Problem: Write a program that: o Initializes a one-dimensional array in three different ways (new int[size], initializer list, and Array syntax sugar). o Assigns values to each element in the array and prints them. o Demonstrates an IndexOutOfRangeException. • Question: What is the default value assigned to array elements in C#?

C# Program

```
csharp
CopyEdit
using System;

class Program
{
    static void Main()
    {
        Console.WriteLine("=== Array Initialization and Access ===");

        // 1. Using new int[size]
        int[] array1 = new int[3]; // default values are 0
        array1[0] = 10;
        array1[1] = 20;
        array1[2] = 30;

        Console.WriteLine("Array1 (using new int[size]):");
        PrintArray(array1);

        // 2. Using initializer list
        int[] array2 = new int[] { 40, 50, 60 };
        Console.WriteLine("Array2 (using initializer list):");
        PrintArray(array2);

        // 3. Using array syntax sugar
        int[] array3 = { 70, 80, 90 };
        Console.WriteLine("Array3 (using syntax sugar):");
        PrintArray(array3);

        // Demonstrating IndexOutOfRangeException
        try
        {
            Console.WriteLine("Attempting to access array1[3]...");
            Console.WriteLine(array1[3]); // Invalid index (only 0 to 2
valid)
        }
        catch (IndexOutOfRangeException e)
        {
            Console.WriteLine("Caught Exception: " + e.Message);
        }
    }

    static void PrintArray(int[] arr)
    {
        for (int i = 0; i < arr.Length; i++)
        {
            Console.WriteLine($"Element {i}: {arr[i]}");
        }
        Console.WriteLine();
    }
```

```
}
```

💬 Answer to the Question:

Q: What is the default value assigned to array elements in C#?
A:
In C#, when you initialize an array using `new int[size]`, **each element is assigned the default value of the data type**. For `int`, the default value is `0`.

This applies to all types: `bool` → `false`, `string` → `null`, etc.

Write a program to: o Create two arrays (arr1 and arr2). o Perform a shallow copy and demonstrate how modifying one affects the other. o Perform a deep copy using the Clone method and show that modifications do not affect the copied array. • Question: What is the difference between Array.Clone() and Array.Copy()?

C# Program

```
csharp
CopyEdit
using System;

class Program
{
    static void Main()
    {
        Console.WriteLine("=== Shallow Copy Example ===");

        // Create original array
        int[] arr1 = { 1, 2, 3 };

        // Shallow copy (reference assignment)
        int[] arr2 = arr1;

        // Modify arr2
        arr2[0] = 100;

        Console.WriteLine("arr1 after modifying arr2 (shallow copy):");
        PrintArray(arr1); // arr1 is also modified

        Console.WriteLine("arr2:");
        PrintArray(arr2);

        Console.WriteLine("=== Deep Copy using Clone() ===");

        // Deep copy using Clone
        int[] arr3 = (int[])arr1.Clone();

        // Modify arr3
        arr3[1] = 200;

        Console.WriteLine("arr1 after modifying arr3 (deep copy):");
        PrintArray(arr1); // arr1 is not affected
```

```
        Console.WriteLine("arr3:");
        PrintArray(arr3);
    }

    static void PrintArray(int[] arr)
    {
        foreach (int item in arr)
        {
            Console.Write(item + " ");
        }
        Console.WriteLine();
    }
}
```

💬 Answer to the Question:

Q: What is the difference between `Array.Clone()` and `Array.Copy()`?
A:

Feature	`Array.Clone()`	`Array.Copy()`
Type	Method of `Array`	Static method of `Array` class
Purpose	Creates a **shallow copy** of the array	Copies elements from one array to another
Return Type	Returns a new **object** (cast needed)	Returns `void`, modifies target array
Use Case	Quick clone of the same-size array	Copy part/all of source to destination
Deep Copy?	No (shallow for reference types)	No (still shallow for reference types)

🔹 Both methods **copy only top-level elements**, not nested object instances. For true deep copies of arrays containing **reference types**, you must implement manual copying of each object.

: Write a program to: o Create a 2D array with student grades (3 students, 3 subjects each). o Take input from the user to fill the array. o Print the grades for each student using nested loops. • Question: What is the difference between GetLength() and Length for multi dimensional arrays?

C# Program: 2D Array for Student Grades

```
csharp
CopyEdit
using System;

class Program
{
    static void Main()
    {
        const int students = 3;
        const int subjects = 3;

        // 2D array: 3 students × 3 subjects
        int[,] grades = new int[students, subjects];

        // Taking input from user
        Console.WriteLine("Enter grades for 3 students and 3 subjects:");
```

```
        for (int i = 0; i < students; i++)
        {
            Console.WriteLine($"Enter grades for Student {i + 1}:");
            for (int j = 0; j < subjects; j++)
            {
                Console.Write($"Subject {j + 1}: ");
                grades[i, j] = int.Parse(Console.ReadLine());
            }
        }

        // Printing grades
        Console.WriteLine("\n=== Student Grades ===");
        for (int i = 0; i < students; i++)
        {
            Console.Write($"Student {i + 1}: ");
            for (int j = 0; j < subjects; j++)
            {
                Console.Write(grades[i, j] + " ");
            }
            Console.WriteLine();
        }
    }
}
```

💬 Answer to the Question:

Q: What is the difference between `GetLength()` and `Length` for multi-dimensional arrays?
A:

Feature	`GetLength(dimension)`	`Length`
Purpose	Returns the size of a specific dimension (0-based)	Returns total number of elements
Parameters	Takes an integer (dimension index: 0, 1, ...)	No parameters
For 2D array `[3, 3]`	`array.GetLength(0)` → 3 (rows) `array.GetLength(1)` → 3 (columns)	`array.Length` → 9 (3×3)
Use Case	Looping through rows/columns	Finding total elements in array

- Example:

```
csharp
CopyEdit
int[,] arr = new int[3, 3];
Console.WriteLine(arr.GetLength(0)); // 3 (rows)
Console.WriteLine(arr.GetLength(1)); // 3 (columns)
Console.WriteLine(arr.Length);       // 9 (total elements)
```

Write a program that: o Demonstrates at least 5 array methods (Sort, Reverse, IndexOf, Copy, Clear). o Explains the changes before and after applying each method. • Question: What is the difference between Array.Copy() and Array.ConstrainedCopy()?

C# Program Demonstrating Array Methods

```
csharp
CopyEdit
using System;

class Program
{
    static void Main()
    {
        int[] original = { 5, 3, 8, 1, 2 };

        Console.WriteLine("Original Array:");
        PrintArray(original);

        // 1. Sort
        Console.WriteLine("\n1. After Array.Sort:");
        int[] sorted = (int[])original.Clone(); // clone to preserve
original
        Array.Sort(sorted);
        PrintArray(sorted);
        Console.WriteLine("Explanation: Sorts the array in ascending
order.");

        // 2. Reverse
        Console.WriteLine("\n2. After Array.Reverse:");
        int[] reversed = (int[])sorted.Clone();
        Array.Reverse(reversed);
        PrintArray(reversed);
        Console.WriteLine("Explanation: Reverses the order of elements.");

        // 3. IndexOf
        Console.WriteLine("\n3. Using Array.IndexOf:");
        int index = Array.IndexOf(original, 8);
        Console.WriteLine($"Index of 8 in original array: {index}");
        Console.WriteLine("Explanation: Finds the index of element 8.");

        // 4. Copy
        Console.WriteLine("\n4. After Array.Copy:");
        int[] copied = new int[5];
        Array.Copy(original, copied, original.Length);
        PrintArray(copied);
        Console.WriteLine("Explanation: Copies elements from original to
new array.");

        // 5. Clear
        Console.WriteLine("\n5. After Array.Clear:");
        Array.Clear(copied, 1, 2); // clear 2 elements starting from index
1
        PrintArray(copied);
        Console.WriteLine("Explanation: Sets elements at index 1 and 2 to 0
(default).");
    }
```

```
    static void PrintArray(int[] arr)
    {
        foreach (var item in arr)
        {
            Console.Write(item + " ");
        }
        Console.WriteLine();
    }
}
```

💬 Answer to the Question:

Q: What is the difference between `Array.Copy()` and `Array.ConstrainedCopy()`?
A:

Feature	`Array.Copy()`	`Array.ConstrainedCopy()`
Purpose	General-purpose copy of elements between arrays	More secure copy with strict runtime constraints
Safety	Less strict—may throw exceptions at runtime	Guarantees rollback on failure (atomic operation)
Exception Behavior	Partially copied array may remain modified	All-or-nothing: no changes if exception occurs
Usage Scenario	Regular use	Used in **security-sensitive** or **system-level** code
Performance	Slightly faster	Slight overhead due to safety checks

📝 **Note:** `Array.ConstrainedCopy()` is used in **low-level or critical code**, such as the .NET runtime or secure environments. It ensures **type safety** and **copy rollback** if any part fails.

Create a program that: o Uses a for loop to print all elements of a 1D array. o Uses a foreach loop to print all elements of the same array. o Uses a while loop to print all elements in reverse order. • Question: Why is foreach preferred for read-only operations on arrays?

```
using System;

class Program
{
  static void Main()
  {
    int[] numbers = { 10, 20, 30, 40, 50 };
```

```
        // 1. Using a for loop
        Console.WriteLine("Using for loop:");
        for (int i = 0; i < numbers.Length; i++)
        {
            Console.Write(numbers[i] + " ");
        }

        Console.WriteLine("\n");

        // 2. Using a foreach loop
        Console.WriteLine("Using foreach loop:");
        foreach (int number in numbers)
        {
            Console.Write(number + " ");
        }

        Console.WriteLine("\n");

        // 3. Using a while loop (in reverse order)
        Console.WriteLine("Using while loop (reverse order):");
        int index = numbers.Length - 1;
        while (index >= 0)
        {
            Console.Write(numbers[index] + " ");
            index--;
        }

        Console.WriteLine();
    }
}
```

💬 Answer to the Question:

Q: Why is foreach preferred for read-only operations on arrays?

A:

The foreach loop is preferred for read-only operations because:

✅ Simplicity: It abstracts away indexing and boundary conditions.

✅ Safety: It prevents accidental modification of array elements.

✅ Readability: It clearly conveys the intent to read rather than modify.

❌ Limitations: You cannot modify elements inside a foreach loop.

Example:

csharp

Copy

Edit

```
foreach (int num in numbers)
{
   // num = num + 1; // ❌ Not allowed – compile-time error
   Console.WriteLine(num); // ✅ Allowed
}
```

🔒 Use foreach when you want to access values, not change them.

For modifications, use for or while

Write a program that: o Repeatedly asks the user for a positive odd number. o Uses defensive coding to validate input using int.TryParse and a do-while loop. • Question: Why is input validation important when working with user inputs? • Problem: Write a program to: o Create a 2D array with fixed values. o Print the array elements in a matrix format (rows and columns). • Question: How can you format the output of a 2D array for better readability? • Problem: Write a

program that: o Asks the user to enter a month number. o Uses an if-else statement to determine the month name. o Uses a switch statement to perform the same task. • Question: When should you prefer a switch statement over if-else? • Problem: Write a program to: o Sort an array of integers using Array.Sort(). o Search for a specific value using Array.IndexOf() and Array.LastIndexOf(). • Question: What is the time complexity of Array.Sort()? • Problem: Write a program that: o Creates an array of integers. o Uses a for loop to calculate and print the sum of all elements. o Uses a foreach loop to calculate the same sum. • Question: Which loop (for or foreach) is more efficient for calculating the sum of an array, and why?

Program 1: Input Validation – Ask for a Positive Odd Number

```
csharp
CopyEdit
using System;

class Program
{
    static void Main()
    {
        int number;
        do
        {
            Console.Write("Enter a positive odd number: ");
            string input = Console.ReadLine();

            if (int.TryParse(input, out number) && number > 0 && number % 2
== 1)
            {
                Console.WriteLine($"Valid input: {number}");
                break;
            }
            else
            {
                Console.WriteLine("Invalid input. Please enter a positive
odd number.\n");
            }
        } while (true);
    }
}
```

🗨️ Question: Why is input validation important when working with user inputs?

Answer:
Input validation is important to prevent:

- **Invalid data** causing runtime errors.
- **Security vulnerabilities** (e.g., injection attacks).
- **Unexpected behavior** in program logic.
- Ensures program **robustness** and **user guidance**.

✅ Program 2: Print a 2D Array in Matrix Format

```
csharp
CopyEdit
using System;

class Program
{
    static void Main()
    {
        int[,] matrix = {
            { 1, 2, 3 },
            { 4, 5, 6 },
            { 7, 8, 9 }
        };

        Console.WriteLine("Matrix:");
        for (int i = 0; i < matrix.GetLength(0); i++) // rows
        {
            for (int j = 0; j < matrix.GetLength(1); j++) // columns
            {
                Console.Write(matrix[i, j].ToString().PadLeft(4));
            }
            Console.WriteLine();
        }
    }
}
```

💬 Question: How can you format the output of a 2D array for better readability?

Answer:
Use methods like:

- `PadLeft()` / `PadRight()` for alignment.
- Consistent spacing between elements.
- Consider string formatting (`String.Format`, interpolation).
  This ensures columns align neatly for matrices and tables.

✅ Program 3: Determine Month Name Using if-else and switch

```
csharp
CopyEdit
using System;

class Program
{
    static void Main()
```

```
    {
        Console.Write("Enter a month number (1-12): ");
        int month = int.Parse(Console.ReadLine());

        // Using if-else
        if (month == 1) Console.WriteLine("January");
        else if (month == 2) Console.WriteLine("February");
        else if (month == 3) Console.WriteLine("March");
        else if (month == 4) Console.WriteLine("April");
        else if (month == 5) Console.WriteLine("May");
        else if (month == 6) Console.WriteLine("June");
        else if (month == 7) Console.WriteLine("July");
        else if (month == 8) Console.WriteLine("August");
        else if (month == 9) Console.WriteLine("September");
        else if (month == 10) Console.WriteLine("October");
        else if (month == 11) Console.WriteLine("November");
        else if (month == 12) Console.WriteLine("December");
        else Console.WriteLine("Invalid month");

        // Using switch
        switch (month)
        {
            case 1: Console.WriteLine("January"); break;
            case 2: Console.WriteLine("February"); break;
            case 3: Console.WriteLine("March"); break;
            case 4: Console.WriteLine("April"); break;
            case 5: Console.WriteLine("May"); break;
            case 6: Console.WriteLine("June"); break;
            case 7: Console.WriteLine("July"); break;
            case 8: Console.WriteLine("August"); break;
            case 9: Console.WriteLine("September"); break;
            case 10: Console.WriteLine("October"); break;
            case 11: Console.WriteLine("November"); break;
            case 12: Console.WriteLine("December"); break;
            default: Console.WriteLine("Invalid month"); break;
        }
    }
}
```

💬 Question: When should you prefer a `switch` statement over `if-else`?

Answer:
Use `switch` when:

- Comparing a **single variable** to multiple constant values.
- It improves **readability** and **organization** over long if-else chains.
- It may perform **faster** due to compiler optimization in some cases.

✅ Program 4: Sort and Search in an Array

```
csharp
CopyEdit
using System;

class Program
```

```
{
    static void Main()
    {
        int[] numbers = { 5, 3, 8, 3, 9, 1 };

        Array.Sort(numbers);
        Console.WriteLine("Sorted array:");
        PrintArray(numbers);

        int index = Array.IndexOf(numbers, 3);
        int lastIndex = Array.LastIndexOf(numbers, 3);

        Console.WriteLine($"First index of 3: {index}");
        Console.WriteLine($"Last index of 3: {lastIndex}");
    }

    static void PrintArray(int[] arr)
    {
        foreach (int n in arr)
            Console.Write(n + " ");
        Console.WriteLine();
    }
}
```

💬 Question: What is the time complexity of `Array.Sort()`?

Answer:

- `Array.Sort()` uses **introsort**, a hybrid of quicksort, heapsort, and insertion sort.
- Time complexity:
  - **Average**: `O(n log n)`
  - **Worst case**: `O(n log n)`
- Space complexity: `O(log n)` for recursive calls.

✅ Program 5: Sum Elements Using for and foreach Loops

```
csharp
CopyEdit
using System;

class Program
{
    static void Main()
    {
        int[] numbers = { 10, 20, 30, 40, 50 };

        // Using for loop
        int sumFor = 0;
        for (int i = 0; i < numbers.Length; i++)
        {
            sumFor += numbers[i];
        }
```

```
        Console.WriteLine($"Sum using for loop: {sumFor}");

        // Using foreach loop
        int sumForeach = 0;
        foreach (int num in numbers)
        {
            sumForeach += num;
        }
        Console.WriteLine($"Sum using foreach loop: {sumForeach}");
    }
}
```

💬 Question: Which loop (for or foreach) is more efficient for calculating the sum of an array, and why?

Answer:

- In most cases, **`foreach` is as efficient** as `for`.
- However, `for` might be **slightly faster** when:
  - Indexing is needed.
  - You want to avoid iterator overhead (especially in structs).
- **`foreach`** is preferred for **readability** and simplicity in read-only access.
- **----------**

Define an enum called DayOfWeek with values: Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday. Write a program that takes an integer input from the user (1-7) and prints the corresponding day using the enum. Use Enum.Parse to convert an integer to an enum value. 3- What happens if the user enters a value outside the range of 1 to 7?

C# Program: Enum DayOfWeek with Input

```
csharp
CopyEdit
using System;

class Program
{
    // Define enum (starts from 0 by default)
    enum DayOfWeek
    {
        Monday = 1, // Start from 1 to match user input
        Tuesday,
        Wednesday,
        Thursday,
        Friday,
        Saturday,
        Sunday
    }

    static void Main()
    {
```

```
        Console.Write("Enter a number (1 to 7) for the day of the week: ");
        string input = Console.ReadLine();

        if (int.TryParse(input, out int dayNumber))
        {
            if (Enum.IsDefined(typeof(DayOfWeek), dayNumber))
            {
                DayOfWeek day = (DayOfWeek)Enum.Parse(typeof(DayOfWeek),
dayNumber.ToString());
                Console.WriteLine($"That day is: {day}");
            }
            else
            {
                Console.WriteLine("Invalid input: Please enter a number
between 1 and 7.");
            }
        }
        else
        {
            Console.WriteLine("Invalid input: Not a valid integer.");
        }
    }
}
```

💬 Question: What happens if the user enters a value outside the range of 1 to 7?

Answer:

If the user enters a number **outside 1–7**, two things can happen depending on the code:

- If you **don't check** with `Enum.IsDefined`, then:
  - `Enum.Parse()` will **still parse the number** into the enum type (as a cast), even if it's invalid.
  - You may get a **meaningless enum value** like `0` or `8`, which may not correspond to any named constant.
- If you **check using `Enum.IsDefined`** (like in the program above):
  - You can prevent invalid values and give the user a friendly message.

Best Practice: Always validate user input using `Enum.IsDefined()` or `Enum.TryParse()` with `ignoreCase` and `out`.

What's the default size of the Stack and Heap in C# (.NET)?

📌 Stack:

- **Default Size (on Windows):**

- **~1 MB per thread** for 32-bit processes
- **~1 MB or more per thread** for 64-bit processes (can go higher)

📌 Heap:

- The **Heap size is much larger**, limited by:
  - Available **system memory**
  - **Process architecture**:
    - **32-bit**: Up to ~2–4 GB
    - **64-bit**: Theoretically up to terabytes (limited by OS and physical RAM)

💡 Considerations:

Aspect	Stack	Heap
Usage	Stores value types and function call data	Stores reference types and large objects
Access Speed	Very fast	Slower (due to dynamic allocation)
Lifetime	Managed automatically (LIFO)	Managed by **Garbage Collector**
Size Limit	Small and fixed (default ~1MB/thread)	Large and expandable
Thread Safety	Thread-safe (each thread has its own)	Needs synchronization for shared access

⚠️ Stack Overflow Exception occurs if the stack exceeds its limit (e.g., deep recursion). Heap memory is slower but suited for large or long-lived objects.

✅ 2. What is Time Complexity?

📌 Definition:

Time complexity describes how the **execution time** of an algorithm grows with the **input size** (`n`).

It helps predict performance and efficiency, especially for large inputs.

📊 Common Time Complexities:

Big-O Notation	Description	Example
O(1)	Constant time	Accessing an array index

Big-O Notation	Description	Example
O(log n)	Logarithmic	Binary search
O(n)	Linear	Single loop over array
O(n log n)	Linearithmic	Efficient sorting (MergeSort)
O(n^2)	Quadratic	Nested loops (Bubble Sort)
O(2^n)	Exponential	Recursive Fibonacci

💡 Why It Matters:

- Predicts **scalability** and performance.
- Guides **algorithm selection**.
- Helps in **optimizing** code and choosing data structures
