0)&&(trouve==0))
        {
                if((strcmp(fiche.nom,nn)==0)&&(strcmp(fiche.prenom,pp)==0))
                {
                        trouve=1;
                        printf("FICHE RETROUVEE: FICHE nø%2d\n",compteur);
                }
                compteur++;
```

```
        }
        if (trouve==0)printf("CETTE FICHE N'EXISTE PAS\n");
        fclose(f);
        printf("POUR CONTINUER FRAPPER UNE TOUCHE ");
        getch();
}

void main()
{
        FILE *fichier;
        char nom[10];       /* nom du fichier */
        char choix;
        do
        {
                clrscr();
                printf("\t\t\tGESTION DE FICHIER\n");
                printf("\t\t\t----------------\n\n\n");
                printf("CREATION DU FICHIER  ---> 1\n");
                printf("LECTURE DU FICHIER   ---> 2\n");
                printf("AJOUTER UNE FICHE    ---> 3\n");
                printf("RECHERCHER UNE FICHE ---> 4\n");
                printf("SORTIE               ---> S\n\n");
                printf("VOTRE CHOIX: ");
                choix = (char)getchar();
                flushall();

                switch(choix)
                {
                        case '1':creer_fichier(fichier,nom);break;
                        case '2':lire_fichier(fichier,nom);break;
                        case '3':ajout(fichier,nom);break;
                        case '4':recherche(fichier,nom);break;
                }
        }
        while ((choix!='S') && (choix!='s'));
}
```