# CHAPITRE 04 : CONCEPTS DE BASE DU LANGAGE C

En 1970, Dennis RITCHIE a créé le langage C, un langage de haut niveau, pour écrire le système d'exploitation Unix. La conception de ce langage a été régie par les pré requis suivants :

- la souplesse
- la fiabilité
- la portabilité
- les possibilités de l'assembleur

## 1 STRUCTURE D'UN PROGRAMME EN C

### 1.1 UN PREMIER PROGRAMME EN C

Exemple :

Ce programme affiche le message bonjour

```
main( )
{
        printf("bonjour la première année");
}
```

main () indique qu'il s'agit du programme principal.
{ et } jouent le rôle de début et fin de programme.
Printf est l'instruction d'affichage à l'écran, le message étant entre guillemets.
; indique la fin d'une instruction.

Remarque :

Une représentation claire et aérée du programme, avec indentation sensée, améliore la lisibilité. Elle est vivement conseillée.

### 1.2 STRUCTURE GENERALE D'UN PROGRAMME EN C

Un programme en C se présente en général sous la forme suivante :

```
<Directives de compilation>
<Déclaration de variables externes>
<Déclaration de prototypes de fonctions>
main ( )
{
        corps du programme
        (commentaires,
        déclaration de variables et constantes,
        instructions)
}
<Définition de fonctions>
```

Remarque :

Un programme en C doit contenir au moins le programme principal (la partie main).

Nous nous intéressons dans ce chapitre aux directives de compilation et différents éléments de base composant le corps du programme.

## *2 LES DIRECTIVES DE COMPILATION*

Nous en citons les directives include et define:

Syntaxe :

#include < nom_fichier>

indique le fichier de bibliothèque C à inclure dans le programme. Ce fichier contient les définitions de certaines fonctions prédéfinies utilisées dans le programme.

Syntaxe :

#define expression_à_remplacer expression_de_remplacement

permet de remplacer un symbole par une constante ou un type ou de faire des substitutions avec arguments dans le cas des macros.

Exemples :

Directives de compilation

```
#include <stdio.h>          /*ce fichier contient les fonctions d'entrées/sorties comme
                              printf*/
#define pi 3.14             /*pi sera remplacé par la valeur 3.14*/
#define entier int          /*entier sera remplacé par le type prédéfini int*/
#define somme(x,y) x+y      /*la macro somme(x,y) sera remplacée par x+y*/
```

## *3 LES COMMENTAIRES*

Un commentaire est un texte placé entre les signes /* et */. Il permet de commenter une ou plusieurs lignes de commandes en vue d'éclairer le lecteur.

Syntaxe :

/* texte du commentaire */

Exemples :

Commentaires

```
main( )
{
      printf("bonjour");          /* ce programme affiche bonjour*/
}
```

# *4 LES VARIABLES ET LES CONSTANTES*

## 4.1 DECLARATION DE VARIABLES

A toute variable utilisée dans un programme C doivent être associés d'abord (avant toute utilisation) un nom dit identificateur et un type de données (entier, réel ou caractère…). Lors de l'exécution, une zone mémoire (dont la taille dépend du type) sera réservée pour contenir la variable.

Syntaxe :

```
Type      identificateur;
Ou
Type      identificateur1, identificateur2, … ,identificateur n;
```

Exemples :

Déclaration de variables

```
int i;          /* i est une variable de type entier */
float j,k;      /* j et k sont des variables de type réel */
char c;         /* c est une variable de type caractère */
```

a) Identificateurs

L'emploi des identificateurs doit répondre à un certain nombre d'exigences :

un identificateur doit être composé indifféremment de lettres et chiffres ainsi que du caractère de soulignement ( _ ) qui peut remplacer des espaces.

Un identificateur doit commencer par une lettre ou le caractère de soulignement. Néanmoins, celui-ci est souvent utilisé pour désigner des variables du système.

Seuls les 32 premiers caractères (parfois, uniquement les 8 premiers) sont significatifs (pris en compte par le compilateur).

Majuscules et minuscules donnent lieu à des identificateurs différents.

Un identificateur ne doit pas être un mot réservé (utilisé dans le langage C comme int, char, … ).

Exemples :

Identificateurs

solution1 est un identificateur valide (constitué de lettres et de 1)
1solution n'est pas un identificateur valide.
prix unitaire n'est pas un identificateur valide (Il contient un espace).
prix_unitaire est un identificateur valide.
jour, Jour et JOUR sont 3 identificateurs différents
int n'est pas un identificateur valide. C'est un mot utilisé en C

b) Types de données

Un type est un ensemble de valeurs que peut prendre une variable. Il y a des types prédéfinis et des types qui peuvent être définis par le programmeur.

Types simples prédéfinis en C

| Type | Signification | Représentation système | |
|---|---|---|---|
| | | Taille (bits) | Valeurs limites |
| int | Entier | 16 | -32768 à 32767 |
| short (ou short int) | Entier | 16 | -32768 à 32767 |
| long (ou long int) | Entier en double longueur | 32 | -2147483648 à 2147483647 |
| char | Caractère | 8 | |
| float (ou short float) | Réel | 32 | $\pm 10^{-37}$ à $\pm 10^{38}$ |
| double(ou long float) | Réel en double précision | 64 | $\pm 10^{-307}$ à $\pm 10^{308}$ |
| long double | Réel en très grande précision | 80 | $\pm 10^{-4932}$ à $\pm 10^{4932}$ |
| unsigned | Non signé (positif) | 16 | 0 à 65535 |

Remarques :

int équivaut à short sur PC et à long sur station.

La fonction sizeof retourne la taille en octets d'un objet.

Exemples :

```
n=sizeof(int);          /* n reçoit 2 */
n=sizeof(3.14);         /* n reçoit 8 */
```

c) fonctions prédéfinies sur les types simples

Des fonctions appliquées au différents types de données sont prédéfinies dans des fichiers de bibliothèque C.

Fonctions mathématiques

Math.h

Ce fichier contient des fonctions mathématiques pouvant être appliquées aux types numériques.

Exemples :

Exemples de fonctions mathématiques

```
#include <math.h>       /*pour inclure le fichier math.h*/
main( )
{
int p,i=4,j=-2;         /* p entier et i et j entiers initialisés à 4 et -2*/
float r;                /* r réel*/
p=pow(i,2);             /* p reçoit 16 (4 à la puissance 2) */
r=sqrt (i);             /* r reçoit 2 (racine carrée de 4) */
i=abs(j);               /* i reçoit 2 (valeur absolue de -2)*/
}
```

Fonctions sur les caractères

ctype.h

Ce fichier contient les définitions des fonctions pouvant être appliquées à des caractères.

Ces fonctions permettent de vérifier si un caractère appartient à une catégorie donnée. Elles retournent 0 si faux et une valeur différente si vrai.

Listes des fonctions sur les caractères

| Fonction | Signification |
|---|---|
| isalpha (c) | c est une lettre |
| isupper (c) | c est une lettre majuscule |
| islower (c) | c est une lettre minuscule |
| isdigit (c) | c est un chiffre |
| isxdigit (c) | c est hexadécimal [0-9], [A-F] ou [a-f] |
| isalnum (c) | c est alphanumérique (chiffre ou lettre) |
| isspace (c) | c est un blanc, tabulation, retour chariot, newline ou formfeed |
| ispunct (c) | c est un caractère de ponctuation |
| isprint (c) | c est un caractère imprimable (de 32 (040) à 126 (0176) tilde) |
| isgraph (c) | c est un caractère imprimable différent d'espace |
| iscntrl (c) | c est un caractère de contrôle différent d'espace et (<32) ou delete (0177) |
| isascii (c) | c est un caractère ASCII (0 c < 128) |

## *4.2 DECLARATION DE CONSTANTES*

Une constante est une donnée dont la valeur ne varie pas lors de l'exécution du programme.

Elle doit être déclarée sous forme :

Syntaxe :

const Type Identificateur = Valeur ;

Remarque:

Une valeur constante peut, également, être exprimée au moyen d'un identificateur défini en utilisant la directive define:

#define Identificateur Valeur

Exemples :

Déclaration de constantes

```
1-  main( )
    {
    const float     pi=3.14;            /*déclare la constante pi avec const*/
    printf("pi égale à %f",pi);         /*affiche la valeur de pi*/
    }
2-  #define pi   3.14                  /*définit la constante pi avec define*/
    main( )
    {
    printf("pi égale à %f",pi);         /*affiche la valeur de pi*/
    }
```

## 4.3 INITIALISATION DE VARIABLES

Une valeur initiale peut être affectée à une variable dès la déclaration sous forme :

Syntaxe :

a) Cas de types numériques :

Type Identificateur = Valeur numérique ;

b) Cas du type caractère :

Type Identificateur = 'caractère' ;

Ou

Type Identificateur = code ASCII d'un caractère ;

Exemples :

Initialisation de variables

```
main( )
{
int i, j=3, k;              /* seul j est initialisé à 3*/
float f=1.2 e5;             /* f est initialisé à 120000 (1.2*10^5)*/
int i=011;                  /* i est initialisé à 11 en octal soit 9 en décimal*/
char c='A';                 /* c est initialisé à la lettre A*/
char c=65;                  /* c est initialisé à A dont le code ASCII est 65*/
}
```

Remarque :

Des caractères spéciaux sont représentés à l'aide du métacaractère \.

Exemples :

Initialisation des variables

```
main( )
{
Char c = '\'';          /*c reçoit un apostrophe*/
Char c = '\x041';       /* c reçoit 41 code ASCII en hexadécimal de A*/
}
```

Liste des caractères spéciaux

| Représentation | Signification |
|---|---|
| \0 | Caractère NULL |
| \a | Bip (signal sonore) |
| \b | Espace arrière |
| \t | Tabulation |
| \n | Nouvelle ligne |
| \f | Nouvelle page |
| \r | Retour chariot |
| \" | Guillemet |
| \' | Apostrophe |
| \\ | Antislash (\) |
| \ddd | Caractère ayant pour valeur ASCII octale ddd |
| \x hhh | Caractère ayant pour valeur ASCII hexadécimale ddd |

## *5 L'AFFECTATION (ASSIGNATION)*

L'affectation est l'opération qui attribue à une variable, au moyen de l'opérateur =, une valeur constante ou résultat d'une expression.

Syntaxe :

Variable = Valeur ou expression ;

Exemples :

Affectation

```
main( )
{
int i, j;
i=2;            /*i reçoit 2*/
j=(i*3)+5;      /*j reçoit 11 résultat de (2*3)+5*/
i=j=2;          /*j reçoit d'abord 2 puis i reçoit la valeur de j (2)*/
}
```

Remarque :

C permet de faire des assignations entre des variables de types différents. Des conversions de types sont alors automatiquement réalisées.

Exemples :

Conversion automatique de types

```
1-  main( )
    {
    int i;              /*i entier*/
    float j=3.5;        /*j réel initialisé à 3.5*/
    i=j;                /*i reçoit 3, le résultat est tronqué*/
    }

2-  main( )
    {
    int i;              /*i entier*/
    char c='A';         /*c caractère initialisé à A*/
    i=c;                /*i reçoit 65 le code ASCII de A*/
    }
```

# *6 LES ENTREES/SORTIES*

## 6.1 L'AFFICHAGE

L'instruction printf permet d'obtenir un affichage formaté à l'écran.

Syntaxe :

a) affichage de message constitué de texte et de caractères de contrôle

```
Printf ("texte et caractères de contrôle") ;
```

b) affichage de valeurs de variables ou d'expressions

```
Printf("message et formats d'affichage", arg1, arg2,…,argn);
```

Noms de variables ou expressions

Exemples :

Affichage

```
main( )
{

int i=2,k=5;                          /*i et k entiers initialisés à 2 et 5*/
float j=3.5;                          /*j réel initialisé à 3.5*/

printf("Donnez le prix unitaire");    /*le programme affiche
                                        Donnez le prix unitaire */

printf("Donnez le prix unitaire \n"); /*le programme affiche Donnez le prix
                                        unitaire et retourne à la ligne (\n)*/

printf("la valeur de i est %d\n ",i); /*le programme affiche la valeur
                                       de i est 2 et retourne à la ligne*/

printf("i=%d       j=%f",i,j);        /*le programme affiche
                                        i=2          j=3.5*/


printf("i=%d\nj=%f",i,j);             /*le programme affiche
                                              i=2
                                              j=3.5*/

printf("somme(%d,%d)=%d\nFIN",i,k,i+k);   /*le programme affiche
                                              somme(2,5)=7
                                              FIN*/

printf("j=%4.2f\n",j);                /*le programme affiche
                                        j=  3.50*/
}
```

Remarques :

Un caractère de contrôle est précédé de \ comme \n qui provoque un interligne (voir liste des caractères spéciaux, paragraphe 2-4).

Chaque format d'affichage est introduit par le caractère % suivi d'un caractère qui indique le type de conversion.

Des indications peuvent être rajoutées entre le % et le caractère comme le nombre minimum de caractères réservés à l'affichage de la mantisse d'un nombre et le nombre de décimales.

Liste des formats d'affichage

| Format d'affichage | Signification |
|---|---|
| %d | Conversion en décimal |
| %o | octal |
| %x | hexadécimal (0 à f) |
| %X | hexadécimal (0 à F) |
| %u | entier non signé |
| %c | caractère |
| %s | chaîne de caractères |
| %l | long ou double |
| %L | long double |
| %e | sous forme m.nnnexx |
| %E | sous forme m.nnnExx |
| %f | sous forme mm.nn |
| %g | Semblable à e ou f selon la valeur à afficher |

## 6.2 LA LECTURE DE DONNEES

L'instruction scanf effectue la lecture des variables.

Syntaxe :

scanf("formats d'affichage", variable1, variable2,…,variablen) ;

Remarque :

Seules les variables scalaires (entiers, réels et caractères) doivent être précédées de &.

Exemples :

Lecture

```
#include <stdio.h>
main( )
{
int i;                          /*i entier*/
float k;                        /*k réel*/
char m;                         /* m caractère*/
scanf("%d",&i);                 /*le programme lit une valeur entière et l'affecte à i*/
scanf("%d%f",&i,&k);            /*le programme lit une valeur entière de i
                                  puis une valeur réelle de k*/
scanf("%c",&m);                 /*le programme lit un caractère et l'affecte à
                                  la variable m*/
}
```

Remarques :

La notation &variable est utilisée pour indiquer l'adresse mémoire de la variable en question.

Les données tapées au clavier sont d'abord placées dans un tampon interne. Scanf va chercher ces données dans ce tampon, sans nécessairement le vider entièrement. C'est pourquoi, la

fonction scanf est malheureusement une source permanente de problèmes (tampon associé au clavier encombré de résidus de lectures précédentes). Elle n'est, en général, acceptable qu'à condition de se limiter à des lectures d'entiers ou de réels.

# *7 LES OPERATEURS*

## 7.1 LES OPERATEURS ARITHMETIQUES

Les opérateurs arithmétiques traditionnels sont :

| | |
|---|---|
| + | addition |
| - | soustraction |
| * | multiplication |
| / | division |
| % | reste de la division entière |

Exemples :

Opérateurs arithmétiques

```
main( )
{
int i=9,j=4,k;          /*i, j et k entiers*/
float x=9.0, y;         /*x et y réel*/

k=i+j;                  /*k reçoit 13*/
y=i/j;                  /*y reçoit 2.0 (division entière : i et j entiers)*/
y=x/j;                  /*y reçoit 2.25 (division réelle : x réel*/
y=9.0/j;                /*y reçoit 2.25 (division réelle : 9.0 réel*/
k=i%j;                  /*k reçoit 1 (reste de la division de i par j)*/
}
```

Remarques :

l'opérateur / effectue, en fonction du type des opérandes, une division entière (euclidienne) ou réelle.

L'ordre des priorités des opérateurs est important.

Il est possible de forcer la conversion du type d'une variable ou d'une expression en les préfixant d'un type au choix.

Exemples :

Conversion de types

```
main( )
{
int i=9,j=4;            /*i et j entiers*/
float y;                /*y réel*/
y=(float) i / j;        /*y reçoit 2.25 (i est converti d'abord en réel=9.0)*/
y=(float) (i/j);        /*y reçoit 2.0 (l'entier 2 (i/j) est converti en réel)*/
}
```

## 7.2 LES OPERATEURS +=, -=, *=, /=

Ils sont utilisés pour faciliter l'écriture.

| Ecriture classique | Ecriture équivalente |
|---|---|
| i = i + 20; | i += 20; |
| i = i - 20; | i -= 20; |
| i = i * 20; | i *= 20; |
| i = i / 20; | i /= 20; |

## 7.3 LES OPERATEURS LOGIQUES

Les opérateurs logiques sont, par ordre décroissant de priorité :

| Opérateurs | signification |
|---|---|
| ! | Non logique |
| > >= < <= | Test de supériorité et d'infériorité |
| == et != | Test d'égalité et d'inégalité |
| && et \|\| | ET et OU logique |

Exemples :

Opérateurs logiques

```
#include <stdio.h>
main( )
{
int a,b,c; /*a,b et c entiers*/ printf ("Introduire
a, b et c : "); scanf ("%d%d%d",&a,&b,&c);

if (a==b && b!=c)              /*si a=b et b c affiche le message suivant*/
        printf("a égale à b et b différent de c\n");

if (!(a<b) || a==0)            /*si a>=b ou a=0 affiche le message suivant*/
        printf("a est supérieure ou égale à b ou a égale à 0\n");
}
```

Remarques :

Soit i une variable numérique,

l'expression if (i) est équivalente à l'expression if (i!=0). l'expression if (!i) est équivalente à l'expression if (i==0).

## 7.4 LES OPERATEURS ++ ET --

Ils permettent d'incrémenter ou de décrémenter une variable. L'opérateur ++ (--) effectue une pré-incrémentation (pré-décrémentation) ou une post-incrémentation (post-décrémentation) selon son emplacement après ou avant la variable.
Dans une opération d'affectation qui met en jeu l'opérateur de :

pré-incrémentation (pré-décrémentation), la variable est d'abord incrémentée (décrémentée) de 1. L'affectation est ensuite effectuée.

post-incrémentation (post-décrémentation), L'affectation (sans les ++ (--)) est effectuée avant l'incrémentation (décrémentation).

Exemples :

Soient i=3 et j=5,

| Instruction | Equivalent | Résultats |
|---|---|---|
| i++; | i=i+1; | i=4 |
| ++i; | i=i+1; | i=4 |
| i--; | i=i-1; | i=2 |
| --i; | i=i-1; | i=2 |
| i= ++j; | j=j+1; i=j; | j=6 et i=6 |
| j= ++i + 5; | i=i+1; j=i+5; | i=4 et j=9 |
| j= i++ + 5; | j=i+5; i=i+1; | j=8 et i=4; |

## 7.5 LES OPERATEURS DE TRAITEMENT DE BITS

Rappel : (opérations logiques)

| | | |
|---|---|---|
| 1&1=1 | 1\|1=1 | 1^1=0 |
| 1&0=0 | 1\|0=1 | 1^0=1 |
| 0&1=0 | 0\|1=1 | 0^1=1 |
| 0&0=0 | 0\|0=0 | 0^0=0 |

Les opérateurs de traitement de bits en C s'appliquent uniquement à des entiers ou des caractères. Ils ne s'appliquent, donc, pas à des opérandes réels (de type float ou double).

| Opérateur | Traitement |
|---|---|
| & | et binaire |
| \| | ou binaire |
| ^ | ou exclusif |
| << | décalage à gauche |
| >> | décalage à droite |
| ~ | complément à 1 |

Exemples :

Opérateurs de traitement de bits

```
main( )
{
int i=597;          /*i est 0000 0010 0101 0101 en binaire, 255 en hexadécimal*/
int u=12;           /*u est 0000 0000 0000 1100 en binaire*/

i= i & 0x FFFE; /*i reçoit 0000 0010 0101 0100 en binaire,254 en hexadécimal*/
v= u >> 2;          /*v reçoit 0000 0000 0000 0011 en binaire, 3 en décimal
                      (u est décalé de 2 bits à droite)*/
v= ~u;              /*v reçoit 1111 1111 1111 0011 (complément à 1 de u)*/
}
```

# *8 LES INSTRUCTIONS SELECTIVES*

## *8.1 L'INSTRUCTION SI (IF)*

L'instruction if sélectionne le traitement (bloc d'instructions) à faire si une condition est vérifiée.

Syntaxe :

```
if (condition)
        {
            instruction1;
            instruction2;
            …                    Traitement
            instructionn;

        }
```

Exemples :

Instruction if

```
#include <stdio.h>
main( )
{
int a,b;                        /*a et b entiers*/
printf ("Introduire a et b : ");
scanf ("%d%d",&a,&b);
if (a==0)                       /*si a=0 affiche les messages zéro puis FIN*/
    {
        printf("zéro\n");
        printf("FIN\n");
    }
if (a==1)                        /*si a=1 affiche le message un*/
        printf("un\n");
if (a=b)                    /*faire a=b; puis if (a) [si a 0] affiche le message suivant*/
    {
        printf("a est différent de 0\n");
    }
```

```
}
```

Remarque :

Si le traitement à effectuer est constitué d'une seule instruction, il est possible d'omettre les accolades.

Lorsque if est utilisée avec else, elle indique également le traitement à faire si la condition n'est pas vérifiée.

Syntaxe :

```
if (Condition)
        {
            instruction1;
            instruction2;
                                    Traitement1
            …
            instructionn;
        }
else
        {
            instruction1';
            instruction2';
                                    Traitement2
            …
            instructionn';
        }
```

Exemples :

Instruction if else

```
#include <stdio.h>
main( )
{
int a,b;                    /*a et b entiers*/
printf ("Introduire a et b : ");
scanf ("%d%d",&a,&b);

if (a==b)                   /*si a=b affiche a égale à b sinon affiche a différent de b */
        printf("a égale à b\n");
else
        printf("a différent de b\n");
}
```

Remarque :

Dans une imbrication, else se rapporte toujours à if le plus rapproché (le plus interne).

Exemples :

Instructions if else imbriquées

```
#include <stdio.h>

main( )
{
int a,b,z,n;                /*a, b,z et n entiers*/

printf ("Introduire a, b et n : ");
scanf ("%d%d%d",&a,&b,&n);

if (n>0)
        if (a>b)            /*si n>0 et a>b*/
                z=a;
        else                /*si n>0 et a<=b (else se rapporte à if (a>b))*/
                z=b;

if (n>0)
        {
          if (a>b)          /*si n>0 et a>b*/
                  z=a;
        }
else                        /*si n<=0 (else se rapporte à if (n>0))*/
                z=b;
}
```

## 8.2 L'INSTRUCTION SWITCH

Elle réalise un aiguillage vers différentes instructions en fonction du contenu d'une variable de contrôle.
Le sélecteur de switch (la variable de contrôle) doit être un entier ou un caractère.

Syntaxe :

```
switch (Variable de contrôle)
{
            case Valeur1 : Traitement1 (bloc d'instructions)
                           break;
    case Valeur2 :         Traitement2
                           break;
                …
    case Valeurn :         Traitementn
                           break;
    default :              Traitementm
}
```

Remarque : Valeuri est une constante.

Exemples :

Instruction switch

```
#include <stdio.h>
main( )
{
int a;                    /*a entier*/
char c;                   /*c char*/
printf ("Introduire un nombre et une lettre: ");
scanf ("%d%c",&a,&c);
switch (a)                /*le programme traite tous les cas de a (0,1 ou autres)*/
{
      case 0 :    printf ("Le nombre introduit est zéro\n");
                  break;
      case 1 :    printf ("Le nombre introduit est 1\n");
                  break;
      default :   printf ("Le nombre introduit est différent de 0 et 1\n");
}
switch (c)                /*Ici, on traite uniquement les cas où c égale à x ou y*/
{
      case 'x' :  printf ("La lettre saisie est la lettre x\n");
                  break;
      case 'y' :  printf ("La lettre saisie est la lettre y\n");
                  break;
}
}
```

Remarque :

L'instruction break fait sortir de switch.

# *9 LES INSTRUCTIONS ITERATIVES*

## 9.1 L'INSTRUCTION TANT QUE (WHILE)

L'instruction while permet de répéter un traitement autant de fois qu'une condition est vérifiée. Les instructions en question sont alors exécutées tant que la condition est vraie.

Syntaxe :

```
while (Condition)
      {
            Traitement (bloc d'instructions qui se terminent par ;)
      }
```

fonctionnement :

Le système teste d'abord si la condition est vraie; si oui, exécute le traitement et remonte automatiquement à la ligne while pour tester de nouveau la condition. Elle s'arrête quant la condition devient fausse.

Exemples:

Instruction while

```
#include <stdio.h>
main( )
{
int n,somme=0;          /*n entier et somme entier initialisé à 0*/

printf("Introduire n : " );
scanf("%d",&n);
while (n>0)             /*tant que n>0, le programme rajoute n à la somme obtenue
                        puis décrémente n*/
        {
          s=s+n;
          n--;
        }
 printf ("%d",somme); /*le programme affiche la somme des nombres compris entre 0
                            et n*/
}
```

Remarque :

Lors de l'utilisation de l'instruction while, à chaque itération, la condition est évaluée en premier, avant l'exécution du traitement.

## 9.2 L'INSTRUCTION FAIRE TANT QUE (DO WHILE)

L'instruction do while permet de répéter un traitement jusqu'à ce qu'une condition ne soit plus vérifiée. Elle joue le même rôle que while. Néanmoins, Lors de l'utilisation de l'instruction while, à chaque itération (fois), le traitement est exécuté en premier, avant que la condition ne soit évaluée.

Syntaxe :

```
do
    {
        Traitement (bloc d'instructions qui se terminent par ;)
    }
while (Condition) ;
```

fonctionnement :

Le système exécute d'abord le traitement puis teste si la condition est vraie; si oui, il remonte automatiquement à la ligne do pour exécuter de nouveau le traitement. Il s'arrête quant la condition devient fausse.

Exemples:

Instruction do while

```
#include <stdio.h>
main( )
{
int n, somme=0, i=0;            /*n entier et somme et i entiers initialisés à 0*/

printf("Introduire n : " );
scanf("%d",&n);
do                              /*le programme rajoute i à somme puis l'incrémente tant que i ≤ n*/
        {
                somme=somme+i
                i++;
        }
while (i<=n)        ;
printf ("%d",somme); /*le programme affiche la somme des nombres compris entre 0
                                et n*/
}
```

Remarque :

Lors de l'utilisation de l'instruction do while, le traitement sera exécuté au moins une fois quelle que soit la condition.

Exemple:

Instructions while et do while

```
#include <stdio.h>
main( )
{
int i=3;                /*i entier initialisé à 3*/

do
        {
        printf ("%d",i);
        i++;
        }
while (i<3) ;           /*avec do while, i (3) sera affiché même si la condition i<3 est fausse
                        au début*/

i=3;                    /*remet i à 3*/
while (i<3)             /*avec while, rien n'est affiché car la condition i < 3 est fausse*/
        {
        printf ("%d",i);
        i++;
        }
}
```

## 9.3 L'INSTRUCTION POUR (FOR)

L'instruction for permet de répéter un traitement donné un nombre de fois précis.

Syntaxe :

```
for (Initialisations;Condition;Instructions)
    {
        Traitement (bloc d'instructions qui se terminent par ;)
    }
```

fonctionnement :

for commence au départ, par effectuer les initialisations (en premier argument), exécute le traitement tant que la condition (en deuxième argument) est vérifiée et exécute les instructions (en troisième argument) à chaque fin d'itération.

Exemples:

Instruction for

```
#include <stdio.h>
main( )
{
int i,j,n,somme=0;        /*i, j, n entiers et somme entier initialisée à 0*/

printf("Introduire n : " );
scanf("%d",&n);
for (i=1; i<=n; i++) /*pour i allant de 1 à n, le programme rajoute i à somme*/
        somme=somme+i ;
printf ("%d",somme);      /*le programme affiche la somme des valeurs comprises entre 0
                           et n*/

for (i=2, j=4; i<5 && j>2; i++, j--)        /*pour i allant de 2 à 4 et j de 4 à 3,
                                              le programme affichera i et j si i<5 et j>2*/
        printf ("i:%d et j:%d\n",i,j);

                /*Cette boucle affichera donc i:2 et j:4
                                              i:3 et j:3*/
}
```

## 9.4 LES INSTRUCTIONS DE SORTIES DE BOUCLES (BREAK ET CONTINUE)

Break permet de sortir directement de la boucle (for, while ou do while) la plus interne.
Continue permet de passer directement à l'itération suivante de la boucle la plus interne.

Exemple:

Instructions break et continue

```
#include <stdio.h>
main( )
{
int i;                          /*i entier*/
for (i=5; i>0; i--)             /*pour i allant de 5 à 1*/
  {
   if (i==5) continue;          /*arrête l'exécution de l'itération1 et passe à l'itération2*/
   printf ("Ok");
   if (i==4) break;             /*arrête l'exécution à l'itération2 (sort de for)*/
  }                             /*le programme affichera donc une fois OK*/
}
```

## 9.5 L'INSTRUCTION ALLER A (GOTO)

L'instruction goto permet de brancher (inconditionnellement) à une ligne du programme. Celle-ci doit avoir été étiquetée (précédée d'une étiquette constituée d'un identificateur suivi de :).

Syntaxe :

goto Etiquette ;

fonctionnement :

Le système interrompt l'exécution séquentielle du programme, remonte ou déscend à la ligne appelée étiquette et poursuit l'exécution à partir de celle-ci.

Exemple:

Instruction Goto

```
#include <stdio.h>
main( )
{
int i=0;                                /*i entier initialisé à 0*/

printf("%d",i);                         /*affiche 0*/
goto message;                           /*saute à l'étiquette message*/
i++;                                    /*ne sera alors pas exécuté*/
printf("%d",i);                         /*ne sera alors pas exécuté*/
message : printf("OK\n");               /*affiche OK*/
printf("FIN\n");                        /*affiche FIN*/

}                                       /*le programme affichera donc 0, OK et FIN*/
```

Remarque :

Goto a la réputation de rendre les programmes moins lisibles. Néanmoins, son utilisation est importante dans des cas qui l'impose.