

NGÔN NGỮ LẬP TRÌNH C#

Nội dung

- ❑ Ứng dụng Console Application
- ❑ Các thành phần cơ bản của ngôn ngữ C#
- ❑ Kiểu dữ liệu (Data Type), Biến (Variable) và Hằng (Constant)
- ❑ Các toán tử (Operators)
- ❑ Câu lệnh trong C#
 - ❑ Phân loại lỗi
 - ❑ Xử lý lỗi
- ❑ Kiểu cấu trúc - struct
- ❑ Kiểu mảng - Array
- ❑ Collections và Generic Collections
- ❑ Tập tin

Ứng dụng Console Application

- ❑ Các không gian tên đính kèm
- ❑ Cả project ⇒ không gian tên
 - Lớp Program chứa phương thức Main.
 - Trong Main, các câu lệnh thực hiện yêu cầu cụ thể.
 - Từ khoá “static” có nghĩa rằng phương thức Main được gọi không cần phải đối tượng lớp này. Nó mà một phương thức của class, không phải phương thức của đối tượng

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            //Statement 1
            //Statement 2
            //...
            //Statement n
        }
    }
}
```

Các thành phần cơ bản của ngôn ngữ C#

- **Bộ ký tự dùng trong C# (Character set):** Được xây dựng trên bộ ký tự chuẩn quốc tế Unicode gồm
 - 26 chữ cái hoa A...Z và 26 chữ cái thường a...z
 - 10 chữ số 0...9
 - Các ký hiệu toán học + - * / % = ()...
 - Dấu nối _
 - Các ký hiệu đặc biệt khác ; : {} [] ? \ & | # \$...

Các thành phần cơ bản của ngôn ngữ C#

- **Từ khoá (Keyword):** Định nghĩa trước trong C#, có ý nghĩa xác định, phải dùng đúng cú pháp, viết bằng chữ thường, không dùng vào đặt tên mới.
 - Từ khoá khai báo: namespace, public, private, static, const, class, new...
 - Từ khoá điều khiển: switch, case, break, if, return, for, while, continue, try, catch...
 - Từ khoá toán tử: is
 - Từ khoá kiểu dữ liệu: bool, byte, sbyte, short, ushort, float, double, null, void, ...

Các thành phần cơ bản của ngôn ngữ C#

□ Định danh (Identifier)

- Phân biệt chữ hoa, thường
- Bắt đầu bằng ký tự chữ cái hoặc dấu gạch nối dưới _, các ký tự còn lại phải là ký tự chữ cái, chữ số, dấu gạch nối dưới.
- Không được đặt tên trùng với từ khóa
- Đặt tên nên theo quy tắc đặt tên định nghĩa sẵn của C# cho dễ nhớ là tất cả các tên của lớp, phương thức, biến, giao tiếp, không gian tên... đều viết chữ hoa đầu từ, ngoại trừ tên hằng viết chữ hoa.

Các thành phần cơ bản của ngôn ngữ C#

❑ Lời chú thích (Comment)

- Lời chú thích thêm vào chương trình với mục đích giải thích, giúp cho người lập trình dễ dàng bổ sung, sửa chữa, nâng cấp chương trình. Khi chạy chương trình, trình dịch sẽ bỏ qua chú thích.
- Chú thích gồm một hay nhiều dòng: bắt đầu chú thích với dấu /* và kết thúc bởi */
- Chú thích một dòng: bắt đầu dòng chú thích bởi dấu //

Kiểu dữ liệu (Data type)

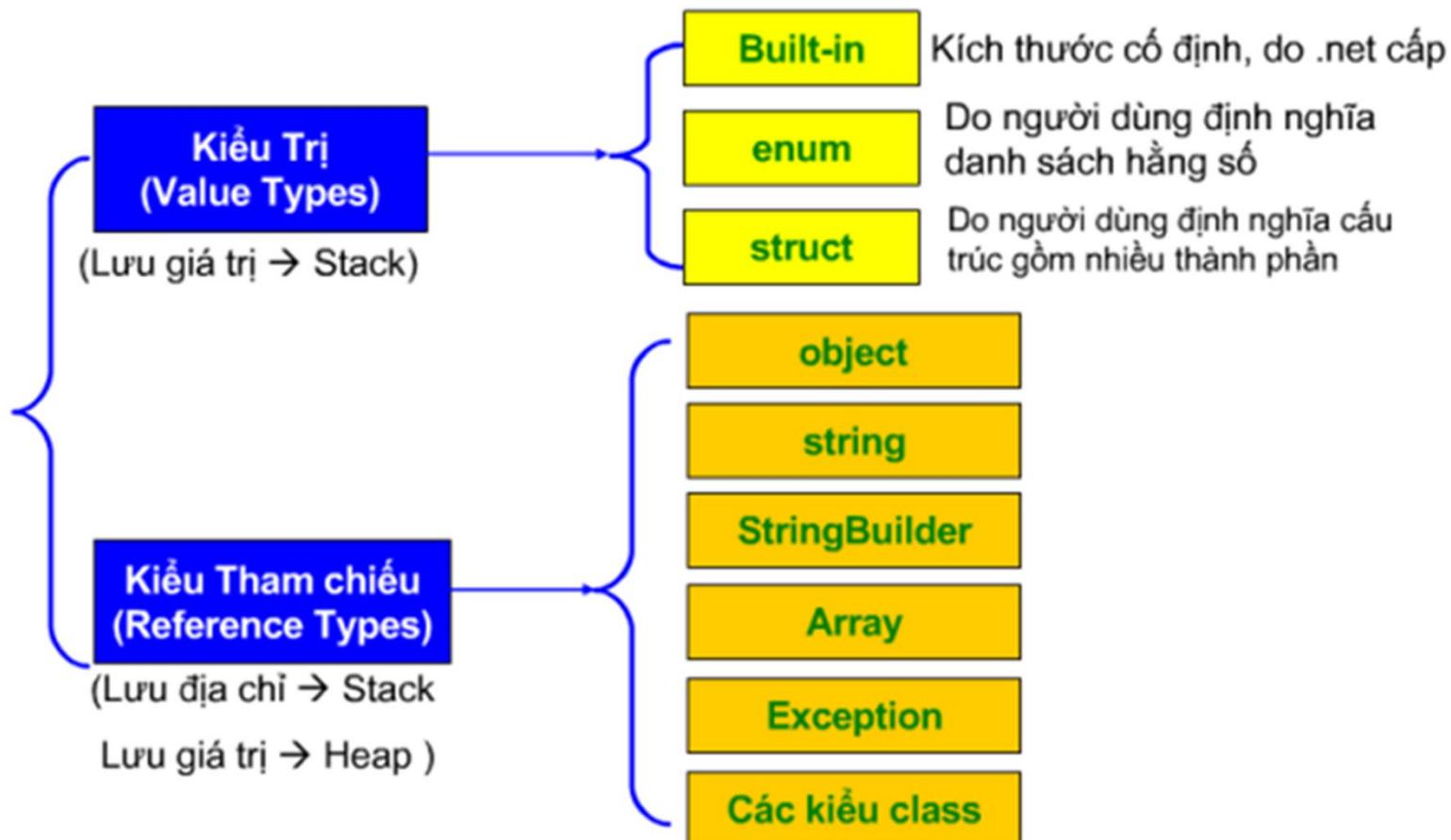
❖ Khái niệm:

- ❑ Là tập hợp các giá trị mà một biến thuộc kiểu đó có thể nhận được.
- ❑ Kiểu dữ liệu trong .NET được mô tả chi tiết trong Common Type System (CTS)
- ❑ Mỗi kiểu trong .NET là một đối tượng, nghĩa là nó có các thuộc tính và phương thức riêng. Một trong những phương thức thường dùng nhất là Parse và ToString

Kiểu dữ liệu (Data type)

- ❖ Các kiểu dữ liệu trong C# được phân làm 2 loại: Kiểu trị (Value Types) và kiểu tham chiếu (Reference Types)
- ❖ Việc phân chia này do sự khác nhau khi lưu kiểu dữ liệu giá trị và kiểu dữ liệu tham chiếu trong bộ nhớ. Cụ thể là:
 - ❑ Biến kiểu dữ liệu giá trị lưu giữ giá trị của nó trong bộ nhớ tĩnh (stack).
 - ❑ Trong khi đó biến kiểu tham chiếu lưu một địa chỉ hay tham chiếu đến đối tượng trong bộ nhớ stack, còn đối tượng thật sự lưu trong bộ nhớ động (heap).

Kiểu dữ liệu (Data type)



Kiểu dữ liệu cơ sở

- ❖ Ngôn ngữ C# đưa ra các kiểu dữ liệu cơ sở rất hữu dụng, mỗi kiểu dữ liệu được ánh xạ đến một kiểu dữ liệu hỗ trợ bởi .NET.
- ❖ Việc ánh xạ các kiểu dữ liệu cơ sở của C# đến các kiểu dữ liệu của .NET sẽ đảm bảo các đối tượng được tạo ra trong C# có thể được sử dụng đồng thời với các đối tượng được tạo bởi bất kỳ ngôn ngữ khác được biên dịch bởi .NET.

Kiểu dữ liệu cơ sở

Kiểu/Alias C#	Kích thước	Miền giá trị (Range)
<code>System.SByte</code> / <code>sbyte</code>	1 byte	-128 → 127
<code>System.Byte</code> / <code>byte</code>	1 byte	0 → 255
<code>System.Int16</code> / <code>short</code>	2 bytes	-32768 → 32767
<code>System.Int32</code> / <code>int</code>	4 bytes	-2147483648 → 2147483647
<code>System.UInt32</code> / <code>uint</code>	4 bytes	0 → 4294967295
<code>System.Int64</code> / <code>long</code>	8 bytes	-9223372036854775808 → 9223372036854775807
<code>System.Single</code> / <code>float</code>	4 bytes	-3.402823E+38 → 3.402823E+38
<code>System.Double</code> / <code>double</code>	8 bytes	-1.79769313486232E+308 → 1.79769313486232E+308
<code>System.Decimal</code> / <code>decimal</code>	16 bytes	-79228162514264337593543950335 → 79228162514264337593543950335
<code>System.Char</code> / <code>char</code>	2 bytes	N/A
<code>System.Boolean</code> / <code>bool</code>	4 bytes	N/A
<code>System.DateTime</code>	8 bytes	1/1/0001 12:00:00 AM → 12/31/9999 11:59:59 PM

Kiểu liệt kê (enum)

- ❖ Là tập hợp các tên hằng có giá trị không thay đổi
- ❖ Để định nghĩa một kiểu liệt kê ta thực hiện theo cú pháp sau:

```
enum TênKiểuLiệtKê [KiểuCơSở]
{
    //Danh sách các thành phần liệt kê
};
```

Kiểu liệt kê (enum)

- ❖ Danh sách liệt kê là các hằng được gán giá trị và mỗi thành phần phải phân cách nhau dấu phẩy.
- ❖ Nếu không khởi tạo giá trị cho các thành phần này thì chúng sẽ nhận các giá trị liên tiếp bắt đầu từ 0.
- ❖ Mỗi tên hằng trong kiểu liệt kê thuộc bất kỳ một kiểu dữ liệu cơ sở nguyên nào như int, short, long..., ngoại trừ kiểu ký tự.
- ❖ Khi truy xuất giá trị của danh sách liệt kê, cần phải chuyển kiểu một cách tường minh.

Kiểu liệt kê (enum)

❖ Cách sử dụng:

<tên enum> <tên biến> = <tên enum>.<hằng số>;

Kiểu liệt kê (enum)

❖ Ví dụ:

```
using System;
class Example
{
    enum VisibleType {None = 0, Hidden = 2, Visible = 4};
    // In ra tên và giá trị thành phần của enum
    static void Main(string[] args)
    {
        VisibleType visEnum = VisibleType.Hidden;
        Console.WriteLine(visEnum.ToString());
        Console.WriteLine((int)VisibleType.Hidden);
    }
}
```

Kiểu dữ liệu tham chiếu (Reference types)

- ❖ Các kiểu tham chiếu thường dùng:

Kiểu	Ý nghĩa sử dụng
<i>System.Object</i>	Kiểu tổng quát
<i>System.String</i>	Dữ liệu dạng Text
<i>System.Text.StringBuilder</i>	Dữ liệu dạng Dynamic text
<i>System.Array</i>	Mảng dữ liệu
<i>System.IO.Stream</i>	Bộ đệm (Buffer) cho tập tin, thiết bị
<i>System.Exception</i>	Kiểm soát lỗi trong quá trình thực thi UĐ

Kiểu chuỗi

- ❖ Kiểu dữ liệu chuỗi (string hay String) là kiểu tham chiếu xây dựng sẵn, lưu giữ một dãy ký tự.
- ❖ Tạo một đối tượng chuỗi:

```
String s = "Hello World";
```

```
Hoặc string s = "Hello World";
```

Khởi tạo chuỗi chứa 2 ký tự ‘a’

```
string s = new string('a', 2); //s = “aa”
```

Khởi tạo chuỗi từ mảng ký tự

```
char[] ch = { 'a', 'b', 'c' };
```

```
string s = new string(ch); //s = “abc”
```

Khởi tạo chuỗi từ 2 phần tử trong mảng ch, bắt đầu từ phần tử thứ 1:

```
string s = new string(ch, 1, 2); //s = “bc”
```

Kiểu chuỗi

- ❖ Tạo chuỗi động bằng đối tượng StringBuilder

```
System.Text.StringBuilder sb = new System.Text.StringBuilder(30);
sb.Append("Trung tam");
sb.Append(" Chuan dau ra");
sb.Append(" DH");
sb.Append(" Nam Can Tho");
string s = sb.ToString();
label1.Text = s;
```

Kiểu chuỗi

- ❖ Để chuyển 1 kiểu dữ liệu nào đó về String, sử dụng phương thức đổi tương ToString() của đối tượng cần chuyển kiểu, hay phương thức tĩnh ToString của lớp Convert.

```
int Number = 10;  
String Str = Number.ToString();  
Str = Convert.ToString(Number);
```

- ❖ Truy cập các ký tự của chuỗi bằng chỉ số và sử dụng dấu [] như mảng, chỉ số có giá trị bắt đầu từ 0

Kiểu chuỗi

- ❖ Các thuộc tính và phương thức thường dùng của string
 - ❑ Xem thêm tài liệu tham khảo

Biến (Variable) & Hằng (Constant)

- ❑ Khái niệm
- ❑ Khai báo
- ❑ Chuyển đổi giữa các kiểu dữ liệu
- ❑ Biến cục bộ được khai báo với từ khóa var

Biến (Variable) & Hằng (Constant)

□ Khái niệm Biến

- **Biến** là đối tượng dùng để lưu trữ tạm thời các giá trị trong quá trình xử lý tính toán
- Một số lưu ý khi đặt tên cho Biến:
 - Tên có phân biệt chữ HOA, chữ thường
 - Tuân theo quy tắc đặt tên:
 - Không có khoảng trắng
 - Không có dấu
 - Không đặt trùng với các từ khóa
 - Không sử dụng các ký tự đặt biệt (#, \$, %, +, -, ...)
 - Không bắt đầu bằng số
 - Tên có thể bắt đầu bằng ký tự _

Biến (Variable) & Hằng (Constant)

□ Khai báo Biến

- Cú pháp: **Kiểu_dữ_liệu Tên_biến;**
 - Ví dụ:

```
int tuoi;  
string dia_chi;
```
- Gán giá trị cho biến: **Tên_biến = giá trị;**
 - Ví dụ:

```
tuoi = 15;  
dia_chi="357 Lê Hồng Phong";
```
- Chú ý: có thể khai báo và khởi tạo giá trị cho biến cùng lúc.
 - Ví dụ: double diem=7.5;

Biến (Variable) & Hằng (Constant)

□ Khái niệm Hằng

- Hằng là những giá trị không thay đổi trong suốt quá trình hoạt động của ứng dụng

□ Khai báo Hằng

- Cú pháp:

const <Kiểu_dữ_liệu> <Tên_hằng> = Giá trị;

- Ý nghĩa

- const: từ khoá khai báo hằng số

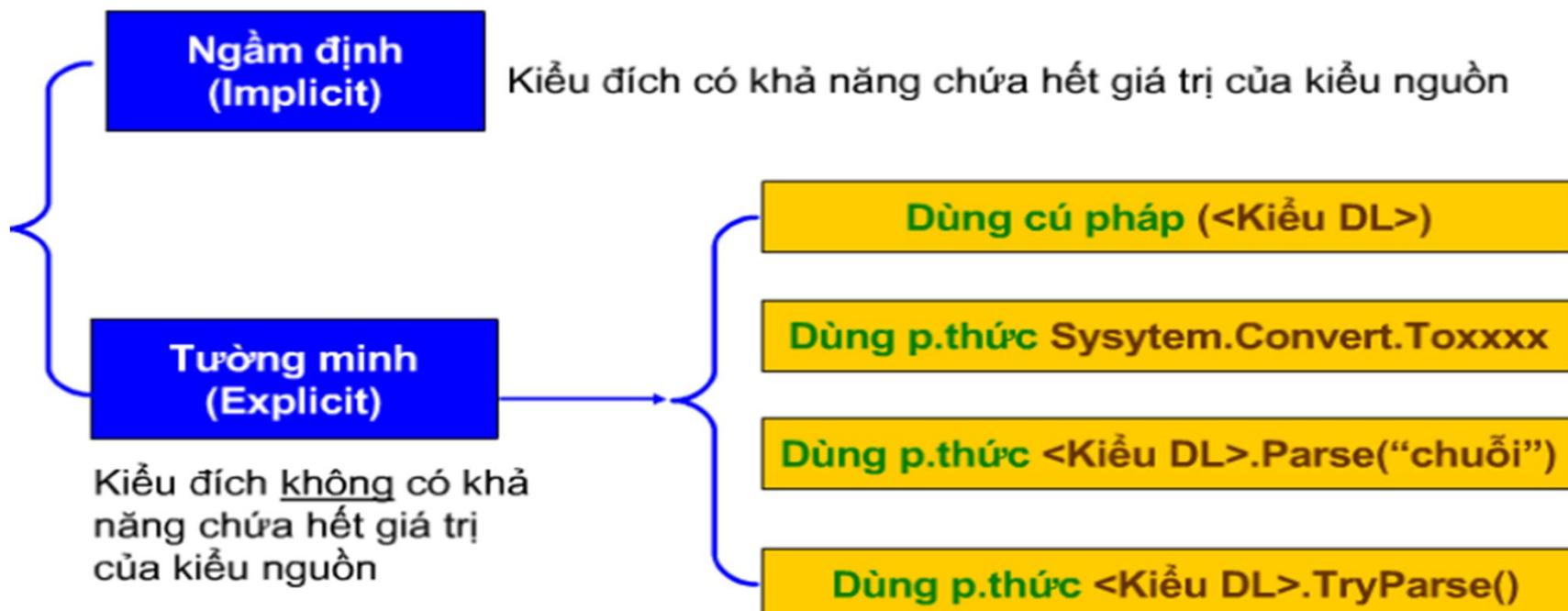
- Ví dụ:

const double PI = 3.14;

Biến (Variable) & Hằng (Constant)

□ Chuyển đổi giữa các kiểu dữ liệu

- C# cho phép chuyển đổi giữa 2 kiểu theo 2 cách là ngầm định (Implicit Conversions) và tường minh (Explicit Conversions)



Biến (Variable) & Hằng (Constant)

□ Chuyển đổi giữa các kiểu dữ liệu

- Ví dụ: chuyển kiểu ngầm định

```
int i = 1;  
double d ;  
d = i ;           //biên dịch không lỗi  
//nhưng nếu  
i = d ;           //biên dịch có lỗi
```

- Ví dụ: chuyển kiểu tường minh

```
int i;  
double d = 1.0025 ;  
i = (int) d ;           //biên dịch không lỗi và i = 1  
//Hoặc  
i = System.Convert.ToInt32(d);
```

Biến (Variable) & Hằng (Constant)

▫ Biến cục bộ được khai báo với từ khóa var

- Cú pháp: **var <tên biến> = <biểu thức>;**
- Phải được khởi tạo giá trị lúc khai báo biến
- Compiler sẽ xác định kiểu dữ liệu từ <biểu thức>
- Không thể gán **null** cho biến khai báo var
- Ví dụ:

```
var i = 5;  
var s = "Hello";  
var d = 1.0;  
//Tương đương với khai báo tường minh sau:  
int i = 5;  
string s = "Hello";  
double d = 1.0;
```

Các toán tử (Operators)

□ Toán tử gán

- `=, +=, -=, *=, /=, %=`

□ Toán tử luận lý

- `&& (And), || (Or) và ! (Not)`

□ Toán tử so sánh

- `==, >=, <=, >, < và !=`

□ Toán tử số học

- `+, -, *, / và %`

□ Toán tử nối chuỗi: `+`

□ Toán tử điều kiện: `<điều kiện đúng>?<giá trị true>:<giá trị false>`

Nhập xuất với lớp Console

- ❖ Lớp Console định nghĩa một tập các phương thức tĩnh để thực hiện việc nhập dữ liệu từ bàn phím và xuất ra màn hình.
 - ❑ Phương thức Write() xuất một giá trị hay đối tượng ra màn hình. Nếu xuất đối tượng, phương thức ToString() của đối tượng sẽ chuyển đổi tượng thành chuỗi và xuất ra màn hình.
 - ❑ Phương thức WriteLine() xuất một giá trị hay đối tượng ra màn hình, sau đó xuống dòng.
 - ❑ Phương thức string ReadLine() trả về chuỗi nhập từ bàn phím, kết thúc bởi phím xuống dòng.
 - ❑ Phương thức int Read() trả về ký tự nhập từ bàn phím.

Nhập xuất với lớp Console

- ❖ Ví dụ: Viết chương trình nhập vào họ tên, và năm sinh với điều kiện họ tên không quá 25 ký tự, và năm sinh từ 1980 đến 1985.

```
using System;
class Program
{
    static void Main(string[] args)
    {
        ConfigureCUI();
        Console.ReadLine();
    }
    private static void ConfigureCUI()
    {
        string s = null;
        do {
            Console.Write("Nhập Họ và tên không
qua 25 ký tự:");
            s = Console.ReadLine();
        }while (s.Length>25 || s.Length==0);
        Console.WriteLine("Họ và tên là : {0}",s);
    }
}
```

```
//Nhập năm sinh
while (true) {
    try {
        Console.Write("Nhập năm sinh : ");
        s = Console.ReadLine();
        int ns = int.Parse(s);
        if (ns<1980 || ns>1985)
            throw new
FormatException();
        Console.WriteLine("Năm
sinh là : "+ns);
        break;
    }
    catch (FormatException e1) {
        Console.WriteLine("Nhập
năm sinh từ 1980 -1985");
    }
}
```

Định dạng kết xuất

- ❖ Lớp Console cung cấp một số đặc tính định dạng kết xuất sau:
 - ❑ ConsoleColor, ConsoleColor: hai thuộc tính này cho phép bạn thiết lập màu nền và màu chữ cho kết xuất ra màn hình.
 - ❑ BufferHeight, BufferWidth: hai thuộc tính này điều chỉnh chiều cao và chiều rộng cho buffer của console.
 - ❑ Clear(): phương thức xóa buffer và vùng hiển thị của console.
 - ❑ WindowHeight, WindowWidth, WindowTop, WindowLeft: các thuộc tính này điều khiển các chiều cao, rộng, lề trên, lề trái của console so với buffer.

Định dạng kết xuất

❖ Ví dụ:

```
using System;
class Program
{
    static void Main(string[] args)
    {
        ConfigureCUI();
    }
    private static void ConfigureCUI()
    {
        Console.Title = "My Application";
        Console.ForegroundColor = ConsoleColor.Yellow;
    }
}
```

```
Console.BackgroundColor = ConsoleColor.Blue;
Console.WriteLine("*****");
*****;
Console.WriteLine("****");
Welcome to My Application ****);
Console.WriteLine("*****");
*****;
```

Thiết lập vị trí kết xuất

- ❖ Có thể thiết lập vị trí kết xuất cho phương thức WriteLine trên màn hình. Ví dụ:

```
using System;
class Program
{
    static void Main(string[] args){
        int theInt = 90;
        double theDouble = 9.99;
        bool theBool = true;
        Console.CursorLeft = 10;
        Console.CursorTop = 5;
        //Console.SetCursorPosition(10,5);
        Console.WriteLine("Int is: {0}\nDouble is: {1}\nBool is: {2}",
            theInt, theDouble, theBool);
    }
}
```

Thiết lập vị trí kết xuất

- ❖ Phương thức WriteLine() cũng cho phép bạn truyền vào một mảng các đối tượng

```
object[] stuff = {"Hello", 20.9, 1, "There", "83", 99.99933};  
Console.WriteLine("The Stuff: {0} , {1} , {2} , {3} , {4} , {5}  
", stuff);
```

- ❖ Nó cũng cho phép đặt cùng một đánh dấu ở nhiều nơi:

```
Console.WriteLine("{0}, Number {0}, Number {0}", 9);
```

Thiết lập vị trí kết xuất

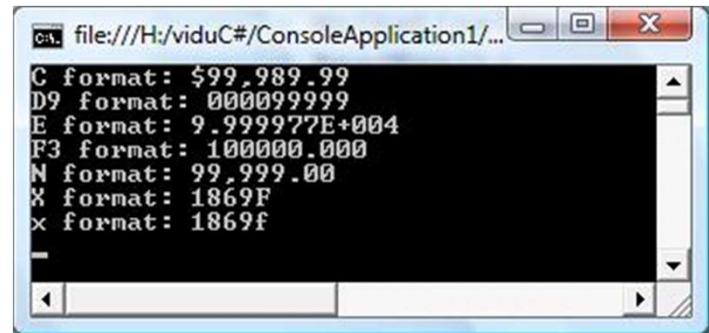
- ❖ Nếu truyền giá trị không khớp với các đánh dấu, sẽ phát sinh ngoại lệ FormatException
- ❖ Nếu bạn muốn định dạng thêm nhiều kiểu khác nữa, có thể dùng các cờ sau (viết hoa hay thường đều được), được viết sau dấu hai chấm sau ký hiệu đánh dấu ({0:C}, {1:d}, {2:X},...):
 - ❑ C hay c: Sử dụng để định dạng tiền tệ.
 - ❑ D hay d: Sử dụng để định dạng số thập phân.
 - ❑ E hay e: Sử dụng để thể hiện dạng số mũ.
 - ❑ F hay f: Sử dụng cho định dạng dấu chấm tinh.
 - ❑ G hay g: Viết tắt của general. Ký tự này dùng để định dạng kiểu chấm tinh hay số mũ.
 - ❑ N hay n: Sử dụng định dạng phần ngàn (với dấu phẩy)
 - ❑ X hay x: Sử dụng định dạng thập lục phân. Nếu bạn dùng X viết hoa thì ký tự thập lục cũng sẽ được viết hoa.

Thiết lập vị trí kết xuất

❖ Ví dụ:

```
using System;
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("C format: {0:C}", 99989.987);
        Console.WriteLine("D9 format: {0:D9}", 99999);
        Console.WriteLine("E format: {0:E}", 99999.76543);
        Console.WriteLine("F3 format: {0:F3}", 99999.9999);
        Console.WriteLine("N format: {0:N}", 99999);
        Console.WriteLine("X format: {0:X}", 99999);
        Console.WriteLine("x format: {0:x}", 99999);
    }
}
```

Kết quả:



Câu lệnh if

Cú pháp:

Dạng 1: **if** (BiểuThứcĐiềuKiện)

Lệnh;

Dạng 2: **if** (BiểuThứcĐiềuKiện)

Lệnh1;

else Lệnh2;

- ❖ *Ý nghĩa:* Nếu biểu thức điều kiện thỏa mãn (có giá trị true) thì lệnh 1 thực hiện, ngược lại nếu biểu thức điều kiện không thỏa mãn (có giá trị false) thì không làm gì cả (với dạng 1) hay thực hiện lệnh 2 (với dạng 2).

Câu lệnh if

Ví dụ: Giải phương trình bậc 2 : $ax^2 + bx + c = 0$

```
using System;
class PTB2
{
    static void Main(string[] args)
    {
        Console.Write("Nhập hệ số a:");
        string s=Console.ReadLine(); double a=double.Parse(s);
        Console.Write("Nhập hệ số b:");
        s = Console.ReadLine(); double b=double.Parse(s);
        Console.Write("Nhập hệ số c:");
        s = Console.ReadLine(); double c=double.Parse(s);
        if (a == 0)
```

```
            if (b == 0)
                if (c == 0) s="co vo so nghiem";
                else s = "vo nghiem";
                else s = "co mot nghiem la "+(-c/b);
            else
            {
                double delta=b*b-4*a*c;
                if (delta<0) s = "vo nghiem";
                else if (delta==0)
                    s = "co nghiem kep la "+(-b/(2*a));
                else s= "co 2 nghiem
                    x1= " + ((-b+ Math.Sqrt(delta))/(2*a)) + " và x2= "+((-b-Math.Sqrt(delta))/(2*a));
            }
            Console.WriteLine("Phương trình " + s);
        }
```

Câu lệnh switch

- ❖ switch là câu lệnh lựa chọn, cho phép rẽ nhánh thực hiện lệnh theo nhiều hướng khác nhau căn cứ trên giá trị của một biểu thức.

Cú pháp:

```
switch (BiểuThứcĐiềuKiện)
{
    case Biểu thức 1:
        Lệnh 1;
        break;
    ...
    case Biểu thức n:
        Lệnh n;
        break;
    default:
        Lệnh n + 1;
        break;
}
```

Câu lệnh switch

❖ *Ví dụ:* Nhập tháng, năm và in ra số ngày trong tháng, biết rằng tháng 1, 3, 5, 7, 8, 10, 12: số ngày trong tháng là 31, tháng 4, 6, 9, 11: số ngày trong tháng là 30, tháng 2: năm nhuận: số ngày là 29, năm không nhuận: số ngày là 28. Năm nhuận là năm chia hết cho 400, hay năm chia hết cho 4 và không chia hết cho 100.

```
using System;
class NumberDay
{
    static void Main(string[] args)
    {
        int month = 7, year = 2004, numDays = 0;
        switch (month)
        {
            case 1:
            case 3:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12: numDays = 31; break;
            case 4:
            case 6:
            case 9:
            case 11: numDays = 30; break;
            case 2:
                if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0)
                    numDays = 29;
                else numDays = 28;
                break;
            default:
                if (numDays > 0)
                    Console.WriteLine("So ngay trong thang la {0}", numDays);
                else Console.WriteLine("Thang khong hop le");
        }
    }
}
```

Vòng lặp for

Cú pháp:

for (BiểuThứcKhởiTạo; BiểuThứcĐiềuKiện; BiểuThứcTăng)
 Lệnh;

❖ Ví dụ:

```
static void Main(string[] args)
{
    int i;

    for (i = 0; i < 10; i++)
    {
        Console.WriteLine(i);
    }
    Console.ReadKey();
}
```

Vòng lặp while

Cú pháp:

while (BiểuThứcĐiềuKiện)
Lệnh;

Ví dụ:

```
using System;
class Program
{
    static void Main(string[] args)
    {
        int x = 12, y = 6;
        Console.WriteLine("USCLN cua {0} va {1} la : ", x, y);
        while (x!=y)
            if (x>y) x = x - y;
            else y = y - x;
        Console.WriteLine(x);
    }
}
```

Vòng lặp do...while

Cú pháp:

do

Lệnh;

while (BiểuThứcĐiềuKiện);

- ❖ *Ví dụ:* Cho biết số nguyên dương n có phải là số nguyên tố không?

```
using System;
namespace NguyenTo
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Nhập số nguyên:");
            int n = int.Parse(Console.ReadLine());
            double k = Math.Sqrt(n);
            int i = 1;
            do
                i++;
            while (i <= k && n % i != 0);
            if (i > k) Console.WriteLine(n + " là số nguyên tố");
            else Console.WriteLine("{0} không là số nguyên tố",n);
        }
    }
}
```

Vòng lặp *foreach*

- ❖ Cấu trúc foreach cho phép tạo vòng lặp duyệt qua tất cả phần tử của một danh sách (collection) hay một mảng.
- ❖ Câu lệnh foreach có cú pháp chung như sau:

foreach (KiểuPhânTử TênPhânTử in TênTậpHợp)

Lệnh;

Vòng lặp *foreach*

Ví dụ:

```
using System;
class Program
{
    static void Main(string[] args)
    {
        string[] nhom = {"Lan", "Thu", "Hoa", "Xuan"};
        foreach(string ten in nhom)
            Console.Write("{0} ", ten);
    }
}
```

Lệnh ***break***& lệnh ***continue***

- ❖ Lệnh break dùng để thoát tức thời ra khỏi vòng lặp while, do... while, for hay lệnh rẽ nhánh switch chứa nó và thực hiện lệnh tiếp theo.
- ❖ Lệnh continue để bỏ qua các lệnh sau continue và quay trở về đầu vòng lặp chứa nó.

Lệnh *break*& lệnh *continue*

❖ Ví dụ:

```
int i = 0;
while (i < 10)
{
    Console.WriteLine(i);
    i++;
    if (i == 4)
    {
        break;
    }
}
```

```
int i = 0;
while (i < 10)
{
    if (i == 4)
    {
        i++;
        continue;
    }
    Console.WriteLine(i);
    i++;
}
```

Phân loại lỗi

❑ Lỗi cú pháp (syntax error)

- Xuất hiện khi ta viết code
- Được thông báo ngay khi viết sai cú pháp
- Nguyên nhân: viết sai hoặc thiếu cú pháp

❑ Lỗi thực thi (runtime error)

- Xảy ra khi ta thực thi chương trình
- Khó xác định hơn lỗi cú pháp
- Nguyên nhân: Mở một tập tin đang tồn tại, chia cho 0, truy xuất bảng không tồn tại trong CSDL, ...

❑ Lỗi luận lý (Logic error)

- Xảy ra khi ta thực thi chương trình. Được thể hiện dưới những hình thức hoặc những kết quả không mong đợi.
- Nguyên nhân: sai lầm trong thuật giải.

Xử lý lỗi

- ❑ Sử dụng lệnh throw
- ❑ Sử dụng cú pháp try ... catch

Xử lý lỗi

❑ Sử dụng lệnh throw

- Câu lệnh **throw** thường được dùng để báo hiệu sự cố xảy ra của một tình trạng bất thường (sự ngoại lệ - exception) trong khi chương trình thực thi.
- Cú pháp:
throw new <loại Exception>([“Câu thông báo lỗi”]);
- Ví dụ:

```
string s=null;
if (s==null)
    throw new ArgumentNullException();
```

Xử lý lỗi

❑ Sử dụng cú pháp try ... catch

- Cho phép thử thực hiện một khối lệnh xem có bị lỗi hay không, nếu có sẽ bắt và xử lý lỗi
- Cú pháp:

```
try
{
    //Khối lệnh xử lý
}
catch
{
    //Khối lệnh xử lý khi khối try bị lỗi
}
[ finally
{
    //Khối lệnh xử lý cho dù khối try có bị lỗi hay không?
} ]
```

Xử lý lỗi

❑ Sử dụng cú pháp try ... catch

- Ví dụ 1:

```
int a;
try
{
    a = int.Parse(txtSo.Text);
}
catch (FormatException ex)
{
    tbLoi.Text = "Lỗi: " + ex.Message;
}
finally
{
    Ketqua.Text = "Câu lệnh này luôn được thực hiện";
}
```

Xử lý lỗi

❑ Sử dụng cú pháp try ... catch

- Lưu ý:
 - Một khối try có thể dùng một hay nhiều khối catch
 - Mỗi khối catch hiển thị một loại lỗi khác nhau. Lúc đó công cụ vận hành sẽ chỉ thực thi khối catch với kiểu exception phù hợp đầu tiên
 - Trường hợp có nhiều khối catch thì khối catch với các exception cụ thể phải được liệt kê trước các exception tổng quát.
 - Các khai báo biến trong khối lệnh try sẽ không có hiệu lực đối với các lệnh sử dụng nó nằm ở bên ngoài. Do vậy, nếu bên trong khối lệnh finally có sử dụng các biến liên quan đến các biến được khai báo và sử dụng trong khối lệnh try thì phải đưa các khai báo biến này ra ngoài khối lệnh try.

Xử lý lỗi

❑ Sử dụng cú pháp try ... catch

- Ví dụ 2:

```
try
{
    object zero = 0;
    int res=0, num=0;
    res = (num/ (int)zero);
}
catch (DivideByZeroException ex)
{
    tbLoi.Text = "Error 1: " + ex;
}
catch (Exception ex)
{
    tbLoi.Text = "Error 2: " + ex;
}
```

Các kiểu ngoại lệ

Kiểu ngoại lệ	Mô tả	Ví dụ
Exception	Lớp cha của tất cả ngoại lệ	
SystemException	Lớp cha của tất cả các ngoại lệ xây dựng sẵn	
IndexOutOfRangeException	Phát sinh khi chỉ số truy cập phần tử mảng không hợp lệ	arr[arr.Length+1]
ArrayTypeMismatchException	Phát sinh khi kiểu thành phần mảng không đúng	
NullReferenceException	Phát sinh khi truy cập thành viên của đối tượng null	object o = null; o.ToString();

Các kiểu ngoại lệ

InvalidOperationException	Phát sinh bởi phương thức khi ở trạng thái không đúng	int? a = default(int?); int b = a.Value; a kiểu Nullable không chứa bất kỳ giá trị
ArgumentException	Lớp cha của tất cả ngoại lệ tham đối	
ArgumentNullException	Phát sinh bởi phương thức không cho phép một tham đối là null	String s = null; "Calculate".IndexOf (s);
ArgumentOutOfRangeException	Phát sinh bởi phương thức kiểm tra các tham đối trong phạm vi đã cho	String s = string"; s.Chars[9];

Các kiểu ngoại lệ

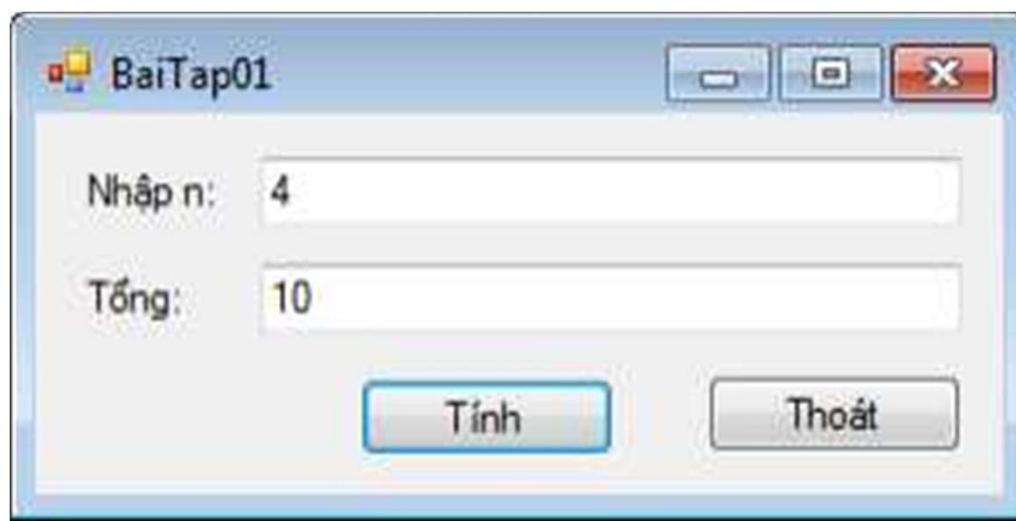
MemberAccessException	Lớp cha của tất cả ngoại lệ phát sinh khi truy cập thành viên	
MethodAccessException	Phát sinh khi truy cập đến phương thức không được truy cập	
ArithmeticException	Phát sinh khi có lỗi liên quan đến các phép toán	
DivideByZeroException	Phát sinh khi chia cho 0	
NotFiniteNumberException	Phát sinh khi số 0 là hữu hạn, không hợp lệ	

Các kiểu ngoại lệ

FormatException	Phát sinh do định dạng không đúng	
InvalidCastException	Phát sinh khi phép gán không hợp lệ	
OutOfMemoryException	Phát sinh khi đầy bộ nhớ	
StackOverflowException	Phát sinh khi tràn stack	
NotSupportedException	Phát sinh khi gọi phương thức không tồn tại trong lớp	

Bài tập

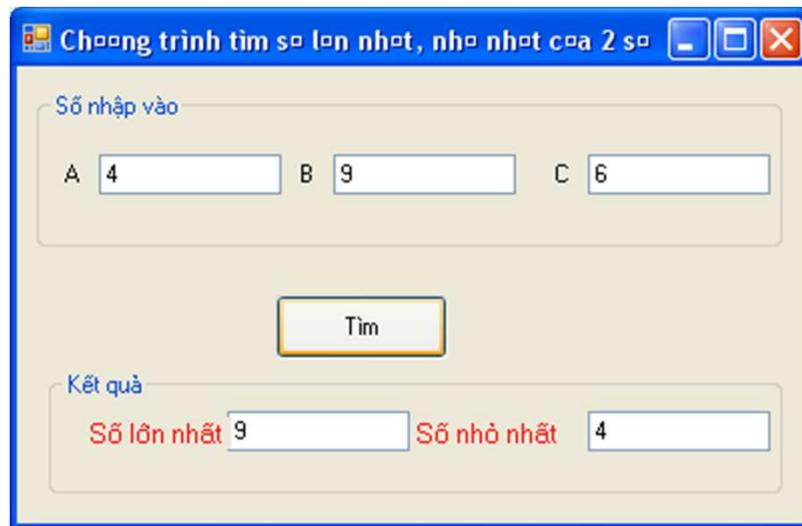
- ❖ **Bài 01:** Viết chương trình tính tổng từ 1 đến n.



Gợi ý: Sử dụng vòng lặp for hoặc while.

Bài tập

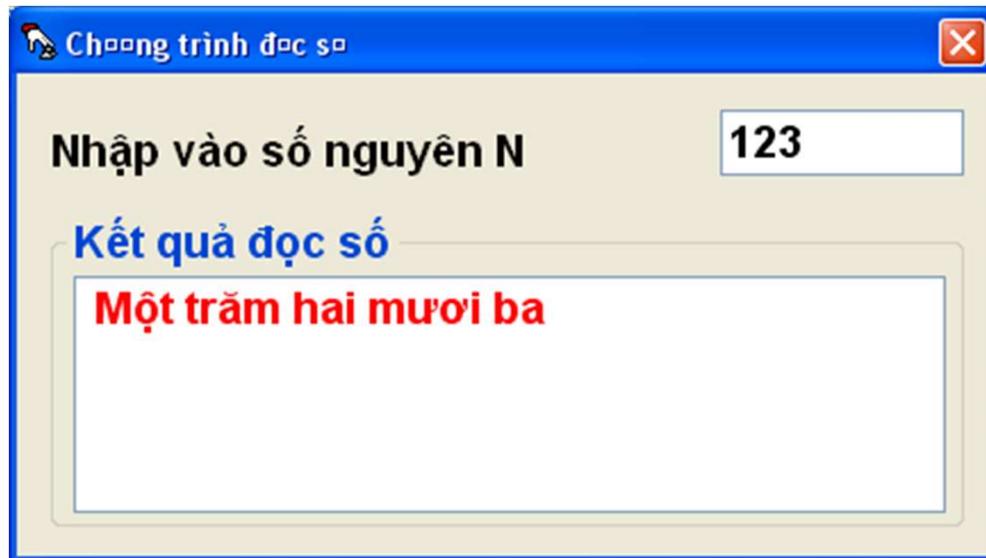
❖ **Bài 02:** Viết chương trình cho phép nhập vào 3 số a, b và c. Tìm và hiển thị số lớn nhất, nhỏ nhất như form sau:



Gợi ý: sử dụng câu lệnh if...else...

Bài tập

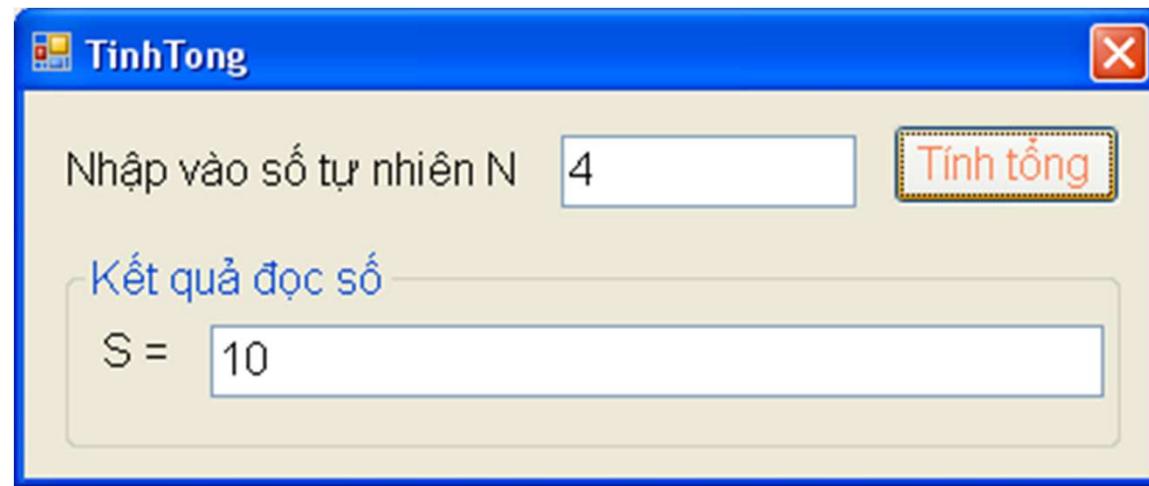
- ❖ **Bài 03:** Nhập vào một số nguyên N, hiển thị bằng chữ số vừa nhập.



Gợi ý: sử dụng cấu trúc switch ... case ...

Bài tập

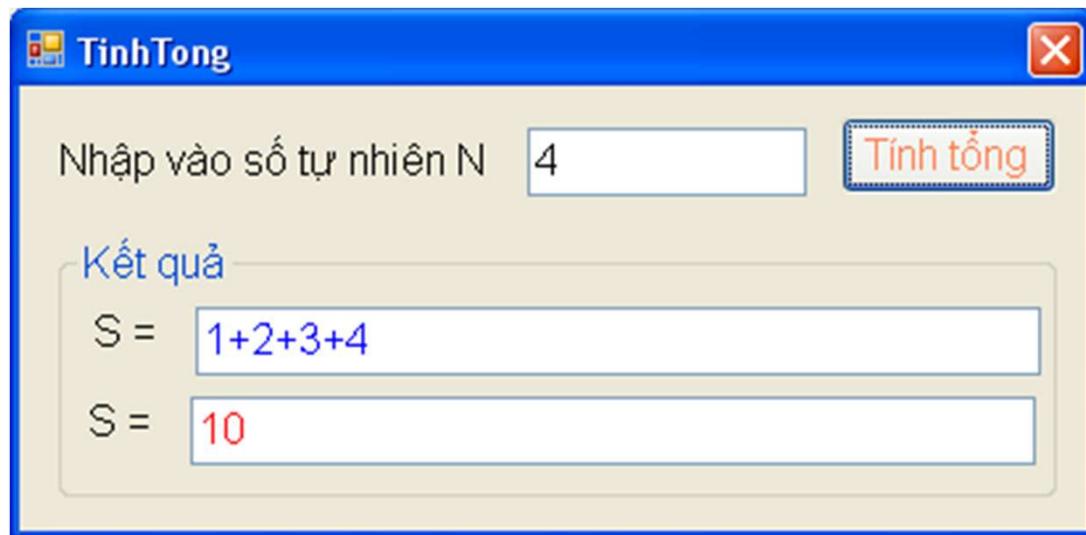
❖ **Bài 04:** Viết chương trình nhập vào giá trị nguyên dương N, tính tổng $S = 1 + 2 + 3 + \dots + N$



Gợi ý: sử dụng vòng lặp for, while.

Bài tập

❖ **Bài 05:** Viết chương trình nhập vào giá trị số nguyên dương N. Tính tổng như giao diện sau:



❖ Tương tự tính tổng $S = 1 + 3 + 5 + 7 + \dots + (2N + 1)$

Bài tập

❖ **Bài 06:** Viết chương trình nhập vào giá trị nguyên dương N, số thực X. Tính tổng như giao diện sau:



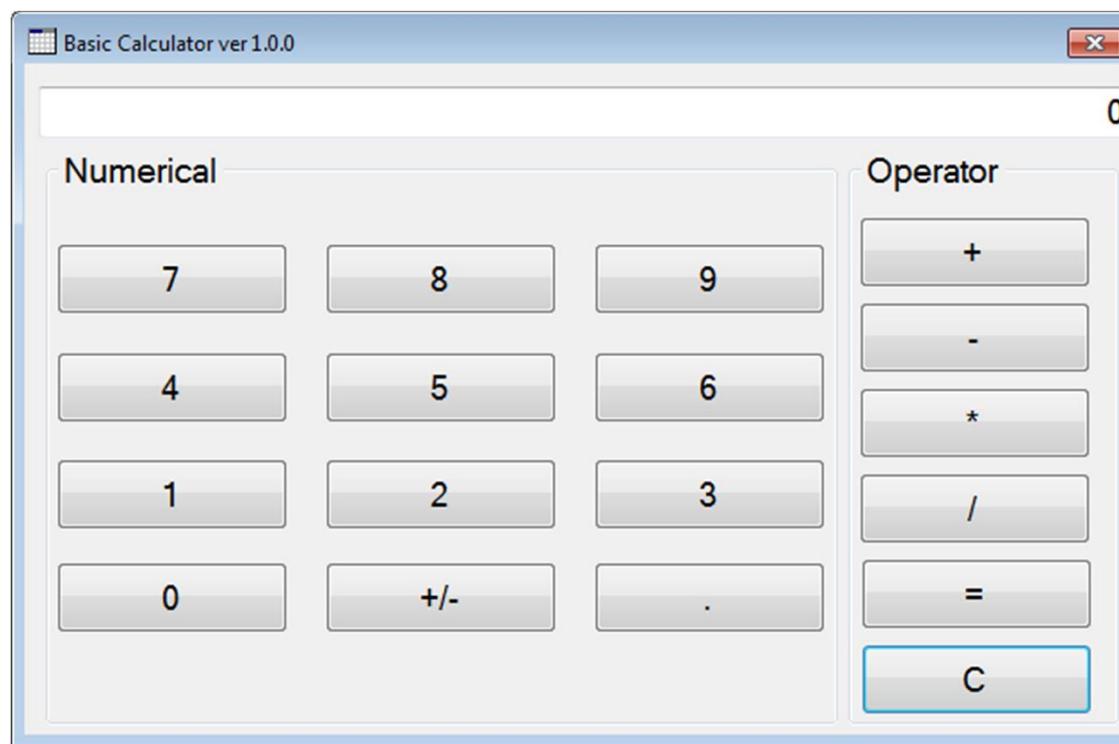
Bài tập

❖ **Bài 07:** Thiết kế giao diện và cài đặt chương trình như sau:



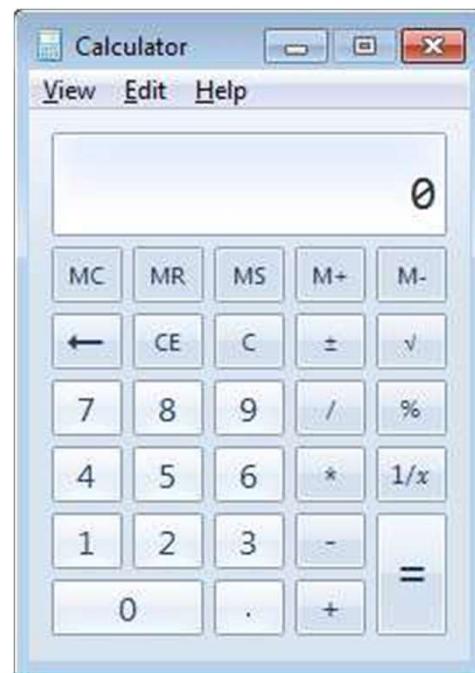
Bài tập

❖ **Bài 08:** Viết chương trình máy tính đơn giản



Bài tập

- ❖ **Bài 09:** Thiết kế máy tính bỏ túi có các chức năng như hình dưới. Không cần thiết kế menu.



Kiểu cấu trúc - struct

- Khái niệm
- Khai báo và khởi tạo

Kiểu cấu trúc - struct

□ Khái niệm

- Là kiểu do người dùng định nghĩa. Giống như các kiểu giá trị khác, các instance của các kiểu do người dùng định nghĩa được lưu trữ trên stack và chứa trực tiếp dữ liệu của chúng
- struct là một kiểu cấu trúc chứa nhiều thành phần có kiểu dữ liệu khác nhau

Kiểu cấu trúc - struct

❑ Khai báo và khởi tạo

- Khai báo:

```
<Từ khóa khai báo phạm vi sử dụng> struct <Tên_struct>
{
    // Khai báo các biến thành viên
    Kiểu_DL thành_phần_1;
    Kiểu_DL thành_phần_2;
    ....
}
```

Kiểu cấu trúc - struct

□ Khai báo và khởi tạo

- Ví dụ: Khai báo struct mô tả thông tin nhân viên gồm: mã số, họ tên, ngày sinh, hệ số lương

```
public struct NhanVien
{
    public string maSo;
    public string hoTen;
    public DateTime ngaySinh;
    public double heSoLuong;
}
```

Kiểu cấu trúc - struct

□ Khai báo và khởi tạo

• Khởi tạo struct

- Khởi tạo một thê hiện của struct bằng cách sử dụng từ khóa **new**
- Cú pháp:

<Tên_struct> tên_đối_tượng = new <Tên_struct>();

- Ví dụ: Khởi tạo đối tượng nv kiểu struct NhanVien

```
NhanVien nv= new NhanVien();
nv.maSo = "A01";
nv.hoTen = "Hoàng Thị Ngọc";
nv.ngaySinh = DateTime.Parse("1/24/1968");
nv.heSoLuong = 2.0;
```

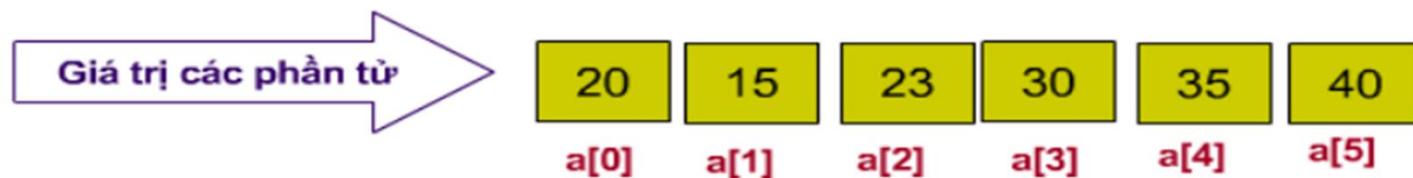
Kiểu mảng - Array

- Khái niệm
- Khai báo và khởi tạo mảng
- Các thao tác trên mảng

Kiểu mảng - Array

□ Khái niệm

- Mảng là kiểu tham chiếu
- Là một tập hợp các biến có cùng tên, cùng kiểu dữ liệu nhưng khác chỉ số
- Chỉ số (Index) là một con số dùng để xác định vị trí phần tử có trong mảng, và chỉ số luôn bắt đầu bằng **0**



- Trong đó:
 - a : là biến
 - 0, 1, 2, 3, 4, 5 : là chỉ số
 - **0** là chỉ số đầu và **5** là chỉ số cuối

Kiểu mảng - Array

□ Khai báo và khởi tạo mảng

- Khai báo không khởi tạo kích thước và giá trị:
 - Cú pháp: <Kiểu_dữ_liệu>[] Tên_mảng;
 - Ví dụ: int[] arr;
- Khai báo có khởi tạo kích thước nhưng không khởi tạo giá trị ban đầu:
 - Cú pháp:

<Kiểu_dữ_liệu>[] Tên_mảng = new <Kiểu_dữ_liệu>[<số p. tử>];

- Ví dụ: Khai báo mảng nguyên 10 phần tử
int[] arr = new int[10];

Kiểu mảng - Array

□ Khai báo và khởi tạo mảng

- Khai báo có khởi tạo kích thước và khởi tạo giá trị ban đầu:

- Cú pháp:

```
<Kiểu_dữ_liệu>[] Tên_mảng = new <Kiểu_dữ_liệu>[] {giá_trị_1,  
giá_trị_2, giá_trị_3, ... };
```

hoặc:

```
<Kiểu_dữ_liệu>[] Tên_mảng = { giá_trị_1, giá_trị_2, giá_trị_3,  
... };
```

- Ví dụ: Khai báo và khởi tạo mảng nguyên có 5 phần tử

```
int[ ] arr = new int[ ]{2, 10, 4, 8, 5}; //hoặc  
int[ ] arr = {2, 10, 4, 8, 5};
```

Kiểu mảng - Array

❑ Khai báo và khởi tạo mảng

- Sử dụng từ khóa **var**

- Ví dụ: khởi tạo mảng có kiểu ngầm định

```
var a = new[] { 1, 10, 100, 1000 };           // int[]
var b = new[] { 1, 1.5, 2, 2.5 };            // double[]
var c = new[] { "hello", null, "world" };     // string[]
```

- Kiểu dữ liệu của mảng được hiểu ngầm định dựa vào giá trị các phần tử
 - Giá trị của các phần tử phải có cùng khả năng chuyển kiểu được

Ví dụ: Các khai báo sau không thể chuyển kiểu ngầm định

```
var d = new[] { 1, "mot", 2, "hai" };    // Giá trị các p.tử không cùng kiểu
var e = new[] { 3, 2, null, 1 };          // int là kiểu trị, không được phép null
var f = new[] { 3, 2, true, 1 };          // true không thể chuyển kiểu int
```

Kiểu mảng - Array

❑ Các thao tác trên mảng

- Truy xuất giá trị của một phần tử trong mảng
- Lấy chiều dài của mảng (kích thước mảng)
- Duyệt mảng
- Tìm phần tử có giá trị lớn nhất/nhỏ nhất
- Sắp xếp mảng
- Tìm kiếm giá trị phần tử trong mảng
- Thay đổi kích thước mảng

Kiểu mảng - Array

❑ Các thao tác trên mảng

- Truy xuất giá trị của một phần tử trong mảng
 - Cú pháp: **Tên_mảng[<chỉ_số>]**
 - **<chỉ_số>**: trong khoảng từ 0 → Số phần tử -1
- Lấy chiều dài của mảng
 - Sử dụng thuộc tính **Length** của mảng
 - Ví dụ:

```
int[ ] a = {2, 10, 4, 8, 5};  
txtKetqua.Text= a.Length.ToString(); //→ kết quả là 5
```

Kiểu mảng - Array

❑ Các thao tác trên mảng

- Duyệt mảng
 - Duyệt mảng sử dụng vòng lặp **for**
 - Cú pháp:

```
for (int i = 0; i < Tên_mảng.Length; i++)
{
    // Xử lý phần tử Tên_mảng[i]
}
```

- Ví dụ: duyệt và xuất mảng

```
int[] arr = {1, 2, 3, 4, 5};
string chuoimang = "";
for (int i = 0; i < arr.Length; i++)
    chuoimang += arr[i].ToString() + " ";
txtKetqua.Text = chuoimang;      //xuất ra màn hình
```

Kiểu mảng - Array

❑ Các thao tác trên mảng

- Duyệt mảng

- Duyệt mảng sử dụng vòng lặp **foreach**
 - Cú pháp:

```
foreach (<kiểu_dữ_liệu> <phản_tử> in Tên_mảng)
{
    // Xử lý phản tử <phản_tử>
}
```

- Ví dụ:

```
int[] arr = {1, 2, 3, 4, 5};
string chuoimang = "";
foreach (int i in arr)
    chuoimang += i.ToString() + " ";
txtKetqua.Text = chuoimang;      //xuất ra màn hình
```

Kiểu mảng - Array

❑ Các thao tác trên mảng

- Tìm phần tử có giá trị lớn nhất/nhỏ nhất

```
//Tìm phần tử lớn nhất
```

```
int i, max;
max = arr[0]; // phần tử đầu tiên
for (i = 0; i < arr.Length; i++)
{
    // nếu gặp phần tử lớn hơn
    if (max < arr[i])
        max = arr[i];
}
```

```
//Tìm phần tử nhỏ nhất
```

```
int i, min;
min = arr[0]; // phần tử đầu tiên
for (i = 0; i < arr.Length; i++)
{
    // nếu gặp phần tử nhỏ hơn
    if (min > arr[i])
        min = arr[i];
}
```

Kiểu mảng - Array

❑ Các thao tác trên mảng

- Sắp xếp

- Sắp tăng dần

- Cú pháp: **Array.Sort(<Mảng>);**

- Array là lớp được xây dựng sẵn thuộc namespace System

- Ví dụ:

- // Sắp xếp mảng nguyên arr tăng dần*

- Array.Sort(arr);

- Đảo ngược:

- Cú pháp: **Array.Reverse(<Mảng>);**

- Ví dụ:

- // Sắp xếp mảng nguyên arr giảm dần*

- Array.Sort(arr);

- Array.Reverse(arr);

Kiểu mảng - Array

❑ Các thao tác trên mảng

- Tìm kiếm:

- Tìm một giá trị trong mảng, trả về chỉ số của phần tử nếu tìm thấy, ngược lại trả về -1

- Cú pháp:

Array.IndexOf (<tên_mảng>, <giá_trị_tìm>) //tìm từ phần tử đầu
Array.LastIndexOf(<tên_mảng>, <giá_trị_tìm>) //tìm từ phần tử cuối

- Ví dụ:

```
int giaTri = 5;    //Tìm vị trí của phần tử có giá trị là 5
int viTri = Array.IndexOf(arr, giaTri);
if (viTri < 0)
    txtKetqua.Text="Không tìm thấy!";
else
    txtKetqua.Text = "Đã tìm thấy " + giaTri + " tại vị trí "+ viTri;
```

Kiểu mảng - Array

❑ Các thao tác trên mảng

- Thay đổi kích thước mảng:

- Sử dụng phương thức Resize của lớp Array để thay đổi kích thước của mảng

- Cú pháp:

Array.Resize <Kiểu_dữ_liệu>(ref <tên_mảng>, <Số_p.tử_mới>)

- Ví dụ 1:

```
int[ ] arr = {4, 2, 5}; //ban đầu mảng a có 3 phần tử
```

```
Array.Resize <int> (ref arr, 4); //tăng thêm 1 phần tử trong mảng
```

Kiểu mảng - Array

❑ Các thao tác trên mảng

- Thay đổi kích thước mảng:

- Ví dụ 2: Thêm 1 phần tử có giá trị là 9 vào cuối mảng arr

```
int[ ] arr = {4, 2, 5};           // ban đầu mảng arr có 3 phần tử
```

```
Array.Resize <int> (ref arr, 4); // tăng kích thước của mảng arr
```

```
a[3] = 9;
```

Trước khi thêm



Sau khi thêm



Kiểu mảng - Array

❑ Các thao tác trên mảng

- Thay đổi kích thước mảng:

- Ví dụ 3: Thêm 1 phần tử có giá trị là 25 vào vị trí thứ 2 trong mảng arr

```
Array.Resize <int> (ref arr, 5);           //tăng kích thước của mảng arr
```

```
for(int i = arr.Length - 1; i > 2; i--)    //dịch chuyển phần tử
```

```
    arr[i] = arr[i - 1];
```

```
arr[2] = 25;                                //gán giá trị cho phần tử có chỉ số thứ 2
```

Trước khi thêm



Sau khi thêm



Kiểu mảng - Array

❑ Các thao tác trên mảng

- Thay đổi kích thước mảng:

- Ví dụ 4: Xóa 1 phần tử ở vị trí cuối mảng arr

//mảng arr hiện tại có 5 phần tử

Array.Resize <int> (ref arr, 4); //giảm kích thước của mảng arr

Trước khi xóa



Sau khi xóa



Kiểu mảng - Array

❑ Các thao tác trên mảng

- Thay đổi kích thước mảng:

- Ví dụ 5: Xóa 1 phần tử đứng vị trí thứ 2 trong mảng arr

```
//mảng arr hiện tại có 5 phần tử
```

```
for(int i = 3; i < arr.Length; i++)      //dịch chuyển phần tử  
    arr[i - 1] = arr[i];
```

```
//giảm kích thước của mảng arr xuống một phần tử
```

```
Array.Resize <int> (ref arr, 4);
```

Trước khi xóa



Sau khi xóa



Collections và Generic Collections

Cho mảng sau:

```
string[] arr = new string[]{"Xin", "chào", "các", "bạn"};
```

/Vấn đề thứ nhất

Thêm mới một phần tử có kiểu số nguyên được không?

/Vấn đề thứ hai

```
Array.Resize <string> (ref arr, 5);  
for(int i = arr.Length - 1; i > 2; i--)  
    arr[i] = arr[i-1];  
arr[2] = "tất cả";
```

Khi thêm mới một phần tử bất kỳ trong mảng thì đoạn code dài. → có cách nào khác ???

Collections

- ❑ Giới thiệu
- ❑ Giới thiệu một số Collections thông dụng
- ❑ Các thao tác trên:
 - ArrayList
 - HashTable
 - SortedList

Collections

□ Giới thiệu

- Collections là các lớp hỗ trợ thu thập và quản lý các đối tượng
 - Một cách có thứ tự
 - Hỗ trợ lưu, tìm kiếm và duyệt các đối tượng trong tập hợp
- Namespace **System.Collections** của .NET Framework cung cấp nhiều kiểu tập hợp khác nhau

Collections

□ Giới thiệu một số Collections thông dụng

- **ArrayList:** là mảng động, cho phép các phần tử có giá trị null và trùng nhau
 - Giá trị của các phần tử được phép lưu trên nhiều kiểu dữ liệu khác nhau
 - Các phần tử trong ArrayList được quản lý thông qua chỉ số (index) và giá trị (value)
- **Hashtable:** tập hợp các phần tử, gồm một cặp giá trị và khóa → [key, value]
 - Giá trị của khóa (Key) không được trùng, không được null
- **SortedList:** giống HashTable nhưng được sắp xếp theo khóa (Key)
 - Là sự kết hợp giữa ArrayList và Hashtable
 - Tự động sắp xếp tăng dần theo giá trị khóa (key)

Collections

❑ Các thao tác trên ArrayList

- **Thêm phần tử:** sử dụng các phương thức sau:

- **Add:** Thêm 1 phần tử vào cuối danh sách
- **AddRange:** Thêm 1 mảng phần tử vào cuối danh sách
- **Insert:** Chèn 1 phần tử vào 1 vị trí bất kỳ
- **InsertRange:** Chèn 1 mảng phần tử vào 1 vị trí bất kỳ
- **Ví dụ 1:** dùng phương thức Add

```
ArrayList taphop=new ArrayList();
string s="Xin chào";
taphop.Add(s);           //chỉ số =0; giá trị = "Xin chào"
taphop.Add("Chào");     //chỉ số =1; giá trị = "Chào"
taphop.Add(50);          //chỉ số =0; giá trị = 50
taphop.Add(new object()); //chỉ số =0; giá trị = System.Object
```

Collections

❑ Các thao tác trên ArrayList

- **Thêm phần tử:** sử dụng các phương thức sau:
 - Ví dụ 2: dùng phương thức AddRange

```
ArrayList taphop = new ArrayList();
string[] mang = new string[]{"Mot","Hai","Ba"};
taphop.AddRange(mang);
object[] mang_1=new object[]{new object(),new ArrayList()};
taphop.AddRange(mang_1);
```

Collections

❑ Các thao tác trên ArrayList

- **Thêm phần tử:** sử dụng các phương thức sau:

- Ví dụ 3: dùng phương thức Insert và InsertRange

```
ArrayList taphop = new ArrayList();
taphop.Insert(0, "Kỹ thuật");           //Chèn vào vị trí đầu tiên
string[] arr = new string[] { "Lập trình", "C#" };
//Chèn mảng arr vào vị trí thứ 2
taphop.InsertRange(1, mang);
```

Collections

❑ Các thao tác trên ArrayList

- **Xóa phần tử:** sử dụng các phương thức sau:

- Remove: xóa 1 phần tử có xác định giá trị cụ thể
- RemoveAt: xóa 1 phần tử tại vị trí xác định
- RemoveRange: xóa các phần tử từ vị trí xác định
- Clear: xóa tất cả các phần tử
- Ví dụ:

Ví dụ:

```
taphop.Remove("Xin chào");
```

```
taphop.RemoveAt(1);           // xóa phần tử ở vị trí thứ 2
```

```
taphop.RemoveRange(0,4);      // xóa 4 phần tử bắt đầu từ chỉ số 0
```

Collections

❑ Các thao tác trên ArrayList

- Truy xuất phần tử:

Tên_ArrayList[chi_số]

- Duyệt ArrayList

- ArrayList hỗ trợ 3 cách duyệt các phần tử trong collection:

- Dùng chỉ số của phần tử (index)
 - Dùng phương thức **GetEnumerator**, trả về kiểu **IEnumerator**
 - Dùng cấu trúc foreach

Collections

❑ Các thao tác trên ArrayList

• Duyệt ArrayList

▪ Ví dụ:

Chỉ số

```
for (int i=0;i<taphop.Count;i++){  
    str.AppendLine(taphop[i].ToString());  
}
```

GetEnumerator

```
IEnumerator e=taphop.GetEnumerator();  
while(e.MoveNext())  
    str.AppendLine(e.Current.ToString());
```

foreach

```
foreach (object item in taphop){  
    str.AppendLine(item.ToString());  
}
```

Collections

❑ Các thao tác trên ArrayList

- **Sắp xếp phần tử:**

- Dùng phương thức Sort()
 - Sẽ bị lỗi nếu các phần tử có các kiểu dữ liệu khác nhau
- Ví dụ:

```
taphop.Sort();      //sắp xếp phân biệt chữ HOA/thường  
//sắp xếp không phân biệt chữ HOA/thường  
taphop.Sort(new CaseInsensitiveComparer());
```

Collections

❑ Các thao tác trên ArrayList

- **Tìm kiếm phần tử:** dùng các phương thức
 - **IndexOf:** Tìm từ phần tử đầu tiên thỏa điều kiện. Nếu tìm thấy thì trả về chỉ số của phần tử được tìm thấy, ngược lại trả về -1
 - **LastIndexOf:** Giống IndexOf nhưng tìm từ phần tử cuối
 - **Contains:** Dùng để kiểm tra trong tập hợp có chứa giá trị cần tìm hay không. Nếu có thì trả về true, ngược lại trả về false.
 - **Ví dụ:**

```
ArrayList taphop = new ArrayList[]{"Mai", "Lan", "Cúc", "Trúc"};
if (taphop.Contains("Lys"))           // kiểm tra tập hợp có chứa chuỗi "Lys" không
{
    int vitri=taphop.IndexOf(chuoi);      //vị trí chuỗi trong tập hợp
    taphop.RemoveAt(vitri);             }
else    taphop.Clear();                //xóa tất cả các phần tử
```

Collections

❑ Các thao tác trên Hashtable

- Là tập hợp dùng để lưu trữ các phần tử, mỗi phần tử gồm một cặp thông tin: khóa (key) và giá trị (value) → được phân loại là Dictionary
- Key của các phần tử phải duy nhất
- Chỉ cho phép tìm kiếm phần tử theo key
- **Thêm phần tử:**
 - Sử dụng phương thức **Add()** để thêm 1 phần tử có khóa **key**, giá trị **value** vào cuối danh sách
 - Ví dụ: thêm 3 loại trái cây sau vào hashtable

```
Hashtable danhsach = new Hashtable();
danhsach.Add("apple","Trái táo");
danhsach.Add("grape","Trái nho");
danhsach["orange"]="Trái cam";
```

Collections

❑ Các thao tác trên Hashtable

- Xóa phần tử:

- Sử dụng phương thức Remove() và Clear()
- Ví dụ: danhsach.Remove("orange");

- Truy xuất phần tử:

- Cú pháp: **Tên_hashtable[Key];**
- Ví dụ:

```
Hashtable danhsach = new Hashtable();
danhsach.Add("apple", "Trái táo");
danhsach.Add("grape", "Trái nho");
Danhsach.Add("orange", "Trái cam");
MessageBox.Show(danhsach["orange"].ToString()); // Trái cam
```

Collections

❑ Các thao tác trên Hashtable

- Duyệt Hashtable:

- Dùng lớp DictionaryEntry, có thuộc tính:
 - Key: khóa
 - Value: giá trị
- Duyệt trên tập hợp Keys hoặc Values (kiểu ICollection)
- Dùng giao tiếp IDictionaryEnumerator
- Ví dụ:

```
StringBuilder str = new StringBuilder();
foreach (DictionaryEntry entry in danh sach)
    str.AppendLine(entry.Key + "/" + entry.Value);
//Duyệt tập hợp keys hoặc tập hợp Values
foreach (object ob in danh sach.Keys)      //hoặc danh sach.Values
    str.AppendLine("Key là: " + ob.ToString());
```

Collections

❑ Các thao tác trên Hashtable

- **Tìm kiếm:** sử dụng các phương thức
 - **ContainsKey :** Trả về True/False, kiểm tra giá trị key có trong HashTable không
 - **ContainsValue:** Trả về True/False, kiểm tra giá trị value có trong HashTable không
 - **Ví dụ:**

```
//Kiểm tra danh sách có key là "banana"  
if (!danh sach.ContainsKey("banana")) //nếu chưa có  
    danh sach.Add("banana", "Trái chuối");
```

Collections

❑ Các thao tác trên SortedList

- Giống HashTable nhưng Key được sắp xếp và cho phép truy xuất phần tử theo chỉ số và theo key
- Tự động sắp xếp tăng dần theo giá trị khóa (key)
- **Thêm phần tử:**
 - Sử dụng phương thức **Add()** để thêm 1 phần tử có khóa **key**, giá trị **value** vào cuối danh sách
- **Xóa phần tử:**
 - Sử dụng phương thức **Remove**, **RemoveAt** và **Clear**
- **Truy xuất phần tử:** Sử dụng cú pháp giống Hashtable, hoặc dùng:
 - Phương thức **GetKey(chỉ số)**: truy xuất key của phần tử có **chỉ số**
 - Phương thức **GetByIndex(chỉ số)**: truy xuất value của phần tử có **chỉ số**

Collections

❑ Các thao tác trên SortedList

- Ví dụ:

```
SortedList srt = new SortedList();
srt["b"] = "Hoa Hồng"; srt["a"] = "Hoa Lys"; srt["d"] = "Hoa Lan";
if (!srt.ContainsKey("c"))
    srt.Add("c", "Hoa Đào");
StringBuilder str = new StringBuilder();
foreach (DictionaryEntry di in srt)           //đuyệt dùng foreach
    str.AppendLine(string.Format("Key: {0} ; Value: {1}", di.Key, di.Value));
IDictionaryEnumerator ic = srt.Getenumerator();  //đuyệt dùng p.thúc Getenumerator
while (ic.MoveNext())
{
    str.AppendLine(string.Format("Key: {0} ; Value: {1} ", ic.Key, ic.Value));
}
//Điểm khác biệt với HashTable: cho phép truy xuất theo chỉ số
for (int i = 0; i < srt.Count; i++)
    str.AppendLine(String.Format("Key: {0} ; Value: {1} ;", srt.GetKey(i), srt.GetByIndex(i)));
```

Generic Collections

❑ Generic là gì?

- Generic là 1 phần trong hệ thống kiểu của .NET Framework, cho phép định kiểu mà không quan tâm nhiều đến các chi tiết bên trong
- .NET Framework cung cấp nhiều generic class trong namespace **System.Collections.Generic**, các lớp này:
 - Hoạt động như những lớp khác
 - Tăng hiệu năng thực hiện và truy xuất an toàn kiểu (type-safety)
- Ngoài những Generic class do .Net cung cấp, người lập trình có thể tự tạo ra các Generic class tùy biến

Generic Collections

□ Giới thiệu các Generic Collections class

- Generic Collections là một tập hợp định kiểu mạnh. Có các chức năng tương tự như các tập hợp đã giới thiệu
- Điểm khác biệt so với các tập hợp đã giới thiệu:
 - Phải khai báo kiểu dữ liệu của các phần tử trong tập hợp khi khởi tạo
 - Tất cả các phần tử trong tập hợp đều có cùng kiểu với kiểu được khai báo lúc đầu
- Tăng hiệu năng thực thi và định kiểu một cách an toàn
- .NET framework hỗ trợ tính generic cho hầu hết các tập hợp đã được giới thiệu trước đó. Bảng sau trình bày các Collection tương đương:

Collections	Generic Collections
ArrayList	List<>
Hashtable	Dictionary<>
SortedList	SortedList<>
DictionaryEntry	KeyValuePair<>

Generic Collections

❑ Giới thiệu các Generic Collections class

- **List<> class**

- Tương tự như ArrayList nhưng các phần tử phải được xác định kiểu dữ liệu trước
- Khi tạo List, ta có thể:
 - Thêm phần tử vào danh sách
 - Truy xuất phần tử qua chỉ số
 - Dùng cấu trúc foreach duyệt danh sách

Generic Collections

❑ Giới thiệu các Generic Collections class

- List<> class

- Ví dụ:

```
List<int> dSach = new List<int>();  
dSach.Add(10);  
dSach.Add(20);  
dSach.Add(30);  
dSach.Add("abc");           //bị lỗi dòng này khi biên dịch  
int number = dSach[0];       // 10  
StringBuilder str = new StringBuilder();  
foreach (int i in dSach)  
    str.AppendLine(i.ToString());
```

Generic Collections

□ Giới thiệu các Generic Collections class

• Dictionary<> class

- Tương tự như HashTable nhưng key và value của các phần tử phải được xác định kiểu dữ liệu trước
- Dictionary<> lưu 1 cặp key/ value trong tập hợp. Khai báo 2 tham số generic khi khởi tạo
- Ví dụ:

```
Dictionary<string, int> dic = new Dictionary<string, int>();  
dic["Three"] = 3;  
dic["One"] = 1;  
dic.Add("Two", 2);  
  
StringBuilder str = new StringBuilder();  
foreach (KeyValuePair<string, int> i in dic)  
    str.AppendLine(i.ToString()); // [Three, 3] [One, 1] [Two, 2]
```

Kết quả
trên màn
hình

[Three, 3]
[One, 1]
[Two, 2]

Generic Collections

□ Giới thiệu các Generic Collections class

• **SortedList<> class**

- Tương tự như Dictionary<> nhưng các phần tử được sắp xếp theo thứ tự của key
- **Duyệt các phần tử:** giống Dictionary<>
- Ví dụ:

```
SortedList<string, int> dic = new SortedList<string, int>();  
dic["Three"] = 3;  
dic["One"] = 1;  
dic.Add("Two", 2);  
  
StringBuilder str = new StringBuilder();  
foreach (KeyValuePair<string, int> i in dic)  
    str.AppendLine(i.ToString());    // [Three, 3] [One, 1][Two, 2]
```

Kết quả
trên màn
hình

[One, 1]
[Three, 3]
[Two, 2]

Khởi tạo nhanh Collections

- ❑ Từ C#3.0 trở lên, .Net cung cấp một phương pháp khởi tạo nhanh một tập hợp mà không dùng phương thức Add/AddRange như phiên bản trước
- ❑ Ví dụ 1:

```
//Thay vì phải viết code như sau:  
List<string> dshoa = new List<string>();  
dshoa.Add("Hong");  
dshoa.Add("Cuc");  
dshoa.Add("Lys");  
  
//Ta có thể viết lại bằng cú pháp sau :  
List<string> dshoa2 = new List<string> {  
    "Hong", "Cuc", "Lys", "Mai", "Dao"  
};
```

Khởi tạo nhanh Collections

- ❑ Ví dụ 2: khởi tạo Dictionary<>

```
//Với danh sách hoa và đơn giá  
Dictionary<string, int> dsHoa = new Dictionary<string, int>() {  
    {"Hoa Lan",20000}, {"Hoa Lys",100000}, {"Hoa Hồng",50000}  
};
```

- ❑ Ví dụ 3: Khởi tạo List<> với kiểu class

```
//Tạo class Hoa  
class Hoa  
{  
    public string Mau { get; set; }  
    public string Ten { get; set; }  
    public double DonGia { get; set; }  
}
```

Khởi tạo nhanh Collections

- ❑ Ví dụ 3: Khởi tạo List<> với kiểu class

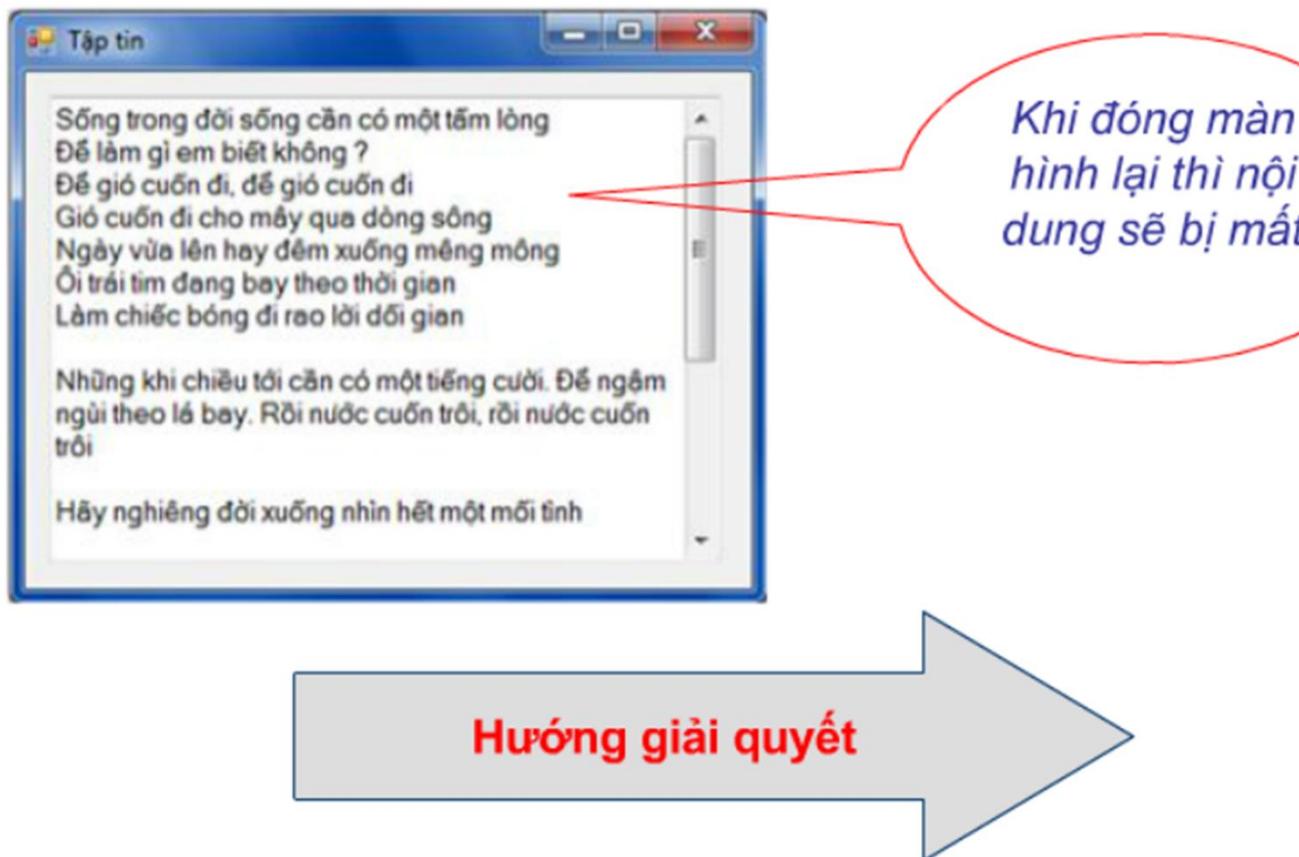
```
//Tạo danh sách hoa
```

```
List<Hoa> dshoa = new List<Hoa>();  
dshoa.Add(new Hoa {Mau="Trắng", Ten="Hoa hồng", DonGia=20000 });  
dshoa.Add(new Hoa { Mau = "Trắng", Ten = "Hoa lam", DonGia = 10000 });  
dshoa.Add(new Hoa { Mau = "Trắng", Ten = "Hoa hồng", DonGia = 20000 });
```

```
//Viết lại
```

```
List<Hoa> dshoa = new List<Hoa>() {  
    new Hoa {Mau="Trắng", Ten="Hoa hồng", DonGia=20000 },  
    new Hoa { Mau = "Trắng", Ten = "Hoa lam", DonGia = 10000 },  
    new Hoa { Mau = "Trắng", Ten = "Hoa hồng", DonGia = 20000 }  
};
```

Tập tin



Tập tin

- ❑ Đọc và ghi tập tin sử dụng File class
- ❑ Đọc tập tin sử dụng class: FileStream và StreamWriter
- ❑ Ghi tập tin sử dụng class: FileStream và StreamWriter
- ❑ Xử lý tập tin và thư mục qua các lớp: FileInfo và DirectoryInfo

Đọc và ghi tập tin sử dụng File class

- Giới thiệu File class**
- Đọc tập tin**
- Ghi tập tin**

Đọc và ghi tập tin sử dụng File class

□ Giới thiệu File class

- Cung cấp các chức năng cơ bản trong việc đọc và ghi nội dung vào tập tin
- Chứa các phương thức tĩnh, do đó không cần khởi tạo khi sử dụng nó

□ Đọc tập tin

- Dùng 2 phương thức cơ bản:
 - **ReadAllText**(“Đường_dẫn_tập_tin”): trả về chuỗi chứa nội dung của tập tin
 - **ReadAllLines**(“Đường_dẫn_tập_tin”): trả về mảng kiểu chuỗi, chứa các dòng văn bản trong tập tin

Đọc và ghi tập tin sử dụng File class

❑ Đọc tập tin

- Ví dụ 1: Đọc bằng ReadAllText

```
 OpenFileDialog myDialog = new OpenFileDialog();
myDialog.Filter = "Text Files (*.txt)|*.txt" + "|All files (*.*)|*.*";
myDialog.CheckFileExists = true;
myDialog.Title = "Xin chon mot File";
if (myDialog.ShowDialog() == DialogResult.OK)
{
    string Tenfile = myDialog.FileName;
    //Đọc nội dung tập tin và hiển thị lên textBox
    txtNoidung.Text = File.ReadAllText(Tenfile);
}
```

Đọc và ghi tập tin sử dụng File class

□ Đọc tập tin

- Ví dụ 2: Đọc bằng ReadAllLines (thay code trong khối if như sau)

```
string[] arrChuoi = File.ReadAllLines(Tenfile);
string kq = "";
foreach(string s in arrChuoi) //duyệt mảng
{
    kq += s + "\r\n";
}
txtNoidung.Text = kq;
```

Đọc và ghi tập tin sử dụng File class

❑ Ghi tập tin

- Dùng phương thức:
 - **WriteAllText("Đường_dẫn_tập_tin", "Nội_dung_tập_tin")**

```
//Sử dụng điều khiển hộp thoại SaveFileDialog, tương tự OpenFileDialog nhưng  
//dùng để lưu tập tin  
  
SaveFileDialog myDialog = new SaveFileDialog();  
  
myDialog.Filter = "Text Files (*.txt)|*.txt" + "|All files (*.*)|*.*";  
  
myDialog.Title = "Xin chọn một thư mục";  
  
if (myDialog.ShowDialog() == DialogResult.OK)  
{  
    string Tenfile = myDialog.FileName;  
  
    File.WriteAllText(Tenfile, txtNoidung.Text);  
}
```

Đọc tập tin sử dụng class: FileStream và StreamWriter

❑ Giới thiệu FileStream class

- FileStream cung cấp các chức năng cơ bản trong việc đọc và ghi nội dung vào tập tin từ một Stream (biểu diễn mảng byte)
- Kế thừa từ lớp Stream (là lớp trừu tượng)
- FileStream thường được sử dụng cùng với các class: File, StreamReader và StreamWriter để đọc, ghi tập tin

❑ Đọc tập tin với StreamReader class

- StreamReader cung cấp các chức năng cơ bản để đọc dữ liệu nhận được từ lớp Stream
- Sử dụng 2 phương thức chính:
 - **ReadLine()**: trả về 1 dòng văn bản của stream hiện hành
 - **ReadToEnd()**: đọc tất cả nội dung từ vị trí hiện hành đến cuối stream

Đọc tập tin sử dụng class: FileStream và StreamWriter

□ Đọc tập tin với StreamReader class

- Ví dụ:

```
 OpenFileDialog myDialog = new OpenFileDialog();
myDialog.Filter = "Text Files (*.txt)|*.txt" + "|All files (*.*)|*.*";
myDialog.CheckFileExists = true;
myDialog.Title = "Xin chon mot File";
if (myDialog.ShowDialog() == DialogResult.OK)
{
    string Tenfile = myDialog.FileName;
    //Mở tập tin để đọc
    FileStream taptin = File.Open(Tenfile, FileMode.Open, FileAccess.Read);
    StreamReader srd = new StreamReader(taptin);      //Đọc dữ liệu từ Stream
    txtNoidung.Text = srd.ReadToEnd();      //Đọc dữ liệu từ StreamReader
    srd.Close();
}
```

FileMode:
chế độ mở file

FileAccess: chế
độ đọc file

Ghi tập tin sử dụng: FileStream và StreamWriter

❑ Ghi tập tin với StreamWriter class

- StreamWriter cung cấp các chức năng cơ bản để ghi dữ liệu vào Stream.
- Sử dụng phương thức:
 - **Write(Nội_dung_được_ghi)**
- Ví dụ:

```
//đoạn code giống ví dụ WriteAllText
.....
string Tenfile = myDialog.FileName;
FileStream taptin = File.Open(Tenfile, FileMode.OpenOrCreate, FileAccess.Write);
StreamWriter srw = new StreamWriter(taptin);
srw.WriteLine(txtNoidung.Text);
srw.Close();
```

Xử lý tập tin và thư mục

□ Giới thiệu FileInfo class

- FileInfo cung cấp các chức năng cơ bản để truy xuất và thao tác với một tập tin trong hệ thống tập tin
- Các thuộc tính cơ bản giúp truy xuất các thông tin của tập tin
 - **Directory:** Trả về thư mục chứa tập tin hiện hành đang truy xuất
 - **DirectoryName:** Trả về tên của thư mục chứa tập tin đang truy xuất
 - **IsReadOnly:** Lấy và thiết lập thông tin chỉ đọc cho tập tin
 - **Length:** trả về Kích thước của tập tin
 - **Name, FullName:** Trả về tên, tên và đường dẫn đầy đủ của tập tin
- Các phương thức cơ bản giúp thao tác tập tin
 - **Exist:** kiểm tra tập tin có tồn tại hay không
 - **MoveTo:** Di chuyển tập tin đến thư mục khác
 - **CopyTo:** Sao chép tập tin đến thư mục khác
 - **Delete:** Xóa tập tin

Xử lý tập tin và thư mục

□ Giới thiệu FileInfo class

- Ví dụ: Kiểm tra tập tin có tồn tại hay không

```
FileInfo taptin=new FileInfo(@"D:\Baihat.txt");
if (taptin.Exists){
    lblKetqua.Text = "Tên tập tin: " + taptin.Name + "\n" +
                    "Đường dẫn: " + taptin.FullName ;
    // Sao chép tập tin sang ổ đĩa E
    taptin.CopyTo(@"E:\Baihat.txt"); //Lưu ý: nếu tập tin có rồi thì bị lỗi
    //xóa tập tin mới copy
    FileInfo tt = new FileInfo(@"E:\Baihat.txt");
    tt.Delete();
}
```

Xử lý tập tin và thư mục

□ Giới thiệu DirectoryInfo class

- DirectoryInfo cung cấp các chức năng cơ bản để truy xuất và thao tác với một thư mục trong hệ thống
- Các thuộc tính cơ bản giúp truy xuất các thông tin của thư mục
 - Parent: Trả về thư mục cha
 - Root: Trả về thư mục gốc
- Các phương thức cơ bản giúp thao tác thư mục
 - GetFiles: trả về mảng các đối tượng kiểu FileInfo
 - GetDirectories: trả về mảng đối tượng kiểu DirectoryInfo (các thực mục con)
 - MoveTo: Di chuyển thư mục hiện hành sang thư mục khác

Xử lý tập tin và thư mục

❑ Giới thiệu DirectoryInfo class

- Ví dụ 1: Duyệt qua các tập tin có trong thư mục Windows

```
DirectoryInfo thumuc=new DirectoryInfo(@"c:\windows");
lblKetqua.Text ="Directory: " + thumuc.FullName;
StringBuilder str = new StringBuilder();
foreach (FileInfo taptin in thumuc.GetFiles() )
    str.AppendLine(taptin.Name );
//Xuất thông tin các tập tin trong thư mục c:\Windows ra TextBox
txtDsFile.Text = str.ToString(); //txtDsFile là điều khiển TextBox
```

Xử lý tập tin và thư mục

□ Giới thiệu DirectoryInfo class

- Ví dụ 2: Duyệt qua các thư mục con có trong thư mục Windows

```
DirectoryInfo tm = new DirectoryInfo(@"c:\windows");
StringBuilder str = new StringBuilder("**** Thư mục: " + tm.FullName);
//Duyệt để xem thư mục con
foreach (DirectoryInfo dir in tm.GetDirectories())
{
    str.AppendLine(String.Format("Thư mục con: {0}", dir.Name));
}
txtNoidung.Text = str.ToString();
```