LECTURE 7

INTRODUCTION TO FUNCTIONS

CALL BY VALUE

INTRODUCTION TO FUNCTIONS

- A function is a block of code that performs a specific task.
- There are two types of function in C programming:
  - Standard library functions
  - User defined functions
- We have already known that in a C program, there must be a main() function

INTRODUCTION TO FUNCTIONS

- In C, we used many Standard library functions as well- printf(), scanf(), clrscr(), or getch()
- printf() prints on the output
- scanf() takes input from the user and assigns that input to a variable by going to the variable's address
- clrscr() clears the output buffer
- getch() waits for a key-stroke from the user

INTRODUCTION TO FUNCTIONS

- In C, user can define functions as well- as many functions as they wish. These functions are called user defined functions.
- There are three parts:
  - Function declaration
  - Function definition
  - Function call

FUNCTION DECLARATION

- *return_type function(parameter_list); /*function declaration*/*
- Function declaration is also known as function prototype.
- It inform the compiler about three thing, those are name of the function, number and type of argument received by the function and the type of value returned by the function.
- function prototype always terminated by the semicolon

FUNCTION DEFINITION

```
return type function(type 1 arg1, type2 arg2, type3 arg3)    /*function header*/
{
Local variable declaration;
Statement 1;
Statement 2;
Return value
}
```

INTRODUCTION TO FUNCTIONS

- Function definition consists of the whole description and code of the function.
- It consists of two parts function header and function body
- The arguments of the function definition are known as formal arguments.

INTRODUCTION TO FUNCTIONS

- **Function call**

  function(arg1, arg2, arg3)

- when the function get called by the calling function then that is called, function call.
- The compiler execute these functions when the semicolon is followed by the function name.
- The arguments of the function definition are known as formal arguments.

EXAMPLE OF FUNCTION

```
#include<stdio.h>
void message();
void main() {
    message();
    printf("This is inside main function");
}
void message() {
    printf("This is inside message function\n");
}
```

ANOTHER EXAMPLE OF FUNCTION

```
#include<stdio.h>


void italy();
void brazil();
void argentina();


void main() {
      printf("I am in main function\n");
    italy();
    brazil();
    argentina();
    printf("I came back in main function");
}
```

```
void italy() {
    printf("Italy: 1934 1938 1982 2006\n");
}

void brazil() {
    printf("Brazil: 1958 1962 1970 1994 2002\n");
}

void argentina() {
    printf("Argentina: 1978 1986 2022\n");
}
```

DIFFERENCE BETWEEN THE FORMAL ARGUMENT AND THE ACTUAL ARGUMENT

- The basic difference between the formal argument and the actual argument are:

1) The formal argument are declared inside the parenthesis where as the local variable declared at the beginning of the function block.

2) The formal argument are automatically initialized when the copy of actual arguments are passed while other local variable are assigned values through the statements.

- Order number and type of actual arguments in the function call should be match with the order number and type of the formal arguments.

EXAMPLE:

```
#include<stdio.h>
int sum(int, int);
int main()
{
    int a, b;
    printf("enter two no");
    scanf("%d%d",&a,&b);
    int s=sum(a,b);
     printf("summation is = %d", s);
}

int sum(int x1,int y1)
 {
       int z=x1+y1;
       return z;
}
```

WHAT WE HAVE LEARNT?

- 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.
- After each function has done its job, control returns to the place of calling that function.

USER DEFINED FUNCTIONS

- Well, then, should all the function calls take place inside the main () function? No! Functions defined by the user can call other functions as well.

ANOTHER EXAMPLE OF FUNCTION

```
#include<stdio.h>


void italy();
void brazil();
void argentina();


void main() {
      printf("I am in main function\n");
    italy();
    brazil();
    argentina();
    printf("I came back in main function");
}
```

```
void italy() {
    printf("Italy: 1934 1938 1982 2006\n");
}

void brazil() {
    printf("Brazil: 1958 1962 1970 1994 2002\n");
}

void argentina() {
    printf("Argentina: 1978 1986 2022\n");
}
```

WE HAVE KNOWN A LOT!!

- C program is a collection of one or more functions.
- A function gets called when the function name is followed by a semicolon.

WE HAVE KNOWN A LOT!!

- A function is defined when function name is followed by a pair of braces in which one or more statements may be present.

```
void hala_madrid()
{
    statement 1;
    statement 2;
    statement 3;
}
```

WE HAVE KNOWN A LOT!!

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

```
#include<stdio.h>

void message();

void main() {
    message();
}

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

WE HAVE KNOWN A LOT!!

- A function can be called any number of times.

```
#include <stdio.h>

void message();

int main() {
    message();
    message();
    return 0;
}

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

WE HAVE KNOWN A LOT!!

- The order in which the functions are defined in a program and the order in which they get called need not necessarily be same

```
#include <stdio.h>

void message1();
void message2();

int main() {
    message1();
    message2();
    return 0;
}

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

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

WE HAVE KNOWN A LOT!!

- 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 **madrid( )** is being defined inside another function, **main( )**.

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

void madrid( )
 {
printf ( "\nI am in madrid" ) ;
  }
}
```

WE HAVE KNOWN A LOT!!

- There are basically two types of functions:
- Library functions Ex. **printf( )**, **scanf( )** etc.
- User-defined functions Ex. **argentina( )**, **brazil( )** etc.

```c
#include <stdio.h>

void sum();
void sub();
void mul();
void div();
void mod();

int main() {
    sum();
    sub();
    printf("\nPress enter to exit.");
    getchar(); // Wait for input before closing
    return 0;
}
```

```c
void sum() {
    int a, b;
    printf("Enter 2 integers: ");
    scanf("%d %d", &a, &b);
    printf("Summation: %d\n", a + b);
}

void sub() {
    int a, b;
    printf("Enter 2 integers: ");
    scanf("%d %d", &a, &b);
    printf("Subtraction: %d\n", a - b);
}
```

FUNCTION DEFINITION VS DECLARATION

- Function Declaration or function prototyping
- Function Definition or function body
- The basic format of function definition is-

```
return-type function-name (data_type parameter_1, data_type
                           parameter_2,…, data_type parameter_n){
      declarations;
      statements;
}
```

CALL BY VALUE

- Sending the values of the arguments.
- The ‘value’ of each of the actual arguments in the calling function is copied into corresponding formal arguments of the called function.
- With this method the changes made to the formal arguments in the called function have no effect on the values of actual arguments in the calling function.

CALL BY VALUE

```
#include <stdio.h>

int maximum(int, int, int); // function prototype

int main() {
    int x, y, z;
    printf("Enter three integers: ");
    scanf("%d %d %d", &x, &y, &z);
    printf("The maximum is %d\n", maximum(x, y, z));
    return 0; // Properly terminate main with a return statement
}

// function definition
int maximum(int a, int b, int c) {
    int max = a;
    if(b > max)
        max = b;
    if(c > max)
        max = c;
    return max;
}
```

10,20,30

maximum(10,20,30)

maximum(10,20,30)

CALL BY VALUE

```
#include <stdio.h>

int square(int); // function prototype

int main() {
    int x, y;
    for (x = 1; x <= 10; x++) {
        y = square(x); // we called square with the current value of x
                       // the square gets it into z and returns z*z
                       // the value is assigned into y
        printf("The square of %d is %d \n", x, y);
    }
    return 0; // Properly terminate main with a return statement
}

// function definition
int square(int z) {
    return z * z;
}
```