Introduction au

Langage C

Philippe Letenneur Philippe Lecardonnel

Lycée Julliot de la Morandière

G r a n v i l l e

Dernière mise à jour : 11/2004

I Introduction.

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

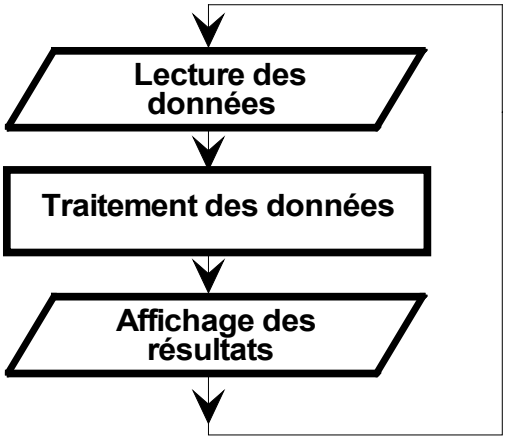
Les avantages du **C** sont nombreux:

- **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 proche de la machine en pouvant accéder aux adresses des variables.
- **Très rapide:** Aucun contrôle de débordement n'est effectué, ce qui apporte une plus grande vitesse.

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 prototype se termine toujours
                                            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

void 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`**) alors 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 ****/*** et peut éventuellement être écrit sur plusieurs lignes.

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

- Le langage C est sensible à la « casse » : les mots du langage doivent être écrits en minuscules ; les noms de variables, constantes, fonctions doivent toujours être écrits de la même façon : Ma_Fonction ≠ ma_fonction ≠ MA_FONCTION.

- Les seuls caractères autorisés pour les noms de variables, constantes… sont :
  - lettres NON accentuées ;
  - chiffres (sauf au début du nom) ;
  - caractères souligné « _ ».

III Les Variables et les Constantes.

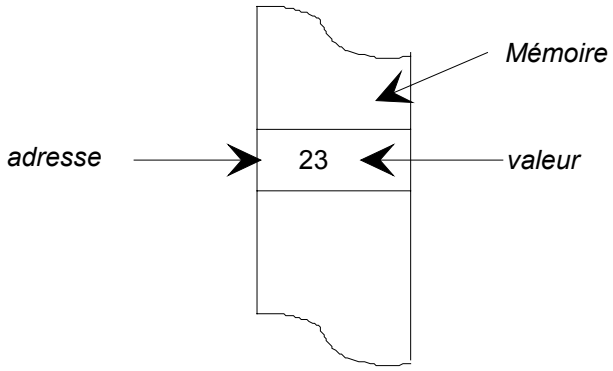
Définition d'une constante: Elle **ne change jamais de valeur** pendant l'exécution d'un programme. Elle est généralement stockée dans la **mémoire morte** d'une machine.

Définition d'une variable: Elle **peut changer de valeur** pendant l'exécution d'un programme. Elle est généralement stockée dans la **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 attribue à la variable ou à la constante.
- **L'adresse**: C'est l'emplacement où est stocké la valeur de la variable ou de la constante. L'adresse 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**.

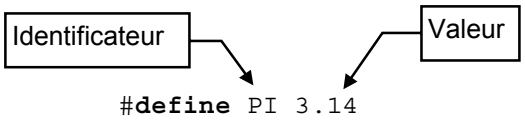
  Exemple $12{,}344.10^{-5}$.

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:

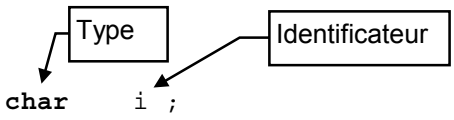


III.2 Les variables.

Les variables sont définies par ***le type et l'identificateur***.

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

Exemple:



- **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 de la variable, alors la variable devient non signée et cela permet d'étendre la plage de valeurs.

- **L'identificateur:** C'est le ***nom*** affecté à la variable. Le nombre de caractères peut être limité, cela dépend du compilateur utilisé. L'identificateur d'une variable doit toujours être écrit de la même façon : Mon_Identificateur ≠ mon_identificateur ≠ MON_IDENTIFICATEUR.

  Les seuls caractères autorisés pour les noms de variables sont :
  - lettres NON accentuées ;
  - chiffres (sauf au début du nom) ;
  - caractères souligné « _ ».

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 +32767	0 à +65535
Entier Court → **Short**	16	-32768 à +32767	0 à +65535
Entier long → **Long**	32	-2147483648 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;

void 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;

void main()
{
     /* Décimal */
     i=21;
     /* Octal */
     i=025;
     /* Hexadécimal */
     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;`
`void main()`
`{`
`  /* Initialisation */`
`  i=15;`
`}`
	`/* Déclaration et Initialisation */`
	`int i=15;`
	
	`void main()`
	`{`
	`  .`
	`  .`
	`}`

Remarque: La méthode d'initialisation après déclaration est meilleure car elle permet de bien séparer 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];
```

<classe> est facultatif

- **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] */

void main()
{
          .
          .
}
```

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

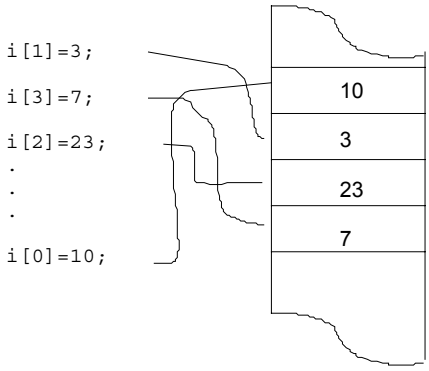
Initialisation d'un tableau:

```
/* 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] */

void 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] */

void main()
{
            .
            .
            .
}
```

Représentation mémoire:



- **Tableau à plusieurs dimensions.**

Exemple:

```
/* Déclaration d'un tableau de 6 éléments à deux dimensions */
int k[2][3];
/* Le premier élément est k[0][0] */
/* Le dernier élément est k[1][2] */

void main()
{
            .
            .
            .
}
```

Initialisation:

```
/* Déclaration d'un tableau de 6 éléments à deux dimensions */
int k[2][3];
/* Le premier élément est k[0][0] */
/* Le dernier élément est k[1][2] */

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

ou encore.

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

void main()
{
             .
             .
             .
}
```

ou encore.

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

void 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];

<classe> est facultatif

Exemple:

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

On a défini 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**'.

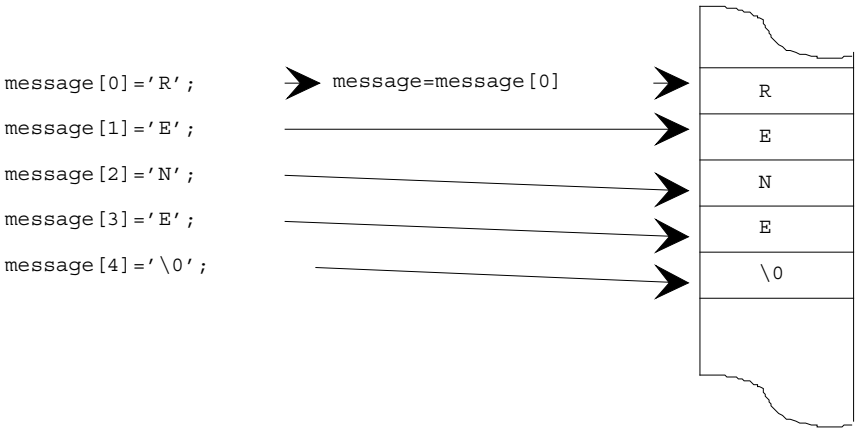
Initialisation de chaîne:

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

void 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";

void main()
{
            .
            .
            .
}
```

Représentation mémoire:



Il n'est **pas** possible d'initialiser une chaîne de cette façon.

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

void 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 caractères** (**strcpy**), celle-ci recopie un à un tous les caractères de **"RENE"** à partir de "**message".**

Remarque: **strcpy** est déclaré 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];

void 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 standards du C ».

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 n'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, quelques exemples:

```
#include <stdio.h>

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

void 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** est accessible de n'importe où.

```
#include <stdio.h>

void 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** n'est accessible que dans le bloc **main**.

```
#include <stdio.h>

void 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 */
}
```

Si 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.

Autre exemple:

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

void 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 (5 dans l'exemple).
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 (1 dans l'exemple).
6)Fin de bloc le programme supprime la réservation mémoire de la variable locale.
7)Affichage de la variable globale i (5 dans l'exemple).

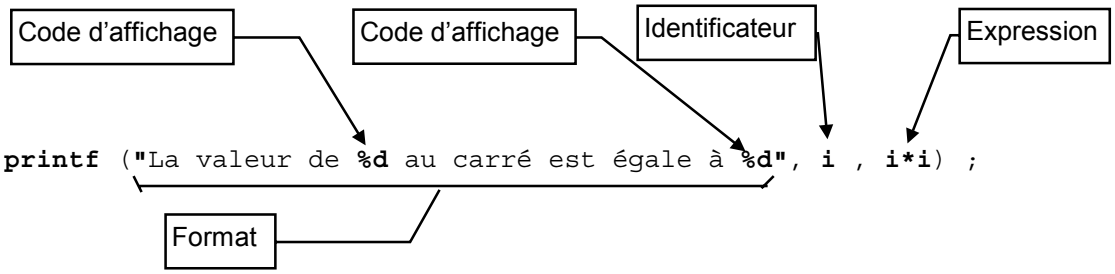
IV Les fonctions d'affichage et de saisie.

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 affiché les valeurs des variables. Il est composé de texte et de codes d'affichage suivant le type de variable.



Codes d'affichage:

Type	*Format*
Entier décimal	%d
Entier Octal	%o
Entier Hexadécimal (minuscules)	%x
Entier Hexadécimal (majuscules)	%X
Entier Non Signé	%u
Caractère	%c
Chaîne de caractères	%s
Flottant (réel)	%f
Scientifique	%e
Long Entier	%ld
Long entier non signé	%lu
Long flottant	%lf

Important: Au début d'un programme utilisant les fonctions d'affichage et de saisie 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 */

void 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 Hexadécimal */
}
```

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>

void 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 ou flottants en rajoutant des caractères de remplissage.

- 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 de forcer l'affichage du signe.
- Un nombre qui précise **le nombre de caractères qui doit être affiché** suivi d'un **point** et **d'un nombre précisant combien de chiffres après la virgule doivent être affichés**.

```
Syntaxe:
<Nb caractères affichés>.<Nombre de chiffres significatifs>
```

Exemple:

```
#include <stdio.h>

float i;

void 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
 15.600000
     15.60
15.60
+15.60
0000015.60
```

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

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

```
Syntaxe:
putchar(identificateur1);
```

Exemple:

```
#include <stdio.h>

char car1,car2;

void 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];

void 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'affichage pour **printf** deviennent les codes d'entrée 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;

void 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 variables réelles */
float a,b,c,det;

void 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 d'une variable chaîne de caractères.

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

void main()
{
     /* Affichage de "Quel est ton nom ?" */
     puts("Quel est ton 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;

void main()
{
     /* Affichage de "Tapez un caractère ?" */
     printf("Tapez un caractère ?");
     /* Saisie d'un caractère */
     car1=getchar();
     /* Changement de ligne */
     putchar('\n');
     /* Affichage du caractère saisi */
     printf("Le caractère saisi = %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 d'une chaîne de 19 caractères */
char nom[20];

void 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 prioritaires sur les autres 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;

void 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;

void 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>
unsigned char porta,i;     /* Déclaration deux octets */
void 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;

void 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 **booléenne**.

- **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;

void 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;

void 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 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`**, 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 des variables */
char car;
int  a,b,c;
float g;

void 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 effectuer l'évaluation en flottant.

V.9 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 prioritaire ➔ ***Le moins prioritaire***

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

VI Les structures conditionnelles.

Elles permettent en fonction d'une condition, de choisir de faire une instruction 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;

void 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;

void 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 **alors** elle exécute l'instruction ou le bloc d'instructions qui suit le `if`(si), par contre si la condition est **fausse** 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;
      ...
}
```

Exemple:

```
#include <stdio.h>

int a,b;

void 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;

void 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 ou égal à b=%d",a,b);
            printf("\n");
     }
}
```

VI.3 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 ce sont les instructions qui suivent **`default`** qui sont exécutées .

```
Syntaxe:
switch (identificateur)
{
case valeur1 :
        instruction_1 ;         ou         {instructions_1...} ;
     break;
case valeuurm :
        instruction_2 ;         ou         {instructions_2...} ;
     break;
case valeurk:
        instruction_j ;         ou         {instructions_j...} ;
     break;
default :
        instruction_i ;         ou         {instructions_i...} ;
     break;
}
```

Exemple:

```
#include <stdio.h>

char choix;

void 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;

void 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.

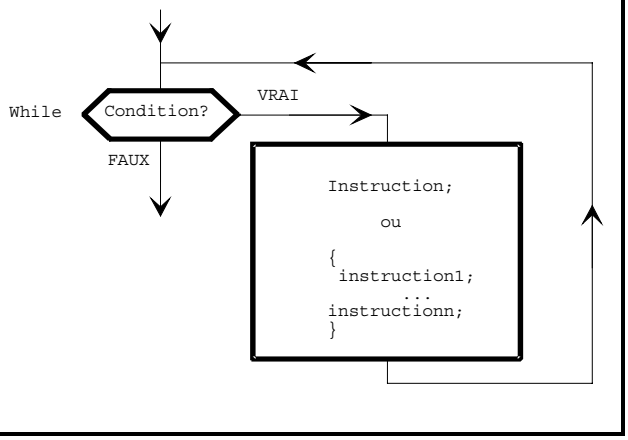
Une structure itérative est la répétition d'une ou plusieurs instructions tant que la condition de sortie est ***VRAIE***, en fonction des différents types de structures itératives la condition pourra être testée en début ou en fin de la structure.

VII.1 La structure <*TANT QUE ... FAIRE ...*>.

Dans cette structure la **condition** est testée au **début** .

Syntaxe:

```
while (condition) instruction;

ou

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



Exemple:

```
#include <stdio.h>

int i;

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

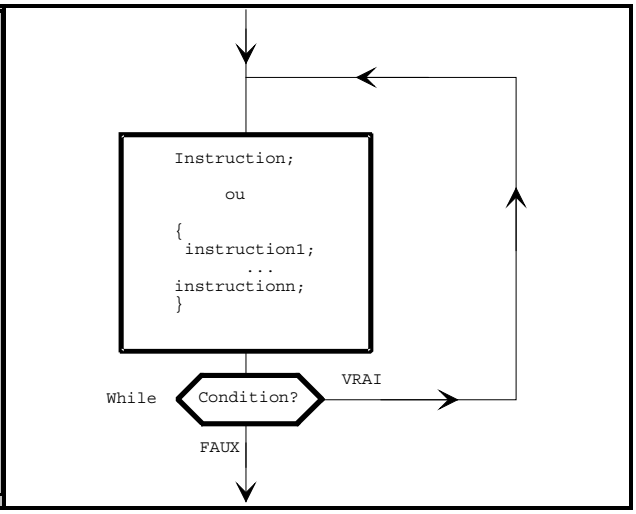
ou

```
#include <stdio.h>

int i;

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

VII.2 La structure <*FAIRE ... TANT QUE ...*>.

Dans cette structure la **condition** est testée à la **fin**.

```
Syntaxe:
do instruction;
while (condition) ;

ou

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



Exemple:

```
#include <stdio.h>

int i;

void main()
{
     i=0;

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

ou

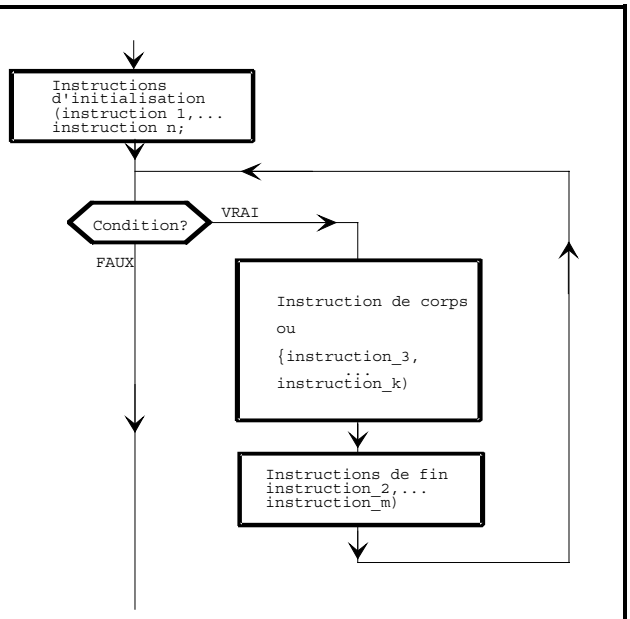
```
#include <stdio.h>

int i;

void main()
{
     i=0;

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

VII.3 La structure <*POUR ... FAIRE ... JUSQU'A ...*>.

Dans cette structure la **condition** 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 exécuté à chaque itération.
- L'instruction ou plusieurs instructions qui sont exécutées à la fin de chaque itération.

```
Syntaxe:
for
(instr_1,...,instr_n d'init;
   condition ;
instr_2,...,instr_m de fin)
 instruction de corps;

ou

for
(instr_1,...,instr_n d'init;
   condition ;
instr_2,...,instr_m de fin)
{
         instruction_3;
           ...
         instruction_k;
}
```



Exemple:

```c
#include <stdio.h>

char car;

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

ou

```c
#include <stdio.h>

char car;

void main()
{
     car=65;

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

IX 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.

IX.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;

void 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";

void 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);
}
```

IX.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;
```

<classe> est facultatif

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

Exemple:

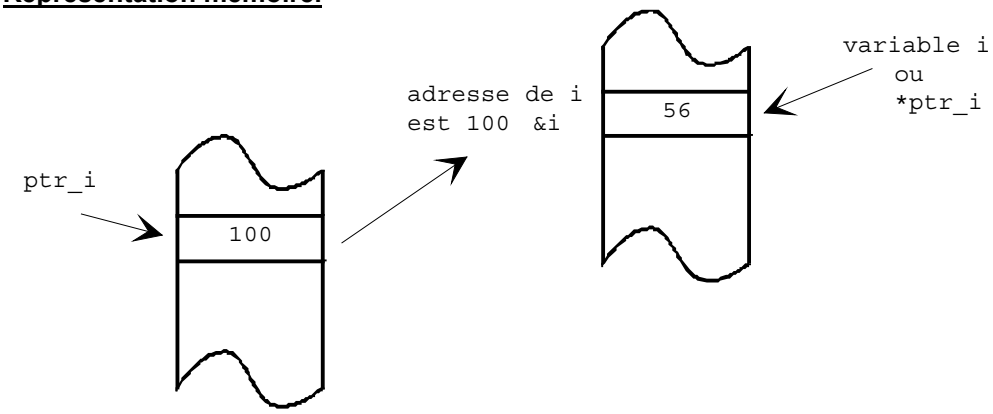
```
#include <stdio.h>

/* Déclaration d'un entier */
int i;

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

void 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) pointé par**.

Exemple:

```
#include <stdio.h>

/* Déclaration d'un entier */
int i;

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

void 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`**.

IX.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 2 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;

void 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 entier codé sur 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 la 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;

void 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 complètement 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;

void 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 la 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;

void 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;

void 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 Les fonctions.

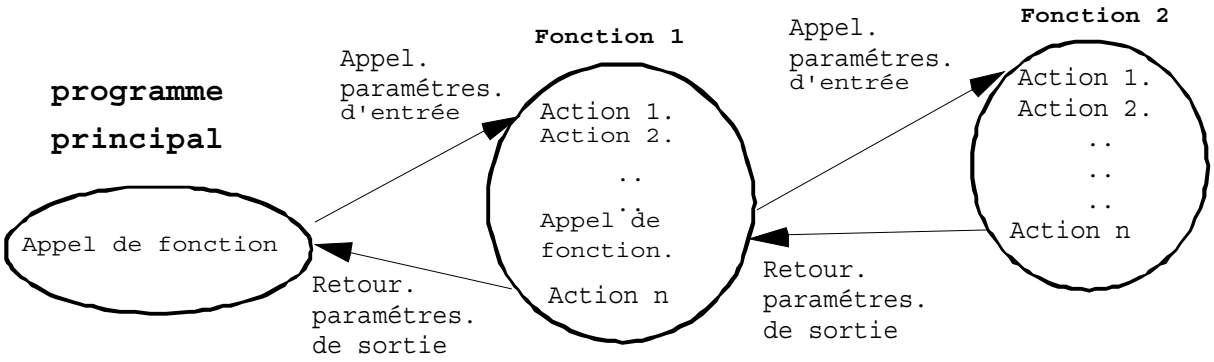
Une fonction ou procédure ou encore sous programme est **une suite d'instructions élémentaires**, mais vue du programme principal **`main()`**, elle représente **une seule action**. Elle est la base des langages structurés.

En effet, l'écriture de tout programme 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 continue juste après l'appel de la fonction.

Une fonction peut appeler 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).

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.



X.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 prototype se différencie de la déclaration de fonction par l'ajout d'un point virgule **`;`** en fin de ligne. Elle est située au début du programme, voir chapitre I structure d'un programme.

```
Syntaxe:
<type_iden_sortie> iden_fonc(<type> iden_1,..., <type> iden_n);
```

`type_iden_sortie:` type de l'identificateur du 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_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);
}
```

type_iden_sortie: type de l'identificateur du 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 à exécuter en donnant les variables d'entrées et ou l'affectation de la variable de sortie.

```
Syntaxe:
iden_var_sortie = iden_fonc(iden_1_var,...,iden_n_var) ;

          ou

iden_fonc.(iden_1_var,...,iden_n_var) ;
```

iden_var_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.

X.1.1 Les fonctions sans paramètre d'entrée et de sortie.

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

Pour dire au compilateur qu'aucun paramètre n'est nécessaire, 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 */
void 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 */
void main()
{
     saisie();
     traitement();
     affichage();
}
```

Que faut-il constater ? : Le listing complet est plus long mais, le programme principal est très court et plus structuré.

X.1.2 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 */
void 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 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 */
void 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 , l**a valeur de la variable locale val3 sera recopiée dans la variable globale `res`** par l'instruction **`return(val3)`**.

X.1.3 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 sortie**.

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);
}

void 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 changé au deuxième affichage. C'est normal car on a fait un passage par valeur. c'est-à-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);
}

void 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 fonction **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 changé 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, toute modification de valeurs des objets pointés par **val1** et **val2** dans **fonction** seront répercutées dans **a** et **b**.

exemple de passage par adresse de tableau:

fonction calculant la longueur d'une 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 caractères dans un 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 */
}

void 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 standards du **C**. Il y en une qui donne le nombre de caractères d'une chaîne elle s'appelle **`strlen`**, voir chapitre suivant.

X.2 Les fonctions standards du C.

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

X.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 ce 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 formatée.
gets()	Saisie d'une chaîne de caractères.
puts()	Affichage d'une chaîne de caractères.

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

X.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 : exemple conversion en 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 hexadécimal
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 **VRAI**.

X.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.

X.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 Cosinus.
asin()	Arc Sinus.
atan()	Arc Tangente.
atan2()	Arc Tangente X/Y.
ceil()	Arrondit par excès.
cos()	Cosinus.
cosh()	Cosinus 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.
sinh()	Sinus Hyperbolique.
sqrt()	Racine Carrée.
tan()	Tangente.
tanh()	Tangente Hyperbolique.