Indian Institute of Technology, Kharagpur

Department of Computer Science and Engineering

Assignment 4: C++ Programming, Spring 2013-14

Software Engineering (CS 29006)

Assignment Date: 13-Feb-2014 **Submission Deadline:** 23:55 hrs, 02-Mar-2014

Instructions

- All assignments in this set should be coded in C/C++.
- Zero marks for a submission if it does not pass the plagiarism test or if you copied from someone in the class.
- Zero marks for a submission and 20% deduction from all previous assignments if someone in the class copied from you.

---

1. You had developed a Poly Data type in Assignment 3 for value type `Fraction` and coefficient type `int`. Generalize that now to a parametrized Poly Data type to deal with polynomials of value type `T` and coefficient type `U`. These polynomials should support the operations for a Poly type as defined in the following class definition (`Polynomial.hxx`, same as Assignment 3):

```
#include <iostream>// Defines istream & ostream for IO
#include <vector>
using namespace std;

template<
    class T,    // Type of Value
    class U>    // Type of Coefficients
class Poly {

public:
    // CONSTRUCTORS
    Poly(unsigned int = 0);    // Uses default parameters.
    Poly(const Poly&);         // Copy Constructor

    // DESTRUCTOR
    ~Poly() {}                 // No virtual destructor needed

    // BASIC ASSIGNMENT OPERATOR
    Poly& operator=(const Poly&);

    // UNARY ARITHMETIC OPERATORS
    Poly operator-();           // Operand ’this’ implicit
    Poly operator+();

    // BINARY ARITHMETIC OPERATORS
    Poly operator+(const Poly&);
    Poly operator-(const Poly&);

    // ADVANCED ASSIGNMENT OPERATORS
    Poly& operator+=(const Poly&);
    Poly& operator-=(const Poly&);

    // BASIC I/O using FRIEND FUNCTIONS
    template<class T, class U>
    friend ostream& operator<<(ostream& os, const Poly<T, U>& p);
```

```
        template<class T, class U>
        friend istream& operator>>(istream& is, Poly<T, U>& p);

        // METHODS
        T Evaluate(const T&); // Evaluates the polynomial - use Horner's Rule

    private:

        // DATA MEMBERS
        unsigned int    degree_;
        vector<U>       coefficients_;
    };
```

(a) Implement this class. **[10 Marks]**

Grading Guideline:

Completeness & Correctness	*70%*
Code Quality (simplicity, readability, efficiency, reusability, standard library usage, robustness)	*20%*
Comments	*10%*

(b) Your class will be tested by the TA using the following function (`TestPoly.cxx`):

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

#include "Fraction.hxx"
#include "Polynomial.hxx"

void TestPoly()
{
    cout << "\nTest Poly Data Type" << endl;

    // Polynomial with int value and int coefficients
    Poly<int, int> p(10);

    cout << "Input Poly<int, int>: p(x)" << endl;
    cin >> p;
    cout << "\np(x) = " << p << endl;

    int x = 5;
    cout << "p(" << x << ") = " << p.Evaluate(5) << endl;

    Poly<int, int> q = p;
    cout << "Copied Polynomial: " << q << endl;

    Poly<int, int> r;
    r = p;
    cout << "Assigned Polynomial: " << r << endl;

    r = -p;
    cout << "Negated Polynomial -p(x) = " << r << endl;

    cout << "Input Poly<int, int>: q(x)" << endl;
    cin >> q;
    cout << "\nq(x) = " << q << endl;

    r = p + q;
    cout << "p(x) + q(x) = " << r << endl;

    r = p - q;
    cout << "p(x) - q(x) = " << r << endl;

    p += q;
    cout << "p(x) <-- p(x) + q(x): " << p << endl;

    q -= p;
    cout << "q(x) <-- q(x) - p(x): " << q << endl;

    // Polynomial with Fraction value and int coefficients
    Poly<Fraction, int> pFi(10);

    cout << "Input Poly<Fraction, int>: pFi(x)" << endl;
    cin >> pFi;
    cout << "pFi(x) = " << pFi << endl;

    Fraction f;
    cout << "Input Fraction" << endl;
    cin >> f;
    cout << "At " << f << ": " << pFi.Evaluate(f) << endl;

    // Polynomial with Fraction value and Fraction coefficients
    Poly<Fraction, Fraction> piF(10);

    cout << "Input Poly<Fraction, Fraction>: piF(x)" << endl;
    cin >> piF;
    cout << "piF(x) = " << piF << endl;

    cout << "At " << f << ": " << piF.Evaluate(f) << endl;

    return;
  }
```

Hence compliance to this function is critical. [**10 Marks**]

Grading Guideline: Based on percentage of tests passed / failed.

(c) State the relationships between value parameters type `T` and coefficient parameters type `U` and justify. [**10 Marks**]

2. You need to write a quadratic equation solver that would read three double constants a, b, and c and output a solution. You need to handle all cases for solution including single root, repeated roots and complex roots.

   (a) Implement the solver in C using setjmp and longjmp for handling corner cases. [**10 Marks**]
   Note: This has not been discussed in class. You are expected to know this as you know C. If you do not, google to find out. Or, refer to:

   https://www.cs.purdue.edu/homes/cs240/lectures/Lecture-19.pdf
   http://web.eecs.utk.edu/ huangj/cs360/360/notes/Setjmp/lecture.html
   http://en.wikipedia.org/wiki/Setjmp.h.

   (b) Implement the solver in C++ using exception handling. Define and use a simple Complex data type for complex roots. [**15 Marks**]

   (c) Discuss why the solution (b) is superior to solution (a). [**5 Marks**]

Grading Guideline:

Completeness of Design	*20%*
Completeness & Correctness of Implementation	*40%*
Completeness of Tests	*20%*
Code Quality (simplicity, readability, efficiency, re-usability, standard library usage, robustness)	*10%*
Comments	*10%*

3. This problem tests your understanding of implementing a data structure in C++ specifically when underlying types are not known a priori. Hence it extends on the stack of `int` you had done in Assignment 3. As a part of Assignment 3, you have done the following:

(a) Design and implement a stack of integers (int) in C. Use your stack to convert an infix expression with integer constants to postfix and evaluate the expression. Assume the operators +, -, * and / in your expression.

For example, if the infix expression is 2+3*4 then the postfix expression is 234*+ and the evaluated value is 14.

Handle all corner cases in your code. The container for your stack should be dynamically allocated as a linked list.

(b) Repeat (a) in C++. For the stack of int, you should implement a Stack class for underlying int element types.

Now you should extend as follows:

(c) Repeat (b) in C++ using a Stack class for unspecified element type (using templates) and instantiate with int type. Design a suitable interface for your stack.

(d) Repeat (c) in C++ using std::stack from STL (you should use other types from STL as appropriate).

(e) Compare and contrast the above four implementations from the perspectives of software engineering metrics including:

i. Efficiency

ii. Ease of Implementation

iii. Testability

iv. Robustness & Maintainability

v. Readability

[20+5+5*3 = 40 Marks]

Grading Guideline:

Completeness of Design	*20%*
Completeness & Correctness of Implementation	*40%*
Completeness of Tests	*20%*
Code Quality (simplicity, readability, efficiency, reusability, standard library usage, robustness)	*10%*
Comments	*10%*