**Task 7: Design and implement for Priority Queue.**

**Code:**

**PriortyQueue Class:**

class PriortyQueue

{

int[,] array;

int Length, front, rear;

public PriortyQueue(int length)

{

Length =length;

array = new int[length, 2];

front = 0;

rear = 0;

}

public bool SortHigh(int value, int x)

{

int temp = rear-1;

if (!UnderFlow())

{

for (int i = rear-1; i >=0; i--)

{

if(array[i,0]==2 || array[i, 0] == 3) {

array[i + 1, 0] = array[i, 0];

array[i + 1, 1] = array[i, 1];

}

else

{

temp = i;

break;

}

}

//while ((array[temp, 0] == 2 || array[temp, 0] == 3) && temp >= 0)

//{

// if (temp <= rear && temp > 0)

// {

// array[temp+1, 0] = array[temp , 0];

// array[temp+1, 1] = array[temp , 1];

// temp--;

// }

// else

// {

// break;

// }

//}

if (Enqueue(value, x, temp))

{

return true;

}

else return false;

}

else

return Enqueue(value, x);

}

public int Dequeue()

{

if (!UnderFlow())

{

int val = array[front, 1];

for (int i = front; i < rear-1; i++)

{

array[i, 0] = array[i + 1, 0];

array[i, 1] = array[i + 1, 1];

}

rear--;

return val;

}

return -1;

}

public int Front()

{

if (!UnderFlow())

{

return array[front, 1];

}

return -1;

}

public bool Enqueue(int val, int j)

{

if (!Overflow())

{

array[rear, 1] = val;

array[rear, 0] = j;

rear++;

return true;

}

return false;

}

public bool Enqueue(int val, int j,int indx)

{

if (!Overflow())

{

if (indx >= 0)

{

array[indx, 1] = val;

array[indx, 0] = j;

rear++;

return true;

}

else

{

array[0, 1] = val;

array[0, 0] = j;

rear++;

return true;

}

}

return false;

}

public bool Overflow()

{

if (rear<Length)

return false;

return true;

}

public bool SortMedium(int value, int x)

{

int temp = rear;

if (!UnderFlow())

{

while (array[temp, 0] == 3 && temp >= 0)

{

if (temp < 0)

{

break;

}

else

{

if (temp < Length - 1)

{

array[temp + 1, 0] = array[temp, 0];

array[temp + 1, 1] = array[temp, 1];

temp--;

}

else

{

break;

}

}

}

if (temp >= 0 && temp < Length)

{

if (Enqueue(value, x, temp ))

{

return true;

}

else

{

return false;

}

}

}

else

return Enqueue(value, x);

return false;

}

public bool UnderFlow()

{

if (rear==front)

return true;

return false;

}

public void Display()

{

Console.WriteLine("Your array");

Console.WriteLine("Value\tPriorty");

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

{

Console.WriteLine(array[i , 1] + "\t" + array[i, 0]);

}

Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

}

}

**Program Class:**

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Enter length of Queue");

int length = Convert.ToInt32(Console.ReadLine());

PriortyQueue obj = new PriortyQueue(length);

while (true)

{

Console.WriteLine("You want to perform any function on Queue? Press 0 to exit");

Console.WriteLine("1- Enqueue\n2- Dequeue\n3- Front");

int answer = Convert.ToInt32(Console.ReadLine());

if (answer == 1)

{

int priorty, tempvalue;

Console.WriteLine("Enter priorty");

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

Console.WriteLine("Enter value");

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

if (priorty == 1)

{

if (obj.SortHigh(tempvalue, priorty))

{

Console.WriteLine("Value Enqueued");

obj.Display();

}

else

{

Console.WriteLine("Queue Overflow");

}

}

else if (priorty == 2)

{

if (obj.SortMedium(tempvalue, priorty))

{

Console.WriteLine("Value Enqueued");

obj.Display();

}

else

{

Console.WriteLine("Queue Overflow");

}

}

else if (priorty == 3)

{

if (obj.Enqueue(tempvalue, priorty))

{

Console.WriteLine("Value Enqueued");

obj.Display();

}

else

{

Console.WriteLine("Queue Overflow");

}

}

else

{

Console.WriteLine("Invalid priorty");

}

}

else if (answer == 2)

{

int val = obj.Dequeue();

if (val >= 0)

{

Console.WriteLine("Value successfully dequeued");

Console.WriteLine("Dequeued value = " + val);

}

else

{

Console.WriteLine("Stack Underflow");

}

}

else if (answer == 3)

{

int val = obj.Front();

if (val >= 0)

{

Console.WriteLine("Front value = " + val);

}

else

{

Console.WriteLine("Stack empty");

}

}

else

{

break;

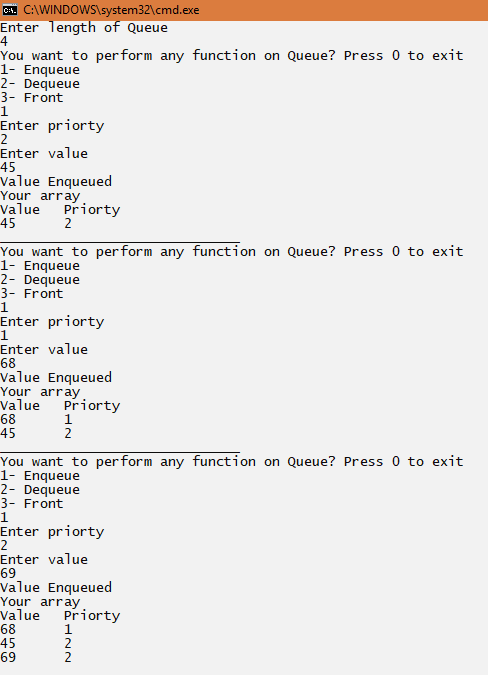
}

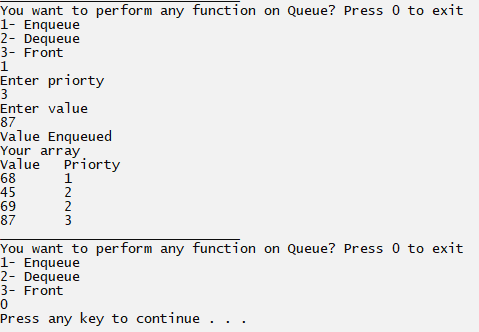
}

}

}

**Output:**





**Task 3: With the help of Stacks, implement Polish notation in which you have to convert given expression to postfix notation.**

**Code:  
Program Class:**

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Enter expression");

string infix = Console.ReadLine();

infix = infix.Trim();

if (ValidExpression(infix))

{

string postfix = IntoPost(infix);

Console.WriteLine("Postfix = "+postfix);

}

}

public static string IntoPost(string infix)

{

Stack obj = new Stack();

char[] arr = infix.ToCharArray();

char[] postarr = new char[arr.Length];

int count = 0;

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

{

int ascii = (int)arr[i];

if (ascii >= 48 && ascii <= 57)

{

postarr[count] = arr[i];

count++;

}

else if ((ascii >= 65 && ascii <= 90) || (ascii >= 97 && ascii <= 122))

{

postarr[count] = arr[i];

count++;

}

else if (arr[i] == '(' || arr[i] == '^')

{

obj.Push(arr[i]);

}

else if (arr[i] == '\*' || arr[i] == '/' || arr[i] == '%')

{

char temp = obj.Peek();

if (temp == '(' || temp == '-' || temp == '+')

{

obj.Push(arr[i]);

}

else if (temp == '\*' || temp == '/' || temp == '%')

{

postarr[count] = obj.Pop();

count++;

obj.Push(arr[i]);

}

else if (temp == '^')

{

postarr[count] = obj.Pop();

count++;

char temp1 = obj.Peek();

if (temp1 == '\*' || temp1 == '/' || temp1 == '%')

{

postarr[count] = obj.Pop();

count++;

obj.Push(arr[i]);

}

else

{

obj.Push(arr[i]);

}

}

else

{

obj.Push(arr[i]);

}

}

else if (arr[i] == '+' || arr[i] == '-')

{

char temp = obj.Peek();

if (temp == '^')

{

postarr[count] = obj.Pop();

count++;

char temp1 = obj.Peek();

if (temp1 == '\*' || temp1 == '/' || temp1 == '%')

{

postarr[count] = obj.Pop();

count++;

}

char temp2 = obj.Peek();

if (temp2 == '+' || temp2 == '-')

{

postarr[count] = obj.Pop();

count++;

obj.Push(arr[i]);

}

else

{

obj.Push(arr[i]);

}

}

else if (temp == '\*' || temp == '/' || temp == '%')

{

postarr[count] = obj.Pop();

count++;

char temp1 = obj.Peek();

if (temp1 == '+' || temp1 == '-')

{

postarr[count] = obj.Pop();

count++;

obj.Push(arr[i]);

}

else

{

obj.Push(arr[i]);

}

}

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

{

postarr[count] = obj.Pop();

count++;

obj.Push(arr[i]);

}

else

{

obj.Push(arr[i]);

}

}

else if (arr[i] == ')')

{

char temp = obj.Peek();

while (temp != '(')

{

postarr[count] = obj.Pop();

count++;

temp = obj.Peek();

}

obj.Pop();

}

else

{

}

}

while (!obj.Underflow())

{

postarr[count] = obj.Pop();

count++;

}

string post = "";

for (int i = 0; i < count; i++)

{

post += postarr[i];

}

return post;

}

private static bool ValidExpression(string value)

{

char[] array = value.ToCharArray();

int left = 0, right = 0;

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

{

if (array[i] == '(')

{

left++;

}

if (array[i] == ')')

{

right++;

}

}

if (left == right)

{

return true;

}

return false;

}

}

**Stack Class:**

class Stack

{

Node start;

public Stack()

{

start = new Node();

}

public Stack(Node obj)

{

start = obj;

}

public bool Underflow()

{

if (start.next != null)

{

return false;

}

else

{

return true;

}

}

public bool Push(char val)

{

Node n = new Node(val);

if (!Underflow())

{

n.next = start.next;

start.next = n;

return true;

}

else

{

start.next = n;

return true;

}

}

public char Peek()

{

if (!Underflow())

{

char val;

val = start.next.data;

return val;

}

return ' ';

}

public char Pop()

{

if (!Underflow())

{

char val;

if (start.next.next != null)

{

val = start.next.data;

start.next = start.next.next;

return val;

}

else

{

val = start.next.data;

start.next = null;

return val;

}

}

return ' ' ;

}

public void Display()

{

if (!Underflow())

{

Node temp = start.next;

Console.WriteLine("Your list");

while (temp.next != null)

{

Console.Write(temp.data + " ");

temp = temp.next;

}

Console.Write(temp.data);

Console.WriteLine();

}

else

{

Console.WriteLine("List empty");

}

}

**Node Class:**

class Node

{

internal Node next;

internal char data;

public Node()

{

next = null;

}

public Node(char val)

{

this.data = val;

}

public Node(Node obj)

{

this.next = obj;

}

}

**Output:**

