Chapitre II

Tableaux, Pointeurs Fonctions et Structures en C++

II. 1. Introduction

Dans ce chapitre, nous allons étudier les tableaux les pointeurs et les fonctions en C++. La première partie de ce chapitre sera consacrée aux différents types des tableaux statiques et dynamiques. Dans la deuxième partie, nous allons étudier les pointeurs et leur avantage d'utilisation avec les tableaux et pour l'allocation dynamique de la mémoire. Des exemples avec solutions seront présentés après chaque nouvelle notion étudiée pour faciliter la compréhension et acquérir rapidement les compétences visée.

II. 2. Tableaux statiques unidimensionnels

Un tableau uni-dimensionnel est une suite limitée de case mémoire contenant des éléments simples et de même type (int, char, float, double...). Ces éléments sont appelés composants du tableau et leur nombre représente la taille du tableau.

II.2.1 Syntaxe de la déclaration d'un tableau statique unidimensionnel

La syntaxe de déclaration des tableaux unidimensionnels est la suivante :

type nom $[taille] = \{valeurs\ initiales\}$;

Exemple II.1

```
int age [5] = {15, 20, 70, 11,18} ; /* tableau appelé 'age'de type 'int'et de taille 5. Les cinq
                                       composantes sont initialisées respectivement par 15, 20…18.*/
float moy [5]; //tableau appelé 'moy'de type ' float', de taille 5, et des composantes non initialisées.
```

Pour accéder aux composantes du tableau, il suffit d'indiquer le nom du tableau suivi de l'indice de la composante, par exemple :

```
age[0] ;   // pour accéder à la première composantes (15).
age[4] ;   // pour accéder à la dernière composantes (18).
```

Remarque importante :

1. Le nom d'un tableau représente l'adresse du premier élément du tableau.
2. l'accès à la première composante du tableau se fait par nom [0] et l'accès à la dernière composante se fait par nom [taille -1].

3. Le tableau occupe un espace mémoire égale : taille du tableau*nombre d'octets constituant chaque composante. Ainsi le tableau '*age*' occupe 10 octets et le tableau '*moy*' occupe 20 octets.

II.2.2 Parcourir un tableau unidimensionnel

L'un des avantages les plus importants des tableaux, c'est qu'on peut utiliser les boucles pour lire, afficher, initialiser ou modifier chaque composante séparément des autres.

Exemple II.2

Le programme suivant affiche les composantes d'un tableau en utilisant la boucle for.

```
#include <iostream>
using namespace std;

int main()
{
 int const taille (5);//taille du tableau,il est très recommandé
                      //de la déclarer comme constante

 int age [taille]={15,20,70,11,18}; /* Déclaration d'un tableau
 /    unidimensionnel de type int contient cinq valeurs entières*/
 for(int i=0;i<taille;i++)
 {
  cout<<"age["<<i<<"]=\t"<<age[i]<<endl;/* affichage des composants
  du tableau 'age' l'un après l'autre*/
 }
return 0;
}
```

L'exécution du programme ci-dessus donne le résultat suivant :

```
age[0]= 15
age[1]= 20
age[2]= 70
age[3]= 11
age[4]= 18

Process returned 0 (0x0)   execution time : 0.093 s
Press any key to continue.
```

Exemple II.3

Le programme suivant lit 05 valeurs entières et les stocker dans un tableau.

```
#include <iostream>
using namespace std;

int main()
{
 int const taille (5);//taille du tableau

 int tab [taille];// Déclaration d'un tableau unidimensionnel
                  //  de cinq (05) composants (éléments)
 int i(0);//indice
 int size_tab (taille);/*Stocker la taille de tab dans une variable
                     pour qu'on puisse l'incémenter ou la décrémenter*/
 do
 {
  cout<<" Entrer la valeur du tab["<<i<<"]= ?\t ";
  cin >> tab[i];
  i++;
  size_tab--;
  }
```

```
  while (size_tab>0);
return 0;
}
```

Résultat d'exécution:

```
"C:\Users\Malik\Desktop\Programmation OrientÚe Objet\Exemples_chapitre2\ex02\bin\Debug\ex02.exe"
Entrer la valeur du tab[0]= ?   12
Entrer la valeur du tab[1]= ?   24
Entrer la valeur du tab[2]= ?   85
Entrer la valeur du tab[3]= ?   32
Entrer la valeur du tab[4]= ?   47

Process returned 0 (0x0)   execution time : 12.216 s
Press any key to continue.
```

Exercice II.1

Ecrire un programme qui remplit les éléments (composants) d'un tableau de taille sept (04) par des valeurs entrées au clavier et affiche ensuite:

- Le tableau rempli ;
- La somme des composantes du tableau ;
- L'indice et la valeur de la composante la plus grande ;
- L'indice et la valeur de la composante la plus petite ;

Solution d'Exercice II.1

```
#include <iostream>
using namespace std;
int main ()
{
 int const taille (4);//taille du tableau
 int max_tab(0),idice_max(0), min_tab(0),idice_min(0);
 long int som_Tab(0);// variable à utiliser pour stocker
                     // la somme des composants du tableau

 float Tab[taille]; //Déclaration du tableau

 //Remplissage du tableau
 cout<<"Entrer la valeur du Tab[0]= ?\t";
 cin >> Tab[0];
 max_tab=Tab[0]; idice_max=0;
 min_tab=Tab[0]; idice_min=0;
 som_Tab+=Tab[0];
 for(int i=1;i<taille;i++)
 {
  cout<<"Entrer la valeur du Tab["<<i<<"]= ?\t";
  cin >> Tab[i];
  som_Tab +=Tab[i];//Calcul la somme des composants du tableau
  //chercher la composante dont la valeur la plus grande et son indice
  if (Tab[i]>max_tab)
  {
   max_tab=Tab[i];
   idice_max=i;
  }
  //chercher la composante dont la valeur la plus petite et son indice
  if (Tab[i]<min_tab)
  {
   min_tab=Tab[i];
   idice_min=i;
```

```
34      }
35    }
36     //Affichage du tbleau
37     for(int i=0;i<taille;i++)
38     {
39      cout<<"Tab["<<i<<"]=\t"<<Tab[i]<<endl;
40     }
41     cout<<"la somme des composantes du tableau= "<<som_Tab<<endl;
42     cout<<"la valeur la plus grande= Tab["<<idice_max<<"]= "<<max_tab<<endl;
43     cout<<"la valeur la plus faible= Tab["<<idice_min<<"]= "<<min_tab<<endl;
44       return 0;
45  }
```

Exemple d'exécution:

```
Entrer la valeur du Tab[0]= ?   14
Entrer la valeur du Tab[1]= ?   -9
Entrer la valeur du Tab[2]= ?   84
Entrer la valeur du Tab[3]= ?   20
Tab[0]= 14
Tab[1]= -9
Tab[2]= 84
Tab[3]= 20
la somme des composantes du tableau= 109
la valeur la plus grande= Tab[2]= 84
la valeur la plus faible= Tab[1]= -9
```

II. 3. Tableaux multidimensionnels

Un tableau multidimensionnel est un tableau de tableau ; c.à.d. un tableau **bidimensionnel** est un tableau unidimensionnel d'un tableau unidimensionnel et un tableau **tridimensionnel** est un tableau unidimensionnel d'un tableau bidimensionnel… etc.

II.3.1 Syntaxe de la déclaration des tableaux multidimensionnels

La syntaxe de la déclaration des tableaux multidimensionnels est similaire à celle des tableaux unidimensionnel sauf qu'on ajout la taille pour chaque dimension.

Exemple :

```
float tab1[10][5] ; // déclaration d'un tableau bidimensionnel (10 x 5= 50 composants)
double tab2[10][5][7] ; // déclaration d'un tableau tridimensionnel (10x5x7= 350 composants)
```

Pour accéder aux composantes du tableau multidimensionnel, il suffit d'indiquer le nom du tableau suivi des indices du composant. Par exemple:

```
tab1[1][2] ;        // Pour accéder au composant identifié par les indices (1,2)
tab2[0][1][3] ;   // Pour accéder au composant identifié par les indices (0,1,3).
tab2[1][5][0]=2018 ;// Affecter la valeur 2018 au composant identifié par les indices (1,5,0)
```

Remarque:

Le tableau de n dimensions occupe un espace mémoire égale : taille de la 1ère dimension * taille de la 2ème dimension*….. *taille de la nième dimension *nombre d'octets constituant chaque composante. Ainsi, si la variable du type float occupe 4 octets et le type double occupe 8 octets le tableau *tab1* occupe 200 octets et le tableau *tab2* occupe 2800 octets.

II.3.2 Parcourir un tableau multidimensionnel

Pour parcourir le contenu d'un tableau de *n* dimensions, on a besoin à *n* indices et une boucle imbriquée *n* fois.

Exemple II.4

Le programme suivant affiche les composantes d'un tableau à deux dimensions ainsi que leur somme.

```
#include <iostream>
using namespace std;

int main()
{

    long int somme(0);
//Déclaration et initialisation d'un tableau à deux dimensions
    int Tabl[4][4]={
                    {1,2,3,4},    // première ligne
                    {2,4,6,8},    // deuxième ligne
                    {4,8,12,18}, // troisième ligne
                    {8,16,24,36} // quatrième ligne
                    };
//Affichage du tableau
for(int i=0; i<4;i++)//pour parcourir les 4 lignes
{
    for(int j=0; j<4;j++)//pour parcourir les 4 colonnes
    {
     cout <<Tabl[i][j]<<"\t";
     somme+=Tabl[i][j];//calcul de la somme des composantes du tableau
    }
    cout <<"\n";
}
    cout <<"la somme de toutes les composante du tableau= "<<somme<<endl;
    return 0;
}
```

L'exécution du programme ci-dessus donne le résultat suivant :

```
"C:\Users\Malik\Desktop\Programmation OrientÚe Objet\Exemple_Prog\Exemples_chapitre2\Tamblau_2dimensions'
1       2       3       4
2       4       6       8
4       8       12      18
8       16      24      36
la somme de toutes les composante du tableau= 156
```

Exemple II.5

Le programme suivant :

- rempli d'un tableau bidimensionnel par des valeurs entrées au clavier ;
- affiche le tableau et la somme de chaque ligne.

```
#include <iostream>
using namespace std;

int main()
{
 long int somme(0);
 int const L(3),C(3);//dimensions du tableau
```

```
int  i(0),j(0);//indices
int Tab[L][C];//Déclaration du tableau bidimensionnel

//Lecture des composantes du tableau 'Tab'
for(int i=0; i<L; i++)//pour parcourir les 4 lignes
{
 for(int j=0; j<C;j++)//pour parcourir les 4 colonne
 {
  cout <<"Tab["<<i<<"]["<<j<<"]= ?"<<"\t";
  cin>> Tab[i][j];
 }
}

//Affichage du tableau la somme de chaque ligne
for(i=0; i<L; i++)
{
 somme=0;
 for(j=0; j<C;j++)
 {
  cout <<Tab[i][j]<<"\t";
  somme +=Tab[i][j]; //calcul de la somme de ligne i
 }
 cout << " La somme des composantes de la ligne " <<i+1<< "=\t"<< somme<
}
return 0;
}
```

Exemple d'exécution:

```
Tab[0][0]= ?    14
Tab[0][1]= ?    2
Tab[0][2]= ?    1
Tab[1][0]= ?    12
Tab[1][1]= ?    -2
Tab[1][2]= ?    4
Tab[2][0]= ?    0
Tab[2][1]= ?    1
Tab[2][2]= ?    15
14      2       1       La somme des composantes de la ligne 1=        17
12      -2      4       La somme des composantes de la ligne 2=        14
0       1       15      La somme des composantes de la ligne 3=        16
```

II. 4. Tableaux dynamiques

Un tableau dynamique est une suite variable de case mémoire (taille variable), i.e. on peut varier la taille du tableau en ajoutant ou en supprimant des cases mémoires.

II.4.1 Syntaxe de la déclaration d'un tableau dynamique

La déclaration des tableaux dynamiques est un peu différente à celle des tableaux statiques. Par exemple, on doit écrire tous d'abord le mot '**vector'** suivi ensuite par le **type entre deux chevrons**, ensuite **le nom du tableau** et enfin la **taille du tableau <u>entre deux parenthèses</u>**

Syntaxe: **vector** < **type** > **Nom**(**Taille**)

<u>Exemple II.6</u>

vector <int> age (5) ; /* Déclaration d'un tableau dynamique unidimensionnel de type *int* de taille égale à 5. Le tableau 'age' est non initialisé*/

vector <int> annee (5, 2022) ; /* annee est un tableau dynamique dont toutes ses composantes sont

initialisées par 2019, i.e. annee contient {2022, 2022, 2022, 2022, 2022} */

vector <string> mois (3, " *Novembre* ") ; /* mois est un tableau dynamique de type *string* dont toutes ses composantes sont initialisées par la chaine de caractère Décembre, i.e. mois contient : { *Novembre*, *Novembre*, *Novembre*,} */

II.4.2 Accès aux composantes d'un tableau dynamique

L'accès aux composantes d'un tableau dynamique est identique à celui du tableau statique.

Exemple II.7

Le programme suivant remplit les composantes d'un tableau dynamique ensuite les afficher l'une après l'autre.

```
#include <iostream>
#include<vector>// Obligatoire pour utiliser les tableaux dynamiques
using namespace std;

int main()
{
    vector<int> age(5);//Déclaration d'un tableau dynamique de type int, de
                       // taille égale 5 et non initialisé.
    age[0]=15;// Affectation de la première composante du tableau
    age[1]=20;// Affectation de la deuxième composante du tableau
    age[2]=70;
    age[3]=11;
    age[4]=18;

    for (int i=0;i<5;i++)//Affichage du tableau
    {
        cout<<"age["<<i<<"]= "<<age[i]<<endl;
    }
    return 0;
}
```

Remarques importantes :

- Pour utiliser les tableaux dynamiques, on doit ajouter la bibliothèque #include <vector>.
- Contrairement aux tableaux fixes :
  1. la déclaration des tableaux dynamiques commence par le mot *vector* ensuite le type, le nom et la taille du tableau.
  2. Le type de tableau doit être entre deux chevrons '< >'.
  3. La taille du tableau doit être entre deux parenthèses (n'est pas entre crochets).

II.4.3 Incrémenter et décrémenter la taille d'un tableau dynamique

On peut changer la taille d'un tableau dynamique en ajoutant des nouvelles composantes via l'instruction *push_back( )*, ou en supprimant des composantes en utilisant l'instruction *pop_back( )*. La syntaxe de l'utilisation de ces instructions est la suivante:

NomDuTableaux**.** push_back (val) // pour ajoute une composante initialisée par la valeur '*val'*.

NomDuTableaux**.** pop_back() // pour supprimer **la dernière** composante du tableau.

Exemple II.8

Le programme suivant déclare un tableau dynamique '*age*' de taille 5 et le remplit par les valeurs {15, 20, 70, 11, 18}. Le programme ajoute 03 composantes au tableau initialisées respectivement par {05, 10, 99} et ensuite affiche le tableau.

```
#include <iostream>
#include<vector>// Obligatoire pour utiliser les tableaux dynamiques
using namespace std;

int main()
{
    vector<int> age(5);//Déclaration d'un tableau dynamique de type int,
                       // detaille égale 5.
// Initialisation du tablaux age par les valeur {15, 20, 70, 11, 18}
    age[0]=15;
    age[1]=20;
    age[2]=70;
    age[3]=11;
    age[4]=18;
    age.push_back(05);//ajoute une composante au tableau age initlalisé par 5
    age.push_back(10);//ajoute une composante au tableau age initlalisé par 10
    age.push_back(99);//ajoute une composante au tableau age initlalisé par 99

 //Le tablau age contien alors les valeurs {15,20,70,11,18,05,10,99}

 int taille= age.size();// Calcul la nouvelle taille du tableau

    for (int i=0;i<taille;i++)//Affichage du tableau age
    {
        cout<<"age["<<i<<"]= "<<age[i]<<endl;
    }
    return 0;
}
```

Remarque importante :
Le langage C++ ne contrôle pas les dépassements des indices des tableaux comme certains langages (Pascal par exemple). Le programmeur doit assurer que les valeurs des indices ne dépassent pas leurs limites.

II. 5. Pointeur en C++

Un pointeur est une variable qui contient l'adresse d'une autre variable et qui peut pointer (positionné) sur différentes adresses.

II.5.1 Syntaxe de déclaration d'un pointeur

type *NomPointeur (0) ; // Déclaration d'un pointeur qui peut recevoir des adresses des variables
// du type <type>.

Exemple:

int *age (0); // Un pointeur qui peut contenir l'adresse d'un nombre entier ;

string *expression (0); // Un pointeur qui peut contenir l'adresse d'une chaîne de caractères

vector<int> *Tab (0); // Un pointeur qui peut contenir l'adresse d'un tableau dynamique de
// nombres entier.

Remarque **: '0' signifie** que le pointeur ne contient aucune adresse.

Pour utiliser les pointeurs, nous avons besoin à :

- L'opérateur **'adresse' (&)** pour obtenir l'adresse de la variable à laquelle le pointeur est appliqué (avec les tableaux on peut utiliser directement le nom du tableau).
- Opérateur **'contenu de' (*)** pour accéder au contenu d'une adresse (déréférencement).

Exemple II.9

Le programme suivant montre comment utiliser un pointeur pour accéder et changer le contenu d'une variable.

```
#include <iostream>

using namespace std;
int main()
{
    int annee (2018) ;// variable de type int contient la valeur 2018
    int *p(0); // Un pointeur ne contient aucune adresse entier
    p = &annee; // affecte l'adresse de la variable N à le pointeur 'p'
    cout <<  *p<< endl;   //  donne 2018
    (*p)++ ; // le contenu de la variable annee (référencé par *p)
             // est incrémenté par 1.
    cout <<  annee<< endl;   //  vaut 2019.
    annee ++ ;
    cout <<  *p<< endl;   //  vaut 2020.
}
```

II.5.2 Allocation dynamique d'un espace mémoire

Pour réserver une case mémoire par un pointeur, il faut utiliser l'opérateur '**new'**. Cette dernière demande une case mémoire à l'ordinateur et renvoie un pointeur pointant vers cette case.

Exemple II.10

Le programme suivant montre comment utiliser un pointeur pour l'allocation dynamique de la mémoire.

```
#include <iostream>
using namespace std;
int main()
{
    int *p (0);
    p= new int;// demande une case mémoire pouvant stocker un entier
               // (l'adresse de cette case est stockée dans le pointeur p)

    *p = 2018;      // affecter la case adressée par p par la valeur 2018
    *(p+1) =2019 ; // affecter la case adressée par p+1 par la valeur 2019
    *(p+2) =1954 ; // affecter la case adressée par p+2 par la valeur 1954
}
```

Une fois que l'on n'a plus besoin de la case mémoire, on peut la libérer. Cela se fait *via* l'opérateur '**delete'**.

Exemple II.11

Dans le programme suivant, on va réserver une case mémoire (allocation dynamique) et ensuite la libérer en utilisant l'instruction 'delete'.

```cpp
#include <iostream>
using namespace std;
int main()
{
    int *p(0);
    p = new int;
    delete p;  // libérer la case mémoire adressé par le pointeur p
    p= 0;  // le pointeur ne pointe plus à aucune case mémoire

}
```

Remarque importante :
il est très recommandé d'initialiser le pointeur (p=0) après la libération de la case mémoire

II.5.3 Pointeurs et tableaux

Comme nous l'avons déjà étudié, le nom d'un tableau représente l'adresse de la première composante (c.à.d. tableau = &tableau[0]). Alors un pointeur appliqué sur la première composante peut accéder à toutes les composantes du tableau.

Exemple II.12

```cpp
#include <iostream>
using namespace std;
int main()
{
    int  age[6]={15, 20, 30, 80, 50, 18}; // tableau de type int.
    int A(0), B(0), C(0) ;  // Variables de type int
    int *p (0);//  déclaration d'un pointeur ne contient aucune adresse
    p = age; // est équivalente à P = &A[0], c.à.d. p contient l'adresse du tableau
    A= *(p+1) ; // A vaut 20
    B= *(p+2) ; // A vaut 30
    C= *(p+5) ;//  A vaut 18
    cout <<"A= "<< A <<endl;
    cout <<"B= "<< B <<endl;
    cout <<"C= "<< C <<endl;
}
```

Remarques:

- Toutes les opérations avec les pointeurs tiennent compte automatiquement du type et de la grandeur des objets pointés.
- Le pointeur peut être affecté par les opérateurs simple ou composé (exemple: p=23A, p+=3, p-=5H, p*=2, …).
- Le pointeur peut être incrémenté et décrémenté par les opérateurs (exemple: p++, p--);
- La soustraction de deux pointeurs (p1-p2) fournit le nombre de composantes comprises entre p1 et p2.
- On peut comparer les pointeurs par <, >, <=, >=, ==, !=.

Exemple II.13

Le programme suivant lit au clavier deux tableaux et les affiche en utilisant seulement les pointeurs.

```cpp
#include <iostream>
using namespace std;
int main()
{ int tail_TabA (0), tail_TabB(0);
 cout << " Entrer la taille du prmeir tableau "<< endl;
 cin >> tail_TabA;
 cout << " Entrer la taille du deuxieme tableau "<< endl;
 cin >> tail_TabB;

 int TabA[tail_TabA],TabB[tail_TabB];//Declaration des tableaux
 int *pA(0),*pB(0);//Declaration des pointeurs

//Saisie du premier tableau en utilisant le pointeur *pA
cout << "La saisie du premier tableau" <<endl;
  for(pA=TabA; pA<(TabA+tail_TabA);pA++)
  {
      cout << " Entrer TabA["<<pA-TabA<<"]= ";
      cin >> *pA;
  }
//Saisie du deuxième tableau en utilisant le pointeur *pB
cout << "La saisie du deuxième tableau" <<endl;
  for(pB=TabB; pB<(TabB+tail_TabB);pB++)
  {
      cout << " Entrer TabB["<<pB-TabB<<"]= ";
      cin >> *pB;
  }
//Affichage du premier tableau en utilisant le pointeur *pA
// sous forme TabA=[.....]
  cout << " TabA=[";
  for(pA=TabA; pA<(TabA+tail_TabA);pA++)
  {
     cout << *pA <<' ';
  }
  cout << "]"<<endl;

//Affichage du deuxième tableau en utilisant le pointeur *pB
// sous forme TabB=[.....]
  cout << " TabB=[";
  for(pB=TabB; pB<(TabB+tail_TabB);pB++)
  {
     cout << *pB <<' ';
  }
  cout << "]"<<endl;
   return 0;
}
```

Exemple II.14

Le programme suivant lit et stocker N valeurs entières en utilisant l'allocation dynamique de la mémoire.

```cpp
#include <iostream>
using namespace std;
int main()
{ int nbr_valeurs (0);
 cout << " Combien de valeurs vous aller introduir ? "<< endl;
 cin >> nbr_valeurs;
  int *pT(0);//Déclaration d'un pointeur
```

```
8     pT= new int;//Demande une case mémoire pouvant stocker un entier
9                 // (l'adresse de cette case est stockée dans le pointeur pT)
10    int *ad_pT (0);
11    ad_pT=pT;    // sauvegarder l'adresse de pT dans ad_pT
12  //Lecture et stockage des valeurs
13    while(pT< ad_pT+nbr_valeurs )
14    {
15       cout<< " Entrer une valeur ";
16       cin >>*pT;
17       pT++;
18    }
19    pT=ad_pT;
20    //Affichage des valeurs introduites
21   cout << " Valeurs introduites=[";
22    while(pT< ad_pT+nbr_valeurs)
23    {
24       cout <<*pT<<' ';
25       pT++;
26    }
27     cout << "]"<<endl;
28  }
```

II. 6. Fonction en C++

Les problèmes complexes nous arrivent parfois à écrire des programmes de centaine ou millième instructions, ainsi nous obtenons des programmes peu compréhensibles et peu structurés. Pour éviter ce problème, on subdivise le programme global en plusieurs sous-programmes (**fonctions**) qui peuvent interagir entre eux pour résoudre un problème donné.

Donc une **fonction** est une sous-programme réalise une tache particulière, elle peut être compilée/testée séparément et réutilisée dans d'autres programmes.

II.6.1 Avantage de la subdivision d'un programme en plusieurs fonctions

- Meilleure lisibilité
- Diminution du risque d'erreurs
- Possibilité de tests sélectifs
- Réutilisation de fonctions déjà existantes
- Favorisation du travail en équipe

II.6.2 Développement des fonctions en C++

Le développement d'une fonction est composé de deux parties :

- Définition de la fonction (ou prototype de la fonction) : on indique le nom, le type de retour (int, char, string…) et les paramètres de la fonction (variables, constantes, pointeurs...) selon la syntaxe suivante :

$$type\_de\_retour\ nom\_de\_la\ fonction(paramètre1,\ paramètre2,\ldots\ paramètreN\ )$$

Exemple II.15

```
4
5   int max2Val(int x, int y)
6   /* la fonction 'max2Val' besoin de deux paramètres et fournit un
7      résultat du type entier (elle retourne un entier)*/
```

Exemple II.16

```
19  char SelectChar (int b)
20  /* la fonction 'SelectChar' besoin d'un seul paramètre et fournit
21  un résultat du type caractère (elle retourne un caractère)*/
```

Implémentation de la fonction (corps de la fonction) : c'est l'ensemble des instructions qui réalisent l'action de la fonction.

Exemple II.17

Le corps (l'implémentation) de la fonction 'max2Val' peut être donnée comme suit:

```
 8  {
 9    if ( x>y)
10      {
11      return x;
12      }
13    else
14      {
15      return y;
16      }
17  }
```

Exemple II.18

Le corps (l'implémentation) de la fonction '*SelectChar*" peut être donnée comme suit:

```
22  {
23   switch (b)
24   {
25      case 1 :
26      {return 'A' ;}break ;
27      case 2 :
28      {return 'B' ;}break ;
29      case 3:
30      {return 'C' ;}break ;
31      default :
32      {return 'x' ;}
33   }
34  }
```

Ainsi le code complet qui déclare les deux fonctions ('max2Val' et 'SelectChar') est le suivant :

```
1   #include <iostream>
2   using namespace std;
3
4   int max2Val(int x, int y)
5   /* la fonction 'max2Val' besoin de deux paramètres et
6   fournit unrésultat du type entier (elle retourne un entier)*/
7   {
8     if ( x>y)
9       {
10      return x;
11      }
12    else
13      {
14      return y;
15      }
16  }
17  char SelectChar (int b)
18  /* la fonction 'SelectChar' besoin d'un seul paramètre et fournit
19  un résultat du type caractère (elle retourne un caractère)*/
20  {
21   switch (b)
```

```
22  {
23      case 1 :
24      {return 'A' ;}break ;
25      case 2 :
26      {return 'B' ;}break ;
27      case 3:
28      {return 'C' ;}break ;
29      default :
30      {return 'x' ;}
31  }
32 }
```

II.6.3 Appeler une fonction

Pour appeler une fonction déjà déclarée on doit spécifier son nom et ses paramètres.

Exemple II.19

Le programme suivant montre comment appeler les fonctions ‘max2Val’ et ‘SelectChar’ dans la fonction principale ‘main’.

```
36  int main()
37  {
38      int a (0);
39      a=max2Val(5,8); //appel de la fonction 'max2Val'
40      cout << a << endl;
41      cout << SelectChar (3);//appel de la fonction 'SelectChar'
42      return 0;
43  }
```

L’exécution du code précédent donne : 8 et C

Remarques importantes :

1. Il est possible de définir une fonction sans paramètres en n’écrivant rien entre les parenthèses de son entête.
2. Il est possible de définir une fonction qui ne renvoie rien en écrivant le mot ‘**void**’ comme un type de retour.

II.6.4 Variables locales d’une fonction

Toutes les variables déclarées au sein du corps de la fonction ne sont pas accessibles au d’autres fonctions, alors elles sont considérées comme des **variables locales**.

II.6.5 Passage par valeur et passage par référence

Comme on a étudié dans la section précédente ; lorsqu’on appelle une fonction on doit initialiser ses paramètres. Si les paramètres sont des variables on dit que la fonction utilise le passage par des valeurs. Si les paramètres sont de type pointeurs ou référence on dit que la fonction utilise le passage par référence (adresse).

Exemple II.20

Dans le programme suivant la fonction ‘Somme’ utilise le passage par valeur.

```cpp
#include <iostream>
using namespace std;
//fonction calcul la somme de deux nombres
double Somme(int x, int y)
{
    return x+y;
}

int main()
{
 double s(0);
 cout << Somme(5, 11)<<endl; //appel de la fonction Somme

return 0;
}
```

Exemple II.21

Dans le programme suivant la fonction 'permute' utilise le passage par référence.

```cpp
#include <iostream>
using namespace std;
//fonction fait la permutation de deux nombres
permute(double *x, double *y)
{
double VarAide(0);
 VarAide=*x;
 *x=*y;
 *y=VarAide;

}

int main()
{
 double a(5), b(10);

  permute(&a,&b);//appel de la fonction Permute
  cout << a<<endl;
  cout << b<<endl;
return 0;
}
```

Exercice II.2

En utilisant le passage par référence implémenter trois fonctions *Lire_Tab* , *Affiche_Tab et Somme_Tab*.

1- La fonction *Lire_Tab* permet de lire la taille et les éléments d'un tableau (elle exige que la taille de tableau ne dépasse pas 100) ;
2- La fonction *Affiche_Tab* permet d'afficher le tableau entré par la fonction *Lire_Tab* ;
3- La fonction *Somme_Tab* permet de retourner la somme des éléments du tableau ;

Solution d'Exercice II.2

```
#include <iostream>

using namespace std;
// Fonction Lire_Tab permet de lir un tableau
 void Lire_Tab (double NomTab [], unsigned int &Taille_Tab )
 {
    while (Taille_Tab<=0 ||Taille_Tab>100 )
    {
    cout << " Entrer la taille du tableau (Max=100) "<<endl;
    cin>> Taille_Tab;
    }
    for(int i=0;i<Taille_Tab;i++ )
    {
    cout<< " Tab["<<i<<"]= ?    "; // Tab[i]=?
    cin>> NomTab[i];
    }
}

// Fonction Affiche_Tab permet d'afficher un tableau
 void Affiche_Tab (double NomTab[], unsigned int &Taille_Tab )
 {
    cout<< "le tableau enrtree est le suivant:"<<endl;
    for(int i=0;i<Taille_Tab;i++ )
    {
    cout<< "Tab["<<i<<"]=" << "\t"<< NomTab [i]<<endl;
    }
}

/* Fonction Somme_Tab permet de calculer la somme
    des élements d'un tableau*/
 double Somme_Tab (double NomTab[], unsigned int &Taille_Tab )
{  double somme=0;
    for(int i=0; i<Taille_Tab;i++ )
    {
     somme+=NomTab[i];
    }
    return (somme) ;
}

int main()
{
   unsigned int N(5); double Tab1[N]={0};

    Lire_Tab(Tab1, N);// le nom du tableau représente l'adresse du
                      // premeir element Tab1[0]
    Affiche_Tab(Tab1, N);
    double s= Somme_Tab (Tab1, N);
    cout << "la somme des elements du tableau=  "<< s<<endl;
    return 0;
}
```

Remarques importantes :

Dans le programme précédent,

- au lieu de passer tous élément du tableau à fonction *Somme_Tab* , on est la transmis juste la référence du tableau (Tab1), ce qui permet d'éviter la recopier de tous les éléments du tableau (***avantage de passage par référence*** !!).
- la taille du tableau N est passée par référence, alors la fonction *Lire_Tab* peut modifier la valeur de N si 0<N<100 (***avantage de passage par référence*** !!).

II.6.6 Passage par référence constante

Le passage par référence constante est utilisé lorsqu'on souhaite que le passage par référence ne modifie pas les paramètres transmis.

Exemple II.22

Dans le programme suivant la fonction *division* reçois trois paramètre :

- le premier est passé par valeur alors la fonction *division* **ne peut pas modifier** la valeur de la variable a.
- Le deuxième paramètre est passé par référence alors la fonction *division* **peut modifier** la valeur de b.
- Le dernier paramètre est transmis par une référence constante, alors la fonction **ne peut pas modifier** la valeur de c ;

```
1   #include <iostream>
2
3   using namespace std;
4
5   void division  (float x, float& y , const float & z )
6   {
7    x=x/z; y=y/z;
8   }
9   int main()
10  {
11   float a(1),b(8),c(12);
12   division (a,b,c);
13   cout<< "a= "<<a <<endl;
14   cout<< "b= "<<b <<endl;
15   cout<< "c= "<<c <<endl;
16  }
```

Le programme précédent affiche :

```
a= 1
b= 0.666667
c= 12
```

II.6.7 Surcharge d'une fonction

La surcharge d'une fonction permet d'utiliser le même nom pour plusieurs fonctions à condition qu'elles possèdent des listes de paramètres différentes (c.à.d. type ou nombre de paramètres différents).

Exemple II.23

Le programme suivant contient deux fonctions ont le même nom (Definir_Tab) ; la première à un seul paramètre (x) sert à déclarer un tableau statique à une seule dimension et renvoi l'adresse du premier élément (&Tableau1 [0] ainsi que la taille de tableau déclaré). La deuxième fonction à deux paramètre (x et y) sert à déclarer un tableau statique à deux dimensions et renvoi sa taille et l'adresse du premier élément du tableau ((&Tableau2 [0][0]).

```
1  #include <iostream>
2  using namespace std;
3
4  int* Definir_Tab (int x )
5  {
6  int Tableau1[x];
7  cout <<sizeof(Tableau1)/sizeof(int)<<endl;
8  return Tableau1;//ou &Tableau1[0]
9  }
10
11 int* Definir_Tab (int x, int y)
12 {
13 int Tableau2[x][y];
14 cout <<sizeof(Tableau2)/sizeof(int)<<endl;
15 return &Tableau2[0][0];
16 }
17
18 int main()
19 {
20 int *T1;
21 int *T2;
22 T1= Definir_Tab(5);
23 T2= Definir_Tab(5,5);
24 }
```

L'exécution du programme précédent s'affiche 5 et 25.

II.6.8 Séparation des prototypes et des implémentations des fonctions

Dans les sous sections précédentes, la fonction principale 'main' inclus tout le code du programme des fonctions utilisées (prototype et implémentations). Cependant, il est possible de séparer les prototypes et les implémentations des fonctions dans des fichiers séparées afin de rendre le programme principal plus modulaire et plus compréhensible. La séparation prototypes-implémentation, offre la possibilité d'intégrer directement les fonctions dans des autres programmes sans besoin de le récrire. Un autre avantage, c'est que l'utilisateur ne dépend plus de détails techniques des fonctions, par exemple, nous avons utilisé précédemment la fonction '*size*' sans besoin au retour à son code d'implémentation.

Pour réaliser cette opération (séparation prototypes-implémentation), il nécessaire de crée deux fichiers, le premier est d'extension '**.h'(**header**)** dont lequel on met le prototype de la fonction. Le second est d' d'extension '**.cpp'** (fichier source) dont lequel on met l'implémentation de la fonction.

Pour ajouter le fichier header **(.h),** aller vers le menu du Cod::Block et cliquer sur '**File'** ensuite '**New'** et ensuite '**File' voir** figure II. 1(a). Après double clic sur le '**fichier C/C++ header'** voir figure II. 1(b). Après une fenêtre de dialogue sera ouverte et demande un **nom** pour la fonction et le **chemin d'enregistrement** voir figure II. 1(c).

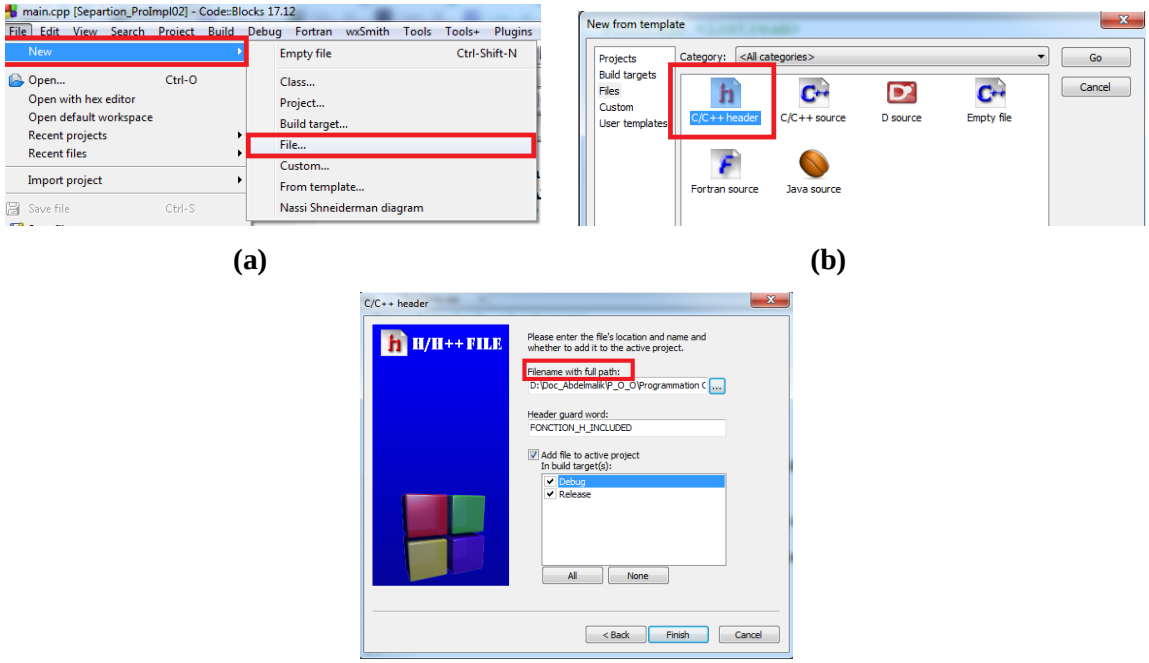


Figure II. 1 : Etapes de création d'un fichier entête (header) dans le Code::Block.

Pour ajouter le fichier source **(.cpp),** aller vers le menu du Cod::Block et cliquer sur '**File'** ensuite '**New'** et ensuite '**File' voir** figure II. 1(a). Après double clic sur le '**fichier C/C++ source** voir figure II. 2(a). Dans la fenêtre de dialogue, entrer le nom de la fonction (même nom choisi pour le fichier header) et le **chemin d'enregistrement** voir figure II. 1(c).

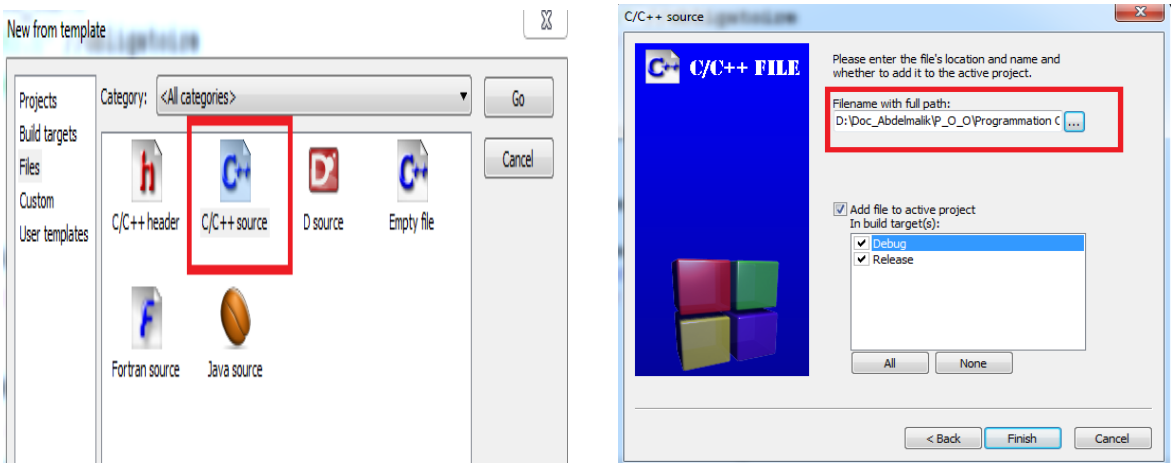
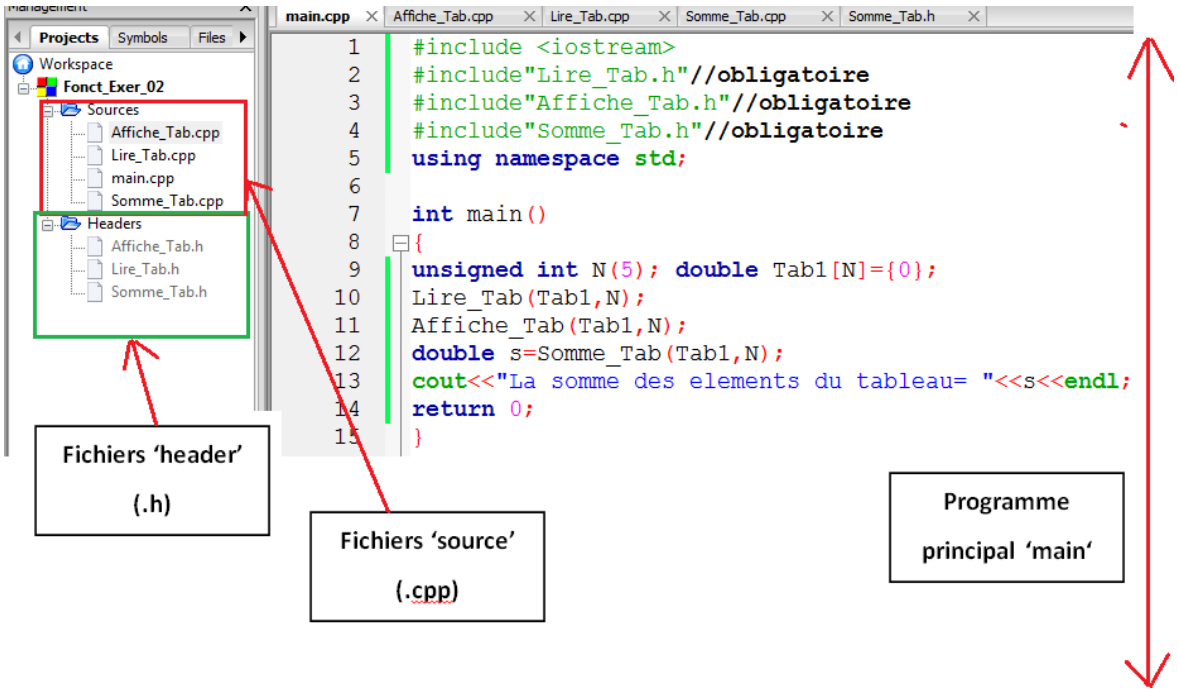


Figure II. 1 : Etapes de création d'un fichier source (.cpp) dans le Code::Block.

Exemple II.24

Dans cet exemple, nous allons séparer les fonctions '*Lire_Tab'*, '*Affiche_Tab'* et '*Somme_Tab'* développées précédemment dans l'Exercice II.2 de la fonction principale 'main'.

Programme principal 'main'.



Prototypes de la fonction ' Lire_Tab' (**Lire_Tab.h**)

```
#ifndef LIRE_TAB
#define LIRE_TAB
// prototype de la fonction Lire_Tab
void Lire_Tab(double NOmTab[], unsigned int &Taille_Tab);
#endif // LIRE_TAB

```

Prototypes de la fonction ' Affiche_Tab' (**Affiche_Tab.h**)

```
#ifndef AFFICHE_TAB
#define AFFICHE_TAB
// prototype de la fonction Affiche_Tab
void Affiche_Tab(double NOmTab[], unsigned int &Taille_Tab);

#endif // AFFICHE_TAB
```

Prototypes de la fonction ' Somme_Tab' (**Somme_Tab.h**)

```
#ifndef SOMME_TAB
#define SOMME_TAB
// prototype de la fonction Somme_Tab
double Somme_Tab(double NOmTab[], unsigned int &Taille_Tab);

#endif // SOMME_TAB
```

Implémentation de de la fonction ' Lire_Tab' (**Lire_Tab.cpp**)

```
#include<iostream>
#include"Lire_Tab.h"//obligatoire
using namespace std;
// implémentation de la fonction Lire_Tab
void Lire_Tab(double NOmTab[], unsigned int &Taille_Tab)
{
    while (Taille_Tab<=0||Taille_Tab>100)
    {
```

```
        cout << "Entrer la taille du tableau (max=100)"<<endl;
        cin >> Taille_Tab;
    }
    for(int i=0; i<Taille_Tab; i++)
    {
        cout << "Tab["<<i<<"]=   ?"; //Tab[i]=?
        cin >> NOmTab[i];
    }
}
```

Implémentation de de la fonction ‘ Affiche_Tab’ (**Affiche_Tab.cpp**)

```
#include<iostream>
#include"Affiche_Tab.h"//obligatoire
using namespace std;
// implémentation de la fonction Affiche_Tab
void Affiche_Tab(double NOmTab[], unsigned int &Taille_Tab)
{
    cout << "le tableau entré est le suivant: "<<endl;
    for(int i=0; i<Taille_Tab; i++)
    {
        cout << "Tab["<<i<<"]="<<"\t"<< NOmTab[i]<<endl;
    }
}
```

Implémentation de de la fonction ‘ Somme_Tab’ (**Somme_Tab.cpp**)

```
#include<iostream>
#include"Somme_Tab.h"//obligatoire
using namespace std;
// Implémentation de la fonction Somme_Tab
double Somme_Tab(double NOmTab[], unsigned int &Taille_Tab
{
 double somme=0;
 for(int i=0; i<Taille_Tab; i++)
    {
       somme+=NOmTab[i];
    }
 return somme;
 }
```

Remarques importantes :

1- Dans les fichiers header (.h), les prototypes des fonctions doivent se termine par **point-virgule**.

2- Il est indispensable d’ajouter l’instruction **#include** suivi par le **nom de la fonction** et **l’extension (.h)** entre deux **guillemet** doublé dans chaque fichier sources (.cpp) et dans le programme principale ‘main’

3- Les instructions **‘#ifndefin’**, **‘#defin’** et **‘#endif’** sont utilisés pour éviter que le compilateur à inclure le fichier ‘header’ plusieur fois (protection contre les inclusions multipes).

II. 7. Structure en C++

La structure est une variable spéciale et différente au tableau qui exige que ces éléments doivent le même type et de même taille, la structure permet d’avoir des éléments de types et de tailles

différents (int, chr, double..). De plus, elle peut contenir des fonctions, des pointeurs, des tableaux et même de sous-structures.

II.7.1 Syntaxe de déclaration d'une structure

Pour déclarer une structure, on utilise le mot clé **struct** suivi par le **nom** et les **variables** de la structure comme le montre l'exemple suivant :

Exemple II.25

```
#include <iostream>
using namespace std;
struct Etudiant //structure appellé Etudiant
{
// variables de la structue Etudiant
string nom="0",prenom="0",specialite="0";
unsigned int age=0;
double moyen=0;
 char classement='0';
};// pont vergule est obligatoire
```

Dans l'exemple précédent on a déclaré une nouvelle variable nommé (structure) **Etudiant** se compose de quatre (04) variables classiques (*string, unsigned int, double et char*).

II.7.2 Utilisation de la structure

Après avoir déclaré la structure Etudiant, on peut l'utiliser et manipuler toutes ses paramètres comme le montre l'exemple suivant :

Exemple II.26

```
#include <iostream>
using namespace std;
struct Etudiant //structure appellé Etudiant
{
// variables de la structue Etudiant
string nom="",prenom="",specialite="";
unsigned int age=0;
double moyen=0;
 char classement=' ';
};// pont vergule est obligatoire


int main()
{
struct Etudiant Etud01, Etud02;// deux structure non intialisées
// structure intialisée
Etud01.nom="Zorig";Etud01.prenom="MAlik";Etud01.specialite="Control";
Etud01.age=33;Etud01.moyen=15.08;Etud01.classement='B';

Etud02.nom="Zorig";Etud02.prenom="Anouar";Etud02.specialite="Electrotechnique";
Etud02.age=25;Etud02.moyen=17.00,Etud02.classement='A';
//Afichage des information du Etud01
cout << "Etud01:"<< endl;
cout << "Nom :"<< Etud01.nom<<endl;
cout << "Prenom :"<< Etud01.prenom<<endl;
cout << "Specialite :"<< Etud01.specialite<<endl;
cout << "Age :"<< Etud01.age<<endl;
cout << "moyenne :"<< Etud01.moyen<<endl;
cout << "classement :"<< Etud01.classement<<endl;
```

```
30
31  //Afichage des information du Etud02
32  cout << "Etud02 :"<< endl;
33  cout << "Nom :"<< Etud02.nom<<endl;
34  cout << "Prenom :"<< Etud02.prenom<<endl;
35  cout << "Specialite :"<< Etud02.specialite<<endl;
36  cout << "Age :"<< Etud02.age<<endl;
37  cout << "moyenne :"<< Etud02.moyen<<endl;
38  cout << "classement :"<< Etud02.classement<<endl;
39
40  return 0;
41  }
```

Ainsi l'exécution du programme précédent s'affiche :

```
Etud01:
Nom :Zorig
Prenom :MAlik
Specialite :Control
Age :33
moyenne :15.08
classement :B
Etud02 :
Nom :Zorig
Prenom :Anouar
Specialite :Electrotechnique
Age :25
moyenne :17
classement :A
```

II.7.3 Initialisation d'une structure

Il est possible d'initialiser d'une structure au moment de sa déclaration en mettant les valeurs de ces paramètres entre deux accolades et en les séparant l'une sur l'autre par des virgules. Dans l'exemple suivant le tableau de la structure *Chahid* a été définit et initialisé.

Exemple II.27

```
1   #include <iostream>
2   using namespace std;
3
4   struct Chahid //déclaration d'une structure
5   {
6   string nom,prenom,Grade_militaire;
7   unsigned int annee_naissance,annee_deces;
8   }// ne mettre pas ; !!
9   Chahid_revolution54[3]={
10  {"Didouche"   , "Mourad"   , "Colonel", 1927,1955},/*Chahid[01]*/
11  {"Ait Hamouda", "Amirouche", "Colonel", 1926,1959},/*Chahid[02]*/
12  {"Si"         , "El Haoues", "Colonel", 1923,1959} /*Chahid[03]*/
13  };
```

L'initilaisation suivante est également valable en C++ :

```
1   #include <iostream>
2   using namespace std;
3
4   struct Chahid //déclaration d'une structure
5   {
6   string nom,prenom,Grade_militaire;
7   unsigned int annee_naissance,annee_deces;
8   };
```

```
/* si vous mettez ; après la définition de la structure
il est nécessaire de mentionner le nom de la structure */
 struct Chahid Chahid_revolution54[3]={
{"Didouche"    , "Mourad"    , "Colonel", 1927,1955},/*Chahid[01]*/
{"Ait Hamouda", "Amirouche", "Colonel", 1926,1959},/*Chahid[02]*/
{"Si"          , "El Haoues", "Colonel", 1923,1959} /*Chahid[03]*/
};
```

Exercice II.3 :

Ecrire un programme permet de réaliser les tâches suivantes:

- Lire les informations de N étudiants (nom, prénom, spécialité, age, moyenne, rang de classement (A,B,C, ou D).
- Calcul la moyenne de classement de chaque étudiant (rang* moyenne, A=1,B=0.8,C=0.6,D=0.5).
- Affiche toutes les données saisis et calculés pour chaque étudiant.

Solution d'Exercice II.3

```
#include <iostream>
#include <string>
using namespace std;
struct Etudiant //déclaration d'une structure
{
string nom,prenom,specialite;
unsigned int age=0;
float moyen=0;
char classement='0';
float moyen_classement=0;
/* la fonction suivante permet de
 calculer la moyenne de classement*/
float calcul_moy_classement(char& x, float y)
 {
  jump:
  switch(x)
  {
   case 'A':
    { return y;     } break;
   case 'B':
    { return (y*0.8);     } break;
   case 'C':
    { return y*0.6;     } break;
   case 'D':
    { return y*0.5;     } break;
   default:// entrée non valide==> réintroduire le
    { do    // rang de classement
      {
       cout<<" votre classement ?( A ,B, C,ou D) ";
       cin >> x;
```

```
        goto jump;// aller ver l'étiquette jump
      }
      while ( (x!='A') && (x!='B') && (x!='C') && (x!='D') );
    }
   }//fin switch
 }// fin fonction
};// fin structure
```

Test de la structure 'Etudiant'

Dans le programme suivant, nous allons tester la structure 'Etudiant' dans la fonction principale 'main'.

```
int main ()
{
 int N(2);//N représente le nombre des étudiants
 struct Etudiant etudiant_master[N];// tableau de N structures
  // affectation de N structures
   for (int i=0;i<N;i++)
   {
    cout << " Nom de l'etudiant " <<i+1<<"?"
    /* la fonction getline() lit une chaine
    de caractère se termenne par un delimiter (exemple '#')*/
    getline(cin,etudiant_master[i].nom,'#');
    cout << " Prenom de l'etudiant " <<i+1<< "?";
    getline(cin,etudiant_master[i].prenom,'#');
    cout << " Specialite de l'etudiant "<<i+1<< "?";
    getline(cin, etudiant_master[i].specialite,'#');
    cout << " Age ? de l'etudiant "<<i+1<<" ?";
    cin>> etudiant_master[i].age;
    cout << " Moyenne de l'etudiant "<<i+1<<" ?";
    cin>> etudiant_master[i].moyen;
    cout << " Classement de l'etudiant " <<i+1<<"?"
    cin>> etudiant_master[i].classement;
    etudiant_master[i].moyen_classement=
etudiant_master[i].calcul_moy_classement(etudiant_master[i].classement, etudiant_master[i].moyen);
   }
   // affichage du contenu de N structures

   for (int i=0;i<N;i++)
   {
   cout << "Etudiant "<< i+1<< endl;
   cout << "Nom :  "<<etudiant_master[i].nom<<endl;
   cout << "Prenom : "<<etudiant_master[i].prenom<<endl;
   cout << "Specialite :  "<<etudiant_master[i].specialite<<endl;
   cout << "Age :  "<<etudiant_master[i].age<<endl;
   cout << "Moyenne :  "<<etudiant_master[i].moyen<<endl;
   cout << "Rang classement :  "<<etudiant_master[i].classement<<endl;
   cout << "moyen de classement :  "<<etudiant_master[i].moyen_classement<<endl
   }
return 0;
}
```

Résultat d'exécution du programme précédent pour N=2:

```
 Nom de l'etudiant 1?  Zabana#
 Prenom de l'etudiant 1?  Ahmed#
 Specialite de l'etudiant 1?  Systems Embarques#
 Age de l'etudiant 1 ?  25
 Moyenne de l'etudiant 1 ?  15.00
 Classement de l'etudiant 1? V
 votre classement ?( A ,B, C,ou D) B
 Nom de l'etudiant 2?  Ben Boulaid#
 Prenom de l'etudiant 2?  Mustapha#
 Specialite de l'etudiant 2?  Instrumentation#
 Age de l'etudiant 2 ?  24
 Moyenne de l'etudiant 2 ?  17.00
 Classement de l'etudiant 2? A
Etudiant 1
Nom :  Zabana
Prenom :
Ahmed
Specialite :
Systems Embarques
Age :  25
Moyenne :  15
Rang classement :  B
moyen de classement :  12
Etudiant 2
Nom :
Ben Boulaid
Prenom :
Mustapha
Specialite :
Instrumentation
Age :  24
Moyenne :  17
Rang classement :  A
moyen de classement :  17
```

Remarque:

Vous pouvez améliorer le programme précédent en ajoutant des vérifications sur la lecture de l'âge (0<age<130<) et de la moyenne (0<moyen<20) de l'étudiant comme on a fait pour le rang de classement (Rang={A, B, C, ou D}.

II.7.4 Opération de résolution de la portée (::)

Dans le programme précédent (*Exercice II.3),* la fonction *calcul_moy_classement* a été déclarée et définit à l'intérieur de la structure *Etudiant.* Cependant, l'implémentation (définition) de la fonction peut être faite à l'extérieur de la structure en utilisant l'opérateur de résolution de la portée (::). *L'*suivant (*Exemple II.28*) montre comment définir la *calcul_moy_classement* hors de la structure *Etudiant.*

Exemple II.28

```
3   #include <iostream>
4   using namespace std;
5   struct Etudiant //déclaration d'une structure
6   {
7   string nom="0",prenom="0",specialite="0";
8   unsigned int age=0;
9   float moyen=0;
10  char classement='0';
11  float moyen_classement=0;
12  // déclaration de  la fonction 'calcul_moy_classement':
```

```
13    float calcul_moy_classement(char& , float);
14    };
15    // définition de la fonction 'calcul_moy_classement'
16     float Etudiant::calcul_moy_classement(char& x, float y)
17     {
18      jump:
19      switch(x)
20      {
21       case 'A':
22        { return y;     } break;
23       case 'B':
24        { return (y*0.8);     } break;
25       case 'C':
26        { return y*0.6;     } break;
27       case 'D':
28        { return y*0.5;     } break;
29       default:// entrée non valide==> réintroduire le
30        { do    // rang de classement
31          {
32           cout<<" votre classement ?( A ,B, C,ou D) ";
33           cin >> x;
34           goto jump;// aller ver l'étiquette jump
35          }
36          while ( (x!='A') && (x!='B') && (x!='C') && (x!='D') );
37        }
38      }//fin switch
39    }// fin fonction
```

II.7.5 Opérations sur les structures

Il est possible de faire une affectation globale d'une structure à condition que les variables de structures soient issues de la même déclaration.

Exemple II.29

Dans le programme suivant tous les paramètres de la structure X2 sont affectés par les paramètres de la structure X1, alors X2 contient les paramètres de la structure X1. **Par contre les autres opérations ne sont pas valables** telles que X3=X1+X2, X3=X1*X2….). Pour le faire il fallait ce qu'on appelle surcharge d'opérateurs (voir section suivante).

```
1   #include <iostream>
2
3   using namespace std;
4
5   struct x //déclaration d'une structure
6   {
7   unsigned int a;
8   float b;
9   int c;
0   };
1
2   int main()
```

```
using namespace std;

struct x //déclaration d'une structure
{
unsigned int a;
float b;
int c;
};

int main()
{
 struct x X1={10,3.14,5},X2,X3;
 X2= X1; // equivalent à X2.a= X1.a; X2.b= X1.b;X2.c= X1.c
 X3.b= X1.b+ X2.b;
 X3.c= X1.c+ X2.b;// Attenstion on peut par faire
 X3.a= X1.a+ X2.a;//l'opération suivante  X3= X1+X2;
 X3.b= X1.b+ X2.b;
 X3.c= X1.c+ X2.b;
 return 0;
}
```

II.7.6 Surcharge des opérateurs

En C++ les opérateurs tels que (+, -, *, +=, -=, *=, >…..) peuvent être utilisés seulement avec les variables simples (float, int , double ,….). Cependant, en C++ on peut définir des opérateurs qui peuvent être utilisés avec des variables complexes comme les structures par exemple, cette définition appelée surcharge des opérateurs.

Exemple II.30

Dans le programme suivant, on déclarée une structure *ma_stru* à deux membres de type float, ensuite, nous avons surchargé un opérateur de nommé + qui fait la somme de deux structures et un autre nommé – qui fait la soustraction de deux structures. De la même manière on peut surcharger les opérateurs (=+, >, <, *….) pour la structure *ma_stru.*

```
#include <iostream>
using namespace std;

struct  ma_stru{
float x,y;
};
//Surcharge de l'opérateur '+' pour la structure  ma_stru
ma_stru operator+(ma_stru A, ma_stru B)
{
 ma_stru C={C.x=A.x+B.x, C.y=A.y+B.y};
return C;
}
//Surcharge de l'opérateur '-' pour la structure  ma_stru
ma_stru operator-(ma_stru A, ma_stru B)
{
 ma_stru C={C.x=A.x-B.x, C.y=A.y-B.y};
return C;
}
```

```
19
20  int main()
21  {
22  struct ma_stru D={3.14, 1.5},E,F,G;
23  E=D;// affectation simple, alors E.x=10 et E.y=1.5
24  F=D+E;/* addition de deux structures appelle l'opérateur '+'
25  définit pour la structure 'ma_stru', alors F.x=6.28 F.y=3;*/
26  cout << "F.x=  "<< F.x<<endl;
27  cout << "F.y=  "<< F.y<<endl;
28  G=E-F;/* soustraction de des structures appelle 'opérateur '-'
29  définit pour la structure 'ma_stru', alors F.x=-3.14 F.y=-1.5;*/
30  cout << "G.x= "<< G.x<<endl;
31  cout << "G.y= "<< G.y<<endl;
32  return 0;
33  }
```

II. 8. Résumé

Ce chapitre abord quatre notions essentielles en C++. La première partie du chapitre été consacrée à la notion des tableaux statique dynamiques. Dans la deuxième partie, nous avons abordé la notion du pointeur et son utilisation avec les tableaux et pour l'allocation dynamique de la mémoire. Dans la troisième partie nous abordé en détail la déclaration, l'implémentation et l'utilisation des fonctions. La dernière partie du chapitre a été consacrée pour la définition, la déclaration, et l'utilisation des structures. Plusieurs exemples et des exercices avec solutions sont présentés après chaque notion étudiée pour faciliter la compréhension.