



# MAHARASHTRA STATE BOARD OF TECHNICAL EDUCATION

## Certificate

This is to certify that Mr. / Ms. Savant Omkar Vitthal.....  
Roll No. ....45....., of Third Semester of Diploma in  
.....Computer Technology..... of Institute,  
.....M.V.P.S.'s Rajarshi Shahu Matheraj Polytechnic, Nashik  
(Code: ....1002....) has completed the term work satisfactorily in course  
**Data Structures Using 'C' (22317)** for the academic year 20..20.. to 20..21..  
as prescribed in the curriculum.

Place: ....Nashik....

Enrollment No.:1.a10020362

Date: 9/2/21....

Exam. Seat No: 240630.

Subject Teacher

Head of the Department

Principal



# 1. Program to Perform Operations on Array

Date: / /

Page No.:

Resources Used:

S.No.	Name of Resource	Specification
1	Computer System	Dual Core, 6gb RAM CPU i7, 6GB [S]
2	Software	Torbox C++ IDE gdb [S]
3	Any other Resource	-

\* Algorithm

(W.A)  
(C=loop)

1) Insertion

- 1) Start
  - 2) J=N
  - 3) Repeat step 4 and 5 while  $J \geq K$
  - 4) Set  $I[A[J+1]] = I[A[J]]$
  - 5) Set  $J = J - 1$
- End of step 3 loop
- 6) Set  $I[A[1]] = Item$
  - 7) Reset N
  - 8) Exit

## ii) Deletion

ii) swap

ii) swap J+K

ii) swap top 4 and 5 while J < n

ii) swap  $A[2] = A[1+2]$

ii) for J = J + 1

ii) for K = K - 1

ii) swap  $A[2] = A[1+2]$

ii) swap J = J + 1

ii) swap

→ Framework

Build C  
(array)

→  
Integer Index

Index = 0 to size(C) - 1

Done

No. 2

E=0

Mihiranshu

ii) t

void rotate()

int \*a, int size, int start

{

int i, index = start, size, start;

a = new int;

for (i = index; i < size - 1; i++)

{

a[i] = \*(a + 32)

size--;

size--;

a[i] = \*(a + 32)

start++;

start++;

printf("%d ", \*(a + 32))

}

void insert(int \*a, int size, int start, int pos)

{

i = 0;

a += 32;

for (i = size; i > pos; i--)

{

a[i] = \*(a + 32);

size--;

a[i] = \*(a + 32);

size--;

start++;

```

1. #include <iostream.h>
2. 
3. int main()
4. {
5.     int arr[100], size, i, elem, pos;
6.     cout << "Enter size of array: ";
7.     cin >> size;
8.     cout << "Enter elements: ";
9.     for (i = 0; i < size; i++)
10.    {
11.        cin >> arr[i];
12.    }
13.    cout << "Enter element to insert: ";
14.    cin >> elem;
15.    cout << "Enter position: ";
16.    cin >> pos;
17.    cout << "Enter size: ";
18.    cin >> size;
19.    if (pos == size)
20.    {
21.        cout << "Enter element to delete: ";
22.        cin >> elem;
23.        cout << "Enter index: ";
24.        cin >> pos;
25.        cout << "Wrong choice";
26.    }
27.    else
28.    {
29.        cout << "Wrong choice";
30.    }
31. }

```

### Result 1

Implemented a 'C' program to perform following operation on array:

- 1] Creation
- 2] Insertion
- 3] Display
- 4] Deletion

### Conclusion:

Arrays are used to implement other data structures, such as - Tree, queue, stacks, storage, hash, etc.

### Practical Related Question

I. Consider array size as 10. There are 8 elements in the array. The value 9 is to be inserted at index 6. What will be the output.

II. The array will be full.

III. Consider array size as 10. There are 6 elements in array. The value as index 7 is to be deleted what will happen.

IV. Array will remain as it is after deleting this value.

\* Exercise:

- 1) Perform the five operations following array and show diagrammically of each output given.

0 1 2 3 4 5 6 7 8 9  
10 20 30 40 50 60 70 80 90

- a) Delete element at index 9;  
b) Add element at index 4 with value 90;  
c) Reallocate at index 0.

0 1 2 3 4 5 6 7 8 9  
10 20 30 40 50 60 70 80 90

b2

0 1 2 3 4 5 6 7 8 9  
10 20 30 40 50 60 70 80 90

c1

0 1 2 3 4 5 6 7 8 9  
10 20 30 40 50 60 70 80 90

Insert Array 2 at index 2 of array 1

Array 1:  
0 1 2 3 4 5 6 7 8 9 10  
10 20 30 40 50 60 70 80 90

Array 2:  
0 1 2  
100 200 300

Final Array:  
0 1 2 3 4 5 6 7 8 9 10  
10 20 30 40 50 60 70 80 90

IV. Review Questions:

- A. Apply skills.

- B. Parallel arrays implementation & C++ Language Syntax.

- C. Relocate elements.  
1. Follow rules  
2. Follow references

D. Delete Element(s)

- Searching: Assuming it is an array with size 10, search linearly. If the search is done by searching and found there that particular element is not present in the array, then what will happen?

Michigan State Board of Education

Instructional  
Materials  
Development  
and Revision  
Project

## Search a Data using Linear Search.

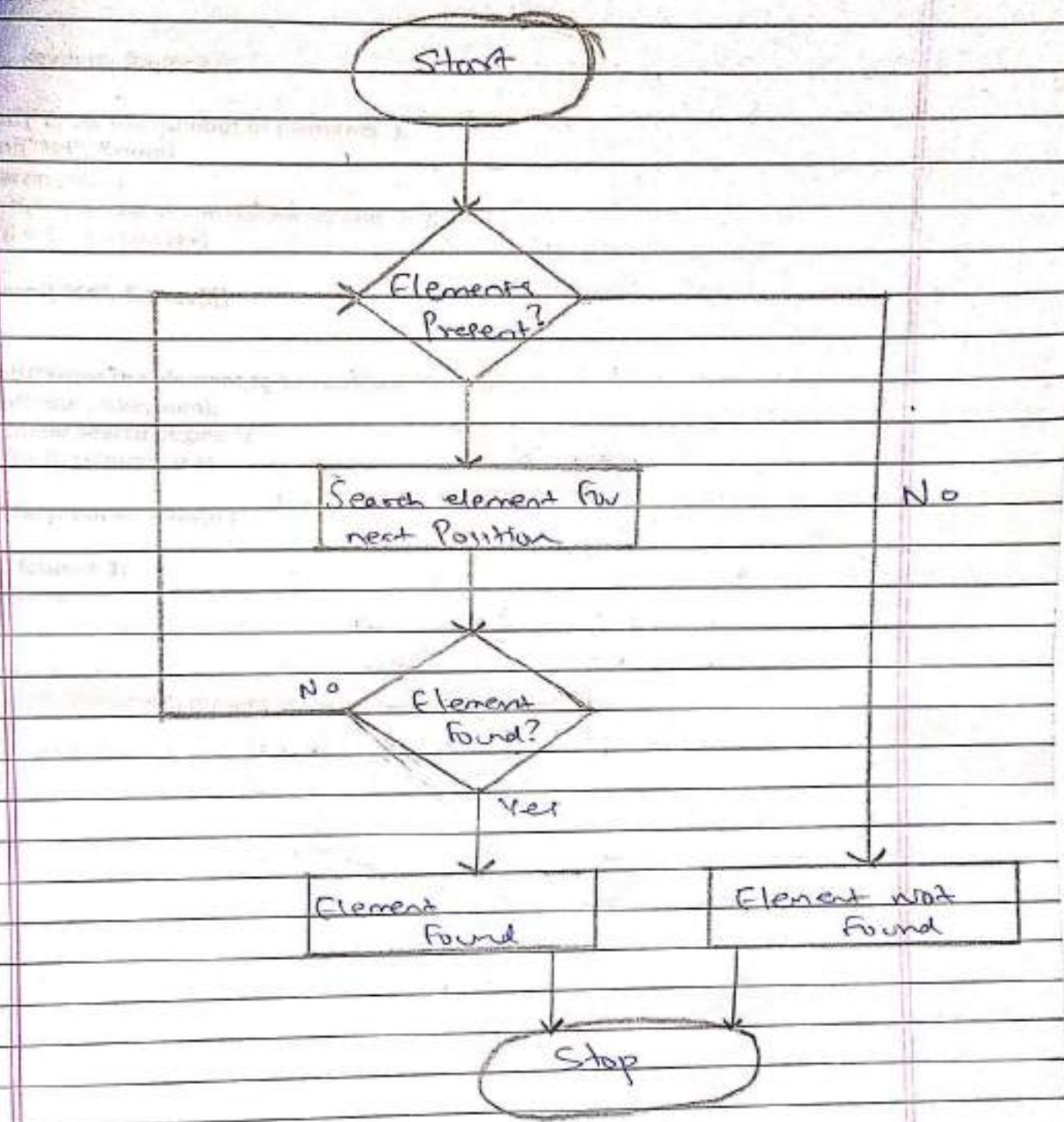
### Resources Required

S.No.	Name of Resource	Specification
1	Computer with Specification	Dual-Case, 6 GB RAM
2	Software	Turbo C
3	Any other Resource	—

### \* Algorithm

- 1] Set  $i = 1$
- 2] Set  $i \leftarrow i + 1$
- 3] if  $i > n$  then go to step 8
- 4] if  $A[i] = x$  then go to step 7
- 5] Set  $i = i + 1$
- 6] Go to step 2
- 7] Print element  $x$  found at index  $i$  and go to step 9
- 8] Print element not found
- 9] Stop

Flowchart



```
<stdio.h>

main()
{
    int num;
    int i, keynum, found = 0;

    printf("Enter the number of elements ");
    scanf("%d", &num);
    int array[num];
    printf("Enter the elements one by one \n");
    for (i = 0; i < num; i++)
    {
        scanf("%d", &array[i]);
    }

    printf("Enter the element to be searched ");
    scanf("%d", &keynum);
    /* Linear search begins */
    for (i = 0; i < num ; i++)
    {
        if (keynum == array[i] )
        {
            found = 1;
            break;
        }
    }
    if (found == 1)
        printf("Element is present in the array at position %d",i+1);
    else
        printf("Element is not present in the array\n");
}
```

## Result :

Implemented a 'C' program to search a particular data from given array using linear search.

### \* Conclusion :-

In order to perform some operation on specific data element, the data has to be searched in data collection, and system requires a searching method. Linear search is one of them.

### \* Practical Related Question.

→ It is most suitable for data list with only a few elements and when performing a single search in an unordered list.

→ The output if list of item having same data item is to be searched is based on its order and preference.

1980

## Exercises

56

i	21	56	36	89	56	31	00	67	59
56		36	89						
05	456	36	89						

4-59  
Wabash and Funeral

56.01  
01-28-20  
not found

1  
1

55	26	89	76	91	00	87	53
58							

5/19/09  
Males not tested

150-4000-  
150-4000-  
150-4000-

III.

### This project

- Wings X

674

八四下

- Technical Circular**  
Volume 1  
May 1961

4

#### **Summary**

- #### **From This**

卷之三

by James R. S.

三

10 of 10

56	36	89	56	6	01	00
					01	56

01 + 56 :  
match not found

55  
O. "O"  
water found  
spec. found

55  
56

55-36  
water at home

55	56	57	58	59	60	61	62	63
56	58	60	56	54	51	50	57	52
56	58	60	56	54	51	50	57	52
55	56	58	56	54	51	50	57	52

56 + 57  
Not Found

Linear Search

55 & 59  
Metal not found.  
Area of linear search  
impossibility.

III.	Concepts This year	(2) State the in terms of the Voronoi search tree
	Develop V.	(2) A Voronoi search tree is a tree structure where a is the target compacted, where a is the target of the tree.
IV.	Different Searches • Apply different Practices	(2) Linear Search is exactly traversal of linear search or exactly traversal and recursion other search algorithms and Search such as the binary search algorithm and hash table also significantly faster searching.
	Relevant Applications 1. Pollution sensors 2. Following particles	
	Mathematical Methods	
	Remarks: Searching is, finding one possible set among them. Binary Search: Binary Search is a search method that divides and conquer problem	

### 3. Search a data using

binary search

Resources Required

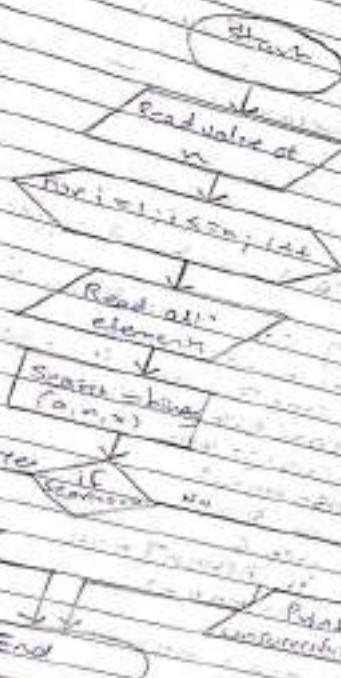
S.No.	Name of Resource	Specification
1	Computer System	Dual-core, 6 GB RAM
2	Software	Turbo C
3	Any other resource	-

#### \* Algorithm

Binary search (A, lower bound, upper bound, VAL)

- 1] [Initialize] Set BEG = Lower bound  
END = upper bound, POS = -1
- 2] Repeat step 3 and 4 while BEG <= END
- 3] Set MID = (BEG + END) / 2
- 4] If A[MID] = VAL  
Set POS = MID  
Else Print POS
- 5] Go to Step 6
- else if A[MID] > VAL  
Set END = MID - 1
- else  
Set BEG = MID + 1

5] IF Pos = -1  
 Print "Value is not present in array"  
 6] Exit  
 \* Flaschbach



```

    void binary_search()
    {
        int a[], n, v, pos;
        cout << "Enter size of an array: ";
        cin >> n;
        cout << "Enter elements of an array in sorted form: ";
        for (int i = 0; i < n; i++)
            cin >> a[i];
        cout << "Enter element to be searched: ";
        cin >> v;
        pos = binary_search(a, n, v);
        if (pos == -1)
            cout << "Element not found";
        else
            cout << "Element found at position " << pos;
    }

    int binary_search(int a[], int n, int v)
    {
        int beg = 0, end = n - 1, mid;
        while (beg <= end) {
            mid = (beg + end) / 2;
            if (a[mid] < v)
                beg = mid + 1;
            else if (a[mid] > v)
                end = mid - 1;
            else
                return mid;
        }
        return -1;
    }
  
```

VII. **Bubble Sort**  
 In Bubble sort:  
 if the array is not sorted then  
 bubble sort is used.  
 swapping of adjacent  
 elements in the array  
 until the array is sorted.  
 process continues  
 until the array is sorted.

Maharashtra State Board of Secondary Education

Result:

Implemented a C program to search a particular data from the given Array using Binary Search.

\* Condition:

In order to perform some operation on specific data element, the data has to be searched in data collection and it requires a searching method.

Binary Search is most commonly used.

\* Practical Related Questions.

1) Binary search is most suitable for what kind of data list?

→ Binary search is most suitable for sorted data list

2) If  $mid = 4.5$  then what should be the value of mid?

$$mid = low + (high - low) / 2$$

$$low = mid + 1$$

$$low = 4.5 + 1 = 5.5$$

$$high = mid * 2 = 4.5 * 2 = 9$$

$$mid = 5.5 + (9 - 5.5) / 2$$

$$mid = 7.25$$

## Exercise

State the benefits and limitation of Binary Search in terms of Time Complexity.

### Benefits:

- 1] It eliminates half of the list from further searching by using the result of each comparison.
- 2] It indicates whether the element being searched is before or after the current position in the list.
- 3] It works better than linear search.

### Limitation:

- 1] It employ recursive approach which require more stack space.
- 2] Caching is poor, because of its random access nature.
- 3] Consider following list and write down steps to perform binary search.

- i] 12
- ii] 67
- iii] 100

Low	23	34	34	45	56	High
05	12	34	34	45	56	67

i]

Data

Sr. No.	0	1	2	3	4	5	6	7	8
1	05	12	34	34	45	56	67	73	
2	05	12	34	7	45	56	67	06	3
3								62	1

The element 12 is found at location 2

Data

Sr. No.	0	1	2	3	4	5	6	7	8
1	05	12	34	34	45	56	67	06	3
2								45	56
3								45	56
4								67	06

The element 57 is found at location 6

Data

Sr. No.	0	1	2	3	4	5	6	7	8
1	05	12	34	34	45	56	67	06	3
2								45	56
3								45	56
4								45	56
5								45	56

The element 12 is not found

#### IV. Radix Sort

- + Advantages

#### V. Partition

- + Advantages

#### VI. Selection Sort

- 1. Selection

#### VII. Insertion Sort

- 2. Insertion

#### VIII. Merge Sort

In bubble sort we compare adjacent elements and if they are not in order, we swap them. In selection sort we compare first and last elements and if they are not in order, we swap them. In insertion sort we compare current element with previous elements and if they are not in order, we swap them. In merge sort we divide the array into two halves and then merge them back again. This is a recursive process.

## Program To Sort an Array Using Bubble Sort.

### Resources Required

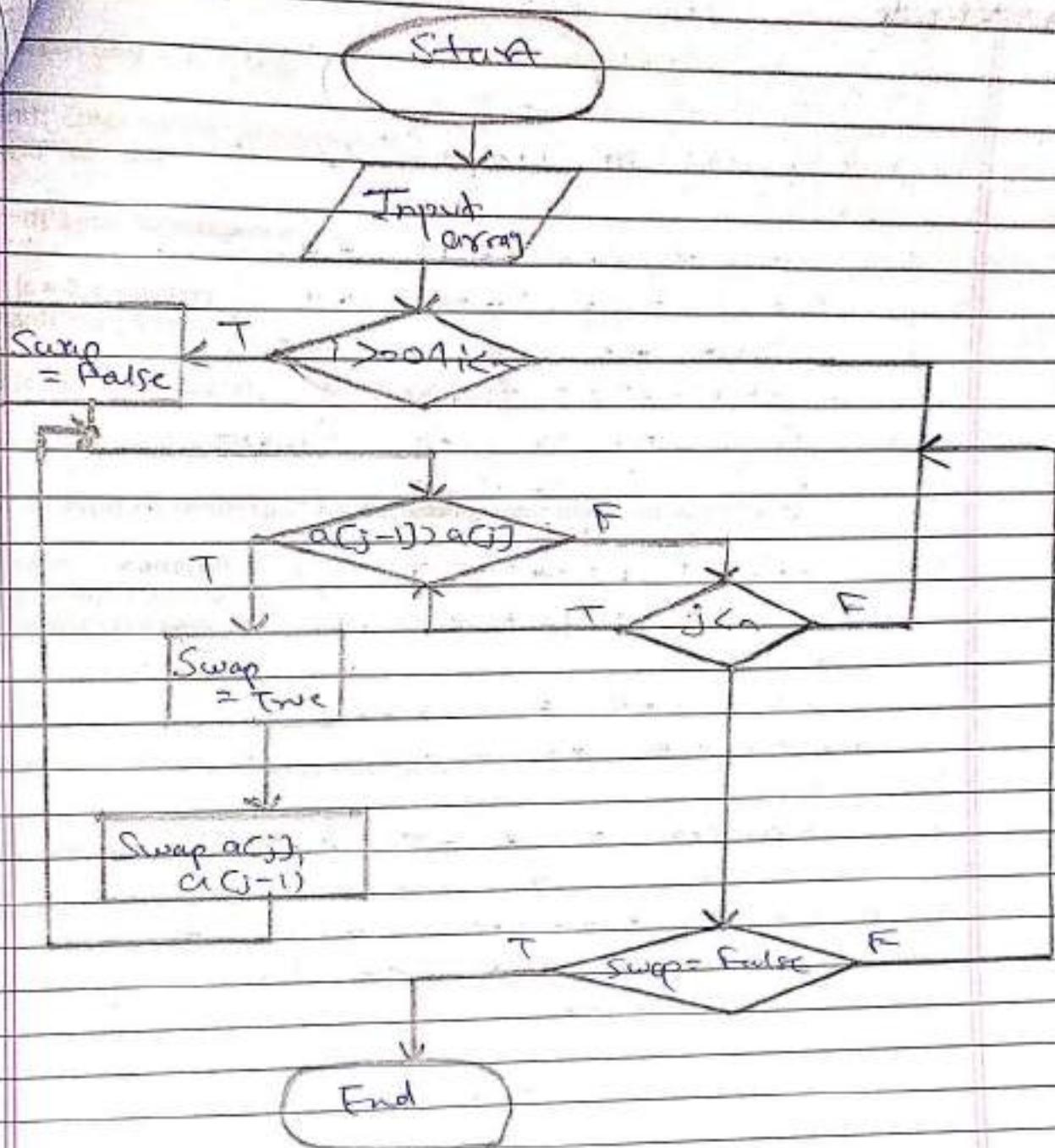
S.no.	Name of Resource	Specifications
1	Computer System with Specification	Dual - Core, 6gb RAM 500 GB HDD
2	Software	Turbo C
3	Any other	—

### \* Algorithm

- 1] Start
- 2] begin BubbleSort (list)
- 3] for all elements of list
- 4] if list [i] > list [i+1]
- Swap (list [i], list [i+1])
- 5] End if
- 6] End for
- 7] return list
- 8] end BubbleSort
- 9] Stop

Flowchart

Date: / /  
Page No.:



```
#include <stdio.h>

main()
{
    int array[100], n, c, d, swap;

    printf("Enter number of elements\n");
    scanf("%d", &n);

    printf("Enter %d integers\n", n);

    for (c = 0; c < n; c++)
        scanf("%d", &array[c]);

    for (c = 0 ; c < n - 1; c++)
    {
        for (d = 0 ; d < n - c - 1; d++)
        {
            if (array[d] > array[d+1]) /* For decreasing order use '<' instead of '>' */
            {
                swap      = array[d];
                array[d]  = array[d+1];
                array[d+1] = swap;
            }
        }
    }

    printf("Sorted list in ascending order:\n");

    for (c = 0; c < n; c++)
        printf("%d\n", array[c]);

    return 0;
}
```

Result is obtained in the following

Implemented a 'C' program to sort an array using Bubble Sort method.

### \* Conclusion :

In order to perform some operations on specific data element, the data has to be sorted in data collection, and system requires a sorting method. One of most commonly used method is Bubble Sort.

### \* Practical Related Questions :

1] What is output of following code?

```
For (i=0; i<=n-1, i++)
{
    For (j=0, j<=n-i; j++)
    {
        if (a[j]>a[i+1])
        {
            temp = a[i]
            a[i] = a[i+1]
            a[i+1] = temp;
        }
    }
}
```

→ It's the code for Bubble  
procedure actual Sorting

2) Find example C language from the  
code:

```
for(i=0; i<n-1; i++)
    for(j=1; j<n-i; j++)
        if(a[j] < a[i])
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
```

→ No error's found

Exercise

1) Find the number of Comparisons required  
in Bubble sort of the following list  
having 5 members.

10 22 38 42 50

Sorted  
Comparisons required, list is already

Sort the given array in ascending order using  
Bubble Sort and three steps

1000, -20, 300, 100, 50

-20 1000 300 100 50  
1000 1000 300 100 50  
-20 300 100 50 1000  
1000 100 50 1000 300

300 100 50 1000 1000  
1000 300 100 50 1000  
-20 300 100 50 1000  
1000 300 100 50 1000  
-20 100 50 300 1000

1000 100 50 300 1000  
1000 100 50 300 1000  
-20 100 50 300 1000  
1000 100 50 300 1000  
-20 100 50 300 1000

1000 100 50 300 1000

#### Minimum Theoretical Background

Selection sort is similar to the hand picking up  
Fruit in the first position and the second position.  
We follow the following steps to perform selection sort.

1. Start from the first element in the array and look  
array.
2. Swap it with the value in the first position
3. Repeat the steps above for the remainder of the list  
(and Advancing each time)

3) Sort the given array using ascending order using bubble sort and show steps  
500, 120, 2000, -210, 89

→

I]

500 120 2000 -210 89

120 500 2000 -210 89

120 500 -210 2000 89

120 500 -210 89 2000

120 500 -210 89 2000

II]

120 500 -210 89 2000

120 -210 500 89 2000

120 -210 89 500 2000

120 -210 89 500 2000

III]

-210 120 500 89 2000

-210 120 89 500 2000

-210 89 120 500 2000

-210 89 120 500 2000

the list is sorted.

bubble sort with 5 steps

## Program to sort an array Using Selection Sort.

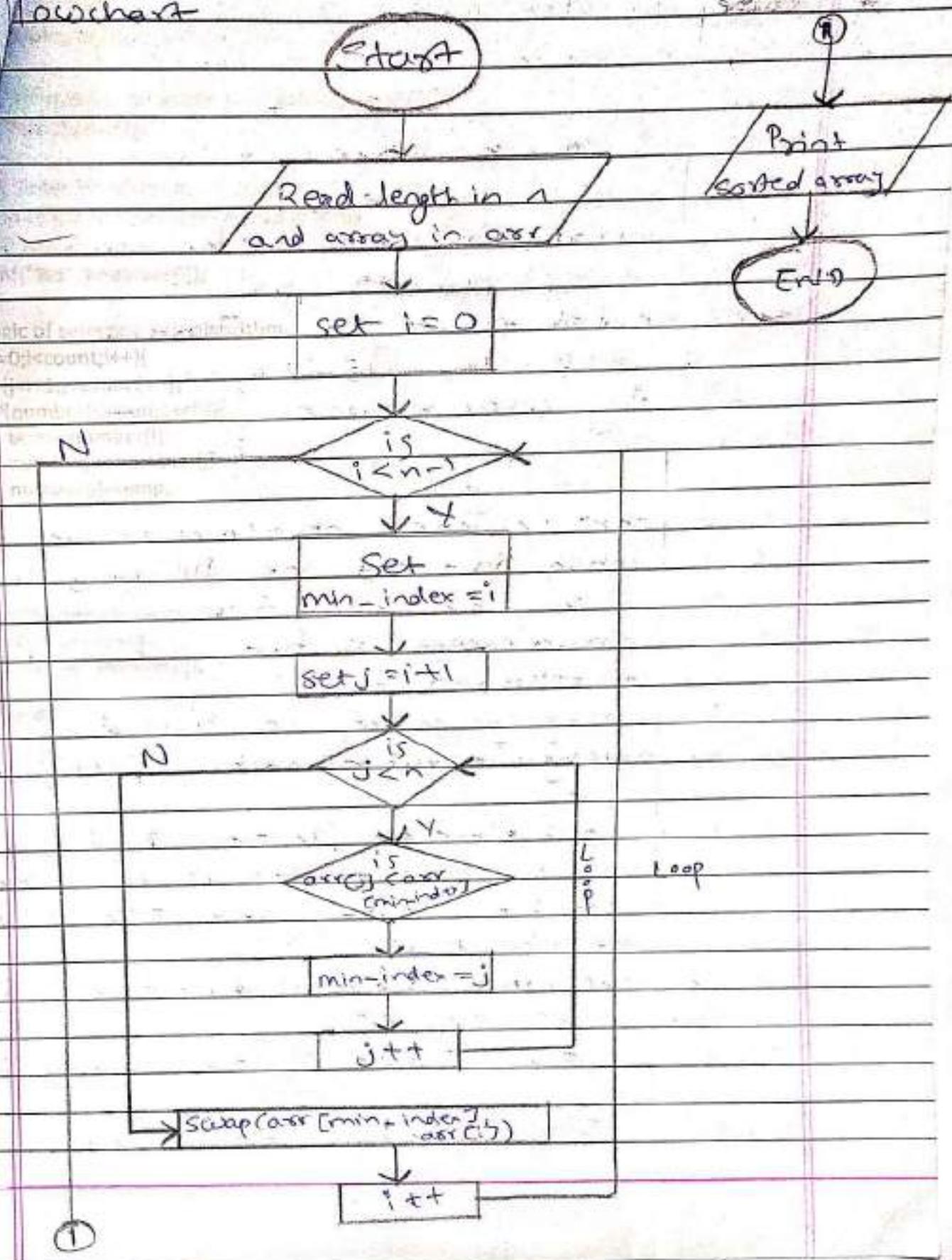
Example Used:

No.	Name of Resource	Specification
1	Computer System	Dual-core, 6GB RAM
2	Software	Turbo C
3	Any other resource	—

\* Algorithm:

- 1] Start
- 2] Set MIN to location 0
- 3] Search the minimum element in the list.
- 4] Swap with value at location MIN
- 5] Increment MIN to point to next element.
- 6] Repeat until list is sorted
- 7] Stop

owchart



#include <stdio.h>

int count, temp, number[25];

"How many numbers u are going to enter?: ";  
("%d",&count);

if("Enter %d elements: ", count);  
Loop to get the elements stored in array  
for(i=0;i<count;i++)  
scanf("%d",&number[i]);

// Logic of selection sort algorithm

```
for(i=0;i<count;i++){  
    for(j=i+1;j<count;j++){  
        if(number[i]>number[j]){  
            temp=number[i];  
            number[i]=number[j];  
            number[j]=temp;  
        }  
    }  
}  
  
printf("Sorted elements: ");  
for(i=0;i<count;i++)  
    printf(" %d",number[i]);
```

return 0;

}

## Result :

Implemented a 'C' program to sort an array using Selection Sort method.

### \* Conclusion :

In order to perform some operation on specific data element, the data needs to be sorted. One of the most commonly used method is Selection Sort.

### \* Practical Related Questions :

1] A Selection Sort compare adjacent elements, and swap them if they are in wrong order.

- a. True
- b. False
- c. Depend on element
- d. None of these.

→ a. True.

2] For each from 1 to  $n-1$ , there are exchanges for Selection Sort.

- a. 1
- b.  $n$
- c.  $n-1$
- d. none of these

→ a. 1

1 Sort

2

3

4

5

6

7

3) What is the output of Selection Sort  
given numbers:  
20, 12, 15, 13, 2

0, 2, 15, 12, 10, 20  
16, 2, 10, 12, 15, 20  
2, 2, 12, 10, 15, 20

4) Name of above

→ 6, 12, 10, 15, 20

\* Exercise

1) Find the number of comparisons  
in Selection Sort of the following list.

10, 20, 30, 40, 50

→ 2) Sort the given array in ascending  
order using Insertion Sort.

1000, -20, 300, 150, 50  
1000, -20, 300, 150, 50  
-20, 1000, 300, 150, 50  
-20, 50, 300, 150, 1000, 7  
-20, 50, 150, 300, 1000, 7

→ Array has been sorted.  
Sort the given array in ascending order  
using Insertion Sort.

→ 500, 120, 200, -110, 30  
500, 120, 200, -110, 30  
-110, 120, 200, 500, 30  
-110, 120, 200, 500, 30  
30, -110, 120, 200, 500, 30  
30, -110, 120, 200, 500, 30

→ → → → → →  
the array has been sorted

Relevant:

• No.

Practical:

• High

VII. Relevant:

1. Follows

2. Follows

VIII. Minimum Time:

Insertion Sort in

distinct elements

Consider we use p

## Program to sort an array using insertion sort.

Date: / /  
Page No.:

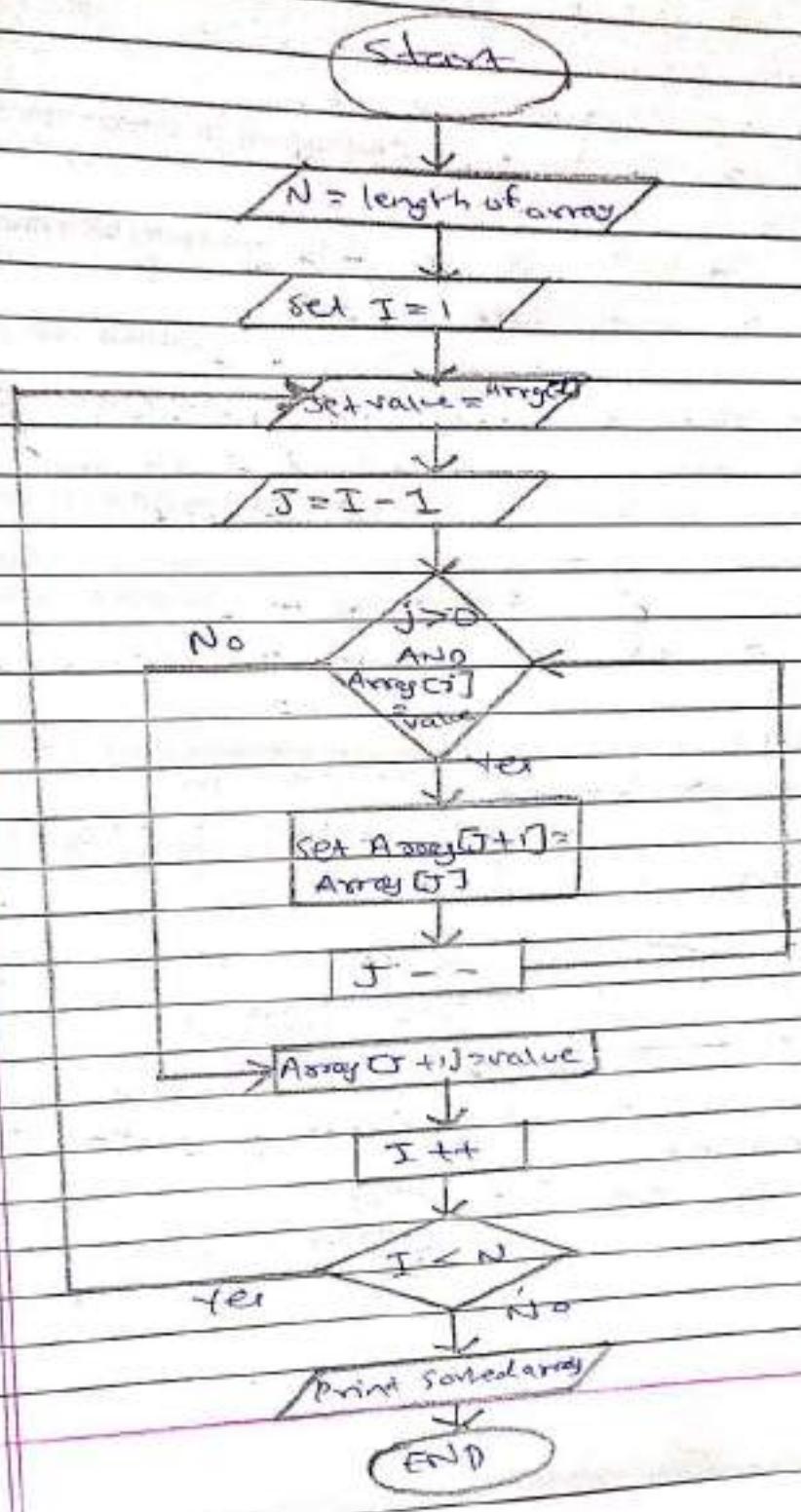
### Resources Used:

Sr.No.	Name of Resource	Specification
1	Computer System	Dual-core, 6gb RAM
2	Software	Turbo C
3	Any other resource	-

### Algorithm :

- 1] Start
- 2] If the element is first, and is already sorted, return 1;
- 3] Pick next element
- 4] Compare with all elements in the sorted sub-list
- 5] Shift all elements in the sorted sub-list that is greater than the value to be sorted.
- 6] Insert the value
- 7] Repeat until list is sorted
- 8] Stop

## Flowchart



de

```
#include <stdio.h>
main()
{
    int n, i, j, temp;
    int arr[64];

    printf("Enter number of elements\n");
    scanf("%d", &n);

    printf("Enter %d integers\n", n);
    for (i = 0; i < n; i++)
    {
        scanf("%d", &arr[i]);
    }
    for (i = 1; i <= n - 1; i++)
    {
        j = i;
        while (j > 0 && arr[j-1] > arr[j])
        {
            temp = arr[j];
            arr[j] = arr[j-1];
            arr[j-1] = temp;
            j--;
        }
    }
    printf("Sorted list in ascending order:\n");
    for (i = 0; i <= n - 1; i++)
    {
        printf("%d\n", arr[i]);
    }
    return 0;
}
```

Result :

Implemented a 'C' program to sort an array using Insertion Sort method.

→ ~~Qn~~ Conclusion :

In order to perform some operation on data, the data has to be sorted. One of the most commonly used method is Insertion Sort for sorting data in list.

→ Practical Related Questions :

1) Define Insertion Sort.

→ Insertion Sort is a simple sorting algorithm that builds the final sorted array one item at a time. It is much less efficient on large lists.

2) Differentiate between Bubble and Insertion Sort.

Bubble Sort	Insertion Sort
Check neighbour elements and swaps them	Transfers an element at a time to sort
More number of Swaps	Less number of Swaps
Slower than insertion sort	Twice as fast as bubble sort

Date:		Page No.:	
Score the game away using Toss coin Sort			
and show steps.			
1650 -70 300 100 50			→
105 -70 300 100 50			→
100 200 100 50			→
100 200 100 50			→
50 50 50 50			→
25 25 25 25			→
25 25 25 25			→
125 125 125 125			→
125 125 125 125			→
62 62 62 62			→
31 31 31 31			→
15 15 15 15			→
7 7 7 7			→
3 3 3 3			→
1 1 1 1			→
0 0 0 0			→

1) Insertion Sort is  
 a) In-place      b) Stable  
 c) Out-place     d) Unstable

2)  $1, 1$  and  $2$      $2, 3$  and  $4$

3)  $a, b$        $b, c$   
 a) Bubblesort      b) Insertionsort  
 c) Selectionsort     d) Quicksort

4) Table number of Comparisons for Insertionsort is  
 a)  $n^2$       b)  $n$   
 c)  $n + 1$       d)  $n - 1$

5) a)  $n^2/2$       b)  $n^2$   
 c)  $n$       d)  $n^2 + n/2$

6) True  
 7) Find the number of Comparisons required for Insertionsort of following given list having 5 numbers  
 10, 12, 30, 50, 20 →  
 8) Passes

- I. Every element is compared with all other elements.
- II. Only adjacent elements are compared.
- III. It is comparison based sort.
- IV. It is stable sort.
- V. It is not a divide and conquer algorithm.
- VI. It has time complexity of  $O(n^2)$ .
- VII. It is not an in-place sorting algorithm.
- VIII. It can be implemented using recursion.
- IX. It is a slow sorting algorithm.

8) Which of the following statements is true?  
 a) It is a comparison based sort.  
 b) It is not a divide and conquer algorithm.  
 c) It is not a stable sort.  
 d) It has time complexity of  $O(n^2)$ .

9) If  $n$  is the size of array then the time complexity of Insertion sort is  
 a)  $O(n^2)$       b)  $O(n)$   
 c)  $O(n^3)$       d)  $O(1)$

10) What is the best case time complexity of Insertion sort?  
 a)  $O(n^2)$       b)  $O(n)$   
 c)  $O(n \log n)$       d)  $O(n^3)$

11) What is the worst case time complexity of Insertion sort?  
 a)  $O(n^2)$       b)  $O(n)$   
 c)  $O(n \log n)$       d)  $O(n^3)$

12) The number of comparisons required for Insertionsort of 100 elements will be  
 a)  $O(n^2)$       b)  $O(n)$   
 c)  $O(n \log n)$       d)  $O(n^3)$

13) If  $n$  is the size of array then the time complexity of Selection sort is  
 a)  $O(n^2)$       b)  $O(n)$   
 c)  $O(n \log n)$       d)  $O(1)$

14) The number of comparisons required for Selection sort of 100 elements will be  
 a)  $O(n^2)$       b)  $O(n)$   
 c)  $O(n \log n)$       d)  $O(n^3)$

15) If  $n$  is the size of array then the time complexity of Bubblesort is  
 a)  $O(n^2)$       b)  $O(n)$   
 c)  $O(n \log n)$       d)  $O(1)$

16) The number of comparisons required for Bubblesort of 100 elements will be  
 a)  $O(n^2)$       b)  $O(n)$   
 c)  $O(n \log n)$       d)  $O(n^3)$

17) If  $n$  is the size of array then the time complexity of Quicksort is  
 a)  $O(n^2)$       b)  $O(n)$   
 c)  $O(n \log n)$       d)  $O(1)$

18) The number of comparisons required for Quicksort of 100 elements will be  
 a)  $O(n^2)$       b)  $O(n)$   
 c)  $O(n \log n)$       d)  $O(n^3)$

19) If  $n$  is the size of array then the time complexity of Mergesort is  
 a)  $O(n^2)$       b)  $O(n)$   
 c)  $O(n \log n)$       d)  $O(1)$

20) The number of comparisons required for Mergesort of 100 elements will be  
 a)  $O(n^2)$       b)  $O(n)$   
 c)  $O(n \log n)$       d)  $O(n^3)$

21) If  $n$  is the size of array then the time complexity of Heapsort is  
 a)  $O(n^2)$       b)  $O(n)$   
 c)  $O(n \log n)$       d)  $O(1)$

22) The number of comparisons required for Heapsort of 100 elements will be  
 a)  $O(n^2)$       b)  $O(n)$   
 c)  $O(n \log n)$       d)  $O(n^3)$

23) If  $n$  is the size of array then the time complexity of shellsort is  
 a)  $O(n^2)$       b)  $O(n)$   
 c)  $O(n \log n)$       d)  $O(1)$

24) The number of comparisons required for shellsort of 100 elements will be  
 a)  $O(n^2)$       b)  $O(n)$   
 c)  $O(n \log n)$       d)  $O(n^3)$

25) If  $n$  is the size of array then the time complexity of quick selection is  
 a)  $O(n^2)$       b)  $O(n)$   
 c)  $O(n \log n)$       d)  $O(1)$

26) The number of comparisons required for quick selection of 100 elements will be  
 a)  $O(n^2)$       b)  $O(n)$   
 c)  $O(n \log n)$       d)  $O(n^3)$

27) If  $n$  is the size of array then the time complexity of cocktail sort is  
 a)  $O(n^2)$       b)  $O(n)$   
 c)  $O(n \log n)$       d)  $O(1)$

28) The number of comparisons required for cocktail sort of 100 elements will be  
 a)  $O(n^2)$       b)  $O(n)$   
 c)  $O(n \log n)$       d)  $O(n^3)$

29) If  $n$  is the size of array then the time complexity of bucket sort is  
 a)  $O(n^2)$       b)  $O(n)$   
 c)  $O(n \log n)$       d)  $O(1)$

30) The number of comparisons required for bucket sort of 100 elements will be  
 a)  $O(n^2)$       b)  $O(n)$   
 c)  $O(n \log n)$       d)  $O(n^3)$

31) If  $n$  is the size of array then the time complexity of radix sort is  
 a)  $O(n^2)$       b)  $O(n)$   
 c)  $O(n \log n)$       d)  $O(1)$

32) The number of comparisons required for radix sort of 100 elements will be  
 a)  $O(n^2)$       b)  $O(n)$   
 c)  $O(n \log n)$       d)  $O(n^3)$

33) If  $n$  is the size of array then the time complexity of count sort is  
 a)  $O(n^2)$       b)  $O(n)$   
 c)  $O(n \log n)$       d)  $O(1)$

34) The number of comparisons required for count sort of 100 elements will be  
 a)  $O(n^2)$       b)  $O(n)$   
 c)  $O(n \log n)$       d)  $O(n^3)$

35) If  $n$  is the size of array then the time complexity of bitonic sort is  
 a)  $O(n^2)$       b)  $O(n)$   
 c)  $O(n \log n)$       d)  $O(1)$

36) The number of comparisons required for bitonic sort of 100 elements will be  
 a)  $O(n^2)$       b)  $O(n)$   
 c)  $O(n \log n)$       d)  $O(n^3)$

37) If  $n$  is the size of array then the time complexity of bitonic sort is  
 a)  $O(n^2)$       b)  $O(n)$   
 c)  $O(n \log n)$       d)  $O(1)$

38) The number of comparisons required for bitonic sort of 100 elements will be  
 a)  $O(n^2)$       b)  $O(n)$   
 c)  $O(n \log n)$       d)  $O(n^3)$

39) If  $n$  is the size of array then the time complexity of bitonic sort is  
 a)  $O(n^2)$       b)  $O(n)$   
 c)  $O(n \log n)$       d)  $O(1)$

40) The number of comparisons required for bitonic sort of 100 elements will be  
 a)  $O(n^2)$       b)  $O(n)$   
 c)  $O(n \log n)$       d)  $O(n^3)$

Q] What is output of Insertions Sort after the 2nd iteration given the following sequence of numbers?

7 3 5 1 9 8 4 6

7 3 5 1 9 8 4 6

1 7 3 5 9 8 4 6 [I]  
1 1 7 3 5 9 8 4 6  
Sorted                  Unsorted

1 3 7 5 9 8 4 6 [II]  
1 1 7 3 5 9 8 4 6  
Sorted                  Unsorted

#### \* Assessment Scheme

100

20%  
20%

20% 20% 20% 20%

10% 10%

10% 10%

20% 20% 20% 20%

10% 10%

10% 10%

10% 10% 10% 10%

10% 10% 10% 10%

# 7. Perform Push and Pop Operation on Stack

Date \_\_\_\_\_

Page No. \_\_\_\_\_

## \* Resources Used :

Sr.no	Name of Resource	Specifications
1	Computer System	Dual core, 6gb RAM
2	Software	Turbo C
3	Any other	—

## \* Algorithm

### i) PUSH

- [1] Check if the stack is full.
- [2] If the stack is full, produces an error & exit.
- [3] If the stack is not full, increment top to point next empty step.
- [4] Add data element to the stack location where top is pointing.
- [5] Return success.

### i) POP

- 1] check if stack is empty
- 2] If the stack is empty, produce an error.
- 3] If the stack is not empty, remove data element at which top is.
- 4] Decrease the value of top by 1.
- 5] Return Success

### ii) Flushout

#### D) Push()

(Push())

Push element,  
St. Pol. box

M = St\_Police + len(M) \* C \* 1000 + 1000

M → M + C

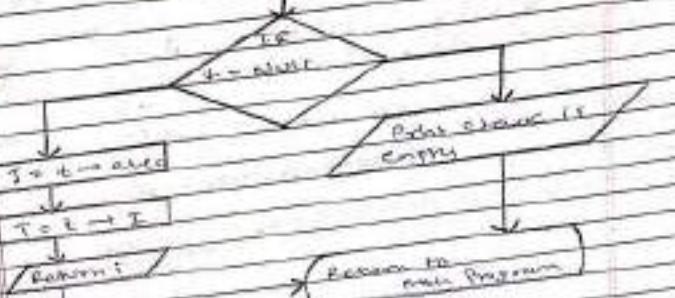
M → M + E

M → M + T

Return to  
main program

### ii) pop (C)

(for c)



## Code C

```
#include<stdio.h>
#include<process.h>
#include<stdlib.h>

define MAX 5 //Maximum number of elements that can be stored

int top=-1,stack[MAX];
void push();
void pop();
void display();

void main()
{
    int ch;
    while(1) //Infinite loop, will end when choice will be 4
    {
        printf("\n*** Stack Menu ***");
        printf("\n\n1.Push\n2.Pop\n3.Display\n4.Exit");
        printf("\nEnter your choice(1-4):");
        scanf("%d",&ch);

        switch(ch)
        {
            case 1: push(); break;
            case 2: pop(); break;
            case 3: display(); break;
            case 4: exit(0);
            default: printf("\nWrong Choice!!");
        }
    }

    void push()
    {
        int val;
        if(top==MAX-1)
        {
            printf("\nStack is full!!!");
        }
        else
        {
```

Date: \_\_\_\_\_  
Page No.: \_\_\_\_\_

Result :-  
Implemented basic operations to create  
'C' program to perform Push and Pop  
operations on stack array.

#### Conclusion :-

In order to perform insertion and deletion  
of data item from given data list based  
on last in First Out (LIFO) suitable  
data structure is required. So Stack is  
structure having below properties.

#### Practical Related Question

Q1) Draw stack after POP operation using given  
diagram.

→ POP [ ]

Ans:

IV.				
V.				
VI.		60		15
VII.		15		
VIII.	1			
	2			
	3			
	4			

2) Draw stack after Push operation  
given Stack

Push(45)  
Ans:

30		45
60		30
15		60

Exercise Related Questions  
T-1

3) Write a TAF value after Push operation

Push(45)  
Ans:

30		45	← Top
60		30	
15		60	

- IV. Re  
V. Pre  
VI. Motiv  
I. Disk  
VII. Minim  
A que  
only occ  
residua



2] Implement a 'c' program for stack size-8  
Ex:-

push(10), push(20), pop; push(10), push(20),  
pop, pop, pop, push(20), pop and  
and draw final output.



#include <stdio.h>

int MAXSIZE = 8;

int Stack[8];

int top = -1;

int isempty() {

if (top == -1)

return 1;

else

return 0;

}

int isfull() {

if (top == MAXSIZE)

return 1;

else

return 0;

}

int peek() {

return Stack[top];

}

```

int pop() {
    int data;
    if (!is_empty())
        data = stack[top];
    top = top - 1;
    return data;
}
3 print("Stack is empty");
4 set push(int data);
5 if (!is_full())
    top = top + 1;
    stack[top] = data;
6 else
7     print("Stack is full");

```

```

int main()
{
    push(1);
    push(2);
    top();
    push(3);
    pop();
    top();
}

```

Date \_\_\_\_\_  
Page No. \_\_\_\_\_

```

push();
pop();
print("Element at top of stack is");
print(stack[0]);
print("Element is");
while (!is_empty())
    int data = pop();
    print("Deleted int", data);
    print("Stack");
    final output:
Empty Stack

```

III.	Q	D
IV.	Stack	
V.	Print	
VI.	Balance	
VII.	Illustration	

1 - 3

stack  
sc

```
"C:\Users\Orange\Desktop\manual dsu\bin\Debug\manual dsu.exe"
Element at top of the stack: 78
Elements: 78
106
Stack full: false
Stack empty: true
Process returned 0 (0x0) execution time : 0.992 s
Press any key to continue . . .
```

8. Perform insert and delete operations on linear queue using array.

Date:	/	/
Page No.:		

Resources Used:

Sr.No	Name of Resource	Specifications
1	Computer System	Dual-core, 6 gb RAM
2	Software	Turbo C
3	Any other	—

\* Algorithm

i) Insertion

1] IF Rear = Max - 1

    write Overflow

    Go to step

    [End of IF]

2] IF Front = -1 and Rear = -1

    Set Front = Rear = 0

    else

        Set Rear = Rear + 1

    [End of IF]

3] Set Queue [Rear] = Num

4] Exit

## ] Deletion

1] If  $\text{front} = -1$  or  $\text{Front} > \text{Rear}$ 

Write Underflow

else

Set  $\text{VAL} = \text{Queue}[\text{Front}]$ Set  $\text{Front} = \text{Front} + 1$ 

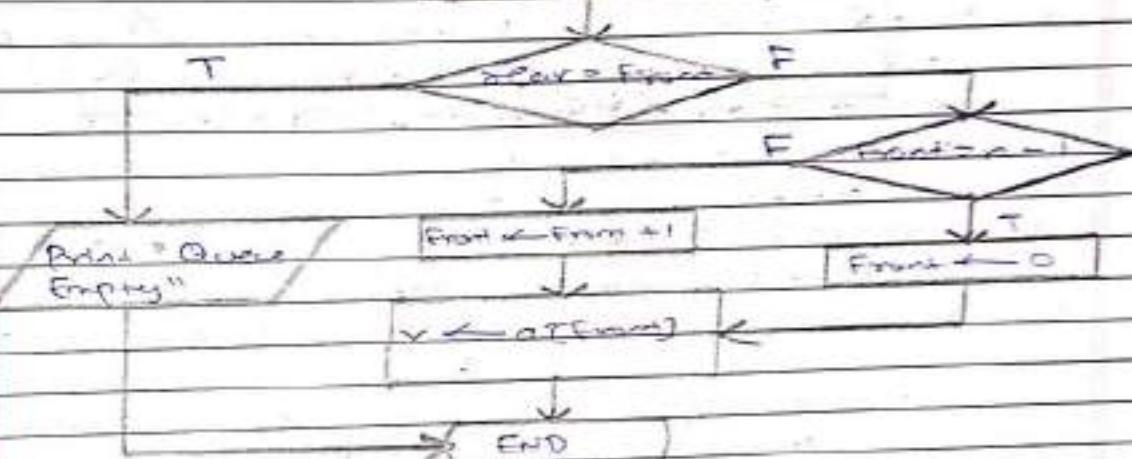
[End of IF]

2] Exit

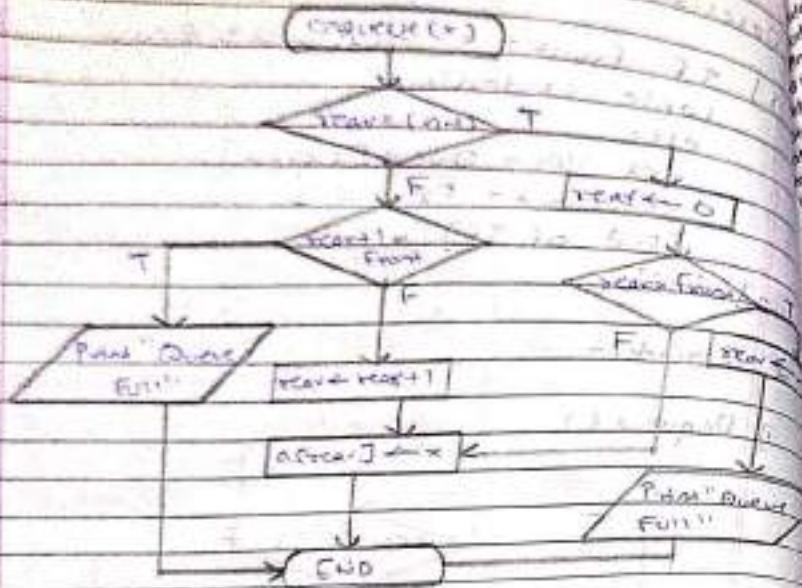
## \* Flowchart

i) Dequeue()

(dequeue(x))



## 2) Enqueue()



Code

```

int studio();
int studio();
int MAX = 50;
int front;
int rear;
int queue[MAX];
int rear = 0;
int front = 0;
int max;
int choice;
while (1)
  
```

```

    cout<<"1.Insert element to queue n";
    cout<<"2.Delete element from queue n";
    cout<<"3.Display all elements of queue n";
    cout<<"4.Quit n";
    cout<<"Enter your choice : ";
    scanf("%d", &choice);
    switch(choice)
    {
  
```

```

      case 1:
        insert();
        break;
      case 2:
        delete();
        break;
      case 3:
        display();
        break;
      case 4:
        exit();
        default:
        cout<<"Wrong choice n";
    }
    void insert()
    {
        int item;
        if(rear == MAX - 1)
            cout<<"Queue Overflow n";
        else
            front++;
        cout<<"front = ";
        cout<<front<<endl;
    }
  
```

insert the element in queue : 7  
and '8', item  
rear = rear + 1;  
queue, arr[front] = item;

add item

(front == -1) front = rear;

print("Queue is full");

else

print("Queue deleted from front : ", arr[front], queue, arr[front]);  
front = front + 1;

return;

else

front == -1

print("Queue is empty");

else

print("Queue is empty");

try < front < rear; i++  
arr[front] = queue, arr[i];  
front = front + 1;

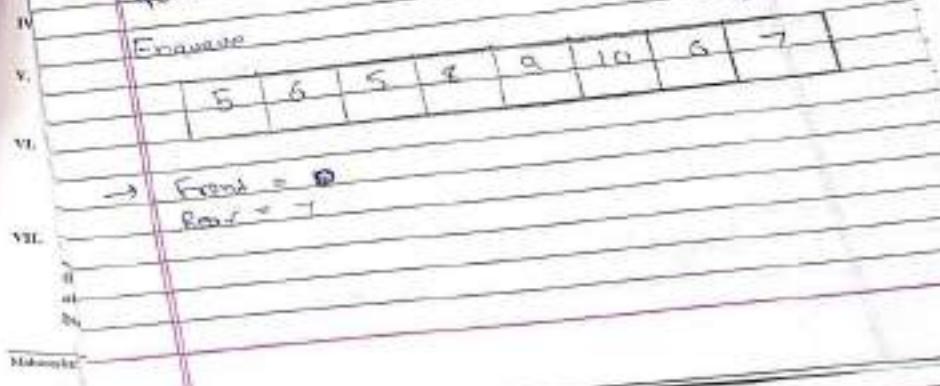
Result :  
Implemented basic 6 programs to perform  
Insert and Delete operations on Queue  
queue using Array.

Conclusion :

Linear Queue is collection of elements  
that follows FIFO order which keeps  
track of two indices, front and rear  
one for Insert and another for  
Delete.

\* Practical Related Questions

1. write a front and rear value of linear  
queue.



Q1 Sketch the front and rear in empty queue.

→ [0] [1] [2] [3] [4] [5] [6] [7]

Front = -1

Rear = -1

a) Sketch the Front and rear in this queue

→ [27] [19] [17] [7] [ ] [ ] [ ] [ ]

Front = 0

Rear = 3

#### \* Exercise

1) Implement a C program to Linear Queue in array size 5 with this Operations Insert(25), Insert(55), Insert(14), delete, delete, Insert(40).

→ #include <stdio.h>

struct Queue {

int front, rear, capacity;  
int \* queue;

C:\Users\Orange\Desktop\manual dsu\bin\Debug\manual dsuexe

```
25 enqueued to queue
35 enqueued to queue
14 enqueued to queue
25 dequeued from queue
35 dequeued from queue
40 enqueued to queue
Front item is 14
Rear item is 40
Process returned 0 (0x0)
Press any key to continue...
```

"C:\Users\Orange\Desktop\manual dsu\bin\Debug\manual dsu.exe"

```
10 enqueued to queue
40 enqueued to queue
10 dequeued from queue
40 enqueued to queue
10 enqueued to queue
40 dequeued from queue
Front item is 40
Rear item is 40
```

```
Process returned 0 (0x0)  execution time : 0.666 s
```

```
#include<stdio.h>
int size=2;
int queue[size],front=-1,rear=-1;
void insert(int item){
    if(rear==size-1){
        printf("Queue is full \n");
    }
    else{
        if(front==-1){
            front=0;
        }
        rear=rear+1;
        queue[rear]=item;
        printf("Element Inserted %d\n",item);
    }
}
void delete(){
    if(front==-1){
        printf("Queue is Empty\n");
    }
    else{
        printf("Element is deleted %d\n",queue[front]);
        front=front+1;
        if(front>rear){
            front=-1;
            rear=-1;
        }
    }
}
void display(){
    if(rear== -1)
        printf("Queue is empty\n");
    else{
        int i;
        printf("Queue:");
        for(i=front;i<=rear;i++)
            printf("%d",queue[i]);
    }
}
int main()
{
    display();
    insert(1);
    insert(2);
    delete();
```

9. Perform insert and delete operations on Circular Queue using array.

\* Resources Used :

Sr. No.	Name of Resource	Specification
1	Computer System	Dual core, 6gb RAM
2	Software	Type C
3	Any other	-

\* Algorithm.

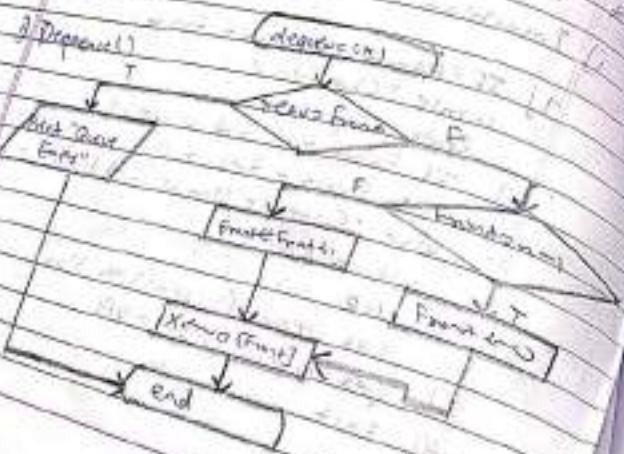
i) Insertion

- 1] IF  $(rear+1) \% \text{Max} = \text{Front}$   
Write "Overflow"  
Goto Step 4
- 2] IF  $\text{Front} = -1$  and  $\text{rear} = -1$   
Set  $\text{front} = \text{rear} = 0$   
else if  $\text{rear} = \text{Max}-1$  and  $\text{front} = 0$   
Set  $\text{rear} = 0$   
else  
Set  $\text{rear} = (\text{rear}+1) \% \text{Max}$
- 3] Set  $\text{arr}[\text{rear}] = \text{VAL}$
- 4] Exit

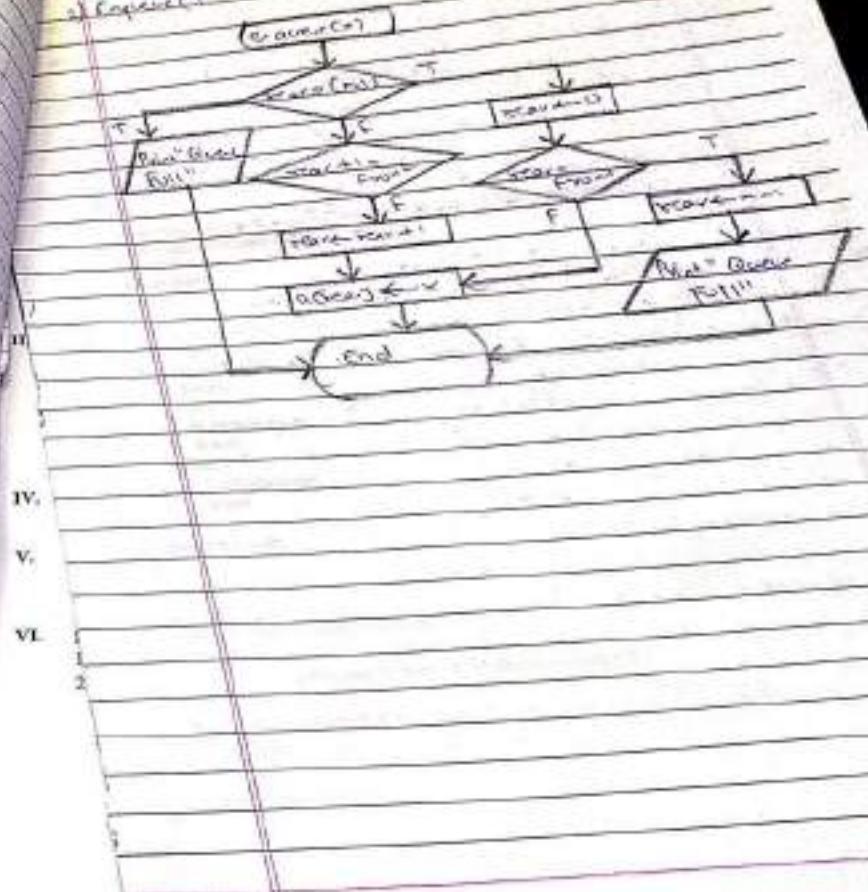
### 3) Delete

- 1) If Front == 1  
Write "Underflow"  
Get Stop
- 2) Set Nxt = Queue[Front]
- 3) If Front >= rear - 1  
exit
- 4) If Front == rear - 1  
Set Front = 0
- 5) If
- 6) Exit

### 4) Insert



### 2) Enqueue



IV.

V.

VI.

1

2

## Code C

```
#include <stdio.h>
#define size 5

void insertq(int[], int);
void deleteq(int[]);
void display(int[]);

int front = -1;
int rear = -1;

int main()
{
    int n, ch;
    int queue[size];
    do
    {
        printf("\n\n Circular Queue:\n1. Insert\n2. Delete\n3. Display\n0. Exit");
        printf("\nEnter Choice 0-3? ");
        scanf("%d", &ch);
        switch (ch)
        {
            case 1:
                printf("\nEnter number: ");
                scanf("%d", &n);
                insertq(queue, n);
                break;
            case 2:
                deleteq(queue);
                break;
            case 3:
                display(queue);
                break;
        }
    }while (ch != 0);
}

void insertq(int queue[], int item)
{
    if ((front == 0 && rear == size - 1) || (front == rear + 1))
    {
        printf("queue is full");
        return;
    }
    else if (rear == -1)
    {
        rear++;
    }
}
```

```

        }
        return 0;
    }

    void display(queue)
    {
        int i;
        printf("Elements in Queue are : ");
        if (front > rear)
        {
            for (i = front; i < size; i++)
                printf("%d ", queue[i]);
        }
        else if (front == rear)
        {
            printf("%d ", queue[0]);
        }
        else
        {
            for (i = front; i <= rear; i++)
                printf("%d ", queue[i]);
        }
    }

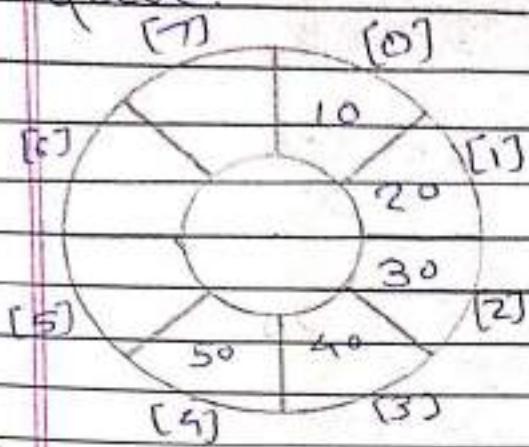
    void delete(queue)
    {
        if (front == -1)
        {
            printf("Queue is empty");
        }
        else if (front == rear)
        {
            printf("A %d deleted", queue[front]);
            front = -1;
            rear = -1;
        }
        else
        {
            if (rear == size - 1 && front > 0)
            {
                rear = 0;
            }
            else
            {
                rear++;
                queue[rear] = 0;
            }
            printf("A %d deleted", queue[front]);
            front++;
        }
    }
}

```



## \* Practical Related Questions,

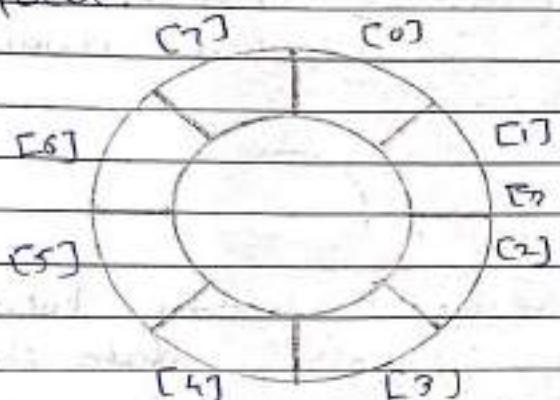
- 1) Write a front and rear values of circular queue.



→ Front = 0

rear = 4

- 2) Show the front and rear in empty circular queue.



→ front = -1

rear = -1

Q3. Show the Circular Queue  
Value of array size 5  
→ 15, 20, 30, 10, 5

Q4.



Q5.

Implement a C program Insert and  
Delete operations in Circular Queue  
using Array.

Q6.

To linear queue deleted item location  
is wasted because another data item  
cannot occupy that location. So to  
overcome this Greater time can be the  
feasible option.

Q7. Related Question  
T-1

- IV. Ref.  
A. 1  
V. Print  
A. M  
B. N  
VI. Delete  
1. Full  
2. Empty



9-2

III.

Re  
rac  
W  
or  
za  
llt  
o

"C:\Users\Orange\Desktop\p\manual dsu\b1n\Debug\manual dsu.exe"

```
Inserted -> 10
Inserted -> 40
Inserted -> 20
Deleted element -> 10
Inserted -> 40
Process returned 0 (0x0)
Press any key to continue. execution time : 0.616 s
```

- 3

```
#include <stdio.h>
#define SIZE 10
int items[SIZE];
int front = -1, rear = -1;

int isFull() {
    if ((front == rear + 1) || (front == 0 && rear == SIZE - 1)) return 1;
    return 0;
}

int isEmpty() {
    if (front == -1) return 1;
    return 0;
}

void enqueue(int element) {
    if (isFull())
        printf("\n Queue is full!! \n");
    else {
        if (front == -1) front = 0;
        rear = (rear + 1) % SIZE;
        items[rear] = element;
        printf("\n Inserted -> %d", element);
    }
}

int dequeue() {
    int element;
    if (isEmpty()) {
        printf("\n Queue is empty !! \n");
        return (-1);
    } else {
        element = items[front];
        if (front == rear) {
            front = -1;
            rear = -1;
        } else {
            front = (front + 1) % SIZE;
        }
        printf("\n Deleted element -> %d \n", element);
        return (element);
    }
}

void display() {
    int i;
```

```
if (isEmpty())
    printf("\n Empty Queue\n");
else {
    printf("\n Front -> %d ", front);
    printf("\n Items -> ");
    for (i = front; i != rear; i = (i + 1) % SIZE) {
        printf("%d ", items[i]);
    }
    printf("\n Rear -> %d \n", rear);
}
}

int main() {
    enQueue(23);
    enQueue(12);
    enQueue(45);
    enQueue(23);
    enQueue(47);
    enQueue(50);
    return 0;
}
```

## 10. Perform Operations of

Singly linked list.

Flawless Resource used

Code:	1
Page No.:	

S. No.	Name of Resource	Specifications
1	Computer System	Dual-core, 8gb Ram
2	Software	Turbo C
3	Any other	-

\* Algorithm

i] Insertion at head

- 1) addFirst (string newData);
- 2) Create a new node  $v$  containing newData
- 3)  $v.setNext (head)$
- 4)  $head = v$
- 5) size = size + 1

ii] Insertion at last

- 1) addLast (string newData);
- 2) Create a new node  $v$  containing newData
- 3)  $v.setNext (null)$
- 4) if  $(head == null)$   
 $head = v$

~~Code~~

else  
    tail->next = v  
    tail = v

    1) tail  
    2) size = size + 1

iii) Deletion

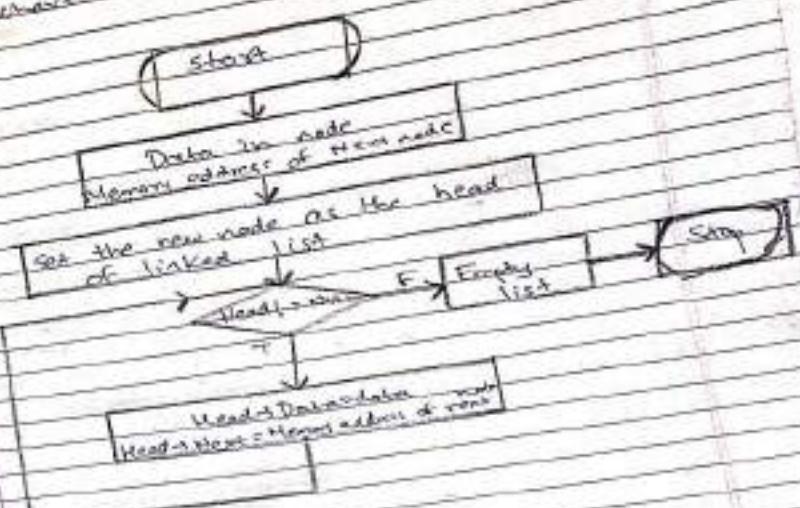
    1) traverse first  
    2) if head == null  
        Error: list is empty

    3) if head  
    4) head->next = head->next->next  
    5) if head->next == null  
    6) size = size - 1

#### iv) Traversing

- 1) Traverse list()
- 2) current = head
- 3) while (current != null)  
    current = current->next + 1

#### Elements



## Code..

```
#include<iostream>
#include<cstdio>
#include<cstdlib>
using namespace std;
struct node
{
    int info;
    struct node *next;
}*start;
class single_llist
{
public:
    node* create_node(int);
    void insert_begin();
    void delete_pos();
    void search();
    void display();
    single_llist()
    {
        start = NULL;
    }
};
int main()
{
    int choice, nodes, element, position, i;
    single_llist sl;
    start = NULL;
    while (1)
    {
        cout<<"1.Insert Node at beginning"<<endl;
        cout<<"2.Delete a Particular Node"<<endl;
        cout<<"3.Search Element"<<endl;
        cout<<"4.Display Linked List"<<endl;
        cout<<"5.Exit "<<endl;
        cout<<"Enter your choice : ";
        cin>>choice;
        switch(choice)
        {
            case 1:
                cout<<"Inserting Node at Beginning: "<<endl;
                sl.insert_begin();
                cout<<endl;
                break;
            case 2:
                cout<<"Delete a particular node: "<<endl;
                sl.delete_pos();
                break;
        }
    }
}
```

```

case 3:
    cout<<"Search element in Link List: "<<endl;
    sl.search();
    cout<<endl;
    break;
case 4:
    cout<<"Display elements of link list"<<endl;
    sl.display();
    cout<<endl;
    break;
case 5:
    cout<<"Exiting..."<<endl;
    exit(1);
    break;
default:
    cout<<"Wrong choice"<<endl;
}

node *single_llist::create_node(int value)
{
    struct node *temp, *s;
    temp = new(struct node);
    if (temp == NULL)
    {
        cout<<"Memory not allocated "<<endl;
        return 0;
    }
    else
    {
        temp->info = value;
        temp->next = NULL;
        return temp;
    }
}

void single_llist::insert_begin()
{
    int value;
    cout<<"Enter the value to be inserted: ";
    cin>>value;
    struct node *temp, *p;
    temp = create_node(value);
    if (start == NULL)
    {
        start = temp;
        start->next = NULL;
    }
    else
    {
        temp->next = start;
        start = temp;
    }
}

```

```

else
{
    p = start;
    start = temp;
    start->next = p;
    cout<<"Element Inserted at beginning"<<endl;
}

void single_llist::delete_pos()
{
    int pos, l, counter = 0;
    if (start == NULL)
    {
        cout<<"List is empty"<<endl;
        return;
    }
    cout<<"Enter the position of value to be deleted: ";
    cin>>pos;
    struct node **s, *p;
    s = &start;
    l = pos - 1;
    if (l >= 0)
    {
        start = s->next;
        s = start;
        while (s != NULL)
        {
            s = s->next;
            counter++;
        }
        if (pos > 0 && pos <= counter)
        {
            s = start;
            for (l = 1; l < pos; l++)
            {
                p = s;
                s = s->next;
            }
            p->next = s->next;
        }
        else
        {
            cout<<"Position out of range"<<endl;
            free(s);
            cout<<"Element Deleted!"<<endl;
        }
    }
}

```

1. **Search**  
 Search operation on a linked list  
 -> **Search**(  
 <-> Node \*ptr, int value)  
 {  
 if (ptr == NULL) return NULL;  
 if (ptr->data == value) return ptr;  
 else return Search(ptr->next, value);  
 }  
  
 2. **Insert**  
 Insert operation on a linked list  
 -> **Insert**(  
 <-> Node \*ptr, int value)  
 {  
 if (ptr == NULL) {  
 Node \*temp = new Node;  
 temp->data = value;  
 temp->next = NULL;  
 return temp;  
 }  
 else {  
 Node \*temp = new Node;  
 temp->data = value;  
 temp->next = ptr->next;  
 ptr->next = temp;  
 return ptr;  
 }  
 }  
  
 3. **Delete**  
 Delete operation on a linked list  
 -> **Delete**(  
 <-> Node \*ptr, int value)  
 {  
 if (ptr == NULL) return ptr;  
 if (ptr->data == value) {  
 Node \*temp = ptr->next;  
 free(ptr);  
 return temp;  
 }  
 else {  
 Node \*temp = ptr->next;  
 ptr->next = Delete(temp, value);  
 return ptr;  
 }  
 }  
  
 4. **Display**  
 Display operation on a linked list  
 -> **Display**(  
 <-> Node \*ptr)  
 {  
 if (ptr == NULL) cout << "List is empty";  
 else {  
 cout << "List is : ";  
 while (ptr != NULL) {  
 cout << ptr->data << " ";  
 ptr = ptr->next;  
 }  
 }

1. **Search**  
 Write a program to perform the operations (Search, Insert, Insert at Start) on singly linked list.  
  
**1. Program Construction**  
 In order to use list and perform operations on dynamic memory allocated application non-allocated static function is required. So linked list is used.  
 i.e. Cognizant data is non-allocated function  
 logical related function  
  
**2. Single Linked List**  
 pointers  
 a) zero  
 b) one  
 c) two  
 d) three  
  
**IV.** R  
 \* → true  
  
**V.** P  
 \* →  
 1. No. of pointers to be manipulated is a linked list to insert as item in the middle  
 2. If two  
 3. If three  
 → all three  
  
**VI.** Rules  
 1. For  
 2. For  
  
 1. One  
 2. All three  
  
 Maharashtra State Board  
 78

- Q) Linked list are not suitable for  
 structure of which one of the  
 problems?  
 a) Insertion Sort  
 b) Binary Search  
 c) Radix Sort  
 d) Polynomial Manipulation problem

- Q) The linked list fields are  
 a) Data by pointer  
 b) Pointer next node  
 Please note to node

a) Data is public  
 b) Next node

### Exercise

- 1) Give syntax of creation of a node.  
 typedef struct linkedlist \*node;  
 node createNode();  
 node temp;  
 temp = (node)malloc(sizeof(struct linkedlist));  
 temp->next = NULL;  
 return temp;
- 2) Represent linked list as stack and  
 queue.
- 3) Give output of  
 void printLL()
- IV.  
 1. print node & pcr-head();
   
 print ("lnl");
   
 II. Stack from begin
   
 while (ptr != null);
 print ("Cll");
 print ("Cll, null");
 pcr = pcr->next;
 print ("2");
- V.  
 1. Print
   
 2. Print
- VI.  
 Node  
 1. P1  
 2. P2
- print ("Cll, null");
 pcr = pcr->next;
 print ("2");

## Exercise - 2 - 1

```
linked list as stack
#include <iostream>
using namespace std;
struct Node
{
    int data;
    struct Node* link;
};

struct Node* top;
void push(int data)
{
    struct Node* temp;
    temp = new Node();
    if (!temp)
    {
        cout << "\nHeap Overflow";
        exit(1);
    }
    temp->data = data;
    temp->link = top;
    top = temp;
}
int isEmpty()
{
    return top == NULL;
}
int peek()
{
    if (isEmpty())
        return top->data;
    else
        exit(1);
}
void pop()
{
    struct Node* temp;
    if (top == NULL)
    {
        cout << "\nStack Underflow" << endl;
        exit(1);
    }
    else
    {
        temp = top;
        top = top->link;
        temp->link = NULL;
    }
}
```

## Exercise 2-2

```
struct List *allocList()
{
    struct List *list;
    list = (struct List *)malloc(sizeof(struct List));
    list->front = list->rear = NULL;
    return list;
}

struct Queue *createQueue()
{
    struct Queue *q = (struct Queue *)malloc(sizeof(struct Queue));
    q->front = q->rear = NULL;
    return q;
}

void enQueue(struct Queue *q, int x)
{
    struct Queue *temp = q->rear;
    if(q->rear == NULL)
        q->front = q->rear = temp;
    else
        q->rear->next = &x;
    q->rear = temp;
}

void deQueue(struct Queue *q)
{
    if(q->front == NULL)
        return;
    struct Queue *temp = q->front;
    q->front = q->front->next;
    if(q->front == NULL)
        q->rear = NULL;
    free(temp);
}

int main()
{
    struct Queue *q = createQueue();
}
```

Mathematics (State Board of)

```
    enQueue(q, 10);
    enQueue(q, 20);
    deQueue(q);
    deQueue(q);
    enQueue(q, 30);
    enQueue(q, 40);
    printf("Queue Front : %d \n", q->front->key);
    printf("Queue Rear : %d", q->rear->key);
    return 0;
}
```

# Perform Operations

On  
~~On~~

Date: / /  
Page No.:

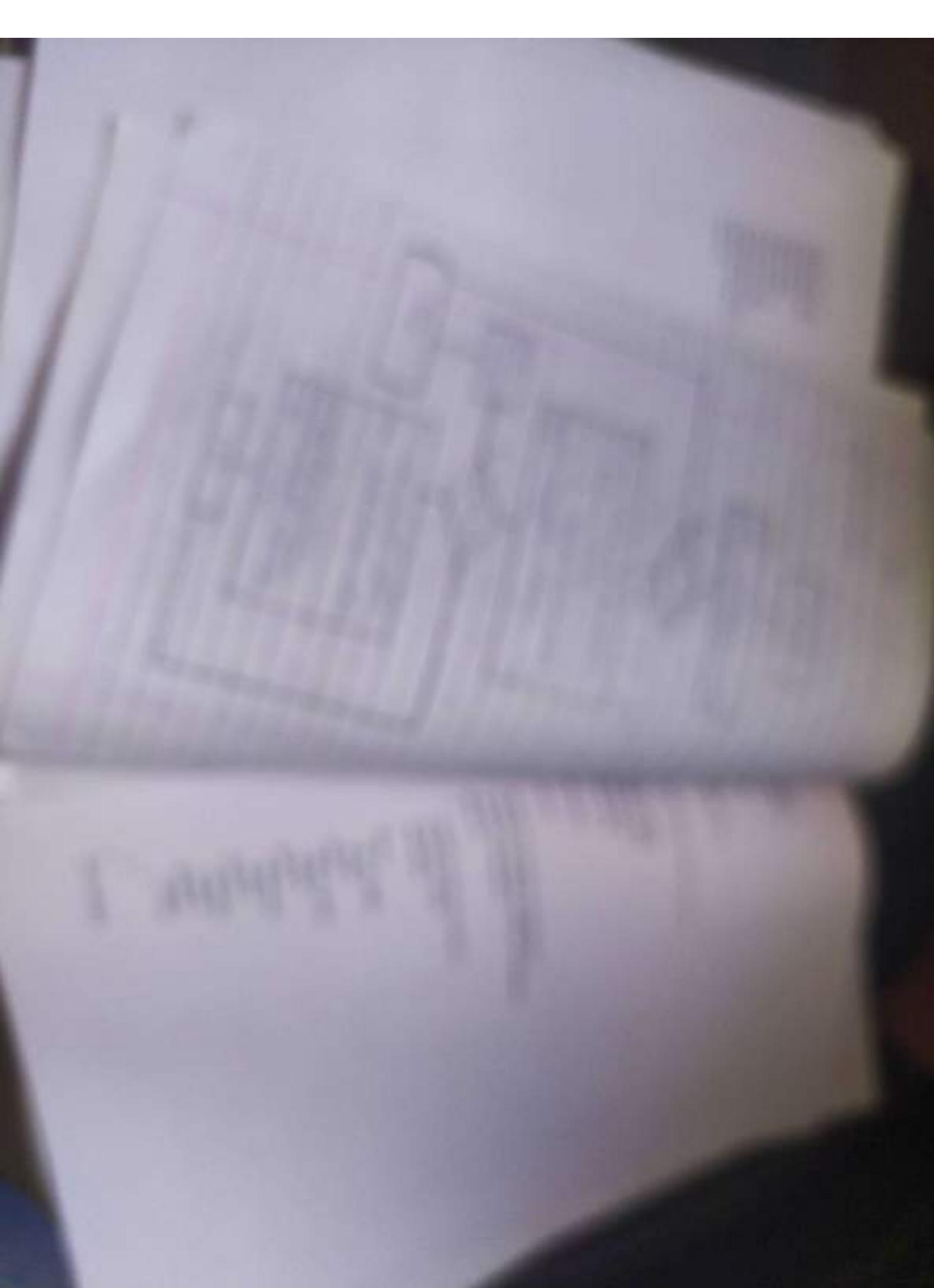
Circular singly linked list

Resources Used:

Sr. No.	Name of Resource	Specifications
1	Computer System	Pixel core, 6GB RAM
2	Software	Turbo C
3	Any other Resources	-

## \* Algorithm

- 1] IF  $\text{ptr} = \text{null}$   
    write overflow  
    Goto Step 11  
    [End of IF]
- 2] Set new\_node =  $\text{ptr}$
- 3] Set  $\text{ptr} = \text{ptr} \rightarrow \text{Next}$
- 4] Set  $\text{new\_node} \rightarrow \text{Data} = \text{Val}$
- 5] Set  $\text{temp} = \text{head}$
- 6] Repeat Step 8 while  $\text{temp} \rightarrow \text{Next} \neq \text{head}$
- 7] Set  $\text{temp} = \text{temp} \rightarrow \text{Next}$   
    [End of Loop]
- 8] Set  $\text{new\_node} \rightarrow \text{next} = \text{head}$
- 9] Set  $\text{temp} \rightarrow \text{next} = \text{new\_node}$
- 10] Set  $\text{head} = \text{new\_node}$
- 11] Exit



## Flashcard

Start

```
int i, num;  
Struct node *pre_pv, *node;
```



```
Struct node *start_node = malloc(
```

```
SizeOf(Struct node));
```

```
num = 1; start_node->data = num;
```

```
start_node->next_pv = NULL; start_node-
```

```
pre_pv = start_node;
```

```
[Eg]
```



```
Pre_pv  
next_pv  
start_node;
```

Stop

```
new_node = (Struct node*) -  
malloc(SizeOf(Struct node));
```

```
num = 2; new_node->data = num;
```

```
new_node->next_pv = num;
```

```
new_node->pre_pv = NULL;
```

```
pre_pv = new_node->pre_pv;
```

```
(pre_pv->next_pv) = new_node;
```

```
i++;
```

```
#include <iostream.h>  
#include <stdlib.h>  
  
using namespace std;  
  
Struct node *head = NULL, *x, *y, *c;  
  
class node {  
public:  
    int data;  
    node *next_pv;  
    node *pre_pv;  
};  
  
void ins_at_begin();  
void ins_at_pos();  
void traversal();  
void search();  
  
int main()  
{  
    cout << "1.Insertion at beginning";  
    cout << "2.Deletion at position (traverse)";  
    cout << "3.Search";  
    cout << "4.Exit";  
    cout << endl;  
    cout << "Enter your choice";  
    cin >> ch;  
    switch(ch){  
        case 1:  
            ins_at_begin();  
            break;  
        case 2:  
            ins_at_pos();  
            break;  
        case 3:  
            traversal();  
            break;  
        case 4:  
            search();  
            break;  
        default:  
            exit(0);  
    }  
}
```

```

int c;
x = (struct node*)malloc(sizeof(struct node));
printf("\nEnter the data:");
scanf("%d", &x->data);
x->link = x;
head = x;
printf("\n If you wish to continue press 1 otherwise 0:");
scanf("%d", &c);
while (c != 0)
{
    y = (struct node*)malloc(sizeof(struct node));
    printf("\nEnter the data:");
    scanf("%d", &y->data);
    x->link = y;
    y->link = head;
    x = y;
    printf("\n If you wish to continue press 1 otherwise 0:");
    scanf("%d", &c);
}
void ins_at_beg()
{
    x = head;
    y = (struct node*)malloc(sizeof(struct node));
    printf("\nEnter the data:");
    scanf("%d", &y->data);
    while (x->link != head)
    {
        x = x->link;
    }
    x->link = y;
    y->link = head;
    head = y;
}
void del_at_pos()
{
    if (head == NULL)
        printf("\n List is empty");
    else
    {
        int c = 1, pos;
        printf("\nEnter the position to be deleted:");
        scanf("%d", &pos);
        x = head;
        while (c < pos)
    }
}

```

```

x->link;
c();
reverse();
if (head == NULL)
    printf("\n List is empty");
else
{
    x = head;
    while (x->link != head)
    {
        printf("%d", x->data);
        x = x->link;
    }
    printf("%d", x->data);
}
void search()
{
    int search_val, count = 0, flag = 0;
    printf("\nEnter the element to search:");
    scanf("%d", &search_val);
    if (head == NULL)
        printf("\n List is empty nothing to search");
    else
    {
        x = head;
        while (x->link != head)
        {
            if (x->data == search_val)
            {
                printf("\nthe element is found at %d", count);
                flag = 1;
                break;
            }
            count++;
            x = x->link;
        }
        if (x->data == search_val)
    }
}

```

```
    printf("Element found at position %d", count);
}
if (flag == 0)
{
    printf("Element not found");
}
```

### Result:

write a C program to perform the operations (Insert, Delete, Traverse, Search) on Circular Singly linked list

### Conclusion:

In some applications Dynamic memory allocations using Singly linked list may pose to storage wastage if data items are deleted.

So Singly linked list can be used in Circular manner. It gives flexibility in storage management.

### Practical Related Questions

1. Circular Singly linked list count  
a) 10  
b) 100  
c) 1000  
d) None

2. No. of pointers to be manipulated in a Circular linked list to insert an item  
a) One  
b) Two  
c) Three  
d) One

Date: / /  
Page No.: /

linked list are not suitable for data structures of which one of the following problems?

- a) Insertion sort
- b) Binary search
- c) Radix sort
- d) Polynoamial manipulation

1] The last node of Circular linked list fields having

- a) data
- b) Pointer
- c) Pointer to next node
- d) Pointer to first node.

### \* Exercise

1] Give the benefit of Circular singly linked list.

→ 1] In circular linked list, end node will point to first node whereas in singly linked list it won't point to first node.

2] Circular linked list can be used for implementation of Queue.

3] It saves time when we have to go to the first node from the last node.

It can be done in single step because there is no need to traverse the in between nodes.

27 Represent Linked List as Queue.

```
class queue {
public:
    struct node {
        int data;
        struct node* next;
    };
    struct node* f = NULL;
    struct node* r = NULL;
    enqueue(int d) {
        struct node* n;
        n = (struct node*)malloc(sizeof(struct node));
        n->data = d;
        n->next = NULL;
        if(r == NULL) {
            f = r = n;
        } else {
            r->next = n;
            r = n;
        }
    }
    void dequeue() {
        struct node* t;
        if(f == NULL) {
            cout << "Queue is Empty";
        } else {
            t = f;
            f = f->next;
            free(t);
        }
    }
    void print() {
        struct node* t;
        if(f == NULL) {
            cout << "Queue is Empty";
        }
```

```
        printf("\nQueue is Empty");
    else{
        do{
            printf("\n%d",t->data);
            t = t->next;
        }while(t != f);
    }
}

int main()
{
    int opt,n,i,data;
    printf("Enter Your Choice:-");
    do{
        printf("\n\n1 for Insert the Data in Queue\n2 for show the Data in Queue\n3
for Delete the data from the Queue\n0 for Exit");
        scanf("%d",&opt);
        switch(opt){
            case 1:
                printf("\nEnter the number of data");
                scanf("%d",&n);
                printf("\nEnter your data");
                i=0;
                while(i<n){
                    scanf("%d",&data);
                    enqueue(data);
                    i++;
                }
                break;
            case 2:
                print();
                break;
            case 3:
                dequeue();
                break;
            case 0:
                break;
            default:
                printf("\nincorrect Choice");
        }
    }while(opt!=0);
    return 0;
}
```

~~2. Perform traversing on binary Search tree.~~

\* Resources Used :

Sr. No.	Name of Resource	Specifications
1	Computer System	Dual-core, 8gb RAM
2	Software	Turbo C
3	Any other resources	—

\* Algorithm : Visit elements in a binary search tree in reverse

1) Create B.S.T. (using insert & delete)

```

struct node* search (int data)
{
    struct node *current = root;
    printf ("Visiting element %d\n", current->data);
    while (current->data != data)
    {
        if (current != NULL)
            printf ("%d", current->data);
        if (current->data > data)
            current = current->leftchild;
        else
            current = current->rightchild;
    }
}
```

16 CENNEDY & NELSON

3

SUMMARY

reduction option

1) Insurance

2) Insurance assessment

2) Recreational activities high volume  
3) Work book trade low volume

3) Recreational activities option 2

1) Work book media

2) Recreational activities high volume  
3) Recreational activities option 1

1) Recreational activities  
2) Recreational activities high volume  
3) Recreational activities option 1

1) Recreational activities high volume  
2) Recreational activities option 1  
3) Work book media high volume

Costs

low rate estimate

high rate estimate

fixed rate

variable rate

total rate = fixed + variable

2) variable rate

3) variable rate

4) variable rate

5) variable rate

6) variable rate

7) variable rate

8) variable rate

9) variable rate

10) variable rate

11) variable rate

12) variable rate

13) variable rate

14) variable rate

15) variable rate

16) variable rate

17) variable rate

18) variable rate

19) variable rate

20) variable rate

21) variable rate

22) variable rate

23) variable rate

if (current == NULL) {

return NULL;

3

3

3

return current; // 07 (Return)

3

## 2) Traversal

### i) In-order traversal

- 1] Recursively traverse left subtree
- 2] Visit root node.
- 3] Recursively traverse right subtree

### ii) Preorder traversal

- 1] Visit root node.
- 2] Recursively traverse left subtree
- 3] Recursively traverse right subtree

### iii) Postorder traversal

- 1] Recursively traverse left subtree
- 2] Recursively traverse right subtree
- 3] Visit root node



## Code

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

```
#include < stdlib.h >
```

```
struct node {
```

```
    int Key;
```

```
    struct node * left, * right;
```

```
};
```

```
struct node * new(int item)
```

```
{
```

```
    struct node * temp
```

```
    = (struct node *) malloc(sizeof(struct node));
```

```
    temp -> Key = item;
```

```
    temp -> left = temp -> right = NULL;
```

```
    return temp;
```

```
}
```

```
void inorder(struct node * root)
```

```
{
```

```
    if (root != NULL) {
```

```
        inorder(root -> left);
```

```
        printf("%d\n", root -> Key);
```

```
        inorder(root -> right);
```

```
}
```

```
}
```

```
struct node * insert(struct node * node, int key)
```

```
{
```

```
    if (node == NULL)
```

```
        return new(key);
```

```
    if (key < node -> Key)
```

```
        node -> left = insert(node -> left, key);
```

```
        elseif (Key > node->Key)
            node->right = Insert (node->right, Key);
        return node;
    }
```

```
int main()
{
```

```
    struct node *root = NULL;
    root = insert (root, 50);
    insert (root, 30);
    insert (root, 70);
    insert (root, 40);
    insert (root, 60);
    insert (root, 80);
    inorder (root);
    return 0;
}
```

Result :

wrote a C program to implement BST and traverse the tree (inorder, preorder, postorder)

\* Conclusion

To perform operations in some applications which work on hierarchical data a Tree data structure is most known data structure. Tree is used to represent data in hierarchical manner. A BST can be used to represent ordered array and a linked list.

\* Practical Related Questions.

1) Every binary tree with  $n$  element have no. of edges

- a)  $n$
- b)  $n-1$
- c)  $n-2$
- d)  $n+1$

2) Define inorder traversal.

- a) left, root, right
- b) root, left, right
- c) left, root, right

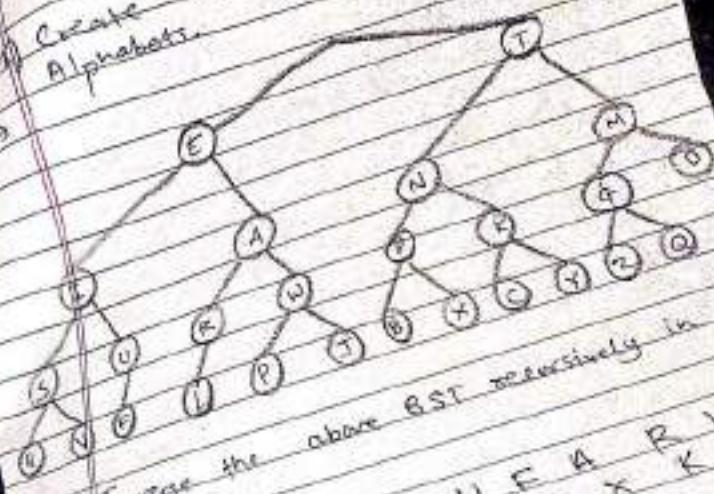
c) left, right, root

- d) none

Date: / /  
Page No.:

- 3) Define postorder traversal  
of tree  
a) left, right  
→ b) root, left, right  
c) right, left, root  
4) Define converse postorder traversal  
of tree  
a) left, right, root  
b) root, left, right  
c) right, left, root

Exercise  
Create binary search tree of Capital  
Alphabets.



→ Traverse the above BST recursively in  
Breadth.

→ E T S H V U F A R L  
N P J I N D B X K C  
T M Q Z O D