Chapter No: 5
Functions

CHAPTER NO: 5	Functions
1	What is UDF? Explain its advantages.
2	Explain function declaration, definition & calling of function with example.
3	Explain different category of function.
4	Explain local & global variable with example.
5	Explain call by value & call by reference with example.

Q-1. what is UDF? Explain its advantages.

Answer:

- UDF – User Defined Function.
- Function which is developed by programmer is known as user defined function.
- To make programming easier, we break larger program into smaller sub -programs to perform a specific well defined task. These sub-programs are called functions.
- Function is group of statements that performs some specific task, which will be repeated in many programs.
- So to avoid writing code repeatedly we can create function and just need to call that function wherever it will used.
- Thus, it saves both time and space.

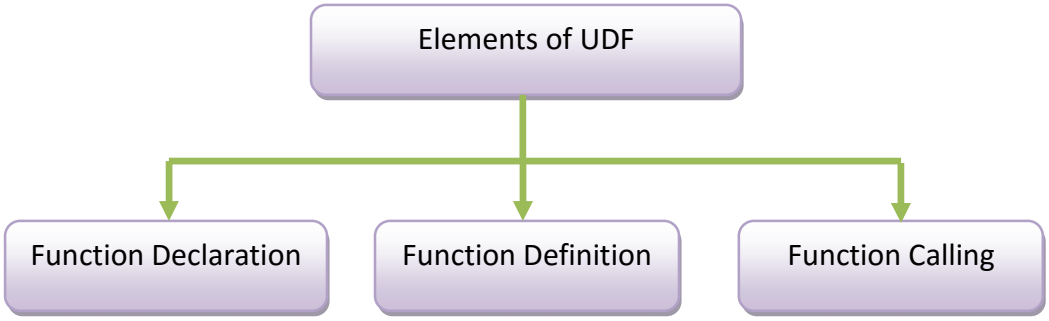
Advantages:

1. It reduces the program **complexity** and makes programming simple and easy to understand.
2. It **reduces the length** of the program code.
3. It provides **reusability** of the same program code.
4. It provides **ease of testing** of small program code.
5. It increases the **speed** of execution.

Q-2. Explain function declaration, definition & function calling.

Answer:

There are 3 elements of UDF.



1) Function Declaration:

Function declaration only defines the name of function, number of arguments, types of arguments and return type of that function.
Function declaration is only optional if we write function definition before the function calling.
Function declaration tells the compile about function name, types of arguments return type of that function.

Syntax:

```
<Return Type> <function name> (<Parameter list>);
```

Here, return type defines which type of value that a function can return at time of function calling.
Function name is the name of function that user want to crate.
Parameter list can define the name of parameter and type of parameter.

2) Function Definition:

Function definition provides the actual code or body of the function.
Whenever function is called the control is goes to function definition and statements inside function definition are executed.

Syntax:

```
<Return Type> <function name> (<Parameter list>)
{
        Statements;
        return (value);
}
```

Here, first line of function definition is known as function header, which is same as function declaration.
If function return the value then, return statement is compulsory at the end of function.

3) Function Calling:

To use a function, user will have to call that function to perform specific task.
When program calls a function then control is transferred to called function.
To call the function user simply need to write the name of function with no of arguments to function if available.

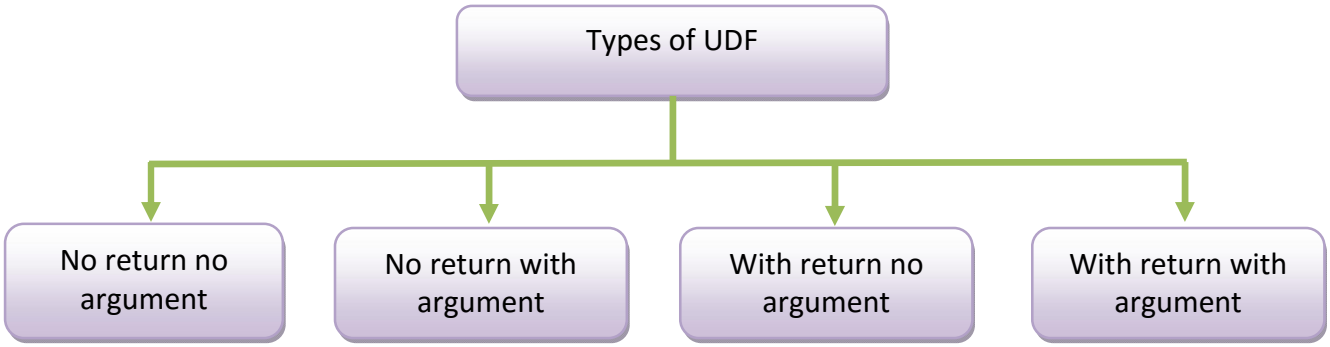
Syntax:
<function name> (<Parameter list>);

Example:

```
#include<stdio.h>
#include<conio.h>
void f();                   //Function declaration
void main()
{
        clrscr();
        f();                //Function calling
        getch();
}
void f()                    //Function definition
{
        printf("hello");
}
```

Q-3. Explain different category of function.
Or
Explain different types of user defined function.
Answer:

There are four different category of user define function



1) No Return No Argument:

This type of UDF does not receive any values from the calling function as well as it does not return any value.
The calling function does not receive any values from the called function that means there is no transfer of data between the calling and called function.

Syntax of declaration:

```
void <function name> ();
```

Syntax of definition:

```
void <function name> ()
{
        Statement(s);
}
```

Syntax of calling:

```
<Function name> ();
```

Example:

```
#include<conio.h>
#include<stdio.h>
void add();                    //Function declaration
void main()                    //Calling function
{
```

```
        clrscr();
        add();          //Function calling
        getch();
}
void add()              //Function Definition  (called function)
{
        int a,b,c;
        printf("enter a");
        scanf("%d",&a);
        printf("enter b");
        scanf("%d",&b);
        c=a+b;
        printf("ans is %d",c);
}
```

In example, as we know execution of C language program is always start from main().
So first line of main is clrscr() which clear the output screen.
Second line is to call the add(), which is user define function. So control goes to definition of add().
This function does not have any argument and return type of add() is void so does not return any value.

2) No Return with Argument:

In this, the function has arguments so called function receives some data from the calling function, but does not have return type so calling function does not receive any data from the called function.

Syntax of declaration:

```
void <function name> (<list of arguments>);
```

Syntax of definition:

```
void <function name> (<list of arguments>)
{
        Statement(s);
}
```

Syntax of calling:

```
<Function name> (<list of arguments>);
```

Example:

```
#include<stdio.h>
#include<conio.h>
void add(int,int);        //Function Declaration
void main()
{
        int a,b;
        printf("\nEnter value of a");
        scanf("%d",&a);
        printf("\nEnter value of b");
        scanf("%d",&b);
        add(a,b);         // Function Calling
}
void add(int x, int y)   //Function Definition
{
        int c;
        c = a+b;
        printf("\nAnswer is %d",c);
}
```

Here function declaration has two arguments of integer types. So whenever user calls the function need to pass two values.

3) With Return No Argument:

In this, the function has no arguments so called function does not receive any data from the *calling function*, but has return type so calling function receive data from the *called function.*

Syntax of declaration:

```
<Return Type> <function name> ();
```

Syntax of definition:

```
<Return Type> <function name> ()
{
        Statement(s);
}
```

Syntax of calling:

```
<Variable name> =<Function name> ();
```

Here Return type defines the data type of value return by the called function. It may be int, float and so on.
At the time of calling the function user need to create a variable of type of return type, so when calling function returns the value then that value is catch by the variable.

Example:

```
#include<stdio.h>
#include<conio.h>
int add();                  //Function Declaration
void main()
{
        int ans;
        ans=add();      //Function Calling
        printf("\nAnswer is %d",ans);
        getch();
}
int add()                   //Function Definition
{
        int a,b,c;
        a=10,b=20;
        c = a+b;
        return c;
}
```

In this example, add() return integer value and that value is stored into variable ans. Function **can return only one value** at a time and return statement is used to return the value at calling function.

4. With Return With Argument:

In this, the function has arguments so called function receives some data from the calling function and it also have return type so calling function also receives some data from the called function.

Syntax of declaration:

<Return Type> <function name> (<list of arguments>);

Syntax of definition:

```
<Return Type> <function name> (<list of arguments>)
{
        Statement(s);
}
```

Syntax of calling:

<Variable name> =<Function name> (<list of arguments>);

Example:

```
#include<stdio.h>
#include<conio.h>
int add(int,int);                    //Function Declaration
void main()
{
        int a,b,ans;
        a=10;
        b=20;
        ans=add(a,b);                //Function Calling
        printf("\nAnswer is %d",ans);
        getch();
}
int add(int x, int y)                //Function Definition
{
        int c;
        c = x+y;
        return c;
}
```

Q-4 Explain local & global variable with example.
Answer:

Local Variable:

- Variables that are declared inside a function or block are called local variables.
- They can be used only by statements that are inside that function or block of code.
- Local variables are not known to functions outside their own.
- When local variable is defined then it is not initialized automatically, you must initialize it yourself.
- When execution of block is start then variable is available and when the block ends then variable dies.

Global Variable:

- Variables that are declared outside a function or block are called global variables.
- Global variables can hold their values throughout the lifetime of program.
- Global variable can be accessed from any function of program.
- When you define global variable then it is automatically initialized by the system.

Example:

```
#include<stdio.h>
#include<conio.h>
int g; //Global declaration
int main ()
{
        int a, b; //Local Declaration
        a = 10;
        b = 20;
        g = a + b;
        printf ("value of a = %d, b = %d and g = %d\n", a, b, g);
}
```

Q-5 Explain call by value and call by reference with example.
Answer:

In c language we have two different methods to pass parameters in function.

1) Call by value
2) Call by reference

1) Call by value:

At the time of calling function if we pass the value as parameter to call any function then it is known as **call by value.**
When passing parameter as call by value then **original parameters values are copied** and then these copied values are passed to the function.
The parameter which is passed at the time of calling is known as **actual parameter**.
The parameter which is passed at the time of definition is known **as formal parameter**.
In call by value, actual parameters are copied into formal parameters and all the processes are done with formal parameters.

Example:

Example:

```
#include<stdio.h>
#include<conio.h>
void swap(int,int); //function declaration
void main()
{
        int a=10,b=5;
        clrscr();
```

```
        printf("\n value of a before swapping =%d",a);
        printf("\n value of b before swapping=%d",b);
        swap(a,b); //function call Actual Parameter
        printf("\n value of a after swapping=%d",a);
        printf("\n value of b after swapping=%d",b);
        getch();
}
void swap(int x,int y) //function definition Formal parameter
{
        int temp;
        temp=x;
        x=y;
        y=temp;
}
```

In this program, actual variables are a and b which are copied into formal variables x and y respectively.
Now all process is done with formal parameters, so no change in actual parameters.

2) Call By Reference:

At the time of calling function if we pass the reference as parameter to call any function then it is known as **call by reference.**
When passing parameter as call by reference then **original parameters addresses are copied** and then these copied values are passed to the function.
So, to store address we need pointer variable as formal arguments. We pass the address of actual parameters so all the process are done with actual parameters.

Example:

```
#include<stdio.h>
#include<conio.h>
void swap(int *,int *); //function declaration
void main()
{
        int a=10,b=5;
        clrscr();
        printf("\n value of a before swapping =%d",a);
        printf("\n value of b before swapping=%d",b);
        swap(&a,&b); //function call Actual Parameter
```

```
        printf("\n value of a after swapping=%d",a);
        printf("\n value of b after swapping=%d",b);
        getch();
}
void swap(int *x,int *y) //function definition Formal parameter
{
        int temp;
        temp=*x;
        *x=*y;
        *y=temp;
}
```