# CHƯƠNG 1: GIỚI THIỆU VỀ C#

## 1.1. Giới thiệu về Microsoft .NET Framework

### 1.1.1. .NET Framework là gì ?

.NET Framework là nền tảng phát triển hoàn hảo của Microsoft, cung cấp cho lập trình viên các thư viện dùng chung hỗ trợ cho việc phát triển các kiểu ứng dụng khác nhau bao gồm:

- Ứng dụng ASP.NET
- Ứng dụng Windows Form
- Web Services
- Windows Services
- Ứng dụng mạng và các ứng dụng điều khiển truy cập từ xa

### 1.1.2. Các thành phần của Microsoft .NET Framework

.NET Framework gồm hai thành phần chính là: Common Language Runtime (CLR) và thư viện lớp.

- CLR: là nền tảng của .NET Framework, giúp Microsoft có thể tích hợp nhiều ngôn ngữ lập trình khác nhau như: VB.NET, C#.NET, ASP.NET,… vào bộ công cụ lập trình Visual Studio.NET. Đồng thời giúp các ứng dụng viết trên các ngôn ngữ này có thể chạy được chung trên nền tảng hệ điều hành Windows. Sở dĩ Microsoft có thể làm được điều này, bởi vì các ngôn ngữ lập trình đều được biên dịch ra một ngôn ngữ trung gian (Intermediate Language – IL) và sử dụng chung kiểu dữ liệu hệ thống (Common Type System). Sau đó CLR sử dụng một trình biên dịch gọi là Just-in-Time (JIT) Compiler chuyển các đoạn mã IL thành mã máy và thực thi.

  Ngoài ra CLR còn làm các thành phần khác như:

  - *Garbage Collection (GC)*: Gọi là bộ phận thu gom rác; có chức năng tự động quản lí bộ nhớ. Tại một thời điểm nhất được định sẵn, GC sẽ tiến hành thực hiện việc thu hồi những vùng nhớ không còn được sử dụng.
  - *Code Access Security (CAS)*: Cung cấp quyền hạn cho các chương trình, tùy thuộc vào các thiết lập bảo mật của máy. Chẳng hạn, thiết lập bảo mật của máy cho phép chương trình chạy trên đó được sửa hay tạo file mới, nhưng không cho phép nó xóa file. CAS sẽ chăm sóc các đoạn mã, không cho phép chúng làm trái với các qui định này.
  - *Code Verification*: Bộ phận chứng nhận đoạn mã. Bộ phận này đảm bảo cho việc chạy các đoạn mã là đúng đắn, và đảm bảo an toàn kiểu dữ liệu.

ngăn chặn các đoạn mã hoạt động vượt quyền như truy nhập vào các vùng nhớ không được phép.

- Thư viện lớp: là một tập hợp lớn các lớp được viết bởi Microsoft, những lớp này được xây dựng một cách trực quan và dễ sử dụng; cho phép lập trình viên thao tác rất nhiều các tác vụ sẵn có trong Windows.
  - Base class library: Đây là thư viện các lớp cơ bản nhất, được dùng trong khi lập trình hay bản thân những người xây dựng .NET Framework cũng phải dùng nó để xây dựng các lớp cao hơn. Một số thư viện lớp base class library như: String, Interger, Exception, …
  - ADO.NET và XML: Bộ thư viện này gồm các lớp dùng để xử lý dữ liệu. ADO.NET thay thế ADO để trong việc thao tác với các dữ liệu thông thường. Các lớp đối tượng XML được cung cấp để xử lý các dữ liệu theo định dạng mới XML. Một số thư viện trong ADO.NET và XML như: sqlDataAdapter, SqlCommand, DataSet, …
  - Windows Forms: Bộ thư viện về lập trình Windows Forms gồm các lớp đối tượng dành cho việc xây dựng các ứng dụng Windows cơ bản. Một số thư viện thường dùng như: Form, UserControl, TextBox, Label, Button, ComboBox, ListBox, ListView, TreeView, …
  - Web Services: là các dịch vụ được cung cấp qua Web (hay Internet). Dịch vụ được coi là Web Service không nhắm vào người dùng mà nhằm vào người xây dựng phần mềm. Web Services có thể dùng để cung cấp các dữ liệu hay một chức năng tính toán.
  - ASP.NET: Ứng dụng Web xây dựng bằng ASP.NET tận dụng được toàn bộ khả năng của .NET Framework. ASP.Net cung cấp một bộ các Server Control để lập trình viên bắt sự kiện và xử lý dữ liệu của ứng dụng như đang làm việc với ứng dụng của Windows. Một số thư như: WebControl, HTML Control, …

## 1.2. Tổng quan về C#

Vào ngày 12/2/2002, Microsoft chính thức cho ra mắt Visual Studio 2002 cùng với bản .NET Framework 1.0. Với nền tảng .NET Framework thì các ứng dụng về Windows Form, web, … đều có thể được xây dựng trên nền tảng .NET. Hai ngôn ngữ chính mà Microsoft sử dụng để phát triển các ứng dụng trên là C# và Visual Basic.NET.

C# được phát triển bởi đội ngũ kỹ sư của Microsoft, trong đó hai người dẫn đầu là Anders Hejlsberg và Scott Wiltamult. Vì là ngôn ngữ ra đời sau các ngôn ngữ khác

như: Java, C, C++, Perl, Visual Basic, … do đó C# có những đặc điểm nổi trội và mạnh mẽ hơn nhưng không kém phần đơn giản.

- Các đặc điểm của C#:

  Microsoft thêm vào những đặc điểm mới vào để C# dễ sử dụng hơn:

  - C# là ngôn ngữ đơn giản: Các cú pháp, biểu thức, toán tử đều giống với C, C++. Ngoài ra C# loại bỏ một số khái niệm phức tạp khó hiểu như: Con trỏ, Macro, đa kế thừa, template, lớp cơ sở ảo.
  - C# là ngôn ngữ hiện đại: Có tất cả các đặc tính của một ngôn ngữ hiện đại như: xử lý ngoại lệ, thu gom bộ nhớ tự động, kiểu dữ liệu mở rộng, và bảo mật mã nguồn.
  - C# là ngôn ngữ hướng đối tượng: Là một ngôn ngữ thuần hướng đối tượng, do đó chứa tất cả các đặc điểm để có thể lập trình theo hướng đối tượng: Sự đóng gói (Encapsulation), sự thừa kế (Inheritance), và sự đa hình (Polymorphism).
  - C# là ngôn ngữ mạnh mẽ và mềm dẻo: Có thể sử dụng cho những dự án khác nhau như: xử lý văn bản, ứng dụng đồ họa, bảng tính, …
  - C# là ngôn ngữ có ít từ khóa: là ngôn ngữ sử dụng giới hạn các từ khóa, chỉ khoảng 80 từ khóa và hơn 10 kiểu dữ liệu được xây dựng sẵn như bảng 1.1.

*Bảng 1.1: Bảng mô tả các từ khóa của C#*

| | | | | | |
|---|---|---|---|---|---|
| abstract | default | foreach | object | sizeof | unsafe |
| as | delegate | goto | operator | stackalloc | ushort |
| base | do | if | out | static | using |
| bool | double | implicit | override | string | virtual |
| break | else | in | params | struct | volatile |
| byte | enum | int | private | switch | void |
| case | event | interface | protected | this | while |
| catch | explicit | internal | public | throw | decimal |
| char | extern | is | readonly | true | for |
| checked | false | lock | ref | try | null |
| class | finally | long | return | typeof | short |
| const | fixed | namespace | sbyte | uint | unchecked |

| | | | | | |
|---|---|---|---|---|---|
| continue | float | new | sealed | ulong | |

## 1.3. Cấu trúc tổng quát của một chương trình C#

Các đoạn mã C# có thể được soạn thảo trong một trình soạn thảo văn bản bất kỳ, sau khi biên soạn xong cần lưu tập tin với đuôi .cs

### 1.3.1. Soạn thảo bằng Notepad

Trong quá trình soạn thảo cần lưu ý C# là một ngôn ngữ thuần hướng đối tượng. Do đó tất cả đoạn mã của chương trình đều phải thuộc lớp.

Ví dụ 1.1: Soạn thảo một chương trình C# in dòng chữ Hello World ra màn hình

- Bước 1:

```
namespace hello
{
  class Program
  {
    static void Main()
    {
        System.Console.WriteLine("Hello World");
        System.Console.ReadLine();
    }
  }
}
```

Giải thích:

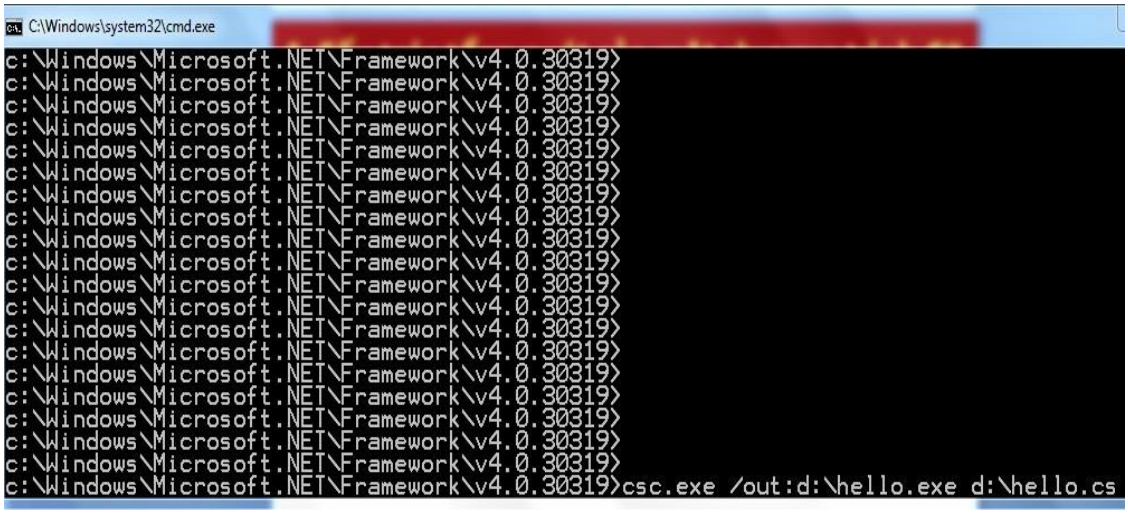
- Ở đoạn chương trình trên, hàm Main được khai báo là static. Việc khai báo này cho biết hàm Main có thể được sử dụng mà không cần thông qua đối tượng
- Hàm WriteLine nằm trong lớp **Console**, do đó nếu sử dụng phải khai báo thêm dòng lệnh "Using System;" phía trên cùng để việc gõ lệnh được nhanh chóng hơn như sau: "**Console**.WriteLine()". Nếu không khai báo "Using System;" thì buộc phải viết đầy đủ cú pháp: "System.**Console**.WriteLine()", với System là một namespace.
- C# phân biệt chữ thường và chữ hoa, do đó lệnh writeline khác lệnh WriteLine.

- Bước 2: Khi soạn thảo xong. Lưu tập tin với tên: **ChaoMung.cs**
- Bước 3: Thực thi chương trình phải thực hiện hai bước:
  - Mở cửa sổ Command Line và biên dịch tập tin ChaoMung.cs vừa tạo ra sang mã máy (tạo ra file có đuôi .exe) như hình 1.1: Để biên dịch dùng trình biên dịch dòng lệnh C# (csc.exe) chương trình này được chép vào máy

trong quá trình cài .NET Framework. Nếu cài Framwork 4.0 thì tiến hành vào đường dẫn:

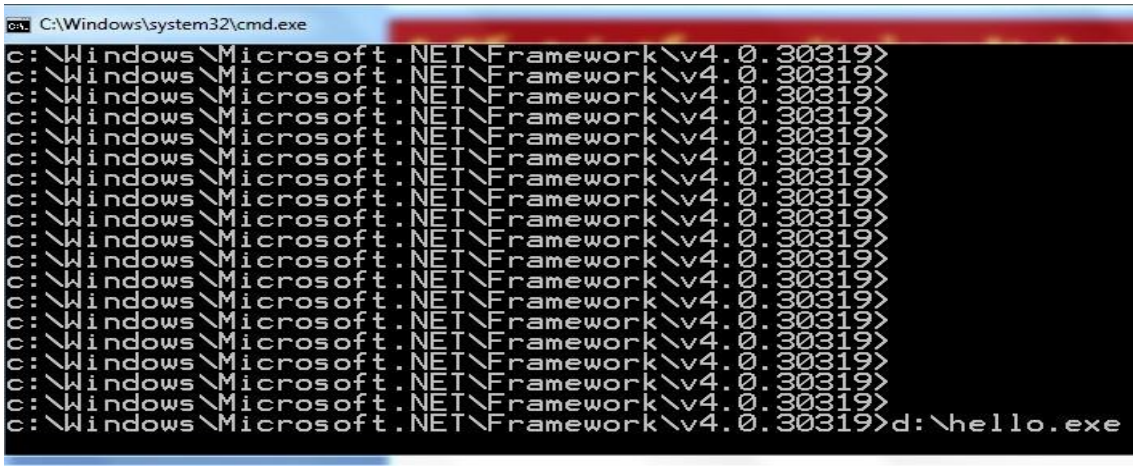
C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe

**Cú pháp**:
csc.exe [/out: <file thực thi>] <file nguồn>




*Hình 1.1: Biên dịch tập tin hello.cs*

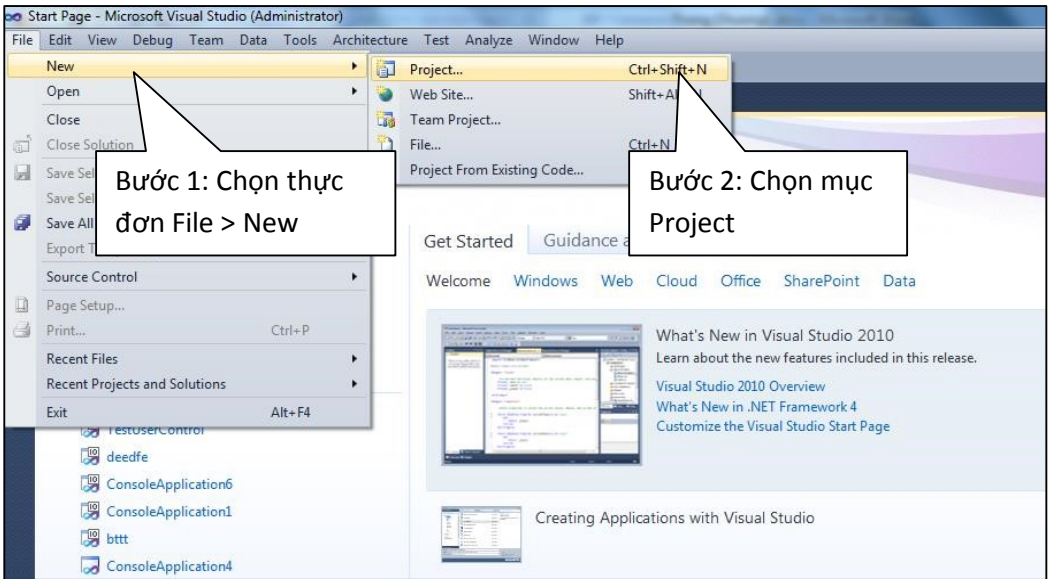
- Thực thi tập tin có đuôi .exe như hình 1.2:




*Hình 1.2: Thực thi tập tin Hello.exe*

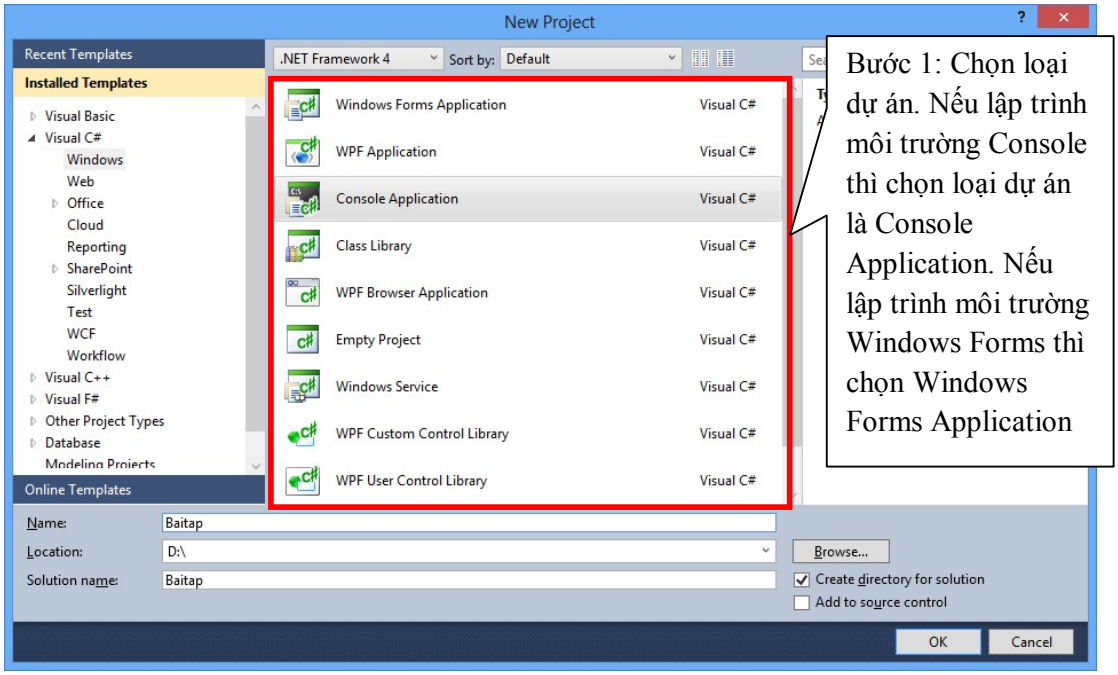
### 1.3.2. Soạn thảo bằng Microsoft Visual Studio 2010

- Bước 1: Mở Microsoft Visual Studio 2010 và tạo 1 dự án mới như hình 1.3



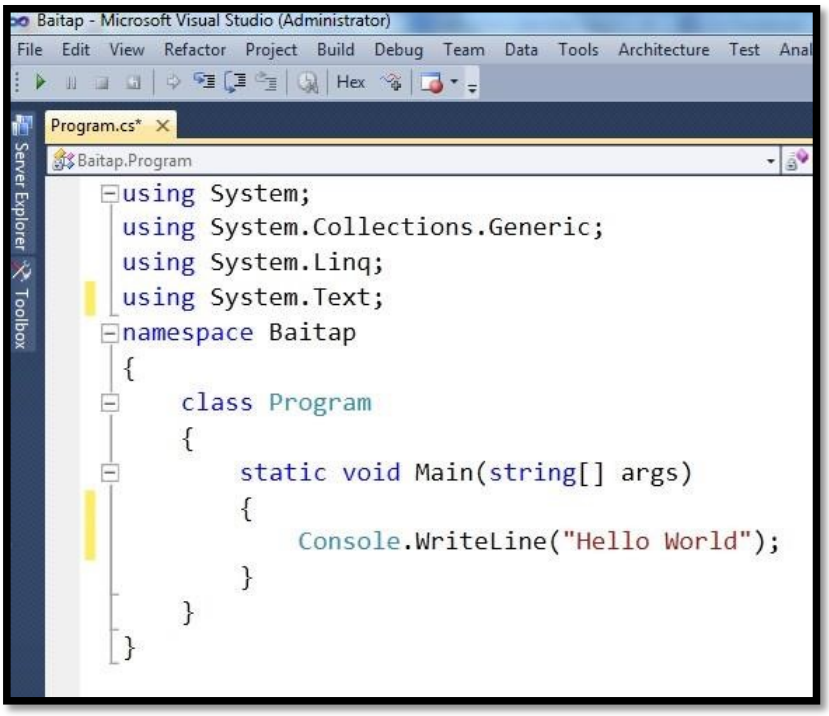

*Hình 1.3: Tạo dự án mới*

- Bước 2: Tạo một loại dự án đơn giản là Console Application để minh họa bằng ngôn ngữ C# và đặt tên Baitap lưu ở ổ đĩa D như hình 1.4



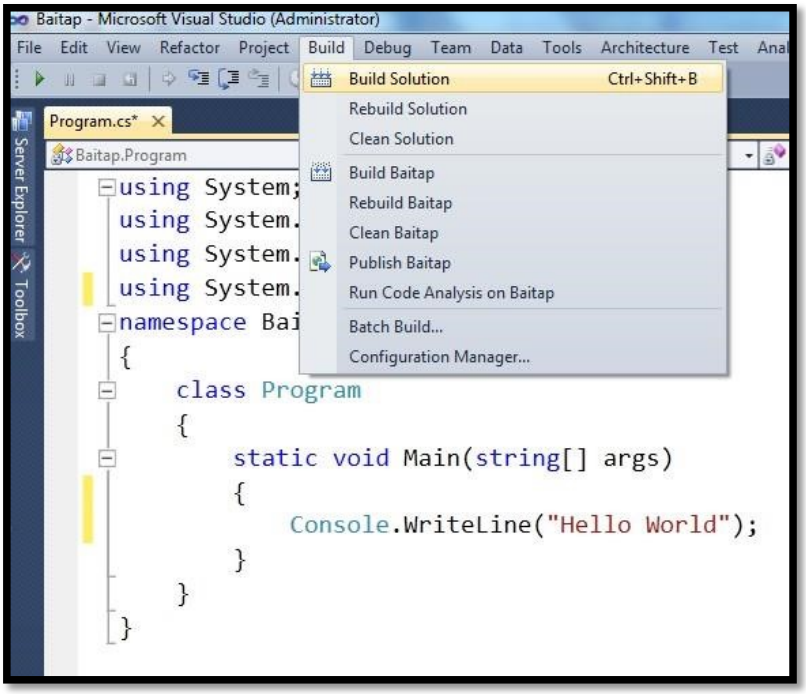

*Hình 1.4: Tạo loại dự án và đặt tên cho dự án*

- Bước 3: Soạn thảo mã C# trong trình biên dịch như hình 1.5



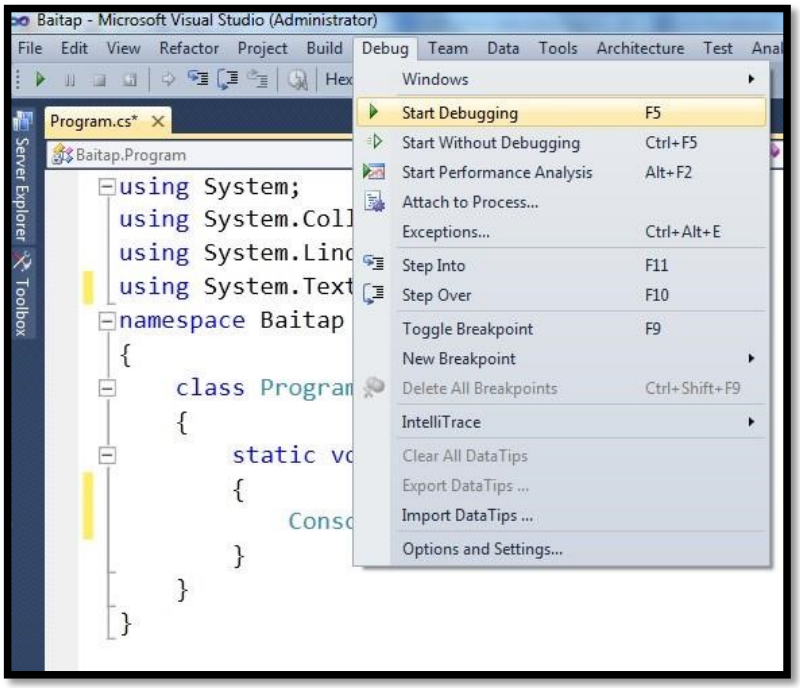

*Hình 1.5: Giao diện soạn thảo mã lệnh*

- Bước 4: Biên dịch chương trình: trên thanh menu chọn Build > Build Solution như hình 1.6 hoặc ấn tổ hợp phím Ctrl + Shift + B.




*Hình 1.6: Biên dịch chương trình*

- Bước 5: Thực thi chương trình: trên thanh menu chọn Debug > Start Debugging như hình 1.7 hoặc phím F5.




*Hình 1.7: Thực thi chương trình*

## 1.4. Biến và Kiểu dữ liệu

### 1.4.1. Biến

Biến là nơi lưu trữ dữ liệu của chương trình. Dữ liệu của biến nằm trong bộ nhớ vật lý của Ram và có thể thay đổi được. Muốn sử dụng biến trước tiên lập trình phải chỉ định biến có một kiểu dữ liệu cụ thể nào đó.

- Cú pháp khai báo biến:

| <Kiểu dữ liệu> <Tên biến> |
|---|

Lưu ý: Tên biến phải được đặt phù hợp với quy tắc đặt tên

- Cú pháp khởi tạo giá trị cho biến:

| <Tên biến> = <giá trị> |
|---|

Trong đó:

= là phép toán gán giá trị

**Quy tắc đặt tên:** Các tên biến, tên hằng, tên hàm, … trong C# đề phải đặt tên đúng với quy tắc sau:

- Một tên biến có thể bắt đầu bằng chữ hoa hay chữ thường. Tên có thể chứa ký tự hay số và ký tự gạch dưới (_).
- Ký tự đầu tiên của biến phải là ký tự , không được là số.
- Trong C# phân biệt hoa thường.

- Các từ khóa không thể sử dụng để đặt tên cho biến. Nếu muốn dùng từ khóa đặt tên thì ta thêm ký tự @ phía trước.

Ví dụ 1.2:

| | |
|---|---|
| Employee: | Hợp lệ |
| student: | Hợp lệ |
| _Name: | Hợp lệ |
| Emp_Name: | Hợp lệ |
| @goto: | Hợp lệ |
| static: | Không hợp lệ, trùng từ khóa |
| 4myclass: | Không hợp lệ, không thể bắt đầu bằng ký tự số |
| Student&Falculty: | Không hợp lệ, không được chứa ký tự đặc biệt |

### 1.4.2. Kiểu dữ liệu

Cũng như ngôn ngữ lập trình C++ hay Java, C# chia thành hai tập kiểu dữ liệu chính:

- Kiểu giá trị
- Kiểu tham chiếu

Mọi kiểu dữ liệu trong C# sẽ thuộc một trong hai kiểu giá trị hoặc kiểu tham chiếu. Kiểu giá trị thì lưu trong stack, còn kiểu tham chiếu lưu trong heap.

**Stack**: một vùng bộ nhớ dành lưu trữ dữ liệu với chiều dài cố định.

Ví dụ 1.3: số nguyên kiểu int luôn chiếm dụng 4 bytes.

Mỗi chương trình khi thực thi đều được cấp riêng 1 stack mà các chương trình khác không truy cập tới được

**Heap**: một vùng bộ nhớ dùng lưu trữ dữ liệu có dung lượng thay đổi.

Ví dụ 1.4: như kiểu string, khi ta tạo đối tượng thuộc lớp string, thì đối tượng sẽ được xác định bởi hai thành phần: Địa chị đối tượng lưu ở stack, còn giá trị đối tượng thì sẽ lưu ở heap.

#### 1.4.2.1. Kiểu giá trị

Kiểu giá trị thường là các kiểu do C# định nghĩa sẵn bao gồm: double, char, int, float, enum, struct, ….

Biến của kiểu giá trị lưu trữ một giá trị thực, giá trị này được lưu trữ trong stack, không thể mang giá trị null và phải chứa giá trị xác định.

*Bảng 1.2: Bảng mô tả các kiểu dữ liệu giá trị trong C#*

| Kiểu C# | Số Byte | Kiểu .NET | Mô tả |
|---|---|---|---|
| byte | 1 | Byte | Số nguyên dương không dấu: 0 đến 255 |
| char | 2 | Char | Ký tự Unicode |
| bool | 1 | Boolean | Giá trị logic true/ false |
| sbyte | 1 | Sbyte | Số nguyên có dấu: -128 đến 127 |
| short | 2 | Int16 | Số nguyên có dấu: -32.768 đến 32.767 |

| | | | |
|---|---|---|---|
| ushort | 2 | Uint16 | Số nguyên không dấu: 0 đến 65.535 |
| int | 4 | Int32 | Số nguyên có dấu: -2.147.483.647 đến 2.147.483.647 |
| uint | 4 | Uint32 | Số nguyên không dấu: 0 đến 4.294.967.295 |
| float | 4 | Single | Kiểu dấu chấm động, giá trị xấp xỉ từ 3,4E-38 đến 3,4E+38; với 7 chữ số có nghĩa |
| double | 8 | Double | Kiểu dấu chấm động, giá trị xấp xỉ từ 1,7E-308 đến 1,7E+308; với 15, 16 chữ số có nghĩa |
| decimal | 8 | Decimal | Có độ chính xác đến 28 con số và giá trị thập phân, được dùng trong tính toán tài chính, kiểu này đòi hỏi phải có hậu tố "m" hay "M" theo sau giá trị |
| long | 8 | Int64 | Kiểu số nguyên có dấu có giá trị trong khoảng: -9.233.370.036.854.775.808 đến -9.233.372.036.854.775.807 |
| ulong | 8 | Uint64 | Số nguyên không dấu từ 0 đến 0xffffffffffffff |

Lưu ý:

Kiểu số thực: Trình biên dịch luôn hiểu bất cứ một số thực nào cũng là một số kiểu double trừ khi khai báo rõ ràng, do đó để gán một số kiểu float thì số phải có ký tự f theo sau.

ví dụ 1.5: float a=0.15f;

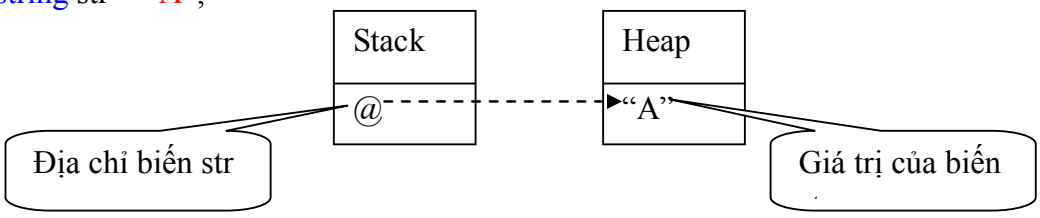
Kiểu giá trị bool trong C# nhất thiết phải là true hoặc false.

### 1.4.2.2. Kiểu tham chiếu

Biến có kiểu dữ liệu tham chiếu lưu trữ địa chỉ của biến khác trên bộ nhớ heap. Như vậy các lớp đối tượng String, Object, … đều được hiểu như kiểu dữ liệu tham chiếu. Ngoài ra các kiểu dữ liệu do người dùng xây dựng nên cũng được gọi là kiểu tham chiếu.

Ví dụ 1.6:

```
string str = "A";
```



#### 1.4.2.3. Bảng so sánh sự khác biệt giữa kiểu giá trị và kiểu tham chiếu

*Bảng 1.3: Bảng so sánh sự khác biệt giữa kiểu giá trị và kiểu tham chiếu*

| | Kiểu giá trị | Kiểu tham chiếu |
|---|---|---|
| Biến lưu trữ | Giá trị của biến | Địa chỉ của biến |
| Vị trí lưu trữ giá trị | Stack | Heap |
| Giá trị mặc định | 0 | Null |
| Phép toán gán | Sao chép giá trị | Sao chép địa chỉ |

#### 1.4.2.4. Chuyển kiểu dữ liệu

Việc chuyển đổi kiểu dữ liệu trong C# có thể thực hiện bằng một trong các cách sau:

- Sử dụng lớp Convert:

Đây là một lớp được C# xây dựng sẵn phục vụ cho việc chuyển đổi từ kiểu dữ liệu này sang kiểu dữ liệu khác. Các phương thức trong lớp Convert phần lớn đều là phương thức tĩnh:

- ToInt32(<Tham số>)
- ToInt64(<Tham số>)
- ToString(<Tham số>)
- ToDouble(<Tham số>)
- ToBoolean(<Tham số>)
- ToByte(<Tham số>)
- …

<Tham số>: có thể là chuỗi ký tự, hằng số, biến số nguyên, số thực hoặc kiểu bool. Nếu trong quá trình chuyển kiểu đặt tham số là null thì hàm Convert sẽ trả về giá trị mặc định.

Ví dụ 1.7:

```
int a = Convert.ToInt32("10");    //chuyển chuỗi 10 sang số nguyên
bool b = Convert.ToBoolean(27); //chuyển số 27 sang kiểu boolean
bool a = Convert.ToBoolean("hello"); //Sai định dạng, không chuyển được
int b = Convert.ToInt32("123456787654"); //Tràn bộ nhớ, không chuyển được
double d = Convert.ToDouble(null); //Trả về giá trị mặc định
```

- Sử dụng phương thức Parse:

Phương thức Parse thường được sử dụng để chuyển đổi chuỗi sang một kiểu dữ liệu cụ thể nào đó. Mỗi kiểu dữ liệu xây dựng sẵn trong C# đều có hỗ trợ phương thức Parse để chuyển đổi chuỗi sang kiểu dữ liệu tương ứng:

- Double.Parse(<chuoi>): Chuyển chuỗi về kiểu Double
- Int32.Parse(<chuoi>): Chuyển chuỗi về kiểu Int32
- Int64.Parse(<chuoi>): Chuyển chuỗi về kiểu Int64
- Boolean.Parse(<chuoi>): Chuyển chuỗi về kiểu Boolean
- Single.Parse(<chuoi>): Chuyển chuỗi về kiểu Single
- …

Ví dụ 1.8:

```
int a = Int32.Parse("123"); //a sẽ mang giá trị 123
float b = Float.Parse("20.7"); //b sẽ mang giá trị 20.7
bool c = Boolean.Parse("true"); //c sẽ mang giá trị true
int a = Int32.Parse(“Hello”); //sai định dạng, không chuyển được
byte b = Byte.Parse(“10000000000”); //quá giới hạn, không chuyển được
bool c = Boolean.Parse(null); //tham số là null, không chuyển được
```

➢ Sử dụng phương thức TryParse:

Phương thức TryParse cũng được sử dụng để chuyển đổi chuỗi sang một kiểu dữ liệu cụ thể như Parse. Các kiểu dữ liệu xây dựng sẵn trong C# đều có phương thức TryParse. Tuy nhiên, sự khác biệt của TryParse và Parse là:

- Cú pháp sử dụng có sự khác biệt: Có hai tham số truyền vào TryParse; Tham số thứ nhất là chuỗi cần chuyển đổi và tham số thứ hai là biến chứa giá trị chuyển đổi (biến truyền vào phải truyền ở dạng tham chiếu)
- Giá trị trả về của TryParse sẽ là true nếu chuyển kiểu thành công và trả về false nếu chuyển kiểu không thành công.

Ví dụ 1.9:

```
int a;
Int32.TryParse(“123”, out a); //a sẽ mang giá trị 123
bool b;
Boolean.TryParse(“false”, out b); //b sẽ mang giá trị false
int a;
Int32.TryParse("hello", out a); //trả về giá trị false, a mang giá trị 0
bool b;
Boolean.TryParse("", out b); //trả về giá trị false, b mang giá trị false
```

➢ Casting (ép kiểu):

Đây là cách chuyển kiểu được sử dụng khi muốn chuyển đổi giữa những kiểu dữ liệu gần tương tự nhau. Thường áp dụng với kiểu số.

Ví dụ 1.10:

```
int x= 10;
float y = x;          //chuyển đổi ngầm định, y = 10
int z = (int)y;       //chuyển đổi rõ ràng, z = 10
int x = 10;
```

Trong quá trình chuyển kiểu có thể xảy ra lỗi do chuyển giữa các kiểu dữ liệu không tương thích: Kiểu chuỗi sang kiểu số, kiểu bool sang kiểu float, …

## 1.5. Câu lệnh phân nhánh

Việc phân nhánh có thể được tạo ra bằng các từ khóa: if, else, switch, case. Sự phân nhánh chỉ được thực hiện khi biểu thức điều kiện phân nhánh được xác định là đúng.

### 1.5.1. Câu lệnh if

- Cú pháp:

```
if (biểu thức điều kiện)
        <Câu lệnh thực hiện khi điều kiện đúng>
[else
        <Câu lệnh thực hiện khi điều kiện sai> ]
```

Nếu các câu lệnh trong thân của if hay else mà lớn hơn một lệnh thì các lệnh này phải được bao trong một khối lệnh, tức là phải nằm trong dấu khối { }:

- Cú pháp:

```
if (biểu thức điều kiện)
{
        <lệnh 1>; < lệnh 2>;….
}
[else
{
        <lệnh 3>; < lệnh 3>;….
}]
```

### 1.5.2. Câu lệnh switch

Khi có quá nhiều điều kiện để chọn thực hiện thì dùng câu lệnh if sẽ rất rối rắm và dài dòng, Các ngôn ngữ lập trình cấp cao đều cung cấp một dạng câu lệnh switch liệt kê các giá trị và chỉ thực hiện các giá trị thích hợp.

➢ Cú pháp:

```
switch (biểu thức điều kiện)
{
        case <giá trị>:
                <Các câu lệnh thực hiện>
                <lệnh nhảy>
        [default:
                <Các câu lệnh thực hiện mặc định>]
}
```

Lưu ý:

- <lệnh nhảy>: Là lệnh chỉ định việc kết thúc một case, lệnh nhảy trong switch là break hoặc goto; Trường hợp trong case không có lệnh nhảy thì trình biên dịch sẽ tuần tự thực hiện các câu lênh trong từng case của switch theo thứ tự từ trên xuống.
- Trong trường hợp switch sử dụng nhiều case, có thể nhảy trực tiếp đến case cụ thể nào đó bằng goto; hoặc thoát khỏi switch sử dụng break.
- <giá trị>: là hằng nguyên hoặc biến; Nếu là biến bắt buộc phải là biến kiểu số nguyên.

## 1.6. Câu lệnh lặp

C# kế thừa cú pháp câu lệnh của C/C++:

- ➢ Lệnh lặp while
- ➢ Lệnh lặp do/ while
- ➢ Lệnh lặp for

Ngoài ra 3 lệnh lặp cơ bản trên, C# còn có thêm lệnh lặp foreach dùng để làm việc với mảng, tập hợp.

C# cũng cung cấp các lệnh nhảy như: break, goto, continue và return sử dụng kết hợp với lệnh lặp

### 1.6.1. Lệnh lặp while

Ý nghĩa của vòng lặp while là: "*Trong khi điều kiện đúng thì thực hiện các công việc này*".

➢ Cú pháp:

```
while (Biểu thức)
{
    <Câu lệnh thực hiện>
}
```

Biểu thức của vòng lặp **while** là điều kiện để các lệnh được thực hiện, biểu thức này bắt buộc phải trả về một giá trị kiểu bool là true/false

Ví dụ 1.11: Viết chương trình tính tổng các số lẻ >0 và nhỏ hơn 10

```
static void Main(string[] args)
{
        int i = 1, tong = 0;
         while (i < 10)
         {
                tong = tong + i; i = i+2;
         }
 }
```

### 1.6.2. Lệnh lặp do/ while

Ý nghĩa của vòng lặp **do/ while** là: "*làm điều này trong khi điều kiện vẫn còn đúng*".

- Cú pháp

```
do
 {
    <Câu lệnh thực hiện>
 }while ( điều kiện );
```

Sự khác biệt tối quan trọng của **while** và **do..while** là tối thiểu sẽ có một lần các câu lệnh trong **do**...**while** được thực hiện

Ví dụ 1.12: Viết chương trình tính tổng các số lẻ >0 và nhỏ hơn 10

```
static void Main(string[] args)
{
        int i = 1, tong = 0;
         do
         {
                tong = tong + i;
                i = i+2;
         }while(i<10);
```

### 1.6.3. Lệnh lặp for

Vòng lặp for bao gồm ba phần chính:

- Khởi tạo biến đếm vòng lặp
- Kiểm tra điều kiện biến đếm, nếu đúng thì sẽ thực hiện các lệnh bên trong vòng for

- Thay đổi bước lặp.
- Cú pháp:

```
for ( [phần khởi tạo] ; [biểu thức điều kiện]; [bước lặp])
{
        <Câu lệnh thực hiện>
}
```

Ví dụ 1.13: Viết chương trình tính tổng các số lẻ >0 và nhỏ hơn 10

```
static void Main(string[] args)
{
        int i, tong = 0;
        for(i=1; i < 10; i=i+2)
                tong = tong + i;
}
```

### 1.6.4. Lệnh lặp foreach

Vòng lặp foreach cho phép tạo vòng lặp thông qua một tập hợp hay một mảng.

- Cú pháp:

```
foreach ( <kiểu tập hợp> <tên truy cập thành phần > in < tên tập hợp>)
{
        <Các câu lệnh thực hiện>
}
```

Do lặp dựa trên một mảng hay tập hợp nên toàn bộ vòng lặp sẽ duyệt qua tất cả các thành phần của tập hợp theo thứ tự được sắp. Khi duyệt đến phần tử cuối cùng trong tập hợp thì chương trình sẽ thoát ra khỏi vòng lặp foreach

Ví dụ 1.14: Tính tổng các số lẻ trong mảng A = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

```
static void Main(string[] args)
{
   int[] intArray = {0,1,2,3,4,5,6,7,8,9,10}, tong = 0;
   foreach( int item in intArray)
   {
      if (item % 2 != 0)
           tong = tong + item;
   }
}
```

## 1.7. Lớp String

### 1.7.1. Giới thiệu về chuỗi ký tự

C# xem những chuỗi như là những kiểu dữ liệu cơ bản , việc sử dụng chuỗi trong C# rất linh hoạt, mạnh mẽ, và quan trọng nhất là dễ sử dụng. Chỉ mục trong chuỗi được tính từ số 0.

Để làm việc với chuỗi, C# cung cấp lớp System.String. Khi khai báo một chuỗi C# bằng cách dùng từ khóa string, đồng nghĩa với việc người dùng đã khai báo một đối tượng của lớp System.String.

Ví dụ 1.15:

```
string   chuoi="hello";
```

Ngoài việc, ta cũng có thể khởi tạo chuỗi từ 1 mảng char:

Ví dụ 1.16:

```
// Khởi tạo mảng ký tự
        char[]  chars = {'c', 's', 'h', 'a', 'r', 'p'};
// Khởi tạo chuỗi từ mảng ký tự
        string   str = new string(chars);
```

Lưu ý:

- Khi gán giá trị trong chuỗi có thể chứa ký tự escape: '\n', '\t', '\a', …

  Ví dụ 1.17:

  ```
  string   newString = "Day la chuoi \n trich dan";
  ```

  Ký tự xuống

- Trong trường hợp muốn lưu đường dẫn thư mục vào một chuỗi:

  Ví dụ 1.18:

  ```
  string   path = " c:\baitap";
  ```

  Trình biên dịch báo lỗi, vì không hiểu ký tự escape '\b'

- Do đó muốn trình biên dịch hiểu '\b' như những ký tự thông thường thì người dùng phải thêm một ký tự '\' phía trước:

  Ví dụ 1.19:

  ```
  string   path = " c:\\baitap";
  ```

- Ngoài việc sử dụng dấu \ để giữ nguyên các ký tự trong chuỗi ta cũng có thể khai báo chuỗi như là 1 chuỗi nguyên văn bằng cách thêm @ phía trước chuỗi.

  Ví dụ 1.20:

  ```
  string    path = @"D:\soft\VMWare"
  ```

### 1.7.2. Phương thức và thuộc tính lớp String

*Bảng 1.4: Bảng tổng hợp các phương thức lớp String*

| System.String | |
|---|---|
| Phương thức/ Thuộc tính | Ý nghĩa |
| Empty | Thuộc tính tĩnh; Thể hiện chuỗi rỗng |
| Compare() | Phương thức tĩnh; So sánh hai chuỗi |
| CompareOrdinal() | Phương thức tĩnh; So sánh hai chuỗi không quan tâm đến thứ tự |
| Concat() | Phương thức tĩnh; Tạo ra chuỗi mới từ một hay nhiều chuỗi khác |
| Copy() | Phương thức tĩnh; Tạo ra một chuỗi mới bằng cách sao chép từ chuỗi khác |
| Equal() | Phương thức tĩnh; Kiểm tra xem hai chuỗi có cùng giá trị hay không |
| Insert() | Trả về chuỗi mới đã được chèn một chuỗi xác định |
| LastIndexOf() | Chỉ ra vị trí xuất hiện cuối cùng của một chuỗi xác định trong chuỗi ban đầu |
| PadLeft() | Canh lề phải những ký tự trong chuỗi, chèn vào bên trái khoảng trắng hay các ký tự xác định |
| PadRight() | Canh lề trái những ký tự trong chuỗi, chèn vào bên phải khoảng trắng hay các ký tự xác định |
| Remove() | Xóa đi một số ký tự xác định |
| Split() | Trả về chuỗi được phân định bởi những ký tự xác định trong chuỗi |
| StartWith() | Xem chuỗi có bắt đầu bằng một số ký tự xác định hay không |
| SubString() | Lấy một chuỗi con |
| ToCharArray() | Sao chép những ký tự từ một chuỗi đến mảng ký tự |
| ToLower() | Trả về bản sao của chuỗi ở kiểu chữ thường |
| ToUpper() | Trả về bản sao của chuỗi ở kiểu chữ hoa |
| Format() | Phương thức tĩnh; Định dạng một chuỗi dùng ký tự lệnh định dạng xác định |
| Join() | Phương thức tĩnh; Kết nối các chuỗi xác định giữa mỗi thành phần của mảng chuỗi |
| Length | Trả về chiều dài của chuỗi |
| CompareTo() | So sánh hai chuỗi |
| CopyTo() | Sao chép một số các ký tự xác định đến một mảng ký tự Unicode |
| EndsWith() | Chỉ ra vị trí của chuỗi xác định phù hợp với chuỗi đưa ra |
| Trim() | Xóa bỏ tất cả sự xuất hiện của khoảng trắng trong chuỗi |
| TrimEnd() | Xóa các khoảng trắng ở vị trí cuối |
| TrimStart() | Xóa các khoảng trắng ở vị trí đầu |

➢ **Thuộc tính Empty:** Đại diện cho 1 chuỗi rỗng. Sử dụng khi muốn khai báo 1 chuỗi là rỗng.

Ví dụ 1.21:

```
string chuoi =  string.Empty;
```

Khi ta khai báo như trên nghĩa là tương đương với khai báo:

```
string chuoi = "";
```

➢ **Thuộc tính Length:** Dùng để xác định chiều dài của chuỗi.

Ví dụ 1.22:

```
static void Main(string[] args)
{
        string str = "pham phuong nguyen";
        int chieudai = str.Length;
}
//Kết quả: chieudai = 18
```

➢ **Nhóm các phương thức so sánh chuỗi:** Compare, CompareOrdinal, CompareTo, Equal

- Phương thức Compare: Phương thức tĩnh so sánh hai chuỗi

  Cách sử dụng:

```
int  gt = String.Compare(<chuoi1>, <chuoi2>, [true]);
```

  - Nếu gt = 0: hai chuỗi bằng nhau
  - Nếu gt = 1: chuỗi 1 lớn hơn chuỗi 2
  - Nếu gt = -1: chuỗi 1 nhỏ hơn chuỗi 2
  - Nếu sử dụng phương thức không có tham số true, thì sẽ phân biệt chữ thường, chữ hoa. Nếu có tham số true thì sẽ không biệt chữ thường hay chữ hoa.

  Ví dụ 1.23:

```
public static void Main(string []args)
{
  int gt =  String.Compare("abDb", "abdb");  //gt = 1
  int gt1= String.Compare("abDb", "abdb", true); //gt1 = 0
  int gt2= String.Compare("abdaF", "abdb");  //gt2 = -1
}
```

- Phương thức so sánh chuỗi CompareOrdinal: So sánh chuỗi trả về một giá trị kiểu int; Lần lượt tìm từ đầu chuỗi đến cuối chuỗi nếu thấy có ký tự khác biệt giữa hai chuỗi. Tính hiệu mã ASCII của hai ký tự này và trả kết quả vừa tính được.

Cách sử dụng:

```
int  gt = String.CompareOrdinal(<chuoi1>,<chuoi2>);
```

Ví dụ 1.24:

```
public static void Main(string []args)
{
        int gt1 = String.CompareOrdinal("abcd", "aBcd");       //gt1=32
        int gt2 = String.CompareOrdinal("abcd", "aBcdefg"); //gt2=32
        int gt3 = String.CompareOrdinal("Abcd", "aBcdefg"); //gt3=-32
        int gt4 = String.CompareOrdinal("abcd", "abcd");       //gt4=0
}
```

- Phương thức so sánh chuỗi CompareTo: Phương thức thành viên lớp, giúp so sánh 2 chuỗi và trả về 1 giá trị kiểu int.

  Cách sử dụng:

```
int  gt = <chuoi1>.CompareTo(<chuoi2>);
```

  - Nếu gt = 0: hai chuỗi bằng nhau
  - Nếu gt = 1: chuỗi 1 lớn hơn chuỗi 2
  - Nếu gt = -1: chuỗi 1 nhỏ hơn chuỗi 2

- Phương thức so sánh chuỗi Equal: Phương thức này dùng để so sánh các chuỗi ký tự có giống nhau hay không. Nếu 2 chuỗi hoặc ký tự giống nhau sẽ trả về giá trị true , ngược lại trả về giá trị false.

  Cách sử dụng:

```
bool gt = String.Equal(<str1>,<str2>, [StringComparisonType]);
```

  Lưu ý:

  Nếu sử dụng phương thức không chỉ định tham số StringComparisonType thì sẽ so sánh từng ký tự một giữa hai chuỗi và phân biện hoa thường.

  Nếu muốn không phân biệt chữ hoa thường thì thêm tham số StringComparison.OrdinalIgnoreCase

Ví dụ 1.25:

```
public static void Main(string []args)
{
    string str1 = "ITlab";
     string str2 = "itlac";
     if (String.Equals(str1, str2, StringComparison.OrdinalIgnoreCase))
       MessageBox.Show("Chuoi giong nhau ");
      else
        MessageBox.Show("Chuoi khong giong nhau ");
}
```

➢ **Nhóm phương thức nối chuỗi và chèn chuỗi:** Concat, Join, Inser

- Phương thức Concat: Phương thức tĩnh Concat dùng để nối hai hay nhiều chuỗi lại với nhau và trả về 1 chuỗi mới

  Cách sử dụng:

```
string <chuoiKQ> = String.Concat(<chuoi1>,<chuoi2>);
```

Ví dụ 1.26:

```
public static void Main(string []args)
{
        string chuoi1="abc";
        string chuoi2="aBc";
        string kq = String.Concat(chuoi1,chuoi2); //kq = "abcaBc"
}
```

- Phương thức Join: dùng để nối 1 mảng chuỗi lại với nhau, giữa các phần tử có thể chèn thêm 1 chuỗi nào đó để phân cách.

  Cách sử dụng:

```
string <chuoiKQ> = String.Join(<chuỗi phân cách>,<Mảng chuỗi>);
```

Ví dụ 1.27:

```
public static void Main(string []args)
{
    string[] chuoi = { "mot", "hai", "ba" };
    string str = String.Join("---", chuoi);
}
//str = "mot---hai---ba
```

- Phương thức Insert: Phương thức thuộc lớp, giúp chèn 1 chuỗi vào 1 vị trí xác định trong chuỗi khác.

Cách sử dụng:

```
string <chuoiKQ> = <Tên đối tượng>.Join(<vị trí chèn>,<Chuỗi chèn vào>);
```

Ví dụ 1.28:

```
public static void Main(string []args)
{
    string chuoi = “mot hai bon”;
    string str = chuoi.Insert(3,  “ hai”);  //str = “mot hai ba bon”
}
```

➢ **Nhóm phương thức kiểm tra và định vị:** Contain, StartsWith, EndsWith, IndexOf, IndexOfAny, LastIndexOf, LastIndexOfAny

- Phương thức Contain: Phương thức thuộc lớp, giúp các tìm một từ hoặc một chuỗi bất kỳ trong một chuỗi khác. Nếu tìm thấy trả về true, không tìm thấy trả về false

  Cách sử dụng:

```
bool  gt= <chuoi1>.Contain(<chuoi2>)
```

  Chuoi2: có thể là một ký tự hoặc 1chuỗi. Phương thức Contain sẽ kiểm tra chuoi2 có trong chuoi1 không, việc tìm kiếm này có sự phân biệt giữa ký tự hoa và ký tự thường.

  Ví dụ 1.29:

```
public static void Main(string []args)
{
        string str = null;
        str = "Diễn đàn công nghệ và giải pháp";
        if (str.Contains("công nghệ") == true)
                MessageBox.Show("Chuỗi trên có chứa từ 'công nghệ' ");
        else
                MessageBox.Show("Chuỗi trên không chứa từ 'công nghệ'");
}//Hiển thị Messagebox: Chuỗi trên có chứa từ ‘công nghệ’
```

- Phương thức StartsWith: Phương thức thuộc lớp, kiểm tra xem chuỗi có bắt đầu bởi 1 số ký tự nào đó không. Phương thức này có kiểu trả về là Boolean:  true/false.

```
bool  gt= <chuoi1>.StartsWith(<chuoi2>)
```

  Chuoi2: Là chuỗi cần kiểm tra xem có là chuỗi bắt đầu của chuoi1 không. Việc kiểm tra này có phân biệt ký tự hoa và ký tự thường.

Ví dụ 1.30:

```
public static void Main(string []args)
{
        string str = null;
        str = "Diễn đàn công nghệ và giải pháp";
        if (str. StartsWith ("diễn đàn") == true)
                MessageBox.Show("Chuỗi str bắt đầu với 'diễn đàn'");
        else
                MessageBox.Show("Chuỗi str không bắt đầu với 'diễn
        đàn'");
}//Hiển thị Messagebox: Chuỗi str không bắt đầu với 'diễn đàn'
```

Nếu muốn việc tìm kiếm không phân biệt ký tự hoa và ký tự thường cần thêm tham số StringComparisonType trong phương thức:

Cách sử dụng:

```
bool  gt= <chuoi1>.StartsWith(<chuoi2>, [StringComparisonType])
```

Ví dụ 1.31:

```
public static void Main(string []args)
{
   string str = null;
   str = "Diễn đàn công nghệ và giải pháp";
   if   (str.StartsWith("diễn   đàn",StringComparison.OrdinalIgnoreCase)
   == true)
        MessageBox.Show("Chuỗi str bắt đầu với 'diễn đàn'");
   else
        MessageBox.Show("Chuỗi str không bắt đầu với 'diễn đàn'");
}//Hiển thị Messagebox: Chuỗi str bắt đầu với 'diễn đàn'
```

- Phương thức EndsWith: Phương thức thuộc lớp, kiểm tra xem chuỗi có kết thúc bởi 1 số ký tự nào đó không. Phương thức này có kiểu trả về là Boolean:  true/false.

```
bool  gt= <chuoi1>.EndsWith(<chuoi2>)
```

Chuoi2: Là chuỗi cần kiểm tra xem có là chuỗi kết thúc của chuoi1 không. Việc kiểm tra này có phân biệt ký tự hoa và ký tự thường.

Ví dụ 1.32:

```
public static void Main(string []args)
{
        string str = null;
        str = "Diễn đàn công nghệ và giải pháp";
        if (str. EndsWith ("giải pháp") == true)
            MessageBox.Show("Chuỗi str kết thúc với 'giải pháp'");
        else
            MessageBox.Show("Chuỗi str không kết thúc với 'giải
        pháp'");
}//Hiển thị Messagebox: Chuỗi str kết thúc với 'giải pháp'
```

- Phương thức IndexOf: Phương thức thuộc lớp, trả về chỉ số của ký tự đầu tiên mà chuỗi xuất hiện, nếu không tìm thấy sẽ trả giá trị -1.

  Cách sử dụng:

```
int  vitri= <chuoi1>.IndexOf(<chuoi2>)
```

Chuoi2: là chuỗi cần tìm kiếm trong chuoi1.

Ví dụ 1.33:

```
public static void Main(string []args)
{
        string str1 = "Chao ngay ngay moi";
        string str2 = "ngay";
        int vitri = str1.IndexOf(str2));
}
//vitri = 5
```

- Phương thức LastIndexOf: Phương thức thuộc lớp, tìm kiếm bắt đầu từ cuối chuỗi về đầu, trả về chỉ số của ký tự đầu tiên mà chuỗi xuất hiện. Nếu không tìm thấy sẽ trả giá trị -1.

  Cách sử dụng:

```
int  vitri= <chuoi1>.LastIndexOf(<chuoi2>)
```

Chuoi2: là chuỗi cần tìm kiếm trong chuoi1.

Ví dụ 1.34:

```
public static void Main(string []args)
{
        string str1 = "Chao ngay ngay moi";
        string str2 = "ngay";
        int vitri = str1.LastIndexOf(str2));
}
//vitri = 10
```

- Phương thức IndexOfAny: Phương thức thuộc lớp, tìm kiếm chỉ số xuất hiện của từng ký tự trong chuỗi so với mảng ký tự đầu vào, nếu không tìm thấy sẽ trả giá trị -1.

  Cách sử dụng:

```
int  vitri= <chuoi1>.LastIndexOf(<mảng các ký tự>)
```

  Từng ký tự trong chuoi1, bắt đầu từ ký tự đầu tiên của chuoi1 sẽ lần lượt đươc so sánh với các ký tự trong mảng ký tự  đầu vào, nếu tìm thấy một ký tự nào đó trong chuoi1 xuất hiện trong mảng ký tự sẽ trả về vị trí của ký tự đó trong chuoi1.

  Ví dụ 1.35:

```
public static void Main(string []args)
{
        string str1 = "Chao ngay ngay ngay moi";
        char[] mangkytu = {'n','g','a','y'};
        int vitri = str1.IndexOfAny(mangkytu);   //vitri = 2
}
```

- Phương thức LastIndexOfAny: Phương thức thuộc lớp, tìm kiếm chỉ số xuất hiện của từng ký tự trong chuỗi so với mảng ký tự đầu vào, nếu không tìm thấy sẽ trả giá trị -1.

  Cách sử dụng:

```
int  vitri= <chuoi1>.LastIndexOf(<mảng các ký tự>)
```

  Từng ký tự trong chuoi1, bắt đầu từ ký tự cuối chuoi1 sẽ lần lượt đươc so sánh với các ký tự trong mảng ký tự đầu vào, nếu tìm thấy một ký tự nào đó trong chuoi1 xuất hiện trong mảng ký tự sẽ trả về vị trí của ký tự đó trong chuoi1.

  Ví dụ 1.36:

```
public static void Main(string []args)
{
        string str1 = "Chao ngay ngay ngay moi";
        char[] mangkytu = {'n','g','a','y'};
        int vitri = str1.LastIndexOfAny(mangkytu);   //vitri = 18
}
```

➢ **Nhóm phương thức trích chọn xâu con từ chuỗi: Substring, Split.**

- Phương thức Substring: Trích một chuỗi con từ một chuỗi có sẵn, trả về một chuỗi mới.

  Cách sử dụng:

```
string kq = <chuoi1>.Substring(start, len);
```

start: Vị trí ký tự bắt đầu trích trong chuoi1

len: Số ký tự sẽ trích ra trong chuoi1

Ví dụ 1.37:

```
public static void Main(string []args)
{
        string str1 = "Chao  ngay  moi";
        string str=str1.Substring(5, 4);
        //str = “ngay”
}
```

Ngoài ra có thể trích chọn chuỗi con từ vị trí chỉ định đến cuối chuỗi bằng cách bỏ đi tham số len trong phương thức Substring.

Cách sử dụng:

```
string kq = <chuoi1>.Substring(start);
```

start: Vị trí ký tự bắt đầu trích trong chuoi1

Ví dụ 1.38:

```
public static void Main(string []args)
{
        string str1 = "Chao  ngay  moi";
        string str=str1.Substring(5);
        //str = “ngay moi”
}
```

- Phương thức Split: Phương thức thành viên của lớp. Sử dụng tách chuỗi lớn thành các chuỗi con.

  Cách sử dụng:

```
string []kq = <chuoi1>.Substring(<danh sách ký tự>);
```

  <danh sách ký tự>: Là một ký tự hoặc một mảng các ký tự sử dụng để làm tiêu chí tách chuoi1 thành các chuỗi con.

  Ví dụ 1.39:

```
public static void Main(string []args)
{
        string str1 = "Son,Tung,Tuan";
        string[] ten = str1.Split(',');
}
//Kết quả được mảng tên gồm các phần tử: {“Son”, “Tung”, “Tuan”}
```

- Phương thức Remove:  Phương thức thuộc lớp, sử dụng để xóa chuỗi

  Cách sử dụng:

```
string kq = <chuoi1>.Remove(<Vị trí>, [<Số ký tự>]);
```

<Vị trí>: Là vị trí của ký tự trong chuoi1 bắt đầu xóa.

<Số ký tự>: Là số lượng ký tự sẽ xóa trong chuoi1

Ví dụ 1.40:

```
public static void Main(string []args)
{
        string str = “mothaibabonnam";
        string str1 = str.Remove(3,8);
}
//str1 = “motnam”
```

Ngoài ra, Remove còn có thể sử dụng để xóa từ một vị trí chỉ đính đến cuối chuỗi.

Cách sử dụng:

```
string kq = <chuoi1>.Remove(<Vị trí>);
```

<Vị trí>: Vị trí ký tự bắt đầu xóa trong chuoi1.

Ví dụ 1.41:

```
public static void Main(string []args)
{
        string str = “mothaibabonnam";
        string str1 = str.Remove(8);
}
//str1 = “motnamba”
```

- **Nhóm phương thức sao chép chuỗi**: Copy, ToCharArray, CopyTo
  - Phương thức Copy: Phương thức tĩnh giúp trả về 1 chuỗi nằm trên 1 vùng nhớ khác nhưng cùng nội dung với chuỗi ban đầu.

    Cách sử dụng:

```
string kq = String.Copy(<chuoi>);
```

    Ví dụ 1.42:

```
public static void Main(string []args)
{
        string str1 = "Chao ngay moi";
        string str = String.Copy( str1);
}
//Tạo ra vùng nhớ mới của str1 đồng thời sao chép nội dung chuỗi str
//vào str1. Kết quả str = “Chao ngay moi”
```

- Phương thức ToCharArray: Phương thức thuộc lớp, chuyển 1 chuỗi về 1 mảng ký tự và trả giá trị về là mảng ký tự đó.

  Cách sử dụng:

```
char[] <tên mảng ký tự> = <chuoi>.ToCharArray();
```

  <chuoi>: Là chuỗi ký tự cần chuyển đổi về mảng các ký tự.

  <Tên mảng ký tự>: chứa các ký tự chuyển từ <chuoi>

  Ví dụ 1.43:

```
public static void Main(string []args)
{
        string str1 = "Chao";
        char []chars =str1.ToCharArray();
}
//mảng ký tự chars = {'C', 'h', 'a', 'o'}
```

- Phương thức CopyTo: Phương thức thuộc lớp, giúp sao chép các ký tự trong một chuỗi vào một mảng ký tự, có thể quy định được vị trí và số lượng ký tự trong chuỗi sẽ copy vào mảng ký tự, đồng thời chỉ định copy vào vị trí thứ mấy trong mảng ký tự.

  Cách sử dụng:

```
void CopyTo(int sourceIndex, char[] destination, int destinationindex,
                                                        int count)
```

  Ví dụ 1.44:

```
public static void Main(string []args)
{
        string str= "chao mung mot ngay moi";
        char  []chars   = new char[10];
        str.CopyTo( 5,  chars,  0,  8);
}
//mảng ký tự chars = {'m', 'u', 'n', 'g', ' ', 'm', 'o', 't'}
```

➢ **Nhóm các phương thức định dạng chuỗi**: Padleft, PadRight, ToLower, ToUpper, Trim, TrimStart, TrimEnd.

- Phương thức PadLeft: Phương thức thuộc lớp, canh lề phải đồng thời thêm vào bên trái chuỗi một số ký tự đặc biệt.

  Cách sử dụng:

```
string <chuoi> = <chuoi> . PadLeft(len, [ký tự])
```

  [ký tự]: Ký tự sẽ thêm, nếu trong hàm PadLeft không có tham số [ký tự] thì mặc định sẽ thêm khoảng trắng vào.

Ví dụ 1.45:

```
public static void Main(string []args)
{
        string str1 = "Chao";
         str1= str1.PadLeft(7);        //str1 = “   Chao”
         string str2=“hello”;
         str2=str2.PadLeft(8, ‘*’); //str2 = “***hello”
}
```

- Phương thức PadRight: Phương thức thuộc lớp, canh lề trái đồng thời thêm vào bên phải chuỗi một số ký tự đặc biệt.

  Cách sử dụng:

```
string <chuoi> = <chuoi> . PadRight(len, [ký tự])
```

  [ký tự]: Ký tự sẽ thêm, nếu trong hàm PadRight không có tham số [ký tự] thì mặc định sẽ thêm khoảng trắng vào.

  Ví dụ 1.46:

```
public static void Main(string []args)
{
        string str1 = "Chao";
         str1= str1.PadRight(7);       //str1 = “Chao   ”
         string str2=“hello”;
         str2=str2.PadRight(8, ‘*’); //str2 = “hello***”
}
```

- Phương thức ToUpper: Phương thức thuộc lớp, trả về một chuỗi in hoa

  Cách sử dụng:

```
string <chuoi1> = <chuoi2> . ToUpper()
```

  Ví dụ 1.47:

```
public static void Main(string []args)
{
        string str1 = "Chao";
        string str2= str1.ToUpper();  //str2 = “CHAO”
}
```

- Phương thức ToLower: Phương thức thuộc lớp, trả về một chuỗi in thường

  Cách sử dụng:

```
string <chuoi1> = <chuoi2> . ToLower()
```

Ví dụ 1.48:

```
public static void Main(string []args)
{
        string str1 = "Chao";
        string str2= str1.ToLower();  //str2 = “chao”
}
```

- Phương thức Trim: Phương thức thuộc lớp, trả về 1 chuỗi đã bỏ khoảng trắng đầu chuỗi và cuối chuỗi.

  Cách sử dụng:

```
string <chuoi1> = <chuoi2> . Trim()
```

Ví dụ 1.49:

```
public static void Main(string []args)
{
        string str1 = "     Chao ngay moi     ";
        string str2= str1.Trim();  //str2 = “Chao ngay moi”
}
```

- Phương thức TrimStart: Phương thức thuộc lớp, trả về 1 chuỗi đã bỏ khoảng trắng đầu chuỗi.

  Cách sử dụng:

```
string <chuoi1> = <chuoi2> . TrimStart()
```

Ví dụ 1.50:

```
public static void Main(string []args)
{
        string str1 = "     Chao ngay moi     ";
        string str2= str1.TrimStart();  //str2 = “Chao ngay moi     ”
}
```

- Phương thức TrimEnd: Phương thức thuộc lớp, trả về 1 chuỗi đã bỏ khoảng trắng cuối chuỗi.

  Cách sử dụng:

```
string <chuoi1> = <chuoi2> . TrimEnd()
```

Ví dụ 1.51:

```
public static void Main(string []args)
{
        string str1 = "     Chao ngay moi     ";
        string str2= str1.TrimEnd();  //str2 = “     Chao ngay moi”
}
```

## 1.8. Mảng

Mảng là một tập hợp có thứ tự của những đối tượng, các đối tượng này cùng một kiểu. Cú pháp khai báo mảng được kết hợp giữa cú pháp khai báo mảng của C và định nghĩa lớp của C#, do đó các đối tượng của mảng có thể truy cập những phương thức và thuộc tính của lớp System.Array.

### 1.8.1. Mảng một chiều

➢ Cú pháp:

```
<kiểu dữ liệu> [] <tên mảng> = new <kiểu dữ liệu>[<số pt>];
```

Ví dụ 1.52: Khai báo mảng số nguyên có 100 phần tử

```
int [] mang = new int[100];
```

Hoặc có thể khai báo mảng đồng thời khởi tạo giá trị cho các phần tử mang:

➢ Cú pháp:

```
<kiểu dữ liệu> [] <tên mảng> = {<gt1>, <gt2>, <gt3>, …., <gtn>};
```

Lưu ý: khi khởi tạo mảng bằng toán tử new thì các phần tử của mảng sẽ mang giá trị mặc định như bảng 1.5 sau:

*Bảng 1.5: Bảng mô tả các giá trị mặc định của kiểu dữ liệu*

| Kiểu dữ liệu | Giá trị mặc định |
|---|---|
| int, long, byte, … | 0 |
| bool | false |
| char | '\0' (null) |
| enum | 0 |
| reference | null |

➢ Truy cập các phần tử của mảng:

Để truy cập các thành phần trong mảng phải sử dụng toán tử chỉ mục []. Với phần tử đầu tiên trong mảng mang chỉ số số 0.

Ngoai ra C# hỗ trợ thuộc tính Length để xác định chiều dài của mang.

Ví dụ 1.53: Tính tổng các phần tử của mảng A = {1, 2, 3, 4, 5}

```
public static void Main(string []args)
{       int[] mang={1,2,3,4,5,6};
        int tong = 0;
        for(int i=0; i<mang.Length; i++)   {
                tong = tong + mang[i];
        }
}
```

### 1.8.2. Mảng hai chiều

Trong C#, ngoài việc khai báo mảng một chiều như mục 1.8.1 thì lập trình viên có thể khai mảng hai chiều hoặc nhiều chiều tùy thuộc vào mục đích sử dụng. Và mỗi mảng khi khai báo lại có thể khai báo theo hai dạng khác nhau là mảng đa chiều cùng kích thước hoặc mảng đa chiều không cùng kích thước.

- Mảng hai chiều cùng kích thước: Là mảng được tổ chức thành các dòng và các cột, trong đó dòng được tính theo hàng ngang và cột được tính theo hàng dọc của mảng. Trong mảng hai chiều cùng kích thước thì số lượng phần tử trên mỗi dòng là bằng nhau.

– Cú pháp:

```
<kieu du lieu> [,] <ten mang> = new <kieu du lieu>[<dòng>,<cột>];
```

Ví dụ 1.54: Khai báo mảng hai chiều cùng kích thước kiểu số nguyên có 3 dòng 4 cột.

```
int [,] M = new int[3,4];
```

4 cột

| | | | |
|---|---|---|---|
| M[0,0] | M[0,1] | M[0,2] | M[0,3] |
| M[1,0] | M[1,1] | M[1,2] | M[1,3] |
| M[2,0] | M[2,1] | M[2,2] | M[2,3] |

3 dòng

Ví dụ 1.55: Khai báo mảng M có kích thước 2 dòng, 3 cột với các phần tử tương ứng như sau:

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |

Tính tổng các phần tử trong mảng M

```
public static void Main(string []args)
{       int[,] M = new int[2,3];
        int tong = 0, i, j, k=1;
        for(i=0; i<2; i++)
                for(j=0; j<3; j++)     {
                        M[i, j] = k;
                        k = k+1;
                }
         for(i=0; i<2; i++)
                for(j=0; j<3; j++)
                        tong = tong + M[i, j];
}
```

➢ Mảng hai chiều không cùng kích thước: Là mảng được tổ chức thành các dòng và các cột, trong đó dòng được tính theo hàng ngang và cột được tính theo hàng dọc của mảng. Trong mảng hai chiều không cùng kích thước thì số lượng phần tử trên mỗi dòng có thể khác nhau.

- Cú pháp:

Khi khai báo mảng hai không chiều cùng kích thước phải khai báo số dòng của mảng trước

```
<kieu du lieu> [][] <Ten mang> = new <kieu du lieu>[<dòng>][];
```

Sau khi đã khởi tạo số dòng của mảng, cần khai báo số cột trên từng dòng của mảng:

```
<ten mang> [0] = new <kieu du lieu>[<Số cột 0>];
<ten mang> [1] = new <kieu du lieu>[<Số cột 1>];
…..
<ten mang> [dòng] = new <kieu du lieu>[< Số cột n>];
```

Ví dụ 1.56: Khai báo mảng hai chiều có 3 dòng, dòng 1 có 3 cột, dòng hai có 4 cột, dòng 3 có 5 cột.

Gán phần tử vị trí dòng 0 và cột 1 giá trị 10

Gán phần tử vị trí dòng 1 và cột 0 giá trị 5

Gán phần tử vị trí dòng 2 và cột 4 giá trị 15

```
public static void Main(string []args)
{
        int[][] M = new int[3][];
        M[0] = new int[3];
        M[1] = new int[4];
        M[2] = new int[5];
        M[0][1] = 10;
        M[1][0] = 5;
        M[2][4] = 15;
}
```

| | | | | |
|---|---|---|---|---|
| 0 | 10 | 0 | | |
| 5 | 0 | 0 | 0 | |
| 0 | 0 | 0 | 0 | 15 |

## 1.9. Tạo và sử dụng DLL trong C#

DLL là từ viết tắt của Dynamic linking library, gọi là thư viện liên kết động, là các module chứa các phương thức và dữ liệu.các phương thức và dữ liệu trong Module này sẽ được đóng gói vào 1 tập tin có đuôi .DLL.

### 1.9.1. Ưu điểm và nhược điểm khi sử dụng DLL

*Một số ưu điểm khi sử dụng DLL:*