UNIT-4

FUNCTIONS

A C-language program is nothing but collection of Function; these are the building blocks of a „C" program. Generally, a function mans a task.

"Function is a collection of logically related instructions which performs a particular task."

Functions are also used for modular programming in which a big task is divided into small modules and all the modules are inter connected.

How to create the function (Function declaration):-

```
<return value type><function name> (parameter list)
{
Body of the
function;
[return<value>];
}
```

<return value type>: It is specify the data type of the returning value.
<function name>:It is specify the name of the function. (Parameter list): It is the list of parameters used in the function.

Function definition:-

The collection of program statements in C that describes the specific task done by the function is called a Function definition. It consists of the function header and a function body.

```
<return type><function name>(parameter list)

{
<local variable declaration>;
<Statements>;
<return (value)>;
}
```

Function header:

Function header is similar to the function declaration but does not require the semicolon at the end. List of variables in the parenthesis are called Formal parameters. It consists of three parts:

1. Return type.
2. Function name.
3. Formal

parameters. Example:

int add(a, b)

Function body:

After the function header the statements in the function body (including variables and statements) called body of the function.

Example:

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

Return statement:

Return statement is used return some value given by the executable code within the function body to the point from where the call was made. General form:

```
return expression;
        or
return
value;
```

Where expression must evaluate to a value of the type specified in the function header for the return value.

```
EX: return a+b;
          Or
      return a;
```

If the type of return value has been specified as void, there must be no expression appearing in the return statement it must be written simply as

return;

Function call (Function calling):

All functions within standard library or user-written, are called with in this main() function, the function call statements involves the function, which means the program control passes to that of the function. Once the function completes its task, the program control is passed back to the calling environment.

Syntax:

<function name><(parameter
list)>; or
variable name=<function
name><(parameter list)>; Example:

1. Write a program to find the addition of

two numbers operations.

```
#include<stdio.h>
 int sum(int , int );
int main()
{
  int num1, num2, res;
  printf("\nEnter the two numbers : ");
  scanf("%d %d", &num1, &num2);
  res = sum(num1, num2);
  printf("Addition of two number is : %d"res);
  return (0);
}
int sum(int num1, int num2)
{
  int num3;
  num3 = num1 + num2;
  return (num3);
}
```

2. Write a program to find the all arthametic operations.

```
#include<stdio.h>
int add(int n1, int n2);
int subtract(int n1, int n2);
int multiply(int n1, int n2);
int divide(int n1, int n2);
main()
{
 int num1, num2,x,y,z,m;
printf("Enter two numbers: ");
scanf("%d %d", &num1, &num2);
x=add(num1, num2);
y=subtract(num1, num2);
z=multiply(num1, num2);
m=divide(num1, num2);
  printf("%d\n", x);
 printf("%d\n", y);
  printf("%d\n", z);
printf("%d",m );

return 0;
}
```

```
   int add(int n1, int n2)
  {
     int result;
     result = n1 + n2;
     return result;
  }

   int subtract(int n1, int n2)
   {
     int result;
     result = n1 - n2;
     return result;
   }
 int multiply(int n1, int n2)
   {
   int result;
     result = n1 * n2;
   return result;
   }

   int divide(int n1, int n2)
   {
   int result;
     result = n1 / n2;
     return result;
   }
```

3. Write a program that uses a function to check whether a number is even or odd

```
#include <stdio.h>

int find_Num(int);
int main()
{
   int num;
   printf("Enter a number to check odd or even\n");
   scanf("%d",&num);
   find_Num(num);
   return 0;
}
```

```
int find_Num(int num){
     if(num%2==0){
    printf("\n%d is an even number",num);
}
else{
    printf("\n%d is an odd number",num);
}
     }
```

Types of functions:

The functions are also used for modular programming in which a big task is divided into small modules and all the modules are interconnected. Functions are basically classified as:

a. Standard library functions(built-in functions).
b. user defined functions.

Standard library functions:-

Library functions come along with the compiler and they can be used in any program directly. User can"t be modified the library functions.

Example:

sqrt();, printf();, clrscr();, abs();, strlen();, getch();etc………………

Example program:

```
#include<math.h>
main()
{
int x;
printf("enter x value \n");
```

```
        scanf(“%d”,&x);
        printf(“square root value of x is %d”,
        sqrt(x)); getch();
        }
```

user defined functions: user defined functions is one which will be defined by the ser in program to group certain statements together. The scope of a user defined function is to the extent of the program only, in which it has been defined. main is also a user defined function.

Syntax:

```
        <returntype><function name)(<parameter list>)
        {
        Function body;
        <return value>;
        }
```

Example program: to find the area of triangle

```
#include<stdio.h>
int area(int x);

main()
{
int r,rd;
printf("enter radius for circle\n");
scanf("%d",&r);
rd=area(r);
printf("area of the circle is %d",rd);
}
int area(int x)
{
return(3.14*x*x);
}
```

Types of parameters

The nature of data communication between the calling function and the called function with arguments (parameter).

The parameters are two types:

1. Actual parameters or original parameters.
2. formal parameters or duplicate parameters.

```
main()
{
 …………
 Function1(a1, a2, a3,………am);
 …………
}
```

```
 Function1(x1, x2, x3,………xn);
{
 ……….
 ……......
 ……….
}
```

Actual parameters(arguments):

Actual parameters means original parameters for the function call. These parameters are declared at the time of function declaration. In the above example

a1,a2,a3,…… am are the actual parameters. When a function call is made, only a copy of the values of actual arguments is passed into the called function.

Formal parameters (duplicate arguments):

Formal parameters mean duplicate parameters for function call. These

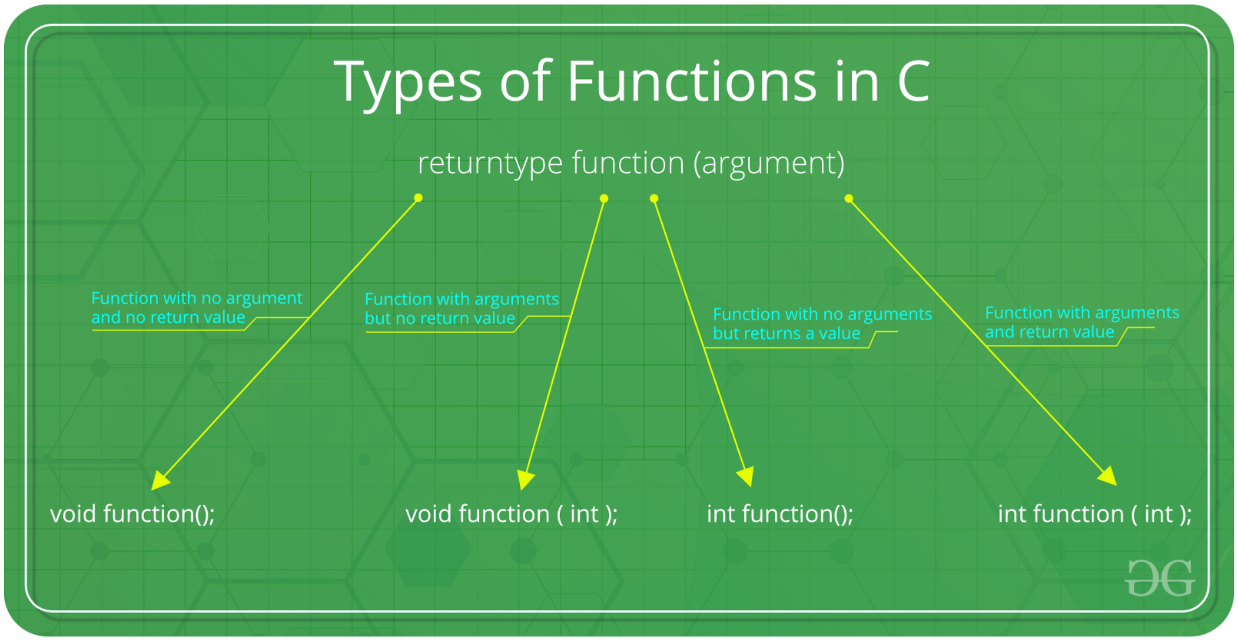
parameters are declared at the time of function definition. In the above example x1, x2, x3,………x n are the formal parameters, the actual and formal arguments should match in the number, type and order, the values of actual arguments are assigned to the formal arguments on a one to one basis, starting with the first argument.

Types of functions based on parameters:

A function depending on whether arguments are present or not and whether a value is returned or not, may belong to one of the following categories:

1. Functions with no arguments and no return value.
2. Functions with arguments and no return value.
3. Functions with arguments and return value.
4. Functions with no arguments and return value

.


Types of Functions in C
returntype function (argument)
Function with no argument and no return value
Function with arguments but no return value
Function with no arguments but returns a value
Function with arguments and return value
void function();
void function (int);
int function();
int function (int);

Functions with no arguments and no return value:

1. When a function has no arguments, it does not receive any data from the calling function. Similarly when it does not return a value, the calling function does not receive any data from the called function.
   Syntax :

   Function declaration : void function();
   Function call : function();
   Function definition :

```
                    void function()
                    {
                      statements;
  example:                }
```

```
#include <stdio.h>

void value(void);

void main()

{
    value();

}

void value(void)

{

    int year = 1, period = 5, amount = 5000, inrate = 0.12;

    float sum;

    sum = amount;

    while (year <= period) {

        sum = sum * (1 + inrate);

        year = year + 1;

    }

    printf(" The total amount is %f:", sum);

}
```

Output:

```
The total amount is 5000.000000
```

Functions with arguments and no return value:

This is another type where one way communication is possible between the callings and called function. That is, the called function will receive data from the calling function, but will not transfer any data to it.

Syntax :

Function declaration : void function (int);
Function call : function(x);
Function definition:

```
        void function( int x )
```

```
{
  statements;
}
```

Example program:

```
#include<stdio.h>
main()
{
int a,b,c;
printf("enter 3 numbers");
scanf("%d%d%d",&a,&b,
&c);
big(a,b,c);
}
big(x,y,z)
int x,y,z;
{
if(x>y&&x>z)
printf("\n %d is
biggest",x);
 else if(y>z)
printf("\n %d is
biggest",y);
else
printf("\n %d is biggest",z);
}
```

Functions with arguments and return value:

In the type, two way data communication takes place. That is, both the called and calling functions receive and transfer data from each other. Example program:

Syntax :

Function declaration : int function (int);

Function call : function(x);

Function definition:

```
int function( int x )
{
  statements;
  return x;
}
```

// C code for function with arguments

```
// and with return value


#include <stdio.h>

#include <string.h>

int function(int, int[]);

int main()

{

   int i, a = 20;

   int arr[5] = { 10, 20, 30, 40, 50 };

   a = function(a, &arr[0]);

   printf("value of a is %d\n", a);

   for (i = 0; i < 5; i++) {

       printf("value of arr[%d] is %d\n", i, arr[i]);

   }

   return 0;

}


int function(int a, int* arr)

{
   int i;

   a = a + 20;

   arr[0] = arr[0] + 50;

   arr[1] = arr[1] + 50;

   arr[2] = arr[2] + 50;

   arr[3] = arr[3] + 50;

   arr[4] = arr[4] + 50;

   return a;

}
```

Output:

```
value of a is 40
value of arr[0] is 60
value of arr[1] is 70
value of arr[2] is 80
value of arr[3] is 90
value of arr[4] is 100
```

<u>Functions with no arguments and return value.</u>

Function which does not have any argument, receive no data from the calling function and return a value to calling function.

Syntax :
Function declaration : int function();
Function call : function();
Function definition :

```
int function()
{
   statements;
    return x;
 }
```

Example program:

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

int sum();
int main()
{
   int num;
   num = sum();
   printf("\nSum of two given values = %d", num);
   return 0;
}

int sum()
{
   int a = 50, b = 80, sum;
   sum = sqrt(a) + sqrt(b);
   return sum;
}
```

Nested functions:-

One important advantage of c functions is that they can be called from and within another function. All the called and calling functions transfer and receive data with ach other and this is called “Nested function”.

Example program:

```
#include<stdio.h>
main()
{
printf("\n I am in main");
fun1();
printf("\n again I am in main");
}
fun1()
{
printf("\n I am in function1");
fun2();
}
fun2()
{
printf("\n I am in function2");
}
```

Recursion:

A recursion function is one that calls itself directly or indirectly to solve a smaller version of its task until a final call which does not require a self call.

Need of recursion

1. decomposition into smaller problems of same type
2. recursive calls must reduce problem size
3. a recursive function would call itself indefinitely
4. recursion is like a top-down

approach Factorial of a number:

n!=n*(n-1)*(n-2)*…*1 for n>0

0!=1

Example program:

```
#include<stdio.h>
main()
{
int
m,n;
printf("enter n
value");
scanf("%d",&n);
m=fact(n);
printf("factorial of n is
%d",m);
}
fact(int x)
{
if(x==1)
return
1; else
return(x*(fact(x-1));
    }
```

Recursion for fibonacci sequence:

```
#include<stdio.h>
int fib(int
val); main()
{
int i,j;
clrscr(
);
printf("\n enter th number of
terms"); scanf("%d",&i);
printf("Fibonacci sequence for %d terms
is:",i); for(j=0;j<=i;j++)
printf("%d",fib[j]);
getch();
}
int fib(int val)
{
if(val<=2)
return 1;
else
` return(fib(val-1)+fib(val-2));
}
```

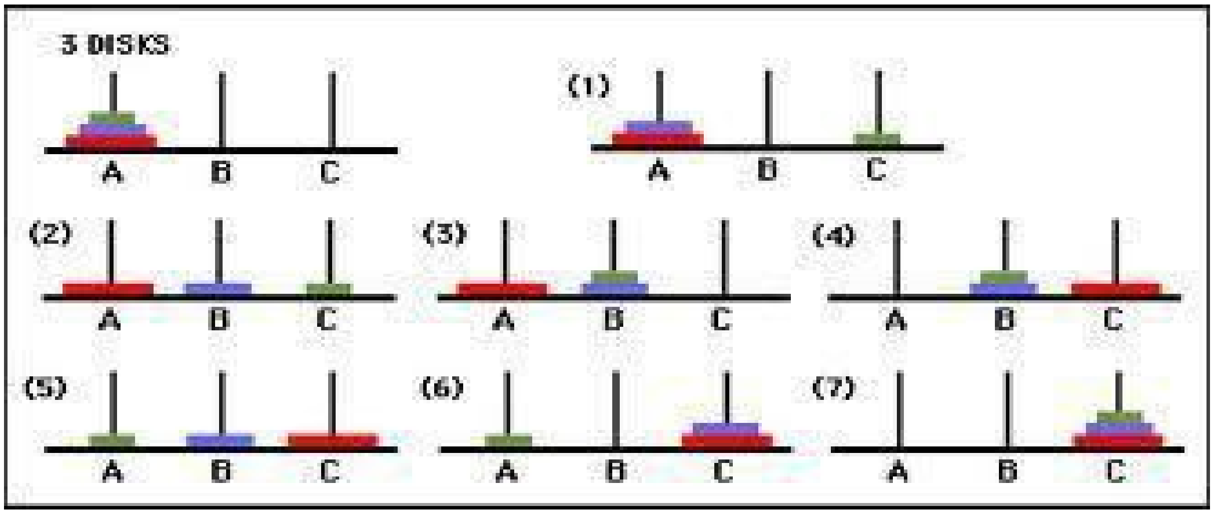
The towers of Hanoi:

The towers of Hanoi problem is a classic case study in recursion, it involves moving a specified number if disks from one tower to another using a third as an auxiliary tower.

Specification move n disks from peg A to peg C, using peg B as needed.

- Only one disk may be moved at a time.
- This disk must be the top disk on a peg.
- A larger disk can never be placed on top of a smaller disk.

Towers of Hanoi



The problem is not to focus on the first step, but on the hardest step i.e, moving the bottom disk to peg C.

- move disk3 from peg A to peg C
- move disk2 from pg A to peg B
- move disk3 from peg C to peg B
- moving disk1 from peg A to peg C
- moving disk3 from peg B to peg A
- moving disk2 from peg B to peg C
- moving disk3 from peg A to peg C