



C# (Cont..)



Main content



Part 1

- I/O in C#

Part 2

- Interface in C#

Part 3

- Collections in C#

Part 4

- Delegate in C#





I/O in C#



- ❖ Basic structure and meaning of internal input and output commands in C#
- ❖ In C#, there are 5 commands used to input and output:
 - Console.**Write**();
 - Console.**WriteLine**();
 - Console.**Read**();
 - Console.**ReadLine**();
 - Console.**ReadKey**();





❖ **Console.Write();**

- **Syntax:** Console.Write(<value to print to screen>);
- **Meaning:** Print the value to the console screen.
- This value can be a character, a string, a value that can be converted to string type

```
static void Main(string[] args)
{
    Console.Write("DH Nong Lam");
    Console.Write(10);
}
```





I/O in C#



❖ Console.WriteLine();

- **Syntax:**
 - Console.**WriteLine**(< value to print to screen >);
- **Meaning:**
 - This command is similar to Console.Write() but when printing the value to the screen, the cursor will automatically move to the next line.
 - Additionally, we can use characters “\n”

```
static void Main(string[] args){  
    Console.Write("ĐH Nông Lâm\n");  
    Console.WriteLine(5);  
    Console.Write(Environment.NewLine);  
    Console.Write(true);  
    Console.ReadLine();  
}
```





I/O in C#



❖ Print out the value of the variable :

- You can specify where to print the value of a variable in a string using syntax **{<count_number>}**.

❖ Ex:

❖ `int a = 5;`

❖ `Console.Write("a = {0}", a);`

❖ Syntax:

❖ `Console.Write("{0} {1} {2} {...}", <value 0>, <value 1>, <value 2>, ... <value n>);`

❖ In there:

❖ **<value 0>** will be filled in position 0, similarly for the remaining values.





I/O in C#



❖ Console.Read();

- **Syntax:**
 - Console.**Read()**;
- **Meaning:**
 - Read 1 character from the keyboard and return it as an integer (ASCII code)
 - **Note:** this command cannot read function keys such as Ctrl, Shift, Alt, Caps Lock, Tab, . . .
 - **Ex:** Enter a → 97 (which is the ASCII code of character a).

❖ Console.ReadLine();

- **Meaning:** Read data from the keyboard until it stops when it encounters a newline character (when it encounters the enter key, it stops) and the read value is always a string.





❖ Console.ReadKey();

- **Syntax:**
- Console.**ReadKey**(< **bool parameter** >)
- **Meaning:**
 - This command is also used to read a character from the keyboard
 - **bool parameter** includes 2 values: **true** or **false**. If true is passed, the pressed key will not be displayed on the console screen but will be read implicitly. Conversely, if false is passed, the pressed key will be displayed on the screen





Foreach loop in C#



```
foreach (<Data Type> <Variable name> in <array name or collection name>)
```

```
{
```

```
    // Code xử lý
```

```
}
```

❖ In there:

- keywords **foreach**, **in** is a required keyword.
- <Data type> is the data type of the elements in the array or collection.
- <Variable name> is the name of a temporary variable that represents the element being considered when browsing the array or collection.
- < **array name or collection name** > is the name of the array or collection to be traversed.





Foreach loop in C#



```
int[] IntArray = { 1, 5, 2, 4, 6 };
```

```
int Sum = 0;
```

```
/*Use foreach to browse the array and print the  
values of the elements in the array, calculate the  
total of the elements in the array. */
```

```
foreach (int item in IntArray)
```

```
{
```

```
    Console.WriteLine("\t" + item);
```

```
    Sum += item;
```

```
}
```

```
Console.WriteLine("\n Sum = " + Sum);
```





Foreach loop in C#



Compare For and Foreach ???

Criteria	For	Foreach
Element retrieval capabilities	Random access (can call any element in the array to use)	Sequential access (only the value of the element under consideration can be used)
Change the value of elements	yes	No
Iterate arrays and sets when the number of elements in the array or set is unknown	No	Yes





❖ What are interfaces? Why use interface?

- Interface (interface or communication class) is a set of components that only have declarations without definitions (like abstract methods).
- These components can be:
 - Method
 - Properties
 - Events
- An interface is understood as a template that every class that implements it must follow. The interface will define the part (declaration) and the classes that implement this interface will have to implement those methods.





❖ Interface characteristics

- Overriding a component in an interface does not require the override keyword.
- Cannot declare access scope for components inside the interface. These components will default to public.
- Interface has no constructor
- Classes can implement multiple interfaces at the same time (multiple inheritance).
- An interface can inherit many other interfaces but cannot inherit any class.





Interface in C#



❖ **Ex:**

```
interface ISpeak{
```

```
void Speak();
```

```
}
```

```
class Animal : ISpeak
```

```
{
```

```
/* ..... */
```

```
public void Speak()
```

```
{
```

```
Console.WriteLine("Animal is speaking. . .");
```

```
}}
```





Collections in C#



❖ What are Collections in C#?

- Classes support storing, managing, and manipulating objects in an orderly manner.
- These classes are located in the **System.Collections namespace**.

❖ Some features of Collections:

- As a dynamically sized array
- No need to declare size when initializing.
- The number of elements in the array can be increased or decreased flexibly.
- Can store a collection of objects of many different types.
- Supports many methods to manipulate collections such as: search, sort, reverse, . . .
- Each collection is organized into a class, so the object needs to be initialized before use.





Collections in C#



- ❖ When to use Collection?
- ❖ We have already learned about a data type used to manage lists of objects, which is the array type.
- ❖ So what makes Collections better than arrays?
- ❖ When to use Arrays and when to use Collections?





Collections in C#



- ❖ What are the strengths of Collections?
- ❖ Inside Collections there are many diverse classes that support different purposes.
- ❖ If Arrays can only access elements through indexes, Collections can access them through indexes or keys.
- ❖ For lists that need a lot of searching operations, Collections also has a support class that makes searching much faster than primitive arrays (HASHTABLE IN C#).
- ❖ In case the list needs to change the number of elements continuously (add or remove elements), Collections also supports it (ARRAYLIST IN C#).
- ❖ In addition, the System.Collections namespace also supports two classic data structures: STACK and QUEUE.





Collections in C#



LỚP	MÔ TẢ
ArrayList	Lớp cho phép lưu trữ và quản lý các phần tử giống mảng. Tuy nhiên, không giống như trong mảng, ta có thể thêm hoặc xoá phần tử một cách linh hoạt và có thể tự điều chỉnh kích cỡ một cách tự động.
HashTable	Lớp lưu trữ dữ liệu dưới dạng cặp Key – Value . Khi đó ta sẽ truy xuất các phần tử trong danh sách này thông qua Key (thay vì thông qua chỉ số phần tử như mảng bình thường).
Stack	Lớp cho phép lưu trữ và thao tác dữ liệu theo cấu trúc LIFO (Last In First Out).
Queue	Lớp cho phép lưu trữ và thao tác dữ liệu theo cấu trúc FIFO (First In First Out).
BitArray	Lớp cho phép lưu trữ và quản lý một danh sách các bit. Giống mảng các phần tử kiểu bool với true biểu thị cho bit 1 và false biểu thị cho bit 0. Ngoài ra BitArray còn hỗ trợ một số phương thức cho việc tính toán trên bit.





ArrayList in C#:



- ❖ ArrayList in C#:
- ❖ Is a Collections that helps store and manage a list of objects in array style (accessing the internal elements through the index).
- ❖ Elements can be added or removed dynamically and can be resized automatically.
- ❖ To use Collections in .NET we need to add the System.Collections library with the command: `using System.Collections;`





ArrayList in C#:



ArrayList in C#:

```
ArrayList MyArray = new ArrayList();
```

```
// initialize an empty ArrayList
```

```
ArrayList MyArray2 = new ArrayList(5);
```

```
// initialize an ArrayList and specify an initial Capacity of 5
```

Additionally, you can also initialize an ArrayList containing elements copied from another Collections:

```
/* Initialize an ArrayList with the same size as  
MyArray2.Copy all elements in MyArray2 into MyArray3.*/
```

```
ArrayList MyArray3 = new ArrayList(MyArray2);
```





ArrayList in C#:



Some properties and methods are available in ArrayList

Tên thuộc tính	Ý nghĩa
Count	Trả về 1 số nguyên là số phần tử hiện có trong ArrayList .
Capacity	Trả về 1 số nguyên cho biết số phần tử mà ArrayList có thể chứa (sức chứa). Nếu số phần tử được thêm vào chạm sức chứa này thì hệ thống sẽ tự động tăng lên. Ngoài ra ta có thể gán 1 sức chứa bất kỳ cho ArrayList .





ArrayList in C#:



TÊN PHƯƠNG THỨC	Ý NGHĨA
Add(object Value)	Thêm đối tượng Value vào cuối ArrayList .
Clear()	Xoá tất cả các phần tử trong ArrayList .
Clone()	Tạo 1 bản sao từ ArrayList hiện tại.
Contains(object Value)	Kiểm tra đối tượng Value có tồn tại trong ArrayList hay không.
GetRange(int StartIndex, int EndIndex)	Trả về 1 ArrayList bao gồm các phần tử từ vị trí StartIndex đến EndIndex trong ArrayList ban đầu.
IndexOf(object Value)	Trả về vị trí đầu tiên xuất hiện đối tượng Value trong ArrayList . Nếu không tìm thấy sẽ trả về -1.





ArrayList in C#:



Insert(int Index, object Value)	Chèn đối tượng Value vào vị trí Index trong ArrayList .
LastIndexOf(object Value)	Trả về vị trí xuất hiện cuối cùng của đối tượng Value trong ArrayList . Nếu không tìm thấy sẽ trả về -1.
Remove(object Value)	Xoá đối tượng Value xuất hiện đầu tiên trong ArrayList .
Reverse()	Đảo ngược tất cả phần tử trong ArrayList .
Sort()	Sắp xếp các phần tử trong ArrayList theo thứ tự tăng dần.
ToArray()	Trả về 1 mảng các object chứa các phần tử được sao chép từ ArrayList .





ArrayList in C#:



- ❖ Sort() method in ArrayList
- ❖ The Sort() method will sort the list in ascending order.
- ❖ If the list consists of objects where each object is a class with many attributes, which attribute does the Sort function know to sort in ascending order?
- ❖ The extended Sort function in ArrayList has the following syntax:
- ❖ Sort(IComparer comparer)





ArrayList trong C#:



- ❖ Sort() method in ArrayList
- ❖ Sort(IComparer comparer) Uses:
 - ❖ Allows you to define your own arbitrary arrangement The parameter passed is a class that inherits from the IComparer interface.
 - ❖ Interface IComparer contains a single method: `int Comparer(object x, object y)`.
 - ❖ This method will return 3 values:
 - Less than 0 if $x < y$.
 - Greater than 0 if $x > y$.
 - Equals 0 if $x = y$.





Map in C#:

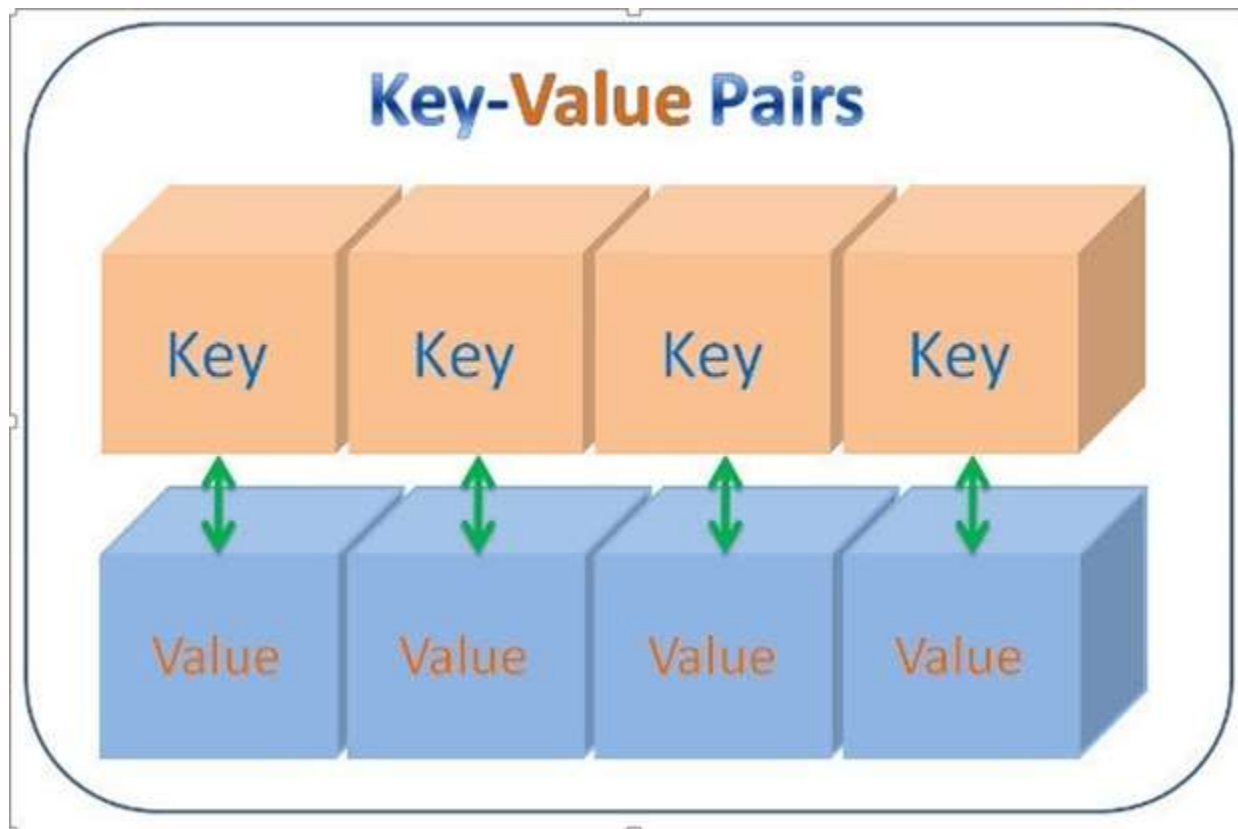


- ❖ What is Map?
- ❖ In some programming languages, Map is also called Dictionary (like Python or C#). In Java we often call it Map.
- ❖ When accessing data with data structures such as arrays, you will use a parameter called an **index**, such as `arr[1]`, `str[2]`, ... For the map data structure, to retrieve data whether you will use a parameter called **key**
- ❖ A map data structure is a data structure that maps (ánh xạ) between a key and the value of that key (value).
- ❖ In this data structure, each key will receive a different value






Map in C#:





Map in C#:



- ❖ There are many applications of maps.
- ❖ **Some examples:**
- ❖ Provide a list of phone numbers with the subscriber's name. The input request is a phone number (key), please provide the name of the subscriber (value)
- ❖ Provide a list showing the lateness history of employees of a certain company. Find out which employee (key) has the most number of late times (value)?
- ❖ Given a list of IPs with domains. Please return the ip (value) corresponding to the domain (key) 



Delegate in C#



❖ What is Delegate?

- Delegates in C# are similar to function pointers in C or C++.
- Delegate is a **reference type variable**(references) that holds a **reference to a method**.
- Delegate is a normal variable, this variable contains the method we need to call. The value of the Delegate variable is now a reference to the method. can be changed while the program is running.

❖ Declaring Delegate in C#

- **delegate** **<Return type>** **< delegate_name>** (**< parameter list if any >**);

Ex:

- **delegate** **int** MyDelegate(**string** s);





Delegate in C#



❖ Initializing and using Delegate in C#

- When the **Delegate type** is declared, the Delegate object must be created with the **new keyword** and be referenced to a specific method. This method must have **the same return type and parameters** as the created Delegate.
- When creating a Delegate, the parameter passed with the **new** expression has only the **method name**, but no parameters to the method. That is, **only pass the method name in.**
- **Ex:**





Delegate trong C#



```
class Program {  
    delegate int MyDelegate(string s);  
    static void Main(string[] args)  
    {  
        Console.OutputEncoding = Encoding.Unicode;  
        MyDelegate convertToInt = new MyDelegate(ConvertStringToInt);  
        string numberSTR = "35";  
        int valueConverted = convertToInt(numberSTR);  
        Console.WriteLine("Giá trị đã convert thành int: " + valueConverted);  
        Console.ReadLine();  
    }  
    static int ConvertStringToInt(string stringValue)  
    {  
        int valueInt = 0;  
        Int32.TryParse(stringValue, out valueInt);  
        Console.WriteLine("Đã ép kiểu dữ liệu thành công");  
        return valueInt;    }  
}
```

2

3

1



31

34



Delegate Example C#



```
using System;
delegate int CalculatorDelegate(int num1, int num2);
class Program {
    static void Main(string[] args) {
        // Khởi tạo delegate
        CalculatorDelegate calculatorDelegate = new CalculatorDelegate(MaxNumber);
        // Gọi phương thức thông qua delegate
        int result = calculatorDelegate(10, 20);
        // In ra giá trị lớn nhất của hai số
        Console.WriteLine("Max number: " + result);
    }
    static int MaxNumber(int num1, int num2) {
        return (num1 > num2) ? num1 : num2;
    }
}
```





Delegate Example C (cont..)#



```
using System;
delegate int Calculator(int a, int b);
class Program {
    static int Add(int a, int b) {
        return a + b;
    }
    static int Subtract(int a, int b) {
        return a - b;
    }
    static void Main(string[] args) {
        Calculator calculator = new Calculator(Add);
        int result = calculator(10, 5);
        Console.WriteLine("10 + 5 = " + result);
        calculator = new Calculator(Subtract);
        result = calculator(10, 5);
        Console.WriteLine("10 - 5 = " + result);
        Console.ReadKey();
    }
}
```





Exercise



❖Viết chương trình **Quản lý thông tin sinh viên** gồm các Class:

☞ SinhVien(MSSV, HoVaTen, GioiTinh, NgaySinh, DS MonHoc)

☞ MonHoc(MSMH, SoTinChi, TenMonHoc, MSGV)

☞ Sử dụng Collections tương ứng để lưu trữ danh sách sinh viên, thực hiện các phương thức tìm kiếm, thêm xóa sửa sv trong danh sách. Sắp xếp tăng dần theo điểm trung bình.





Quiz Test



❖ **Yêu cầu:** Mỗi bàn chuẩn bị một tờ giấy.

Điền thông tin gồm: MSSV, Họ và Tên.

❖ **Câu hỏi:**

Sự khác nhau và giống nhau của C# và Java, cho các ví dụ minh họa để làm sáng tỏ lập luận của nhóm!





Q & A

