Chapter 7 – Functions

Functions are used to modularise code or sort it into different blocks or sub-tasks. Functions perform a particular job that the programmer assigns it to do. They make code look neater and more elegant and can be "called" or used as many times as you like. You actually used a function in every program you wrote up to now – remember the main function – int main() ?

As an example, let us write some code to make a virtual cup of coffee.

We could have a function to add the coffee...AddCoffee()

Then a function to add some sugar....AddSugar()

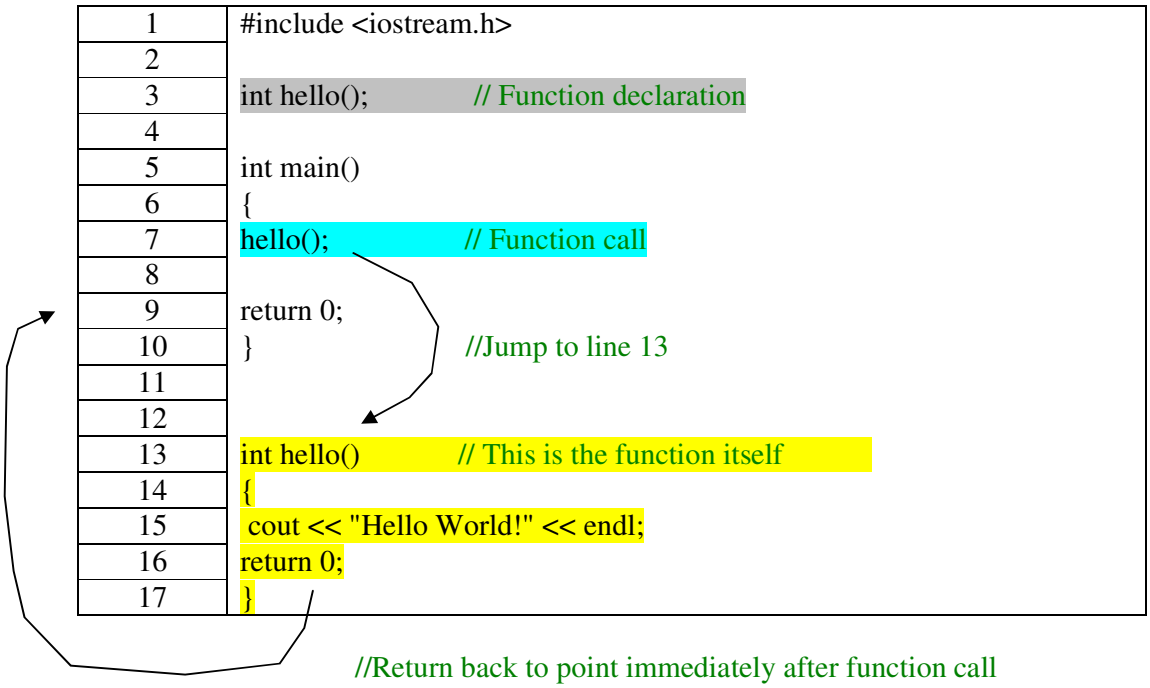
Then a function to add some milk....AddMilk()

Now if we wanted to make a cup of coffee with plenty of sugar we could write;

```
#include<iostream.h>....
.........
.....
...
..
AddCoffee();    // Call the function AddCoffee()

AddSugar();    // Call the function AddSugar()

AddSugar();     // Call the function AddSugar()

AddMilk();      // Call the function AddMilk()
...
..
.}
```

You can see from above that the function AddSugar() was called twice. This is the main advantage of functions. Instead of *writing* code twice to add some sugar to the coffee, we can write a function *once* and then "call" it twice.

Now let's look at a real example. Let's write our Hello World! program again by using functions.



```
1   #include <iostream.h>
2
3   int hello();              // Function declaration
4
5   int main()
6   {
7   hello();                  // Function call
8
9   return 0;
10  }                         //Jump to line 13
11
12
13  int hello()               // This is the function itself
14  {
15   cout << "Hello World!" << endl;
16  return 0;
17  }

                              //Return back to point immediately after function call
```

Example 7a

Line 3 is the function declaration. Just like a variable, a function has to be declared. Note that the function is declared AFTER #include<iostream.h>, but BEFORE int main(). Also note that there is a semi-colon after functions we make, but never after int main().

The int in "int hello();", means that the function hello() returns an integer, here it is 0. Return 0; means when the function completes its job it returns or does nothing.

Later we will see functions declared like this: int DoSomething(int);

This means that the function receives an integer (in the brackets), does something with it, then returns another integer back to main() or from wherever it was called from.

I.e. return Function(receive)

Line 7 is the "function call". When the computer sees this it jumps straight down to line 13 where the function is and does whatever the function tells it. In this case it

prints out "Hello World!". When it is finished the computer jumps back up to line 9 and the main program stops.

You might say “wouldn’t it be easier to just write a simple program to print "Hello World!"”. Well in this case you would be right. But this example is just to explain the concept of a function. Next we will see an example of how functions can be really useful.

Now study and try to understand the next program.

```
1  #include <iostream.h>
2
3  int add(int, int);              // Function declaration
4
5  int main()
6  {
7  int a,b;
8
9  cout<< "Enter Two integers " <<endl;
10                                          //Function call
11 cin>> a >> b;
12
13 cout<< "The sum of "<< a << " and " << b << " is " << add( a , b );
14
15 return 0;
16 }                    a
17
18                                  b
19 int add(int a , int b)
20 {
21 int sum;
22                                  // Replace function call with sum
23 sum=( a + b );
24
25 return sum;
26 }
```

Example 7b

Note: cin>> a >> b; is the same as cin>>a;
cin>>b;

If you were to enter 20 and 30 for a and b above the program would output:

The sum of 20 and 30 is 50

This program is a bit longer and makes better use of functions. The function takes two integers and returns their sum.

You may have noticed that a and b were declared as integers in main() and then declared again in add(). Why did we have to declare them twice? Well the reason is that to the computer they are actually different. When the computer is in main() it cannot see the other a and b. And vice versa when the computer is in add(). I.e. both functions main() and add() have a transient existence, so their variables do not conflict.
We could have easily replaced the a and b in add() with x and y. Try doing this, it will make no difference to the program.

Edit the above program to read in two floats and print out their sum.

Try and add on another function to the above program that will find the difference of two numbers and return both answers to main(). Answer is below, but try it yourself first.

```
1  #include <iostream.h>
2
3  int add(int, int);                  // Function declaration
4  int diff(int, int);                 // Function declaration
5
6  int main()
7  {
8  int a,b;
9
10 cout<< "Enter Two integers " <<endl;
11
12 cin>> a >> b;
13
14 cout<< "The sum of "<< a << " and " << b << " is " << add( a , b );
15
16 cout<< "The difference of " << a << " and " << b << " is " << diff( a , b );
17
18 return 0;
19 }
20
21
22 int add(int a , int b)
23 {
24 return (a+b);
25 }
26
27
28 int diff(int a , int b)
29 {
30 return (a-b);
31 }
```

Example 7c

Write a function to read in a number and return its square and cube.

Here is a nice way of tiding up your code. Cut lines out 22-31 (the two functions) and paste them in notepad. Save the file as “MyFunctions.h” in the “include” folder of the Borland directory (You will see lots of other .h or header files in here too). Now all you have to do to use your add() and diff() functions in your program is add the line #include<MyFunctions.h> under #include<iostream.h>. This is a good way of keeping clutter to a minimum, and when you need to change or update your functions you can just edit them in the header file. Simple!