**WEEKS 7: MODULAR PROGRAMMING WITH FUNCTIONS**

**Objectives**

Following completion of this topic, students should be able to:

* Understand what a function is
* Understand the concept of C++ functions
* Understand the concept of pass by value and pass by reference

A function is a portion of code within a larger program, which performs a specific task and is relatively independent of the remaining code. In C++ there are three components to functions: *prototype*, *definition* and *call*.

**Function Prototype:**

The function prototype is a declaration to the compiler of the function stating its return type and the type of its parameters. It does not contain the code to be executed upon invocation. The function *prototype* is usually placed before main, in the global area. The general syntax for a prototype is:

return-type function-name(parameter-type-list); function-name:

The identifier for the function. This follows the same rules as that of variable/constant identifiers. The name should reflect the main task that the function performs.

return type:

Each function must declare a return type whether or not it actually returns a value. If a function does NOT return a value, then the void type is used. The return type can be any of the built-in (*int, float, char, void, etc*) or user-defined (*to be introduced later*) types. e.g.

void func(); // Returns NO value int func(); // Returns an integer value float func(); // Returns a floating point value char func(); // Returns a character parameter-type-list:

Variables declared within a function are ***local*** to that function and come under the ***scope*** rules. Thus, such variables cannot by default be accessed by code inside another function. However, as functions often need to be supplied with data in order to perform their task, a mechanism needs to be in place for data to be ***passed*** into a function. Data may be passed into a function by way of parameters. There are two ways to pass data.

o Pass-By-Value:

Copies the *value* of the input data to the parameter. The input data is completely independent of the parameter variable. The only link between the two is the initial value of the parameter upon invocation of the function. *Any changes to the parameter within the function do NOT affect the input data.* To declare a parameter as pass-by-value the type of the parameter and an (optional) name are placed between the brackets ().

void func(int byVal);

If more than one parameter is used, then they are separated by a comma ,. e.g.

void func(int byVal, char byVal2); o Pass-By-Reference:

Copies the *address* of the input data to the parameter. Thus, the input data and the parameter both have access to the memory location where the data is located. *Any changes to the parameter within the function DO affect the input data*. When declaring a parameter as pass-by-reference, an ampersand & is placed after the type and before the (optional) name.

void func(int& byParam);

Parameters can be any mix of pass-by-value and pass-by-reference. e.g. void func(int& byParam, float byVal);

Note that the names of variables that appear in a funcvtion prototype are within *prototype scope.* In fact the names are often left out, as the compiler is only seeking the types of the parameters. Thus, the above prototypes could have been written

void func(int); void func(int, char); void func(int&); void func(int&, float);

**Function Definition:**

The function *definition* contains the code, which will be executed when the function is invoked. It appears similar to the prototype (including parameter names) with the addition of the function body. The definition is usually placed placed after main. The syntax is:

return-type function-name(parameter-list)

{

statement(s); return return-value;

} return-value:

Where a function has declared a return type that is not void, the function must, as its last executable statement, return a value of that particular type. e.g.

int func(); // prototype – returns int

return 0; return 1; return -9;

float func(); // prototype – returns float return 0; return -1.0; return 4.5;

char func(); // prototype – returns char return 'a'; return 'Z';

statement(s):

The code to be executed when the function is called.

**Function Call:**

In order to execute the code in the function you need to invoke it. The statement to invoke a function, the function ***call***, has the following syntax:

function-name(argument-list); argument-list

contains zero or more data items to be passed to the function as declared in the *prototype*. The arguments can be variables/constants/literal values. e.g.

void func(int param1); // prototype

int anInt = 5;

const int CONST\_INT = 10; func(anInt); func(CONST\_INT); func(12);

# TASK 0. FUNCTION BASICS

*NOTE: This task is to be completed in your own time* ***before*** *the start of your lab This task is a pen and paper task – I.e. no computer is required.*

1. The function\_\_\_\_\_\_Prototype\_\_\_\_\_ is the declaration to the compiler of the function.
2. The function \_\_\_\_\_\_Call\_\_\_\_\_ is the statement that invokes the function.
3. The function \_\_\_\_\_\_\_\_Stub\_\_\_ contains the body of the function.
4. The \_&\_\_ is placed after the type of a pass-by-reference parameter.
5. A function that does *not* return a value should declare a return type of \_\_void\_\_\_\_.
6. List some ways in which the value returned from a function may be used:

Transfer data from 1 function to the next

Return true if the function executed properly

Return true if a check the function is performing finds the data it was looking for

Tell the compiler that compilation was successful

# TASK 1. UNDERSTANDING VOID FUNCTIONS

1. Type the following in a program named *’task1.cpp’*. Fill in the blanks (*do not include the commenting*) then compile the program.

\_\_\_\_\_\_\_\_\_\_\_\_\_ // add include files

\_\_\_\_\_\_\_\_\_\_\_\_\_ /\* Add the function prototype for a function named

printStars which does not return a value and takes no parameters.\*/

int main()

{ return 0;

}

1. Add the following code, *including* the function *call* for the function printStars to main then compile the program.

cout << "Before function call.\n" << endl;

// add function call here

cout << "\nAfter function call." << endl;

1. You will receive an error similar to:

Undefined first referenced symbol in file printStars() /var/tmp//ccMOCDoe.o ld: fatal: Symbol referencing errors. No output written to a.out collect2: ld returned 1 exit status

This error indicates that the function printStars has not been defined. That is, the *definition* has not been written.

1. Add the function *definition*, leaving the *body* blank for now, and compile the program.
2. Add the following code to the function *body* then compile and run the program.

int stars;

cout << "Inside printStars.\n" << endl; cout << "How many stars to print?: "; cin >> stars;

for (int i = 0; i < stars; i++) cout << '\*'; cout << endl;

1. Close the file.

# TASK 2. UNDERSTANDING FUNCTIONS THAT RETURN A VALUE

a. Type the following in a program named *‘task2.cpp’* then compile it.

\_\_\_\_\_\_\_\_\_\_ // add include files

\_\_\_\_\_\_\_\_\_\_ // Add prototype for function named getAge that takes

// no parameters and returns an int

int main( ) { int age;

cout << "Enter your age: "; age = getAge(); if (age > 0) cout << "\nAre you really " << age

<< " years old?\n" << endl; else

cout << "You entered an invalid age!" << endl; return 0;

}

\_\_\_\_\_\_\_\_\_\_ // add function definition leaving the body blank

b. Notice the statement: age = getAge();

This statement assigns the value returned from getAge to the variable age. c. You should receive a warning similar to:

task2.cpp:15: warning: control reaches end of non-void function

This warning indicates that there is no return statement in a function that returns a value. d. Add the following code to the function definition.

int age;

cin >> age; return age;

1. Notice the statement:

return age;

The value of the variable age is returned to main.

1. Compile and run the program . What is the output when the input is (i) -1, (ii) your age ?
2. Modify the code in main to use the return value from getAge in an if statement.

cout << "Enter your age: ";  if (getAge () > 0)

cout << "\nAre you really THAT old?” << endl; else

cout << “You entered an invalid age!” << endl; return 0;

1. Modify the code in main to use the return value from getAge in a cout statement.

cout << "Enter your age: ";

cout << "Are you really " << getAge () << " years old "

<< "or did you make a mistake?" << endl; return 0;

1. Modify the code in both main and getAge so that the function now includes the prompt, and inputs less than 0 and greater than 120 generate an error message (use cerr to report the message). Place the request and read in a suitable loop that repeats until an acceptable age is entered.

# TASK 3. UNDERSTANDING PASS-BY-VALUE PARAMETERS

a. Type the following in a program named *‘task3.cpp’* then compile and run it.

\_\_\_\_\_\_\_\_\_\_ // include files

\_\_\_\_\_\_\_\_\_\_ // Function prototype for a function named passByVal that // takes one integer parameter and returns no value

int main ()

{ int x = 10; passByVal(x);

cout << "After calling passByVal\nx = "

<< x << endl;

return 0;

} void passByVal(int x)

{

cout << "In passByVal\nx = " << x << endl;

}

In function calls, the input data is referred to as arguments. Thus, x in passByVal(x) is an argument.

The parameter variable x in passByVal is passed by *value*.

1. Add the following code to passByVal before the cout statement. x++;
2. Compile and run the program.

Notice that the value of x in main remains 10, while the value of x in passByVal is 11. This is because the two variables are NOT related. They are distinct entities. Thus, any change to the parameter variable x in passByVal does NOT affect the variable x in main. Remember, the only connection between the two variables is the initial value of x in passbyVal. Thus, x in passByVal may have a completely different name.

# TASK 4. PROGRAMMING TASK – RETURN VALUES

Write a program *‘task4.cpp’* that has a function that returns the smaller value of two floats passed to it. The program should produce the following output: Text in **bold** is user input.

Do not assign the value returned from the function to a variable, rather use it directly in the cout statement.

Enter value 1: **10.6**

Enter value 2: **-67.8**

The lowest value is -67.8.

Redesign your solution *‘task4\_v2.cpp’* to make your function a void function, make other changes sensibly, the calling program should still be able to make the same output as before.

# TASK 5. UNDERSTANDING PASS-BY-REFERENCE PARAMETERS

1. Add the following function definition to the program from task 3 and save it as *‘task5.cpp’.*

void passByRef(int& x)

{

x++;

cout << "In passByRef\nx = " << x << endl; }

The ampersand & after int and before x indicates that x is a reference parameter. That is, the location of the argument in main is used as the location of the parameter x in passByRef.

1. Add the corresponding function prototype then compile the program.
2. Add the following code to main after the current code then compile and run it.

passByRef(x);

cout << "After calling passByRef\nx = " << x << endl;

This time you will notice that the value of x has changed after the call to passByRef. This is because x in passByRef and x in main use the same memory location. Thus, any changes to x in passByRef WILL affect x in main. The name of the parameter variable in passByRef may still be different to the variable in main.

***Note: The names of the parameters, both pass-by-value and pass-by-reference can appear in the prototype and can actually be different from those in the definition.***

# TASK 6. PROGRAMMING TASK – PASS-BY-REFERENCE

1. Write a program that has a function that swaps the value of two integer variables. The task of the function is only to swap the values passed to it from main. The program should produce the following output: Text in **bold** is user input.

Enter value 1: **10**

Enter value 2: **55**

Before the swap:

Value 1: 10

Value 2: 55

After the swap:

Value 1: 55

Value 2: 10

1. Compile and run the program.
2. Modify the program so that the types swapped are chars.
3. Compile and run the program.
4. Close the file.

**TASK 7. PUTTING IT ALL TOGETHER NOW….**

Write a program that gets from user input the total number of cups, which is then passed to a function, which will convert this value to gallons, quarts and pints.

* Use reference parameters to pass the values for gallons, quarts and pints back to the main function.
* The return value of the function should be of type bool.
* The returned value will indicate if the value in totCups was valid **(**i.e. **>= 0).**
* If the function returns true, output the results in a tabular format, otherwise output a message indicating that an error has occurred.
* Use relationships of 2 cups to a pint, 4 cups to a quart and 16 cups to gallon.
* Use the function *call* as an if statement *condition*.
* Test your program with valid data. I.e. negative, positive, zero.