Université Paris Cité **Langages à objets avancés**
M1 **(c++)**

Surcharge d'opérateurs - Héritage multiple

Exercice 1

1. Définissez une classe `Vecteur` qui représente un vecteur de $\mathbb{R}^2$ avec ses constructeurs.
2. Surchargez les opérateurs `==` et `!=`, afin de pouvoir tester si deux vecteurs sont égaux ou différents.
3. Surchargez les opérateurs `+` et `-` afin de pouvoir faire la somme et la différence de deux vecteurs, et l'opérateur `*` afin de pouvoir réaliser le produit scalaire de deux vecteurs.
4. Surchargez l'opérateur `[]` afin de pouvoir accéder à une coordonnées du vecteur. Tester en particulier :

```
Vecteur v;
v[0] = 9;
```

5. Ajoutez une méthode `double norm()` renvoyant la norme du vecteur.
6. Faites en sorte que l'on puisse afficher un vecteur `v` en invoquant l'instruction `cout << v << endl`.
7. Que pouvez vous dire des destructeurs, constructeurs par copie et affectation qui sont implémentés par défaut ? (Faut-il les conserver, les interdire, les modifier ?)

Exercice 2 [Figures et diamants] Dans cet exercice, on utilise l'héritage pour définir des classes représentant des figures 2-dimensionnelles. On pourra utiliser la classe `Vecteur` précédente pour coder les coordonnées des points.

1. Déclarez la classe `Figure` : un attribut de type `string` représentant le nom de la figure, un constructeur, munissez la de méthodes virtuelles pures `void perimetre()`, `double area()`, `void test()`.
2. Déclarez une classe `Triangle` qui hérite de `Figure` : un triangle sera caractérisé par la donnée de trois points. Définissez un affichage de façon à produire "`triangle [nom] :` " suivi des coordonnées des trois points. Définissez ses méthodes héritées qui calculent périmètre et surface des triangles.
3. Déclarez une classe `Quadrilatere` héritant de `Figure` et définie par quatre points.
4. Déclarez une classe `Rectangle` héritant de `Quadrilatere`. Proposez une manière réaliste de les construire. Définir ses méthodes héritées.
5. Déclarez une classe `Losange`
6. Déclarez une classe `Carre` qui héritera à la fois de `Rectangle` et `Losange`. Faites bien en sorte qu'un `Carre` ne soit associé qu'à un seul `Quadrilatere`.
7. Tester sur des exemples simples. Vérifier en particulier que lorsque plusieurs calculs sont possibles ils donnent le même résultat, que les classes de bases ne sont pas dupliquées (changez les noms par exemple)

Exercice 3 — Opérateur « () »

On définit l'« interface » (i.e. classe avec seulement des méthodes virtuelles pures) ci-dessous.

```
class ValeursAdmises {//"interface"
 public :
  virtual  bool operator()(char val) = 0;
};
```

On y indique que l'opérateur « `()` » est défini comme prenant un argument `val` de type `char`, cela permettra d'utiliser une notation fonctionnelle sur l'objet pour vérifier qu'une valeur `val` répond à certains critères.

1. Ecrivez une sous-classe concrète `Intervalle` implémentant l'interface dont les objets sont définis par une valeur `char min` et une valeur `char max`, et dont la « fonction » correspondant à l'opérateur redéfini vérifiera que la valeur donnée en argument est dans cet intervalle.

   Exemple d'utilisation :

   ```
   Intervalle inter {'a', 'd'};
   if(inter('e'))  // utilisation d'operator(char)
    cout << "la valeur 'e' est ok" << endl;
   else
    cout << "la valeur 'e' n'est pas ok" << endl;
   if(inter('c'))
    cout << "la valeur 'c' est ok" << endl;
   else
    cout << "la valeur 'c' n'est pas ok" << endl;
   ```

2. Faites de même pour `TableauValeurs` dont les objets encapsulent un tableau de caractères et dont la « fonction » va vérifier que la valeur donnée en argument est dans ce tableau.

   Exemple d'utilisation :

   ```
   char tab[] {'b', 'o', 'n', 'j', 'u', 'r'};
   TableauValeurs tableau { tab, 6 };
    if(tableau('j'))
     cout << "la valeur 'j' est ok" << endl;
   else
      cout<< "la valeur 'j' n'est pas ok"<< endl;
   if(tableau('c'))
     cout << "la valeur 'c' est ok" << endl;
   else
      cout << "la valeur 'c' n'est pas ok" << endl;
   ```

3. Écrivez une fonction :

   ```
   std::vector<char> filtre(const std::vector<char>&,
                            const ValeursAdmises&);
   ```

   qui prend en argument un tableau d'éléments de type `char` et un objet de type `ValeursAdmises` et retournera un tableau ne contenant que les éléments `l` qui sont admis. Testez la.

   Exemple d'utilisation :

```
vector<char> res =
  filtre(vector<char> {'a', 'b','z','o'}, tableau);

for(char const& val: res)
  cout << val << ";";

cout << endl;
```

Exercice 4 On souhaite écrire une classe `Fraction` qui représente les nombres en fractions entières. L'un des constructeurs de cette classe prendra en entrée deux entiers n et d (numérateur et dénominateur) et stockera les deux entiers correspondant à la fraction irréductible : (c.à.d tel que seul n peut être négatif, et tel que n et d ont été divisés par leur *pgcd*)

1. Définissez cette classe et écrivez la méthode statique privée qui calcule le *pgcd*. Pour gagner du temps, voici le code du *pgcd* (adaptez en fonction de vos besoins) :

```
int pgcd(int x, int y) {
        if (x<0) return pgcd(-x,y);
        if (y<0) return pgcd(x,-y);
        if (x==0) return y;
        if (x>y) return pgcd(y,x);
        return pgcd(x,y-x);
}
```

2. Écrivez le constructeur `Fraction(int, int)` décrit au début de l'exercice. Que penser des versions par défaut du destructeur, du constructeur de copie, de l'affection ?
3. Redéfinissez l'opérateur de sortie `<<` pour permettre un affichage et écrivez un petit main pour la tester.
4. Surchargez les opérateurs `+` et `-` sur les fractions.
5. Vérifiez qu'une opération $\frac{1}{2}+1$ ne compile pas. Modifiez simplement le constructeur de fractions pour qu'il accepte un seul argument (le second valant 1). Vérifiez que l'expression précédente est à présent évaluée. Quel est le mécanisme qui a été mis en œuvre ?
6. Complétez afin de pouvoir évaluer également $1 + \frac{2}{4}$.
7. Relisez vos déclarations de fonctions pour utiliser au maximum le mot-clé **const** et des variables références au lieu de variables copiées lorsqu'elles sont passées en argument.