Functions and Parameter Passing

A function is a logical grouping of statements. It is a reusable chunk of code that you only have to write once but can use as much as you need to. Functions are beneficial in that they are reusable, work at a higher level of abstraction (it's much easier that way), reduce level of complexity and amount of code. Functions have many different names depending on your book or the context. Some of those names are, procedure, method (OOP), module, behavior (OOP), member function (OOP).

Purpose:
Do you notice anything familiar in the following code?

```
int myInt1, myInt2;
int sum;
cout<<"Enter 2 numbers: ";
cin>>myInt1;
cin>>myInt2;
sum = myInt1 + myInt2;
cout>>sum;

//more code
cout<<"Enter 2 numbers: ";
cin>>myInt1;
cin>>myInt2;
sum = myInt1 + myInt2;
cout>>sum;

//a lot of other code
cout<<"Enter 2 numbers: ";
cin>>myInt1;
cin>>myInt2;
sum = myInt1 + myInt2;
cout>>sum;
```

The lines of code that prompt the user, reads in the numbers, adds them together, and prints out the result is repeated; creating a function would get rid of these unnecessary redundant lines of code.

```
cout<<"Enter 2 numbers: ";
cin>>myInt1;
cin>>myInt2;
sum = myInt1 + myInt2;
cout>>sum;
```

Give the function a name and call it in your main program. Let's name it, `myFunction`.

```
int myInt1, myInt2;
int sum;
myFunction()

//more code
myFunction()

//a lot of other code
myFunction()
```

The code is written once but is used many times. Putting code into a function reduces the size of the code and makes it easier to comprehend – this is called *procedural abstraction*.

Scope of Variables:
Variables that are defined within a function can only be seen by that function, these variable have what is known as local scope. Take these two functions for example:

`fucntion1`	`function2`
`int myInt;`	`int num;`

`function1` cannot see `num` and `function2` cannot see `myInt` -- they have different scopes.

The Mantra:
return type, function name, parameters…
return type, function name, parameters…
return type, function name, parameters…
return type, function name, parameters…

Functions, like everything else in C++, have a skeleton:

```
<return type> <function name>(<parameters>)
{
    //cool code here
}
```

Return Types:
Functions need a way to send information back to their calling function – this is its return type.
If the function does not need to return any information, the return type is `void`. Otherwise, the return type is the same as the data type the function is going to return, such as `int`, `bool`, and `char`. To return information to the calling function, use the `return` keyword.

Function Name:

Naming a function is similar to naming a variable. Function names cannot start with numbers or contain funky characters except, _ and $. They should also begin with lowercase letters and for names with multiple words capitalize the first letter of the words, except for the first one, such as `function`, `myFunction`, `myMathFunction`.

Parameters:
Functions also need a way to receive information from their calling function – this is where parameters come in. Parameters are located between parentheses, ( ), and are special variables used to catch the information being passed. If there is nothing to be passed to the function, leave the parentheses empty. Parameters are the ***only*** way for the function to communicate with its calling function.

Let's put this all together. Here are some examples of how to declare functions. Remember the mantra!

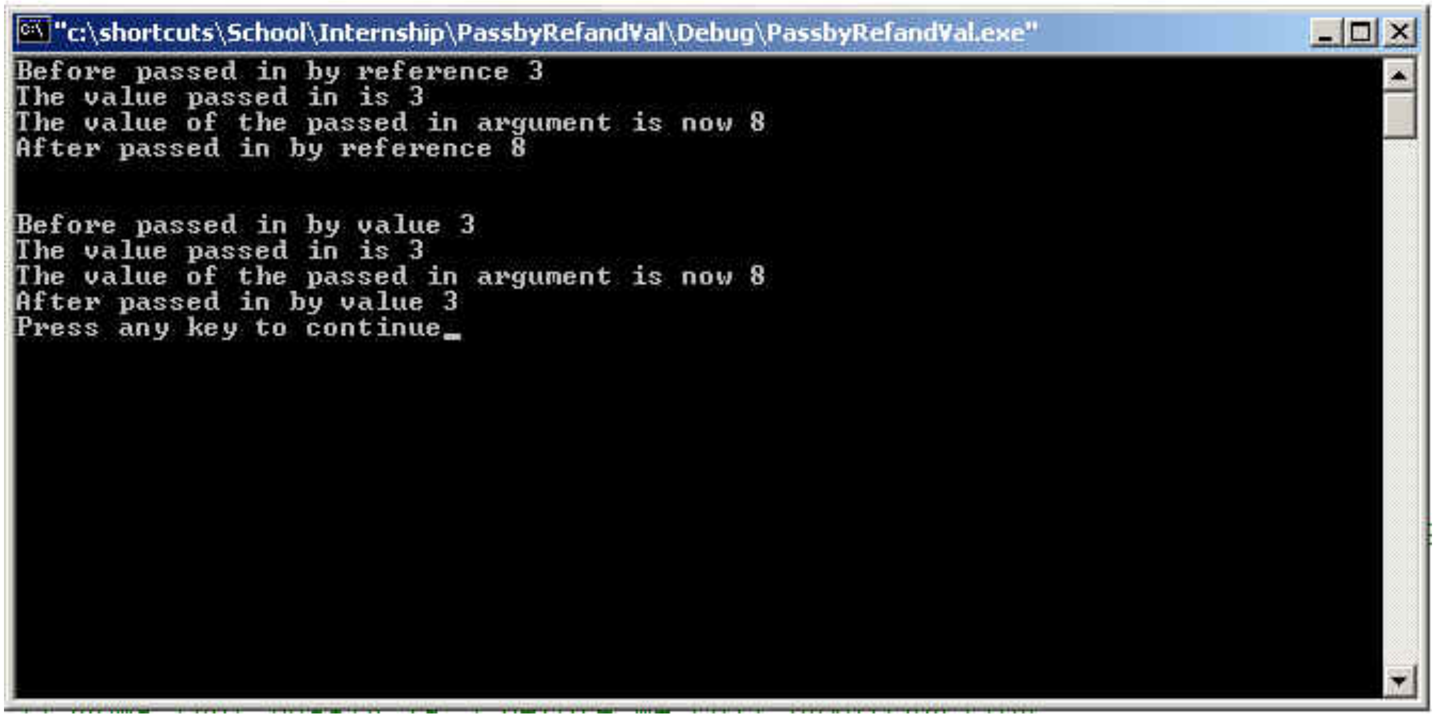
```
void print (int data){
     cout<<data;
}
//takes in an int and prints it.

int addSomething(int num1, int num2){
     int sum = num1 + num2;
     return sum;
}
/*takes in two numbers and adds them
together and returns the result in the
variable sum which needs to be put into
a variable in the calling function.*/

bool areYouAwake (){
     return true;
}
//returns a boolean
```

Rules:
Make sure you pass it the right information.
Make sure that the parameters match exactly (if the function asks for an `int`, don't pass it a `char`).
If the function returns something, do something with the value it returns.
Another function cannot be defined within a function.
Functions usually reside in a class (you'll learn about these later).
To put the function in the same class as the driver `prototype` it if it will be placed bellow main (tell the program there is a function coming up).
Functions cannot see the variables in another function.

Example:
Parameters can be passed by reference and by value.

```
#include <iostream>

using namespace std;

//Parameter passed in by reference. byRefFunction prototype
void byRefFunction (int & value);
//Parameter passed in by value. byValFunction prototype
void byValFunction (int value);

void main ()
{
int passIn = 3;
//Shows that passIn is 3 before we call byRefFunction
cout<<"Before passed in by reference "<<passIn<<endl;
//calls byRefFunction passing in passIn
byRefFunction (passIn);
//Shows that after byRefFunction is finished executing
passIns value changes
cout<<"After passed in by reference "<<passIn<<endl;
cout<<endl;
cout<<endl;
passIn = 3;
//Shows that passIn is 3 before we call byValFunction
cout<<"Before passed in by value "<<passIn<<endl;
//calls byValFunction passing in passIn
byValFunction (passIn);
//Shows that after byValFunction is finished executing
passIns value remains unchanged
cout<<"After passed in by value "<<passIn<<endl;
}

void byRefFunction (int & value)
{
//Shows that byRefFunction got what we passed in
cout<<"The value passed in is "<<value<<endl;
//changes value to value +5 within the function
value+=5;
//Shows that in the function the value changed
cout<<"The value of the passed in argument is now "
<<value<<endl;
}

void byValFunction (int value)
{
//Shows that byRefFunction got what we passed in
cout<<"The value passed in is "<<value<<endl;
//changes value to value +5 within the function
value+=5;
//Shows that in the function the value changed
cout<<"The value of the passed in argument is now "
<<value<<endl;
}
```

When you execute the above code you will get the window below:


"c:\shortcuts\School\Internship\PassbyRefandVal\Debug\PassbyRefandVal.exe"

```
Before passed in by reference 3
The value passed in is 3
The value of the passed in argument is now 8
After passed in by reference 8


Before passed in by value 3
The value passed in is 3
The value of the passed in argument is now 8
After passed in by value 3
Press any key to continue
```

As you can see above when you pass in a variable by **reference** and change its value the value of the variable changes for every one. Meanwhile when that same variable is passed in by **value** and the function changes its value the change is only seen by the function making the change.