EECE 1080C / Programming for ECE

Summer 2019

Laboratory 3: C++ Loop Types

Plagiarism will not be tolerated:

all students who share files will receive a 75% penalty on this assignment

Topics covered:

- C++ program development practice.
- Arithmetic, comparative and logical operators.
- Loop types.

Objective:

- To practice program development by using loops in C++. The student will create several short programming assignments to create a solid understanding of how looping concepts work in programming environments.

Highlights:

- To receive full credit for this laboratory please sign the attendance sheet.
- Please access the laboratory assignment via the canopy/blackboard link. The descriptions for each problem are contained within this document.
- **Submit on Blackboard using the assignment dropbox.**

Grading:

- Each part should be worked on separately. You will need a separate project for each part of this assignment when working within your IDE.

Rubric: 100 points

- Part A = 20
- Part B = 10
- Part C = 15
- Part D = 15
- Part E = 20
- Part F = 20

Tasks:

A. Basic Loops

- Write a program that completes three separate loops, based on the instructions for each.

Loop #1

- Write a **for loop** that displays the sum and floating-point average of values from 2 up to and including 35 with a loop interval (step count) of 3.

Result:

```
Sum: 222
Average: 18.5
```

Loop #2

- Ask the user for 1) the starting index, 2) ending index, and 3) the loop interval (step count).
- Write a **for loop** *similar to Loop #1 using the user inputs.* Again, output the average and sum.

- **Example:**

```
Starting index: 0
Ending index: 10
Step count: 5
```

```
Sum: 15
Average: 5
```

Loop #3

- Ask the user for a new starting index (greater than zero)
- Write a **for loop** that counts down from the starting point to 0 in loop steps of 0.5 and outputs a comma separated list of values without a trialing comma.
- Insert a **time delay** into the loop that will count down at 1 second intervals

- **Example:**

```
Input value: 5
```

```
Output is:
5, 4.5, 4, 3.5, 3, 2.5, 2, 1.5, 1, 0.5, 0
```

Hint to time delay:

You will need to use the **time.h** library to create the time delay function:

```
#include<time.h>
void timeDelay(int sec)
{for (time_t t = time(0) + sec; time(0) < t; ) {}}
```

Then you can use the function inside the for loop with timeDelay(1)

B. Data entry

- Write a program including a **while loop** that allows the user to enter floating-point numbers greater than zero, adds these numbers and prints the sum after the loop. The loop should exit after a number less than or equal to 0 is entered.

- **Example:**

```
Enter positive number to add or enter zero or negative number to end: 99
Enter positive number to add or enter zero or negative number to end: 67.5
Enter positive number to add or enter zero or negative number to end: 0

Sum of entered numbers: 166.5
```

C. Rectangle and Frame

- First
  - ask the user for a width and a height, and a keyboard character
  - create a double loop to display a **solidly filled rectangle** of the width and height given by the user using the chosen character
- **Example 1:**

```
Width: 50
Height: 5
Character: *
```

```
**************************************************
**************************************************
**************************************************
**************************************************
**************************************************
```

- Second, modify the program
  - ask the user for a second keyboard character
  - change the loop to display a **framed rectangle** of the width and height given by the user.
  - the first character is the border; the second character is the inside

- **Example:**

```
Width: 8
Height: 5
Outside Character: *
Inside Character: $
```

```
********
*$$$$$$*
*$$$$$$*
*$$$$$$*
********
```

D. Object height over time

- Write a program that creates a table to show the change in height of an object launched straight up for each second from launch time (time zero) until the object reaches the ground (height less than or equal to zero). The height after **t** seconds is given by:

$$y = V_0 t - \frac{1}{2} g t^2$$

  Where:

  - V_0 is the **initial** velocity in m/s.
  - g is the gravitational acceleration and has a value of 9.80665 m/s^2.
  - t is the instant time in seconds.

- The program should prompt the user for the **initial** velocity.
- The program should exit when the object returns to the ground
- Format your output to that it appears the same as the example.

- **Example:**

```
Enter the initial velocity V0: 60
```

```
Initial Velocity of Object: 60 m/s

     Time              Height
     0                 0
     1                  55.0967
     2                  100.387
     3                  135.87
     4                  161.547
     5                  177.417
     6                  183.48
     7                  179.737
     8                  166.187
     9                  142.831
     10                 109.668
     11                 66.6977
     12                 13.9212
     13                 0
             Total Time: 13-seconds
Maximum Height: 183.48 @ 6-seconds
```

E. Metal expansion table

- In this section, you will create a table of metal expansion versus temperature. You will determine if the metal expansion is within a certain tolerance.
- The fact that most metals expand when heated and contract when cooled has implications when laboratory equipment is involved. The expansion of a typical aluminum bar that is **w** [cm] wide at $70°F$ can found using the following equation:

$$x = w + 0.0001(T - 70)$$

  where T is the specified temperature of the bar.
- Write a program that prompts the user for **w** [cm] the tolerance (plus or minus the standard bar width). Display the modified width from 60 to $85°F$ in $1°F$ intervals marking with a star the temperatures which lay within the tolerance: (**w**-t < x < **w+t**)
- Format your output to that it appears the same as the example.
- **Example:**

```
Width: 10
Tolerance: .00050
```

Temperature	Width	Within Tolerance
60	9.9990000	
61	9.9991000	
62	9.9992000	
63	9.9993000	
64	9.9994000	
65	9.9995000	
66	9.9996000	*
67	9.9997000	*
68	9.9998000	*
69	9.9999000	*
70	10.0000000	*
71	10.0001000	*
72	10.0002000	*
73	10.0003000	*
74	10.0004000	*
75	10.0005000	
76	10.0006000	
77	10.0007000	
78	10.0008000	
79	10.0009000	
80	10.0010000	
81	10.0011000	
82	10.0012000	
83	10.0013000	
84	10.0014000	
85	10.0015000	

- **Try This:** C++ has a library called **iomanip** that formats the output of this code.

  Example

```
#include <iomanip> // add to top !!! before int main() !!!
double x = 999.999;
cout << fixed << setw(11) << setprecision(7) << x << endl;
Will display 999.9990000
```

F. Incremental Mathematics

Factorial

- The factorial of a **positive integer n** is denoted $n!$ and it is the product of all positive integers less than or equal to **n**:

$$n! = \prod_{i=1}^{n} i = 1 \cdot 2 \cdot 3 \cdots (n-2) \cdot (n-1) \cdot n$$

  For example:

$$5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 120$$

  Note that, by definition:

$$0! = 1$$

$$1! = 1$$

- Calculate the factorial using a **for loop** inside a function called **factorial(int n)**. Make sure your program works with the numbers zero and one
- **Example:**

```
Enter positive integer: 5

5! = 120
```

Exponential

- The Natural Exponential Function $\mathbf{e^x}$ can be found using Taylor Series, and it is defined by the following mathematical expression:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$$

- Your task is to convert this expression into C++ code: use your function **factorial(int n)**, inside another function called **exp(double x)** that loops through each term.
- You should calculate as many terms as necessary to get an accurate answer (but less than 20).
- Check the accuracy by taking the difference between the **sum at the beginning of a cycle** and the **sum at the end of a cycle.**
- The difference should be less than 0.0001 (ie. abs(sum_begin – sum_after) < 0.0001)
- Inform the user of the solution is found before 20 iterations are complete, as well as the number of calculations the program ran before an acceptable solution was found.

- **Example:**

```
Enter value for x: 1

Solution converged before reaching 20 calculations
Number of calculations: 8
e^1 = 2.718281828
```