Laboratoire 5
Création d'objets

Vous devez créer un projet du nom de manipRectangle qui permettra de faire une manipulation de l'objet rectangle que vous ferez dans ce laboratoire. Ce programme saisiera les coordonnées, la largeur et la hauteur d'un rectangle et génèrera un nb. aléatoire de rectangles et une couleur aléatoire à l'écran pour créer des formes tel que une ligne, un triangle et une grille du nb de rectangle et de la couleur des nb. générés aléatoirement. Faîtes une fonction pour chaque forme et faites afficher toutes les formes une après l'autre, avec une pause entre chacune. Faites une boucle while(true) pour dessiner tant qu'on veut avec ce rectangle mais en changeant les nb. aléatoires chaque fois.

En programmation orientée objet, avant de commencer le programme principal, on se demande qu'est-ce que le programme manipule, de quoi ça parle et on en découle des principaux objets. Ici, c'est très simple, on parle de rectangle. Ensuite, on se demande qu'est-ce que le rectangle devra contenir? Et là, on parle de coordonnées, largeur et hauteur.

Donc, un **rectangle** est défini par les propriétés suivantes :

1. **Coordonnée en x (int)**
2. **Coordonnée en y (int)**
3. **Largeur (int)**
4. **Hauteur (int)**

On peut regrouper les coordonnées du rectangle et en faire un objet aussi, ce qui serait encore plus efficace que de manipuler toujours 2 entiers pour les coordonnées du rectangle.

Le rectangle sera donc défini par les propriétés suivantes :

5. **Coordonnee (x,y) (point)**
6. **Largeur (int)**
7. **Hauteur (int)**

Donc avant de créer la class rectangle, nous devons créer la class point :

1. Créer la classe point

```
class point{
	private :
		int _x, _y;		//coordonnées du point

};
void main(){
	point p1;			//instanciation d'un point
}
```

Quelques questions très importantes:

1. Pourquoi vous êtes capable de déclarer un point dans le main, sans même avoir programmer de méthodes constructeurs dans votre class?
2. Quelles sont les 4 méthodes qui existent sans que vous ayez à les programmer?
3. Que pouvez-vous faire de plus dans le main, sans programmer autre chose?
4. Que contiennent ces 4 méthodes par défaut (vous devez être en mesure de les définir : entête et corps)
5. Ajouter un constructeur avec 2 paramètres dans votre classe comme ci-dessous et tenter de compiler le main à présent avec seulement la déclaration du **point p1** à l'intérieur et expliquer pourquoi ça ne marche plus.

```
point(int x, int y){
	_x = x;
	_y = y;
}
```

- **Attention, le nom d'une classe** est toujours au **singulier**, car une class représente un modèle unique d'objet.
- Le nom des propriétés sont écrit en **minuscule** avec une **majuscule** à chaque mot comme toutes les variables (camelCase)
- Les **propriétés** doivent être **précédé du souligné** et être sans préfixe
- Le **nom des méthodes** doit être plus **bref** et représenter le **résultat** produit par la méthode et non un verbe action et le résultat. (print au lieu de printDate)
- Mettre une entête au début du .h comme les entêtes de programme pour expliquer l'objet
- Mettre un **commentaire** pour **chaque propriété** si son nom n'est pas assez explicite.
- Mettre un **commentaire** au dessus de **chaque méthode** dans le .cpp

1. Mettre la **déclaration** de la classe point dans un fichier d'entête **(point.h)** et la **définition** des méthodes dans un fichier sources **(point.cpp)**

2. Créer les méthodes

Pour le moment, si vous tentez d'accéder aux propriétés en faisant p1._x, vous verrez que c'est impossible. Pourquoi? C'est ce qu'on appelle **l'encapsulation**. Les propriétés étant privées, celles-ci sont inaccessibles directement de l'extérieur rendant l'objet sécuritaire aux manipulations incorrectes.

Nous devons donc programmer des méthodes pour manipuler ces propriétés. Il y a plusieurs types de méthodes dans chaque objet, selon l'objet lui-même, mais certaines sont toujours présentes.

2.1. Constructeurs : méthodes appelées automatiquement lorsqu'on déclare (instancie) un objet
2.2. Destructeur : méthode unique appelée automatiquement lors de la destruction du rectangle
2.3. Getteur/Accesseur : méthode permettant de recevoir la valeur d'une propriété
2.4. Setteur/Mutateur : méthode permettant de modifier la valeur d'une propriété
2.5. Autres méthodes utiles pour le point

2.1 Les constructeurs

Il existe un **constructeur sans paramètre** par défaut, qui est vide, ce qui vous a permis de déclarer un point dans le main à l'étape 1. Si vous regardez avec un point d'arrêt, vous verrez que x, y ne sont pas initialisé.

```
void main(){
	point p1;			//appel du constructeur par défaut vide
}
```

Ceci fait rarement notre affaire, car il laisse les propriétés non initialisées. Nous allons le reprogrammer.

```
class point{
    private :
	int _x, _y;		//coordonnées du point supérieur gauche
    public :
	point(){		//le nom de la méthode = tjrs nom de la classe
		_x = _y = 0;	//permet de nettoyer la mémoire
	}
};
```

Si nous voulons initialiser le point lors de la déclaration, mais avec des valeurs, comme nous avons déjà fait avec plusieurs autres variables, par exemple

```
string nom("bozo");		//appel du constructeur de string avec un char[]
ifstream entree("notes.txt");
point p2(2,3);			//appel du constructeur de point avec 2 int
```

Il peut y avoir autant de **constructeur avec paramètres** que l'objet en a besoin, tout dépend de la logique de l'objet. Pour le point, il est logique de le créer sans coordonnées, celui-ci est donc créée à l'origine et avec la position (x,y). On pourrait aussi recevoir un seul int et déterminé que le point attribut cet int au x et met 0 dans les y. Mais ce n'est pas très logique de prioriser x au lieu d'y. Créer donc 2 constructeurs, un sans paramètre et un avec 2 paramètres.

```
//constructeur de point avec une position
point::point(int x, int y)
{
    _x = x;
    _y = y;
}
```

Testez bien que tout fonctionne au fur et à mesure. Pour appeler ces méthodes, il suffit de déclarer un point sans valeur et un autre avec sa position (x,y)

```
void main()
{
    point p1;                  //appel du constructeur de point sans paramètre
    point p2(2,3);             //appel du constructeur de point avec 2 int
}
```

Qu'en est-il du **constructeur de copie?** Le copieur est une des 4 méthodes qui existe par défaut dans toute classe et si on n'a pas d'allocation dynamique dans notre objet, il fait toujours notre affaire. C'est de même pour l'affectateur. Pour apprendre à bien les implémenter, recodez les comme leur version par défaut.

```
//constructeur de copie par défaut
point::point(const point &p)
{
    _x = p._x;
    _y = p._y;
}
//affecteteur par défaut
const point& point::operator=(const point &p)
{
    _x = p._x;
    _y = p._y;
    return *this;
}
```

Quelles instructions appelle ces 2 méthodes? Il y en a 3 pour le copieur et 1 pour l'affectateur. Codez ces instructions et suivez la trace pour bien voir que votre programme va dans ces méthodes.

```
void main()
{
    point p1;           //appel du constructeur de point sans paramètre
    point p2(2,3);      //appel du constructeur de point avec 2 int
    point p3(p3);       //appel du constructeur de copie (1ère utilisation)

    p3 = fonct(p2);     //appel du copeur pour passer par valeur et return par valeur
    p1 = p2 = p3;       //appel de l'affectateur 2x et utilisation du point en return
}
point fonct(point p)
{
    p.setPosition(3,2);
    return p;
}
```

2.2 Le destructeur

Il existe toujours un **destructeur** par défaut qui est vide, pour ce type d'objet, il ne serait pas obligatoire de le programmer, mais pour se pratiquer, nous allons remettre les propriétés de notre rectangle à 0 pour effacer nos valeurs en mémoire.

```
void main()
{
    point p1;                  //appel du constructeur par défaut vide
}

//le destructeur de point est appelé à l'accolade fermante du main car c'est là
//que le compilateur tente de détruire le rectangle
```

Voici la méthode destructeur

```
class point{
    private :
           int _x, _y;         //coordonnées du point supérieur gauche
    public :
           ~point()            //le nom de la méthode = tjrs nom de la classe + ~
           {
                  _x = _y = 0; //permet de nettoyer la mémoire
           }
};
```

2.3 Les getteurs/accesseurs

C'est bien beau de pouvoir créer et détruire nos points, mais nous voulons pouvoir accéder aux valeurs des propriétés quand on veut, soit pour les manipuler ou pour les modifier.

Les getteurs/accesseurs sont des méthodes qui permettent d'accéder aux valeurs, c'est à dire que nous allons programmer des méthodes qui nous retournent les valeurs contenues dans chacune des propriétés. Avec les structures, tout était public, alors nous en avions pas besoin, mais avec les objets, c'est privé, alors nous devons accéder aux contenues par l'intermédiaire des méthodes.

```
void main()
{
    point p2(2,3);
    cout << "(" << p2.getX() << ", " << p2.getY() << ")";
}
```

Voici le prototype de la méthode getX. Coder la définition et faîtes de même pour getY ()

```
class point
{
    private :
           int _x, _y;         //coordonnées du point supérieur gauche
    public :
           int getX()const;    //retourne la coordonnée x
};
```

Pourquoi on ajoute un const après les parenthèses du prototype de la méthode?

Testez ces méthodes avec des instructions dans le main qui manipulent vos points

2.4 Les setteurs/mutateurs

Ces méthodes permettent de modifier les valeurs de nos propriétés. Nous voulons donc à différents moments modifier seulement le x, le y, ou les 2, alors ça nous prend minimalement 3 setteurs.

Voici le prototype de la méthode setX, coder sa définition et faîtes la méthode setY (), setPosition()

```
class point{
    private :
           int _x, _y;          //coordonnées du point supérieur gauche
    public :
           void setX(int x);    //modifie la coordonnée x
};
```

A-t-on besoin d'un assert dans nos setteurs de point ?

Testez bien ces 3 méthodes avec des instructions dans le main qui manipule vos points.

2.5 Autres méthodes intéressantes pour le point

Maintenant, voici les méthodes qui seraient utiles pour le point, codez-les

```
void main()
{
    point p1, p2;
    cout << "Entrer les coordonnées d'un point : ";
    p1.read(cin);       //saisie les coordonnées comme ceci: (x,y)

    cout << "Entrer les coordonnées d'un point : ";
    cin >> p2;          //fonction operator>> qui appel read

    cout << endl;
    p1.print(cout);     //on affiche les coordonnées comme ceci: (x,y)
    cout << endl << p2; //fonction operator<< qui appel print

    //surcharge de l'operator== qui compare 2 points
    if(p1 == p2)
           cout << endl << "les coordonnées sont identiques";
    else
           cout << endl << "les coordonnées ne sont pas identiques";

    //surcharger aussi l'operator!=

    cout << p1 + p2;    //surcharge de l'operator+ qui retourne un point par valeur
    cout << p1 - p2;    //surcharge de l'operator- qui retourne un point par valeur

    cout << distance(p1, p2);  //retourne la distance entre les 2 points
```

La distance entre les deux points (x_1, y_1) et (x_2, y_2) est égale à

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

```
}
```

Programmez et testez ces méthodes.

3. Créer une méthode qui permettra d'afficher votre point en console à sa position

```
r2.draw(cout);      //on dessine un point à la position (x,y) en console
```

3.1 Pour que afficher un point avec une forme au lieu d'une lettre, c'est très facile avec les codes ascii, on peut afficher tous les caractères de la table ascii à l'adresse ci-dessous en choisissant le code hexadécimal et en le précédent de `\x` dans les guillemets.

```
cout << "\xFE";     //on dessine un petit carré
```

http://www.table-ascii.com/

3.2 Si vous voulez changer la couleur de celui-ci, voici la fonction qui se trouve dans windows.h (on aurait pu mettre la couleur comme propriété du point et du rectangle)

```
SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),
      FOREGROUND_INTENSITY | FOREGROUND_RED | FOREGROUND_GREEN);
```

3.3 Voici une **fonction** qui vous permettra de vous positionner en x, y dans la console et de dessiner séquentiellement vos rectangles. Ajouter cette fonction à votre classe rectangle.

```
//fonction pour se positionner dans l'écran à x,y
void gotoxy(int xpos, int ypos){
      COORD scrn;
      HANDLE hOuput = GetStdHandle(STD_OUTPUT_HANDLE);
      scrn.X = xpos;
      scrn.Y = ypos;
      SetConsoleCursorPosition(hOuput, scrn);
}
```

Montrez moi ce premier objet ainsi que les tests complets pour le point dans votre main avant de continuer avec le 2e objet, le rectangle.

4. Créer la classe rectangle

Créer un nouveau fichier d'entête rectangle.h et un nouveau fichier source rectangle.cpp pour y insérer la déclaration de votre classe et les définitions de vos méthodes.

```
//déclaration de la classe
class rectangle
{
   private :
         point _coord;       //coordonnées du point supérieur gauche
         int _w;             //largeur
         int _h;             //hauteur
};
```

5. Créer les méthodes

Vous allez effacer toutes les instructions qui testent le point de votre main et vous testerez maintenant vos rectangles.

Créez les méthodes

- **5.1. Constructeurs : méthodes appelées automatiquement lorsqu'on déclare un rectangle**
- **5.2. Destructeur : méthode unique appelée automatiquement lors de la destruction du rectangle**
- **5.3. Getteur/Accesseur : méthode permettant de recevoir la valeur d'une propriété**
- **5.4. Setteur/Mutateur : méthode permettant de modifier la valeur d'une propriété**
- **5.5. Autres méthodes utiles pour les rectangles**

5.1 Les constructeurs

Nous allons créer un contructeur sans paramètre, un constructeur avec seulement une coordonnée en paramètre et un constructeur avec une coordonnée et les dimensions du rectangle en paramètre.

```
void main()
{
    rectangle r1;       //appel du constructeur par défaut qui itilisalise tout à 0
    rectangle r2(3,4);  //appel du constructeur avec une coordonnée qui met w et h à 0
    rectangle r2(3,4,10,10);   //appel du constructeur avec 4 paramètres
}
```

Le **constructeur sans paramètre** n'a qu'à initialiser la largeur et la hauteur à 0, car le point est construit automatiquement à 0 grâce au constructeur sans paramètre créée dans la class point.

Les **constructeurs avec 2 et 4 paramètres** peuvent être le même en utilisant l'initialisation des paramètres par défaut. Quand les derniers paramètres peuvent avoir une valeur par défaut, c'est-à-dire que si la méthode est appelée avec moins de paramètres que ce qui est prévu dans le prototype, les paramètres manquants ont comme valeur ces valeurs par défaut. Exemple :

```
class rectangle{
    private :
           point _coord;       //coordonnées du point supérieur gauche
           int _w;             //largeur
           int _h;             //hauteur
    public :
           rectangle(int x, int y, int w = 0, int h = 0);
};
rectangle::rectangle(int x, int y, int w, int h){
    _coord.setPosition(x,y);
    _w = w;       //si ce constructeur est appelé avec seulement 2 int
    _h = h;       //les 2 autres seront initialisés à 0
}
```

5.2 Le destructeur

Pour se pratiquer à implémenter toutes les méthodes des objets, implémenter le destructeur de rectangle qui remet à 0 la largeur et hauteur. Le destructeur de point est automatiquement appelé lors de la destruction d'un rectangle et on n'a donc pas à remettre à 0 les coordonnées du point, ça se fera automatiquement.

5.3 Les getteurs/accesseurs

Vous devez implémenter les getteurs nécessaires pour l'objet rectangle (5). Comme le point implémente déjà des getteurs pour x et y, vous n'avez pas à les faire dans le rectangle, mais vous devez faire un getteur de coordonnées qui permettra l'accès à getX et getY du point comme ceci.

```
cout << r1.getPosition().getX();
```

5.4 Les setteurs/mutateurs

Vous devez implémenter les setteurs nécessaires pour l'objet rectangle (5). On doit pouvoir modifier chaque propriété individuellement et par groupe de 2 et les 4 en même temps. Attention, un rectangle ne peut avoir de coordonnées négatives, une largeur ou une hauteur négatives. Un rectangle avec une dimension à 0 est possible, il deviendrait une ligne ou un point si les 2 sont à 0. On ne veut pas générer d'erreur pour ça.

Après avoir créer ces méthodes, remplacer les instructions du constructeur avec paramètres pour qu'il appelle le setteurs qui a un assert pour protéger les propriétés en contrôlant le contenu des paramètres.

5.5 Autres méthodes intéressantes pour le rectangle

Maintenant, voici les méthodes qui seraient utiles pour le rectangle

```
void main()
{
    cout << "Entrer les coordonnées d'un rectangle, sa largeur et hauteur: (x,y) w x h ";
    r1.read(cin);       //saisie les infos comme ceci: (x,y) w x h. appel read de point

    cout << "Entrer les coordonnées d'un rectangle, sa largeur et hauteur: (x,y) w x h ";
    cin >> r2;          //fonction operator>> qui appel read

    cout << endl;
    r1.print(cout);     //on l'affiche comme ceci: (x,y) de h x w. appel print de point
    cout << endl << r2; //fonction operator<< qui appel print

    r2.draw(cout);      //on dessine le contour du rectangle (choisir un caractères)
                        //selon w et h a la position (x,y) en console

    cout << r2.surface() << r2.perimetre(); //calcul l'air et le périmètre du rectangle

    //surcharge de l'operator== qui compare les coordonnées et dimensions des rectangles
    if(r1 == r2)  cout << endl << "les rectangles sont identiques";
    else          cout << endl << "les rectangles ne sont pas identiques";

    //surcharger aussi l'operator!=

    //surcharge de l'operator< qui compare la taille (air ou périmètre) des rectangles
    if(r1 < r2)   cout << endl << "r1 est plus petit que r2";
    else          cout << endl << "r1 est plus grand que r2";

    //surcharger aussi l'operator>, >=, <=
}
```

Programmez et testez toutes ces méthodes.

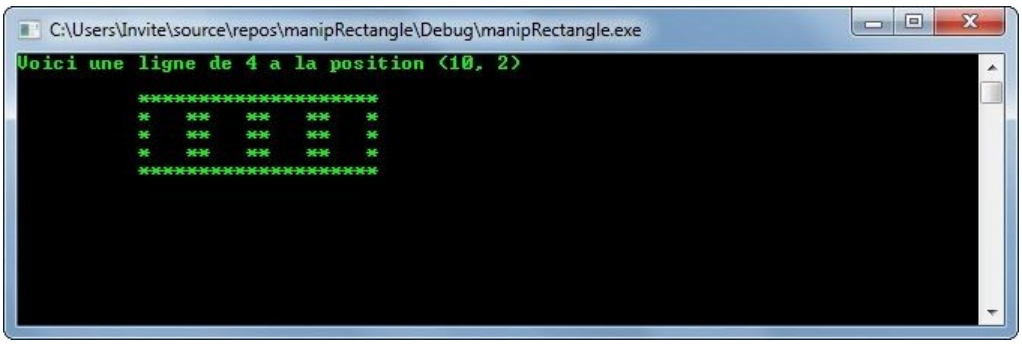
6. Dans le main, saisir les coordonnées et dimensions d'un rectangle et faite une boucle while infinie qui dessine ligne, triangle et grille avec un nombre de rectangle et une couleur qui varie aléatoirement à chaque itération de la boucle. Les couleurs varient entre 1 et 10. Effacer l'écran et faire une pause entre chaque forme. Vous pouvez effacer la console à l'aide de system("cls");

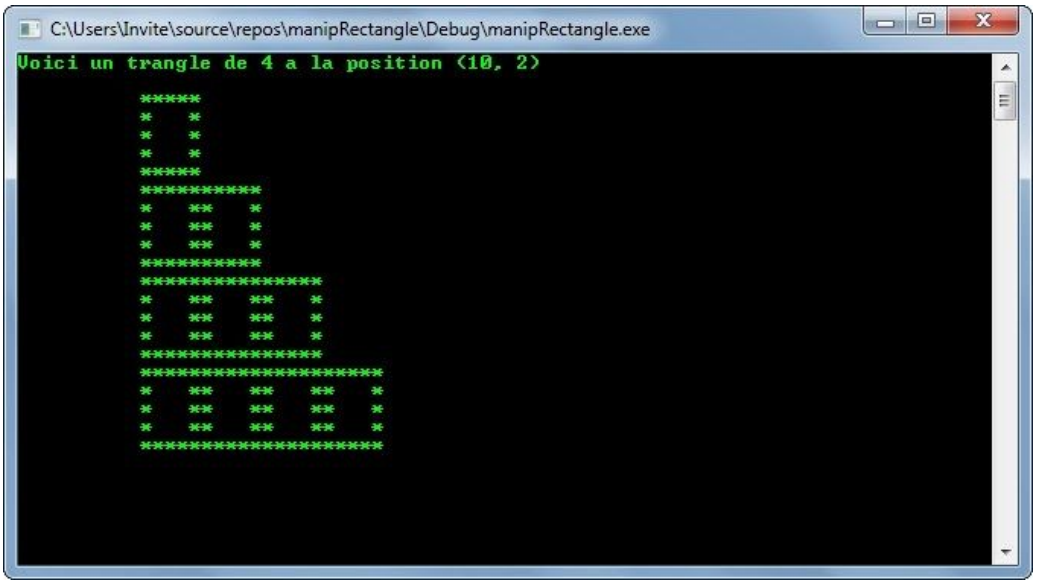
   On peut générer un nombre aléatoire de plusieurs façons, mais une des méthodes les plus simples, c'est avec les fonctions srand et rand de stdlib. Voici un exemple:

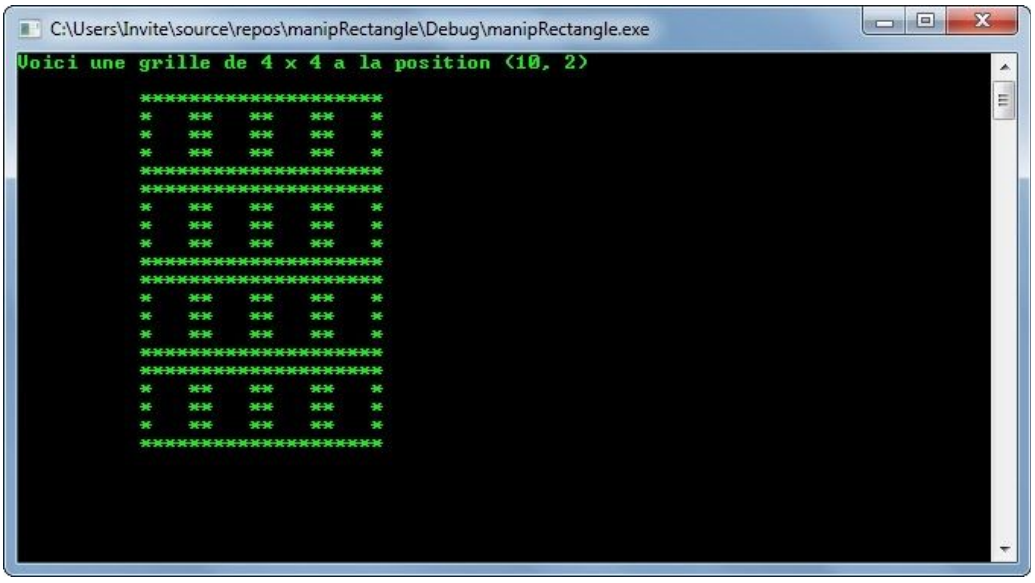
   ```
   /* initialise le point de départ du random : */
   srand (time(NULL));

   /* genere un nombre entre 1 et 10: */
   mb = rand() % 10 + 1;
   ```

   On doit faire 1 seul fois le srand pour initialiser le point de départ des randoms et ensuite chaque rand va générer un nb. différent. Si on omet cette instruction au début, nos nombres aléatoires seront toujours les mêmes.







Montrez-moi ce programme terminé avant le 2e cours de la semaine 6, soit avant le jeudi 27 février pour le 4118 ou avant le vendredi 28 février pour le 4120.