Nom :
Prénom :

Date : 22/01/2021

PRG1 – Test écrit 3

Exercice 1 (5 pts)

Soit la classe C définie comme suit

```
class C {
public:
    C()                         { cout << "1"; }
    C(int)                      { cout << "2"; }
    C(C const& c)               { cout << "3"; }
    C& operator=(C const& c) { cout << "4"; return *this; }
    ~C()                        { cout << "5"; }
};
```

Qu'affichent un appel aux fonctions suivantes ?

1.1	`void f1() {`
	`  C c[1];`
	`  c[0] = C(1);`
	`  cout << " / ";`
	`}`
		1245 / 5

1.2	`void f2() {`
	`  C c(1);`
	`  C d = c;`
	`  cout << " / ";`
	`}`
		23 / 55

1.3	`void f3() {`
	`  C c(1);`
	`  {`
	`  C d(2);`
	`  d = c;`
	`  cout << " / ";`
	`  }`
	`  cout << " / ";`
	`}`
		224 / 5 / 5

1.4	`void f4() {`
	`  array<C,3> a { C(1), C(2) };`
	`  a[2] = a[0];`
	`  cout << " / ";`
	`}`
		2214 / 555

1.5	`void f5() {`
	`  C c;`
	`  c = 5;`
	`  cout << " / ";`
	`}`
		1245 / 5

Exercice 2 (10 points)

Soient les 5 fonctions suivantes

```
template <typename T, typename U> int f(T, U) { return 1; }
template <typename T> int f(T, T)             { return 2; }
template <typename T> int f(T, short)         { return 3; }
template <typename T> int f(T, double)        { return 4; }
int f(int, double)                            { return 5; }
```

Qu'affichent les appels suivants s'ils compilent ? Si l'appel est ambigu, indiquez-le.

2.1	`f(int(),int());`	2
2.2	`f(int(),short());`	3
2.3	`f(int(),float());`	1
2.4	`f(short(), short());`	ambigu
2.5	`f(short(),double());`	4
2.6	`f(int(),double());`	5
2.7	`f<>(int(),double());`	4
2.8	`f<double>(int(),double());`	ambigu
2.9	`f<double>(short(),short());`	3
2.10	`f<int,int>(short(),short());`	1

Exercice 3 (15 pts)

Soit le fichier `main.cpp` suivant

```
#include <iostream>

using namespace std;

int main() {
    Complexe a;
    Complexe b(2, 3);

    cout << a << " + " << b << " = " << a+b << endl;

    b += Complexe(1, 1);

    cout << "b = " << b << endl;
}
```

Dont l'exécution affiche

```
(0,0) + (2,3) = (2,3)
b = (3,4)
```

Ecrivez les fichiers `complexe.h` et `complexe.cpp` qui déclarent et définissent la classe `Complexe` gérant des nombres complexes et dont la partie privée permet de stocker les parties réelles et imaginaires sous forme de `double`.

```
class Complexe {
    double real, imag;
public:
    // à compléter
}
```

Ne déclarez et définissez que les éléments nécessaires à l'exécution du programme ci-dessus.

`complexe.h`

```cpp
#ifndef TE1_20_21_Q1_COMPLEXE_H
#define TE1_20_21_Q1_COMPLEXE_H

#include <ostream>

class Complexe {
    double real, imag;
public:
    Complexe(double r = 0, double i = 0);
    Complexe& operator+=(Complexe const& other);
    friend std::ostream& operator<<( std::ostream&,
                                     Complexe const&);
};

Complexe operator+(Complexe lhs, Complexe const& rhs);

#endif //TE1_20_21_Q1_COMPLEXE_H
```

complexe.cpp

```cpp
#include "complexe.h"

Complexe::Complexe(double r, double i) : real(r), imag(i) {
}

Complexe& Complexe::operator+=(Complexe const& other) {
    real += other.real;
    imag += other.imag;
    return *this;
}

Complexe operator+ (Complexe lhs, Complexe const& rhs) {
    return lhs += rhs;
}

std::ostream& operator<<(std::ostream& o, Complexe const& c) {
    return o << '(' << c.real << ',' << c.imag << ')';
}
```

Exercice 4 (15 pts)

Écrivez les fonctions génériques somme et afficher et leurs surcharges ou spécialisations éventuelles pour que le code ci-dessous

```
#include <iostream>
#include <array>
#include <iomanip>

// le code que vous écrivez vient ici

using namespace std;

int main() {
    cout << fixed << setprecision(1);

    array<int, 5> a{1, 2, 3, 4, 5};
    afficher("a = ",a);

    array<double, 3> b{10, 20, 30};
    afficher("b = ",b);

    auto c = somme<short, 4>(a, b);
    afficher("c = ",c);

    auto d = somme<double, 6>(a, c);
    afficher("d = ",d);

    array<short, 0> e{};
    afficher("e = ",e);

    array<bool, 0> f{};
    afficher("f = ",f);
}
```

affiche le résultat suivant à la console :

```
a = | 1 | 2 | 3 | 4 | 5 |
b = | 10.0 | 20.0 | 30.0 |
c = | 11 | 22 | 33 | 4 |
d = | 12.0 | 24.0 | 36.0 | 8.0 | 5.0 | 0.0 |
e = | tableau vide |
f = | tableau vide |
```

La généricité ne doit permettre d'appeler ces fonctions qu'avec des `std::array`, mais doit accepter des types et tailles quelconques en entrées et sortie.

Important : votre fonction affichage ne peut effectuer aucun test sur la taille du `std::array` à afficher.

[Utilisez cette page pour répondre à la question 4]

```cpp
template<typename T, size_t N,
         typename T1, size_t N1,
         typename T2, size_t N2>
std::array<T,N> somme(std::array<T1,N1> const& a,
                      std::array<T2,N2> const& b)
{
    std::array<T,N> c{};
    for(size_t i = 0; i < N; ++i) {
        if(i<N1) c[i] += T(a[i]);
        if(i<N2) c[i] += T(b[i]);
    }
    return c;
}

template<typename T, size_t N>
void afficher(std::string const& s,
              std::array<T,N> const& a)
{
    using std::cout, std::endl;
    cout << "|";
    for(auto const& e : a)
        cout << " " << e << " |";
    cout << endl;
}

template<typename T>
void afficher(std::string const& s,
              std::array<T,0> const&)
{
    using std::cout, std::endl;
    cout << "| tableau vide |" << endl;
}
```

Exercice 5 (15 pts)

Écrivez la classe générique `Instances<T>` qui permet de monitorer le nombre d'objets de type `T` qui sont actuellement vivants en incluant simplement un objet de type `Instances<T>` dans la section privée de la classe `T`. Ce nombre est accessible via la méthode `getCount()`. Le code ci-dessous - qui affiche **35352**, donne un exemple d'utilisation de cette classe. Par ailleurs, il doit être impossible de tricher en créant explicitement un objet de type `Instances<T>` depuis un autre endroit que la classe `T`.

Séparez déclaration et définition de la classe générique.

```
class A {
    Instances<A> a;
public:
    // ...
};

class B {
    Instances<B> b;
public:
    // ...
};

int main() {

    A a[3];
    cout << Instances<A>::getCount();     // 3
    {
        A aa[2];
        cout << Instances<A>::getCount(); // 5
    }
    cout << Instances<A>::getCount();     // 3

    vector<B> b(5);
    cout << Instances<B>::getCount();     // 5
    b.resize(2);
    cout << Instances<B>::getCount();     // 2

}
```

[Utilisez cette page pour répondre à la question 5]

```
template<typename T>
class Instances
{
    static unsigned cnt;
    Instances();
    ~Instances();
public:
    static unsigned getCount();
    friend T;
};

template<typename T>
unsigned Instances<T>::cnt = 0;

template<typename T>
Instances<T>::Instances() {
    ++cnt;
}

template<typename T>
Instances<T>::~Instances() {
    --cnt;
}


template<typename T>
unsigned Instances<T>::getCount() {
    return cnt;
}
```