CSCI804

Object and Generic Programming in C++

Autumn 2016

Assignment 3 (10 marks)

Due by 11:59pm Sunday 15 May 2016.

Task One: Stack and expression (5 marks)

Human generally write expressions like 3 + 4 and 7 / 9 in which the operator (+ or / here) is written between its operands-this is called *infix notation*. Computers "prefer" *postfix notation* in which the operator is written to the right of its two operands. The preceding infix expressions would appear in postfix notation as 3 4 + and 7 9 /, respectively.

To evaluate a complex infix expression, a compiler would first convert the expression to postfix notation and then evaluate the postfix version of the expression. Each of these algorithms requires only a single left-to-right pass of the expression. Each algorithm uses a stack object in support of its operation, and in each algorithm the stack is used for a different purpose.

In this task, you are to write a C++ program of the infix-to-postfix conversion algorithm.

Define and implement a **Stack** template class in a file **stack.h**. Your Stack template class must have the following public functions:

- void push(T): to insert a data to the Stack.
- T pop(): to retrive a data from the Stack.
- bool isEmpty(): this function will return 1 if the Stack is empty or 0 otherwise.
- void print() : this function will print the contents of the Stack WITHOUT destroying the contents of the Stack.
- T stackTop() : this function will return the TOP element of the Stack WITHOUT destroying the contents of the original Stack.

Define a class called **InfixToPostfix** in a file **postfix.h**, which is able to convert an infix notation to a postfix notation. The class has two private variables:

```
string infix;
string postfix;
```

These variables are used to hold the infix and postfix notations, respectively.

Define the following public member functions for the class **InfixToPostfix**:

- void setInfix(const std::string &) : This is a function to set initial value to the data member *infix*.
- void convert (): This is a function that convert the infix notation hold in the infix variable to a postfix notation variable. You will use the template class Stack defined above in this function.
- std::string getPostfix(): This is a function that returns a postfix notation from the data member *postfix*.

Implement the member functions for the class Infix2Postfix in a file **postfix.cpp**.

The algorithm to convert an infix notation to a postfix notation is as follows:

1. Push the left parenthesis '(' to the stack.
2. Append a right parenthesis ')' to the end of infix expression.
3. While the stack is not empty, read infix from left to right and do the following:
   - If the current input in infix is an operand, copy it to the next element of postfix.
   - If the current input in infix is a left parenthesis, push it onto the stack.
   - If the current input in infix is an operator, then:
     - Pop operators (if there are any) at the top of the stack while they have equal or higher precedence than the current operator, and insert the popped operators in postfix.
     - Push the current operator in infix onto the stack.
   - If the current input in infix is a right parenthesis, then:
     - Pop operators from the top of the stack and insert them in postfix until a left parenthesis is at the top of the stack.
     - Pop (and discard) the left parenthesis from the stack.

The following arithmetic operations are allowed in an expression:

+	addition
-	subtraction
*	multiplication
/	division
%	modulus

The precedence of operators from the lowest to highest is defined as following:

+ -
* / %

Write a driver program include main function in a file **task1Main.cpp** to declare an instance of **InfixToPostfix**. Then get an infix notation from the keyboard and convert the infix notation to the postfix notation. Finally print out the postfix notation.

For example, when the input is

```
1 + 3
```

The output of postfix notation should be

```
1 3 +
```

You can download the files **exp1.txt** and **exp2.txt** to test your program. See the testing for more details.

Testing:

You can compile the program by

```
g++ -o task1 task1Main.cpp postfix.cpp
```

Run the program like

```
./task1 < exp1.txt
```

```
AB CDE * RST UV XX / - 3 * + X5 -
```

Test the program by exp2.txt like

```
./task1 < exp2.txt
```

```
12.53 21.2 33.2 15.4 2 / - * + 8.5 2.6 12 2 / + * -
```

Task Two: Templates & manipulators for matrices (5 marks)

Background Information

A matrix is a set of numbers arranged in rows and columns as in a conventional 2D array. If *M* is a matrix with **R** rows and **C** columns, we say the matrix *M* is of the size RxC. Every element of a matrix has a row position *i* and a column position *j*. For example, *M[1,3]* is an element of the matrix *M* with the row position 1 and column position 3. In the context of this assignment, matrix elements can be real numbers, or complex numbers. Two matrixes A and B can be added, but only if they have the same size, by adding elements from the corresponding positions.

`C = A + B`, where `C[i,j] = A[i,j] + B[i,j]`.

Two matrixes A and B can be subtracted, again only if they have the same size, by subtracting elements from the corresponding positions.

`C = A – B`, where `C[i,j] = A[i,j] – B[i,j]`.

Two matrices A and B, of sizes R_axC_a and R_bxC_b respectively, can be multiplied together if and only if C_a=R_b. Multiplication is performed using the following algorithm:

```
For i=1 to $R_a$
     For j=1 to $C_b$
          C[i,j]=0
          For k=1 to $C_a$
               C[i,j]=C[i,j]+A[i,k]*B[k,j]
          End For
     End For
End For
```

If elements of a matrix are complex, the matrix is called a complex matrix. A complex number *(r,m)* is a pair that consist of a real part *r* and an imaginary part *m*, where r and m are themselves both real numbers. Two complex numbers can be added and subtracted, that is defined by the following rules:

```
(a,b) + (c,d) = (a+c, b+d)
(a,b) - (c,d) = (a-c, b-d)
(a,b)*(c,d) = (ac-bd,bc+ad)
```

The magnitude of a complex number *(a, b)* is a real number $g = \sqrt{a^2 + b^2}$.

Design Requirements

Define a class template `matrix` that can be instantiated to support basic operations on matrixes with **real,** or **complex elements**. The class shall input matrixes of any size using streams. It must have a constructor `matrix(int rows, int columns)`, the operators +, -, *, <<, and >> overloaded and **two new manipulators**. A manipulator `info` should be used to complement the matrix output with its size. For example:

```
cout<<info<<M<<endl;
```

produces the following output:

```
3x4 matrix
     2.1   4.3   0.1     1.3
     0.4  18.2   7.3     2.5
    45.3   5.0   5.3     7.6
```

Another manipulator, `noinfo`, shall be used to reset output to being without the matrix size information.

As addition and subtraction are defined only for matrixes of the same size, and a similar rule was stated earlier for multiplication, an exception must be thrown and handled if the sizes, or the types (complex, real) of two matrixes do not match. (Note that in practice complex and real matrices can be added.)

Define a class `complexn` to represent elements of complex matrices. The class must have a constructor `complexn(float real, float img)`, the operators +, -, *, <<, >> overloaded, and four new manipulators.
`cplx` – set output to complex format *(a, b).*
`real` – set output only to the real part *a* of the complex number.
`img` – set output only to the imaginary part *b* of the complex number.
`magnitude` – set output only to the magnitude *g* of the complex number.

After that, the output << of complex numbers shall follow the specified format.
Output of numbers shall be with one decimal point precision.

Input of complex numbers will use two real numbers. To represent real numbers you can use the `float` data type.

You should have a simple `main()` function which

1. **inputs four matrices, two real and two complex, of size at least 3 in each dimension and not more than 5 in each dimension.**
2. **For the real matrices, outputs**
    a. **each matrix using `info` manipulator,**
    b. **their sum and difference using `noinfo` manipulator, and,**
    c. **their product using the `info` manipulator.**
3. **For the complex matrices, outputs**
    a. **each matrix in complex format using `info`.**
    b. **the imaginary part, real part and magnitude of their sum, difference and product using the `noinfo` manipulator.**

You should provide a file, **testmatrix.txt** which contains the input values for use with the following command:

```
cat testmatrix.txt | ./Matrix
```

where **`Matrix`** has been generated, on Banshee, using the directive:

```
g++ task2Main.cpp –o Matrix
```

To facilitate an automatic compilation and testing, organize the code into:

```
matrix.h              // declaration
matrix.cpp       // implementation
task2Main.cpp         // main() function
testmatrix.txt        // test data
```

Inclusion model will be assumed in compiling the code.

Submission

Assignments are submitted electronically via the ***submit*** system. For this assignment you must submit all the files via the command:

```
$ submit -u your_user_name -c CSCI804 -a 3 task1Main.cpp stack.h postfix.h postfix.cpp
task2Main.cpp matrix.h matrix.cpp testmatrix.txt
```

and input your password.

Make sure that you use the correct file names. The Unix system is case sensitive. You must submit all files in one ***submit*** command line.

Your program code must be in a good programming style, such as good names for variables, methods, classes, and keep indentation.

Submission via e-mail is NOT acceptable.

After submit your assignment successfully, please check your email of confirmation. **You would loss 50%~100% of the marks if your program codes could not be compiled correctly.**

Late submissions do not have to be requested. Late submissions will be allowed for a few days after close of scheduled submission (up to 3 days). Late submissions attract a mark penalty; this penalty may be waived if an appropriate request for special consideration (for medical or similar problem) is made via the university SOLS system *before* the close of the late submission time. No work can be submitted after the late submission time.

A policy regarding late submissions is included in the course outline.

The assignment is an **individual assignment** and it is expected that all its tasks will be solved **individually without any cooperation** with the other students. If you have any doubts, questions, etc. please consult your lecturer or tutor during tutorial classes or office hours. Plagiarism will result in a **FAIL** grade being recorded for that assessment task.