C# hands on:

1. **program**

**Task:**Write a C# method called *FilterAndSort* that takes a list of students (objects) and returns a sorted list of student names who have a GPA greater than 3.5. The list should be sorted in descending order.

**Input Format:**The input will be a *List<Student>*, where the *Student* class is defined as follows:

public *class* Student

{

public string Name { get; set; }

public double GPA { get; set; }

}

**Constraints:**

* The list will contain at least one *Student* object.
* Student names will be non-empty strings.
* GPA will be a double between 0.0 and 4.0.

**Output Format:** The output will be a *List<string>* containing the names of students who have a GPA greater than 3.5, sorted in descending order.

**Sample Input:**

var students = new List<Student>

{

new Student { Name = "Alice", GPA = 3.6 },

new Student { Name = "Bob", GPA = 3.2 },

new Student { Name = "Charlie", GPA = 3.8 }

};

**Sample Output:** [“Charlie”, “Alice”]

1. Program

**Introduction**

**Arrays**

In C#, data structures that can hold zero or more elements are known as *collections*. An array is a collection that has a fixed size/length and whose elements must all be of the same type. Elements can be assigned to an array or retrieved from it using an index. C# arrays are zero-based, meaning that the first element's index is always zero:

*// Declare array with explicit size (size is 2)*

int[] twoInts = new int[2];

*// Assign second element by index*

twoInts[1] = 8;

*// Retrieve the second element by index*

twoInts[1] == 8; *// => true*

Arrays can also be defined using a shortcut notation that allows you to both create the array and set its value. As the compiler can now tell how many elements the array will have, the length can be omitted:

*// Three equivalent ways to declare and initialize an array (size is 3)*

int[] threeIntsV1 = new int[] { 4, 9, 7 };

int[] threeIntsV2 = new[] { 4, 9, 7 };

int[] threeIntsV3 = { 4, 9, 7 };

Arrays can be manipulated by either calling an array instance's methods or properties, or by using the static methods defined in the Array class.

**For Loops**

A for loop allows one to repeatedly execute code in a loop until a condition is met.

for (int i = 0; i < 5; i++)

{

System.Console.Write(i);

}

*// => 01234*

A for loop consists of four parts:

1. The initializer: executed once before entering the loop. Usually used to define variables used within the loop.
2. The condition: executed before each loop iteration. The loop continues to execute while this evaluates to true.
3. The iterator: execute after each loop iteration. Usually used to modify (often: increment/decrement) the loop variable(s).
4. The body: the code that gets executed each loop iteration.

**Foreach Loops**

The fact that an array is also a *collection* means that, besides accessing values by index, you can iterate over *all* its values using a foreach loop:

char[] vowels = new [] { 'a', 'e', 'i', 'o', 'u' };

foreach (char vowel in vowels)

{

*// Output the vowel*

System.Console.Write(vowel);

}

*// => aeiou*

**Instructions**

You're an avid bird watcher that keeps track of how many birds have visited your garden in the last seven days.

You have six tasks, all dealing with the numbers of birds that visited your garden.

**1. Check what the counts were last week**

For comparison purposes, you always keep a copy of last week's counts nearby, which were: 0, 2, 5, 3, 7, 8 and 4. Implement the (*static*) BirdCount.LastWeek() method that returns last week's counts:

BirdCount.LastWeek();

*// => [0, 2, 5, 3, 7, 8, 4]*

**2. Check how many birds visited today**

Implement the BirdCount.Today() method to return how many birds visited your garden today. The bird counts are ordered by day, with the first element being the count of the oldest day, and the last element being today's count.

int[] birdsPerDay = { 2, 5, 0, 7, 4, 1 };

var birdCount = new BirdCount(birdsPerDay);

birdCount.Today();

*// => 1*

**3. Increment today's count**

Implement the BirdCount.IncrementTodaysCount() method to increment today's count:

int[] birdsPerDay = { 2, 5, 0, 7, 4, 1 };

var birdCount = new BirdCount(birdsPerDay);

birdCount.IncrementTodaysCount();

birdCount.Today();

*// => 2*

**4. Check if there was a day with no visiting birds**

Implement the BirdCount.HasDayWithoutBirds() method that returns true if there was a day at which zero birds visited the garden; otherwise, return false:

int[] birdsPerDay = { 2, 5, 0, 7, 4, 1 };

var birdCount = new BirdCount(birdsPerDay);

birdCount.HasDayWithoutBirds();

*// => true*

**5. Calculate the number of visiting birds for the first number of days**

Implement the BirdCount.CountForFirstDays() method that returns the number of birds that have visited your garden from the start of the week, but limit the count to the specified number of days from the start of the week.

int[] birdsPerDay = { 2, 5, 0, 7, 4, 1 };

var birdCount = new BirdCount(birdsPerDay);

birdCount.CountForFirstDays(4);

*// => 14*

**6. Calculate the number of busy days**

Some days are busier that others. A busy day is one where five or more birds have visited your garden. Implement the BirdCount.BusyDays() method to return the number of busy days:

int[] birdsPerDay = { 2, 5, 0, 7, 4, 1 };

var birdCount = new BirdCount(birdsPerDay);

birdCount.BusyDays();

*// => 2*