

BÀI 2. DỮ LIỆU CÓ CẤU TRÚC, PHƯƠNG THỨC VÀ XỬ LÝ NGOẠI LỆ TRONG C#.

1. Dữ liệu có cấu trúc

1.1. Mảng (Array)

Mảng là một cấu trúc dữ liệu cấu tạo bởi một số biến được gọi là những phần tử mảng. Tất cả các phần tử này đều thuộc một kiểu dữ liệu. Người lập trình có thể truy xuất phần tử thông qua chỉ số (index). Chỉ số bắt đầu bằng 0. Người ta thường chia mảng thành 2 loại: Mảng một chiều và mảng nhiều chiều. Đối với mảng nhiều chiều giáo trình này chỉ trình bày mảng hai chiều bởi vì mảng hai chiều là đặc trưng tiêu biểu cho mảng nhiều chiều

1.1.1. Mảng một chiều

Cú pháp khai báo

+ Khai báo không khởi tạo kích thước và giá trị:

<Kiểu dữ liệu>[] Tên_mảng;

+ Khai báo có khởi tạo kích thước nhưng không khởi tạo giá trị ban đầu:

<Kiểu dữ liệu>[] Tên_mảng = new <Kiểu dữ liệu>[<số phần tử>];

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

<Kiểu dữ liệu>[] Tên_mảng = new <Kiểu dữ liệu>[<Số phần tử>] {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 2.1:

```
int[ ] a;  
int[ ] a = new int[10];  
int[ ] b = new int[5]{2,10,4,8,5};  
int[ ] b = {2, 10, 4, 8, 5};
```

Cách sử dụng

Để làm việc với mảng, người lập trình thường can thiệp trực tiếp vào từng phần tử của mảng thông qua chỉ số index với cú pháp Tên_mảng[chỉ số].

Ví dụ 2.2:

```
x = a[0];  
st = b[i];  
b[1] = x;
```

Để lấy kích thước của array, chúng ta sử dụng thuộc tính Length của nó. Một số phương thức thường dùng đối với dữ liệu kiểu array là:

- **Array.Sort(arr):** hàm tĩnh, sử dụng để sắp xếp *array arr*; *arr* là *array* của các phần tử có kiểu được định nghĩa sẵn trong C#.
- **Array.Reverse(arr):** hàm tĩnh, sử dụng để đảo ngược vị trí của các phần tử có trong *array arr*.

Ví dụ 2.3: Minh họa việc khai báo một array gồm các string, sắp xếp theo thứ tự ABC, đảo ngược các phần tử, rồi in các phần tử ra:

```
string[] hoTen = { "Nguyen Van Trung", "Nguyen Hoang Ha", "Tran Nguyen Phong" };
//in danh sách các phần tử trong array
for (int i = 0; i < hoTen.Length; i++)
    Console.WriteLine(hoTen[i]);
Array.Sort(hoTen);
Array.Reverse(hoTen);
//in danh sách các phần tử trong array sử dụng cú pháp foreach
foreach (string stHoTen in hoTen)
    Console.WriteLine(stHoTen);
```

1.1.2. Mảng hai chiều

Cú pháp khai báo

<Kiểu dữ liệu>[,] Tên_mảng;
 hoặc *<Kiểu dữ liệu>[,] Tên_mảng = new <Kiểu dữ liệu>[số hàng, số cột];*
 hoặc *<Kiểu dữ liệu>[,] Tên_mảng = new <Kiểu dữ liệu>[,]{<các giá trị>;}*
 hoặc *<Kiểu dữ liệu>[,] Tên_mảng = {<các giá trị>;}*

Ví dụ 2.4:

```
//khai báo mảng số nguyên có 2 dòng và 3 cột
int[,] myRectArray = new int[2,3];
//khai báo mảng số nguyên 4 hàng 2 cột
int[,] myRectArray = new int[,] { {1,2},{3,4},{5,6},{7,8}};
//khai báo mảng string có 2 dòng 4 cột
string[,] beatleName = { {"Lennon","John"}, {"McCartney","Paul"}, {"Harrison","George"}, {"Starkey","Richard"} };
```

Cách sử dụng

Để làm việc với mảng hai chiều, người lập trình có thể can thiệp trực tiếp vào từng phần tử của mảng thông qua chỉ số hàng và chỉ số cột.

Ví dụ 2.5:

```
A[0,1]=2; //Gán giá trị 2 vào phần tử ở hàng 0 cột 1
Console.Write(A[2,3]); //In giá trị của phần tử ở hàng 2 cột 3 ra màn hình
```

Vì cấu trúc lưu trữ của ma trận bao gồm các hàng và các cột nên người lập trình thường sử dụng hai vòng lặp lồng nhau để thực hiện việc duyệt cả ma trận.

Ví dụ 2.6: Duyệt ma trận bằng cách sử dụng hai vòng lặp lồng nhau, trong quá trình lặp, chương trình sẽ hiển thị các chuỗi trong ma trận:

```
using System;

class Program
```

```
{
    static void Main(string[] args)
    {
        string[,] beatleName = { { "Lennom", "John" }, { "McCartney", "Paul" }, {
"Harrison", "George" }, { "Starkey", "Richard" } };
        for (int i = 0; i < 4; i++)
        {
            for (int j = 0; j < 2; j++)
            {
                Console.Write(beatleName[i, j] + " ");
            }
            Console.WriteLine();
        }
        Console.ReadLine();
    }
}
```

1.2. Chuỗi ký tự

1.2.1. Khái niệm

Chuỗi là một đối tượng kiểu String, có giá trị là văn bản. Bên trong, văn bản được lưu trữ dưới dạng một bộ sưu tập chỉ đọc tuần tự của các đối tượng kiểu Char. Không có ký tự kết thúc null ở cuối chuỗi C#, do đó một chuỗi C# có thể chứa bất kỳ số ký tự null nào ('\0'). Thuộc tính Length của một chuỗi đại diện cho số đối tượng kiểu Char mà nó chứa, không phải là số lượng ký tự Unicode. Để truy cập các mã Unicode riêng lẻ trong một chuỗi, hãy sử dụng đối tượng StringInfo.

string vs. System.String

Trong C #, string là từ khóa dành cho Chuỗi. Do đó, String và string là tương đương, bất chấp khuyến nghị nên sử dụng từ khóa được cung cấp string vì nó hoạt động ngay cả khi không có using System. Các lớp String cung cấp nhiều phương thức để tạo, thao tác, và so sánh chuỗi một cách an toàn. Ngoài ra, ngôn ngữ C# nạp chồng một số toán tử để đơn giản hóa các hoạt động chuỗi phổ biến.

1.2.2. Khai báo và khởi tạo chuỗi

Ta có thể khai báo và khởi tạo chuỗi theo nhiều cách khác nhau, như trong ví dụ sau.

Ví dụ 2.7:

```
// Declare without initializing.
string message1;

// Initialize to null.
string message2 = null;

// Initialize as an empty string.
// Use the Empty constant instead of the literal "".
string message3 = System.String.Empty;

// Initialize with a regular string literal.
string oldPath = "c:\\Program Files\\Microsoft Visual Studio 8.0";

// Initialize with a verbatim string literal.
string newPath = @"c:\Program Files\Microsoft Visual Studio 9.0";

// Use System.String if you prefer.
```

```
System.String greeting = "Hello World!";

// In local variables (i.e. within a method body)
// you can use implicit typing.
var temp = "I'm still a strongly-typed System.String!";

// Use a const string to prevent 'message4' from
// being used to store another string value.
const string message4 = "You can't get rid of me!";

// Use the String constructor only when creating
// a string from a char*, char[], or sbyte*. See
// System.String documentation for details.
char[] letters = { 'A', 'B', 'C' };
string alphabet = new string(letters);
```

Lưu ý rằng ta không sử dụng toán tử **new** để tạo một đối tượng chuỗi ngoại trừ khi khởi tạo chuỗi bằng một mảng ký tự.

Khởi tạo một chuỗi với giá trị hằng số Empty để tạo một đối tượng String mới là chuỗi có độ dài bằng không. Biểu diễn theo nghĩa đen của chuỗi có độ dài bằng 0 là "". Bằng cách khởi tạo chuỗi với giá trị Empty thay vì null, ta có thể giảm khả năng xảy ra NullReferenceException. Người lập trình nên sử dụng phương thức IsNullOrEmpty (String) tĩnh để xác minh giá trị của một chuỗi trước khi cố gắng truy cập vào nó.

Tính bất biến của các đối tượng chuỗi

Đối tượng chuỗi là bất biến: chúng không thể thay đổi sau khi đã được tạo. Tất cả các phương thức String và toán tử C# xuất hiện để sửa đổi một chuỗi thực sự trả về kết quả trong một đối tượng chuỗi mới. Trong ví dụ sau, khi nội dung của s1 và s2 được nối với nhau để tạo thành một chuỗi đơn, hai chuỗi ban đầu không bị sửa đổi. Các toán tử += sẽ tạo ra một chuỗi mới có chứa các nội dung kết hợp. Đối tượng mới đó được gán cho biến s1 và đối tượng ban đầu tương ứng với s1 sẽ được giải phóng để thu gom rác vì không có biến nào khác giữ tham chiếu đến nó.

Ví dụ 2.8:

```
string s1 = "A string is more ";
string s2 = "than the sum of its chars.";

// Concatenate s1 and s2. This actually creates a new
// string object and stores it in s1, releasing the
// reference to the original object.
s1 += s2;

System.Console.WriteLine(s1);
// Output: A string is more than the sum of its chars.
```

Bởi vì "modification" một chuỗi thực sự là tạo một chuỗi mới, do vậy phải thận trọng khi tạo tham chiếu đến chuỗi. Nếu bạn tạo một tham chiếu đến một chuỗi và sau đó "modify" chuỗi ban đầu, thì tham chiếu sẽ tiếp tục trỏ đến đối tượng ban đầu thay vì đối tượng mới được tạo khi chuỗi được sửa đổi. Đoạn mã sau minh họa điều này.

Ví dụ 2.9:

```
string s1 = "Hello ";
string s2 = s1;
s1 += "World";

System.Console.WriteLine(s2);
//Output: Hello
```

1.2.3. Sửa đổi nội dung chuỗi

Thay thế văn bản

Đoạn mã sau tạo một chuỗi mới bằng cách thay thế văn bản hiện có bằng một văn bản khác.

Ví dụ 2.10:

```
using System;

string source = "The mountains are behind the clouds today.";

// Replace one substring with another with String.Replace.
// Only exact matches are supported.
var replacement = source.Replace("mountains", "peaks");
Console.WriteLine($"The source string is <{source}>");
Console.WriteLine($"The updated string is <{replacement}>");
```

Đoạn mã trên thể hiện thuộc tính bất biến này của chuỗi. Ta có thể thấy trong ví dụ trên rằng chuỗi gốc source, không được sửa đổi. Phương thức String.Replace tạo ra một string mới có chứa những sửa đổi.

Phương thức Replace có thể thay thế các chuỗi hoặc các ký tự cụ thể trong chuỗi. Trong cả hai trường hợp, mọi lần xuất hiện của văn bản được tìm thấy đều được thay thế. Ví dụ sau thay thế tất cả các ký tự trắng ' ' bằng ký tự gạch dưới '_'.

Ví dụ 2.10:

```
using System;

string source = "The mountains are behind the clouds today.";

// Replace all occurrences of one char with another.
var replacement = source.Replace(' ', '_');
Console.WriteLine(source);
Console.WriteLine(replacement);
```

Chuỗi nguồn không thay đổi và một chuỗi mới được trả về cùng với chuỗi thay thế.

Cắt bỏ khoảng trắng

Ta có thể sử dụng các phương thức String.Trim(), String.TrimStart() và String.TrimEnd() để loại bỏ bất kỳ khoảng trắng nào ở đầu hoặc cuối. Đoạn mã sau đây cho thấy một ví dụ về từng loại. Chuỗi nguồn không thay đổi; các phương thức này trả về một chuỗi mới với nội dung đã được sửa đổi.

Ví dụ 2.11:

```
// Remove trailing and leading white space.
using System;
```

```
string source = "    I'm wider than I need to be.    ";
// Store the results in a new string variable.
var trimmedResult = source.Trim();
var trimLeading = source.TrimStart();
var trimTrailing = source.TrimEnd();
Console.WriteLine($"<{source}>");
Console.WriteLine($"<{trimmedResult}>");
Console.WriteLine($"<{trimLeading}>");
Console.WriteLine($"<{trimTrailing}>");
```

Xóa văn bản

Ta có thể xóa văn bản khỏi một chuỗi bằng phương thức `String.Remove()`. Phương thức này loại bỏ một số ký tự bắt đầu từ một chỉ mục cụ thể. Ví dụ sau đây cho thấy cách sử dụng `String.IndexOf` theo sau bởi `Remove` để xóa văn bản khỏi một chuỗi.

Ví dụ 2.12:

```
using System;

string source = "Many mountains are behind many clouds today.";
// Remove a substring from the middle of the string.
string toRemove = "many ";
string result = string.Empty;
int i = source.IndexOf(toRemove);
if (i >= 0)
{
    result = source.Remove(i, toRemove.Length);
}
Console.WriteLine(source);
Console.WriteLine(result);
```

Thay thế các mẫu phù hợp

Ta có thể sử dụng biểu thức chính quy (regular expressions) để thay thế các mẫu đối sánh văn bản bằng văn bản mới, có thể được xác định bởi một mẫu. Ví dụ sau sử dụng lớp `System.Text.RegularExpressions.Regex` để tìm một mẫu trong chuỗi nguồn và thay thế nó bằng cách viết hoa thích hợp. Phương thức `Regex.Replace(String, String, MatchEvaluator, RegexOptions)` nhận một hàm cung cấp logic của phép thay thế làm một trong các đối số của nó. Trong ví dụ này, hàm `LocalReplaceMatchCase` là một hàm cục bộ được khai báo bên trong phương thức mẫu. `LocalReplaceMatchCase` sử dụng lớp `System.Text.StringBuilder` để xây dựng chuỗi thay thế với cách viết hoa thích hợp.

Biểu thức chính quy hữu ích nhất để tìm kiếm và thay thế văn bản tuân theo một mẫu, thay vì văn bản đã biết. Mẫu tìm kiếm `"the\s"` tìm kiếm từ `"the"` theo sau là một ký tự khoảng trắng. Phần đó của mẫu đảm bảo rằng nó không khớp với từ `"there"` trong chuỗi nguồn. Để biết thêm thông tin về các thành phần của ngôn ngữ biểu thức chính quy, hãy tìm hiểu thêm về Regular Expression Language.

Ví dụ 2.13:

```
using System;

string source = "The mountains are still there behind the clouds today.";
```

```
// Use Regex.Replace for more flexibility.
// Replace "the" or "The" with "many" or "Many".
// using System.Text.RegularExpressions
string replaceWith = "many ";
source = System.Text.RegularExpressions.Regex.Replace(source, "the\\s",
    LocalReplaceMatchCase,
    System.Text.RegularExpressions.RegexOptions.IgnoreCase);
Console.WriteLine(source);

string LocalReplaceMatchCase(System.Text.RegularExpressions.Match
matchExpression)
{
    // Test whether the match is capitalized
    if (Char.IsUpper(matchExpression.Value[0]))
    {
        // Capitalize the replacement string
        System.Text.StringBuilder replacementBuilder = new
        System.Text.StringBuilder(replaceWith);
        replacementBuilder[0] = Char.ToUpper(replacementBuilder[0]);
        return replacementBuilder.ToString();
    }
    else
    {
        return replaceWith;
    }
}
```

Phương thức `StringBuilder.ToString()` trả về một chuỗi bất biến với những nội dung trong đối tượng `StringBuilder`.

Sửa đổi các ký tự riêng lẻ

Ta có thể tạo một mảng ký tự từ một chuỗi, sửa đổi nội dung của mảng, sau đó tạo một chuỗi mới từ nội dung đã sửa đổi của mảng.

Ví dụ sau đây cho thấy cách thay thế một tập hợp các ký tự trong một chuỗi. Đầu tiên, nó sử dụng phương thức `String.ToCharArray()` để tạo một mảng ký tự. Nó sử dụng phương thức `IndexOf()` để tìm chỉ mục bắt đầu của từ "fox". Ba ký tự tiếp theo được thay thế bằng một từ khác. Cuối cùng, một chuỗi mới được xây dựng từ mảng ký tự được cập nhật.

Ví dụ 2.14:

```
using System;

string phrase = "The quick brown fox jumps over the fence";
Console.WriteLine(phrase);

char[] phraseAsChars = phrase.ToCharArray();
int animalIndex = phrase.IndexOf("fox");
if (animalIndex != -1)
{
    phraseAsChars[animalIndex++] = 'c';
    phraseAsChars[animalIndex++] = 'a';
    phraseAsChars[animalIndex] = 't';
}

string updatedPhrase = new string(phraseAsChars);
Console.WriteLine(updatedPhrase);
```


Lập trình xây dựng nội dung chuỗi

Vì chuỗi là bất biến, các ví dụ trước đều tạo chuỗi tạm thời hoặc tạo mảng ký tự. Trong các tình huống hiệu suất cao, có thể nên tránh việc truy cập bộ nhớ heap. .NET cung cấp phương thức `String.Create()` cho phép ta “lập trình” các ký tự nội dung của chuỗi thông qua lệnh callback trong khi tránh cấp phát chuỗi tạm thời trung gian.

Ví dụ 2.15:

```
// constructing a string from a char array, prefix it with some additional
// characters
using System;

char[] chars = { 'a', 'b', 'c', 'd', '\0' };
int length = chars.Length + 2;
string result = string.Create(length, chars, (Span<char> strContent, char[]
charArray) =>
{
    strContent[0] = '0';
    strContent[1] = '1';
    for (int i = 0; i < charArray.Length; i++)
    {
        strContent[i + 2] = charArray[i];
    }
});

Console.WriteLine(result);
```

Bạn có thể sửa đổi một chuỗi trong một khối cố định với mã không an toàn, nhưng bạn không nên sửa đổi nội dung một chuỗi sau khi tạo mới chuỗi. Làm như vậy sẽ phá vỡ mọi thứ theo những cách không thể đoán trước.

1.2.4. Regular và Verbatim String

Sử dụng các ký tự chuỗi regular khi bạn phải nhúng các ký tự đặc biệt do C# cung cấp, như được thể hiện trong ví dụ sau.

Ví dụ 2.16:

```
string columns = "Column 1\tColumn 2\tColumn 3";
//Output: Column 1      Column 2      Column 3

string rows = "Row 1\r\nRow 2\r\nRow 3";
/* Output:
Row 1
Row 2
Row 3
*/

string title = "\"The \u00C6olean Harp\"", by Samuel Taylor Coleridge";
//Output: "The Æolean Harp", by Samuel Taylor Coleridge
```

Bảng sau thể hiện một số ký tự đặc biệt trong C#:

Escape sequence	Character name	Unicode encoding

\'	Single quote	0x0027
\"	Double quote	0x0022
\\	Backslash	0x005C
\0	Null	0x0000
\a	Alert	0x0007
\b	Backspace	0x0008
\f	Form feed	0x000C
\n	New line	0x000A
\r	Carriage return	0x000D
\t	Horizontal tab	0x0009
\v	Vertical tab	0x000B
\u	Unicode escape sequence (UTF-16)	\uHHHH (range: 0000 - FFFF; example: \u00E7 = "ç")
\U	Unicode escape sequence (UTF-32)	\U00HHHHHH (range: 000000 - 10FFFF; example: \U0001F47D = "👹")
\x	Unicode escape sequence similar to "\u" except with variable length	\xH[H][H][H] (range: 0 - FFFF; example: \x00E7 or \x0E7 or \xE7 = "ç")

Sử dụng chuỗi verbatim để thuận tiện và dễ đọc hơn khi văn bản chuỗi chứa các ký tự gạch chéo ngược, ví dụ như trong đường dẫn tệp. Bởi vì chuỗi verbatim bảo toàn các ký tự xuống dòng như một phần của văn bản chuỗi, chúng có thể được sử dụng để khởi tạo chuỗi nhiều dòng. Sử dụng dấu ngoặc kép để nhúng dấu ngoặc kép bên trong một chuỗi verbatim. Ví dụ sau đây cho thấy một số cách sử dụng phổ biến cho chuỗi verbatim.

Ví dụ 2.17:

```
string filePath = @"C:\Users\scoleridge\Documents\";
//Output: C:\Users\scoleridge\Documents\

string text = @"My pensive SARA ! thy soft cheek reclined
    Thus on mine arm, most soothing sweet it is
    To sit beside our Cot,...";
/* Output:
```

```
My pensive SARA ! thy soft cheek reclined
  Thus on mine arm, most soothing sweet it is
  To sit beside our Cot, ...
*/
```

```
string quote = @"Her name was ""Sara.""";
//Output: Her name was "Sara."
```

1.3. Cấu trúc:

1.3.1. Khái niệm Struct

Struct là một kiểu dữ liệu có cấu trúc, được kết hợp từ các kiểu dữ liệu nguyên thủy do người lập trình định nghĩa để thuận tiện trong việc quản lý dữ liệu và lập trình.

Xét bài toán sau:

Ta cần lưu trữ thông tin nhân sự của một công ty gồm có các thông tin như

- Mã nhân viên.
- Họ tên.
- Nơi sinh.
- Số chứng minh thư nhân dân.

Khi đó, để lưu thông tin của 1 nhân viên ta cần 4 biến chứa 4 thông tin trên. Việc phải duy trì và thao tác với các biến riêng lẻ như vậy khiến công việc trở nên phiền toái, khó thao tác, khó kiểm soát. Từ đó người ta đưa ra khái niệm kiểu dữ liệu có cấu trúc để giải quyết vấn đề trên.

Ý tưởng là đóng gói các thông tin đó vào 1 đối tượng duy nhất. Như vậy thay vì phải khai báo 40 biến cho 10 nhân viên thì ta chỉ cần khai báo 1 mảng 10 phần tử mà mỗi phần tử có kiểu dữ liệu ta đã định nghĩa.

Đặc điểm của struct:

- Là một kiểu dữ liệu tham trị (**kiểu dữ liệu tham trị** đã được trình bày ở bài trước, trong phần Kiểu dữ liệu)
- Dùng để đóng gói các trường dữ liệu khác nhau nhưng có liên quan đến nhau.
- Bên trong struct ngoài các biến có kiểu dữ liệu cơ bản còn có các phương thức, các struct khác.
- Muốn sử dụng phải khởi tạo cấp phát vùng nhớ cho đối tượng thông qua toán tử new.
- Struct không được phép kế thừa (khái niệm về kế thừa sẽ trình bày trong phần Lập trình hướng đối tượng).

1.3.2. Khai báo và sử dụng Struct

Cú pháp khai báo:

```
struct <tên struct>
```

```
{
    public <danh sách các biến>;
}
```

Trong đó:

- <tên struct> là tên kiểu dữ liệu do mình tự đặt và tuân thủ theo quy tắc đặt tên biến.
- <danh sách các biến> là danh sách các biến thành phần được khai báo như khai báo biến bình thường.
- Từ khoá public là từ khoá chỉ định phạm vi truy cập. Trong ngữ cảnh hiện tại thì có thể hiểu từ khoá này giúp cho người khác có thể truy xuất được để sử dụng.

Ví dụ 2.18:

```
struct Coordinate
{
    public int x;
    public int y;
}

Coordinate point;
Console.WriteLine(point.x); // Compile time error

point.x = 10;
point.y = 20;
Console.WriteLine(point.x); //output: 10
Console.WriteLine(point.y); //output: 20
```

Nhận xét chung:

- Kiểu dữ liệu Coordinate có thể dùng làm kiểu dữ liệu cho biến, parameter cho phương thức. Ngoài ra còn có thể làm kiểu trả về cho phương thức.
- Các thành phần dữ liệu bên trong được truy xuất thông qua dấu “.”
- Vì struct là kiểu tham trị nên khi truyền vào các phương thức thì giá trị của nó sau khi kết thúc phương thức sẽ không thay đổi. Do đó cần sử dụng từ khoá out để có thể cập nhật giá trị thay đổi khi kết thúc phương thức.

1.4. Các Collections

Đối với nhiều ứng dụng, ta muốn tạo và quản lý các nhóm đối tượng liên quan. Có hai cách để nhóm các đối tượng: bằng cách tạo mảng đối tượng và bằng cách tạo tập hợp các đối tượng.

Mảng hữu ích nhất để tạo và làm việc với một số lượng cố định các đối tượng có cùng kiểu.

Collection cung cấp một cách linh hoạt hơn để làm việc với các nhóm đối tượng. Không giống như mảng, nhóm đối tượng mà ta làm việc có thể mở rộng và thu hẹp

một cách linh hoạt khi nhu cầu của ứng dụng thay đổi. Đối với một số collection, ta có thể gán khóa cho bất kỳ đối tượng nào mà ta đưa vào collection để có thể nhanh chóng truy xuất đối tượng bằng cách sử dụng khóa.

Collection là một lớp, vì vậy ta phải khai báo một thể hiện của lớp trước khi ta có thể thêm các phần tử vào collection đó.

Nếu collection chỉ chứa các phần tử của một kiểu dữ liệu, ta có thể sử dụng một trong các lớp trong namespace `System.Collections.Generic`. Generic collection thực thi an toàn kiểu để không có kiểu dữ liệu nào khác có thể được thêm vào nó. Khi truy xuất một phần tử từ một Generic collection, ta không phải xác định kiểu dữ liệu của nó hoặc chuyển đổi kiểu dữ liệu cho nó.

1.4.1. Sử dụng collection đơn giản

Các ví dụ trong phần này sử dụng lớp `List<T>`, cho phép làm việc với một danh sách các đối tượng có chung một kiểu dữ liệu (T).

Ví dụ 2.19:

Ví dụ sau tạo một danh sách các chuỗi và sau đó lặp qua các chuỗi bằng cách sử dụng câu lệnh `foreach`.

```
// Create a list of strings.
using System.Collections.Generic;
using System;

var salmons = new List<string>();
salmons.Add("chinook");
salmons.Add("coho");
salmons.Add("pink");
salmons.Add("sockeye");

// Iterate through the list.
foreach (var salmon in salmons)
{
    Console.WriteLine(salmon + " ");
}

// Output: chinook coho pink sockeye
```

Nếu nội dung của collection được biết trước, ta có thể sử dụng collection constructor để khởi tạo collection.

Ví dụ 2.20:

Ví dụ sau giống với ví dụ trước, ngoại trừ một constructor được sử dụng để thêm các phần tử vào danh sách.

```
// Create a list of strings by using a
// collection initializer.
using System.Collections.Generic;
using System;

var salmons = new List<string> { "chinook", "coho", "pink", "sockeye" };

// Iterate through the list.
```

```
foreach (var salmon in salmons)
{
    Console.WriteLine(salmon + " ");
}

// Output: chinook coho pink sockeye
```

Có thể sử dụng câu lệnh for thay vì câu lệnh foreach để lặp qua một danh sách. Ta thực hiện điều này bằng cách truy cập các phần tử của danh sách theo vị trí chỉ mục. Chỉ số của các phần tử bắt đầu từ 0 và kết thúc ở số phần tử trừ đi 1.

Ví dụ 2.21:

Ví dụ sau lặp lại qua các phần tử của một danh sách bằng cách sử dụng for thay vì foreach.

```
// Create a list of strings by using a
// collection initializer.
using System.Collections.Generic;
using System;

var salmons = new List<string> { "chinook", "coho", "pink", "sockeye" };

for (var index = 0; index < salmons.Count; index++)
{
    Console.WriteLine(salmons[index] + " ");
}

// Output: chinook coho pink sockeye
```

Ví dụ 2.22:

Ví dụ sau đây loại bỏ một phần tử khỏi danh sách bằng cách chỉ định đối tượng cần loại bỏ.

```
// Create a list of strings by using a
// collection initializer.
using System.Collections.Generic;
using System;

var salmons = new List<string> { "chinook", "coho", "pink", "sockeye" };

// Remove an element from the list by specifying
// the object.
salmons.Remove("coho");

// Iterate through the list.
foreach (var salmon in salmons)
{
    Console.WriteLine(salmon + " ");
}

// Output: chinook pink sockeye
```

Ta cũng có thể sử dụng phương thức RemoveAt() để loại bỏ một phần tử của danh sách theo vị trí của nó trong danh sách.

Bảng dưới đây liệt kê một số phương thức thường dùng đối với List<T>:

Phương thức	Chức năng
-------------	-----------

Add()	Thêm một phần tử vào cuối List
Insert()	Thêm một phần tử vào vị trí xác định
Remove()	Xóa phần tử lần đầu tiên xuất hiện trong List
RemoveAt()	Xóa phần tử có chỉ số xác định
Clear()	Xóa tất cả các phần tử khỏi List
Contains()	Kiểm tra 1 phần tử có trong List không

1.4.2. Một số Generic collection khác

Bảng dưới đây liệt kê một số collection khác nằm trong namespace Generic collection. Người đọc có thể tự tìm hiểu thêm qua các nguồn tài liệu khác để vận dụng vào công việc.

Collection	Mô tả
Dictionary<TKey,TValue>	Represents a collection of key/value pairs that are organized based on the key.
<u>List<T></u>	Represents a list of objects that can be accessed by index. Provides methods to search, sort, and modify lists.
Queue<T>	Represents a first in, first out (FIFO) collection of objects.
SortedList<TKey,TValue>	Represents a collection of key/value pairs that are sorted by key based on the associated IComparer<T> implementation.
Stack<T>	Represents a last in, first out (LIFO) collection of objects.

1.5. Vào ra với file

1.5.1. Khái niệm file

File còn được gọi là tập tin hoặc được gọi là tệp, là khái niệm dùng để chỉ tập hợp những thông tin liên quan với nhau và những thông tin này được lưu trữ trên đĩa. Khác với những thông tin được lưu trữ trên RAM, các thông tin được lưu trong file phải được truy cập một cách tuần tự, nghĩa là muốn đọc hoặc ghi dữ liệu ở vị trí thứ i thì bộ xử lý phải thực hiện việc đọc hoặc ghi từ vị trí 0 đến vị trí i-1.

1.5.2. Phân loại

Tùy vào nội dung dữ liệu chứa trong file, file được chia thành 2 loại: file nhị phân (binary file) và file văn bản (text file). Trong phần này giáo trình chỉ trình bày phương pháp đọc ghi đối với file văn bản, người học tự nghiên cứu phương pháp đọc ghi file nhị phân từ các tài liệu tham khảo trong phần tài liệu tham khảo ở cuối giáo trình.

File văn bản (text file)

File văn bản là file mà dữ liệu ghi trong đó là những ký tự thuộc bảng mã ASCII chuẩn (bảng mã không bao gồm các ký tự điều khiển) trong đó các ký tự được tổ chức thành nhiều hàng. Các hàng được phân cách bằng ký tự xuống hàng. Đối với các file văn bản được tổ chức trên hệ điều hành Dos hoặc Windows của Microsoft thì các hàng được phân cách bởi một cặp ký tự bao gồm: ký tự xuống dòng và ký tự về đầu dòng. Đối với các file văn bản được tổ chức trên hệ điều hành Unix hoặc Linux thì các hàng được phân cách bởi một dấu xuống hàng. Điều này giải thích tại sao khi người sử dụng mở một số file vốn được soạn thảo trong hệ điều hành Linux bằng chương trình soạn thảo của Windows (cụ thể là notepad) thì tất cả các dòng của file được nối thành một dòng duy nhất. Các file văn bản thông thường là những file văn bản đơn giản hoặc những file mã nguồn của những ngôn ngữ lập trình hoặc những file cơ sở dữ liệu XML.

File nhị phân (binary file)

File nhị phân là file mà thông tin lưu trữ một cách tổng quát. Dữ liệu lưu trữ trên file nhị phân được chia thành từng byte, mỗi byte được đặc trưng bởi một ký tự của bảng mã ASCII mở rộng. Các file nhị phân thường có một cấu trúc hoặc định dạng riêng, những file được biên soạn theo định dạng nào thì chỉ có những phần mềm hiểu được định dạng đó mới có thể đọc được. Ví dụ những file hình ảnh thì chỉ có thể đọc được bằng những chương trình hiển thị hình ảnh. Các chương trình soạn thảo cho file văn bản thì không hiển thị chính xác file nhị phân. Cụ thể là khi người sử dụng dùng chương trình soạn thảo văn bản để mở một file thực thi (.exe) thì chương trình soạn thảo văn bản ấy chỉ hiển thị các ký tự đặc biệt. Các file nhị phân thông thường là những file thực thi (.exe), file hình ảnh (.jpg, .gif, .png, .bmp, ...), file âm thanh (.mp3, .wma, .mid, .wav, ...), file chứa các đoạn phim (.mpg, .wmv, .rm, ...) và nhiều dạng định dạng mặc định khác.

1.5.3. Đọc và ghi file văn bản

Visual Studio cung cấp nhiều đối tượng để thực hiện việc đọc và ghi file văn bản. Giáo trình này trình bày việc đọc bằng cặp đối tượng StreamReader và StreamWriter. Đây là cặp đối tượng đọc/ghi tổng quát, người lập trình có thể sử dụng cặp đối tượng này để thực hiện việc đọc hoặc ghi trên các luồng nhập xuất khác. Để sử dụng phương thức này, người lập trình phải khai báo sử dụng namespace System.IO của Visual Studio bằng lệnh using System.IO.

Đọc file văn bản bằng StreamReader

StreamReader là một lớp và lớp này thực tạo ra các đối tượng phục vụ việc đọc dữ liệu từ một luồng hoặc một file. Khi khởi tạo đối tượng, người lập trình phải cung cấp tên file cho phương thức khởi tạo. Việc khởi tạo đối tượng có thể sinh ra các lỗi như file không tồn tại, đĩa bị hỏng, ..., do đó, hàm khởi tạo phải được đặt trong khối try/catch để xử lý ngoại lệ – Xử lý ngoại lệ sẽ được trình bày ở chương tiếp theo. Đối tượng *StreamReader* cung cấp nhiều phương pháp để đọc dữ liệu, trong đó phương thức hữu hiệu nhất là phương thức *ReadLine()*. Phương thức *ReadLine()* thực hiện việc đọc từng dòng trong file văn bản và trả về một chuỗi string.

Ví dụ 2.23: cho file văn bản có tên là *input.txt*, file này chứa các số nguyên, mỗi số nguyên nằm trên một dòng. Chương trình sau sử dụng đối tượng *StreamReader* để đọc file *input.txt* và tính tổng các số nguyên trong file đó.

```
static void Main(string[] args)
{
    string inputFileName = "input.txt";
    StreamReader streamReader = null;
    try
    {
        streamReader = new StreamReader(inputFileName);
        string temp;
        int N;
        temp = streamReader.ReadLine();
        N = Convert.ToInt32(temp);
        int Sum = 0;
        for (int i = 0; i < N; i++)
        {
            temp = streamReader.ReadLine();
            int x = Convert.ToInt32(temp);
            Sum += x;
        }
        Console.WriteLine("Summary: " + Sum);
    }

    catch (Exception e)
    {
        Console.WriteLine("Can not open file " + inputFileName);
        Console.WriteLine("Error: " + e.ToString());
    }
    finally
    {
        if (streamReader != null) streamReader.Close();
    }
    Console.ReadLine();
}
```

Ghi file văn bản bằng StreamWriter

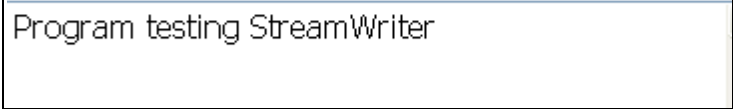
Cũng giống *StreamReader*, *StreamWriter* là một lớp và lớp này tạo ra các đối tượng phục vụ việc ghi dữ liệu vào một luồng hoặc một file. Khi khởi tạo đối tượng, người lập trình phải cung cấp tên file cho phương thức khởi tạo. Việc khởi tạo đối tượng có thể sinh ra các lỗi như đĩa bị hỏng hoặc lỗi nhập xuất, ..., do đó, hàm khởi tạo phải

được đặt trong khối try/catch để xử lý ngoại lệ. Đối tượng StreamWriter cung cấp nhiều phương pháp để ghi dữ liệu, trong đó phương thức hữu hiệu nhất là phương thức Write(). Phương thức Write() thực hiện việc ghi một chuỗi string vào file văn bản.

Ví dụ 2.24:

```
static void Main(string[] args)
{
    string outputFileName = "output.txt";
    StreamWriter streamWriter = null;
    try
    {
        streamWriter = new StreamWriter(outputFileName);
        streamWriter.Write("Program testing StreamWriter");
        Console.WriteLine("Writing file complete.");
    }
    catch (Exception e)
    {
        Console.WriteLine("Can not open file " + outputFileName);
        Console.WriteLine("Error: " + e.ToString());
    }
    finally
    {
        if (streamWriter != null) streamWriter.Close();
    }
    Console.ReadLine();
}
```

Sau khi chạy chương trình, file văn bản thu được như sau:



2. Phương thức

2.1. Khái niệm

Phương thức (method) là một chuỗi các câu lệnh được đặt tên, nhằm thực hiện các tác vụ cụ thể nào đó. Mỗi chương trình C# có ít nhất một lớp với một phương thức là Main. Một phương thức có thể có tham số hoặc không, có thể trả về hoặc không trả về giá trị, được thực thi bằng cách gọi tên phương thức và cung cấp các đối số cần thiết

Như vậy, để sử dụng một phương thức trong C#, ta cần:

- Định nghĩa phương thức
- Gọi phương thức

2.2. Định nghĩa phương thức

Khi định nghĩa một phương thức, về cơ bản, ta khai báo các phần tử trong cấu trúc của nó. Cú pháp để định nghĩa một phương thức trong C# là như sau:

<Access Specifier> <Kiểu_trả_về> <Tên_phương_thức>(<Danh_sách_tham_số>)

{

Phần thân phương thức

}

Dưới đây là chi tiết về các phần tử trong một phương thức:

- **Access Specifier:** Quy định cách thức truy cập phương thức (*private* – phương thức chỉ có thể được gọi từ một phương thức khác trong cùng lớp; *public*: phương thức được gọi từ bên ngoài lớp)
- **Kiểu_trả_về:** Một phương thức có thể trả về một giá trị. Kiểu trả về là kiểu dữ liệu của giá trị mà phương thức trả về. Nếu phương thức không trả về bất kỳ giá trị nào, thì kiểu trả về là *void*.
- **Tên_phương_thức:** Tên phương thức là một định danh duy nhất và nó là phân biệt kiểu chữ. Nó không thể giống bất kỳ định danh nào khác đã được khai báo trong lớp đó.
- **Danh_sách_tham_số:** Danh sách tham số được bao quanh trong dấu ngoặc đơn, các tham số này được sử dụng để truyền và nhận dữ liệu từ một phương thức. Danh sách tham số liên quan tới kiểu, thứ tự, và số tham số của một phương thức. Các tham số là tùy ý, tức là một phương thức có thể không chứa tham số nào.
- **Phần thân phương thức:** Phần thân phương thức chứa tập hợp các chỉ thị cần thiết để hoàn thành hoạt động đã yêu cầu.

Ví dụ 2.25:

Đoạn mã sau minh họa một phương thức nhận hai giá trị integer và trả về tổng hai số. Nó có Access Specifier, vì thế nó có thể được truy cập từ bên ngoài lớp bởi sử dụng một Instance (sự thể hiện) của lớp đó.

```
using System;

namespace MethodDemo
{
    class Test
    {
        // định nghĩa phương thức
        public static int Cong(int so1, int so2)
        {
            int tong = so1 + so2;
            return tong;
        }
        ...
    }
}
```

2.3. Gọi phương thức

Ta có thể gọi một phương thức bằng cách sử dụng tên của phương thức đó. Ví dụ sau minh họa cách gọi phương thức trong C#:

Ví dụ 2.26:

```
using System;

namespace MethodDemo
{
    class Test
    {
        // định nghĩa phương thức
        public int Cong(int so1, int so2)
        {
            int tong = so1 + so2;
            return tong;
        }

        static void Main(string[] args)
        {
            Console.WriteLine("Goi phuong thuc trong C#");
            Console.WriteLine("-----");
            /* khai bao bien cuc bo */
            int a = 15;
            int b = 20;
            int tong;
            Test n = new Test();

            //goi phuong thuc Cong
            tong = n.Cong(a, b);
            Console.WriteLine("Tong cua {0} và {1} la: {2}", a, b, tong);
            Console.ReadLine();

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

3. Xử lý ngoại lệ

3.1. Chương trình và lỗi

Chương trình (theo quan điểm của công nghệ thông tin) là một khái niệm dùng để chỉ tập hợp các mệnh lệnh có trật tự mà dựa vào đó máy tính sẽ thi hành. Quá trình con người soạn thảo tập hợp các lệnh cho chương trình được gọi là lập trình. Trong quá trình lập trình, có thể vì lý do này hoặc lý do khác mà kết quả thu được của chương trình không như mong muốn, những tình huống không mong muốn như thế gọi chung là lỗi (error hoặc defect). Trong lĩnh vực lập trình, lỗi thường được chia thành 2 loại chính: lỗi tiền biên dịch (pre-compiled error) và lỗi khi thực thi (runtime error). Lỗi tiền biên dịch (pre-compiled error) là những lỗi xuất hiện ngay khi xây dựng chương trình và được trình biên dịch thông báo trong quá trình biên dịch từ ngôn ngữ lập trình sang ngôn ngữ máy hoặc ngôn ngữ trung gian. Các lỗi thuộc dạng này thông thường là các lỗi liên quan đến cú pháp, liên quan đến khai báo dữ liệu hoặc liên quan các câu lệnh vốn được hỗ trợ sẵn. Các lỗi tiền biên dịch thông thường không nghiêm trọng và thường được phát hiện cũng như chỉnh sửa dễ dàng. Một số IDE hiện nay hỗ trợ việc kiểm tra lỗi tự động ngay trong quá trình soạn thảo điều đó làm cho các lỗi về cấu trúc giảm đi đáng kể. Các IDE hỗ trợ tính năng này bao gồm: Visual Studio (tất cả các phiên bản), NetBean, Eclipse, Jbuilder, ...

Lỗi khi thực thi (runtime error) là những lỗi xảy ra khi chạy chương trình. Đây là một dạng lỗi tiềm ẩn vì khi chương trình chạy đến một trạng thái nhất định mới sinh ra lỗi. Các lỗi thuộc dạng này rất nghiêm trọng và rất khó khắc phục vì lúc đó, chương trình đã được biên dịch sang mã máy hoặc mã trung gian. Các lỗi thuộc dạng này thường có rất nhiều nguyên nhân: có thể do thuật toán, có thể sự không tương thích của hệ thống khi dữ liệu giãn nở, hoặc cũng có thể do chính người lập trình không lường hết được tất cả các trạng thái có thể xảy ra đối với chương trình. Có một vài ví dụ cụ thể cho các lỗi dạng này mà người lập trình thường gặp phải như: lỗi khi chuyển kiểu từ kiểu chuỗi ký tự sang kiểu số nguyên, lỗi khi truy xuất đến phần tử nằm ngoài mảng, lỗi khi kết nối cơ sở dữ liệu, ...

3.2. Khái niệm ngoại lệ

Ngoại lệ (Exception) là khái niệm được sử dụng trong các ngôn ngữ lập trình bậc cao dùng để chỉ những biến cố không mong đợi khi chạy chương trình. Bản chất ngoại lệ là những lỗi xảy ra trong quá trình thực thi (runtime error). Các ngôn ngữ lập trình bậc cao như Java và .Net đều cung cấp một cơ chế xử lý ngoại lệ (gọi là cơ chế try/catch) và đặc tả các lớp đặc trưng cho từng dạng ngoại lệ mà một chương trình thường gặp.

3.3. Xử lý ngoại lệ (Exception Handling)

3.3.1. Cơ chế try/catch

Cơ chế try/catch là cơ chế được Visual Studio đưa ra để xử lý các ngoại lệ. Cơ chế này được lập trình theo cú pháp sau:

```
try
    {<Các lệnh có nguy cơ gây lỗi>}
catch (loại_biệt_lệ_tên)
    {<Các lệnh thực hiện ứng xử khi có lỗi>}
[finally
    {<Các lệnh xử lý cuối cùng>}]
```

Các lệnh có nguy cơ gây lỗi được đặt trong khối try. Các lệnh này có thể bao gồm các lệnh chuyển đổi kiểu dữ liệu (từ kiểu chuỗi sang kiểu số), lệnh liên quan đến nhập xuất và thậm chí cả toán tử chia... Khi gặp một lệnh gây ra lỗi trong khối try, chương trình thực thi sẽ lập tức triệu gọi các lệnh trong khối catch tương ứng để thực thi.

Khối catch là khối lệnh thực hiện các ứng xử khi có lỗi, trong khối này, thông thường người lập trình thực hiện việc hiển thị lỗi. Khi xây dựng khối catch, đầu tiên phải khai báo loại ngoại lệ (Exception-type). Tất cả Exception-type là những lớp kế thừa từ lớp Exception. Trong mỗi lớp đều có phương thức ToString(), phương thức này được dùng để hiển thị thông tin lỗi cho người sử dụng. Với mỗi dạng câu lệnh sẽ có một Exception-type tương ứng, người lập trình xem hướng dẫn chi tiết của các phương

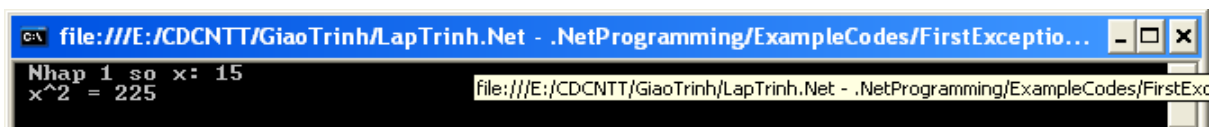
thức trong bộ hướng dẫn MSDN để chọn Exception-type tương ứng. Trong trường hợp người lập trình không quan tâm đến các loại ngoại lệ, người lập trình đó có thể sử dụng dạng cơ bản là Exception cho tất cả các ngoại lệ. Phương thức ToString() thích hợp sẽ tự động triệu gọi dựa theo tính đa hình của hướng đối tượng (Polymorphism) và chương trình luôn hiển thị chính nội dung lỗi cho dù tất cả các dạng ngoại lệ đều khai báo chung là Exception.

Khối lệnh finally được dùng để chứa các lệnh xử lý cuối cùng. Các lệnh trong khối lệnh này được tự động triệu gọi cho dù các lệnh trong khối try có sinh ra lỗi hay không. Người lập trình không nhất thiết phải khai báo khối lệnh finally, tuy vậy, khối lệnh này rất hữu ích trong việc thu dọn rác hoặc giải phóng vùng nhớ.

Ví dụ 2.27: Tính bình phương của một số nguyên được nhập từ bàn phím. Nếu dữ liệu nhập vào không phải số nguyên thì chương trình sẽ báo lỗi:

```
static void Main(string[] args)
{
    Console.Write("Nhập 1 số x: ");
    string temp;
    temp = Console.ReadLine();
    int x = 0;
    try
    {
        x = Convert.ToInt32(temp);
        Console.WriteLine("x^2 = " + x * x);
    }
    catch (FormatException e)
    {
        Console.WriteLine("Số nguyên được nhập vào không đúng định dạng.");
        Console.WriteLine("Error " + e.ToString());
    }
    Console.ReadLine();
}
```

Trong ví dụ trên, lệnh Convert.ToInt32() thực hiện việc chuyển từ kiểu chuỗi ký tự sang kiểu số nguyên và lệnh này có khả năng sinh ra lỗi khi chuỗi ký tự nhập vào không đúng định dạng số nguyên. Khi xảy ra lỗi thì chương trình sẽ không chạy những dòng tiếp theo trong khối try mà chuyển sang khối catch. Trong khối catch, các lệnh hiển thị lỗi được thực hiện. Trong trường hợp dữ liệu được nhập vào đúng định dạng của kiểu số nguyên, chương trình sẽ chạy đúng kết quả như sau:



Trong trường hợp dữ liệu nhập vào không đúng với định dạng số nguyên, chương trình sẽ báo lỗi và hiển thị cụ thể lỗi ra màn hình:

```

Nhập 1 số x: abc123
Số nguyên được nhập vào không đúng định dạng.
Error System.FormatException: Input string was not in a correct format.
   at System.Number.StringToNumber(String str, NumberStyles options, NumberBuffer& number, NumberFormatInfo info, Boolean parseDecimal)
   at System.Number.ParseInt32(String s, NumberStyles style, NumberFormatInfo info)
   at System.Convert.ToInt32(String value)
   at FirstException.Program.Main(String[] args) in E:\CDCNTT\GiaoTrinh\LapTrinh\
.Net - .NetProgramming\ExampleCodes\FirstException\Program.cs:line 17
  
```

Trong khối catch, lệnh `e.ToString()` thường được gọi để hiển thị lỗi, lỗi này được hiển thị chính xác với nội dung lỗi và dòng mã sinh ra lỗi. Với cơ chế xử lý lỗi này, người lập trình dễ dàng tìm ra lỗi và chỉnh sửa phù hợp.

3.3.2. Xử lý ngoại lệ lồng nhau

Trong quá trình lập trình, có một số tình huống mà chương trình phải thực hiện nhiều lệnh có khả năng gây lỗi liên tiếp. Để nắm bắt lỗi một cách chính xác, lập trình viên thường thực hiện việc bắt lỗi theo từng bước, nghĩa là xử lý ngoại lệ của lệnh này xong thì sẽ xử lý ngoại lệ ở lệnh tiếp theo.

Ví dụ 2.28: Tính số dư khi chia số nguyên x cho số nguyên y , hai số nguyên này được nhập từ bàn phím. Trong ví dụ này, chương trình có hai lệnh có khả năng sinh lỗi. Lệnh thứ nhất là lệnh chuyển kiểu khi thực hiện việc chuyển từ kiểu chuỗi ký tự sang kiểu số thực, lệnh này gây ra ngoại lệ khi chuỗi ký tự được nhập vào không đúng định dạng số thực. Lệnh thứ hai là lệnh thực hiện phép chia lấy số dư (%), lệnh này gây ra ngoại lệ khi số chia bằng 0.

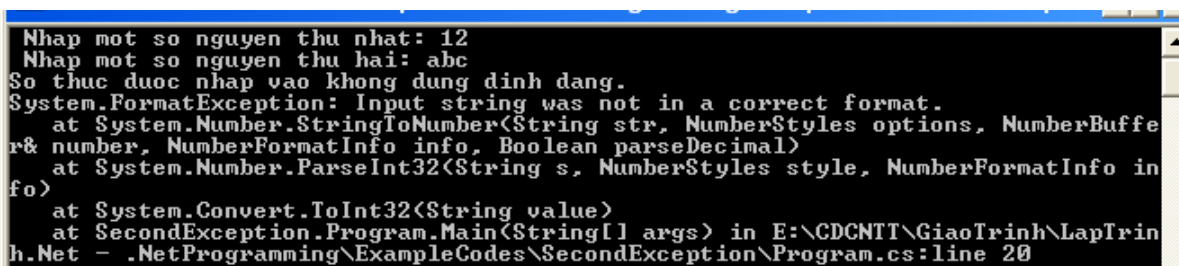

```
static void Main(string[] args)
{
    Console.Write("Nhap mot so nguyen thu nhat: ");
    string tempX = Console.ReadLine();
    Console.Write("Nhap mot so nguyen thu hai: ");
    string tempY = Console.ReadLine();
    int x = 0;
    int y = 0;
    try
    {
        x = Convert.ToInt32(tempX);
        y = Convert.ToInt32(tempY);
        int z = 0;
        try
        {
            z = x % y;
            Console.WriteLine("x % y = " + z);
        }
        catch (DivideByZeroException e1)
        {
            Console.WriteLine("Khong the chia cho 0");
            Console.WriteLine(e1.ToString());
        }
    }
    catch (FormatException e)
    {
        Console.WriteLine("So thuc duoc nhap vao khong dung dinh dang.");
        Console.WriteLine(e.ToString());
    }
    Console.ReadLine();
}
```

Trong ví dụ trên, khối lệnh try/catch bên trong thực hiện việc xử lý ngoại lệ khi chia số x cho số y để lấy số dư. Nếu $y = 0$ thì ngoại lệ được tạo ra, ngoại lệ này thuộc lớp `DivideByZeroException`. Khối lệnh try/catch bên ngoài thực hiện việc xử lý ngoại lệ khi chuyển kiểu từ kiểu số chuỗi ký tự sang kiểu số nguyên. Trong trường hợp cả hai số nhập vào x, y đều có định dạng phù hợp và y khác 0 thì chương trình chạy và cho ra kết quả đúng:



```
Nhap mot so nguyen thu nhat: 12
Nhap mot so nguyen thu hai: 5
x % y = 2
```

Trong trường hợp một trong hai số nhập vào có định dạng không phải là số nguyên thì ngoại lệ đầu tiên được xử lý. Khi đó chương trình sẽ thông báo lỗi và chương trình không tiếp tục chạy để thực hiện phép toán chia:



```
Nhap mot so nguyen thu nhat: 12
Nhap mot so nguyen thu hai: abc
So thuc duoc nhap vao khong dung dinh dang.
System.FormatException: Input string was not in a correct format.
   at System.Number.StringToNumber(String str, NumberStyles options, NumberBuffer& number, NumberFormatInfo info, Boolean parseDecimal)
   at System.Number.ParseInt32(String s, NumberStyles style, NumberFormatInfo info)
   at System.Convert.ToInt32(String value)
   at SecondException.Program.Main(String[] args) in E:\CDCNTT\GiaoTrinh\LapTrinh\h.Net - .NetProgramming\ExampleCodes\SecondException\Program.cs:line 20
```

Trong trường hợp cả hai số nhập vào đúng định dạng số nguyên nhưng y có giá trị 0 thì ngoại lệ tiếp theo được xử lý. Trong trường hợp này phép chia lấy số dư sẽ không thực hiện được và chương trình sẽ thông báo lỗi:

```
Nhap mot so nguyen thu nhât: 12
Nhap mot so nguyen thu hai: 0
Khong the chia cho 0
System.DivideByZeroException: Attempted to divide by zero.
   at SecondException.Program.Main(String[] args) in E:\CDCNTT\GiaoTrinh\LapTrinh
h.Net - .NetProgramming\ExampleCodes\SecondException\Program.cs:line 24
```

Với ví dụ trên, ta có thể dễ dàng nhận thấy việc xử lý ngoại lệ lồng nhau một cách phân cấp sẽ làm cho chương trình chặt chẽ và lỗi được nắm bắt dễ dàng. Với mỗi ngoại lệ được thông báo ra màn hình, người lập trình sẽ dễ dàng nhận thấy và chỉnh sửa ở đoạn mã phù hợp. Tuy nhiên, việc xử lý nhiều ngoại lệ lồng nhau một cách phân cấp làm cho mã nguồn của chương trình thêm nặng nề, các khối try/catch xuất hiện nhiều lần và lồng nhau làm cho cấu trúc của chương trình trở nên phức tạp.

3.3.3. Xử lý ngoại lệ song song

Như đã trình bày ở phần trên, khi có nhiều lệnh có khả năng gây ra ngoại lệ thì ta cần phải thực hiện xử lý tất cả các ngoại lệ. Để xử lý các ngoại lệ, ta thực hiện việc sử dụng cấu trúc try/catch một cách tuần tự: lệnh nào thực hiện trước thì xử lý trước, sau đó, ngay bên trong khối try, người lập trình sử dụng một cấu trúc try/catch khác để xử lý ngoại lệ cho lệnh tiếp theo. Cách làm này tuy xử lý tất cả các ngoại lệ nhưng cũng làm cho chương trình trở nên phức tạp vì phải sử dụng nhiều cấu trúc try/catch lồng nhau. Để tối ưu hóa mã lệnh và làm cho chương trình thêm tường minh, thông thường người lập trình thực hiện việc xử lý ngoại lệ song song.

Xử lý ngoại lệ song song là quá trình xử lý ngoại lệ trong đó chỉ có một khối try nhưng lại có nhiều khối catch. Tất cả các lệnh cần thiết đều đặt trong khối try và mỗi khối catch sẽ thực hiện việc bắt một ngoại lệ tương ứng. Khi chạy chương trình, ứng với một ngoại lệ được tạo ra, chương trình thực thi của .Net Framework sẽ tự động tham chiếu đến khối catch có ngoại lệ tương ứng để gọi lệnh xử lý.

Ví dụ 2.29: Trình bày việc cải tiến ví dụ ở trên, trong ví dụ này, ngoại lệ được xử lý bằng cách sử dụng hai khối catch khác nhau để bắt lấy 2 ngoại lệ khác nhau. Kết quả chạy chương trình cải tiến này hoàn toàn giống với kết quả khi chạy ví dụ trong phần trên:

```
static void Main(string[] args)
{
    Console.Write(" Nhap mot so nguyen thu nhat: ");
    string tempX = Console.ReadLine();
    Console.Write(" Nhap mot so nguyen thu hai: ");
    string tempY = Console.ReadLine();
    int x = 0;
    int y = 0;
    try
    {
        x = Convert.ToInt32(tempX);
        y = Convert.ToInt32(tempY);
        int z = 0;
        z = x % y;
        Console.WriteLine(" x % y = " + z);
    }
    catch (FormatException e)
    {
        Console.WriteLine("So thuc duoc nhap vao khong dung dinh dang.");
        Console.WriteLine(e.ToString());
    }
    catch (DivideByZeroException e1)
    {
        Console.WriteLine("Khong the chia cho 0");
        Console.WriteLine(e1.ToString());
    }
    Console.ReadLine();
}
```