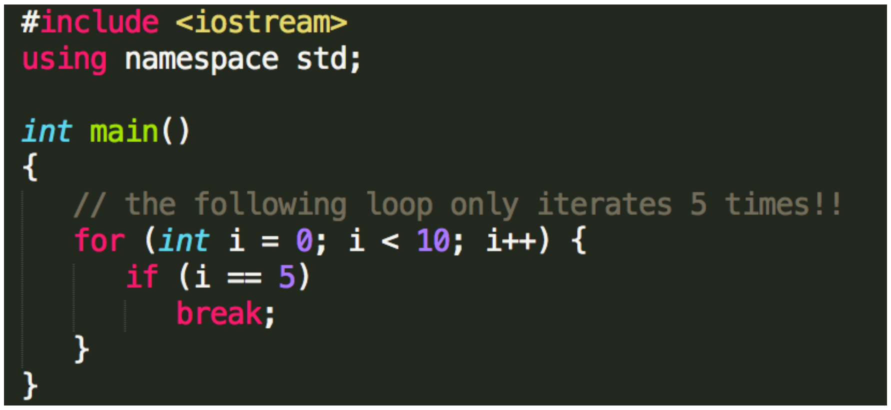
Disc 4

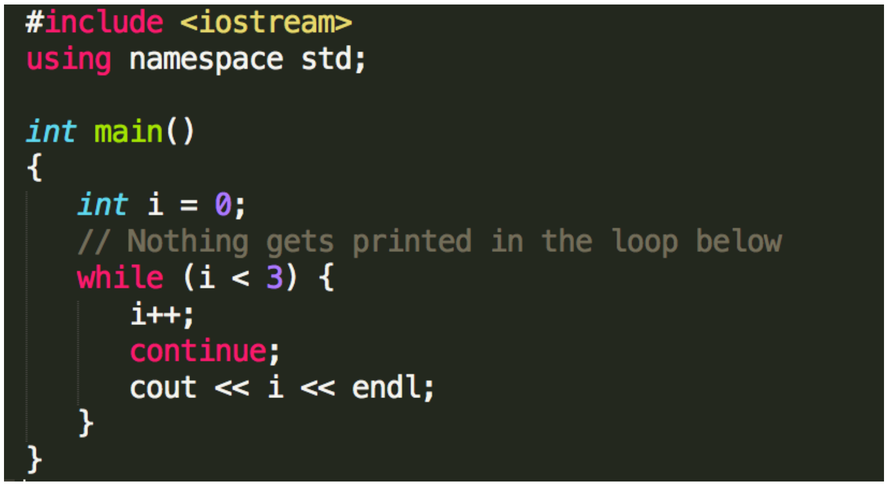
Wednesday, October 21, 2015 7:13 PM

A quick review of loops

- There are 3 types of loops:
  - for
  - while
  - do-while
- What is the main difference between a for and while loop?
- What is the main difference between a while and do-while loop?
- You can control the flow of loops with the continue and break statements
  - break immediately exits out of the innermost loop
  - continue causes the loop to skip the remaining code and start a new iteration



```cpp
#include <iostream>
using namespace std;

int main()
{
    // the following loop only iterates 5 times!!
    for (int i = 0; i < 10; i++) {
        if (i == 5)
            break;
    }
}
```

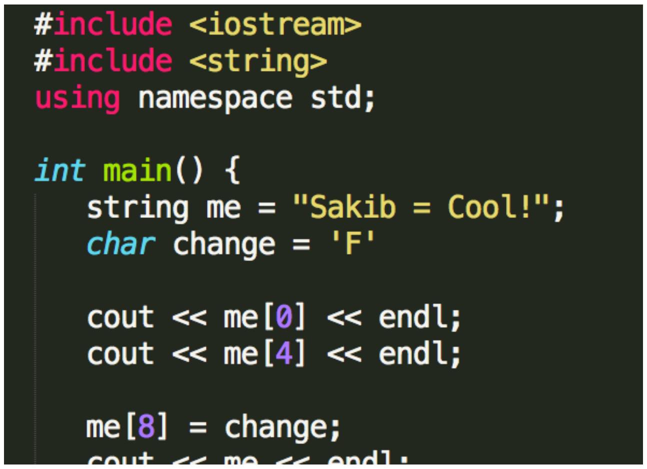


```cpp
#include <iostream>
using namespace std;

int main()
{
    int i = 0;
    // Nothing gets printed in the loop below
    while (i < 3) {
        i++;
        continue;
        cout << i << endl;
    }
}
```

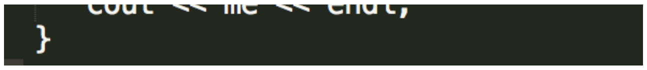
String essentials

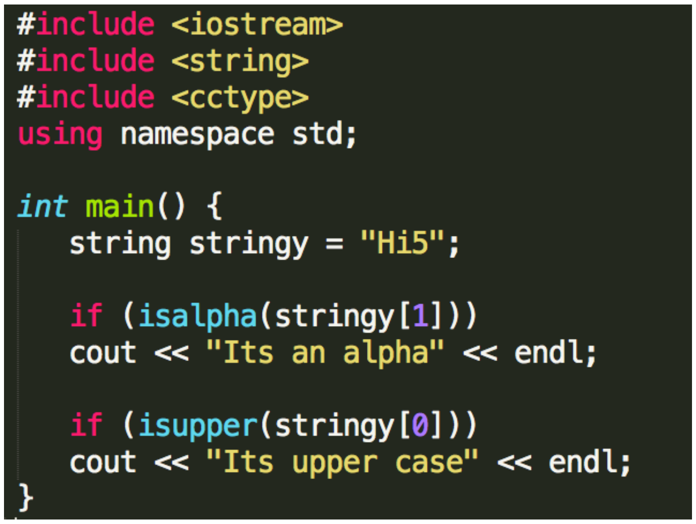
- There are several essential features in the string library.
- You can extract any character in a string with the [] operator
  - It takes the index of the character
- The index of the first character is 0.
- Given a string of length n, the last character is located at index (n-1).
- A single character can be stored in the char data type.
  - A char type is designated with **single quotes**
  - There is a useful library to work with chars that provides useful functions
    `#include <cctype>`
  - Useful functions from cctype
    - isnum
    - isalpha
    - tolower
    - toupper
    - And more: http://www.cplusplus.com/reference/cctype/
- You can get a substring of a string with the substr function

```
substr(s, l) // is the index to start extracting, l is
how many characters to extract

string s = "duplicate";   // duplicate
cout << s.substr(5,3);    // writes cat

// what if I don't specify how many characters to
extract?
```



```
#include <iostream>
#include <string>
using namespace std;

int main() {
    string me = "Sakib = Cool!";
    char change = 'F'

    cout << me[0] << endl;
    cout << me[4] << endl;

    me[8] = change;
```

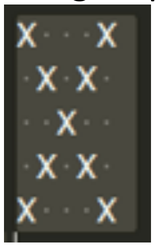

```
    cout << me << endl;
 }
```


```
#include <iostream>
#include <string>
#include <cctype>
using namespace std;

int main() {
   string stringy = "Hi5";

   if (isalpha(stringy[1]))
   cout << "Its an alpha" << endl;

   if (isupper(stringy[0]))
   cout << "Its upper case" << endl;
}
```

Lets put together some of what we learned with some example problems

Example 1: Big X

- Design a program that writes a large X made of little Xs on a 5 by 5 grid


```
X   X
 X X 
  X  
 X X 
X   X
```

- Can you make it so you print it for any n?

Example 2: Swap upper and lower case letters in a string

- Design a program that asks for a user input and changes all capital letters to lowercase and vice versa.

Example 3: Write a program to reverse a string

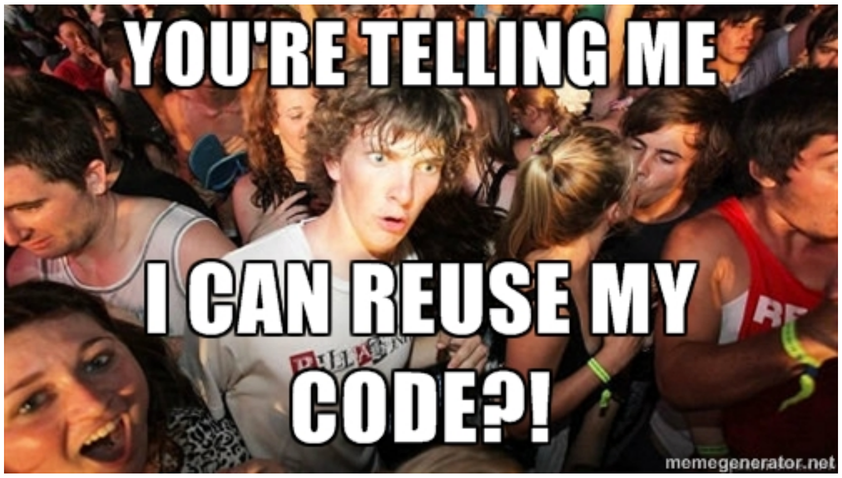
- Can you do it without an extra variable?

Solutions:

- Ex1: ideone.com/CnI44V
- Ex2: ideone.com/NpwVWj

Functions

- By now you might have noticed that you might be writing repetitive code
- You can package that repetitive code into functions. Then you can call that function as many times as you need to



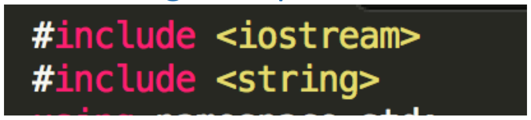
- In C++ functions work similarly to math functions.
  f(x) = x + x^2
  f(x,y,z) = x + y + z
  f(x) = 1
  - They have arguments/parameters
  - They do computations on those parameters
  - They return the results
- C++ expands and allows you to do more with functions.
  - You can pass in different argument types (booleans, chars, strings, etc)

Example 4: Write a function power that takes two arguments x, n and calculates x^n.

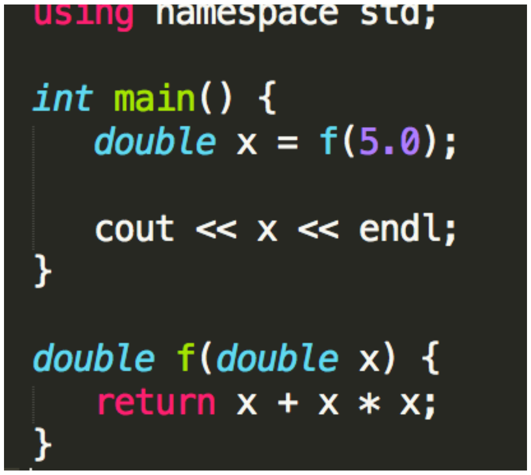
- What are some of the ways you can use a functions return values?
- What happens in the following example?

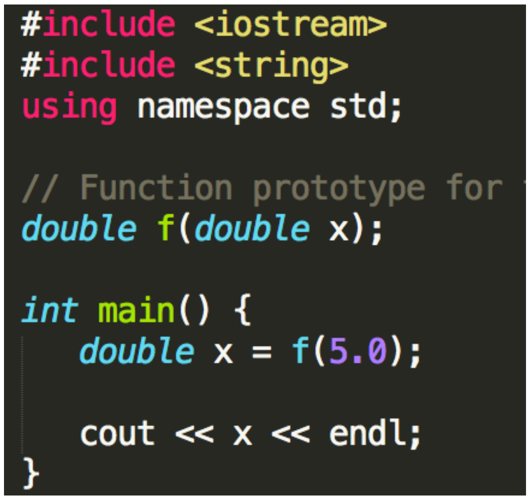
- **Function Prototypes** are hints to the compiler that say, "Hey, here's a function with a name, a return type, and some parameters... I'm not going to define it now, but if you use it, I promise I'll have it defined later!

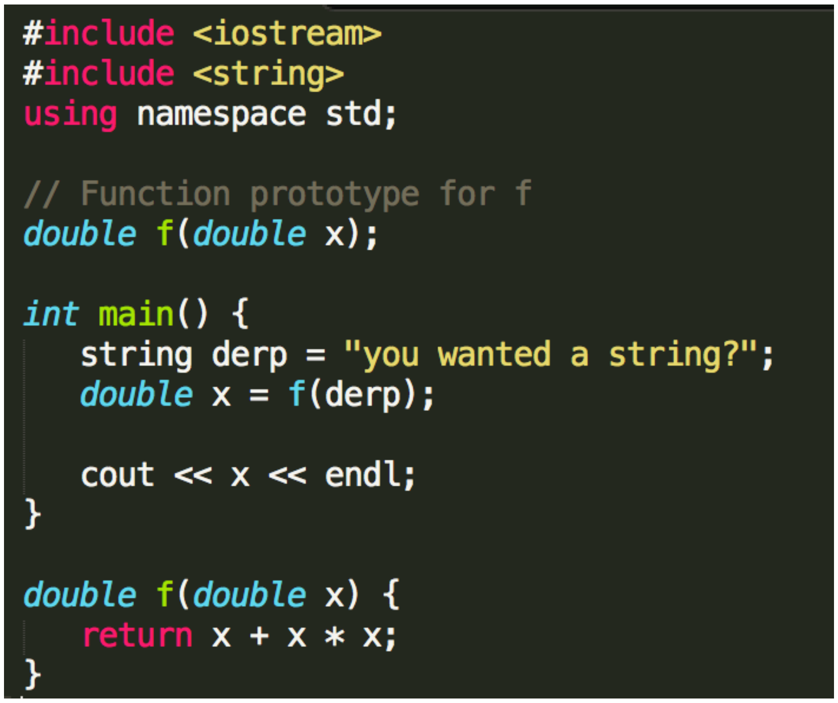
Are the following examples valid?

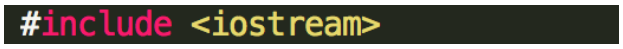


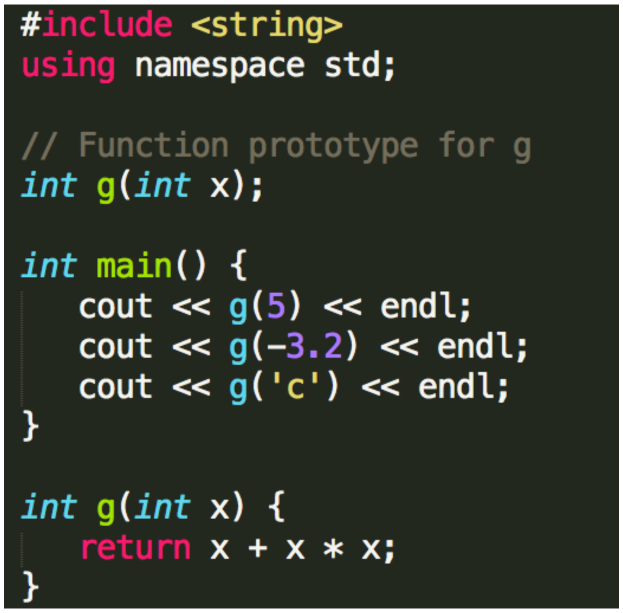
```
#include <iostream>
#include <string>
```


```
using namespace std;

int main() {
    double x = f(5.0);

    cout << x << endl;
}

double f(double x) {
    return x + x * x;
}
```


```
#include <iostream>
#include <string>
using namespace std;

// Function prototype for
double f(double x);

int main() {
    double x = f(5.0);

    cout << x << endl;
}
```


```
#include <iostream>
#include <string>
using namespace std;

// Function prototype for f
double f(double x);

int main() {
    string derp = "you wanted a string?";
    double x = f(derp);

    cout << x << endl;
}

double f(double x) {
    return x + x * x;
}
```


```
#include <iostream>
```



```
#include <string>
using namespace std;

// Function prototype for g
int g(int x);

int main() {
    cout << g(5) << endl;
    cout << g(-3.2) << endl;
    cout << g('c') << endl;
}

int g(int x) {
    return x + x * x;
}
```

Example 5: Write a function go get the nth number in a fibonacci sequence

0	1	2	3	4	5	6
0	1	1	2	3	5	8

A few more important details about functions

- Functions can call functions
- If you need find yourself repeating code in a function, you can write yet another helper function
- What if you want a function that doesn't return anything
- So far we can only return one value from a function
    - Can we return or change multiple values in a function?
    - How can we write a function to swap two values?

Passing by Reference

- A reference is another name for the original variable.
- If a variable is passed by reference to another function and that function changes the value of the variable, then it changes the original variable.

Example 6: Write a function `void strToNum(string s, int &num)` to convert a string that represents a number into an actual number and set it to the variable num.

```
string s = "12345"
int n;
```

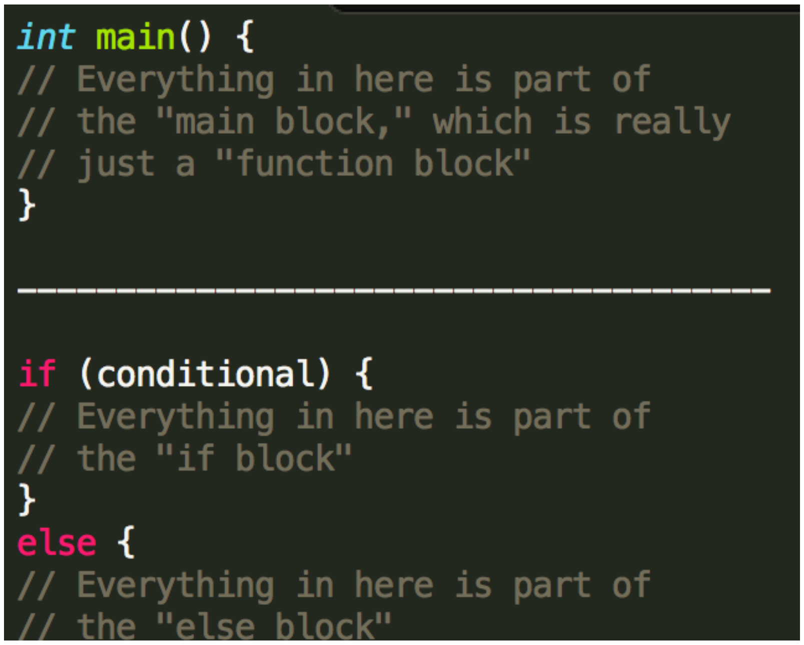
```
strToNum(s, n) // sets n to 12345.
```
- Return -1 if any non numbers in the string
- How can we make it work for negative numbers?

Scope

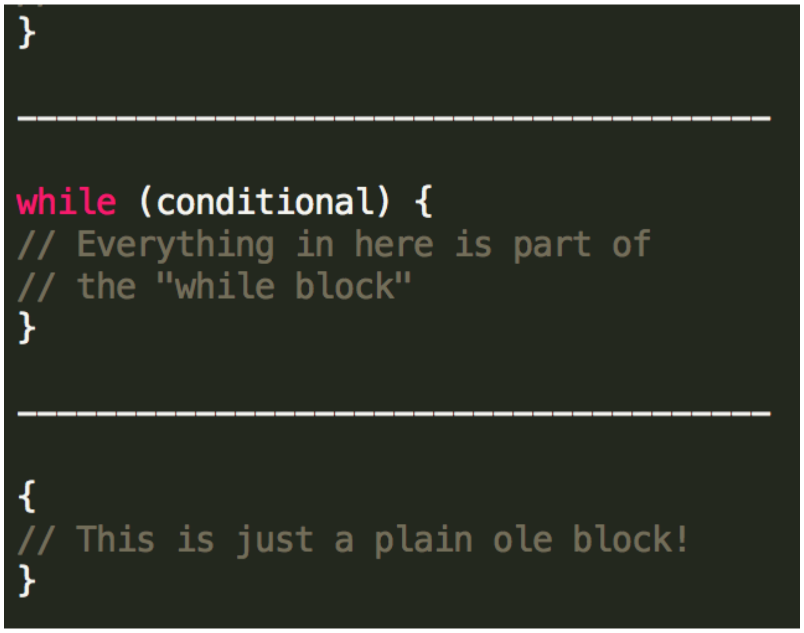
- If I declare a variable char actor in main and then in a helper function I declare another variable char actor. Is it OK?

- Variables have a "scope" of declaration.
- Scope is limited by the {brackets} they were declared in (unless if it is a global variable)
- AND Variables come into scope **AFTER** they are declared
  BUT!
- { A variable declared is available to the {brackets within} the brackets it is declared in }
- Once the scope of the variable ends, the variable's life ends ☹ and the memory is reclaimed

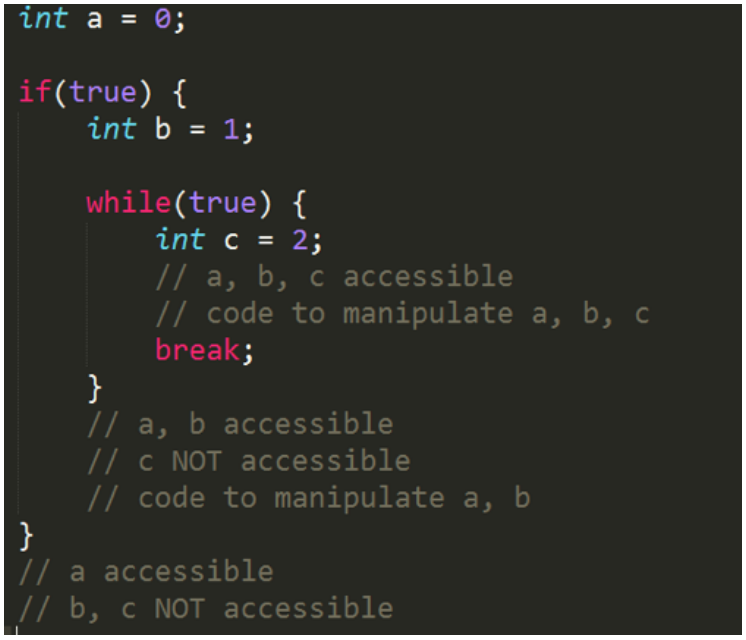
The general idea is
- Look in the current block, if it is use that definition. Otherwise
- Look for a definition of that identifier in the **next block up**. If it is defined there, then use that definition. Keep looking in the next enclosing block, and if I don't find it in the global space.



```
int main() {
// Everything in here is part of
// the "main block," which is really
// just a "function block"
}

------------------------------------

if (conditional) {
// Everything in here is part of
// the "if block"
}
else {
// Everything in here is part of
// the "else block"
```

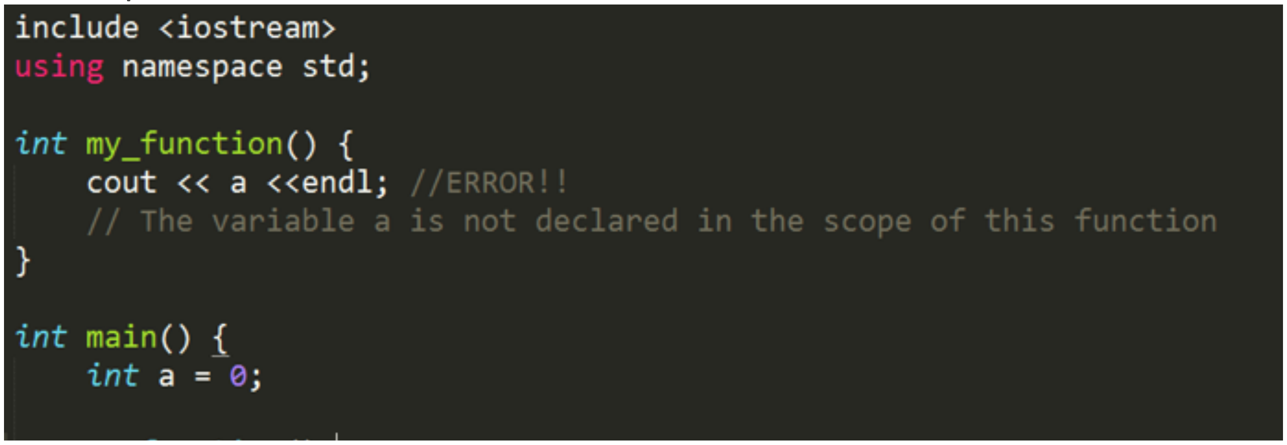


```
}

--------------------------------------

while (conditional) {
// Everything in here is part of
// the "while block"
}

--------------------------------------

{
// This is just a plain ole block!
}
```

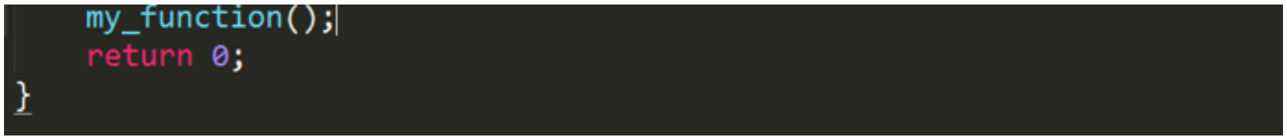


```
int a = 0;

if(true) {
    int b = 1;

    while(true) {
        int c = 2;
        // a, b, c accessible
        // code to manipulate a, b, c
        break;
    }
    // a, b accessible
    // c NOT accessible
    // code to manipulate a, b
}
// a accessible
// b, c NOT accessible
```

The scope of a variable does not extend to function calls

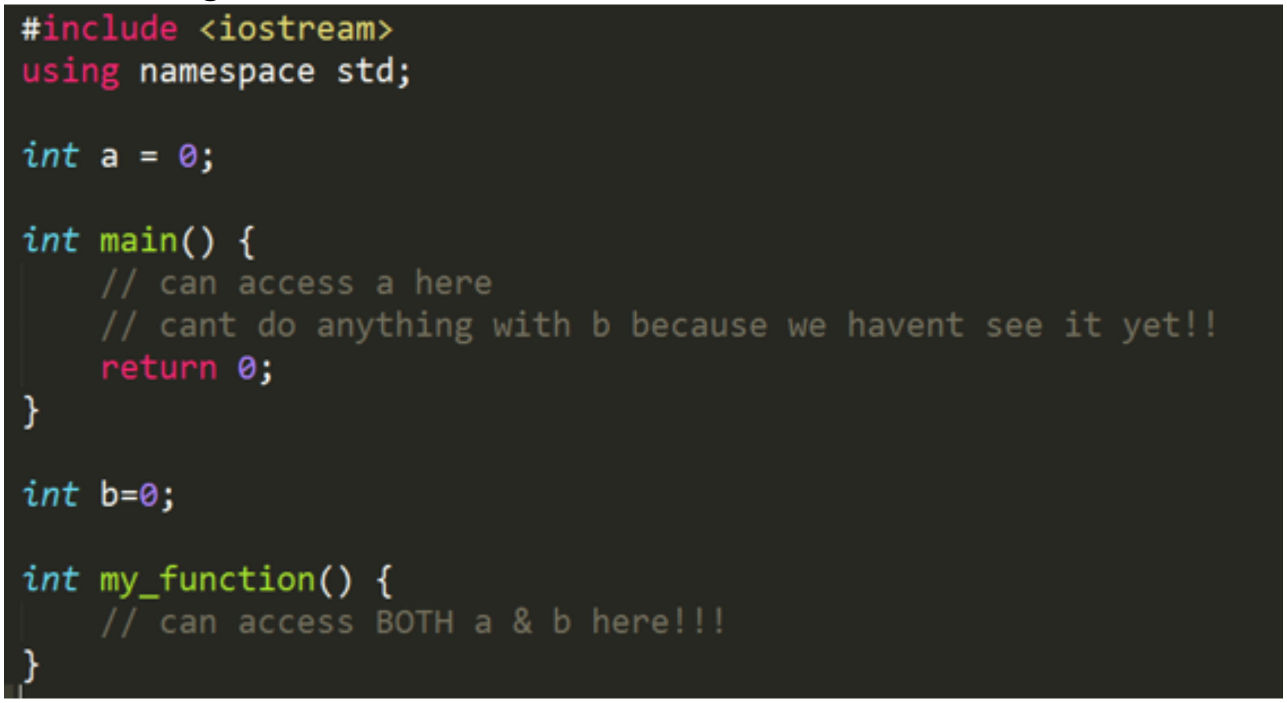


```
include <iostream>
using namespace std;

int my_function() {
    cout << a <<endl; //ERROR!!
    // The variable a is not declared in the scope of this function
}

int main() {
    int a = 0;
```



```
    my_function();
    return 0;
}
```

What about global variables?



```
#include <iostream>
using namespace std;

int a = 0;

int main() {
    // can access a here
    // cant do anything with b because we havent see it yet!!
    return 0;
}

int b=0;

int my_function() {
    // can access BOTH a & b here!!!
}
```

- Available to all functions defined after they are declared
- If global variable's value changed, it retains the new values.
- Best to avoid use of global variables.

Testing

What are some of the techniques you've been using to test and debug?

- How do you test if you got the correct output?

If you have a program that has many functions, it can be tedious and hard to test them. You can only use cout so much.
Since you already know what output to expect from functions based on a specific input, you should be able to just check that.
There is a library to help you do that: the cassert library
#include <cassert>

- Has only one function assert

- Checks if expression is true or false.
- Stops the program if it is false

Another useful tool to help you debug your program is your IDE.

- Full suite of testing tools in debug mode.
- Follow along and try your hand with this program. Find the bug
    - http://ideone.com/epRAIF
    - The program is intended to count the number of occurrences of each character in a string.

For a string like:

```
string s = "the cat in the hat";
```

The output should be

- order does not matter as long as count is correct
- Upper and lowercase letters different

```
t: 4
h: 3
e: 2
 : 4
c: 1
a: 2
i: 1
n: 1
```