Searching Techniques

By Prakash

Agenda

- What is Searching?
- Types of Searches
- Linear Search
- Binary Search
- Example programs

Searching

What is Searching?

Searching is the process of fetching a specific element in a collection of elements.

The collection can be an array or a linked list. If you find the element in the list, the process is considered successful, and it returns the location of that element.

And in contrast, if you do not find the element, it deems the search unsuccessful.

Types of Searching

Types of Searching

There are two popular search algorithms we have. They are

1. Linear Search
2. Binary Search

Linear Search

What is Linear Search?

A linear search is the simplest approach employed to search for an element in a data set.

It examines each element until it finds a match, starting at the beginning of the data set, until the end.

The search is finished and terminated once the target element is located. If it finds no match, the algorithm must terminate its execution and return an appropriate result.

How Linear Search works?

Linear search, often known as sequential search, is the most basic search technique.

In this type of search, you go through the entire list and try to fetch a match for a single element.

If you find a match, then the address of the matching target element is returned.

On the other hand, if the element is not found, then it returns a NULL value.

When to use Linear Search?

The linear search algorithm is easy to implement and efficient in two scenarios:

- When the list contains lesser elements
- When searching for a single element in an unordered array

The Complexity of Linear Search Algorithm

You have three different complexities faced while performing Linear Search Algorithm, they are mentioned as follows.

- Best Case
- Worst Case
- Average Case

Best Case Complexity

The element being searched could be found in the first position.

In this case, **the search ends with a single successful comparison.**

Thus, in the best-case scenario, the linear search algorithm performs **O(1)** operations.

Worst Case Complexity

The element being searched may be at the last position in the array or not at all.

In the first case, the search succeeds in ‘n’ comparisons.

In the next case, the search fails after ‘n’ comparisons.

Thus, in the worst-case scenario, the linear search algorithm performs **O(n)** operations.

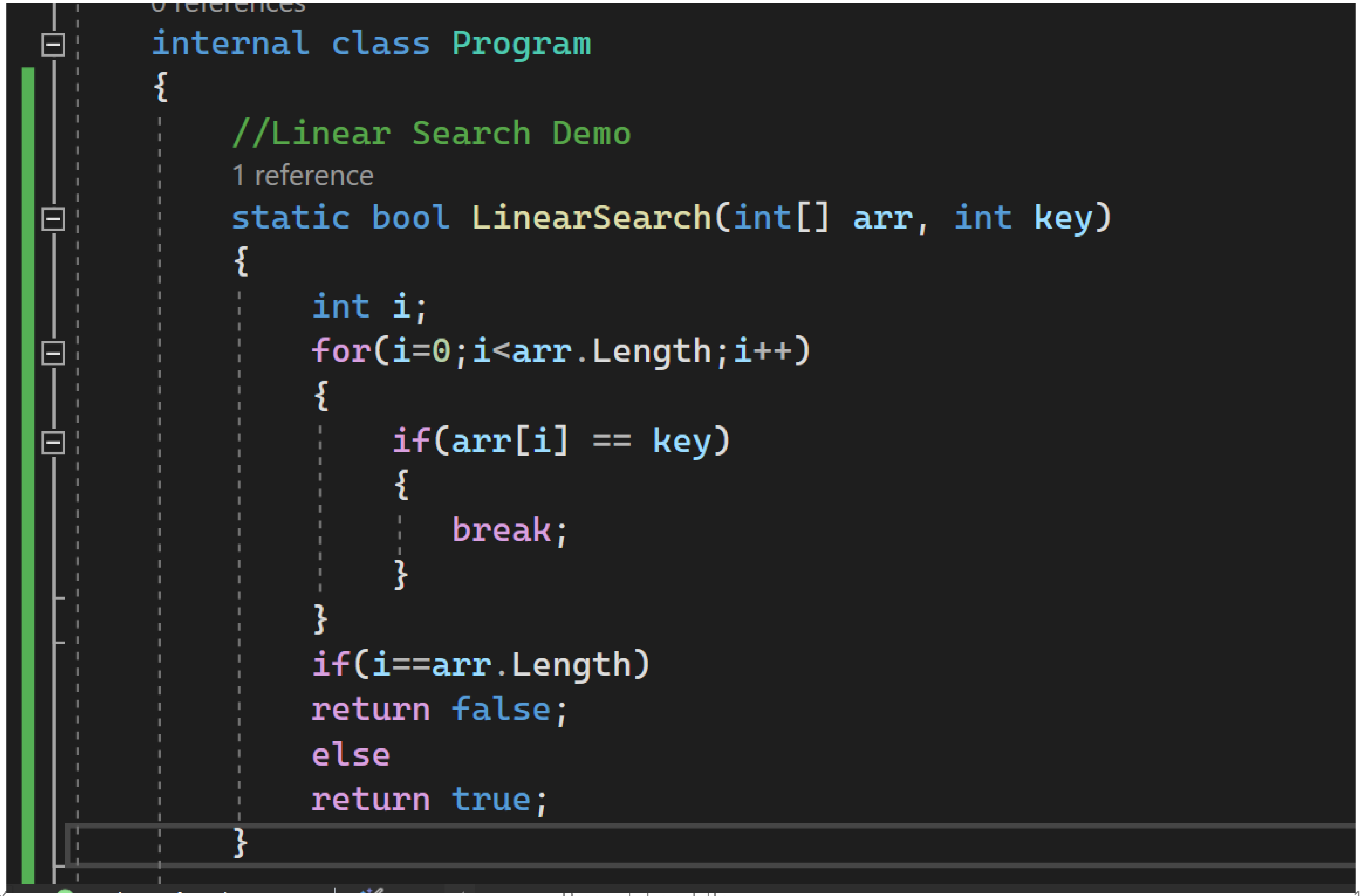
Average Case Complexity

When the element to be searched is in the middle of the array, the average case of the Linear Search Algorithm is **O(n).**

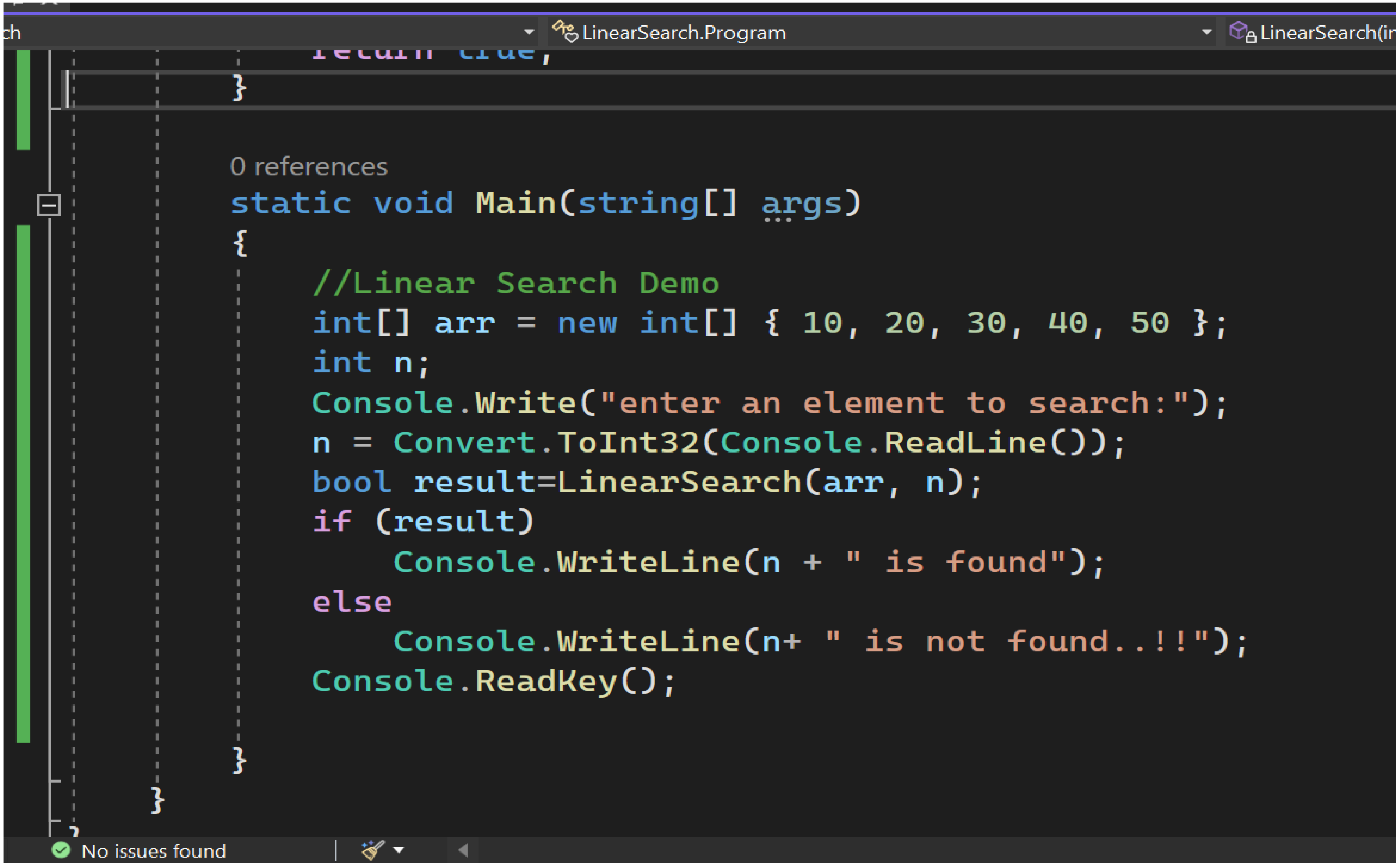
Applications of Linear Search Algorithm

- Linear search can be applied to both single-dimensional and multi-dimensional arrays.
- Linear search is easy to implement and effective when the array contains only a few elements.
- Linear Search is also efficient when the search is performed to fetch a single search in an unordered-List.

Linear Search Program



```
0 references
internal class Program
{
    //Linear Search Demo
    1 reference
    static bool LinearSearch(int[] arr, int key)
    {
        int i;
        for(i=0;i<arr.Length;i++)
        {
            if(arr[i] == key)
            {
                break;
            }
        }
        if(i==arr.Length)
        return false;
        else
        return true;
    }
```



```
            }

        static void Main(string[] args)
        {
            //Linear Search Demo
            int[] arr = new int[] { 10, 20, 30, 40, 50 };
            int n;
            Console.Write("enter an element to search:");
            n = Convert.ToInt32(Console.ReadLine());
            bool result=LinearSearch(arr, n);
            if (result)
                Console.WriteLine(n + " is found");
            else
                Console.WriteLine(n+ " is not found..!!");
            Console.ReadKey();

        }
    }
```

Binary Search

What is Binary Search?

Binary searches are efficient algorithms based on the concept of **"divide and conquer"** that improves the search by recursively dividing the array in half until you either find the element or the list gets narrowed down to one piece that doesn't match the needed element.

How Binary Search works?

Binary searches work under the principle of using the sorted information in the array to reduce the time complexity to zero **(Log n).**

How Binary Search works?

Binary searches work under the principle of using the sorted information in the array to reduce the time complexity to zero **(Log n).**

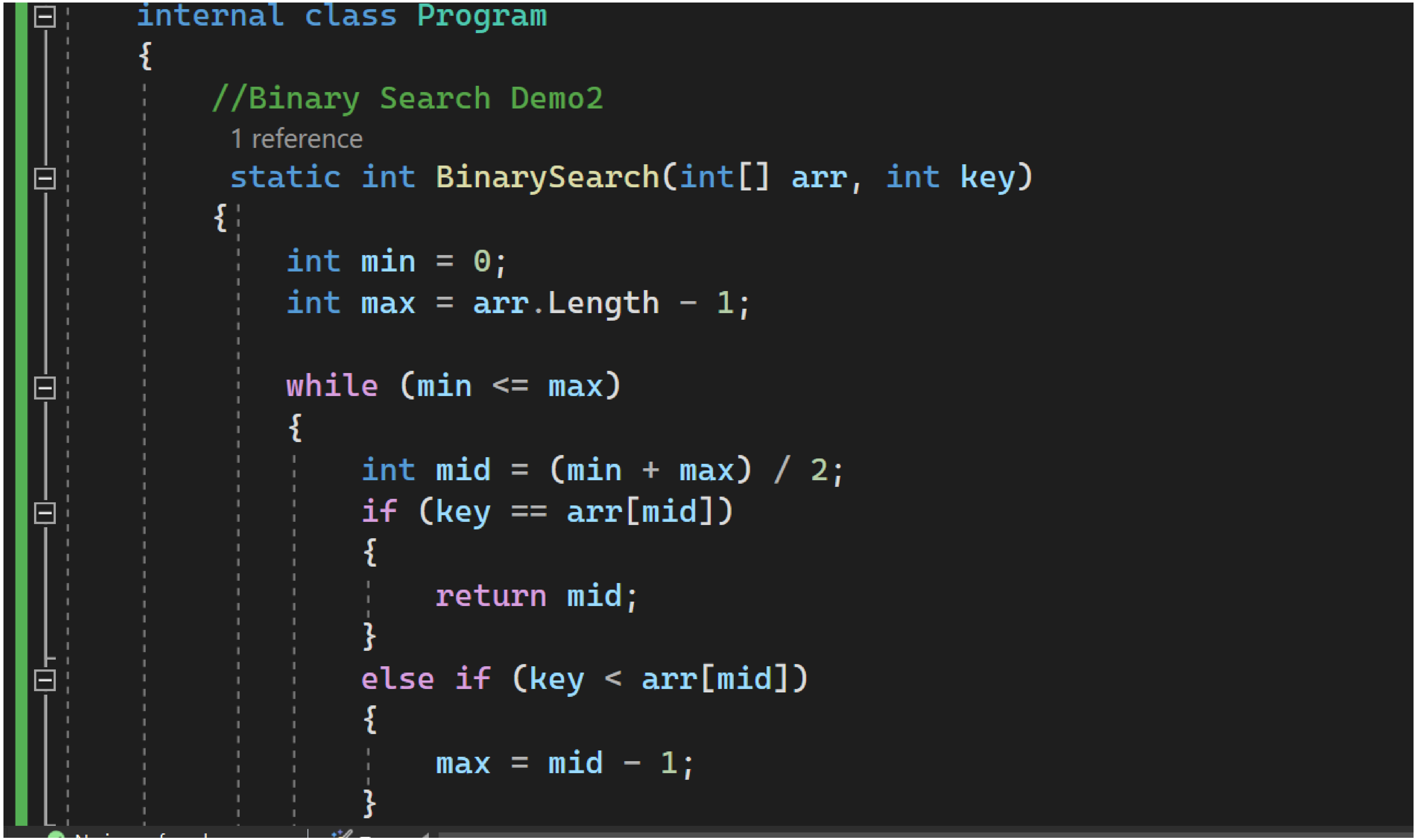
How Binary Search works?

- Begin with an interval that covers the entire array
- If the search key value is less than the middle-interval item, narrow the interval to that lower half. Otherwise, narrow the interval to the upper half.
- Keep checking the chosen interval until either the value is found or the interval's empty

When to Use Binary Search?

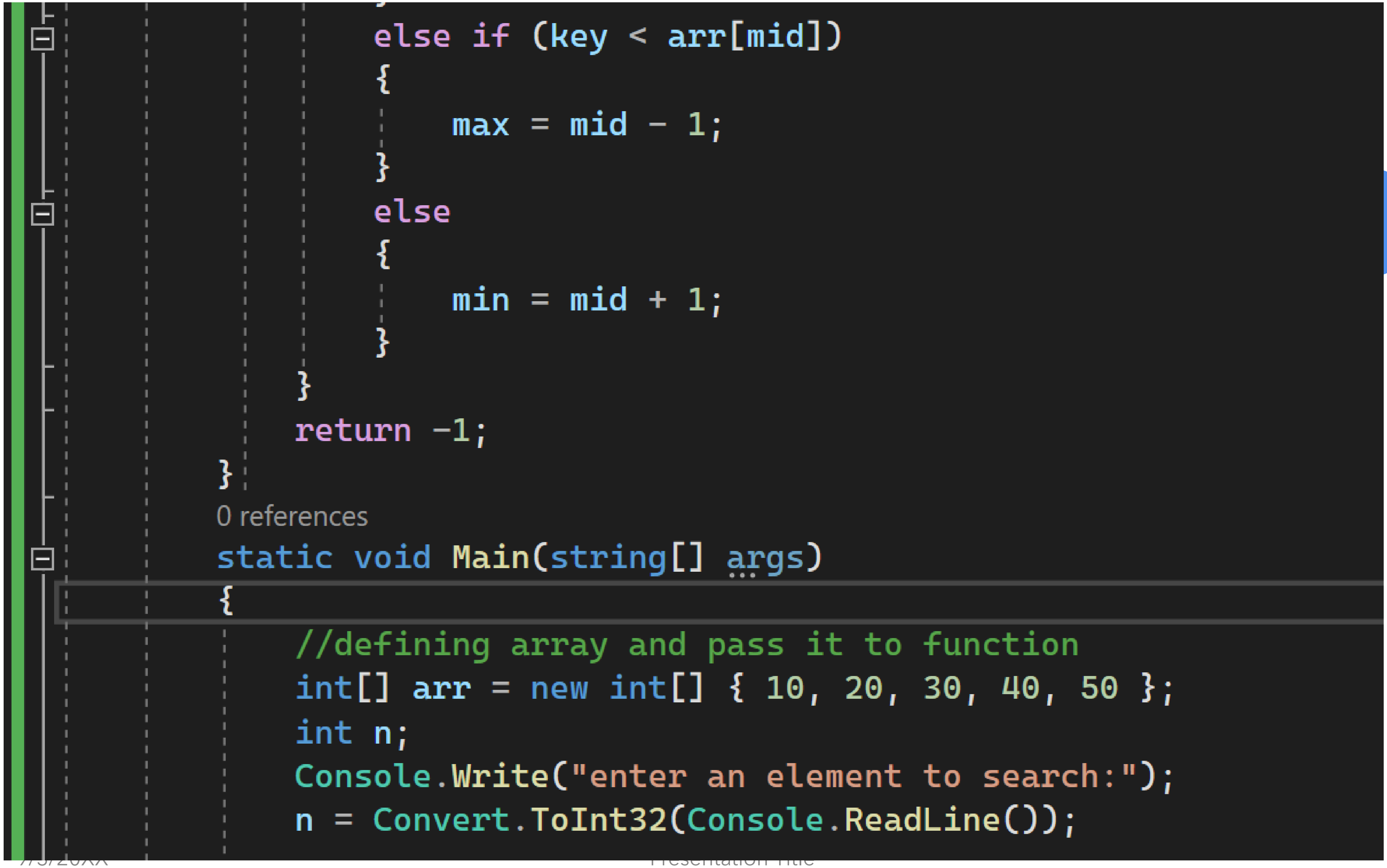
- Binary search algorithms are typically used to find one element in a sorted sequence.

Binary Search Program

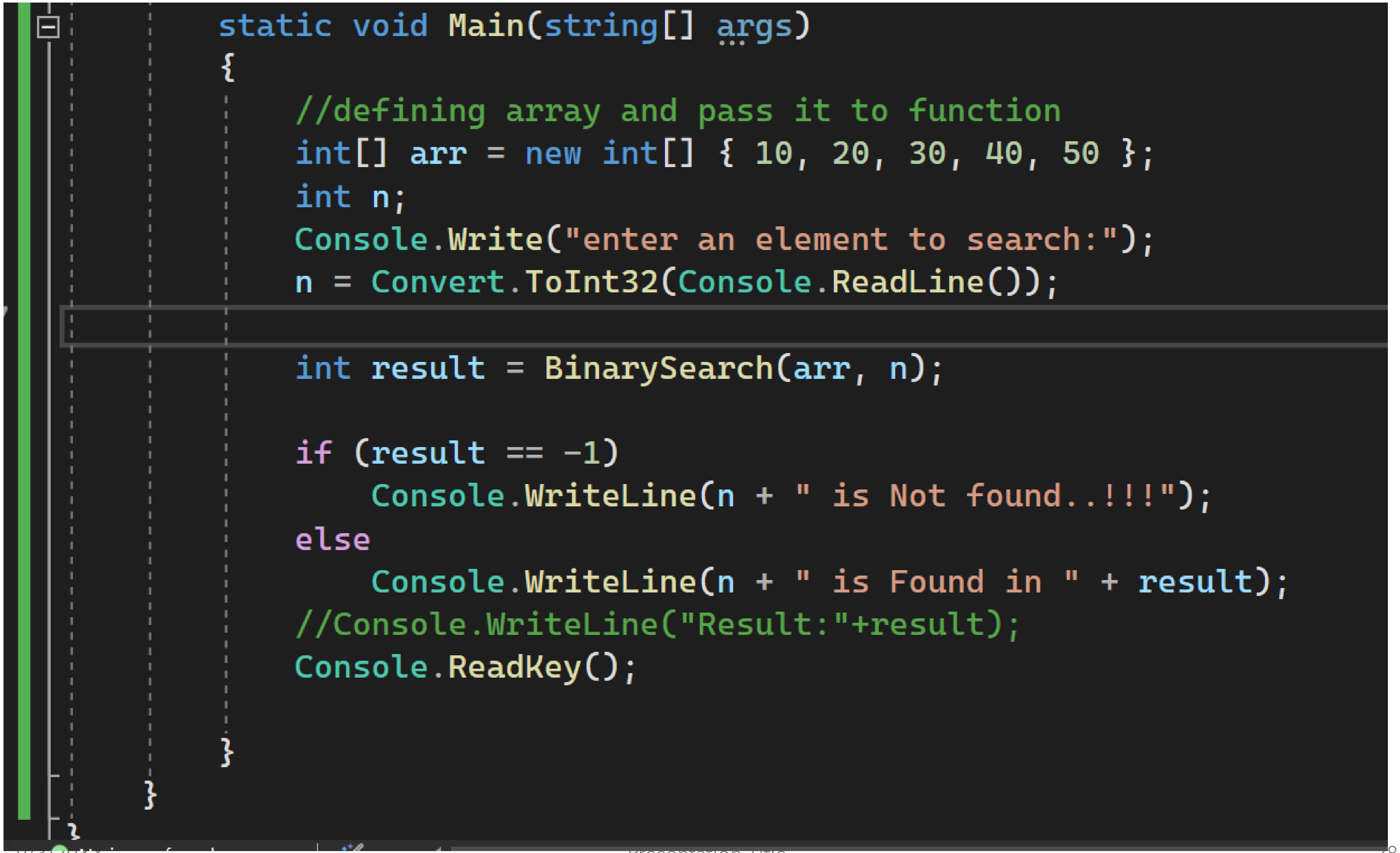


```
internal class Program
{
    //Binary Search Demo2
    1 reference
    static int BinarySearch(int[] arr, int key)
    {
        int min = 0;
        int max = arr.Length - 1;

        while (min <= max)
        {
            int mid = (min + max) / 2;
            if (key == arr[mid])
            {
                return mid;
            }
            else if (key < arr[mid])
            {
                max = mid - 1;
            }
```



```
            else if (key < arr[mid])
            {
                max = mid - 1;
            }
            else
            {
                min = mid + 1;
            }
        }
        return -1;
    }
    0 references
    static void Main(string[] args)
    {
        //defining array and pass it to function
        int[] arr = new int[] { 10, 20, 30, 40, 50 };
        int n;
        Console.Write("enter an element to search:");
        n = Convert.ToInt32(Console.ReadLine());
```



```
        static void Main(string[] args)
        {
            //defining array and pass it to function
            int[] arr = new int[] { 10, 20, 30, 40, 50 };
            int n;
            Console.Write("enter an element to search:");
            n = Convert.ToInt32(Console.ReadLine());

            int result = BinarySearch(arr, n);

            if (result == -1)
                Console.WriteLine(n + " is Not found..!!!");
            else
                Console.WriteLine(n + " is Found in " + result);
            //Console.WriteLine("Result:"+result);
            Console.ReadKey();

        }
    }
}
```

Thank you

Prakash (.Net Trainer)

Prakash.trainer1@gmail.com

Thank You