**Functions**

A function is a block of code, separate from the main function. This code will not run until called upon in the main function. Useful for utilizing certain sections of multiple times. Fuctions can call/use other functions.   
  
 **Function Template:**

#include <iostream>

using namespace std;

**// DECLARATION**

return-type name (parameters);

**// FUNCTION CALL**

int main () {

Insert-code-here;

name(parameters);  
 }

**// DEFINITION**

Return-type’ ‘name’ (parameters) {

Insert-code-here;

}

**You can define & declare functions at the same time if done before main().**

**Function Example:**

*#include <iostream>*

*using namespace std;*

*int Numbers (int x);*

*int main () {*

*int y = 5;*

*cout << Numbers(y);* // function call (substitute *int y as int x*, and returned *int x as int y*) *}*

*int Numbers (int x) {* // int x == int a (5)

*return x + x;* // returns sum of int x and int x (y + y ((or 5 + 5)) *) as int y to main function*

***}***

**Function Return Types**

***int, double, char, and string*** - Used when you want to return a variable, without changing any of the variables. Think of it as the function working as its own new variable.

**Template:**

int name (variables) {

commands;  
 return variable/value

}

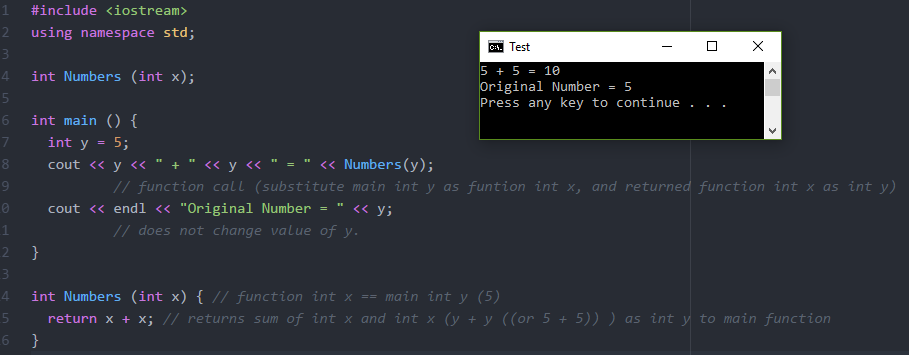
int main() {

Commands;

Cout << name (variables);

}

**Example:**

****

***bool*** - Used when you want to return a statement as true or false.

**Example:**

bool isValid(int x) {  
 if (x>=1)  
 return true;  
 return false;  
 }

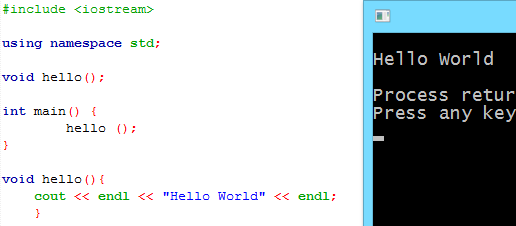
int main(){

int n = 2;  
 if (isValid(n) == true)  
 cout >> “true”;  
 else

cout << “false”;

***void***

- Most commonly used, it is a blank function, that will run and automatically return anything it is told to do when called upon. Will not change data unless using pass-by-reference.



**Memory and Pointers**

Every variable created in code takes up memory space and has a specific address.

**Variable Anatomy**:

X *(label*) = 1 (value) = stored in 0x28fedc (memory address)

*If you make a copy of X, such as when Y = X, Y will have a different memory address, thus taking more memory space*

**Reference Operator (&)**

The & symbol will return the memory location of a variable

**Example 1:**

*int x = 5;  
 cout << x; // Outputs variable value (5)*

*cout << &x; // Outputs mem location (0x28fedc)*

**Example 2:**

*int x = 5;*

*int xRef = &x;  
 cout << xRef; // Outputs mem location (0x28fedc)*

*cout << &xRef; // Outputs diff mem location (0x28fe18)*

**Pointers**:

Pointers are variables that store the address location of other variables.

*The pointer variable will also have its own unique memory address*

**Pointer Anatomy**:

int\* x-pointer = (memory address of variable) &x;

**Example:**

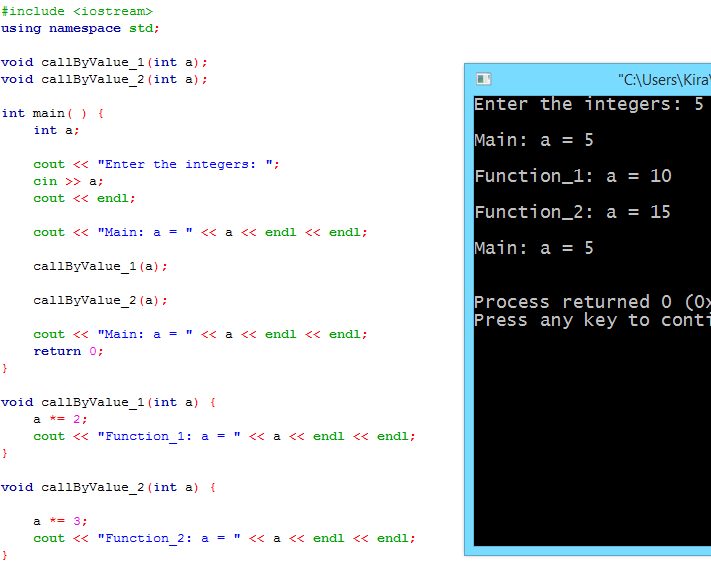
*int x = 5;*

*int\* xpointer = &x;  
 cout << x; // Outputs variable value (5)*

*cout << xpointer; // Outputs mem location of x(0x28fedc)*

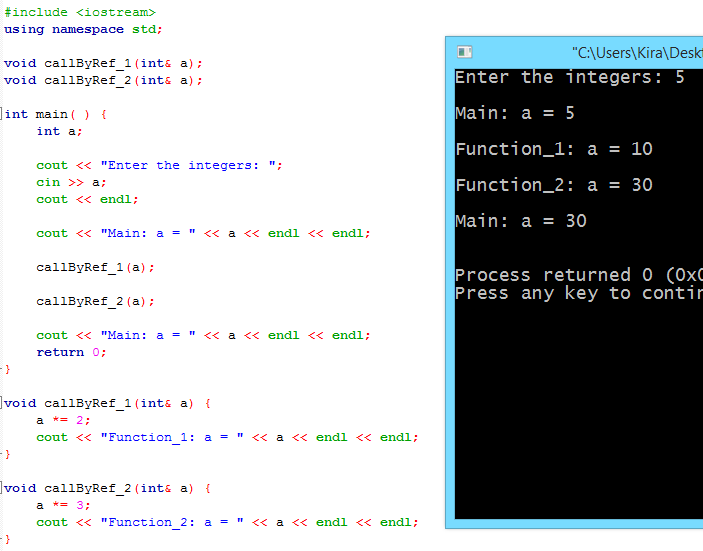
*cout << &xpointer; // Outputs diff mem location (0x28fe18)*

**Function Call/Pass by Value**



Used whenever variables in the main program or calling function **do not need to be updated** by the local function

**Function Call/Pass by Reference**

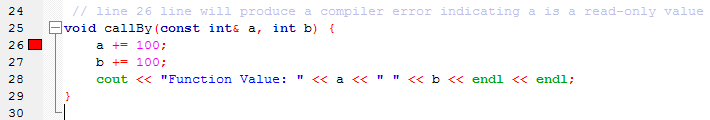


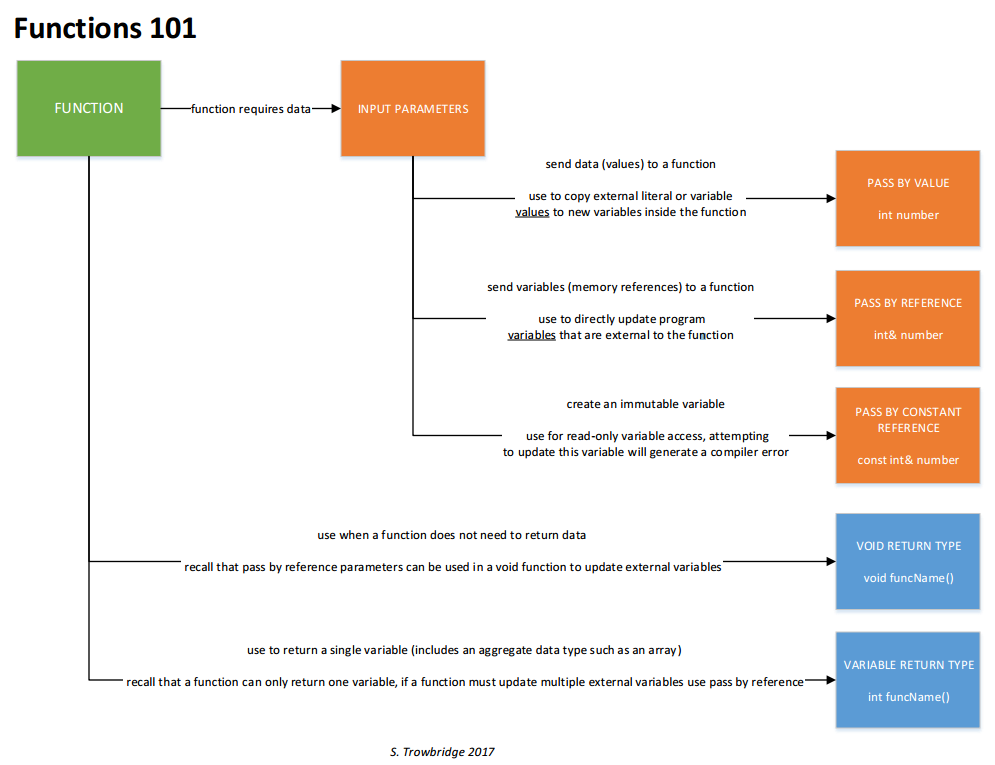
Used whenever variables in the main program or calling function **must be updated** by the local function

int& a – Calls the memory location of a and is replaces its value when going through the function call

**Function Call/Pass by Constant Reference**

Used whenever variables in the main function must be accessed without the ability to update or change the original variable.

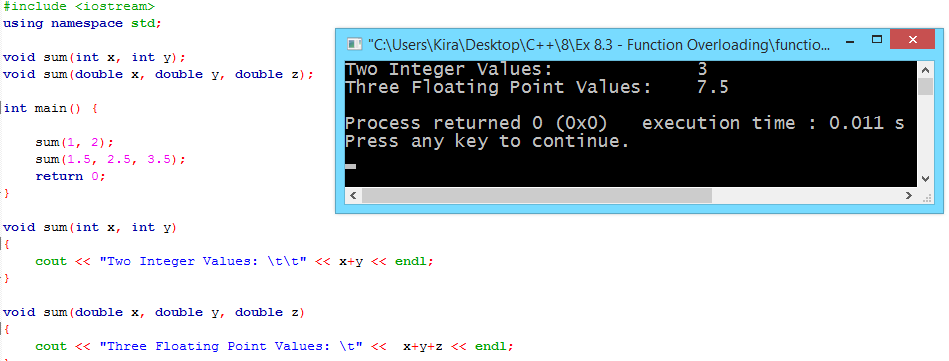




**Function Overloading**

When two functions share the same name. Program differentiates them based upon parameter needs, and only uses function with precise/exact matching parameters.

**Example:**

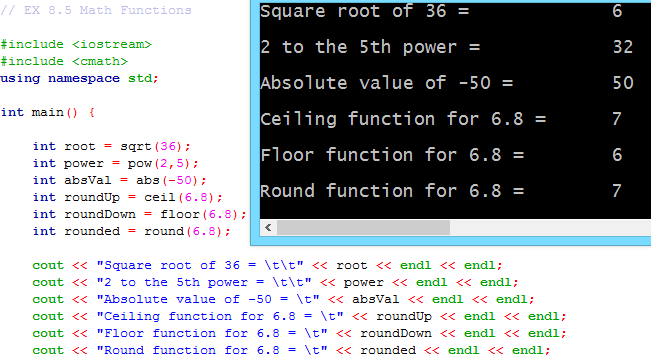
****

**Predefined Functions**

Functions made by other people! These functions can be accessed by calling libraries, which are called upon in the very beginning of the code. (#include <iostream> is an example of a libary)

***REFERENCE WEBSITE:*** [***http://www.cplusplus.com/***](http://www.cplusplus.com/)

**Math library Example:**



**Pseudo-Random Functions (Using libraries <cstdlib> and <time.h>)**

True randomization of numbers is impossible with computer generation. The reason is because the machine is following the same algorithm to generate that ‘random’ number. That is why any randomization done with a computer is called ‘Pseudorandomization’.

**PseudoRandom Anatomy**:

include <iostream>

include <cstdlib>

using namespace std;

int main(){

srand( ‘seed’ )

cout << rand() % ‘range (n – 1)’

**srand():** ‘Randomizes’ a number based upon a starting seed given.

**Seed:** The ‘seed’ is the number you give that the rand function will ‘generate’ a ‘random’ number from. Basically, each number has already been pre-defined to another ‘random’ number on an outside library function. So every time you use that ‘seed’ the same ‘random’ number will appear.  
 **rand():** Represents the random number srand creates. If there is no srand, rand will generate a number starting from the seed of 1 (which always ends up being randomized to 41), and if repeated or looped, will follow the same sequence of ‘random’ numbers each time program is run.

**Range:** Range will limit your ‘random’ number within the boundaries given by taking the random number equivalent and ‘modding (%)’ it to fit within the desired restriction. *(Remember to give one value more since sequence starts at 0. For example: if you want a range from 0 – 10, you need a range of 11)*

**Using Time as a Randomizer:**

include <iostream>

include <cstdlib>

include <time.h>

using namespace std;

int main(){

srand(time(NULL))

cout << rand() % ‘range (n – 1)’

**time(NULL):** will generate a number based upon the current time, and by setting that as our starting see, we can most closely get a randomized number.

Updated 9/12/2017