TD 1 : PREMIERES CLASSES EN C++

Compilation et exécution en C++ :

Compilation :

```
g++ fic1.cc fic2.cc … -o fichierExecutable
      ↳ fichierExecutable
```

Exécution :

```
      fichierExecutable
```

Exercice 1 :

a) Définir un tableau de 10 entiers et un pointeur vers ce tableau noté p. Ensuite effectuer un parcours du tableau avec le pointeur p.

b) Définir un tableau de 3 chaînes de caractères {"truc", "machin", "chose"}. Faites une allocation avec l'opérateur new et une désallocation.

Exercice 2 :

Ecrire une procédure **echange** qui prend deux arguments en entrée et les échanges.

Exercice 3 :

a) Définir une fonction **`tableauEcriture(…)`** qui prend un tableau d'entiers en entrée et met dans chaque case du tableau son indice. Le passage du tableau doit être en lecture et écriture.

b) Définir une autre fonction **`tableauLecture(…)`** qui prend aussi un tableau d'entiers en entrée et qui imprime sur la console la valeur des cases du tableau. Le passage du tableau donc doit être en lecture uniquement.

Exercice 4 :

Définir une classe **`Point`** permettant de représenter des points à deux dimensions :

a) Définir trois constructeurs pour cette classe : le premier donne aux deux coordonnées du point deux valeurs par défaut ; le deuxième affecte aux deux coordonnées du point les valeurs passées en paramètres ; le troisième donne aux deux coordonnées du point les coordonnées du point en paramètre.

b) Définir une fonction **`afficher()`** permettant d'afficher le point.

c) Définir une fonction **`cloner(const Point &)`** permettant d'affecter aux deux coordonnées du point le contenu du point en paramètre ;

d) Définir un destructeur pour la classe. Mettre dans cette fonction une seule instruction d'affichage à l'écran (i.e. **`cout << "appel au destructeur";`**) .

Tester la classe et observer les résultats.

Exercice 5 :

Ecrire une classe **`Segment`**. Un segment a deux points qui indiquent ses extrémités. Définir le(s) constructeur(s) et le(s) destructeur(s). Ajouter les méthodes **`longueur`** pour calculer sa longueur, **`estVertical`**, **`estHorizontal`** et **`estSurDiagonale`**. Ajouter aussi une méthode **`estcroise(const Segment &)`** pour déterminer si le segment croise le segment en paramètre.

Exercice 6 :

Concevoir la classe **`CString`** permettant de manipuler des chaînes de caractères et pouvant être utilisée avec le programme suivant. Les allocations et les desallocations de chaînes de caractères doivent se faire dynamiquement.

```
int main()
{
   CString s1( "toto" ),
       s2( 'q' ),
       s3 ;

   cout << "nbrChaines" << CString::nbrChaines() << endl ;
   //afficher le nombre de chaines créées

   s3 = s1.plus( 'w' ) ;
   cout << "s3=" << s3.getString() << endl ;

   if( s1.plusGrandQue(s2) ) // si s1 > s2 au sens alphabétique
      cout << "plus grand" << endl ;

   if( s1.infOuEgale(s2) ) // si s1 <= s2 au sens alphabétique
      cout << "plus petit" << endl ;

   s3 = s1.plusGrand( s2 ) ;// retourner s1 si s1>s2, s2 sinon
}
```

Exercice 7 :

On souhaite ré-utiliser la classe **`CString`** pour créer une nouvelle classe permettant de manipuler des définitions de mots. Soit **`Definition`** cette nouvelle classe. En remarquant que la définition d'un mot est composée du mot lui-même ainsi que de sa définition, concevoir la classe **`Definition`** pour qu'elle puisse fonctionner avec le programme suivant :

```
int main()
{
      Definition homer( "Homer", "Buveur de biere" ) ;
      cout<<"la definition du mot "<< homer.getClef()<<" est "<<
      homer.getDef() << endl ;
}
```