Q1: - 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

Input:-

Stack Class :-

class Stack

{

int[] A;

int top = -1;

public Stack(int limit)

{

A = new int[limit];

}

public void Push(int val)

{

if (!Overflow())

{

top++;

A[top] = val;

}

else

{

Console.WriteLine("Stack Overflow");

}

}

public int Pop()

{

if (!Underflow())

{

top = top - 1;

return top;

}

else return -00;

}

public bool Overflow()

{

if (top >= A.Length-1)

{

return true;

}

else return false;

}

public bool Underflow()

{

if (top==-1)

{

return true;

}

else return false;

}

public int Peek()

{

return A[top];

}

public void Display()

{

Console.WriteLine("OverFlow : "+Overflow());

Console.WriteLine("UnderFlow : "+Underflow());

Console.WriteLine("Peek Value : "+Peek());

}

Main:-

static void Main(string[] args)

{

Console.Write("Enter Limit of Stack : ");

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

Stack obj = new Stack(limit);

Console.WriteLine("Enter Values for Stack");

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

{

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

obj.Push(value);

}

obj.Display();

Console.WriteLine("If you want to add a value Enter it:");

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

obj.Push(newval);

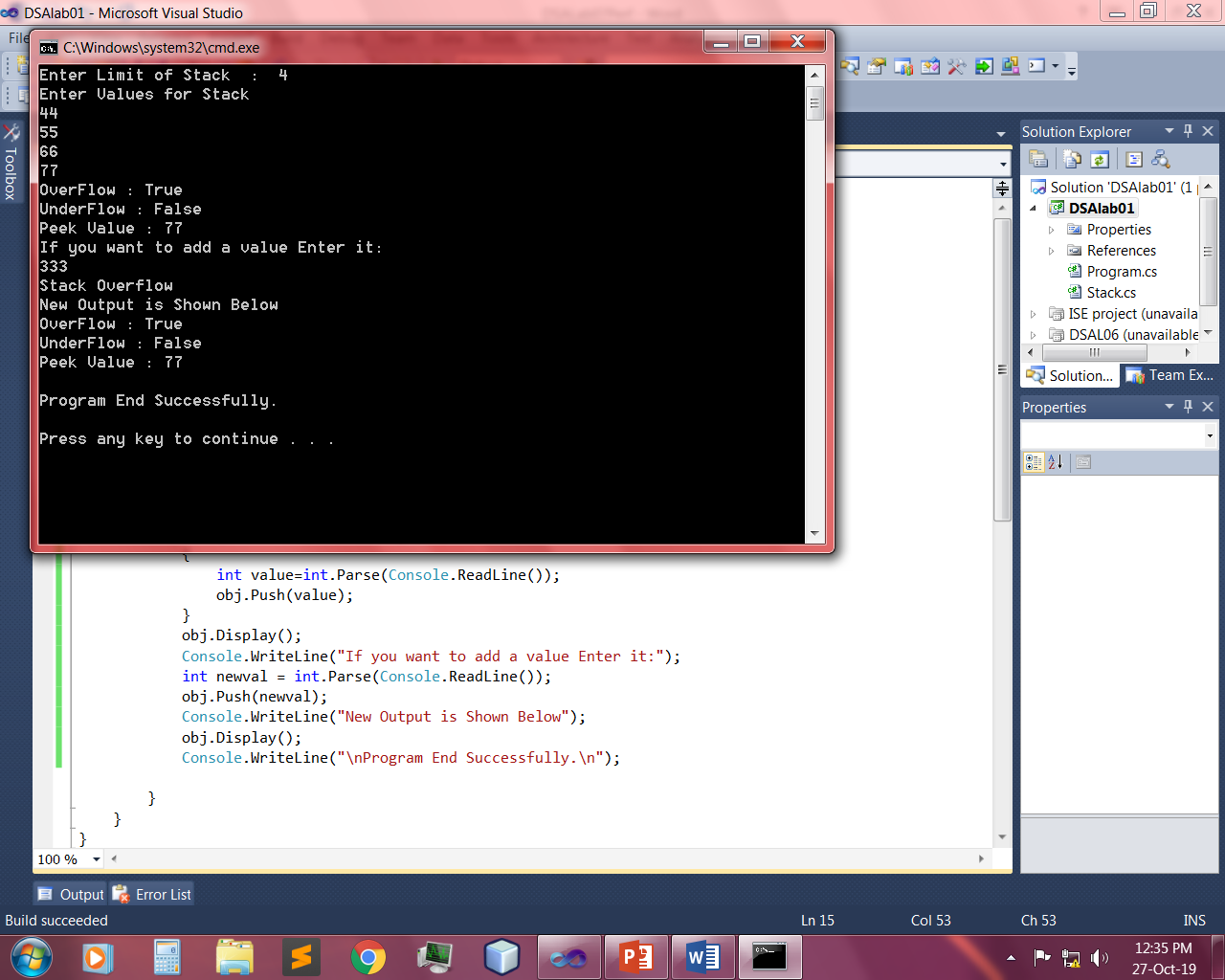
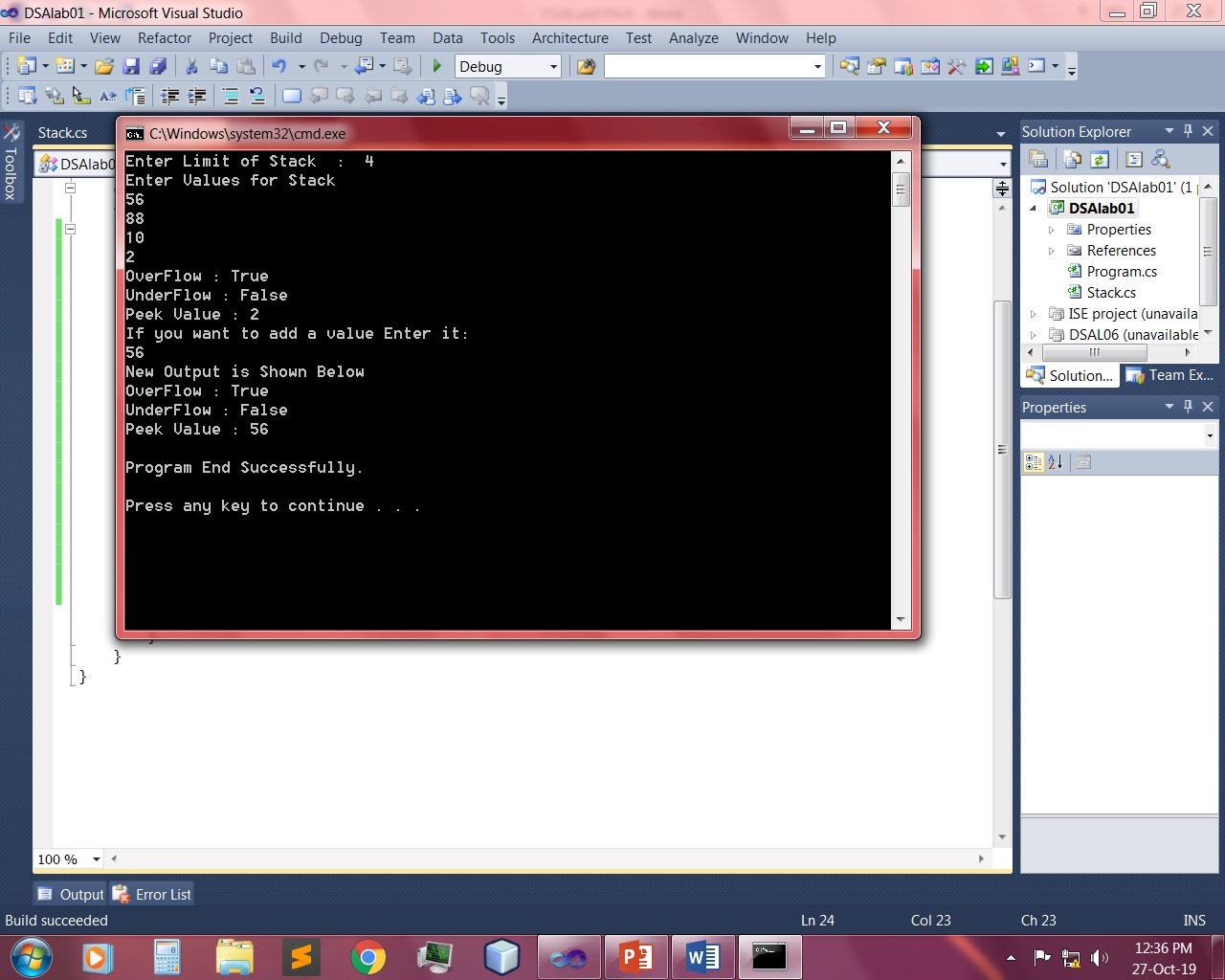
Console.WriteLine("New Output is Shown Below");

obj.Display();

Console.WriteLine("\nProgram End Successfully.\n");

}

Output:-

Q2: - Using Stack class write a code for max and min.

Input:-

Class Stack:-

class Stack

{

int[] A;

int top = -1;

public Stack(int limit)

{

A = new int[limit];

}

public void Push(int val)

{

if (!Overflow())

{

top++;

A[top] = val;

}

else

{

Console.WriteLine("Stack Overflow");

}

}

public int Pop()

{

if (!Underflow())

{

top = top - 1;

return A[top];

}

else return -00;

}

public bool Overflow()

{

if (top >= A.Length-1)

{

return true;

}

else return false;

}

public bool Underflow()

{

if (top==-1)

{

return true;

}

else return false;

}

public int Peek()

{

return A[top];

}

public int GetMinValue()

{

if (!Underflow())

{

int min = A[0];

for (int i = 0; i <= this.top; i++)

{

if (min > A[i])

min = A[i];

}

return min;

}

else

{

return -1;

}

}

public int GetMaxValue()

{

if (!Underflow())

{

int max = A[0];

for (int i = 0; i <= this.top; i++)

{

if (max < A[i])

{

max = A[i];

}

}

return max;

}

else

{

return -1;

}

}

public void Display()

{

Console.WriteLine("Max Value : "+GetMaxValue());

Console.WriteLine("Min Value : "+GetMinValue());

}

Main:-

static void Main(string[] args)

{

Console.Write("Enter Limit of Stack : ");

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

Stack obj = new Stack(limit);

Console.WriteLine("Enter Values for Stack");

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

{

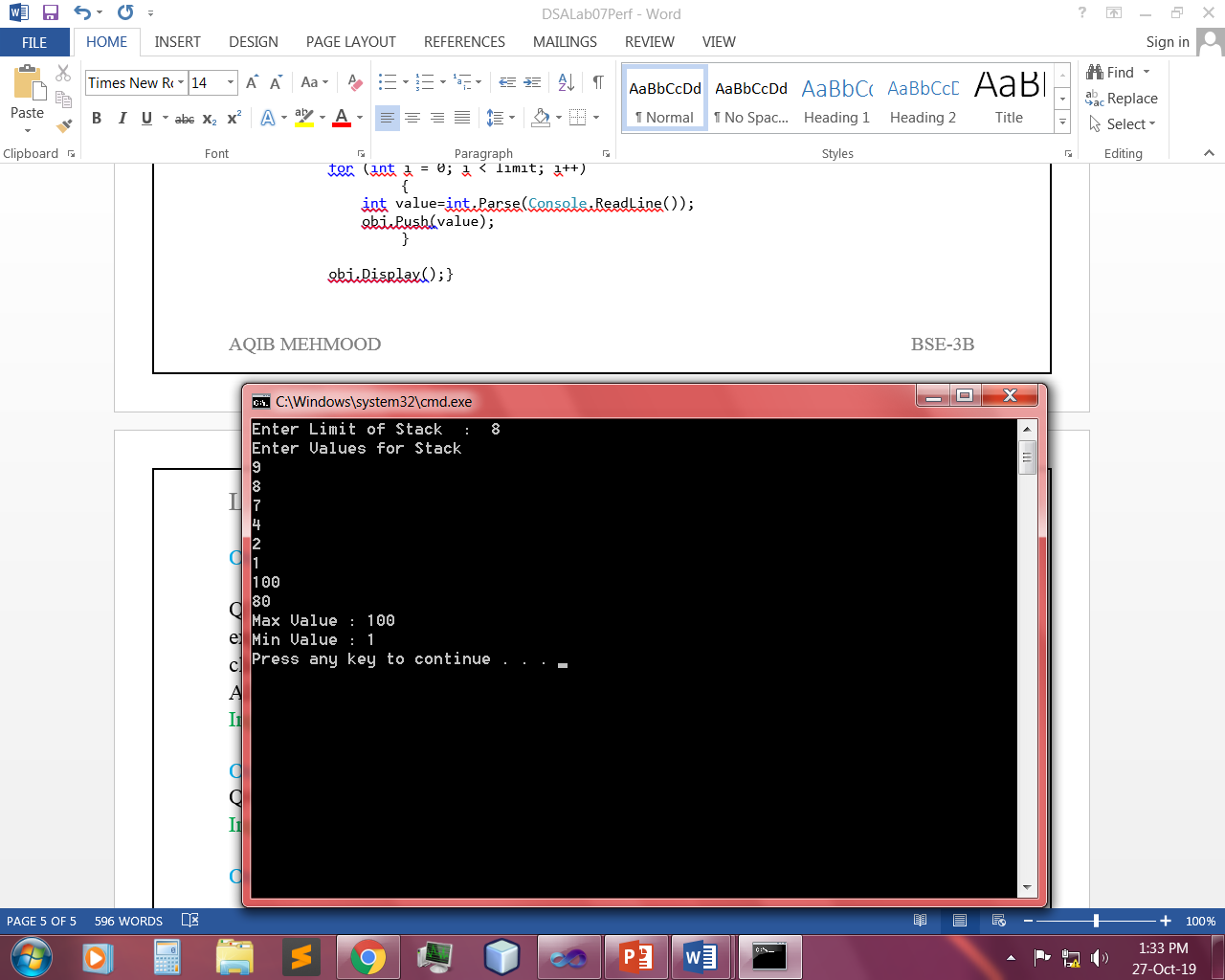
int value=int.Parse(Console.ReadLine());

obj.Push(value);

}

obj.Display();}

Output:-



Q4: - Take a String as input and print its reverse using your Stack class.

Input:-

Stack Class:-

class Stack

{

char[] A;

int top;

public Stack(string val)

{

A = val.ToCharArray();

this.top = A.Length;

}

public void ReverseString()

{

while (top>0)

{

top--;

Console.Write(A[top]);

}

Console.WriteLine("\n");

}

Main:-

static void Main(string[] args)

{

Console.Write("Simple String : ");

string val = Console.ReadLine();

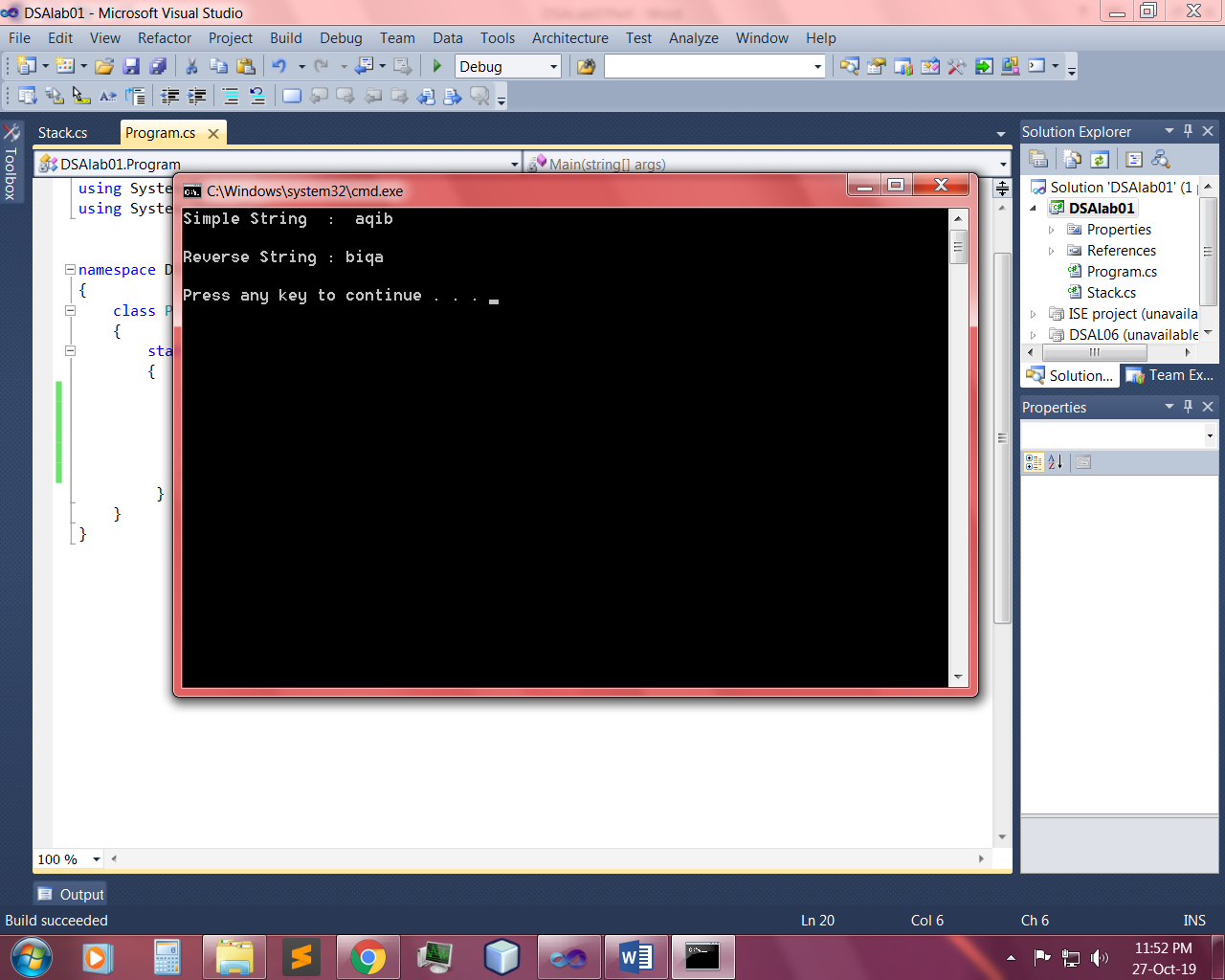
Stack obj = new Stack(val);

Console.Write("\nReverse String : ");

obj.ReverseString();

}

Output:-



Q5: - Design & implement all methods of Simple Queue.

Input:-

Queue:

class Queue

{

public int rare = 0;

public int front = 0;

public int[] arr = new int[3];

public Queue()

{ }

public bool enqueue(int a)

{

if (!overflow())

{

arr[rare] = a;

rare++;

return true;

}

else

{

return false;

}

}

public bool overflow()

{

if (rare == arr.Length)

{

return true;

}

else

{

return false;

}

}

public int dequeue()

{

if (!underflow())

{

return arr[front++];

}

else

{

return -1;

}

}

public bool underflow()

{

if (front < 0)

{

return true;

}

else

{

return false;

}

}

public void ArrayShow()

{

Console.Write("Queue : ");

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

{

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

}

Console.WriteLine();

}

public int Front()

{

if (!underflow())

{

return arr[front];

}

return -1;

}

public int Rear()

{

if (!overflow())

{

return arr[--rare];

}

return -1;

}

public void display()

{

Console.WriteLine("Arr Length : "+arr.Length);

Console.WriteLine("Underflow : "+underflow());

Console.WriteLine("Overflow : " + overflow());

Console.WriteLine("Front Value : "+Front());

Console.WriteLine("Rare Value : "+Rear());

}

class Queue

{

public int rare = 0;

public int front = 0;

public int[] arr = new int[3];

public Queue()

{ }

public bool enqueue(int a)

{

if (!overflow())

{

arr[rare] = a;

rare++;

return true;

}

else

{

return false;

}

}

public bool overflow()

{

if (rare == arr.Length)

{

return true;

}

else

{

return false;

}

}

public int dequeue()

{

if (!underflow())

{

return arr[front++];

}

else

{

return -1;

}

}

public bool underflow()

{

if (front < 0)

{

return true;

}

else

{

return false;

}

}

public void ArrayShow()

{

Console.Write("Queue : ");

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

{

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

}

Console.WriteLine();

}

public int Front()

{

if (!underflow())

{

return arr[front];

}

return -1;

}

public int Rear()

{

if (!overflow())

{

return arr[--rare];

}

return -1;

}

public void display()

{

Console.WriteLine("Arr Length : "+arr.Length);

Console.WriteLine("Underflow : "+underflow());

Console.WriteLine("Overflow : " + overflow());

Console.WriteLine("Front Value : "+Front());

Console.WriteLine("Rare Value : "+Rear());

}

Main:

static void Main(string[] args)

{

Queue obj = new Queue();

obj.enqueue(5);

obj.enqueue(6);

obj.ArrayShow();

obj.display();

Console.WriteLine();

obj.enqueue(4);

obj.ArrayShow();

obj.display();

Console.WriteLine();

obj.dequeue();

obj.dequeue();

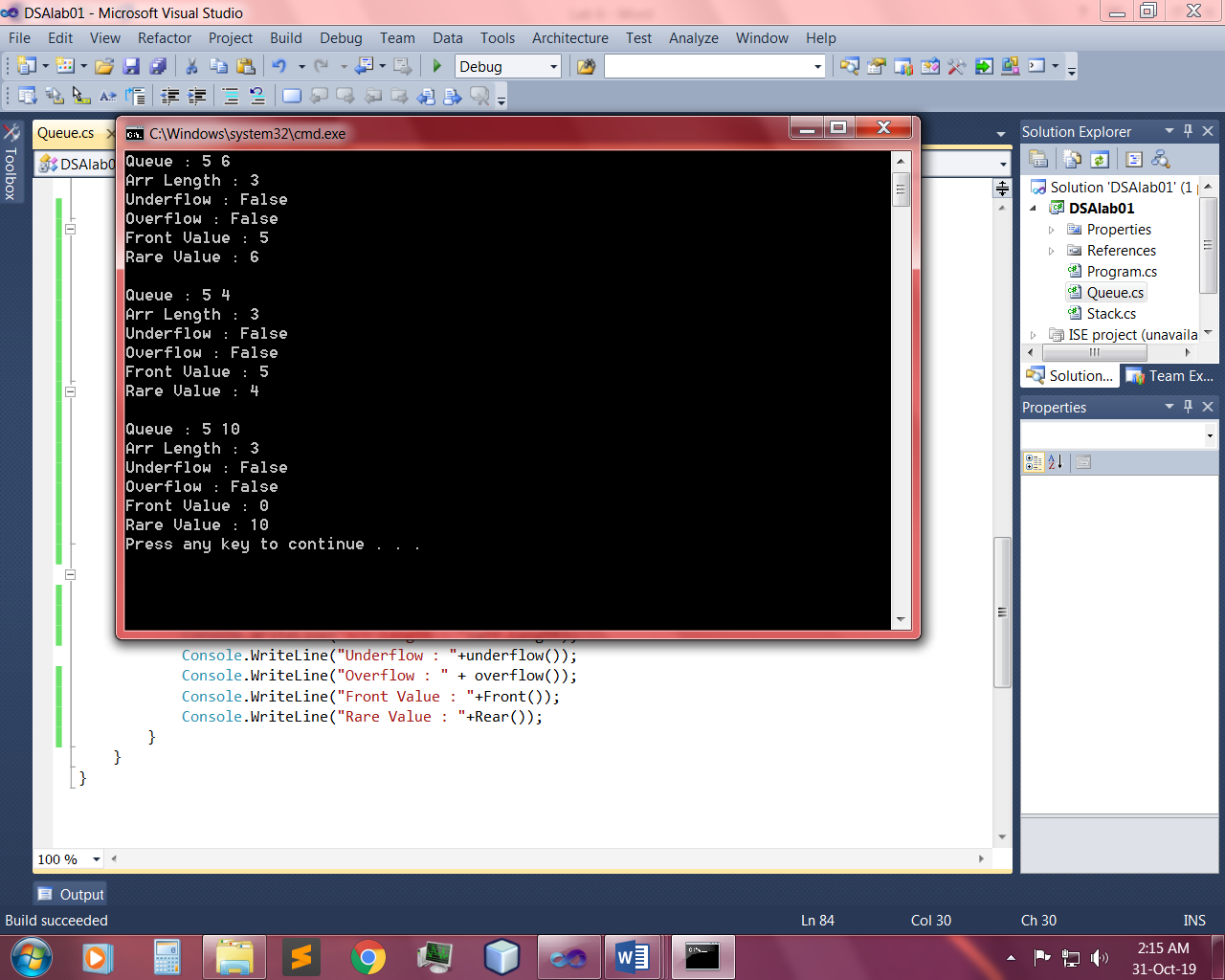
obj.enqueue(10);

obj.ArrayShow();

obj.display();

}

Output:-



Q6: - Design & implement all methods of Circular Queue.

Input:-

CircularQueue:

class CircularQueue

{

private int[] array;

private int Length;

private int front;

private int count=0;

private int rear;

public CircularQueue(int length)

{

array = new int[length];

this.Length = length;

rear = -1;

front = 0;

}

public int RearCount()

{

if(rear == -1)

{

return ++rear;

}

rear = (rear + 1) % Length;

return rear;

}

public int FrontCount()

{

front = (front + 1) % Length;

return front;

}

public bool Enqueue(int val)

{

if (!Overflow())

{

rear = RearCount();

array[rear]=val;

count++;

return true;

}

return false;

}

public int Dequeue()

{

if (!UnderFlow())

{

int val = array[front];

front = FrontCount();

count--;

return val;

}

return -1;

}

public int Front()

{

if (!UnderFlow())

{

return array[front];

}

return -1;

}

public void Display()

{

Console.WriteLine("Your array");

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

{

Console.Write(array[i+1] + " ");

}

Console.WriteLine();

}

public bool UnderFlow()

{

if (count <= 0)

return true;

return false;

}

public bool Overflow()

{

if (count >= Length)

return true;

return false;

}

Main:

static void Main(string[] args)

{

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

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

CircularQueue obj = new CircularQueue(length);

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

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

obj.Enqueue(val);

Console.WriteLine("Value successfully enqueued");

obj.Display();

Console.WriteLine("If you want to Dequeue Value Enter 1 Else Enter 0 to Exit");

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

if (opt==1)

{

int val1 = obj.Dequeue();

obj.Dequeue();

Console.WriteLine("Value successfully dequeued");

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

}

else

{

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

}

Console.WriteLine();

Console.WriteLine("If you want to View Front Value Enter 1 Else Enter 0 to Exit");

int opt1 = int.Parse(Console.ReadLine());

if (opt1 == 1)

{

int val2 = obj.Front();

Console.WriteLine("Front value = " + val2);

}

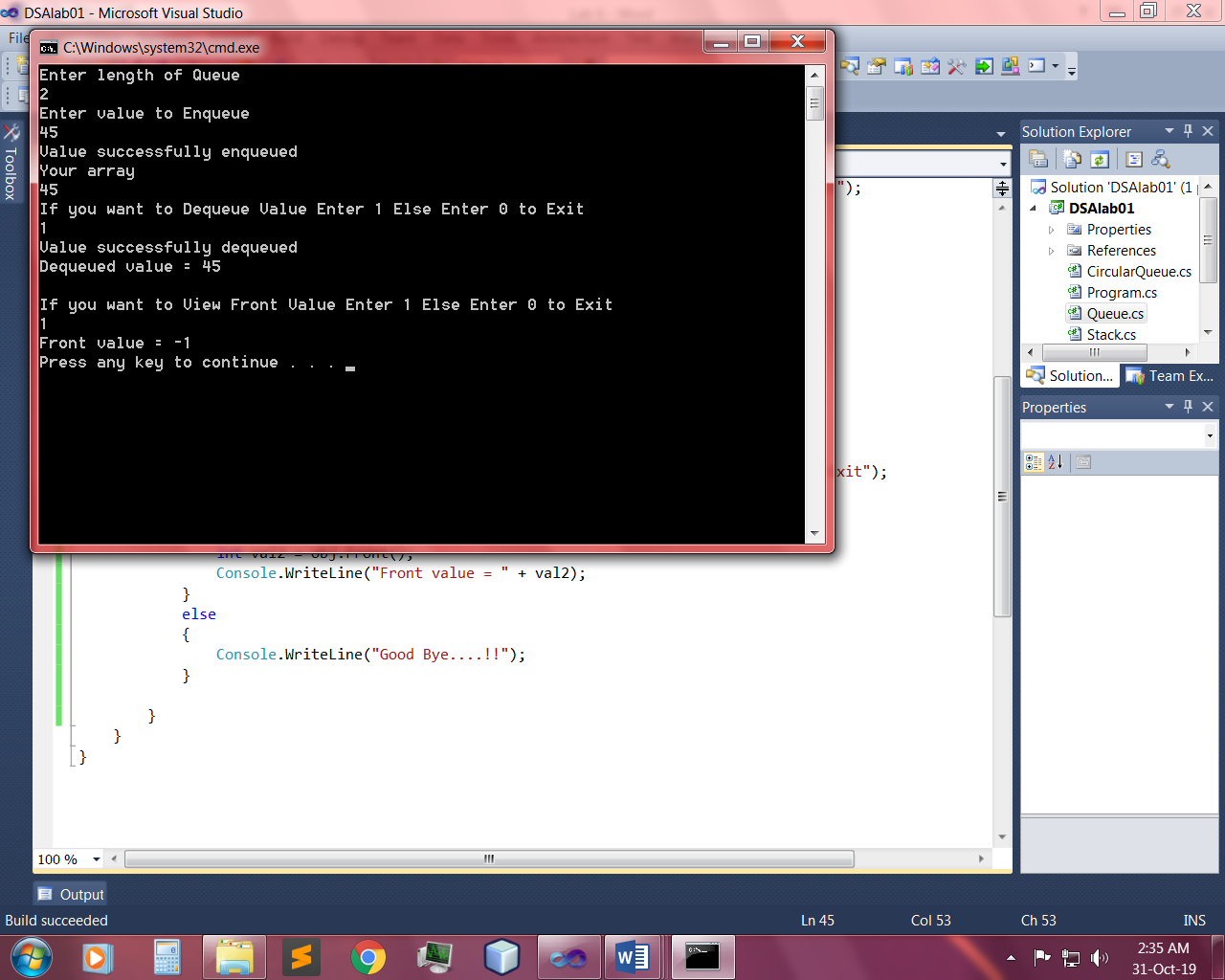
else

{

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

}

Output:-



Q7: - Design and implement for Priority Queue.

1. Method 1: Ordering in/ after Enqueue method
2. Method 2: Separate queues for different priorities.

Input:-

Main:

static void Main(string[] args)

{

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

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

PriorityQueue obj = new PriorityQueue(length);

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");

}

Console.WriteLine();

Console.WriteLine("If you want to Dequeue Value Enter 1 Else Enter 0 to Exit");

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

if (opt == 1)

{

int val = obj.Dequeue();

if (val >= 0)

{

Console.WriteLine("Value successfully dequeued");

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

}

else

{

Console.WriteLine("Stack Underflow");

}

}

else

{

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

}

Console.WriteLine();

Console.WriteLine("If you want to View Front Value Enter 1 Else Enter 0 to Exit");

int opt1 = int.Parse(Console.ReadLine());

if (opt1 == 1)

{

int val1 = obj.Front();

if (val1 >= 0)

{

Console.WriteLine("Front value = " + val1);

}

else

{

Console.WriteLine("Stack empty");

}

}

else

{

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

}

}

PriorityQueue:

class PriorityQueue

{

int[,] array;

int Length, front, rear;

public PriorityQueue(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;

}

}

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++)

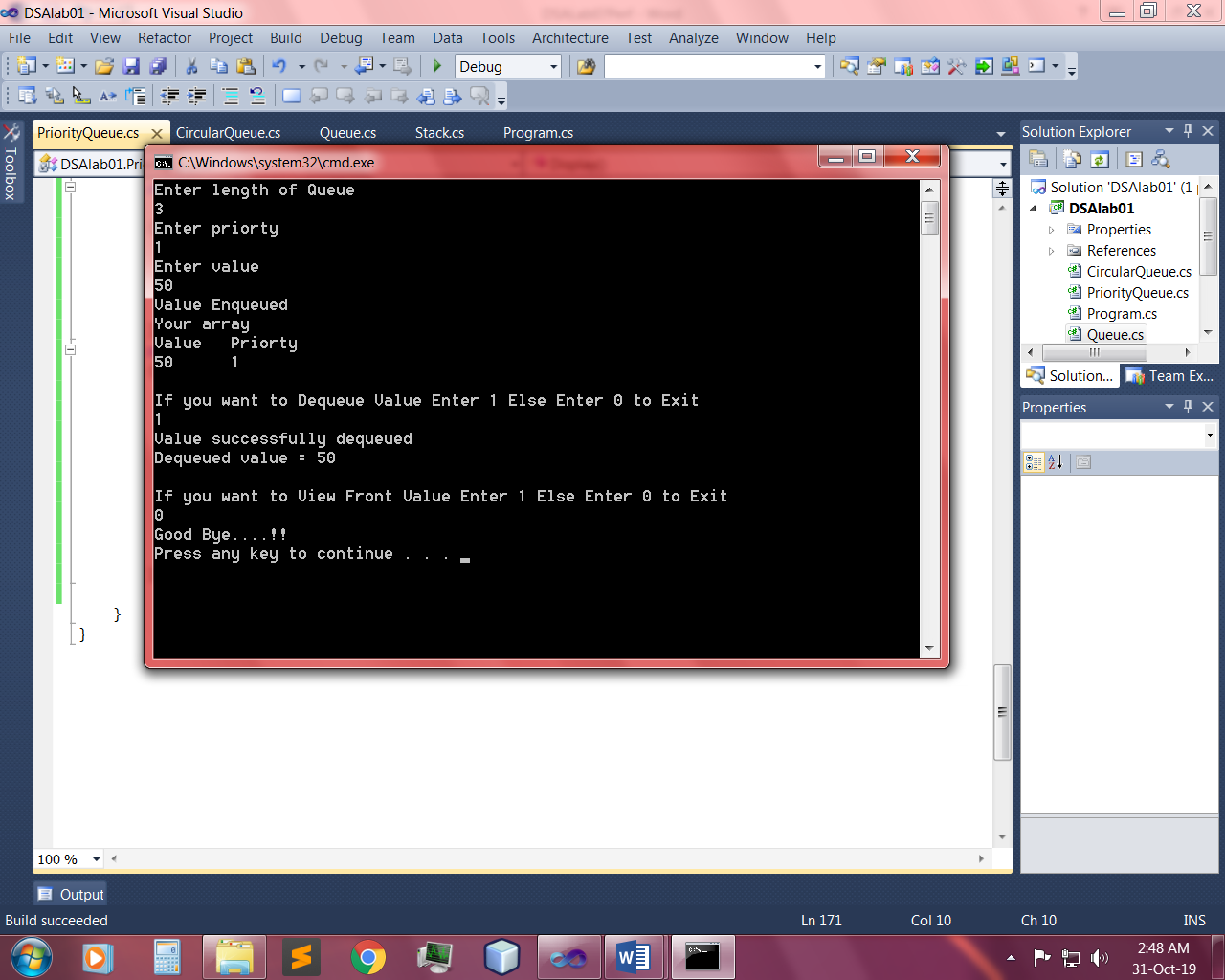
{

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

}

}

Output:-



Q8: - With the help of Stacks, implement Polish notation in which you have to convert given expression to postfix notation.

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("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)

{

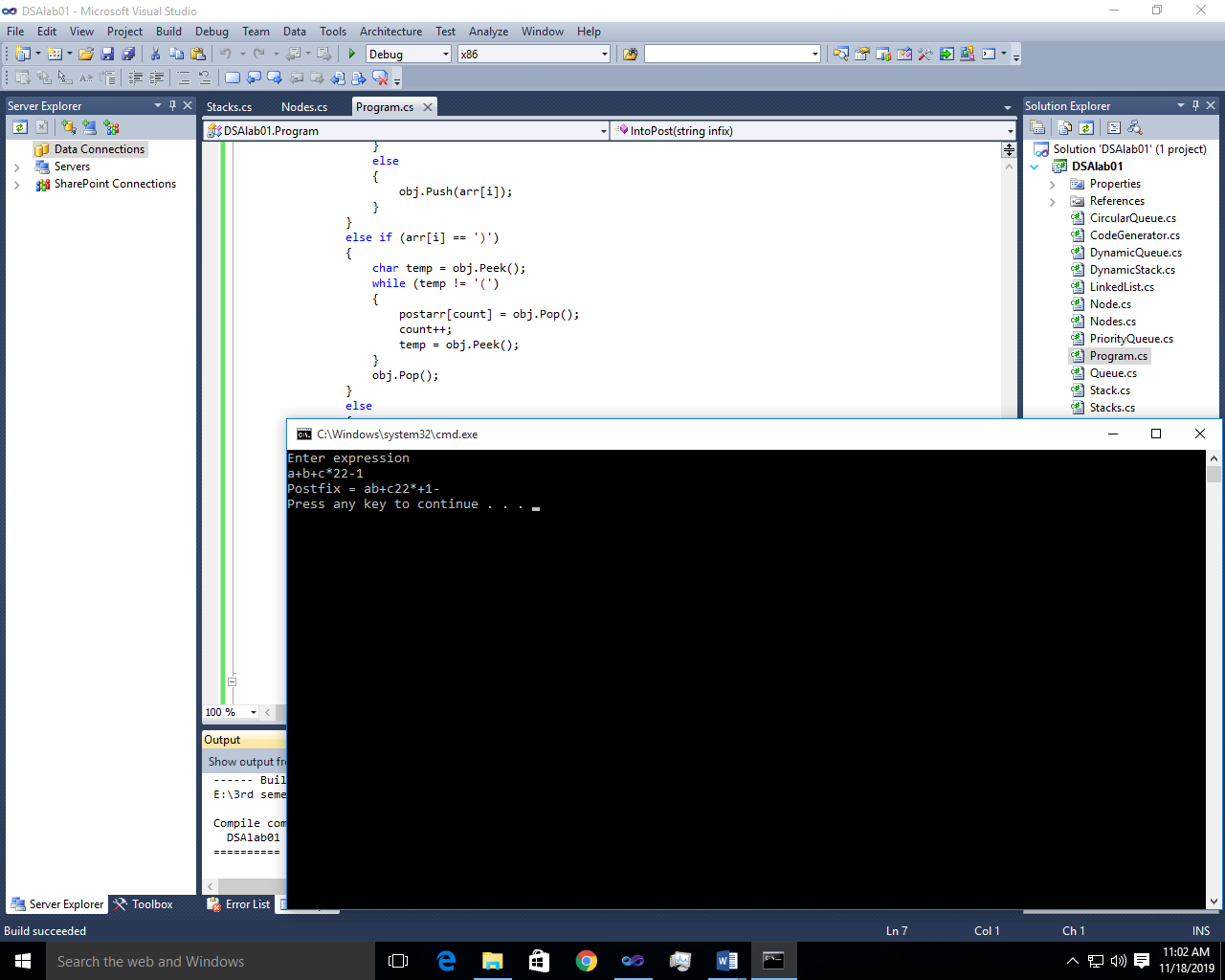
return true;

}

return false;

}

Output:-



Q3: - 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.

Input:-

public partial class Form2 : Form

{

public Form2()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

string Input = textBox1.Text;

if (ValidExpression(Input))

{

label3.Text = "Valid";

label3.ForeColor = Color.Green;

}

else

{

label3.Text = "Invalid";

label3.ForeColor = Color.Red;

}

}

private 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)

{

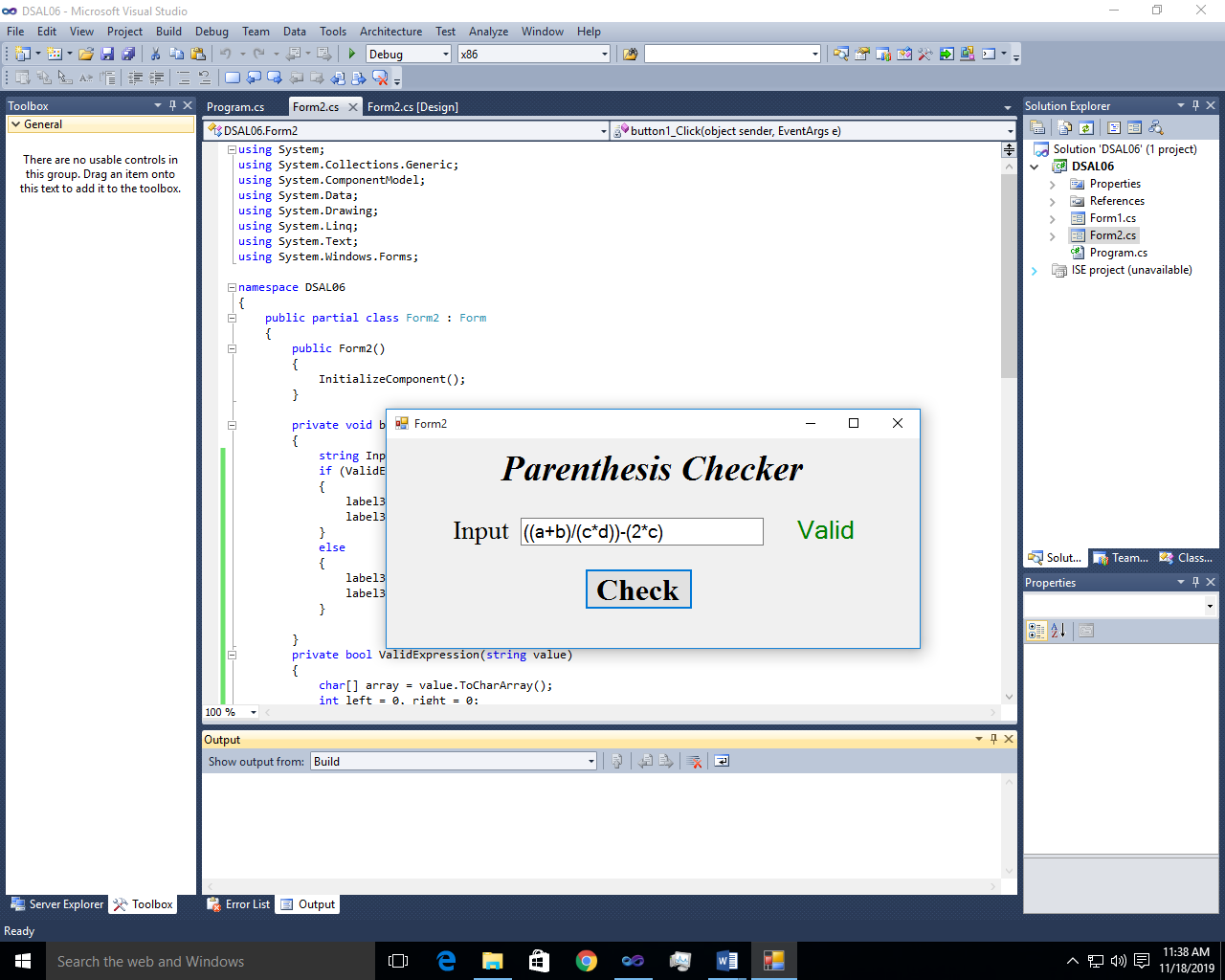
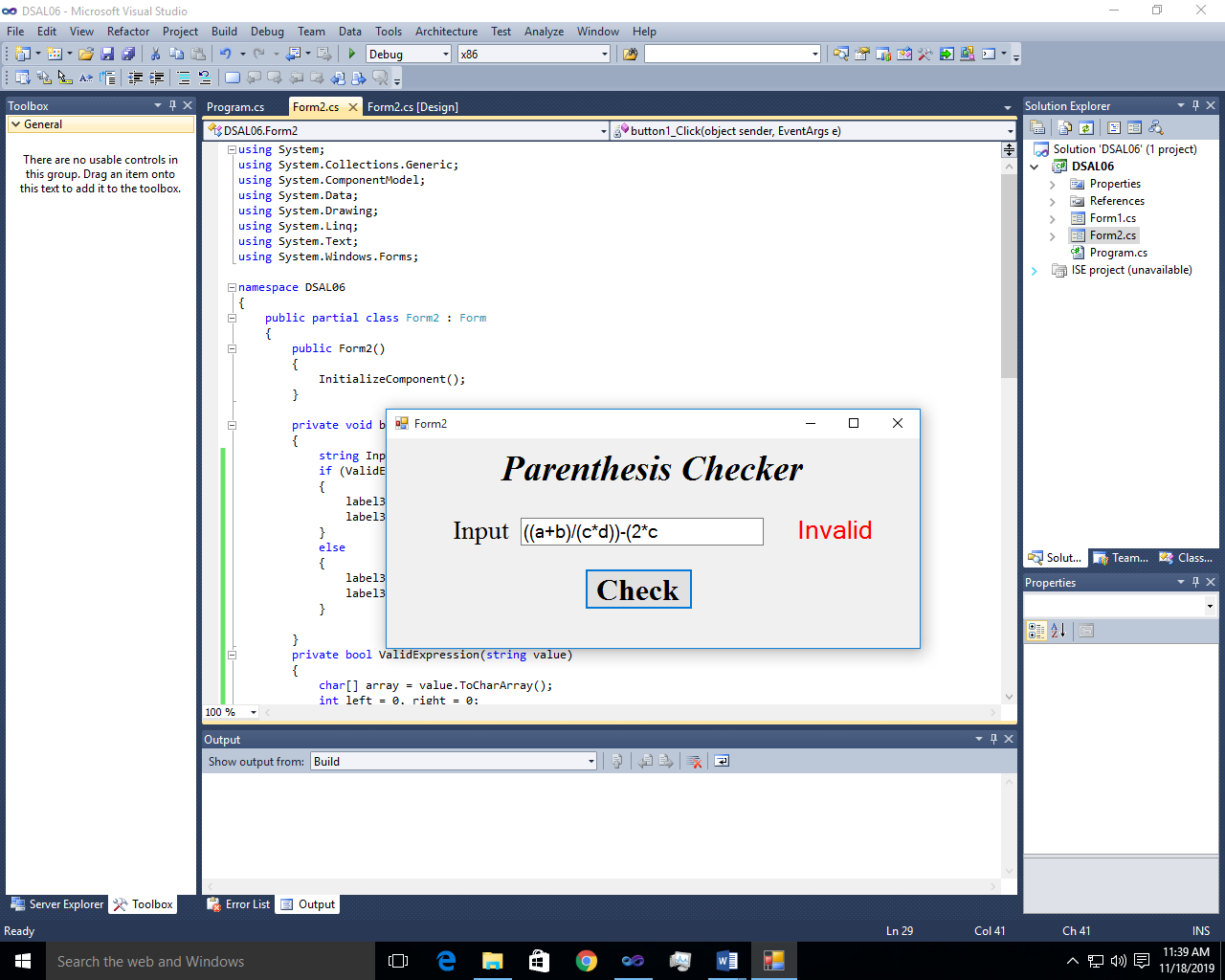
return true;

}

return false;

}

Output:-

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