ses composantes sont initialisées par la chaine de caractère Décembre, i.e. mois contient :

{ *Novembre*, *Novembre*, *Novembre*,} */

I.4.2 Accès aux composantes d'un tableau dynamique

L'accès aux composantes d'un tableau dynamique est identique à celui du tableau statique. L'exemple 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 :
  - la déclaration des tableaux dynamiques commence par le mot *vector* ensuite le type, le nom et la taille du tableau.
  - Le type de tableau doit être entre deux chevrons '< >'.
  - La taille du tableau doit être entre deux parenthèses (n'est pas entre crochets).

I.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 :

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.

```cpp
#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.

I.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.

I.6.1 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 :

```cpp
#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.
}
```

I.6.2 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 :

```cpp
#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 : 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

I.6.3 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 :

```
#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 <, >, <=, >=, ==, !=.

Exemple01 : le programme suivant lit au clavier deux tableaux et les affiche en utilisant seulement les pointeurs.

```
#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;
}
```

Exemple02 : Le programme suivant lit et stocker N valeurs entières en utilisant l'allocation dynamique de la mémoire.

```
#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
  pT= new int;//Demande une case mémoire pouvant stocker un entier
              // (l'adresse de cette case est stockée dans le pointeur pT)
  int *ad_pT (0);
  ad_pT=pT;   // sauvegarder l'adresse de pT dans ad_pT
//Lecture et stockage des valeurs
  while(pT< ad_pT+nbr_valeurs )
  {
     cout<< " Entrer une valeur ";
     cin >>*pT;
     pT++;
  }
  pT=ad_pT;
  //Affichage des valeurs introduites
 cout << " Valeurs introduites=[";
  while(pT< ad_pT+nbr_valeurs)
  {
     cout <<*pT<<' ';
     pT++;
  }
   cout << "]"<<endl;
}
```

I.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.

I.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

I.6.2 Définition des fonctions en C++

La définition d'une fonction est composée de deux parties :

L'entête 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,… paramètreN)

Exemple01 :

```
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)*/
```

Exemple02 :

```
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)*/
```

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

Exemple : corps (implémentation) de la fonction 'max2Val' :

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

Exemple : implémentation de la fonction 'SelectChar' :