Ministère de l'Enseignement Supérieur,
de la Recherche Scientifique et de la Technologie
Direction Générale des Etudes Technologiques
Institut Supérieur des Etudes Technologiques de Djerba

SUPPORT DE COURS

PROGRAMMATION I

Élaboré par :

Anis ASSÈS

Mejdi BLAGHGI

Mohamed Hédi ElHajjej

Mohamed Salah Karouia

Public cible :

Informatique (Niveau I)

Version 2007

PROGRAMMATION I

AVANT-PROPOS

Objectifs généraux

Il s'agit d'un cours d'initiation au langage C. Il a pour but de :

-découvrir l'environnement de programmation C,

-savoir réagir devant un problème de programmation.

Public Cible

Ce cours est destiné essentiellement aux étudiants du premier niveau des Etudes Supérieures Technologiques, Option Informatique.

Volume horaire

Ce cours est présenté, de manière hebdomadaire, comme suit :

-1h 30mn du Cours Intégré

-3h de Travaux Pratiques pour chaque groupe

Soit en total : 22,5h de Cours Intégré et 45h de Travaux Pratiques

Moyens pédagogiques

-Tableau

-Micro-ordinateurs équipés de Borland C ou Turbo C

-Polycopiés de Travaux Dirigés et Travaux Pratiques

Evaluation

-Coefficient : 3

-Devoir Surveillé, moyenne des tests 30%

-Examen TP, moyenne des TPs (comptes rendus) 30%

-Examen final 40%

TABLE DES MATIERES

PROGRAMMATION I

CHAPITRE : 1

INTRODUCTION AU LANGAGE C

Objectifs spécifiques

-Introduire l'informatique

-Introduire les langages de programmation

-Introduire le langage C

-Connaître la structure générale d'un programme C

Plan du chapitre

I. Généralités

II. Caractéristiques générales du langage C

III. Etapes d'exécution d'un programme C

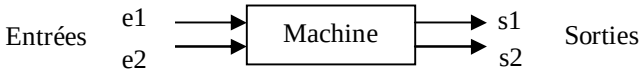
Volume horaire

1 séance de cours intégré

I. GENERALITES

I.1. Informatique

C'est une science qui permet de traiter des informations en utilisant des machines appelées : ordinateurs.



I.2. Architecture d'un ordinateur

Un ordinateur est une machine qui permet de stocker des données (informations) structurées et de les traiter à la demande de l'utilisateur, afin de produire un résultat voulu.

Le terme « ordinateur » a été proposé en 1955 par J. Perret à la demande d'IBM, pour remplacer le terme « Calculateur ».

Les ordinateurs sont distingués soit par leurs tailles (micro / mini / mainframe), leurs domaines d'applications (terminal / serveur / station de travail) ou leurs interopérabilités (système d'exploitation ouvert, MS-DOS, MacOS, etc.).

Les constituants de base d'un ordinateur sont :

- Unité de traitement et de commande :
    - C'est la partie intelligente de la machine qui prend en charge l'exécution des programmes. Le processeur, le « chef d'orchestre » de l'ordinateur, est lui qui gouverne tout l'ordinateur.
    - Traitements : opérations de calcul arithmétiques et logiques.
    - Caractéristiques du microprocesseur : taille, fréquence de l'horloge.
    - Premier microprocesseur utilisé par IBM s'appelle 8086. Il est composé de registres (8 bits). -> 80286 (16 bits, 12-16 Mhz), 80386 (32 bits, 20-33 Mhz), 80486 (33-100 Mhz), Pentium, Pentium II, …

- Unité mémoire
    - Cette unité est représentée par un nombre défini de cellules (ou de composants).
    - A chaque cellule ou composant élémentaire de la mémoire correspond un bit (1 ou 0).
    - 1 octet correspond à 8 bits consécutifs.
    - 1 Ko correspond à 1024 octets.
    - 1 Mo correspond à 1024 Ko.

- Système de codification des caractères. Exemple : ASCII (American Standard Code for Information Interchange). Il dispose d'une table de codification de caractères (un code étant une loi de correspondance de caractères). (Voir Annexe)
- Supports mémoires : disques durs, disquettes, CD-ROM, DVD-ROM, Flash Disk …

- Unité d'échange
  - Elle permet de gérer les échanges d'informations entre la partie interne de l'ordinateur et la partie externe.
  - Les périphériques ou les organes externes : clavier, écran, imprimante, …

I.3. Système d'exploitation

- Un système d'exploitation est un ensemble de programmes qui permettent d'exploiter les possibilités matérielles d'un ordinateur.
- A chaque système d'exploitation correspond un nombre défini de commandes respectant une syntaxe appropriée.
- On distingue différents types de systèmes d'exploitation :
  - Systèmes d'exploitation mono-postes, mono-tâches : un seul poste de travail est reconnu, une seule tâche est permise. Exemple : DOS.
  - Systèmes mono-postes, multi-tâches : un seul poste est reconnu. Les tâches peuvent être activées en parallèle. C'est à travers le partage de temps que le Système d'exploitation peut exécuter plusieurs programmes au même temps. Exemples : OS2, Windows 95.
  - Systèmes multi-postes, multi-tâches : plusieurs postes de travail sont reconnus. Chaque poste peut exécuter plusieurs tâches. Exemple : Unix.

I.4. Langages de programmation

Un programme est un ensemble d'instructions (ordres machines) qui aboutissent à l'exécution d'un traitement souhaité.

- Langage machine : c'est un langage de programmation qui utilise des instructions machine (instruction directement exécutée par l'unité de traitement (processeur)). Chaque unité de traitement possède un catalogue d'instructions machine.
- Assembleur : c'est un langage qui utilise des mots clés en langage pseudo-naturel pour remplacer les codifications utilisées dans les langages machine.

- Langage évolué : apporte une simplification au niveau de la programmation en intégrant des instructions qui se rapprochent du langage naturel et scientifique. Exemples : Fortran, Cobol, Basic, Pascal, C, …

II. CARACTERISTIQUES GENERALES DU LANGAGE C

II.1. Caractéristiques

- Créé dans les années 70 par Denis-Ritchie / Labo Bell Telephone Laboratories. [1]
- Conçu pour l'écriture du système d'exploitation UNIX.
- C'est l'intermédiaire entre langage haut niveau (indépendamment des machines) et langage bas niveau (assembleur).
- C'est un langage évolué (proche du langage naturel et scientifique), procédural (modulaire), portable (réutilisé et réintégrable dans plusieurs plates formes).

II.2. Structure générale d'un programme C

Un programme C s'écrit généralement sous la forme suivante :

```
/* Déclaration des bibliothèques */
#include<nomfichier1.h>
#include<nomfichier2.h>
.
.
/* Déclaration des constantes */
#define cons1 val1
#define cons2 val2
.
.

/* Déclaration des fonctions */
type-de-retour  Nom-fonction1(type1 arg1, ... , typeN argN)
{/* Début du corps de la fonction1 */
 instruction1 ;
 instruction2 ;
 ..
}/* Fin du corps de la fonction1 */
type-de-retour Nom-fonction2(type1 arg1, ... , typeN argN)
 /* Début du corps de la fonction2 */
 instruction1 ;
 instruction2 ;
 ..
}/* Fin du corps de la fonction2 */
.
.
/* Fonction principale */
type-de-retour main( )
{
 /* Déclaration des variables */
 type-variable1 variable1 ;
 type-variable2 variable2 ;
 .
 .
 /* Corps du programme */
 instruction1 ;
 instruction2 ;
 ..
}
```

Remarques :

- Une bibliothèque est un fichier comportant un ensemble de fonctions (routines) prêtes à l'emploi. Elles peuvent être livrées en standard avec le langage ou créées par le développeur.

  Exemples : stdio.h, math.h, string.h, …
- Chaque instruction en C doit se terminer par un point virgule « ; ».
- Les accolades « { » et « } » correspondent au début et à la fin du corps des fonctions.
- Une fonction peut ne pas contenir de paramètres.
- Les commentaires en C (texte non interprété) sont soit délimités par « /* » et « */ » soit débutés par « // » dans le cas où ils s'écrivent sur une seule ligne.

Exemple :

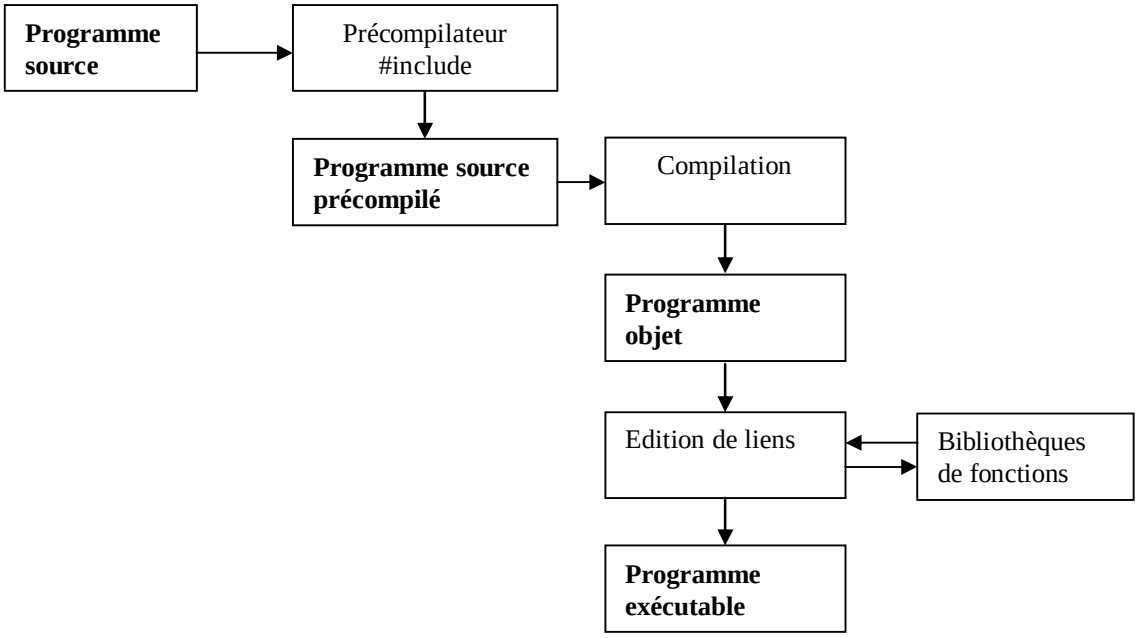
```
/* ceci est un commentaire
sur plusieurs lignes */

// Ceci est un commentaire sur une seule ligne
```

III. ETAPES D'EXECUTION D'UN PROGRAMME C

Le C est un langage compilé (par opposition aux langages interprétés). Cela signifie qu'un programme C est décrit par un fichier texte, appelé fichier source. Ce fichier n'étant évidemment pas exécutable par le microprocesseur, il faut le traduire en langage machine. Cette opération est effectuée par un programme appelé compilateur. [2]

L'exécution d'un programme C passe par les étapes suivantes (Précompilation, Compilation, Edition de liens, Exécution) comme le montre la figure ci-dessous :



Exécution d'un programme C

PROGRAMMATION I

CHAPITRE : 2

ELEMENTS DE BASE

Objectifs spécifiques

-Manipuler les variables et les constantes,

-Manipuler les instructions d'affectation et d'entrée/ sortie,

-Ecrire des programmes C simples.

Plan du chapitre

I. Les variables et les opérateurs

II. Les instructions d'affectation

III. Les fonctions standards d'Entrées/Sorties

IV. Exercice résolu

Volume horaire

2 séances de Cours Intégré

I. LES VARIABLES

Une instruction de déclaration permet de définir une ou plusieurs variables et de leur associer un espace mémoire afin de stocker une ou plusieurs informations.

Syntaxe de déclaration :

```
<type> <identificateur>;
<type> <identificateur1>, <identificateur2>, …, <identificateurN>;
```

<type> :	int	→	entier	(2 octets)
	short	→	entier court	(2 octets)
	long ou long int	→	entier long	(4 octets)
	char	→	caractère	(1 octet)
	float	→	réel	(4 octets)
	double	→	réel ou double précision	(8 octets)

<identificateur> : chaîne de caractères qui définissent le nom de la variable. Ce nom permet d'y faire référence dans le code soit pour récupérer sa valeur soit pour la modifier. Il obéit aux contraintes suivantes :

- Peut être composé des lettres (a-z, A-Z), des chiffres (0-9), du caractère souligné (_),
- Ne peut pas commencer par un chiffre,
- Le langage C différencie les majuscules et les minuscules.

Exemples :

```
char c ;
float x ;
int i, j ;
```

Remarques :

- Le type booléen est déclaré comme étant un entier où la valeur 0 correspond à la valeur « faux ». Toute autre valeur est supposée « vrai »,
- La valeur d'une variable du type caractère (char) doit être mise entre quottes (' '),
  Exemples : 'a', 'h', 'E'.
- Les variables du type numérique (int, long, float, double) peuvent être positives ou négatives.
- Pour déclarer des variables positives, il suffit d'ajouter « unsigned » au type concerné.
  Exemple :

```
unsigned int k ;        /* k est un entier positif */
```

II. LES OPERATEURS

II.1. Opérateurs arithmétiques

+	Addition	5 + 8
-	Soustraction	5 - 8
*	Multiplication	5 * 8
/	Division	5 / 8
%	Reste de la division euclidienne	5 % 8

Remarques :

- Le résultat de la division de deux entiers est un entier. Si le résultat n'est pas entier la partie décimale est ignorée.
- L'opérateur modulo (%) ne peut pas être utilisé avec des nombres décimaux (float, double, …).

II.2. Opérateurs relationnels

= =	Egal
!=	Différent
>	Strictement supérieur
<	Strictement inférieur
>=	Supérieur ou égal
>=	Inférieur ou égal

II.3. Opérateurs logiques

!	Non
&&	Et
| |	Ou

II.4. Priorité des opérateurs

Entre les opérateurs que nous venons de présenter, nous pouvons distinguer les classes de priorités suivantes :

Priorité 1 (la plus forte):	**( )**
Priorité 2:	**!, ++, --, -**
Priorité 3:	***, /, %**
Priorité 4:	**+, -**
Priorité 5:	**<, <=, >, >=**
Priorité 6:	**==, !=**

Priorité 7: **&&**

Priorité 8: ||

Priorité 9 (la plus faible): **=, +=, -=, *=, /=, %=**

- Dans chaque classe de priorité, les opérateurs ont la même priorité. Si nous avons une suite d'opérateurs binaires de la même classe, l'évaluation se fait en passant *de la gauche vers la droite* dans l'expression.
- Pour les opérateurs unaires (**!, ++, --**) et pour les opérateurs d'affectation (**=, += , -=, *=, /=, %=**), l'évaluation se fait *de droite à gauche* dans l'expression.

III. LES CONSTANTES

Une constante est une donnée dont la valeur reste fixe tout au long de l'exécution d'un programme.

Syntaxe de déclaration :

- Soit hors de la fonction principale main() :

```
#define <nom-constante> <valeur>
```

- Soit dans le corps de la fonction principale main() :

```
const <type> <nom-constante> = <valeur> ;
```

Exemple :

Les deux déclarations suivantes sont similaires :

```
#define pi 3.14
const pi=3.14 ;
```

Remarques :

- Une constante ne peut jamais figurer à gauche d'une affectation.
- Voici quelques constantes caractères :

'\n'	Interligne
'\t'	Tabulation horizontale
'\v'	Tabulation verticale
'\f'	Saut de page
'\\'	Back slash
'\''	Cote
'\"'	Guillemets

IV. L'INSTRUCTION D'AFFECTATION

IV.1. L'instruction d'affectation ' = '

L'affectation est une opération qui permet d'attribuer une valeur ou le résultat d'une expression à une variable déjà déclarée.

Syntaxe d'affectation:

```
<variable> = <expression> ;
```

Une expression peut prendre deux formes différentes :

<expression> : <opérande><opérateur><opérande>

<opérateur><opérande>

Exemple :

```
i = 5 ;
c= 'a' ;
j = 10 ;
k = 10+5 ;
k = k + 1 ;
```

On peut initialiser les variables dès leurs déclarations.

Exemple :

```
int k=0 ;
char c='A' ;
```

IV.2. Les opérateurs d'incrémentation et de décrémentation

x++ ;	x=x+1 ;
x--;	x=x-1 ;

IV.3. Simplification des opérateurs arithmétiques

x += y ;	x = x + y ;
x -= y ;	x = x - y ;
x *= y ;	x = x * y ;
x /= y ;	x = x / y ;
x %= y ;	x = x % y ;

IV.4. Conversion de types

Il est possible de convertir explicitement une valeur en un type quelconque en forçant la transformation (casting (conversion de type forcée)).

Syntaxe : (<Type>) <Expression>

Exemple : on se propose de diviser deux variables du type entier. Pour avoir plus de précision, nous voulons avoir un résultat de type rationnel. Pour ce faire, nous convertissons l'une des deux

opérandes en float. Automatiquement, le langage C convertira l'autre opérande en float et effectuera une division rationnelle:

```
char A = 3;
int B = 4;
float C;
C = (float)A/B;
```

La valeur de *A* est explicitement convertie en **float**. La valeur de *B* est automatiquement convertie en **float**. Le résultat de la division (type rationnel, valeur 0.75) est affecté à *C*.

V. LES FONCTIONS STANDARD D'ENTREES/SORTIES

V.1. Bibliothèque adoptée

A chaque utilisation d'une fonction d'entrée ou de sortie (lecture, écriture), il faut inclure le fichier « stdio.h » à l'entête du programme. [3]

Ceci se traduit par l'instruction suivante : #include<stdio.h>

« stdio.h » est le nom d'un fichier de définitions, des fonctions et des constantes utilisées dans un programme pour des entrées ou des sorties standards.

std	→	standard
i	→	input
o	→	output
h	→	header (entête)

V.2. Fonction getchar()

Elle permet de renvoyer le code d'un caractère saisi à partir de l'organe standard d'entrée à savoir le clavier.

Exemple :

```
char c ;
c=getchar( ) ;
```

V.3. Fonction putchar()

Elle permet d'afficher le caractère mis en paramètre sur l'organe standard de sortie à savoir l'écran.

Exemple :

```
char c ;
c='a' ;
putchar(c) ;
```

V.4. Fonction scanf()

Elle permet d'affecter la valeur saisie à partir du clavier à la variable mise en paramètre selon le type concerné.

Syntaxe :

```
scanf(<format>, <liste des adresses de variables>) ;
```

<format> : chaîne de caractères délimitée par " " définissant les spécifications de conversion de types à appliquer au niveau de la lecture.

Une spécification de conversion commence par le caractère % suivi d'un type de conversion comme suit :

%d ou %i	entier relatif (int)
%u	entier naturel (unsigned)
%ld	long
%c	char
%f	float
%lf	double

Exemple :

```
char c,y ;
float x ;
int i ;
scanf(" %c ", & c) ;     /* saisir un caractère et le stocker dans
                            la variable c */
scanf(" %f %c %d ",&x, &y, &i) ;
```

V.5. Fonction printf

Elle permet d'afficher sur l'écran les variables mises en paramètre selon le type concerné.

Syntaxe :

```
printf(<format>, <liste d'arguments>) ;
```

<format> : chaîne de caractères délimitée par " " définissant les spécifications de conversion de types à appliquer au niveau de l'affichage.

Exemple :

```
int i ;
char c ;
i=5 ;
c='a' ;
printf(" %c %d " , c , i ) ;   /* permet d'afficher l'entier 5 et
                                  le caractère a sur l'écran */
```

- La fonction printf() peut ne pas contenir des arguments.

  Exemple :

```
printf(" bonjour ") ;
```

- Lors de l'affichage, on peut donner un format spécifique aux résultats.

  Exemple :

```
int d=203 ;
float f=23.54 ;
printf("%5d", d ) ;      /* Ceci va réserver 5 cellules en cadrant
                          l'entier 203 à droite */
printf(" %-5 d ", d ) ;  /* Cadrage à gauche */
printf(" %2.1 f ", f) ;  /* L'affichage se fait avec un seul
                          chiffre après la virgule */
```

- La fin de ligne ainsi que la tabulation peuvent s'énoncer comme suit :

\n	fin de ligne
\t	tabulation

VI. EXERCICE RESOLU

Enoncé :

Ecrire un programme C permettant de saisir 2 entiers a et b et d'afficher la somme, la soustraction, le produit, la moyenne, le quotient et le reste de la division euclidienne de ces deux entiers.

Solution :

```
#include<stdio.h>
main()
{
 int a, b;
 scanf("%d%d",&a,&b) ;
 printf("%d + %d = %d\n", a, b, a+b) ;
 printf("%d - %d = %d\n", a, b, a-b) ;
 printf("%d * %d = %d\n", a, b, a*b) ;
 printf("%d % %d = %d\n", a, b, a%b) ;
 printf("%d / %d = %d\n", a, b, a/b) ;
}
```

PROGRAMMATION I

CHAPITRE : 3

LES STRUCTURES DE CONTROLE CONDITIONNELLES

Objectifs spécifiques

-Manipuler les structures alternatives qui permettent d'évaluer une condition,

-Choisir la structure adéquate en fonction du traitement à effectuer,

-Ecrire des programmes faisant appel aux différentes structures conditionnelles.

Plan du chapitre

Volume horaire

2 séances de Cours Intégré

I. DEFINITION

L'instruction conditionnelle permet d'évaluer une condition et d'exécuter en conséquence un bloc d'instructions.

II. L'INSTRUCTION CONDITIONNELLE « IF »

II.1 L'instruction conditionnelle « if » simple

Syntaxe :

```
if (<condition>)
{
 <bloc d'instructions >
}
```

<condition> : expression arithmétique ou logique.

- La condition est évaluée. Le résultat est une valeur numérique,
- Si le résultat est égal à 1, la condition est interprétée à vrai et par conséquent le bloc d'instructions associé est exécuté,
- Si le résultat est égal à 0, la condition est interprétée à faux et par conséquent on ne tient pas compte du bloc d'instructions.

Exemple : Ecrire un programme qui en testant sur la moyenne permet de vérifier si l'étudiant a réussi son examen.

```
#include<stdio.h>
main()
{
 float moyenne ;
 scanf ("%f ", &moyenne) ;
 if(moyenne >= 10)
 {
       printf(" l'etudiant a réussi son examen") ;
 }
}
```

II.2. L'instruction conditionnelle « if … else»

Syntaxe :

```
if (<condition>)
{
 <bloc d'instructions 1>
}
else
{
 <bloc d'instructions 2>
}
```

<condition> : expression arithmétique ou logique.

- La condition est évaluée. Le résultat est une valeur numérique

- Si le résultat est égal à 1, la condition est interprétée à vrai et par conséquent le bloc d'instructions 1 est exécuté.
- Si le résultat est égal à 0, la condition est interprétée à faux et par conséquent c'est le bloc d'instructions 2 qui est exécuté.

Exemple : Ecrire un programme en C qui permet de lire à partir du clavier deux entiers et d'afficher l'entier le plus petit.

```
#include<stdio.h>  /* bibliothèque standard d'entrée/sortie */
main()             /* fonction principale */
{

 int i, j;         /* déclaration de deux entiers */

 printf(" Saisissez deux entiers : ") ;
 scanf(" %d %d",&i, &j); /* Saisie de 2 entiers */

 if(i < j
{
       /* Affichage de l'entier1 s'il est le plus petit */
       printf(" %d est plus petit que %d ", i, j);
 }
 else
 {
       /* Affichage de l'entier2 s'il est le plus petit */
       printf(" %d est plus petit que %d ", j, i);
 }
}
```

NB :

- La présence de « else » dans une instruction conditionnelle est facultative.

  Exemple :

```
if ( condition )
{
 <bloc d'instructions>
}
```

- Si un bloc d'instructions associé à « if » ou à « else » se réduit à une seule instruction, les accolades peuvent être évitées.

  Exemple :

```
If ( condition )
 <instruction> ;
else
 <instruction> ;
```

- Le bloc d'instructions suivant :

```
If ( a != 0 )
 ;
else
 b=0 ;
```

  est équivalent à :

```
if ( a == 0)
 b=0 ;
```

- La suite d'instructions suivante :

```
if (A>B)
 MAX=A;
else
 MAX=B;
```

peut être remplacée par :

```
MAX = (A > B) ? A : B;
```

Dans le cas général:

```
<expr1> ? <expr2> : <expr3> ;
```

C'est-à-dire : Si <expr1> fournit la valeur zéro, alors la valeur de <expr3> est fournie comme résultat. Sinon c'est la valeur de <expr2> qui est fournie comme résultat.

II.3. Instructions « if » imbriquées

Il est à noter que les instructions « if » peuvent être imbriquées comme le montre l'exemple suivant :

```
if ( a >= 0)
{
 if ( a <= 9)
 {
       printf ( " Chiffre ") ;
 }
 else
 {
       printf ( " >> 9 ") ;
 }
}
else
{
 printf ( " << 0 ") ;
}
```

III. L'INSTRUCTION DE SÉLECTION MULTIPLE « SWITCH »

L'instruction « switch » permet de faire plusieurs tests de valeurs sur le contenu d'une même variable.

Syntaxe :

```
switch ( <variable> )
{
 case Cte1 :
       <bloc d'instructions 1>
       break;
 case Cte2 :
       <bloc d'instructions 2>
       break;
 .
 .
 case CteN:
       <bloc d'instructions N>
  [ default :
```

```
            <bloc d'instructions> ]
    }
```

- <variable>: de type char, int, long
- Cte1, Cte2, …, CteiN : constantes de type compatible avec la variable évaluée.
- La variable est évaluée. Si elle est égale à la Ctei alors c'est le bloc d'instructions associé qui est exécuté. Si elle est différente de toutes les constantes, c'est le bloc d'instructions associé à « default » qui est exécuté.
- « default » ainsi que le bloc d'instructions associé est facultatif.
- L'instruction « break » permet en général de quitter un bloc d'instructions et par conséquent de quitter le bloc « switch ».

IV. EXERCICE RESOLU

Enoncé :

Ecrire un programme C qui permet de tester si un caractère saisi à partir du clavier est une voyelle.

Solution 1: Utilisation de if

```
#include<stdio.h>
main()
{
 char c;
 printf("Saisissez un caractère : ");
 scanf("%c",&c);
 if(c== 'a' || c== 'e' || c== 'i' || c== 'o' || c== 'u' || c== 'y'
 || c== 'A' || c== 'E' || c== 'I' || c== 'O' || c== 'Y')
       printf("%c est une voyelle", c);
 else
       printf("%c n'est pas une voyelle", c);
 }
}
```

Solution 2: Utilisation de switch

```
#include<stdio.h>
main()
{
 char c;
 printf("Saisissez un caractère : ");
 scanf("%c",&c);
 switch(c)
 {
       case 'a' :
       case 'e' :
       case 'i' :
       case 'o' :
       case 'u' :
       case 'y' :
       case 'A' :
       case 'E' :
       case 'I' :
       case 'O' :
       case 'U' :
       case 'Y' :  printf("%c est une voyelle", c);
```

```
            default :   printf("%c n'est pas une voyelle", c);
        }
    }
```

PROGRAMMATION I

CHAPITRE : 4

LES STRUCTURES DE CONTROLE ITERATIVES

Objectifs spécifiques

-Manipuler les structures répétitives,

-Choisir la structure adéquate en fonction du traitement à effectuer,

-Ecrire des programmes faisant appel aux différentes structures répétitives.

Plan du chapitre

Volume horaire

2 séances de Cours Intégré

I. DEFINITION

Les boucles sont des structures qui permettent de répéter ou refaire plusieurs fois la même série d'instructions pour des valeurs différentes jusqu'à ce qu'une condition ne soit plus réalisée.
On appelle parfois ces structures *instructions répétitives* ou *itératives*.
Une itération doit se terminer. Il ne faut jamais écrire des boucles infinies. Pour ce faire, il faut une condition d'arrêt.

II. L'INSTRUCTION ITERATIVE « FOR »

Syntaxe :

```
for ( <initialisation> ; <test_de_boucle> ; <incrémentation>)
{
 <bloc d'instructions>
}
```

- <initialisation> : séparées par une virgule, ces instruction définissent les initialisations de la boucle. Elles sont exécutées une seule fois avant la première itération.
- <test_de_boucle> : expression qui doit être égale à « faux » pour stopper la boucle. Elle est évaluée avant chaque itération.
- <incrémentation> : séparées par virgule, elles définissent la séquence d'instructions à réaliser entre deux itérations (avant d'effectuer l'évaluation de <test_de_boucle>).
- Tant que le <test_de_boucle> est évalué à vrai, le bloc d'instructions est exécuté.

Exemple : Ecrire un programme en C qui permet d'afficher les entiers naturels de 1 à 10.

```
#include<stdio.h>
main( )
{
 int I ;
 for( I=1 ; I <= 10 ; I++)
 {
       printf (" %d ", I) ;
 }
}
```

NB : Lorsque le bloc d'instructions est limité à une seule instruction, les accolades associées à l'instruction « for » peuvent être évitées.

III. L'INSTRUCTION ITÉRATIVE « WHILE »

Syntaxe :

```
while ( <condition> )
{
<bloc d'instructions>
}
```

- <condition> : expression arithmétique ou logique à évaluer.
- Tant que la condition <condition> est évaluée à vrai, le bloc d'instructions est exécuté.

Exemple : Même exemple traité précédemment en faisant maintenant recours à l'instruction « while ».

```
#include<stdio.h>
main()
{
 int I=1;
 while ( I <=10)
 {
       printf("%d",I);
       I++;
 }
}
```

IV. L'INSTRUCTION ITÉRATIVE « DO.. WHILE »

Syntaxe :

```
do
{
 <bloc d'instructions>
}
while (<condition>);
```

- La condition est évaluée à la fin de chaque itération.
- Le bloc d'instructions est exécuté au moins une fois.
- Tant que la condition est évaluée à vrai, le bloc d'instructions est exécuté.

Exemple : Reprendre l'exemple précédent en utilisant cette fois l'instruction « do .. while »

```
#include<stdio.h>
main( )
{
 int I=1;
 do
 {
       printf("%d",I);
       I++;
 }
 while ( I <=10);
}
```

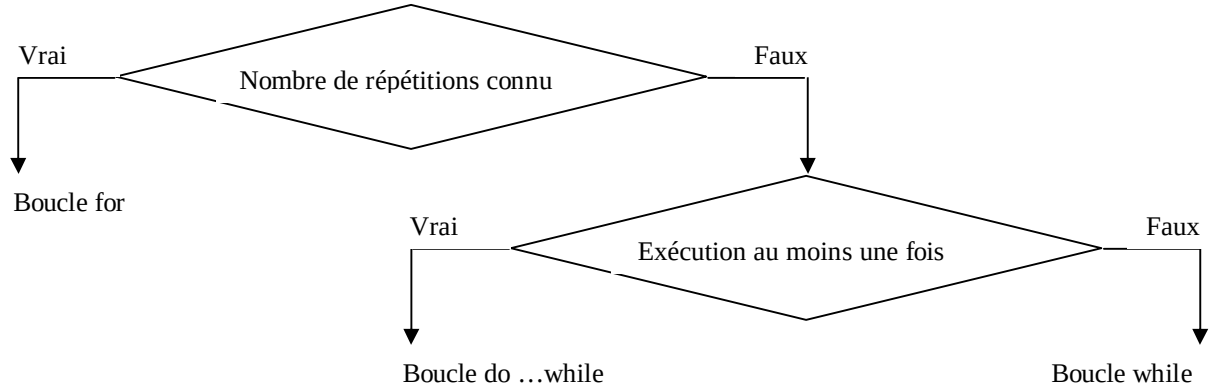
Remarque : [4]

Il arrive fréquemment qu'en évaluant un test à l'intérieur d'une boucle, on aimerait arrêter brutalement la boucle ou alors « sauter » certains cas non significatifs. C'est le rôle de l'instruction « break » :

break : permet de sortir immédiatement d'une boucle for, while ou do while. Il est à noter que si la boucle se trouve elle-même dans une autre boucle, seule la boucle où l'instruction break se trouvait est stoppée.

V. CHOIX DE LA STRUCTURE REPETITIVE ADEQUATE

Le choix de la structure de contrôle itérative adéquate dépend du problème à résoudre. En effet, si le nombre de répétitions est connu, il est conseillé d'utiliser la boucle FOR. Par contre, si le nombre d'itérations varie de 1 à n, il est recommandé d'utiliser la boucle DO..WHILE. Enfin si le nombre d'itérations varie de 0 à n, il est possible d'utiliser la boucle WHILE. La figure suivante résume tous ces cas.



Choix de structure itérative

VI. EXERCICE RESOLU

Enoncé : Calculez la somme des N premiers termes de la série harmonique :

$$1 + 1/2 + 1/3 + ... + 1/N$$

Solution :

```
#include<stdio.h>
main( )
{
 int i, N;
 float S = 0;                  // Initialisation de la série

 scanf(" %d ", &N);            // Saisie de N

 for( i=1 ; i <= N ; i++)
 {
      S = S +(float)1/i;       // Conversion de type
 }
 printf (" %f ", S) ;
}
```

PROGRAMMATION I

CHAPITRE : 5

LES FONCTIONS

Objectifs spécifiques

-Décomposer un problème en plusieurs modules,

-Ecrire des programmes modulaires,

-Passer des paramètres à des fonctions.

Plan du chapitre

Volume horaire

2 séances de Cours Intégré

I. DEFINITION

Une fonction est un ensemble d'instructions structurées dans un bloc accessible via un nom propre appelé le nom de la fonction.

Syntaxe :

```
<type de la valeur de retour>] <identificateur> ([<liste des
 paramètres>])
[<déclaration des paramètres>]
{
 <bloc d'instructions>
}
```

La valeur de retour d'une fonction est transmise à la fonction appelante via l'instruction : **return**

Syntaxe :

```
return(<expression>)
```

Cette instruction permet :

- l'évaluation de l'<expression>,
- la conversion automatique du résultat de l'expression au type de la fonction,
- le renvoi du résultat,
- la terminaison de la fonction.

Exemple : La fonction MAX est du type int et elle a besoin de deux paramètres du type int. Le résultat de la fonction MAX peut être intégré dans d'autres expressions.

```
int MAX(int N1, int N2)
{
 if (N1>N2)
       return N1;
 else
       return N2;
}
```

NB :

- La spécification du type de la valeur de retour est facultative. Si on ne précise pas le type de valeur de retour, le type int est pris par défaut.
- Il n'est pas nécessaire de spécifier le type de valeur de retour pour une fonction qui ne renvoie pas un résultat. On peut à la limite lui associer le type **void** comme type du résultat.
- Une fonction peut ne pas contenir de paramètres. Dans ce cas, on peut déclarer la liste des paramètres comme **void** ou simplement comme ().
- En principe, l'ordre des définitions dans le texte du programme ne joue pas de rôle, mais chaque fonction doit être déclarée ou définie avant d'être appelée.

- Une fonction ne peut pas fournir comme résultat des tableaux, des chaînes de caractères ou des fonctions.

II. DECLARATION DES FONCTIONS

En C, il faut déclarer chaque fonction avant de pouvoir l'utiliser. La déclaration informe le compilateur du type des paramètres et du résultat de la fonction. A l'aide de ces données, le compilateur peut contrôler si le nombre et le type des paramètres d'une fonction sont corrects. Si dans le texte du programme la fonction est définie avant son premier appel, elle n'a pas besoin d'être déclarée. [5]

La déclaration d'une fonction se fait par un *prototype* de la fonction qui indique uniquement le type des données transmises et reçues par la fonction.

Syntaxe :

```
<TypeRésultat> <NomFonction> (<TypePar1>, <TypePar2>, ...);
```

Exemple : Reprenons l'exemple de la fonction MAX, sa déclaration s'écrit :

```
int MAX(int, int) ;
```

Nous pouvons distinguer les différents types de déclaration des fonctions comme suit :

- Déclaration **locale** : Une fonction peut être déclarée localement *dans la fonction qui l'appelle* (avant la déclaration des variables). Elle est alors disponible à cette fonction.
- Déclaration **globale** : Une fonction peut être déclarée globalement *au début du programme* (derrière les instructions #include). Elle est alors disponible à toutes les fonctions du programme.
- Déclaration **implicite** par la définition : La fonction est automatiquement disponible à toutes les fonctions qui suivent sa définition.

Remarques :

- La fonction principale ' main ' n'a pas besoin d'être déclarée.
- Les variables déclarées dans un bloc d'instructions sont uniquement visibles à l'intérieur de ce bloc. On dit que ce sont des variables locales à ce bloc. Ainsi, aucune autre fonction n'a accès à ces variables.

III. APPEL DES FONCTIONS

Pour appeler une fonction à l'intérieur d'une autre fonction, on fait référence à son identificateur (nom) suivi éventuellement de la liste des paramètres.

Au moment de l'exécution, un appel à une fonction se traduit par le branchement au bloc associé à la fonction, l'exécution des instructions du bloc et retour à l'instruction qui suit l'appel.

Lors du passage des paramètres à une fonction, on distingue deux modes à savoir :

- Passage des paramètres par valeurs,
- Passage des paramètres par adresses.

Passage par valeurs :

Le mode de passage des paramètres par valeurs ne permet pas de modifier le contenu des variables utilisées comme paramètres réels d'une fonction appelante.

Les paramètres d'une fonction sont à considérer comme des ***variables locales*** qui sont initialisées automatiquement par les valeurs indiquées lors d'un appel.

A l'intérieur de la fonction, nous pouvons donc changer les valeurs des paramètres sans influencer les valeurs originales dans les fonctions appelantes.

Remarque : Le mode de passage par adresses sera traité au deuxième niveau lors du module Programmation II.

IV. EXERCICE RESOLU

Enoncé : Ecrire une fonction qui permet de convertir un caractère minuscule (s'il l'est) en majuscule. Utiliser cette fonction pour convertir un texte en majuscule.

Solution :

```
#include<stdio.h>

int est_miniscule(char c)
{
 return(c >= 'a' && c<= 'z') ;
}
char min_maj(char c)
{
 return('A' – 'a' + c);
}

main( )
{
 char c ;

 while((c=getchar()) !='#')
 {
       if (est_miniscule(c))
             c=min_maj(c);
       putchar(c) ;
 }
}
```

PROGRAMMATION I

CHAPITRE : 6

LES TABLEAUX

Objectifs spécifiques

-Ecrire des programmes utilisant la structure tableau,

-Manipuler les éléments d'un tableau,

-Effectuer des traitements spécifiques sur les tableaux,

-Maîtriser le mécanisme de tri.

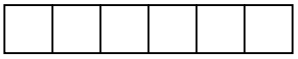
Plan du chapitre

Volume horaire

2 séances de Cours Intégré

I. DEFINITION

Un tableau est un ensemble de cellules mémoires de même type consécutives et accessibles à travers un identificateur (le nom du tableau) et un indice (le numéro de la cellule).[6]

Exemple : 

Le tableau T disposant de 6 éléments. Mathématiquement parlant, T est un vecteur de dimension 6. En C, le nom d'un tableau est le représentant de ***l'adresse du premier élément*** du tableau. Les adresses des autres composantes sont calculées (automatiquement) relativement à cette adresse.

II. TABLEAUX UNIDIMENSIONNELS

II.1. Déclaration d'un tableau unidimensionnel

Syntaxe :

```
<type> <identificateur>[<dimension>];
```

- <type> : type du tableau. C'est le type des différents éléments du tableau.
- <identificateur> : nom accordé au tableau
- <dimension> : le nombre de cases réservées au tableau

Exemple :

```
char ch[10] ;      /* un tableau de caractères de dimension 10 */
int tab[5] ;       /* un tableau d'entiers de dimension 5 */
```

II.2. Accès à une case d'un tableau unidimensionnel

- Il est à noter que la première cellule du tableau est d'indice 0
- Pour accéder à la $i^{ème}$ case d'un tableau, il suffira d'écrire : <nom_du_tableau>[i]

  Exemple :

```
int Tab[3] ;
Tab[0]=2 ;
Tab[1]=3 ;
Tab[2]=-9 ;
```

- On peut initialiser un tableau unidimensionnel de la manière suivante :

```
Tab = {2, 3, 9} ;
```

III. TABLEAUX MULTIDIMENSIONNELS

III.1. Déclaration d'un tableau multidimensionnel

Syntaxe :

```
<type><identificateur>[<nombre_tableaux>]<taille_lignes>
[<taille_co  lonnes>] ;
```

Exemple :

```
char matrice[4][5] ;     /* matrice est un tableau de 4 lignes et 5
                            colonnes. */
```

Ceci réserve une zone mémoire de 4 * 5 = 20 caractères.

```
int tableau [3] [4] [5] ;/* il s'agit de 3 tableaux de tableaux de
                            4 * 5 entiers.
```

Remarque :

Un tableau multidimensionnel peut être déclaré comme un tableau unidimensionnel comme le montre l'exemple suivant :

```
int tab [2] [3] ;
int tab[6] ; /* les 3 premiers éléments correspondent à la première
                ligne. Les autres correspondent à la deuxième ligne */
```

III.2. Accès à une case d'un tableau multidimensionnel

Pour accéder à un élément se trouvant à la ligne d'indice i et à la colonne d'indice j de la matrice, il suffira d'écrire : matrice [i] [j]

Exemple :

```
int tab [2][2] ;
tab[0][0]=12 ;
tab[1][0]=3 ;
tab[0][1]=2 ;
tab[1][1]=8 ;
tab[0][2]=5 ;
tab[1][2]=6 ;
```

Remarque :

On peut initialiser un tableau de la manière suivante :

```
int tab[2][3]={1,2,3,4,5,6} ;
```

ou encore

```
int tab[2][3]={{1,2,3},{4,5,6}} ;
```

IV. TABLEAUX ET FONCTIONS

- Une fonction disposant d'un tableau comme paramètre se déclare comme suit :

```
<type_retour><nom_fonction>(<type_tableau> <nom_tableau> [ ])
```

ou bien

```
<type_retour><nom_fonction>(<type_tableau> *<nom_tableau>)
```

- L'appel de la fonction s'énonce comme suit :

```
    <nom_fonction>(<nom_tableau>) ;
```

V. OPERATIONS DU TRI

V.1. Définition

Une opération de tri consiste à mettre en ordre une suite de valeurs stockées dans un tableau par rapport à un critère déterminé (croissant, décroissant).

Les opérations de tri sont très utilisées en informatique pour traiter l'information.

Il existe de nombreux algorithmes de tri avec des performances différentes. Parmi ces algorithmes, nous détaillons :

V.2. Tri par sélection

C'est l'un des algorithmes de tri le plus simple.

On recherche le plus petit élément du tableau. On le place dans sa position finale, c'est-à-dire en première position.

On recherche le second plus petit élément. On le place également dans sa position finale, c'est à dire en deuxième position. On réitère le même processus jusqu'avant dernier élément du tableau. Le dernier sera dans sa position finale.

Exemple :

Etant donné le tableau T comportant les éléments suivants :

		20	90	10	150	-2	30

Etape 1 :

Chercher l'indice du plus petit élément et permuter l'élément correspondant avec l'élément d'indice 0.

Index cherché	4 =>	**-2**	90	10	150	**20**	30

Etape 2 :

Index cherché	2 =>	-2	**10**	**90**	150	20	30

Etape 3 :

Index cherché	4 =>	-2	10	**20**	150	**90**	30

Etape 4 :

Index cherché	5 =>	-2	10	20	**30**	90	**150**

Etape 5 :

Index cherché	4 =>	-2	10	20	30	90	150
D'où le tableau T trié s'écrit :		-2	10	20	30	90	150

Le programme s'écrit comme suit :

```
#include<stdio.h>
#define N 6
main()
{
 int tab[N],i,posMin,j,temp;
 printf("Vous faites insérer les éléments du tableau : ");
 for(i=0;i<N;i++)
 {
       scanf("%d",&tab[i]);
 }

 for(i=0;i<(N-1);i++)
 {
       posMin =i;
       for(j=i+1;j<N;j++)
       {
             if(tab[j]<tab[posMin])
                         posMin=j;
       }
       temp=tab[i];
       tab[i]=tab[posMin];
       tab[posMin]=temp;
 }
 printf("\nLes éléments du tableau trié: ");
 for(i=0;i<N;i++)
 {
       printf("%d ",tab[i]);
 }
}
```

V.3. Tri à bulles

On fait remonter le plus grand élément du tableau en comparant les éléments successifs.

On compare le 1er et le 2ème élément, et on les échange si le 2ème est inférieur au premier. On recommence jusqu'à la fin du tableau. Le maximum du tableau a été « remonté » en dernière position. On recommence en s'arrêtant à l'avant dernière place du tableau et ainsi de suite. On arrête lorsqu'il ne reste plus qu'un seul élément à trier.

Exemple :

Etant donné le tableau T comportant les éléments suivants :

	20	90	10	150	-2	30
Etape 1 :						
20 < 90 alors :	**20**	**90**	10	150	-2	30
90 > 10 alors :	20	**10**	**90**	150	-2	30
90 < 150 :	20	10	**90**	**150**	-2	30
150 > -2 :	20	10	90	**-2**	**150**	30
150 > 30 :	20	10	90	-2	**30**	**150**

Etape 2 :

20 > 10 :	**10**	**20**	90	-2	30	150
20 < 90 :	10	**20**	**90**	-2	30	150
90 > -2 :	10	20	**-2**	**90**	30	150
90 > 30 :	10	20	-2	**30**	**90**	150
Etape 3 :						
10 < 20 :	**10**	**20**	-2	30	90	150
20 > -2 :	10	**-2**	**20**	30	90	150
20 < 30 :	10	-2	**20**	**30**	90	150
Etape 4 :						
10 > -2 :	**-2**	**10**	20	30	90	150
10 < 20 :	-2	**10**	**20**	30	90	150
Etape 5 :						
-2 < 10 :	**-2**	**10**	20	30	90	150
D'où le tableau T trié :	-2	10	20	30	90	150

Le programme associé à cette opération du tri s'écrit alors :

```
#include<stdio.h>
#define N 6
main()
{
 int tab[N],i,j,temp;

 printf("Vous faites insérer les éléments du tableau: ");
 for(i=0;i<N;i++)
 {
       scanf("%d",&tab[i]);
 }
 j=N-1;
 do
 {
       for(i=0;i<j;i++)
       {
             if(tab[i]>tab[i+1])
             {
                   temp=tab[i+1];
                   tab[i+1]=tab[i];
                   tab[i]=temp;
             }
       }
       j--;
 }
 while(j>=0);

 printf("\nLes éléments du tableau trié: ");
 for(i=0;i<N;i++)
 {
       printf("%d ",tab[i]);
 }
}
```

V.4. Tri par insertion

Parmi les 2 premiers éléments du tableau, placer le plus petit dans la case d'indice 0.

On compare le 3ème élément avec ces précédents dans l'ordre pour avoir les 3 éléments triés.

On réitère alors le même processus jusqu'à arriver au dernier élément du tableau.

Exemple :

Etant donné le tableau T comportant les éléments suivants :

	20	90	10	150	-2	30
Etape 1 :						
20 < 90 :	**20**	**90**	10	150	-2	30
Etape 2 :						
Insérer 10 dans {20, 90} :	**10**	**20**	**90**	150	-2	30
Etape 3 :						
Insérer 150 dans {10, 20, 90} :	**10**	**20**	**90**	**150**	-2	30
Etape 4 :						
Insérer -2 dans {10, 20, 90, 150} :	**-2**	**10**	**20**	**90**	**150**	30
Etape 5 :						
Insérer 30 dans {-2, 10, 20, 90, 150} :	**-2**	**10**	**20**	**30**	**90**	**150**

Le programme correspondant à cette méthode s'écrit :

```
#include<stdio.h>
#define N 6
main()
{
 int tab[N],i,pos,b,j,temp;

 printf("Vous faites insérer les éléments du tableau: ");
 for(i=0;i<N;i++)
 {
       scanf("%d",&tab[i]);
 }

 for(i=1;i<N;i++)
 {
       pos=i-1;
       b=1;
       j=i;
       while(pos>=0 && b)
       {
             if(tab[pos]>tab[j])
             {
                   temp=tab[j];
                   tab[j]=tab[pos];
                   tab[pos]=temp;
                   j=pos;
                   pos--;
             }
             else
                   b=0;
       }
 }
 printf("\nLes éléments du tableau trié: ");
 for(i=0;i<N;i++)
 {
```

```
            printf("%d ",tab[i]);
     }
    }
```

VI. EXERCICE RESOLU

Enoncé : Ecrire un programme qui permet de remplir un tableau de 10 entiers et de remplacer toute occurrence d'un entier saisi x par 0.

Solution :

```
    #include<stdio.h>
    main()
    {
     int tab[10], i, x;

    // Remplissage du tableau
     for(i = 0; i < 10 ; i ++)
     {
           printf(" Saisir l'élément N°: %d", i);
           scanf("%d",&tab[i]);
     }

     printf(" Saisir un entier : ");
     scanf("%d",&tab[i]);

    // Modification du tableau
     for(i = 0; i < 10 ; i ++)
     {
           if(tab[i] == x)
           tab[i] = 0;
     }

    // Affichage du tableau
     for(i = 0; i < 10 ; i ++)
     {
     printf(" %d ", tab[i]);
     }

    }
```

PROGRAMMATION I

CHAPITRE : 7

LES CHAINES DE CARACTERES

Objectifs spécifiques

-Distinguer entre le type chaîne de caractères et le type tableau unidimensionnel,

-Faire le tour d'horizon sur les principales fonctions de manipulation de chaînes de caractères,

-Ecrire des programmes faisant appel à ce type de données.

Plan du chapitre

Volume horaire

2 séances de Cours Intégré

I. DEFINITION

Une chaîne de caractères est un tableau de caractères qui se termine par le caractère '\0'.

Syntaxe de déclaration :

```
char <nom-chaine>[<taille>+1] ;
```

Remarque : Le 1 ajouté à <taille> symbolise le caractère de fin de chaîne '\0'.

Exemple :

```
char chaine[5] ;
```

Syntaxe d'initialisation :

```
<nom-chaine> = "contenu de la chaîne";
```

ou bien

```
<nom-chaine> = {<car1>,{<car2>,…,{<carN>,'\0'};
```

Exemple :

```
char chaine[5] ;
chaine = "Voici une chaîne de caractères";
```

II. FONCTIONS STANDARDS DE MANIPULATION DES CHAINES

II.1. Fonction scanf()

Elle permet de lire une chaîne de caractères à partir du clavier. La spécification de conversion en chaîne de caractères est définie par %s.

Exemple :

```
char chaine[30] ;
scanf (" %s ", chaine) ;
/* Attention, y a pas de référence d'adresse & */
```

II.2. Fonction printf()

Elle permet d'afficher une chaîne de caractères sur l'écran. La spécification de conversion en chaîne de caractères est définie par %s.

Exemple :

```
char chaine[30] ;
scanf(" %s ",chaine) ;
printf("La chaîne de caractères saisie est: %s ",chaine) ;
```

II.3. Fonction sscanf()

Elle permet d'effectuer une lecture formatée. Elle a le même rôle que la fonction « scanf() » sauf que la lecture se fait à travers une chaîne de caractères.

Exemple :

```
char c1, c2, chaine[30] ;
int e ;
sscanf(chaine, " %c %c %d " , &c1, &c2, &e);
```

II.4. Fonction sprintf()

Elle permet d'effectuer une écriture formatée de variables dans une chaîne de caractères. Elle a le même rôle que la fonction « printf() » sauf que l'écriture est dirigée vers une chaîne.

Exemple :

```
int i=3 ;
char s[30] ;
sprintf( s, "entier : %d", i);
```

II.5. Fonction gets()

Elle permet de lire une chaîne de caractères à partir du clavier jusqu'à l'identification du caractère Entrée.

La valeur de retour est la chaîne lue. Si rien, la valeur NULL est retournée.

Exemple :

```
gets(s) ;/* saisir une chaîne de caractères et la stocker dans s */
```

II.6. Fonction puts()

Elle permet d'écrire une chaîne de caractères sur l'écran.

Exemple :

```
char s[20] ;
gets(s) ;    /* saisir une chaîne à partir du clavier */
puts(s) ;    /* l'afficher sur l'écran */
```

III. AUTRES FONCTIONS DE MANIPULATION DES CHAINES

De nombreuses fonctions de manipulation de chaînes, se trouvant dans la bibliothèque <string.h>, sont directement fournies.

Parmi ces fonctions, nous pouvons citer :

III.1. Fonction strlen()

Elle permet de récupérer la longueur d'une chaîne de caractères. Elle renvoie le nombre de caractères stockés dans la chaîne.

Exemple :

```
char s[20] ;
gets(s) ;
printf("Longueur de la chaîne : %d", strlen(s));
```

Si jamais la chaîne de caractères saisie est « bonjour », cette portion de code va nous afficher sur l'écran 7.

III.2. Fonction strcpy()

Elle permet de copier une chaîne de caractères dans une autre chaîne.

Exemple :

```
char s1[20],s2[20] ;
gets(s1) ;              /* saisie d'une chaîne de caractères s1 */
strcpy(s2, s1) ;            /* copie de s1 dans s2 */
puts(s2) ;              /* affichage de s2 */
```

III.3. Fonction strcat()

Elle permet d'ajouter une chaîne de caractères à la fin d'une autre.

Exemple :

```
char s1[20], s2[20];
strcpy(s1,"ABCD"); /* copie de la première chaîne de caractères */
strcpy(s2,"EF");   /* copie de la deuxième chaîne de caractères */
strcat(s1,s2);     /* concaténer s2 à s1 */
puts(s1);          /* afficher la chaîne de caractères s1 */
```

III.4. Fonction strcmp()

Elle permet de comparer deux chaînes de caractères.

Valeur de retour :	0	si S1 = S2.
	< 0 (-1)	si S1 < S2.
	> 0 (1)	si S1 > S2.

Exemple :

```
strcpy ( s1, "ABCD") ;   /* copie de la première chaîne */
strcpy ( s2, " ABCE");   /* copie de la deuxième chaîne */
printf("%d", strcmp( s1, s2));
/* la valeur -1 s'affiche sur écran */
```

IV. EXERCICE RESOLU

Enoncé : Ecrire un programme qui permet de mettre un texte saisi à partir du clavier en majuscule.

Solution :

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

int est_miniscule(char c)
{
 return ( c >= 'a' && c<= 'z') ;
}

main( )
{
 char texte[100] ;
 int N, i ;

 gets(texte) ;                        /* Saisie du texte */
 N = strlen(texte) ;                  /* longueur du texte */
 for (i=0 ; i< N ; i++)
       if (est_miniscule(texte[i]))
             texte[i] = texte[i] + ('A' - 'a') ; /* Conversion */
 puts(texte) ;                        /* Affichage du texte */
}
```

BIBLIOGRAPHIE

[1] Claude Delannoy « Programmer en langage C » Eyrolles 2002

[2] Anne Canteaut « Programmation en langage C » disponible sur le site : http://www-rocq.inria.fr/codes/Anne.Canteaut/COURS_C

[3] Achille Braquelaire « Méthodologie de la programmation en C » Dunod 2005

[4] Département Informatique de Wikibooks « Programmation C disponible sur le site : http://fr.wikibooks.org/wiki/fr:Programmation_C »

[5] Philippe Figon « C précis et concis » O'Reilly 2003

[6] Brian W. Kernighan, Dennis M. Ritchie (Traduit de l'anglais par Jean-François Groff et Eric Mottier) « Le langage C » Editions Dunod 2004

ANNEXES

ANNEXE 1 :

RECAPITULATIF DES TYPES DU LANGAGE C

ANNEXE 2 :

CODE ASCII

ANNEXE 1 :

RECAPITULATIF DES TYPES DU LANGAGE C

Type	Taille	Bornes de l'ensemble
char	1 octet	-128 à 128 (seuls les nombres positifs peuvent coder un caractère)
unsigned char	1 octet	0 à 255
int	2 ou 4 octets	Dépend de la taille
unsigned int	2 ou 4 octets	Dépend de la taille
short	2 octets	-32768 à 32767
unsigned short	2 octets	0 à 65535
long	4 octets	-2147483648 à 2147483647
unsigned long	4 octets	0 à 4294967295
float	4 octets	3.4*10^-38 à 3.4*10^38
double	8 octets	1.7*10^-308 à 1.7*10^308
long double	10 octets	3.4*10^-4932 à 3.4*10^4932

Notation complète	Peut être remplacé par
signed char	char
signed short int	signed short
signed short	short int
signed int	short
signed long int	signed long
signed long	long int
long int	long
Signed int	int
unsigned char	unsigned char
unsigned int	unsigned
unsigned short int	unsigned short
unsigned long int	unsigned

ANNEXE 2 :

CODE ASCII

(AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE)

Dec	Hex	Char	Désignation	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	**NUL**	(null)	32	20	**Space**	64	40	**@**	96	60	**`**
1	01	**SOH**	(start of header)	33	21	**!**	65	41	**A**	97	61	**a**
2	02	**STX**	(start of text)	34	22	**"**	66	42	**B**	98	62	**b**
3	03	**ETX**	(end of text)	35	23	**#**	67	43	**C**	99	63	**c**
4	04	**EOT**	(end of transmission)	36	24	**$**	68	44	**D**	100	64	**d**
5	05	**ENQ**	(enquiry)	37	25	**%**	69	45	**E**	101	65	**e**
6	06	**ACK**	(acknowledge)	38	26	**&**	70	46	**F**	102	66	**f**
7	07	**BEL**	(bell)	39	27	**'**	71	47	**G**	103	67	**g**
8	08	**BS**	(backspace)	40	28	**(**	72	48	**H**	104	68	**h**
9	09	**TAB**	(horizontal tab)	41	29	**)**	73	49	**I**	105	69	**i**
10	0A	**LF**	(line feed)	42	2A	*****	74	4A	**J**	106	6A	**j**
11	0B	**VT**	(vertical tab)	43	2B	+	75	4B	**K**	107	6B	**k**
12	0C	**FF**	(form feed)	44	2C	**,**	76	4C	**L**	108	6C	**l**
13	0D	**CR**	(carriage return)	45	2D	**-**	77	4D	**M**	109	6D	**m**
14	0E	**SO**	(shift out)	46	2E	**.**	78	4E	**N**	110	6E	**n**
15	0F	**SI**	(shift in)	47	2F	/	79	4F	**O**	111	6F	**o**
16	10	**DLE**	(data link escape)	48	30	**0**	80	50	**P**	112	70	**p**
17	11	**DC1**	(device control 1)	49	31	**1**	81	51	**Q**	113	71	**q**
18	12	**DC2**	(device control 2)	50	32	**2**	82	52	**R**	114	72	**r**
19	13	**DC3**	(device control 3)	51	33	**3**	83	53	**S**	115	73	**s**
20	14	**DC4**	(device control 4)	52	34	**4**	84	54	**T**	116	74	**t**
21	15	**NAK**	(negative acknowledge)	53	35	**5**	85	55	**U**	117	75	**u**
22	16	**SYN**	(synchronous idle)	54	36	**6**	86	56	**V**	118	76	**v**
23	17	**ETB**	(end of transmission block)	55	37	**7**	87	57	**W**	119	77	**w**
24	18	**CAN**	(cancel)	56	38	**8**	88	58	**X**	120	78	**x**
25	19	**EM**	(end of medium)	57	39	**9**	89	59	**Y**	121	79	**y**
26	1A	**SUB**	(substitute)	58	3A	**:**	90	5A	**Z**	122	7A	**z**
27	1B	**ESC**	(escape)	59	3B	**;**	91	5B	**[**	123	7B	**{**
28	1C	**FS**	(file separator)	60	3C	<	92	5C	\	124	7C	**|**
29	1D	**GS**	(group separator)	61	3D	=	93	5D	**]**	125	7D	**}**
30	1E	**RS**	(record separator)	62	3E	>	94	5E	^	126	7E	**~**
31	1F	**US**	(unit seperator)	63	3F	**?**	95	5F	**_**	127	7F	**DEL**