CSCI 1300 CS1: Starting Computing
Ashraf, Fleming, Correll, Cox, Fall 2019
Homework 2
Due: Saturday, September 14th, by 6 pm
(5 % bonus on the total score if submitted by 11:59 pm Sep. 13th)

2 components (Moodle CodeRunner attempts, and zip file) must be completed and submitted by Saturday, September 14th, 6:00 pm for your homework to receive points.

---

1. Objectives

- Understanding C++ data types and functions (parameter passing and return values)
- Writing and testing C++ functions
    - Understand problem description
    - Design your function:
        - come up with a step by step algorithm,
        - convert the algorithm to pseudocode
        - imagine many possible scenarios and corresponding sample runs or outputs
    - Convert the pseudocode into a program written in the C++ programming language
    - Test it in the Cloud9 IDE and submit it for grading on Moodle

---

2. Background

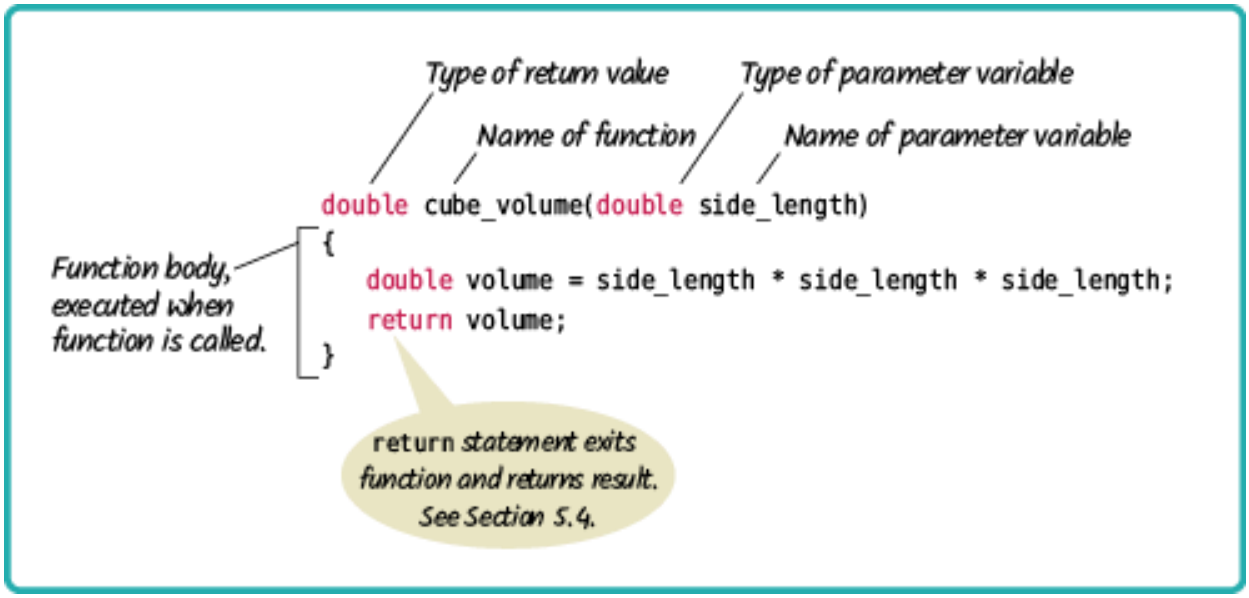
Functions in C++

A function is a block of code which is used to perform a particular task. Depending on whether a function is predefined or created by programmer; there are two types of function:

1. Library Function
2. User-defined Function

Library functions are the built-in functions in C++ programming. For example, C++ library provides `sqrt()` function to calculate the square root of a number.

C++ allows programmers to define their own functions. These are called user-defined functions. Every valid C++ program has at least one function, that is, `main()` function. Other user-defined functions are called from the `main()` function, or from within *other* user-defined functions.



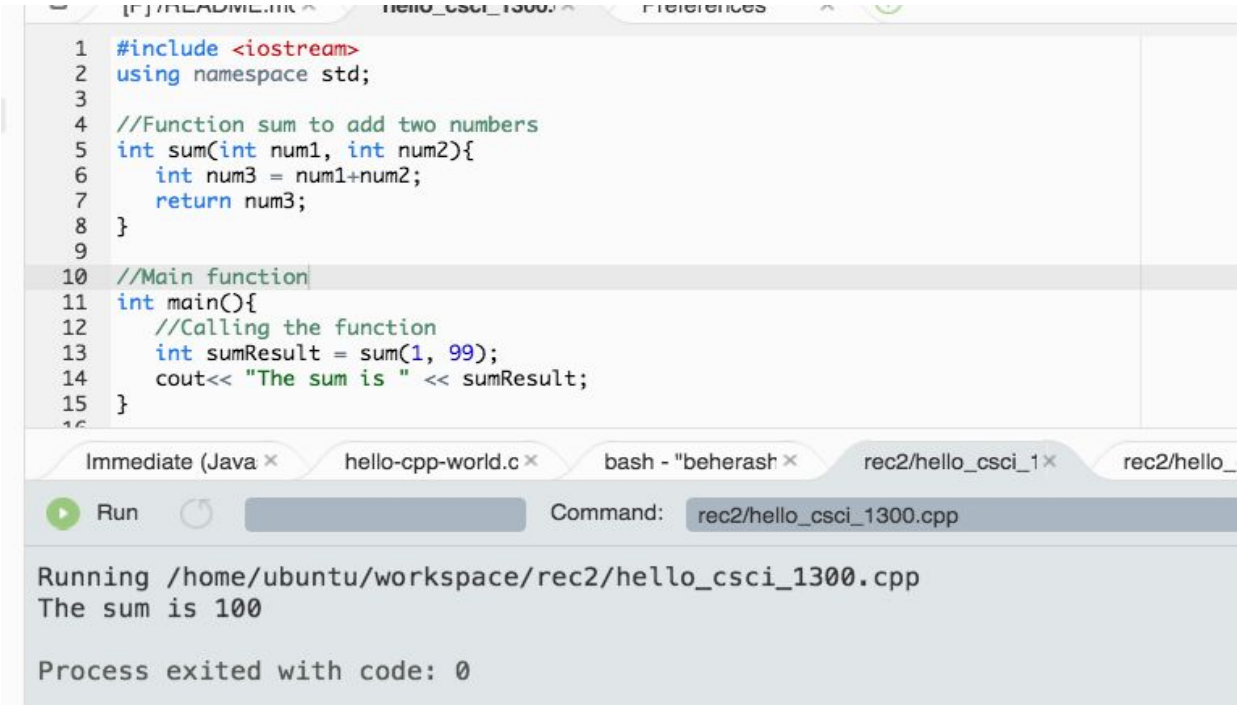
Here is the syntax for the function definition:

```
return_type function_name(parameter_list)
{
      //function_body
}
```

- `return_type` is the data type of the value that the function returns.
- `function_name` is the actual name of the function.
- `parameter_list` refers to the type, order, and number of the parameters of a function. A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument.
- `function_body` contains a collection of statements that define what the function does. The statements inside the function body are executed when a function is called.

When we don't have user-defined functions, all statements are executed sequentially, one after another. When we have used defined functions, the **flow of execution** changes. Let's follow in the example below.

When a program begins running, the system calls the `main()` function. When control of the program reaches the `sum()` function inside the `main()` (in the example below, at line **13**), it moves to function `sum()` and all code inside the function is executed (lines **6** and **7)**. After the `sum()` function finishes, the execution returns in the `main()` function and the following statement is executed (line **14**).



Data Types in C++

When programming, we store the variables in our computer's memory, but the computer needs to know what kind of data we want to store in them, since it is not going to occupy the same amount of memory to store a simple number than to store a single letter or a large number, and they are not going to be interpreted the same way. Some commonly used data types in C++ are:

1. `int` (for integers)
    - `int myInt = 5;`
2. `char` (for characters)
    - `char myChar = 'c';`
3. `float` (for floating-point numbers)

    - `float myFloat = 4.4531;`
4. `double` (for double precision floating-point numbers)
    - `double myDouble = 4.4531;`

---

3. Submission Requirements

All three steps must be fully completed by the submission deadline for your homework to be graded.

1. ***Work on questions on your Cloud 9 workspace:*** You need to write your code on Cloud 9 workspace to solve questions and test your code on your Cloud 9 workspace before submitting it to Moodle. (Create a directory called **hmwk2** and place all your file(s) for this assignment in this directory to keep your workspace organized)

2. ***Submit to the Moodle CodeRunner:*** Head over to Moodle to the link **Homework 2 CodeRunner**. You will find one programming quiz question for each problem in the assignment. Submit your solution for the first problem and press the Check button. You will see a report on how your solution passed the tests, and the resulting score for the first problem. You can modify your code and re-submit (press *Check* again) as many times as you need to, up until the assignment due date. Continue with the rest of the problems.

3. ***Submit a .zip file to Moodle:*** After you have completed all 10 questions from the Moodle assignment, zip all 10 solution files you compiled in Cloud9 (one cpp file for each problem), and submit the zip file through the **Homework 2** link on Moodle.

---

4. Rubric

Aside from the points received from the **Homework 2 CodeRunner** quiz problems, your TA will look at your solution files (zipped together) as submitted through the **Homework 2** link on Moodle and assign points for the following:

Style, Comments, Algorithm (10 points):

Style:

- Your code should be well-styled, and we expect your code to follow some basic guidelines on whitespace, naming variables and indentation, to receive full credit. Please refer to the **CSCI 1300 Style Guide** on Moodle.

Comments:

- Your code should be well-commented. Use comments to explain what you are doing, especially if you have a complex section of code. These comments are intended to help other developers understand how your code works. These comments should begin with two backslashes (`//`) or the multi-line comments (`/* … comments here… */`).
- Please also include a comment at the top of your solution with the following format:

```
// CS1300 Fall 2019
// Author: my name
// Recitation: 123 – Favorite TA
// Homework 2 - Problem # ...
```

Algorithm:

- Before each function that you define, you should include a comment that describes the inputs and outputs of your function and what algorithms you are using inside the function.
- This is an example C++ solution. Look at the code and the algorithm description for an example of what is expected.

Example 1:

```
/*
 * Algorithm: convert money from U.S. Dollars (USD) to Euros.
 *    1. Take the value of number of dollars involved
 *       in the transaction.
 *    2. Current value of 1 USD is equal to 0.86 euros
 *    3. Multiply the number of dollars got with the
```

```
 *      currency exchange rate to get Euros value
 *    4. Return the computed Euro value
 * Input parameters: Amount in USD (double)
 * Output (prints to screen): nothing
 * Returns: Amount in Euros (double)
 */
```

Example 2:

```
double convertUSDtoEuros(double dollars)
{
     double exchange_rate = 0.86; //declaration of exchange
rate
     double euros = dollars * exchange_rate; //conversion
     return euros; //return the value in euros
}
```

The algorithm described below does not mention in detail what the algorithm does and does not mention what value the function returns. Also, the solution is not commented. This would work properly, but would not receive full credit due to the lack of documentation.

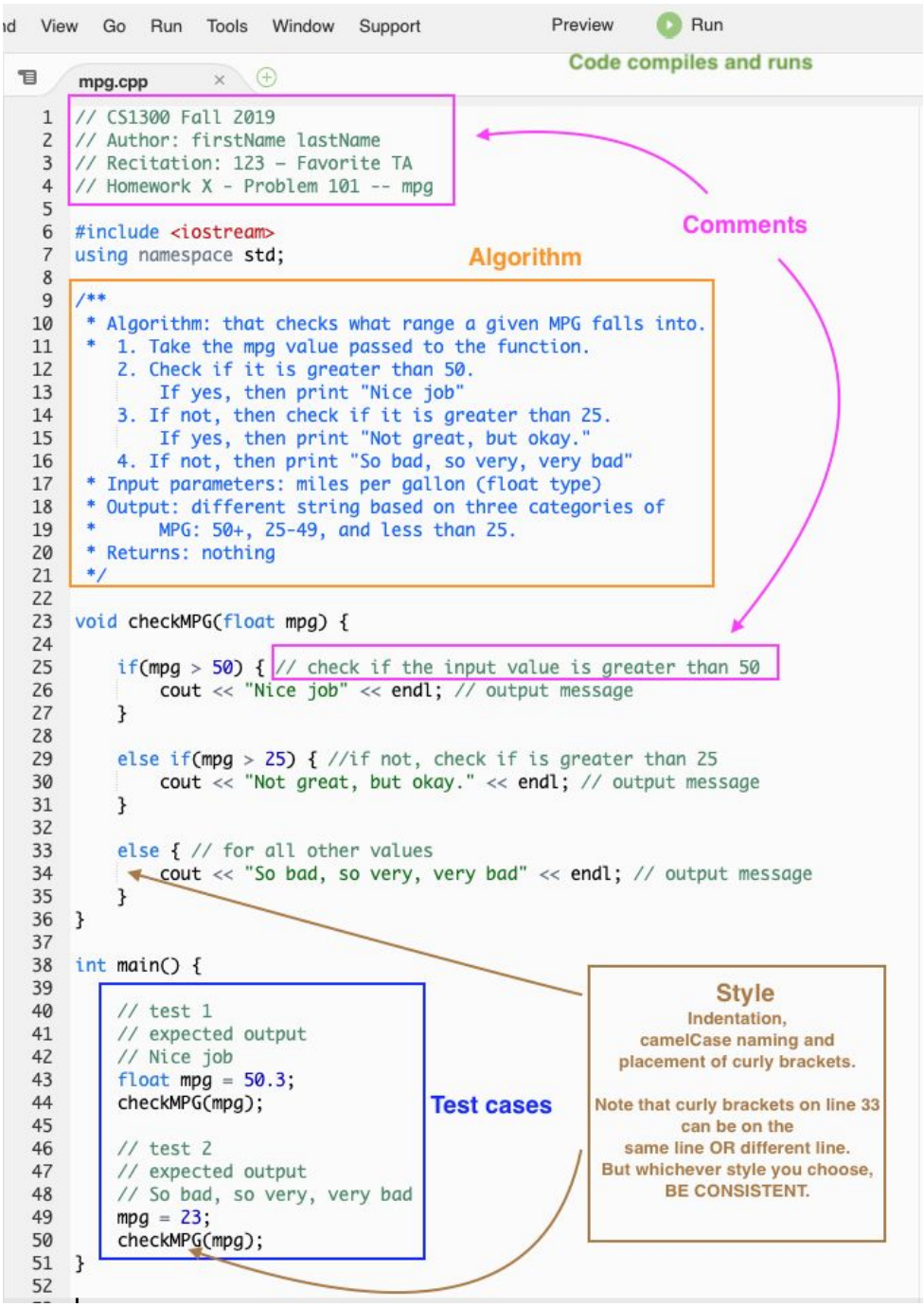
```
/*
 * conversion
 */
double convertUSDtoEuros(double dollars)
{
     double euros = dollars * 0.86;
     return euros;
}
```

Test Cases (20 points)

1. *Code compiles and runs* (**6 points**):
    - The zip file you submit to Moodle should contain **10** full programs (with a `main()` function), saved as .cpp files. It is important that your programs can be compiled and run on Cloud9 with no errors. The functions included in these programs should match those submitted to the CodeRunner on Moodle.

2. *Test cases* (**14 points**):
   For this week's homework, 7 out of the 10 problems are asking you to create a function (Problems 3 and 5-10). In your Cloud9 solution file for each function, you should have 2 test

cases present in their respective `main()` function, for a total of 14 test cases (see the diagram on the next page). Your test cases should follow the guidelines, **Writing Test Cases**, posted on Moodle under Week 3.




```
1  // CS1300 Fall 2019
2  // Author: firstName lastName
3  // Recitation: 123 - Favorite TA
4  // Homework X - Problem 101 -- mpg
5
6  #include <iostream>
7  using namespace std;
8
9  /**
10  * Algorithm: that checks what range a given MPG falls into.
11  *  1. Take the mpg value passed to the function.
12     2. Check if it is greater than 50.
13        If yes, then print "Nice job"
14     3. If not, then check if it is greater than 25.
15        If yes, then print "Not great, but okay."
16     4. If not, then print "So bad, so very, very bad"
17  * Input parameters: miles per gallon (float type)
18  * Output: different string based on three categories of
19  *      MPG: 50+, 25-49, and less than 25.
20  * Returns: nothing
21  */
22
23 void checkMPG(float mpg) {
24
25     if(mpg > 50) { // check if the input value is greater than 50
26         cout << "Nice job" << endl; // output message
27     }
28
29     else if(mpg > 25) { //if not, check if is greater than 25
30         cout << "Not great, but okay." << endl; // output message
31     }
32
33     else { // for all other values
34         cout << "So bad, so very, very bad" << endl; // output message
35     }
36 }
37
38 int main() {
39
40     // test 1
41     // expected output
42     // Nice job
43     float mpg = 50.3;
44     checkMPG(mpg);
45
46     // test 2
47     // expected output
48     // So bad, so very, very bad
49     mpg = 23;
50     checkMPG(mpg);
51 }
52
```

5. Problem Set

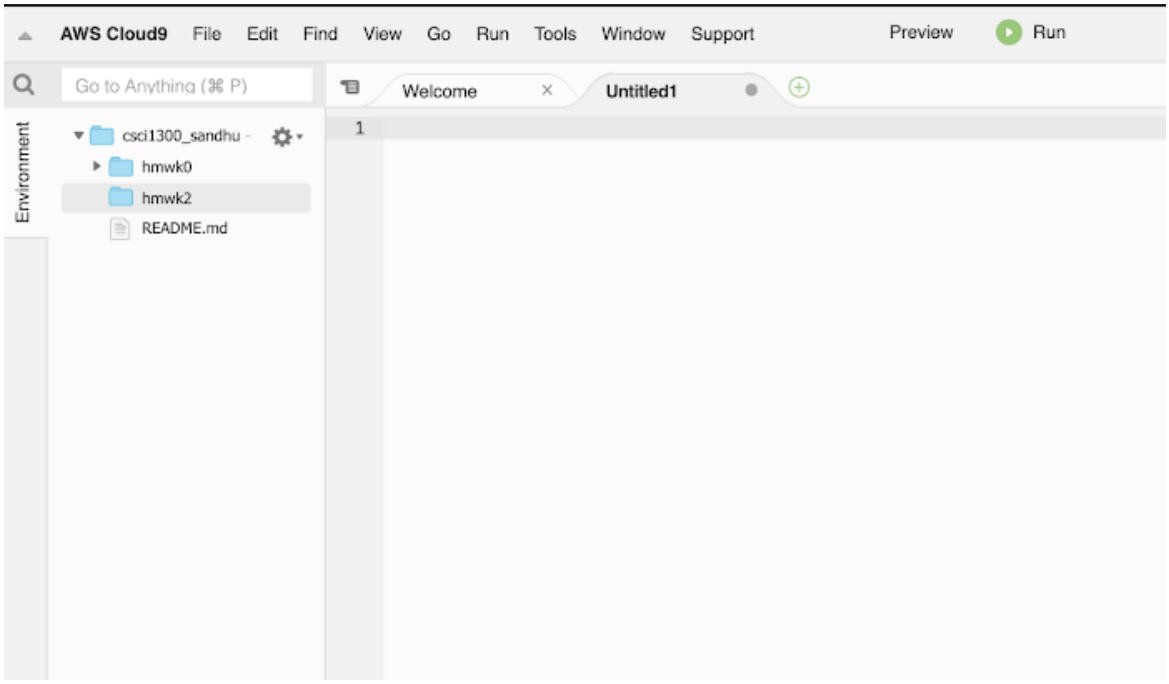
Note: To stay on track for the week, we recommend to finish/make considerable progress on problems 1-6 by Wednesday. Students with recitation on Thursday are encouraged to come to recitation with questions and have made a start on all of the problems.

Problem 1 (5 points): Hello World program

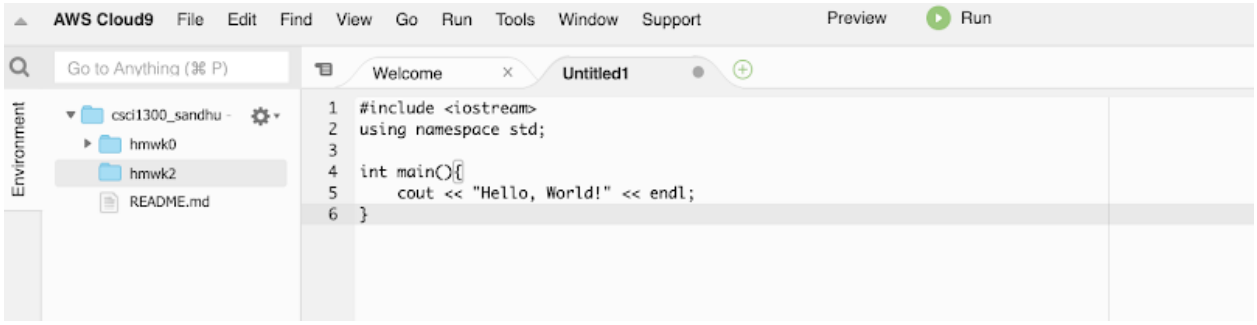
The first program that we usually write in any language we're learning is *Hello, World*. Your task is to write a *Hello, World* program just prints "`Hello, World!`" to the screen (the console window in Cloud9).

Here are some suggested steps:

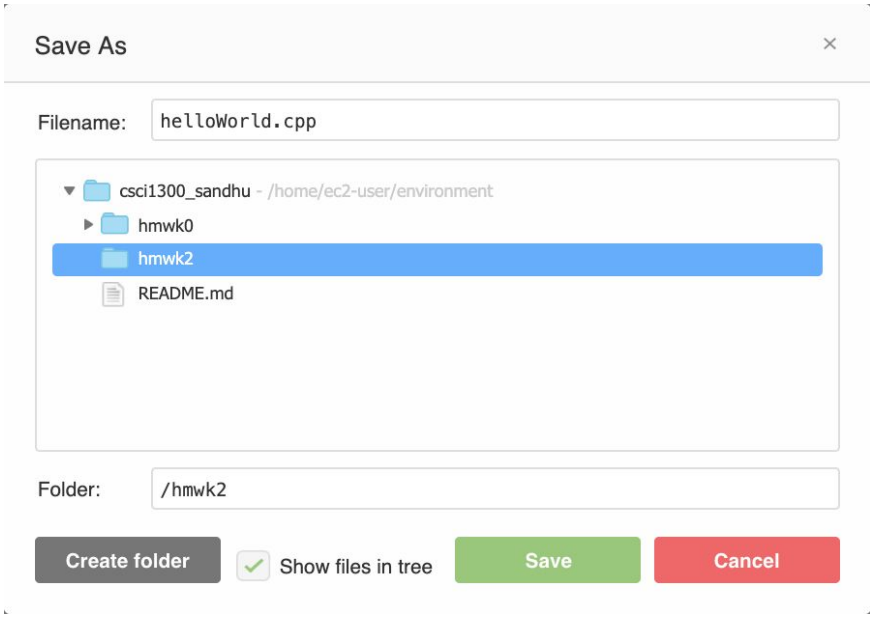
Step 1: Open an Empty File
In Cloud9, select File -> New File. A new, blank file called Untitled1 will be opened.



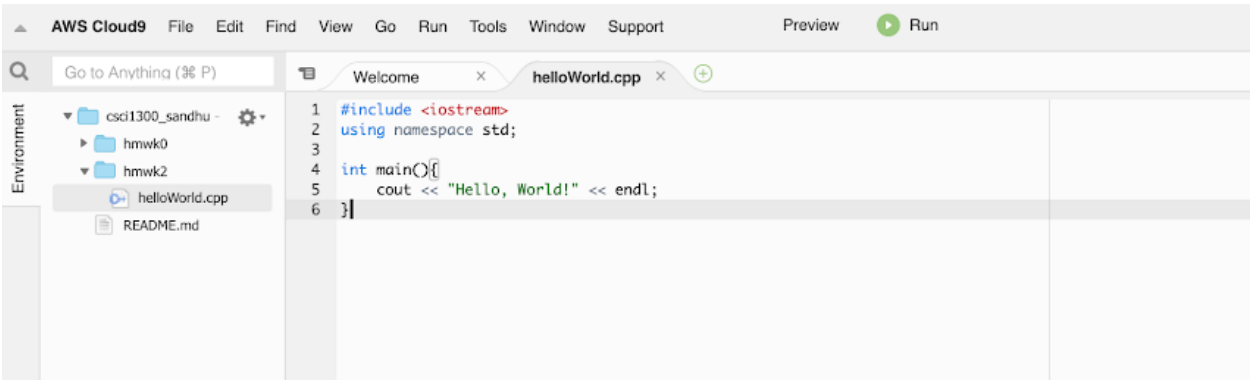
Step 2: Your First Code
Starting on line 1 in Untitled1, type the following code.



Step 3: Saving Your File
Save the file: go to File -> Save As... A dialog box will open. Name it **helloWorld.cpp** and save it in the **hmwk2** folder.
Note: *make sure you save it with the .cpp extension or it will not compile correctly!*

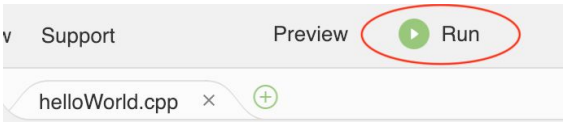


The `.cpp` extension on the filename tells Cloud9 that the file should be read in the C++ programming language. Once you save it, the lines in the file should be color-coded to reflect what they do in the program. This is called syntax highlighting.



Important: You should save your work frequently in Cloud9 to avoid losing your work in the event of the program crashing.

Step 4: Running Your Code



To run the program, click on the icon with the green arrow next to the word Run. If it works, you should see new terminal tab window open at the bottom. The title of the tab shows the file being run (hmwk2/ helloWorld.cpp), and inside the window you should see "Running ...." (again the name and full path of the file), and underneath it, the output of our program: "`Hello, World!`"

Step 5: Running Your Code from Command Line

Move to the "bash" tab (the first tab in the bottom panel). Right-click again and Clear the Buffer. Make sure you are inside the **hmwk2** directory. Type:

```
$ g++ helloWorld.cpp -g -std=c++11
```

the `-g` option turns on debugging, which we will use later in the semester, so we should get used to it.

the `-std=c++11` option makes sure that the c++ version used to run the program is c++ 11. If you don't give this option then default version(which is usually C++98) is used.

This creates an executable called "a.out" (see figure below). You can run it by typing

```
$ ./a.out
```

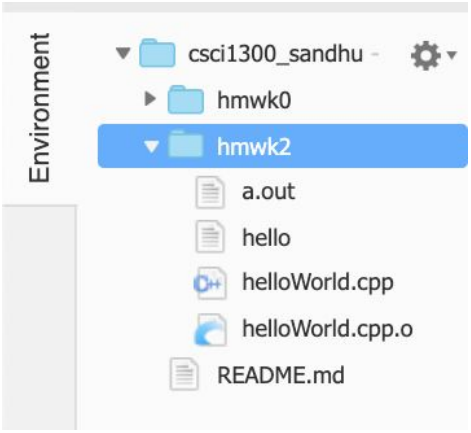
Since no executable name was specified to g++, a.out is chosen by default. You can alternatively use the "-o" option to change the name :

```
$ g++ helloWorld.cpp -g -std=c++11 -o hello
```

creates an executable called "`hello`" (see figure below). You can run it by typing

```
$ ./hello
```

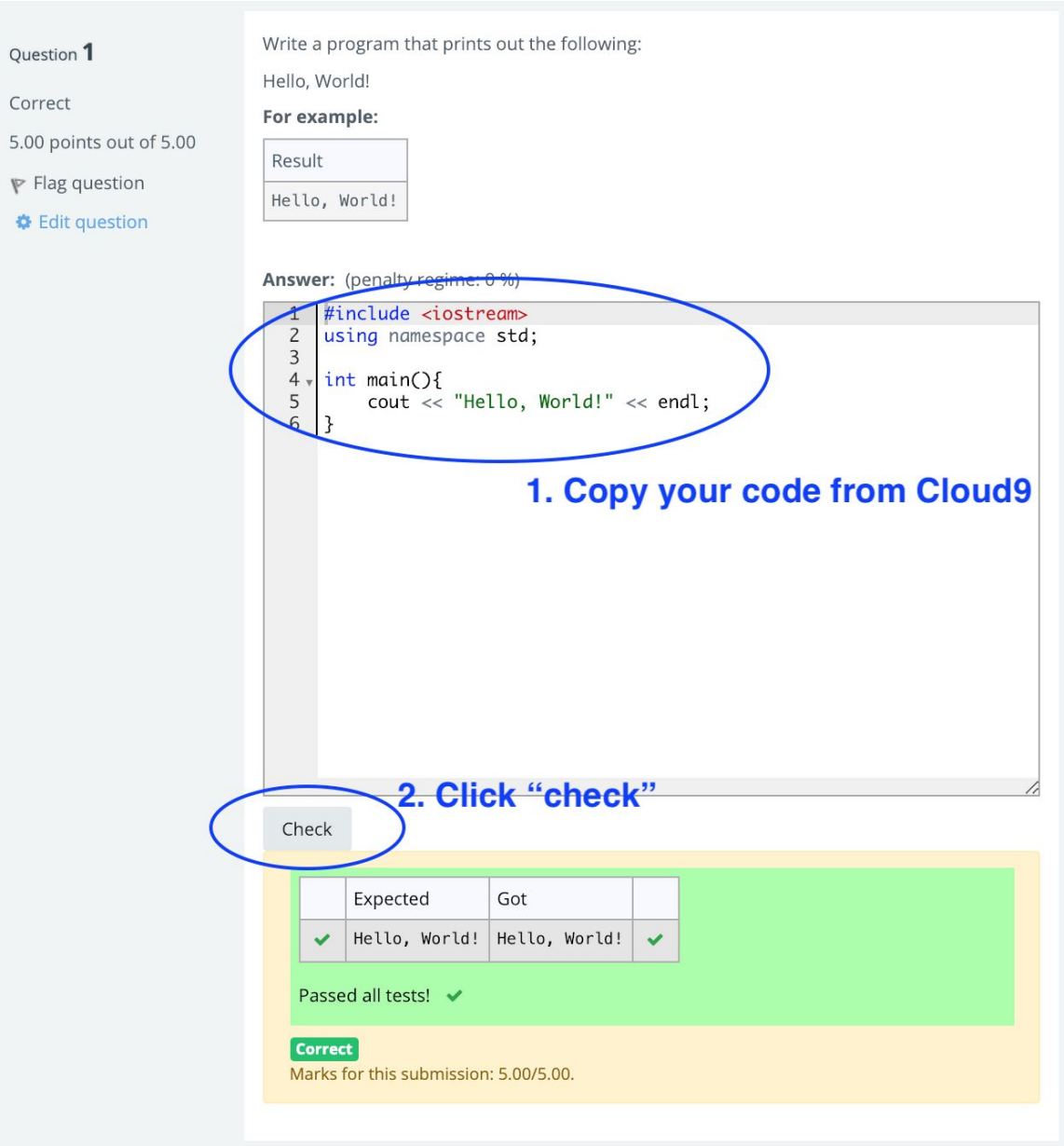
Notice the output in the same: `Hello, world!`, followed by the return of the prompt, for new commands.




```
bash - "ip-172-31
vocstartsoft:~/environment $ cd hmwk2/
vocstartsoft:~/environment/hmwk2 $ g++ helloWorld.cpp -g -std=c++11
vocstartsoft:~/environment/hmwk2 $ ./a.out
hello world!
vocstartsoft:~/environment/hmwk2 $ g++ helloWorld.cpp -g -std=c++11 -o hello
vocstartsoft:~/environment/hmwk2 $ ./a.out
hello world!
vocstartsoft:~/environment/hmwk2 $
```

Step 6: Submit to Moodle CodeRunner

Head over to Moodle to the link **Homework 2 CodeRunner**. Submit your solution for the first problem and press the Check button. You will see a report on how your solution passed the tests, and the resulting score for the first problem. You can modify your code and re-submit (press Check again) as many times as you need to.



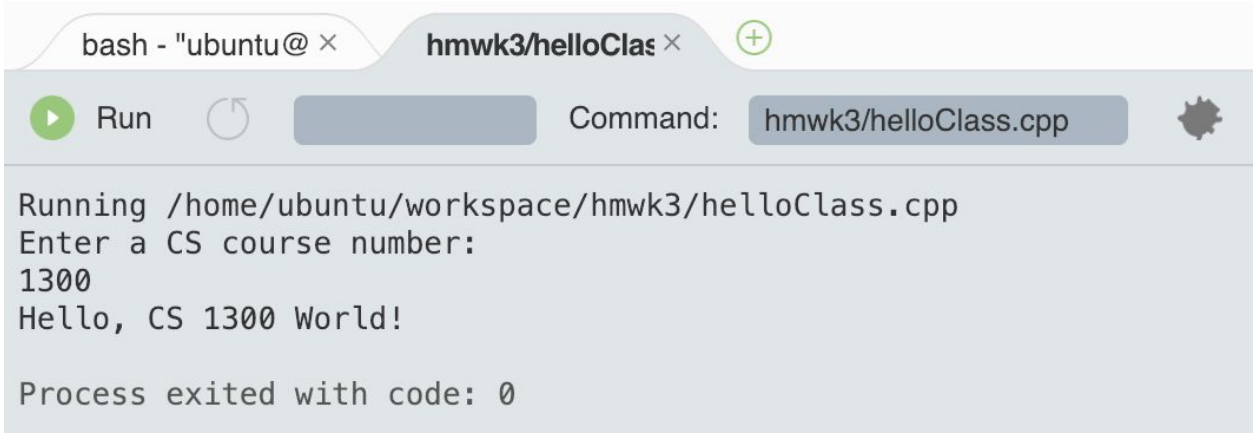
Step 7: The zip file submission

Remember that the file **helloWorld.cpp** will be one of 10 files you need to zip together for the **Homework 2** on Moodle.

Problem 2 (5 points): Hello Class program

Write a program that takes as input from the user the integer value of a CS course number. Then, your program should print "`Hello, CS [course number] World!`" to the screen. You will first need to prompt the user to enter a CS course number with a statement requesting them to "`Enter a CS course number: `". Once the user enters the course number, print the required output.

For example: If the user enters the input `1300`, the output of the program should be,

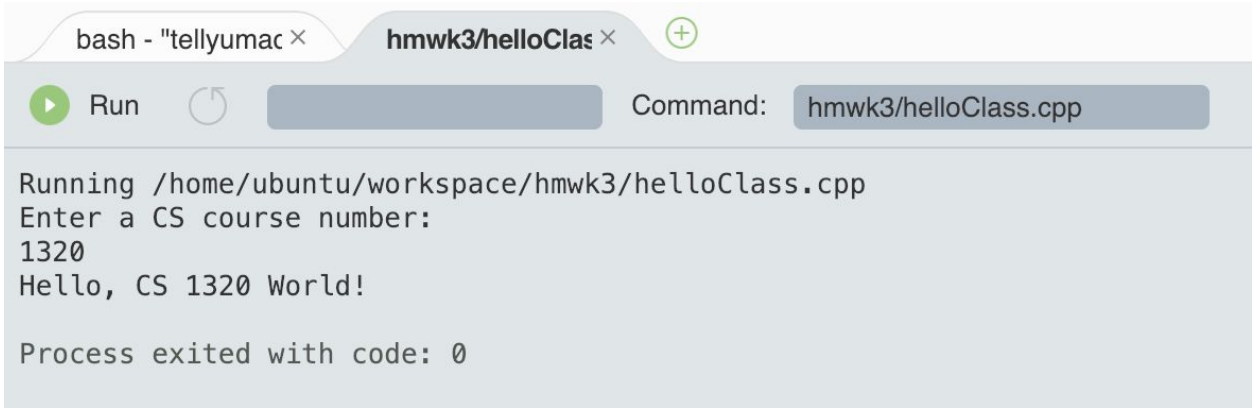


Here are some suggested steps:

Step 1: Create a new file
Just like we did in Problem 1, create a file called **helloClass.cpp**.

Step 2: Write the program
Create a `main()` function, just like for Problem 1. You will need to modify the Problem 1 solution as follows:

- Create an integer variable to store the value of course number
- Prompt the user to enter a course number using the output: "`Enter a CS course number: `"
- Generate the final output as a combination of text (strings) and the value of the variable holding the course number entered by the user. The output should match exactly the example below.



Step 3: Submit to the Moodle CodeRunner
Head over to Moodle to the link **Homework 2 CodeRunner**. Submit your solution for Problem 2 and press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

Important: the `cout` formats provided for each problem are not suggestions – they MUST be followed precisely, word-for-word and including all punctuation marks, otherwise the CodeRunner will not recognize your results and you will not receive credit.

Step 4: The zip file submission
Remember that the file **helloClass.cpp** will be one of 10 files you need to zip together for the **Homework 2** on Moodle.

Problem 3 (5 points): classGreeting function

Write a function called `classGreeting` that takes a single integer parameter and prints a greeting to the screen. If `1300` is the given argument value, the function should print:

```
Hello, CS 1300 World!
```

Specifications:

- Your function should have **one** input argument as an integer:
  - An integer parameter representing the course number
- Your function should NOT return anything.
- Your function should **print** the course number in the following format
  - `Hello, CS **** World!`, where **** should be replaced by the value of the input argument
- Your function *MUST* be named **classGreeting**.

Here are some suggested steps:

Step 1: Create a new file
Just like we did in Problems 1 and 2, create a file called **classGreeting.cpp**.

Step 2: Write the program
Write the pre-processor directives (`#include <...>` and using `namespace std;`), followed by the `main()` function. Above the `main()` function, you need to write the code for the `classGreeting` function. Use the information provided above in the *Specifications,* and follow the syntax for a function definition provided in the Background section.

- Start with the return type
- Then the function name
- Then, in parenthesis, the type and name of the input parameter

Write the solution for the function body, in between the {}.

Step 3: Calling the function
After writing a function, it's very important to check if your function is working as we expect. In the `main()` function, call the `classGreeting` function to test if it accomplishes the required output. Remember you will need to pass a value as an argument to the function.

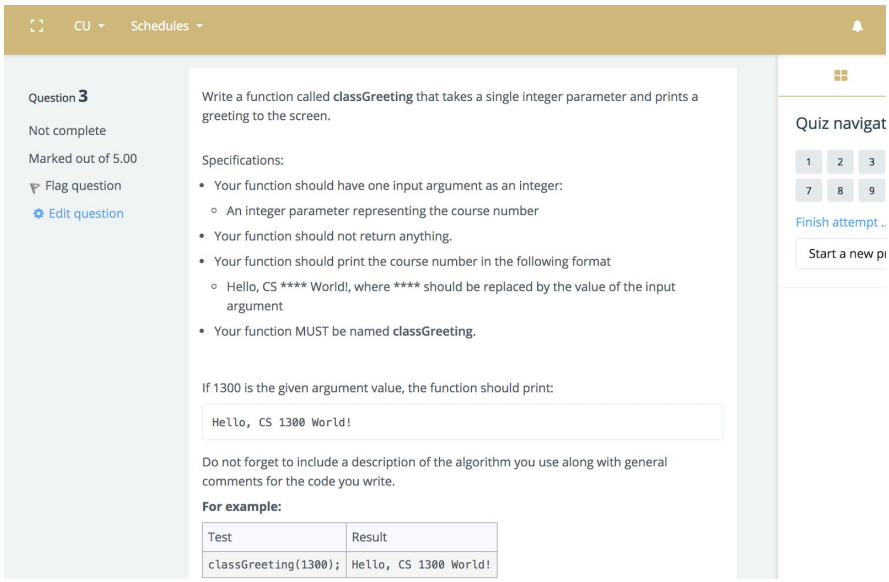
Note: the submission requirements are to have at least 2 tests for each function. Follow the **Writing Test Cases** examples posted on Moodle

Your test cases should look like in the example below, where the name of the function is `sayHello`, and the `main()` function has two function calls.

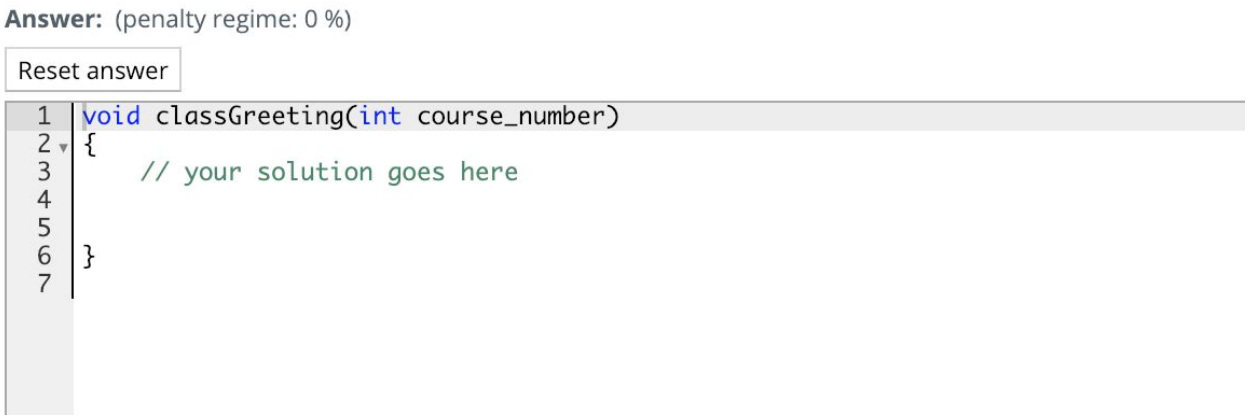
```
void sayHello(string name)
{
    cout << "Hello " << name << endl;
}

int main()
{
    // test 1
    // expected output
    // Hello Bob
    sayHello("Bob"); // first function call to sayHello()

    // test 2
    // expected output
    // Hello Mary
    sayHello("Mary"); // second function call to sayHello()
}
```

Step 4: Submit to the Moodle CodeRunner

Head over to Moodle to the link [Homework 2 CodeRunner](). Go to Problem 3.



In the Answer Box, paste **only your function definition, not the entire program.**



Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

You must name the functions as indicated in each problem description. **Importantly**, *the* `cout` *formats provided for each problem are not suggestions – they* ***MUST*** *be followed precisely, word-for-word and including all punctuation marks*, otherwise the CodeRunner will not recognize your results and you will not receive credit.

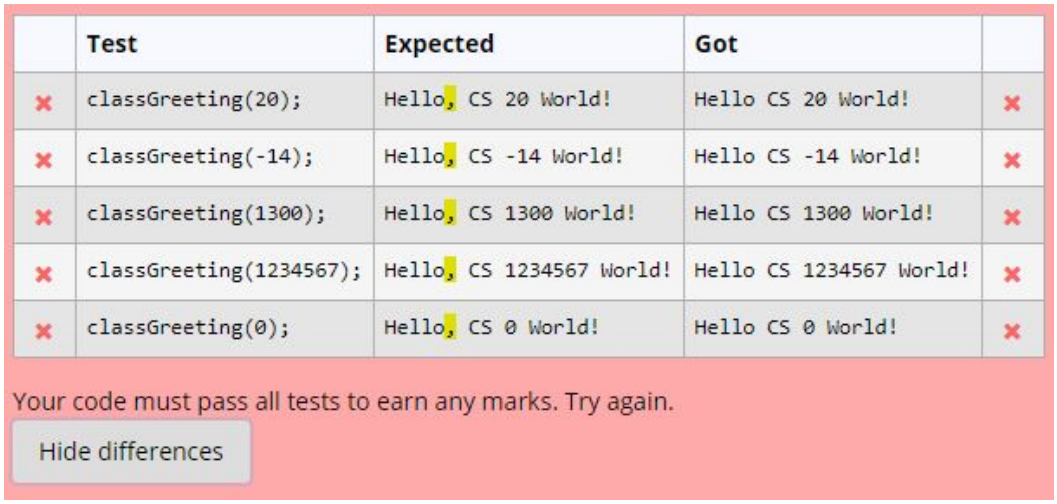
If there are errors in your solution to a particular problem, a button labeled “Show differences” will appear below the table of tests after you hit “check”. This can be a very useful tool in helping you find small typos, especially in cout statements.

	Test	Expected	Got	
✖	classGreeting(20);	Hello, CS 20 World!	Hello CS 20 World!	✖
✖	classGreeting(-14);	Hello, CS -14 World!	Hello CS -14 World!	✖
✖	classGreeting(1300);	Hello, CS 1300 World!	Hello CS 1300 World!	✖
✖	classGreeting(1234567);	Hello, CS 1234567 World!	Hello CS 1234567 World!	✖
✖	classGreeting(0);	Hello, CS 0 World!	Hello CS 0 World!	✖

Your code must pass all tests to earn any marks. Try again.

Show differences

For example, below we hit “check” for a solution to problem 3 of this homework and have failed all the test cases despite getting the correct values. Hitting “Show differences”, we can see that a comma (`,`) is missing. When characters are in the expected output but not in your output they are highlighted in the “Expected” column.



	Test	Expected	Got	
✖	classGreeting(20);	Hello, CS 20 World!	Hello CS 20 World!	✖
✖	classGreeting(-14);	Hello, CS -14 World!	Hello CS -14 World!	✖
✖	classGreeting(1300);	Hello, CS 1300 World!	Hello CS 1300 World!	✖
✖	classGreeting(1234567);	Hello, CS 1234567 World!	Hello CS 1234567 World!	✖
✖	classGreeting(0);	Hello, CS 0 World!	Hello CS 0 World!	✖

Your code must pass all tests to earn any marks. Try again.

Hide differences

On the other hand, when we include extra, unexpected characters in output they are highlighted in the “Got” column. Below we added additional exclamation points (!) to the output.

	Test	Expected	Got	
✖	classGreeting(20);	Hello, CS 20 World!	Hello, CS 20 World!!!!	✖
✖	classGreeting(-14);	Hello, CS -14 World!	Hello, CS -14 World!!!!	✖
✖	classGreeting(1300);	Hello, CS 1300 World!	Hello, CS 1300 World!!!!	✖
✖	classGreeting(1234567);	Hello, CS 1234567 World!	Hello, CS 1234567 World!!!!	✖
✖	classGreeting(0);	Hello, CS 0 World!	Hello, CS 0 World!!!!	✖

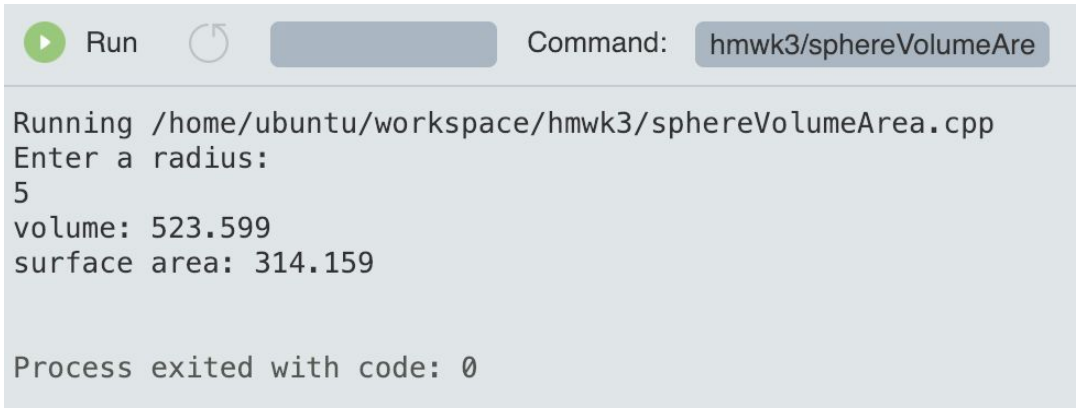
Your code must pass all tests to earn any marks. Try again.

Hide differences

Step 5: The zip file submission
Remember that the file **classGreeting.cpp** will be one of 10 files you need to zip together for the **Homework 2** on Moodle.

Problem 4 (5 points): Calculating sphere volume and area

Alter the provided `main()` in **sphereVolumeArea.cpp** file (on Moodle) to print both the volume and surface area of a sphere with given radius. For a radius of 5, the output of the program should look like this:



Here are some suggested steps:

Step 1: Download sphereVolume.cpp from Moodle
You can find sphereVolumeArea.cpp, and upload it to your workspace on cloud9.
Step 2: Add new statements for the surface area calculation
Add statements to compute the surface area of a sphere, which is **$4\pi r^2$**, where **r** is the radius.

Step 3: Submit to the Moodle CodeRunner
Head over to Moodle to the link **Homework 2 CodeRunner**. Submit your solution for Problem 4 and press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

Step 4: The zip file submission
Remember that the file **sphereVolumeArea.cpp** (after your modifications) will be one of 10 files you need to zip together for the **Homework 2** on Moodle.

Problem 5 (5 points): sphereVolume function

Write a function called **sphereVolume** that determines the volume of a sphere with a given radius and prints the result to the screen.

- Your function **MUST** be named **sphereVolume**
- Your function should have **one** input argument:
    - a floating point parameter representing the radius - as a double
- Your function should not return anything.
- Your function should **print** the calculated volume.
    - The output format should resemble that of the previous problem. For a radius of 5, the function should print: `volume: 523.599`

Follow the steps suggested for Problem 3. In Cloud9 the file should be called **sphereVolume.cpp** and it will be one of 10 files you need to zip together for the **Homework 2** on Moodle.

Don't forget to head over to Moodle to the link **Homework 2 CodeRunner**. For Problem 5, in the Answer Box, paste **only your function definition, not the entire program**, just like you did for Problem 3. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

Problem 6 (5 points): sphereSurfaceArea function

Write a function called **sphereSurfaceArea** that determines the surface area of a sphere with given radius and prints the result to the screen. *Hint: Recycle some of your work for Problem 5 to save some time here!*

- Your function **MUST** be named **sphereSurfaceArea**
- Your function should have **one** input argument:
    - a floating point parameter representing the radius - as a double
- Your function should not return anything.
- Your function should **print** the calculated surface area.
    - The output format should resemble that of the previous problem. For a radius of 5, the function should print: `surface area: 314.159`

In Cloud9 the file should be called **sphereSurfaceArea.cpp** and it will be one of 10 files you need to zip together for the **Homework 2** on Moodle.

Don't forget to head over to Moodle to the link **Homework 2 CodeRunner**. For Problem 6, in the Answer Box, paste **only your function definition, not the entire program**, just like you did for Problem 3. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

Problem 7 (10 points): convertSeconds function

Write a function **convertSeconds** that takes one integer input as seconds and converts it to hours, minutes and seconds. Your function can round off the seconds to the nearest whole value after conversion. Then, your function will print the time in hours, minutes and seconds to the screen, as demonstrated below. You should convert the amount of time in such a way that maximizes the whole numbers of hours and minutes.

- Your function **MUST** be named **convertSeconds**
- Your function should accept only one input value, seconds, as an integer
- Your function should not return any value
- Your function should print the string in the following format:

    h hour(s) *m* minute(s) *s* second(s)

    For example, given input seconds as 3671, it should print :

    `1 hour(s) 1 minute(s) 11 second(s)`

In Cloud9 the file should be called **convertSeconds.cpp** and it will be one of 10 files you need to zip together for the **Homework 2** on Moodle.

Don't forget to head over to Moodle to the link **Homework 2 CodeRunner**. For Problem 7, in the Answer Box, paste **only your function definition, not the entire program**, just like you did for Problem 3. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

Problem 8 (10 points): population function

The U.S. Census provides information about the current U.S. population as well as approximate rates of change. Using those rates and the current US population, write a function to calculate the U.S. population in exactly one year (365 days). Your function should return the result of your calculations. If you end up with a non-integer projected population, then round down to the nearest whole person.

Three rates of change are provided:

- ➢ There is a birth every 8 seconds
- ➢ There is a death every 12 seconds
- ➢ There is a new immigrant arriving in the US every 27 seconds

Specifications:

- Your function **MUST** be named **population**
- Your function should accept only one input argument: the initial population, as an integer
- Your function should return the population value in a year - you should be able to figure out the type of the return value.
- Your function should not print/display/output anything to the screen

For example, given an initial population of 1,000,000, your function would return **3,482,000**.

In Cloud9 the file should be called **population.cpp** and it will be one of 10 files you need to zip together for the **Homework 2** on Moodle.

Don't forget to head over to Moodle to the link **Homework 2 CodeRunner**. For Problem 8, in the Answer Box, paste **only your function definition, not the entire program**, just like you did for Problem 3. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

Problem 9 (10 points): carnot function

In thermodynamics, the Carnot efficiency is the maximum possible efficiency of a heat engine operating between two reservoirs at different temperatures. The Carnot efficiency is given as

$$\eta = 1 - \frac{T_C}{T_H}$$

, where T_C and T_H are the absolute temperatures at the cold and hot reservoirs, respectively. Write a function carnot that will compute the Carnot efficiency.

- The function **must** be named **carnot**.
- The function takes in two parameters, in this order:
    - the cold temperature reservoir operates (T_C), integer value
    - the hot temperature reservoir operates (T_H), integer value
- The function should return the Carnot efficiency as **double**.

In Cloud9 the file should be called **carnot.cpp** and it will be one of 10 files you need to zip together for the **Homework 2** on Moodle.

Don't forget to head over to Moodle to the link **Homework 2 CodeRunner**. For Problem 9, in the Answer Box, paste **only your function definition, not the entire program**, just like you did for Problem 3. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

Problem 10 (10 points): calculateSalary function

A field worker is paid $10 an hour. The hours of work for a day vary based on the weather for that day. On a rainy day, 5 hours of work can be achieved, 4 hours of work on a cold day and 8 hours of work on a bright sunny day.

Write a function called calculateSalary that returns the total money earned based on the number of rainy, cold and sunny days worked.

- Your function name **MUST** be named **calculateSalary**
- Your function should accept 3 integer arguments in the order:
    - number of rainy days
    - number of cold days
    - number of sunny days respectively
- Your function should return the salary earned as int
- Your function should not print anything

For example:

> The worker worked 5 rainy days, 8 cold days and 19 sunny days. Then the total money earned will be calculated by (5*5 + 4*8 + 8*19) * 10 = $ 2090

In Cloud9 the file should be called **calculateSalary.cpp** and it will be one of 10 files you need to zip together for the **Homework 2** on Moodle.

Don't forget to head over to Moodle to the link **Homework 2 CodeRunner**. For Problem 10, in the Answer Box, paste **only your function definition, not the entire program**, just like you did for Problem 3. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

6. Homework 2 checklist

Here is a checklist for submitting the assignment:

1. Complete the code **Homework 2 CodeRunner**
2. Submit one zip file to **Homework 2**. The zip file should be named, **hmwk2_lastname.zip**. It should have following 10 files:
    - **helloWorld.cpp**
    - **helloClass.cpp**
    - **classGreeting.cpp**
    - **sphereVolumeArea.cpp**
    - **sphereVolume.cpp**
    - **sphereSurfaceArea.cpp**
    - **convertSeconds.cpp**
    - **population.cpp**
    - **carnot.cpp**
    - **calculateSalary.cpp**

7. Homework 2 points summary

Criteria	Pts
CodeRunner (problem 1 - 10)	70
Style, Comments, Algorithms	10
Test cases	20
Recitation attendance (week 3)*	-30
Total	100
5% early submission bonus	+5%

* if your attendance is not recorded, you will lose points. Make sure your attendance is recorded on Moodle.