PROGRAM

```
#include <stdio.h>

int linearSearch(int *A ,int n ,  int x){
        for( int i = 0 ; i < n ; i++){
                if( A[i] == x){
                        return i ;
                        }
                }
        return -1 ;
        }

int main(){
        int n , i , x ;
        printf("ENTER THE SIZE OF ARRAY  : ") ;
        scanf("%d" , &n) ;

        int arr[n] ;
        printf("ENTER THE ARRAY ELEMENTS : ") ;
        for(i = 0 ; i < n ; i++){
                scanf("%d" , &arr[i]) ;
                }

        printf("ENTER THE ELEMENT TO BE SEARCHED : ") ;
        scanf("%d" , &x) ;

        int index = linearSearch(arr, n , x) ;
        if( index == -1){
                printf("ELEMENT NOT FOUND.\n") ;
                }
        else{
                printf("THE POSITION IS : %d\n" , index) ;
                }
        }
```

OUTPUT

```
ENTER THE SIZE OF ARRAY  : 6
ENTER THE ARRAY ELEMENTS : 1 2 3 4 5 6
ENTER THE ELEMENT TO BE SEARCHED : 3
THE POSITION IS : 2
```

DATE : 26-09-2023

LINEAR SEARCH

AIM

To implement linear search algorithm.

PSEUDOCODE

```
INPUT : Array of integers, search key
OUTPUT : Index of element if found or ‘Element not found’


PROCEDURE linearSearch(INT n , INT x, INT A[n])
    FOR i = 0 TO n - 1
        IF A[i] == x THEN
            RETURN i
        END IF
    END FOR
    RETURN -1
END PROCEDURE

PROCEDURE main()
    DECLARE INT n, i, x
    PRINT "ENTER THE SIZE OF ARRAY: "
    READ n

    DECLARE INT arr[n]
    PRINT "ENTER THE ARRAY ELEMENTS: "
    FOR i = 0 TO n - 1
        READ arr[i]
    END FOR

    PRINT "ENTER THE ELEMENT TO BE SEARCHED: "
    READ x

    DECLARE INT index
    SET index = linearSearch(n , x , arr)
    IF index == -1 THEN
        PRINT "ELEMENT NOT FOUND."
    ELSE
        PRINT "THE POSITION IS: ", index
    END IF
END PROCEDURE
```

RESULT

Program Executed Successfully

PROGRAM

```
#include <stdio.h>
int binarySearch(int *A , int l , int h , int x){
        if(l<=h){
                int mid = (l+h)/2 ;
                if( A[mid] == x){
                        return mid ;
                        }
                else if(A[mid] > x){
                        return binarySearch(A ,l , mid-1 ,x) ;
                        }
                else{
                        return binarySearch(A ,mid+1 , h , x) ;
                        }
                }
        return -1 ;
        }

int main(){
        int n , i , x ;

        scanf("%d" , &n) ;
        int arr[n] ;

        for(i = 0 ; i < n ; i++){
                scanf("%d" , &arr[i]) ;
                }

        scanf("%d" , &x) ;
        int index = binarySearch(arr, 0 ,  n-1 , x) ;
        if( index == -1){
                printf("ELEMENT NOT FOUND.\n") ;
                }
        else{
                printf("THE POSITION IS : %d\n" , index) ;
                }
        }
```

OUTPUT

```
ENTER THE SIZE OF ARRAY  : 5
ENTER THE ARRAY ELEMENTS : 1 2 3 4 5
ENTER THE ELEMENT TO BE SEARCHED : 4
THE POSITION IS : 3
```

DATE : 26-09-2023

BINARY SEARCH

AIM

To implement Binary Search algorithm.

PSEUDOCODE

```
INPUT : Ordered array of integers, search key
OUTPUT : Index of element if found or ‘Element not found’


PROCEDURE binarySearch(int A[], INT l, INT h , INT x)
    IF l <= h THEN
        DECLARE INT mid
        SET mid = (l + h) / 2
        IF A[mid] == x THEN
            RETURN mid
        ELSE IF A[mid] > x THEN
            RETURN binarySearch(A, l, mid - 1, x)
        ELSE
            RETURN binarySearch(A, mid + 1, h, x)
        END IF
    END IF
    RETURN -1
END PROCEDURE

PROCEDURE main()
    DECLARE INT n, i, x
    READ n
    DECLARE INT arr[n]
    FOR i = 0 TO n - 1
        READ arr[i]
    END FOR
    READ x
    DECLARE INT index
    SET index = binarySearch(arr, 0, n - 1, x)
    IF index == -1 THEN
        PRINT "ELEMENT NOT FOUND."
    ELSE
        PRINT "THE POSITION IS: ", index
    END IF
END PROCEDURE
```

RESULT

Program Executed Successfully

PROGRAM

```
#include<stdio.h>

void swap(int *x , int *y){
    int temp = *x ;
    *x = *y ;
    *y = temp ;
}

void bubbleSort(int *A , int n){
    int i , j , comp;
    for(i = 0 ; i < n ; i++){
        comp = 0 ;
        for(j = 0 ; j < n-i-1 ; j++){
            if(A[j]>A[j+1]){
                swap(&A[j] , &A[j+1]) ;
                comp++ ;
            }
        }
        if(comp == 0){
            break ;
        }
    }
}

int main(){
        int n , i , x ;
        printf("ENTER THE SIZE OF ARRAY  : ") ;
        scanf("%d" , &n) ;

        int arr[n] ;
        printf("ENTER THE ARRAY ELEMENTS : ") ;
        for(i = 0 ; i < n ; i++){
                scanf("%d" , &arr[i]) ;
                }

    bubbleSort(arr, n );

    printf("THR SORTED ARRAY IS : ") ;
    for(i = 0 ; i < n ; i++){
        printf("%d " , arr[i]) ;
        }
    printf("\n") ;
        }
```

OUTPUT

```
ENTER THE SIZE OF ARRAY  : 5
ENTER THE ARRAY ELEMENTS : 5 4 3 2 1
THR SORTED ARRAY IS : 1 2 3 4 5
```

DATE : 26-09-2023

BUBBLE SORT

AIM

To implement Bubble Sort algorithm.

PSEUDOCODE

```
INPUT : Unsorted array of integers
OUTPUT : Sorted array of integers

PROCEDURE swap(INT *x, INT *y)
    DECLARE INT temp
    SET temp = *x
    *x = *y
    *y = temp
END PROCEDURE
PROCEDURE bubbleSort(INT n INT A[n])
    DECLARE INT i, j, comp
    FOR i = 0 TO n - 1
        SET comp = 0
        FOR j = 0 TO n - i - 1
            IF A[j] > A[j + 1] THEN
                CALL swap(&A[j], &A[j + 1])
                INCREMENT comp BY 1
            END IF
        END FOR
        IF comp == 0 THEN
            BREAK
        END IF
    END FOR
END PROCEDURE
PROCEDURE main()
    DECLARE INT n, i, x
    READ n
    DECLARE INT arr[n]
    FOR i = 0 TO n - 1
        READ arr[i]
    END FOR
    CALL bubbleSort(n , arr)
    FOR i = 0 TO n - 1
        PRINT arr[i]
    END FOR
END PROCEDURE
```

RESULT

Program Executed Successfully

PROGRAM

```
#include <stdio.h>

void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
}
void selectionSort(int *A, int n) {
    int i, j, minIndex;
    for (i = 0; i < n - 1; i++) {
        minIndex = i;
        for (j = i + 1; j < n; j++) {
            if (A[j] < A[minIndex]) {
                minIndex = j;
            }
        }
        if (minIndex != i) {
            swap(&A[i], &A[minIndex]);
        }
    }
}
int main() {
    int n, i;
    printf("ENTER THE SIZE OF ARRAY  : ");
    scanf("%d", &n);

    int arr[n];
    printf("ENTER THE ARRAY ELEMENTS : ");
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    selectionSort(arr, n);
    printf("THE SORTED ARRAY IS : ");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    }
```

OUTPUT

```
ENTER THE SIZE OF ARRAY  : 5
ENTER THE ARRAY ELEMENTS : 1 5 4 3 7
THE SORTED ARRAY IS : 1 3 4 5 7
```

DATE : 26-09-2023

SELECTION SORT

AIM

To implement Selection Sort Algorithm.

PSEUDOCODE

```
INPUT : Unsorted array of integers
OUTPUT : Sorted array of integers


PROCEDURE swap(INT *x , INT *y)
    DECLARE INT temp
    SET temp = *x
    *x = *y
    *y = temp
END PROCEDURE
PROCEDURE selectionSort(INT A[] , INT n)
    DECLARE INT i, j, minIndex
    FOR i = 0 TO n - 1
        SET minIndex = i
        FOR j = i + 1 TO n
            IF A[j] < A[minIndex] THEN
                SET minIndex = j
            END IF
        END FOR
        IF minIndex != i THEN
            CALL swap(&A[i], &A[minIndex])
        END IF
    END FOR
END PROCEDURE
PROCEDURE main()
    DECLARE INT n, i
    READ n
    DECLARE INT arr[n]
    FOR i = 0 TO n - 1
        READ arr[i]
    END FOR
    CALL selectionSort(arr, n)
    FOR i = 0 TO n - 1
        PRINT arr[i]
    END FOR
END PROCEDURE
```

RESULT

Program Executed Successfully

PROGRAM

```
#include <stdio.h>

void insertionSort(int *A, int n) {
    int i, j, key;
    for (i = 1; i < n; i++) {
        key = A[i] ;
        j = i-1 ;

        while (j >= 0 && A[j] > key) {
            A[j + 1] = A[j];
            j-- ;
        }

        A[j+1] = key;
    }
}

int main() {
    int n, i;
    printf("ENTER THE SIZE OF ARRAY  : ");
    scanf("%d", &n);

    int arr[n];
    printf("ENTER THE ARRAY ELEMENTS : ");
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    insertionSort(arr, n);

    printf("THE SORTED ARRAY IS : ");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

}
```

OUTPUT

```
ENTER THE NUMBER OF ELEMENTS : 5
ENTER ARRAY ELEMENTS : 1 2 3 4 5
SECOND LARGEST ELEMENT IS : 4
```

DATE : 03-10-2023

INSERTION SORT

AIM

To implement Insertion Sort Algorithm.

PSEUDOCODE

```
INPUT : Unsorted array of integers
OUTPUT : Sorted array of integers


PROCEDURE insertionSort(INT A[] , INT n)
    DECLARE INT i, j, key
    FOR i = 1 TO n - 1
        SET key = A[i]
        SET j = i - 1

        WHILE j >= 0 AND A[j] > key
            SET A[j + 1] = A[j]
            DECREMENT j BY 1
        END WHILE

        SET A[j + 1] = key
    END FOR
END PROCEDURE

PROCEDURE main()
    DECLARE INT n, i
    READ n

    DECLARE INT arr[n]

    FOR i = 0 TO n - 1
        READ arr[i]
    END FOR

    CALL insertionSort(arr, n)

    FOR i = 0 TO n - 1
        PRINT arr[i]
    END FOR
END PROCEDURE
```

RESULT

Program Executed Successfully

PROGRAM

```
#include <stdio.h>

struct stack{
        int top;
        int size ;
        int arr[100] ;
        };

int isFull(struct stack *st){
        return (st->size == st->top +1) ;
        }

int isEmpty(struct stack *st){
        return (st->top == -1) ;
        }

void push(struct stack *st , int ele){
        if(isFull(st)){
                printf("STACK IS FULL.\n") ;
                }
        else{
                st->arr[++(st->top)] = ele ;
                }
        }

void pop(struct stack *st){
        if(isEmpty(st)){
                printf("STACK IS EMPTY.\n") ;
                }
        else{
                st->arr[(st->top)--];
                }
        }

void peek(struct stack *st){
        if(isEmpty(st)){
                printf("STACK IS EMPTY.") ;
                }
        else{
                printf("%d\n" ,st->arr[st->top]) ;
                }
        }

void display(struct stack *st){
        if(isEmpty(st)){
                printf("STACK IS EMPTY.\n") ;
                }
        else{
                for(int i = st->top ; i >= 0 ; i--){
                        printf("%d\n" , st->arr[i]) ;
                        }
                }
        }

void displayMenu() {
    printf("\n------ MENU ------\n");
```

DATE : 03-10-2023

STACK

AIM

To implement Data Structure Stack.

PSEUDOCODE

```
INPUT : Input of selected operation from menu
OUTPUT :  Output of selected operation

STRUCT stack
    DECLARE INT top
    DECLARE INT size
    DECLARE INT arr[100]
END STRUCT

PROCEDURE isFull(st)
    RETURN (st.size == st.top + 1)
END PROCEDURE

PROCEDURE isEmpty(st)
    RETURN (st.top == -1)
END PROCEDURE

PROCEDURE push(st, ele)
    IF isFull(st) THEN
        PRINT "STACK IS FULL."
    ELSE
        st.arr[++(st.top)] = ele
    END IF
END PROCEDURE

PROCEDURE pop(st)
    IF isEmpty(st) THEN
        PRINT "STACK IS EMPTY."
    ELSE
        st.arr[(st.top)--]
    END IF
END PROCEDURE

PROCEDURE peek(st)
    IF isEmpty(st) THEN
        PRINT "STACK IS EMPTY."
    ELSE
        PRINT st.arr[st.top]
    END IF
END PROCEDURE
```

```
    printf("1. PUSH\n");
    printf("2. POP\n");
    printf("3. PEEK\n");
        printf("4. DISPLAY\n");
    printf("5. EXIT\n");
    printf("--------------------\n");
}

int main() {
    struct stack st;
    st.top = -1;

        printf("ENTER THE SIZE OF STACK : ") ;
        scanf("%d" , &st.size) ;

    int choice = 0, ele;
        displayMenu() ;
    while (choice != 5){
        printf("ENTER THE CHOICE : ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("ENTER THE NUMBER : ");
                scanf("%d", &ele);
                push(&st, ele);
                break;

            case 2:
                pop(&st);
                break;

            case 3:
                peek(&st);
                break;

            case 4:
                display(&st);
                break;

                        case 5:
                                printf("EXITING.\n") ;
                                break ;

            default:
                printf("WRONG CHOICE.\n");
                }
                printf("\n") ;
        }
}
```

```
PROCEDURE display(st)
    IF isEmpty(st) THEN
        PRINT "STACK IS EMPTY."
   ELSE
        FOR i = st.top TO 0 STEP -1
            PRINT st.arr[i]
        END FOR
    END IF
END PROCEDURE

PROCEDURE displayMenu()
    PRINT "------ MENU ------"
    PRINT "1. PUSH"
    PRINT "2. POP"
    PRINT "3. PEEK"
    PRINT "4. DISPLAY"
    PRINT "5. EXIT"
    PRINT "-------------------"
END PROCEDURE

PROCEDURE main()
    DECLARE stack st
    st.top = -1

    PRINT "ENTER THE SIZE OF STACK : "
    READ st.size

    DECLARE INT choice, ele
    CALL displayMenu()
    WHILE choice != 5
        PRINT "ENTER THE CHOICE : "
        READ choice

        SWITCH choice
            CASE 1
                PRINT "ENTER THE NUMBER : "
                READ ele
                CALL push(st, ele)
            CASE 2
                CALL pop(st)
            CASE 3
                CALL peek(st)
            CASE 4
                CALL display(st)
            CASE 5
                PRINT "EXITING."
            DEFAULT
                PRINT "WRONG CHOICE."
        END SWITCH
        PRINT ""
    END WHILE
END PROCEDURE
```

OUTPUT

```
ENTER THE SIZE OF STACK : 4

------ MENU ------
1. PUSH
2. POP
3. PEEK
4. DISPLAY
5. EXIT
-------------------
ENTER THE CHOICE : 1
ENTER THE NUMBER : 12

ENTER THE CHOICE : 1
ENTER THE NUMBER : 24

ENTER THE CHOICE : 1
ENTER THE NUMBER : 36

ENTER THE CHOICE : 1
ENTER THE NUMBER : 48

ENTER THE CHOICE : 1
ENTER THE NUMBER : 60
STACK IS FULL.

ENTER THE CHOICE : 4
48
36
24
12

ENTER THE CHOICE : 2

ENTER THE CHOICE : 2

ENTER THE CHOICE : 4
24
12

ENTER THE CHOICE : 3
24

ENTER THE CHOICE : 5
EXITING.
```

RESULT

Program Executed Successfully

PROGRAM

```
#include <stdio.h>

struct queue{
        int size ;
        int rear ;
        int front ;
        int arr[100] ;
        };

int isFull(struct queue *q){
        return (q->rear == q->size -1) ;
        }

int isEmpty(struct queue *q){
        return ( q->front == q->rear) ;
        }

void enqueue(struct queue *q , int n){
        if(isFull(q)){
                printf("QUEUE IS FULL.\n") ;
                }
        else{
                q->arr[++(q->rear)] = n ;
                }
        }

void dequeue(struct queue *q){
        if(isEmpty(q)){
                 printf("THE QUEUE IS EMPTY.\n") ;
                }
        else{
                q->arr[++(q->front)];
                }
        }

void display(struct queue *q){
        if( isEmpty(q)){
                printf("QUEUE IS EMPTY.\n") ;
                }
        else{

                for( int i = q->front+1 ; i <= q->rear ; i++){
                        printf("%d " , q->arr[i]) ;
                        }
                printf("\n") ;
                }
        }

void displayMenu() {
    printf("\n------ MENU ------\n");
    printf("1. ENQUEUE\n");
    printf("2. DEQUEUE\n");
    printf("3. DISPLAY\n");
    printf("4. EXIT\n");
    printf("-------------------\n");
}
```

DATE : 03-10-2023

QUEUE

AIM

To implement Data Structure Queue.

PSEUDOCODE

```
INPUT : Input of selected operation from menu
OUTPUT :  Output of selected operation

STRUCT queue
    INT size
    INT rear
    INT front
    INT arr[100]
END STRUCT

PROCEDURE isFull(q)
    RETURN (q.rear == q.size - 1)
END PROCEDURE

PROCEDURE isEmpty(q)
    RETURN (q.front == q.rear)
END PROCEDURE

PROCEDURE enqueue(q, n)
    IF isFull(q) THEN
        PRINT "QUEUE IS FULL."
    ELSE
        q.arr[++(q.rear)] = n
    END IF
END PROCEDURE

PROCEDURE dequeue(q)
    IF isEmpty(q) THEN
        PRINT "THE QUEUE IS EMPTY."
    ELSE
        q.arr[++(q.front)]
    END IF
END PROCEDURE

PROCEDURE display(q)
    IF isEmpty(q) THEN
        PRINT "QUEUE IS EMPTY."
    ELSE
        FOR i = q.front + 1 TO q.rear
            PRINT q.arr[i]
        END FOR
        PRINT ""
    END IF
END PROCEDURE
```

```
int main() {
    struct queue q;
    q.front = q.rear = -1;

    printf("ENTER THE SIZE OF QUEUE : ");
    scanf("%d", &q.size);

    int choice = 0, n;
    displayMenu();
    while(choice != 4){
        printf("ENTER CHOICE : ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("ENTER THE ELEMENT : ");
                scanf("%d", &n);
                enqueue(&q, n);
                break;

            case 2:
                dequeue(&q);
                break;

            case 3:
                display(&q);
                break;

            case 4:
                printf("EXITING.\n");
                break;

            default:
                printf("WRONG CHOICE.\n");
        }
        printf("\n");
    }
}
```

```
PROCEDURE displayMenu()
    PRINT "------ MENU ------"
    PRINT "1. ENQUEUE"
    PRINT "2. DEQUEUE"
    PRINT "3. DISPLAY"
    PRINT "4. EXIT"
    PRINT "--------------------"
END PROCEDURE

PROCEDURE main()
    DECLARE queue q
    q.front = q.rear = -1

    PRINT "ENTER THE SIZE OF QUEUE : "
    READ q.size

    DECLARE INT choice, n
    CALL displayMenu()
    WHILE choice != 4
        PRINT "ENTER CHOICE : "
        READ choice

        SWITCH choice
            CASE 1
                PRINT "ENTER THE ELEMENT : "
                READ n
                CALL enqueue(q, n)
            CASE 2
                CALL dequeue(q)
            CASE 3
                CALL display(q)
            CASE 4
                PRINT "EXITING."
            DEFAULT
                PRINT "WRONG CHOICE."
        END SWITCH
        PRINT ""
    END WHILE
END PROCEDURE
SPARSE MATRIX REPRESENTATION
PROCEDURE insertElements(m, n, A)
    DECLARE INT i,j
    PRINT "ENTER THE ELEMENTS"
    FOR i = 0 TO m - 1
        PRINT "ENTER THE ROW ", i + 1, " : "
        FOR j = 0 TO n - 1
            READ A[i][j]
        END FOR
    END FOR
END PROCEDURE

PROCEDURE display(m, n, A)
    DECLARE INT i,j
    PRINT "THE SPARSE MATRIX IS :"
    FOR i = 0 TO m - 1
        FOR j = 0 TO n - 1
            PRINT A[i][j], " "
        END FOR
        PRINT "\n"
```

OUTPUT

```
------ MENU ------
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
-------------------
ENTER CHOICE : 1
ENTER THE ELEMENT : 12

ENTER CHOICE : 1
ENTER THE ELEMENT : 13

ENTER CHOICE : 1
ENTER THE ELEMENT : 14

ENTER CHOICE : 1
ENTER THE ELEMENT : 15
QUEUE IS FULL.

ENTER CHOICE : 3
12 13 14

ENTER CHOICE : 2

ENTER CHOICE : 3
13 14

ENTER CHOICE : 2

ENTER CHOICE : 3
14

ENTER CHOICE : 4
EXITING.
```

```
    END FOR
END PROCEDURE

PROCEDURE convertToThreeTuple(r, c, A, C)
    DECLARE INT i,j,k = 1

    FOR i = 0 TO r - 1
        FOR j = 0 TO c - 1
            IF A[i][j] != 0 THEN
                C[k][0] = i
                C[k][1] = j
                C[k][2] = A[i][j]
                INCREMENT k
            END IF
        END FOR
    END FOR

    C[0][0] = r
    C[0][1] = c
    C[0][2] = k - 1
    RETURN k
END PROCEDURE

PROCEDURE main()
    DECLARE INT r, c
    PRINT "ENTER THE ROW AND COLUMN OF MATRIX : "
    READ r, c

    DECLARE INT A[r][c], C[r * c][3]

    insertElements(r, c, A)

    DECLARE INT k1 = convertToThreeTuple(r, c, A, C)

    display(k1, 3, C)
END PROCEDURE
```

RESULT

Program Executed Successfully

PROGRAM

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

int stack[100] ;
int top = -1 ;

void push(int x){
    stack[++top] = x ;
}

int pop(){
    return stack[top--] ;
}

int isOperator(char ch){
    if(ch == '+' || ch == '-' || ch == '*' || ch == '/')
        return 1;
    return 0;
}

int Precedence(char ch){
    if(ch == '*' || ch == '/')
        return 2;
    else if(ch == '+' || ch == '-')
        return 1;
    return 0;
}

int operation(int a , int b , char ch){
    if(ch == '+')
        return a + b ;
    else if(ch == '-')
        return a - b ;
    else if(ch == '*')
        return a * b ;
    else if(ch == '/')
        return a / b ;
}

int postfixEval(char postfix[]){
    int res = 0 ;
    for( int i = 0 ; i< strlen(postfix) ; i++){
        if(isOperator(postfix[i]) == 0){
            push(postfix[i] - '0') ;
        }
        else{
            int a = pop() ;
            int b = pop() ;
            res = operation(b , a , postfix[i]) ;
            push(res) ;
        }
    }
    return res ;
}
```

DATE : 03-10-2023

POSTFIX EVALUATION

AIM

To evaluate a Postfix Expression Using Stack.

PSEUDOCODE

```
INPUT : Postfix Expression
OUTPUT :  Solution  of postfix expression

DECLARE INT STACK[100] ;
DECLARE INT TOP
SET TOP = -1

PROCEDURE push(INT x)
    stack[++top] = x
END PROCEDURE

PROCEDURE pop()
    RETURN stack[top--]
END PROCEDURE

PROCEDURE isOperator(CHAR ch) : integer
    IF ch is '+' OR ch is '-' OR ch is '*' OR ch is '/'
        RETURN 1
    ELSE
        RETURN 0
END PROCEDURE

PROCEDURE Precedence(CHAR ch) : integer
    IF ch is '*' OR ch is '/'
        RETURN 2
    ELSE IF ch is '+' OR ch is '-'
        RETURN 1
    ELSE
        RETURN 0
END PROCEDURE

PROCEDURE operation(INT a , INT b , CHAR ch) : integer
    IF ch is '+'
        RETURN a + b
    ELSE IF ch is '-'
        RETURN a - b
    ELSE IF ch is '*'
        RETURN a * b
    ELSE IF ch is '/'
        RETURN a / b
END PROCEDURE
```

```
int main(){
    char postfix[100] ;
    printf("ENTER THE EXPRESSION: ") ;
    scanf("%s" , postfix) ;
    int res = postfixEval(postfix) ;
    printf("THE RESULT IS : %d\n" , res) ;
    return 0 ;
}
```

OUTPUT

```
ENTER THE EXPRESSION: 231*+9-
THE RESULT IS : -4
```

```
PROCEDURE postfixEval(CHAR POSTFIX[]) : integer
    DECLARE INT res
    SET res = 0

    FOR i = 0 TO length of postfix - 1
        IF isOperator(postfix[i]) is 0 THEN
            push(postfix[i] - '0')
        ELSE
            DECLARE INT a
            DECLARE INT b
            SET a = pop()
            SET b = pop()
            SET res = operation(b, a, postfix[i])
            push(res)
        END IF
    END FOR

    RETURN res
END PROCEDURE

PROCEDURE main()
    DECLARE CHAR postfix[]
    PRINT "ENTER THE EXPRESSION: "
    READ postfix
    DECLARE INT res
    SET res = postfixEval(postfix)
    PRINT "THE RESULT IS : " + res
END PROCEDURE
```

RESULT

Program Executed Successfully

PROGRAM

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

int isOperator(char ch) {
    if (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '^')
        return 1;
    return 0;
}

int Precedence(char ch) {
    if (ch == '*' || ch == '/')
        return 2;
    else if (ch == '+' || ch == '-')
        return 1;
    else if(ch == '^')
        return 3;
    return 0;
}

void infixToPostfix(char infix[], char postfix[]) {
    char stack[100];
    int top = -1;
    int i, j = 0;

    for( i = 0 ; i < strlen(infix) ; i++){
        if(infix[i] == '('){
            stack[++top] = infix[i];
        }
        else if(infix[i] == ')'){
            while(stack[top] != '('){
                postfix[j++] = stack[top--];
            }
            top--;
        }
        else if(isOperator(infix[i])){
            while(top != -1 && Precedence(stack[top]) >=
Precedence(infix[i])){
                postfix[j++] = stack[top--];
            }
            stack[++top] = infix[i];
        }
        else{
            postfix[j++] = infix[i];
        }
    }

    while(top != -1){
        postfix[j++] = stack[top--];
    }
    postfix[j] = '\0';

}

int main(){
    char infix[100];
```

DATE : 10-10-2023

INFIX TO POSTFIX

AIM

To convert an Infix Expression to Postfix Expression.

PSEUDOCODE

```
INPUT : Infix Expression
OUTPUT :  Corresponding Postfix Expression

PROCEDURE isOperator(CHAR ch)
    IF ch is '+' OR ch is '-' OR ch is '*' OR ch is '/' OR ch is '^'
        RETURN 1
    ELSE
        RETURN 0
END PROCEDURE

PROCEDURE Precedence(CHAR ch)
    IF ch is '*' OR ch is '/'
        RETURN 2
    ELSE IF ch is '+' OR ch is '-'
        RETURN 1
    ELSE IF ch is '^'
        RETURN 3
    ELSE
        RETURN 0
END PROCEDURE

PROCEDURE infixToPostfix(CHAR INFIX[] . CHAR POSTFIX[])
    DECLARE CHAR stack[100]
    DECLARE INT top
    SET top = -1
    DECLARE INT i, j
    SET j = 0

    FOR i = 0 TO length of infix - 1
        IF infix[i] is '(' THEN
            SET stack[++top] = infix[i]
        ELSE IF infix[i] is ')' THEN
            WHILE stack[top] !=  '('
                SET postfix[j++] = stack[top--]
            END WHILE
            SET top = top - 1
        ELSE IF isOperator(infix[i]) THEN
            WHILE top is not -1 AND Precedence(stack[top]) >=
Precedence(infix[i])
                SET postfix[j++] = stack[top--]
            END WHILE
            SET stack[++top] = infix[i]
```

```
    printf("ENTER THE INFIX EXPRESSION: ");
    char postfix[100];
    fgets(infix, 100, stdin);

    infixToPostfix(infix, postfix);

    printf("THE POSTFIX EXPRESSION IS : %s\n", postfix);

    return 0;
}
```

OUTPUT

```
ENTER THE INFIX EXPRESSION: ((A+B)-C*(D/E))+F
THE POSTFIX EXPRESSION IS : AB+-CDE/*F+
```

```
        ELSE
            SET postfix[j++] = infix[i]
        END IF
    END FOR
    WHILE top is not -1
        SET postfix[j++] = stack[top--]
    END WHILE
    SET postfix[j] = '\0'
END PROCEDURE

PROCEDURE main()
    DECLARE CHAR infix[100]
    DECLARE CHAR postfix[100]

    PRINT "ENTER THE INFIX EXPRESSION: "
    READ infix

    CALL infixToPostfix(infix, postfix)

    PRINT "THE POSTFIX EXPRESSION IS : " + postfix
END PROCEDURE
```

RESULT

Program Executed Successfully

PROGRAM

```
#include <stdio.h>

struct term{
        int exp ;
        int coeff ;
        };

struct poly{
        int n ;
        struct term arr[500] ;
        };

int main(){
        struct poly p1 , p2 , p3;
        p3.n = 0 ;

        printf("ENTER THE NUMBER OF TERMS IN P1 : ") ;
        scanf("%d" , &p1.n) ;
        for(int i = 0 ;i < p1.n ; i++){
                printf("ENTER THE COEFF AS WELL AS EXP : ") ;
                scanf("%d %d" , &p1.arr[i].coeff , &p1.arr[i].exp) ;
                }
        printf("\n") ;

        printf("ENTER THE NUMBER OF TERMS IN P2 :" ) ;
        scanf("%d" , &p2.n) ;
        for( int i = 0 ; i< p2.n ; i++){
                printf("ENTER THE COEFF AS WELL AS EXP : " ) ;
                scanf("%d %d" , &p2.arr[i].coeff , &p2.arr[i].exp) ;
                }

        printf("\nFIRST  :") ;
        for( int i = 0 ; i < p1.n ; i++){
                printf("%dx^%d",p1.arr[i].coeff, p1.arr[i].exp) ;
                i==p1.n-1 ? printf(" ") : printf(" + ") ;
                }

        printf("\nSECOND :") ;
        for( int i = 0 ; i < p2.n ; i++){
                printf("%dx^%d",p2.arr[i].coeff, p2.arr[i].exp) ;
                i==p2.n-1 ? printf(" ") : printf(" + ") ;
                }

        int i = 0 , j = 0 , k = 0 ;
        while(i < p1.n && j < p2.n){
                if( p1.arr[i].exp == p2.arr[j].exp){
                        p3.arr[k].exp = p1.arr[i].exp ;
                        p3.arr[k++].coeff = p1.arr[i++].coeff+p2.arr[j+
+].coeff ;
                        }
                else if( p1.arr[i].exp < p2.arr[j].exp){
                        p3.arr[k].exp = p1.arr[i].exp ;
                        p3.arr[k++].coeff = p1.arr[i++].coeff ;
                        }
                else{
                        p3.arr[k].exp = p2.arr[j].exp ;
```

DATE : 10-10-2023

POLYNOMIAL ADDITION-I

AIM

To add two Polynomials using Arrays.

PSEUDOCODE

```
INPUT : Degree and coefficients of each term of both polynomials.
OUTPUT :  Sum of inputted polynomials.

STRUCT term
    DECLARE INT exp, coeff
END STRUCT

STRUCT poly
    DECLARE INT n
    DECLARE term arr[500]
END STRUCT

PROCEDURE main()
    DECLARE poly p1, p2, p3
    DECLARE INT i, j, k
    SET p3.n = 0

    PRINT "ENTER THE NUMBER OF TERMS IN P1: "
    READ p1.n

    FOR i = 0 TO p1.n - 1
        PRINT "ENTER THE COEFF AS WELL AS EXP: "
        READ p1.arr[i].coeff, p1.arr[i].exp
    END FOR

    PRINT "ENTER THE NUMBER OF TERMS IN P2: "
    READ p2.n

    FOR i = 0 TO p2.n - 1
        PRINT "ENTER THE COEFF AS WELL AS EXP: "
        READ p2.arr[i].coeff, p2.arr[i].exp
    END FOR

    PRINT "FIRST: "
    FOR i = 0 TO p1.n - 1
        PRINT p1.arr[i].coeff, "x^", p1.arr[i].exp
        IF i == p1.n - 1
            PRINT " "
        ELSE
            PRINT " + "
        END IF
```

```
                        p3.arr[k++].coeff = p2.arr[j++].coeff ;
                        }
                (p3.n)++ ;
                }
        for(i = i ; i< p1.n ; i++){
                p3.arr[k].exp = p1.arr[i].exp ;
                p3.arr[k++].coeff = p1.arr[i].coeff ;
                p3.n++ ;
                }
        for( j = j ; j < p2.n ; j++){
                p3.arr[k].exp = p2.arr[j].exp ;
                p3.arr[k++].coeff = p2.arr[j].coeff ;
                p3.n++ ;
                }

        printf("\nSUM :") ;
        for( int i = 0 ; i < p3.n ; i++){
                printf("%dx^%d",p3.arr[i].coeff , p3.arr[i].exp) ;
                i == (p3.n)-1 ? printf("\n") : printf(" + ") ;
                }
        }
```

OUTPUT

```
ENTER THE NUMBER OF TERMS IN P1 : 3
ENTER THE COEFF AS WELL AS EXP : 1 1
ENTER THE COEFF AS WELL AS EXP : 2 2
ENTER THE COEFF AS WELL AS EXP : 3 3

ENTER THE NUMBER OF TERMS IN P2 :4
ENTER THE COEFF AS WELL AS EXP : 2 1
ENTER THE COEFF AS WELL AS EXP : 3 2
ENTER THE COEFF AS WELL AS EXP : 4 3
ENTER THE COEFF AS WELL AS EXP : 5 4

FIRST  :1x^1 + 2x^2 + 3x^3
SECOND :2x^1 + 3x^2 + 4x^3 + 5x^4
SUM :3x^1 + 5x^2 + 7x^3 + 5x^4
```

```
    END FOR
    FOR i = 0 TO p2.n - 1
        PRINT p2.arr[i].coeff, "x^", p2.arr[i].exp
        IF i == p2.n - 1
            PRINT " "
        ELSE
            PRINT " + "
        END IF
    END FOR
    SET i = 0, j=0 , k=0
    WHILE i < p1.n AND j < p2.n
        IF p1.arr[i].exp == p2.arr[j].exp
            SET p3.arr[k].exp = p1.arr[i].exp
            SET p3.arr[k].coeff = p1.arr[i].coeff + p2.arr[j].coeff
            INCREMENT i BY 1
            INCREMENT j BY 1
        ELSE IF p1.arr[i].exp < p2.arr[j].exp
            SET p3.arr[k].exp = p1.arr[i].exp
            SET p3.arr[k].coeff = p1.arr[i].coeff
            INCREMENT i By 1
        ELSE
            SET p3.arr[k].exp = p2.arr[j].exp
            SET p3.arr[k].coeff = p2.arr[j].coeff
            INCREMENT j BY 1
        END IF
        INCREMENT k BY 1 AND INCREMENT p3.n BY 1
    END WHILE
    FOR i = i TO p1.n - 1
        SET p3.arr[k].exp = p1.arr[i].exp
        SET p3.arr[k].coeff = p1.arr[i].coeff
        INCREMENT k BY 1
        INCREMENT p3.n BY 1
    END FOR
    FOR j = j TO p2.n - 1
        SET p3.arr[k].exp = p2.arr[j].exp
        SET p3.arr[k].coeff = p2.arr[j].coeff
        INCREMENT k BY 1
        INCREMENT p3.n BY 1
    END FOR
    PRINT "SUM: "
    FOR i = 0 TO p3.n - 1
        PRINT p3.arr[i].coeff, "x^", p3.arr[i].exp
        IF i == p3.n - 1
            PRINT "\n"
        ELSE
            PRINT " + "
        END IF
    END FOR
END PROCEDURE
```

RESULT

```
Program Executed Successfully
```

PROGRAM

```
#include<stdio.h>
#include<stdbool.h>
int Msize=30,f=-1,r=-1;
int A[30];
bool isEmpty(){
    return (f==-1 || f>r);
}
bool isFull(){
    return (r==Msize-1);
}
void deQueue(){
    if(isEmpty()){
        printf("Queue Underflow\n");
    }
    else{
        printf("Dequeued element is %d",A[f++]);
    }
}
void enQueue(){
    int key;
    if(isEmpty()){
        printf("Enter element : ");
        f++;
        r++;
        scanf("%d",&A[r]);
    }
    else if(isFull()){
        printf("Queue Overflow\n");
    }
    else{
        printf("Enter elements : ");
        scanf("%d",&A[++r]);
        key = A[r];
        int j = r - 1;
        while(j>=0 && A[j] > key){
            A[j+1] = A[j];
            j = j - 1;
        }
        A[j+1] = key ;
    }
}
void size(){
    if(isEmpty()){
        printf("Queue Underflow");
    }
    else{
        printf("Size is %d\n",r-f+1);
    }
}
void display(){
    if(isEmpty()){
        printf("Queue Underflow\n");
    }
    else{
        printf("The queue is :\n");
        for (int i=f;i<=r;i++){
             printf("%d  ",A[i]);
```

DATE : 10-10-2023

PRIORITY QUEUE

AIM

To implement The Data Structure Priority Queue.

PSEUDOCODE

```
INPUT : Input of selected operation from menu
OUTPUT :  Output of selected operation

DEFINE SIZE 5

DECLARE ARRAY queue[SIZE]
SET front = -1
SET rear = -1

PROCEDURE ISEMPTY
    RETURN front == -1 OR front>rear
END PROCEDURE

PROCEDURE ISFULLINPUT : Degree and coefficients of each term of both
polynomials.
OUTPUT :  Sum of inputted polynomials.
    RETURN rear == SIZE - 1
END PROCEDURE

PROCEDURE ENQUEUE
    IF ISEMPTY
        SET front = 0
        SET rear = 0
        READ key
        SET queue[rear] = key
    ELSE IF ISFULL
        PRINT "Queue Overflow"
    ELSE
        SET rear = rear + 1
        READ key
        SET j = rear - 1
        WHILE j >= 0 AND queue[j] > key
            SET queue[j + 1] = queue[j]
            SET j = j - 1
        END WHILE
        SET queue[j + 1] = key
    END IF
END PROCEDURE

PROCEDURE DEQUEUE
    IF ISEMPTY
```

```
        }
        printf("\n");
    }
}
int main(){
    int ch,c=0;
    printf("ASCENDING ORDER PRIORITY QUEUE\n");
    printf("1. Display\n");
    printf("2. Enqueue\n");
    printf("3. Dequeue\n");
    printf("4. Size\n");
    printf("5. Exit\n\n");
    while (true){
        printf("Enter choice :");
        scanf("%d",&ch);
        switch (ch){
            case 1 :display();break;
            case 2 :enQueue();break;
            case 3 :deQueue();break;
            case 4 :size();break;
            case 5 :return 0;break;
            default:printf("Invalid Choice\n");
        }
        printf("\n");
    }
}
```

OUTPUT

```
ASCENDING ORDER PRIORITY QUEUE

1. Display
2. Enqueue
3. Dequeue
4. Size
5. Exit

Enter choice :2
Enter element : 4

Enter choice :2
Enter elements : 1

Enter choice :2
Enter elements : 6

Enter choice :1
The queue is : 1  4  6
Enter choice :3
Dequeued element is 1
Enter choice :4
Size is 2

Enter choice :5
```

```
        PRINT "Queue underflow"
    ELSE
        PRINT queue[front] " is dequeued"
        SET front = front + 1
    END IF
END PROCEDURE

PROCEDURE SIZE
    IF ISEMPTY
        PRINT "Queue underflow"
    ELSE
        PRINT rear - front + 1
    END IF
END PROCEDURE

PROCEDURE DISPLAY
    IF !ISEMPTY
        FOR i = front TO rear
            PRINT queue[i]
            INCREMENT i
        END FOR
    ELSE
        PRINT "Queue underflow"
    END IF
END PROCEDURE
PROCEDURE MAIN
    PRINT "1. Display"
    PRINT "2. Enqueue"
    PRINT "3. Dequeue"
    PRINT "4. Size"
    PRINT "5. Exit"
    WHILE TRUE
        READ choice
        CASE choice OF
            1: CALL DISPLAY
            2: CALL ENQUEUE
            3: CALL DEQUEUE
            4: CALL SIZE
            5: CALL EXIT
        END CASE
    END WHILE
END PROCEDURE
END PROGRAM
```

RESULT

Program Executed Successfully

PROGRAM

```
#include<stdio.h>

void insertElements(int m , int n , int A[m][n]){
    printf("ENTER THE ELEMENTS  \n") ;
    for( int i = 0 ; i < m ; i++){
        printf("ENTER THE ROW %d : " , i+1) ;
        for( int j = 0 ; j < n ; j++){
            scanf("%d" , &A[i][j]) ;
        }
    }
}
void display(int m , int n , int A[m][n]){
    printf("THE SPARSE MATRIX IS : \n") ;
    for( int i = 0 ; i < m ; i++){
        for( int j = 0 ; j < n ; j++){
            printf("%d " , A[i][j]) ;
        }
        printf("\n") ;
    }
}
int convertToThreeTuple(int r , int c, int A[r][c] , int C[101][3]){
    int k = 1 ;
    for( int i = 0 ; i < r ; i++){
        for( int j = 0 ; j < c ; j++){
            if(A[i][j] !=0){
                C[k][0] = i ;
                C[k][1] = j ;
                C[k][2] = A[i][j] ;
                k++ ;
            }
        }
    }
    C[0][0] = r ;
    C[0][1] = c ;
    C[0][2] = k-1 ;
    return k ;
}
int addSparseMatrices(int A[][3], int B[][3], int C[][3]) {
    int i = 1, j = 1, k = 1;
    int m = A[0][2], n = B[0][2];
    while(i <= m && j <= n) {
        if(A[i][0] < B[j][0] || (A[i][0] == B[j][0] && A[i][1] < B[j][1])) {
            C[k][0] = A[i][0];
            C[k][1] = A[i][1];
            C[k][2] = A[i][2];
            i++;
        }
        else if(A[i][0] > B[j][0] || (A[i][0] == B[j][0] && A[i][1] > B[j][1]))
{
            C[k][0] = B[j][0];
            C[k][1] = B[j][1];
            C[k][2] = B[j][2];
            j++;
        }
        else {
            C[k][0] = A[i][0];
            C[k][1] = A[i][1];
            C[k][2] = A[i][2] + B[j][2];
            i++;
```

DATE : 20-10-2023

SPARSE MATRIX ADDITION

AIM

To add two Sparse Matrices.

PSEUDOCODE

```
INPUT : Order and elements of matrices to be added
OUTPUT :  Tuple representation of matrix after addition

FUNCTION insertElements(m, n, A)
    DECLARE INT i,j
    PRINT "ENTER THE ELEMENTS"
    FOR i = 0 TO m - 1
        PRINT "ENTER THE ROW ", i + 1, " : "
        FOR j = 0 TO n - 1
            READ A[i][j]
        END FOR
    END FOR
END FUNCTION

FUNCTION display(m, n, A)
    DECLARE INT i,j
    PRINT "THE SPARSE MATRIX IS :"
    FOR i = 0 TO m - 1
        FOR j = 0 TO n - 1
            PRINT A[i][j], " "
        END FOR
        PRINT "\n"
    END FOR
END FUNCTION

FUNCTION convertToThreeTuple(r, c, A, C)
    DECLARE INT k = 1,i,j

    FOR i = 0 TO r - 1
        FOR j = 0 TO c - 1
            IF A[i][j] != 0 THEN
                C[k][0] = i
                C[k][1] = j
                C[k][2] = A[i][j]
                INCREMENT k
            END IF
        END FOR
    END FOR

    C[0][0] = r
    C[0][1] = c
```

```
            j++;
        }
        k++;
    }
    while(i <= m) {
        C[k][0] = A[i][0];
        C[k][1] = A[i][1];
        C[k][2] = A[i][2];
        i++;
        k++;
    }
    while(j <= n) {
        C[k][0] = B[j][0];
        C[k][1] = B[j][1];
        C[k][2] = B[j][2];
        j++;
        k++;
    }
    C[0][0] = A[0][0];
    C[0][1] = A[0][1];
    C[0][2] = k - 1;
    return k;
}

int main(){
    int r , c ;
    printf("ENTER THE ROW AND COLUMN OF MATRICES : ") ;
    scanf("%d %d" , &r , &c) ;

    int A[r][c] , B[r][c], C[r*c][3] , D[r*c][3], result[r*c][3];

    insertElements(r , c ,A) ;
    insertElements(r , c ,B) ;

    int k1 = convertToThreeTuple(r , c , A , C) ;
    int k2 = convertToThreeTuple(r , c , B , D) ;
    int k3 = addSparseMatrices(C, D, result);

    display(k3 , 3, result) ;

}
```

```
    C[0][2] = k - 1
    RETURN k
END FUNCTION
FUNCTION addSparseMatrices(A, B, C)
    DECLARE INT i = 1, j = 1, k = 1
    DECLARE INT m = A[0][2], n = B[0][2]

    WHILE i <= m AND j <= n
        IF A[i][0] < B[j][0] OR (A[i][0] == B[j][0] AND A[i][1] <
B[j][1]) THEN
            C[k][0] = A[i][0]
            C[k][1] = A[i][1]
            C[k][2] = A[i][2]
            INCREMENT i
        ELSE IF A[i][0] > B[j][0] OR (A[i][0] == B[j][0] AND A[i][1]
> B[j][1]) THEN
            C[k][0] = B[j][0]
            C[k][1] = B[j][1]
            C[k][2] = B[j][2]
            INCREMENT j
        ELSE
            C[k][0] = A[i][0]
            C[k][1] = A[i][1]
            C[k][2] = A[i][2] + B[j][2]
            INCREMENT i
            INCREMENT j
        END IF
        INCREMENT k
    END WHILE

    WHILE i <= m
        C[k][0] = A[i][0]
        C[k][1] = A[i][1]
        C[k][2] = A[i][2]
        INCREMENT i
        INCREMENT k
    END WHILE

    WHILE j <= n
        C[k][0] = B[j][0]
        C[k][1] = B[j][1]
        C[k][2] = B[j][2]
        INCREMENT j
        INCREMENT k
    END WHILE

    C[0][0] = A[0][0]
    C[0][1] = A[0][1]
    C[0][2] = k - 1
    RETURN k
END FUNCTION
```

OUTPUT

```
ENTER THE ROW AND COLUMN OF MATRICES : 3 3
ENTER THE ELEMENTS
ENTER THE ROW 1 : 1 0 0
ENTER THE ROW 2 : 0 2 0
ENTER THE ROW 3 : 0 0 3
ENTER THE ELEMENTS
ENTER THE ROW 1 : 0 0 1
ENTER THE ROW 2 : 0 2 0
ENTER THE ROW 3 : 3 0 0
THE SPARSE MATRIX IS :
3 3 5
0 0 1
0 2 1
1 1 4
2 0 3
2 2 3
```

```
FUNCTION main()
    DECLARE INT r , c
    PRINT "ENTER THE ROW AND COLUMN OF MATRICES : "
    READ r, c
    DECLARE INT A[r][c], B[r][c], C[r * c][3], D[r * c][3]
    DECLARE INT result[r * c][3]

    insertElements(r, c, A)
    insertElements(r, c, B)

    DECLARE INT k1 = convertToThreeTuple(r, c, A, C)
    DECLARE INT k2 = convertToThreeTuple(r, c, B, D)
    DECLARE INT k3 = addSparseMatrices(C, D, result)

    display(k3, 3, result)
END FUNCTION
```

RESULT

```
Program Executed Successfully
```

PROGRAM

```
#include <stdio.h>
void merge(int arr[], int l, int m, int r){
    int i, j, k;
    int n1 = m-l+1;
    int n2 = r-m;

    int L[n1], R[n2];

    for(i=0; i<n1; i++){
        L[i] = arr[l+i];
    }
    for(j=0; j<n2; j++){
        R[j] = arr[m+1+j];
    }

    i=0;
    j=0;
    k=l;

    while(i<n1 && j<n2){
        if(L[i]<=R[j]){
            arr[k] = L[i];
            i++;
        }
        else{
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while(i<n1){
        arr[k] = L[i];
        i++;
        k++;
    }

    while(j<n2){
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[] , int l , int r){
    if(l<r){
        int m = (l+r)/2 ;

        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);

        merge(arr, l, m ,r);
    }

}
```

DATE : 20-10-2023

MERGE SORT

AIM

To implement Merge Sort.

PSEUDOCODE

```
INPUT : Unsorted array of integers
OUTPUT : Sorted array of integers

PROCEDURE merge(INT arr[] , INT l , INT m, INT r)
    DECLARE INT i, j, k
    DECLARE INT n1, n2

    SET n1 = m - l + 1
    SET n2 = r - m

    DECLARE INT L[n1]
    DECLARE INT R[n2]

    FOR i = 0 TO n1 - 1
        SET L[i] = arr[l + i]
    END FOR

    FOR j = 0 TO n2 - 1
        SET R[j] = arr[m + 1 + j]
    END FOR

    SET i = 0
    SET j = 0
    SET k = l

    WHILE i < n1 AND j < n2
        IF L[i] <= R[j] THEN
            SET arr[k] = L[i]
            INCREMENT i BY 1
        ELSE
            SET arr[k] = R[j]
            INCREMENT j BY 1
        END IF
        INCREMENT k BY 1
    END WHILE

    WHILE i < n1
        SET arr[k] = L[i]
        INCREMENT i
        INCREMENT k
    END WHILE
```

```
int main(){
    int n ;
    printf("ENTER THE SIZE OF ARRAY : ") ;
    scanf("%d",&n) ;

    int arr[n] ;
    printf("ENTER THE ELEMENTS OF ARRAY : ") ;
    for(int i=0 ; i<n ; i++){
        scanf("%d",&arr[i]) ;
    }

    mergeSort(arr, 0, n-1);

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

OUTPUT

```
ENTER THE SIZE OF ARRAY : 5
ENTER THE ELEMENTS OF ARRAY : 6 3 8 2 0
SORTED ARRAY : 0 2 3 6 8
```

```
    WHILE j < n2
        SET arr[k] = R[j]
        INCREMENT j
        INCREMENT k
    END WHILE
END PROCEDURE

PROCEDURE mergeSort(INT arr[] , INT l , INT r)
    IF l < r THEN
        DECLARE INT m
        SET m = (l+r)/2

        CALL mergeSort(arr, l, m)
        CALL mergeSort(arr, m + 1, r)

        CALL merge(arr, l, m, r)
    END IF
END PROCEDURE

PROCEDURE main()
    DECLARE INT n
    PRINT "ENTER THE SIZE OF ARRAY: "
    READ n

    DECLARE INT arr[n]
    PRINT "ENTER THE ELEMENTS OF ARRAY: "
    FOR i = 0 TO n - 1
        READ arr[i]
    END FOR

    CALL mergeSort(arr, 0, n - 1)

    PRINT "SORTED ARRAY: "
    FOR i = 0 TO n - 1
        PRINT arr[i]
    END FOR
END PROCEDURE
```

RESULT

Program Executed Successfully

PROGRAM

```
#include <stdio.h>

void swap(int *a , int *b){
        int temp = *b ;
        *b = *a ;
        *a = temp ;
        }

int partition(int A[] , int low, int high){
        int pivot = A[low] , i = low , j = high ;
        while(i<j){
                while(A[i] <= pivot){
                        i++ ;
                        }
                while(A[j] > pivot){
                        j-- ;
                        }
                if(i<j){
                        swap(&A[i] ,&A[j]) ;
                        }
                }
        swap(&A[low] ,&A[j]) ;
        return j ;
        }

void quicksort(int A[] , int low , int high){
        if(low<high){
                int pivot = partition(A , low , high) ;
                quicksort(A , low , pivot-1) ;
                quicksort(A , pivot+1 , high) ;
                }
        }

int main(){
        int n ;
        printf("ENTER THE NUMBER OF ELEMENTS : ") ;
        scanf("%d" , &n) ;
        int A[n] ;
        printf("ENTER THE ELEMENTS : ") ;
        for(int i = 0 ; i<n ; i++){
                scanf("%d" , &A[i]) ;
                }
        quicksort(A , 0 , n-1) ;
        printf("THE SORTED ARRAY IS : ") ;
        for(int i = 0 ; i<n ; i++){
                printf("%d " , A[i]) ;
                }
        printf("\n") ;
        }
```

DATE : 20-10-2023

QUICK SORT

AIM

To implement Quick Sort.

PSEUDOCODE

```
INPUT : Unsorted array of integers
OUTPUT : Sorted array of integers

PROCEDURE swap(a, b)
    DECLARE temp
    SET temp = b
    SET b = a
    SET a = temp
END PROCEDURE

PROCEDURE partition(A, low, high)
    DECLARE pivot, i, j
    SET pivot = A[low]
    SET i = low
    SET j = high

    WHILE i < j
        WHILE A[i] <= pivot
            INCREMENT i
        END WHILE

        WHILE A[j] > pivot
            DECREMENT j
        END WHILE

        IF i < j THEN
            CALL swap(A[i], A[j])
        END IF
    END WHILE

    CALL swap(A[low], A[j])
    RETURN j
END PROCEDURE
PROCEDURE quicksort(A, low, high)
    DECLARE pivot
    IF low < high THEN
        SET pivot = CALL partition(A, low, high)
        CALL quicksort(A, low, pivot - 1)
        CALL quicksort(A, pivot + 1, high)
    END IF
END PROCEDURE
```

OUTPUT

```
ENTER THE NUMBER OF ELEMENTS : 5
ENTER THE ELEMENTS : 3 2 1 6 7
THE SORTED ARRAY IS : 1 2 3 6 7
```

```
PROCEDURE main()
    DECLARE n
    PRINT "ENTER THE NUMBER OF ELEMENTS : "
    READ n
    DECLARE A[n]
    PRINT "ENTER THE ELEMENTS : "
    FOR i = 0 TO n - 1
        READ A[i]
    END FOR
    CALL quicksort(A, 0, n - 1)
    PRINT "THE SORTED ARRAY IS : "
    FOR i = 0 TO n - 1
        PRINT A[i] + " "
    END FOR
    PRINT "\n"
END PROCEDURE
```

RESULT

Program Executed Successfully

PROGRAM

```
#include <stdio.h>

int items[100];
int front = -1;
int rear = -1 ;
int size ;

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

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

void enQueue(int element) {
  if (isFull()){
    printf("QUEUE IS FULL \n");
  }
  else {
    if (front == -1){
        front = 0;
    }
    rear = (rear + 1) % size;
    items[rear] = element;
  }
}
int deQueue() {
  int element;
  if (isEmpty()) {
    printf("QUEUE IS EMPTY\n");
    return (-1);
  }
  else{
    element = items[front];
    if (front == rear) {
      front = -1;
      rear = -1;
    }
    else {
      front = (front + 1) % size;
    }
    return (element);
  }
}
void display() {
  int i;
  if (isEmpty()){
    printf("QUEUE IS EMPTY\n");
  }
  else {
    for (i = front; i != rear; i = (i + 1) % size) {
      printf("%d ", items[i]);
    }
    printf("%d ", items[i]);
  }
```

DATE : 07-11-2023

CIRCULAR QUEUE

AIM

To implement the Data Structure Circular Queue.

PSEUDOCODE

```
INPUT : Input of selected operation from menu
OUTPUT :  Output of selected operation

DECLARE INT items[100]
DECLARE INT front = -1
DECLARE INT rear = -1
DECLARE INT size

PROCEDURE isFull()
    RETURN (front == rear + 1) OR (front == 0 AND rear == size - 1)
END PROCEDURE

PROCEDURE isEmpty()
    RETURN (front == -1)
END PROCEDURE

PROCEDURE enQueue(INT element)
    IF isFull() THEN
        PRINT "QUEUE IS FULL"
    ELSE
        IF front == -1 THEN
            SET front = 0
        END IF
        SET rear = (rear + 1) % size
        SET items[rear] = element
    END IF
END PROCEDURE

PROCEDURE deQueue()
    DECLARE INT element
    IF isEmpty() THEN
        PRINT "QUEUE IS EMPTY"
        RETURN (-1)
    ELSE
        SET element = items[front]
        IF front == rear THEN
            SET front = -1
            SET rear = -1
        ELSE
            SET front = (front + 1) % size
        END IF
        RETURN (element)
    END IF
END PROCEDURE
```

```
}
void menu() {
  printf("CHOICES...\n"); printf("1. INSERT \n");
  printf("2. DELETE \n"); printf("3. DISPLAY \n");
  printf("4. EXIT \n");
}
int main() {
  int choice , element ;
  scanf("%d", &size); menu() ;
  do {
    printf("ENTER YOUR CHOICE :");
    scanf("%d", &choice);
    switch (choice) {
      case 1:
        printf("ENTER ELEMENT TO BE INSERTED : ");
        scanf("%d", &element);enQueue(element);break;
      case 2:
        deQueue(); break;
      case 3:
        display(); break;
      case 4:
        printf("EXITING\n"); break;
      default:
        printf("INVALID CHOICE\n");
    }
  printf("\n") ;
  } while (choice != 4);
  return 0;
}
```

OUTPUT

```
ENTER THE SIZE OF QUEUE : 3
CHOICES...
1. INSERT
2. DELETE
3. DISPLAY
4. EXIT
ENTER YOUR CHOICE :1
ENTER ELEMENT TO BE INSERTED : 12
ENTER YOUR CHOICE :1
ENTER ELEMENT TO BE INSERTED : 13
ENTER YOUR CHOICE :1
ENTER ELEMENT TO BE INSERTED : 1
ENTER YOUR CHOICE :1
ENTER ELEMENT TO BE INSERTED : 15
QUEUE IS FULL
ENTER YOUR CHOICE :3
12 13 1
ENTER YOUR CHOICE :2
ENTER YOUR CHOICE :3
13 1
ENTER YOUR CHOICE :1
ENTER ELEMENT TO BE INSERTED : 100
ENTER YOUR CHOICE :3
13 1 100
ENTER YOUR CHOICE :4
EXITING
```

```
PROCEDURE display()
    DECLARE INT i
    IF isEmpty() THEN
        PRINT "QUEUE IS EMPTY"
    ELSE
        FOR i = front TO rear
            PRINT items[i]
        END FOR
    END IF
END PROCEDURE

PROCEDURE menu()
    PRINT "CHOICES..."
    PRINT "1. INSERT"
    PRINT "2. DELETE"
    PRINT "3. DISPLAY"
    PRINT "4. EXIT"
END PROCEDURE

PROCEDURE main()
    DECLARE INT choice, element
    PRINT "ENTER THE SIZE OF QUEUE : "
    READ size
    menu()
    DO
        PRINT "ENTER YOUR CHOICE :"
        READ choice
        SWITCH choice
            CASE 1:
                PRINT "ENTER ELEMENT TO BE INSERTED : "
                READ element
                enQueue(element)
                BREAK
            CASE 2:
                deQueue()
                BREAK
            CASE 3:
                display()
                BREAK
            CASE 4:
                PRINT "EXITING"
                BREAK
            DEFAULT:
                PRINT "INVALID CHOICE"
        END SWITCH
        PRINT ""
    WHILE choice != 4
    RETURN 0
END PROCEDURE
```

RESULT

```
Program Executed Successfully
```

PROGRAM

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

#define SIZE 5

int Q[SIZE];
int F = -1 ;
int R = -1 ;


void addRear();
void addFront();
void rmRear();
void rmFront();

void display();
void size();

bool isEmpty(){
            return F == -1;
}

bool isFull(){
            return (R+1)%SIZE == F;
}

void main(){
            int option;
            printf("\n\nQueue operations\n");
            printf("1.Display\n");
            printf("2.Add to rear\n");
            printf("3.Add to front\n");
            printf("4.Remove from rear\n");
            printf("5.Remove from front\n");
            printf("6.Size\n");
            printf("7.Exit\n");
            while(true){
                        printf("Select an option : ");
                        scanf("%d",&option);
                        printf("\n");
                        switch(option){
                                    case 1      :display();break;
                                    case 2      :addRear();break;
                                    case 3      :addFront();break;
                                    case 4      :rmRear();break;
                                    case 5      :rmFront();break;
                                    case 6      :size();break;
                                    case 7      :exit(0);break;
                                    default     :printf("Invalid option
entered\n");
                        }
            }
}

void addRear(){
            if(isFull()){
```

DATE : 07-11-2023

DOUBLE ENDED QUEUE

AIM

To implement the Data Structure Double Ended Queue.

PSEUDOCODE

```
INPUT : Input of selected operation from menu
OUTPUT :  Output of selected operation


PROGRAM DEQUE
DEFINE SIZE 5
DECLARE ARRAY Q[SIZE]
SET front = -1
SET rear = -1

PROCEDURE ISEMPTY
    RETURN front == -1
END PROCEDURE

PROCEDURE ISFULL
    RETURN (rear + 1) % SIZE == front
END PROCEDURE

PROCEDURE ADDREAR
    IF ISFULL
        PRINT "Queue overflow"
    ELSE
        IF ISEMPTY
            SET front = 0
            SET rear = 0
        ELSE
            SET rear = (rear + 1) % SIZE
        END IF
        INPUT Q[rear]
    END IF
END PROCEDURE

PROCEDURE ADDFRONT
    IF ISFULL
        PRINT "Queue overflow"
    ELSE
        IF ISEMPTY
            SET front = 0
            SET rear = 0
        ELSE IF front == 0
            SET front = SIZE - 1
        ELSE
```

```
                        printf("Queue overflow\n");
            }
            else{
                        if(isEmpty()){
                                    F = 0;
                                    R = 0;
                        }
                        else{
                                    R = (R+1)%SIZE;
                        }
                        printf("Enter element : ");
                        scanf("%d" , &Q[R]);
            }
}

void addFront(){
            if(isFull()){
                        printf("Queue overflow\n");
            }
            else{
                        if(isEmpty()){
                                    F = 0;
                                    R = 0;
                        }
                        else if(F == 0){
                                    F = SIZE -1;
                        }
                        else{
                                    F--;
                        }
                        printf("Enter element : ");
                        scanf("%d" , &Q[F]);
            }
}

void rmRear(){
            if(isEmpty()){
                        printf("Queue underflow\n");
            }
            else{
                        printf("Removing %d\n" , Q[R]);
                        if(F == R){
                                    F = -1;
                                    R = -1;
                        }
                        else if(R == 0){
                                    R = SIZE - 1;
                        }
                        else{
                                    R--;
                        }
            }
}

void rmFront(){
            if(isEmpty()){
                        printf("Queue underflow\n");
            }
            else{
```

```
            SET front = front - 1
        END IF
        INPUT Q[front]
    END IF
END PROCEDURE

PRCEDURE REMOVEREAR
    IF ISEMPTY
        PRINT "Queue underflow"
    ELSE
        PRINT Q[rear] " is removed"
        IF front == rear
            SET front = -1
            SET rear = -1
        ELSE IF rear == 0
            SET rear = SIZE - 1
        ELSE
            SET rear = rear - 1
        END IF
    END IF
END PROCEDURE

PROCEDURE REMOVEFRONT
    IF ISEMPTY
        PRINT "Queue underflow"
    ELSE
        PRINT Q[front] " is removed"
        IF front == rear
            SET front = -1
            SET rear = -1
        ELSE
            SET front = (front + 1) % SIZE
        END IF
    END IF
END PROCEDURE

PROCEDURE DISPLAY
    IF ISEMPTY
        PRINT "Queue is empty"
        RETURN
    END IF
    SET i = front
    WHILE i != rear
        PRINT Q[i]
        SET i = (i + 1) % SIZE
    END WHILE
    PRINT Q[rear]
END PROCEDURE

PROCEDURE SIZE
    DECLARE INTEGER size
    IF ISEMPTY
```

```
                        printf("Removing %d\n" , Q[F]);
                        if (F == R){
                                    F = -1;
                                    R = -1;
                        }
                        else{
                                    F = (F+1)%SIZE;
                        }
            }
}
void display(){
            if(isEmpty()){
                        printf("Queue is Empty ! \n");
                        return;
            }
            printf("Queue : ");
            int i = F;
            while(i != R){
                        printf("%d\t",Q[i]);
                        i = (i+1)%SIZE;
            }
            printf("%d\n",Q[R]);
}
void size(){
            int size_v;
            if(isEmpty())
                        size_v = 0;
            else if(isFull())
                        size_v = SIZE;
            else if(F > R)
                        size_v = SIZE - (F - R) + 1;
            else
                        size_v = F - R + 1;
            printf("Size is %d\n" , size_v);
}
```

OUTPUT

```
Queue operations
1.Display
2.Add to rear
3.Add to front
4.Remove from rear
5.Remove from front
6.Size
7.Exit
Select an option : 2
Enter element : 1
Select an option : 3
Enter element : 2
Select an option : 1
Queue : 2   1
Select an option : 4
Removing 1
Select an option : 5
Removing 2
Select an option : 6
Size is 0
Select an option : 7
```

```
        SET size = 0
    ELSE IF ISFULL
        SET size = SIZE
    ELSE IF front > rear
        SET size = SIZE - (front - rear) + 1
    ELSE
        SET size = rear - front + 1
    END IF
    PRINT size
END PROCEDURE

PRCEDURE MAIN
    PRINT "1. Display"
    PRINT "2. Add rear"
    PRINT "3. Add front"
    PRINT "4. Remove rear"
    PRINT "5. Remove front"
    PRINT "6. Size"
    PRINT "7. Exit"
    WHILE TRUE
        INPUT choice
        CASE choice OF
            1: CALL DISPLAY
            2: CALL ADDREAR
            3: CALL ADDFRONT
            4: CALL REMOVEREAR
            5: CALL REMOVEFRONT
            6: CALL SIZE
            7: EXIT
        END CASE
    END WHILE
END PROCEDURE

END PROGRAM
```

RESULT

Program Executed Successfully

PROGRAM

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

struct node {
        int data ;
        struct node *next ;
        }*head = NULL ;

void create(){
        struct node *temp ;

        int n ;
        printf("ENTER THE NUMBER OF ELEMENTS  : ") ;
        scanf("%d" , &n) ;

        printf("ENTER THE ELEMENTS : ") ;
        for ( int i = 0 ; i< n ; i++){
                struct node *ptr ;
                ptr = malloc(sizeof(struct node)) ;

                scanf("%d" ,&ptr->data) ;
                if(i==0){
                        head = ptr ;
                        ptr->next =NULL ;
                        temp = ptr ;
                        }
                else{
                        temp->next = ptr ;
                        ptr->next =NULL ;
                        temp = ptr ;
                        }
                }
        }

void display(){
        struct node *ptr ;
        ptr = head ;
        while(ptr!= NULL){
                printf("%d " , ptr->data) ;
                ptr = ptr->next ;
                }
        printf("\n") ;
        }

void insertFront(){
        struct node *ptr ;
        ptr = malloc(sizeof(struct node)) ;
        printf("ENTER THE ELEMENT TO BE INSERTED : ") ;
        scanf("%d" , &ptr->data) ;

        ptr->next = head ;
        head = ptr ;
        }

void insertBack(){
        struct node *ptr ;
        ptr = malloc(sizeof(struct node)) ;
```

DATE : 14-11-2023

LINKED LIST OPERATIONS

AIM

To implement Linked List Data Structure Operations.

PSEUDOCODE

```
INPUT : Input of selected operation from menu
OUTPUT :  Output of selected operation

STRUCT node
    DECLARE data
    DECLARE next
END STRUCT

DECLARE head = NULL

PROCEDURE create()
    DECLARE temp
    DECLARE n

    PRINT "ENTER THE NUMBER OF ELEMENTS : "
    READ n

    FOR i = 0 TO n - 1
        DECLARE ptr
        ALLOCATE ptr

        PRINT "ENTER THE ELEMENTS : "
        READ ptr.data

        IF i == 0 THEN
            SET head = ptr
            SET ptr.next = NULL
            SET temp = ptr
        ELSE
            SET temp.next = ptr
            SET ptr.next = NULL
            SET temp = ptr
        END IF
    END FOR
END PROCEDURE

PROCEDURE display()
    DECLARE ptr
    SET ptr = head

    WHILE ptr != NULL
        PRINT ptr.data + " "
        SET ptr = ptr.next
    END WHILE
```

```
        printf("ENTER THE ELEMENT TO BE INSERTED  : ") ;
        scanf("%d" , &ptr->data) ;

        if(head == NULL){
                head = ptr ;
                ptr->next =NULL ;
                }
        else{
                struct node *temp = head ;
                while(temp->next != NULL){
                        temp = temp->next ;
                        }

                temp->next = ptr ;
                ptr->next = NULL ;
        }
}

void insertBetween(){
        struct node *ptr, *temp= head ;
        int i = 0 , pos ;
        ptr = malloc(sizeof(struct node)) ;

        printf("ENTER THE NUMBER TO BE INSERTED : ") ;
        scanf("%d", &ptr->data) ;

        printf("ENTER THE POSITION ON WHICH TO BE INSERTED  :") ;
        scanf("%d" , &pos) ;

        while(temp != NULL && i < pos-1){
                temp = temp->next ;
                i++ ;
                }
        if(temp == NULL){
                printf("INVALID POSITION\n") ;

        }
        else{
                ptr->next = temp->next ;
                temp->next = ptr ;
        }
        }


void deleteFront(){
        if(head == NULL){
                printf("LIST IS EMPTY\n") ;
                }
        else{
                struct node *temp= head ;
                head = head->next ;
                free(temp);
        }
}

void deleteBack(){
        if(head == NULL){
                printf("LIST IS EMPTY\n") ;
                }
```

```
    PRINT "\n"
END PROCEDURE
PROCEDURE insertFront()
    DECLARE ptr
    ALLOCATE ptr

    PRINT "ENTER THE ELEMENT TO BE INSERTED : "
    READ ptr.data

    SET ptr.next = head
    SET head = ptr
END PROCEDURE

PROCEDURE insertBack()
    DECLARE ptr
    ALLOCATE ptr

    PRINT "ENTER THE ELEMENT TO BE INSERTED : "
    READ ptr.data

    IF head == NULL THEN
        SET head = ptr
        SET ptr.next = NULL
    ELSE
        DECLARE temp
        SET temp = head

        WHILE temp.next != NULL
            SET temp = temp.next
        END WHILE

        SET temp.next = ptr
        SET ptr.next = NULL
    END IF
END PROCEDURE

PROCEDURE insertBetween()
    DECLARE ptr, temp
    DECLARE i, pos
    ALLOCATE ptr

    PRINT "ENTER THE NUMBER TO BE INSERTED : "
    READ ptr.data

    PRINT "ENTER THE POSITION ON WHICH TO BE INSERTED :"
    READ pos

    SET temp = head
    SET i = 0

    WHILE temp != NULL AND i < pos - 1
        SET temp = temp.next
        INCREMENT i
    END WHILE

    IF temp == NULL THEN
        PRINT "INVALID POSITION"
    ELSE
        SET ptr.next = temp.next
```

```
        else if(head->next == NULL){
                free(head) ;
                head = NULL ;
                }
        else{
                struct node *temp = head ;
                while(temp->next->next != NULL){
                        temp = temp->next ;
                        }
                free(temp->next) ;
                temp->next = NULL ;
        }
}

void search(){
        int x, i = 0 ;
        struct node *temp = head ;
        printf("ENTER THE ELEMENT TO BE SEARCHED : ") ;
        scanf("%d", &x) ;

        while(temp != NULL){
                if( temp->data == x){
                        printf("POSITION IS : %d\n" , i+1) ;
                        break ;
                        }
                i++ ;
                temp = temp->next ;
                }
        if(temp == NULL){
                printf("ELEMENT NOT FOUND.\n") ;
                }
        }

void menu(){
        printf("1. CREATE\n") ;
        printf("2. DISPLAY\n") ;
        printf("3. INSERT AT FRONT\n") ;
        printf("4. INSERT AT BACK\n") ;
        printf("5. INSERT BETWEEN\n") ;
        printf("6. DELETE FRONT\n") ;
        printf("7. DELETE BACK\n") ;
        printf("8. SEARCH\n") ;
        printf("9. EXIT\n") ;
        }


int main(){
        int choice ;
        menu() ;
        while(1){
                printf("ENTER YOUR CHOICE : ") ;
                scanf("%d" , &choice) ;
                switch(choice){
                        case 1 : create() ;
                                 break ;
                        case 2 : display() ;
                                 break ;
                        case 3 : insertFront() ;
                                 break ;
```

```
        SET temp.next = ptr
    END IF
END PROCEDURE

PROCEDURE deleteFront()
    IF head == NULL THEN
        PRINT "LIST IS EMPTY"
    ELSE
        DECLARE temp
        SET temp = head
        SET head = head.next
        DEALLOCATE temp
    END IF
END PROCEDURE

PROCEDURE deleteBack()
    IF head == NULL THEN
        PRINT "LIST IS EMPTY"
    ELSE IF head.next == NULL THEN
        DEALLOCATE head
        SET head = NULL
    ELSE
        DECLARE temp
        SET temp = head

        WHILE temp.next.next != NULL
            SET temp = temp.next
        END WHILE

        DEALLOCATE temp.next
        SET temp.next = NULL
    END IF
END PROCEDURE

PROCEDURE search()
    DECLARE x, i
    DECLARE temp
    SET temp = head
    PRINT "ENTER THE ELEMENT TO BE SEARCHED : "
    READ x

    SET i = 0

    WHILE temp != NULL
        IF temp.data == x THEN
            PRINT "POSITION IS : " + (i + 1)
            BREAK
        END IF

        INCREMENT i
        SET temp = temp.next
    END WHILE

    IF temp == NULL THEN
        PRINT "ELEMENT NOT FOUND."
    END IF
END PROCEDURE
```

```
                        case 4 : insertBack() ;
                                 break ;
                        case 5 : insertBetween() ;
                                 break ;
                        case 6 : deleteFront() ;
                                 break ;
                        case 7 : deleteBack() ;
                                 break ;
                        case 8 : search() ;
                                 break ;
                        case 9 : exit(0) ;
                                 break ;
                        default : printf("INVALID CHOICE\n") ;
                                  break ;
                        }
                printf("\n") ;
                }
        return 0 ;
        }
```

OUTPUT

```
1. CREATE
2. DISPLAY
3. INSERT AT FRONT
4. INSERT AT BACK
5. INSERT BETWEEN
6. DELETE FRONT
7. DELETE BACK
8. SEARCH
9. EXIT
ENTER YOUR CHOICE : 1
ENTER THE NUMBER OF ELEMENTS  : 3
ENTER THE ELEMENTS : 1 2 3

ENTER YOUR CHOICE : 6

ENTER YOUR CHOICE : 4
ENTER THE ELEMENT TO BE INSERTED  : 12

ENTER YOUR CHOICE : 2
2 3 12

ENTER YOUR CHOICE : 8
ENTER THE ELEMENT TO BE SEARCHED : 11
ELEMENT NOT FOUND.

ENTER YOUR CHOICE : 9
```

```
PROCEDURE menu()
    PRINT "1. CREATE"
    PRINT "2. DISPLAY"
    PRINT "3. INSERT AT FRONT"
    PRINT "4. INSERT AT BACK"
    PRINT "5. INSERT BETWEEN"
    PRINT "6. DELETE FRONT"
    PRINT "7. DELETE BACK"
    PRINT "8. SEARCH"
    PRINT "9. EXIT"
END PROCEDURE

PROCEDURE main()
    DECLARE choice
    CALL menu()
    WHILE TRUE
        PRINT "ENTER YOUR CHOICE : "
        READ choice

        SWITCH choice
            CASE 1
                CALL create()
                BREAK
            CASE 2
                CALL display()
                BREAK
            CASE 3
                CALL insertFront()
                BREAK
            CASE 4
                CALL insertBack()
                BREAK
            CASE 5
                CALL insertBetween()
                BREAK
            CASE 6
                CALL deleteFront()
                BREAK
            CASE 7
                CALL delesteBack()
                BREAK
            CASE 8
                CALL search()
                BREAK
            CASE 9
                EXIT
                BREAK
            DEFAULT
                PRINT "INVALID CHOICE"
                BREAK
        END SWITCH
    END WHILE

    RETURN 0
END PROCEDURE
```

RESULT

```
Program Executed Successfully
```

PROGRAM

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

struct node{
    int coeff ;
    int exp ;
    struct node *next ;
};

struct node *createPoly(struct node *head){
    struct node *temp , *current ;
    current = head ;
    int n ;
    printf("ENTER NUMBER OF TERMS IN POLYNOMIAL : ") ;
    scanf("%d",&n) ;

    for(int i = 0 ; i< n ; i++){
        temp = malloc(sizeof(struct node)) ;
        printf("ENTER COEFFICIENT AND EXPONENT OF TERM %d : ",i+1) ;
        scanf("%d%d",&temp->coeff,&temp->exp) ;
        temp->next = NULL ;

        if(current == NULL){
            current = temp ;
            head = current ;
        }
        else{
            current->next = temp ;
            current = temp ;
        }
    }
    return head ;
}

void displayPoly(struct node *head){
    struct node *temp = head ;
    while(temp != NULL){
        if(temp->next != NULL){
            printf("%dx^%d + ",temp->coeff,temp->exp) ;
        }
        else{
            printf("%dx^%d",temp->coeff,temp->exp);
        }
        temp = temp->next ;
    }
    printf("\n") ;
}

int main(){
    struct node *head = NULL ;
    head = createPoly(head) ;
    displayPoly(head) ;
}
```

DATE : 14-11-2023

POLYNOMIAL REPRESENTATION

AIM

To implement a program to Represent Polynomial Using Linked List.

PSEUDOCODE

```
INPUT : Coefficient and exponent of each term
OUTPUT :  Polynomial with the inputted coefficients and exponents

STRUCT node
    DECLARE data
    DECLARE next
END STRUCT

PROCEDURE createPoly(head)
    DECLARE temp, current, n
    SET current = head

    PRINT "ENTER NUMBER OF TERMS IN POLYNOMIAL : "
    READ n

    FOR i = 0 TO n-1
        ALLOCATE temp
        PRINT "ENTER COEFFICIENT AND EXPONENT OF TERM " + (i+1) + " :
"
        READ temp.coeff, temp.exp
        SET temp.next = NULL

        IF current == NULL THEN
            SET current = temp
            SET head = current
        ELSE
            SET current.next = temp
            SET current = temp
        END IF
    END FOR

    RETURN head
END PROCEDURE

PROCEDURE displayPoly(head)
    DECLARE temp
    SET temp = head

    WHILE temp != NULL
        IF temp.next != NULL THEN
            PRINT temp.coeff + "x^" + temp.exp + " + "
```

OUTPUT

```
ENTER THE SIZE OF ARRAY  : 6
ENTER THE ARRAY ELEMENTS : 1 2 3 4 5 6
ENTER THE ELEMENT TO BE SEARCHED : 3
THE POSITION IS : 2
```

```
        ELSE
            PRINT temp.coeff + "x^" + temp.exp
        END IF
        SET temp = temp.next
    END WHILE

    PRINT "\n"
END PROCEDURE

PROCEDURE main()
    DECLARE head
    SET head = NULL

    SET head = createPoly(head)
    CALL displayPoly(head)
END PROCEDURE
```

RESULT

```
Program Executed Successfully
```

PROGRAM

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

struct Node {
    int key;
    struct Node* left;
    struct Node* right;
};

struct Node *root=NULL,*temp=NULL;

struct Node* newNode(int key)
{
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->key = key;
    node->left = node->right = NULL;
    return (node);
}

void inorder(struct Node* root)
{
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->key);
        inorder(root->right);
    }
}

void preorder(struct Node* root)
{
    if (root != NULL) {
        printf("%d ", root->key);
        preorder(root->left);
        preorder(root->right);
    }
}

void postorder(struct Node* root)
{
    if (root != NULL) {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->key);
    }
}

struct Node* insert(struct Node* node, int key)
{
    if (node == NULL)
        return newNode(key);
    if (key < node->key)
        node->left = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);
    return node;
}
```

DATE : 14-11-2023

TREE TRAVERSALS

AIM

To implement Data Structure Tree using Linked List.

PSEUDOCODE

```
INPUT : Input of selected operation from menu
OUTPUT :  Output of selected operation

STRUCTURE Node
    INTEGER key
    Node POINTER left
    Node POINTER right
END STRUCTURE
SET Node POINTER root = NULL
SET Node POINTER temp = NULL

PROCEDURE newNode ACCEPTS key
    SET Node POINTER node = Allocate memory for Node
    SET node.key = key
    SET node.left = NULL
    SET node.right = NULL
    RETURN node
END PROCEDURE

PROCEDURE inorder ACCEPTS root
    IF root != NULL
        CALL inorder(root.left)
        PRINT root.key
        CALL inorder(root.right)
    END IF
END PROCEDURE

PROCEDURE preorder ACCEPTS root
    IF root != NULL
        PRINT root.key
        CALL preorder(root.left)
        CALL preorder(root.right)
    END IF
END PROCEDURE

PROCEDURE postorder ACCEPTS root
    IF root != NULL
        CALL postorder(root.left)
        CALL postorder(root.right)
        PRINT root.key
    END IF
```

```
struct Node* search(struct Node* root, int key)
{
    if (root == NULL || root->key == key)
        return root;
    if (root->key < key)
        return search(root->right, key);
    return search(root->left, key);
}

int minimum(struct Node* root){
    if(root==NULL)
        return 0;
    else{
        while(root->left!=NULL){
            root=root->left;
        }
        return root->key;
    }
}

int maximum(struct Node* root){
    if(root==NULL)
        return 0;
    else{
        while(root->right!=NULL){
            root=root->right;
        }
        return root->key;
    }
}

void deleteNode(struct Node** root, int key) {
    if (*root == NULL) return;
    if (key < (*root)->key)
        deleteNode(&((*root)->left), key);
    else if (key > (*root)->key)
        deleteNode(&((*root)->right), key);
    else {
        if ((*root)->left == NULL) {
            struct Node *temp = *root;
            *root = (*root)->right;
            free(temp);
        }
        else if ((*root)->right == NULL) {
            struct Node *temp = *root;
            *root = (*root)->left;
            free(temp);
        }
        else {
            int minKey = minimum((*root)->right);
            (*root)->key = minKey;
            deleteNode(&((*root)->right), minKey);
        }
    }
}

int main()
{
    int ch,key;
```

```
END PROCEDURE

PROCEDURE delete ACCEPTS root, key
    IF root == NULL
        RETURN
    END IF
    IF key < root.key
        SET root.left = CALL delete(root.left, key)
    ELSE IF key > root.key
        SET root.right = CALL delete(root.right, key)
    ELSE
        IF root.left == NULL AND root.right == NULL
            FREE root
            SET root = NULL
        ELSE IF root.left == NULL
            SET Node POINTER temp = root
            SET root = root.right
            FREE temp
        ELSE IF root.right == NULL
            SET Node POINTER temp = root
            SET root = root.left
            FREE temp
        ELSE
            SET INTEGER temp = CALL minimum(root.right)
            SET root.key = temp
            CALL delete(root.right, temp)
        END IF
    END IF
END PROCEDURE

PROCEDURE insert ACCEPTS node, key
    IF node == NULL
        RETURN CALL newNode(key)
    END IF
    IF key < node.key
        SET node.left = CALL insert(node.left, key)
    ELSE IF key > node.key
        SET node.right = CALL insert(node.right, key)
    END IF
    RETURN node
END PROCEDURE

PROCEDURE search ACCEPTS root, key
    IF root == NULL OR root.key == key
        RETURN root
    END IF
    IF root.key < key
        RETURN CALL search(root.right, key)
    END IF
    RETURN CALL search(root.left, key)
END PROCEDURE

PROCEDURE minimum ACCEPTS root
    IF root == NULL
```

```
    printf("\n1.Insert\n2.Search\n3.Minimum\n4.Maximum\n5.Inorder\
n6.Preorder\n7.Postorder\n8.Delete\n9.Exit\n");
    while(1){
        printf("Enter your choice:");
        scanf("%d",&ch);
        switch(ch){
            case 1:
                printf("Enter the key:");
                scanf("%d",&key);
                root=insert(root,key);
                break;
            case 2:
                printf("Enter the key:");
                scanf("%d",&key);
                temp=search(root,key);
                if(temp==NULL)
                    printf("Key not found\n");
                else
                    printf("Key found\n");
                break;
            case 3:
                printf("Minimum key is %d\n",minimum(root));
                break;
            case 4:
                printf("Maximum key is %d\n",maximum(root));
                break;
            case 5:
                inorder(root);
                printf("\n");
                break;
            case 6:
                preorder(root);
                printf("\n");
                break;
            case 7:
                postorder(root);
                printf("\n");
                break;
            case 8:
                printf("Enter the key to delete:");
                scanf("%d", &key);
                deleteNode(&root, key);
                break;
            case 9:
                exit(0);
        }
    }
    return 0;
}
```

```
        RETURN 0
    END IF
    WHILE root.left != NULL
        SET root = root.left
    END WHILE
    RETURN root.key
END PROCEDURE
PROCEDURE maximum ACCEPTS root
    IF root == NULL
        RETURN 0
    END IF
    WHILE root.right != NULL
        SET root = root.right
    END WHILE
    RETURN root.key
END PROCEDURE
PROCEDURE main
    SET INTEGER ch, key
    PRINT "1.Insert 2.Search 3.Minimum 4.Maximum 5.Inorder 6.Preorder
7.Postorder 8.Delete 9.Exit"
    WHILE TRUE
        INPUT ch
        CASE ch OF
            1:
                INPUT key
                SET root = CALL insert(root, key)
            2:
                INPUT key
                SET temp = CALL search(root, key)
                IF temp == NULL
                    PRINT "Key not found"
                ELSE
                    PRINT "Key found"
                END IF
            3:
                PRINT "Minimum key is ", CALL minimum(root)
            4:
                PRINT "Maximum key is ", CALL maximum(root)
            5:
                CALL inorder(root)
            6:
                CALL preorder(root)
            7:
                CALL postorder(root)
            8:
                INPUT key
                CALL delete(root, key)
            9:
                EXIT program
        END CASE
    END WHILE
END PROCEDURE
```

OUTPUT

```
1.Insert
2.Search
3.Minimum
4.Maximum
5.Inorder
6.Preorder
7.Postorder
8.Delete
9.Exit
Enter your choice:1
Enter the key:1
Enter your choice:1
Enter the key:2
Enter your choice:1
Enter the key:3
Enter your choice:3
Minimum key is 1
Enter your choice:4
Maximum key is 3
Enter your choice:5
1 2 3
Enter your choice:6
1 2 3
Enter your choice:7
3 2 1
Enter your choice:2
Enter the key:3
Key found
Enter your choice:8
Enter the key to delete:2
Enter your choice:5
1 3
Enter your choice:9
```

RESULT

```
Program Executed Successfully
```

PROGRAM

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

int degree = 0;

struct node{
    int coeff;
    int pow;
    struct node *next;
}*poly1 = NULL , *poly2 = NULL , *poly3 = NULL;

struct node *createPoly(struct node *head){
    int n , i , m = 0;
    struct node *ptr, *temp = head;
    printf("ENTER NUMBER OF TERMS IN POLYNOMIAL : ");
    scanf("%d", &n);
    for(i = 0 ; i < n ; i++){
        ptr = malloc(sizeof(struct node));
        printf("ENTER COEFFICIENT AND POWER : ");
        scanf("%d %d", &ptr->coeff, &ptr->pow);
        if(i == 0){
            head = ptr;
            ptr->next = NULL;
            temp = ptr;
        }
        else{
            temp->next = ptr;
            ptr->next = NULL;
            temp = ptr;
        }
        if(ptr->pow > m){
            m = ptr->pow;
        }
    }
    degree += m;
    return head;
}
void createPolyFromHash(int hash[] , int m){
    struct node *ptr , *temp = poly3 ;

    for(int i = 0 ; i<m+1 ; i++){
        if(hash[i] != 0){
            ptr = malloc(sizeof(struct node));
            ptr->coeff = hash[i];
            ptr->pow = i;
            ptr->next = NULL;

            if(temp == NULL){
                poly3 = ptr;
                temp = ptr;
            }
            else{
                temp->next = ptr;
                temp = ptr;
            }
        }
    }}
```

DATE : 21-11-2023

POLYNOMIAL MULTIPLICATION

AIM

To implement a program to Multiply Two Polynomials.

PSEUDOCODE

```
INPUT : Coefficient and exponent of each term of both polynomials
OUTPUT :  Product of both polynomials

DECLARE degree
SET degree = 0

PROCEDURE createPoly(head)
    DECLARE n, i, m
    DECLARE ptr, temp

    PRINT "ENTER NUMBER OF TERMS IN POLYNOMIAL : "
    READ n

    FOR i = 0 TO n - 1
        ALLOCATE ptr
        PRINT "ENTER COEFFICIENT AND POWER : "
        READ ptr.coeff, ptr.pow

        IF i = 0 THEN
            SET head = ptr
            SET ptr.next = NULL
            SET temp = ptr
        ELSE
            SET temp.next = ptr
            SET ptr.next = NULL
            SET temp = ptr
        END IF

        IF ptr.pow > m THEN
            SET m = ptr.pow
        END IF
    END FOR

    INCREMENT degree BY m

    RETURN head
END PROCEDURE

PROCEDURE createPolyFromHash(hash[], m)
    DECLARE ptr, temp
```

```
void displayPoly(struct node *head){
    struct node *ptr = head;
    while(ptr != NULL){
        if(ptr->next == NULL){
            printf("%dx^%d", ptr->coeff, ptr->pow);
            break;
        }
        printf("%dx^%d + ", ptr->coeff, ptr->pow);
        ptr = ptr->next;
    }
    printf("\n");
}

void multiplyPoly(struct node *head1 , struct node *head2, int m){
    struct node *ptr1 , *ptr2 ;

    int hash[m+1] ;
    for(int i = 0 ; i<m+1 ; i++){
        hash[i] = 0;
    }

    for (ptr1 = head1; ptr1 != NULL; ptr1 = ptr1->next) {
        for (ptr2 = head2; ptr2 != NULL; ptr2 = ptr2->next) {
            int c = ptr1->coeff * ptr2->coeff;
            int p = ptr1->pow + ptr2->pow;
            hash[p] += c;
        }
    }
    createPolyFromHash(hash , m);
}

int main(){
    poly1 = createPoly(poly1);
    poly2 = createPoly(poly2);

    printf("POLYNOMIAL 1 : ");
    displayPoly(poly1);
    printf("POLYNOMIAL 2 : ");
    displayPoly(poly2);

    multiplyPoly(poly1 , poly2, degree);

    printf("MULTIPLIED POLYNOMIAL : ");
    displayPoly(poly3);
}
```

```
    FOR i = 0 TO m
        IF hash[i] != 0 THEN
            ALLOCATE ptr
            SET ptr.coeff = hash[i]
            SET ptr.pow = i
            SET ptr.next = NULL

            IF temp = NULL THEN
                SET poly3 = ptr
                SET temp = ptr
            ELSE
                SET temp.next = ptr
                SET temp = ptr
            END IF
        END IF
    END FOR
END PROCEDURE

PROCEDURE displayPoly(head)
    DECLARE ptr

    SET ptr = head

    WHILE ptr != NULL
        IF ptr.next == NULL THEN
            PRINT ptr.coeff + "x^" + ptr.pow
            BREAK
        END IF

        PRINT ptr.coeff + "x^" + ptr.pow + " + "
        SET ptr = ptr.next
    END WHILE

    PRINT "\n"
END PROCEDURE

PROCEDURE multiplyPoly(head1, head2, m)
    DECLARE ptr1, ptr2
    DECLARE hash[m+1]

    FOR i = 0 TO m
        SET hash[i] = 0
    END FOR

    FOR ptr1 = head1 TO ptr1 != NULL STEP ptr1 = ptr1.next
        FOR ptr2 = head2 TO ptr2 != NULL STEP ptr2 = ptr2.next
            SET c = ptr1.coeff * ptr2.coeff
            SET p = ptr1.pow + ptr2.pow
            INCREMENT hash[p] BY c
        END FOR
    END FOR

    CALL createPolyFromHash(hash, m)
END PROCEDURE
```

OUTPUT

```
ENTER NUMBER OF TERMS IN POLYNOMIAL : 3
ENTER COEFFICIENT AND POWER : 1 0
ENTER COEFFICIENT AND POWER : 2 1
ENTER COEFFICIENT AND POWER : 3 2
ENTER NUMBER OF TERMS IN POLYNOMIAL : 3
ENTER COEFFICIENT AND POWER : 1 0
ENTER COEFFICIENT AND POWER : 2 1
ENTER COEFFICIENT AND POWER : 3 2
POLYNOMIAL 1 : 1x^0 + 2x^1 + 3x^2
POLYNOMIAL 2 : 1x^0 + 2x^1 + 3x^2
MULTIPLIED POLYNOMIAL : 1x^0 + 4x^1 + 10x^2 + 12x^3 + 9x^4
```

```
PROCEDURE main()
    SET poly1 = CALL createPoly(poly1)
    SET poly2 = CALL createPoly(poly2)

    PRINT "POLYNOMIAL 1 : "
    CALL displayPoly(poly1)
    PRINT "POLYNOMIAL 2 : "
    CALL displayPoly(poly2)

    CALL multiplyPoly(poly1, poly2, degree)

    PRINT "MULTIPLIED POLYNOMIAL : "
    CALL displayPoly(poly3)
END PROCEDURE
```

RESULT

```
Program Executed Successfully
```

PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
struct node
{
    int coefficent;
    int degree;
    struct node *next;
};

struct node* poly1 = NULL;
struct node* poly2 = NULL;
struct node* poly3 = NULL;

struct node* read_Poly(struct node *poly, int coeff, int deg)
{
    struct node *temp = poly;
    struct node* newnode = malloc(sizeof(struct node));
    newnode->next = NULL;
    newnode->coefficent = coeff;
    newnode->degree = deg;
    if (poly == NULL)
    {
        poly  = newnode;
    }
    else
    {
        while (temp->next!= NULL)
        {
           temp = temp->next;
        }
        temp->next = newnode;
    }
    return poly;
}
void display(struct node *poly)
{
    while(poly->next!= NULL)
    {
        printf("%dx^%d+ ",poly->coefficent, poly->degree);
        poly = poly->next;
    }
    printf("%dx^%d\n",poly->coefficent, poly->degree);
}

void addPoly()
{
    struct node *newnode = malloc(sizeof(struct node));
    newnode->next = NULL;
    struct node *temp1 = poly1;
    struct node *temp2 = poly2;
    struct node *temp3 = poly3;

    while (temp1!= NULL && temp2 != NULL)
    {
        if (temp1->degree == temp2->degree)
        {
```

DATE : 21-11-2023

POLYNOMIAL ADDITION-II

AIM

To implement Polynomial Addition Using Linked List.

PSEUDOCODE

```
INPUT : Coefficient and exponent of each term of both polynomials
OUTPUT :  Sum of both polynomials

DECLARE STRUCTURE node
           DECLARE INTEGER coefficent
    DECLARE INTEGER degree
    DECLARE POINTER node next
END STRUCTURE

DECLARE pointers poly1, poly2 and poly3 and SET to NULL

PROCEDURE read_Poly ACCEPTS poly , coeff and deg
    DECLARE temp pointer and SET to poly
    DECLARE newnode pointer and ALLOCATE MEMORY
    SET newnode→next = NULL
    SET newnode→coefficent = coeff
    SET newnode→degree = deg
    IF poly = NULL
        SET poly = newnode
    ELSE
        WHILE temp->next!= NULL
            SET temp = temp->next
        ENDWHILE
        SET temp->next = newnode
    ENDIF
    RETURN poly
END PROCEDURE

PROCEDURE display ACCEPTS POINTER poly
    WHILE poly->next!= NULL
        DISPLAY poly->coefficent, poly->degree
        SET poly = poly->next
    ENDWHILE
    DISPLAY poly->coefficent, poly->degree
END PROCEDURE
PROCEDURE addPoly
    CREATE pointer newnode and ALLOCATE MEMORY
    SET newnode→next = NULL
    CRETE pointer temp1 and SET temp1 = poly1
    CRETE pointer temp2 and SET temp2 = poly2
    CRETE pointer temp3 and SET temp3 = poly3
```

```
            poly3 = read_Poly(poly3, (temp1->coefficent+temp2->coefficent),
temp1->degree);
            temp1 = temp1->next;
            temp2 = temp2->next;

        }
        else if (temp1->degree > temp2->degree)
        {
            poly3 = read_Poly(poly3, temp1->coefficent, temp1->degree);
            temp1 = temp1->next;
        }
        else
        {
            poly3 = read_Poly(poly3, temp2->coefficent, temp2->degree);
            temp2 = temp2->next;
        }
    }
    while (temp1!=NULL)
    {
        poly3 = read_Poly(poly3, temp1->coefficent, temp1->degree);
        temp1 = temp1->next;
    }
    while (temp2!=NULL)
    {
        poly3 = read_Poly(poly3, temp2->coefficent, temp2->degree);
        temp2 = temp2->next;
    }
    display(poly3);

}



void main()
{
    int choice, coeff, deg;
    printf("\nSelect an option:\n");
    printf("1. Enter term for polynomial 1\n");
    printf("2. Enter term for polynomial 2\n");
    printf("3. Perform Addition\n");
    printf("4. Exit\n");
    while(1)
    {
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice)
        {
            case 1:
                printf("Enter coefficient and degree for the term: ");
                scanf("%d%d", &coeff, &deg);
                poly1 = read_Poly(poly1, coeff, deg);
                break;
            case 2:
                printf("Enter coefficient and degree for the term: ");
                scanf("%d%d", &coeff, &deg);
                poly2 = read_Poly(poly2, coeff, deg);
                break;
            case 4:
```

```
    WHILE temp1!= NULL and temp2!= NULL
        IF temp1->degree = temp2->degree
            SET poly3 = read_Poly(poly3, (temp1->coefficent+temp2-
>coefficent), temp1->degree)
            SET temp1 = temp1->next
            SET temp2 = temp2->next
        ELSE IF temp1->degree > temp2->degree
            SET poly3 = read_Poly(poly3, temp1->coefficent, temp1-
>degree)
            SET temp1 = temp1->next
        ELSE
            SET poly3 = read_Poly(poly3, temp2->coefficent, temp2-
>degree)
            SET temp2 = temp2->next
        ENDIF
    ENDWHILE
    WHILE temp1!= NULL
        SET poly3 = read_Poly(poly3, temp1->coefficent, temp1-
>degree)
        SET temp1 = temp1->next
    ENDWHILE
    WHILE temp2!= NULL
        SET poly3 = read_Poly(poly3, temp2->coefficent, temp2-
>degree)
        SET temp2 = temp2->next
    ENDWHILE
    CALL PROCEDURE display(poly3)
END PROCEDURE

PROCEDURE main
           DISPLAY MENU
    INPUT choice FROM USER
    WHILE TRUE
        CASE choice OF
            1:
                INPUT VALUE coeff and deg FROM USER
                SET poly1 = read_Poly(poly1, coeff, deg)
            2:
                INPUT VALUE coeff and deg FROM USER
                SET poly2 = read_Poly(poly1, coeff, deg)
            3:
                CALL addPoly()
            4:
                CALL exit()
            DEFAULT:
                DISPLAY Invalid choice
        END CASE
    END WHILE
END PROCEDURE
END PROGRAM
```

```
                exit(0);
                break;
            case 3:
                addPoly();
                break;

            default:
                printf("Invalid choice!\n");
        }
    }
}
```

OUTPUT

```
ENTER THE NUMBER OF TERMS IN P1 : 3
ENTER THE COEFF AS WELL AS EXP : 1 1
ENTER THE COEFF AS WELL AS EXP : 2 2
ENTER THE COEFF AS WELL AS EXP : 3 3

ENTER THE NUMBER OF TERMS IN P2 :4
ENTER THE COEFF AS WELL AS EXP : 2 1
ENTER THE COEFF AS WELL AS EXP : 3 2
ENTER THE COEFF AS WELL AS EXP : 4 3
ENTER THE COEFF AS WELL AS EXP : 5 4

FIRST  :1x^1 + 2x^2 + 3x^3
SECOND :2x^1 + 3x^2 + 4x^3 + 5x^4
SUM :3x^1 + 5x^2 + 7x^3 + 5x^4
```

RESULT

Program Executed Successfully

PROGRAM

```
#include<stdio.h>
void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
}
void maxheapify(int a[],int n,int i){
    int largest=i;
    int l=2*i+1;
    int r=2*i+2;
    if(l<n&&a[l]>a[largest])
      largest=l;
    if(r<n&&a[r]>a[largest])
      largest=r;
    if(largest!=i){
        swap(&a[i],&a[largest]);
        maxheapify(a,n,largest);
    }
}
void heapsort(int a[],int n){

    for(int i=n/2-1;i>=0;i--){
       maxheapify(a,n,i);
    }
    for(int i=n-1;i>0;i--){
        swap(&a[0],&a[i]);
        maxheapify(a,i,0);

    }
}
int main(){
    int n,a[20],i;
    printf("Enter size of array : ");
    scanf("%d",&n);
    printf("Enter array : ");
    for(i=0;i<n;i++){
        scanf("%d",&a[i]);
    }
    heapsort(a,n);

    printf("Sorted array is\n");
    for(i=0;i<n;i++){
        printf("%d\t",a[i]);
    }
    printf("\n");
}
```

OUTPUT

```
Enter size of array : 5
Enter array : 7 3 0 9 8
Sorted array is
0 3 7 8 9
```

DATE : 28-11-2023

HEAP SORT

AIM

To implement Heap Sort Algorithm.

PSEUDOCODE

```
INPUT : Unsorted array of integers
OUTPUT : Sorted array of integers

PROCEDURE swap ACCEPTS POINTER x,POINTER y
    SET temp=*x
    SET *x=*y
    SET *y=temp
END PROCEDURE
PROCEDURE maxheapify ACCEPTS array,size,i
    SET largest= i
    SET l=2*i+1
    SET r=2*i+2
    IF l<size AND array[l]>array[largest]
        SET largest=l
    IF r<size AND array[r]>array[largest]
        SET largest=r
    IF largest!=i
        CALL swap(array[i] ,array[largest])
        CALL maxheapify(array,size,largest)
    END IF
END PROCEDURE
PROCEDURE heapsort ACCEPTS array,size
    FOR i=size/2 -1 TO 0
         CALL maxheapify(array,size,i)
         INCREMENT i
    END FOR
    FOR i=size-1 TO 1
        CALL swap(*array[0],*array[i])
        CALL maxheapify(array,i,0)
        INCREMENT i
    END FOR
END PROCEDURE
PROCEDURE main
    INPUT size
    INPUT array[size]
    CALL heapsort(array,size)
    PRINT array
END PROCEDURE
```

RESULT

Program Executed Successfully

PROGRAM

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

#define SIZE 10

struct Node{
    int data;
    struct Node *next;
};

struct HashTable{
    struct Node *chain[SIZE];
};

void init(struct HashTable *ht){
    for (int i = 0; i < SIZE; i++){
        ht->chain[i] = NULL;
    }
}

void insert(struct HashTable *ht, int value){
    int key = value % SIZE;

    struct Node *newNode = malloc(sizeof(struct Node));

    newNode->data = value;
    newNode->next = NULL;

    if (ht->chain[key] == NULL){
        ht->chain[key] = newNode;
    }

    else{
        struct Node *temp = ht->chain[key];
        while (temp->next){
            temp = temp->next;
        }
        temp->next = newNode;
    }
    printf("Entered value %d added to hash table\n", value);
}

void printHashTable(struct HashTable *ht){
    for (int i = 0; i < SIZE; i++){
        printf("chain[%d]-->", i);
        struct Node *temp = ht->chain[i];

        while (temp != NULL){
            printf("%d -->", temp->data);
            temp = temp->next;
        }
        printf("NULL\n");
    }
}

void main()
{
```

DATE : 28-11-2023

HASHING

AIM

To implement Hashing Algorithm.

PSEUDOCODE

```
INPUT : No.of elements and value of each element to be added to the
hash table.
OUTPUT : Hash table

DEFINE SIZE 10

STRUCTURE Node
    DECLARE INTEGER data
    DECLARE POINTER Node next
END STRUCTURE

STRUCTURE HashTable
    DECLARE POINTER TO Node chain[SIZE]
END STRUCTURE

PROCEDURE init ACCEPTS POINTER HashTable ht
    FOR i = 0 to SIZE - 1
        SET ht->chain[i] = NULL
        INCREMENT i
    END FOR
END PROCEDURE

PROCEDURE insert ACCEPTS POINTER HashTable ht , value
    SET key = value % SIZE
    ALLOCATE newNode
    SET newNode->data = value
    SET newNode->next = NULL
    IF ht->chain[key] = NULL
        SET ht->chain[key] = newNode
    ELSE
        SET temp = ht->chain[key]
        WHILE temp->next != NULL
            SET temp = temp->next
        END WHILE
        SET temp->next = newNode
    END IF
END PROCEDURE

PROCEDURE printHashTable ACCEPTS POINTER HashTable ht
    FOR i = 0 TO SIZE - 1
        PRINT "chain[" + i + "]-->"
```

```
    struct HashTable ht;
    init(&ht);

    int num, val;
    printf("Enter the number of elements: ");
    scanf("%d", &num);

    for (int i = 0; i < num; i++){
        printf("Enter element %d: ", i + 1);
        scanf("%d", &val);
        insert(&ht, val);
    }

    printf("\n\nThe hash table is:\n\n");
    printHashTable(&ht);
}
```

OUTPUT

```
Enter the number of elements: 5
Enter element 1: 2
Entered value 2 added to hash table
Enter element 2: 1
Entered value 1 added to hash table
Enter element 3: 12
Entered value 12 added to hash table
Enter element 4: 4
Entered value 4 added to hash table
Enter element 5: 44
Entered value 44 added to hash table


The hash table is:

chain[0]-->NULL
chain[1]-->1 -->NULL
chain[2]-->2 -->12 -->NULL
chain[3]-->NULL
chain[4]-->4 -->44 -->NULL
chain[5]-->NULL
chain[6]-->NULL
chain[7]-->NULL
chain[8]-->NULLchain[9]-->NULL
```

```
        SET temp = ht->chain[i]
        WHILE temp != NULL
            PRINT temp->data , " -->"
            SET temp = temp->next
        END WHILE
        PRINT "NULL"
        INCREMENT i
    END FOR
END PROCEDURE

PROCEDURE main
    DECLARE HashTable ht
    CALL init(&ht)
    PRINT "Enter the number of elements: "
    INPUT num
    FOR i = 0 TO num - 1
        PRINT "Enter element " , (i + 1) , ": "
        INPUT val
        CALL insert(&ht, val)
        INCREMENT i
    END FOR
    CALL printHashTable(&ht)
END PROCEDURE
```

RESULT

```
Program Executed Successfully
```

PROGRAM

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

#define SIZE 5

int Q[SIZE];
int F = -1 ;
int R = -1 ;

bool isEmpty(){
            return F == -1;
}

bool isFull(){
            return (R+1)%SIZE == F;
}

void addRear(){
            if(isFull()){
                        printf("Queue overflow\n");
            }
            else{
                        if(isEmpty()){
                                    F = 0;
                                    R = 0;
                        }
                        else{
                                    R = (R+1)%SIZE;
                        }
                        printf("Enter element : ");
                        scanf("%d" , &Q[R]);
            }
}
void addFront(){
            if(isFull()){
                        printf("Queue overflow\n");
            }
            else{
                        if(isEmpty()){
                                    F = 0;
                                    R = 0;
                        }
                        else if(F == 0){
                                    F = SIZE -1;
                        }
                        else{
                                    F--;
                        }
                        printf("Enter element : ");
                        scanf("%d" , &Q[F]);
            }
}
void rmRear(){
            if(isEmpty()){
                        printf("Queue underflow\n");
            }
```

DATE : 28-11-2023

DOUBLY LINKED LIST

AIM

To implement the Data Structure Doubly Linked List.

PSEUDOCODE

```
INPUT : Input of selected operation from menu
OUTPUT :  Output of selected operation

DEFINE SIZE 5
DECLARE ARRAY Q[SIZE]
SET front = -1
SET rear = -1

PROCEDURE ISEMPTY
    RETURN front == -1
END PROCEDURE
PROCEDURE ISFULL
    RETURN (rear + 1) % SIZE == front
END PROCEDURE
PROCEDURE ADDREAR
    IF ISFULL
        PRINT "Queue overflow"
    ELSE
        IF ISEMPTY
            SET front = 0
            SET rear = 0
        ELSE
            SET rear = (rear + 1) % SIZE
        END IF
        INPUT Q[rear]
    END IF
END PROCEDURE

PROCEDURE ADDFRONT
    IF ISFULL
        PRINT "Queue overflow"
    ELSE
        IF ISEMPTY
            SET front = 0
            SET rear = 0
        ELSE IF front == 0
            SET front = SIZE - 1
        ELSE
            SET front = front - 1
        END IF
        INPUT Q[front]
    END IF
END PROCEDURE
PRCEDURE REMOVEREAR
    IF ISEMPTY
```

```
            else{
                        printf("Removing %d\n" , Q[R]);
                        if(F == R){
                                    F = -1;
                                    R = -1;
                        }
                        else if(R == 0){
                                    R = SIZE - 1;
                        }
                        else{
                                    R--;
                        }
            }
}
void rmFront(){
            if(isEmpty()){
                        printf("Queue underflow\n");
            }
            else{
                        printf("Removing %d\n" , Q[F]);
                        if (F == R){
                                    F = -1;
                                    R = -1;
                        }
                        else{
                                    F = (F+1)%SIZE;
                        }
            }
}
void display(){
            if(isEmpty()){
                        printf("Queue is Empty ! \n");
                        return;
            }
            printf("Queue : ");
            int i = F;
            while(i != R){
                        printf("%d\t",Q[i]);
                        i = (i+1)%SIZE;
            }

            printf("%d\n",Q[R]);
}
void size(){
            int size_v;
            if(isEmpty())
                        size_v = 0;
            else if(isFull())
                        size_v = SIZE;
            else if(F > R)
                        size_v = SIZE - (F - R) + 1;
            else
                        size_v = F - R + 1;

            printf("Size is %d\n" , size_v);
}
void main(){
            int option;
            printf("\n\nQueue operations\n");
```

```
        PRINT "Queue underflow"
    ELSE
        PRINT Q[rear] " is removed"
        IF front == rear
            SET front = -1
            SET rear = -1
        ELSE IF rear == 0
            SET rear = SIZE - 1
        ELSE
            SET rear = rear - 1
        END IF
    END IF
END PROCEDURE

PROCEDURE REMOVEFRONT
    IF ISEMPTY
        PRINT "Queue underflow"
    ELSE
        PRINT Q[front] " is removed"
        IF front == rear
            SET front = -1
            SET rear = -1
        ELSE
            SET front = (front + 1) % SIZE
        END IF
    END IF
END PROCEDURE

PROCEDURE DISPLAY
    IF ISEMPTY
        PRINT "Queue is empty"
        RETURN
    END IF
    SET i = front
    WHILE i != rear
        PRINT Q[i]
        SET i = (i + 1) % SIZE
    END WHILE
    PRINT Q[rear]
END PROCEDURE

PROCEDURE SIZE
    DECLARE INTEGER size
    IF ISEMPTY
        SET size = 0
    ELSE IF ISFULL
        SET size = SIZE
    ELSE IF front > rear
        SET size = SIZE - (front - rear) + 1
    ELSE
        SET size = rear - front + 1
    END IF
    PRINT size
END PROCEDURE
```

```
            printf("1.Display\n");
            printf("2.Add to rear\n");
            printf("3.Add to front\n");
            printf("4.Remove from rear\n");
            printf("5.Remove from front\n");
            printf("6.Size\n");
            printf("7.Exit\n");
            while(true){
                        printf("Select an option : ");
                        scanf("%d",&option);
                        printf("\n");
                        switch(option){
                                    case 1      :display();break;
                                    case 2      :addRear();break;
                                    case 3      :addFront();break;
                                    case 4      :rmRear();break;
                                    case 5      :rmFront();break;
                                    case 6      :size();break;
                                    case 7      :exit(0);break;
                                    default     :printf("Invalid option
entered\n");
                        }
            }
}
```

OUTPUT

```
1.CREATE N NODES
2.DISPLAY
3.INSERTION AT FRONT
4.INSERTION AT REAR
5.INSERTION IN BETWEEN
6.DELETION AT FRONT
7.DELETION AT REAR
8.DELETION IN BETWEEN
9.EXIT
ENTER CHOICE: 1
ENTER NO OF NODES: 4
ENTER ELEMENTS: 1 2 3 4
ENTER CHOICE: 2
1234
ENTER CHOICE: 3
ENTER ELEMENT TO BE ADDED IN THE FRONT: 0
ENTER CHOICE: 4
ENTER ELEMENT TO BE ADDED AT THE END: 5
ENTER CHOICE: 5
ENTER POSITION WHERE ELEMENT IS TO BE ADDED: 3
ENTER ELEMENT: 7
ENTER CHOICE: 2
0172345
ENTER CHOICE: 6
Deleted Element: 0
ENTER CHOICE: 7
Deleted Element: 5
ENTER CHOICE: 8
ENTER POSITION: 4
Deleted Element: 3
ENTER CHOICE: 9
```

```
PRCEDURE MAIN
    PRINT "1. Display"
    PRINT "2. Add rear"
    PRINT "3. Add front"
    PRINT "4. Remove rear"
    PRINT "5. Remove front"
    PRINT "6. Size"
    PRINT "7. Exit"
    WHILE TRUE
        INPUT choice
        CASE choice OF
            1: CALL DISPLAY
            2: CALL ADDREAR
            3: CALL ADDFRONT
            4: CALL REMOVEREAR
            5: CALL REMOVEFRONT
            6: CALL SIZE
            7: EXIT
        END CASE
    END WHILE
END PROCEDURE
```

RESULT

```
Program Executed Successfully
```

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
struct node {
    int data;
    struct node *next;
};
int top=-1,f=0,r=-1;
struct node * addLast(struct node *head,int data){
    struct node *newnode;
    newnode = malloc(sizeof(struct node));
    newnode->data = data;
    newnode->next = NULL;
    if (head==NULL){
        head=newnode;
    }
    else{
        struct node *current;
        current = head;
        while (current->next != NULL){
            current =  current->next;
        }
        current->next=newnode;
    }
    return head;
}
bool isEmptyS(){
    return (top==-1);
}
bool isEmptyQ(){
    return (f>r);
}
void  push(int n, int S[n],int data){
    S[++top]=data;
}
int  pop(int n,int S[n]){
    top=top-1;
    return S[top+1];
}
int  deQueue(int  n,int Q[n]){
    f=f+1;
    return Q[f-1];
}
void enQueue(int n,int Q[n],int data){
    Q[++r]=data;
}
void main(){
    int n,data,m;
    printf("Enter number of vertices : ");
    scanf("%d",&n);
    struct node *aList[n];
    printf("Enter vertices :\n");
    for (int i = 0; i < n ; i++){
        aList[i]=NULL;
        scanf("%d",&data);
        aList[i]=addLast(aList[i],data);
```

DATE : 20-12-2023

GRAPH TRAVERSALS

AIM

To implement Graph Traversal Algorithms.

PSEUDOCODE

```
INPUT : No. of vertices and their neighbours
OUTPUT :  Result of BFS and DFS on the inputted graph

STRUCTURE node
    INT data
    STRUCTURE node next
END STRUCTURE
SET top = -1, f = 0,  r = -1

PROCEDURE addLast ACCEPTS head, data
    ALLOCATE newnode
    SET newnode.data = data
    SET newnode.next = NULL
    IF head == NULL
        SET head = newnode
    ELSE
        SET current=head
        WHILE current.next != NULL
            SET current = current.next
        END WHILE
        SET current.next=newnode
    END IF
    RETURN head
END PROCEDURE

PROCEDURE isEmptyS
    RETURN  top == -1
END PROCEDURE

PROCEDURE isEmptyQ
    RETURN  f > r
END PROCEDURE
PROCEDURE push ACCEPTS n, S[n], data
    SET top = top + 1
    SET S[top] = dataEND PROCEDURE

PROCEDURE pop ACCEPTS n, S[n]
    SET top = top – 1
    RETURN  S[top + 1]
END PROCEDURE

PROCEDURE deQueue ACCEPTS n, Q[n]
    SET f = f + 1
    RETURN  Q[f – 1]
END PROCEDURE
```

```
    }
    for (int i=0;i<n;i++){
        printf("Enter no neighbours of vertex %d\n",aList[i]->data);
        scanf("%d",&m);
        printf("Enter neighbours : \n");
        for (int j=0;j<m;j++){
            scanf("%d",&data);
            aList[i]=addLast(aList[i],data);
        }
    }
    printf("Adjescentsy List :\n");
    for (int i = 0;i<n;i++){
        struct node *current;
        current=aList[i];
        while(current->next!=NULL){
            printf("%d-->",current->data);
            current=current->next;
        }
        printf("%d\n",current->data);
    }
    int Stack[n],visit[n],Queue[n];
    push(n,Stack,aList[0]->data);
    int index = -1;
    while(!isEmptyS()){
        struct node *current;
        int high = pop(n,Stack);
        visit[++index]=high;
        for (int i=0;i<n;i++){
            if (aList[i]->data==high){
                current = aList[i]->next;
            }
        }
        while(current!=NULL){
            int f=0;
            for (int i=0;i<=index;i++){
                if(visit[i]==current->data){
                    f=1;
                }
            }
            for(int i=0;i<=top;i++){
                if(Stack[i]==current->data){
                    f=1;
                }
            }
            if (f!=1){
                push(n,Stack,current->data);
            }
            current=current->next;
        }
    }

    printf("Depth First Search : \n");
    for (int i = 0;i<=index;i++){
        printf("%d ",visit[i]);
    }
    printf("\n");
    index=-1;
    visit[++index] = aList[0]->data;
    enQueue(n,Queue,aList[0]->data);
```

```
PROCEDURE enQueue ACCEPTS n, Q[n], data
    SET r = r + 1
    SET Q[r] = data
END PROCEDURE

PROCEDURE main
    INPUT n
    DECLARE node ARRAY aList[n]
    FOR i = 0 to n – 1
        SET aList[i] = NULL
        INPUT data
        SET aList[i] = CALL addLast(aList[i] , data)
    END FOR
    FOR i = 0 to n – 1
        INPUT m
        FOR j = 0 to m – 1
            INPUT data from user
            SET aList[i] = CALL addLast(aList[i], data)
        END FOR
    END FOR
    FOR i = 0 to n – 1
        SET current = aList[i]
        WHILE current.next != NULL
            PRINT current.data  -->
            current = current.next
        END WHILE
        PRINT current.data
        INCREMENT i
    END FOR
    DECLARE arrays Stack , visit , Queue[n]
    CALL push (n, Stack, aList[0].data)
    SET index = -1
    WHILE PROCEDURE isEmptyS RETURNS FALSE
        SET high = CALL PROCEDURE pop PASSING n, Stack
        SET index = index + 1, visit[index] = high
        FOR i = 0 to n – 1
            IF aList[i]→data == high
                SET current = aList[i]→next
                END IF
            INCREMENT i
        END FOR
        WHILE current != NULL
            SET f = 0
            FOR i = 0 to index
                IF visit[i] == current→data
                    SET f = 1
                END IF
                INCREMENT i
            END FOR
            FOR i = 0 to top
                IF Stack[i] == current→data
                    SET f = 1
                END IF
                INCREMENT i
            END FOR
            IF f != 1
                CALL push(n, Stack, current.data)
```

```
    while(!isEmptyQ()){
        struct node *current;
        int low = deQueue(n,Queue);
        for (int i = 0;i<n;i++){
            if (aList[i]->data==low){
                current = aList[i];
            }
        }
        while(current->next!=NULL){
            current=current->next;
            int f=0;
            for (int i = 0 ; i<=index;i++){
                if(visit[i]==current->data){
                    f=1;
                }
            }
            if (f!=1){
                visit[++index]=current->data;
                enQueue(n,Queue,current->data);
            }
        }
    }
    printf("Breadth First Search : \n");
    for(int i =0;i<=index;i++){
        printf("%d ",visit[i]);
    }
    printf("\n");
}
```

OUTPUT

```
Enter number of vertices : 2
Enter vertices :
1 2
Enter no neighbours of vertex 1
2
Enter neighbours :
3 4
Enter no neighbours of vertex 2
3
Enter neighbours :
5 6 7
Adjescentsy List :
1-->3-->4
2-->5-->6-->7
Depth First Search :
1 4 3
Breadth First Search :
1 3 4
```

```
            END IF
            SET current = current→next
        END WHILE
    END WHILE
    FOR i = 0 to index
        PRINT visit[i]
        INCREMENT i
    END FOR
    SET index = -1
    SET index = index + 1, visit[index] = aList[0]→data
    CALL enQueue (n, Queue, aList[0].data)
    WHILE !isEmptyQ
        SET low = CALL deQueue (n, Queue)
        FOR i = 0 to n – 1
            IF aList[i]→data == low
                SET current = aList[i]
            END IF
            INCREMENT i
        END FOR
        WHILE current→next != NULL
            current = current→next
            SET f = 0
            FOR i = 0 to index
                IF visit[i] == current.data
                    SET f = 1
                END IF
                INCREMENT i
            END FOR
            IF f != 1
                 SET index = index + 1, visit[index] = current.data
                 CALL enQueue (n, Queue, current.data)
             END IF
        END WHILE
    END WHILE
    FOR i = 0 to index
        PRINT visit[i]
        INCREMENT i
    END FOR
```

RESULT

```
Program Executed Successfully
```

PROGRAM

```
#include <stdio.h>
int complete_node = 15;
char tree[] = { 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', '\0', '\0',
'J', '\0', 'K', 'L'};

int get_right_child(int index){
    if(tree[index]!='\0' && ((2*index)+2)<=complete_node)
        return (2*index)+2;
    return -1;
}

int get_left_child(int index){
    if(tree[index]!='\0' && (2*index+1)<=complete_node)
        return 2*index+1;
    return -1;
}

void preorder(int index){
    if(index>=0 && tree[index]!='\0'){
        printf(" %c ",tree[index]);
        preorder(get_left_child(index));
        preorder(get_right_child(index));
    }
}

void postorder(int index){
    if(index>=0 && tree[index]!='\0'){
        postorder(get_left_child(index));
        postorder(get_right_child(index));
        printf(" %c ",tree[index]);
    }
}

void inorder(int index){
    if(index>=0 && tree[index]!='\0'){
        inorder(get_left_child(index));
        printf(" %c ",tree[index]);
        inorder(get_right_child(index));
    }
}

int main(){
    printf("Preorder:\n");
    preorder(0);
    printf("\nPostorder:\n");
    postorder(0);
    printf("\nInorder:\n");
    inorder(0);
    printf("\n");
    return 0;
}
```

DATE : 20-12-2023

TREE USING ARRAY

AIM

To implement Tree Data Structure Using Array.

PSEUDOCODE

```
INPUT : Nodes in binary tree
OUTPUT :  Result of Inorder, preorder and postorder traversal in
given tree.

PROCEDURE get_right_child ACCEPTS index
    IF tree[index] != '\0' AND ((2 * index) + 2) <= complete_node
        RETURN (2 * index) + 2
    ELSE
        RETURN -1
    END IF
END PROCEDURE

PROCEDURE get_left_child ACCEPTS index
    IF tree[index] != '\0' AND (2 * index + 1) <= complete_node
        RETURN 2 * index + 1
    ELSE
        RETURN -1
    END IF
END PROCEDURE

PROCEDURE preorder ACCEPTS index
    IF index >= 0 AND tree[index] != '\0'
        PRINT tree[index]
        CALL preorder(get_left_child(index))
        CALL preorder(get_right_child(index))
    END IF
END PROCEDURE
PROCEDURE postorder ACCEPTS index
    IF index >= 0 AND tree[index] != '\0'
        CALL postorder(get_left_child(index))
        CALL postorder(get_right_child(index))
        PRINT tree[index]
    END IF
END PROCEDURE
PROCEDURE inorder ACCEPTS index
    IF index >= 0 AND tree[index] != '\0'
        CALL inorder(get_left_child(index))
        PRINT tree[index]
        CALL inorder(get_right_child(index))
    END IF
END PROCEDURE
```

OUTPUT

```
Preorder:
 A B D H I E C F J G K L
Postorder:
H I D E B J F K L G C A
Inorder:
 H D I B E A J F C K G L
```

```
PROCEDURE main
    CALL preorder(0)
    CALL postorder(0)
    CALL inorder(0)
END PROCEDURE
```

RESULT

```
Program Executed Successfully
```