TP 6 : compilation séparée, opérateurs et généricité

Toutes les classes dont il est question dans ce TP devront être de Coplien, être documentées comme expliqué au TP 2 et proposer des méthodes *getName()* et *Print()* telles que définies dans les TP précédents. Il va de soi que vous mettez à jour la fonction *main()* afin de tester les nouvelles fonctionnalités au fur et à mesure des questions.

Exercice 6a : Repartez de la situation de la fin du TP 5 et réorganisez le code afin créer un fichier *.h* et un fichier *.cpp* par classe, dont le nom sera le même que celui de la classe, ainsi qu'un fichier *main.cpp* qui ne définira que la fonction *main()*. Créez un *Makefile* afin de faciliter la compilation du projet ainsi structuré. Le *Makefile* doit également comporter un objectif clean permettant de nettoyer le répertoire des fichiers objet, exécutable et sauvegardes de sécurité de l'éditeur par exemple. Optimisez le *Makefile* afin d'utiliser les variables et règles automatiques.

Exercice 6b : Ajoutez les opérateurs unaires ++ et -- à la classe *ImplicitShape* afin de pouvoir respectivement doubler et diviser par 2 le rayon du cercle, en version préfixée (*++instance*) et postfixée (*instance++*), avec la différence de sémantique habituelle évidemment. Que remarquez-vous sur la sortie ? Réfléchissez jusqu'à ce que vous ayiez compris ce qui se passe.
La fonction *main()* doit être celle-ci :

```
int
main(int argc, char *argv[])
{
  // on cree une premiere forme de type implicite
  ImplicitShape sh1(8, 8, 5);
  sh1.Print();
  sh1.PrintSceneBox();
  cout << " \n";
  sh1++;
  sh1.Print();
  sh1.PrintSceneBox();
  cout << " \n";
  --sh1;
  sh1.Print();
  sh1.PrintSceneBox();
  cout << " \n";
  // puis une seconde forme de type explicite
  vector<t_point> pts;
  pts.push_back({2, 8});
  pts.push_back({5, 16});
  pts.push_back({1, 11});
  // on teste le constructeur par copie
  PointSet sh2(pts);
  sh2.Print();
  sh2.PrintSceneBox();
  cout << " \n";
  // on cree un nouveau point afin de verifier que la taille de la scene est correctement mise a jour
  pts[1].x = 15;
  pts[1].y = 2;
  PointSet sh3(pts);
  sh3.Print();
  sh3.PrintSceneBox();
  cout << " \n";
  return 0;
}
```

Exercice 6c : Créer un type structuré *t_color* qui contient 3 *unsigned int* codant les proportions d'une couleur dans le système à 3 couleurs fondamentales RGB (Red Green Blue). Créer une classe *Color* permettant de représenter une couleur au format RGB, avec un constructeur admettant 3 paramètres optionnels *r*, *g* et *b*. Par défaut la couleur est le noir (valeur 0 0 0). Ajoutez des accesseur et mutateur afin d'accéder à la variable interne de type *t_color*.

Exercice 6d : Surchargez la méthode << de la classe *ostream* afin que l'on puisse afficher le contenu de chaque classe en écrivant *cout* << *instance* au lieu de *instance.Print()*. Et ce pour toutes les classes. Pour la classe *ImplicitShape* cette méthode doit afficher le centre et le rayon du cercle.
La fonction *main()* doit être celle-ci :

```
int
main(int argc, char *argv[])
{
  // on cree une premiere forme de type implicite
  ImplicitShape sh1(8, 8, 5);
  cout << sh1 << endl;
  sh1++;
  cout << sh1 << endl;
  --sh1;
  cout << sh1 << endl;
  // puis une seconde forme de type explicite
  vector<t_point> pts;
  pts.push_back({2, 8});
  pts.push_back({5, 16});
  pts.push_back({1, 11});
  // on teste le constructeur par copie
  PointSet sh2(pts);
  cout << sh2 << endl;
  // on cree un nouveau point afin de verifier que la taille de la scene est correctement mise a jour
  pts[1].x = 15;
  pts[1].y = 2;
  PointSet sh3(pts);
  cout << sh3 << endl;
  // on cree 2 couleurs
  Color black, red(255, 0, 0);
  cout << black;
  cout << red << endl;
  return 0;
}
```

Sortie attendue :

```
MacBook-Pro-8:Ex6d Wilfrid$ ./a.out
Shape::Shape(0,0,1,1)
Shape::Shape()
ImplicitShape::ImplicitShape(8,8,5)
ImplicitShape:
  Scene box: 3x3  10x10
  Shape box: 3x3  10x10
  center: (8, 8)
  radius: 5

ImplicitShape::operator++(int)
Shape::Shape(0,0,1,1)
Shape::Shape()
ImplicitShape::ImplicitShape(ImplicitShape&)
ImplicitShape::~ImplicitShape()
Shape::~Shape()
ImplicitShape:
  Scene box: -2x-2  20x20
```

```
  Shape box: -2x-2  20x20
  center: (8, 8)
  radius: 10

ImplicitShape::operator--()
ImplicitShape:
  Scene box: 3x3  10x10
  Shape box: 3x3  10x10
  center: (8, 8)
  radius: 5

Shape::Shape(0,0,1,1)
Shape::Shape()
PointSet::PointSet(vector<3 t_point>)
PointSet:
  Scene box: 1x8  4x8
  Shape box: 1x8  4x8
  Vertices: (2, 8) (5, 16) (1, 11)

Shape::Shape(0,0,1,1)
Shape::Shape()
PointSet::PointSet(vector<3 t_point>)
PointSet:
  Scene box: 1x2  14x9
  Shape box: 1x2  14x9
  Vertices: (2, 8) (15, 2) (1, 11)

Color: (0, 0, 0)
Color: (255, 0, 0)

PointSet::~PointSet()
Shape::~Shape()
PointSet::~PointSet()
Shape::~Shape()
ImplicitShape::~ImplicitShape()
Shape::~Shape()
```

Exercice 6e : Dans la fonction *main()* créez une fonction template *Average()* qui renvoie la moyenne de 2 valeurs d'un type quelconque. Testez la fonction en traitant les 3 fondamentales séparément puis modifiez le type *t_color* afin que la méthode puisse s'appliquer à un type *t_color*. Pourquoi n'est-il pas possible d'écrire *Color cyan(Average(green, blue))* ? (tip: essayez et voyez ce que dit le compilateur)

La fonction *main()* doit être celle-ci :

```
int
main(int argc, char *argv[])
{
  […]
  // les couleurs
  Color black, red(255, 0, 0), green(0, 255, 0), blue(0, 0, 255);
  Color yellow(
                Average(red.getRgb().r, green.getRgb().r),
                Average(red.getRgb().g, green.getRgb().g),
                Average(red.getRgb().b, green.getRgb().b)
               );
  Color cyan(Average(green.getRgb(), blue.getRgb()));
  cout << red;
  cout << green;
  cout << blue;
  cout << cyan;
```

```
  cout << yellow << endl;
  return 0;
 }
```

Sortie attendue :

```
MacBook-Pro-8:ex6e Wilfrid$ ./a.out
[…]
Color: (255, 0, 0)
Color: (0, 255, 0)
Color: (0, 0, 255)
Color: (0, 127.5, 127.5)
Color: (127.5, 127.5, 0)
```

Exercice 6f : Afin de pouvoir faire la moyenne de 2 couleurs sans devoir passer par leur type agrégé *t_color*, nous allons maintenant écrire une classe template *Mixer*, qui admettra un seul constructeur à 2 paramètres classes et fera la moyenne des 2 instances. On pourra récupérer l'instance moyenne par la méthode *getInstance()*. La classe *Mixer* utilise *ostream::<<()* afin d'écrire le contenu de l'instance moyenne. Pensez à faire en sorte qu'une modification du fichier *Mixer.hpp* provoque la compilation du fichier *main.cpp* !

La fonction *main()* doit être celle-ci :

```
int
main(int argc, char *argv[])
{
  […]
  // les couleurs
  Color black, red(255, 0, 0), green(0, 255, 0), blue(0, 0, 255);
  Color yellow(
                Average(red.getRgb().r, green.getRgb().r),
                Average(red.getRgb().g, green.getRgb().g),
                Average(red.getRgb().b, green.getRgb().b)
               );
  Color cyan(Average(green.getRgb(), blue.getRgb()));
  Mixer<Color> magenta(red, blue);
  cout << red;
  cout << green;
  cout << blue;
  cout << cyan;
  cout << yellow;
  cout << magenta << endl;
  return 0;
}
```

Sortie attendue :

```
MacBook-Pro-8:ex6f Wilfrid$ ./a.out
[…]
Color: (255, 0, 0)
Color: (0, 255, 0)
Color: (0, 0, 255)
Color: (0, 127.5, 127.5)
Color: (127.5, 127.5, 0)
Color: (127.5, 0, 127.5)
```