

Compiled by ab

Programming in C (HCAC-104)

I semester, BCA

Ambition Guru College

Compiled by :
Ankit Bhattarai

Unit IV

(7 hrs.)

Function

- Function Concept: Function definition and return statement, Function prototype
- Different ways of using function, Call by value, call by reference, Recursion

- Function is a self-contained block of code or statement that performs a particular task.
- Every program must contain one function named `main()` from where the program execution begins.
- Complex problem can be solved by breaking them into 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 single unit.

Advantages of function

- It makes program significantly easier to understand and maintain by breaking up them into easily manageable chunks.
- A single function can be used multiple times which avoids rewriting of same code again and again.
- Different programmers working on one large project divide the workload by writing different functions.
- Program that use functions are easier to design debug and maintain.
- C functions can be used to build a customized library of frequently used routines

Disadvantage

- It increase the execution time because when function is called the control have to jump to function definition to perform the particular task ,and after completion of task , control again come back to the function call.

Example of Function:

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

int sum(int,int); //function declaration

int main()
{
    int a, b,result;
    printf("Enter the first number\n");
    scanf("%d",&a);
    printf("Enter the second number\n");
    scanf("%d",&b);
    /* Calling the function here, the function return type is integer
    so we need an integer variable to hold the returned value of this
    function.*/

    result =sum(a,b);
    /*actual arguments a and b are passed in calling function*/

    printf ("Sum of two numbers= %d, result);
    return 0;
}
```

Example of Function:

```
/*formal arguments x and y are passed in called  
function*/
```

```
int sum(int x, int y) //function definition  
{
```

```
    int r;
```

```
    r=x+y;
```

```
/* Function return type is integer so we are returning an  
integer value, the sum of the passed numbers.*/
```

```
    return r;
```

```
}
```

Types of Function

Types of Function

Compiled by ab

Library functions	User defined functions
1. They are predefined functions/built in functions in C library.	1. They are the function which are created by the user as per his own requirements.
2. They are the part of header files (such as stdio.h, conio.h, math.h) which are called during runtime.	2. They are the part of program which is compiled at runtime.
3. The name of function ID is given by developers which can't be changed.	3. The name of function id is declared by user which can be changed.
4. The function's name, its return type, their arguments number and their types have been already defined .	4. The user has choice to choose function name, its return type, number of argument and their types.
5.Example. printf(),scanf(),sqrt(),pow(),clrscr() etc.	5.Example. int fact(int n), float sum(float x,float y)

Types of Function

- Program to illustrate use of user defined function and library functions.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int square(int);
int main()
{
    int num=5,result;
    result=pow(num,2);
    printf("Square of number using library function = %d",
    result);
    result= square(num);
    printf("\nSquare of number using user defined function=%d"
    ,result);
    return 0;
}
int square(int x)
{
    return (x * x);
}
```

Components associated with function

Function definition

The collection of program statements that describes the specific task to be done by the function is called function definition. It consists of:

Function header: which defines function's name, its return type and its argument list

Function body : which is a block of code enclosed in parenthesis.

Syntax:

```
return_type function_name(data_type variable1,data_type variable2,...,data_type variable n)
{
statements ;
.....
.....
return value;
}
```

Function definition

Note: If a function doesn't return any value , then its return type is void

Syntax:

```
void function_name (data_type varibale1,data_type variable2,....,data_type
variable n)
{
statements ;
.....
.....
}
```

Function Prototype / Function Declaration

The function declaration or prototype is a model or blueprint of a function. If a function is used before it is defined in a program, then function declaration or prototype is needed to provide the following information to the compiler.

- The name of function
- The type of the value returned by the function
- The number and type of arguments that must be supplied while calling the function

The syntax for function declaration is :

return_type function_name(type1, type2 ,type3 type n);

Here, return_type specifies the data type of the value returned by the function. A function can return value of any data type.If there is no return value, the keyword void is used.

Function Prototype / Function Declaration

The function declaration and declarator or header in function definition must use the same function name, number of arguments, argument types and return types. Some examples of function prototype are:

```
int add(int a,int b);
```

```
int factorial(int n);
```

Components associated with function : Accessing/calling a function

Compiled by ab

- When a program calls a function, the program control is transferred to the called function. A called function performs a defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns the program control back to the main program.
- A function can be called or accessed by specifying its name, followed by a list of arguments enclosed in parentheses and separated by commas. **For example, the function `sum()` with two arguments is called by `sum(a,b)` to add two numbers.**
- During function call, function name must match with function prototype name, the type of return type, the number of arguments and order of arguments.

The job of return statement is to hand over some value given by function body to the point from where the call was made. The function defined with its return type must return a value of that type.

For example: If a function has return type int, it returns an integer value from the called function to the calling function.

The main functions return statement are:

- It immediately transfers the control back to the calling program after execution of return statement (i.e. no statement within the function body is executed after the return statement.)
- It returns value to the calling function.

Components associated with function : **Function Parameter (Arguments)**

Compiled by ab

Parameters provide the data communication between calling function and called function. They can be classified into actual parameters and formal parameters.

Actual parameters:

- These are the parameters that transfer data from the calling function to the called function.
- This type of parameter is used in function call.

Formal parameters:

- These parameters used to receive the data sent by the calling function. These are also used to send data from calling function to the called function.
- These parameters used in function definition.

In given example the parameter a and b are actual parameter which are used in calling function and x and y are formal parameters which are used in called function.

Types of Function

- Program to illustrate use of user defined function and library functions.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int square(int);
int main()
{
    int num=5,result;
    result=pow(num,2);
    printf("Square of number using library function = %d",
    result);
    result= square(num);
    printf("\nSquare of number using user defined function=%d"
    ,result);
    return 0;
}
int square(int x)
{
    return (x * x);
}
```

Classwork

WAP to calculate the area and perimeter of rectangle using function

```
#include<stdio.h>
#include<conio.h>
void area(int,int);
void perimeter(int,int);
int main()
{
    int l,b;
    printf("Enter the length and breadth of rectangle\n");
    scanf("%d%d",&l,&b);
    area(l,b);
    perimeter(l,b);
    getch();
    return 0;
}
void area(int x,int y)
{
    int area;
    area=x*y;
    printf("area=%d",area);
}
void perimeter(int x,int y)
{
    int peri;
    peri=2*(x+y);
    printf("Perimeter=%d",peri) ;
}
```

Classwork

WAP to calculate the area of two circles having different radius using the same function area.

```
#include<stdio.h>
#include<conio.h>
float area(float r);
int main()
{
    float r1,r2,a1,a2;
    printf("Enter the radius of first circle");
    scanf("%f",&r1);
    a1=area(r1);
    printf("Enter the radius of second circle");
    scanf("%f",&r2) ;
    a2=area(r2);
    printf("Area of first circle=%f",a1);
    printf("\nArea of second circle=%f",a2);
    getch();
}
float area(float r)
{
    float a;
    a=3.14*r*r;
    return a;
}
```

Classwork

WAP to check given number is
palindrome or not using function

```
#include<stdio.h>
#include<conio.h>
void palindrome(int n);
int main()
{
    int num;
    printf("Enter the number you want to check\n");
    scanf("%d",&num);
    palindrome(num);
    getch();
    return 0;
}
```

Classwork

WAP to check given number is
palindrome or not using function

```
void palindrome(int n)
{
    int rem, rev=0, a;
    a=n;
    while(n!=0)
    {
        rem=n%10;
        rev=rev*10+rem;
        n=n/10;
    }
    if(a==rev)
    {
        printf("Number is palindrome");
    }
    else
    {
        printf("Number is not palindrome");
    }
}
```

Classwork

WAP to check given number is
palindrome or not using function

```
void palindrome(int n)
{
    int rem, rev=0, a;
    a=n;
    while(n!=0)
    {
        rem=n%10;
        rev=rev*10+rem;
        n=n/10;
    }
    if(a==rev)
    {
        printf("Number is palindrome");
    }
    else
    {
        printf("Number is not palindrome");
    }
}
```


Global variables:

Global variables are accessible to all functions defined in the program. Global variables are declared outside any function, generally at the top of program after preprocessor directives. It is useful to declare variable global if it is to be used by many functions in the program. The default initial values for this variable is zero. The lifetime is as long as the program execution does not come to an end.

Local variables:

The variables that are defined within the body of a function or block and local to that function or block only are known as local variables. The name and value of local variables are valid within the function in which it is declared. They are unknown to other. The local variables are created when the function is called and destroyed automatically when function is exited function.

Global Vs Local Variable

```
#include<stdio.h>
#include<conio.h>
void fun();
int x=5; //global declaration
int main()
{
int a=10; //local declaration
printf("value of x=%d,a=%d",x,a);
fun();
getch();
return 0;
}
```

Output

Value of x=5, a=10

Value of x=5, b=20

```
void fun()
{
int b=20; //local declaration
printf("\nvalue of x=%d,b=%d",x,b);
}
```

Different ways of using function

Different ways of using function

Compiled by ab

Function can be categorized in four types, on the basis of arguments and return value.

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

1. Function with no arguments and no return values

When a function has no arguments, it does not receive any data from the calling function. Similarly, when it doesn't return a value, the calling function does not receive any data from the called function. Thus in such type of functions, there is no transfer between the calling function and the called function. This type of function is defined as.

```
void function_name( )  
{  
    /*body of the function*/  
}
```

1. Function with no arguments and no return values

- The keyword **void** means the function does not return any value.
- There is no arguments within parenthesis which implies function has no argument and it does receive any data from the called function.

Program to illustrate the “function with no arguments and no return values”

```
#include<stdio.h>
#include<conio.h>
void sum();
int main()
{
    sum();
    getch();
    return 0;
}

void sum()
{
    int x,y,r;
    printf("Enter the two numbers\n");
    scanf("%d%d",&x,&y);
    r=x+y;
    printf("The sum of numbers=%d",r);
}
```

Different ways of using function

Compiled by ab

2. Function with arguments but no return value

- These type of function has arguments and receives the data from the calling function.
- The function completes the task and does not return any values to the calling function. Such type of functions are defined as

```
void function_name(argument list)
{
/*body of function*/
}
```

Program to illustrate the “function with arguments but no return values”

```
#include<stdio.h>
#include<conio.h>
void sum(int,int);
int main()
{
    int a,b;
    printf("Enter the two numbers\n");
    scanf("%d%d",&a,&b);
    sum(a,b);
    getch();
    return 0;
}

void sum(int x,int y)
{
    int r;
    r=x+y;
    printf("The sum of numbers=%d",r);
}
```

3. Function with no arguments but with return values

- These type of function does not receive any arguments but the function return values to the calling function. Such type of functions are defined as

```
return_type function_name( )  
{  
/*body of function*/  
}
```

Program to illustrate the “function with no arguments and with return values”

```
#include<stdio.h>  
#include<conio.h>  
int sum();  
int main()  
{  
    int result;  
    result=sum();  
    printf("The sum of numbers=%d",result);  
    getch();  
    return 0;  
}  
  
int sum()  
{  
    int x,y,r;  
    printf("Enter the two numbers\n");  
    scanf("%d%d",&x,&y);  
    r=x+y;  
    return r;  
}
```

4. Function with arguments and return value

- The function of this category has arguments and receives the data from the calling function.
- After completing its task ,it returns the result to the calling function through return statement.
- Thus there is data transfer between called function and calling function using return values and arguments. These type of functions are defined as

```
return_type function_name(argument  
list)  
{  
/*body of the function*/  
}
```

Program to illustrate the “function with arguments and return values”

```
#include<stdio.h>  
#include<conio.h>  
int sum(int,int);  
int main()  
{  
int a,b,result;  
printf("Enter the two numbers\n");  
scanf("%d%d",&a,&b);  
result=sum(a,b);  
printf("The sum of numbers=%d",result);  
getch();  
return 0;  
}  
  
int sum(int x,int y)  
{  
int r;  
r=x+y;  
return r;  
}
```


Call by Value

The arguments in function can be passed in two ways, namely **call by value** and **call by reference**.

Function call by value (or pass by value)

- In this method the value of each 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 the actual argument in the calling function.

Call by Value

Explanation:

In this example, the values of a and b are passed in function swap() by value. The value of a is copied into formal argument x and value of b is copied into formal argument y. The copy of value of a and b to x and y means the original values of a and b remain same and these values are copied into the variables x and y also. Thus when x and y are changed within the function, the values of the variables x and y are changed but the original value of a and b remains same.

```
#include<stdio.h>
#include<conio.h>
void swap(int x,int y);
int main()
{
    int a,b;
    a=10;
    b=20;
    printf("value before swapping a=%d and b=%d",a,b);
    swap(a,b);
    printf("\nvalue after swapping a=%d and b=%d",a,b);
    getch();
    return 0;
}

void swap(int x,int y)
{
    int temp;
    temp=x;
    x=y;
    y=temp;
}
```

Output:

Value before swapping a=10 and b=20

Value after swapping a=10 and b=20

Call by Reference

Function call by reference (Or pass by Reference)

- In this method the address of actual arguments in the calling function are copied into formal arguments of the called function .This means that using these addresses the actual arguments can be accessed.
- In call by reference since the address of the value is passed any changes made to the value reflects in the calling function.

Call by Reference

Explanation: *In this example ,the address of variables (i.e. &a and &b) are passed in function. These addresses are received by corresponding formal arguments in function definition. To receive addresses, the formal arguments must be of type pointers which stores the address of variables. The address of a and b are copied to the pointer variable x and y in function definition.*

When the values in addresses pointed by these pointer variables are altered, the values of original variables are also changed as the content of their addresses have been changed .Thus, while arguments are passed in function by reference ,the original values are changed if they are changed within function.

```
#include<stdio.h>
#include<conio.h>
void swap(int *x,int *y);
int main()
{
    int a,b;
    a=10;
    b=20;
    printf("The value before swapping are a=%d and
    b=%d",a,b);
    swap(&a,&b);
    printf("\nThe values after swapping are a=%d
    and b=%d",a,b);
    getch();
    return 0;
}

void swap(int *x,int *y)
{
    int temp;
    temp=*x;
    *x=*y;
    *y=temp;
}
```

Output:

Value before swapping a=10 and b=20

Value after swapping a=20 and b=10

Recursion

A function that calls itself is known as a recursive function. Recursion is a process by which function calls itself repeatedly until some specified condition will be satisfied.

To solve a problem using recursive method, two conditions must be satisfied .They are

- Problem could be written or defined in terms of previous result
- Problem statement must include stopping condition

Advantages

- Avoid unnecessary calling of functions.
- Through Recursion one can solve problems in easy way while its iterative solution is very big and complex
- Reduces time complexity.
- Using recursion, the length of the program can be reduced.
- Extremely useful when applying the same solution repeatedly.

Disadvantages

- Recursive solution is always logical and it is very difficult to trace. (debug and understand).
- For each step we make a recursive call to a function. For which it occupies significant amount of stack memory with each step.
- It is usually slower due to the overhead of maintaining the stack.
- May cause stack-overflow if the recursion goes too deep to solve the problem
- If the programmer forgets to specify the exit condition in the recursive function, the program will execute out of memory.

Recursion

- WAP to find the factorial of a given number using recursive function

```
#include<stdio.h>
#include<conio.h>
long int fact(int n);
int main()
{
    int num;
    long int f;
    printf("Enter the number\n");
    scanf("%d",&num);
    f=fact(num);
    printf("The factorial of a given number is %ld",f);
    getch();
    return 0;
}

long int fact(int n)
{
    if(n==1||n==0)
        return 1;
    else
        return n*fact(n-1);
}
```

Recursion

- Write a program to find the sum of first n natural number using recursive function

```
#include<stdio.h>
#include<conio.h>
int snatural(int n);
int main()
{
    int num,s;
    printf("Enter the value of n \n");
    scanf("%d",&num);
    s=snatural(num);
    printf("The sum of n natrual number is %d",s);
    getch();
    return 0;
}

int snatural(int n)
{
    if(n==1)
        return 1;
    else
        return n + snatural(n-1);
}
```

Recursion Vs Iteration

Compiled by ab

Recursion	Iteration
The statement in a body of function calls the function itself.	Allows the set of instructions to be repeatedly executed.
Recursion is always applied to functions.	Iteration is applied to iteration statements or "loops".
In recursive function, only termination condition (base case) is specified	Iteration includes initialization, condition, execution of statement within loop and updating (increments/decrements) the control Variable
Infinite recursion can crash the system.	Infinite loop uses CPU cycles repeatedly.
Slow in execution	Fast in execution

THANK YOU
Any Queries ?