COURS
de Techniques de programmation en
LANGAGE C

Par :
M. Khalifa MANSOURI

Référence :

Ce cours est pris du livre d'informatique :

PROGRAMMATION
DES APPLICATIONS INFORMATIQUES
Rappels de cours et exercices corrigés

Maison d'édition TOUBKAL, 2004
Dépôt légal : 2004 / 1431 * ISBN : 9954 - 409– 58 – 0**
Auteur M. Khalifa MANSOURI

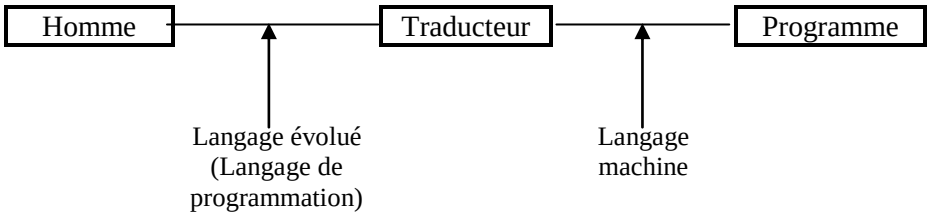
TABLE DES MATIERES

M. Khalifa MANSOURI

1. GENERALITES SUR LE LANGAGE C

1.1. Introduction

Si les Hommes ont du développer au cours des siècles différents langages leur permettant de dialoguer entre eux, ils ont du faire de même pour établir la communication avec la machine, c'est pourquoi ils ont créé plusieurs langages spécifiques appelés : langages de programmation.



Le programmeur écrit son programme avec u n langage évolué, ce programme se traduit en langage machine, que la machine comprendra et exploitera pour effectuer des opérations.

Le langage machine est principalement le binaire.

Les langages évolués possèdent des traducteurs qui permettent de traduire le programme source (écris par le programmeur) en un programme « dit objet » écrit en langage machine. On distingue deux types de traducteurs :

- ✓ Les interpréteurs (Exemple : le Basic) :

Un interpréteur traduit chaque ligne du programme en langage machine et l'exécute, puis passe à la ligne suivante. Si jamais il trouve une erreur de syntaxe dans une ligne donnée, il la mentionne à l'opérateur et doit reprendre la traduction et l'exécution dès le début.

- ✓ Les compilateurs (Exemple : Pascal, C, Fortran …) :

Un compilateur traduit tout le programme en langage machine avant de l'exécuter. Si jamais il trouve une erreur de syntaxe dans une ligne donnée, il la mentionne à l'opérateur et ne doit reprendre dès le début que la traduction.

Remarque : Les compilateurs sont plus rapides que les interpréteurs.

Les langages de programmation possèdent certaines caractéristiques :On note que comme la transmission de l'information dans les langues parlées et écrites, un langage se compose :

- d'un alphabet,
- d'une grammaire,
- d'une orthographe.

Parmi les langages de programmation, on peut citer :

- le COBOL ; plus particulièrement utilisé pour la gestion de l'entreprise,
- le FORTRAN et L'ALGOL : utilisé pour les opérations dans le domaine scientifique,
- Le PASCAL et le Langage C : utilisés dans les applications structurées,
- Le C++, Le JAVA : utilisés dans les application de programmation orientée objet,
- Le VISUAL BASIC : utilisé dans la programmation événementielle.
- … etc.

1.2. Généralités sur le langage C

1.2.1. Pourquoi le langage C

Le langage C est un langage de programmation structuré qui possède des caractéristiques intéressantes :

- une syntaxes simple
- des structures clairement définies
- un jeu étendue d'opérateurs et de fonctions standards.

Le langage C offre en plus les avantages suivants :

- une grande vitesse de compilation
- un environnement intégré fournissant notamment un traitement de texte.

Cela signifie que l'on peut compiler, modifier et exécuter un programme sans sortir de turbo C

Le langage C a connu un sucés considérable du en partie à un autre phénomène : le système d'exploitation UNIX.

Ce système d'exploitation développé pour l'enseignement de l'informatique a été adapté par de nombreux constructeurs d'ordinateurs. Le système étant principalement écrit en C, ce langage a connu un développement parallèle a celui d'UNIX

1.2.2. Premiers programmes en langage C

Calcul du périmètre d'un cercle

1ère Version :

```
#include <STDIO.H>
#define Pi 3.14

float R, P;

void main()
{
      printf("Programme de calcul du périmètre d'un
      cercle \n");
      printf("Entrer le rayon du cercle :");
      scanf(" %f ",&R) ;
      P = 2 * Pi * R ;
      printf(" périmètre de ce cercle est : %f",P);
      getch() ;
}
```

Dans ce premier programme, on trouve plusieurs constituants d'un fichier C, on y trouve :

- l'utilisation des directives du co-processeur,
- la définition de variables,
- la fonction principale main( ).

On va améliorer ce programme en le décomposant en plusieurs sous programmes.

2ème Version :

```
# include <STDIO.H>
# define Pi 3.14

void saisie();
float calcul(float R1);
void affichage();

float R, P;

void saisie()
{
      printf("Entrer le rayon du cercle :");
      scanf(" %f ",&R) ;
}

float calcul(float R1)
{
      float P1 ;
      P1 = 2 * Pi * R1 ;
      Return(P1)
}

void afficahge()
{
      printf("Le périmètre de ce cercle est : %f",
      P);
      getch() ;
}

void main()
{
      printf("Programme de calcul du périmètre d'un
      cercle \n");
      saisie();
      P = calcul(R);
      Affichage();
}
```

Dans deuxième programme on trouve en plus des constituants des constituants précédents :

- Les déclarations de fonctions,
- Les définitions de fonctions,
- Les appels aux fonctions.

✓ Il demande au préprecesseur l'inclusion du fichier "header" fouines avec le compilateur : STDIO.H,

✓ Il se définit une constante : Pi dont la valeur associée est 3.14,

✓ Il contient 4 fonctions :

- Main : programme principal,
- Saisie : fonction sans type,
- Calcul : fonction réelle,
- Affichage : fonction sans type.

✓ Les variables globales définies sont :

- R : variable réelle,
- P : variable réelles,

✓ La seule variable locale est :

- P1 : variable réelle locale à la fonction calcul( ) ;

✓ Le seul paramètre utilisé est :

- R1 : paramètre d'entrée à la fonction calcul( ) ;

2. STRUCTURE D'UN PROGRAMME C

2.1. Introduction

La structure d'un programme C peut se décomposer de la façon suivante :

- directives de compilation
- définition des types
- prototypes de fonctions
- déclarations de variables et constantes
- fonctions

Un programme C consiste donc en une série de fonctions parmi celles-ci, l'une doit s'appeler main( ) : elle représente le programme principal.

2.2 . Les directives de compilations

Ces directives permettent de demander à l'ordinateur d'effectuer certaines opérations avant la compilation du programme.

Les directives les plus importantes sont :

- **#INCLUDE < fichier >** où fichier est le nom d'un fichier qui contient des informations utiles pour le programme

Exemple :

```
#  INCLUDE <stdio.h>
# INCLUDE <conio.h>
```

ATTENTION : ne pas oublier les < > !!!

- **# DEFINE Symbole Suite_de_caractères**

Cette directive remplace dans le programme toutes les occurrences du mot **Symbole** par la suite de caractères. Cette suite de caractères peut représenter un nombre, un texte, une formule … etc.

Elle permet la définition de nouveaux opérateurs. la déclaration de constantes, … etc.

Exemple : # DEFINE Max 9

Chaque fois que l'on rencontre le symbole **Max** dans le programme, il sera remplacé par 9.

#DEFINE Ecrire printif("tapez une touche pour continuer")

Chaque fois que l'on rencontre le symbole **Ecrire** dans le programme, il sera remplacé par l'instruction printif("tapez une touche pour continuer").

2.3 . Définitions de type

Chaque donnée utilisée dans un programme C doit posséder un type.

Ce type détermine l'ensemble des valeurs possibles que peut prendre la donnée pendant l'exécution du programme.

C'est ainsi qu'une donnée pourra contenir des valeurs de type : Entier, Réel, Caractère, …etc.

Le langage C offre la possibilité d'utiliser des types prédéfinis ou de composer ses propres types.

Une déclaration de type doit se faire de la manière suivante :

TYPEDEF Description_du_Type Indentificateur_du_Type

Exemple :

```
typedef struct
{
        Char nom[30];
        Char Prenom [20];
        Int age;
} personne;
```

On a ainsi défini un nouveau type structuré **personne** qui contient un nom, un prénom et un âge.

2.4. Déclarations de variables

Les données qu'un programme utilise peuvent varier au cours de son exécution. On les appelle variables.

Les valeurs de ces variables peuvent être modifiées au moyen d'une série d'opérations.

Toute variable utilisée par le programme doit posséder un type.

En déclarant une variable, on précise donc le type de la valeur qu'elle va contenir.

Toute déclaration de variable doit être de la forme suivante :

Type_de_la_variable Nom_de_la_variable ;

Exemple :

Int i,j ; => i et j sont deux variables entières
Char c ; => c est une variable qui pourra contenir des valeurs caractères.

Remarque :

On peut, lors de la déclaration d'une variable, lui assigner une valeur initiale : (initialisation). Cela se fait de la manière suivante :

Type_de_la_variable Nom_de_la_variable = Valeur_initiale ;

Exemples :

```
Int i = 1;
Char c = 'a' ;
```

2.5. Déclaration des constantes

Lorsque l'on désire avoir une donnée dont la valeur ne pourra être modifiée pendant l'exécution du programme, on utilise alors une constante.

La déclaration d'une constante peut être de deux formes différentes :

CONST Type nom = valeur ;

Ou bien

 # DEFINE nom valeur

Exemples :

```
# DEFINE Max 32767
CONST  Int max = 32767 ;
CONST Float pi = 3.14 ;
```

2.6. Prototypes de fonctions

Dans un programme, on peut avoir besoin d'exécuter le même ensemble d'instructions plusieurs fois, mais avec des données différentes. On regroupe alors cet ensemble d'instructions dans une fonction.

La déclaration d'un prototype de fonction se fait de la façon suivante :

Type_de_fonction Nom_de_fonction (Liste des paramètres);

2.7. Fonctions

La définition d'une fonction devra se faire de la manière suivante :

```
Type_de_fonction  Nom_de_fonction (Liste des paramètres )
{
        déclarations de variables locales à la fonction ;
        instructions ;
}
```

3. SYNTAXE DU LANGAGE

3.1. Introduction

La syntaxe du langage constitue l'orthographe du langage. Il faut s'y conformer pour que le compilateur puisse transformer le programme C en langage machine.

Toute erreur de syntaxe provoquera une erreur de compilation.

3.2. Les identificateurs

Ils représentent les noms que l'on donne aux constantes, aux types de données , aux variables, aux fonctions, …etc.

Ils doivent obéir aux règles de composition suivantes:

- un identificateur soit **TOUJOURS** commencer par une lettre (a…z),
- le reste de l'identificateur peut être composé de lettres (a …z), de chiffres (0…9) ou du caractère souligné (_),
- les 32 premiers caractères sont pris en compte par le compilateur,

- **ATTENTION** : il existe une **DIFFERENCE ENTRE LES MAJUSCULES ET LES MINUSCULES !!!**

De ce fait , I et i ne représente donc pas le même identificateur.

Exemples :

Ident, I01, Texte, Var_1 : sont les identificateurs corrects.
#e, 1eE, var 1 : sont des identificateurs incorrects.

3.3.Les mots- clés

Ils sont prédéfinis dans la syntaxe du langage. Ils représentent les instructions, les descriptions de types prédéfinis.

IL N'EST PAS PERMIS D'UTILISER UN MOT CLE COMME IDENTIFICATEUR.

Exemples :

INT, IF, ELSE, WHILE, PRINTF, …

3.4. Les expressions

Une expression est une combinaison d'opérandes et d'opérateurs.

Une opérande peut être :

- une variable ;
- une constante ;
- une autre expression.

Toute expression fournit toujours une valeur résultat. Cette valeur peut être de différents types :

- numérique ;
- caractère ;
- …

3.5. Les opérateurs

Les opérateurs indiquent quel type d'opération on désire exécuter.

3.5.1. L'assignation

C'est l'opération la plus fondamentale de tout langage de programmation.

Symbole : " = "

La valeur qui se trouve à la droite de ce signe est mise dans la variable située à sa gauche.

Elle constitue également la valeur résultat de l'expression.

Syntaxe : Variable = expression ;

Exemples :

```
A = 2 ;
B = A + 3 ;
```

3.5.2. Opérateurs arithmétiques

- L'addition : +
- La soustraction : -

- La multiplication : *
- La division : /
- Le reste de la division entière : %

Toutes ces opérations peuvent être combinées avec des parenthèses ( ).

Exemple :

Y = a + b* (c - 1) ;

On dispose également d'opérations moins classiques telles que :

- l'incrémentation : **+ +** qui ajoute 1 à la variable à laquelle elle est appliquée,
- la décrémentation : **- -** qui enlève 1 à la variable à laquelle elle est appliquée.

Exemples :

X = Y++ ; => assigne la valeur de Y à X et ajoute 1 à Y
X = ++Y ; => ajoute 1 à Y et assigne la nouvelle valeur obtenu à X .

3.5.3. Les opérateurs relationnels

Ils permettent de comparer certaines valeurs. Leur résultat peut être soit vrai, soit faux.

NOTA:

La valeur FAUX est représentée par une valeur égale à 0.
La valeur VRAI est représentée par une valeur égale à 1.

La liste des opérateurs relationnels est la suivantes :

- strictement plus grand : >
- plus grand ou égale : > =
- strictement plus petit : <
- plus petit ou égal : < =
- égal : = =
- non égal (différent) : !=

Exemples :

Si son utilise trois variables entières A, B, et C telles que : A = 1 ; B = 100 ; C = 1000
Les expressions suivantes donneront un résultat VRAI :
A <= B; C > B; C ! = B

Les expressions suivantes donneront un résultat FAUX :
A = = B; A >= B; C < A

Ces opérateurs sont particulièrement utilisés avec les instructions conditionnelles et répétitives.

3.5.4. Les opérateurs logiques

Ces opérateurs effectuent les opérations classiques sur des valeurs logiques.
Leurs résultats donnent également des valeurs vraies ou fausses.

Ils sont au nombre de trois :

- et logique : &&
- ou logique : ||
- négation logique : !

&& : donne un résultat VRAI si les deux expressions sont VRAIES.

|| : donne un résultat VRAI si une des deux expressions est VRAIE.

! : inverse la valeur logique de l'expression à laquelle elle est appliquée.

3.5.5. L'opérateur "adresse de"

La mémoire d'un ordinateur est composée d'une suite de valeurs binaires 0 ou 1 appelées bits.
Un ensemble de 8 bits est appelé **octet** ou **byte**.

Un ensemble de 16 bits est appelé **mot**.

Un ensemble de 32 bits est appelé **long mot**.

Chaque ensemble a une position bien définie dans la mémoire. Cette position porte le nom d'adresse.

Lorsqu'on utilise l'opérateur "**adresse de**", on demande l'adresse qu'occupe la variable dans la mémoire. Le format de cette opération est le suivant :

&nom_de_la_variable

3.5.6. Les opérateurs abrégés

Dans certaines conditions, quelques opérations admettent une forme abrégée. En effet, l'expression suivante :

Variable1 = Variable1 Opérateur Expression ;

Peut s'abréger comme suit :

Variable1 Opérateur = Expression ;

Exemple :

I = I + 1; peut s'écrire comme suit : **I + = 1 ;**

3.6. Le séparateurs

Ils sont en quelque sorte la ponctuation du langage.

3.6.1. Le point virgule : ";"

Il doit séparer toutes les instructions du programme. Il marque la séquentialité.

Exemple :

```
Int x ;
x = x + 1;
```

3.6.2. La virgule : ","

Elle sépare une liste d'éléments.

Exemple:

```
Int i, j, k, l ;
```

3.6.3. Les parenthèses : "( )"

Elles sont utilisées dans certaines expressions pour déterminer l'ordre dans lequel les opérations doivent être effectuées.

Elles peuvent aussi encadrer une liste de paramètres pour des instructions d'entrée / sortie ou pour des fonctions.

Exemples :

```
I = ( J + 1 ) *3 ;
Scanf("%s", nom) ;
```

3.6.4. L'apostrophe : " ' "

Elle est utilisée pour entourer UN caractère unique.

Exemple : 'a'

3.6.5. Les guillemets : ' " '

Ils sont utilisés pour délimiter une chaîne de caractères.

Exemple : "Turbo C"

3.6.6. Les accolades : " { } "

Elles déterminent des blocs d'instructions et notamment le bloc principal.

Exemple :

```
MAIN( )
{
        INT i ;
        …;
        i = i + 1;
        …;
}
```

3.6.7. Le point : " . "

Il est utilisé pour séparer la partie entière de la partie décimale d'un nombre réel. Il aussi est utilisé pour faire référence à un champ d'une variable structurée.

Exemple :

pi = 3.1415927 ;
Personne.Taille = 1.80 ;

3.6.8. Les crochets : " [ ] "

Ils entourent l'indice d'une variable dimensionnée.

Exemple : T[1] = 0 ;

3.7. Les commentaires

Il est parfois utile pour la compréhension d'un programme d'insérer un texte explicatif. Cela se fait en encadrant le texte avec les symboles /* et */.

Exemple:

Int i ; /* i est une variable entière */

Le texte placé entre ces deux symboles est simplement ignoré par le compilateur, et n'est pas traduit en langage machine.

4. LES TYPES DE DONNEES

4.1. Introduction

En langage C, il existe 4 types de base :

- les entiers : type INT (Exemples : 1, 2, -3, 145 , …)
- les nombres en points flottants : type FLOAT (Exemples : 123, -67.89, -34.5 , …)
- les textes (caractères ou string) : type CHAR (Exemples : 'A' , "BONJOUR" , …)
- les pointeurs qui représentent des adresses mémoire contenant de l'information.

4.2. Le type INT

Ce type représente des variables entières.

Déclaration :

Int Nom_de_la_variable ;

Le langage C permet aussi de préciser si les entiers doivent être de types longs (LONG INT) ou de types courts (SHORT INT).

Ces extensions permettent de choisir les valeurs maximales et minimales que peuvent contenir les variables.

La différence provient du nombre d'octets utilisés pour représenter la variables en mémoire :

- deux octets pour les types INT et SHORT INT
- quatre octets pour le type LONG INT.

Enfin, le langage C permet de préciser si la variable ne peut contenir que des valeurs positives ou nulles. Dans ce cas, on utilisera le type UNSIGNED INT.

Les valeurs limites des types entiers sont données dans le tableau suivant :

Type	Valeur minimale	Valeur maximale
INT	- 32768	32767
LONG INT	- 2147483648	2147483647
UNSIGNED INT	0	65535
UNSIGNED LONG INT	0	4294967295

Exemple :

```
Int I, J;
LONG  Int K ;
I = -1
J = 9 + I ;
K = 589L ;
```

Remarque : Il est possible de préciser qu'une constante entière doit être du type LONG. Il suffit de faire suivre cette constante de la lettre L (en majuscules ou en minuscule).

4.3. Le type FLOAT

Ce type représente les données numériques réelles.

Déclaration :

Float Nom_de_la_variable ;

Il y a également comme pour les entiers une extension de type Float, il s'agit du type DOUBLE.

Les valeurs limites des types réels sont données dans le tableau suivant :

Type	Valeur minimale	Valeur maximale
FLOAT	3.4^{E-38}	3.4^{E+38}
DOUBLE	1.7^{E-308}	1.7^{E+308}

Remarques :

- 3.4^{E-38} signifie $3.4 * 10^{-38}$
- Les calculs sur les nombres réels prennent beaucoup de temps !

3.4. Le type CHAR

Le langage C permet d'utiliser des variables caractères et des textes.

Déclaration :

Char Nom_de_la_variable ;

Les valeurs possibles pour ce type de variables correspondent aux caractères.

Un caractère est représenté en mémoire par un seul octet.

Exemples :

```
Char C;
C = 'A' ;
```

Le langage C offre également la possibilité de définir et de manipuler des variables textes. Cela peut se faire de DEUX façons différentes :

- soit en précisant la longueur lors de la déclaration,
- soit en ne précisant pas cette longueur.

Premier cas : Tableau de caractères

Déclaration : Char Nom_du_texte[longueur];

Exemple : Char Ch[10];

La variable Ch pourra contenir un texte de 9 caractères. La $10^{ème}$ case étant réservée par l'ordinateur pour signaler la fin du texte. Il ne doit donc pas être utilisé par le programmeur.
Pour assigner une valeur texte à une variable de ce type, il faut utiliser une fonction spéciale. Il s'agit de la fonction **STRCPY** dont le format est le suivant :

```
STRCPY(nom_de_variable, valeur_texte);
```

Où :

- nom_de_variable est une variable déclarée comme un tableau de caractères,

- Valeur_texte correspond à la valeur que l'on veut assigner à la variable.

Exemple :

```
Char Texte[30] ;
STRCPY( Texte, "Ceci est un exemple");
```

Second cas : Pointeur vers un ensemble de caractères

Déclaration :

```
Char *Nom_du_texte ;
```

Exemple :

```
Char *C1 ;
C1 = "ceci est un exemple" ;
```

Remarques :

- Dans le premier cas, la place nécessaire pour la représentation de la variable dans la mémoire de l'ordinateur est réservée définitivement lors de la déclaration.
- Dans le second cas, la place n'est réservée qu'au moment où l'on assigne une valeur à la variable.
- Toute chaîne est terminée par un caractère spécial que le langage C ajoute automatiquement. Il s'agit du caractère de contrôle **/0**.

5. LES INSTRUCTION D'ENTREES / SORTIES

5.1. Introduction

Le but général d'un programme est de traiter un certain nombre de données et de produire des résultats.

Les données sont fournies au programme via des instructions d'entrée et les résultats sont répercutés vers le monde extérieur via des instructions de sortie.

Ces instructions d'entrée / sortie ou I / 0 (Input / Output) transmettent donc des informations entre la mémoire de l'ordinateur et un support extérieur.

Ce support peut être dans le cas d'une instruction d'entrée :

- le clavier,
- un fichier disque,
- une souris,
- un instrument de mesure quelconque,
- …etc.

Dans le cas d'une instruction de sortie, le support peut être :

- l'écran,
- un fichier disque,
- une imprimante,
- …etc.

En Langage C, les entrées / sorties se font par l'intermédiaire de fonctions standards.

5.2. La fonction SCANF

Cette fonction lit une liste de variables selon un format donné.

Structure :

Scanf(format, liste_adresses) ;

Où :

- format est une chaîne de caractères, composée de spécifications qui indiquent le type de variables que l'on va lire,
- liste_adresses est la liste des adresses des variables à lire. On doit mettre l'opérateur & devant chaque variable à lire.

Chaque spécification correspond à une variable et doit donc apparaître dans la même ordre que la variable dans la liste.

Les principales spécifications sont :

%d variable de type INT
%u variable de type UNSIGNED INT
%f variable de type FLOAT
%e variable de type FLOAT mise sous la forme scientifique

%c variable de type CHAR
%s variable de type TEXTE

Remarques :

- On peut placer entre le signe % et la lettre la longueur de la variable. Ainsi %3d indique que l'on va lire un entier de 3 chiffres,
- Un blanc mis entre deux spécifications de format signifie que l'on peut mettre autant de blancs ou de lignes que l'on veut entre les valeurs à lire,
- Si l'on sépare les spécifications par une virgule alors les valeurs à lire seront séparées par des virgules,
- Lorsque la liste des données entrées est complète, pour signaler la fin de cette liste, il faut enfoncer la touche Entrée,
- Le format peut être une chaîne explicite ou une variable dont la valeur représente le format souhaité.

Exemple :

```
Int i ;
Float r;
Char c ;

Char *Form;
Scanf( "%d %f %c” , &i,&r,&c );
scanf( "%2d, %f", &i,&r );
Form = "%d %f %c";
Scanf(form, &i, &r,&c );
```

5.3. La fonction GETS

Cette fonction lit un chaîne de caractères.

Exemple :

```
Char CH[12] ;
Gets(CH) ;
```

5.4. La fonction GETCH( )

Cette fonction lit un caractère unique.

Exemple :

```
Char C ;
C = GETCHAR( ) ;
```

5.5. La fonction PRINTF

Cette fonction écrit des informations selon un format donné.

Structure :

```
Printf( format , liste_items );
```

Où :

- format peut être composé de la même manière que pour la commande SCANF
- liste_items est :
  - soit une variable
  - soit une valeur

Remarques :

- Dans la chaîne format, on peut intercaler du texte qui sera écrit tel quel,

- Pour éviter toute confusion avec une spécification de format, lorsque l'on doit écrire le caractère "%" explicitement, on met "%%" dans la chaîne,
- On peut insérer également des séquences spéciales appelées séquences escape qui permettent de contrôler l'impression des données.
  Les principales séquences escape sont :

  \n : passage à la ligne suivante
  \b : déplacement du curseur d'un caractère vers la gauche
  \f : déplacement à la page suivante
  \t : déplacement du curseur d'un tabulation vers la droite
  \v : déplacement du curseur d'une tabulation vers le bas
  \a : déclenchement d'un bip sonore

- Comme pour les spécifications, si l'on désire insérer le caractère "\" dans la chaîne à imprimer, on double le caractère, c'est à dure que l'on met "\\".

Exemple :

```
Int i ;
Printf( " Entrez la valeur de i : \n") ;
Scanf(" %d", &i );
i = i * 2;
Printf( "valeur de i * 2 = %d", i );
```

5.6. La fonction PUTS

Cette fonction écrit une chaîne de caractères.

Structure :

PUTS (Nom_de_chaîne)

Lorsque l'écriture est effectuée, la fonction force également un passage à la ligne.

D'où :

Puts("Ceci est un exemple");

Est équivalent à

Printf("Ceci est un exemple \n");

5.7. La fonction PUTCHAR

Cette fonction écrit un caractère.

Structure :

Putchar(item)

Où : item est une variable de type caractère ou une valeur caractère.

Exemple :

```
Char C;
C = 'a';
Putchar( C );
Putchar( 'B');
```

5.8. Exercices

Voir Série de TD N° 1

6. LES INSTRUCTIONS CONDITIONNELLES

6.1. L'instruction IF

Cette instruction permet de tester une condition et d'exécuter une instruction en fonction du résultat.

Structure :

```
IF ( expression )
        Instruction1;
ELSE
        Instruction2
```

Où :

expression est une expression conditionnelle.

Cette instruction teste la valeur de l'expression. Si celle-ci est vraie (non nulle) alors instruction1 est exécutée, sinon c'est instruction2 qui est exécutée.

Remarques :

- La partie ELSE n'est pas obligatoire,

- Les instructions1 et instructions2 peuvent être des instructions simples ou des blocs d'instructions encadrés par des accolades.

Exemple :

```
Int I , J;
I = 9 ;
J = 7 ;
IF ( I > J )
        Printf("I est supérieur à J  \n");
ELSE
        Printf("J est supérieur à I  \n");
```

Imbrications des IF

```
IF ( condition1)
        IF ( condition 2)
        {
                instructions;
        }
        ELSE
        {
                instructions;
        }
ELSE
{
        instructions,
}
```

Il faut que les structures IF soient complètement imbriquées.

6.2.L'instruction SWITCH

Il peut arriver que selon les différentes valeurs d'une variable, on doive effectuer un traitement précis. Cela peut se faire en imbriquant une série de IF.

Exemple :

```
Int val ;
IF ( val = = 1)
        Traitement1;
ELSE
        IF ( val = =2 )
                Traitement2;
        ELSE
                IF ( val = = 3 )
                        Traitement3;
```

Cette imbrication peut devenir assez fastidieuse si le nombre de valeurs possibles est élevé.

L'instruction SWITCH remplace avantageusement la suite des IF imbriqué.

Structure :

```
Switch(expression )
{
        Case valeur1 : instruction1; Break;
        Case valeur2 : instruction2; Break;
                        . . .    . . .    . . .
        Case valeurN : instructionN; Break;
        Default : instruction; Break;
}
```

Où :

- expression doit être du type entier, caractère ou tout autre type énumératif (c'est à dire que toutes ses valeurs possibles sont connues). Elle ne peut donc être de type réel ou string,
- instruction1, …, instructionN peuvent être des instructions simples ou des blocs d'instructions.

Les instructions Break et l'option Default ne sont pas obligatoires.

Rôle de l'instruction SWITCH

- Elle détermine la valeur de l'expression,
- Elle compare cette valeur aux différentes valeurs mentionnées après le CASE :
  Si la valeurI est égale à l'expression, alors l'exécution du programme continue à partir de l'instructionI.

  Si aucune valeurI n'est égale à l'expression , alors deux cas peuvent se présenter :

  - soit on a mentionné l'option Default, alors le programme continue son exécution à partir de l'instruction Default,
  - soit on n'a pas précisé cette option et le programme ignore purement et simplement l'instruction Switch.

- Dans le cas où l'on a exécuté une instructionI (I compris entre 1 et N) :
  - si on a placé un Break après cette instruction alors le programme reprend son exécution à partir de l'accolade } délimitant le bloc Switch,

- si le Break n'est pas mentionné alors les instructions correspondant aux Cases suivants seront exécutées, jusqu'à ce que l'on rencontre un Break ou jusqu'à la fin du Switch.

Exemple :

```
Char choix ;
Choix = Getchar( );
Switch( choix )
{
        Case  'A'       : traitement_A;
        Case  'B'       : traitement_B ; Break;
        Case  'C'       : traitement_C ; Break;
        Default         : traitement-D ;
}
```

6.3. Exercices

Voir Série de TD N° 2

7. LES INSTRUCTIONS REPETITIVES

7.1. Introduction

Dans un programme, il arrive fréquemment qu'une séquence d'instructions doive être répétée un certain nombre de fois. Cela se réalise au moyen d'instructions répétitives.

En général, le nombre de répétitions dépend de la réalisation ou non d'une condition.

En langage C, il existe 3 types d'instructions répétitives:

- l'instruction WHILE ,
- l'instruction FOR ,
- l'instruction DO…WHILE.

7.2.L'instruction WHILE

Structure :

```
While( expression )
        Instructions
```

Où :

- expression est une expression conditionnelle qui représentera la condition d'arrêt de la boucle,
- Instructions est une instruction simple ou un bloc d'instructions.

Fonctionnement de cette boucle

1. on évalue l'expression,
2. si l'expression est vraie, alors on exécute instructions et on recommence en 1,
3. si l'expression est fausse, le programme poursuit son exécution à partir de l'instruction suivante.

Exemple :

```
Char C ;
C = Getchar( ) ;
While ( C ! = 'F' )
{
…
}
```

Remarques :

- l'expression conditionnelle peut être fausse lors de la première évaluation, dans ce cas, le programme ignore le contenu de l'instruction WHILE et passe à l'instruction suivante,
- Il faut veiller à ce que la condition devienne fausse pour pouvoir sortir de la boucle. Dans le cas contraire, on obtient une boucle infinie.

7.3. L'instruction FOR

Cette instruction est utilisée dans le cas où le nombre de répétition de la boucle est connu avant son exécution.

Structure :

```
For ( expression1; expression2; expression3 )
        Instructions;
```

Où :

- expression1 est l'initialisation d'une variable de contrôle appelée aussi index,
- expression2 est une expression conditionnelle définissant la condition d'arrêt de la boucle,
- expression3 est une expression modifiant l'index,
- instruction est une instruction simple ou un bloc d'instructions.

Exemple :

```
Int I;
For ( i = 1; i <= 10; i++ )
        Printf( " %d", i )
```

Remarque :

expression1, expression2 et expression3 peuvent être des expressions multiples. Les expressions qui les composent doivent alors être séparées par des virgules.

Exemple :

```
For ( i = 1, j = 1; i <= 10, j <= 10; i++, j++ )
```

7.4. L'instruction DO . . . WHILE

Structure :

```
DO
        Instructions
WHILE ( expression )
```

Où :

- instructions est une instruction simple ou un bloc d'instructions,
- expression est une expression conditionnelle.

Le principe de fonctionnement est similaire à une boucle WHILE. Mais ici, l'instruction est exécutée **AU MOIS UNE FOIS** avant que l'on évalue l'expression conditionnelle.

Exemple:

```
Char c;
DO
{
        Getch( c );
        Putch( c );
} While ( c != 'F' );
```

7.5. Exercices

Voir Série de TD N° 3

8. LES FONCTIONS

8.1. Caractéristiques des fonctions

Dans un programme, on est amené souvent à répéter plusieurs fois les mêmes opérations à différents endroits et avec des données différentes.

Pour éviter de devoir réécrire chaque fois cette suite d'opérations, la plupart des langages de programmation permettent de définir des sous-programmes qu'il suffit d'appeler aux endroits voulus.

Ces sous programmes sont aussi très utiles lorsqu'on a un problème très complexe à traiter. En effet, il est plus facile de le découper en sous-problèmes et d'écrire un sous-programme pour chacun d'eux. Il suffit ensuite de rassembler l'ensemble dans un programme principal.

En langage C, TOUS LES SOUS-PROGRAMMES SONT DES FONCTIONS

Chaque fonction est caractérisée par :

- Un nom : un identificateur qui désignera la fonction lors de la déclaration, lors de la définition et lors des différents appels,
- Un type, qui correspond au type de la valeur calculée par la fonction. En effet, une fonction fournit une valeur résultat, qui est utilisable par le programme appelant,
- Des paramètres : qui pourront prendre des valeurs différentes lors de chaque appel,
- Un ensemble d'instructions : constituant le traitement à effectuer par la fonction. Cet ensemble est appelé le corps de la fonction.

8.2.Déclaration d'une fonction

La déclaration se fait de la manière suivante :

```
Type_de_fonction  Nom ( paramètres) ;
```

Où :

- Type_de_fonction représente le type de la valeur calculée par la fonction,
- Nom correspond au nom désignant la fonction,
- Paramètres est une liste contenant des informations sur les paramètres. On y trouve, pour chacun d'eux, son type et en option son nom.

Exemples :

```
INT max ( Int i , Int j);
Float min( Float, Float);
```

Remarques :

- On précise le type de chaque paramètre, pour que le compilateur puisse effectuer un contrôle lors des appels de la fonction et détecter toute erreur de type éventuelle,
- La déclaration d'une fonction n'est pas obligatoire mais conseillée. Si on ne déclare pas une fonction et si on l'appelle avant sa définition, alors le compilateur C considère que, par défaut, le type de la valeur retournée est INT,
- Le type particulier VOID permet de préciser que la fonction ne retourne pas de valeur.

8.3.Définition d'une fonction

Le format de définition est le suivant :

```
Type_de_fonction nom ( paramètres )
{
        déclarations locales ;
        instructions;
}
```

où :

- paramètres représente la liste des noms des paramètres,

Remarques :

- Il n'y a pas de point virgule après la parenthèse")",
- Les déclarations locales peuvent être des déclarations de type, de constantes, ou de variables. Ces variables qui sont déclarées à l'intérieur de la fonction sont locales à celle-ci, c'est à dire qu'elles ne peuvent être utilisées que par les instructions de la fonction. Elles constituent en

quelque sorte les variables de travail pour la fonction et en dehors de celle-ci on ne peut y accéder,

- Par contre, les variables qui sont déclarées en dehors d'une fonction sont globales à tout le programme, c'est à dire qu'après leurs déclarations, elles peuvent être utilisées à l'intérieur de chaque fonction, y compris, bien sûr, la fonction MAIN( ),
- Si la fonction retourne une valeur, l'instruction **RETURN (expression)** doit être une des instructions de la fonction,
- On ne peut pas imbriquer les définitions de fonctions.

Exemple :

```
Int max ( Int  i, Int j )
{
        Int diff ;
        diff = i - j
        If (diff >= 0)
                Return( i )
        Else
                Return( j );
}
```

8.4.Appel d'une fonction

L'appel d'une fonction se fait de la façon suivante :

```
Nom_de_la_fonction( liste des paramètres ) ;
```

La liste des paramètres correspond à la liste des paramètres lors de la définition.

Au moment de l'appel, les valeurs ces paramètres sont assignées aux paramètres de la définition. Ce type de passage de paramètres est appelé transmission par valeur.

Une des conséquences de ce type de passage est que l'on ne peut pas modifier la valeur d'un paramètre à l'intérieur de fonction.

Nous verrons plus lois dans le cours comment, avec les pointeurs, on peut faire pour modifier un paramètre à l'intérieur d'une fonction. On appelle ce type de passage : transmission de données par adresses.

Exemple :

```
Int k, I;
I = MAX (k,1);
Printf("Le maximum est : %d",I);
```

8.5. Exercices

Voir Série de TD N° 6

9. LES VARIABLES DIMENSIONNEES

9.1. Introduction

Certains programmes nécessitent parfois l'utilisation de nombreuses variables d'un même type. Plutôt que de donner un nom différent à chacune de ces variables, on peut utiliser une seule variable mais à composantes multiples, appelée variable dimensionnée ou tableau.

Chaque composante peut être représentée en faisant référence au nom de la variable suivi d'un indice mis entre crochets [].

L'indice représente la position de la composante dans la variable.

9.2. Tableaux à une dimension

La déclaration d'une telle variable doit se faire de la façon suivante: **Type nom[taille] ;**

Où :

- Nom doit correspondre au nom de la variable dimensionnée,

- Type est un type quelconque représentant le type de composantes,
- Taille est le nombre de composantes que va contenir la variable,

ATTENTION:
L'indice de la première composante est 0 et la dernière composante
correspondra à nom [taille-1]

Exemple :

```
Int table[20], i;
For (i = 0; i <= 19;i++)
        Scanf("%d",&table[i]);
```

Remarques :

- Un indice peut être une variable, une constante ou une expression. ,
- Un tableau à une dimension est aussi appelé vecteur,
- Lors de la déclaration, il est possible d'initialiser le tableau en procédant de la façon suivante :

```
Type nom [ taille ] = { liste de valeurs };
```

Exemple:

```
INT tab [3] = { -1,2,90 }
```

Cela correspond à 3 assignations ;

```
Tab [0] = -1;  Tab [1] = 2; Tab [2] =90;
```

- Dans le même ordre d'idée, il est possible de définir des vecteurs de constantes, c'est à dire que la valeur de chaque composante restera invariable dans toute l'exécution du programme. un tel vecteur pourra être déclaré de la manière suivante:

  CONST type nom [taille] = { liste de valeurs};

Exemple :

Const Int min_max [2] = { -32768 , 32767 };

9.3. Tableaux multidimensionnés

La déclaration d'une telle variable doit être de la forme suivante:

Type nom [taille1] [taille2]…[tailleN]

Où :

Les taillesI représentent le nombre de composantes associées à la Ième dimension.

Tout élément du tableau pourra être référencé par :

Nom [indice 1][indice 2]…[indiceN]

Exemple :

Char ecran [25] [80] ;

Cette commande définit une variable à deux dimensions contenant 25 lignes et 80 colonnes.

Ecran [14][2] représente la 3ème colonne de la 15ème ligne.

Remarques :

- Le nombre total d'éléments du tableau est égal à taille1*taille2*…*tailleN,
- Un tableau à plusieurs dimensions est aussi appelé matrice,
- Les éléments sont stockés en mémoire ligne par ligne,
- L'initialisation d'une matrice peut se faire comme pour un vecteur lors de la déclaration.

Exemple :

Int mat [3] [5] = { {15,6,9,104,26},{12,13,67,90,-98},{0,8,1,0,15}}

9.4. Tableau et fonctions

Il est possible d'utiliser une variable dimensionnée comme paramètre d'une fonction . Dans ce cas, on n'est pas obligé de connaître la taille exacte du tableau lors de la déclaration et lors de la définition de la fonction.

Cette taille pourra être un paramètre à part entière de la fonction. Elle pourra donc varier d'un appel à l'autre.

Exemple : Int min(int table[], int taille)

Remarques :

- On pourra modifier le tableau à l'intérieur de la fonction. En effet, lors de l'appel de la fonction, ce n'est pas tout le tableau qui est transmis mais son adresse. Notons que l'adresse d'un tableau est l'adresse de son premier élément,

- Si on modifie un élément du tableau, son adresse exacte dans le programme principal sera calculée et la modification sera directement faite à cet endroit,
- Nous venons de rencontrer un premier cas de la transmission de données par adresses.

9.5. Exercices

Voir Série de TD N° 4 et 4 Bis

10. L'ALLOCATION DYNAMIQUE ET LES POINTEURS

10.1. Introduction

Toutes les variables rencontrées jusqu'à maintenant étaient des variables statiques, c'est à dire que lors de leurs déclarations, on réservait immédiatement la place nécessaire pour stocker les valeurs qu'elles allaient contenir.
La taille d'une variable ne pouvait donc varier au cours de l'exécution du programme. Il en était de même pour les tableaux car il n'était pas possible de modifier le nombre de composantes en cours d'exécution.
De plus, il peut arriver que le nombre de variables nécessaires dans un programme puisse varier d'une exécution à l'autre.
Il est donc utile d'avoir ce qu'on appelle des variables dynamiques.

10.2. Définitions

Une variable dynamique est une variable que l'on peut créer et détruire quand on veut, pendant l'exécution d'un programme.

On n'accède pas directement à ces variables par leurs noms, mais par l'intermédiaire de pointeurs.

Un pointeur est une variable pouvant contenir l'adresse d'une autre variable.

En modifiant l'adresse que contient un pointeur, on peut donc accéder à différents endroits de la mémoire et par conséquent à différentes variables.

10.3. Déclaration d'un pointeur

Structure :

Type *nom_du_pointeur;

Où :

- nom du pointeur est le nom de la variable pointeur,
- Type est le type de la variable dont le pointeur va contenir l'adresse.

Exemple : Int *ptr;

ptr est un pointeur qui contiendra l'adresse d'un emplacement mémoire pouvant contenir une valeur de type int.

La valeur pointée pourra être référencée par : *nom_du_pointeur

Remarques:

- Il faut toujours assigner une adresse à un pointeur avant de pouvoir le manipuler. Cela peut se faire de DEUX manières différentes :

- en utilisant l'opérateur "adresse de" : &,
- en utilisant l'allocation dynamique.

10.4. Utilisation de l'opérateur "&"

Cette assignation peut se présenter de la manière suivante :

Nom_du_pointeur = &variable;

Où :

Nom_du_pointeur doit être déclaré comme étant un pointeur vers le type de la variable.

Exemple :

```
Int *ptr, i;
I = 1;
ptr = &i;
```

Après la deuxième assignation, ptr et i représentent deux noms différents pour un même emplacement mémoire.

Tant que ptr contiendra l'adresse de i, on pourra utiliser invariablement i ou ptr pour faire référence à la valeur que contient cet emplacement

Les deux assignations suivantes seront donc équivalentes :

```
i = *ptr + 1;
*ptr = i +1;
```

10.5. L'allocation dynamique

Cette allocation permet de créer une ou plusieurs variables au cours de l'exécution du programme.

Cette allocation peut se faire de la façon suivante :

```
Nom_pointeur = (type_pointé*) MALLOC(SIZEOF(type_pointé));
```

Où :

- Nom_pointeur est le nom du pointeur auquel on veut assigner une valeur,
- Type_pointé est le type de la variable dont le pointeur va contenir l'adresse,
- SIZEOF(type_pointé) est une fonction prédéfinie qui retourne comme valeur la taille en octets qu'occupe une variable de type : type_pointé. Ainsi, SIZEOF(INT) retournera comme valeur 2,
- MALLOC (expression numérique) est une fonction qui réserve un nombre d'octets consécutifs égal à l'expression numérique. Cette fonction retournera comme valeur l'adresse du premier octet réservé.

L'expression fonctionne de la façon suivante :

- Elle détermine la taille nécessaire pour représenter la variable pointée,
- Elle réserve le nombre d'octets nécessaire pour cette représentation. Les octets réservés font partie de la mémoire encore disponible,
- Elle assigne l'adresse du premier octet réservé au Non_pointeur.

Exemple :

```
Char *ptr ;
ptr = (Char*) MALLOC (SIZEOF (Char)) ;
```

ptr représente une nouvelle variable de type char, on pourra donc la manipuler comme une variable normale.

```
*ptr = 'a' ;
```

10.6. Libération d'une variable dynamique

On peut à tout moment libérer la place mémoire occupée par une variable dynamique, qui est appelée par un pointeur. Pour cela, il suffit de faire appel à la fonction standard **FREE**.

Structure :

```
FREE (nom_du_pointeur) ;
```

Où :

- nom du pointeur doit contenir le nom du pointeur par lequel on accède à la variable à libérer.

Exemple :

```
Int *p;
...
p = (Int*)MALLOC(2)
…
FREE(p);
```

10.7. Pointeurs et variables dimensionnées

Une autre fonction peut être utilisée pour l'allocation dynamique. Il s'agit de la fonction CALLOC.

Cette fonction est particulièrement utilisée pour la création de variables dimensionnées dont la taille peut être variable.

L'appel à cette fonction doit se faire de la façon suivante :

CALLOC (nombre, taille) ;

Où :

- nombre est le nombre de variables que l'on veut créer,
- taille est la taille d'une variable en octets.

L'accès à l'une de ces variables pourra toujours se faire par l'intermédiaire d'indice, la notation est cependant modifiée.

Pour faire référence à la composante occupant la position i, il faudra écrire :

*(nom_du_pointeur + i)

Cette instruction est équivalente à : nom [i].

Exemple :

```
Int *table, i;
table = (Int T*) CALLOC (3,2);
```

On s'est ainsi réservé de la place mémoire pour 3 variables entières de 2 octets.

```
*(table) = 0;
*(table +1) = 1;
*(table + 2) = 144;
```

Remarques :

- Il peut arriver que lors d'une tentative d'allocation dynamique avec les fonctions MALLOC ou CALLOC, il n'y ait plus de place mémoire disponible, Les deux fonctions renvoient alors une valeur spéciale dans le pointeur. Il s'agit de la valeur NULL.
- Cette valeur NULL peut également servir à initialiser un pointeur. Elle indique tout simplement que le pointeur ne pointe vers rien.

10.8. Pointeurs et fonctions

Lorsque nous avons évoqué la transmission de paramètres par valeurs lors de l'appel d'une fonction, nous avons vu qu'il était impossible de modifier un paramètre dans le corps de la fonction. En utilisant les variables pointeurs, cela devient possible.

En effet, à la déclaration et à la définition de la fonction, il suffit de déclarer le paramètre comme étant un pointeur.

Lors de l'appel, on passera alors l'adresse de la variable que l'on veut voir modifier par la fonction.

Cette adresse sera assignée au pointeur et toute référence à la variable pointée par *nom_du_pointeur correspondra à une référence au paramètre dont on veut modifier la valeur.

Exemple :

- **Création de la fonction minmax :**

```
void minmax( int i, int j, int *max, int *min )
```

```
{
        if ( i >= j )
        {
            *max = i;
            *min =j;
        }
        else
        {
            *max = j;
             *min = i;
        }
}
```

- **Programme d'appel :**

```
Main( )
{
        int nb1, nb2, maxi, mini;
        Scanf(" %d  %d ", &nb1,&nb2 );
        minmax(nb1, nb2, &maxi, &mini );
        printf( " %d  %d ", mini, maxi );
}
```

Lors de l'appel de la fonction minmax, on passe l'adresse de maxi et celle de mini qui sont les paramètres modifiés dans le corps de la fonction. Cela se passe alors comme si on avait les deux assignations suivantes:

```
max = &maxi;
min = &mini;
```

max et maxi représentent alors le même emplacement mémoire.

11. LES VARIABLES STRUCTUREES

11.1. Introduction

Nous avons vu que pour les variables dimensionnées, toutes les composantes devraient être de même type. Nous allons maintenant décrire un type de variables dont les composantes pourront être de différents types. Ces variables sont appelées variables structurées.

11.2. Généralités

Considérons l'exemple suivant:

Une personne possède un certain nombre de caractéristiques:

- Un nom,
- Un prénom,
- Une adresse,
- Un age,
- . . .

On regroupe ces caractéristiques dans une variable structurée. On pourra accéder à la variable globalement ou par l'intermédiaire de l'une de ses composantes.

Les composantes d'une variable structurée sont appelés champs. On n'y accède plus via un indice, mais directement par leurs noms.
Pour utiliser une variable structurée il faut :

- définir un type structuré,
- déclarer une variable comme étant de ce type.

11.3. déclaration d'un type structuré

Structure:

```
TYPEDEF STRUCT
{
        Déclaration champs 1;
        Déclaration champs 2;
        . . . . .
        Déclaration champs N;
} Nom du type;
```

Où :

- Nom type est le nom que l'on donne au type structuré,
- Déclaration champs 1, . . . , Déclaration champs N : représentent les déclarations des différents champs qui vont composer la variable structurée.
  Ces déclarations vont être de la forme :
  Type_du_champs Nom_du_champs;

Exemple :

```
TYPEDEF STRUCT
{
        Char nom[25];
        Char adresse[30];
        Int age;
        Float taille;
} personne;
```

On vient de définir un type structuré "personne".
Une variable de ce type aura 4 composantes : nom, adresse, age et taille.

11.4. Déclaration d'une variable structurée

Cette déclaration est semblable à toute autre déclaration, c'est à dire :

```
Type_structuré variable;
```

Ou bien

```
STRUCT nom du type
{
    déclaration des champs;
} Variable;
```

Dans le deuxième cas, le nom du type est facultatif.
On peut donc déclarer directement une variable comme étant structurée sans définir auparavant un type structuré.

Exemple:

```
Personne P1;
```

Ou bien

```
STRUCT personne
{
    Char nom[25];
    Char adresse[30];
    Int age;
    Float taille;
} P1;
```

11.5. Utilisation d'une variable structurée

Toute référence à un champs de variable structurée doit avoir la forme suivante :

```
Nom_variable.nom_champ;
```

Exemple :

```
P1.age = 25;
P1.taille = 1.75;
```

On peut également assigner une variable structurée à une autre variable structurée de même type. La variable est alors manipulée globalement.

Exemple :

```
Personne P1, P2;
. . .
P1 = P2;
```

11.6. Exercices :

Voir Série de TD N° 5

12. LES FONCTIONS MATHEMATIQUES EN LANGAGE C

Fonction	Include	résultat
Int ABS (int nombre)	Stdlib.h	Revoie la valeur absolue de nombre
Double ACOS (double x)	Math.h	Retourne l'arc cosinus de x
Double ASIN (double x)	Math.h	Retourne l'arc sinus de x
Double ATAN (double x)	Math.h	Retourne l'arc tangente de x
Double ATAN2 (double y, double x)	Math.h	Retourne l'arc tangente de y/x

Double CABS (struct complex z)	Math.h	Retourne la valeur absolue d'un nombre complexe
		STRUCT complex
		{
		double x; /* partie réelle */
		double y; /* partie imaginaire */
		}

Double CEIL (double x)	Math.h	Arrondit un nombre à l'entier immédiatement supérieur
Double COS (double x)	Math.h	Retourne le cosinus de x
Double COSH (double x)	Math.h	Retourne le cosinus hyperbolique de x
DIV (int num , int div)	Stdlib.h	Divise le numérateur num par le dénominateur div. Retourne dans une structure de type div_t le quotient est le reste. Typedef struct { int quot; /* quotient */ int rem; /* reste */ }div_t;
Double EXP (double x)	Math.h	Retourne l'exponentielle de x
Double FABS (double x)	Math.h	Retourne la valeur absolue d'un nombre réel
Double FLOOR (double x)	Math.h	Arrondit un nombre à l'entier immédiatement inférieur
Double FMOD (double x , double y)	Math.h	Retourne x modulo y (c_a_d le reste de la division de x par y)
Double HYPOT (double x , double y)	Math.h	Calcule la langueur de l'hypoténuse d'un triangle d'après la formule Sqrt(x*x + y*y)
Long LABS (double x)	Stdlib.h	Retourne la valeur absolue des variables de type "long"

Double LDEXP (double x , int exp)	Math.h	Elévation d'un nombre x à la puissance exp

1LDIV (long num , long div)	Stdlib.h	Division de deux entiers de type " long ".
		Retourne le quotient et le reste dans une structure de type:
		Typedef struct
		{
		long quot; /* quotient */
		long reste; /* reste */
		}

Double LOG (double x)	Math.h	Calcule le logarithme népérien (ln) de x
Double LOG10 (double x)	Math.h	Calcule le logarithme décimal de x
MAX (a , b)	Stdlib.h	Retourne la plus grande des deux valeurs a et b.
MIN (a , b)	Stdlib.h	Retourne la plus petite des deux valeurs a et b.
Double POW (double x , double y)	Math.h	Retourne un nombre élevé à la puissance
Double SIN (double x)	Math.h	Retourne le sinus de x
Double SINH (double x)	Math.h	Retourne le sinus hyperbolique de x
Double SQRT (double x)	Math.h	Retourne la racine carrée de x (x >= 0)
Double TAN (double x)	Math.h	Calcule la tangente de x
Double TANH (double x)	Math.h	Retourne le tangente hyperbolique de x