1 Instructions

You may work with one partner on this lab project if you wish or you may work alone (that is, a group or team of two students) . If you work with a partner, only one of of you will submit the lab project to Canvas for grading. To ensure you each receive the same score, it is imperative that you follow the instructions in §4 *Submission Information* (i.e., put both of your ASURITE ID's in the submission archive filename and write both of your names, ASURITE ID's, and email addresses in the AUTHOR1: and AUTHOR2: lines of the header comment block for each source code file). What to submit for grading, and by when, is discussed in §4; read it now.

2 Learning Objectives

After completing this assignment the student shall be able to,

- Use a C++ integrated development environment to to create a C++ **project**, add a C++ **source code file** to the project, **edit** and **compile** the source code file to eliminate all **syntax errors**, **run** the program, and **verify** the correctness of the output.
- Use the ***cout*** statement and the **stream insertion operator <<** to display a **string literal** to the output window.
- Use the ***cin*** and the **stream extraction operator >>** to read double values from the keyboard.
- Declare and use **double and string variables** and use the basic **arithmetic operators** to perform calculations.
- Update/write **comments** to document the code.
- **Upload** the project file to Canvas for grading.

3 Lab Exercise

Your TA will show you how to use a very simple online C++ software development environment at a site named named Repl.it (RIT) to complete this project. The primary advantage of Repl.it is that all of your software development is done online and you do not need to install any files on your own computer. Additionally, you can access your project files and write your code from any internet-connected computer.

3.1 Creating your First C++ Program

In Repl.it, create a new C++ project named **lab01**. When your project is created, RIT will create a source code file named *main.cpp* which contains the code for the Hello World program. Select all of the text in the file and delete it. Then enter the code shown below exactly as it appears (well, you do not need to count the stars) with one exception: if you work alone, omit the AUTHOR2: comment line and include your information in the AUTHOR1: comment. However, if you work with a partner, it is imperative that you include both author's information in the AUTHOR1: and AUTHOR2: comments. We instruct the graders to look for two AUTHOR lines and when found, they will assign each author the same score.

```
//**************************************************************************
// FILE: main.cpp
//
// DESCRIPTION
// This program does x, y, and z. Then it explodes with such force that it
// triggers massive world-wide geothermal volcanic activity that ultimately
// results in the almost complete extinction of all life on Earth with the
// exception of members of the order Blattodea, which have always been destined
// to someday rule the world anyway.
//
// AUTHOR1: your-name, your-asurite-id, your-email-address
// AUTHOR2: your-name, your-asurite-id, your-email-address   note: omit this line if you work alone
//
// COURSE INFO
// CSE100 Principles of Programming with C++, Fall 2020
// Lab Project 1   Day/Time: your-lab-date-and-time   TA: your-lab-ta-name
//**************************************************************************
#include <iostream>

using namespace standard;

int main()
{
   cout << "Hello World!" << endl;
   return 0;
}
```

Note that the number of space characters between the left margin and the beginning of the lines starting with *cout* and *return* can vary. The spaces are referred to as *indentation*. I indent these types of lines using 4 spaces but the number does not have to be 4. Some programmers indent 2 spaces, some 3, some 4, some 5, some 8, etc. If you search the internet for the "acceptable number" of spaces to indent, you will find passionate arguments from various programmers of why the number of spaces *they* indent is considered proper. Generally, it is widely accepted that 2 to 8 spaces is acceptable, with many programmers agreeing that around 4 to 5 is optimal. The main point is to choose a number for how many spaces you are going to indent and then be consistent in writing the code for a project so you always using the same indentation amount (of course, in a different project, you could certainly choose a different number) The number of spaces which are inserted when you press the Tab key can be configured in RIT by clicking on the gear symbol on the far left side of the browser window and then entering the desired integer for the *Indent Size* configuration setting.

3.2 Building Hello World and Syntax Errors

This program is called the *Hello World* program because all it does when it is run is print the phrase "Hello World!" on the *output window pane* to the far right of the browser window. *Output window pane* or simply *output window* is the term we shall use to refer to the window area where the output from your program is displayed. The output window acts as the interface between the user and the program. Other common terms for the output window are: the *console window*, the *terminal window*, the *console*, or the *terminal*.

The *Hello World* program is the simplest C++ program you can write that does anything which is visible to the user. As we discussed in the lecture, C++ code cannot be directly executed by the CPU. First, the code has to be translated into the low-level binary machine language code required by the processor. To do this, we must *compile* the C++ code. Compiling the code of a project so it can be run is often called *building*. To build the project, click the *Run* button in the menu strip above the text editor pane.

When we type code, we inevitably make syntax errors. If you typed the *Hello World* code exactly as it appears above, then your code will contain an intentionally-planted *syntax error*. Syntax errors are displayed in the *output window* pane. For example, I have intentionally made a mistake in the code I gave you by having you write *standard* rather than *std* on line 21. When I build my program, this is what I see in the *Results* pane.

```
clang++-7 -pthread -std=c++17 -o main main.cpp
main.cpp:3:17: error: expected namespace name
using namespace standard;
                ^
main.cpp:6:3: error: use of undeclared identifier 'cout'; did you mean
      'std::cout'?
  cout << "Hello World!" << endl;
  ^~~~
  std::cout
/usr/bin/../lib/gcc/x86_64-linux-gnu/7.5.0/../../../../include/c++/7.5.0/iostream:61:18: note:
      'std::cout' declared here
  extern ostream cout;          /// Linked to standard output
                 ^
main.cpp:6:29: error: use of undeclared identifier 'endl'; did you mean
      'std::endl'?
  cout << "Hello World!" << endl;
                            ^~~~
                            std::endl
/usr/bin/../lib/gcc/x86_64-linux-gnu/7.5.0/../../../../include/c++/7.5.0/ostream:590:5: note:
      'std::endl' declared here
    endl(basic_ostream<_CharT, _Traits>& __os)
    ^
3 errors generated.
compiler exit status 1
```

Ouch! The C++ compiler has reported three syntax errors. Do not be intimidated by the number of error messages the compiler generates. As we discussed in the lecture, just one simple typing mistake in the code can cause the compiler to generate many syntax errors. So, we always focus on fixing the very first error message and ignoring the rest because fixing the first error may fix the mistake that was causing the compiler to report all of the remaining error messages.

Starting with the our first syntax error, we see *main:3:17 error: expected namespace name*. The **3** and **17** following the filename are the row number and column number within row 3 where the compiler became confused. Then below that, the compiler displays the code on line 3 with a ^ symbol pointing to the character in column 17.

Recall from the lectures that a syntax error is a mistake in the code that violates one or more of the syntax rules of the C++ language. This type of error prevents the program from being built. I want to mention a couple of important points about syntax errors. First, when the compiler reports a syntax error, it will list a source code file line number and a message. The message can be abstruse—especially to beginners—and is sometimes just flat out wrong. This is because—depending on the specific mistake in the code—it can be very difficult for the compiler to identify the actual error and its location. As it is compiling, it determines that something is wrong, because it is not seeing the code it expects to see, but it does not know exactly what went wrong. Therefore, it will make its best guess and report what it thinks went wrong, when in fact, this might not be the truth. Its just a limitation of current compiler technology. And that leads to my second comment. Just because the compiler reports an error on, say, line 413 of a source code file does not mean the error is actually on line 413. Rather, it simply means that there was an error somewhere on lines 1 to 413 and it was not until the compiler reached line 413 that it figured out that something was amiss. Again, its just a limitation of the compiler that we have to live with. Therefore, knowing and remembering these two points is very important.

When your code has syntax errors, you must locate and fix them in the text editor and keep rebuilding until you have no syntax errors. Go to the editor pane and change **standard** to **std** and rebuild. Assuming you made no typos when entering the code, you will know you have a successful build when the program runs and displays its output in the output window pane.

Note that all of the programs we write in CSE100 will be text-based programs (also called **console-based programs**)—i.e., our programs will display only letters, numbers, and other symbols that you will find on the keyboard—for two reasons: (1) Writing programs which use graphics is significantly more difficult and time-consuming than writing text-based programs; and (2) The C++ Standard Library does not contain any functionality to support graphical programming. That does not mean it is not possible to write graphical-based C++ programs. There are many available graphics libraries that are compatible with C++, but learning to use most of these libraries is not trivial. Almost everyone who learns to program, and especially in C++, starts with console-based programs because there is so much fundamental C++ which must be acquired first before moving on to more intermediate-level programs using graphics.

3.3 Beyond Hello World – Lab Project 1 Software Requirements

Okay, enough with the easy stuff. Let us move on to a slightly more advanced program. For your lab project, we want you to modify the source code file so the program will operate as described below. These are the **software requirements** for the project:

1. When the program runs, it shall display the text as shown on lines 1-11 in the example below.
2. Next, it shall display a prompt message (or simply a **prompt**) asking the user to enter his or her first name.
3. Next, it shall display a prompt asking the user to enter his or her yearly salary.
4. Next, it shall shall read the user's salary from the keyboard (in the example below, user input is in **bold**).
5. Next, it shall shall calculate the user's monthly salary by dividing the yearly salary by 12.
6. Next, it shall shall calculate the user's biweekly salary by dividing the yearly salary by 26.
7. Next, it shall shall display the user's yearly salary as shown on line 13.
8. Next, it shall shall display the user's monthly salary as shown on line 14.
9. Next, it shall shall display the user's biweekly salary as shown on line 15.
10. And finally, it shall end by displaying the message shown on line 16.

The output from your project should appear like this, with the exception that the numbers will vary depending on what value the user enters for his or her yearly salary.

```
[001] Hello, this is Lab Project 1. There are several goals:
[002]
[003] 1. Learn how to use the Repl.it Integrated Development Environment (IDE).
[004] 2. Learn how to locate and correct syntax errors in source code.
[005] 3. Learn how to build and run my program.
[006] 4. Learn the basic template for every C++ program.
[007] 5. Learn how to write cout statements.
[008] 6. Learn how to write cin statements.
[009] 7. Learn how to define and use double and string variables.
[010] 8. Learn how to use the basic arithmetic operators.
[011]
[012] What is your first name? Cletus
[013] What is your yearly salary? 70000
[014] Cletus, your yearly salary is $70000.00
[015] Your monthly salary is $5833.33
[016] Your biweekly salary is $2692.31
[017] Cletus, this was a blast. Let's do it again sometime.
```

3.4 Testing

Just because a program compiles does not mean that it is perfect. All it means is that there are no syntax errors in the code but the program might now meet any of the software requirements, or in other words, the program might not do what it is supposed to do. Consequently, testing as one of the most important of the five software development stages we will discuss.

Testing is a large area of study and is a complicated topic. Software engineering majors often must complete at least one course one software quality assurance (testing) and sometimes more than one. However, for our programs and especially this small program, the testing will be easy to perform. The testing process we shall follow is:

1. Before you design or implement the software, we will create one or more *test cases*. A test case is simply the data that we shall use as input when we run the program, along with the expected program behavior or output. To ensure the program meets the software requirements, there should be at least one test case testing every software requirement. Here is an example test case for software requirement 1 (SWR1).

   Test Case 1
   Software Requirements: SWR1
   Description: Tests that the program displays the messages specified in SWR1.
   Input Data: None
   Expected Output (we write this section before writing the program):

   ```
   1. Learn how to use the Repl.it Integrated Development Environment (IDE).
   2. Learn how to locate and correct syntax errors in source code.
   3. Learn how to build and run my program.
   4. Learn the basic template for every C++ program.
   5. Learn how to write cout statements.
   6. Learn how to write cin statements.
   7. Learn how to define and use double and string variables.
   8. Learn how to use the basic arithmetic operators.
   ```

   Actual Output (after we write the program, we run it, and this is the output the program produced):

   ```
   1. Learn how to use the Repl.it Integrated Development Environment (IDE).
   2. Learn how to locate and correct syntax errors in source code.
   3. Learn how to build and run my program.
   4. Learn the basic template for every C++ program.
   5. Learn how to write cout statements.
   6. Learn how to write cin statements.
   7. Learn how to define and use double and string variables.
   8. Learn how to use the basic arithmetic operators.
   ```

   Test Case Result: **Passed** (write **passed** if the expected output exactly matches the actual output; otherwise the test case **failed**)

   The test case consists of a description of what software requirement is being tested, any input data, and the expected output from the program given the input data. When we actually perform the test case by running the program and giving it the input data, we document the program's behavior or output in the *Actual Output* section. Finally, we verify that the actual output matched the expected output and if so, we say the test case *passed*; otherwise it *failed*.

   Of course, failure is bad because it means the program failed to meet one or more of the software requirements, i.e., it did not do what it was supposed to do. But failure is also good because it means we located a *logical error* or *bug* in the code. So the next step is to *debug* the code to locate the error and correct it. Once we do that, we must perform the test case over again to verify that the program now correctly meet the software requirement.

   Here are some two tests cases for Lab Project 1 which test that the monthly and biweekly salaries are correct.

   Test Case 2
   Software Requirements: SWR 3-5, 8
   Description: Tests that the program correctly calculates and displays the user's monthly salary.
   Input Data: 654321
   Expected Output: the monthly salary shall be displayed as $54526.75.
   Test Case Result: *document if the test case passed or failed*

Test Case 3
Software Requirements: SWR 3, 4, 6, 9
Description: Tests that the program correctly calculates and displays the user's biweekly salary.
Input Data: 654321
Expected Output: the biweekly salary shall be displayed as $25166.19
Test Case Result: *document if the test case passed or failed*

Just because the program ran correctly on one set of input data, it is always a good idea to test it on two or three different sets. Here are two more test cases you should perform.

Test Case 4
Software Requirements: SWR 3-5, 8
Description: Tests that the program correctly calculates and displays the user's month salary.
Input Data: 0
Expected Output: the monthly salary shall be displayed as $0.00
Test Case Result: *document if the test case passed or failed*

Test Case 5
Software Requirements: SWR 3, 4, 6, 9
Description: Tests that the program correctly calculates and displays the user's biweekly salary.
Input Data: 0
Expected Output: the biweekly salary shall be displayed as $0.00
Test Case Result: *document if the test case passed or failed*

And one more set for good measure,

Test Case 6
Software Requirements: SWR 3-5, 8
Description: Tests that the program correctly calculates and displays the user's month salary.
Input Data: 1212121212
Expected Output: the monthly salary shall be displayed as $10101010**1**.00
Test Case Result: *document if the test case passed or failed*

Test Case 7
Software Requirements: SWR 3, 4, 6, 9
Description: Tests that the program correctly calculates and displays the user's biweekly salary.
Input Data: 12121212
Expected Output: the biweekly salary shall be displayed as $466200.46
Test Case Result: *document if the test case passed or failed*

3.5 Additional Programming Requirements

1. In a C++ source code file, **comments** start with // which indicates the beginning of the comment. The comment continues to the end of the line of text. Comments are not C++ code (as we discussed in the lecture, the C++ compiler never even sees them, as they are removed by the C++ preprocessor). Rather, the purpose of comments is to document the code for human readers. A **comment header block** is a block of comments that appears at the top (the head) of a source code file, e.g., see the program on page 1 with the header comment block on lines 1-17. Conventionally, most programmers will place a comment header block at the top of each source code file they create. Common information written in the header comments include: the name of the source code file; a description of what the program does; the author(s) information; copyright information; modification history; and a version or revision number. Before submitting your source code file for grading, update the information in the **comment header block** at the top of your **main.cpp** source code with your author information, your partner's author information if you worked with a partner, the day and time your of your lab, and your lab TA's name.

2. Carefully **format** your code and consistently **indent** the statements inside *main*() using anywhere from 3-5 spaces, as shown in the example programs of the textbook and the lecture notes.

4 Submission Information

- For help information about how to use Canvas, please visit the Canvas Student Guide.
- You are permitted to work in a group of 2 students. In Canvas, I have created 125 groups, each limited to a maximum of two students. You may work alone or if you wish to work on a team of two, then read this help documentation about how to join a Canvas group in this page of the Student Guide, and then the two team members can each add themselves to an empty group.
- When you are ready to submit your Lab 1 project, begin by using the file explorer program of your operating system to create a new empty folder named **lab01-*asurite*** where *asurite* is your ASURITE user id that you use to log in to MyASU (e.g., mine is *kburger2* so my folder would be named **lab01-kburger2**). If you worked with a partner in a team of two on the project, put both of your ASURITE id's in the name of the new folder, e.g., **lab01-*asurite1-asurite2*.**
- If you named your RIT program *lab01* as directed in §3.1, then when you download your project, RIT will download a zip archive named *lab01.zip* containing *main.cpp*. Rename *lab01.zip* to lab01-*asurite*.zip or lab01-*asurite1-asurite2*.zip.
- Submit the zip archive to Canvas by the deadline using the *Lab Project 1* submission page.
- If you worked with a partner, the first student who adds him- or herself to the group will be designated as the *group leader*. If you worked in a team, the group leader shall upload the solution zip archive to Canvas. If you worked along, then obviously, you shall submit it.
- Once you submit a file in Canvas for grading, you can download the zip archive and verify that it is not corrupted and that it is named correctly and contains the correct *main.cpp* file. Do this as soon as you submit!
- If your program does not compile due to syntax errors or if it does not run correctly because one or more of the software requirements are violated, then upload what you have completed for grading anyway because you will generally receive partial credit points for effort.
- The submission deadline is **11:59pm Sat 06 Feb**. We shall permit lab submissions, up to 48 hours after the deadline, or in other words, you must submit before the 48-hour late submission deadline of **11:59pm Mon 08 Feb**. The penalty for a submission after the Sat deadline but prior to 11:59pm Sun 07 Feb will result in a 10% penalty deduction (-0.5 pts). The penalty for a submission after 11:59pm Sun 07 Feb but prior to the 48 hour late submission deadline of 11:59pm Mon 08 Feb shall be a 20% deduction (that is, -1 pt).
- The Canvas submission link will become unavailable exactly at 11:59pm on Mon 08 Feb and you will not be able to submit your project for grading after that time. We do not accept emailed submissions or submissions that are more than 48 hours later. Do not ask.
- Consult the online syllabus for the academic integrity policies.

5 Grading Rubric

1. Lab Project (5 pts)
   a. Eyeball and compile the student's submitted code in *main.cpp* and compare it to the attached solution source code file. If the student's program appears to be completely correct, then assign **5 pts**.
   b. Otherwise, if the student's code appears to not have any syntax errors, but it has one or more obvious logical errors and the student made a very good attempt to complete the entire program, then assign **4 pts**.
   c. Otherwise, if the student's code has one or more obvious syntax errors, but the student made a very good attempt to complete the entire program, then assign **3 pts**.
   d. Otherwise, if the student simply submitted the *Hello World!* Program given to them on page 1, then assign **2 pts**.
   e. Otherwise, if the student's zip archive contains a C++ source code file named *main.cpp* then assign **1 pt**.
   f. Otherwise, assign **0 pts** for a submitted zip archive which does not contain *main.cpp*.
2. The Lab Project 1 Deadline was **11:59pm on Sat 06 Feb**.
   a. Canvas will automatically deduct 10% (-0.5 pts) for a submission between **11:59pm Sat 06 Feb** and **11:59pm Sun 07 Feb**.
   b. Canvas will automatically deduct 20% (1 pt) for a submission between **11:59pm Sun 07 Feb** and **11:59pm Mon 08 Feb**.
   c. After Mon 08 Feb, submissions are not accepted for any reason.