

**1. (a) Write a method in C# that takes an array as an input parameter and uses two methods, one to find the largest array element and other to compute the average of array elements.**

**Aim :**

**To write a method in C# that takes an array as an input parameter and uses two methods, one to find the largest array element and other to compute the average of array elements.**

**Algorithm :**

1. **Start Program**
2. **Declare the necessary variables.**
3. **This program takes n number of elements from the user and stores it in the arr array. To find the largest element,**
4. **the first two elements of array are checked and the largest of these two elements are placed in arr[0]**
5. **The first and third elements are checked and largest of these two elements is placed in arr[0]. This process continues until the first and last elements are checked**
6. **The largest number will be stored in the arr[0] position**
7. **Declare new variable sum as 0 and add elements in the array and store in sum**
8. **divide sum by total number of elements in array**
9. **return the results and print the result.**

**Program :**

**using System;**

**namespace CS1A**

**{**

**class ArrayLargest**

**{**

**int large(int[] a, int n)**

**{**

**int large = a[0];**

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

**{**

**if (a[i] > large)**

**large = a[i];**

**}**

**return large;**

**}**

**float avg(int[] a, int n)**

**{**

**float avg = 0;**

**for (int i = 0; i < n; i++)**

**avg += a[i];**

**avg /= n;**

**return avg;**

**}**

**public static void Main()**

**{**

**ArrayLargest obj = new ArrayLargest();**

**int n;**

**float large, avg;**

**int[] a = new int[50];**

**Console.Write("Enter the size of Array : ");**

**string s, s1;**

**s = Console.ReadLine();**

**n = Int32.Parse(s);**

**Console.WriteLine("Enter the array elements");**

**for (int i = 0; i < n; i++)**

**{**

**s1 = Console.ReadLine();**

**a[i] = Int32.Parse(s1);**

**}**

**Console.Write("");**

**large = obj.large(a, n);**

**avg = obj.avg(a, n);**

**Console.WriteLine("Largest element in the given array is {0}", large);**

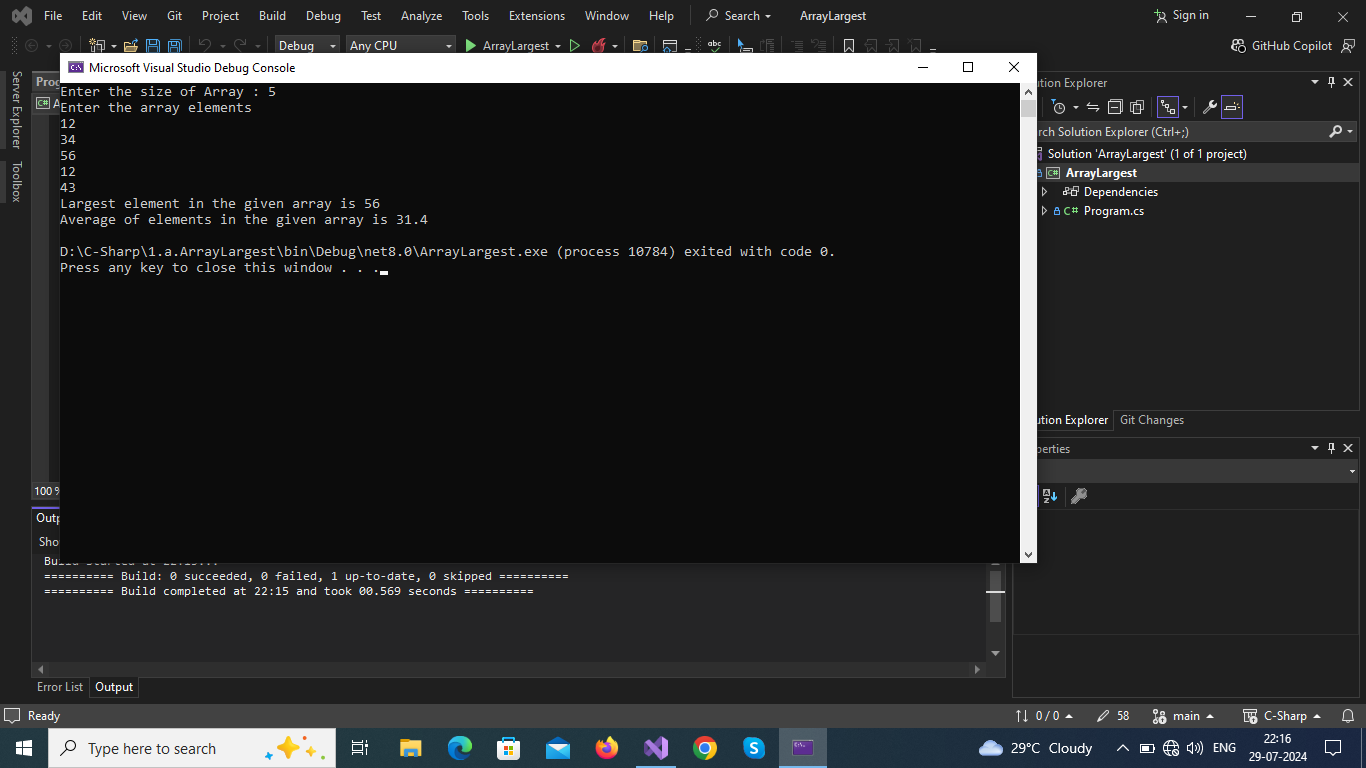
**Console.WriteLine("Average of elements in the given array is {0}", avg);**

**}**

**}**

**}**

**Output :**

****

**Result :**

**Thus the program executed successfully and output was verified.**

1. (b) Write a C# program to find the sum of all the elements present in a jagged array

Aim:

To write a C# program to find the sum of all the elements present in a jagged array

Algorithm:

1. **Start Program**
2. **Jagged Array : Collection of array is known with different size is known as jagged array**
3. **Each element of an array is accessed by a subscript operator([]).**
4. **Similiarly in jagged array each array is a element of jagged array and each array elements accessed by 2D accessing(ja[array\_index][array\_element\_index])**
5. **Declaration of jagged array:**

**Syntax**

**data\_type[][] variable\_name = new data\_type[Size\_of\_jagged\_array][]**

1. **Traverse over each elements of each array in jagged array add each of them together and store them in variable (eg;sum+=jagged\_array[i][j])**
2. **Print value of sum**
3. **Stop program**

**Program:**

using System;

namespace CS1B

{

public class SumOfElementsInJaggedArray

{

static void Main(string[] args)

{

int sum = 0;

int n, val, k;

Console.Write("Enter How Many Arrays in Jagged Array : ");

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

int[][] jagged\_array = new int[n][];

for (int i = 0; i < jagged\_array.Length; i++)

{

Console.Write("\nEnter How Many Elements in ({0}) - Array in Jagged Array : ", i + 1);

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

jagged\_array[i] = new int[k];

}

for (int i = 0; i < jagged\_array.Length; i++)

{

Console.WriteLine("\nEnter Elements of ({0}) - Array in Jagged Array ", i + 1);

for (int j = 0; j < jagged\_array[i].Length; j++)

{

Console.Write(" Enter Element {0} : ", j + 1);

jagged\_array[i][j] = Int32.Parse(Console.ReadLine());

}

}

for (int i = 0; i < jagged\_array.Length; i++)

{

val = 0;

for (int j = 0; j < jagged\_array[i].Length; j++)

{

val += jagged\_array[i][j];

}

Console.WriteLine("\nThe Sum of array ({0}) = {1}", i + 1, val);

sum += val;

}

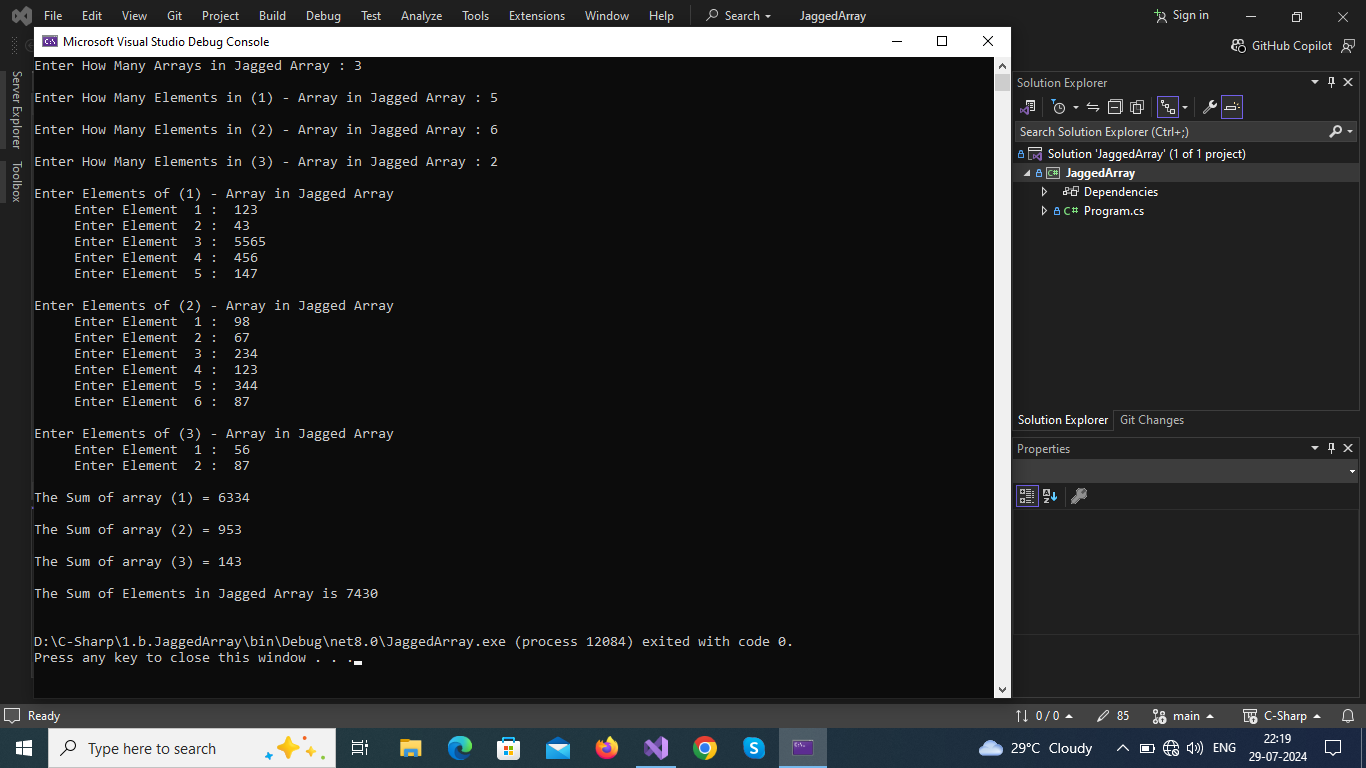
Console.WriteLine("\nThe Sum of Elements in Jagged Array is {0} \n", sum);

}

}

}

Output:



**Result :**

**Thus the program executed successfully and output was verified.**

2. (a) Write a program using while loop to reverse the digits of the number.

**Aim:**

To write a program using while loop to reverse the digits of the number

.

**Algorithm :**

1. **Start Program**
2. **Get Input from user & store in variable**
3. **Declare a variable to store reversed number.**
4. **Using following logics we can perform reverse operation.**
   1. **To find last digit n%10;**
   2. **To find remaining part n/10;**
5. **To get reverse (use loop)**
   1. **reverse = reverve\*10 + lastdigit;**
   2. **number = number/10;**
6. **Print value of reverse**
7. **Stop program**

**Program :**

**using System;**

**namespace CS2A**

**{**

**class NumberReverse**

**{**

**public static void Main(string[] args)**

**{**

**int n;**

**Console.Write("Enter an Integer : ");**

**n = Int32.Parse(Console.ReadLine());**

**int num = n;**

**int rev = 0;**

**while (n != 0)**

**{**

**rev = rev \* 10 + (n % 10);**

**n /= 10;**

**}**

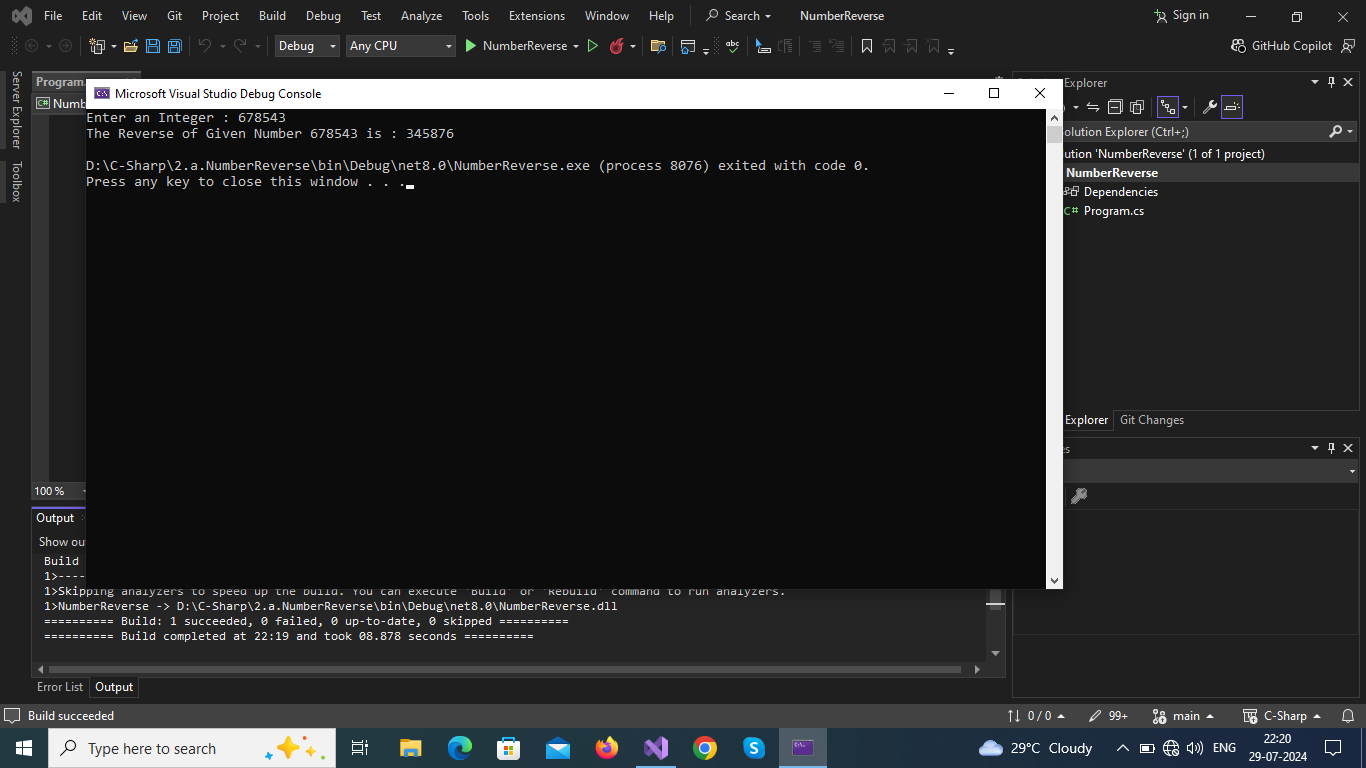
**Console.WriteLine("The Reverse of Given Number {0} is : {1} ", num, rev);**

**}**

**}**

**}**

**Output :**

****

**Result :**

**Thus the program executed successfully and output was verified.**

2. (b) Write a C# program that generates the Fibonacci series and prime numbers for the given number.

**Aim:**

To write a program that generates the Fibonacci series and prime numbers for the given number.

.

**Algorithm :**

1. **Start Program**
2. **Write two methods :** 
   1. **void fibo(param n) to generate Fibonacci series**

**logic => n1=0,n2=0**

**n3=n1+n2**

**n1=n2**

**n2=n3**

* 1. **void fibo(param n) to generate Prime Number**

**logic => use two loops to check whether the number is prime or not if prime print it.**

1. **Print value of results**
2. **Stop program**

**Program :**

**using System;**

**namespace CS2B**

**{**

**class FiboPrime**

**{**

**void Prime(int n)**

**{**

**int c, k = 0;**

**for (int i = 1; i < n; k++)**

**{**

**c = 0;**

**int j = 1;**

**for (j = 1; j <= k; j++)**

**{**

**if (k % j == 0)**

**{**

**c++;**

**}**

**}**

**if (c == 2)**

**{**

**Console.WriteLine(k);**

**i++;**

**}**

**}**

**}**

**void Fibo(int n)**

**{**

**ulong n1 = 0, n2 = 1, n3;**

**Console.WriteLine(" {0}\n {1}", n1, n2);**

**for (int i = 2; i <= n; i++)**

**{**

**n3 = n1 + n2;**

**Console.WriteLine(" " + n3);**

**n1 = n2;**

**n2 = n3;**

**}**

**}**

**public static void Main(string[] args)**

**{**

**int n;**

**Console.Write("Enter Interger to generate Fibonacci Series or Prime Number : ");**

**n = Int32.Parse(Console.ReadLine());**

**Console.Write("1.Fibonacci Series\n2.Prime Numbers\nChoose Option : ");**

**int ch = Int32.Parse(Console.ReadLine());**

**FiboPrime fb = new FiboPrime();**

**switch (ch)**

**{**

**case 1:**

**Console.WriteLine("\n{0} Fibonacci Terms ", n);**

**fb.Fibo(n);**

**break;**

**case 2:**

**Console.WriteLine("\n{0} Prime Numbers ", n);**

**fb.Prime(n);**

**break;**

**default:**

**Console.WriteLine("Wrong Choice ");**

**Main(args);**

**break;**

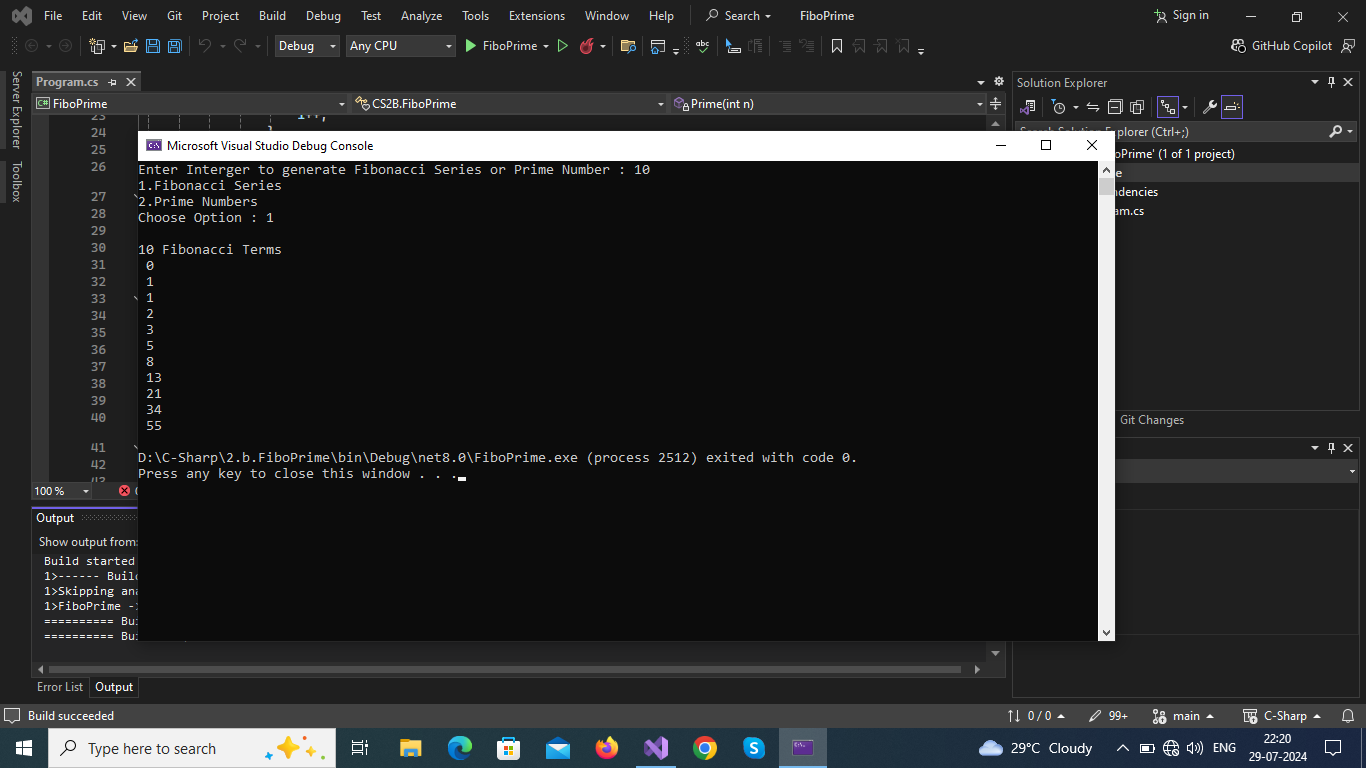
**}**

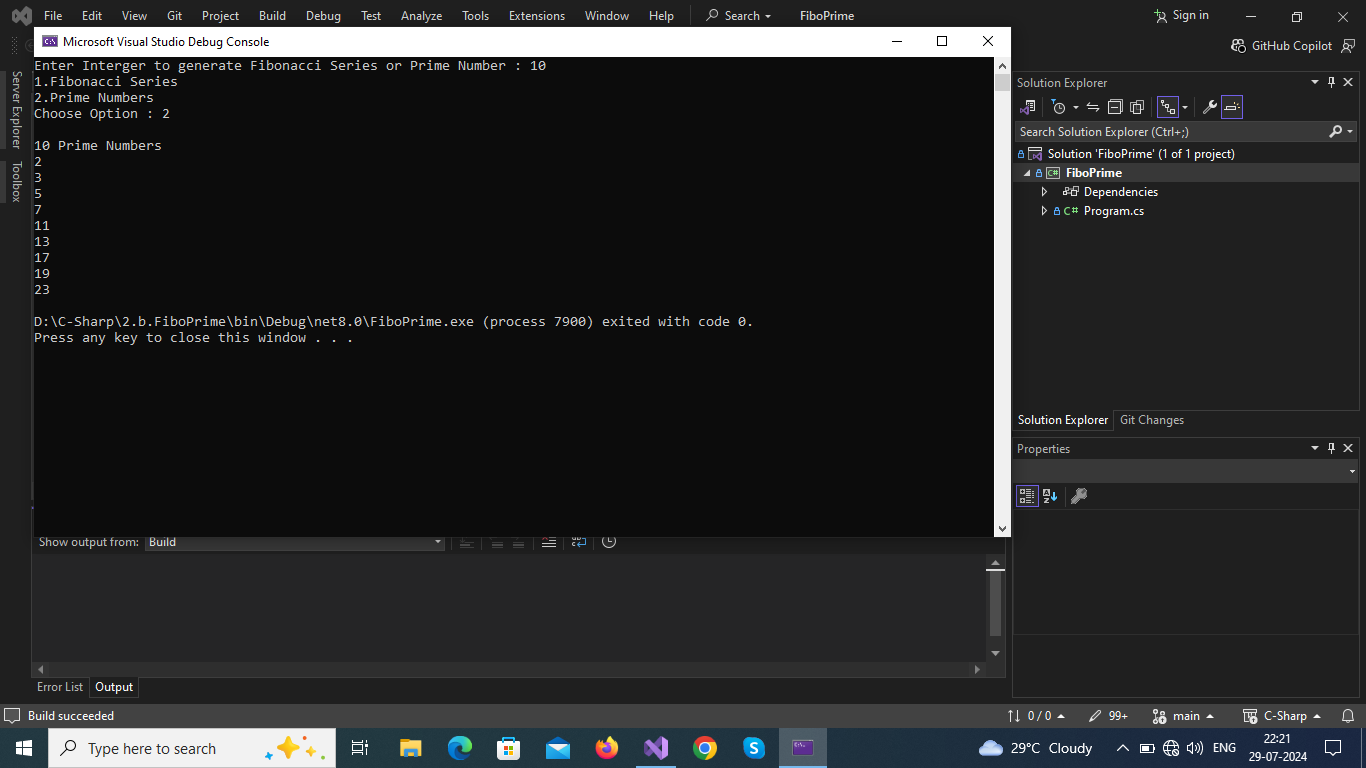
**}**

**}**

**}**

**Output :**

****

****

**Result :**

**Thus the program executed successfully and output was verified.**

3.Write a C# program to create a Stack class and appropriate methods to

implement the stack operations.

Aim :

To Write a C# program to create a Stack class and appropriate methods to implement the stack operations.

**Algorithm:**

1.Start the program

2.Create a class with Stack with its basic function:

1.push(data)

2.pop()

3.peek()

4.Display()

3.initialize stack total size as you own.

4.Make a default constructor to top as -1

5.Push(data):

check stack is full or not

if full:

give message

else:

push a element to stack and increase top by 1

6.Pop():

check stack is empty or not

if empty:

give message

else :

return a element on top and reduce top by 1

7.Peek():

like Pop()

but don't reduce top

8.Display():

use loop to display all elements.

9.Get user input as per need and use switch to select appropriate function.

10.stop the program

Program :

using System;

namespace CS3

{

public class Stack

{

static readonly int MAX = 100;

int[] stack = new int[MAX];

int top;

bool isStackEmpty()

{

return (top < 0);

}

public Stack()

{

top -= 1;

}

internal bool Push(int data)

{

if (top >= MAX)

{

Console.WriteLine("\nStack Overflow");

return false;

}

else

{

stack[++top] = data;

Console.WriteLine("\n{0} --> pushed into stack ", data);

return true;

}

}

internal int Pop()

{

if (isStackEmpty())

{

Console.WriteLine("\nStack Underflow");

return 0;

}

else

{

int value = stack[top--];

return value;

}

}

internal void Peek()

{

if (isStackEmpty ())

{

Console.WriteLine("\nStack Underflow");

return;

}

else

{

Console.WriteLine("\nThe topmost element of the stack - {0} .\n", stack[top]);

return;

}

}

internal void PrintStack()

{

if (isStackEmpty ())

{

Console.WriteLine("\nStack Underflow");

return;

}

else

{

Console.WriteLine("\nElements in Stack are ...");

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

{

Console.WriteLine(stack[i] + "\n");

}

}

}

internal int Menu()

{

Console.WriteLine("\nStack Operations..\n1.Push\n2.Pop\n3.Peek\n4.Allelements\n5.exit");

Console.Write("Enter Choice : ");

return (Int32.Parse(Console.ReadLine()));

}

public static void Main(string[] args)

{

Stack st = new Stack();

int ch = 1;

while (ch != 0)

{

ch = st.Menu();

Console.Clear();

switch (ch)

{

case 1:

Console.Write("\nEnter Data to push into stack : ");

int data = Int32.Parse(Console.ReadLine());

st.Push(data);

break;

case 2:

Console.WriteLine("\n{0} <-- Popped from stack", st.Pop());

break;

case 3:

st.Peek();

break;

case 4:

st.PrintStack();

break;

case 5:

Console.WriteLine("\nThank You ");

ch = 0;

break;

default:

Main(args);

break;

}

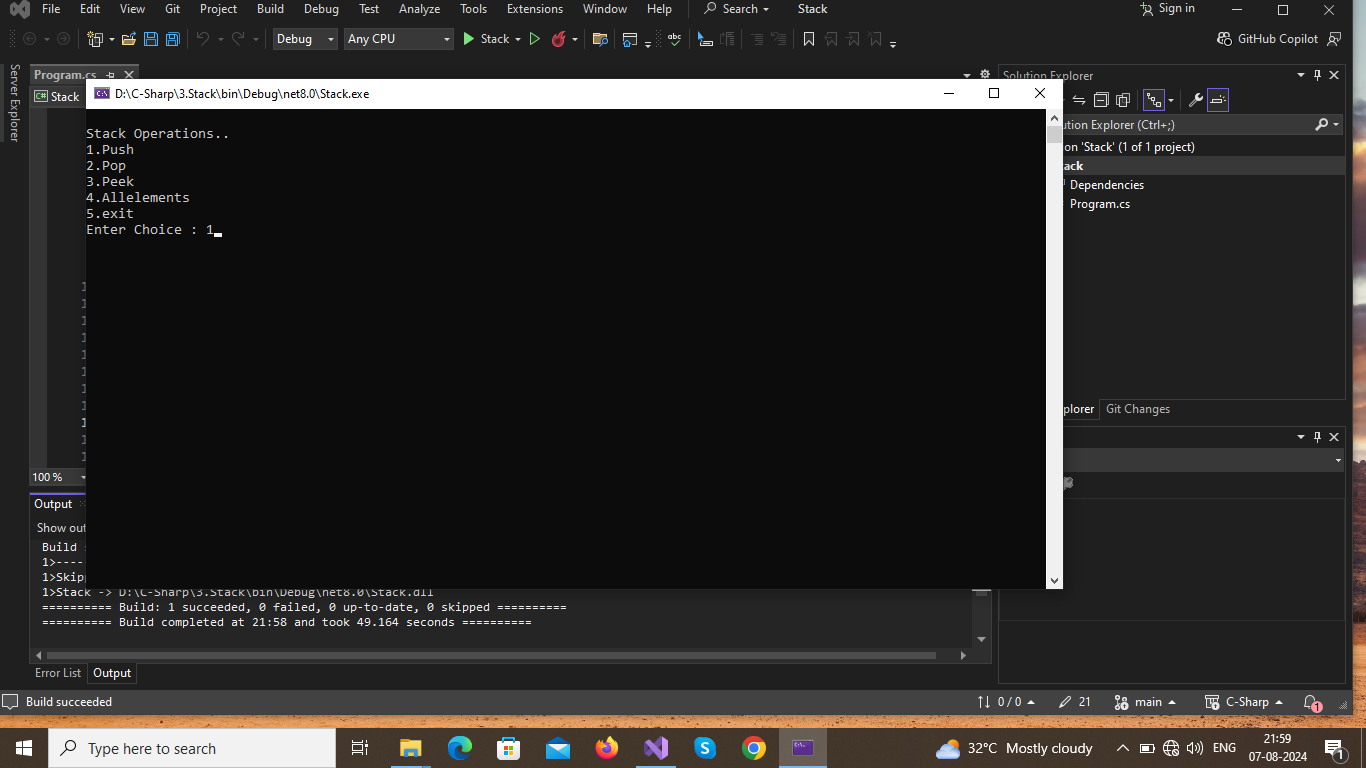
}

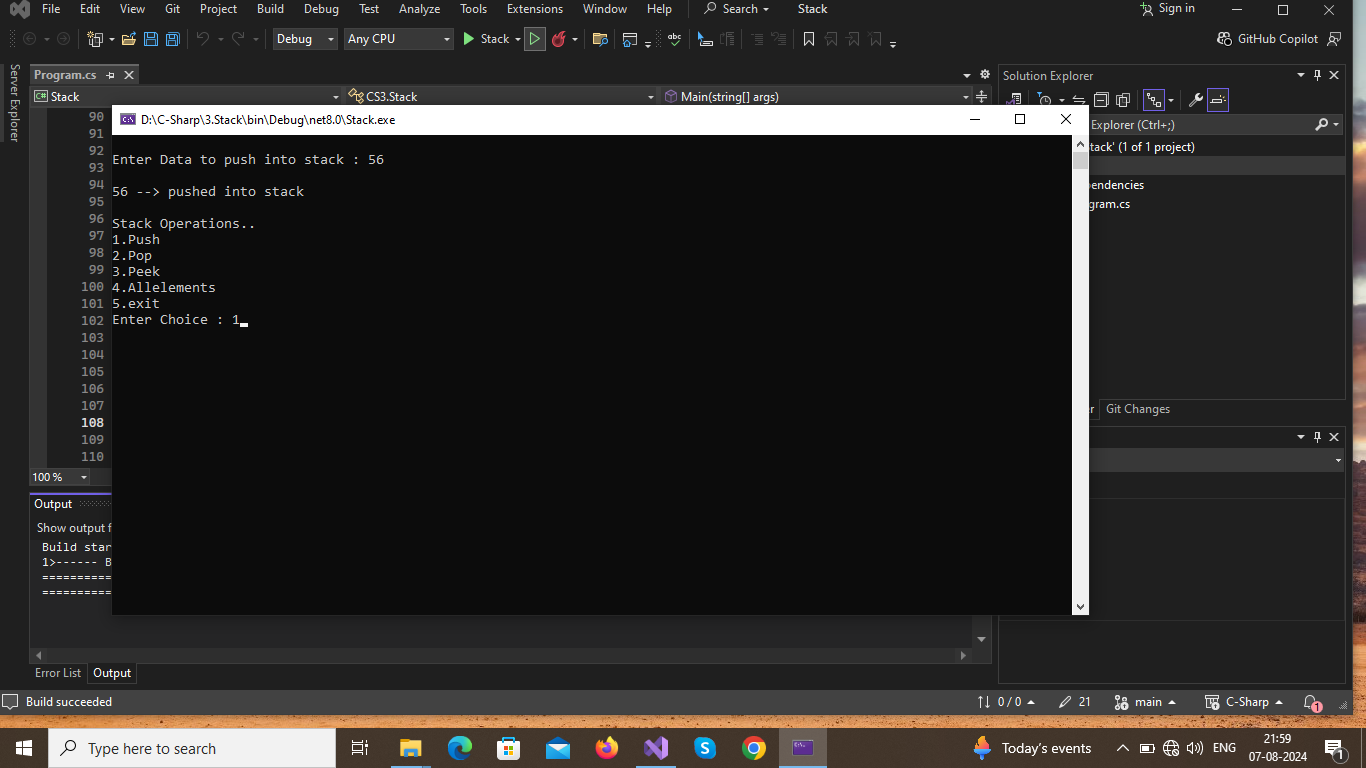
}

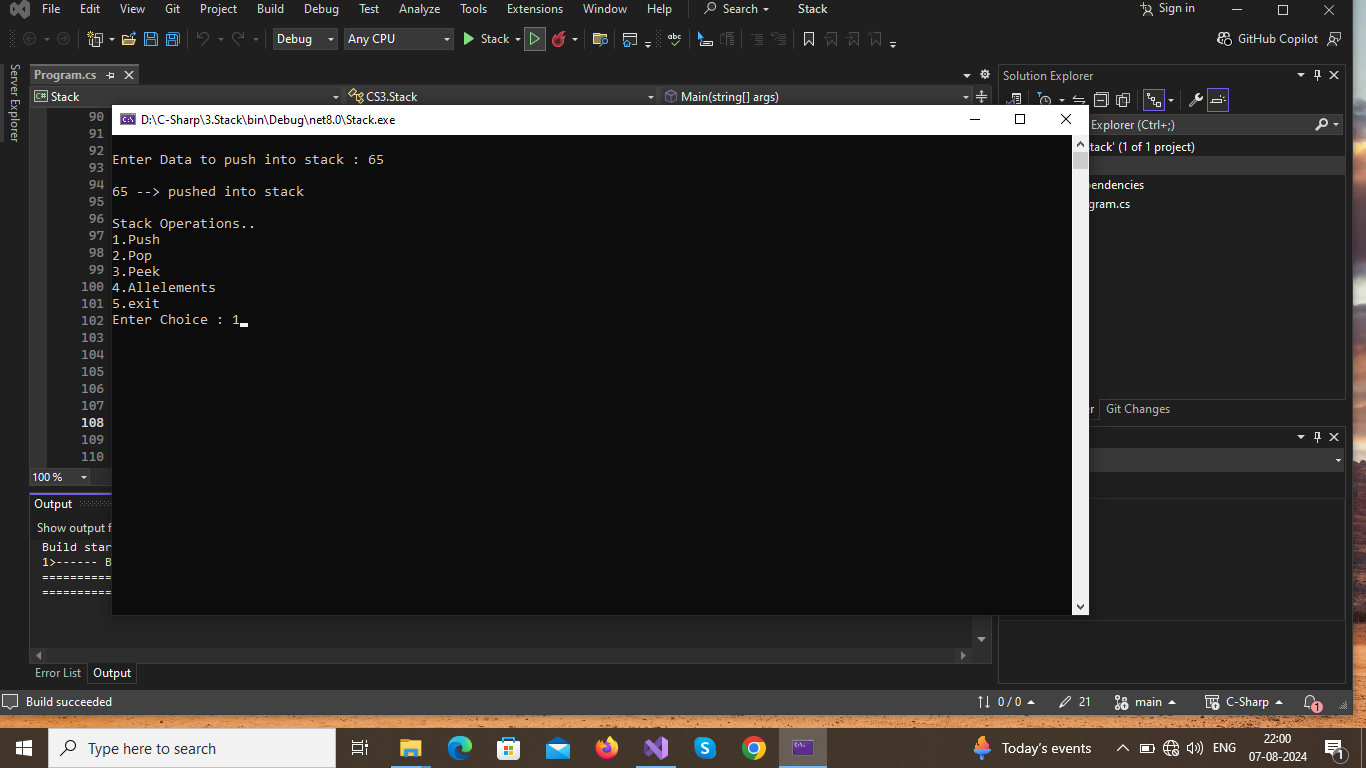
}

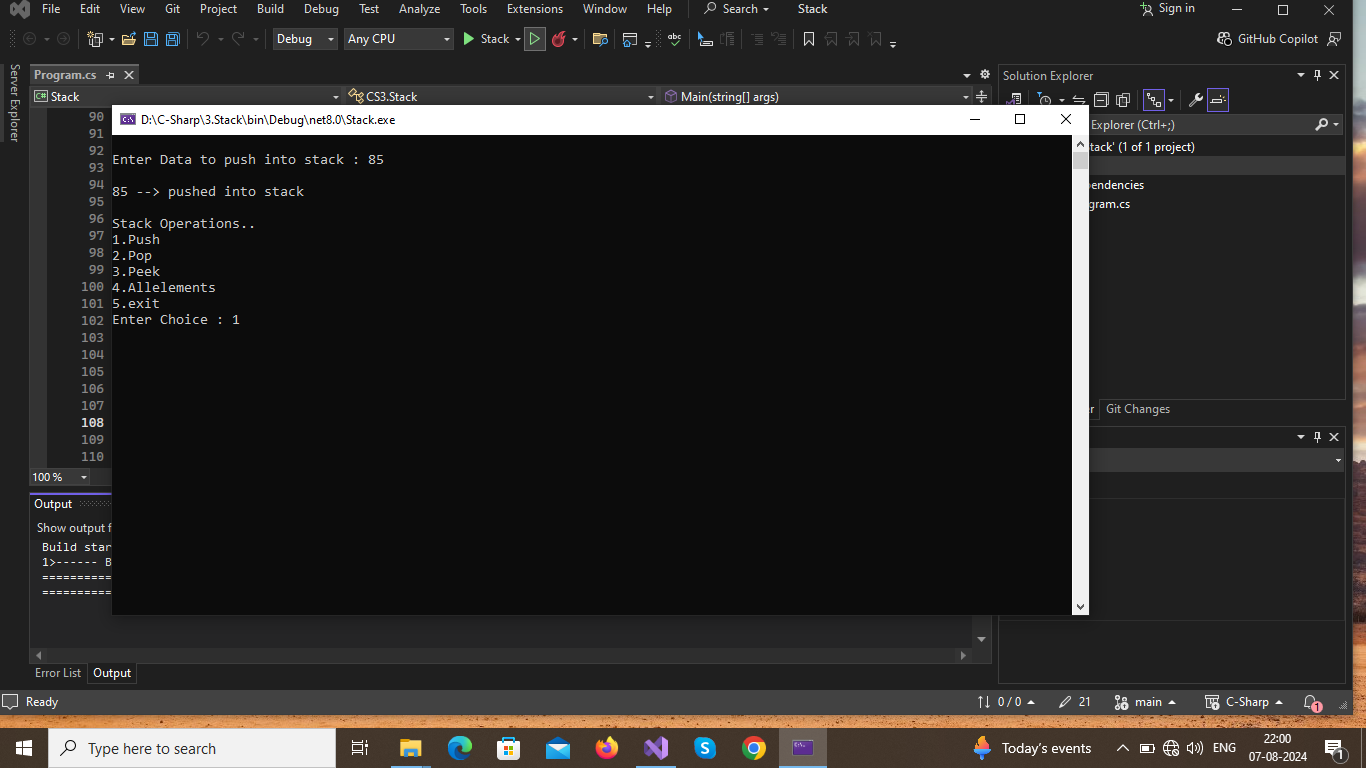
}

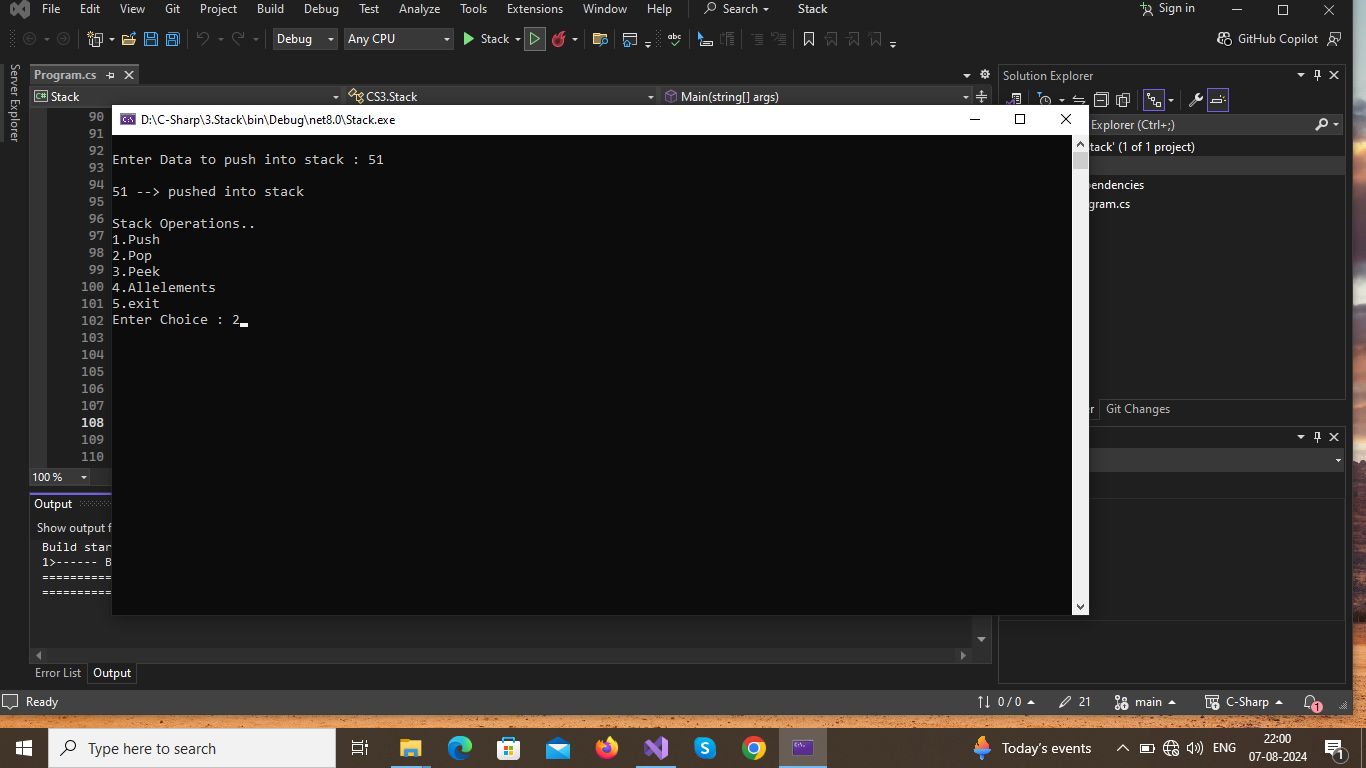
Output :

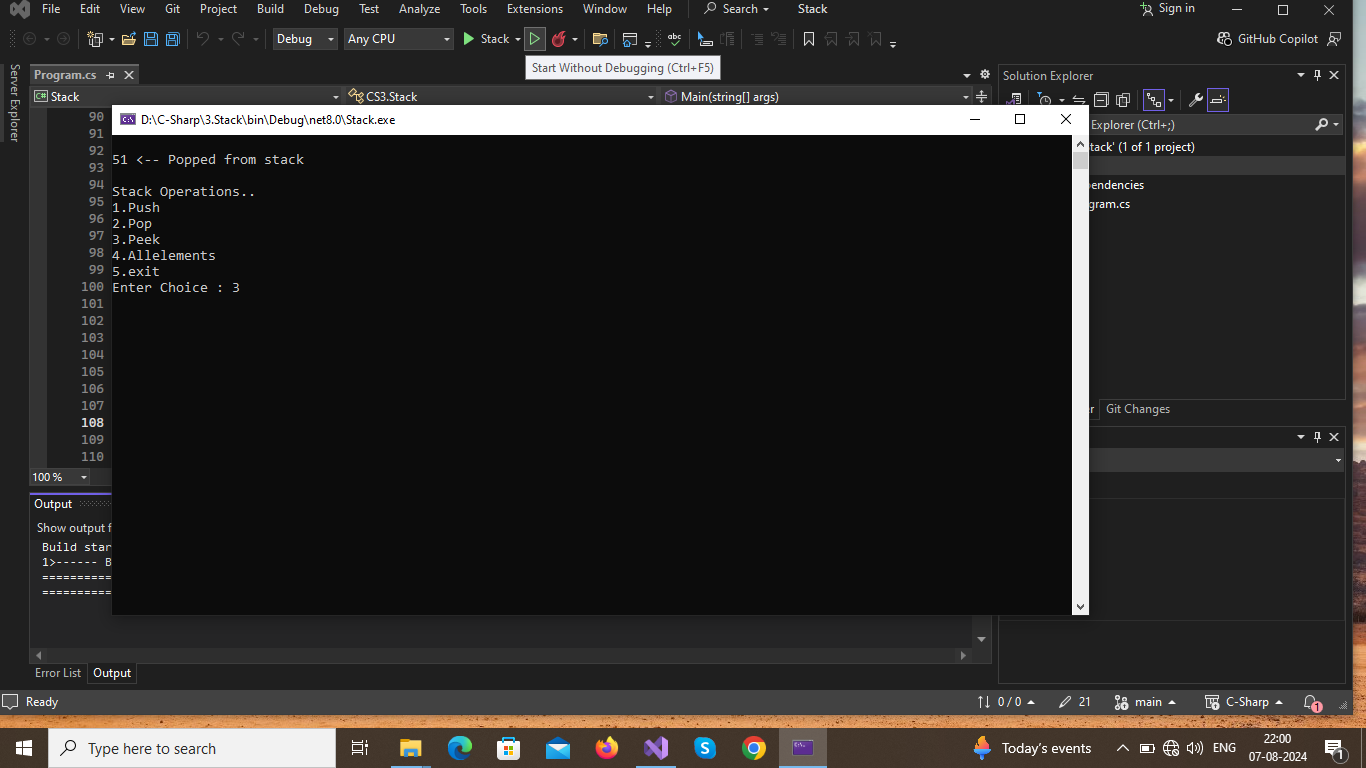


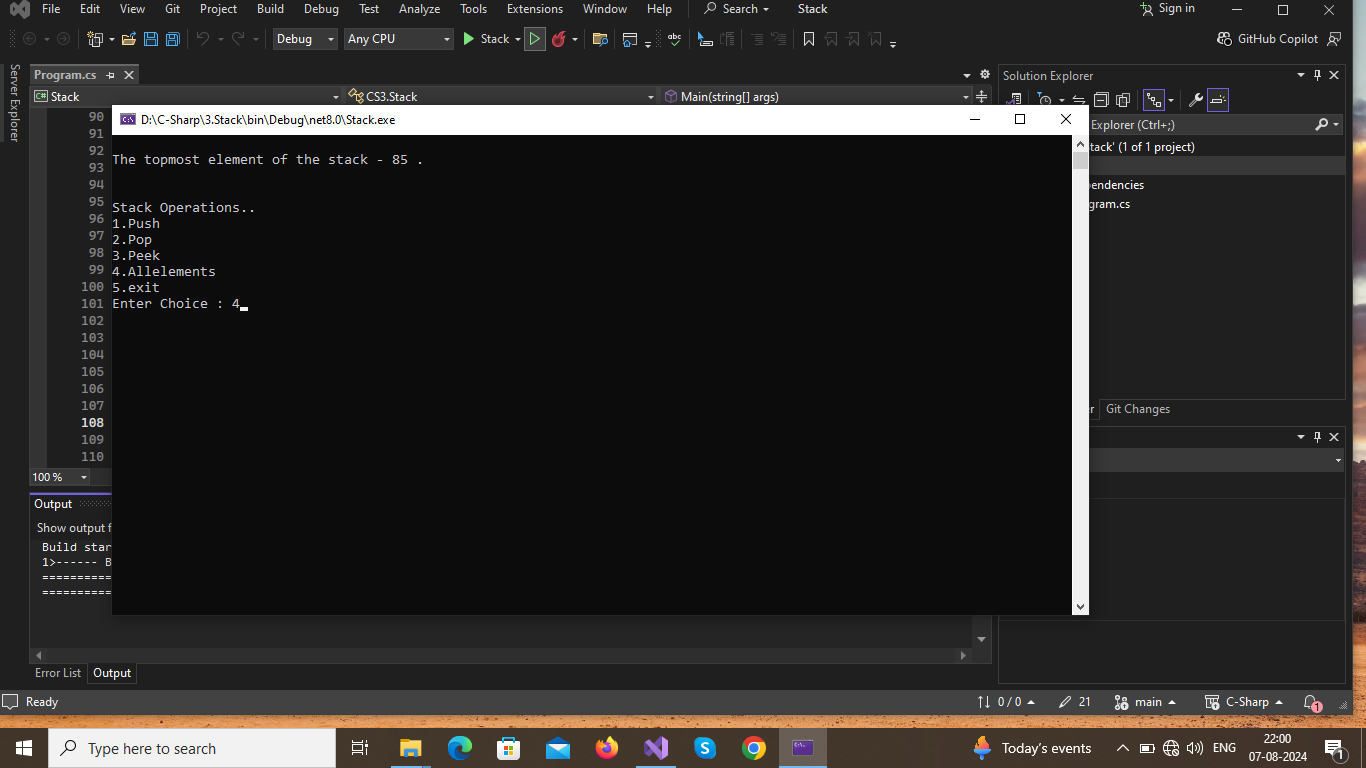


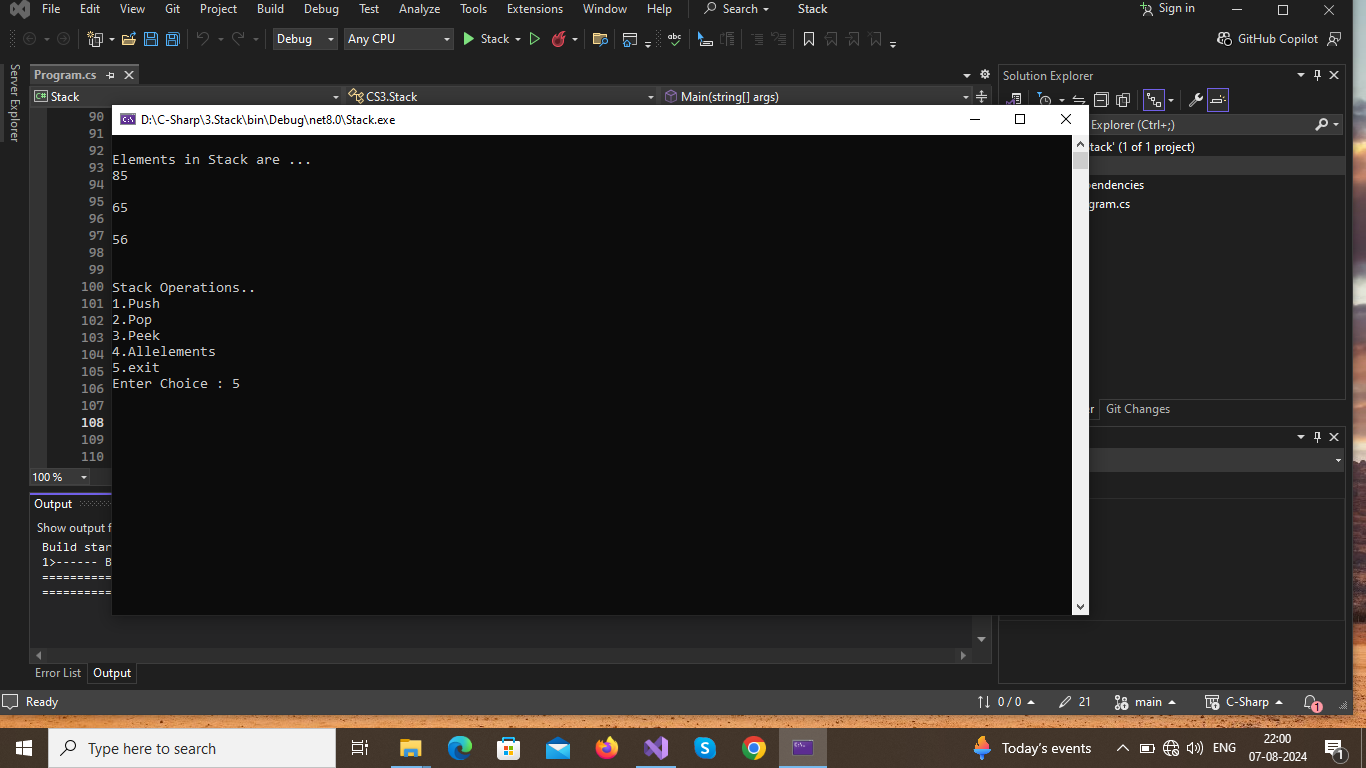














**Result :**

**Thus the program executed successfully and output was verified.**

4.Store a string "123456789" in a string variable and use it to display the following pattern.

1

232

34543

4567654

567898765

Aim :

To store a string "123456789" in a string variable and use it to display the following pattern.

1

232

34543

4567654

567898765

Algorithm :

1.Start The Program  
2.Store value as string in variable as S="123456789"  
3.initialize 3 variables eg)i,j,k as 0 (i=0,j=0,k=0)  
4.Run two loop to iterate over row and column  
5.First loop should run for half of string length(row loop)   
6.Second loop should run for twice of the string length(columb loop)  
7.print value of string based on below condition:  
if(col<=2\*row):  
print value of string  
to get value(s[j]-'0')  
store value in k   
else:  
--k  
if(k>row):  
print(k)  
8.insert new line after each row completion  
9.stop the program

Program :

using System;  
namespace CS4  
{  
public class StringPattern {  
 public static void Main()  
 {  
 string s = "123456789";  
 int k = 1;  
 for (int i = 0; i < (s.Length + 1) / 2; i++)  
 {  
for (int j = i; j < 2\*s.Length; j++)  
{  
 if (j <= 2 \* i)  
{  
 k = s[j] - '0';   
 Console.Write(k + " ");  
}  
else  
{  
 --k;  
 if (k > i)  
 {  
 Console.Write(k + " ");  
 }  
 }   
 }  
Console.WriteLine();

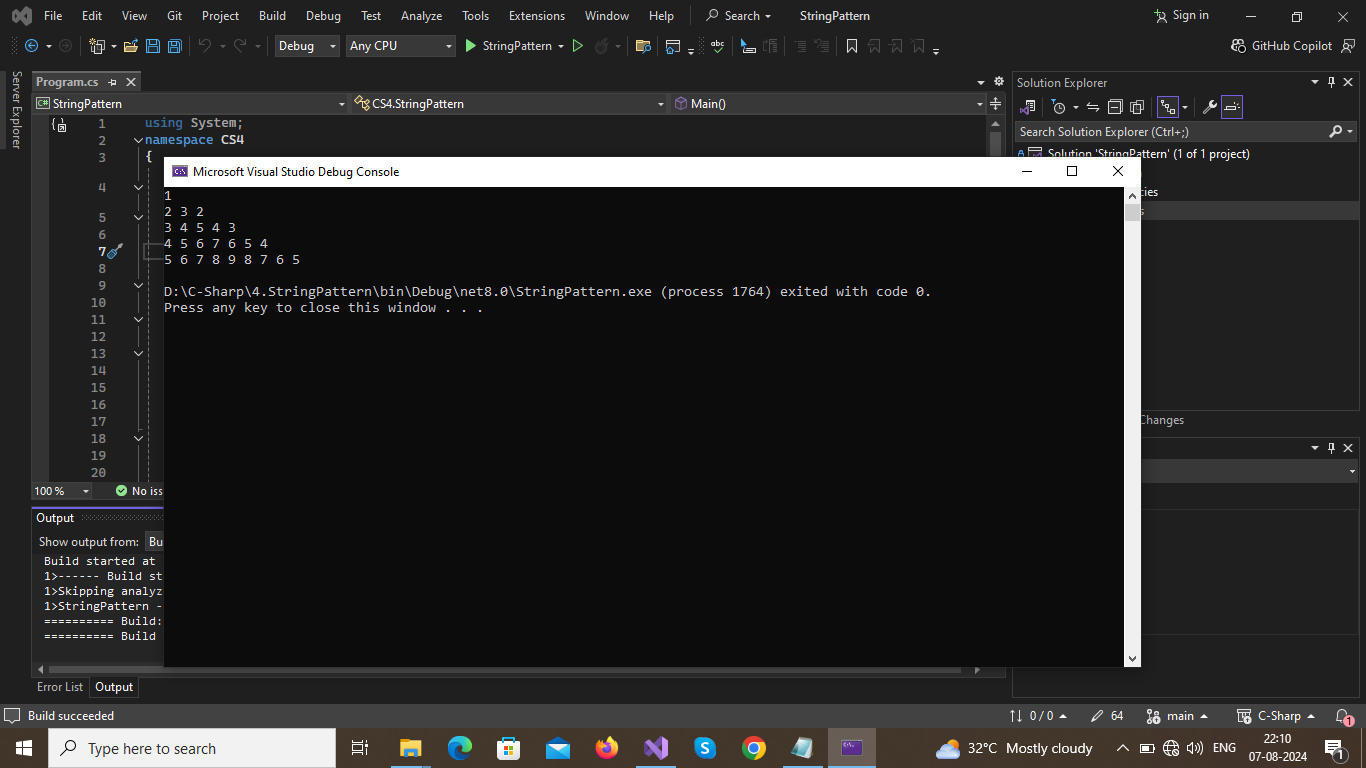
}

}

}

}

Output :



**Result :**

**Thus the program executed successfully and output was verified.**

5.Develop a C# program with more than one class and illustrates the concept of constructor overloading and method overloading.

Aim :

To develop a C# program with more than one class and illustrates the concept of constructor overloading and method overloading.

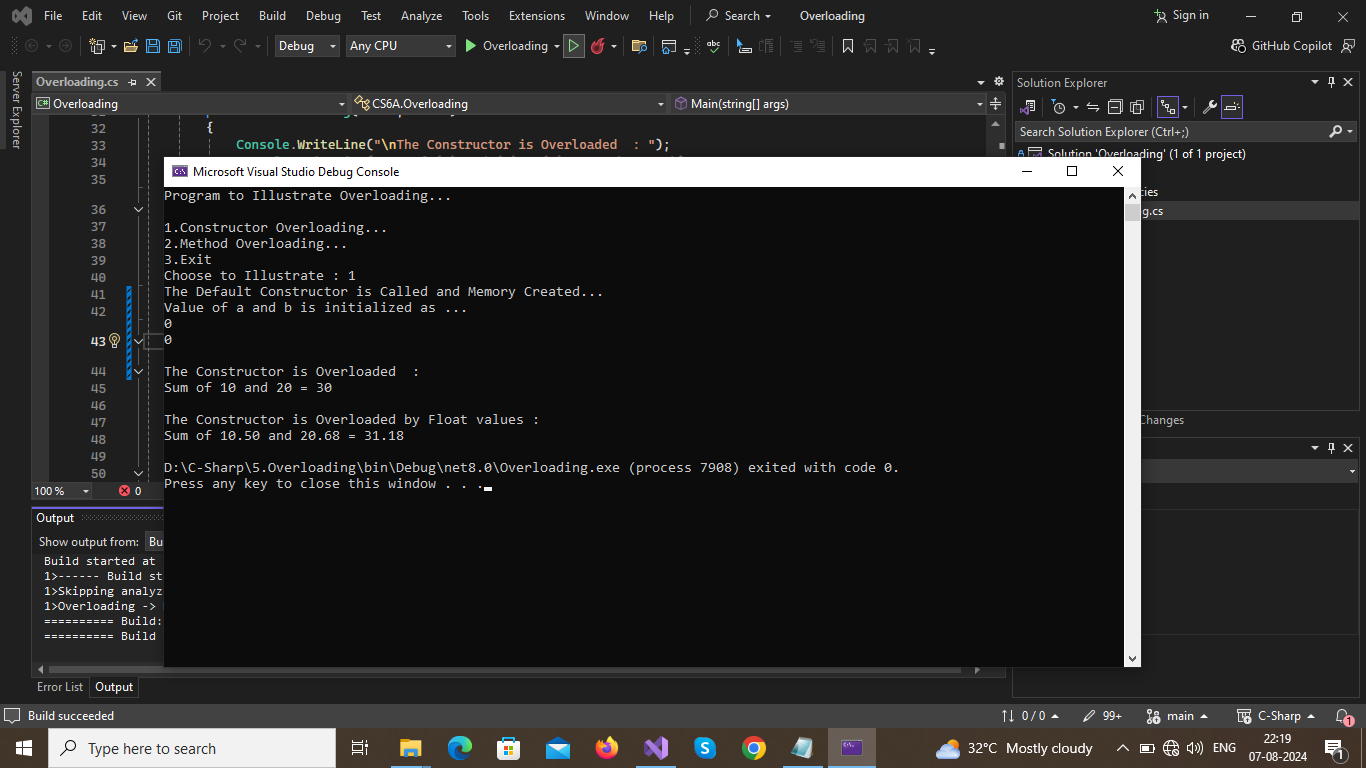
Algorithm :

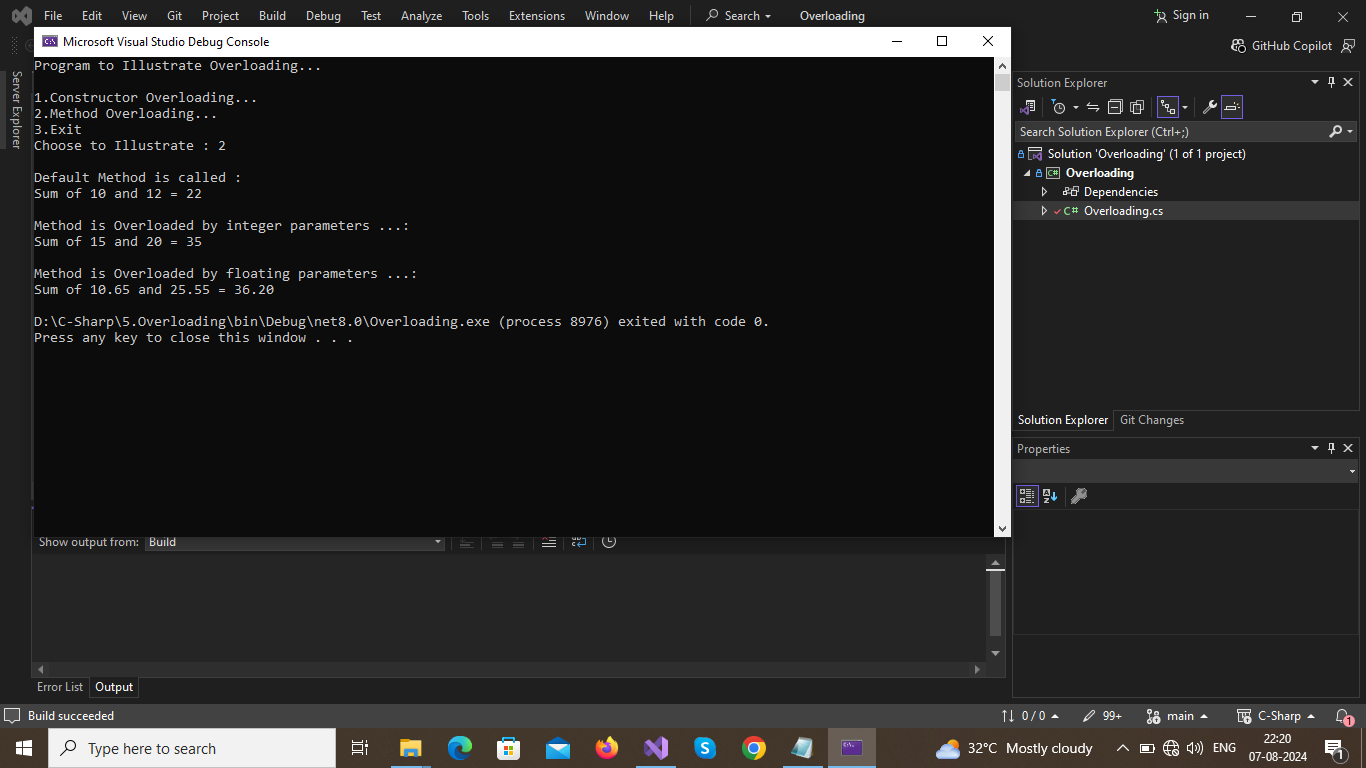
1.Start the Program  
2.Create two class one for illustrate Constructor overloading and another for Method  
Overloading  
3.Create 3 or More Method with same name and different parameter count or different  
parameter type in Method Overloading class.  
4.Likely For Constructor but name of method should be same as Class Name in Constructor  
overload class.  
5.Create Objects of both class and observe the result  
6.Stop the program

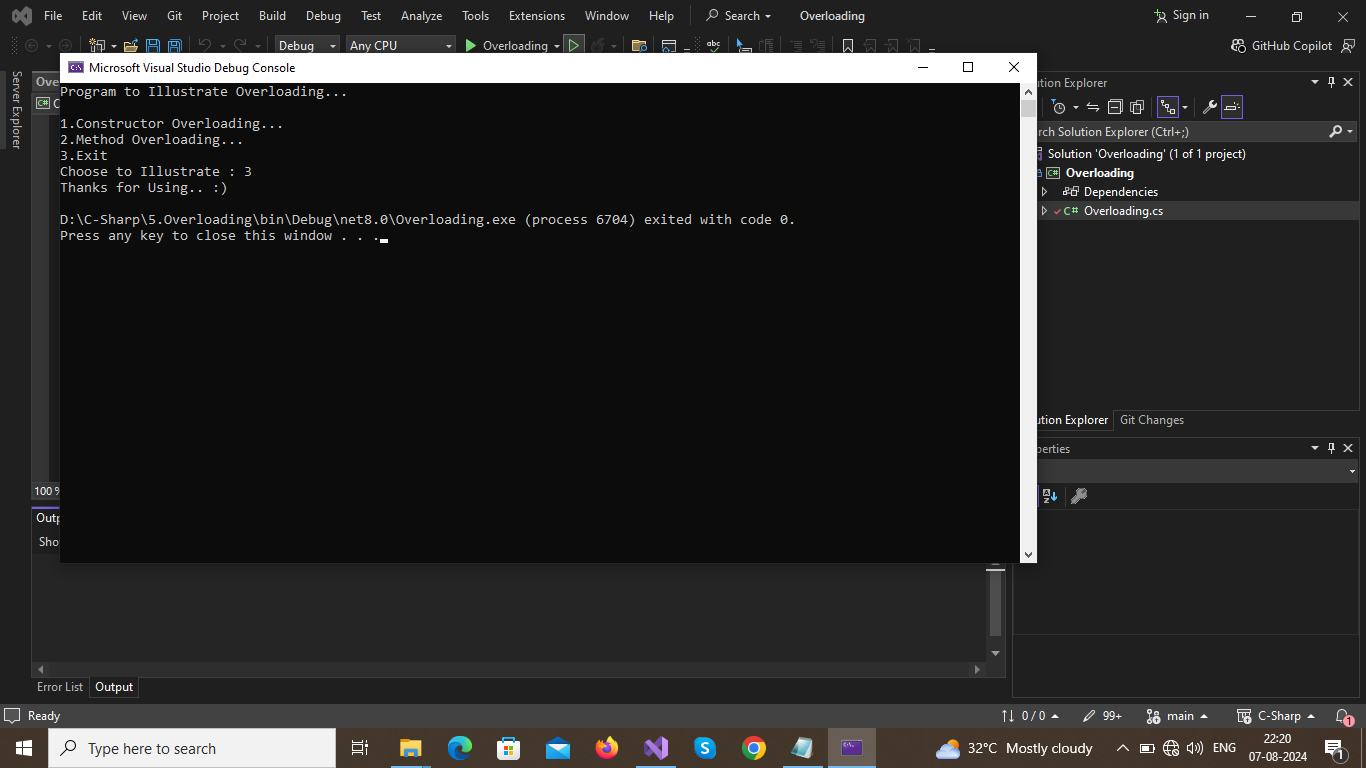
Program :

﻿using System;  
namespace CS6A  
{  
public class MOverloading {  
public void sum()  
{  
Console.WriteLine("\nDefault Method is called : ");  
int a = 10, b = 12;  
Console.WriteLine("Sum of {0} and {1} = {2} ", a, b, a + b);  
}  
public void sum(int a, int b)  
{  
Console.WriteLine("\nMethod is Overloaded by integer parameters ...: ");  
Console.WriteLine("Sum of {0} and {1} = {2} ", a, b, a + b);  
}  
public void sum(float a, float b)  
{  
Console.WriteLine("\nMethod is Overloaded by floating parameters ...: ");  
Console.WriteLine("Sum of {0:N2} and {1:N2} = {2:N2} ", a, b, a + b);  
}  
}  
public class COverloading  
{  
public COverloading()  
{  
int a = 0;  
int b = 0;  
Console.WriteLine("The Default Constructor is Called and Memory Created... ");  
Console.WriteLine("Value of a and b is initialized as ... \n{0} \n{1}", a, b);  
}  
public COverloading(int a, int b)  
{  
Console.WriteLine("\nThe Constructor is Overloaded : ");  
Console.WriteLine("Sum of {0} and {1} = {2} ", a, b, a + b);  
}  
public COverloading(float a, double b)  
{  
Console.WriteLine("\nThe Constructor is Overloaded by Float values : ");  
Console.WriteLine("Sum of {0:N2} and {1:N2} = {2:N2} ", a, b, a + b);  
}  
  
}  
public class Overloading {  
public static void Main(string[] args)  
{  
int ch;  
Console.WriteLine("Program to Illustrate Overloading...");  
Console.Write("\n1.Constructor Overloading...\n2.Method Overloading...\n3.Exit\nChoose to Illustrate : ");  
ch = Int32.Parse(Console.ReadLine());  
switch (ch)  
{  
case 1:  
COverloading co = new COverloading();  
COverloading co1 = new COverloading(10, 20);  
COverloading co2 = new COverloading(10.5f, 20.676f);  
break;  
case 2:  
MOverloading mo = new MOverloading();  
mo.sum();  
mo.sum(15, 20);  
mo.sum(10.65f, 25.55f);  
break;  
case 3:  
Console.WriteLine("Thanks for Using.. :)");  
return;  
default:  
Console.WriteLine("Wrong Choice ...");  
Main(args);  
break;  
}  
}  
}  
  
}

Output :







**Result :**

**Thus the program executed successfully and output was verified.**

**6.(a) Demonstrate use of virtual and override key words in C# with a simple program**

Aim:

To demonstrate use of virtual and override key words in C# with a simple program

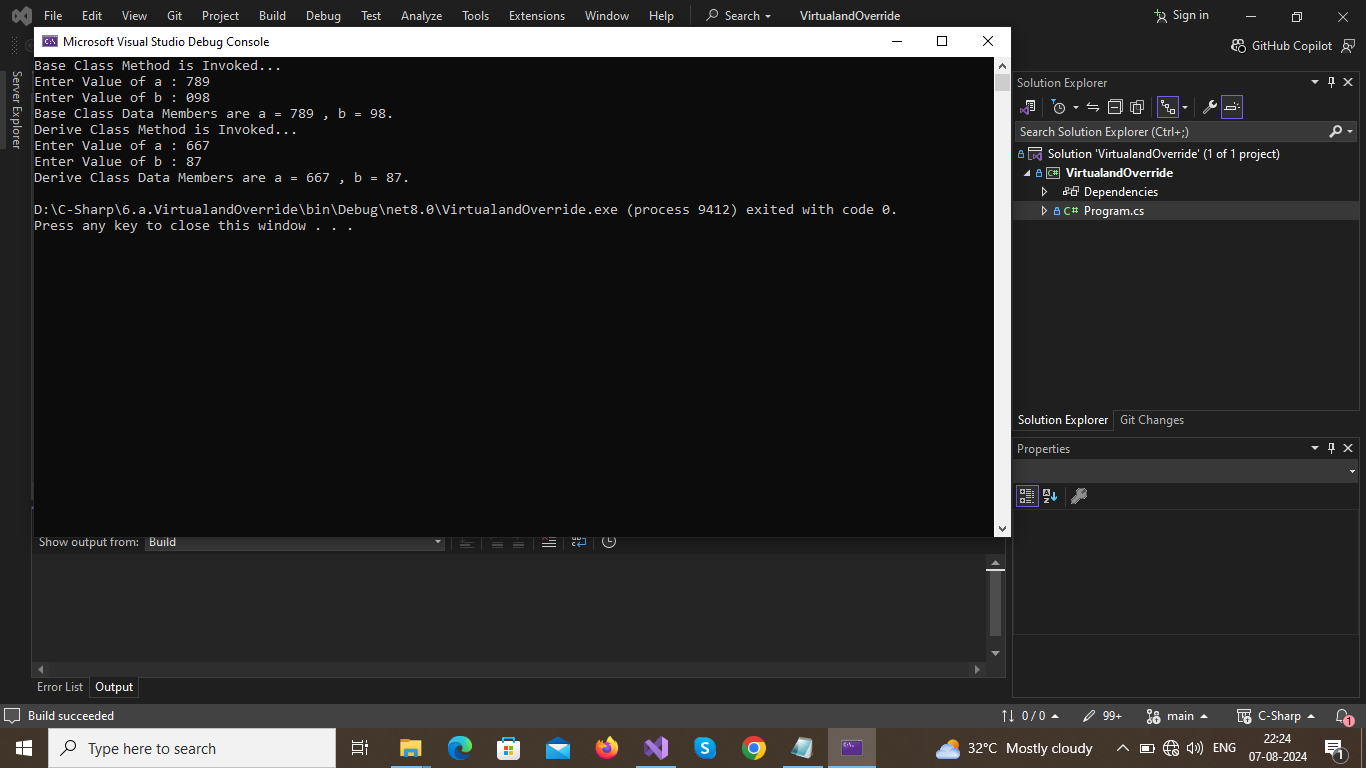
Algorithm :

1.Start the Program  
2.Create a Base class with virtual methods.  
3.Derive a class from Base class . Create a Override methods as per Base class Methods.  
4.Create a object for Base Class.  
5.Access Base class methods.  
6.Convert object to Derive Class(Override)  
7.Acess same method.  
8.Observe the difference.  
9.Print Output

.  
Program :

﻿using System;  
namespace CS5A  
{  
public class Virtual  
{  
int a, b;  
public virtual void getValue()  
{  
Console.WriteLine("Base Class Method is Invoked...");  
Console.Write("Enter Value of a : ");  
a = Int32.Parse(Console.ReadLine());  
Console.Write("Enter Value of b : ");  
b = Int32.Parse(Console.ReadLine());  
}  
public virtual void disp()  
{  
Console.WriteLine("Base Class Data Members are a = {0} , b = {1}.", a, b);  
}  
}  
public class VirtualExt : Virtual  
{  
int a, b;  
public override void getValue()  
{  
Console.WriteLine("Derive Class Method is Invoked...");  
Console.Write("Enter Value of a : ");  
a = Int32.Parse(Console.ReadLine());  
Console.Write("Enter Value of b : ");  
b = Int32.Parse(Console.ReadLine());  
}  
public override void disp()  
{  
Console.WriteLine("Derive Class Data Members are a = {0} , b = {1}.", a, b);  
}  
}  
public class VirtualOverride  
{  
public static void Main()  
{  
Virtual obj = new Virtual();  
obj.getValue();  
obj.disp();  
//Overriding Derive class method and Base class is Overriden.  
obj = new VirtualExt();  
obj.getValue();  
obj.disp();  
}  
}  
}

**Output :**

****

**Result :**

**Thus the program executed successfully and output was verified.**

**6.(b) Write a program to demonstrate abstract class and abstract methods in C#.**

Aim :

To write a program to demonstrate abstract class and abstract methods in C#.

Algorithm :

1.Start Program  
2.Create a abstract class and define a method as abstract.  
3.Derive another class and write code for method which is in base class.  
4.We can't Create a object for Abstract class directly. So we have to create object for  
derived class and access the method in abstract class.  
5.Observe the results.  
6.Stop the program.

Program :

﻿using System;  
namespace CS6B  
{  
public abstract class Abstract  
{  
public int[] a;  
public abstract void getValue(int n);  
public void show()  
{  
Console.WriteLine("Array Created and Elements are ... ");  
for (int i = 0; i < a.Length; i++)  
{  
Console.Write(" Element {0} : {1}\n", i + 1, a[i]);  
}  
}  
public class AbstractImp : Abstract  
{  
public override void getValue(int n)  
{  
a = new int[n];  
for (int i = 0; i < n; i++)  
{  
Console.Write("Enter Element {0} : ", i + 1);  
a[i] = Int32.Parse(Console.ReadLine());  
}  
}  
}  
public static void Main(string[] args)  
{  
AbstractImp ab = new AbstractImp();  
Console.WriteLine("Implementation of Array using Abstract Class ... ");  
Console.Write("Enter Size of Array : ");  
ab.getValue(Int32.Parse(Console.ReadLine()));  
ab.show();  
}  
}  
}

Output:



**Result :**

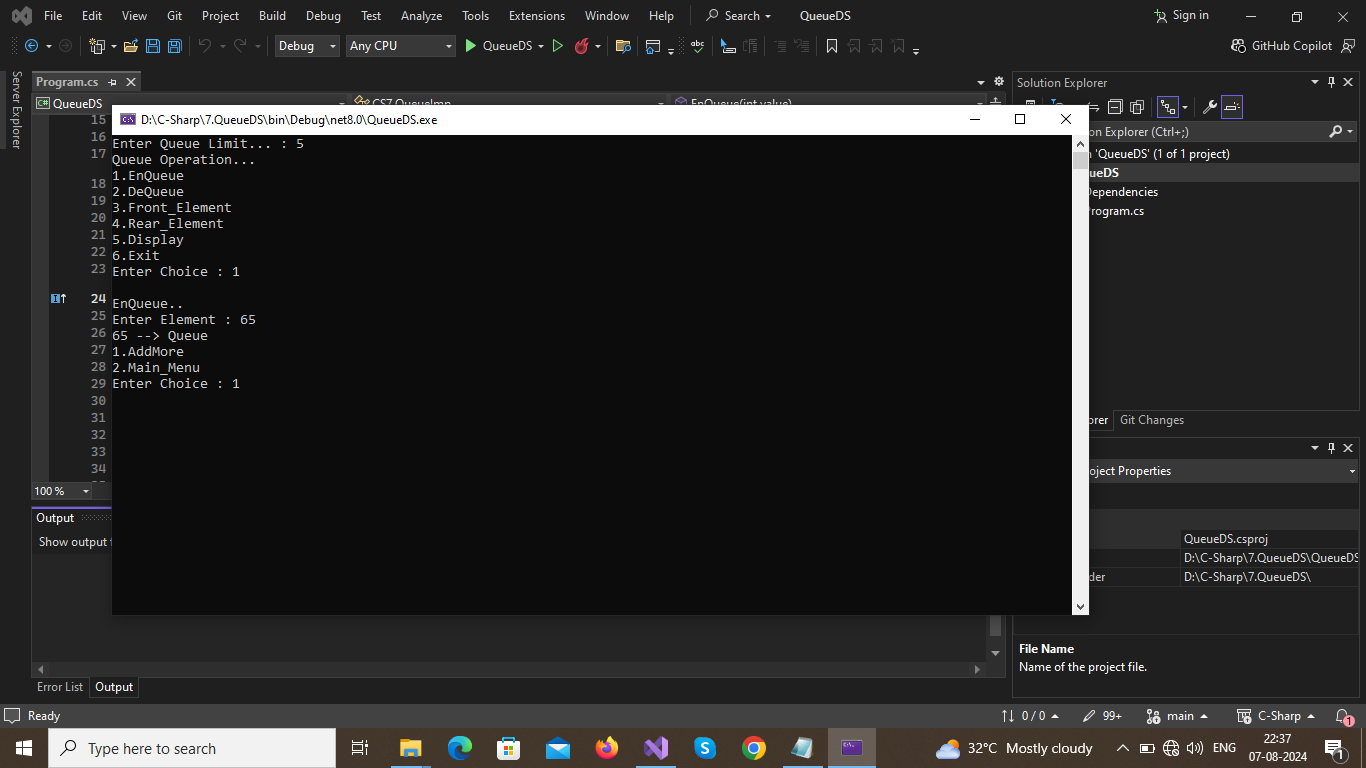
**Thus the program executed successfully and output was verified.**

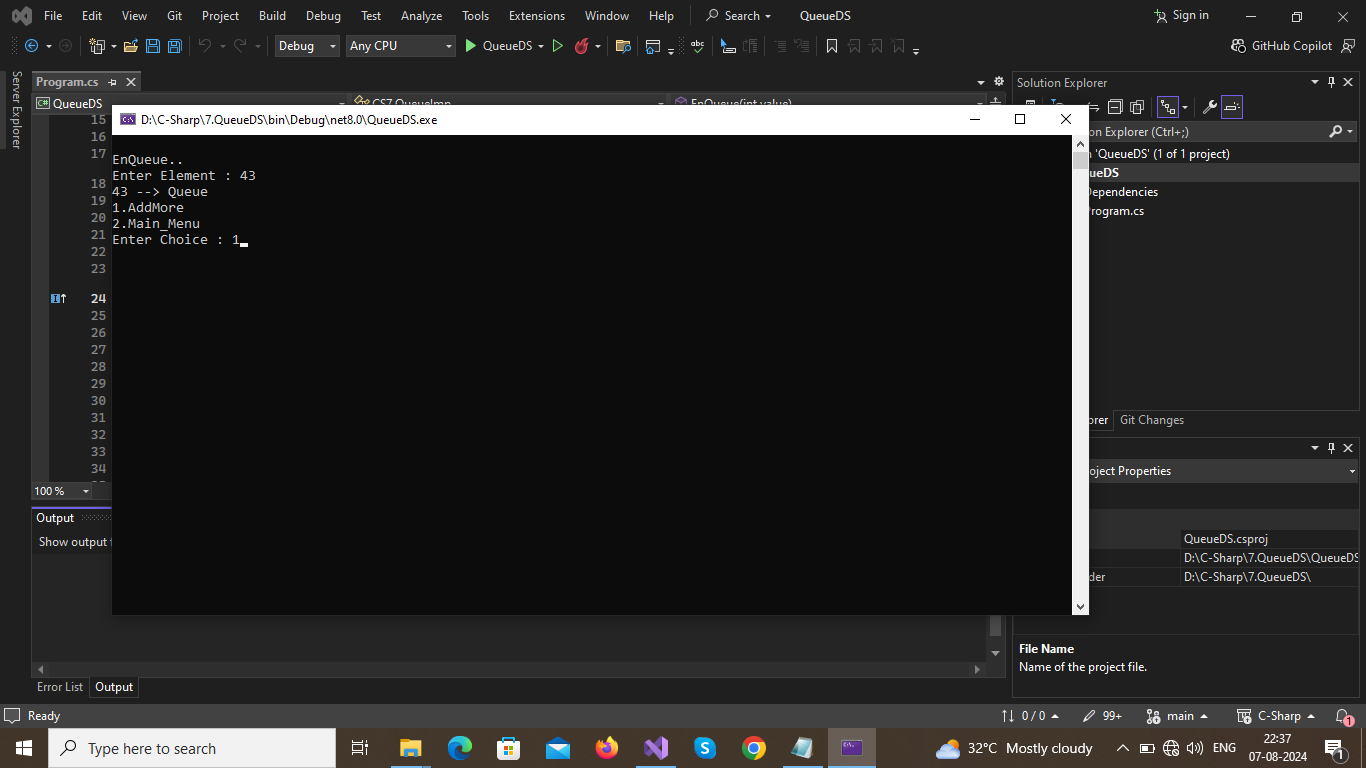
**7. Design a C# interface for Queue data structure. Develop a class that implements this interface using array. Provide necessary exception handling in the implementations.**

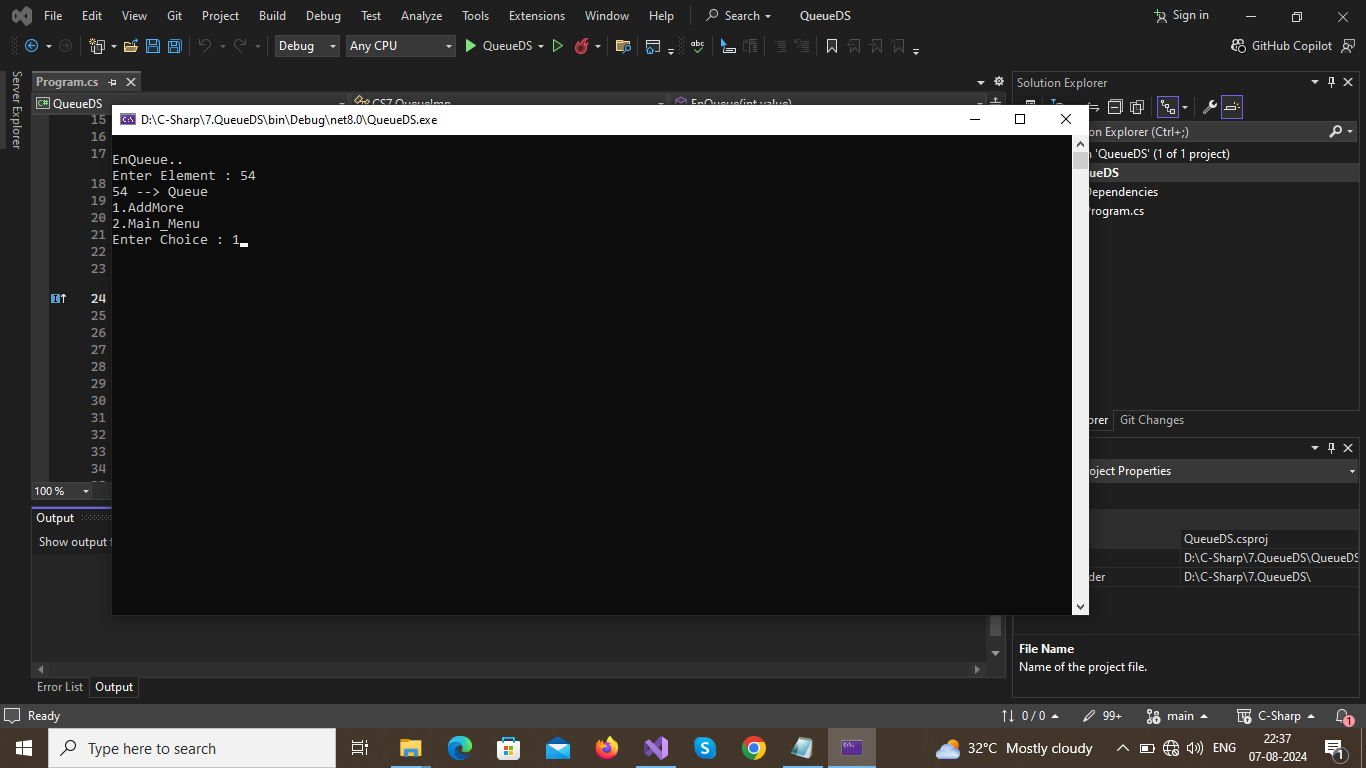
Aim :

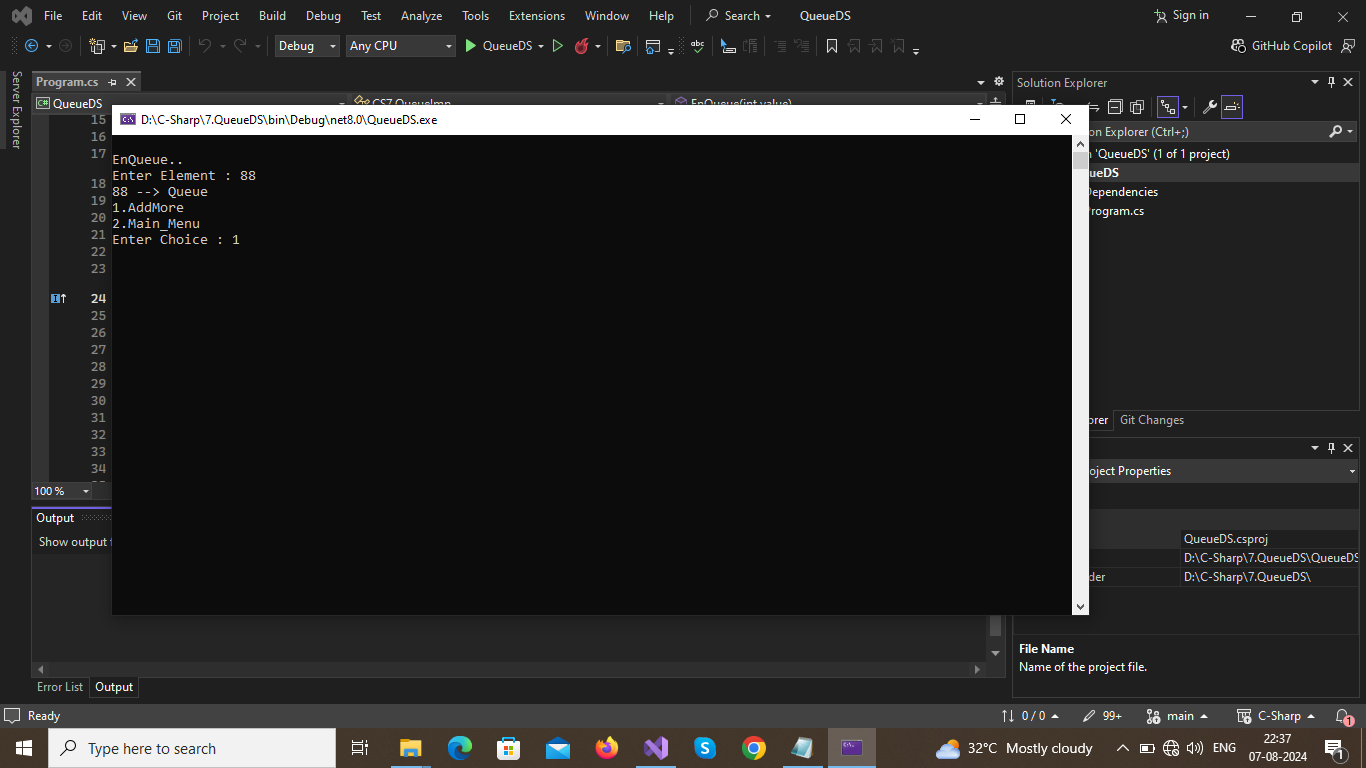
To design a C# interface for Queue data structure. Develop a class that implements this interface using array. Provide necessary exception handling in the implementations.  
  
Algorithm :  
1.Start Program  
2.create an interface that consist of basic queue operations likely  
1.Enque()  
2.Deque()  
3.Front()  
4.Rear()  
5.Display()   
3.Implement using a class (eg:QueueImp)  
4.Enqueue() to add element in rear of queue data structure  
5.Dequeue() to remove element in front of queue data structure  
6.Front() to see element in front of queue ds  
7.Rear() to see element in rear of queue ds  
8.Display() to see all elements in a queue ds.  
9.Stop the program  
  
Program :   
﻿using System;  
namespace CS7  
{  
public interface Queue  
{  
public void EnQueue(int value);  
public int DeQueue();  
public int front();  
public int rear();  
public bool isEmpty();  
public bool isFull();  
public void Display();  
}  
public class QueueImp : Queue  
{  
int[] Que;  
private int Max,Rear,Front;  
public QueueImp(int n)  
{  
Que = new int[n];  
Max = n;  
Rear = Front = 0;  
}  
public void EnQueue(int value)  
{  
if (isFull())  
{  
Console.WriteLine("Queue Is Full {0} is not added.",value);  
}  
else  
{  
Que[Rear] = value;  
Console.WriteLine("{0} --> Queue",value);  
Rear++;  
}  
  
}  
public void Display()  
{  
if (isEmpty())  
{  
Console.WriteLine("No Queue is Formed...");  
}  
else {  
for (int i = 0; i < Rear; i++)  
{  
Console.WriteLine("Queue {0} : {1} ", i + 1, Que[i]);  
}  
Console.WriteLine();  
}  
}  
public int DeQueue()  
{  
if (isEmpty())  
{  
Console.WriteLine("No Queue is Formed...");  
return 0;  
}  
else  
{  
int val = Que[Front];  
for(int i=0;i<Que.Length-1;i++)  
{  
Que[i] = Que[i+1];  
}  
Rear--;  
return val;  
}  
  
}  
public int front()  
{  
if(isEmpty())  
{  
Console.WriteLine("Queue is Empty");  
return 0;  
}  
else{  
return(Que[Front]);  
}   
}  
public int rear()  
{  
if (isEmpty())  
{  
Console.WriteLine("Queue is Empty");  
return 0;  
}  
else  
{  
return (Que[--Rear]);  
}  
}  
public bool isEmpty()  
{  
if(Rear==0)  
{  
return true;  
}  
return false;  
  
}  
public bool isFull()  
{  
if (Rear == Max )  
{   
return true;  
}  
return false;  
}  
private static int Choice()  
{  
Console.WriteLine("Queue Operation...\n1.EnQueue\n2.DeQueue\n3.Front\_Element\n4.Rear\_Element\n5.Display\n6.Exit");  
Console.Write("Enter Choice : ");  
int ch = Int32.Parse(Console.ReadLine());  
return ch;  
}  
private static int Choice(int n)   
{  
int[] re = new int[]{1,7};  
Console.WriteLine("1.AddMore\n2.Main\_Menu");  
Console.Write("Enter Choice : ");  
int ch = Int32.Parse(Console.ReadLine());  
return re[ch-1];  
}  
public static void Main()  
{  
Console.Write("Enter Queue Limit... : ");  
int n = Int32.Parse(Console.ReadLine());  
QueueImp queue1 = new QueueImp(n);  
int ch = Choice();  
while (true)  
{  
switch (ch)  
{  
case 1:  
Console.Write("\nEnQueue..\nEnter Element : ");  
queue1.EnQueue(Int32.Parse(Console.ReadLine()));  
ch = Choice(1);  
Console.Clear();  
break;  
case 2:  
Console.WriteLine("Queue --> {0} \n", queue1.DeQueue());  
ch = Choice();  
Console.Clear();  
break;  
case 3:  
Console.WriteLine("Queue Front : {0} \n", queue1.front());  
ch = Choice();  
Console.Clear();  
break;  
case 4:  
Console.WriteLine("Queue Rear : {0} \n ", queue1.rear());  
ch = Choice();  
Console.Clear();   
break;  
case 5:  
Console.WriteLine("Queue Elements are ...");  
queue1.Display();  
ch = Choice();  
break;  
case 6:  
Console.WriteLine("Thanks For using...");  
return;  
case 7:  
Console.Clear();  
ch = Choice();  
break;  
}  
}  
  
}  
}  
}

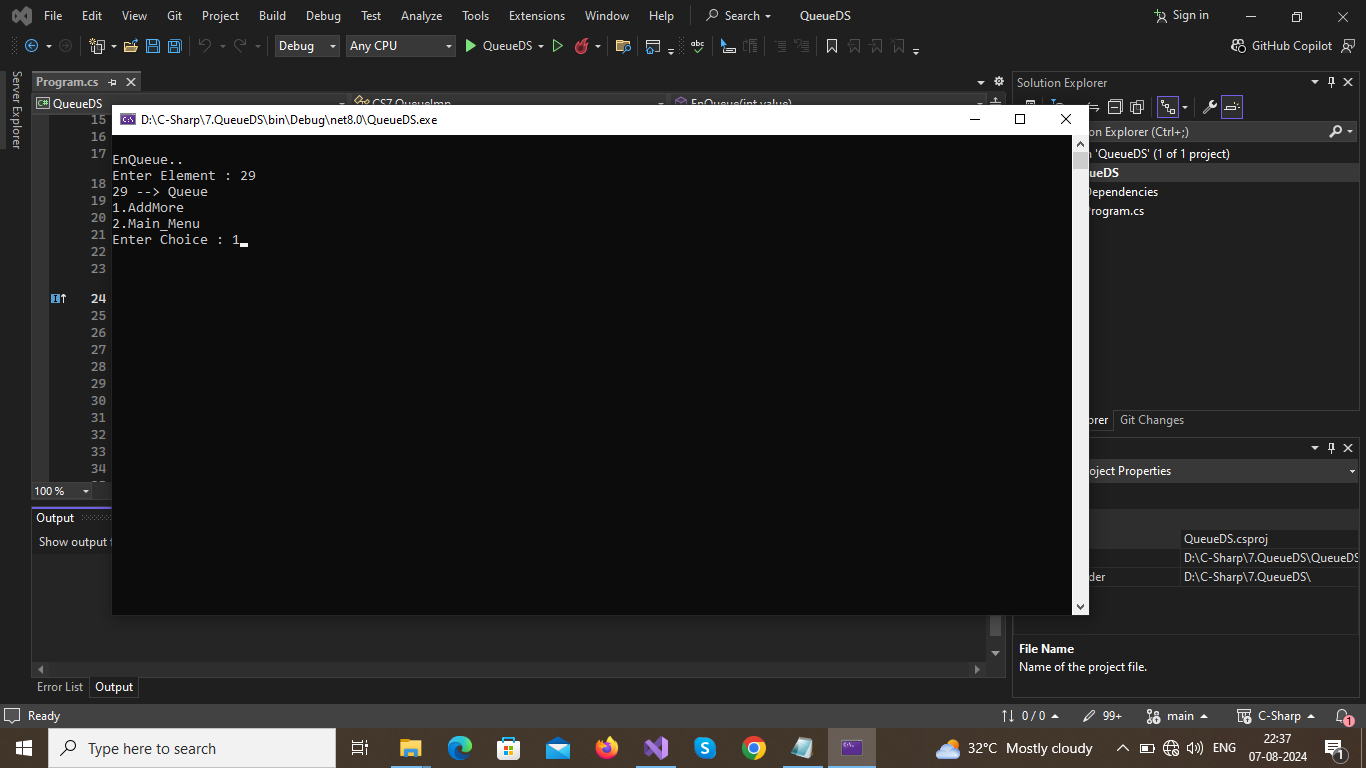
Output :

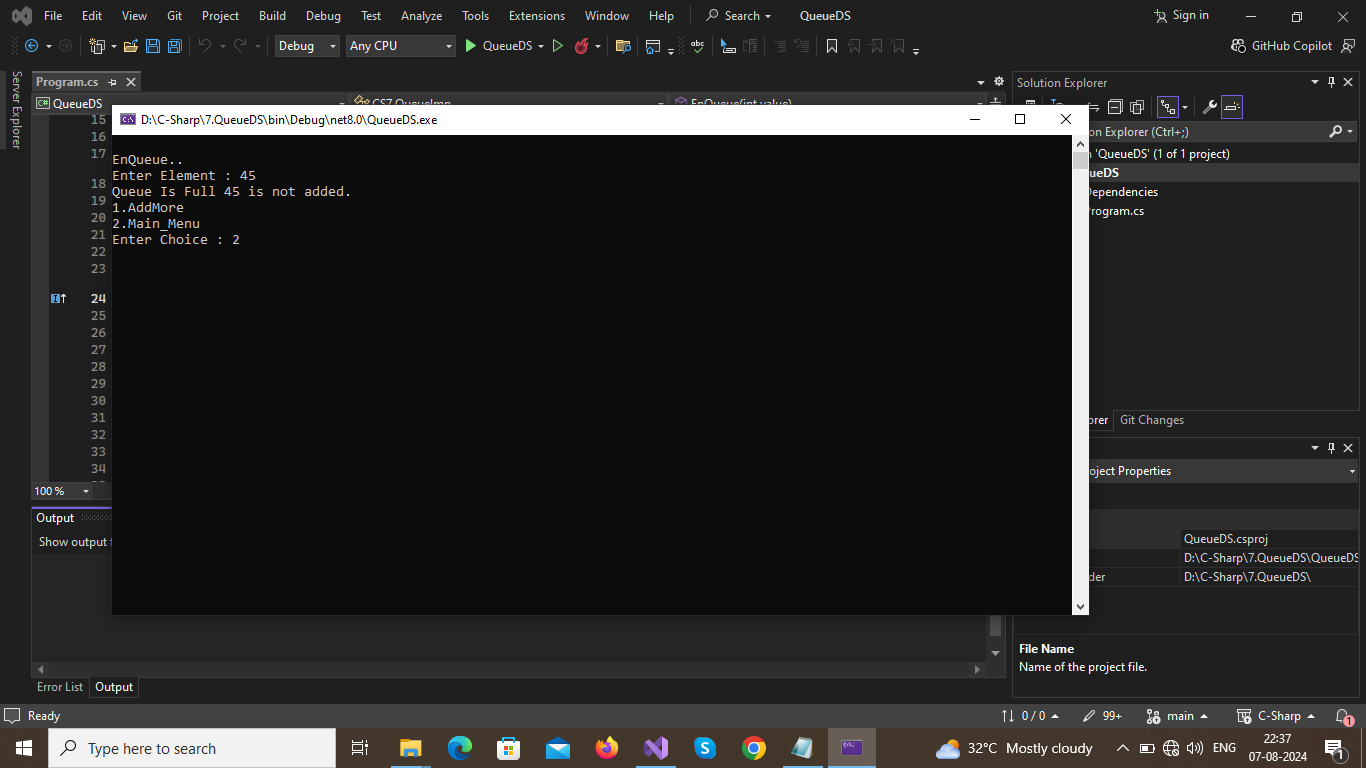


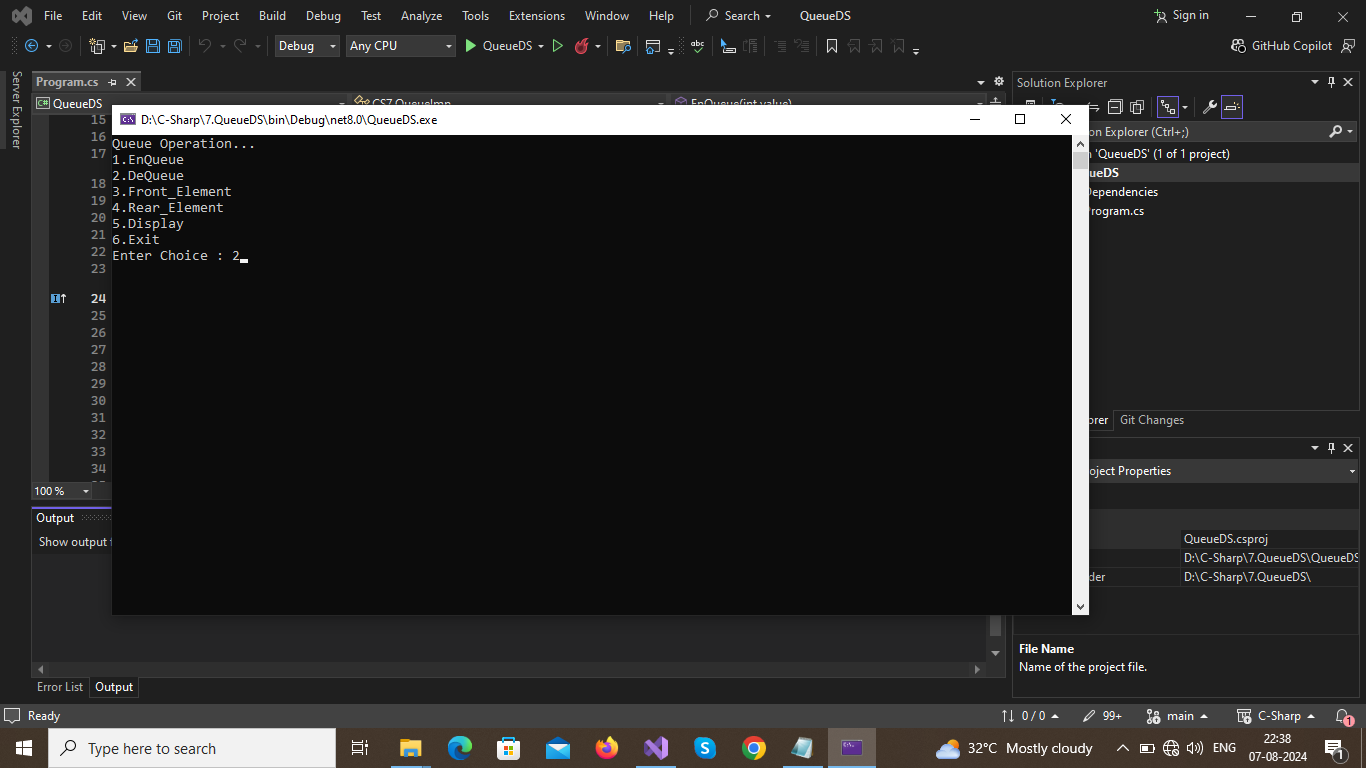


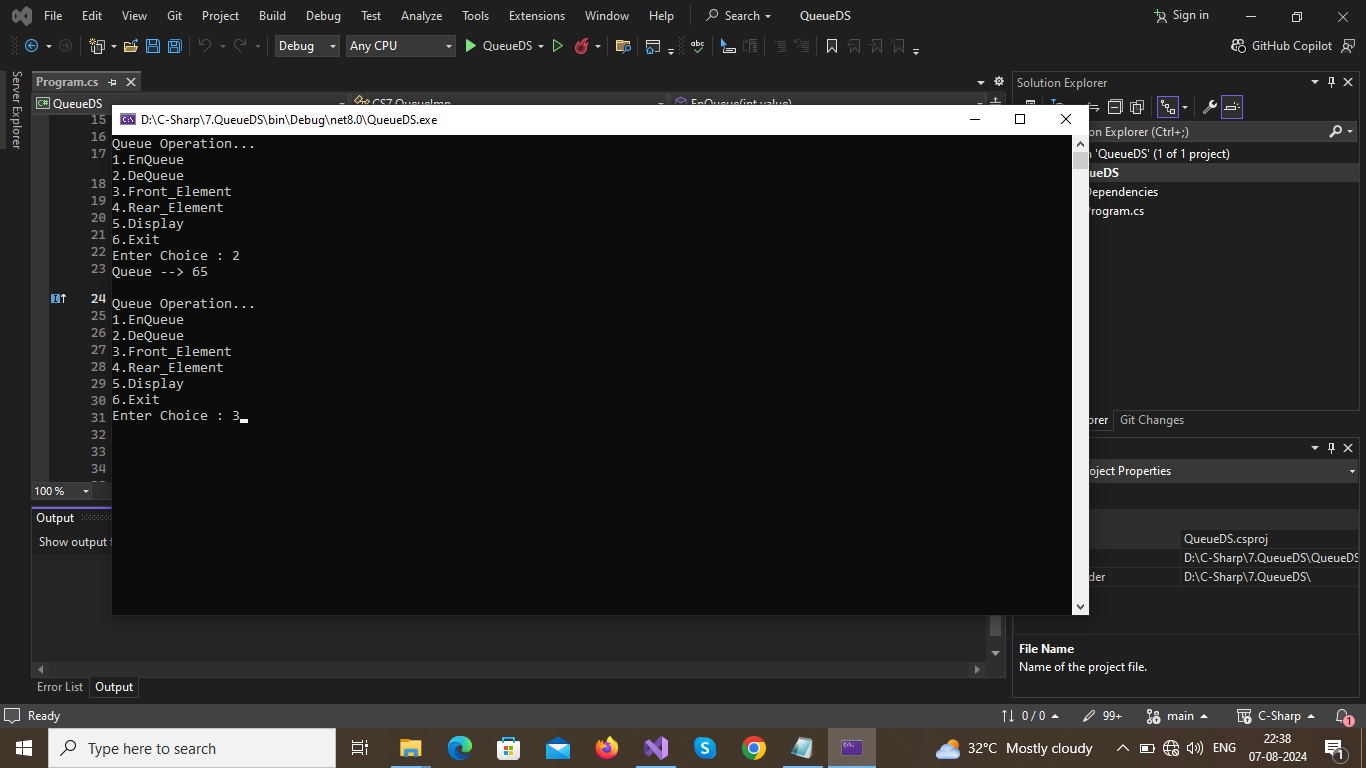


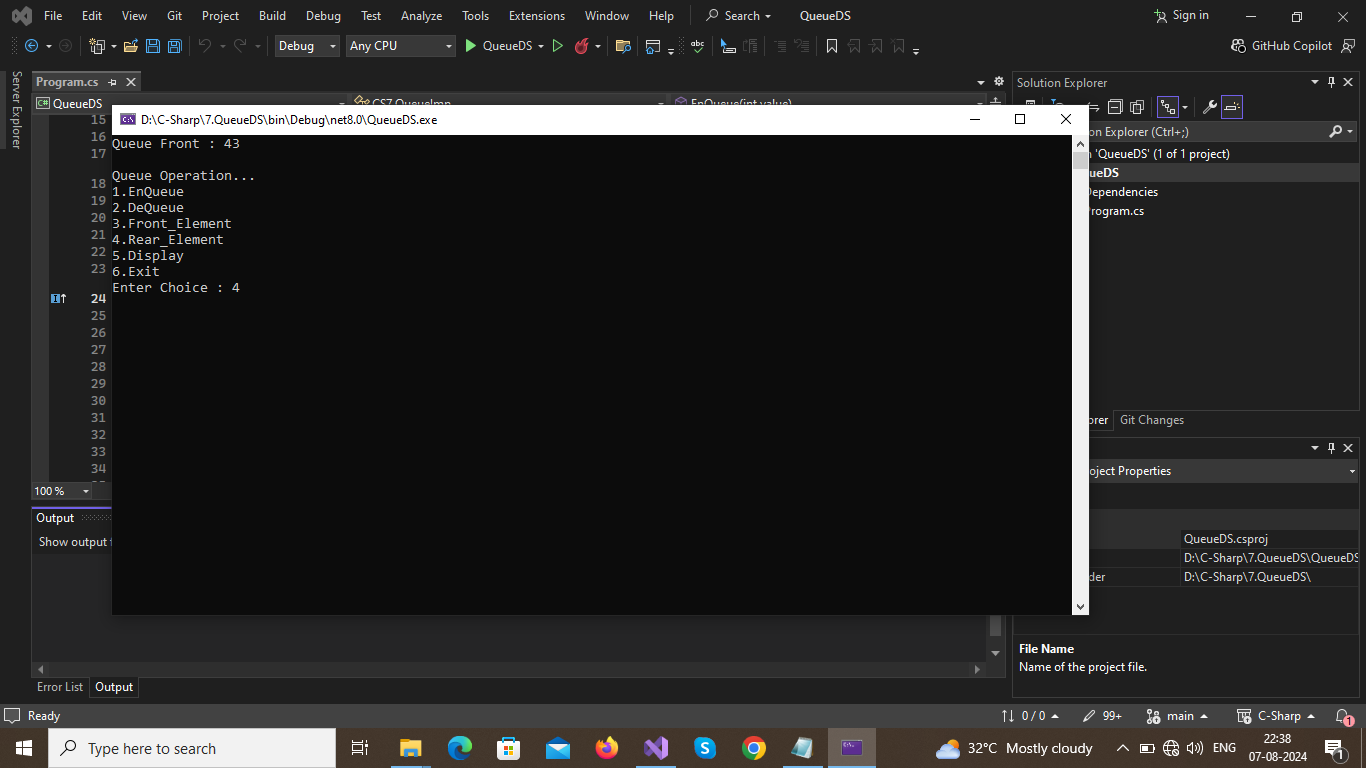


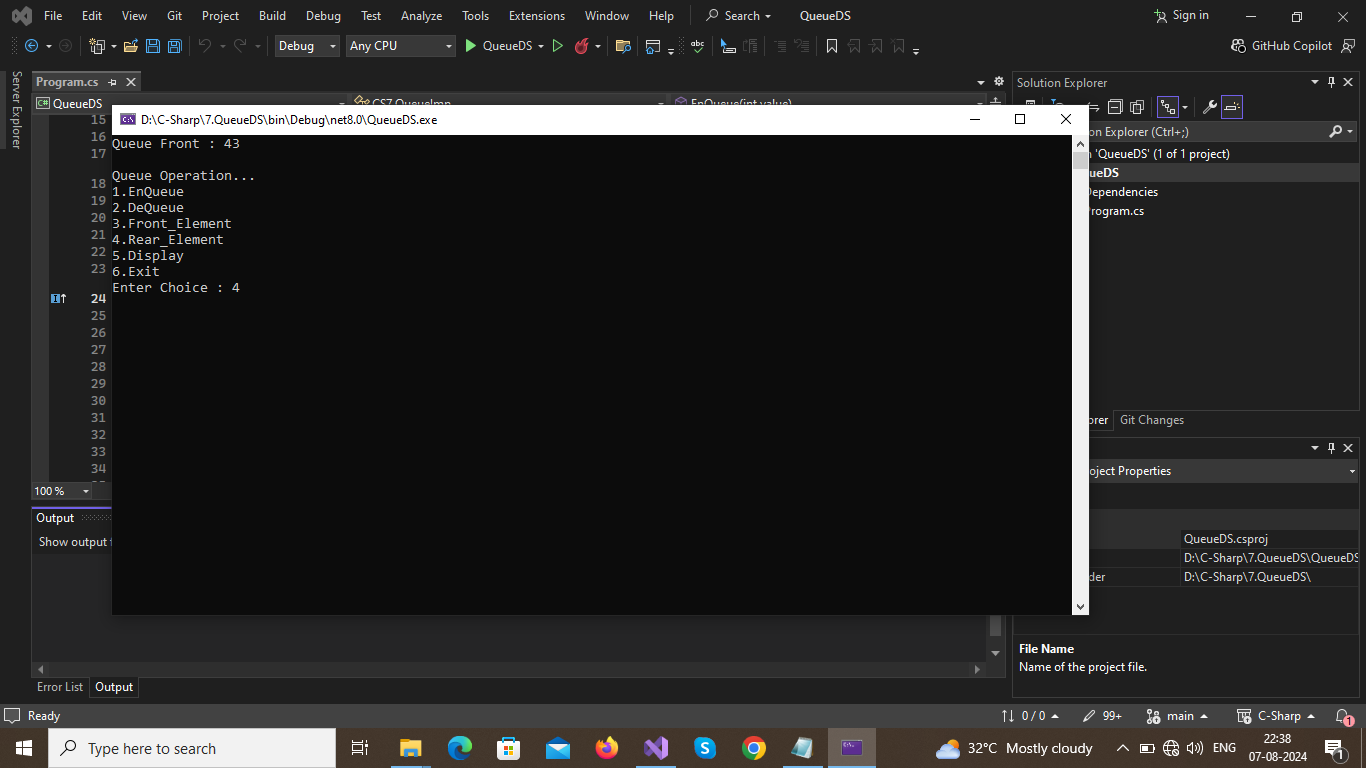
****

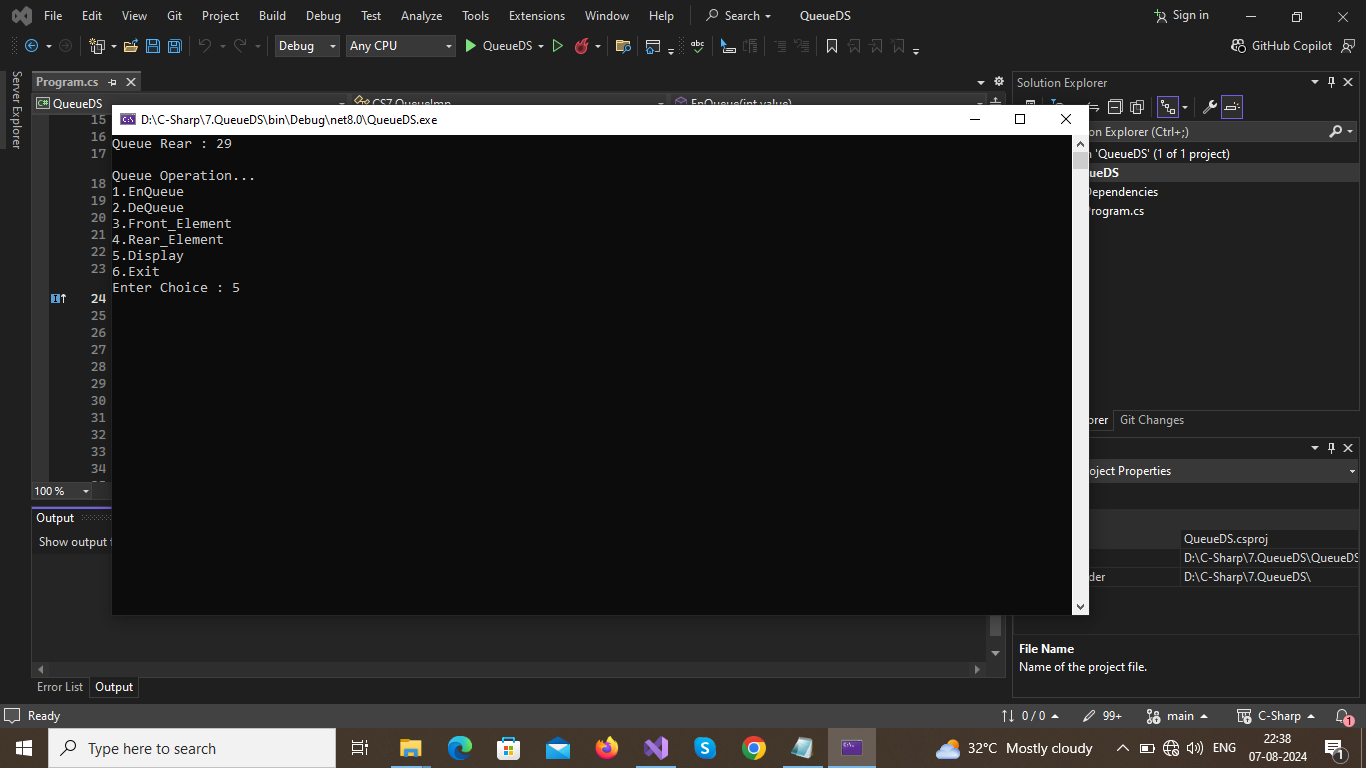
****

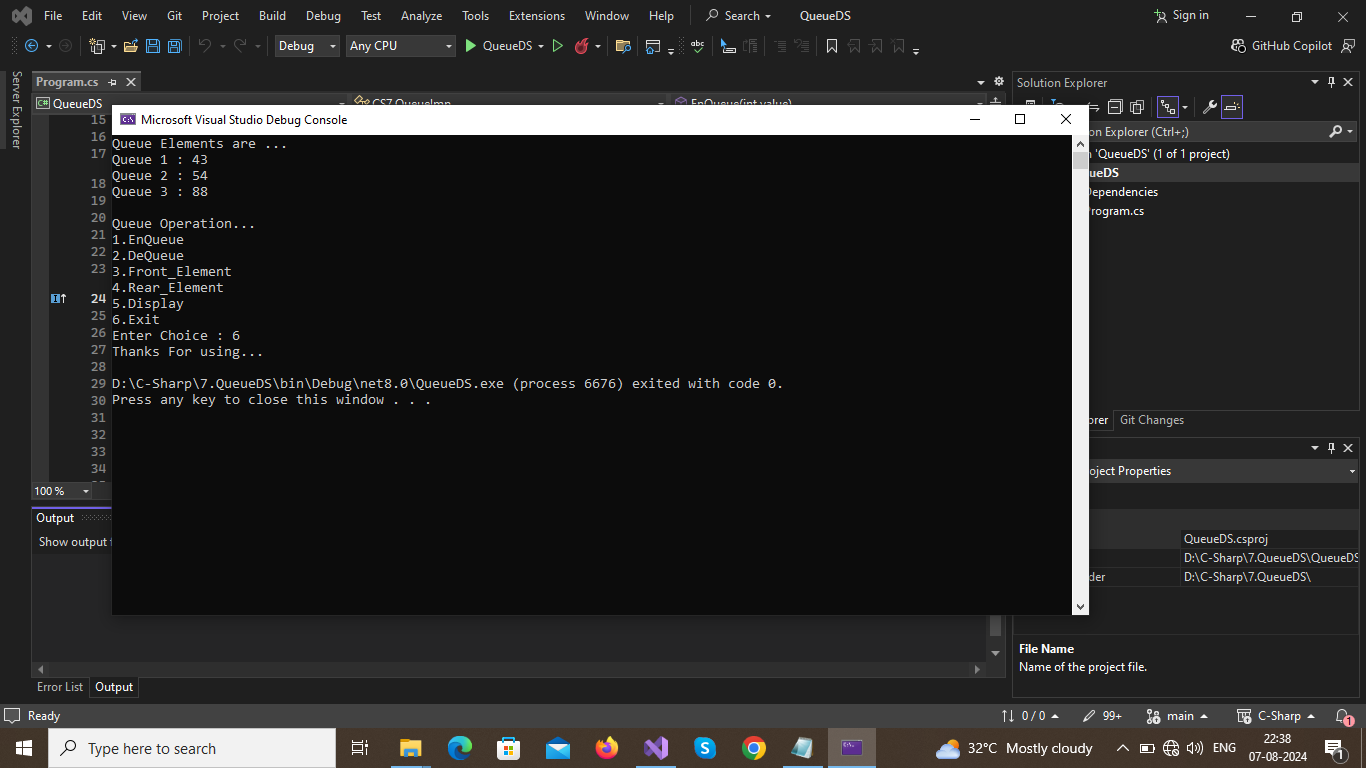
****

****

****

****

****

****

**Result :**

**Thus the program executed successfully and output was verified.**

8.Write a C# program which declares two properties: name and gender in an interface, and provide its implementation in a class which checks for gender and greets the user.

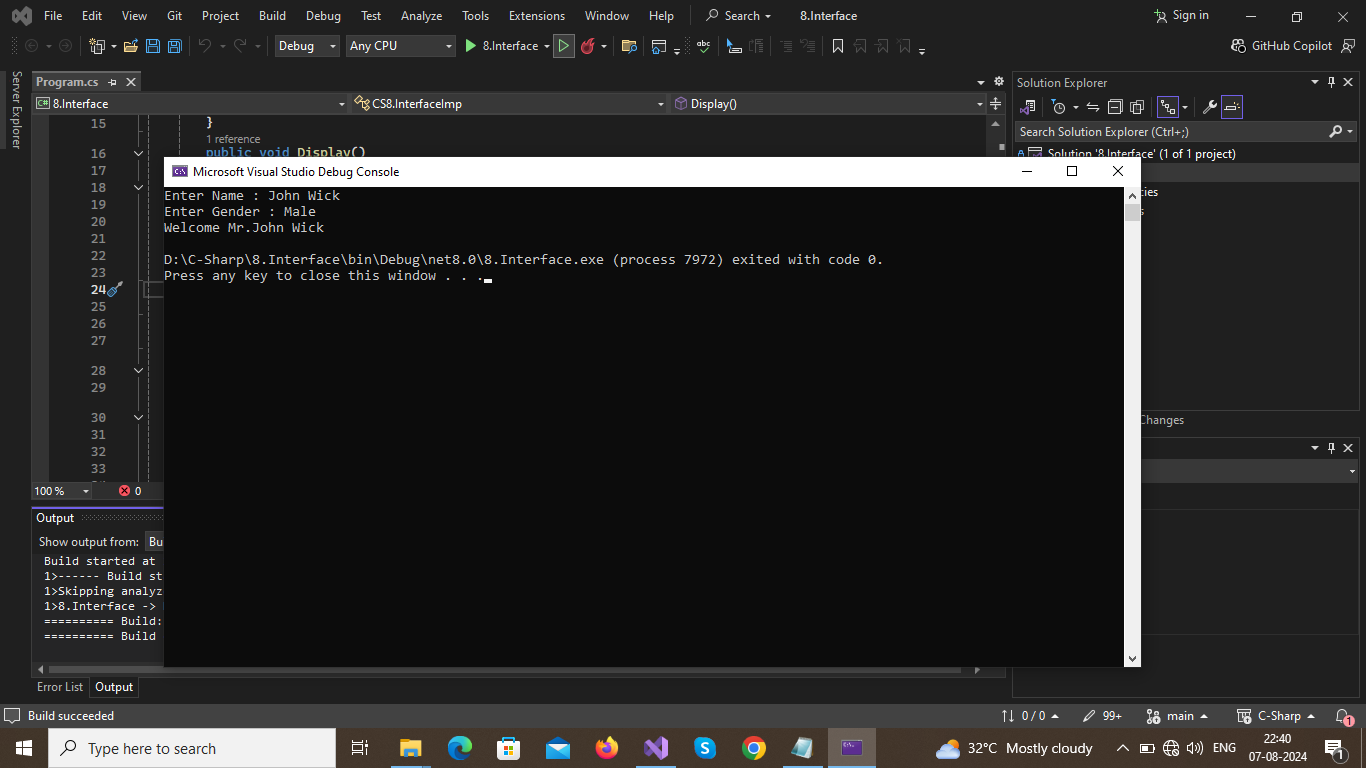
Aim :

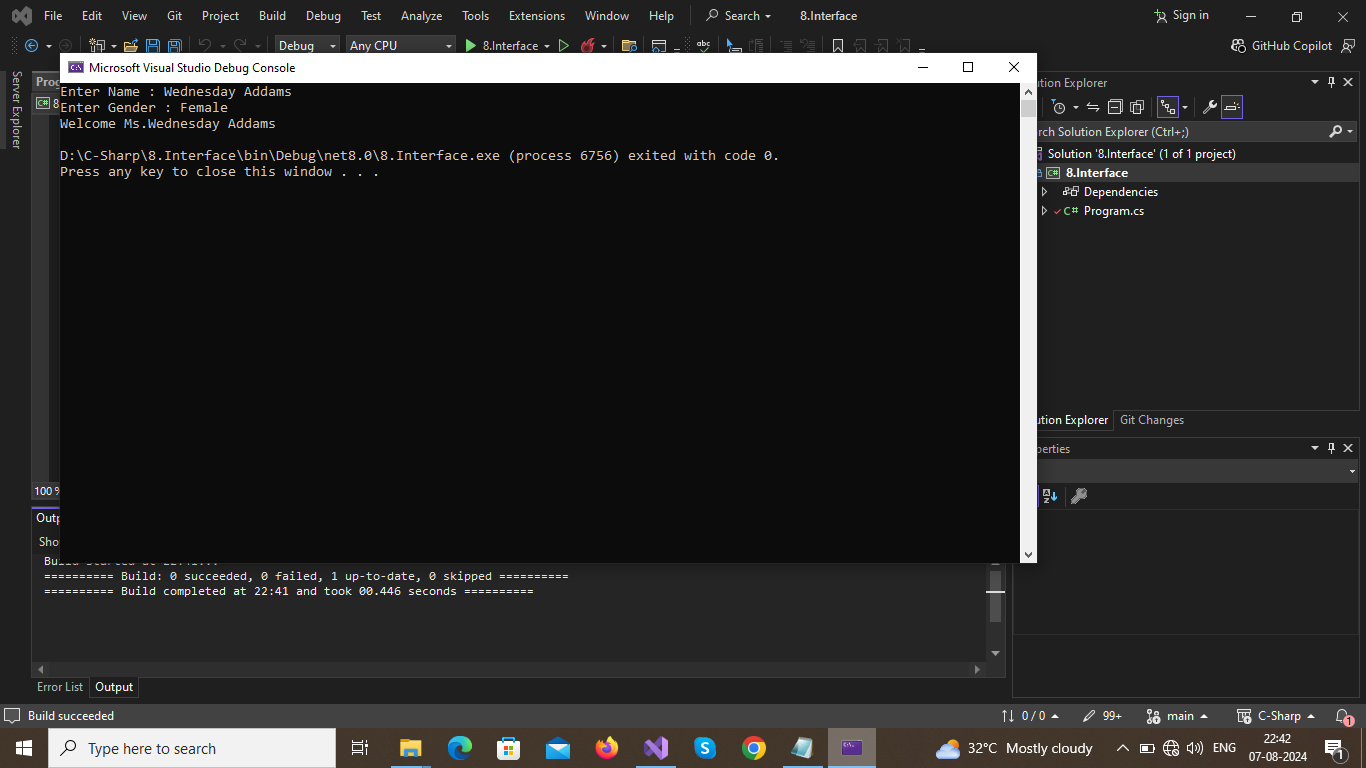
To write a C# program which declares two properties: name and gender in an interface, and provide its implementation in a class which checks for gender and greets the user.

Algorithm :

1.Start program  
2.create an interface that consist method with parametera as name and gender  
3.create an implementation class to implement an interface method to get data such as name and gender.  
4.Print a greeting message based on gender to welcome user.  
5. Stop the Program  
  
Program :   
﻿using System;  
namespace CS8  
{  
public interface Props  
{  
public void Get\_Data(string Name, string Gender);  
}  
public class InterfaceImp : Props  
{  
string Name = "", Gender = "";  
public void Get\_Data(string Name, string Gender)  
{  
this.Name = Name;  
this.Gender = Gender;  
}  
public void Display()  
{  
if (this.Gender.ToLower() == "male" || this.Gender.ToLower()=="m")  
{  
Console.WriteLine("Welcome Mr.{0}",this.Name);  
return;  
  
}  
Console.WriteLine("Welcome Ms.{0}", this.Name);  
}  
  
}  
public class Interface  
{  
public static void Main(string[] args)  
{  
string Name ="", Gender="";  
Console.Write("Enter Name : ");  
Name = Console.ReadLine();  
Console.Write("Enter Gender : ");  
Gender = Console.ReadLine();  
InterfaceImp II = new InterfaceImp();  
II.Get\_Data(Name, Gender);  
II.Display();  
  
}  
}  
  
}

Output :





**Result :**

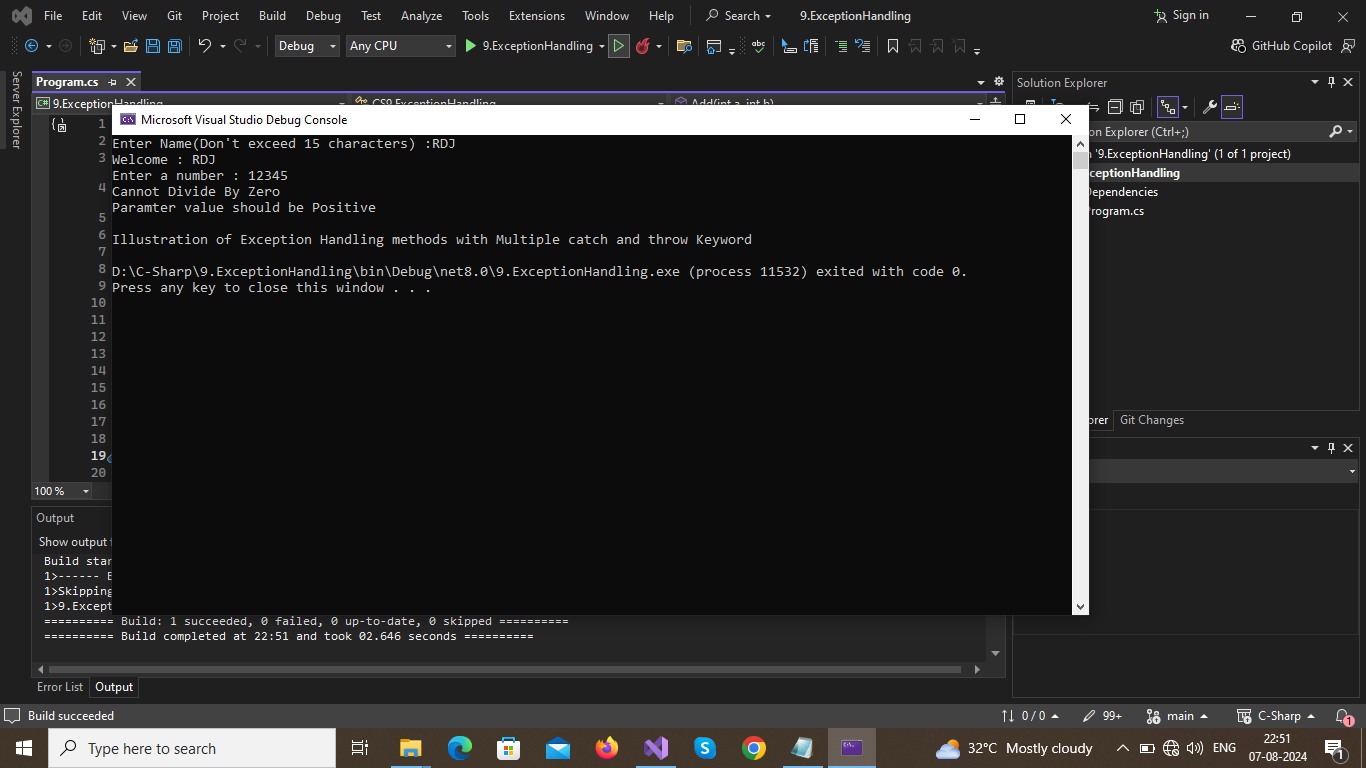
**Thus the program executed successfully and output was verified.**

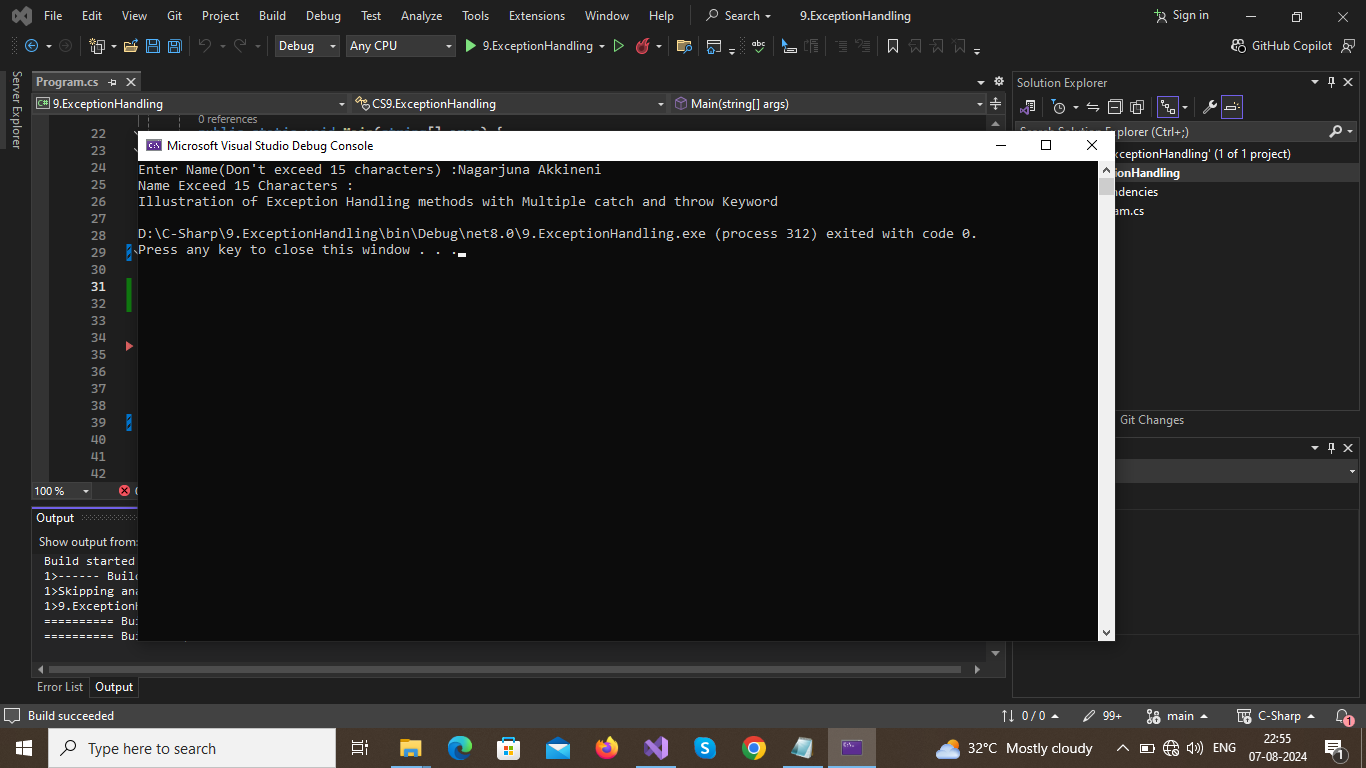
**9.Develop a program that is likely to throw multiple exceptions that are handled using catch and finally blocks. As a part, the program must read a name from the keyboard and display it on the screen. The program should throw an exception when the length of the name is more than 15 characters. Design your own exception handling mechanism**

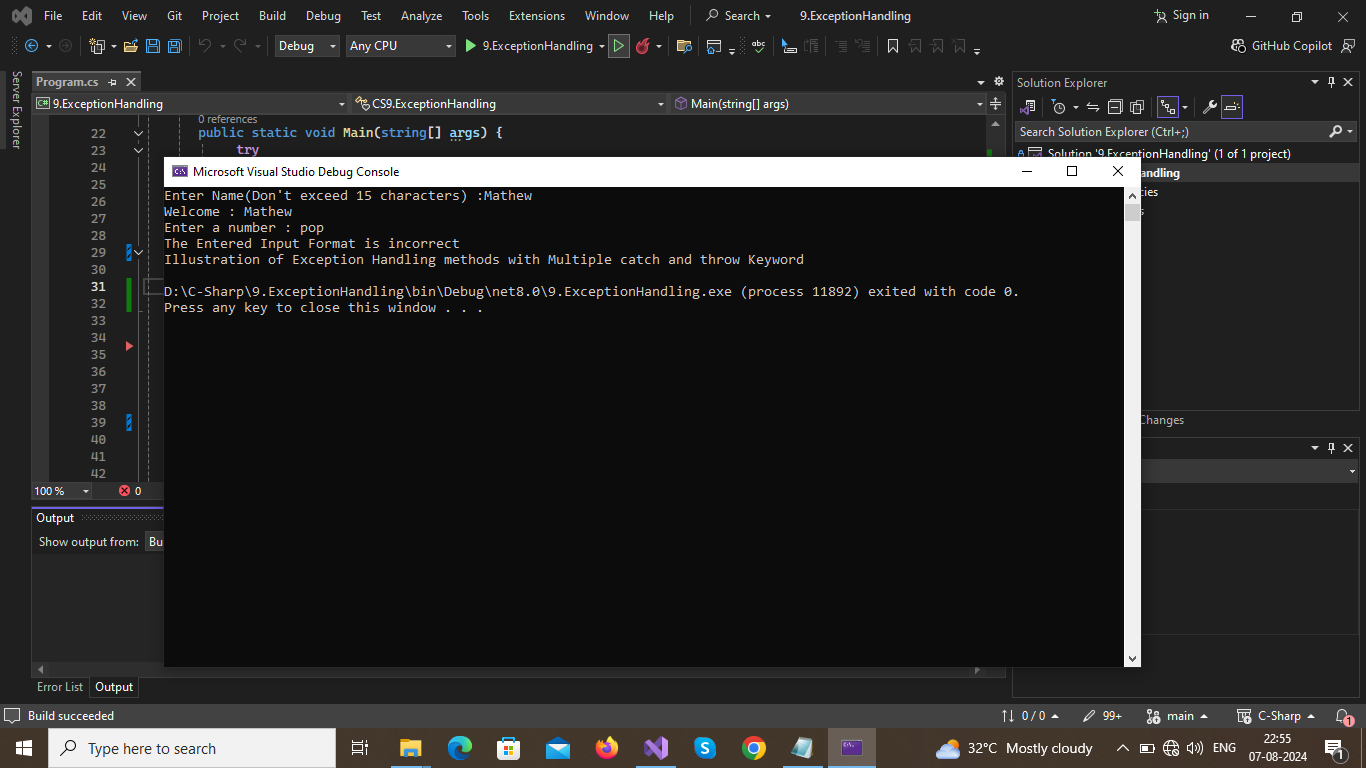
Aim :

To develop a program that is likely to throw multiple exceptions that are handled using catch and finally blocks. As a part, the program must read a name from the keyboard and display it on the screen. The program should throw an exception when the length of the name is more than 15 characters. Design your own exception handling mechanism  
  
Algorithm :   
1.Start the program  
2.Create a try catch block to illustrate exception handling  
3.Get a Name as input in try block and check the name has less than or equal to 15 characters   
4.if it is more than 15 then raise an exception and give a message  
5.add more exceptions like division by 0,formatexception,and so on.  
6.stop the program  
Program :   
﻿using System;  
namespace CS9 {  
class ExceptionHandling{  
public static object Add(int a,int b)  
{  
int sum = 0;  
if (a == 0 && b == 0)  
{  
ArgumentException e = new ArgumentException("any one paramters value should be 1 or greater");  
return e;  
}  
sum = a + b;  
return sum;  
}  
public static void Main(string[] args) {  
try  
{  
char[] Name = new char[15];  
int m=0, n=0;  
Console.Write("Enter Name(Don't exceed 15 characters) :");  
Name = Console.ReadLine().ToCharArray();  
if (Name.Length > 15)  
{  
throw new IndexOutOfRangeException();  
}  
Console.Write("Welcome : ");  
Console.WriteLine(Name);  
Console.Write("Enter a number 1 : ");  
n = Convert.ToInt32(Console.ReadLine());  
Console.Write("Enter a number 2 : ");  
m = Convert.ToInt32(Console.ReadLine());  
Console.WriteLine("The Sum Of {0}+{1} : {2}", m, n, Add(m, n));  
Console.Write("The {0}/0 Divided by 0 goes : ",m);  
Console.WriteLine(m / 0);  
  
  
}  
catch (IndexOutOfRangeException)  
{  
Console.WriteLine("Name Exceed 15 Characters : ");  
}  
catch (DivideByZeroException)  
{  
Console.WriteLine(new DivideByZeroException("Cannot Divide By Zero "));  
}  
catch (NotFiniteNumberException)  
{  
Console.WriteLine("It is Not-a-Number(NAN)");  
}  
catch (FormatException) {  
Console.WriteLine("The Entered Input Format is invalid");  
}  
finally  
{  
Console.WriteLine("Illustration of Exception Handling methods with\n1.throw Keyword\n2.try-catch block\n3.finally");  
}  
}  
}  
}

Output :



. 



**Result :**

**Thus the program executed successfully and output was verified.**