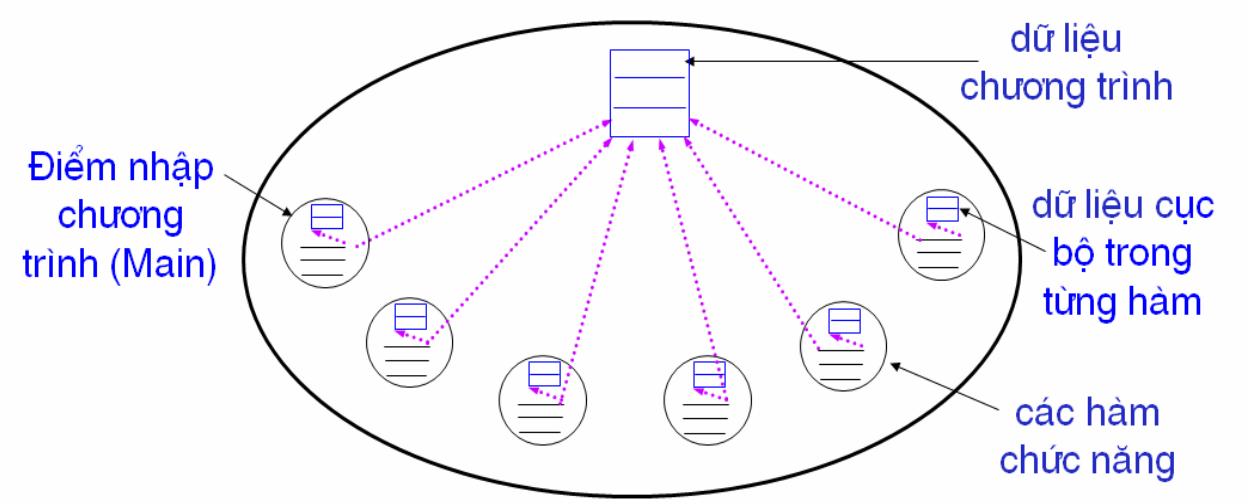
Chương 1

Các kiến thức cơ bản về lập trình C# đã học

1.1 Cấu trúc của 1 ứng dụng C# nhỏ

Trong môn kỹ thuật lập trình, chúng ta đã viết được 1 số ứng dụng C# nhỏ và đơn giản. Trong trường hợp này, 1 ứng dụng C# là 1 class gồm nhiều thuộc tính dữ liệu và nhiều hàm chức năng. Chương trình bắt đầu chạy từ hàm Main.



Xem đoạn chương trình giải phương trình bậc 2 ở chế độ text-mode sau đây :

```
using System;
namespace GPTB2 {
   class Program {
      //định nghĩa các biến cần dùng
      static double a, b, c;
      static double delta;
      static double x1, x2;
      //định nghĩa hàm nhập 3 thông số a,b,c của phương trình bậc 2
      static void NhapABC() {
         String buf;
         Console.Write("Nhập a : "); buf= Console.ReadLine();
         a = Double.Parse(buf);
```

```
    Console.Write("Nhập b : "); buf = Console.ReadLine();
    b = Double.Parse(buf);

    Console.Write("Nhập c : "); buf = Console.ReadLine();
    c = Double.Parse(buf);
}
//định nghĩa hàm tính nghiệm của phương trình bậc 2
static void GiaiPT()
{
    //tính biệt số delta của phương trình
    delta = b * b - 4 * a * c;
    if (delta >= 0) //nếu có nghiệm thực
    {
        x1 = (-b + Math.Sqrt(delta)) / 2 / a;
        x2 = (-b - Math.Sqrt(delta)) / 2 / a;
    }
}

//định nghĩa hàm xuất kết quả
static void XuatKetqua()
{
    if (delta < 0)
        //báo vô nghiệm
       Console.WriteLine("Phương trình vô nghiệm");
    else //báo có 2 nghiệm
    {
        Console.WriteLine("Phương trình có 2 nghiệm thực : ");
        Console.WriteLine("X1 = " + x1);
        Console.WriteLine("X2 = " + x2);
    }
}
//định nghĩa chương trình (hàm Main)
static void Main(string[] args)
{
```

```
            NhapABC();          //1. nhập a,b,c
            GiaiPT();                   //2. giải phương trình
            XuatKetqua();       //3. xuất kết quả
            //4. chờ người dùng ấn Enter để đóng cửa sổ Console lại.
            Console.Write("Ấn Enter để dừng chương trình : ");
            Console.Read();
        }
    }  //kết thúc class
} //kết thúc namespace
```

Quan sát cấu trúc của chương trình C# nhỏ phía trên, chúng ta có 1 số nhận xét sau :

1. Dữ liệu chương trình thường rất phong phú, đa dạng về chủng loại → Cơ chế định nghĩa kiểu dữ liệu nào được dùng để đảm bảo người lập trình có thể định nghĩa kiểu riêng mà ứng dụng của họ cần dùng ?
2. Nếu ứng dụng lớn chứa rất nhiều hàm chức năng và phải xử lý rất nhiều dữ liệu thì rất khó quản lý chúng trong 1 class đơn giản → cần 1 cấu trúc phù hợp để quản lý ứng dụng lớn.
3. Chương trình thường phải nhờ các hàm chức năng ở các class khác để hỗ trợ mình. Thí dụ ta đã gọi hàm Read, Write của class Console để nhập/xuất dữ liệu cho chương trình → Cơ chế nhờ vả nào được dùng để đảm bảo các thành phần trong ứng dụng không "quậy phá" nhau?

1.2 Kiểu dữ liệu cơ bản định sẵn

Các thuật giải chức năng của chương trình sẽ xử lý dữ liệu. Dữ liệu của chương trình thường rất phong phú, đa dạng về chủng loại. Trước hết ngôn ngữ C# (hay bất kỳ ngôn ngữ lập trình nào) phải định nghĩa 1 số kiểu được dùng phổ biến nhất trong các ứng dụng, ta gọi các kiểu này là "kiểu định sẵn".

Mỗi dữ liệu thường được để trong 1 biến. Phát biểu định nghĩa biến sẽ đặc tả các thông tin về biến đó :

- tên nhận dạng để truy xuất.
- kiểu dữ liệu để xác định các giá trị nào được lưu trong biến.
- giá trị ban đầu mà biến chứa...

Biến thuộc kiểu định sẵn sẽ chứa trực tiếp giá trị, thí dụ biến nguyên chứa trực tiếp các số nguyên, biến thực chứa trực tiếp các số thực → Ta gọi kiểu định sẵn là kiểu giá trị (value type) để phân biệt với kiểu tham khảo (reference type) trong lập trình hướng đối tượng ở các chương sau.

Kiểu tham khảo (hay kiểu đối tượng) sẽ được trình bày trong chương 2 trở đi. Đây là kiểu quyết định trong lập trình hướng đối tượng. Một biến đối tượng là biến có kiểu là tên interface hay tên class. Biến đối tượng không chứa trực tiếp đối tượng, nó chỉ chứa thông tin để truy xuất được đối tượng → Ta gọi kiểu đối tượng là kiểu tham khảo (reference type).

Sau đây là danh sách các tên kiểu cơ bản định sẵn :

- bool : kiểu luận lý, có 2 giá trị true và false.
- byte : kiểu nguyên dương 1 byte, có tầm trị từ 0 đến 255.
- sbyte : kiểu nguyên có dấu 1 byte, có tầm trị từ -128 đến 127.
- char : kiểu ký tự Unicode 2 byte, có tầm trị từ mã 0000 đến FFFF.
- short : kiểu nguyên có dấu 2 byte, tầm trị từ -32768 đến 32767.
- ushort : kiểu nguyên dương 2 byte, tầm trị từ 0 đến 65535.
- int : kiểu nguyên có dấu 4 byte, tầm trị từ -2,147,483,648 đến 2,147,483,647.

- uint : kiểu nguyên dương 4 byte, tầm trị từ 0 đến 4,294,967,295.
- long : kiểu nguyên có dấu 8 byte, tầm trị từ -2^{63} đến $2^{63}-1$.
- ulong : kiểu nguyên dương 8 byte, tầm trị từ 0 đến $2^{64}-1$.
- float : kiểu thực chính xác đơn, dùng 4 byte để miêu tả 1 giá trị thực, có tầm trị từ $\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$. Độ chính xác khoảng 7 ký số thập phân.
- double : kiểu thực chính xác kép, dùng 8 byte để miêu tả 1 giá trị thực, có tầm trị từ $\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$. Độ chính xác khoảng 15 ký số thập phân.
- decimal : kiểu thực chính xác cao, dùng 16 byte để miêu tả 1 giá trị thực, có tầm trị từ $\pm 1.0 \times 10^{-28}$ to $\pm 7.9 \times 10^{28}$. Độ chính xác khoảng 28-29 ký số thập phân.
- object (Object) : kiểu đối tượng bất kỳ, đây là 1 class định sẵn đặc biệt.

1.3 Kiểu do người lập trình tự định nghĩa - Liệt kê

Ngoài các kiểu cơ bản định sẵn, C# còn hỗ trợ người lập trình tự định nghĩa các kiểu dữ liệu đặc thù trong từng ứng dụng.

Kiểu liệt kê bao gồm 1 tập hữu hạn và nhỏ các giá trị đặc thù cụ thể. Máy sẽ mã hóa các giá trị kiểu liệt kê thành kiểu byte, short...

```
//định nghĩa kiểu chứa các giá trị ngày trong tuần
enum DayInWeek {Sat, Sun, Mon, Tue, Wed, Thu, Fri};
//định nghĩa kiểu chứa các giá trị ngày trong tuần
enum DayInWeek {Sat=1, Sun, Mon, Tue, Wed, Thu, Fri};
//định nghĩa biến chứa các giá trị ngày trong tuần
DayInWeek day = DayInWeek.Tue;
//định nghĩa kiểu chứa các giá trị nguyên trong tầm trị đặc thù
enum ManAge : byte {Max = 130, Min = 0};
```

1.4 Kiểu do người lập trình tự định nghĩa - Record

Kiểu record bao gồm 1 tập hữu hạn các thông tin cần quản lý.

```
//định nghĩa kiểu miêu tả các thông tin của từng sinh viên cần quản lý
    public struct Sinhvien {
        public String hoten;
        public String diachi;
        //các field khác
    }
```

Thật ra kiểu struct là trường hợp đặc biệt của class đối tượng mà ta sẽ trình bày chi tiết từ chương 2.

1.5 Kiểu do người lập trình tự định nghĩa - Array

Trong trường hợp ta có nhiều dữ liệu cần xử lý thuộc cùng 1 kiểu (thường xảy ra), nếu ta định nghĩa từng biến đơn để miêu tả từng dữ liệu thì rất nặng nề, thuật giải xử lý chúng cũng gặp nhiều khó khăn. Trong trường hợp này, tốt nhất là dùng kiểu Array để quản lý nhiều dữ liệu cần xử lý. Array có thể là :

- array 1 chiều.
- array nhiều chiều.
- array "jagged".

Array 1 chiều

```
int[] intList; //1.định nghĩa biến array là danh sách các số nguyên
//2. khi biết được số lượng, thiết lập số phần tử cho biến array
intList = new int[5];
//3. gán giá trị cho từng phần tử khi biết được giá trị của nó
intList[0] = 1; intList[1] = 3; intList[2] = 5;
intList[3] = 7; intList[4] = 9;
```

Nếu có đủ thông tin tại thời điểm lập trình, ta có thể viết lệnh định nghĩa biến array như sau :

```
int[] intList = new int[5] {1, 3, 5, 7, 9};
```
hay đơn giản :
```
int[] intList = new int[] {1, 3, 5, 7, 9};
```
hay đơn giản hơn nữa :
```
int[] intList = {1, 3, 5, 7, 9};
```

Array nhiều chiều

```
int[,] matran; //1. định nghĩa biến array là ma trận các số nguyên
//2. khi biết được số lượng, thiết lập số phần tử cho biến array
matran = new int[3,2];
//3. gán giá trị cho từng phần tử khi biết được giá trị của nó
matran[0,0] = 1; matran[0,1] = 2; matran[1,0] = 3;
matran[1,1] = 4; matran[2,0] = 5; matran[2,1] = 6;
```

Nếu có đủ thông tin tại thời điểm lập trình, ta có thể viết lệnh định nghĩa biến array như sau :
```
int[,] matran = new int[3,2] {{1, 2}, {3, 4}, {5,6}};
```
hay đơn giản :
```
int[,] matran = new int[,] {{1, 2}, {3, 4}, {5,6}};
```
hay đơn giản hơn nữa :
```
int[,] matran = {{1, 2}, {3, 4}, {5,6}};
```

Array "jagged"

Array "jagged" là array mà từng phần tử là array khác, các array được chứa trong array "jagged" có thể là array 1 chiều, n chiều hay là array "jagged' khác.

```
int[][] matran; //1. định nghĩa biến array "jagged"
//2. khi biết được số lượng, thiết lập số phần tử cho biến array
matran = new int[3][];
for (int i = 0; i < 3; i++) matran[i] = new int[2];
//3. gán giá trị cho từng phần tử khi biết được giá trị của nó
matran[0][0] = 1; matran[0][1] = 2; matran[1][0] = 3;
matran[1][1] = 4; matran[2][0] = 5; matran[2][1] = 6;
```

Nếu có đủ thông tin tại thời điểm lập trình, ta có thể viết lệnh định nghĩa biến array như sau :

```
int[][] array = new int [3][];
array[0] = new int[] {1, 2};
array[1] = new int[] {3, 4};
array[2] = new int[] {5,6};
```

hay đơn giản :

```
int[][] array = new int [][] {new int[]{1, 2}, new int[]{3, 4}, new int[] {5,6}};
```

hay đơn giản hơn nữa :

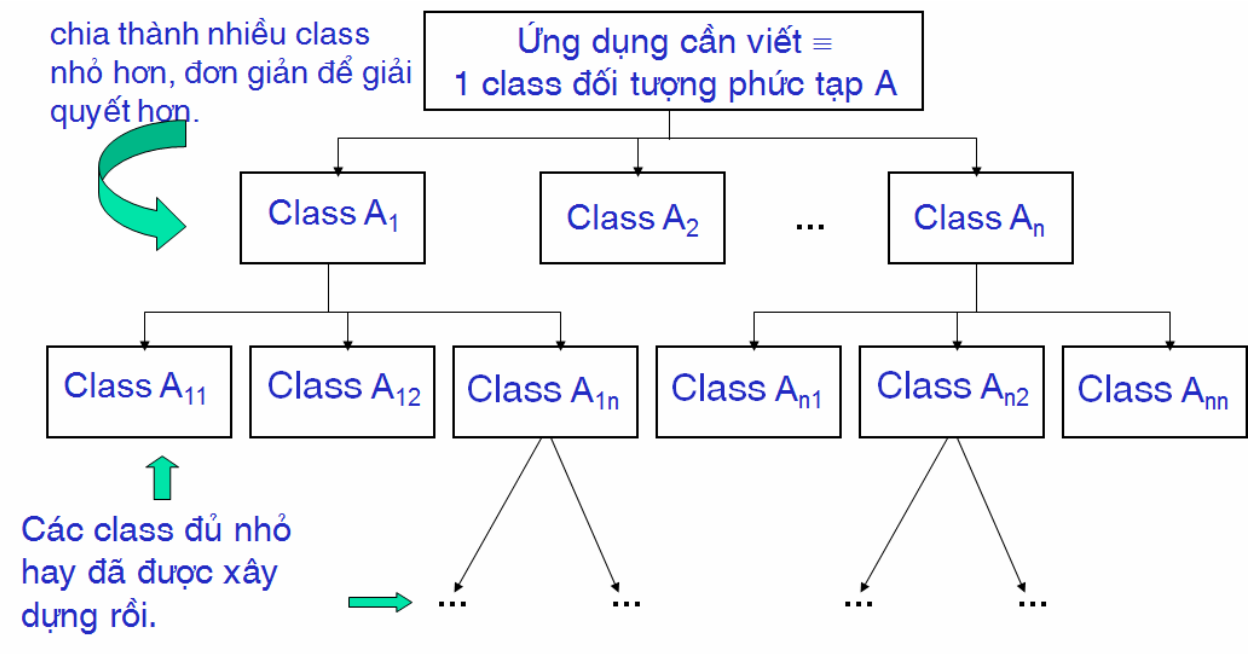
```
int[][] array = {new int[]{1, 2}, new int[]{3, 4}, new int[] {5,6}};
```

1.6 Phương pháp phân tích từ-trên-xuống

Như đã thấy ở slide trước, nếu ứng dụng lớn chứa rất nhiều hàm chức năng và phải xử lý rất nhiều dữ liệu thì rất khó quản lý chúng trong 1 class đơn giản → cần 1 cấu trúc phù hợp để quản lý ứng dụng lớn. Phương pháp được dùng phổ biến nhất là phương pháp phân tích top-down.

Nội dung của phương pháp này là phân rã class ứng dụng lớn thành n class nhỏ hơn (với n đủ nhỏ để việc phân rã đơn giản). Mỗi class nhỏ hơn, nếu còn quá phức tạp, lại được phân rã thành m class nhỏ hơn nữa (với m đủ nhỏ), cứ như vậy cho đến khi các class tìm được hoặc là class đã xây dựng rồi hoặc là class khá đơn giản, có thể xây dựng dễ dàng.

Hình vẽ sau đây cho thấy trực quan của việc phân tích top-down theo hướng đối tượng.



1.7 Namespace

Trên mỗi máy có 1 hệ thống quản lý các đối tượng được dùng bởi nhiều ứng dụng đang chạy. Mỗi ứng dụng lớn gồm rất nhiều class đối tượng khác nhau. Mỗi phần tử trong hệ thống tổng thể đều phải có tên nhận dạng duy nhất. Để đặt tên các phần tử trong hệ thống lớn sao cho mỗi phần tử có tên hoàn toàn khác nhau (để tránh tranh chấp, nhặp nhằng), C# (và các ngôn ngữ .Net khác) cung cấp phương tiện Namespace (không gian tên).

Namespace là 1 không gian tên theo dạng phân cấp : mỗi namespace sẽ chứa nhiều phần tử như struct, enum, class, interface và namespace con. Để truy xuất 1 phần tử trong namespace, ta phải dùng tên dạng phân cấp, thí dụ System.Windows.Forms.Button là tên của class Button, class miêu tả đối tượng giao diện button trong các form ứng dụng.

Trong file mã nguồn C#, để truy xuất 1 phần tử trong không gian tên khác, ta có thể dùng 1 trong 2 cách :

- dùng tên tuyệt đối dạng cây phân cấp. Thí dụ :

```
//định nghĩa 1 biến Button
System.Windows.Forms.Button objButton;
```

- dùng lệnh using <tên namespace>; Kể từ đây, ta nhận dạng phần tử bất kỳ trong namespace đó thông qua tên cục bộ. Thí dụ :

```
using System.Windows.Forms;
Button objButton; //định nghĩa 1 biến Button
TextBox objText;     //định nghĩa 1 biến TextBox
```

Microsoft đã xây dựng sẵn hàng ngàn class, interface chức năng phổ biến và đặt chúng trong khoảng 500 namespace khác nhau :

- System chứa các class và interface chức năng cơ bản nhất của hệ thống như Console (nhập/xuất văn bản), Math (các hàm toán học),..
- System.Windows.Forms chứa các đối tượng giao diện phổ dụng như Button, TextBox, ListBox, ComboBox,...
- System.Drawing chứa các đối tượng phục vụ xuất dữ liệu ra thiết bị vẽ như class Graphics, Pen, Brush,...
- System.IO chứa các class nhập/xuất dữ liệu ra file.
- System.Data chứa các class truy xuất database theo kỹ thuật ADO .Net.
- ...

1.8 Assembly

Ngoài khái niệm namespace là phương tiện đặt tên luận lý các phần tử theo dạng cây phân cấp thì C# còn cung cấp khái niệm assembly.

Assembly là phương tiện đóng gói vật lý nhiều phần tử. Một assembly là 1 file khả thi (EXE, DLL,...) chứa nhiều phần tử bên trong. Khi lập trình bằng môi trường Visual Studio .Net, ta sẽ tạo Project để quản lý việc xây dựng module chức năng nào đó (thư viện hay ứng dụng), mỗi project chứa nhiều file mã nguồn đặc tả

các thành phần trong Project đó. Khi máy dịch Project mã nguồn nó sẽ tạo ra file khả thi, ta gọi file này là 1 assembly.

Mỗi assembly có thể chứa nhiều phần tử nằm trong các namespace luận lý khác nhau. Ngược lại, 1 namespace có thể chứa nhiều phần tử mà về mặt vật lý chúng nằm trong các assembly khác nhau.

1.9 Kết chương

Chương này đã giới thiệu cấu trúc của chương trình VC# nhỏ và đơn giản gồm 1 số biến dữ liệu và 1 số hàm xử lý các biến dữ liệu, từ đó tổng kết lại các kiểu dữ liệu khác nhau có thể được dùng trong 1 chương trình, đặc biệt là các kiểu liệt kê, kiểu array, kiểu record.

Chương này cũng giới thiệu phương pháp đặt tên cho các phần tử cấu thành ứng dụng lớn 1 cách khoa học thông qua khái niệm namespace dạng cây phân cấp, cách chứa các phần tử cấu thành ứng dụng lớn trong các module vật lý được gọi là assembly.