

# Arrays

FYCS-UNIT-2

# What are Arrays?

- Arrays are a kind of data structure that can store a fixed-size sequential collection of elements of the same type.
- An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.
- Instead of declaring individual variables, such as `number0`, `number1`, ..., and `number99`, you declare one array variable such as `numbers` and use `numbers[0]`, `numbers[1]`, and ..., `numbers[99]` to represent individual variables.
- A specific element in an array is accessed by an index.
- In array, functions like Sorting, deletion, insertion, searching and traversing can be done.

# Cont.

- All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

40	55	63	17	22	68	89	97	89
0	1	2	3	4	5	6	7	8

<- Array Indices

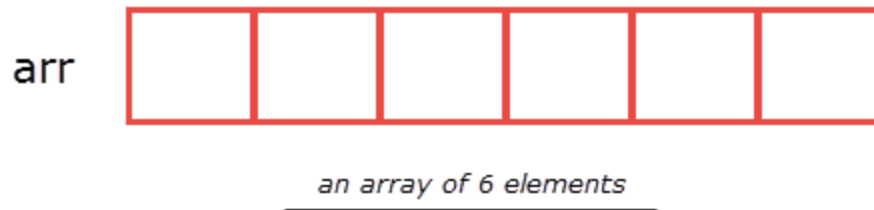
**Array Length = 9**

**First Index = 0**

**Last Index = 8**

# One-Dimensional Arrays

- Conceptually you can think of a one-dimensional array as a row, where elements are stored one after another.
- Syntax: `datatype array_name[size];`
- Here, size is the number of elements that array holds.



# Declaring Array

- The general form of declaring an array is:

Datatype array\_name[size];

- Here are some type specifies the data type of elements contained in the array such as int, float, or char.
- The size of the array must be a numeric constant or a symbolic constant.
- **For e.g.: int group[5];**
- Here, int is the type and group is the array name and 5 is the size of the array and the subscripts(index) starts from 0 to 4.

# Initializing an array

- After array has declared the array must initialized.
- In C, arrays are initialized in two ways i.e:

1. Compile time: The arrays are initialized by,

```
int arr[5]={2,4,5,6,8};
```

2. Run time: The arrays are initialized by looping statements.

```
for(int i=0;i<5;i++)  
{  
  //statements  
}
```

# Accessing array

## Accessing from the declaring

```
#include <stdio.h>
int main() {
    int a[5]={20,30,40,50,60};
    for(int i=0;i<5;i++)
    {
        printf("\nElement of array[%d] is:%d",i,a[i]);
    }
    return 0;
}
```

## Accessing from user input

```
#include<stdio.h>
int main()
{
    int arr[5];
    int i;
    printf("\n Enter the array elemnts : ");
    for(i = 0; i<5; i++)
    {
        scanf("%d", &arr[i]);
    }
    printf("\n The array elemnts are : ");
    for(i = 0; i<5; i++)
    {
        printf(" %d ", arr[i]);
    }
    return 0;
}
```

# Two Dimensional Array

- An array of arrays is known as 2D array. The two dimensional (2D) array in C programming is also known as **matrix**. A matrix can be represented as a table of rows and columns.
- Syntax: `datatype array_name[][];`
- Declaring 2D-array: `int arr[][];`
- **Initializing** 2D-array: `int arr[5][3]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};`



# Accessing 2D-Array example

```
#include <stdio.h>
int main() {
    int my[4][2] = {
        {1, 2},
        {4, 5},
        {7, 8},
        {10, 11}
    };
    // accessing and printing the elements
    for ( int i = 0; i < 4; i++ )
    {
        for ( int j = 0; j < 2; j++ ) {
            // variable j traverses the columns
            printf("my [%d][%d] = %d\n", i, j, my[i][j]);
        }
    }
    return 0;
}
```

# Array types Vs List & Tuple in Python

- Just like arrays in C, lists and tuples are used to store elements in Python.
- List are just like the arrays, declared in C language. These are mutable in nature.
- The list stores elements using [] brackets.

Syntax: `list_data=['a', 'my', 'our']`

- Tuple is also a sequence data type that can contain elements of different data types, but these are immutable in nature.
- The tuple stores elements using () brackets.

Syntax: `t=('a', 'my', 'our')`

# Difference

List	Tuple
1) List is a Group of Comma separated Values within Square Brackets and Square Brackets are mandatory. Eg: i = [10, 20, 30, 40]	1) Tuple is a Group of Comma separated Values within Parenthesis and Parenthesis are optional. Eg: t = (10, 20, 30, 40) t = 10, 20, 30, 40
2) List Objects are Mutable i.e. once we creates List Object we can perform any changes in that Object. Eg: i[1] = 70	2) Tuple Objeccts are Immutable i.e. once we creates Tuple Object we cannot change its content. t[1] = 70 → ValueError: tuple object does not support item assignment.
3) If the Content is not fixed and keep on changing then we should go for List.	3) If the content is fixed and never changes then we should go for Tuple.
4) List Objects can not used as Keys for Dictionries because Keys should be Hashable and Immutable.	4) Tuple Objects can be used as Keys for Dictionries because Keys should be Hashable and Immutable.

# Character input/output Functions

- The C has built-in statements for input and output.
- Character input functions is used to input a single character. All these functions are available in 'stdio.h' header file.
- The I/O library functions are listed the header file <stdio.h>
- These are also be called as **Unformatted** functions. Input and output functions in C are:

**getchar()**

**putchar()**

# Input functions

- **Unformatted input functions** used to read the input from the keyboard by the user accessing the console. These input values could be of *any primitive data type* or an array type. Some of the most important unformatted console input functions are –
- **getch()**: Reads a *single* character from the user at the console, ***without*** echoing it.
- **getche()**: Reads a *single* character from the user at the console, *and* echoing it.
- **getchar()**: This function reads a character-type data from standard input.
- **gets()**: Reads a *single* string entered by the user at the console.
- Syntax: gets(string\_variable);

# Example

```
#include<stdio.h>
main(){

    printf("This is getch enter the value\n");
    printf("%c",getch()); // printed as U write

    printf("\nThis is getchar enter the value\n");
    printf("%c",getchar()); //need to press enter key

    printf("\nThis is getche enter the value\n");
    printf("%c",getche()); // printed twice as U write

}
```

Output:

```
This is getch enter the value
i
This is getchar enter the value
a
a
This is getche enter the value
mm
```

# Output functions

- **Unformatted output functions** used to display the output to the user at the console. These output values could be of *any primitive data type* or an array type. Some of the most important unformatted console output functions are –
- **puts()**: Displays a *single* string's value at the console.

Syntax: puts(string\_variable);

- **putchar()**: Prints a *single* character value at the console.

Syntax: putchar(character\_variable);

# gets() function

- The gets() and puts() are declared in the header file stdio.h. Both the functions are involved in the input/output operations of the strings.
- C gets() function

The gets() function enables the user to enter some characters followed by the enter key. All the characters entered by the user get stored in a character array. The null character is added to the array to make it a string. The gets() allows the user to enter the space-separated strings. It returns the string entered by the user.

Syntax: `ch[] gets(ch[]);`



# puts() function

- The puts() function is very much similar to printf() function.
- The puts() function is used to print the string on the console which is previously read by using gets() or scanf() function.
- The puts() function returns an integer value representing the number of characters being printed on the console.
- Since, it prints an additional newline character with the string, which moves the cursor to the new line on the console, the integer value returned by puts() will always be equal to the number of characters present in the string.

Syntax: `int puts(char[])`

# Cont.

- **getc():** It is used to read a character from a file that has been opened. `getc(stdin)` make it read from a file presented by a standard input device.

Syntax: `variable=getc(file pointer);`

- **putc():** It is used to write a character to the file that has been opened.

Syntax: `putc(character, file pointer);`

- The `getc()` and `putc()` functions are analogues to `getchar()` and `putchar()` functions that handles one character at a time.

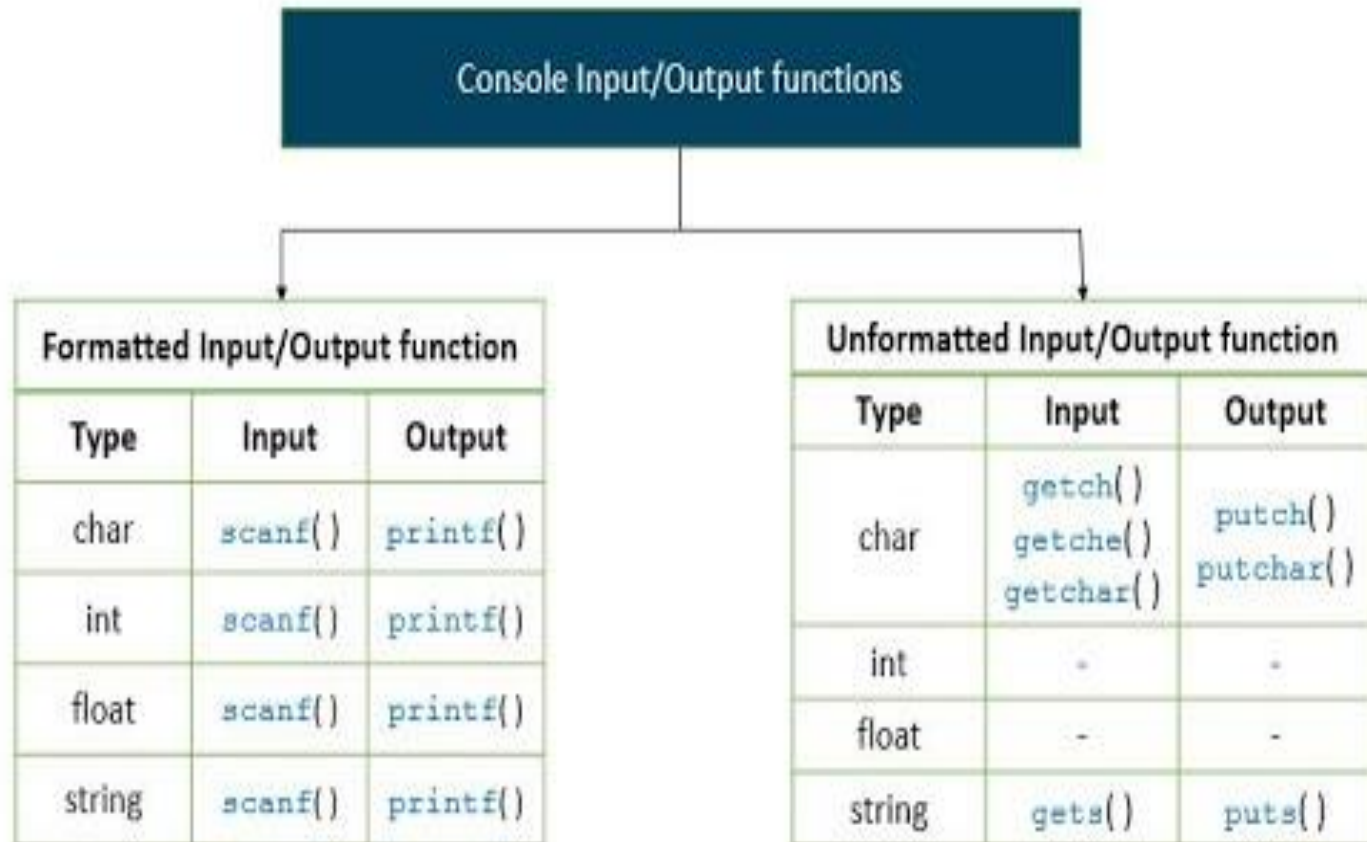
- Example:

```
char c;
```

```
c=getc(stdin);
```

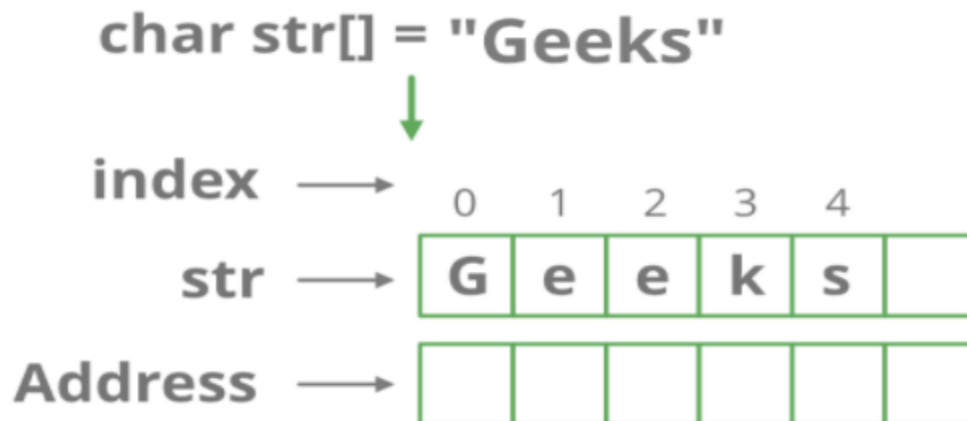
```
putc(c, stdout);
```

# Cont.



# Strings

- Strings are array of characters i.e. they are characters arranged one after another in memory. Thus, a character array is called string.
- Each character within the string is stored within one element of the array successively.
- The null or string-terminating character is represented by another character escape sequence, `\0`.
- For example, "hello students" is a string constant.



# Declaring

- A string variable is declared as an array of characters.
- Syntax:

`char string_name[size];`

- E.g. `char s[50];`
- When the compiler assign a character string to a character array, it automatically supplies a null character(‘\0’) at the end of the string.

# Initializing String Variables

- Strings are initialized in either of the following two forms:

```
char name[4]={'R','A','M', '\0'};
```

```
char name[]={'R','A','M', '\0'};
```

OR

```
char name[4]="RAM";
```

```
char name[]="RAM";
```

R	A	M	\0
name[0]	name[1]	name[2]	name[3]

- When we initialize a character array by listing its elements, the null terminator or the size of the array must be provided explicitly.

# Using scanf() function

- An alternative method for the input of strings is to use scanf() with the %c conversion which may have a count associated with it.
- This conversion does not recognize the new-line character as special.
- The count specifies the number of characters to be read in.

```
#include <stdio.h>
int main(){
    char name[20];
    printf("Enter name: ");
    scanf("%s",name);
    printf("Your name is %s.",name);
    return 0;
}
```

# Using gets() function

- The best approach to string input is to use a library function called gets().
- This takes the start address of an area of memory suitable to hold the input as a single parameter.

```
#include <stdio.h>
int main()
{
    char name[10];
    printf("Enter name: ");
    gets(name);
    printf("\nName:");
    puts(name);
    return 0;
}
```



# String handling functions:

- There are various string handling functions define in (string.h) header file and they are:-

Function	Work of Function
<code>strlen()</code>	Calculates the length of string
<code>strcpy()</code>	Copies a string to another string
<code>strcat()</code>	Concatenates(joins) two strings
<code>strcmp()</code>	Compares two string
<code>strlwr()</code>	Converts string to lowercase
<code>strupr()</code>	Converts string to uppercase

# Cont.

## strlen() FUNCTION

- This function counts and return the number of characters in a string
  - SYNTAX:  
`n = strlen(string);`
- Where n is the integer variable which receives the value of the length of the string
- The counting ends at the first null character
  - EXAMPLE:  
`Ch = "NEW YORK";`  
`n = strlen(ch);`  
`printf(String Length = %d\n", n);`
  - OUTPUT:  
String Length = 8

# Strcpy() function

- It copies the contents of one string into another string.

Syntax: `char *strcpy(char *destination, const char *source);`

- Example:

`strcpy(str1, str2)`- it copies the contents of `s2` into `s1`.

`strcpy(str2, str1)`- it copies the contents of `s1` into `s2`.

Example code:

```
#include <stdio.h>
#include <string.h>

int main() {
    char str1[20] = "C programming";
    char str2[20];

    strcpy(str2, str1);    // copying str1 to str2
    puts(str2);

    return 0;
}
```

# Strcat() function

- The strcat(first\_string, second\_string) function concatenates two strings and result is returned to first\_string.

- **strcat() arguments: It takes two arguments i.e.**

**destination** - destination string

**source** - source string

- The strcat() function concatenates the destination string and the source string, and the result is stored in destination string.

```
#include<stdio.h>
#include <string.h>
int main(){
    char ch[10]={'h', 'e', 'l', 'l', 'o', '\0'};
    char ch2[10]={'c', '\0'};
    strcat(ch,ch2);
    printf("Value of first string is: %s",ch);
    return 0;
}
```

# Strcmp() function

- The strcmp(first\_string, second\_string) function compares two strings and returns 0 if both strings are equal.
- Syntax: strcmp(s1,s2); Here s1 and s2 are strings.

```
#include <stdio.h>
#include <string.h>

int main()
{
    char str1[] = "abcd", str2[] = "abcd";
    int result;

    result = strcmp(str1, str2);
    printf("strcmp(str1, str2) = %d\n", result);

    return 0;
}
```

# Strlwr() and strupr() function

- The strlwr(string) function returns string characters in lowercase. Let's see a simple example of strlwr() function.
- The strupr(string) function returns string characters in uppercase. Let's see a simple example of strupr() function.

```
#include<stdio.h>
#include<string.h>
void main()
{
    char x[10]="Welcome";
    strupr(x);
    printf("%s" , x);
}
```

```
#include<stdio.h>
#include<string.h>
void main()
{
    char x[10]="WELCOME";
    strlwr(x);
    printf("%s" , x);
}
```

# Strings in C and Strings in Python

- Strings in python are surrounded by either single quotation marks, or double quotation marks.
- Eg. 'Hello' or same as "Hello".
- Assigning string to variable in python compared to C is:

```
a = "Hello"
```

- For comparing two strings: Python string uses the function cmp to compare two string:

```
cmp(string1, string2)
```

- **Strings are Arrays:**
- Like many other popular programming languages, strings in Python are arrays of bytes. However, Python does not have a character data type, a single character is simply a string with a length of 1.
- Square brackets can be used to access elements of the string.

E.g.

```
a = "Hello, World!"  
print(a[1])
```

# String Functions in Python:

- Python has many functions for strings just as C has.
- The mostly or widely used functions in python strings are:

String operation	Explanation	Example
+	Adds two strings together	x = "hello " + "world"
*	Replicates a string	x = " " * 20
upper	Converts a string to uppercase	x.upper()
lower	Converts a string to lowercase	x.lower()
title	Capitalizes the first letter of each word in a string	x.title()
find, index	Searches for the target in a string	x.find(y) x.index(y)
rfind, rindex	Searches for the target in a string, from the end of the string	x.rfind(y) x.rindex(y)
startswith, endswith	Checks the beginning or end of a string for a match	x.startswith(y) x.endswith(y)
replace	Replaces the target with a new string	x.replace(y, z)
strip,rstrip, lstrip	Removes whitespace or other characters from the ends of a string	x.strip()
encode	Converts a Unicode string to a bytes object	x.encode("utf_8")



# Functions in C

- A large C program is divided into basic building blocks called as C Function.
- C functions contains set of instructions enclosed by “{ }” which perform specific operation in C program.
- The collection of all these functions creates a C program.
- **There are two types of Functions in C:**
  - 1. User-defined functions.
  - 2. Pre-defined functions.(also called as library functions)

# Library Functions

- The standard library functions are built-in functions in C programming.
- These functions are defined in header files. For example:-
- `printf()` is a standard library function to send formatted output to the screen (display output on the screen). This function is defined in the `stdio.h` header file.

# User defined 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. For example:
- Suppose, you need to find area of rectangle or circle you can create function to solve this problem.

# FUNCTION DECLARATIONS

- A function **declaration(function prototype)** tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.
- A function declaration has the following parts:

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

- For the above defined function *max()*, following is the function declaration:

```
int max(int num1, int num2);
```

- Parameter names are not important in function declaration only their type is required, so following is also valid declaration:

```
int max(int, int);
```

- Function declaration is required when you define a function in one source file and you call that function in another file. In such case you should declare the function at the top of the file calling the function.

# DEFINING A FUNCTION

- The general form of a function definition is as follows:

```
return_type function_name( parameter list )  
{  
    body of the function  
}
```

- A function definition in C language consists of
  - a *function header* and
  - a *function body*.

## EXAMPLE: FUNCTION

- Following is the source code for a function called **max()**. This function takes two parameters **num1** and **num2** and returns the maximum between the two:

```
/* function returning the max between two numbers */  
int max(int num1, int num2)  
{  
    /* local variable declaration */  
    int result;  
  
    if (num1 > num2)  
        result = num1;  
    else  
        result = num2;  
  
    return result;  
}
```

## CALLING A FUNCTION

- While creating a C function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task.
- When a program calls a function, program control is transferred to the called function. A called function performs defined task and when its *return* statement is executed or when its function-ending closing brace is reached, it returns program control back to the main program.
- To call a function, you simply need to pass the required parameters along with function name, and if function returns a value, then you can store returned value. For example:





# CODE EXAMPLE: CALLING A FUNCTION

```
#include <stdio.h>

/* function returning the max between two numbers */
int max(int num1, int num2)
{
    /* local variable declaration */
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}

void main ()
{
    /* local variable definition */
    int a = 100;
    int b = 200;
    int ret;
    /* calling a function to get max value */
    ret = max(a, b);
    printf( "Max value is : %d\n", ret );
}
```





# Example: 2

```
#include <stdio.h>
float square(float x); //declaration of user-defined function
int main() {
    float m,n;
    printf("\nenter a number:");
    scanf("\n%f",&m);

    n=square(m); //calling of function
    printf("The square is:%f",n);
}
float square(float x) //definition
{
    float p;
    p=x*x; //logic
    return p;
}
```

# GLOBAL AND LOCAL VARIABLES

- Local variable:
  - A local variable is a variable that is declared inside a function.
  - A local variable can only be used in the function where it is declared.
- Global variable:
  - A global variable is a variable that is declared outside **all** functions.
  - A global variable can be used in all functions.
- See the following example( see the next slide )



# GLOBAL AND LOCAL VARIABLES

```
#include <stdio.h>

// Global variables
int A;
int B;

int Add()
{
    return A + B;
}

void main()
{
    int answer; // Local variable
    A = 5;
    B = 7;
    answer = Add();
    printf("%d\n", answer);
    return 0;
}
```

- As you can see two global variables are declared, **A** and **B**. These variables can be used in main() and Add().
- The local variable **answer** can only be used in main().




## TYPE OF FUNCTION CALL

While calling a function, there are two ways that arguments can be passed to a function:

Call Type	Description
<b>Call by value</b>	This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.
<b>Call by reference</b>	This method copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.

## CALL BY VALUE

- The **call by value** method of passing arguments to a function copies the actual value of an argument into the formal parameter of the function.
- In this case, changes made to the parameter inside the function have no effect on the argument.
- By default, C programming language uses *call by value* method to pass arguments. In general, this means that code within a function cannot alter the arguments used to call the function. Consider the function **swap()** definition as follows.



```
/* function definition to swap the values */  
void swap(int x, int y)  
{  
    int temp;  
    temp = x; /* save the value of x */  
    x = y;    /* put y into x */  
    y = temp; /* put temp into y */  
}
```

## CALL BY REFERENCE

- The **call by reference** method of passing arguments to a function copies the address of an argument into the *formal parameter*.
- Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the passed argument.
- To pass the value by call by reference, argument pointers are passed to the functions.

```
/* function definition to swap the values */  
void swap(int *x, int *y)  
{  
    int temp;  
    temp = *x; /* save the value of x */  
    *x = *y;   /* put y into x */  
    *y = temp; /* put temp into y */  
}
```





# RECURSION

- Recursion is the process of repeating items in a self-similar way. Same applies in programming languages as well where if a programming allows you to call a function inside the same function that is called recursive call of the function as follows.

```
void recursion()  
{  
    recursion(); /* function calls itself */  
}  
  
int main()  
{  
    recursion();  
}
```

- The C programming language supports recursion, i.e., a function to call itself.
- But while using recursion, programmers need to be careful to define an exit condition (base condition) from the function, otherwise it will go in infinite loop.
- Recursive function are very useful to solve many mathematical problems like to calculate factorial of a number, generating Fibonacci series, etc.

# Flow diagram

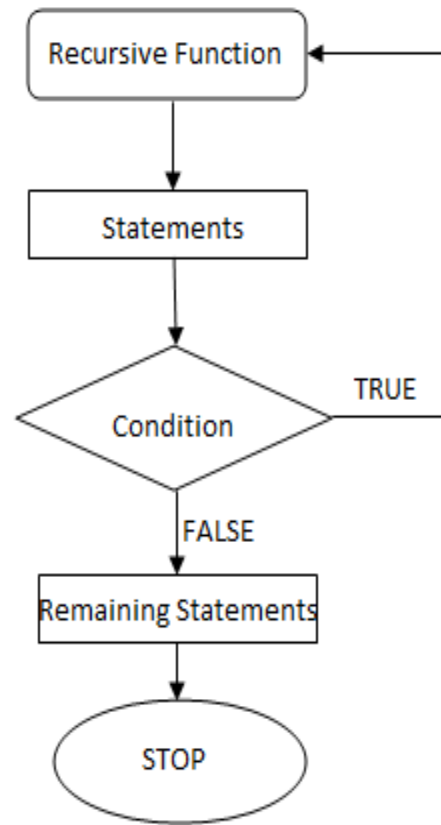


Fig: Flowchart showing recursion



# Factorial of a number Using Recursion

```
#include<stdio.h>
int find_factorial(int);
int main()
{
    int num, fact;
    //Ask user for the input and store it in num
    printf("\nEnter any integer number:");
    scanf("%d",&num);

    //Calling our user defined function
    fact =find_factorial(num);

    //Displaying factorial of input number
    printf("\nfactorial of %d is: %d",num, fact);
    return 0;
}
int find_factorial(int n)
{
    //Factorial of 0 is 1
    if(n==0)
        return(1);

    //Function calling itself: recursion
    return(n*find_factorial(n-1));
}
```

# FIBONACCI SERIES

- Following is another example, which generates Fibonacci series for a given number using a recursive function:

```
#include <stdio.h>

int fibonaci(int i)
{
    if(i == 0)
    {
        return 0;
    }
    if(i == 1)
    {
        return 1;
    }
    return fibonaci(i-1) + fibonaci(i-2);
}

int main()
{
    int i;
    for (i = 0; i < 10; i++)
    {
        printf("%d\t", fibonaci(i));
    }
    return 0;
}
```

When the above code is compiled and executed, it produces the following result:

0	1	1	2	3	5	8	13	21	34
---	---	---	---	---	---	---	----	----	----

THANK YOU