**Task No. 1:** Write a program to build your own stack class. The minimum your stack class should include is:

A Push(Object) method

A Pop() method

A Peek() method

A IsFull() method

A IsEmpty() method

A Display() method

A Count() method

**Solution:**

class StackA

{

private int[] stackArray;

private int top;

public StackA()

{

stackArray = new int[10];

top = -1;

}

public StackA(int maxSize)

{

stackArray = new int[maxSize];

top = -1;

}

public int Size()

{

return top + 1;

}

public bool IsEmpty()

{

return (top == -1);

}

public bool IsFull()

{

return (top == stackArray.Length -1);

}

public void Push(int x)

{

if (IsFull())

{

Console.WriteLine("Stack overflow");

return;

}

top = top + 1;

stackArray[top] = x;

}

public int Pop()

{

int x;

if (IsEmpty())

throw new System.InvalidOperationException("Stack underflow");

x = stackArray[top];

top = top - 1;

return x;

}

public int Peek()

{

if (IsEmpty())

throw new System.InvalidOperationException("Stack underflow");

return stackArray[top];

}

public void Display()

{

if (IsEmpty())

{

Console.WriteLine("Stack is empty");

return;

}

Console.WriteLine("Stack is: ");

for (int i = top; i >= 0; i--)

Console.WriteLine(stackArray[i] + " ");

Console.WriteLine();

}

}

}

class Program

{

static void Main(string[] args)

{

int choice, x;

StackA st = new StackA(8);

while (true)

{

Console.WriteLine("1.Push an element on the stack");

Console.WriteLine("2.Pop an element from the stack");

Console.WriteLine("3.Display the top element");

Console.WriteLine("4.Display all stack element");

Console.WriteLine("5.Display size of Stack");

Console.WriteLine("6.Quit");

Console.Write("Enter your choice: ");

choice = Convert.ToInt32(Console.ReadLine());

if (choice == 6)

break;

switch (choice)

{

case 1:

Console.Write("Enter the element to be pushed: ");

x = Convert.ToInt32(Console.ReadLine());

st.Push(x);

break;

case 2:

x = st.Pop();

Console.WriteLine("Popped element is: " + x);

break;

case 3:

Console.WriteLine("Element at the top is: " + st.Peek());

break;

case 4:

st.Display();

break;

case 5:

Console.WriteLine("Size of stack" + st.Size());

break;

default:

Console.WriteLine("Wrong choice");

break;

}

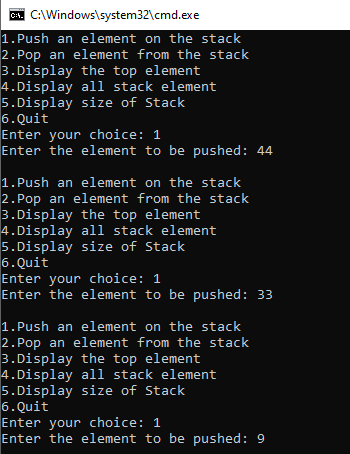
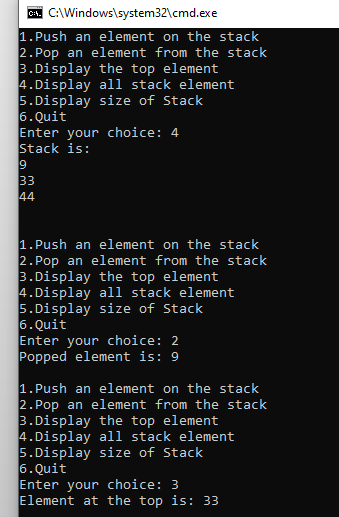
Console.WriteLine("");

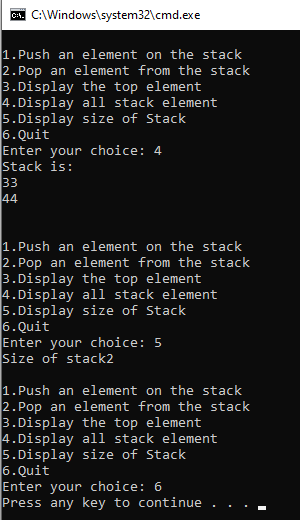
}

}

}

}

**Output:**



**Task No. 2:** Using Stack class write a code for max and min.

**Solution:**

public int min()

{

int temp = this.top;

int min = this.list[this.top];

while (!IsEmpty())

{

if (min > this.Peek())

min = this.Peek();

this.top--;

}

this.top = temp;

return min;

}

public int max()

{

int temp = this.top;

int max = this.list[this.top];

while(!IsEmpty())

{

if (max < this.Peek())

max = this.Peek();

this.top--;

}

this.top = temp;

return max;

}

**TestClass:**

Stack obj = new Stack(5);

obj.Push(5);

obj.Push(8);

obj.Push(13);

obj.Push(3);

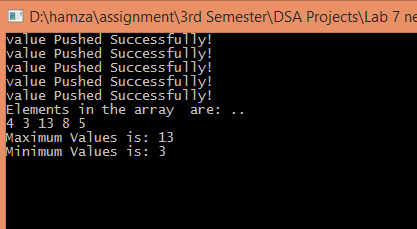
obj.Push(4);

obj.Display();

Console.WriteLine("Maximum Values is: "+obj.max());

Console.WriteLine("Minimum Values is: " + obj.min());

**Output:**



**Task No. 3:** Write a Windows application that provides a text box for the user to enter an expression with parenthesis. Provide a Check Parenthesis button that, when clicked, runs a program that checks the number of parentheses in the expressions. Also tells either the expression is correct or not.

**Solution:**

private void Button1\_Click(object sender, EventArgs e)

{

string equation = Input.Text;

Stack obj = new Stack(equation.Length);

foreach (char c in equation)

obj.Push(c);

int i = 0;

int openP = 0;

int closeP = 0;

while (i < equation.Length)

{

if (obj.Peek() == "(")

openP++;

else if (obj.Peek() == ")")

closeP++;

i++;

obj.top--;

}

open.Text = "" + openP;

close.Text = "" + closeP;

if (openP == closeP)

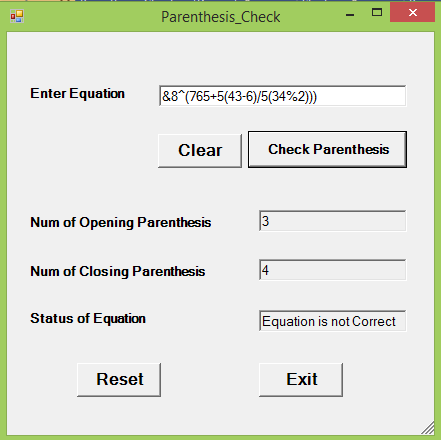
status.Text = "Equation is Correct";

else

status.Text = "Equation is not Correct";

}

**Output:**



**Task No. 4:** Take a String as input and print its reverse using your Stack class

**Solution:**

All Code is same as Stack Class, just way of use is different:

**Use:**

Console.Write("Enter Sting: ");

string value = Console.ReadLine();

char[] array = value.ToCharArray();

Stack obj = new Stack(array.Length);

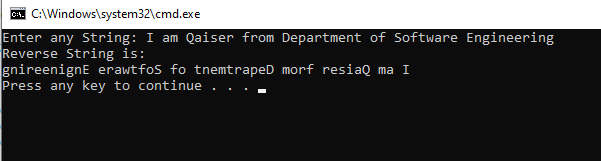
foreach (char s in array)

obj.Push(s);

Console.WriteLine("Reverse String is: ");

obj.Display();

**Output:**



**Task No. 5:** Design & implement all methods of Simple Queue

**Solution:**

class QueueA

{

private int[] queueArray;

private int front;

private int rear;

public QueueA()

{

queueArray = new int[10];

front = -1;

rear = -1;

}

public QueueA(int maxSize)

{

queueArray = new int[maxSize];

front = -1;

rear = -1;

}

public bool IsEmpty()

{

return (front == -1 || front == rear + 1);

}

public bool IsFull()

{

return (rear== queueArray.Length -1);

}

public int Size()

{

if (IsEmpty())

return 0;

else

return rear - front + 1;

}

public void Insert(int x)

{

if (IsFull())

{

Console.WriteLine("Queue overflow \n");

return;

}

if (front == -1)

front = 0;

rear = rear + 1;

queueArray[rear] = x;

}

public int Delete()

{

int x;

if (IsEmpty())

throw new System.InvalidOperationException("Queue underflow");

x = queueArray[front];

front = front+1;

return x;

}

public int Peek()

{

if (IsEmpty())

throw new System.InvalidOperationException("Queue underflow");

return queueArray[front];

}

public void Display()

{

if (IsEmpty())

{

Console.WriteLine("Queue is empty\n");

return;

}

Console.WriteLine("Queue is :\n\n");

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

Console.WriteLine(queueArray[i] + " ");

Console.WriteLine();

}

}

}

class Program

{

static void Main(string[] args)

{

int choice, x;

QueueA qu = new QueueA(8);

while (true)

{

Console.WriteLine("1.Insert an element in the queue");

Console.WriteLine("2.Delete an element from the queue");

Console.WriteLine("3.Display element at the front");

Console.WriteLine("4.Display all elements of queue");

Console.WriteLine("5.Display size of queue");

Console.WriteLine("6.Quit");

Console.Write("Enter your choice: ");

choice = Convert.ToInt32(Console.ReadLine());

if (choice ==6)

break;

switch (choice)

{

case 1:

Console.Write("Enter the element to be inserted: ");

x = Convert.ToInt32(Console.ReadLine());

qu.Insert(x);

break;

case 2:

x = qu.Delete();

Console.WriteLine("element deleted is: " + x);

break;

case 3:

Console.WriteLine("Element at the front is: " + qu.Peek());

break;

case 4:

qu.Display();

break;

case 5:

Console.WriteLine("Size of queue" + qu.Size());

break;

default:

Console.WriteLine("Wrong choice");

break;

}

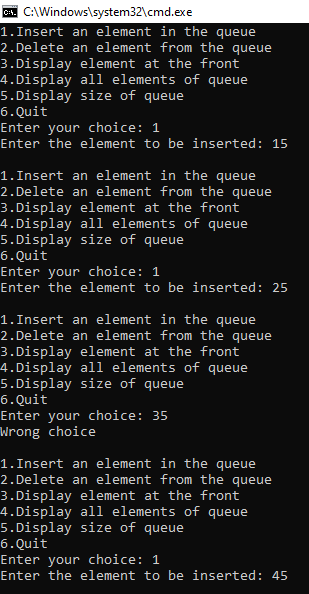
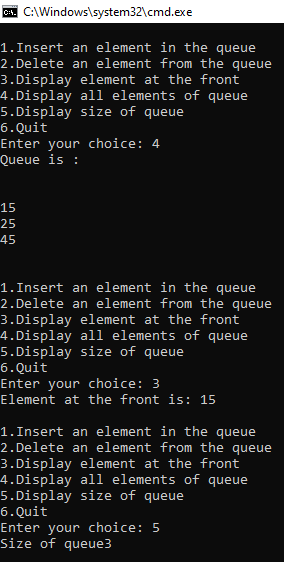
Console.WriteLine("");

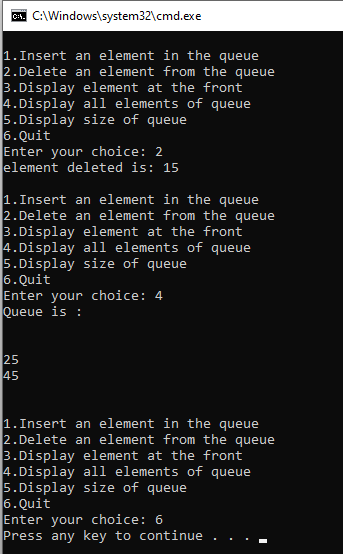
}

}

}

}

**Output:**



**Task No. 6:** Design & implement all methods of Circular Queue.

**Solution:**

class CircularQueue

{

private int[] queueArray;

private int front;

private int rear;

public CircularQueue()

{

queueArray = new int[10];

front = -1;

rear = -1;

}

public CircularQueue(int maxSize)

{

queueArray = new int[maxSize];

front = -1;

rear = -1;

}

public bool IsEmpty()

{

return (front == -1 );

}

public bool IsFull()

{

return ((front == 0 && rear ==queueArray.Length-1)||(front == rear+1));

}

public void Insert(int x)

{

if (IsFull())

{

Console.WriteLine("Queue overflow \n");

return;

}

if (front == -1)

front = 0;

if (rear == queueArray.Length - 1)

rear = 0;

else

rear = rear + 1;

queueArray[rear] = x;

}

public int Delete()

{

if (IsEmpty())

throw new System.InvalidOperationException("Queue underflow");

int x = queueArray[front];

if (front == rear) // queue has only one element

{

front = -1;

rear = -1;

}

else if (front == queueArray.Length - 1)

front = 0;

else

front = front + 1;

return x;

}

public int Peek()

{

if (IsEmpty())

throw new System.InvalidOperationException("Queue underflow");

return queueArray[front];

}

public void Display()

{

if (IsEmpty())

{

Console.WriteLine("Queue is empty\n");

return;

}

Console.WriteLine("Queue is :");

int i = front;

if (front <= rear)

{

while (i <= rear)

Console.Write(queueArray[i++] + " ");

}

else

{

while (i <= queueArray.Length-1)

Console.Write(queueArray[i++] + " ");

i = 0;

while (i <= rear)

Console.Write(queueArray[i++] + " ");

}

Console.WriteLine();

}

public int Size()

{

if (IsEmpty())

return 0;

if (IsFull())

return queueArray.Length;

int i = front;

int sz = 0;

if (front <= rear)

while (i <= rear)

{

sz++;

i++;

}

else

{

while (i<=queueArray.Length-1)

{

sz++;

i++;

}

i = 0;

while (i<=rear)

{

sz++;

i++;

}

}

return sz;

}

}

}

class Program

{

static void Main(string[] args)

{

int choice, x;

CircularQueue qu = new CircularQueue(8);

while (true)

{

Console.WriteLine("1.Insert an element in the Circular queue");

Console.WriteLine("2.Delete an element from the Circular queue");

Console.WriteLine("3.Display element at the front");

Console.WriteLine("4.Display all elements of the Circular queue");

Console.WriteLine("5.Display size of Circular queue");

Console.WriteLine("6.Quit");

Console.Write("Enter your choice: ");

choice = Convert.ToInt32(Console.ReadLine());

if (choice ==6)

break;

switch (choice)

{

case 1:

Console.Write("Enter the element to be inserted: ");

x = Convert.ToInt32(Console.ReadLine());

qu.Insert(x);

break;

case 2:

x = qu.Delete();

Console.WriteLine("Element deleted is: " + x);

break;

case 3:

Console.WriteLine("Element at the front is: " + qu.Peek());

break;

case 4:

qu.Display();

break;

case 5:

Console.WriteLine("Size of queue: " + qu.Size());

break;

default:

Console.WriteLine("Wrong choice");

break;

}

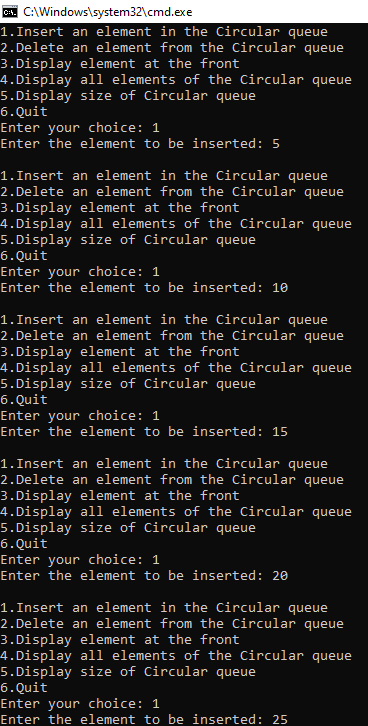
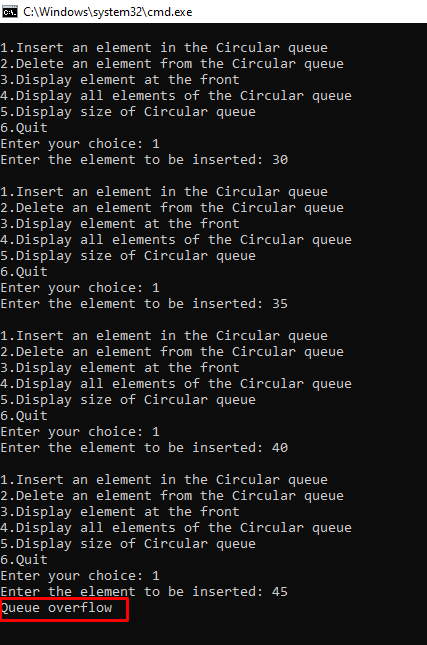
Console.WriteLine("");

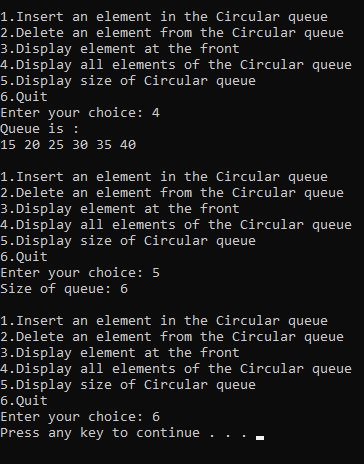
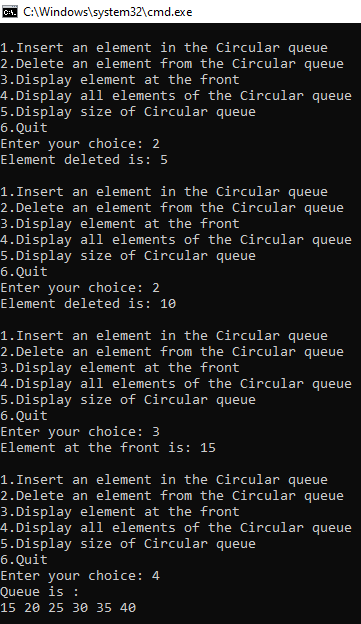
}

}

}

}

**Output:**



**Task No. 7:** Design and implement for Priority Queue.

Method 1: Ordering in/ after Enqueue method

Method 2: Separate queues for different priorities

**Solution:**

class Node

{

public int priority;

public int info;

public Node link;

public Node(int i, int pr)

{

info = i;

priority = pr;

link = null;

}

}

}

class PriorityQueueL

{

private Node front;

public PriorityQueueL()

{

front = null;

}

public void Insert(int element, int elementPriority)

{

Node temp, p;

temp = new Node(element, elementPriority);

/\* Queue is empty or element to be added has priority more than first element \*/

if (IsEmpty() || elementPriority < front.priority)

{

temp.link = front;

front = temp;

}

else

{

p = front;

while (p.link != null && p.link.priority <= elementPriority)

p = p.link;

temp.link = p.link;

p.link = temp;

}

}

public int Delete()

{

int element;

if ( IsEmpty() )

throw new System.InvalidOperationException("Queue Underflow");

else

{

element = front.info;

front = front.link;

}

return element;

}

public bool IsEmpty()

{

return (front==null);

}

public void Display()

{

Node p=front;

if ( IsEmpty())

Console.WriteLine("Queue is empty\n");

else

{

Console.WriteLine("Queue is :");

Console.WriteLine("Element Priority");

while (p!=null)

{

Console.WriteLine(p.info + " " + p.priority);

p=p.link;

}

}

Console.WriteLine("");

}

}

}

class Program

{

static void Main(string[] args)

{

int choice, element, elementPriority;

PriorityQueueL pq = new PriorityQueueL();

while (true)

{

Console.WriteLine("1.Insert a new element");

Console.WriteLine("2.Delete an element");

Console.WriteLine("3.Display the queue");

Console.WriteLine("4.Quit");

Console.Write("Enter your choice : ");

choice = Convert.ToInt32(Console.ReadLine());

if (choice ==4)

break;

switch (choice)

{

case 1:

Console.WriteLine("Enter the element to be inserted: ");

element = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Enter its priority : ");

elementPriority = Convert.ToInt32(Console.ReadLine());

pq.Insert(element,elementPriority);

break;

case 2:

Console.WriteLine("Deleted element is: " + pq.Delete());

break;

case 3:

pq.Display();

break;

default:

Console.WriteLine("Wrong choice");

break;

}

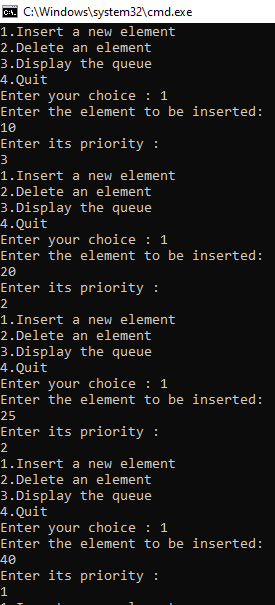
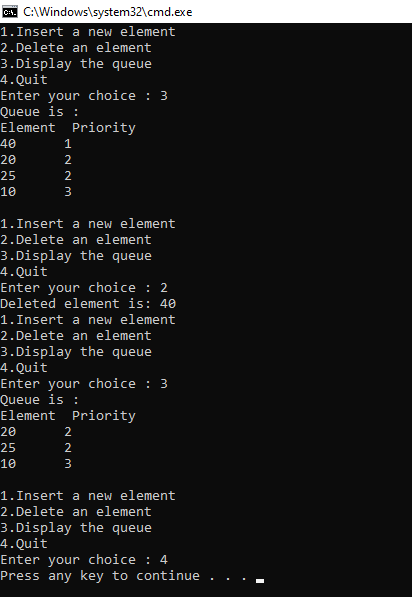
}

}

}

}

**Output:**



**Task No. 8:** With the help of Stacks, implement Polish notation in which you have to convert given expression to postfix notation.

**Solution:**

**Stack of Integer:**

class StackInt

{

private int[] stackArray;

private int top;

public StackInt()

{

stackArray = new int[10];

top = -1;

}

public StackInt(int maxSize)

{

stackArray = new int[maxSize];

top = -1;

}

public int Size()

{

return top + 1;

}

public bool IsEmpty()

{

return (top == -1);

}

public bool IsFull()

{

return (top == stackArray.Length -1);

}

public void Push(int x)

{

if (IsFull())

{

Console.WriteLine("Stack overflow");

return;

}

top = top + 1;

stackArray[top] = x;

}

public int Pop()

{

int x;

if (IsEmpty())

throw new System.InvalidOperationException("Stack underflow");

x = stackArray[top];

top = top - 1;

return x;

}

public int Peek()

{

if (IsEmpty())

throw new System.InvalidOperationException("Stack underflow");

return stackArray[top];

}

}

}

**Stack of Character:**

class StackChar

{

private char[] stackArray;

private int top;

public StackChar()

{

stackArray = new char[10];

top = -1;

}

public StackChar(int maxSize)

{

stackArray = new char[maxSize];

top = -1;

}

public int size()

{

return top + 1;

}

public bool IsEmpty()

{

return (top==-1);

}

public bool IsFull()

{

return (top == stackArray.Length - 1);

}

public void Push(char x)

{

if (IsFull())

{

Console.WriteLine("Stack Overflow\n");

return;

}

top = top + 1;

stackArray[top] = x;

}

public char Pop()

{

char x;

if (IsEmpty())

{

Console.WriteLine("Stack underflow\n");

throw new System.InvalidOperationException();

}

x = stackArray[top];

top = top - 1;

return x;

}

public char Peek()

{

if (IsEmpty())

throw new System.InvalidOperationException("Stack underflow");

return stackArray[top];

}

}

}

**Main**

class Program

{

static void Main(string[] args)

{

string infix;

Console.Write("Enter infix expression :");

infix = Console.ReadLine();

String postfix = infixToPostfix(infix);

Console.WriteLine("postfix expression is : " + postfix);

Console.WriteLine("Value of Expression: " + evaluatePostfix(postfix));

}

public static String infixToPostfix(String infix)

{

String postfix = "";

StackChar st = new StackChar(20);

Char next, symbol;

for (int i = 0; i < infix.Length; i++)

{

symbol = infix[i];

if (symbol == ' ' || symbol == '\t') //ignore blank and tabs

continue;

switch (symbol)

{

case '(':

st.Push(symbol);

break;

case ')':

while ((next = st.Pop()) != '(')

postfix = postfix + next;

break;

case '+':

case '-':

case '\*':

case '/':

case '%':

case '^':

while (!st.IsEmpty() && Precedence(st.Peek()) >= Precedence(symbol))

postfix = postfix + st.Pop();

st.Push(symbol);

break;

default: //operands

postfix = postfix + symbol;

break;

}

}

while (!st.IsEmpty())

postfix = postfix + st.Pop();

return postfix;

}

public static int Precedence(char symbol)

{

switch (symbol)

{

case '(':

return 0;

case '+':

case '-':

return 1;

case '\*':

case '/':

case '%':

return 2;

case '^':

return 3;

default:

return 0;

}

}

public static int evaluatePostfix(String postfix)

{

StackInt st = new StackInt(20);

int x, y;

for (int i = 0; i < postfix.Length; i++)

{

if (Char.IsDigit(postfix[i]))

st.Push(Convert.ToInt32(char.GetNumericValue(postfix[i])));

else

{

x = st.Pop();

y = st.Pop();

switch (postfix[i])

{

case '+':

st.Push(y + x); break;

case '-':

st.Push(y - x); break;

case '\*':

st.Push(y \* x); break;

case '/':

st.Push(y / x); break;

case '%':

st.Push(y % x); break;

case '^':

st.Push(power(x, y));

break;

}

}

}

return st.Pop();

}

public static int power(int b, int a)

{

int i, x = 1;

for (i = 1; i <= a; i++)

x = x \* b;

return x;

}

}

}

**Output:**

