

Unit -4

Introduction to Functions

Functions

- A function is a block of statements that performs a specific task.
- C allows you to define functions according to your need. These functions are known as **user-defined functions**.
- Why functions are used:
 - To improve the readability of code.
 - Improves the reusability of the code, same function can be used in any program rather than writing the same code from scratch.
 - Debugging of the code would be easier if you use functions, as errors are easy to be traced.
 - Reduces the size of the code, duplicate set of statements are replaced by function calls.

Functions

- **Types of functions**
- **Predefined standard library functions:**
 - Standard library functions are also known as built-in functions. Functions such as `puts()`, `gets()`, `printf()`, `scanf()` etc are standard library functions.
 - For example, `printf()` function is defined in `<stdio.h>` header file so in order to use the `printf()`
- **User Defined functions:**
 - The functions that we create in a program are known as user defined functions or in other words you can say that a function created by user is known as user defined function.

Functions

- **Advantages of Functions**

- **Reusability of code.** Functions once defined can be used any several times. You can use functions of one program in another program. It saves time and effort.
- **Functions provides abstraction.** To use any function you only need is name and arguments it accepts. You need not to know how it works internally. For example - You have used `printf()` function hundreds of time.
- **Function allows modular design of code.** We can divide program into small modules. Modular programming leads to better code readability, maintenance and reusability.
- **It is easier to write programs using functions.** You can write code for separate task individually in separate function.
- **Code maintenance and debugging is easier.** In case of errors in a function, you only need to debug that particular function instead of debugging entire program.

Functions

- **Function is divide in 3 parts**
 - **1. Function declaration/function prototype**
 - **2. Function defination**
 - **3. Function calling**

Function declaration/prototype

- Like variable declarations, functions are also declared.
- Function declaration tells the compiler that there exists a function with some name that may be used later in the program.
- **Function declaration is also known as function prototype or function signature.**
- **Syntax :** `return_type function_name(parameter list);`
 - **Return type** - Return type defines the data type of value returned by the function.
 - A function does some calculation and may return a resultant value. So that the result of one function can be used by another function. For example - `sqrt()` function returns square root of given number.
 - You must mention return type as `void` if your function does not return any value.

Function declaration/prototype

- **Function name** - Function name is a valid C identifier that uniquely identifies the function. Follow variables naming rules while naming a function.
- **Parameter list** - A function may accept input. Parameter list contains input type and variable name given to the function. Multiple inputs are separated using comma ,.
- A function declaration must be terminated using **semicolon ;**. You can declare a function anywhere in the program. However it is best practice to declare functions below header files.
- **Important note:** Function declaration is optional. You can directly define and call a function skipping its declaration.

Function declaration/prototype

- For the above defined function `max()`, the function declaration is as follows
- `int max(int num1, int num2);`
- Parameter names are not important in function declaration only their type is required, so the following is also a valid declaration –
- `int max(int, int);`

Function Defination

- Function definition contains block of code to do task that the function is intended to do. Block of code must contain a **return** statement if return type of the function is not **void**.
- **Syntax of function definition**

```
return_type function_name(parameter list)
```

```
{
```

```
    // Function body
```

```
}
```

- Function declaration and definition syntax must be same.

Function Defination

- A function definition in C programming consists of a function header and a function body. Here are all the parts of a function –
- **Return Type** – A function may return a value. The `return_type` is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the `return_type` is the keyword `void`.
- **Function Name** – This is the actual name of the function. The function name and the parameter list together constitute the function signature.
- **Parameters** – A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
- **Function Body** – The function body contains a collection of statements that define what the function does.

Function Calling

- **Function calling**
- The most important part of function is its calling. Calling of a function in general is execution of the function. It transfers program control from current function to called function. We can optionally pass input to the calling function.
- **Syntax of function call:** `function_name(parameter_list);`
- **Function name** - Name of the function to execute.
- **Parameter list** - Comma separated input given to the function. Parameter list must match type and order mentioned in function declaration. Leave parameter list as blank if function does not accept any input.