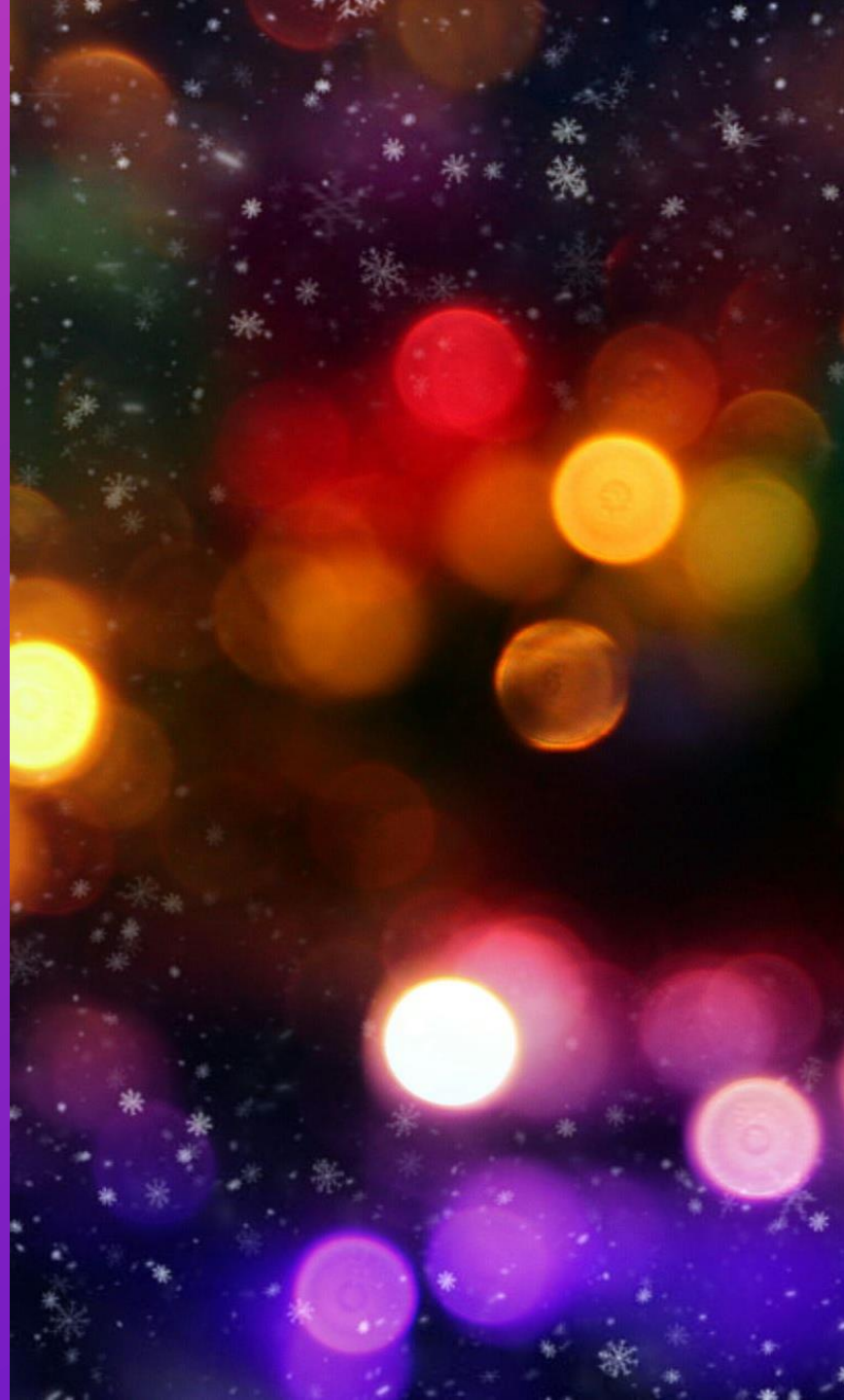
MNNIT COMPUTER CODING CLUB

CLASS-5

BASICS OF C



FUNCTIONS IN C

A function is a group of statements that together perform a task. Every C program has at least one function, which is **main()**, and all the most trivial programs can define additional functions.

You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division is such that each function performs a specific task.

Advantages of using functions -

- Reusability of code, avoids repetition of code
- Makes program easier to modify and debug in case of errors
- Enhances readability of code, a big code is difficult to read

DEFINITION OF A FUNCTION

The general form of a function definition in C programming language is as follows –

```
return_type function_name  (parameters){
        body of the 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. This value is referred to as actual parameter or argument. 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.

EXAMPLE

A program to calculate the sum of digits of a number WITHOUT USING function

```c
#include <stdio.h>

int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    int digitSum = 0;
    while(num > 0)
    {
        digitSum += num % 10;
        num /= 10;
    }
    printf("Sum of digits = %d", digitSum);
    return 0;
}

```

EXAMPLE

A program to calculate the sum of digits of a number USING function

```c
#include <stdio.h>

// Method to calulate sum of digits a number
int sumOfDigits(int num)
{
    int sum = 0;
    while(num > 0)
    {
        sum += num % 10;
        num /= 10;
    }
    return sum;
}

int main() {
    int i, digitSum;
    for(i = 1; i <= 100; i++)
    {
        digitSum = sumOfDigits(i);
        printf("Sum of digits of number %d = %d\n", i, digitSum);
    }
    return 0;
}

```

DECLARATION OF FUNCTION

Here the function *sumOfDigits()* is written before *main()*, so *main()* knows everything about the function *sumOfDigits()*. But generally, the function *main()* is placed at the top and all other functions are placed after it. In this case, function declaration is needed. The function declaration is also known as the function prototype, and it informs the compiler about the following three things -

1. Name of the function.
2. Number and type of arguments received by the function.
3. Type of value returned by the function.

A function declaration has the following parts –

```
return_type function_name ( parameters );
```

For the above defined function *sumOfDigits()*, the function declaration is as follows –

```
int sumOfDigits(int num);
```

Parameter names are not important in function declaration only their type is required, so the following is also a valid declaration –

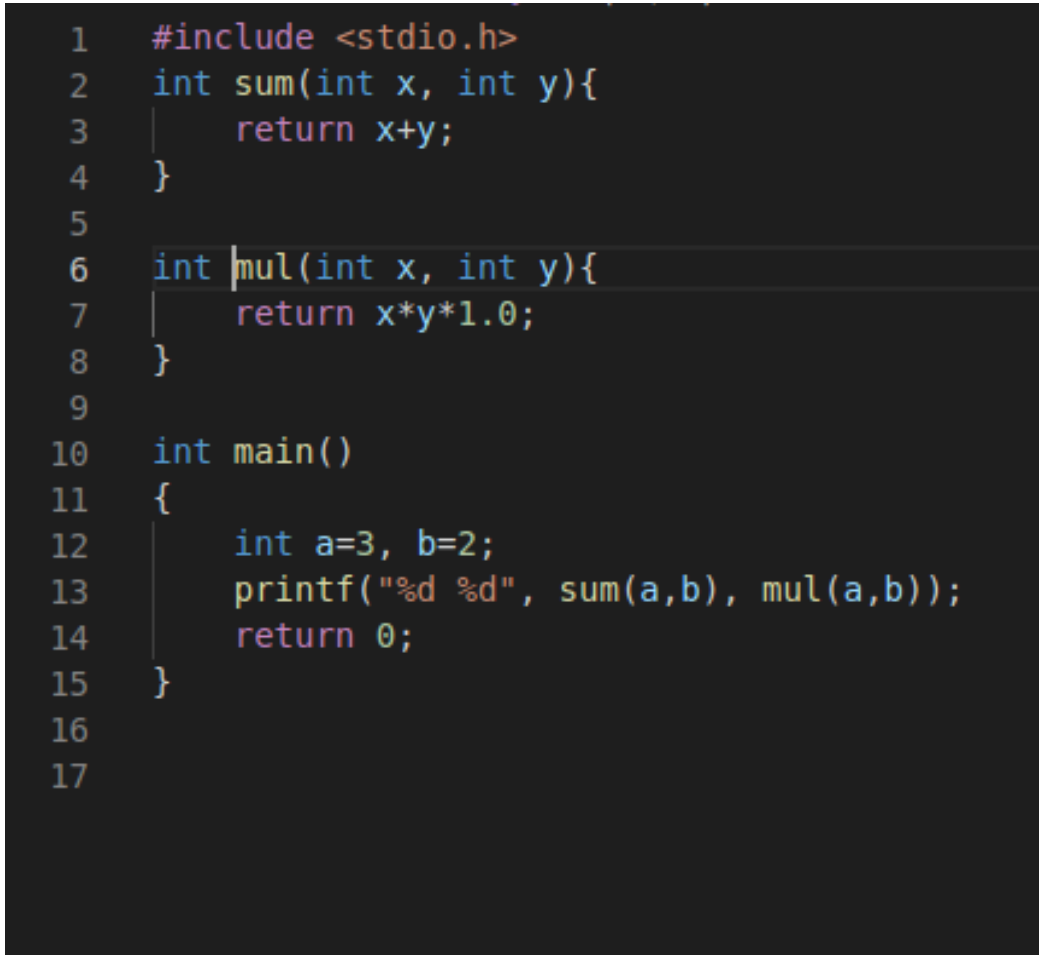
```
int sumOfDigits(int);
```

CALLING

- Function definition tells what it does and to use the function you have to call it.
- The control and current line shift to the function
- Returned value can either be used or stored in different variable
- Calling of functions is done on stack (Ignore for now)
- If a function is not called, it won't execute on its own.
- In previous example *sumOfDigits()* is referred as **called function** and *main()* is the **calling function**

ARGUMENTS

- Actual arguments: The arguments which are mentioned in the function call.
- Formal arguments: The name of the arguments, which are mentioned in the function definition are called formal or dummy arguments
- The order, number and type of actual arguments in the function call should match with the order, number and type of formal arguments in the function definition



TYPES OF USER DEFINED FUNCTIONS

The functions can be classified into four categories on the basis of the arguments and return value

1. Functions with arguments and a return value
2. Functions with arguments and no return value
3. Functions with no arguments and a return value
4. Functions with no arguments and no return value

```c
#include <stdio.h>

// Function with arguments, and a return value
int add(int a, int b)
{
    int sum = a + b;
    return sum;
}

int main()
{
    int a, b;
    printf("Enter first number: ");
    scanf("%d", &a);
    printf("Enter second number: ");
    scanf("%d", &b);
    printf("Sum of %d and %d = %d", a, b, add(a, b));
    return 0;
}
```

FUNCTION WITH ARGUMENT, AND A RETURN VALUE

```c
#include <stdio.h>

// Function with arguments, and no return value
void add(int a, int b)
{
    int sum = a + b;
    printf("Sum of %d and %d = %d\n", a, b, sum);
}

int main()
{
    int a, b;
    printf("Enter first number: ");
    scanf("%d", &a);
    printf("Enter second number: ");
    scanf("%d", &b);
    add(a, b);
    return 0;
}
```

FUNCTION WITH ARGUMENT, AND NO RETURN VALUE

```c
#include <stdio.h>

// Function with no arguments, and a return value
int add()
{
    int a, b;
    printf("Enter first number: ");
    scanf("%d", &a);
    printf("Enter second number: ");
    scanf("%d", &b);
    return a + b;
}

int main()
{
    int sum = add();
    printf("Sum = %d\n", sum);
    return 0;
}
```

FUNCTION WITH NO ARGUMENT, AND A RETURN VALUE

```c
#include <stdio.h>

// Function with no arguments, no return value
void add()
{
    int a, b;
    printf("Enter first number: ");
    scanf("%d", &a);
    printf("Enter second number: ");
    scanf("%d", &b);
    int sum = a + b;
    printf("Sum of %d and %d = %d", a, b, sum);
}

int main()
{
    add();
    return 0;
}
```

FUNCTION WITH NO ARGUMENT, NO RETURN VALUE

MAIN

- Execution of every C program always begins with the function main().
- Every function is called in main() and after execution control is given back to main()
- The name, number and type of arguments are predefined in the language.
- main() returns 0 on successful execution and nonzero (garbage value) when error.
- The definition, declaration and call of main() function
- Function Declaration - By the C compiler
- Function Definition - By the programmer
- Function Call - By the operating system

LIBRARY FUNCTIONS

1. The definition, declaration and call of library functions-
    1. Function Definition - Predefined, precompiled and present in the library.
    2. Function Declaration. - In header files (files with a .h extension)
    3. Function Call - By the programmer

2. To use a library function in our program we should know-
    1. Name of the function and its purpose
    2. Type and number of arguments it accepts
    3. Type of the value it returns
    4. Name ,of the header file to be included.

3. We can deflne any function of our own with the same name as that of any function in the C library. If we do so then the function that we have defined will take precedence over the library function with the same name.

MATH.H EXAMPLE

How to compile a code
of math.h?
gcc code.c -lm