FUNCTIONS

Introduction

The programs written in C language are highly dependent on functions. The C program is nothing but the combination of one or more number of functions. Every C program starts with the user defined function main(). The main() calls other functions to share the work.

Definition of the function:

The function is a sub program or sub routine or self contained block of one or more executable statements, which performs a well defined task.

We have two types of functions.

1. Standard/ Pre Defined/ Library Functions.
2. User Defined Functions.

Pre Defined Functions:

These functions are pre defined set of statements. Their task is limited i.e., a function which is already exist in C header file is called as Standard or Pre defined or Library Function.

In these functions the user cannot understand the internal working of the function. The user can only use these functions but cannot modify them.

For example: printf(), scanf(), sqrt() and so on. If we want to use these functions in our programs we need to include the corresponding header file in our program.

User Defined functions in C

The functions defined by users according to the user requirements are called as user defined functions. In these functions the user can able to modify the function definition according to his or her requirement. The user has full scope on user defined functions.

The user certainly understands t he internal working of the function also.

Need of User Defined Functions or Advantages

- If we want to perform a task repeatedly then it is not to write a particular block of program again and again. Keep the block of statements in user defined functions.
- The function defined can be used for any number of times to perform the task.
- Using function the large programs can be reduced to smaller programs.
- It is easy to find out errors.

- It also increases the readability to the compiler.

Parts of User Defined Function or Steps for creating a user defined function

Every user defined function must have the following components.

1. Function prototype or Function Declaration.
2. Function calling or caller or Function Call.
3. Function Definition or Callee or Called Function.

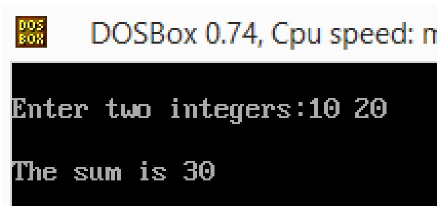
The following program gives a brief explanation on user defined functions.

Aim: To write a C program to illustrate the user defined functions in C

```
#include<stdio.h>

#include<conio.h>

int sum(int,int);

void main()

{

        int a,b,c;

        clrscr();

        printf("\nEnter two integers:");

        scanf("%d%d",&a,&b);

        c=sum(a,b);

        printf("\nThe sum is %d",c);

        getch();

}

int sum(int x, int y)

{

        int z;

        z=x+y;

        return z;
```

}

Output



```
Enter two integers:10 20

The sum is 30
```

1. **Function Proto type**

It is the declaration portion of the user defined function. It tells to the compiler about the following information.

- Return type of the function
- Name of the function
- Number of arguments of the function
- Type of arguments.

Syntax:

Return_type function_name(argument_list);

Example: int sum(int, int);

We can place this function prototype within the main() method (local declaration) or outside of the main() method(global declaration).

2. **Function calling**

The function calling takes the responsibility of sending the data from main() function to sub function.

The values that are entered in the calling function are send to called function.

Syntax:

Returned value= function_name(arguments list);

Example: c=sum(a,b);

3. **Called Function or Callee**

This is the actual user defined function., it will get the data from the calling functions, and do the necessary processing as per the instructions written by the programmer and also returns or of may not return that data to the calling functions.

Syntax:

```
Return_type function_name(argument_list)
{
        //body of the method
}
```

Example:

```
int sum(int x,int y)
{
        //Body of the function
        return 1;
}
```

Working of a Function

Actual Parameter

The arguments of calling function are called as Actual parameters. For example from the above structure the variables a,b are called as actual parameters.

Formal Parameters

The arguments of called functions are called as Formal Parameters. For example from the above structure the variables x and y are called as Formal parameters.

Arguments list

The argument list means the variable names enclosed within the parenthesis () and they must be separated by comma(,).

By default every user defined function return an integer value. If we donot require any return value we have to use the keyword “void”.

Q. Explain the types of user defined functions in C.

Depending on the return type of the method, the arguments of the method the user defined functions can be classified into four categories. They are

1. Function without return type and without arguments.
2. Function without return type and with arguments.

3. Function with return type and without arguments
4. Function with return type and with arguments.

1. A function without return type and without arguments

- In these functions neither the data is passed through the calling function nor the data is sent back from the called function.
- There is no data transfer between calling and called functions.
- The function is only executed and nothing is obtained.
- The function acts as independent. They read the data values and print the results in same blocks.
- These functions may be useful to print some message, draw a line and etc….

Syntax:

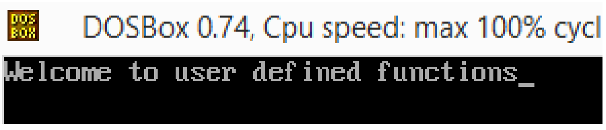
Function call or Function Calling	**Function analysis**	**Function definition or callee**

Void main()
{
    ------- 	
    ------- 	
    Function();
    -------- 	
    ---------- 	
}
	No arguments are passed and no values are sent back.	Void function()
		{
		  //body of the function;
		}

The following program illustrates the user defined function without return type and without arguments.

```
#include<stdio.h>

#include<conio.h>

void main()

{

        void msg();

        clrscr();

        msg();

        getch();
}
void msg()
{
        printf("Welcome to user defined functions");
}
```

Output


DOSBox 0.74, Cpu speed: max 100% cycl

```
Welcome to user defined functions_
```

2. A function without return type and with arguments

- In these functions the arguments are passed through the calling function.
- The called function operates on the received values.
- No result is sent back from called function to calling function.
- These functions are partially dependent on calling functions.

Syntax

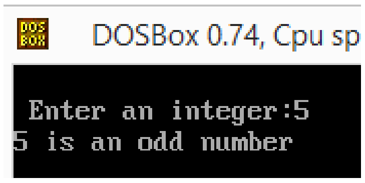
Function protocol	Function Analysis	Function Definition

Void function_name(arg_list);
Void main()
{
--------- 	
---------- 	
 	
Function_name(arg_list);
-------- 	
}
	Arguments are passed but no values are sent back.	Void function_name(arg_list)
		{
		//body of the function
		}

The following program illustrates the user defined function without return type and with arguments.

```
#include<stdio.h>
#include<conio.h>
void evenodd(int);
void main()
{
	int n;
	clrscr();
	printf("\n Enter an integer:");
	scanf("%d",&n);
	evenodd(n);
	getch();
}
void evenodd(int x)
{
	if(x%2==0)
		printf("%d is an even number",x);
	else
		printf("%d is an odd number",x);
}
```

Output



```
 Enter an integer:5
5 is an odd number
```

3.A function with return type and without arguments

- In these functions no arguments are passed through the calling functions.
- The called function returns a value to the calling functions.
- The called function is independent i.e., it reads values from the keyboard and returns the values.
- Here both calling and called functions are partially communicated with each other.

Syntax:

Function Prototype	Function Analysis	Function definition

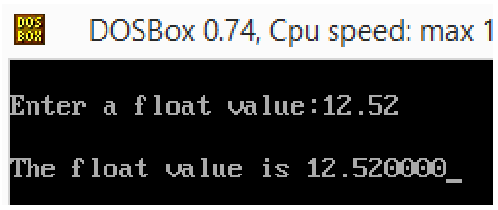
Return_type function_name();
Main()
{
--------- 	
----------- 	
C=function_name();
------- 	
------ 	
}
	No arguments are passed but a value is sent back.	Return_type function_name()
		{
		------- 	
		--------- 	
		Return value;
		}

The following program illustrates the use of a user defined function without arguments and with return type.

```
#include<stdio.h>

#include<conio.h>

float value();

void main()

{

        clrscr();

        printf("\nThe float value is %f",value());

        getch();

}

float value()

{

        float f;

        printf("\nEnter a float value:");

        scanf("%f",&f);

        return f;
```

}

Output:


DOSBox 0.74, Cpu speed: max 1

```
Enter a float value:12.52

The float value is 12.520000_
```

4.A function with arguments and with return type

- ➢ In such function the copy of the actual arguments are passed to the formal arguments.
- ➢ Value is written from callee to the function call.
- ➢ In these functions the data is transferred between calling and called functions i.e., the communication between the caller and callee is made.

Syntax:

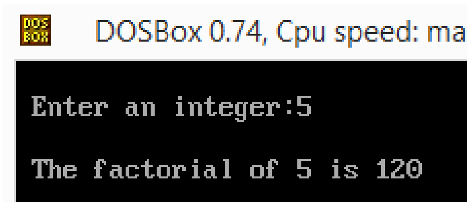
Function protocol	**Function Analysis**	**Function definition**

Return_type function_name(arg_list);
Void main()
{
------- 	
-------- 	
 	
C=function_name(arg_list);
------- 	
}
	Arguments are passed and the value is sent back.	Return_type function_name(arg_list)
		{
		----- 	
		------ 	
		------- 	
		Return value;
		}

The following program explains the use of user defined function with return type and with arguments.

```
#include<stdio.h>

#include<conio.h>

int factorial(int);

void main()

{
	int n,res;
	clrscr();
	printf("\n Enter an integer:");
	scanf("%d",&n);
	res=factorial(n);
	printf("\n The factorial of %d is %d",n,res);
	getch();
}
int factorial(int x)
{
	int fact=1,i;
	if(x==0)
	{
		return 1;
	}
	else
	{
		for(i=1;i<=x;i++)
		{
			fact=fact*i;
		}
		return fact;
	}
}
```

Output



```
Enter an integer:5

The factorial of 5 is 120
```

Functions with arrays

Explain how to pass the arrays as arguments to the functions

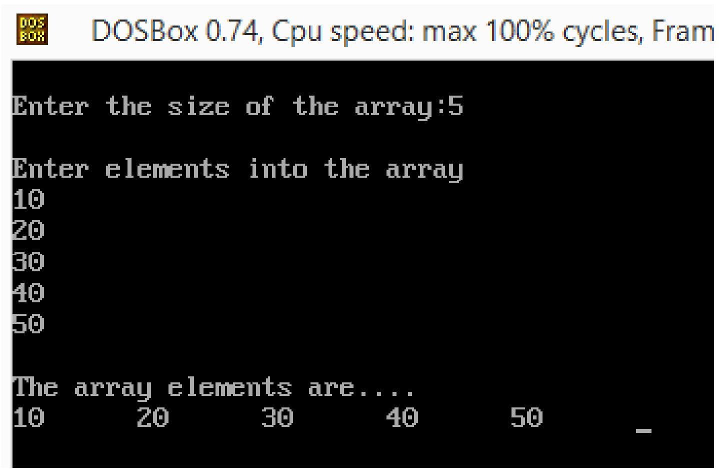
Some times we need to send more values at a time to the called function, in such type of cases the arrays are suitable to send as an argument to the called function.

To use the arrays as an argument to the functions we need to send the name of the arrays as an argument.

The following program illustrates the above concept.

```
#include<stdio.h>

#include<conio.h>

void array(int[],int);

void main()

{

        int a[10],n,i;

        clrscr();

        printf("\nEnter the size of the array:");

        scanf("%d",&n);

        printf("\nEnter elements into the array\n");

        for(i=0;i<=n-1;i++)

        {

                scanf("%d",&a[i]);

        }

        array(a,n);
        getch();
}
void array(int b[], int x)
{
        int i;
        printf("\nThe array elements are....\n");
        for(i=0;i<=x-1;i++)
        {
                printf("%d\t",b[i]);
        }
}
```

Output



```
Enter the size of the array:5

Enter elements into the array
10
20
30
40
50

The array elements are....
10      20      30      40      50      _
```

Define a recursive function and explain.

Recursive function:

Recursive function is the process of defining something in terms of itself. If a function is called by itself is known as a recursive function and this process is known as recursion.

There are two types of recursion functions in c.

- Direct recursion (Call by itself)
- Indirect recursion (A function is passed as an argument to another function).

Direct Recursion

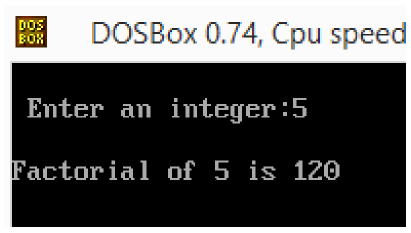
If a function is call by itself then it is known as a direct recursive function.

The following program illustrates the direct recursive function.

```
#include<stdio.h>

#include<conio.h>

int factorial(int);

void main()

{

        int n,f;

        clrscr();

        printf("\n Enter an integer:");

        scanf("%d",&n);

        f=factorial(n);

        printf("\nFactorial of %d is %d",n,f);

        getch();

}

int factorial(int n)

{

        int f;

        if(n==0)

                return 1;

        else

                return n*factorial(n-1);
```

}

Output



```
 Enter an integer:5

Factorial of 5 is 120
```

Indirect recursion

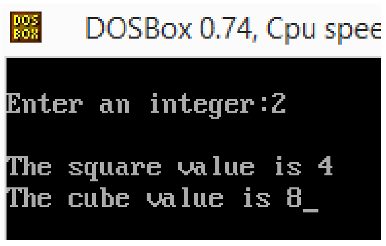
If a function is passed as an argument to another function, then it is called as indirect recursion and this process is known as indirect recursion.

We can pass not only the values, arrays and address as an argument to a function but we can also pass a function as an argument to another function.

The following program illustrate the use of indirect recursive function

```
#include<stdio.h>

#include<conio.h>

int square(int);

int cube(int,int);

void main()

{

        int n,sv,cv;

        clrscr();

        printf("\nEnter an integer:");

        scanf("%d",&n);

        sv=square(n);

        cv=cube(square(n),n);

        printf("\nThe square value is %d",sv);

        printf("\nThe cube value is %d",cv);
        getch();
}
int square(int x)
{
        return x*x;
}
int cube(int x,int y)
{
        return x*y;
}
```

Output



```
Enter an integer:2

The square value is 4
The cube value is 8_
```