Travaux Dirigés de C++ n°6

Cours de programmation

—Licence d'informatique—

L'héritage

Après avoir détaillé les différentes extensions du C++ par rapport au C, nous abordons maintenant la notion d'héritage entre classes.

▸ **Exercice 1.** *(**Points 2D et Points 3D**) Reprendre la classe* `Point2D`*, déjà vue sous le nom* `Point`*, définie par deux champs de type* `double` *pour ses coordonnées. Cette classe définit un constructeur par défaut, un constructeur de copie et un constructeur avec les deux coordonnées, ainsi que les méthodes* `translate` *et* `print` *qui, respectivement, déplacent et affichent une représentation d'un objet de la classe.*

Écrire une classe `Point3D` *qui hérite de la classe* `Point` *et ajoute un champ* `z` *de type* `double`*. Définir pour cette classe un constructeur par défaut, un constructeur par copie et un constructeur avec les trois coordonnées. Définir également les méthodes* `translate` *et* `print` *propres à la classe* `Point3D`*.*

Comment obtenir un affichage "2D" d'un objet "3D" ? Depuis le `main()` *? À partir d'une méthode ?*

Quelles affectations sont possibles entre des variables déclarées de type `Point2D` *et* `Point3D` *?*

Même question pour des variables déclarées de type pointeur vers `Point2D` *et pointeur vers* `Point2D` *?*

Ajouter dans chacune des classes une méthode `is2D` *qui renvoit* `true` *si l'objet interrogé est un* `Point2D` *et* `false` *si c'est un* `Point3D`*.*

▸ **Exercice 2.** *(**Expressions arithmétique**)*

On désire manipuler des expressions arithmétiques sur des `double` *et, en particulier, on souhaite pouvoir les évaluer.*

Définir la classe purement abstraite `Expr` *et ses méthodes purement virtuelles* `double eval()` *et* `void print()` *que devront définir les classes non purement abstraites dérivées implantant les expressions arithmétiques.*

Définir la hierarchie des classes d'expressions. On pourra considérer dans un premier temps que :

- *le type des expressions générales est représenté par une classe purement abstraite* `Expr`*, définissant les méthodes purement virtuelles* `double eval()` *et* `void print()`*;*

- *une variable est représentée par une classe* `Variable`*, qui hérite de* `Expr` *(puisqu'une variable est une expression) et qui définit deux champs :* `name` *de type* `string` *et* `value` *de type* `double`*. Cette classe doit disposer d'un constructeur et fournir l'implantation des deux méthodes déclarées dans* `Expr`*;*

- *un opérateur binaire est représenté par une classe* `BinOp` *qui hérite de* `Expr` *(puisqu'un opérateur binaire muni de ses opérandes est une expression) et qui définit trois champs (un champ* `sign` *de type* `string` *représentant le signe de l'opérateur et deux champs de type pointeur sur* `Expr` *représentant les opérandes gauche et droite de l'opérateur. Cette classe doit contenir un constructeur qui initialise chacun de ces champs. La classe* `BinOp` *doit également être capable de fournir l'implantation de la méthode* `print` *pour toute expression construite comme un opérateur binaire. En revanche, la méthode* `eval` *ne peut pas être implantée dans cette classe ;*

- *les opérateurs binaires "concrets" habituels (*`Plus`*,* `Div`*,* `Mult`*, etc...) sont représentés par des classes héritant de la classe* `BinOp`*. Ils définissent chacun l'implantation de la méthode* `eval` *pour l'opérateur correspondant et disposent chacun d'un constructeur qui permet de créer des expressions (*`new Mult(operand1, operand2)`*);*

À partir de cette hierarchie de classes, il doit être possible de créer des expression, de les afficher et de les évaluer. Que pensez vous de l'affichage obtenu.

Revoir la définition de la méthode `print` *afin d'ajouter des parenthèses dans la représentation des expressions, mais seulement là ou c'est indispensable (en fonction des priorités des opérateurs). Pour cela, on pourra utiliser un champ entier représentant la priorité de chaque opérateur.*

Modifier votre hierarchie de classes pour pouvoir accepter des opérateurs unaires.