Rajmani Kumar,
Lecturer, Dept. of BCA
S.U.College, Hilsa (Nalanda)
Patliputra University, Patna

BCA-2nd Year **Paper-III**

Function

A function is a group of statements that together perform a task. Every C program has at least one function, which is **main()**, and all the most trivial programs can define additional functions.

Function Declaration OR Function Prototype:

1. It is also known as function prototype .
2. It inform the computer about the three things
   a) Name of the function
   b) Number and type of arguments received by the function.
   c) Type of value return by the function

Syntax :
return_type function_name (type1 arg1 , type2 arg2);
OR
return_type function_name (type1 type2);

3. Calling function need information about called function .If called function is place before calling function then the declaration is not needed.

Function Definition:

1. It consists of code description and code of a function .
   It consists of two parts
   a) Function header
   b) Function coding

Function definition tells what are the I/O function and what is going to do.
Syntax:
return_type function_name (type1 arg1 , type2 arg2)
{
local variable;
statements ;
return (expression);

}
2. Function definition can be placed any where in the program but generally placed after the main function .
3. Local variable declared inside the function is local to that function. It cannot be used anywhere in the program and its existence is only within the function.
4. Function definition cannot be nested.
5. Return type denote the type of value that function will return and return type is optional if omitted it is assumed to be integer by default.

USER DEFINE FUNCTIONS VS STANDARD FUNCTION:
User Define Function:
A function that is declare, calling and define by the user is called user define function. Every user define function has three parts as:
1. Prototype or Declaration
2. Calling
3. Definition

Standard Function:
The **C standard library** is a standardized collection of header files and library routines used to implement common operations, such as input/output and character string handling. Unlike other languages (such as COBOL, FORTRAN, and PL/I) C does not include built in keywords for these tasks, so nearly all C programs rely on the standard library to function.

FUNCTION CATAGORIES

There are four main categories of the functions these are as follows:
1. Function with no arguments and no return values.
2. Function with no arguments and a return value.
3. Function with arguments and no return values.
4. Function with arguments and return values.

Function with no arguments and no return values*:*
syntax:
```
void funct (void);
main ( )
{
funct ( );
}
void funct ( void );
{
}
```

NOTE: There is no communication between calling and called function. Functions are executed independently, they read data & print result in same block.

Example:

```
void link (void) ; int main ()
{
link ();
}
void link ( void );
{
printf (" link the file ")
}
```

Function with no arguments and a return value:

This type of functions has no arguments but a return value

Example:

```
int msg (void) ;
int main ( )
{
int s = msg ( );
printf( "summation = %d" , s);
}
int msg ( void )
{
int a, b, sum ; sum = a+b ; return (sum) ;
}
```

NOTE: Here called function is independent, it read the value from the keyboard, initialize and return a value .Both calling and called function are partly communicated with each other.

Function with arguments and no return values:

Here functions have arguments so, calling function send data to called function but called function does no return value. such functions are partly dependent on calling function and result obtained is utilized by called function .

Example:

```
void msg ( int , int );
int main ( )
{
int a,b;
a= 2; b=3;
msg( a, b);
}
void msg ( int a , int b)
{
int s ;
sum = a+b;
printf (“sum = %d” , s ) ;
}
```

Function with arguments and return value:

Here calling function of arguments that passed to the called function and called function return value to calling function.

example:

```
int msg ( int , int ) ;
int main ( )
{
int a, b;
a= 2; b=3;
int s = msg (a, b);
printf (“sum = %d” , s ) ;
}
int msg( int a , int b)
{
int sum ;
sum =a+b ; return (sum);
}
```

ACTUAL ARGUMENTS AND FORMAL ARGUMENTS

Actual Arguments:

1. Arguments which are mentioned in the function in the function call are known as calling function.
2. These are the values which are actual arguments called to the function.

It can be written as constant , function expression on any function call which return a value .

ex: funct (6,9) , funct (a,b)

Formal Arguments:

1. Arguments which are mentioned in function definition are called dummy or formal argument.
2. These arguments are used to just hold the value that is sent by calling function.
3. Formal arguments are like other local variables of the function which are created when function call starts and destroyed when end function.

Basic difference between formal and local argument are:

a) Formal arguments are declared within the ( ) where as local variables are declared at beginning.

b) Formal arguments are automatically initialized when a value of actual argument is passed.

c) Where other local variables are assigned variable through the statement inside the function body.

Note: Order, number and type of actual argument in the function call should be matched with the order , number and type of formal arguments in the function definition .

PARAMETER PASSING TECHNIQUES:

1. call by value
2. call by reference

Call by value:

Here value of actual arguments is passed to the formal arguments and operation is done in the formal argument.

Since formal arguments are photo copy of actual argument, any change of the formal arguments does not affect the actual arguments
Changes made to the formal argument t are local to block of called function, so when control back to calling function changes made vanish.

Example:

```
void swap (int a , int b) /* called function */
{ int t; t = a; a=b; b = t;
}
main( )
{
int k = 50,m= 25;
swap( k, m) ; / * calling function */ print (k, m); / * calling function */
}
```

Output:

```
50, 25
```

Explanation:

int k= 50, m=25 ;

Means first two memory space are created k and m , store the values 50 and 25 respectively.

swap (k,m);

When this function is calling the control goes to the called function.

    void swap (int a , int b),
    k and m values are assigned to the ‘a’ and ‘b’.
    then a= 50 and b= 25 ,

After that control enters into the function a temporary memory space ‘t’ is created when int t is executed.

    t=a; Means the value of a is assigned to the t , then t= 50.
    a=b; Here value of b is assigned to the a , then a= 25;
    b=t; Again t value is assigned to the b , then b= 50;

after this control again enters into the main function and execute the print function print (k,m). it returns the value 50 , 25.

NOTE:

Whatever change made in called function not affects the values in calling function.

Call by reference:

Here instead of passing value address or reference are passed. Function operators or address rather than values .Here formal arguments are the pointers to the actual arguments.

Example:

```
#include<stdio.h> void add(int *n); int main()
```

```
{
int num=2;
printf("\n The value of num before calling the function=%d", num); add(&num);
printf("\n The value of num after calling the function = %d", num); return 0;
}
void add(int *n)
{
*n=*n+10;
printf("\n The value of num in the called function = %d", n);
}
```

Output:

```
The value of num before calling the function=2
The value of num in the called function=20
 The value of num after calling the function=20
```

NOTE:

In call by address mechanism whatever change made in called function affect the values in calling function.

EXAMPLES:

1: Write a function to return larger number between two numbers:

```
int fun(int p, int q)
{
int large;
if(p>q)
{
large = p;
}
else
{
large = q;
}
return large;
}
```

2: Write a program using function to find factorial of a number.

```
#include <stdio.h> int factorial (int n)
{
int i, p;
p = 1;
for (i=n; i>1; i=i-1)
{
p = p * i;
}
return (p);
}
void main()
{
int a, result;
printf ("Enter an integer number: ");
scanf ("%d", &a);
result = factorial (a);
printf ("The factorial of %d is %d.\n", a, result);
}
```

EXERCISE:

1. What do you mean by function?
2. Why function is used in a program?
3. What do you mean by call by value and call by address?
4. What is the difference between actual arguments and formal arguments?
5. How many types of functions are available in C?
6. How many arguments can be used in a function?
7. Add two numbers using

a) with argument with return type
b) with argument without return type
c) without argument without return type
d) without argument with return type

8. Write a program using function to calculate the factorial of a number entered through the keyboard.
9. Write a function power(n,m), to calculate the value of n raised to m.
10. A year is entered by the user through keyboard. Write a function to determine whether the year is a leap year or not.