"Ss. Cyril and Methodius" University in Skopje
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Object oriented programming

Exercises 11

Version 1.0, 4 May, 2017

Table of Contents

1. Generic programming and templates

1.1. Function templates

1.1.1. Template function for swapping the value of two variables

Solution oop_av111_en.cpp

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

template <typename T>
void mySwap(T &a, T &b) {
    T temp;
    temp = a;
    a = b;
    b = temp;
}

int main() {
    int i1 = 1, i2 = 2;
    mySwap(i1, i2); // generated mySwap(int &, int &)
    cout << "i1 is " << i1 << ", i2 is " << i2 << endl;
    char c1 = 'a', c2 = 'b';
    mySwap(c1, c2); // generated mySwap(char &, char &)
    cout << "c1 is " << c1 << ", c2 is " << c2 << endl;
    double d1 = 1.1, d2 = 2.2;
    mySwap(d1, d2); // generated mySwap(double &, double &)
    cout << "d1 is " << d1 << ", d2 is " << d2 << endl;
    mySwap(i1, d1); // error 'mySwap(int&, double&)'
    return 0;
}
```

1.1.2. Template function for computing absolute value of a number

Solution oop_av112_en.cpp

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

template<typename T>
T abs(T value) {
    T result; // result is also from type T
    result = (value >= 0) ? value : -value;
    return result;
}

int main() {
    int i = -5;
    cout << abs(i) << endl;
    double d = -55.5;
    cout << abs(d) << endl;
    float f = -555.5f;
    cout << abs(f) << endl;
}
```

1.1.3. Overloading function templates

Solution `oop_av113_en.cpp`

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

template <typename T>
void mySwap(T &a, T &b) {
    T temp;
    temp = a;
    a = b;
    b = temp;
}

template <typename T>
void mySwap(T a[], T b[], int size) {
    T temp;
    for (int i = 0; i < size; ++i)
    {
        temp = a[i];
        a[i] = b[i];
        b[i] = temp;
    }
}

template <typename T>
void print(const T * const array, int size) {
    cout << "(";
    for (int i = 0; i < size; ++i)
    {
        cout << array[i];
        if (i < size - 1)
            cout << ",";
    }
    cout << ")" << endl;
}

int main() {
    const int SIZE = 3;
    int i1 = 1, i2 = 2;
    mySwap(i1, i2);
    cout << "i1 is " << i1 << ", i2 is " << i2 << endl;
    int ar1[] = {1, 2, 3};
    int ar2[] = {4, 5, 6};
    mySwap(ar1, ar2, SIZE);
    print(ar1, SIZE);
    print(ar2, SIZE);
}
```

1.2. Class templates

1.2.1. Example 1

Solution `oop_av114_en.cpp`

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

template <typename T>
class MyComplex {
private:
    T real, imag;
public:
    MyComplex<T> (T real = 0, T imag = 0) : real(real), imag(imag) { }
    MyComplex<T> & operator+= (const MyComplex<T> & rhs) {
        real += rhs.real;
        imag += rhs.imag;
        return *this;
    }

    MyComplex<T> & operator+= (T value) {
        real += value;
        return *this;
    }

    bool operator== (const MyComplex<T> & rhs) {
        return (real == rhs.real && imag == rhs.imag);
    }

    bool operator!= (const MyComplex<T> & rhs) {
        return !(*this == rhs);
    }

    MyComplex<T>  operator++() {
        ++real;
        return *this;
    }

    MyComplex<T> operator++ (int dummy) {
        MyComplex<T>  saved(*this);
        ++real;
        return saved;
    }

    friend ostream & operator<< (ostream & out, const MyComplex<T> & c) {
        out << '(' << c.real << ',' << c.imag << ')';
        return out;
    }

    friend istream & operator>> (istream & in, MyComplex<T> & c) {
        T inReal, inImag;
        char inChar;
        bool validInput = false;
        in >> inChar;
        if (inChar == '(')
        {
            in >> inReal >> inChar;
            if (inChar == ',')
            {
                in >> inImag >> inChar;
                if (inChar == ')')
                {
                    c = MyComplex<T>(inReal, inImag);
                    validInput = true;
                }
            }
        }
if (!validInput) cout << “Vnesete go brojot vo format: (real,   imag)” << endl;
        return in;
    }

    friend  MyComplex<T> operator+ (const MyComplex<T> & lhs, const MyComplex<T> & rhs) {
        MyComplex<T> result(lhs);
        result += rhs;
        return result;
    }

    friend  MyComplex<T> operator+ (const MyComplex<T> & lhs, T value) {
        MyComplex<T> result(lhs);
        result += value;
        return result;
    }

    friend const MyComplex<T> operator+ (T value, const MyComplex<T> & rhs) {
        return rhs + value;
    }
};

int main() {
    MyComplex<double> c1(3.1, 4.2);
    cout << c1 << endl; // (3.10,4.20)
    MyComplex<double> c2(3.1);
    cout << c2 << endl; // (3.10,0.00)
    MyComplex<double> c3 = c1 + c2;
    cout << c3 << endl; // (6.20,4.20)
    c3 = c1 + 2.1;
    cout << c3 << endl; // (5.20,4.20)
    c3 = 2.2 + c1;
    cout << c3 << endl; // (5.30,4.20)
    c3 += c1;
    cout << c3 << endl; // (8.40,8.40)
    c3 += 2.3;
    cout << c3 << endl; // (10.70,8.40)
    cout << ++c3 << endl; // (11.70,8.40)
    cout << c3++ << endl; // (11.70,8.40)
    cout << c3 << endl; // (12.70,8.40)
    MyComplex<int> c5;
    cout << "Enter complex number in format (real, imag): ";
    cin >> c5;
    return 0;
}
```

2. Source code of the examples and problems

https://github.com/finki-mk/SP/

Source code ZIP