**EASY SECTION:**

**1)Create a C# console application which has a class with name “EmployeePromotion” that will**

**take employee names in the order in which they are eligible for promotion.**

using System;

using System.Collections.Generic;

using System.Text;

namespace CollectionEmployeeAssignment

{

class EmployeePromotion

{

/// <summary>

/// After taking a list and names are added in the list

/// </summary>

List<String> EmployeeNames = new List<String>();

public void EmployeePromotionList()

{

Console.WriteLine("Enter the Employee names");

String input= Console.ReadLine();

if (!string.IsNullOrEmpty(input))

{

EmployeePromotionList();

EmployeeNames.Add(input);

//PrintEmployee();

}

}

/// <summary>

/// prints the names of the employee in the list

/// </summary>

public void PrintEmployee()

{

Console.WriteLine("after adding all employee");

foreach (var user in EmployeeNames)

{

Console.WriteLine(user);

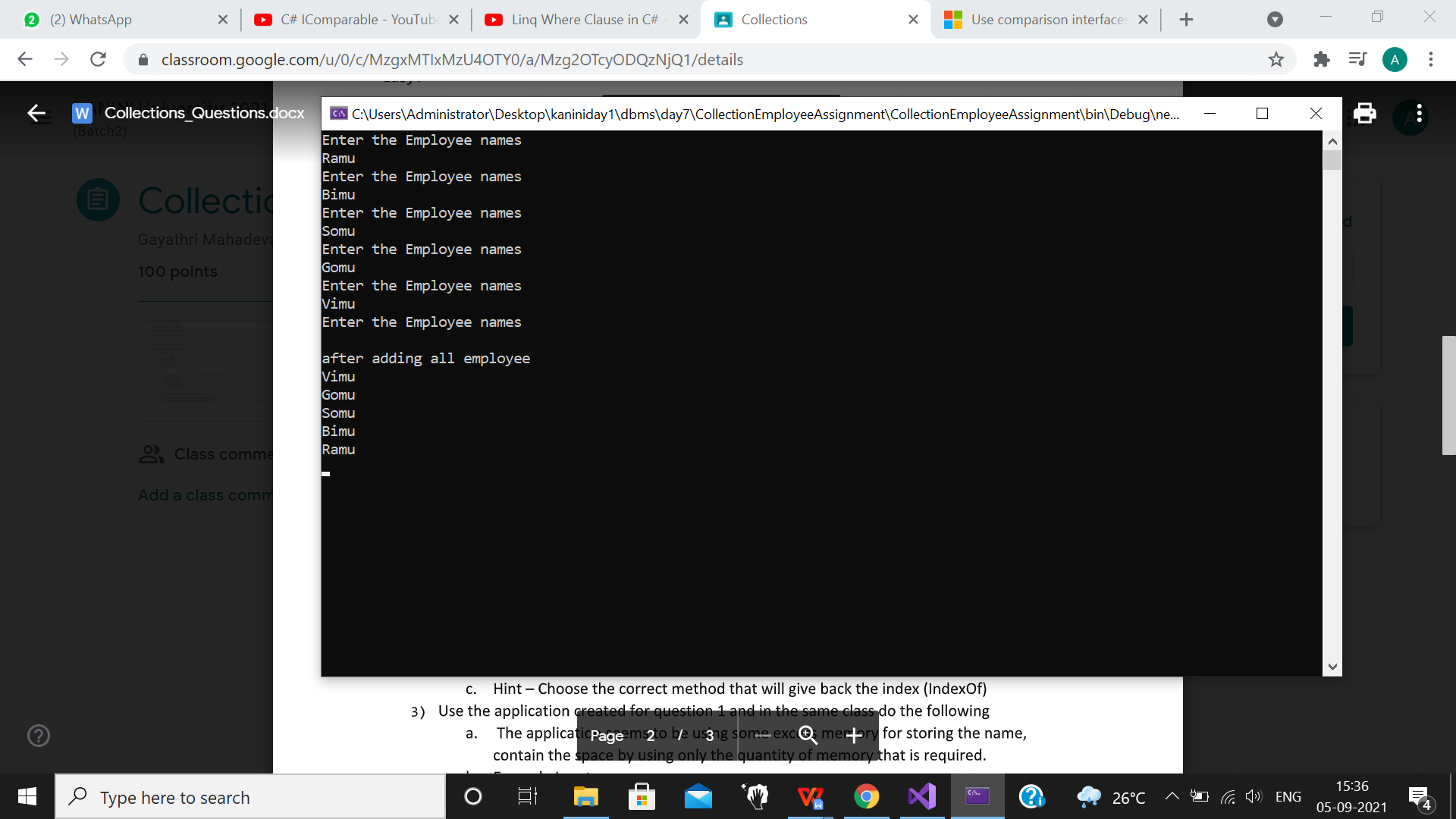
}

}

}

}

Output:



**2)Use the application created for question 1 and in the same class do the following**

**a. Given an employee name find his position in the promotion list**

/// <summary>

/// prints the position of the given employee with employee name

/// </summary>

public void PrintPostion()

{

Console.WriteLine("Please enter the name of the employee to check promotion position");

string EmpNames = Console.ReadLine();

for (int i = 0; i < EmployeeNames.Count; i++)

{

if (EmpNames == EmployeeNames[i])

{

int index1 = EmployeeNames.IndexOf(EmployeeNames[i]);

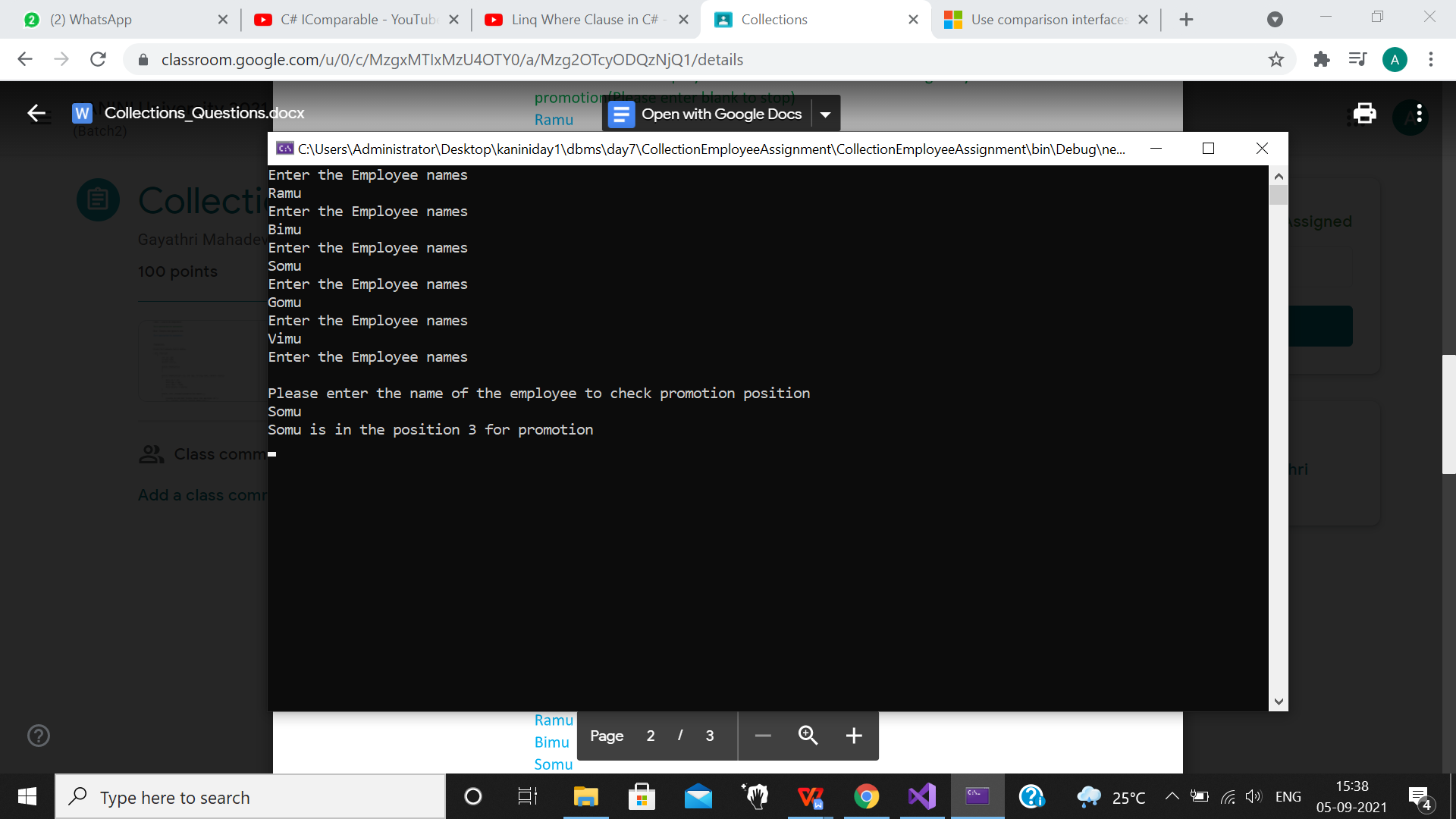
Console.WriteLine($"{EmployeeNames[i]} is in the position {index1+1} for promotion");

}

}

}

Output:



**3)Use the application created for question 1 and in the same class do the following**

**a. The application seems to be using some excess memory for storing the name,**

**contain the space by using only the quantity of memory that is required.**

/// <summary>

/// prints the employee capacity and after trimming the excess memory

/// </summary>

public void RemoveExcessMemory()

{

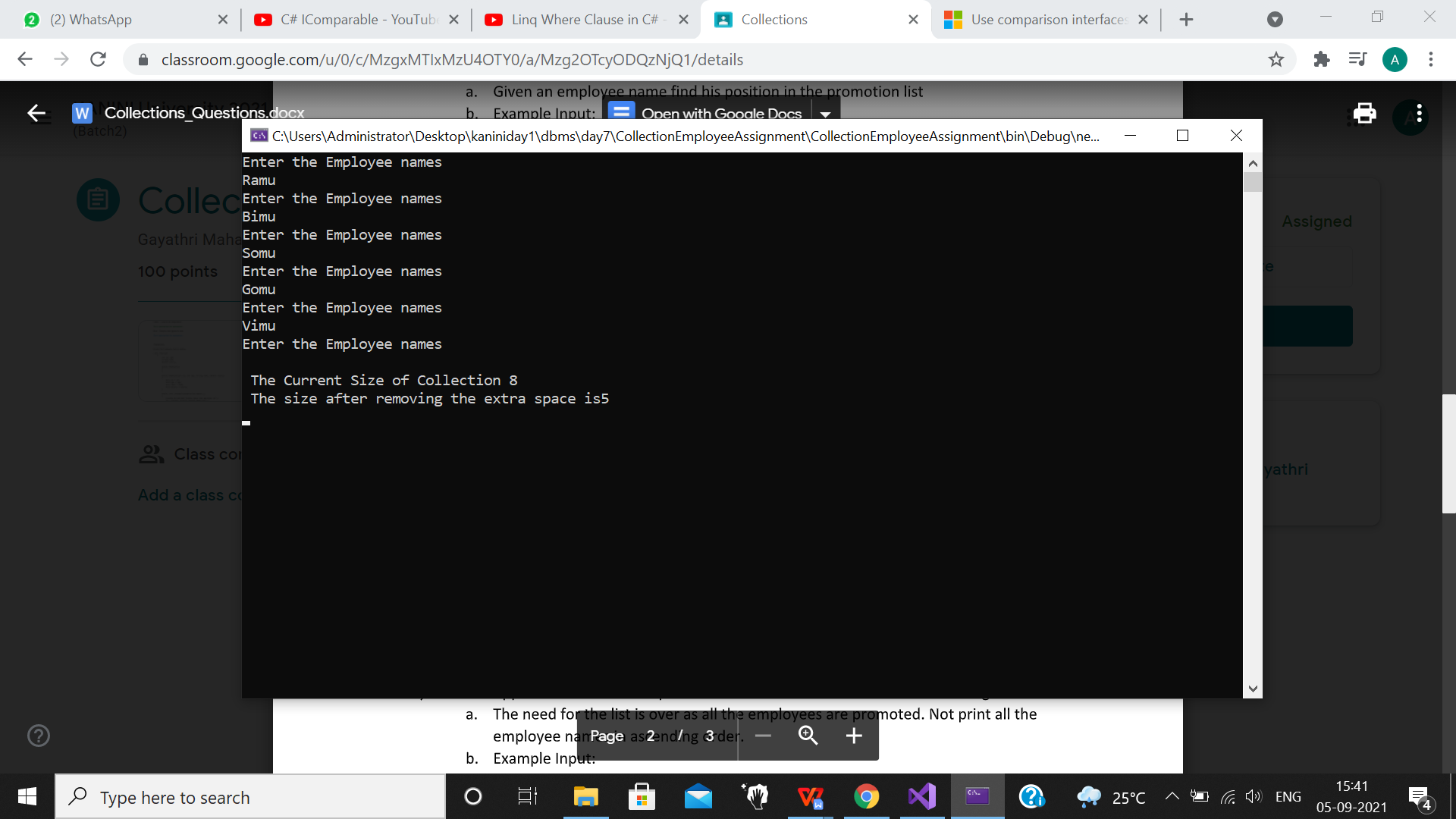
Console.WriteLine(" The Current Size of Collection " + EmployeeNames.Capacity);

EmployeeNames.TrimExcess();

Console.WriteLine(" The size after removing the extra space is" + EmployeeNames.Capacity);

}

Output:



**4)Use the application created for question 1 and in the same class do the following**

**a. The need for the list is over as all the employees are promoted. Not print all the**

**employee names in ascending order.**

/// <summary>

/// prints the ascending order of names of the employee in the list

/// </summary>

public void AscendingOrderSorting()

{

EmployeeNames.Sort();

Console.WriteLine("promoted employee list:");

foreach (var item in EmployeeNames)

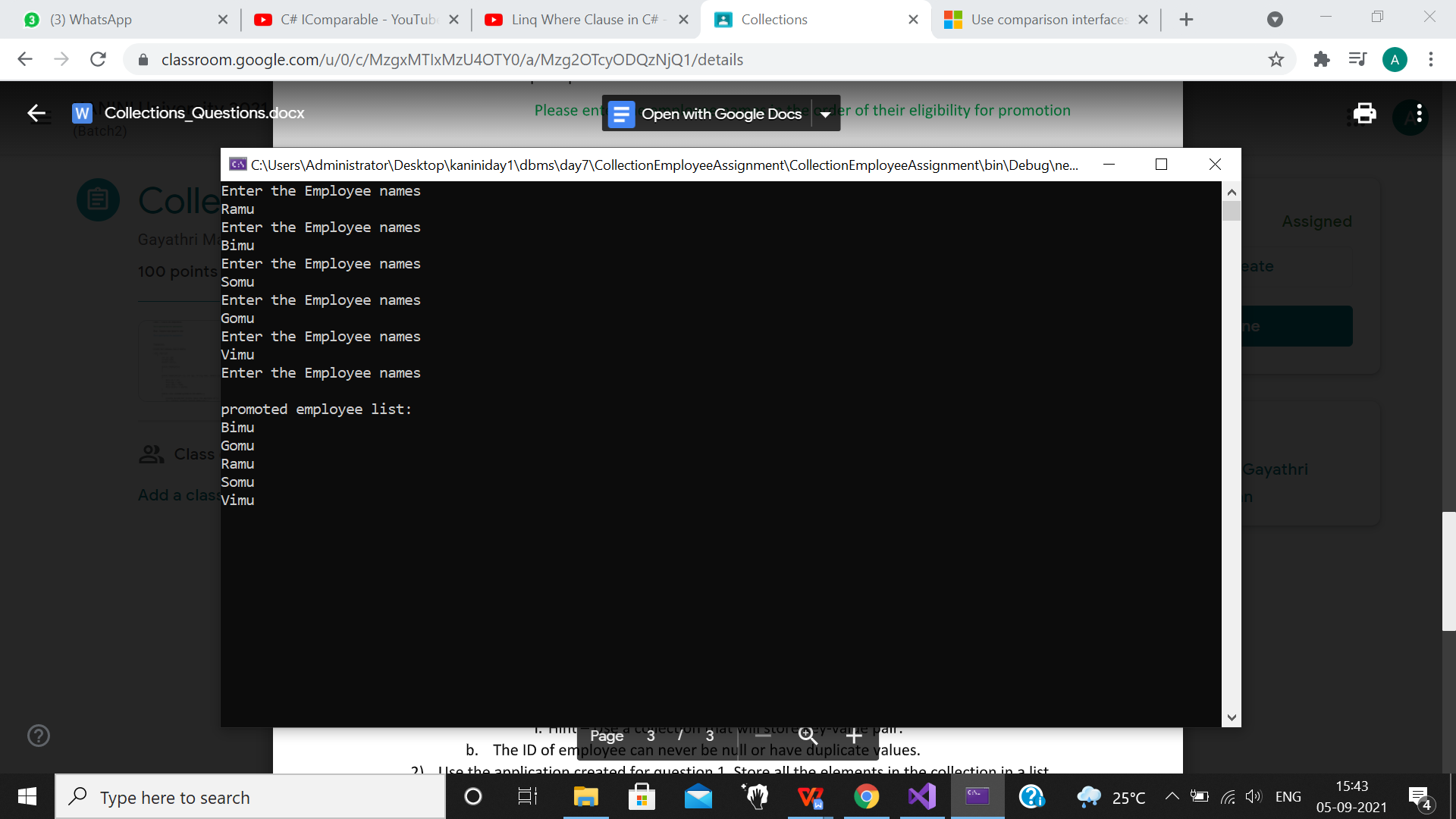
{

Console.WriteLine(item);

}

}

Output:



**------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------**

**MEDIUM PART:**

**Userchoice taking:**

/// <summary>

/// providing the choices for the user

/// </summary>

public void EmployeeInformation()

{

string UserChoices;

do

{

Console.WriteLine("Please Enter The following Character To Add and Display The Employee Details");

Console.WriteLine("key in 'a' for AddEmployee and 'p' for print employee details and 'S' for sorting");

Console.WriteLine("Key in 'i' to print employee details by id");

Console.WriteLine("Key in 'n' to print employee details by name");

Console.WriteLine("Key in 'el' to print employee details by elder");

Console.WriteLine("Key in e to exit");

UserChoices = Console.ReadLine().ToLower();

switch (UserChoices)

{

case "a":

EmployeeDetailStoring();

break;

case "p":

PrintEmployee();

break;

case "s":

EmployeeSorting();

break;

case "i":

PrintEmployeeDetailsById();

break;

case "n":

PrintEmployeeDetailsByName();

break;

case "el":

ElderThanCurrentEmploye();

break;

case "e":

Console.WriteLine("we are done creating accounts");

break;

default:

Console.WriteLine("Ïnvalid option, please try again");

break;

}

} while (UserChoices != "e");

}

**1)Create an application that will take employee details (Use the employee class) and store it in**

**a collection**

**a. The collection should be able to give back the employee object if the employee id is**

**provided.**

**i. Hint – Use a collection that will store key-value pair.**

**b. The ID of employee can never be null or have duplicate values.**

/// <summary>

/// A dictionary is taken and a list is taken for storing the employee details

/// employee stored in the dictionary using the key value id

/// </summary>

public class EmployeeDetailsCollection

{

Dictionary<int, Employee> employee;

List<Employee> sortedbysalary;

public EmployeeDetailsCollection()

{

employee = new Dictionary<int, Employee>();

//addemployee = new List<Employee>();

sortedbysalary = new List<Employee>();

}

public void EmployeeDetailStoring()

{

bool flag = true;

Employee newEmployeeDetails;

do

{

newEmployeeDetails = new Employee();

try

{

newEmployeeDetails.TakeEmployeeDetailsFromUser();

employee.Add(newEmployeeDetails.Id, newEmployeeDetails);

flag = false;

}

catch (Exception e)

{

//addemployee.RemoveAt(addemployee.Count - 1);

Console.WriteLine(e.Message);

flag = true;

}

} while (flag);

sortedbysalary = employee.Values.ToList();

//sortedbysalary.Sort();

}

/// <summary>

/// prints the employee details stored in dictionary

/// </summary>

public void PrintEmployee()

{

/\*for (int i = 0; i < employee.Count; i++)

{

if (EmpId ==)

{

int index1 = EmployeeNames.IndexOf(EmployeeNames[i]);

Console.WriteLine($"{EmployeeNames[i]} is in the position {index1 + 1} for promotion");

}\*/

foreach (var item in employee)

{

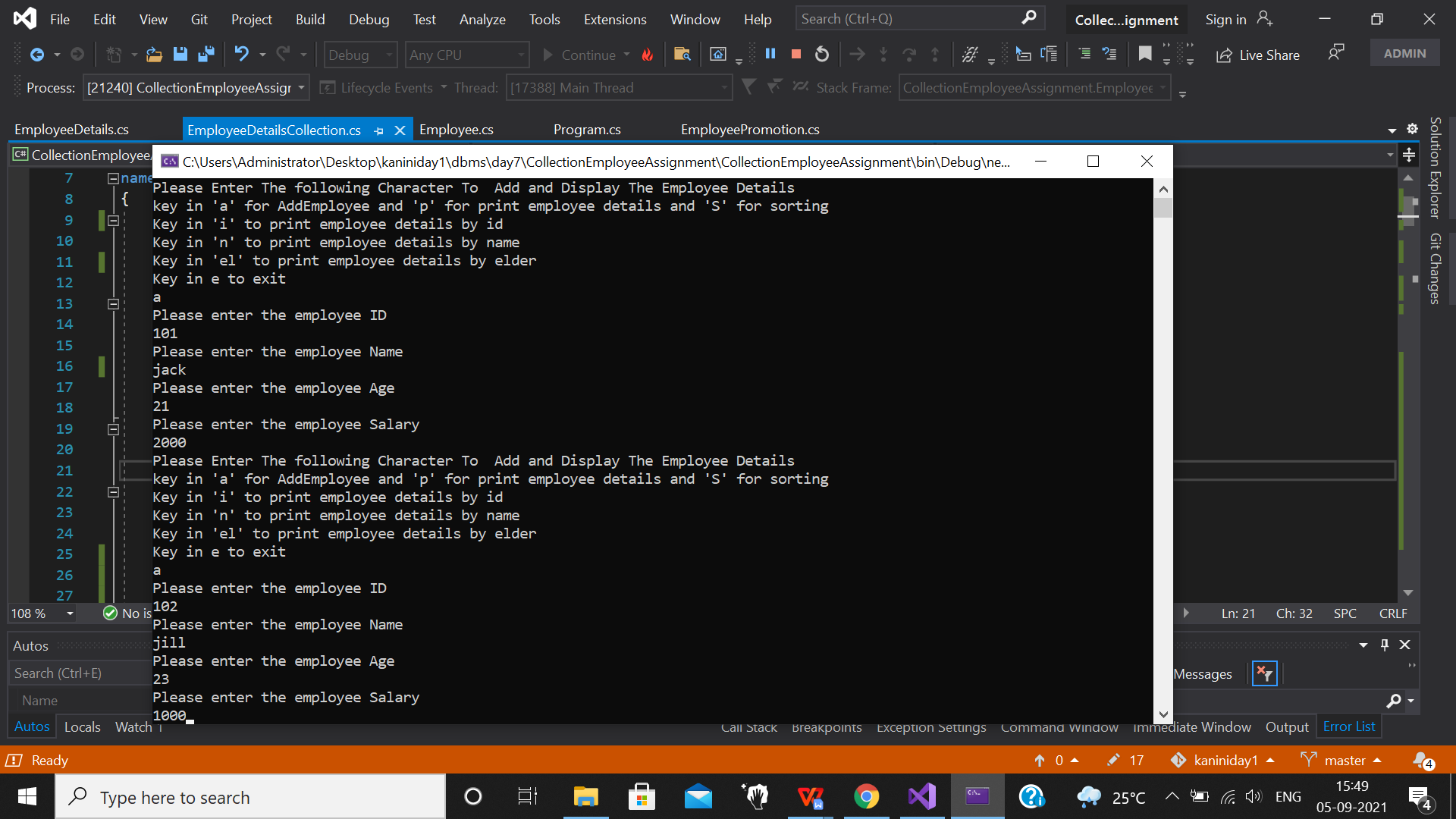
Console.WriteLine(item);

}

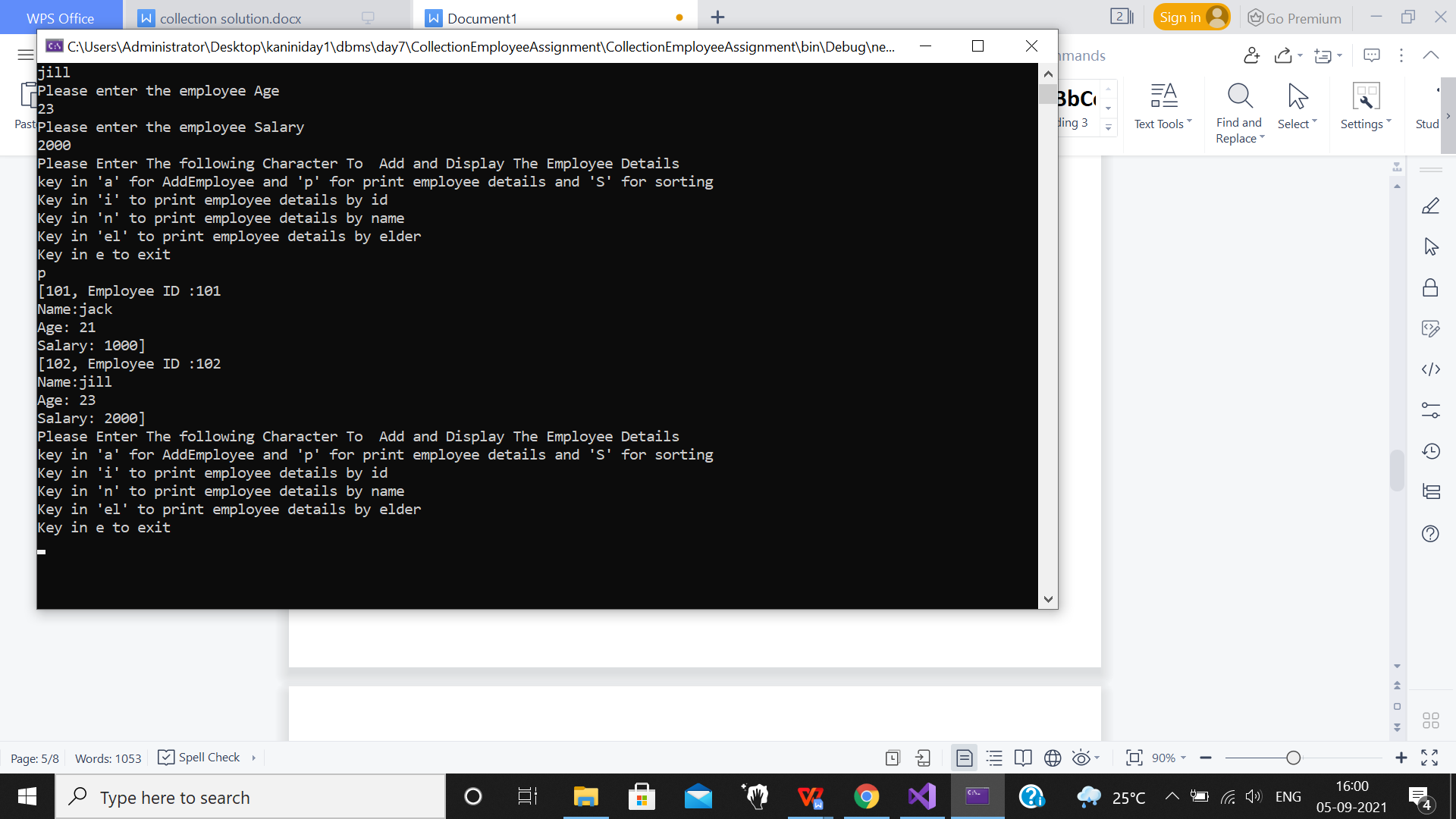
}

}

Output:



Printing employee details:



**2)Use the application created for question 1. Store all the elements in the collection in a list.**

**a. Sort the employees based on their salary.**

**i. Hint – Implement the IComparable interface in the Employee class.**

/// <summary>

/// The id of the employee is taken and compared with stored employee details

/// Employees are sorted by using the IComparable interface and printed

/// </summary>

public void EmployeeCompair()

{

Console.WriteLine("Enter the id :");

int EmpId =Convert.ToInt32( Console.ReadLine());

foreach (var item in employee)

{

if (item.Key == EmpId)

{

Console.WriteLine(item);

}

}

}

public void EmployeeSorting()

{

//addemployee.foreach (Employee => Console.WriteLine(Employee.sal)) ;

sortedbysalary.Sort();

foreach (var item in sortedbysalary)

{

PrintEmployeeDetails(item);

}

}

void PrintEmployeeDetails(Employee item)

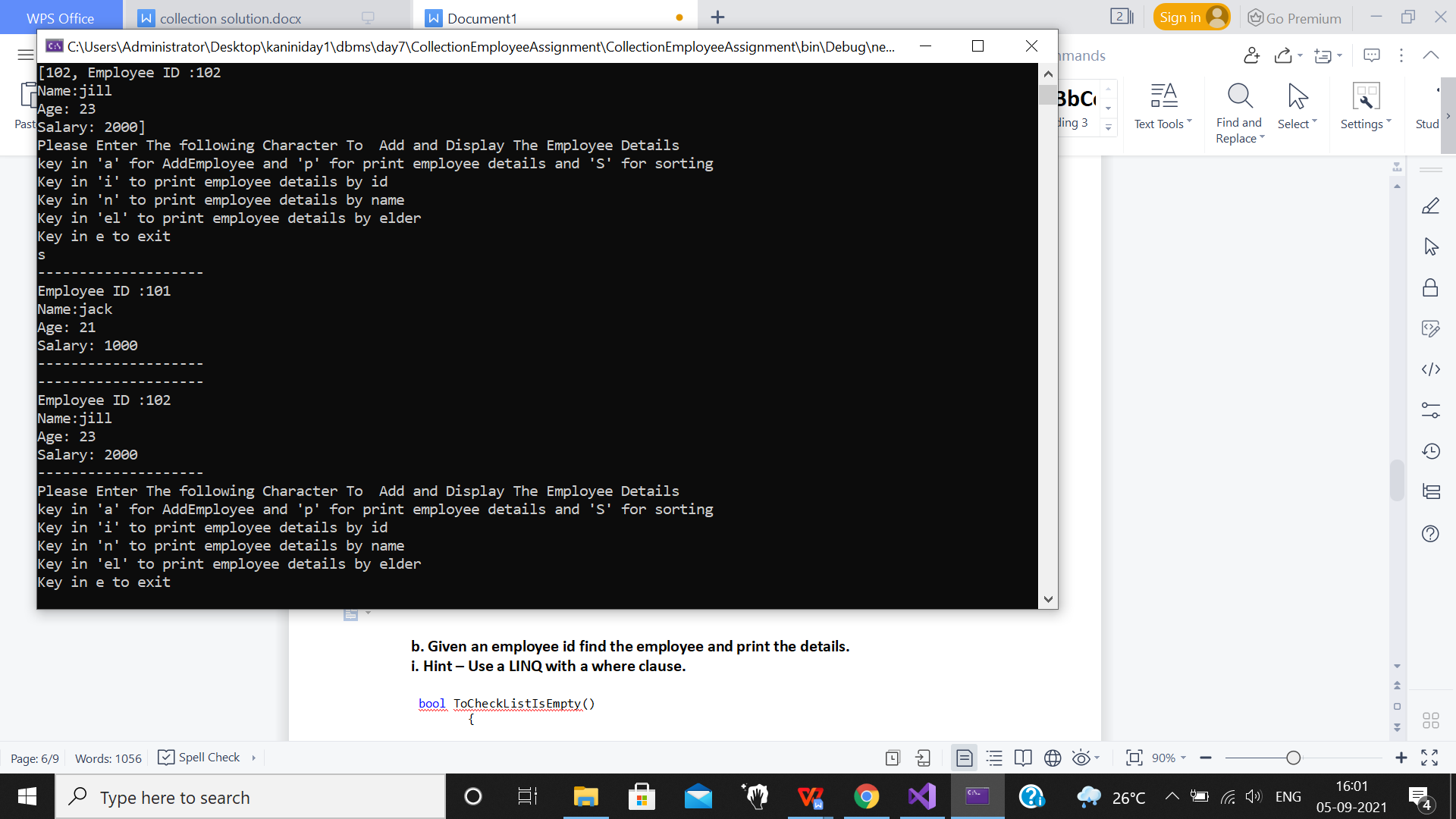
{

Console.WriteLine("--------------------");

Console.WriteLine(item);

Console.WriteLine("--------------------");

}



**b. Given an employee id find the employee and print the details.**

**i. Hint – Use a LINQ with a where clause.**

/// <summary>

/// First checked list is empty or not, then after that user input for id is

/// taken. Employee details are printed by LINQ and where clause

/// </summary>

bool ToCheckListIsEmpty()

{

if (employee.Count == 0)

{

Console.WriteLine("List is empty");

return false;

}

else

return true;

}

int UserIntInput()

{

bool flag;

int id = 0;

do

{

try

{

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

flag = false;

}

catch (Exception e)

{

Console.WriteLine(e.Message);

Console.WriteLine("Enter id");

flag = true;

}

} while (flag);

return id;

}

public void PrintEmployeeDetailsById()

{

if (ToCheckListIsEmpty())

{

Console.WriteLine("Enter employee id:");

int id = UserIntInput();

var employeeDetailsById = from obj in sortedbysalary

where obj.Id == id

select obj;

if (employeeDetailsById.Count() == 0)

Console.WriteLine("No employee with given id");

else

{

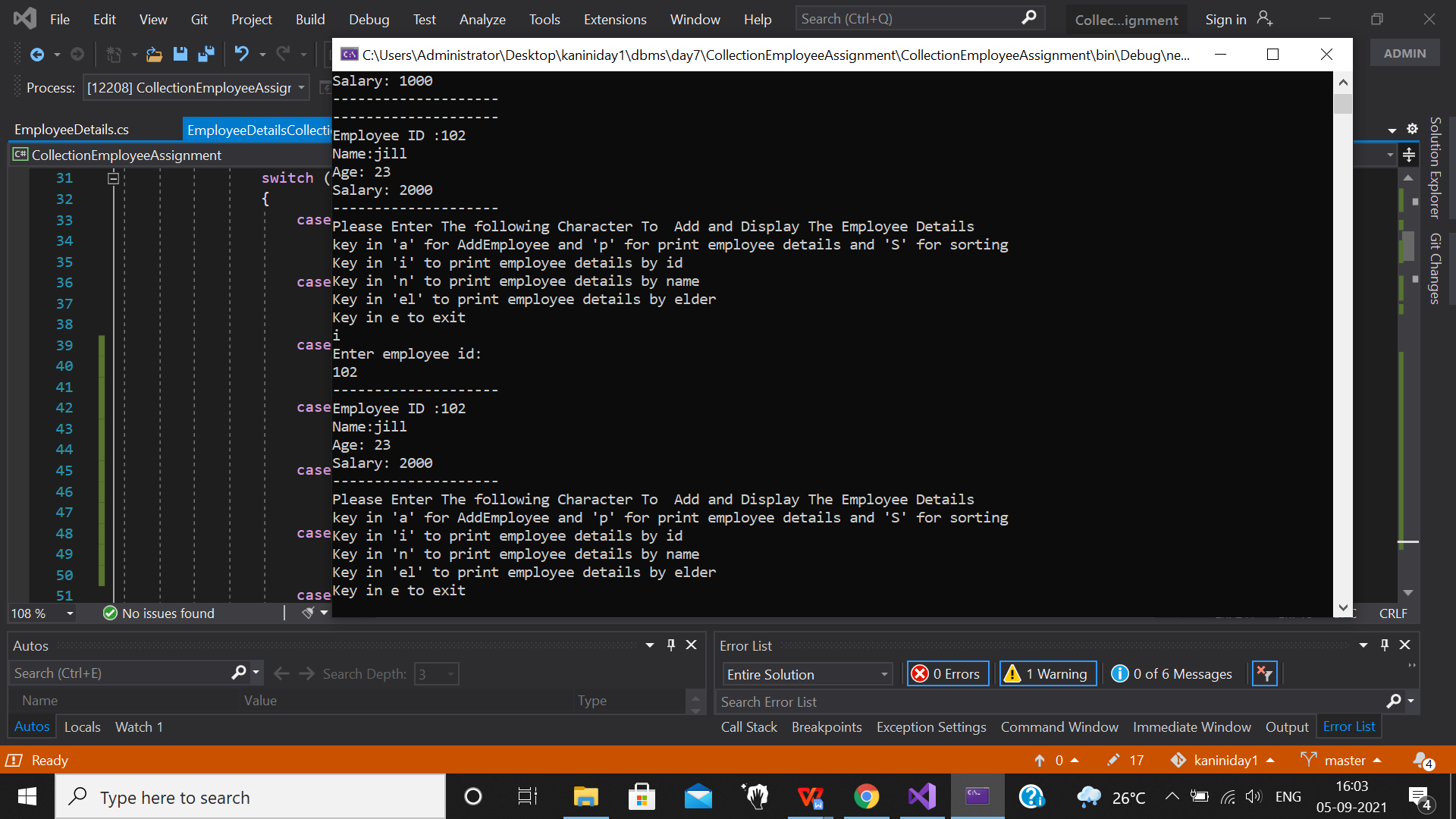
foreach (var item in employeeDetailsById)

PrintEmployeeDetails(item);

}

}

}



1. **Use the application created for question 2. Find all the employees with the given name**

**(Name to be taken from user)**

/// <summary>

/// Employee details are printed by taking the name of the employee

/// </summary>

public void PrintEmployeeDetailsByName()

{

if (ToCheckListIsEmpty())

{

Console.WriteLine("Enter employee name");

string employeeName = Console.ReadLine();

var employeeDetailsByName = from obj in sortedbysalary

where obj.Name == employeeName

select obj;

if (employeeDetailsByName.Count() == 0)

Console.WriteLine($"No employe with name {employeeName}");

else

{

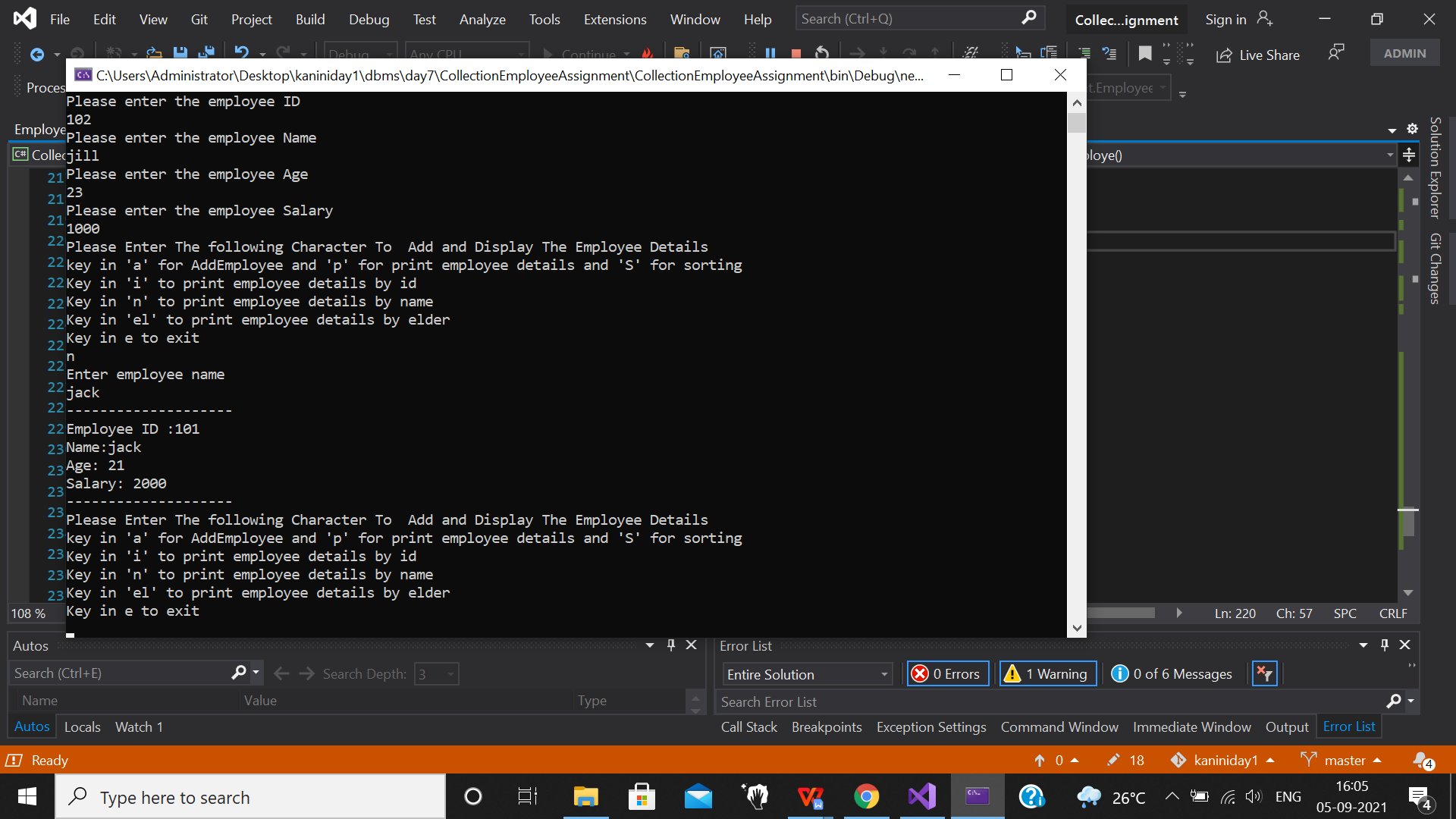
foreach (var item in employeeDetailsByName)

PrintEmployeeDetails(item);

}

}

}



**4)Use the application created for question 3. Find all the employees who are elder than a**

**given employee (Employee given by user)**

/// <summary>

/// Printing the employee is elder elder or not by comparing the age of the

/// Employee and details are printed by LINQ and where clause

/// </summary>

public void ElderThanCurrentEmploye()

{

if (ToCheckListIsEmpty())

{

int count = 0, employeeAge = 0;

Console.WriteLine("Enter employee name");

string employeeName = Console.ReadLine();

foreach (var item in sortedbysalary)

{

if (item.Name == employeeName)

{

employeeAge = item.Age;

count++;

break;

}

}

if (count == 0)

Console.WriteLine($"No employee with name {employeeName}");

else

{

var employeeDetails = from obj in sortedbysalary

where obj.Age > employeeAge

select obj;

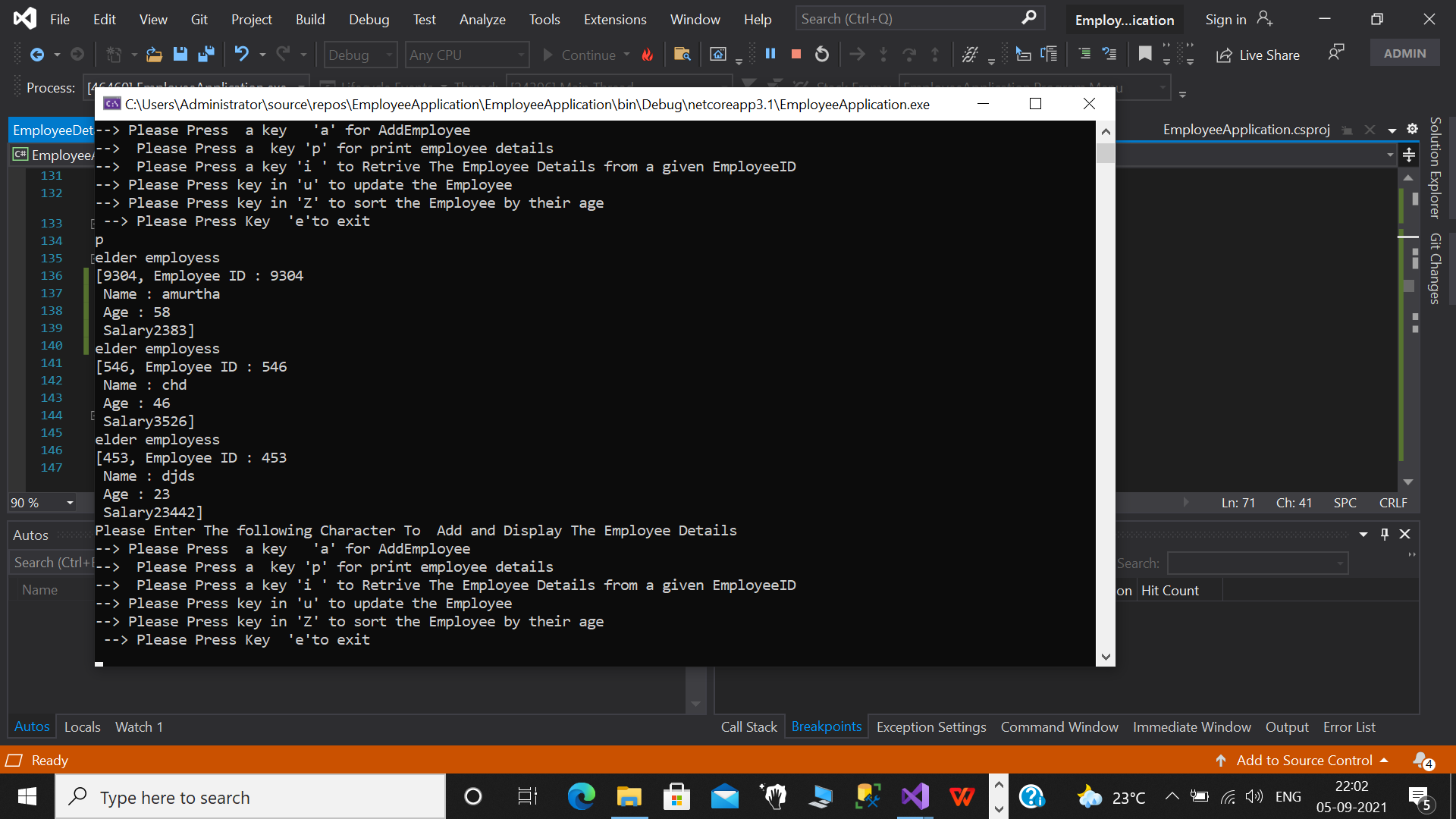
foreach (var item in employeeDetails)

PrintEmployeeDetails(item);

}

}

}



**------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------**

**HARD SECTION:**

**Use the application created in Question 1 of medium.**

**a. Display a menu to user which will enable to print all the employee details, add an**

**employee, modify the details of an employee (all except id), print an employee**

**details given his id and delete an employee from the collection**

**b. Ensure the application does not break at any point. Handles all the cases with proper**

**response**

**i. Example – If user enters an employee id that does not exists the response**

**should inform the user the same.**

/// <summary>

/// Taking choices from the user for different conditions using this menu

/// </summary>

public void EmployeeInformation()

{

string UserChoices;

do

{

Console.WriteLine("Please Enter The following Character To Add and Display The Employee Details");

Console.WriteLine("key in 'a' for AddEmployee and 'p' for print employee details and 'S' for sorting");

Console.WriteLine("Key in 'i' to print employee details by id");

Console.WriteLine("Key in 'n' to print employee details by name");

Console.WriteLine("Key in 'el' to print employee details by elder");

Console.WriteLine("enter 1 to modify name");

Console.WriteLine("enter 2 to modify age");

Console.WriteLine("enter 3 to modify salary");

Console.WriteLine("enter 4 to remove employee");

Console.WriteLine("Key in e to exit");

UserChoices = Console.ReadLine().ToLower();

switch (UserChoices)

{

case "a":

EmployeeDetailStoring();

break;

case "p":

PrintEmployee();

break;

case "s":

EmployeeSorting();

break;

case "i":

PrintEmployeeDetailsById();

break;

case "n":

PrintEmployeeDetailsByName();

break;

case "el":

ElderThanCurrentEmploye();

break;

case "1":

ModifyEmployeeDetails("name");

break;

case "2":

ModifyEmployeeDetails("age");

break;

case "3":

ModifyEmployeeDetails("salary");

break;

case "4":

ModifyEmployeeDetails("remove");

break;

case "e":

Console.WriteLine("we are done creating accounts");

break;

default:

Console.WriteLine("Ïnvalid option, please try again");

break;

}

} while (UserChoices != "e");

}

/// <summary>

/// Checks if the list is empty or not, and taking the employee id from user

/// </summary>

Employee EmployeDetailsOfId()

{

if (!ToCheckListIsEmpty())

{

return null;

}

else

{

Console.WriteLine("Enter employee ID");

int id = UserIntInput();

foreach (var item in sortedbysalary)

{

if (item.Id == id)

return item;

}

}

return null;

}

/// <summary>

/// Used to modify the name of the employee of id which is entered by the user

/// </summary>

/// <param name="modifyEmployee"></param>

void ModifyEmployeName(Employee modifyEmployee)

{

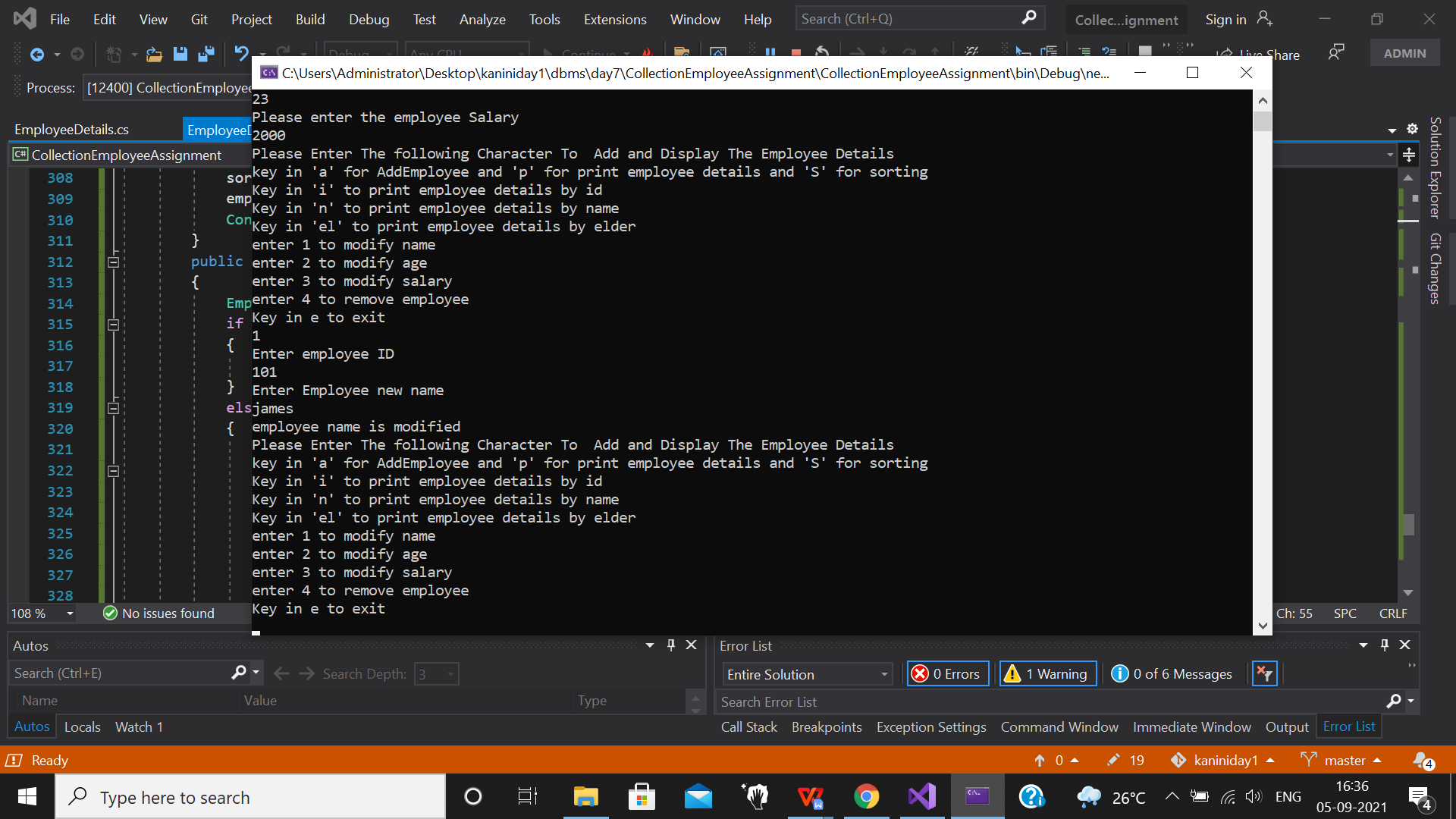
Console.WriteLine("Enter Employee new name");

string newEmployeName = Console.ReadLine();

modifyEmployee.Name = newEmployeName;

Console.WriteLine("employee name is modified");

}



/// <summary>

/// Used to modify the age of the employee of id which is entered by the user

/// </summary>

/// <param name="modifyEmployee"></param>

void ModifyEmployeAge(Employee modifyEmployee)

{

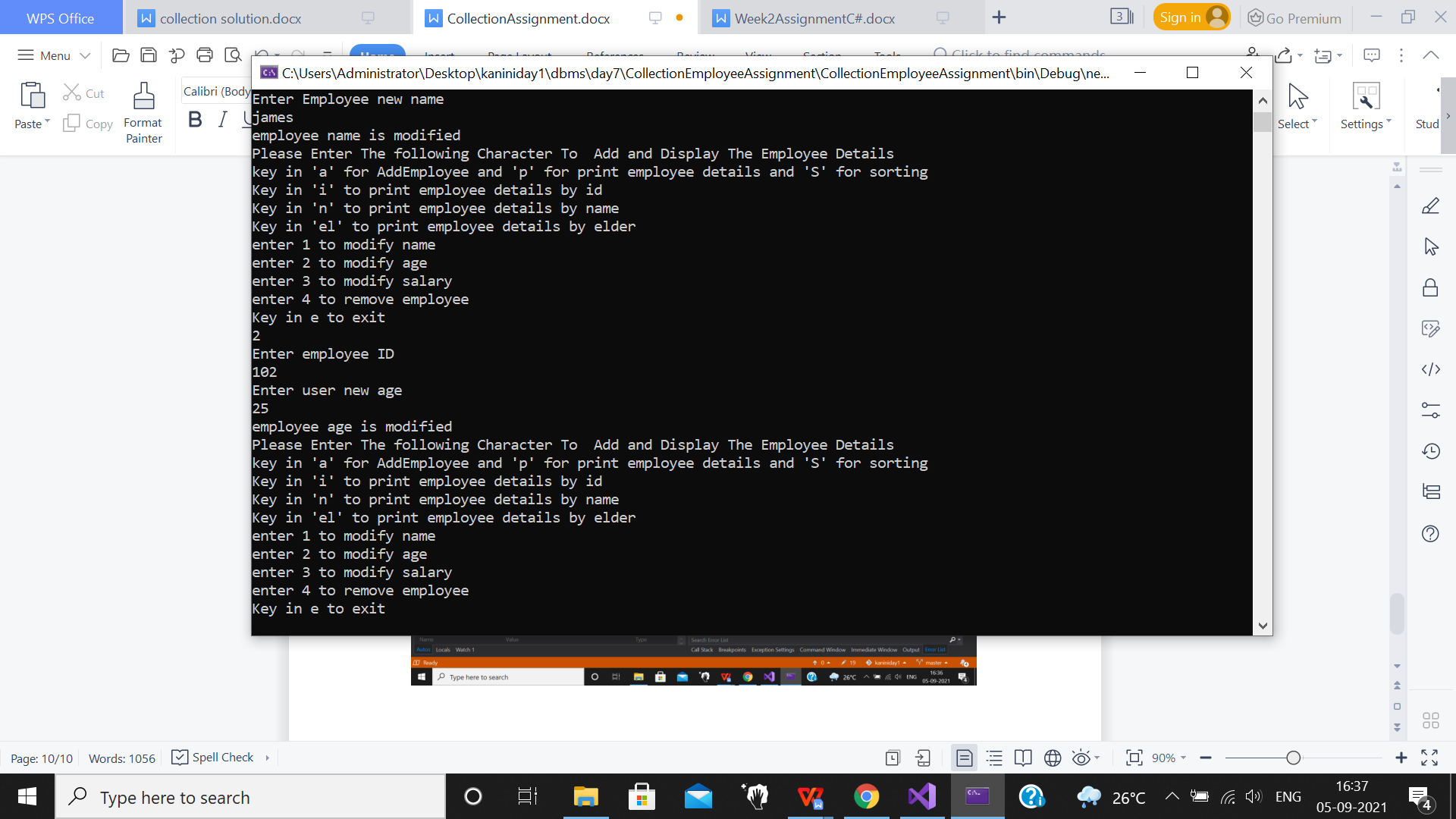
Console.WriteLine("Enter user new age");

int newEmployeeAge = UserIntInput();

modifyEmployee.Age = newEmployeeAge;

Console.WriteLine("employee age is modified");

}



/// <summary>

/// Used to modify the Salary of the employee of id which is entered by the user

/// </summary>

/// <param name="modifyEmployee"></param>

void ModifyEmployeeSalary(Employee modifyEmployee)

{

Console.WriteLine("enter new salary");

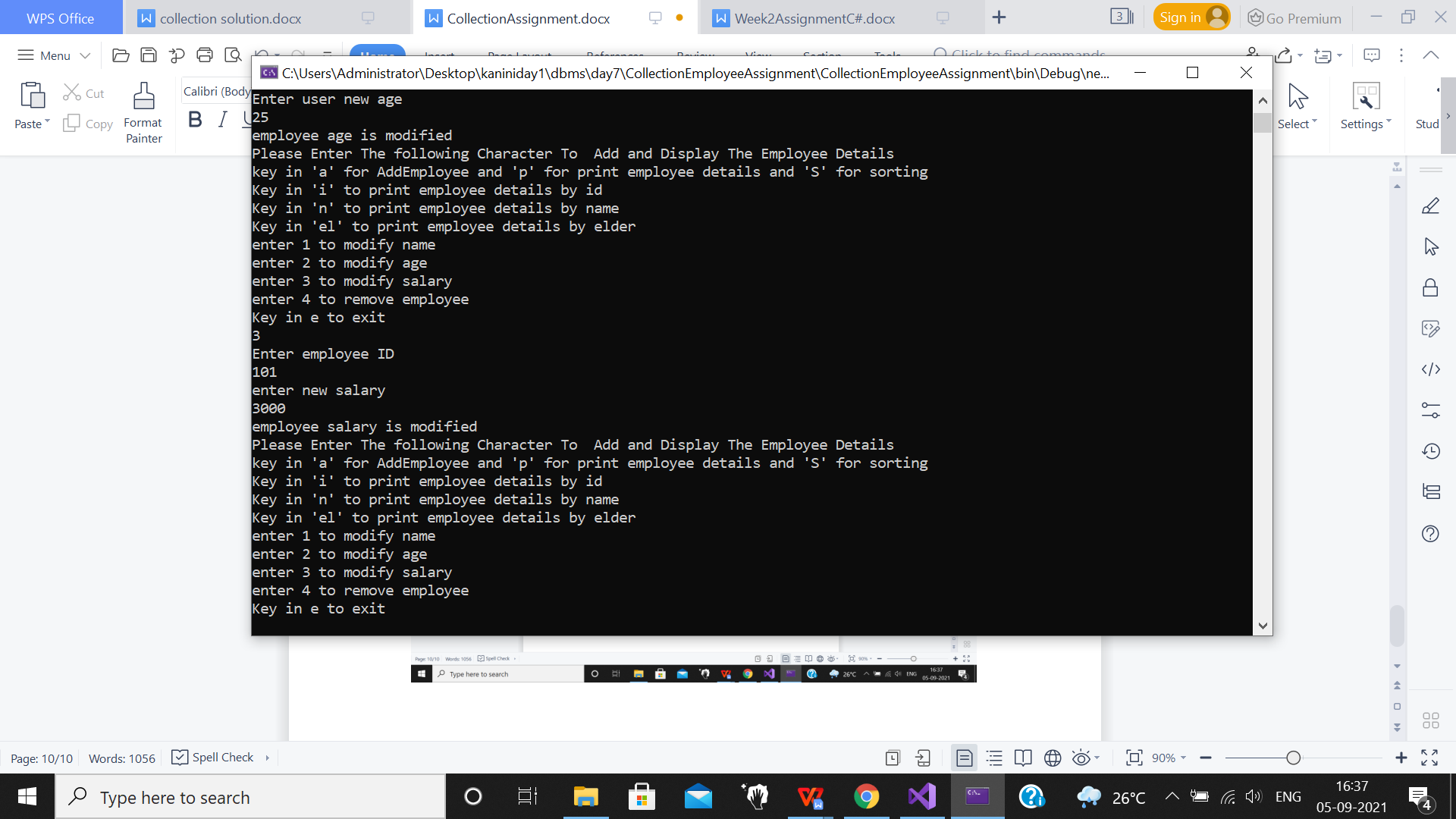
double newEmployeeSalary = Convert.ToDouble(Console.ReadLine());

modifyEmployee.Salary = newEmployeeSalary;

sortedbysalary.Sort();

Console.WriteLine("employee salary is modified");

}



/// <summary>

/// choice of the user is taken from the EmployeeInformation function and implemented in this ModifyEmployeeDetails function

/// </summary>

/// <param name="userChoice"></param>

void RemoveEmployee(Employee modifyEmployee)

{

sortedbysalary.Remove(modifyEmployee);

employee.Remove(modifyEmployee.Id);

Console.WriteLine("EMPLOYEE REMOVED...");

}

public void ModifyEmployeeDetails(string userChoice)

{

Employee modifyEmployee = EmployeDetailsOfId();

if (modifyEmployee == null)

{

Console.WriteLine(" or No employee with given id");

}

else

{

switch (userChoice)

{

case "name":

ModifyEmployeName(modifyEmployee);

break;

case "age":

ModifyEmployeAge(modifyEmployee);

break;

case "salary":

ModifyEmployeeSalary(modifyEmployee);

break;

case "remove":

RemoveEmployee(modifyEmployee);

break;

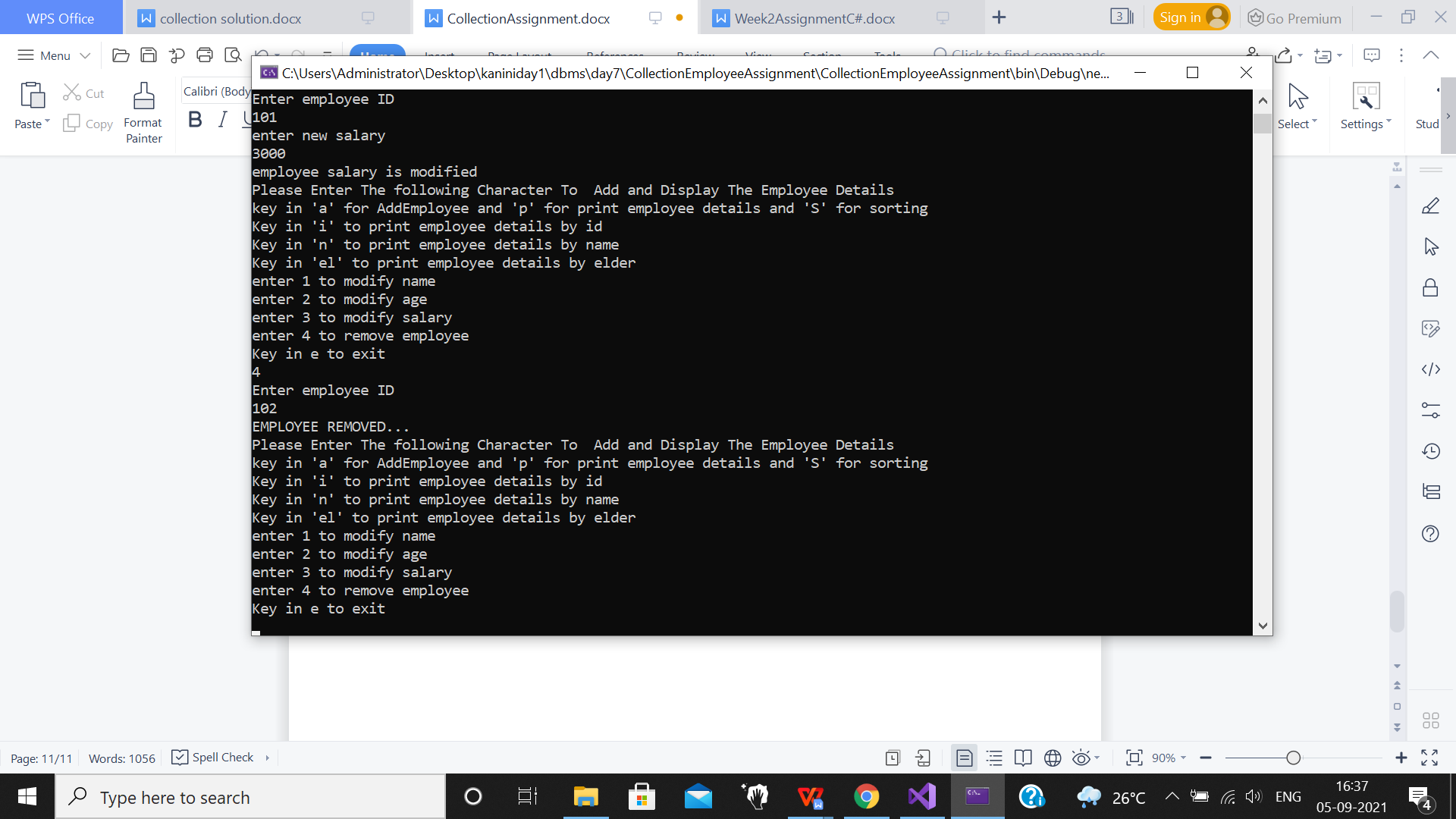
default:

break;

}

}

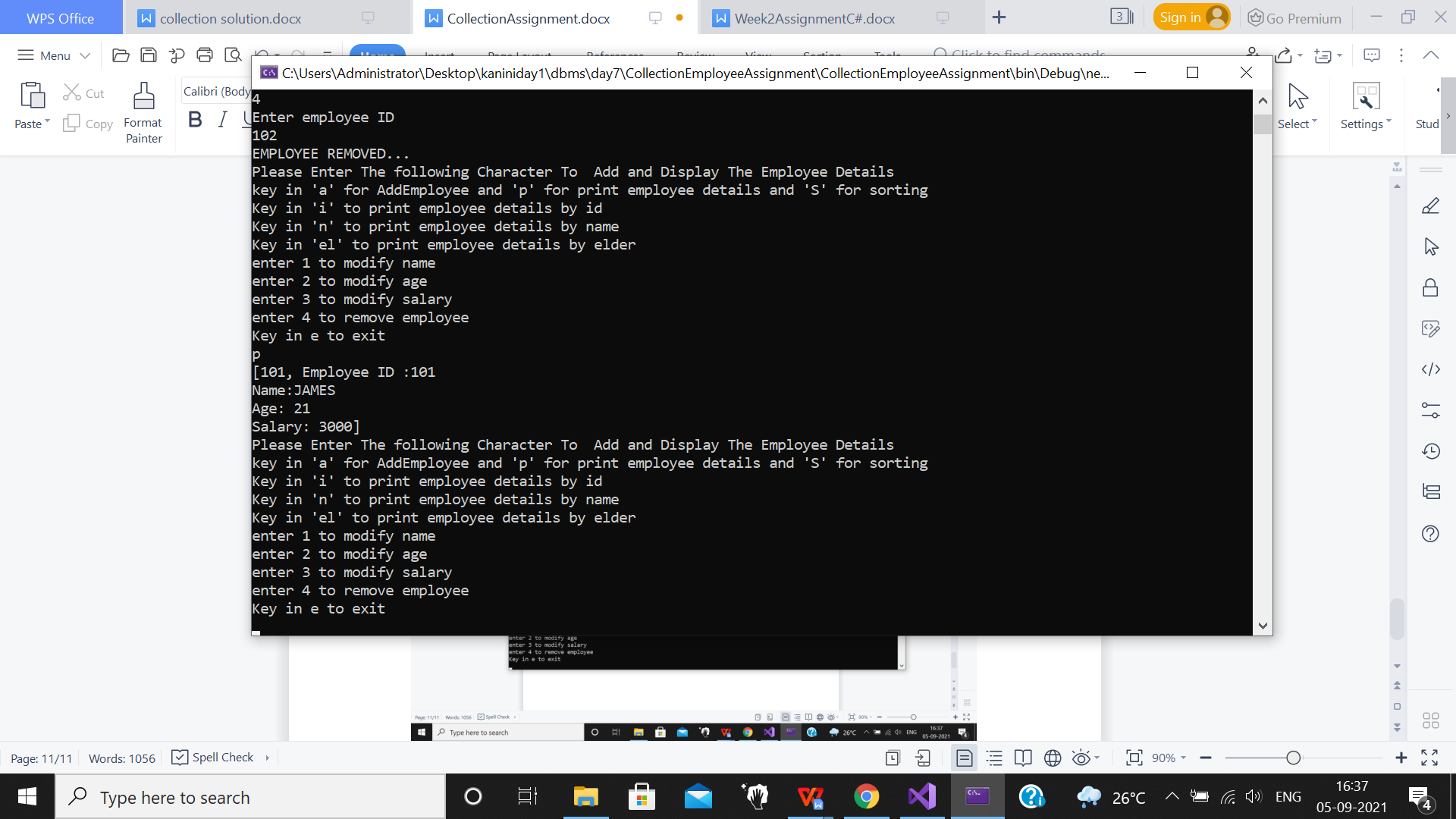
}



/// <summary>

/// Prints the current employee details after modifying the details of particular employee

/// </summary>



**------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------**

**EMPLOYEE.CS:**

using System;

using System.Collections.Generic;

using System.Diagnostics.CodeAnalysis;

using System.Text;

namespace CollectionEmployeeAssignment

{

/// <summary>

/// IComparable interface is used in employee class

/// </summary>

class Employee:IComparable<Employee>

{

int id, age;

string name;

double salary;

public Employee()

{

}

public Employee(int id, int age, string name, double salary)

{

this.id = id;

this.age = age;

this.name = name;

this.salary = salary;

}

public void TakeEmployeeDetailsFromUser()

{

Console.WriteLine("Please enter the employee ID");

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

Console.WriteLine("Please enter the employee Name");

name = Console.ReadLine();

Console.WriteLine("Please enter the employee Age");

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

Console.WriteLine("Please enter the employee Salary");

salary = Convert.ToDouble(Console.ReadLine());

}

public override bool Equals(object obj)

{

Employee e1,e2;

e1 = this;

e2 = (Employee)obj;

if (e1.Id == e2.Id)

return true;

else

return false;

}

public override string ToString()

{

return "Employee ID :" +id + "\nName:" +name + "\nAge: " +age +

"\nSalary: " +salary;

}

/// <summary>

/// ICpmarable operator is overloaded and compareTo is used to compair which other employees salary

/// </summary>

/// <param name="other"></param>

/// <returns>sorted elmployee list according to salary</returns>

public int CompareTo(Employee other)

{

return this.salary.CompareTo(other.salary);

}

public int Id

{

get => id; set => id = value;

}

public int Age

{

get => age; set => age = value;

}

public string Name

{

get => name; set => name = value;

}

public double Salary

{

get => salary; set => salary = value;

}

}

}

**------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------**

**PROGRAM.CS:**

using System;

using System.Collections.Generic;

namespace CollectionEmployeeAssignment

{

class Program

{

static void Main(string[] args)

{

EmployeeDetailsCollection employee = new EmployeeDetailsCollection();

employee.EmployeeInformation();

Console.ReadKey();

}

}

}