ESIEE
ENGINEERING

LANGAGE C
IN3E11

Ibrahim ALAME
2023-2024

Introduction

- Langage créé en 1970 par Denis Ritchie sous UNIX, portable, rapide, proche de la machine et possède un grand nombre de bibliothèques.
- Un programme C décrit un algorithme dans un fichier texte, appelé fichier source.
- Ce programme n'est pas directement exécutable.
- Il faut le traduire dans un langage compréhensible par le microprocesseur; **LA COMPILATION.**

La compilation en 4 étapes

- Le traitement par le préprocesseur : Le préprocesseur effectue des transformations textuelles dans le fichier source. (substitution de chaine de caractère, prise en compte des directives de compilation inclusion des autres fichiers sources…)
- La compilation : C'est la traduction en une suite d'instruction du microprocesseur (code assembleur).
- L'assemblage : Transforme le code assembleur en binaire directement compréhensible par le processeur.
- L'édition de liens : C'est la liaison de tous les fichiers objets utilisés par le programme (source(s), librairies de fonctions standard ou autres…) → **EXECUTABLE**

Comment créer un programme

Il faut un éditeur de texte, un compilateur, un debugger

Deux solutions :

- On utilise trois programmes différents
- On utilise un IDE (environnement de développement intégré)

```
1  #include <stdio.h>
2  #define PI 3.14
3  float r,p;
4  void perimetre(float rayon, float *peri);
5  void main(){
6      r=5;
7      perimetre(r,&p);
8      printf("Le périmetre du cercle de rayon %f = %f", r, p);
9  }
10
11 void perimetre(float rayon, float *peri){
12     *peri = 2*PI*rayon;
13 }
14
```

Structure d'un programme C

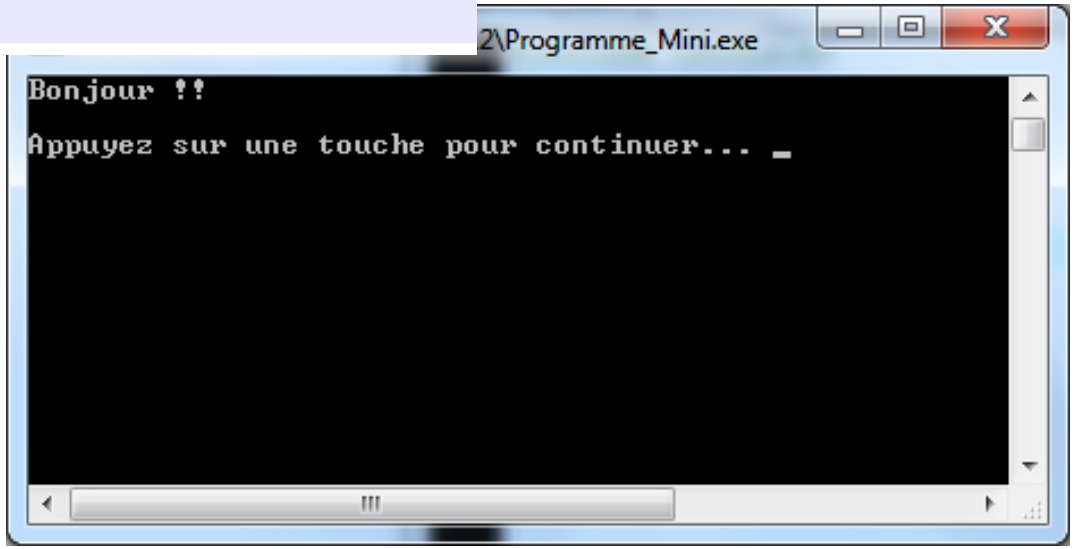
Règles de bases

- #include inclure les fichiers d'entête
- Toutes instructions ou actions se termine par un **;**
- Commentaires sur éventuellement sur plusieurs lignes
  début : /* fin : */
  ou sur une seule ligne // commentaire…
- Un bloc d'instruction commence par **{** et se termine par **}**
- Le langage C est sensible à la casse. Les mots du langage en minuscules.
- Caractères autorisés pour les noms de variables
  - Lettres **non accentuées**
  - Chiffres **sauf au début du nom**
  - Caractère souligné **"_"**

Programme C minimum


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

int main(int argc, char *argv[ ]){
    printf("Bonjour !! \n\n");
          system("PAUSE");
          return(0);
}
```



Variables et Constantes

Définitions:

- Constante : Ne change pas pendant l'exécution du programme
- Variable : Peut changer de valeur pendant l'exécution du programme

Une variable ou une constante est définie par cinq éléments

- **L'identificateur** : c'est son nom
- **Le type** : entier, caractère, chaine de caractère, réel, boolean
- **La taille** : nombre d'octets occupés en mémoire
- **La valeur** : c'est la valeur que l'on attribue à la variable ou la constante
- **L'adresse** : c'est l'emplacement mémoire où est stocké la valeur de la variable ou de la mémoire

Variables et Constantes

Entier : C'est un nombre positif ou négatif exemple : 1584 ou -458.

```
int I;
short S;
long L;
```

Caractère : C'est un nombre entier positif compris entre 0 et 255 et c'est le code ASCII du caractère exemple : 65 est le code ASCII de "A".

```
char C;
```

Chaine de caractère : C'est un ensemble de caractères exemple :

```
char Chaine[8];
```

L	A	N	G	A	G	E	\0
76	65	78	71	65	71	69	\0

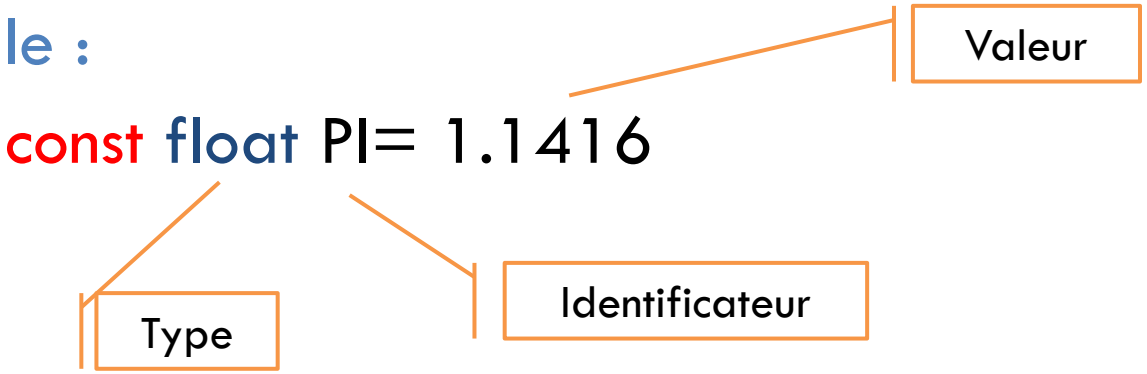
Réel : C'est un nombre à virgule positif ou négatif avec un exposant exemple: $12,45\ 10^4$

```
float F;
double DF;
```

Variables et Constantes

Constantes : const

Exemple :



Remarque :

On peut aussi utilise la directive de compilation **#define**

```
#define  PI  1.1416
```

Variables et Constantes

Syntaxe de déclaration des variables :

<type> <identificateur>,...,<identificateur> = valeur;

Exemple : unsigned int k = 15 ;

int n , p, q= -15 ;

On peut écrire une valeur entière en base hexadécimal ou en octal

- **Hexadécimal :** on ajoute avant la valeur 0x

  exemple : 0xFF ~ 255 décimal

- **Octal :** on ajoute avant la valeur 0

  exemple : 010 ~ 8 décimal

```
1
2  int i,j,k; // Déclaration d'entiers
3
4  void main(){
5      i=2; // Décimal
6      j=031; // Octal  1+3*8=25
7      k=0x25; // Hexadécimal  5+2*16=37
8  }
9
```

Variables et Constantes

Les principaux type de variables :

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

Variables en un sistema de 32-bits						
Variable		card	Limites max y min		Ocupa	
signo	tipo		Min	Max	bytes	bits
signed	char	255	-128	127	1	8
unsigned	char	255	0	255	1	8
signed	short	65535	-32768	32767	2	16
unsigned	short	65535	0	65535	2	16
signed	int	4294967295	-2147483648	2147483647	4	32
unsigned	int	4294967295	0	4294967295	4	32
signed	long	18446744073709600000	-9223372036854780000	9223372036854780000	8	64
unsigned	long	18446744073709600000	0	18446744073709600000	8	64
signed	long long	18446744073709600000	-9223372036854780000	9223372036854780000	8	64
unsigned	long long	18446744073709600000	0	18446744073709600000	8	64

Variables et Constantes

Exemple de déclarations et d'affectations:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  float rayon; // Déclaration de réel
5  int i,j; // Déclaration d'un entier
6  char t; // Déclaration de caractère
7  double pi; // Déclaration de réel double
8  unsigned char octet; // Déclaration d'un octet
9  void main(){
10     rayon = 10.14;
11     i=2;
12     j=3;
13     t='A'; // t=65 Code ASCII de A
14     pi=3.14159;
15     octet=129; // On oeut aller au dessus de 127
16 }
```

Variables et Constantes

Les tableaux:

Ils permettent de stocker des variables de même type de façon contiguë. Ils sont caractérisés par **Le nombre de dimensions** et **Le nombre d'éléments** dans chaque dimension:

Syntaxe :

<type> <identificateur> [dim_1] … [dim_n]

exemple :

int Tab [10] **tableau d'entiers à 1 dimension**

int T [10][5][3] **tableau d'entiers à 3 dimensions**

Remarque :

le premier élément d'un tableau est l'indice 0

```
1
2  int t[10]; // Déclaration d un tableau de 10 éléments
3  /* Le premier élément est t[0]
4     Le dernier élément est t[9]
5  */
6  void main(){
7       t[1]=2;
8       t[3]=31;
9       t[0]=25;
10 }
11
```

Variables et Constantes

Les tableaux : déclaration et initialisation

On peut initialiser un tableau lors de la déclaration

```
int Tableau[4] = { 12, 14, 5, 8 };

int Tableau[2][3] = { 10, 20, 30, 40, 50, 60 };
```

ou

```
int Tableau[2][3] = { {10, 20, 30} ,
                      {40, 50, 60} };
```

```
1
2  int t[10]={2,3,5,7,11}; //
3  /*  Déclaration d un tableau de 10 éléments
4      Le premier élément est t[0]
5      Le dernier élément est t[9]
6  */
7  void main(){
8      t[5]=2;
9      t[6]=31;
10     t[7]=25;
11 }
```

Variables et Constantes

Les chaines de caractères :

Elles sont vue en C comme un tableau.
La fin de la chaine est identifié par un caractère nul '\0' (chaine à zéro terminal):

- **Déclaration d'une chaine :**
  char MaChaine [10];
- **Déclaration d'une chaine avec son initialisation :**
  char MaChaine [10] = "Texte";
  ou char Chaine[ ]="bidule"; **(calcule automatique de la taille)**

0	1	2	3	4	5	6	7	8	9
T	e	x	t	e	\0	?	?	?	?
84	101	120	116	101	0	?	?	?	?

Variables et Constantes

Les chaines de caractères :

Attention pour stocker le mot "bonjour" (qui comprend 7 lettres) il faut un tableau de 8 char (7 + 1) afin de stocker le code nul '\0' en fin de chaine.

Remarque :
La présence du code nul en fin de chaine est un avantage pour la manipulation des tableaux contenant des chaines de caractères, car se code évite la gestion de la taille des tableaux (passage de paramètres dans les fonctions)

Remarque sur la syntaxe : Pour un char on l'affecte avec 'A'. Pour une chaine on l'affecte avec **"ABCD"**

On peut aussi initialiser une chaine de caractère à l'aide de

strcpy(Variable, Chaine)

cette fonction est déclaré dans le fichier string.h Ne pas oublier en début de code source

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

Attention on ne peut pas écrire :

MaChaine = "Toto";

Variables et Constantes

Les variables dans les blocs:
La position des déclarations des variables dans les différents bloc du programme détermine leur portée.

- **Déclaration avant le programme principal (main)**
  les variables sont globales : elles sont accessibles n'importe où dans le programme.
- **Déclaration dans un bloc { … }**
  les variables sont locales : elles n'existe que dans le bloc où elles ont été déclarées

Variables et Constantes


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

void main(){// programme principal
    i=5; // initialisation de la variable globale i
    printf("i du programme principal = %d\n",i);// affichage de i
    {// debut d un bloc
        int i; // Déclaration d'une variable locale i
        i=1; // initialisation de la variable locale i
        printf("i dans le bloc = %d\n",i);// affichage de i
    }// fin du bloc
    printf("i du programme principal = %d\n",i);// affichage de i
}// fin du programme principal

```

Fonctions d'affichage et de saisie

La fonction d'affichage:
Elle permet d'afficher des messages et/ou des valeurs de variables sous différents formats.

- **Syntaxe :**
  printf (<"Format">, Variable_1, … ,Variable_n);

- **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.

  printf("La valeur de %d au carré est égale à %d", i , i*i);

```
1  #include <stdio.h>
2  int i; // Déclaration d'une variable globale
3
4  void main(){
5      i=25; // initialisation de  i
6      printf("i en base 10 = %d\n",i);// affichage de i en décimal
7      printf("i en base 8 = %o\n",i);// affichage de i en octal
8      printf("i en base 16 = %x\n",i);// affichage de i en hexadécimal
9  }
10
```

Fonctions d'affichage et de saisie

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

Fonctions d'affichage et de saisie

Codes d'affichage:

On peut compléter les cotes d'affichage des variables pour les nombres signé ou flottants.

- **Un caractère de remplissage :**
  '0' au lieu de ' ' pour les numériques

- **Justifier à gauche:**
  '-' qui permet de justifier à gauche l'affichage

- **Affichage signe:**
  '+' qui permet de forcer l'affichage du signe

- **Nombre de chiffres affichés:**
  Syntaxe

  <Nb car affiché>.<Nb chiffre significatif>

Fonctions d'affichage et de saisie


```c
#include <stdio.h>
float i;
void main(){
    i=16.5;
    printf("%f\n",i);// affichage de i normal
    printf("%10f\n",i);// affichage de i avec 10 caractères
    printf("%10.2f\n",i);/* affichage de i avec 10 caractères
                            et 2 chiffres apres la virgule */
    printf("%-10.2f\n",i);/* affichage de i avec 10 caractères
                             et 2 chiffres apres la virgule
                              et à gauche*/
    printf("%+-10.2f\n",i);/* affichage de i avec 10 caractères
                              et 2 chiffres apres la virgule
                               et à gauche avec affichage du signe*/
    printf("%010.2f\n",i);/* affichage de i avec 10 caractères
                             et 2 chiffres apres la virgule
                              avec des zéro avant la valeur */
    }
```

Résultat

```
15.600000
 15.600000
      15.60
15.60
+15.60
0000015.60
```

Fonctions d'affichage et de saisie

Code de contrôle:

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

Fonctions d'affichage et de saisie

Autres fonctions d'affichage:
il existe deux autres fonctions d'affichage.

- **putchar : affiche un caractère**

  Syntaxe :

  putchar (<identificateur>);

  exemple :

```
putchar('C');
putchar('\0');
```

- **puts : affiche une chaine de caractères**

  Syntaxe :

  puts (<identificateur>);

  exemple :

```
puts('Un texte');
puts('Autre texte\0');

char message[10] = "bonjour";
puts(message);
```

Fonctions d'affichage et de saisie

La fonction de saisie scanf permet de saisir des valeurs de variables formatées à partir du clavier. Comme prinft elle est composée d'un format et des identificateurs des variables à saisir.

- **Syntaxe :**

```
scanf ( <"Format">, &Variable_1, … ,&Variable_n );
```

- **Remarque:**
Le symbole & est obligatoire devant les identificateurs car scanf attend des adresses et non des valeurs, sauf devant un identificateur de chaine de caractères qui est déjà une adresse.

Exemple :

```
scanf(" %d", &i );    // saisie d'un entier  et copie dans  i
```

Fonctions d'affichage et de saisie

Les codes d'entrées pour scanf:

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

Fonctions d'affichage et de saisie

Exemple de saisie d'une variable:

```c
#include <stdio.h>
#define PI 3.14159  // Déclaration de constante
float rayon, perimetre;
void main(){
    i=16.5;
    puts("Donner le rayon en mètre ?");
    scanf("%f",&rayon);// Saisie du rayon
    perimetre=2*PI*rayon; // Calcul du perimètre
    printf("\n\n Le perimètre est %f",perimetre);// affichage du perimètre
}
```

Exemple de saisie de plusieurs variables:

```c
#include <stdio.h>
float a, b, c, det;// Déclaration de variables réelles
void main(){
    puts("Donner les valeurs de a, b et c ?");
    scanf("%f %f %f",&a,&b,&c);// Saisie de a, b et c
    det=b*b-4*a*c; // Calcul du déterminant
    printf("\n\n Le determinant est %f",det);// affichage du déterminant
}
```

Fonctions d'affichage et de saisie


```c
#include <stdio.h>
char nom[10];// Déclaration de variable nom
void main(){
    puts("Quel est votre nom ?");
    scanf("%s",nom);// Saisie de a, b et c
    printf("\n\n%s",nom);// affichage du nom
}
```

Fonctions d'affichage et de saisie

Autres fonctions de saisie:

- **getchar() : saisie d'un caractère**
  syntaxe :

<identificateur> = getchar(void);

```c
#include <stdio.h>
char car1;// Déclaration d'un caractère
void main(){
    puts("Taper un caractère ?");
    car1=getchar();// Saisie d'un caractère
    putchar('\n'); // Changement de ligne
    printf("Le caractère saisie est %c",car1);// affichage du caractère
}
```

Fonctions d'affichage et de saisie

Autres fonctions de saisie:

- **gets() : saisie d'une chaine de caractères avec des espaces (impossible avec scanf)**

  syntaxe :

  gets(< identificateur de chaine >);

```c
#include <stdio.h>
char nom[20];// Déclaration d'une chaîne de 19 caractères
void main(){
    printf("Taper votre nom ?");
    gets(nom);// Saisie d'une chaîne
    putchar('\n'); // Changement de ligne
    printf("Votre nom est %s", nom);// affichage du nom
}
```

Fonctions d'affichage et de saisie

Autres fonctions liées à la saisie:

- **getch() : saisie d'un caractère à la "volé" pas de touche [entrée]**

  syntaxe : identificateur = getch(void);
  A utiliser par exemple dans la gestion de menu.

- **fflush(stdin) : vide le buffer clavier**
  Dans certains cas le buffer du clavier n'est pas complétement vide (exemple après la saisie d'une chaine avec un scanf). Cette fonction force la lecture du buffer du clavier jusqu'à rencontrer un caractère nul '\0'

  Pour info une autre méthode :

```
do
{
    c = getchar();
} while (c != '\n' && c != EOF);
```

Les Opérateurs

Ils régissent toutes les opérations ou transformations sur les valeurs des variables.

- **Opérateur d'affectation**
- **Opérateurs arithmétiques**
- **Opérateurs d'incrémentation et de décrémentation**
- **Opérateurs binaires**
- **Opérateurs combinés**
- **Opérateurs relationnels**
- **Opérateurs logiques**
- **Opérateurs de conversion de type**

Les Opérateurs

Opérateur d'affectation

C'est l'opérateur qui permet de modifier la valeur d'une variable.

Syntaxe : < identificateur > = < expression>

Exemples :

A = 2;	A ← valeur 2
A = B * C;	A ← résultat de l'opération
C = 'X';	C ← le caractère X

Les Opérateurs

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.

Les Opérateurs

Les Opérateurs d'incrémentation et de décrémentation :

++	Incrémente de 1
--	Décrémente de 1

Attention :

Si l'opérateur d'incrémentation ou de décrémentation est placé avant l'identificateur, alors la variable sera incrémentée ou décrémentée avant d'être utilisée.

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

Les Opérateurs

Les Opérateurs d'incrémentation et de décrémentation :

Exemple opérateur avant:

```
#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);
}
```

Résultat :

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

Les Opérateurs

Exemple opérateur (++) ou (--) après:

```
#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);
}
```

Résultat :

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

Les Opérateurs

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

Les Opérateurs

Les Opérateurs binaires :

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);
}
```

Résultat :

```
i(hex) = 20

i(hex) = 27

i(hex) = dc

i(hex) = 11

i(hex) = 30
```

Les Opérateurs

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

Les Opérateurs

Les Opérateurs combinés :

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);
}
```

Résultat :

```
i = 5

i = 10

i = 20
```

Les Opérateurs

Les Opérateurs relationnels :

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

FAUX : false correspond à **0**

VRAI : true correspond à **toute valeur ≠ 0**

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

Les Opérateurs

Les Opérateurs relationnels :

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);
}
```

Résultat :

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

Les Opérateurs

Les Opérateurs logiques :

Ils sont utilisés exactement comme les opérateurs relationnels.

!	Négation
&&	ET Logique
||	OU Logique

Les Opérateurs

Les Opérateurs logiques :

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);
}
```

Résultat :

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

Les Opérateurs

L'Opérateur de conversion de type :

Ils 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.

- **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 la variable à convertir.

Les Opérateurs

L'opérateur de conversion de type :

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);
}
```

Résultat :

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

Les Opérateurs

La priorité des opérateurs:

Le plus prioritaire → *Le moins prioritaire*

Le plus prioritaire ↓

() [] -> .
! - ++ -- - * & (case) sizeof
* / %
+ -
<< >>
== !=
&
-
|
&&
||
? :
= += -= *= /= %= >>= <<= &= |= ^=
,

Le moins prioritaire

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.

- **La structure <SI … ALORS …>**
- **La structure <SI … ALORS … SINON …>**
- **La structure choix**

Les Structures Conditionnelles

La structure <SI … ALORS …> :

Syntaxe :

if (condition) instruction;

```
#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);
}
```

Les Structures Conditionnelles

La structure <SI … ALORS …> :

Syntaxe :

```
if (condition) {
        instruction_1;
        . . .
        instruction_N;
}
```

```
#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");
     }
}
```

Les Structures Conditionnelles

La structure <SI … ALORS … SINON … > :

Syntaxe :

```
if (condition) {
        instructions;
} else {
        instructions;
}
```

```
#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");
     }
}
```

Les Structures Conditionnelles

La structure choix switch :

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;
                break;
        case valeur2 :
                instruction_2;
                break;
        case valeur3 :
                instruction_3;
                break;
        default :
                instruction_i;
                break;
}
```

```c
#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");
     }
}
```

Les Structures Conditionnelles

La structure choix switch :

Si deux valeurs correspondent à un même traitement on double le "case valeur :"

Syntaxe :

```
switch( identificateur )
{
        case valeur1 :
                instruction_1;
                break;
        case valeur2 :     ( même traitement pour valeur2 et valeur3)
        case valeur3 :
                instruction_3;
                break;
        default :
                instruction_i;
                break;
}
```

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 type de structures itératives la condition pourra être testée en début ou en fin de la structure.

- **La structure <TANT QUE … FAIRE …>**
- **La structure <FAIRE … TANT QUE …>**
- **La structure <POUR … FAIRE … JUSQU'A …>**

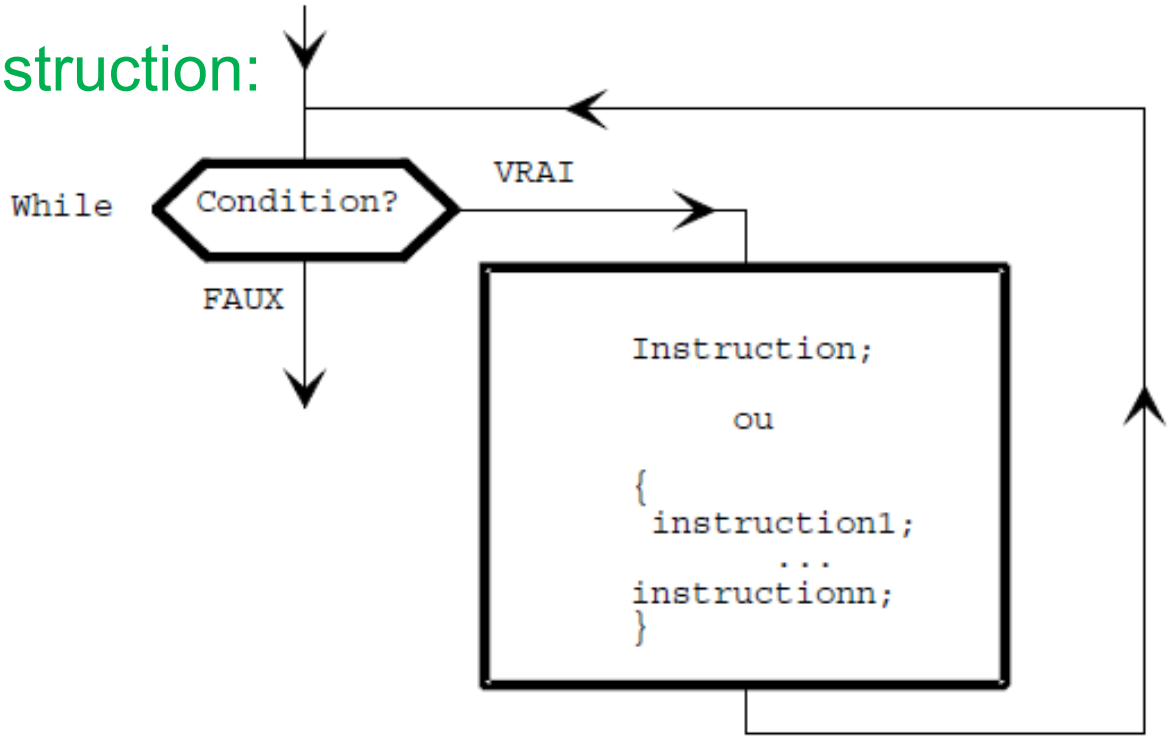
Les Structures Itératives ou boucles

La structure <TANT QUE … FAIRE …>

Dans cette structure la condition est testée au début.

Syntaxe :

```
while   ( condition )  instruction:
Ou
while   (condition)
{
        instructions1 ;
        . . .
        instructionN;
}
```



Les Structures Itératives ou boucles

La structure <TANT QUE … FAIRE …>

Exemples.

```
#include <stdio.h>

int i;

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

```
#include <stdio.h>

int i;

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

Les Structures Itératives ou boucles

La structure <FAIRE … TANT QUE …>

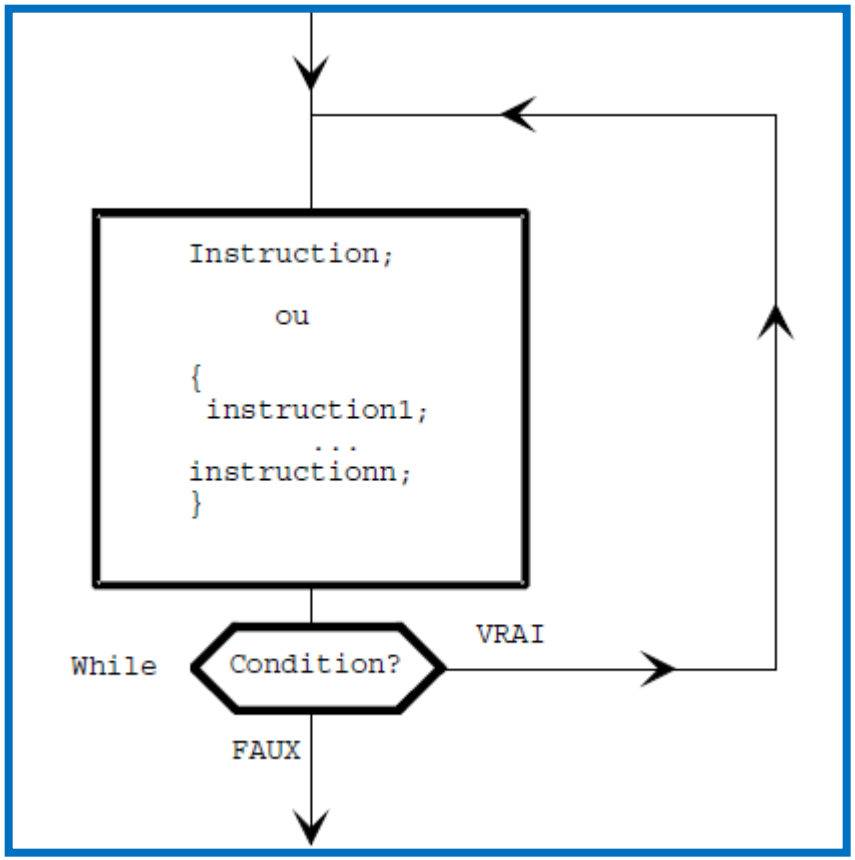
Dans cette structure la condition est testée à la fin.

Syntaxe :

```
do

        instruction ;

while   ( condition ) ;
```



Les Structures Itératives ou boucles

La structure <FAIRE … TANT QUE …>

Exemple :

```
#include <stdio.h>

int i;

void main()
{
     i=0;

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

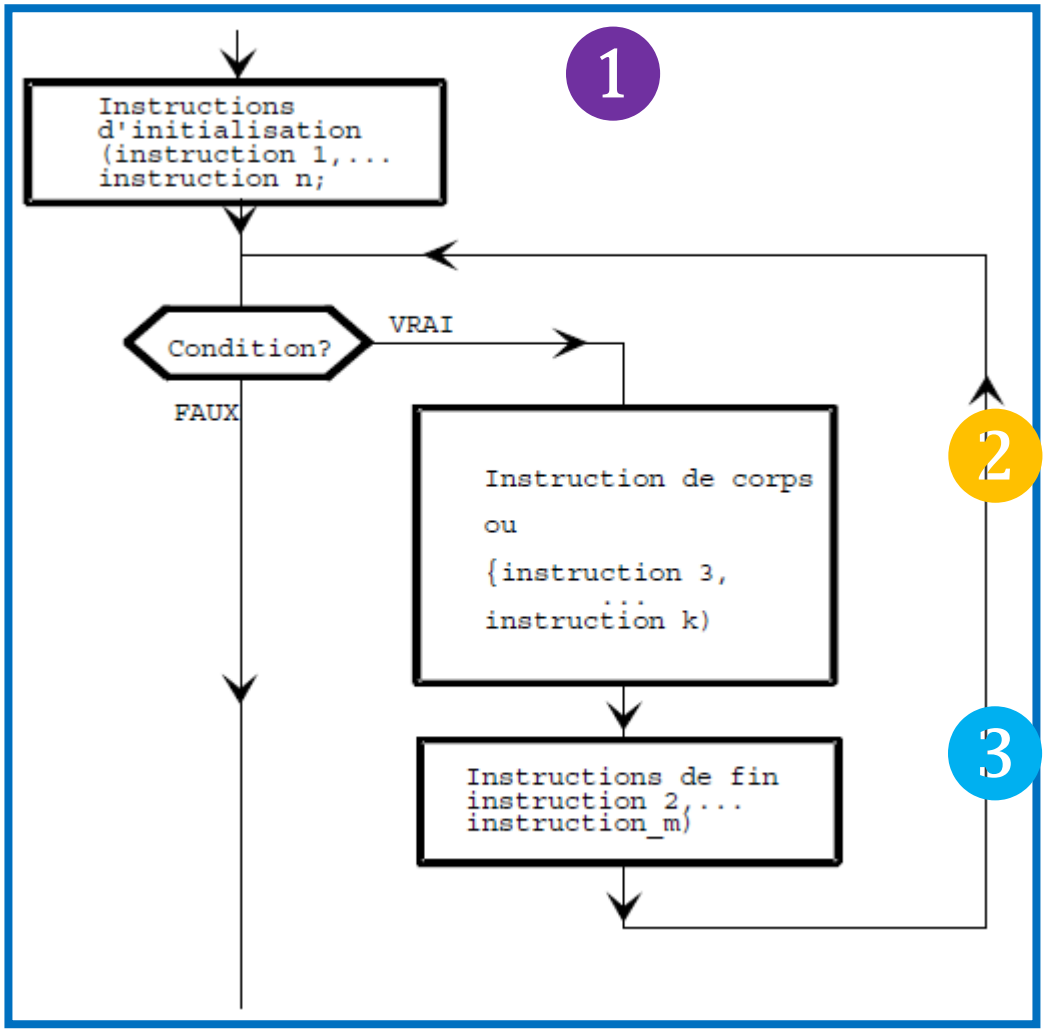
Les Structures Itératives ou boucles

La structure <POUR … FAIRE … JUSQU'A …>

Dans cette structure la condition est testée au début. Elle est composé de trois parties :

- partie 1 : Instruction ou plusieurs instructions qui sont exécutées une seule fois au début de la structure.
- Partie 2 : L'instruction ou le bloc d'instructions exécutées à chaque itération.
- Partie 3 : L'instruction ou plusieurs instructions qui sont exécutées à la fin de chaque itération

Les Structures Itératives ou boucles



Les Structures Itératives ou boucles

La structure <POUR … FAIRE … JUSQU'A …>

Syntaxe :

```
for (inst_1, … , inst_N d'init ; condition ; instr_2 , … , intr_M de fin) {
        instruction(s) ;
 }
```

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

int main(int argc, char *argv[])
{
  int I,A;
  for ( I=0, A=10 ; I<10 ; I++, A-- )
      {
      printf("I = %d et A = %d\n",I,A);
      }
  system("PAUSE");
  return 0;
}
```

résultat

```
I = 0 et A = 10
I = 1 et A = 9
I = 2 et A = 8
I = 3 et A = 7
I = 4 et A = 6
I = 5 et A = 5
I = 6 et A = 4
I = 7 et A = 3
I = 8 et A = 2
I = 9 et A = 1
```

Les Structures Itératives ou boucles

La structure <POUR … FAIRE … JUSQU'A …>

```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);
}
```

```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++;
     }
}
```