

**KHOA CÔNG NGHỆ THÔNG TIN**

**BÀI GIẢNG**  
**LẬP TRÌNH .NET**  
**(LƯU HÀNH NỘI BỘ)**

TÊN HỌC PHẦN

MÃ HỌC PHẦN :

TRÌNH ĐỘ ĐÀO TẠO : ĐẠI HỌC CHÍNH QUY

DÙNG CHO SV NGÀNH : CÔNG NGHỆ THÔNG TIN

**HÀ NỘI - 2020**



## Lời nói đầu.

Môn học Lập trình windows 1 là môn học **chuyên ngành**, Khoa Công nghệ thông tin. Sử dụng **ngôn ngữ lập trình C#**, được kế thừa, phát huy những ưu điểm từ các ngôn ngữ C/C++, Java và hạn chế được những nhược điểm từ các ngôn ngữ này. C# là ngôn ngữ lập trình mới được thiết kế bởi Microsoft từ năm 2000 cho nhiều ứng dụng doanh nghiệp chạy trên nền tảng .NET.

Như một bước phát triển của C và C++, ngôn ngữ này là hiện đại, đơn giản, hướng đối tượng và chức năng.

Hơn nữa C# cũng là **ngôn ngữ phát triển Game** mạnh thứ 2 sau C/C++ có khả năng execute nhanh. Với C #, trò chơi có thể được phát triển cho hầu hết mọi nền tảng trên mạng ở mọi qui mô. Xbox hay Windows hay Play Station hay IOS hay Android?. Mặt khác, C # được cho là thân thiện với người mới bắt đầu hơn cả C và C ++, vì vậy có thể dễ dàng để làm chủ và sử dụng.

Trong phạm vi môn học, nội dung chủ yếu:

- Lập trình Console.
- Các kiểu dữ liệu cơ bản.
- Khai thác tài nguyên trên máy tính Windows như thư mục, file.
- Lập trình hướng đối tượng với C#
- Thực thi giao diện, Cơ chế ủy quyền và sự kiện

Xin cảm ơn Khoa công nghệ thông tin, Đại học thành Đô đã giúp hoàn thành bài giảng này. Mọi ý kiến phản hồi nhằm nâng cao chất lượng xin gửi về hòm thư tnhoang@thanhdo.edu.vn .

Hà nội tháng 06 năm 2020.

Lịch giảng dạy môn học Lập trình windows 1 .....	<b>Error! Bookmark not defined.</b>
ĐỀ CƯƠNG CHI TIẾT HỌC PHẦN.....	<b>Error! Bookmark not defined.</b>
Chương 1: TỔNG QUAN VỀ .NET VÀ NGÔN NGỮ C# .....	1
1.1. Tổng quan Microsoft .Net.....	1
1.2 Ngôn ngữ C#.....	1
1.3 Giới thiệu công cụ lập trình Visual Studio .....	1
1.4 Viết một chương trình C# đơn giản .....	3
1.5. Biên dịch và thực thi một chương trình ứng dụng C# .....	5
1.6 Bài tập cuối chương .....	6
Chương 2 KIỂU DỮ LIỆU, BIẾN, HẰNG, BIỂU THỨC VÀ TOÁN TỬ.....	7
2.1. Kiểu dữ liệu.....	7
2.2. Biến và hằng .....	8
2.3. Biểu thức .....	11
2.4. Các toán tử .....	12
2.5. Không gian tên.....	19
2.6. Các chỉ thị tiên xử lý .....	21
2.7 Bài tập cuối chương .....	24
Chương 3: CÁC CẤU TRÚC ĐIỀU KHIỂN.....	26
3.1. Cấu trúc rẽ nhánh.....	26
3.2. Cấu trúc lặp .....	33
3.3. Nhãn.....	39
3.4. Bẫy lỗi và xử lý lỗi hay Xử lý ngoại lệ.....	39
3.5 Bài tập cuối chương. ....	42
Chương 4: MẢNG, CHỈ MỤC VÀ TẬP HỢP.....	46
4.1. Mảng .....	46
4.2. Chỉ mục.....	54
4.3. Tập hợp .....	56
4.4 Bài tập cuối chương .....	58
Chương 5: XỬ LÝ CHUỖI VÀ TẬP TIN .....	60
5.1. Tạo chuỗi .....	60
5.2. Hàm ToString().....	61

5.3. Thao tác trên chuỗi .....	62
5.4. Tập tin và thư mục .....	64
5.5. Đọc và ghi dữ liệu.....	65
5.5 Bài tập cuối chương.....	74
Chương 6: LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG VỚI C#.....	75
6.1. Xây dựng lớp, đối tượng.....	75
6.2. Kế thừa, đa hình.....	77
6.3. Nạp chồng toán tử.....	82
Chương 7: THỰC THI GIAO DIỆN.....	90
7.1. Khai báo giao diện .....	90
7.2. Thực thi giao diện .....	90
7.3 Bài tập cuối chương.....	92
Chương 8: CƠ CHẾ ỦY QUYỀN VÀ SỰ KIỆN .....	93
8.1. Cơ chế ủy quyền (delegate) .....	93
8.2. Sự kiện (event).....	94
8.3 Bài tập cuối chương .....	99
Chương 9: CÁC ĐỐI TƯỢNG ĐIỀU KHIỂN TRÊN FORM.....	100
9.1. Biểu mẫu (form).....	100
9.2. Điều khiển Label.....	103
9.3. Điều khiển LinkLabel .....	105
9.4. Điều khiển TextBox.....	105
9.5. Điều khiển Button.....	107
9.6. Điều khiển CheckBox .....	108
9.7. Điều khiển RadioButton .....	108
9.8. Điều khiển PictureBox.....	108
9.9 Bài tập cuối chương.....	109
Đề thi mẫu .....	109
Danh sách bài tập lớn .....	111
Tài liệu tham khảo .....	113



## **Chương 1: TỔNG QUAN VỀ .NET VÀ NGÔN NGỮ C#**

- 1.1. Tổng quan Microsoft .Net**
- 1.2. Ngôn ngữ C#**
- 1.3. Giới thiệu công cụ lập trình Visual Studio**
- 1.4. Viết một chương trình C# đơn giản**
- 1.5. Biên dịch và thực thi một chương trình ứng dụng C#**

## **Chương 2: KIỂU DỮ LIỆU, BIẾN, HẲNG, BIỂU THỨC VÀ TOÁN TỬ**

- 2.1. Kiểu dữ liệu**
- 2.2. Biến và hằng**
- 2.3. Biểu thức**
- 2.4. Các toán tử**
- 2.5. Không gian tên**
- 2.6. Các chỉ thị tiền xử lý**

## **Chương 3: CÁC CẤU TRÚC ĐIỀU KHIỂN**

- 3.1. Cấu trúc rẽ nhánh**
- 3.2. Cấu trúc lặp**
- 3.3. Nhãn**
- 3.4. Bẫy lỗi và xử lý lỗi**

## **Chương 4: MẢNG, CHỈ MỤC VÀ TẬP HỢP**

- 4.1. Mảng**
- 4.2. Chỉ mục**
- 4.3. Tập hợp**

## **Chương 5: XỬ LÝ CHUỖI VÀ TẬP TIN**

- 5.1. Tạo chuỗi**
- 5.2. Hàm ToString()**
- 5.3. Thao tác trên chuỗi**
- 5.4. Tập tin và thư mục**
- 5.5. Đọc và ghi dữ liệu**

## **Chương 6: LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG VỚI C#**

- 6.1. Xây dựng lớp, đối tượng**
- 6.2. Kế thừa, đa hình**
- 6.3. Nạp chồng toán tử**

## **Chương 7: THỰC THI GIAO DIỆN**

- 7.1. Khai báo giao diện**
- 7.2. Truy cập phương thức giao diện**
- 7.3. Thực thi phủ quyết giao diện**
- 7.4. Thực thi giao diện tường minh**

## **Chương 8: CƠ CHẾ ỦY QUYỀN VÀ SỰ KIỆN**

- 8.1. Cơ chế ủy quyền (delegate)**
- 8.2. Sự kiện (event)**

## **Chương 9: CÁC ĐỐI TƯỢNG ĐIỀU KHIỂN TRÊN FORM**

- 9.1. Biểu mẫu (form)**
- 9.2. Điều khiển Label**
- 9.3. Điều khiển LinkLabel**
- 9.4. Điều khiển TextBox**
- 9.5. Điều khiển Button**
- 9.6. Điều khiển CheckBox**
- 9.7. Điều khiển RadioButton**
- 9.8. Điều khiển PictureBox**
- 13. Tài liệu học tập**
- 14. Phê duyệt**



# Chương 1: TỔNG QUAN VỀ .NET VÀ NGÔN NGỮ C#

## 1.1. Tổng quan Microsoft .Net

Bắt đầu làm quen với một ngôn ngữ lập trình nào đó với sinh viên là một việc khá khó khăn và bối rối, với những bạn có khả năng và tư duy tốt thì sẽ ít gặp trở ngại hơn nhưng chúng ta đều gặp vấn đề chung là không biết phải bắt đầu nó từ đâu, học sao cho đúng. *Học C# nên bắt đầu từ đâu?*

Bắt đầu với việc học C# trên Console.

## 1.2 Ngôn ngữ C#

Giới thiệu một về ngôn ngữ C#: C# là một sản phẩm của Microsoft, là một ngôn ngữ hướng đối tượng khá thân thiện, mềm dẻo và chúng ta có thể dùng nó để tạo nên những ứng dụng trên desktop hay web. Ngoài ra C# còn có sẵn một khối thư viện khổng lồ với các hàm hỗ trợ mạnh trong việc lập trình.

Chọn Project -> New -> ConsoleApplication.

Chúng ta nên chú ý một vài nguyên tắc sau:

*C# là ngôn ngữ phân biệt hoa thường .*

*Quy tắc đặt tên trong C# .*

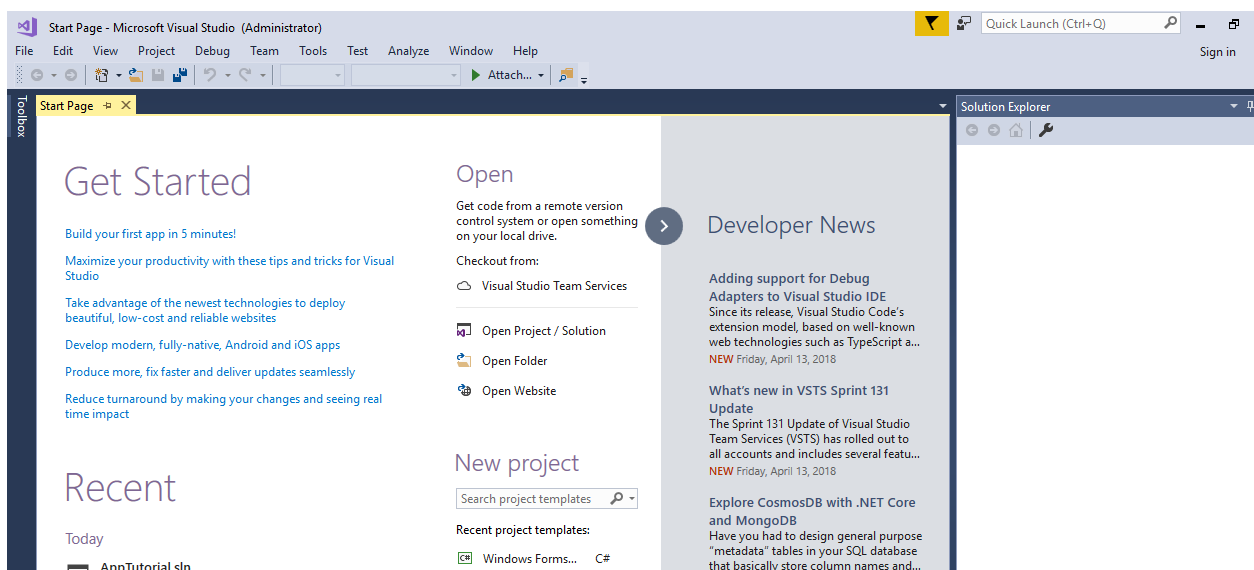
*Quy tắc viết chú thích.*

Cách khai báo các thư viện dùng trong chương trình.

## 1.3 Giới thiệu công cụ lập trình Visual Studio

Microsoft Visual Studio là một môi trường phát triển tích hợp từ Microsoft. Nó được sử dụng để phát triển chương trình máy tính cho Microsoft Windows, cũng như các trang web, các ứng dụng web và các dịch vụ web. Visual Studio sử dụng nền tảng phát triển phần mềm của Microsoft như Windows API, Windows Forms, Windows Presentation Foundation, Windows Store và Microsoft Silverlight. Nó có thể sản xuất cả hai ngôn ngữ máy và mã số quản lý.

Visual Studio bao gồm một trình soạn thảo mã hỗ trợ IntelliSense cũng như cải tiến mã nguồn. Trình gỡ lỗi tích hợp hoạt động cả về trình gỡ lỗi mức độ mã nguồn và gỡ lỗi mức độ máy. Công cụ tích hợp khác bao gồm một mẫu thiết kế các hình thức xây dựng giao diện ứng dụng, thiết kế web, thiết kế lớp và thiết kế giản đồ cơ sở dữ liệu. Nó chấp nhận các plug-in nâng cao các chức năng ở hầu hết các cấp bao gồm thêm hỗ trợ cho các hệ thống quản lý phiên bản (như Subversion) và bổ sung thêm bộ công cụ mới như biên tập và thiết kế trực quan cho các miền ngôn ngữ cụ thể hoặc bộ công cụ dành cho các khía cạnh khác trong quy trình phát triển phần mềm.



## GIAO DIỆN LÀM VIỆC

**Màu cam:**(Solution Explorer)là cửa sổ hiển thị Solution, các Project và các tập tin trong project.

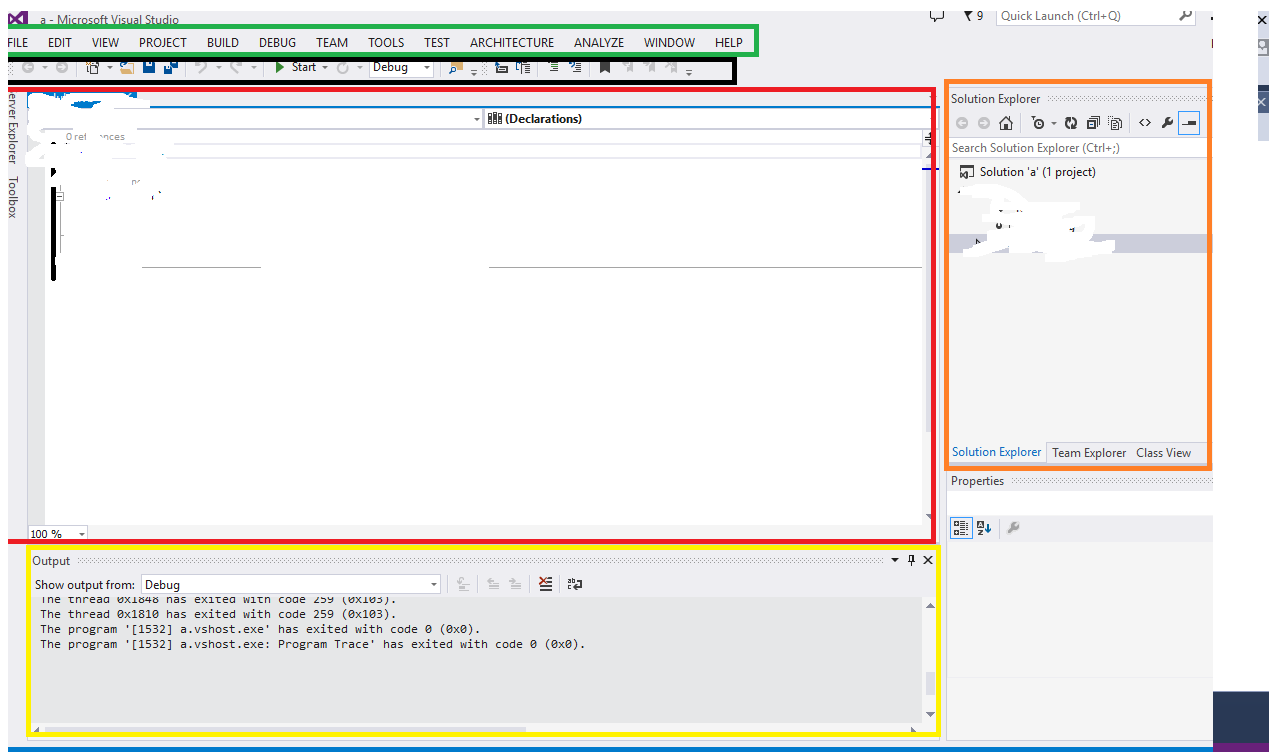
**Màu đỏ:**đây là khu vực để lập trình viên viết mã nguồn cho chương trình. Cửa sổ lập trình cho một tập tin trong Project sẽ hiển thị khi người dùng nhấp đúp chuột lên tập tin đó trong cửa sổ Solution Explorer.

**Màu vàng:**(Output)đây là cửa sổ hiển thị các thông tin, trạng thái của Solution khi build hoặc của chương trình khi debug.

**Màu đen**(Toolbar) với các công cụ hỗ trợ người dùng trong việc viết mã và debug (các công cụ trên thanh có thể thay đổi khi bắt đầu debug).

**Màu xanh lá cây** (Thanh menu) với đầy đủ các danh mục chứa các chức năng của VS. Khi người dùng cài thêm những trình cắm hỗ trợ VS (ví dụ như Visual Assist), thanh menu này sẽ cập nhật thêm menu của các trình cắm (nếu có).

Một số cửa sổ bị ẩn bạn có thể tìm ở trong :View -> Other Windows, các thanh công cụ được đặt trong View -> Toolbars.



#### 1.4 Viết một chương trình C# đơn giản

Chúng ta sẽ bắt đầu với việc in ra dòng chữ “Hello World”, nó sẽ tạo ra sự thân thiện trong việc tiếp xúc ngôn ngữ.

Vd:

```
using System;

using System.Collections.Generic;

using System.Text;

namespace DaiHocTinK8KhoaCNTT
{
    class Program
    {
        static void Main(string[] args)
        {
            //xuất ra màn hình dòng chữ "Hello World"

            Console.WriteLine("Hello World ");
        }
    }
}
```

```
Console.ReadLine(); }  
  
}  
  
}
```

Sau khi viết xong chúng ta lưu nó lại với đuôi .cs (mặc định trong visual đã có) sau đó ấn F6 để biên dịch và ấn F5 để chạy chương trình. Kết quả xuất hiện trên màn hình dòng chữ “Hello World”.

### ***Giải thích ví dụ:***

Ví dụ trên đây là sự thể hiện ứng dụng console. Hiểu đơn giản ứng dụng console là giao tiếp với người dùng bằng bàn phím và không có giao diện người dùng. Bây giờ chúng ta sẽ cùng nhau khai phá ví dụ này nhé.

Namespace: trong ví dụ trên namespace được khai báo như sau

```
Namespace : DaiHocTinK8KhoaCNTT  
  
{  
  
//nơi chứa định các class ()  
  
}
```

Trong đó: namespace là từ khóa khai báo.

DaiHocTinK8KhoaCNTT là tên của namespace.

class program được chứa trong namespace và trong namespace có thể chứa được nhiều class. Việc sử dụng namespace nhằm giải quyết xung đột tên class trong cùng một project. Namespace là một gói những thực thể có thuộc tính và hành vi độc lập với bên ngoài và có 2 ưu điểm như sau:

- *Tránh được sự trùng lặp tên giữa các class.*
- *Cho phép tổ chức mã nguồn một cách có khoa học và hợp lý.*

Từ khóa using: để làm cho chương trình gọn hơn và nhất cần phải viết từng namespace cho từng đối tượng, C# cung cấp từ khóa “using”. Có thể sử dụng dòng lệnh “using System” ở ngay đầu chương trình, trước định nghĩa lớp và khi chúng ta dùng tới đối tượng console thay vì phải viết là “System.Console” thì chỉ cần viết Console . Việc này sẽ làm cho code ngắn gọn, rõ ràng hơn.

Toán tử “.”: dùng để truy cập đến dữ liệu hay phương thức trong cùng một lớp, và ngăn cách giữa tên lớp đến một namespace theo chiều từ trên xuống.

Hàm **Main()**: trong C# hàm Main() được quy định ký tự đầu viết hoa và hàm có thể trả về giá trị void hay int và luôn khai báo là static.

Chú thích (comment): //xuất ra màn hình dòng chữ "Hello World"

Trong một project việc chú thích là vô cùng quan trọng và không thể thiếu nó làm cho chương trình dễ hiểu, rõ ràng và chúng ta không bị gặp trở ngại khi quay lại xem code. Trước mình làm project kỳ 2 điểm cho phần chú thích code chiếm khá cao vì thế mọi người nên chú ý nhé. Chúng ta có thể chú thích code bằng những cách như sau:

**Cách 1:** sử dụng ký tự “//”. Khi gặp ký tự này trình biên dịch sẽ bỏ qua cả dòng đó vì thế nó thích hợp nhất khi chúng ta muốn chú thích trong 1 dòng.

**Cách 2:** chúng ta khai báo “/\*” ở đầu phần chú thích và “\*/” ở cuối phần chú thích vì thế mà thích hợp cho việc chú thích trong nhiều dòng hay một khối.

**Cách 3:** chú thích XML ghi chép tài liệu cho một lớp hoặc phương thức bằng cách sử dụng 1 phần XML (cái này các bạn sẽ tự nghiên cứu sau khi nào đã hiểu hết được về C# để tránh tẩu hỏa nhập ma).

*Trong ví dụ trên để viết ra dòng chữ “Hello World” sử dụng phương thức WriteLine() và phương thức ReadLine() sẽ dừng lại màn hình để cho nó không bị chạy mất sau 1, 2 giây xuất hiện.*

## 1.5. Biên dịch và thực thi một chương trình ứng dụng C#

## 1.6 Bài tập cuối chương

**Bài tập 1:** Nhập vào chương trình sau và biên dịch nó. Cho biết chương trình thực hiện điều gì?

```
using System;
class variables
{
    public static void Main()
    {
        int radius = 4;
        const double PI = 3.14159;
        double circum, area;
        area = PI * radius * radius;
        circum = 2 * PI * radius;
        // in kết quả
        Console.WriteLine("Ban kinh = { 0}, PI = { 1}", radius, PI);
        Console.WriteLine("Dien tich { 0}", area); Console.WriteLine("Chu vi { 0}", circum);
    }
}
```

```
class AClass
{
    static void Main()
    {
        int x, y;
        for (x = 0; x < 10; x++, System.Console.Write("\n")) ;
        for (y = 0; y < 10; y++, System.Console.WriteLine("{ 0}",y));
    }
}
```

**Bài tập 2:** Viết chương trình xuất ra bài thơ:

Rằm Tháng Giêng

Rằm xuân lồng lộng trăng soi,  
Sông xuân nước lẫn màu trời thêm xuân.

Giữa dòng bàn bạc việc quân  
Khuya về bát ngát trăng ngân đầy thuyền.

Hồ Chí Minh.

**Bài tập 3:** Viết chương trình Console hiển thị các số từ 1-10;

## Chương 2 KIỂU DỮ LIỆU, BIẾN, HẰNG, BIỂU THỨC VÀ TOÁN TỬ

### 2.1. Kiểu dữ liệu

Kiểu dữ liệu được dùng để khai báo biến, khai báo hằng ...

C# chia thành hai tập hợp kiểu dữ liệu chính: Kiểu xây dựng sẵn (built-in) mà ngôn ngữ cung cấp cho người lập trình và kiểu được người dùng định nghĩa (user-defined) do người lập trình tạo ra. C# phân tập hợp kiểu dữ liệu này thành hai loại: Kiểu dữ liệu giá trị (value) và kiểu dữ liệu tham chiếu (reference).

**Kiểu dữ liệu giá trị:** là các kiểu dữ liệu lưu trữ các thông tin thức tế.

Các kiểu dữ liệu xây dựng sẵn

**byte:** số nguyên dương không dấu từ 0-255

**char:** ký tự Unicode

**int:** số nguyên có dấu -2.147.483.647 và 2.147.483.647

**float:** kiểu dấu chấm động, giá trị xấp xỉ từ 3,4E-38 đến 3,4E+38, với 7 chữ số có nghĩa..

**sbyte** : số nguyên có dấu ( từ -128 đến 127)

**short:** số nguyên có dấu giá trị từ -32768 đến 32767.

**ushort:** số nguyên không dấu 0 – 65.535

**uint:** số nguyên không dấu 0 – 4.294.967.295

**double:** kiểu dấu chấm động có độ chính xác gấp đôi, giá trị xấp xỉ từ 1,7E-308 đến 1,7E+308, với 15,16 chữ số có nghĩa.

**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:** kiểu số nguyên có dấu có giá trị trong khoảng :-9.223.370.036.854.775.808 đến 9.223.372.036.854.775.807

**ulong:** Số nguyên không dấu từ 0 đến 0xffffffffffffffff

**bool:** Giá trị logic true/ false

**Kiểu dữ liệu tham chiếu:**

**object:** đây là kiểu dữ liệu cơ sở chứa tất cả các kiểu dữ liệu khác trong C#.

**string:** kiểu dữ liệu chuỗi ký tự.

class: kiểu dữ liệu class.

delegate: kiểu dữ liệu chuyển giao.

interface: kiểu dữ liệu giáo tiếp.

array: kiểu dữ liệu mảng.

## 2.2. Biến và hằng

**Định nghĩa biến:** Biến là một vùng nhớ được đặt tên, được sử dụng để lưu trữ dữ liệu trong chương trình.

**Cú pháp khai báo biến:** Để khai báo biến chúng ta có cấu trúc như sau:

<kiểu dữ liệu><tên biến>;

Vd: để khai báo 1 biến a kiểu một số nguyên ta làm như sau:

```
int a;
```

Ví dụ:

```
using System;
using System.Collections.Generic;
using System.Text;
namespace SuDungBien
{
    class MinhHoa
    {
        static void Main()
        {
            int bien1 = 3; // khai báo và khởi tạo giá trị cho một biến
            Console.WriteLine("Sau khi khai tạo: bien1 ={0}", bien1);
            bien1 = 10; // gán giá trị cho biến
            Console.WriteLine("Sau khi gán: bien1 ={0}", bien1);
            Console.ReadLine();
        }
    }
}
```

Khai báo nhiều biến có cùng kiểu dữ liệu:

<kiểu dữ liệu><biến 1><biến 2>, ..., <biến n>;

Ví dụ:

```
int a,b,c;
```

Gán cùng một giá trị cho nhiều biến( “đa gán”).



```
a = b = c = 3;
```

**Chú ý:**

*a, trong C# yêu cầu các biến phải được khởi tạo trước khi sử dụng.*

*b, tên biến trong C# phân biệt chữ hoa, chữ thường.*

*c, tên biến chỉ có thể bắt đầu với ký tự hoặc dấu \_ chứ không thể bắt đầu với chữ số.*

*d, tên biến không được chứa các ký tự đặc biệt như \$, #, %, ^...*

*e, tên biến không được trùng với từ khóa nhưng nếu muốn đặt trùng tên với từ khóa thì dùng @ ở đằng trước.*

*Và cuối cùng hãy nhớ đặt tên biến sao cho nó phản ánh được ý nghĩa mà chúng ta sử dụng nó để thuận lợi hơn trong việc kiểm tra lại code.*

Ví dụ:

```
int iSiSoLopDHTINK8;
```

**Định nghĩa Hằng** là một biến nhưng giá trị của nó không thay đổi.

Đầu tiên sẽ là cấu trúc khai báo hằng

**<const><kiểu dữ liệu><tên hằng> = <giá trị>;**

Hằng được phân thành 3 loại:

Ví dụ:

```
const int a = 20;
```

Giá trị hằng: từ ví dụ trên thì giá trị 20 chính là giá trị hằng và giá trị 20 là không đổi.

**Biểu tượng hằng:** một biểu tượng hằng phải được khởi tạo trước khi khai báo và chỉ duy nhất một lần trong suốt quá trình và không được thay đổi. Trong ví dụ trên thì a chính là biểu tượng hằng có kiểu số nguyên.

Ví dụ:

```
class KhoaTinDHTD
{
    static void Main()
    {
        const int diemmax = 10; // Điểm cao nhất
        const int diemliet = 0; // Điểm liệt
    }
}
```

```
        System.Console.WriteLine( "điểm thi cao nhất là { 0}", diemmax );
        System.Console.WriteLine( "điểm liệt là { 0}", diemliet );
    }
}
```

**Kiểu liệt kê:** kiểu liệt kê là tập hợp các tên hằng có giá trị không đổi (danh sách liệt kê).

Trong ví dụ trên ta có:

```
const int diemmax = 10;
```

```
const int diemliet = 0;
```

Bây giờ theo yêu cầu mở rộng của bài toán chúng ta thêm một số hằng khác vào danh sách trên:

Và dĩ nhiên các biểu tượng hằng trên đều có ý nghĩa quan hệ với nhau, chúng cùng nhắc đến các mốc điểm để đánh giá điểm thi. Một vấn đề đặt ra nếu chúng ta để riêng lẻ từng hằng trên thì chương trình nhìn sẽ không đẹp mắt và cồng kềnh. Lúc này đây C# cung cấp cho người lập trình một giải pháp để giải quyết vấn đề trên đó là kiểu liệt kê:

### Cú pháp

```
enum [kiểu cơ sở]
```

```
{
```

Danh sách các thành phần liệt kê

```
};
```

```
enum diemthi : int
{
    diemmax = 10,
    diemliet = 0,
    diemtb = 5,
    diemkha = 7,
};
```

**Chú ý:** mỗi kiểu liệt kê đều có một kiểu dữ liệu cơ sở có thể là bất kỳ kiểu dữ liệu nào như int, short, long...tuy nhiên kiểu dữ liệu liệt kê không chấp nhận kiểu ký tự và nếu chúng ta bỏ qua phần này thì trình biên dịch sẽ gán giá trị mặc định là kiểu nguyên (int). Kiểu liệt kê là một kiểu hình thức do đó bắt buộc phải thực hiện phép chuyển đổi tương minh với các kiểu giá trị nguyên:

```
using System;
using System.Collections.Generic;
using System.Text;
namespace
{
    class KieuEnum
    {
        enum diemthi : int
```

```

    {
        diemmax = 10,
        diemliet = 0,
        diemtb = 5,
        diemkha = 7,
    };
    static void Main()
    {
        System.Console.WriteLine("diem cao nhat: {0}", (int)diemthi.diemmax);
        System.Console.WriteLine("diem liet: {0}", (int)diemthi.diemliet);
        System.Console.WriteLine("diem trung binh {0}", (int)diemthi.diemtb);
        System.Console.WriteLine("diem kha: {0}", (int)diemthi.diemkha);
        Console.ReadLine();
    }
}

```

Chú ý: Mỗi thành phần trong kiểu liệt kê tương ứng với một giá trị số, trong trường hợp này là một số nguyên. Nếu chúng ta không khởi tạo cho các thành phần này thì chúng sẽ nhận các giá trị tiếp theo với thành phần đầu tiên là 0.

```

using System;
using System.Collections.Generic;
using System.Text;
namespace ConsoleApplication18
{
    class Program
    {
        enum diemthi : int
        {
            diemmax,
            diemliet,
            diemtb,
            diemkha,
        };
        static void Main(string[] args)
        {
            System.Console.WriteLine("diem cao nhat: {0}", (int)diemthi.diemmax);
            System.Console.WriteLine("diem liet: {0}", (int)diemthi.diemliet);
            System.Console.WriteLine("diem trung binh {0}", (int)diemthi.diemtb);
            System.Console.WriteLine("diem kha: {0}", (int)diemthi.diemkha);
            Console.ReadLine();
        }
    }
}

```

Khi đó giá trị thứ nhất là 0, các giá trị sau sẽ là 1; 2; 3.(đây là trường hợp với số nguyên).

### 2.3. Biểu thức

Những câu lệnh mà thực hiện việc đánh giá một giá trị gọi là biểu thức. Một phép gán một giá trị cho một biến cũng là một biểu thức:

$$var1 = 24;$$

Trong câu lệnh trên phép đánh giá hay định lượng chính là phép gán có giá trị là 24 cho biến var1. Lưu ý là toán tử gán ('=') không phải là toán tử so sánh. Do vậy khi sử dụng toán tử này thì biến bên trái sẽ nhận giá trị của phần bên phải. Các toán tử của ngôn ngữ C# như phép so sánh hay phép gán sẽ được trình bày chi tiết trong mục toán tử của chương này.

Do  $var1 = 24$  là một biểu thức được định giá trị là 24 nên biểu thức này có thể được xem như phần bên phải của một biểu thức gán khác:

$$var2 = var1 = 24;$$

Lệnh này sẽ được thực hiện từ bên phải sang khi đó biến var1 sẽ nhận được giá trị là 24 và tiếp sau đó thì var2 cũng được nhận giá trị là 24. Do vậy cả hai biến đều cùng nhận một giá trị là 24. Có thể dùng lệnh trên để khởi tạo nhiều biến có cùng một giá trị như:

$$a = b = c = d = 24;$$

## 2.4. Các toán tử

Một toán tử là một biểu tượng, mà nói cho compiler thực hiện các thao tác toán học và logic cụ thể. C# cung cấp nhiều toán tử có sẵn, đó là:

- Toán tử số học
- Toán tử quan hệ
- Toán tử logic
- Toán tử so sánh bit
- Toán tử gán
- Toán tử hỗn hợp

### Toán tử số học trong C#

Bảng dưới liệt kê các toán tử số học được hỗ trợ bởi ngôn ngữ C#. Giả sử biến A giữ giá trị 10, biến B giữ 20 thì:

Toán tử	Miêu tả	Ví dụ
+	Thêm hai toán hạng	$A + B$ sẽ cho kết quả là 30
-	Trừ giá trị toán hạng hai từ toán hạng đầu	$A - B$ sẽ cho kết quả là -10
*	Nhân hai toán hạng	$A * B$ sẽ cho kết quả là 200

/	Chia lấy phần nguyên hai toán hạng	B / A sẽ cho kết quả là 2
%	Chia lấy phần dư	B % A sẽ cho kết quả là 0
++	Lượng gia giá trị toán hạng thêm 1 đơn vị	A++ sẽ cho kết quả là 11
--	Lượng giảm giá trị toán hạng một đơn vị	A-- sẽ cho kết quả là 9

### Toán tử quan hệ trong C#

Bảng dưới đây liệt kê các toán tử quan hệ được hỗ trợ bởi ngôn ngữ C#. Giả sử biến A giữ giá trị 10, biến B giữ 20 thì:

Toán tử	Miêu tả	Ví dụ
==	Kiểm tra nếu 2 toán hạng bằng nhau hay không. Nếu bằng thì điều kiện là true.	(A == B) là không đúng.
!=	Kiểm tra 2 toán hạng có giá trị khác nhau hay không. Nếu không bằng thì điều kiện là true.	(A != B) là true.
>	Kiểm tra nếu toán hạng bên trái có giá trị lớn hơn toán hạng bên phải hay không. Nếu lớn hơn thì điều kiện là true.	(A > B) là không đúng.
<	Kiểm tra nếu toán hạng bên trái nhỏ hơn toán hạng bên phải hay không. Nếu nhỏ hơn thì là true.	(A < B) là true.
>=	Kiểm tra nếu toán hạng bên trái có giá trị lớn hơn hoặc bằng giá trị của toán hạng bên phải hay không. Nếu đúng là true.	(A >= B) là không đúng.
<=	Kiểm tra nếu toán hạng bên trái có giá trị nhỏ hơn hoặc bằng toán hạng bên phải hay không. Nếu đúng là true.	(A <= B) là true.

### Toán tử logic trong C#

Bảng dưới đây chỉ rõ tất cả các toán tử logic được hỗ trợ bởi ngôn ngữ C#. Giả sử biến A có giá trị 1 và biến B có giá trị 0:

Toán tử	Miêu tả	Ví dụ
&&	Được gọi là toán tử logic AND (và). Nếu cả hai toán tử đều có giá trị khác 0 thì điều kiện trở lên true.	(A && B) là false.
	Được gọi là toán tử logic OR (hoặc). Nếu một trong hai toán tử khác 0, thì điều kiện là true.	(A    B) là true.
!	Được gọi là toán tử NOT (phủ định). Sử dụng để đảo ngược lại trạng thái logic của toán hạng đó. Nếu điều kiện toán hạng là true thì phủ định nó sẽ là false.	!(A && B) là true.

### Toán tử so sánh bit trong C#

Toán tử so sánh bit làm việc trên đơn vị bit, tính toán biểu thức so sánh từng bit. Bảng dưới đây về &, |, và ^ như sau:

p	q	p & q	p   q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Giả sử nếu A = 60; và B = 13; thì bây giờ trong định dạng nhị phân chúng sẽ là như sau:

A = 0011 1100

B = 0000 1101

-----  
 $A \& B = 0000\ 1100$

$A | B = 0011\ 1101$

$A \wedge B = 0011\ 0001$

$\sim A = 1100\ 0011$

Các toán tử so sánh bit được hỗ trợ bởi ngôn ngữ C# được liệt kê trong bảng dưới đây. Giả sử ta có biến A có giá trị 60 và biến B có giá trị 13, ta có:

Toán tử	Miêu tả	Ví dụ
&	Toán tử AND (và) nhị phân sao chép một bit tới kết quả nếu nó tồn tại trong cả hai toán hạng.	(A & B) sẽ cho kết quả là 12, tức là 0000 1100
	Toán tử OR (hoặc) nhị phân sao chép một bit tới kết quả nếu nó tồn tại trong một hoặc hai toán hạng.	(A   B) sẽ cho kết quả là 61, tức là 0011 1101
^	Toán tử XOR nhị phân sao chép bit mà nó chỉ tồn tại trong một toán hạng mà không phải cả hai.	(A ^ B) sẽ cho kết quả là 49, tức là 0011 0001
~	Toán tử đảo bit (đảo bit 1 thành bit 0 và ngược lại).	(~A) sẽ cho kết quả là -61, tức là 1100 0011.
<<	Toán tử dịch trái. Giá trị toán hạng trái được dịch chuyển sang trái bởi số các bit được xác định bởi toán hạng bên phải.	A << 2 sẽ cho kết quả 240, tức là 1111 0000 (dịch sang trái hai bit)
>>	Toán tử dịch phải. Giá trị toán hạng trái được dịch chuyển sang phải bởi số các bit được xác định bởi toán hạng bên phải.	A >> 2 sẽ cho kết quả là 15, tức là 0000 1111 (dịch sang phải hai bit)

### Toán tử gán trong C#

Đây là những toán tử gán được hỗ trợ bởi ngôn ngữ C#:

Toán tử	Miêu tả	Ví dụ
=	Toán tử gán đơn giản. Gán giá trị toán hạng bên phải cho toán hạng trái.	$C = A + B$ sẽ gán giá trị của $A + B$ vào trong $C$
+=	Thêm giá trị toán hạng phải tới toán hạng trái và gán giá trị đó cho toán hạng trái.	$C += A$ tương đương với $C = C + A$
-=	Trừ đi giá trị toán hạng phải từ toán hạng trái và gán giá trị này cho toán hạng trái.	$C -= A$ tương đương với $C = C - A$
*=	Nhân giá trị toán hạng phải với toán hạng trái và gán giá trị này cho toán hạng trái.	$C *= A$ tương đương với $C = C * A$
/=	Chia toán hạng trái cho toán hạng phải và gán giá trị này cho toán hạng trái.	$C /= A$ tương đương với $C = C / A$
%=	Lấy phần dư của phép chia toán hạng trái cho toán hạng phải và gán cho toán hạng trái.	$C \% = A$ tương đương với $C = C \% A$
<<=	Dịch trái toán hạng trái sang số vị trí là giá trị toán hạng phải.	$C <<= 2$ tương đương với $C = C << 2$
>>=	Dịch phải toán hạng trái sang số vị trí là giá trị toán hạng phải.	$C >>= 2$ tương đương với $C = C >> 2$
&=	Phép AND bit	$C \&= 2$ tương đương với $C = C \& 2$
^=	Phép OR loại trừ bit	$C \wedge= 2$ tương đương với $C = C \wedge 2$
=	Phép OR bit.	$C  = 2$ tương đương với $C = C   2$

### Các toán tử hỗn hợp trong C#



Dưới đây là một số toán tử hỗn hợp quan trọng gồm sizeof, typeof và ? : được hỗ trợ bởi ngôn ngữ C#.

Toán tử	Miêu tả	Ví dụ
sizeof()	Trả về kích cỡ của một kiểu dữ liệu	sizeof(int), trả về 4
typeof()	Trả về kiểu của một lớp	typeof(StreamReader);
&	Trả về địa chỉ của một biến	&a; trả về địa chỉ thực sự của biến
*	Trở tới một biến	*a; tạo con trỏ với tên là a tới một biến
? :	Biểu thức điều kiện (Conditional Expression)	Nếu Condition là true ? Thì giá trị X : Nếu không thì Y
is	Xác định đối tượng là một kiểu cụ thể hay không	If( Ford is Car) // Kiểm tra nếu Ford là một đối tượng của lớp Car
as	Ép kiểu mà không tạo một exception nếu việc ép kiểu thất bại	Object obj = new StreamReader("Hello");  StreamReader r = obj as StreamReader;

### Thứ tự ưu tiên toán tử trong C#

Thứ tự ưu tiên toán tử trong C# xác định cách biểu thức được tính toán. Ví dụ, toán tử nhân có quyền ưu tiên hơn toán tử cộng, và nó được thực hiện trước.

Ví dụ,  $x = 7 + 3 * 2$ ; ở đây, x được gán giá trị 13, chứ không phải 20 bởi vì toán tử \* có quyền ưu tiên cao hơn toán tử +, vì thế đầu tiên nó thực hiện phép nhân  $3 * 2$  và sau đó thêm với 7.

Bảng dưới đây liệt kê thứ tự ưu tiên của các toán tử. Các toán tử với quyền ưu tiên cao nhất xuất hiện trên cùng của bảng, và các toán tử có quyền ưu tiên thấp nhất thì ở bên dưới cùng của bảng. Trong một biểu thức, các toán tử có quyền ưu tiên cao nhất được tính toán đầu tiên.

Loại	Toán tử	Thứ tự ưu tiên
Postfix	() [] -> . ++ - -	Trái sang phải
Unary	+ - ! ~ ++ - - (type)* & sizeof	Phải sang trái
Tính nhân	* / %	Trái sang phải
Tính cộng	+ -	Trái sang phải
Dịch chuyển	<< >>	Trái sang phải
Quan hệ	< <= > >=	Trái sang phải
Cân bằng	== !=	Trái sang phải
Phép AND bit	&	Trái sang phải
Phép XOR bit	^	Trái sang phải
Phép OR bit		Trái sang phải
Phép AND logic	&&	Trái sang phải
Phép OR logic		Trái sang phải
Điều kiện	?:	Phải sang trái
Gán	= += -= *= /= %= >>= <<= &= ^=  =	Phải sang trái
Dấu phẩy	,	Trái sang phải

## 2.5. Không gian tên

### Khái niệm namespace

Namespace trong C# là kỹ thuật phân hoạch không gian các định danh, các kiểu dữ liệu thành những vùng dễ quản lý hơn, nhằm tránh sự xung đột giữa việc sử dụng các thư viện khác nhau từ các nhà cung cấp. Ví dụ khi bạn tạo một lớp trong một namespace nào đó, bạn không cần phải kiểm tra xem nó có bị trùng tên với một lớp nào đó trong namespace khác không.

Ngoài thư viện namespace do Microsoft .NET và các hãng thứ ba cung cấp, ta có thể tạo riêng cho mình các namespace.

### Định nghĩa namespace

Để tạo một namespace sử dụng cú pháp sau:

```
namespace KhôngGianTên
{
    //Định nghĩa lớp A

    //Định nghĩa lớp B...
}
```

Ví dụ:

```
using System;
namespace MyLib
{
    namespace Demo1
    {
        class Example1
        {
            public static void Show1()
            {
                Console.WriteLine("Lop Example1");
            }
        }
    }
    namespace Demo2
    {
        public class Tester
        {
            public static void Main()
            {
                Demo1.Example1.Show1();
                Demo1.Example2.Show2();
            }
        }
    }
}
```

```

    }
}
}

```

Lớp Example2 có cùng namespace MyLib.Demo1 với lớp Example1 nhưng hai khai báo không cùng một tập tin

```

namespace MyLib.Demo1
{
    class Example2
    {
        public static void Show2()
        {
            Console.WriteLine("Lop Example2");
        }
    }
}

```

## Sử dụng namespace

C# đưa ra từ khóa using để khai báo không gian tên cho việc sử dụng các định danh, kiểu dữ liệu định nghĩa thuộc không gian tên trong chương trình:

**using KhôngGianTên;**

Thay vì sử dụng lệnh using, có thể sử dụng dấu chấm truy cập namespace.

Ví dụ:

```

using System;
// cho phép ta sử dụng Console.WriteLine() thay cho System.Console.WriteLine()

```

Ví dụ:

```

using Demo1;
//cho phép ta truy cập Example1.Show1() thay cho Demo1.Example1.Show1();

```

using cũng có thể cung cấp một không gian tên bí danh

Ví dụ:

```

using Utils = Company.Application.Utilities;

```

Lệnh using

Từ khóa using được sử dụng với hai ý nghĩa hoàn toàn không liên quan. Ngoài việc sử dụng như một khai báo không gian tên cho việc tham chiếu, từ khóa using còn được sử dụng để tự động gọi phương thức Dispose() cho đối tượng cụ thể.

Ví dụ: Sau khi đọc tập tin, đối tượng reader sẽ được tự động hủy

```
using (StreamReader reader = new StreamReader("vb.txt"))
{
    //Đọc tập tin
}
```

Ví dụ:

```
using System;
using System.IO;
class Program
{
    static void Main(string[] args)
    {
        using (StreamReader reader = new StreamReader("vb.txt"))
        {
            string line;
            while ((line = reader.ReadLine()) != null)
                Console.WriteLine(line);
        }
    }
}
```

## 2.6. Các chỉ thị tiền xử lý

Các chỉ thị tiền xử lý (Preprocessor Directive) cung cấp chỉ lệnh tới compiler để tiền xử lý thông tin trước khi sự biên dịch thực sự bắt đầu.

Tất cả chỉ thị tiền xử lý trong C# bắt đầu với #, và chỉ có các ký tự white-space có thể xuất hiện ở trước một chỉ thị tiền xử lý trong một dòng. Các chỉ thị tiền xử lý trong C# không là các lệnh, vì thế chúng không kết thúc với một dấu chấm phẩy (;).

Bộ biên dịch của C# không có một bộ tiền xử lý riêng biệt, tuy nhiên, các chỉ thị này được xử lý như khi thực sự có một bộ tiền xử lý riêng vậy. Trong C#, các chỉ thị tiền xử lý được sử dụng để giúp ích việc biên dịch có điều kiện. Không giống các chỉ thị tiền xử lý trong C và C++, chúng không được sử dụng để tạo các macro. Một chỉ thị tiền xử lý phải chỉ là một chỉ lệnh trên một dòng.

Dưới đây là bảng liệt kê các chỉ thị tiền xử lý có sẵn trong C#:

Chỉ thị tiền xử lý	Miêu tả
#define	Nó định nghĩa một dãy ký tự, được gọi là các biểu tượng
#undef	Nó cho phép bạn không định nghĩa (undefine) một biểu tượng

#if	Nó cho phép kiểm tra một biểu tượng hoặc nhiều biểu tượng để thấy nếu chúng ước lượng là true
#else	Nó cho phép tạo một chỉ thị có điều kiện phức hợp, cùng với #if
#elif	Nó cho phép tạo một chỉ thị có điều kiện phức hợp
#endif	Xác định phần cuối của một chỉ thị có điều kiện (conditional directive)
#line	Nó cho phép bạn sửa đổi số dòng của compiler và (tùy ý) tên file cho Error và Warning
#error	Nó cho phép tạo một error từ một vị trí cụ thể trong code của bạn
#warning	Nó cho phép tạo một mức độ cảnh báo từ một vị trí cụ thể trong code của bạn
#region	Nó cho phép bạn xác định một khối code mà bạn có thể mở rộng hoặc thu gọn bởi sử dụng đặc điểm của Visual Studio Code Editor
#endregion	Nó đánh dấu phần cuối của một khối #region

Chỉ thị tiền xử lý #define trong C#

Chỉ thị tiền xử lý #define trong C# tạo các hằng biểu tượng. #define cho phép bạn tạo một biểu tượng như vậy, bởi sử dụng biểu tượng dạng biểu thức được truyền tới chỉ thị tiền xử lý #if, biểu thức ước lượng là true. Cú pháp của nó như sau:

```
#define symbol
```

```
#define PI
using System;
namespace DaihocTink8
{
    class TestClass
    {
        static void Main(string[] args)
```

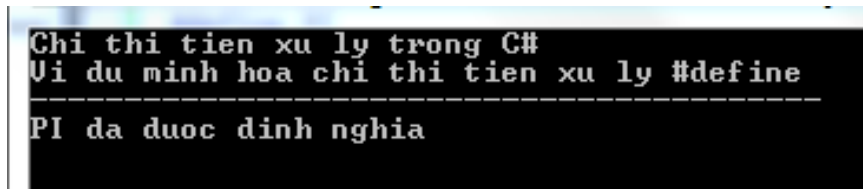
```

    {
        Console.WriteLine("Chi thi tien xu ly trong C#");
        Console.WriteLine("Vi du minh hoa chi thi tien xu ly #define");
        Console.WriteLine("-----");
        #if (PI)
            Console.WriteLine("PI da duoc dinh nghia");
        #else
            Console.WriteLine("PI chua duoc dinh nghia");
        #endif
        Console.ReadKey();
    }
}

```

Nếu bạn không sử dụng lệnh `Console.ReadKey();` thì chương trình sẽ chạy và kết thúc luôn (nhanh quá đến nỗi bạn không kịp nhìn kết quả). Lệnh này cho phép chúng ta nhìn kết quả một cách rõ ràng hơn.

Biên dịch và chạy chương trình C# trên sẽ cho kết quả sau:



```

Chi thi tien xu ly trong C#
Vi du minh hoa chi thi tien xu ly #define
-----
PI da duoc dinh nghia

```

Chỉ thị có điều kiện (Conditional Directive) trong C#

Bạn có thể sử dụng chỉ thị tiền xử lý `#if` trong C# để tạo một chỉ thị có điều kiện (Conditional Directive). Các chỉ thị có điều kiện là hữu ích khi kiểm tra một biểu tượng hoặc các biểu tượng để kiểm tra nếu chúng ước lượng là true. Nếu chúng ước lượng là true, compiler ước lượng tất cả code giữa chỉ thị `#if` và chỉ thị tiếp theo.

Cú pháp cho chỉ thị có điều kiện trong C# là:

`#if symbol [operator symbol]...`

Tại đây, `symbol` là tên của biểu tượng bạn muốn kiểm tra. Bạn cũng có thể sử dụng `true` và `false` hoặc phụ thêm vào sau biểu tượng với toán tử phủ định.

`operator symbol` là toán tử được sử dụng để ước lượng biểu tượng đó. Các toán tử có thể là một trong các:

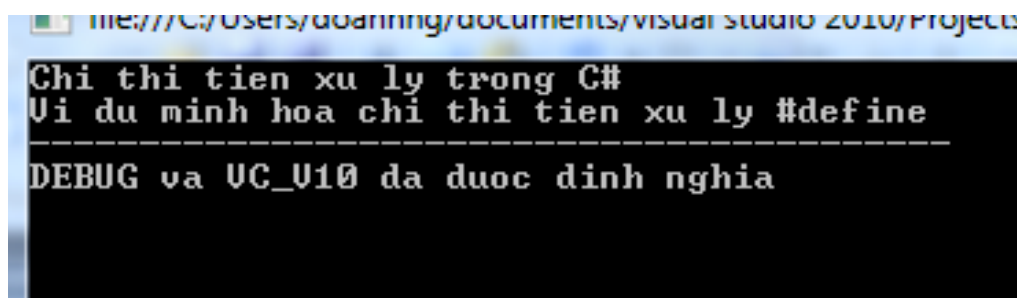
- `==` (bằng)
- `!=` (không bằng)
- `&&` (và)
- `||` (hoặc)

Bạn cũng có thể nhóm các biểu tượng và toán tử bởi các dấu ngoặc đơn. Các chỉ thị có điều kiện được sử dụng để biên dịch code cho debug hoặc khi biên dịch cho một sự định cấu hình cụ thể. Một chỉ thị có điều kiện trong C# bắt đầu với một chỉ thị tiền xử lý #if phải được kết thúc một cách rõ ràng bởi một chỉ thị #endif.

Ví dụ sau minh họa sự sử dụng các chỉ thị có điều kiện trong C#:

```
#define DEBUG
#define VC_V14
using System;
namespace DaiHocTINK8
{
    class TestClass
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Chỉ thị tiền xử lý trong C#");
            Console.WriteLine("Ví dụ minh họa chỉ thị tiền xử lý #define");
            Console.WriteLine("-----");
            #if (DEBUG && !VC_V10)
            Console.WriteLine("DEBUG đã được định nghĩa");
            #elif (!DEBUG && VC_V10)
            Console.WriteLine("VC_V10 đã được định nghĩa");
            #elif (DEBUG && VC_V10)
            Console.WriteLine("DEBUG và VC_V10 đã được định nghĩa");
            #else
            Console.WriteLine("DEBUG và VC_V10 chưa được định nghĩa");
            #endif
            Console.ReadKey();
        }
    }
}
```

Biên dịch và chạy chương trình C# trên sẽ cho kết quả sau:



```
Chỉ thị tiền xử lý trong C#
Ví dụ minh họa chỉ thị tiền xử lý #define
-----
DEBUG và VC_V10 đã được định nghĩa
```

## 2.7 Bài tập cuối chương

**Bài tập 1.** Viết chương trình nhập vào 2 số, tính tổng, hiệu, tích thương của 2 số.

**Bài tập 2.** Viết chương trình tạo ra một không gian tên, có một lớp nhập vào chiều cao, chiều rộng, chiều dài của hình hộp chữ nhật. Hiển thị diện tích xung quanh, toàn phần.



Sử dụng không gian tên trên, và tính thêm thể tích của hình hộp chữ nhật.

**Bài tập 3.** Viết chương trình nhập chiều cao, bán kính của hình trụ, tính diện tích xung quanh, thể tích của hình Trụ.

**Bài tập 4.** Viết chương trình sử dụng kiểu liệt kê và switch case để khi nhập 1 số bất kỳ, hiển thị trên màn hình cách đọc số đó. Ví dụ: năm sáu (56)

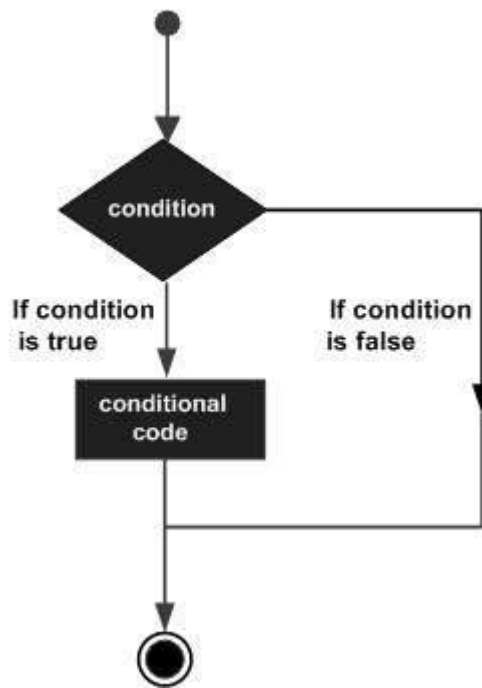
---

## Chương 3: CÁC CẤU TRÚC ĐIỀU KHIỂN

### 3.1. Cấu trúc rẽ nhánh

Các cấu trúc điều khiển luồng yêu cầu lập trình viên xác định một hoặc nhiều điều kiện để được đánh giá và kiểm tra bởi chương trình, cùng với các lệnh được thực hiện nếu điều kiện được xác định là đúng, hoặc các lệnh khác được thực hiện nếu điều kiện xác định là sai.

Dưới đây là mẫu chung của một cấu trúc điều khiển luồng hay gặp trong ngôn ngữ lập trình.



### Lệnh if trong C#

Cú pháp

Sau đây là cú pháp của một lệnh if trong C#:

```
if(biểu_thức)
{
    /* các lệnh được thực thi nếu biểu_thức là true */
}
```

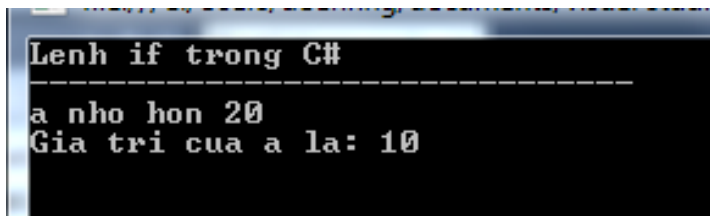
Nếu biểu thức logic được ước lượng là true, thì sau đó khối code bên trong lệnh if sẽ được thực thi. Nếu biểu thức logic được ước lượng là false, thì khi đó, lệnh ngay sau lệnh if sẽ được thực thi.

```

using System;
namespace KhoaCNTTDaihocThanhDo
{
    class DaiHocTinK8
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Lenh if trong C#");
            Console.WriteLine("-----");
            /* phan dinh nghĩa biến cục bộ */
            int a = 10;
            /* kiểm tra điều kiện của biểu thức trong lệnh if*/
            if (a < 20)
            {
                /* nếu điều kiện là true thì sẽ in ra dòng sau: */
                Console.WriteLine("a nhỏ hơn 20");
            }
            Console.WriteLine("Giá trị của a là: {0}", a);
            Console.ReadLine();
            Console.ReadKey();
        }
    }
}

```

Kết quả của chương trình là:



```

Lenh if trong C#
-----
a nhỏ hơn 20
Giá trị của a là: 10

```

### Lệnh if...else trong C#

```

if(biểu_thức)
{
    /* các lệnh được thực thi nếu biểu thức là true */
}
else
{
    /* các lệnh được thực thi nếu biểu thức là false */
}

```

Ví dụ.

```

using System;

```

```

namespace KhoaTinThanhDo
{
    class DaiHocTinK8
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Lenh if...else trong C#");
            Console.WriteLine("-----");
            /* phan dinh nghia bien cuc bo */
            int a = 100;
            /* kiem tra dieu kien */
            if (a > 20)
            {
                /* neu dieu kien la true thi in dong sau: */
                Console.WriteLine("a nho hon 20");
            }
            else
            {
                /* neu dieu kien la false thi in dong sau: */
                Console.WriteLine("a khong nho hon 20");
            }
            Console.WriteLine("Gia tri cua a la: {0}", a);
            Console.ReadLine();
            Console.ReadKey();
        }
    }
}

```

### Lệnh if...else if...else trong C#

Một lệnh if có thể được theo sau bởi một lệnh else if...else tùy ý, mà rất có ích để kiểm tra các điều kiện đa dạng.

Khi sử dụng lệnh if, else if, else, có một số điểm chính cần ghi nhớ:

- Một lệnh if có 0 hoặc một lệnh else và chúng phải theo sau bởi bất kỳ lệnh else if nào. Một lệnh if có 0 tới nhiều lệnh else if và chúng phải ở trước lệnh else.
- Một khi lệnh else if thực hiện, nó sẽ không kiểm tra lại bất kỳ lệnh else if hoặc lệnh else còn lại khác.

### Cú pháp

Cú pháp của một lệnh if...else if...else trong C# là:

```

if(biểu_thức 1)
{
    /* các lệnh được thực thi nếu biểu thức 1 là true */
}
else if( biểu_thức 2)

```

```

{
    /* các lệnh được thực thi nếu biểu thức 2 là true */
}
else if( biểu_thức 3)
{
    /* các lệnh được thực thi nếu biểu thức 3 là true */
}
else
{
    /* các lệnh được thực thi nếu không có biểu thức nào là true */
}

```

Ví dụ:

```

using System;
namespace KhoaCNTT
{
    class TestCsharp
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Lenh if...else trong C#");
            Console.WriteLine("-----");
            /* phan dinh nghia bien cuc bo */
            int a = 100;
            /* kiem tra dieu kien */
            if (a == 10)
            {
                /* neu dieu kien la true thi in dong sau: */
                Console.WriteLine("Gia tri cua a la 10");
            }
            else if (a == 20)
            {
                /* neu dieu kien cua else if nay la true */
                Console.WriteLine("Gia tri cua a la 20");
            }
            else if (a == 30)
            {
                /* neu dieu kien cua else if nay la true */
                Console.WriteLine("Gia tri cua a la 30");
            }
            else
            {
                /* neu khong co dieu kien nao o tren la true */
                Console.WriteLine("Khong co dieu kien nao o tren la true");
            }
            Console.WriteLine("Gia tri chinh xac cua a la: {0}", a);
            Console.ReadLine();
            Console.ReadKey();
        }
    }
}

```

```
}  
}
```

Nó là hợp lệ để lồng các lệnh **if-else** trong C#, nghĩa là bạn có thể sử dụng một lệnh if hoặc else bên trong lệnh if hoặc else khác.

## Cú pháp

Cú pháp để lồng các lệnh if trong C# là như sau:

```
if( biểu_thức 1)  
{  
    /* các lệnh được thực thi nếu biểu thức 1 là true */  
    if(biểu_thức 2)  
    {  
        /* các lệnh được thực thi nếu biểu thức 2 là true */  
    }  
}
```

Bạn có thể lồng **else if...else** theo cách tương tự như bạn đã lồng lệnh if.

Ví dụ:

```
using System;  
namespace KhoaTINThanhDo  
{  
    class TestCsharp  
    {  
        static void Main(string[] args)  
        {  
            Console.WriteLine("Long cac lenh if trong C#");  
            Console.WriteLine("-----");  
            /* phan dinh nghia bien cuc bo */  
            int a = 100;  
            int b = 200;  
            /* kiem tra dieu kien */  
            if (a == 100)  
            {  
                /* neu dieu kien la true, tiep tục kiem tra: */  
                if (b == 200)  
                {  
                    /* neu dieu kien la true thi in dòng sau: */  
                    Console.WriteLine("Gia tri của a là 100 và b là 200");  
                }  
            }  
            Console.WriteLine("Gia tri chính xác của a là: {0}", a);  
            Console.WriteLine("Gia tri chính xác của b là : {0}", b);  
            Console.ReadLine();  
            Console.ReadKey();  
        }  
    }  
}
```

```
    }  
    }  
}
```

## Lệnh switch case trong C#

Một lệnh switch trong C# cho một biến được kiểm tra một cách bình đẳng trong danh sách các giá trị. Mỗi giá trị được gọi là một case - trường hợp và biến được chuyển tới được kiểm tra cho mỗi trường hợp switch.

### Cú pháp

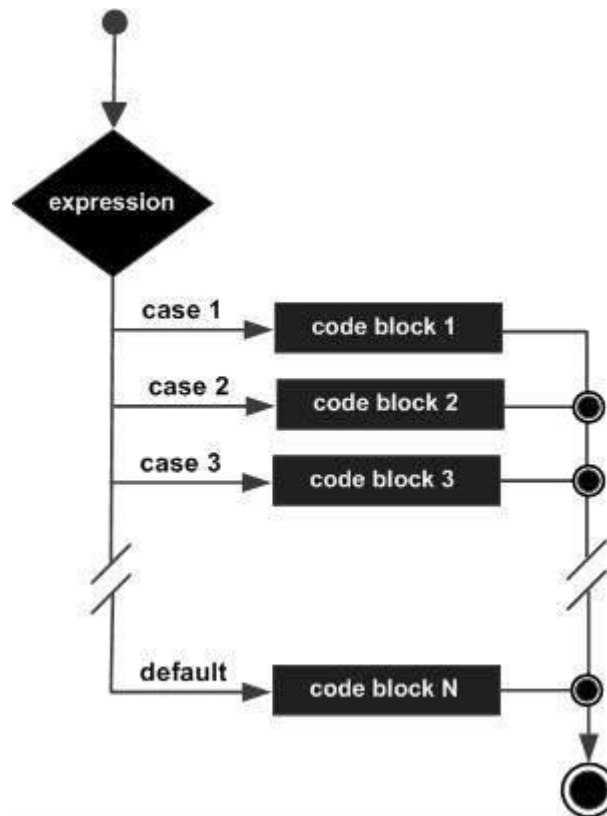
Cú pháp của lệnh switch trong C# như sau:

```
switch(biểu_thức) {  
    case biểu_thức_hằng :  
        các_lệnh_cần_thực_thi;  
        break; /* tùy ý */  
    case biểu_thức_hằng :  
        các_lệnh_cần_thực_thi;  
        break; /* tùy ý */  
    /* số lượng lệnh case là tùy theo bạn */  
    default : /* tùy ý */  
        các_lệnh_cần_thực_thi;  
}
```

Các quy tắc sau được áp dụng tới một lệnh switch:

- Biểu thức biểu\_thức được sử dụng trong một lệnh switch phải có kiểu là integer hoặc liệt kê, hoặc là một trong các kiểu lớp trong đó lớp có một hàm biến đổi đơn tới một kiểu integer hoặc kiểu liệt kê.
- Bạn có thể có bất kỳ số lệnh case nào trong một switch. Mỗi case được theo sau bởi giá trị để được so sánh và một dấu hai chấm.
- biểu\_thức\_hằng cho một case phải cùng kiểu dữ liệu với biến trong switch, và nó phải là hằng số.
- Khi biến được chuyển tới là cân bằng với một case, lệnh theo sau case đó sẽ thực thi tới khi gặp lệnh break.
- Khi gặp lệnh break, switch kết thúc, và dòng điều khiển nhảy tới dòng lệnh tiếp theo của lệnh switch đó.
- Không phải mỗi case cần chứa một lệnh break. Nếu không có lệnh break nào xuất hiện, dòng điều khiển sẽ không tới được case tiếp theo cho tới khi bắt gặp một lệnh break.

- Một lệnh switch có thể có một case mặc định tùy chọn, mà phải xuất hiện ở cuối cùng của switch. Case mặc định này có thể được sử dụng để thực hiện một nhiệm vụ khi không có case nào true. Trong trường hợp case mặc định này thì không cần lệnh break.



Ví dụ:

```
using System;
namespace TruongDHThanhDo
{
    class KhoaCNTT
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Lenh switch trong C#");
            Console.WriteLine("-----");
            /* phan dinh nghia bien cuc bo */
            char grade = 'B';
            switch (grade)
            {
                case 'A':
                    Console.WriteLine("Xuat sac!");
                    break;
                case 'B':
                case 'C':
                    Console.WriteLine("Gioi");
                    break;
                case 'D':
```



```

        Console.WriteLine("Trung binh");
        break;
    case 'F':
        Console.WriteLine("Hoc lai");
        break;
    default:
        Console.WriteLine("Gia tri khong hop le");
        break;
    }
    Console.WriteLine("Hoc luc cua ban la: {0}", grade);
    Console.ReadLine();
    Console.ReadKey();
}
}
}

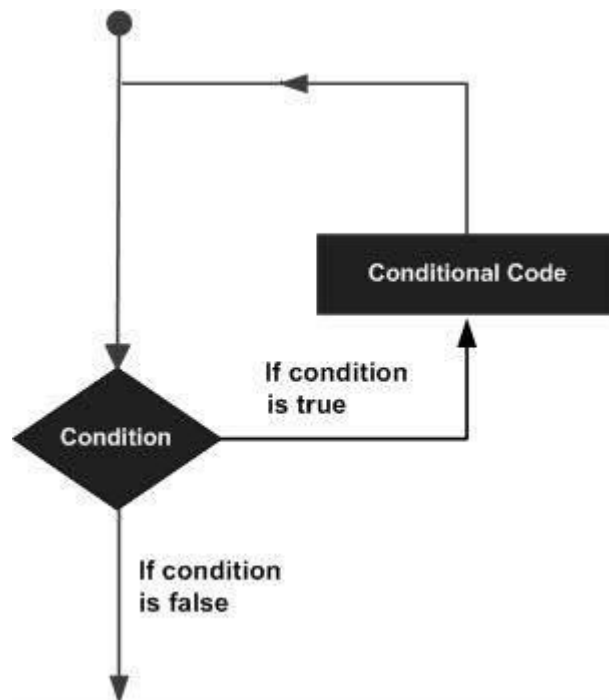
```

### 3.2. Cấu trúc lặp

Có một tình huống mà bạn cần phải thực hiện một đoạn code một vài lần. Nhìn chung, các câu lệnh được thực hiện một cách tuần tự. Câu lệnh đầu tiên của hàm được thực hiện trước, sau đó đến câu thứ 2 và tiếp tục.

Ngôn ngữ lập trình cung cấp cho chúng ta nhiều cấu trúc điều khiển và cho phép bạn thực hiện những phần phức tạp.

Vòng lặp cho phép thực hiện một lệnh và một nhóm lệnh nhiều lần , dưới đây là dạng tổng quát:



#### Vòng lặp while trong C#

---

Một vòng lặp while trong C# thực hiện lặp đi lặp lại một lệnh mục tiêu đến khi nào điều kiện đã cho còn là đúng.

## Cú pháp

Cú pháp của vòng lặp while trong C# là:

```
while(điều_kiện)
{
    statement - các lệnh cần thực thi
}
```

Tại đây, statement có thể là lệnh đơn hoặc một khối các lệnh. điều\_kiện có thể là bất kỳ biểu thức nào, và giá trị true là bất kỳ giá trị nào khác 0. Vòng lặp lặp đi lặp lại trong khi điều\_kiện là true.

Khi điều kiện trở thành false, chương trình điều khiển ngay lập tức chuyển tới dòng lệnh ngay sau vòng lặp.

Ở đây, điểm chính của vòng lặp while là nó có thể không chạy. Bởi vì khi kiểm tra điều kiện và kết quả là false, phần thân vòng lặp được bỏ qua và lệnh đầu tiên ngay sau vòng lặp sẽ được thực thi.

Ví dụ:

```
using System;
namespace KhoaTIN
{
    class TestCsharp
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Vong lap while trong C#");
            Console.WriteLine("-----");
            /* phan dinh nghia bien cuc bo */
            int a = 10;
            /* su thuc thi cua vong lap while */
            while (a < 20)
            {
                Console.WriteLine("Gia tri cua a la: {0}", a);
                a++;
            }
            Console.ReadLine();
            Console.ReadKey();
        }
    }
}
```

**Vòng lặp do...while**

Không giống như các vòng lặp for và while, mà kiểm tra điều kiện vòng lặp ở ngay bước đầu tiên của vòng lặp, vòng lặp do...while trong Ngôn ngữ C# kiểm tra điều kiện của nó tại phần cuối của vòng lặp.

Một vòng lặp do...while là tương tự như vòng lặp while, ngoại trừ ở điểm một vòng lặp do...while bảo đảm thực hiện vòng lặp ít nhất một lần.

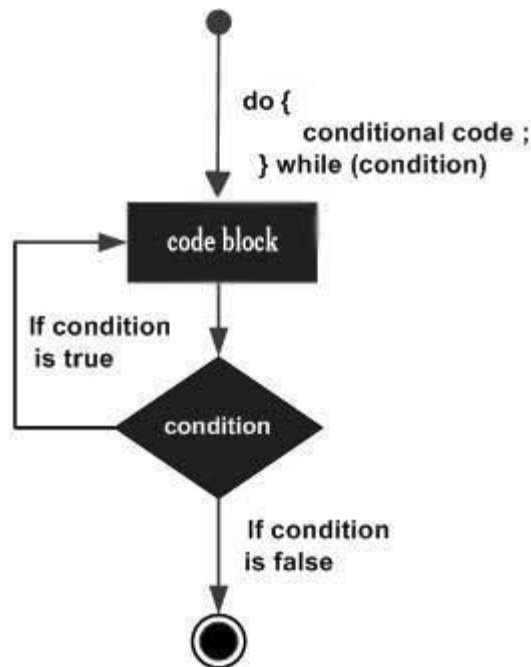
### Cú pháp

Cú pháp của một vòng lặp do...while trong C# là:

```
do
{
    các lệnh được thực thi
}while( điều_kiện );
```

Bạn chú ý rằng, biểu thức điều kiện xuất hiện ở cuối cùng của vòng lặp, vì thế các lệnh trong vòng lặp thực hiện một lần trước khi điều kiện được kiểm tra.

Nếu điều kiện là true, dòng điều khiển vòng lặp quay trở lại, và các lệnh trong vòng lặp được thực hiện lần nữa. Tiến trình này lặp đi lặp lại tới khi nào điều kiện đã cho trở thành false.



Ví dụ.

```
namespace KhoaTinCSharp
{
    class TestCsharp
```

```

{
    static void Main(string[] args)
    {
        Console.WriteLine("Vòng lặp do...while trong C#");
        Console.WriteLine("-----");
        /* phân định nghĩa biến cục bộ */
        int a = 10;
        /* su thực thi vòng lặp do...while */
        do
        {
            Console.WriteLine("Giá trị của a là: {0}", a);
            a = a + 1;
        }
        while (a < 20);
        Console.ReadLine();
        Console.ReadKey();
    }
}

```

## Vòng lặp for trong C#

Vòng lặp for trong C# là một cấu trúc điều khiển lặp đi lặp lại mà cho phép bạn viết một vòng lặp một cách hiệu quả, mà cần thực hiện trong một khoảng thời gian cụ thể nào đó.

### Cú pháp

Cú pháp của một vòng lặp for trong Ngôn ngữ chương trình C# là:

```

for ( khởi_tạo_biến_vòng_lặp; điều_kiện; tăng_giảm_biến_vòng_lặp )
{
    các_lệnh_được_thực_thi;
}

```

Dưới đây là miêu tả dòng điều khiển trong một vòng lặp for:

- Bước khởi\_tạo\_biến\_vòng\_lặp được thực hiện đầu tiên và chỉ một lần. Bước này cho phép bạn khai báo và khởi tạo bất kỳ biến điều khiển vòng lặp nào. Bạn không được yêu cầu để đặt một lệnh ở đây, miễn là một dấu hai chấm xuất hiện.
- Tiếp theo, điều\_kiện được ước lượng. Nếu điều kiện là true, phần thân vòng lặp được thực thi. Nếu nó là false, phần thân vòng lặp không được thực thi và dòng điều khiển nhảy tới lệnh tiếp theo ngay sau vòng lặp for.
- Sau khi phần thân vòng lặp for thực thi, dòng điều khiển nhảy tới lệnh tăng\_giảm\_biến\_vòng\_lặp. Lệnh này cho phép bạn cập nhật bất kỳ biến điều khiển vòng lặp nào. Lệnh này có thể để trống, miễn là một dấu hai chấm xuất hiện sau điều kiện.

- điều\_kiện bây giờ được ước lượng lần nữa. Nếu là true, vòng lặp thực thi và tiến trình lặp đi lặp lại chính nó (phần thân vòng lặp, sau đó là tăng\_giảm\_biến\_vòng\_lặp, và sau đó kiểm tra điều kiện lần nữa). Sau khi điều kiện trở thành false, vòng lặp for kết thúc.

```
using System;
namespace KhoaCNTT
{
    class TestCsharp
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Vong lap for trong C#");
            Console.WriteLine("-----");
            //su thuc thi vong lap for
            for (int a = 10; a < 20; a = a + 1)
            {
                Console.WriteLine("Gia tri cua a la: {0}", a);
            }
            Console.ReadLine();
            Console.ReadKey();
        }
    }
}
```

### Lồng vòng lặp trong C#

C# cho phép bạn sử dụng một vòng lặp bên trong một vòng lặp. Dưới đây là một số ví dụ minh họa khái niệm này.

#### Cú pháp

Cú pháp để lồng vòng lặp for trong C# như sau:

```
for ( khởi_tạo_biến_vòng_lặp; điều_kiện; tăng_giảm_biến_vòng_lặp )
{
    for ( khởi_tạo_biến_vòng_lặp; điều_kiện; tăng_giảm_biến_vòng_lặp )
    {
        các_lệnh_được_thực_thi
    }
    các_lệnh_được_thực_thi
}
```

Cú pháp để lồng vòng lặp while trong C# như sau:

```
while(điều_kiện)
{
    các_lệnh_được_thực_thi
}
```

```

while(điều_kiện)
{
    các lệnh được thực thi
}

các lệnh được thực thi
}

```

Cú pháp để lồng vòng lặp do...while trong C# như sau:

```

do
{
    các lệnh được thực thi
    do
    {
        các lệnh được thực thi
    }
    while( điều_kiện );
}
while( điều_kiện );

```

Ghi chú cuối cùng về lồng vòng lặp là bạn có thể đặt bất kỳ kiểu vòng lặp bên trong kiểu vòng lặp khác. Ví dụ, một vòng lặp for có thể bên trong một vòng lặp while, và ngược lại.

## Ví dụ

Chương trình sau sử dụng lồng vòng lặp for để tìm các số nguyên tố từ 2 đến 100:

```

using System;
namespace KhoaTIN
{
    class TestCsharp
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Long vong lap trong C#");
            Console.WriteLine("Tim so nguyen to trong C#");
            Console.WriteLine("-----");
            /*phan dinh nghia bien cuc bo */
            int i, j;
            for (i = 2; i < 100; i++)
            {
                for (j = 2; j <= (i / j); j++)
                    if ((i % j) == 0) break; // neu tim thay uoc so thi khong phai la
so nguyen to
            }
        }
    }
}

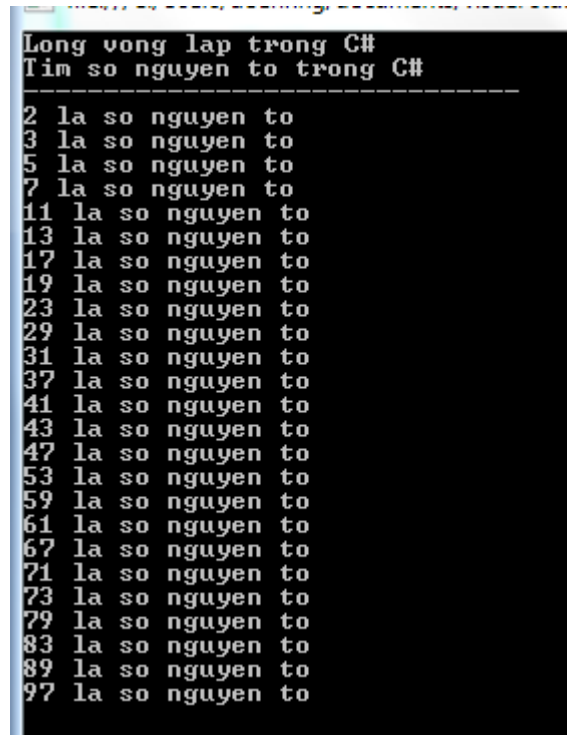
```

```

        if (j > (i / j))
            Console.WriteLine("{0} la so nguyen to", i);
    }
    Console.ReadLine();
    Console.ReadKey();
}
}
}

```

Kết quả



```

Long vong lap trong C#
Tim so nguyen to trong C#
-----
2 la so nguyen to
3 la so nguyen to
5 la so nguyen to
7 la so nguyen to
11 la so nguyen to
13 la so nguyen to
17 la so nguyen to
19 la so nguyen to
23 la so nguyen to
29 la so nguyen to
31 la so nguyen to
37 la so nguyen to
41 la so nguyen to
43 la so nguyen to
47 la so nguyen to
53 la so nguyen to
59 la so nguyen to
61 la so nguyen to
67 la so nguyen to
71 la so nguyen to
73 la so nguyen to
79 la so nguyen to
83 la so nguyen to
89 la so nguyen to
97 la so nguyen to

```

### 3.3. Nhãn

### 3.4. Bẫy lỗi và xử lý lỗi hay Xử lý ngoại lệ.

Một Exception (ngoại lệ) là một vấn đề xuất hiện trong khi thực thi một chương trình. Một Exception trong C# là một phản hồi về một tình huống ngoại lệ mà xuất hiện trong khi một chương trình đang chạy, ví dụ như chia cho số 0.

Exception cung cấp một cách để truyền điều khiển từ một phần của một chương trình tới phần khác. Exception Handling (Xử lý ngoại lệ) trong C# được xây dựng dựa trên 4 từ khóa là: try, catch, finally, và throw.

**try:** Một khối try nhận diện một khối code mà ở đó các exception cụ thể được kích hoạt. Nó được theo sau bởi một hoặc nhiều khối catch.

**catch:** Một chương trình bắt một Exception với một Exception Handler tại vị trí trong một chương trình nơi bạn muốn xử lý vấn đề đó. Từ khóa catch trong C# chỉ dẫn việc bắt một exception.

---

**finally:** Một khối finally được sử dụng để thực thi một tập hợp lệnh đã cho, dù có hay không một exception được ném hoặc không được ném. Ví dụ, nếu bạn mở một file, nó phải được đóng, nếu không sẽ có một exception được tạo ra.

**throw:** Một chương trình ném một exception khi có một vấn đề xuất hiện. Điều này được thực hiện bởi sử dụng từ khóa throw trong C#.

### Cú pháp

Giả sử một khối tạo một Exception, một phương thức bắt một exception bởi sử dụng kết hợp các từ khóa try và catch. Một khối try/catch được đặt xung quanh code mà có thể tạo một exception. Code bên trong một khối try/catch được xem như là code được bảo vệ, và cú pháp để sử dụng try/catch trong C# như sau:

```
try
{
    // các lệnh có thể gây ra ngoại lệ (exception)
}
catch( tênngoạilệ e1 )
{
    // phần code để xử lý lỗi
}
catch( tênngoạilệ e2 )
{
    // phần code để xử lý lỗi
}
catch( tênngoạilệ eN )
{
    // phần code để xử lý lỗi
}
finally
{
    // các lệnh được thực thi
}
```

Bạn có thể liệt kê nhiều lệnh catch để bắt các kiểu exception khác nhau trong trường hợp khối try của bạn xuất hiện nhiều hơn một exception trong các tình huống khác nhau.

### Lớp Exception trong C#



Các Exception trong C# được biểu diễn bởi các lớp. Các lớp Exception trong C# chủ yếu được kế thừa một cách trực tiếp hoặc không trực tiếp từ lớp System.Exception trong C#. Một số lớp Exception kế thừa từ lớp System.Exception là các lớp System.ApplicationException và System.SystemException.

Lớp System.ApplicationException hỗ trợ các exception được tạo bởi các chương trình ứng dụng. Vì thế, các exception được định nghĩa bởi lập trình viên nên kế thừa từ lớp này.

Lớp System.SystemException là lớp cơ sở cho tất cả system exception tiền định nghĩa.

Bảng sau cung cấp một số lớp Exception tiền định nghĩa được kế thừa từ lớp Sytem.SystemException trong C#:

Lớp Exception	Miêu tả
System.IO.IOException	Xử lý các I/O error
System.IndexOutOfRangeException	Xử lý các lỗi được tạo khi một phương thức tham chiếu tới một chỉ mục bên ngoài dãy mảng
System.ArrayTypeMismatchException	Xử lý các lỗi được tạo khi kiểu là không phù hợp với kiểu mảng
System.NullReferenceException	Xử lý các lỗi được tạo từ việc tham chiếu một đối tượng null
System.DivideByZeroException	Xử lý các lỗi được tạo khi chia cho số 0
System.InvalidCastException	Xử lý lỗi được tạo trong khi ép kiểu
System.OutOfMemoryException	Xử lý lỗi được tạo từ việc thiếu bộ nhớ rồi
System.StackOverflowException	Xử lý lỗi được tạo từ việc tràn ngăn xếp (stack)

### **Xử lý ngoại lệ ( Exception Handling) trong C#**

C# cung cấp một giải pháp mang tính cấu trúc cao để xử lý ngoại lệ trong form các khối try và catch. Sử dụng các khối này, các lệnh chương trình được phân biệt riêng rẽ với các lệnh xử lý ngoại lệ trong C#.

Những khối xử lý ngoại lệ này được triển khai bởi sử dụng các từ khóa try, catch và finally trong C#. Ví dụ sau ném một exception khi chia cho số 0.

```
using System;
namespace KhoaCNTT
{
    class TestCsharp
    {
        int result;
        TestCsharp()
        {
            result = 0;
        }
        public void phepChia(int num1, int num2)
        {
            try
            {
                result = num1 / num2;
            }
            catch (DivideByZeroException e)
            {
                Console.WriteLine("Bat Exception: {0}", e);
            }
            finally
            {
                Console.WriteLine("Ket qua: {0}", result);
            }
        }
        static void Main(string[] args)
        {
            Console.WriteLine("Vi du minh hoa Exception trong C#");
            Console.WriteLine("-----");
            TestCsharp d = new TestCsharp();
            d.phepChia(25, 0);
            Console.ReadKey();
        }
    }
}
```

### 3.5 Bài tập cuối chương.

**Bài tập 1.** Nhập một nhiệt độ và in thông báo tương ứng trong C#, Hiển thị thông báo theo các tiêu chí sau:

```
Nhiệt độ < 0 thì thông báo là thời tiết giá rét
Nhiệt độ 0-10: thời tiết rất lạnh
Nhiệt độ 10-20: thời tiết lạnh
```

Nhiệt độ 20-30: thời tiết bình thường

Nhiệt độ 30-40: thời tiết nóng

Nhiệt độ  $\geq 40$ : thời tiết rất nóng

**Bài tập 2.** Sử dụng câu lệnh rẽ nhánh để giải các bài tập sau:

1. Nhập hai số và kiểm tra xem hai số đó có bằng nhau không trong C#
2. Kiểm tra một số là chẵn hay lẻ trong C#.
3. Chương trình C# để kiểm tra số dương, số âm
4. Kiểm tra năm nhuận trong C#
5. Chương trình C# để nhập tuổi của một người, kiểm tra và thông báo xem người đó có đủ độ tuổi bầu cử không (tuổi bầu cử là 18)
6. Chương trình C# để nhập một số m, kiểm tra xem số đó là lớn hơn, nhỏ hơn hoặc bằng 0 và in ra giá trị tương ứng của một số n là 1, -1 hoặc 0
7. Chương trình C# để nhập chiều cao của một người và sau đó phân loại chiều cao của người đó
8. Tìm số lớn nhất trong C#
9. Chương trình C# để nhập tọa độ (x, y) của một điểm và thông báo góc phần tư.
10. Chương trình C# để nhập điểm thi ba môn Toán, Lý, Hóa của một thí sinh, kiểm tra xem thí sinh đó có trúng tuyển không
11. Giải phương trình bậc hai trong C#
12. Chương trình C# để nhập nhiệt độ và in ra thông báo tương ứng
13. Kiểm tra tam giác đều, cân, lệch trong C#
14. Chương trình C# để nhập 3 số, và kiểm tra xem 3 số này có phải là 3 góc của tam giác không
15. Kiểm tra nguyên âm, phụ âm trong C#
16. Chương trình C# để kiểm tra lợi nhuận hoặc thua lỗ
17. Chương trình C# để nhập hạng (ví dụ: A, B, C, ...) và hiển thị miêu tả tương ứng
18. Chương trình C# để nhập một số và hiển thị ngày trong tuần tương ứng
19. Nhập một số bất kỳ và hiển thị số bằng chữ tương ứng trong C#
20. Nhập số tháng bất kỳ và hiển thị số ngày của tháng đó trong C#
21. Viết một chương trình C# hiển thị một menu có các lựa chọn để tính diện tích các hình tròn, hình chữ nhật, hình tam giác tương ứng với dữ liệu đã nhập.
22. Viết một chương trình C# hiển thị một menu có các lựa chọn để thực hiện các phép toán cơ bản của hai số a, b

**Bài tập 3.** Sử dụng vòng lặp trong C# để viết chương trình giải các bài sau:

1. Tính tổng dãy số trong C#
2. Chương trình C# để hiển thị các số Hex có giá trị từ 0 - 255
3. In bảng cửu chương trong C#
4. Kiểm tra số nguyên tố trong C#
5. Tìm số nguyên tố trong C#
6. Chương trình C# để in bảng nhân của một số bất kỳ
7. Hiển thị và tính tổng các số lẻ trong C#

- 
8. Hiển thị và tính tổng các số chẵn trong C#
  9. Vẽ tam giác sao trong C#
  10. Vẽ tam giác số trong C# (I)
  11. Vẽ tam giác số trong C# (II)
  12. Vẽ tam giác số trong C# (III)
  13. Vẽ tam giác số trong C# (IV)
  14. Vẽ tam giác số trong C# (V)
  15. Vẽ tam giác sao đều trong C#
  16. Vẽ tam giác sao cân trong C#
  17. Vẽ hình kim cương bằng cách dấu sao trong C#
  18. Tìm giai thừa của một số trong C#
  19. Chương trình C# để tính tổng dãy số:  $1 - x^2/2! + x^4/4! - \dots$
  20. Chương trình C# để tính tổng dãy số:  $1/1 + 1/2 + 1/3 + 1/4 + \dots$
  21. Tính tổng dãy số  $9 + 99 + 999 + \dots$  trong C#
  22. Kiểm tra số hoàn hảo trong C#, trong đó số hoàn thiện (hay còn gọi là số hoàn chỉnh, số hoàn hảo hoặc số hoàn thành) là một số nguyên dương mà tổng các ước nguyên dương của nó (số nguyên dương chia hết cho nó) bằng chính nó.
  23. Chương trình C# để tìm số hoàn hảo trong một dãy đã cho
  24. Kiểm tra số armstrong trong C#
  25. Chương trình C# để tìm số armstrong trong một dãy số đã cho
  26. Vẽ tam giác Pascal trong C#
  27. In dãy Fibonacci trong C#
  28. Chương trình C# để vẽ một hình tam giác số theo mẫu đã cho
  29. In số theo chiều đảo ngược trong C#
  30. Kiểm tra số đối xứng (số Palindrome) trong C#
  31. Vẽ tam giác chữ cái trong C#
  32. Chương trình C# để tìm ước số chung lớn nhất (USCLN)
  33. Tìm bội số chung nhỏ nhất (BSCNN) trong C# (sử dụng USCLN)
  34. Tìm bội số chung nhỏ nhất (BSCNN) trong C# (không sử dụng USCLN)
  35. Chương trình C# để kiểm tra Strong Number
  36. Tìm Strong Number trong C#
  37. Chương trình C# để kiểm tra xem một số bất kỳ có thể biểu diễn bằng tổng của hai số nguyên tố không
  38. In chuỗi theo chiều đảo ngược trong C#
  39. Chuyển đổi số nhị phân
    - a. Chuyển đổi nhị phân thành thập phân trong C#
    - b. Chuyển đổi nhị phân thành thập phân bằng cách sử dụng hàm trong C#
    - c. Chương trình C# để chuyển đổi hệ thập phân thành hệ bát phân
    - d. Chương trình C# để chuyển đổi hệ thập phân thành hệ nhị phân
    - e. Chuyển đổi bát phân thành thập phân trong C#
    - f. Chương trình C# để chuyển đổi nhị phân thành bát phân
    - g. Chuyển đổi bát phân thành nhị phân trong C#
    - h. Chương trình C# để chuyển đổi thập phân thành thập lục phân

**Bài tập 4.** Viết chương trình nhập vào năm sinh (ví dụ 1998) hiển thị năm đó là năm gì theo địa chi (năm Dần).

**Bài tập 5.** Viết chương trình nhập vào năm sinh (ví dụ 1998) hiển thị năm đó là năm gì theo thiên can địa chi (Mậu Dần).

*Biết rằng danh sách thiên can theo thứ tự: Giáp, Ất, Bính, Đinh, Mậu, Kỷ, Canh, Tân, Nhâm, Quý. Và danh sách địa chi theo thứ tự: Tý, Sửu, Dần, Mão, Thìn, Tỵ, Ngọ, Mùi, Thân, Dậu, Tuất, Hợi.*

---

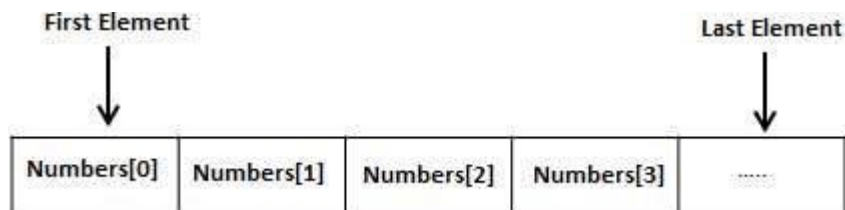
## Chương 4: MẢNG, CHỈ MỤC VÀ TẬP HỢP

### 4.1. Mảng

Một mảng lưu giữ một tập hợp các phần tử có kích cỡ cố định trong cùng kiểu. Một mảng được sử dụng để lưu giữ một tập hợp dữ liệu, nhưng nó thường hữu ích hơn khi nghĩ về một mảng như là một tập hợp các biến cùng kiểu được lưu giữ tại các vị trí bộ nhớ kế nhau.

Thay vì khai báo biến một cách rời rạc, như biến `number0`, `number1`,... và `number99`, bạn có thể khai báo một mảng các giá trị như `numbers[0]`, `numbers[1]` và ... `numbers[99]` để biểu diễn các giá trị riêng biệt. Một thành viên cụ thể của mảng có thể được truy cập qua index (chỉ số).

Tất cả mảng đều bao gồm các vị trí nhớ liên kế nhau. Địa chỉ thấp nhất tương ứng với thành viên đầu tiên và địa chỉ cao nhất tương ứng với thành viên cuối cùng của mảng.



### Khai báo mảng trong C#

Để khai báo một mảng trong ngôn ngữ C#, bạn có thể sử dụng cú pháp:

```
kiểu_dữ_liệu[] tên_mảng;
```

Tại đây:

- `kiểu_dữ_liệu` được sử dụng để xác định kiểu của phần tử trong mảng.
- `[ ]` xác định rank hay kích cỡ của mảng.
- `tên_mảng` xác định tên mảng.

Ví dụ:

```
double[] balance;
```

### Khởi tạo mảng trong C#

Việc khai báo một mảng không khởi tạo mảng trong bộ nhớ. Khi biến mảng được khởi tạo, bạn có thể gán giá trị cho mảng đó.

Mảng là một kiểu tham chiếu, vì thế bạn cần sử dụng từ khóa `new` trong C# để tạo một Instance (sự thể hiện) của mảng đó. Ví dụ:

```
double[] balance = new double[10];
```

Gán giá trị cho một mảng trong C#

Bạn có thể gán giá trị cho các phần tử mảng riêng biệt bởi sử dụng chỉ số mảng, như:

```
double[] balance = new double[10];  
balance[0] = 4500.0;
```

Bạn có thể gán giá trị cho mảng tại thời điểm khai báo mảng, như sau:

```
double[] balance = { 2340.0, 4523.69, 3421.0};
```

Bạn cũng có thể tạo và khai báo một mảng, như sau:

```
int [] marks = new int[5] { 99, 98, 92, 97, 95};
```

Bạn cũng có thể bỏ qua kích cỡ mảng, như:

```
int [] marks = new int[] { 99, 98, 92, 97, 95};
```

Bạn có thể sao chép một biến mảng vào trong biến mảng mục tiêu khác. Trong tình huống này, cả biến mục tiêu và biến nguồn đều trở tới cùng vị trí bộ nhớ:

```
int [] marks = new int[] { 99, 98, 92, 97, 95};  
int[] score = marks;
```

Khi bạn tạo một mảng, C# compiler ngầm định khởi tạo mỗi phần tử mảng thành một giá trị mặc định phụ thuộc vào kiểu mảng. Ví dụ, với một mảng int, thì tất cả phần tử được khởi tạo là 0.

C# hỗ trợ các mảng đa chiều. Các mảng đa chiều cũng được gọi là mảng hình chữ nhật. Bạn có thể khai báo một mảng chuỗi hai chiều như sau:

```
string [,] names;
```

Hoặc, một mảng các biến int 3 chiều, như sau:

```
int [ , , ] m;
```

## Mảng hai chiều trong C#

Mẫu đơn giản nhất của mảng đa chiều là mảng hai chiều. Một mảng hai chiều về bản chất là danh sách của các mảng một chiều.

Một mảng 2 chiều có thể được nghĩ như là một bảng, có x hàng và y cột. Dưới đây là một mảng hai chiều có 3 hàng và 4 cột.

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

### Khởi tạo mảng hai chiều trong C#

Các mảng đa chiều có thể được khởi tạo bởi xác định các giá trị trong dấu móc vuông cho mỗi hàng. Sau đây là một mảng với 3 hàng và mỗi hàng chứa 4 cột.

```
int [,] a = int [3,4] = {
    {0, 1, 2, 3} ,    /* khởi tạo cho hàng bắt đầu với chỉ mục 0 */
    {4, 5, 6, 7} ,    /* khởi tạo cho hàng bắt đầu với chỉ mục 1 */
    {8, 9, 10, 11}    /* khởi tạo cho hàng bắt đầu với chỉ mục 2 */
};
```

### Jagged Array trong C#

Một Jagged Array trong C# là một mảng của các mảng. Bạn có thể khai báo một Jagged Array với tên *scores* với kiểu **int**, như sau:

```
int [][] scores;
```

Khai báo một mảng, không tạo mảng đó trong bộ nhớ. Để khai báo mảng như vậy:

```
int[][] scores = new int[5][];
for (int i = 0; i < scores.Length; i++)
{
    scores[i] = new int[4];
}
```

Bạn có thể khởi tạo một Jagged Array trong C# như sau:

```
int[][] scores = new int[2][]{new int[]{92,93,94},new int[]{85,66,87,88}};
```

Tại đây, *scores* là một mảng của hai mảng integer: trong đó, *scores[0]* là một mảng 3 integer và *scores[1]* là một mảng 4 integer.

Ví dụ.



```

using System;
namespace KhoaTinCSharp
{
    class TestCsharp
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Jagged Array trong C#");
            Console.WriteLine("-----");
            /* mot jagged array cua 5 mang so nguyen*/
            int[][] a = new int[][] { new int[] { 0, 0 }, new int[] { 1, 2 }, new
int[] { 2, 4 }, new int[] { 3, 6 }, new int[] { 4, 8 } };
            int i, j;
            /* hien thi gia tri cac phan tu trong mang */
            for (i = 0; i < 5; i++)
            {
                for (j = 0; j < 2; j++)
                {
                    Console.WriteLine("Phan tu a[{0}][{1}] = {2}", i, j, a[i][j]);
                }
            }
            Console.ReadKey();
        }
    }
}

```

## Truyền mảng như là các tham số hàm trong C#

Bạn có thể truyền một mảng như là một tham số hàm trong C#. Dưới đây là ví dụ minh họa khái niệm này:

```

using System;
namespace KhoaTinCSharp
{
    class TestCsharp
    {
        double tinhTrungBinh(int[] arr, int size)
        {
            int i;
            double avg;
            int sum = 0;
            for (i = 0; i < size; ++i)
            {
                sum += arr[i];
            }
            avg = (double)sum / size;
            return avg;
        }
        static void Main(string[] args)
        {
            Console.WriteLine("Truyen mang nhu la tham so trong C#");
            Console.WriteLine("-----");
            //tao doi tuong TestCsharp
            TestCsharp app = new TestCsharp();
            /* mot mang gom 5 so nguyen */
        }
    }
}

```

```

        int[] balance = new int[] { 1000, 2, 3, 17, 50 };
        double avg;
        /* truyen con tro toi mang giống như là tham số */
        avg = app.tinhTrungBinh(balance, 5);
        /* hiển thị kết quả */
        Console.WriteLine("Giá trị trung bình bằng: {0} ", avg);
        Console.ReadKey();
    }
}
}

```

## Mảng tham số trong C#

Đôi khi, trong khi khai báo một phương thức, bạn không chắc chắn số tham số được truyền như là một tham số. Mảng tham số (Parameter Array) trong C# giúp giải quyết vấn đề này.

Dưới đây là ví dụ minh họa mảng tham số trong C#: bạn tạo hai lớp có tên lần lượt là MangThamSo và TestCsharp như sau:

Lớp MangThamSo: có chứa phương thức CongPhanTu()

```

using System;
namespace KhoaTinCSharp
{
    class MangThamSo
    {
        public int CongPhanTu(params int[] arr)
        {
            int sum = 0;
            foreach (int i in arr)
            {
                sum += i;
            }
            return sum;
        }
    }
}

```

Lớp TestCsharp: có chứa phương thức main() để thao tác trên đối tượng MangThamSo.

```

using System;
namespace KhoaTinCSharp
{
    class TestCsharp
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Mang tham số trong C#");
            Console.WriteLine("-----");
            //tạo đối tượng MangThamSo
            MangThamSo app = new MangThamSo();
            int sum = app.CongPhanTu(512, 720, 250, 567, 889);
            Console.WriteLine("Tổng bằng: {0}", sum);
            Console.ReadKey();
        }
    }
}

```

```
}
```

## Lớp Array trong C#

Lớp Array trong C# là lớp cơ sở cho tất cả mảng trong C#. Nó được định nghĩa trong không gian tên System. Lớp Array cung cấp nhiều thuộc tính và phương thức đa dạng để làm việc với mảng.

*Thuộc tính của lớp Array trong C#*

Bảng dưới liệt kê một số thuộc tính được sử dụng phổ biến nhất của lớp Array trong C#:

STT	Thuộc tính
1	<b>IsFixedSize</b>  Lấy một giá trị chỉ rằng có hay không Array đó có một kích cỡ cố định
2	<b>IsReadOnly</b>  Lấy một giá trị chỉ rằng có hay không Array đó là read-only
3	<b>Length</b>  Lấy một số integer (32 bit) mà biểu diễn tổng số phần tử trong tất cả các chiều của Array
4	<b>LongLength</b>  Lấy một số integer (64 bit) mà biểu diễn tổng số phần tử trong tất cả các chiều của Array
5	<b>Rank</b>  Lấy số chiều của Array

*Phương thức của lớp Array trong C#*

Bảng dưới liệt kê một số phương thức được sử dụng phổ biến nhất của lớp Array trong C#:

STT	Phương thức
-----	-------------

1	<b>Clear</b>  Thiết lập một dãy các phần tử trong Array về 0, về false hoặc về null, phụ thuộc vào kiểu phần tử
2	<b>Copy(Array, Array, Int32)</b> Sao chép một dãy các phần tử từ một Array bắt đầu từ phần tử đầu tiên và paste chúng vào trong Array khác bắt đầu tại phần tử đầu tiên. Độ dài (length) được xác định là một integer (32 bit)
3	<b>CopyTo(Array, Int32)</b> Sao chép tất cả phần tử của Array một chiều hiện tại tới Array một chiều đã xác định bắt đầu tại chỉ mục mảng đích đến đã cho. Chỉ mục được xác định là một integer (32 bit)
4	<b>GetLength</b> Lấy một số integer (32 bit) mà biểu diễn số phần tử trong chiều đã xác định của Array
5	<b>GetLongLength</b> Lấy một số integer (64 bit) mà biểu diễn số phần tử trong chiều đã xác định của Array
6	<b>GetLowerBound</b> Lấy giới hạn thấp hơn của chiều đã xác định trong Array
7	<b>GetType</b> Lấy kiểu của instance (sự thể hiện) hiện tại (được kế thừa từ Object)
8	<b>GetUpperBound</b> Lấy giới hạn ở trên của chiều đã xác định trong Array
9	<b>GetValue(Int32)</b> Lấy giá trị tại vị trí đã xác định trong mảng một chiều. Chỉ mục được xác định

	là số integer (32)
10	<b>IndexOf(Array, Object)</b> Tìm kiếm object đã cho và trả về chỉ mục về sự xuất hiện đầu tiên bên trong cả mảng một chiều đó
11	<b>Reverse(Array)</b> Đảo ngược dãy phần tử trong cả mảng một chiều đó
12	<b>SetValue(Object, Int32)</b> Thiết lập giá trị cho phần tử tại vị trí đã cho trong mảng một chiều. Chỉ mục được xác định là một số integer (32 bit)
13	<b>Sort(Array)</b> Sắp xếp các phần tử trong cả mảng một chiều bởi sử dụng IComparable implementation của mỗi phần tử của mảng đó
14	<b>ToStringk</b> Trả về một chuỗi mà biểu diễn object hiện tại (được kế thừa từ Object)

Ví dụ.

```
using System;
namespace KhoaTinCSharp
{
    class TestCsharp
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Lop Array trong C#");
            Console.WriteLine("-----");
            int[] list = { 34, 72, 13, 44, 25, 30, 10 };
            int[] temp = list;
            //in cac phan tu trong mang ban dau
            Console.Write("\nMang ban dau: ");
            foreach (int i in list)
            {
                Console.Write(i + " ");
            }
            Console.WriteLine();
            // dao chieu mang
            Array.Reverse(temp);
            Console.Write("\nMang dao chieu: ");
            foreach (int i in temp)
```

```

        {
            Console.Write(i + " ");
        }
        Console.WriteLine();
        //sắp xếp mảng
        Array.Sort(list);
        Console.WriteLine("\nMảng đã qua sắp xếp: ");
        foreach (int i in list)
        {
            Console.Write(i + " ");
        }
        Console.WriteLine();
        Console.ReadKey();
    }
}

```

## 4.2. Chỉ mục

### Truy cập các phần tử mảng trong C#

Một phần tử được truy cập bởi chỉ mục mảng. Điều này được thực hiện bởi việc đặt chỉ số của phần tử bên trong dấu ngoặc vuông ở sau tên mảng. Ví dụ:

```
double salary = balance[9];
```

Ví dụ sau minh họa khái niệm về khai báo, gán và truy cập mảng trong C# đã đề cập ở trên:

```

using System;
namespace KhoaTinCSharp
{
    class TestCsharp
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Mảng trong C#");
            Console.WriteLine("-----");
            int[] n = new int[10]; /* n là một mảng gồm 10 số nguyên */
            int i, j;
            /* khởi tạo các phần tử của mảng n */
            for (i = 0; i < 10; i++)
            {
                n[i] = i + 100;
            }
            /* hiển thị giá trị các phần tử của mảng n */
            for (j = 0; j < 10; j++)
            {
                Console.WriteLine("Phần tử [{0}] = {1}", j, n[j]);
            }
            Console.ReadKey();
        }
    }
}

```

## Sử dụng vòng lặp foreach trong C#

Trong ví dụ trước, chúng ta đã sử dụng một vòng lặp for để truy cập mỗi phần tử trong mảng. Bạn cũng có thể sử dụng một lệnh foreach để duyệt qua một mảng trong C#:

```
using System;
namespace KhoaTinCSharp
{
    class TestCsharp
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Mang trong C#");
            Console.WriteLine("-----");
            int[] n = new int[10]; /* n la mot mang gom 10 so nguyen */
            /* khoi tao cac phan tu trong mang n */
            for (int i = 0; i < 10; i++)
            {
                n[i] = i + 100;
            }
            /* hien thi cac gia tri cua phan tu trong mang n */
            foreach (int j in n)
            {
                int i = j - 100;
                Console.WriteLine("Phan tu [{0}] = {1}", i, j);
                i++;
            }
            Console.ReadKey();
        }
    }
}
```

## Truy cập các phần tử của mảng hai chiều trong C#

Các phần tử mảng hai chiều được truy cập bởi sử dụng các chỉ số, ví dụ chỉ số hàng và chỉ số cột. Ví dụ:

```
int val = a[2,3];
```

Lệnh trên sẽ truy cập vào phần tử thứ 4 từ hàng thứ 3 của mảng. Bạn có thể kiểm tra lại nó trong sơ đồ trên. Bây giờ chúng ta xem xét ví dụ dưới đây, chúng tôi đã sử dụng các vòng lặp lồng vào nhau để xử lý một mảng hai chiều:

```
using System;
namespace KhoaTinCSharp
{
    class TestCsharp
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Mang da chieu trong C#");
            Console.WriteLine("-----");
            /* mot mang hai chieu gom 5 hang va 2 cot*/
            int[,] a = new int[5, 2] { { 0, 0 }, { 1, 2 }, { 2, 4 }, { 3, 6 }, { 4, 8 }
        }
    }
};
```

```

int i, j;
/* hiển thị giá trị các phần tử trong mảng */
for (i = 0; i < 5; i++)
{
    for (j = 0; j < 2; j++)
    {
        Console.WriteLine("Phần tử a[{0},{1}] = {2}", i, j, a[i, j]);
    }
}
Console.ReadKey();
    }
}
}

```

### 4.3. Tập hợp

Các lớp Collection là các lớp đặc biệt để lưu giữ và thu hồi dữ liệu. Những lớp này cung cấp sự hỗ trợ cho Stack, Queue, List, và Hash Table. Đa số các lớp Collection trong C# triển khai cùng các Interface.

Trong C#, các lớp Collection phục vụ các mục đích đa dạng, chẳng hạn như cấp phát bộ nhớ động cho các phần tử và truy cập một danh sách các item dựa trên một chỉ mục, ... Những lớp này tạo tập hợp các đối tượng của lớp Object, mà là lớp cơ sở cho tất cả kiểu dữ liệu trong C#.

Ghi chú: Stack: ngăn xếp, push: thêm nút mới vào đỉnh stack, pop: thao tác lấy 1 phần tử từ đỉnh stack.

#### Các lớp Collection và cách sử dụng của chúng trong C#

Bảng dưới liệt kê các lớp được sử dụng phổ biến của System.Collection namespace. Bạn truy cập link để tìm hiểu chi tiết.

Lớp	Miêu tả và Cách sử dụng
<b><u>ArrayList trong C#</u></b>	<p>Nó biểu diễn một tập hợp được sắp xếp của một đối tượng mà có thể được <b>lập chỉ mục</b> cho từng item riêng rẽ.</p> <p>Về cơ bản, nó là một sự thay thế cho một mảng. Tuy nhiên, không giống như trong mảng, bạn có thể thêm và gỡ bỏ các item từ một list tại một vị trí đã xác định bởi sử dụng một chỉ mục và mảng chính nó có thể tự điều</p>



	<p>chỉnh kích cỡ một cách tự động. Nó cũng cho phép cấp phát bộ nhớ động, thêm, tìm kiếm và sắp xếp các item trong một list.</p>
<p><b><u>Hashtable trong C#</u></b></p>	<p>Nó sử dụng một cặp <b>key-value</b> để truy cập các phần tử trong collection này.</p> <p>Một Hash Table được sử dụng khi bạn cần truy cập các phần tử bởi sử dụng key, và bạn có thể nhận diện một giá trị key hữu ích. Mỗi item trong Hash Table có một cặp <b>key/value</b>. Key được sử dụng để truy cập các item trong dạng collection này.</p>
<p><b><u>SortedList trong C#</u></b></p>	<p>Nó sử dụng một <b>key</b> cũng như một <b>index</b> để truy cập các item trong một list.</p> <p>Một danh sách đã được sắp xếp là sự tổ hợp của một mảng và một Hash Table. Nó chứa một danh sách các item mà có thể được truy cập bởi sử dụng một key hoặc một chỉ mục. Nếu bạn truy cập item bởi sử dụng một chỉ mục, nó là một ArrayList, và nếu bạn truy cập item bởi sử dụng key, nó là một Hashtable. Tập hợp các item luôn luôn được sắp xếp bởi giá trị key</p>
<p><b><u>Stack trong C#</u></b></p>	<p>Nó biểu diễn một tập hợp <b>Last-in, First-out</b> của các đối tượng.</p> <p>Nó được sử dụng khi bạn cần truy cập các item theo dạng Last-in, First-out. Khi bạn thêm một item vào trong danh sách, nó được gọi là <b>pushing</b> và khi bạn gỡ bỏ một item, nó</p>

	được gọi là <b>popping</b> .
<p><b><u>Queue trong C#</u></b></p> <p>Nó biểu diễn một tập hợp <b>First-in, First-out</b> của các đối tượng.</p> <p>Nó được sử dụng khi bạn cần truy cập các item theo dạng First-in, First-out. Khi bạn thêm một item vào trong danh sách, nó được gọi là <b>enqueue</b> và khi bạn gỡ bỏ một item, nó được gọi là <b>deque</b>.</p>	
<p><b><u>BitArray trong C#</u></b></p>	<p>Nó biểu diễn một mảng ở dạng <b>biểu diễn nhị phân</b> bởi sử dụng các giá trị 1 và 0.</p> <p>Nó được sử dụng khi bạn cần lưu giữ các Bit nhưng không biết trước số lượng Bit. Bạn có thể truy cập các item từ BitArray collection bởi sử dụng một chỉ mục là số nguyên, mà bắt đầu từ 0.</p>

#### 4.4 Bài tập cuối chương

##### Bài tập 1. Bài tập mảng 1 chiều

1. Đọc và in các phần tử mảng trong C#
2. Cách in mảng theo chiều đảo ngược trong C#
3. Chương trình C# để tìm tổng các phần tử mảng
4. Sao chép mảng trong C#
5. Chương trình C# để tìm số phần tử giống nhau trong một mảng
6. In các phần tử duy nhất của mảng trong C#
7. Chương trình C# để trộn (ghép) hai mảng
8. Chương trình C# để Đếm số lần xuất hiện của từng phần tử trong mảng

9. Chương trình C# để tìm phần tử lớn nhất, nhỏ nhất trong mảng
10. Chia mảng thành mảng chẵn, mảng lẻ trong C#
11. Sắp xếp mảng theo thứ tự tăng dần trong C#
12. Sắp xếp mảng theo thứ tự giảm dần trong C#
13. Chèn phần tử vào mảng trong C# (mảng đã qua sắp xếp)
14. Chèn phần tử vào mảng trong C# (mảng không có thứ tự)
15. Chương trình C# để xóa phần tử trong mảng
16. Tìm phần tử lớn thứ hai trong mảng C#
17. Tìm phần tử nhỏ thứ hai trong mảng C#

## **Bài tập 2. Bài tập mảng 2 chiều**

1. Đọc và in mảng hai chiều trong C#
2. Cộng hai ma trận trong C#
3. Trừ ma trận trong C#
4. Chương trình C# để nhân hai ma trận
5. Tìm ma trận chuyển vị trong C#
6. Chương trình C# để tính tổng các phần tử trên đường chéo chính của ma trận
7. Tính tổng các phần tử trên đường chéo phụ của ma trận trong C#
8. Chương trình C# để tính tổng các hàng, các cột của ma trận
9. In ma trận tam giác trên trong C#
10. Chương trình C# để in ma trận tam giác dưới
11. Tính định thức ma trận trong C#
12. Chương trình C# để kiểm tra ma trận thưa (Sparse Matrix)
13. Chương trình C# để so sánh hai ma trận xem chúng có bằng nhau không

---

## Chương 5: XỬ LÝ CHUỖI VÀ TẬP TIN

### 5.1. Tạo chuỗi

Trong C#, bạn có thể sử dụng các chuỗi (string) như là mảng các ký tự. Tuy nhiên, phổ biến hơn là để sử dụng từ khóa string để khai báo một biến chuỗi. Từ khóa string là một alias cho lớp System.String trong C#.

#### Tạo một đối tượng String trong C#

- Bạn có thể tạo đối tượng String bởi sử dụng một trong các phương thức sau:
- Bằng việc gán một hằng chuỗi cho một biến String
- Sử dụng một constructor của lớp String
- Sử dụng toán tử nối chuỗi (+)
- Bởi việc thu nhận một thuộc tính hoặc gọi một phương thức mà trả về một chuỗi
- Bằng việc gọi một phương thức định dạng để chuyển đổi một giá trị hoặc một đối tượng thành biểu diễn chuỗi của nó.

Ví dụ sau minh họa các phương thức để tạo một chuỗi trong C#:

```
using System;

namespace KhoaTinCSharp
{
    class Dhockink8
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Cac cach tao chuoi trong C#");
            Console.WriteLine("-----");

            //su dung phep gan hang chuoi va toan tu noi chuoi
            string fname, lname;
            fname = "Tran Minh";
            lname = "Chinh";

            string fullname = fname + " " + lname;
            Console.WriteLine("Ho va ten: {0}", fullname);

            //su dung constructor cua lop string
            char[] letters = { 'H', 'e', 'l', 'l', 'o' };
            string greetings = new string(letters);
            Console.WriteLine("\nLoi chao bang tieng Anh: {0}", greetings);

            //tu cac phuong thuc ma tra ve mot chuoi
            string[] sarray = { "Lop DH Tin k8", "xin", "chao", "cac", "ban" };
            string message = String.Join(" ", sarray);
            Console.WriteLine("\nThong diep: {0}", message);

            //dinh dang phuong thuc de chuyen doi mot gia tri
            DateTime waiting = new DateTime(2016, 8, 1, 17, 58, 1);
            string chat = String.Format("Thong diep duoc gui luc {0:t} ngay {0:D}",
```

```

waiting);
        Console.WriteLine("\nThong diep: {0}", chat);

        Console.ReadKey();
    }
}

```

## 5.2. Hàm ToString()

ToString() là phương thức để chuyển bất kỳ 1 đối tượng nào đó thành 1 chuỗi

ToString() vốn dĩ là 1 phương thức của đối tượng Object. Nó trả về 1 chuỗi đại diện cho đối tượng đó.

Nhưng trong .Net, tất cả các đối tượng đều là con của object (một cách gián tiếp hoặc trực tiếp), vì thế, các đối tượng sau này thường override lại phương thức ToString() để trả về 1 chuỗi đại diện cho lớp đó hợp lý hơn.

Mặc định ban đầu của ToString() (chưa bị override) là sẽ xuất ra fullname của đối tượng đó.

```

1. Object a = new Object();
2. a.ToString() //Xuat ra la : System.Object

```

Int32 là 1 đối tượng đã override class Object

```

1. Int32 a = new Int32();
2. a.ToString() //Xuat ra la : 0

```

Vậy là đối tượng Int32 đã override lại Object và xuất ra giá trị của đối tượng đó

Giờ tạo 1 class riêng:

```

1.     public class MyProfile{
2.         String firstName;
3.         String lassName;
4.         public MyProfile()
5.         {
6.             firstName = "A";
7.             lassName = "Nguyen Van";
8.         }
9.
10.        public override string ToString()
11.        {
12.            return string.Format("My name is : {0} {1}. First name is : {0},

```

```
        Llass name is {1}", this.firstName, this.llassName);  
13.        }  
14.    }
```

### 5.3. Thao tác trên chuỗi

#### Khai báo chuỗi

string str ,str1,str2; // khai báo danh sách chuỗi

str="hàm xử lý, xử lý chuỗi c#"; //gán giá trị chuỗi

str1="xử lý chuỗi 1";

str2="xử lý chuỗi 2";

#### Lấy chiều dài chuỗi

str.Length: lấy chiều dài.

#### So sánh 2 chuỗi

String.Compare(str1,str2,true) == 0, <0,>0 so sánh hai chuỗi không phân biệt hoa thường.(str1 bằng, nhỏ hơn, lớn hơn str2)

VD: Kiểm tra xem 2 chuỗi có giống nhau hay không.

```
if (String.Compare(str1, str2, true) == 0)  
{  
    Console.WriteLine("Giống nhau, không phân biệt hoa thường");  
}  
else if (String.Compare(str1, str2, true) < 0)  
{  
    Console.WriteLine("str1 nhỏ hơn str2, không phân biệt hoa thường");//cái khác tương tự  
}
```

String.Compare(str1,str2,false) giống như trên, nhưng phân biệt chữ hoa và chữ thường.

#### Kiểm tra sự tồn tại của chuỗi 1 trong chuỗi 2

Str1.Contains(Str2) :Kiểm tra trong chuỗi Str1 có chuỗi Str2 hay không?

#### Tìm vị trí hiển thị của chuỗi

Str1.IndexOf("chuỗi"): Vị trí xuất hiện đầu tiên của ký tự "chuỗi" trong Str1.

Trả về -1 nếu trong Str1 không có ký tự "chuỗi".

### Tìm vị trí xuất hiện cuối cùng của chuỗi

Str1.LastIndexOf("chuỗi"): Vị trí xuất hiện cuối cùng của ký tự "chuỗi" trong Str1.

Trả về -1 nếu trong Str1 không có ký tự "chuỗi".

### Kiểm tra xem chuỗi 1 có bắt đầu bằng ký tự trong chuỗi 2 không.

Str1.StartsWith(Str2): Kiểm tra xem chuỗi Str1 có bắt đầu bằng chuỗi Str2 không?

### Thay thế chuỗi

Str = Str.Replace(",", "."): Thay thế dấu ',' thành dấu '.' trong chuỗi Str.

Str = Str.Replace("xử lý", "hàm chuỗi"): Thay thế chuỗi 'xử lý' thành chuỗi 'hàm chuỗi' trong chuỗi Str

### Cắt chuỗi con

Str1 = Str.SubString(2): Tạo chuỗi con từ chuỗi Str bắt đầu từ vị trí 2 đến hết

Str1 = Str.Substring(0,6): Cắt chuỗi từ vị trí đầu tiên(vị trí 0) đến vị trí số 6, kết quả là 'hàm xử'

### Tách chuỗi .Split

Dùng để phân tách các ký tự bất kỳ.

Ví dụ:

```
string[] arrListStr = str.Split(',');//tách trong chuỗi str trên khi gặp ký tự ','  
//kết quả arrListStr[0]='hàm xử lý' và arrListStr[1]='xử lý chuỗi c#'
```

Hoặc

```
string[] arrListStr = str.Split(new char[] { ',' });//tách trong chuỗi str trên khi gặp ký  
tự ','  
//kết quả arrListStr[0]='hàm xử lý' và arrListStr[1]='xử lý chuỗi c#'
```

VD: kiểm tra số lần hiển thị của 1 chuỗi bao nhiêu lần thì làm như thế nào? nhanh nhất ra sao.

Kiểm tra số lần hiển thị của chữ "i" trong chuỗi "diễn đàn seo itseovn xử lý chuỗi"

```
string[] arrListStr = str.Split(new char[] { 'i' });//tách trong chuỗi str trên khi  
gặp ký tự 'i'  
int chieudaichuoi = arrListStr.Length - 1;
```

### Chuyển chữ hoa sang chữ thường và ngược lại

Str1 = Str.ToLower(): Chuyển chuỗi sang chữ thường

Str1 = Str.ToUpper() Chuyển chuỗi sang chữ hoa

**Cắt hết khoảng trắng ở đầu và cuối.**

Str = Str.Trim() Cắt hết khoảng trắng ở đầu và cuối chuỗi

Str = Str.TrimLeft() Cắt hết khoảng trắng ở đầu chuỗi

Str = Str.TrimRight() Cắt hết khoảng trắng ở cuối chuỗi

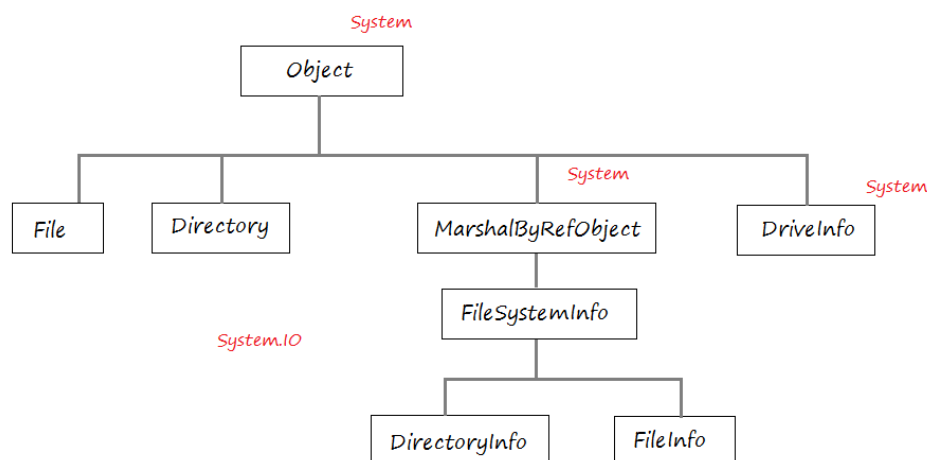
**Xóa chuỗi**

Str1.Remove(1): Xóa chuỗi Str1 từ vị trí 1 đến hết.

Str.Remove(1,5) : Xóa 1 chuỗi con trong Str1 có chiều dài là 5. Từ vị trí 1 đến vị trí 5

## 5.4. Tập tin và thư mục

**Hệ thống phân cấp các lớp**



Class	Mô tả
<b>File</b>	<b>File</b> là một class tiện ích. Nó cung cấp các phương thức tĩnh cho việc tạo, copy, xóa, di chuyển và mở một file, và hỗ trợ tạo đối tượng <b>FileStream</b> .
<b>Directory</b>	<b>Directory</b> là một class tiện ích. Nó cung cấp các phương thức tĩnh để tạo, di chuyển, và liệt kê các thư mục và các thư mục con. Class này không cho phép có class con.
<b>FileInfo</b>	<b>FileInfo</b> là một class đại diện cho một file, nó cung cấp các thuộc tính, phương thức cho việc tạo, copy, xóa, di chuyển và mở file. Nó hỗ trợ tạo đối tượng <b>FileStream</b> . Class này không cho phép có class con.



<b>DirectoryInfo</b>	<b>DirectoryInfo</b> là một class đại diện cho một thư mục, nó cung cấp phương thức cho việc tạo, di chuyển, liệt kê các thư mục và các thư mục con. Class này không cho phép có class con.
<b>DriveInfo</b>	<b>DirveInfo</b> là một class, nó cung cấp các phương thức truy cập thông tin ổ cứng.

### Tệp tin (file)

Một file là một tập hợp dữ liệu được lưu giữ trong một disk với một tên cụ thể và một path thư mục. Khi một file được mở để đọc hoặc ghi, nó trở thành một stream.

Về cơ bản, stream là dãy các byte truyền qua path. Có hai stream quan trọng: Input stream và Output stream. Input stream được sử dụng để đọc dữ liệu từ file (hoạt động read) và Output stream được sử dụng để ghi vào trong file (hoạt động write).

### Thư mục (Directory)

Directory là một class tiện ích. Nó cung cấp các phương thức tĩnh để tạo, di chuyển, và liệt kê các thư mục và các thư mục con. Class này không cho phép có class con.

## 5.5. Đọc và ghi dữ liệu

### Lớp I/O trong C#

Không gian tên System.IO có nhiều lớp đa dạng mà được sử dụng để thực hiện các hoạt động khác nhau với File, như tạo và xóa file, đọc và ghi một File, đóng một File, ...

Bảng sau hiển thị một số lớp non-abstract được sử dụng phổ biến trong System.IO namespace trong C#:

I/O Class	Miêu tả
BinaryReader	Đọc dữ liệu gốc (primitive data) từ một binary stream
BinaryWriter	Ghi dữ liệu gốc trong định dạng nhị phân
BufferedStream	Một nơi lưu giữ tạm thời cho một stream
Directory	Giúp ích trong việc thao tác một cấu trúc thư mục

DirectoryInfo	Được sử dụng để thực hiện các hoạt động trên các thư mục
DriveInfo	Cung cấp thông tin cho các Drive
File	Giúp ích trong việc thao tác các File
FileInfo	Được sử dụng để thực hiện các hoạt động trên các File
FileStream	Được sử dụng để đọc và ghi bất kỳ vị trí nào trong một File
MemoryStream	Được sử dụng để truy cập ngẫu nhiên tới stream được lưu giữ trong bộ nhớ
Path	Thực hiện các hoạt động trên thông tin path
StreamReader	Được sử dụng để đọc các ký tự từ một stream
StreamWriter	Được sử dụng để ghi các ký tự tới một stream
StringReader	Được sử dụng để đọc từ một string buffer
StringWriter	Được sử dụng để ghi vào một string buffer

## Lớp FileStream trong C#

Lớp FileStream trong System.IO namespace trong C# giúp đỡ trong việc đọc từ, ghi và đóng các File. Lớp này kế thừa từ lớp abstract là Stream.

Bạn cần tạo một đối tượng FileStream để tạo một File mới hoặc mở một File đang tồn tại. Cú pháp để tạo một đối tượng FileStream trong C# như sau:

```
FileStream <tên_đối_tượng> = new FileStream( <tên_file>, <FileMode>, <FileAccess>, <FileShare>);
```

Ví dụ: chúng ta tạo một đối tượng FileStream là F để đọc một File với tên **dslopdhtink8.txt**, như sau:

```
using System.IO;
FileStream F = new FileStream("dslopdhtink10.txt", FileMode.Open, FileAccess.Read, FileShare.Read);
```

Các thuộc tính của FileStream:

Tham số	Miêu tả
FileMode	<p><b>FileMode</b> enumerator định nghĩa các phương thức đa dạng để mở các File. Các thành viên của FileMode enumerator là:</p> <p><b>Append:</b> Nó mở một File đang tồn tại và đặt con trỏ tại phần cuối của File, hoặc tạo File, nếu File đó chưa tồn tại</p> <p><b>Create:</b> Nó tạo một File mới</p> <p><b>CreateNew:</b> Nó xác định tới Hệ điều hành rằng nó nên tạo một File mới</p> <p><b>Open:</b> Nó mở một File đang tồn tại</p> <p><b>OpenOrCreate:</b> Nó xác định tới Hệ điều hành rằng nó nên mở một File nếu nó tồn tại, nếu không thì nó nên tạo một File mới</p> <p><b>Truncate:</b> Nó mở một File đang tồn tại và truncate (cắt) kích cỡ của nó về 0 byte</p>
F FileAccess	<p><b>F FileAccess</b> enumerators có các thành viên là: <b>Read</b>, <b>ReadWrite</b> và <b>Write</b>.</p>
FileShare	<p><b>FileShare</b> enumerators có các thành viên sau:</p> <p><b>Inheritable:</b> Nó cho phép một File truyền tính kế thừa tới các tiến trình con</p> <p><b>None:</b> Nó từ chối việc chia sẻ File hiện tại</p>

	<p><b>Read:</b> Nó cho phép mở File để đọc</p> <p><b>ReadWrite:</b> Nó cho phép mở File để đọc và ghi</p> <p><b>Write:</b> Nó cho phép mở File để ghi</p>
--	---

Ví dụ: Dưới đây là ví dụ minh họa cách sử dụng của lớp FileStream trong C#:

```
using System;
using System.IO;

namespace KhoaCNTT
{
    class DHTinK8
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Vi du minh hoa File I/O trong C#");
            Console.WriteLine("-----");

            FileStream F = new FileStream("binary.dat", FileMode.OpenOrCreate,
            FileAccess.ReadWrite);
            for (int i = 1; i <= 20; i++)
            {
                F.WriteByte((byte)i);
            }

            F.Position = 0;
            for (int i = 0; i <= 20; i++)
            {
                Console.Write(F.ReadByte() + " ");
            }
            F.Close();

            Console.ReadKey();
        }
    }
}
```

**Ví dụ xóa 1 file trong C#**

```
using System;
using System.IO;
namespace DaihocThanhDo
{
    class DeleteFileDemo
    {
        public static void Main(string[] args)
        {
            string filePath = "C:/test/test.txt";
```

```

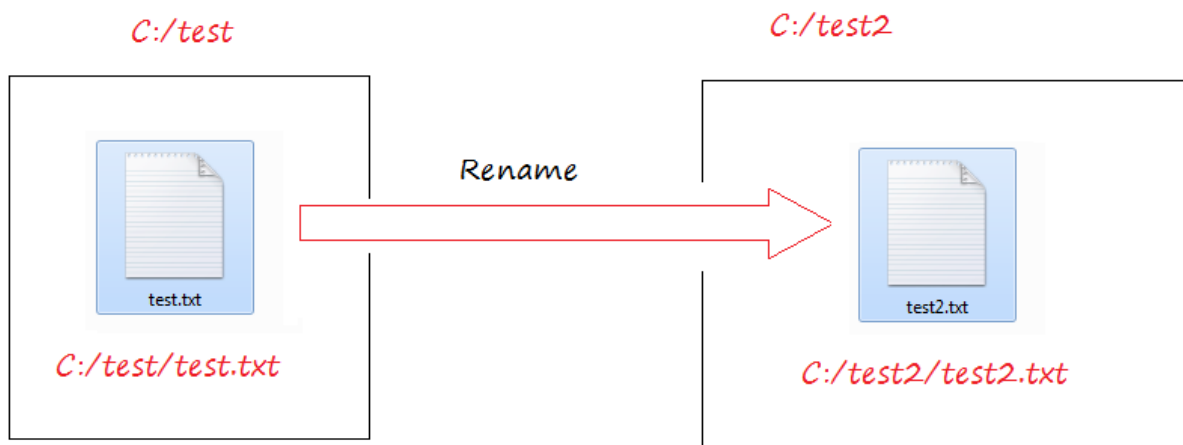
// Kiểm tra đường dẫn này có tồn tại hay không?
if (File.Exists(filePath))
{
    // Xóa file
    File.Delete(filePath);

    // Kiểm tra lại xem file còn tồn tại không.
    if (!File.Exists(filePath))
    {
        Console.WriteLine("File deleted...");
    }
}
else
{
    Console.WriteLine("File test.txt does not yet exist!");
}

Console.ReadKey();
}
}

```

Đổi tên file là một hành động có thể bao gồm di chuyển file tới một thư mục khác và đổi tên file. Trong trường hợp file bị di chuyển tới một thư mục khác phải đảm bảo rằng thư mục mới đã tồn tại.



```

using System;
using System.IO;
namespace DaihocThanhDo
{
    class RenameFileDemo
    {
        public static void Main(string[] args)
        {
            String filePath = "C:/test/test.txt";

            if (File.Exists(filePath))

```

```

    {
        Console.WriteLine(filePath + " exist");

        Console.WriteLine("Please enter a new name for this file:");

        // Một String mà người dùng nhập vào.
        // Ví dụ: C:/test/test2.txt
        string newFilename = Console.ReadLine();
        if (newFilename != String.Empty)
        {
            // Đổi tên file:
            // Có thể bao gồm, chuyển file tới một thư mục cha khác, và đổi tên
file.

            // Phải đảm bảo rằng thư mục cha mới tồn tại.
            // (nếu không ngoại lệ DirectoryNotFoundException sẽ được ném ra).
            File.Move(filePath, newFilename);

            if (File.Exists(newFilename))
            {
                Console.WriteLine("The file was renamed to " + newFilename);
            }
        }
        else
        {
            Console.WriteLine("Path " + filePath + " does not exist.");
        }

        Console.ReadLine();
    }
}

```

## Hoạt động File nâng cao trong C#

Ví dụ trước minh họa các hoạt động đơn giản trên File trong C#. Tuy nhiên, để lợi dụng tối đa sức mạnh của các lớp System.IO trong C#, bạn cần biết các thuộc tính và phương thức được sử dụng phổ biến về các lớp này.

### Đọc, ghi thư mục

Kiểm tra một đường dẫn thư mục có tồn tại hay không, nếu không tồn tại tạo thư mục đó, ghi ra thông tin thời gian tạo, lần ghi dữ liệu cuối vào thư mục, ....

```

using System;
using System.IO;

namespace DaihocThanhDo
{
    class DirectoryInfoDemo
    {

```

```

public static void Main(string[] args)
{
    String dirPath = "C:/test/CSharp";

    // Kiểm tra xem đường dẫn thư mục tồn tại không.
    bool exist = Directory.Exists(dirPath);

    // Nếu không tồn tại, tạo thư mục này.
    if (!exist)
    {
        Console.WriteLine(dirPath + " does not exist.");
        Console.WriteLine("Create directory: " + dirPath);

        // Tạo thư mục.
        Directory.CreateDirectory(dirPath);
    }

    Console.WriteLine("Directory Information " + dirPath);

    // In ra các thông tin thư mục trên.
    // Thời điểm tạo thư mục.
    Console.WriteLine("Creation time: " + Directory.GetCreationTime(dirPath));

    // Thời điểm cuối cùng thư mục có sự thay đổi.
    Console.WriteLine("Last Write Time: " + Directory.GetLastWriteTime(dirPath));

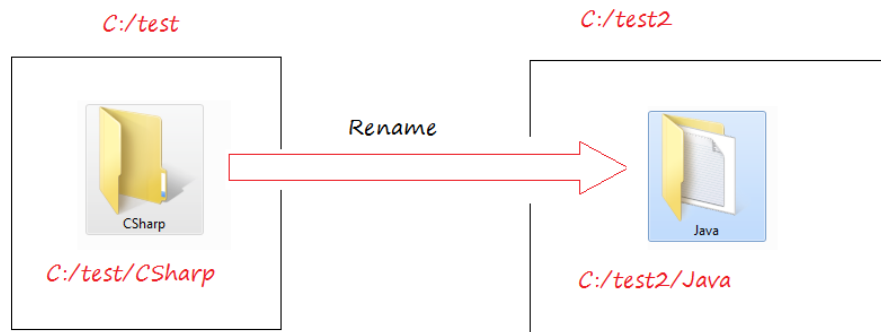
    // Thư mục cha.
    DirectoryInfo parentInfo = Directory.GetParent(dirPath);

    Console.WriteLine("Parent directory: " + parentInfo.FullName);

    Console.Read();
}
}

```

### Đổi tên thư mục:



Bạn có thể thay đổi tên một thư mục. Nó có thể làm thư mục đó chuyển ra khỏi thư mục cha hiện tại. Nhưng bạn phải đảm bảo rằng thư mục cha mới đã tồn tại. Ví dụ dưới đây minh họa đổi tên một thư mục:

```
using System;
using System.IO;

namespace DaihocThanhDo
{
    class RenameDirectoryDemo
    {
        public static void Main(string[] args)
        {
            // Một đường dẫn thư mục.
            String dirPath = "C:/test/CSharp";

            // Nếu đường dẫn này tồn tại.
            if (!Directory.Exists(dirPath))
            {
                Console.WriteLine(dirPath + " does not exist.");
                Console.Read();

                return;
            }

            Console.WriteLine(dirPath + " exist");

            Console.WriteLine("Please enter a new name for this directory:");

            // String mà người dùng nhập vào.
            // Ví dụ: C:/test2/Java
            string newDirname = Console.ReadLine();

            if (newDirname == String.Empty)
            {
                Console.WriteLine("You not enter new directory name. Cancel rename.");
                Console.Read();

                return;
            }

            // Nếu đường dẫn mà người dùng nhập vào là tồn tại.
            if (Directory.Exists(newDirname))
            {
                Console.WriteLine("Cannot rename directory. New directory already exist.");
                Console.Read();

                return;
            }

            // Thư mục cha.
            DirectoryInfo parentInfo = Directory.GetParent(newDirname);

            // Tạo thư mục cha của thư mục mà người dùng nhập vào.
            Directory.CreateDirectory(parentInfo.FullName);
```



```

        // Bạn có thể đổi đường dẫn (path) của một thư mục.
        // nhưng phải đảm bảo đường dẫn cha của đường dẫn mới phải tồn tại.
        // (Nếu không ngoại lệ DirectoryNotFoundException sẽ được ném ra).
        Directory.Move(dirPath, newDirname);

        if (Directory.Exists(newDirname))
        {
            Console.WriteLine("The directory was renamed to " + newDirname);
        }

        Console.ReadLine();
    }
}
}

```

Ví dụ dưới đây đệ quy và in ra tất cả các thư mục hậu duệ (con, cháu,...) của một thư mục.

```

using System;
using System.Collections.Generic;
using System.IO;

namespace DaiHocThanhDo
{
    class EnumeratingDirectoryDemo
    {
        public static void Main(string[] args)
        {
            string dirPath = "C:/Windows/System32";

            PrintDirectory(dirPath);

            Console.Read();
        }

        // Phương thức đệ quy (Recursive) liệt kê ra các thư mục con của một thư mục.
        public static void PrintDirectory(string dirPath)
        {
            try
            {
                // Nếu bạn không có quyền truy cập thư mục 'dirPath'
                // một ngoại lệ UnauthorizedAccessException sẽ được ném ra.
                IEnumerable<string> enums = Directory.EnumerateDirectories(dirPath);

                List<string> dirs = new List<string>(enums);

                foreach (var dir in dirs)
                {
                    Console.WriteLine(dir);

                    PrintDirectory(dir);
                }
            }
        }
    }
}

```

```
    }  
    }  
    // Lỗi bảo mật khi truy cập vào thư mục mà bạn không có quyền.  
    catch (UnauthorizedAccessException e)  
    {  
        Console.WriteLine("Can not access directory: " + dirPath);  
        Console.WriteLine(e.Message);  
    }  
    }  
    }  
}
```

## 5.5 Bài tập cuối chương.

**Bài tập 1.** Lấy thông tin về File, Directory hoặc Drive

**Bài tập 2.** Sao chép, di chuyển và xóa tập tin hoặc thư mục, Chép nội dung của một thư mục sang một thư mục được tạo ra.

**Bài tập 3.** Đọc nội dung của 1 file.

**Bài tập 4.** Ghi nội dung vào 1 file. (Ví dụ danh sách lớp, danh sách hàng hóa...)

**Bài tập 5.** Đọc nội dung từ 1 file, chép vào cuối nội dung của 1 file khác.

**Bài tập 6.** Đọc nội dung 1 file, thay thế từ trong thư file này bằng 1 từ khác, lưu thành 1 file khác.

**Bài tập 7.** Nhập 1 văn bản, ghi văn bản đó vào 1 file, trước khi lưu, hãy cho người dùng 2 lựa chọn, lựa chọn 1 là lưu với sự **chuẩn hóa văn bản** và lựa chọn 2, lưu mà không chuẩn hóa văn bản.

Biết rằng, văn bản chuẩn hóa là: đầu đoạn văn bản, viết hoa, sau dấu câu, phải có 1 dấu cách, sau dấu chấm thì phải viết hoa ...

**Bài tập 8.** Tạo ra các chỉ thị lệnh cho thư mục, file. Ví dụ: Del Thumuc, thí máy tính sẽ thực hiện việc xóa thư mục Thumuc...

**Bài tập 9.** Viết chương trình nhập vào các bộ gồm 3 số thực, lưu mỗi bộ trên 1 dòng trong file text, mỗi số thực cách nhau bởi 1 dấu cách, lưu file text có tên là data.txt.

Đọc file text, đọc từng dòng trong file text trên, xét xem 3 số đó có phải 3 cạnh của một tam giác hay không, là tam giác gì, lưu tất cả lời giải tam giác vào file “giaitamgiac.txt”.

Đọc file text, đọc từng dòng trong file text trên, coi 3 số thực đó là 3 hệ số trong phương trình bậc nhất 2 ẩn. Lưu tất cả lời giải vào 1 file text “Giaiphuongtrinh.txt”

## Chương 6: LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG VỚI C#

### 6.1. Xây dựng lớp, đối tượng

#### Lớp trong C#

**Lớp:** Một tập hợp các đối tượng. Nó là một thực thể logic.

Lớp có các đặc tính đặc thù của hướng đối tượng:

- Tính kế thừa.
- Tính bao đóng.
- Tính đa hình.
- Tính trừu tượng.

Định nghĩa lớp trong C# bắt đầu với từ khóa **class** được theo sau bởi tên lớp; và phần thân lớp được bao quanh bởi các dấu ngoặc ôm. Dưới đây là form chung của một định nghĩa lớp trong C#:

```
<access specifier> class tên_lớp
{
    // các biến thành viên
    <access specifier> <kiểu_dữ_liệu> biến1;
    <access specifier> <kiểu_dữ_liệu> biến2;
    ...
    <access specifier> <kiểu_dữ_liệu> biếnN;
    // các phương thức thành viên
    <access specifier> <kiểu_trả_về> tên_phương_thức1(danh_sách_tham_số)
    {
        // phần thân phương thức
    }
    <access specifier> <kiểu_trả_về> tên_phương_thức2(danh_sách_tham_số)
    {
        // phần thân phương thức
    }
    ...
    <access specifier> <kiểu_trả_về> tên_phương_thứcN(danh_sách_tham_số)
    {
        // phần thân phương thức
    }
}
```

Ghi chú:

- Access specifier xác định các quy tắc truy cập cho các thành viên cũng như chính lớp đó. Nếu không được đề cập, thì Access Specifier mặc định cho một kiểu lớp là internal. Chế độ truy cập mặc định cho các thành viên là private.
- kiểu\_dữ\_liệu xác định kiểu biến, và trả về kiểu dữ liệu mà phương thức trả về.
- Để truy cập các thành viên lớp, bạn sử dụng toán tử dot (.).
- Toán tử dot (.) liên kết với tên của một đối tượng với tên của một thành viên.
- Ví dụ sau minh họa các khái niệm về lớp trong C# được đề cập ở trên: tạo hai class có tên lần lượt là Box và TestCsharp trong hai file riêng biệt.

Ví dụ lớp hình hộp (box)

```
namespace DaihocThanhDo
{
    class Box
    {
        public double chieu_dai;
        public double chieu_rong;
        public double chieu_cao;
    }
}
```

Lớp TestCsharp: chứa phương thức main() để thao tác trên đối tượng Box

```
using System;
namespace DaihocThanhDo
{
    public class TestCsharp
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Class trong C#");
            Console.WriteLine("-----\n");

            Box Box1 = new Box(); // tao doi tuong Box1
            Box Box2 = new Box(); // tao doi tuong Box2
            double the_tich = 0.0; // the tich cua box

            // thong tin cua box1
            Box1.chieu_cao = 5.0;
            Box1.chieu_dai = 6.0;
            Box1.chieu_rong = 7.0;

            // thong tin cua box2
            Box2.chieu_cao = 10.0;
            Box2.chieu_dai = 12.0;
            Box2.chieu_rong = 13.0;
        }
    }
}
```

```

        // Tính và in the tích của box1
        the_tich = Box1.chieu_cao * Box1.chieu_dai * Box1.chieu_rong;
        Console.WriteLine("The tích của Box1 là: {0}", the_tich);

        // Tính và in the tích của box2
        the_tich = Box2.chieu_cao * Box2.chieu_dai * Box2.chieu_rong;
        Console.WriteLine("The tích của Box2 là: {0}", the_tich);

        Console.ReadKey();
    }
}
}

```

Object (đối tượng) nghĩa là một thực thể trong thế giới thực, chẳng hạn như bàn, quả bóng, con bò, ... Lập trình hướng đối tượng là một phương pháp để thiết kế một chương trình bởi sử dụng các lớp và các đối tượng. Nó làm đơn giản hóa việc duy trì và phát triển phần mềm bằng việc cung cấp một số khái niệm:

**Đối tượng:** Một thực thể có trạng thái và hành vi. Ví dụ như xe đạp, bàn, ghế, ... Nó có thể mang tính vật lý hoặc logic.

## 6.2. Kế thừa, đa hình

Từ polymorphism (tính đa hình) nghĩa là có nhiều hình thái. Trong lập trình hướng đối tượng, tính đa hình thường được diễn đạt như là "một Interface, nhiều hàm".

Tính đa hình trong C# có thể là static hoặc dynamic. Trong đó, kiểu đa hình static có thể được gọi là đa hình tĩnh và kiểu đa hình dynamic có thể được gọi là đa hình động.

Trong đa hình tĩnh, phần phản hồi tới một hàm được xác định tại compile time. Trong khi đó với đa hình động, nó được quyết định tại runtime.

**Tính kế thừa:** Khi một đối tượng đạt được các thuộc tính và các hành vi của đối tượng cha, thì đó là tính kế thừa. Điều này làm tăng tính tái sử dụng cho code. Nó được sử dụng để đạt được tính đa hình tại runtime.

### Đa hình static trong C#

**Tính đa hình:** Khi một tác vụ được thực hiện theo nhiều cách khác nhau được gọi là tính đa hình. Ví dụ: như vẽ hình chữ nhật hoặc hình tam giác, ... Trong C#, chúng ta sử dụng nạp chồng phương thức (method overloading) và ghi đè phương thức (method overriding) để có tính đa hình. Một ví dụ khác: con mèo kêu meooo, còn chú chó thì sủa goooo.

Kỹ thuật liên kết một hàm với một đối tượng trong thời gian biên dịch được gọi là Early Binding. Nó cũng được gọi là Static Binding. C# cung cấp hai kỹ thuật để triển khai đa hình tĩnh. Chúng là:

- Nạp chồng hàm (Function overloading)

- Nạp chồng toán tử (Operator overloading)

Chúng ta sẽ bàn luận về nạp chồng toán tử trong chương sau.

Ví dụ sau minh họa cách sử dụng hàm print() để in các kiểu dữ liệu khác nhau trong C#:

```
using System;
namespace DaihocThanhDo
{
    public class TestCsharp
    {
        void print(int i)
        {
            Console.WriteLine("In so nguyen: {0}", i);
        }
        void print(double f)
        {
            Console.WriteLine("In so thuc: {0}", f);
        }
        void print(string s)
        {
            Console.WriteLine("In chuoi: {0}", s);
        }

        static void Main(string[] args)
        {
            Console.WriteLine("Tinh da hinh trong C#");
            Console.WriteLine("-----");
            //tao doi tuong TestCsharp
            TestCsharp p = new TestCsharp();

            // goi ham print()
            p.print(5);
            p.print(500.263);
            p.print("Hoc C# co ban va nang cao");
            Console.ReadKey();
        }
    }
}
```

### Đa hình dynamic trong C#

C# cho phép bạn tạo các lớp abstract (trừu tượng) mà được sử dụng để cung cấp trình triển khai cục bộ lớp của một Interface. Trình triển khai (Implementation) được hoàn thành khi một lớp kế thừa kế thừa từ nó. Các lớp Abstract chứa các phương thức abstract, mà được triển khai bởi lớp kế thừa. Lớp kế thừa này có tính năng chuyên dụng hơn.

Dưới đây là một số qui tắc về các lớp abstract trong C#:

Bạn không thể tạo một Instance (sự thể hiện) của một lớp abstract.

Bạn không thể khai báo một phương thức abstract ở bên ngoài một lớp abstract.

Khi một lớp được khai báo là sealed, nó không thể được kế thừa, các lớp abstract không thể được khai báo là sealed.

Ví dụ sau minh họa một lớp abstract trong C#: tạo 3 lớp có tên lần lượt là Shape, HìnhChuNhat, TestCsharp như sau:

Lớp Shape: là một lớp abstract

```
using System;

namespace DaiHocThanhDo
{
    abstract class Shape
    {
        public abstract int tinhDienTich();
    }
}
```

Lớp **HìnhChuNhat**: là một lớp kế thừa lớp Shape

```
using System;

namespace DaiHocThanhDo
{
    class HìnhChuNhat : Shape
    {
        private int chieu_dai;
        private int chieu_rong;
        public HìnhChuNhat(int a = 0, int b = 0)
        {
            chieu_dai = a;
            chieu_rong = b;
        }
        public override int tinhDienTich()
        {
            Console.WriteLine("Dien tich hình chu nhat:");
            return (chieu_rong * chieu_dai);
        }
    }
}
```

Lớp **TestCsharp**: chứa phương thức **main()** để thao tác trên đối tượng HìnhChuNhat

```

using System;
namespace DaiHocThanhDo
{
    public class TestCsharp
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Tính đa hình trong C#");
            Console.WriteLine("Ví dụ minh họa Đa hình động");
            Console.WriteLine("-----");

            HìnhChuNhat r = new HìnhChuNhat(10, 7);
            double a = r.tinhDienTich();
            Console.WriteLine("Diện tích: {0}", a);
            Console.ReadKey();
        }
    }
}

```

Đa hình động trong C# được triển khai bởi các lớp abstract và các hàm virtual.

Ví dụ sau minh họa điều này: tạo 5 lớp có tên lần lượt là như sau:

Lớp Shape: lớp cơ sở

```

using System;

namespace DaihocThanhDo
{
    class Shape
    {
        protected int chieu_rong, chieu_cao;
        public Shape(int a = 0, int b = 0)
        {
            chieu_rong = a;
            chieu_cao = b;
        }
        public virtual int tinhDienTich()
        {
            Console.WriteLine("Diện tích của class cha: ");
            return 0;
        }
    }
}

```

Lớp HìnhChuNhat: là lớp kế thừa lớp Shape

```

using System;

namespace DaihocThanhDo
{
    class HìnhChuNhat : Shape
    {
    }
}

```



```

    {
        public HìnhChuNhat(int a = 0, int b = 0) : base(a, b)
        {
        }
        public override int tinhDienTich()
        {
            Console.WriteLine("Dien tich cua class HìnhChuNhat: ");
            return (chieu_rong * chieu_cao);
        }
    }
}

```

Lớp TamGiac: là lớp kế thừa lớp Shape

```

using System;

namespace DaihocThanhDo
{
    class TamGiac : Shape
    {
        public TamGiac(int a = 0, int b = 0) : base(a, b)
        {
        }
        public override int tinhDienTich()
        {
            Console.WriteLine("Dien tich cua class TamGiac: ");
            return (chieu_cao * chieu_rong / 2);
        }
    }
}

```

Lớp HienThiDuLieu: in các dữ liệu

```

using System;

namespace DaihocThanhDo
{
    class HienThiDuLieu
    {
        public void hienThiDienTich(Shape sh)
        {
            int a;
            a = sh.tinhDienTich();
            Console.WriteLine("Dien tich: {0}", a);
        }
    }
}

```

Lớp TestCsharp: chứa phương thức main() để thao tác trên các đối tượng

```

using System;
namespace DaihocThanhDo
{
    public class TestCsharp
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Tinh da hình trong C#");
            Console.WriteLine("Ví dụ minh họa Da hình dòng");
            Console.WriteLine("-----");

            HienThiDuLieu c = new HienThiDuLieu();
            HìnhChuNhat r = new HìnhChuNhat(10, 7);
            TamGiac t = new TamGiac(10, 5);
            c.hienThiDienTich(r);
            c.hienThiDienTich(t);

            Console.ReadKey();
        }
    }
}

```

**Tính đóng gói:** Đó là gắn kết code và dữ liệu cùng với nhau vào trong một đơn vị unit đơn. Ví dụ: có thể bạn đã biết đến viên thuốc con nhộng (hay đơn giản hơn là gói bột giặt), các viên thuốc (hạt bột giặt) khác nhau được đóng gói.

### 6.3. Nạp chồng toán tử

#### Nạp chồng hàm trong C#

Bạn có thể có nhiều định nghĩa cho cùng tên hàm trong cùng một phạm vi. Các định nghĩa này của hàm phải khác nhau: như kiểu và/hoặc số tham số trong danh sách tham số. Trong C#, bạn không thể nạp chồng các khai báo hàm mà chỉ khác nhau ở kiểu trả về.

Operator Overloading là Nạp chồng toán tử. Bạn có thể tái định nghĩa hoặc nạp chồng hầu hết các toán tử có sẵn trong C#. Vì thế, một lập trình viên có thể sử dụng các toán tử với các kiểu tự định nghĩa (user-defined). Các toán tử được nạp chồng trong C# là các hàm với các tên đặc biệt: từ khóa operator được theo sau bởi biểu tượng cho toán tử đang được định nghĩa. Tương tự như bất kỳ hàm nào khác, một toán tử được nạp chồng có một kiểu trả về và một danh sách tham số.

Ví dụ, bạn xét hàm sau:

```

public static Box operator +(Box b, Box c)
{
    Box box = new Box();
    box.chieu_dai = b.chieu_dai + c.chieu_dai;
    box.chieu_rong = b.chieu_rong + c.chieu_rong;
    box.chieu_cao = b.chieu_cao + c.chieu_cao;
    return box;
}

```

Hàm trên triển khai toán tử cộng (+) cho một lớp Box tự định nghĩa (user-defined). Nó cộng các thuộc tính của hai đối tượng Box và trả về đối tượng kết quả Box.

### Triển khai Nạp chồng toán tử trong C#

Ví dụ dưới đây minh họa cách triển khai nạp chồng toán tử trong C#: tạo hai lớp có tên lần lượt là Box, TestCsharp như sau:

Lớp Box: chứa các thuộc tính và phương thức

```
using System;

namespace DaihocThanhDo
{
    class Box
    {
        private double chieu_dai;
        private double chieu_rong;
        private double chieu_cao;

        public double tinhTheTich()
        {
            return chieu_dai * chieu_rong * chieu_cao;
        }

        public void setChieuDai(double len)
        {
            chieu_dai = len;
        }

        public void setChieuRong(double bre)
        {
            chieu_rong = bre;
        }

        public void setChieuCao(double hei)
        {
            chieu_cao = hei;
        }

        // nạp chồng toán tử + để cộng hai đối tượng Box.
        public static Box operator +(Box b, Box c)
        {
            Box box = new Box();
            box.chieu_dai = b.chieu_dai + c.chieu_dai;
            box.chieu_rong = b.chieu_rong + c.chieu_rong;
            box.chieu_cao = b.chieu_cao + c.chieu_cao;
            return box;
        }
    }
}
```

**Trong hàm main sử dụng toán tử nạp chồng + cho 2 box.**

```
using System;
namespace KhoaCNTTDHTHanhdo
{
    public class TestCsharp
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Nap chong toan tu trong C#");
            Console.WriteLine("Vi du minh hoa nap chong toan tu");
            Console.WriteLine("-----");
            //tao cac doi tuong Box1, Box2 va Box3
            Box Box1 = new Box();
            Box Box2 = new Box();
            Box Box3 = new Box();
            double the_tich = 0.0;

            // nhap thong tin Box1
            Box1.setChieuDai(6.0);
            Box1.setChieuRong(7.0);
            Box1.setChieuCao(5.0);

            // nhap thong tin Box2
            Box2.setChieuDai(12.0);
            Box2.setChieuRong(13.0);
            Box2.setChieuCao(10.0);

            // tinh va hien thi the tich Box1
            the_tich = Box1.tinhTheTich();
            Console.WriteLine("The tich cua Box1 la: {0}", the_tich);

            // tinh va hien thi the tich Box2
            the_tich = Box2.tinhTheTich();
            Console.WriteLine("The tich cua Box2 la: {0}", the_tich);

            // con hai doi tuong
            Box3 = Box1 + Box2;

            // tinh va hien thi the tich Box3
            the_tich = Box3.tinhTheTich();
            Console.WriteLine("The tich cua Box3 la: {0}", the_tich);
            Console.ReadKey();
        }
    }
}
```

**Toán tử có thể nạp chồng và không thể nạp chồng trong C#**

Bảng dưới miêu tả các toán tử có thể nạp chồng trong C#:

Toán tử	Miêu tả
---------	---------

+, -, !, ~, ++, --	Những toán tử một ngôi này nhận một toán hạng và <b>có thể</b> được nạp chồng
+, -, *, /, %	Những toán tử nhị phân này nhận một toán hạng và <b>có thể</b> được nạp chồng
==, !=, <, >, <=, >=	Các toán tử so sánh <b>có thể</b> được nạp chồng
&&,	Các toán tử logic điều kiện <b>không thể</b> được nạp chồng một cách trực tiếp
+=, -=, *=, /=, %=	Các toán tử gán <b>không thể</b> được nạp chồng
=, ., ?:, ->, new, is, sizeof, typeof	Các toán tử này <b>không thể</b> được nạp chồng

## Ví dụ

Từ các khái niệm trên, chúng ta kế thừa ví dụ trên và nạp chồng thêm một số toán tử trong C#: ở trên chúng ta đã tạo hai lớp Box, TestCsharp, bây giờ chúng ta sửa lại code của mỗi lớp như sau:

Lớp Box: chứa các thuộc tính và phương thức

```
using System;

namespace DaiHocThanhDo
{
    class Box
    {
        private double chieu_dai;
        private double chieu_rong;
        private double chieu_cao;

        public double tinhTheTich()
        {
            return chieu_dai * chieu_rong * chieu_cao;
        }

        public void setChieuDai(double len)
        {
            chieu_dai = len;
        }

        public void setChieuRong(double bre)
        {
            chieu_rong = bre;
        }
    }
}
```

```

    }

    public void setChieuCao(double hei)
    {
        chieu_cao = hei;
    }
    // nap chong toan tu +
    public static Box operator +(Box b, Box c)
    {
        Box box = new Box();
        box.chieu_dai = b.chieu_dai + c.chieu_dai;
        box.chieu_rong = b.chieu_rong + c.chieu_rong;
        box.chieu_cao = b.chieu_cao + c.chieu_cao;
        return box;
    }
    //nap chong toan tu ==
    public static bool operator ==(Box lhs, Box rhs)
    {
        bool status = false;
        if (lhs.chieu_dai == rhs.chieu_dai && lhs.chieu_cao == rhs.chieu_cao &&
lhs.chieu_rong == rhs.chieu_rong)
        {
            status = true;
        }
        return status;
    }
    //nap chong toan tu !=
    public static bool operator !=(Box lhs, Box rhs)
    {
        bool status = false;
        if (lhs.chieu_dai != rhs.chieu_dai || lhs.chieu_cao != rhs.chieu_cao ||
lhs.chieu_rong != rhs.chieu_rong)
        {
            status = true;
        }
        return status;
    }
    //nap chong toan tu <
    public static bool operator <(Box lhs, Box rhs)
    {
        bool status = false;
        if (lhs.chieu_dai < rhs.chieu_dai && lhs.chieu_cao < rhs.chieu_cao &&
lhs.chieu_rong < rhs.chieu_rong)
        {
            status = true;
        }
        return status;
    }
    //nap chong toan tu >
    public static bool operator >(Box lhs, Box rhs)
    {
        bool status = false;
        if (lhs.chieu_dai > rhs.chieu_dai && lhs.chieu_cao > rhs.chieu_cao &&
lhs.chieu_rong > rhs.chieu_rong)
        {
            status = true;
        }
    }

```

```

        return status;
    }
    //nap chong toan tu <=
    public static bool operator <=(Box lhs, Box rhs)
    {
        bool status = false;
        if (lhs.chieu_dai <= rhs.chieu_dai && lhs.chieu_cao <= rhs.chieu_cao &&
lhs.chieu_rong <= rhs.chieu_rong)
        {
            status = true;
        }
        return status;
    }
    //nap chong toan tu >=
    public static bool operator >=(Box lhs, Box rhs)
    {
        bool status = false;
        if (lhs.chieu_dai >= rhs.chieu_dai && lhs.chieu_cao >= rhs.chieu_cao &&
lhs.chieu_rong >= rhs.chieu_rong)
        {
            status = true;
        }
        return status;
    }
    //nap chong phuong thuc ToString()
    public override string ToString()
    {
        return String.Format("{0}, {1}, {2}", chieu_dai, chieu_rong, chieu_cao);
    }
}
}

```

Lớp TestCsharp: chứa phương thức main() để thao tác trên đối tượng Box

```

using System;
namespace DaiHocThanhDo
{
    public class TestCsharp
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Nap chong toan tu trong C#");
            Console.WriteLine("Vi du minh hoa nap chong toan tu");
            Console.WriteLine("-----");
            //tao cac doi tuong Box1, Box2, Box2, Box4
            Box Box1 = new Box();
            Box Box2 = new Box();
            Box Box3 = new Box();
            Box Box4 = new Box();
            double the_tich = 0.0;

            // nhap thong tin Box1
            Box1.setChieuDai(6.0);
            Box1.setChieuRong(7.0);
            Box1.setChieuCao(5.0);

```

```

// nhập thông tin Box2
Box2.setChieuDai(12.0);
Box2.setChieuRong(13.0);
Box2.setChieuCao(10.0);

//hiển thị thông tin các Box bởi sử dụng phương thức nạp chồng ToString():
Console.WriteLine("Thông tin Box 1: {0}", Box1.ToString());
Console.WriteLine("Thông tin Box 2: {0}", Box2.ToString());

// tính thể tích Box1
the_tich = Box1.tinhTheTich();
Console.WriteLine("Thể tích của Box1 là: {0}", the_tich);

// tính thể tích Box2
the_tich = Box2.tinhTheTich();
Console.WriteLine("Thể tích của Box2 là: {0}", the_tich);

// cộng hai đối tượng
Box3 = Box1 + Box2;
Console.WriteLine("Thông tin Box 3: {0}", Box3.ToString());

// tính thể tích của Box3
the_tich = Box3.tinhTheTich();
Console.WriteLine("Thể tích của Box3 là: {0}", the_tich);

//so sánh các Box
if (Box1 > Box2)
    Console.WriteLine("Box1 là lớn hơn Box2");
else
    Console.WriteLine("Box1 là không lớn hơn Box2");

if (Box1 < Box2)
    Console.WriteLine("Box1 là nhỏ hơn Box2");
else
    Console.WriteLine("Box1 là không nhỏ hơn Box2");

if (Box1 >= Box2)
    Console.WriteLine("Box1 là lớn hơn hoặc bằng Box2");
else
    Console.WriteLine("Box1 là không lớn hơn hoặc bằng Box2");

if (Box1 <= Box2)
    Console.WriteLine("Box1 là nhỏ hơn hoặc bằng Box2");
else
    Console.WriteLine("Box1 là không nhỏ hơn hoặc bằng Box2");

if (Box1 != Box2)
    Console.WriteLine("Box1 là không bằng Box2");
else
    Console.WriteLine("Box1 bằng Box2");
Box4 = Box3;

if (Box3 == Box4)
    Console.WriteLine("Box3 bằng Box4");
else
    Console.WriteLine("Box3 là không bằng Box4");

```



```
        Console.ReadKey();  
    }  
}
```

## 6.4 Bài tập cuối chương

**Bài tập 1.** Xây dựng một lớp Số (SO) có thể tự kiểm tra xem giá trị là chẵn, lẻ, nguyên tố, hoàn hảo... hay không? Nạp chồng phương thức +,-,\*,/ để cộng trừ nhân chia hai SO.

**Bài tập 2.** Xây dựng lớp phân số, có thể tự giản ước, cộng, trừ, nhân, chia 2 phân số.

**Bài tập 3.** Xây dựng lớp sinh viên, có thể xem điểm, xem họ tên, các thông tin đầy đủ về sinh viên...

**Bài tập 4.** Hãy xem xét một tình huống cần được triển khai thành một hệ thống trên máy vi tính: việc mua bán xe hơi. Vấn đề vi tính hóa việc mua bán xe hơi bao gồm những gì?

Những yếu tố rõ ràng nhất liên quan đến việc mua bán xe hơi là:

- 1) Các kiểu xe hơi (model).
- 2) Nhân viên bán hàng.
- 3) Khách hàng.

Những hoạt động liên quan đến việc mua bán:

- 1) Nhân viên bán hàng đưa khách hàng tham quan phòng trưng bày.
- 2) Khách hàng chọn lựa một xe hơi.
- 3) Khách hàng đặt hóa đơn.
- 4) Khách hàng trả tiền.
- 5) Chiếc xe được trao cho khách hàng.

Hãy viết chương trình, tạo ra các lớp để thực thi được các hoạt động trên.

**Bài tập 5.** Xây dựng chương trình quản lý điểm, có thể nhập danh sách học sinh, nhập danh sách môn học, nhập điểm và hiển thị trên màn hình kết quả học tập của sinh viên.

---

## Chương 7: THỰC THI GIAO DIỆN

### 7.1. Khai báo giao diện

Một Interface được định nghĩa như là một giao ước có tính chất cú pháp (syntactical contract) mà tất cả lớp kế thừa Interface đó nên theo. Interface định nghĩa phần "Là gì" của giao ước và các lớp kế thừa định nghĩa phần "Cách nào" của giao ước đó.

Interface định nghĩa các thuộc tính, phương thức và sự kiện, mà là các thành viên của Interface đó. Các Interface chỉ chứa khai báo của các thành viên này. Việc định nghĩa các thành viên là trách nhiệm của lớp kế thừa. Nó thường giúp ích trong việc cung cấp một Cấu trúc chuẩn mà các lớp kế thừa nên theo.

Các Interface được khai báo bởi sử dụng từ khóa interface trong C#. Nó tương tự như khai báo lớp. Theo mặc định, các lệnh Interface là public. Ví dụ sau minh họa một khai báo Interface trong C#:

```
public interface ITransactions
{
    // các thành viên của interface

    //các phương thức
    void hiểnThiThôngTinGiaoDich();
    double laySoLuong();
}
```

Sau đây là ví dụ minh họa trình triển khai của Interface trên: tạo 2 lớp có tên lần lượt là GiaoDichHangHoa, TestCsharp và một interface có tên là GiaoDich

```
namespace DaiHocThanhDo
{
    public interface GiaoDich
    {
        // cac thanh vien cua interface

        //cac phuong thuc
        void hiểnThiThôngTinGiaoDich();
        double laySoLuong();
    }
}
```

### 7.2. Thực thi giao diện

Lớp GiaoDichHangHoa kế thừa (thực thi) interface GiaoDich

```

using System;
namespace DaiHocThanhDo
{
    class GiaoDichHangHoa : GiaoDich
    {
        private string ma_hang_hoa;
        private string ngay;
        private double so_luong;
        public GiaoDichHangHoa()
        {
            ma_hang_hoa = " ";
            ngay = " ";
            so_luong = 0.0;
        }

        public GiaoDichHangHoa(string c, string d, double a)
        {
            ma_hang_hoa = c;
            ngay = d;
            so_luong = a;
        }

        public double laySoLuong()
        {
            return so_luong;
        }

        public void hienThiThongTinGiaoDich()
        {
            Console.WriteLine("Ma hang hoa: {0}", ma_hang_hoa);
            Console.WriteLine("Ngay giao dich: {0}", ngay);
            Console.WriteLine("So luong: {0}", laySoLuong());
        }
    }
}

```

Lớp TestCsharp chứa phương thức main() để thao tác trên đối tượng GiaoDichHangHoa

```

using System;
namespace DaiHocThanhDo
{
    public class TestCsharp
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Interface trong C#");
            Console.WriteLine("Vi du minh hoa interface");
            Console.WriteLine("-----");

            //tao cac doi tuong GiaoDichHangHoa
            GiaoDichHangHoa t1 = new GiaoDichHangHoa("001", "8/10/2018", 78900.00);
            GiaoDichHangHoa t2 = new GiaoDichHangHoa("002", "9/10/2018", 451900.00);
            t1.hienThiThongTinGiaoDich();
            t2.hienThiThongTinGiaoDich();
            Console.ReadKey();
        }
    }
}

```

---

---

}

### **7.3 Bài tập cuối chương.**

Thực hiện tạo các giao diện (interface) cho các bài tập cuối chương 6, sau đó thực thi lại các giao diện này.

## Chương 8: CƠ CHẾ ỦY QUYỀN VÀ SỰ KIỆN

### 8.1. Cơ chế ủy quyền (delegate)

Delegate trong C# là tương tự như con trỏ tới các hàm, trong C hoặc trong C++. Một Delegate là một biến kiểu tham chiếu mà giữ tham chiếu tới một phương thức. Tham chiếu đó có thể được thay đổi tại runtime.

Đặc biệt, các delegate được sử dụng để triển khai các sự kiện và các phương thức call-back. Tất cả delegate được kế thừa một cách ngầm định từ lớp System.Delegate trong C#.

#### Khai báo Delegate trong C#

Khai báo Delegate trong C# quyết định các phương thức mà có thể được tham chiếu bởi Delegate đó. Một Delegate có thể tham chiếu tới một phương thức, mà có cùng dấu hiệu như của Delegate đó.

Ví dụ, bạn xét một delegate sau đây:

```
public delegate int MyDelegate (string s);
```

Delegate trên có thể được sử dụng để tham chiếu bất kỳ phương thức mà có một tham số string đơn và trả về một biến kiểu int.

Cú pháp để khai báo delegate trong C# là:

```
delegate <kiểu_trả_về> <tên_delegate> (<danh_sách_tham_số>)
```

#### Khởi tạo Delegate trong C#

Khi một kiểu delegate được khai báo, một đối tượng delegate phải được tạo với từ khóa new và được liên kết với một phương thức cụ thể. Khi tạo một delegate, tham số được truyền tới biểu thức new được viết tương tự như một lời gọi phương thức, nhưng không có tham số tới phương thức đó. Ví dụ:

```
public delegate void printString(string s);  
...  
printString ps1 = new printString(WriteToScreen);  
printString ps2 = new printString(WriteToFile);
```

Ví dụ sau minh họa cách khai báo, khởi tạo và sử dụng một delegate mà có thể được sử dụng để tham chiếu các phương thức mà nhận một tham số integer và trả về một giá trị integer.

```
using System;  
  
delegate int NumberChanger(int n);  
namespace DaiHocTahnhDo
```

```

{
    class TestCsharp
    {
        static int num = 10;
        public static int AddNum(int p)
        {
            num += p;
            return num;
        }

        public static int MultNum(int q)
        {
            num *= q;
            return num;
        }
        public static int getNum()
        {
            return num;
        }

        static void Main(string[] args)
        {
            Console.WriteLine("Vi du minh hoa Delegate trong C#");
            Console.WriteLine("-----");

            //tao cac doi tuong delegate
            NumberChanger nc1 = new NumberChanger(AddNum);
            NumberChanger nc2 = new NumberChanger(MultNum);

            //goi cac phuong thuc boi su dung cac doi tuong delegate
            nc1(25);
            Console.WriteLine("Gia tri cua num la: {0}", getNum());
            nc2(5);
            Console.WriteLine("Gia tri cua num la: {0}", getNum());
            Console.ReadKey();
        }
    }
}

```

## 8.2. Sự kiện (event)

Sự kiện (Event) là các hành động của người dùng, ví dụ như nhấn phím, click, di chuyển chuột, ... Các Application cần phản hồi các sự kiện này khi chúng xuất hiện. Ví dụ, các ngắt (interrupt). Các sự kiện (Event) được sử dụng để giao tiếp bên trong tiến trình.

### Sử dụng Delegate với Event trong C#

Các Event được khai báo và được tạo trong một lớp và được liên kết với Event Handler bởi sử dụng các Delegate bên trong cùng lớp đó hoặc một số lớp khác. Lớp mà chứa Event được sử dụng để công bố event đó. Điều này được gọi là lớp Publisher. Một số lớp khác mà chấp nhận Event này được gọi là lớp Subscriber. Các Event trong C# sử dụng mô hình Publisher-Subscriber.

Một Publisher trong C# là một đối tượng mà chứa định nghĩa của event và delegate đó. Mỗi liên hệ event-delegate cũng được định nghĩa trong đối tượng này. Một đối tượng lớp Publisher triệu hồi Event và nó được thông báo tới các đối tượng khác.

Một Subscriber trong C# là một đối tượng mà chấp nhận event và cung cấp một Event Handler. Delegate trong lớp Publisher triệu hồi phương thức (Event Handler) của lớp Subscriber.

Khai báo Event trong C#

Để khai báo một Event bên trong một lớp, đầu tiên một kiểu delegate cho Event đó phải được khai báo. Ví dụ:

```
public delegate void BoilerLogHandler(string status);
```

Tiếp theo, chính Event đó được khai báo, bởi sử dụng từ khóa event trong C#:

```
//định nghĩa event dựa vào delegate ở trên  
public event BoilerLogHandler BoilerEventLog;
```

Code trên định nghĩa một delegate với tên là BoilerLogHandler và một Event với tên là BoilerEventLog, mà triệu hồi delegate đó khi nó được tạo ra.

### Ví dụ 1

Tạo hai lớp có tên lần lượt là EventTest, TestCsharp như sau:

*Lớp EventTest:*

```
using System;  
namespace DaiHocThanhDo  
{  
    class EventTest  
    {  
        private int value;  
        public delegate void NumManipulationHandler();  
        public event NumManipulationHandler ChangeNum;  
        protected virtual void OnNumChanged()  
        {  
            if (ChangeNum != null)  
            {  
                ChangeNum();  
            }  
            else  
            {  
                Console.WriteLine("Kich hoạt su kien!");  
            }  
        }  
        public EventTest(int n)  
        {  
            SetValue(n);  
        }  
    }  
}
```

```

    }

    public void SetValue(int n)
    {
        if (value != n)
        {
            value = n;
            OnNumChanged();
        }
    }
}

```

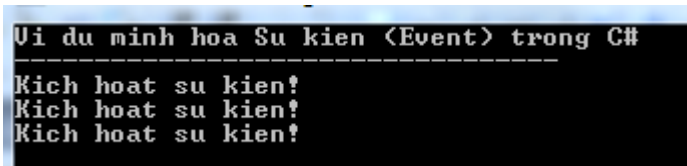
*Lớp TestCsharp:*

```

using System;
namespace DaiHocThanhDo
{
    class TestCsharp
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Vi du minh hoa Su kien (Event) trong C#");
            Console.WriteLine("-----");
            //tao doi tuong EventTest
            EventTest e = new EventTest(5);
            e.SetValue(7);
            e.SetValue(11);
            Console.ReadKey();
        }
    }
}

```

Khi biên dịch xong, kết quả:



```

Vi du minh hoa Su kien (Event) trong C#
-----
Kich hoạt su kien!
Kich hoạt su kien!
Kich hoạt su kien!

```

## Ví dụ 2

Ví dụ này cung cấp một ứng dụng đơn giản để xử lý sự cố cho một hệ thống nồi hơi đun nước nóng. Khi kỹ sư bảo dưỡng kiểm tra nồi hơi, nhiệt độ và áp suất nồi hơi được tự động ghi lại vào trong một log file cùng với các ghi chú của kỹ sư bảo dưỡng này.

Tạo 4 lớp có tên lần lượt là Boiler, DelegateBoilerEvent, BoilerInfoLogger, TestCsharp như sau:

*Lớp Boiler:*

```

using System;

```



```

namespace DaiHocThanhDo
{
    class Boiler
    {
        private int temp;
        private int pressure;
        public Boiler(int t, int p)
        {
            temp = t;
            pressure = p;
        }

        public int getTemp()
        {
            return temp;
        }

        public int getPressure()
        {
            return pressure;
        }
    }
}

```

Lớp *DelegateBoilerEvent*: đóng vai trò như là event publisher

```

namespace DaiHocThanhDo
{
    class DelegateBoilerEvent
    {
        public delegate void BoilerLogHandler(string status);

        //định nghĩa sự kiện dựa vào delegate trên
        public event BoilerLogHandler BoilerEventLog;

        public void LogProcess()
        {
            string remarks = "OK!";
            Boiler b = new Boiler(100, 12);
            int t = b.getTemp();
            int p = b.getPressure();
            if (t > 150 || t < 80 || p < 12 || p > 15)
            {
                remarks = "Can duy tri";
            }
            OnBoilerEventLog("Thong tin log:\n");
            OnBoilerEventLog("Nhiệt độ: " + t + "\nÁp suất: " + p);
            OnBoilerEventLog("\nThông báo: " + remarks);
        }

        protected void OnBoilerEventLog(string message)
        {
            if (BoilerEventLog != null)
            {
                BoilerEventLog(message);
            }
        }
    }
}

```

```
}  
}
```

*Lớp BoilerInfoLogger:*

```
using System.IO;  
  
namespace DaiHocThanhDo  
{  
    class BoilerInfoLogger  
    {  
        FileStream fs;  
        StreamWriter sw;  
        public BoilerInfoLogger(string filename)  
        {  
            fs = new FileStream(filename, FileMode.Append, FileAccess.Write);  
            sw = new StreamWriter(fs);  
        }  
  
        public void Logger(string info)  
        {  
            sw.WriteLine(info);  
        }  
  
        public void Close()  
        {  
            sw.Close();  
            fs.Close();  
        }  
    }  
}
```

*Lớp TestCsharp:*

```
using System;  
  
namespace DaiHocThanhDo  
{  
    class TestCsharp  
    {  
        static void Logger(string info)  
        {  
            Console.WriteLine(info);  
        }  
  
        static void Main(string[] args)  
        {  
            Console.WriteLine("Vi du minh hoa su kien trong C#");  
            Console.WriteLine("-----");  
  
            BoilerInfoLogger filelog = new BoilerInfoLogger("e:\\boiler.txt");  
            DelegateBoilerEvent boilerEvent = new DelegateBoilerEvent();  
            boilerEvent.BoilerEventLog += new  
            DelegateBoilerEvent.BoilerLogHandler(Logger);  
            boilerEvent.BoilerEventLog += new  
            DelegateBoilerEvent.BoilerLogHandler(filelog.Logger);  
        }  
    }  
}
```

```
        boilerEvent.LogProcess();  
        Console.ReadLine();  
        Console.ReadKey();  
        filelog.Close();  
    }  
}  
}
```

### 8.3 Bài tập cuối chương

Làm các bài tập trong Chương 6, sử dụng cơ chế ủy quyền và sự kiện.

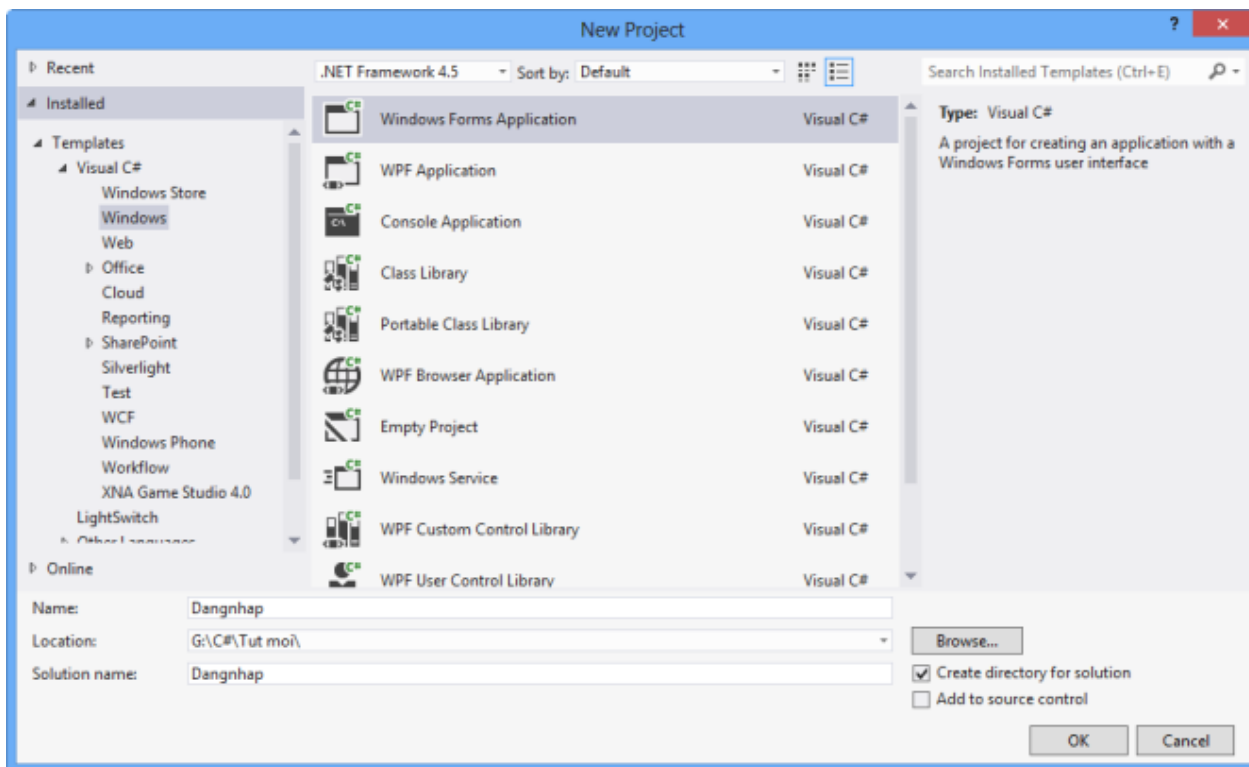
## Chương 9: CÁC ĐỐI TƯỢNG ĐIỀU KHIỂN TRÊN FORM

### 9.1. Biểu mẫu (form)

Lập Trình Xử Lý Giao Diện Trong Winform với C#. NET Framework cho ta ba cách để user giao diện với chương trình áp dụng, đó là Windows Forms (có khi được gọi tắt là WinForms), Web Forms và Console applications. Trong chương trình của chúng ta chỉ học phần Windows Form cho lập trình xử lý giao diện và các ứng dụng cơ sở dữ liệu và phần Console applications cho phần hướng dẫn về lập trình hướng đối tượng.

Một Windows Form thật sự là một class.

**Bước 1:** tạo project mới



**Bước 2:** các bạn design 2 form như dưới đây: form 1 là form login, form 2 là form hiển thị thông tin

Đăng nhập

Thông tin đăng nhập

Tên người dùng

Mật khẩu

Đăng nhập Thoát

Màn hình chính

**CHÀO MỪNG BẠN ĐẾN VỚI LẬP TRÌNH C#**

Tên đăng nhập của bạn: label4

Mật khẩu: label5

**Bước 3:** Ta bắt đầu xử lý các sự kiện cho từng form

```
private void btLogin_Click(object sender, EventArgs e)
{
    if (txtUser.Text == "" && txtPass.Text == "")
    {
        MessageBox.Show("User và pas không được trống");
    }
    else if (txtUser.Text=="")
    {
        MessageBox.Show("User không được trống");
    }
    else if (txtPass.Text=="")
    {
        MessageBox.Show("Pass không được trống");
    }
    else
    {
        if (txtUser.Text=="admin" && txtPass.Text=="12345")
        {
            Controller obj = new Controller();
            obj.User = "admin";
            obj.Pass = "12345";

            Thongbao f = new Thongbao(obj);
            f.Show();
        }
        else
        {
            MessageBox.Show("Tên đăng nhập hoặc mật khẩu sai, xin vui lòng nhập lại");
        }
    }
}
```

Tiếp theo là nút thoát

```
private void btExit_Click(object sender, EventArgs e)
{
    DialogResult traloi;
    traloi = MessageBox.Show("Chắc chắn không?", "Trả lời",
        MessageBoxButtons.OKCancel, MessageBoxIcon.Question);
    if (traloi==DialogResult.OK)
    {
        Application.Exit();
    }
}
```

-Để truyền 2 tham số là User và Pass qua form 2 thì các bạn cần tạo 1 lớp tôi gọi là Controller và tạo các thuộc tính như dưới

```
public class Controller
{
    public string User{set;get;}
    public string Pass { set; get; }
}
```

-Ở phần đăng nhập khi người dùng đăng nhập đúng thì ta sẽ cho show ra form2 đồng thời gán các thuộc tính bằng với các đối tượng mà ta đã tạo, đơn giản thôi ta chỉ cần gọi nó ra từ lớp Controller.

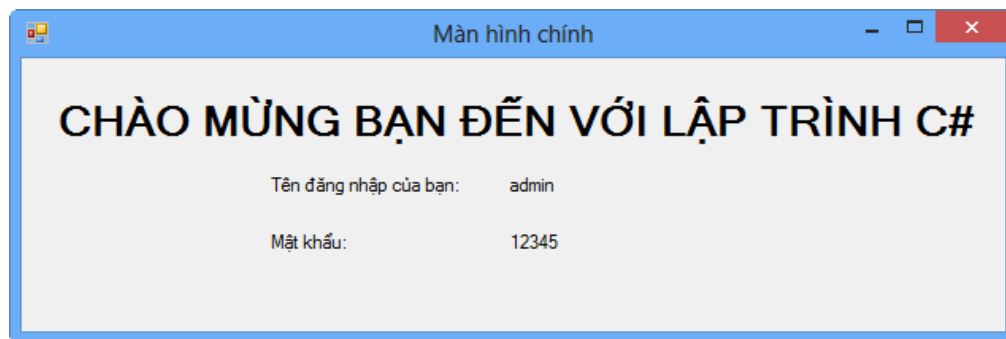
**Bước 4:** Tại form 2 ta cũng làm tương tự, để nó có thể load dữ liệu ngay khi mở form thì các bạn có thể xử lý sự kiện load form hoặc gọi ngay tại phương thức khởi tạo các thuộc tính của form.

```
public partial class Thongbao : Form
{
    private Controller obj;

    public Thongbao(Controller obj)
    {
        InitializeComponent();
        // Lưu lại thông tin của đối tượng truyền qua
        this.obj = obj;

        if (obj!=null)
        {
            lbUser.Text = obj.User;
            lbPass.Text = obj.Pass;
        }
    }
}
```

Hoàn thành các bạn có thể thấy được kết quả như hình dưới.



## 9.2. Điều khiển Label

Lớp Label được định nghĩa trong namespace System.Windows.Forms. Label là thành phần đơn giản nhất và cũng là một trong những thành phần quan trọng nhất và thường xuyên được sử dụng

nhất trong lập trình Winform. Đối tượng nhãn (Label) chỉ để dùng trình bày một chuỗi văn bản thông thường nhằm mục đích mô tả thêm thông tin cho các đối tượng khác. Ta cũng có thể dùng Label để làm công cụ đưa kết quả ra màn hình dưới dạng một chuỗi.

*Khởi tạo Label:* Click vào Label trong thanh ToolBox và kéo nó vào chỗ mình muốn trên form

Cũng như Button, Label cũng có các thuộc tính cơ bản như: Name, Text, BackColor, ForeColor, và nhiều thuộc tính khác để điều chỉnh kích thước và tọa độ, mình sẽ không trình bày lại...

Thuộc tính thường dùng của Label là Border Style, đại loại là thiết kế khung viền cho đẹp. Có ba kiểu Border Style (None, Fixed Single, Fixed3D) Kết quả như dưới:



*Sự kiện của label*

```
private void btnXanh_Click(object sender, System.EventArgs e)
{
    this.BackColor = Color.Blue;

    string msg = "";
    msg = msg + "Màu nền form: " + this.BackColor.ToString() + "\n";
    msg = msg + "Button click: " + btnXanh.Text;
    lblMsg.Text = msg;
}

private void btnDo_Click(object sender, System.EventArgs e)
{
    this.BackColor = Color.Red;

    string msg = "";
    msg = msg + "Màu nền form: " + this.BackColor.ToString() + "\n";
    msg = msg + "Button click: " + btnDo.Text;
    lblMsg.Text = msg;
}

private void btnMacDinh_Click(object sender, System.EventArgs e)
{
}
```



```

this.BackColor = Color.Empty;

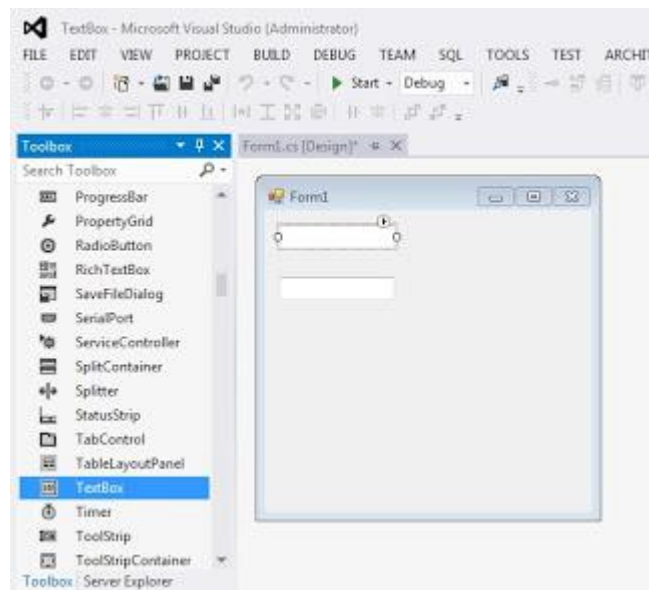
string msg = "";

msg = msg + "Màu nền form: " + this.BackColor.ToString() + "\n";
msg = msg + "Button click: " + btnMacDinh.Text;
lblMsg.Text = msg;
}

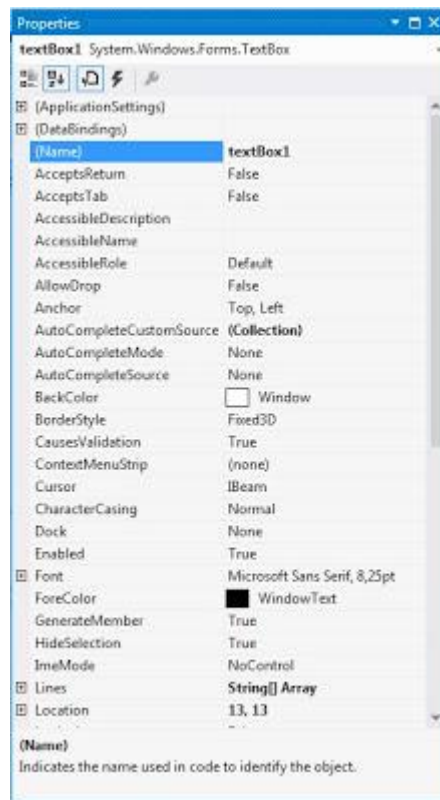
```

### 9.3. Điều khiển LinkLabel

### 9.4. Điều khiển TextBox



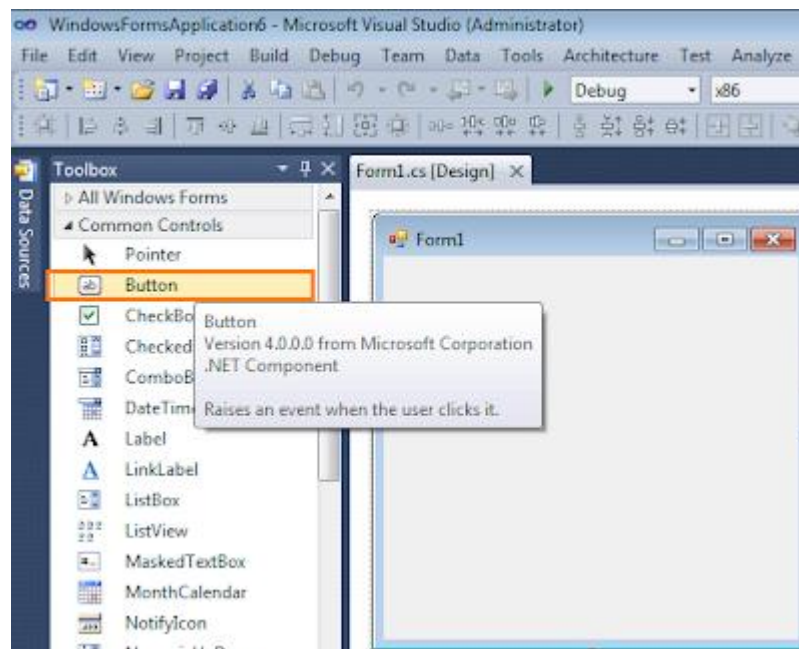
*Các thuộc tính của Textbox*



Tên thuộc tính	Chức năng
(name)	Tên của textbox (như kiểu tên biến vậy). Nó rất quan trọng
Autocomplete	(Sẽ nói ở dưới)
BackColor	Màu nền của textbox
BorderStyle	Kiểu viền textbox
Cursor	Con trỏ chuột khi rê chuột qua textbox
Enabled	Cho phép copy chữ và sửa trong textbox hay không (true/false)
Font	Các thuộc tính trong này cho phép thay đổi kiểu chữ, cỡ chữ...trong textbox
ForeColor	Màu chữ trong textbox

Multiline	Cho phép nhiều dòng
PaswordChar	Khi nhập vào sẽ được mã hóa thành dấu *
ReadOnly	Gần giống Enabled. Nhưng vẫn có thể bôi đen copy được
ScrollBars	Khi chữ vượt quá kích thước textbox thì sẽ xuất hiện thanh cuộn. Nên bật Multiline
Text	Nội dung (giá trị) trong textbox. Kiểu dữ liệu: string
Visible	True: hiện trên form. False: Ẩn trên form
WordWrap	Ngắt dòng khi chạm lề

## 9.5. Điều khiển Button



*Các thuộc tính của button*

Tên thuộc tính	Chức năng
BackColor	màu nền của button
BackgroundImage	Ảnh nền button
Cursor	hình con trỏ chuột khi di qua button
Enabled	cho phép click vào nút (true) hoặc không cho phép Click vào nút (false)
FlatStyle	chọn kiểu của nút. Cái này có kiểu Flat rất đẹp@!
Font	các thuộc tính trong phần này cho phép thay đổi kiểu chữ, cỡ chữ...hiển thị trên button
ForeColor	màu chữ hiển thị trên button
Text	Nội dung chữ hiển thị trên button
(name)	tên của button. Cái này quan trọng, nó có thể con như tên biến vậy
TextAlign	Căn lề của chữ trên button

#### 9.6. Điều khiển CheckBox

#### 9.7. Điều khiển RadioButton

#### 9.8. Điều khiển PictureBox

## 9.9 Bài tập cuối chương.

Sử dụng Winform để giải các bài tập trong chương 5,6.

### Đề thi mẫu

#### ĐỀ 1

**Câu 1.**(2 điểm) Viết chương trình bằng ngôn ngữ C# thực hiện:

- Nhập vào một mảng số nguyên, mỗi phần tử phải nằm trong khoảng [5,100], ngoài khoảng đó thì thông báo nhập lại. Sau đó in ra màn hình mảng vừa nhập. (1 điểm)
- Đếm số phần tử là số chẵn trong mảng, tính tổng của các phần tử là số chẵn. (1 điểm)

**Câu 2** (3 điểm). Viết chương trình bằng ngôn ngữ C# thực hiện:

- Viết hàm đọc số nguyên trong đoạn [0,9] (1,5 điểm)
- Chỉ cho phép nhập vào số nguyên có 3 chữ số. Đọc số đó ra màn hình. (1,5 điểm)

**Câu 3.**(5 điểm).Viết chương trình bằng ngôn ngữ C# thực hiện:

- Tạo interface gồm các thuộc tính  $ia, ib, ic$ , phương thức  $Kiemtratamgiac()$ ,  $Ketluantamgiac()$ ,  $TinhCV()$ ,  $TinhDienTich()$ . (2,5 điểm)
- Tạo lớp thực thi interface trên. Viết chương trình nhập 3 số thực, tiến hành xem 3 số đó có phải là 3 cạnh trong một tam giác (dùng phương thức  $Kiemtratamgiac()$ ), là tam giác gì (dùng phương thức  $Ketluantamgiac()$ ) và tính chu vi ( $TinhCV()$ ), diện tích ( $TinhDienTich()$ ) của tam giác đó. (2,5 điểm)

#### ĐỀ 2.

**Câu 1.**(2 điểm) Viết chương trình bằng ngôn ngữ C# thực hiện:

- Hiện thị một hình chữ nhật rỗng giữa được vẽ bằng các ký tự (\*), chiều dài và rộng nhập vào từ bàn phím. (1 điểm)
- Vẽ tam giác cân bằng các ký tự (\*), chiều cao bằng chiều dài của hình chữ nhật trên. (1 điểm)

**Câu 2** (3 điểm). Viết chương trình bằng ngôn ngữ C# thực hiện:

- Viết hàm đọc số nguyên trong đoạn [0,9] (1,5 điểm)
- Chỉ cho phép nhập vào số nguyên có 3 chữ số. Đọc số đó ra màn hình. (1,5 điểm)

**Câu 3.**(5 điểm).Viết chương trình bằng ngôn ngữ C# thực hiện:

- Tạo interface  $iSV$  có các thuộc tính:  $hoten, diem1, sotinchi1, diem2, sotinchi2, diem3, sotinchi3$ , và các phương thức  $TinhDTB()$ ,  $Hocbang()$  (2,5 điểm)
- Tạo lớp  $clsSV$  thực thi interface trên. Nhập 1 danh sách từ 2 sinh viên trở lên. In ra danh sách thông tin sinh viên và tên sinh viên có điểm cao nhất. (2,5 điểm)



# Danh sách bài tập lớn

## Môn học: Lập trình Windows 1

**Bài tập 1.** Nhóm bạn hãy viết chương trình tạo ra một cái máy tính điện tử thông thường, có chức năng ở rộng ra máy tính có nhiều chức năng hơn.

**Bài tập 2.** Nhóm hãy viết chương trình mô phỏng phần mềm word, có chức năng tự chuẩn hóa, lưu, hiển thị, soạn thảo văn bản, điều chỉnh được khoảng cách giữa các đoạn văn bản, có chức năng xem những file mới soạn thảo...

**Bài tập 3.** Nhóm hãy viết chương trình để quản lý file và thư mục, có chức năng phân loại file và tự động lưu các loại file đó vào thư mục khác nhau.

**Bài tập 4.** Nhóm hãy viết chương trình để quản lý hồ sơ học sinh, hồ sơ cần lưu trữ, truy xuất được thông tin họ tên, ngày sinh, quê, lớp học, ngành học, năm vào trường, dự kiến năm ra trường.

**Bài tập 5.** Nhóm hãy viết chương trình cho phép người dùng thi trắc nghiệm môn lập trình windows 1 phần lý thuyết, mỗi câu có 4 phương án trả lời, 1 đáp án. Gợi ý: Hãy lưu mỗi câu trên 1 dòng của file text.

**Bài tập 6.** Viết chương trình cho phép người dùng tạo trò chơi Picachu với các hình ảnh các con vật picachu do người dùng đưa vào. Luật của trò chơi, chỉ cần kích vào 2 hình ảnh giống nhau thì các hình ảnh được kích sẽ biến mất và người chơi được 1 điểm. Gợi ý, mỗi picachu hiển thị trên 1 button.

**Bài tập 7.** Nhóm hãy viết chương trình cho phép người dùng nhập ngày, tháng, năm sinh. Hãy “bói” cho người đó với thông tin nhập vào, biết rằng năm sinh được hợp thành 2 yếu tố, thiên can và địa chi. Gợi ý: sẽ có 60 kết quả được tìm thấy, nhóm hãy thu thập 60 file thông tin, khi người dùng nhập vào sẽ hiển thị tương ứng.

**Bài tập 8.** Nhóm hãy viết chương trình hỗ trợ học sinh cấp 1 học toán.

**Bài tập 9.** Nhóm hãy viết chương trình quản lý file và thư mục, cho phép tạo file, chép file, tạo thư mục, chép thư mục, xóa file, xóa thư mục.... trên máy tính. Nhóm tự tạo các mô hình quản lý thư mục tối ưu gợi ý người dùng.

**Bài tập 10.** Nhóm hãy viết chương trình mô tả trò chơi caro.

**Bài tập 11.** Nhóm hãy viết chương trình hỗ trợ giải toán đại số lớp 6.

***Yêu cầu chung:** Mỗi nhóm 3 thành viên, cần vẽ sơ đồ thuật toán, có có interface, class tương ứng, nên viết trên form.*

**LỊCH: 19/02/2018-----> BÁO CÁO TIẾN ĐỘ BÀI TẬP LỚN**

**05/05----->BV BÀI TẬP LỚN**

---

**Nội dung: ghi trên file word, slide, nộp project ngày 21/08 nén project, file word gửi vào  
hòm thư tnhoang@thanhdo.edu.vn**



## **Tài liệu tham khảo**

1. Trang web của Microsoft (<https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/>)
2. FUNDAMENTALS OF COMPUTER PROGRAMMING WITH C#, Svetlin Nakov & Co, 2013
3. <https://www.w3resource.com/csharp-exercises/basic-algo/index.php>
4. Trang tài liệu: tinhoccoban.net.