



School of Diploma Studies

Faculty of Engineering and Technology

Computer Engineering Department

Semester - 2

Data Structure (21DCECC201)

Unit – 1 Introduction to Data Structure

Mrs. Krishna M. Satani

Lecturer (DCE)



Q - 1 What is Data Structure? Give examples and applications of Data Structure.

- A data structure is a way of organizing data items that considers not only the data item stored but also their relationship to each other.
- Data structure is a way to storing and organizing data in a computer so that it can be used efficiently.
- Data structures are used in almost every program or software system.
- Elements of data structure
 - Data: value or set of values.
 - Record: collection of data.
 - File: collection of related records
- Some common examples of data structures are arrays, linked lists, queues, stacks, binary trees, and hash tables.
- Data structures are widely applied in the following areas:
 - Compiler design
 - Operating system
 - Statistical analysis package
 - DBMS
 - Numerical analysis
 - Simulation
 - Artificial intelligence
 - Graphics

Q - 2 Define Data, Information, Cells, Record, Field, Key and Search Key.

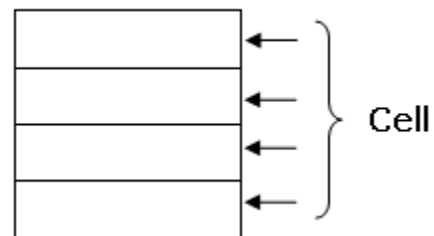
➤ **Data**

- Data is a collection of raw facts (or information) and it may or may not be meaningful.
- Examples: Roll No: 054, Semester: 3, Balance: 5000 etc.

➤ **Information**

- Information means processed or organized data.
- Examples: Percentage: 78.50%, Percentile: 91.23 etc.

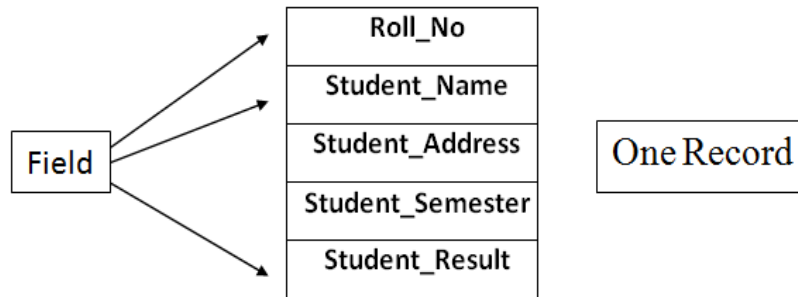
➤ **Cells**



- A cell is a memory location used to store elements of data items. It can be bit, byte or group of bytes.

➤ **Record**

- Record is a collection of information about particular item.
- A student record contain student name, student address, mobile no, result etc. An employee record contains the employee ID, employee Name, Employee salary etc.



➤ **Field**

- A record consists of several fields.
- Roll No, Name, result, employee name, employee salary are called field

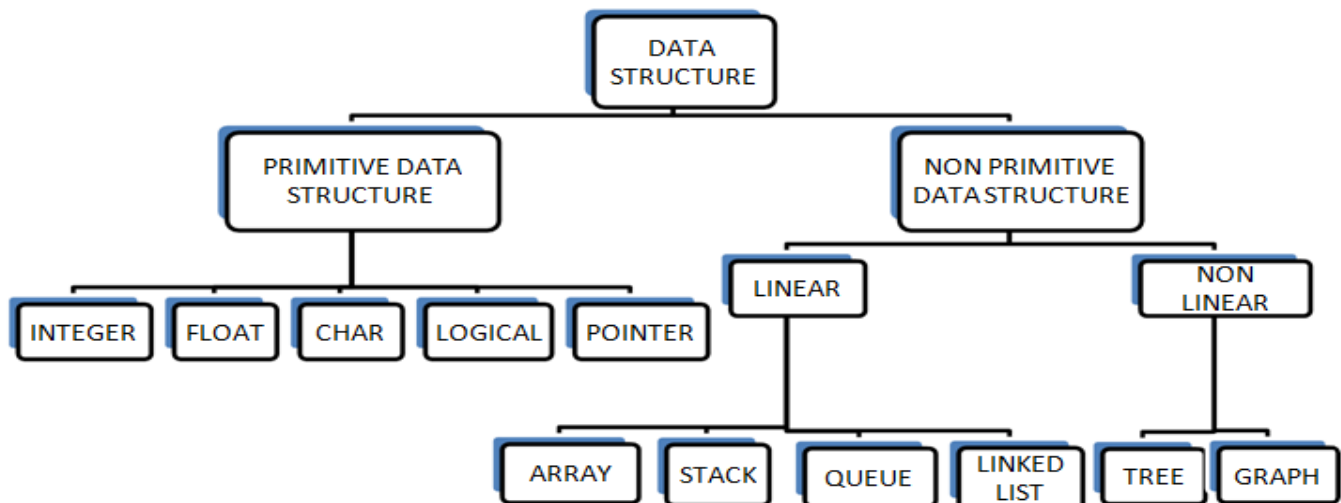
➤ **Key**

- To search for a record within a data file you have to select a field known as key.
- For example to search detail of a student you have to select Roll no as key.

➤ **Search Key**

- Every record has a key. The key **that** you are looking for In search is called search key.
- The search key is compared with the key field, if it matched then the record returns.

Q - 3 Explain Types of Data Structure.

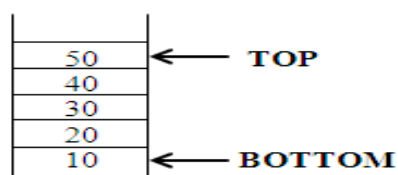


➤ **Primitive Data Structure**

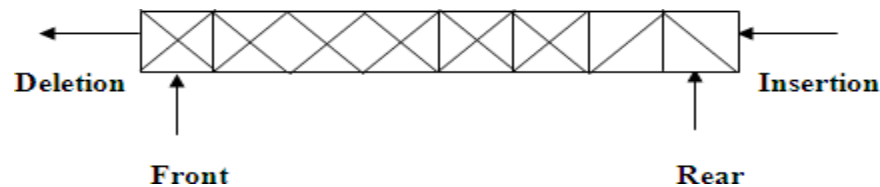
- The data structure that directly operated upon by machine level instructions is called primitive data structure.
- Example of primitive data structure are Numeric, real, character, pointer and logical data.
 - **Numeric (Integer):** An Integer can be positive or negative. **Example is 1078, 456** etc. Different methods are used to represent such as signed & magnitude, Radix etc.
 - **Real:** The number having fractional part i.e. decimal point is called real number. Common way of representing real number is normalized floating point representation. **Example: 128.32.**
 - **Character:** Character data structure is used to store nonnumeric information. It can be letters [A-Z], [a-z], operators and special symbols. Two most commonly known character set supported by computer are ASCII and EBCDIC.
 - **Pointer:** Pointer is a variable which points to the memory address. This memory address is the location of other variable in memory.
 - **Logical Data:** A logical data item is a primitive data structure that can assume the values of either “true” or “false”. Most commonly used logical operators Are AND, OR and NOT.

➤ **Non Primitive Data Structure:**

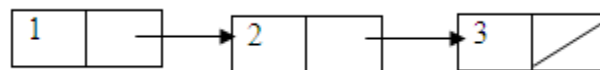
- The data structure that does not directly operate upon machine level instruction are called non primitive data structure.
- They are two types of non primitive data structure.
 - Linear data structure
 - Non Linear data structure
- **In the linear data structure processing of data items is possible in linear fashion.**
- Data are processed one by one sequentially.
- Examples of the linear data structure are:
 - Arrays
 - Stack
 - Queue
 - Linked List
- **Array:** Array is an ordered set which consist of a fixed number of object.
- **Stack:** A stack is a linier list in which insertion and deletion operations are performed at only one end of the list.



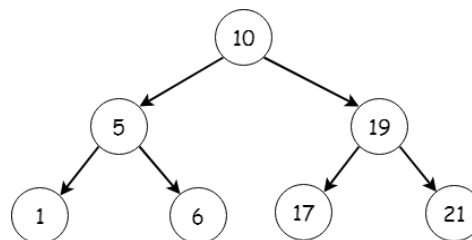
- **Queue:** A queue is a linear list in which insertion is performed at one end called rear and deletion is performed at another end of the list called front.



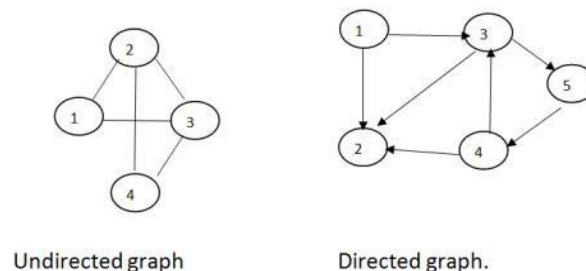
- **Linked list:** A linked list is a collection of nodes.
- Each node has two fields:
 - Information
 - Address, which contains the address of the next node



- **In the Non linear data structure processing of data items is not possible in linear fashion.**
- Examples of the non linear data structure are:
 - Tree
 - Graph
- **Tree:** In tree data contains hierarchical relationship.



- **Graph:** This data structure contains relationship between pairs of element.



- **Graphs are many types:**
 - Directed graph
 - Undirected graph
 - Mixed graph
 - Simple graph
 - Null graph

Q - 4 Explain the operations of Data Structure?

- Following operations are performed on data structure
 - **Traversing:** Access each element of the data structure and doing some processing over the data.
 - **Inserting:** Insertion of new elements into the data structure.
 - **Deleting:** Deletion of specific elements.
 - **Searching:** Searching for a specific element.
 - **Sorting:** Sorting the data in ascending or descending ordered.
 - **Merging:** Combination of two data structure.

Q - 5 Explain difference between List and Array?

List	Array
No. of elements in a list are not fixed	No. of elements in an array is fixed.
It uses pointer variable which occupies an extra memory space.	It does not use pointer variable so it does not occupies extra memory space.
Insertion and deletion operation are very easy to perform.	Insertion and deletion operation are very hard to perform.
There are two types of List: 1. Linear List 2. Non Linear List	There are two types of array: 1. One dimensional array 2. Two dimensional array
Searching is slower in case of List.	Searching is faster in case of array.

Q - 6 Define Algorithm and explain the key features of an Algorithm.

- **An algorithm is a finite set of instructions that accomplishes a particular task.**
- Algorithms are mainly used to achieve software reuse. Once we have an idea or a blueprint of a solution, we can implement it in any high-level language like C, C++, or Java.
- An algorithm is basically a set of instructions that solve a problem.
- It is considered to be an effective procedure for solving a problem in finite number of steps.
- That is, a well-defined algorithm always.
- Provides an answer and is guaranteed to terminate.
- It is not uncommon to have multiple algorithms to tackle the same problem, but the choice of a

particular algorithm must depend on the time and space complexity of the algorithm.

- **All algorithms must satisfy the following criteria:**

- **Input:** Zero or more quantities are input.
- **Output:** At least one output is produced.
- **Definiteness:** Each steps of algorithm must be precisely defined
- **Finiteness:** An algorithm must always terminate after a finite number of steps
- **Effectiveness:** All the operations to be performed in the algorithm must be sufficient.

- **Key Features of Algorithm:**

- **Name of Algorithm:** Every algorithm is given identifying name, Written in capital letters.
- **Steps:** Algorithm is made up of sequence of number of steps, each beginning with phrase enclosed in square brackets, which gives description of those steps.
- **Assignment Statements:** This is indicated by a placing an arrow (->) between righthand side of the statement and the variable receiving the value.
- **If statement:** It has two form
If
(condition)
then _____
If (condition)
then_____ else ____
- **Case statement:** The case statement is used for multiple choice type solution.
- **Repeat statement:** It is used to repeat a step until a given logical condition is false.
Forexample WHILE Loop, FOR loop
- **Goto statement:** The goto statement causes unconditional transfer of control to the step referenced.
- **Exit statement:** The exit statement is used to terminate an algorithm. It is usually last step of algorithm.

Q - 7 Explain the Analysis of Algorithm.

- **Analyzing an algorithm means determining the amount of resources (such as time and memory) needed to execute it.**
- Algorithms are generally designed to work with an arbitrary number of inputs, so the efficiency or complexity of an algorithm is stated in terms of time and space complexity.

➤ Time Complexity:

- **The time complexity of an algorithm is basically the running time of a program as a function of the input size.**
- The number of (Machine) instruction which a program executes during its running time is called Time Complexity.
- We try to keep this idea of time separate from "wall clock" time, since this number depends on the size of program's input that is approximately on the number of the string to be sorted and the algorithm used.
- So approximately the time complexity of the program "Sort an array of n strings by minimum search" is described by the expression ranging from $n \cdot \log_2 n$ or n^2 .
- If we double the number of strings to be sorted, the computing time quadruples.
- "Time" means the number of memory accesses performed, the number of comparisons between integers, the number of times some inner loop is executed.

➤ Space Complexity:

- **The space complexity of an algorithm is the amount of computer memory that is required during the program execution as a function of the input size.**
- The better the time complexity of an algorithm is, the faster the algorithm will carry out his work in practice
- Space complexity is Space complexity is the number of memory cells which an algorithm needs.

➤ Asymptotic Notations:

- Generally we use asymptotic notation as a convenient way to examine what can happen in a function in the worst case or in the best case.

➤ Big 'O' Notation:

- In algorithm $T(n)$ is measure of how much time is required to execute algorithm. With given n value.
- Number of stat. executed in function for n elements of data is function of number of elements,

express as $f(n)$

- The running time complexity is always specified in the so called O-notation.
- The sorting method has running time $O(n^2)$.
- The O is the formal method of expressing the upper bound of an algorithm's running time. It is a measure of the longest amount of time it could possibly take for the algorithm to complete.
- This factor is Big-O notation and expressed as $O(n)$, that is on the order of n .

➤ **Worst-case time Complexity:**

- This denotes the behavior of an algorithm with respect to the worst possible case of the input instance.
- The worst-case running time of an algorithm is an upper bound on the running time for any input.
- Therefore, having the knowledge of worst-case running time gives us an assurance that the algorithm will never go beyond this time limit.

➤ **Average-case time Complexity:**

- The average-case running time of an algorithm is an estimate of the running time for an 'average' input.
- It specifies the expected behavior of the algorithm when the input is randomly drawn from a given distribution.
- Average-case running time assumes that all inputs of a given size are equally likely.

➤ **Best-case time Complexity:**

- The term 'best-case performance' is used to analyze an algorithm under optimal conditions.
- For example, the best case for a simple linear search on an array occurs when the desired element is the first in the list.
- However, while developing and choosing an algorithm to solve a problem, we hardly base our decision on the best-case performance.
- It is always recommended to improve the average performance and the worst-case performance of an algorithm.

Q - 8 What is an Array? Write the characteristic of Array.

➤ **Array :**

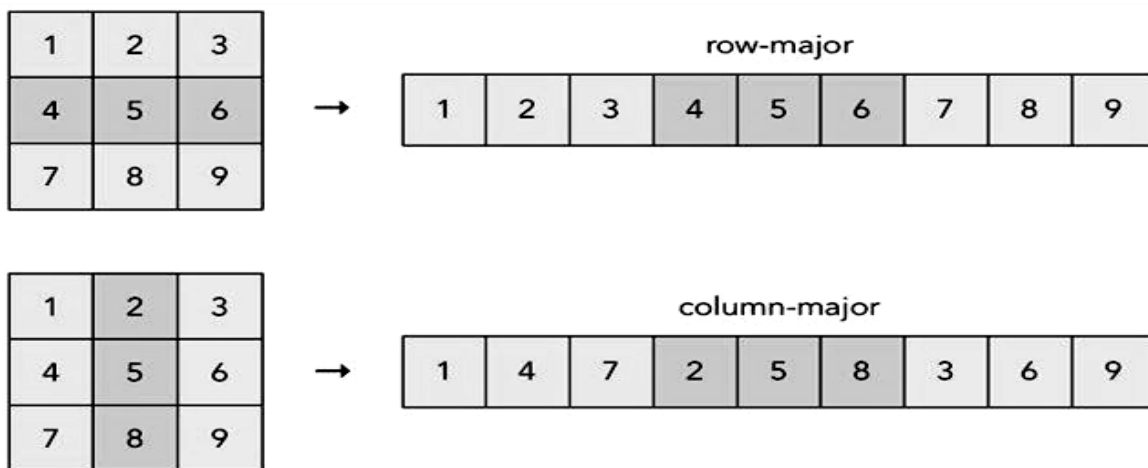
- Array is a set of finite number of elements that have same data type.
- It means it contains one type of data only.
- Array stores its elements in adjacent memory locations.

➤ **Characteristic of Array:**

- Array is non primitive and linear data structure
- Array is group of elements that have same data type.
- Every element has assigned unique number which is called address of particular element.
- Memory occupied by array is depending on data type and number of elements of array.
- Array elements are stored sequentially or in linear fashion.
- When array is declared and not initialized, it contains garbage values. If array is declared as static, all elements are initialized to zero.
- In array insertion and deletion operation is slower and time consuming but searching and sorting operation is faster.

Q - 9 What is Row major and Column major Array?

- In Row-major form, all the elements of the first row are printed, then the elements of the second row and so on up to the last row
- Row-major order is used in C/C++/Objective-C (for C-style arrays), Mathematical, PL/I, Pascal etc.
- In Column-major form, all the elements of the first column are printed, then the elements of the second column and so on up to the last column
- Column-major order is used in FORTRAN, OpenGL and OpenGL ES, MATLAB, GNU Octave, S-Plus etc.



➤ **Row Major Array**

- One method of representing a two dimensional array in memory is the row major order representation.
- In this representation the first row of the array occupies the first set of memory locations reserved for the array, the second row occupies the next set and so on.

- An array consisting of n rows and m columns can be stored sequentially in row major order as :

- Row1 A[1,1] A[1,2] A[1,3] A[1,4] A[1,m]
- Row 2 A[2,1] A[2,2] A[2,3] A[2,4] A[2,m]
-
- Row n A[n,1] A[n,2] A[n,3] A[n,4] A[n,m]

Column 1	Column 2	Column 3
A[1][1]	A[1][2]	A[1][3]
A[2][1]	A[2][2]	A[2][3]

A[1][1]	ROW 1
A[1][2]	
A[1][3]	
A[2][1]	ROW 2
A[2][2]	
A[2][3]	

- The address of element A[i, j] can be obtained by evaluating expression:
 - $\text{Loc}(A[i, j]) = L0 + (i - 1) * m + (j - 1)$
- Where L0 is the address of the first element (Base Address) in the array.
 - i is the row index.
 - j is the column index.
 - m is the no. of columns.
- For example the address of element A[1,2] is calculated as:
 - $A[1, 2] = L0 + (i - 1) * m + (j - 1)$
 - Here, m=3, n=2, i=1, j=2
 - $= L0 + (1 - 1) * 3 + (2 - 1)$
 - $= L0 + 0 + 1$
 - $= L0 + 1$

➤ Column Major Array

- One method of representing a two dimensional array in memory is the column major order representation.
- In this representation the first column of the array occupies the first set of memory locations reserved for the array, the second column occupies the next set and so on.
- An array consisting of n rows and m columns can be stored sequentially in column major order as :
 - Column 1 A[1,1] A[2,1] A[3,1] A[4,1] A[n,1]
 - Column 2 A[1,2] A[2,2] A[3,2] A[4,2] A[n,2]
 -
 - Column m A[1,m] A[2,m] A[3,m] A[4,m] A[n,m]
- Example:** A two dimensional array consist of two rows and three columns is stored sequentially in column major order as:

	Column 1	Column 2	Column 3
Row 1	A[1][1]	A[1][2]	A[1][3]
Row 2	A[2][1]	A[2][2]	A[2][3]

- The address of element $A[i, j]$ can be obtained by evaluating expression:
 - $\text{Loc}(A[i, j]) = L0 + (j - 1) * n + (i - 1)$
- Where $L0$ is the address of the first element (Base Address) in the array.
 - i is the row index
 - j is the column index
- for example the address of element $A[1,2]$ is calculated as:
 - $A[1,2] = L0 + (j - 1) * n + (i - 1)$
 - Here, $m=3, n=2, i=1, j=2$
 - $=L0 + (2 - 1) * 2 + (1 - 1)$
 - $=L0 + 2 + 0$
 - $=L0 + 2$

$A[1][1]$	}	Column 1
$A[2][1]$		
$A[1][2]$	}	Column 2
$A[2][2]$		
$A[1][3]$	}	Column 3
$A[2][3]$		

Q - 10 Explain various Array operations.

➤ Traversal:

- This operation is used to traverse one dimensional array for inserting or displaying elements of one dimensional array.
 - Algorithm: TRAVERSE(A,LB,UB)**
 - A is an array.
 - LB is lower limit of an array.
 - UB is upper limit of an array.
 - [Initialize] $COUNT \leftarrow LB$
 - [perform traversal and increment counter]
- While ($COUNT \leq UB$)
- (1) Write $A[COUNT]$ or Read $A[COUNT]$
- (2) $COUNT \leftarrow COUNT + 1$
3. [Finished] & Exit

➤ Insertion:

- This operation is used to insert an element into one dimensional array.
 - Algorithm: INSERT(A,POS,N,VALUE)**
 - A is an array.
 - POS indicates position at which you want to insert element.
 - N indicates number of elements in an array.
 - $VALUE$ indicates value (element) to be inserted.
 - [Initialize] $TEMP \leftarrow N$
 - [Move the elements one position down]
- Repeat While ($TEMP \geq POS$)
- (1) $A[TEMP + 1] \leftarrow A[TEMP]$
- (2) $TEMP \leftarrow TEMP - 1$
3. [Insert element]
 $A[POS] \leftarrow VALUE$

4. [Increase size of array]

$N \leftarrow N+1$

5. [Finished] & Exit

➤ **Deletion:**

- This operation is used to delete an element from one dimensional array.

- **Algorithm: DELETE(A,POS,N,VALUE)**

- A is an array.
- POS indicates position at which you want to delete element.
- N indicates number of elements in an array.
- VALUE indicates value (element) to be deleted.

1. [Initialize] $TEMP \leftarrow N$

2. [Move the elements one position up]

Repeat While ($TEMP \leq N-1$)

(1) $A[TEMP] \leftarrow A[TEMP + 1]$

(2) $TEMP \leftarrow TEMP + 1$

3. [Decrease size of array]

$N \leftarrow N-1$

4. [Finished] & Exit

➤ **Searching:**

- This operation is used to search particular element in one dimensional array

- **Algorithm: SEARCH(A,N,X)**

- A is an array.
- N indicates number of elements in an array.
- X indicates value (element) to be search.

1. [Initialize] $TEMP \leftarrow 1$

$A[N+1] \leftarrow X$

2. [Perform search]

Repeat While $A[COUNT] \neq X$

$COUNT \leftarrow COUNT + 1$

3. [Successful Search?]

If $COUNT = N+1$ then

Write "Unsuccessful Search"

Else

Write "Successful Search"

4. [Finished] & Exit

Q - 11 Explain searching an element into an array. (Explain Linear and Binary Search with an example)

- Searching means to find whether a particular value is present in an array or not.
- If the value is present in the array, then searching is said to be successful and the searching process gives the location of that value in the array.

- If the value is not present in the array, the searching process displays an appropriate message and in this case searching is said to be unsuccessful.
- There are two popular methods for searching the array elements
 - **Linear Search**
 - **Binary Search**

➤ **Linear Search:**

- Linear search, also called as sequential search, is a very simple method used for searching an array for a particular value. It works by comparing the value to be searched with every element of the array one by one in a sequence until a match is found.
- Linear search is mostly used to search an unordered list of elements (array in which data elements are not sorted)

Key	List
3	6 4 1 9 7 3 2 8
3	6 4 1 9 7 3 2 8
3	6 4 1 9 7 3 2 8
3	6 4 1 9 7 3 2 8
3	6 4 1 9 7 3 2 8
3	6 4 1 9 7 3 2 8

- **Algorithm: SEARCH(A,N,X)**
 - A is an array.
 - N indicates number of elements in an array.
 - X indicates value (element) to be search.
 1. [Initialize] $TEMP \leftarrow 1$
 $A[N+1] \leftarrow X$
 2. [Perform search]

Repeat While $A[COUNT] \neq X$

$COUNT \leftarrow COUNT + 1$
 3. [Successful Search?]

If $COUNT = N+1$ then

Write "Unsuccessful Search"

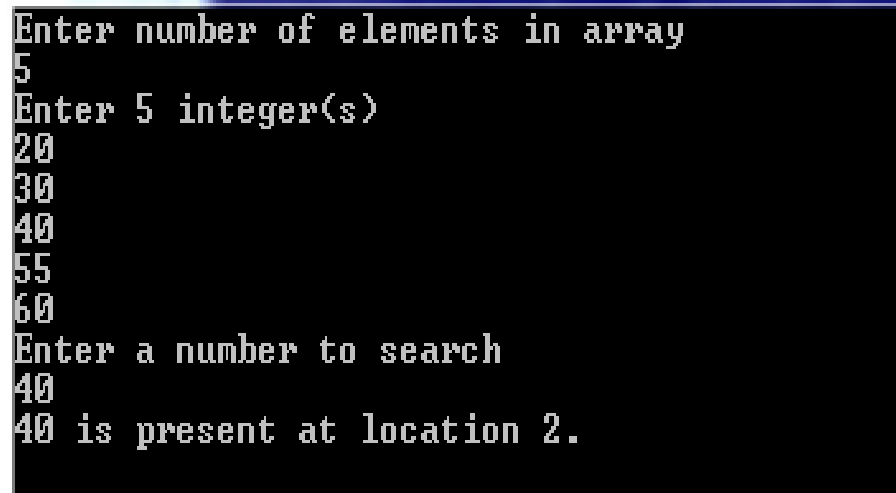
Else

Write "Successful Search"
 4. [Finished] & Exit

- **Program:**

```
#include <stdio.h>
int main()
{
    int array[100], search, i, n;
    clrscr();
    printf("Enter number of elements in array\n");
    scanf("%d", &n);
    printf("Enter %d integer(s)\n", n);
    for (i = 0; i < n; i++)
    {
        scanf("%d", &array[i]);
    }
    printf("Enter a number to search\n");
    scanf("%d", &search);
    for (i = 0; i < n; i++)
    {
        if (array[i] == search) /* If required element is found */
        {
            printf("%d is present at location %d.\n", search, i);
            break;
        }
    }
    if (i == n)
        printf("%d isn't present in the array.\n", search);
    return 0;
}
```

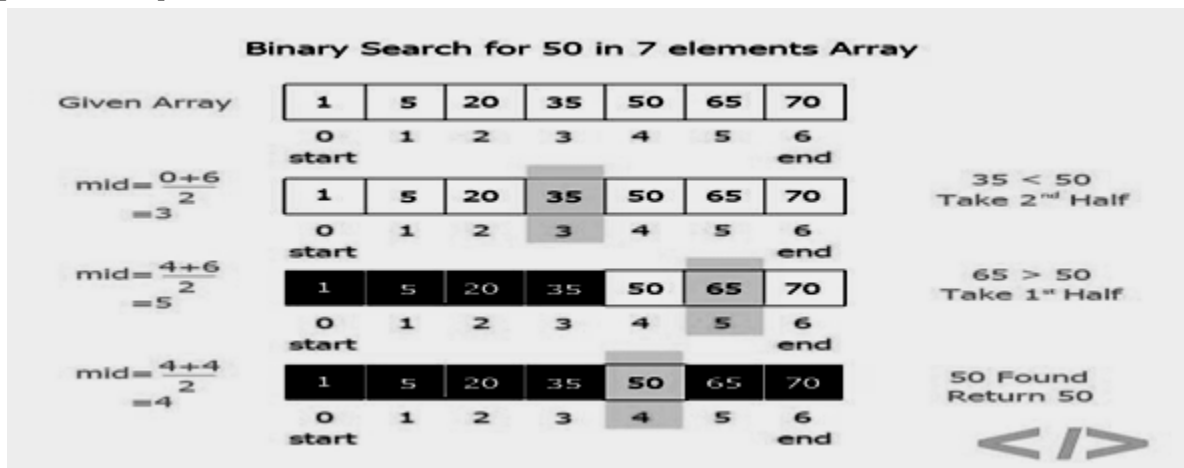
- **Output:**



```
Enter number of elements in array
5
Enter 5 integer(s)
20
30
40
55
60
Enter a number to search
40
40 is present at location 2.
```

➤ **Binary Search:**

- Binary search is an extremely efficient algorithm and work on divide and conquer method.
- This search technique consumes less time in searching the given item in minimum possible comparisons.
- To do the binary search, first, we have to sort the array elements.
- The logic behind this technique is given below:
 - **First, find the middle element of the array.**
 - **The middle element of the array is compared to the element to be searched.**
- **There are three cases could arise:**
 - If the middle element is the required element, then the search is successful.
 - When the middle element is greater than the desired item, then search only the first half (left side) of the array.
 - When the middle element is less than the desired element, then search in the second half (right side) of the array.
- Repeat the steps 2 and 3 until an element is found or not found in the search area.



- **Algorithm: BINARY SEARCH(A,N,X)**
 - A is an array.
 - N indicates number of elements in an array.
 - X indicates value (element) to be search.
 1. [Initialize] $low \leftarrow 0$
 $high \leftarrow n - 1$
 2. [Perform search]

Repeat While $low \leq high$
Repeat step 3 & 4
 3. $mid = (low+high)/2$;
 4. if($num < A[mid]$)

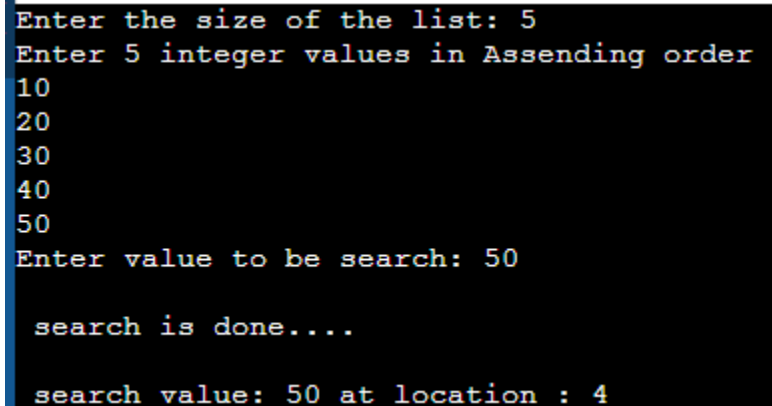
then $high = mid - 1$

else
if($num > A[mid]$)

- ```
 then low = mid +1
 else if (num == A[mid])
 write("search value: %d at location : %d",num,mid")
5. if(low > high)
 write(("Element Not found in the list.")
6. [Finished] & Exit
```

- **Program:**

```
#include <stdio.h>
int main()
{
 int low, high, mid, n, i, num, a[100];
 printf("Enter the size of the list: ");
 scanf("%d",&n);
 printf("Enter %d integer values in Assending order\n",n);
 for (i = 0; i < n; i++)
 scanf("%d",&a[i]);
 printf("Enter value to be search: ");
 scanf("%d", &num);
 low = 0;
 high = n - 1;
 while(low <= high)
 {
 mid = (low+high)/2;
 if(num<a[mid])
 {
 high = mid - 1;
 }
 else
 {
 if(num>a[mid])
 {
 low = mid + 1;
 }
 else if(num == a[mid])
 {
 printf("\n search is done....\n");
 printf("\n search value: %d at location : %d",num,mid);
 break;
 }
 }
 }
}
```



```
Enter the size of the list: 5
Enter 5 integer values in Assending order
10
20
30
40
50
Enter value to be search: 50

search is done....

search value: 50 at location : 4
```

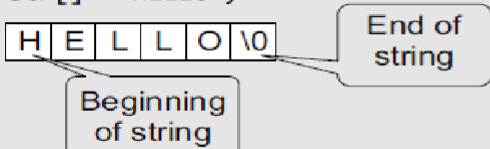


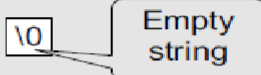
```

if (low > high)
 printf("Element Not found in the list.");
return 0;
}

```

## Q - 12 What is String? How strings are represented?

- **A String is an array of characters.**
- It is defined as number of characters written in double quotation marks.
- In C, a string is a null-terminated character array. This means that after the last character, a null character ('\0') is stored to signify the end of the character array.
- For example, if we write `char str[] = "HELLO";` then we are declaring an array that has five characters, namely, H, E, L, L, and O. Apart from these characters, a null character ('\0') is stored at the end of the string.
- So, the internal representation of the string becomes `HELLO\0`. To store a string of length 5, we need 5 + 1 locations (1 extra for the null character).

|                                                                                                                                                                                                                                                                                        |                                                                                                                                                                                                                        |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><code>char str[] = "HELLO";</code></p>                                                                                                                                                            | <p><code>char ch = 'H';</code></p> <p>Here H is a character not a string. The character H requires only one memory location.</p>  |
| <p><code>char str[] = "H";</code></p>  <p>Here H is a string not a character. The string H requires two memory locations. One to store the character H and another to store the null character.</p> | <p><code>char str[] = "";</code></p>  <p>Although C permits empty string, it does not allow an empty character.</p>               |

- Character set of string
  - Alphabets ( A to Z and a to z)
  - Numeric (0 to 9)
  - Special character ( + - \* [ ] { } ...)
- Common operations performed on string
  - Reading and writing
  - Length of string
  - Concatenation of string
  - Copying one string to another
  - Comparing string
  - Extracting some portion of a string
  - Text editing
  - Pattern matching

➤ **Reading and Writing strings**

- The string is declared as `char name[10];`
- Where name is a string which can store maximum of 9 characters and one byte is needed to store '\0' at the end of string.
- String can be initialized as `char name[6] = "Hello";`
- String can also be declared as `char name[6] = {'H','e','l','l','o'};`

➤ **Reading a string**

- Then str can be read by the user in two ways:
  - using `scanf` function for example `scanf("%s", str);`
  - using `gets()` function for example `gets(str);`

➤ **Writing a string**

- Strings can be displayed on the screen using the following two ways:
  - using `printf()` function for example `printf("%s", str);`
  - using `puts()` function for example `puts(str);`

➤ **Program**

```
#include <stdio.h>
int main()
{
 char str[100];
 printf("Enter a value :");
 gets(str);
 printf("\nYou entered: ");
 puts(str);
 return 0;
}
```

**Q - 13 Explain String length Operation and write algorithm and the program.**

- **String Length Operation**
- The number of characters in a string constitutes the length of the string.
- For example, `LENGTH ("C PROGRAMMING IS FUN")` will return 20. Note that even blank spaces are counted as characters in the string.
- **Algorithm: String Length**

**Procedure LEN(str)**

Step 1: [Initialization]

$l \leftarrow 0$

[Read string]

Step 2: Read (str)  
Step 3: While (str <> NULL) Repeat  
     $l \leftarrow l + 1$   
Step 4: Write ('Length of string: l')  
Step 5: [Finished] & Exit

- **Program**

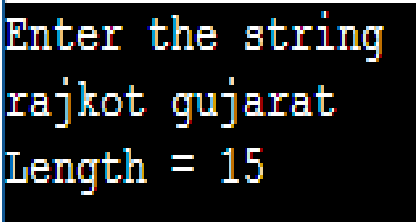
```
#include<stdio.h>
#include<string.h>
int main()
{
 char str[100];
 int i, l = 0;

 printf("Enter the string\n");
 fgets(str,100,stdin);

 for(i = 0; str[i] != '\0'; i++)
 l++;

 printf("Length = %d\n",l);

 return 0;
}
```



```
Enter the string
rajkot gujarat
Length = 15
```

**Q - 14 Explain String Copy Operation and write algorithm and the program.**

- **String Copy Operation:**
- Suppose we have two strings S1 and S2, then if we have to copy string S2 into S1 we required copy function.
- S1 = Blank or NULL String,
- S2 = "Computer"
- STRCOPY (S1, S2) will copy string S2 into S1.
- **Algorithm: STRING COPY ALGORITHM**

**Procedure STRCOPY(str1 , str2)**

Step 1: [Initialization]  
    Str1  $\leftarrow$  Null  
     $i \leftarrow 0$   
    [Read string str2]  
Step 2: Read (str2)  
    [copy operation performed]

Step 3: While (str2 <> NULL) Repeat

Str1[i] ← str2[i]

i ← i + 1

Step 4: [print the string after copy operation performed]

Write("str1")

Step 5: [Finished] & Exit

- **Program**

```
#include<stdio.h>
#include<string.h>
int main()
{
 char s1[100], s2[100];
 int i=0;
 printf("\nEnter the string :");
 fgets(s2,100,stdin);
 while (s2[i] != '\0')
 {
 s1[i] = s2[i];
 i++;
 }
 s1[i] = '\0';
 printf("\nCopied String is.... %s ", s1);
 return 0;
}
```

```
Enter the string :rajkot
Copied String is.... rajkot
```

**Q - 15 Explain String Concatenation Operation and write algorithm and the program.**

- Suppose we have two string S1 and S2 then concatenation of S1 and S2 are combined two sting together in one string.
- If S1 = "Comp" and S2 = "uter"
- Then concatenation of S1 and S2 = "Computer"
- **Algorithm: String Concatenation**

**Procedure STRCAT (STR1, STR2, STR3)**

Step 1: [Initialization]

Str3 ← Null

i ← 0

j ← 0

k ← 0

Step 2: [copy str1 string into str3]

While (str1 <> NULL) Repeat

Str3[k] ← str1[i]

```
i ← i + 1
k ← k + 1
Step 3: [copy str2 string into str3]
While (str2 <> NULL) Repeat
 Str3[k] ← str2[i]
 j ← j + 1
 k ← k + 1
Step 4: [print the string after concatenation operation performed]
Write("str3")
Step 5: [Finished] & Exit
```

- **Program**

```
#include<stdio.h>
#include<string.h>
void main(void)
{
 char str1[25],str2[25],str3[50];
 int i=0,j=0,k=0;
 printf("\nEnter First String:");
 gets(str1);
 printf("\nEnter Second String:");
 gets(str2);
 while(str1[i]!='\0')
 {
 str3[k] = str1[i];
 i++;
 k++;
 }
 while(str2[j]!='\0')
 {
 str3[k]=str2[j];
 j++;
 k++;
 }
 str3[k]='\0';
 printf("\n Concatenated String is..... %s",str3);
}
```

Enter First String:raj  
Enter Second String:kot  
Concatenated String is..... rajkot

**Q - 16 Explain Sub String Operation and write algorithm and the program.**

- **Sub string Operation**
- To extract a substring from a given string, we need the following three parameters:
  - The main string,

- The position of the first character of the substring in the given string, and
- The maximum number of characters/length of the substring.
- For example, if we have a string str[] = "Welcome to the world of programming"; Then, SUBSTRING(str, 15, 5) = world
- String is str = "Computer"
- pos = starting position of sub string
- num = number of character of sub string
- A sub string function as substr(str, pos, num)
- Suppose pos = 4 and num = 3
- substr(STR, 4, 3) = "put"
- **Program:**

```
#include <stdio.h>
#include <conio.h>
int main()
{
 char str[100], substr[100];
 int i=0, j=0, n, m;
 clrscr();
 printf("\n Enter the main string : ");
 gets(str);
 printf("\n Enter the position from which to start the substring: ");
 scanf("%d", &m);
 printf("\n Enter the length of the substring: ");
 scanf("%d", &n);
 i=m;
 while(str[i] != '\0' && n>0)
 {
 substr[j] = str[i];
 i++;
 j++;
 n--;
 }
 substr[j] = '\0';
 printf("\n The substring is : ");
 puts(substr);
 getch();
 return 0;
}
```

### **Output**

```
Enter the main string : Hi there
Enter the position from which to start the substring: 1
Enter the length of the substring: 4
The substring is : i th
```

- **Algorithm: Procedure SUBSTR(str, cursor, num)**

```
Step 1: [Initialization]
 i ← 0
 j ← 0
 pos
 num
 substr ← Null
 [input the value]
Step 2: Read (str, pos, num)
 [Copy the string Str to subject string]
Step 3: i ← pos
Step 4: while (str[i] <> 0 && num <> 0) repeat
 substr[j] ← str[i]
 pos ← pos + 1
 i ← i + 1
 num ← num - 1
Step 5: [print original string and string]
 write ("substr")
Step 6: [finished]Exit
```

**Q - 17 Explain String Comparison Operation and write algorithm and the program.**

- **String Comparison Operation**
- If S1 and S2 are two strings, then comparing the two strings will give either of the following results:
  - S1 and S2 are equal
  - S1 > S2
  - S1 < S2
- To compare the two strings, each and every character is compared from both the strings. If all the characters are the same, then the two strings are said to be equal
- Compare s1 string and s2 string character by character, if both are same then given result "equal" and both are different then given result "not equal"
  - EX: S1 = "comp"                      EX: S1 = "comp"
  - S2 = "comp"                      S2 = "computer"
  - Now compare of S1 and S2              Now compare of S1 and S2
  - Answer: equal                      Answer: not equal
- **Algorithm STRING COMPARISON**
- **PROCEDURE STRCMP(str1, str2)**



```
Step 1: [Initialization]
 i ← 0
Step 2: [READ TWO STRING]
 Read (str1)
 Read(str2)
Step 3: [COMPARE TWO STRINGS CHARACTER BY CHARACTER]
 Repeat while (Str1[i]== Str2[i] && Str1[i] == '\0')
 i++
 If (str1 [i] < str2 [i])
 then
 Write ("str1 is Less than str2")
 Elseif(str[i] >str[i])
 then
 Write ("str1 is Greater than str2")
 else
 Write ("Both strings are same")
Step 6: [finished]& Exit
```

- **Program:**

```
#include <stdio.h>
#include <string.h>
int main()
{
 char Str1[100], Str2[100];
 int result, i;
 i = 0;
 printf("\n Please Enter the First String : ");
 gets(Str1);
 printf("\n Please Enter the Second String : ");
 gets(Str2);
 while(Str1[i] == Str2[i] && Str1[i] == '\0')
 i++;
 if(Str1[i] < Str2[i])
 {
 printf("\n str1 is Less than str2");
 }
 else if(Str1[i] > Str2[i])
 {
 printf("\n str2 is Less than str1");
 }
 else
 {
 printf("\n str1 is Equal to str2");
 }
}
```

```
Please Enter the First String : John
Please Enter the Second String : Ani
str2 is Less than str1
```

```
 return 0;
 }
```

**Q - 18 Explain String Insertion Operation and write algorithm and the program.**

- **String Insertion Operation**
- The insertion operation inserts a string S in the main text T at the kth position. The general syntax of this operation is INSERT(text, position, string)
- For example, INSERT("XYZXYZ", 3, "AAA") = "XYZAAAXYZ"
- **Algorithm Insertion(text, position, string)**
- Text: Original string
- Position: to insert a new string at this position
- String: small string to be inserted

```
Step 1: [Initialization]
 I ← 0
 Read (str1, pos, str2)
Step 2: [Find length of string]
 L1 ← LEN(str1)
 L2 ← LEN(str2)
Step 3: [Shift the str1 elements from given position to right]
 for(I = L1; i>=pos ; i--) repeat
 Str1[i + L2] ← str1[i]
Step 4: [Insert substring at given position]
 for(I = 0; i<L2;i++) repeat
 A[pos + i] ← str2[i]
Step 5: [print the final string]
 Write("str1")
Step 6: [Finished] & Exit
```

- **Program:**  
 #include <stdio.h>  
 #include <string.h>  
 void main()  
 {  
 char a[10];  
 char b[10];  
 int pos,i,l1,l2;

```
puts("Enter First String:");
gets(a);
puts("Enter Second String:");
gets(b);
printf("Enter the position where the item has to be inserted: ");
scanf("%d",&pos);
l1 = strlen(a);
l2 = strlen(b);
if(pos>l1)
{
 printf("Invalid");
}
for(i=l1;i>=pos;i--)
{
 a[i+l2] = a[i];
}
for(i=0;i<l2;i++)
{
 a[pos + i] = b[i];
}
printf("New string....%s",a);
}
```

Enter First String:  
abcd  
Enter Second String:  
xyz  
Enter the position where the item has to be inserted: 0  
New string....xyzabcd

**Q - 19 Explain String Deletion Operation and write algorithm and the program.**

- **String Deletion Operation**
- The deletion operation deletes a substring from a given text.
- We can write it as DELETE (text, position, length)
- For example, DELETE("ABCDXXXABCD", 4, 3) = "ABCDABCD"
- **Algorithm: Deletion (text, position, length)**

Step 1: [Initialization]

$I \leftarrow 0$

Read (str, pos, num)

Step 2: [Find length of string]

$L \leftarrow \text{LEN}(\text{str})$

Step 3: [ delete elements from str and shift up all elements]

for( $i = \text{pos} + \text{num}$ ;  $i \leq l$ ;  $i = i - 1$ ) repeat

$\text{Str}[i - \text{num}] \leftarrow \text{str}[i]$

Step 5: [print the final string]

Write("str1")

Step 6: [Finished] & Exit

- **Program:**

```
#include <stdio.h>
#include <string.h>
void main()
{
 char a[100];
 int pos,i,l,num;
 puts("Enter the String:");
 fgets(a,100,stdin);
 printf("Enter the position where the item has to be deleted: ");
 scanf("%d",&pos);
 printf("Enter the number of charcters you want to delete: ");
 scanf("%d",&num);
 l = strlen(a);
 if(pos+num>l)
 {
 printf("Invalid");
 }
 for(i= pos+num ; i<=l; i++)
 {
 a[i-num] =a[i];
 }
 printf("New string....%s",a);
}
```

Enter the String:  
rajkot india  
Enter the position where the item has to be deleted: 0  
Enter the number of charcters you want to delete: 6  
New string.... india

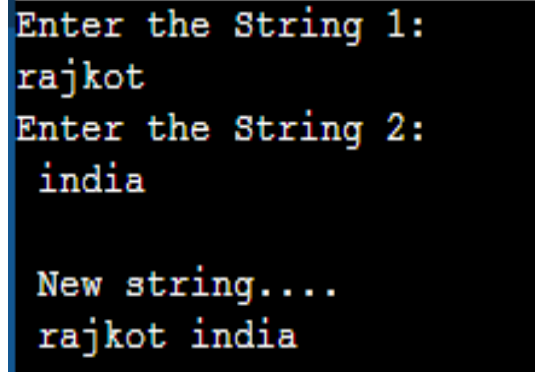
**Q - 20 Explain String Appending Operation and write algorithm and the program.**

- **String Appending Operation**
- Appending a string in existing string means we add new string at end of existing string.
- Text: Original string
- String: New string which is to be appended
- **Algorithm: Append(text, string)**

```
Step 1 [Initialization]
 i ← 0
 l ← 0
Step 2 : [To reach at end of text]
 For(i=0; str1[i]!='\0'; i++)
 l ← l+1
Step 3:[Append a string]
 For(i=0; str2[i]!='\0'; i++)
 Str1[i+ l]← str2[i]
Step 4 : [Print the string after appending]
 Write ("str1")
Step 5 : [Finished] & Exit
```

- **Program:**

```
#include <stdio.h>
#include <string.h>
void main()
{
 char a[100],b[100];
 int i,l=0;
 puts("Enter the String 1:");
 fgets(a,100,stdin);
 puts("Enter the String 2:");
 fgets(b,100,stdin);
 for(i=0; a[i]!='\0'; i++)
 {
 l++;
 }
 for(i=0; b[i]!='\0'; i++)
 {
 a[l+i-1] = b[i];
 }
 a[l+i] = '\0';
 printf("\n New string....\n %s",a);
}
```



```
Enter the String 1:
rajkot
Enter the String 2:
india

New string....
rajkot india
```

**Q - 21 Explain String Reverse Operation and write algorithm and the program.**

- **String Reverse Operation**
- If S1 = "HELLO", then reverse of S1 = "OLLEH". To reverse a string, we just need to swap the first character with the last, second character with the second last character, and so on.
- String1 : Original string
- String2 : Used for storing reverse string
- **Algorithm Reverse(string1)**

|                                                                                                                                                                                                                                                                   |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>Step 1: [Initialization]         i ← 0 Step 2: [Finding the length of the string]         l = LEN(str) Step 3: [Store reverse string from original string]         For(i=i-1;i&gt;=0; i--) repeat             Write("str") Step 4: [Finished]&amp;Exit</pre> |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

- **Program:**

```
#include<stdio.h>
#include<string.h>
void main()
{
 int i,n;
 char str[100];
 printf("Enter the String to get reversed\n");
 gets(str);
 n=strlen(str);
 printf("\nReversed string is... \n");
 for(i=n-1;i>=0;i--)
 {
 printf("%c",str[i]);
 }
}
```

```
Enter the String to get reversed
rajkot india
Reversed string is
aidni tokjar|
```

**Q - 22 Explain String converting character into upper and lower case and write algorithm and the program.**

- **String Reverse Operation**
- We have already discussed that in the memory ASCII codes are stored instead of the real values.
- The ASCII code for A–Z varies from 65 to 91 and the ASCII code for a–z ranges from 97 to 123.
- So if we have to convert a lower case character into uppercase, we just need to subtract 32 from the ASCII value of the character.
- And if we have to convert an upper case character into lower case, we need to add 32 to the ASCII value of the character
- **Algorithm: Upper\_Case(String1, String 2)**

Step 1 : [Initialization]  $I \leftarrow 0$

Step 2 : [To convert lower to upper case]

While ( $i < \text{LEN}(\text{string1})$ )

(i) IF ( $\text{string1}[i] > 'a'$  AND  $\text{string1}[i] <='z'$  )

String2[i] = string1 – 32

(ii)  $I \leftarrow I+1$

Step 3: [Print the upper case string]

Write ("string2")

Step 4 : [Finished]

Exit

- **Program:**

```
#include <stdio.h>
#include <conio.h>
int main()
{
 char str[100], upper_str[100];
 int i=0;
 clrscr();
 printf("\n Enter the string :");
 gets(str);
 while(str[i] != '\0')
 {
 if(str[i]>='a' && str[i]<='z')
 upper_str[i] = str[i] - 32;
 else
 upper_str[i] = str[i];
 i++;
 }
 upper_str[i] = '\0';
 printf("\n The string converted into upper case is : ");
 puts(upper_str);
 return 0;
}
```

**Output**

```
Enter the string : Hello
The string converted into upper case is : HELLO
```