

CSCI 1300 CS1:Starting Computing

Common Errors

Ashraf, Fleming, Correll, Cox, Fall 2019

There are a number of very common programming mistakes which you are likely to encounter at some point during this course. Recognizing the symptoms of these errors and being able to interpret the messages provided by the compiler will help you solve problems quickly and efficiently.

Here we have included screenshots of different errors that you might come across on recitation 2 both in Cloud 9 and on Moodle:

- If you're developing and testing your solution in Cloud9, the error messages will show up as soon as you try to *Run* your program.
- If you're on Moodle and you have pasted your solution into the Answer box, the errors will show up as soon as you press the button *Check*.

Part 1: Hello, World!

Anatomy of an Error in Cloud9:

Error messages in Cloud9, and those produced by most C++ compilers, provide three general pieces of information. First is the name of the file causing the error. This might be something like `"/home/ubuntu/workspace/rec2/recitation2.cpp"` for recitation 2.

Next, the approximate location of the error is given in the format (line number):(character number). So an error on the third line and eighth character would be represented as `"3:8"`. It is important to realize that this is where the compiler ran into trouble, not necessarily exactly where the error is. Frequently, the error itself will be above or before the line given.

Finally, the error itself is provided. Some error messages are very useful, others can be cryptic. If you encounter an unfamiliar error try Googling the main portions of the message excluding pieces which might be specific to your code like variable names. You can also post your error text message on Piazza and we'll let you know what it means.

Example 1: Missing Angle Brackets:

Cloud9-



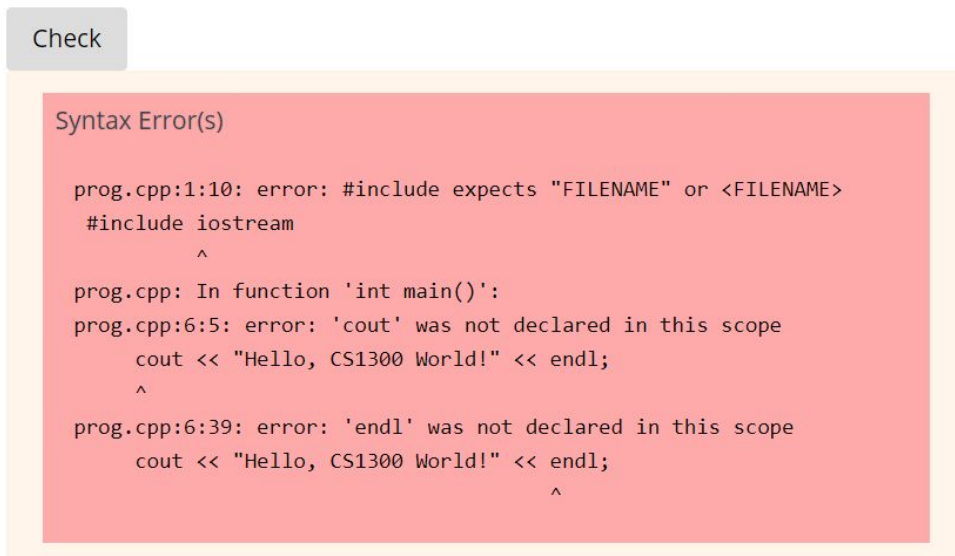
The screenshot shows the Cloud9 IDE interface. At the top, there's a tab for 'recitation2.cpp'. The code editor displays the following C++ code:

```
1 #include iostream
2 using namespace std;
3
4 int main()
5 {
6     cout << "Hello, world!" << endl;
7 }
```

Below the code editor, there's a terminal window showing the output of running the program. The terminal output includes the following error messages:

```
Running /home/ubuntu/workspace/rec2/recitation2.cpp
/home/ubuntu/workspace/rec2/recitation2.cpp:1:10: error: #include expects "FILENAME" or <FILENAME>
#include iostream
      ^
/home/ubuntu/workspace/rec2/recitation2.cpp: In function 'int main()':
/home/ubuntu/workspace/rec2/recitation2.cpp:6:5: error: 'cout' was not declared in this scope
cout << "Hello, world!" << endl;
  ^
/home/ubuntu/workspace/rec2/recitation2.cpp:6:32: error: 'endl' was not declared in this scope
cout << "Hello, world!" << endl;
                        ^
Process exited with code: 1
```

Moodle-



The screenshot shows the Moodle LMS interface. At the top, there's a 'Check' button. Below it, a red box contains the following text:

Syntax Error(s)

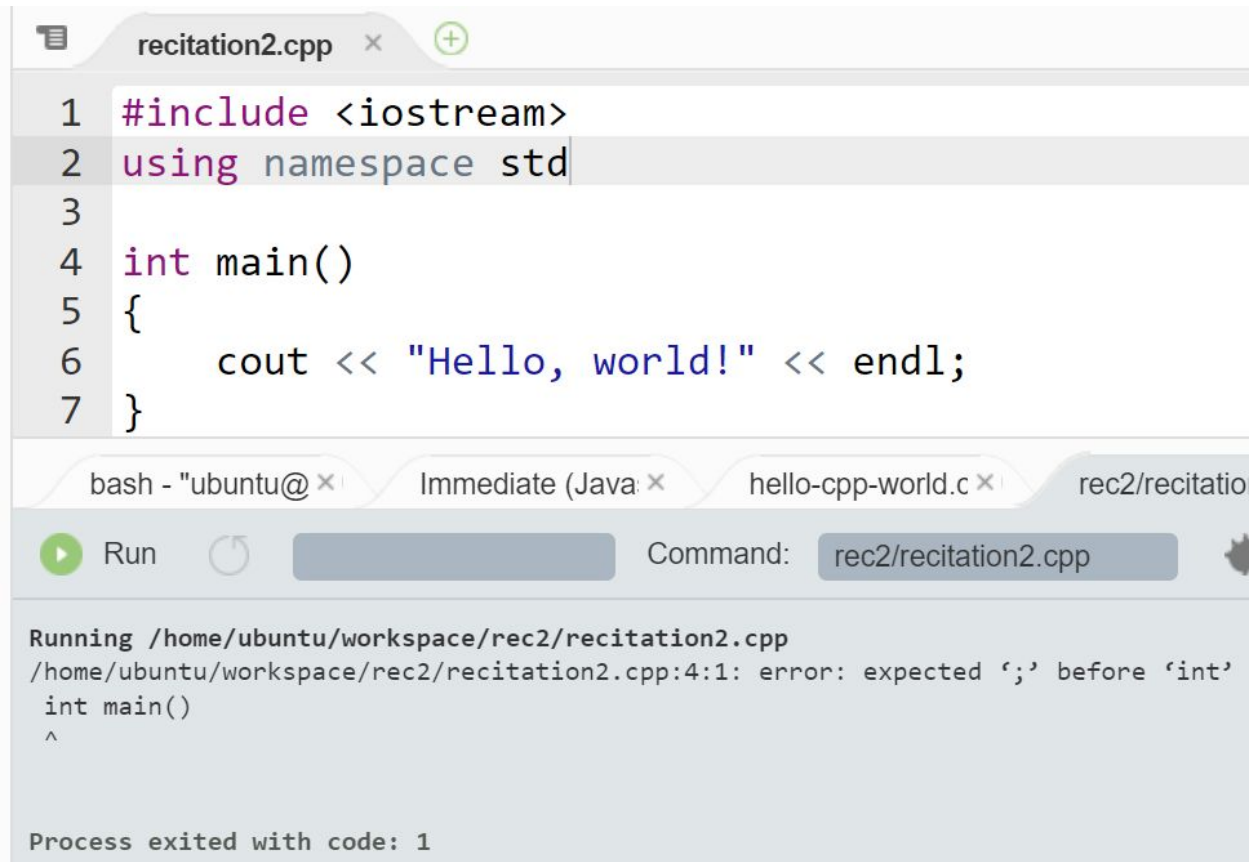
```
prog.cpp:1:10: error: #include expects "FILENAME" or <FILENAME>
#include iostream
      ^
prog.cpp: In function 'int main()':
prog.cpp:6:5: error: 'cout' was not declared in this scope
cout << "Hello, CS1300 World!" << endl;
  ^
prog.cpp:6:39: error: 'endl' was not declared in this scope
cout << "Hello, CS1300 World!" << endl;
                        ^
```

Oftentimes a single error in a program will propagate and cause multiple error messages. Here we forgot to include angle brackets (" $<$ " and " $>$ ") around 'iostream' on the first line. The first error message makes it clear that the appropriate syntax is either " $\#include <iostream>$ " or " $\#include "iostream"$ ".

Example 2: Missing Semicolons:

Semicolons are ubiquitous in C++ programs. They appear after *nearly every line* which is not a control structure or function definition. Notice, however, that there is no semicolon after the include statement. Forgotten semicolons can be difficult to track down. The error usually indicates a line number after the line missing the semicolon.

Cloud9 -



```
recitation2.cpp x +
1 #include <iostream>
2 using namespace std
3
4 int main()
5 {
6     cout << "Hello, world!" << endl;
7 }
```

bash - "ubuntu@x" Immediate (Java: x) hello-cpp-world.c x rec2/recitatio

Run Command: rec2/recitation2.cpp

Running /home/ubuntu/workspace/rec2/recitation2.cpp
/home/ubuntu/workspace/rec2/recitation2.cpp:4:1: error: expected ';' before 'int'
int main()
^

Process exited with code: 1

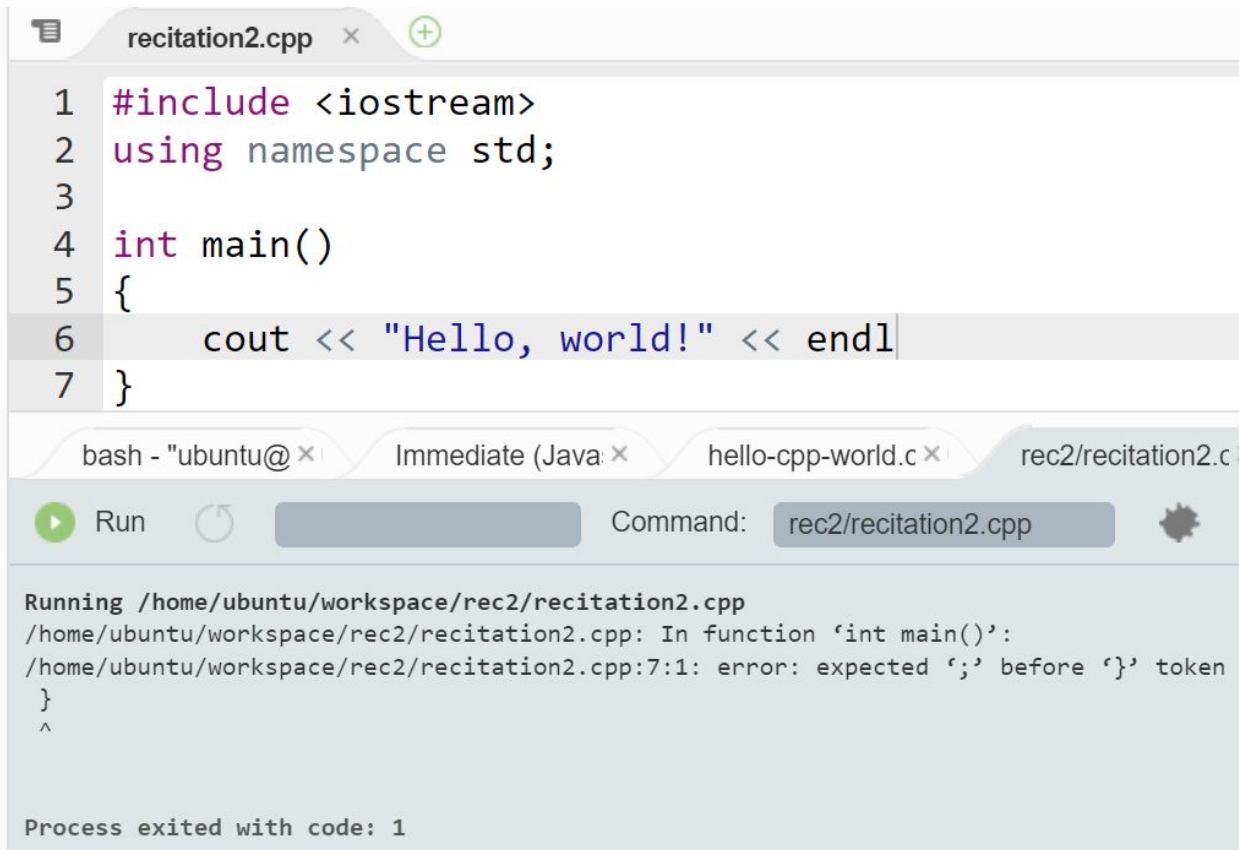
Moodle-

Check

Syntax Error(s)

prog.cpp:4:1: error: expected ';' before 'int'
int main()
^

Cloud9-



```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     cout << "Hello, world!" << endl
7 }
```

bash - "ubuntu@ x Immediate (Java: x hello-cpp-world.c x rec2/recitation2.c

Run Command: rec2/recitation2.cpp

Running /home/ubuntu/workspace/rec2/recitation2.cpp
/home/ubuntu/workspace/rec2/recitation2.cpp: In function 'int main()':
/home/ubuntu/workspace/rec2/recitation2.cpp:7:1: error: expected ';' before '}' token
}
^

Process exited with code: 1

Moodle-

Check

Syntax Error(s)

```
prog.cpp: In function 'int main()':
prog.cpp:7:1: error: expected ';' before '}' token
}
^
```

Here we forgot a semicolon in two different places. In both cases the error message tells us “expected ‘;’ before...”. We don’t want to put a semicolon at the beginning of the indicated lines. Instead, we can look for a missing semicolon at the end of a preceding line.

Example 3: Misspelled Reserved Word:

Misspelling reserved words (int, main, cout, endl) can cause a variety of errors in C++.

Cloud9-

```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     cout << "Hello, world!" << end;
7 }
```

Running /home/ubuntu/workspace/rec2/recitation2.cpp
/home/ubuntu/workspace/rec2/recitation2.cpp: In function 'int main()':
/home/ubuntu/workspace/rec2/recitation2.cpp:6:29: error: no match for 'operator<<' (operand type mismatch)
 cout << "Hello, world!" << end;
 ^

Moodle-

Check

Syntax Error(s)

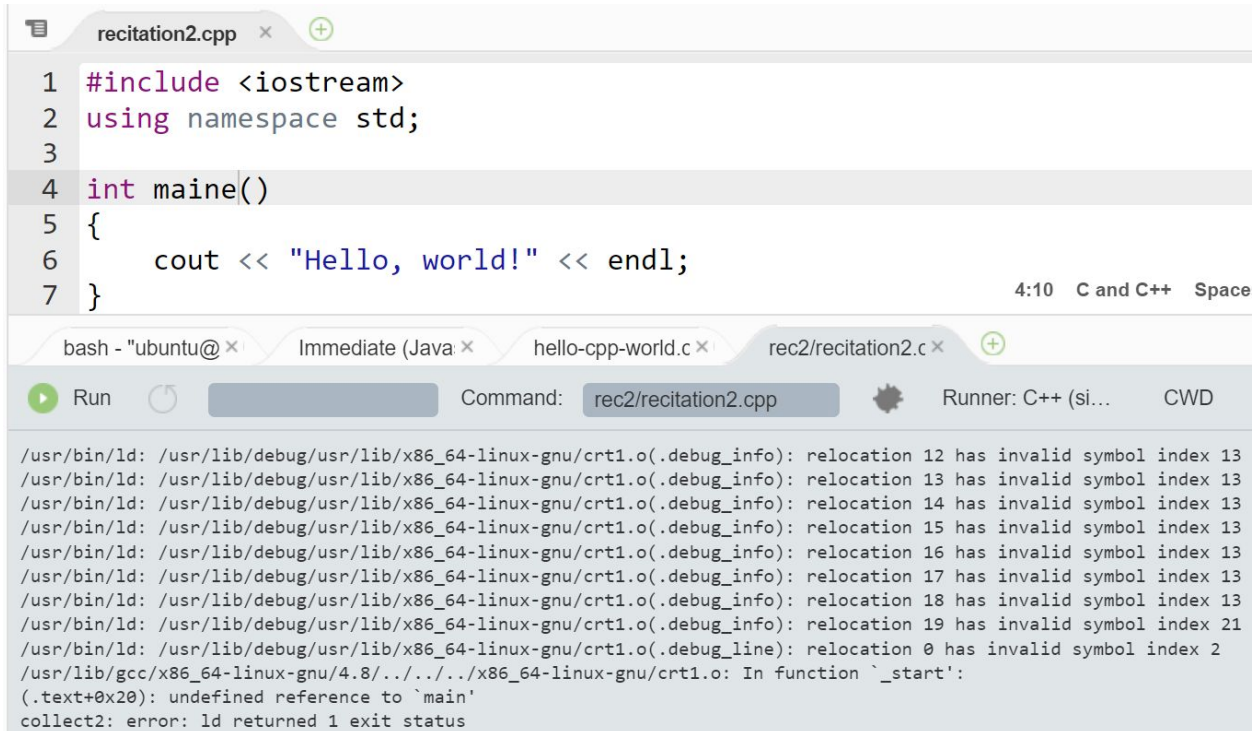
prog.cpp: In function 'int main()':
prog.cpp:6:39: error: 'end' was not declared in this scope
 cout << "Hello, CS1300 World!" << end;
 ^

In this case we forgot the “l” in “endl”. A lot of output is produced in Cloud9. Don’t let this scare you! Looking at the errors one at a time, it’s clear that there’s confusion around the “<<” operator and the “end”. Knowing nothing about exactly what “<<” is or how it works, we can carefully examine this region of our code and notice our spelling mistake.

The error produced in Moodle is slightly different. In particular, it tells us directly that we are trying to use some variable called “end” which has not been defined. This is a very common message when a variable name has been misspelled.

Example 4: No main() Function:

Cloud9-



```
recitation2.cpp x +
1 #include <iostream>
2 using namespace std;
3
4 int maine()
5 {
6     cout << "Hello, world!" << endl;
7 }
```

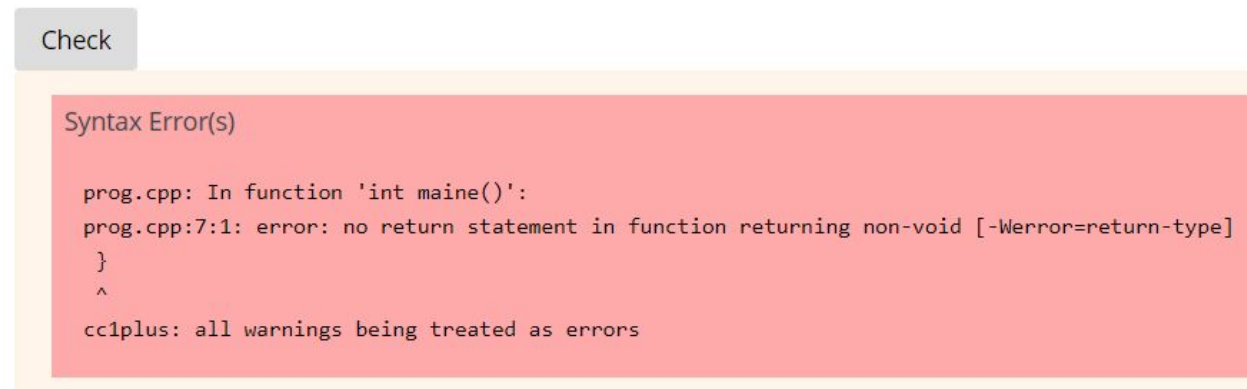
4:10 C and C++ Space

bash - "ubuntu@x Immediate (Java: x hello-cpp-world.c x rec2/recitation2.c x +

Run Command: rec2/recitation2.cpp Runner: C++ (si... CWD

```
/usr/bin/ld: /usr/lib/debug/usr/lib/x86_64-linux-gnu/crt1.o(.debug_info): relocation 12 has invalid symbol index 13
/usr/bin/ld: /usr/lib/debug/usr/lib/x86_64-linux-gnu/crt1.o(.debug_info): relocation 13 has invalid symbol index 13
/usr/bin/ld: /usr/lib/debug/usr/lib/x86_64-linux-gnu/crt1.o(.debug_info): relocation 14 has invalid symbol index 13
/usr/bin/ld: /usr/lib/debug/usr/lib/x86_64-linux-gnu/crt1.o(.debug_info): relocation 15 has invalid symbol index 13
/usr/bin/ld: /usr/lib/debug/usr/lib/x86_64-linux-gnu/crt1.o(.debug_info): relocation 16 has invalid symbol index 13
/usr/bin/ld: /usr/lib/debug/usr/lib/x86_64-linux-gnu/crt1.o(.debug_info): relocation 17 has invalid symbol index 13
/usr/bin/ld: /usr/lib/debug/usr/lib/x86_64-linux-gnu/crt1.o(.debug_info): relocation 18 has invalid symbol index 13
/usr/bin/ld: /usr/lib/debug/usr/lib/x86_64-linux-gnu/crt1.o(.debug_info): relocation 19 has invalid symbol index 21
/usr/bin/ld: /usr/lib/debug/usr/lib/x86_64-linux-gnu/crt1.o(.debug_line): relocation 0 has invalid symbol index 2
/usr/lib/gcc/x86_64-linux-gnu/4.8/../../../../x86_64-linux-gnu/crt1.o: In function `_start':
(.text+0x20): undefined reference to `main'
collect2: error: ld returned 1 exit status
```

Moodle-



```
Check
```

Syntax Error(s)

```
prog.cpp: In function 'int maine()':
prog.cpp:7:1: error: no return statement in function returning non-void [-Werror=return-type]
}
^
cc1plus: all warnings being treated as errors
```

This is another error that produces a lot of output in Cloud9. The important part this time is near the bottom where it says “undefined reference to ‘main’”. When a C++ program is run, execution starts in the “main()” function. When you try to run a program without a main, this is the kind of error that you might get. Looking at line 4, where we thought main was defined, we see another typo. After fixing this problem, the program should run.

In Moodle, the error is slightly different. We have not discussed how to define functions in C++ yet. In this case we have tried to define a general function, called “maine()” and not “main()”, with return type “int” which does not return anything. Moodle notices this and

throws an error. At this point, the important thing to notice is that something strange is going on with “int main()”. Again, the given line number is not particularly useful in this case.

Part 2: Surface Area of a Sphere

Example 1: Missing Semicolons:

In this example, the very last line of the function (line 7) is missing a semicolon. The error that Moodle returns indicates that the compiler is expecting to see a ‘;’ before the function ends with a ‘}’.

Moodle-

```
1 void findSphereStats(float radius)
2 {
3     float volume;
4     float surface_area = 0;
5     volume = (4.0/3.0) * M_PI * pow(radius, 3);
6     cout << "volume: " << volume << endl;
7     cout << surface_area << endl
8 }
```

Check

Syntax Error(s)

```
prog.cpp: In function 'void findSphereStats(float)':
prog.cpp:15:1: error: expected ';' before '}' token
}
^
```

Cloud9-

```
/home/ubuntu/workspace/rec2/rec2.cpp:11:1: error: expected ';' before '}' token
}
^
```

Example 2: Typo in variable name

In this example, I named my variable for volume “volume” on line 3. When I attempted to print it out on line 5, I misspelled it as “volumee”. Since volume and volumee are two different strings, C++ will not recognize the misspelling. Thus, I received a variable was not declared in this scope error. Note that in the last line, `cout << “volume: ” << volumee << endl;`, we have the string version of the word volume and the variable that represents the value of volume. Whenever something is written in quotations, that is not interpreted as anything other than a literal string, whereas if the word is not written in quotations, it will be interpreted as a variable.

Cloud9-


```
/home/ubuntu/workspace/rec2/rec2.cpp:8:26: error: 'volumee' was not declared in this scope
cout << "volume: " << volumee << endl;
                        ^
```

Moodle-

```
1 void findSphereStats(float radius)
2 {
3     float volume;
4     volume = (4.0/3.0) * M_PI * pow(radius, 3);
5     cout << "volume: " << volumee << endl;
6 }
```

Check

Syntax Error(s)

```
prog.cpp: In function 'void findSphereStats(float)':
prog.cpp:12:26: error: 'volumee' was not declared in this scope
cout << "volume: " << volumee << endl;
                        ^
```

Example 3: Misspelled Reserved Word

In this example, we misspelled the reserved word “cout”, meaning **character/console output**. C++ will see the misspelling “cuot” and treat it as something user-defined as it is no longer one of the reserved words. Since we never defined cuot as anything, it thus indicates that the word cuot was never declared.

Moodle-

```
1 void findSphereStats(float radius)
2 {
3     float volume;
4     volume = (4.0/3.0) * M_PI * pow(radius, 3);
5     cuot << "volume: " << volume << endl;
6 }
```

Check

Syntax Error(s)

```
prog.cpp: In function 'void findSphereStats(float)':
prog.cpp:12:4: error: 'cuot' was not declared in this scope
cuot << "volume: " << volume << endl;
   ^
```


Cloud9-

```
/home/ubuntu/workspace/rec2/rec2.cpp:8:4: error: 'cuot' was not declared in this scope
cuot << "volume: " << volume << endl;
^
```

Example 4: Misspelled Include Statement

When coding in the Moodle quiz, you will not be required to include C++ libraries. For instance, in the function call `pow(radius, 3)` we use the `pow` function, which comes from the `math.h` library. In Moodle, this will already be included for you, but when coding in cloud9, we need to be sure to include any necessary libraries in order for the code to compile. So in order to use `pow`, we will place the line `#include <math.h>` at the top of our file. Now let's assume we misspelled something in this statement.

In this first case, we misspelled include as incude:

```
1 #include <iostream>
2 #incude <math.h>
3 using namespace std;
4
5 void findSphereStats(float radius)
6 {
7     float volume;
8     volume = (4.0/3.0) * M_PI * pow(radius, 3);
9     cout << "volume: " << volume << endl;
10 }
11
```

bash - "chelseact" x Immediate x hello-cpp-world.c x hello-cpp-world.c x hello-csci-1300.c x

Run Run Config Name Command: rec2/rec2.cpp Run

Running /home/ubuntu/workspace/rec2/rec2.cpp
/home/ubuntu/workspace/rec2/rec2.cpp:2:2: error: invalid preprocessing directive #incude
#incude <math.h>
^
/home/ubuntu/workspace/rec2/rec2.cpp: In function 'void findSphereStats(float)':
/home/ubuntu/workspace/rec2/rec2.cpp:8:25: error: 'M_PI' was not declared in this scope
volume = (4.0/3.0) * M_PI * pow(radius, 3);
^
/home/ubuntu/workspace/rec2/rec2.cpp:8:45: error: 'pow' was not declared in this scope
volume = (4.0/3.0) * M_PI * pow(radius, 3);
^

In the second case, we included a library called `math`, not `math.h`:

```
1 #include <iostream>
2 #include <math>
3 using namespace std;
4
5 void findSphereStats(float radius)
6 {
7     float volume;
8     volume = (4.0/3.0) * M_PI * pow(radius, 3);
9     cout << "volume: " << volume << endl;
10 }
11
```

bash - "chelseact" x Immediate x hello-cpp-world.c x hello-cpp-world.c x hello-csci-1300.c x

Run Run Config Name Command: rec2/rec2.cpp Run

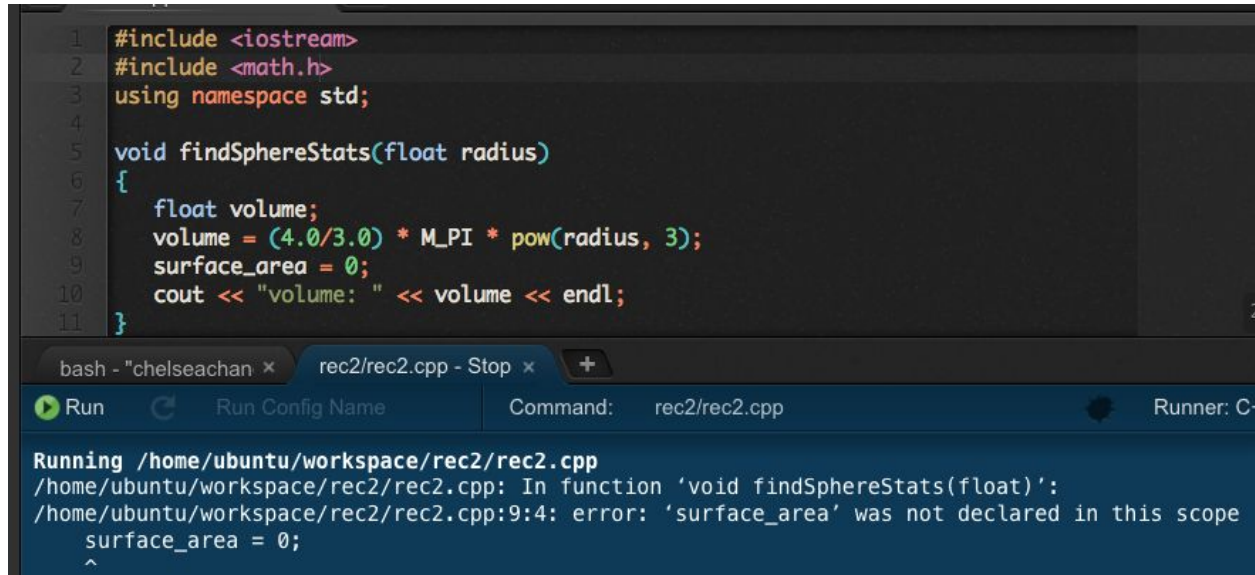
Running /home/ubuntu/workspace/rec2/rec2.cpp
/home/ubuntu/workspace/rec2/rec2.cpp:2:16: fatal error: math: No such file or directory
#include <math>
^
compilation terminated.

As you can see, these are two similar mistakes, but they lead to different error messages from the compiler. Thus, it is very important to be able to understand what each error message is talking about.

Example 5: Creating a variable before declaring the type

In this example, I attempted to define a surface area variable without first declaring its type. As we did with volume, we need to declare that it is a float before anything else.

cloud9-



```
1 #include <iostream>
2 #include <math.h>
3 using namespace std;
4
5 void findSphereStats(float radius)
6 {
7     float volume;
8     volume = (4.0/3.0) * M_PI * pow(radius, 3);
9     surface_area = 0;
10    cout << "volume: " << volume << endl;
11 }
```

bash - "chelseachan" x rec2/rec2.cpp - Stop x +

Run Run Config Name Command: rec2/rec2.cpp Runner: C++

Running /home/ubuntu/workspace/rec2/rec2.cpp

/home/ubuntu/workspace/rec2/rec2.cpp: In function 'void findSphereStats(float)':

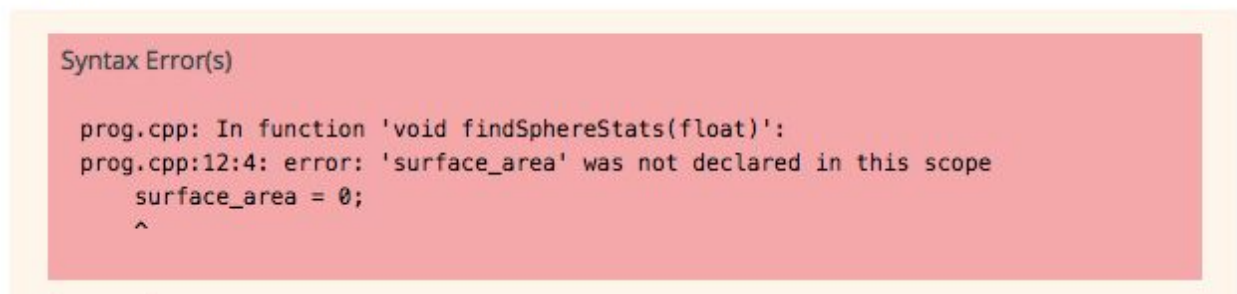
/home/ubuntu/workspace/rec2/rec2.cpp:9:4: error: 'surface_area' was not declared in this scope

surface_area = 0;

^

Moodle-

Check



```
Syntax Error(s)

prog.cpp: In function 'void findSphereStats(float)':
prog.cpp:12:4: error: 'surface_area' was not declared in this scope
    surface_area = 0;
    ^
```

Now there are two ways to do remedy this error:

The first is in two lines; the first with the declaration (*float volume*) and the second with the value of the variable.

The second is simply one line including all of the information: *float volume = (4.0/3.0) * M_PI * pow(radius,3);*

No matter the way that it is done, it is important to declare the variable as a specific type before use.

Example 6: Not following the rules about naming variables

- a) According to c++ standards, variable name can only start with a letter or an underscore ("_"), but not a number.

```
1 void findSphereStats(float radius){
2     float 3volume = (4.0/3.0) * M_PI * pow(radius, 3);
3     cout << "volume: " << 3volume << endl;
4 }
```

C:\Users\WillN\Desktop\BadExamples.cpp: In function 'void findSphereStats(float)':
C:\Users\WillN\Desktop\BadExamples.cpp:2:8: error: expected unqualified-id before numeric constant
 float 3volume = (4.0/3.0) * M_PI * pow(radius, 3);
 ^~~~~~

Moodle-

Syntax Error(s)

```
prog.cpp:10:10: error: invalid suffix "volume" on integer constant
    float 3volume = (4.0/3.0) * M_PI * pow(radius, 3);
           ^
```

Since all variable names must begin with either an uppercase/lowercase letter or an underscore, any variable names starting with numbers will cause an error. It is convention to write variable names in lower camelcase (ex: lowerCamelCase). If you want to have variables with numbers in them, place them somewhere after the first character (ex: volume3)

- b) Variable name cannot be one of the *reserved C++ words* (see lecture slides)

```
1 void findSphereStats(float radius){
2     float for = (4.0/3.0) * M_PI * pow(radius, 3);
3     cout << "volume: " << for << endl;
4 }
```

C:\Users\WillN\Desktop\BadExamples.cpp: In function 'void findSphereStats(float)':
C:\Users\WillN\Desktop\BadExamples.cpp:2:8: error: expected unqualified-id before 'for'
 float for = (4.0/3.0) * M_PI * pow(radius, 3);
 ^~~

Moodle-

```
Syntax Error(s)

prog.cpp: In function 'void findSphereStats(float)':
prog.cpp:10:10: error: expected unqualified-id before 'for'
    float for = (4.0/3.0) * M_PI * pow(radius, 3);
        ^
prog.cpp:11:26: error: expected primary-expression before 'for'
    cout << "volume: " << for << endl;
                        ^
```

Since variable names cannot be a reserved C++ word, if you named a variable “for” the compiler wouldn’t be able to differentiate between declaration of a for loop and your variable. Variable names can still start with or contain reserved C++ words as long as they differ somewhere (ex: forThisSphere). **A hint to not making this mistake in Cloud9 is that the IDE will auto-color variable names differently than it will color reserved words.**

Part 3: Runtime Errors

Runtime errors are errors that occur during the execution of a program. Unlike syntax errors and other compile-time errors, runtime errors could be caused by bad logic, running out of memory, infinite loops, etc.

Example 1: Missing return statement in function that must return a value

```

1 #include <iostream>
2 using namespace std;
3 #include <cmath>
4
5 float sphereVolume(float radius){
6     float volume;
7     volume = (4.0/3.0) * M_PI * pow(radius, 3);
8 }
9
10 int main () {
11     cout << sphereVolume(1.0) << endl;
12 }

```

The sphereVolume function in the above code snippet should return the volume as a float to main, but no return statement is included. In this case, running this program will not result in an error like the ones that we have seen before. Instead, the program will compile and run, but will result in undefined behavior. The function will return a value, but not the value you calculated and intended to return in your function.



The screenshot shows a code execution environment with a 'Run' button and a 'Command:' field. Below the command field, the output is displayed: 'Running /home/ec2-user/environment/week2/test.cpp' followed by '0' on the next line. At the bottom, it says 'Process exited with code: 0'.

For example, running the above code in cloud9 results in the function returning 0 when the expected volume is 4.18879. Similarly, running this code in Moodle would result in unexpected behavior and incorrect results.