AST20105 Data Structures & Algorithms

Lab 3 – Recursive Functions

A. Submission Details

In this lab, you are required to submit **ONE** C++ program to solve the given problem shown in the section "Exercise". To make the program implementation easier, you are suggested to write your program using Visual Studio 2012 or later version instead of doing it directly using paper & pencil. After you have completed the implementation, submit your program file (i.e. BisectionMethodRecursive.cpp for this lab) using Canvas, **TWO WEEKS** after your lab section is conducted. For details, please refer to the following:

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 recursive function (recursion) taught during the lecture. Also, you will be introduced to the concept of class template, which would be useful for you to write C++ code for this course.

The first part of this lab is meant to be a briefing on the idea of class template followed by a simple review of recursion. The second part of this lab is a practical task that allows you to use the techniques taught during this lab.

Class Template

1. *Introduction to class template*

The concept of class template is very much similar to function template in which data type could be a parameter. To explain this, let's take an example.

```
// MyArray.h
#ifndef MYARRAY_H
#define MYARRAY_H

template <typename T>
class MyArray {
  private:
    T* arr;
    int size;
  public:
    MyArray(int s);
    ~MyArray();
};

template <typename T>
MyArray<T>::MyArray(int s) {
  arr = new T[s];
  size = s;
}

template <typename T>
MyArray<T>::~MyArray() {
  delete [] arr;
}

#endif
```

MyArray has 2 data members: `arr` of type `T*`, and `size` of type `int`. To create a **MyArray** class with `T` as double and an object of this class, we do:

```
MyArray<double> obj(10);
```

The first half of the statement, i.e. MyArray<double> substitutes T as double and generates the following:

```
class MyArray {
   private:
      double* arr;
      int size;
   public:
     MyArray(int s);
     ~MyArray();
};
MyArray::MyArray(int s) {
   arr = new double[s];
   size = s;
}
MyArray::~MyArray() {
   delete [] arr;
}
```

In addition, an object of this type is created, namely obj with value 10 assigned to the data member size.

2. *General form of class template*

- Syntax:

    template <typename T>

    class <class name> {

        private:

            // Data members

        public:

            // Member functions

    };

3. *General form of template class functions*

Class template functions could be declared normally, but **SHOULD BE** preceded by the statement:

```
template <typename T>
```

4. *Object creation using class template*

- Syntax:

    <class name><type> <object name>;

5. *Final remarks of class template*

All the code of class template should be placed in a header file, i.e. .h file. Separation of code is not possible for class template. If member function implementation is placed in .cpp, class template is not able to be instantiated.

D. Recursive Functions

1. ***Introduction to recursive function (Recursion)***

   As mentioned during the lecture, it might be useful to define some problems in terms of the problem itself. This may look strange at the first glance, but indeed is very useful for writing complicated programs. Most computer programming languages support this by allowing a function to call itself and this is referred as recursive function or recursion.

   Examples:

```
// Compute n!
int factorial(int n)
{
   if(n == 0 || n == 1)
     return 1;
   else
     return n * factorial(n-1);
}

// Compute x^y
double exp(double x, int y)
{
  if(y == 0)
     return 1;
  return x * exp(x, y-1);
}
```

2. ***General idea of recursive function***

   A recursive function consists of at least TWO PARTS:

   i. Base case:
      The problem is simple enough that can be solved without other help.
      For instance, factorial(0) and factorial(1) are simple enough to solve.

   ii. Recursive case:
      The problem is somehow difficult to tackle and therefore we may consider call the function itself with a small input and then combine the result to form the solution of the larger input.
      For instance, factorial(2) is relatively difficult to solve. So, we make a recursive call factorial(1) and multiply its result with value 2 to produce the final answer, i.e. 2 * factorial(1).

3. General form of recursive function

- Syntax:

```
<type> <name of recursive function>(<parameters>)
{
    if(<stopping conditions>)
        return <stopping value>;
    return <name of recursive function>(<revised parameters>);
}
```

E. Exercise – Bisection Method Implemented in Recursive Way

Root(s) of a Cubic Equation:

A cubic equation, $Ax^3 + Bx^2 + Cx + D$, always has at least one root. Within a narrow interval between x1 and x2, we can guarantee a root if f(x1) and f(x2) have different signs.

Bisection Method:

A simple bisection procedure for iteratively converging on a solution which is known to lie inside some interval [x1, x2] proceeds by evaluating the function in question at the midpoint of the original interval mid=(x1+x2)/2 and testing to see in which of the subintervals [x1, mid] or [mid, x2] the solution lies. The procedure is then repeated with the new interval as often as needed to locate the solution to the desired accuracy.

Task:

You are asked to develop a recursive version of bisection method. Your bisection method should layout all possible base case(s) and recursive case(s). Your program should:

1. Prompt users for coefficients for A, B, C, and D of a cubic equation, $Ax^3 + Bx^2 + Cx + D$.
2. Prompt users for a narrow range [x1, x2], x1 < x2.
3. Check to see if f(x1) and f(x2) have different signs.
4. If step 3 is true, invoke your recursive method.
5. Use the bisection method to find the root between the x1 and x2. You may leave your recursive function when the interval is smaller than 0.0001. If the interval is small, you may return the smaller value of the interval.

Reminder:

1. To reduce unnecessary operations, all pre-checking procedures should be done BEFORE invoking the recursive method. Pre-checking procedures are like checking x1 < x2 and f(x1) and f(x2) having different signs.
2. You may download the .cpp file provided to begin with. You may also develop your own version as long as the bisection method is implemented in a recursive way.

Sample Outputs:

Sample Output 1: $f(x) = x^3 – x – 2$

```
Input coefficients of Ax^3 + Bx^2 + Cx + D (delimited with a space): 1 0 -1 -2
Input roots x1 and x2: 1 2
Root found at 1.52051
Press any key to continue . . .
```

Sample Output 2: $f(x) = x^2 - 3$

```
Input coefficients of Ax^3 + Bx^2 + Cx + D (delimited with a space): 0 1 0 -3
Input roots x1 and x2: 1 2
Root found at 1.73145
Press any key to continue . . . _
```

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?	☐

Marking scheme:

Graded items	Weighting
1. Correctness of program	30%
2. Readability	15%
3. Re-usability	10%
4. Documentation	15%
5. Delivery	20%
6. Efficiency	10%
Total:	100%

Grading rubric:

Area of Assessment	**Exceptional**	**Acceptable**	**Amateur**	**Unsatisfactory**
Correctness	The program works and meets all the specifications.	The program works and produces the correct results and displays them correctly. It also meets most of the other specifications.	The program produces correct results but does not display them correctly.	The program is producing incorrect results.
Readability	The code is exceptionally well organized and very easy to follow.	The code is fairly easy to read.	The code is readable only by someone who knows what it is supposed to be doing.	The code is poorly organized and very difficult to read.

Re-usability	The code could be re-used as a whole or each routine could be re-used.	Most of the code could be re-used in other programs.	Some parts of the code could be re-used in other programs.	The code is not organized for re-usability.
Documentation	The documentation is well written and clearly explains what the code is accomplishing and how.	The document consists of embedded comment and some simple header documentation that is somewhat useful in understanding the code.	The documentation is simply comments embedded in the code with some simple header comments separating routines.	The documentation is simply comments embedded in the code and does not help the reader understand the code.
Delivery	The program was submitted on time.	The program was submitted within 1 day of the due date.	The code was submitted within 2 days of the due date.	The code was submitted more than 2 days overdue.
Efficiency	The code is extremely efficient without sacrificing readability and understanding.	The code is fairly efficient without sacrificing readability and understanding.	The code is brute force and un-necessarily long.	The code is huge and appears to be patched together.

Adopted from California State University (Long Beach)

http://www.csulb.edu/colleges/coe/cecs/views/programs/undergrad/grade_prog.shtml

-End-