**3a.Queue Implementation**

#include<stdio.h>

#define MAX 50

int queue\_array[MAX];

int rear=-1;

int front=-1;

display()

{

int i;

if(front==-1)

printf("queue is empty\n");

else

{

printf("queue is :\n");

for(i=front;i<=rear;i++)

printf("%d",queue\_array[i]);

printf("\n");

}

}

main()

{

int choice;

while(1)

{

printf("1.insert\n");

printf("2.delete\n");

printf("3.display\n");

printf("4.exit\n");

printf("enter your choice:");

scanf("%d",&choice);

switch(choice)

{

case 1:

insert();

break;

case 2:

delete();

break;

case 3:

display();

break;

case 4:

exit(1);

break;

default:

printf("invalid choice\n");

}

}

}

insert()

{

int add\_item;

if(rear==MAX-1)

printf("queue overflow\n");

else

{

if(front==-1)

front=0;

printf("insert the element in the queue:");

scanf("%d",&add\_item);

rear+=1;

queue\_array[rear]=add\_item;

}

}

delete()

{

if(front==-1 || front>rear)

{

printf("queue underflow\n");

return;

}

else

{

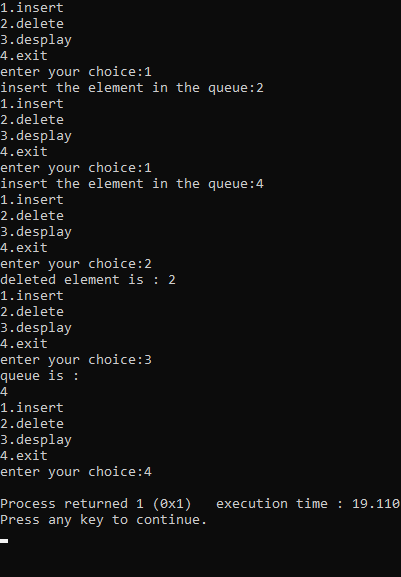
printf("deleted element is : %d\n",queue\_array[front]);

front+=1;

}

}

**output**



**Infix to Postfix conversion**

#include <stdio.h>

#include<ctype.h>

#define SIZE 50

char stack[SIZE];

int top=-1;

push(char ele)

{

stack[++top]=ele;

}

char pop()

{

return(stack[top--]);

}

int pr(char symbol)

{

if (symbol == '^')

{

return(3);

}

else if (symbol == '\*' || symbol == '/')

{

return(2);

}

else if (symbol == '+' || symbol == '-')

{

return(1);

}

else

{

return (0);

}

}

void main()

{

char infix[50],postfix[50],ch,ele;

int i=0,k=0;

printf("enter the infix expression:");

scanf("%s",infix);

push('#');

while( (ch=infix[i++]) != '\0')

{

if(ch=='(') push(ch);

else

if(isalnum(ch)) postfix[k++]=ch;

else

if(ch ==')')

{

while(stack[top]!='(')

postfix[k++]=pop();

ele=pop();

}

else

{

while(pr(stack[top])>=pr(ch))

postfix[k++]=pop();

push(ch);

}

}

while(stack[top]!='#')

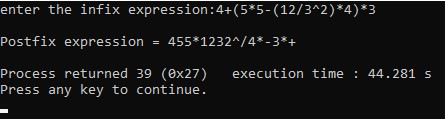
postfix[k++]=pop();

postfix[k]='\0';

printf("\nPostfix expression = %s\n",postfix);

}

**output**



**Evaluation of Postfix Expression**

#include<stdio.h>

#include <ctype.h>

int stack[20];

int top=-1;

void push(int x)

{

stack[++top]=x;

}

int pop()

{

return stack[top--];

}

int main()

{

char exp[20];

char \*e;

int n1,n2,n3,num;

printf("enter the expresion :: ");

scanf("%s",exp);

e=exp;

while(\*e!= '\0')

{

if(isdigit(\*e))

{

num=\*e-48;

push(num);

}

else

{

n1=pop();

n2=pop();

switch(\*e)

{

case '+':

{

n3=n1+n2;

break;

}

case '-':

{

n3=n1-n2;

break;

}

case '\*':

{

n3=n1\*n2;

break;

}

case '/':

{

n3=n1/n2;

break;

}

}

push(n3);

}

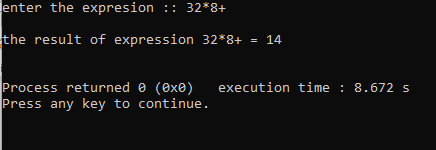
e++;

}

printf("\nthe result of expression %s = %d\n\n",exp,pop());

}

**output**



**3b. Circular Queue Implementation**

#include<stdio.h>

#define SIZE 5

int items[SIZE];

int front = -1, rear = -1;

int isFull() {

if ((front == rear + 1) || (front == 0 && rear == SIZE - 1))

return 1;

return 0;

}

int isEmpty() {

if (front == -1)

return 1;

return 0;

}

void enQueue(int element) {

if (isFull())

printf("\nQueue is full");

else {

if (front == -1)

front = 0;

rear = (rear + 1) % SIZE;

items[rear] = element;

printf("\nInserted -> %d", element);

}

}

int deQueue() {

int element;

if (isEmpty()) {

printf("\nQueue is empty");

return -1;

} else {

element = items[front];

if (front == rear) {

front = -1;

rear = -1;

} else {

front = (front + 1) % SIZE;

}

printf("\nDeleted element -> %d\n", element);

return element;

}

}

void display() {

int i;

if (isEmpty())

printf("\nEmpty queue\n");

else {

printf("\nFront -> %d", front);

printf("\nItems -> ");

for (i = front; i != rear; i = (i + 1) % SIZE) {

printf("%d ", items[i]);

}

printf("%d", items[i]);

printf("\nRear -> %d\n", rear);

}

}

int main() {

enQueue(1);

enQueue(2);

enQueue(3);

enQueue(4);

enQueue(5);

display();

deQueue();

deQueue();

display();

enQueue(6);

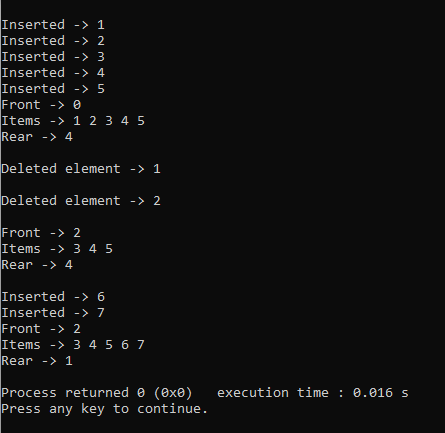
enQueue(7);

display();

return 0;

}

**Output**

****