

Core C# Programming	
Arrays in C#	<p>Array : a collection of the same data type , stored in contiguous memory location and can be accessed via zero based index.</p> <p>Why we need array when we have so many types of variable in C# ?</p> <pre>int [] marks = new int [100];</pre> <p><b>We need arrays to simplify data storage, processing, and scalability when dealing with multiple related values.</b></p> <p>What are the features of Arrays/ in what scenarios we should use them ?</p> <p>Limitation of using arrays in C#?</p> <p>types of arrays in C#</p> <ul style="list-style-type: none"> <li>Single array</li> <li>Multi-dimension</li> <li>Jagged array</li> </ul>
Looping Construct	<ul style="list-style-type: none"> <li>Loops and Statement</li> <li>IF statement</li> <li>For loop</li> <li>Foreach loop</li> <li>Switch statement</li> <li>While loop</li> <li>Do-while loop</li> </ul>

Access modifiers in C#	<ul style="list-style-type: none"> <li>Access modifiers</li> <li>Private</li> <li>Public</li> <li>Protected</li> <li>Internal</li> <li>Protected internal</li> </ul>
Methods in C#	<ul style="list-style-type: none"> <li>Methods</li> <li>Void method</li> <li>With parameter</li> <li>Return type method</li> <li>Static method</li> <li>Instance method</li> <li>Namespaces</li> </ul>

## Unit testing

today's case study :

### User Story 1: Version Control Adoption : CI/CD operations via Github

**As a** software development team

**I want** a version control system to track and manage code changes

**So that** multiple developers can work in parallel without conflicts and we can safely revert to stable versions when needed.

#### Acceptance Criteria

- All code changes are committed to a central repository
- Each change is traceable (who, what, when)
- Developers can create branches for features and fixes
- Code can be merged and reviewed before release
- Remote repositories enable team-wide collaboration

---

### User Story: Enabling Faster, Safer Software Delivery

**As a business leader,**

I want development teams to collaborate efficiently and release software updates reliably,  
**so that** the organization can respond faster to market needs while minimizing operational risk.

#### The Challenge

As teams grow, managing frequent changes becomes complex.

Without structured control and automation:

- Errors go unnoticed
- Releases are delayed
- Accountability becomes unclear
- Customer impact increases

Solution 

- Version control to maintain a single , trusted source for all changes.
- Automated delivery pipeline (CI/CD) to validate and deploy updates consistently.

this ensures

- Every change is tracked and auditable.
- Quality checks happens automatically
- deployments are predictable and repeatable.

What is the business outcome of above system adoption :

- Reduced release failure.
- faster time to market.
- Improved product stability
- Higher team productivity
- Better customer trust.

//Steps for creating Arrays in C#

//Step 1: Declare the array of type int

//Step 2: Initialize the array with size 5

//Step 3: Assign values to each index of the array

//Step 4: Print the values of the array using a for loop

The screenshot shows the Microsoft Visual Studio IDE. The top menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and a Search dropdown. The toolbar below has icons for file operations like Open, Save, and Print, along with other development tools. The solution explorer on the left shows a project named "Day\_2\_Demo1\_Arrays\_In\_C\_Sharp". The code editor window contains "Program.cs" with the following C# code:

```

//Step 1: Declare the array of type int
//Step 2: Initialize the array with size 5
//Step 3: Assign values to each index of the array
//Step 4: Print the values of the array using a for loop

int[] numbers = ...;
numbers[0] = 10;
numbers[1] = 20;
numbers[2] = 30;
numbers[3] = 40;
numbers[4] = 50;

//ForLoop consists of (Initialization; Condition; Increment/Decrement)
//For loop is used for iterating over a sequence (like an array) a specific number of times
for (int i = 0; i < numbers.Length; i++) // here numbers.Length gives the size of the array
{
    Console.WriteLine("Element at index " + i + ": " + numbers[i]);
}

//In C# we have get, set methods to access and modify the elements of an array

```

The code editor highlights several lines with red numbers 1, 2, and 3, corresponding to the steps listed above. Below the code editor is a terminal window titled "Microsoft Visual Studio" showing the output of the program:

```

Hello, World!
Element at index 0: 10
Element at index 1: 20
Element at index 2: 30
Element at index 3: 40
Element at index 4: 50

```

The terminal window also shows the path "C:\Users\Parth\source\repos\Wipro\_MS\_Dynam" and a message "You are screen sharing".

```
//Case study : Using Arrays to Manage Student Grades  
//if i want to store marks of students in different subjects: "Subject wise marks for each student"  
//Step 1: Declare a 2D array to store marks of 3 students in 4 subjects  
//Step 2: Initialize the array with sample marks  
//Step 3: Calculate and print the average marks for each student  
//Step 4: Calculate and print the average marks for each subject  
//Step 5: Find and print the highest and lowest marks in the class  
// Syntax for declaring 2D array as per above scenario is  
// dataType[,] arrayName = new dataType[rows, columns];  
  
// There is a difference between storing marks and storing subject wise marks for each student  
// In first case each row represents a student and each column represents a subject  
// In second case each row represents a subject and each column represents a student
```

---

A mid-sized **e-commerce platform** is developing an **Order Management System (OMS)** using C#.

The system must:

- Store customer orders
- Add and remove orders dynamically
- Search, sort, and update orders in real time
- Scale during seasonal sales (Flash Sales, Festivals)

**order[] orders = new orders[100]; // implementing an array**

- **it can take only 100 orders**
  - **it can access stored data sequentially.**
-

In C# we have collections

Array Limitation	Collection Solution
Fixed size	Dynamic resizing
Manual insert/delete	Add(), Remove()
Custom search logic	Find(), Any()
Sorting complexity	Sort()
Memory wastage	Auto capacity management (Common language runtime CLR)

type of collection in C# 

- 1) Generic Collection : List
- 2) Non generic Collection

Non generic Collection ( system.collections)	Generic Collection (system.collections.generic)
Stores data as object .	Stores data as specific type <T>
boxing/unboxing is required for value types	it is not required
performance is slower	faster in performance

<b>compile time errors are not detected</b>	<b>they are detected</b>
<b>higher memory consumption</b>	<b>low memory consumption</b>
<b>ArrayList , hashtable, Stack, queue, Dictionary(K,V)</b>	<b>List&lt;t&gt;, hashtable&lt;t&gt;, Stack&lt;t&gt;, queue&lt;t&gt;, Dictionary(tkey&gt;, tvalue&gt;)</b>

The screenshot shows the Microsoft Visual Studio IDE interface with the title bar "Microsoft Visual Studio Debug". The main area displays the following C# program output:

```

Hello, World!
Non Generic Collection Implementation
total no of elelemnt in the collection 1
Right now the memory of collection is 4
total no of elelemnt in the collection 3
Right now the memory of collection is 4
total no of elelemnt in the collection 5
Right now the memory of collection is 8
total no of elelemnt in the collection 6
Right now the memory of collection is 8
Items in the Order Collection:
laptop
12345
99.99
01-06-2024 00:00:00
True
A
Items in the Order Collection with their types:
laptop - Type: System.String
12345 - Type: System.Int32
99.99 - Type: System.Double
01-06-2024 00:00:00 - Type: System.DateTime
True - Type: System.Boolean
A - Type: System.Char

```

At the bottom of the output window, the path is shown as: C:\Users\Parth\source\repos\Wipro MS\_Dynamics\_8thJan26\Day 2 C# Arrays, loops and collections\_IN\_C\_Sharp\bin\Debug\net8.0\Day 2\_Demo2\_OMS\_Collections\_IN\_C\_Sharp.exe (process 27436). Press any key to close this window . . .

The screenshot shows a Microsoft Visual Studio interface. On the left is the code editor with a file named 'Program.cs'. The code demonstrates creating a class 'Order' and a generic collection 'List<Order>'. It adds two sample orders to the collection and then prints them to the console. The Solution Explorer on the right shows the project structure with files 'order.cs' and 'Program.cs'. The 'order.cs' file contains the definition of the 'Order' class with properties: OrderId (int), ProductName (string), Quantity (int), and Price (double). The 'ToString()' method is also defined. The 'Program.cs' file contains the main logic. The Task List and Output windows are also visible at the bottom.

```

79 //Step 1: Create a class to represent an Order
80
81 //Step 2: Create a generic collection to store Order objects
82
83
84
85 List<Order> orderList = new List<Order>();
86
87 //Adding some sample orders to the collection
88 orderList.Add(new Order { OrderId = 1, ProductName = "Laptop", Quantity = 2, Price = 1500 });
89 orderList.Add(new Order { OrderId = 2, ProductName = "Mobile", Quantity = 5, Price = 800 });
90
91 //displaying the orders in the collection
92 Console.WriteLine("Orders in the Order Collection:");
93 foreach (var order in orderList)
94 {
95     Console.WriteLine(order);
96 }
97
98
99 Orders in the Order Collection:
100 OrderId: 1, ProductName: Laptop, Quantity: 2, Price: 1500
101 OrderId: 2, ProductName: Mobile, Quantity: 5, Price: 800

```

### Case Study on types of collections :

A **banking application** needs a **fast lookup mechanism** to fetch **customer profiles** using a **Customer ID**.

### System Requirements

- Store thousands of customer records
- Retrieve customer details in real time( faster)
- Ensure data accuracy and performance(need to consider generic Collection)
- Support future scalability( No way arrays can be considered, we have to think in terms of collections only)

Non generic collection hashtable	generic collection Dictionary(tkey, tvalue) //To overcome above issue we can use
-------------------------------------	--

<pre>Hashtable customer = new Hashtable(); customer.Add("C001", "John Doe"); customer.Add("C002", "Jane Smith"); customer.Add("C003", "Alice Johnson"); customer.Add("c004", 45000);</pre>	<pre>generic dictionary Dictionary&lt;string, string&gt; customerGeneric = new Dictionary&lt;string, string&gt;(); customerGeneric.Add("C001", "John Doe"); customerGeneric.Add("C002", "Jane Smith"); customerGeneric.Add("C003", "Alice Johnson"); customerGeneric.Add("c004", 45000.ToString());//this line will cause compile time error</pre>
<ul style="list-style-type: none"> <li>● No type safety : it is allowing mixed types/value types</li> <li>● run time errors</li> <li>● overhead of performance</li> <li>● poor maintainability</li> </ul>	<ul style="list-style-type: none"> <li>● type safety</li> <li>● no run time error</li> <li>● no overhead</li> <li>● good maintainability</li> </ul>

case study based on different types of constructs in C#:

A **Smart Parking System** is developed to manage vehicle entry, parking allocation, billing, and exit processing.

The system must:

- Track parking slots
  - Handle vehicle types
  - Continuously monitor availability
  - Process user choices until exit
1. Loop : for initialized parking slots.
  2. for each loop can be used for Displaying parking status
  3. Switch : vehicle type handling ( car, bike, truck, others)
  4. While :
  5. Do While

below steps can be used for above case study implementation 

Step 1: Initialize parking with the help of for loop

Step 2: parking vehicles

Step 3: exit vehicles

Step 4: calculate charges based on vehicle type

Step 5: Keeping the application running until the user exits.

C# Concept	Used For	Reason
for	Slot initialization	Fixed iteration
foreach	Display slot status	Safe traversal
switch	Vehicle type logic	Multiple conditions
while	Slot search	Condition-based
do-while	Menu loop	At least once execution

Why we need function in C#? :

1. Reusability
2. Better readability
3. Easier to maintain
4. Modular code

A school needs a small program to:

- Calculate total marks
- Calculate average marks
- Determine pass or fail

Step 1: Function to calculate total marks : should accept parameters and returns a value

Step 2: call function inside main

