UNIVERSITÉ DE POITIERS

INSTITUT UNIVERSITAIRE DE TECHNOLOGIE
6, allée Jean Monnet
BP 389

ELEMENTS DE BASE DU LANGAGE C



1. Introduction

L'architecture minimum du texte source d'un programme contenant la fonction principale (première fonction exécutée dans un programme) est la suivante :

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

int main ()
{
// Bloc déclaratif
   .
   .
/* Bloc exécutable */
   .
   .
   .
   return 0 ;
}
```

Directives de compilations permettant d'inclure des objets de la bibliothèque d'entrées-sorties et de la bibliothèque standard

Les symboles **//** sont suivis d'un commentaire

Les symboles **/*** ……. ***/** encadrent un commentaire.

Chaque ligne d'instruction se termine par le caractère **;**

2. Les différents types de variable prédéfinis

2.1. Entier

TYPE	DESCRIPTION	TAILLE MEMOIRE
int	**Entier standard signé**	4 octets : $-2^{31} \leq n \leq 2^{31}-1$
unsigned int	Entier standard positif	4 octets : $0 \leq n \leq 2^{32}$
short	Entier court signé	2 octets : $-2^{15} \leq n \leq 2^{15}-1$
unsigned short	Entier court positif	2 octets : $0 \leq n \leq 2^{16}-1$

2.2. Réel

TYPE	DESCRIPTION	TAILLE MEMOIRE
float	Réel standard	4 octets
double	Réel double précision	8 octets

2.3. Caractère

TYPE	DESCRIPTION	TAILLE MEMOIRE
`char`	Caractère signé	1 octet : $-128 \leq n \leq 127$
`unsigned char`	Caractère non signé	1 octet : $0 \leq n \leq 255$

En langage C, le type **`char`** permet de stocker des caractères mais peut aussi être considéré comme un cas particulier d'entier codé sur 1 octet.

Remarque :

Un caractère peut aussi être représenté par sa valeur exprimée dans la table ASCII en utilisant une notation en **base hexadécimale**. Cette valeur est précédée par une barre de fraction inversée (*antislash*) à l'intérieur de "quotes" (*apostrophes*).

Exemple : **`'\x41'`** désigne le caractère dont le code **hexadécimal** est **41**, donc, il s'agit du caractère **`'A'`** .

Caractères spéciaux (dont le code ASCII est inférieur à la valeur décimale 32) :

Ces caractères spéciaux ont des désignations standards ainsi qu'un certain nombre d'abréviations utilisables :

Désignation standard	Code ASCII (décimal)	Code ASCII (hexa)	Abréviations	Fonction spéciale
`HT`	**`9`**	**`'\x9'`**	**`'\t'`**	Tabulation 8 colonnes
`LF`	**`10`**	**`'\xA'`**	**`'\n'`**	Saut de ligne et retour en début de ligne.
`CR`	**`13`**	**`'\xD'`**	**`'\r'`**	Retour en début de ligne.

Le caractère **`NUL ('\x0')`** permet la gestion des chaînes de caractères**.**

3. Déclaration de variables

3.1. Déclaration de variables de types simples prédéfinis

```
int i, j, k ;    // Entiers
float a, b, c ; // Réels
char lettre1, lettre2 ; // Caractère
```

Il est possible d'initialiser les variables dans la zone des déclarations :

```
int i=0, j=1, k=10 ;              // Entiers
float a=3.5, b=6.5, c=3.2 ;       // Réels
char lettre1='A', lettre2='a' ;  // Caractères
```

3.2. Déclaration de variables de type tableau

Tableaux à une dimension :

Déclaration :

type nom [*dim*] ;

Exemple :

```
int compteur [10] ; /* compteur est une variable
                       tableau d'au plus 10 entiers */
float nombre [20] ; /* nombre est une variable
                       tableau d'au plus 20 réels */
char mot [30]     ; /* mot est une variable tableau
                       d'au plus 30 caractères */
```

Initialisation dans la zone des déclarations :

```
int compteur [10] = {1, 2, 3, 4, 5} ;
float nombre [20] = {1.1, 1.2, 1.3, 1.4};
char mot [30] ="Bonjour"  ;
```

Remarque : dans le cas de tableaux de nombres (entiers ou réels), les éléments restant qui n'ont pas été spécifiés dans la liste d'initialisation sont **initialisés à 0**. Dans le cas de tableau de caractères (chaîne de caractères), le caractère **NUL ('\x0'** ou **'\0')** est rajouté après le dernier caractère.

Tableaux à plusieurs dimensions :

Déclaration :

type nom **[***dim1***][***dim2***] ;**

Exemple :

```
float tab[2][3] ; /* tab est une variable tableau
                     de 2 lignes et 3 colonnes de réels */
```

Initialisation dans la zone des déclarations :

```
float tab[2][3] = {{1.1, 1.2, 1.3},
                   {2.1, 2.2, 2.3} } ;
```

4. Déclaration de constantes

1ère méthode : déclaration d'une **variable** dont la valeur sera **constante** pour tout le programme :

```
const int N_MAX = 100 ;
```

2ème méthode : définition d'un symbole à l'aide de la directive de compilation **#define** :

```
#define N_MAX 100
```

La 2ème méthode présente l'avantage de ne pas réserver de place en mémoire.

5. Instructions d'entrées-sorties

5.1. Introduction

Les fonctions décrites dans ce paragraphe ont leurs prototypes écrits dans le fichier **`stdio.h`** de la bibliothèque d'entrées-sorties (**st**an**d**ard **i**nput **o**utput)

5.2. Sorties (Ecritures)

5.2.1. Sorties quelconques (printf)

La fonction **`printf`**, permet l'écriture, en particulier sur le moniteur (*instruction* **Ecrire** *en langage algorithmique*).

Exemple : affichage d'un texte :

```
printf("Bonjour") ;  /* pas de retour à la ligne du curseur
                        après l'affichage */
printf("Bonjour\n"); /* affichage du texte, puis retour à
                        la ligne du curseur (à l'aide du
                        caractère \n) */
```

La fonction **`printf`** transforme des objets d'une représentation manipulable par la machine (vision machine), en une représentation constituée **d'une chaîne de caractères** (vision humaine). Pour réaliser ces transformations cette fonction est guidée par des ***formats de sortie*** qui décrivent le type des objets manipulés (vision interne) et la représentation en chaîne de caractères cible (vision externe).

Pour **`printf`**, un ***format de sortie*** est **une suite de caractères précédée de %** dans laquelle sont insérés les caractères représentant la variable à écrire.

Par exemple, un ***format*** du type **%d** signifie que la variable est du type **int** (entier).

Syntaxe générale d'écriture du contenu d'une variable :

```
printf("format",nom_de_variable) ;
```

Exemple : affichage du contenu d'une variable de type entier :

```
printf("%d",i) ;  /* affichage du contenu de i seulement */
```

Il est possible d'insérer le format à l'intérieur d'un texte :

```
printf("Je vous parle pour la %deme fois !",i) ;
```

Si **`i`** contient la valeur **3** le message affiché à l'écran sera :

```
Je vous parle pour la 3eme fois !
```

Le tableau suivant donne un résumé des déclarations de variables et des formats nécessaires à leurs manipulations :

Déclaration	Syntaxe	Format externe
`int i ;`	`printf("%d",i) ;`	Décimal
`int i ;`	`printf("%x",i) ;`	Hexadécimal
`int i ;`	`printf("%X",i) ;`	
`float v ;`	`printf("%f",v) ;`	Point décimal
`float v ;`	`printf("%e",v) ;`	Exponentiel (virgule flottante)
`float v ;`	`printf("%E",v) ;`	
`char car ;`	`printf("%c",car) ;`	Caractère
`char mot[10] ;`	`printf("%s",mot) ;`	Chaîne de caractères
`int i ;` `float v ;`	`printf("%p",&i) ;` `printf("%p",&v) ;`	Adresse

Il est possible de réserver le nombre de caractères dédiés à l'affichage du contenu des valeurs.

La première valeur numérique située après **%** indique le nombre total de caractères réservés pour afficher le résultat. (*Si ce nombre ne suffit pas, printf rajoute automatiquement le nombre de caractères nécessaires*).
Exemple :

Instructions

```
// si i (int) contient 12
printf("i = %d",i) ;
printf("i = %2d",i) ;
printf("i = %3d",i) ;
printf("i = %6d",i) ;
printf("i = %x",i) ;
printf("i = %2X",i) ;
```

```
/* si v (float) contient
   0.03386 */
printf("v = %f",v) ;
printf("v = %.2f",v) ;
printf("v = %.3f",v) ;
printf("v = %8.3f",v) ;
printf("v = %e",v) ;
printf("v = %12.2e",v) ;
printf("v = %13.2E",v) ;
```

Sortie écran correspondante

```
i = 12
i = 12
i =  12
i =     12
i = c
i =  C
```

```
v = 0.033860
v = 0.03
v = 0.034
v =    0.034
v = 3.386000e-002
v =    3.39e-002
v =     3.39E-002
```

Instructions	Sortie écran correspondante

```
// si c (char) contient  'A'
printf("c = %c",c) ;
printf("c = %2c",c) ;
```

```
c = A
c =  A
```

```
/* si mot est une chaîne de
caractères contenant "Hello"
*/
printf("mot = %s",mot) ;
printf("mot = %8s",mot) ;
```

```
mot = Hello
mot =   Hello
```

Il est enfin possible d'afficher une liste de variable :

Exemple :

```
printf("Les valeurs des variables sont les suivantes:\n) ;
printf (" i = %2d \n v = %8.3f \n c = %c \n mot = %s \n",
          i,v,c,mot) ;
```

Résultats à l'écran :

```
Les valeurs des variables sont les suivantes:
 i = 12
 v =    0.034
 c = A
 mot = Hello
```

ATTENTION : LE COMPILATEUR N'INDIQUE PAS D'ERREUR SI LE FORMAT NE CORRESPOND PAS AU TYPE DE LA VARIABLE CORRESPONDANT

Pour pouvoir écrire les caractères **"**,**'** et **** dans une chaînes de caractères, il suffit d'utiliser leur abréviations : **\"**,**\'** et ****.

Exemple :

```
printf ("\"") ;
printf ("\'") ;
printf ("\\") ;
```

Sorties à l'écran

```
"
'
\
```

5.2.2. Ecriture d'un caractère

La fonction **putchar** permet d'afficher un caractère avec un saut de ligne.

Syntaxe :

```
char c ;
putchar (c) ;
```
équivalent à **printf("%c\n",c) ;**

5.2.3. Ecriture d'une chaîne de caractères

La fonction **puts** permet d'afficher une chaîne de caractères avec un saut de ligne.

Syntaxe :

```
char mot[80] ;
puts (mot) ;
```
équivalent à **printf("%s\n",mot) ;**

5.3. Entrées standard à partir du clavier (Lectures)

5.3.1. Saisie d'une valeur numérique

La fonction générale de lecture d'un nombre à partir du clavier (*instruction* **Lire** *en langage algorithmique*) est assurée par la fonction **`scanf`**

Un argument fondamental de la fonction **`scanf`** est le **format** qui permet d'assurer le codage binaire à partir des caractères stockés (en code ASCII) dans le **buffer clavier** en fonction du type de la variable dans laquelle on souhaite ranger le résultat.

Par exemple, une variable de type entier (**`int`**) devra être saisie avec un format **`%d`**.

```
scanf("%d",&i) ;
```

Syntaxe générale :

`scanf ("`*format*`", `*adresse de la variable*`) ;`

Lors de l'exécution de **`scanf`**, on doit passer la variable (paramètre effectif) par adresse !

(Ne pas oublier le symbole **`&`** désignant **l'adresse** de la variable !)

Le tableau suivant donne un résumé des déclarations de variables et des formats nécessaires à leur lecture :

Déclaration	Syntaxe	Format externe
`int i ;`	`scanf("%d",&i) ;`	décimal
`int i ;`	`scanf("%x",&i) ;`	Hexadécimal
`float v ;`	`scanf("%f",&v) ;`	Réel

5.3.2. Saisie d'un caractère (`getche()`)

La fonction **`getche`** permet la lecture directe d'un caractère saisi au clavier (sans passer par le buffer clavier). La saisie est terminée dès l'entrée du caractère au clavier. Le caractère saisi au clavier est visualisé à l'écran (**e**cho). Cette fonction nécessite d'inclure la librairie **`<conio.h>`**.

Syntaxe :

```
char c ;
c = getche () ;
```

5.3.3. Saisie d'un caractère (`getch()`)

La fonction **`getch`** est identique à la fonction **`getche`**, simplement, le caractère saisi au clavier n'est pas visualisé à l'écran.

Syntaxe :

```
char c ;
c = getch() ;
```

5.3.4. Saisie d'une chaîne de caractères

La fonction **gets** permet la lecture d'une chaîne de caractères au clavier. Tous les caractères saisis sont extraits du tampon jusqu'au caractère **LF** et qui est converti en caractère **NUL** (fin de chaîne de caractères)**.**

Syntaxe :

```
char mot[80] ;
gets (mot) ;
```

6. Instructions

Une instruction est :

– soit une instruction simple,

– soit un instruction composée.

Une instruction simple est toujours terminée par un **;**.

Les instructions composées sont contenues **entre deux accolades** :

```
{
   instruction_1 ;
   instruction_2 ;
}
```

7. Opérateurs

7.1. Opérateurs arithmétiques

Ces opérateurs permettent d'effectuer les opérations mathématiques élémentaires.

Opérateur	Fonction
+	Addition
-	Soustraction
*	Multiplication
/	Division (pour les réels) ou division entière (pour les entiers)
%	Reste de la division entière

Exemple :

```
float a=5.5,b ;
int p=7, d = 3, q, r ;

b = a*4+5.5/a ; // b contient 23.0
q = p / d     ; /* q contient le résultat de la division
                   entière de p par d (ici 2) */
r = p % d ;     /* r contient le reste de la division
                   de p par d (ici 1) */
```

7.2. Opérateur d'affectation

7.2.1. Syntaxe

L'instruction élémentaire d'affectation (notée ← en langage algorithmique) se traduit par le symbole **=**.

variable **=** *expression* **;**

Exemple :

```
d = b*b – 4*a*c ;
```

7.2.2. Règle de conversion implicite (casting)

Dans une affectation, si les opérandes d'une expression sont de même type, le résultat hérite de ce type, même si ce résultat est rangé dans une variable d'un autre type (compatible).

Règle de Conversion Implicite :

- ✓ Convertir les éléments de la partie droite d'une expression d'affectation dans le type de la variable ou de la constante le plus riche (liste des type du moins riche au plus riche : **`unsigned char`**, **`char`**, **`unsigned int`**, **`int`**, **`float`**, **`double`**) .
- ✓ Faire les opérations de calcul dans ce type.
- ✓ Puis convertir le résultat dans le type de la variable affectée.

Exemple de conversion implicite non désirée:

```
float a ;
int i=5 ;

a = 12/i ;
```

`12` *est une valeur entière et* **`i`** *une variable de type* **`int`**, *par conséquent, le résultat est de type entier (casting) correspondra à la division entière. L'opération* **`12/i`** *fournira la valeur* **`2`** *qui sera ensuite stockée dans la variable* **`a`** *de type* **`float`** (résultat souhaité erroné).

On peut contourner cette erreur d'arrondi en forçant le type d'un des opérandes en **`float`**.

```
a = 12.0/i ;
```

`12.0` est une valeur réelle (**`float`**) et **`i`** de type **`int`**, donc le calcul sera effectué de type **`float`** et fournira la valur **`2.4`** qui est la valeur attendue !

Il est possible de forcer la conversion d'une variable (ou d'une expression) sans passer par une conversion implicite. Cette opération est appelée "**cast**". Elle se réalise de la manière suivante :

variable = (*type*) expression

Exemple :

```
a = (float) 12/i ;
```

Le résultat fournit **`2.4`** qui est la valeur attendue !

7.3. Opérateurs relationnels

(Langage algorithmique) : A, B variables (scalaires) de même type			**Traduction C**
A = B	test d'égalité	:	`A == B`
A ≠ B	test de différence	:	`A != B`
A < B	test inférieur strictement	:	`A < B`
A <= B	test inférieur ou égal	:	`A <= B`
A > B	test supérieur strictement	:	`A > B`
A >= B	test supérieur ou égal	:	`A >= B`

Remarque : **Attention à ne pas confondre le symbole d'égalité == avec le symbole d'affectation =.**

7.4. Opérateurs logiques (booléens)

Ces opérateurs permettent d'élaborer des prédicats (ou test logique). Le résultat de ces opérations est binaire :

- ✓ 0 si le prédicat est FAUX,
- ✓ une valeur différente de 0 si le prédicat est VRAI.

Opérateur	Fonction
`&&`	ET logique
`|| (ALTGr6)`	OU logique
`!`	NON logique

7.5. Opérateur d'incrémentation, décrémentation

Le langage C autorise des écritures simplifiées pour l'incrémentation et la décrémentation de variables :

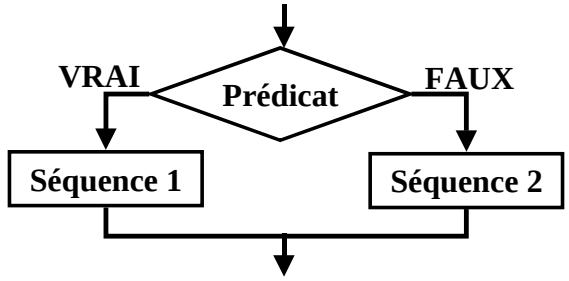
```
i = i+1 ;  est équivalent à i++ ;
i = i-1 ;  est équivalent à i-- ;
```

8. Structures de contrôle

8.1. Alternatives

8.1.1. Alternative

Organigramme



Langage algorithmique

Si (Prédicat VRAI**)**

Alors Séquence 1

Sinon Séquence 2

Fin si

Langage C :

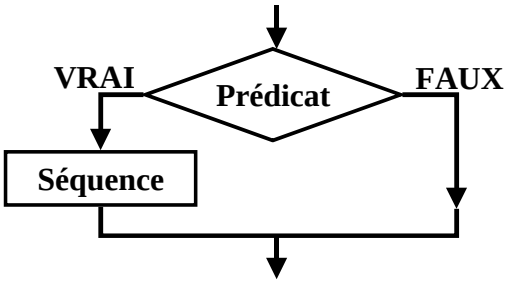
```
if ( Prédicat VRAI )
{
    ... ;  // Séquence 1
    ... ;
}
else
{
    ... ;   // Séquence 2
    ... ;
}
```

*Dans le cas où il n'y a qu'**une seule** instruction, on peut omettre les accolades :*

```
if ( Prédicat VRAI )
   ... ;  // Séquence 1
else
   ... ;   // Séquence 2
```

8.1.2. Alternative simple

Organigramme



Langage algorithmique

Si (Prédicat VRAI**)**

Alors Séquence

Fin si

Langage C :

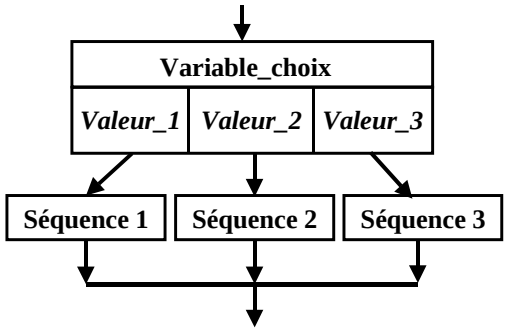
```
if ( Prédicat VRAI )
{
    ... ;  // Séquence
    ... ;
}
```

*Dans le cas où il n'y a qu'**une seule** instruction, on peut omettre les accolades :*

```
if ( Prédicat VRAI )
   ... ;  // Séquence
```

8.1.3. Alternative multiple

Organigramme



Langage algorithmique

Cas de (Variable_choix)

- ***Valeur_1*** **:** Séquence 1
- ***Valeur_2*** **:** Séquence 2
- ***Valeur_3*** **:** Séquence 3

Fin cas

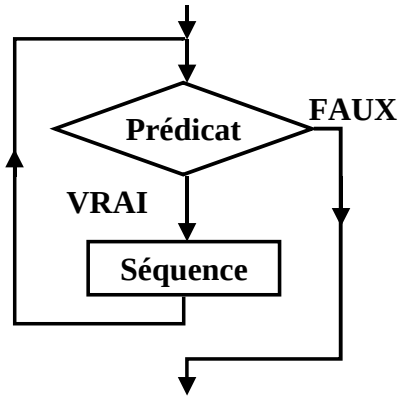
Langage C :

```
switch (Variable_choix)
{
    case Valeur_1 :  ...   ;  // Séquence 1
                     ...   ;
                    break ;
    case Valeur_2 :  ...   ;  // Séquence 2
                     ...   ;
                    break ;
    case Valeur_3 :  ...   ; // Séquence 3
                     ...   ;
                    break ;
     default  :     ...   ; // Séquence
                    ...   ; // optionnelle
}
```

8.2. Itérations

8.2.1. Itération universelle Tant que

Organigramme



Langage algorithmique

Tant que (Prédicat VRAI**) Faire**

Séquence

Fin tant que

Langage C :

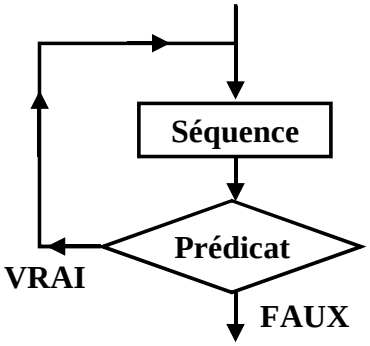
```
while ( Prédicat VRAI )
{
   ... ;
   ... ;  // Séquence
   ... ;
}
```

*Dans le cas où il n'y a qu'**une seule** instruction, on peut omettre les accolades :*

```
while ( Prédicat VRAI )
   ... ;  // Séquence
```

8.2.2. Itération Faire … Tant que

Organigramme



Langage algorithmique

Faire

Séquence

Tant que (Prédicat VRAI**)**

Langage C :

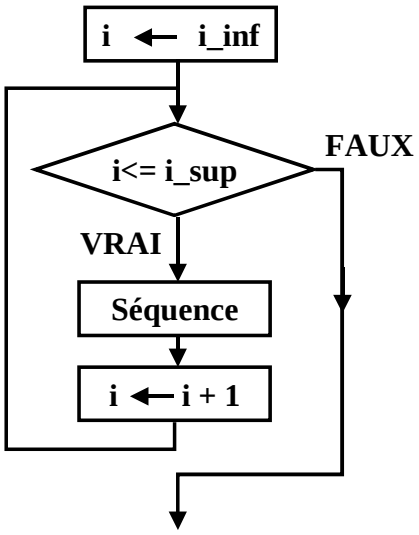
```
do
{
   ... ;
   ... ;  // Séquence
   ... ;
} while ( Prédicat VRAI );
```

*Dans le cas où il n'y a qu'**une seule** instruction, on peut omettre les accolades :*

```
do
   ... ;  // Séquence
while ( Prédicat VRAI );
```

8.2.3. Itération Pour

Organigramme



Langage algorithmique

Pour i **de** i_inf **à** i_sup **par pas de 1** **Faire**

Séquence

Fin pour

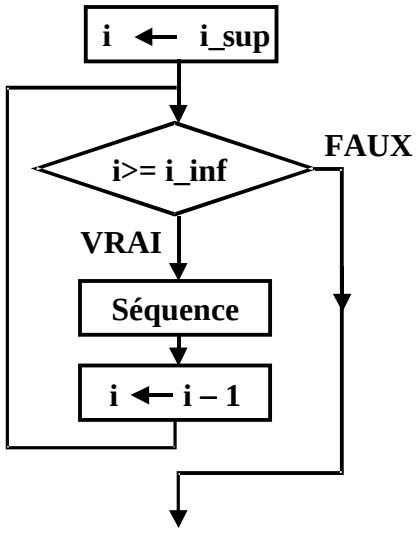
Langage C :

```
for (i=i_inf ; i <= i_sup ; i++)
{
   ... ;
   ... ;      // Séquence
   ... ;
}
```

*Dans le cas où il n'y a qu'**une seule** instruction, on peut omettre les accolades :*

```
for (i=i_inf ; i <= i_sup ; i++)
  ... ;  // Séquence
```

Organigramme



Langage algorithmique

Pour i **de** i_sup **à** i_inf **par pas de –1** **Faire**

Séquence

Fin pour

Langage C :

```
for (i=i_sup ; i >= i_inf ; i--)
{
   ... ;
   ... ;      // Séquence
   ... ;
}
```

9. Déclaration de type synonyme

L'utilisateur a la possibilité de créer ses propres types grâce au déclarateur **`typedef`** : il suffit de les déclarer en début de programme (avant la fonction **`main`**).
Ceci est particulièrement intéressant pour définir des types **tableau**.

Exemple :

```
#define TAILLE 100
#define NB_CAR_MAX 50
typedef float t_reels [TAILLE] ;
typedef int  t_entiers [TAILLE] ;
typedef char t_chaine [NB_CAR_MAX] ;
int main ()
{
   t_reels val_r ;     // tableau d'au plus 100 réels
   t_entiers val_ent ; // tableau d'au plus 100 entiers
   t_chaine mot      ; // chaîne d'au plus 50 caractères
   .
   .
   .
   return 0 ;
}
```

10. Pointeurs

Pour réaliser le passage de paramètres effectifs **par ad**resse lors de l'appel de sous-programmes, on doit utiliser des **variables particulières** permettant de stocker **une adresse** d'**entier**, une **adresse** de **réel** ou une **adresse** de **caractère**. De telles variables s'appellent des **variables pointeurs**.

10.1. L'opérateur adresse &

En langage C, l'opérateur adresse **&** fournit **l'adresse** d'une variable en mémoire.

Exemple :

```
int i ;
printf("Voici i :  %d \n", i) ;
printf("Voici son adresse en hexadécimal : %p \n", &i) ;
```

10.2. Déclaration d'une variable pointeur

La déclaration d'une variable pointeur pointant sur une variable de type donné s'effectue à l'aide du symbole ***** .

L'opérateur ***** désigne le contenu de l'adresse pointée par une variable pointeur :

Exemple :

```
char *pc  ;  /*  pc est un pointeur pouvant pointer sur
                 une variable de type char */
int *pi   ;  /*  pi est un pointeur pouvant pointer sur
                 une variable de type int */
float *pr ;  /*  pr est un pointeur pouvant pointer sur
                 une variable de type float */
char c  ;
int i   ;
float r ;

pc = &c ; // pc contient l'adresse de c
pi = &i ; // pi contient l'adresse de i
pr = &r ; // pr contient l'adresse de r

*pc = 'a' ; /* le contenu de la variable d'adresse
               pointée par pc est chargée avec le
               caractère 'a' */
*pi = 15   ;
*pr = 0.25 ;
```

Remarques :
Les deux instructions suivantes sont équivalentes :

`*pc = 'a' ;` *et* `c = 'a' ;`

De même, on a les instructions équivalentes suivantes :

`*pi = 15 ;` *et* `i = 15 ;`

`*pr = 0.25 ;` *et* `r = 0.25 ;`

Les variables pointeurs permettent donc de contrôler directement le contenu d'une autre variable.

10.3. Tableaux à une dimension et pointeurs

En déclarant un **tableau** en C, on définit automatiquement un pointeur (on définit en fait l'adresse du premier élément du tableau.

Exemple :

```
float  Liste[10]  ;  /* liste désigne l'adresse du
                        premier élément des 10 valeurs
                        réelles */
```

Les écritures suivantes sont équivalentes :

```
*Liste          Liste[0]   // le 1er élément
*(Liste+i)      Liste[i]   // le i+1 eme élément
Liste           &Liste[0]  // adresse du 1er élément
Liste+i         &Liste[i]  // adresse du i+1 eme élément
```

Exemple d'utilisation d'un pointeur pour contrôler une variable tableau à une dimension:

```
float *t   ; /* t est un pointeur pouvant contenir
                l'adresse d'un float */
int i ;

t = Liste ; // t contient l'adresse du 1er élément
for (i=0 ; i < 10 ; i++)
{
   t[i] = 0.1*i ;
}
```

Ces instructions ont permis d'initialiser les valeurs du tableau **`liste`**. Elles sont équivalentes aux instructions suivantes effectuées directement sur la variable **`liste`** :

```
for (i=0 ; i < 10 ; i++)
{
   Liste[i] = 0.1*i ;
}
```

Remarques :

`t+i` *désigne l'adresse du* **`i`**^{e} *élément situé après l'élément pointé par* **`t`**

`*(t+i)` *désigne le contenu de l'adresse du* **`i`**^{e} *élément situé après l'élément pointé par* **`t`**

`*(t+i)` *et* **`t[i]`** *sont des notations équivalentes*

11. Sous-programmes

En langage C, tous les sous-programmes sont appelés fonctions. On distingue néanmoins clairement :

- les fonctions qui ne renvoient aucune valeur (**procédures**),
- les fonctions qui renvoient une valeur.

Le programme démarre toujours par l'exécution de la première fonction dite principale (**`main`**) que l'on assimile au programme principal.

11.1. Procédures sans passage de paramètres et ne renvoyant rien au programme principal

Exemple :

```
#include <stdio.h>
#include <stdlib.h>
void bonjour(void) ;     // prototype
int main ()
{
   bonjour () ;  // appel de la fonction bonjour
   system("pause") ;  // pause d'affichage
   return 0 ;
}
void bonjour(void)       //  développement complet de
{                        //  la fonction bonjour
   printf("Bonjour\n") ;
}
```

11.2. Fonctions sans passage de paramètres et renvoyant une valeur au programme principal

Dans ce cas, la fonction, après exécution, renvoie une valeur. Le type de cette valeur est déclarée avec cette fonction. La valeur retournée est spécifiée à l'aide du mot réservé **`return`**.

Exemple : On suppose que l'on dispose d'une fonction **`random(n)`** (dont le prototype est décrit dans un fichier interface **`tir_alea.h`**) qui renvoie un nombre entier aléatoire compris entre **`0`** et **`n-1`**. La suite aléatoire doit être initialisée par la fonction **`randomize`**.

```
#include <stdio.h>
#include <stdlib.h>
#include "tir_alea.h"  /* contient le prototype de random et
                          randomize */
int lance_de(void) ;   // prototype
int main ()
{
   int resultat ;
   resultat = lance_de () ;
```

```
   printf("Vous avez obtenu le nombre %d",resultat) ;
   system("pause") ;
   return 0 ;
}
int lance_de(void)       //  développement complet de
{                        //  la fonction lance_de
   int test ;  // variable dont on retournera la valeur
   test = random (6)+1 ;
   return test ;
}
```

11.3. Sous-programmes avec passage de paramètres

Ce paragraphe traite directement du cas général : procédure ou fonction ne renvoyant rien (type **`void`**) ou fonction renvoyant une valeur d'un certain type.

On peut distinguer 2 types de passages de paramètres :

- passage de paramètres **par valeur** : on recopie **la valeur du paramètre effectif** du module appelant dans un paramètre de **même type** (sens module appelant vers fonction),
- passage de paramètre **par adresse** : on recopie l'**adresse du paramètre effectif** dans un paramètre de type **pointeur**, **capable de stocker l'adresse**. On peut alors accéder au contenu du paramètre effectif et le modifier connaissant son adresse !

11.3.1. Passage de paramètres par valeur

Exemple :

```
#include <stdio.h>
#include <stdlib.h>
float somme_reels(float a, float b) ;     // prototype
int main ()
{
   float A, B, C ;

   printf ("Valeur de A ?") ;  // Saisies
   scanf("%f",&A) ;
   printf ("Valeur de B ?") ;
   scanf("%f",&B) ;
   C = somme_reels(A, B) ; /* passage des paramètres effectifs A
                              et B par valeur */
   printf("%5.2f + %5.2f = %5.2f \n",A, B, C) ;
   system("pause") ;
   return 0 ;
}
```

```
float somme_reels(float a, float b)  //  développement complet de
 {                                    //  la fonction
    float c ;  // variable dont on retournera la valeur
    c = a + b ;
    return c ;
 }
```

11.3.2. Passage de paramètres par adresse

Exemple :

```
#include <stdio.h>
#include <stdlib.h>
void permute(float *x, float *y) ;     // prototype
int main ()
{
   float A, B

   printf ("Valeur de A ?") ;  // Saisies
   scanf("%f",&A) ;
   printf ("Valeur de B ?") ;
   scanf("%f",&B) ;
   permute(&A, &B) ; /* passage des paramètres effectifs A et B
                        par adresse */
   printf("Après permutation, A = %5.2f, B = %5.2f",A, B) ;
   system("pause") ;
   return 0 ;
}
void permute(float *x, float *y)  //  développement complet de
{                                 //  la procédure
   float aux ;  // variable locale

   aux = *x ;  // *a est le contenu de l'adresse pointée par a
   *x = *y  ;  // *b est le contenu de l'adresse pointée par b
   *y = aux ;
}
```

11.4. Passage de paramètres de type tableau

En langage C, la déclaration d'un tableau définit automatiquement un pointeur sur une zone d'un nombre fini d'éléments.

Exemple :

```
float Liste[10] ; /* liste désigne alors un tableau d'au plus
                     10 réels */
```

`Liste` désigne **l'adresse** du premier élément c'est à dire **`&Liste[0]`**.

Donc, lorsque l'on passe un paramètre effectif de type tableau en invoquant son identificateur, on passe nécessairement ce paramètre **par adresse** !

11.4.1. Passage de paramètres tableau par adresse

Exemple :

```
#include <stdio.h>
#include <stdlib.h>
#define N_MAX 100
typedef float t_reels[N_MAX] ;
void chs(float *t, int n) ;     // prototype
int main ()
{
   t_reels Liste = {-1.5, 1.5, 3.2, -9.7, 10.5} ;
   int N = 5 ;
   chs (Liste,N) ;
   system("pause") ;
   return 0 ;
}
void chs(float *t, int n) // change le signe de chaque élément
{                         // du tableau transmis par adresse
   int i ;  // variable locale
   for (i=0 ; i < n ; i++)
   {
      t[i] = - t[i] ;
   }
}
```

Remarque : On peut aussi utiliser le type **`t_reels`** qui définit automatiquement un pointeur adapté au tableau qui est transmis (méthode la plus simple en cas de tableaux à plusieurs dimensions). Le prototype de l'exemple précédent s'écrit alors :
`void chs(t_reels t, int n) ;`

11.4.2. Passage de paramètres tableau par valeur

Ce mode de transmission est impossible ! il est toutefois possible d'indiquer au compilateur que toute modification du contenu des valeurs du tableau **est interdite** en plaçant le mot-clé **`const`** devant le paramètre formel correspondant .

Exemple :

```
#include <stdio.h>
#include <stdlib.h>
#define N_MAX 100
typedef float t_reels[N_MAX] ;
void affiche(const float *t, int n) ;     // prototype
int main ()
{
   t_reels Liste = {-1.5, 1.5, 3.2, -9.7, 10.5} ;
   int N = 5 ;
   affiche (Liste,N) ;
```

```
   system("pause") ;
   return 0 ;
}
void affiche(const float *t, int n) // affiche les n éléments
{                                   // du tableau transmis
   int i ;  // variable locale
   for (i=0 ; i < n ; i++)
   {
      printf ("%d eme valeur : %5.2E", i+1, t[i]);
   }
}
```

Remarque : On peut aussi utiliser le type **`t_reels`** qui définit automatiquement un pointeur adapté au tableau qui est transmis (méthode la plus simple en cas de tableaux à plusieurs dimensions).

Le prototype de l'exemple précédent s'écrit alors :

```
void affiche(const t_reels t, int n) ;
```

12. Bibliothèques

L'ensemble de toutes les fonctions disponibles dans les bibliothèques ne sera pas indiqué. On fournit ici les conditions d'utilisation de celles qui seront les plus utilisées en TP.

12.1. Fonctions du fichier interface `stdio.h`

***Fonctions d'entrées-sorties* I/O (Input/Output)**

Nom	Fonction	Exemple d'utilisation
`getchar`	Renvoie un caractère lu au clavier	**`char c ;`** **`c = getchar () ;`**
`gets`	Lecture d'une chaîne de caractères	**`char s[10] ;`** **`gets (s) ;`**
`putchar`	Ecriture d'un caractère à l'écran	**`char c ;`** **`putchar (c) ;`**
`puts`	Ecriture d'une chaîne de caractères à l'écran	**`char s[10] ;`** **`puts (s) ;`**
`scanf`	Lecture et formatage depuis le clavier (ou un fichier)	**`scanf ("`*`format`*`",`** *adresse de variable*`) ;`
`printf`	Ecriture formatée vers l'écran (ou un fichier)	**`printf ("`*`format`*`",`** *liste de variables*`) ;`

12.2. Fonctions du fichier interface `stdlib.h`

Fonctions particulières de la librairie standard **(STanDard LIBrary)**

Nom	Fonction	Exemple d'utilisation
rand	Renvoie un nombre pseudo aléatoire.	`int i ;` `i= rand () ;`
srand	Initialise le générateur de nombre pseudo-aléatoire.	`unsigned int seed ;` `srand (seed) ;`

atoi	Convertit une chaîne de caractère en entier.	`char s[80] ;`
		`int i ;`
		`i = atoi(s) ;`

atof	Convertit une chaîne de caractère en réel.	`char s[80] ;`
		`float r ;`
		`r = atof(s) ;`

system	Invoque une commande du mode console (**MS-DOS**) spécifiée dans la chaîne de caractères transmise.	`system ("cls") ;` * `system ("pause") ;` **

***** *provoque l'effacement de l'écran*
****** *provoque une pause d'affichage*

12.3. Fonctions du fichier interface `math.h`

Fonctions mathématiques élaborées

Nom	Fonction	Exemple d'utilisation
acos	Renvoie l'arc cosinus de l'argument.	`float y, x ;` `y = acos (x) ;`
asin	Renvoie l'arc sinus de l'argument.	`float y, x ;` `y = asin(x) ;`
atan	Renvoie l'arc tangente de l'argument.	`float y, x ;` `y = atan(x) ;`
cos	Renvoie le cosinus de l'argument.	`float y, x ;` `y = cos(x) ;`
sin	Renvoie le sinus de l'argument.	`float y, x ;` `y = sin(x) ;`
cosh	Renvoie le cosinus hyperbolique de l'argument.	`float y, x ;` `y = cosh(x) ;`
sinh	Renvoie le sinus hyperbolique de l'argument.	`float y, x ;` `y = sin(x) ;`
tanh	Renvoie la tangente hyperbolique de l'argument.	`float y, x ;` `y = tanh(x) ;`
ceil	Arrondi par excès à l'unité.	`float y, x ;` `y = ceil(x) ;`
floor	Troncature (partie entière).	`float y, x ;` `y = floor(x) ;`
fabs	Valeur absolue d'un réel.	`float y, x ;` `y = fabs(x) ;`

Nom	Fonction	Exemple d'utilisation
exp	Exponentielle.	`float y, x ;` `y = exp(x) ;`
pow	Fonction puissance : $y = x^p$	`float y, x, p ;` `y = pow(x,p) ;`
sqrt	Fonction racine carrée : $y = \sqrt{x}$	`float y, x, ;` `y = sqrt(x) ;`
log	Fonction logarithme népérien : y = Ln(x)	`float y, x, ;` `y = log(x) ;`
log10	Fonction logarithme à base 10 : y = Log(x)	`float y, x, ;` `y = log10(x) ;`

Le fichier interface **math.h** définit également les constantes suivantes

Nom de la constante	Valeur
M_PI	π
M_PI_2	$\pi/2$
M_PI_4	$\pi/4$

Nom de la constante	Valeur
M_SQRT2	$\sqrt{2}$
M_E	e^1
M_LN2	Ln(2)
M_LN10	Ln(10)

12.4. Fonctions du fichier interface `string.h`

Fonctions relatives aux chaînes de caractères

Nom	Fonction	Exemple d'utilisation
strcat	Concatène **s2** à **s1** (met **s2** au bout de **s1**). Le résultat est dans **s1**.	`char s1[80], s2[80];` `strcat (s1, s2) ;`

strcmp	Compare **s1** et **s2** : **test** < 0 si **s1** < **s2**
	test = 0 si s1 = **s2**
	test > 0 si **s1** > **s2**
		`char s1[80], s2[80];`
		`int test ;`
		`test =strcmp(s1,s2);`

strcpy	Copie **s2** dans **s1**.	`char s1[80], s2[80];` `strcpy (s1, s2) ;`

strlen	Renvoie la longueur d'une chaîne.	`char s[80] ;`
		`int long_s ;`
		`long_s =strlen(s);`

strlwr	Convertit une chaîne en minuscules.	`char s[80] ;` `strlwr(s) ;`
strupr	Convertit une chaîne en majuscules.	`char s[80] ;` `strupr(s) ;`

12.5. Fonctions du fichier interface `ctype.h`

Fonctions relatives aux caractères

Nom	Fonction	Exemple d'utilisation
`toupper`	Convertit un caractère c en majuscule.	`char c;` `toupper(c) ;`
`tolower`	Convertit un caractère c en minuscule.	`char c;` `tolower(c);`

`isdigit`	Retourne un entier différent de 0 si l'argument est un caractère chiffre de **`'0'`** à **`'9'`**.	`char c;`
		`int test ;`
		`test = isdigit(c);`

`isalpha`	Retourne un entier différent de 0 si l'argument est une lettre non accentuée de l'alphabet de **`'A'`** à **`'Z'`** ou de **`'a'`** à **`'z'`**.	`char c;`
		`int test ;`
		`test = isalpha(c);`

`isalnum`	Retourne un entier différent de 0 si l'argument est un caractère chiffre ou une lettre non accentuée de l'alphabet .	`char c;`
		`int test ;`
		`test = isalnum(c);`

`islower`	Retourne un entier différent de 0 si l'argument est un caractère minuscule.	`char c;`
		`int test ;`
		`test = islower(c);`

`isupper`	Retourne un entier différent de 0 si l'argument est un caractère majuscule.	`char c;`
		`int test ;`
		`test = isupper(c);`

TABLE DES CODES ASCII STANDARD

N°dec	HEX	CAR	N°dec	HEX	CAR	N°dec	HEX	CAR	N°dec	HEX	CAR
32	20	' '	58	3A	':'	84	54	'T'	110	6E	'n'
33	21	'!'	59	3B	';'	85	55	'U'	111	6F	'o'
34	22	'"'	60	3C	'<'	86	56	'V'	112	70	'p'
35	23	'#'	61	3D	'='	87	57	'W'	113	71	'q'
36	24	'$'	62	3E	'>'	88	58	'X'	114	72	'r'
37	25	'%'	63	3F	'?'	89	59	'Y'	115	73	's'
38	26	'&'	64	40	'@'	90	5A	'Z'	116	74	't'
39	27	'''	65	41	'A'	91	5B	'['	117	75	'u'
40	28	'('	66	42	'B'	92	5C	'\'	118	76	'v'
41	29	')'	67	43	'C'	93	5D	']'	119	77	'w'
42	2A	'*'	68	44	'D'	94	5E	'^'	120	78	'x'
43	2B	'+'	69	45	'E'	95	5F	'_'	121	79	'y'
44	2C	','	70	46	'F'	96	60	'`'	122	7A	'z'
45	2D	'-'	71	47	'G'	97	61	'a'	123	7B	'{'
46	2E	'.'	72	48	'H'	98	62	'b'	124	7C	'|'
47	2F	'/'	73	49	'I'	99	63	'c'	125	7D	'}'
48	30	'0'	74	4A	'J'	100	64	'd'	126	7E	'~'
49	31	'1'	75	4B	'K'	101	65	'e'			
50	32	'2'	76	4C	'L'	102	66	'f'			
51	33	'3'	77	4D	'M'	103	67	'g'			
52	34	'4'	78	4E	'N'	104	68	'h'			
53	35	'5'	79	4F	'O'	105	69	'i'			
54	36	'6'	80	50	'P'	106	6A	'j'			
55	37	'7'	81	51	'Q'	107	6B	'k'			
56	38	'8'	82	52	'R'	108	6C	'l'			
57	39	'9'	83	53	'S'	109	6D	'm'			

QUELQUES CARACTERES SPECIAUX NON IMPRIMABLES

Code ASCII (Décimal)	Ecriture en C	Désignation standard	Action sur certains périphériques
0	'\x0' ou '\0'	NUL	
7	'\x7'	BEL	*Emission d'un bip*
8	'\x8'	BS	*Retour en arrière d'un caractère*
9	'\x9'	HT	*Saut à la tabulation horizontale suivante*
10	'\xA'	LF	*Saut d'une ligne vers le bas*
12	'\xC'	FF	*Saut d'une page ver le bas*
13	'\xD'	CR	*Retour au début de la ligne courante*
27	'\x1B'	ESC	*Echappement*
127	'\x7F'	DEL	*Effacement*