AST21105 Object-Oriented Programming & Design

Lab 13 – Template

A. Submission Details

In this lab, you are required to submit **ONE** C++ program with **ONE** files to solve the given problem shown in the section “Rational Class”. To make the program implementation easier, you are suggested to write your program using Visual Studio .NET instead of doing it directly using paper & pencil. After you have completed the implementation, submit your program files (i.e. Rationa,h) using the electronic “drop-box” in Canvas, **One week** after your lab section is conducted. For details, please refer to the following:

Tuesday sections	*by*	*19:00 on 26 April 2016 (Tuesday)*
Wednesday sections	*by*	*19:00 on 27 April 2016 (Wednesday)*

You are reminded to double-check your solution to verify everything is correct before submission. You are also required to put your name and student ID at the beginning of your source file.

Important: You are only able to submit your work once.

B. Objective

The objective of this lab is to help you get familiar with template that you have learned in class. The first part of this lab is meant to be a simple review of template. While the second part of this lab is a practical task that allows you to use all the techniques introduced.

C. Review

Template

It is often useful to be able to reuse the same algorithms with many different data types. This is called generic programming. This week you will learn generic programming with templates. By the end of the lab you should be able to take a function or class, create a templated version of it and apply multiple data types to it.

A template is a mechanism in C++ that lets you write a function or a class that uses a generic data type. A placeholder is used instead of a real type and a substitution is done by the compiler whenever a new version of the function or class is needed by your program. The compiler literally fills in the blanks in the template.

Using templates can:
- save you typing
- help you reuse old code without introducing new mistakes

You can use templates on functions or classes. Because the syntax is different for the two we will discuss them in separate sections.

Function Template

```
template <typename T>
void printIt( T a, T b)
{
        T c = a + b;
        cout << "You gave me " << a << " and " << b << ".\n";
        cout << "Together they make " << c << "." << endl;
}
```

A few notes on the syntax.
- The template <typename T> bit can be on the same line as the function type declaration, but it is usually on the line above.
- The value T stands for the type the template will be instantiated with. T can be any valid token, but watch out for namespace clashes.
- T's scope is limited to just one function, in this case printIt.
- Notice that we don't need to do anything special to make printIt work for new types.
- If a placeholder appears more than once in a function's parameter list the types you use in their place in the function call must match.

Class Template

```
template <typename M_type>
class Matrix
{
   private:
      M_type doubleArray[MAXROWS][MAXCOLS];
      int rows;
      int cols;
   public:
      Matrix();
      void printMatrix();
      void setElement(int row, int col, M_type value); //set an element of the matrix
      void setMatrix(M_type [][MAXCOLS]); //set the doubleArray to what is sent
      void addMatrix(M_type [][MAXCOLS]); //add an array to doubleArray
      void addMatrix(M_type [][MAXCOLS], M_type[][MAXCOLS]); //add two arrays together
};
```

To make an instance of a class you use this form:

```
class_name<type> variablename;
```

To create a Matrix with float you would type:

```
Matrix<float> floatMatrix;
```

The definition for a templated member function is a little surprising at first. Recall that a member function starts like this:

```
return_type class_name::function_name(parameter_list,...)
```

Pay attention to class name. The class name of a templated class is partly defined by the type it was instantiated with. The class name of the floatMatrix object above is Matrix. To refer to the class in a generic way you must include the placeholder in the class name like so:

```
template <typename T>
return_type class_name<T>::function_name(parameter_list,...)
```

Put it together

Normally when you write a C++ class you break it into two parts: a header file with the interface, and a .cpp file with the implementation. With templates this doesn't work so well because the compiler needs to see the definition of the member functions to create new instances of the templated class. Some compilers are smart enough to figure out what to do. Others have a mechanism to give them hints. These are usually the most efficient way to use templates. The most portable way is to make the templated class look like one file to the compiler. An easy way to do this is to **put all the code in the header file**.

D. Rational Class

In this part, you are required to:

i) Create a New Project and give your project the name Lab13.

ii) In this lab, you are required to modify the class **Rational** that created in Lab11. You are required to make a template out of the Rational class.

iii) A main function (in main.cpp) has been provided for you to test out your class.

At startup, it prompts the user to input two fractions. The following is the sample output.

```
Enter numerator 1: 5
Enter denominator 1: 4
Enter numerator 2: 3
Enter denominator 2: 1
```

After the input is done, the system should clear the screen and show a text-based menu:

```
--------------------
Operation:
--------------------
1.    Add
2.    Subtract
3.    Multiply
4.    Divide
5.    Exit
--------------------
```

```
Choice:
```

After reading the choice, the program should perform the operation according to the user input. Example output should look like this:

```
a + b = 17/4
a + b = 4.25
Press any key to continue ...
```

```
a – b = -7/4
a – b = -1.75
Press any key to continue ...
```

```
a * b = 15/4
a * b = 3.75
Press any key to continue ...
```

```
a / b = 5/12
a / b = 0.416667
Press any key to continue ...
```

Marking Scheme:

Graded items	Weighting
1. Correctness of program (i.e. whether your code is implemented in a way according to the requirements as specified.)	60%
2. Indentation	30%
3. Documentation (with reasonable amount of comments embedded in the code to enhance the readability.)	10%
	100%

Program Submission Checklist

Before submitting your work, please check the following items to see you have done a decent job.

Items to be checked	☑ / ☒
1. Did I put my name and student ID at the beginning of all the source files?	☐
2. Did I put reasonable amount of comments to describe my program?	☐
3. Are they all in .cpp extension and named according to the specification?	☐
4. Have I checked that all the submitted code are compliable and run without any errors?	☐
5. Did I zip my source files using Winzip / zip provided by Microsoft Windows? Also, did I check the zip file and see if it could be opened? ***(Only applicable if the work has to be submitted in zip format.)***	☐
6. Did I submit my lab assignment to Canvas?	☐

-End-