**Bahria University**

**Software Engineering Department**



**Course: CSC-221 DATA STRUCTURES & ALGORITHMS**

**Term: FALL 2019, Class: BSE 3(B)**

**Assignment No:**

|  |  |
| --- | --- |
| **0** | **2** |

**Submitted By:**

**(Name) Qaiser Abbas (Reg. No.) 57245**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Submission Date** | | | | | | | |  |  |  |  |  |
| **2** | **6** | **/** | **1** | **2** | **/** | **1** | **9** |  |  |  |  |

**(Date: DD/MM/YY)**

**Submitted To:**

**Engr. Saniya Shaikh**

**Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Max Marks: \_\_\_\_\_\_\_\_\_\_\_ Marks Obtained: \_\_\_\_\_\_\_\_\_\_\_\_\_**

**Task No. 1:** Using dynamic Stack class write a code which takes n number of inputs from user and create sorted list. I.e. in ascending order.

**Solution:**

class Node

{

public int info;

public Node link;

public Node(int i)

{

info = i;

link = null;

}

}

}

class DynamicStack

{

private Node start;

public DynamicStack()

{

start = null;

}

public void DisplayStackList()

{

Node p;

if (start == null)

{

Console.WriteLine("Stack is Empty");

return;

}

Console.Write("Original Stack elements are : ");

p = start;

while (p != null)

{

Console.Write(p.info + " ");

p = p.link;

}

Console.WriteLine();

}

public void CountElements()

{

int n = 0;

Node p = start;

while (p != null)

{

n++;

p = p.link;

}

Console.WriteLine("Number of elements in the stack = " + n);

}

public bool Search(int x)

{

int position = 1;

Node p = start;

while (p != null)

{

if (p.info == x)

break;

position++;

p = p.link;

}

if (p == null)

{

Console.WriteLine(x + "not found in stack");

return false;

}

else

{

Console.WriteLine(x + "is at position " + position);

return true;

}

}

public void InsertElement(int data)

{

Node p;

Node temp = new Node(data);

if (start == null)

{

start = temp;

return;

}

p = start;

while (p.link != null)

p = p.link;

p.link = temp;

}

public void CreateList()

{

int i, n, data;

Console.Write("Enter the number of elements for dynamic stack : ");

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

if (n == 0)

return;

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

{

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

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

InsertElement(data);

}

}

public void DeleteElement()

{

if (start == null)

return;

if (start.link == null)

{

start = null;

return;

}

Node p = start;

while (p.link.link != null)

p = p.link;

p.link = null;

}

}

public void Sort()

{

Node end, p, q;

for (end = null; end != top.link; end = p)

{

for (p = top; p.link != end; p = p.link)

{

q = p.link;

if (p.info > q.info)

{

int temp = p.info;

p.info = q.info;

q.info = temp;

}

}

} Display();

}

}

}

}

class Program

{

static void Main(string[] args)

{

int choice, data;

DynamicStack stackQQ = new DynamicStack();

stackQQ.CreateList();

while (true)

{

Console.WriteLine("1.Display Stack");

Console.WriteLine("2.Count the number of element in stack");

Console.WriteLine("3.search for an element");

Console.WriteLine("4.Insert an element in the stack");

Console.WriteLine("5.Delete element from stack");

Console.WriteLine("6.Quit");

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

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

if (choice == 6)

break;

switch (choice)

{

case 1:

stackQQ.DisplayStackList();

break;

case 2:

stackQQ.CountElements();

break;

case 3:

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

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

stackQQ.Search(data);

break;

case 4:

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

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

stackQQ.InsertElement(data);

break;

case 5:

stackQQ.DeleteElement();

break;

default:

Console.WriteLine("Wrong choice");

break;

}

Console.WriteLine();

}

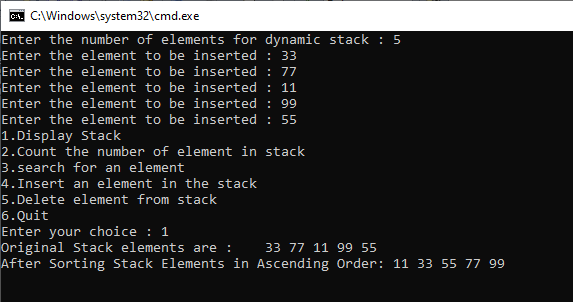
Console.WriteLine("Exiting....");

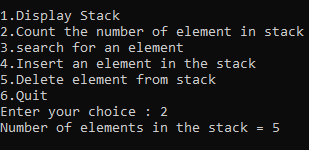
}

}

}

**Output:**





**Task No. 2:** Using dynamic queue class write a code which takes n number of inputs from user and create sorted list in descending order.

**Solution:**

class Node

{

public int info;

public Node link;

public Node(int i)

{

info = i;

link = null;

}

}

}

class DynamicQueue

{

private Node start;

public DynamicQueue()

{

start = null;

}

public void DisplayQueue()

{

Node p;

if (start == null)

{

Console.WriteLine("Queue is Empty");

return;

}

Console.Write("Queue is : ");

p = start;

while (p != null)

{

Console.Write(p.info + " ");

p = p.link;

}

Console.WriteLine();

Console.WriteLine();

Console.WriteLine();

}

public void CountElements()

{

int n = 0;

Node p = start;

while (p != null)

{

n++;

p = p.link;

}

Console.WriteLine("Number of elements in the Queue = " + n);

}

public bool SearchElement(int x)

{

int position = 1;

Node p = start;

while (p != null)

{

if (p.info == x)

break;

position++;

p = p.link;

}

if (p == null)

{

Console.WriteLine(x + "not found in Queue");

return false;

}

else

{

Console.WriteLine(x + "is at position " + position);

return true;

}

}

public void InsertElement(int data)

{

Node p;

Node temp = new Node(data);

if (start == null)

{

start = temp;

return;

}

p = start;

while (p.link != null)

p = p.link;

p.link = temp;

}

public void CreateQueue()

{

int i, n, data;

Console.Write("Enter the number of elements for dynamic Queue : ");

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

if (n == 0)

return;

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

{

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

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

InsertElement(data);

}

}

public void DeleteElement()

{

if (start == null)

return;

start = start.link;

}

}

public void Sort()

{

Node end, p, q;

for (end = null; end != front.link; end = p)

{

for (p = front; p.link != end; p = p.link)

{

q = p.link;

if (p.info < q.info)

{

int temp = p.info;

p.info = q.info;

q.info = temp;

}

}

} Display();

}

}

}

class Program

{

static void Main(string[] args)

{

int choice, data;

DynamicQueue Queue = new DynamicQueue();

Queue.CreateQueue();

while (true)

{

Console.WriteLine("1.Display Queue");

Console.WriteLine("2.Count the number of elements in Queue");

Console.WriteLine("3.search for an element in Queue");

Console.WriteLine("4.Insert an element in Queue");

Console.WriteLine("5.Delete element from Queue");

Console.WriteLine("6.Quit");

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

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

if (choice == 6)

break;

switch (choice)

{

case 1:

Queue.DisplayQueue();

break;

case 2:

Queue.CountElements();

break;

case 3:

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

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

Queue.SearchElement(data);

break;

case 4:

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

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

Queue.InsertElement(data);

break;

case 5:

Queue.DeleteElement();

break;

default:

Console.WriteLine("Wrong choice");

break;

}

Console.WriteLine();

}

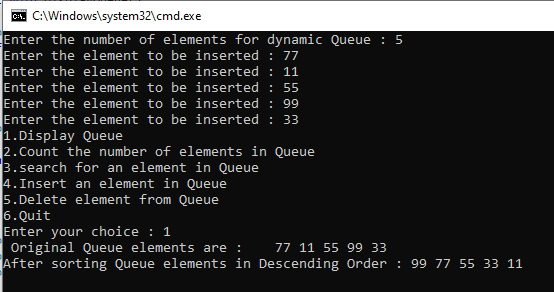
Console.WriteLine("Exiting....");

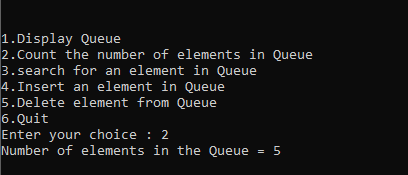
}

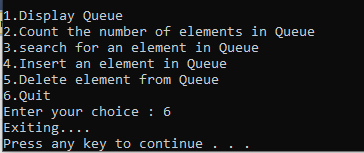
}

}

**Output:**







**Task No. 3:** With the help of Stacks, implement Polish notation in which you have to convert given expression to postfix notation. Show all the conversion using arrays or linked list.

**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:**

