DATA STRUCTURES LAB RECORD

TOTAL NO. OF EXPERIMENTS – 13

AMAL NATH M

R3 11

TKM19CS011

EXPERIMENT 1

a.) BUBBLE SORT

AIM :- To perform BUBBLE SORT in an array and to arrange the elements of the array in ascending order.

DATA STRUCTURE USED :- ARRAY is the data structure used in bubble sort.

ALGORITHM :-

START

1. Declare n, arr[SIZE], temp, i, j, count

2. Read n

3. i=0 till n, read arr[i]

4. i=0, count=0, flag=0

5. If i<n-1, goto 6. Else, goto 10.

6. j=0

7. If j<n-i-1, increment flag and goto 8. Else, goto 9.

8. If arr[j]>arr[j+1], swap these elements using temp, increment count and j and goto 7. Else, increment j and goto 7.

9. If flag is equal to 0, goto 10. Else, increment i and goto 5.

10. i=0 till n, print arr[i]

11. Print count and flag

STOP

PROGRAM CODE :-

```
#include<stdio.h>
void main()
{
	int i,n,a[100],temp,count=0,flag=0;

	printf("Enter the no. of integers\n");
	scanf("%d",&n);

	printf("Enter the array elements\n");
	for(i=0;i<n;i++)
		scanf("%d",&a[i]);

	for(int i=0;i<n-1;i++)
	{
```

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

		if(count==0)
			break;
	}

	printf("The elements in ascending order are\n");
	for(i=0;i<n;i++)
		printf("%d\n",a[i]);
	printf("\nNo. of swaps = %d\nNo. of iterations = %d\n", count,flag);
}
```

SAMPLE OUTPUTS :-

```
Enter the no. of integers
5
Enter the array elements
1
2
3
4
5
The elements in ascending order are
1
2
3
4
5

No. of swaps = 0
No. of iterations = 4

Enter the no. of integers
5
Enter the array elements
5
4
3
2
1
The elements in ascending order are
1
```

2
3
4
5

No. of swaps = 10
No. of iterations = 10

Enter the no. of integers
5
Enter the array elements
1
4
2
5
3
The elements in ascending order are
1
2
3
4
5

No. of swaps = 3
No. of iterations = 10

Enter the no. of integers
4
Enter the array elements
-1
23
-45
100
The elements in ascending order are
-45
-1
23
100

No. of swaps = 2
No. of iterations = 6

RESULT :- Bubble sort was performed in the array and the array elements were arranged in ascending order. Also, the number of comparisons and swaps performed were found out. Number of comparisons performed was found to be n(n-1)/2 where n is the number of array elements (Except for best case).
Time complexity :

Best case – O(n)
Average case – $O(n^2)$
Worst case – $O(n^2)$

b.) SELECTION SORT

AIM :- To perform SELECTION SORT in an array and to arrange the elements of the array in ascending order.

DATA STRUCTURE USED :- ARRAY is the data structure used in selection sort.

ALGORITHM :-

START

1. Declare n, arr[SIZE], temp, i, j, count, flag, min

2. Read n

3. i=0 till n, read arr[i]

4. i=0, count=0, flag=0

5. If i<n-1, goto 6. Else, goto 10

6. min=i, j=i+1

7. If j<n, increment flag and goto 8. Else, goto 9

8. If arr[j]>arr[j+1], assign min=j, increment j and goto 7. Else, increment j and goto 7.

9. if min!=1, swap arr[min] and arr[i], increment count and i and goto 5.

10. i=0 till n, print arr[i]

11. Print count and flag

STOP

PROGRAM CODE :-

```
#include<stdio.h>

void main()
{
	int n,arr[100],min,temp,count=0,flag=0;

	printf("Enter the no. of elements\n");
	scanf("%d",&n);

	printf("Enter array elements\n");
	for(int i=0;i<n;i++)
		scanf("%d",&arr[i]);

	for(int i=0;i<n-1;i++)
	{
		min=i;
```

```
		for(int j=i+1;j<n;j++)
		{
			flag++;
			if(arr[j]<arr[min])
			{
				min=j;
			}
		}

		if(min!=i)
		{
			temp=arr[min];
			arr[min]=arr[i];
			arr[i]=temp;
			count++;
		}
	}

	printf("The sorted elements are\n");
	for(int i=0;i<n;i++)
		printf("%d\n",arr[i]);
	printf("\nNo. of swaps = %d\nNo. of iterations = %d\n", count,flag);
}
```

SAMPLE OUTPUTS :-

```
Enter the no. of elements
5
Enter array elements
1
2
3
4
5
The sorted elements are
1
2
3
4
5

No. of swaps = 0
No. of iterations = 10

Enter the no. of elements
5
Enter array elements
5
4
3
2
1
```

The sorted elements are
1
2
3
4
5

No. of swaps = 2
No. of iterations = 10

Enter the no. of elements
5
Enter array elements
3
1
5
2
4
The sorted elements are
1
2
3
4
5

No. of swaps = 4
No. of iterations = 10

Enter the no. of elements
3
Enter array elements
100
0
-100
The sorted elements are
-100
0
100

No. of swaps = 1
No. of iterations = 3

RESULT :- Selection sort was performed in the array and the array elements were arranged in ascending order. Also, the number of comparisons and swaps performed were found out. Number of comparisons performed was found to be n(n-1)/2 where n is the number of array elements.

Time complexity :

Best case – $O(n^2)$

Average case – $O(n^2)$

Worst case – $O(n^2)$

c.) INSERTION SORT

AIM :- To perform INSERTION SORT in an array and to arrange the elements of the array in ascending order.

DATA STRUCTURE USED :- ARRAY is the data structure used in selection sort.

ALGORITHM :-

START

1. Declare n, temp, flag, count, arr[SIZE]

2. Read n

3. i=0 till n, read arr[i]

4. flag=0, count=0, i=1

5. If i<=n-1, goto 6. Else, goto 10.

6. Assign j=i

7. If j>0 and arr[j-1]>arr[j](increment count during each comparison), goto 8. Else, increment i and go to 5.

8. Swap arr[j-1] and arr[j] using temp.

9. Increment flag, decrement j and goto 7.

10 .i=0 till n, print arr[i]

11 .Print count and flag

STOP

PROGRAM CODE :-

```
#include <stdio.h>

void main()
{
   int n, i, j, temp, flag = 0, count=0;
   int arr[50];

   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 && count++ >= 0 && arr[j-1] > arr[j] )
         {
            temp     = arr[j];
            arr[j]   = arr[j-1];
            arr[j-1] = temp;
            j--;
            flag++;
         }

    }

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

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

    printf("Number of comparisons = %d\nNo. of swaps = %d\n", count, flag);
}
```

SAMPLE OUTPUTS :-

```
Enter number of elements
5
Enter 5 integers
1
2
3
4
5
Sorted list in ascending order:
1
2
3
4
5
Number of comparisons = 4
No. of swaps = 0

Enter number of elements
5
Enter 5 integers
5
4
3
2
1
Sorted list in ascending order:
1
2
```

3
4
5
Number of comparisons = 10
No. of swaps = 10

Enter number of elements
5
Enter 5 integers
3
4
1
5
2
Sorted list in ascending order:
1
2
3
4
5
Number of comparisons = 8
No. of swaps = 5

Enter number of elements
4
Enter 4 integers
-11
23
-1
0
Sorted list in ascending order:
-11
-1
0
23
Number of comparisons = 5
No. of swaps = 2

RESULT :- Insertion sort was performed in the array and the array elements were arranged in ascending order. Also, the number of comparisons and swaps performed were found out. Number of comparisons performed was found to be (n-1) for best case and n(n-1)/2 for worst case where n is the number of array elements.

Time complexity :

Best case – O(n)
Average case – $O(n^2)$
Worst case – $O(n^2)$

EXPERIMENT 2

a.) LINEAR SEARCH

AIM :- To perform LINEAR SEARCH in an array to find a specific element.

DATA STRUCTURE USED :- ARRAY is the data structure used in linear search.

ALGORITHM :-

START

1. Declare n, arr[SIZE], num, count=0, flag=0
2. Read n
3. From i=0 till n, read arr[i]
4. Read the search element num
5. From i=0 till n, in each iteration, increment count and check if arr[i] is equal to num.
6. If true, increment flag and break from the loop. Else, continue the next iteration.
7. If flag is not equal to 0, print num found. Else, print num not found.
8. Print count

STOP

PROGRAM CODE :-

```
#include<stdio.h>

void main()
{
	int n, arr[100], num, count=0, flag=0;

	printf("Enter the array size\n");
	scanf("%d", &n);

	printf("Enter the array elements\n");
	for(int i=0;i<n;i++)
		scanf("%d", &arr[i]);

	printf("Enter the number to be searched\n");
	scanf("%d", &num);

	for(int i=0;i<n;i++)
	{
		count++;
		if(arr[i]==num)
		{
```

```
				flag++;
				break;
			}
		}
		if(flag==0)
			printf("%d Not Found !\n", num);
		else
			printf("%d Found !\n", num);
		printf("\nNo. of comparisons = %d\n", count);
}
```

SAMPLE OUTPUTS :-

```
Enter the array size
5
Enter the array elements
1
6
2
8
3
Enter the number to be searched
1
1 Found !

No. of comparisons = 1

Enter the array size
4
Enter the array elements
1
2
3
4
Enter the number to be searched
4
4 Found !

No. of comparisons = 4

Enter the array size
3
Enter the array elements
1
2
3
Enter the number to be searched
4
4 Not Found !

No. of comparisons = 3
```

Enter the array size
6
Enter the array elements
-2
0
2
1
6
123
Enter the number to be searched
1
1 Found !

No. of comparisons = 4

RESULT :- Linear search was performed in an array and the required element, if present, was found out. Also, the number of comparisons performed was found out to be 1 for best case and n for worst case where n is the number of array elements.

Time complexity:

Best case – O(1)
Average case – O(n/2)
Worst case – O(n)

b.) BINARY SEARCH

AIM :- To perform BINARY SEARCH in a sorted array to find a specific element.

DATA STRUCTURE USED :- ARRAY is the data structure used in binary search.

ALGORITHM :-

START

1. Declare n, arr[SIZE], num, r=0, flag=0, count=0
2. Read n
3. From i=0 till n, read arr[i]
4. Read search element num
5. If n greater than or equal to r(index of first element), goto 6. Else, goto 10.
6. Assign mid=(n+r)/2
7. If arr[mid] equal to num, increment count and flag, then break and goto 10. Else, goto 8.
8. If arr[mid] less than num, assign r=mid+1, increment count and continue the next iteration by jumping to 5. Else, goto 9.
9. If arr[mid] greater than num, assign n=mid-1, increment count and continue the next iteration by jumping to 5.
10. If flag equal to zero, print num not found. Else, print num found.
11. Print count

STOP

PROGRAM CODE :-

```
#include <stdio.h>

void main()
{
	int n, mid, r=0, arr[100], num, flag=0, count=0;

	printf("Enter the array size\n");
	scanf("%d", &n);

	printf("Enter the array elements in ascending order\n");
	for(int i=0;i<n;i++)
		scanf("%d", &arr[i]);

	printf("Enter the number to be searched\n");
	scanf("%d", &num);
```

```
	while(n>=r)
	{
		mid=(n+r)/2;

		if(arr[mid]==num)
		{
			count++;
			flag++;
			break;
		}
		else if(arr[mid]<num)
		{
			r=mid+1;
			count++;
			continue;
		}
		else
		{
			n=mid-1;
			count++;
			continue;
		}
	}


	if(flag==0)
		printf("\n%d not found!\n", num);
	else
		printf("\n%d found!\n", num);

	printf("No. of comparisons = %d\n", count);
}
```

SAMPLE OUTPUTS :-

```
Enter the array size
5
Enter the array elements in ascending order
0
12
32
45
55
Enter the number to be searched
32

32 found!
No. of comparisons = 1

Enter the array size
4
Enter the array elements in ascending order
```

1
2
3
4
Enter the number to be searched
1

1 found!
No. of comparisons = 2

Enter the array size
6
Enter the array elements in ascending order
1
2
3
4
5
6
Enter the number to be searched
6

6 found!
No. of comparisons = 2

Enter the array size
3
Enter the array elements in ascending order
1
2
3
Enter the number to be searched
4

4 not found!
No. of comparisons = 3

Enter the array size
3
Enter the array elements in ascending order
11
22
33
Enter the number to be searched
1

1 not found!
No. of comparisons = 2

Enter the array size
4
Enter the array elements in ascending order

1
2
3
4
Enter the number to be searched
5

5 not found!
No. of comparisons = 3
Enter the array size
4
Enter the array elements in ascending order
4
5
6
7
Enter the number to be searched
1

1 not found!
No. of comparisons = 2

RESULT :- Binary search was performed in an already sorted array and the required element, if present, was found out. Also, the number of comparisons performed was found out to be 1 for best case and approximately (log_2n+1) for worst case where n is the number of array elements.
Time complexity:

Best case – O(1)
Average case – $O(log_2n)$
Worst case – $O(log_2n)$

EXPERIMENT 3

AIM :- Write a program to read two polynomials and store them in an array. Calculate the sum of the two polynomials and display the first polynomial, second polynomial and the resultant polynomial.

DATA STRUCTURE USED :- ARRAY is the data structure used.

ALGORITHM :-

```
START
   1  Initialise the exponent and coeffcient arrays and t1 (no of terms in p1), t2 (no of terms in p2)
   2  Read the first polynomial and store it in the p1 coeff and exp arrays
   3  Read the second polynomial to the p2 coeff and exp arrays
   4  while i<=t1 || j<=t2
         if i >= t1
                 p3.exp[k] = p2.exp[j]
                 p3.coeff[k] = p2.coeff[j]
                 j++, k++
         else if j >= t2
                 p3.exp[k] = p1.exp[i]
                 p3.coeff[k] = p1.coeff[i]
                 i++, k++
         else if  p1.exp[i] == p2.exp[j]
                 p3.coeff[k] = p1.coeff[i] + p2.coeff[j]
                 p3.exp[k] = p1.exp[i]
                 i++, j++, k++
         else if p1.exp[i] > p2.exp[j]
                 p3.exp[k] = p1.exp[i]
                 p3.coeff[k] = p1.coeff[i]
                 i++, k++
         else
                 p3.exp[k] = p2.exp[j]
                 p3.coeff[k] = p2.coeff[j]
                 j++, k++
   5. Print p1, p2 and p3
STOP
```

PROGRAM CODE :-

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

struct poly
{
	int coeff[20];
	int exp[20];
};
```

```
void main()
{
	struct poly p1, p2, p3;
	int t1, t2, i=0, j=0, k=0;

	printf("Enter the no. of terms in first polynomial\n");
	scanf("%d", &t1);

	printf("Enter the no. of terms in second polynomial\n");
	scanf("%d", &t2);

	printf("Enter the first polynomial (Eg:- ax^2 + b^x + c) in the decreasing order of the powers
of the terms \nand in such a way that the coefficient of the first term (Here,a) is entered first \nand
then the power of the variable in the same term (Here,2)\n");

	for(i=0;i<t1;i++)
	{
		scanf("%d", &p1.coeff[i]);
		scanf("%d", &p1.exp[i]);
	}

	printf("Enter the second polynomial as mentioned above\n");

	for(j=0;j<t2;j++)
	{
		scanf("%d", &p2.coeff[j]);
		scanf("%d", &p2.exp[j]);
	}

	i=0, j=0;

	while (i<t1 || j<t2)
	{
		if (i>=t1)
		{
			p3.exp[k] = p2.exp[j];
			p3.coeff[k] = p2.coeff[j];
			j++, k++;
		}

		else if (j>=t2)
		{
			p3.exp[k] = p1.exp[i];
			p3.coeff[k] = p1.coeff[i];
			i++, k++;
		}
		else if (p1.exp[i] == p2.exp[j])
		{
			p3.coeff[k] = p1.coeff[i] + p2.coeff[j];
			p3.exp[k] = p1.exp[i];
			i++, j++, k++;
		}
```

```
		else if (p1.exp[i] > p2.exp[j])
		{
			p3.exp[k] = p1.exp[i];
			p3.coeff[k] = p1.coeff[i];
			i++, k++;
		}

		else
		{
			p3.exp[k] = p2.exp[j];
			p3.coeff[k] = p2.coeff[j];
			j++, k++;
		}
	}

	printf("The sum of\n");

	for(i=0;i<t1;i++)
	{
		printf("(%dx^%d) ", p1.coeff[i], p1.exp[i]);
	}

	printf("and\n");

	for(j=0;j<t2;j++)
	{
		printf("(%dx^%d) ", p2.coeff[j], p2.exp[j]);
	}

	printf("is\n");

	for(i=0;i<k;i++)
	{
		printf("(%dx^%d) ", p3.coeff[i], p3.exp[i]);
	}

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

SAMPLE OUTPUTS :-

```
Enter the no. of terms in first polynomial
3
Enter the no. of terms in second polynomial
5
Enter the first polynomial (Eg:- ax^2 + b^x + c) in the decreasing order of the powers of the terms
and in such a way that the coefficient of the first term (Here,a) is entered first
and then the power of the variable in the same term (Here,2)
3
2
4
```

1
5
0
Enter the second polynomial as mentioned above
4
10
5
5
3
2
2
1
4
0
The sum of
(3x^2) (4x^1) (5x^0) and
(4x^10) (5x^5) (3x^2) (2x^1) (4x^0) is
(4x^10) (5x^5) (6x^2) (6x^1) (9x^0)

Enter the no. of terms in first polynomial
1
Enter the no. of terms in second polynomial
1
Enter the first polynomial (Eg:- ax^2 + b^x + c) in the decreasing order of the powers of the terms
and in such a way that the coefficient of the first term (Here,a) is entered first
and then the power of the variable in the same term (Here,2)
4
4
Enter the second polynomial as mentioned above
3
4
The sum of
(4x^4) and
(3x^4) is
(7x^4)

Enter the no. of terms in first polynomial
3
Enter the no. of terms in second polynomial
2
Enter the first polynomial (Eg:- ax^2 + b^x + c) in the decreasing order of the powers of the terms
and in such a way that the coefficient of the first term (Here,a) is entered first
and then the power of the variable in the same term (Here,2)
5
5
4
4
3
3
Enter the second polynomial as mentioned above
2
2

1
1
The sum of
(5x^5) (4x^4) (3x^3) and
(2x^2) (1x^1) is
(5x^5) (4x^4) (3x^3) (2x^2) (1x^1)

Enter the no. of terms in first polynomial
3
Enter the no. of terms in second polynomial
3
Enter the first polynomial (Eg:- ax^2 + b^x + c) in the decreasing order of the powers of the terms and in such a way that the coefficient of the first term (Here,a) is entered first
and then the power of the variable in the same term (Here,2)
6
6
5
5
4
0
Enter the second polynomial as mentioned above
4
3
2
1
5
0
The sum of
(6x^6) (5x^5) (4x^0) and
(4x^3) (2x^1) (5x^0) is
(6x^6) (5x^5) (4x^3) (2x^1) (9x^0)

RESULT :- Two polynomials are stored in an array and are added to obtain a resultant polynomial. All three polynomials are displayed.

EXPERIMENT 4

AIM :- Write a program to enter two matrices in normal form. Write a function to convert two matrices to tuple form and display it. Also find the transpose of the two matrices represented in tuple form and display it. Find the sum of the two matrices in tuple form and display the sum in tuple form.

DATA STRUCTURE USED :- ARRAY is the data structure used.

ALGORITHM :-

START

1. Accept the two matrix in normal form and R is the Resultant Matrix
2. Traverse throught the matrix such that k starts from 1
3. Find non zero values
4. Store its row in R[i][0] and column in R[i][1] and value in R[i][2]
5. Store R[0][0] = num of rows
6. Store R[0][1] = num of columns
7. Store R[0][0] = k-1 (Number of non-zero values)
8. Print the resultant Tuple Representation

9. Function Transpose(int sp[][3])
10. Check whether sp[0][2] is 0: then return "No elements"
11. Copy sp[0][0] into spt[0][0]
12. Copy sp[0][1] into spt[0][1]
13. Copy sp[0][2] into spt[0][2]
14. k = 1
15. for i=0 till number of columns
16. for j=1 till the number of non zero values
17. if i == a[j][1], insert the entire row into Resultant Array
18. k++
19. End if
20. End for
21. End for
22. Print Resultant Array

23. Function Addition(int sp1[][3],int sp2[][3])
24. If matrices doesn't match in size (i.e, rows and columns are not equal), print "Invalid operation"
25. Else
26. while i <= sp1[0][2] or j <= sp2[0][2] do
27. If sp1[i][0] < sp2[j][0]
28. Copy the data of i^{th} row of sp1 to Resultant, i++, k++
29. Else if sp1[i][0] > sp2[j][0]
30. Copy the data of j^{th} row of sp2 to Resultant, j++, k++
31. Else
32. If sp1[i][1] < sp2[j][1]
33. Copy the data of i^{th} row of sp1 to Resultant, i++, k++
34. Else if sp1[i][1] > sp2[j][1]
35. Copy the data of j^{th} row of sp2 to Resultant, j++, k++
36. Else
37. Add the values and insert to Resultant along with the row and column data, i++, j++, k++
38. End if
39. End if
40. End while
41. End if
42. Print the Resultant Tuple Representation

STOP

PROGRAM CODE :-

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

struct sparse
{
	int row, col;
	int arr[10][10];
	int tuple[100][3];
};


void readArray(struct sparse *sp, int i)
{
```

```
	printf("Enter no. of rows and columns of matrix %d\n", i);
	scanf("%d%d", &sp->row, &sp->col);

	printf("Enter the matrix %d elements\n", i);
	for(int i=0;i<sp->row;i++)
		for(int j=0;j<sp->col;j++)
			scanf("%d", &sp->arr[i][j]);
}

void dispArray(struct sparse *sp, int i)
{
	printf("Matrix %d:\n", i);

	for(int i=0;i<sp->row;i++)
	{
		for(int j=0;j<sp->col;j++)
			printf("%d ", sp->arr[i][j]);
		printf("\n");
	}
}


void dispTuple(struct sparse *sp, int i)
{
	printf("Tuple representation of Sparse Matrix %d:\n",i);
	for(int i=0;i<=sp->tuple[0][2];i++)
	{
		for(int j=0;j<3;j++)
			printf("%d ", sp->tuple[i][j]);
		printf("\n");
	}
}


void makeTuple(struct sparse *sp)
{
	int k=0;
	sp->tuple[0][0] = sp->row;
	sp->tuple[0][1] = sp->col;

	for(int i=0;i<sp->row;i++)
		for(int j=0;j<sp->col;j++)
			if(sp->arr[i][j] != 0)
			{
				k++;
				sp->tuple[k][0] = i;
				sp->tuple[k][1] = j;
				sp->tuple[k][2] = sp->arr[i][j];
			}
	sp->tuple[0][2] = k;
}
```

```
void makeArray(struct sparse *sp)
{
	sp->row = sp->tuple[0][0];
	sp->col = sp->tuple[0][1];

	for(int i=0; i<sp->row;i++)
		for(int j=0;j<sp->col;j++)
			sp->arr[i][j] = 0;

	for(int i=1;i<=sp->tuple[0][2];i++)
			sp->arr[sp->tuple[i][0]][sp->tuple[i][1]] = sp->tuple[i][2];
}


void transTuple(struct sparse *sp1, struct sparse *sp2, int a)
{

	if(sp1->tuple[0][2] == 0)
		printf("Matrix %d cannot be transposed!\n", a);
	else
	{
		sp2->tuple[0][0] = sp1->tuple[0][1];
		sp2->tuple[0][1] = sp1->tuple[0][0];
		sp2->tuple[0][2] = sp1->tuple[0][2];


		int k=1;

		for(int i=0;i<sp1->tuple[0][1];i++)
			for(int j=1;j<=sp1->tuple[0][2];j++)
				if(i == sp1->tuple[j][1])
				{
					sp2->tuple[k][0] = sp1->tuple[j][1];
					sp2->tuple[k][1] = sp1->tuple[j][0];
					sp2->tuple[k][2] = sp1->tuple[j][2];
					k++;
				}
		printf("Transpose of\n");
		dispTuple(sp2, a);
	}
}

void addTuple(struct sparse *sp1, struct sparse *sp2, struct sparse *sp3, int a, int b)
{
	int i=1, j=1, k=1;

	if(sp1->tuple[0][0] != sp2->tuple[0][0] || sp1->tuple[0][1] != sp2->tuple[0][1])
		printf("Matrix %d and Matrix %d cannot be added!\n", a, b);

	else
	{
```

```
while(i<=sp1->tuple[0][2] || j<=sp2->tuple[0][2])
{
	if(i>sp1->tuple[0][2])
	{
		sp3->tuple[k][0] = sp2->tuple[j][0];
		sp3->tuple[k][1] = sp2->tuple[j][1];
		sp3->tuple[k][2] = sp2->tuple[j][2];
		k++, j++;
	}
	else if(j>sp2->tuple[0][2])
	{
		sp3->tuple[k][0] = sp1->tuple[i][0];
		sp3->tuple[k][1] = sp1->tuple[i][1];
		sp3->tuple[k][2] = sp1->tuple[i][2];
		k++, i++;
	}
	else if(sp1->tuple[i][0] == sp2->tuple[j][0])
	{
		if(sp1->tuple[i][1] == sp2->tuple[j][1])
		{
			sp3->tuple[k][2] = sp1->tuple[i][2] + sp2->tuple[j][2];
			sp3->tuple[k][1] = sp1->tuple[i][1];
			sp3->tuple[k][0] = sp1->tuple[i][0];
			k++, i++, j++;
		}
		else if(sp1->tuple[i][1] < sp2->tuple[j][1])
		{
			sp3->tuple[k][0] = sp1->tuple[i][0];
			sp3->tuple[k][1] = sp1->tuple[i][1];
			sp3->tuple[k][2] = sp1->tuple[i][2];
			k++, i++;
		}
		else
		{
			sp3->tuple[k][0] = sp2->tuple[j][0];
			sp3->tuple[k][1] = sp2->tuple[j][1];
			sp3->tuple[k][2] = sp2->tuple[j][2];
			k++, j++;
		}
	}

	else if(sp1->tuple[i][0] < sp2->tuple[j][0])
	{
		sp3->tuple[k][0] = sp1->tuple[i][0];
		sp3->tuple[k][1] = sp1->tuple[i][1];
		sp3->tuple[k][2] = sp1->tuple[i][2];
		k++, i++;
	}
	else
	{
		sp3->tuple[k][0] = sp2->tuple[j][0];
		sp3->tuple[k][1] = sp2->tuple[j][1];
```

```
                    sp3->tuple[k][2] = sp2->tuple[j][2];
                    k++, j++;
                }
            }
            sp3->tuple[0][0] = sp1->tuple[0][0];
            sp3->tuple[0][1] = sp1->tuple[0][1];
            sp3->tuple[0][2] = k-1;

            printf("Sum of Matrix %d and %d:\n", a, b);
            dispTuple(sp3, 3);
    }
}


void main()
{
    struct sparse sp1, sp2, transp1, transp2, sumsp3;

    readArray(&sp1, 1);
    readArray(&sp2, 2);

    makeTuple(&sp1);
    makeTuple(&sp2);

    dispTuple(&sp1, 1);
    dispTuple(&sp2, 2);

    transTuple(&sp1, &transp1, 1);
    transTuple(&sp2, &transp2, 2);

    addTuple(&sp1, &sp2, &sumsp3, 1, 2);
}
```

SAMPLE OUTPUTS :-

Enter no. of rows and columns of matrix 1
3
3
Enter the matrix 1 elements
3
3
0
1
3
0
2
0
4
Enter no. of rows and columns of matrix 2
3
3
Enter the matrix 2 elements

0
0
1
1
2
3
0
4
5
Tuple representation of Sparse Matrix 1:
3 3 6
0 0 3
0 1 3
1 0 1
1 1 3
2 0 2
2 2 4
Tuple representation of Sparse Matrix 2:
3 3 6
0 2 1
1 0 1
1 1 2
1 2 3
2 1 4
2 2 5
Transpose of
Tuple representation of Sparse Matrix 1:
3 3 6
0 0 3
0 1 1
0 2 2
1 0 3
1 1 3
2 2 4
Transpose of
Tuple representation of Sparse Matrix 2:
3 3 6
0 1 1
1 1 2
1 2 4
2 0 1
2 1 3
2 2 5
Sum of Matrix 1 and 2:
Tuple representation of Sparse Matrix 3:
3 3 9
0 0 3
0 1 3
0 2 1
1 0 2
1 1 5
1 2 3

2 0 2
2 1 4
2 2 9

Enter no. of rows and columns of matrix 1
1
1
Enter the matrix 1 elements
1
Enter no. of rows and columns of matrix 2
1
1
Enter the matrix 2 elements
0
Tuple representation of Sparse Matrix 1:
1 1 1
0 0 1
Tuple representation of Sparse Matrix 2:
1 1 0
Transpose of
Tuple representation of Sparse Matrix 1:
1 1 1
0 0 1
Matrix 2 cannot be transposed!
Sum of Matrix 1 and 2:
Tuple representation of Sparse Matrix 3:
1 1 1
0 0 1

Enter no. of rows and columns of matrix 1
2
2
Enter the matrix 1 elements
1
0
0
0
Enter no. of rows and columns of matrix 2
3
3
Enter the matrix 2 elements
1
0
0
0
1
0
0

0
1
Tuple representation of Sparse Matrix 1:
2 2 1
0 0 1
Tuple representation of Sparse Matrix 2:
3 3 3
0 0 1
1 1 1
2 2 1
Transpose of
Tuple representation of Sparse Matrix 1:
2 2 1
0 0 1
Transpose of
Tuple representation of Sparse Matrix 2:
3 3 3
0 0 1
1 1 1
2 2 1
Matrix 1 and Matrix 2 cannot be added!

Enter no. of rows and columns of matrix 1
1
3
Enter the matrix 1 elements
1
2
3
Enter no. of rows and columns of matrix 2
3
1
Enter the matrix 2 elements
0
0
1
Tuple representation of Sparse Matrix 1:
1 3 3
0 0 1
0 1 2
0 2 3
Tuple representation of Sparse Matrix 2:
3 1 1
2 0 1
Transpose of
Tuple representation of Sparse Matrix 1:
3 1 3
0 0 1
1 0 2
2 0 3

Transpose of
Tuple representation of Sparse Matrix 2:
1 3 1
0 2 1
Matrix 1 and Matrix 2 cannot be added!

Enter no. of rows and columns of matrix 1
3
3
Enter the matrix 1 elements
0
0
0
0
0
0
0
2
0
Enter no. of rows and columns of matrix 2
3
3
Enter the matrix 2 elements
1
1
1
2
3
4
0
0
5
Tuple representation of Sparse Matrix 1:
3 3 1
2 1 2
Tuple representation of Sparse Matrix 2:
3 3 7
0 0 1
0 1 1
0 2 1
1 0 2
1 1 3
1 2 4
2 2 5
Transpose of
Tuple representation of Sparse Matrix 1:
3 3 1
1 2 2
Transpose of
Tuple representation of Sparse Matrix 2:

3 3 7
0 0 1
0 1 2
1 0 1
1 1 3
2 0 1
2 1 4
2 2 5
Sum of Matrix 1 and 2:
Tuple representation of Sparse Matrix 3:
3 3 8
0 0 1
0 1 1
0 2 1
1 0 2
1 1 3
1 2 4
2 1 2
2 2 5

RESULT :- Two sparse matrices entered in normal form are converted to their tuple forms. The tuple representations of their sum and each of their transposes are also found out.

EXPERIMENT 5

AIM :- Implement a Stack using arrays with the operations:

5.1. Pushing elements to the Stack.
5.2. Popping elements from the Stack.
5.3. Display the contents of the Stack after each operation.

DATA STRUCTURE USED :- STACK is the data structure used.

ALGORITHM :-

START

1. Initilize an array (STACK[]) and set STACK_TOP = -1
2. Choose functions according to the menu

Function PUSH(X)
1. If STACK is full (STACK_TOP >= STACK_SIZE-1), print "Stack Overflow"
2. Else, increase the value of STACK_TOP and assign STACK[STACK_TOP] = X
3. End if

Function POP()
1.If STACK is empty (STACK_TOP == -1), print "Stack Underflow"
2. Else, assign X = STACK[STACK_TOP], decrement STACK_TOP and return X
3. End if

Function DISPLAY()
1. for i=STACK_TOP till i=0, print STACK[i]

STOP

PROGRAM CODE :-

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

struct stack
{
	int size;
	int TOP;
	int *arr;
};


int isFull(struct stack *st)
{
	if(st->TOP >= st->size-1)
		return 1;
	return 0;
}
```

```
int isEmpty(struct stack *st)
{
	if(st->TOP == -1)
		return 1;
	return 0;
}

void push(struct stack *st)
{
	int x;

	if(isFull(st))
	{
		printf("\nStack Overflow\n\n");
	}

	else
	{
		printf("Enter element to push\n");
		scanf("%d", &x);

		st->arr[++st->TOP] = x;
	}
}
int pop(struct stack *st)
{
	if(isEmpty(st))
		printf("\nStack Underflow\n\n");

	else
	{
		int x = st->arr[st->TOP];
		st->TOP--;
		return x;
	}
}

void create(struct stack *st)
{
	printf("Enter stack size\n");
	scanf("%d", &st->size);

	st->arr = (int*) malloc (st->size * sizeof(int));

	st->TOP = -1;
}

void display(struct stack *st)
{
	printf("\nCURRENT STACK:\n");

	for(int i=st->TOP; i>=0; i--)
		printf("%d\n", st->arr[i]);
	printf("\n");
}

void main()
```

```
{
	struct stack st;
	int n;

	create(&st);
L1:
	printf("Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit\n");
	scanf("%d", &n);

	switch(n)
	{
		case 1:
			push(&st);
			goto L1;
		case 2:
			pop(&st);
			goto L1;
		case 3:
			display(&st);
			goto L1;
		case 4:
			exit(-1);
		default:
			printf("Enter valid input\n");
			goto L1;
	}
}
```

SAMPLE OUTPUTS :-

```
Enter stack size
5
Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
1
Enter element to push
10
Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
1
Enter element to push
8
Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
1
Enter element to push
6
Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
1
Enter element to push
5
Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
3

CURRENT STACK:
5
6
```

8
10

Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
2
Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
3

CURRENT STACK:
6
8
10

Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
1
Enter element to push
4
Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
1
Enter element to push
2
Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
3

CURRENT STACK:
2
4
6
8
10

Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
1

Stack Overflow

Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
2
Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
3

CURRENT STACK:
4
6
8
10

Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
2
Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
2
Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit

2
Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
2
Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
2

Stack Underflow

Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
3

CURRENT STACK:

Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
4

Enter stack size
3
Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
1
Enter element to push
1
Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
3

CURRENT STACK:
1

Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
1
Enter element to push
2
Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
3

CURRENT STACK:
2
1

Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
1
Enter element to push
3
Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
3

CURRENT STACK:
3
2

1

Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
1

Stack Overflow

Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
3

CURRENT STACK:
3
2
1

Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
2
Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
3

CURRENT STACK:
2
1

Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
2
Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
3

CURRENT STACK:
1

Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
2
Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
3

CURRENT STACK:

Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
2

Stack Underflow

Enter 1 to push, 2 to pop, 3 to display the stack and 4 to exit
4

RESULT :-

A Stack data structure is implemented using an array. PUSH(), POP() and DISPLAY() operations are performed on it.

EXPERIMENT 6

AIM :- Write a program to convert a given infix expression to its postfix expression and evaluate it.

DATA STRUCTURE USED :- STACK is the data structure used.

ALGORITHM :-

```
Algorithm infix_to_postfix
START
    TOP = -1, push('(')
    While TOP > -1 do
        ITEM = infix.Readsymbol()
        X = pop()
        Case : ITEM = Operand
            push(X)
            postfix(ITEM)
        Case : ITEM = ')'
            While X != '('
                postfix(X)
                X = pop()
            EndWhile
        Case : ISP(X) >= ICP(ITEM)
            While ISP(X) >= ICP(ITEM) do
                postfix(X)
                X = pop()
            EndWhile
            push(X)
            push(ITEM)
        Case : ISP(X) < ICP(ITEM)
            push(X)
            push(ITEM)
        Otherwise :
            Print "Invalid Expression"
    EndWhile
    Return postfix
STOP
```

```
Algorithm postfix_evaluation
START
    While (TOP >= -1) do
        ITEM = postfix.Readsymbol()
        Case : ITEM = Operand
            push(ITEM)
        Case : ITEM = Operator
            x2 = pop()
            x1 = pop()
            x = Operation(x1, x2, ITEM)
            push(x)
        Case : ITEM = '!'
            x = pop()
            push(-x);
        Case : ITEM = '#'
            x = pop()
            Return x
        Otherwise :
            Print "Invalid Expression"
    EndWhile
```

STOP

PROGRAM CODE :-

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

struct stack
{
	int TOP;
	int SIZE;
	char *arr;
	int *arr1;
};

struct expression
{
	char *infix;
	char *postfix;
};

void push(struct stack *s, char x)
{
	if(s->TOP >= s->SIZE-1)
	{
		printf("Cannot evaluate\n");
		exit(0);
	}

	else
	{
		s->arr[++s->TOP] = x;
	}
}

char pop(struct stack *s)
{
	if(s->TOP == -1)
	{
		printf("Cannot evaluate\n");
		exit(0);
	}

	else
	{
		char x = s->arr[s->TOP];
		s->TOP--;
		return x;
	}
}

void push1(struct stack *s, int x)
{
	if(s->TOP >= s->SIZE-1)
	{
		printf("Cannot evaluate\n");
		exit(0);
	}
```

```
	else
	{
		s->arr1[++s->TOP] = x;
	}
}

int pop1(struct stack *s)
{
	if(s->TOP == -1)
	{
		printf("Cannot evaluate\n");
		exit(0);
	}

	else
	{
		int x = s->arr1[s->TOP];
		s->TOP--;
		return x;
	}
}

int ISP(char X)
{
	if(X == '+' || X == '-')
		return 2;
	else if(X == '*' || X == '/')
		return 4;
	else if(X == '^')
		return 5;
	else if(X >= '0' && X <= '9' || X >= 'a' && X<= 'z' || X >= 'A' && X<= 'Z')
		return 8;
	else if(X == '(')
		return 0;
}

int ICP(char X)
{
	if(X == '+' || X == '-')
		return 1;
	else if(X == '*' || X == '/')
		return 3;
	else if(X == '^')
		return 6;
	else if(X >= '0' && X <= '9' || X >= 'a' && X<= 'z' || X >= 'A' && X<= 'Z')
		return 7;
	else if(X == '(')
		return 9;
	else if(X == ')')
		return 0;
}

void infix_to_postfix(struct expression *exp)
{
	int i=0, j=0;
	struct stack s;

	s.TOP = -1;
	s.SIZE = strlen(exp->infix);
```

```
	s.arr = (char*) malloc(s.SIZE * sizeof(char));

	push(&s, '(');

	while(s.TOP > -1)
	{
		char X = pop(&s);

		if(exp->infix[i] == '(')
		{
			push(&s, X);
			push(&s, exp->infix[i]);
		}
		else if(exp->infix[i] == ')')
		{
			while(X != '(')
			{
				exp->postfix[j] = X;
				X = pop(&s);
				j++;
			}
		}
		else if(exp->infix[i] >= 'a' && exp->infix[i] <= 'z' || exp->infix[i] >= 'A' && exp->infix[i] <= 'Z'
|| exp->infix[i] >= '0' && exp->infix[i] <= '9')
		{
			push(&s, X);
			exp->postfix[j] = exp->infix[i];
			j++;
		}
		else if(ISP(X) >= ICP(exp->infix[i]))
		{
			while(ISP(X) >=  ICP(exp->infix[i]))
			{
				exp->postfix[j] = X;
				X = pop(&s);
				j++;
			}
			push(&s, X);
			push(&s, exp->infix[i]);
		}
		else if(ISP(X) < ICP(exp->infix[i]))
		{
			push(&s, X);
			push(&s, exp->infix[i]);
		}
		else if(exp->infix[i] == ' ')
		{
			//skip
		}
		else
		{
			printf("INVALID EXPRESSION!\n");
			exit(0);
		}

		i++;
	}
}

int evaluate_postfix(char* postfix)
```

```
{
	postfix[strlen(postfix)] = '#';

	int i=0;
	int exp[strlen(postfix)];

	while(i<strlen(postfix))
	{
		if(postfix[i] >= 'a' && postfix[i] <= 'z' || postfix[i] >= 'A' && postfix[i] <= 'Z')
		{
			printf("Enter value for %c\n", postfix[i]);
			scanf("%d", &exp[i]);
		}
		else if(postfix[i] >= '0' && postfix[i] <= '9')
		{
			exp[i] = postfix[i] - 48;
		}
		i++;
	}

	int x1, x2;
	struct stack s;
	i = 0;

	s.TOP = -1;
	s.SIZE = strlen(postfix);
	s.arr1 = (int*) malloc(s.SIZE * sizeof(int));

	while(1)
	{
		switch(postfix[i])
		{
			case '+' :
				x2 = pop1(&s);
				x1 = pop1(&s);
				push1(&s, x1+x2);
				break;
			case '-' :
				x2 = pop1(&s);
				x1 = pop1(&s);
				push1(&s, x1-x2);
				break;
			case '*' :
				x2 = pop1(&s);
				x1 = pop1(&s);
				push1(&s, x1*x2);
				break;
			case '/' :
				x2 = pop1(&s);
				x1 = pop1(&s);
				push1(&s, x1/x2);
				break;
			case '^' :
				x2 = pop1(&s);
				x1 = pop1(&s);
				push1(&s, pow(x1, x2));
				break;
			case '!' :
				x1 = pop1(&s);
				push1(&s, -x1);
```

```
                    break;
                case '#' :
                    x1 = pop1(&s);
                    return x1;
                default :
                    push1(&s, exp[i]);
            }
            i++;
    }
}

void main()
{
    struct expression exp;

    exp.infix = (char*) malloc(100 * sizeof(char));
    exp.postfix = (char*) malloc(strlen(exp.infix) * sizeof(char));

    printf("Enter expression to be evaluated:\n");
    printf("NB:- For numbers with more than 1 digit, enter a variable (eg:- A, x, y, etc...) and for negative
numbers add '!' after it.\n");
    fgets(exp.infix, 100, stdin);

    exp.infix[strlen(exp.infix)-1] = ')';

    infix_to_postfix(&exp);

    puts(exp.postfix);

    printf("Result = %d\n", evaluate_postfix(exp.postfix));
}
```

SAMPLE OUTPUTS :-

```
Enter expression to be evaluated:
NB:- For numbers with more than 1 digit, enter a variable (eg:- A, x, y, etc...) and for negative numbers add
'!' after it.
a+b^3*5
ab3^5*+
Enter value for a
10
Enter value for b
5
Result = 635


Enter expression to be evaluated:
NB:- For numbers with more than 1 digit, enter a variable (eg:- A, x, y, etc...) and for negative numbers add
'!' after it.
5+6/3*6^2
563/62^*+
Result = 77


Enter expression to be evaluated:
NB:- For numbers with more than 1 digit, enter a variable (eg:- A, x, y, etc...) and for negative numbers add
'!' after it.
a/b+c*d-e
```

ab/cd*+e-
Enter value for a
5
Enter value for b
4
Enter value for c
3
Enter value for d
2
Enter value for e
1
Result = 6

Enter expression to be evaluated:
NB:- For numbers with more than 1 digit, enter a variable (eg:- A, x, y, etc...) and for negative numbers add '!' after it.
a/0
a0/
Enter value for a
5
Floating point exception (core dumped)

Enter expression to be evaluated:
NB:- For numbers with more than 1 digit, enter a variable (eg:- A, x, y, etc...) and for negative numbers add '!' after it.
a^b/(3*5-5)
ab^35*5-/
Enter value for a
10
Enter value for b
2
Result = 10

Enter expression to be evaluated:
NB:- For numbers with more than 1 digit, enter a variable (eg:- A, x, y, etc...) and for negative numbers add '!' after it.
a^b/3*5-5
ab^3/5*5-
Enter value for a
10
Enter value for b
2
Result = 160

Enter expression to be evaluated:
NB:- For numbers with more than 1 digit, enter a variable (eg:- A, x, y, etc...) and for negative numbers add '!' after it.
a+b)
ab+
Enter value for a
123
Enter value for b
456

Result = 579

Enter expression to be evaluated:
NB:- For numbers with more than 1 digit, enter a variable (eg:- A, x, y, etc...) and for negative numbers add '!' after it.
(a+b
Cannot evaluate

Enter expression to be evaluated:
NB:- For numbers with more than 1 digit, enter a variable (eg:- A, x, y, etc...) and for negative numbers add '!' after it.
a++
a++
Enter value for a
3
Cannot evaluate

Enter expression to be evaluated:
NB:- For numbers with more than 1 digit, enter a variable (eg:- A, x, y, etc...) and for negative numbers add '!' after it.
!a
a!
Enter value for a
435
Result = -435

Enter expression to be evaluated:
NB:- For numbers with more than 1 digit, enter a variable (eg:- A, x, y, etc...) and for negative numbers add '!' after it.
a+!b
ab!+
Enter value for a
3
Enter value for b
2
Result = 1

RESULT :- Given infix expression is converted to postfix form and then the result of the expression is displayed.

Time complexity for infix to postfix conversion – O(n)

Time complexity for postfix evaluation – O(n)

EXPERIMENT 7

AIM :- Perform the following operations on various Queue data structures implemented using arrays :

1. Insertion

2. Deletion

3. Display

DATA STRUCTURE USED :- QUEUE is the data structure used.

ALGORITHM :-

Algorithm Insert_Front_DQ

START

```
If FRONT = 0
        print "Cannot perform insertion at FRONT”
Else if FRONT = -1
        FRONT = 0
        REAR = 0
        DQ[FRONT] = X
Else
        FRONT -= 1
        DQ[FRONT] = X
End If
```

STOP

Algorithm Insert_Rear_DQ

START

```
If REAR = SIZE-1
        print Queue is full!”
Else
        If FRONT = -1
                FRONT = 0
        End If
        REAR += 1
        DQ[REAR] = X
End if
End
```

STOP

Algorithm Delete_Front_DQ

START

```
If FRONT = -1
        print "Queue is empty!"
```

```
Else
    X = DQ[FRONT]
    If FRONT = REAR
        FRONT = -1
        REAR = -1
    Else
        FRONT += 1
    End if
    Return X
End if
```
STOP

Algorithm Delete_Rear_DQ

START

```
If REAR = -1
    print “Queue is empty!"
Else
    X = DQ[REAR]
    If FRONT = REAR
        FRONT = -1
        REAR = -1
    Else
        REAR -= 1
    End if
 End if
    Return X
```
STOP

Algorithm Insert_Front_CDQ

START

```
If FRONT = (REAR + 1) % SIZE
    print "Queue is full!"
Else if FRONT = -1
    FRONT = 0
    REAR = 0
    CDQ[FRONT] = X
Else
    FRONT = (FRONT + SIZE-1) % SIZE
    CDQ[FRONT] = X
End if
```
STOP

Algorithm Insert_Rear_CDQ

START

```
If FRONT = (REAR + 1) % SIZE
    print "Queue is full!"
Else if FRONT = -1
    FRONT = 0
    REAR = 0
```

```
        CDQ[REAR] = X
Else
        REAR = (REAR+1) % SIZE
        CDQ[REAR] = X
End if
STOP
```

Algorithm Delete_Front_CDQ (Same for Delete_Front_PQ)

```
START
If FRONT = -1
        print "Queue is empty!"
Else
        X = CDQ[FRONT]
        If FRONT = REAR
                FRONT = -1
                REAR = -1
        Else
                FRONT = (FRONT+1) % SIZE
        End if
        Return X
End if
STOP
```

Algorithm Delete_Rear_CDQ

```
START
If REAR = -1
        print "Queue is empty!"
Else
        X = CDQ[REAR]
        If FRONT = REAR
                FRONT = -1
                REAR = -1
        Else
                REAR = (REAR + SIZE-1) % SIZE
        End if
        Return X
End if
STOP
```

Algorithm Insert_Rear_PQ

```
START
If (FRONT = (REAR + 1) % SIZE)
        print "Queue is full!”
Else if (FRONT == -1)
        FRONT = 0
        REAR = 0
        PQ[REAR] = X
        PRIOARR[REAR] = priority //Priority index of element
Else
```

```
            REAR = (REAR+1) % SIZE
            PQ[REAR] = X;
            PRIOARR[REAR] = priority
    End if
STOP
```

Algorithm Sort_PQ

START

```
    IF (FRONT == REAR)
            //do nothing
    Else if (FRONT < REAR)
            For i = (FRONT+1) % SIZE till i <= REAR do
                    j = i
                    While (j > FRONT && PRIOARR[j] < PRIOARR[j-1])
                            Swap (PRIOARR[j], PRIOARR[j-1])
                            Swap (  PQ[j], PQ[j-1])
                            j--
                    End while
            End for
    Else
            For i = (FRONT+1) % SIZE do
                    j = i
                    While PRIOARR[j] < PRIOARR[(j+SIZE-1) % SIZE])
                            Swap (PRIOARR[j], PRIOARR[(j+q->SIZE-1) % q->SIZE])
                            Swap (  PQ[j], PQ[(j+q->SIZE-1) % q->SIZE])
                            j = (j+q->SIZE-1)%q->SIZE; //Increment condition
                            If(j = q->FRONT)
                                    Exit while
                            End if
                    End while
                    If (i = q->REAR)
                            Exit for
                    i = (i+1) % SIZE //Increment condition
            End for
    End if
```

PROGRAM CODE :-

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

struct queue
{
        int FRONT;
        int REAR;
        int *arr;
        int SIZE;
        int count;
        int *prioarr;
};

void insert_rear_dq(struct queue *q)
{
```

```
	int X;

	if(q->REAR == q->SIZE-1)
		printf("Queue is full!\n");
	else
	{
		printf("Enter the number to be inserted\n");
		scanf("%d", &X);

		if(q->FRONT == -1)
			q->FRONT = 0;
		q->REAR += 1;
		q->arr[q->REAR] = X;
	}
}

void insert_front_dq(struct queue *q)
{
	int X;

	if(q->FRONT == 0)
		printf("Cannot perform insertion at FRONT!\n");

	else if(q->FRONT == -1)
	{
		printf("Enter the number to be inserted\n");
		scanf("%d", &X);

		q->FRONT = 0;
		q->REAR = 0;
		q->arr[q->FRONT] = X;
	}
	else
	{
		printf("Enter the number to be inserted\n");
		scanf("%d", &X);

		q->FRONT -= 1;
		q->arr[q->FRONT] = X;
	}
}

int delete_rear_dq(struct queue *q)
{
	if(q->REAR == -1)
		printf("Queue is empty!\n");
	else
	{
		int X = q->arr[q->REAR];

		if(q->FRONT == q->REAR)
		{
			q->FRONT = -1;
			q->REAR = -1;
		}
		else
			q->REAR -= 1;
```

```
		return X;
	}
}

int delete_front_dq(struct queue *q)
{
	if(q->FRONT == -1)
		printf("Queue is empty!\n");
	else
	{
		int X = q->arr[q->FRONT];

		if(q->FRONT == q->REAR)
		{
			q->FRONT = -1;
			q->REAR = -1;
		}
		else
			q->FRONT += 1;
		return X;
	}
}

void insert_rear_cdq(struct queue *q)
{
	int X;

	if(q->FRONT == (q->REAR + 1) % q->SIZE)
		printf("Queue is full!\n");

	else if(q->FRONT == -1)
	{
		printf("Enter the number to be inserted\n");
		scanf("%d", &X);

		q->FRONT = 0;
		q->REAR = 0;
		q->arr[q->REAR] = X;
	}
	else
	{
		printf("Enter the number to be inserted\n");
		scanf("%d", &X);

		q->REAR = (q->REAR+1) % q->SIZE;
		q->arr[q->REAR] = X;
	}
}

void insert_front_cdq(struct queue *q)
{
	int X;

	if(q->FRONT == (q->REAR + 1) % q->SIZE)
		printf("Queue is full!\n");

	else if(q->FRONT == -1)
```

```
	{
		printf("Enter the number to be inserted\n");
		scanf("%d", &X);

		q->FRONT = 0;
		q->REAR = 0;
		q->arr[q->FRONT] = X;
	}
	else
	{
		printf("Enter the number to be inserted\n");
		scanf("%d", &X);

		q->FRONT = (q->FRONT + q->SIZE-1) % q->SIZE;
		q->arr[q->FRONT] = X;
	}
}

void insert_rear_pq(struct queue *q)
{
	int X;
	int priority;

	if(q->FRONT == (q->REAR + 1) % q->SIZE)
		printf("Queue is full!\n");

	else if(q->FRONT == -1)
	{
		printf("Enter the number to be inserted\n");
		scanf("%d", &X);

		printf("Enter the index priority of the element (Most priority = 0)\n");
		scanf("%d", &priority);

		q->FRONT = 0;
		q->REAR = 0;

		q->arr[q->REAR] = X;
		q->prioarr[q->REAR] = priority;
	}
	else
	{
		printf("Enter the number to be inserted\n");
		scanf("%d", &X);

		printf("Enter the index priority of the element (Most priority = 0)\n");
		scanf("%d", &priority);

		q->REAR = (q->REAR+1) % q->SIZE;

		q->arr[q->REAR] = X;
		q->prioarr[q->REAR] = priority;
	}
}

void pq_sort(struct queue *q)
{
```

```
	if(q->FRONT == q->REAR)
	{
		//do nothing
	}
	else if(q->FRONT < q->REAR)
		for(int i = (q->FRONT+1)%q->SIZE; i <= q->REAR; i++)
		{
			int j = i;

			while(j > q->FRONT && q->prioarr[j] < q->prioarr[j-1])
			{
				//swap prioarr
				q->prioarr[j] += q->prioarr[j-1];
				q->prioarr[j-1] = q->prioarr[j] - q->prioarr[j-1];
				q->prioarr[j] -= q->prioarr[j-1];

				//swap queue
				q->arr[j] += q->arr[j-1];
				q->arr[j-1] = q->arr[j] - q->arr[j-1];
				q->arr[j] -= q->arr[j-1];

				j--;
			}
		}
	else
		for(int i = (q->FRONT+1)%q->SIZE; ; i = ((i+1) % q->SIZE))
		{
			int j = i;

			while(q->prioarr[j] < q->prioarr[(j+q->SIZE-1) % q->SIZE])
			{
				//swap prioarr
				q->prioarr[j] += q->prioarr[(j+q->SIZE-1) % q->SIZE];
				q->prioarr[(j+q->SIZE-1) % q->SIZE] = q->prioarr[j] - q->prioarr[(j+q-
>SIZE-1) % q->SIZE];
				q->prioarr[j] -= q->prioarr[(j+q->SIZE-1) % q->SIZE];

				//swap queue
				q->arr[j] += q->arr[(j+q->SIZE-1) % q->SIZE];
				q->arr[(j+q->SIZE-1) % q->SIZE] = q->arr[j] - q->arr[(j+q->SIZE-1) % q-
>SIZE];
				q->arr[j] -= q->arr[(j+q->SIZE-1) % q->SIZE];

				j = (j+q->SIZE-1)%q->SIZE;

				if(j == q->FRONT)
					break;
			}

			if(i == q->REAR)
				break;
		}
}

int delete_rear_cdq(struct queue *q)
{
	if(q->REAR == -1)
```

```
		printf("Queue is empty!\n");
	else
	{
		int X = q->arr[q->REAR];

		if(q->FRONT == q->REAR)
		{
			q->FRONT = -1;
			q->REAR = -1;
		}
		else
			q->REAR = (q->REAR + q->SIZE-1) % q->SIZE;
		return X;
	}
}

int delete_front_cdq(struct queue *q)
{
	if(q->FRONT == -1)
		printf("Queue is empty!\n");
	else
	{
		int X = q->arr[q->FRONT];

		if(q->FRONT == q->REAR)
		{
			q->FRONT = -1;
			q->REAR = -1;
		}
		else
			q->FRONT = (q->FRONT+1) % q->SIZE;
		return X;
	}
}

void create(struct queue *q, int flag)
{
	printf("\nEnter size of queue\n");
	scanf("%d", &q->SIZE);

	if(flag == 7)
		q->prioarr = malloc(q->SIZE * sizeof(int));

	q->arr = malloc(q->SIZE * sizeof(int));

	q->FRONT = -1;
	q->REAR = -1;
}

void display(struct queue *q)
{
	printf("\nCurrent queue:\n");

	if(q->FRONT == -1)
		printf("\n");
	else
	{
```

```
			if(q->FRONT <= q->REAR)
				for(int i=q->FRONT; i <= q->REAR; i++)
					printf("%d\n", q->arr[i]);
			else
			{
				for(int i=q->FRONT; i < q->SIZE; i++)
					printf("%d\n", q->arr[i]);
				for(int i=0; i <= q->REAR; i++)
					printf("%d\n", q->arr[i]);
			}
		}
}

void main()
{
		int flag;
		struct queue q;

L:
		printf("Which type of queue do you want to use ? \n(Enter the corresponding number)\n");

		printf("1.Simple Queue\n2.Circular Queue\n3.Double-ended Queue (Deque)\n4.Circular Deque\
n5.Input-restricted Queue\n6.Output-restricted Queue\n7.Priority Queue\n");

		scanf("%d", &flag);

		create(&q, flag);

		switch(flag)
		{
			case 1 :
L1:
				printf("\nEnter the corresponding number for the given operations:\n");

				printf("\t1.ENQUEUE\n\t2.DEQUEUE\n\t3.DISPLAY Queue\n\t4.EXIT Program\
n");

				scanf("%d", &flag);

				switch(flag)
				{
					case 1 :
						insert_rear_dq(&q);
						goto L1;
					case 2 :
						delete_front_dq(&q);
						goto L1;
					case 3 :
						display(&q);
						goto L1;
					case 4 :
						exit(0);
					default :
						printf("Enter valid number!\n");
						goto L1;
				}
```

```
			case 2 :
L2:
				printf("\nEnter the corresponding number for the given operations:\n");

				printf("\t1.ENQUEUE\n\t2.DEQUEUE\n\t3.DISPLAY Queue\n\t4.EXIT Program\
n");

				scanf("%d", &flag);

				switch(flag)
				{
					case 1 :
						insert_rear_cdq(&q);
						goto L2;
					case 2 :
						delete_front_cdq(&q);
						goto L2;
					case 3 :
						display(&q);
						goto L2;
					case 4 :
						exit(0);
					default :
						printf("Enter valid number!\n");
						goto L2;
				}

			case 3 :
L3:
				printf("\nEnter the corresponding number for the given operations:\n");

				printf("\t1.INSERT at FRONT\n\t2.INSERT at REAR\n\t3.DELETE from FRONT\
n\t4.DELETE from REAR\n\t5.DISPLAY Queue\n\t6.EXIT Program\n");

				scanf("%d", &flag);

				switch(flag)
				{
					case 1 :
						insert_front_dq(&q);
						goto L3;
					case 2 :
						insert_rear_dq(&q);
						goto L3;
					case 3 :
						delete_front_dq(&q);
						goto L3;
					case 4 :
						delete_rear_dq(&q);
						goto L3;
					case 5 :
						display(&q);
						goto L3;
					case 6 :
						exit(0);
					default :
						printf("Enter valid number!\n");
```

```
                    goto L3;
            }

        case 4 :
L4:
            printf("\nEnter the corresponding number for the given operations:\n");

            printf("\t1.INSERT at FRONT\n\t2.INSERT at REAR\n\t3.DELETE from FRONT\
n\t4.DELETE from REAR\n\t5.DISPLAY Queue\n\t6.EXIT Program\n");

            scanf("%d", &flag);

            switch(flag)
            {
                case 1 :
                    insert_front_cdq(&q);
                    goto L4;
                case 2 :
                    insert_rear_cdq(&q);
                    goto L4;
                case 3 :
                    delete_front_cdq(&q);
                    goto L4;
                case 4 :
                    delete_rear_cdq(&q);
                    goto L4;
                case 5 :
                    display(&q);
                    goto L4;
                case 6 :
                    exit(0);
                default :
                    printf("Enter valid number!\n");
                    goto L4;
            }

        case 5 :
L5:
            printf("\nEnter the corresponding number for the given operations:\n");

            printf("\t1.INSERT at REAR\n\t2.DELETE from FRONT\n\t3.DELETE from
REAR\n\t4.DISPLAY Queue\n\t5.EXIT Program\n");

            scanf("%d", &flag);

            switch(flag)
            {
                case 1 :
                    insert_rear_cdq(&q);
                    goto L5;
                case 2 :
                    delete_front_cdq(&q);
                    goto L5;
                case 3 :
                    delete_rear_cdq(&q);
                    goto L5;
                case 4 :
```

```
						display(&q);
						goto L5;
					case 5 :
						exit(0);
					default :
						printf("Enter valid number!\n");
						goto L5;
				}

			case 6 :
L6:
				printf("\nEnter the corresponding number for the given operations:\n");

				printf("\t1.INSERT at FRONT\n\t2.INSERT at REAR\n\t3.DELETE from FRONT\
n\t4.DISPLAY Queue\n\t5.EXIT Program\n");

				scanf("%d", &flag);

				switch(flag)
				{
					case 1 :
						insert_front_cdq(&q);
						goto L6;
					case 2 :
						insert_rear_cdq(&q);
						goto L6;
					case 3 :
						delete_front_cdq(&q);
						goto L6;
					case 4 :
						display(&q);
						goto L6;
					case 5 :
						exit(0);
					default :
						printf("Enter valid number!\n");
						goto L6;
				}
			case 7 :
L7:
				printf("\nEnter the corresponding number for the given operations:\n");

				printf("\t1.ENQUEUE\n\t2.DEQUEUE\n\t3.DISPLAY Queue\n\t4.EXIT Program\
n");

				scanf("%d", &flag);

				switch(flag)
				{
					case 1 :
						insert_rear_pq(&q);
						pq_sort(&q);
						goto L7;
					case 2 :
						delete_front_cdq(&q);
						goto L7;
					case 3 :
```

```
                        display(&q);
                        goto L7;
                case 4 :
                        exit(0);
                default :
                        printf("Enter valid number!\n");
                        goto L7;
            }

        default :
            printf("Enter valid number!");
            goto L;
    }
}
```

SAMPLE OUTPUT :-

```
Which type of queue do you want to use ?
(Enter the corresponding number)
1.Simple Queue
2.Circular Queue
3.Double-ended Queue (Deque)
4.Circular Deque
5.Input-restricted Queue
6.Output-restricted Queue
7.Priority Queue
1

Enter size of queue
3

Enter the corresponding number for the given operations:
        1.ENQUEUE
        2.DEQUEUE
        3.DISPLAY Queue
        4.EXIT Program
1
Enter the number to be inserted
1

Enter the corresponding number for the given operations:
        1.ENQUEUE
        2.DEQUEUE
        3.DISPLAY Queue
        4.EXIT Program
1
Enter the number to be inserted
2

Enter the corresponding number for the given operations:
        1.ENQUEUE
        2.DEQUEUE
```

3.DISPLAY Queue
4.EXIT Program
1
Enter the number to be inserted
3

Enter the corresponding number for the given operations:
1.ENQUEUE
2.DEQUEUE
3.DISPLAY Queue
4.EXIT Program
3

Current queue:
1
2
3

Enter the corresponding number for the given operations:
1.ENQUEUE
2.DEQUEUE
3.DISPLAY Queue
4.EXIT Program
2

Enter the corresponding number for the given operations:
1.ENQUEUE
2.DEQUEUE
3.DISPLAY Queue
4.EXIT Program
3

Current queue:
2
3

Enter the corresponding number for the given operations:
1.ENQUEUE
2.DEQUEUE
3.DISPLAY Queue
4.EXIT Program
1
Queue is full!

Enter the corresponding number for the given operations:
1.ENQUEUE
2.DEQUEUE
3.DISPLAY Queue
4.EXIT Program
2

Enter the corresponding number for the given operations:

1.ENQUEUE
2.DEQUEUE
3.DISPLAY Queue
4.EXIT Program
3

Current queue:
3

Enter the corresponding number for the given operations:
1.ENQUEUE
2.DEQUEUE
3.DISPLAY Queue
4.EXIT Program
2

Enter the corresponding number for the given operations:
1.ENQUEUE
2.DEQUEUE
3.DISPLAY Queue
4.EXIT Program
3

Current queue:

Enter the corresponding number for the given operations:
1.ENQUEUE
2.DEQUEUE
3.DISPLAY Queue
4.EXIT Program
2
Queue is empty!

Enter the corresponding number for the given operations:
1.ENQUEUE
2.DEQUEUE
3.DISPLAY Queue
4.EXIT Program
4

RESULT :- Seven different types of Queues are implemented using arrays. Insertion, Deletion and Display operations are performed on them.

EXPERIMENT 8a

AIM :- Write a menu driven program for performing the following operations on a Linked List:

1. Display
2. Insert at Beginning
3. Insert at End
4. Insert at a specified Position
5. Delete from Beginning
6. Delete from End
7. Delete from a specified Position

DATA STRUCTURE USED :- LINKED LIST is the data structure used.

ALGORITHM :-

```
Algorithm Display
START
    ptr = HEADER->LINK
    While ptr != NULL
        Print ptr->DATA
        ptr = ptr->LINK
    EndWhile
STOP

Algorithm InsertFront
START
    ptr = GetNode(NODE)
    ptr->DATA = X
    ptr->LINK = HEADER->LINK
    HEADER->LINK = ptr
STOP

Algorithm InsertEnd
START
    ptr = HEADER
    While ptr->LINK != NULL
        ptr = ptr->LINK
    EndWhile
    ptr->LINK = GetNode(NODE)
    ptr->LINK->DATA = X
    ptr->LINK->LINK = NULL
STOP

Algorithm InsertAny
START
    ptr = HEADER->LINK
```

```
If ptr == NULL)
    Print "List is empty!"
    Exit
Else
    While ptr->DATA != KEY
        If ptr->LINK = NULL
            Print "Key not found!"
            Exit
        Else
            ptr = ptr->LINK
        EndIf
    EndWhile
    If ptr->DATA = KEY
        ptr1 = GetNode(NODE)
        ptr1->DATA = X
        ptr1->LINK = ptr->LINK
        ptr->LINK = ptr1
    EndIf
EndIf
STOP
```

Algorithm DeleteFront

```
START
If HEADER->LINK = NULL
    Print "List is empty"
    Exit
Else
    ptr = HEADER->LINK
    X = ptr->DATA
    HEADER->LINK = ptr->LINK
    ReturnNode(ptr)
    Return X
EndIf
STOP
```

Algorithm DeleteEnd

```
START
ptr = HEADER
If ptr->LINK = NULL
    Print "List is empty"
    Exit
Else
    While ptr->LINK->LINK != NULL
        ptr = ptr->LINK
    EndWhile
    X = ptr->LINK->DATA
    ReturnNode(ptr->LINK)
    ptr->LINK = NULL
    Return X
EndIf
STOP
```

```
Algorithm DeleteAny
START
    ptr1 = HEADER
    ptr = ptr1->LINK
    If ptr = NULL
        Print "List is empty"
        Exit
    Else
        While ptr->DATA != KEY and ptr->LINK != NULL
            ptr1 = ptr
            ptr = ptr->LINK
        EndWhile
        If ptr->DATA = KEY
            ptr1->LINK = ptr->LINK
            ReturnNode(ptr)
        Else
            Print "Key not found"
        EndIf
    EndIf
STOP
```

PROGRAM CODE :-

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

struct node
{
	int DATA;
	struct node* LINK;
};

void display(struct node* ptr)
{
	printf("The list:\n");
	while(ptr!=NULL)
	{
		printf("%d\n", ptr->DATA);
		ptr = ptr->LINK;
	}
}

void insertFront(struct node* HEADER)
{
	struct node* ptr = (struct node*) malloc(1* sizeof(struct node*));

	int X;
	printf("Enter element\n");
	scanf("%d", &X);

	ptr->DATA = X;
	ptr->LINK = HEADER->LINK;
	HEADER->LINK = ptr;
}
```

```
void insertEnd(struct node* ptr)
{
	int X;
	printf("Enter element\n");
	scanf("%d", &X);

	while(ptr->LINK != NULL)
	{
		ptr = ptr->LINK;
	}

	ptr->LINK = (struct node*) malloc(1* sizeof(struct node*));
	ptr->LINK->DATA = X;
	ptr->LINK->LINK = NULL;
}

void insertAny(struct node* ptr)
{

	if(ptr == NULL)
	{
		printf("Cannot be inserted! (List is empty)\n");
	}
	else
	{
		int X, KEY;
		printf("Enter element\n");
		scanf("%d", &X);
		printf("Enter the KEY element\n");
		scanf("%d", &KEY);

		while(ptr->DATA != KEY)
		{
			if(ptr->LINK == NULL)
			{
				printf("Cannot be inserted at specified position! (KEY not found)\n");
				break;
			}
			else
				ptr = ptr->LINK;
		}

		if(ptr->DATA == KEY)
		{
			struct node* ptr1 = (struct node*) malloc(1*sizeof(struct node*));
			ptr1->DATA = X;
			ptr1->LINK = ptr->LINK;
			ptr->LINK = ptr1;
		}
	}
}

int deleteFront(struct node* HEADER)
{
	if(HEADER->LINK == NULL)
	{
		printf("Cannot be deleted! (List is empty)\n");
```

```
	}
	else
	{
		struct node* ptr = HEADER->LINK;

		int X = ptr->DATA;
		HEADER->LINK = ptr->LINK;
		free(ptr);

		return X;
	}
}

int deleteEnd(struct node* ptr)
{
	if(ptr->LINK == NULL)
	{
		printf("Cannot be deleted! (List is empty)\n");
	}
	else
	{
		while(ptr->LINK->LINK != NULL)
		{
			ptr = ptr->LINK;
		}

		int X = ptr->LINK->DATA;
		free(ptr->LINK);
		ptr->LINK = NULL;

		return X;
	}
}

void deleteAny(struct node* ptr1)
{
	struct node* ptr = ptr1->LINK;

	if(ptr == NULL)
	{
		printf("Cannot be deleted! (List is empty)\n");
	}
	else
	{
		int KEY;
		printf("Enter the KEY element\n");
		scanf("%d", &KEY);

		while(ptr->DATA != KEY && ptr->LINK != NULL)
		{
			ptr1 = ptr;
			ptr = ptr->LINK;
		}

		if(ptr->DATA == KEY)
		{
			ptr1->LINK = ptr->LINK;
```

```
			free(ptr);
		}
		else
		{
			printf("Cannot be deleted from specified position! (KEY not found)\n");
		}
	}
}

void main()
{
	struct node* HEADER = (struct node*) malloc(1*sizeof(struct node));
	HEADER->LINK = NULL;

	int flag;
L:
	printf("\nChoose the Linked List operation\n");
	printf("1.Display\n2.Insert at Beginning\n3.Insert at End\n4.Insert at a Specified Position\n");
	printf("5.Delete from Beginning\n6.Delete from End\n7.Delete from a Specified Position\n8.Exit
Program\n\n");
	scanf("%d", &flag);

	switch(flag)
	{
		case 1:
			display(HEADER->LINK);
			goto L;

		case 2:
			insertFront(HEADER);
			goto L;

		case 3:
			insertEnd(HEADER);
			goto L;

		case 4:
			insertAny(HEADER->LINK);
			goto L;

		case 5:
			deleteFront(HEADER);
			goto L;

		case 6:
			deleteEnd(HEADER);
			goto L;

		case 7:
			deleteAny(HEADER);
			goto L;

		case 8:
			exit(0);

		default:
			printf("Invalid input\n\n");
```

```
                    goto L;
        }
}
```

SAMPLE OUTPUT :-

Choose the Linked List operation
1.Display
2.Insert at Beginning
3.Insert at End
4.Insert at a Specified Position
5.Delete from Beginning
6.Delete from End
7.Delete from a Specified Position
8.Exit Program

1
The list:

Choose the Linked List operation
1.Display
2.Insert at Beginning
3.Insert at End
4.Insert at a Specified Position
5.Delete from Beginning
6.Delete from End
7.Delete from a Specified Position
8.Exit Program

2
Enter element
1

Choose the Linked List operation
1.Display
2.Insert at Beginning
3.Insert at End
4.Insert at a Specified Position
5.Delete from Beginning
6.Delete from End
7.Delete from a Specified Position
8.Exit Program

3
Enter element
3

Choose the Linked List operation
1.Display
2.Insert at Beginning
3.Insert at End

4.Insert at a Specified Position
5.Delete from Beginning
6.Delete from End
7.Delete from a Specified Position
8.Exit Program

4
Enter element
2
Enter the KEY element
1

Choose the Linked List operation
1.Display
2.Insert at Beginning
3.Insert at End
4.Insert at a Specified Position
5.Delete from Beginning
6.Delete from End
7.Delete from a Specified Position
8.Exit Program

1
The list:
1
2
3

Choose the Linked List operation
1.Display
2.Insert at Beginning
3.Insert at End
4.Insert at a Specified Position
5.Delete from Beginning
6.Delete from End
7.Delete from a Specified Position
8.Exit Program

6

Choose the Linked List operation
1.Display
2.Insert at Beginning
3.Insert at End
4.Insert at a Specified Position
5.Delete from Beginning
6.Delete from End
7.Delete from a Specified Position
8.Exit Program

1
The list:

1
2

Choose the Linked List operation
1.Display
2.Insert at Beginning
3.Insert at End
4.Insert at a Specified Position
5.Delete from Beginning
6.Delete from End
7.Delete from a Specified Position
8.Exit Program

5

Choose the Linked List operation
1.Display
2.Insert at Beginning
3.Insert at End
4.Insert at a Specified Position
5.Delete from Beginning
6.Delete from End
7.Delete from a Specified Position
8.Exit Program

1
The list:
2

Choose the Linked List operation
1.Display
2.Insert at Beginning
3.Insert at End
4.Insert at a Specified Position
5.Delete from Beginning
6.Delete from End
7.Delete from a Specified Position
8.Exit Program

7
Enter the KEY element
2

Choose the Linked List operation
1.Display
2.Insert at Beginning
3.Insert at End
4.Insert at a Specified Position
5.Delete from Beginning
6.Delete from End
7.Delete from a Specified Position
8.Exit Program

1
The list:

Choose the Linked List operation
1.Display
2.Insert at Beginning
3.Insert at End
4.Insert at a Specified Position
5.Delete from Beginning
6.Delete from End
7.Delete from a Specified Position
8.Exit Program

6
Cannot be deleted! (List is empty)

Choose the Linked List operation
1.Display
2.Insert at Beginning
3.Insert at End
4.Insert at a Specified Position
5.Delete from Beginning
6.Delete from End
7.Delete from a Specified Position
8.Exit Program

8

RESULT :- The given operations are performed on a Linked List.

EXPERIMENT 8b

AIM :- Implement a Stack using linked list with the operations:

1. Push elements to the stack

2. Pop elements from the stack

3. Display the stack after each operation

DATA STRUCTURE USED :- STACK is the data structure used.

ALGORITHM :-

```
Algorithm PUSH
START
    1   ptr = GetNode(NODE)
    2   ptr->DATA = X
    3   ptr->LINK = HEADER->LINK
    4   HEADER->LINK = ptr
    5   TOP = ptr
STOP

Algorithm POP
START
    1   If TOP = NULL
    2           Print "Stack is empty!"
    3           Exit
    4   Else
    5           X = TOP->DATA
    6           TOP = TOP->LINK
    7           ReturnNode(HEADER->LINK)
    8           HEADER->LINK = TOP
    9           Return X
    10  EndIf
STOP

Algorithm DISPLAY
START
    1   ptr = HEADER->LINK
    2   While ptr != NULL
    3           Print ptr->DATA
    4           ptr = ptr->LINK
    5   EndWhile
STOP
```

PROGRAM CODE :-

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

```
struct node
{
	int DATA;
	struct node* LINK;
};

void display(struct node* ptr)
{
	printf("Stack:\n");
	while(ptr != NULL)
	{
		printf("%d\n", ptr->DATA);
		ptr = ptr->LINK;
	}
}

struct node* push(struct node* HEADER)
{
	struct node* ptr = (struct node*) malloc(1*sizeof(struct node));

	int X;
	printf("Enter element\n");
	scanf("%d", &X);

	ptr->DATA = X;
	ptr->LINK = HEADER->LINK;
	HEADER->LINK = ptr;
	return ptr;
}

int pop(struct node* HEADER)
{
	if(HEADER->LINK == NULL)
	{
		printf("Stack is empty!\n");
	}
	else
	{
		struct node* ptr = HEADER->LINK;

		int X = ptr->DATA;
		HEADER->LINK = ptr->LINK;
		free(ptr);

		return X;
	}
}

void main()
{
	struct node* STACK_HEAD = (struct node*) malloc(1*sizeof(struct node));
	struct node* TOP = NULL;
```

```
	STACK_HEAD->LINK = TOP;

	int flag;
L:
	printf("\nChoose the Stack operation\n");
	printf("1.PUSH\n2.POP\n3.DISPLAY\n4.EXIT\n\n");
	scanf("%d", &flag);

	switch(flag)
	{
		case 1:
			TOP = push(STACK_HEAD);
			goto L;
		case 2:
			pop(STACK_HEAD);
			TOP = STACK_HEAD->LINK;
			goto L;
		case 3:
			display(TOP);
			goto L;
		case 4:
			exit(0);
		default:
			printf("Invalid input!\n");
			goto L;
	}
}
```

SAMPLE OUTPUT :-

```
Choose the Stack operation
1.PUSH
2.POP
3.DISPLAY
4.EXIT

1
Enter element
5

Choose the Stack operation
1.PUSH
2.POP
3.DISPLAY
4.EXIT

3
Stack:
5

Choose the Stack operation
1.PUSH
2.POP
```

3.DISPLAY
4.EXIT

1
Enter element
10

Choose the Stack operation
1.PUSH
2.POP
3.DISPLAY
4.EXIT

1
Enter element
15

Choose the Stack operation
1.PUSH
2.POP
3.DISPLAY
4.EXIT

1
Enter element
20

Choose the Stack operation
1.PUSH
2.POP
3.DISPLAY
4.EXIT

1
Enter element
25

Choose the Stack operation
1.PUSH
2.POP
3.DISPLAY
4.EXIT

3
Stack:
25
20
15
10
5

Choose the Stack operation
1.PUSH
2.POP
3.DISPLAY
4.EXIT

2

Choose the Stack operation
1.PUSH
2.POP
3.DISPLAY
4.EXIT

3
Stack:
20
15
10
5

Choose the Stack operation
1.PUSH
2.POP
3.DISPLAY
4.EXIT

2

Choose the Stack operation
1.PUSH
2.POP
3.DISPLAY
4.EXIT

3
Stack:
15
10
5

Choose the Stack operation
1.PUSH
2.POP
3.DISPLAY
4.EXIT

2

Choose the Stack operation
1.PUSH
2.POP
3.DISPLAY
4.EXIT

3
Stack:
10
5

Choose the Stack operation
1.PUSH
2.POP

3.DISPLAY
4.EXIT

2

Choose the Stack operation
1.PUSH
2.POP
3.DISPLAY
4.EXIT

3
Stack:
5

Choose the Stack operation
1.PUSH
2.POP
3.DISPLAY
4.EXIT

2

Choose the Stack operation
1.PUSH
2.POP
3.DISPLAY
4.EXIT

3
Stack:

Choose the Stack operation
1.PUSH
2.POP
3.DISPLAY
4.EXIT

2
Stack is empty!

Choose the Stack operation
1.PUSH
2.POP
3.DISPLAY
4.EXIT

4

RESULT :- The given operations are performed on a Stack implemented using linked list.

EXPERIMENT 8c

AIM :- Implement a Queue using linked list with the operations:

1. Insert elements to the queue

2. Delete elements from the queue

3. Display the queue after each operation

DATA STRUCTURE USED :- QUEUE is the data structure used.

ALGORITHM :-

```
Algorithm ENQUEUE
START
    If REAR = NULL
        FRONT = GetNode(NODE)
        QUEUE_HEAD->LINK = FRONT
        REAR = FRONT
        REAR->DATA = X
        REAR->LINK = NULL
    Else
        REAR->LINK = GetNode(NODE)
        REAR->LINK->DATA = X
        REAR->LINK->LINK = NULL
        REAR = REAR->LINK
    EndIf
STOP
```

```
Algorithm DEQUEUE
START
    If FRONT = NULL
        Print "Queue is empty!"
        Exit
    Else
        X = FRONT->DATA
        QUEUE_HEAD->LINK = FRONT->LINK
        ReturnNode(FRONT)
        FRONT = QUEUE_HEAD->LINK
        If FRONT = NULL
            REAR = NULL
        EndIf
        Return X
    EndIf
STOP
```

```
Algorithm DISPLAY
START
    ptr = FRONT
    While ptr != NULL
```

3 Print ptr->DATA
4 ptr = ptr->LINK
5 EndWhile
STOP

PROGRAM CODE :-

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

struct node
{
	int DATA;
	struct node* LINK;
};

struct queue
{
	struct node* FRONT;
	struct node* REAR;
};

void display(struct node* ptr)
{
	printf("Queue:\n");
	while(ptr != NULL)
	{
		printf("%d\n", ptr->DATA);
		ptr = ptr->LINK;
	}
}

void enqueue(struct queue *q, struct node* HEADER)
{
	int X;
	printf("Enter element\n");
	scanf("%d", &X);

	if(q->REAR == NULL)
	{
		q->FRONT = (struct node*) malloc(1*sizeof(struct node));
		HEADER->LINK = q->FRONT;
		q->REAR = q->FRONT;
		q->REAR->DATA = X;
		q->REAR->LINK = NULL;
	}
	else
	{
		q->REAR->LINK = (struct node*) malloc(1*sizeof(struct node));
		q->REAR->LINK->DATA = X;
		q->REAR->LINK->LINK = NULL;
		q->REAR = q->REAR->LINK;
```

```
	}
}

int dequeue(struct queue *q, struct node* HEADER)
{
	if(q->FRONT == NULL)
	{
		printf("Queue is empty!\n");
	}
	else
	{
		int X = q->FRONT->DATA;
		HEADER->LINK = q->FRONT->LINK;

		free(q->FRONT);

		q->FRONT = HEADER->LINK;

		if(q->FRONT == NULL)
			q->REAR = NULL;

		return X;
	}
}

void main()
{
	struct node* QUEUE_HEAD = (struct node*) malloc(1*sizeof(struct node));
	struct queue q;
	q.FRONT = NULL;
	q.REAR = NULL;
	QUEUE_HEAD->LINK = q.FRONT;

	int flag;
L:
	printf("\nChoose the Queue operation\n");
	printf("1.ENQUEUE\n2.DEQUEUE\n3.DISPLAY\n4.EXIT\n\n");
	scanf("%d", &flag);

	switch(flag)
	{
		case 1:
			enqueue(&q, QUEUE_HEAD);
			goto L;
		case 2:
			dequeue(&q, QUEUE_HEAD);
			goto L;
		case 3:
			display(q.FRONT);
			goto L;
		case 4:
			exit(0);
```

```
            default:
                printf("Invalid input!\n");
                goto L;
    }
}
```

SAMPLE OUTPUT :-

```
Choose the Queue operation
1.ENQUEUE
2.DEQUEUE
3.DISPLAY
4.EXIT

1
Enter element
1

Choose the Queue operation
1.ENQUEUE
2.DEQUEUE
3.DISPLAY
4.EXIT

1
Enter element
2

Choose the Queue operation
1.ENQUEUE
2.DEQUEUE
3.DISPLAY
4.EXIT

1
Enter element
3

Choose the Queue operation
1.ENQUEUE
2.DEQUEUE
3.DISPLAY
4.EXIT

3
Queue:
1
2
3

Choose the Queue operation
```

1.ENQUEUE
2.DEQUEUE
3.DISPLAY
4.EXIT

2

Choose the Queue operation
1.ENQUEUE
2.DEQUEUE
3.DISPLAY
4.EXIT

2

Choose the Queue operation
1.ENQUEUE
2.DEQUEUE
3.DISPLAY
4.EXIT

2

Choose the Queue operation
1.ENQUEUE
2.DEQUEUE
3.DISPLAY
4.EXIT

3
Queue:

Choose the Queue operation
1.ENQUEUE
2.DEQUEUE
3.DISPLAY
4.EXIT

2
Queue is empty!

Choose the Queue operation
1.ENQUEUE
2.DEQUEUE
3.DISPLAY
4.EXIT

4

RESULT :- The given operations are performed on a Queue implemented using linked list.

EXPERIMENT 9a

AIM :- Create a Doubly Linked List from a string taking each character from the string. Check if the given string is palindrome in an efficient method.

DATA STRUCTURE USED :- DOUBLY LINKED LIST is the data structure used.

ALGORITHM :-

```
Algorithm Display
START
    ptr = HEADER->RLINK
    While ptr != NULL
        Print ptr->DATA
        ptr = ptr->RLINK
    EndWhile
STOP

Algorithm InsertList
START
    ptr = HEADER
    While ptr->RLINK != NULL
        ptr = ptr->RLINK
    EndWhile
    ptr->RLINK = GetNode(NODE)
    ptr->RLINK->DATA = str[i]
    ptr->RLINK->RLINK = NULL
    ptr->RLINK->LLINK = ptr
    HEADER->LLINK = ptr->RLINK
STOP

Algorithm  CheckPalindrome
START
    ptr1 = HEADER->LLINK;
    ptr = HEADER->RLINK;
    While ptr != ptr1
        If ptr->DATA != ptr1->DATA
            Print "Not Palindrome"
            Exit
        Else if ptr->RLINK = ptr1
            Print "Palindrome"
            Exit
        EndIf
        ptr = ptr->RLINK
        ptr1 = ptr1->LLINK
    EndWhile
    Print "Palindrome"
STOP
```

PROGRAM CODE :-

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

struct node
{
	char DATA;
	struct node* LLINK;
	struct node* RLINK;
};

void display(struct node* ptr)
{
	while(ptr!=NULL)
	{
		printf("%c", ptr->DATA);
		ptr = ptr->RLINK;
	}
}

struct node* insertList(struct node* ptr, char X)
{
	while(ptr->RLINK != NULL)
	{
		ptr = ptr->RLINK;
	}

	ptr->RLINK = (struct node*) malloc(1*sizeof(struct node));

	ptr->RLINK->DATA = X;
	ptr->RLINK->RLINK = NULL;
	ptr->RLINK->LLINK = ptr;

	return ptr->RLINK;
}

int checkPalindrome(struct node* ptr)
{
	struct node* ptr1 = ptr->LLINK;
	ptr = ptr->RLINK;

	while(ptr != ptr1)
	{
		if(tolower(ptr->DATA) != tolower(ptr1->DATA))
			return 0;
		else if(ptr->RLINK == ptr1)
			return 1;
		ptr = ptr->RLINK;
		ptr1 = ptr1->LLINK;
	}

	return 1;
}
```

```
void main()
{
	struct node* HEADER = (struct node*) malloc(1*sizeof(struct node));
	HEADER->RLINK = NULL;
	HEADER->LLINK = NULL;

	char str[20];

	printf("Enter a string\n");
	fgets(str, 20, stdin);

	for(int i=0; i<strlen(str)-1; i++)
		HEADER->LLINK = insertList(HEADER, str[i]);

	display(HEADER->RLINK);

	if(checkPalindrome(HEADER))
		printf(" is a Palindrome\n");
	else
		printf(" is not a Palindrome\n");
}
```

SAMPLE OUTPUTS :-

Enter a string
amma
amma is a Palindrome

Enter a string
amal
amal is not a Palindrome

Enter a string
malayalam
malayalam is a Palindrome

Enter a string
a
a is a Palindrome

Enter a string
haa aah
haa aah is a Palindrome

Enter a string
ama amal
ama amal is not a Palindrome

Enter a string
Malayalam
Malayalam is a Palindrome

RESULT :- The given string was checked for palindromity using Doubly Linked List.

EXPERIMENT 9b

AIM :- The details of students (number, name, total-mark) are to be stored in a linked list. Write functions for the following operations:

1. Insert

2. Delete

3. Search

4. Sort on the basis of number

5. Display the resultant list after every operation

DATA STRUCTURE USED :- LINKED LIST is the data structure used.

ALGORITHM :-

```
Algorithm InsertList
START
    ptr = HEADER->LINK
    HEADER->LINK = GetNode(NODE)
    Read HEADER->LINK->NAME
    Read HEADER->LINK->NUM
    Read HEADER->LINK->MARK
    HEADER->LINK->LINK = ptr
STOP
```

```
Algorithm DeleteList
START
    ptr = HEADER
    If ptr->LINK = NULL
        Print "List is empty!\n");
        Exit
    EndIf
    Read KEY
    While ptr->LINK != NULL
        If ptr->LINK->NUM = KEY
            ptr1 = ptr->LINK->LINK
            ReturnNode(ptr->LINK)
            ptr->LINK = ptr1
            Exit
        Else
            ptr = ptr->LINK
        EndIf
    EndWhile
    If ptr->LINK = NULL
        Print "Key not found!"
    EndIf
STOP
```

Algorithm SearchList

```
START
    ptr = HEADER
    If ptr->LINK = NULL
        Print "List is empty!\n");
        Exit
    EndIf
    Read KEY
    While ptr->LINK != NULL
        If ptr->LINK->NUM = KEY
            Print "Key found!”
            Print ptr->LINK->NAME
            Print ptr->LINK->NUM
            Print ptr->LINK->MARK
            Exit
        Else
            ptr = ptr->LINK
        EndIf
    EndWhile
    If ptr->LINK = NULL
        Print "Key not found!"
    EndIf
STOP
```

Algorithm SortList

```
START
    ptr = HEADER->LINK
    If ptr = NULL
        Print "List is empty!"
        Exit
    Else if(ptr->LINK == NULL)
        Print "List is sorted successfully!"
        Exit
    EndIf
    While ptr->LINK != NULL
        min = ptr->NUM
        ptr1 = ptr->LINK
        minNode = ptr
        While ptr1 != NULL
            If ptr1->NUM < min
                min = ptr1->NUM
                minNode = ptr1
            EndIf
            ptr1 = ptr1->LINK
        EndWhile
        Swap(minNode->NUM, ptr->NUM)
        Swap(minNode->NAME, ptr->NAME)
        Swap(minNode->MARK, ptr->MARK)
        ptr = ptr->LINK
    EndWhile
    Print "List is sorted successfully!"
STOP
```

Algorithm Display

```
START
    ptr = HEADER->LINK
    If ptr = NULL
        Print "List is empty!"
        Exit
    EndIf
    While ptr!=NULL
        Print ptr->NAME
        Print ptr->NUM
        Print ptr->MARK
        ptr = ptr->LINK
    EndWhile
STOP
```

PROGRAM CODE :-

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

struct node
{
    int NUM;
    char* NAME;
    float MARK;
    struct node* LINK;
};

void insertList(struct node* HEADER)
{
    struct node* ptr = HEADER->LINK;

    HEADER->LINK = malloc(1*sizeof(struct node));

    printf("\nEnter student name\n");
    scanf("\n");
    HEADER->LINK->NAME = malloc(20*sizeof(char));
    fgets(HEADER->LINK->NAME, 20, stdin);
    HEADER->LINK->NAME[strlen(HEADER->LINK->NAME)-1] = '\0';

    printf("Enter student roll no.\n");
    scanf("%d", &HEADER->LINK->NUM);

    printf("Enter student total-marks\n");
    scanf("%f", &HEADER->LINK->MARK);

    HEADER->LINK->LINK = ptr;
}

int deleteList(struct node* ptr)
```

```
{
	if(ptr->LINK == NULL)
	{
		printf("\nList is empty!\n");
		return 0;
	}

	int KEY;

	printf("\nEnter the student roll no. to be deleted\n");
	scanf("%d", &KEY);

	while(ptr->LINK != NULL)
	{
		if(ptr->LINK->NUM == KEY)
		{
			struct node* ptr1 = ptr->LINK->LINK;
			printf("\n%s's details has been deleted\n", ptr->LINK->NAME);
			free(ptr->LINK);
			ptr->LINK = ptr1;
			return 1;
		}
		else
			ptr = ptr->LINK;
	}

	if(ptr->LINK == NULL)
		printf("\nStudent not found!\n");
}

int searchList(struct node* ptr)
{
	if(ptr->LINK == NULL)
	{
		printf("\nList is empty!\n");
		return 0;
	}

	int KEY;

	printf("\nEnter the student roll no. to be searched\n");
	scanf("%d", &KEY);

	while(ptr->LINK != NULL)
	{
		if(ptr->LINK->NUM == KEY)
		{
			printf("\nStudent found!\nName = %s\n", ptr->LINK->NAME);
			printf("Roll No. = %d\n",ptr->LINK->NUM);
			printf("Total Marks = %0.2f\n",ptr->LINK->MARK);
			return 1;
		}
```

```
		else
			ptr = ptr->LINK;
	}

	if(ptr->LINK == NULL)
		printf("\nStudent not found!\n");
}

int sortList(struct node* ptr)
{
	if(ptr == NULL)
	{
		printf("\nList is empty!\n");
		return 0;
	}
	else if(ptr->LINK == NULL)
	{
		printf("\nList is sorted successfully!\n");
		return 1;
	}

	while(ptr->LINK != NULL)
	{
		int min = ptr->NUM;
		struct node* ptr1 = ptr->LINK;
		struct node* minNode = ptr;

		while(ptr1 != NULL)
		{
			if(ptr1->NUM < min)
			{
				min = ptr1->NUM;
				minNode = ptr1;
			}
			ptr1 = ptr1->LINK;
		}

		int num = minNode->NUM;
		minNode->NUM = ptr->NUM;
		ptr->NUM = num;

		char* name = minNode->NAME;
		minNode->NAME = ptr->NAME;
		ptr->NAME = name;

		float mark = minNode->MARK;
		minNode->MARK = ptr->MARK;
		ptr->MARK = mark;

		ptr = ptr->LINK;
	}
	printf("\nList is sorted successfully!\n");
```

```
}

int display(struct node* ptr)
{
	if(ptr == NULL)
	{
		printf("\nList is empty!\n");
		return 0;
	}

	int i=1;

	while(ptr!=NULL)
	{
		printf("\nStudent %d Details:\nName = %s\n", i, ptr->NAME);
		printf("Roll No. = %d\n",ptr->NUM);
		printf("Total Marks = %0.2f\n",ptr->MARK);
		ptr=ptr->LINK;
		i++;
	}
}

void main()
{
	struct node* HEADER = (struct node*) malloc(1*sizeof(struct node));
	HEADER->LINK = NULL;

	int flag;
L:
	printf("\nChoose the option:\n");
	printf("1.INSERT\n2.DELETE\n3.SEARCH\n4.SORT\n5.DISPLAY\n6.EXIT\n");
	scanf("%d", &flag);

	switch(flag)
	{
		case 1:
			insertList(HEADER);
			goto L;
		case 2:
			deleteList(HEADER);
			goto L;
		case 3:
			searchList(HEADER);
			goto L;
		case 4:
			sortList(HEADER->LINK);
			goto L;
		case 5:
			display(HEADER->LINK);
			goto L;
		case 6:
			printf("\nEXIT.\n");
```

```
				exit(0);
			default:
				printf("INVALID INPUT!\n");
				goto L;
		}
}
```

SAMPLE OUTPUT :-

```
Choose the option:
1.INSERT
2.DELETE
3.SEARCH
4.SORT
5.DISPLAY
6.EXIT
1

Enter student name
Amal Nath
Enter student roll no.
11
Enter student total-marks
100

Choose the option:
1.INSERT
2.DELETE
3.SEARCH
4.SORT
5.DISPLAY
6.EXIT
1

Enter student name
Ajith Kumar
Enter student roll no.
7
Enter student total-marks
99.75

Choose the option:
1.INSERT
2.DELETE
3.SEARCH
4.SORT
5.DISPLAY
6.EXIT
1

Enter student name
```

Jijo Johnson
Enter student roll no.
31
Enter student total-marks
99.5

Choose the option:
1.INSERT
2.DELETE
3.SEARCH
4.SORT
5.DISPLAY
6.EXIT
1

Enter student name
Emil Joji
Enter student roll no.
22
Enter student total-marks
99.5

Choose the option:
1.INSERT
2.DELETE
3.SEARCH
4.SORT
5.DISPLAY
6.EXIT
1

Enter student name
Dinoy Raj
Enter student roll no.
21
Enter student total-marks
99.75

Choose the option:
1.INSERT
2.DELETE
3.SEARCH
4.SORT
5.DISPLAY
6.EXIT
3

Enter the student roll no. to be searched
11

Student found!
Name = Amal Nath

Roll No. = 11
Total Marks = 100.00

Choose the option:
1.INSERT
2.DELETE
3.SEARCH
4.SORT
5.DISPLAY
6.EXIT
2

Enter the student roll no. to be deleted
11

Amal Nath's details has been deleted

Choose the option:
1.INSERT
2.DELETE
3.SEARCH
4.SORT
5.DISPLAY
6.EXIT
5

Student 1 Details:
Name = Dinoy Raj
Roll No. = 21
Total Marks = 99.75

Student 2 Details:
Name = Emil Joji
Roll No. = 22
Total Marks = 99.50

Student 3 Details:
Name = Jijo Johnson
Roll No. = 31
Total Marks = 99.50

Student 4 Details:
Name = Ajith Kumar
Roll No. = 7
Total Marks = 99.75

Choose the option:
1.INSERT
2.DELETE
3.SEARCH
4.SORT
5.DISPLAY

6.EXIT
4

List is sorted successfully!

Choose the option:
1.INSERT
2.DELETE
3.SEARCH
4.SORT
5.DISPLAY
6.EXIT
5

Student 1 Details:
Name = Ajith Kumar
Roll No. = 7
Total Marks = 99.75

Student 2 Details:
Name = Dinoy Raj
Roll No. = 21
Total Marks = 99.75

Student 3 Details:
Name = Emil Joji
Roll No. = 22
Total Marks = 99.50

Student 4 Details:
Name = Jijo Johnson
Roll No. = 31
Total Marks = 99.50

Choose the option:
1.INSERT
2.DELETE
3.SEARCH
4.SORT
5.DISPLAY
6.EXIT
6

EXIT.

RESULT :- The given operations are performed on a Student linked list.

EXPERIMENT 9c

AIM :- Write a program to read two polynomials and store them using linked list. Calculate the sum and product and display the first polynomial, second polynomial and the resultant polynomial.

DATA STRUCTURE USED :- LINKED LIST is the data structure used.

ALGORITHM :-

Algorithm InsertList

```
START
    ptr = HEADER->LINK
    HEADER->LINK = GetNode(NODE)
    Read HEADER->LINK->COEFF
    Read HEADER->LINK->EXPO
    HEADER->LINK->LINK = ptr
STOP
```

Algorithm AddList

```
START
    ptr1 = HEADER1->LINK, ptr2 = HEADER2->LINK, ptr3 = HEADER3
    While ptr1 != NULL and ptr2 != NULL do
        If ptr1->EXPO = ptr2->EXPO
            If ptr1->COEFF + ptr2->COEFF != 0
                ptr3->LINK = GetNode(NODE)
                ptr3->LINK->EXPO = ptr1->EXPO
                ptr3->LINK->COEFF = ptr1->COEFF + ptr2->COEFF
                ptr3->LINK->LINK = NULL
                ptr3 = ptr3->LINK
            Endif
            ptr1 = ptr1->LINK
            ptr2 = ptr2->LINK
        Else if ptr1->EXPO < ptr2->EXPO
            ptr3->LINK = GetNode(NODE)
            ptr3->LINK->EXPO = ptr1->EXPO
            ptr3->LINK->COEFF = ptr1->COEFF
            ptr3->LINK->LINK = NULL
            ptr1 = ptr1->LINK
            ptr3 = ptr3->LINK
        Else
            ptr3->LINK = GetNode(NODE)
            ptr3->LINK->EXPO = ptr2->EXPO
            ptr3->LINK->COEFF = ptr2->COEFF
            ptr3->LINK->LINK = NULL
            ptr2 = ptr2->LINK
            ptr3 = ptr3->LINK
        Endif
    EndWhile
    If ptr1 != NULL and ptr2 = NULL do
        While ptr1 != NULL
```

```
                ptr3->LINK = GetNode(NODE)
                ptr3->LINK->EXPO = ptr1->EXPO
                ptr3->LINK->COEFF = ptr1->COEFF
                ptr3->LINK->LINK = NULL
                ptr1 = ptr1->LINK
                ptr3 = ptr3->LINK
        EndWhile
Else if ptr1 = NULL and ptr2 != NULL do
        While ptr2 != NULL
                ptr3->LINK = GetNode(NODE)
                ptr3->LINK->EXPO = ptr2->EXPO
                ptr3->LINK->COEFF = ptr2->COEFF
                ptr3->LINK->LINK = NULL
                ptr2 = ptr2->LINK
                ptr3 = ptr3->LINK
        EndWhile
EndIf
STOP
```

```
Algorithm ProductList
START
ptr1 = HEADER1->LINK
While ptr1 != NULL
        ptr2 = HEADER2->LINK
        While ptr2 != NULL
                C = ptr1->COEFF * ptr2->COEFF
                E = ptr1->EXPO + ptr2->EXPO
                ptr = GetNode(NODE)
                ptr3 = HEADER4
                While ptr3->LINK != NULL
                        If ptr3->LINK->EXPO = E
                                ptr3->LINK->COEFF += C
                                Exit while
                        Else if ptr3->LINK->EXPO > E
                                ptr->EXPO = E
                                ptr->COEFF = C
                                ptr->LINK = ptr3->LINK
                                ptr3->LINK = ptr
                                Exit while
                        EndIf
                        ptr3 = ptr3->LINK
                EndWhile
                If ptr3->LINK = NULL
                        ptr3->LINK = ptr
                        ptr->COEFF = C
                        ptr->EXPO = E
                        ptr->LINK = NULL
                EndIf
                ptr2 = ptr2->LINK
        EndWhile
        ptr1 = ptr1->LINK
EndWhile
```

```
STOP

Algorithm SortList
START
    ptr = HEADER->LINK
    If ptr = NULL
            Exit
    Else if ptr->LINK = NULL
            Exit
    EndIf
    While ptr->LINK != NULL
            min = ptr->EXPO
            ptr1 = ptr->LINK
            minNode = ptr
            While ptr1 != NULL
                    If ptr1->EXPO < min
                            min = ptr1->EXPO
                            minNode = ptr1
                    EndIf
                    ptr1 = ptr1->LINK
            EndWhile
            Swap(minNode->EXPO, ptr->EXPO)
            Swap(minNode->COEFF, ptr->COEFF)
            ptr = ptr->LINK
    EndWhile
STOP

Algorithm Display
START
    ptr = HEADER->LINK
    If ptr = NULL)
            Print "0"
            Exit
    EndIf
    While ptr != NULL
            Print ptr->COEFF, "x^", ptr->EXPO
            ptr = ptr->LINK
    EndWhile
STOP
```

PROGRAM CODE :-

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

struct node
{
        int COEFF;
        int EXPO;
        struct node* LINK;
};
```

```
void insertList(struct node* HEADER, int coeff, int expo)
{
	struct node* ptr = HEADER->LINK;

	HEADER->LINK = malloc(1*sizeof(struct node));

	HEADER->LINK->COEFF = coeff;
	HEADER->LINK->EXPO = expo;
	HEADER->LINK->LINK = ptr;
}

void display(struct node* ptr)
{
	if(ptr == NULL)
	{
		printf("0 ");
	}
	while(ptr != NULL)
	{
		printf("%dx^%d ", ptr->COEFF, ptr->EXPO);
		ptr = ptr->LINK;
	}
	printf(")");
}

void addList(struct node* ptr1, struct node* ptr2, struct node* ptr3)
{
	while(ptr1 != NULL && ptr2 != NULL)
	{
		if(ptr1->EXPO == ptr2->EXPO)
		{
			if(ptr1->COEFF + ptr2->COEFF != 0)
			{
				ptr3->LINK = malloc(1*sizeof(struct node));
				ptr3->LINK->EXPO = ptr1->EXPO;
				ptr3->LINK->COEFF = ptr1->COEFF + ptr2->COEFF;
				ptr3->LINK->LINK = NULL;
				ptr3 = ptr3->LINK;
			}
			ptr1 = ptr1->LINK;
			ptr2 = ptr2->LINK;
		}
		else if(ptr1->EXPO < ptr2->EXPO)
		{
			ptr3->LINK = malloc(1*sizeof(struct node));
			ptr3->LINK->EXPO = ptr1->EXPO;
			ptr3->LINK->COEFF = ptr1->COEFF;
			ptr3->LINK->LINK = NULL;
			ptr1 = ptr1->LINK;
			ptr3 = ptr3->LINK;
		}
		else
		{
			ptr3->LINK = malloc(1*sizeof(struct node));
```

```
                ptr3->LINK->EXPO = ptr2->EXPO;
                ptr3->LINK->COEFF = ptr2->COEFF;
                ptr3->LINK->LINK = NULL;
                ptr2 = ptr2->LINK;
                ptr3 = ptr3->LINK;
            }
        }
        if(ptr1 != NULL && ptr2 == NULL)
        {
            while(ptr1 != NULL)
            {
                ptr3->LINK = malloc(1*sizeof(struct node));
                ptr3->LINK->EXPO = ptr1->EXPO;
                ptr3->LINK->COEFF = ptr1->COEFF;
                ptr3->LINK->LINK = NULL;
                ptr1 = ptr1->LINK;
                ptr3 = ptr3->LINK;
            }
        }
        else if(ptr1 == NULL && ptr2 != NULL)
        {
            while(ptr2 != NULL)
            {
                ptr3->LINK = malloc(1*sizeof(struct node));
                ptr3->LINK->EXPO = ptr2->EXPO;
                ptr3->LINK->COEFF = ptr2->COEFF;
                ptr3->LINK->LINK = NULL;
                ptr2 = ptr2->LINK;
                ptr3 = ptr3->LINK;
            }
        }
}

void productList(struct node* ptr1, struct node* HEADER2, struct node* HEADER4)
{
        while(ptr1 != NULL)
        {
            struct node* ptr2 = HEADER2;

            while(ptr2 != NULL)
            {
                int C = ptr1->COEFF * ptr2->COEFF;
                int E = ptr1->EXPO + ptr2->EXPO;

                struct node* ptr = malloc(1*sizeof(struct node*));
                struct node* ptr3 = HEADER4;

                while(ptr3->LINK != NULL)
                {
                    if(ptr3->LINK->EXPO == E)
                    {
                        ptr3->LINK->COEFF += C;
                        break;
                    }
```

```
                        else if(ptr3->LINK->EXPO > E)
                        {
                                ptr->EXPO = E;
                                ptr->COEFF = C;
                                ptr->LINK = ptr3->LINK;
                                ptr3->LINK = ptr;
                                break;
                        }
                        ptr3 = ptr3->LINK;
                }

                if(ptr3->LINK == NULL)
                {
                        ptr3->LINK = ptr;
                        ptr->COEFF = C;
                        ptr->EXPO = E;
                        ptr->LINK = NULL;
                }

                ptr2 = ptr2->LINK;
            }

            ptr1 = ptr1->LINK;
        }
}

int sortList(struct node* ptr)
{
        if(ptr == NULL)
        {
                return 0;
        }
        else if(ptr->LINK == NULL)
        {
                return 1;
        }

        while(ptr->LINK != NULL)
        {
                int min = ptr->EXPO;
                struct node* ptr1 = ptr->LINK;
                struct node* minNode = ptr;

                while(ptr1 != NULL)
                {
                        if(ptr1->EXPO < min)
                        {
                                min = ptr1->EXPO;
                                minNode = ptr1;
                        }
                        ptr1 = ptr1->LINK;
                }

                int temp = minNode->EXPO;
```

```
        minNode->EXPO = ptr->EXPO;
        ptr->EXPO = temp;

        temp = minNode->COEFF;
        minNode->COEFF = ptr->COEFF;
        ptr->COEFF = temp;

        ptr = ptr->LINK;
    }
}

void main()
{
    int x, coeff, expo;
    struct node* HEADER1 = malloc(1*sizeof(struct node));
    struct node* HEADER2 = malloc(1*sizeof(struct node));
    struct node* HEADER3 = malloc(1*sizeof(struct node));
    struct node* HEADER4 = malloc(1*sizeof(struct node));
    HEADER1->LINK = NULL;
    HEADER2->LINK = NULL;
    HEADER3->LINK = NULL;
    HEADER4->LINK = NULL;

    printf("Enter the no. of terms in the first polynomial\n");
    scanf("%d", &x);
    printf("\nEnter the coefficient and then exponent of each terms respectively in the first
polynomial\n");

    for(int i=0; i<x; i++)
    {
        scanf("%d", &coeff);
        scanf("%d", &expo);
        if(coeff != 0)
            insertList(HEADER1, coeff, expo);
    }

    printf("\nEnter the no. of terms in the second polynomial\n");
    scanf("%d", &x);
    printf("\nEnter the coefficient and then exponent of each terms respectively in the second
polynomial\n");

    for(int i=0; i<x; i++)
    {
        scanf("%d", &coeff);
        scanf("%d", &expo);
        if(coeff != 0)
            insertList(HEADER2, coeff, expo);
    }

    sortList(HEADER1->LINK);
    sortList(HEADER2->LINK);

    addList(HEADER1->LINK, HEADER2->LINK, HEADER3);
    productList(HEADER1->LINK, HEADER2->LINK, HEADER4);
```

```
        printf("\nSUM :\n( ");
        display(HEADER1->LINK);
        printf(" +\n( ");
        display(HEADER2->LINK);
        printf(" =\n( ");
        display(HEADER3->LINK);
        printf("\n");

        printf("\nPRODUCT :\n( ");
        display(HEADER1->LINK);
        printf(" *\n( ");
        display(HEADER2->LINK);
        printf(" =\n( ");
        display(HEADER4->LINK);
        printf("\n");
}
```

SAMPLE OUTPUTS :-

1.)

```
Enter the no. of terms in the first polynomial
3

Enter the coefficient and then exponent of each terms respectively in the first polynomial
4
3
3
2
2
1

Enter the no. of terms in the second polynomial
3

Enter the coefficient and then exponent of each terms respectively in the second polynomial
1
1
2
2
3
3

SUM :
( 2x^1 3x^2 4x^3 ) +
( 1x^1 2x^2 3x^3 ) =
( 3x^1 5x^2 7x^3 )

PRODUCT :
( 2x^1 3x^2 4x^3 ) *
( 1x^1 2x^2 3x^3 ) =
( 2x^2 7x^3 16x^4 17x^5 12x^6 )
```

2.)
Enter the no. of terms in the first polynomial
4

Enter the coefficient and then exponent of each terms respectively in the first polynomial
4
2
5
3
2
1
6
4

Enter the no. of terms in the second polynomial
4

Enter the coefficient and then exponent of each terms respectively in the second polynomial
1
2
3
4
5
6
7
8

SUM :
(2x^1 4x^2 5x^3 6x^4) +
(1x^2 3x^4 5x^6 7x^8) =
(2x^1 5x^2 5x^3 9x^4 5x^6 7x^8)

PRODUCT :
(2x^1 4x^2 5x^3 6x^4) *
(1x^2 3x^4 5x^6 7x^8) =
(2x^3 4x^4 11x^5 18x^6 25x^7 38x^8 39x^9 58x^10 35x^11 42x^12)

3.)
Enter the no. of terms in the first polynomial
3

Enter the coefficient and then exponent of each terms respectively in the first polynomial
0
1
2
3
4
5

Enter the no. of terms in the second polynomial

2

Enter the coefficient and then exponent of each terms respectively in the second polynomial
10
2
20
1

SUM :
(2x^3 4x^5) +
(20x^1 10x^2) =
(20x^1 10x^2 2x^3 4x^5)

PRODUCT :
(2x^3 4x^5) *
(20x^1 10x^2) =
(40x^4 20x^5 80x^6 40x^7)

4.)
Enter the no. of terms in the first polynomial
1

Enter the coefficient and then exponent of each terms respectively in the first polynomial
0
2

Enter the no. of terms in the second polynomial
2

Enter the coefficient and then exponent of each terms respectively in the second polynomial
1
2
3
4

SUM :
(0) +
(1x^2 3x^4) =
(1x^2 3x^4)

PRODUCT :
(0) *
(1x^2 3x^4) =
(0)

RESULT :- Two polynomials are stored using linked list. Both are displayed along with their sum and product.

EXPERIMENT 10a

AIM :- Create a binary tree with the following operations

1. Insert a new node

2. Inorder traversal

3. Preorder traversal

4. Postorder traversal

5. Delete a node

DATA STRUCTURE USED :- TREE using LINKED LIST is the data structure used.

ALGORITHM :-

```
Algorithm BuildTree(ptr)
START
    If ptr != NULL          //Initially ptr = ROOT
        Read ptr->DATA
        Read option if ptr->DATA has a left child
        If opion = yes
            ptr->LC = GetNode(NODE)
            buildTree(ptr->LC)
        Else
            ptr->LC = NULL
        Endif
        Read option if ptr->DATA has a right child
        If option = yes
            ptr->RC = GetNode(NODE)
            buildTree(ptr->RC)
        Else
            ptr->RC = NULL
        Endif
    Endif
STOP
```

```
Algorithm InsertTree
START
    If ROOT = NULL
        ROOT = GetNode(NODE)
        ROOT->LC = ROOT->RC = NULL
        Read ROOT->DATA
        Exit
    EndIf
    ptr = SearchLink(ROOT, KEY)
    If ptr = NULL
        Print "KEY not found"
        Exit
    Else
```

```
        If ptr->LC = NULL or ptr->RC = NULL
                Read option insert as left child or right child
                If option = left
                        If ptr->LC = NULL
                                ptr->LC = GetNode(NODE)
                                ptr->LC->LC = NULL
                                ptr->LC->RC = NULL
                                Read ptr->LC->DATA
                        Else
                                Print "KEY has a left child"
                        Endif
                Else if option = right
                        If ptr->RC = NULL
                                ptr->RC = GetNode(NODE)
                                ptr->RC->LC = NULL
                                ptr->RC->RC = NULL
                                Read ptr->RC->DATA
                        Else
                                Print "KEY has a right child"
                        Endif
                Endif
        Else
                Print "KEY has both left child and right child"
        Endif
Endif
STOP
```

```
Algorithm  DeleteTree
START
If ROOT = NULL
        Print "Tree is empty"
        Exit
Else if KEY = ROOT->DATA
        If ROOT->LC = NULL and ROOT->RC = NULL
                ReturnNode(ROOT)
                ROOT = NULL
        Else
                Print "KEY is not a leaf node"
        Endif
        Exit
Endif
parent = SearchParent(ROOT, KEY, ROOT)
If parent = NULL
        Print "KEY not found"
Else
        If parent->LC != NULL and parent->LC->DATA = KEY
                If parent->LC->LC = NULL and parent->LC->RC = NULL
                        ReturnNode(parent->LC)
                        parent->LC = NULL
                Else
                        Print "KEY is not a leaf node"
                Endif
```

```
        Else
                If parent->RC->LC = NULL and parent->RC->RC = NULL
                        ReturnNode(parent->RC)
                        parent->RC = NULL
                Else
                        Print "KEY is not a leaf node"
                Endif
        Endif
Endif
STOP
```

```
Algorithm  InorderTraversal(ptr)
START
If ptr = NULL           //Initially, ptr = ROOT
        Print " Tree is empty"
Else
        If ptr->LC != NULL
                InorderTraversal(ptr->LC)
        Endif
        Print ptr->DATA
        If ptr->RC != NULL
                InorderTraversal(ptr->RC)
        Endif
Endif
STOP
```

```
Algorithm  PreorderTraversal(ptr)
START
If ptr = NULL           //Initially, ptr = ROOT
        Print " Tree is empty"
Else
        Print ptr->DATA
        If ptr->LC != NULL
                PreorderTraversal(ptr->LC)
        Endif
        If ptr->RC != NULL
                PreorderTraversal(ptr->RC)
        Endif
Endif
STOP
```

```
Algorithm  PostorderTraversal(ptr)
START
If ptr = NULL           //Initially, ptr = ROOT
        Print " Tree is empty"
Else
        If ptr->LC != NULL
                PostorderTraversal(ptr->LC)
        Endif
        If ptr->RC != NULL
                PostorderTraversal(ptr->RC)
        Endif
```

```
        Print ptr->DATA
Endif
STOP
```

```
Algorithm  SearchLink(ptr, KEY)
START
If ptr->DATA != KEY          //Initially ptr = ROOT
        If ptr->LC != NULL
                ptr1 = SearchLink(ptr->LC, KEY)
                If ptr1 != NULL
                        Return ptr1
                Endif
        Endif
        If ptr->RC != NULL
                ptr1 = SearchLink(ptr->RC, KEY)
                If ptr1 != NULL
                        Return ptr1
                Endif
        Endif
        Return NULL
Else
        Return ptr
Endif
STOP
```

```
Algorithm  SearchParent(ptr, KEY, parent)
START
If ptr->DATA != KEY          //Initially, ptr = ROOT
        If ptr->LC != NULL
                parent = SearchParent(ptr->LC, KEY, ptr)
                If parent != NULL
                        Return parent
                Endif
        Endif
        If ptr->RC != NULL
                parent = SearchParent(ptr->RC, KEY, ptr)
                If parent != NULL
                        Return parent
                Endif
        Endif
        Return NULL
Else
        Return parent
Endif
STOP
```

PROGRAM CODE :-

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

struct node
```

```
{
        int DATA;
        struct node* RC;
        struct node* LC;
};

void buildTree(struct node* ptr)
{
        if(ptr != NULL)
        {
                printf("Enter data\n");
                scanf("%d", &ptr->DATA);

                char c;
                printf("Does %d have a left child ? (Y/N)\n", ptr->DATA);
L1:
                scanf("\n%c", &c);

                if(c == 'y' || c == 'Y')
                {
                        ptr->LC = (struct node*) malloc(sizeof(struct node));
                        buildTree(ptr->LC);
                }
                else if(c == 'n' || c == 'N')
                {
                        ptr->LC = NULL;
                }
                else
                {
                        printf("Enter Y/N\n");
                        goto L1;
                }

                printf("Does %d have a right child ? (Y/N)\n", ptr->DATA);
L2:
                scanf("\n%c", &c);

                if(c == 'y' || c == 'Y')
                {
                        ptr->RC = (struct node*) malloc(sizeof(struct node));
                        buildTree(ptr->RC);
                }
                else if(c == 'n' || c == 'N')
                {
                        ptr->RC = NULL;
                }
                else
                {
                        printf("Enter Y/N!\n");
                        goto L2;
                }
        }
}
```

```
void preOrder(struct node* ptr)
{
	if(ptr == NULL)
		printf(" Tree is empty!");
	else
	{
		printf(" %d", ptr->DATA);
		if(ptr->LC != NULL)
			preOrder(ptr->LC);
		if(ptr->RC != NULL)
			preOrder(ptr->RC);
	}
}

void inOrder(struct node* ptr)
{
	if(ptr == NULL)
		printf(" Tree is empty!");
	else
	{
		if(ptr->LC != NULL)
			inOrder(ptr->LC);
		printf(" %d", ptr->DATA);
		if(ptr->RC != NULL)
			inOrder(ptr->RC);
	}
}

void postOrder(struct node* ptr)
{
	if(ptr == NULL)
		printf(" Tree is empty!");
	else
	{
		if(ptr->LC != NULL)
			postOrder(ptr->LC);
		if(ptr->RC != NULL)
			postOrder(ptr->RC);
		printf(" %d", ptr->DATA);
	}
}

struct node* searchParent(struct node* ptr, int KEY, struct node* parent)
{
	if(ptr->DATA != KEY)
	{
		if(ptr->LC != NULL)
		{
			parent = searchParent(ptr->LC, KEY, ptr);
			if(parent != NULL)
				return parent;
		}
		if(ptr->RC != NULL)
		{
```

```
				parent = searchParent(ptr->RC, KEY, ptr);
				if(parent != NULL)
					return parent;
			}
			return NULL;
		}
		else
			return parent;
}

void insertTree(struct node* ptr, int KEY)
{
		if(ptr == NULL)
			printf("\n%d not found!\n", KEY);
		else
		{
			if(ptr->LC == NULL || ptr->RC == NULL)
			{
				char c;
				printf("\nDo you want to insert as left child or right child of %d ? (L/R)\n", ptr-
>DATA);
L3:
				scanf("\n%c", &c);

				if(c == 'l' || c == 'L')
				{
					if(ptr->LC == NULL)
					{
						ptr->LC = (struct node*) malloc(sizeof(struct node));
						ptr->LC->LC = NULL;
						ptr->LC->RC = NULL;
						printf("\nEnter data\n");
						scanf("%d", &ptr->LC->DATA);
						printf("\n%d inserted successfully!\n", ptr->LC->DATA);
					}
					else
						printf("\n%d has a left child already!\n", KEY);
				}
				else if(c == 'r' || c == 'R')
				{
					if(ptr->RC == NULL)
					{
						ptr->RC = (struct node*) malloc(sizeof(struct node));
						ptr->RC->LC = NULL;
						ptr->RC->RC = NULL;
						printf("\nEnter data\n");
						scanf("%d", &ptr->RC->DATA);
						printf("\n%d inserted successfully!\n", ptr->RC->DATA);
					}
					else
						printf("\n%d has a right child already!\n", KEY);
				}
				else
				{
```

```
				printf("\nEnter L/R!\n");
				goto L3;
			}
		}
		else
			printf("\n%d has both left and right children!\n", KEY);
	}
}

void deleteTree(struct node* parent, int KEY)
{
	if(parent == NULL)
		printf("\n%d not found!\n", KEY);
	else
	{
		if(parent->LC != NULL && parent->LC->DATA == KEY)
			if(parent->LC->LC == NULL && parent->LC->RC == NULL)
			{
				free(parent->LC);
				parent->LC = NULL;
				printf("\n%d deleted successfully!\n", KEY);
			}
			else
				printf("\n%d is not a leaf node!\n", KEY);
		else
			if(parent->RC->LC == NULL && parent->RC->RC == NULL)
			{
				free(parent->RC);
				parent->RC = NULL;
				printf("\n%d deleted successfully!\n", KEY);
			}
			else
				printf("\n%d is not a leaf node!\n", KEY);
	}
}

struct node* searchLink(struct node* ptr, int KEY)
{
	struct node* ptr1;

	if(ptr->DATA != KEY)
	{
		if(ptr->LC != NULL)
		{
			ptr1 = searchLink(ptr->LC, KEY);
			if(ptr1 != NULL)
				return ptr1;
		}
		if(ptr->RC != NULL)
		{
			ptr1 = searchLink(ptr->RC, KEY);
			if(ptr1 != NULL)
				return ptr1;
		}
```

```
			return NULL;
		}
		else
			return ptr;
}

void main()
{
		struct node* ROOT = (struct node*) malloc(sizeof(struct node));

		int n;

		printf("Build your tree\n\n");
		buildTree(ROOT);
L:
		printf("\nChoose the operation\n\n");
		printf("1. Insert a node\n");
		printf("2. Delete a node\n");
		printf("3. Inorder traversal\n");
		printf("4. Preorder traversal\n");
		printf("5. Postorder traversal\n");
		printf("6. Exit\n");

		scanf("%d", &n);
		switch(n)
		{
			case 1:
				if(ROOT == NULL)
				{
					ROOT = (struct node*) malloc(sizeof(struct node));
					ROOT->LC = NULL;
					ROOT->RC = NULL;
					printf("\nEnter data\n");
					scanf("%d", &ROOT->DATA);
					printf("\n%d inserted successfully!\n", ROOT->DATA);
				}
				else
				{
					printf("\nEnter the KEY data\n");
					scanf("%d", &n);
					insertTree(searchLink(ROOT, n), n);
				}
				goto L;
			case 2:
				if(ROOT == NULL)
					printf("\nTree is empty!\n");
				else
				{
					printf("\nEnter the data to be deleted\n");
					scanf("%d", &n);
					if(n == ROOT->DATA)
					{
						if(ROOT->LC == NULL && ROOT->RC == NULL)
						{
```

```
                                        free(ROOT);
                                        ROOT = NULL;
                                        printf("\n%d deleted successfully!\n", n);
                                }
                                else
                                        printf("\n%d is not a leaf node!\n", n);
                        }
                        else
                                deleteTree(searchParent(ROOT, n, ROOT), n);
                }
                goto L;
        case 3:
                printf("\nInorder :");
                inOrder(ROOT);
                printf("\n");
                goto L;
        case 4:
                printf("\nPreorder :");
                preOrder(ROOT);
                printf("\n");
                goto L;
        case 5:
                printf("\nPostorder :");
                postOrder(ROOT);
                printf("\n");
                goto L;
        case 6:
                exit(0);
        default:
                printf("\nInvalid entry!\n");
                goto L;
        }
}
```

SAMPLE OUTPUT :-

Build your tree

Enter data
3
Does 3 have a left child ? (Y/N)
y
Enter data
1
Does 1 have a left child ? (Y/N)
n
Does 1 have a right child ? (Y/N)
y
Enter data
2
Does 2 have a left child ? (Y/N)
n
Does 2 have a right child ? (Y/N)
n

Does 3 have a right child ? (Y/N)
y
Enter data
4
Does 4 have a left child ? (Y/N)
n
Does 4 have a right child ? (Y/N)
y
Enter data
5
Does 5 have a left child ? (Y/N)
n
Does 5 have a right child ? (Y/N)
n

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Exit
3

Inorder : 1 2 3 4 5

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Exit
4

Preorder : 3 1 2 4 5

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Exit
5

Postorder : 2 1 5 4 3

Choose the operation

1. Insert a node

2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Exit
1

Enter the KEY data
1

Do you want to insert as left child or right child of 1 ? (L/R)
l

Enter data
0

0 inserted successfully!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Exit
3

Inorder : 0 1 2 3 4 5

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Exit
1

Enter the KEY data
4

Do you want to insert as left child or right child of 4 ? (L/R)
r

4 has a right child already!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal

5. Postorder traversal
6. Exit
1

Enter the KEY data
3

3 has both left and right children!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Exit
1

Enter the KEY data
4

Do you want to insert as left child or right child of 4 ? (L/R)
l

Enter data
10

10 inserted successfully!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Exit
3

Inorder : 0 1 2 3 10 4 5

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Exit
2

Enter the data to be deleted
10

10 deleted successfully!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Exit
3

Inorder : 0 1 2 3 4 5

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Exit
2

Enter the data to be deleted
0

0 deleted successfully!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Exit
2

Enter the data to be deleted
5

5 deleted successfully!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Exit
2

Enter the data to be deleted
4

4 deleted successfully!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Exit
2

Enter the data to be deleted
2

2 deleted successfully!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Exit
2

Enter the data to be deleted
1

1 deleted successfully!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Exit
3

Inorder : 3

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal

5. Postorder traversal
6. Exit
2

Enter the data to be deleted
3

3 deleted successfully!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Exit
3

Inorder : Tree is empty!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Exit
4

Preorder : Tree is empty!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Exit
5

Postorder : Tree is empty!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Exit
1

Enter data
3

3 inserted successfully!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Exit
1

Enter the KEY data
3

Do you want to insert as left child or right child of 3 ? (L/R)
r

Enter data
4

4 inserted successfully!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Exit
3

Inorder : 3 4

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Exit
2

Enter the data to be deleted
3

3 is not a leaf node!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Exit
2

Enter the data to be deleted
4

4 deleted successfully!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Exit
2

Enter the data to be deleted
3

3 deleted successfully!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Exit
3

Inorder : Tree is empty!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Exit
6

RESULT :- The given operations are performed on a binary tree.

EXPERIMENT 10b

AIM :- Create a binary search tree with the following operations:

1. Insert a new node

2. Inorder traversal

3. Preorder traversal

4. Postorder traversal

5. Delete a node

6. Count the number of leaf nodes

DATA STRUCTURE USED :- TREE using LINKED LIST is the data structure used.

ALGORITHM :-

```
Algorithm InsertBST
START
    If ROOT = NULL
        ROOT = GetNode(NODE)
        ROOT->LC = ROOT->RC = NULL
        ROOT->DATA = ITEM
        Exit
    Endif
    ptr = ROOT
    flag = false
    While ptr != NULL and flag = false
        If ITEM < ptr->DATA
                ptr1 = ptr
                ptr = ptr->LC
        Else if ITEM > ptr->DATA
                ptr1 = ptr
                ptr = ptr->RC
        Else
            flag = true
            Print "ITEM already exists"
        Endif
    Endwhile
    If ptr = NULL
        If ptr1->DATA < ITEM
            ptr1->RC = GetNode(NODE)
            ptr1->RC->LC = NULL
            ptr1->RC->RC = NULL
            ptr1->RC->DATA = ITEM
        Else
            ptr1->LC = GetNode(NODE)
            ptr1->LC->LC = NULL
            ptr1->LC->RC = NULL
            ptr1->LC->DATA = ITEM
        Endif
```

```
Endif
STOP

Algorithm DeleteBST
START
If ROOT = NULL
        Print "Tree is empty"
        Exit
Endif
If ITEM = ROOT->DATA
        If ROOT->LC = NULL and ROOT->RC = NULL
                ReturnNode(ROOT)
                ROOT = NULL
        Else
                If ROOT->LC != NULL and ROOT->RC != NULL
                        ptr1 = SuccessorOf(ROOT)
                        int temp = ptr1->DATA
                        DeleteBST(ptr1->DATA)
                        ROOT->DATA = temp
                Else
                        If ROOT->LC = NULL
                                ptr1 = ROOT->RC
                                ReturnNode(ROOT)
                                ROOT = ptr1
                        Else
                                ptr1 = ROOT->LC
                                ReturnNode(ROOT)
                                ROOT = ptr1
                        Endif
                Endif
        Endif
        Exit
Endif
ptr = ROOT
flag = false
While ptr != NULL and flag = false
        If ITEM < ptr->DATA
                parent = ptr
                ptr = ptr->LC
        Else if ITEM > ptr->DATA
                parent = ptr
                ptr = ptr->RC
        Else
                flag = true
        Endif
Endwhile
If flag = false
        Print "ITEM doesn't exist"
        Exit
Endif
If ptr->LC = NULL and ptr->RC = NULL
        CASE = 1
Else
        If ptr->LC != NULL and ptr->RC != NULL
```

```
            CASE = 3
        Else
            CASE = 2
        Endif
Endif
If CASE = 1
        If parent->LC = ptr
            parent->LC = NULL
        Else
            parent->RC = NULL
        Endif
        ReturnNode(ptr)
Else if CASE = 2
        If parent->LC = ptr
            If ptr->LC = NULL
                parent->LC = ptr->RC
            Else
                parent->LC = ptr->LC
            Endif
        Else
            If ptr->LC = NULL
                parent->RC = ptr->RC
            Else
                parent->RC = ptr->LC
            Endif
        Endif
        ReturnNode(ptr)
Else
        parent = SuccessorOf(ptr)
        ITEM = parent->DATA
        DeleteBST(parent->DATA)
        ptr->DATA = ITEM
Endif
STOP
```

```
Algorithm  InorderTraversal(ptr)
START
If ptr = NULL  //Initially, ptr = ROOT
        Print " Tree is empty"
Else
        If ptr->LC != NULL
            InorderTraversal(ptr->LC)
        Endif
        Print ptr->DATA
        If ptr->RC != NULL
            InorderTraversal(ptr->RC)
        Endif
Endif
STOP
```

```
Algorithm  PreorderTraversal(ptr)
START
If ptr = NULL  //Initially, ptr = ROOT
        Print " Tree is empty"
```

```
Else
        Print ptr->DATA
        If ptr->LC != NULL
                PreorderTraversal(ptr->LC)
        Endif
        If ptr->RC != NULL
                PreorderTraversal(ptr->RC)
        Endif
Endif
STOP
```

```
Algorithm  PostorderTraversal(ptr)
START
If ptr = NULL  //Initially, ptr = ROOT
        Print " Tree is empty"
Else
        If ptr->LC != NULL
                PostorderTraversal(ptr->LC)
        Endif
        If ptr->RC != NULL
                PostorderTraversal(ptr->RC)
        Endif
        Print ptr->DATA
Endif
STOP
```

```
Algorithm LeafCount(ptr)
START
count = 0
If ptr = NULL           //Initially, ptr = ROOT
        Return 0
Else
        If ptr->LC != NULL
                count += LeafCount(ptr->LC)
        Endif
        If ptr->RC != NULL
                count += LeafCount(ptr->RC)
        Endif
        If ptr->LC = NULL and ptr->RC = NULL
                count++
        Endif
Endif
Return count
STOP
```

```
Algorithm SuccessorOf(ptr)
START
ptr1 = ptr->RC
If ptr1 != NULL
        While ptr1->LC != NULL
                ptr1 = ptr1->LC
        Endwhile
Endif
Return(ptr1)
```

STOP

PROGRAM CODE :-

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

struct node
{
	int DATA;
	struct node* LC;
	struct node* RC;
};

void preOrder(struct node* ptr)
{
	if(ptr == NULL)
		printf(" Tree is empty!");
	else
	{
		printf(" %d", ptr->DATA);
		if(ptr->LC != NULL)
			preOrder(ptr->LC);
		if(ptr->RC != NULL)
			preOrder(ptr->RC);
	}
}

void inOrder(struct node* ptr)
{
	if(ptr == NULL)
		printf(" Tree is empty!");
	else
	{
		if(ptr->LC != NULL)
			inOrder(ptr->LC);
		printf(" %d", ptr->DATA);
		if(ptr->RC != NULL)
			inOrder(ptr->RC);
	}
}

void postOrder(struct node* ptr)
{
	if(ptr == NULL)
		printf(" Tree is empty!");
	else
	{
		if(ptr->LC != NULL)
			postOrder(ptr->LC);
		if(ptr->RC != NULL)
			postOrder(ptr->RC);
		printf(" %d", ptr->DATA);
```

```
	}
}

int leafNum(struct node* ptr)
{
	int count = 0;

	if(ptr == NULL)
		return 0;
	else
	{
		if(ptr->LC != NULL)
			count += leafNum(ptr->LC);
		if(ptr->RC != NULL)
			count += leafNum(ptr->RC);
		if(ptr->LC == NULL && ptr->RC == NULL)
			count++;
	}

	return count;
}

struct node* succ(struct node* ptr)
{
	struct node* ptr1 = ptr->RC;

	if(ptr1 != NULL)	//No need to check in this program
		while(ptr1->LC != NULL)
			ptr1 = ptr1->LC;

	return(ptr1);
}

void insertBST(struct node* ptr, int ITEM)
{
	struct node* ptr1;
	bool flag = false;

	while(ptr != NULL && flag == false)
	{
		if(ITEM < ptr->DATA)
		{
			ptr1 = ptr;
			ptr = ptr->LC;
		}
		else if(ITEM > ptr->DATA)
		{
			ptr1 = ptr;
			ptr = ptr->RC;
		}
		else
		{
			flag = true;
			printf("\n%d already exists!\n", ITEM);
```

```
		}
	}

	if(ptr == NULL)
	{
		if(ptr1->DATA < ITEM)
		{
			ptr1->RC = (struct node*) malloc(sizeof(struct node));
			ptr1->RC->LC = NULL;
			ptr1->RC->RC = NULL;
			ptr1->RC->DATA = ITEM;
			printf("\n%d inserted successfully!\n", ITEM);
		}
		else
		{
			ptr1->LC = (struct node*) malloc(sizeof(struct node));
			ptr1->LC->LC = NULL;
			ptr1->LC->RC = NULL;
			ptr1->LC->DATA = ITEM;
			printf("\n%d inserted successfully!\n", ITEM);
		}
	}
}

bool deleteBST(struct node* ROOT, int ITEM)
{
	struct node* ptr = ROOT;
	bool flag = false;
	struct node* parent;
	int CASE;

	while(ptr != NULL && flag == false)
	{
		if(ITEM < ptr->DATA)
		{
			parent = ptr;
			ptr = ptr->LC;
		}
		else if(ITEM > ptr->DATA)
		{
			parent = ptr;
			ptr = ptr->RC;
		}
		else
			flag = true;
	}

	if(flag == false)
	{
		return flag;
	}

	if(ptr->LC == NULL && ptr->RC == NULL)
		CASE = 1;
```

```
	else
		if(ptr->LC != NULL && ptr->RC != NULL)
			CASE = 3;
		else
			CASE = 2;

	if(CASE == 1)
	{
		if(parent->LC == ptr)
			parent->LC = NULL;
		else
			parent->RC = NULL;
		free(ptr);
	}
	else if(CASE == 2)
	{
		if(parent->LC == ptr)
			if(ptr->LC == NULL)
				parent->LC = ptr->RC;
			else
				parent->LC = ptr->LC;
		else
			if(ptr->LC == NULL)
				parent->RC = ptr->RC;
			else
				parent->RC = ptr->LC;
		free(ptr);
	}
	else
	{
		parent = succ(ptr);
		ITEM = parent->DATA;
		deleteBST(ROOT, parent->DATA);
		ptr->DATA = ITEM;
	}
	return flag;
}

void main()
{
	struct node* ROOT = NULL;
	struct node* ptr1;

	int n;
L:
	printf("\nChoose the operation\n\n");
	printf("1. Insert a node\n");
	printf("2. Delete a node\n");
	printf("3. Inorder traversal\n");
	printf("4. Preorder traversal\n");
	printf("5. Postorder traversal\n");
	printf("6. Count no. of leaf nodes\n");
	printf("7. Exit\n");
```

```
scanf("%d", &n);
switch(n)
{
	case 1:
		if(ROOT == NULL)
		{
			ROOT = (struct node*) malloc(sizeof(struct node));
			ROOT->LC = NULL;
			ROOT->RC = NULL;
			printf("\nEnter data\n");
			scanf("%d", &ROOT->DATA);
			printf("\n%d inserted successfully!\n", ROOT->DATA);
		}
		else
		{
			printf("\nEnter data\n");
			scanf("%d", &n);
			insertBST(ROOT, n);
		}
		goto L;
	case 2:
		if(ROOT == NULL)
			printf("\nTree is empty!\n");
		else
		{
			printf("\nEnter the data to be deleted\n");
			scanf("%d", &n);
			if(n == ROOT->DATA)
			{
				if(ROOT->LC == NULL && ROOT->RC == NULL)
				{
					free(ROOT);
					ROOT = NULL;
					printf("\n%d deleted successfully!\n", n);
				}
				else
				{
					if(ROOT->LC != NULL && ROOT->RC != NULL)
					{
						ptr1 = succ(ROOT);
						int temp = ptr1->DATA;
						deleteBST(ROOT, ptr1->DATA);
						ROOT->DATA = temp;
						printf("\n%d deleted successfully!\n", n);
					}
					else
					{
						if(ROOT->LC == NULL)
						{
							ptr1 = ROOT->RC;
							free(ROOT);
							ROOT = ptr1;
						}
						else
```

```
                        {
                                ptr1 = ROOT->LC;
                                free(ROOT);
                                ROOT = ptr1;
                        }
                        printf("\n%d deleted successfully!\n", n);
                    }
                }
            }
            else
            {
                if(deleteBST(ROOT, n))
                    printf("\n%d deleted successfully!\n", n);
                else
                    printf("\n%d not found!\n", n);
            }
        }
        goto L;
    case 3:
        printf("\nInorder :");
        inOrder(ROOT);
        printf("\n");
        goto L;
    case 4:
        printf("\nPreorder :");
        preOrder(ROOT);
        printf("\n");
        goto L;
    case 5:
        printf("\nPostorder :");
        postOrder(ROOT);
        printf("\n");
        goto L;
    case 6:
        printf("\nNo of leaf nodes : %d\n", leafNum(ROOT));
        goto L;
    case 7:
        exit(0);
    default:
        printf("Invalid entry!\n");
        goto L;
  }
}
```

SAMPLE OUTPUT :-

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal

6. Count no. of leaf nodes
7. Exit
1

Enter data
3

3 inserted successfully!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
1

Enter data
1

1 inserted successfully!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
1

Enter data
2

2 inserted successfully!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
1

Enter data
5

5 inserted successfully!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit

1

Enter data

4

4 inserted successfully!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit

3

Inorder : 1 2 3 4 5

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit

6

No of leaf nodes : 2

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit

4

Preorder : 3 1 2 5 4

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
5

Postorder : 2 1 4 5 3

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
2

Enter the data to be deleted
3

3 deleted successfully!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
3

Inorder : 1 2 4 5

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes

7. Exit
2

Enter the data to be deleted
1

1 deleted successfully!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
3

Inorder : 2 4 5

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
2

Enter the data to be deleted
2

2 deleted successfully!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
3

Inorder : 4 5

Choose the operation

1. Insert a node
2. Delete a node

3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
2

Enter the data to be deleted
5

5 deleted successfully!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
3

Inorder : 4

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
2

Enter the data to be deleted
4

4 deleted successfully!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
2

Tree is empty!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
1

Enter data
1

1 inserted successfully!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
1

Enter data
2

2 inserted successfully!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
1

Enter data
3

3 inserted successfully!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal

5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
3

Inorder : 1 2 3

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
2

Enter the data to be deleted
2

2 deleted successfully!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
3

Inorder : 1 3

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
2

Enter the data to be deleted
1

1 deleted successfully!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
3

Inorder : 3

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
2

Enter the data to be deleted
3

3 deleted successfully!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
3

Inorder : Tree is empty!

Choose the operation

1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
7

RESULT :- The given operations are performed on a binary search tree.

EXPERIMENT 10c

AIM :- Write a program to sort a set of numbers using a binary search tree.

DATA STRUCTURE USED :- TREE using LINKED LIST is the data structure used.

ALGORITHM :-

```
Algorithm InsertBST
START
    While ptr != NULL               //Initially, ptr = ROOT
        If ITEM <= ptr->DATA
            ptr1 = ptr
            ptr = ptr->LC
        Else if ITEM > ptr->DATA
            ptr1 = ptr
            ptr = ptr->RC
        Endif
    Endwhile
    If ptr = NULL
        If ptr1->DATA < ITEM
            ptr1->RC = GetNode(NODE)
            ptr1->RC->LC = NULL
            ptr1->RC->RC = NULL
            ptr1->RC->DATA = ITEM
        Else
            ptr1->LC = GetNode(NODE)
            ptr1->LC->LC = NULL
            ptr1->LC->RC = NULL
            ptr1->LC->DATA = ITEM
        Endif
    Endif
STOP
```

```
Algorithm SortBST(ptr, arr)
START
    If ptr->LC != NULL              //Initially, ptr = ROOT, i = 0
        SortBST(ptr->LC, arr)
    Endif
    arr[i] = ptr->DATA
    i++
    If ptr->RC != NULL
        SortBST(ptr->RC, arr)
    Endif
STOP
```

PROGRAM CODE :-

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

```
int i = 0;

struct node
{
	int DATA;
	struct node* LC;
	struct node* RC;
};

void sortBST(struct node* ptr, int* arr)
{
	if(ptr->LC != NULL)
		sortBST(ptr->LC, arr);

	arr[i] = ptr->DATA;
	i++;

	if(ptr->RC != NULL)
		sortBST(ptr->RC, arr);
}

void insertBST(struct node* ptr, int ITEM)
{
	struct node* ptr1;

	while(ptr != NULL)
	{
		if(ITEM <= ptr->DATA)
		{
			ptr1 = ptr;
			ptr = ptr->LC;
		}
		else if(ITEM > ptr->DATA)
		{
			ptr1 = ptr;
			ptr = ptr->RC;
		}
	}

	if(ptr == NULL)
	{
		if(ptr1->DATA < ITEM)
		{
			ptr1->RC = (struct node*) malloc(sizeof(struct node));
			ptr1->RC->LC = NULL;
			ptr1->RC->RC = NULL;
			ptr1->RC->DATA = ITEM;
		}
		else
		{
			ptr1->LC = (struct node*) malloc(sizeof(struct node));
```

```
				ptr1->LC->LC = NULL;
				ptr1->LC->RC = NULL;
				ptr1->LC->DATA = ITEM;
			}
		}
}

void main()
{
	int* arr;
	int n;

	printf("Enter the array size\n");
	scanf("%d", &n);

	arr = malloc(n*sizeof(int));

	printf("Enter the numbers\n");
	for(int i=0; i<n; i++)
		scanf("%d", &arr[i]);

	struct node* ROOT = (struct node*) malloc(sizeof(struct node));
	ROOT->LC = NULL;
	ROOT->RC = NULL;
	ROOT->DATA = arr[0];

	for(int i=1; i<n; i++)
		insertBST(ROOT, arr[i]);

	sortBST(ROOT, arr);

	printf("Sorted array: ");

	for(int i=0; i<n; i++)
		printf("%d ", arr[i]);
	printf("\n");
}
```

SAMPLE OUTPUTS :-

1.)

Enter the array size
3
Enter the numbers
3
2
1
Sorted array: 1 2 3

2.)

Enter the array size

5
Enter the numbers
1
5
2
4
3
Sorted array: 1 2 3 4 5

3.)
Enter the array size
4
Enter the numbers
-1
2
-1
0
Sorted array: -1 -1 0 2

4.)
Enter the array size
10
Enter the numbers
-2
24
12
12
14
-12
1
23
6
15
Sorted array: -12 -2 1 6 12 12 14 15 23 24

RESULT :- The given set of numbers are sorted using a binary search tree.

EXPERIMENT 11

AIM :- Write a program to create a graph using arrays and perform the following operations:

1. DFS Traversal

2. BFS Traversal

DATA STRUCTURE USED :- GRAPH using ARRAY is the data structure used (STACK and QUEUE are also used for the traversal algorithms).

ALGORITHM :-

```
Algorithm CreateGraph
START
    1   For i = 1 till n            //n is the total number of vertices
    2           Read vertex in arr[i][0] and arr[0][i]
    3   EndFor
    4   For i = 1 till n
    5           For j = i till n
    6                   Read option "Are arr[i][0] and arr[0][j] adjacent ?"
    7                   If option = Y
    8                           arr[i][j] = 1
    9                           arr[j][i] = 1
    10                  Else if option = N
    11                          arr[i][j] = 0
    12                          arr[j][i] = 0
    13                  EndIf
    14          EndFor
    15  EndFor
STOP

Algorithm DFS
START
    16  Push the starting vertex into the stack
    17  While stack not empty
    18          Pop a vertex v
    19          If v is not in VISIT
    20                  Visit the vertex x
    21                  Store v in VISIT
    22                  Push all the adjacent vertices of v into stack
    23          EndIf
    24  EndWhile
STOP

Algorithm BFS
START
    1   Enqueue starting vertex
    2   Visit the vertex
    3   Store the vertex in VISIT
    4   While queue not empty
```

```
        Dequeue a vertex v
        For all the adjacent vertices w of v
                If w is not in VISIT
                        Enqueue w
                        Visit w
                        Store w in VISIT
                EndIf
        EndFor
    EndWhile
STOP
```

PROGRAM CODE :-

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

struct stack
{
	int size;
	int TOP;
	int *arr;
};

struct queue
{
	int FRONT;
	int REAR;
	int *arr;
	int SIZE;
};

int isFull(struct stack *st)
{
	if(st->TOP >= st->size-1)
		return 1;
	return 0;
}

int isEmpty(struct stack *st)
{
	if(st->TOP == -1)
		return 1;
	return 0;
}

void push(struct stack *st, char x)
{
	if(!isFull(st))
	{
		st->arr[++st->TOP] = x;
	}
}

char pop(struct stack *st)
```

```
{
	if(!isEmpty(st))
	{
		char x = st->arr[st->TOP];
		st->TOP--;
		return x;
	}
}

void createStack(struct stack *st, int n)
{
	st->size = n;

	st->arr = (int*) malloc (st->size * sizeof(int));

	st->TOP = -1;
}

void enqueue(struct queue *q, char X)
{
	if(q->REAR != q->SIZE-1)
	{
		if(q->FRONT == -1)
			q->FRONT = 0;
		q->REAR += 1;
		q->arr[q->REAR] = X;
	}
}

char dequeue(struct queue *q)
{
	if(q->FRONT != -1)
	{
		char X = q->arr[q->FRONT];

		if(q->FRONT == q->REAR)
		{
			q->FRONT = -1;
			q->REAR = -1;
		}
		else
			q->FRONT += 1;
		return X;
	}
}

void createQueue(struct queue *q, int n)
{
	q->SIZE = n;

	q->arr = malloc(q->SIZE * sizeof(int));

	q->FRONT = -1;
	q->REAR = -1;
```

```
}

void dfs(int n, char arr[][n+1])
{
	struct stack *st = malloc(sizeof(struct stack));
	int count = 0;
	int i = 0;
	char v;
	char visit[n];

	createStack(st, n*n);

	push(st, arr[0][1]);

	while(!isEmpty(st))
	{
		v = pop(st);

		for(int j=0; j<n; j++)
			if(visit[j] == v)
				count++;
		if(count == 0)
		{
			printf("%c ", v);
			visit[i] = v;
			i++;

			for(int j=1; i<=n; j++)
				if(arr[0][j] == v)
				{
					for(int k=1; k<=n; k++)
						if(arr[k][j] == '1')
							push(st, arr[k][0]);
					break;
				}
		}

		count = 0;
	}
}

void bfs(int n, char arr[][n+1])
{
	struct queue *q = malloc(sizeof(struct queue));
	int i = 1;
	int count = 0;
	char visit[n];
	char v;

	createQueue(q, n*n);

	enqueue(q, arr[0][1]);
	printf("%c ", arr[0][1]);
	visit[0] = arr[0][1];
```

```
	while(q->FRONT != -1)
	{
		v = dequeue(q);
		for(int j=1; i<=n; j++)
			if(arr[0][j] == v)
			{
				for(int k=1; k<=n; k++)
					if(arr[k][j] == '1')
					{
						for(int l=0; l<n; l++)
							if(visit[l] == arr[k][0])
								count++;
						if(count == 0)
						{
							enqueue(q, arr[k][0]);
							printf("%c ", arr[k][0]);
							visit[i] = arr[k][0];
							i++;
						}
						count = 0;
					}
				break;
			}
	}
}

void main()
{
	int n;
	char c;
	int m;

	printf("Enter the no. of vertices\n");
	scanf("%d", &n);

	char arr[n+1][n+1];

	arr[0][0] = ' ';

	printf("Enter the vertices\n");
	for(int i=1; i<=n; i++)
	{
		scanf("\n%c", &arr[i][0]);
		arr[0][i] = arr[i][0];
	}

	for(int i=1;i<=n;i++)
		for(int j=i;j<=n;j++)
		{
			if(arr[i][0] == arr[0][j])
			{
				printf("Is %c a self loop ? (Y/N)\n", arr[i][0]);
L1:
```

```
				scanf("\n%c", &c);

				if(c == 'y' || c == 'Y')
					arr[i][j] = '1';
				else if(c == 'n' || c == 'N')
					arr[i][j] = '0';
				else
				{
					printf("Enter Y/N!\n");
					goto L1;
				}
				continue;
			}

			printf("Are %c and %c adjacent ? (Y/N)\n", arr[i][0], arr[0][j]);
L2:
			scanf("\n%c", &c);

			if(c == 'y' || c == 'Y')
			{
				arr[i][j] = '1';
				arr[j][i] = '1';
			}
			else if(c == 'n' || c == 'N')
			{
				arr[i][j] = '0';
				arr[j][i] = '0';
			}
			else
			{
				printf("Enter Y/N!\n");
				goto L2;
			}
		}

	printf("\nAdjacency matrix of the graph:\n");
	for(int i=0;i<=n;i++)
	{
		for(int j=0;j<=n;j++)
			printf("%c ", arr[i][j]);
		printf("\n");
	}
L3:
	printf("\nChoose the operation\n");
	printf("1. DFS Traversal\n2. BFS Traversal\n3. Quit\n");
	scanf("%d", &m);

	if(m == 1)
	{
		printf("\nDFS Traversal : ");
		dfs(n, arr);
		printf("\n");
		goto L3;
	}
```

```
	else if(m == 2)
	{
		printf("\nBFS Traversal : ");
		bfs(n, arr);
		printf("\n");
		goto L3;
	}
	else if(m == 3)
		exit(0);
	else
	{
		printf("Invalid entry\n");
		goto L3;
	}
}
```

SAMPLE OUTPUT :-

```
Enter the no. of vertices
8
Enter the vertices
1
2
3
4
5
6
7
8
Is 1 a self loop ? (Y/N)
n
Are 1 and 2 adjacent ? (Y/N)
y
Are 1 and 3 adjacent ? (Y/N)
y
Are 1 and 4 adjacent ? (Y/N)
n
Are 1 and 5 adjacent ? (Y/N)
n
Are 1 and 6 adjacent ? (Y/N)
n
Are 1 and 7 adjacent ? (Y/N)
n
Are 1 and 8 adjacent ? (Y/N)
y
Is 2 a self loop ? (Y/N)
n
Are 2 and 3 adjacent ? (Y/N)
n
Are 2 and 4 adjacent ? (Y/N)
y
Are 2 and 5 adjacent ? (Y/N)
y
```

Are 2 and 6 adjacent ? (Y/N)
n
Are 2 and 7 adjacent ? (Y/N)
n
Are 2 and 8 adjacent ? (Y/N)
n
Is 3 a self loop ? (Y/N)
n
Are 3 and 4 adjacent ? (Y/N)
y
Are 3 and 5 adjacent ? (Y/N)
n
Are 3 and 6 adjacent ? (Y/N)
y
Are 3 and 7 adjacent ? (Y/N)
n
Are 3 and 8 adjacent ? (Y/N)
n
Is 4 a self loop ? (Y/N)
n
Are 4 and 5 adjacent ? (Y/N)
n
Are 4 and 6 adjacent ? (Y/N)
n
Are 4 and 7 adjacent ? (Y/N)
y
Are 4 and 8 adjacent ? (Y/N)
y
Is 5 a self loop ? (Y/N)
n
Are 5 and 6 adjacent ? (Y/N)
n
Are 5 and 7 adjacent ? (Y/N)
y
Are 5 and 8 adjacent ? (Y/N)
n
Is 6 a self loop ? (Y/N)
n
Are 6 and 7 adjacent ? (Y/N)
y
Are 6 and 8 adjacent ? (Y/N)
n
Is 7 a self loop ? (Y/N)
n
Are 7 and 8 adjacent ? (Y/N)
n
Is 8 a self loop ? (Y/N)
n

Adjacency matrix of the graph:
  1 2 3 4 5 6 7 8
1 0 1 1 0 0 0 0 1

2 1 0 0 1 1 0 0 0
3 1 0 0 1 0 1 0 0
4 0 1 1 0 0 0 1 1
5 0 1 0 0 0 0 1 0
6 0 0 1 0 0 0 1 0
7 0 0 0 1 1 1 0 0
8 1 0 0 1 0 0 0 0

Choose the operation
1. DFS Traversal
2. BFS Traversal
3. Quit
1

DFS Traversal : 1 8 4 7 6 3 5 2

Choose the operation
1. DFS Traversal
2. BFS Traversal
3. Quit
2

BFS Traversal : 1 2 3 8 4 5 6 7

Choose the operation
1. DFS Traversal
2. BFS Traversal
3. Quit
3

RESULT :- The given operations are performed on a graph using arrays.

EXPERIMENT 12

AIM :-

1. Create a text file containing the name, height, weight of the students in a class. Perform Quick sort and Merge sort on this data and store the resultant data in two separate files. Also write the time taken by the two sorting methods into the respective files. Eg. Sony Mathew 5.5 60 Arun Sajeev 5.7 58 Rajesh Kumar 6.1 70

2. Write a program to sort a set of numbers using Heap sort and find a particular number from the sorted set using Binary Search.

DATA STRUCTURES USED :- ARRAY and HEAP are the data structures used.

ALGORITHMS :-

```
Algorithm Partition(A, p, r)
START
    x = A[r]
    i = p-1
    for j = p to r
        if (A[j] <= x)
            i = i+1
            if (i != j)
                swap A[i] and A[j]
            endif
        endif
    endfor
    if (r != i+1)
        swap A[i+1] and A[r]
    endif
    return i+1
STOP
```

```
Algorithm QuickSort(A, p, r)
START
    if (p < r)
        q = Partition(A, p ,r)
        QuickSort(A, p, q-1)
        QuickSort(A, q+1, r)
    endif
STOP
```

```
Algorithm Merge(A, p, q, r)
START
    n1 = q - p + 1
    n2 = r - q
    Declare L[n1], R[n2]
    for i = 0 till n1
        L[i] = A[p+i]
```

```
endfor
for j = 0 till n2
        R[j] = A[q+j+1]
endfor
i = 0, j = 0
for k = p to r
        if (L[i] <= R[j])
                A[k] = L[i]
                i = i+1
                if (i == n1)
                        k = k+1
                        break
                endif
        else
                A[k] = R[j]
                j = j+1
                if (j == n2)
                        k = k+1
                        break
                endif
        endif
endfor
while (i < n1)
        A[k] = L[i]
        i = i+1
        k = k+1
endwhile
while (j < n2)
        A[k] = R[j]
        j = j+1
        k = k+1
endwhile
STOP
```

```
Algorithm MergeSort(A, p, r)
START
if (p < r)
        q = floor((p+r)/2)
        MergeSort(A, p, q)
        MergeSort(A, q+1, r)
        Merge(A, p, q, r)
endif
STOP
```

```
Algorithm CreateHeap(A, n)
START
i = 0
while (i < n)
        j = i
        while (j > 0)
                if (A[j] > A[(j-1)/2])
                                swap A[j] and A[(j-1)/2]
```

```
                j = (j-1)/2
        else
            break
        endif
        i = i+1
    endwhile
endwhile
STOP
```

```
Algorithm RemoveMax(A, i)
START
swap A[i] and A[0]
STOP
```

```
Algorithm RebuildHeap(A, i)
START
if (i == 0)
    return
endif
j = 0
while(1)
    lc = 2 * j + 1
    rc = 2 * (j + 1)
    if (rc <= i)
        if(A[j] <= A[lc] && A[lc] >= A[rc])
            swap A[j] and A[lc]
            j = lc
        else if (A[j] <= A[rc] && A[rc] >= A[lc])
            swap A[j] and A[rc]
            j = rc
        else
            break
    else if (lc <= i)
        if (A[j] <= A[lc])
            swap A[j] and A[lc]
            break
        else
            break
    else
        break
    endif
endwhile
STOP
```

```
Algorithm HeapSort(A, n)
START
CreateHeap(A, n)
for i = n-1 down till 0
    RemoveMax(A, i)
    RebuildHeap(A, i-1)
endfor
STOP
```

```
Algorithm BinarySearch(A, num, l, r)
START
    while (l <= r)
        m = l + (r - l) / 2
        if (A[m] == num)
            return m
        else if (A[m] < num)
            l = m + 1
        else
            r = m - 1
        endif
    return -1
STOP
```

PROGRAM CODE :-

1.)

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<time.h>
#include<math.h>

struct student
{
	char name[20];
	float height;
	float weight;
};

int partition(struct student* st, int p, int r)
{
	struct student temp;
	float x = st[r].height;
	int i = p-1;
	for(int j = p; j < r; j++)
		if(st[j].height <= x)
		{
			i++;
			if(i != j)
			{
				temp = st[i];
				st[i] = st[j];
				st[j] = temp;
			}
		}
	if(r != i+1)
	{
		temp = st[i+1];
		st[i+1] = st[r];
		st[r] = temp;
	}
```

```
	return i+1;
}

void quicksort(struct student* st, int p, int r)
{
	if(p < r)
	{
		int q = partition(st, p ,r);
		quicksort(st, p, q-1);
		quicksort(st, q+1, r);
	}
}

void merge(struct student* st1, int p, int q, int r)
{
	int n1 = q - p + 1;
	int n2 = r - q;

	struct student L[n1], R[n2];

	for(int i = 0; i < n1; i++)
		L[i] = st1[p+i];
	for(int j = 0; j < n2; j++)
		R[j] = st1[q+j+1];

	int i = 0, j = 0;
	int k;

	for(k = p; k <= r; k++)
	{
		if(L[i].height <= R[j].height)
		{
			st1[k] = L[i];
			i++;
			if(i == n1)
			{
				k++;
				break;
			}
		}
		else
		{
			st1[k] = R[j];
			j++;
			if(j == n2)
			{
				k++;
				break;
			}
		}
	}
	while(i < n1)
	{
		st1[k] = L[i];
```

```
		i++;
		k++;
	}
	while(j < n2)
	{
		st1[k] = R[j];
		j++;
		k++;
	}
}

void mergesort(struct student* st1, int p, int r)
{
	if(p < r)
	{
		int q = floor((p+r)/2);
		mergesort(st1, p, q);
		mergesort(st1, q+1, r);
		merge(st1, p, q, r);
	}
}

void main()
{
	int n;
	char c;

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

	FILE *fp = fopen("Student details.txt", "w");
	FILE *fpq = fopen("Quick student details.txt", "w");
	FILE *fpm = fopen("Merge student details.txt", "w");

	fprintf(fp, "NAME\t\tHEIGHT\tWEIGHT\n");
	fprintf(fpq, "NAME\t\tHEIGHT\tWEIGHT\n");
	fprintf(fpm, "NAME\t\tHEIGHT\tWEIGHT\n");

	struct student* st = malloc(n * sizeof(struct student));

	for(int i=0; i<n; i++)
	{
		printf("\nEnter the student details\n");
		printf("Name = ");
		scanf("%c", &c);
		fgets(st[i].name, 20, stdin);
		st[i].name[strlen(st[i].name) - 1] = '\0';
		printf("Height = ");
		scanf("%f", &st[i].height);
		printf("Weight = ");
		scanf("%f", &st[i].weight);
	}

	printf("\nWriting to file...\n");
```

```
	for(int i = 0; i < n; i++)
		fprintf(fp, "%s\t\t%.2f\t%.2f\n", st[i].name, st[i].height, st[i].weight);

	printf("\nPerforming quick sort...\n");
	clock_t t = clock();
	quicksort(st, 0, n-1);
	t = clock() - t;

	for(int i = 0; i < n; i++)
		fprintf(fpq, "%s\t\t%.2f\t%.2f\n", st[i].name, st[i].height, st[i].weight);
	fprintf(fpq, "\nTime taken = %lf", (double) t / CLOCKS_PER_SEC);

	for(int i = 0; i < n; i++)
		fscanf(fp, "%s\t\t%f\t%f\n", st[i].name, &st[i].height, &st[i].weight);

	printf("\nPerforming merge sort...\n");
	t = clock();
	mergesort(st, 0, n-1);
	t = clock() - t;

	for(int i = 0; i < n; i++)
		fprintf(fpm, "%s\t\t%.2f\t%.2f\n", st[i].name, st[i].height, st[i].weight);
	fprintf(fpm, "\nTime taken = %lf", (double) t / CLOCKS_PER_SEC);
	printf("\nWrite successful.\n\n");
}
```

2.)

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

void createheap(int* arr, int n)
{
	int i = 0, temp, j;

	while(i < n)
	{
		j = i;

		while(j > 0)
		{
			if(arr[j] > arr[(j-1)/2])
			{
				temp = arr[j];
				arr[j] = arr[(j-1)/2];
				arr[(j-1)/2] = temp;
				j = (j-1)/2;
			}
			else
				break;
		}
		i++;
	}
}
```

```
void removemax(int* arr, int i)
{
	int temp = arr[i];
	arr[i] = arr[0];
	arr[0] = temp;
}

void rebuildheap(int* arr, int i)
{
	if(i == 0)
		return;

	int j = 0, temp, lc, rc;

	while(1)
	{
		lc = 2 * j + 1;
		rc = 2 * (j + 1);

		if(rc <= i)
		{
			if(arr[j] <= arr[lc] && arr[lc] >= arr[rc])
			{
				temp = arr[j];
				arr[j] = arr[lc];
				arr[lc] = temp;
				j = lc;
			}
			else if(arr[j] <= arr[rc] && arr[rc] >= arr[lc])
			{
				temp = arr[j];
				arr[j] = arr[rc];
				arr[rc] = temp;
				j = rc;
			}
			else
				break;
		}
		else if(lc <= i)
		{
			if(arr[j] <= arr[lc])
			{
				temp = arr[j];
				arr[j] = arr[lc];
				arr[lc] = temp;
				break;
			}
			else
				break;
		}
		else
			break;
	}
```

```
}

void heapsort(int* arr, int n)
{
	createheap(arr, n);
	for(int i = n-1; i > 0; i--)
	{
		removemax(arr, i);
		rebuildheap(arr, i-1);
	}
}

int binarysearch(int* arr, int num, int l, int r)
{
	while(l <= r)
	{
		int m = l + (r - l) / 2;	//For small size, (l + r) / 2

		if(arr[m] == num)
			return m;
		else if(arr[m] < num)
			l = m + 1;
		else
			r = m - 1;
	}
	return -1;
}

void main()
{
	int n, num;

	printf("Enter the array size\n");
	scanf("%d", &n);

	int* arr = malloc(n * sizeof(int));

	printf("Enter the elements\n");
	for(int i = 0; i < n; i++)
		scanf("%d", &arr[i]);

	heapsort(arr, n);

	printf("\nThe sorted array: ");
	for(int i = 0; i < n; i++)
		printf("%d ", arr[i]);
	printf("\n");

	while(1)
	{
		printf("\nEnter the number to search (Enter -1 to exit)\n");
		scanf("%d", &num);

		if(num == -1)
```

```
                break;

            int index = binarysearch(arr, num, 0, n);
            if(index != -1)
                printf("%d found at index %d\n", num, index);
            else
                printf("Search unsuccessful!\n");
        }
}
```

SAMPLE OUTPUTS :-

1.)

Enter the number of students
3

Enter the student details
Name = Amal
Height = 156.555
Weight = 45

Enter the student details
Name = Vishnu
Height = 123.54
Weight = 34

Enter the student details
Name = Arjun
Height = 156.34
Weight = 56

Writing to file...

Performing quick sort...

Performing merge sort...

Write successful.

Student details.txt

NAME	HEIGHT	WEIGHT
Amal	156.55	45.00
Vishnu	123.54	34.00
Arjun	156.34	56.00

Quick student details.txt

NAME	HEIGHT	WEIGHT
Vishnu	123.54	34.00
Arjun	156.34	56.00
Amal	156.55	45.00

Time taken = 0.000002

<u>Merge student details.txt</u>

NAME	HEIGHT	WEIGHT
Vishnu	123.54	34.00
Arjun	156.34	56.00
Amal	156.55	45.00

Time taken = 0.000002

2.)
Enter the array size
5
Enter the elements
5
2
4
1
3

The sorted array: 1 2 3 4 5

Enter the number to search (Enter -1 to exit)
3
3 found at index 2

Enter the number to search (Enter -1 to exit)
1
1 found at index 0

Enter the number to search (Enter -1 to exit)
5
5 found at index 4

Enter the number to search (Enter -1 to exit)
2
2 found at index 1

Enter the number to search (Enter -1 to exit)
4
4 found at index 3

Enter the number to search (Enter -1 to exit)
6
Search unsuccessful!

Enter the number to search (Enter -1 to exit)
-1

RESULT :- Quick sort, Merge sort, Heap sort and Binary search are performed on the respective data.

EXPERIMENT 13

AIM :-

1. Implement a Hash table using Chaining method. Let the size of hash table be 10 so that the index varies from 0 to 9.

2. Implement a Hash table that uses Linear Probing for collision resolution.

DATA STRUCTURES USED :- HASH TABLE using LINKED LIST and ARRAY are the data structures used.

ALGORITHMS :-

```
Algorithm OpenHash
START
    Read key
    h = key % 10
    ptr = hash[h]
    while (ptr->LINK != NULL)          //OR INSERT AT FRONT
        ptr = ptr->LINK
    endwhile
    ptr->LINK = GetNode(NODE)
    ptr->LINK->DATA = key
    ptr->LINK->LINK = NULL
STOP
```

```
Algorithm ClosedHash
START
    Read key
    h = key % size
    if (hash[h] == 0)
        hash[h] = key
    else
        for i = h+1 till n
            if (hash[i] == 0)
                hash[i] = key
                return
            endif
        endfor
        for i = 0 till h
            if (hash[i] == 0)
                hash[i] = key
                return
            endif
        endfor
        print "Hash table is full!"
    endif
STOP
```

PROGRAM CODE :-

1.)

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

struct node
{
	int DATA;
	struct node* LINK;
};

void display(struct node** hash)
{
	struct node* ptr;
	for(int i = 0; i < 10; i++)
	{
		ptr = hash[i]->LINK;
		printf("\n%d - ", i);
		while(ptr != NULL)
		{
			printf("%d ", ptr->DATA);
			ptr = ptr->LINK;
		}
	}
	printf("\n");
}

void new_entry(struct node** hash)
{
	int key;

	printf("Enter the element\n");
	scanf("%d", &key);

	int h = key % 10;

	struct node *ptr = hash[h];

	while(ptr->LINK != NULL)
		 ptr = ptr->LINK;

	ptr->LINK = malloc(sizeof(struct node));
	ptr->LINK->DATA = key;
	ptr->LINK->LINK = NULL;

	display(hash);
}

void main()
{
	int flag;
```

```
	struct node** hash = malloc(10 * sizeof(struct node*));
	for(int i = 0; i < 10; i++)
	{
		hash[i] = malloc(sizeof(struct node));
		hash[i]->LINK = NULL;
	}

	while(1)
	{
		printf("\nEnter\n1. New entry\n2. Display Hash table\n3. Exit\n");
		scanf("%d", &flag);

		switch(flag)
		{
			case 1:
				new_entry(hash);
				break;
			case 2:
				display(hash);
				break;
			case 3:
				exit(0);
			default:
				printf("\nInvalid entry!\n");
		}
	}
}
```

2.)

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

void display(int hash[], int n)
{
	printf("\n");
	for(int i = 0; i < n; i++)
		printf("%d\n", hash[i]);
}

void new_entry(int hash[], int n)
{
	int key;

	printf("\nEnter the element\n");
	scanf("%d", &key);

	int h = key % n;

	if(hash[h] == 0)
	{
		hash[h] = key;
```

```
		display(hash, n);
	}
	else
	{
		for(int i = h+1; i < n; i++)
			if(hash[i] == 0)
			{
				hash[i] = key;
				display(hash, n);
				return;
			}
		for(int i = 0; i < h; i++)
			if(hash[i] == 0)
			{
				hash[i] = key;
				display(hash, n);
				return;
			}
		printf("\nHash table is full!\n");
	}
}

void main()
{
	int size, flag;

	printf("\nEnter size of hash table\n");
	scanf("%d", &size);

	int* hash = calloc(size, sizeof(int));

	while(1)
	{
		printf("\nEnter\n1. New entry\n2. Display Hash table\n3. Exit\n");
		scanf("%d", &flag);

		switch(flag)
		{
			case 1:
				new_entry(hash, size);
				break;
			case 2:
				display(hash, size);
				break;
			case 3:
				exit(0);
			default:
				printf("\nInvalid entry!\n");
				break;
		}
	}
}
```

SAMPLE OUTPUTS :-

1.)

Enter
1. New entry
2. Display Hash table
3. Exit
2

0 -
1 -
2 -
3 -
4 -
5 -
6 -
7 -
8 -
9 -

Enter
1. New entry
2. Display Hash table
3. Exit
1
Enter the element
5

0 -
1 -
2 -
3 -
4 -
5 - 5
6 -
7 -
8 -
9 -

Enter
1. New entry
2. Display Hash table
3. Exit
1
Enter the element
15

0 -
1 -
2 -
3 -

4 -
5 - 5 15
6 -
7 -
8 -
9 -

Enter
1. New entry
2. Display Hash table
3. Exit
1
Enter the element
2

0 -
1 -
2 - 2
3 -
4 -
5 - 5 15
6 -
7 -
8 -
9 -

Enter
1. New entry
2. Display Hash table
3. Exit
1
Enter the element
12

0 -
1 -
2 - 2 12
3 -
4 -
5 - 5 15
6 -
7 -
8 -
9 -

Enter
1. New entry
2. Display Hash table
3. Exit
1
Enter the element
15

0 -
1 -
2 - 2 12
3 -
4 -
5 - 5 15 15
6 -
7 -
8 -
9 -

Enter
1. New entry
2. Display Hash table
3. Exit
1
Enter the element
44

0 -
1 -
2 - 2 12
3 -
4 - 44
5 - 5 15 15
6 -
7 -
8 -
9 -

Enter
1. New entry
2. Display Hash table
3. Exit
2

0 -
1 -
2 - 2 12
3 -
4 - 44
5 - 5 15 15
6 -
7 -
8 -
9 -

Enter
1. New entry
2. Display Hash table
3. Exit

1
Enter the element
78

0 -
1 -
2 - 2 12
3 -
4 - 44
5 - 5 15 15
6 -
7 -
8 - 78
9 -

Enter
1. New entry
2. Display Hash table
3. Exit
1
Enter the element
16

0 -
1 -
2 - 2 12
3 -
4 - 44
5 - 5 15 15
6 - 16
7 -
8 - 78
9 -

Enter
1. New entry
2. Display Hash table
3. Exit
3

2.)

Enter size of hash table
5

Enter
1. New entry
2. Display Hash table
3. Exit
2

0
0
0
0
0

Enter
1. New entry
2. Display Hash table
3. Exit
1

Enter the element
5

5
0
0
0
0

Enter
1. New entry
2. Display Hash table
3. Exit
1

Enter the element
4

5
0
0
0
4

Enter
1. New entry
2. Display Hash table
3. Exit
1

Enter the element
14

5
14
0
0
4

Enter

1. New entry
2. Display Hash table
3. Exit
1

Enter the element
15

5
14
15
0
4

Enter
1. New entry
2. Display Hash table
3. Exit
1

Enter the element
3

5
14
15
3
4

Enter
1. New entry
2. Display Hash table
3. Exit
1

Enter the element
13

Hash table is full!

Enter
1. New entry
2. Display Hash table
3. Exit
3

RESULT :- Hash tables are implemented using open hashing (chaining) and closed hashing (linear probing).