Support de cours Langage C++

Dr. ASSIE Brou Ida, UFHB

Table des matières

Chapitre 1 : Présentation de C++

Apparu au début des années 90, le langage C++ est actuellement l'un des plus utilisés dans le monde, aussi bien pour les applications scientifiques que pour le développement des logiciels.

Le langage C++ est le descendant du langage C. Bien que semblables au premier abord, ces deux langages sont néanmoins *différents*. Le langage C++ propose de nouvelles fonctionnalités, comme la programmation orientée objet (POO). Elles en font un langage très puissant qui permet de programmer avec une approche différente du langage C.

1. Qu'est-ce que le programme C++ ?

Ecrire un programme, c'est fournir à un ordinateur, une série d'instructions qu'il doit exécuter. Ces instructions sont généralement écrites dans un langage dit *évolué* et sont traduites en *langage machine* (qui est le langage du microprocesseur) avant d'être exécutées. Cette traduction s'appelle *compilation* et elle est effectuée automatiquement par un programme appelé *compilateur*.

Cette traduction automatique implique au programmeur d'écrire les instructions selon une syntaxe rigoureuse et de déclarer les données et fonctions à utiliser. Ainsi, le compilateur pourra réserver aux données une zone adéquate en mémoire et pourra vérifier que les fonctions sont correctement employées.

Pour un programme écrit en C++, il faut commencer par écrire un ou plusieurs fichiers source. Ensuite, il faut compiler ces fichiers source grâce au compilateur afin d'obtenir un programme exécutable. Les fichiers source sont des fichiers texte lisibles dont le nom se termine en général par l'extension « .cpp » ou « .h ». Les fichiers exécutables portent en général l'extension «.exe » sous Windows et ne portent pas d'extension sous Linux.

Les fichiers avec l'extension « .cpp » contiennent des instructions et ceux avec l'extension « .h », ne contiennent que des déclarations communes à plusieurs fichiers d'extension « .cpp ». Ce qui permet une compilation correcte de ceux-ci.

Pour ce faire, dans un fichier « .cpp » on prévoit l'inclusion automatique des fichiers « .h » qui lui sont nécessaires, grâce aux directives de compilation « #include ».

2. Exemple de programme C++

Voici un premier programme

```
#include <iostream.h>
 using namespace std;
 int main()              //Ici, débute le programme principal
          {
           cout << "BONJOUR"  << endl;
           return 0;
          }
```

1. La directive « **#include <iostream.h>** » : En général au début du programme, un certain nombre d'instructions commençant par #include est placé. Cette instruction permet d'inclure dans le programme la définition de certains objets, types ou fonctions. Le fichier « **iostream.h** » est le fichier à inclure. Il peut être soit à l'intérieur des chevrons < et >, soit entre guillemets. #include inclut le fichier **iostream.h** en le cherchant d'abord dans les chemins configurés, puis dans le même répertoire que le fichier source.

2. Le fichier « **iostream.h** » contient un certain nombre de définitions d'objets intervenant dans les entrées/sorties du programme, c'est-à-dire dans l'affichage à l'écran ou dans des fichiers. La définition de « cout » se trouve dans ce fichier.

3. La directive **« using namespace std** » indique l'utilisation de l'espace de nommage std. Un espace de nommage est un ensemble de classes dont cout fait partie.

4. La directive « **//Ici, débute le programme principal** » est un commentaire. Il est utilisé pour rendre un programme plus lisible.

5. La fonction **main()**, c'est à cet endroit que va commencer l'exécution du programme. Exécuter un programme en C++, c'est exécuter la fonction main de ce programme. Tout programme en C++ doit donc comporter une fonction main.

   Il y a 3 éléments sur cette ligne :

   - cout : commande l'affichage d'un message à l'écran ;
   - "BONJOUR" : indique le message à afficher ;
   - endl : crée un retour à la ligne dans la console.

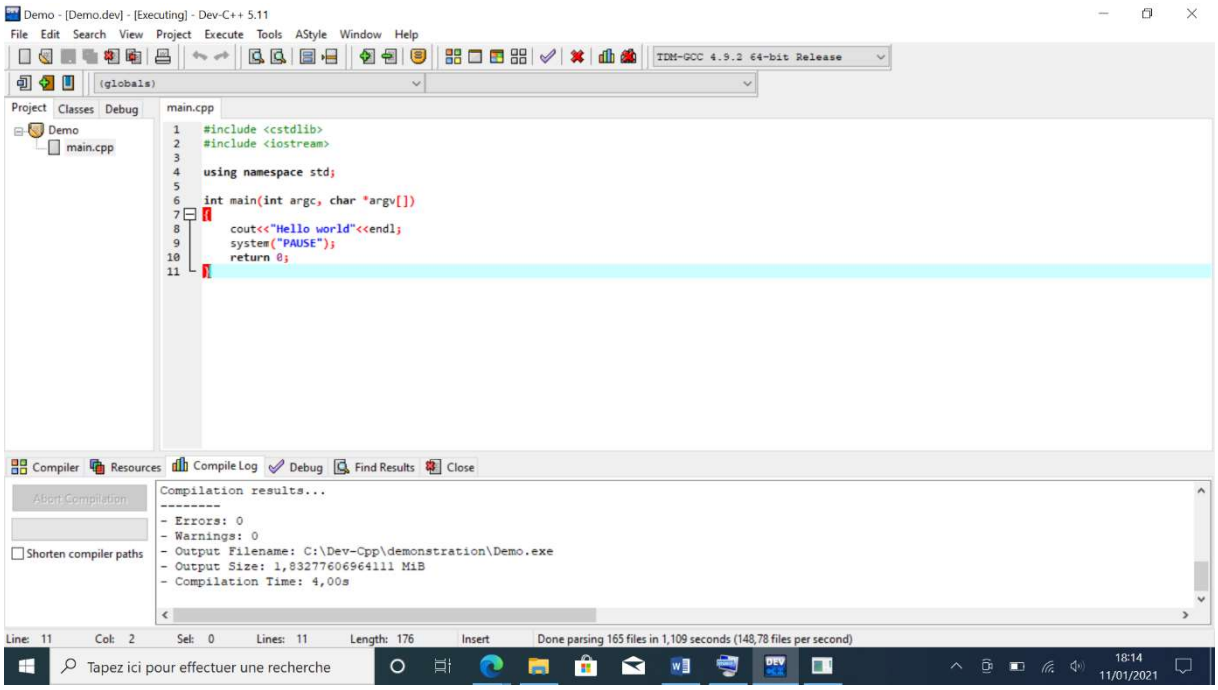
➢ **<u>Remarque</u> :**

- ❖ Contrairement au langage Pascal, *le langage C++ fait la différence entre lettres minuscules et majuscules* : par exemple, les mots toto et Toto représentent deux variables différentes.
- ❖ Il faut savoir qu'il existe 2 types de programmes : les programmes graphiques et les programmes console.

3. Téléchargement et Installation de Dev C++

Après avoir télécharger le logiciel, il faut procéder à l'installation en suivant les différentes instructions.

Interface de présentation



<u>Travaux pratiques</u> : Installation de Dev C++

1. Lancez Dev C++,
2. Créez un projet et donnez-lui un nom avec l'extension .cpp,
3. Ecrivez notre premier programme C++
4. Compilez et exécutez le programme

Chapitre 2 : Eléments du langage

1. Variables

a- Terminologie

Une *variable* est caractérisée par :

- son nom : mot composé de lettres ou chiffres,
- son type : précisant la nature de cette variable (nombre entier, caractère, objet etc.),
- sa valeur : qui peut être modifiée à tout instant.

b- Types de base

- **vide** : void . Aucune variable ne peut être de ce type. On verra l'usage un peu plus loin.
- **entiers**, par taille-mémoire croissante :
  - char, stocké sur un octet ; valeurs : de -2^7 à $2^7- 1$ (-128 à 127),
  - short, stocké sur 2 octets ; valeurs : de -2^{15} à $2^{15}-1$ (-32768 à 32767),
  - long, stocké sur 4 octets ; valeurs : de -2^{31} à $2^{31} - 1$,
  - int, coïncide avec short ou long, selon l'installation.
- **réels**, par taille-mémoire croissante :
  - float, stocké sur 4 octets ; précision : environ 7 chiffres,
  - double, stocké sur 8 octets ; précision : environ 15 chiffres,
  - long double, stocké sur 10 octets ; précision : environ 18 chiffres.

➢ **Remarque :** Le mot « unsigned » précédant le nom d'un type entier indique qu'il s'agit d'un entier *non signé* ; un tel entier est toujours positif.

- **Valeurs littérales (ou explicites)**

❖ caractères usuels entre apostrophes, correspondant à des entiers de type char. Par exemple :

'A' (= 65).

A connaitre aussi les caractères spéciaux suivants :

- '\n': retour à la ligne,
- '\t': tabulation.

- ❖ chaînes de caractères entre guillemets, pour les affichages. Par exemple :

"Au revoir!\n".

c- Déclaration des variables

En C++, toute variable doit être déclarée avant d'être utilisée. La forme générale d'une déclaration :

<type> <liste de variables>;

Où, <type> est un nom de type ou de classe et <liste de variables> est un ou plusieurs noms de variables, séparés par des virgules.

- o **Exemples** :

  int i, j, k; // déclare trois entiers i, j, k

  float x, y; // déclare deux réels x, y

En même temps qu'on déclare une variable, il est possible de lui attribuer une valeur initiale. Cela est appelé l'initialisation. On pourra écrire par exemple :

float x, y = 1.0;

En particulier, on peut déclarer une *constante* en ajoutant const devant le nom du type, par exemple :

const double PI = 3.14159265358979323846;

Les valeurs des constantes ne peuvent pas être modifiées.

2. Expressions

a- Définition

Les *expressions* sont obtenues par la combinaison des noms de variables, des opérateurs, des parenthèses et des appels de fonctions. En C++, on appelle expression tout ce qui a une valeur.

b- Opérateurs arithmétiques

+ : addition,

- : soustraction,

* : multiplication,

/ : division. Attention : entre deux entiers, donne le quotient entier,

% : entre deux entiers, donne le reste modulo.

- **Exemple :**

19.0 / 5.0 vaut 3.8 et 19 % 5 vaut 4.

Dans les expressions, les règles de priorité sont les mêmes règles usuelles des mathématiques.

++ : incrémentation. Si i est de type entier, les expressions i++ et ++i ont toutes deux pour valeur la valeur de i. Mais, elles ne sont pas interprétées de la même manière :

- la première ajoute ensuite 1 à la valeur de i (*post-incrémentation*),
  Exemple : i = 2
  i++ = i+1,
  i++ = 3
- la seconde ajoute d'abord 1 à la valeur de i (*pré-incrémentation*).
  ++i = 1+i,
  ++i = 3

-- : décrémentation. i-- et --i fonctionnent comme i++ et ++i, mais retranchent 1 à la valeur de i au lieu d'ajouter 1.

c- Operateurs d'affectation

- = (égal), est la forme générale de l'expression d'affectation :

  <*variable*> = <*expression*>

L'<*expression*> est d'abord évaluée ; cette valeur donne la valeur de l'expression d'affectation, la <*variable*> reçoit ensuite cette valeur.

Exemple :

i = (j = k = 1) ;
i = 1.

- +=, -=, *=, /=, %=, <variable> <opérateur>= < expression>, l'expression est équivalente à :

<variable> = <variable> <opérateur> <expression>

Par exemple, l'expression i += 3 équivaut à i = i + 3.

d- Conversion de type

L'expression d'affectation peut provoquer une conversion de type. Par exemple, supposons déclarés :

```
int i;
float x;
```

Alors, si i vaut 3, l'expression x = i donne à x la valeur 3.0 (conversion entier → réel). Inversement, si x vaut 4.21, l'expression i = x donne à i la valeur 4, partie entière de x (conversion réel → entier).

On peut également provoquer une conversion de type grâce à l'opérateur () (*type casting*).

Avec x comme ci-dessus, l'expression (int)x est de type int et sa valeur est la partie entière de x.

e- Opérateurs d'entrées-sorties

Ce sont les opérateurs << et >>, utilisés avec les objets prédéfinis cout et cin déclaré dans la bibliothèque <iostream>.

- cout << <*expression*> : affiche à l'écran de la valeur de <*expression*>,
- cin >> <*variable*> : lire au clavier de la valeur de <*variable*>

On peut aussi modifier l'apparence des sorties grâce aux expressions comme endl qui provoque un passage à la ligne.

f- Instructions

Il existe (2) deux types d'instructions : l'instruction-expression et l'instruction-bloc.

- instruction-expression :

  <*expression*>;

  Cette instruction n'est utile que si l'<*expression*> a un effet.
- instruction-bloc :

```
{
    <déclarations et instructions>
}
```

3. Structures de contrôle

Ce sont des instructions qui permettent de contrôler le déroulement des opérations effectuées par le programme : instructions if et switch (branchements conditionnels), instructions while, do et for (boucles).

a- Instructions conditionnelles

❖ **L'instruction if**

if (<*expression entière*>)
    <*instruction1*>
else
    <*instruction2*>

L'<*expression entière*> est évaluée. Si sa valeur est différente de 0, l'<*instruction1*> est effectuée ; sinon, l'<*instruction2*> est effectuée.

➢ **NB :** En C++, il n'y a pas de type *booléen*. On retiendra que :

- *Toute expression entière* différente de 0 *(égale `a* 0*) est considérée comme vraie (resp. fausse).*
- La partie else <*instruction2*> est facultative.
- Les instructions <*instruction1*> et <*instruction2*> sont en général des instructions-blocs pour permettre d'effectuer plusieurs actions.

❖ **Opérateurs booléens**

! : non,
&& : et,
|| : ou.

❖ **Opérateurs de comparaison**

== : égal,
!= : différent,
< : strictement inférieur,
> : strictement supérieur,
<= : inférieur ou égal,
>= : supérieur ou égal.

❖ **L'instruction switch**

Cette instruction permet un branchement conditionnel multiple.

```
switch (<expression>)
 {
     case <valeur1> :
            <instructions>
     case <valeur2> :
            <instructions>
         ...
     case <valeurn> :
           <instructions>
     default :
            <instructions>
     }
```

L'*<expression>* est évaluée. S'il y a une case avec une valeur égale à la valeur de l'*<expression>*, l'exécution est transférée à la première instruction qui suit ce « case » ; si un tel « case » n'existe pas, l'exécution est transférée à la première instruction qui suit default.

La partie default : *<instructions>* est facultative.

L'instruction switch est en général utilisée avec l'instruction break; qui permet de sortir immédiatement du switch.

b- Boucles

❖ L'instruction while

while (*<expression entière>*)

<instruction>

L'*<expression entière>* est d'abord évaluée. Tant qu'elle est vraie, l'*<instruction>* est effectuée.

❖ L'instruction do

do

<instruction>

while (*<expression entière>*);

L'*<instruction>* est effectuée, puis l'*<expression entière>* est évaluée. Tant qu'elle est vraie, l'*<instruction>* est effectuée.

❖ L'instruction for

for (*<expression1>*;*<expression2>*;*<expression3>*)

<instruction>

- **Remarque** :
  - Le test est effectué avant la boucle dans l'instruction While.
  - Le test est effectué après la boucle.
  - Une boucle infinie peut se programmer avec for.
  - Pour le choix d'une boucle :
    - Si l'on sait combien de fois effectuer la boucle : utiliser for.
    - Sinon, utiliser while ou do :
      - s'il y a des cas où l'on ne passe pas dans la boucle : utiliser while,
      - sinon, utiliser do.

Travaux pratiques

Exercice 1 : Ecrire un programme en C++ qui permet d'afficher le message

« **Hello world, je suis votre nom** ».

Exercice 2 : Ecrire un programme en C ++ qui permet à une station d'essence de servir du gazole aux clients selon la capacité de sa réserve (fixée100000 Litres). Le litre du gazole coute 650 FCFA.

Le programme doit aussi permettre de connaitre le prix de la quantité consommée par un client donné et la capacité restante de sa réserve à chaque service rendu.

Chapitre 3 : Fonctions

Le chapitre précédent nous a permis de savoir comment faire varier le déroulement d'un programme en utilisant des boucles et des branchements. Dans ce chapitre, nous nous intéressons à la notion de fonctions.

1. Définition

Une fonction est un morceau de code qui accomplit une tâche particulière. Elle reçoit des données à traiter, effectue des actions avec et enfin renvoie une valeur. Les fonctions sont des sous programmes qui découpent un programme en petits programmes réutilisables. Autrement, découper un programme permet de s'organiser.

Les données entrantes s'appellent des **arguments** et on utilise l'expression **valeur retournée** pour les éléments qui sortent de la fonction.

2. Syntaxe d'écriture d'une fonction

Toutes les fonctions ont la forme suivante :

```
type nomFonction(arguments)
                {
        //Instructions effectuées par la fonction
                }
```

- Le « type » permet d'indiquer le type de variable renvoyée par la fonction.
- Le « nomFonction » est le nom de la fonction. L'important est de choisir un nom de fonction qui décrit ce qu'elle fait réellement.
- Le(s) « arguments » sont les données avec lesquelles la fonction va travailler.
- Les accolades délimitent le contenu de la fonction.

3. Construction d'une fonction

Donnons par exemple la définition d'une fonction qui reçoit un nombre entier et ajoute 5 à ce nombre. Elle renvoie le résultat obtenu.

```
int AjoutVal( int NbreEnt){
```

```
int ValeurObt( NbreEnt + 5 );
return ValeurObt ;
}
```

L'instruction « Return » indique quelle valeur ressort de la fonction. Ici, il s'agit de la valeur.

4. Utilisation d'une fonction

L'utilisation d'une fonction se fait grâce à l'appel de la fonction. Cet appel est une expression de la forme :

<nomFonction> (<liste d'expressions>)

Le mécanisme de l'appel de fonction est le suivant :

- chaque expression de la <*liste d'expressions*> est évaluée,
- les valeurs ainsi obtenues sont transmises dans l'ordre aux paramètres formels,
- le corps de la fonction est ensuite exécuté,
- la valeur renvoyée par la fonction donne le résultat de l'appel.

Autrement dit, lors de l'appel de la fonction, le programme exécute la totalité des instructions du corps de la fonction, puis reprend le programme juste après l'appel de la fonction.

Si la fonction ne renvoie pas de valeur, le résultat de l'appel est de type void.

Exemple : Appel de la fonction « AjoutVal » précédente.

```
#include <iostream>
using namespace std;
int AjoutVal( int NbreEnt)
  {
    int ValeurObt( NbreEnt + 5 );
    return ValeurObt ;
      }
  int main()              //Ici, débute le programme principal
    {
```

```
    int  a, b;
    cout << "Bonjour, on va ajouter 5 au nombre saisi "  << endl ;
    cout << "nombre saisi "<< endl ;
    cin>> a;
    b = AjoutVal(a) ;
    cout << "La nouvelle valeur est " <<  b << endl ;
    return 0;
}
```

5. Passage de paramètre par référence et passage de paramètre par valeur

Cette partie est consacrée à des notions un peu plus avancées des fonctions. Il s'agit du passage de paramètre par référence et passage de paramètre par valeur. C'est la manière dont l'ordinateur gère la mémoire dans le cadre des fonctions.

- **Passage de paramètre par valeur :** Lors de l'appel de la fonction, le programme évalue la valeur passée en paramètre. Puis, il alloue un nouvel espace dans la mémoire pour écrire cette valeur passée en paramètre. Cet espace mémoire alloué a pour adresse, le nom de la variable dans la fonction. Ensuite, le programme entre dans la fonction et exécute le bloc d'instructions. Enfin, une fois exécutée, la valeur de l'espace mémoire est ensuite copiée et affectée à la variable assignée à l'appel de fonction. On sort alors de la fonction.

  Dans l'exemple précédent, l'instruction « **b = AjoutVal(a)** ; » correspond à un appel de fonction par valeur. Nous pouvons faire l'interprétation suivante :

  1. Le programme évalue la valeur de « a ». Il trouve la valeur saisie « a ».
  2. Il réserve un nouvel espace dans la mémoire et y écrit la valeur saisie « a ». Cet espace mémoire possède l'étiquette « NbreEnt », le nom de la variable dans la fonction.
  3. Le programme entre dans la fonction et ajoute 5 à la valeur de la variable « NbreEnt ».
  4. La valeur de variable « NbreEnt » contenu dans la variable « ValeurObt » est ensuite copiée et affectée à la variable « b ». Et, on sort alors de la fonction.

- **Passage de paramètre par référence :** Plutôt que de copier la valeur comme dans le passage de paramètre par valeur, le programme alloue un nouvel espace dans la mémoire avec une deuxième adresse à la variable à l'intérieur de la fonction. Et, c'est une référence qu'il faut utiliser comme argument de la fonction. On peut passer un paramètre par référence (et non par copie) en indiquant dans l'entête de la fonction un & après le type. Il y a alors identification du paramètre de la fonction et de la variable de l'environnement appelant.

  Pour illustrer ce qui a été dit, créons le programme qui permet de saisir et d'afficher un nombre saisi au clavier.

[*] ParReference.cpp

```
1   #include <iostream>
2   using namespace std;
3   void saisir(int & n)
4   {
5    cout<<"Tapez un entier : "; cin>>n;
6   }
7   void affiche(int n)
8   {
9    cout<<"La valeur de l'entier : "<<n<<endl;
10  }
11  int main()
12  {
13   int x;
14   x=54;
15   saisir(x);
16   affiche(x);
17   return 0;
18  }
19
20
```

  Lors de l'appel de la fonction « saisir(x) », il y a identification des variables x et n. On remarque que toute modification de n modifie la valeur de x. Lorsqu'une valeur de n est saisie dans l'instruction « cin>>n », il y a modification du contenu de la variable x. Car, la variable « x » prend la valeur saisie. Et, ensuite la fonction « affiche » est appelée.

- **Remarque** : Cette méthodologie est indispensable pour créer de longs programmes. Il est donc indispensable de bien comprendre le rôle des paramètres d'une fonction.

Exercices :

1. Ecrire une fonction ayant en paramètre un entier et renvoie un booléen vrai si le nombre est premier et faux sinon.
2. Écrire une fonction qui reçoit en arguments 2 nombres flottants et un caractère, et qui fournit un résultat correspondant à l'une des 4 opérations appliquées à ses deux premiers arguments, en fonction de la valeur du dernier, à savoir : addition pour le caractère +, soustraction pour -, multiplication pour * et division pour / (tout autre caractère que l'un des 4 cités sera interprété comme une addition).

   Écrire un petit programme (main) utilisant cette fonction pour effectuer les 4 opérations sur les 2 nombres fournis en donnée.

Chapitre 4 : Tableaux

Dans de très nombreux programmes, plusieurs variables de même type sont manipulées et jouent quasiment le même rôle. Le langage C++, comme presque tous les langages de programmation, proposent un moyen simple de regrouper des données identiques dans un seul paquet. On appelle ces regroupements de variables des **tableaux**. Ce chapitre va permettre d'apprendre à manipuler les tableaux.

1. Enumérations

Les énumérations sont des listes de noms représentant des valeurs entières successives 0, 1, 2, . . . Une énumération se définit par un énoncé de la forme :

enum <nom> { <liste de noms> };

Exemple :

enum Jour {dimanche, lundi, mardi, mercredi, jeudi, vendredi, samedi};
enum Couleur {Jaune, vert, rouge, bleu};

Le type énumération peut être utilisé dans la déclaration de variables et de constantes. Par exemple :

- Jour JrExamen ; // déclaration de la variable « JrExamen » de type Jour.
- const Couleur Az = bleu ; // déclaration de la constante «Az » de type Couleur.

2. Tableaux

Un tableau est une collection de variables de même type, c'est-à-dire qu'il permet de stocker plusieurs variables de même type. La syntaxe de la déclaration d'un tableau est :

<type> <nomTableau> [<taille>];

où : <type> est le type des éléments du tableau,
<nomTableau> est le nom du tableau,
<taille> est une constante entière égale au nombre d'éléments du tableau.

Pour accéder à une case du tableau ou un élément du tableau, on utilise la syntaxe :

nomTableau[numeroCase]

Remarque :

1. Si t est un tableau et i une expression entière, on note t[i] l'élément d'indice i du tableau. Les éléments d'un tableau sont indicés de 0 à taille-1.
2. Les éléments d'un tableau sont stockés en mémoire de façon contiguë, dans l'ordre des indices.

Exemple : Le tableau fraction déclaré ci-dessous représente deux variables entières qui sont fraction[0] et fraction[1]. Ces variables se traitent comme des variables ordinaires :

```
int fraction[2]; // tableau de deux entiers.
    fraction[0] = 1; // La case d'indice 0 du tableau « fraction » a pour contenu 1.
    fraction[1]= 9 ; // La case d'indice 1 du tableau « fraction » a pour contenu 9.
```

3. Tableaux multidimensionnels

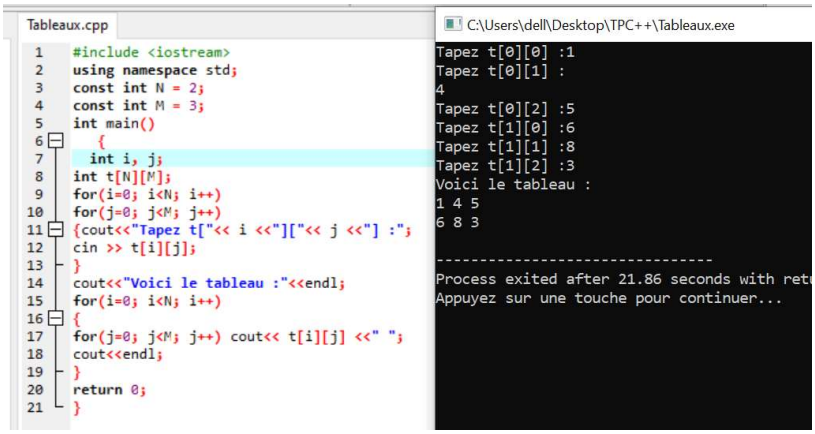
Les tableaux multidimensionnels sont des tableaux qui contiennent des tableaux. Pour déclarer un tel tableau, il faut indiquer les dimensions les unes après les autres entre crochets :

<type> <nomTableau> [<taille>][<taille>];

Pour accéder à une case d'un tableau multidimensionnel, il faut indiquer la position en X et en Y de la case voulue.

Par exemple, soit le tableau M d'entiers à deux dimensions (m, n) noté int M[m][n]. Le premier indice variant entre 0 et m-1, le second entre 0 et n-1. On peut voir M comme une matrice d'entiers à m lignes et n colonnes. Les éléments de M se notent M[i][j].

Exemple : On déclare un tableau t d'entiers comportant 2 lignes et 3 colonnes. Par 2 boucles imbriquées, on saisit un à un les 6 éléments du tableau, soit :



4. Utilisation des tableaux

Les tableaux peuvent être passés en paramètres dans les fonctions. Les tableaux ne peuvent être renvoyés (avec return) comme résultats de fonctions et l'affectation entre tableaux est interdite.

Passage de tableaux en paramètre dans une fonction

Lorsque l'on passe un tableau, il y a identification entre le tableau de l'environnement appelant et le paramètre de la fonction. Toute modification du tableau dans la fonction est répercutée dans le tableau de l'environnement appelant.

Exemple :

```
#include <iostream>
  using namespace std;
  const int n=4;
  void saisir(int t[n])
       {
        int i;
        for (i=0; i<n; i++)
        {
          cout<<"Tapez la valeur numero "<<i<<" : ";
          cin >> t[i];
        }
       }
    void affiche(int t[n])
       {
```

```
int i;
for (i=0; i<n; i++) cout<<"La valeur numero "<<i<<" est : "<<t[i]<<endl;
    }
int main()
{
 int a[n];
 saisir(a);
 affiche(a);
 return 0;
 }
```

Lors de l'appel de la fonction « saisir(a) », il y a identification du tableau « a » et du paramètre « t » de la fonction « saisir(a) » : toute modification de t modifie le tableau a. La fonction saisir « a» permet de demander à l'utilisateur de saisir une à une toutes les cases d'un tableau de n cases. La fonction « affiche(a) » à un tableau de n entiers en paramètres et affiche toutes les cases de ce tableau.

Le programme est structuré : il est constitué d'un ensemble de fonctions courtes dont le rôle peut être facilement identifié.

La fonction « main()» devient un programme très court.

Exercices :

1- Ecrire un programme en C++ qui permet d'inverser le contenu d'un tableau. Attention à ne pas échanger 2 fois le contenu de chaque case.

2- Ecrire un programme qui demande à l'utilisateur de saisir 10 entiers stockés dans un tableau ainsi qu'un entier V. Le programme doit rechercher si V se trouve dans le tableau et afficher "V se trouve dans le tableau" ou "V ne se trouve pas dans le tableau".

```
#include <iostream>
using namespace std ;
float oper (float v1, float v2, char op)
{ float res ;
switch (op)
{ case '+' : res = v1 + v2 ;
break ;
case '-' : res = v1 - v2 ;
break ;
case '*' : res = v1 * v2 ;
```

```
break ;
case '/' : res = v1 / v2 ;
break ;
default : res = v1 + v2 ;
}
return res ;
}
main()
{ float oper (float, float, char) ; // déclaration de oper
float x, y ;
cout << "donnez deux nombres réels : " ;
cin >> x >> y ;
cout << "leur somme est : " << oper (x, y, '+') << "\n" ;
cout << "leur différence est : " << oper (x, y, '-') << "\n" ;
cout << "leur produit est : " << oper (x, y, '*') << "\n" ;
cout << "leur quotient est : " << oper (x, y, '/') << "\n" ;
        }
```

Chapitre 5 : Programmation Orientée Objet

Apparue au début des années 70, la programmation orientée objet répond aux nécessités de l'informatique professionnelle. Elle offre aux concepteurs de logiciels une grande souplesse de travail tout en permettant une maintenance et une évolution plus aisée des produits.

Ce cours a pour but d'expliquer les règles de cette programmation.

1. Concept d'objet

Avant d'aborder le concept d'« objet» en informatique, nous rappelons ici le concept d'objet usuel.

Un objet usuel est un accessoire de la vie au quotidien qui peut être par exemple, une voiture. Une voiture a des fonctionnalités et des caractéristiques. Elle sert à transporter les hommes ou des marchandises.

Pour mener à bien cette fonctionnalité, elle possède des caractéristiques telles que :

- Puissance de sa vitesse,
- Capacité,
- ….

En utilisant l'ordinateur, nous allons simuler la plupart de ces objets usuels. Ces objets simulés sont appelés **objet** en informatique.

Un objet en informatique désigne une structure informatique regroupant :

- des variables, caractérisant l'état de l'objet,
- des fonctions, caractérisant le comportement de l'objet.

Les variables (resp. fonctions) s'appellent données-membres (resp. fonctions-membres ou encore méthodes) de l'objet. L'originalité dans la notion d'objet, c'est que variables et fonctions sont regroupées dans une même structure appelée **classe**.

2. Classe

Une classe est un ensemble d'objets de même type. Tout objet appartient à une classe, et on dit que cet objet est une instance de cette classe.

Par exemple, si l'on dispose de plusieurs voitures similaires à celle décrite à la section précédente, ces voitures appartiennent toutes à une même classe « Voiture », chacune est une instance de cette classe. En décrivant la classe « Voiture », on décrit la structure commune à tous les objets appartenant à cette classe.

Pour utiliser les objets, il faut d'abord décrire les classes auxquelles ces objets appartiennent.

La description d'une classe comporte deux parties :

- Une partie déclaration, représentant une fiche descriptive des données et fonctions-membres des objets de cette classe, sert d'interface avec le monde extérieur,
- Une partie implémentation, contenant la programmation des fonctions-membres.

Rappelons que la Programmation Orientée Objet est aussi dirigée par trois fondamentaux qu'il convient de toujours garder à l'esprit : **encapsulation**, **héritage** et **polymorphisme**.

3. Encapsulation

Dans la déclaration d'une classe, il est possible de protéger certaines données-membres ou fonctions-membres en les rendant invisibles de l'extérieur : c'est ce qu'on appelle l'encapsulation.

L'intérêt de l'encapsulation est de permettre à l'objet de contrôler certaines données-membres ou fonctions-membres. Dans notre exemple sur la classe « Voiture », supposons qu'on programme cette classe avec des données-membres comme :

- Kilomètre parcouru, (Kilometre)
- Temps mis pour atteindre la distance AB, (Temps)
- Vitesse par kilomètre heure, (Vitesse)

Permettre donc l'accès direct à la variable « Vitesse », est s'exposer à ce qu'elle soit modifiée depuis l'extérieur, et cela serait catastrophique puisque l'objet risque de perdre sa

cohérence (la vitesse dépend en fait du kilomètre parcouru). Il faut alors interdire l'accès ou permettre à l'objet de le contrôler.

4. Déclaration de données et fonctions membres publique et /ou privée

Dans la déclaration d'une classe, on dit qu'une donnée et/ou fonction-membre d'un objet est déclarée publique, si son utilisation se fait en dehors de l'objet, et privée si seul l'objet peut y faire référence.

Une approche simple et sûre consiste à déclarer systématiquement les données-membres privées et les fonctions-membres publiques.

Remarque : Par les « fonctions d'accès », on peut autoriser l'accès aux données-membres (pour consultation ou modification).

- Chaque fonction-membre est une unité de traitement correspondant à une fonctionnalité bien précise et qui sera propre à tous les objets de la classe. Cette unité de traitement dispose des informations suivantes :
    - Les valeurs des données-membres (publiques ou privées) de l'objet auquel elle appartient,
    - Les valeurs des paramètres qui lui sont transmises.

  En retour, elle fournit un résultat qui pourra être utilisé après l'appel. Ainsi, avant de programmer une fonction-membre, il faut identifier quelle est l'information qui doit y entrer (paramètres) et celle qui doit en sortir (résultat).

5. Programmation d'une classe

En C++, la programmation d'une classe se fait en trois phases :

- Déclaration,
- Définition,
- Utilisation.

a) Déclaration

La déclaration est la partie interface de la classe. Elle se fait dans un fichier dont le nom se termine par l'extension « .h ». Ce fichier se présente comme suit :

class Maclasse

{

public:

// déclarations des données et fonctions-membres publiques

private:

// déclarations des données et fonctions-membres privées

};

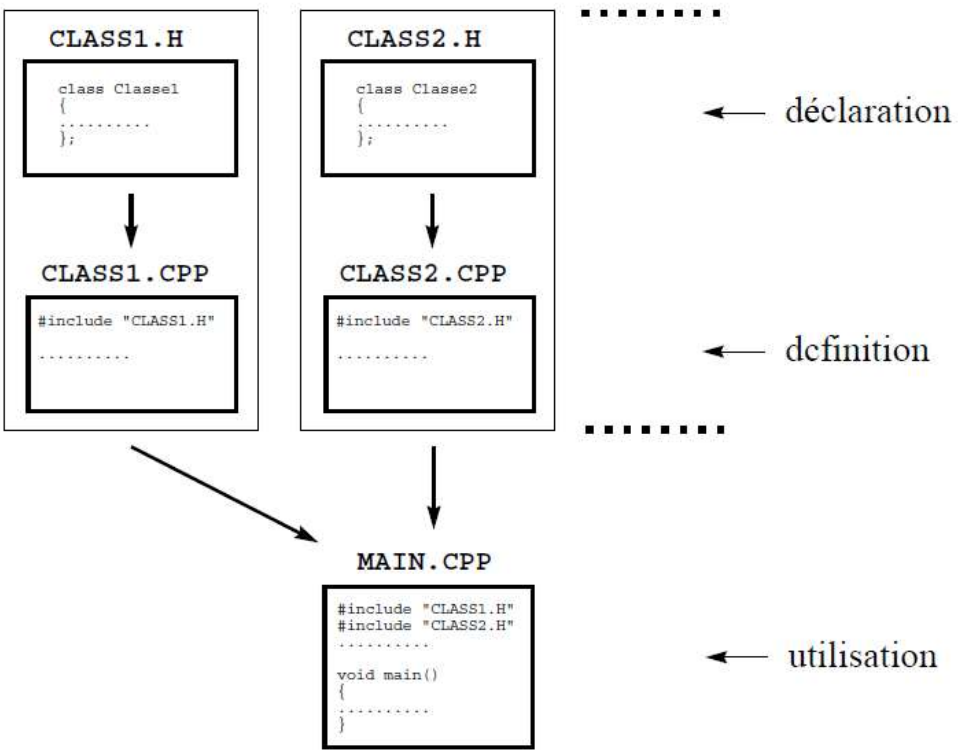
b) Définition

La définition est la partie implémentation de la classe. Elle se fait dans un fichier dont le nom se termine par l'extension « .cpp ». Ce fichier contient les définitions des fonctions-membres de la classe, c'est-à-dire le code complet de chaque fonction.

c) Utilisation

L'utilisation de la classe se fait dans un fichier dont le nom se termine par l'extension « .cpp ».

Remarque : Avec les classes, la structure d'un programme en C++ peut être schématisée par :



Les programmes sont généralement composés de fichiers :

- Pour chaque classe :
  un fichier .h contenant sa déclaration,
  un fichier .cpp contenant sa définition,
- Un fichier .cpp contenant le traitement principal.

Ce dernier fichier contient la fonction main, et c'est par cette fonction que commence l'exécution du programme.

6. Exercice d'application

A partir d'un exemple, nous allons détailler les différentes étapes qui mènent à la réalisation d'un programme. Il s'agit de simuler le jeu du "c'est plus, c'est moins" où un joueur tente de deviner un nombre choisi par le meneur de jeu.

Le fait de programmer avec des objets nous force à modéliser soigneusement notre application avant d'aborder le codage en C++.

Description du jeu : « *Le jeu oppose un joueur à un meneur. Le meneur choisit un numéro secret (entre 1 et 100). Le joueur propose un nombre. Le meneur répond par : "c'est plus", "c'est moins" ou "c'est exact". Si le joueur trouve le numéro secret en six essais maximum, il gagne, sinon il perd.* »

- Etape 1 : **Identification des classes**

(02) deux acteurs : Joueur et Meneur. Donc, nous avons 2 classes : classe « Joueur », classe « Meneur ».

- Etape 2 : **Fiches descriptives des classes**

Il s'agit de déterminer les données et fonctions-membres de chaque classe.

classe : Meneur	
privé :	*public :*
numsecret	Choisis
	Reponds

classe : Joueur	
privé :	*public :*
proposition	Propose
min	
max	