Informatique II : Programmation C

2ème année Cycle Préparatoire

ENSAH

AU: 2016/2017

Semestre Autonome

Pr. S.MOQQADDEM

Plan du cours

1. Généralités sur le langage C.
2. Types de données en C
3. Opérateurs et expressions en C.
4. Les entrées sorties conversationnelles en C.
5. Structures et instructions de contrôle en C.
6. Programmation modulaire
7. Tableaux et pointeurs.
8. Structures, fichiers et allocation dynamique de la mémoire

Chapitre 1: Généralités sur le langage C

1. Généralités sur le langage C

1.1. Historique

- C a été inventé aux «Bells Laboratories » en 1972 par **Dennis M.Ritchie** avec l'objectif d'écrire un système d'exploitation(UNIX).

- En 1978 publication de livre «The C Programming Language ». par **Brian W. Kernighan** et **Dennis M.Ritchie**.

- En 1983, l'organisme ANSI (American National Standards Institute) commence le processus de normalisation du langage C. Le résultat était le standard ANSI-C.

- En 1988: deuxième édition du livre «The C Programming Language », qui respecte le standard ANSI-C. Elle est devenue la référence des programmes en C.

1. Généralités sur le langage C

1. 2. Caractéristiques du Langage C

- **Structuré:** traiter les tâches d'un programme en les mettant dans des blocs.
- **Efficace:** Possède les mêmes possibilités de contrôle de la machine que l'assembleur et il génère un **code compact et rapide.**
- **Modulaire**: Permet de découper une application en modules qui peuvent être compilés séparément.
- **Souple:** Hormis la syntaxe, peu de vérifications et d'interdits ce qui peu poser des problèmes.
- **Portable:** Permet d'utiliser le même code source sur d'autres types de machines simplement en le recompilant.
- **Extensible:** Animé par des bibliothèques de fonctions qui enrichissent le langage.

1. Généralités sur le langage C

1.3. Programme source, objet et exécutable

❑Un programme écrit en langage C forme un texte qu'on nomme **programme ou code source**, qui peut être formé de plusieurs fichiers sources.

❑ Chaque fichier source est traduit par le compilateur pour obtenir un **fichier ou module objet** (formé d'instructions machine).

❑L'éditeur de liens réunit les différents modules objets et les fonctions de la bibliothèque standard afin de former un **programme exécutable**.

1. Généralités sur le langage C

1.4. Compilateurs C

❑ Pour pouvoir écrire des programmes en C, vous avez besoin d'un compilateur C sur votre machine.

❑ Il existe plusieurs compilateurs respectant le standard ANSI-C. Une bonne liste est disponible sur : c.developpez.com/compilateurs/

❑ Nous allons utiliser l'environnement de développement CodeBlocks avec le système d'exploitation Windows.

1. Généralités sur le langage C

1.5. Composantes d'un programme C(1)

- Directives du préprocesseur
  - ✓Inclusion des fichiers d'en-tête (fichiers avec extension .h)
  - ✓Définitions des constantes avec **#define**

- Déclaration des variables globales
- Définition des fonctions (En C, le programme principal et les sous programmes sont définis comme fonctions)
- Les commentaires : texte ignoré par le compilateur, destiné à améliorer la compréhension du code /*…..*/

1. Généralités sur le langage C

1.5. Composantes d'un programme C(2)

[*directives au préprocesseur]*

[*déclarations de variables globales]*

[*fonctions secondaires]*

main()

{

déclarations de variables locales

instructions

}

1. Généralités sur le langage C

1.5. Composantes d'un programme C(3)

Exemple :

```
#include<stdio.h>
main()
{
printf( "notre premier
programme C \n");
/*ceci est un
commentaire*/
}
```

- *#include<stdio.h> informe le compilateur d'inclure le fichier stdio.h qui* contient les fonctions d'entrées-sorties dont la fonction printf
- La fonction **main est la fonction principale des programmes en C:** Elle se trouve obligatoirement dans tous les programmes. L'exécution d'un programme entraîne automatiquement l'appel de la fonction **main.**
- L'appel de printf avec l'argument "notre premier programme C\n" permet d'afficher : notre premier programme C et \n ordonne le passage à la ligne suivante
- En C, *toute instruction simple est terminée par un point-virgule ;*
- Un commentaire en C est compris entre // et la fin de la ligne ou bien entre /* et */

Chapitre 2: Types de données en C

2. Types de données en C

2.1. Instructions élémentaires(Variables)

- Une variable désigne un emplacement mémoire dont le contenu peut changer au cours d'un programme;
- Les variables servent à stocker les valeurs des données utilisées pendant l'exécution d'un programme.
- Les variables doivent être **déclarées avant d'être** utilisées, elles doivent être caractérisées par :
  - ✓ Un nom (**Identificateur)**
  - ✓ Un **type (entier, réel, …)**

2. Types de données en C

2.1. Instructions élémentaires(Variables: Identificateur)

Le choix d'un identificateur (nom d'une variable ou d'une fonction) est soumis à quelques règles :

- ❑ Doit être constitué uniquement de lettres, de chiffres et du caractère souligné _ (Eviter les caractères de ponctuation et les espaces)

correct: VAR_HT, VARHT **incorrect: VAR-HT, VAR HT, VAR.HT**

- ❑ Doit commencer par une lettre (y compris le caractère souligné)

correct : VAR1, _VAR1 **incorrect: 1VAR**

- ❑ Doit être différent des mots réservés du langage(L'ANSI C compte 32 mots clefs) :

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

2. Types de données en C

2.2.Types de variables.

- Le type d'une variable détermine l'ensemble des valeurs qu'elle peut prendre et le nombre d'octets à lui réserver en mémoire.

- En langage C, il n'y a que deux types de base **les entiers** et **les réels** avec différentes variantes pour chaque type.

2. Types de données en C

2.2.Types de variables(Types Entier)

- Le langage C distingue plusieurs types d'entiers:

TYPE	DESCRIPTION	TAILLE MEMOIRE
int	entier standard signé	4 octets: $-2^{31} \leq n \leq 2^{31}-1$
unsigned int	entier positif	4 octets: $0 \leq n \leq 2^{32}$
short	entier court signé	2 octets: $-2^{15} \leq n \leq 2^{15}-1$
unsigned short	entier court non signé	2 octets: $0 \leq\leq n \leq 2^{16}$
char	caractère signé	1 octet : $-2^{7} \leq\leq n \leq 2^{7}-1$
unsigned char	caractère non signé	1 octet : $0 \leq n \leq 2^{8}$

2. Types de données en C

2.2.Types de variables(Types Réel).

- 3 variantes de réels :

✓**float :** réel simple précision codé sur 4 octets de

$-3.4*10^{38}$ à $3.4*10^{38}$

✓**double :** réel double précision codé sur 8 octets de

$-1.7*10^{308}$ à $1.7*10^{308}$

✓ **long double :** réel très grande précision codé sur 10 octets de **$-3.4*10^{4932}$ à $3.4*10^{4932}$**

2. Types de données en C

2.3. Déclaration des variables.

❑Les déclarations introduisent les variables qui seront utilisées, fixent leur type et parfois aussi leur valeur de départ (initialisation)

❑ Syntaxe de déclaration en C

<Type> <NomVar1>,<NomVar2>,...,<NomVarN>;

❑ Exemple:

```
int i, j,k;
float x, y ;
double z=1.5; // déclaration et initialisation
short compteur;
char c=`A`;
```

2. Types de données en C

2.4. Déclaration des constantes

❑ Une constante conserve sa valeur pendant toute l'exécution d'un programme

❑ En C, on associe une valeur à une constante en utilisant :

✓ la directive ***#define*** : ***#define nom_constante valeur***

Ici la constante ne possède pas de type.

exemple: ***#define*** P_i *3.141592*

✓ le mot clé ***const*** : **const *type nom* = *expression*** ;

Dans cette instruction la constante est typée

exemple : ***const float*** P_i *=3.141592*

Chapitre 3: Opérateurs et expressions en C

3. Opérateurs et expressions en C.

3.1. L'affectation(1)

❑ En C, l'affectation est un opérateur à part entière. Elle est symbolisée par le signe =. Sa syntaxe est la suivante :

variable = expression

❑ Un opérateur est un symbole qui permet de manipuler une ou plusieurs variables pour produire un résultat.

❑ Une expression peut être une valeur, une variable ou une opération constituée par des valeurs, des constantes et des variables reliées entre eux par des opérateurs.

Exemples: 1, b, a*2, a+ 3*b-c, …

3. Opérateurs et expressions en C.

3.1. L'affectation(2)

➔L'affectation effectue une conversion de type implicite : la valeur de l'expression (terme de droite) est convertie dans le type du terme de gauche.

➔Par exemple, le programme suivant:

```
main()
 {      int i, j = 2;
      float x = 2.5;
       i = j + x;
      x = x + i;
      }
```

➔ imprime pour x la valeur 6.5 (et non 7), car dans l'instruction i = j + x;, l'expression j +x a été convertie en entier.

3. Opérateurs et expressions en C.

3.2. Les opérateurs arithmétiques

❑ Il sont l'opérateur **unaire** (– changement de signe) ainsi que les opérateurs **binaires**: (+ addition) (– soustraction) (* multiplication) (/ division) (**%** reste de la division (modulo)).

❑ Le C ne dispose que de la notation / pour désigner à la fois la division entière et la division entre flottants.

❑ Si les deux opérandes sont de type entier, l'opérateur / produira une division entière (quotient de la division). Par contre, il délivrera une valeur flottante dès que l'un des opérandes est un flottant. Par exemple:

float x;
x = 3 / 2; affecte à x la valeur 1.
Par contre x = 3 / 2.; affecte à x la valeur 1.5.

3. Opérateurs et expressions en C.

3.3. Les opérateurs relationnels

> strictement supérieur
>= supérieur ou égal
< strictement inférieur
<= inférieur ou égal
== égal
!= différent

- Leur syntaxe est: ***expression-1 op expression-2***

Les deux expressions sont évaluées puis comparées. La valeur rendue est de type int (il n'y a pas de type booléen en C); elle vaut 1 si la condition est vraie, et 0 sinon.

➔Attention à ne pas confondre l'opérateur de test d'égalité == avec l'opérateur d'affection=.

3. Opérateurs et expressions en C.

3.4. Les opérateurs logiques booléens

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

❑ Comme pour les opérateurs de comparaison, la valeur retournée par ces opérateurs est un **int** qui vaut 1 si la condition est vraie et 0 sinon.

❑ Dans une expression de type

expression-1 op-1 expression-2 op-2 ...expression-n

L'évaluation se fait de gauche à droite et s'arrête dès que le résultat final est déterminé.

3. Opérateurs et expressions en C.

3.5. Les opérateurs d'affectation composée

- Les opérateurs d'affectation composée sont:

+= -= *= /= %= &= ^= |= <<= >>=

Pour tout opérateur *op, l'expression*

expression-1 op expression-2

est équivalente à

expression-1 = expression-1 op expression-2.

3. Opérateurs et expressions en C.

3.6. Les opérateurs d'incrémentation et de décrémentation

Les opérateurs d'incrémentation ++ et de décrémentation -- s'utilisent aussi bien en suffixe (i++) qu'en préfixe (++i). Dans les deux cas la variable i sera incrémentée, la notation suffixe la valeur retournée sera l'ancienne valeur de i alors que dans la notation préfixe se sera la nouvelle. Par exemple,

```
int a = 3, b, c;
b = ++a; /* a et b valent 4 */
c = b++; /* c vaut 4 et b vaut 5 */
```

3. Opérateurs et expressions en C.

3.7. L'opérateur virgule

Une expression peut être constituée d'une suite d'expressions séparées par des virgules : ***expression-1, expression-2, ... , expression-n***

➔Cette expression est alors évaluée de gauche à droite. Sa valeur sera la valeur de l'expression de droite.

➔Par exemple, le programme:

```
main()
{
int a, b;
b = ((a = 3), (a + 2));
printf("\n b = %d \n",b);
}
```

imprime b = 5.

3. Opérateurs et expressions en C.

3.8. L'opérateur conditionnel ternaire

L'opérateur conditionnel ? est un opérateur ternaire. Sa syntaxe est la suivante : ***condition ? expression-1 : expression-2***

Cette expression est égale à ***expression-1*** si la condition est satisfaite, et à ***expression-2*** sinon.

Par exemple, l'expression

x >= 0 ? x : -x ➔correspond à la valeur absolue d'un nombre.

De même l'instruction

m = ((a > b) ? a : b); ➔ affecte à m le maximum de a et de b.

3. Opérateurs et expressions en C.

3.9. L'opérateur de conversion de type

L'opérateur de conversion de type permet de modifier explicitement le type d'un objet. On écrit ***(type) objet***

Par exemple:

```
main()
{  int i = 3, j = 2;
printf("%f \n",(float)i/j);}
```

retourne la valeur 1.5.

3.10. L'opérateur adresse

L'opérateur d'adresse **&** appliqué à une variable retourne l'adresse-mémoire de cette variable. La syntaxe est ***&objet***

Exercices: ANALYSE DE CODE

1. Déterminer les valeurs de res1, res2 et res3:

```
int nbr1, nbr2;
int res1, res2, res3;
nbr1 = 3;
nbr2 = 4;
res1 = nbr2%nbr1;
res2 = 45%4;
res3 = 45%nbr1;
```

3. Chercher et expliquer les erreurs du programme :

```
const int X=10;
int y,z;
y+z = 10;
10 = z;
X=100;
z==10;
```

2. Déterminer les valeurs successives de k et x:

```
int i=10, j=4, k;
float x;
k=i+j;
x=i;
x=i/j;
x=(float)i/j;
x=5/i;
k=i%j;
x=105; i=5;
x= x+i;
```

Chapitre 4: Les entrées sorties conversationnelles en C.

4. Les entrées sorties conversationnelles en C

Il s'agit des fonctions de la librairie standard **stdio.h** utilisées avec les unités classiques d'entrées-sorties, qui sont respectivement le clavier et l'écran.

- ❑ Lecture des caractères: **getchar.**
- ❑ Ecriture des caractères: **putchar.**
- ❑ Entrée des données: **scanf.**
- ❑ Sortie des données: **printf.**

N.B: Un programme écrit en C s'exécute séquentiellement, c'est à dire instruction après instruction.

4. Les entrées sorties conversationnelles en C

- **4.1.Lecture des caractères:** La fonction externe **getchar** a pour rôle de lire un caractère en entrée(clavier). Syntaxe : var_car=**getchar()**

Exemple:

```
char A;
A=getchar();
```

- **4.2Ecriture des caractères**: le rôle de la fonction externe **putchar** est de permettre l'affichage des caractères à l'écran. Syntaxe: **putchar(var_car)**

Exemple:

```
Char G='S';
putchar(G);
```

4. Les entrées sorties conversationnelles en C

4.3. Entrée des données: le rôle de la fonction externe scanf est de permettre la lecture des données en entrée. Ces données peuvent être de différents type: numériques, caractères simples, chaînes de caractères…

- Syntaxe: **scanf(**chaîne de format, arg1, arg2,….,argn)
- La chaîne de format se compose de groupes de caractères de format associés individuellement à chacune des données en entrée.
- Chaque groupe de caractère de format est constitué de % suivi d'une spécification de forme indiquant le type de la donnée à laquelle elle est associée(voir tableau).

4. Les entrées sorties conversationnelles en C

4.3. Entrée des données:

caractère de conversion	signification
c	caractère simple
d	donnée type entier décimal
e	donnée type virgule flottante
f	donnée type virgule flottante
g	donnée type virgule flottante
s	type chaîne de caractères
o	donnée type entier octal
x	donnée type entier hexadécimal
U	donnée type entier décimal non signé

N.B: Le type données argument doit être cohérent avec la chaîne de format associée.

4. Les entrées sorties conversationnelles en C

4.3. Entrée des données:

- arg1, arg2,…,argn représentent les différentes données figurent en entrée. En fait, ces arguments indiquent les adresses des données en mémoires (&variable).
- Tous les noms de variables à l'exception des tableaux doivent être précédés par le symbole &.

Exemple 1:

```
int a;
scanf("%d",&a);
```

Exemple 2:

```
float x;
scanf("%f",&x);
```

4. Les entrées sorties conversationnelles en C

4.4. Sortie des données: La fonction **printf** permet d'écrire des données en provenance de l'ordinateur sur l'unité de sortie standard. Les données peuvent être de différents types: numérique, caractères simples ou chaînes…

- **Syntaxe: printf(chaîne de format, arg1, arg2,…,argn)**
- Chaîne de format fait référence à une chaine contenant des informations précisant les caractéristiques d'affichage
- arg1, arg2,…,argn représentent les différentes données à afficher: constantes, variables, tableaux, expressions.
- Contrairement à la fonction scanf, les arguments ne représentent pas des adresses mémoire.

4. Les entrées sorties conversationnelles en C

4.4. Sortie des données:

- La chaîne de format se compose de groupes de caractères commençant par % et suivi de spécification de format(voir tableau ci-dessous)

caractère de conversion	signification
C	caractère simple
d	entier décimal signé
e	valeur réelle avec exposant
f	valeur réelle sans exposant
g	valeur réelle simple
s	chaîne de caractères
o	entier octal non précédé d'un o
x	entier hexadécimal non précédé d'un ox
u	entier décimal non signé
I	Entier décimal signé

Chapitre 5: Structures et instructions de contrôle en C.

5. Structures et instructions de contrôle en C.

5.1. Les instructions de contrôle(de branchement conditionnel)

a) L'instruction **if else**: Dans certain programme, où on désire choisir entre deux instructions selon une condition, on utilise la structure alternative **if**.

Syntaxe

```
if (condition)
        instruction1
else
        instruction2
```

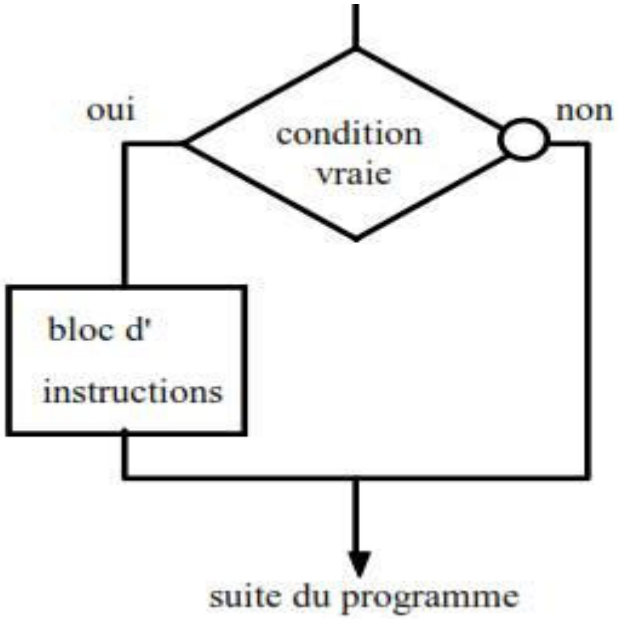
avec condition quelconque, instruction1 et instruction2 sont soit:

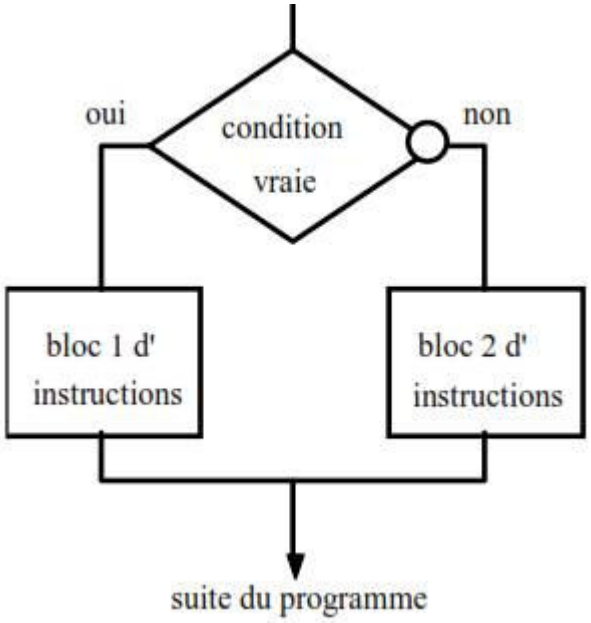
- Simples(terminées par un point-virgule;)
- Blocs (délimités par { et })
- Instructions structurés(boucles).

5. Structures et instructions de contrôle en C.

5.1. Les instructions de contrôle(de branchement conditionnel)

a) L'instruction **if et if…else**





5. Structures et instructions de contrôle en C.

5.1. Les instructions de contrôle(de branchement conditionnel)

Exemple1: if sans else	Exemple2: if avec else	Exemple 3 : if imbriquée

int a =3, b=7, max ;
max=a ;
if (max < b)
max = b ;
printf("%d \n", max) ;
	int a = -3, valeurAbs;
	if (a > 0)
	valeurAbs = a ;
	else
	valeurAbs = -a;
	printf("%d\n",valeurAbs);
		int a = -3, b= 7, c= 2 , max;
		if (a > b)
		if (a > c)
		max = a ;
		else
		max = c;
		else
		if (b > c)
		max = b ;
		else
		max = c;

5. Structures et instructions de contrôle en C.

5.1. Les instructions de contrôle

b) L'instruction switch :

Syntaxe:

```
switch (expression)
    { case constante1: [suite_instructions1]
      case constante2: [suite_instructions2]
      ……
      case constanten : [suite_instructionsn]
      [default        :  suite_instructions  ]
}
```

5. Structures et instructions de contrôle en C.

5.1. Les instructions de contrôle

b) L'instruction switch :

avec

- constantei une expression constante entière (char sera accepté car il sera converti en int).
- Suite_instructions: séquence d'instructions quelconque. Ce qui est entre crochets [] est facultatif.
- L'usage du break dans l'instruction switch correspond à un débranchement qui permet de quitter le switch.

5. Structures et instructions de contrôle en C.

5.2. Les instructions de branchement inconditionnel

a) L'instruction break :

Elle sert à interrompre le déroulement de la boucle en cours d'exécution à l'instruction qui suit cette boucle.

b) L'instruction continue:

Elle permet d'interrompre l'itération courante de la boucle et de passer à l'itération suivante.

5. Structures et instructions de contrôle en C.

5.2. Les instructions de branchement inconditionnel

a) L'instruction break peut être utilisée dans une boucle (for, while, ou do .. while).Elle permet d'arrêter le déroulement de la boucle et le passage à la première instruction qui la suit.

En cas de boucles imbriquées, break ne met fin qu' à la boucle la plus interne

```
 {int i,j;
for(i=0;i<4;i++)
for (j=0;j<4;j++)
{ if(j==1) break;
printf("i=%d,j=%d\n ",i,j);
}
}
```

résultat: i=0,j=0
i=1,j=0
i=2,j=0
i=3,j=0.

5. Structures et instructions de contrôle en C.

5.2. Les instructions de branchement inconditionnel

L'instruction continue: peut être utilisée dans une boucle (for, while, ou do .. while). Elle permet l'abandon de l'itération courante et le passage à l'itération suivante

➔ **{int i;**
for(i=1;i<5;i++)
{printf("début itération %d\n " ,i);
if(i<3) continue;
printf(" fin itération %d\n " ,i);}}

résultat: début itération 1
début itération 2
début itération 3
fin itération 3
début itération 4
fin itération 4

5. Structures et instructions de contrôle en C.

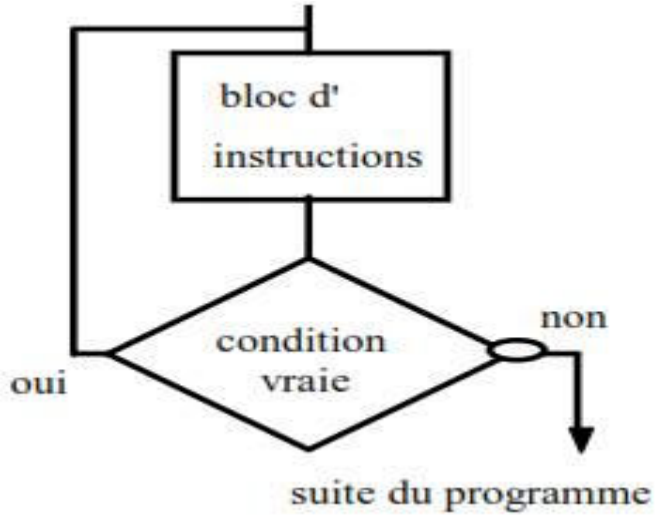
5.3. Les instructions structurés(Boucles):

Dans certain programme, où on désire répéter un bloc instructions plusieurs fois, on utilise les boucles.

a) Boucle do…..while

La syntaxe de la boucle **do … while** a la forme suivante:

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



5. Structures et instructions de contrôle en C.

5.3. Les instructions structurés(Boucles):

a) Boucle do…..while

Exemple 1 : s= 1 + 2 + 3 + ….+ 10

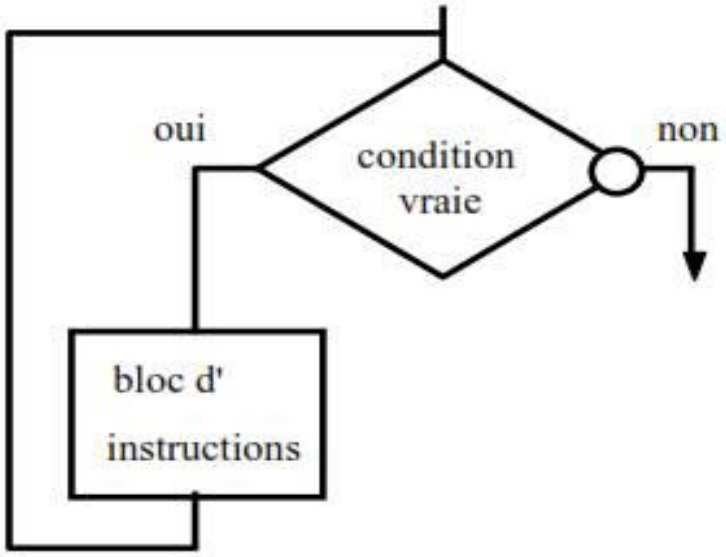
```
int s = 0 ;
int i=1 ;
do
{ s = s + i;
i++;
} while(i<=10);
printf ("la somme est: %d\n " ,s);
```

5. Structures et instructions de contrôle en C.

5.3. Les instructions structurés(Boucles):

b) Boucle **while:** Il s'agit de l'instruction:

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



- L'expression utilisée comme condition de la boucle est évaluée avant la première itération de la boucle. Ainsi, il est nécessaire que sa valeur soit définie à ce moment.

5. Structures et instructions de contrôle en C.

5.3. Les instructions structurés(Boucles):

b) L'instruction **while:**

Exemple 1 : s= 1 + 2 + 3 + ….+ 10

```
int s = 0 ;
int i=1 ;
while(i<=10)
{ s = s + i;
i++; }
printf ("la somme est:%d \n " ,s);
```

Exemple 2 : Ecrire un programme qui lit deux entiers a et b au clavier et affiche leur Plus Grand Commun Diviseur (PGCD). On suppose que a >=b.

5. Structures et instructions de contrôle en C.

5.3. Les instructions structurés(Boucles):

Boucle **while:**

```
#include <stdio.h>
  main()
{ int a, b , r ;
 printf("Entrez a: \t");
   scanf("%d\n",&a);
 printf("Entrez b: \t");
   scanf("%d\n",&b);
   r = a%b;
 while (r!=0)
{ a = b;
b = r;
 r = a%b; }
printf ("PGCD =%d \n" ,b ); }
```

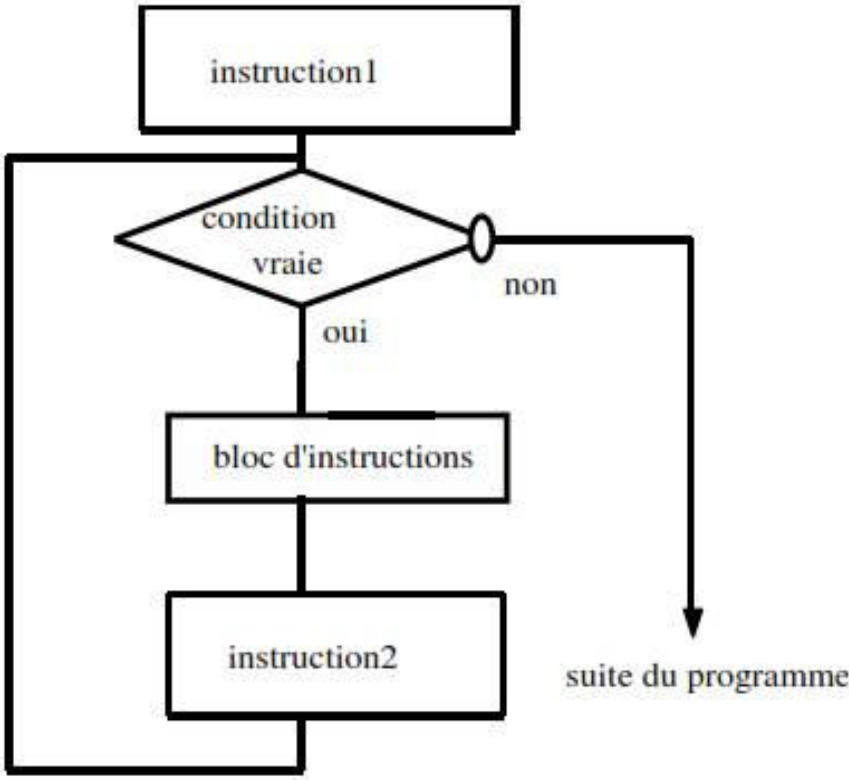
5. Structures et instructions de contrôle en C.

5.3. Les instructions structurés(Boucles):

c) Boucle **for**

Il s'agit de l'instruction:

for (instruction1 ; condition ; instruction2)
{
bloc d'instructions
}



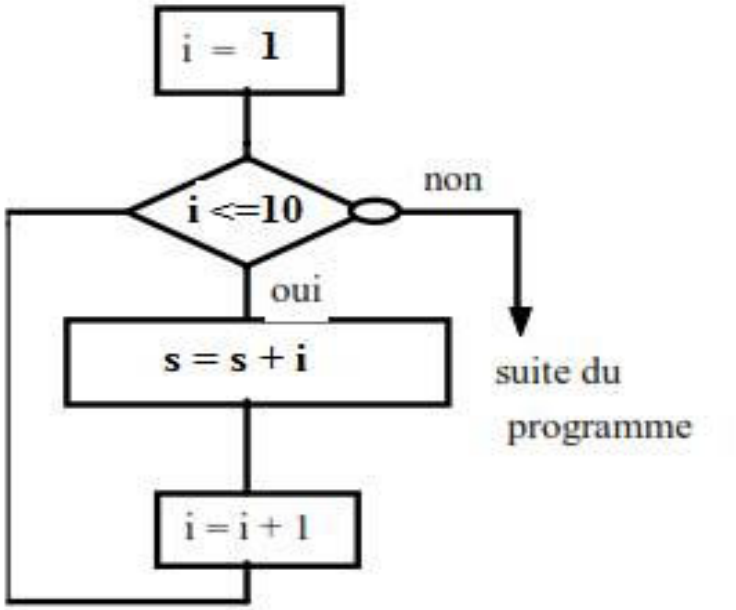
5. Structures et instructions de contrôle en C.

5.3. Les instructions structurés(Boucles):

c) Boucle **for**

Exemple: s= 1 + 2 + 3 + ….+ 10

```
int s = 0 ;
for( int i=1 ; i<=10 ; i++)
s = s + i;
printf("la somme est: %d",s);
```



5. Structures et instructions de contrôle en C.

5.3. Les instructions structurés(Boucles):

c) Boucles **for et while**

Cet expression de **for** est équivalent à:

```
instruction1;
 while(condition)
{ bloc d'instructions
instruction2;
 }
```

```
for ( instruction1 ; condition ; instruction2)
  {
      bloc d'instructions
  }
```

5. Structures et instructions de contrôle en C.

5.3. Les instructions structurés(Boucles):

c) Boucles **for**

Exemple 2 : Ecrire un programme qui lit un entier n au clavier et affiche ses diviseurs.

```
#include <stdio.h>
main()
{ int n ;
printf("Entrez n: ");
scanf("%d", &n);
printf ("Les diviseurs de %d sont: \n", n);
for( int i=1 ; i<=n ; i++)
{ if (n%i == 0)
printf("%d \n", i );
 }
```

Chapitre 6: Programmation modulaire.

6. Programmation modulaire: les fonctions

6.1. Introduction:

- Caractéristiques du Langage C: modulaire.
- Certains problèmes conduisent à des programmes longs, difficiles à écrire et à comprendre ➔ On les découpe en des parties appelées **sous-programmes** ou **modules**
- Les modules sont des groupes d'instructions qui fournissent une solution à des parties bien définies d'un problème plus complexe.

6. Programmation modulaire: les fonctions

6.1. Introduction:

Les modules ont plusieurs **intérêts** :

- permettent de **"factoriser" les programmes**, càd de mettre en commun les parties qui se répètent.
- permettent une **structuration** et une **meilleure lisibilité** des programmes.
- **facilitent la maintenance** du code (il suffit de modifier une seule fois) peuvent éventuellement être **réutilisées** dans d'autres programmes.

6. Programmation modulaire: les fonctions

6.1. Introduction:

➔ La structuration de programmes en sous-programmes se fait en C à l'aide des **fonctions**

- Une seule de ces fonctions existe obligatoirement ; c'est la fonction principale appelée main. Cette fonction principale peut, éventuellement, appeler une ou plusieurs fonctions secondaires.
- De même, chaque fonction secondaire peut appeler d'autres fonctions secondaires ou s'appeler elle-même (dans ce dernier cas, on dit que la fonction est *récursive*).

6. Programmation modulaire: les fonctions

6.2. Présentation des fonctions:

- Pour permettre de réutiliser une portion de programme sans réécrire les lignes, on utilise les fonctions. Lorsqu'on appelle une fonction, une portion de code est exécutée.

- Une fonction est un sous-programme nommé destiné à faire un traitement bien précis. Elle
  - reçoit (en entrée) des informations : les arguments : données.
  - renvoie (en sortie) une information, la valeur de retour.

6. Programmation modulaire: les fonctions

6.2. Présentation des fonctions:

On peut distinguer, en langage C, les fonctions prédéfinies des bibliothèques (telles que printf() ou scanf()), livrées avec le compilateur et « intégrées » au programme lors de l'édition des liens, et les fonctions que le programmeur écrit lui-même en tant que partie du texte source.

6. Programmation modulaire: les fonctions

6.2. Présentation des fonctions:

La syntaxe de définition d'une fonction a la forme suivante :

type nomFonction(liste des types et noms des arguments)

{ Corps de la fonction }

Où :

- **type** est le type de la valeur retournée,
- **nomFonction** est le nom de la fonction,
- la liste des types et nom des arguments séparés par des virgules,
- le corps de la fonction : les instructions entre accolades.

6. Programmation modulaire: les fonctions

6.2. Présentation des fonctions:

Dans la première ligne (appelée **en-tête de la fonction**) :

type nomFonction(liste des types et noms des arguments)

- **type** est le type du résultat retourné. Si la fonction n'a pas de résultat à retourner, elle est de type **void.**
- le choix d'un nom de fonction doit respecter les mêmes règles que celles adoptées pour les noms de variables.(à voir)
- entre parenthèses, on spécifie les **arguments** de la fonction et leurs types. Si une fonction n'a pas de paramètres, on peut déclarer la liste des paramètres comme **(void)** ou simplement comme **()**

6. Programmation modulaire: les fonctions

6.2. Présentation des fonctions:

➔ Pour fournir un résultat en quittant une fonction, on dispose de la commande **return.**

{ Corps de la fonction }

6. Programmation modulaire: les fonctions

6.3. Exemples:

❑ Une fonction qui calcule la somme de deux réels x et y :

```
int Som(int x, int y )
{return (x+y);}
```

❑ Une fonction qui affiche la somme de deux réels x et y :

```
void AfficheSom(float x, float y)
{printf (" %f", x+y );}
```

❑ Une fonction qui renvoie un entier saisi au clavier

```
int RenvoieEntier( void )
{int n;
printf (" Entrez n \n");
scanf (" %d ", &n);
return n;}
```

6. Programmation modulaire: les fonctions

6.4. Appel d'une fonction

des paramètres : **nom_fonction (para1,…, paraN)**

- Lors de l'appel d'une fonction, les paramètres sont appelés **paramètres effectifs** : ils contiennent les valeurs pour effectuer le traitement.
- Lors de la définition, les paramètres sont appelés **paramètres formels**.
- L'ordre et les types des paramètres effectifs doivent correspondre à ceux des paramètres formels.

6. Programmation modulaire: les fonctions

6.4. Appel d'une fonction

Exemple d'appels	Exemples de définition

main( )
{ double z;
z=Som(2.5, 7.3);
void AfficheSom(2.5, 7.3);
}
	int Som(int x, int y)
	{return (x+y);}
	
	void AfficheSom(float x, float y)
	{printf (" %f", x+y);}

6. Programmation modulaire: les fonctions

6.5. Déclaration des fonctions

- Il est nécessaire pour le compilateur de connaître la définition d'une fonction au moment où elle est appelée.
- Si une fonction est définie après son premier appel (en particulier si elle définie après main), elle doit être **déclarée** auparavant.
- La déclaration d'une fonction se fait par son **prototype** qui indique les types de ses paramètres et celui de la fonction :

type nom_fonction (**type1**,…, **typeN**)

6. Programmation modulaire: les fonctions

6.5. Déclaration des fonctions

- Il est interdit en C de définir des fonctions à l'intérieur d'autres fonctions. En particulier, on doit définir les fonctions soit avant, soit après la fonction principale main.

6. Programmation modulaire: les fonctions

6.5. Déclaration des fonctions: Exemple

```
#include<stdio.h>
float ValeurAbsolue(float);     //prototype de la fonction ValeurAbsolue
main( )
{
float x=-5.7,y;
y= ValeurAbsolue(x);            //L'appel de cette fonction
printf("La valeur absolue de %f est : %f \n " , x,y);
}
float ValeurAbsolue(float a) //Définition de la fonction ValeurAbsolue
{if (a<0) a=-a;
return a;}
```

6. Programmation modulaire: les fonctions

6.6. Variables locales et globales

➔ On peut manipuler 2 types de variables dans un programme C : des **variables locales** et des **variables globales**. Elles se distinguent par ce qu'on appelle leur **portée** (leur "espace de visibilité", leur "durée de vie")

6. Programmation modulaire: les fonctions

6.6. Variables locales et globales

- Une variable définie à l'intérieur d'une fonction est une **variable locale**, elle n'est connue qu'à l'intérieur de cette fonction. Elle est créée à l'appel de la fonction et détruite à la fin de son exécution.

- Une variable définie à l'extérieur des fonctions est une **variable globale**. Elle est définie durant toute l'application et peut être utilisée et modifiée par les différentes fonctions du programme.

6. Programmation modulaire: les fonctions

6.6. Variables locales et globales: remarques

- Les variables déclarées au début de la fonction principale main ne sont pas des variables globales, mais elles sont locales à main.

- Une variable locale cache la variable globale qui a le même nom.

➔ Il faut utiliser autant que possible des variables locales. Ceci permet d'économiser la mémoire et d'assurer l'indépendance de la fonction.

6. Programmation modulaire: les fonctions

6.6. Variables locales et globales: remarques

➔ En C, une variable déclarée dans un bloc d'instructions est uniquement visible à l'intérieur de ce bloc. C’est une variable locale à ce bloc, elle cache toutes les variables du même nom des blocs qui l'entourent.

6. Programmation modulaire: les fonctions

6.6. Variables locales et globales: Exemple

```
#include<stdio.h>
int x = 7;
int f(int);
int g(int);
main( )
{ printf("x = %d\t", x);
 int x = 6; printf("x = %d\t", x);
printf("f(%d) = %d\t", x, f(x));
printf("g(%d) = %d\t", x, g(x));}

int f(int a) { int x = 9; return (a + x); }
int g(int a) { return (a * x); }
```

Qu'affiche ce programme?

x=7

x=6

f(6)=15

g(6) = 42

6. Programmation modulaire: les fonctions

6.6. Variables locales et globales: Exemple

```
#include<stdio.h>
void f(void);
int i;
main()
{ int k = 5;
i=3; f(); f();
printf("i = %d et k=%d \n", i,k); }

void f(void)
{ int k = 1;
printf("i = %d et k=%d \n", i,k);
i++;k++;}
```

Qu'affiche ce programme?

i=3 et k=1

i=4 et k=1

i=5 et k=5

Chapitre 7: Tableaux , Chaînes de caractères et pointeurs.

7. Tableaux, Chaînes de caractères et pointeurs.

7.1. Les Tableaux

Les tableaux correspondent aux **vecteurs** et **matrices** en mathématiques. Un tableau est caractérisé par sa taille et par le type de ses éléments.

A. **Tableaux à une dimension (vecteurs)**

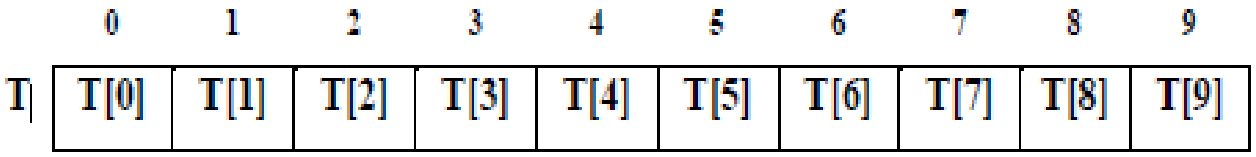
B. **Tableaux de deux dimensions (matrices)**

7. Tableaux, Chaînes de caractères et pointeurs.

7.1. Les Tableaux

A. Tableaux à une dimension :

- Un tableau (uni-dimensionnel) est une variable structurée formée d'un nombre entier N de variables simples du même type, qui sont appelées les ***composantes*** du tableau.
- Le nombre de composantes N est alors la ***dimension*** du tableau (appelée taille du tableau), elle doit être une valeur **constante entière**.
- Soit T un tableau de taille N=10, un élément du tableau est repéré par son indice. Les indices d'un tableau commencent de 0. L'indice maximum est donc N-1. i est l'indice de l'élément T[i], avec i=0, …, N-1



7. Tableaux, Chaînes de caractères et pointeurs.

7.1. Les Tableaux

A. Tableaux à une dimension :

- Déclaration sans initialisation:

<TypeSimple> <NomTableau>[<Dimension>];

- ✓ int mois[12];
- ✓ double abscisse[100], ordonnee[100];
- ✓ float notes[500] ;

- Déclaration avec initialisation : Lors de la déclaration d'un tableau, on peut initialiser les composantes du tableau, en indiquant la liste des valeurs respectives entre accolades. Par exemple:

int T[6] = {1,3,8,0,5,9};

T :

0	1	2	3	4	5
T[0]=1	T[1]=3	T[2]=8	T[3]=0	T[4]=5	T[5]=9

7. Tableaux, Chaînes de caractères et pointeurs.

7.1. Les Tableaux

A. Tableaux à une dimension : Initialisation

- Dans le cas ou l'initialisation et la déclaration sont simultanées, la taille du tableau peut être omise, le compilateur la calcule d'après le nombre de valeurs d'initialisation :

int T[] = {10,20,30,40,50,60,70,80,90};

- On peut se contenter de n'initialiser que le début du tableau.

double resistances[12]={1., 1.2, 1.8, 2.2};

Si la liste ne contient pas assez de valeurs pour toutes les composantes, les composantes restantes sont initialisées par zéro.

7. Tableaux, Chaînes de caractères et pointeurs.

7.1. Les Tableaux

A. Tableaux à une dimension : Initialisation

Un tableau peut être initialisé plus tard, en affectant une valeur à chaque élément :

```
float notes[50];
for (int i=0; i<50; i++)
{notes[i]=20; }
notes[3] = 13;
notes[5] = 9;
```

7. Tableaux, Chaînes de caractères et pointeurs.

7.1. Les Tableaux

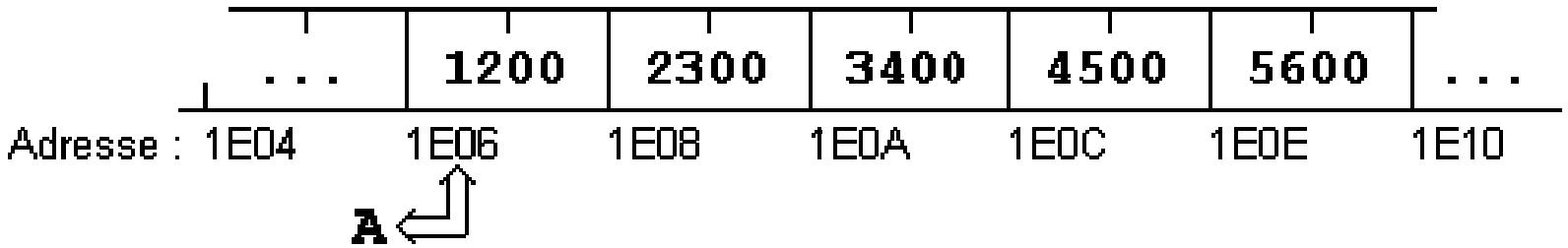
A. Tableaux à une dimension :

- Taille du tableau : une valeur constante.

```
const int NMAX = 100;
double mesures[NMAX];
```

- **Mémorisation:** 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. Par exemple**:**

short A[5] = {1200, 2300, 3400, 4500, 5600};



7. Tableaux, Chaînes de caractères et pointeurs.

7.1. Les Tableaux

A. Tableaux à une dimension : Remplissage

➔ **Affectation avec des valeurs provenant de l'extérieur**

```
int main() {
int A[5];
int I; /* Compteur */
for (I=0; I<5; I++)
scanf("%d", &A[I]);
 return 0; }
```

Comme **scanf** a besoin des adresses des différentes composantes du tableau, il faut faire précéder le terme A[I] par l'opérateur adresse '**&**'.

7. Tableaux, Chaînes de caractères et pointeurs.

7.1. Les Tableaux

A. Tableaux à une dimension :

- Affichage du contenu d'un tableau

```
int main()
{ int A[5];
 int I; /* Compteur */
 for (I=0; I<5; I++)
printf("%d \t ", A[I]);
return 0; }
```

7. Tableaux, Chaînes de caractères et pointeurs.

7.1. Les Tableaux

B. Tableaux de deux dimension : est une matrice en mathématiques.

Un tableau à deux dimensions A est à interpréter comme un tableau (uni-dimensionnel) de dimension L dont chaque composante est un tableau (uni-dimensionnel) de dimension C. On appelle L le **nombre de lignes** du tableau et C le **nombre de colonnes** du tableau. L et C sont alors les deux **dimensions** du tableau. Un tableau A à deux dimensions contient donc **L*C**

Chaque élément **A[i][j]** est repéré par ses indices **i** et **j**.

L'indice **i** commence de **0** à **L-1** et l'indice j commence de **0** à **C-1**.

7. Tableaux, Chaînes de caractères et pointeurs.

7.1. Les Tableaux

B. Tableaux de deux dimension :

Exemple

Considérons un tableau NOTES à une dimension pour mémoriser les notes de 20 élèves d'une classe dans un devoir:

int NOTE[20] = {45, 34, ... , 50, 48};

Pour mémoriser les notes des élèves dans les 10 devoirs d'un trimestre, nous pouvons rassembler plusieurs de ces tableaux uni-dimensionnels dans un tableau NOTES à deux dimensions :

```
int NOTE[10][20] = {{45, 34, ... , 50, 48},
                    {39, 24, ... , 49, 45},
                    ... ... ...
                    {40, 40, ... , 54, 44}};
```

7. Tableaux, Chaînes de caractères et pointeurs.

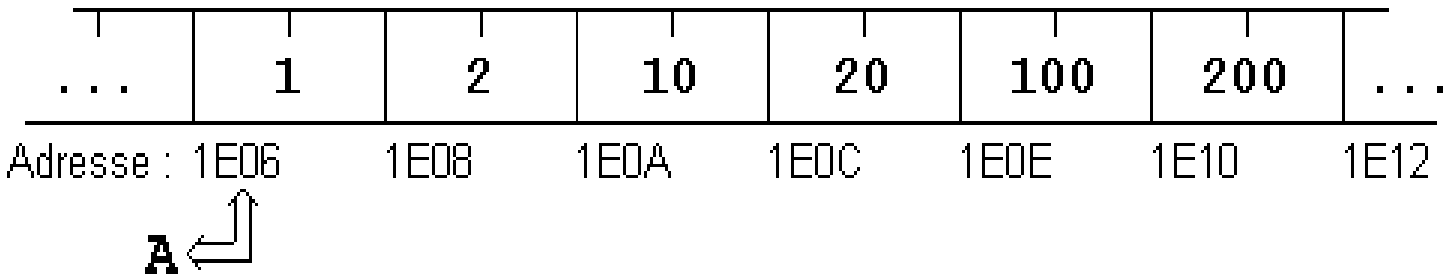
7.1. Les Tableaux

B. Tableaux de deux dimension : Déclarations

<TypeSimple> <NomTabl>[<DimLigne>][<DimCol>];

- **Exemples:**
  - int A[10][10];
  - float B[2][20];
  - int C[3][3];
  - char D[15][40];
- **Mémorisation:** Les composantes d'un tableau à deux dimensions sont stockées ligne par ligne dans la mémoire.

short A[3][2] = {{1, 2 }, {10, 20 }, {100, 200}};



7. Tableaux, Chaînes de caractères et pointeurs.

7.1. Les Tableaux

B. Tableaux de deux dimension : Initialisation et réservation automatique

- Lors de la déclaration d'un tableau, on peut initialiser les composantes du tableau, en indiquant la liste des valeurs respectives entre accolades.
- A l'intérieur de la liste, les composantes de chaque ligne du tableau sont encore une fois comprises entre accolades.
- Pour améliorer la lisibilité des programmes, on peut indiquer les composantes dans plusieurs lignes.

```
int A[3][10]={{ 0,10,20,30,40,50,60,70,80,90},
              {10,11,12,13,14,15,16,17,18,19},
              {1,12,23,34,45,56,67,78,89,90}};
```

7. Tableaux, Chaînes de caractères et pointeurs.

7.1. Les Tableaux

B. Tableaux de deux dimension : Initialisation et réservation automatique

Si le nombre de **lignes L** n'est pas indiqué explicitement lors de l'initialisation, l'ordinateur réserve automatiquement le nombre d'octets nécessaires.

```
int A[][10] ={{ 0,10,20,30,40,50,60,70,80,90},
              {10,11,12,13,14,15,16,17,18,19},
              {1,12,23,34,45,56,67,78,89,90}};
```

Réservation de 3*10*2 = 60 octets

7. Tableaux, Chaînes de caractères et pointeurs.

7.1. Les Tableaux

B. Tableaux de deux dimension : Accès aux composantes

<NomTableau>[<Ligne>][<Colonne>]

➔ Considérons un tableau A de dimensions **L** et **C**.

les indices du tableau varient de **0** à **L-1**, respectivement de **0** à **C-1**.

la composante de la N^{ième} ligne et M^{ième} colonne est notée:

A[N-1][M-1]

7. Tableaux, Chaînes de caractères et pointeurs.

7.1. Les Tableaux

B. Tableaux de deux dimension : Affectation et affichage

Pour parcourir les lignes et les colonnes des tableaux à deux dimensions, nous utiliserons deux indices (i et j), et la structure **for**, souvent imbriquée.

❑ **Affectation avec des valeurs provenant de l'extérieur**

```
main()
{ int A[5][10];
int i,j; /* Pour chaque ligne ... */
for (I=0; I<5; I++) /* ... considérer chaque composante */
for (J=0; J<10; J++)
scanf("%d", &A[I][J]);
return 0; }
```

7. Tableaux, Chaînes de caractères et pointeurs.

7.1. Les Tableaux

B. Tableaux de deux dimension : Affectation et affichage

Pour parcourir les lignes et les colonnes des tableaux à deux dimensions, nous utiliserons deux indices (i et j), et la structure **for**, souvent imbriquée.

❑ **Affichage**

```
int main()
 { int A[5][10];
int I,J; /* Pour chaque ligne ... */
for (I=0; I<5; I++) { /* ... considérer chaque composante */
 for (J=0; J<10; J++)
printf("%d", A[I][J]); /* Retour à la ligne */
 printf("\n"); } return 0; }
```

7. Tableaux, Chaînes de caractères et pointeurs.

7.1. Les Tableaux

- **Schéma de traitement:**
  - Déclaration des variables
  - Lecture des dimensions des tableaux
  - Lecture des données des tableaux
  - Traitements
  - Affichage des résultats
- **Exemples d'application**
  - Ecrire un programme qui calcule le produit scalaire de deux vecteurs d'entiers U et V (de même dimension).
  - Ecrire un programme qui met à zéro les éléments de la diagonale principale d'une matrice ***carrée*** A donnée.

7. Tableaux, Chaînes de caractères et pointeurs.

7.2. Chaîne de caractères

- Les chaînes de caractères servent à stocker les informations non numériques comme par exemple une liste de nom de personne ou des adresses.

- Il n'existe pas de type spécial *chaîne* ou *string* en C. **Une chaîne de caractères est traitée comme un tableau à une dimension de caractères (vecteur de caractères).**

- Il existe quand même des notations particulières et une bonne quantité de fonctions spéciales pour le traitement de tableaux de caractères.

7. Tableaux, Chaînes de caractères et pointeurs.

7.2. Chaîne de caractères: Déclaration d'une chaîne

- Une chaîne de caractères est un tableau de type char. La déclaration est identique à un tableau normal:

  char <nom_chaine> [<dimension>].

- La représentation interne d'une chaîne de caractères est terminée par le symbole '\0' (NULL)➔ Ainsi, pour un texte de *n* caractères, nous devons prévoir n+1 octets.

- Le compilateur C ne contrôle pas si nous avons réservé un octet pour le symbole de fin de chaîne; l'erreur se fera seulement remarquer lors de l'exécution du programme ...

7. Tableaux, Chaînes de caractères et pointeurs.

7.2. Chaîne de caractères: Initialisation

- En général, les tableaux sont initialisés par l'indication de la liste des éléments du tableau entre accolades:

char MACHAINE[ ] = {'H','e','l','l','o','\0'};

- Pour le cas spécial des tableaux de caractères, nous pouvons utiliser une initialisation plus confortable en indiquant simplement une chaîne de caractères constante: char MACHAINE[ ] = "Hello";

- Lors de l'initialisation par [ ], l'ordinateur réserve automatiquement le nombre d'octets nécessaires pour la chaîne, c.-à-d.: le nombre de caractères + 1 (ici: 6 octets).

7. Tableaux, Chaînes de caractères et pointeurs.

7.2. Chaîne de caractères: Initialisation

- **Exemples:**
  - char MACHAINE[ ] = “Hello”;
  - char MACHAINE[6] = “Hello”;
  - char MACHAINE[ ] = {‘H’,’e’,’l’,’l’,’o’,’\0’};
  - char MACHAINE[8] = “Hello”;

➔ Nous pouvons aussi indiquer explicitement le nombre d'octets à réserver, si celui-ci est supérieur ou égal à la longueur de la chaîne d'initialisation. Par contre:

- char MACHAINE[5] = “Hello”; donnera une erreur à l’exécution
- char MACHAINE[4] = “Hello”; donnera une erreur à la compilation.

7. Tableaux, Chaînes de caractères et pointeurs.

7.2. Chaîne de caractères: Chaînes constantes et caractères

Pour la mémorisation de la chaîne de caractères "Hello", C a besoin de six (!!) octets.

- 'x' est un caractère constant, qui a une valeur numérique: Par exemple: 'x' à la valeur 120 dans le code ASCII.
- "x" est un tableau de caractères qui contient deux caractères: la lettre 'x' et le caractère NUL: '\0'

➔ 'x' est codé dans un octet

➔ "x" est codé dans deux octets

7. Tableaux, Chaînes de caractères et pointeurs.

7.2. Chaîne de caractères: Chaînes constantes et caractères

- Les chaînes de caractères constantes (string literals) sont indiquées entre guillemets. La chaîne de caractères vide est alors: " "
- Dans les chaînes de caractères, nous pouvons utiliser toutes les séquences d'échappement définies comme caractères constants: "Ce \n texte \n sera réparti sur 3 lignes."
- Le symbole " peut être représenté à l'intérieur d'une chaîne par la séquence d'échappement \".

7. Tableaux, Chaînes de caractères et pointeurs.

7.2. Chaîne de caractères: Chaînes constantes et caractères

- Le symbole ' peut être représenté à l'intérieur d'une liste de caractères par la séquence d'échappement \' : {'L','\'','a','s','t','u','c','e','\0'}

- Plusieurs chaînes de caractères constantes qui sont séparées par des signes d'espacement (espaces, tabulateurs ou interlignes) dans le texte du programme seront réunies en une seule chaîne constante lors de la compilation:

"un " "deux" " trois " sera évalué à "un deux trois"

7. Tableaux, Chaînes de caractères et pointeurs.

7.2. Chaîne de caractères: Initialisation ➔ Exercice d'application:

Lesquelles des chaînes suivantes sont initialisées correctement ?

- 1) char a[] = "un\ndeux\ntrois\n";
- 2) char b[12] = "un deux trois";
- 3) char c[] = 'abcdefg';
- 4) char d[10] = 'x';
- 5) char e[5] = "cinq";
- 6) char f[] = "Cette " "phrase" "est coupée";
- 7) char g[2] = {'a', '\0'};
- 8) char h[4] = {'a', 'b', 'c'};
- 9) char i[4] = "'o'";

7. Tableaux, Chaînes de caractères et pointeurs.

7.2. Chaîne de caractères: Initialisation ➔ Exercice d'application

1) char a[] = "un\ndeux\ntrois\n"; Déclaration correcte; Espace: 15o
2) char b[12] = "un deux trois"; Déclaration incorrecte;

La chaîne d'initialisation dépasse le bloc de mémoire réservé. Espace:14o

- Correction: char b**[14]** = "un deux trois";
- Ou mieux: char b**[]** = "un deux trois";

3) char c[] = 'abcdefg'; Déclaration incorrecte:;

Les symboles ' ' encadrent des caractères; pour initialiser avec une chaîne de caractères, il faut utiliser les guillemets. Espace: 8 octets

- Correction: char c[] = **"**abcdefg**"**;

4) char d[10] = 'x'; Déclaration incorrecte;

Il faut utiliser une liste de caractères ou une chaîne pour l'initialisation. Espace: 2 o

- Correction: char d[10] = **{'x', '\0'}** ou mieux: char d[10] = **"x"**;

7. Tableaux, Chaînes de caractères et pointeurs.

7.2. Chaîne de caractères: Initialisation ➔ Exercice d'application

5) char e[5] = "cinq"; Déclaration correcte. Espace: 5 octets
6) char f[] = "Cette " "phrase" "est coupée"; Déclaration correcte
Espace: 23 octets
7) char g[2] = {'a', '\0'}; Déclaration correcte Espace: 2 octets
8) char h[4] = {'a', 'b', 'c'}; Déclaration incorrecte:
Dans une liste de caractères, il faut aussi indiquer le symbole de fin de chaîne. Espace: 4 octets

- Correction: char h[4] = {'a', 'b', 'c', **'\0'**};

9) char i[4] = "'o'"; Déclaration correcte,
mais d'une chaîne contenant les caractères '\', '**o**', '\' et '**\0**'.
Espace: 4 octets

7. Tableaux, Chaînes de caractères et pointeurs.

7.2. Chaîne de caractères: Accès aux éléments d'une chaîne

- Une chaîne de caractères est une variable comme une autre pour un programme: on y accède en l'appelant par son nom de variable.

- Une chaîne est un tableau de caractères: pour accéder à ses éléments on suit la logique d'un tableau.

Exemple:

char A[6] = "Hello";

→ A[0] contient 'H', A[1] contient 'e' … A[5] contient '\0'.

7. Tableaux, Chaînes de caractères et pointeurs.

7.2. Chaîne de caractères: Précédence alphabétique et lexicographique

- Une chaîne de caractères est une variable : on peut utiliser des opérations logiques et mathématiques.

- La précédence des caractères dans l'alphabet d'une machine est dépendante du code de caractères utilisé. Pour le code ASCII, nous pouvons constater l'ordre suivant:

  . . . ,0,1,2, ... ,9, . . . ,A,B,C, ... ,Z, . . . ,a,b,c, ... ,z, . . .

- Les symboles spéciaux (' ,+ ,- ,/ ,{ ,] , ...) et les lettres accentuées (é ,è ,à ,û , ...) se trouvent répartis autour des trois grands groupes de caractères (chiffres, majuscules, minuscules). Leur précédence ne correspond à aucune règle d'ordre spécifique.

7. Tableaux, Chaînes de caractères et pointeurs.

7.2. Chaîne de caractères: Précédence alphabétique et lexicographique

- Précédence alphabétique des caractères induit une relation de précédence 'est inférieur à' sur l'ensemble des caractères. Ainsi, on peut dire que '0' est inférieur à 'Z‘ et noter '0' < 'Z’ , Ceci est possible car dans l'alphabet de la machine, le code du caractère '0' (ASCII: 48) est inférieur au code du caractère 'Z' (ASCII: 90).

- Exemples:
  - "ABC" précède "BCD" car 'A'<'B'
  - "ABC" précède "B" car 'A'<'B'
  - "Abc" précède "abc" car 'A'<'a'
  - "ab" précède "abcd" car "" précède "cd"
  - " ab" précède "ab" car ' '<'a'

7. Tableaux, Chaînes de caractères et pointeurs.

7.2. Chaîne de caractères: Tests logiques

- En tenant compte de l'ordre alphabétique des caractères, on peut contrôler le type du caractère (chiffre, majuscule, minuscule).

- Exemples:
  - if (C>='0' && C<='9') printf("Chiffre\n", C);
  - if (C>='A' && C<='Z') printf("Majuscule\n", C);
  - if (C>='a' && C<='z') printf("Minuscule\n", C);

- Il est facile, de convertir des lettres majuscules dans des minuscules:
  - if (C>='A' && C<='Z') C = C-'A'+'a';

ou vice-versa:

  - if (C>='a' && C<='z') C = C-'a'+'A';

7. Tableaux, Chaînes de caractères et pointeurs.

7.2. Chaîne de caractères: Tableaux de chaînes

➔ Une chaîne de caractères est un tableau à 1 dimension de caractères.

- On peut également définir des tableaux à plusieurs dimensions qui peuvent contenir des mots: char JOUR[7][9] = {“lundi” , ”mardi” , ”mercredi” , ”jeudi” , ”vendredi”,”samedi”,”dimanche”}; et on peut accéder à ces mots en utilisant la syntaxe suivante:

int I=0; printf(“Aujourd’hui nous sommes %s”,JOUR[I]);

qui affichera “Aujourd’hui nous sommes lundi”.

- pour accéder à une lettre dans un mot: JOUR[I][j] avec %c pour l’affichage.

7. Tableaux, Chaînes de caractères et pointeurs.

7.2. Chaîne de caractères: Fonctions de bibliothèque

- Les bibliothèques de fonctions de C contiennent une série de fonctions spéciales pour le traitement de chaînes de caractères.
- Les fonctions décrites dans ce chapitre sont portables conformément au standard ANSI-C.
  - **Les fonctions de <stdio.h>**
  - **Les fonctions de <string.h>**
  - **Les fonctions de <stdlib.h>**
  - **Les fonctions de <ctype.h>**

7. Tableaux, Chaînes de caractères et pointeurs.

7.2. Chaîne de caractères: Fonctions de bibliothèque

➢ Les fonctions de <stdio.h>

❑ **scanf, printf** en utilisant %s dans le format

attention: scanf prend une adresse en argument (&x),
une chaîne de caractères étant un tableau (ie, l'adresse du premier élément), il n'y a pas de &.

Exemple :

```
char LIEU[25];
 int JOUR, MOIS, ANNEE;
printf("Entrez lieu et date de naissance : \n");
scanf("%s %d %d %d", LIEU, &JOUR, &MOIS, &ANNEE);.
```

7. Tableaux, Chaînes de caractères et pointeurs.

7.2. Chaîne de caractères: Fonctions de bibliothèque

➢ Les fonctions de <stdio.h>

- la bibliothèque **<stdio>** nous offre des fonctions qui effectuent l'entrée et la sortie des données. A côté des fonctions **printf** et **scanf** que nous connaissons déjà, nous y trouvons les deux fonctions **puts** et **gets,** spécialement conçues pour l'écriture et la lecture de chaînes de caractères.

- **scanf, printf** en utilisant %s dans le format.

7. Tableaux, Chaînes de caractères et pointeurs.

7.2. Chaîne de caractères: Fonctions de bibliothèque

➢ Les fonctions de <stdio.h>

❑ **Affichage➔** Syntaxe: **puts(<Chaîne>)**

➢ **puts** écrit la chaîne de caractères désignée par <Chaîne> et provoque un retour à la ligne.

➢ En pratique, puts(TXT); est équivalent à printf("%s\n",TXT);

Exemples:

✓ **char TEXTE[] = "Voici une première ligne.";**
puts(TEXTE);

✓ **puts("Voici une deuxième ligne.");**

7. Tableaux, Chaînes de caractères et pointeurs.

7.2. Chaîne de caractères: Fonctions de bibliothèque

➢ Les fonctions de <stdio.h>

❑ **Lecture ➔** Syntaxe: **gets(<Chaîne>)**

➢ **gets** est idéal pour lire une ou plusieurs lignes de texte (p.ex. des phrases) terminées par un retour à la ligne.

➢ gets(TXT); lit une ligne jusqu'au retour chariot et remplace le '\n' par '\0' dans l'affectation de la chaîne.

Exemple:

```
int Max = 100;
char LIGNE[Max];
gets(LIGNE);
```

7. Tableaux, Chaînes de caractères et pointeurs.

7.2. Chaîne de caractères: Fonctions de bibliothèque

➢ Les fonctions de <string .h>

- **strlen(<s>)** fournit la longueur de la chaîne sans compter le '\0' final
- **strcpy(<s>, <t>)** copie <t> vers <s>
- **strcat(<s>, <t>)** ajoute <t> à la fin de <s>
- **strcmp(<s>, <t>)** compare <s> et <t> lexicographiquement et fournit un résultat:
  - ➢ négatif si <s> précède <t>
  - ➢ zéro si <s> est égal à <t>
  - ➢ positif si <s> suit <t>
- **strncpy(<s>, <t>, <n>)** copie au plus <n> caractères de <t> vers <s>
- **strncat(<s>, <t>, <n>)** ajoute au plus <n> caractères de <t> à la fin de <s>

7. Tableaux, Chaînes de caractères et pointeurs.

7.2. Chaîne de caractères: Fonctions de bibliothèque

➢ Les fonctions de <string .h>

- La nature de tableau d'une chaîne de caractères (ie, l'adresse en mémoire du premier élément) interdit des affections du type

  A= "hello" en dehors de la phase d'initialisation.

  - ➢ char A[ ]="Hello"; est correct mais
  - ➢ char A[6]; A= "Hello"; ne l'est pas.

- Il faut bien copier la chaîne caractère par caractère ou utiliser la fonction strcpy respectivement strncpy: **strcpy(A, "Hello");**

7. Tableaux, Chaînes de caractères et pointeurs.

7.2. Chaîne de caractères: Fonctions de bibliothèque

➢ Les fonctions de <stdlib.h>

conversion chaîne -> nombre

- **atoi(<s>)** retourne la valeur numérique représentée par <s> comme int
- **atol(<s>)** retourne la valeur numérique représentée par <s> comme long
- **atof(<s>)** retourne la valeur numérique représentée par <s> comme double (!)

7. Tableaux, Chaînes de caractères et pointeurs.

7.2. Chaîne de caractères: Fonctions de bibliothèque

➔ **Les fonctions de <ctype.h>** servent à classifier et à convertir des caractères

- Les fonctions de **classification** suivantes fournissent un résultat du type **int** différent de zéro, si la condition respective est remplie, sinon zéro.

- **isupper(<c>):** si <c> est une majuscule **('A'...'Z')**
- **islower(<c>):**si <c> est une minuscule **('a'...'z')**
- **isdigit(<c>):**si <c> est un chiffre décimal **('0'...'9')**
- **isalpha(<c>):**si **islower(<c>)** ou **isupper(<c>)**
- **isalnum(<c>):** si **isalpha(<c>)** ou **isdigit(<c>)**

7. Tableaux, Chaînes de caractères et pointeurs.

7.2. Chaîne de caractères: Fonctions de bibliothèque

➔ Les fonctions de <ctype.h>

Les fonctions de **classification:**

- **isxdigit(<c>):** si <c> est un chiffre hexadécimal
  (**'0'...'9'** ou **'A'...'F'** ou **'a'...'f'**)

- **isspace(<c>):** si <c> est un signe d'espacement
  (**' '**, **'\t'**, **'\n'**, **'\r'**, **'\f'**)

7. Tableaux, Chaînes de caractères et pointeurs.

7.2. Chaîne de caractères: Fonctions de bibliothèque

➔ Les fonctions de <ctype.h>

- Les fonctions de ***conversion*** suivantes fournissent une valeur du type **int** qui peut être représentée comme caractère; la valeur originale de <c> reste inchangée:

- **tolower(<c>):** retourne <c> converti en minuscule si <c> est une majuscule.
- **toupper(<c>):** retourne <c> converti en majuscule si <c> est une minuscule.

Les séquences d'échappement

\n	nouvelle ligne
\t	tabulation horizontale
\v	tabulation verticale (descendre d'une ligne)
\a	sonnerie
\b	curseur arrière
\r	retour au début de ligne
\f	saut de page
\\	trait oblique (back-slash)
\?	point d'interrogation
\'	apostrophe
\"	guillemets
\0	fin de chaîne