Code

```
#include <stdio.h>
Void input_Sparse (int SM[][10], int m, int n){
        printf(" Enter the no. of rows and
                        columns of the matrix)
        Scanf (" %d %d", &m, &n);
        printf(" Enter the elements: \n);
        for(int i=0; i<m; i++){
                for(int j=0; j<n; j++){
                        Scanf(" %d", &SM[i][j]);
        }
    }
}

int compsparse(int CM[][3], int SM[][10], int m, int n){
        int temp=0,
        for(int i=0; i<m; i++){
                for(int j=0; j<n; j++){
                if (SM[i][j]!= 0) {
                        CM[temp][0]=i
                        CM[temp][1]=j
                        CM[temp][2]=SM[i][j]
                        temp++
                }
            }
        }
        return temp;
}

Void printSparse ( int CM[][3], int temp. int m, int n){
        float Sparsity= ((m*n)-temp) /(m*n);
        printf(" Sparsity is %f \n", sparsity);
        printf(" ——— Row no. ——— Col no ——— Value \n);
        printf(" %15d %15d %15d \n", m,n, temp);
```

17/10/23.

EXP - 6

SPARSE REPRESENTATION

Aim: Given a Sparse matrix, represent and store it using an efficient method. Also find the Sparsity.

[Sparsity of a matrix can be quantified with a score which is the number of elements which are zero divided by total number of elements]

Algorithm

Step 0: Start

Step 1: Initialise array a[100][100], its row size and column size

Step 2: Use structures to represent in triplet form

Step 3: Ask the user to enter the element

Step 4: Call function sparse.

Function

Step 0: Start

Step 1: for (i=0; i<row, i++ count = 0, k=0)

for j=0, j<col, j++

check if a[i][j] != 0

count = count+1

b[k].row = i

b[k] col = j

b[k] value = a[i][j]

k++

```
for(int i=0; i<temp; i++){
        printf("%15d %15d %15d \n", CM[i][0], CM[i][1],
                                    CM[i][2]);
    }
}

Void main(){
    int SM[10][10], CM[10][3], m, n;
    printf("Enter the no. of rows and colums of matrix");
    Scanf("%d %d", &m, &n);
    input Sparse (SM, m, n);
    int temp = Comp Sparse (CM, SM, m, n);
    print Sparse (CM, temp, m, n)
}
```

output

Enter the no. of rows and Columns of matrix

3 3

Enter the elements

0
0
1
2
0
0
3
0
0

Sparsity is 0.667

Row no.	Col no.	value
3	3	3
0	2	1
1	0	2
2	0	3

Step 3: Initialise b[0].row = row

b[0].col = col

b[0].val = count

Step 4: print the output

Step 5: sparsity = count / (row * col)

Step 6: print sparcity

Step 7: STOP

RESULT

The desired output is obtained

Code

```
#include <stdio.h>
void main() {
        int r1, c1, n1, r2, c2, n2;
printf(" Enter rows, columns and no. of non zero elements
scanf("%d %d %d", &r1, &c1, &n1);
int m1[n1+1][3];
printf(" Enter rows, columns and no of non zero of m2 ");
scanf("%d %d %d", &r2, &c2, &n2);
int m2[n2+1][3];
if(r1 = r2 && c1 == c2){
        Printf("Addition possible");
        printf(" Enter the triplet representation row wise");
        for(int i=0; i<n1+1; i++)
                scanf("%d %d %d", &m1[i][0], &m1[i]
                &m1[i][2])

        printf(" Enter triplet representation row wise ")
        for(int i=0; i<n2+1; i++)
        scanf("%d %d %d", &m2[i][0], &m2[i][1], &
                m2[i][2])
int result[n1+n2+1][3];
int i=0, j=0; k=0;
while(i<n1 && j<n2)
if(m1[i][0]<m2[j][0] || m1[i][0] == m2[j][0] &&
                m1
{
        result[k][0] = m[i][0];
        result[k][1] = m[i][i];
        result[k][2] = m[i][2];
        k++; i++; }
```

17/10/23

EXP-7

SPARSE SUM

Aim: Input a sparse matrix and find its sum

Algorithm

Step 0: START

Step 1: Declare 3 arrays a[][], b[][] and sum[][] row, col

Step 2: Ask the user to enter the elements of 2 arrays in tripled representation

Step 3: Call the function.

Step 4: Stop

Function

step 1: start

step 2: The function recieves the arrays, rows, col, initilise m=1, n=1, k=1

Step 3: for i=0 to i<row do

for j=0 to j<row do

check if a[m][0] == i a[m][1] = j and

b[n][0] = i b[n][1] = j

if yes

sum[k][0] = a[m][0] sum[k][1] = a[m][1]

sum[k][2] = a[m][2] + b[n][2]

k++, m++, n++

```
else if (
{
    result[i][0] = m2[j][0];
    result[k][1] = m2[j][1];
    result[k][2] = m2[j][2];
        k++; j++ }.
else
{
    result[k][0] = m1[i][0];
    result[k][1] = m1[i][1]
    result[k][2] = m1[i][2] + m2[j][2];
        i++, j++, k++; }
while (i<n1) {
    result[k][0] = m1[i][0];
    result[k][1] = m1[i][1];
    result[k][2] = m2[i][2];
        k++; i++; }
while (j<n2) {
    result[k][0] = m2[j][0];
    result[k][1] = m2[j][1];
    result[k][2] = m2[j][2];
        k++; j++ }
printf("Sum matrix is ");
printf("%d %d %d", r1, c1, k);
for int(i=0; i<k; i++)
    printf("%d %d %d", result[i][0], result
                                [i][1]
                                result[i][2]
}
```

Step 4: Else if, check if a[m][0] == i & a[m][1] = j

If yes sum[k][0] = a[m][0] sum[k][1] a[m][1]

sum[k][2] = a[m][2]

m++, k++.

Step 5: Else if. check b[n][0] == i and b[n][1] = j

if yes, sum[k][0] = b[n][0] sum[k][1]

b[n][1]

sum[k][2] = b[n][2]

n++, k++

end for j

end for i

Step 6: Sum[0][0] = a[0][0]

Sum[0][1] = a[0][1]

Sum[0][2] = K - 1

Step 7: Print the Sum matrix

Step 8: for i = 0 to sum[0][2] do

for j = 0 to 3 do

print C[i][j]

```
else {
    printf ("Addition not possible");
    }
}
```

OUTPUT

Enter rows, columns and no. of non zero value of matrix : 3

Enter row columns and no. of non zero values of matrix 3

Enter triplet representation 1

3 3 2
0 2 1
2 2 2

~~Sum matrix~~

Enter triplet representation 2

3 3 2
0 2 1
2 2 2

Sum matrix is

3 3 2
0 2 2
2 2 4

RESULT

The desired output is obtained.

Code

```
#include <stdio.h>
Struct Sparse{
        int row;
        int col;
        int value;
    } a[100], b[100]

Void transpose (struct Sparse a[0], struct Sparse b[0]){
        int i, j, currentb;
            b[0].row = a[0].col;
            b[0].col = a[0].row;
            int t = a[0].value;
                b[0].value = t;
            if t>0 {
                currentb = 1;
            for(i=0; i<a[0].col; i++){
                    for(j=1; j<=t; j++){
                        if (a[j].col == i){
                        b[currentb].row = a[j].col;
                        b[currentb].col = a[j].row
                        b[currentb].value = a[j].value;
                        currentb++;
                    }
                }
            }
        }
        printf("the transpose matrix is \n");
        for(i=0; i<=b[0].value; i++){
                printf("%d %d %d", b[i].row, b[i].col,
                                    b[i].value);
            }
    }
```

17/10/23 Exp - 8

TRANSPOSE

Aim: Input the representation of the sparse matrix find the representation and its transpose

Algorithm

step 0: Start

step 1: Initialise a structure with elements, row, col and value

Step 2: Enter the sparse matrix representation of elements

Step 3: Call transpose function.

Step 4: STOP.

Function

Step 1: Start

Step 2: Assign b[0]. row = a[0]. col

b[0] col = a[0]. row

t = a[0]. value b[0]. value = t

Step 3: Initilise count = 0, current b = 1.

if (t>0) then

for i = 0, to i < a[0]. col - 1 do

for j = 1 to j < t do

check if a[j]. col = i

b[current b]. row = a[j]. col

b[current b]. col = a[j]. row

b[current b]. val = a[j]. val

Increment current b

```
int main() {
    int r1, i ;
    printf("enter r1 : ");
    scanf("%d", &r1);
    printf("Enter triplet form of matrix ");
    for(i=0; i<r1 ; i++) {
        scanf("%d %d %d", &a[i].row,
                &a[i].col, &a[i].value);
    }

    transpose (a, b);
    return 0;
}
```

Output

Enter r1 : 4

Enter triplet form of matrix

3	3	3
0	0	1
1	2	3
2	1	5

The transposed matrix is

3	3	3
0	0	1
1	2	5
2	1	3

Step 4: end if

Step 5: end for

Step 6: end for

Step 7: end if

Step 8: STOP

Result

The desired output was obtained

code

```
#include <stdio.h>
struct sparse {
        int row;
        int col;
        int val;
    } a[100], b[100]

Void Symmetric (struct sparse a[100], struct sparse b[100])
  {
    int i, j, currentb, flag=0;
    b[0].row = a[0].col;
    b[0].col = a[0].row;
    int t = a[0].Value
     b[0].val = t;
     if(t>0){
          currentb = 1
     for (i=0; i<a[0].col; i++){
          for (j=1; j<=t; j++){
               if (a[j].col==i){
                    b[currentb].row = a[j].col;
                    b[currentb].col = a[j].row;
                    b[currentb].Val = a[j].Val;
                    currentb++;
               }
          }
     }
  }
```

Exp-9

17/10/23

Symmetric

Aim : To check whether the given matrix is sparse symmetric using the given representation.

Algorithm

Step 0: Start

Step 1: Initilise the array a[10], transpose array b[10] row, col, ~~transpose~~ and the structure containing elements row, col and value

Step 2: Enter the elements of the sparse matrix in triplet form.

Step 3: Call the transpose() function to find transpose matrix.

Step 4: call the ~~transpose~~ symmetric() function to ~~find transpose matrix~~ check if the matrix is symmetric

Step 5: Stop.

transpose()

Step 0: Start

Step 1: ~~Row~~ pass a[10], Row, col

Step 2: b[0].row , a[0].col , b[0].col = a[0].row

t = a[0].val b[0].val = t

Step 3: Initilise current B = 1.

for i=0 to i<a[0].col-1

for j=1 to j<t

if (a[j].col==i) {

```
for (i=0; i<=t; i++){
    if(a[i].row != b[i].row || a[i].col != b[i].col ||
                                a[i].val != b[i].val)
    {
        printf("not symmetric")
        flag=1;
        break;
    }
}
if(flag==0){
    printf("symmetric \n");
    }
}

int main (){
    int n,i,j;
    printf("enter the no : ")
    scanf("%d", &n);
    printf("enter triple for of matrix \n");
    for (i=0; i<n; i++){
        scanf("%d %d %d", &a[i].row, &a[i].col,
                          &a[i].val);
    }

    transpose(a, b)
    return 0;
}
```

b[current b]. row = a[j]. col

b[current b]. col = a[j]. row

b[current b]. Val = a[j]. val

current b ++

Step 4: End if

Step 5: End for j

Step 6: End for i

Step 7: stop.

is_Symmetric()

Step 1: Start

Step 2: pass a[10], b[10], row, col, t

Step 3: Initilise flag = 0

Step 4. for i=0 to i<t, i++

if (a[i]. col != b[i]. col || a[i]. row != b[i] row || t[i]. value != c[i]. val)

display not symmetric

break

flag = 1

Step 5: If flag is 0, display symmetric

Step 6: stop.

RESULT

The desired output is obtained.

output

enter r, v, 4

Enter triple form of matrix

3 3 3

0 0 1

1 2 3

2 1 5

Not Symmetric

code

```
#include <stdio.h>
int max=5
int top=-1
int stack_arr[5]
Void push (int item){
        if (top= max -1) {
            printf("stack is full \n");
            }
        top ++;
        Stack_arr [top] =item;
        }

    Void pop () {
        if (top= = -1)
            printf ("stack is empty \n");
        int value = stack_arr[top];
        printf ("The removed element is %d",
                                stack_arr [top])
        }

    Void display () {
        printf ("The current stack elements are \n")
        for (int i=0; i<= top ; i ++){
            printf("%d", stack_arr[j]));
        }
    }
```

EXP 10

STACK

26/10/23

Aim: To implement stack using array and perform the Push, Pop, Peak operations.

Algorithm

Step 0: Start

Step 1: Input the size of stack and initilise top = -1

Step 2: using switch statement select the operation to be performed

Step 3: Call the respective functions

Step 4: Stop.

Push () function

Step 1: Start

Step 2: if top == n
display "stack full"
go to step 5

Step3: else
input the value to be inserted

Step4: top = top+1
Stack [top] = Value

Step5: Stop

```
void main() {
    int i, n, choice, n;
    printf(" Enter choice  1: push  2: pop  3: peak );
    scanf(" %d", &choice)
    switch(choice) {
      case1:
          printf ("enter no. of elements");
          scanf(" %d", &n);
          for(i=0; i<n; i++){
            scanf ("%d", &x);
            push(x)
          }
          display()
          break;
      case 2:
          pop();
          display();
          break;

      case3:
          peak()
          break;
      default:
          printf("Wrong choice");
    }
}
```

output

Enter the choice 1. push 2. pop 3. peek 1

enter no. of elements 3

5 26 36

The current stack elements are

5, 26, 36

Pop () function

Step 0: Start

Step 1: if top == -1
display "stack is empty"
go to step 3

Step 2: else
top = top - 1

Step 3 : stop

Peak () function

Step 1: Start

Step 2: if top = -1
display stack empty

Step 3: else
item = stack[top]

Step 4: Return item

Step 5 : STOP

Result :

The desired output is obtained

```
#include <stdio.h>
#include <stlib.h>
#define max 4
int top=-1, stack_arr[4], min top = -1, min-stack[4]

void push (int data) {
        top++;
        stack_arr[top] = data;
        if(min-top==-1 || data <min-stack [min-top]
        {
            min-top++
            minstack [mintop] = data;
        }
    }

int peek () {
        return min-stack[min-top ]
        }

void display() {
        printf(" The elements of minstack are \n")
        for (int p = 0; i<min-top , i++) {
            printf("%d"; minstack[]);
        }
    }
void main () {
        int i, x;
        printf(" Enter the elements \n");
        for (i =0; i<max; i++) {
            scanf ("%d ; &x);
            push(x)
        }
```

Exp 11

Min element in a Stack

26/10/23

Aim: Find the min element in a stack in the order of 1 O(1) using an auxillary stack which keeps track of min

Algorithm

Step 0: start

step 1: initilise two stacks main stack and min stack

step 2: initilise top = -1 and mintop = -1. Input size of stack 'n'

step 3: Call the respective functions

step 4: stop

Push()

step 0: start

step 1: If top < n,

Input the item to be inserted

top = top + 1

item = mainstack [top]

If (mintop == -1 or item < minstack [mintop])

Step 2

mintop ++

minstack [mintop] = item

else

mintop ++

minstack [mintop] = minstack [mintop - 1]

step 3 else

stack full

Step 4: STOP

```
printf("The minimum element is %d \n", peek());
display()
}
```

output

Enter the elements -12 -11 0 -90

The min elemenent is -90

The elements of stack are 0 -11 -12 -90

pop()

Step 0: Start
Step 1: If top >= 0
data = mainstack[TOP]
top --
min top --

else stack is empty

getMin()

step 1: start
step 2: Return minstack [mintop]
step 3: else
return stack is empty

step 4: stop

RESULT

The desired output is obtained

```
#include <stdio.h>
#define max 5
int stack[10], temp-stack[10], top=-1, temp_top=-1;
Void push(int data){
        if (top==max-1) {
            printf("stack is full\n");
            return;
            }
        temp-Stack[++top] = value;
        }
    }
int pop() {
        if(top==-1) {
        return 0;
        }
        return [stack(top--)];
        }

int temp_pop() {
        if (temp_top==-1) {
            return 0;
            }
        return tempstack[temptopp--];
        }
    }
int temp-peek() {
        return temp-stack[temp-top];
        }
Void sort() {
        while (top!=-1) {
            int temp= pop();
```

EXP-12

Sorted push

26/10/23

Aim : Implement a sorted push so that stack is maintained in sorted order

Algorithm

Step 0: Start

Step 1: Initilise two stacks stack[] and tempstack[]

Step 2: Initilise top = -1 , Input stack size

Step 3: Call neccessary functions

Step 4: Stop

Sort() function

Step 0: Start

Step 1:
```
for(i = 0; i<top; i++) {
    for(j = i+1, j<=top; j++) {
        if (stack[i] > stack[j]) {
            temp = stack[i]
            stack[i] = stack[j]
            stack[j] = temp;
        }
    }
```

Step 2: Stop

```
while (temp_top != -1 && temp_peek() > temp)
    {
        push(temp_pop())
    }
    temp_push(temp);
    }
}
int main(){
    int i, x;
    printf("enter 5 elements");
    for(i=0; i<5; i++){
        scanf("%d", &x);
        push(x);
    }
    sort();
    printf("The sorted stack is \n");
    for(i=0; i<temp_top; i++){
        printf("%d", temp_stack[i]);
    }
    return 0;
}
```

Output

Enter 5 elements 12 -90 0 1 567

The sorted stack is

-90 0 1 12 567

RESULT

The desired output is obtained

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

char infix[100], postfix[100], stack_array[100];
int top = -1;

void push (int data) {
        stack_arr (++ top ) = data

char pop () {
        char sym = stack_arr [top];
        top --;
        return sym;
    }
int isEmpty() {
        if (top == -1) {
            return 1;
        else
            return 0;
        }

int precedence (char symbol)
    {
    switch (symbol)
      {
      case '^' :
            return 3;
            break;

      case '*' :
      case '/' :
            return 2;
            break;

      case '+' :
      case '-' :
            return 1;
            break;
```

EXP 13

Infix to Postfix

26/10/23

Aim: construct a given infix expression to postfix expression

Algorithm

Step 0: Start

Step 1: Declare the stack_arr [MAX] array

Step 2: Enter the expression in string format

Step 3: Scan from left to right

Step 4: If symbol is operand print to screen

Step 5: If symbol is ~~left~~ right paranthesis pop all the operators from stack until ~~right~~ left paranthesis

Step 6: Discard left and right parenthesis

Step 7: If symbol is an operator,

If precedence of operators in stack is greater than or equal to current operator then

Step 8: pop the operators in the stack and print them to the screen and push the operator onto stack.

Step 9: Else push the current operator

```
default :
        return 0;

    }
}

void intopo() {
        char symbol, nesct;
        int i, j = 0;
        for(i=0; infix [i] != '\0'; i++){

                symbol = infix [i];
                switch(symbol){

        case : '('

                push (symbol);
                break;

        case ')' :
                while (nesct=pop() != '(') {
                    postfix [j++] = nesct;
                }
                break;

    case '+':
    case '-':
    case '*':
    case '/':
    case '^':
            while ( ! is emptly && precedence(symbol)<=
                                precedence stack_arr[i]))
            {

            postfix [j++] = pop ();
            }
```

```
push (symbol);
break;
default:
    postfix[j++] = symbol
  }
}
while (! is Empty ) {
        postfix[j++] = pop();
      }
      postfix[j++] = '\0';
printf ("The postfix expression is \n");
for (i=0, postfix[i] != '\0'; i++){
        printf("%c", postfix[i]);
    }
}
Void main ){
  int p
  ~~Postfix~~ printf ("enter the ~~post~~ infix expression ");
  scanf ("%s", & infix);
  in to po();
}
```

OUTPUT

Enter the infix expression 7+(9-5)*2

The postfix expression is

795-2*+

RESULT

The desired output is obtained

```
#include <stdio.h>
#include <math.h>
#define MAX 10
int top = -1, stack [100];
char postfix [100];
void push (int data) {
        if (top == MAX-1) {
        printf("stack is full");
        }
        stack [++top] = data;
        }
int pop() {
        if (top == -1) {
            printf("empty stack \n");
            }
        int val = stack (top--);
        return val;

int evaluate () {
        int i;
        char symbol;
        for (i=0; postfix [i] != '\0'; i++) {
            symbol = postfix [i];

if (symbol >= '0' & symbol <= '9') {
        push(symbol - '0');
        }
```

Exp 14

26/10/23

Postfix Evaluation

Aim

Evaluate a postfix expression using a stack

Algorithm

Step 0: start

Step 1: Enter the expression using string format

Step 2: call the evaluate() function

Step 3: print values after evaluation

function()

Step 1: Start

Step 2: for i = 0 to i < strlen (postfix)

Symbol = postfix [i]

Step 3: Inside

if (postfix[i] >= '0' && postfix[i] <= '9')

Push postfix [i] = '0':

Step 4: else

a = pop() b = pop().

Step 5: using switch case if post fix [i] is

+ then push (b + a)

- then push (b - a)

*, then push (b * a)

/ then push (b / a)

^ push pow (b, a) & be a^x

```
int a=pop();
int b=pop();
switch(symbol){
 case '+':
        push(b+a)
        break;

 Case '-': push(b-a)
        break;

 case '*':
        push(b*a)
        break;

 case '/': push(b/a)
        break;

 Case '^'
        push(pow(b,a))
      }
    }
  }
  return pop();
}
void main(){

 printf("Enter the postfix expressions ");
 scanf("%s", postfix);
 printf("The result is %d", evaluate());
}
```

OUTPUT

Enter the postfix expression 8581-1/-+

The result is 6

Step 6: Return pop() (popped element)

RESULT

The desired output is obtained.

```
# include <stdio.h>
# define n 5
int front = -1;
int rear = -1
int q[100];
void enqueue (int item) {
        if (rear = n-1) {
            print ("queue is full" \n");
        }
    else {
        if (front == -1 && rear = -1) {
            front = 0;
            q[++rear] = item;
        }
    else {
        q[++rear] = item;
        }
    }
}

void dequeue () {
        int item;
        if (front = = -1) {
            printf ("que is empty \n");
        }
        else { item = q[front];
            printf ("The dequed element is %d", item);
            if (front = = rear) {
                front = -1;
                rear = -1;
            }
        else { front ++;
```

Exp 15
07/11/23

QUEUE

Aim: Write a program to insert (enqueue) and dequeue elements from a queue.

Algorithm

Step 0: Start

Step 1: Initialise front and rear as -1 & declare an array with max = 5

Step 2: Enter the array elements and call function (enqueue) and dequeue

Step 3: stop.

Algorithm (enqueue)

Step 0: Start

Step 1: If rear = max - 1
print "queue full"
exit

Step 2: else
if front == -1 & rear == -1
front = 0
q[++rear] = item

else
~~que~~ q[++rear] = item

end if
end if
stop

```
Void display() {
    Printf("The current queue is \n");
    for(int i= front; i<=rear; i++){
        printf("%d", q[i]);
    }
}

Void main() {
    int choice, i, n;
    while(i) {
     printf("enter the choice (n");
      scanf("%d", choice);
     switch(choice) {
    case 1:
        printf("enter elements into queue \n")
        scanf("%d", &n);
        enqueue(n);
        break;
    case 2:
        dequeue();
        breake;
    case 3:
        display();
        break();
    default: printf("wrong choice\n");
    }
  }
}
```

output

enter the choice 1

enter element into queue 20

enter the choice

enter element into queue 40

Algorithm Dequeue

Step 1: start

Step 2: If front == -1, then

print "queue is empty"

exit

end if

Step 3: item = Q[front]

Step 4: If front == rear

front = -1

rear = -1

Step 5: Else

front = front + 1

end if

Step 6: Stop

Result

The desired output is obtained

```
#include <stdio.h>
#define n 5
int front = -1;
int rear = -1;
int q[100];
void enqueue (int item){
        if (front == -1 && rear == -1) {
                front = 0;
                rear = 0;
                q [rear] = item;
        }
        else {
                if (rear+1) % n == front) {
                        printf ("queue is full \n");
                }
                else {
                        rear = (rear+1) % n;
                        q[rear] = item;
                }
        }
}

void dequeue ( ){
        int item;
        if front == -1) {
                Printf ("queue is empty \n");
                }
        else
                item = q [front];
                if (front == rear) {
                        front = -1;
```

EXP 16

07/11/23

CIRCULAR QUEUE

Aim: Write a program in C to perform enqueue and dequeue in circular queue.

Algorithm

step 0: Start

Step 1: Initilise top = -1, rear = -1 q[ ] is declared & MAX = 5

Step 2: Enter the q elements & call functions enqueue and dequeue.

step 3: STOP

Algorithm enqueue (Q, item)

step 0: start

Step 1: if front == -1

front = 0

rear = 0

q[rear] = item

step 3: Else

if ((rear+1) mod n != front)

rear = (rear+1) mod n

q[rear] = item

step 4: else

print "queue is full"

end if

end if

stop

```
        rear = -1;
    }
    else {
        front = (from+1) %n;
    }
  }
}
Void display() {
    printf(" The element in queue are ");
    if front == -1){
        printf(" queue is empty \n");
    }
    else if (front < rear) {
        for(int i = front; i<=rear; i++){
        printf("%d", q[i]);
        }
    }
    else {
        i = front
        while (i != rear+1) {
            printf("%d ", q[i]);
            i = (i+1) %n;
        }
Void main() {
    int i, choice, x;
    while (1) {
        Printf ("enter the choice 1: insert 2: delete
                3: display \n");
    scanf("%d", &choice);
```

Algorithm Dequeue (Q)

step 1: Start

step 2: if front == -1 then

print "Queue empty"

exit

step 3: else

item = Q[front]

if front == rear

front = rear = -1

else

front = (front + 1) mod n

end if

end if

step 4: STOP

```
switch(choice){
  case 1:
      printf("enter element\n")
      scanf("%d", &x)
      enqueue(x);
      printf("\n");
      break;

  case 2:
      dequeue();
      break;

  case 3:
      display()
      break;
  default:
      printf("Wrong choice");
  }
}
}
```

output

enter the choice
1: insert 2: delete 3. display

1

enter element 20

enter the choice
1: insert 2: delete
3: display

1

enter element 30

enter the choice
1: insert 2: delete
3: display

2

deleted element is 20