I Introduction.

Le langage C fait parti des langages structurés. Il fût créé en 1970 par Denis Ritchie pour créer le système d'exploitation UNIX (Multipostes et Multitâche).

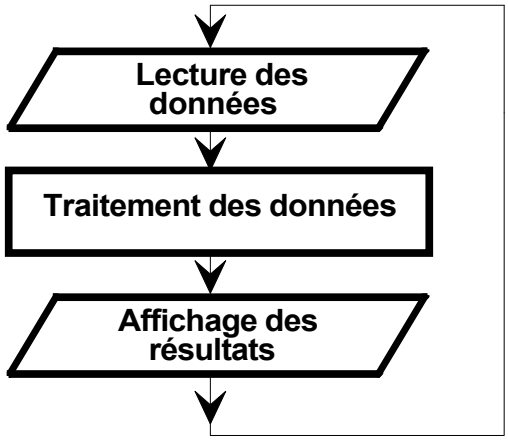
Les avantages de C sont nombreux:

- **La portabilité: Un programme développé en C sur une machine donnée peut être porté sur d'autres machines sans le modifier.**
- **Une grande bibliothèque de fonctions: Le C, suivant les machines utilisées dispose d'un grand nombres de fonctions, que ce soit des fonctions mathématiques, de gestion de fichiers ou d'entrées / sorties.**
- **Proche de la machine: Le C est très très proche de la machine, en effet il permet accéder aux adresses des variables.**
- **Très rapide: Aucun contrôle de débordement n'est effectué, ce qui apporte une grande vitesse d'exécution.**

Attention le C n'est pas un langage pour débutant, il faut apporter beaucoup de rigueur au développement d'un programme.

II Un Programme en C.

Tout programme est toujours constitué de trois phases, à savoir:
- Lecture des données.
- Traitement des données. (suite d'actions élémentaires).
- Affichage des résultats.



Remarque: On parle parfois pour un programme donné d'application ou de logiciel.

Structure d'un programme:

```
#include <STDIO.H>                          ←Inclusion des fichiers de bibliothèque.

#define PI 3.14                             ←Déclaration des constantes.

float  r,p;                                 ←Déclaration des variables globales

void perimetre(float rayon,float *peri);    ←Déclaration des Prototypes.(Une
                                            déclaration de prototypes se termine tou-
                                            jours par un point virgule ;.
void perimetre(float rayon,float *peri)     ←Déclaration des procédures, fonctions ou
                                            de Sous Programmes.
{                                           ←Début de la procédure
       *peri=2*PI*rayon;
}                                           ←Fin de la Procédure

main()                                      ←Programme Principal ou ordonnancement.
{                                           ←Début du programme Principal
    r=3;
    perimetre(r,&p);                        ←Instructions
    printf("Le perimetre du cercle
    de rayon %f est égal à %f",r,p);
}                                           ←Fin du Programme Principal
```

**** Règles de bases:***

- En Langage **C**, il est souvent nécessaire d'inclure des fichiers dit d'en-tête (***HEADER:*.H***) contenant la déclaration de variables, constantes ou de procédures élémentaires. Le fichier à inclure se trouve en général dans le répertoire des fichiers d'en-têtes (***Include***), dans ce cas là on écrira:
  #Include <*Nom_du_fichier.H*>.
  Si le fichier se trouve dans le répertoire courant, on écrira:
  #Include "*Nom_du_fichier.H*"

- **Toutes instructions ou actions se terminent par un point virgule ;**

- **Une ligne de commentaires doit commencer par /* et se terminer par */.**

- **Un bloc d'instructions commence par { et se termine par }.**

III Les Variables et les Constantes.

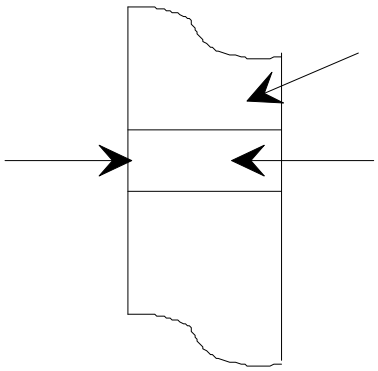
Définition d'une constante: **Elles ne changent jamais de valeurs pendant l'exécution d'un programme. Elles sont généralement stockées dans la mémoire morte d'une machine.**

Définition d'une variable: **Elles peuvent changer de valeur pendant l'exécution d'un programme. Elles sont généralement stockées en mémoire vive d'une machine.**

Une variable ou une constante est souvent définie par *cinq éléments*

- **L'identificateur: C'est le nom que l'on donne à la variable ou à la constante.**
- **Le type: Si la variable est un *entier* ou un *caractère* ou une *chaîne de caractère* ou un *réel* ou un *booléen*.**
- **La taille: C'est *le nombre d'octets occupés* en mémoire, elle est fonction du type.**
- **La valeur: C'est la *valeur* que l'on attribut à la variable ou à la constante.**
- **L'adresse: C'est là où sont stocké la valeur de la variable ou de la constante. L'adrese peut encore être appelée *pointeur*.**

Représentation mémoire d'une constante ou d'une variable:



Définition des types de variables ou de constantes.

- **Un entier: C'est un nombre positif ou négatif.**
  ***Exemples*: +1234, -56.**

- **Un caractère: C'est un nombre entier positif compris entre 0 et 255 et c'est le code ASCII d'un caractère.**
  ***Exemple*: 65 est le code ASCII de 'A'.**

- **Une chaîne de caractères: C'est un ensemble de caractères mis les uns à la suite des autres pour former un mot.**
  ***Exemple:* "Police" c'est la suite des caractères 'P', 'o', 'l', 'i', 'c' et 'e' ou encore des codes ASCII 80, 111, 108, 105, 99, 101.**

- **Un booléen: Il ne peut prendre que deux états, VRAI ou FAUX.**

- **Un réel: C'est un nombre à virgule positif ou négatif avec un exposant.**
  ***Exemple* $12{,}344.10^{e-5}$.**

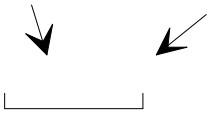
III.1 Les Constantes.

Les constantes n'existent pas, c'est à dire qu'il n'y a pas d'allocation mémoire, mais on peut affecter à un identificateur (Nom) une valeur constante par l'instruction #`define`.

```
Syntaxe:  <#define> <identificateur> <valeur>;
```

Exemple:

```
#define PI 3.14
```



III.2 Les variables.

Les variables sont défines par *la classe, le type et l'identificateur*.

```
Syntaxe:
<classe> <type> <identificateur1>,..,<identificateurn>
```

Exemple:

- **La classe: Elle détermine le comportement des variables pendant l'exécution d'un programme, très peu utilisée. Elle est très souvent supprimée dans la déclaration de variable.**

Classe	*Signification*
`Auto`	**Classe par défaut.**
`Register`	**Force le compilateur à utiliser un registre du microprocesseur, dans la mesure du possible.**
`Static`	**La valeur est conservée dans tout le programme**
`Extern`	**La variable est accesible par d'autres programmes.**

- **L'identificateur: C'est le *nom* affecté à la variable. Le nombre de caractères peut être limité, cela dépend du compilateur utilisé.**

- **Le type: Il détermine la taille de la variable et les opérations pouvant être effectuées. Le type est fonction de la machine ou du compilateur utilisé. On peut rajouter le mot `unsigned` devant le type du variable, alors la variable devient non signée et cela permet d'étendre la plage de valeurs.**

En TURBO C.

Type	*Taille (En bits)*	*Signé*	*Non Signé (Unsigned)*
Caractère → `Char`	**8**	**-128 à +127**	**0 à +255**
Entier → `Int`	**16**	**-32768 à +32768**	**0 à +65535**
Entier Court → `Short`	**16**	**-32768 à +32768**	**0 à +65535**
Entier long → `Long`	**32**	**-2147483647 à +2147483647**	**0 à +4294967295**
Réel → `Float`	**32**	**+/- 3,4*10^{-38} à +/- 3,4*10^{+38}**	**Aucune Signification**
Réel double précision → `Double`	**64**	**+/- 1,7*10^{-308} à +/-1,7*10^{+308}**	**Aucune Signification**

Exemple:

```
/* Déclaration de réel */
float       rayon;
/* Déclaration d'entier */
int i,j;
/* Déclaration de caractère */
char t;
/* Déclaration de réel double */
double pi;
/* Déclaration d'un octet */
unsigned char octet;
/* Déclaration d'un octet avec la classe registre */
register unsigned char port;

main()
{
     rayon=10.14;
     i=2;
     j=3;
     t='A'; /* t=65 Code Ascii de A */
     pi=3.14159;
     octet=129; /* On peut aller au dessus de +127 */
     port=34;
}
```

Remarque: Lors de l'affectation des variables si on met `0` avant la valeur, elle sera en Octal et si on met `0x` devant une valeur elle sera hexadécimale.

***Exemple*:**

```
/* Déclaration d'entier */
int i;

main()
{
     /* Décimal */
     i=21;
     /* Octal */
     i=025;
     /* Hexadécimale */
     i=0x15;
}
```

III 2.1 Les initialisations de variables.

Deux solutions sont possibles:

L'initialisation après déclaration.	*L'initialisation lors de la déclaration.*

`/* Déclaration */`
`int i;`
`main()`
`{`
`  /* Initialisation */`
`  i=15;`
`}`
	`/* Déclaration et Initilisation */`
	`int i=15;`
	
	`main()`
	`{`
	`  .`
	`  .`
	`}`

Remarque: La méthode d'initialisation après déclaration est conseillée car elle permet de bien séparée la déclaration de l'initialisation.

III 2.2 Les tableaux.

Ils permettent de stocker des variables de même type de façon contiguë. Ils sont caractérisés par:

- **Le nombre d'éléments.**
- **Le nombre de dimensions.**

Syntaxe:
<classe> <type> <identificateur>[nb.éléments de la première dimension].....[nb.éléments de nième dimension];

- **Tableau à une dimension.**

Exemple:

```
/* Déclaration d'un tableau de 10 éléments à une dimension */
int i[10];
/* Le premier élément est i[0] */
/* Le dernier élément est i[10-1] -> i[9] */

main()
{
          .
          .
          .
}
```

Remarque: **Dans un tableau le premier élément est l'indice 0 et le dernier élément est l'indice nombre éléments-1.**

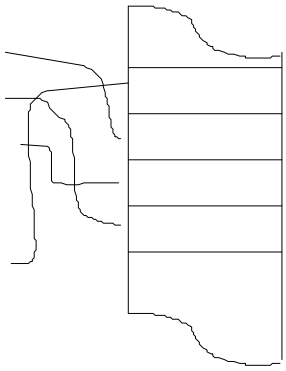
Initialisation d'un tableau:

```
/* Déclaration d'un tableau de 10 éléments à une dimension */
int i[10];
/* Le premier élément est i[0] */
/* Le dernier élément est i[10-1] -> i[9] */

main()
{
     i[1]=3;
     i[3]=7;
     i[2]=23;
     i[0]=10;
}
```

ou encore

```
/* Déclaration et initialisation
   d'un tableau de 10 éléments à une dimension */
int i[10]={10,3,23,7};
/* Le premier élément est i[0] */
/* Le dernier élément est i[10-1] -> i[9] */

main()
{
            .
            .
            .
}
```

Représentation mémoire:



- **Tableau à plusieurs dimensions.**

Exemple:

```
/* Déclaration d'un tableau de 9 éléments à deux dimensions */
int k[3][3];
/* Le premier élément est k[0][0] */
/* Le dernier élément est k[2][2] */

main()
{
            .
            .
            .
}
```

Initialisation:

```
/* Déclaration d'un tableau de 9 éléments à deux dimensions */
int k[3][3];
/* Le premier élément est k[0][0] */
/* Le dernier élément est k[2][2] */

main()
{
     /* Initialisation du tableau k */
     k[0][0]=1;
     k[1][0]=2;
     k[2][0]=3;
     k[0][1]=4;
     k[1][1]=5;
     k[2][1]=6;
     k[0][2]=7;
     k[1][2]=8;
     k[2][2]=9;
}
```

ou encore.

```
/* Déclaration et initialisation d'un tableau de 9 éléments
   à deux dimensions */
int k[3][3]={1,2,3,4,5,6,7,8,9};

main()
{
            .
            .
            .
}
```

ou encore.

```
/* Déclaration et initialisation d'un tableau de 9 éléments
   à deux dimensions */
int k[3][3]={       {1,2,3},
                    {4,5,6},
                    {7,8,9}};

main()
{
            .
            .
            .
}
```

III.2.3 Les chaînes de caractères.

Elles sont vues par le C comme un tableau de caractères se terminant par un code de fin appelé le caractère nul '\0'.

Syntaxe:
<classe> <type> <identificateur>[nb de caractères+1];

Exemple:

```
char message[10];
```

On a définit un tableau de caractères de 10 éléments. Le message ne pourra contenir au plus que neuf caractères car le dixième est réservé pour le caractère de fin '\0'.

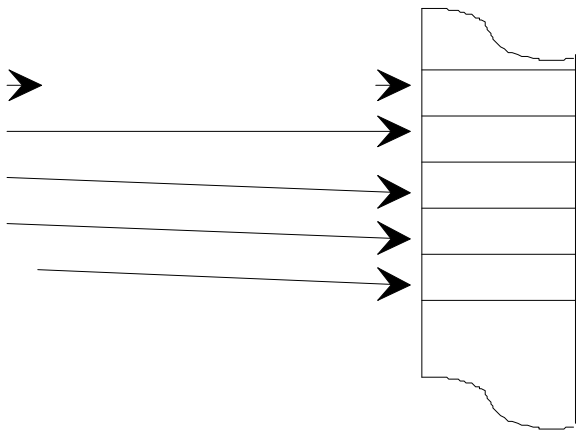
Initialisation de chaîne:

```
/* Déclaration d'une chaîne de caractères de 9 éléments */
char message[10];

main()
{
     message[0]='R';
     message[1]='E';
     message[2]='N';
     message[3]='E';
     message[4]='\0'; /* Caractère de fin */
}
```

ou

```
/* Déclaration d'une chaîne de caractères de 9 éléments */
char message[10]="RENE";

main()
{
            .
            .
            .
}
```

Représentation mémoire:



Il faut éviter d'initialiser une chaîne de cette façon.

```
/* Déclaration d'une chaîne de caractères de 9 éléments */
char message[10];

main()
{
     message="RENE"; /* Interdit en C */
}
```

On ne peut pas initialiser une chaîne de caractères de cette façon car "`message`" est considéré comme une adresse par le compilateur. Pour que l'initialisation soit correcte il faut utiliser une fonction de copie de caractère (`strcpy`), celle-ci recopie un à un tous les caractères de `"RENE"` à partir de "`message`".

Remarque: `strcpy` est déclarée dans le fichier `string.h`.

Exemple:

```
#include <stdio.h> /* Pour Printf */
#include <string.h> /* Pour Strcpy */

/* Déclaration d'une chaîne de caractères de 9 éléments */
char message[10];

main()
{
     /* Initialisation correcte d'une chaîne */
     strcpy(message,"RENE");
     /* Affichage du message */
     printf("%s",message);
}
```

Il existe une multitude de fonctions de manipulations de chaînes, voir le chapitre les fonctions de la librairie standard.

III.2.4 Les variables dans les blocs

Il est temps de vous parler des endroits où l'on peut déclarer les variables. Deux possibilités vous sont offertes:

- **Avant le programme principal, les variables sont dites globales. C'est à dire qu'elles sont accessibles n'importe où dans le programme.**
- **Dans un bloc, les variables sont dites locales. C'est à dire qu'elles existent que dans le bloc où elles ont été déclarées.**

Rappel: Un bloc d'instructions commence par { et se finit par }.

Pour bien comprendre la notion de variable globale et de variable locale, quelque exemples:

```
#include <stdio.h>

/* Déclaration d'une variable globale */
int i;

main()
{
     i=1; /* Initialisation de i */
     printf("%d",i); /* Affichage de i */
}
```

Dans l'exemple ci-dessus la variable `i` est globale à l'ensemble du logiciel. La valeur de `i` n'est accessible de n'importe où.

```
#include <stdio.h>

main()
{
     /* Déclaration d'une variable locale */
     int i;

     i=1; /* Initialisation de i */
     printf("%d",i); /* Affichage de i */
}
```

Dans l'exemple ci-dessus la variable `i` est locale au bloc `main`. La valeur de `i` est accessible que dans le bloc `main`.

```
#include <stdio.h>

main()
{
     {  /* Début de bloc */
            int i; /* Déclaration d'une variable locale i*/
            i=1;   /* Initialisation de i */
            printf("i  = %d\n",i);        /* Affichage de i */
            /* \n provoque un retour à la ligne */
     }  /* Fin du bloc */
     printf("i = %d\n",i); /* Affichage de i */
}
```

Ci vous tapez l'exemple ci-dessus, le compilateur va refuser d'exécuter le programme car la variable locale i existe seulement dans le bloc où elle a été créée.

<u>Autre exemple</u>:

```
#include <stdio.h>
int i; /* Déclaration d'une variable globale */

main() /* Début du programme principal */
{
     i=5; /* Initialisation de la variable globale i */
     printf("i du programme principal = %d\n",i); /* Affichage de i */
     { /* Début de Bloc */
            int i;  /* Déclaration d'une variable locale i*/
            i=1; /* Initialisation de i */
            printf("i dans le bloc = %d\n",i); /* Affichage de i */
     } /* Fin de bloc */
     printf("i du programme principal = %d\n",i); /* Affichage de i */
} /* Fin du programme principal */
```

Si vous tapez ce programme vous obtiendrez l'affichage suivant:

```
i du programme principal = 5
i dans le bloc = 1
i du programme principal = 5
```

Que se passe t'il dans le programme ?

1)Déclaration de i comme variable globale.
2)Initialisation de i avec la valeur 5.
3)Affichage de la variable globale i.
4)Déclaration de i comme variable locale, alors le programme réserve de la mémoire dans la machine pour la variable locale i et seule la variable locale i est accessible. Si une autre variable globale portait un autre nom que i elle serait accessible.
5)Affichage de la variable locale i.
6)Fin de bloc le programme supprime la réservation mémoire de la variable locale.
7)Affichage de la variable globale i.

III.2.4 Les types pré définis.

Le langage C a la possibilité de définir de nouveaux types de variables avec l'instruction `Typedef`, cela peut être dans certain cas très intéressant.

Syntaxe:
typedef <ancien_type> <nouveau_type>

Exemple:

```
#include <stdio.h>

/* Définition de type entier : int */
typedef int entier;
/* Définition de type caractere : char */
typedef char caractere;

/* Déclaration de variables de type entier */
entier a,b; /* Equivalent à int a,b; */
/* Déclaration de variables de type caractère */
caractere car1,car2; /* Equivalent à char car1,car2; */

main()
{
     .
     .
     .
}
```

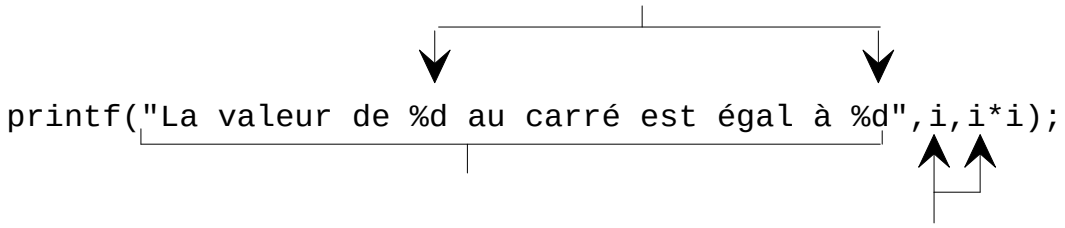
IV Les fonctions d'affichage et de saisie.

IV.1 La fonction d'affichage.

Elle permet d'afficher des messages et ou des valeurs de variables sous différents formats.

Syntaxe:
printf(<"Format">,identificateur1, ...,identificateurn);

Le format: Il indique comment vont être affichés les valeurs des variables. Il est composé de texte et de codes d'affichages suivant le type de variable.



Codes d'affichages:

Type	*Format*
Entier décimal	%d
Entier Octal	%o
Entier Hexadécimal	%x
Entier Non Signé	%u
Caractère	%c
Chaîne de caractères	%s
Flottant	%f
Long Entier	%ld
Long flottant	%lf
Long entier non signé	%lu

Important: Au début d'un programme utilisant les fonctions d'affichages et de saisies il est nécessaire d'écrire `#include <stdio.h>`, car toutes les fonctions sont déclarées dans ce fichier d'en-tête.

Exemple:

```
#include <stdio.h>

int i; /* Déclaration de variable globale */

main()
{
     i=23; /* Initialisation de i */
     printf(" i(dec) = %d\n",i);  /* Affichage de i en décimal */
     printf(" i(octal) =%o\n",i); /* Affichage de i en octal */
     printf(" i(hex)= %x\n",i);   /* Affichage de i en Hexadécimale */
}
```

On peut aussi mettre des codes de contrôles:

Code de contrôle	*Signification*
\n	Nouvelle ligne
\a	Bip code ascii 7
\r	Retour chariot
\b	Espace arrière
\t	Tabulation
\f	Saut de Page
****	Antislash
\"	Guillemet
\'	Apostrophe
\'0'	Caractère nul
\0ddd	Valeur octale (ascii) ddd
\xdd	Valeur hexadécimale dd

Exemple:

```
#include <stdio.h>

main()
{
     printf("Salut je suis:\n\t Le roi \n\t\t A bientôt\a");
}
```

Vous verrez:

```
Salut je suis
        Le roi
                A bientôt
```

Plus un Bip sonore.

On peut aussi définir le type d'affichage des variables pour les nombres signés et ou flottants en rajoutant des caractères de remplissages.

- **Un caractère de remplissage `'0'` au lieu de `' '` pour les numériques.**
- **Un caractère de remplissage `'-'` qui permet de justifier à gauche l'affichage sur la taille minimale (défaut à droite).**
- **Un caractère de remplissage `'+'` qui permet l'affichage du signe.**
- **Un nombre qui précise le nombre de caractères qui doit être affiché suivit d'un point et d'un nombre précisant combien de chiffres après la virgule doivent être affichés.**

Syntaxe:
<Nb caractères afficher>.<Nombre de chiffres significatifs>

Exemple:

```
#include <stdio.h>

float i;

main()
{
            i=15.6;
       /* Affichage de i normal */
            printf("%f\n",i);
       /* Affichage de i avec 10 caractères */
            printf("%10f\n",i);
            /* Affichage de i avec 10 caractères
            et 2 chiffres après la virgule */
            printf("%10.2f\n",i);
            /* Affichage de i avec 10 caractères
            et 2 chiffres après la virgule
            et à gauche */
            printf("%-10.2f\n",i);
            /* Affichage de i avec 10 caractères
            et 2 chiffres après la virgule
            et à gauche
            avec l'affichage du signe */
            printf("%+-10.2f\n",i);
            /* Affichage de i avec 10 caractères
            et 2 chiffres après la virgule
            avec des zéros avant le valeur */
            printf("%010.2f\n",i);
}
```

Vous verrez:

```
    15.600000
  15.60000
     15.60
 15.6
 +15.6
 000015.60
```

Il existe d'autres fonctions qui permettent d'affichées des variables.

- **putchar: Elle permet d'afficher un caractère à l'écran.**

Syntaxe: ***putchar(identificateur1);***

Exemple:

```
#include <stdio.h>

char car1,car2;

main()
{
     car1='A';
     car2=0x42;
     /* Affichage du caractère 1 -> donne un A */
     putchar(car1);
     /* Affichage du caractère 2 -> donne un B */
     putchar(car2);
     /* Affichage du caractère C */
     putchar('C');
     /* Retour de ligne */
     putchar('\n');
}
```

- **Puts: Elle permet d'afficher une chaîne de caractères à l'écran.**

Syntaxe: *puts(identificateur de chaîne de caractères);*

Exemple:

```
#include <stdio.h>
#include <string.h>
/* Déclaration et Initialisation de message1 */
char message1[10]="Bonjour";
/* Déclaration de message2 */
char message2[10];

main()
{
     /* Initialisation de message2 */
     strcpy(message2,"Monsieur");
     /* Affichage du message1 */
     puts(message1);
     /* Affichage du message2 */
     puts(message2);
}
```

vous verrez

```
Bonjour
Monsieur
```

IV.2 La fonction de saisie.

Elle permet de saisir des valeurs de variables formatées à partir du clavier. Comme `printf` elle est composée d'un format et des identificateurs de variables à saisir. A la différence de `printf`, le format ne peut contenir de texte, il est juste composé du format des valeurs à saisir.

```
Syntaxe:
scanf(<"Format">,&identificateur1, ...., &identificateurn);
```

Remarque: Le symbole `&` est obligatoire devant les identificateurs car scanf attend des adresses et non des valeurs, sauf devant un identificateur de chaîne de caractères.

Les codes d'affichages pour `printf` deviennent les codes d'entrées pour `scanf`.

Type	*Format*
Entier décimal	%d
Entier Octal	%o
Entier Hexadécimal	%x
Entier Non Signé	%u
Caractère	%c
Chaîne de caractères	%s
Flottant	%f
Long Entier	%ld
Long flottant	%lf
Long entier non signé	%lu

Pour l'utilisation de `scanf` il faut inclure le fichier `stdio.h` au début du programme.

Exemples:

Saisie d'une variable.

```
#include <stdio.h>
/* Déclaration de constante */
#define PI 3.14159
/* Déclaration de variable rayon et périmètre */
float rayon,perimetre;

main()
{
     /* Affichage de "Donne ton Rayon en mètre ?" */
     puts("Donne ton Rayon en mètre ?");
     /* Saisie du Rayon */
     scanf("%f",&rayon);
     /* Calcul du perimètre */
     perimetre=2*PI*rayon;
     /* Deux sauts de ligne */
     printf("\n\n");
     /* Affichage de périmetre */
     printf("Le périmètre = %f",perimetre);
}
```

Saisie de plusieurs variables.

```
#include <stdio.h>
/* Déclaration de variable rayon et périmètre */
float a,b,c,det;

main()
{
     /* Affichage de "Donne les valeurs de a,b et c ?" */
     puts("Donne les valeurs de a,b et c ?");
     /* Saisie de a,b,c */
     scanf("%f %f %f",&a,&b,&c);
     /* Calcul du déterminant */
     det=(b*b)+(4*a*c);
     /* Deux sauts de ligne */
     printf("\n\n");
     /* Affichage du déterminant */
     printf("Le déterminant = %f",det);
}
```

Saisie du variable du chaîne de caractères.

```
#include <stdio.h>
/* Déclaration de variable nom */
char nom[10];

main()
{
     /* Affichage de "Quel est ton nom ?" */
     puts("Quel est nom ?");
     /* Saisie du nom */
     scanf("%s",nom);
     /* Trois sauts de ligne */
     printf("\n\n\n");
     /* Affichage de nom */
     printf("%s",nom);
}
```

Il existe d'autres fonctions qui permettent de saisir des variables.

- **getchar: Elle permet de saisir un caractère au clavier.**

Syntaxe:
```
identificateur1 = getchar( void );
```

Exemple:

```
#include <stdio.h>

char car1;

main()
{
     /* Affichage de "Tapez un caractère ?" */
     printf("Tapez un caractère ?");
     /* Saisie d'un caractère */
     car1=getchar();
     /* Changement de ligne */
     putchar('\n');
     /* Affichage du caractère saisie */
     printf("Le caractère saisie = %c",car1);
}
```

- **gets: Elle permet de saisir une chaîne de caractères au clavier.**

Syntaxe:
```
gets(identificateur de chaîne de caractères);
```

Exemple:

```
#include <stdio.h>

/* Déclaration du chaîne de 19 caractères */
char nom[20];

main()
{
     /* Affichage de "Tapez votre nom ?" */
     printf("Tapez votre nom ?");
     /* Saisie d'un caractère */
     gets(nom);
     /* Changement de ligne */
     putchar('\n');
     /* Affichage de la chaîne saisie */
     printf("Votre nom est %s",nom);
}
```

V Les opérateurs

Ce sont eux qui régissent toutes les opérations ou transformations sur les valeurs de variables. Un problème courant est de savoir quel est l'opérateur qui est le plus prioritaire sur l'autre. Pour résoudre ce problème il est conseillé de mettre des parenthèses pour bien séparer les différentes opérations.

V.1 L'opérateur d'affectation

C'est l'opérateur le plus utilisé. Il permet de modifier la valeur d'une variable. L'affectation est toujours effectuée de la droite vers la gauche. En effet le programme commence par évaluer la valeur de l'expression puis l'affecte avec le signe = à la variable par son identificateur.

Syntaxe:
```
<identificateur> = <expression>;
```

V.2 Les opérateurs arithmétiques.

+	Addition
-	Soustraction ou changement de signe
*	Multiplication
/	Division
%	Modulo (Reste)

Remarque: La multiplication et la division restent prioritaire sur les opérateurs arithmétiques.

V.3 Les opérateurs d'incrémentation et de décrémentation.

++	Incrémente de 1
-	Décrémente de 1

- **Si l'opérateur d'incrémentation ou de décrémentation est placé avant L'identificateur alors la variable sera incrémetée ou décrémentée avant d'être utilisée.**

Exemple:

```
#include <stdio.h>
/* Déclaration de variable a et b */
int a,b;

main()
{
     /* Initialisation de a et b */
     a=2;
     b=3;
     /* Affichage de a avec incrémentation avant l'utilisation */
     printf("a = %d\n",++a);
     /* Affichage de b avec décrémentation avant l'utilisation */
     printf("b = %d",--b);
}
```

vous verrez

```
a=3
b=2
```

- **Si l'opérateur d'incrémentation ou de décrémentation est placé après l'identificateur alors la variable sera incrémetée ou décrémentée après avoir été utilisée.**

Exemple:

```
#include <stdio.h>
/* Déclaration de variables a et b */
int a,b;

main()
{
     /* Initialisation de a et b */
     a=2;
     b=3;
     /* Affichage de a avec incrémentation après l'utilisation */
     printf("a = %d\n",a++);
     /* Affichage de b avec décrémentation après l'utilisation */
     printf("b = %d\n",b--);

     /* Affichage de a */
     printf("a = %d\n",a);
     /* Affichage de b */
     printf("b = %d",b);
}
```

vous verrez

```
a=2
b=3
a=3
b=2
```

V.4 Les opérateurs binaires.

Ils permettent d'agir sur les bits constituants les variables de type entier.

&	ET
|	OU
^	OU Exclusif
~	Non (Complément à 1)
>>	Décalage à droite
<<	Décalage à gauche

Exemple:

```
#include <stdio.h>
/* Déclaration deux octets */
unsigned char porta,i;

main()
{
     porta=0x23;    /* Initialisation */

     i=porta & 0xF0; /* Masquage ET */
     printf("i(hex) = %x\n\n",i);

     i=porta | 0x05; /* Masquage OU */
     printf("i(hex) = %x\n\n",i);

     i=~porta; /* Complément à 1 */
     printf("i(hex) = %x\n\n",i);

     i=porta>>1; /* Décalage à droite de 1 */
     printf("i(hex) = %x\n\n",i);

     i=porta<<4; /* Décalage à gauche de 4 */
     printf("i(hex) = %x\n\n",i);
}
```

vous verrez

```
i(hex) = 20

i(hex) = 27

i(hex) = dc

i(hex) = 11

i(hex) = 30
```

V.5 Les opérateurs combinés.

Ils réalisent une opération avec une variable et affectent le résultat à cette même variable. Ils sont constitués d'un opérateur arithmétique ou binaire, avec l'opérateur d'affectation.

+=	Addition et affectation
-=	Soustraction et affectation
*=	Multiplication et affectation
/=	Division et affectation
%=	Modulo et affectation
&=	ET et affectation
|=	OU et affectation
^=	OU exclusif et affectation
<<=	Décalage à gauche et affectation
>>=	Décalage à droite et affectation

Exemple:

```
#include <stdio.h>
/* Déclaration d'un entier */
int i;

main()
{
     i=2;    /* Initialisation */

     i+=3; /* i=i+3 -> i=5 */
     printf("i = %d\n\n",i);

     i*=2; /* i=i*2 -> i=10 */
     printf("i = %d\n\n",i);

     i<<=1; /* i=i<<1 -> i=20 */
     printf("i = %d\n\n",i);
}
```

vous verrez

```
i = 5

i = 10

i = 20
```

V.6 Les opérateurs relationnels

Ils sont utilisés pour les structures conditionnelles, de choix et itératives, voir chapitres suivants. Ils permettent de comparer une variable par rapport à une autre variable ou une valeur ou une expression. Le résultat ne peut être que VRAI ou FAUX, on parle de valeur booléen.

- FAUX correspond à 0.
- VRAI correspond à toute valeur différente de 0.

>	Supérieur à
>=	Supérieur ou égal à
<	Inférieur
<=	Inférieur ou égal à
==	Egal à
!=	Différent

Exemple:

```
#include <stdio.h>
/* Déclaration d'entiers */
int a,b;
int res;

main()
{
     a=2,b=3;

     res= (a>3); /* FAUX donc res=0 */
     printf("res = %d\n",res);

     res= (a>b); /* FAUX donc res=0 */
     printf("res = %d\n",res);

     res= (a<b); /* VRAI donc res différent de 0 */
     printf("res = %d\n",res);

     res= (a==b); /* FAUX donc res=0 */
     printf("res = %d\n",res);
}
```

vous verrez

```
res = 0
res = 0
res = 1
res = 0
```

Attention l'opérateur relationnel == (égalité) n'est pas à confondre avec l'opérateur d'affectation =. c'est le piège classique des débuts.

V.7 Les opérateurs logiques.

Ils sont utilisés exactement comme les opérateurs relationnels

!	**Négation**
&&	**ET Logique**
||	**OU Logique**

Exemple:

```
#include <stdio.h>
/* Déclaration d'entiers */
int a,b,c;
int res;

main()
{
     a=2,b=3,c=5;

     res= ( (a>3) && (c>5) );
     /* (a>3) faux ET (c>5) faux DONC res=0(faux) */
     printf("res = %d\n",res);

     res= ( (b>2) || (c<4) );
     /* (b>2) vrai OU (c<4) faux DONC res différent de 0(vrai) */
     printf("res = %d\n",res);

     res= !(a<b);
     /* (a<b) vrai -> !(Non) -> faux DONC res=0(faux) */
     printf("res = %d\n",res);
}
```

vous verrez

```
res = 0
res = 1
res = 0
```

V.8 L'opérateur ternaire conditionnel.

Cet opérateur permet de prendre une décision. Si la condition est vrai alors le programme exécutera l'expression situé entre ? et : sinon l'expression entre : et ;.

Syntaxe:
<condition> ? <expression(vrai) : expression(faux)>;

Exemple:

```
#include <stdio.h>
/* Déclaration d'entiers */
int a,b,c;
char car;
main()
{
     a=2,b=4;
     car='B';  /* Code ASCII de B c'est 66(dec) */

     /* Maximum de a et b */
     c = (a>b) ? a : b;
     /* a>b faux DONC c=b=4 */
     printf("c = %d\n",c);

     /* Passage en minuscule */
     car = (( car>= 'A') && (car<='Z') ) ? car+'a'-'A' : car;
     /* car='B' => car>='A'->vrai et car<='Z'->vrai
        DONC car=car+'a'-'A' => car=car+32
        => car=66+32 => car=98 (98 est le code ASCII de 'b' */
     printf("car = %c\n",car);
}
```

vous verrez

```
c = 4
car = b
```

V.9 L'opérateur virgule

Il permet de séparer:

- **la définition de plusieurs variables.**
- **Les instructions de la structure itérative `for` (voir chapitre structure itérative).**
- **Les paramètres dans l'appel de fonction (voir chapitre fonction).**

Exemple de définition de plusieurs variables:

```
#include <stdio.h>
/* Déclaration d'entiers */
int a,b,c,d,e,f;

main()
{
     .
     .
     .
}
```

V.10 L'opérateur de taille.

Il donne la taille d'un type ou d'une variable. C'est l'opérateur de portabilité, car il permet de connaître la taille suivant la machine.

Syntaxe:
taille = sizeof(identificateur ou type)

Exemple:

```
#include <stdio.h>
/* Déclaration d'entiers */
int i;

main()
{
     /* taille de la variable i en octet(s) */
     printf("La taille de i = %d\n", sizeof(i) );

     /* taille du type char en octet(s) */
     printf("La taille d'un char %d\n", sizeof(int) );
}
```

vous verrez

```
La taille de i = 2
La taille d'un char = 1
```

V.11 L'opérateur de conversion de type.

Il existe deux conversions possibles:

- **La conversion implicite. Elle est effectuée pour évaluer le même type de données lors d'évaluation d'expressions. Les conversions systématiques de `char` en `int` et en `float` en `double`, la conversion se fait toujours du type le plus petit vers le plus long (exemple: de `char` vers `double`).**

- **La conversion explicite. On peut changer le type d'une variable vers un autre type en utilisant l'opérateur `cast` (type) en le mettant devant l'identificateur de variable à convertir.**

Exemple:

```
#include <stdio.h>
/* Déclaration d'entiers */
char car;
int  a,b,c;
float g;

main()
{
     a=4;
     b=7;
     c=0x41; /* Code Ascii de 'A' */

     /* Conversion implicite de a et b en float */
     g =  (a + b) /100. ;
     printf("g= %f\n",g);

     /* Conversion explicite c en char */
     car = (char) c +3;
     /* c est de type entier et sera converti en char */
     printf("car = %c\n",car);
}
```

vous verrez

```
g = 0.110000
car = D
```

Remarque: Dans l'affectation de `g`, le compilateur évalue d'abord la partie droite `(a+b)/100.` `a` et `b` sont de type entier, on a ajouté un point derrière `100.` pour forcer le compilateur a éffectuer l'évaluation en flottant.

V.12 La priorité des opérateurs.

Un tableau vaut mieux qu'un long discours, des plus prioritaire (haut et gauche du tableau) au moins (bas et droite du tableau).

Le plus priorirtaire ➔ *Le moins prioritarire*
`() [] -> .`
`! - ++ -- - * & (case) sizeof`
`* / %`
`+ -`
`<< >>`
`== !=`
`&`
`^`
`|`
`&&`
`||`
`? :`
`= += -= *= /= %= >>= <<= &= |= ^=`
`,`

Le plus priorirtaire ↓ *Le moins prioritarire*

VI Les structures conditionnelles.

Elles permettent en fonction d'une condition, de choisir de faire une ou un bloc d'instructions plutôt qu'un autre.

VI.1 La structure SI ALORS.

C'est la structure de base.

```
Syntaxe:
if (condition) instruction;

ou

if (condition)
{
      instruction1;
            .
            .
            .
      instructionn;
}
```

Exemple:

```
#include <stdio.h>

int a,b;

main()
{
     /* Saisie de a  et de b */
     printf("Donnez les valeurs de a et de b ");
     scanf("%d %d",&a,&b);

     /* Structure SI ALORS */
     if (a<b) printf("a=%d est inférieur à b=%d\n",a,b);
}
```

ou

```
#include <stdio.h>

int a,b;

main()
{
     /* Saisie de a  et de b */
     printf("Donnez les valeurs de a et de b ");
     scanf("%d %d",&a,&b);

     /* Structure SI ALORS */
     if (a>b)
     {
            printf("a=%d est supérieur à b=%d\n",a,b);
            printf("\n");
     }
}
```

VI.2 La structure SI ALORS SINON.

Si la condition est vraie <u>alors</u> elle exécute l'instruction ou le bloc d'instructions qui suit le `if`(si), par contre si la condition est <u>fausse</u> alors elle exécute l'instruction ou le bloc d'instructions qui suit `else` (sinon).

```
Syntaxe:
if (condition) instruction1; else instruction2;

ou

if (condition)
{
      instruction1;
            .
            .
            .
}
else
{
      instruction2;
            .
            .
            .
}
```

<u>Exemple</u>:

```
#include <stdio.h>

int a,b;

main()
{
     /* Saisie de a  et de b */
     printf("Donnez les valeurs de a et de b ");
     scanf("%d %d",&a,&b);

     /* Structure SI ALORS SINON */
     if (a<b) printf("a<b"); else printf("a>b");

}
```

ou

```
#include <stdio.h>

int a,b;

main()
{
     /* Saisie de a  et de b */
     printf("Donnez les valeurs de a et de b ");
     scanf("%d %d",&a,&b);

     /* Structure SI ALORS SINON */
     if (a>b)
     {
            printf("a=%d est supérieur à b=%d",a,b);
            printf("\n");
     }
     else
     {
            printf("a=%d est inférieur à b=%d",a,b);
            printf("\n");
     }
}
```

VI.3 La structure SI ALORS SINON SI.

Si la condition est <u>vraie</u> alors elle exécute l'instruction ou le bloc d'instructions qui suit le `if(si)`, par contre si la condition est <u>fausse</u> alors elle teste de nouveau une condition et exécute l'instruction ou le bloc d'instructions qui suit `else if(sinon si)`.

```
Syntaxe:
if (condition1) instruction1;
else if (condition2) instruction2;
                     .
                     .
else if (conditionn) instructionn;

ou

if (condition1)
{
     instruction1;
          .
          .
}
else if (condition2)
{
     instruction2;
          .
          .
}
else if (conditionn)
{
     instructionn;
          .
          .
}
```

<u>Exemple</u>:

```
#include <stdio.h>

int temp;

main()
{
     /* Saisie de a  et de b */
     printf("Donnez une température ? ");
     scanf("%d",&temp);

     /* Structure SI ALORS SINON SI*/
     if ( (temp>0) && (temp<10) ) printf("Il fait froid\n");
     else if ( (temp>=10) && (temp<20) ) printf("Il fait bon\n");
     else if ( (temp>=20) && (temp<30) ) printf("Il fait chaud\n");
     else if (temp>=30) printf("Il fait trop chaud");

}
```

ou

```
#include <stdio.h>

int temp;

main()
{
     /* Saisie de a  et de b */
     printf("Donnez une température ? ");
     scanf("%d",&temp);

/* Structure SI ALORS SINON SI*/
     if ( (temp>0) && (temp<10) )
     {
            printf("Il fait froid");
            printf("\n");
     }
     else if ( (temp>=10) && (temp<20) )
     {
            printf("Il fait bon");
            printf("\n");
     }
     else if ( (temp>=20) && (temp<30) )
     {
            printf("Il fait chaud");
            printf("\n");
     }
     else if (temp>=30)
     {
            printf("Il fait trop chaud");
            printf("\n");
     }
}
```

VI.4 La structure de choix.

Elle permet en fonction de différentes valeurs d'une variable de faire plusieurs actions, si aucune valeur n'est trouvée alors ceux sont les instructions qui suivent `default` qui sont exécutées .

```
Syntaxe:
switch (identificateur)
{
      case valeur1 : instruction1
                              ou
                        {
                              .
                              .
                        } break;
case valeuurm : instruction2
                              ou
                        {
                              .
                              .
                        } break;
                        .
                        .
                        .
case valeurk: instructionj
                              ou
                        {
                              .
                              .
                        } break;
default : instructioni
                        ou
                  {
                        .
                        .
                  } break;
}
```

Exemple:

```
#include <stdio.h>

char choix;

main()
{
     /* Affichage du menu */
     printf("\n\n\n\n\n");
     printf("\t\t\t\t MENU\n");
     printf("\t a --> ACTION 1\n");
     printf("\t b --> ACTION 2\n");
     printf("\t c --> ACTION 3\n");
     printf("\t d --> ACTION 4\n");
     printf("\n\n\t\t tapez sur une touche en minuscule  ");

     /* saisie de la touche */
     choix=getchar();

     /* Structure de choix switch*/
     switch(choix)
     {
            case 'a': printf("Exécution de l'ACTION1");break;
            case 'b': printf("Exécution de l'ACTION2");break;
            case 'c': printf("Exécution de l'ACTION3");break;
            case 'd': printf("Exécution de l'ACTION4");break;
            default : printf("Mauvaise touche, pas d'ACTION");
     }
}
```

ou

```
#include <stdio.h>

char choix;

main()
{
     /* Affichage du menu */
     printf("\n\n\n\n\n");
     printf("\t\t\t\t MENU\n");
     printf("\t a --> ACTION 1\n");
     printf("\t b --> ACTION 2\n");
     printf("\t c --> ACTION 3\n");
     printf("\t d --> ACTION 4\n");
     printf("\n\n\t\t tapez sur une touche en minuscule  ");

     /* saisie de la touche */
     choix=getchar();

     /* Structure de choix switch*/
     switch(choix)
     {
            case 'a':{
                           printf("Exécution de l'ACTION1");
                           printf("\n");
                      } break;
            case 'b':{
                           printf("Exécution de l'ACTION2");
                           printf("\n");
                      } break;
            case 'c':{
                           printf("Exécution de l'ACTION3");
                           printf("\n");
                      } break;
            case 'd':{
                           printf("Exécution de l'ACTION4");
                           printf("\n");
                      } break;
            default :{
                           printf("Mauvaise touche, pas d'ACTION");
                           printf("\n");
                      }
     }
}
```

VII Les structures itératives ou boucles.

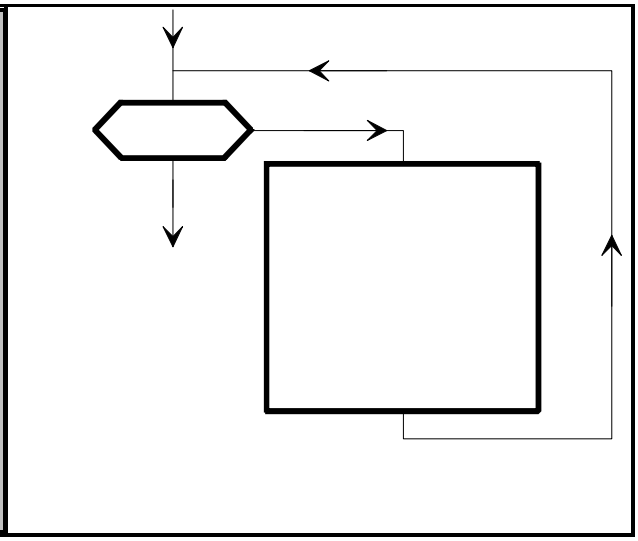
Une structure itérative est la répétition d'une ou plusieurs instructions tant que la condition de sortie est *VRAIE*, en fonction des différents types de structures itératives la condition pourra être testée en début ou en fin de la structure.

VII.1 La structure <tant *que refaire*>.

Dans cette structure la condition est testée au début .

Syntaxe:

```
while (condition) instruction;
ou
while (condition)
{
     instruction1;
           .
           .
           .
     instructionn;
}
```



Exemple:

```
#include <stdio.h>

int i;

main()
{
     /* Boucle while <tant que refaire> */
     while(i!=10) printf("i= %d\n",i++);
}
```

ou

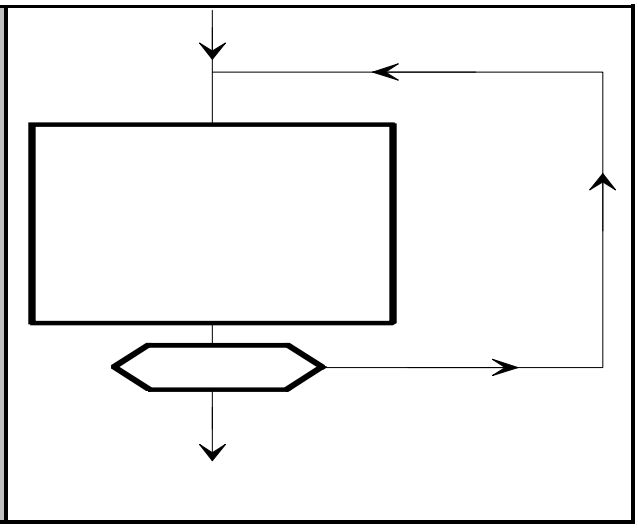
```
#include <stdio.h>

int i;

main()
{
     /* Boucle while <tant que refaire> */
     while(i!=10)
     {
            printf("i= %d\n",i);
            i++;
     }
}
```

VII.2 La structure <*faire ..... tant que*>.

Dans cette structure la <u>condition</u> est testée à la <u>fin</u>.

<u>Syntaxe:</u>

```
do      instruction;      while
(condition) ;
ou
do
{
     instruction1;
           .
           .
           .
     instructionn;
}
while (condition);
```



<u>Exemple</u>:

```
#include <stdio.h>

int i;

main()
{
     i=0;

     /* Boucle while <tant que refaire> */
     do printf("i= %d\n",i++); while(i!=10);
}
```

ou

```
#include <stdio.h>

int i;

main()
{
     i=0;

     /* Boucle while <tant que refaire> */
     do
     {
            printf("i= %d\n",i);
            i++;
     }
     while(i!=10);
}
```

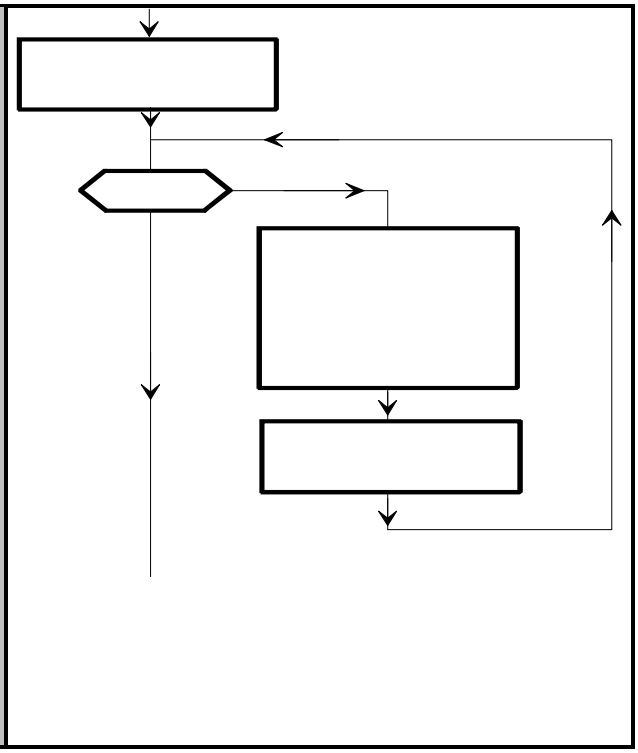
VII.3 La structure <*faire .....jusqu'à*>.

Dans cette structure la <u>condition</u> est testée au début. Elle est très intéressante car elle est composée de trois parties:

- **L' instruction ou plusieurs instructions d'Initialisation qui sont exécutées une seule fois au début de la structure.**
- **L'instruction ou le bloc d'instructions du corps de la structure qui est exécute à chaque itération.**
- **L'instruction ou plusieurs instructions qui sont exécutées à la fin de chaque itération.**

<u>Syntaxe:</u>

```
for
(instruction1,....,instruction
n d'init ; condition ;
instruction2,.........,instruct
ion m de fin)
 instruction de corps;
ou
for
(instruction1,....,instruction
n d'init ; condition ;
instruction2,.........,instruct
ion m de fin)
{
          instruction 3;
                    .
                    .
                    .
          instruction k;
}
```



<u>Exemple</u>:

```
#include <stdio.h>

char car;

main()
{
    /* Affichage des codes ASCII des Lettres majuscules */
    for(car=65;car!=91;car++) printf("%c pour code ASCII: %d\n",car,car);
}
```

ou

```
#include <stdio.h>

char car;

main()
{
    car=65;

    /* Affichage des codes ASCII des Lettres majuscules */
    for(;car!=91;)
    {
           printf("%c pour code ASCII: %d\n",car,car);
           car++;
    }
}
```

VIII Les instructions de contrôle des structures itératives et conditionnelles.

VIII .1 L'instruction de sortie `break`.

Elle permet de sortir d'une structure à n'importe quel endroit.

Syntaxe:
```
break;
```

Exemple:

```
#include <stdio.h>

char car;

main()
{
     car=65;

     /* Affichage des codes ASCII des Lettres majuscules */
     for(;car!=91;)
     {
            printf("%c pour code ASCII: %d\n",car,car);
            /* Dès que car sera égal à H */
            /* alors on sortira de la boucle */
            if (car=='H') break;
            car++;
     }
}
```

VIII .2 L'instruction de branchement sur la condition de sortie `continue`.

Elle permet de se brancher sur la condition de sortie de la boucle à partir de n'importe quel endroit.

Syntaxe:
```
continue;
```

Exemple:

```
#include <stdio.h>

int i;

main()
{
     /* Affichage des nombres 1 à 10 et 20 à 30 */
     for(i=1;i!=30;i++)
     {
            /* Branchement sur la condition de sortie i!=30
               si i>10 ET i<20 */
            if ( (i>10) && (i<20) ) continue;
            printf("%d ",i);
     }
}
```

VIII .3 L'instruction de branchement `goto`.

Elle permet de se brancher sur un label n'importe où dans le programme.

Syntaxe:
<goto> label;

Exemple:

```
#include <stdio.h>

int i;

main()
{
     /* Affichage des nombres 1 à 10 et 20 à 30 */
     for(i=1;i!=30;i++)
     {
            /* Branchement sur le label FIN si
               si i>10 ET i<20 */
            if ( (i>10) && (i<20) ) goto FIN;
            printf("%d ",i);
            FIN:;
     }
}
```

IX Les structures, unions et énumération de données ou objets.

IX.1 Les structures de données.

Elles permettent de stocker pour un type de variable donné plusieurs variables caractérisant celui-ci, on parle encore d'objet. Les variables d'une structure ne sont pas accessibles en dehors de celle-ci.

Déclaration de structure.

- **Déclaration du type.**

```
Syntaxe:
     struct indetificateur_de_type;
     {
          type identificateur 1;
                .
                .
                .
          type identificateur n;
     };
```

- **Déclaration d'une structure ou objet.**

```
Syntaxe:
<classe> <struct> identificateur_de_type identificateur_d'objet
= {valeur1, ....., valeurn};
```

Exemple:

```
#include <stdio.h>

/* Déclaration du type de structure: Cercle */
struct cercle
{
     int x,y; /* Position du centre du cercle*/
     int rayon; /* Rayon du cercle */
};

/* Déclaration de trois structures de type cercle */
struct cercle cercle1,cercle2,cercle3;

/* Déclaration d'un tableau 5 structures de type cercle */
struct cercle tab_cercle[5];

main()
{
     .
     .
     .
}
```

ou

```
#include <stdio.h>

/* Déclaration du type de structure: Cercle */
struct cercle
{
     int x,y; /* Position du centre du cercle*/
     int rayon; /* Rayon du cercle */
};

/* Déclaration et Initialisation de deux structure de type cercle */
struct cercle cercle1={2,3,10};
struct cercle cercle2={4,5,9};

main()
{
}
```

IX.1.1 L'opérateur point "." d'accès aux données d'objets.

Il permet d'accéder aux différentes valeurs des variables d'un objet.

Syntaxe:
identificateur_d'objet . identificateur_de_variable = valeur;
ou
valeur = identificateur_d'objet . identificateur_de_variable;

Exemple:

```
#include <stdio.h>

/* Déclaration du type de structure: Cercle */
struct cercle
{
     int x,y; /* Position du centre du cercle*/
     int rayon; /* Rayon du cercle */
};

/* Déclaration de trois structures de type cercle */
struct cercle cercle1,cercle2,cercle3;

/* Déclaration d'un tableau 5 structures de type cercle */
struct cercle tab_cercle[5];

main()
{
      /* Initialisation des variables du premier élément
         du tableau de structure de cercle */
      tab_cercle[0].x=2;
      tab_cercle[0].y=3;
      tab_cercle[0].rayon=10;

      /* Initialisation des variables du cercle 3 */
      cercle3.x=2;
      cercle3.y=3;
      cercle3.rayon=10;

      /* Affichage des données du cercle1 */
      printf("Le cercle 1 a pour rayon %d\n",cercle1.rayon);
      printf("Le cercle 1 a pour centre x=%d,y=%d\n",cercle1.x,cercle1.y);
}
```

IX.1.2 Les champs de bits.

Ils permettent de manipuler les valeurs bit à bit d'une variable.

- **Déclaration du type.**

```
Syntaxe:
     struct identificateur_de_type;
     {
          type indentificateur_de_bits : nb_de_bits;
               .
               .
               .
          type identificateur_de_bits : nb_de_bits;
     };
```

- **Déclaration d'une structure ou objet de bits.**

```
Syntaxe:
<classe><struct>identificateur_de_type identificateur_d'objet;
```

exemple:

```
#include <stdio.h>

/* Déclaration du type de structure: Octet */
struct octet
{
     unsigned bit1:1;
     unsigned bit24:3;
     unsigned :2;
     unsigned bit67:2;
};

/* Déclaration d'un octet */
struct octet port;

main()
{
     /* Initialisation de port avec la valeur 0x45=%01000101 */
     port.bit1=1;
     port.bit24=2;
     port.bit67=1;

     /* Affichage de la valeur de port */
     printf("la valeur de port = %x",port);
}
```

IX.2 Les unions.

Elles permettent de stocker pour un objet donné plusieurs variables, mais à la différence des structures les données sont stockées à la même adresse. Ce type d'objet est utilisé pour tout programme qui est en relation directe avec les entrées sorties de la machine.

Déclaration d'union.

- **Déclaration du type.**

```
Syntaxe:
      union indetificateur_de_type;
      {
            type indentificateur 1;
                  .
                  .
                  .
            type identificateur n;
      };
```

- **Déclaration d'un objet de type union.**

```
Syntaxe:
<classe> <union> identificateur_de_type identificateur_d'objet
={valeur1, ....., valeurn};
```

Exemple:

```
#include <stdio.h>

/* Déclaration du type de structure: vari */
union vari
{
     int i;
     unsigned char a[2];
};

/* Déclaration d'une variable var de type vari */
union vari var;

main()
{
     /* Initialisation de l'union avec 1230 */
     var.i=1230;

     /* Rappel un entier occupe deux cases mémoires */
     /* Donc a[0]=206 */
     /* Et a[1]=4 */
     /* Donc var = var.a[1]*256+var.a[0] */
     printf("La valeur de var est %d",var.a[1]*256+var.a[0]);
}
```

IX.3 Les énumérations.

Elles permettent de donner des valeurs à un objet, cela revient à définir des constantes, ce type d'objet n'est guère utilisé en général.

Déclaration d'énumération.

- **Déclaration du type.**

```
Syntaxe:
      enum indetificateur_de_type;
      {
            élément 1 = valeur,
                  .
                  .
                  .
            élément n = valeur
      };
```

- **Déclaration d'un objet de type énumération.**

```
Syntaxe:
<classe> <enum> identificateur_de_type identificateur_d'objet;
```

Exemple:

```
#include <stdio.h>

/* Déclaration d'une énumération pour les jours de la semaine */
enum semaine { lundi=1,mardi,mercredi,jeudi,vendredi,samedi,dimanche };

/* Déclaration d'une variable jour de type semaine */
enum semaine jour;

main()
{
    /* Affichage du jour correspondant à vendredi */
    printf("Le jour de vendredi est %d",vendredi);
}
```

X Les pointeurs.

C'est toute la puissance du langage C. Ils sont ni plus ni moins que des variables contenant l'adresse d'autres variables.

X.1 L'opérateur d'adresse &.

Il permet de connaître l'adresse de n'importe quelle variable ou constante.

Exemple:

```
#include <stdio.h>

/* Déclaration d'un entier */
int i;

main()
{
     i=3;
     /* Affichage de la valeur et de l'adresse de i */
     printf("La valeur de i est %d\n",i);
     printf("L'adresse de i est %d\n",&i);
}
```

Pour les tableaux:
L'adresse du premier élément est *L'identificateur* ou *&identificateur[0]*.
L'adresse de l'élément *n* est *&identificateur[n-1]*.

Exemple:

```
#include <stdio.h>

/* Déclaration d'un tableau d'entiers */
int tab[3];

/* Déclaration d'une chaîne */
char nom[10]="Bonjour";

main()
{
     /* Affichage de l'adresse de tab */
     printf("La valeur de tab est %d\n",tab);
     printf("\t\t ou \n");
     printf("L'adresse de tab est %d\n",&tab[0]);

     /* Affichage de l'adresse de nom */
     printf("La valeur de nom est %d\n",nom);
     printf("\t\t ou \n");
     printf("L'adresse de nom est %d\n",nom);
}
```

X.2 Déclaration et manipulation de pointeur.

Un pointeur est une variable contenant l'adresse d'une autre variable, on parle encore d'indirection. La déclaration de pointeur se fait en rajoutant une étoile * devant *L'identificateur de pointeur.*

```
Syntaxe:
<classe> <type> *ptr_identificateur_de_pointeur;
```

Il est conseillé mais pas obligatoire de faire commencer chaque *identificateur de pointeur* par *ptr_* pour les distinguer des autres variables.

Exemple:

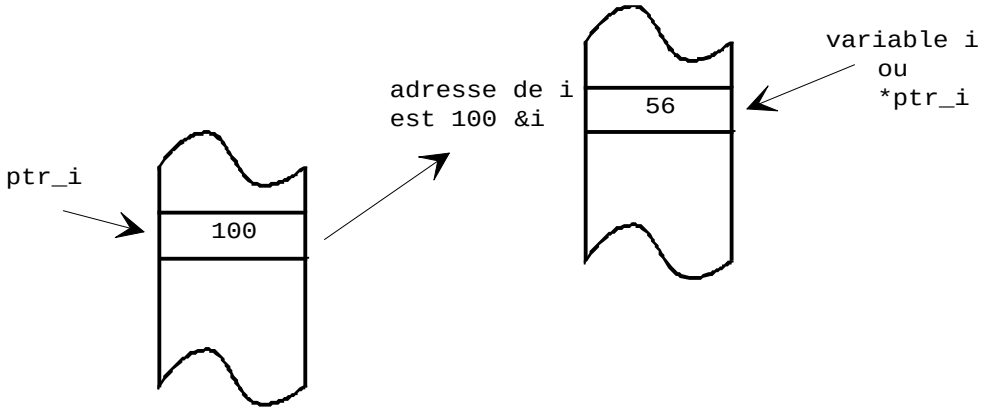
```
#include <stdio.h>

/* Déclaration d'un entier */
int i;

/* Déclaration d'un pointeur d'entier */
int *ptr_i;

main()
{
     /* Initialisation de ptr_i sur l'adresse de i */
     ptr_i=&i;
}
```

Pour pouvoir accéder à la valeur d'un pointeur il suffit de rajouter une étoile * devant *l'identificateur de pointeur*. L'étoile devant l'identificateur de pointeur veut dire *objet(valeur) pointer par*.

Exemple:

```
#include <stdio.h>

/* Déclaration d'un entier */
int i;

/* Déclaration d'un pointeur d'entier */
int *ptr_i;

main()
{
     /* Initialisation de i */
     i=5;

     /* Initialisation de ptr_i sur l'adresse de i */
     ptr_i=&i;

     /* Affichage de la valeur de i */
     printf("La valeur de i est %d\n",i);

     /* Changement de valeur i par le pointeur ptr_i */
     *ptr_i=56;

     /* Affichage de la valeur de i par le pointeur ptr_i */
     printf("La valeur de i est %d\n",*ptr_i);
}
```

Représentation mémoire.



Remarque: Si vous tapez l'exemple ci-dessus le pointeur `ptr_i` ne sera sûrement pas égal à `100`, car c'est le système d'exploitation de la machine qui fixe l'adresse de la variable `i`.

X.3 L'arithmétique des pointeurs.

On peut effectuer sur les pointeurs les opérations suivantes:

- **On peut incrémenter ou décrémenter, ce qui permet de sélectionner tous les éléments d'un tableau. Un pointeur sera toujours incrémenté ou décrémenté du nombre d'octets du type de variable pointée.**
  Exemple: **un `entier` occupe deux octets donc un pointeur de type `int` sera incrémenté ou décrémenté de `deux octets`.**

Exemple:

```
#include <stdio.h>

/* Déclaration et Initialisation d'un tableau d'entier */
int i[5]={4,-4,5,6,7};

/* Déclaration d'une variable de boucle j */
int j;

/* Déclaration d'un pointeur d'entier */
int *ptr_i;

main()
{
     /* Initialisation de ptr_i sur l'adresse de i */
     ptr_i=i; /* Equivalent à ptr_i=&i[0] */

     /* Affichage de tous les éléments du tableau */
     for(j=0;j!=5;j++)
     {
            /* Affichage des éléments du tableau */
            printf("i[%d]=%d\n",j,*ptr_i);

            /* Incrémentation du pointeur ptr_i de */
            /* 2 Octets car chaque élément de i est */
            /* un entiers donc deux octets */
            ptr_i++;

            /* Affichage de la valeur du pointeur i */
            printf("Le pointeur ptr_i=%d\n",ptr_i);
     }
}
```

- **Ajouter une constante `n`, qui permet au pointeur de se déplacer de `n` éléments dans un tableau. Le pointeur sera augmenté ou diminué de `n` fois le nombre d'octet(s) du type de variable.**

Exemple:

```
#include <stdio.h>

/* Déclaration et initialisation d'une chaîne de caractères */
char mes[10]="Morgan";

/* Déclaration d'un pointeur de caractère */
char *ptr_mes;

main()
{
     /* Initialisation de ptr_mes sur l'adresse de mes */
     ptr_mes=mes; /* Equivalent à ptr_mes=&mes[0] */

     /* Affichage du dernier caractère de mes */
     printf("Le dernier caractère de mes est %c\n",*(ptr_mes+5));
}
```

Ici `ptr_mes` sera augmenté de 5 octets car le type `char` réserve un octet.

Attention: `*(ptr_mes+5)` est complétement différent de `*ptr_mes+5`.
`*(ptr_mes+5)` c'est la valeur de l'objet pointé par `ptr_mes+5`.
`*ptr_mes+5` c'est l'addition de la valeur de l'objet pointé par `ptr_mes` et de 5.

Exemple:

```
#include <stdio.h>

/* Déclaration et initialisation d'une chaîne de caractères */
char mes[10]="Morgan";

/* Déclaration d'un pointeur de caractère */
char *ptr_mes;

main()
{
     /* Initialisation de ptr_mes sur l'adresse de mes */
     ptr_mes=mes; /* Equivalent à ptr_mes=&mes[0] */

     /* Affichage du dernier caractère de mes */
     printf("Le dernier caractère de mes est %c\n",*(ptr_mes+5));

     /* Affichage du caractère ayant pour code ASCII(M+5) -> R */
     printf("Le caractère de code ASCII(M+5) %c",*ptr_mes+5);
}
```

- **Additionner ou soustraire une constante `n`, qui permet au pointeur de se déplacer de `n` éléments dans un tableau. Le pointeur sera augmenté ou diminué de `n` fois le nombre d'octet(s) du type de variable.**

Exemple:

```
#include <stdio.h>
/* Déclaration et initialisation d'un tableau d'entiers */
char tab[4]={4,3,2,-5};
/* Déclaration d'un pointeur d'entier */
char *ptr_tab;

main()
{
    /* Initialisation de ptr_tab sur l'adresse de tab */
    ptr_tab=tab; /* Equivalent à ptr_tab=&tab[0] */

    /* Déplacement de 3 éléments de ptr_tab */
    ptr_tab=ptr_tab+3;

    /* Affichage du quatrième élément de tab */
    printf("tab[3]= %d\n",*ptr_tab);

    /* Déplacement de 2 éléments de ptr_tab */
    ptr_tab-=2; /* Equivalent à ptr_tab=ptr_tab-2; */

    /* Affichage du deuxième élément de tab */
    printf("tab[1]= %d\n",*ptr_tab);
}
```

Attention: **Le changement de valeur du pointeur doit-être contrôlé. Si le pointeur pointe sur une zone mémoire non définie, la machine peut se planter à tout moment. Pour l'exemple ci-dessus le pointeur `ptr_tab` doit être compris entre `&tab[0]` et `&tab[3]`.**

Exemple de synthèse:

```
#include <stdio.h>
/* Déclaration et initialisation d'une chaîne de caractères */
char mes[10]="Morgan";
/* Déclaration de trois pointeurs de caractère */
char *ptr_mes1,*ptr_mes2,*ptr_mes3;

main()
{
    /* Initialisation des pointeurs ptr_mes1,ptr_mes2,ptr_mes3 */
    ptr_mes1=mes; /* ptr_mes1 pointe sur mes[0] */
    ptr_mes2=ptr_mes1+2; /* ptr_mes2 pointe sur mes[2] */
    ptr_mes2=&mes[5]; /* ptr_mes3 pointe sur mes[5] */

    /* Affichage des caractères pointer par ptr_mes1,ptr_mes2,ptr_mes3 */
    printf("%c %c %c\n",*ptr_mes1,*ptr_mes2,*ptr_mes3);

    /* Changement des valeurs des pointeurs ptr_mes3 et ptr_mes2 */
    ptr_mes3++; /* ptr_mes3 pointe sur mes[6] */
    ptr_mes2 += 3; /* ptr_mes2 pointe sur mes[5] */

    /* Affichage de : ur, car la fonction printf attend
       pour le code %s une adresse et affiche la zone
       mémoire jusqu'au caractère nul '\0' */
    printf("%s\n",ptr_mes2);

    /* Met 'A' dans mes[5] pointé par ptr_mes2 */
    *ptr_mes2='A';
    /* Mes 'A'+1='B' dans mes[0] pointé par ptr_mes1 */
    *ptr_mes1=ptr_mes2+1;

    /* Affiche la chaîne de caractères mes */
    printf("%s \n",mes);
}
```

vous verrez

```
b n u
ur
BonjoAr
```

X.4 Les pointeurs et les structures de données

On peut définir des pointeurs de structures de données ou unions.

L'opérateur -> permet d'accéder aux différentes variables de la structure ou de l'union.

Syntaxe:
pointeur_d'objet -> identificateur_de_variable = valeur;
ou
valeur = pointeur_d'objet -> identificateur_de_variable;

Exemple:

```
#include <stdio.h>

/* Déclaration du type de structure: Cercle */
struct cercle
{
     int x,y; /* Position du centre du cercle*/
     int rayon; /* Rayon du cercle */
};

/* Déclaration d'un tableau de 2 structures de type cercle */
struct cercle tab_cercle[2];

/* Déclaration d'un pointeur de structure cercle */
struct cercle *ptr_cercle;

/* Déclaration de variable de boucle */
int i;

main()
{
     /* Initialisation du pointeur ptr_cercle
        sur le premier élément de la structure */
     ptr_cercle=&tab_cercle[0];

     /* Initialisation des variables du cercle 1 */
     tab_cercle[0].x=2;tab_cercle[0].y=2;tab_cercle[0].rayon=10;


     /* Ptr_cercle pointe sur le 2eme élément */
     ptr_cercle=&tab_cercle[1];

     /* Initialisation des variables du cercle 2 */
     ptr_cercle->x=1;ptr_cercle->y=8;ptr_cercle->rayon=40;

     /* Affichage des informations des cinq premiers cercles
        sans pointeur */
     for(i=0;i!=2;i++)
     {
            printf("Cercle %d\n",i);
            printf("Centre %d, %d\n",i,tab_cercle[i].x,tab_cercle[i].y);
            printf("Rayon %d\n\n",tab_cercle[i].rayon);
     }

     /* Affichage des informations des cinq premiers cercles
        avec un pointeur */
     /* ptr_cercle pointe le début du tableau tab_cercle */
     ptr_cercle=tab_cercle;
     for(i=0;i!=2;i++)
     {
            printf("Cercle %d\n",i);
            printf("Centre %d, %d\n",i,ptr_cercle->x,ptr_cercle->y);
            printf("Rayon %d\n\n",ptr_cercle->rayon);
            ptr_cercle++; /*Incrémentation pour pointer l'élément suivant*/
     }

}
```

X.5 Les pointeurs et l'allocation dynamique de mémoire.

X.5.1 Les fonctions de bases.

On peut réserver ou allouer pour une variable une partie de la mémoire en utilisant des fonctions de gestion de mémoire. Nous étudierons la fonction `malloc` et `free`.
Ces fonctions sont définies dans le fichier d'en-tête `stdlib.h`.

- **La fonction malloc.**

Elle permet de réserver une zone mémoire. Elle réserve une zone mémoire et renvoie un pointeur de type indéfini, il faudra mettre un `cast` devant pour spécifier le type de variable. Si la fonction renvoie 0 c'est qu'il n'a pas été possible d'allouer la zone mémoire.

```
Syntaxe:
pointeur_d'objet = (cast *)-> malloc(nb d'octets à réserver);
```

- **La fonction free.**

Elle permet de libérer une zone mémoire réservée avec `malloc`.

```
Syntaxe:
free(pointeur_d'objet);
```

Exemple:

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

/* Déclaration d'un pointeur d'entier */
int *ptr_int;

main()
{
     /* Réservation mémoire pour un entier */
     ptr_int = (int *) malloc(sizeof(int));

     /* Test si l'allocation a eu lieu */
     if (ptr_int==0) printf("Pas assez de mémoire\n");
     else
     {
            /* Initialisation de valeur pointé par ptr_int */
            *ptr_int=6;

            /* Affichage de la valeur pointé par ptr_int */
            printf("La valeur pointé par ptr_int est %d\n",*ptr_int);

            /* Libération de la zone mémoire pointé par ptr_int */
            free(ptr_int);
     }
}
```

Remarque: il existe d'autres fonctions pour gérer la mémoire, je vous renvoie à la documentation du compilateur utilisé.

X.5.2 L'allocation dynamique des tableaux, chaînes de caractères.

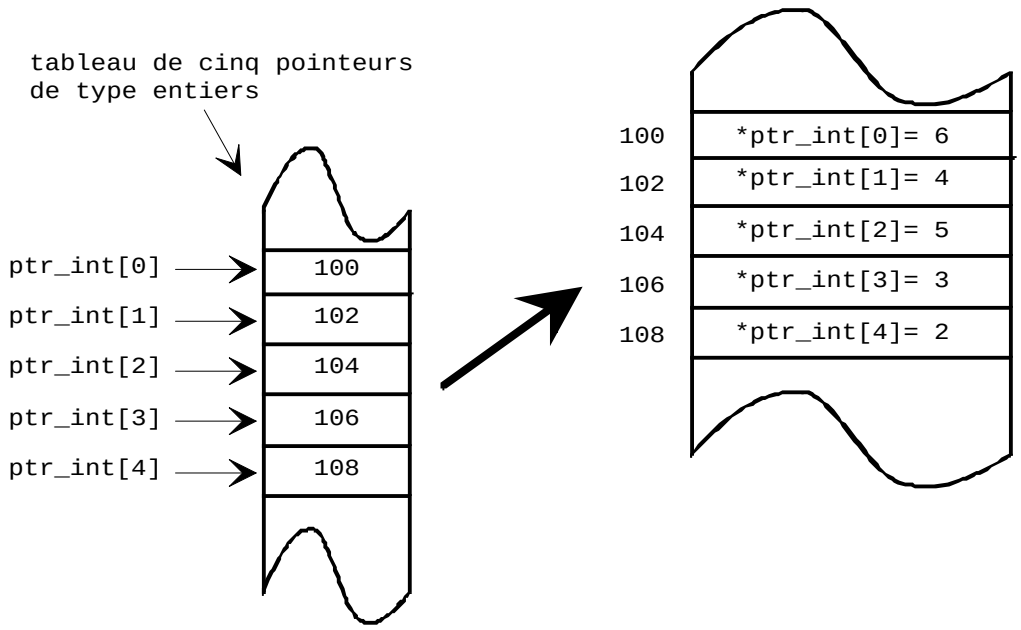
Accrochez vous, les notions qui suivent ne sont pas des plus triviales. Plutôt qu'un long discours plusieurs exemples.

· **tableau d'entiers.**

exemple:

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

/* Déclaration d'un tableau de 5 pointeurs d'entier */
int *ptr_int[5];

/* Déclaration d'une variable de boucle */
int i;

main()
{
     /* Réservation mémoire de taille:
        5 * la taille du type entier -> 5 *2 = 10 octets */
     for(i=0;i!=5;i++) ptr_int[i]=(int*)malloc(sizeof(int));

     /* Initialisation des valeurs du tableau ptr_int */
     *ptr_int[0]=6;
     *ptr_int[1]=4;
     *ptr_int[2]=5;
     *ptr_int[3]=3;
     *ptr_int[4]=2;

     /* Affichage des valeurs du tableau ptr_int */
     for(i=0;i!=5;i++)
     {
        printf("La valeur pointé par ptr_int[%d] est %d\n",i,*ptr_int[i]);
     }

     /* Libération de la zone mémoire  */
     for (i=0;i!=5;i++) free(ptr_int[i]);
}
```

Représentation mémoire:

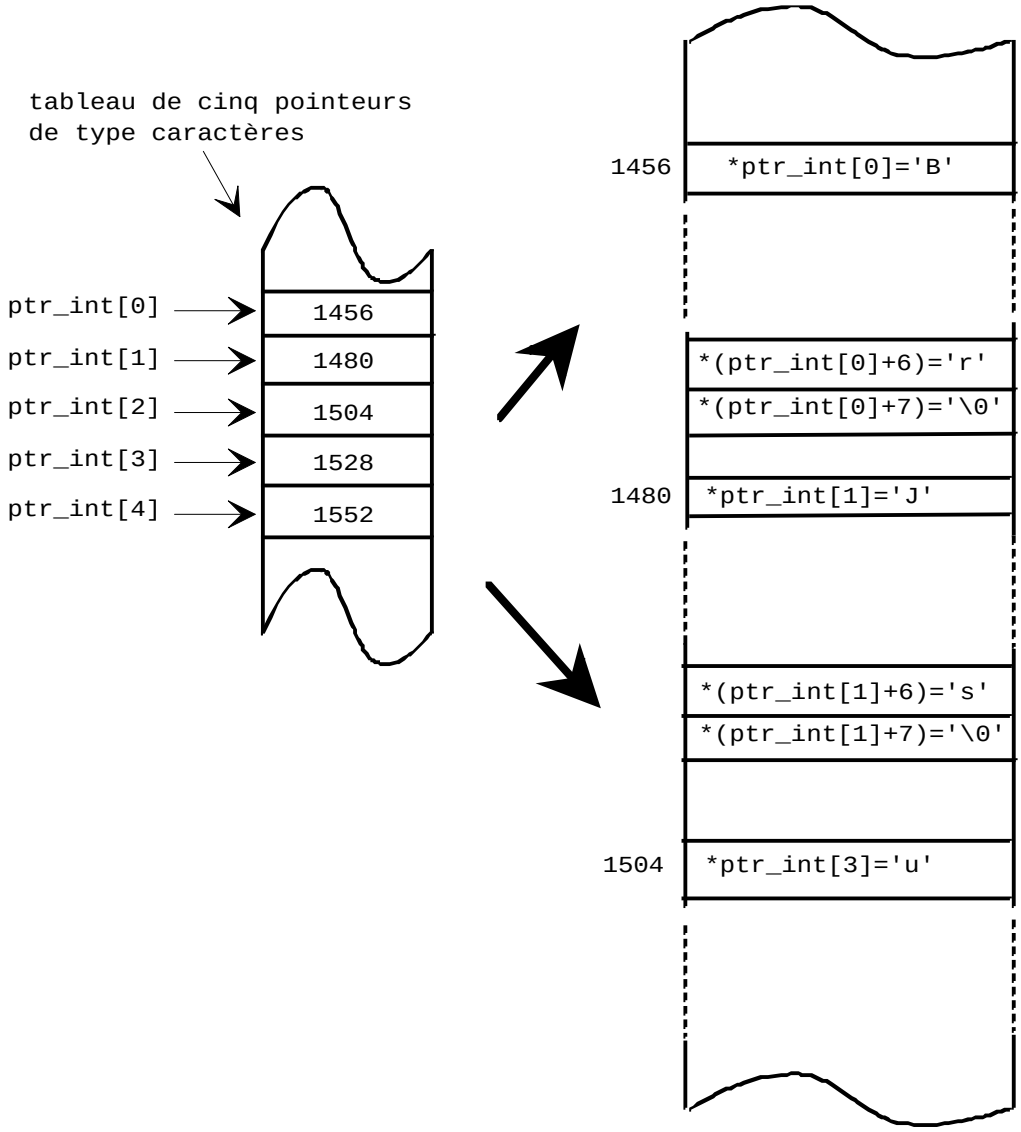


· **tableau de chaînes de caractères.**

exemple:

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

/* Déclaration d'un tableau de 5 chaînes */
char *ptr_char[5];

/* Déclaration d'une variable de boucle */
int i;

main()
{
    /* Réservation mémoire de taille:
       5 * taille de d'une chaîne de 15 caractères
       5 * 15 = 75 octets */
    for(i=0;i!=5;i++) ptr_char[i]=(char*)malloc(sizeof(char)*15);

    /* Initialisation des différentes chaînes du tableau */
    strcpy(ptr_char[0],"Bonjour");
    strcpy(ptr_char[1],"Je suis");
    strcpy(ptr_char[2],"un gentil");
    strcpy(ptr_char[3],"programmeur");
    strcpy(ptr_char[4],"A bientôt");

    /* Affichage des valeurs du tableau ptr_char */
    for(i=0;i!=5;i++)
    {
       printf("%s\n",ptr_char[i]);
    }

    /* Libération de la zone mémoire */
    for (i=0;i!=5;i++) free(ptr_char[i]);
}
```

Représentation mémoire:



XI Les fonctions.

Une fonction ou procédure ou encore sous programmes est une suite d'instructions élémentaires, mais vue du programme principal `main()`, elle représente une seule action. Elle est la base des langages structurés.

En effet, l'écriture de tous programmes commence par définir un algorithme qui décrit l'enchaînement de toutes les actions (fonctions). La réalisation d'un algorithme c'est la décomposition en une suite de fonctions simples pouvant réaliser une ou plusieurs fonctions beaucoup plus compliquées. Un algorithme doit être le plus clair possible. Une fonction est caractérisée par un appel et un retour.

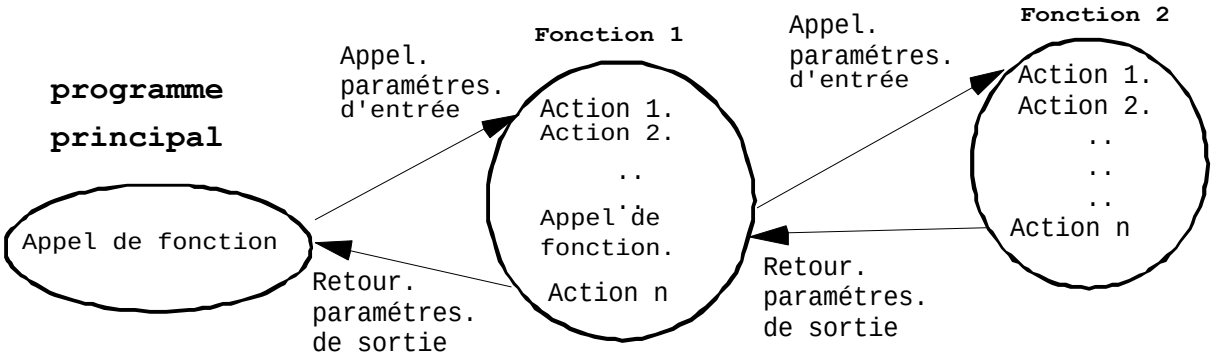
- L'appel est effectué par le programme principal `main()` ou une autre procédure.

- Le retour s'effectue après la dernière action de la fonction appelée et le programme continu juste après l'appel de la fonction.

Une fonction peut appelée une autre fonction. Le nom donné à une fonction doit-être représentatif de la fonction réalisée (exemple : perimetre_cercle pour une procédure calculant le périmètre d'un cercle).

Analogie: Pour un travail donné, l'homme décompose toujours en tâches élémentaires avant de réaliser la globalité.

Le programme principal `main()` et les fonctions ont besoin de se communiquer des valeurs. Lors d'un appel d'une procédure les valeurs passées sont les paramètres d'entrée et à la fin d'une fonction les paramètres renvoyés sont les paramètres de sortie.



XI.1 L'utilisation des fonctions.

La déclaration se fait en deux temps:

- **La déclaration des prototypes. Elle permet de définir au compilateur les paramètres d'entrée et de sortie afin de pouvoir vérifier lors d'appel de fonction l'ordre de passage des paramètres. La déclaration de prototypes se différencie de la déclaration de fonction par un point virgule `;`. Elle est située au début du programme, voir chapitre I structure d'un programme.**

```
Syntaxe:
<type>  <iden._de_sortie>  iden_fonc.(<type>  iden.1,..,  <type>
iden.n);
```

`iden._de_sortie`: identificateur de paramètre de sortie.
`iden_fonc`: identificateur de fonction.
`iden.1`: identificateur du paramètre `1` d'entrée.
`iden.n`: identificateur du paramètre `n` d'entrée.

- **La déclaration de fonction.** **Elle décrit l'enchaînement de toutes les instructions permettant de réaliser la fonction. Une variable déclarée dans une fonction est locale à celle-ci et elle n'a aucune existence en dehors de la fonction.**

```
Syntaxe:
<type> <iden._de_sortie> iden_fonc.(<type> iden.1,.., <type> iden.n)
{
      /* Déclaration de variable(s) locale(s) */
      <type> iden.2,...,iden.m;
            .
            .
            .
      /* renvoie dans le paramètre de sortie *
      return(valeur);
}
```

`iden._de_sortie`: **identificateur de paramètre de sortie.**
`iden_fonc`: **identificateur de fonction.**
`iden.1`: **identificateur du paramètre** `1` **d'entrée.**
`iden.n`: **identificateur du paramètre** `n` **d'entrée.**
`iden.2`: **identificateur** `2` **de variable locale..**
`iden.m`: **identificateur** `m` **de variable locale.**

- **L'appel de fonction: Il dirige le programme principal ou une autre fonction sur la fonction a exécuter en donnant les variables d'entrées et ou l'affectation de la variable de sortie.**

```
Syntaxe:
iden.var_de_sortie = iden_fonc.(iden.1_var,......,iden.n_var)

            ou
iden_fonc.(iden1.var,......,idenn.var)
```

`iden.var_de_sortie`: **identificateur de sortie.**
`iden_fonc`: **identificateur de fonction.**
`iden.1_var`: **identificateur du paramètre** `1` **d'entrée.**
`iden.n_var`: **identificateur du paramètre** `n` **d'entrée.**

XI.1.2 Les fonctions sans paramètre d'entrées et de sorties.

Elles servent en générale à structurer un programme. Au lieu d'écrire des milliers de lignes dans le programme principal `main()` **les unes derrières les autres, il faut mieux les rassembler dans des fonctions.**

Pour dire au compilateur qu'aucun paramètre est à passer on utilise le mot clé `void` **qui veut dire** `rien`**.**

Exemple sans faire appel aux fonctions:

```
#include <stdio.h>

/* Déclaration des variables globales */
int a,b,res;

/* Programme Principal */
main()
{
     printf("Donnez les valeurs de a et b ");
     scanf("%d %d",&a,&b);
     res=a+b;
     printf("\t\tJe traite\n");
     printf("L'addition de a=%d et de b=%d est égale à %d\n",a,b,res);
}
```

Même programme en utilisant les fonctions:

```
#include <stdio.h>

/* Déclaration des variables globales */
int a,b,res;

/* Déclaration des prototypes */
void saisie(void);
void traitement(void);
void affichage(void);

/* Déclaration des fonctions */
void saisie(void)
{
     printf("Donnez les valeurs de a et b ");
     scanf("%d %d",&a,&b);
}

void traitement(void)
{
     res=a+b;
     printf("\t\tJe traite\n");
}

void affichage(void)
{
     printf("L'addition de a=%d et de b=%d est égale à %d\n",a,b,res);
}

/* Programme Principal */
main()
{
     saisie();
     traitement();
     affichage();
}
```

Que faut-il constater ? : Le programme principal est très court, plus structuré mais aussi plus long.

XI.1.3 Les fonctions avec des paramètres d'entrée et ou un paramètre de sortie, passage de paramètres par valeur.

Elles permettent de communiquer des valeurs de variables globales en général aux variables locales de la fonction appelée. En retour la fonction appelée peut renvoyer une valeur de retour.

Exemple sans valeur de retour:

```
#include <stdio.h>

/* Déclaration des variables globales */
int a,b,res;

/* Déclaration des prototypes */
void saisie(void);
void traitement(int val1,int val2);
void affichage(void);

/* Déclaration des fonctions */
void saisie(void)
{
     printf("Donnez les valeurs de a et b ");
     scanf("%d %d",&a,&b);
}

void traitement(int val1,int val2)
{
     printf("\t\tJe traite\n");
     /* traitement */
     res=val1+val2;
}

void affichage(void)
{
     printf("L'addition de a=%d et de b=%d est égale à %d\n",a,b,res);
}

/* Programme Principal */
main()
{
     saisie();
     traitement(a,b);
     affichage();
}
```

Explications:

1. **La déclaration de la fonction `traitement` indique au compilateur le passage de deux valeurs de variables de type `int`, mais pas de valeur de retour car `void` est écrit devant l'identificateur de fonction.**
2. **Lors de l'appel de la fonction `traitement` dans le programme principal, il y d'abord la recopie des valeurs des variables globales a et b dans les variables locales val1 et val2, puis exécution de la procédure `traitement`.**

Exemple avec valeur de retour:

```
#include <stdio.h>

/* Déclaration des variables globales */
int a,b,res;

/* Déclaration des prototypes */
void saisie(void);
int traitement(int val1,int val2);
void affichage(void);

/* Déclaration des fonctions */
void saisie(void)
{
     printf("Donnez les valeurs de a et b ");
     scanf("%d %d",&a,&b);
}

int traitement(int val1,int val2)
{
     /* Déclaration de variable locale */
     int val3;
     printf("\t\tJe traite\n");
     /* traitement */
     val3=val1+val2;
     /* Renvoie de la valeur de val3 */
     return(val3);
}

void affichage(void)
{
     printf("L'addition de a=%d et de b=%d est égale à %d\n",a,b,res);
}

/* Programme Principal */
main()
{
     saisie();
     res=traitement(a,b);
     affichage();
}
```

Explications:

1. **La déclaration de la fonction `traitement` indique au compilateur le passage de deux valeurs de variables de type `int` avec une valeur de retour de type `int`, écrit devant l'identificateur de fonction.**
2. **Lors de l'appel de la fonction `traitement` dans le programme principal, il y a d'abord la recopie des valeurs des variables globales a et b dans les variables locales val1 et val2, puis exécution de la procédure `traitement`.**
3. **Au retour de la fonction `traitement` dans le programme principal , la valeur de la variable locale val3 sera recopiée dans la variable globale `res` par l'instruction `return(val3)`.**

XI.1.4 Les fonctions avec des paramètres d'entrée et un ou plusieurs paramètres de sortie, passage de paramètres par adresse.

C'est toute la puissance du C de pouvoir passer à une fonction les valeurs et ou les adresses de variables. Ainsi toute modification de valeur d'une variable globale dans une fonction sera répertoriée dans le programme principal, ce qui permet de pouvoir passer des tableaux ou structures de données ou encore d'avoir plusieurs paramètres de sorties.

Exemple avec passage de paramètres par valeur:

```
#include <stdio.h>

/* Déclaration des variables globales */
int a,b;

/* Déclaration des prototypes */
void fonction(int val1,int val2);

void fonction(int val1,int val2)
{
     val1+=4; /* Incrémente val1 de 4 */
     val2+-2; /* Décrémente val2 de 2 */
     printf("val1=%d et val2=%d\n",val1,val2);
}

main()
{
     /* Initialisation de a et b */
     a=4;b=6;

     /* Premier affichage de a et b */
     printf("a=%d et b=%d\n",a,b);

     /* Appel de la fonction */
     fonction(a,b);

     /* Deuxième affichage de a et b */
     printf("a=%d et b=%d\n",a,b);
}
```

Explications:

1) **Affichage des valeurs de `a` et `b`.**
2) **Appel de la fonction `fonction` et exécution de celle-ci; Affichage de `val1` et `val2`;**
3) **Affichage des valeurs de `a` et `b`.**

On peut remarquer que les valeurs de `a` et `b` n'ont pas changées au deuxième affichage. C'est normal car on a fait un passage par valeur. c'est à a dire que l'on a recopié les valeurs de `a` et de `b` dans `val1` et `val2`.

Même exemple avec passage de paramètres par adresse:

```
#include <stdio.h>

/* Déclaration des variables globales */
int a,b;

/* Déclaration des prototypes */
void fonction(int *val1,int *val2);

void fonction(int *val1,int *val2)
{
     *val1+=4; /* Incrémente val1 de 4 */
     *val2+-2; /* Décrémente val2 de 2 */
     printf("val1=%d et val2=%d\n",*val1,*val2);
}

main()
{
     /* Initialisation de a et b */
     a=4;b=6;

     /* Premier affichage de a et b */
     printf("a=%d et b=%d\n",a,b);

     /* Appel de la fonction */
     fonction(&a,&b);

     /* Deuxième affichage de a et b */
     printf("a=%d et b=%d\n",a,b);
}
```

Explications:

1) **Affichage des valeurs de `a` et `b`.**
2. **Lors de l'appel de la fonction `fonction` par le programme principal, il y d'abord la recopie des adresses de a et b (&a,&b) dans les pointeurs val1 et val2, puis exécution de la procédure `fonction` et Affichage de `val1` et `val2`;**
3) **Affichage des valeurs de `a` et `b`.**

On peut remarquer que les valeurs de `a` et `b` ont changées au deuxième affichage. C'est normal car on a fait un passage par adresse. C'est à dire que l'on a recopié les adresses de `a` et de `b` dans `val1` et `val2` qui sont maintenant des pointeurs de type entier, toutes modifications de valeurs des objets pointés par `val1` et `val2` dans `fonction` seront recopiées dans `a` et `b`.

exemple de passage par adresse de tableau:

fonction calculant la longueur du chaîne de caractères:

```
#include <stdio.h>

/* Déclaration et initialisation d'une chaîne de caractères */
char message[10]="René";

/* Déclaration de prototype */
int long_ch(char *ptr_chaine);

int long_ch(char *ptr_chaine)
/* Description: Donne le nombre de caractères d'une chaîne */
/* ptr_chaine : Pointeur de type char, pointant sur la chaîne */
/* renvoie le nombre de caractère dans une entier */
{
     /* Variable locale compteur */
     int nb_car;
     /* Initialisation du compteur */
     nb_car=0;
     /* Faire tant que le caractère est différent de '\0' */
     while(*ptr_chaine!='\0')
     {
            nb_car++; /* Incrémente le compteur de caractères */
            ptr_chaine++; /* Incrémente le pointeur de chaîne */
     }
     return(nb_car); /* Renvoie le nb de caractères */
}

main()
{
     printf("La longueur de %s est %d\n",message,long_ch(message));
}
```

Explications:

Une seule instruction dans le programme principal qui affiche la chaîne de caractères et sa longueur.

1) **Elle appelle la fonction `long_ch` en recopiant l'adresse où est stocké la chaîne dans le pointeur `ptr_chaine`. On peut remarquer que devant `message` il n'y a pas l'opérateur `&` car `message` est une adresse.**
2) **Elle exécute la fonction et la fonction renvoie le nombre de caractères dans un entier avec l'instruction `return(nb_car)` .**
3) **Elle affiche le nombre de caractères, c'est le deuxième `%d` du format.**

Il existe beaucoup de fonctions dans les librairies standard du C. Il y en une qui donne le nombre de caractères d'une chaîne elle s'appelle `strlen`, voir chapitre suivant.

XI.2 Les fonctions standards du C.

Tout compilateur C dispose d'un ensemble de bibliothèques de fonctions. Il y à plusieurs bibliothèques qui sont dites standards car on les retrouve sur la plupart des compilateurs.

XI.2.1 La bibliothèque d'entrée sortie.

Les déclarations des fonctions sont dans le fichier d'en-tête `<stdio.h>`. Elles s'occupent de la gestion des entrées et des sorties que se soit à partir de fichiers ou du clavier.

Les fonctions standards du clavier et de l'écran sont:

Fonctions	Description
`getchar()`	Saisie d'un caractère.
`putchar()`	Affichage d'un caractère.
`printf()`	Affichage formaté.
`scanf()`	Saisie formaté.
`gets()`	Saisie d'une ligne.
`puts()`	Affichage d'une ligne.

Remarque: **Ces fonctions ont été vues dans le chapitre IV.**

Pour les fichiers voir chapitre suivant.

XI.2.2 Les manipulations de caractères.

Les déclarations des fonctions sont dans le fichier d'en-tête `<ctype.h>`. Elles s'occupent de traiter les caractères en autre de définir à quel ensemble ils appartiennent: exemple majuscule ou minuscule.

Fonctions	Description
`isascii()`	Test ASCII
`tiascii()`	Conversion numérique ASCII
`isalnum()`	Test lettre ou chiffre
`isalpha()`	Test Lettre majuscule ou minuscule
`iscntrl()`	Test caractère de contrôle
`isdigit()`	Test chiffre décimal
`isxdigit()`	Test chiffre héxadécimal
`isprint()`	Test caractère imprimable
`ispunct()`	Test ponctuation
`isspace()`	Test séparateur de mots
`isupper()`	Test lettre majuscule
`islower()`	Test lettre minuscule
`toupper()`	Conversion en majuscule
`tolower()`	Conversion en minuscule

Les fonctions `is....` renvoient `0` si `FAUX` ou différent de `0` si `VRAIE`.

XI.2.3 Les manipulations de chaînes de caractères.

Les déclarations des fonctions sont dans le fichier d'en-tête `<string.h>`. Elles s'occupent de traiter les chaînes de caractères.

Fonctions	Description
`strcat()`	Concaténation de deux chaînes.
`strcmp()`	Comparaison de deux chaînes.
`strcpy()`	Copie de chaîne.
`strlen()`	Longueur de chaîne.
`strncat`	Concaténation de caractères
`strncmp()`	Comparaison sur une longueur.
`strncpy()`	Copie de n caractères.
`sscanf()`	Lecture formatée dans une chaîne.
`sprintf()`	Copie formatée dans une chaîne.

XI.2.4 Les fonctions mathématiques.

Les déclarations des fonctions sont dans le fichier d'en-tête `<math.h>`. Elles s'occupent de traiter les nombres.

Fonctions	Description
`abs()`	Valeur absolue.
`acos()`	Arc Cosï nus.
`asin()`	Arc Sinus.
`atan()`	Arc Tangente.
`atan2()`	Arc Tangente X/Y.
`ceil()`	Arrondit par excès.
`cos()`	Cosï nus.
`cosh()`	Cosï nus Hyperbolique.
`exp()`	Exponentielle.
`fabs()`	Valeur absolue d'un nombre réel.
`floor()`	Arrondit par défaut.
`fmod()`	Modulo réel.
`frexp()`	Décompose un réel.
`ldexp()`	Multiplie un réel par 2^x.
`log()`	Logarithme népérien.
`log10()`	Logarithme décimal.
`modf()`	Décomposition d'un réel.
`pow()`	Puissance.
`sin()`	Sinus.
`sin()`	Sinus Hyperbolique.
`sqrt()`	Racine Carrée.
`tan()`	Tangente.
`tanh()`	Tangente Hyperbolique.

XI.2.5 La fonction main() et les paramètres de ligne de commande.

C'est la fonction de base du C. Elle est appelée la première lors de l'éxcution d'un programme c'est le programme principal.

Une fois compilé un programme peut être lancé du système d'exploitation. L'utilisateur peut avoir besoin de passer des paramètres au programme dit paramètres de ligne de commande.

Syntaxe:
```
main(int argc, char *argv[],char *env[]);
```

`int argc`: Nombre de paramètres passés.

`char *argv[]`: Tableau de chaînes de caractères contenant les paramètres d'entrées.

`char *env[]`: Tableau de chaînes de caractères contenant les variables d'environnement.

Exemple:

```
#include <stdio.h>

int i;

main(int argc,char *argv[],char *env[])
{
     /* Affichage du nombre de paramètres */
     printf("Nombre de paramètre = %d\n\n",argc);

     /* Affichage des paramètres */
     for (i=0;i!=argc;i++) printf("Le paramètre %d est %s\n",i,argv[i]);
     putchar('\n');
     i=0;
     /* Affichage des variables d'environnement */
     while(env[i]!=NULL)
     printf("La variable d'environnement %d est %s\n",i,env[i++]);
}
```

XII Les fichiers.

La gestion de fichiers permet d'acquérir des informations généralement des unités de stockage de la machine afin de les traiter, de les afficher, de les récrire dans d'autres fichiers.

En C il y a cinq fichiers standards toujours ouvert:

`stdin`	**L'entrée standard (clavier)**
`stdout`	**L'entrée standard (écran)**
`stderr`	**La sortie d'erreurs (écran)**
`stdprn`	**La sortie imprimante (imprimante)**
`stdaux`	**La sortie série (Port série).**

Il existe deux type de fichiers:

1) **Le fichier binaire où les informations sont mises les unes à la suite des autres et il guère possible de relire ce type de fichier.**
2) **Le fichier texte où chaque information est sur une ligne texte et séparée par retour à la ligne, il est facile de les relire avec n'importe quel éditeur de textes.**

XII.1 La gestion de haut niveau.

Elle utilise des buffers de données. Elle utilise une structure de donnée `FILE` utilsant un pointeur `*FILE` est défini dans le fichier d'en tête `sdtio.h`.

Les fonctions de manipulation de fichiers de base.

Fonctions	Description
`fopen()`	**Ouverture de fichier.**
`fclose()`	**Fermeture de fichier.**
`getc()`	**Lecture d'un caractère.**
`fgetc()`	**Lecture d'un caractère.**
`fgets()`	**Lecture d'une ligne ('\n')**
`fgetw()`	**Lecture d'un mot**
`fread()`	**Lecture d'un groupe d'octets**
`fscanf()`	**Lecture formatée.**
`putc()`	**Ecriture d'un caractère.**
`fputc()`	**Ecriture d'un caractère.**
`fputs()`	**Ecriture d'une chaîne.**
`fputw()`	**Ecriture d'un mot.**
`fwrite()`	**Ecriture d'un groupe d'octets.**
`fprintf()`	**Ecriture formatée.**
`feof()`	**Fin de fichier.**
`ferror()`	**Gestion des erreurs.**
`ftell()`	**Position dans un fichier.**
`fseek()`	**Positionnement dans le buffer.**
`rewind()`	**Positionnement au début du fichier.**

Exemple: Ecriture de messages sur l'imprimante.

```
#include <stdio.h>

main()
{
     /* Mettez en marche votre imprimante */
     /* Ecriture formaté sur l'imprimante */
     fprintf(stdprn,"Bonjour\n");
     fprintf(stdprn,"\t\t Essai d'impression");
}
```

Que remarquez vous ?

la fonction `printf` s'est transformée en `fprintf` pour `file_printf`. C'est exactement la même chose que `printf` sauf qu'il faut indiquer vers quel fichier il faut faire la sortie, ici c'est stdprn mais on aurait pu mettre stdaux.

- **La fonction `fopen()`.**

```
Syntaxe:
FILE *fopen(char *Nom_de_fichier, char *type);
```

Paramètres d'entrée:

- char *Nom_de_fichier:C'est une chaîne de caractères indiquant le nom du fichier à ouvrir.

- char *type: C'est une chaîne de caractères définissant l'opération qui va être effectuée.

Type	Opération
`"r"`	Lecture seule.
`"w"`	Ecriture.
`"a"`	En ajout écriture à la fin du fichier.
`"r+"`	Ouverture en lecture et en écriture.
`"w+"`	Création.
`"a+"`	En ajout en lecture et en écriture à la fin du fichier.

Il faut en plus dire si un fichier texte `"t"` ou binaire `"b"`.
Ce qui donne par exemple pour un fichier binaire ouvert en lecture seule le type "rb".

Elle renvoie:

- Un pointeur de type `FILE` si l'ouverture a été possible.
- Un `0` ou la constante définie dans `stdio.h` `NULL` si n'a été possible d'ouvrir le fichier.

- **Structure d'un programme utilisant un fichier.**

1) Ouverture du fichier fopen().
2) Lecture ou écriture des informations du fichier.
3) fermeture.

Exemple: Lecture d'un fichier texte. Si vous êtes sur une machine utilisant le système d'exploitation MSDOS alors je vous propose de relire le fichier `autoexec.bat` et de l'afficher à l'écran.

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

/* Déclaration d'un pointeur sur un fichier */
FILE *ptr_fichier;

/* Déclaration de chaîne de réception des lignes du fichier texte */
char chaine[80];

main()
{
    /* Ouverture d'un fichier en lecture seule "r"
       et de type texte "t" ayant pour nom autoexec.bat */
    ptr_fichier=fopen("autoexec.bat","rt");
    /* fopen renvoie NULL si impossibilité d'ouvrir le fichier */
    if (ptr_fichier==NULL)
    {
           printf("Impossible d'ouvrir le fichier\n");
    }
    else
    {
           /* boucler tant que la fin du fichier n'est pas atteinte
              feof(ptr_fichier) renvoie une valeur NULL tant que la
              fin n'est pas atteinte.
              feof : File end of file
              feof(ptr_fichier) renvoie une valeur différente de NULL
              si la fin du fichier est atteinte */
           while(feof(ptr_fichier)==NULL)
           {
                  /* Lecture d'une ligne dans le fichier
                     fgets va recopier la ligne de texte
                     dans chaine à raison de 80 caractères
                     du fichier pointé par ptr_fichier */
                     fgets(chaine,80,ptr_fichier);

                  /* Suppresion d'un retour chariot à la fin
                     la ligne */
                     chaine[strlen(chaine)-1]='\0';

                  /* Affichage de chaîne reçu */
                     puts(chaine);
           }
           /* Fermeture du fichier */
           fclose(ptr_fichier);
    }
}
```

Exemple: **Ecriture dans un fichier texte.**

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

/* Déclaration d'un pointeur sur un fichier */
FILE *ptr_fichier;

/* Déclaration de chaîne de réception des lignes du fichier texte */
char chaine[80];

main()
{
    /* Devant d'ouverture d'un fichier en écriture seule "w"
       et de type texte "t" ayant pour nom texte.txt */
    ptr_fichier=fopen("texte.txt","wt");
    /* fopen renvoie NULL si impossibilité d'ouvrir le fichier */
    if (ptr_fichier==NULL)
    {
           printf("Impossible d'ouvrir le fichier\n");
    }
    else
    {
           /* Ecriture de texte dans le fichier */
           fprintf(ptr_fichier,"ce fichier est un fichier texte\n");
           fprintf(ptr_fichier,"c est la deuxieme ligne de texte\n");
           fprintf(ptr_fichier,"c est la fin\n");

           /* Fermeture du fichier */
           fclose(ptr_fichier);
    }
}
```

Exemple: **Lecture du fichier texte `texte.txt`, passage en majuscules des lignes de texte et écriture du fichier `texte.txt`.**

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

/* Définition d'un type de chaine de 80 caractères */
typedef char str80[80];
/* Déclaration d'un pointeur sur un fichier */
FILE *ptr_fichier;
/* Déclaration d'un tableau de 100 chaînes de caractères
   pour recevoir les lignes du fichier texte */
str80 tab_chaine[100];
/* Déclaration d'une variable donnant
   le nombre de ligne dans le fichier */
int long_fich;

/* Déclaration des prototypes de fonctions */
int lit_fichier(char *nom_fich);
void minus_maju(void);
int ecrit_fichier(char *nom_fich);
void affichage(void);

int lit_fichier(char *nom_fich)
{
    int i; /* variable de boucle */
    ptr_fichier=fopen(nom_fich,"rt");
    if (ptr_fichier==NULL)
    {
           /* Si impossible de lire le fichier
           alors renvoyer NULL */
           return(NULL);
    }
    else
    {
           i=0;
           while(feof(ptr_fichier)==NULL)
           {
                  fgets(tab_chaine[i],80,ptr_fichier);
                  tab_chaine[i][strlen(tab_chaine[i])-1]='\0';
                  i++;
           }
           long_fich=i-1;
           fclose(ptr_fichier);
           return(-1);
    }
}

/* Fonction effectuant le passage de minuscule en majuscule */
void minus_maju(void)
{
    int i,j;
    char temp;
    /* Balayage de toutes les lignes */
    for (i=0;i!=long_fich;i++)
    {
           j=0;
           /* Balayage de tous les caractères
              de la ligne tab_chaine[i]
              jusqu'à '\0' */
           while(tab_chaine[i][j]!='\0')
           {
                  /* Conversion de minuscule en majuscule */
                  temp=toupper(tab_chaine[i][j]);
                  tab_chaine[i][j]=temp;
                  j++;
           }
    }
}
```

```
int ecrit_fichier(char *nom_fich)
{
     int i;
     ptr_fichier=fopen(nom_fich,"wt");
     if (ptr_fichier==NULL)
     {
            return(NULL);
     }
     else
     {
            for(i=0;i!=long_fich;i++)
            {
                   fprintf(ptr_fichier,"%s\n",tab_chaine[i]);
            }
            fclose(ptr_fichier);
            return(-1);
     }
}

void affichage(void)
{
     int i;
     for(i=0;i!=long_fich;i++) puts(tab_chaine[i]);
}


main()
{
     if (lit_fichier("texte.txt")==NULL)
     {
            printf("Impossible d'ouvrir le fichier\n");
     }
     else
     {
            minus_maju();
            affichage();
            if (ecrit_fichier("texte.txt")==NULL)
            {
                   printf("Impossible d'écrire le fichier\n");
            }
     }
}
```

Exemple: **Ecriture d'informations numériques dans un fichier texte `nombre.txt`.**

```
#include <stdio.h>

/* Déclaration d'un pointeur sur un fichier */
FILE *ptr_fichier;

/* Déclaration d'un réel et un entier */
int entier;
float reel;

main()
{
     /* Initialisation des variables */
     entier=4560;
     reel=3.4510;

     ptr_fichier=fopen("nombre.txt","wt");
     if (ptr_fichier==NULL)
     {
            printf("Impossible d'ouvrir le fichier\n");
     }
     else
     {
            /* Ecriture d'un entier dans le fichier */
            fprintf(ptr_fichier,"%d\n",entier);
            /* Ecriture d'un réel dans le fichier */
            fprintf(ptr_fichier,"%f\n",reel);
     }
     fclose(ptr_fichier);
}
```

Exemple: **Lecture d'informations numériques dans un fichier texte `nombre.txt`.**

```
#include <stdio.h>

/* Déclaration d'un pointeur sur un fichier */
FILE *ptr_fichier;

/* Déclaration d'un réel et un entier */
int entier;
float reel;

main()
{
     ptr_fichier=fopen("nombre.txt","rt");
     if (ptr_fichier==NULL)
     {
            printf("Impossible d'ouvrir le fichier\n");
     }
     else
     {
            /* Lecture d'un entier dans le fichier */
            fscanf(ptr_fichier,"%d",&entier);
            /* Lecture d'un réel dans le fichier */
            fscanf(ptr_fichier,"%f",&reel);
     }
     fclose(ptr_fichier);

     /* Affichage des valeurs lues */
     printf("L'entier a pour valeur %d\n",entier);
     printf("Le réel a pour valeur %f\n",reel);
}
```

XII.2 La gestion de bas niveau.

Elle n'utilise pas des buffers de données. Elle utilise le descripteur du fichier qui est défini dans le fichier d'en tête `fcntl.h` et `io.h`.

Fonctions	Description
`open()`	Ouverture de fichier.
`close()`	Fermeture de fichier.
`read()`	Lecture.
`write()`	Ecriture.
`lseek()`	positionnement.

A la différence des fonctions de hauts niveaux elles utilisent un numéro (`handle`) pour chaque fichier ouvert. Il existe des numéros pré définis.

numéro	Fichier
0	Entrée standard.
1	Sortie standard.
3	Sortie d'erreur standard.

- **La fonction `fopen()`.**

```
Syntaxe:
int open(char *Nom_de_fichier,int mode);
```

Paramètres d'entrée:

- char *Nom_de_fichier: C'est une chaîne de caractères indiquant le nom du fichier à ouvrir.

- int mode: C'est un entier qui indique le type d'opération.

Type	Opération
`O_RDONLY`	Lecture seule.
`0_WRONLY`	Ecriture seule.
`O_RDWR`	Lecture ou Ecriture.
`O_CREAT`	Création et Ouverture
`O_TRUNC`	Ouverture avec troncature.
`O_EXCL`	Ouverture exclusive.
`O_APPEND`	Ajout à la fin du fichier.
`O_TEXT`	Fichier texte
`O_BINARY`	Fichier binaire

Elle renvoie:

- Un numéro de fichier.

- Un `-1` s'il n'a pas été possible d'ouvrir le fichier.

Exemple: **Lecture du fichier texte.txt.**

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

/* Déclaration d'un numéro de fichier */
int handle;

/* Déclaration de chaîne de réception des lignes du fichier texte */
char chaine[80];

main()
{
     /* Ouverture du fichier en Lecture seule et de type texte
        on met | pour séparer les informations */
     handle=open("texte.txt",O_RDONLY | O_TEXT);
     if (handle==-1)
     {
            printf("Impossible d'ouvrir le fichier\n");
     }
     else
     {
            /* Tant que la fin du fichier n'est pas atteinte */
            /* eof: end of file
               renvoie 0 si la fin du fichier n'est pas atteinte
               renvoie -1 si elle est atteinte */
            while(eof(handle)==NULL)
            {
                   /* Lecture de texte dans le fichier */
                   read(handle,chaine,80);
                   /* Affichage de la ligne lu */
                   puts(chaine);
            };
            /* Fermeture du fichier */
            close(handle);
     }
}
```

Remarque: Le nombre de fonction de bas niveau est inférieur au fonction de haut niveau, pour l'écriture ou la lecture de nombres dans un fichier il faudra utiliser les adresses.

XIII Le pré processeur.

Il s'occupe de toutes les directives de compilation commençant par le symbole *#*.

· **La directive `include`.**

```
Syntaxe:
#include "nom_de_fichier" /* Répertoire courant */
          ou
#include <nom_de_fichier" /* Répertoire include */
```

Elle permet lors de la compilation d'ajouter des fichiers textes qui contiennent généralement des définitions de constantes ou de fonctions se sont les fichiers `header *.h`.

· **La directive `define`.**

1) Définition de symbole.

```
Syntaxe:
#define iden_de_constante  valeur
```

Elle permet de définir des constantes symboliques. C'est à dire lors de la compilation les symboles seront remplacés par leur valeur.

2) Définition de macro-instruction.

```
Syntaxe:
#define iden_de_macro(arguments)instructions
```

Elle permet de définir des constantes symboliques. C'est à dire lors de la compilation les symboles seront remplacés par leur valeur.

Exemple:

```
#include <stdio.h>

#define max(a,b) (a>b) ? a : b
#define carre(x) x*x
#define swap(type,x,y) {type t;t=x;x=y;y=t;}

main()
{
     int a,b;
     a=5;b=-3;
     printf("Le maximum de a=%d et b=%d est %d\n",a,b,max(a,b));
     swap(int,a,b);
     printf("a=%d et b=%d\n",a,b);
     printf("Le carré de %d est %d\n",a,carre(a));
}
```

· **Les directives `#if #else #endif`.**

```
Syntaxe:
#if condition
{
      .
      .
}
#endif
           ou
#if condition
{
      .
      .
}
#else
{
      .
      .
}
#endif
```

Elles permettent de sélectionner lors d'une compilation des blocs instructions

Exemple:

```
#include <stdio.h>

#define FAUX 0
#define VRAI -1
#define DEBUG VRAI

main()
{
#if (DEBUG==VRAI)
     printf("Le programme est en phase de développement\n");
#else
     printf("Le programme est en phase de fonctionnement\n");
#endif
}
```

XIV Le mécanisme de compilation.

Les différentes phases de développemnt d'un programme sont:
1) Edition du fichier source.
2) Compilation.
3) Edtion de lien (liaison avec les bibliothèques).
4) Mise au point (Débogueur).