TP n°4 C++

Classes, Constructeurs, Destructeurs

Thème 1 : public, private

On rappelle que pour l'écriture de programmes C++ avec des classes, la programmation modulaire est plus que souhaitable. Ainsi, pour les exemples et exercices à traiter dans ce cahier de TP et les suivants, il faudra créer au minimum :

- Un fichier main.c avec le programme principal
- Un module comportant :
  - un fichier .h avec la définition de la classe (ou des classes) utilisée(s)
  - un fichier .cpp avec les définitions des méthodes de la ou des classe(s) utilisée(s).

Attention ! Toutes les définitions de méthodes présentes dans un fichier .h sont automatiquement qualifiées 'inline' par le compilateur !

Public ou privé ?

Horreur ! Vous ne vous rappelez plus si, par défaut, les membres d'une classe sont `public` ou `private`. Ecrivez un programme vous permettant de le déterminer.

Complexes

Ecrivez la classe `complex`, avec ses constructeurs par défaut et avec arguments (par défaut, créer le complexe 0+0i) et son destructeur. Toutes ces méthodes doivent afficher un message indiquant qu'elles sont en cours d'exécution.

Thème 2 : complexes et constructeurs

Sans utiliser les surcharges d'opérateurs, ajoutez à la classe complexe les méthodes addition, soustraction, multiplication, division, module, saisie, affichage. Utilisez ces méthodes dans le programme principal.

Constructeur par recopie

Ajoutez un constructeur par recopie affichant un message puis exécutez de nouveau le programme principal. Que constatez-vous ?

Recopie et tableaux statiques

Créez une classe X dont un des membres est un tableau statique. Créez deux objets a et b de la classe X, en créant b à partir de a. Quel est le constructeur appelé pour créer b ? Modifiez un élément du tableau stocké dans a et affichez b. Qu'en déduisez-vous sur le constructeur utilisé pour créer b ?

Ecrivez votre propre version de ce constructeur et testez-la.

Thème 3 : new, new[], delete, delete[]

Recopie et tableaux dynamiques

Créez une classe Y dont un des membres est un tableau dynamique (donc un pointeur).

Créez deux objets e et f de la classe Y, en créant f à partir de e. Quel est le constructeur appelé pour créer f ? Modifiez un élément du tableau stocké dans e et affichez f. Qu'en déduisez-vous sur le constructeur utilisé pour créer f ?

Ecrivez votre propre version de ce constructeur et testez-la. N'oubliez pas d'utiliser les opérateurs d'allocation et de libération dans les constructeurs et destructeurs !

Un destructeur loquace

Créez une classe D dont les constructeurs et le destructeur s'annoncent en écrivant un message. Ecrivez un programme créant des objets de classe D ainsi qu'un ou des pointeurs sur des objets de classe D. Quand les destructions d'objets ont-elles lieu ? Trouvez un moyen de pouvoir visualiser tous les messages affichés par le destructeur. Y a-t-il toujours autant de destruction d'objets que de création d'objets ?

Tableau 2D

Créez une classe T stockant un tableau dynamique 2D comportant l lignes et c colonnes. Les éléments de ce tableau doivent être des objets d'une autre classe O. Quand les constructeurs et les destructeurs de O sont-ils appelés ?

Opérateur delete et void*

En réutilisant la classe Y comportant un tableau dynamique, créez deux pointeurs a et b dans un programme principal, a étant de type Y* et de type `void*`. Les constructeurs et destructeurs de Y doivent faire des affichages pour indiquer qu'ils sont en cours d'exécution.

Initialisez a et b en utilisant l'opérateur `new`, puis libérez a et b grâce à l'opérateur `delete`.

Que constatez-vous ?

Thème 4 : surcharge des opérateurs

En reprenant la classe `complex`, surchargez les opérateurs suivants : `+, -, *, /, <<, >>`

Surchargez l'opérateur d'affectation = et testez tous ces opérateurs dans un programme principal. Que peut-on dire maintenant de la classe complexe ?

Créez une classe `Vector` dont un des membres est un tableau dynamique d'entiers.

Surchargez les opérateurs `+`, `-`, `*`, `<<`, `>>` et =. Testez l'opérateur d'affectation en effectuant une auto-affectation dans le programme principal.

Thème 5 : composition et initialisations d'objets, membres statiques

Ecrivez un programme avec deux classes nommées A et B. Parmi les membres de la classe B doit se trouver un objet de la classe A nommé a. Ecrivez le constructeur de la classe B, trouvez plusieurs manières d'initialiser le membre a.

Ecrivez une classe Sess qui attribue un numéro différent (et croissant) à chaque objet de cette classe. A chaque fois qu'un objet de cette classe est créé ou détruit, afficher le nombre d'objets de cette classe qui existent.

Ecrivez un programme avec deux classes T et U. Parmi les membres de la classe T doit se trouver un pointeur vers un objet de la classe U. Ecrivez tous les constructeurs et destructeurs des classes T et U, testez-les dans un programme principal.

Thème de Synthèse : réalisation d'une table de hashage en C++

Etape1 : chaînes de caractère

Ecrivez une classe Nstring permettant de gérer simplement les chaînes de caractère. Les membres de cette classe sont :

```
char *txt;
long length;
```

Il vous faudra, pour cette classe, implémenter toutes les méthodes permettant de faire fonctionner ce programme principal :

```
cout << "test d'egalite des Nstring :" << endl;
Nstring s1, s2;
cout << s1 << "==" << s2 << ": " << (s1==s2) << endl;
Nstring s3("toto");
cout << s3 << "==" << s1 << ": " << (s1==s3) << endl;
Nstring s4(s3);
cout << s3 << "==" << s4 << ": " << (s4==s3) << endl;
s2="toto";
cout << s3 << "==" << s2 << ": " << (s2==s3) << endl;
s3.setCarAt(2,'F');
cout << s3 << "==" << s2 << ": " << (s2==s3) << endl;
Nstring s5;
s5="toto2";
s5=s5;
cout << s5 << "==" << s2 << ": " << (s2==s5) << endl;
```

dont l'exécution donne :

```
test d'egalite des Nstring :
[vide]==[vide]: 1
toto==[vide]: 0
toto==toto: 1
toto==toto: 1
toFo==toto: 0
```

Etape 2 : liste chaînée

Ecrivez les classes Cell et List, correspondant respectivement à une cellule et à une liste chaînée. La classe List doit comporter les méthodes :

```
List();
List(const Cell &);
List(const List &);
~List();
friend std::ostream & operator<<(std::ostream &, const List &);
void chaineTete(Cell &);
void chaineQueue(Cell &);
bool contains(const Cell &) const;
bool isEmpty() const;
Cell get(const Nstring) const;
Cell *getHead() const;
```

La classe Cell doit comporter les méthodes :

```
Cell();
Cell(Nstring, Nstring);
Cell(const Cell &);
~Cell();
Nstring getKey() const;
Nstring getValue() const;
Cell *getNext() const;
void setNext(Cell *);
void setKey(const Nstring &);
void setValue(const Nstring &);
friend std::ostream& operator<<(std::ostream &, const Cell &);
bool operator==(const Cell &);
Cell & operator=(const Cell &);
```

Chaque cellule contient entre autres deux chaînes de caractère : une clé et une valeur.

Etape 3 : Construction de la table

On cherche à réaliser une table de hashage permettant de stocker des chaînes de caractère en utilisant également des chaînes de caractère comme clés de hashage.

Pour l'exemple, nous utiliserons une table stockant les mois de l'année, auxquels on accède en donnant leur numéro sous forme de chaîne de caractères. Les associations seront donc :

Clé :"un" Û valeur : "Janvier"

Clé : "deux" Û valeur : "Février"

…
Clé : "douze" Û valeur : "Décembre"

La table de hashage sera sous la forme d'un tableau de listes chaînées, afin de réaliser un hashage interne. Ce tableau aura une taille maximum de 12.

Ecrivez la classe Hashtable qui fournit les méthodes :

```
Hashtable();
Hashtable(Nstring);
~Hashtable();
hashCode();
put();
put(,);
get();
friend std::ostream& operator<<(std::ostream &, const Hashtable
&);
```

Il vous appartient de déterminer les paramètres et les types de retour de ces méthodes.