**INDEX**

|  |  |  |
| --- | --- | --- |
| **SL NO** | **DATE** | **TITLE** |
| 1 | 27/08/2020 | Bubble Sort |
| 2 | 27/08/2020 | Selection Sort |
| 3 | 06/09/2020 | Insertion Sort |
| 4 | 18/09/2020 | Polynomial Addition |
| 5 | 20/09/2020 | Sparse Matrix |
| 6 | 20/09/2020 | Stack Using Array |
| 7 | 24/10/2020 | Infix to Postfix Conversion and Evaluation |
| 8 | 31/10/2020 | Queue Using Array |
| 9 | 03/11/2020 | Circular Queue |
| 10 | 04/11/2020 | Priority Queue |
| 11 | 04/11/2020 | Double Ended Queue (Deque) |
| 12 | 13/11/2020 | Single Linked List |
| 13 | 13/11/2020 | Stack Using Linked Lists |
| 14 | 13/11/2020 | Queue Using Linked Lists |
| 15 | 20/11/2020 | Polynomial Using Linked Lists |
| 16 | 22/11/2020 | Student Linked Lists |
| 17 | 22/11/2020 | Doubly Linked Lists |
| 18 | 02/01/2021 | Binary Tree |
| 19 | 04/01/2021 | Binary Search Tree |
| 20 | 04/01/2021 | Sort Using Binary Search Tree |
| 21 | 11/01/2021 | DFS And BFS Graph Traversals |
| 22 | 10/02/2021 | Quick Sort and Merge Sort |
| 23 | 10/02/2021 | Heap Sort |
| 24 | 12/02/2021 | Hashing |

27/08/2020

**Experiment No:1**

**BUBBLE SORT**

**AIM:**

To perform bubble sort in an array and to arrange the elements of the

array in ascending order.

**DATA STRUCTURE USED:**

Arrays.

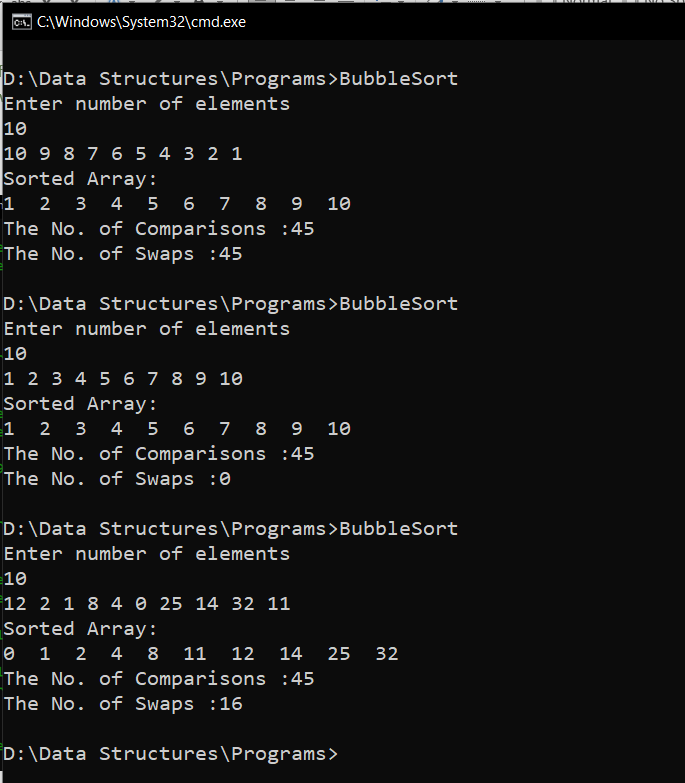
**ALGORITHM:**

1. Declare the array arr[size] and read the values of array
2. for i=0 to size-1
3. for j=0 to size-i-1
4. if arr[j]>arr[j+1]
5. swap arr[j] and arr[j+1]
6. endif
7. endfor
8. endfor
9. print the sorted array

**PROGRAM:**

#include <stdio.h>  
void main(){  
 int arr[100], i,j, n,temp,s=0,c=0;  
  
 printf("Enter number of elements\n");  
 scanf("%d", &n);  
  
 for (i=0; i<n; i++){  
 scanf("%d", &arr[i]);  
 }  
 for (i=0 ; i<n-1;i++){  
 for (j=0 ; j<n-i-1;j++){  
 c++;  
 if (arr[j]>arr[j+1]){  
 temp=arr[j];  
 arr[j]=arr[j+1];  
 arr[j+1]=temp;  
 s++;  
 }  
 }  
 }  
  
 printf("Sorted Array:\n");  
 for (i=0;i< n;i++){  
 printf("%d ", arr[i]);  
 }  
 printf("\nThe No. of Comparisons :%d\n", c);  
 printf("The No. of Swaps :%d\n", s);  
}

**OUTPUT:**



**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(n2)

Worst case – O(n2)

27/08/2020

**Experiment No:2**

**SELECTION SORT**

**AIM:**

To perform selection sort in an array and to arrange the elements of the array in

ascending order.

**DATA STRUCTURE USED:**

Arrays.

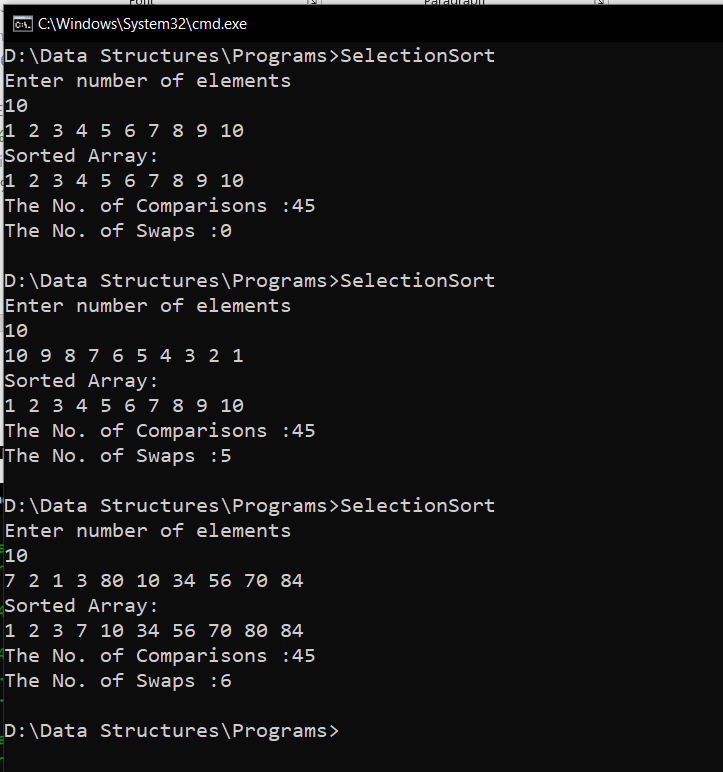
**ALGORITHM:**

1. Read the elements of the array arr[size]
2. for i=0 to size
3. pos=i
4. for j= i+1 to size
5. if arr[pos]>arr[j]
6. pos=j
7. end if
8. endfor
9. if pos !=j
10. swap arr[j] and arr[pos]
11. endif
12. endfor

**PROGRAM:**

#include <stdio.h>  
void main(){  
 int arr[100], n,i, j, pos,c=0,s=0,temp;  
  
 printf("Enter number of elements\n");  
 scanf("%d",&n);  
 for (i=0;i< n;i++){  
 scanf("%d", &arr[i]);  
 }  
 for (i=0;i<n-1;i++){  
 pos = i;  
 for (j=i+1;j<n;j++){ c++;  
 if (arr[pos]> arr[j])  
 pos=j;  
 }  
 if (pos!=i){  
 temp = arr[i];  
 arr[i] = arr[pos];  
 arr[pos] = temp;  
 s++;  
 }  
 }  
  
 printf("Sorted Array:\n");  
  
 for (i=0;i<n; i++){  
 printf("%d ",arr[i]);  
 }  
 printf("\nThe No. of Comparisons :%d\n", c);  
 printf("The No. of Swaps :%d\n", s);  
 return 0;  
}

**OUTPUT:**



**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(n2)

Average case – O(n2)

Worst case – O(n2)

06/09/2020

**Experiment No:3**

**INSERTION SORT**

**AIM:**

To perform insertion sort in an array and to arrange the elements of the

array in ascending order.

**DATA STRUCTURE USED:**

Arrays.

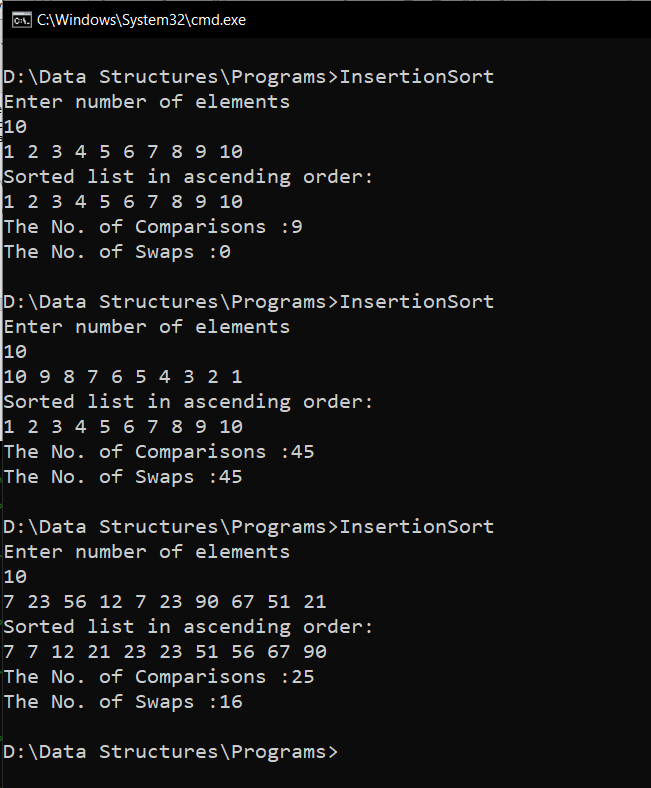
**ALGORITHM:**

1. Read the elements of the array arr[size]
2. for i=1 to size
3. x = arr[i]
4. for j= i-1 to 0
5. if arr[j]>x
6. arr[j+1]=x
7. else
8. break
9. end if
10. endfor
11. arr[i+1]=x
12. endfor

**PROGRAM:**

#include <stdio.h>  
void main(){  
 int n, arr[100], i, j, x,c=0,s=0, flag = 0;  
 printf("Enter number of elements\n");  
 scanf("%d", &n);  
  
 for (i = 0; i < n; i++){  
 scanf("%d", &arr[i]);  
 }  
  
  
 for (i = 1 ; i < n ; i++){  
 x = arr[i];  
 for (j = i - 1 ; j >= 0; j--){  
 c++;  
 if (arr[j] > x) {  
 s++;  
 arr[j+1] = arr[j];  
 flag = 1;  
 }  
 else  
 break;  
  
 }  
  
 if (flag==1)  
 arr[j+1] = x;  
 }  
  
 printf("Sorted list in ascending order:\n");  
  
 for (i = 0; i < n ; i++) {  
 printf("%d ", arr[i]);  
 }  
 printf("\nThe No. of Comparisons :%d\n", c);  
 printf("The No. of Swaps :%d\n", s);  
}

**OUTPUT:**



**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(n2)

Worst case – O(n2)

18/09/2020

**Experiment No:4**

**POLYNOMIAL ADDITTION**

**AIM:**

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

**DATA STRUCTURES USED:**

Arrays

**ALGORITHM:**

1. Initialise the exponent(row 0) and coefficient(row 1) and t1 (no of terms in a), t2 (no of terms in b)
2. Read the first polynomial and store it in the a 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

c.exp[k] = b.exp[j]

c.coeff[k] = b.coeff[j]

j++, k++

else if j >= t2

c.exp[k] = a.exp[i]

c.coeff[k] = a.coeff[i]

i++, k++

else if a.exp[i] == b.exp[j]

c.coeff[k] = a.coeff[i] + b.coeff[j]

c.exp[k] = a.exp[i]

i++, j++, k++

else if a.exp[i] > b.exp[j]

c.exp[k] = a.exp[i]

c.coeff[k] =a.coeff[i]

i++, k++

else

c.exp[k] = b.exp[j]

c.coeff[k] = b.coeff[j]

j++, k++

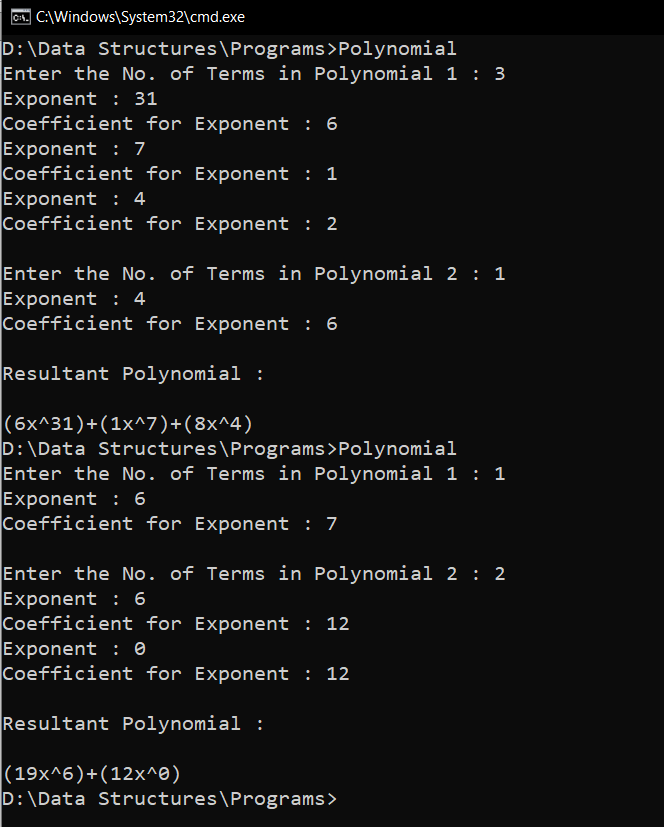
1. print p3

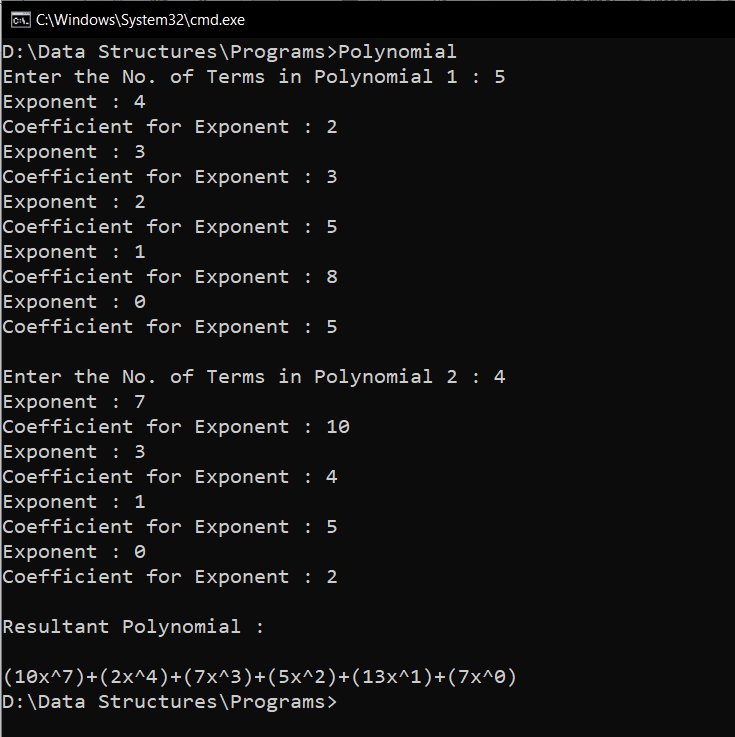
**PROGRAM:**

#include<stdio.h>  
void main(){  
 int a[2][10],b[2][10],c[2][10],i,j,k,t1,t2;  
 printf("Enter the No. of Terms in Polynomial 1 : ");  
 scanf("%d",&t1);  
 for(i=0;i<t1;i++){  
 printf("Exponent : ");  
 scanf("%d",&a[0][i]);  
 printf("Coefficient for Exponent : ");  
 scanf("%d",&a[1][i]);  
 }  
 printf("\nEnter the No. of Terms in Polynomial 2 : ");  
 scanf("%d",&t2);  
 for(i=0;i<t2;i++){  
 printf("Exponent : ");  
 scanf("%d",&b[0][i]);  
 printf("Coefficient for Exponent : ");  
 scanf("%d",&b[1][i]);  
 }  
 i=0;j=0;k=0;  
 while(i<t1 || j<t2){  
 if (i>=t1){  
 c[0][k] = b[0][j];  
 c[1][k] = b[1][j];  
 j++, k++;  
 }  
 else if (j>=t2){  
 c[0][k] = a[0][j];  
 c[1][k] = a[1][j];  
 i++, k++;  
 }  
 else if (a[0][i]==b[0][j]){  
 c[0][k]=a[0][i];  
 c[1][k]=a[1][i]+b[1][j];  
 i++, j++, k++;  
 }  
 else if(a[0][i]>b[0][j]){  
 c[0][k]=a[0][i];  
 c[1][k]=a[1][i];  
 i++;  
 k++;  
 }  
 else{  
 c[0][k]=b[0][j];  
 c[1][k]=b[1][j];  
 k++;  
 j++;  
 }  
 }  
 printf("\nResultant Polynomial :\n\n");  
 for(i=0;i<k;i++){  
 printf("(%dx^%d)",c[1][i],c[0][i]);  
 if(i!=k-1)  
 printf("+");  
 }

}

**OUTPUT:**





**RESULT:**

Two polynomials are stored in an array and are added to obtain a resultant polynomial.

All three polynomials are displayed.

20/09/2020

**Experiment No. 5**

**SPARSE MATRIX**

**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 STRUCTURES USED:**

Arrays

**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 ith row of sp1 to Resultant, i++, k++

29. Else if sp1[i][0] > sp2[j][0]

30. Copy the data of jth row of sp2 to Resultant, j++, k++

31. Else

32. If sp1[i][1] < sp2[j][1]

33. Copy the data of ith row of sp1 to Resultant, i++, k++

34. Else if sp1[i][1] > sp2[j][1]

35. Copy the data of jth 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

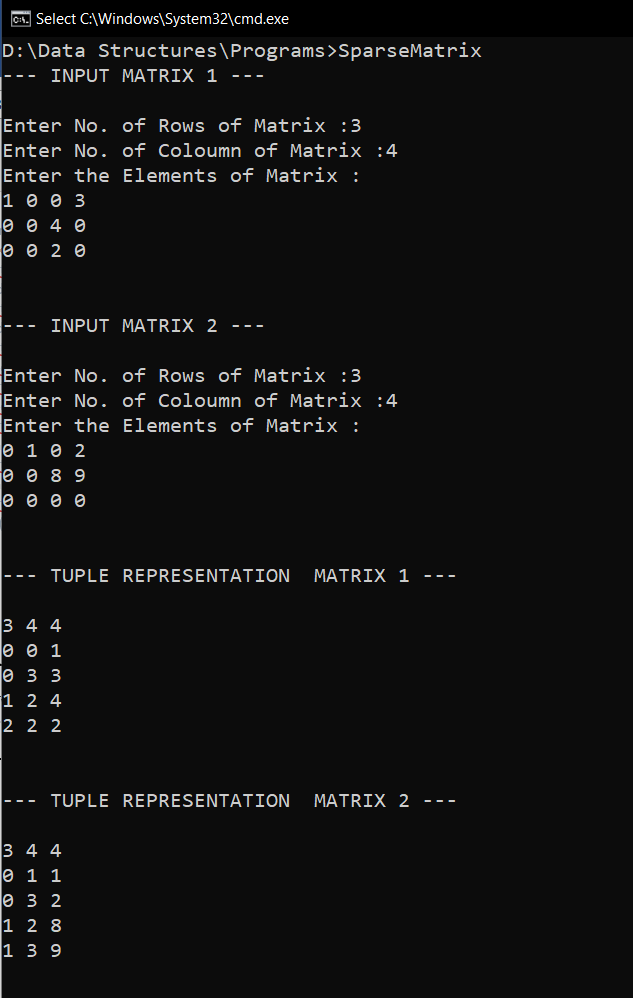
42. Print the Resultant Tuple Representation

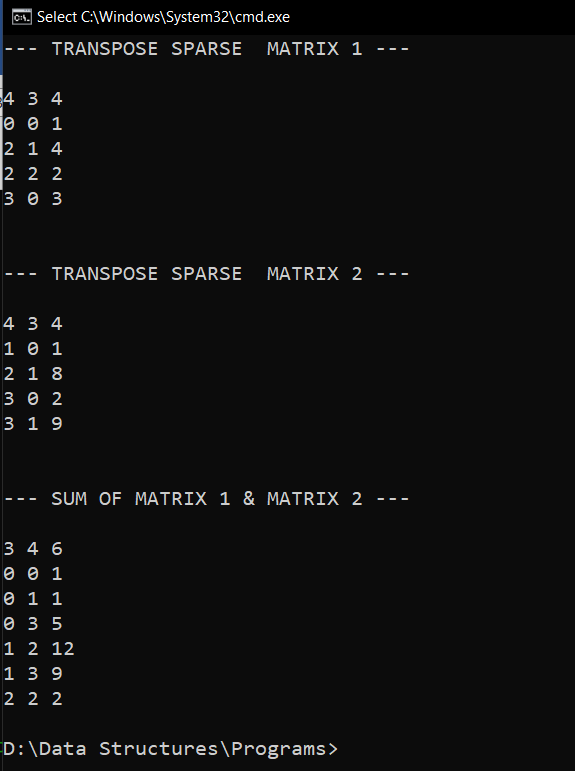
STOP

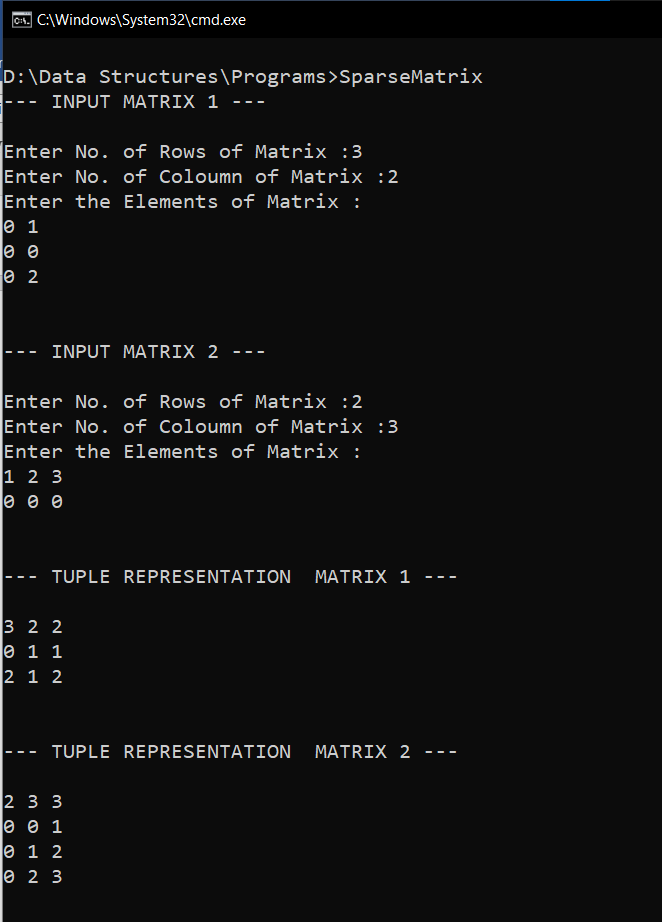
**PROGRAM:**

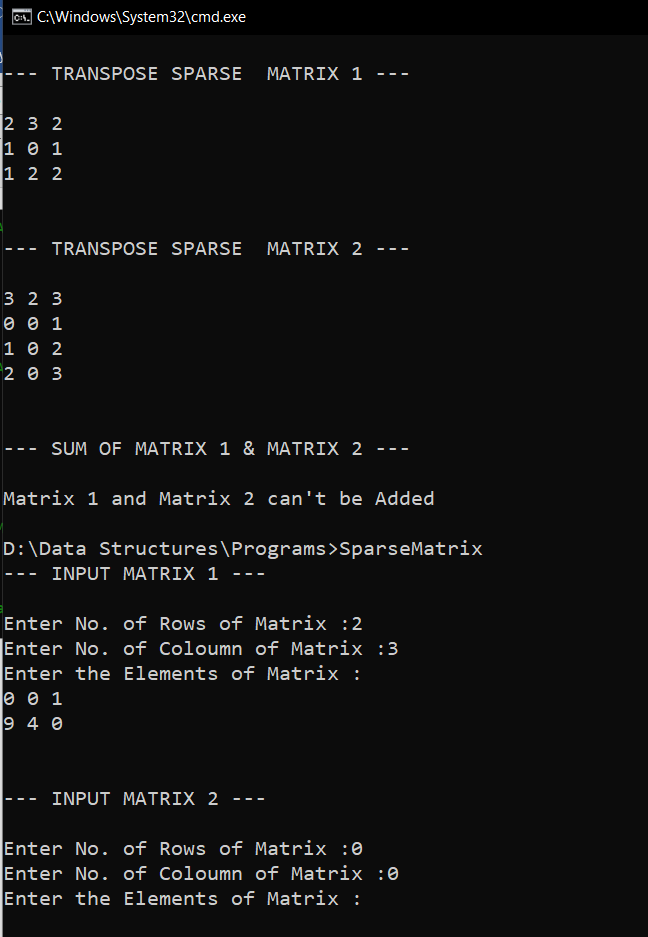
#include <stdio.h>  
struct sparse{  
 int row, col;  
 int arr[10][10];  
 int sarr[50][3];  
 int tarr[50][3];  
};  
  
  
void read(struct sparse \*sp){  
 printf("Enter No. of Rows of Matrix :");  
 scanf("%d", &sp->row);  
 printf("Enter No. of Coloumn of Matrix :");  
 scanf("%d",&sp->col);  
 printf("Enter the Elements of Matrix :\n");  
 for(int i=0;i<sp->row;i++){  
 for(int j=0;j<sp->col;j++){  
 scanf("%d", &sp->arr[i][j]);  
 }  
 }  
}  
  
void tupleRepresentation(struct sparse \*sp){  
 int k=0;  
 sp->sarr[0][0] = sp->row;  
 sp->sarr[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->sarr[k][0] = i;  
 sp->sarr[k][1] = j;  
 sp->sarr[k][2] = sp->arr[i][j];  
 }  
 }  
 }  
 sp->sarr[0][2] = k;  
 for(int i=0;i<=sp->sarr[0][2];i++) {  
 printf("%d ", sp->sarr[i][0]);  
 printf("%d ", sp->sarr[i][1]);  
 printf("%d \n", sp->sarr[i][2]);  
 }  
}  
  
  
  
void transpose(struct sparse \*sp){  
  
 if(sp->sarr[0][2] == 0){  
 printf("Matrix Cannot be Transposed\n");  
 }  
 else{  
 sp->tarr[0][0] = sp->sarr[0][1];  
 sp->tarr[0][1] = sp->sarr[0][0];  
 sp->tarr[0][2] = sp->sarr[0][2];  
 int k=1;  
 for(int i=0;i<sp->sarr[0][1];i++){  
 for(int j=1;j<=sp->sarr[0][2];j++){  
 if(i == sp->sarr[j][1]){  
 sp->tarr[k][0] = sp->sarr[j][1];  
 sp->tarr[k][1] = sp->sarr[j][0];  
 sp->tarr[k][2] = sp->sarr[j][2];  
 k++;  
 }  
 }  
 }  
 for(int i=0;i<=sp->tarr[0][2];i++) {  
 printf("%d ", sp->tarr[i][0]);  
 printf("%d ", sp->tarr[i][1]);  
 printf("%d \n", sp->tarr[i][2]);  
 }  
 }  
  
}  
  
  
void add(struct sparse \*sp1, struct sparse \*sp2, struct sparse \*sp3){  
 int i=1, j=1, k=1;  
 if(sp1->sarr[0][0]!=sp2->sarr[0][0]||sp1->sarr[0][1]!= sp2->sarr[0][1]){  
 printf("Matrix 1 and Matrix 2 can't be Added\n");  
 }  
else{  
 while(i<=sp1->sarr[0][2]||j<=sp2->sarr[0][2]){  
 if(sp1->sarr[i][0]==sp2->sarr[j][0]){  
 if(sp1->sarr[i][1]==sp2->sarr[j][1]){  
 sp3->sarr[k][2]=sp1->sarr[i][2]+sp2->sarr[j][2];  
 sp3->sarr[k][1] = sp1->sarr[i][1];  
 sp3->sarr[k][0] = sp1->sarr[i][0];  
 k++, i++, j++;  
 }  
 else if(sp1->sarr[i][1] < sp2->sarr[j][1]){  
 sp3->sarr[k][0] = sp1->sarr[i][0];  
 sp3->sarr[k][1] = sp1->sarr[i][1];  
 sp3->sarr[k][2] = sp1->sarr[i][2];  
 k++, i++;  
 }  
 else{  
 sp3->sarr[k][0] = sp2->sarr[j][0];  
 sp3->sarr[k][1] = sp2->sarr[j][1];  
 sp3->sarr[k][2] = sp2->sarr[j][2];  
 k++, j++;  
 }  
 }  
 else if(sp1->sarr[i][0] < sp2->sarr[j][0])  
 {  
 sp3->sarr[k][0] = sp1->sarr[i][0];  
 sp3->sarr[k][1] = sp1->sarr[i][1];  
 sp3->sarr[k][2] = sp1->sarr[i][2];  
 k++, i++;  
 }  
 else{  
 sp3->sarr[k][0] = sp2->sarr[j][0];  
 sp3->sarr[k][1] = sp2->sarr[j][1];  
 sp3->sarr[k][2] = sp2->sarr[j][2];  
 k++, j++;  
 }  
 }  
  
 sp3->sarr[0][0] = sp1->sarr[0][0];  
 sp3->sarr[0][1] = sp1->sarr[0][1];  
 sp3->sarr[0][2] = k-1;  
 for(int i=0;i<=sp3->sarr[0][2];i++) {  
 printf("%d ", sp3->sarr[i][0]);  
 printf("%d ", sp3->sarr[i][1]);  
 printf("%d \n", sp3->sarr[i][2]);  
 }  
 }  
}  
  
  
void main(){  
 struct sparse sp1, sp2,sp3;  
 printf("--- INPUT MATRIX 1 ---\n\n");  
 read(&sp1);  
 printf("\n\n--- INPUT MATRIX 2 ---\n\n");  
 read(&sp2);  
 printf("\n\n--- TUPLE REPRESENTATION MATRIX 1 ---\n\n");  
 tupleRepresentation(&sp1);  
 printf("\n\n--- TUPLE REPRESENTATION MATRIX 2 ---\n\n");  
 tupleRepresentation(&sp2);  
 printf("\n\n--- TRANSPOSE SPARSE MATRIX 1 ---\n\n");  
 transpose(&sp1);  
 printf("\n\n--- TRANSPOSE SPARSE MATRIX 2 ---\n\n");  
 transpose(&sp2);  
 printf("\n\n--- SUM OF MATRIX 1 & MATRIX 2 ---\n\n" );  
 add(&sp1, &sp2, &sp3);  
}

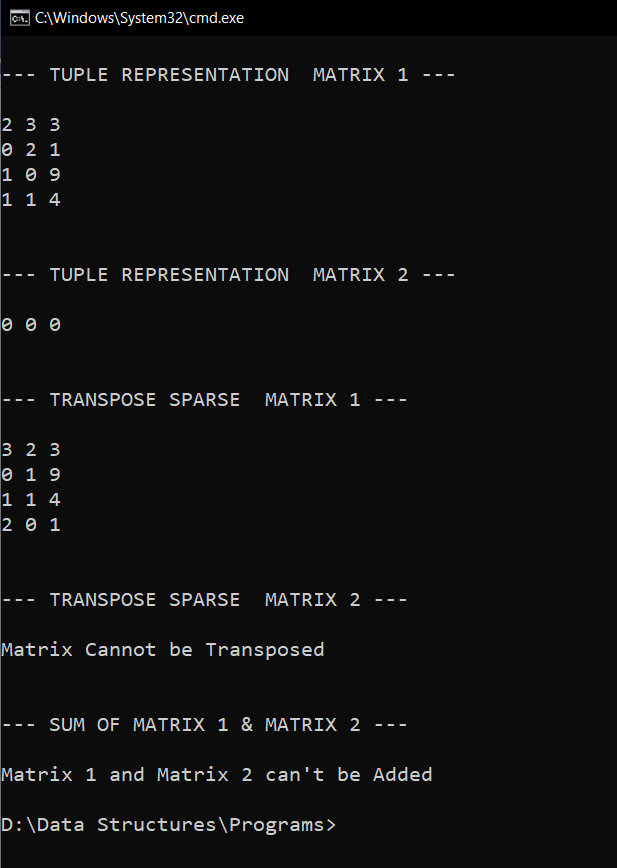
**OUTPUT:**











**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.

20/09/2020

**Experiment No:6**

**STACK USING ARRAYS**

**AIM:**

Implement a Stack using arrays with the operations:

1.Pushing elements to the Stack.

2.Popping elements from the Stack

3.Display the contents of the Stack after each operation.

**DATA STRUCTURES USED:**

Stack

**ALGORITHM:**

Algorithm Push(x)

1. if top=size-1
2. print “stack overflow”
3. else
4. arrr[++top]=x
5. endif

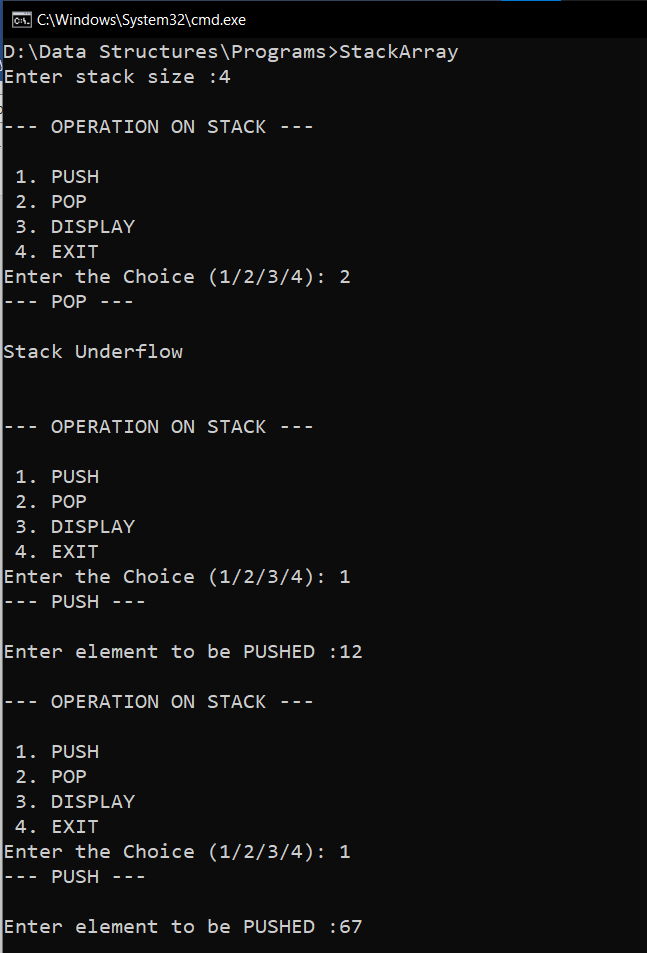
Algorithm Pop()

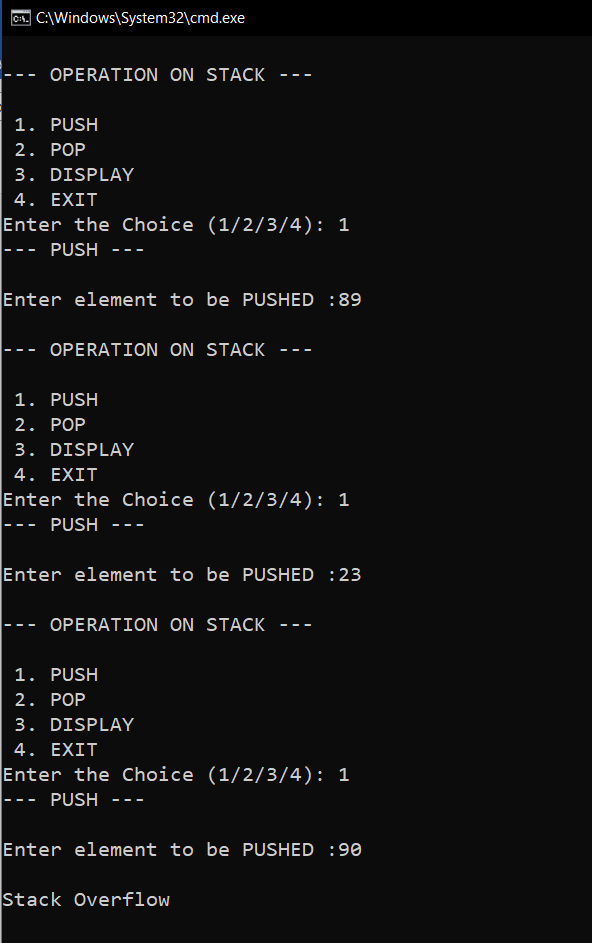
1. if top = -1
2. print “stack is empty”
3. else
4. item =arr[top]
5. top—
6. endif

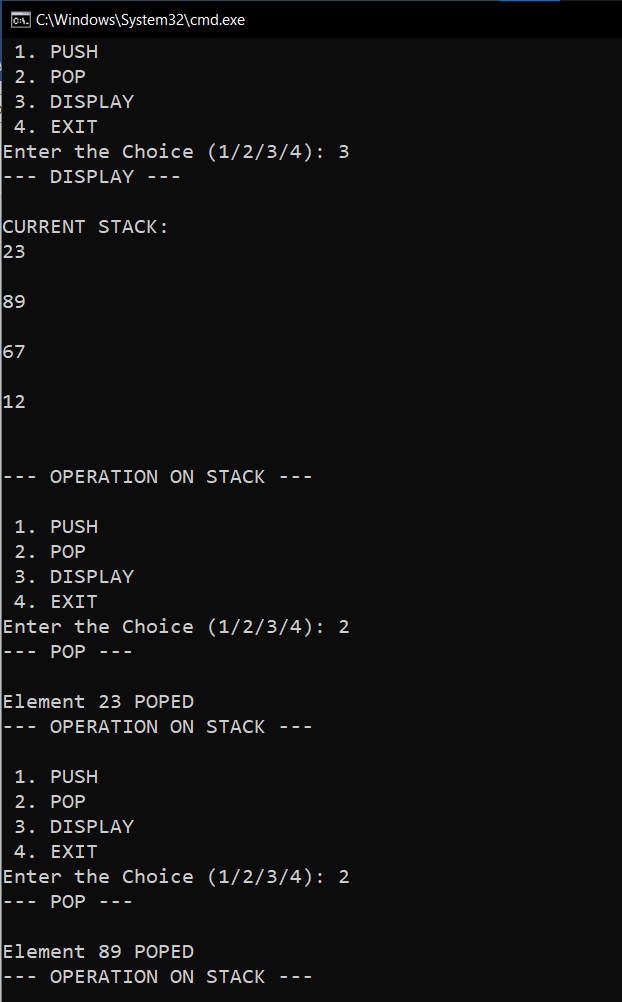
**PROGRAM:**

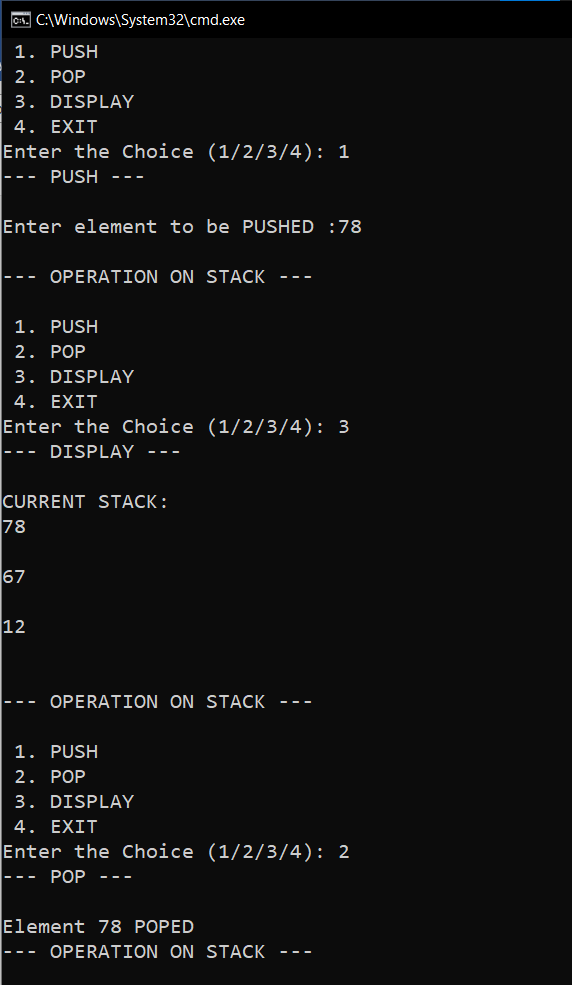
#include <stdio.h>  
#include <stdlib.h>  
struct stack{  
 int size;  
 int top;  
 int \*arr;  
};  
  
int isFull(struct stack \*st){  
 if(st->top >= st->size-1){  
 return 1;  
 }  
 return 0;  
}  
  
int isEmpty(struct stack \*st){  
 if(st->top == -1){  
 return 1;  
 }  
 return 0;  
}  
  
void push(struct stack \*st,int pushitem){  
 if(isFull(st)){  
 printf("\nStack Overflow\n\n");  
 }  
 else{  
 st->arr[++st->top] = pushitem;  
 }  
}  
int pop(struct stack \*st){  
 if(isEmpty(st)){  
 printf("\nStack Underflow\n\n");  
 }  
 else{  
 int popitem = st->arr[st->top];  
 st->top--;  
 return popitem;  
 }  
}  
  
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,x,y;  
 char ans='y';  
 printf("Enter stack size :");  
 scanf("%d", &st.size);  
 st.arr = (int\*) malloc (st.size \* sizeof(int));  
 st.top = -1;  
 while(ans=='y'){  
 printf("\n--- OPERATION ON STACK --- \n\n");  
 printf(" 1. PUSH \n");  
 printf(" 2. POP\n");  
 printf(" 3. DISPLAY\n");  
 printf(" 4. EXIT\n");  
 printf("Enter the Choice (1/2/3/4): ");  
 scanf("%d",&n);  
 switch(n){  
 case 1:printf("--- PUSH ---\n");  
 printf("\nEnter element to be PUSHED :");  
 scanf("%d", &x);  
 push(&st,x);  
 break;  
 case 2:printf("--- POP ---\n");  
 y=pop(&st);  
 printf("\nElement %d POPED ",y);  
 break;  
 case 3:printf("--- DISPLAY ---\n");  
 display(&st);  
 break;  
 case 4:ans='n';  
 break;  
 default:printf("Enter a Valid Input\n");  
 }  
 }  
}

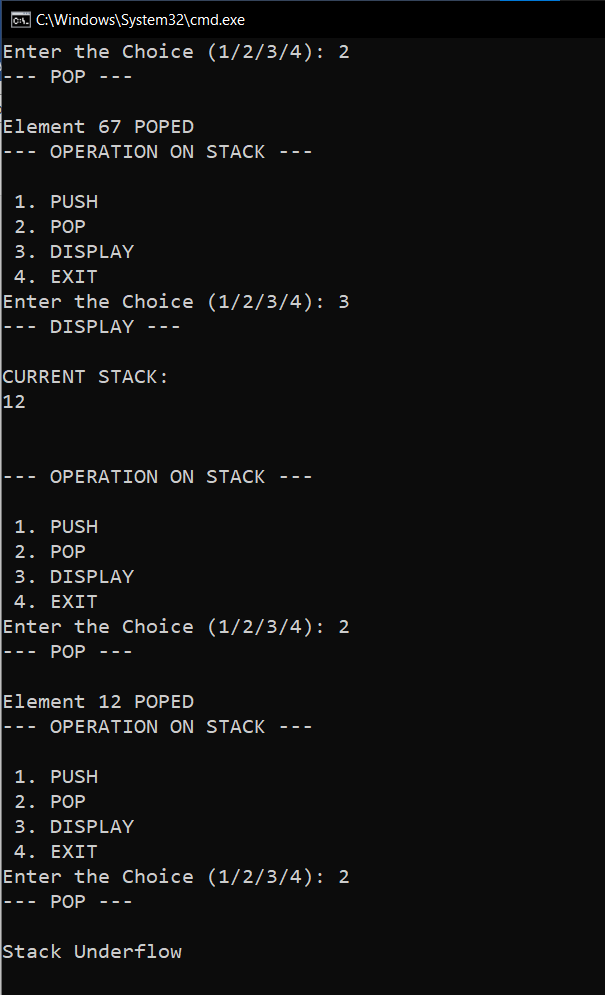
**OUTPUT:**











**RESULT:**

A Stack data structure is implemented using an array. Push(), Pop() and Display() operations were performed on it.

24/10/2020

**Experiment No:7**

**INFIX TO POSTFIX AND EVALUATION**

**AIM:**

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

**DATA STRUCTURES USED:**

Stack

**ALGORITHM:**

Algorithm INFIX\_TO\_POSTFIX()

START

1. TOP = -1, push(‘(‘)

2. While TOP > -1 do

3. ITEM = Readsymbol()

4. X = pop()

5. Case : ITEM = Operand

6. push(X)

7. print ITEM

8. Case : ITEM = ‘)’

9. While X != ‘(’

10. print X

11. X = pop()

12. EndWhile

13. Case : ISP(X) >= ICP(ITEM)

14. While ISP(X) >= ICP(ITEM) do

15. print X

16. X = pop()

17. EndWhile

18. push(X)

19. push(ITEM)

20. Case : ISP(X) < ICP(ITEM)

21. push(X)

22. push(ITEM)

23. Otherwise :

24. Print “Invalid Expression”

25. EndWhile

STOP

Algorithm POSTFIX\_CONVERSION()

START

1. While (TOP >= -1) do

2. ITEM = Readsymbol()

3. Case : ITEM = Operand

4. push(ITEM)

5. Case : ITEM = Operator

6. x2 = pop()

7. x1 = pop()

8. x = Operation(x1, x2, ITEM)

9. push(x)

10. Otherwise :

11. Print “Invalid Expression”

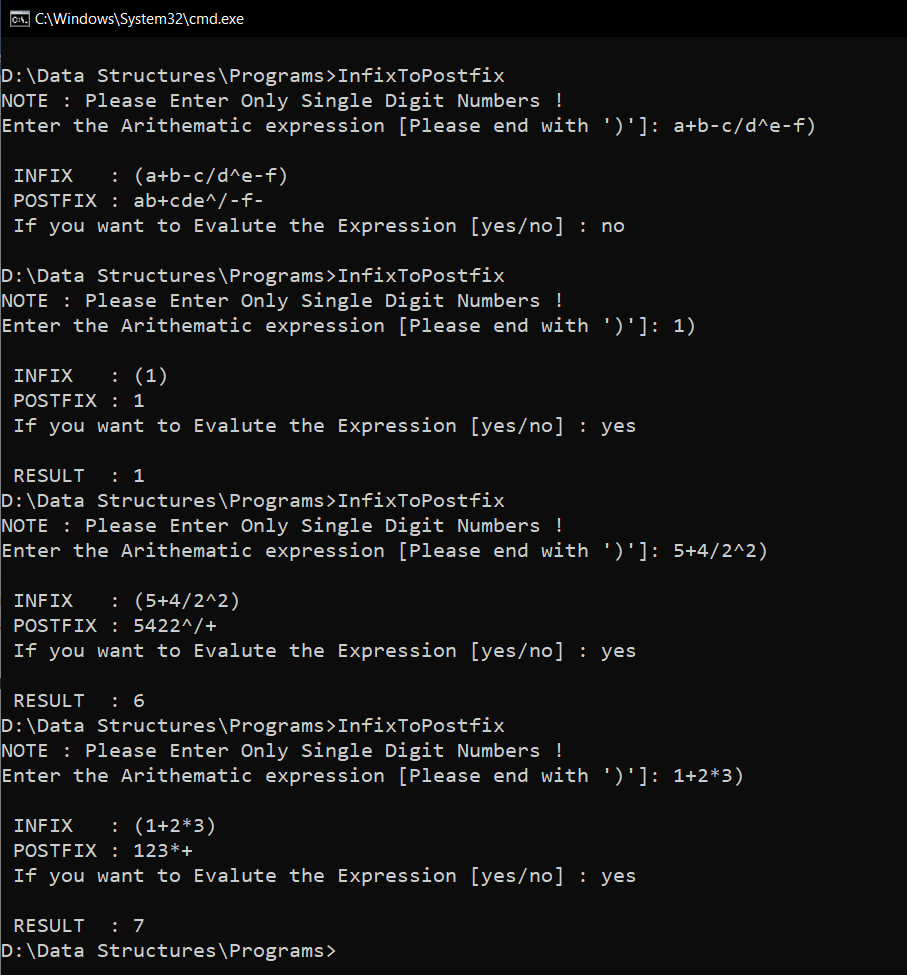
12. EndWhile

STOP

**PROGRAM:**

#include<stdio.h>  
#include<ctype.h>  
#include<math.h>  
#include<string.h>  
  
char stack[50];  
int top = -1;  
void push(char x){  
 stack[++top] = x;  
}  
  
char pop(){  
 if(top == -1)  
 return -1;  
 else  
 return stack[top--];  
}  
  
int Istack[50];  
int Itop = -1;  
  
void Ipush(int x){  
 Istack[++Itop] = x;  
}  
  
int Ipop(){  
 if(Itop == -1)  
 return 0;  
 else  
 return Istack[Itop--];  
}  
  
int ISP(char y){  
 if(y == '(')  
 return 0;  
 if(y == '+' || y == '-')  
 return 1;  
 if(y == '\*' || y == '/')  
 return 4;  
 if(y =='^')  
 return 5;  
 return 0;  
}  
int ICP(char y){  
 if(y == '(')  
 return 0;  
 if(y == '+' || y == '-')  
 return 1;  
 if(y == '\*' || y == '/')  
 return 3;  
 if(y =='^')  
 return 6;  
 return 0;  
}  
  
void main()  
{  
 char input[100];  
 char postfix[100];  
 char \*p,\*t,x;  
 char ans[5]="no";  
 // INFIX TO POSTFIX CONVERSION  
  
 printf("NOTE : Please Enter Only Single Digit Numbers !\n");  
 printf("Enter the Arithematic expression [Please end with \')\']: ");  
 scanf("%s",input);  
 printf("\n");  
 p=input;  
 t=postfix;  
 push('(');  
 printf(" INFIX : (%s",input);  
 printf("\n POSTFIX : ");  
 while(top!=-1){  
 if(isalnum(\*p)){  
  
 printf("%c",\*p);  
 \*t=\*p;  
 t++;  
  
 }else{  
 x=pop();  
 if(\*p == '('){  
  
 push(\*p);  
  
 }else if(\*p == ')'){  
  
 while( x!= '('){  
 printf("%c",x);  
 \*t=x;  
 t++;  
 x=pop();  
 }  
  
 }else if(ISP(x)>= ICP(\*p)){  
  
 while(ISP(x)>=ICP(\*p)){  
 printf("%c",x);  
 \*t=x;  
 t++;  
 x=pop();  
 }  
 push(x);  
 push(\*p);  
  
 }else if(ISP(x) < ICP(\*p)){  
 push(x);  
 push(\*p);  
  
 }else{  
  
 printf("Invalid Expression");  
  
 }  
 }  
  
 p++;  
  
 }  
  
  
 \*t='\0';  
 printf("\n");  
  
  
 // POSTFIX EVALUATION  
 printf(" If you want to Evalute the Expression [yes/no] : ");  
 scanf("%s",ans);  
 if(strcmp(ans,"yes")==0){  
 t = postfix;  
 int a,b,c;  
 while(\*t!='\0'){  
 if(isdigit(\*t)){  
 Ipush(\*t-48);  
 }else{  
 b= Ipop();  
 a= Ipop();  
 switch(\*t)  
 {  
 case '+': c=a+b;  
  
 break;  
  
  
 case '-': c=a-b;  
  
 break;  
  
 case '\*': c=a\*b;  
  
 break;  
  
 case '/': c=a/b;  
  
 break;  
  
 case '^': c=pow(a,b);  
  
 break;  
 }  
  
 Ipush(c);  
 }  
 t++;  
 }  
 int result=Ipop();  
 printf("\n RESULT : %d",result);  
 }  
}

**OUTPUT:**



**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)

31/10/2020

**Experiment No:8**

**QUEUE USING ARRAYS**

**AIM:**

Write a program to implement Queue using arrays.

**DATA STRUCTURES USED:**

Queue

**ALGORITHM:**

Algorithm\_ENQUEUE (ITEM)

1. If (REAR=N-1) // N is the size of Queue
2. print “Queue is full”
3. Exit
4. Else
5. If (REAR =-1 && FRONT =-1)
6. FRONT=REAR=0
7. Queue[REAR]=ITEM
8. Else
9. Queue[++REAR]=ITEM
10. EndIf
11. EndIf

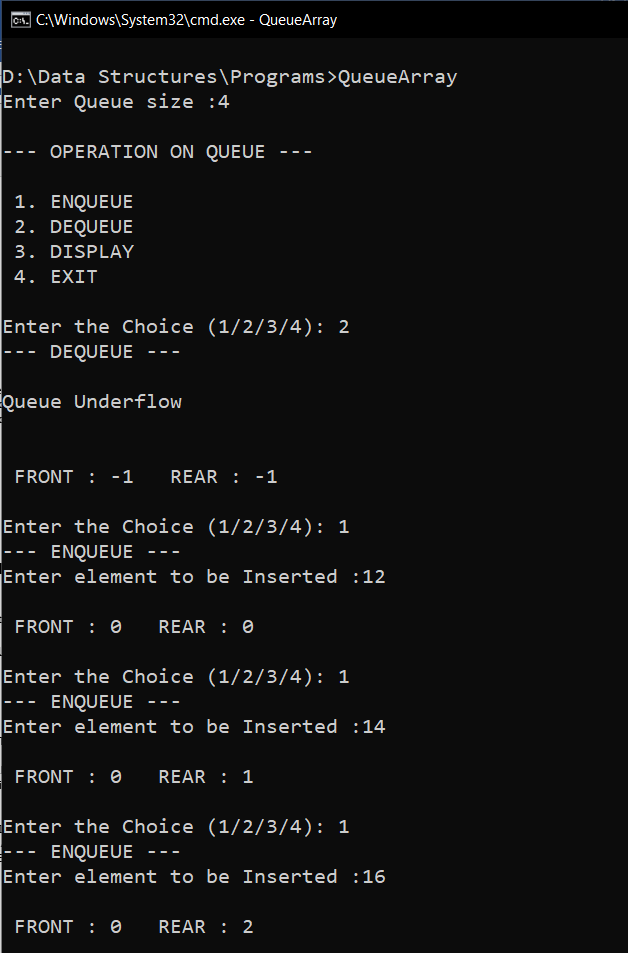
Algorithm\_DEQUEUE

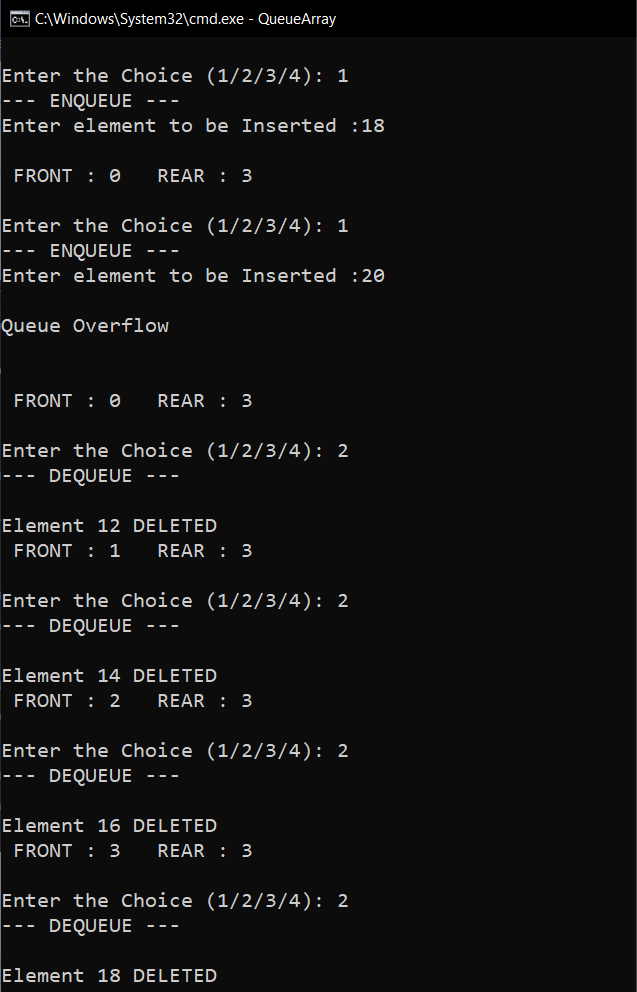
1. If (FRONT=-1)
2. Print “Queue is empty”
3. Exit
4. Else
5. ITEM = Queue[FRONT]
6. If (FRONT == REAR)
7. FRONT=REAR=-1
8. Else
9. FRONT++
10. EndIf
11. EndIf

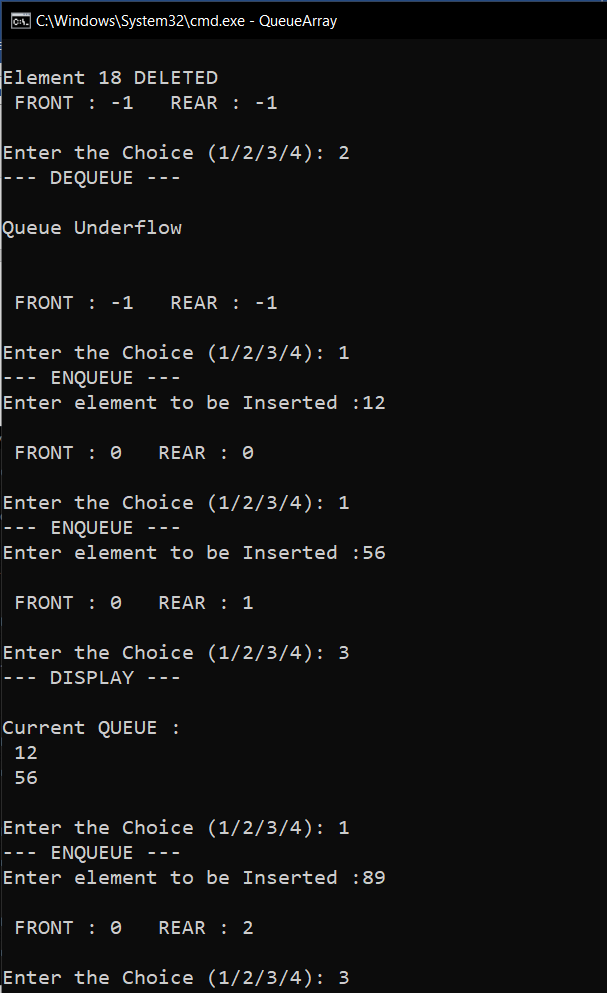
**PROGRAM:**

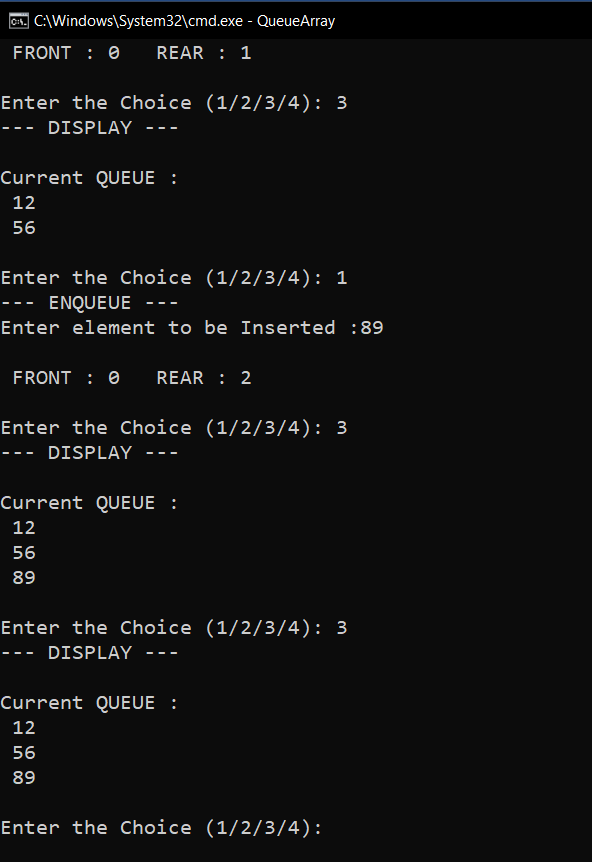
#include <stdio.h>  
#include <stdlib.h>  
int size;  
int front;  
int rear;  
int \*arr;  
  
int isFull(){  
 if(rear==size-1){  
 return 1;  
 }  
 return 0;  
}  
  
int isEmpty(){  
 if(rear==-1){  
 return 1;  
 }  
 return 0;  
}  
  
void insert(int item){  
 if(isFull()){  
 printf("\nQueue Overflow\n\n");  
 }  
 else if(front==-1){  
 arr[++rear] = item;  
 front++;  
 }  
 else {  
 arr[++rear] = item;  
 }  
 printf("\n FRONT : %d REAR : %d\n",front,rear);  
}  
void delete(){  
 if(isEmpty()){  
 printf("\nQueue Underflow\n\n");  
 }  
 else if(front==rear){  
 int item = arr[front];  
 printf("\nElement %d DELETED ",item);  
 front=-1;  
 rear=-1;  
  
 }else{  
 int item = arr[front];  
 front++;  
 printf("\nElement %d DELETED ",item);  
 }  
 printf("\n FRONT : %d REAR : %d\n",front,rear);  
}  
  
void display(){  
 printf("\nCurrent QUEUE :\n");  
 if(isEmpty()){  
 printf("\nQueue is Empty \n");  
 }else{  
 for(int i=front; i<=rear; i++){  
 printf(" %d \n",arr[i]);  
 }  
 }  
}  
  
void main(){  
 int n,x,y;  
 char ans='y';  
 printf("Enter Queue size :");  
 scanf("%d", &size);  
 arr = (int\*) malloc (size \* sizeof(int));  
 front=-1,rear=-1;  
 printf("\n--- OPERATION ON QUEUE --- \n\n");  
 printf(" 1. ENQUEUE \n");  
 printf(" 2. DEQUEUE\n");  
 printf(" 3. DISPLAY\n");  
 printf(" 4. EXIT\n");  
 while(ans=='y'){  
 printf("\nEnter the Choice (1/2/3/4): ");  
 scanf("%d",&n);  
 switch(n){  
 case 1:printf("--- ENQUEUE ---\n");  
 printf("Enter element to be Inserted :");  
 scanf("%d", &x);  
 insert(x);  
 break;  
 case 2:printf("--- DEQUEUE ---\n");  
 delete();  
 break;  
 case 3:printf("--- DISPLAY ---\n");  
 display();  
 break;  
 case 4:ans='n';  
 break;  
 default:printf("Enter a Valid Input\n");  
 }  
 }  
}

**OUTPUT:**









**RESULT:**

The Insert() and Delete() operation on Queue was implemented

Time complexity of Insert() operation is O(1).

Time complexity of Delete() operation is O(1).

03/11/2020

**Experiment No: 9**

**CIRCULAR QUEUE USING ARRAYS**

**AIM:**

Write a program to implement Circular Queue using arrays.

**DATA STRUCTURES USED:**

Queue

**ALGORITHM:**

Algorithm\_ENQUEUE (ITEM)

FRONT=-1 REAR=-1

1. If (FRONT==(REAR+1)%N) // N is the size of Queue
2. print “Queue is full”
3. Exit
4. Else
5. If (FRONT= =-1)
6. FRONT=REAR=0
7. CQueue[REAR]=ITEM
8. Else
9. REAR=(REAR+1)%N
10. CQueue[REAR]=ITEM
11. EndIf
12. EndIf

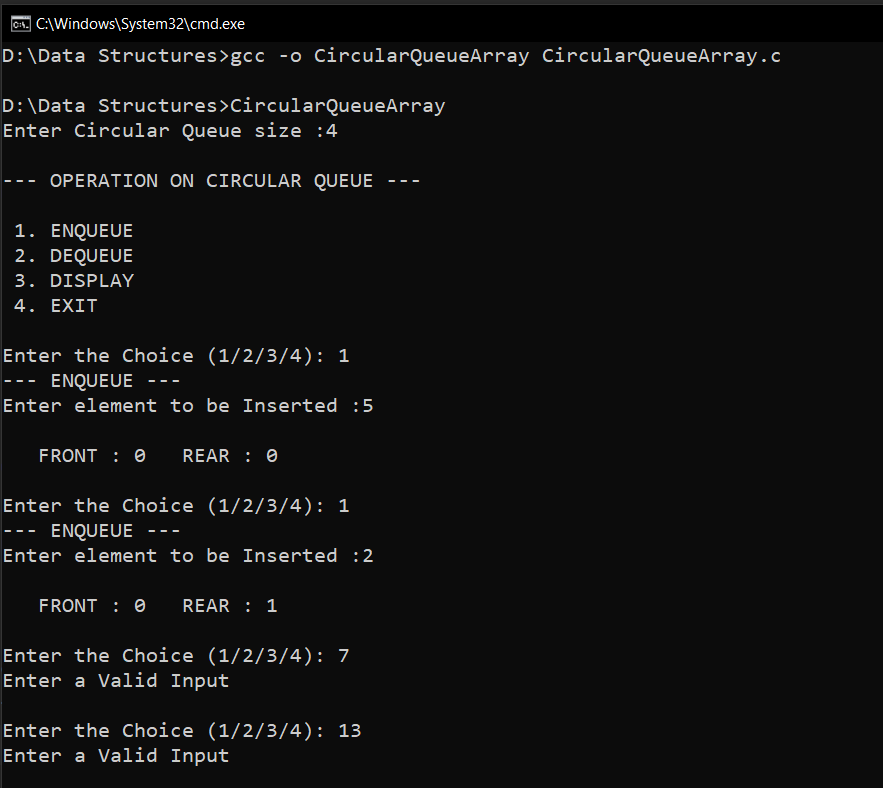
Algorithm\_DEQUEUE ()

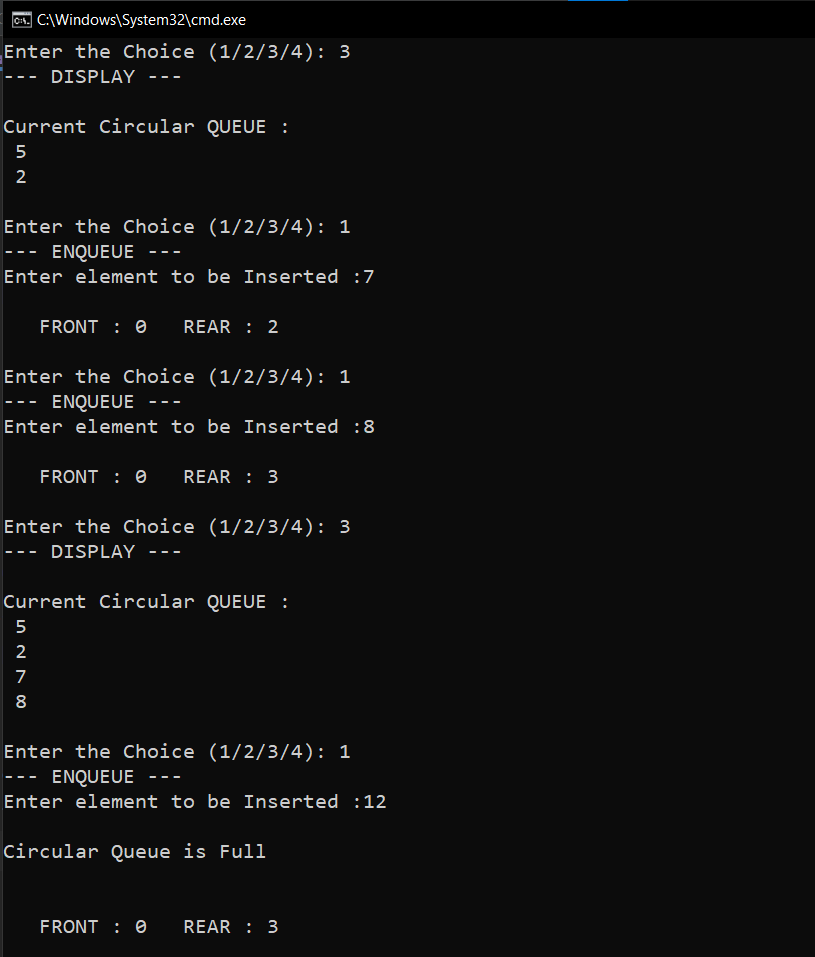
1. If (FRONT=-1)
2. Print “Queue is empty”
3. Exit
4. Else
5. If (FRONT == REAR)
6. ITEM = CQueue[FRONT]
7. FRONT=REAR=-1
8. Else
9. FRONT=(FRONT+1)%N
10. ITEM = CQueue[FRONT]
11. EndIf
12. EndIf

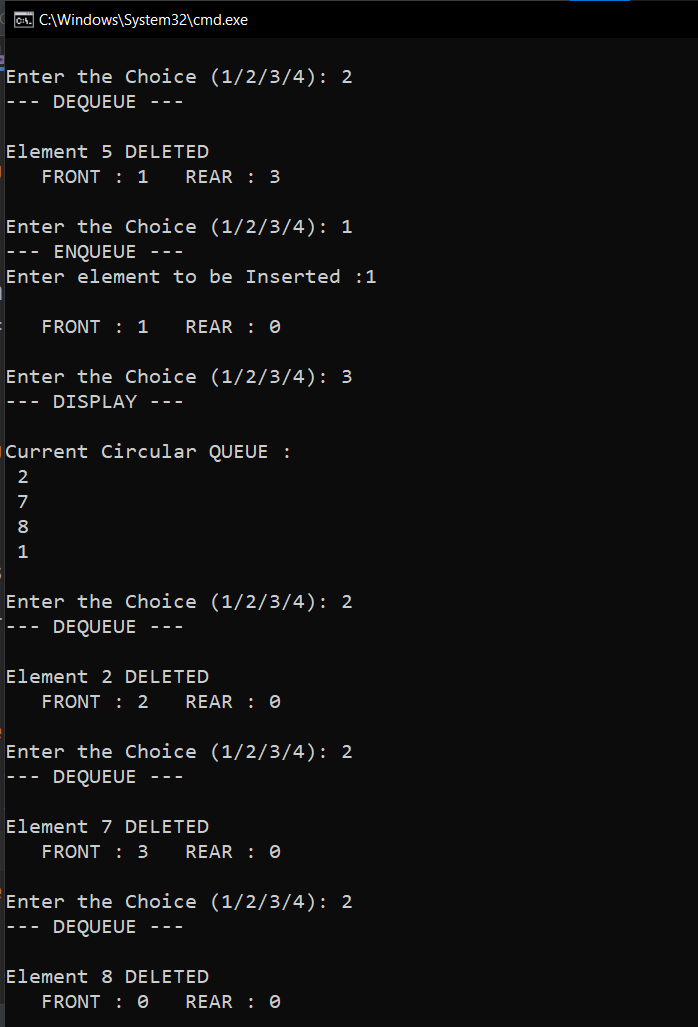
**PROGRAM:**

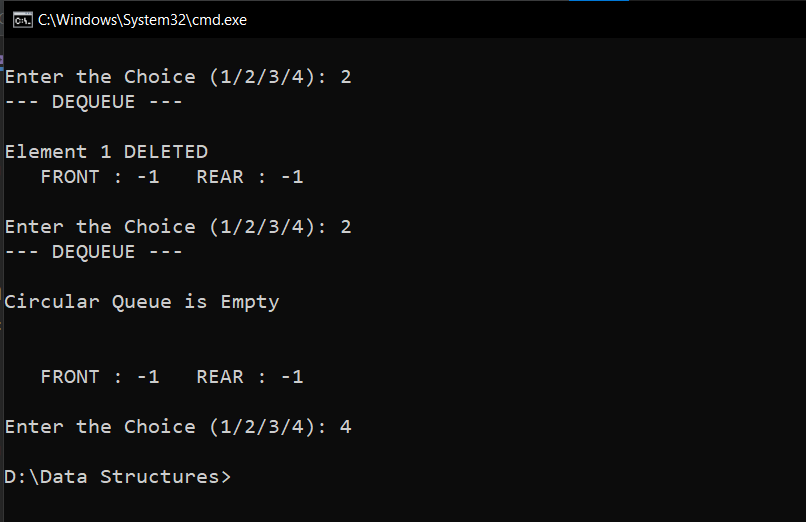
#include <stdio.h>  
#include <stdlib.h>  
int size;  
int front;  
int rear;  
int \*arr;  
  
int isFull(){  
 if(front==(rear+1)%size){  
 return 1;  
 }  
 return 0;  
}  
  
int isEmpty(){  
 if(front==-1){  
 return 1;  
 }  
 return 0;  
}  
  
void insert(int item){  
 if(isFull()){  
 printf("\nCircular Queue is Full\n\n");  
 }  
 else if(front==-1){  
 arr[++rear] = item;  
 front++;  
 }  
 else {  
 rear=(rear+1)%size;  
 arr[rear] = item;  
 }  
 printf("\n FRONT : %d REAR : %d \n",front,rear);  
}  
void delete(){  
 if(isEmpty()){  
 printf("\nCircular Queue is Empty\n\n");  
 }  
 else if(front==rear){  
 int item = arr[front];  
 printf("\nElement %d DELETED ",item);  
 front=-1;  
 rear=-1;  
  
 }else{  
 int item = arr[front];  
 front=(front+1)%size;  
 printf("\nElement %d DELETED ",item);  
 }  
 printf("\n FRONT : %d REAR : %d \n",front,rear);  
}  
  
void display(){  
 printf("\nCurrent Circular QUEUE :\n");  
 if(isEmpty()){  
 printf("\nCircular Queue is Empty \n");  
 }else if (rear >= front){  
 for(int i=front;i<=rear;i++){  
 printf(" %d\n",arr[i]);  
 }  
 }  
 else{  
 for(int i=front;i<size;i++){  
 printf(" %d\n",arr[i]);  
 }  
 for(int i=0;i<=rear;i++){  
 printf(" %d\n",arr[i]);  
 }  
 }  
}  
  
void main(){  
 int n,x,y;  
 char ans='y';  
 printf("Enter Circular Queue size :");  
 scanf("%d", &size);  
 arr = (int\*) malloc (size \* sizeof(int));  
 front=-1,rear=-1;  
 printf("\n--- OPERATION ON CIRCULAR QUEUE --- \n\n");  
 printf(" 1. ENQUEUE \n");  
 printf(" 2. DEQUEUE\n");  
 printf(" 3. DISPLAY\n");  
 printf(" 4. EXIT\n");  
 while(ans=='y'){  
 printf("\nEnter the Choice (1/2/3/4): ");  
 scanf("%d",&n);  
 switch(n){  
 case 1:printf("--- ENQUEUE ---\n");  
 printf("Enter element to be Inserted :");  
 scanf("%d", &x);  
 insert(x);  
 break;  
 case 2:printf("--- DEQUEUE ---\n");  
 delete();  
 break;  
 case 3:printf("--- DISPLAY ---\n");  
 display();  
 break;  
 case 4:ans='n';  
 break;  
 default:printf("Enter a Valid Input\n");  
 }  
 }  
}

**OUTPUT:**









**RESULT:**

The Enqueue() and Dequeue() operation on circular queue was implemented.

Time complexity of Enqueue() operation is O(1).

Time complexity of Dequeue() operation is O(1).

04/11/2020

**Experiment No: 10**

**PRIORITY QUEUE USING ARRAYS**

**AIM**:

Write a program to implement Priority Queue using arrays.

**DATA STRUCTURES USED:**

Queue

**ALGORITHM:**

Algorithm INSERT (ITEM, VALUE) // N is the size

1. If (REAR=N-1)
2. print “Queue is full”
3. Exit
4. Else
5. If (REAR =-1 && FRONT =-1)
6. FRONT=REAR=0
7. Queue[REAR]=ITEM
8. PRIORITY[REAR]=VALUE
9. Else
10. Queue[++REAR]=ITEM
11. PRIORITY[REAR]=VALUE
12. EndIf
13. EndIf

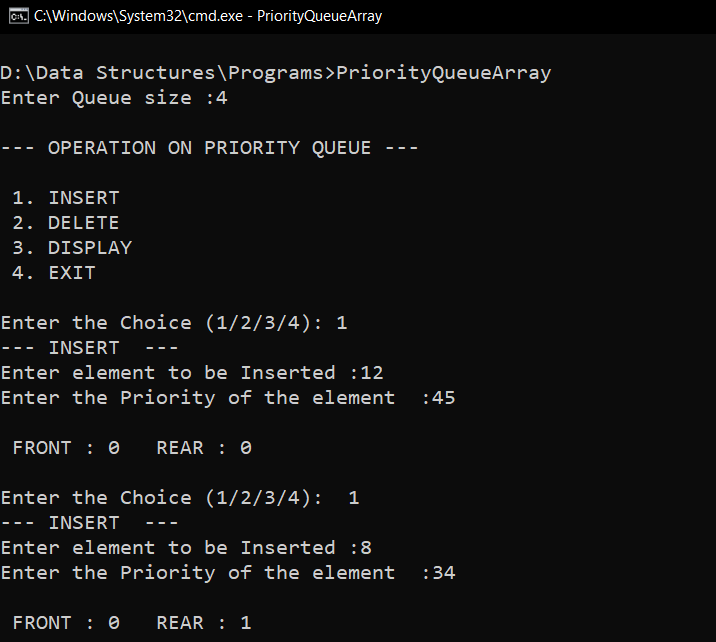
Algorithm DELETE

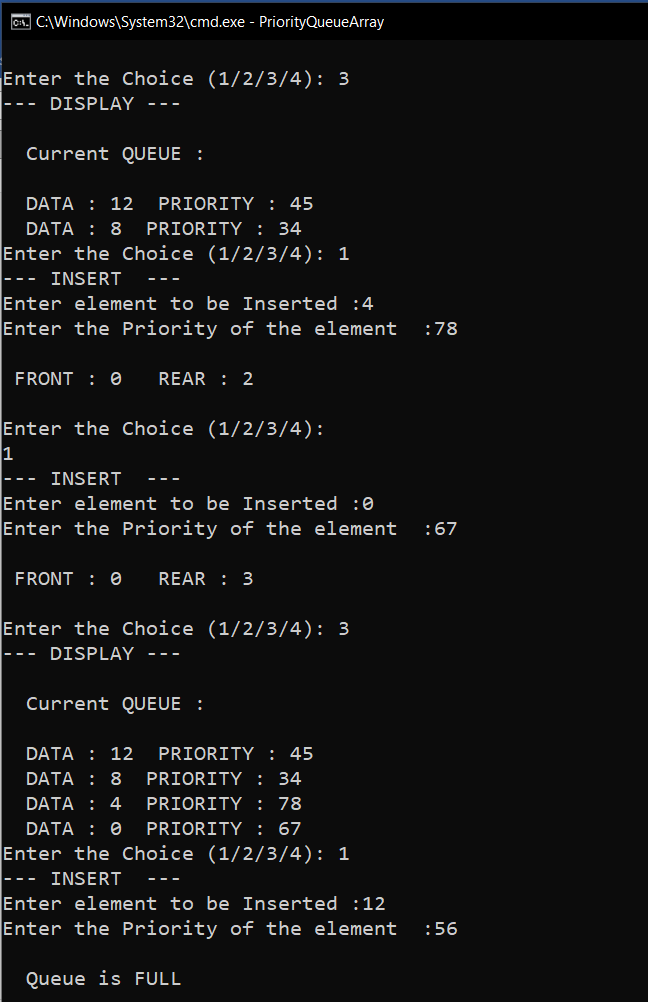
1. If (FRONT=-1)
2. Print “Queue is empty”
3. Exit
4. Else
5. If (FRONT == REAR)
6. ITEM = Queue[FRONT]
7. VALUE=PRIORITY[FRONT]
8. FRONT=REAR=-1
9. Else
10. SORT() // Sort According to the Priority(Descending Order)
11. VALUE=PRIORITY[FRONT]
12. ITEM = Queue[FRONT]
13. FRONT++
14. EndIf
15. EndIf

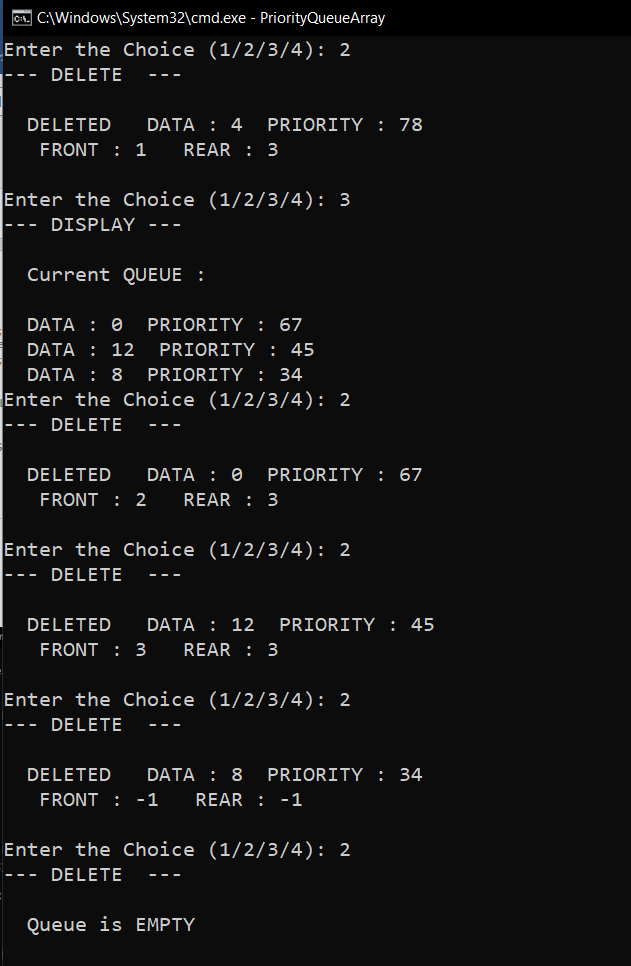
**PROGRAM:**

#include <stdio.h>  
#include <stdlib.h>  
int size;  
int front=-1;  
int rear=-1;  
int \*arr;  
int \*pty;  
  
void enqueue (int item,int priority){  
 if(rear==size-1){  
 printf("\n Queue is FULL\n\n");  
 }  
 else if(front==-1){  
 arr[++front] = item;  
 pty[front] = priority;  
 ++rear;  
 }  
 else {  
 arr[++rear] = item;  
 pty[rear] = priority;  
 }  
 printf("\n FRONT : %d REAR : %d \n",front,rear);  
  
}  
void dequeue(){  
 if(front==-1){  
 printf("\n Queue is EMPTY\n\n");  
 }  
 else if(front==rear){  
 int item = arr[front];  
 int priority = pty[front];  
 printf("\n DELETED DATA : %d PRIORITY : %d ",item,priority);  
 front=-1;  
 rear=-1;  
  
 }else{  
 //Sorting the Queue according to Priority  
 int i,j,n,temp1,temp2;  
 for (i=0 ; i<rear-front;i++){  
 for (j=0 ; j<rear-front-i;j++){  
 if (pty[j] < pty[j+1]){  
 temp1=pty[j];  
 pty[j]=pty[j+1];  
 pty[j+1]=temp1;  
 temp2=arr[j];  
 arr[j]=arr[j+1];  
 arr[j+1]=temp2;  
 }  
 }  
 }  
 int item = arr[front];  
 int priority = pty[front];  
 printf("\n DELETED DATA : %d PRIORITY : %d ",item,priority);  
 front++;  
 }  
 printf("\n FRONT : %d REAR : %d \n",front,rear);  
}  
  
void display(){  
 printf("\n Current QUEUE :\n");  
 if(front==-1){  
 printf("\n Queue is EMPTY \n");  
 }else{  
 for(int i=front; i<=rear; i++){  
 printf("\n DATA : %d PRIORITY : %d ",arr[i],pty[i]);  
 }  
 }  
}  
  
void main(){  
 int n,x,y;  
 char ans='y';  
 printf("Enter Queue size :");  
 scanf("%d", &size);  
 arr = (int\*) malloc (size \* sizeof(int));  
 pty = (int\*) malloc (size \* sizeof(int));  
 printf("\n--- OPERATION ON PRIORITY QUEUE --- \n\n");  
 printf(" 1. INSERT \n");  
 printf(" 2. DELETE\n");  
 printf(" 3. DISPLAY\n");  
 printf(" 4. EXIT\n");  
 while(ans=='y'){  
 printf("\nEnter the Choice (1/2/3/4): ");  
 scanf("%d",&n);  
 switch(n){  
 case 1:printf("--- INSERT ---\n");  
 printf("Enter element to be Inserted :");  
 scanf("%d", &x);  
 printf("Enter the Priority of the element :");  
 scanf("%d", &y);  
 enqueue(x,y);  
 break;  
 case 2:printf("--- DELETE ---\n");  
 dequeue();  
 break;  
 case 3:printf("--- DISPLAY ---\n");  
 display();  
 break;  
 case 4:ans='n';  
 break;  
 default:printf("Enter a Valid Input\n");  
 }  
 }  
}

**OUTPUT:**







**RESULT:**

The Priority Queue was successfully implemented and the required operations were carried out.

Time complexity of Insert() operation is O(1).

Time complexity of Delete() operation is O(n2).

04/11/2020

**Experiment No:11**

**DOUBLE ENDED QUEUE** **(DEQUE) USING ARRAYS**

**AIM:**

Write a program to implement Deque using arrays.

**DATA STRUCTURES USED:**

Queue

**ALGORITHM:**

Algorithm INSERT\_FRONT (ITEM)

1. If (FRONT==0) // N is the size of Queue
2. print “Insertion not possible”
3. Exit
4. Else
5. If (FRONT =-1)
6. FRONT=REAR=0
7. Queue[REAR]=ITEM
8. Else
9. Queue[--FRONT]=ITEM
10. EndIf
11. EndIf

Algorithm INSERT\_REAR (ITEM)

1. If (REAR=N-1)
2. print “Queue is full”
3. Exit
4. Else
5. If (REAR =-1 && FRONT =-1)
6. FRONT=REAR=0
7. Queue[REAR]=ITEM
8. Else
9. Queue[++REAR]=ITEM
10. EndIf
11. EndIf

Algorithm DELETE\_FRONT

1. If (FRONT=-1)
2. Print “Queue is empty”
3. Exit
4. Else
5. If (FRONT == REAR)
6. ITEM = Queue[FRONT]
7. FRONT=REAR=-1
8. Else
9. ITEM = Queue[FRONT]
10. FRONT++
11. EndIf
12. EndIf

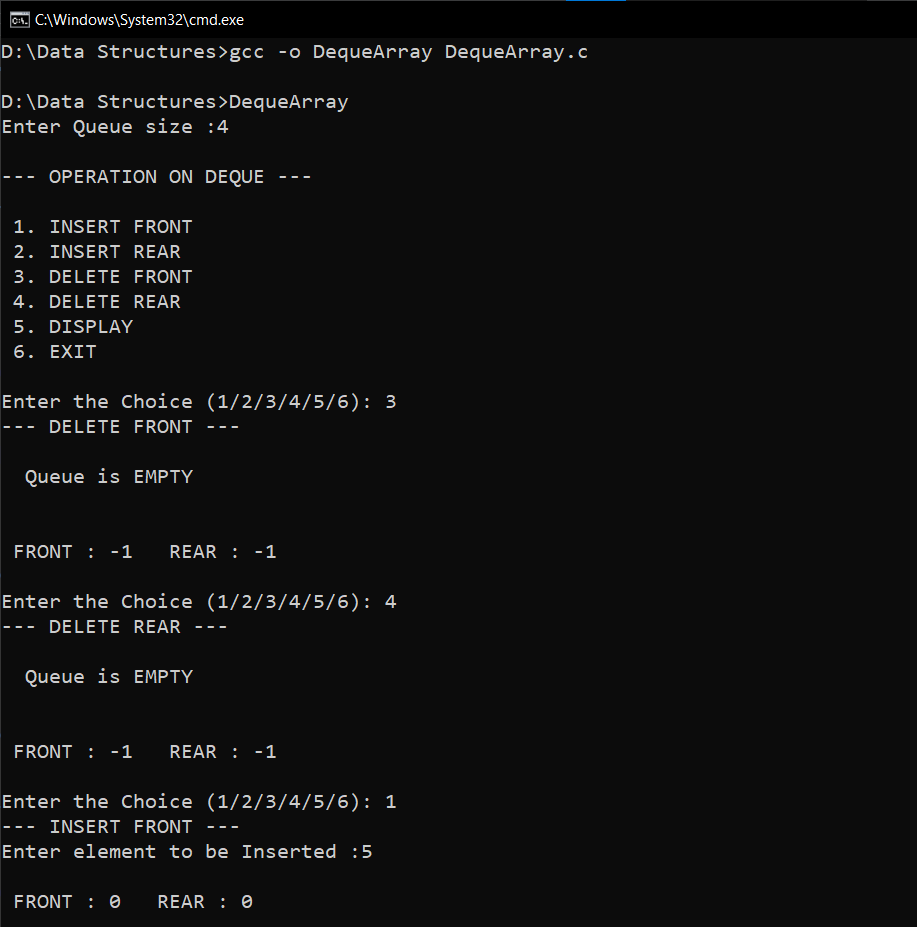
Algorithm DELETE\_REAR

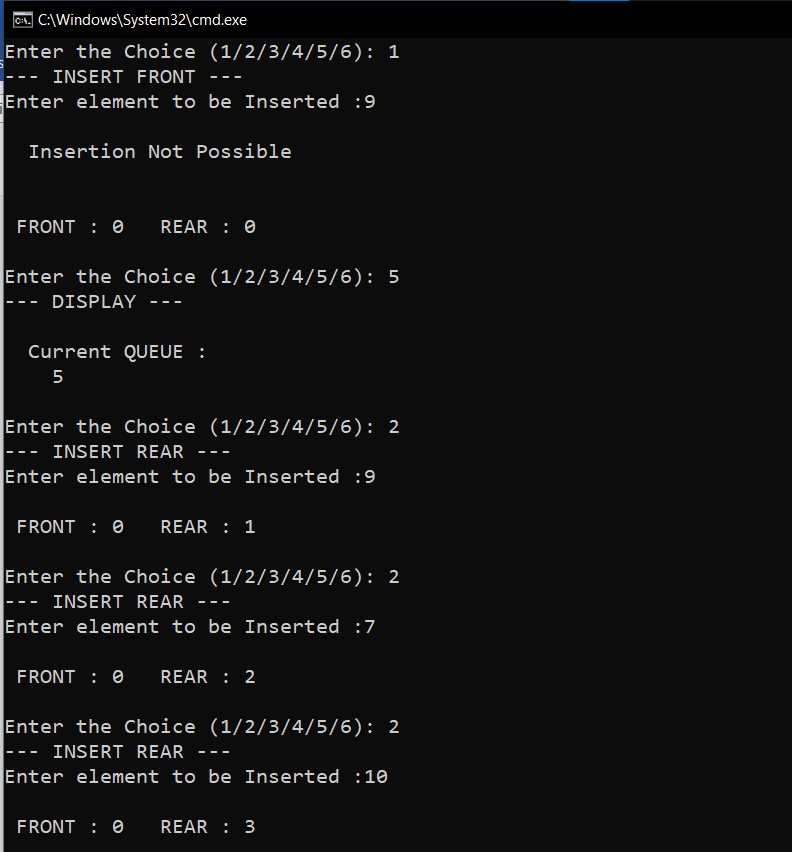
1. If (REAR=N-1)
2. Print “Deletion not possible”
3. Exit
4. Else
5. If (FRONT == REAR)
6. ITEM = Queue[FRONT]
7. FRONT=REAR=-1
8. Else
9. ITEM = Queue[FRONT]
10. REAR--
11. EndIf
12. EndIf

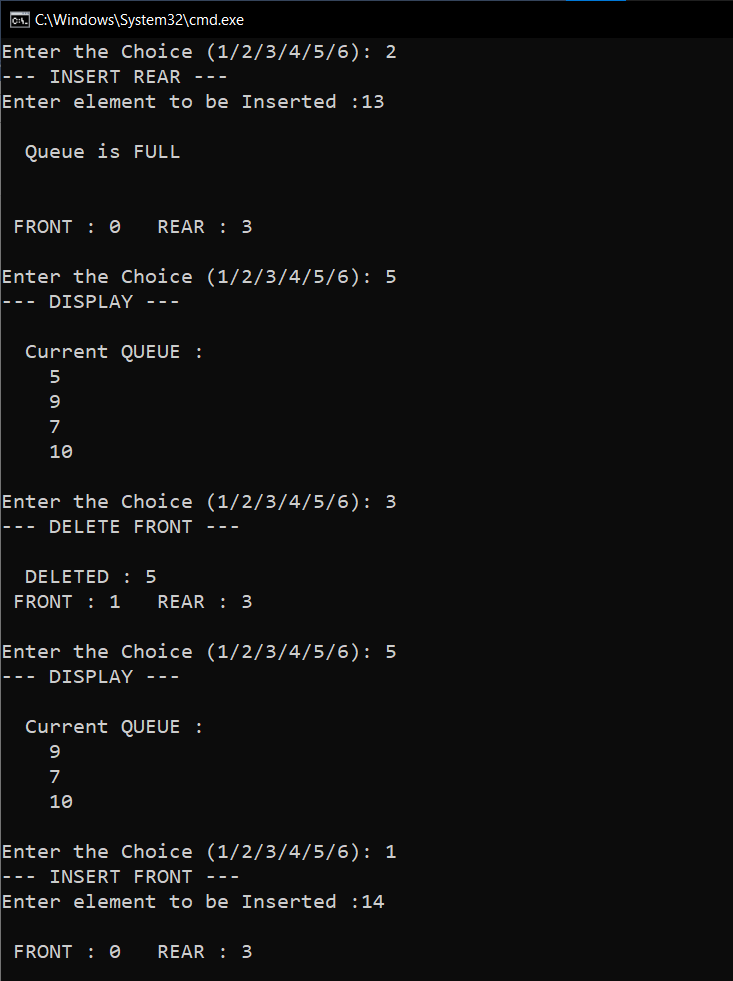
**PROGRAM:**

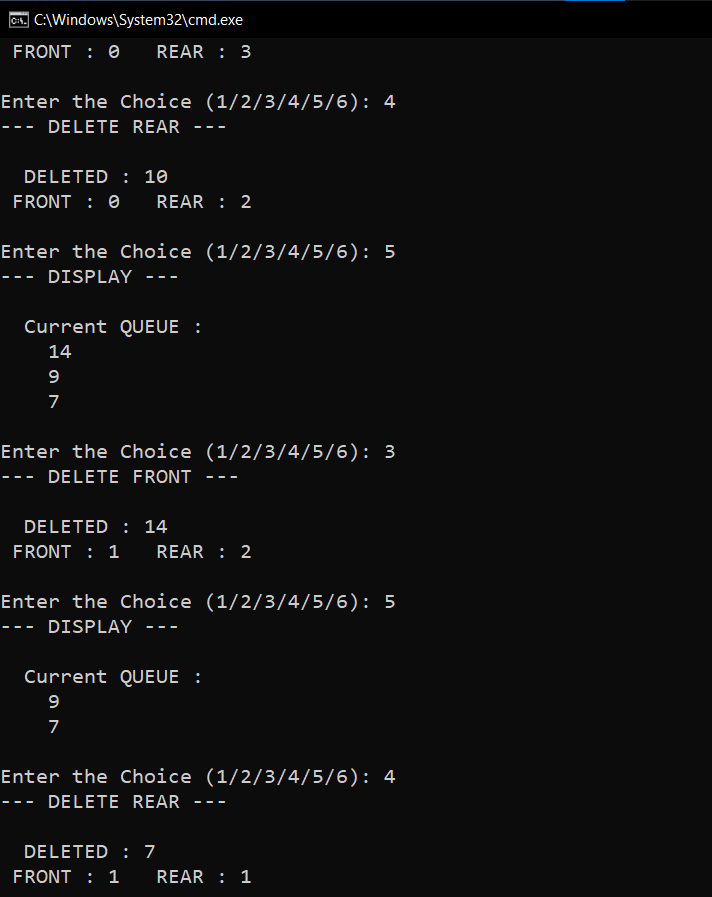
#include <stdio.h>  
#include <stdlib.h>  
int size;  
int front;  
int rear;  
int \*arr;  
  
void insertFront(int item){  
 if(front==0){  
 printf("\n Insertion Not Possible \n\n");  
 }  
 else if(front==-1){  
 arr[++front] = item;  
 rear++;  
 }  
 else {  
 arr[--front] = item;  
 }  
 printf("\n FRONT : %d REAR : %d \n",front,rear);  
}  
void insertRear(int item){  
 if(rear==size-1){  
 printf("\n Queue is FULL\n\n");  
 }  
 else if(front==-1){  
 arr[++front] = item;  
 ++rear;  
 }  
 else {  
 arr[++rear] = item;  
 }  
 printf("\n FRONT : %d REAR : %d \n",front,rear);  
}  
void deleteFront(){  
 if(front==-1){  
 printf("\n Queue is EMPTY\n\n");  
 }  
 else if(front==rear){  
 int item = arr[front];  
 printf("\n DELETED : %d ",item);  
 front=-1;  
 rear=-1;  
  
 }else{  
 int item = arr[front];  
 front++;  
 printf("\n DELETED : %d ",item);  
 }  
 printf("\n FRONT : %d REAR : %d \n",front,rear);  
}  
void deleteRear(){  
 if(rear==-1){  
 printf("\n Queue is EMPTY\n\n");  
 }  
 else if(front==rear){  
 int item = arr[front];  
 printf("\n DELETED : %d ",item);  
 front=-1;  
 rear=-1;  
  
 }else{  
 int item = arr[rear];  
 --rear;  
 printf("\n DELETED : %d ",item);  
 }  
 printf("\n FRONT : %d REAR : %d \n",front,rear);  
}  
void display(){  
 printf("\n Current QUEUE :\n");  
 if(front==-1){  
 printf("\n Queue is EMPTY \n");  
 }else{  
 for(int i=front; i<=rear; i++){  
 printf(" %d \n",arr[i]);  
 }  
 }  
}  
  
void main(){  
 int n,x,y;  
 char ans='y';  
 printf("Enter Queue size :");  
 scanf("%d", &size);  
 arr = (int\*) malloc (size \* sizeof(int));  
 front=-1,rear=-1;  
 printf("\n--- OPERATION ON DEQUE --- \n\n");  
 printf(" 1. INSERT FRONT \n");  
 printf(" 2. INSERT REAR\n");  
 printf(" 3. DELETE FRONT\n");  
 printf(" 4. DELETE REAR\n");  
 printf(" 5. DISPLAY\n");  
 printf(" 6. EXIT\n");  
 while(ans=='y'){  
 printf("\nEnter the Choice (1/2/3/4/5/6): ");  
 scanf("%d",&n);  
 switch(n){  
 case 1:printf("--- INSERT FRONT ---\n");  
 printf("Enter element to be Inserted :");  
 scanf("%d", &x);  
 insertFront(x);  
 break;  
 case 2:printf("--- INSERT REAR ---\n");  
 printf("Enter element to be Inserted :");  
 scanf("%d", &x);  
 insertRear(x);  
 break;  
 case 3:printf("--- DELETE FRONT ---\n");  
 deleteFront();  
 break;  
 case 4:printf("--- DELETE REAR ---\n");  
 deleteRear();  
 break;  
 case 5:printf("--- DISPLAY ---\n");  
 display();  
 break;  
 case 6:ans='n';  
 break;  
 default:printf("Enter a Valid Input\n");  
 }  
 }  
}

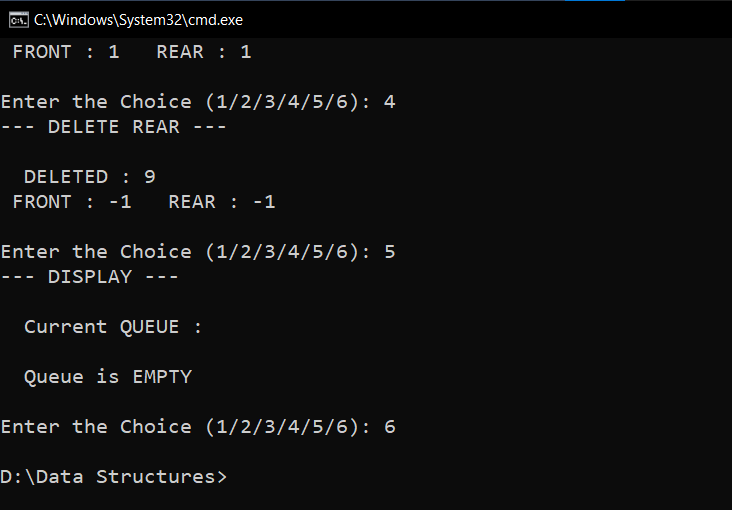
**OUTPUT:**











**RESULT:**

The Deque was successfully implemented and the required output was obtained.

Time complexity of INSERT\_REAR() operation is O(1).

Time complexity of DELETE\_FRONT() operation is O(1).

Time complexity of INSERT\_FRONT() operation is O(1).

Time complexity of DELETE\_REAR() operation is O(1).

13/11/2020

**Experiment No:12**

**LINKED LISTS**

**AIM:**

Write a program to implement operations on Linked List.

**DATA STRUCTURES USED:**

Linked List

**ALGORITHM:**

Algorithm INSERT\_FRONT (ITEM)

1. new = GetNode(Node)
2. If (new = NULL) then
3. Print ”memory underflow”
4. Exit
5. Else
6. new->LINK=HEADER->LINK
7. new-> DATA=ITEM
8. HEADER->LINK=new
9. Endif
10. stop

Algorithm INSERT\_REAR (ITEM)

1. new= GetNodes(Node)
2. if (new = NULL) then
3. print”memory underflow”
4. Exit
5. Else
6. ptr=HEADER
7. While(ptr-> LINK!=NULL)do
8. ptr=ptr->LINK
9. Endwhile
10. ptr->LINK= new
11. new->DATA=ITEM
12. new->LINK=NULL
13. Stop

Algorithm INSERT\_ANY (ITEM,KEY)

1. new= GetNode(Node)
2. if (new = NULL) then
3. print ”memory underflow”
4. Exit
5. Else
6. ptr=HEADER
7. While(ptr->DATA!=KEY) and(ptr-> LINK != NULL)do
8. ptr=ptr->LINK
9. Endwhile
10. If(ptr->LINK=NULL)
11. Print”KEY NOT FOUND”
12. Exit
13. Else
14. new-> LINK=ptr->LINK
15. new->DATA=x
16. ptr->LINK=new
17. Endif
18. Endif
19. stop

Algorithm DELETE\_FRONT

1. ptr=HEADER->LINK
2. if(ptr=NULL)then
3. print “The list is empty”
4. Exit
5. Else
6. HEADER->LINK=ptr->LINK
7. ReturnNode(ptr)
8. stop

Algorithm DELETE\_REAR

1. ptr=HEADER
2. if (ptr->LINK =NULL)then
3. print”the list is empty”
4. exit
5. else
6. while(ptr->LINK!=NULL)
7. ptr1=ptr
8. ptr=ptr->LINK
9. endwhile
10. ptr->LINK=NULL
11. ReturnNode(ptr)
12. Endif
13. Stop

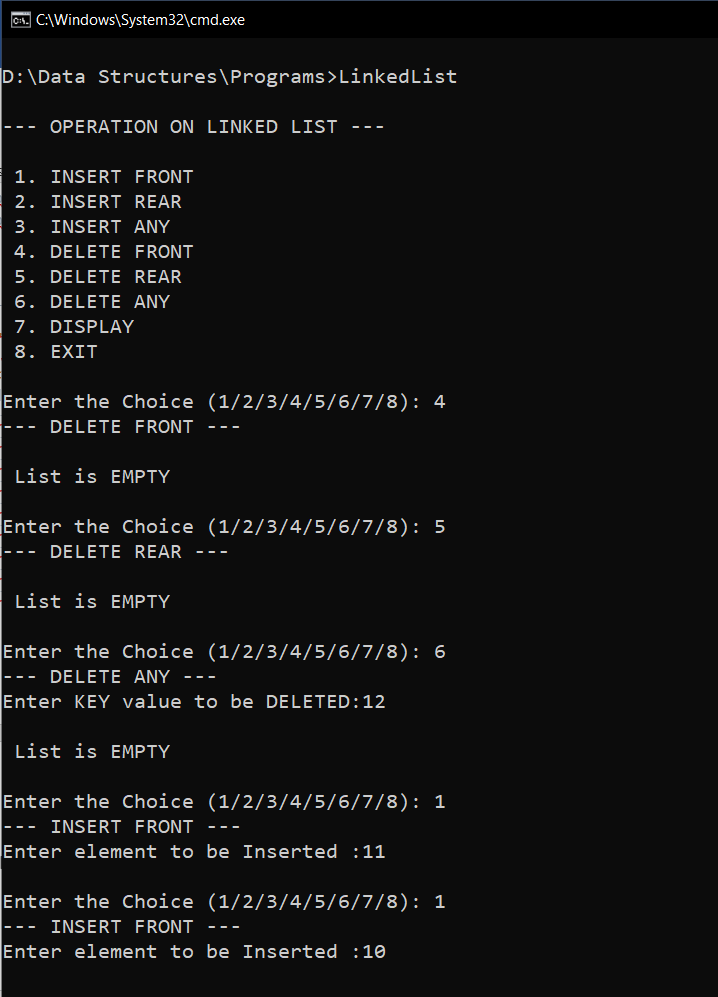
Algorithm DELETE\_ANY

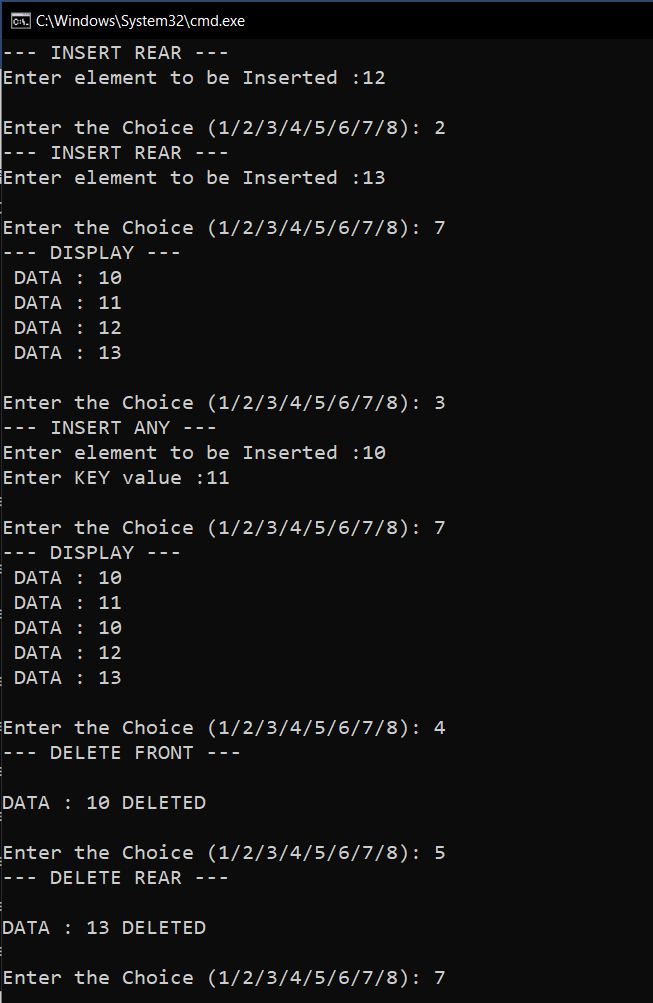
1. ptr1=HEADER
2. ptr=ptr1->LINK
3. while(ptr!=NULL)
4. if(ptr->DATA!= KEY)
5. ptr1=ptr
6. ptr=ptr->LINK
7. else
8. ptr1->LINK=ptr->LINK
9. ReturnNode(ptr)
10. Exit
11. Endif
12. Endwhile
13. If ptr=NULL
14. Print”Node with key doesn’t exist”
15. Endif
16. Stop

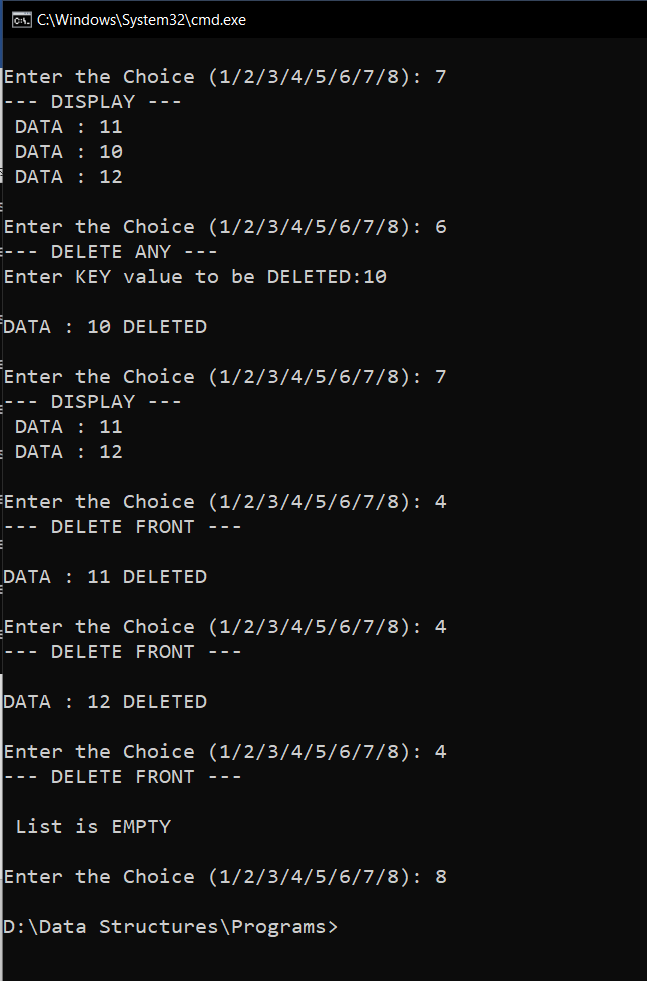
**PROGRAM:**

#include<stdio.h>  
#include<stdlib.h>  
struct node{  
 int data;  
 struct node \*link;  
};  
  
void insert\_front(struct node\* header,int x){  
 struct node\* new = (struct node\*)malloc(sizeof(struct node));  
 new->data=x;  
 new->link=NULL;  
 if(new==NULL){  
 printf("\nMEMORY Underflow\n");  
 }else{  
 if(header->link==NULL){  
 header->link=new;  
 }else{  
 new->link=header->link;  
 header->link=new;  
 }  
 }  
}  
  
void insert\_rear(struct node\* header,int x){  
 struct node\* new = (struct node\*)malloc(sizeof(struct node));  
 new->data=x;  
 new->link=NULL;  
 if(new==NULL){  
 printf("\nMEMORY Underflow\n");  
 }else{  
 if(header->link==NULL){  
 header->link=new;  
 }else{  
 struct node\* ptr=header;  
 while(ptr->link!=NULL){  
 ptr=ptr->link;  
 }  
 ptr->link=new;  
 }  
 }  
}  
void insert\_any(struct node\* header,int x,int key){  
 struct node\* new = (struct node\*)malloc(sizeof(struct node));  
 new->data=x;  
 new->link=NULL;  
 if(new==NULL){  
 printf("\nMEMORY Underflow\n");  
 }else{  
 if(header->link==NULL){  
 printf("KEY Not Found\n");  
 }else{  
 struct node\* ptr=header;  
 while(ptr->data!=key && ptr->link!=NULL){  
 ptr=ptr->link;  
 }  
 if(ptr->data==key){  
 new->link=ptr->link;  
 ptr->link=new;  
 }else{  
 printf("KEY Not Found\n");  
 }  
  
 }  
 }  
}  
void delete\_front(struct node\* header){  
 struct node\* ptr=header->link;  
 if(ptr==NULL){  
 printf("\n List is EMPTY\n");  
 }else{  
 header->link=ptr->link;  
 printf("\nDATA : %d DELETED\n",ptr->data);  
 free(ptr);  
 }  
}  
void delete\_rear(struct node\* header){  
  
 struct node\* ptr=header;  
 struct node\* ptr1;  
 if(ptr->link==NULL){  
 printf("\n List is EMPTY\n");  
 }else{  
 while(ptr->link!=NULL){  
 ptr1=ptr;  
 ptr=ptr->link;  
 }  
 ptr1->link=NULL;  
 printf("\nDATA : %d DELETED\n",ptr->data);  
 free(ptr);  
 }  
}  
void delete\_any(struct node\* header,int key){  
 struct node\* ptr1=header;  
 struct node\* ptr=ptr1->link;  
 if(ptr==NULL){  
 printf("\n List is EMPTY\n");  
 }else{  
 while(ptr->data!=key && ptr->link!=NULL){  
 ptr1=ptr;  
 ptr=ptr->link;  
 }  
 if(ptr->data==key){  
 ptr1->link=ptr->link;  
 printf("\nDATA : %d DELETED\n",ptr->data);  
 free(ptr);  
 }else{  
 printf("\nKEY Not Found\n");  
 }  
 }  
}  
void display(struct node\* header){  
 struct node\* ptr=header;  
 while(ptr->link!=NULL){  
 ptr=ptr->link;  
 printf(" DATA : %d\n",ptr->data);  
 }  
}  
  
void main(){  
 int n,x,y,key;  
 char ans='y';  
 struct node\* header = (struct node\*)malloc(sizeof(struct node));  
 header->link=NULL;  
 printf("\n--- OPERATION ON LINKED LIST --- \n\n");  
 printf(" 1. INSERT FRONT \n");  
 printf(" 2. INSERT REAR \n");  
 printf(" 3. INSERT ANY \n");  
 printf(" 4. DELETE FRONT \n");  
 printf(" 5. DELETE REAR \n");  
 printf(" 6. DELETE ANY \n");  
 printf(" 7. DISPLAY\n");  
 printf(" 8. EXIT\n");  
 while(ans=='y'){  
 printf("\nEnter the Choice (1/2/3/4/5/6/7/8): ");  
 scanf("%d",&n);  
 switch(n){  
 case 1:printf("--- INSERT FRONT ---\n");  
 printf("Enter element to be Inserted :");  
 scanf("%d", &x);  
 insert\_front(header,x);  
 break;  
 case 2:printf("--- INSERT REAR ---\n");  
 printf("Enter element to be Inserted :");  
 scanf("%d", &x);  
 insert\_rear(header,x);  
 break;  
 case 3:printf("--- INSERT ANY ---\n");  
 printf("Enter element to be Inserted :");  
 scanf("%d", &x);  
 printf("Enter KEY value :");  
 scanf("%d", &key);  
 insert\_any(header,x,key);  
 break;  
 case 4:printf("--- DELETE FRONT ---\n");  
 delete\_front(header);  
 break;  
 case 5:printf("--- DELETE REAR ---\n");  
 delete\_rear(header);  
 break;  
 case 6:printf("--- DELETE ANY ---\n");  
 printf("Enter KEY value to be DELETED:");  
 scanf("%d", &key);  
 delete\_any(header,key);  
 break;  
 case 7:printf("--- DISPLAY ---\n");  
 display(header);  
 break;  
 case 8:ans='n';  
 break;  
 default:printf("Enter a Valid Input\n");  
 }  
 }  
}

**OUTPUT:**







**RESULT:**

The given operations are performed on a Linked List.

Time complexity of INSERT\_FRONT() operation is O(1).

Time complexity of INSERT\_REAR() operation is O(n).

Time complexity of INSERT\_ANY() operation is O(n).

Time complexity of DELETE\_FRONT() operation is O(1).

Time complexity of DELETE\_REAR() operation is O(n).

Time complexity of DELETE\_ANY() operation is O(n).

13/11/2020

**Experiment No: 13**

**STACK USING LINKED LISTS**

**AIM:**

Write a program to implement Stack using Linked List.

**DATA STRUCTURES USED:**

Stack

**ALGORITHM:**

Algorithm POP (ITEM)

1. new = GetNode(Node)
2. If (new = NULL) then
3. Print ”memory underflow”
4. Exit
5. Else
6. new->LINK=TOP->LINK
7. new-> DATA=ITEM
8. TOP->LINK=new
9. Endif
10. Stop

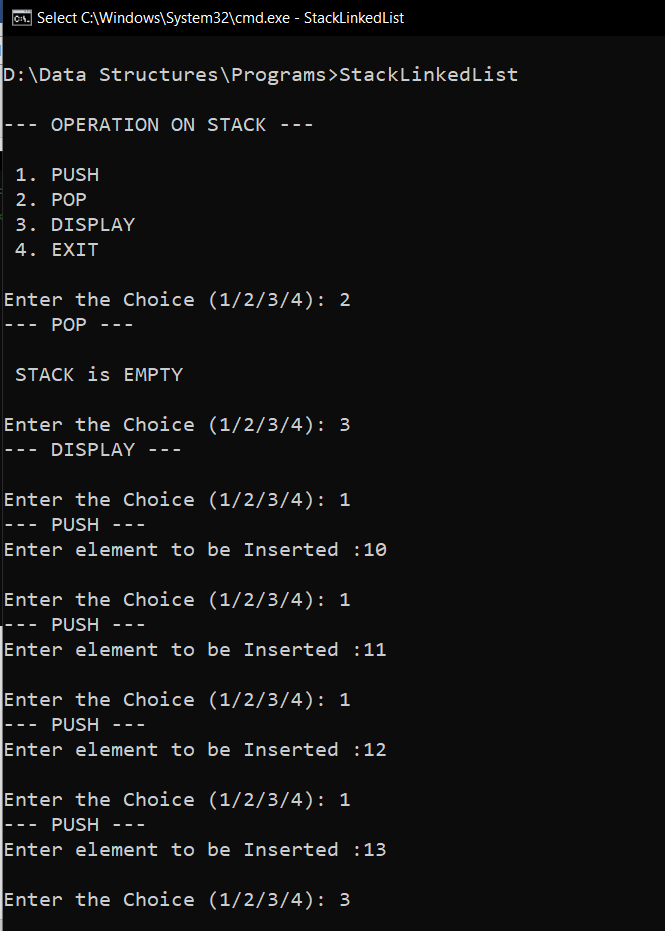
Algorithm POP()

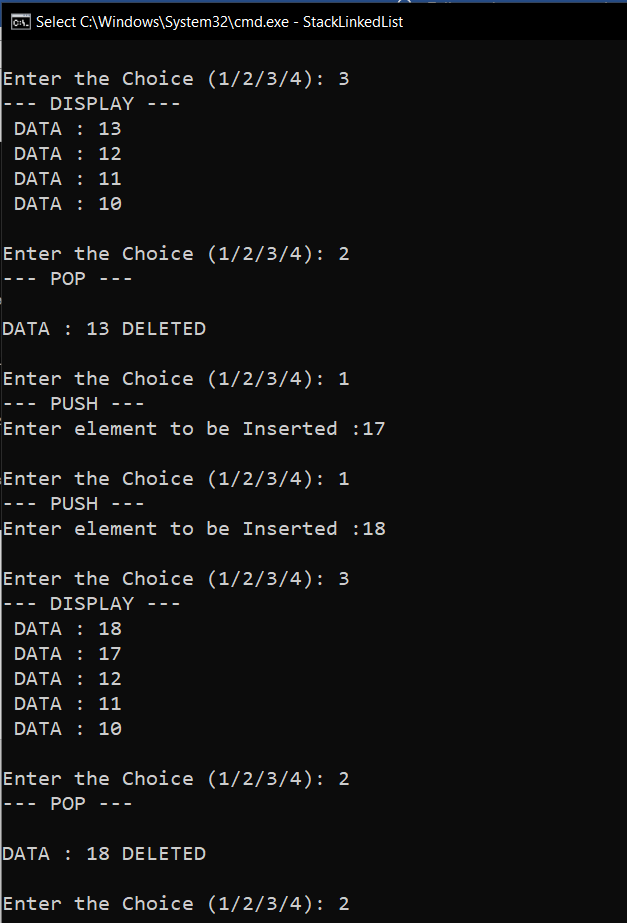
1. ptr=TOP->LINK
2. if(ptr=NULL)then
3. print “The stack is empty”
4. Exit
5. Else
6. TOP->LINK=ptr->LINK
7. ReturnNode(ptr)
8. stop

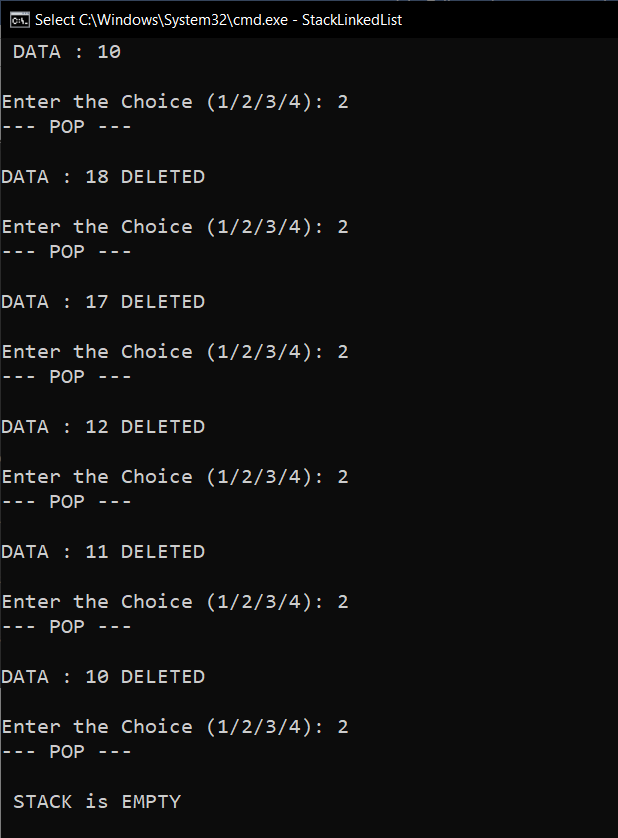
**PROGRAM:**

#include<stdio.h>  
#include<stdlib.h>  
struct node{  
 int data;  
 struct node \*link;  
};  
  
void push(struct node\* top,int x){  
 struct node\* new = (struct node\*)malloc(sizeof(struct node));  
 new->data=x;  
 new->link=NULL;  
 if(new==NULL){  
 printf("\nMEMORY Underflow\n");  
 }else{  
 if(top->link==NULL){  
 top->link=new;  
 }else{  
 new->link=top->link;  
 top->link=new;  
 }  
 }  
}  
  
void pop(struct node\* top){  
 struct node\* ptr=top->link;  
 if(ptr==NULL){  
 printf("\n STACK is EMPTY\n");  
 }else{  
 top->link=ptr->link;  
 printf("\nDATA : %d DELETED\n",ptr->data);  
 free(ptr);  
 }  
}  
  
void display(struct node\* top){  
 struct node\* ptr=top;  
 while(ptr->link!=NULL){  
 ptr=ptr->link;  
 printf(" DATA : %d\n",ptr->data);  
 }  
}  
  
void main(){  
 int n,x,y,key;  
 char ans='y';  
 struct node\* top = (struct node\*)malloc(sizeof(struct node));  
 top->link=NULL;  
 printf("\n--- OPERATION ON STACK --- \n\n");  
 printf(" 1. PUSH \n");  
 printf(" 2. POP \n");  
 printf(" 3. DISPLAY \n");  
 printf(" 4. EXIT \n");  
 while(ans=='y'){  
 printf("\nEnter the Choice (1/2/3/4): ");  
 scanf("%d",&n);  
 switch(n){  
 case 1:printf("--- PUSH ---\n");  
 printf("Enter element to be Inserted :");  
 scanf("%d", &x);  
 push(top,x);  
 break;  
 case 2:printf("--- POP ---\n");  
 pop(top);  
 break;  
 case 3:printf("--- DISPLAY ---\n");  
 display(top);  
 break;  
 case 4:ans='n';  
 break;  
 default:printf("Enter a Valid Input\n");  
 }  
 }  
}

**OUTPUT:**

****





**RESULT:**

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

Time complexity of PUSH() operation is O(1).

Time complexity of POP() operation is O(1).

13/11/2020

**Experiment No:14**

**QUEUE USING LINKED LISTS**

**AIM:**

Write a program to implement Queue using on Linked List.

**DATA STRUCTURES USED:**

Queue

**ALGORITHM:**

Algorithm INSERT(ITEM)

1. new= GetNodes(Node)
2. new->DATA=ITEM
3. new->LINK=NULL
4. if (new = NULL) then
5. print”memory underflow”
6. Exit
7. Else
8. If (FRONT->LINK=NULL)
9. FRONT->LINK=new
10. REAR->LINK=new
11. else
12. REAR->LINK->LINK= new
13. REAR->LINK=new
14. EndIf
15. EndIf
16. Stop

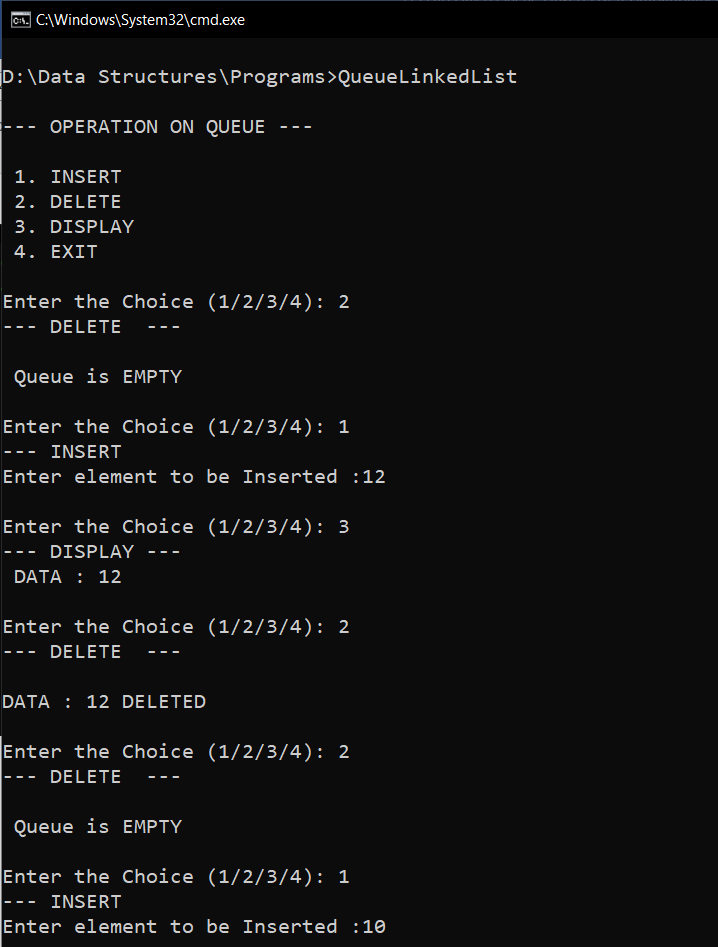
Algorithm DELETE()

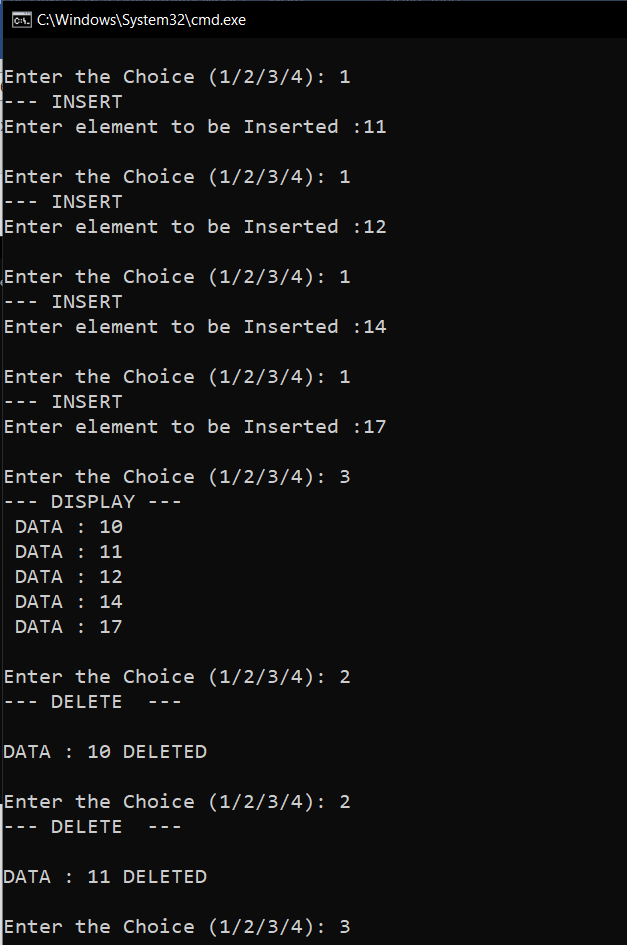
1. ptr=HEADER->LINK
2. if(ptr=NULL)then
3. print “The Queue is empty”
4. Exit
5. Else
6. FRONT->LINK=ptr->LINK
7. ReturnNode(ptr)
8. stop

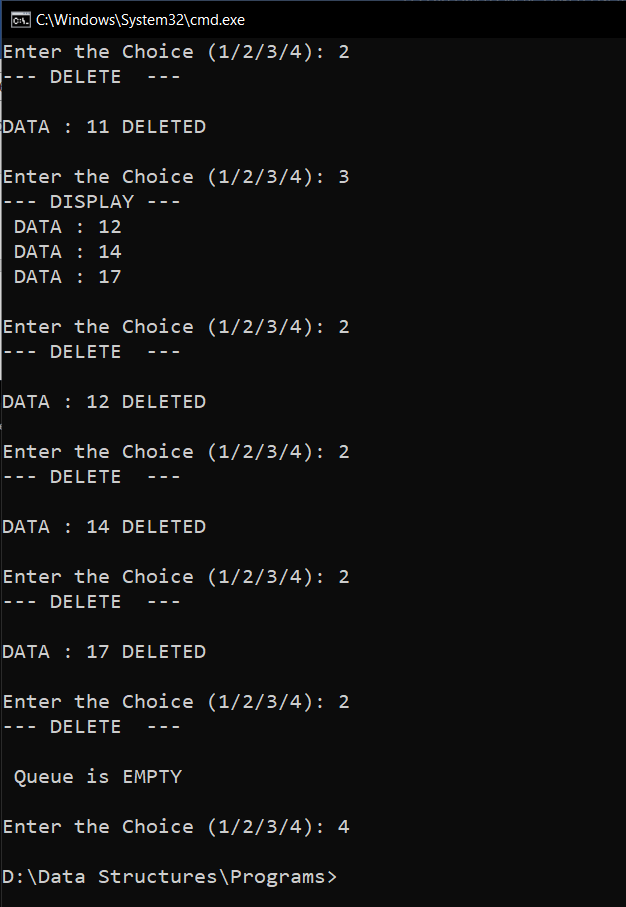
**PROGRAM:**

#include<stdio.h>  
#include<stdlib.h>  
struct node{  
 int data;  
 struct node \*link;  
};  
  
void insert(struct node\* front,struct node\* rear,int x){  
 struct node\* new = (struct node\*)malloc(sizeof(struct node));  
 new->data=x;  
 new->link=NULL;  
 if(new==NULL){  
 printf("\nMEMORY Underflow\n");  
 }else{  
 if(front->link==NULL){  
 front->link=new;  
 rear->link=new;  
 }else{  
 rear->link->link=new;  
 rear->link=new;  
 }  
 }  
}  
void delete(struct node\* front,struct node\* rear){  
 struct node\* ptr=front->link;  
 if(ptr==NULL){  
 printf("\n Queue is EMPTY\n");  
 }else{  
 front->link=ptr->link;  
 printf("\nDATA : %d DELETED\n",ptr->data);  
 free(ptr);  
 }  
}  
void display(struct node\* front,struct node\* rear){  
 struct node\* ptr=front;  
 while(ptr->link!=NULL){  
 ptr=ptr->link;  
 printf(" DATA : %d\n",ptr->data);  
 }  
}  
  
void main(){  
 int n,x,y;  
 char ans='y';  
 struct node\* front = (struct node\*)malloc(sizeof(struct node));  
 struct node\* rear = (struct node\*)malloc(sizeof(struct node));  
 front->link=NULL;  
 rear->link=NULL;  
 printf("\n--- OPERATION ON QUEUE --- \n\n");  
 printf(" 1. INSERT \n");  
 printf(" 2. DELETE \n");  
 printf(" 3. DISPLAY\n");  
 printf(" 4. EXIT\n");  
 while(ans=='y'){  
 printf("\nEnter the Choice (1/2/3/4): ");  
 scanf("%d",&n);  
 switch(n){  
 case 1:printf("--- INSERT \n");  
 printf("Enter element to be Inserted :");  
 scanf("%d", &x);  
 insert(front,rear,x);  
 break;  
 case 2:printf("--- DELETE ---\n");  
 delete(front,rear);  
 break;  
 case 3:printf("--- DISPLAY ---\n");  
 display(front,rear);  
 break;  
 case 4:ans='n';  
 break;  
 default:printf("Enter a Valid Input\n");  
 }  
 }  
}

**OUTPUT:**







**RESULT:**

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

Time complexity of INSERT() operation is O(1).

Time complexity of DELETE() operation is O(1).

20/11/2020

**Experiment No: 15**

**POLYNOMIAL USING LINKED LISTS**

**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 STRUCTURES USED:**

Linked List

**ALGORITHM:**

Algorithm POLYNOMIAL\_ADDITION()

1. Pptr =PHEADER->LINK
2. Qptr=QHEADER->LINK
3. RHEADER=GetNode(NODE)
4. RHEADER->LINK=NULL
5. RHEADER->Coeff=NULL
6. RHEADER->Exp=NULL
7. Rptr=RHEADER
8. While(Pptr!=NULL and Qptr!=NULL)
9. Case : Pptr->Exp=Qptr->Exp
10. new=GetNode(NODE)
11. Rptr->LINK=new
12. Rptr=new
13. Rptr->Coeff=Pptr->Coeff+Qptr->Coeff
14. Rptr->Exp=Pptr->Exp
15. Rptr->LINK=NULL
16. Pptr=Pptr->LINK
17. Qptr=Qptr->LINK
18. Case : Pptr->Exp>Qptr->Exp
19. new=GetNode(NODE)
20. Rptr->LINK=new
21. Rptr=new
22. Rptr->Coeff=Pptr->Coeff
23. Rptr->Exp=Pptr->Exp
24. Rptr->LINK=NULL
25. Pptr=Pptr->LINK
26. Case : Pptr->Exp<Qptr->Exp
27. new=GetNode(NODE)
28. Rptr->LINK=new
29. Rptr=new
30. Rptr->Coeff=Qptr->Coeff
31. Rptr->Exp=Qptr->Exp
32. Rptr->LINK=NULL
33. Qptr=Qptr->LINK
34. EndWhile
35. If(Pptr!=NULL and Qptr=NULL)
36. While(Pptr!=NULL)
37. new=GetNode(NODE)
38. Rptr->LINK=new
39. Rptr=new
40. Rptr->Coeff=Pptr->Coeff
41. Rptr->Exp=Pptr->Exp
42. Rptr->LINK=NULL
43. Pptr=Pptr->LINK
44. EndWhile
45. EndIf
46. If(Pptr=NULL and Qptr!=NULL)
47. While(Qptr!=NULL)
48. new=GetNode(NODE)
49. Rptr->LINK=new
50. Rptr=new
51. Rptr->Coeff=Qptr->Coeff
52. Rptr->Exp=Qptr->Exp
53. Rptr->LINK=NULL
54. Qptr=Qptr->LINK
55. EndWhile
56. EndIf

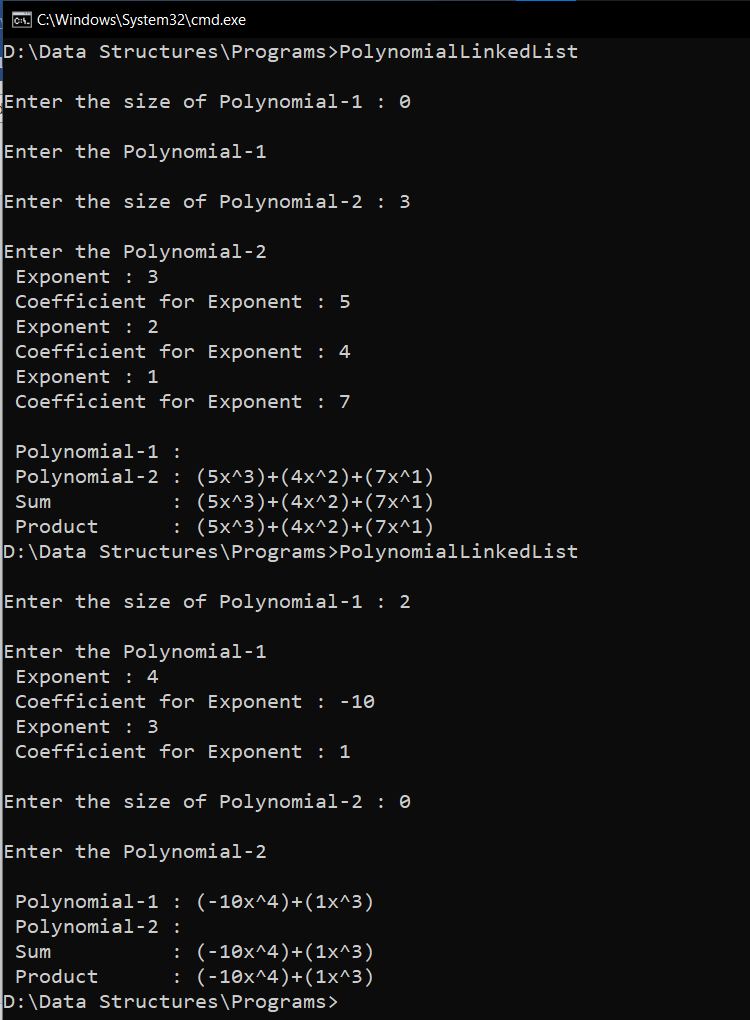
Algorithm POLYNOMIAL\_MULTIPLICATION()

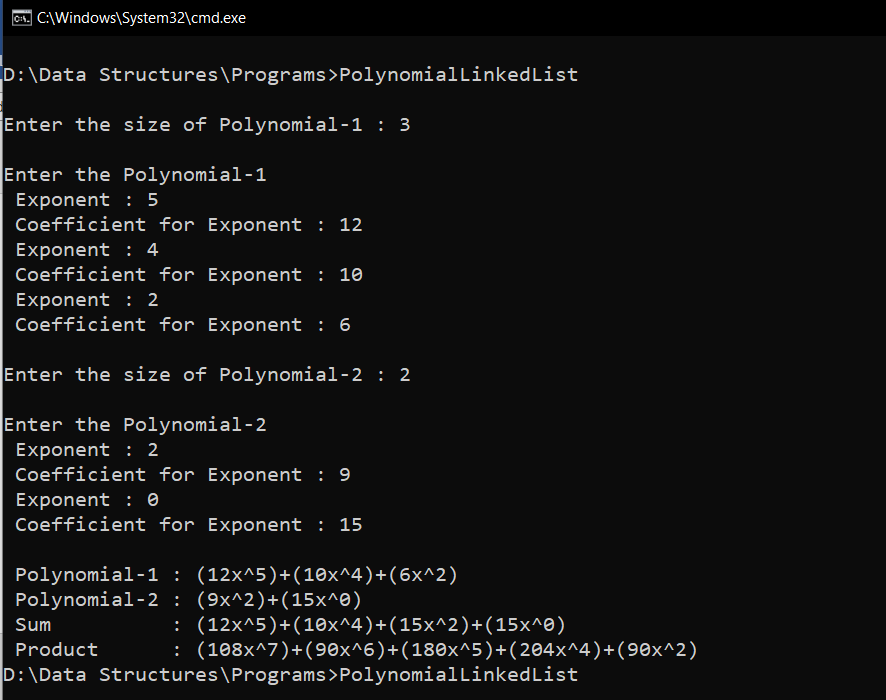
1. Pptr =PHEADER->LINK
2. Qptr=QHEADER->LINK
3. RHEADER=GetNode(NODE)
4. RHEADER->LINK=NULL
5. RHEADER->Coeff=NULL
6. RHEADER->Exp=NULL
7. If(Pptr!=NULL and Qptr=NULL)
8. While(Pptr!=NULL)
9. new=GetNode(NODE)
10. Rptr->LINK=new
11. Rptr=new
12. Rptr->Coeff=Pptr->Coeff
13. Rptr->Exp=Pptr->Exp
14. Rptr->LINK=NULL
15. Pptr=Pptr->LINK
16. EndWhile
17. Else If(Pptr=NULL and Qptr!=NULL)
18. While(Qptr!=NULL)
19. new=GetNode(NODE)
20. Rptr->LINK=new
21. Rptr=new
22. Rptr->Coeff=Qptr->Coeff
23. Rptr->Exp=Qptr->Exp
24. Rptr->LINK=NULL
25. Qptr=Qptr->LINK
26. EndWhile
27. Else
28. While(Pptr!=NULL)
29. Qptr=QHEADER->LINK
30. While(Qptr!=NULL)
31. new=GetNode(NODE)
32. Rptr->LINK=new
33. Rptr=new
34. Rptr->Coeff=Pptr->Coeff\*Qptr->Coeff
35. Rptr->Exp=Pptr->Exp+Qptr->Exp
36. Rptr->LINK=NULL
37. Qptr=Qptr->LINK
38. EndWhile
39. Pptr=Pptr->LINK
40. EndWhile
41. EndIf
42. SORT() //Sort According To Exp and Combine the Add Coeff of same Exp

**PROGRAM:**

#include<stdio.h>  
#include<stdlib.h>  
struct node{  
 int coeff;  
 int exp;  
 struct node \*link;  
};  
  
void insert(struct node\* header){  
 struct node\* new = (struct node\*)malloc(sizeof(struct node));  
 printf(" Exponent : ");  
 scanf("%d",&new->exp);  
 printf(" Coefficient for Exponent : ");  
 scanf("%d",&new->coeff);  
 new->link=NULL;  
 if(new==NULL){  
 printf("\nMEMORY Underflow\n");  
 }else{  
 if(header->link==NULL){  
 header->link=new;  
 }else{  
 struct node\* ptr=header;  
 while(ptr->link!=NULL){  
 ptr=ptr->link;  
 }  
 ptr->link=new;  
 }  
 }  
}  
void sum(struct node\* header1,struct node\* header2,struct node\* header3){  
 struct node\* ptr1=header1->link;  
 struct node\* ptr2=header2->link;  
 struct node\* ptr3=header3;  
 while(ptr1!=NULL && ptr2!=NULL){  
 if(ptr1->exp==ptr2->exp){  
 struct node\* new = (struct node\*)malloc(sizeof(struct node));  
 ptr3->link=new;  
 ptr3=new;  
 ptr3->coeff=ptr1->coeff+ptr2->coeff;  
 ptr3->exp=ptr1->exp;  
 ptr3->link=NULL;  
 ptr1=ptr1->link;  
 ptr2=ptr2->link;  
 }else if(ptr1->exp>ptr2->exp){  
 struct node\* new = (struct node\*)malloc(sizeof(struct node));  
 ptr3->link=new;  
 ptr3=new;  
 ptr3->coeff=ptr1->coeff;  
 ptr3->exp=ptr1->exp;  
 ptr3->link=NULL;  
 ptr1=ptr1->link;  
 }else{  
 struct node\* new = (struct node\*)malloc(sizeof(struct node));  
 ptr3->link=new;  
 ptr3=new;  
 ptr3->coeff=ptr2->coeff;  
 ptr3->exp=ptr2->exp;  
 ptr3->link=NULL;  
 ptr2=ptr2->link;  
 }  
 }  
 if(ptr1!=NULL && ptr2==NULL){  
 while(ptr1!=NULL){  
 struct node\* new = (struct node\*)malloc(sizeof(struct node));  
 ptr3->link=new;  
 ptr3=new;  
 ptr3->coeff=ptr1->coeff;  
 ptr3->exp=ptr1->exp;  
 ptr3->link=NULL;  
 ptr1=ptr1->link;  
 }  
 }  
 if(ptr1==NULL && ptr2!=NULL){  
 while(ptr2!=NULL){  
 struct node\* new = (struct node\*)malloc(sizeof(struct node));  
 ptr3->link=new;  
 ptr3=new;  
 ptr3->coeff=ptr2->coeff;  
 ptr3->exp=ptr2->exp;  
 ptr3->link=NULL;  
 ptr2=ptr2->link;  
 }  
 }  
}  
void product(struct node\* header1,struct node\* header2,struct node\* header4){  
 struct node\* ptr1=header1->link;  
 struct node\* ptr2=header2->link;  
 struct node\* ptr4=header4;  
 if(ptr1==NULL && ptr2!=NULL){  
 while(ptr2!=NULL){  
 struct node\* new = (struct node\*)malloc(sizeof(struct node));  
 ptr4->link=new;  
 ptr4=new;  
 ptr4->coeff=ptr2->coeff;  
 ptr4->exp=ptr2->exp;  
 ptr4->link=NULL;  
 ptr2=ptr2->link;  
 }  
 }else if(ptr1!=NULL && ptr2==NULL){  
 while(ptr1!=NULL){  
 struct node\* new = (struct node\*)malloc(sizeof(struct node));  
 ptr4->link=new;  
 ptr4=new;  
 ptr4->coeff=ptr1->coeff;  
 ptr4->exp=ptr1->exp;  
 ptr4->link=NULL;  
 ptr1=ptr1->link;  
 }  
 }else{  
 while(ptr1!=NULL){  
 ptr2=header2->link;  
 while(ptr2!=NULL){  
 struct node\* new = (struct node\*)malloc(sizeof(struct node));  
 ptr4->link=new;  
 ptr4=new;  
 ptr4->coeff=ptr1->coeff\*ptr2->coeff;  
 ptr4->exp=ptr1->exp+ptr2->exp;  
 ptr4->link=NULL;  
 ptr2=ptr2->link;  
 }  
 ptr1=ptr1->link;  
 }  
 }  
}  
void sort(struct node\* header,struct node\* header5){  
 struct node\* ptr=header->link;  
 struct node\* ptr1=header->link;  
 struct node\* ptr5=header5;  
 int exp,coeff;  
 if(ptr==NULL){  
 }else if(ptr->link==NULL){  
 }else{  
 while(ptr1->link!=NULL){  
 ptr=header->link;  
 while(ptr->link!=NULL){  
 if(ptr->exp < ptr->link->exp){  
 exp=ptr->exp;  
 coeff=ptr->coeff;  
 ptr->exp=ptr->link->exp;  
 ptr->coeff=ptr->link->coeff;  
 ptr->link->exp=exp;  
 ptr->link->coeff=coeff;  
 }  
 ptr=ptr->link;  
 }  
 ptr1=ptr1->link;  
 }  
 }  
 ptr=header->link;  
 while(ptr!=NULL){  
 if(ptr5->exp==ptr->exp){  
 ptr5->coeff=ptr5->coeff+ptr->coeff;  
 ptr5->exp=ptr->exp;  
 ptr=ptr->link;  
 }else{  
 struct node\* new = (struct node\*)malloc(sizeof(struct node));  
 ptr5->link=new;  
 ptr5=new;  
 ptr5->coeff=ptr->coeff;  
 ptr5->exp=ptr->exp;  
 ptr5->link=NULL;  
 ptr=ptr->link;  
 }  
 }  
  
}  
void display(struct node\* header){  
 struct node\* ptr=header;  
 while(ptr->link!=NULL){  
 ptr=ptr->link;  
 printf("(%dx^%d)",ptr->coeff,ptr->exp);  
 if(ptr->link!=NULL){  
 printf("+");  
 }  
 }  
}  
  
void main(){  
 int n1,n2,i;  
 struct node\* header1 = (struct node\*)malloc(sizeof(struct node));  
 header1->link=NULL;  
 struct node\* header2 = (struct node\*)malloc(sizeof(struct node));  
 header2->link=NULL;  
 struct node\* header3 = (struct node\*)malloc(sizeof(struct node));  
 header3->link=NULL;  
 struct node\* header4 = (struct node\*)malloc(sizeof(struct node));  
 header4->link=NULL;  
 struct node\* header5 = (struct node\*)malloc(sizeof(struct node));  
 header5->link=NULL;  
 printf("\nEnter the size of Polynomial-1 : ");  
 scanf("%d",&n1);  
 printf("\nEnter the Polynomial-1 \n");  
 for(i=0;i<n1;i++){  
 insert(header1);  
 }  
 printf("\nEnter the size of Polynomial-2 : ");  
 scanf("%d",&n2);  
 printf("\nEnter the Polynomial-2 \n");  
 for(i=0;i<n2;i++){  
 insert(header2);  
 }  
 sum(header1,header2,header3);  
 product(header1,header2,header4);  
 printf("\n Polynomial-1 : ");display(header1);  
 printf("\n Polynomial-2 : ");display(header2);  
 printf("\n Sum : ");display(header3);  
 printf("\n Product :");sort(header4,header5);display(header5);  
}

**OUTPUT:**





**RESULT:**

Two polynomials are stored using linked list. Both are displayed along with their sum

and product.

22/11/2020

**Experiment No:16**

**STUDENT LINKED LISTS**

**AIM:**

The details of Student (Roll 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 Roll Number.

5.Display the resultant list after every operation.

**DATA STRUCTURES USED:**

Linked List

**ALGORITHM:**

Algorithm INSERT ()

1. new= GetNode(Node)
2. //Input the details and Initialize it to the node
3. if (new = NULL) then
4. print”memory underflow”
5. Exit
6. Else
7. ptr=HEADER
8. While(ptr-> LINK!=NULL)do
9. ptr=ptr->LINK
10. Endwhile
11. ptr->LINK= new
12. new->LINK=NULL
13. Stop

Algorithm DELETE(KEY)

1. ptr1=HEADER
2. ptr=ptr1->LINK
3. while(ptr!=NULL)
4. if(ptr->rollno!= KEY)
5. ptr1=ptr
6. ptr=ptr->LINK
7. else
8. ptr1->LINK=ptr->LINK
9. ReturnNode(ptr)
10. Exit
11. Endif
12. Endwhile
13. If ptr=NULL
14. Print”Roll No Searched doesn’t exist”
15. Endif
16. Stop

Algorithm SEARCH(KEY)

1. ptr=HEADER->LINK
2. while(ptr!=NULL)
3. if(ptr->rollno!= KEY)
4. // Display the Details of the Node
5. Endif
6. Endwhile
7. If (ptr=NULL)
8. Print”Node with key doesn’t exist”
9. Endif
10. Stop

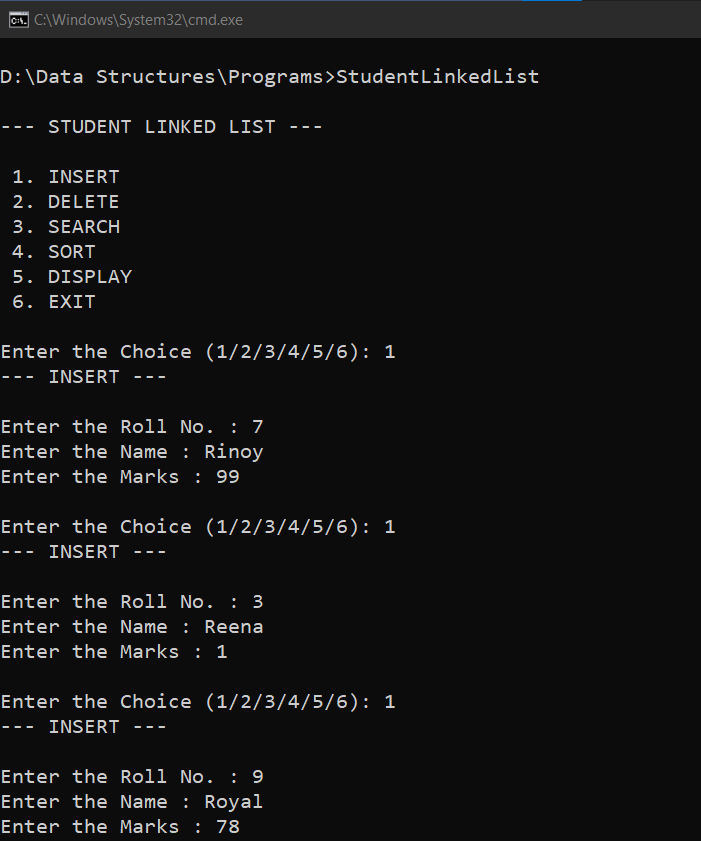
Algorithm SORT()

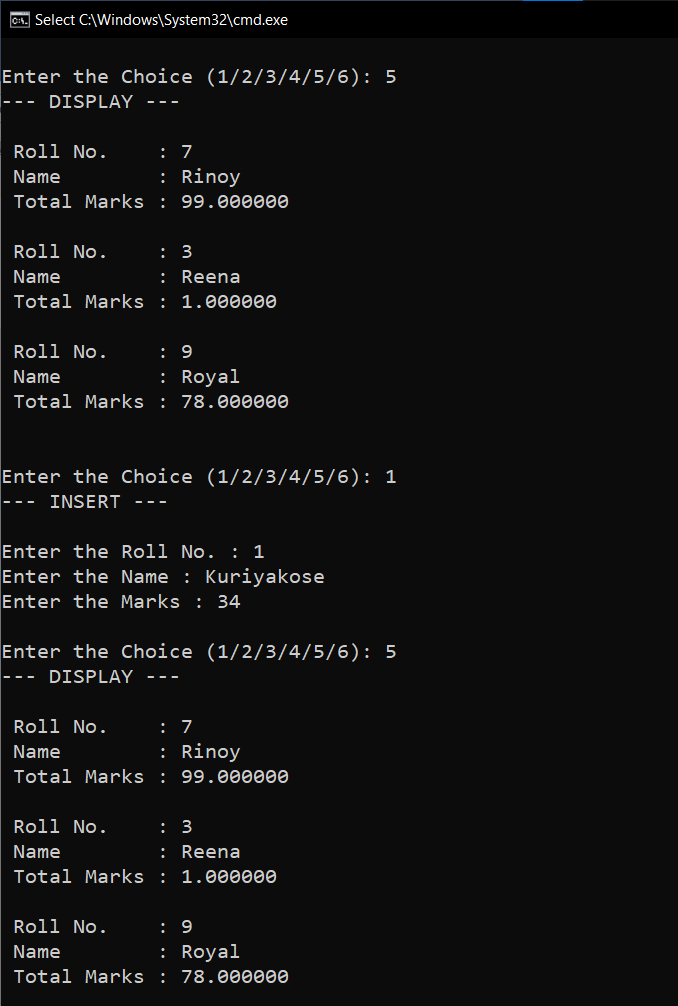
1. ptr1=HEADER->LINK
2. ptr2=HEADER->LINK
3. while(ptr1->LINK!=NULL)
4. ptr2=HEADER->LINK
5. while(ptr1->LINK!=NULL)
6. If(ptr2->rollno> ptr2->LINK->rollno)
7. // Interchange the Values in NODE ptr2 and ptr2->LINK
8. EndIf
9. EndWhile
10. EndWhile

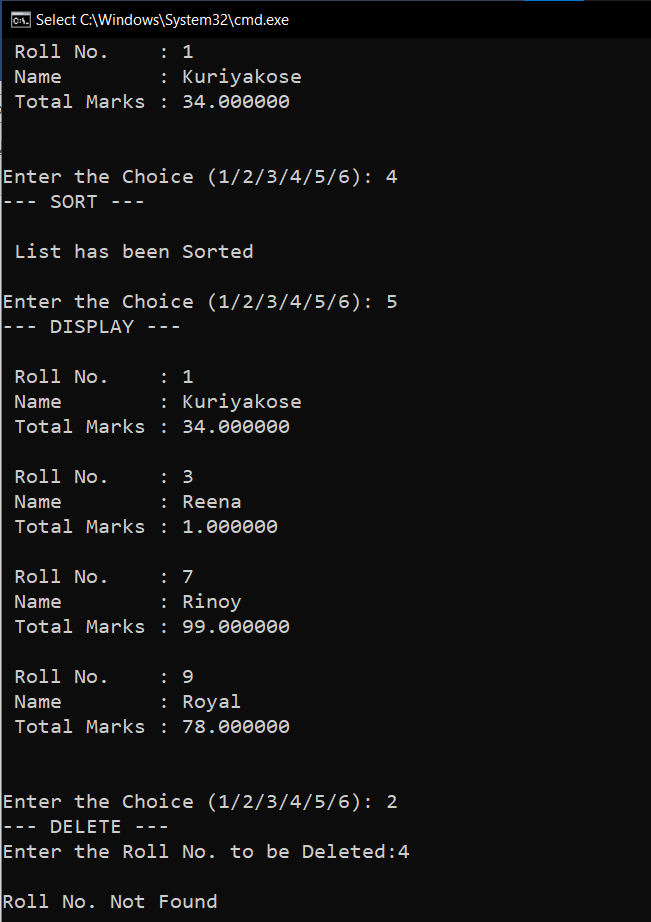
**PROGRAM:**

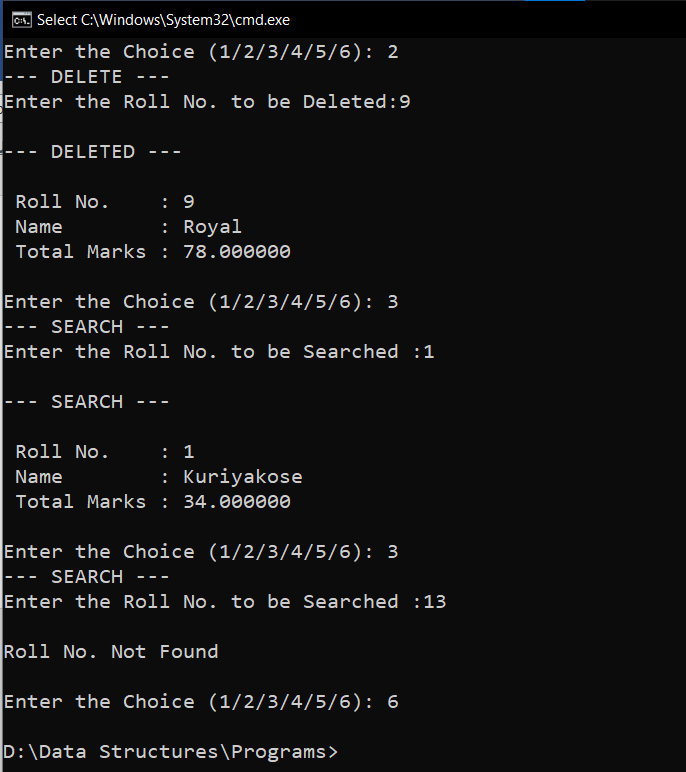
#include<stdio.h>  
#include<stdlib.h>  
#include<string.h>  
struct node{  
 int rollno;  
 char name[50];  
 float marks;  
 struct node \*link;  
};  
  
void insert(struct node\* header){  
 struct node\* new = (struct node\*)malloc(sizeof(struct node));  
 printf("\nEnter the Roll No. : ");  
 scanf("%d",&new->rollno);  
 printf("Enter the Name : ");  
 scanf("%s",new->name);  
 printf("Enter the Marks : ");  
 scanf("%f",&new->marks);  
 new->link=NULL;  
 if(new==NULL){  
 printf("\nMemory Underflow\n");  
 }else{  
 if(header->link==NULL){  
 header->link=new;  
 }else{  
 struct node\* ptr=header;  
 while(ptr->link!=NULL){  
 ptr=ptr->link;  
 }  
 ptr->link=new;  
 }  
 }  
}  
  
void delete(struct node\* header,int key){  
 struct node\* ptr1=header;  
 struct node\* ptr=ptr1->link;  
 if(ptr==NULL){  
 printf("\n List is Empty\n");  
 }else{  
 while(ptr->rollno!=key && ptr->link!=NULL){  
 ptr1=ptr;  
 ptr=ptr->link;  
 }  
 if(ptr->rollno==key){  
 ptr1->link=ptr->link;  
 printf("\n--- DELETED ---\n\n");  
 printf(" Roll No. : %d\n",ptr->rollno);  
 printf(" Name : %s\n",ptr->name);  
 printf(" Total Marks : %f\n",ptr->marks);  
 free(ptr);  
 }else{  
 printf("\nRoll No. Not Found\n");  
 }  
 }  
}  
  
void search(struct node\* header,int key){  
 struct node\* ptr=header->link;  
 if(ptr==NULL){  
 printf("\n List is Empty\n");  
 }else{  
 while(ptr->rollno!=key && ptr->link!=NULL){  
 ptr=ptr->link;  
 }  
 if(ptr->rollno==key){  
 printf("\n--- SEARCH ---\n\n");  
 printf(" Roll No. : %d\n",ptr->rollno);  
 printf(" Name : %s\n",ptr->name);  
 printf(" Total Marks : %f\n",ptr->marks);  
 }else{  
 printf("\nRoll No. Not Found\n");  
 }  
 }  
}  
void sort(struct node\* header){  
 struct node\* ptr=header->link;  
 struct node\* ptr1=header->link;  
 int rollno;  
 char name[50];  
 float marks;  
 if(ptr==NULL){  
 printf("\n List is Empty\n");  
 }else if(ptr->link==NULL){  
 printf("\n List has only One Element\n");  
 }else{  
 while(ptr1->link!=NULL){  
 ptr=header->link;  
 while(ptr->link!=NULL){  
 if(ptr->rollno > ptr->link->rollno){  
 rollno=ptr->rollno;  
 strcpy(name,ptr->name);  
 marks=ptr->marks;  
 ptr->rollno=ptr->link->rollno;  
 strcpy(ptr->name,ptr->link->name);  
 ptr->marks=ptr->link->marks;  
 ptr->link->rollno=rollno;  
 strcpy(ptr->link->name,name);  
 ptr->link->marks=marks;  
 }  
 ptr=ptr->link;  
 }  
 ptr1=ptr1->link;  
 }  
 printf("\n List has been Sorted\n");  
 }  
}  
void display(struct node\* header){  
 printf("\n");  
 struct node\* ptr=header;  
 while(ptr->link!=NULL){  
 ptr=ptr->link;  
 printf(" Roll No. : %d\n",ptr->rollno);  
 printf(" Name : %s\n",ptr->name);  
 printf(" Total Marks : %f\n\n",ptr->marks);  
 }  
}  
  
void main(){  
 int n,x,y,key;  
 char ans='y';  
 struct node\* header = (struct node\*)malloc(sizeof(struct node));  
 header->link=NULL;  
 printf("\n--- STUDENT LINKED LIST --- \n\n");  
 printf(" 1. INSERT \n");  
 printf(" 2. DELETE \n");  
 printf(" 3. SEARCH \n");  
 printf(" 4. SORT \n");  
 printf(" 5. DISPLAY\n");  
 printf(" 6. EXIT\n");  
 while(ans=='y'){  
 printf("\nEnter the Choice (1/2/3/4/5/6): ");  
 scanf("%d",&n);  
 switch(n){  
 case 1:printf("--- INSERT ---\n");  
 insert(header);  
 break;  
 case 2:printf("--- DELETE ---\n");  
 printf("Enter the Roll No. to be Deleted:");  
 scanf("%d", &key);  
 delete(header,key);  
 break;  
 case 3:printf("--- SEARCH ---\n");  
 printf("Enter the Roll No. to be Searched :");  
 scanf("%d", &key);  
 search(header,key);  
 break;  
 case 4:printf("--- SORT ---\n");  
 sort(header);  
 break;  
 case 5:printf("--- DISPLAY ---\n");  
 display(header);  
 break;  
 case 6:ans='n';  
 break;  
 default:printf("Enter a Valid Input\n");  
 }  
 }  
}

**OUTPUT:**









**RESULT:**

The given operations are performed on a Student linked list.

22/11/2020

**Experiment No:17**

**DOUBLY LINKED LISTS**

**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 STRUCTURES USED:**

Doubly Linked List

**ALGORITHM:**

Algorithm INSERT(ITEM)

1. new= GetNodes(Node)
2. new->DATA=ITEM
3. new->lLINK=NULL
4. new->rLINK=NULL
5. if (new = NULL) then
6. print”memory underflow”
7. Exit
8. Else
9. If (FRONT->rLINK=NULL)
10. FRONT->rLINK=new
11. REAR->lLINK=new
12. new->lLINK=FRONT
13. new->rLINK=REAR
14. else
15. REAR->lLINK->rLINK= new
16. REAR->lLINK=new
17. new->lLINK=REAR->lLINK
18. new->rLINK=REAR
19. EndIf
20. EndIf
21. Stop

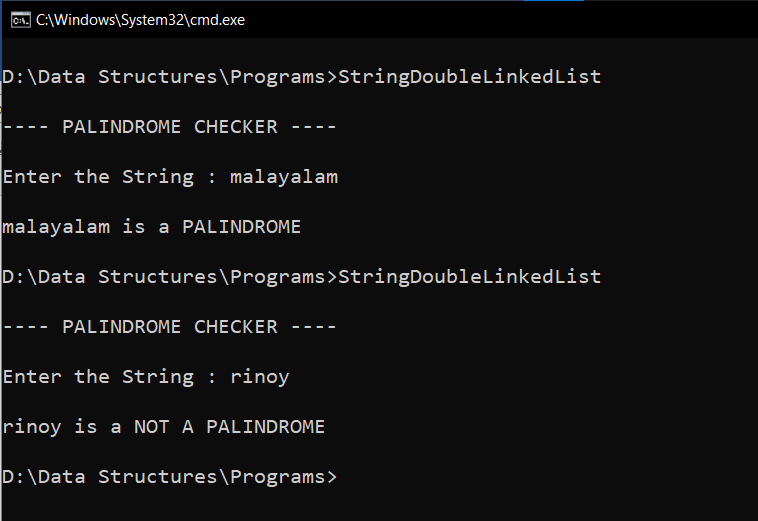
Algorithm CHECK\_PALINDROME()

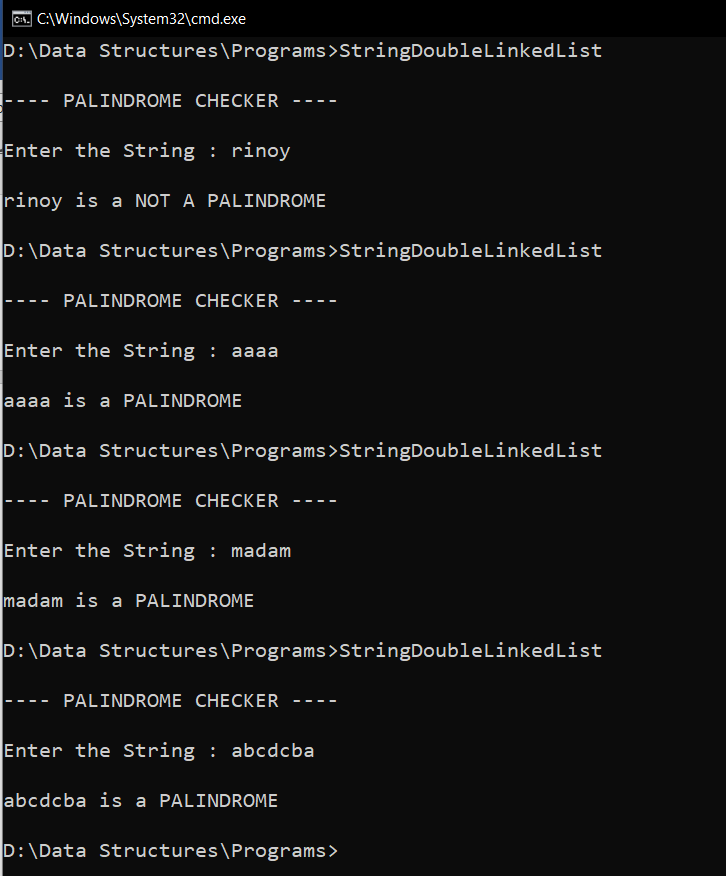
1. ptr1=FRONT
2. ptr2=REAR
3. while(ptr1!=ptr2)
4. if(ptr1->DATA!=ptr2->DATA)
5. Return 0
6. EndIf
7. Ptr1=ptr1->lLINK
8. Ptr2=ptr2->lLINK
9. EndWhile
10. Return 1

**PROGRAM:**

#include<stdio.h>  
#include<stdlib.h>  
struct node{  
 char data;  
 struct node \*rlink;  
 struct node \*llink;  
};  
  
void insert(struct node\* front,struct node\* rear,char x){  
 struct node\* new = (struct node\*)malloc(sizeof(struct node));  
 new->data=x;  
 new->rlink=NULL;  
 new->llink=NULL;  
 if(new==NULL){  
 printf("\nMEMORY Underflow\n");  
 }else{  
 if(front->rlink==NULL){  
 front->rlink=new;  
 new->llink=front;  
 rear->llink=new;  
 new->rlink=rear;  
 }else{  
 new->llink=rear->llink;  
 rear->llink->rlink=new;  
 new->rlink=rear;  
 rear->llink=new;  
 }  
 }  
}  
  
  
int check\_palindrome(struct node\* front,struct node\* rear){  
 struct node\* ptr1=front;  
 struct node\* ptr2=rear;  
 while(ptr1!=ptr2){  
 if(ptr1->data!=ptr2->data){  
 return 0;  
 }  
 ptr1=ptr1->rlink;  
 ptr2=ptr2->llink;  
 }  
 return 1;  
}  
  
void main(){  
 char string[50],\*arr;  
 struct node\* front = (struct node\*)malloc(sizeof(struct node));  
 struct node\* rear = (struct node\*)malloc(sizeof(struct node));  
 front->rlink=NULL;  
 front->llink=NULL;  
 rear->rlink=NULL;  
 rear->llink=NULL;  
 printf("\n---- PALINDROME CHECKER ----\n");  
 printf("\nEnter the String : ");  
 scanf("%s",string);  
 arr=string;  
 while(\*arr!='\0'){  
 insert(front,rear,\*arr);  
 arr++;  
 }  
 if(check\_palindrome(front,rear)){  
 printf("\n%s is a PALINDROME \n",string);  
 }else{  
 printf("\n%s is a NOT A PALINDROME \n",string);  
 }  
 }

**OUTPUT:**





**RESULT:**

The given string was checked for palindromes using Doubly Linked List.

02/01/2021

**Experiment No:18**

**BINARY TREE**

**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 STRUCTURES USED:**

Tree using Linked List

**ALGORITHM:**

Algorithm build\_tree(root)

//ptr=root

1. If ptr != NULL

2. ptr->DATA=item

3. Read option if Node has a left child

4. If opion = yes

5. ptr->LC = GetNode(NODE)

6. build\_tree(ptr->LC)

7. Else

8. ptr->LC = NULL

9. Endif

10. Read option if Node has a right child

11. If option = yes

12. ptr->RC = GetNode(NODE)

13. build\_tree(ptr->RC)

14. Else

15. ptr->RC = NULL

16. Endif

17. Endif

Algorithm search\_link(ptr, KEY)

1. If ptr->DATA != KEY

2. If ptr->LC != NULL

3. ptr1 = SearchLink(ptr->LC, KEY)

4. If ptr1 != NULL

5. Return ptr1

6. Endif

7. Endif

8. If ptr->RC != NULL

9. ptr1 = SearchLink(ptr->RC, KEY)

10. If ptr1 != NULL

11. Return ptr1

12. Endif

13. Endif

14. Return NULL

15. Else

16. Return ptr

17. Endif

Algorithm insert\_tree(ROOT,KEY)

1. ptr = search\_link(ROOT, KEY)

2. If ptr = NULL

3. Print "KEY not found"

4. Exit

5. Else

6. If ptr->LC = NULL or ptr->RC = NULL

7. Read option insert as left child or right child

8. If option = left

9. If ptr->LC = NULL

10. new= GetNode(NODE)

11. new->LC = NULL

12. new->RC = NULL

13. new->DATA=ITEM

14. Else

15. Print "KEY has a left child"

16. Endif

17. Else if option = right

18. If ptr->RC = NULL

19. new= GetNode(NODE)

20. new->LC = NULL

21. new->RC = NULL

22. new->DATA=ITEM

23. Else

24. Print "KEY has a right child"

25. Endif

26. Endif

27. Else

28. Print "KEY has both left child and right child"

29. Endif

30. Endif

Algorithm inorder\_traversal(root)

1. ptr=root

2. If ptr!= NULL

3. inorder\_traversal(ptr->LC)

4. print ptr->DATA

5. inorder\_traversal(ptr->RC)

6. Endif

Algorithm preorder\_traversal(root)

1. ptr=root

2. If ptr!= NULL

3. print ptr->DATA

4. preorder\_traversal(ptr->LC)

5. preorder\_traversal(ptr->RC)

6. Endif

Algorithm postorder\_traversal(root)

1. ptr=root

2. If ptr!= NULL

3. postorder\_traversal(ptr->LC)

4. postorder\_traversal(ptr->RC)

5. print ptr->DATA

6. Endif

Algorithm search\_parent(ptr, parent,KEY)

1. If ptr->DATA != KEY

2. If ptr->LC != NULL

3. parent = SearchParent(ptr->LC, KEY, ptr)

4. If parent != NULL

5. return parent

6. Endif

7. Endif

8. If ptr->RC != NULL

9. parent = SearchParent(ptr->RC, KEY, ptr)

10. If parent != NULL

11. return parent

12. Endif

13. Endif

14. return NULL

15. Else

16. return parent

17. Endif

Algorithm DeleteTree(ROOT,KEY)

1. parent = search\_parent(ROOT, ROOT,KEY)

2. If parent! = NULL

3. Ptr1=parent->LC

4. Ptr2=parent->RC

5. If ptr1 != NULL and ptr1->DATA = KEY

6. If ptr1->LC = NULL and ptr1->RC = NULL

7. parent->LC = NULL

8. Else

9. Print "KEY is not a leaf node"

10. Endif

11. Else

12. If ptr2->LC = NULL and ptr2->RC = NULL

13. parent->RC = NULL

14. Else

15. Print "KEY is not a leaf node"

16. Endif

17. Endif

18. Else

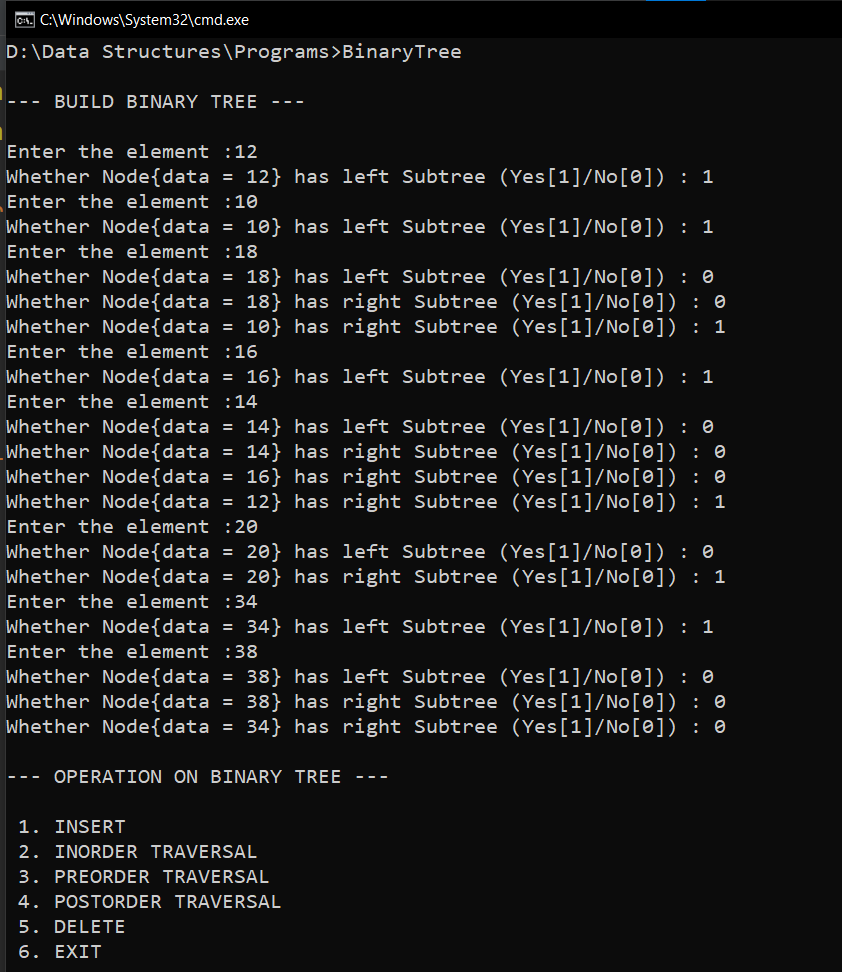
19. Print "KEY not found"

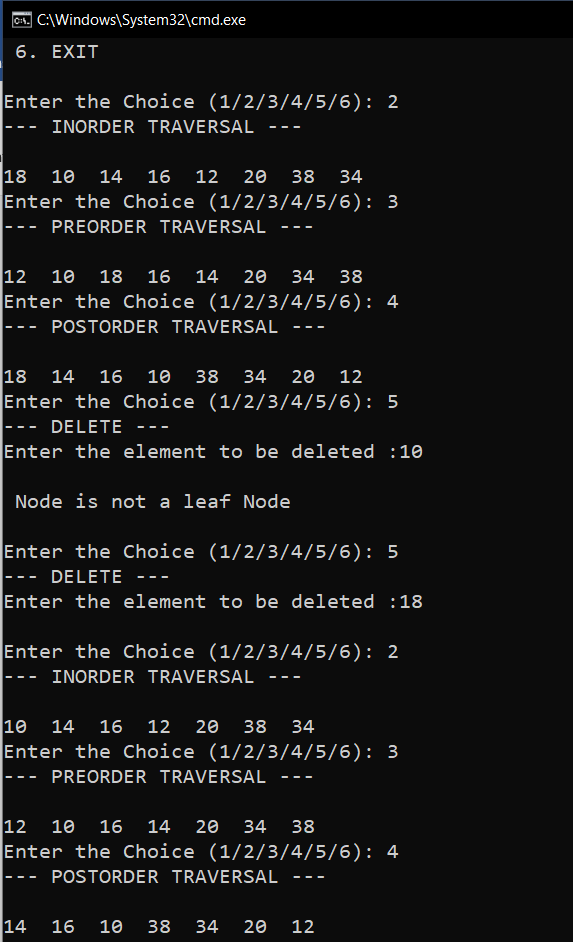
20. Endif

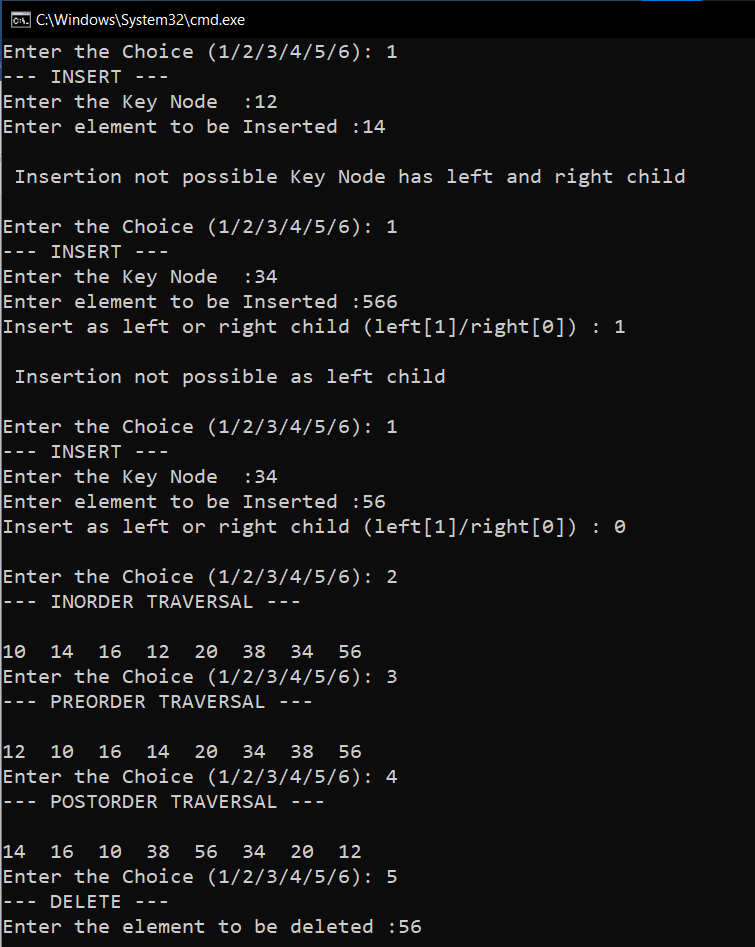
**PROGRAM:**

#include<stdio.h>  
#include<stdlib.h>  
  
struct node{  
 int data;  
 struct node \*lchild;  
 struct node \*rchild;  
};  
  
void build\_tree(struct node\* ptr){  
 int item ,ans;  
 if(ptr!=NULL){  
 printf("Enter the element :");  
 scanf("%d", &item);  
 ptr->data=item;  
 printf("Whether Node{data = %d} has left Subtree (Yes[1]/No[0]) : ",ptr->data);  
 scanf("%d", &ans);  
 struct node\* lcptr = (struct node\*)malloc(sizeof(struct node));  
 if(ans==1){  
 ptr->lchild=lcptr;  
 build\_tree(lcptr);  
 }else{  
 lcptr=NULL;  
 ptr->lchild=NULL;  
 build\_tree(lcptr);  
 }  
 printf("Whether Node{data = %d} has right Subtree (Yes[1]/No[0]) : ",ptr->data);  
 scanf("%d", &ans);  
 struct node\* rcptr = (struct node\*)malloc(sizeof(struct node));  
 if(ans==1){  
 ptr->rchild=rcptr;  
 build\_tree(rcptr);  
 }else{  
 rcptr=NULL;  
 ptr->rchild=NULL;  
 build\_tree(rcptr);  
 }  
 }  
}  
  
struct node \* search\_link(struct node\* root ,int key){  
 struct node\* ptr = root;  
 struct node\* ptr1;  
 if(ptr->data != key){  
 if(ptr->lchild!=NULL){  
 ptr1 = search\_link(ptr->lchild, key);  
 if(ptr1 != NULL){  
 return ptr1;  
 }  
 }  
 if(ptr->rchild!=NULL){  
 ptr1 = search\_link(ptr->rchild, key);  
 if(ptr1 != NULL){  
 return ptr1;  
 }  
 }  
 return NULL;  
 }  
 else{  
 return ptr;  
 }  
}  
  
void insert\_tree(struct node\* root,int key){  
 struct node\* ptr;  
 int item,ans;  
 printf("Enter element to be Inserted :");  
 scanf("%d", &item);  
 ptr=search\_link(root,key);  
 if(ptr==NULL){  
 printf("\n Search Unsucessful \n");  
 }else{  
 if((ptr->lchild==NULL)||(ptr->rchild==NULL)){  
 printf("Insert as left or right child (left[1]/right[0]) : ");  
 scanf("%d", &ans);  
 if(ans==1){  
 if(ptr->lchild==NULL){  
 struct node\* new = (struct node\*)malloc(sizeof(struct node));  
 new->data=item;  
 new->lchild=NULL;  
 new->rchild=NULL;  
 ptr->lchild=new;  
 }else{  
 printf("\n Insertion not possible as left child \n");  
 }  
 }  
 if(ans==0){  
 if(ptr->rchild==NULL){  
 struct node\* new = (struct node\*)malloc(sizeof(struct node));  
 new->data=item;  
 new->lchild=NULL;  
 new->rchild=NULL;  
 ptr->rchild=new;  
 }else{  
 printf("\n Insertion not possible as right child \n");  
 }  
  
 }  
 }  
 else{  
 printf("\n Insertion not possible Key Node has left and right child \n");  
 }  
 }  
}  
  
  
void inorder\_traversal(struct node\* root){  
 struct node\* ptr;  
 ptr = root;  
 if(ptr!=NULL){  
 inorder\_traversal(ptr->lchild);  
 printf("%d ",ptr->data);  
 inorder\_traversal(ptr->rchild);  
 }  
}  
  
void preorder\_traversal(struct node\* root){  
 struct node\* ptr;  
 ptr = root;  
 if(ptr!=NULL){  
 printf("%d ",ptr->data);  
 preorder\_traversal(ptr->lchild);  
 preorder\_traversal(ptr->rchild);  
 }  
}  
  
void postorder\_traversal(struct node\* root){  
 struct node\* ptr;  
 ptr = root;  
 if(ptr!=NULL){  
 postorder\_traversal(ptr->lchild);  
 postorder\_traversal(ptr->rchild);  
 printf("%d ",ptr->data);  
 }  
}  
  
struct node\* search\_parent(struct node\* ptr ,struct node\* parent,int item){  
 if(ptr->data != item){  
 if(ptr->lchild != NULL){  
 parent = search\_parent(ptr->lchild,ptr,item);  
 if(parent != NULL)  
 return parent;  
 }  
 if(ptr->rchild != NULL)  
 {  
 parent = search\_parent(ptr->rchild,ptr,item);  
 if(parent != NULL)  
 return parent;  
 }  
 return NULL;  
 }  
 else  
 return parent;  
}  
  
  
void delete\_tree(struct node\* root,int item){  
 struct node\* parent;  
 struct node\* ptr;  
 struct node\* ptr1;  
 struct node\* ptr2;  
 ptr=root;  
 if(ptr==NULL){  
 printf("\n Tree is Empty\n");  
 }else if(root->data == item ){  
 if(root->lchild==NULL && root->rchild == NULL){  
 root=NULL;  
 }  
 else{  
 printf("\n Node is not a leaf Node\n");  
 }  
 }else{  
 parent = search\_parent(root,root,item);  
 if(parent!=NULL){  
 ptr1 = parent->lchild;  
 ptr2 = parent->rchild;  
 if(ptr1->data==item){  
 if((ptr1->lchild==NULL)&&(ptr1->rchild==NULL)){  
 parent->lchild=NULL;  
 }else{  
 printf("\n Node is not a leaf Node\n");  
 }  
 }  
 if(ptr2->data==item){  
 if((ptr2->lchild==NULL)&&(ptr2->rchild==NULL)){  
 parent->rchild=NULL;  
 }else{  
 printf("\n Node is not a leaf Node\n");  
 }  
 }  
 }else{  
 printf("\n Node with data item doesn't exists\n");  
 }  
 }  
}  
  
void main(){  
 int n,item,key;  
 char ans='y';  
 struct node\* root = (struct node\*)malloc(sizeof(struct node));  
 root->lchild=NULL;  
 root->rchild=NULL;  
 printf("\n--- BUILD BINARY TREE --- \n\n");  
 build\_tree(root);  
 printf("\n--- OPERATION ON BINARY TREE --- \n\n");  
 printf(" 1. INSERT \n");  
 printf(" 2. INORDER TRAVERSAL\n");  
 printf(" 3. PREORDER TRAVERSAL\n");  
 printf(" 4. POSTORDER TRAVERSAL\n");  
 printf(" 5. DELETE \n");  
 printf(" 6. EXIT \n");  
 while(ans=='y'){  
 printf("\nEnter the Choice (1/2/3/4/5/6): ");  
 scanf("%d",&n);  
 switch(n){  
 case 1:printf("--- INSERT ---\n");  
 if(root == NULL){  
 printf("\n Tree is empty \n\n");  
 printf("Enter element to be Inserted :");  
 scanf("%d", &item);  
 root = (struct node\*)malloc(sizeof(struct node));  
 root->lchild=NULL;  
 root->rchild=NULL;  
 root->data=item;  
 }else{  
 printf("Enter the Key Node :");  
 scanf("%d", &key);  
 insert\_tree(root,key);  
 }  
 break;  
 case 2:printf("--- INORDER TRAVERSAL ---\n\n");  
 if(root!=NULL){  
 inorder\_traversal(root);  
 }else{  
 printf("\n Tree is empty \n");  
 }  
 break;  
 case 3:printf("--- PREORDER TRAVERSAL ---\n\n");  
 if(root!=NULL){  
 preorder\_traversal(root);  
 }else{  
 printf("\n Tree is empty \n");  
 }  
 break;  
 case 4:printf("--- POSTORDER TRAVERSAL ---\n\n");  
 if(root!=NULL){  
 postorder\_traversal(root);  
 }else{  
 printf("\n Tree is empty \n");  
 }  
 break;  
 case 5:printf("--- DELETE ---\n");  
 if(root!=NULL){  
 printf("Enter the element to be deleted :");  
 scanf("%d", &item);  
 if(root->data==item && root->lchild==NULL && root->rchild == NULL){  
 root = NULL;  
 }else if(root->data==item && root->lchild==NULL){  
 root=root->rchild;  
 }else if(root->data==item && root->rchild==NULL){  
 root=root->lchild;  
 }else if(root->data==item){  
 printf("\n Node is not a leaf Node\n");  
 }else{  
 delete\_tree(root,item);  
 }  
 }else{  
 printf("\n Tree is empty \n");  
 }  
 break;  
 case 6:ans='n';  
 break;  
 default:printf("Enter a Valid Input\n");  
 }  
 }  
}

**OUTPUT:**







**RESULT:**

The given operations are performed on a binary tree.

04/01/2021

**Experiment No:19**

**BINARY SEARCH TREE**

**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 STRUCTURES USED:**

Tree using Linked List

**ALGORITHM:**

Algorithm Insert()

ptr=root flag = False

1. While ptr != NULL

2. If ITEM <= ptr->DATA

3. ptr1 = ptr

4. ptr = ptr->LC

5. Else if ITEM > ptr->DATA

6. ptr1 = ptr

7. ptr = ptr->RC

8. Else

9. Flag=True

10. print “Item already exists”

11. Endwhile

12. If ptr = NULL

13. new= GetNode(NODE)

14. new->LC = NULL

15. new->RC = NULL

16. new->DATA = ITEM

17. If ptr1->DATA < ITEM

18. ptr1->RC = new

19. if ptr1->DATA>ITEM

20. ptr1->LC = new

21. Endif

22. EndIf

Algorithm inorder\_traversal(root)

1. ptr=root

2. If ptr!= NULL

3. inorder\_traversal(ptr->LC)

4. print ptr->DATA

5. inorder\_traversal(ptr->RC)

6. Endif

Algorithm preorder\_traversal(root)

1. ptr=root

2. If ptr!= NULL

3. print ptr->DATA

4. preorder\_traversal(ptr->LC)

5. preorder\_traversal(ptr->RC)

6. Endif

Algorithm postorder\_traversal(root)

1. ptr=root

2. If ptr!= NULL

3. postorder\_traversal(ptr->LC)

4. postorder\_traversal(ptr->RC)

5. print ptr->DATA

6. Endif

Algorithm successor(ptr)

1. ptr1 = ptr->RC

2. If ptr1 != NULL

3. While ptr1->LC != NULL

4. ptr1 = ptr1->LC

5. Endwhile

6. Endif

7. Return(ptr1)

Algorithm Delete()

1. ptr = ROOT

2. flag = false

3. While ptr != NULL and flag = false

4. If ITEM < ptr->DATA

5. parent = ptr

6. ptr = ptr->LC

7. Else if ITEM > ptr->DATA

8. parent = ptr

9. ptr = ptr->RC

10. Else

11. flag = true

12. Endif

13. Endwhile

14. If flag = false

15. print "ITEM doesn’t exist"

16. Exit

17. Endif

18. If ptr->LC = NULL and ptr->RC = NULL

19. CASE = 1

20. Else If ptr->LC != NULL and ptr->RC != NULL

21. CASE = 3

22. Else

23. CASE = 2

24. Endif

25. Endif

26. If CASE = 1

27. If parent->LC = ptr

28. parent->LC = NULL

29. Else

30. parent->RC = NULL

31. Endif

32. ReturnNode(ptr)

33.EndIf

34. if CASE = 2

35. If parent->LC = ptr

36. If ptr->LC = NULL

37. parent->LC = ptr->RC

38. Else

39. parent->LC = ptr->LC

40. Endif

41. Else

42. If ptr->LC = NULL

43. parent->RC = ptr->RC

44. Else

45. parent->RC = ptr->LC

46. Endif

47. Endif

48. ReturnNode(ptr)

49. If CASE=3

50. ptr1 = successor(ptr)

51. ITEM1 = ptr1->DATA

52. DeleteBST(ITEM1)

53. ptr->DATA = ITEM1

54. Endif

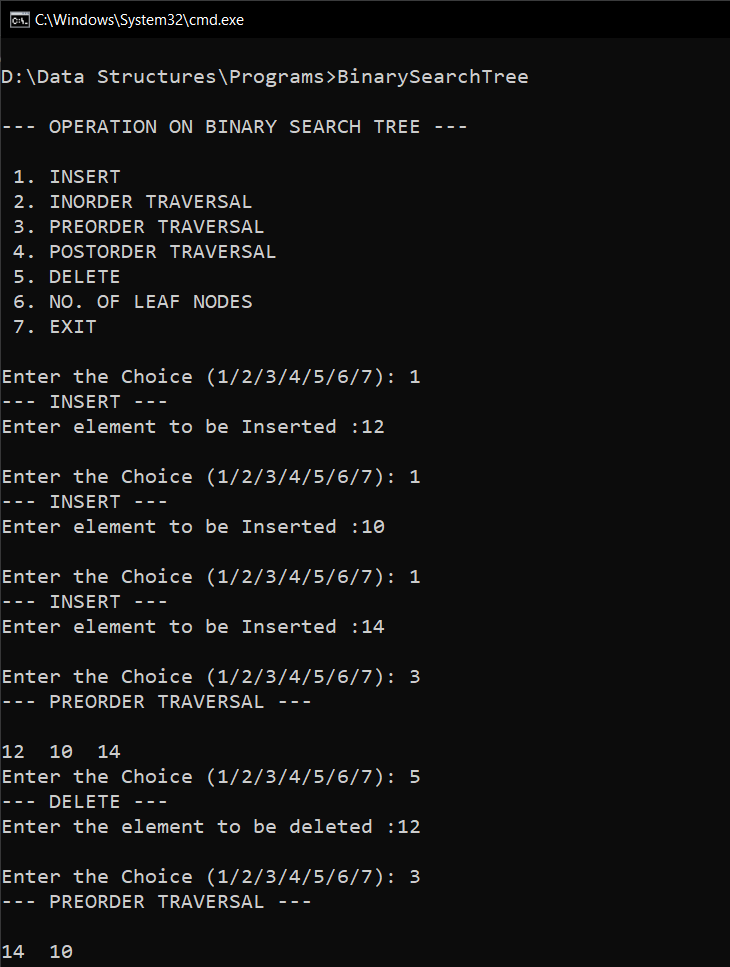
Algorithm no\_of\_leaf\_nodes(root)

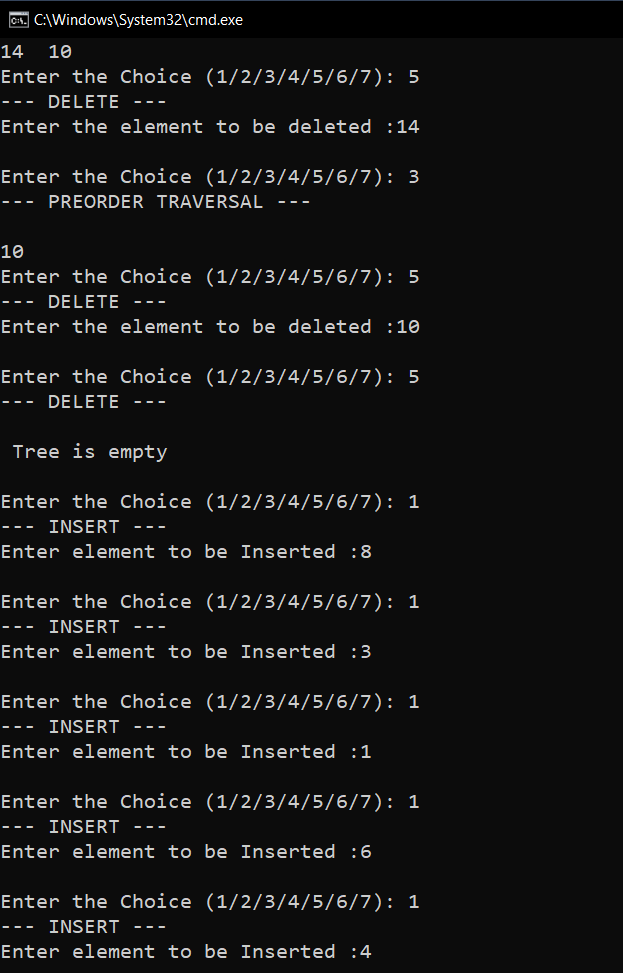
1. ptr=root
2. if ptr =NULL
3. return 0;
4. else if ptr->lchild == NULL && ptr->rchild == NULL
5. return 1;
6. Else
7. return no\_of\_leaf\_nodes(ptr->LC)+ no\_of\_leaf\_nodes(ptr->RC)

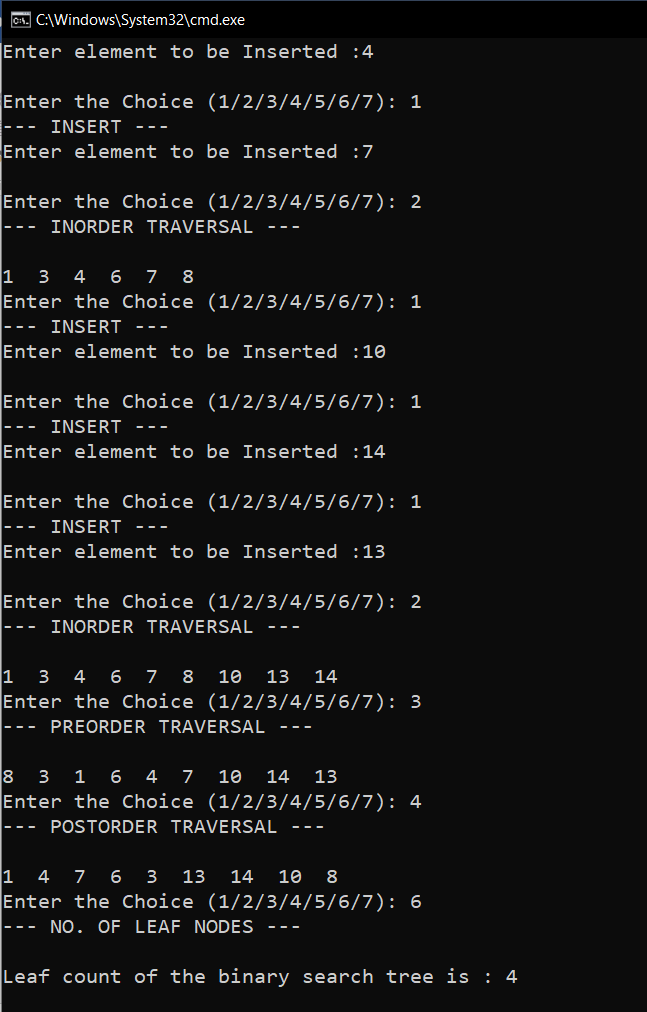
**PROGRAM:**

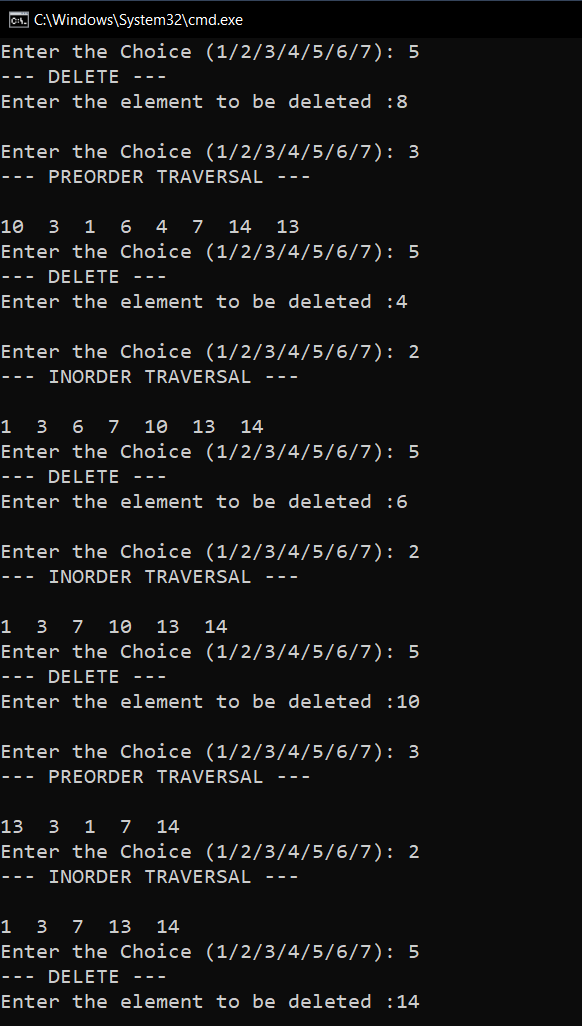
#include<stdio.h>  
#include<stdlib.h>  
  
struct node{  
 int data;  
 struct node \*lchild;  
 struct node \*rchild;  
};  
  
void Insert(struct node\* root,int item){  
 struct node\* ptr=root;  
 struct node\* ptr1;  
 int flag=0;  
 while(ptr!=NULL && flag == 0){  
 if(item<ptr->data){  
 ptr1=ptr;  
 ptr=ptr->lchild;  
 }else if(item>ptr->data){  
 ptr1=ptr;  
 ptr=ptr->rchild;  
 }else{  
 flag=1;  
 printf("\n ITEM already exists \n ");  
 }  
 }  
 if(ptr==NULL){  
 struct node\* new = (struct node\*)malloc(sizeof(struct node));  
 new->data=item;  
 new->lchild=NULL;  
 new->rchild=NULL;  
 if(ptr1->data<item){  
 ptr1->rchild=new;  
 }  
 if(ptr1->data>item){  
 ptr1->lchild=new;  
 }  
 }  
}  
void inorder\_traversal(struct node\* root){  
 struct node\* ptr;  
 ptr = root;  
 if(ptr!=NULL){  
 inorder\_traversal(ptr->lchild);  
 printf("%d ",ptr->data);  
 inorder\_traversal(ptr->rchild);  
 }  
}  
void preorder\_traversal(struct node\* root){  
 struct node\* ptr;  
 ptr = root;  
 if(ptr!=NULL){  
 printf("%d ",ptr->data);  
 preorder\_traversal(ptr->lchild);  
 preorder\_traversal(ptr->rchild);  
 }  
}  
void postorder\_traversal(struct node\* root){  
 struct node\* ptr;  
 ptr = root;  
 if(ptr!=NULL){  
 postorder\_traversal(ptr->lchild);  
 postorder\_traversal(ptr->rchild);  
 printf("%d ",ptr->data);  
 }  
}  
struct node\* successor(struct node\* ptr){  
 struct node\* ptr1;  
 ptr1=ptr->rchild;  
 if(ptr1!=NULL){  
 while(ptr1->lchild!=NULL){  
 ptr1=ptr1->lchild;  
 }  
 }  
 return(ptr1);  
}  
  
void Delete(struct node\* root,int item){  
 struct node\* ptr=root;  
 struct node\* ptr1;  
 struct node\* parent=NULL;  
 int flag=0,temp;  
 while(ptr!=NULL && flag == 0){  
 if(item<ptr->data){  
 parent=ptr;  
 ptr=ptr->lchild;  
 }else if(item>ptr->data){  
 parent=ptr;  
 ptr=ptr->rchild;  
 }else{  
 flag=1;  
 }  
 }  
 if(flag==0){  
 printf(" \nITEM doesn't exists\n");  
 }else{  
 if(ptr->lchild==NULL && ptr->rchild==NULL){  
 if(parent->lchild==ptr){  
 parent->lchild=NULL;  
 }  
 if(parent->rchild==ptr){  
 parent->rchild=NULL;  
 }  
 free(ptr);  
 }else if(ptr->lchild!=NULL && ptr->rchild!=NULL){  
 ptr1 = successor(ptr);  
 temp =ptr1->data;  
 Delete(root,temp);  
 ptr->data=temp;  
 free(ptr1);  
 }else{  
 if(parent->lchild==ptr){  
 if(ptr->lchild==NULL){  
 parent->lchild=ptr->rchild;  
 }else{  
 parent->lchild=ptr->lchild;  
 }  
 }else if(parent->rchild==ptr){  
 if(ptr->lchild==NULL){  
 parent->rchild=ptr->rchild;  
 }else{  
 parent->rchild=ptr->lchild;  
 }  
 }  
 free(ptr);  
 }  
 }  
}  
  
int no\_of\_leaf\_nodes(struct node\* root){  
 struct node\* ptr;  
 ptr = root;  
 if(ptr == NULL){  
 return 0;  
 }else if(ptr->lchild == NULL && ptr->rchild == NULL){  
 return 1;  
 }else{  
 return no\_of\_leaf\_nodes(ptr->lchild)+no\_of\_leaf\_nodes(ptr->rchild);  
 }  
}  
  
void main(){  
 int n,item,var=0;  
 char ans='y';  
 struct node\* root = NULL;  
 printf("\n--- OPERATION ON BINARY SEARCH TREE --- \n\n");  
 printf(" 1. INSERT \n");  
 printf(" 2. INORDER TRAVERSAL\n");  
 printf(" 3. PREORDER TRAVERSAL\n");  
 printf(" 4. POSTORDER TRAVERSAL\n");  
 printf(" 5. DELETE \n");  
 printf(" 6. NO. OF LEAF NODES \n");  
 printf(" 7. EXIT \n");  
 while(ans=='y'){  
 printf("\nEnter the Choice (1/2/3/4/5/6/7): ");  
 scanf("%d",&n);  
 switch(n){  
 case 1:printf("--- INSERT ---\n");  
 printf("Enter element to be Inserted :");  
 scanf("%d", &item);  
 if(root==NULL){  
 root = (struct node\*)malloc(sizeof(struct node));  
 root->lchild=NULL;  
 root->rchild=NULL;  
 root->data=item;  
 }else{  
 Insert(root,item);  
 }  
 var++;  
 break;  
 case 2:printf("--- INORDER TRAVERSAL ---\n\n");  
 if(root!=NULL){  
 inorder\_traversal(root);  
 }else{  
 printf("\n Tree is empty \n");  
 }  
 break;  
 case 3:printf("--- PREORDER TRAVERSAL ---\n\n");  
 if(root!=NULL){  
 preorder\_traversal(root);  
 }else{  
 printf("\n Tree is empty \n");  
 }  
 break;  
 case 4:printf("--- POSTORDER TRAVERSAL ---\n\n");  
 if(root!=NULL){  
 postorder\_traversal(root);  
 }else{  
 printf("\n Tree is empty \n");  
 }  
 break;  
 case 5:printf("--- DELETE ---\n");  
 if(root!=NULL){  
 printf("Enter the element to be deleted :");  
 scanf("%d", &item);  
 if(root->data==item && root->lchild==NULL && root->rchild == NULL){  
 root = NULL;  
 }else if(root->data==item && root->lchild==NULL){  
 root=root->rchild;  
 }else if(root->data==item && root->rchild==NULL){  
 root=root->lchild;  
 }else{  
 Delete(root,item);  
 }  
 }else{  
 printf("\n Tree is empty \n");  
 }  
 break;  
 case 6:printf("--- NO. OF LEAF NODES ---\n");  
 printf("\nLeaf count of the binary search tree is : %d\n",no\_of\_leaf\_nodes(root));  
 break;  
 case 7:ans='n';  
 break;  
 default:printf("Enter a Valid Input\n");  
 }  
 }  
}

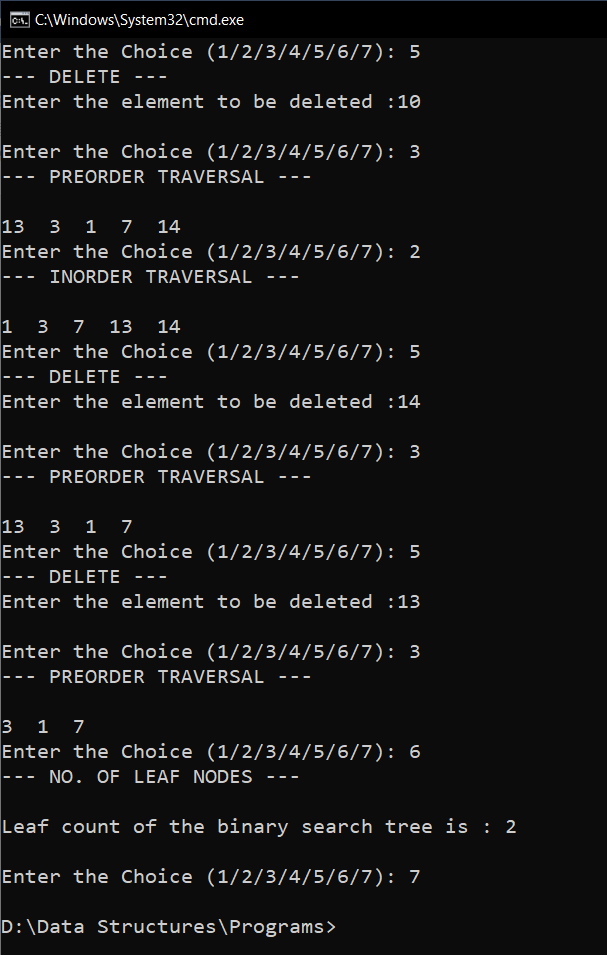
**OUTPUT:**











**RESULT:**

The given operations are performed on a binary search tree.

04/01/2021

**Experiment No: 20**

**SORT USING BINARY SEARCH TREE**

**AIM:**

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

**DATA STRUCTURES USED:**

Tree using Linked List

**ALGORITHM:**

Algorithm Insert()

ptr=root flag = False

1. While ptr != NULL

2. If ITEM <= ptr->DATA

3. ptr1 = ptr

4. ptr = ptr->LC

5. Else if ITEM > ptr->DATA

6. ptr1 = ptr

7. ptr = ptr->RC

8. Else

9. Flag=True

10. print “Item already exists”

11. Endwhile

12. If ptr = NULL

13. new= GetNode(NODE)

14. new->LC = NULL

15. new->RC = NULL

16. new->DATA = ITEM

17. If ptr1->DATA < ITEM

18. ptr1->RC = new

19. if ptr1->DATA>ITEM

20. ptr1->LC = new

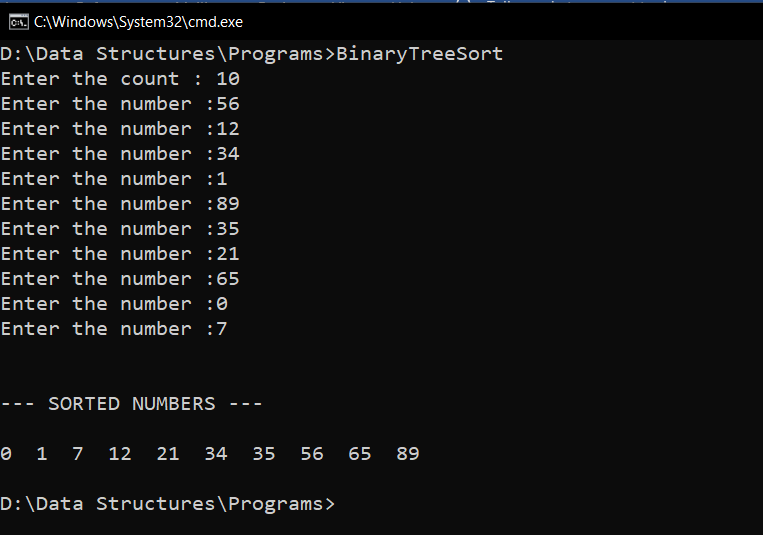
21. Endif

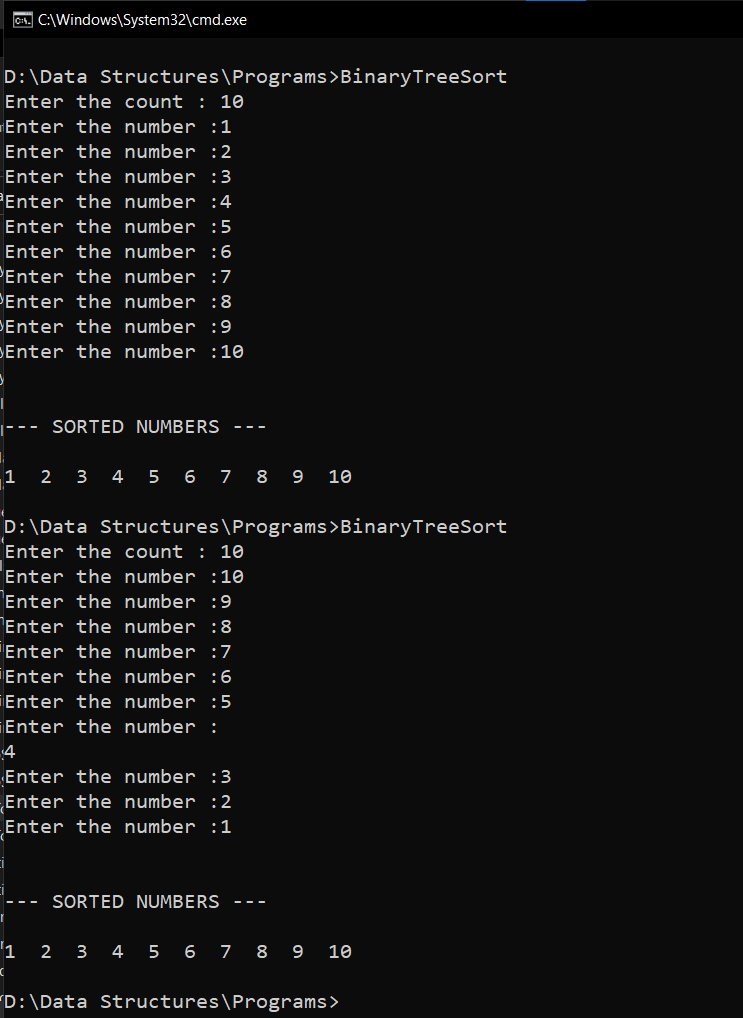
22. EndIf

**PROGRAM:**

#include<stdio.h>  
#include<stdlib.h>  
  
struct node{  
 int data;  
 struct node \*lchild;  
 struct node \*rchild;  
};  
  
  
void Insert(struct node\* root,int item){  
 struct node\* ptr=root;  
 struct node\* ptr1;  
 int flag=0;  
 while(ptr!=NULL && flag == 0){  
 if(item<ptr->data){  
 ptr1=ptr;  
 ptr=ptr->lchild;  
 }else if(item>ptr->data){  
 ptr1=ptr;  
 ptr=ptr->rchild;  
 }else{  
 flag=1;  
 printf("\n ITEM already exists \n ");  
 }  
 }  
 if(ptr==NULL){  
 struct node\* new = (struct node\*)malloc(sizeof(struct node));  
 new->data=item;  
 new->lchild=NULL;  
 new->rchild=NULL;  
 if(ptr1->data<item){  
 ptr1->rchild=new;  
 }  
 if(ptr1->data>item){  
 ptr1->lchild=new;  
 }  
 }  
}  
  
void inorder\_traversal(struct node\* root){  
 struct node\* ptr;  
 ptr = root;  
 if(ptr!=NULL){  
 inorder\_traversal(ptr->lchild);  
 printf("%d ",ptr->data);  
 inorder\_traversal(ptr->rchild);  
 }  
}  
  
  
void main(){  
 int n,i;  
 struct node\* root = (struct node\*)malloc(sizeof(struct node));  
 root->lchild=NULL;  
 root->rchild=NULL;  
 printf("Enter the count : ");  
 scanf("%d", &n);  
 if(n>0){  
 int\* arr = (int\*)malloc(sizeof(int)\*n);  
 for(i=0;i<n;i++){  
 printf("Enter the number :");  
 scanf("%d",&arr[i]);  
 }  
 root->data=arr[0];  
 for(i=1;i<n;i++){  
 Insert(root,arr[i]);  
 }  
 printf("\n\n--- SORTED NUMBERS ---\n\n");  
 inorder\_traversal(root);  
 printf("\n");  
 }else{  
 printf("\n Please Enter valid count ");  
 }  
}

**OUTPUT:**

****

****

**RESULT:**

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

11/01/2021

**Experiment No:21**

**GRAPH TRAVERSAL**

**AIM:**

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

1. DFS Traversal

2. BFS Traversal

**DATA STRUCTURES USED:**

Graph using Arrays,Stack,Queue.

**ALGORITHM:**

Algorithm DFS

START

1. Push the starting vertex into the stack

2. While stack not empty

3. Pop a vertex v

4. If v is not in VISIT

5. Visit the vertex x

6. Store v in VISIT

7. Push all the adjacent vertices of v into stack

8. EndIf

9. EndWhile

STOP

Algorithm BFS

START

1. Enqueue starting vertex

2. Visit the vertex

3. Store the vertex in VISIT

4. While queue not empty

5. Dequeue a vertex v

6. For all the adjacent vertices w of v

7. If w is not in VISIT

8. Enqueue w

9. Visit w

10. Store w in VISIT

11. EndIf

12. EndFor

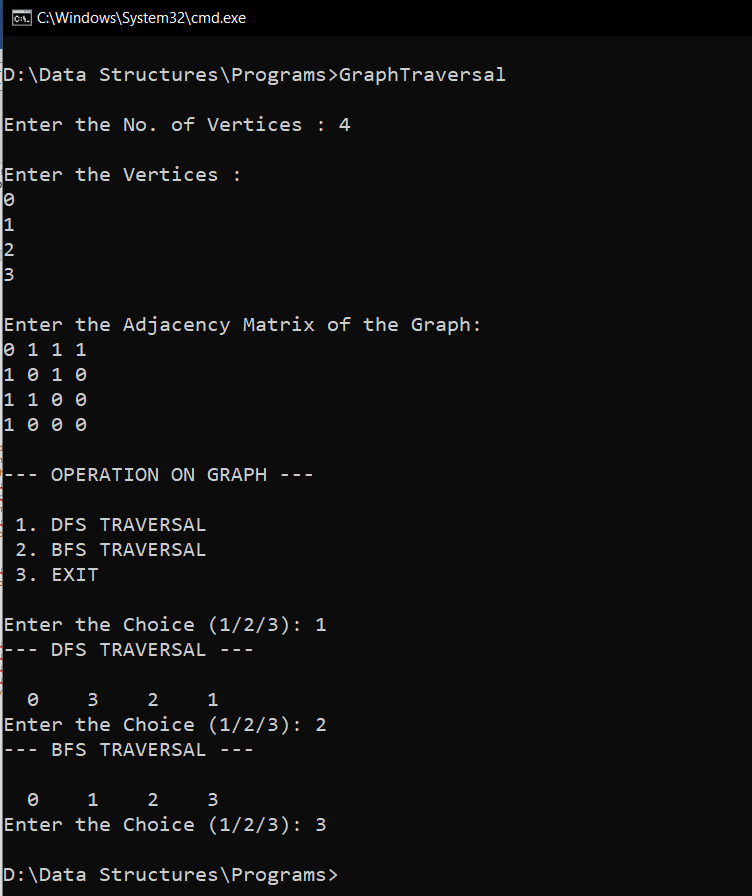
13. EndWhile

STOP

**PROGRAM:**

#include<stdio.h>  
#include<stdlib.h>  
int stack[100];  
int queue[100];  
int top = -1;  
int front = -1,rear=-1;  
void push(int x){  
 stack[++top] = x;  
}  
  
int pop(){  
 if(top!=-1){  
 int x = stack[top];  
 top--;  
 return x;  
 }  
}  
void enqueue(int x){  
 if(front == -1){  
 front = 0;  
 rear = 0;  
 queue[rear]=x;  
 }else{  
 queue[++rear]=x;  
 }  
}  
int dequeue(){  
 if(front != -1){  
 int x = queue[front];  
 if(front == rear){  
 front = -1;  
 rear = -1;  
 }else{  
 front = front + 1;  
 }  
 return x;  
 }  
}  
void dfs\_traversal(int n ,int value[], int adj[][n]){  
 int flag = 0;  
 int index=0,j,k;  
 int vertex;  
 int visit[n];  
 push(value[0]);  
 while(top!=-1){  
 vertex = pop();  
 for(j=0; j<n; j++){  
 if(visit[j] == vertex){  
 flag =1;  
 }  
 }  
 if(flag == 0){  
 visit[index] = vertex;  
 printf(" %d ",vertex);  
 for(j=0; j<n; j++){  
 if(value[j] == vertex){  
 for(k=0; k<n; k++){  
 if(adj[j][k] == 1 ){  
 push(value[k]);  
 }  
 }  
 break;  
 }  
 }  
 index++;  
 }  
 flag = 0;  
 }  
}  
void bfs\_traversal(int n ,int value[], int adj[][n]){  
 int index = 0;  
 int flag = 0,vertex,j,k,i;  
 int visit[n];  
 enqueue(value[0]);  
 printf(" %d ", value[0]);  
 visit[index++] = value[0];  
 while(front!= -1){  
 vertex = dequeue();  
 for( j=0; j<n; j++){  
 if(value[j] == vertex){  
 for(k=0; k<n; k++){  
 if(adj[j][k] == 1){  
 for( i=0; i<n; i++){  
 if(visit[i] == value[k]){  
 flag = 1;  
 }  
 }  
 if(flag == 0){  
 enqueue(value[k]);  
 printf(" %d ", value[k]);  
 visit[index] = value[k];  
 index++;  
 }  
 flag = 0;  
 }  
 }  
 break;  
 }  
 }  
 }  
}  
void main(){  
 int n,i,j,op;  
 char ans='y';  
 printf("\nEnter the No. of Vertices : ");  
 scanf("%d", &n);  
 int adj[n][n],value[n];  
 printf("\nEnter the Vertices : \n");  
 for(i=0; i<n; i++){  
 scanf("%d", &value[i]);  
 }  
 printf("\nEnter the Adjacency Matrix of the Graph:\n");  
 for(i=0; i<n; i++){  
 for(j=0; j<n; j++){  
 scanf("%d", &adj[i][j]);  
 }  
 }  
 printf("\n--- OPERATION ON GRAPH --- \n\n");  
 printf(" 1. DFS TRAVERSAL\n");  
 printf(" 2. BFS TRAVERSAL\n");  
 printf(" 3. EXIT \n");  
 while(ans=='y'){  
 printf("\nEnter the Choice (1/2/3): ");  
 scanf("%d",&op);  
 switch(op){  
 case 1:printf("--- DFS TRAVERSAL ---\n\n");  
 dfs\_traversal(n,value,adj);  
 break;  
 case 2:printf("--- BFS TRAVERSAL ---\n\n");  
 bfs\_traversal(n,value,adj);  
 break;  
 case 3:ans='n';  
 break;  
 default:printf("Enter a Valid Input\n");  
 }  
 }  
}

**OUTPUT:**



**RESULT:**

The given operations are performed on a graph using arrays.

10/02/2021

**Experiment No:22**

**QUICK SORT AND MERGE SORT**

**AIM:**

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

**DATA STRUCTURES USED:**

Arrays

**ALGORITHM:**

Algorithm Partition(A, p, r)

START

1. x = A[r]

2. i = p-1

3. for j = p to r

4. if (A[j] <= x)

5. i = i+1

6. if (i != j)

7. swap A[i] and A[j]

8. endif

9. endif

10. endfor

11. if (r != i+1)

12. swap A[i+1] and A[r]

13. endif

14. return i+1

STOP

Algorithm QuickSort(A, p, r)

START

1. if (p < r)

2. q = Partition(A, p ,r)

3. QuickSort(A, p, q-1)

4. QuickSort(A, q+1, r)

5. endif

STOP

Algorithm Merge(A, p, q, r)

START

1. n1 = q - p + 1

2. n2 = r - q

3. Declare L[n1], R[n2]

4. for i = 0 till n1

5. L[i] = A[p+i]

6. endfor

7. for j = 0 till n2

8. R[j] = A[q+j+1]

9. endfor

10. i = 0, j = 0,L[n1+1]=R[n2+1]=∞

11. for k = p to r

12. if (L[i] <= R[j])

13. A[k] = L[i]

14. i = i+1

15. else

20. A[k] = R[j]

21. j = j+1

25. endif

27. endfor

STOP

Algorithm MergeSort(A, p, r)

START

1. if (p < r)

2. q = floor((p+r)/2)

3. MergeSort(A, p, q)

4. MergeSort(A, q+1, r)

5. Merge(A, p, q, r)

6. endif

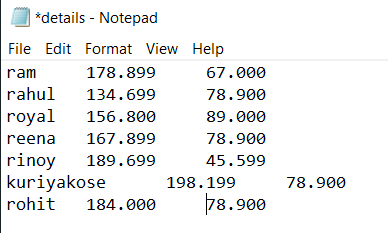
STOP

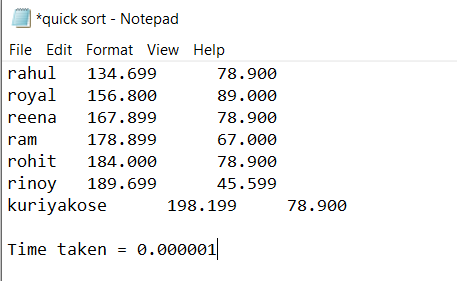
**PROGRAM:**

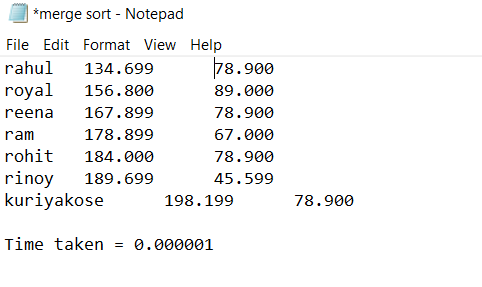
#include<stdio.h>  
#include<stdlib.h>  
#include<string.h>  
#include<time.h>  
#include<math.h>  
  
struct student  
{  
 char name[20];  
 float height;  
 float weight;  
};  
int partition(struct student s[], int p, int r){  
 struct student temp;  
 float x = s[r].height;  
 int i = p-1;  
 for(int j = p; j < r; j++){  
 if(s[j].height <= x){  
 i=i+1;  
 temp = s[i];  
 s[i] = s[j];  
 s[j] = temp;  
 }  
 }  
 temp = s[i+1];  
 s[i+1] = s[r];  
 s[r] = temp;  
 return i+1;  
}  
void quick\_sort(struct student s[], int p, int r)  
{  
 if(p < r)  
 {  
 int q = partition(s, p ,r);  
 quick\_sort(s, p, q-1);  
 quick\_sort(s, q+1, r);  
 }  
}  
void merge(struct student s[], int p, int q, int r){

int n1 = q - p + 1;  
 int n2 = r - q;  
 int i,j;  
 struct student L[n1], R[n2];  
 for( i = 0; i <n1; i++){  
 L[i] = s[p+i];  
 }  
 for(j = 0; j <n2;j++){  
 R[j] = s[q+j+1];  
 }  
 i = 0, j = 0;  
 int k;  
 for(k = p; k <= r; k++){  
 if(L[i].height <= R[j].height){  
 s[k] = L[i];  
 i=i+1;  
 if(i == n1){  
 k++;  
 break;  
 }  
 }  
 else{  
 s[k] = R[j];  
 j=j+1;  
 if(j == n2){  
 k++;  
 break;  
 }  
  
 }  
 }  
 while(i < n1)  
 {  
 s[k] = L[i];  
 i++;  
 k++;  
 }  
 while(j < n2)  
 {  
 s[k] = R[j];  
 j++;  
 k++;  
 }  
}  
  
void merge\_sort(struct student s[], int p, int r)  
{  
 if(p < r)  
 {  
 int q = floor((p+r)/2);  
 merge\_sort(s, p, q);  
 merge\_sort(s, q+1, r);  
 merge(s, p, q, r);  
 }  
}  
void main()  
{  
 int n,i;  
 char c,name[50];  
 float height, weight;  
  
 printf("Enter the Number of Students : ");  
 scanf("%d", &n);  
  
 FILE \*fp1 = fopen("details.txt", "w");  
 FILE \*fp2 = fopen("details.txt", "r");  
 FILE \*fp3 = fopen("quick sort.txt", "w");  
 FILE \*fp4 = fopen("merge sort.txt", "w");  
 struct student s1[50],s2[50];  
 for( i=0; i<n; i++){  
 printf("\nEnter the Details :\n");  
 printf(" Name : ");  
 scanf("%c", &c);  
 gets(name);  
 printf(" Height :");  
 scanf("%f", &height);  
 printf(" Weight : ");  
 scanf("%f", &weight);  
 fprintf(fp1,"%s\t%.3f\t%.3f\n",name,height,weight);  
 }  
 fclose(fp1);  
  
 for( i = 0; i <n; i++){  
 fscanf(fp2,"%s\t%f\t%f\n",s1[i].name, &s1[i].height, &s1[i].weight);  
 strcpy(s2[i].name,s1[i].name);  
 s2[i].height = s1[i].height;  
 s2[i].weight = s1[i].weight;  
 }  
 fclose(fp2);  
  
  
 clock\_t start ,stop;  
 start = clock();  
 quick\_sort(s1, 0, n-1);  
 stop= clock() ;  
 for(i = 0; i < n; i++){  
 fprintf(fp3,"%s\t%.3f\t%.3f\n",, s1[i].name, s1[i].height, s1[i].weight);  
 }  
 fprintf(fp3, "\nTime taken = %f", (double) (stop-start) / CLOCKS\_PER\_SEC);  
 fclose(fp3);  
  
 start = clock();  
 merge\_sort(s2, 0, n-1);  
 stop= clock();  
 for(i = 0; i < n; i++){  
 fprintf(fp4,"%s\t%.3f\t%.3f\n",, s2[i].name, s2[i].height, s2[i].weight);  
 }  
 fprintf(fp4, "\nTime taken = %f", (double) (stop-start) / CLOCKS\_PER\_SEC);  
 fclose(fp4);  
  
}

**OUTPUT:**







**RESULT:**

Quick sort and Merge sort were done on file containing data.

10/02/2021

**Experiment No:23**

**HEAP SORT**

**AIM:**

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:**

Arrays

**ALGORITHM:**

Algorithm CreateHeap(A, n)

START

1. i = 0

2. while (i < n)

3. j = i

4. while (j > 1)

5. if (A[j] > A[(j-1)/2])

6. swap A[j] and A[(j-1)/2]

7. j = j/2

8. else

9. j=1

10. endif

11 endwhile

12.. i = i+1

13. endwhile

STOP

Algorithm RemoveMax(A, i)

START

1. swap A[i] and A[1]

STOP

Algorithm RebuildHeap(A, i)

START

1. if (i =1)

2. exit

3. endif

4. j = 1,flag=TRUE

5. while(flag = TRUE)

6. leftchild = 2 \* j

7. rightchild = 2 \*j + 1

8. if (rightchild <= i)

9. if(A[j] <= A[leftchild] && A[leftchild] >= A[rc])

10. swap A[j] and A[leftchild]

11. j = leftchild

12. else if (A[j] <= A[rightchild] && A[rightchild] >= A[leftchild])

13. swap A[j] and A[rightchild]

14. j = rightchild

15. else

16. flag = FALSE

17. else if (leftchild <= i)

18. if (A[j] <= A[leftchild])

19. swap A[j] and A[leftchild]

20. j=leftchild

21. else

22. break

23. else

24. flag = FALSE

25. endif

26. endwhile

STOP

Algorithm HeapSort(A, n)

START

1. CreateHeap(A, n)

2. for i = n-1 down till 2

3. RemoveMax(A, i)

4. RebuildHeap(A, i-1)

5. endfor

STOP

Algorithm BinarySearch(A, num, l, r)

START

1. while (first <= last)

2. middle = (first+last) / 2

3. if (A[middle] == num)

4. return middle

5. else if (A[middle] < num)

6. first = middle+1

7. else

8. last = middle - 1

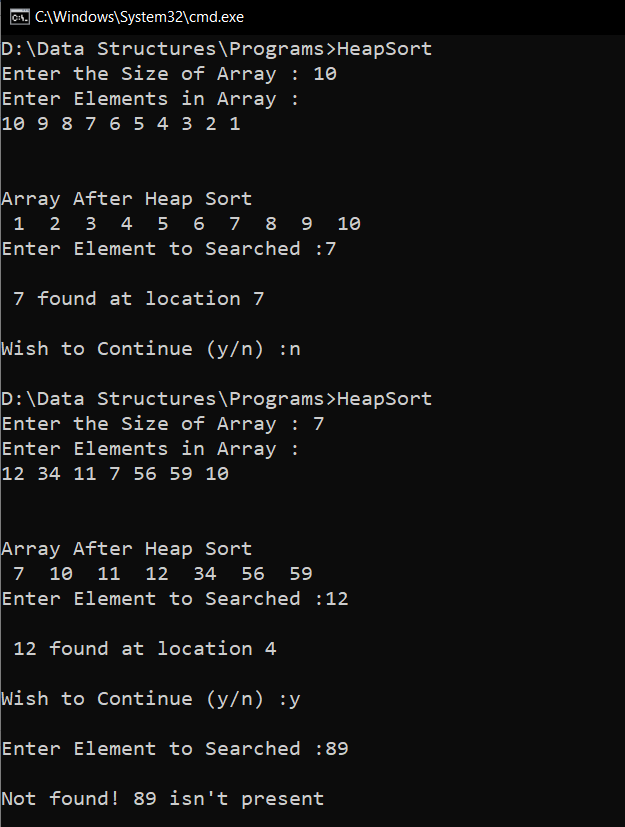
9. endif

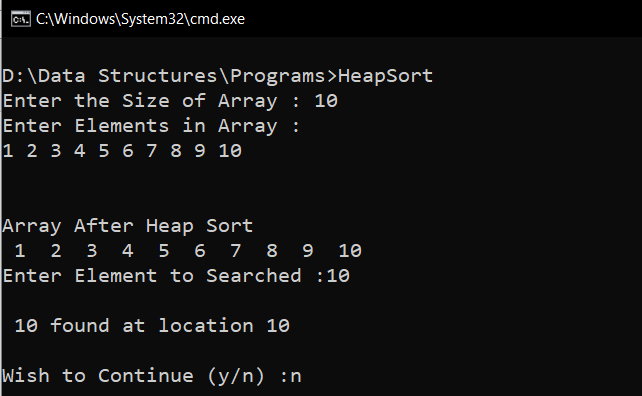
STOP

**PROGRAM:**

#include<stdio.h>  
void create\_heap(int A[],int B[],int n){  
 int i=1;  
 while(i<=n){  
 int x = A[i];  
 B[i] = x;  
 int j = i;  
 while(j>1){  
 if(B[j]>B[j/2]){  
 int temp = B[j];  
 B[j] = B[j/2];  
 B[j/2] = temp;  
 j = j/2;  
 }else{  
 j = 1;  
 }  
 }  
 i++;  
 }  
}  
  
void remove\_max(int B[],int i){  
 int temp = B[i];  
 B[i] = B[1];  
 B[1] = temp;  
}  
  
void rebuild\_heap(int B[],int i){  
 if(i!=1){  
 int j=1;  
 int flag = 1;  
 int temp;  
 while(flag==1){  
 int leftchild = 2\*j;  
 int rightchild = 2\*j+1;  
 if(rightchild<=i){  
 if((B[j]<=B[leftchild])&&(B[rightchild]<=B[leftchild])){  
 temp = B[j];  
 B[j] = B[leftchild];  
 B[leftchild] = temp;  
 j = leftchild;  
 }else if((B[j]<=B[rightchild])&&(B[rightchild]>=B[leftchild])){  
 temp = B[j];  
 B[j] = B[rightchild];  
 B[rightchild] = temp;  
 j = rightchild;  
 }else{  
 flag = 0;  
 }  
 }else if(leftchild<=i){  
 if(B[j]<=B[leftchild]){  
 temp = B[j];  
 B[j] = B[leftchild];  
 B[leftchild] = temp;  
 j = leftchild;  
 }else{  
 flag = 0;  
 }  
 }else{  
 flag=0;  
 }  
 }  
 }  
}  
  
void binary\_search(int B[],int n,int item){  
  
  
 int first = 1;  
 int last = n;  
 int middle = (first+last)/2;  
 while(first<=last) {  
 if(B[middle]<item){  
 first = middle + 1;  
 }  
 else if(B[middle] == item) {  
 printf("\n %d found at location %d \n", item, middle);  
 break;  
 }  
 else{  
 last = middle - 1;  
 }  
 middle = (first + last)/2;  
 }  
 if(first>last){  
 printf("\nNot found! %d isn't present \n", item);  
 }  
}  
  
void main(){  
 int A[100],B[100],n,i,item;  
 char ans;  
 printf("Enter the Size of Array : ");  
 scanf("%d",&n);  
 printf("Enter Elements in Array :\n");  
 for(i=1;i<=n;i++){  
 scanf("%d",&A[i]);  
 }  
 create\_heap(A,B,n);  
 for(i=n;i>1;i--){  
 remove\_max(B,i);  
 rebuild\_heap(B,i-1);  
 }  
 printf("\n\nArray After Heap Sort\n");  
 for(i=1;i<=n;i++){  
 printf(" %d ",B[i]);  
 }  
 do{  
 printf("\nEnter Element to Searched :");  
 scanf("%d", &item);  
 binary\_search(B,n,item);  
 printf("\nWish to Continue (y/n) :");  
 scanf("%c",&ans);  
 scanf("%c",&ans);  
 }while(ans=='y');  
}

**OUTPUT:**





**RESULT:**

Heap Sort was carried out in a set of data.

12/02/2021

**Experiment No:24**

**HASHING**

**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:**

Arrays

**ALGORITHM:**

Algorithm Hashing(using Chaining)

START

1. index = value % 10

2. ptr = hash\_table[index]

3. while (ptr->LINK != NULL)

4. ptr = ptr->LINK

5. endwhile

6. new = GetNode(NODE)

7. new->value = value

8. new->link=NULL

9. ptr->link=new

STOP

Algorithm Hashing(using Linear Probing)

START

1. index = value % size

2. if (hash\_table[index] == ∞)

3. hash\_table[index] = key

4. else

5. for i = index+1 till size

6. if (hash[i] == ∞)

7. hash[i] = key

8. return

9. endfor

10. for i = 0 till index-1

11. if (hash[i] == ∞)

12. hash[i] = key

13. return

14. endfor

15 print "Hash table is full!"

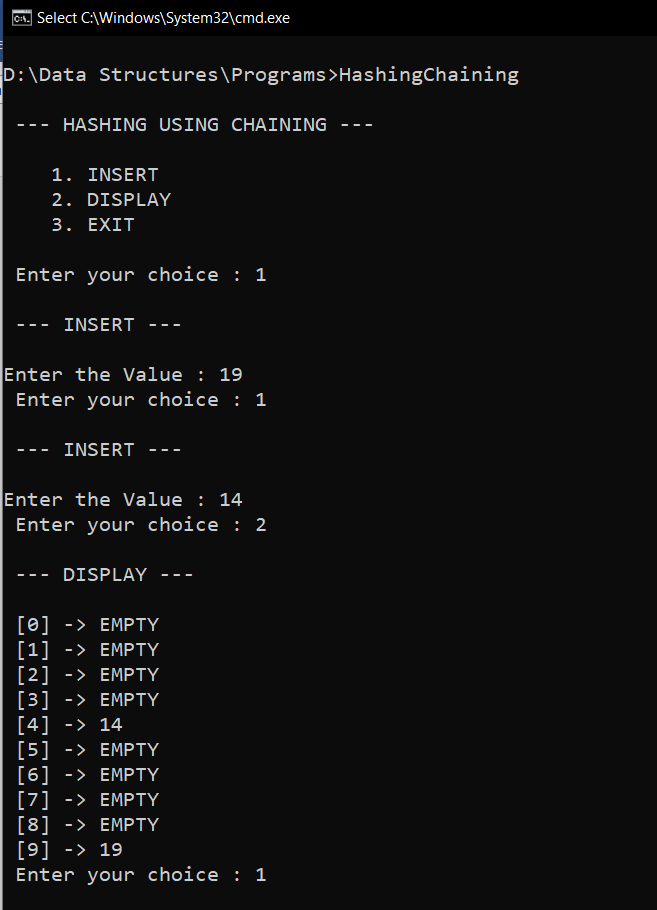
16. endif

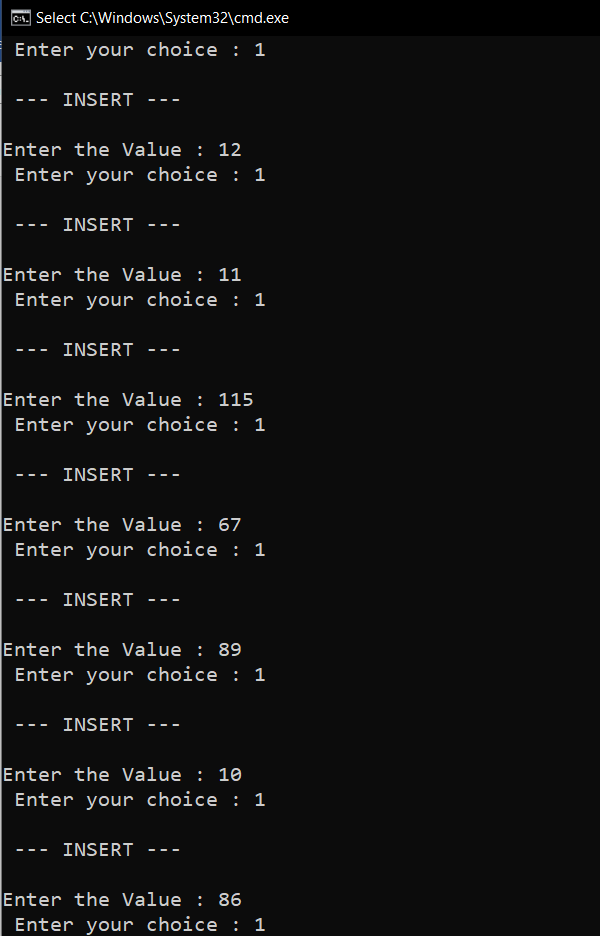
STOP

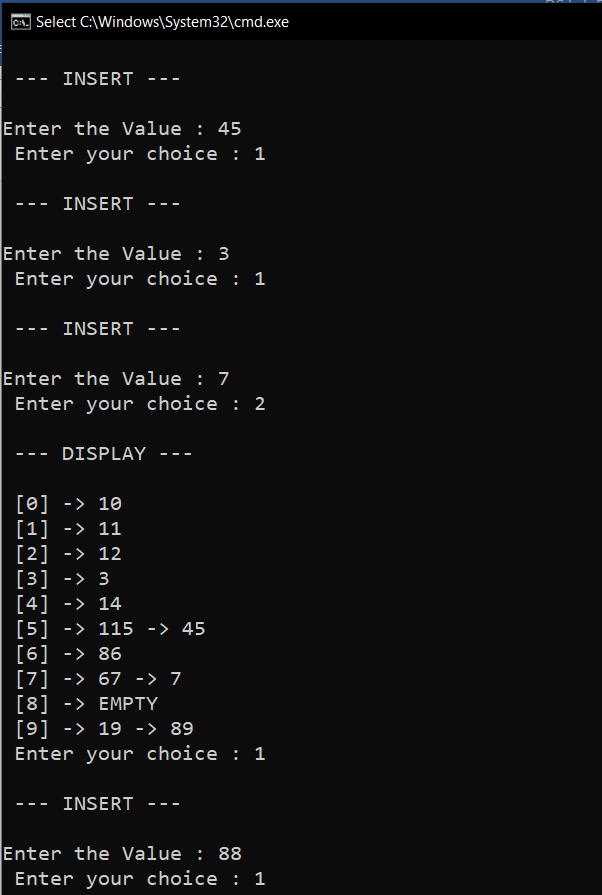
**PROGRAM(1):**

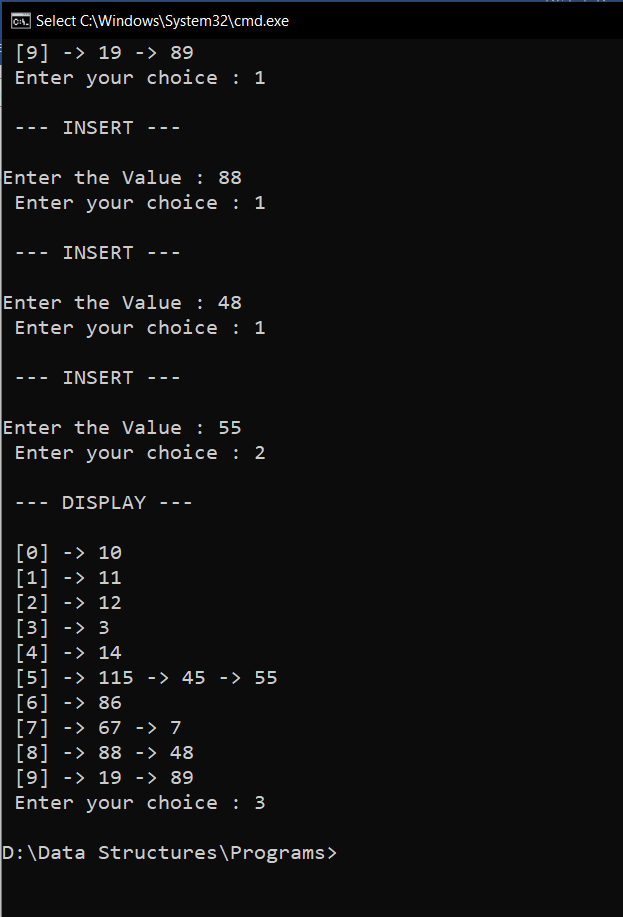
#include<stdio.h>  
#include<stdlib.h>  
struct node {  
 int value;  
 struct node \*link;  
};  
void insert(struct node hash\_table[],int value){  
 int index = value%10;  
 struct node \*ptr = hash\_table;  
 struct node \*new = (struct node\*)malloc(sizeof(struct node));  
 new->link=NULL;  
 new->value = value;  
 ptr=ptr+index;  
 if(ptr->link==NULL){  
 ptr->link=new;  
 }else{  
 while(ptr->link != NULL){  
 ptr=ptr->link;  
 }  
 ptr->link=new;  
 }  
}  
void display(struct node hash\_table[]){  
 struct node \*ptr = hash\_table;  
 for(int i=0;i<10;i++){  
 if(ptr->link==NULL){  
 printf(" [%d] -> EMPTY ",i);  
 }else{  
 printf(" [%d] ",i);  
 struct node \*ptr1=ptr;  
 while(ptr1->link!=NULL){  
 ptr1=ptr1->link;  
 printf("-> %d ",ptr1->value);  
 }  
 }  
 printf("\n");  
 ptr++;  
 }  
  
}  
void main(){  
 int ans=1,op,value,i;  
 struct node hash\_table[10];  
 for(i=0;i<10;i++){  
 hash\_table[i].link=NULL;  
 }  
 printf("\n --- HASHING USING CHAINING --- \n\n");  
 printf(" 1. INSERT \n");  
 printf(" 2. DISPLAY \n");  
 printf(" 3. EXIT \n\n");  
 while(ans==1){  
 printf(" Enter your choice : ");  
 scanf("%d",&op);  
 switch(op){  
 case 1 : printf("\n --- INSERT ---\n\n");  
 printf("Enter the Value : ");  
 scanf("%d",&value);  
 insert(hash\_table,value);  
 break;  
 case 2 : printf("\n --- DISPLAY ---\n\n");  
 display(hash\_table);  
 break;  
 case 3 : ans=0;  
 break;  
 default : printf("\n Enter a Valid Input \n");  
 }  
 }  
}

**OUTPUT(1):**





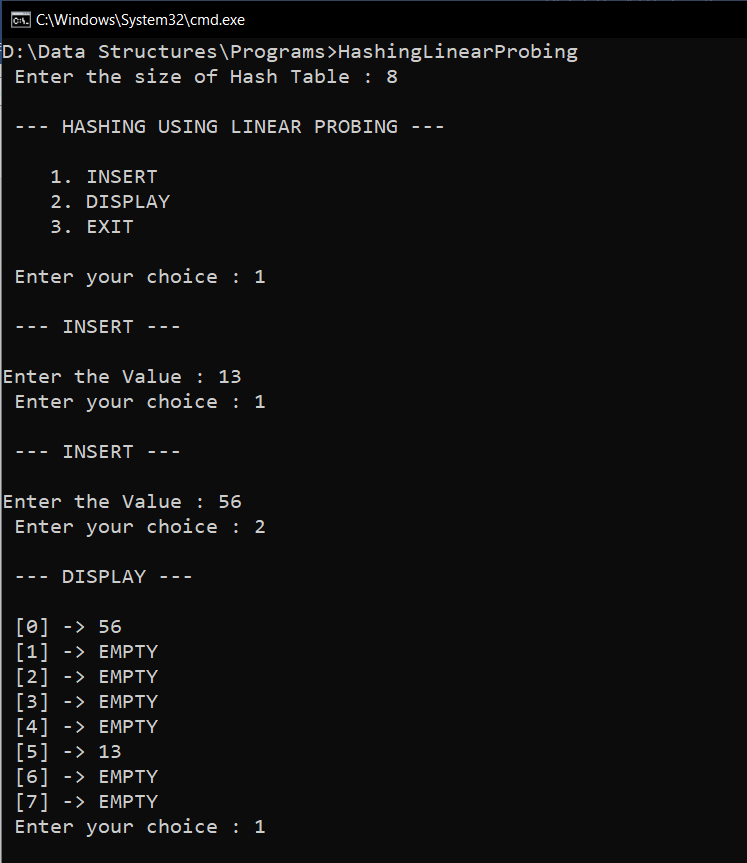


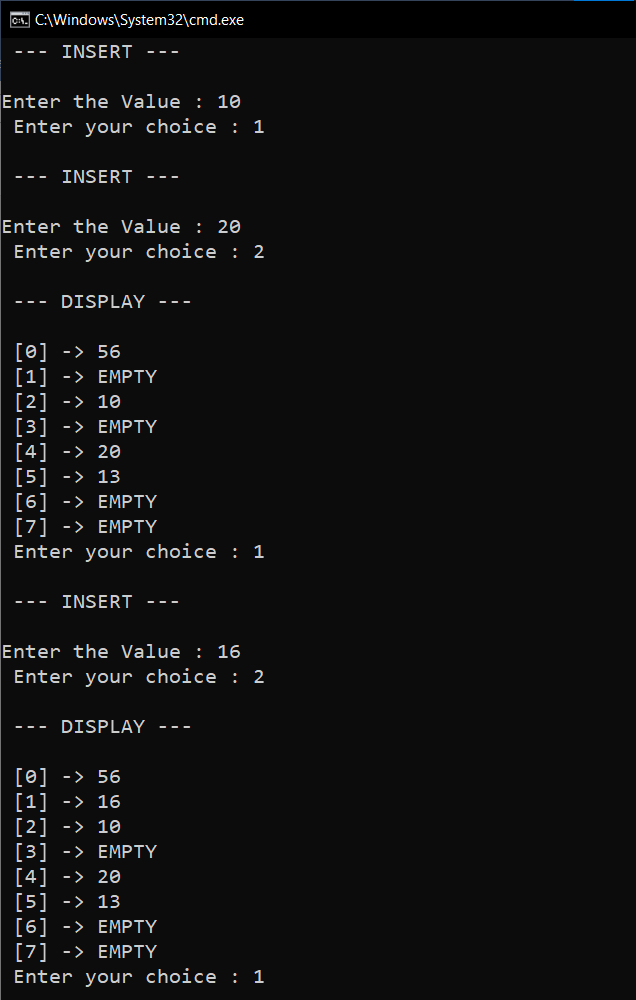


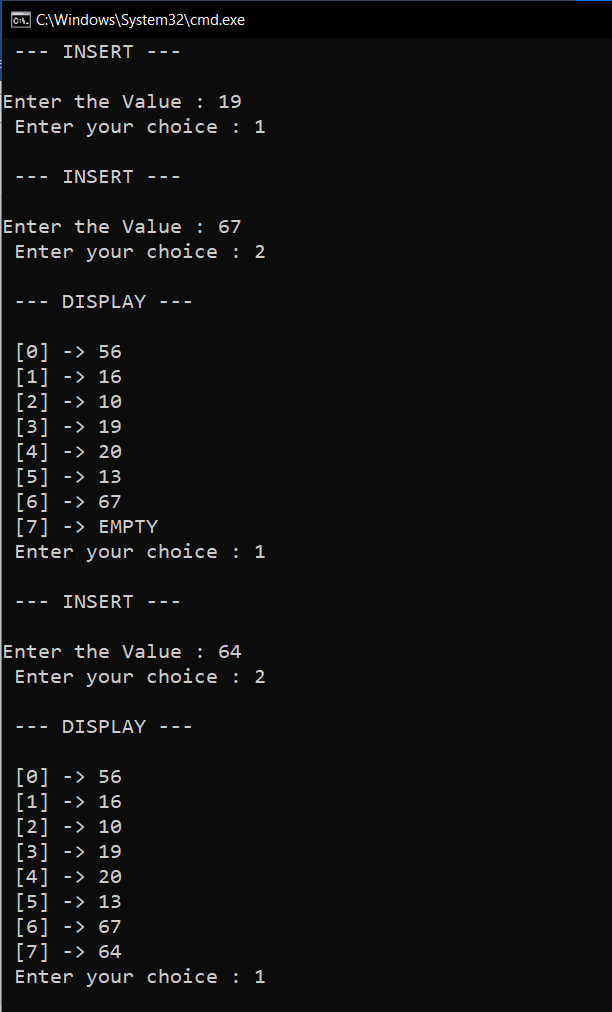
**PROGRAM(2):**

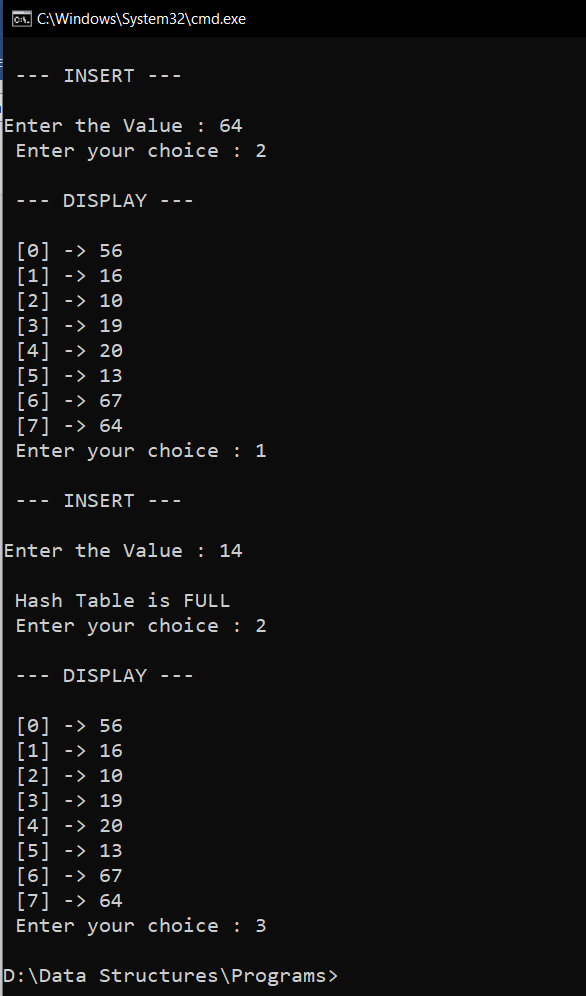
#include<stdio.h>  
#include<stdlib.h>  
#include<math.h>  
void insert(int hash\_table[],int value,int size){  
 int index = value%size;  
 if(hash\_table[index]==(int)INFINITY){  
 hash\_table[index]=value;  
 }else{  
 for(int i= index+1;i<size;i++){  
 if(hash\_table[i]==(int)INFINITY){  
 hash\_table[i]=value;  
 return;  
 }  
 }  
 for(int i=0;i<index;i++){  
 if(hash\_table[i]==(int)INFINITY){  
 hash\_table[i]=value;  
 return;  
 }  
 }  
 printf("\n Hash Table is FULL \n");  
 }  
}  
void display(int hash\_table[],int size){  
  
 for(int i=0;i<size;i++){  
 if(hash\_table[i]==(int)INFINITY){  
 printf(" [%d] -> EMPTY ",i);  
 }else{  
 printf(" [%d] -> %d ",i,hash\_table[i]);  
 }  
 printf("\n");  
 }  
  
}  
void main(){  
 int ans=1,op,value,i,size;  
 printf(" Enter the size of Hash Table : ");  
 scanf("%d",&size);  
 int hash\_table[size];  
 for(i=0;i<size;i++){  
 hash\_table[i]=(int)INFINITY;  
 }  
 printf("\n --- HASHING USING LINEAR PROBING --- \n\n");  
 printf(" 1. INSERT \n");  
 printf(" 2. DISPLAY \n");  
 printf(" 3. EXIT \n\n");  
 while(ans==1){  
 printf(" Enter your choice : ");  
 scanf("%d",&op);  
 switch(op){  
 case 1 : printf("\n --- INSERT ---\n\n");  
 printf("Enter the Value : ");  
 scanf("%d",&value);  
 insert(hash\_table,value,size);  
 break;  
 case 2 : printf("\n --- DISPLAY ---\n\n");  
 display(hash\_table,size);  
 break;  
 case 3 : ans=0;  
 break;  
 default : printf("\n Enter a Valid Input \n");  
 }  
 }  
}

**OUTPUT(2):**









**RESULT:**

Hash tables are implemented using open hashing (chaining) and closed hashing (linear probing).