# iMSCIT

# SEMESTER:I

## 1601102

## PROBLEM SOLVING THROUGH PROGRAMMING

## UNIT: IV

# Unit 4: Introduction to Functions and Types of Function

- **Functions**
  - Definition of a Function
  - Declaration of Function
  - Function Prototype
- **Types of Functions**
  - Function with no parameter and no return values
  - Function with no parameter and return values
  - Function with parameter and return values
  - Function with parameter and no return values
- **• Recursive Functions**

# FUNCTIONS

- **In c, we can divide a large program into the basic building blocks known as function.**
- The function contains the set of programming statements enclosed by { } .
- A function can be called multiple times to provide re-usability and modularity to the program.
- In other words, we can say that the collection of functions creates a program.
- **The function is also known as procedure or subroutine in other programming languages**
- **Every C program has at least one function, which is main(),** and all the most trivial programs can define additional functions.

# FUNCTIONS

- **A function is a block of code that performs a specific task.**
- C allows you to define functions according to your need**.** These functions are known **as user-defined functions.**
- **Function is divide in 3 part**
  - Function declaration/function prototype
  - Function definition
  - Function calling.

# Advantage of functions in C

- By using functions, we can avoid rewriting same logic/code again and again in a program.
- We can call C functions any number of times in a program and from any place in a program.
- We can track a large C program easily when it is divided into multiple functions.
- Re-usability is the main achievement of C functions.

# Function Aspects

**There are three aspects of a C function.**

- **Function declaration:** A function must be declared globally in a c program to tell the compiler about the function name, function parameters, and return type.
- **Function call:** Function can be called from anywhere in the program. The parameter list must not differ in function calling and function declaration. We must pass the same number of functions as it is declared in the function declaration.
- **Function definition:** It contains the actual statements which are to be executed. It is the most important aspect to which the control comes when the function is called. Here, we must notice that only one value can be returned from the function

# Function Declaration/Prototype

- A function declaration tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.
- Syntax :

  **return_type function_name( parameter list );**

- For example in 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 Definition

- It contains the actual statements which are to be executed. It is the most important aspect to which the control comes when the function is called. Here, we must notice that only one value can be returned from the function.
- Syntax:

```
return_type function_name(data_type parameter...)
{
//code to be executed
}
```

# Function Definition

- 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.

# Types of Functions

- A function may or may not accept any argument. It may or may not return any value.
- Based on these facts, There are four different types of functions.

1. **function without arguments and without return value**
2. **function without arguments and with return value**
3. **function with arguments and without return value**
4. **function with arguments and with return value**

# Types of Functions

- A function may or may not accept any argument. It may or may not return any value.
- Based on these facts, There are four different types of functions.

1. **function without arguments and without return value**
2. **function without arguments and with return value**
3. **function with arguments and without return value**
4. **function with arguments and with return value**

# Function without arguments and without return value

- In this type user has no need to pass arguments as well as return value.
- **Declaration :**

```
void function_name();
// declaration
```

- **Definition:**

```
void function_name()
{
//function body
}
```

- **Calling:**

```
function_name();
```

# Function without arguments and without return value

- **Example:**

```
#include <stdio.h>
void printName();
void main ()
{
    printf("Hello ");
    printName();
}
void printName()
{
    printf("Javatpoint");
}
```

# function without arguments and with return value

- **Declaration :**

  datatype function_name();

  // declaration

- **Definition:**

  datatype function_name() //definition

  {

  //function body

  }

- **Calling**:

  variable = function_name();

# function without arguments and with return value

- **Example:**

```
#include<stdio.h>
int sum();
void main()
{
    int result;
    printf("\nGoing to calculate the sum of two numbers:");
    result = sum();
    printf("%d",result);
}
```

# function without arguments and with return value

```
int sum()
{
    int a,b;
    printf("\nEnter two numbers");
    scanf("%d %d",&a,&b);
    return a+b;
}
```

# function with arguments and without return value

- **Declaration :**

  void function_name(datatype, datatype...); // declaration

- **Definition:**

  void function_name(datatype, datatype)

  {

  //function body

  }

- **Calling:**

  function_name(parameter);

# function with arguments and without return value

- **Example:**

```
#include<stdio.h>
void sum(int, int);
void main()
{
    int a,b,result;
    printf("\nGoing to calculate the sum of two numbers:");
    printf("\nEnter two numbers:");
    scanf("%d %d",&a,&b);
    sum(a,b);
}
```

# function with arguments and without return value

```
void sum(int a, int b)
{
printf("\nThe sum is %d",a+b);
}
```

# function with arguments and with return value

- **Declaration :**

  datatype function_name(datatype parameter,datatype parameter..);

  // declaration

- **Definition:**

  datatype function_name(datatype parameter,datatype parameter..)

  {

  //function body

  }

- **Calling:**

  variable=function_name(parameter);

# function with arguments and with return value

- **Example:**

```
#include<stdio.h>
int sum(int, int);
void main()
{
    int a,b,result;
    printf("\nGoing to calculate the sum of two numbers:");
    printf("\nEnter two numbers:");
    scanf("%d %d",&a,&b);
    result = sum(a,b);
    printf("\nThe sum is : %d",result);
}
```

# function with arguments and with return value

```
int sum(int a, int b)
{
return a+b;
}
```

# Recursive Function

- Recursion is the process which comes into existence when a function calls a copy of itself to work on a smaller problem. **Any function which calls itself is called recursive function**, and **such function calls are called recursive calls.**
- Recursion involves several numbers of recursive calls. However, it is important to impose a termination condition of recursion. Recursion code is shorter than iterative code however it is difficult to understand.
- Recursion cannot be applied to all the problem, but **it is more useful for the tasks that can be defined in terms of similar sub tasks.**
- For Example, **recursion may be applied to sorting, searching, and traversal problems.**

# Recursive Function

```c
#include <stdio.h>
int fact (int);
int main()
{
   int n,f;
   printf("Enter the number whose factorial you want to calculate?");
   scanf("%d",&n);
   f = fact(n);
   printf("factorial = %d",f);
}
int fact(int n)
{
   if (n==0)
   {
      return 0;
   }
   else if ( n == 1)
   {
      return 1;
   }
   else
   {
      return n*fact(n-1);
   }
}
```

# THANK YOU