**Bahria University**

**Software Engineering Department**



**Course: CSC 221 DATA STRUCTURE & ALGORITHMS**

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

**Assignment No:**

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

**Submitted By:**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **A** | **Q** | **I** | **B** |  | **M** | **E** | **H** | **M** | **O** | **O** | **D** |

**Enrollment No.:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 2 | - | 1 | 3 | 1 | 1 | 8 | 2 | - | 0 | 1 | 4 |

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

**Submitted To:**

**Engr. Saniya Sarim**

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

Bahria University,

Karachi Campus



LAB Assignment NO.

02.

LIST OF TASKS

|  |  |
| --- | --- |
| TASK NO | OBJECTIVE |
| 01. | Using dynamic Stack class write a code which takes n number of inputs from user and create sorted list. I.e. in ascending order. |
| 02. | Using dynamic queue class write a code which takes n number of inputs from user and create sorted list in descending order. |
| 03. | 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. |
|  |  |
|  |  |
|  |  |

Submitted On:

26-12-2019

(Date: DD/MM/YY)

Q 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.

Input:-

Linked List:-

class linkedlist

{

internal Node start = null;

public class Node

{

public int val;

public Node next;

public Node(int val)

{

this.val = val;

}

}

public Node sortedMerge(Node a, Node b)

{

Node result = null;

if (a == null)

return b;

if (b == null)

return a;

if (a.val <= b.val)

{

result = a;

result.next = sortedMerge(a.next, b);

}

else

{

result = b;

result.next = sortedMerge(a, b.next);

}

return result;

}

public Node MergeSort(Node c)

{

if (c == null || c.next == null)

{

return c;

}

Node middle = getMiddle(c);

Node NextOfMiddle = middle.next;

middle.next = null;

Node left = MergeSort(c);

Node right = MergeSort(NextOfMiddle);

Node sortedlist = sortedMerge(left, right);

return sortedlist;

}

public Node getMiddle(Node c)

{

if (c == null)

return c;

Node first = c.next;

Node second = c;

while (first != null)

{

first = first.next;

if (first != null)

{

second = second.next;

first = first.next;

}

}

return second;

}

public void Push(int val)

{

Node temp = new Node(val);

temp.next = start;

start = temp;

}

public void Peek()

{

while (start.next != null)

{

start = start.next;

}

Console.Write(start.val + " ");

}

public void Display(Node startref)

{

Console.WriteLine("Stack values are : ");

while (startref != null)

{

Console.Write(startref.val + " ");

startref = startref.next;

}

Console.WriteLine();

}

public void SortedDisplay(Node startref)

{

while (startref != null)

{

Console.Write(startref.val + " ");

startref = startref.next;

}

Console.WriteLine();

}

}

DynamicStack:-

class DynamicStack

{

linkedlist obj;

public DynamicStack()

{

obj = new linkedlist();

}

public void push(int val)

{

obj.Push(val);

}

public void Peek()

{

obj.Peek();

}

public void display()

{

obj.Display(obj.start);

}

}

Main:-

Console.WriteLine("\tDSA Lab Assignment No. 2\n\t\tTask No. 01");

DynamicStack obj = new DynamicStack();

linkedlist objl = new linkedlist();

while (true)

{

Console.WriteLine("What action you want to perform");

Console.WriteLine("1-Push\n2-Show Sorted List\n3-Peek\n0-Exit");

int opt = int.Parse(Console.ReadLine());

if (opt == 1)

{

Console.WriteLine("Enter value to push in Stack");

int val = int.Parse(Console.ReadLine());

objl.Push(val);

objl.Display(objl.start);

}

else if (opt == 2)

{

objl.start = objl.MergeSort(objl.start);

Console.Write("\n|<<<<<~~~~~~~~~ Sorted List ~~~~~~~~~>>>>>| \n");

objl.SortedDisplay(objl.start);

Console.Write("|<<<<<~~~~~~~~~---------------~~~~~~~~~>>>>>| \n");

}

else if (opt == 3)

{

obj.Peek();

}

else if (opt == 0)

{

Console.WriteLine("Successfully Exit.");

Environment.Exit(0);

}

else

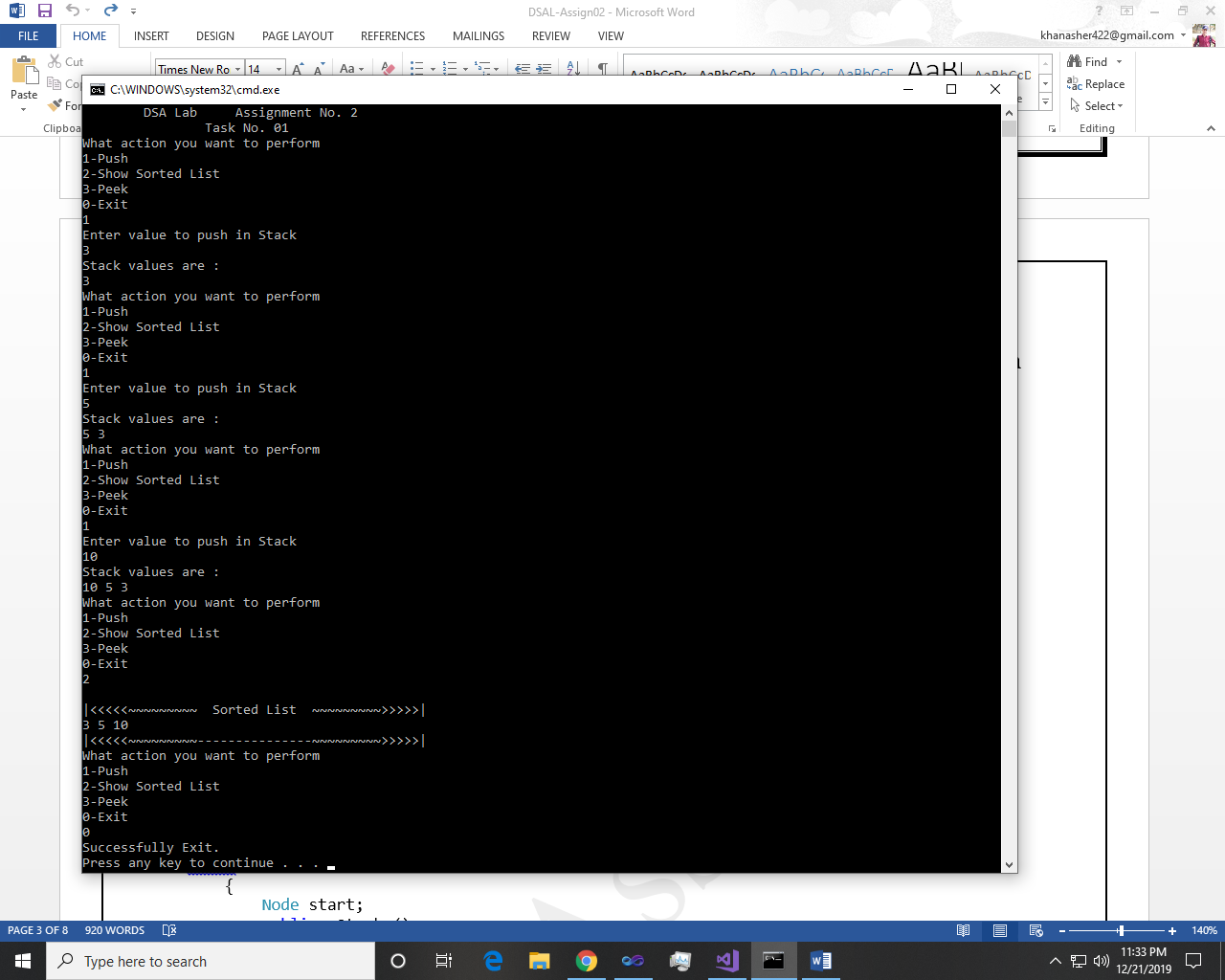
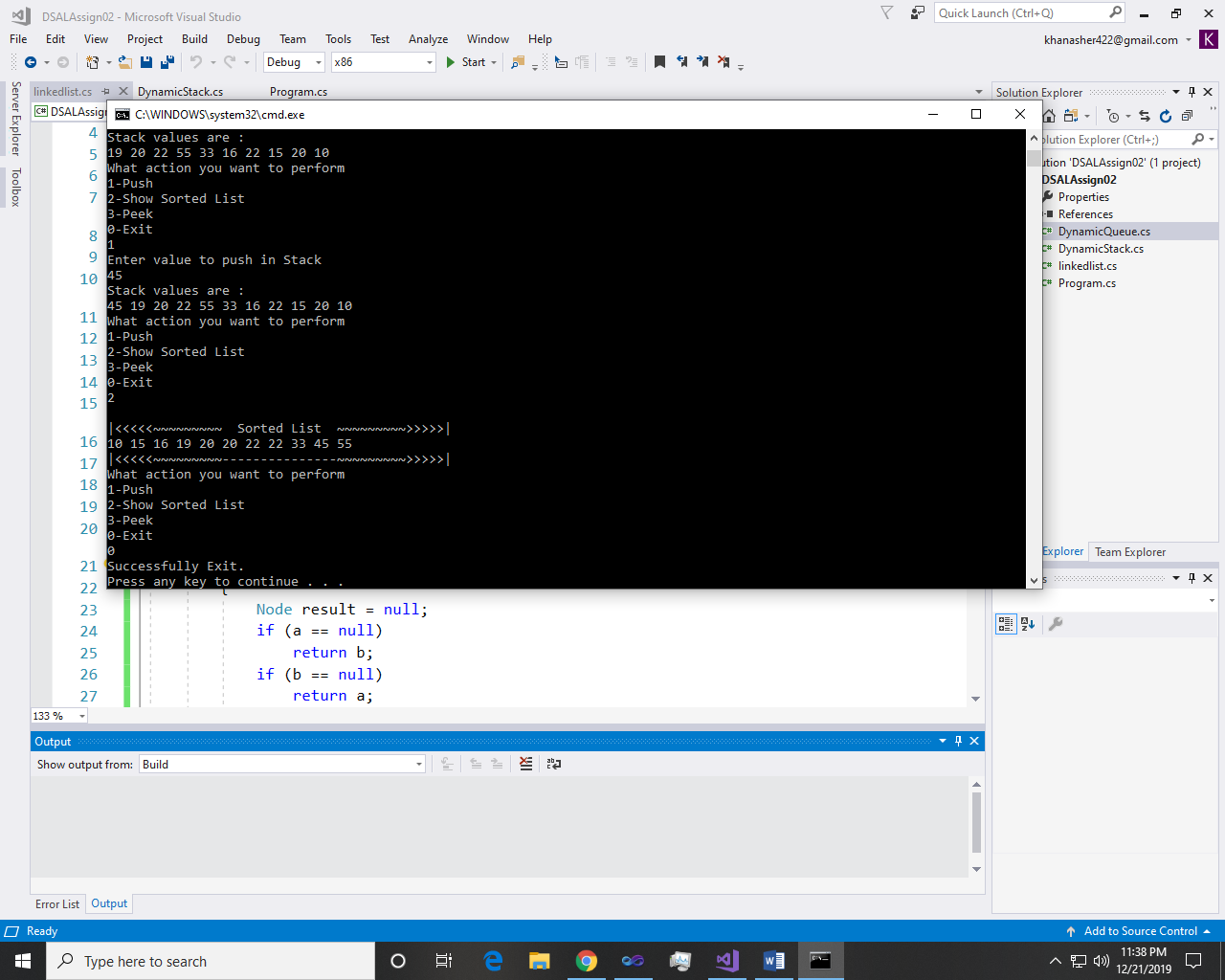
{

Console.WriteLine("Invalid input ...... try Again");

}

}

Output:-

~~~~~~\*\*//\*\*~~~~~~

Q 2: - Using dynamic queue class write a code which takes n number of inputs from user and create sorted list in descending order.

Input:-

Liked List:-

class linkedlist

{

internal Node start = null;

public class Node

{

public int val;

public Node next;

public Node(int val)

{

this.val = val;

}

}

public Node sortedMerge(Node a, Node b)

{

Node result = null;

if (a == null)

return b;

if (b == null)

return a;

if (a.val >= b.val)

{

result = a;

result.next = sortedMerge(a.next, b);

}

else

{

result = b;

result.next = sortedMerge(a, b.next);

}

return result;

}

public Node MergeSort(Node c)

{

if (c == null || c.next == null)

{

return c;

}

Node middle = getMiddle(c);

Node NextOfMiddle = middle.next;

middle.next = null;

Node left = MergeSort(c);

Node right = MergeSort(NextOfMiddle);

Node sortedlist = sortedMerge(left, right);

return sortedlist;

}

public Node getMiddle(Node c)

{

if (c == null)

return c;

Node first = c.next;

Node second = c;

while (first != null)

{

first = first.next;

if (first != null)

{

second = second.next;

first = first.next;

}

}

return second;

}

public void Enqueue(int val)

{

Node temp = new Node(val);

temp.next = start;

start = temp;

}

public void Peek()

{

while (start.next != null)

{

start = start.next;

}

Console.Write(start.val + " ");

}

public void Display(Node startref)

{

Console.WriteLine("Queues value are : ");

while (startref != null)

{

Console.Write(startref.val + " ");

startref = startref.next;

}

Console.WriteLine();

}

public void SortedDisplay(Node startref)

{

while (startref != null)

{

Console.Write(startref.val + " ");

startref = startref.next;

}

Console.WriteLine();

}

}

Dynamic Queue:-

class DynamicQueue

{

linkedlist obj;

public DynamicQueue()

{

obj = new linkedlist();

}

public void Enqueue(int val)

{

obj.Enqueue(val);

}

public void display()

{

obj.Display(obj.start);

}

}

Main:-

Console.WriteLine("\tDSA Lab Assignment No. 2\n\t\tTask No. 02");

DynamicQueue obj = new DynamicQueue();

linkedlist objl = new linkedlist();

while (true)

{

Console.WriteLine("What you want to Perform:");

Console.WriteLine("1-Enqueue\n2-Show Sorted List\n0-Exit");

int opt = int.Parse(Console.ReadLine());

if (opt == 1)

{

Console.WriteLine("Enter value to EnQueue");

int val = int.Parse(Console.ReadLine());

objl.Enqueue(val);

objl.Display(objl.start);

}

if (opt == 2)

{

objl.start = objl.MergeSort(objl.start);

Console.Write("\n|<<<<<~~~~~~~~~ Sorted List in Descending Order ~~~~~~~~~>>>>>| \n");

objl.SortedDisplay(objl.start);

Console.Write("|<<<<<<<~~~~~~~~~~~~~~~---------------~~~~~~~~~~~~~~~~~>>>>>>>| \n");

}

else if (opt == 0)

{

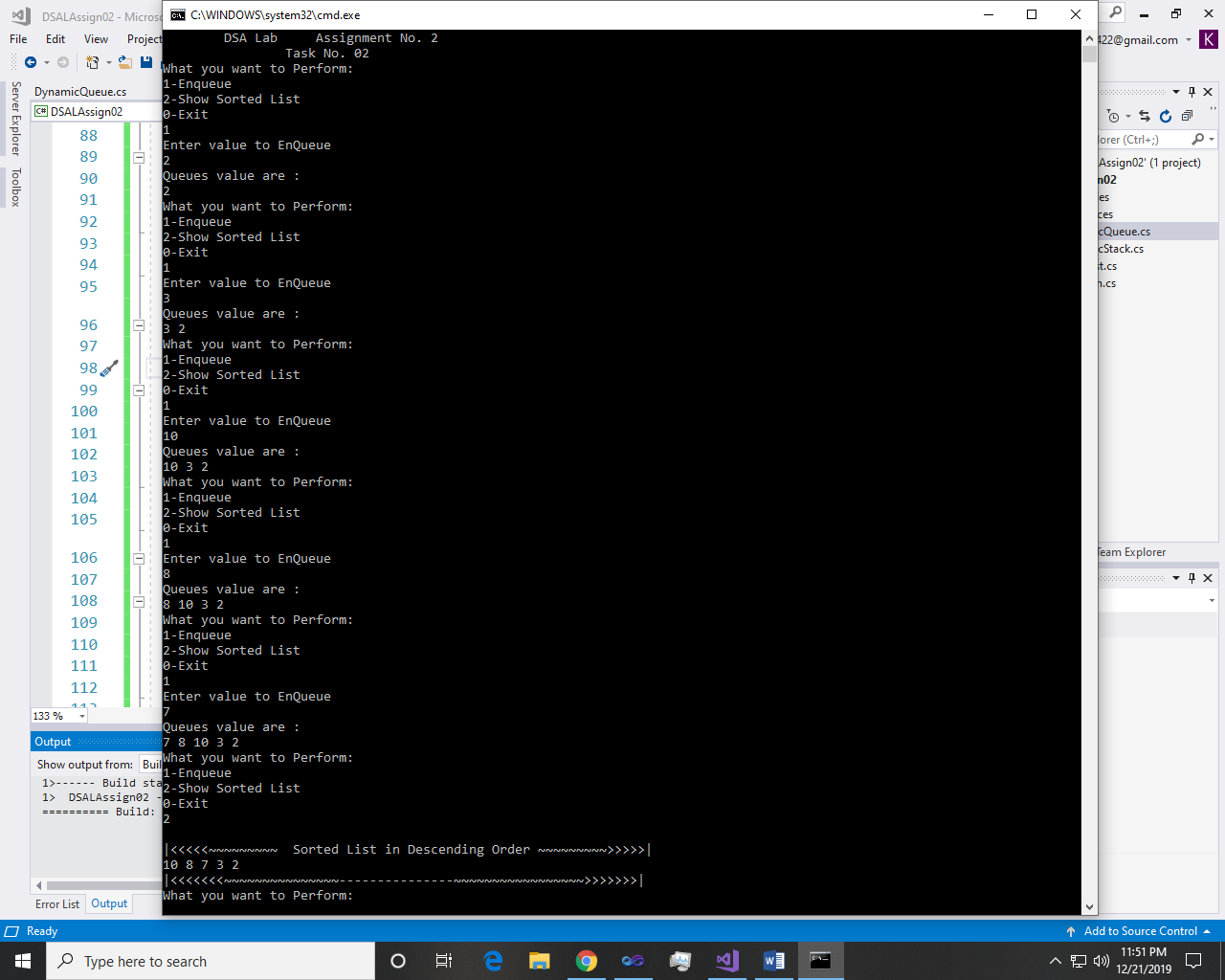
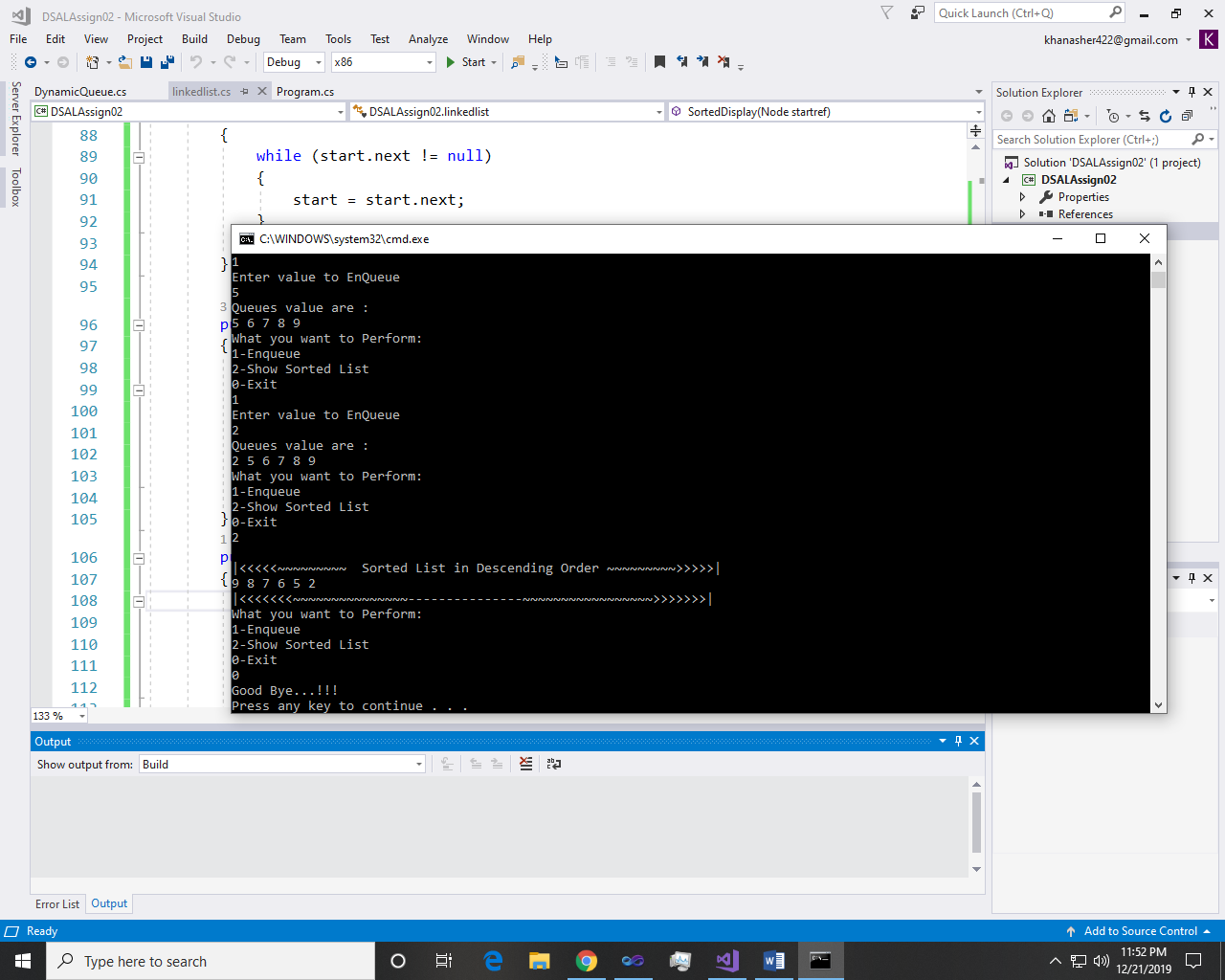
Console.WriteLine("Good Bye...!!!");

Environment.Exit(0);

}

}

Output:-

~~~~~~\*\*//\*\*~~~~~~

Q 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.

Input:-

Stack:-

class Stacks

{

Node start;

public Stacks()

{

start = new Node();

}

public Stacks(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 = (char)start.next.data;

return val;

}

return ' ';

}

public char Pop()

{

if (!Underflow())

{

char val;

if (start.next.next != null)

{

val = (char)start.next.data;

start.next = start.next.next;

return val;

}

else

{

val = (char)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:-

internal Node next;

internal char data;

public Nodes()

{

next = null;

}

public Nodes(char val)

{

this.data = val;

}

public Nodes(Node obj)

{

this.next = obj;

}

Main:-

Console.WriteLine("\tDSA Lab Assignment No. 2\n\t\tTask No. 03");

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)

{

Stacks obj = new Stacks();

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)

{

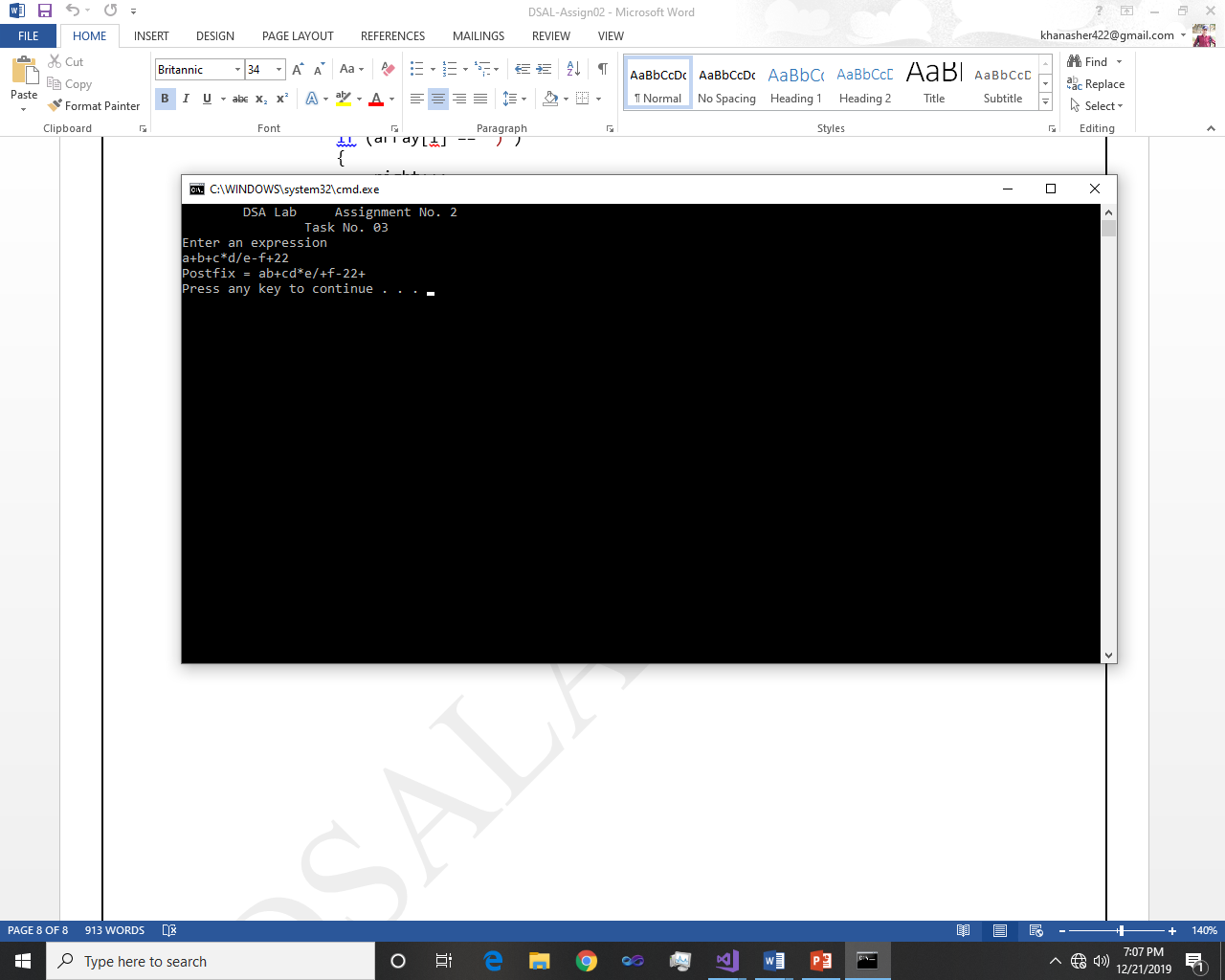
return true;

}

return false;

}

Output:-



~~~~~~\*\*/**THE END**/\*\*~~~~~~