Le Langage C

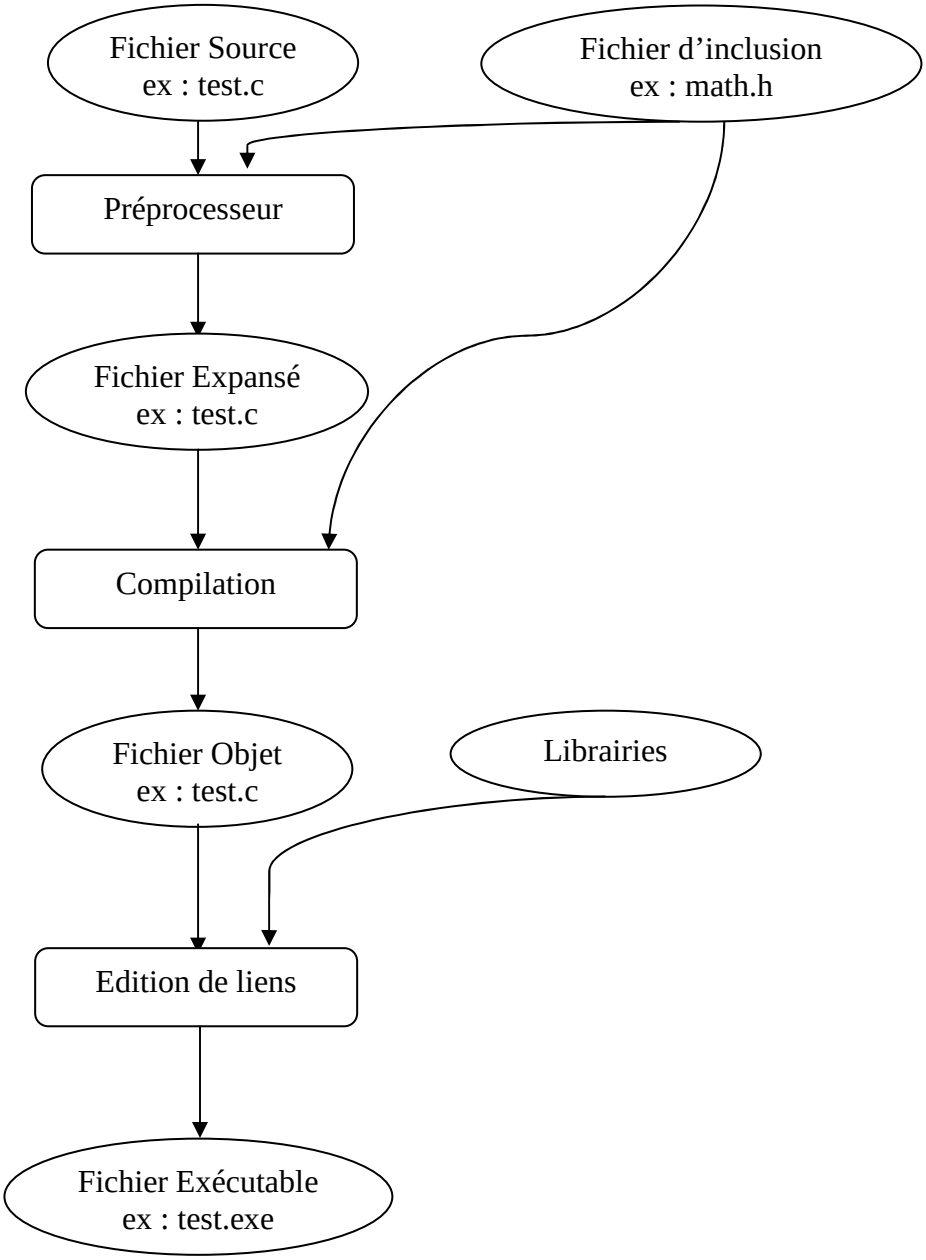
Le langage C a été crée en 1972 par Denis Ritchie avec comme objectif : écrire un système d'exploitation (UNIX). Sa définition été réalisée en 1978 par Kernighan et Ritchie dans un livre intitulé : « The C Programming Language ».

Le succès du langage a conduit à sa normalisation, en premier par l'ANSI (American National Standard Institute) (1989), puis par l'ISO (International Standardization Organization) en 1990.

Le langage C est un langage évolué, structuré et assez proche du langage machine. Il est destiné à des applications de contrôle de processus (gestion d'entrées/sorties, applications temps réel,…). Il fait appel généralement à des bibliothèques fournies avec le compilateur (ex : math.h, stdio.h).

1- Etape de mise en place d'un programme exécutable

Tout programme écrit en langage évolué forme un texte qu'on nomme un programme source. En C, le programme source peut être découpé en un ou plusieurs fichiers sources (modules). Les étapes de création d'un programme exécutable à partir d'un programme source se résument en :



Edition du code
L'édition du programme (saisie) consiste à saisir, à l'aide d'un éditeur, le texte du programme pour créer le programme source qui porte l'extension « .c ».

La compilation
Cette étape consiste à traduire le programme source en langage machine. Elle est réalisée par un logiciel nommé : compilateur. Le rôle du compilateur est d'indiquer les erreurs de syntaxe, et ignore les fonctions bibliothèques appelées par le programme. Il génère un fichier binaire appelé fichier objet : « .obj ».

Edition de liens
L'édition de liens consiste à réunir les différents modules objet et les fonctions des bibliothèques standards afin de créer un programme exécutable.

2- Programme en C

Voici un exemple de programme écrit en C :

```
#include <stdio.h>

int x ;

void main(void)
{
  int y ;
  printf(″Bonjour \n″) ;
}
```

- La première ligne du programme : « #include <stdio.h> » est une directive qui est prise en compte avant la compilation du programme (lors de la phase du préprosseur).

- La ligne « void main(void) » est l'en-tête de la fonction main. Les instructions délimitées par les accolades « { » et « } » forment le bloc de la fonction. La fonction main représente le programme principal (cas de BEGIN et END du pascal).

- Les instructions : « int x » et « int y » sont des déclarations de variables de type entier. Les variables déclarées entre les directives et la fonction main sont des variables globales (cas de int x), alors que les variables déclarées à l'intérieur de la fonction main sont des variables locales (cas de int y).

- La fonction prédéfinie « printf » permet d'afficher la chaîne de caractères « Bonjour » délimitée par les guillemets.

3- Règles d'écriture d'un programme en C

L'écriture d'un programme en C exige un certain nombre de règles générales, dont nous citons :

3.1- Les identificateurs

Dans un programme, les éléments (variables, fonctions, types,…) sont désignés par un nom qu'on appelle identificateur. Cet identificateur est formé d'une suite de caractères : lettres, chiffres et le caractère « souligné » (_). Le premier caractère doit être différent d'un chiffre. Le langage C est sensible à la casse.

Exemple : nom address_pro _somme

3.2- Les mots clés

Le langage C est un langage à mots-clés qui sont des mots réservés pour le langage et ne peuvent pas être employés comme identificateurs.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

3.3- Les séparateurs

Dans un programme en C, la syntaxe impose un séparateur ; un des caractères (que se soit des opérateurs : +, -, *, =, ==, +=,… ou les caractères de ponctuations : () [] {} , ; : …) doit être utiliser pour séparer les identificateurs. Dans le cas contraire, ces identificateurs doivent être impérativement séparés par un Espace.

3.4- Les commentaires

Comme tout langage, le C autorise l'utilisation de commentaires. Il s'agit de caractères placés entre les symboles /* et */

Exemple :

```
/* commentaire s'étendent
sur plusieurs lignes
du programme * /
```

- Si le commentaire s'étend sur une seule ligne, le symbole // est utilisé.

  Exemple : int x ; //déclaration d'une variable de type entier.

4- Les types de base

Généralement, la manipulation d'une information fait intervenir la notion de type. C'est-à-dire la façon dont elle est coder en mémoire. En C, nous distinguons trois types :

- **Les entiers**

Le mot clé désignant les entiers est **int**. Les types entiers sont caractérisés par deux paramètres :
- La taille (nombre de bits) de l'emplacement mémoire utilisé pour les représenter ;*
- L'attribut précisant si le nombre est signé (entiers relatifs), ou non signé (entiers naturels).

Type	Description	Taille mémoire
int	entier standard signé	4 octets : $-2^{31} \leq n \leq 2^{31}-1$
unsigned int	entier standard positif	4 octets : $0 \leq n \leq 2^{32}-1$
short	entier court signé	2 octets : $-2^{15} \leq n \leq 2^{15}-1$
unsigned short	entier court positif	2 octets : $0 \leq n \leq 2^{16}-1$

- **Les réels**

Pour représenter les réels en C, nous distinguons deux types : le type **float** (réel standard) sur 4 octets et le type **double** (réel double) sur 8 octets.

- **Les caractères**

Le mot clé désignant les caractères est **char**. Il peut contenir le code de n'importe quel caractère de l'ensemble des caractères utilisés sur la machine. Nous distinguons deux types de caractères :

1- Les caractères imprimables :

Ce type de caractères s'écrit entre apostrophes.

Exemple : 'a' 'Y' '+' '$' '0' '<'

2- Les caractères avec séquence d'échappement :

Ce type de caractères (non imprimables) possède une représentation conventionnelle dite séquence d'échappement utilisant le caractère : \

Séquence d'échappement	Signification
\n	Saut de ligne
\\	\
\'	'
\ ''	''
\ ?	?.

4.2- Les constantes

En langage C, il y a deux façons de donner un nom à une constante :

1- En utilisant les possibilités du préprocesseur :

#define identificateur valeur

Le préprocesseur remplace, dans toute la suite du code source du programme, toute occurrence de 'identificateur' par 'valeur'.

Exemple : #define PI 3.14

2- En utilisant des énumérations :

enum {liste d'identificateurs}

Cette commande permet la déclaration d'un ensemble de constantes.

Exemple :

enum {SAMEDI, DIMANCHE, LUNDI, MARDI, MERCREDI, JEUDI, VENDREDI} ;

enum {X1 = 1, X2 = 2, X3 = 3} ;

5- Les opérateurs

Nous distinguons :

5.1- Les opérateurs arithmétiques

Sont les opérateurs classiques « binaires » (c'est-à-dire qui porte sur deux opérandes) : l'addition (+), la soustraction (-), la multiplication (*), la division (/), le modulo (%) (reste de la division) et l'opérateur « unaire » (c'est-à-dire qui porte sur un seul opérande) qui correspond à l'opposé (exemple : -2, -1,..).

5.2- Les opérateurs relationnels

Le langage C permet, comme tout langage, de comparer des expressions à l'aide d'opérateurs relationnels (ou de comparaison) qui sont : <, >, <=, >=, == (égal), != (différent).

Le résultat d'une comparaison est un entier qui vaut :
- 0 : si le résultat de la comparaison est faux.
- 1 : si le résultat de la comparaison est vrai.

Exemple :

```
int a = 5, b = 1 ;
…
2 * a > b +5 ;
```

Le résultat est : 1

5.3- Les opérateurs logiques

Ce sont les opérateurs logiques classiques : ET logique (&&), OU logique (||) et le non logique (!).

Exemple :

```
(a < b) && (c < d)
```

- Le résultat est égal à 1 si (a < b)=1 et (c < d)=1 ;
- Le résultat est égal à 0 dans le cas contraire.

5.4- Les opérateurs d'affectation et d'incrémentation

L'opérateur d'affectation (=) fait partie des opérateurs du langage C. Il permet l'affectation d'une valeur (ou résultat d'une expression arithmétique ou logique) à une variable.

Exemple :

```
i =  j + 1 ;
i = 3 ;
i = (j = k) + 1 ;
```

Les instructions d'incrémentation et de décrémentation peuvent être remplacées par des opérateurs « unaire » portant sur la variable concernée. Nous distinguons les deux opérateurs : ++ et --.

L'opérateur ++ (ou --) est un opérateur de :
- Pré-incrémentation s'il est placé à gauche de la variable ;
- Post-incrémentation s'il est placé à droite de la variable.

Exemple :

```
int n, i = 5 ;
……
n = ++i – 5 ; // Résultat : i = 6, n = 1.

n = i++ – 5 ; // Résultat : i = 6, n = 0.
```

6- Les entrées/sorties standards

Ce type d'entrées/sorties (qui est une partie des opérations d'entrées/sorties) regroupe toutes les fonctions de communications avec l'utilisateur à savoir : printf, scanf, getchar, putchar, gets et puts.

6.1- La fonction printf

La fonction printf envoie sur écran une suite de caractères. La forme générale de printf est :

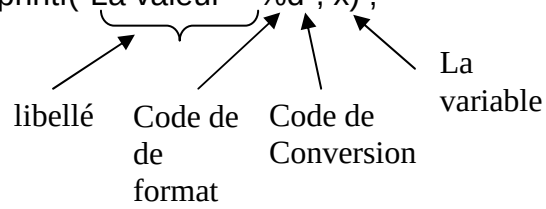
printf (Argument 1, Argument 2**)**

- Argument 1 (appelé : **format**) : est une suite de caractères qui sert au formatage des variables. Il contient :
  - Des caractères à afficher tels quels à l'écran (appelés : libellés) ;
  - Des codes de format désignés par **%** et suivi de « code de conversion » qui précise le type de la variable à afficher.
- Argument 2 : fournit les variables à formater.

Code de conversion	Explications
c	Variable de type caractère (char)
x	Variable de type hexadécimal
d	Variable de type entier
u	Variable de type entier non signé
ld	Variable de type long
lu	Variable de type long non signé
f	Variable de type double ou float (format : x.xxx)
e	Variable de type double ou float (format : exponentielle)
s	Variable de type chaine de caractères

Exemple :

```
printf(″La valeur = %d″, x) ;
```



6.2- La fonction putchar

Cette fonction permet d'afficher un seul caractère. Si la variable c est de type caractère : putchar(c) est équivalente à printf(″%c″, c) ;

Exemple :

```
char x = 'A' ;
…..
putchar(x) ; // affiche A à l'écran
```

6.3- La fonction puts

Cette fonction permet d'afficher un texte. puts(″Bonjour″) est équivalente à printf(″Bonjour \n″) ;

6.4- La fonction scanf

La fonction scanf lit une variable sur le clavier. Les variables à saisir seront formaté par le premier argument « format ».

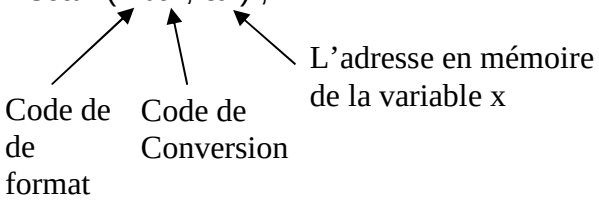
La forme générale de scanf est : **scanf(**Argument 1, Argument 2**)**

- Argument 1 (appelé : **format**) : contient le « code de format » **%** et le code de conversion qui précise le type de la variable à récupérer ;

- Argument 2 : indique l'emplacement (l'adresse) de la variable en mémoire centrale désigné par le symbole **&** où sera rangée la variable saisie.

Exemple :

```
scanf(″%d″, &x) ;
```



6.5- La fonction getchar

Cette fonction permet la saisie d'un caractère (de type char). Si la variable c est de type char : c = getchar() est équivalente à scanf(″%c″, &c).

6.6- La fonction gets

Cette fonction permet de lire une chaine de caractères à partir du clavier.

Exemple :

```
char  ville [ 20 ] ; // déclaration d'un tableau de caractères
……..
gets(ville) ; // récupère la chaine saisie par l'utilisateur et la mettre dans le tableau de caractères « ville ».
```

7- Les instructions de contrôle

Comme tout langage, le langage C dispose d'instructions, nommées : instructions de contrôle permettant de se comporter en fonction d'une condition (cas des instructions de choix : instructions **if** et **switch**) ou répéter plusieurs fois un ensemble d'instructions (cas des boucles). Toutefois, il dispose des instructions de branchement inconditionnels : goto, break, continue et return.

7.1- L'instruction if

L'instruction **if** permet de programmer une structure de choix où il y a lieu de choisir entre deux instructions suivant la valeur d'une condition. Elle a deux format :

- Sans le « else » : if (expression)
  Instruction 1 ;
- Avec le « else » : if (expression)
  Instruction 1 ;
  else
  Instruction 2 ;

Exemples :

1- Une seule instruction :

```
if ( a > b )
        Max = a ;
else
        Max = b;
```

2- Avec un bloc d'instructions :

```
if ( a > b )
{
        Max = a ;
        printf(″a c'est le maximum″) ;
}
else
{
        Max = b;
        printf(″b c'est le maximum″) ;
}
```

7.2- L'instruction switch

Le langage C offre une instruction **switch** qui est un if généralisé.

Syntaxe :

```
switch ( expression )
{
        case constante_1 : liste d'instructions 1 ; //facultatif
                        break ; // facultatif
        …….
        case constante_N : liste d'instructions N ; //facultatif
                        break ; // facultatif
        default : liste d'instructions ; //facultatif
}
```

Exemple :

```
int couleur ;
……..
puts(″Donner un entier″) ;
scanf(″%d″, &couleur) ;
```

```
switch (couleur)
{
        case 0 : puts(″Noir″) ; break ;
        case 1 : puts(″Blanc″) ;
        case 2 : ;
        default : puts(″aucun choix de couleur″) ;
}
```

7.3- Les boucles

Le langage C dispose de trois instructions de boucles :

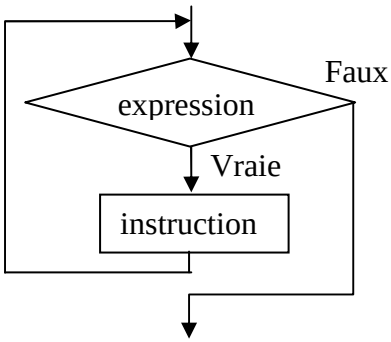
- L'instruction **while** (Tantque…Faire).
- L'instruction **do…while**.
- L'instruction **for** (Pour…allant de…à…faire).

7.3.1- L'instruction while

Syntaxe :

while (expression **)**
instruction ;

Cette instruction répète l'exécution de « instruction » tant que la valeur de « expression » est vraie.



Exemples :

1- Avec une seule instruction

```
int i = 0 ;
……..
while ( i < 10 )
i = i + 1 ;
```

2- Avec un bloc d'instructions

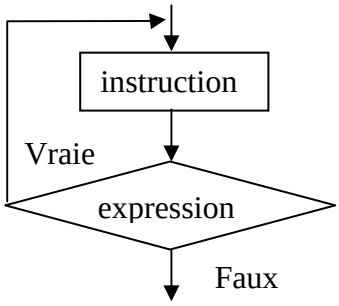
```
int i = 0 ;
……..
while ( i < 10 )
{
printf(″La valeur de i est : %d \n″, i) ;
i = i + 1 ;
}
```

7.3.2- L'instruction do…while

Syntaxe :

do
instruction ;
while (expression **) ;**

Cette instruction exécute « instruction » puis elle évalue « expression ». Si cette dernière est vraie, elle ré-exécute « instruction ».



Exemples :

1- Avec une seule instruction

```
int i = 0 ;
……..
do
i = i + 1 ;
while ( i < 10 ) ;
```

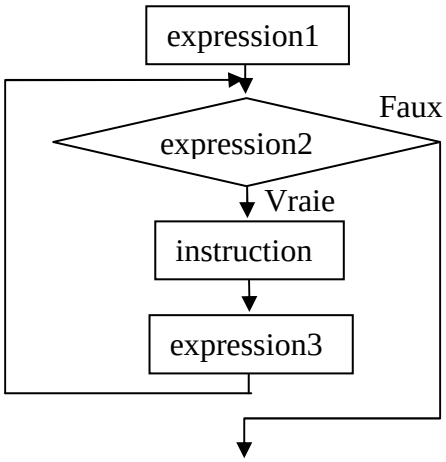
2- Avec un bloc d'instructions

```
int i = 0 ;
……..
do
{
printf(″La valeur de i est : %d \n″, i) ;
i = i + 1 ;
}
while ( i < 10 );
```

7.3.3- L'instruction for

Syntaxe : **for (** expression1 ; expression2 ; expression3 **)**
instruction ;

Cette instruction évalue « expression1 », puis évalue « expression2 ». Si cette dernière est vraie, elle ré-exécute « instruction » et évalue « expression3 ».



Exemples :

1- Avec une seule instruction

```
int i ;
……..
```

```
for ( i = 0 ; i < 10 ; i ++)
printf(″Bonjour \n″) ;
```

2- Avec un bloc d'instructions

```
int i, j ;
……..
for ( i = 0 ; i < 10 ; i ++)
{
        j = i * 5;
        printf(″i multiplié par 5 donne: %d \n″, j) ;
}
```

7.4- Les instructions de branchement inconditionnel

Les instructions de branchement inconditionnel sont les instructions qui ont pour effet de transférer le contrôle d'exécution d'une partie à une autre dans un programme. En C, nous distinguons : break, goto, continue et return.

7.4.1- L'instruction break

Cette instruction peut être utilisée :
- Dans une instruction switch : pour mettre fin à l'exécution de l'instruction
- Dans une boucle (for, while ou do…while) : pour provoquer la sortie de la boucle.

Exemple :

```
int  i, n ;
……..
for ( i = 0 ; i < 100 ; i++)
{
        printf(″Donnez un nombre″) ;
        scanf(″%d″, &n) ;

        if ( n == 19 )
           break ;
}
```

7.4.2- L'instruction continue

Cette instruction est utilisée seulement dans une boucle. A la différence de break, qui met fin à une boucle, continue permet de forcer le passage à l'itération suivante.

Exemple :

```
int  i, n ;
……..
for ( i = 0 ; i < 100 ; i++)
{
        printf(″Donnez un nombre″) ;
        scanf(″%d″, &n) ;

        if ( n < 0 )
           continue ;
```

```
        printf(″Le nombre est positif″) ;
}
```

7.4.3- L'instruction goto

Syntaxe :

```
        goto étiquette ;
```

Cette instruction à pour effet de provoquer un branchement à l'instruction portant l'étiquette.

Remarque : l'étiquette est un identificateur suivi de deux points (:) et qui précède l'instruction.

Exemple :

```
int  i, n ;
……..
for ( i = 0 ; i < 100 ; i++)
{
        printf(″Donnez un nombre″) ;
        scanf(″%d″, &n) ;

        if ( n < 0 )
            goto fin ;

        printf(″Le nombre est positif″) ;
}

fin : printf(″Le programme est sorti de la boucle car l'utilisateur a introduit un nombre négatif″) ;
```