Chapter 7: Functions

Functions are a programming approach to reduce your codebase and stop the need to replicate the same code in multiple places. If approaching C++ from another language, you may be familiar with the term methods, which are very similar to functions, however; C++ standard does not use methods per se but slightly different variations of functions. The C++ equivalent being of a method being a *member function*. The only real difference is a function, or a *free function* is called by name anywhere in the code, and *a member function/method* is associated with an object. Each function is usually a block of code that does a single task. We have already seen one function that has been in every program we have written so far – the main() function.

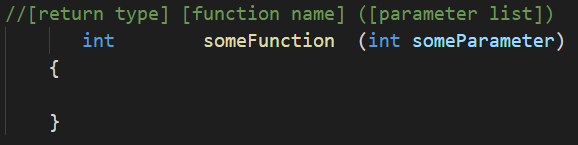
Functions can take any number of parameters and can return any valid type. This includes void types which signifies that nothing is returned. When declaring functions various approaches can be taken.

• The first is to put the function ahead of the main() function in code. This will ensure the compiler knows of its existence before the main() function attempts to use it.

• The second, and preferred, is to use a prototype, which is placed ahead of the main() function and then add the function body below the main() function. The prototype gives the compiler all the information it requires to allow the main() function to use the function within its own body of code.

• The final approach is to use header files and source files. In the header the prototypes are declared, and in the source file the body of each function is added. You need to include the header file at the top of the source file that uses the functions contained within. We will be using this approach when we get to Chapter on Object Oriented Programming.

**Format of a function:**

****

Note: Parameter is just another name for variables that are passed to the function.

The format of the prototype matches the function detailed above, but it ends with a semi-colon rather than the curly braces and code body. This is because it is only used to inform the compiler what the function name is, what the return type is and what the parameters are.

The format of a prototype is as follows:



It is important to note that any variables passed into a function as a parameter will be copied for use within the function. These are placed on the stack, and when the function returns the copies will be deleted. This is important to know because it means any variable passed into a function can is not changed within the function. We can solve this issue using pointers & references, but this will not be covered later weeks with Pointers and References.

**Example functions**

In all examples the programmer is expected to add the correct namespace at the top of the file or use the correct syntax.

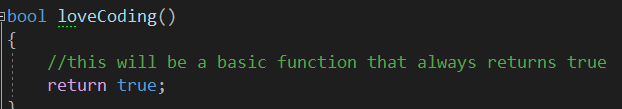
**I love Coding Function**

Firstly, we must add the function prototype. This goes above the int main() function:

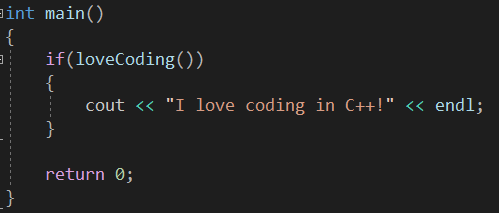


Next, we will add our function body. This will go under the main function after the last closing bracket.

Note: As we progress, we will be using header files and classes to construct our code and typically the main function and cpp file will be but a few lines of code.



Finally, we add code to the main function that uses our new function:



Build and run your program to make sure everything has been done correctly.

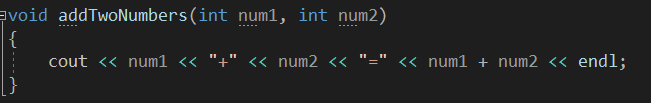
**Passing Parameters**

This example will create a function that takes two integers, adds them together and outputs them to the screen. It does not require a return type, so it will be void. We will create some local integers to be passed in, we will pass through some constant integers and finally pass through a mixture of both. Note that the parameter name does not need to be the same as the integer name passed in.

First, add the function prototype at the top:

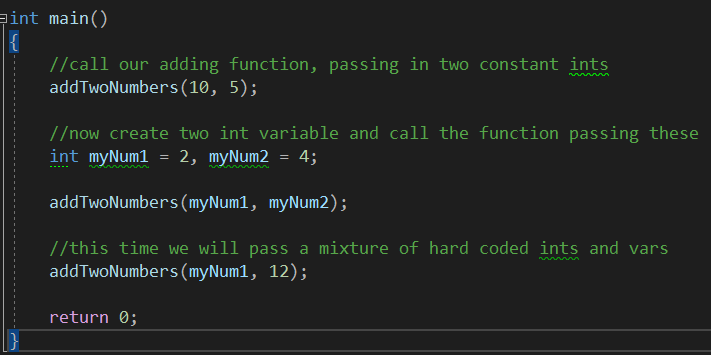


Next, add the function under the main like so:



Notice there is no return keyword. This is due to it being a void function and will automatically return at the end of the code.

And finally, we add our code calling the function to the main. We will do a couple of examples:



Build and run your program and make sure there are no errors. If there is anything you don’t understand about how the function is working now is the time to speak to a lecturer.

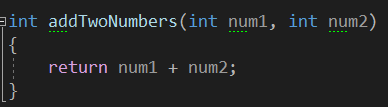
**Return Values**

This example will create a function that takes two integers, adds them together and returns the result. The result is then output to the console.

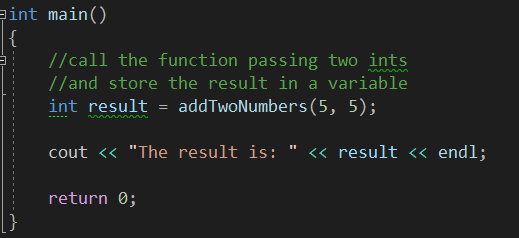
Add the prototype:



Next, add the function body:

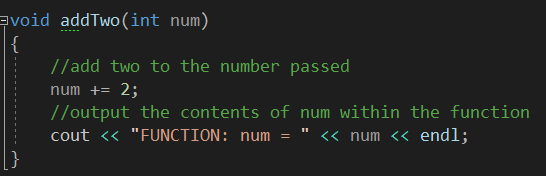


And finally, add our code that calls the function in the main:



**Program 17: Local Variables & Copies**

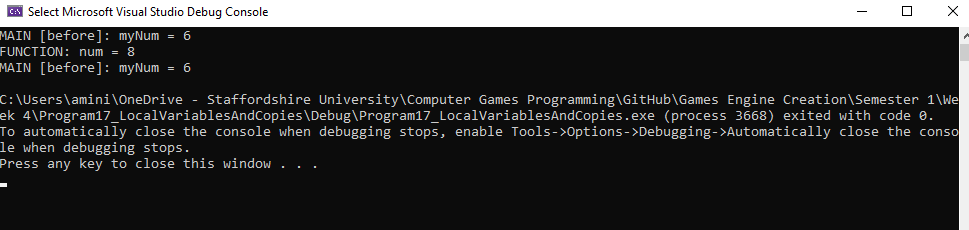
This program will demonstrate how variables do not get affected outside of functions regardless of what you do to the local copies a function makes.

1. Open Visual Studio
2. Create a new project called Program17\_LocalVariablesAndCopies
3. Create a function prototype called addTwo which takes one parameter of type int called num.
4. Add the function body below the main like so:  
   
5. Now for the code within main:
   1. Create an int variable called myNum with the value of 6.
   2. Output to screen the value of myNum before the calling of the function  
      
   3. Call the function, passing myNum:  
      
   4. Now repeat step b but [after] the function call.
6. Build and run the program. If all is correct your outputs should be 6, 8, 6. The value myNum has not been changed but rather a copy of its value is used within the function scope.

**Program 17 Source code:**

****

**Program 17 Screenshot:**

****

**Program 18: Smallest Number**

Write a program, which asks the user to input 2 integers. These values should be then passed into a function to determine which is the smallest. The function should return the smallest value.

This is the function prototype you must use:



Things to consider:

* Creating variables to hold user input
* Not sure how to test which is bigger? Look back at week 2 Conditionals
* Have your program ask for the three sets of numbers before exiting. Look back at week 3 Loops.
* How will you deal with numbers of the same value?

Note: This is a function exercise and the result MUST be determined within the function body and then returned from the function.

Run your program using the following values. The screenshot must show the use of these elements:

1, 2

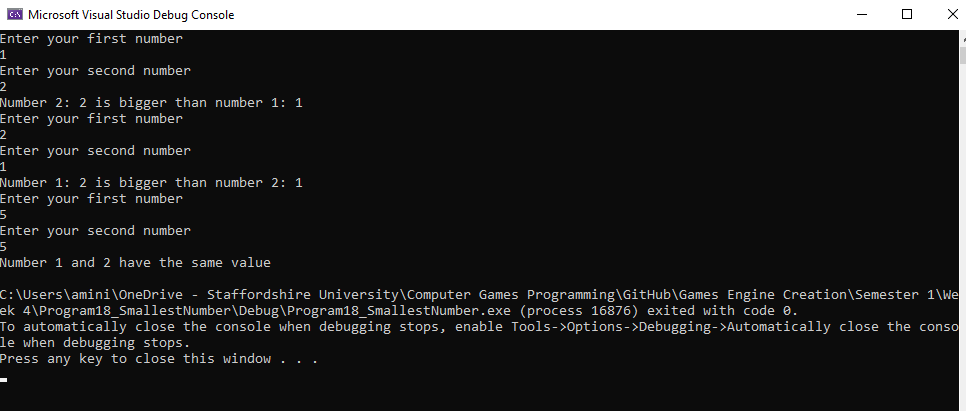
2, 1

5, 5

**Program 18 Source Code:**

****

**Program 18 Screenshot:**

****

**Program 19: Odds and Evens**

Write a program, which asks the user to input 10 integers. These values should be then passed individually to a function, which will determine whether the value is odd or even. This function should return a boolean value.

In the event of the value being odd, it should be added to an odd count, and in the event of it being even it should be added to the even count.

Your program should then call a function which will output how many odd numbers there were and the total, and in a similar manner for the even numbers. Use the following prototype for the output function:



Note: This is a function exercise and MUST demonstrate the use of functions as detailed above

**Things to consider:**

It might be useful to write some pseudo code in a notepad doc to plan out what the program needs,

* What local variables are needed?
* How will you ask for the 10 numbers?
* When will you call the Boolean is odd or even function?
* When will you call the outputResults function and what variables do you need to pass to it?
* The output should tell the user how many odd numbers were entered followed by the grand total of the odd numbers sum and the same with the even numbers.

**Program 19 Source code:**

****

**Program 19 Screenshot:**

****

**Program 20: Simple Text Battle**

Below you have been provided with some simple pseudo code for you to work from. The aim is to recreate the program in working code using everything you have covered in lectures and tutorials so far.

namespace

\*Global variables\*

two integers for player and enemy health set to 1000(p) and 2000(e)

boolean variable for playing set to true

\*Function prototypes\*

attack choice that returns nothing. Parameters required: one integer for choice

play state function that returns a boolean

\*Main Function\*

local variable to hold the players choice

A loop to run as long as playing is true

tell user they have met a troll and have 3 options of attack

1. use sword, 2. use magic, 3. use axe

tell user choices have different effects

store users choice

call attack choice function passing user choice

call play state function

return 0;

\*attack choice function\*

set constant ints for the passed user choice, sword damage = 300, magic\_damage = 650, axe\_damage = 450,

troll\_sword = 350, troll\_magic = 50, troll\_axe = 100;

switch statement(condition user choice)

case 1:

set enemy health to take sword damage

set player health to take troll sword damage

tell user they have hit the troll

tell user they have been hit

if player or enemy health is less than 0 set to 0 (prevents negative health)

tell user current player and enemy health

case 2:

set enemy health to take magic damage

set player health to take troll magic damage

tell user they have hit the troll

tell user they have been hit

if player or enemy health is less than 0 set to 0 (prevents negative health)

tell user current player and enemy health

case 3:

set enemy health to take axe damage

set player health to take troll axe damage

tell user they have hit the troll

tell user they have been hit

if player or enemy health is less than 0 set to 0 (prevents negative health)

tell user current player and enemy health

Any questions don’t hesitate to ask. Can you think of any improvements that could be made? Maybe check for wrong input etc? Ensure to full test your program and show this in your screenshots.

\*play state function\*

create a char for play again option

if enemy health is less than or equal 0

tell user they have killed the troll and won

ask to play again y/n

store choice

if they want to play again

reset enemy and player health

return playing true

else

return playing false

if player health less than or equal 0

tell user they have been killed

ask to play again y/n

store choice

if they want to play again

reset enemy and player health

return playing true

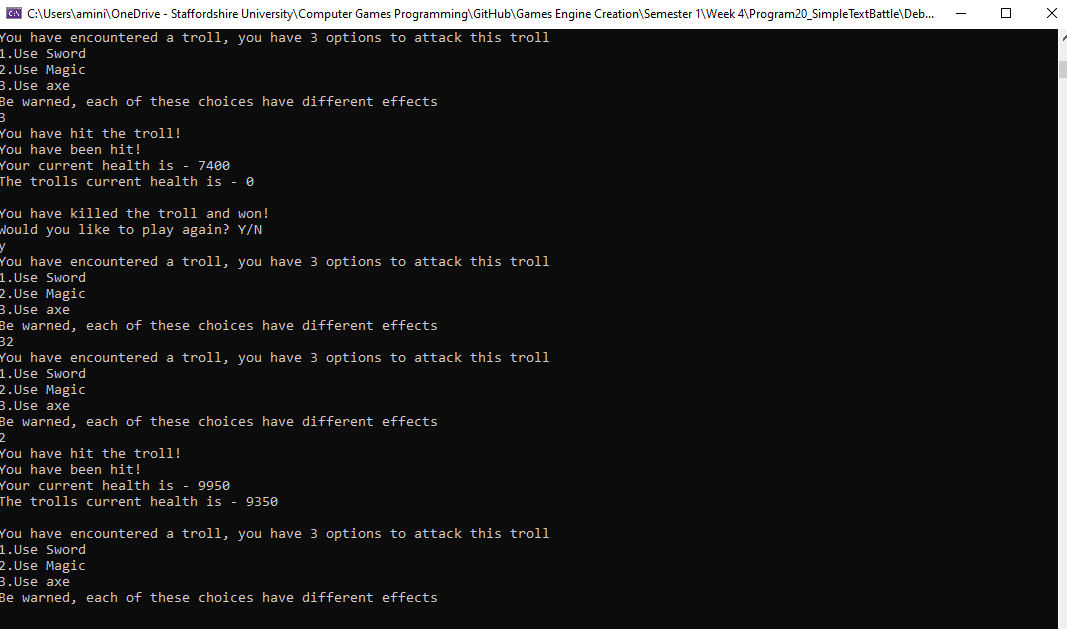
else

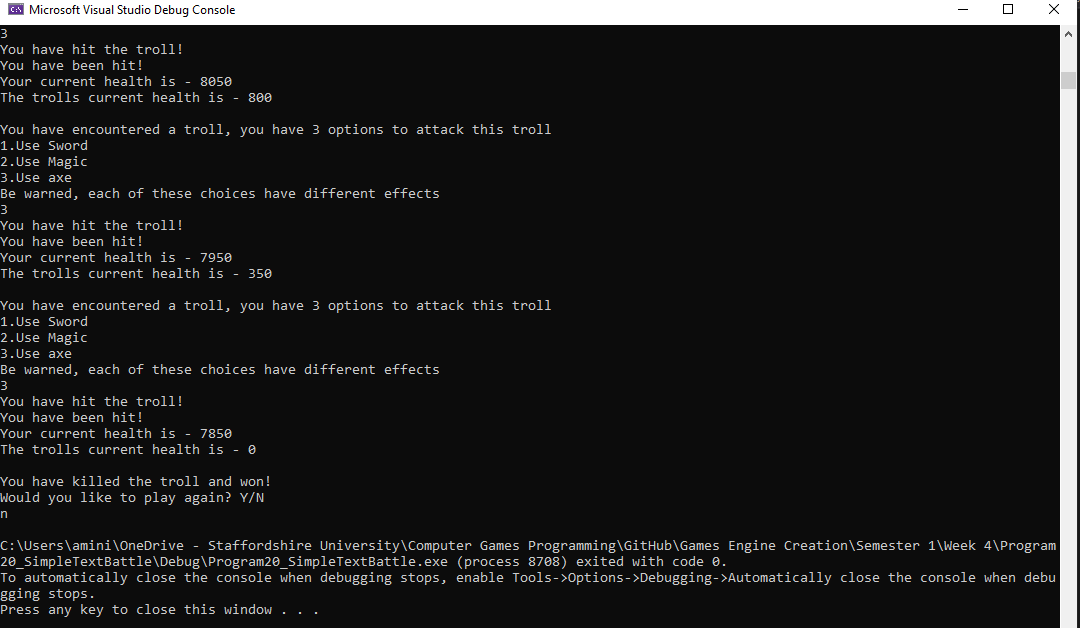
return playing false

return false;

**Program 20 Source Code:**

**Program 20 Screenshots:**

****

**’**