Sujet 2 : L'objet simple et la surcharge des opérateurs

Buts : Notions de classes, constructeurs, destructeur, méthodes, durée de vie des variables, compilation séparée, surcharge des opérateurs, gestion d'exceptions, *std : :vector*.

Exercice 1

L'objectif de cet exercice sera de se familiariser avec les spécificités du C++ dans le domaine de la programmation objet. Nous travaillerons dans un premier temps sans appliquer le principe de la compilation séparée, cet aspect faisant l'objet de la question 9.

Dans cet exercice nous développerons l'objet `Personne`. Une personne est caractérisée par son `nom`, son `prenom` ainsi que son `age`.

Q1. Définissez la classe `Personne`, déclarez y les attributs d'instance de cette classe.

Q2. On souhaiterait pouvoir déclarer un objet de type `Personne` comme suit :

- `Personne p1 ( "Dupont" , "Gaston" , 36 );`
- `Personne p2;`
- `Personne p3(p1);`

Modifiez le programme afin de le permettre. Complétez la classe d'une méthode `affiche` puis testez les fonctionalité développées dans la fonction `main`.

Q3. Complétez la classe `Personne` avec les accesseurs sur ses attributs d'instance.

Q4. Complétez la classe `Personne` avec les modificateurs de ses attributs d'instance.

Q5. On considère la fonction suivante :

```
bool compare ( Personne P1, Personne P2 )
{
    if ( P1.getNom() == P2.getNom() &&
              P1.getPrenom() == P2.getPrenom() &&
              P1.getAge() == P2.getAge() )
        return true;
    return false;
}
```

Que réalise cette fonction ? (Il vous est demandé de l'intéger dans votre code).

Q6. On souhaite contrôler la création d'une nouvelle `Personne`. On peut pour ceci ajouter l'affichage suivant dans le premier constructeur :

```
    cout << "Personne( " << NO ;
    cout << " , " << PR << " , " <<A<< " ) : " << this << endl;
```

Rajoutez le même type d'affichage (affichages **spécifiques** à chacun des constructeurs) dans les deux autres constructeurs. Que réalise finalement la fonction de la question 5 ?

Q7. On considère le destructeur de la classe `Personne` suivant :

```
~Personne()
{
    cout << "~Personne() : " << this << endl;
}
```

A quoi sert cette méthode ? Doit-on l'appeler de façon explicite ? Doit-on obligatoirement la définir dans la classe `Personne`.

Q8. Il existe entre autres deux façons d'afficher une Personne : définir une fonction externe à la classe ou définir une méthode à l'intérieur de la classe (préalablement définie). Définissez en complément la fonction.

Q9. En utilisant le principe de la compilation séparée, modifiez votre programme afin de travailler dans 3 fichiers :
- $Personne.h$ (définition de la classe),
- $Personne.cc$ (définition des méthodes de la classe),
- $principal.cc$ (fonctions externes à la classe et fonction `main`).

Afin de faciliter la compilation écrivez le fichier `makefile` correspondant à ce sujet.

Exercice 2

L'objectif de cet exercice est de simuler l'organisation d'un déménagement, et plus particulièrement le remplissage *intelligent* des cartons. Les remplissage des cartons est soumis à trois contraintes principales :
- le poids total des objets dans un carton ne pourra dépasser un poids maximal fixé ;
- le volume total des objets dans un carton ne pourra dépasser un volume maximal fixé ;
- un carton pourra contenir au maximum 10 objets.

Afin de pouvoir proposer une solution à ce problème, on se propose de travailler en deux temps :
- modéliser une classe `Objet` ;
- modéliser une classe `Carton`.

Vous travaillerez dans les 5 fichiers suivants :
- `Objet.h` (définition de la classe `Objet`),
- `Objet.cc` (définition des méthodes de la classe `Objet`),
- `Carton.h` (définition de la classe `Carton`),
- `Carton.cc` (définition des méthodes de la classe `Carton`),
- `Principal.cc` (fonction `main`).

Q0. Écrivez le fichier `makefile` correspondant à ces fichiers.

Un jeu de tests *pertinent* devra être proposé *progressivement* pour tester l'*ensemble* des fonctionnalités développées dans ce sujet.

La classe Objet

Un `Objet` est caractérisé par son `nom`, son `volume` (en dm^3 = 0.001 m^3) et son `poids` (en dg = 10 g).

Objet
`std::string nom`
`int volume`
`int poids`

Q1. Définissez la classe `Objet` en respectant le principe d'encapsulation des données. Écrivez les méthodes suivantes :

- le constructeur par défaut (il initialisera le premier attribut à `""`, les deux autres à 0),
- le constructeur par copie, le destructeur et l'opérateur = uniquement si ceci est nécessaire,
- les accesseurs aux attributs d'instance `volume` et `poids`,
- les modificateurs des attributs d'instance `nom` et `poids`.

Q2. On souhaite disposer du constructeur avec arguments dans la classe `Objet` suivant :

```
Objet(const std::string&, int = 0, int = 0) throw(std::invalid_argument);
```

Ce constructeur prend en paramètres les valeurs permettant d'initialiser la totalité des attributs d'instance de la classe `Objet`. Les valeurs des deux attributs d'instance numériques seront initialisés par défaut à 0. Si la valeur fournie en paramètre leur correspondant est négative une exception de type `std::invalid_argument` sera lancée (`#include <stdexcept>`). Donnez la définition de ce constructeur.

Est-il possible de modifier ce constructeur pour qu'il remplace le constructeur par défaut ? Si oui, faites cette modification et mettez en commentaire le constructeur par défaut précédemment codé.

Q3. On souhaite disposer d'une méthode `estVide` vérifiant si un `Objet` est vide. Un `Objet` est considéré comme étant vide si au moins une des conditions suivante est vérifiée : son nom est une chaîne vide ; son `poids` est nul ; son `volume` est nul. Cette méthode retournera vrai si l'`Objet` est vide, faux sinon.

Q4. Afin de pouvoir afficher un `Objet obj` on souhaite utiliser la syntaxe suivante :

```
cout « obj « endl ;
```

Définissez la méthode `afficher` de prototype :

```
std::ostream& afficher(std::ostream&) const;
```

Surchargez l'opérateur « en utilisant cette méthode.

Q5. Afin de pouvoir saisir les caractéristiques d'un `Objet obj` au clavier on souhaite utiliser la syntaxe suivante :

```
cin » obj ;
```

Définissez la méthode `saisir` de prototype :

```
std::istream& saisir(std::istream&);
```

Surchargez l'opérateur » en utilisant cette méthode.

Q6. Afin de pouvoir comparer deux instances d'`Objet`, surchargez l'opérateur ==. Cet opérateur teste l'égalité de l'ensemble des attributs d'instance.

La classe Carton

Un `Carton` peut contenir un nombre maximum `max_Objets` d'instances d'`Objet`. Ce nombre est identique et partagé par tous les cartons. Sa valeur est fixée à 10. Chaque carton a sa capacité maximale de `volumeMax` dm^3 et son poids maximal de `poidsMax` dg. À un moment donné, un `Carton` contiendra un nombre `nbObjets` instances d'`Objet` (avec `nbObjets<=max_Objets`) occupant un volume `volumeReel` dm^3 pour un poids `poidsReel` dg.

Voici une définition incomplète de la classe `Carton` :

```
1  class Carton
2  {
3      private:
4        static const int max_Objets;
5        unsigned int nbObjets;
6        int volumeMax;
7        int volumeReel;
8        int poidsMax;
9        int poidsReel;
10       std::vector<Objet> contenu;
11     public:
12       Carton(int, int) throw (std::invalid_argument);
13 };
```

L'attribut `contenu` est un tableau (`std::vector`) d'instances d'`Objet` dont la dimension *constante* est `max_Objets`. Les objets réellement présents dans le `Carton` seront stockés en tête de ce tableau, les autres éléments du tableau étant des instances d'`Objet` « vides ».
Note : une fiche de rappel sur `std::vector` est disponible en fin de sujet.

Q7. Pourquoi l'attribut `max_Objets` est-il déclaré `const` ? Pourquoi est-il déclaré `static` ? Où doit-il être défini et initialisé à 10 ?

Q8. Écrivez la définition du constructeur avec arguments qui prend en paramètres les valeurs permettant d'initialiser les attributs `volumeMax` et `poidsMax`. Si l'une des ces valeurs est négative ou nulle une exception sera lancée avec un message explicite. Ce constructeur initialisera *entre autres* le `contenu` du `Carton` à `max_Objets` instances d'`Objet` vides et `nbObjets` à 0.

Q9. Est-il nécessaire de définir explicitement dans la classe `Carton` le constructeur par copie, le destructeur et l'opérateur = ? Pourquoi ? Même si ce n'est pas nécessaire, donnez la définition de l'opérateur =.

Q10. Afin de permettre la recherche d'un `Objet` dans un `Carton` on décide de compléter la classe `Carton` avec la méthode :

```
int contient(const Objet&) const;
```

Cette méthode retourne l'indice de l'`Objet` recherché dans le `contenu` du `Carton` s'il y est présent et -1 sinon.

Q11. Afin de permettre le remplissage d'un carton on décide de compléter la classe `Carton` avec la méthode :

```
void ajouteObjet(const Objet&) throw (std::invalid_argument);
```

Cette méthode ajoute l'`Objet` passé en paramètre au `contenu` du `carton` si :

- l'`Objet` n'est pas vide,
- le nouveau `poidsReel` du `Carton` n'excèderait pas le `poidsMax`,
- le nouveau `volumeReel` du `Carton` n'excèderait pas le `volumeMax`,
- `nbObjets` n'est pas déjà égal à `max_Objets`.

Si l'une de ces quatre conditions n'est pas vérifiée la méthode `ajouteObjet` lance une exception indiquant la cause de l'erreur.

Cette méthode peut être remplacée par la surcharge de l'opérateur += prenant en paramètre l'`Objet` à ajouter au `Carton`. Celui-ci lancera une exception dans de manière similaire à la méthode `ajouteObjet`. Ajoutez cet opérateur +=.

Q12. On souhaite pouvoir accéder avec l'opérateur `[]` à un `Objet` dont on connaît la position (l'indice) dans le `contenu` d'un `Carton` afin d'afficher ses caractéristiques. Cet opérateur ne devra pas pouvoir autoriser la modification du contenu d'un carton. Ajoutez le code nécessaire à la classe en vous assurant qu'il autorise la compilation du code suivant (à ajouter dans la fonction `main`) :

```
1  Carton C(1000,1000);
2  C += Objet("Lampe", 5 , 50);
3  C[0].afficher(cout);
4  C.afficher(cout); // méthode fournie ci-dessous
```

Attention, aucune hypothèse n'est faite sur l'indice et donc il faudra lancer une exception si on essaie d'accéder en dehors du `contenu` réel du `Carton`.

La méthode `Carton::afficher` est la suivante :

```
1 ostream & Carton :: afficher (ostream & o) const
2 {
3     o << endl << "Contenu du carton : " << endl;
4     for (int i=0; i < nbObjets ; i++)
5         o << i+1 << " : " << (*this)[i] << endl ;
6     return o;
7 }
```

À l'aide de cette méthode surchargez l'écriture d'un `Carton` dans un flot (`ostream`) par l'opérateur «.

Q13. Afin de permettre le déballage d'un carton on décide d'ajouter à la classe `Carton` l'opérateur −=. Cet opérateur prend en paramètre l'`Objet` recherché. Il permet d'enlever l'`Objet` recherché (première itération) au `contenu` à condition que le `Carton` le contienne. Dans le cas contraire une exception de type `std::invalid_argument` sera lancée.

Autour de l'objet *std : :vector*

Inclusion

```
#include <vector>
```

Création

`vector< type > Nom_Tableau ;`
Création d'un `vector` vide (de taille 0).

`vector< type > Nom_Tableau ( taille );`
Création d'un `vector` de `taille` éléments initialisés avec le constructeur par défaut de `type`.

`vector< type > Nom_Tableau ( dimension , valeur );`
Création d'un `vector` de `taille` éléments initialisés par copie de `valeur`.

Accès à un élément

`Nom_Tableau [ i ];`
Permet d'accéder à l'élément d'indice `i` du `vector`.

Divers

La taille d'un `vector` est accessible en utilisant la méthode `size()` :

```
int taille = Nom_Tableau.size() ;
```