Chapitre 6: Programmation modulaire

6. Programmation modulaire: les fonctions

6.1. Introduction

- Un programme dépassant une ou deux pages est difficile 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**, c-à-d 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**(tout est fonction)

- 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 code 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: int, double..void,
- **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() {
   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);`
`  affichSom(2.5, 7.3);`
`}`
	`int Som(int x, int y )`
	`{`
	`  return (x+y);`
	`}`
	
	`void affichSom(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.
- Les variables déclarées dans une fonction sont locales à cette fonction
- Une fonction possède un et un seul point d'entrée, mais éventuellement plusieurs points de sortie (à l'aide du mot **return**).
- L'imbrication de fonctions n'est pas autorisée: une fonction ne peut pas être déclarée à l'intérieur d'une autre fonction. Par contre, une fonction peut **appeler** une autre fonction. Cette dernière doit être déclarée **avant** celle qui l'appelle.

6. Programmation modulaire: les fonctions

6.5. Déclaration des fonctions

- On rencontre le nom des fonctions dans 3 cas :
- Déclaration: le type de la fonction et de ses arguments

  =>1 seule fois
- Définition: codage de la fonction

  =>1 seule fois
- Appels(= utilisations) de la fonction

  => n fois

6. Programmation modulaire: les fonctions

6.5. Déclaration des fonctions: Exemple

```
 #include<stdio.h>
float ValeurAbsolue(float);  /* déclaration : 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; // return (a<0? -a: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 = 5;
int f(int);
int g(int);
int main( )
{ printf("x = %d\t", x);
 int x = 10; 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=5
x=10
f(10)=19
g(10) = 50

6. Programmation modulaire: les fonctions

6.6. Variables locales et globales: Exemple

```
#include<stdio.h>
void f(void); /* ou  void f()*/
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.

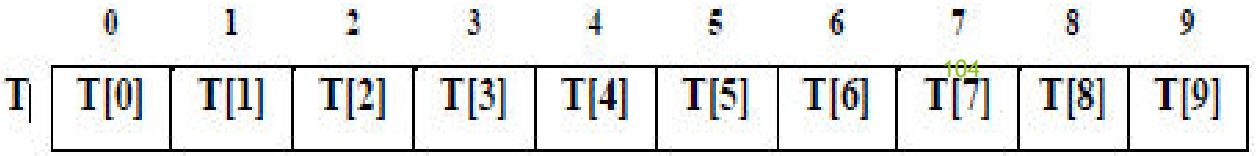
A. **Tableaux à une dimension (vecteurs)**
B. **Tableaux de deux dimensions (matrices)**

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
- int T[10];



7. Tableaux, Chaînes de caractères et pointeurs.

7.1. Les Tableaux

A. Tableaux à une dimension :

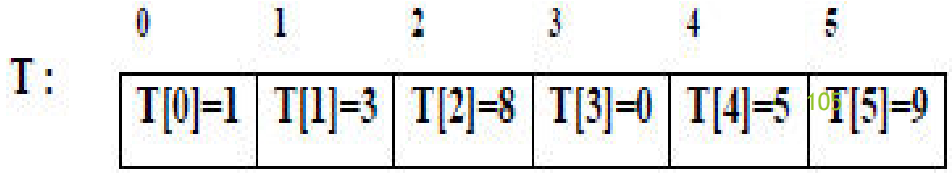
- Déclaration sans initialisation:

<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};
```



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}; // taille =9
```

- 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]; int i;
for (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

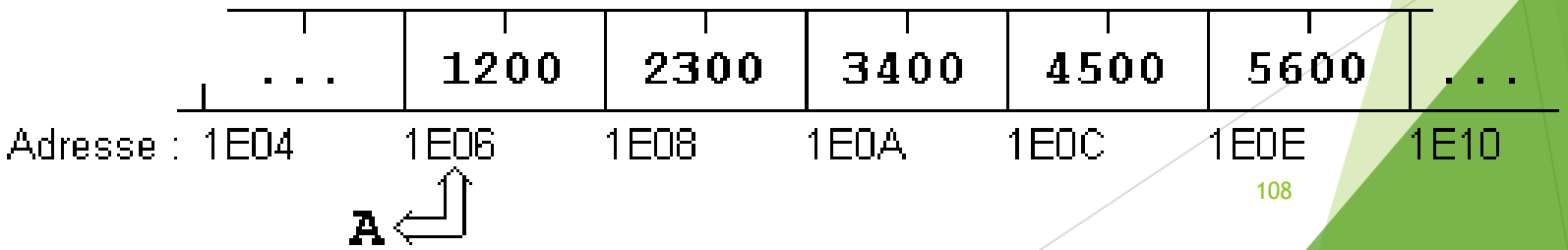
A. Tableaux à une dimension : taille

- 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

- 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;
 }
```

- **Calculer de la somme et le produit des composants d'un tableau saisie au clavier**

```
int main()
  {int A[5];
  int i;          /* Compteur */
  for (i=0; i<5; i++) //saisie
    scanf("%d", &A[i]);
  int s=0;
  for (i=0; i<5; i++) //somme
         s= s + A[i];
   printf(" La somme = %d", s);
return 0;
 }
```

```
int main()
  {int A[5];
  int i;          /* Compteur */
  for (i=0; i<5; i++) //saisie
    scanf("%d", &A[i]);
  int p=1;
  for (i=0; i<5; i++) //somme
         p= p * A[i];
   printf(" La produit = %d", p);
return 0;
 }
```

7. Tableaux, Chaînes de caractères et pointeurs.

7.1. Les Tableaux

A. Tableaux à une dimension : Recherche

- Recherche d'un élément dans un tableau

```
int rechSequenti(int t[], int quoi, int max){
   int i , trouve= 0;//zéro
   i= 0;
   while( (i < max) && (trouve == 0)){
      if (t[i] == quoi) trouve = 1;
            else  i++;
   }
   if (i == max)    return -1;
   return i;
}
```

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

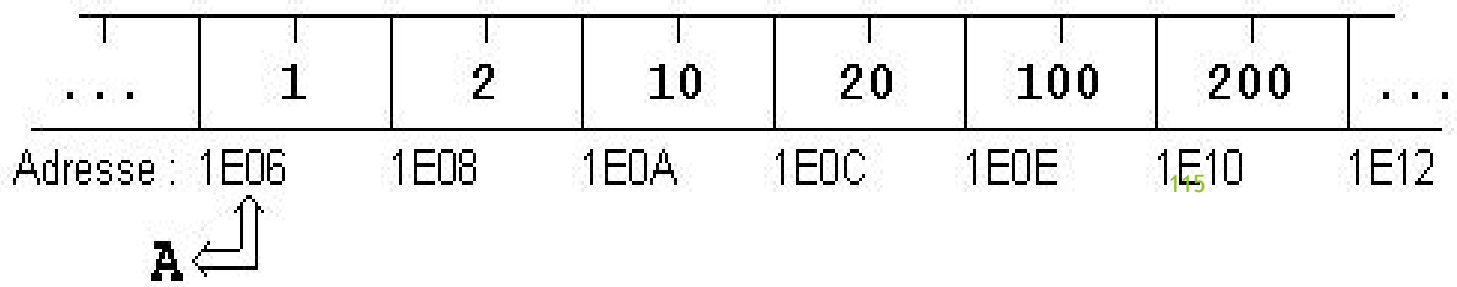
```
<TypeSimple> <NomTabl>[<DimLigne>][<DimCol>];
```

- **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*4 = 120 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**

```
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++)
            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,i; /* 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 ch[ ] = “Hello”;
  - char ch[6] = “Hello”;
  - char ch[ ] = {‘H’,’e’,’l’,’l’,’o’,’\0’};
  - char ch[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 ch[5] = “Hello”; donnera une erreur à l’exécution
- char ch[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:

```
int C ; for(C=0; C<255; C++){
    if (C>='0' && C<='9') printf("Chiffre %c\n", C);
    if (C>='A' && C<='Z') printf("Majuscule %c\n ", C);
    if (C>='a' && C<='z') printf("Minuscule %c\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];//LIEU ==&LIEU[0]
 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:

```
const int Max = 100;
char LIGNE[Max];
gets(LIGNE);// scanf(« %s\n »,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.