Chapter 7: Functions

Functions, also known as methods, is a programming approach to reduce your code base and stop the need to replicate the same code in multiple places. 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 there are various approaches that 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 is to use a prototype, which is placed ahead of the main() function and the 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 15:Object Oriented Programming.

The format of a function is as follows:

[return type] functionName ( [ parameter list ] )

{

// Code goes here.

}

Note: Parameter is just a name for variables that are passed into a function. These are placed in the parenthesis that follow the function name.

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:

[return type] functionName ( [ parameter list ] );

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 not be changed within the function. We can solve this issue using pointers & references, but this will not be covered until Chapter 12 Pointers and Chapter 13 References.

Lets take a look at some examples. All the examples given below expect the programmer to include the relevant header files and use the appropriate namespaces.

7.1 Example: Happy Function

First we create the prototype for the function:

bool areYouHappy ( );

Then we create the main() function, which makes a call to this function.

int main()

{

// Output a line of code if the ‘areYouHappy()’ function returns true;

if( areYouHappy() )

cout << “I am happy” << endl;

return 0;

};

Finally we add the function body below the main() function.

bool areYouHappy ( )

{

// Always return true.

return true;

};

7.2 Example: 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. Here we have created a local integers to be passed in, we have also just passed through constant integers and passed through a mixture. Note that the parameter name does not need to be the same as the integer name passed in.

First we create the prototype for the function:

void addTwoNumbers ( int num1, int num2 );

Then we create the main() function, which makes a call to this function.

int main()

{

// Call the ‘addTwoNumbers()’ function passing through constant integers.

addTwoNumbers( 10, 5 );

// Call the ‘addTwoNumbers()’ function again using variables as parameters.

int myNum1 = 2, myNum2 = 4;

addTwoNumbers( myNum1, myNum2 );

// Call the ‘addTwoNumbers()’ function again using a mixture of

// variables and constants as parameters.

addTwoNumbers( myNum1, 10 );

return 0;

};

Finally we add the function body below the main() function. Notice there is no return keyword as there is nothing to return and the function will automatically return when it hits the end of the function.

void addTwoNumbers (int num1, int num2 )

{

// Output the result of the calculation.

cout << num1 << “ + “ << num2 << “ = “ << num1 + num2 << endl;

};

7.2 Example: 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.

First we create the prototype for the function:

int addTwoNumbers ( int num1, int num2 );

Then we create the main() function, which makes a call to this function.

int main()

{

// Call the ‘addTwoNumbers()’ function passing through constant integers.

int result = addTwoNumbers( 10, 5 );

// Output the result.

cout << “The result is: “ << result << endl;

return 0;

};

Finally we add the function body below the main() function. Notice this time we return the result of the calculation. The calculation will be carried out before the function returns.

void addTwoNumbers (int num1, int num2 )

{

// Return the result of the calculation.

return num1 + num2;

};

**Program 15: 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. To begin, start Visual Studio.
2. Create a new project via File -> New -> Project or Ctrl+Shift+N Name it “Chapter7\_LocalVariablesAndCopies”
3. Click **Next** and you should be greeted with the following screen. Make sure to have **Empty Project** ticked and click **Finish**.
4. Add a new source file and name it “LocalVariables.cpp”
5. Replicate program listing 15.

#include <iostream>

using namespace std;

void add2ToNumber( int num );

int main ()

{

int myNum = 0;

// Output the contents of myNum before the function call.

cout << “MAIN [before]: myNum = “ << myNum << endl;

// Call the function which will add 2 to the value.

add2ToNumber( myNum );

// Output the contents of myNum after the function call.

cout << “MAIN [After]: myNum = “ << myNum << endl;

return 0;

}

void add2ToNumber( int num )

{

// Add 2 to the variable passed in.

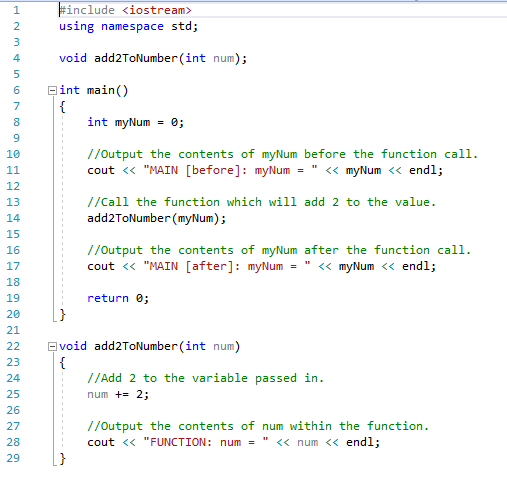
num += 2;

// Output the contents of num within the function.

cout << “FUNCTION: num = “<< num << endl;

}

Program Listing 15





**Program 16: 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:

int smallerNumber ( int a, int b );

Additional thinking:

* How will you deal with numbers of the same value?

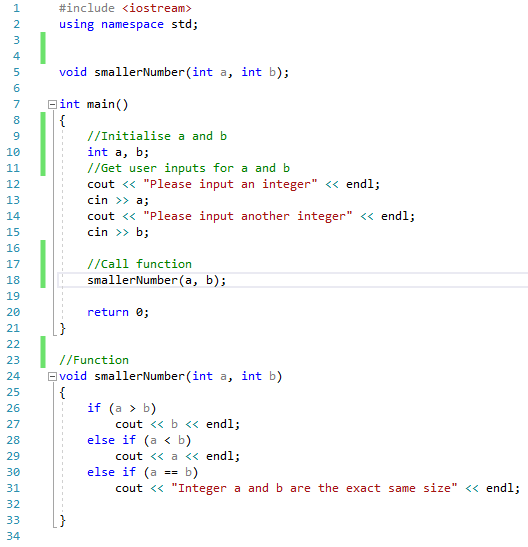
Note: This is a function exercise and the result MUST be determined within the function body and then returned from the function.

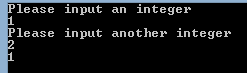
Run your program using the following values. The screenshot must show the use of these elements:

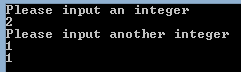
1 2

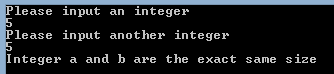
2 1

5 5









P**rogram 17: 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. The 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:

void outputResults ( int numOfOdd, int oddTotal, int numOfEven, int evenTotal );

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

