

LECTURE NOTES BY Bibek Raj Joshi (Functions)

1. Introduction
2. Returning a value from a function
3. Sending a value to a function
4. Arguments
5. External variables
6. Preprocessor directives
7. C libraries
8. Macros
9. Header files and prototyping

1. Function

A function is a collection of related statements that is used to perform a specific task. The specific task is repeated each time when the function is called. So, this avoids the need of rewriting the same code again and again. Every program must contain one function named `main()` from where the program always begins execution.

Complex problems can be solved by breaking them into a set of sub-problems, called Modules or Functions. This technique is called divide and conquer. Each module can be implemented independently and later can be combined into a single unit.

2. Advantages of Function

- **Avoids redundant code:** The repeated instructions can be written as a function, which can then be called whenever it is needed. So there is no chance of code duplication.
- **Code reusability:** The functions created in one program can be accessed in other programs.
- C functions can be used to build a customized library of frequently used routines.
- Program can be developed in short period of time using function.
- It can be used in the minimization of code.

3. Types of Function

C functions are classified into two categories. They are:

3.1 Library functions (c libraries)

The function that does not require the definition to be written by user is called library function. They are also known as c libraries. They are created by the developer. Their function and declaration are predefined. Example: `printf()`, `scanf()`, `gets()`, `puts()`, `strrev()`, `getch()`, `clrscr()`, etc.

3.2 User defined functions

The function developed by the user at the time of writing the program is called user defined function. The user defines the function code according to their requirement. Example: `sum()` to calculate the sum of given values.

Difference Between Library Function And User-Defined Function

S.N	Library Function	User-Defined Function
1	These are predefined and compiled functions that are placed in C-library.	These are defined by the user at the time of writing a program as per the own requirement.
2	The function's name, return type, arguments and their types cannot be changed.	The function's name, return type, arguments and their types can be changed.
3	It is difficult to understand.	It is easier to understand.
4	These are the part of header files which are called during runtime.	These are the part of the program which are compiled during runtime.
5	Example: <code>printf()</code> , <code>scanf()</code> , <code>clrscr()</code> , etc.	Example: <code>sum()</code> , <code>reverse()</code> , etc.

LECTURE NOTES BY Bibek Raj Joshi (Functions)

4. Parts of Function

A function consists of three parts namely,

- Function Prototype or Declaration
- Function call
- Function Definition

4.1 Function Prototype or Declaration

The prototype of a function declares the function in the same way as we declare the variables. To invoke any function, it must be declared first. Function declaration or prototype tells the program about the type of value returned by the function and the number and type of arguments. Function prototyping means writing the function prototypes before their function call occurs. The prototyping is mandatory in C. The advantage of function prototyping is that it enables a compiler to compare each use of function with the prototype to determine whether the function is invoked properly or not. The number and types of arguments can be easily compared and any wrong number or type of the argument is reported. It consists of four parts:

- Function return type
- Function name
- Parameter list
- Terminating semicolon

Syntax:

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

4.2 Defining a function (Function Definition)

A function declaration tells the compiler about a function's name, return type, and parameters. A function definition provides the actual body of the function. The function definition consists of function header and function body.

Syntax:

```
return_type function_name(parameter list )  
{  
body of the function  
}
```

The return_type, function_name and parameter –list represent the function header and the function body contains local variable declaration, function statements and a return statement. The parts of a function are:

- ❖ **Return Type(returning a value from a function):** The return-type is the data type of the value the function returns. A function may return a value or may perform the desired operations without returning a value. If the function does not return any value, then the return-type is represented by keyword **void**. The return statement returns the value to the calling function. The return statement without any value returns the garbage value to the calling function. There is no restriction on the number of return statements that may be present in the function.
- ❖ **Function Name:** This is the actual name of the function. The function name and the parameter list together constitute the function signature.
- ❖ **Parameters (or argument or sending a value to a function):** A parameter is like a placeholder. It is used to convey information to a function. Parameters are optional i.e. a function may or may not contain any parameters. The parameters are of two types: Actual parameters and Formal parameters. Parameters defined inside the function are called formal parameters whereas the parameters passed during the function call are known as actual arguments.
- ❖ **Function Body:** The function body contains a collection of statements used to perform a specific task. The function body is enclosed within the braces.

LECTURE NOTES BY Bibek Raj Joshi (Functions)

4.3 Function Call (Passing arguments to a Function)

A function is used to transfer the control to the function definition. A function can be called by using the function name followed by a list of actual parameters and semicolon. When the function completes its task, the control is return to the position from where the function is called.

Syntax:

Function-name (actual parameters);

There are two approaches to pass the information to a function through arguments. They are: Call by Value and Call by Reference.

4.3.1 Argument (or Parameter) Pass By Value (Function Call By Value)

The process of passing the value of variable in function is called function call by value. Arguments are usually passed by value in C function calls. Actual and formal arguments refer to the different memory locations and the value of actual argument is copied into the formal argument. So, any changes made to the formal argument are not reflected in their corresponding actual arguments. The value of the actual argument will remain same.

Example: Program that illustrates call by value mechanism

```
#include<stdio.h>
#include<conio.h>
void swap(int, int);
void main()
{
    int a, b;
    a=10;
    b=20;
    swap(a, b); //Argument pass by value
    printf("\n%d %d", a, b); // prints 10 20
    getch();
}
void swap(int c, int d)
{
    int temp;
    temp = c;
    c = d;
    d = temp;
    printf("%d %d", c, d); //prints 20 10
}
```

4.3.2 Argument Pass By Reference (Function Call By Reference)

In this approach, the addresses of actual arguments are passed to the function call and the formal arguments will receive the address. The actual and formal arguments refer to the same memory location. So, changes in the formal arguments are reflected in actual arguments. Actual arguments are address of the ordinary variable, pointer variable or array name. It helps in returning back more than one value to the calling function.

Example: Program that illustrates call by reference mechanism

```
#include<stdio.h>
#include<conio.h>
```

LECTURE NOTES BY Bibek Raj Joshi (Functions)

```
void swap(int *x, int *y);
void main()
{
int a, b;
a=10;
b=20;
swap(&a, &b); // Argument pass by reference where the addresses of a and b are passed
printf("\n%d %d", a, b); // prints 20 10
getch();
}
void swap(int *c, int *d) // reference is made
{
int temp;
temp = *c;
*c = *d;
*d = temp;
printf("%d %d", c, d); //prints address represented by c and d
}
```

5. Macro

Macro is a preprocessor which hold the value of any constant so it is also called symbolic constant. It is globally declared before the main functions. It makes execution fast because of pre-execution but takes more memory space. Macro and its expansion are separated by blanks or tabs. Macro definition is never terminated by a semicolon.

Use of macro

- The execution is fast due to the pre-execution
- It reduces the program size.
- The macro is useful in the place where the numbers are not to be allowed.

For example: WAP to calculate the area of a circle using macro.

```
#include<stdio.h>
#include<stdio.h>
#include<conio.h>
#define PI 3.14159
float area(float);
void main()
{
float r,a;
printf("Enter the radius of a circle: ");
scanf("%f",&r);
a=area(r);
printf("The area of a circle is %f",a);
getch();
clrscr();
}
float area(float radius)
{
float area;
```

LECTURE NOTES BY Bibek Raj Joshi (Functions)

```
area=PI*radius*radius;
return(area);
}
```

Here,

= Pre-processor

define = Macro representation

PI = Macro template or Macro

3.14159 = Macro expansion

5.1 Predefined Macros

C defines a number of macros that should not be directly modified.

Macro	Description
__DATE__	The current date as a character literal in "MMM DD YYYY" format
__TIME__	The current time as a character literal in "HH:MM:SS" format
__FILE__	This contains the current filename as a string literal.
__LINE__	This contains the current line number as a decimal constant.

For example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
printf("File :%s\n", __FILE__ );
printf("Date :%s\n", __DATE__ );
printf("Time :%s\n", __TIME__ );
printf("Line :%d\n", __LINE__ );
getch();
}
```

Output

```
File :test.c
Date :Jun 2 2012
Time :03:36:24
Line :8
```

5.2 Parameterized Macros

Macros with arguments must be defined using the #define directive before they can be used. The argument list is enclosed in parentheses and must immediately follow the macro name. Spaces are not allowed between macro name and open parenthesis.

For example - 1:

```
#include <stdio.h>
#define MAX(x,y) ((x) > (y) ? (x) : (y))
void main()
{
printf("Max between 20 and 10 is %d\n", MAX(10, 20));
}
```

Output: Max between 20 and 10 is 20

LECTURE NOTES BY Bibek Raj Joshi (Functions)

6. External or Global Variables

Storage Location	Memory
Default initial value	Zero
Scope	Global
Life time	As long as the programs execution does not come to end.

The variables that are common to all the functions are declared outside the functions. If it is declared in the Global declaration section, it is used by all the functions in the program. Memory for the global variables is allocated when the program gets executed and de allocated only at the end of program execution. The return statement is used to transfer the control back to the calling program.

If we use the 'extern' keyword then we must initialize the value.

Example:

```
#include<stdio.h>
#include<conio.h>
int count=0;
void main()
{
    printf("the initial count value=%d",count);
    fun();
}
fun()
{
    count++;
    printf("\nthe count value=%d",count);
}
```

Output:

```
the initial count value=0
the count value=1
```

7. Preprocessor directive

The C Preprocessor is not part of the compiler, but is a separate step in the compilation process. In simplistic terms, a C Preprocessor is just a text substitution tool and they instruct compiler to do required pre-processing before actual compilation. All preprocessor commands begin with a pound symbol (#). A preprocessor is built into the C compiler. When a program is to be compiled, then the source code is first preprocessed. A preprocessor has following format:

```
#include<header file>
```

Thus, the preprocessor directive line must not end with any symbol and only one preprocessor directive can appear in one line.

8. Header File in C:

Header file contains different predefined functions, which are required to run the program. All header files should be included explicitly before main () function. It allows programmers to separate functions of a program into reusable code or file. Header file has extension like '*.h'. The prototypes of library functions are gathered together into various categories and stored in header files. Various header files with their functions are:

- **stdio.h** (standard input output header file) used for standard function like **printf()** - output function used to print a message or value of a variable on the screen.

LECTURE NOTES BY Bibek Raj Joshi (Functions)

scanf() - input function used to input a value through the keyboard.

putchar() - output function used to print a single character

getchar() - input function used to input a single character

fprintf() - output function used to print a file

fscanf() - input function used to input a file

- **string.h**: Includes the string functions like
 - puts()** - output function used to print a string
 - gets()** - input function used to input a string
 - strlen()** - find the length of given string
 - strcpy()** - copy string from variable to another
- **math.h**: Includes the mathematical functions like
 - sqrt()** - to find the square root of a number
 - pow()** - to calculate a value raised to power
 - abs()** - to find the absolute value of an integer
 - log()** - to calculate the natural logarithm, etc.
- **conio.h** (console input output header file) used for display controlling function like
 - getch()**; - Function used to hold the output value of program until pressing any key. It doesn't echo character.
 - getche()**; - It echoes (show) the character.
 - clrscr()**; - Function used to clear the screen holding the previous values.

9. **Recursive Function (Nesting of Function)**

In C, it is possible for the functions to call themselves. A function that calls itself repeatedly until the specific condition has been satisfied is called recursive function. And recursion is a process by which function call itself repeatedly.

Example1: Factorial of given number using recursive function

```
#include<stdio.h>
#include<conio.h>
int factorial (int);
void main()
{
    int n,res;
    printf("enter the number whose factorial is to be calculated: ");
    scanf("%d",&n);
    res=factorial(n);
    printf("\nthe factorial of %d is %d",n,res);
    getch();
    clrscr();
}
int factorial(int n)
{
    if(n==0||n==1)
    {
        return(1);
    }
    else
    {
        return(n*factorial(n-1));
    }
}
```

LECTURE NOTES BY Bibek Raj Joshi (Functions)

```
    }  
}
```

Example2: Program to find nth Fibonacci number using recursive function

```
#include<stdio.h>  
#include<conio.h>  
int fibo(int);  
void main()  
{  
    int n,res;  
    printf("\n enter the fibonacci term: ");  
    scanf("%d",&n);  
    res=fibo(n);  
    printf("\n the %d th fibonacci number is:%d",n,res);  
    getch();  
    clrscr();  
}  
int fibo(int n)  
{  
    if(n==1)  
    {  
        return(0);  
    }  
    else if(n==2)  
    {  
        return(1);  
    }  
    else  
    {  
        return(fibo(n-1)+fibo(n-2));  
    }  
}
```

Example3: Program to print Fibonacci series up to Nth term using recursive function

```
#include<stdio.h>  
#include<conio.h>  
#define n 5  
void fibo (int, int, int);  
void main()  
{  
    int res,f0=0,f1=1,i=3;  
    printf("\n the fibonacci series is:\n ");  
    printf("%d\t%d",f0,f1);  
    fibo(f0,f1,i);  
    getch();  
    clrscr();  
}  
void fibo(int f1,int f2,int i)
```


LECTURE NOTES BY Bibek Raj Joshi (Functions)

```
{
    int f3;
    f3=f1+f2;
    printf("\t%d",f3);
    f1=f2;
    f2=f3;
    i++;
    if(i<=n)
    {
        fibo(f1,f2,i);
    }
}
```

Examples of Function

Example1: WAP to find the square and sum of square of two arguments using function

```
#include <stdio.h>
#include <conio.h>
int sqr(int);
int sumsqr(int,int);
void main()
{
    int num1,num2,res1,res2,sum;
    printf("Enter two numbers:");
    scanf ("%d %d",&num1,&num2);
    res1= sqr(num1);
    res2= sqr(num2);
    sum= sumsqr(num1,num2);
    printf ("The square of %d is %d",num1,res1);
    printf("\nThe square of %d is %d",num2,res2);
    printf("\nSum of squares is %d",sum); getch();
    getch();
    clrscr();
}

int sqr(int z)
{
    return(z*z);
}

int sumsqr(int x, int y)
{
    return(x*x+y*y);
}
```

Example 2: Write a program to calculate the area and perimeter of a rectangle using function

```
#include<stdio.h>
#include<conio.h>
void area(int,int);
```

LECTURE NOTES BY Bibek Raj Joshi(Functions)

```
void perimeter(int,int);
void main()
{
int l,b;
printf("Enter length and breadth:");
scanf("%d%d",&l,&b);
area(l,b);
perimeter(l,b);
getch();
clrscr();
}
```

```
void area(int x,int y)
{
int area1;
area1=x*y;
printf("area=%d",area1);
}
void perimeter(int x,int y)
{
int peri;
peri=2*(x+y);
printf("\nperimeter=%d",peri);
}
```

Example 3: WAP to find factorial using function

```
#include<stdio.h>
#include<conio.h>
int factorial(int);
void main()
{
int n,result;
printf("Enter the positive number:");
scanf("%d",&n);
result=factorial(n);
printf("\nFactorial=%d",result);
getch();
clrscr();
}
int factorial (int n)
{
int i, fact=1;
for(i=1; i<=n; i++)
{
fact=fact*i;
}
return (fact);
}
```

LECTURE NOTES BY Bibek Raj Joshi (Functions)

Example 4: Write a program to generate the Fibonacci series using function.

```
#include<stdio.h>
#include<conio.h>
void fibo(int, int, int);
void main()
{
int a,b,num;
printf("Enter first number");
scanf("%d",&a);
printf("Enter second number");
scanf("%d",&b);
printf("Enter number of terms of series.");
scanf("%d",&num);
fibo(a,b,num);
getch();
clrscr();
}
void fibo(int x,int y,int n)
{
int sum=0,i;
printf("%d\t%d",x,y);
for(i=3; i<=n; i++)
{
sum=x+y;
printf("\t%d",sum);
x=y;
y=sum;
}
}
```