Unit-V

Functions & File Handling

Syllabus: Introduction to functions, function declaration and definition, function call, Return Types and arguments, Modifying parameters inside function using pointers, arrays as parameters. Scope & life time of variables, Basics of file Handling

Function: A function is a block of code (or) group of statements developed in a program to perform a specific task. a function can be run only when it is called. functions can be defined only once in a program but can be use it many times. So they are important for reusing code.

functions are classified into 2 types. They are

(i) Predefined

(ii) User defined

> we have seen predefined functions like printf(), scanf(), gets(), puts(), malloc(), calloc(), main() and so on. all these are stored in some header files like stdio.h stdlib.h, string.h etc...

> Userdefined functions are defined by the user. for user defined functions the user has to take care on 3 aspects. They are

1. function declaration
2. function definition
3. function calls.

Function declaration / Prototype :-

> A function declaration tells the compiler that there is a function with the given name defined somewhere else in the program
> In a function declaration we must provide function name, its return type, and the no. of parameters and their type
> If you define a function after main function then we must declare a function at the declaration section of the program or simply before main function

Syntax:

returntype functionname (type1 parameter1, type2 parameter2);

The parameter names are not mandatory while declaring a functions. but types of parameters are important to include in the declaration
> Semicolon is mandatory in function declaration.

Examples: int sum (int a, int b);

int sum (int , int); ⟵ ending semicolon

↙ function ↳ type of parameter
returntype name

Note: A function must always be declared globally before calling it.

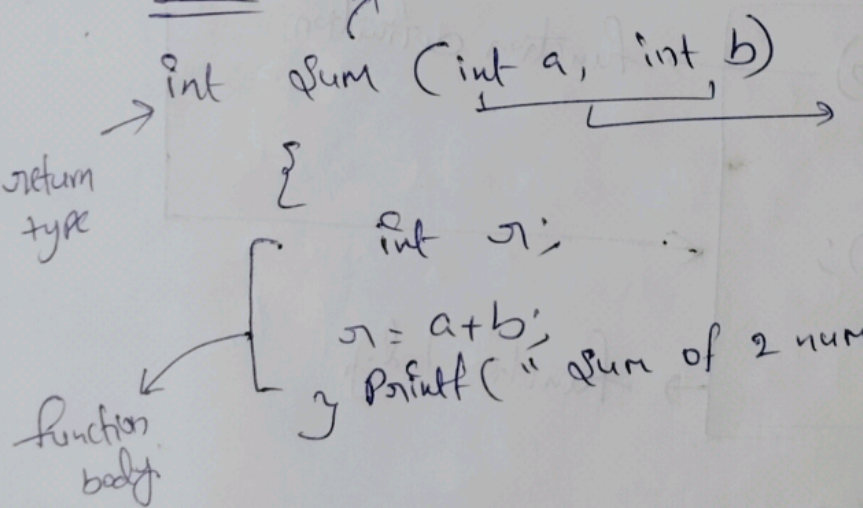
Function definition:

The function definition consists of actual statements that are executed when the function is called

Syntax:

```
returntype functionname (type1 parameter1 , type2 parameter2
                         ---- )
{
    statements;  // function body
}
```

Example:



The body of the function will get executed only when function is called

Function calls: A function call is a statement that instructs the compiler to execute the function

> we use function name, & parameters in the function call.

Note: function call is necessary for the execution of the function. if function is not called, the function statements will not be executed.

A simple Program:

```
#include <stdio.h>
int sum(int a, int b);          → function declaration
int main()
{
    int add = sum(10,30);
    printf("sum is :%d\n", add);
    return 0;
}
int sum(int a, int b)           → function definition
{
    retur (a+b);
}
```

function calling

function Returning value

function body

o/p: sum is : 40.

Working of the above Program:

In main function the 1st stmt is function call. so the sum function is called and 10, 30 are passed as arguments to sum function.

So, the control will execute the function the function returns a+b to the function call. main function the result is assigned to "add" variable

function Returntype:

function return type tells what type of value is returned after all function is executed. when we dont want to return a value, we can use the void datatype

Example:
```
int fun (parameter1 , parameter2) :
↑
returntype
```

The above function will return an integer value after running statements inside the function.

Note: Only one value can be returned from C function To return multiple values, we have to use pointers or structures.

Arguments:

function arguments (also known as function parameters) are the data that is passed to a function.

> The arguments passed in the function call is called actual parameters (or) Arguments.

> The arguments passed in the function definition is called formal parameters.

Call by value and Call by reference :-

There are 2 ways to pass the data as arguments into the function in C language They are (i) Call by value (ii) Call by reference

Call by value :

> In Call by value method, the value of the actual parameters is copied into the formal parameters. In other words, we can say that the value of the variable is used in the function call in the call by value method

> In this method, we can not modify the value of the actual parameter by the formal parameter

> In Call by value, different memory is allocated for actual and formal parameters since the value of the actual parameter is copied into the formal parameter.

> The actual parameter is the argument which is used in the function call where as formal parameter is the argument which is used in the function definition.

> In the below example program before

```
#include <stdio.h>
void swap (int , int);
int main()
    {
        int a=10, b=20;
Printf (" Before swapping the values in main a=%d,
                b=%d \n", a, b);
    swap (a, b);
Printf (" After swapping values in main a=%d, b=%d
    \n", a, b);
}
void swap (int a, int b)
    {
        int temp;
      temp = a;
        a = b;
        b = temp;
Printf (" After swapping values in function a=%d, b=%d
                \n", a, b);
}
```

O/P: Before swapping the values in main a=10, b=20
After swapping the values in function a=20, b=10
After swapping values in main a=10, b=20