Lab 1: Introduction to C++

CPSC 1021

School of COMPUTING Clemson University

Lab Objective

- Practice writing programs using basic C++ syntax
- Review basic programming techniques: control flow, looping, and functions
- Practice program design:
  - using pseudocode to write plain language description of programming solutions
  - writing a test plan
- Getting familiar with the 1020 autograder in Gradescope

Introduction

In this lab, you will develop solutions for three programming problems:

1. A menu-driven program that finds and displays areas of three different objects.
2. A points reward program for Clemson University's new bike-to-campus initiative
3. A compound interest calculator to calculate retirement savings.

For each programming problem, you will submit (1) a design document and test plan via the 1021 Canvas page, (2) a C++ program via the Gradescope autograder (which you can access through the 1021 Canvas page).

Collaboration Policy

Work with your assigned team members from Lab 0 on each problem, but with no more than one or two other students. If you have been assigned to a team of four in Lab 0, split into two pairs; if you have been assigned to a team of three, you can work together as one group. *Please make sure to integrate your team members who – because of COVID requirements – have to participate remotely.* You can use the MS Teams video conferencing feature to collaborate with a remote team member. Use the search bar on the top of the MS Teams app to search for your team member with their CU user name, then click the video icon in the top-right corner of the screen to start a video call.

You and your partner(s) can work on all assignments together, but every student has to submit their own files.

Design and Submission Requirements

Before you start coding, it is essential that you think about the design and functionality of your program. What is the purpose of your program? What information will be put into your program? What processing will take place? What is the desired output of your program? (*If you haven't done so yet, read Chapter 1.6 "The Programming Process" in our textbook* .) Designing a program also includes thinking about ways that you can test it. What is the program's expected output *y* when the user enters *x*? What happens when the user enters incorrect data, e.g. a negative number where a positive number is expected?

To practice program design and develop good programming habits, you are *not* allowed to start coding *until* you have submit a design and test plan. For this week's programming problems, your design and test documents need to consist of two parts:

1. A plain language description of your program in pseudocode
2. A table showing details for at least five test cases

(*Hint*: You can find an example of how to write detailed pseudocode in Chapter 1.6 of our textbook on the bottom of p. 20; you can find an example of a test plan in Chapter 5.13 on p. 306.)

We have uploaded a Word template for the design and test plan to our CPSC 1021 Canvas page. You can also use **pen-and-paper** or one of the **whiteboards** in our lab classroom to work on your program design. In fact, many professional programmers prefer pen-and-paper and whiteboard sketching over electronic tools during the initial phase of developing a new program. We highly encourage you and your team member to pick the tool that works best for you. If you design your program on paper/whiteboard, you can take a picture of your design and copy-paste it into the Word template uploaded to Canvas. Just make sure your picture has a high-enough resolution so that your pseudocode is readable (and *please* write clearly and legibly).

After you have submitted your design and test plan for a programming problem, you can start coding. When you have completed your program, **use the test cases from your test plan to check your program for potential errors**. Once you are satisfied that your program is working correctly, submit it via Gradescope (linked on our 1021 Canvas page).

Each of the following programming problems has detailed submission instructions. **Please carefully read these instructions and follow them exactly**. As a general reminder: the autograder will compile and run your program on a Linux machine. Make sure your program compiles and runs on the SoC Linux machines before you submit it. Submissions that include code that doesn't compile are automatically assigned 0 points.

Problem 1: Areas Calculation

Create a menu-driven program that finds and displays areas of three different objects.

The menu should have the following 4 choices:

1 -- square

2 -- circle

3 -- right triangle

4 -- quit

- If the user selects choice 1, the program should prompt the user for input and, based on the user's input, find the area of a square.
- If the user selects choice 2, the program should prompt the user for input and, based on the user's input, find the area of a circle.
- If the user selects choice 3, the program should prompt the user for input and, based on the user's input, find the area of a right triangle.
- If the user selects choice 4, the program should quit without doing anything.
- If the user selects anything else (i.e., an invalid choice) the program should terminated with the following error message: `You entered an invalid choice. Good bye!`

Sample Run

```
Program to calculate areas of objects

        1 -- square

        2 -- circle

        3 -- right triangle

        4 -- quit

2

Radius of the circle: 3.0

Area = 28.27
```

Code Template

```
 1 // WRITE A COMMENT BRIEFLY DESCRIBING THE PROGRAM.
 2 // PUT YOUR NAME HERE.
 3 // INCLUDE ANY NEEDED LIBRARIES AND FILES HERE.
 4 using namespace std;
 5
 6 int main()
 7 {
 8    // DEFINE THE NAMED CONSTANT PI HERE AND SET ITS VALUE TO 3.14159
 9
10    // DECLARE ALL NEEDED VARIABLES HERE. GIVE EACH ONE A DESCRIPTIVE
```

```
11    // NAME AND AN APPROPRIATE DATA TYPE.
12
13    // WRITE STATEMENTS HERE TO DISPLAY THE 4 MENU CHOICES.
14
15    // WRITE A STATEMENT HERE TO INPUT THE USER'S MENU CHOICE.
16
17    // USE AN IF/ELSE IF STATEMENT TO OBTAIN ANY NEEDED INPUT INFORMATION
18    // AND COMPUTE AND DISPLAY THE AREA FOR EACH VALID MENU CHOICE.
19    // ASSUME THE USER ENTERS VALID METRICS, E.G. DOESN'T ENTER
20    // A STRING OR NEGATIVE NUMBER.
21    // IF AN INVALID MENU CHOICE WAS ENTERED, AN ERROR MESSAGE SHOULD
22    // BE DISPLAYED.
23
24    return 0;
25 }
```

Submission instructions

- Name your file areaCalc.cpp; this is the only file you need to submit
- Make sure your program compiles and runs on the SoC Linux machines; the autograder automatically assigns 0 points to programs that do not compile.
- Your program must produce output formatted as follows:
  - for menu choice 1 and length of the square 2:
    ```
    Area = 4.00
    ```
  - for menu choice 2 and radius 3.0:
    ```
    Area = 28.27
    ```
  - for an invalid menu choice:
    ```
    You entered an invalid choice. Good bye!
    ```

Problem 2: Bike-to-Campus Reward Points

A new bike-to-campus initiative at Clemson University awards points to students for taking their bike to school instead of their car.

Points are awarded as follows:

Number of Bike Rides	Points Earned
0	0
1	3
2	10
3	15
4	30
5 or more	50

Write a program that asks the user to enter the number of bike rides to campus and then displays the number of points awarded. You can assume that the user always enters an integer. Your program must use input validation: if a user enters a number <0, the program should NOT terminate with an error message. Instead, the program should alert the user that the input was invalid and ask the user to enter a new number.

Submission instructions

- Name your file bikeRidePoints.cpp; this is the only file you need to submit
- Make sure your program compiles and runs on the SoC Linux machines; the autograder automatically assigns 0 points to programs that do not compile.
- Your program must produce output formatted as follows:
  - for input 0: "You earned 0 points this semester"
  - for input 1: "You earned 3 points this semester"
  - etc.

  (The message displayed to the user after entering a number <0 is your choice; the exact wording of this message will not be checked by the autogradder.)
- Code style (proper indentation, consistency, readability, use of comments, etc.) will be considered when grading your submission

Problem 3: Retirement Calculator

Once you have started your first full-time employment, you should start saving for retirement. Because of compound interests, even a small amount saved early can grow significantly if interest is accrued over a long time period. To illustrate the power of compound interests, you will build a simple retirement calculator that will take the following input from a user:

- P: principal / present value
- r: interest rate
- t: number of years the money will be left in the account

Based on these user inputs, retirement savings (RS) are calculated as follows:

$$RS = P \times (1 + r)^t$$

Write a program that prompts the user for the above input values. The program should pass these values to a function, `compoundCalc()`, that computes and returns the future value of the retirement account after *t* years. The program should display that value to the user. Implement function `compoundCalc()` in its own .cpp file and write an appropriate header file. The function should take as input values for p, r and t, in this order.

Sample Run 1

```
(user prompt to enter P): 1000
(user prompt to enter r as a decimal): 0.025
(user prompt to enter t): 20
Your retirement savings will be $1638.62
```

Sample Run 2

```
(user prompt to enter P): 1000
(user prompt to enter r): 0.025
(user prompt to enter t): 40
Your retirement savings will be $2685.06
```

Submission instructions

- Your submission must include three files: retirementCalc.cpp (this file includes `main()`), compoundCalc.cpp (this file includes `compoundCalc()`), and compoundCalc.h
- Make sure your program compiles and runs on the SoC Linux machines; the autograder automatically assigns 0 points to programs that do not compile.
- Your program must produce output formatted as shown in the above sample runs, e.g.:
  `Your retirement savings will be $1638.62`
- Code style (proper indentation, consistency, readability, use of comments, etc.) will be considered when grading your submission