Working with Arrays in C#

Array Declaration and Initialization

Declaration

You can declare an array by specifying the type of its elements and the array brackets []:

```csharp
int[] numbers;
```

Initialization

You can initialize an array at the time of declaration using curly braces {}:

```csharp
int[] numbers = new int[5]; // Initializes an array of 5 integers with default values (0)
int[] primes = { 2, 3, 5, 7, 11 }; // Initializes an array with specified values
```

You can also use the new keyword:

```csharp
int[] numbers = new int[] { 2, 4, 6, 8, 10 };
```

Accessing Array Elements

Array elements are accessed using their indices, starting from 0:

```csharp
int firstPrime = primes[0]; // Accesses the first element
primes[2] = 17; // Sets the third element to 17
```

Array Properties and Methods

Length: Gets the total number of elements in all dimensions of the array.

```csharp
int length = numbers.Length; // Gets the length of the array
```

Rank: Gets the number of dimensions of the array.

```csharp
int rank = numbers.Rank; // For a single-dimensional array, this will be 1
```

Multidimensional Arrays

C# supports both rectangular and jagged arrays.

Rectangular Arrays

A rectangular array is a multidimensional array where each row has the same number of columns:

```csharp
int[,] matrix = new int[3, 3]; // A 3x3 matrix
matrix[0, 0] = 1; // Sets the first element of the matrix
```

Jagged Arrays

A jagged array is an array of arrays, where each "row" can have a different length:

```csharp
int[][] jaggedArray = new int[3][]; // An array of 3 arrays
jaggedArray[0] = new int[5]; // First array has 5 elements
jaggedArray[1] = new int[3]; // Second array has 3 elements
```

jaggedArray[2] = new int[2]; // Third array has 2 elements

```

## Iterating Through Arrays

You can use loops to iterate through arrays.

For Loop

```csharp
for (int i = 0; i < primes.Length; i++)
{
    Console.WriteLine(primes[i]);
}
```

Foreach Loop

```csharp
foreach (int prime in primes)
{
    Console.WriteLine(prime);
}
```

Common Array Operations

Copying Arrays

You can use Array.Copy or CopyTo to copy elements from one array to another:

```csharp
int[] destination = new int[primes.Length];
```
```

Array.Copy(primes, destination, primes.Length);

```

Sorting Arrays

The Array.Sort method sorts the elements of an array:

```csharp

Array.Sort(primes);

```

Searching Arrays

Use Array.IndexOf to find the index of an element:

```csharp

int index = Array.IndexOf(primes, 7); // Returns the index of 7 in the primes array

```

Array Class Methods

C# provides a number of static methods for arrays through the Array class, such as Clear, Resize, Reverse, and more.

```csharp

Array.Reverse(primes); // Reverses the order of elements

```

Example Program

Here's a simple example that demonstrates the creation, initialization, and manipulation of an array:

```csharp

using System;

```
class Program
{
    static void Main()
    {
        // Declare and initialize an array
        int[] numbers = { 1, 2, 3, 4, 5 };

        // Access and modify array elements
        numbers[0] = 10;

        // Iterate through the array
        foreach (int number in numbers)
        {
            Console.WriteLine(number);
        }

        // Array properties
        Console.WriteLine("Length: " + numbers.Length);

        // Copy array
        int[] copy = new int[numbers.Length];
        Array.Copy(numbers, copy, numbers.Length);

        // Sort array
        Array.Sort(copy);
```

```
        // Print sorted array
        Console.WriteLine("Sorted array:");
        foreach (int number in copy)
        {
            Console.WriteLine(number);
        }
    }
}
```