TP C++ N°2
Les classes

Exercice N°1

Après avoir récupéré les archives contenant les 2 programmes (Prog1, Prog2) ci-dessous sur Moodle, créez les répertoires nécessaires, compilez et exécutez le code. Analysez l'affichage à l'écran ATTENTIVEMENT. Expliquez par écrit l'information textuelle donnée à l'écran lors de l'exécution.

Programme 1 :

Dans le fichier « CVect1.h »

```
class CVect1 {

      int         m_Nbe;
      double*     m_pAdr;

      public:
            CVect1 ( int n );
            // Destructeur
            ~CVect1();
};
```

Dans le fichier « CVect1.cpp » associé

```
CVect1 :: CVect1 ( int n ) {
      m_Nbe = n;
      m_pAdr = new double[n];
      cout << "Constructeur usuel - adr objet : " << this << " - adr
      vecteur : " << m_pAdr << endl;
}


CVect1 :: ~CVect1 () {

      cout << "Destructeur objet - adr objet : " << this << " - adr
      vecteur: " << m_pAdr << endl;

      delete[] m_pAdr;
}
```

Dans le fichier « Lanceur1.cpp »

```
void fct ( CVect1 b ) {
      cout << "Appel de la fonction" << endl;
}


int main() {

      CVect1      a(5);
      fct(a);
      return 0;
}
```

TP C++ N°2
Les classes

Programme 2 :

Dans le fichier « CVect2.h »

```
class CVect2 {

      int         m_Nbe;
      double*     m_pAdr;

      public:
            CVect2 ( int n );
            // Constructeur de recopie (voir poly)
            CVect2 ( const CVect2& v );
            ~CVect();
};
```

Dans le fichier « CVect2.cpp »

```
CVect2 :: CVect2 ( int n ) {
      m_Nbe = n;
      m_pAdr = new double[n];
      cout << "Constructeur usuel - adr objet : " << this << " - adr
      vecteur : " << m_pAdr << endl;
}

CVect2 :: CVect2 ( const CVect2& v ) {
      m_Nbe = v.m_Nbe;
      m_pAdr = new double[m_Nbe];
      for (int i=0; i<m_Nbe; i++) m_pAdr[i] = v.m_pAdr[i];

      cout << "Constructeur de recopie - adr objet : " << this << " - adr
      vecteur : " << m_pAdr << endl;
}

CVect2 :: ~CVect2 () {
      cout << "Destructeur objet - adr objet : " << this << " - adr
      vecteur: " << m_pAdr << endl;
      delete[] m_pAdr;
}
```

Dans le fichier « Lanceur2.cpp »

```
void fct ( CVect2 b ) {
      cout << "Appel de la fonction" << endl;
}


int main() {
      CVect2      a(5);
      fct(a);
      return 0;
}
```

TP C++ N°2
Les classes

Exercice N°2

Ecrire une classe *CPoint* possèdant :

- Deux membres privés *m_Abs* et *m_Ord* de type *int*.
- Des accesseurs.
- Un constructeur qui prend en arguments deux entiers pour l'initialisation des membres et qui affiche une trace de passage (*cout << "construction de l'objet CPoint d'adresse : " << this << endl*).
- Un constructeur qui ne prend aucun argument et qui initialise les membres à zéro (indispensable pour la création d'un tableau de *CPoint*). Affiche également une trace de passage.
- Un copy-constructeur qui effectue « simplement » la recopie des attributs *m_Abs* et *m_Ord* et qui affiche une trace de passage.
- Un destructeur dans lequel on affichera simplement une trace de passage (*cout << "destruction de l'objet CPoint d'adresse : " << this << endl*).
- Une méthode *Presentation()* qui affiche les attributs de l'objet *CPoint* à l'écran.

Dans une première étape, écrire un programme de test de la classe *CPoint* (fichier *testCPoint.cpp*) mettant en oeuvre le test complet de la classe.

Dans une deuxième étape, créer un fichier *Lanceur.cpp* et écrire dans le lanceur *main ()* l'appel à 4 fonctions. Chacune de ces fonctions utilise en paramètre soit un objet *CPoint* (avec passage par valeur, par référence et par pointeur), soit un tableau d'objets *CPoint* préalablement construit et initialisé. Le type de paramètre passé à la fonction sera :

1. pour la fonction1 : un objet *CPoint theP (passage par valeur)*

2. pour la fonction2 : un objet *CPoint& theP (passage par référence)*

3. pour la fonction3 : un pointeur sur un objet *CPoint*, c'est-à-dire *CPoint* pTheP*

4. pour la fonction4 : un tableau d'objets *CPoint,* c'est-à-dire *CPoint* pTabPt,* de même que sa taille (*int tailleTab*)

Les points (ou tableau de points de taille donnée et complètement rempli) sont chaque fois créés dans le lanceur *main()*, puis ils sont passés en paramètre à la fonction. Chaque fonction a pour rôle d'afficher les attributs de l'objet *CPoint* (ou du tableau d'objets) à l'écran.

Exécuter chaque fonction séparément et analyser les affichages (les traces dans le constructeur et le destructeur).

TP C++ N°2
Les classes

Exercice N°3

Soit une classe (dans un fichier CVoiture.h) :

```
class CVoiture {

      private:
            int   m_Type;
            char* m_pNom;
            char* m_pCouleur;

      public:
            ~CVoiture();                          // éventuellement
            CVoiture ( int typeVt, char* nomVt, char* coulVt );
            CVoiture ( const CVoiture& vt );    // éventuellement
            void SetVoiture (int typeVt, char* nomVt, char* coulVt);
            void Presentation ();
};
```

La fonction membre *Presentation()* permet d'afficher les données membres d'un objet *CVoiture*.

A. Version1

Dans cette version (la + simple), les chaînes de caractères passées en paramètres au constructeur `CVoiture(...)` et à la méthode `SetVoiture(...)` ne sont PAS dupliquées (tous les objets créés se partagent donc les mêmes chaînes). Y-a-t-il dans ce cas nécessité d'écrire un copy-constructeur et un destructeur ?

B. Version2

Dans cette version, les chaînes de caractères passées en paramètres au constructeur `CVoiture(...)` et à la méthode `SetVoiture(...)` sont DUPLIQUEES (tous les objets créés possèdent leur propre copie des chaînes de caractères). Y-a-t-il dans ce cas nécessité d'écrire un copy-constructeur et un destructeur ?

Pour chacune de ces 2 versions :

Ecrire le fichier *CVoiture.cpp*.

Ecrire, dans une première étape, un programme de test de la classe *CVoiture* (fichier *testCVoiture.cpp* contenant un *main()*) mettant en oeuvre le test complet de la classe.

Dans une deuxième étape, le *Type*, le *Nom* et la *Couleur* seront passés sous forme d'arguments au lancement du programme de test.

Exemple : `testCVoiture 8 Espace Bleu`

Remarque : `int main( int argc, char *argv[], char *env[] )`

- `int argc` : contient le nombre de paramètres (y compris le nom du programme)
- `char* argv[]` : tableau de chaînes de caractères où chaque chaîne contient un paramètre de la ligne de lancement du programme (4 paramètres dans ce cas-ci)
- `char* env[]` : tableau de chaînes de caractères où chaque chaîne contient le contenu d'une variable d'environnement (pas utile ici)

Pour la conversion `char* ----> int`, utiliser la fonction C :

```
int atoi ( const char* str )
```
qui nécessite l'inclusion de la bibliothèque `<cstdlib>`

TP C++ N°2
Les classes

Exercice N°4

Ecrire une classe *CChaine* permettant de manipuler une chaîne de caractères.

Cette classe comprendra deux membres privés :

m_longueur (contient la longueur de la chaîne)
m_pTexte (pointeur sur la chaîne de caractères de type *char**)

Les constructions suivantes d'objet de type *CChaine* devront être possible :

```
CChaine   ch1 ( "Vaio" );   //   chaîne à recopier dans l'attribut
CChaine   ch2;              //   chaîne non nulle mais de longueur zéro
CChaine   ch3(100);         //   chaîne de longueur donnée et dont le
```

contenu est le caractère *espace* dans chacune des cases

Ecrire un programme de test de la classe *CChaine* (*testCChaine*) mettant en oeuvre le test complet de la classe.

Références nécessaires

- Les flux d'entrées/sorties « cin » et « cout » instances respectives des classes de la bibliothèque standard C++, « istream » et « ostream ».
- La définition des structures.
- La surcharge de fonction.
- La définition de classe en C++.
- Le constructeur de recopie.
- Le cours C++.

De l'aide sur C++ est disponible sur les sites suivants :

http://www.cplusplus.com/ref

http://www.cppreference.com