IT 105: Introduction to Programming

Functions

Dr. Manish Khare
Dr. Saurabh Tiwari

Lecture 13

What is a Function?

```
#include <stdio.h>
message(); /* function prototype definition*/
main()
{
        message(); /* function call */
        printf("Hello Main!");
}
message() /* function definition */
{
        printf("Smile! I am here\n");
}
```

Calling function

Called function

Functions: Facts

- A function gets called when the function name is followed by a semicolon. For example,

```
main( )
{
        argentina( ) ;
}
```

Functions: Facts

- 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 ;
}
```

Functions: Facts

- 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( ) ;
}
```

Functions: Facts

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

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

Functions: Facts

- 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" ) ;
}
```

Functions: Facts

- A function can be called from other function, but a function cannot be defined in another function. Thus, the following program code would be wrong, since **argentina( )** is being defined inside another function, **main( ).**

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

Functions: Facts

There are basically two types of functions:

- Library functions Ex. **printf( ), scanf( ) etc.**
- User-defined functions Ex. **argentina( ), brazil( ) etc.**

Passing Values between functions

There are basically two types of functions:

- Library functions Ex. **printf( ), scanf( ) etc.**
- User-defined functions Ex. **argentina( ), brazil( ) etc.**

Passing Values between functions

- Consider the following program. In **main( )** we receive the values of **a, b and c** through the keyboard and then output the **sum of a, b and c**.
- However, the calculation of sum is done in a different function called **calsum( ).**
- If sum is to be calculated in **calsum( )** and values of a, b and c are received in **main( ),** then we must pass on these values to **calsum( ),** and once **calsum( )** calculates the sum we must return it from **calsum()** back to **main( ).**

https://ideone.com/vx0GVR

Points to Remember

- The variables a, b and c are called ‘actual arguments’, whereas the variables x, y and z are called ‘formal arguments’.
- Any number of arguments can be passed to a function being called.
  - However, the type, order and number of the actual and formal arguments must always be same.
- Instead of using different variable names x, y and z, we could have used the same variable names a, b and c. But the compiler would still treat them as different variables since they are in different functions.

Points to Remember

- There are two methods of declaring the formal arguments.

```
calsum ( x, y, z )
int x, y, z ;
```

Another method is,

```
calsum ( int x, int y, int z )
```

- There is no restriction on the number of return statements that may be present in a function. Also, the return statement need not always be present at the end of the called function.

Points to Remember

```
fun( )
{
        char ch ;
        printf ( "\nEnter any alphabet " ) ;
        scanf ( "%c", &ch ) ;
        if ( ch >= 65 && ch <= 90 )
                return ( ch ) ;
        else
                return ( ch + 32 ) ;
}
```

Points to Remember

- All the following are valid return statements.

```
return ( a ) ;
return ( 23 ) ;
return ( 12.34 ) ;
return ;
```

- If we want that a called function should not return any value, in that case, we must mention so by using the keyword **void** as shown below.

```
void display( )
{
        printf ( "\nHeads I win..." ) ;
        printf ( "\nTails you lose" ) ;
}
```

Points to Remember

- A function can return only one value at a time. Thus, the following statements are invalid.

```
return ( a, b ) ;
return ( x, 12 ) ;
```

- If the value of a formal argument is changed in the **called function**, the corresponding change does not take place in the **calling function.**

https://ideone.com/7cLARI

Points to Remember

```
main()
{
        int i = 20 ;
        display ( i ) ;
}
                        display ( int j )
                        {
                                int i;
                                int k = 35 ;
                                printf ( "\n%d", j ) ;
                                printf ( "\n%d", i ) ;
                                printf ( "\n%d", k ) ;
                        }
```

Calling Convention

- Order in which arguments are passed to a function when a function call is encountered
  - Arguments might be passed from left to right.
  - ***Arguments might be passed from right to left.***

- Consider the following function call:

fun (a, b, c, d) ;

  - In this call it doesn't matter whether the arguments are passed from left to right or from right to left.

Calling Convention

```
int a = 1 ;
printf ( "%d %d %d", a, ++a, a++ ) ;
```

- It appears that this printf( ) would output 1 2 3.
- Surprisingly, it outputs 3 3 1
- C's calling convention is from right to left

Function Declaration and Prototypes

- Any C function by default returns an **int** value.
- Whenever a call is made to a function, the compiler assumes that this function would return a value of the type **int.**
- If we desire that a function should return a value other than an **int**, then it is necessary to explicitly mention so in the calling function as well as in the called function.

Function Declaration and Prototypes-Eg.

- Suppose we want to find out square of a number using a function.

https://ideone.com/5wT8HR

Function Declaration and Prototypes-Eg.

- In practice you may seldom be required to return a value other than an **int**, you have to explicitly mention it.
- In some programming situations we want that a called function should not return any value. This is made possible by using the keyword **void**.

```
main( )
{
void gospel( ) ;
gospel( ) ;
}
```

```
void gospel( )
{
printf ( "\nViruses are electronic bandits..." ) ;
printf ( "\nwho eat nuggets of information..." ) ;
printf ( "\nand chunks of bytes..." ) ;
printf ( "\nwhen you least expect..." ) ;
}
```

Why functions?

- Avoid writing the same code over and over.
- Structured programming
- **Code reusability** is a method of writing code once and using it many times. Using structured programming technique, we write the code once and use it many times.
- In C, the structured programming can be designed using **functions** concept. Using functions concept, we can divide larger program into smaller subprograms and these subprograms are implemented individually.
- Every subprogram or function in C is executed individually.

Advantages of Functions

- Using functions we can implement modular programming.
- Functions makes the program more readable and understandable.
- Using functions the program implementation becomes easy.
- Once a function is created it can be used many times (**code re-usability**).
- Using functions larger program can be divided into smaller modules.

Exercise?

```
#include<stdio.h>
float circle (int);
main( )
{
        float area ;
        int radius = 1 ;
        area = circle ( radius ) ;
        printf ( "\n%f", area ) ;
}
```

```
float circle ( int r )
{
        float a ;
        a = 3.14 * r * r ;
        return ( a ) ;
}
```

Exercise?

```
#include<stdio.h>
int check(int);
main( )
{
        int i = 45, c ;
        c = check ( i ) ;
        printf ( "\n%d", c ) ;
}
```

```
int check ( int ch )
{
        if ( ch >= 45 )
                return ( 100 ) ;
        else
                return ( 10 * 10 ) ;
}
```

Identify Errors?

```
#include<stdio.h>
int addmult(int, int)
main( )
{
        int i = 3, j = 4, k, l ;
        k = addmult ( i, j ) ;
        l = addmult ( i, j ) ;
        printf ( "\n%d %d", k, l ) ;
}
```

```
int addmult ( int ii, int jj )
{
        int kk, ll ;
        kk = ii + jj ;
        ll = ii * jj ;
        return ( kk, ll ) ;
}
```

Identify Errors?

```
#include<stdio.h>
void message();
main( )
{
        int a ;
        a = message( ) ;
}
```

```
void message ( )
{
        printf ( "\nViruses are written in C" ) ;
        return ;
}
```

Identify Errors?

```
#include<stdio.h>
void printit (float, char);
main( )
{
        float a = 15.5 ;
        char ch = 'C' ;
        printit ( a, ch ) ;
}
```

```
void printit ( a, ch )
{
        printf ( "\n%f %c", a, ch ) ;
}
```

Identify Errors?

```
#include<stdio.h>
main( )
{
        let_us_c( )
        {
                printf ( "\nC is a Cimple minded language !" ) ;
                printf ( "\nOthers are of course no match !" ) ;
        }
}
```

Identify Errors?

```
#include<stdio.h>
void message();
main( )
{
        message( message ( ) ) ;
}
```

```
void message( )
{
printf ( "\nPraise worthy and C worthy are synonyms" ) ;
}
```

Program?

- Write a function power(a, b) to calculate the value of a raised to b.

https://ideone.com/84Lr9K

- A positive integer is entered through the keyboard. Write a function to obtain prime factors of this number.
  - e.g., prime factors of 24 are 2, 2, 2 and 3. Prime factors of 35 are 5 and 7