



PROGRAMMING FUNDAMENTALS LAB

Lab Instructors: Nida Munawar & Romasha Khurshid

Course Code: CL118
Credit Hour: 1
Program Section: SE 19

Function:

A function is a set of statements that take inputs, do some specific computation and produces output.

The idea is to put some commonly or repeatedly done task together and make a function so that instead of writing the same code again and again for different inputs, we can call the function.

USES OF C FUNCTIONS:

- The core concept of C functions is, re-usability, dividing a big task into small pieces to achieve the functionality and to improve understandability of very large C programs.
- Functions help us in reducing code redundancy. If functionality is performed at multiple places in software, then rather than writing the same code, again and again, we create a function and call it everywhere.
- This also helps in maintenance as we have to change at one place if we make future changes to the functionality.

C FUNCTION DECLARATION, FUNCTION CALL AND FUNCTION DEFINITION:

There are 3 aspects in each C function. They are,

1. **Function declaration or prototype:**

A function prototype is simply the declaration of a function that specifies function's name, parameters and return type. It doesn't contain function body. A function prototype gives information to the compiler that the function may later be used in the program.

Syntax of function prototype:

returnType functionName(type1 argument1, type2 argument2,...);

In the below example,

int addNumbers(int a, int b); is the function prototype which provides following information to the compiler:

1. name of the function is addNumbers()
2. return type of the function is int
3. two arguments of type int are passed to the function

The function prototype is not needed if the user-defined function is defined before the main () function.

2. **Function call:**

Control of the program is transferred to the user-defined function by calling it.

Syntax of function call:

functionName(argument1, argument2, ...);

In the above example, function call is made using addNumbers(n1,n2); statement inside the main().

3. **Function definition:**

- Function definition contains the block of code to perform a specific task i.e. in this case, adding two numbers and returning it.
- When a function is called, the control of the program is transferred to the function definition.
- And, the compiler starts executing the codes inside the body of a function.

C functions aspects	syntax
function definition	Return_type function_name (arguments list) { Body of function; }
function call	function_name (arguments list);
function declaration	return_type function_name (argument list);

SIMPLE EXAMPLE PROGRAM FOR C FUNCTION:

```

1  #include<stdio.h>
2  // function prototype, also called function declaration
3  float square ( float x );
4  // main function, program starts from here
5
6  int main( )
7  {
8      float m, n ;
9      printf ( "\nEnter some number for finding square \n");
10     scanf ( "%f", &m ) ;
11     // function call
12     n = square ( m ) ;
13     printf ( "\nSquare of the given number %f is %f",m,n );
14 }
15
16 float square ( float x ) // function definition
17 {
18     float p ;
19     p = x * x ;
20     return ( p ) ;
21 }

```

If you have grasped the concept of ‘calling’ a function you are prepared for a call to more than one function.

Consider the following example:

```
main( )
{ printf ( "\nI am in main" ) ;
  italy( ) ;
  brazil( ) ;
  argentina( ) ; }
italy( )
{
  printf ( "\nI am in italy" ) ;
}
brazil( )
{
  printf ( "\nI am in brazil" ) ;
}
argentina( )
{
  printf ( "\nI am in argentina" ) ;
}
```

The output of the above program when executed would be as under:

I am in main

I am in italy

I am in brazil

I am in argentina

From this program a number of conclusions can be drawn:

- Any C program contains at least one function.
- If a program contains only one function, it must be main().
- If a C program contains more than one function, then one (and only one) of these functions must be main(), because program execution always begins with main().
- There is no limit on the number of functions that might be present in a C program.
- Each function in a program is called in the sequence specified by the function calls in main()
- After each function has done its thing, control returns to main (). When main () runs out of function calls, the program ends.

Let us now summarize what we have learnt so far:

- a) C program is a collection of one or more functions.
- b) A function is defined when function name is followed by a pair of braces in which one or more statements may be present.

For example,

```
argentina( )  
{  
statement 1;  
statement 2;  
statement 3;  
}
```

- c) Any function can be called from any other function. Even main() can be called from other functions.

For example,

```
main( )  
{  
message( ) ;
```

```

}
message( )
{
printf ( "\nCan't imagine life without C" ) ;
main( ) ;
}

```

d) A function can be called any number of times. For example,

```

main( )
{
message( ) ;
message( ) ;
}
message( )
{
printf ( "\nJewel Thief!!" ) ;
}

```

e) The order in which the functions are defined in a program and the order in which they get called need not necessarily be same. For example,

```

main( )
{
message1( ) ;
message2( ) ;
}
message2 ( )
{
printf ( "\nBut the butter was bitter" ) ;
}

```

```
message1(  
{  
printf ( "\nMary bought some butter" ) ;  
}
```

it is advisable to define the functions in the same order in which they are called. This makes the program easier to understand.

- f) A function can call itself. Such a process is called ‘recursion’
- g) A function can be called from other function, but a function cannot be defined in another function.

Passing arguments to a function:

In programming, argument refers to the variable passed to the function. In the above example, two variables **n1 and n2** are passed during function call.

The parameters **a and b** accepts the passed arguments in the function definition. These arguments are called formal parameters of the function.

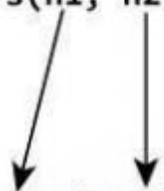
How to pass arguments to a function?

```
#include <stdio.h>

int addNumbers(int a, int b);

int main()
{
    ... ..
    sum = addNumbers(n1, n2);
    ... ..
}

int addNumbers(int a, int b)
{
    ... ..
    ... ..
}
```

Two arrows originate from the arguments 'n1' and 'n2' in the function call 'addNumbers(n1, n2);' within the 'main' function. The arrow from 'n1' points to the parameter 'a' in the function definition 'int addNumbers(int a, int b)'. The arrow from 'n2' points to the parameter 'b' in the function definition.

The type of arguments passed to a function and the formal parameters must match, otherwise the compiler throws error.

If **n1** is of char type, **a** also should be of char type. If **n2** is of float type, variable **b** also should be of float type. A function can also be called without passing an argument.

Return Statement

The return statement terminates the execution of a function and returns a value to the calling function. The program control is transferred to the calling function after return statement.

In the above example, the value of variable result is returned to the variable sum in the main () function.

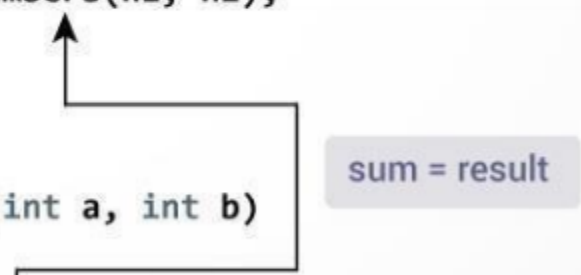
Return statement of a Function

```
#include <stdio.h>

int addNumbers(int a, int b);

int main()
{
    ... ..
    sum = addNumbers(n1, n2);
    ... ..
}

int addNumbers(int a, int b)
{
    ... ..
    return result;
}
```



sum = result

Syntax of return statement

return (expression);

For example,

return a;

return (a+b);

The type of value returned from the function and the return type specified in function prototype and function definition must match.

There are basically two types of functions:

1. Standard library functions
2. User-defined functions

Library functions:

Ex. printf(), scanf() etc.

User-defined functions:

Ex. argentina(), brazil() etc. As the name suggests, library functions are nothing but commonly required functions grouped together and stored in what is called a Library. This library of functions is present on the disk and is written for us by people who write compilers for us. Almost always a compiler comes with a library of standard functions. The procedure of calling both types of functions is exactly same.

Categories of User-Defined function:

Based on prototype, user defined function are further categorized in four categories.

1. Function with no return and no argument
2. Function with no return but arguments
3. Function with return but no argument
4. Function with return and arguments

Syntax to define function with no return no argument

```
void function_name()  
{  
    // Function body  
}
```

Syntax to define function with no return but with arguments

```
void function_name(type arg1, type arg2, ...)  
{  
    // Function body  
}
```

Syntax to define function with return but no arguments

```
return_type function_name()  
{  
    // Function body  
  
    return some_value;  
}
```

Syntax to define function with return and arguments

```
return_type function_name(type arg1, type arg2, ...)  
{  
    // Function body  
  
    return some_variable;  
}
```

Which is the best function type for my program?

Choosing a correct function type completely depends on the programming need.

- Consider function with return and arguments. If your function requires input from other functions and returns the processed result back to the caller function. The returned value might be input for other function.

For example - `double sqrt(double x)`, `double pow(double x, double y)` etc.

- Consider function with return but no argument. If your function does not require any input from other function. Rather, it just produce some output that may be input to other functions.

For example - `int rand(void)`, `int getchar(void)` etc.

- Consider function with no return but arguments. If your function is dependent on the input of some other function. But, does not generates any output that could be consumed by the caller function.

For example - `void exit(int status)` etc.

- Consider function with no return and no argument. If your function works as an independent block. It does not communicate with other functions.

For example - `void abort(void)`

User Defined Header File:

Follow the step to create your header file:

Suppose we want to make header for sum function.

1. Make a header file with .h extension and give it unique name

e.g `sumfile-> sumfile.h`

2. Define your program in header extension file.

a. `int add(int a,int b){return(a+b);}`

3. Make source file of c, where your main program is set.

a. `#include`

b. `#include "sumfile.h" // don't use '<>', instead of it use'''`.

c. `void main()`

d. `{ int num1 = 10, num2 = 10, num3;`

e. `num3 = add(num1, num2);`

f. `printf("Addition of Two numbers : %d", num3);}`

4. Keep .h file path directory same as source file.

5. In the above program the 'add' function is basically called from the header file of sumfile.h Which we have explicitly defined.

Lab Exercise

1. Amanda who works for a retail outlet that also provides online buying service to its customers. The order of three dresses has been placed from a certain customer (you can assume buying cost by your own) there is a special 10% discount on every purchasing of more than two dresses after order confirmation the system generates a confirmation message to its customer but after 2 days the same customer place order for two more dresses and he wants to deliver his both orders together now write a c

program to translate the above scenario and make order billing function where necessary.

2. Bilal is a student of computer science. He is currently self-studying programming in C language. He wants to develop a program for the purpose of reusability, in which he is performing mathematical operations like addition, subtraction, production, and division. Keep in mind, He wants to perform these mathematical operations in different programs since he may need to use any of these operations in any of the developed programs
3. Write a C-program that converts metric measurements to imperial system values. Measurements are provided to your program in meters, grams or degrees Celsius and must be converted to feet, pounds and degrees Fahrenheit, respectively.

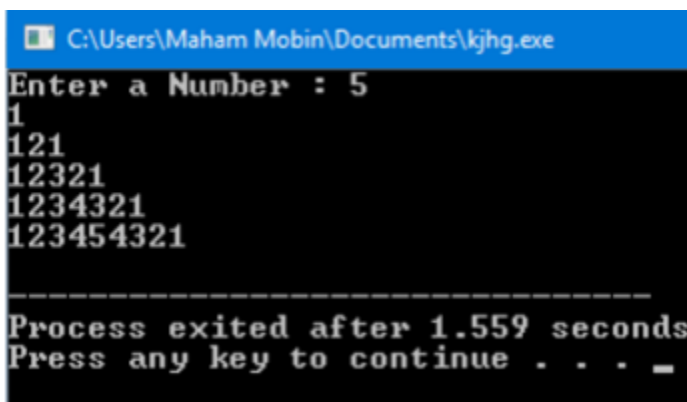
Here are the conversion rules to use:

1 meter = 3.2808 feet;

1 gram = 0.002205 pounds;

temperature in degrees Fahrenheit = $32 + 1.8 \times$ temperature in degrees Celsius.

4. Generate the following patterns:



```
C:\Users\Maham Mobin\Documents\kjhg.exe
Enter a Number : 5
1
121
12321
1234321
123454321
-----
Process exited after 1.559 seconds
Press any key to continue . . . _
```

5. Write a program in C to show the simple structure of a function.

6. Write a program in C to find the square of any number using the function
7. Write a program in C to swap two numbers using function.
8. Write a program in C to check a given number is even or odd using the function.
9. Write a program in C to check whether a number is a prime number or not using the function.