Langage C++

- CIR2 -

TP2 : Création des classes en C++

Nils Beaussé (nils.beausse@isen-ouest.yncrea.fr)

Exercice 1

Partie 1 : méthodes et attributs publiques

Créer une classe « **Equation** » qui contient des attributs et des méthodes **publiques** permettant de résoudre une équation de second degré $ax^2 + bx + c$

On se restreint aux solutions (racines) réelles de cette équation.

- Les attributs de la classe sont les coefficients de l'équation (a, b et c)
- Les méthodes de la classe sont :
    - une méthode pour saisir les coefficients
    - une méthode de résolution de l'équation
    - une méthode pour l'affichage de l'équation :
      Exemple: 2x^2 + 5x + 8 = 0
    - une méthode pour le calcul de l'image d'une variable par cette équation (par ce polynôme).
      Cette dernière méthode permet également de vérifier si les racines trouvées par la méthode de résolution de l'équation sont correctes

Consignes :

- Définir la classe (attributs et prototypes des méthodes décrites ci-dessous) dans un fichier « .h » (ou hpp)
- Définir les méthodes dans un fichier « .cpp »
- Pour la racine, utiliser la fonction `sqrt` de la bibliothèque `cmath`

Partie 2 : restriction d'accès aux attributs

Modifier la classe "**Equation**" pour restreindre l'accès aux attributs. Le but va être de s'assurer que l'équation est bien du second degré en vérifiant que la valeur de "a" est non nulle dans l'équation : $ax^2 + bx + c$

Pour cela, ajouter des accesseurs et des mutateurs dans la classe "**Equation**".

Partie 3 : test des fonctionnalités de la classe

Tester les différentes fonctionnalités de la classe "**Equation**" pour la partie 1 et la partie 2 sachant que les valeurs de "a", "b" et "c" sont saisies par l'utilisateur.

N'oubliez pas d'inclure le fichier ".h" dans le fichier "main.cpp".

Exercice 2

Partie 1 : méthodes et attributs publics

Créer une classe "**Point**" qui encapsule :

- les abscisses et les ordonnées (x,y) comme attributs publics
- une méthode de calcul de distance entre deux points (le point courant de la classe et un autre point) → publique

```
double distance(Point P)
```

Pour le calcul de la racine utiliser la bibliothèque `#include <cmath>`

  - Notez qu'en C++ la librairie mathématique libm, que vous aviez l'habitude d'inclure en C grâce à l'option « -lm » lors de la compilation, est ici incluse de base dans le C++.

- une méthode de calcul du milieu du segment composé de deux points (le point courant de la classe et un autre point) → publique

```
Point milieu(Point P)
```

Partie 2 : test du fonctionnement de la classe en utilisant une allocation statique

Le résultat après l'exécution du programme est :

```
**SAISIE DU POINT A**
Tapez l'abscisse : 3.2
Tapez l'ordonnée : 1.4
L'abscisse vaut : 3.2
L'ordonnée vaut : 1.4

**SAISIE DU POINT B**
Tapez l'abscisse : 5
Tapez l'ordonnée : 6
L'abscisse vaut : 5
L'ordonnée vaut : 6

**MILIEU DE AB**
L'abscisse vaut : 4.1
L'ordonnée vaut : 3.7

** La distance AB vaut : 4.93964
```

Partie 3 : restriction d'accès aux attributs

Modifier la classe "**Point**" en :

- déclarant les attributs comme privés
- ajoutant les méthodes publiques suivantes :

```
void setX(double x); # le nom de l'argument est le même que l'attribut de la classe
void setY(double y); # le nom de l'argument est le même que l'attribut de la classe

double getX(); # pour récupérer X
double getY(); # pour récupérer Y

void saisir(); # pour permettre à l'utilisateur de saisir les coordonnées des points
void afficher(); # pour afficher les coordonnées des points
```

Partie 4 : Instanciation des objets via une allocation dynamique

1. Créer une nouvelle classe "**Point1**" en suivant les instructions suivantes :
    a. La déclaration des attributs (x et y) de la classes "**Point**" sera la même que dans la classe "**Point1**" (déclarer-les comme privés)

b. La création des objets dans le "**main()**" sera, cette fois-ci, dynamique. En prenant ce constat en considération, proposer des nouvelles versions des prototypes et des définitions des fonctions développées dans les parties 2 et 3.
c. N'oublier pas de supprimer les instances créées dynamiquement avec `delete`

Exercice 3 :

Proposer une classe "**JeuMorpion**" qui permet, via ses attributs et ses méthodes, de manipuler le jeu tic-tac-toe (le jeu de morpion)

Ci-dessous une description incomplète de cette classe :

```
#ifndef EXSUPP_JEUMORPION_H
#define EXSUPP_JEUMORPION_H

enum Content {VIDE, ROND, CROIX};

class JeuMorpion {
private:
   int grille[3][3];
   bool placer_coup(int ligne, int colonne, Content c);
public :
   void init();
   bool placer_rond(int ligne, int colonne);
   bool placer_croix(int ligne, int colonne);
   // à compléter
};


#endif //EXSUPP_JEUMORPION_H
```

Pensez à faire l'affichage du jeu !

Exercice 4 :

Construire :

1. Une classe représentant une arme et disposant d'un nom (par exemple « épée ») et de dégâts.

2. Une classe représentant un personnage et disposant de point de vie ainsi que d'une arme et d'une position en X et en Y
3. Des accesseurs et mutateurs pour chaque cas, avec gestion des erreurs (affichage de l'erreur grâce aux cin et cout et gestion de l'erreur ensuite selon les cas).
4. Une fonction capable d'afficher un tableau de 20*20 cases. Les cases seront représentées par des carrés vides et la fonction sera capable d'afficher un X et un O à deux emplacements passés en paramètres (avec des paramètres par défauts qui n'affiche ni X ni O).
5. Faire fonctionner cet ensemble de classes et fonctions pour que, en fonction de la touche pressée par l'utilisateur, les deux personnages puissent être déplacée dans le tableau. (on réaffiche le tableau à chaque fois)
6. En cas de rencontre avec les personnages on utilisera une méthode de la classe personnage qui est capable de prendre en argument un autre personnage et de lui retirer des points de vie en fonction de la valeur d'attaque de l'arme : le personnage n'est alors pas déplacé. Si les PV d'un personnage atteignent 0 il meurt et le programme s'arrête et déclarant le gagnant.