UNIVERSITE IBN TOFAIL
Faculté des Sciences
Département de Mathématiques et d'Informatique
KENITRA

Année universitaire 2019-2020

Rattrapage – 2019/2020
SMI – POOP en C++

Exercice 1 :

Donner la sortie du programme suivant, et justifier votre réponse :

```
#include <iostream>
using namespace std;

class T{
    int i;
public:
    T(int n = 0){ i = n; cout << "+ Constructeur : " << i << endl; }
    T( T & v){ i = v.i; cout << "* Constructeur par recopie " << i << endl; }
    ~T(){ cout << "- Destructeur : " << i << endl;}
    T& operator++(){ i++; return *this;}
    friend ostream& operator<<(ostream& sortie, const T& t);
};

ostream& operator<<(ostream& sortie, const T& t){
    sortie << t.i; return sortie;
}

void f1(T v){
    static T t = v;
    cout << "Fonction f1 - " << ++t << endl;
}

void f2(T &v) {
    static T t = v;
    cout << "Fonction f2 - " << ++t << endl;
}

int main(){
    cout << "DEBUT\n";
    T u;
    cout << "---- Premier appel de f1 " << endl;
    f1(u);
    cout << "---- Second appel de f1 " << endl;
    f1(u);
    cout << "---- Premier appel de f2 " << endl;
    f2(u);
    cout << "---- Second appel de f2 " << endl;
    f2(u);
    cout << "FIN\n";
    return 0;
}
```

Exercice 2 :

On souhaite réaliser une classe C qui permet de manipuler les nombres complexes :

```
class C
{
    double re;          // partie réelle
    double im;          // partie imaginaire
public:
_
};
```

Donner l'interface et le corps de cette classe en définissant :

1. Les constructeurs nécessaires pour que les instructions suivantes, produisent les résultats correspondants :

```
// Pour tous réels x, y donnés
C z1;                          // crée le nombre z1 = 0
C z2 = x;                      // crée le nombre z2 = x+ix
C z3(x,y);                     // crée le nombre z3 = x+iy
```

2. Les surcharges des opérateurs +, - et *, qui permettent d'écrire les instructions suivantes :

```
a1 = u + v;                    // a1 est la somme de u et v
a2 = u - v;                    // a2 est la différence entre u et v
a3 = u * v;                    // a3 est le produit de u et v
a4 = h * u;                    // a4 est le produit de u par le réel h
/* où a1, a2, a3 et a4 sont des objets de C et u et v sont des nombres complexes
constants, et h est un réel */
```

3. Une surcharge de l'opérateur << qui permet d'afficher un nombre complexe.

Exercice 3 :

On se propose de réaliser une classe `Ensemble` qui permet de manipuler les ensembles d'éléments d'un type de base (int, double, char, ...) T donné, dont les membres donnés (privés) sont :

- `conteneur` : tableau dynamique contenant les éléments de l'ensemble.
- `taille` : taille du tableau conteneur.
- `cardinal` : nombre des éléments de l'ensemble.

N.B. La taille du tableau aura une valeur par défaut CAPACITE_MAX, et sera augmentée de CAPACITE_PLUS en cas de besoin (ajout d'un élément lorsque le tableau est plein).

Donner l'interface et le corps de cette classe en définissant :

- Un constructeur qui permet de créer un ensemble vide.
- Un constructeur qui permet de créer un ensemble à partir d'un tableau.
- Un constructeur par recopie.
- Un destructeur.
- Une surcharge de l'opérateur d'affectation.
- Une surcharge de l'opérateur << qui permet d'ajouter un élément à l'ensemble, de telle sorte qu'on peut ajouter plusieurs éléments dans une même expression (par exemple : `e << n << m;` ajoute les entiers n et m à l'ensemble e).
- Une surcharge de l'opérateur << qui permet d'afficher les éléments de l'ensemble.
- Une surcharge de l'opérateur % qui permet de connaître si un élément donné appartient à l'ensemble :

```
n % e        // vaut true si n appartient à e
```

- Une surcharge de l'opérateur == qui permet de tester l'égalité entre deux ensembles.
- Une surcharge de l'opérateur < permettant de tester l'inclusion.
- Des surcharges des opérateurs +, *, et - donnant respectivement la réunion, l'intersection et la différence entre ensembles.