PAGE No.

DATE / /

Arya Sonawane 33

Int B-Tech SYCSE (AIDS)

12622402 73

Friday

PAGE NO:
DATE 11/7/25

# Experiment 1

WAP to implement array operations -

a) Find average of 10 numbers using array

→

```
# include <stdio.h>
int main()
{
    int n[10];
    int s=0;
    float a;
    printf ("Enter 10 number :\n");
    for ( int i=0 ; i<10 ; i++)
    {
        printf ( " Number %d : ", i+1);
        scanf ("%d ", &n[i]);
        s += n[i];
    }
    a = s/10.0;
    printf (" The average of the 10
     numbers is : %.2f \n ", a);
    return 0;
}
```

Output :

```
Enter 10 numbers :
 2
 5
 7
 4
```

8
3
2
7
7
The average of 10 numbers is : 5.30

b) Display the following pattern -

```
*
# #
* * *
# # #
* * * *
```

```
# include <stdio.h>
int main()
{
    for ( int i = 1 ; i <= 4 ; i++)
    {
        char c = (i % 2 == 1) ? '*' : '#';
        for ( int j = 1 ; j <= i ; j++)
            printf ("%c", c);
        printf ("\n");
    }
    return 0;
}
```

Output -

```
*
# #
* * *
# # # #
```

c) Find first repeating number in an array

```
# include <stdio.h>
int main()
{
    int n;
    printf (" Enter number");
    scanf (" %d", &n);
    int a[n];
    printf (" Enter %d elements : \n", n);
    for ( int i = 0 ; i < n ; i++)
        scanf ( "%d ", &a[i]);
    int f = -1;
    for ( int i = 0 ; i < n ; i++)
    {
        for ( int j = i ; j < n ; j++)
        {
            if ( a[i] == a[j])
            {
                f = a[i];
                break;
            }
        }
        if ( f != -1)
            break;
    }
    if ( f != -1)
        printf (" First repeating no is
        %d \n ", f);
```

```
    else
        printf (" No repeating no found .\n");
    return 0;
}
```

Output -

```
Enter the number : 4
Enter 4 elements :
1
2
3
4
No repeating number found.
```

d) Find greatest and smallest element in array.

```
# include <stdio.h>
int main()
{
    int n;
    printf (" Enter number ");
    scanf ("%d ", &n);
    int a[n];
    printf ("Enter %d elements : \n ", n);
    for (int i =0 ; i<n ; i++)
    {
        scanf ( "%d ", & a[i]);
    }
    int g = a[0];
    int s = a[0];
    for ( int i = 1; i<n ; i++)
    {
        if ( a[i] > g)
            g = a[i];
        if ( a[i] < s)
            s = a[i];
    }
    printf (" Greatest : %d \n", g);
    printf (" Smallest : %d \n", s);
    return 0;
}
```

Output :

```
Enter number : 5
Enter 5 elements
1
2
3
4
5
Greatest element : 5
Smallest element : 1
```

c) Square odd element numbers of array

```
# Include <stdio.h>
int main()
{
    int n;
    printf (" Enter number");
    scanf (" %d ", &n);
    int a[n];
    printf (" Enter %d elements : \n ", n);
    for (int i = 0 ; i<n ; i++)
        scanf ( "%d",&a[i]);

    for (int i=0 ; i<n ; i++)
    {
        if ( a[i] % 2 == 0)
        {
            a[i] = a[i] * a[i];
        }
    }
    printf (" After squaring : ");
    for ( int i=0 ; i<n ; i++)
        printf ("%d ", a[i]);

    printf ( " \n ");
    return 0 ;
}
```

Output :

```
Enter number elements : 5
Enter 5 elements :
2
4
7
9
6
Array After squaring :
2 4 49 81 6
```

# Experiment 2

a) Search data using linear search consider the quantities to perform using 56, 36, 89, 57, 1, 0, 67, 59. Search no 1 then 55.

```
# include <stdio.h>
int main()
{
    int n[] = {56,36,89,57,1,0,67,59};
    int i,x,f;
    printf(" Array : ");
    for ( i=0 ; i<7 ; i++)
        printf ("%d ", n[i]);
    printf ( " Enter number to search ");
    scanf ( "%d", &x);
    f = 0;
    for ( i=0 ; i<7; i++)
    {
        if ( x == n[i])
        {
            printf ( " Number found at index
            %d ", i);
            f = 1;
            break;
        }
    }
    if ( f == 0)
        printf (" Number not found");
    return 0;
}
```

Output

```
Array : 56 36 89 57 1 0 67 59
Enter number to search 1
Number found at index 4
Enter number to search 55
Number not found
```

b) Search data using binary search.

```
include <stdio.h>
int BinarySearch (int a[], int s, int t)
{
    int l = 0, ri = s-1;
    while ( l <= ri)
    {
        int m = l + (ri-l)/2;
        if ( a[m] == t)
            return m;
        else if ( a[m] < t)
            l = m+1;
        else
            ri = m-1;
    }
```

```
int main()
{
    int s, t;
    printf("Enter no of elements (sorted array): ");
    scanf("%d", &s);
    int a[s];
    printf("Enter %d sorted elements:", s);
    for (int i=0; i<s; i++)
        scanf("%d", &a[i]);
    printf("Enter target element: ");
    scanf("%d", &t);
    int r = binarySearch(a, s, t);
    if (r != -1)
        printf("Element %d found at index %d\n", t, r);
    else
        printf("Element not found");
    return 0;
}
```

Output -

```
Enter the number of elements (sorted array): 5
Enter 5 sorted array:
2
3
5
7
9
Enter the target element to search: 5
Element 5 found at index 2
```

3) Difference between linear and binary search.

| | Linear Search | Binary Search |
|---|---|---|
| 1) | Checks all elements of array. | Divides array and seacheres in halves |
| 2) | Works on both sorted and unsorted array | Works only on sorted array. |
| 3) | Sequential scan | Logarithmic time |
| 4) | Simple and straigh forward | Slightly more complex. |
| 5) | Slower on large database | Much faster on larger database |

4) Limitations of Linear search in terms of Time Complexity.

→ i) Inefficient for Larger Data Set :
Linear Search consumes more time as it checks each and every element as

The array may be big and it also depends on array data size of data types. Hence it is not suitable for longer arrays.

ii) No early stopping -
If the element is at the end of the array or not present in the array then the scan will not end early and use maximum time.

iii) No advantage for sorted data -
Even if the data is sorted, the linear search does not exploite this advantage.

iv) Scalability issues -
If the data contains millions or billions of data, linear search becomes infeasible due to the time being directly proportional to the elements.

1) WAP in c to copy the elements of one array into another array in reverse order

→
```
# include <stdio.h>
int main()
{
    int a[100], r[100], n;
    printf ("Enter the number of elements");
    scanf ("%d", &n);
    printf ("Enter %d elements : \n", n);
    for (int i=0; i<n; i++)
    {
        scanf ("%d", &a[i]);
    }
    for (int i=0; i<n; i++)
    {
        r[i] = a[n-1-i];
    }
    printf ("Reversed array : \n");
    for (int i=0; i<n; i++)
    {
        printf ("%d", r[i]);
    }
    return 0;
}
```

Output -
Enter the number of elements : 5
Enter 5 elements :

1
2
3
4
5

Reversed array :
5
4
3
2
1

Algorithm -

Step 1: Start
Step 2: Input variable a for array, r to store reversed array, n to input number of elements and i for for loop.
Step 3: Input no of elements.
Step 4: Use for loop to input elements.
Step 5: Use for loop to reverse the elements using r.
Step 6: Print the reversed array.
Step 7: Stop.

2> WAP to count total number of duplicate elements in array.

→

```
# include <stdio.h>
int main()
{
    int a[50], n, c = 0;
    printf(" Enter no of elements : ");
    scanf(" %d ", &n);
    printf(" Enter %d elements : \n ", n);
    for (int i = 0; i < n ; i++)
        scanf (" %d", &a[i]);

    for ( int i = 0; i < n ; i++)
    {
        for ( int j = i+1; j < n ; j++)
        {
            if (a[i] == a[j])
            {
                c++;
                break;
            }
        }
    }
    printf (" Total dupilcate elements : %d
    c);
    return 0;
}
```

Output -
Enter no of elements : 5
Enter 5 elements :

1
2
7
9
2

Total duplicate elements : 1

Algorithm -

Step 1: Start
Step 2: Declare a for array, n for no of elements in array, c for count of duplicate elements.
Step 3: Input the no of elements.
Step 4: Print the statement to ask user to input elements.
Step 5: Input elements for the array
Step 6: Use to for loops and use if condition to check is a[i] == a[j] if true c++;
Step 7: Print c (total no of duplicate elements)
Step 8: Stop

3) WAP in C to print all unique elements in an array. appear once in array

→
```
# include <stdio.h>
int main()
{
    int a[50], n, u;
    printf(" Enter no of elements : ");
    scanf("%d", &n);
    printf(" Enter %d elements :", n);
    for ( int i = 0 ; i<n ; i++)
        scanf("%d", &a[i]);

    printf(" Unique elements are:");
    for (int i =0 ; i<n ; i++)
    {
        for ( int j = 0 ; j<n ; j++)
        {
            if ( i!=j && a[i] == a[j])
            {
                u = 0;
                break;
            }
        }
        if (u)
            printf("%d", a[i]);
    }
    return 0;
}
```

Output -

Enter number of elements : 3
Enter 3 elements :
1
2
1
Unique elements are :
2

Algorithm -

Step 1: Start
Step 2: Declare a for array, n for no of elemen
u for unique, i and j for loop.
Step 3: Enter no of elements.
Step 4: Input elements using for loop.
Step 5: Check if they are unique
Step 6: Print array elements.
Step 7: Stop

4) WAP to seperate odd and even int
into seperate arrays

→
```
# include <stdio.h>
int main()
{
    int a[50], e[50], o[50];
    int n, ev = 0, od = 0;
    printf ("Enter no of elements :");
    scanf ("%d", &n);
    printf ("Enter %d elements : \n", n);
    for (int i = 0; i < n; i++)
        scanf ("%d", &a[i]);

    for (int i = 0; i < n; i++)
    {
        if (a[i] % 2 == 0)
            e[ev++] = a[i];
        else
            o[od++] = a[i];
    }
    printf ("Even elements : \n");
    for (int i = 0; i < ev; i++)
        printf ("%d", e[i]);
    printf ("Odd elements : \n");
    for (int i = 0; i < od; i++)
        printf ("%d", o[i]);
    return 0;
```

Output -

```
Enter no of elements : 5
Enter 5 elements -
1
2
3
4
5
Even elements :
2
4
Odd elements :
1
3
5
```

5) WAP to find second smallest element in given array.

→

```
# include <stdio.h>
int main()
{
    int a[50], n;
    inf  f = INT_MAX , s = INT_MAX
    printf (" Enter no of elements ");
    scanf ("%d", &n);
    printf (" Enter %d elements ", n);
    for (int i = 0 ; i<n ; i++)
    {       scanf("%d", &a[i]);
            if ( a[i] < f)
            {
                s = f
                f = a[i] ;
            }
            else if ( a[i] < s && a[i] != f)
            {
                s = a[i];
            }
    }
    if ( s == INT_MAX)
        printf (" No smallest element. ");
    else
        printf ( " Second smallest : %d \n", s);
    return 0;
}
```

Output -

```
Enter no of elements : 5
Enter 5 elements : 1
2
3
4
5
Second Smallest : 2
```

6) WAP to count the total no of words in a string.

→

```
# include <stdio.h>
# include <string.h>
int main()
{
    char s[100];
    int c=0, iw=0;
    printf("Enter a string:");
    fgets(s, sizeof(s), stdin);

    for (int i=0; s[i]!='\0'; i++)
    {
        if (isspace(s[i]))
        {
            iw=0;
        }
        else if (iw==0)
        {
            iw=1;
            c++;
        }
    }
    printf("Total no of words: %d\n", c);
    return 0;
}
```

Output -

Enter a string : Arya

25/7/25

## Experiment 3

1) Sort elements in ascending order using Bubble Sort -

→

```
# include <stdio.h>
int main()
{
    int a[] = {5,1,4,2,8};
    int n = sizeof(a)/sizeof(a[0]);
    int i,j,t;
    printf("Original array:");
    for (i=0; i<n; i++)
        printf("%d", a[i]);

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

```
    printf (" Sorted array:");
    for ( i=0 ; i<n ;i++)
        printf ("%d ", a[i]);

    return 0;
}
```

Output -

Original array :
5 1 4 2 8
Sorted array :
1 2 4 5 8

2) Sort elements in descending order using Selection Sort

→

```
# include <stdio.h>
int main()
{
    int a[] = { 5, 1, 4, 2, 8};
    int n = sizeof(a) / sizeof(a[0]);
    int i, j, m, t;

    printf(" Original array:");
    for ( i=0 ; i<n ; i++)
        printf ("%d", a[i]);

    for ( i=0 ; i<n-1 ; i++)
    {
        m = i;
        for ( j=i+1 ; j<n ; j++)
        {
            if ( a[j] > a[m])
            {
                m = j;
            }
        }
        t = a[i];
        a[i] = a[m];
        a[m] = t;
    }
    printf (" Sorted array in descending order ");
    for ( i=0 ; i<n ; i++)
        printf ("%d ", a[i]);
    return 0;
}
```

Output -

Original array :
5 1 4 2 8
Sorted array in descending order :
8 5 4 2 1

3) Find the no of comparisons required in bubble sort method of the following data having 5 numbers : 100, 200, 300, 400, 500

→ Given numbers - 100, 200, 300, 400, 500

| | | | | |
|---|---|---|---|---|
| 100 | 200 | 300 | 400 | 500 |
| 100 | 200 | 300 | 400 | 500 |
| 100 | 200 | 300 | 400 | 500 |
| 100 | 200 | 300 | 400 | 500 |

Hence only 1 pass and 4 comparisons take place as the given numbers are already sorted.

Algorithm -

1 - Start with the list of numbers
2 - Compare each pair of adjecent elements and swap if the first element is greater than second element. Continue comparing and sorting.
3 - After each full pass the largest unsorted element moves to the end of array.
4 - Repeat the process of 2 step on the $n$ unsorted array.
5 - Stop when all elements are in the correct order (after n-1 passes)

4) Sort the given array in selection sort in ascending order and show diagrammatic representation of every iteration of the loop : 500, -20, 30, 14, 50

→

| | | | | | |
|---|---|---|---|---|---|
| 500 | -20 | 30 | 14 | 50 | } 1st pa |
| -20 | 500 | 30 | 14 | 50 | } 2nd pa |
| -20 | 30 | 500 | 14 | 50 | |
| -20 | 14 | 500 | 30 | 50 | } 3rd pa |
| -20 | 14 | 30 | 500 | 50 | } 4th pa |
| -20 | 14 | 30 | 50 | 500 | |

Hence only 4 passes and 5 comparisions take place.

Algorithm -

1 - Set min to 0
2 - Search min element of the array
3 - Swap value at the location of min
4 - Increment min to point the next element location
5 - Repeat till sorted
6 - Stop.

28/7/25

# Experiment- 4

a) Sort elements in ascending order using Insertion Sort.

→

```
# include <stdio.h>
int main()
{
    int a[] = { 7, 3, 5, 1, 9, 8, 4, 6};
    int n = sizeof (a) / sizeof( a[0]);
    int i, j, k;

    for ( i=1 ; i<n ; i++)
    {
        k = a[i];
        j = i-1;
        while ( j>=0 && a[j] > k)
        {
            a[j+1] = a[j];
            j = j-1;
        }
        a[j+1] = k;
        if ( i==2)
        {
            printf(" Array :");
            for ( int i=0 ; i<n ; i++)
                printf ("%d ", a[i]);
            printf ("\n");
        }
    }
    printf(" Sorted Array :");
    for ( i=0 ; i<n ; i++)
        printf ("%d", a[i]);
    return 0;
}
```

Output -

```
Array :
3 5 7 1 9 8 4 6
Sorted array :
1 3 4 5 6 7 8 9
```

b) Sort element in ascending order using Radix Sort.

→

```
# include <stdio.h>
int getMax ( int a[], int n)
{
    int m = a[0]; // m = max
    for ( int i = 1 ; i<n ; i++)
    {
        if ( a[i] > m)
        m = a[i];
        return m;
    }
}
void sort ( int a[] ; int n , int exp)
{
```

```
int o[50], c[10] = {0};
// o = output , c = count

for ( int i = 0 ; i<n ; i++)
    c [ (a[i] / exp) % 10 ] ++;
for ( int i = 1 ; i<10 ; i++)
    c[i] += c[i-1];

for ( int i = n-1 ; i >= 0 ; i--)
{
    o [ c [ (a[i] / exp) % 10 ] - 1 ] =
    a [i];
    c [ (a[i] / exp) % 10 ] --;
}
for ( int i = 0 ; i<n ; i++)
    a[i] = o[i];
}
int main ()
{
    int a[] = { 7, 3, 5, 1, 9, 8, 4, 6};
    int n = sizeof (a) / sizeof ( a[0]);
    radix

void radixSort ( int a[], int n)
{
    int m = getMax ( a , n);
    for ( int exp = 1 ; m / exp > 0 ; exp *= 10)
        Sort ( a , n , exp);
}
```

```
int main()
{
    int a[] = { 7, 3, 5, 1, 9, 8, 4, 6};
    int n = sizeof (a) / sizeof (a[0]);

    radixSort ( a, n);
    printf (" Sorted Array : ");
    for ( int i=0 ; i<n ; i++)
        printf (" %d ", a[i]);
    return 0;
}
```

Output -

Sorted Array :

1 3 4 5 6 7 8 9

c) What is the output of insertion sort after the 2nd interation given the following sequence of numbers : 7, 3, 5, 1, 9, 8, 4, 6.

7 3 5 1 9 8 4 6

3 7 5 1 9 8 4 6

3 5 7 1 9 8 4 6

1 3 5 7 9 8 4 6

1 3 5 7 8 9 4 6

1 3 4 5 7 8 9 6

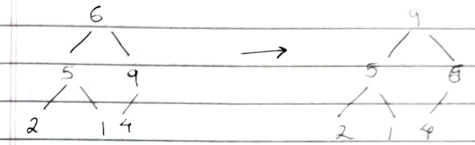
1 3 4 5 6 7 8 9

Algorithm -

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

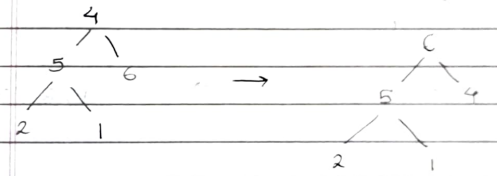
1) Sort the following elements using heap sort.

i) 6, 5, 9, 2, 1, 4



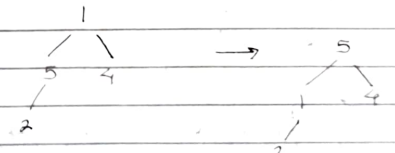

sorted array 9, 5, 6, 2, 1, 4

4, 5, 6, 2, 1, 9

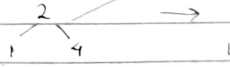



sorted array 6, 5, 4, 2, 1

1, 5, 4, 2, 6



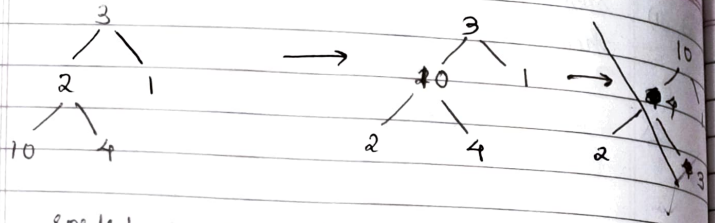

sorted array : 5, 1, 4, 2 → 2, 1, 4, 5, 6, 9



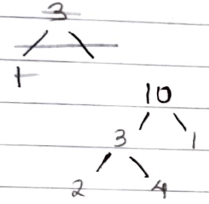

sorted array [4, 1, 2] → 2, 1, 4, 5, 6, 9

sorted array → 1, 2, 4, 5, 6, 9

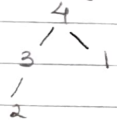
ii) 3, 2, 1, 10, 4



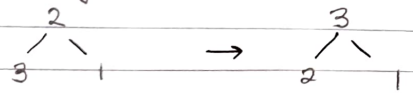

Sorted array : 10, 3, 1, 2, 4 → ~~3, 1, 2, 4, 10~~ Sorted array : 10, 4, 1, 2, 3




array - 10, 3, 1, 2, 4 → 4, 3, 1, 2, 10




array - 4, 3, 1, 2, 10 → 2, 3, 1, 4, 10




array - 3, 2, 1, 4, 10 → ~~2, 1, 4, 10, 3~~ 1, 2, 3, 4, 10

Sorted array : {1, 2, 3, 4, 10}

2) Sort the following using radix sort -

i) 100, 225, 390, 4130, 956, 99, 5431

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 100 | 100 | | | | | | | | | |
| 225 | | | | | | 225 | | | | |
| 390 | 390 | | | | | | | | | |
| 4130 | 4130 | | | | | | | | | |
| 956 | | | | | | | 956 | | | |
| 99 | | | | | | | | | | 99 |
| 5431 | | 5431 | | | | | | | | |

100, 390, 4130, 5431, 225, 956, 99

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 100 | 100 | | | | | | | | | |
| 390 | | | | | | | | | | 390 |
| 4130 | | | | 4130 | | | | | | |
| 5431 | | | | 5431 | | | | | | |
| 225 | | | 225 | | | | | | | |
| 956 | | | | | | 956 | | | | |
| 99 | | | | | | | | | | 99 |

100, 225, 4130, 5431, 956, 390, 99

DATE / /

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 100 | | 100 | | | | | | | | |
| 225 | | | 225 | | | | | | | |
| 4130 | | 4130 | | | | | | | | |
| 5431 | | | | | 5431 | | | | | |
| 956 | | | | | | | | | | 956 |
| 390 | | | | 390 | | | | | | |
| 99 | 99 | | | | | | | | | |

99, 100, 4130, 225, 390, 5431, 956,

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 99 | 99 | | | | | | | | | |
| 100 | 100 | | | | | | | | | |
| 4130 | | | | | 4130 | | | | | |
| 225 | 225 | | | | | | | | | |
| 390 | 390 | | | | | | | | | |
| 5431 | | | | | | 5431 | | | | |
| 956 | 956 | | | | | | | | | |

∴ 99, 100, 225, 390, 956, 4130, 5431

(ii) 25, 6, 99, 145, 239, 20, 18

PAGE NO.
DATE / /

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 25 | | | | | | 25 | | | | |
| 6 | | | | | | 6 | | | | |
| 99 | | | | | | | | | | 99 |
| 145 | | | | | | 145 | | | | |
| 239 | | | | | | | | | | 239 |
| 20 | 20 | | | | | | | | | |
| 18 | | | | | | | | | 18 | |

20, 25, 145, 6, 18, 99, 239

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 20 | | | 20 | | | | | | | |
| 25 | | | 25 | | | | | | | |
| 145 | | | | | 145 | | | | | |
| 6 | 6 | | | | | | | | | |
| 18 | | 18 | | | | | | | | |
| 99 | | | | | | | | | | 99 |
| 239 | | | | 239 | | | | | | |

6, 18, 20, 25, 239, 145, 99

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 6 | 6 | | | | | | | | | |
| 18 | 18 | | | | | | | | | |
| 20 | 20 | | | | | | | | | |
| 25 | 25 | | | | | | | | | |
| 239 | | | 239 | | | | | | | |
| 145 | | 145 | | | | | | | | |
| 99 | 99 | | | | | | | | | |

Sort using shell sort -

i) 22, 18, 29, 7, 9, 55, 21, 8

length of array = 8
array / 2 = 4

22 18 29 7 9 55 21 8

9 18 29 7 22 55 21 8

9 18 29 7 22 55 21 8

9 18 21 7 22 55 29 8

after gap -

9 18 21 7 22 55 29 8

Taking gap = 2

9 18 21 7 22 55 29 8

9 7 21 18 22 55 29 8

9 7 21 18 22 8 29 55

9 7 21 18 22 18 29 55

gap = 1,

9 7 21 8 22 18 29 55

7 9 21 8 22 18 29 55

7 9 8 21 22 18 29 55

7 8 9 21 22 18 29 55

7 8 9 21 18 22 29 55

7 8 9 18 21 22 29 55

∴ sorted array - 7, 8, 9, 18, 21, 22, 29, 55

ii) 3, 10, 15, 12, 1, 5, 2, 6
array / 2 = 4

3 10 15 12 1 5 2 6

1 10 15 12 3 5 2 6

1 5 15 12 3 10 2 6

1 5 15 6 3 10 15 12

after gap -
1, 5, 2, 6, 3, 10, 15, 12

gap = 2

1 5 2 6 3 10 15 12

gap = 3

1 5 2 6 3 10 15 12

1 2 5 6 3 10 15 12

1 2 5 3 6 10 15 12

1 2 3 5 6 10 15 12

1 2 3 5 6 10 12 15

∴ sorted array -

1, 2, 3, 5, 6, 10, 12, 15

Experiment 5

Singly linked list using menu

```
# include <stdio.h>
# include <stdlib.h>
struct node{
  int d;
  struct node *n;
};
struct node *h = NULL;
void ib ( int x )
{
    struct node *t = malloc ( sizeof (
    struct node));
    t -> d = x;
    t -> n = h;
    h = t;
}
void ie ( int x )
{
    struct node * t = malloc ( size of (
    struct node));
    t -> d = x;
    t -> n = NULL;
    if ( !h)
    {
        h = t;
    }
```

```
        struct node *p = h;
        while (p -> n) p = p -> n;
        p -> n = t;
    }
}
void del (int x)
{
    struct node *p = h, *q = NULL;
    while ( p && p -> d != x )
    {
        q = p;
        p = p -> n;
    }
    if ( !p) return;
    if ( !q)
        h = h -> n;
    else
        q -> n = p -> n;
    free (p);
}
void s (int x)
{
    struct node *p = h, *q = NULL;
    while ( p)
    {
        if ( p -> d == x )
        {
            printf ( " Found ");
        }
        p = p -> n;
    }
    printf (" Not Found ");
}
void dis()
{
    struct node *p = h;
    while (p)
    {
        printf ( "%d", p -> d);
        p = p -> n;
    }
    printf("\n");
}
int main()
{
    int c, x;
    for ( ;; )
    {
        printf ( " 1. IB 2. IE 3. Del 4.
        Search 5. Display 6. Exit /n ");
        scanf ("%d", &c);
        if ( c == 6)
        break;
        switch (c)
        {
            case 1
                scanf ( "%d", &x);
                ib (x);
```

```
        case 2:
            scanf("%d", &x);
            ie(x);
            break;
        case 3:
            scanf("%d", &x);
            del(x);
            break;
        case 4:
            scanf("%d", &x);
            s(x);
            break;
        case 5:
            dis();
            break;
    }
  }
}
```

Output -

```
1. IB  2. IE  3. DEL  4. SEARCH  5 DISPLAY  6 EXI
1
10
1. IB  2. IE  3. DEL  4. SEARCH  5 DISPLAY  6 EXIT
2
20
1. IB  2. IE  3. DEL  4. SEARCH  5. DISPLAY  6. EXI
5
20  10
1. IB  2. IE  3. DEL  4. SEARCH  5. DISPLAY  6. EXIT
```

PAGE NO.
DATE / /

## Experiment 6

Doubly linked list menu driven program.

→

```
# include <stdio.h>
# include <stdlib.h>
struct node
{
    int d;
    struct node *p, *n;
};
struct node *h = NULL;
void ib(int x)
{
    struct node *t = malloc(sizeof(struct
    node));
    t -> d = x;
    t -> p = NULL;
    t -> n = h;
    if(h) h -> p = t;
    h = t;
}
void ie(int x)
{
    struct node *t = malloc(sizeof(struct
    node)), *q = h;
    t -> d = x;
    t -> n = NULL;
    if(!h)
    {
```

```
        t -> p = NULL;
        h = t;
        return;
    }
    while ( q -> n)
    {
        q = q -> n;
        q -> n = t;
        t -> p = q;
    }
}
void del (int x)
{
    struct node *q = h;
    while ( q && q -> d != x)
        q = q -> n;
    if ( !q)
        return;
    if ( q -> p)
        q -> p -> n = q -> n;
    else
        h = q -> n;
    if ( q -> n)
        q -> n -> p = q -> p;
    free (q);
}
int s( int x)
{
    struct node *q = h;
```

```
    int pos = 1;
    while (q)
    {
        if ( q -> d == x)
            return pos;
        q = q -> n;
        pos ++;
    }
    return 0;
}
void dis()
{
    struct node *q = h;
    if ( !q)
    {
        printf ("Empty");
        return 0;
    }
    while (q)
    {
        printf ("%d", q -> d);
        q = q -> n;
    }
    printf ("\n");
}
int main()
{
    int ch, x, r;
    while (1)
    {
```

```
printf(" 1.IB 2.IE 3.DEL 4.SEA 5.DIS 6.EXIT");
scanf("%d", &ch);
switch (ch) {
    case 1:
        scanf("%d", &x);
        ib(x);
        break;
    case 2:
        scanf("%d", &x);
        ie(x);
        break;
    case 3:
        scanf("%d", &x);
        del(x);
        break;
    case 4:
        scanf("%d", &x);
        r = s(x);
        if (r)
            printf(" Found ");
        else
            printf(" Not Found ");
        break;
    case 5:
        dis();
        break;
    case 6:
```

~~Experiment 7~~

Output -

```
1. IB 2. IE 3. DEL 4. SEA 5. DIS 6. EXIT
1
5
1. IB 2. IE 3. DEL 4. SEA 5. DIS 6. EXIT
1
6
1. IB 2. IE 3. DEL 4. SEA 5. DIS 6. EXIT
2
7
1. IB 2. IE 3. DEL 4. SEA 5. DIS 6. EXIT
4
6
Found at 1
1. IB 2. IE 3. DEL 4. SEA 5. DIS 6. EXIT
5
6 5 7
1. IB 2. IE 3. DEL 4. SEA 5. DIS 6. EXIT
6
```

Code Execution Successful

30/9

# Experiment 7

Stack menu driven program for push, pop display and exit

→

```
#include <stdio.h>
#include <stdlib.h>
#define size 5
void push();
void pop();
void display();
int stack[size];
int top = -1;
int main()
{
    int ch;
    printf("1. PUSH \n 2. POP \n 3. Display
     \n 4. EXIT");
    do
    {
        printf("Enter choice:");
        scanf("%d", &ch);
        switch (ch)
        {
            case 1:
            push();
            break;
            case 2:
            pop();
            break;
            case 3:
            display();
            break;
            case 4:
            exit(0);
            break;
        }
    }
    void pop()
    {
        if (top == -1)
            printf("Stack is empty");
        else
        {
            printf("Deleted element is = %d",
            stack[top]);
            top--;
        }
    }
    void push()
    {
        int v;
        if (top == MAX - 1)
            printf("Overflow");
        else
        {
            printf("Enter element :");
            scanf("%d", &v);
            stack[++top] = v;
        }
    }
```

```
void display()
{
    if (top == -1)
        printf ("Empty");
    else
    {
        print("Elements are :");
        for (int i=0; i<=top; i++)
        {
            printf("%d", stack[i]);
        }
        printf(")n");
    }
}
```

Output -

```
1. PUSH  2. POP  3. DISPLAY  4. EXIT
Enter choice : 1
Enter element  1

Enter choice : 3
Elements are  :  1

Enter choice  4
```

## Experiment 8

i) Convert infix to postfix or prefix expression

→

```
# include <stdio.h>
# include <string.h>
# include <ctype.h>
# define m 100
char s[m][m];
int t = -1;
void p(char *x)
{
    strcpy(s[++t], x);
}
char * o()
{
    return s[t--];
}
int pr(char c)
{
    if (c=='+' || c=='-')
        return 1;
    if (c=='*' || c=='/')
        return 2;
    if (c=='^')
        return 3;
    return 0;
}
```

```
int op (char c)
{
    return (c=='+' || c=='-' || c=='*' ||
    c=='/' || c=='^');
}
void infixToPostfix (char *in, char *post)
{
    char st[M];
    int top=-1, i, k=0;
    for (i=0; in[i]; i++)
    {
        char c = in[i];
        if (isalnum(c)) post[k++]=c;
        else if (c=='(')
            st[++top]=c;
        else if (c==')')
        {
            while (top!=-1 && st[top]!=
            '(')
                post[k++] = st[top--];
            if (top!=-1)
                top--;
        }
        else if (op(c))
        {
            while (top!=-1 && pr(st[top])
            >= pr(c)) post[k++] = st[top--];
            st[++top]=c;
        }
    }
    while (top!=-1)
        post[k++] = st[top--];
    post[k]='\0';
}
void rev (char *x)
{
    int l=strlen(x), i;
    for (i=0; i<l/2; i++)
    {
        char temp = x[i];
        x[i] = x[l-i-1];
        x[l-i-1] = temp;
    }
}
void swapB (char *x)
{
    for (int i=0; x[i]; i++)
    {
        if (x[i]=='(') x[i]=')';
        else if (x[i]==')') x[i]='(';
    }
}
void infixToPrefix (char *in, char *pre)
{
    char x[M], temp[M];
    strcpy(x, in);
    rev(x);
    swapB(x);
    infixToPrefix (char *in, char *pre)
    {
```

```
    char r[m], temp[m];
    strcpy (r, in);
    rev(r);
    swapB(r);
    infixToPostfix (r, temp);
    rev (temp);
    strcpy (pre, temp);
}

int main()
{
    char in[m], po[m], prf[m];
    printf (" Enter infix : ");
    scanf (" %s", in);
    infixToPostfix (in, po);
    infixToPrefix (in, prf);
    printf (" Postfix : %s\n", po);
    printf (" Prefix : %s\n", prf);
    return 0;
}
```

Output -

Enter infix : A+B*(C/D)

Postfix : ABCD/*+

Prefix : +A*B/CD

ii) A+(B*C-(D/E^F)*G)*H

| Symbol | Stack | Output |
|---|---|---|
| H | | |
| * | * | H |
| ) | *) | H |
| G | *) | HG |
| * | *)* | HG |
| ) | *)*) | HG |
| E | *)*) | HGF |
| ^ | *)*)^ | HGF |
| E | *)*)/ | HGFE |
| / | *)*)/ | HGFE^ |
| D | *)*)/ | HGFE^D |
| ( | *)* | HGFE^D/ |
| - | *)- | HGFE^D/* |
| C | *)- | HGFE^D/*C |
| * | *)-* | HGFE^D/*C |
| B | *)-* | HGFE^D/*CB |
| ( | * | HGFE^D/*CB*- |
| + | + | HGFE^D/*CB*-* |
| A | + | HGFE^D/*CB*-*A |

Reversed,

HGFE^D/*CB*-*A+

+A*-*BC*/D^EFGH

iii) Infix to Postfix (Evaluate)

→

```
# include <stdio.h>
# include <ctype.h>
# define M 100
char st[M];
int top = -1;
void push (char c)
{
    st[++top] = c;
}
char pop()
{
    if (top == -1)
        return '\0';
    return st[top--];
}
int pr (char c)
{
    if (c == '+' || c == '-')
        return 1;
    if (c == '*' || c == '/')
        return 2;
    return 0;
}
int isOp (char c)
{
    return c == '+' || c == '-' || c == '*' ||
```

```
void infixToPostfix (char * infix, char *
postfix)
{
    int i = 0, k = 0;
    while (infix[i])
    {
        char c = infix[i];
        if (isdigit (c)) postfix[k++] = c;
        else if (c == '(')
            push (c);
        else if (c == ')')
        {
            while (top != -1 && st[top]
            != '(')
                postfix[k++] = pop();
            pop();
        }
        else if (isOp (c))
        {
            while (top != -1 && pr(st[top])
            >= pr(c))
            postfix[k++] = pop();
            push (c);
        }
        i++;
    }
    while (top != -1)
        postfix[k++] = pop();
    postfix[k] = '\0';
```

```
int evalPostfix(char * postfix)
{
    int stack[M], t = -1, i = 0;
    while (postfix[i])
    {
        char c = postfix[i];
        if (isdigit(c))
            stack[++t] = c - '0';
        else
        {
            int b = stack[t--];
            int a = stack[t--];
            if (c == '+')
                stack[++t] = a + b;
            else if (c == '-')
                stack[++t] = a - b;
            else if (c == '*')
                stack[++t] = a * b;
            else
                stack[++t] = a / b;
        }
        i++;
    }
    return stack[t];
}
int main()
{
    char infix[M], postfix[M];
    scanf("%s", infix);
    top = -1;
    infixToPostfix(infix, postfix);
    printf("Postfix : %s\n", postfix);
    int res = evalPostfix(postfix);
    printf("Result : %d\n", res);
    return 0;
}
```

Output--

```
Enter infix : 456*+
Result : 34
```

DATE / /

iv) Evaluate prefix -

+ - * + 12 / 4 2 1 $ 4 2

| I/P | OP1 | OP2 | R | S |
|---|---|---|---|---|
| 2 | | | | |
| 4 | | | | 2 |
| $ | 4 | $ 2 | 16 | 2,4 |
| 1 | | | | 16 |
| 2 | | | | 16,1 |
| 4 | | | | 16,1,2 |
| / | 4 | / 2 | 2 | 16,1,2,4 |
| 2 | | | | 16,1,2 |
| 1 | | | | 16,1,2,2 |
| + | 1 | + 2 | 3 | 16,1,2,1 |
| * | 3 | * 2 | 6 | 16,1,6 |
| - | 6 | - 1 | 5 | 16,5 |
| + | 16 | + 5 | 21 | 21 |

PAGE NO.
DATE / /

Algorithms -

1) Infix to Prefix -

Step 1 - Reverse the infix expression
Step 2 - Convert reversed infix to postfix
Step 3 - Reverse the Postfix expression
Step 4 - Write the prefix expression
Step 5 - Stop

2) Infix to Postfix -

Step 1 - Initialize the stack
Step 2 - Scan the expression from left to right.
Step 3 - Pop all remaining operators
Step 4 - Output is postfix expression
Step 5 - Stop

3) Evaluation of postfix -

Step 1 - Initalize from left to right
Step 2 - Scan symbols and operators
Step 3 - After scanning we get postfix expression.
Step 4 - print postfix expression
Step 5 - Stop.

4) Evaluation for prefix operator -

Step 1 - Initialize the operands and operators
Step 2 - Solve the operators from right to left
Step 3 - In the end, stack will have one value (result)
Step 4 - Print result
Step 5 - Stop.

22/9/25

Questions

1. Explain term Linked List

→ It is a linear data structure where elements (nodes) are connected using pointers or links instead of being stored in a continuous block of memory like arrays.
There are three types -
i) Singly Linked list
ii) Doubly Linked list
iii) Circular Linked list.

2. Diff. between linked list and arrays

| Array | Linked list |
|---|---|
| i) Collection of elements stored at contiguous memory location | Collection of nodes, having a pointer to the next node. |
| ii) Static memory location | Dynamic memory location |
| iii) Insertion or deletion of element is hard. | Insertion or deletion is easy |

| | | |
|---|---|---|
| iv) | Searching is only possible if array is sorted. | Searching is possible easily. |
| v) | 1D, 2D, multidimensional arrays. | Singly, Doubly, Circular linked list |

3. Singly vs Doubly linked list.

→

| | Singly | Doubly |
|---|---|---|
| i) | Each node has a pointer which points to the next node. | Each node has two pointers to point to next and previous nodes. |
| ii) | It requires less memory. | It requires more memory. |
| iii) | Can be traversed only in one direction | Can be traversed in both directions |
| iv) | Simple to implement | More complex due to two pointers. |

4) Operations in linked list

→ i) Traversal - It displays or visits each node. It starts from head of the node till the last node.

ii) Insertion - It adds a element in the nodes either at beginning or end or at a specific position.

iii) Deletion - It removes a node either from beginning, end or at a specific position.

iv) Searching - It searches for a value from head to end.

5. Types of linked list

→ i) Singly linked list -
Each node has data and a pointer to the next node. Traversing is only possible in one direction.

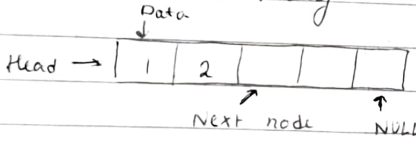
ii) Doubly linked list -
Each node has data and a pointer to the next node and a pointer to the previous node. Traversing is possible in both directions.

iii) Circular linked list - The last node points back to the first node (head) forming a circle.

6. Explain singly / doubly linked list with diagram.

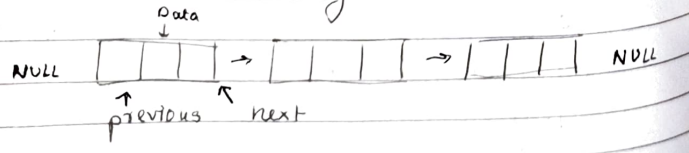
→ Singly linked list -

i) Node stores value or information
ii) It uses next to point to the next node in a list
iii) It has one pointer
iv) It traverses in only one direction.
v) It uses less memory.



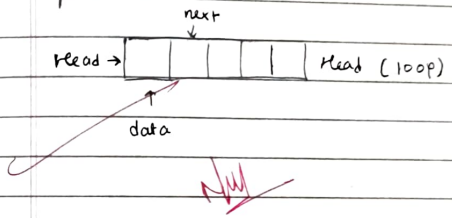

Doubly linked list -

i) Node stores value or information
ii) Next references the next node
iii) It can even point to the previous node
iv) Traverses in both direction
v) Has two pointers.
vi) Uses more memory.




7) Circular linked list -

→ It is a regular linked list where the last node in the list points back to the first node, forming a circle. It can be singly or doubly circular linked list depending on one pointer or two pointers.



9/29/25

# Experiment 9

1)

```
# include <stdio.h>
# define SIZE 10
int q[SIZE]
int f = -1, r = -1;
void insert()
{
    int v;
    if (r == SIZE-1)
    {
        printf("Overflow");
        return;
    }
    printf("Enter value:");
    scanf("%d", &v);
    if (f == -1)
        f = 0;
    r++;
    q[r] = v;
    printf("Inserted %d", v);
}
void delete()
{
    if (f == -1 || f > r)
    {
        printf("Underflow");
        return;
    }
    printf("Deleted %d", q[f]);
    f++;
}
void dis()
{
    if (f == -1 || f > r)
    {
        printf("Empty");
        return;
    }
    printf("Queue:");
    for (int i=f; i<=r; i++)
        printf("%d", q[i]);
    printf("\n");
}
int main()
{
    int c;
    for(  ;  ;  )
    {
        printf("1. Insert 2. Delete 3. Display 4. Exit");
        scanf("%d", &c);
    }
    switch(c)
    {
        case 1:
            insert();
            break;
```

```
        case 2 :
            delete();
            break;
        case 3 :
            dis();
            break;
        case 4 :
            return 0;
    }
}
}
```

Output -

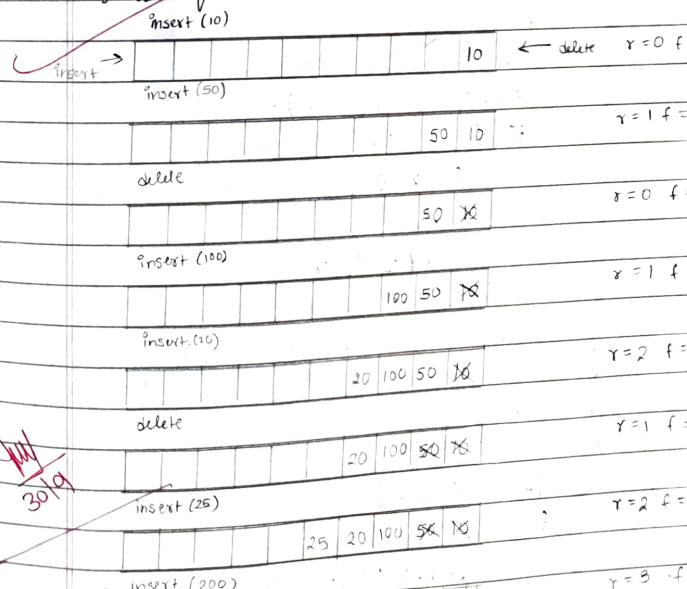
```
1. Insert   2. Delete   3. Dis   4. Exit
1
Enter value : 10
Inserted 10

1
Enter value : 50
Inserted 50

3
Queue : 10 50

4
```

SIZE -10

2) Insert (10), Insert (50), delete, Insert (100), Insert (20), delete, Insert (25), Insert (20

| | Operation | Queue | f | r |
|---|---|---|---|---|
| 1 | insert (10) | 10 | 0 | 0 |
| 2 | insert (50) | 10 50 | 0 | 1 |
| 3 | delete | 50 | 1 | 1 |
| 4 | insert (100) | 50 100 | 1 | 2 |
| 5 | insert (20) | 50 100 20 | 1 | 3 |
| 6 | delete | 100 20 | 2 | 3 |
| 7 | insert (25) | 100 20 25 | 2 | 4 |
| 8 | insert (200) | 100 20 25 200 | 2 | 5 |

Final queue elements : 100, 20, 25, 200



Monday

29/9/25

# Experiment 10

1) Primitive operation on circular queue

→

```
#include <stdio.h>
#define N 5
int q[N], f = -1, r = -1;
void i()
{
    int x;
    if (( f==0 && r == N-1) || (r+1)%N==f)
    {
        printf(" Overflow ");
        return;
    }
    printf(" Enter: ");
    scanf("%d", &x);
    if (f == -1)
        f = r = 0;
    else
        r = (r+1) % N;
    q[r] = x;
    printf(" Inserted %d ", x);
}
void d()
{
    if (f == -1)
    {
        if (f==r)
            f = r = -1;
        else
            f = (f+1)%N;
    }
    void dis()
    {
        if (f == -1)
        {
            printf(" Empty ");
            return;
        }
        int i=f;
        while (1)
        {
            printf("%d", q[i]);
            if (i==r)
                break;
            i = (i+1)%N;
        }
        printf(" ");
    }
    int main()
    {
        int c;
        for(  ;  ;  )
        {   printf(" 1. Insert 2. Delete
        3. Display 4. Exit ");
```

```
switch (c)
{
        case 1:
            i();
            break;
        case 2:
            d();
            break;
        case 3:
            dis();
            break;
        case 4:
            return 0;
    }
  }
}
```

Output -

```
1. Insert  2. Delete  3. Display  4. Exit
Enter : 1
Enter : 10
Inserted 10
1
Enter : 20
Inserted 20
3
10  20
4
---
```

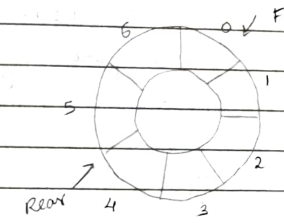
ii) What are the advantages of circular queue, show diagrammatic representation

→ Advantages -

i) It reuses the empty spaces left after deletion to avoid wastage.

ii) There is no need for shifting elements.

iii) Insertion and deletion takes less time as front and rear are updated




Circular Queue

# Extra Questions

Experiment 9 -

1) Explain the concept of priority queue in detail.

→ It is a abstract data type where each element has a priority associated with it. Elements are removed based on priority ( highest or lowest (first). If two have the same priority then it is FIFO.

There are two types -

i) Ascending priority queue - where the smallest element is removed first.
ii) Descending priority queue - where the largest element is removed first.

Applications -
i) Uses Huffman codes to compress data
ii) Many operating system algorithms.
iii) Traffic light controls with sensors
iv) Dijkstra algorithm to find shortest path.

2) Write algorithm for insertion and deletion operation in ~~linear~~ circular queue.

→ Insertion -

1 - If front = 0 and rear = max - 1
" Queue overflow "
if front = rear + 1
" Queue overflow "
2 - if front = -1 and rear = -1
front = rear = 0
~~if~~ else if
rear = max - 1 and front != 0
rear = 0
else
rear = rear + 1
end
3 - queue [ rear ] = val
4 - Exit

Deletion -
1 - If front = -1
write " underflow "
go to step 4
( end of if)
2 - set val = queue ( front )
3 - If front = rear
set front = rear = -1
else

if front = rear = max - 1
set front = 0
else
set front = front + 1
(end of if)
(end of if)
4 - Exit

3) Algorithms of Circular Queue

→ Insertion -

1 - If rear = max - 1
write " overflow "
go to step 4
(end of if)
2 - If fron = -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 - check if front = -1 or
front > rear then
front ++
else go to next value.
(end of if)
2 - Exit

4) Applications of Queue -

i) In CPU scheduling or task scheduling
ii) I/O Buffering for printer queue, keyboard buffer.
iii) In Breadth - first search in graphs
iv) In Data transfer
v) Simulation and event management.
vi) Level order traversal ( binary tree)
vii) Call Center / Ticketing system.
viii) Undo, redo

5) Difference Stack and Queue.

| | Stack | Queue |
|---|---|---|
| i) | Last in first out data structure | First in first out data structure |
| ii) | Insertion / deletion at one end (top) | Insertion takes place at rear, deletion at front. |
| iii) | Has only 1 pointer | 2 pointers |

| | | |
|---|---|---|
| iv) | No memory wastage | Wastage in linear queue |
| v) | Operations - push, pop | enqueue, dequeue |
| vi) | Computer systems used for procedure calls. | Used for time / resources. |
| Eg - | Plate counter at marriage | Student standing in a line at counter |

Experiment 12

a)

```
# include <stdio.h>
# define V 5
void init (int a[][V])
{
    int i,j;
    for(i=0; i<V; i++)
        for(j=0; j<V; j++)
            a[i][j] = 0;
}
void insertEdge (int a[][V], int i, int j)
{
    a[i][j] = 1;
    a[j][i] = 1;
}
Void printAdjMatrix (int a[][V])
{
    int i,j;
    for(i=0; i<V; i++)
    {
        printf("%d:", i);
        for(j=0; j<V; j++)
            printf("%d", a[i][j]);
    }
```

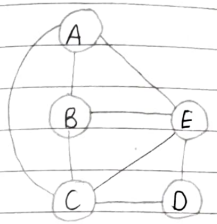
```
int main()
{
    int adjMatrix[v][v];
    init(adjMatrix);
    insertEdge(adjMatrix, 0, 1);
    insertEdge(adjMatrix, 0, 2);
    insertEdge(adjMatrix, 1, 2);
    insertEdge(adjMatrix, 2, 0);
    insertEdge(adjMatrix, 2, 3);
    insertEdge(adjMatrix, 0, 3);
    insertEdge(adjMatrix, 0, 0);
    insertEdge(adjMatrix, 1, 3);
    insertEdge(adjMatrix, 4, 3);
    insertEdge(adjMatrix, 2, 4);
    printAdjMatrix(adjMatrix);
    return 0;
}
```

Output -

```
0: 1 1 1 1 0
1: 1 0 1 1 0
2: 1 1 0 1 1
3: 1 1 1 0 1
4: 0 0 1 1 0
```

Theory Questions -

a)




| | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 0 | 1 |
| B | 1 | 0 | 1 | 0 | 1 |
| C | 1 | 1 | 0 | 1 | 1 |
| D | 0 | 0 | 1 | 0 | 1 |
| E | 1 | 1 | 1 | 1 | 0 |

A → [B] → [C] → [E]

B → [A] → [C] → [E]

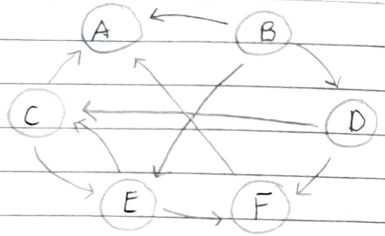
C → [A] → [B] → [D] → [E]

D → [C] → [E]

E → [A] → [B] → [C] → [D]

| | | | | |
|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 |

b)




| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| A | 0 | 0 | 0 | 0 | 0 | 0 |
| B | 1 | 0 | 0 | 1 | 1 | 0 |
| C | 1 | 0 | 0 | 0 | 1 | 0 |
| D | 0 | 0 | 1 | 0 | 0 | 1 |
| E | 0 | 0 | 1 | 0 | 0 | 1 |
| F | 1 | 0 | 0 | 0 | 0 | 1 |

A → NULL

B → [A] → [D] → [E]

C → [A] → [E]

D → [C] → [F]

E → [C] → [F]

F → [A]

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

3/11/25

# Experiment 11

a) Pre-order -

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int d;
    struct node *l, *r;
};
struct node * newN(int v)
{
    struct node * n = malloc(sizeof(struct
    node));
    n->d = v;
    n->l = n->r = NULL;
    return n;
}
struct node * ins(struct node * r, int v)
{
    if (r == NULL)
        return newN(v);
    if (v < r->d)
        r->l = ins(r->l, v);
    else if (v > r->d) r->r = ins(r->r,
    v);
    return r;
}
```

```
void pre (struct node * r)
{
    if (r)
    {
        printf ( "%d ", r->d);
        pre ( r->l);
        pre ( r->r);
    }
}
int main()
{
    struct node * r = NULL;
    int ch, v;
    while (1)
    {
        printf (" 1. Ins, 2. Pre, 3. Exit ");
        scanf ( "%d", &ch)
        {
            case 1 :
                printf ("Enter the value ");
                scanf ("%d ", &v);
                r = ins (r, v);
                break;
            case 2 :
                printf (" Preorder :");
                pre(r);
                printf ("\n");
                break;
            case 3 :
```

```
        }
    }
}
```

Output -

```
1. Ins, 2. Pre, 3. Exit
1
Enter the value : 10
1
Enter the value : 5
1
Enter the value : 15
2
Preorder : 10 5 15
```

b) Post - order -

```
# include <stdio.h>
# include <stdlib.h>
struct node
{
    int d;
    struct node *l, *r;
};
struct node * newN (int v)
{
    struct node * n = malloc ( sizeof (
    struct node));
    n -> d = v ;
    n -> l = n -> r = NULL ;
    return n ;
}
Struct node * ins ( struct node * r, int v)
{
    if ( r == NULL )
        return newN( v );
    if ( v < r -> d )
        r -> l = ins ( r -> l, v);
    else if ( v > r -> d)
        r -> r = ins ( r -> r, v);
    return r;
}
```

```
void post (struct node * r)
{
    if (r)
    {
        post (r -> l);
        port (r -> r);
        printf (" %d ", r -> d);
    }
}
int main()
{
    struct node * r = NULL;
    int ch, v;
    while (1)
    {
        printf (" 1. Ins, 2. Post, 3 Exit");
        scanf ("%d", &v);
        r = ins (r, v);
        break;
        case 2 :
            printf
        switch (ch)
        {
            case 1:
            printf (" Enter value : ");
            scanf (" %d ", &v);
            r = ins (r,v);
            break;
```

```
        post(r);
        printf("\n");
        break;
    case 3:
        exit(0);
    }
  }
}
```

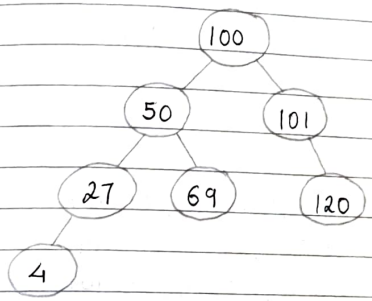
Output -

```
1. Ins 2. Post 3. Exit
1
Enter value : 10
1
Enter value : 5
1
Enter value : 15
2
Postorder : 5 15 10
```

Theory Questions -

a) 100 50 101 27 120 4 69

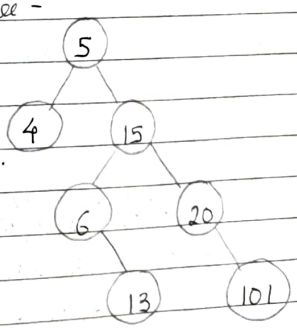
Binary Tree -




Preorder - 100, 50, 27, 4, 69, 101, 120
Inorder - 4, 27, 50, 69, 100, 101, 120
Postorder - 4, 27, 69, 50, 120, 101, 100

b) 5 15 4 6 20 13 101

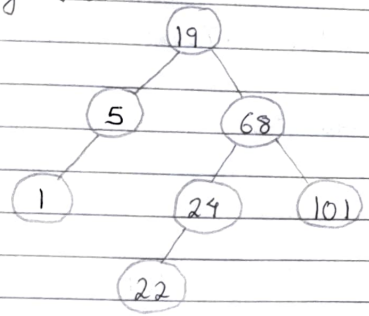
Binary Tree -




Preorder - 5, 4, 15, 6, 20, 13, 101
Inorder - 4, 5, 6, 13, 15, 20, 101
Postorder - 4, 13, 6, 101, 20, 15, 5

c) 19 5 68 24 101 1 22

Binary Tree -



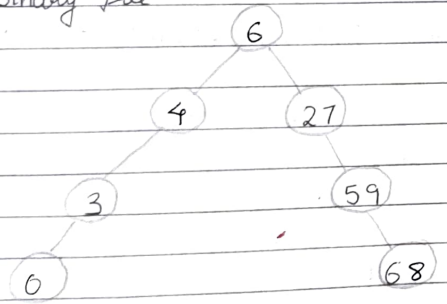

Preorder - 19, 5, 1, 68, 24, 22, 101

Postorder - 1, 5, 22, 24, 101, 68, 19

Inorder - 1, 5, 19, 22, 24, 68, 101

d) 6 4 3 27 59 0 68

Binary Tree -




6, 4, 3, 0, 27, 59, 68

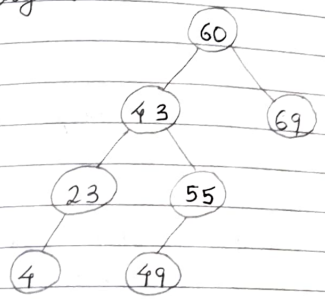
Pre order - ~~60, 43, 23, 4, 5~~

Post order - 0, 3, 4, 68, 59, 27, 6

In order - 0, 3, 4, 6, 27, 59, 68

e) 60 43 69 55 49 23 4

Binary Tree -




Pre order - 60, 43, 23, 4, 55, 49, 69

In order - 4, 23, 43, 49, 55, 60, 69

Post order - 4, 23, 49, 55, 43, 69, 60

3/11