ECE 2036 Lab 1
Exploring Numerical Error

(100 pts)

Due: Feb 2, 2020 by 11:59 PM

This lab is intended to help you explore various situations that can occur during numerical calculations. Please make sure that you look at the appendices in this lab as well for additional information.

You need to create a different application for each of the four parts of the lab. This means that you need to create separate code for each part with a main() function and any additional functions specified in the instructions.

PART 1: OUT OF RANGE ERROR (25 pts)

In C++, the fundamental type of int on the PACE-ICE system uses 32 bits to represent an integer; however, you can extend this to 64 bits by using another fundamental type in C++ called long long. (Please note that these sizes can vary from platform to platform and are not guaranteed in the C++ standard which we will talk about in class.) I would like for you to write a program to calculate the results of $\mathbf{7^n}$ for n = 1, 2, 3, 4, … using first the **int** type and then the **long long** type. Write an application showing the maximum value of **n** that can still store the results of $\mathbf{7^n}$ in an **int**, and **long long** variables without overflow. In addition to the main() function, to get practice using global functions, you need to create two **global functions** to calculate the value of $\mathbf{7^n}$. These two global functions need to have one input argument that is an **int n**. The first function needs to return an **int**, and the second needs to return a **long long** so that you can compare the ranges of these two data types.

Your main() function and the two global functions need to be in a single file called **lab1part1.cc**. Your code should generate output to the terminal in the format shown below. This output format needs to continue until one-step past the roll over point. Think about how to detect the rollover.

```
7 to the n power results using int
7^1 = 7
7^2 = 49
7^3 = 343
```
(Continue this output until one past the rollover point.)

```
7 to the n power results using long long
7^1 = 7
7^2 = 49
7^3 = 343
```
(Continue this output until one past the rollover point.)

PART 2: ROUND-OFF ERROR (25 pts)

You should know from ECE2020 that a digital representation of certain real numbers is only an approximation. For example, irrational numbers such as π, e, $\sqrt{2}$, etc... have only a finite number of digits that can be represented in a digital computer. The two main representations that we will use in C++ will be single precision (32-bit) and double precision (64-bits) numbers. In each of the number formats, a certain number of bits is dedicated to the exponent (8 bits for single precision and 11 bits for double precision), the mantissa (23 bits for single precision and 52 bits for double precision), and 1 bit for the sign bit.

You need to develop a C++ program to calculate the roots of the following quadratic equation:

$$3x^2 + 9000.003x + 9 = 0$$

Please use the following **non-conventional Muller's method** for the quadratic equation in your program to solve for the roots.

$$ax^2 + bx + c = 0$$

$$x = \frac{2c}{-b \pm \sqrt{b^2 - 4ac}}$$

You need to create a global function that has four parameters that you pass to it (i.e. a boolean flag to indicate if it should return the plus or minus solution, a, b, and c). You should make one function using only floats and the other only doubles so that you can compare the results. To calculate the square root, you can use the sqrt(x) function that is a part of the cmath library. To include this in your program you must have the following preprocessor directive. #include <cmath>

The actual exact roots of this equation are $x_1 = -0.001$ and $x_2 = -3000$. Please compare how your program calculates this with first float types and then with doubles. You can calculate the error given by:

$$\% \text{ error} = 100 * (\text{actual} - \text{approximation})/\text{actual}$$

QUESTION YOU NEED TO ANSWER: Given the fact that both roots can easily be represented with a float, why do you get a couple percent error for one of the roots? **(Answer On Turn-in sheet)**

Your code for this part needs to all be in a single file called **lab1part2.cc**. Your code needs to have the following output.

```
Using the float data type the roots are:
       x(+) = <you calculate>      % error = <you calculate>
       x(-) = <you calculate>      % error = <you calculate>

Using the double data type the roots are:
       x(+) = <you calculate>      % error = <you calculate>
       x(-) = <you calculate>      % error = <you calculate>
```

PART 3: TRUNCATION ERROR (25pts)

In numerical analysis, the term "truncation error" is typically related to the error introduced by not using all the terms in a series expansion or a limited number of iterations used in approximations to differential equation solutions. To illustrate this, I would like for you to calculate the truncation error in calculating the value of $\sqrt{2}$ using the following power series.

$$\sqrt{2} = \sum_{k=0}^{\infty} \frac{(2k+1)!}{2^{3k+1}(k!)^2} = \frac{1}{2} + \frac{3}{8} + \frac{15}{64} + \frac{35}{256} + \frac{315}{4096} + \frac{693}{16384} + \cdots.$$

You need to create code to calculate the $\sqrt{2}$ using

- 20 terms using float precision
- 80 terms using double precision

You will compare your answer to the actual value to 20 decimal places:

$$\sqrt{2} = 1.41421356237309504880$$

In your program declare two arrays:

float fTerms[20];

double dTerms[80];

to hold the individual values for each term in the power series. Calculate and store the value of each term once using float and the other using doubles. **Beware of the large numbers that result in the numerator and denominator of the terms. Design ways to deal with any overflows that occur.**

Once you have calculated the terms for both floats and double you need to sum up the terms for each type from largest to smallest. Then you need to determine the sum from smallest to largest for both types.

In theory you would think it should not make a difference which order you sum the values, but you should see a difference in the result.

Is there a difference between the two summations? Why do you think there would be a difference between the forward and backward summations? **(Answer On Turn-in sheet)**

All your source code for this part needs to be a file called **lab1part3.cc**. Sample output of your program should look like:

```
Float Results:

forward answer = <your value>

forward %error = <your value>

backward answer = <your value>

backward %error = <your value>

Double Results:

forward answer = <your value>

forward %error = <your value>

backward answer = <your value>

backward %error = <your value>
```

PART 4: Collatz Sequence (www.projecteuler.net) (25 points)

This problem is to get you used to using C++ to solve simple problems.

The following iterative sequence is defined for the set of positive integers:

n → n/2 (if n is even, divide it by 2)
n → 3n + 1 (if n is odd, then multiple it by 3 and then add one)

For example, using the rules above and starting with 13, we generate the following sequence:

13 -> 40 -> 20 -> 10 -> 5 -> 16 -> 8 -> 4 -> 2 -> 1

It can be seen that this sequence (starting at 13 and finishing at 1) contains 10 terms. Although it has not been proved yet (Collatz Problem), it is thought that all starting numbers finish at 1.

Write a program that continuously prompts the user for a positive integer (unsigned int) and then outputs the terms in the format shown below, until the term with a value of 1 is reached. Limit the number of terms per line to 10. If more than 10 terms, then start on a new line. The program exits when the user enters 0.

```
Please enter a positive number (0 to exit): 13

Terms: 13 -> 40 -> 20 -> 10 -> 5 -> 16 -> 8 -> 4 -> 2 -> 1
Total number of terms: 10

Please enter a positive number (0 to exit): -3

Error!

Please enter a positive number (0 to exit): 0

Bye!
```

APPENDIX A - Creating Lab1 Program and Submission Instructions

Creating Source Code.

You can create the code either on your personal PC or on the PACE-ICE server.

PACE-ICE:

Once logged into PACE-ICE, you will need to use one of several available text editors found on the PACE-ICE cluster. Those are vi/vim, nano and emacs. I will use nano in class, and I will recommend that you become familiar with this editor. There are many online tutorials to help you get used to nano.

Basic Unix Commands. Please make a directory (i.e. "folder") called `Lab1` where you keep your files. To make this directory use the following command in your home directory: **`mkdir Lab1`**

To go into this directory from your home directory, you can use the following command: **`cd Lab1`**

To get back to your home directory you can use the command: **`cd`**

Compiling Source Code. On either system we will be using the gnu g++ compiler. The following command is an example of how to compile your source code and create an executable file called `Part1`.

```
g++ lab1part1.cc –o Part1
```

To run your program at the command prompt:

In linux type in the executable filename `./ Part1`

In windows type `.\ Part1`

Submitting Assignments:

You will need to submit **BOTH** your **SOURCE CODE AND THE TURN IN SHEET ON CANVAS**. If you use the PACE-ICE server, you will need to download the SOURCE code from PACE-ICE to your local machine, and then upload to the Canvas assignment.

Make sure that all your C++ files have the following header format at the start of the file:

```
/*
Author:  <Your name>
Date last modified: <date last modified>
Organization: ECE2036 Class

Description:

Describe what is done in this file.

*/
```

1. I would like for you to make a directory called Lab1.
    a. To make this directory on the PACE-ICE server, type the following while you are in your home directory.

        ```
        mkdir Lab1
        ```

2. Please make all your source code files (i.e. lab1part1.cc, lab1part2.cc, lab1part3.cc, and lab1part4.cc) in this folder.
    a. To get into this folder on the PACE-ICE server you can use at the command prompt:

        ```
        cd Lab1
        ```

3. When you are finished editing and building the different parts of the lab, please do the following
    a. If you have used the PACE_ICE server then download all the contents (*.cc and built files) to your local PC.
    b. On your personal PC zip up the files using the filename **"yourusernameLab1.zip"**
4. In Appendix C, there is a turn-in sheet. Please make sure that you fill this out and upload this to Canvas as well.

Appendix A: Coding Standards

Indentation:

When using *if/for/while* statements, make sure you indent 4 spaces for the content inside those. Also make sure that you use spaces to make the code more readable.
For example:

```
for (int i; i < 10; i++)
{
    j = j + i;
}
```

If you have nested statements, you should use multiple indentions. Each { should be on its own line (like the *for* loop) If you have *else* or *else if* statements after your *if* statement, they should be on their own line.

```
for (int i; i < 10; i++)
{
   if (i < 5)
   {
       counter++;
       k -= i;
   }
   else
   {
       k +=1;
   }
   j += i;
}
```

Camel Case:

This naming convention has the first letter of the variable be lower case, and the first letter in each new word be capitalized (e.g. firstSecondThird). This applies for functions and member functions as well! The main exception to this is class names, where the first letter should also be capitalized.

Variable and Function Names:

Your variable and function names should be clear about what that variable or function is. Do not use one letter variables, but use abbreviations when it is appropriate (for example: “imag" instead of
“imaginary”). The more descriptive your variable and function names are, the more readable your code will be. This is the idea behind self-documenting code.

File Headers:

Every file should have the following header at the top

```
/*
Author: <your name>
Date Last Modified: <date last modified>
Organization: ECE2036 Class

Description:

Describe what is done in this file.

*/
```

Code Comments:

1. Every function must have a comment section describing the purpose of the function, the input and output parameters, the return value (if any).
2. Every class must have a comment section to describe the purpose of the class.
3. Comments need to be placed inside of functions/loops to assist in the understanding of the flow of the code.

Appendix C - Turn-in Sheet

Part 1: OUT OF RANGE ERROR - In addition to submitting your source code, enter the results below:

The largest power that can be represented with regular int type is _____.

The largest power that can be represented with a long long type is _____.

Part 2: ROUND-OFF ERROR - In addition to submitting your source code, enter the results below:

Using a float

x(+) = _____ %error from actual root = _____

x(-) = _____ %error from actual root = _____

Using a double

x(+) = _____ %error from actual root = _____

x(-) = _____ %error from actual root = _____

Given the fact that both roots can easily be represented with a float, why do you get a couple percent error for one of the roots?.

Part 3: TRUNCATION ERROR - In addition to submitting your source code, enter the results below:

Float Results:

forward answer = _____ forward %error = _____

backward answer = _____ backward %error = _____

Double Results:

forward answer = _____ forward %error = _____

backward answer = _____ backward %error = _____

Is there a difference between the two summations? Why do you think there would be a difference between the forward and backward summations?

Part 4: COLLATZ SEQUENCE - In addition to submitting your source code, enter the results below:

Assuming an input value of **n=21** what is the total number of terms and the sequence of terms:

terms: _____

Values of Terms:

APPENDIX D: ECE2036 Lab Grading Rubric

If a student's program runs correctly and produces the desired output, the student has the potential to get a 100 on his or her lab; however, TA's will **randomly** look through this set of "perfect-output" programs to look for other elements of meeting the lab requirements. The table below shows typical deductions that could occur.

In addition, if a student's code does not compile, then he or she will have an automatic 40% deduction on the lab. Code that compiles, but does not match the sample output can incur additional deductions from 10% to 30% depending on how poorly the output matches the output specified by the lab. This is in addition to the other deductions listed below or due to the student not attempting the entire assignment.

AUTOMATIC GRADING POINT DEDUCTIONS

Element	Percentage Deduction	Details
Does Not Compile	40%	Programs do not compile on PACE-ICE! (10% for each part)
Does Not Match Output	10%-30%	The programs compile but don't match all output
Answers on Turn-In Sheet in Appendix C	30%	Make sure you fill the turn-in sheet and post this to canvas.

ADDITIONAL GRADING POINT DEDUCTIONS FOR RANDOMLY SELECTED PROGRAMS

Element	Percentage Deduction	Details
Global Functions	10%	Does not use any global functions.
Clear Self-Documenting Coding Styles	5%-15%	This can include incorrect indentation, using unclear variable names, unclear comments, or compiling with warnings. (See Appendix E)

LATE POLICY

Element	Percentage Deduction	Details
Late Deduction Function	score - (20/24)*H	H = number of hours (ceiling function) passed deadline note : Sat/Sun count has one day; therefore H = $0.5*H_{weekend}$

Appendix E: Accessing PACE-ICE Instructions

ACCESSING LINUX PACE-ICE CLUSTER (SERVER)

To access the PACE-ICE cluster you need certain software on your laptop or desktop system, as described below.

Windows Users:

Option 0 (Using SecureCRT)- THIS IS THE EASIEST OPTION!

The Georgia Tech Office of Information Technology (*OIT)* maintains a web page of software that can be downloaded and installed by students and faculty. That web page is:

http://software.oit.gatech.edu

From that page you will need to install SecureCRT.

To access this software, you will first have to log in with your Georgia Tech user name and password, then answer a series of questions regarding export controls.

Connecting using SecureCRT should be easy.

- Open SecureCRT, you'll be presented with the "Quick Connect" screen.
- Choose protocol "ssh2".
- Enter the name of the CoC machine you wish to connect to in the "HostName" box (i.e. *coc-ice.pace.gatech.edu*)
- Type your username in the "Username" box.
- Click "Connect".
- A new window will open, and you'll be prompted for your password.

Option 1 (Using Ubuntu for Windows 10):

Option 1 uses the Ubuntu on Windows program. This can only be downloaded if you are running Windows 10 or above. If using Windows 8 or below, use Options 2 or 3. It also requires the use of simple bash commands.

1. Install Ubuntu for Windows 10 by following the guide from the following link:

   https://msdn.microsoft.com/en-us/commandline/wsl/install-win10

2. Once Ubuntu for Windows 10 is installed, open it and type the following into the command line:

   *ssh **YourGTUsername**@ coc-ice.pace.gatech.edu* where **YourGTUsername** is replaced with

   your alphanumeric GT login. Ex: bkim334

3. When it asks if you're sure you want to connect, type in:
   yes

   and type in your password when prompted (Note: When typing in your password, it will not show any characters typing)

4. You're now connected to PACE-ICE. You can edit files using vim by typing in:

vi filename.cc OR *nano vilename.cpp*

For a list of vim commands, use the following link:

https://coderwall.com/p/adv71w/basic-vim-commands-for-getting-started

5. You're able to edit, compile, run, and submit your code from this command line.

Option 2 (Using PuTTY):

Option 2 uses a program called PuTTY to ssh into the PACE-ICE cluster. It is easier to set up, but doesn't allow you to access any local files from the command line. It also requires the use of simple bash commands.

1. Download and install PuTTY from the following link: www.putty.org
2. Once installed, open PuTTY and for the Host Name, type in:
   coc-ice.pace.gatech.edu and for the port, leave

   it as 22.
3. Click Open and a window will pop up asking if you trust the host. Click Yes and it will then ask you for your username and password. (Note: When typing in your password, it will not show any characters typing)
4. You're now connected to PACE-ICE. You can edit files using vim by typing in: *vim filename.cc* OR
   nano vilename.cpp

   For a list of vim commands, use the following link:

   https://coderwall.com/p/adv71w/basic-vim-commands-for-getting-started
5. You're able to edit, compile, run, and submit your code from this command line.

MacOS Users:.

Option 0 (Using the Terminal to SSH into PACE-ICE):

This option uses the built-in terminal in MacOS to ssh into PACE-ICE and use a command line text editor to edit your code.

1. Open Terminal from the Launchpad or Spotlight Search.
2. Once you're in the terminal, ssh into PACE-ICE by typing:

   *ssh **YourGTUsername**@ coc-ice.pace.gatech.edu* where **YourGTUsername** is replaced with

   your alphanumeric GT login. Ex: bkim334
3. When it asks if you're sure you want to connect, type in:

yes

and type in your password when prompted (Note: When typing in your password, it will not show any characters typing)

4. You're now connected to PACE-ICE. You can edit files using vim by typing in:

   vi filename.cc OR *nano filename.cpp*

   For a list of vim commands, use the following link: https://coderwall.com/p/adv71w/basic-vim-commands-for-getting-started

5. You're able to edit, compile, run, and submit your code from this command line.

Linux Users:

If you're using Linux, follow Option 0 for MacOS users.