C# Practicals

Name : Nigel Fernandes

UID : 205031

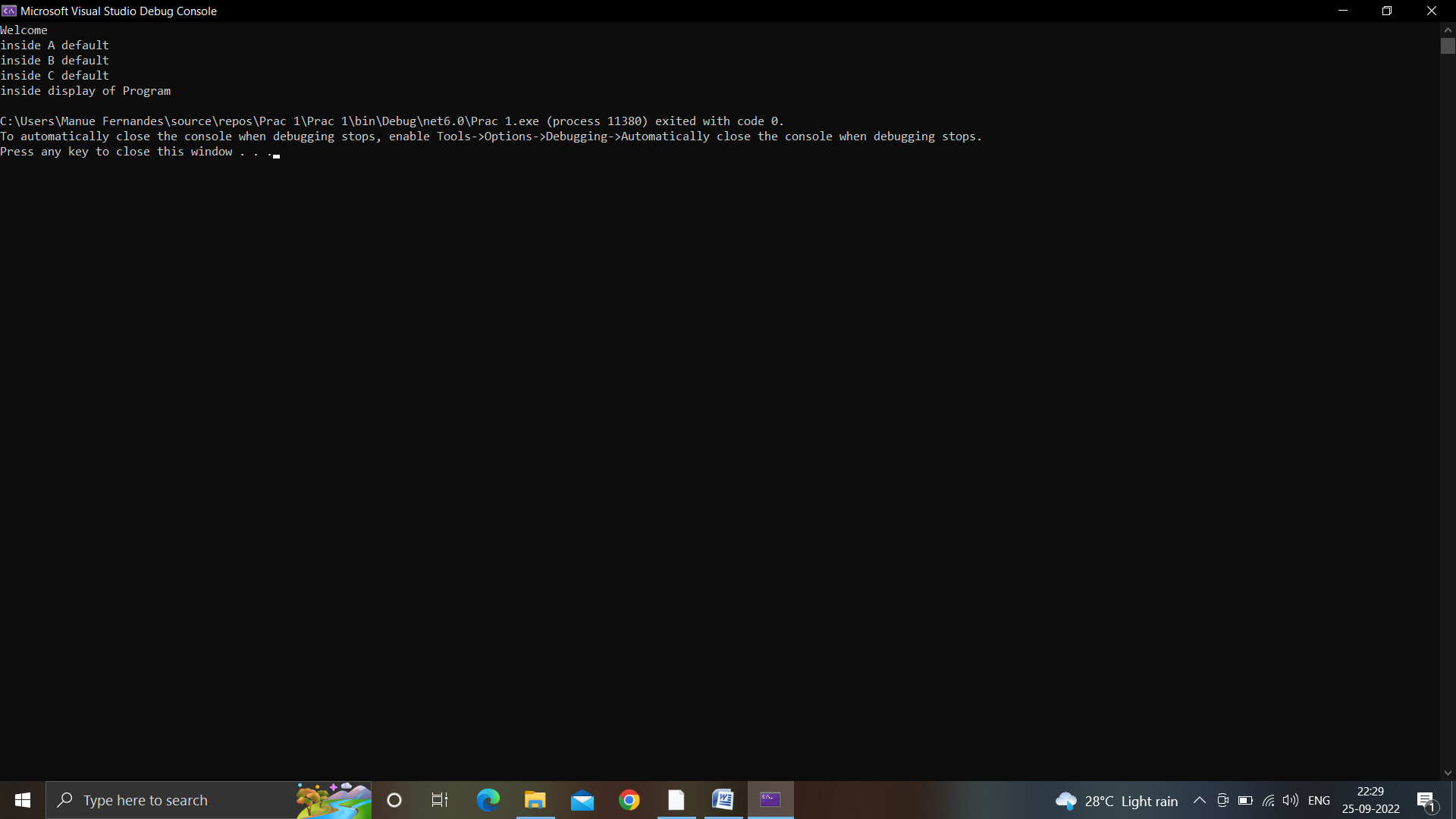
RollNo : 14

PRACTICAL 1

Q) Different types of calling different methods

**TYPE 1**

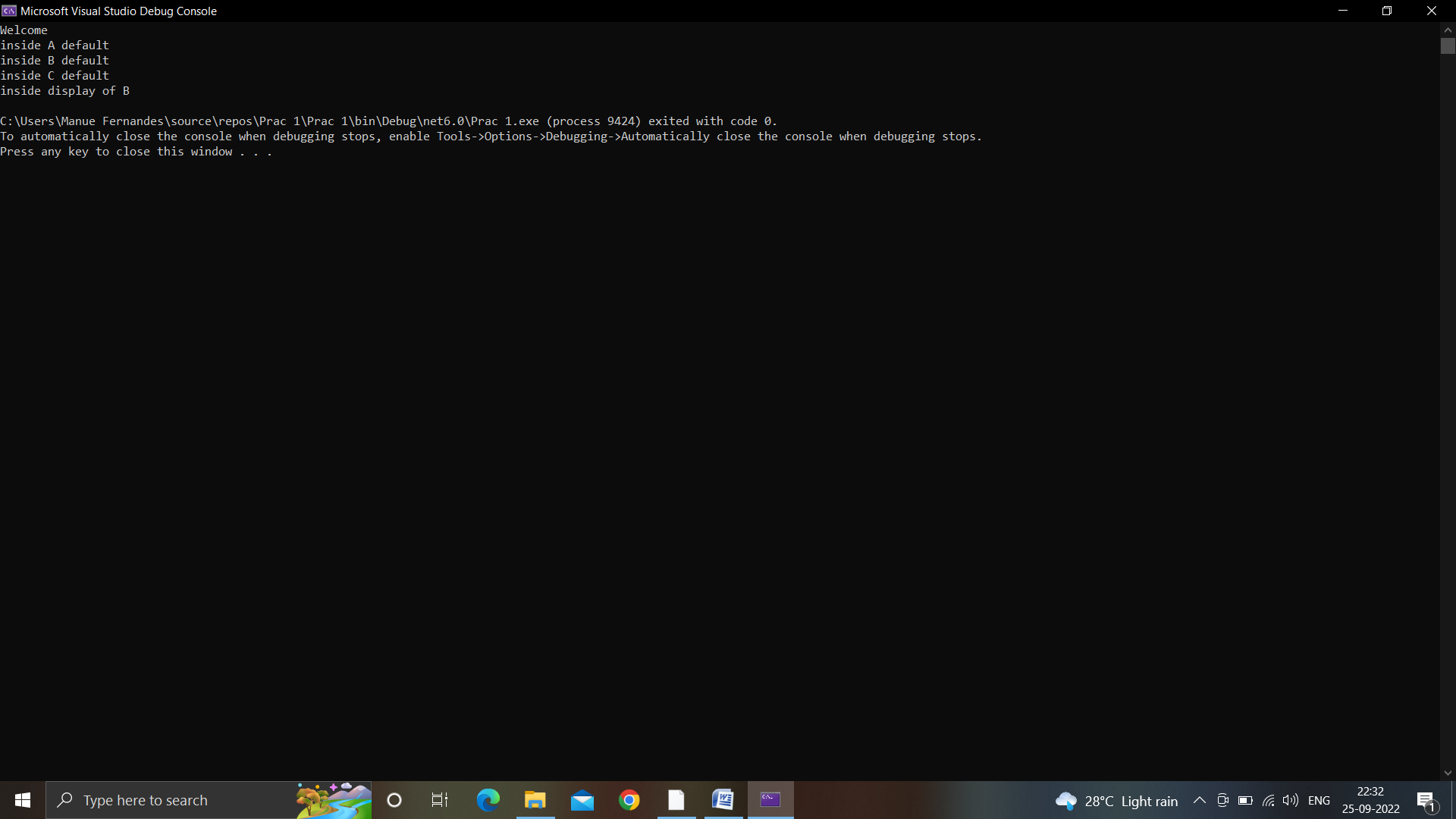
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
  
namespace BasicDemo  
{  
    class A  
    {  
        public A()  
        {  
            Console.WriteLine("inside A default");  
        }  
        public virtual void display()  
        {  
            Console.WriteLine("inside display of A");  
        }  
    }  
    class B:A  
    {  
        public B()  
        {  
            Console.WriteLine("inside B default");  
        }  
        public override void display()  
        {  
            Console.WriteLine("inside display of B");  
        }  
    }  
    class C:B  
    {  
        public C()  
        {  
            Console.WriteLine("inside C default");  
        }  
        public override void display()  
        {  
            Console.WriteLine("inside display of C");  
        }  
    }  
    class Program:C  
    {  
        public override void display()  
        {  
            Console.WriteLine("inside display of Program");  
        }  
        static void Main(string[] args)  
        {  
            Console.WriteLine("Welcome");  
            Program p = new Program();  
            p.display();  
            Console.ReadKey();  
        }  
    }  
}

**Outpugt:** 

**TYPE 2**

using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
  
namespace BasicDemo  
{  
    class A  
    {  
        public A()  
        {  
            Console.WriteLine("inside A default");  
        }  
        public virtual void display()  
        {  
            Console.WriteLine("inside display of A");  
        }  
    }  
    class B:A  
    {  
        public B()  
        {  
            Console.WriteLine("inside B default");  
        }  
        public override void display()  
        {  
            Console.WriteLine("inside display of B");  
        }  
    }  
    class C:B  
    {  
        public C()  
        {  
            Console.WriteLine("inside C default");  
        }  
        public new virtual void display()  
        {  
            Console.WriteLine("inside display of C");  
        }  
    }  
    class Program:C  
    {  
        public override void display()  
        {  
            Console.WriteLine("inside display of Program");  
        }  
        static void Main(string[] args)  
        {  
            Console.WriteLine("Welcome");  
            A p = new Program();  
            p.display();  
            Console.ReadKey();  
        }  
    }  
}

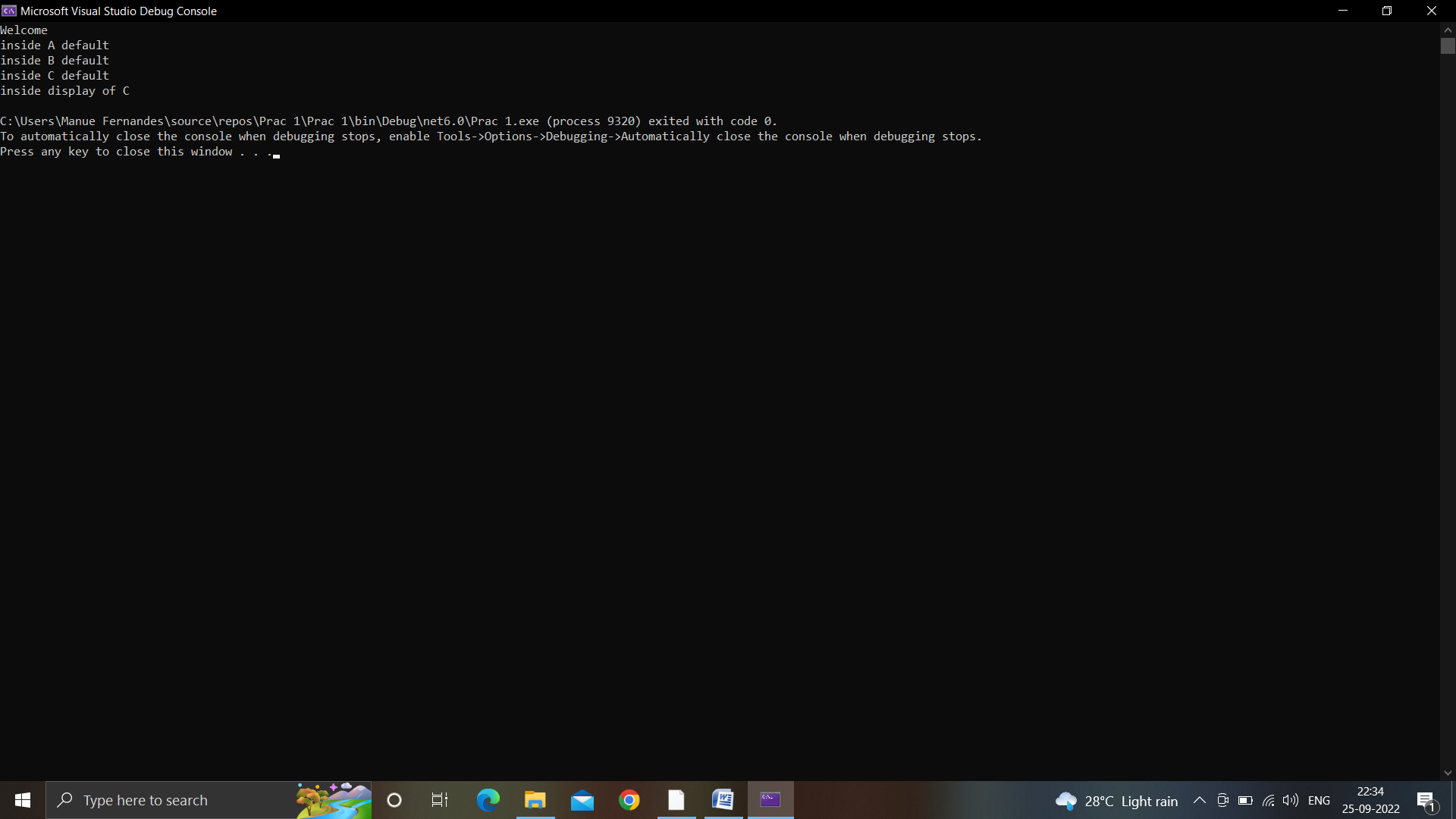
**Output:**

****

**TYPE 3**

using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
  
namespace BasicDemo  
{  
    class A  
    {  
        public A()  
        {  
            Console.WriteLine("inside A default");  
        }  
        public virtual void display()  
        {  
            Console.WriteLine("inside display of A");  
        }  
    }  
    class B:A  
    {  
        public B()  
        {  
            Console.WriteLine("inside B default");  
        }  
        public override void display()  
        {  
            Console.WriteLine("inside display of B");  
        }  
    }  
    class C:B  
    {  
        public C()  
        {  
            Console.WriteLine("inside C default");  
        }  
        public override void display()  
        {  
            Console.WriteLine("inside display of C");  
        }  
    }  
    class Program:C  
    {  
        public new void display()  
        {  
            Console.WriteLine("inside display of Program");  
        }  
        static void Main(string[] args)  
        {  
            Console.WriteLine("Welcome");  
            A p = new Program();  
            p.display();  
            Console.ReadKey();  
        }  
    }  
}

**Output:**

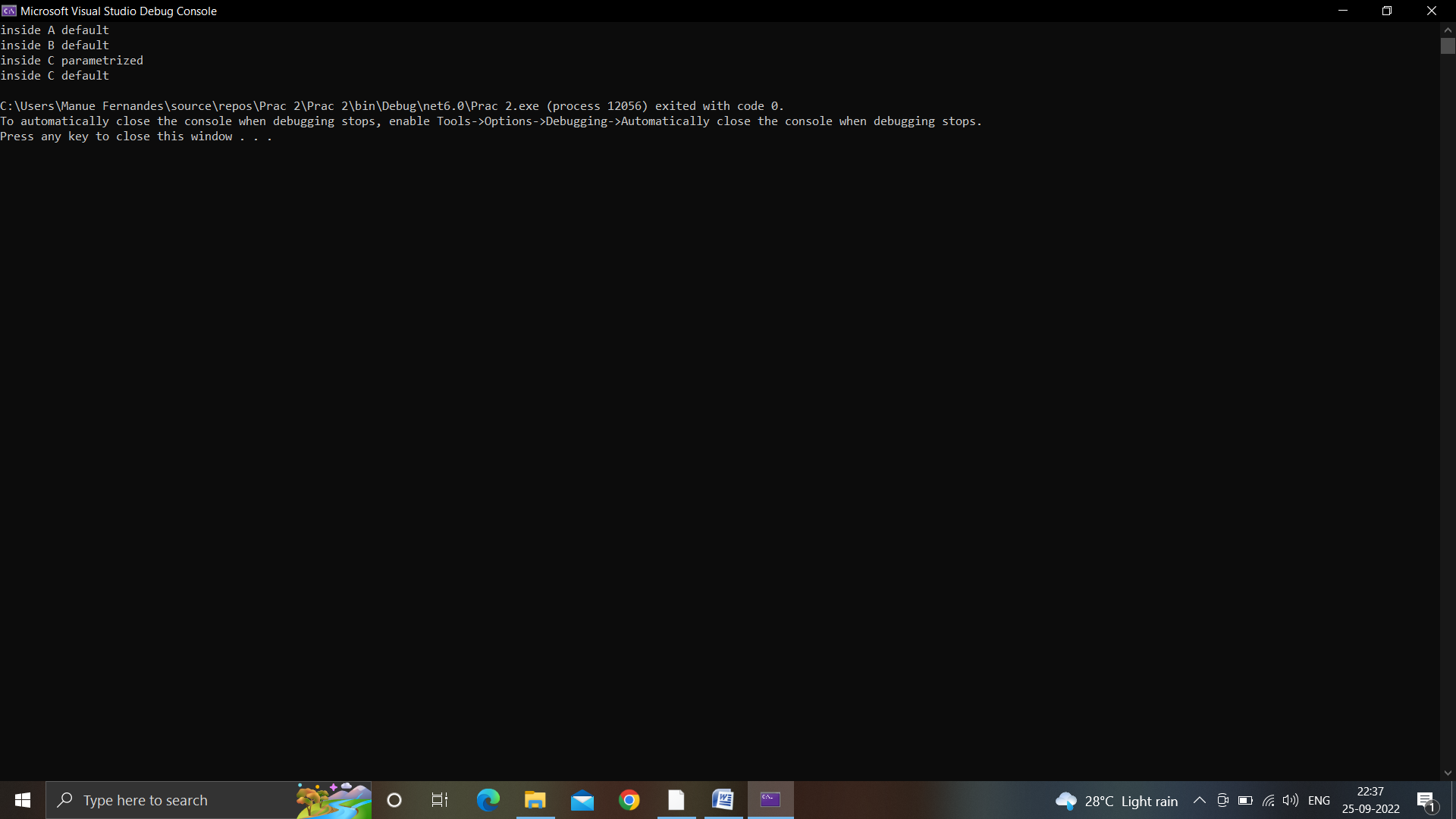
****

PRACTICAL 2

Q) Understanding Constructor.

using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
  
namespace BasicDemo  
{  
    class A  
    {  
        public A()  
        {  
            Console.WriteLine("inside A default");  
        }  
    }  
    class B : A  
    {  
        public B()  
        {  
            Console.WriteLine("inside B default");  
        }  
    }  
    class C : B  
    {  
     
        public C():this(8)  
        {  
            Console.WriteLine("inside C default");  
        }  
        public C(int n)  
        {  
            Console.WriteLine("inside C parametrized");  
        }  
    }  
    class Program : C  
    {  
        static void Main(string[] args)  
        {  
            A p = new Program();  
            Console.ReadKey();  
        }  
    }  
}

**Output:**



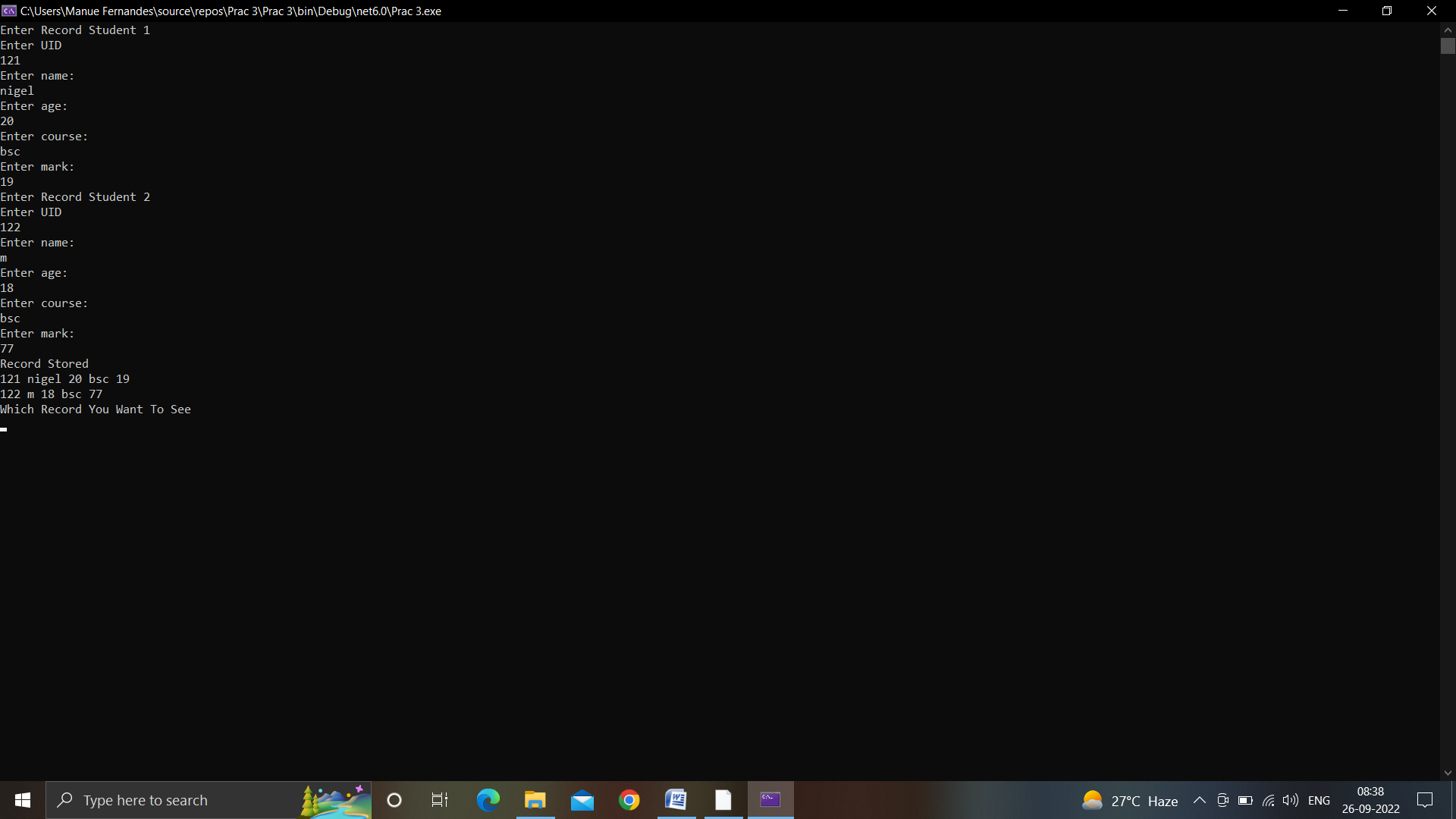
PRACTICAL 3

Q) To create a DS of student

1. Store the record
2. Retrieve as per requirement

namespace StorageProg  
{  
  
    class Student  
    {  
        public int UID;  
        public string name;  
        public int age;  
        public string course;  
        public int mark;  
   
        public Student(int UID, string name, int age, string course, int mark)  
        {  
            this.UID = UID;  
            [this.name](http://this.name/) = name;  
            this.age = age;  
            this.course = course;  
            this.mark = mark;  
        }  
    }  
    internal class Program  
    {  
        static void Main(string[] args)  
        {  
            //Console.WriteLine("Hello, World!");  
            Student[] s = new Student[2];  
            int UID;  
            string name;  
            int age;  
            string course;  
            int mark;  
            int totalMarks = 0;  
  
            for(int i = 0; i < s.Length; i++)  
            {  
                Console.WriteLine("Enter Record Student " + (i + 1));  
                Console.WriteLine("Enter UID");  
                UID = int.Parse(Console.ReadLine());  
                Console.WriteLine("Enter name:");  
                name = Console.ReadLine();  
                Console.WriteLine("Enter age:");  
                age = int.Parse(Console.ReadLine());  
                //age = Convert.ToInt32(Console.ReadLine())  
                Console.WriteLine("Enter course:");  
                course = Console.ReadLine();  
                Console.WriteLine("Enter mark:");  
                mark = int.Parse(Console.ReadLine());  
  
                s[i] = new Student(UID, name, age, course, mark);  
  
            }  
            Console.WriteLine("Record Stored");  
  
            for(int i = 0; i < s.Length; i++)  
            {  
                Console.WriteLine(s[i].UID + " " + s[i].name + " " + s[i].age + " " + s[i].course + " " + s[i].mark + " ");  
            }  
  
            Console.WriteLine("Which Record You Want To See");  
            int search = int.Parse(Console.ReadLine());  
             
            for (int i = 0; i < s.Length; i++)  
            {  
                if(search == s[i].UID)  
                {  
                    Console.WriteLine(s[i].UID + " " + s[i].name + " " + s[i].age + " " + s[i].course + " " + s[i].mark + " ");  
                }  
            }  
  
            Console.WriteLine("Enter Course Name to search Marks of specific department");  
            string searchCourseMarks = Console.ReadLine();  
  
            for (int i = 0; i < s.Length; i++)  
            {  
                if (searchCourseMarks == s[i].course)  
                {  
                    totalMarks += s[i].mark;  
                }  
            }  
            Console.WriteLine(searchCourseMarks + " " + totalMarks + " ");  
        }  
    }  
}

**Output:**

****

PRACTICAL 4

Q) To understand the array in C#

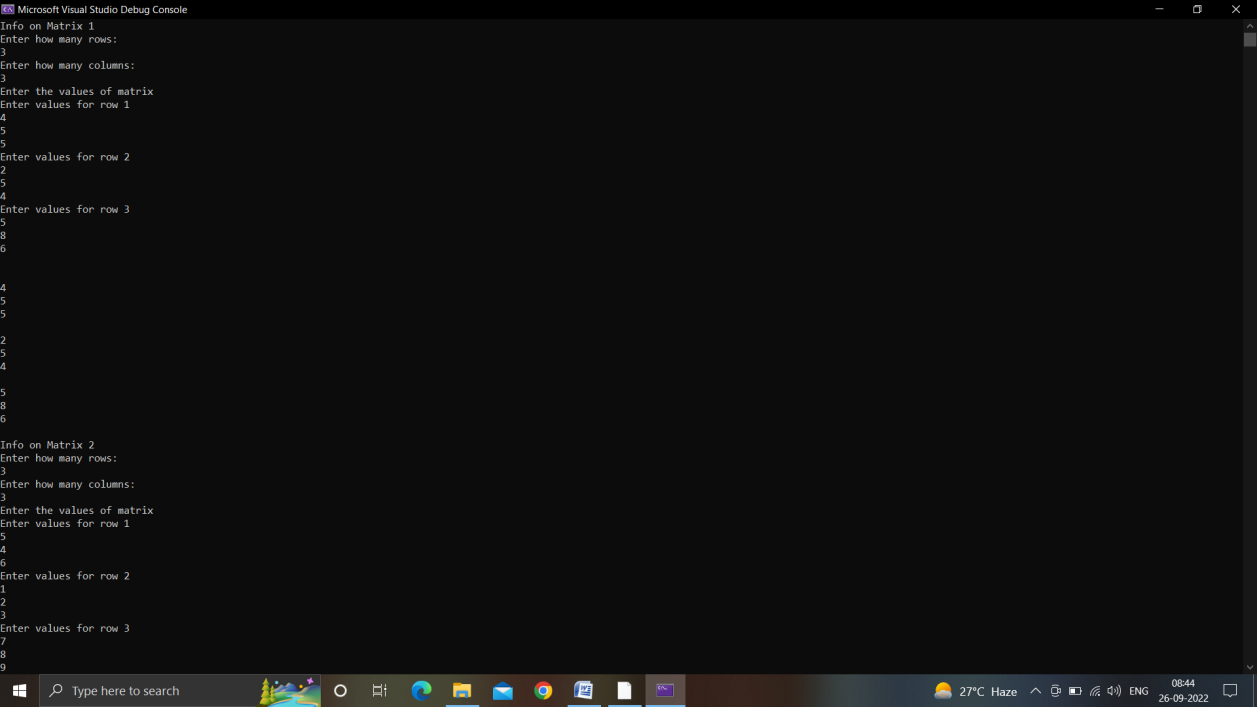
Part A: Accept the matrix of 3x3 from the user (2 matrix). Perform matrix multiplication

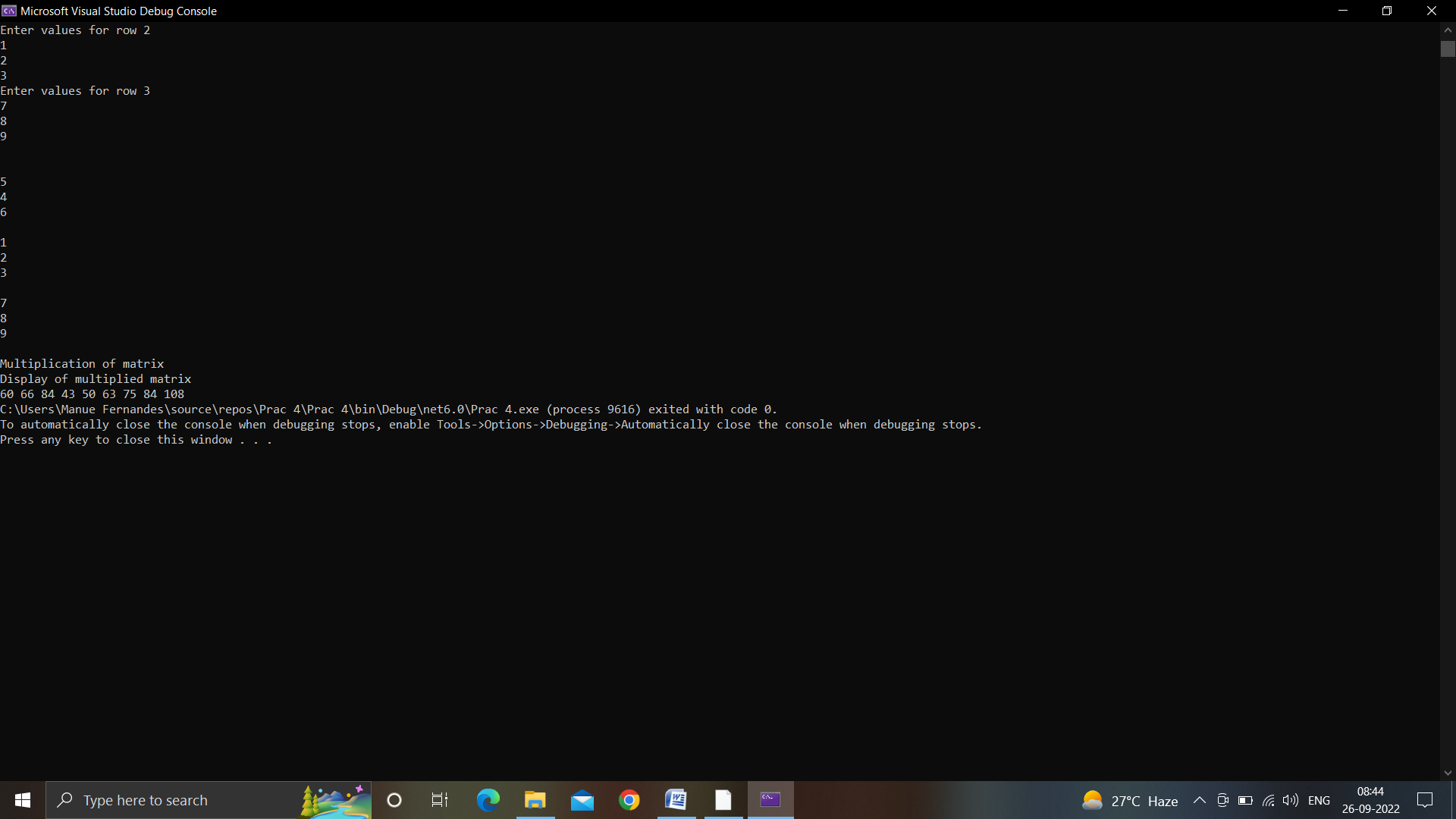
Part B: Accept 2 matrix from user of any dimension. Initially check whether the matrix compatible for multiplication and accordingly multiply

Part C: Find the difference between maximum and minimum from (Part b)

namespace MatrixMultiplication  
{  
    internal class Program  
    {  
        static void Main(string[] args)  
        {  
            Console.WriteLine("Info on Matrix 1");  
            Console.WriteLine("Enter how many rows: ");  
            int row = int.Parse(Console.ReadLine());  
            Console.WriteLine("Enter how many columns: ");  
            int col = int.Parse(Console.ReadLine());  
  
  
            //Taking input from user for filling matrix  
            int[,] mat = new int[row, col];  
            Console.WriteLine("Enter the values of matrix");  
            for(int i = 0; i < row; i++)  
            {  
                Console.WriteLine("Enter values for row " + (i + 1));  
                for (int j = 0; j < col; j++)  
                {  
                    mat[i, j] = int.Parse(Console.ReadLine());  
                }  
            }  
            Console.WriteLine("\n");  
  
            //Printing the matrix which user typed  
            for (int i = 0; i < row; i++)  
            {  
                for (int j = 0; j < col; j++)  
                {  
                    Console.WriteLine(mat[i,j]);  
                }  
                Console.WriteLine();  
            }  
            //////////////////////////////////////  
            Console.WriteLine("Info on Matrix 2");  
            Console.WriteLine("Enter how many rows: ");  
            int row1 = int.Parse(Console.ReadLine());  
            Console.WriteLine("Enter how many columns: ");  
            int col1 = int.Parse(Console.ReadLine());  
  
  
            //Taking input from user for filling matrix  
            int[,] mat2 = new int[row1, col1];  
            Console.WriteLine("Enter the values of matrix");  
            for (int i = 0; i < row1; i++)  
            {  
                Console.WriteLine("Enter values for row " + (i + 1));  
                for (int j = 0; j < col1; j++)  
                {  
                    mat2[i, j] = int.Parse(Console.ReadLine());  
                }  
            }  
            Console.WriteLine("\n");  
  
            //Printing the matrix which user typed  
            for (int i = 0; i < row1; i++)  
            {  
                for (int j = 0; j < col1; j++)  
                {  
                    Console.WriteLine(mat2[i, j]);  
                }  
                Console.WriteLine();  
            }  
  
            Console.WriteLine("Multiplication of matrix");  
            int[,] mulmatrix = new int[row, col1];  
            for(int i = 0; i < row; i++)  
            {  
                for(int j = 0; j < col1; j++)  
                {  
                    for(int k = 0; k < row1; k++)  
                    {  
                        mulmatrix[i, j] = mulmatrix[i, j] + (mat[i, k] \* mat2[k, j]);  
                    }        
                }  
            }  
  
            Console.WriteLine("Display of multiplied matrix");  
            for(int i = 0; i < row; i++)  
            {  
                for(int j = 0; j < col1; j++)  
                {  
                    Console.Write(mulmatrix[i, j] + " ");  
                }  
            }  
        }  
    }  
}

**Output:**

****



PRACTICAL 5:

1. Display the name and price of the products whose price is < 52,000
2. Display the name and price of product whose price is > 55,000 but < 65,000
3. Display all the details of products which is not a dell brand
4. Display all the details of products which HP or Dell brand but the price is < 65,000

**xml file:**

<?xml version="1.0" encoding="utf-8" ?>

<Products>

<Product>

<pid>100</pid>

<pname>dell</pname>

<price>50000</price>

<dom>09-02-17</dom>

</Product>

<Product>

<pid>200</pid>

<pname>HP</pname>

<price>70000</price>

<dom>11-02-19</dom>

</Product>

<Product>

<pid>300</pid>

<pname>Lenovo</pname>

<price>60000</price>

<dom>12-02-20</dom>

</Product>

</Products>

**Linqpracs.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Linq.Expressions;

using System.Xml.Linq;

namespace LINQDemo

{

internal class Linqpracs

{

static void Main()

{

XDocument pd = XDocument.Load("C:\\Users\\Hp\\source\\repos\\LINQDemo\\LINQDemo\\XMLFile1.xml");

var product = pd.Elements();

///Display all products///

Console.WriteLine("===Display all product===");

foreach (var item in product)

{

Console.WriteLine(item.ToString());

}

Console.WriteLine();

///Printing all products whose price is > 55000

Console.WriteLine("===Display all products whose price > 55000===");

var result = product.Elements("Product") //Product is row

.Where(x => (int)x.Element("price") > 55000) //each row is called by x

.Select(x =>

new

{

Name = x.Element("pname").Value,

price = x.Element("price").Value,

dom = x.Element("dom").Value,

}

);

foreach(var r in result)

{

Console.WriteLine(r.ToString());

}

Console.WriteLine();

//Display the name and price of the products whose price is < 52,000

Console.WriteLine("===Display all products whose price > 52000===");

var result2 = product.Elements("Product")

.Where(x => (int)x.Element("price") < 52000)

.Select(x =>

new

{

Name = x.Element("pname").Value,

price = x.Element("price").Value,

}

);

foreach (var r in result2)

{

Console.WriteLine(r.ToString());

}

Console.WriteLine();

//Display the name and price of products whose price is > 55000 but < 65000

Console.WriteLine("===Display all products whose price > 55000 but < 65000===");

var result3 = product.Elements("Product")

.Where(x => (int)x.Element("price") > 55000 && (int)x.Element("price") < 65000)

.Select(x =>

new

{

Name = x.Element("pname").Value,

price = x.Element("price").Value,

}

);

foreach (var r in result3)

{

Console.WriteLine(r.ToString());

}

Console.WriteLine();

//Display all the details of products which is not a dell brand

Console.WriteLine("===Display all the details of products which is not a dell brand===");

var result4 = product.Elements("Product")

.Where(x => (string)x.Element("pname") != "dell")

.Select(x =>

new

{

Name = x.Element("pname").Value,

price = x.Element("price").Value,

dom = x.Element("dom").Value,

}

);

foreach (var r in result4)

{

Console.WriteLine(r.ToString());

}

Console.WriteLine();

//Display all the details of products which is HP or Dell brand but the price is < 65000

Console.WriteLine("===Display all the details of products which is HP or Dell brand but the price is < 65000===");

var result5 = product.Elements("Product")

.Where((x => (string)x.Element("pname") == "dell" || (string)x.Element("pname") == "HP" && (int)x.Element("price") < 65000))

.Select(x =>

new

{

Name = x.Element("pname").Value,

price = x.Element("price").Value,

dom = x.Element("dom").Value,

}

);

foreach (var r in result5)

{

Console.WriteLine(r.ToString());

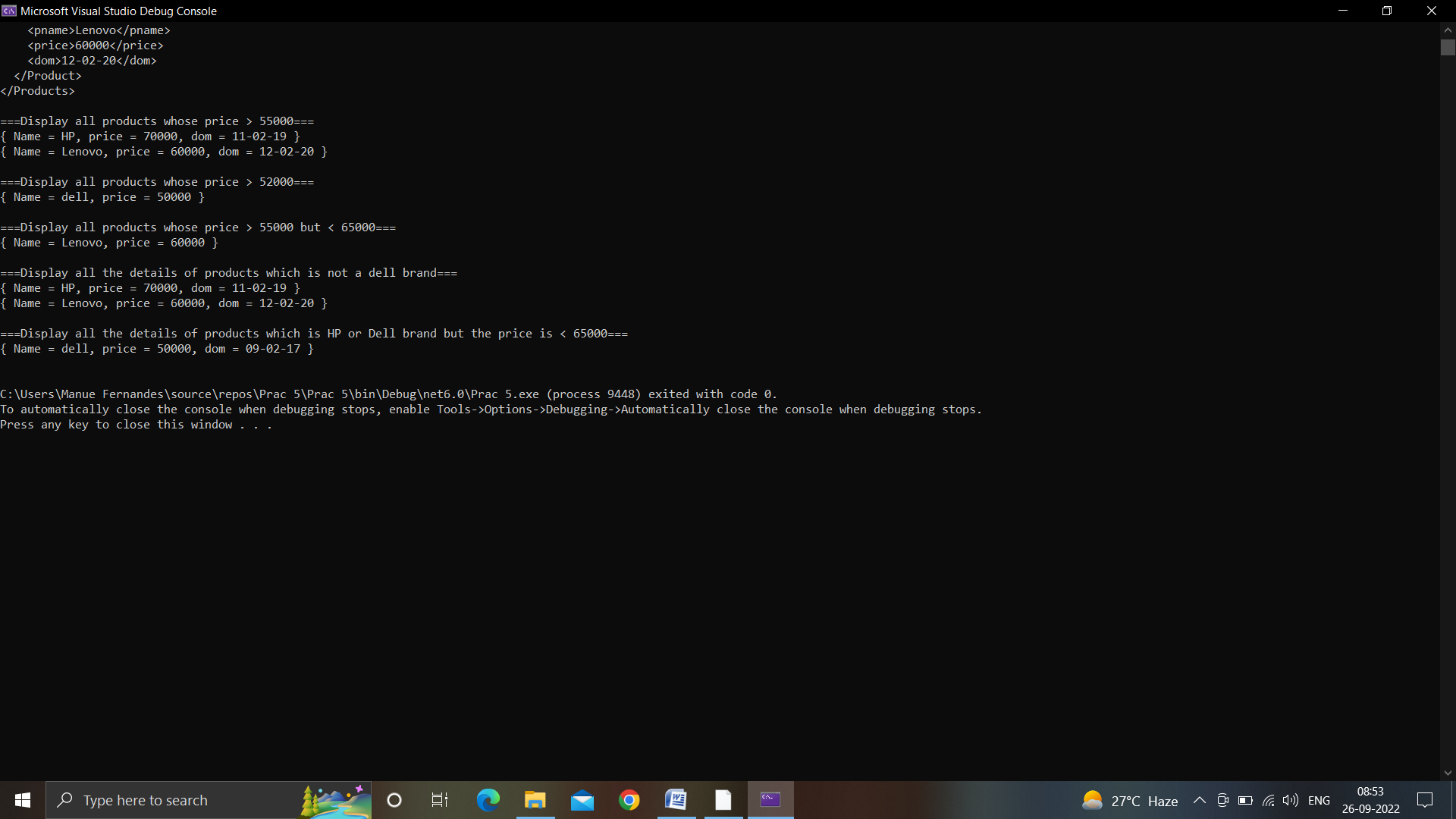
}

Console.WriteLine();

}

}

}



PRACTICAL 6:

**XML file**

<?xml version="1.0" encoding="utf-8" ?>

<Employees>

<Employee>

<emp\_no>1</emp\_no>

<emp\_name>Ankit</emp\_name>

<emp\_salary>20000</emp\_salary>

<emp\_dept>IT</emp\_dept>

</Employee>

<Employee>

<emp\_no>2</emp\_no>

<emp\_name>Salvin</emp\_name>

<emp\_salary>50000</emp\_salary>

<emp\_dept>Sales</emp\_dept>

</Employee>

<Employee>

<emp\_no>3</emp\_no>

<emp\_name>Kriti</emp\_name>

<emp\_salary>20000</emp\_salary>

<emp\_dept>Production</emp\_dept>

</Employee>

<Employee>

<emp\_no>4</emp\_no>

<emp\_name>Priti</emp\_name>

<emp\_salary>70000</emp\_salary>

<emp\_dept>Sales</emp\_dept>

</Employee>

<Employee>

<emp\_no>5</emp\_no>

<emp\_name>Rochelle</emp\_name>

<emp\_salary>60000</emp\_salary>

<emp\_dept>Design</emp\_dept>

</Employee>

<Employee>

<emp\_no>6</emp\_no>

<emp\_name>Esther</emp\_name>

<emp\_salary>30000</emp\_salary>

<emp\_dept>Sales</emp\_dept>

</Employee>

<Employee>

<emp\_no>7</emp\_no>

<emp\_name>Vivek</emp\_name>

<emp\_salary>55000</emp\_salary>

<emp\_dept>IT</emp\_dept>

</Employee>

<Employee>

<emp\_no>8</emp\_no>

<emp\_name>Yojit</emp\_name>

<emp\_salary>90000</emp\_salary>

<emp\_dept>Production</emp\_dept>

</Employee>

<Employee>

<emp\_no>9</emp\_no>

<emp\_name>Nigel</emp\_name>

<emp\_salary>35000</emp\_salary>

<emp\_dept>Production</emp\_dept>

</Employee>

<Employee>

<emp\_no>10</emp\_no>

<emp\_name>Elson</emp\_name>

<emp\_salary>20000</emp\_salary>

<emp\_dept>IT</emp\_dept>

</Employee>

</Employees>

**Program.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Linq.Expressions;

using System.Xml.Linq;

namespace DetailsOfEmployees

{

internal class Program

{

static void Main(string[] args)

{

XDocument emp = XDocument.Load("C:\\Users\\Hp\\source\\repos\\DetailsOfEmployees\\DetailsOfEmployees\\XMLFile1.xml");

var employee = emp.Elements();

///Display all products///

Console.WriteLine("===Display all Employees===");

foreach (var item in employee)

{

Console.WriteLine(item.ToString());

}

Console.WriteLine();

///Display the name and salary of all employees who work for production department///

Console.WriteLine("===Display name and salary of all employees in production department===");

var result = employee.Elements("Employee")

.Where(x => (string)x.Element("emp\_dept") == "Production")

.Select(x =>

new

{

Name = x.Element("emp\_name").Value,

Salary = x.Element("emp\_salary").Value,

});

foreach (var r in result)

{

Console.WriteLine(r.ToString());

}

Console.WriteLine();

///Display the combined salary of each dept along with dept name///

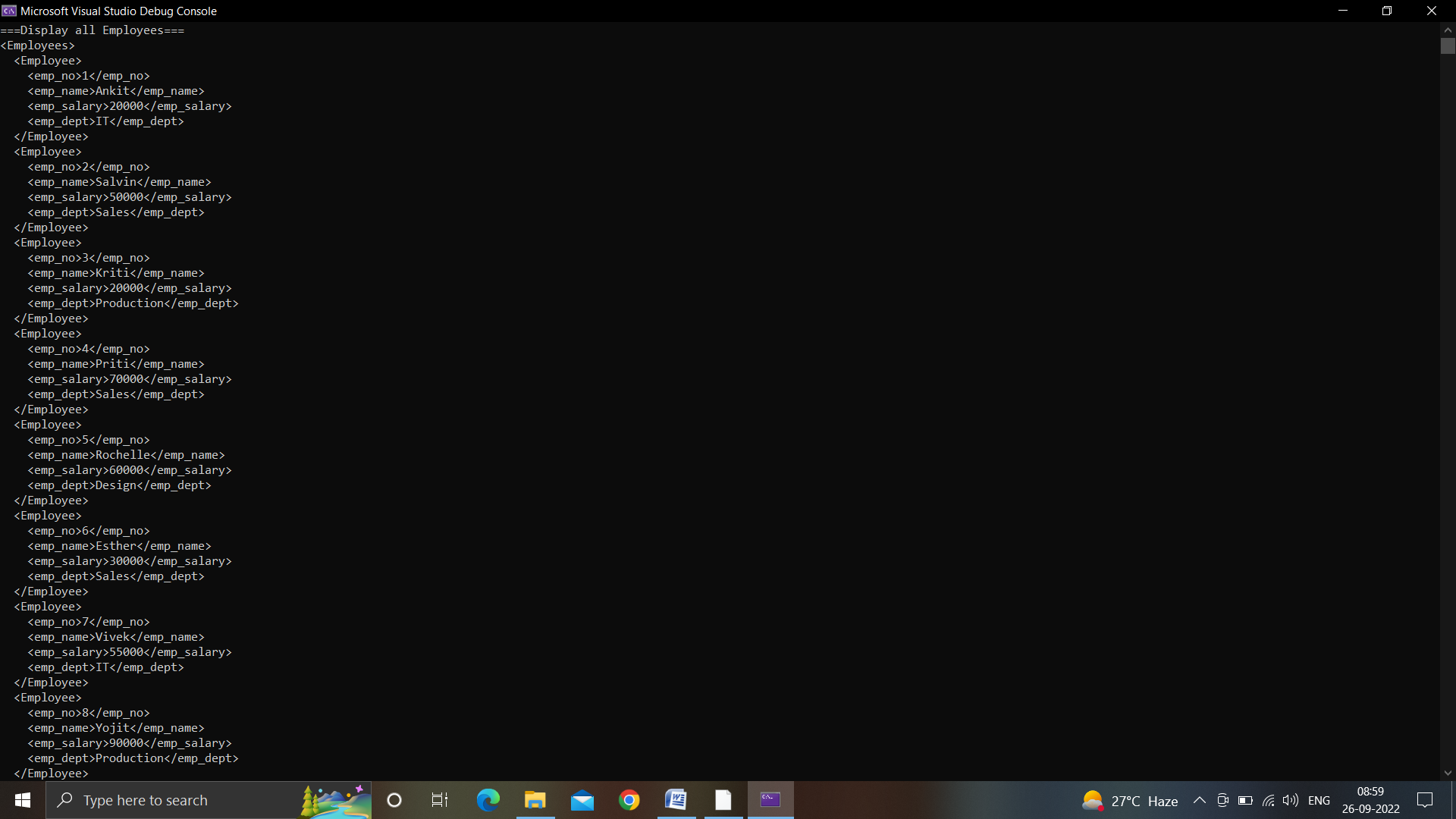
Console.WriteLine("===Display the combined salary of each dept along with dept name===");

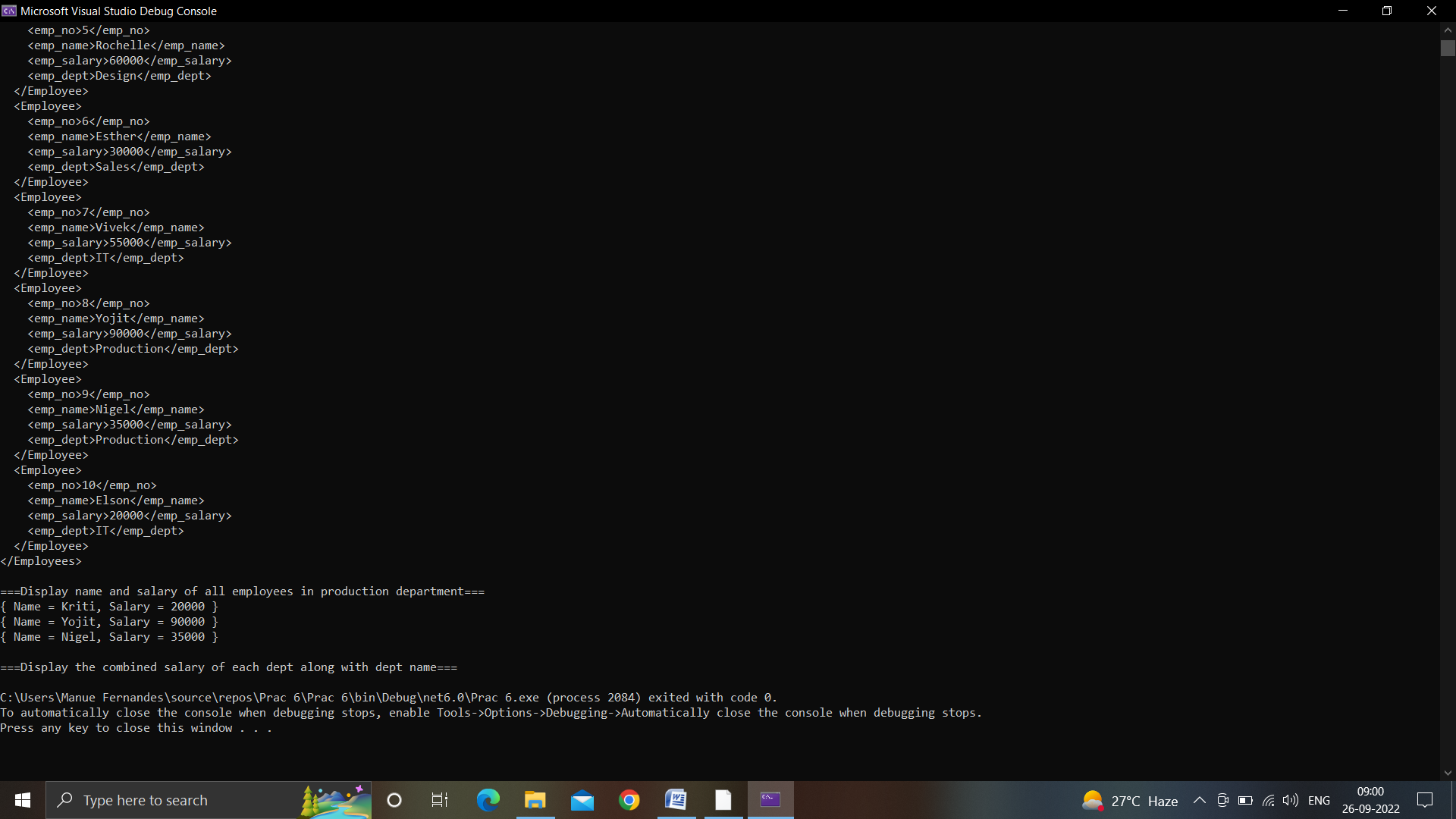
}

}

}

**Output:**

****



PRACTICAL 7:

1. ‘dbdemo’ name

2. Create a folder ‘models’ in dbdemo,

right click on dbdemo -> add -> new folder-> ‘models’ (name)

3. Right click models -> add -> new -> class -> ‘employee’ (name)

4. Write the line of codes in ‘employee’ class.

5. Right click on pages -> Add -> folder -> ‘employeeS’ (name)

6. Right click on ‘employeeS’ -> Add -> razor page -> using 3rd option [CRUD] -> Add

7. For adding razor page we need one library and one command

Library will be installed from follow steps:

right click dbdemo -> manage nuget packages -> in browse type core design

[Select -> Microsoft.EntityFrameworkCore.Design] install this.

command: go to tools -> nuget package manager -> package manager console

8. Now a small console window will open, write these 2 command lines one by one:

1. add-migration InitData

2. update-database

[one by one, run each command line]

9. Now run ‘dbdemo’.

10. An URL on your web browser will open up, let "/employees" be the endpoint of your URL (local host address), once you run this, you'll see your page (web form).

11. Now when you update your code (by either increasing your no of rows or implementing validation into it) you'll first go for razor page step then command line step

12. This web formation has 2 steps:

1. UI formation using razor page.

2. database using command lines.

Remember \*\*

When you are making the web page for the first time:

You'll make UI first and then add the database, whereas while updating your webpage, first update the database (command lines) then your UI (razor page).

"Add-migration InitData": every time you update your site and write command line

change name of file

i.e.

InitData2, InitData3, ..., InitData(n)

**Code:**

using System.ComponentModel.DataAnnotations;

namespace WebApplication2.Models

{

    public class Employee

    {

        [Required(ErrorMessage="Required")]

        //[Required(ErrorMessage ="Enter valid data")]

        public int id { get; set; } //generating dynamically

        public string Name { get; set; }

        public int Salary { get; set; }

        public string Dept { get; set; }

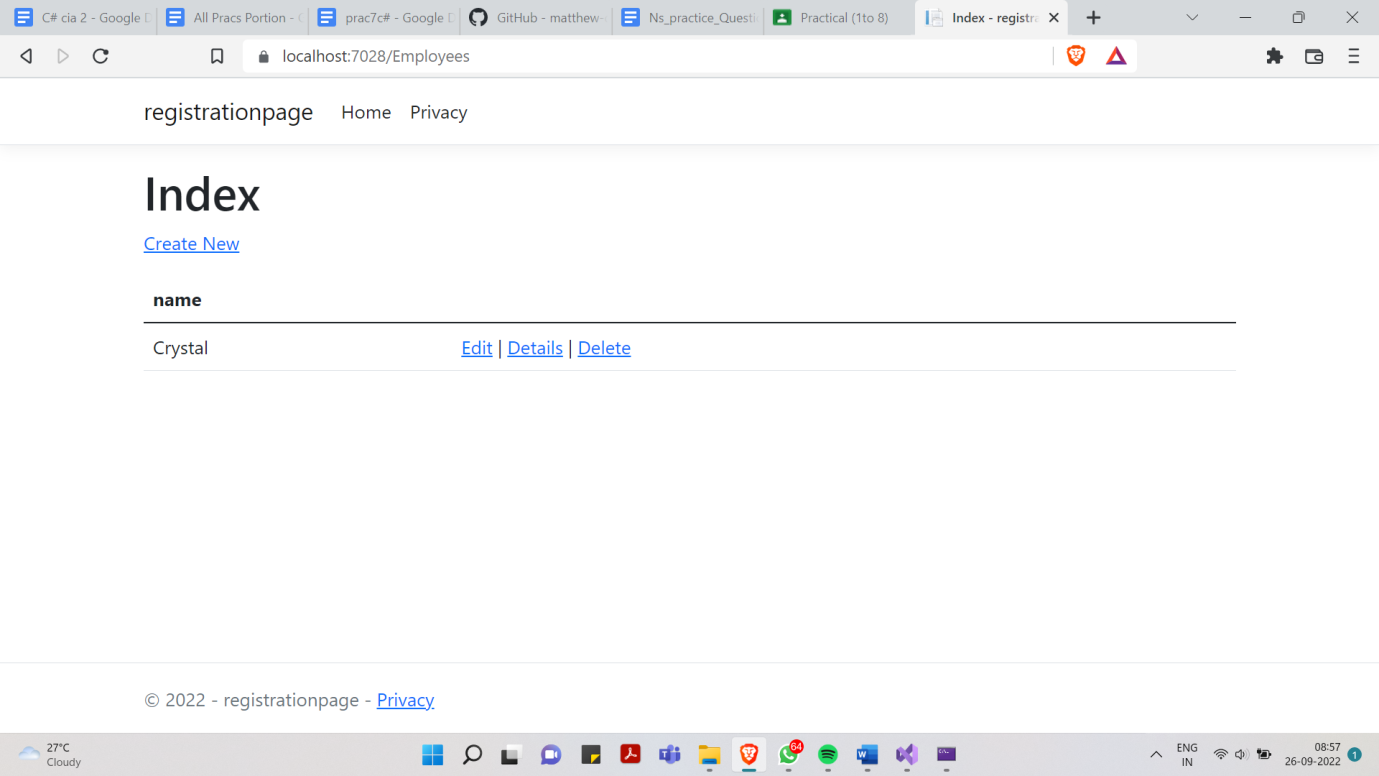
        //[EmailAddress]

        //public int Email { get; set; }

    }

}

**Output:**

****

PRACTICAL 8:

Code:

using System.ComponentModel.DataAnnotations;

namespace registration.Models

{

    public class Employee

    {

        [Key]

        [Required]

        public int UID { get; set; }

        [Required]

        [RegularExpression("^[A-Za-z]{1,20}$", ErrorMessage = "Please enter correct name")]

        public string Name { get; set; }

        [Required]

        [RegularExpression("^[6-9][0-9]{9}$", ErrorMessage = "Please enter 10 digit phone number")]

        public string PhoneNumber { get; set; }

        [Required]

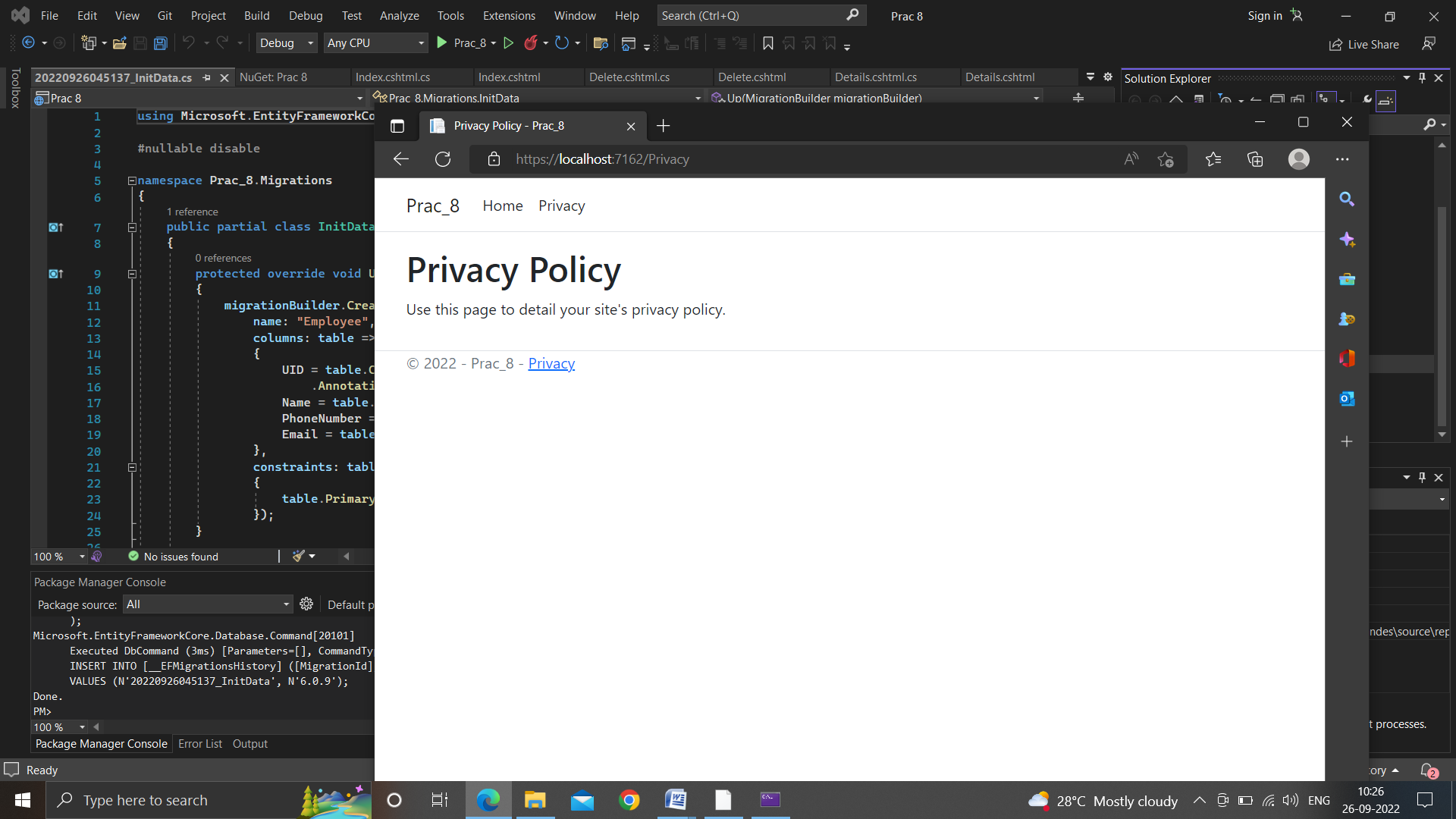
        [RegularExpression("^[a-z0-9+\_.]+@[a-z0-9.]+$", ErrorMessage = "Please enter correct email address")]

        public string Email { get; set; }

    }

}

Output:



PRACTICAL 8: