khai.hq@ou.edu.vn

Giới thiệu môn học

Tên môn học:

Lập trình giao diện

Khoa: Công nghệ thông tin

Giới thiệu môn học

- Tên môn học: Lập trình giao diện
- Giảng viên: Hồ Quang Khải – khai.hq@ou.edu.vn; hqkhai@gmail.com
- Thời lượng: 35 tiết lý thuyết + 35 tiết thực hành

Giới thiệu môn học

- **Mục tiêu, yêu cầu môn học:**
  - » **Môn học cung cấp cho sinh viên các nguyên lý cơ bản về thiết kế và lập trình các ứng dụng có giao diện trực tiếp với người dùng cuối. Môn học này cũng giúp cho sinh viên làm quen môi trường lập trình trực quan dựa trên môi trường Microsoft Visual Studio 2005.**
  - » **Học xong môn học này sinh viên phải có khả năng sử dụng ngôn ngữ lập trình Visual C# 2005 để tạo nên các ứng dụng đơn giản sử dụng lập trình đáp ứng sự kiện và các đối tượng giao diện đồ họa cơ bản.**

Giới thiệu môn học

Tài liệu tham khảo:

[A] H.M.Deitel, P.J.Deitel, *Visual C#® 2005: How to Program, Second Edition*, Deitel™, PRENTICE HALL, 2005

[B] Jenifer Tidwell, *Designing Interfaces*, O'Reilly, 2005

[C] Karli Watsonet al. , Beginning Visual C# 2005 , Wrox Press, 2006

Giới thiệu môn học

Tài liệu tiếng Việt (không chính thức):

1. **Tự học lập trình C Sharp (pdf)**
2. **Kỹ thuật lập trình C# 2.0 (chm)**

khai.hq@ou.edu.vn

Giới thiệu môn học (2)

Nội dung chi tiết các chương:

Xem đề cương

khai.hq@ou.edu.vn

NỘI DUNG PHẦN THỰC HÀNH

- **Phần bài tập thực hành:**

  Sinh viên được hướng dẫn thực hành và cho bài tập thực hành

khai.hq@ou.edu.vn

ĐÁNH GIÁ KẾT QUẢ MÔN HỌC

- **Đánh giá kết quả học tập:**
  - **Sinh viên làm bài kiểm tra thực hành trên máy: 50%**
  - **Sinh viên thi cuối kỳ: 50% (thi viết trắc nghiệm trên giấy)**

Những điều sinh viên cần lưu ý

- SV nên chủ động tham khảo tài liệu (trước và sau buổi học), khuyến khích sử dụng tài liệu tiếng Anh
- SV nên làm đầy đủ tất cả các bài thực hành, đây là phần rèn luyện kỹ năng quan trọng khi viết các chương trình phần mềm
- SV nên tổ chức học tập theo nhóm, nếu có thắc mắc trong học tập thì trao đổi với nhau trước khi hỏi Thầy, Cô.

Chúc các bạn học tập tốt

khai.hq@ou.edu.vn

Môn học: Lập trình giao diện

Chương 1

TỔNG QUAN
VỀ LẬP TRÌNH GIAO DIỆN

khai.hq@ou.edu.vn

Nội dung:

1. Mô hình xây dựng ứng dụng phần mềm 3 tầng và đa tầng
2. Nội dung, ý nghĩa của tầng giao diện (presentation tier)
3. Các loại giao diện phần mềm thường sử dụng (cho người dùng cuối)
4. Giới thiệu nội dung chính của môn học: lập trình tầng giao diện sử dụng môi trường MS Visual Studio 2005 và ngôn ngữ lập trình C# (C Sharp)
5. Giới thiệu môi trường MS Visual Studio 2005 và ngôn ngữ lập trình C#

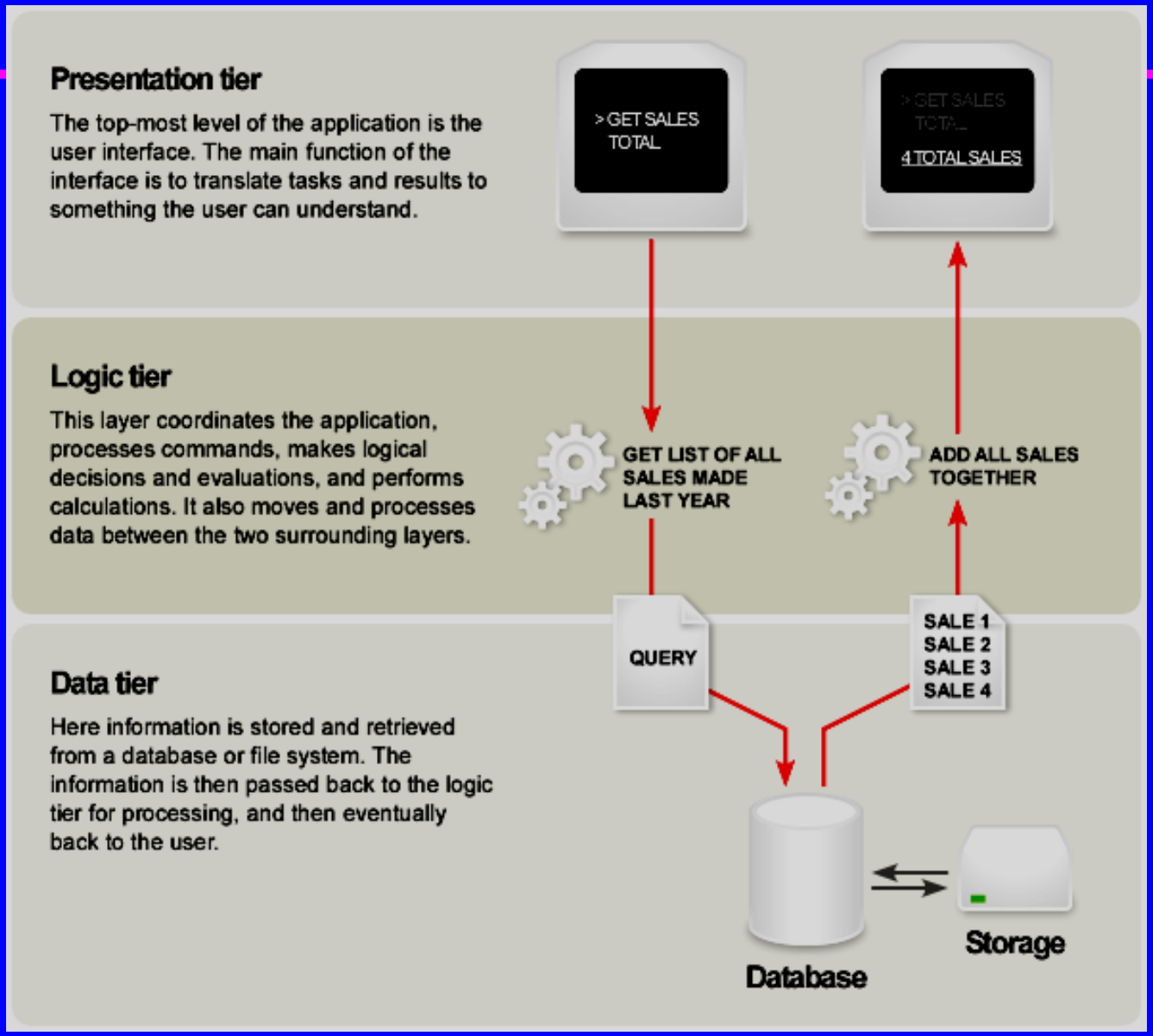
khai.hg@ou.edu.vn

Mô hình xây dựng ứng dụng phần mềm 3 tầng và đa tầng

- Thuật ngữ: 3 TẦNG ~ 3 LỚP ~ 3-TIER
- Thuật ngữ: ĐA TẦNG ~ ĐA LỚP ~ N-TIER

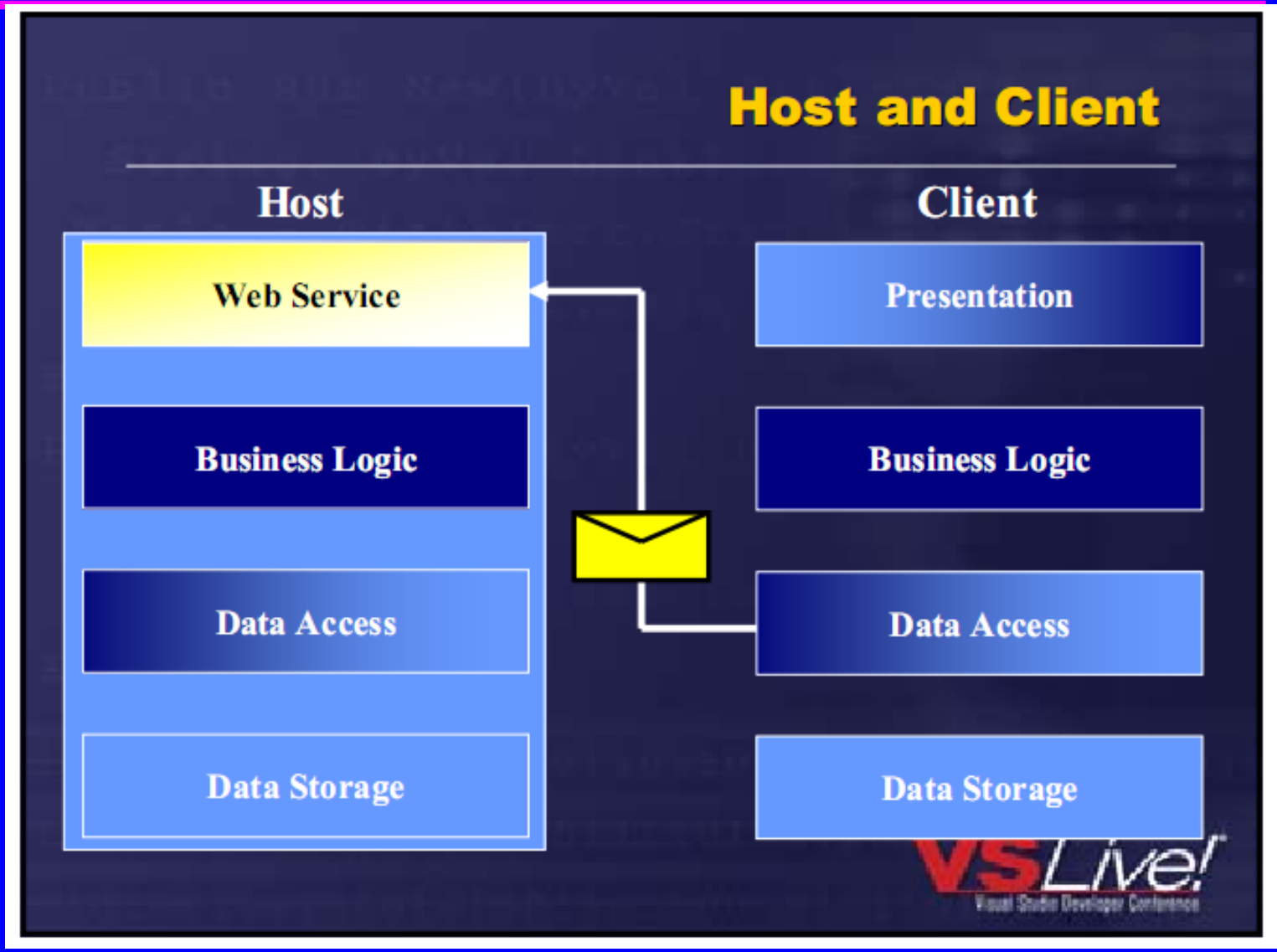
Mô hình xây dựng ứng dụng phần mềm đa tầng

khai.hq@ou.edu.vn



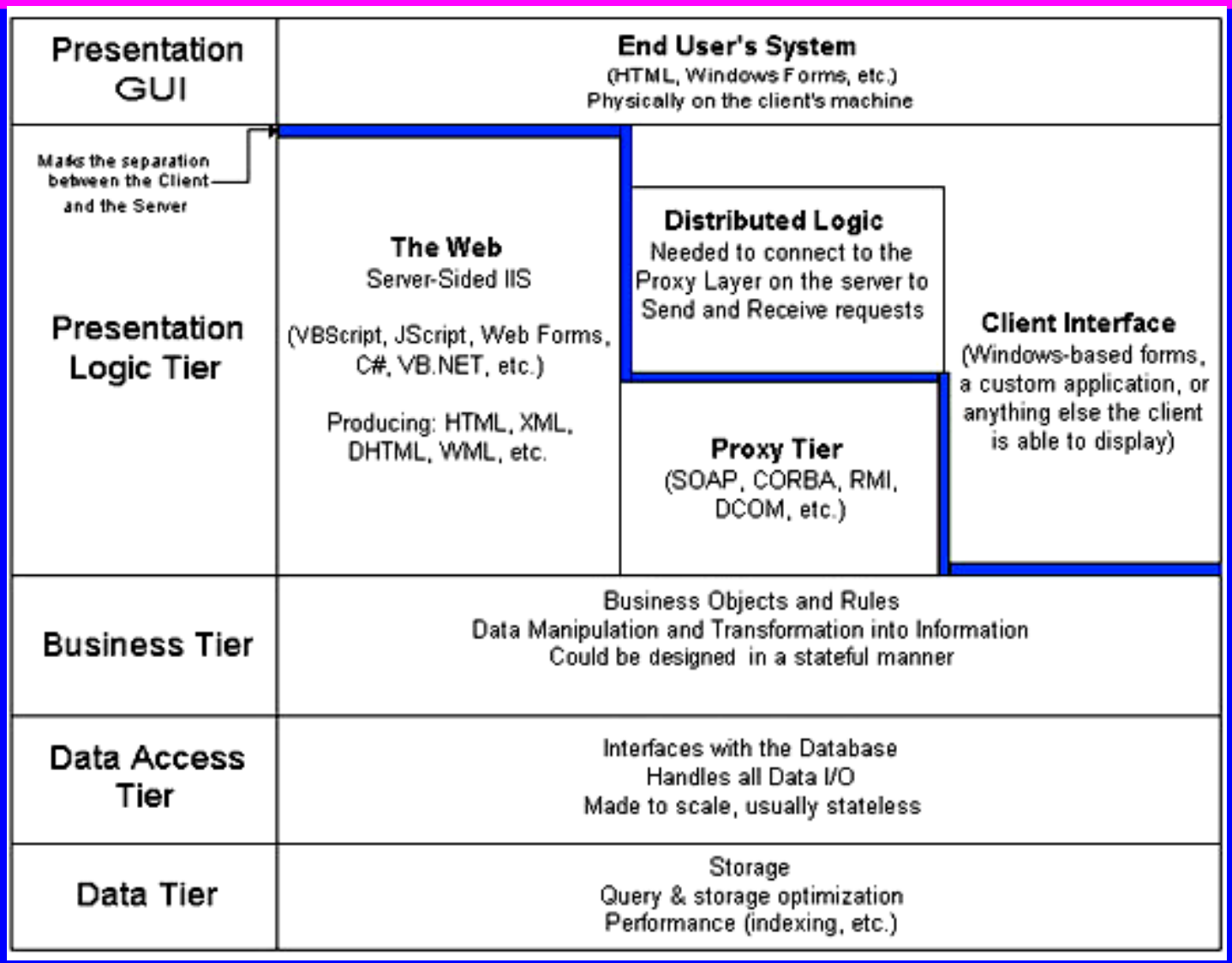
Mô hình xây dựng ứng dụng phần mềm đa tầng

khai.hq@ou.edu.vn



khai.hq@ou.edu.vn

Mô hình xây dựng ứng dụng phần mềm đa tầng



khai.hq@ou.edu.vn

Nội dung, ý nghĩa của tầng giao diện (presentation tier)

- Giao tiếp trực tiếp với người dùng
- Nhận yêu cầu từ người dùng
- Trả kết quả người dùng mong muốn
- Tạo sự tiện lợi, dễ dùng, dễ học
- Giúp thao tác nhanh chóng, chính xác
- Giúp đỡ, hướng dẫn người sử dụng

khai.hq@ou.edu.vn

Các loại giao diện phần mềm thường sử dụng (cho người dùng cuối)

- Giao diện dòng lệnh
- Giao diện cửa sổ
- Giao diện trang web

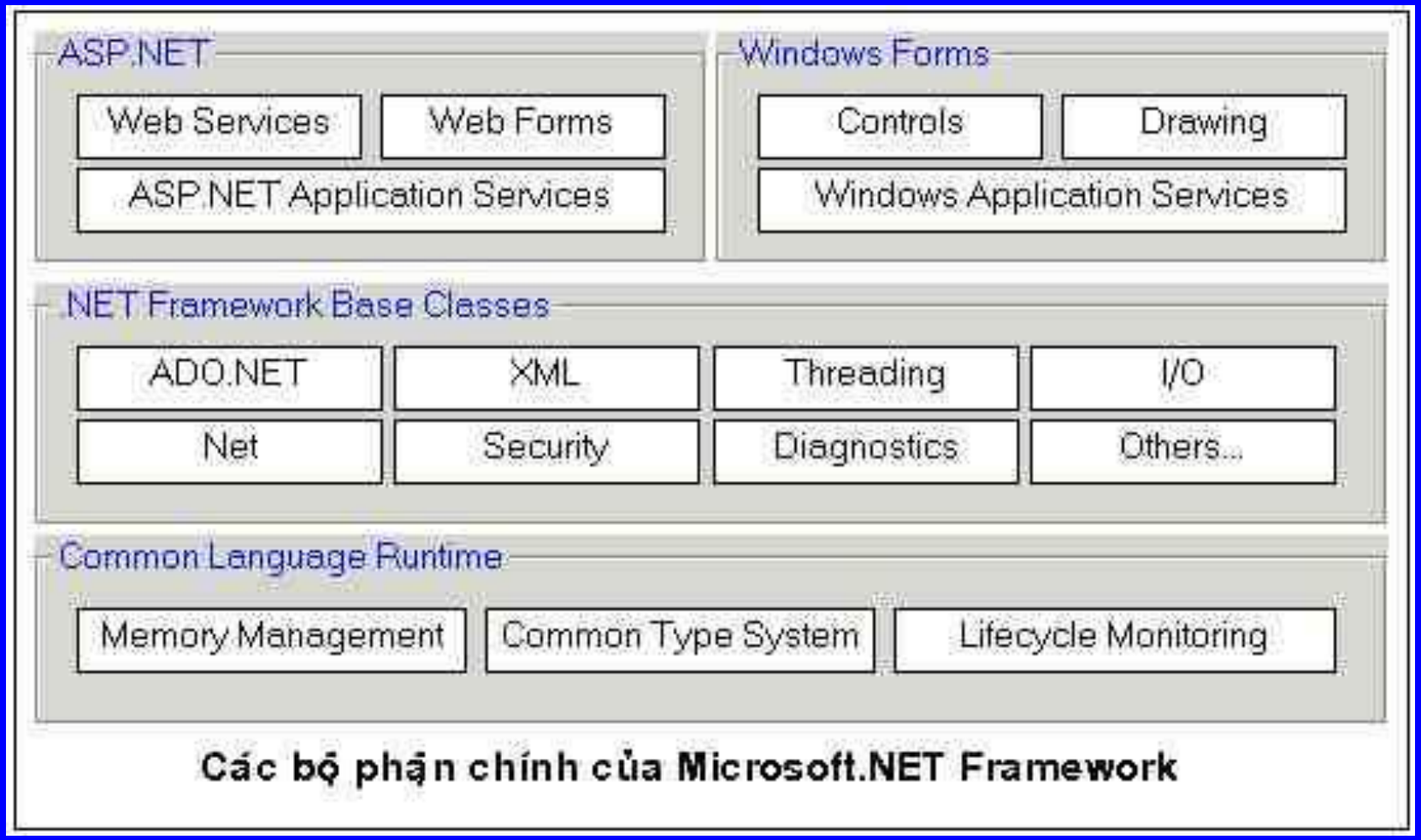
khai.hq@ou.edu.vn

Giới thiệu nội dung chính của môn học:

- Tạo tầng presentation tier
- Sử dụng giao diện cửa sổ (windows form)
- Sử dụng môi trường MS Visual Studio 2005
- Lập trình bằng ngôn ngữ C# (C Sharp)

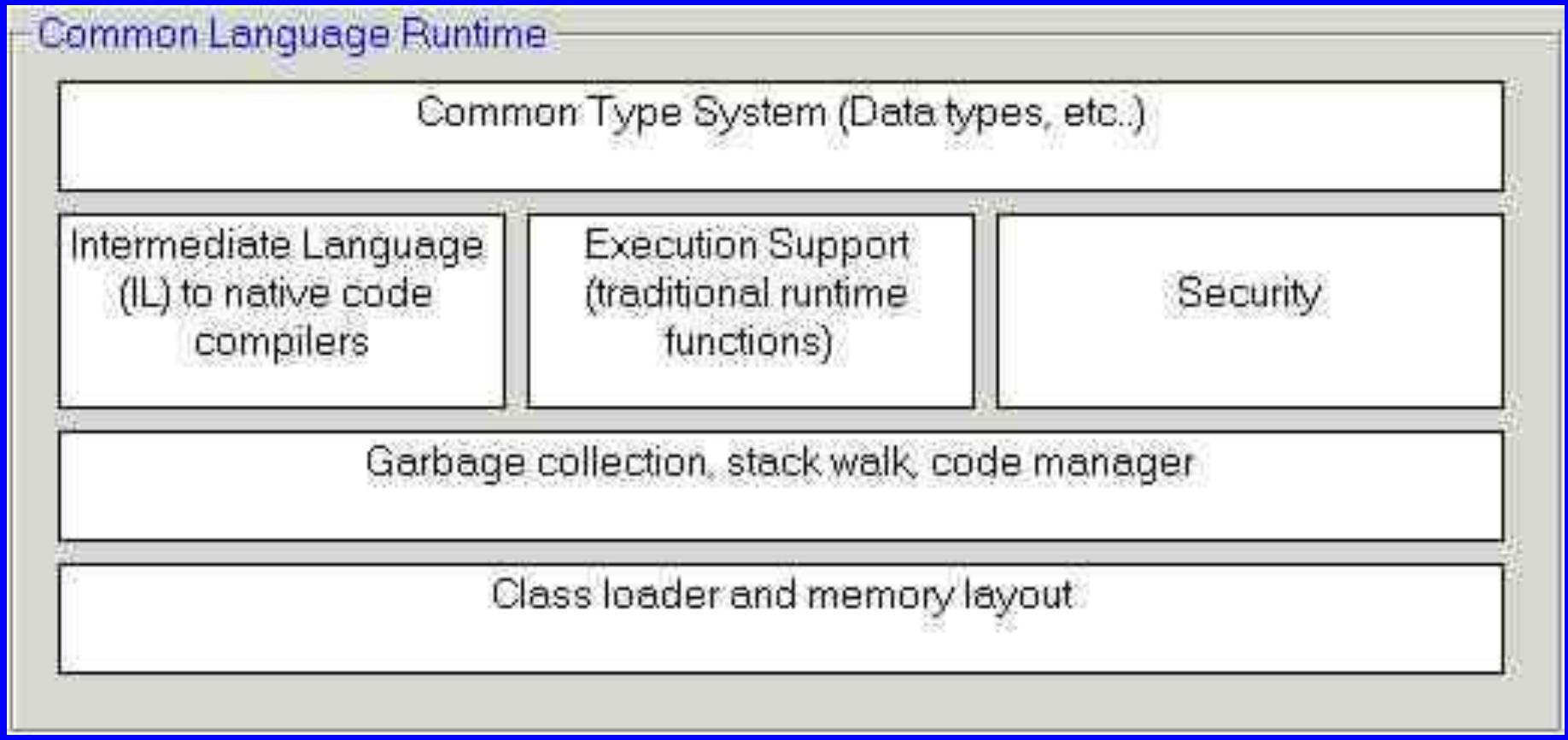
khai.hg@ou.edu.vn

Giới thiệu môi trường MS Visual Studio 2005 và ngôn ngữ lập trình C#



Các bộ phận chính của Microsoft.NET Framework

Giới thiệu môi trường MS Visual Studio 2005 và ngôn ngữ lập trình C#



khai.hq@ou.edu.vn

Giới thiệu môi trường MS Visual Studio 2005 và ngôn ngữ lập trình C#

- Giới thiệu môi trường MS Visual Studio 2005
- Giớ thiệu ngôn ngữ lập trình C#
- Demo bằng máy tính
- Hướng dẫn sử dung giao diện VS.NET 2005
- Tạo ứng dụng ví dụ bằng C Sharp

VỀ NHÀ:

- **Cài đặt MS VS.NET 2005 có C Sharp**
- **Cài đặt MSDN cho C Sharp**
- **Tạo các ứng dụng đơn giản sử dụng C Sharp**

-

HẾT CHƯƠNG 1

LẬP TRÌNH GIAO DIỆN

Ngôn ngữ C#

Nội dung chính

1. Giới thiệu C#
2. Lớp (Class) và Đối tượng(Object)
3. Cấu trúc điều khiển chương trình (Control Statements)
4. Phương thức (Method)
5. Dãy (Array)

1. Giới thiệu C#

Trong phần này chúng ta sẽ được học:

- Cách viết một ứng dung C# đơn giản.
- Cách viết các câu lệnh để nhập, xuất dữ liệu.
- Cách khai báo và sử dụng kiểu dữ liệu.
- Cách lưu trữ và truy lục dữ liệu trong bộ nhớ.
- Cách sử dụng các toán tử số học.
- Thứ tự ưu tiên giữa các toán tử.
- Cách sử dụng các toán tử so sánh.
- Cách sử dụng các hộp thoại để hiển thị các thông điệp.

1. Giới thiệu C#

1.1 Giới thiệu C#

1.2 Chương trình hiển thi một dòng văn bản

1.3 Chương trình cộng số nguyên

1.4 Bộ nhớ

1.5 Toán tử số học

1.6 Toán tử so sánh

1.1 Giới thiệu C#

- Console applications
  - Không có các thành phần trực quan
  - Chỉ kết xuất dưới dạng văn bản
  - Có 2 loại:
    - MS-DOS prompt: Windows 95/98/ME
    - Command prompt: Windows 2000/NT/XP
- Windows applications
  - Các biểu mẫu (Forms) với nhiều loại phần tử nhập
  - Chứa giao diện người dùng đồ họa (Graphical User Interfaces - GUIs)

1.1 Giới thiệu C# (tt)

- Chú thích (Comments)
  - Chú thích đơn: //…
  - Chú thích có nhiều dòng: /* … */
  - Chú thích được lờ đi khi biên dịch
  - Chỉ dùng khi người đọc mã nguồn chương trình
- Namespaces
  - Là các nhóm đặc trưng có liên quan với nhau của C# trong một phân loại
  - Cho phép sử dụng mã nguồn lại dễ dàng
  - Có rất nhiều Namespaces trong thư viện .NET framework
  - Namespace cung cấp cho ta cách mà chúng ta tổ chức quan hệ giữa các lớp và các kiểu khác
- Khoảng trắng (White Space)
  - Bao gồm: spaces, newline characters và tabs

1.1 Giới thiệu C# (tt)

- Các từ khóa (Keywords)
  - Là các từ không được phép sử dụng để khai báo biến, tên lớp
  - Có chức năng đặc biệt không thay đổi trong ngôn ngữ C#. Ví dụ: **`class`**
  - Tất cả các từ khóa đều ở dạng ký tự thường
- Các lớp (Classes)
  - Tên lớp chỉ bao gồm một từ
  - Tên lớp được viết hoa ở ký tự đầu tiên.
    Ví dụ: MyFirstProgram
  - Mỗi tên lớp là một từ định danh
    - Có thể chứ ký tự (letters), ký số (digits), và underscores (_)
    - Không được bắt đầu bằng ký số
    - Có thể bắt đầu bằng ký tự @

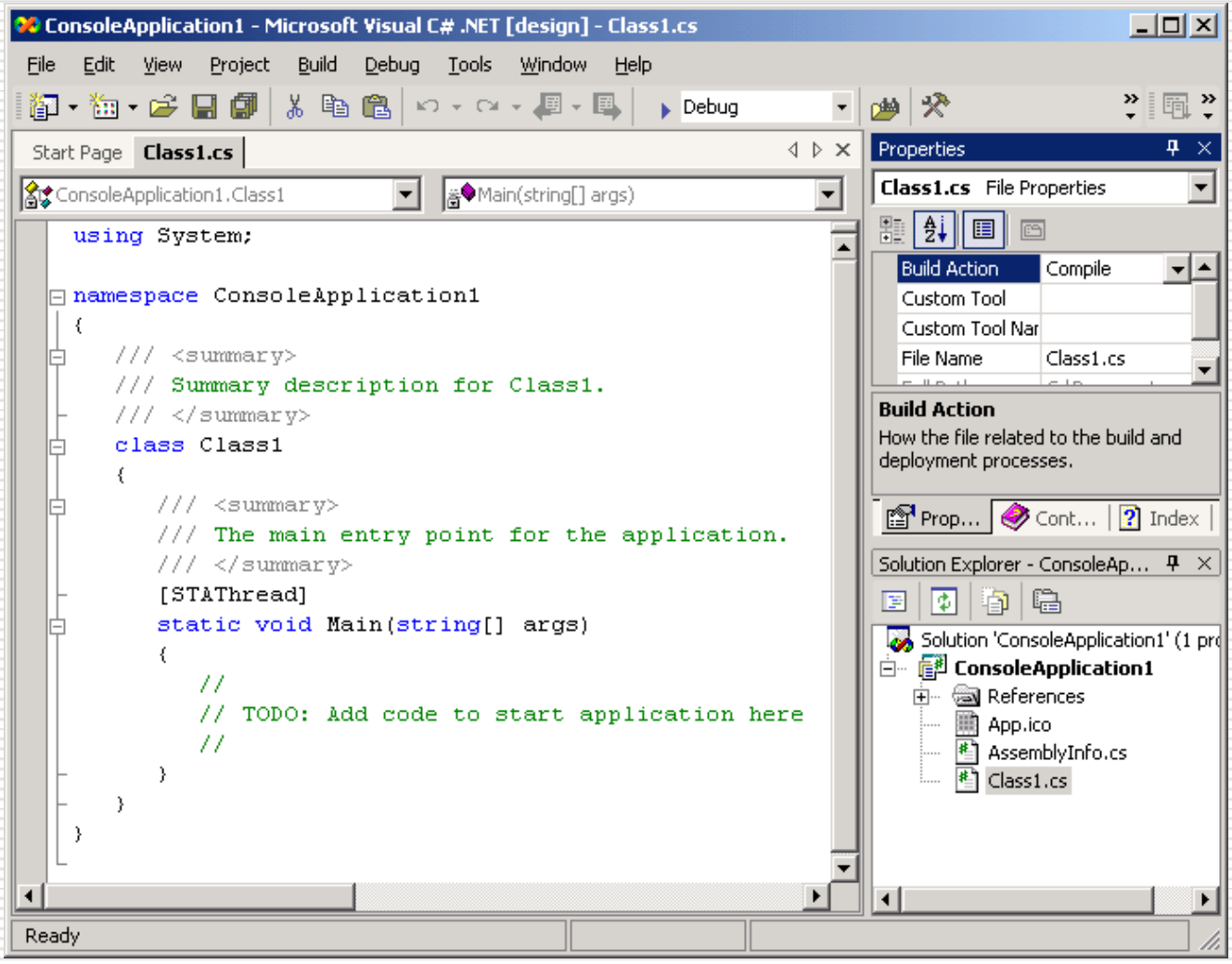
1.1 Giới thiệu C# (tt)

- Phần thân của lớp bắt đầu bằng ký tự { và kết thúc bằng ký tự }
- Phương thức (Methods)
  - Xây dựng các khối chương trình (blocks of programs)
  - Phương thức **`Main`**
    - Mỗi console hoặc windows application đều phải có
    - Tất cả các chương trình bắt đầu bằng cách thực thi phương thức **`Main`**
  - Phần thân của phương thức bắt đầu bằng ký tự { và kết thúc bằng ký tự }
- Các câu lệnh (Statements)
  - Tất cả mọi thứ đặt trong (“”) được xem là một chuỗi ký tự
  - Mỗi câu lệnh phải được kết thúc bằng dấu chấm phẩy (;)

1.1 Giới thiệu C# (tt)

- Giao diện người dùng đồ họa (GUI - **G**raphical **U**ser **I**nterface)
  - GUIs được dùng để lấy dữ liệu từ phía người dùng cũng như hiển thị dữ liệu dễ dàng hơn
  - Các hộp thông điệp (Message boxes)
    - Nằm trong **`System.Windows.Forms`** namespace
    - Dùng để nhập hoặc hiển thị thông tin

1.1 Giới thiệu C# (tt)



1.2 Chương trình hiển thi một dòng văn bản

Ví dụ:

fig3.01, fig3.14, fig3.15, fig3.17

Console.Write và Console.WriteLine

- Chúng ta có hai phương thức dùng để viết ra chuỗi ký tự như sau
  - Console.Write() - Viết một giá trị ra cửa sổ window
  - Console.WriteLine() - tương tự trên nhưng sẽ tự động xuống hàng khi kết thúc lệnh
- Thí dụ sau sẽ cho giá trị nhập kiểu int và giá trị in ra kiểu chuỗi

```
int x = Console.Read();
Console.WriteLine((char)x);
```

Giá trị trả về kiểu string:

```
string s = Console.ReadLine();
Console.WriteLine(s);
```

Một số Escape sequence

Escape sequence	Description
`\n`	Newline. Position the screen cursor to the beginning of the next line.
`\t`	Horizontal tab. Move the screen cursor to the next tab stop.
`\r`	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line. Any characters output after the carriage return overwrite the previous characters output on that line.
`\\`	Backslash. Used to print a backslash character.
`\"`	Double quote. Used to print a double quote (") character.

Kiểu dữ liệu

- Các kiểu dữ liệu nguyên thủy (Primitive data types)
  - Là các kiểu dữ liệu được xây dựng sẵn trong C#
    - String, Int, Double, Char, Long…
    - Có 15 kiểu dữ liệu nguyên thủy
  - Tên mỗi kiểu dữ liệu là một từ khóa trong C#
  - Có thể khai báo các biến có cùng kiểu dữ liệu trên cùng một hàng hoặc nhiều hàng khác
- Console.ReadLine()
  - Dùng để đọc một ký tự văn bản từ của sổ console, giá trị trả về sẽ là kiểu int hoặc kiểu string tuỳ ý.
- Int32.Parse()
  - Dùng để chuyển từ chuỗi sang số

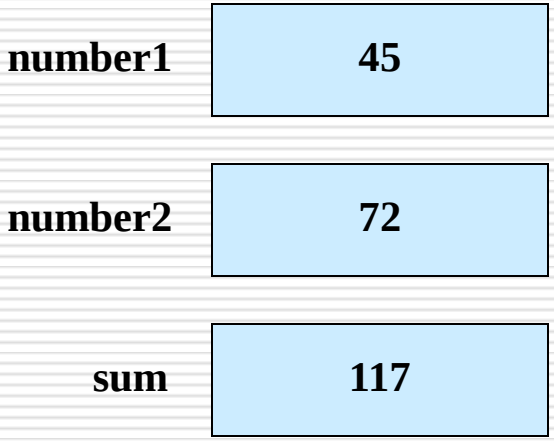
1.3 Chương trình cộng số nguyên

Ví dụ: exercise03_06, fig3.18

1.4 Bộ nhớ

- Vùng nhớ
  - Mỗi biến được chứa trong một vùng nhớ
    - Bao gồm tên (name), loại (type), kính thước (size) và giá trị (value)
  - Khi có giá trị mới được nhập thì giá trị cũ bị mất đi
  - Sử dụng biến để duy trì dữ liệu sau khi sử dụng

1.4 Bộ nhớ (tt)



1.5 Toán tử số học

- Toán tử số học (Arithmetic operations)
  - Nhân: (*), Chia: (/)
  - Modulus: (%)
  - Cộng (+), Trừ: (-)
  - Phải được viết trên cùng một dòng
  - Không có số mũ
- Phép chia
  - Kết quả trả về của phép chia tùy thuộc vào kiểu dữ liệu của biến:
    - Khi chia hai số nguyên thì kết quả luôn là một số nguyên được làm tròn
    - Để có kết quả chính xác cần sử dụng các biến có hỗ trợ số thập phân

1.5 Toán tử số học (tt)

- Thứ tự ưu tiên
  - Trong dấu ngoặc đơn thực hiện trước
  - Chia, nhân và modulus được thực hiện kế tiếp
    - Từ trái sang phải
  - Cộng trừ thực hiện sau cùng
    - Từ trái sang phải

1.5 Toán tử số học (tt)

Toán tử	Ký hiệu	Biểu thức đại số	C#
Cộng	+	*f + 7*	*f + 7*
Trừ	–	*p – c*	*p - c*
Nhân	*	*bm*	*b * m*
Chia	/	x / y	*x / y*
Modulus	%	r *mod* s	*r % s*

1.5 Toán tử số học (tt)

Bước 1.
```
y = 2 * 5 * 5 + 3 * 5 + 7;
```
2 * 5 là **10**

Bước 2.
```
y = 10 * 5 + 3 * 5 + 7;
```
10 * 5 là **50**

Bước 3.
```
y = 50 + 3 * 5 + 7;
```
3 * 5 là **15**

Bước 4.
```
y = 50 + 15 + 7;
```
50 + 15 là **65**

Bước 5.
```
y = 65 + 7;
```
65 + 7 là **72**

Bước 6.
```
y = 72;
```

1.6 Toán tử so sánh (tt)

Toán tử	Ký hiệu	C#	Ý nghĩa
=	==	**x == y**	**x** bằng **y**
≠	!=	**x != y**	**x** khác **y**
>	>	**x > y**	**x** lớn hơn **y**
<	<	**x < y**	**x** nhỏ hơn **y**
≥	>=	**x >= y**	**x** lớn hơn hoặc bằng **y**
≤	<=	**x <= y**	**x** lớn hơn hoặc bằng **y**

Ví dụ: fig3.26

Câu hỏi

- ?
- ?
- ?

2. Lớp (Class) và Đối tượng(Object)

Trong phần này chúng ta sẽ được học:

- Khái niệm về Lớp (class), Đối tượng (object), Phương thức (method) và Biến thực thể (instance variable).
- Cách khai báo một lớp và sử dụng nó để tạo Đối tượng
- Cách hiện thực các hành vi của Lớp (các phương thức).
- Cách hiện thực các tính chất của Lớp (biến thực thể và thuộc tính).
- Cách gọi các phương thức của một đối tượng.
- Sự khác biệt giữa biến thực thể của một lớp và biến cục bộ của một phương thức.
- Cách sử dụng phương thức xây dựng lớp (Constructor).
- Sự khác biệt giữa kiểu giá trị (value type) và kiểu tham chiếu (reference type).

2. Lớp (Class) và Đối tượng(Object)

2.1 Lớp (Class), Đối tượng (Object), Phương thức (Method) và Biến thực thể (Instance variable)

2.2 Khai báo một lớp và khởi tạo đối tượng

2.3 Khai báo một phương thức với thông số (Parameter)

2.4 Biến thực thể (Instance Variables) và thuộc tính (Properties)

2.5 Khác biệt giữa kiểu giá trị (value type) và kiểu tham chiếu (reference type)

2.6 Khởi tạo Đối tượng bằng phương thức xây dựng lớp (Constructor)

2.7 Số Floating-Point và Decimal

2.1 Lớp (Class), Đối tượng (Object), Phương thức (Method) và Biến thực thể (Instance variable)

- Objects: đóng gói tính chất và hành vi của các đối tượng trong tự nhiên
- Class: tập hợp các đối tượng giống nhau
- Object có thể che giấu thông tin
- Methods: các khối trong chương trình
- User-defined type: class được viết bởi lập trình viên
- Class chứa:
  - Data members: member variable hoặc instance variables
  - Methods: thao tác trên data members

2.1 Lớp (Class), Đối tượng (Object), Phương thức (Method) và Biến thực thể (Instance variable)

- Abstract Data Types – che giấu hiện thực từ những Object khác
- Các biến (variables) được định nghĩa bên trong Class nhưng không phải là Method được gọi là biến thực thể (*instance variables)*
- *Member Access Modifiers*
  - *public*: thành phần được truy xuất bất cứ nơi nào trong thực thể của đối tượng hiện có
  - *private :* thành phần chỉ được truy xuất bên trong định nghĩa Class

2.1 Lớp (Class), Đối tượng (Object), Phương thức (Method) và Biến thực thể (Instance variable)

- Access methods: đọc hay cập nhật dữ liệu
- Predicate methods: kiểm tra các điều kiện
- Phương thức xây dựng lớp (Constructor)
  - Xây dựng Object của Class
  - Có thể có đối số
  - Không trả về giá trị
  - Có thể có nhiều constructor trong một class
- Toán tử **new** được sử dụng để khởi tạo thực thể của Class
- **Project < Add Class**: thêm một Class mới vào Project

2.2 Khai báo một lớp và khởi tạo đối tượng

Ví dụ: *Fig. 4.1*

```
1  // Fig. 4.1: GradeBook.cs
2  // Class declaration with one method.
3  using System;
4
5  public class GradeBook
6  {
7     // display a welcome message to the GradeBook user
8     public void DisplayMessage()
9     {
10       Console.WriteLine( "Welcome to the Grade Book!" );
11    } // end method DisplayMessage
12 } // end class GradeBook
```

2.3 Khai báo một phương thức với thông số (Parameter)

Ví dụ: *Fig. 4.4*

```
 1 // Fig. 4.4: GradeBook.cs
 2 // Class declaration with a method that has a parameter.
 3 using System;
 4
 5 public class GradeBook
 6 {
 7    // display a welcome message to the GradeBook user
 8    public void DisplayMessage( string courseName )
 9    {
10       Console.WriteLine( "Welcome to the grade book for\n{0}!",
11          courseName );
12    } // end method DisplayMessage
13 } // end class GradeBook
```

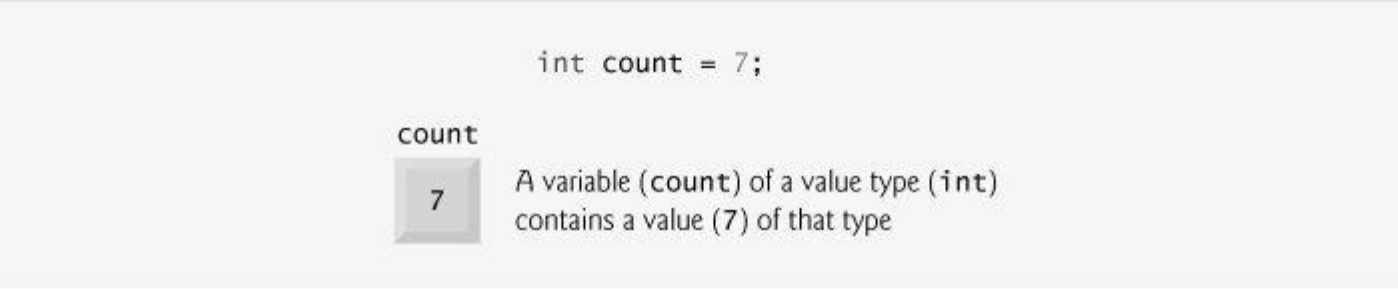
2.4 Biến thực thể (Instance Variables) và thuộc tính (Properties)

Ví dụ: Fig. 4.7

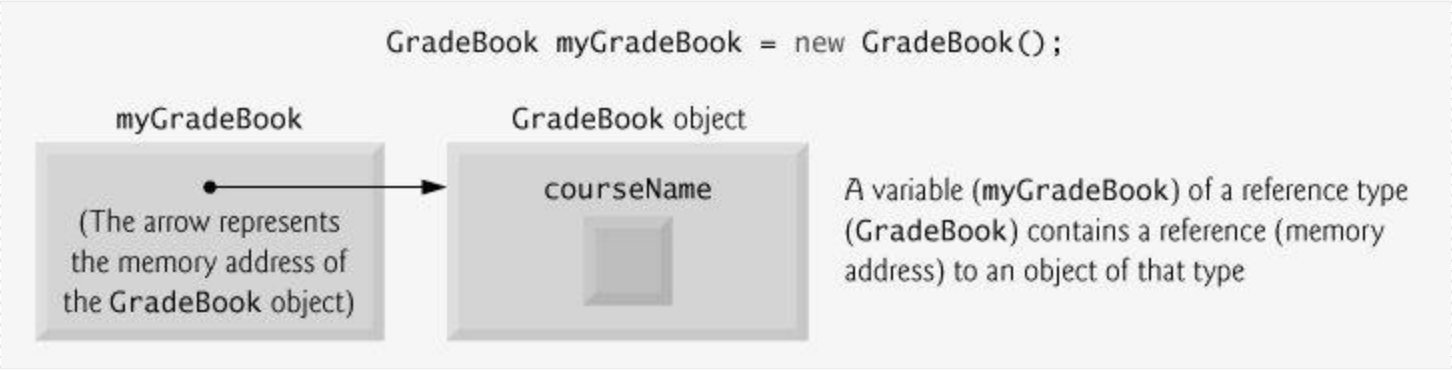
```
 8   private string courseName; // course name for this GradeBook
 9
10   // property to get and set the course name
11   public string CourseName
12   {
13      get
14      {
15         return courseName;
16      } // end get
17      set
18      {
19         courseName = value;
20      } // end set
21   } // end property CourseName
```

2.5 Khác biệt giữa kiểu giá trị (value type) và kiểu tham chiếu (reference type)

- kiểu giá trị (value type)



- kiểu tham chiếu (reference type)



2.6 Khởi tạo Đối tượng bằng phương thức xây dựng lớp (Constructor)

Ví dụ:

Fig. 4.12 GradeBook.cs

```
 9   // constructor initializes courseName with string supplied as argument
10    public GradeBook( string name )
11    {
12       CourseName = name; // initialize courseName using property
13    } // end constructor
```

Fig. 4.13: GradeBookTest.cs

```
 6 public class GradeBookTest
 7 {
 8   // Main method begins program execution
 9   public static void Main( string[] args )
10    {
11       // create GradeBook object
12       GradeBook gradeBook1 = new GradeBook( // invokes constructor
13          "CS101 Introduction to C# Programming" );
14       GradeBook gradeBook2 = new GradeBook( // invokes constructor
15     "CS102 Data Structures in C#" );
        …
```

2.7 Số Floating-Point và Decimal

- Kiểu ***float*** và ***double*** được gọi là kiểu dấu chấm động (floating-point).
- Điểm khác nhau chính giữa chúng và ***Decimal*** là ***Decimal*** lưu trữ một số thực giới hạn chính xác, trong khi **floating-point** lưu trữ một số thực giới hạn xấp xỉ nhưng có tầm giá trị lớn hơn.

Ví dụ: Fig. 4.15, Fig. 4.16

Câu hỏi

- ?
- ?
- ?

3. Cấu trúc điều khiển chương trình (Control Statements)

Trong phần này chúng ta sẽ:

- Sử dụng kỹ thuật giải quyết vấn đề.
- Phát triển giải thuật theo quy trình từ trên xuống và cải tiến từng bước.
- Sử dụng câu lệnh **if** và **if...else** để lựa chọn.
- Sử dụng câu lệnh **while** để thực thi các câu lệnh trong ứng dụng có sự lặp đi lặp lại nhiều lần.
- Sử dụng lặp theo biến đếm (counter-controlled) và lặp theo phần tử cầm canh (sentinel-controlled).
- Sử dụng các toán tử tăng, giảm, gán.
- Hiểu được cơ sở của việc lặp theo biến đếm.
- Sử dụng câu lệnh **for** và **do...while** để thực thi các câu lệnh trong ứng dụng có sự lặp đi lặp lại nhiều lần.
- Sử dụng câu lệnh **switch** trong đa lựa chọn.
- Sử dụng câu lệnh **break** và **continue** trong điều khiển luồng chương trình.
- Sử dụng các toán tử luận lý để thiết lập các biểu thức điều kiện phức tạp.

3. Cấu trúc điều khiển chương trình (Control Statements)

3.1 Giải thuật (Algorithms)
3.2 Mã giả (Pseudocode)
3.3 Cấu trúc điều khiển
3.4 Câu lệnh **if**
3.5 Câu lệnh **if...else**
3.6 Câu lệnh **while**
3.7 Lặp theo biến đếm (counter-controlled)
3.8 Lặp theo phần tử cầm canh (sentinel-controlled)
3.9 Các câu lệnh điều khiển lồng nhau (Nested Control Statements)
3.10 Các toán tử gán
3.11 Các toán tử tăng giảm
3.12 Các kiểu dữ liệu cơ bản
3.13 Cơ sở của việc lặp theo biến đếm (counter-controlled)
3.14 Câu lệnh **for**
3.15 Câu lệnh **do...while**
3.16 Câu lệnh **switch**
3.17 Câu lệnh **break** và **continue**
3.18 Các toán tử luận lý
3.19 Tóm tắt lập trình cấu trúc

3.1 Giải thuật (Algorithms)

- Thủ tục (Procedure)
  - Các hành vi mà một chương trình sẽ thực hiện
  - Thứ tự thực hiện các hành vi đó
  - Còn được gọi là giải thuật
- Điều khiển chương trình (Program control)
  - Thứ tự các công việc để một thủ tục thực hiện đúng.

3.2 Mã giả (Pseudocode)

- Là dạng ngôn ngữ nhân tạo (Artificial) và không hình thức (informal)
- Giúp các lập trình viên phát thảo giải thuật dễ dàng
- Giống như ngôn ngữ tiếng Anh
- Không phải là một ngôn ngữ lập trình thật sự
- Chuyển đổi sang mã nguồn C# một cách đơn giản

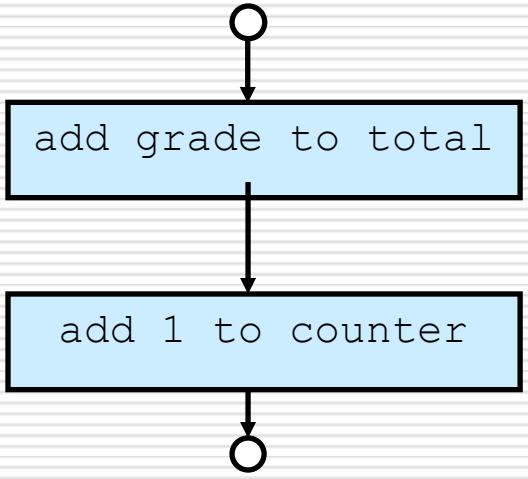
3.3 Cấu trúc điều khiển

- Chương trình điều khiển (Program of control)
  - Chương trình thực hiện một câu lệnh, sau đó chuyển đến dòng kế tiếp
    - Thực thi tuần tự
  - Chương trình thực hiện câu lệnh khác
    - Cấu trúc lựa chọn
      - Câu lệnh **`if`** và **`if/else`**
      - Câu lệnh **`goto`** (hiếm khi sử dụng)
    - Cấu trúc lặp
      - Câu lệnh lặp **`while`** và **`do/while`**
      - Câu lệnh **`for`** và **`foreach`**

3.3 Cấu trúc điều khiển

- Lưu đồ (Flow charts)
  - Được dùng để ánh xạ chương trình
  - Minh họa thứ tự các sự kiện
    - Hình chử nhật: biểu diễn một hành động
    - Hình Oval: biểu diễn điểm bắt đầu
    - Hình tròn: biểu diễn một bộ nối
    - Hình thoi: biểu diễn một điều kiện
  - Sự kết hợp giữa các cấu trúc điều khiển
    - Ngăn xếp (Stacking)
      - Đặt một cái sau một cái khác
    - Lồng nhau (Nesting)
      - Chèn một cấu trúc điều khiển sau một cấu trúc điều khiển khác

3.3 Cấu trúc điều khiển

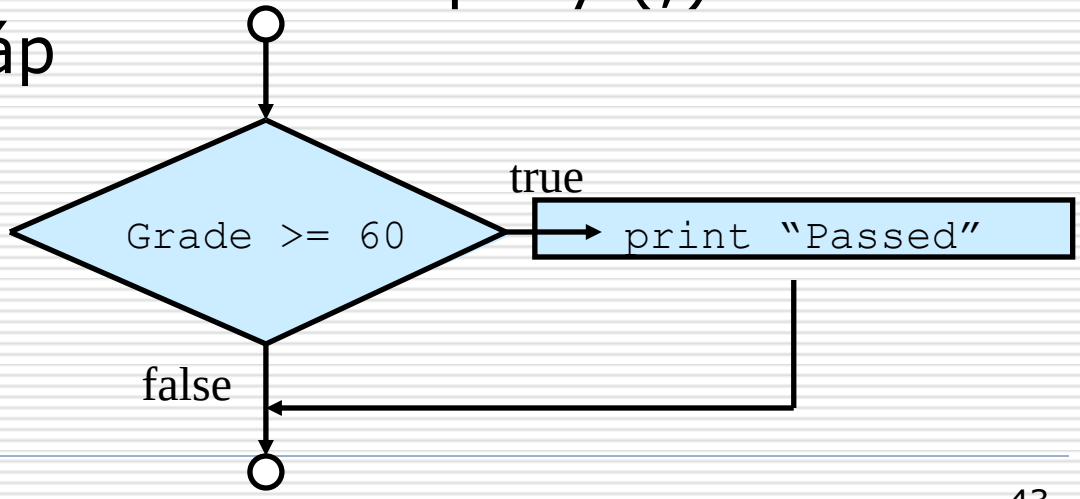



```
total = total + grade;
```

```
counter = counter + 1;
```

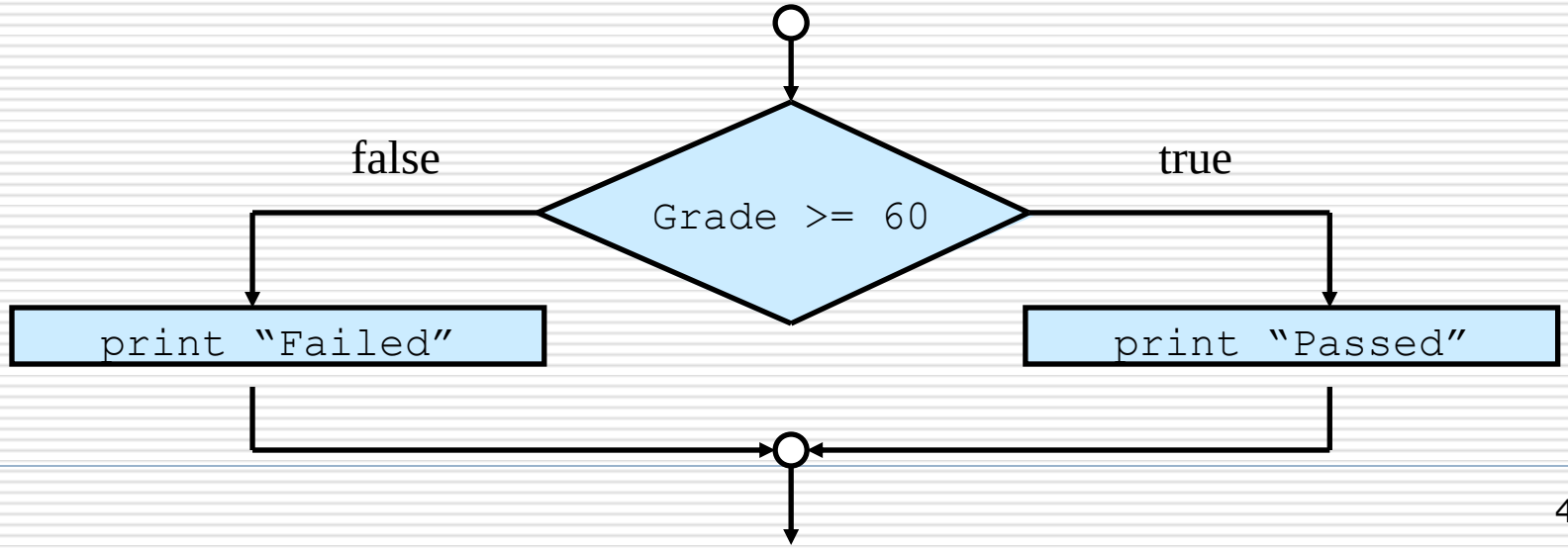
3.4 Câu lệnh **if**

- Giúp chương trình thực hiện lựa chọn
- Lựa chọn dựa trên điều khiện
  - Các biểu thức được đánh giá theo kiểu **`bool`**
  - Đúng (True): thực hiện một hành động
  - Sai (False): bỏ qua hành động
- Nhập/xuất đơn
- Không yêu cầu dấu chấm phẩy (;) trong cú pháp



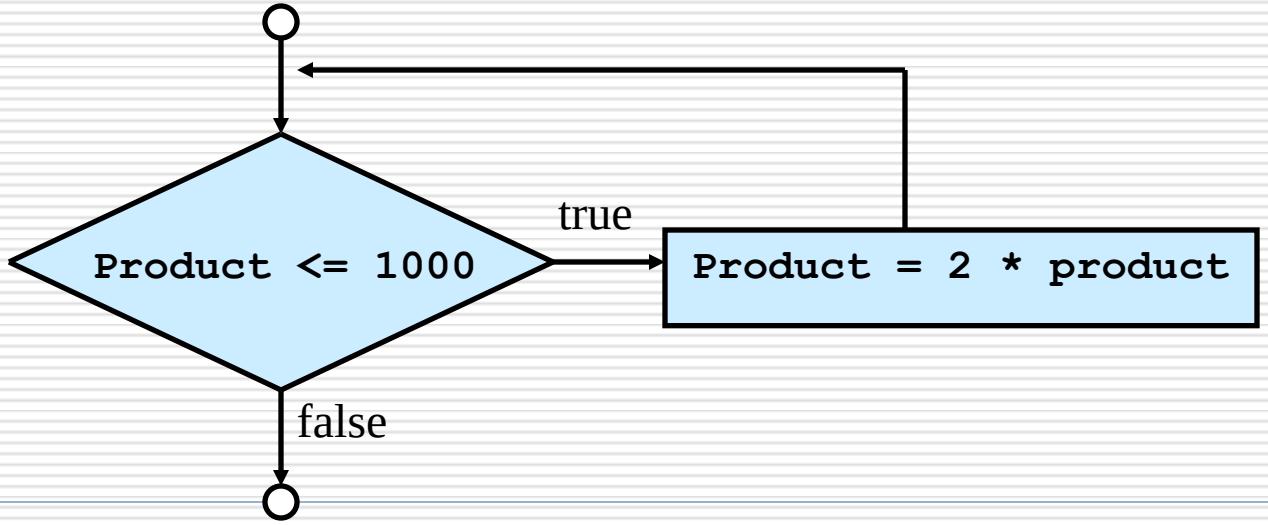
3.5 Câu lệnh **if...else**

- Tiến trình thay phiên nhau diễn ra khi câu lệnh sai
- Thích hợp hơn với một hành động có hai lựa chọn
- Cấu trúc lồng nhau có thể kiểm tra nhiều trường hợp
- Các cấu trúc có nhiều dòng lệnh phải đặt trong ({ })
  - Có thể là nguyên nhân gây ra lỗi
    - Lỗi thiên về luận lý (Fatal logic error)
    - Lỗi không thiên về luận lý (Nonfatal logic error)



3.6 Câu lệnh **while**

- Thực hiện một hành động lặp đi lặp lại
  - Tiếp tục thực hiện khi câu lệnh **while** đúng
  - Kết thúc khi khi câu lệnh **while** sai
- Có thể chứa một hay nhiều dòng lệnh
  - Phải thay đổi điều kiện
    - Lặp vô tận (Endless loop)



3.7 Lặp theo biến đếm (counter-controlled)

- Lặp theo biến đếm (Counter Controlled)
  - Được sử dụng để nhập dữ liệu cho mỗi lần lặp
    - Số lượng không thay đổi
  - Một biến đếm (counter) được dùng để xác định số lần lặp
  - Khi biến đếm đạt đến một giá trị xác định, câu lệnh kết thúc
- Mã giả

```
Set total to zero
Set grade counter to one

While grade counter is less than or equal to ten
        Input the next grade
        Add the grade into the total
        Add one to the grade counter

Set the class average to the total divided by ten
Print the class average
```

3.8 Lặp theo phần tử cầm canh (sentinel-controlled)

- Lặp theo phần tử cầm canh (Sentinel controlled)
  - Không bị bó buộc bởi số lần lặp
  - Giá trị cầm canh (Sentinel value)
    - Sử dụng để ngừng vòng lặp
    - Tránh các xung đột
      - *When flag value = user entered value*
- Chuyển đổi (Casting)
  - Cho phép một biến tạm thời sử dụng như một biến khác

3.8 Lặp theo phần tử cầm canh (sentinel-controlled)

Initialize total to zero
Initialize counter to zero

Input the first grade (possibly the sentinel)

While the user has not as yet entered the sentinel
    Add this grade into the running total
    Add one to the grade counter
    Input the next grade (possibly the sentinel)

If the counter is not equal to zero
    Set the average to the total divided by the counter
    Print the average
Else
    Print "No grades were entered"

3.9 Các câu lệnh điều khiển lồng nhau (Nested Control Statements)

- Là việc chèn một cấu trúc điều khiển vào bên trong một cấu trúc điều khiển khác
  - Nhiều vòng lặp (Multiple loops)
  - Thực hiện lặp bằng câu lệnh **`if`**

```
Initialize passes to zero
Initialize failures to zero
Initialize student to one

While student counter is less than or equal to ten
        Input the next exam result

        If the student passed
                Add one to passes
        Else
                Add one to failures

        Add one to student counter

Print the number of passes
Print the number of failures

If more than eight students passed
        Print "Raise tuition"
```

3.10 Các toán tử gán

- Có thể rút ngắn mã chương trình
  - x += 2 tương đương với x = x + 2
- Có thể thực hiện với tất cả các toán tử số học:
  - ++, -=, *=, /=, %=

Toán tử gán	Ví dụ	Diễn giải	Gán
Giả sử: `int c = 3, d = 5, e = 4, f = 6, g = 12;`			
`+=`	`c += 7`	`c = c + 7`	10 cho `c`
`-=`	`d -= 4`	`d = d - 4`	1 cho `d`
`*=`	`e *= 5`	`e = e * 5`	20 cho `e`
`/=`	`f /= 3`	`f = f / 3`	2 cho `f`
`%=`	`g %= 9`	`g = g % 9`	3 cho `g`

3.11 Các toán tử tăng giảm

- Toán tử tăng (Increment operator)
  - Sử dụng để tăng giá trị thêm 1 đơn vị
  - x++
  - Giống như x = x + 1
- Toán tử giảm (Decrement operator)
  - Sử dụng để giảm giá trị 1 đơn vị
  - y--
- Khác nhau giữa Pre-increment và post-increment
  - x++ hoặc x--
    - Thực hiện hành động trước, sau đó mới thêm hoặc bớt đi 1 đơn vị
  - ++x hoặc --x
    - Thêm hoặc bớt đi 1 đơn vị trước, sau đó mới thực hiện hành động

3.12 Các kiểu dữ liệu cơ bản

Name	CTS Type	Description	Range (min:max)
sbyte	System.SByte	8-bit signed integer	-128:127 (-2^7:2^7-1)
short	System.Int16	16-bit signed integer	-32,768:32,767 (-2^{15}:2^{15}-1)
int	System.Int32	32-bit signed integer	-2,147,483,648:2,147,483,647 (-2^{31}:2^{31}-1)
long	System.Int64	64-bit signed integer	-9,223,372,036,854,775,808: 9,223,372,036,854,775,807 (-2^{63}:2^{63}-1)
byte	System.Byte	8-bit signed integer	0:255 (0:2^8-1)
ushort	System.UInt16	16-bit signed integer	0:65,535 (0:2^{16}-1)
uint	System.UInt32	32-bit signed integer	0:4,294,967,295 (0:2^{32}-1)
ulong	System.UInt64	64-bit signed integer	0:18,446,744,073,709,551,615 (0:2^{64}-1)

Các kiểu Integer

3.12 Các kiểu dữ liệu cơ bản

Kiểu dữ liệu số dấu chấm di động (Floating Point Types)

Name	CTS Type	Description	Significant Figures	Range (approximate)
Float	System.Single	32-bit single-precision floating- point	7	$\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$
Double	System.Double	64-bit double-precision floating- point	15/16	$\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$

Kiểu dữ liệu số thập phân (Decimal Type):

Name	CTS Type	Description	Significant Figures	Range (approximate)
decimal	System.Decimal	128-bit high precision decimal notation	28	$\pm 1.0 \times 10^{-28}$ to $\pm 7.9 \times 10^{28}$

3.12 Các kiểu dữ liệu cơ bản

Kiểu Boolean :

Name	CTS Type	Value
Bool	System.Boolean	true or false

Kiểu Character Type:

Name	CTS Type	Value
char	System.Char	Represents a single 16-bit (Unicode)character

Kiểu tham khảo tiền định nghĩa:

Name	CTS Type	Description
object	System.Object	The root type, from which all other types in the CTS derive (including value types)
string	System.String	Unicode character string

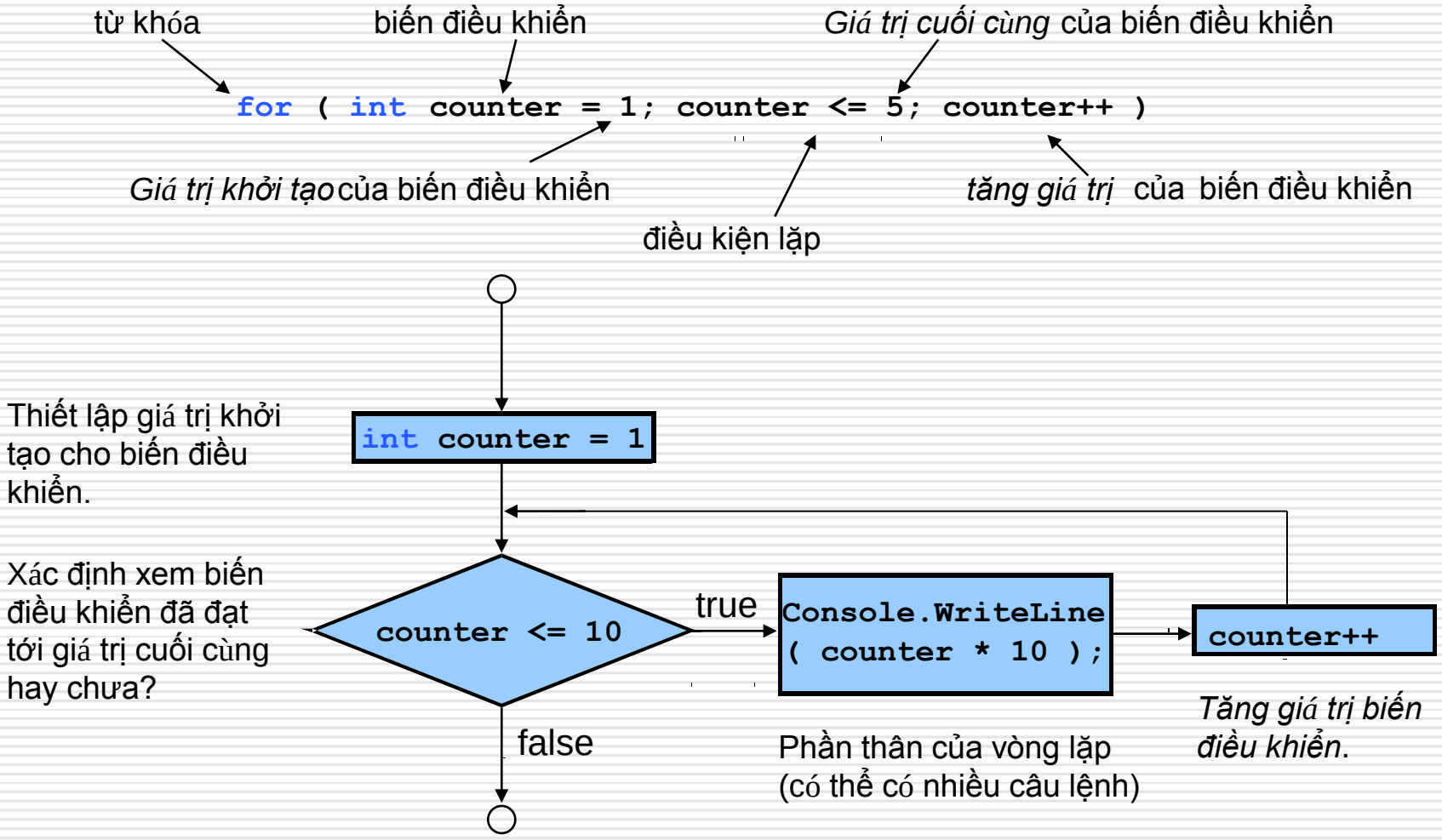
3.13 Cơ sở của việc lặp theo biến đếm (counter-controlled)

- Biến điều khiển
  - Là biến dùng để xác định nếu việc lặp được tiếp diễn
- Giá trị khởi tạo của biến điều khiển
- Sự tăng/giảm giá trị của biến điều khiển
- Điều kiện
  - Khi nào việc lặp lại được tiếp diễn

3.14 Câu lệnh **for**

- Cú pháp: **`for`** (Expression1, Expression2, Expression3)
  - Expression1 = tên của biến điều khiển
    - Có thể có nhiều biến
  - Expression2 = điều kiện để việc lặp tiếp diễn
  - Expression3 = tăng/giảm giá trị
    - Nếu Expression1 có nhiều biến, Expression3 phải có nhiều biến tương ứng
    - ++counter và counter++ là tương đương nhau
- Phạm vi của biến
  - Expression1 chỉ được sử dụng trong thân của vòng lặp **`for`**
  - Khi vòng lặp kết thúc thì biến được giải phóng

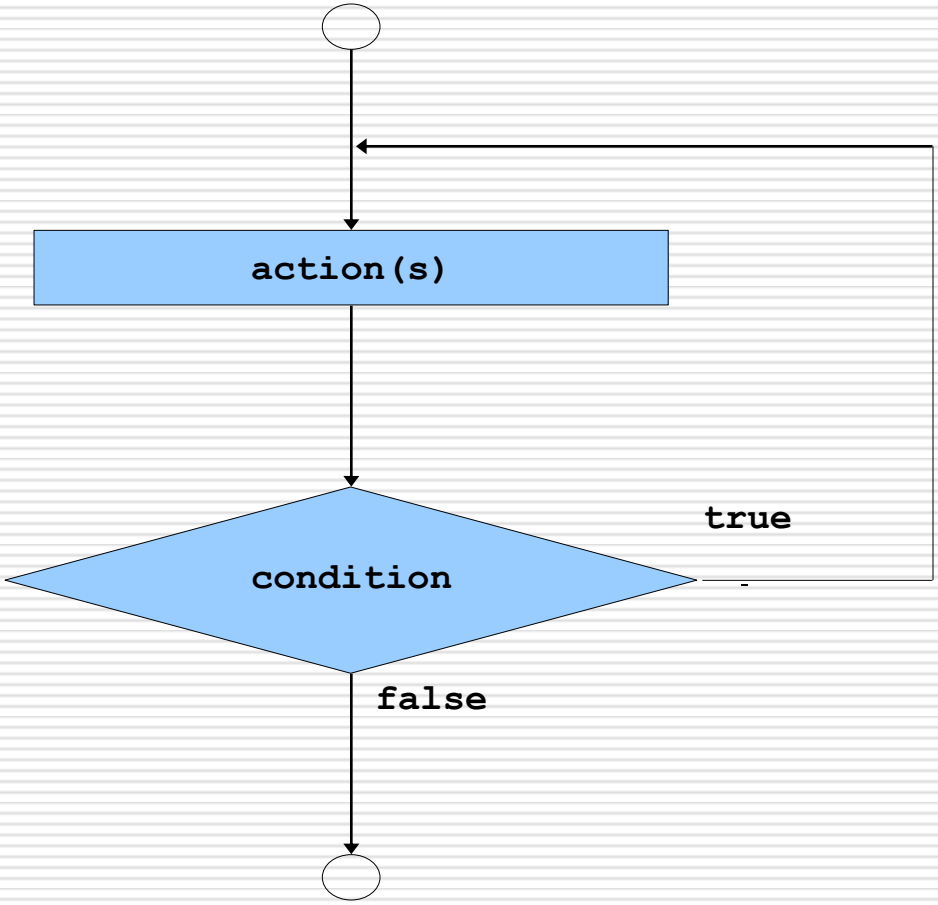
3.14 Câu lệnh **for**



3.15 Câu lệnh **do...while**

- Sử dụng vòng lặp **`while`**
  - Điều kiện được kiểm tra trước
  - Hành động được thực hiện sau
  - Việc lặp có thể bị bỏ qua
- Sử dụng vòng lặp **`do/while`**
  - Hành động được thực hiện trước
  - Sau đó kiểm tra điều kiện
  - Việc lặp diễn ra ít nhất là một lần
  - Luôn sử dụng ({ }) để tránh nhầm lẫn

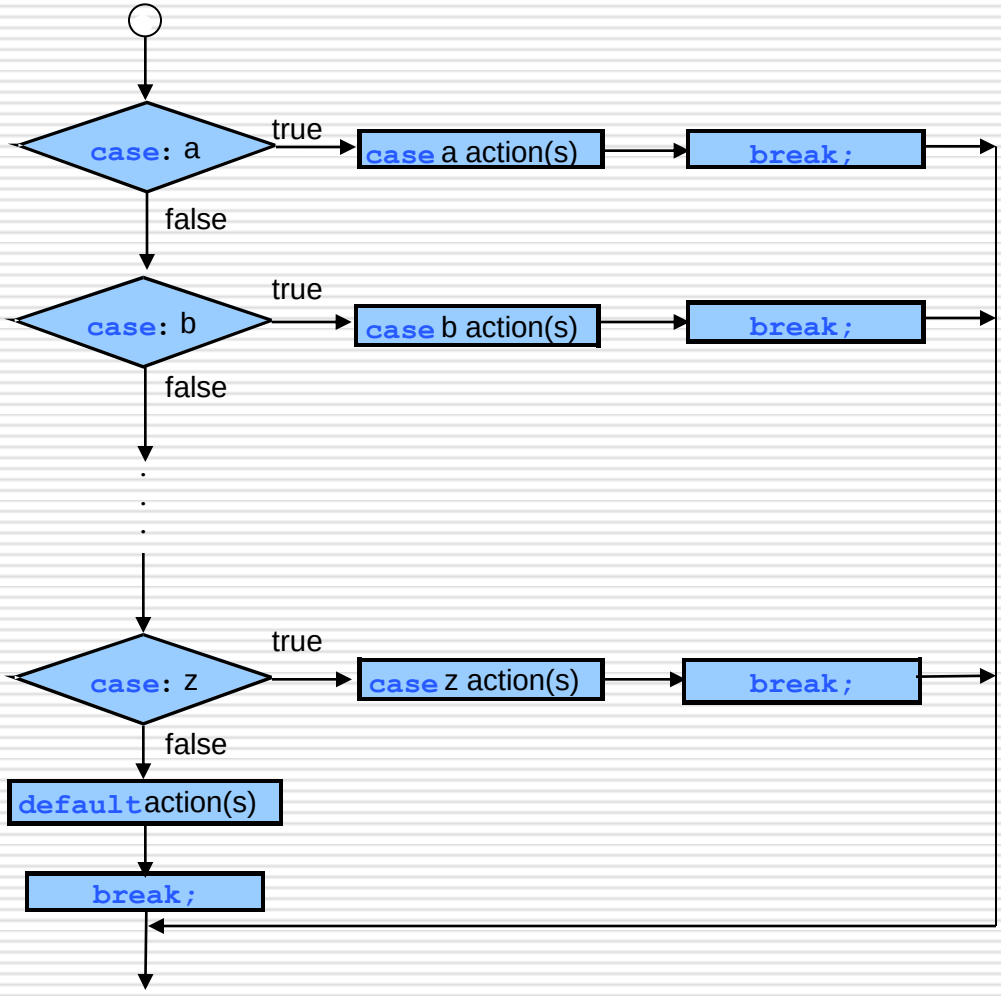
3.15 Câu lệnh **do...while**



3.16 Câu lệnh **switch**

- Các biểu thức hằng số
  - Chuỗi (String)
  - Tích phân (Integral)
- Các trường hợp (Cases)
  - Case '*x*' :
    - Dùng trong trường hợp biến là hằng
  - Các trường hợp rỗng (Empty cases)
  - Trường hợp đặc biệt (default case)
- Câu lệnh **`break`**
  - Dùng để thoát ra khỏi **`switch`**

3.16 Câu lệnh **switch**



3.17 Câu lệnh **break** và **continue**

- Sử dụng để thay đổi luồng điều khiển (flow of control)
- Câu lệnh **`break`**
  - Dùng để sớm thoát ra khỏi vòng lặp
- Câu lệnh **`continue`**
  - Dùng để bỏ qua liệc thực hiện các câu lệnh còn lại và bắt đầu lặp lại ở câu lệnh đầu tiên trong vòng lặp
- Các chương trình có thể hoàn thành mà không cần sử dụng chúng

3.18 Các toán tử luận lý

- **Các toán tử luận lý**
  - Logical AND (&)
  - Conditional AND (&&)
  - Logical OR (|)
  - Conditional OR (||)
  - Logical exclusive OR or XOR (^)
  - Logical NOT (!)
- Dùng để kết hợp nhiều điều kiện vào trong một câu lệnh

3.18 Các toán tử luận lý

expression1	expression2	expression1 && expression2
false	false	false
false	true	false
true	false	false
true	true	true

Fig. 5.16 Truth table for the && (logical AND) operator.

expression1	expression2	expression1 || expression2
false	false	false
false	true	true
true	false	true
true	true	true

Fig. 5.17 Truth table for the || (logical OR) operator.

3.18 Các toán tử luận lý

expression1	expression2	expression1 ^ expression2
`false`	`false`	`false`
`false`	`true`	`true`
`true`	`false`	`true`
`true`	`true`	`false`

Fig. 5.18 Truth table for the logical exclusive OR (^) operator.

expression	!expression
`false`	`true`
`True`	`false`

Fig. 5.19 Truth table for operator ! (logical NOT).

3.19 Tóm tắt lập trình cấu trúc

- Các cấu trúc điều khiển
  - Chỉ có một đầu vào
  - Chỉ có một đầu ra
  - Xây dựng các khối của chương trình
  - Cho phép lồng nhau
  - Tạo mã rõ ràng và dễ dàng hơn
  - Các cấu trúc điều khiển không chồng chéo lên nhau
    - Từ khóa `goto`

3.19 Tóm tắt lập trình cấu trúc

- 3 dạng điều khiển thiết yếu
  - Có nhiều cách để thực hiện các điều khiển này
  - Tuần tự (Sequential) (chỉ có 1 cách)
    - Straight forward programming
  - Lựa chọn (Selection): có 3 cách
    - Lựa chọn **`if`** (có 1 lựa chọn)
    - Lựa chọn **`if/else`** (có 2 lựa chọn)
    - Câu lệnh **`switch`** (có nhiều lựa chọn)
  - Lặp (Repetition): có 4 cách
    - cấu trúc **`while`**
    - cấu trúc **`do/while`**
    - cấu trúc **`for`**
    - cấu trúc **`foreach`**

3.19 Tóm tắt lập trình cấu trúc

Operators	Associativity	Type
`()` `++ --`	left to right right to left	parentheses unary postfix
`++ -- + - ! (type)`	right to left	unary prefix
`* / %`	left to right	multiplicative
`+ -`	left to right	additive
`< <= > >=`	left to right	relational
`== !=`	left to right	equality
`&`	left to right	logical AND
`^`	left to right	logical exclusive OR
`|`	left to right	logical inclusive OR
`&&`	left to right	conditional AND
`||`	left to right	conditional OR
`?:`	right to left	conditional
`= += -= *= /= %=`	right to left	assignment

Fig. 5.21 Precedence and associativity of the operators discussed so far.

3.19 Tóm tắt lập trình cấu trúc

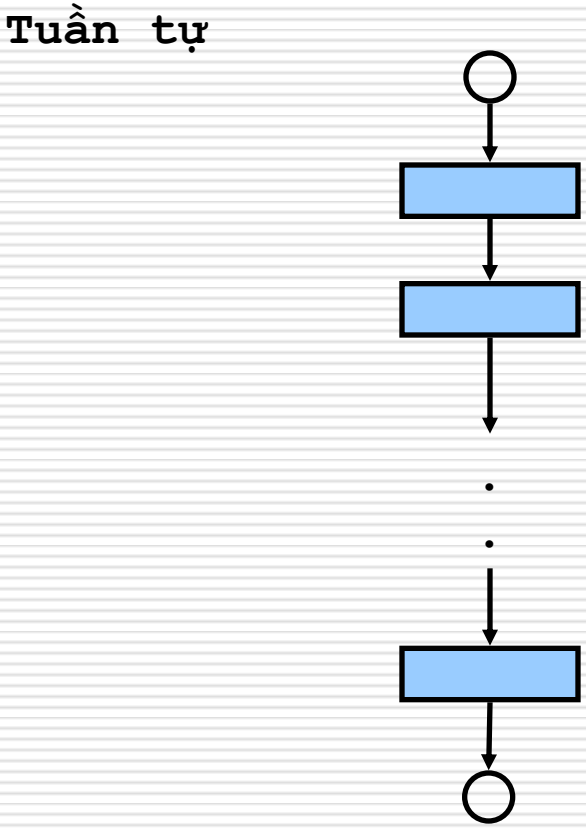
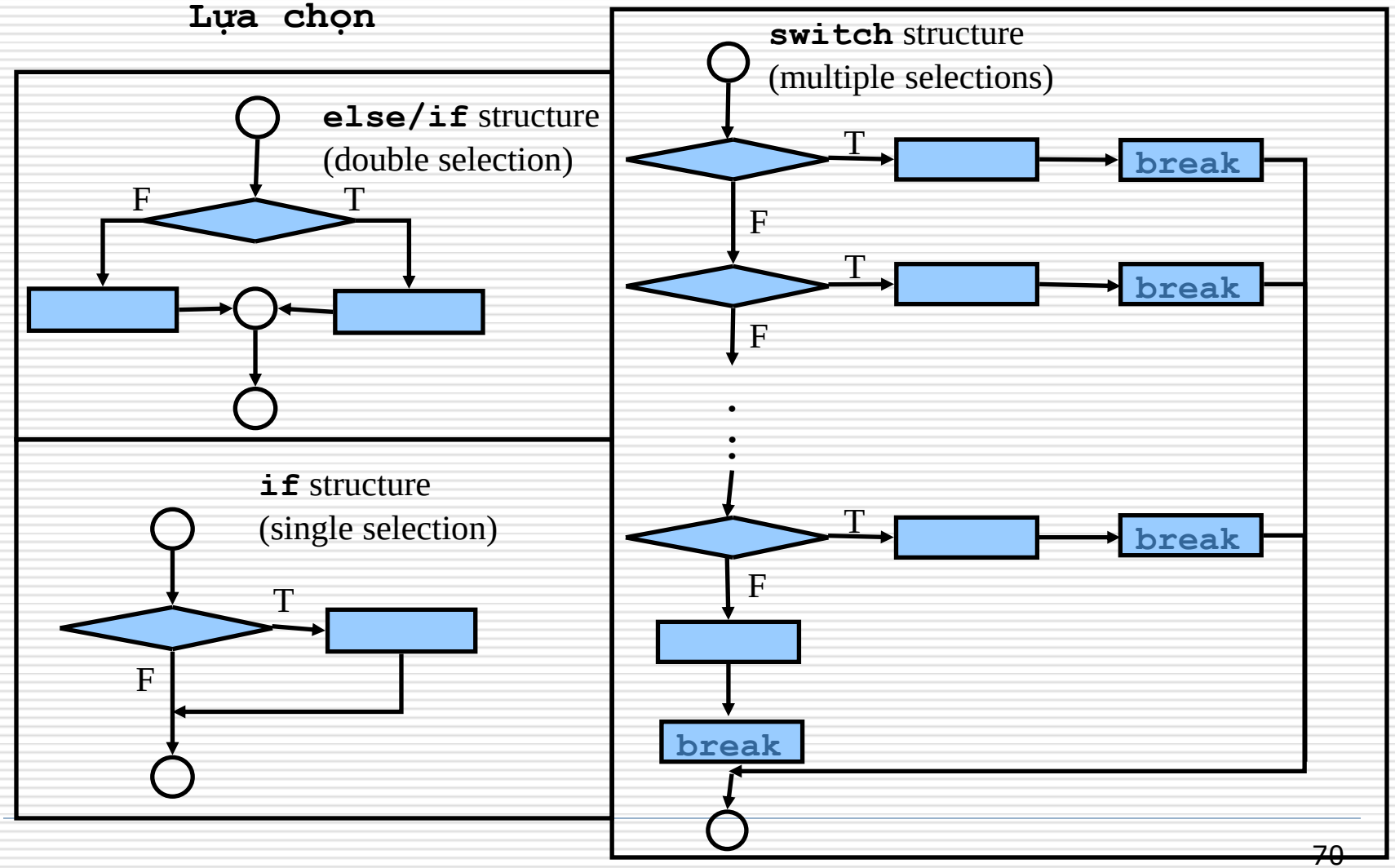


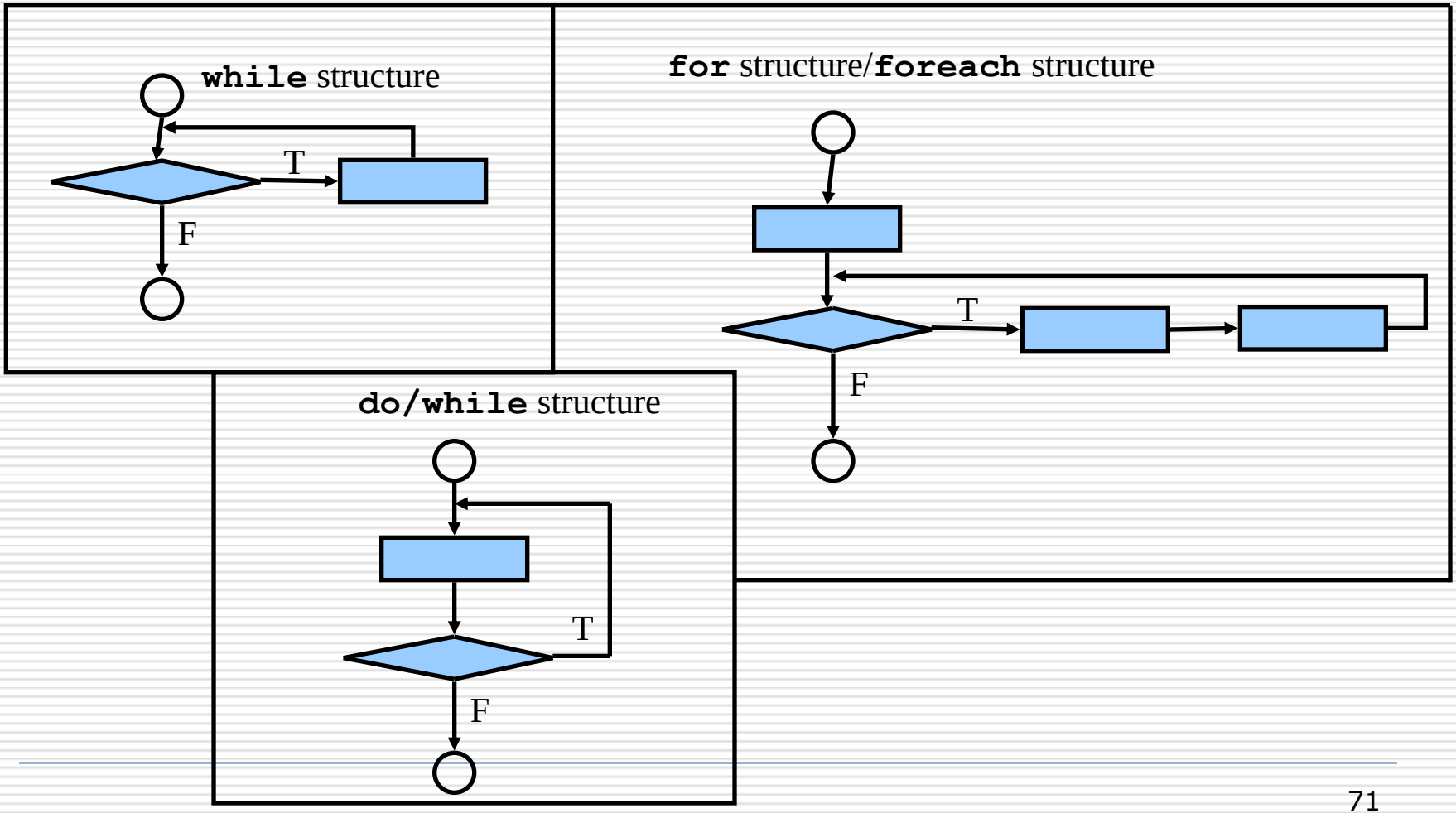
Fig. 5.22 C#'s single-entry/single-exit sequence, selection and repetition structures. (part 1)

3.19 Tóm tắt lập trình cấu trúc



3.19 Tóm tắt lập trình cấu trúc

Lặp

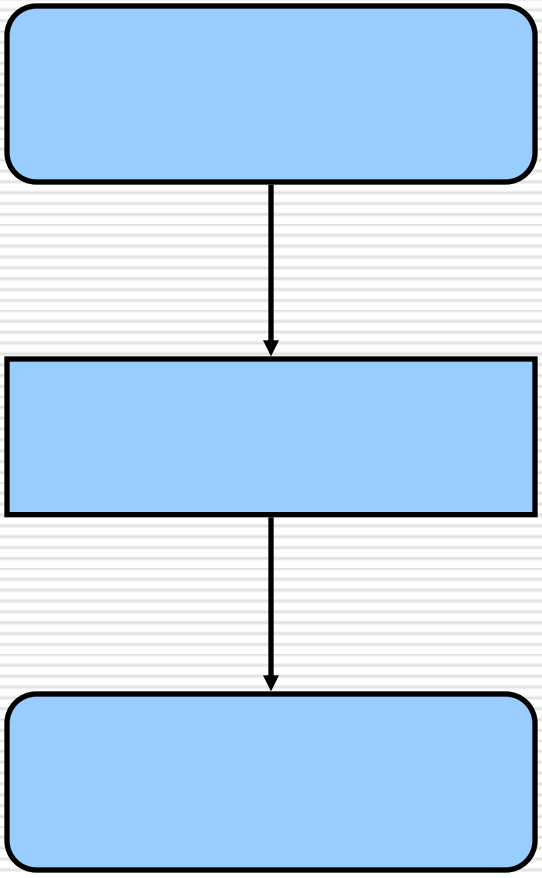


3.19 Tóm tắt lập trình cấu trúc

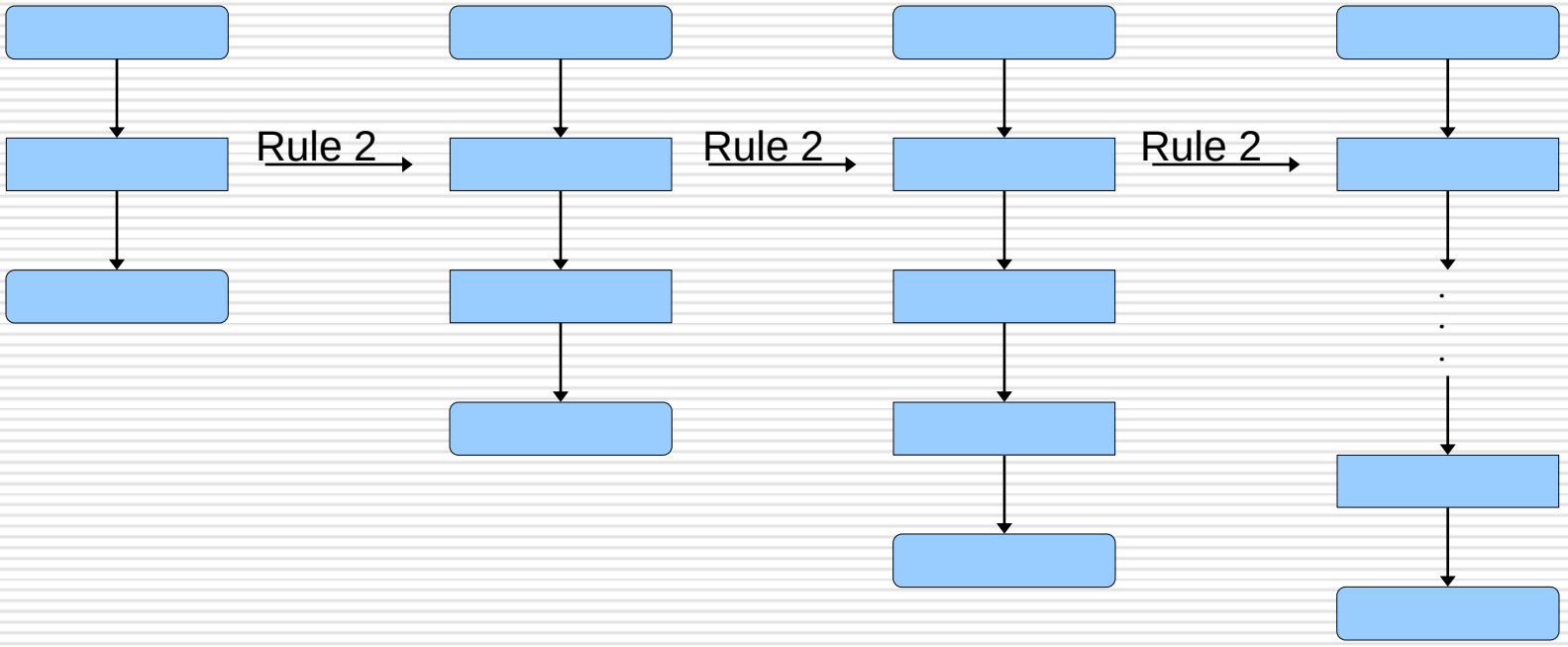
Quy tắc thiết lập cấu trúc chương trình	
1)	Bắt đầu với “simplest flowchart” (Fig. 5.24).
2)	Bất kỳ hoạt động nào cũng có thể được thay thế bởi hai hoạt động khác trong dạng tuần tự.
3)	Bất kỳ hoạt động nào cũng có thể được thay thế bởi bất kỳ cấu trúc điều khiển (sequence, **`if`**, **`if`**/**`else`**, **`switch`**, **`while`**, **`do`**/**`while`**, **`for`** or **`foreach`**).
4)	Quy tắc 2 và 3 có thể được áp dụng ngay khi bạn thích và trong bất kỳ thứ tự nào.

3.19 Tóm tắt lập trình cấu trúc

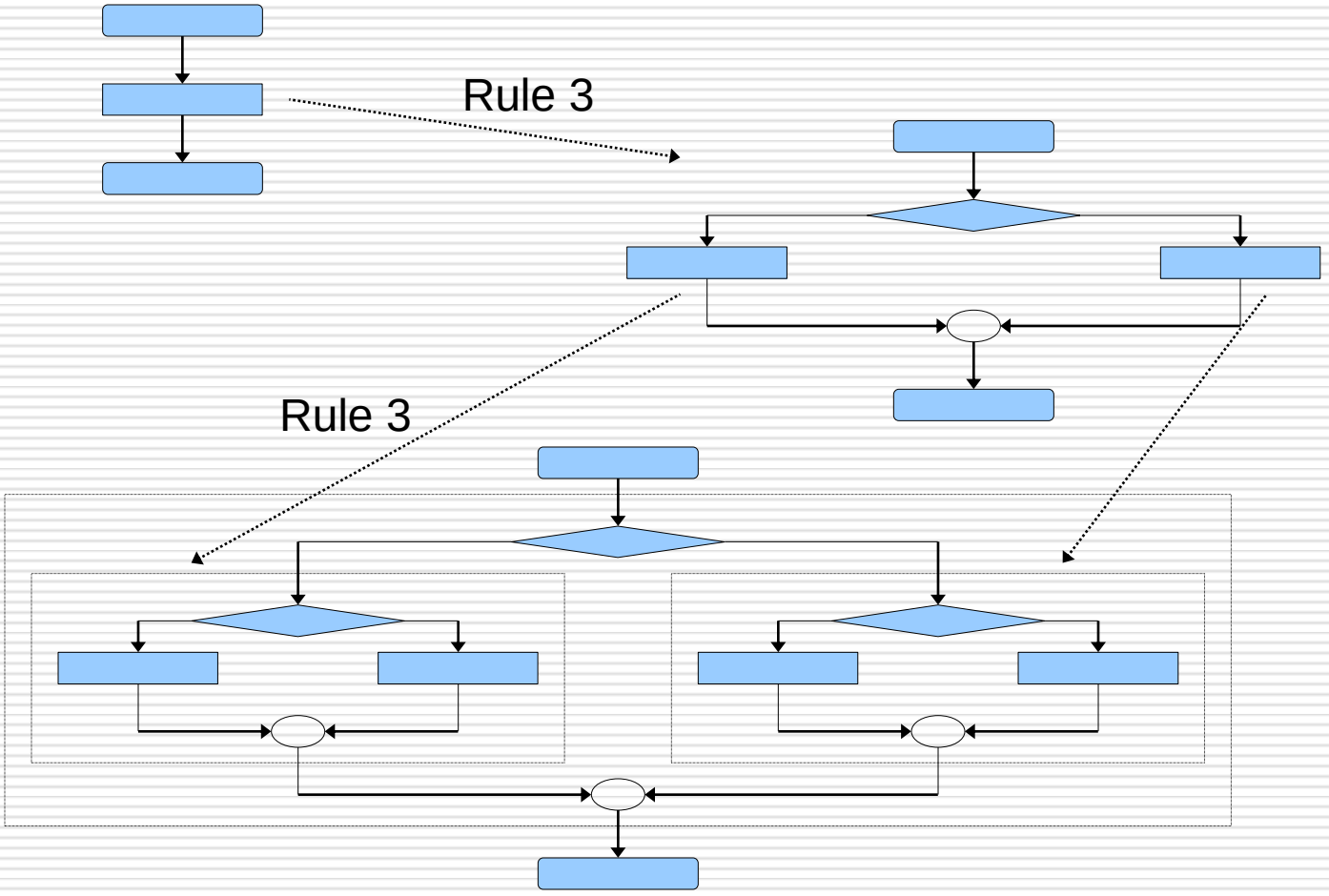
Simplest flowchart.



3.19 Tóm tắt lập trình cấu trúc

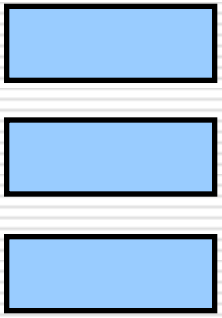


3.19 Tóm tắt lập trình cấu trúc

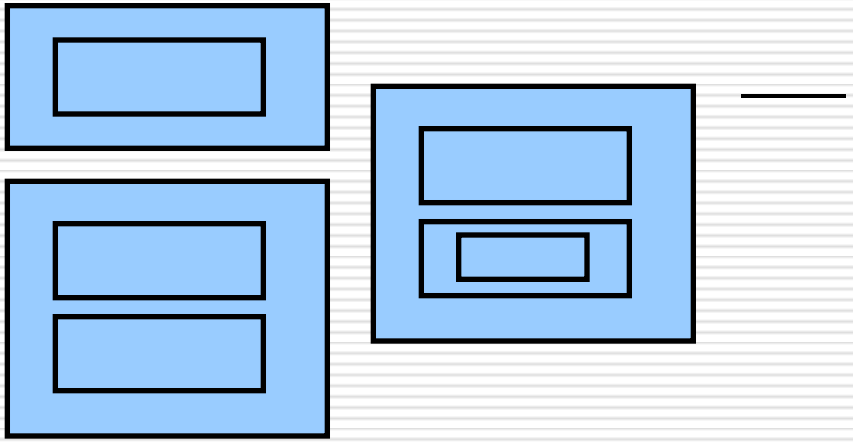


3.19 Tóm tắt lập trình cấu trúc

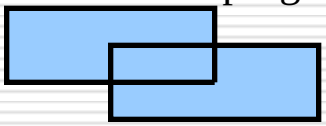
Stacked building blocks



Nested building blocks

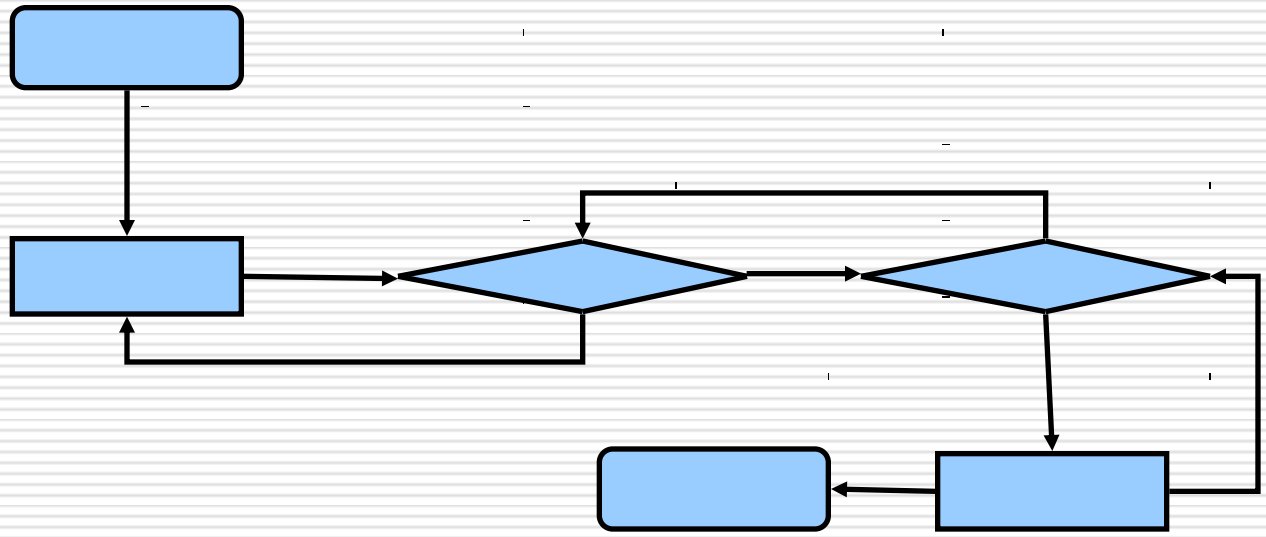


Overlapping building blocks
(illegal in structured programs)



3.19 Tóm tắt lập trình cấu trúc

Unstructured flowchart.



Câu hỏi

- ?
- ?
- ?

4. Phương thức (Method)

Trong phần này chúng ta sẽ học:

- Cách các phương thức tĩnh (static methods) và các biến (variables) kết hợp với một lớp toàn vẹn (entire class) hơn là thực thể (instances) của lớp.
- Cơ chế gọi và trả về của phương thức.
- Cách sử dụng bộ sinh số ngẫu nhiên (random-number generation) để thực hiện các ứng dụng trò chơi.
- Hiểu được cách khai báo tường minh bị giới hạn bởi miền cụ thể của ứng dụng.
- Cách nạp chồng phương thức (methods overloading).
- Phương thức đệ quy (recursive methods) là gì?
- Sự khác nhau giữa truyền thông số theo giá trị và theo tham chiếu.

4. Phương thức (Method)

4.1 Đóng gói (Packaging Code) trong C#

4.2 `static Methods, static Variables và Class Math`

4.3 Khai báo phương thức với nhiều thông số

4.4 Lưu ý trong khai báo và sử dụng phương thức

4.5 Ngăn xếp gọi phương thức (Method Call Stack) và bảng ghi hoạt động (Activation Records)

4.6 Sự tăng cấp tham số và khuôn mẫu (Argument Promotion and Casting)

4.7 Framework Class Library

4.8 Bộ sinh số ngẫu nhiên (Random-Number Generation)

4.9 Giới thiệu kiểu liệt kê

4.10 Phạm vi khai báo (Scope of Declarations)

4.11 Nạp chồng phương thức (Method Overloading)

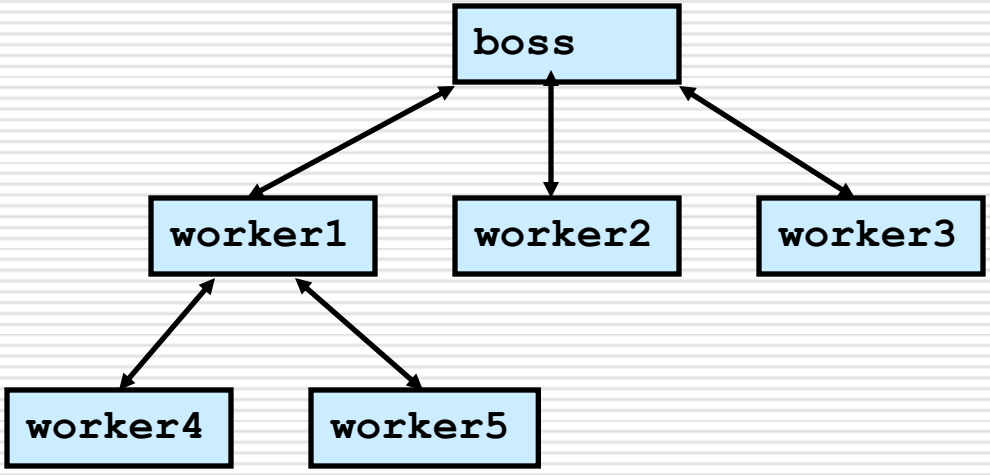
4.12 Đệ quy (Recursion)

4.13 Truyền thông số thông số theo trị và theo tham chiếu (Passing Arguments: Pass-by-Value vs. Pass-by-Reference)

4.1 Đóng gói (Packaging Code) trong C#

- Modules
  - Namespace
  - Class
  - Method
- Cho phép sử dụng các lớp và các phương thức mà không cần biết bên trong chúng làm việc thế nào, chỉ cần biết chúng làm được gì
- .NET Framework Class Library (FCL)
  - Giúp tăng khả năng sử dụng lại
  - **`Console`**
  - **`MessageBox`**

4.1 Đóng gói (Packaging Code) trong C#



4.2 static Methods, static Variables và Class Math

- **Math** class
  - Cho phép người sử dụng thực hiện các phép tính toán học thông dụng
  - Các hằng số
    - **Math.PI** = 3.1415926535…
    - **Math.E** = 2.7182818285…

Method	Description	Example

`Abs( x )`	absolute value of *x*	`Abs( 23.7 )` is `23.7`
		`Abs( 0 )` is `0`
		`Abs( -23.7 )` is `23.7`

`Ceiling( x )`	rounds *x* to the smallest integer not less than *x*	`Ceiling( 9.2 )` is `10.0` `Ceiling( -9.8 )` is `-9.0`
`Cos( x )`	trigonometric cosine of *x* (*x* in radians)	`Cos( 0.0 )` is `1.0`
`Exp( x )`	exponential method e^x	`Exp( 1.0 )` is approximately `2.718281828459045` `Exp( 2.0 )` is approximately `7.389056098930504`
`Floor( x )`	rounds *x* to the largest integer not greater than *x*	`Floor( 9.2 )` is `9.0` `Floor( -9.8 )` is `-10.0`
`Log( x )`	natural logarithm of *x* (base *e*)	`Log( 2.718281828459045 )` is approximately `1.0` `Log( 7.389056098930504 )` is approximately `2.0`
`Max( x, y )`	larger value of *x* and *y* (also has versions for `float`, `int` and `long` values)	`Max( 2.3, 12.7 )` is `12.7` `Max( -2.3, -12.7 )` is `-2.3`
`Min( x, y )`	smaller value of *x* and *y* (also has versions for `float`, `int` and `long` values)	`Min( 2.3, 12.7 )` is `2.3` `Min( -2.3, -12.7 )` is `-12.7`
`Pow( x, y )`	*x* raised to power *y* (x^y)	`Pow( 2.0, 7.0 )` is `128.0` `Pow( 9.0, .5 )` is `3.0`
`Sin( x )`	trigonometric sine of *x* (*x* in radians)	`Sin( 0.0 )` is `0.0`
`Sqrt( x )`	square root of *x*	`Sqrt( 900.0 )` is `30.0` `Sqrt( 9.0 )` is `3.0`
`Tan( x )`	trigonometric tangent of *x* (*x* in radians)	`Tan( 0.0 )` is `0.0`

4.2 static Methods, static Variables và Class Math

- **Tại sao phương thức Main được khai báo là static?**
  - Trong thời gian ứng dụng khởi động, lúc không có đối tượng của lớp được tạo, phương thức Main phải được gọi để bắt đầu thực thi chương trình.
  - Phương thức Main đôi khi được gọi tại một mục nhập (entry point) của ứng dụng. Khai báo Main là static cho phép môi trường thực thi triệu gọi phương thức Main mà không cần tạo một thực thể của lớp.

```
public static void Main( string args[] )
```

4.3 Khai báo phương thức với nhiều thông số

- Variables
  - Khai báo bên trong một method = local variables
  - Khai báo bên ngoài một method = global variables
  - Chỉ có phương thức định nghĩa các local variables mới biết chúng tồn tại
    - Gửi các thông số (parameters) để liên lạc với các phương thức khác
- Lý do sử dụng
  - Chia để trị (Divide and conquer)
  - Khả năng sử dụng lại (Reusability)
    - Sử dụng các lớp và phương thức khi xây dựng các khối cho những cái mới
  - Giảm việc lặp lại
    - Các phương thức có thể được gọi tại bất kỳ nơi nào của chương trình.

4.3 Khai báo phương thức với nhiều thông số

- Cách viết một phương thức
  - Phần đầu (Header)

    ReturnType Properties *Name*(*Param1*, *Param2*,…)
  - Phần thân (Body)
    - Chứa các đoạn mã về những gì mà phương thức làm
    - Chứa giá trị trả về nếu cần thiết
  - Dùng khi gọi tại một nơi nào khác trong chương trình
    - Truyền các thông số (parameters) nếu cần thiết
  - Tất cả các phương thức phải được khai báo bên trong một lớp
- Ví dụ: Fig. 7.3, Fig. 7.4

4.4 Lưu ý trong khai báo và sử dụng phương thức

- Có 3 cách để gọi phương thức:
  - Sử dụng tên phương thức để gọi phương thức trong cùng một lớp
    `Maximum(number1, number2, number3)`
  - Sử dụng một biến chứa tham chiếu đến một đối tượng, theo sau là toán tử (.) và tên phương thức để gọi một phương thức non-static method của đối tượng được tham chiếu:
    `maximumFinder.DetermineMaximum()`
  - Sử dụng tên lớp và toán tử (.) để gọi một phương thức static của lớp đó
    `Math.Sqrt(900.0)`
- Lưu ý: một phương thức static chỉ có thể gọi các phương thức static khác trong cùng lớp và chỉ thao tác được trên các static variables trong cùng lớp đó.

4.4 Lưu ý trong khai báo và sử dụng phương thức

- Nếu phương thức không trả về một giá trị nào đó, điều khiển được trả về khi luồng chương trình đi đến cuối phương thức hoặc khi câu lệnh **`return;`** được thực thi.
- Nếu phương thức trả về một giá trị, câu lệnh **`return expression;`** sẽ đánh giá expression, sau đó trả về giá trị và điều khiển về cho nơi gọi.

4.5 Ngăn xếp gọi phương thức (Method Call Stack) và bảng ghi hoạt động (Activation Records)

- Stack là một cấu trúc dữ liệu dạng last-in-first-out (LIFO).
- Khi một ứng dụng gọi một phương thức, phương thức được gọi phải biết cách trả về nơi gọi nó, vì thế địa chỉ trả về của việc gọi phương thức được đưa vào trong stack thực thi chương trình (method call stack). Nếu thực hiện việc gọi chuỗi các phương thức, các địa chỉ trả về lần lượt được đưa vào method call stack theo trật tự last-in-first-out.

4.5 Ngăn xếp gọi phương thức (Method Call Stack) và bảng ghi hoạt động (Activation Records)

- Stack thực thi chương trình cũng chứa vùng nhớ của các local variables sử dụng trong mỗi lần gọi phương thức trong quá trình thực thi một ứng dụng.
  - Dữ liệu này được lưu trữ trong một phần của stack thực thi chương trình, gọi là bảng ghi hoạt động (Activation Records).
  - Khi việc gọi một phương thức được thực hiện, bảng ghi hoạt động tương ứng được đưa vào stack thực thi chương trình.
  - Khi phương thức trả về nơi gọi, bảng ghi hoạt động tương ứng được lấy ra khỏi stack, các local variables tương ứng không còn được biết đến nữa, nếu một local variable chứa một tham chiếu đến một đối tượng thì đối tượng đó không được truy xuất tới nữa và được xóa khỏi bộ nhớ bởi bộ don rác (garbage collection).
- Do sửa chứa trong bộ nhớ là giới hạn, nếu có nhiều việc gọi phương thức được thực hiện, kéo theo có nhiều bảng ghi hoạt động tương ứng được lưu trữ trong stack thực thi chương trình , có thể xảy ra trường hợp stack bị tràn (stack overflow).

4.6 Sự tăng cấp tham số và khuôn mẫu (Argument Promotion and Casting)

- Chuyển đổi tường minh (Implicit Conversion)
  - Đối tượng được chuyển đổi sang dạng thích hợp một cách tường minh
  - Chỉ thực hiện khi trình biên dịch biết rằng không có dữ liệu bị mất
- Chuyển đổi không tường minh (Explicit Conversion)
  - Đối tượng được chuyển đổi bằng tay (manually)
  - Được yêu cầu nếu có sự mất mát dữ liệu
  - Mở rộng ra
    - Chuyển đối tượng sang dạng của lớp dẫn xuất (derived class) và sẽ phức tạp hơn.
  - Làm nhỏ lại
    - Chuyển đối tượng sang dạng của lớp cơ sở (base class) và là nguyên nhân của việc mất dữ liệu.

Type	Can be Converted to Type(s)
bool	**object**
byte	**decimal**, **double**, **float**, **int**, **uint**, **long**, **ulong**, **object**, **short** or **ushort**
sbyte	**decimal**, **double**, **float**, **int**, **long**, **object** or **short**
char	**decimal**, **double**, **float**, **int**, **uint**, **long**, **ulong**, **object** or **ushort**
decimal	**object**
double	**object**
float	**double** or **object**
int	**decimal**, **double**, **float**, **long** or **object**
uint	**decimal**, **double**, **float**, **long**, **ulong**, or **object**
long	**decimal**, **double**, **float** or **object**
ulong	**decimal**, **double**, **float** or **object**
object	None
short	**decimal**, **double**, **float**, **int**, **long** or **object**
ushort	**decimal**, **double**, **float**, **int**, **uint**, **long**, **ulong** or **object**
string	**object**

4.7 Framework Class Library

- Namespace
  - Một nhóm các lớp và phương thức của chúng
  - FCL được hình thành từ các namespaces
  - Namespaces được tập hợp, lưu trữ trong các tập tin .dll
  - .NET Framework: thư viện tập hợp tất cả các namespaces
  - Được khai báo trong chương trình bằng từ khóa **`using`**

Namespace	Description
System	Contains essential classes and data types (such as **int**, **double**, **char**, etc.). Implicitly referenced by all C# programs.
System.Data	Contains classes that form ADO .NET, used for database access and manipulation.
System.Drawing	Contains classes used for drawing and graphics.
System.IO	Contains classes for the input and output of data, such as with files.
System.Threading	Contains classes for multithreading, used to run multiple parts of a program simultaneously.
System.Windows.Forms	Contains classes used to create graphical user interfaces.
System.Xml	Contains classes used to process XML data.

4.8 Bộ sinh số ngẫu nhiên (Random-Number Generation)

- Class **`Random`**
  - Nằm bên trong namespace **`System`**
  - Thật sự ngẫu nhiên (Truly random)
    - Các số được sinh ra bằng một phương trình (equations) với một hạt mầm (seed)
  - **`randomObject.Next`**()
    - Trả về một số nằm trong khoảng từ 0 đến **`Int32.MaxValue`**
      - Int32.MaxValue = 2,147,483,647
  - **`randomObject.Next`**(*x*)
    - Trả về một số nằm trong khoảng từ 0 đến **`Int32.MaxValue`** nhưng không có số *x*
  - **`randomObject.Next`**(*x*, *y*)
    - Trả về một số nằm trong khoảng từ *x* đến số *y-1*
- *Ví dụ: Fig. 7.7, Fig. 7.8*

4.9 Giới thiệu kiểu liệt kê

- Phương án thay thế hằng là enumeration (liệt kê), gồm một tập hợp những hằng được đặt tên.
- Chúng ta định nghĩa một enumeration giống như sau:

```
public enum TimeOfDay
{
   Morning = 0,
   Afternoon = 1,
   Evening = 2
}
```

- *Ví dụ: Fig. 7.9, Fig. 7.10*

4.10 Phạm vi khai báo (Scope of Declarations)

- Thời hiệu (Duration)
  - Là thời gian để một định danh (identifier) tồn tại trong bộ nhớ
- Phạm vi (Scope)
  - Là một bộ phận chương trình mà trong đó đối tượng được tham chiếu đến
- Biến cục bộ (Local variables)
  - Được tạo ra khi có khai báo
  - Được hủy khi thoát ra khỏi khối chương trình
  - Trường hợp không có giá trị khởi tạo
    - Thông thường -> 0
    - **`bool`** -> **`false`**
    - reference variables -> **`null`**

4.10 Phạm vi khai báo (Scope of Declarations)

- Quy tắc cơ bản như sau:
  - Phạm vi của một thông số trong phần thân một phương thức là tại nơi khai báo đó xuất hiện.
  - Phạm vi của một local-variable là từ vi trí khai báo đó xuất hiện đến kết thúc khối chứa khai báo đó.
  - Phạm vi của một method, property hoặc field của một class là toàn bộ phần thân class đó.

4.10 Phạm vi khai báo (Scope of Declarations)

- Phạm vi (Scope)
  - Phạm vi của lớp (Class scope)
    - Từ lúc khai báo ({) đến lúc kết thúc (})
    - Toàn cục cho tất cả các phương thức trong lớp
      - Thay đổi trực tiếp
    - Có thể lặp lại tên
  - Phạm vi của khối (Block scope)
    - Từ lúc khai báo ({) đến lúc kết thúc (})
    - Chỉ sử dụng bên trong khối đó
      - Phải được truyền và thay đổi gián tiếp
    - Không được lặp lại tên biến
- *Ví dụ: Fig. 7.11, Fig. 7.12*

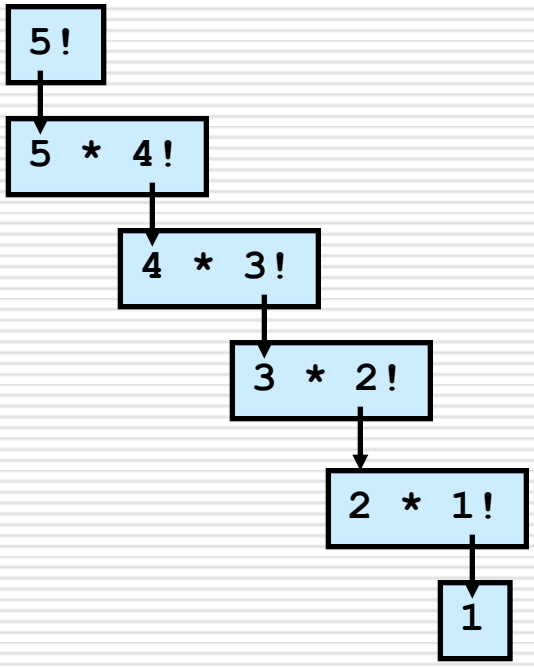
4.11 Nạp chồng phương thức (Method Overloading)

- Các phương thức có thể có cùng tên
  - Có thể có cùng tên nhưng cần có các đối số khác biệt (different arguments)
    - Các biến được truyền phải khác nhau
      - Cả trong kiểu nhận và trật tự được truyền đi
  - Thường thực hiện cùng một công việc
    - Trên các kiểu dữ liệu khác nhau
- *Ví dụ: Fig. 7.13, Fig. 7.14, Fig. 7.15*

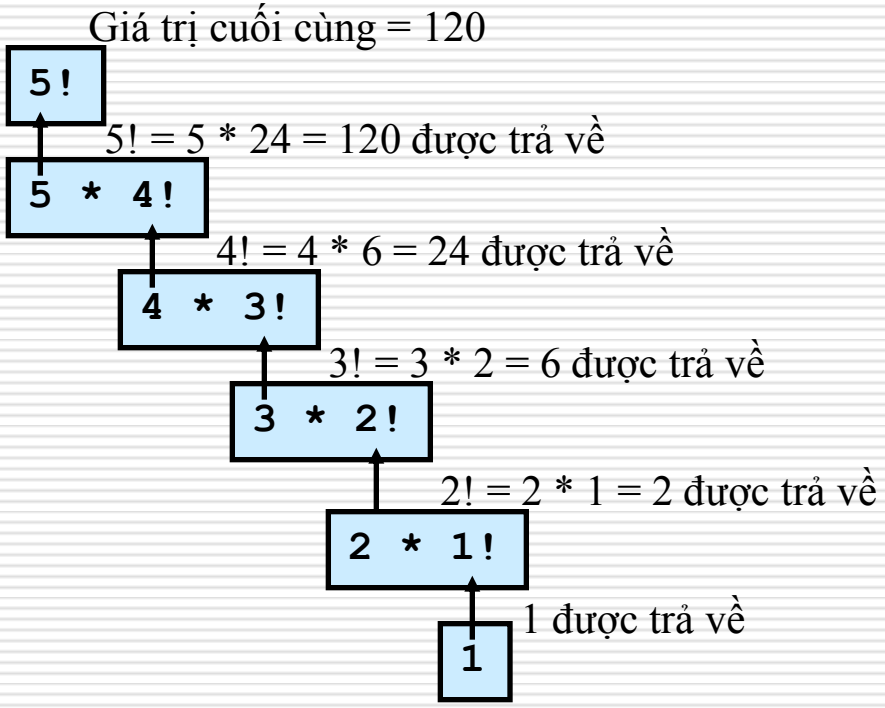
4.12 Đệ quy (Recursion)

- Các phương thức đệ quy
  - Là các phương thức có thể gọi chính nó
    - Trực tiếp (Directly)
    - Gián tiếp (Indirectly)
      - Gọi thông qua phương thức khác
  - Liên tục chia bài toán sang dạng đơn giản hơn
  - Phải hội tụ để kết thúc đệ quy
- *Ví dụ: Fig. 7.17*

4.12 Đệ quy (Recursion)



(a) Quá trình gọi đệ quy.

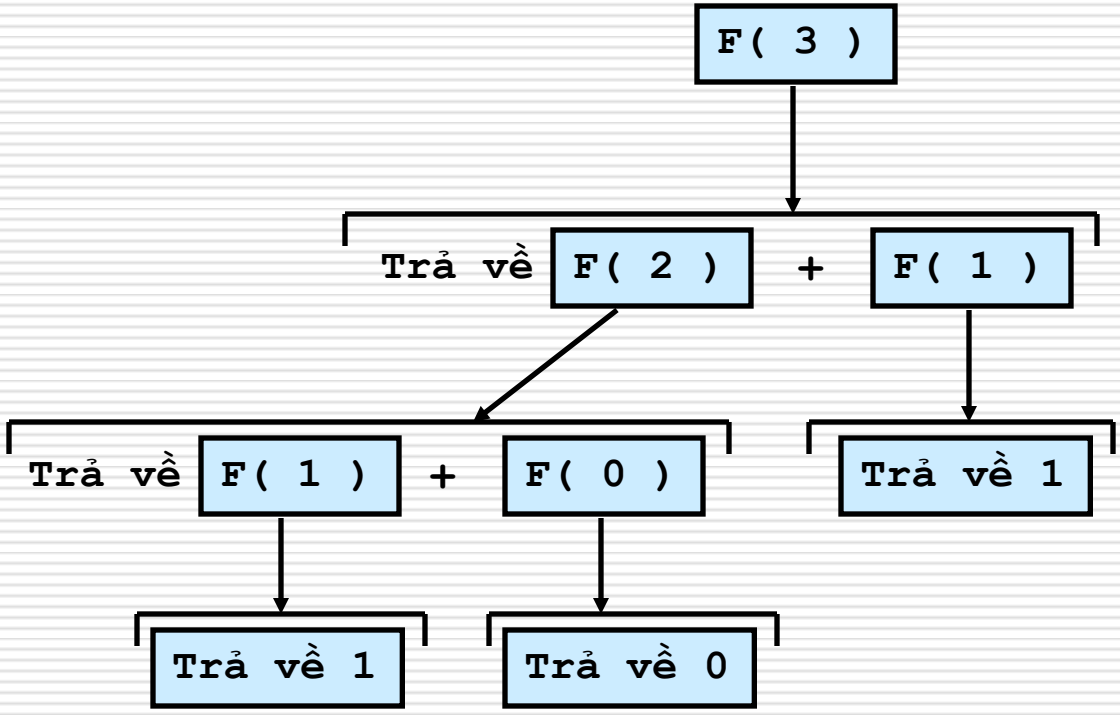


(b) Giá trị trả về mỗi lần gọi đệ quy.

4.12 Đệ quy (Recursion)

- Chuỗi số Fibonacci
  - F(0) = 0
  - F(1) = 1
  - F(n) = F(n - 1) + F(n - 2)
  - Đệ quy được sử dụng để tính F(n)
- Complexity theory
  - How hard computers need to work to perform algorithms

4.12 Đệ quy (Recursion)



4.12 Đệ quy (Recursion)

- Sự lặp lại (Iteration)
  - Sử dụng các cấu trúc lặp
    - **`while`**, **`do/while`**, **`for`**, **`foreach`**
  - Tiếp tục cho đến khi không thõa điều kiện lặp
- Đệ quy (Recursion)
  - Sử dụng các cấu trúc chọn lựa
    - **`if`**, **`if/else`**, **`switch`**
  - Lặp thông qua việc gọi phương thức
  - Tiếp tục cho đến khi đạt đến trường hợp cơ sở (base case)
  - Tạo ra một bản sao (duplicate) của các biến
    - Có thể làm lãng phí bộ nhớ và tốc độ xử lý

4.13 Truyền thông số thông số theo trị và theo tham chiếu (Passing Arguments: Pass-by-Value vs. Pass-by-Reference)

- Truyền thông số thông số theo trị (Passing by value)
  - Gửi đến một bản sao của đối tượng
  - Luôn trả về giá trị
  - Thiết lập giá trị theo mặc định
- Truyền thông số thông số theo tham chiếu (Passing by reference)
  - Gửi đến một điểm tham chiếu thật sự
    - Nguyên nhân các biến bị thay đổi trong suốt chương trình
  - Luôn trả về bằng tham chiếu
  - Từ khóa **`ref`** được xác định bằng tham chiếu
  - Từ khóa **`out`** có nghĩa là phương thức gọi sẽ thực hiện khởi tạo nó
- *Ví dụ: Fig. 7.18, Fig. 7.19*

Câu hỏi

- ?
- ?
- ?

5. Dãy (Array)

Trong phần này chúng ta sẽ học:

- Dãy (arrays) là gì?
- Sử dụng dãy đử lưu trữ và truy lục dữ liệu từ danh sách và bảng giá trị.
- Khai báo, khởi tạo dãy và đề cập đến các phần tử riêng biệt của dãy.
- Sử dụng câu lệnh **foreach** lặp đi lặp lại thông qua dãy.
- Truyền dãy vào phương thức.
- Khai báo và thao tác trên dãy đa chiều.
- Viết phương thức sử dụng danh sách đối số chiếu dài thay đổi được.
- Cách đọc các đối số command-line vào ứng dụng.

5. Dãy (Array)

5.1 Dãy (Arrays)

5.2 Khai báo và tạo dãy

5.3 Ví dụ về sử dụng dãy

5.4 Câu lệnh **`foreach`**

5.5 Truyền dãy và phần tử của dãy vào phương thức

5.6 Truyền dãy bằng trị và bằng tham chiếu

5.7 Dãy nhiều chiều

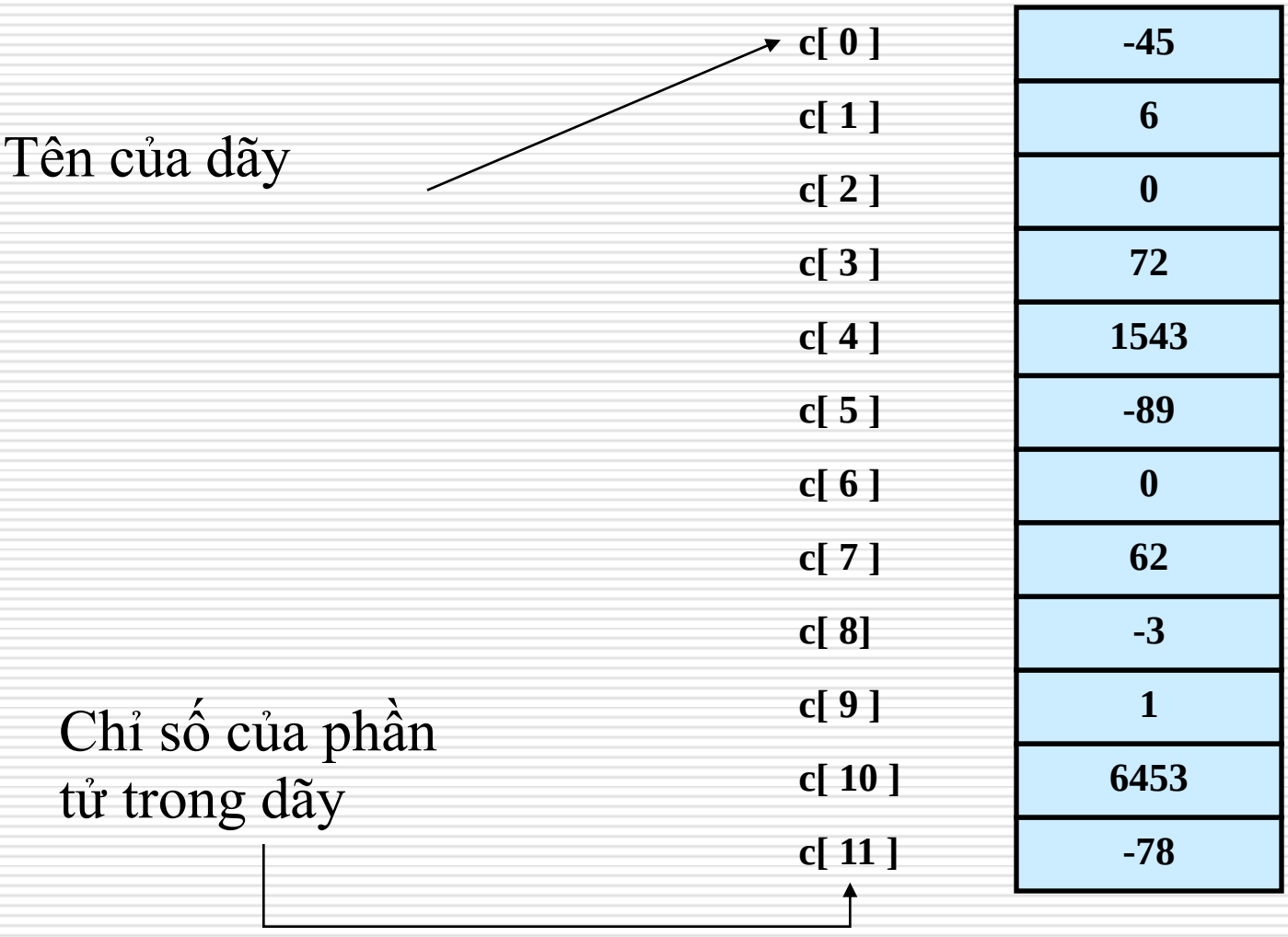
5.8 Danh sách đối số chiều dài thay đổi
(Variable-Length Argument Lists)

5.9 Sử dụng đối số Command-Line
(Using Command-Line Arguments)

5.1 Dãy (Arrays)

- Là nhóm các vùng nhớ liên tiếp nhau
  - Cùng tên
  - Cùng loại
- Các phần tử trong dãy được đánh chỉ số
- Phần tử đầu tiên của dãy được đánh chỉ số bắt đầu là 0
  - Ví dụ: Phần tử đầu tiên của dãy **c** là **c[0]**

5.1 Dãy (Arrays)



Operators	Associativity	Type
`() [] . ++ --`	left to right	highest (unary postfix)
`++ -- + - ! (type)`	right to left	unary (unary prefix)
`* / %`	left to right	multiplicative
`+ -`	left to right	additive
`< <= > >=`	left to right	relational
`== !=`	left to right	equality
`&`	left to right	boolean logical AND
`^`	left to right	boolean logical exclusive OR
`|`	left to right	boolean logical inclusive OR
`&&`	left to right	logical AND
`||`	left to right	logical OR
`?:`	right to left	conditional
`= += -= *= /= %=`	right to left	assignment

5.2 Khai báo và tạo dãy

- Lập trình viên phải xác định kiểu dữ liệu của các phần tử trong dãy
- Toán tử **new** dùng để cấp phát linh động số phần tử trong dãy
- Việc khai báo và khởi tạo dãy không nhất thiết phải trong cùng một câu lệnh
- Trong các dãy của kiểu giá trị, mỗi phần tử chứa một giá trị của kiểu khai báo
- Trong các dãy của kiểu tham chiếu, mỗi phần tử làm một tham chiếu đến một đối tượng của kiểu dữ liệu khai báo

5.2 Khai báo và tạo dãy

- Toán tử **new** có thể được dùng để xác định có bao nhiêu phần tử mà một dãy sẽ chứa
- Dãy có thể được khởi tạo từ các danh sách khởi tạo
  - Cấp phát vùng nhớ cho dãy – số lượng phần tử trong danh sách khởi tạo sẽ xác định độ lớn của dãy
  - Các phần tử trong dãy được khởi tạo với các giá trị trong danh sách khởi tạo

5.3 Ví dụ về sử dụng dãy

- Khai báo và khởi tạo dãy (Fig. 8.2)
- Sử dụng bộ khởi tạo dãy (Fig. 8.3)
- Tính giá trị được lưu trong mỗi phần tử của dãy (Fig. 8.4)
- Tính tổng các phần tử của dãy (Fig. 8.5)
- Sử dụng Bar Chart để hiển thị dữ liệu của dãy ở dạng đồ họa (Fig. 8.6)
- Sử dụng phần tử của dãy làm biến đếm (Fig. 8.7)
- Sử dụng dãy để phân tích kết quả khảo sát (Fig. 8.8)

5.4 Câu lệnh `foreach`

- Cấu trúc lặp **`foreach`** được sử dụng để lặp đi lặp lại thông qua các giá trị trong các cấu trúc dữ liệu như dãy
- Không có biến đếm
- Có một biến được dùng để biểu diễn cho giá trị của mỗi phần tử
- *Ví dụ: Fig. 8.12*

5.5 Truyền dãy và phần tử của dãy vào phương thức

- Việc truyền dãy như là thông số cho các phương thức bằng cách xác định tên dãy
- Dãy được truyền theo kiểu tham chiếu
- Các phần tử riêng biệt của dãy được truyền theo giá trị
- Các biến lưu trữ đối tượng thật ra là tham chiếu đến các đối tượng đó
- Mỗi tham chiếu là một vùng nhớ trong đó bản thân đối tượng được lưu trữ
- *Ví dụ: Fig. 8.13*

5.6 Truyền dãy bằng trị và bằng tham chiếu

- Truyền tham trị cho phương thức
  - Tạo ra một phiên bản của biến
  - Bất kỳ sự thay đổi có liên quan đến phiên bản này đều không tác động đến biến gốc
- Truyền tham trị cho phương thức
  - Tạo ra một phiên bản của tham chiếu đến đối tượng
  - Bất kỳ sự thay đổi có liên quan đến tham chiếu này đều không tác động đến biến gốc
  - Bất kỳ sự thay đổi về nôi dung của đối tượng bên trong phương thức đều tác động đến đối tượng gốc

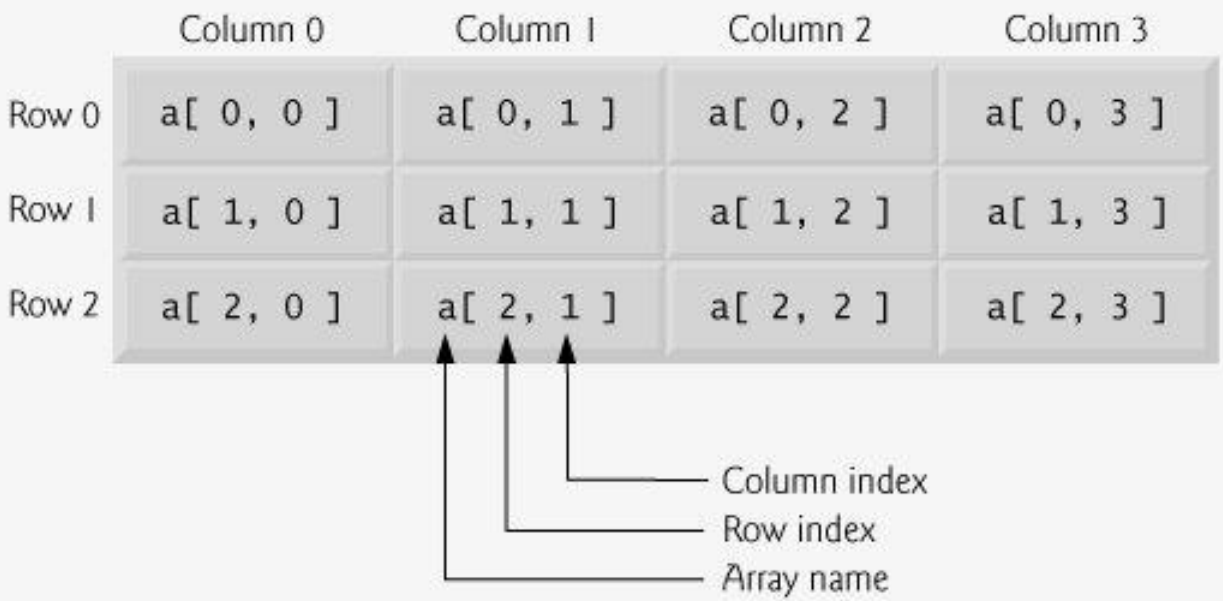
5.6 Truyền dãy bằng trị và bằng tham chiếu

- Từ khóa **ref** được dùng để truyền tham số cho phương thức theo tham chiếu
  - Kiểu giá trị của biến không được nhân bản – thay đổi của biến bên trong phương thức sẽ thay đổi đến biến bên ngoài phương thức
  - Tham chiếu đến đối tượng không được nhân bản – thay đổi của tham chiếu bên trong phương thức sẽ thay đổi đến tham chiếu bên ngoài phương thức
- Các lập trình viên cẩn thận khi sử dụng **ref**
  - Có thể tham chiếu đến null
  - Có thể dẫn đến việc phương thức làm thay đổi biến giá trị và biến tham chiếu theo cách không mong muốn
- *Ví dụ: Fig. 8.14*

5.7 Dãy nhiều chiều

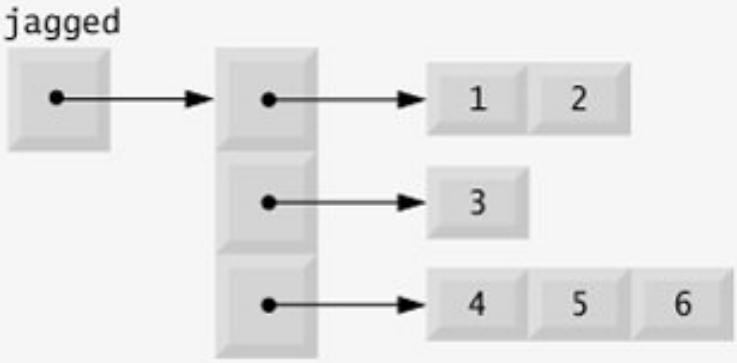
- Cần hai hay nhiều chỉ số để xác định một phần tử cụ thể
- Dãy cần hai chỉ số để xác định một phần tử gọi là dãy hai chiều
- Dãy hình chữ nhật
  - Thường biểu diễn các bảng trong đó các hàng có cùng độ lớn và các cột cũng có cùng độ lớn
  - Chỉ số thứ nhất xác định vị trí hàng và chỉ số thứ hai xác định chỉ số cột của phần tử
- Dãy không đồng nhất (Jagged Arrays)
  - Dãy của dãy (Arrays of arrays)
  - Các dãy tạo thành jagged arrays có thể có độ dài khác nhau

Dãy hình chữ nhật



Ví dụ: Fig. 8.19

Dãy không đồng nhất (Jagged Arrays)



5.8 Danh sách đối số chiều dài thay đổi (Variable-Length Argument Lists)

- Danh sách đối số chiều dài thay đổi cho phép tạo các phương thức nhận số lượng thông số tùy ý.
- Ví dụ: Fig. 8.22

5.9 Sử dụng đối số Command-Line (Using Command-Line Arguments)

- Trong nhiều hệ thống, khả năng truyền thông số từ command line vào một ứng dụng bằng cách sử dụng **`string[]`** (một dãy các chuỗi - array of strings) trong danh sách thông số của phương thức **`Main`**

- Ví dụ: Fig. 8.23

Câu hỏi

- ?
- ?
- ?

LẬP TRÌNH GIAO DIỆN

Chương 1

Giới thiệu Visual C# 2005

Mục tiêu

Trong chương này chúng ta sẽ:

- Tìm hiểu lịch sử của ngôn ngữ lập trình Visual C#.
- Tìm hiểu tổng quan về Microsoft's .NET.
- Chay thử một ứng dụng Visual C# 2005.
- Tìm hiểu Visual Studio Integrated Development Environment (IDE).
- Tìm hiểu các tính năng trợ giúp của Visual Studio 2005.
- Tìm hiểu các thành phần chính của IDE's Menus và Toolbars.
- Tìm hiểu các cửa sổ chính trong Visual Studio 2005 IDE.
- Tìm hiểu lập trình trực quan và cách phát triển chương trình đơn giản, nhanh chóng.
- Tạo, biên dịch và thực thi một chương trình Visual C# đơn giản bằng Visual Studio IDE và các kỹ thuật lập trình trực quan .

Nội dung chính

1. Ngôn ngữ C#
2. Giới thiệu Microsoft .NET
3. .NET Framework và Common Language Runtime
4. Chạy thử ứng dụng C#
5. Tổng quan về Visual Studio 2005 IDE
6. Menu Bar và Toolbar
7. Sử dụng Visual Studio 2005 IDE
8. Sử dụng trợ giúp
9. Sử dụng lập trình trực quan để tạo một chương trình đơn giản

1. Ngôn ngữ C#

- Được phát triển tại Microsoft bởi một đội ngũ đứng đầu là Anders Hejlsberg và Scott Wiltamuth
- Là ngôn ngữ lập trình trực quan, hướng đối tượng dẫn xuất sự kiện
- Dựa trên C, C++ và Java
- Được thiết kế riêng để dùng cho Microsoft's .NET Framework
  - Các ứng dụng trên nền Web có thể phân bố
    - Các thiết bị và máy tính để bàn
  - Chương trình có thể được truy xuất bởi bất kỳ ai thông qua bất kỳ thiết bị nào
  - Cho phép giao tiếp với các ngôn ngữ máy tính khác

1. Ngôn ngữ C# (tt)

- Môi trường thiết kế tích hợp (IDE - **I**ntegrated **D**esign **E**nvironment)
  - Làm cho việc lập trình và gở lỗi nhanh chóng và dễ dàng
  - Phát triển nhanh ứng dụng (RAD - **R**apid **A**pplication **D**evelopment)
- Giao thức truy xuất đối tượng đơn giản (SOAP - **S**imple **O**bject **A**ccess **P**rotocol)
  - Cho phép giao tiếp giữa các ngôn ngữ
    - Bất kỳ ngôn ngữ trên nền .NET
  - Giúp chia sẻ chương trình "phức tạp" thông qua internet

2. Giới thiệu Microsoft .NET

- Sự khởi xướng .NET
  - Được Microsoft vào tháng 06/2000
    - Tầm nhìn cho việc bao hàm Internet trong phát triển phần mềm
  - Độc lập ngôn ngữ hay nền tảng (platform)
    - Các ứng dụng được phát triển bởi bất kỳ ngôn ngữ tương thích .NET
      - Visual Basic .NET, Visual C++ .NET, C#...
    - Các lập trình viên có thể đóng góp bằng cách sử dụng ngôn ngữ mà họ thành thạo nhất
  - Cấu trúc có khả năng tồn tại đa nền
  - Quy trinh phát triển chương trình mới
    - Làm gia tăng hiệu suất

2. Giới thiệu Microsoft .NET (tt)

- Các thành phần chủ yếu của .NET
  - Các dịch vụ web (Web services)
    - Các chương trình ứng dụng được sử dụng thông qua Internet
  - Phần mềm có khả năng sử dụng lại (Software reusability)
    - Các dịch vụ web cung cấp các giải pháp:
      - Chi phí ít hơn việc phát triển nhiều giải pháp cùng một lúc không sử dụng lại được
      - Các ứng dụng đơn giản thực hiện tất cả các hoạt động của doanh nghiệp: Quản lý thuế, hóa đơn,…
    - Các thành phần trước khi đóng gói (Pre-packaged components)
      - Làm cho quá trình phát triển ứng dụng nhanh chóng và dễ dàng hơn
      - Những người phát triển phần mềm không cần quan tâm đến các chi tiết của các thành phần

3 .NET Framework và Common Language Runtime

- **Các ngôn ngữ lập trình trên nền .NET**

APL	Mondrian
C#	Oberon
COBOL	Oz
Component Pascal	Pascal
Curriculum	Perl
Eiffel	Python
Forth	RPG
Fortran	Scheme
Haskell	Smalltalk
Java	Standard ML
JScript	Visual Basic
Mercury	Visual C++

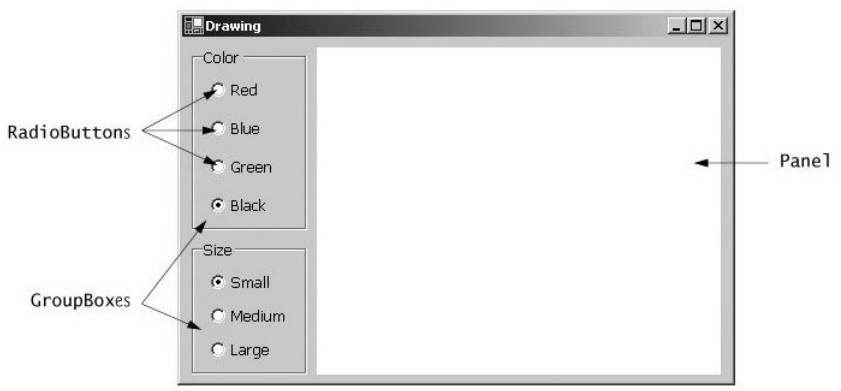
.NET Framework

- Trái tim của chiến lược .NET
  - Quản lý, thực thi các ứng dụng và dịch vụ Web
  - Quản lý bảo mật, bộ nhớ và các năng lực lập trình khác
- Bao gồm Framework class library (FCL)
  - Các lớp trước khi đóng gói (Pre-packaged classes) sẵn sàng cho việc sử dụng lại
  - Được sử dụng bởi bất kỳ ngôn ngữ .NET
  - Các chi tiết được chứa trong Common Language Specification (CLS)
  - Làm cho framework dễ chuyển đổi sang các nền khác
- Thực thi chương trình bằng Common Language Runtime (CLR)

Common Language Runtime (CLR)

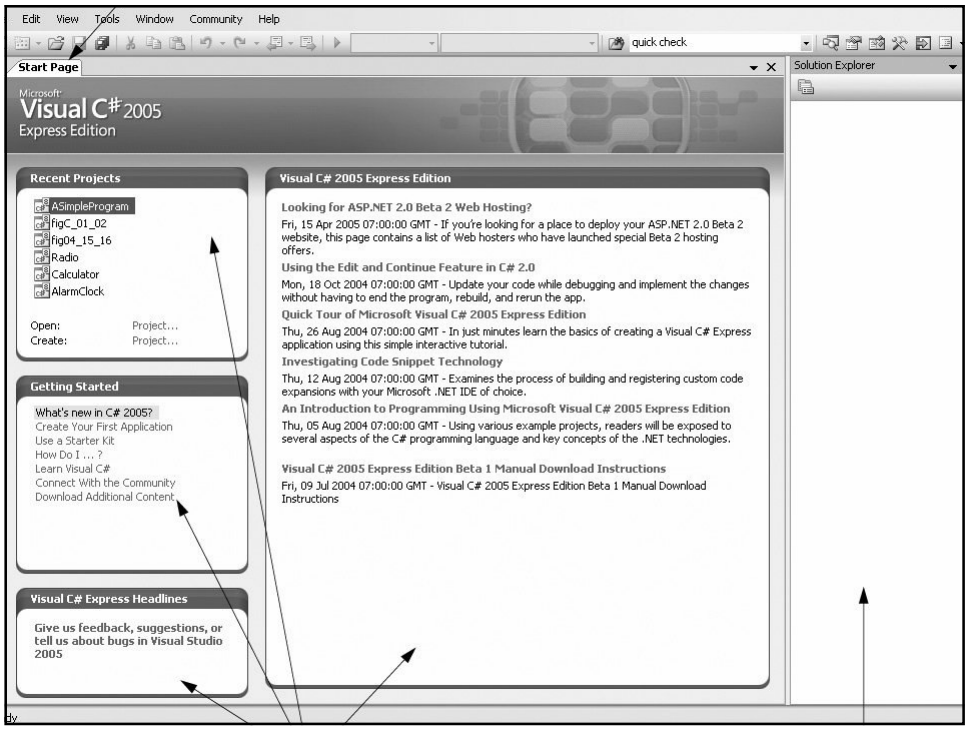
- Là bộ phận trung tâm của framework
  - Thực thi các chương trình Visual Basic .NET
- Quy trình biên dịch
  - Có 2 quá trình biên dịch xảy ra
    - Các chương trình được biên dịch bởi Microsoft Intermediate Language (MSIL)
      - Xác định các lệnh cho CLR
    - Mã MSIL được dịch sang mã máy
      - mã máy cho một nền riêng biệt

4. Chạy thử ứng dụng C#



5. Tổng quan về Visual Studio 2005 IDE

- Visual Studio® 2005 là môi trường phát triển tích hợp của Microsoft (Microsoft's Integrated Development Environment - IDE) để tạo, thực thi và gở lỗi được viết cho các ngôn ngữ lập trình.NET.

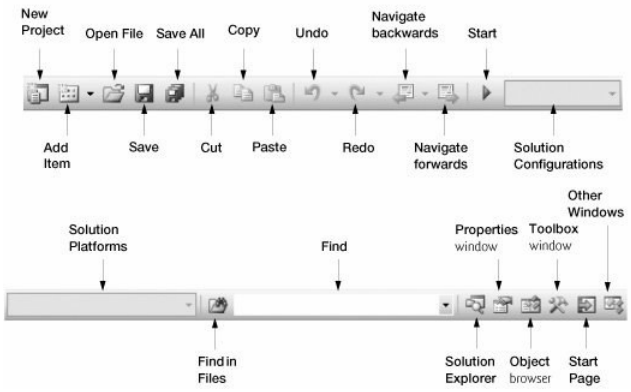


6. Menu Bar và Toolbar

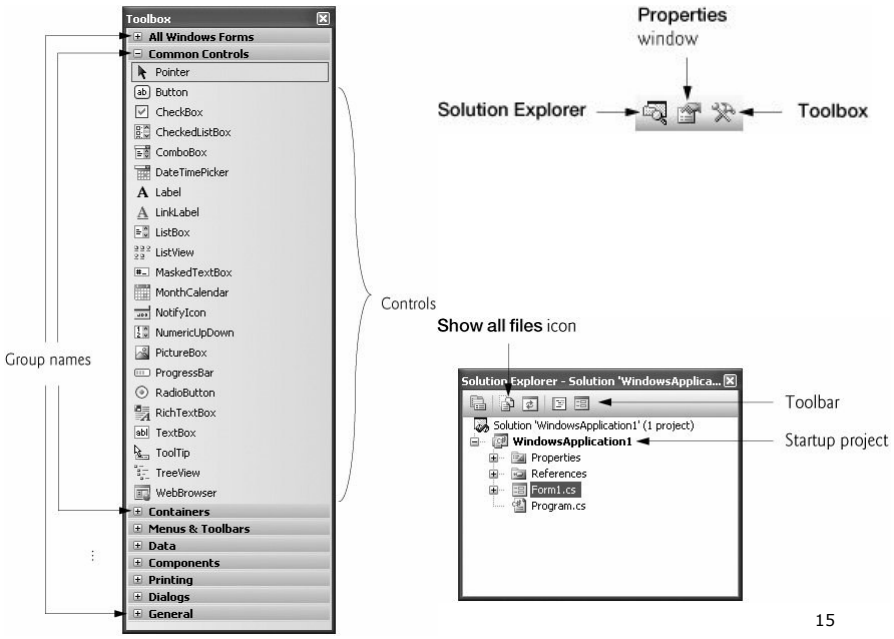
- Menu Bar

File Edit View Project Build Debug Data Format Tools Window Community Help

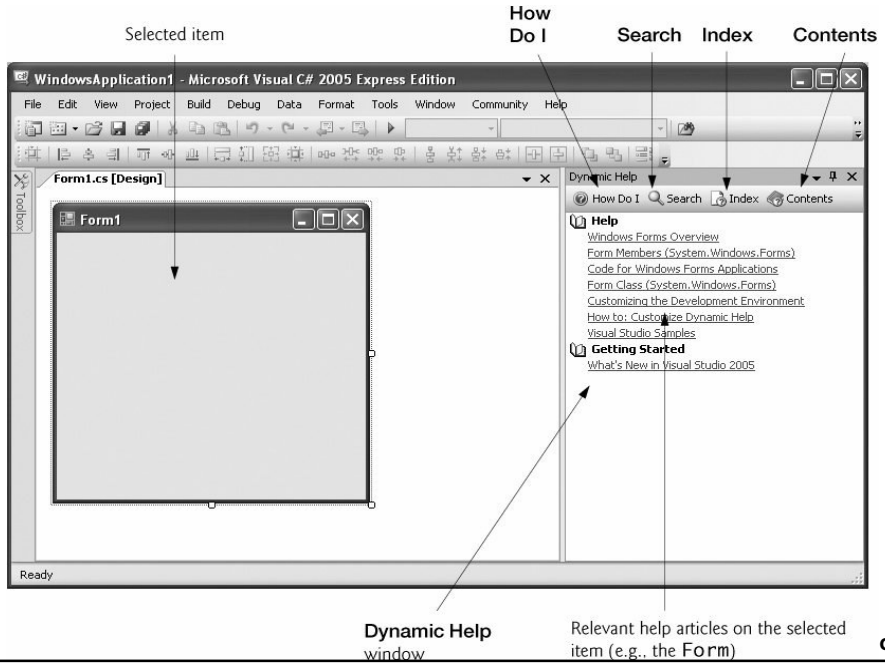
- Toolbar



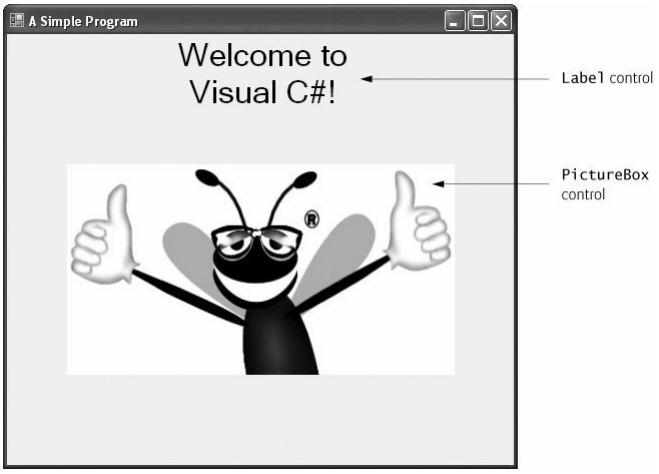
7. Sử dụng Visual Studio 2005 IDE



8. Sử dụng trợ giúp



9. Sử dụng lập trình trực quan để tạo một chương trình đơn giản



Câu hỏi

- ?
- ?
- ?

LẬP TRÌNH GIAO DIỆN

Chương 2

Ngôn ngữ C#

1

Nội dung chính

1. **Giới thiệu C#**
2. **Lớp (Class) và Đối tượng(Object)**
3. **Cấu trúc điều khiển chương trình (Control Statements)**
4. **Phương thức (Method)**
5. **Dãy (Array)**

1. Giới thiệu C#

Trong phần này chúng ta sẽ được học:

- Cách viết một ứng dung C# đơn giản.
- Cách viết các câu lệnh để nhập, xuất dữ liệu.
- Cách khai báo và sử dụng kiểu dữ liệu.
- Cách lưu trữ và truy lục dữ liệu trong bộ nhớ.
- Cách sử dụng các toán tử số học.
- Thứ tự ưu tiên giữa các toán tử.
- Cách sử dụng các toán tử so sánh.
- Cách sử dụng các hộp thoại để hiển thị các thông điệp.

1. Giới thiệu C#

1.1 Giới thiệu C#

1.2 Chương trình hiển thi một dòng văn bản

1.3 Chương trình cộng số nguyên

1.4 Bộ nhớ

1.5 Toán tử số học

1.6 Toán tử so sánh

1.1 Giới thiệu C#

- Console applications
  - Không có các thành phần trực quan
  - Chỉ kết xuất dưới dạng văn bản
  - Có 2 loại:
    - MS-DOS prompt: Windows 95/98/ME
    - Command prompt: Windows 2000/NT/XP
- Windows applications
    - Các biểu mẫu (Forms) với nhiều loại phần tử nhập
    - Chứa giao diện người dùng đồ họa (Graphical User Interfaces - GUIs)

1.1 Giới thiệu C# (tt)

- Chú thích (Comments)
  - Chú thích đơn: //…
  - Chú thích có nhiều dòng: /* … */
  - Chú thích được lờ đi khi biên dịch
  - Chỉ dùng khi người đọc mã nguồn chương trình
- Namespaces
  - Là các nhóm đặc trưng có liên quan với nhau của C# trong một phân loại
  - Cho phép sử dụng mã nguồn lại dễ dàng
  - Có rất nhiều Namespaces trong thư viện .NET framework
  - Namespace cung cấp cho ta cách mà chúng ta tổ chức quan hệ giữa các lớp và các kiểu khác
- Khoảng trắng (White Space)
  - Bao gồm: spaces, newline characters và tabs

1.1 Giới thiệu C# (tt)

- Các từ khóa (Keywords)
  - Là các từ không được phép sử dụng để khai báo biến, tên lớp
  - Có chức năng đặc biệt không thay đổi trong ngôn ngữ C#. Ví dụ: **`class`**
  - Tất cả các từ khóa đều ở dạng ký tự thường
- Các lớp (Classes)
  - Tên lớp chỉ bao gồm một từ
  - Tên lớp được viết hoa ở ký tự đầu tiên. Ví dụ: MyFirstProgram
  - Mỗi tên lớp là một từ định danh
    - Có thể chứ ký tự (letters), ký số (digits), và underscores (_)
    - Không được bắt đầu bằng ký số
    - Có thể bắt đầu bằng ký tự @

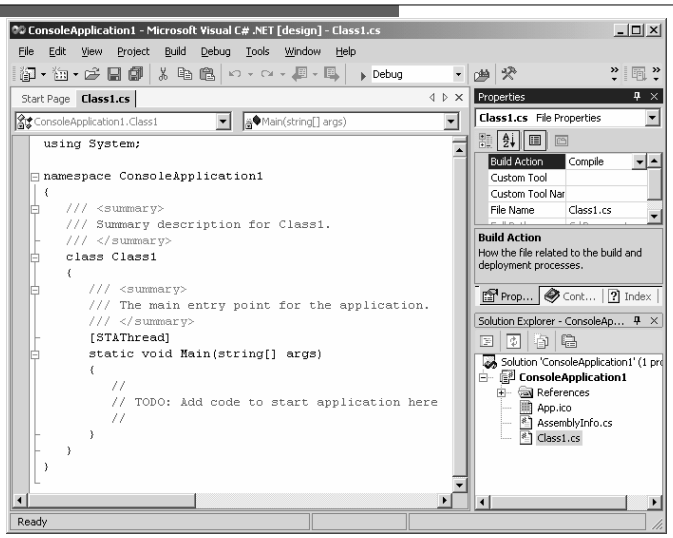
1.1 Giới thiệu C# (tt)

  - Phần thân của lớp bắt đầu bằng ký tự { và kết thúc bằng ký tự }
- Phương thức (Methods)
  - Xây dựng các khối chương trình (blocks of programs)
  - Phương thức **`Main`**
    - Mỗi console hoặc windows application đều phải có
    - Tất cả các chương trình bắt đầu bằng cách thực thi phương thức **`Main`**
  - Phần thân của phương thức bắt đầu bằng ký tự { và kết thúc bằng ký tự }
- Các câu lệnh (Statements)
  - Tất cả mọi thứ đặt trong (“”) được xem là một chuỗi ký tự
  - Mỗi câu lệnh phải được kết thúc bằng dấu chấm phẩy (;)

1.1 Giới thiệu C# (tt)

- Giao diện người dùng đồ họa (GUI - **G**raphical **U**ser **I**nterface)
  - GUIs được dùng để lấy dữ liệu từ phía người dùng cũng như hiển thị dữ liệu dễ dàng hơn
  - Các hộp thông điệp (Message boxes)
    - Nằm trong **`System.Windows.Forms`** namespace
    - Dùng để nhập hoặc hiển thị thông tin

1.1 Giới thiệu C# (tt)



1.2 Chương trình hiển thi một dòng văn bản

Ví dụ:

fig3.01, fig3.14, fig3.15, fig3.17

Console.Write và Console.WriteLine

- Chúng ta có hai phương thức dùng để viết ra chuỗi ký tự như sau
  - Console.Write() - Viết một giá trị ra cửa sổ window
  - Console.WriteLine() - tương tự trên nhưng sẽ tự động xuống hàng khi kết thúc lệnh
- Thí dụ sau sẽ cho giá trị nhập kiểu int và giá trị in ra kiểu chuỗi

```
int x = Console.Read();
Console.WriteLine((char)x);
```

Giá trị trả về kiểu string:

```
string s = Console.ReadLine();
Console.WriteLine(s);
```

Một số Escape sequence

Escape sequence	Description
`\n`	Newline. Position the screen cursor to the beginning of the next line.
`\t`	Horizontal tab. Move the screen cursor to the next tab stop.
`\r`	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line. Any characters output after the carriage return overwrite the previous characters output on that line.
`\\`	Backslash. Used to print a backslash character.
`\"`	Double quote. Used to print a double quote (") character.

Kiểu dữ liệu

- Các kiểu dữ liệu nguyên thủy (Primitive data types)
  - Là các kiểu dữ liệu được xây dựng sẵn trong C#
    - String, Int, Double, Char, Long…
    - Có 15 kiểu dữ liệu nguyên thủy
  - Tên mỗi kiểu dữ liệu là một từ khóa trong C#
  - Có thể khai báo các biến có cùng kiểu dữ liệu trên cùng một hàng hoặc nhiều hàng khác
- Console.ReadLine()
  - Dùng để đọc một ký tự văn bản từ của sổ console, giá trị trả về sẽ là kiểu int hoặc kiểu string tuỳ ý.
- Int32.Parse()
  - Dùng để chuyển từ chuỗi sang số

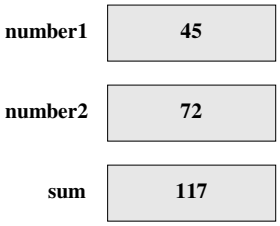
1.3 Chương trình cộng số nguyên

Ví dụ: exercise03_06, fig3.18

1.4 Bộ nhớ

- Vùng nhớ
  - Mỗi biến được chứa trong một vùng nhớ
    - Bao gồm tên (name), loại (type), kính thước (size) và giá trị (value)
  - Khi có giá trị mới được nhập thì giá trị cũ bị mất đi
  - Sử dụng biến để duy trì dữ liệu sau khi sử dụng

1.4 Bộ nhớ (tt)



1.5 Toán tử số học

- Toán tử số học (Arithmetic operations)
  - Nhân: (*), Chia: (/)
  - Modulus: (%)
  - Cộng (+), Trừ: (-)
  - Phải được viết trên cùng một dòng
  - Không có số mũ
- Phép chia
  - Kết quả trả về của phép chia tùy thuộc vào kiểu dữ liệu của biến:
    - Khi chia hai số nguyên thì kết quả luôn là một số nguyên được làm tròn
    - Để có kết quả chính xác cần sử dụng các biến có hỗ trợ số thập phân

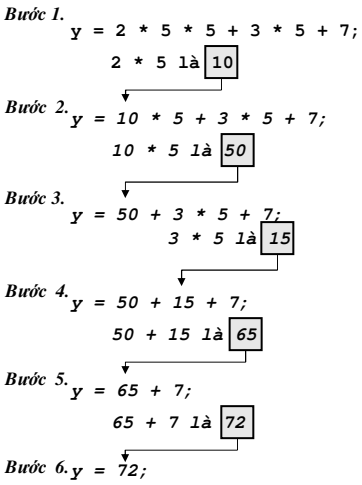
1.5 Toán tử số học (tt)

- Thứ tự ưu tiên
  - Trong dấu ngoặc đơn thực hiện trước
  - Chia, nhân và modulus được thực hiện kế tiếp
    - Từ trái sang phải
  - Cộng trừ thực hiện sau cùng
    - Từ trái sang phải

1.5 Toán tử số học (tt)

Toán tử	Ký hiệu	Biểu thức đại số	C#
Cộng	**+**	***f + 7***	***f + 7***
Trừ	**–**	***p – c***	***p - c***
Nhân	*****	***bm***	***b * m***
Chia	**/**	**x / y**	***x / y***
Modulus	**%**	**r *mod* s**	***r % s***

1.5 Toán tử số học (tt)



1.6 Toán tử so sánh (tt)

Toán tử	Ký hiệu	C#	Ý nghĩa
=	==	**x == y**	**x** bằng **y**
≠	!=	**x != y**	**x** khác **y**
>	>	**x > y**	**x** lớn hơn **y**
<	<	**x < y**	**x** nhỏ hơn **y**
≥	>=	**x >= y**	**x** lớn hơn hoặc bằng **y**
≤	<=	**x <= y**	**x** lớn hơn hoặc bằng **y**

Ví dụ: fig3.26

Câu hỏi

- ?
- ?
- ?

2. Lớp (Class) và Đối tượng(Object)

Trong phần này chúng ta sẽ được học:

- Khái niệm về Lớp (class), Đối tượng (object), Phương thức (method) và Biến thực thể (instance variable).
- Cách khai báo một lớp và sử dụng nó để tạo Đối tượng
- Cách hiện thực các hành vi của Lớp (các phương thức).
- Cách hiện thực các tính chất của Lớp (biến thực thể và thuộc tính).
- Cách gọi các phương thức của một đối tượng.
- Sự khác biệt giữa biến thực thể của một lớp và biến cục bộ của một phương thức.
- Cách sử dụng phương thức xây dựng lớp (Constructor).
- Sự khác biệt giữa kiểu giá trị (value type) và kiểu tham chiếu (reference type).

2. Lớp (Class) và Đối tượng(Object)

2.1 Lớp (Class), Đối tượng (Object), Phương thức (Method) và Biến thực thể (Instance variable)

2.2 Khai báo một lớp và khởi tạo đối tượng

2.3 Khai báo một phương thức với thông số (Parameter)

2.4 Biến thực thể (Instance Variables) và thuộc tính (Properties)

2.5 Khác biệt giữa kiểu giá trị (value type) và kiểu tham chiếu (reference type)

2.6 Khởi tạo Đối tượng bằng phương thức xây dựng lớp (Constructor)

2.7 Số Floating-Point và Decimal

2.1 Lớp (Class), Đối tượng (Object), Phương thức (Method) và Biến thực thể (Instance variable)

- Objects: đóng gói tính chất và hành vi của các đối tượng trong tự nhiên
- Class: tập hợp các đối tượng giống nhau
- Object có thể che giấu thông tin
- Methods: các khối trong chương trình
- User-defined type: class được viết bởi lập trình viên
- Class chứa:
  - Data members: member variable hoặc instance variables
  - Methods: thao tác trên data members

2.1 Lớp (Class), Đối tượng (Object), Phương thức (Method) và Biến thực thể (Instance variable)

- Abstract Data Types – che giấu hiện thực từ những Object khác
- Các biến (variables) được định nghĩa bên trong Class nhưng không phải là Method được gọi là biến thực thể (*instance variables)*
- *Member Access Modifiers*
  - *public*: thành phần được truy xuất bất cứ nơi nào trong thực thể của đối tượng hiện có
  - *private :* thành phần chỉ được truy xuất bên trong định nghĩa Class

2.1 Lớp (Class), Đối tượng (Object), Phương thức (Method) và Biến thực thể (Instance variable)

- Access methods: đọc hay cập nhật dữ liệu
- Predicate methods: kiểm tra các điều kiện
- Phương thức xây dựng lớp (Constructor)
  - Xây dựng Object của Class
  - Có thể có đối số
  - Không trả về giá trị
  - Có thể có nhiều constructor trong một class
- Toán tử **new** được sử dụng để khởi tạo thực thể của Class
- **Project < Add Class**: thêm một Class mới vào Project

2.2 Khai báo một lớp và khởi tạo đối tượng

Ví dụ: *Fig. 4.1*

```
1  // Fig. 4.1: GradeBook.cs
2  // Class declaration with one method.
3  using System;
4
5  public class GradeBook
6  {
7     // display a welcome message to the GradeBook user
8     public void DisplayMessage()
9     {
10       Console.WriteLine( "Welcome to the Grade Book!" );
11    } // end method DisplayMessage
12 } // end class GradeBook
```


2.3 Khai báo một phương thức với thông số (Parameter)

Ví dụ: *Fig. 4.4*

```
1 // Fig. 4.4: GradeBook.cs
2 // Class declaration with a method that has a parameter.
3 using System;
4
5 public class GradeBook
6 {
7    // display a welcome message to the GradeBook user
8    public void DisplayMessage( string courseName )
9    {
10      Console.WriteLine( "Welcome to the grade book for\n{0}!",
11         courseName );
12   } // end method DisplayMessage
13 } // end class GradeBook
```

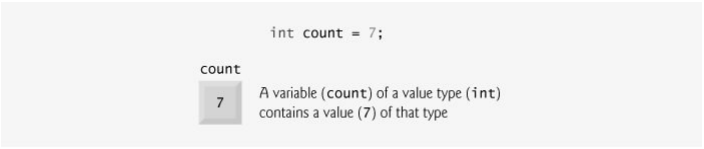
2.4 Biến thực thể (Instance Variables) và thuộc tính (Properties)

Ví dụ: Fig. 4.7

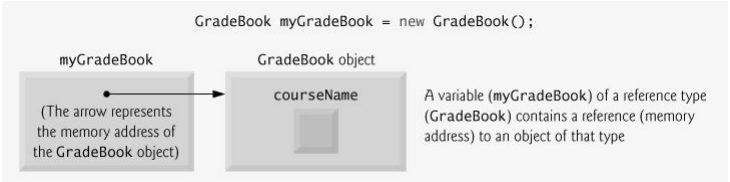
```
 8   private string courseName; // course name for this GradeBook
 9
10   // property to get and set the course name
11   public string CourseName
12   {
13      get
14      {
15         return courseName;
16      } // end get
17      set
18      {
19         courseName = value;
20      } // end set
21   } // end property CourseName
```


2.5 Khác biệt giữa kiểu giá trị (value type) và kiểu tham chiếu (reference type)

- kiểu giá trị (value type)



- kiểu tham chiếu (reference type)



2.6 Khởi tạo Đối tượng bằng phương thức xây dựng lớp (Constructor)

Ví dụ:

Fig. 4.12 GradeBook.cs

```
 9   // constructor initializes courseName with string supplied as argument
10    public GradeBook( string name )
11    {
12       CourseName = name; // initialize courseName using property
13    } // end constructor
```

Fig. 4.13: GradeBookTest.cs

```
 6 public class GradeBookTest
 7 {
 8    // Main method begins program execution
 9    public static void Main( string[] args )
10    {
11       // create GradeBook object
12       GradeBook gradeBook1 = new GradeBook( // invokes constructor
13          "CS101 Introduction to C# Programming" );
14       GradeBook gradeBook2 = new GradeBook( // invokes constructor
15       "CS102 Data Structures in C#" );
         …
```


2.7 Số Floating-Point và Decimal

- Kiểu ***float*** và ***double*** được gọi là kiểu dấu chấm động (floating-point).
- Điểm khác nhau chính giữa chúng và ***Decimal*** là ***Decimal*** lưu trữ một số thực giới hạn chính xác, trong khi **floating-point** lưu trữ một số thực giới hạn xấp xỉ nhưng có tầm giá trị lớn hơn.

Ví dụ: Fig. 4.15, Fig. 4.16

Câu hỏi

- ?
- ?
- ?

3. Cấu trúc điều khiển chương trình (Control Statements)

Trong phần này chúng ta sẽ:

- Sử dụng kỹ thuật giải quyết vấn đề.
- Phát triển giải thuật theo quy trình từ trên xuống và cải tiến từng bước.
- Sử dụng câu lệnh **if** và **if...else** để lựa chọn.
- Sử dụng câu lệnh **while** để thực thi các câu lệnh trong ứng dụng có sự lặp đi lặp lại nhiều lần.
- Sử dụng lặp theo biến đếm (counter-controlled) và lặp theo phần tử cắm canh (sentinel-controlled).
- Sử dụng các toán tử tăng, giảm, gán.
- Hiểu được cơ sở của việc lặp theo biến đếm.
- Sử dụng câu lệnh **for** và **do...while** để thực thi các câu lệnh trong ứng dụng có sự lặp đi lặp lại nhiều lần.
- Sử dụng câu lệnh **switch** trong đa lựa chọn.
- Sử dụng câu lệnh **break** và **continue** trong điều khiển luồng chương trình.
- Sử dụng các toán tử luận lý để thiết lập các biểu thức điều kiện phức tạp.

3. Cấu trúc điều khiển chương trình (Control Statements)

3.1 Giải thuật (Algorithms)
3.2 Mã giả (Pseudocode)
3.3 Cấu trúc điều khiển
3.4 Câu lệnh **if**
3.5 Câu lệnh **if...else**
3.6 Câu lệnh **while**
3.7 Lặp theo biến đếm (counter-controlled)
3.8 Lặp theo phần tử cầm canh (sentinel-controlled)
3.9 Các câu lệnh điều khiển lồng nhau (Nested Control Statements)
3.10 Các toán tử gán
3.11 Các toán tử tăng giảm
3.12 Các kiểu dữ liệu cơ bản
3.13 Cơ sở của việc lặp theo biến đếm (counter-controlled)
3.14 Câu lệnh **for**
3.15 Câu lệnh **do...while**
3.16 Câu lệnh **switch**
3.17 Câu lệnh **break** và **continue**
3.18 Các toán tử luận lý
3.19 Tóm tắt lập trình cấu trúc

3.1 Giải thuật (Algorithms)

- Thủ tục (Procedure)
  - Các hành vi mà một chương trình sẽ thực hiện
  - Thứ tự thực hiện các hành vi đó
  - Còn được gọi là giải thuật
- Điều khiển chương trình (Program control)
  - Thứ tự các công việc để một thủ tục thực hiện đúng.

3.2 Mã giả (Pseudocode)

- Là dạng ngôn ngữ nhân tạo (Artificial) và không hình thức (informal)
- Giúp các lập trình viên phát thảo giải thuật dễ dàng
- Giống như ngôn ngữ tiếng Anh
- Không phải là một ngôn ngữ lập trình thật sự
- Chuyển đổi sang mã nguồn C# một cách đơn giản

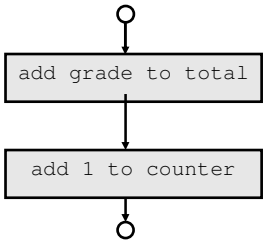
3.3 Cấu trúc điều khiển

- Chương trình điều khiển (Program of control)
  - Chương trình thực hiện một câu lệnh, sau đó chuyển đến dòng kế tiếp
    - Thực thi tuần tự
  - Chương trình thực hiện câu lệnh khác
    - Cấu trúc lựa chọn
      - Câu lệnh **`if`** và **`if/else`**
      - Câu lệnh **`goto`** (hiếm khi sử dụng)
    - Cấu trúc lặp
      - Câu lệnh lặp **`while`** và **`do/while`**
      - Câu lệnh **`for`** và **`foreach`**

3.3 Cấu trúc điều khiển

- Lưu đồ (Flow charts)
  - Được dùng để ánh xạ chương trình
  - Minh họa thứ tự các sự kiện
    - Hình chử nhật: biểu diễn một hành động
    - Hình Oval: biểu diễn điểm bắt đầu
    - Hình tròn: biểu diễn một bộ nối
    - Hình thoi: biểu diễn một điều kiện
  - Sự kết hợp giữa các cấu trúc điều khiển
    - Ngăn xếp (Stacking)
      - Đặt một cái sau một cái khác
    - Lồng nhau (Nesting)
      - Chèn một cấu trúc điều khiển sau một cấu trúc điều khiển khác

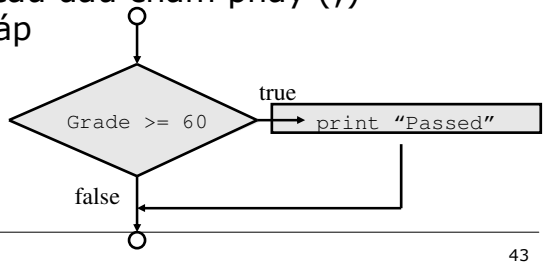
3.3 Cấu trúc điều khiển




```
total = total + grade;
counter = counter + 1;
```

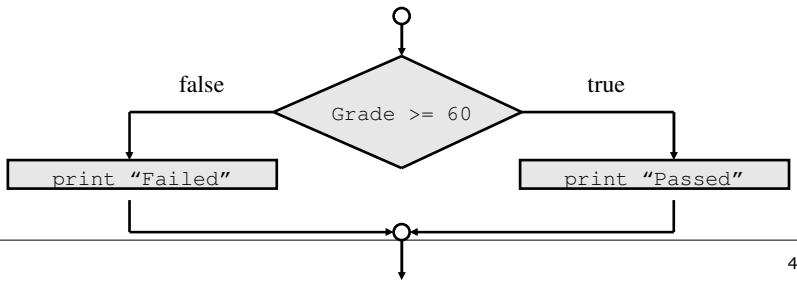
3.4 Câu lệnh **if**

- Giúp chương trình thực hiện lựa chọn
- Lựa chọn dựa trên điều khiện
  - Các biểu thức được đánh giá theo kiểu **`bool`**
  - Đúng (True): thực hiện một hành động
  - Sai (False): bỏ qua hành động
- Nhập/xuất đơn
- Không yêu cầu dấu chấm phẩy (;) trong cú pháp



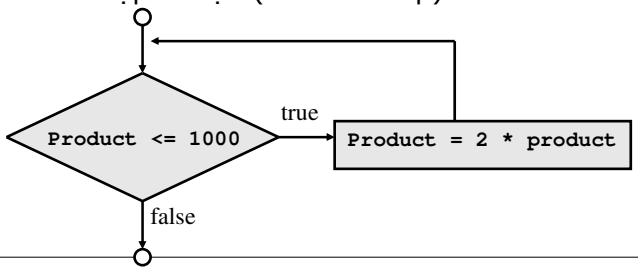
3.5 Câu lệnh **if...else**

- Tiến trình thay phiên nhau diễn ra khi câu lệnh sai
- Thích hợp hơn với một hành động có hai lựa chọn
- Cấu trúc lồng nhau có thể kiểm tra nhiều trường hợp
- Các cấu trúc có nhiều dòng lệnh phải đặt trong ({ })
  - Có thể là nguyên nhân gây ra lỗi
    - Lỗi thiên về luận lý (Fatal logic error)
    - Lỗi không thiên về luận lý (Nonfatal logic error)



3.6 Câu lệnh **while**

- Thực hiện một hành động lặp đi lặp lại
  - Tiếp tục thực hiện khi câu lệnh **while** đúng
  - Kết thúc khi khi câu lệnh **while** sai
- Có thể chứa một hay nhiều dòng lệnh
  - Phải thay đổi điều kiện
    - Lặp vô tận (Endless loop)



3.7 Lặp theo biến đếm (counter-controlled)

- Lặp theo biến đếm (Counter Controlled)
  - Được sử dụng để nhập dữ liệu cho mỗi lần lặp
    - Số lượng không thay đổi
  - Một biến đếm (counter) được dùng để xác định số lần lặp
  - Khi biến đếm đạt đến một giá trị xác định, câu lệnh kết thúc
- Mã giả

```
Set total to zero
Set grade counter to one

While grade counter is less than or equal to ten
      Input the next grade
      Add the grade into the total
      Add one to the grade counter

Set the class average to the total divided by ten
Print the class average
```

3.8 Lặp theo phần tử cầm canh (sentinel-controlled)

- Lặp theo phần tử cầm canh (Sentinel controlled)
  - Không bị bó buộc bởi số lần lặp
  - Giá trị cầm canh (Sentinel value)
    - Sử dụng để ngừng vòng lặp
    - Tránh các xung đột
      - *When flag value = user entered value*
- Chuyển đổi (Casting)
  - Cho phép một biến tạm thời sử dụng như một biến khác

3.8 Lặp theo phần tử cầm canh (sentinel-controlled)

Initialize total to zero
Initialize counter to zero

Input the first grade (possibly the sentinel)

While the user has not as yet entered the sentinel
    Add this grade into the running total
    Add one to the grade counter
    Input the next grade (possibly the sentinel)

If the counter is not equal to zero
    Set the average to the total divided by the counter
    Print the average
Else
    Print "No grades were entered"

3.9 Các câu lệnh điều khiển lồng nhau (Nested Control Statements)

- Là việc chèn một cấu trúc điều khiển vào bên trong một cấu trúc điều khiển khác
  - Nhiều vòng lặp (Multiple loops)
  - Thực hiện lặp bằng câu lệnh **`if`**

```
Initialize passes to zero
Initialize failures to zero
Initialize student to one

While student counter is less than or equal to ten
      Input the next exam result

      If the student passed
            Add one to passes
      Else
            Add one to failures

      Add one to student counter

Print the number of passes
Print the number of failures

If more than eight students passed
      Print "Raise tuition"
```


3.10 Các toán tử gán

- Có thể rút ngắn mã chương trình
  - x += 2 tương đương với x = x + 2
- Có thể thực hiện với tất cả các toán tử số học:
  - ++, -=, *=, /=, %=

Toán tử gán	Ví dụ	Diễn giải	Gán
Giả sử: **`int c = 3, d = 5, e = 4, f = 6, g = 12;`**			
`+=`	**`c += 7`**	**`c = c + 7`**	**`10`** cho **`c`**
`-=`	**`d -= 4`**	**`d = d - 4`**	**`1`** cho **`d`**
`*=`	**`e *= 5`**	**`e = e * 5`**	**`20`** cho **`e`**
`/=`	**`f /= 3`**	**`f = f / 3`**	**`2`** cho **`f`**
`%=`	**`g %= 9`**	**`g = g % 9`**	**`3`** cho **`g`**

3.11 Các toán tử tăng giảm

- Toán tử tăng (Increment operator)
  - Sử dụng để tăng giá trị thêm 1 đơn vị
  - x++
  - Giống như x = x + 1
- Toán tử giảm (Decrement operator)
  - Sử dụng để giảm giá trị 1 đơn vị
  - y--
- Khác nhau giữa Pre-increment và post-increment
  - x++ hoặc x--
    - Thực hiện hành động trước, sau đó mới thêm hoặc bớt đi 1 đơn vị
  - ++x hoặc --x
    - Thêm hoặc bớt đi 1 đơn vị trước, sau đó mới thực hiện hành động

3.12 Các kiểu dữ liệu cơ bản

Name	CTS Type	Description	Range (min:max)
sbyte	System.SByte	8-bit signed integer	-128:127 (-2^7:2^7-1)
short	System.Int16	16-bit signed integer	-32,768:32,767 (-2^{15}:2^{15}-1)
int	System.Int32	32-bit signed integer	-2,147,483,648:2,147,483,647 (-2^{31}:2^{31}-1)
long	System.Int64	64-bit signed integer	-9,223,372,036,854,775,808: 9,223,372,036,854,775,807 (-2^{63}:2^{63}-1)
byte	System.Byte	8-bit signed integer	0:255 (0:2^8-1)
ushort	System.UInt16	16-bit signed integer	0:65,535 (0:2^{16}-1)
uint	System.UInt32	32-bit signed integer	0:4,294,967,295 (0:2^{32}-1)
ulong	System.UInt64	64-bit signed integer	0:18,446,744,073,709,551,615 (0:2^{64}-1)

Các kiểu Integer

3.12 Các kiểu dữ liệu cơ bản

Kiểu dữ liệu số dấu chấm di động (Floating Point Types)

Name	CTS Type	Description	Significant Figures	Range (approximate)
Float	System.Single	32-bit single-precision floating- point	7	$\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$
Double	System.Double	64-bit double-precision floating- point	15/16	$\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$

Kiểu dữ liệu số thập phân (Decimal Type):

Name	CTS Type	Description	Significant Figures	Range (approximate)
decimal	System.Decimal	128-bit high precision decimal notation	28	$\pm 1.0 \times 10^{-28}$ to $\pm 7.9 \times 10^{28}$

3.12 Các kiểu dữ liệu cơ bản

Kiểu Boolean :

Name	CTS Type	Value
Bool	System.Boolean	true or false

Kiểu Character Type:

Name	CTS Type	Value
char	System.Char	Represents a single 16-bit (Unicode)character

Kiểu tham khảo tiền định nghĩa:

Name	CTS Type	Description
object	System.Object	The root type, from which all other types in the CTS derive (including value types)
string	System.String	Unicode character string

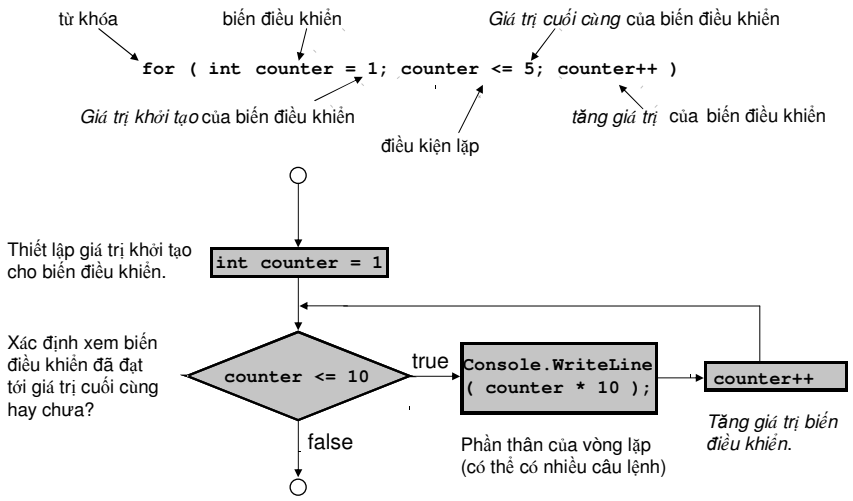
3.13 Cơ sở của việc lặp theo biến đếm (counter-controlled)

- Biến điều khiển
  - Là biến dùng để xác định nếu việc lặp được tiếp diễn
- Giá trị khởi tạo của biến điều khiển
- Sự tăng/giảm giá trị của biến điều khiển
- Điều kiện
  - Khi nào việc lặp lại được tiếp diễn

3.14 Câu lệnh **for**

- Cú pháp: **`for`** (Expression1, Expression2, Expression3)
  - Expression1 = tên của biến điều khiển
    - Có thể có nhiều biến
  - Expression2 = điều kiện để việc lặp tiếp diễn
  - Expression3 = tăng/giảm giá trị
    - Nếu Expression1 có nhiều biến, Expression3 phải có nhiều biến tương ứng
    - ++counter và counter++ là tương đương nhau
- Phạm vi của biến
  - Expression1 chỉ được sử dụng trong thân của vòng lặp **`for`**
  - Khi vòng lặp kết thúc thì biến được giải phóng

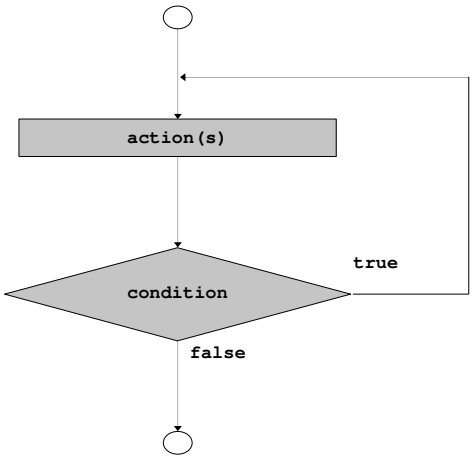
3.14 Câu lệnh **for**



3.15 Câu lệnh **do...while**

- Sử dụng vòng lặp **`while`**
  - Điều kiện được kiểm tra trước
  - Hành động được thực hiện sau
  - Việc lặp có thể bị bỏ qua
- Sử dụng vòng lặp **`do/while`**
  - Hành động được thực hiện trước
  - Sau đó kiểm tra điều kiện
  - Việc lặp diễn ra ít nhất là một lần
  - Luôn sử dụng ({ }) để tránh nhầm lẫn

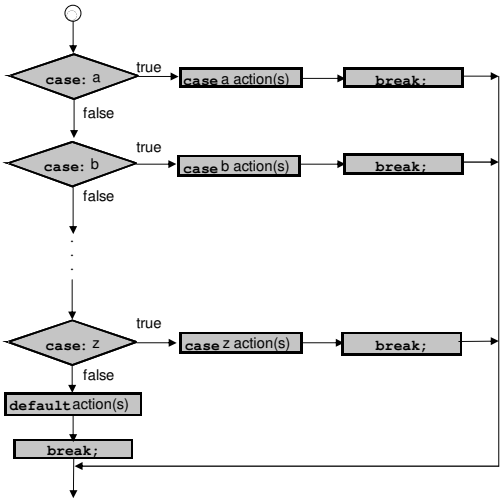
3.15 Câu lệnh **do...while**



3.16 Câu lệnh **switch**

- Các biểu thức hằng số
  - Chuỗi (String)
  - Tích phân (Integral)
- Các trường hợp (Cases)
  - Case '*x*' :
    - Dùng trong trường hợp biến là hằng
  - Các trường hợp rỗng (Empty cases)
  - Trường hợp đặc biệt (default case)
- Câu lệnh **`break`**
  - Dùng để thoát ra khỏi **`switch`**

3.16 Câu lệnh **switch**



3.17 Câu lệnh **break** và **continue**

- Sử dụng để thay đổi luồng điều khiển (flow of control)
- Câu lệnh **`break`**
  - Dùng để sớm thoát ra khỏi vòng lặp
- Câu lệnh **`continue`**
  - Dùng để bỏ qua liệc thực hiện các câu lệnh còn lại và bắt đầu lặp lại ở câu lệnh đầu tiên trong vòng lặp
- Các chương trình có thể hoàn thành mà không cần sử dụng chúng

3.18 Các toán tử luận lý

- **Các toán tử luận lý**
  - Logical AND (&)
  - Conditional AND (&&)
  - Logical OR (|)
  - Conditional OR (||)
  - Logical exclusive OR or XOR (^)
  - Logical NOT (!)
- Dùng để kết hợp nhiều điều kiện vào trong một câu lệnh

3.18 Các toán tử luận lý

expression1	expression2	expression1 `&&` expression2
`false`	`false`	`false`
`false`	`true`	`false`
`true`	`false`	`false`
`true`	`true`	`true`

Fig. 5.16 Truth table for the `&&` (logical AND) operator.

expression1	expression2	expression1 `||` expression2
`false`	`false`	`false`
`false`	`true`	`true`
`true`	`false`	`true`
`true`	`true`	`true`

Fig. 5.17 Truth table for the `||` (logical OR) operator.

3.18 Các toán tử luận lý

expression1	expression2	expression1 ^ expression2
`false`	`false`	`false`
`false`	`true`	`true`
`true`	`false`	`true`
`true`	`true`	`false`

Fig. 5.18 Truth table for the logical exclusive OR (^) operator.

expression	!expression
`false`	`true`
`True`	`false`

Fig. 5.19 Truth table for operator ! (logical NOT).

3.19 Tóm tắt lập trình cấu trúc

- Các cấu trúc điều khiển
  - Chỉ có một đầu vào
  - Chỉ có một đầu ra
  - Xây dựng các khối của chương trình
  - Cho phép lồng nhau
  - Tạo mã rõ ràng và dễ dàng hơn
  - Các cấu trúc điều khiển không chồng chéo lên nhau
    - Từ khóa `goto`

3.19 Tóm tắt lập trình cấu trúc

- 3 dạng điều khiển thiết yếu
  - Có nhiều cách để thực hiện các điều khiển này
  - Tuần tự (Sequential) (chỉ có 1 cách)
    - Straight forward programming
  - Lựa chọn (Selection): có 3 cách
    - Lựa chọn **`if`** (có 1 lựa chọn)
    - Lựa chọn **`if/else`** (có 2 lựa chọn)
    - Câu lệnh **`switch`** (có nhiều lựa chọn)
  - Lặp (Repetition): có 4 cách
    - cấu trúc **`while`**
    - cấu trúc **`do/while`**
    - cấu trúc **`for`**
    - cấu trúc **`foreach`**

3.19 Tóm tắt lập trình cấu trúc

Operators	Associativity	Type
`()` `++ --`	left to right right to left	parentheses unary postfix
`++ -- + - !` ***(type)***	right to left	unary prefix
`* / %`	left to right	multiplicative
`+ -`	left to right	additive
`< <= > >=`	left to right	relational
`== !=`	left to right	equality
`&`	left to right	logical AND
`^`	left to right	logical exclusive OR
`|`	left to right	logical inclusive OR
`&&`	left to right	conditional AND
`||`	left to right	conditional OR
`?:`	right to left	conditional
`= += -= *= /= %=`	right to left	assignment

Fig. 5.21 Precedence and associativity of the operators discussed so far.

3.19 Tóm tắt lập trình cấu trúc

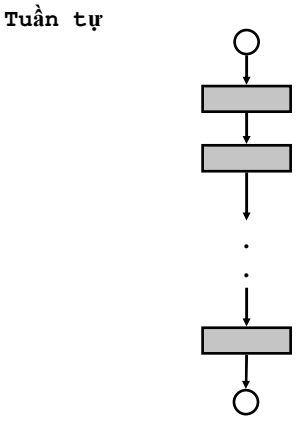
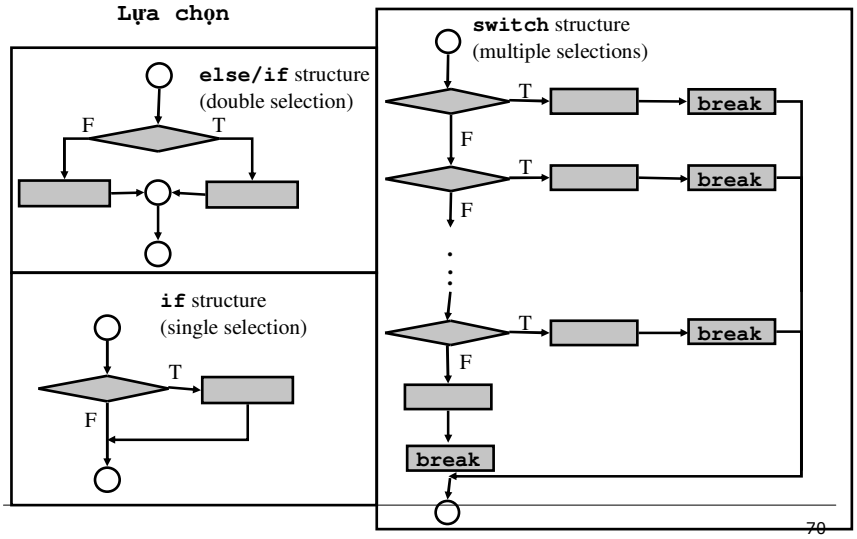


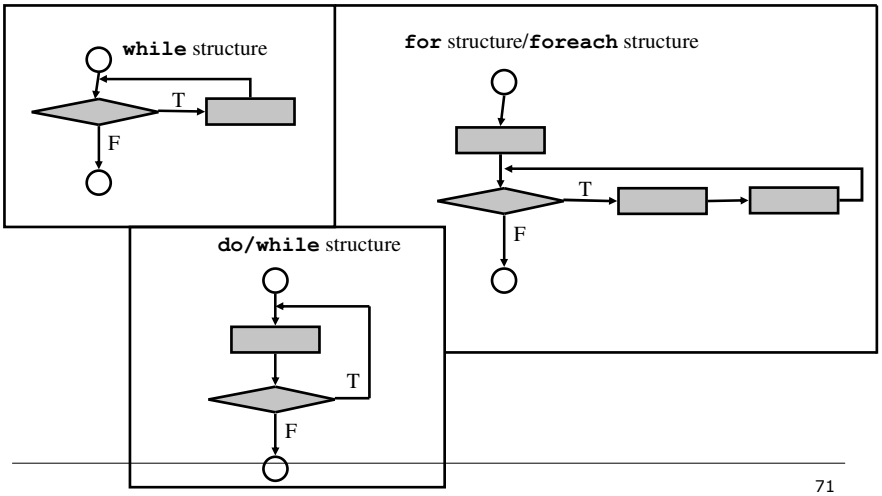
Fig. 5.22 C#'s single-entry/single-exit sequence, selection and repetition structures. (part 1)

3.19 Tóm tắt lập trình cấu trúc



3.19 Tóm tắt lập trình cấu trúc

Lặp

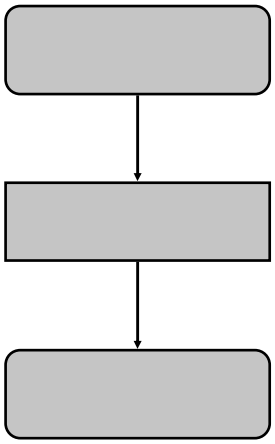


3.19 Tóm tắt lập trình cấu trúc

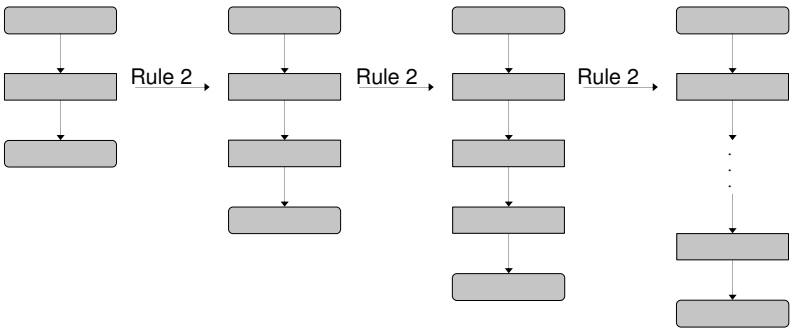
Quy tắc thiết lập cấu trúc chương trình	
1)	Bắt đầu với "simplest flowchart" (Fig. 5.24).
2)	Bất kỳ hoạt động nào cũng có thể được thay thế bởi hai hoạt động khác trong dạng tuần tự.
3)	Bất kỳ hoạt động nào cũng có thể được thay thế bởi bất kỳ cấu trúc điều khiển (sequence, **if**, **if/else**, **switch**, **while**, **do/while**, **for** or **foreach**).
4)	Quy tắc 2 và 3 có thể được áp dụng ngay khi bạn thích và trong bất kỳ thứ tự nào.

3.19 Tóm tắt lập trình cấu trúc

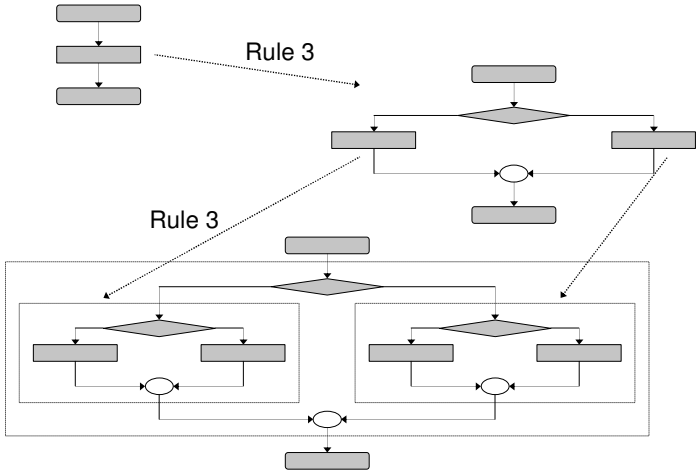
Simplest flowchart.



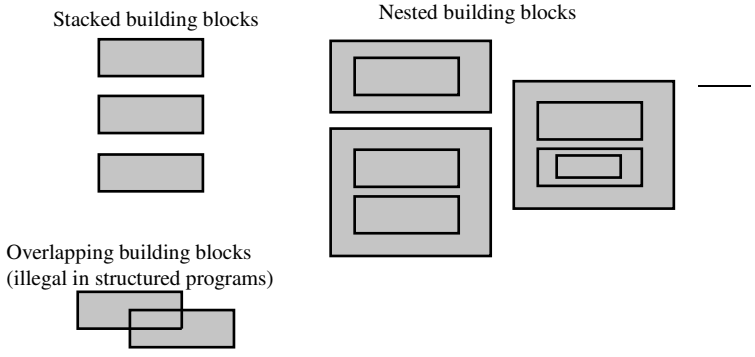
3.19 Tóm tắt lập trình cấu trúc



3.19 Tóm tắt lập trình cấu trúc

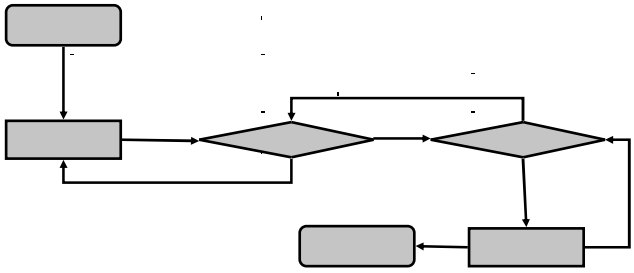


3.19 Tóm tắt lập trình cấu trúc



3.19 Tóm tắt lập trình cấu trúc

Unstructured flowchart.



Câu hỏi

- ?
- ?
- ?

4. Phương thức (Method)

Trong phần này chúng ta sẽ học:

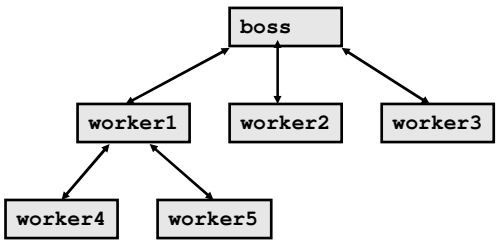
- Cách các phương thức tĩnh (static methods) và các biến (variables) kết hợp với một lớp toàn vẹn (entire class) hơn là thực thể (instances) của lớp.
- Cơ chế gọi và trả về của phương thức.
- Cách sử dụng bộ sinh số ngẫu nhiên (random-number generation) để thực hiện các ứng dụng trò chơi.
- Hiểu được cách khai báo tường minh bị giới hạn bởi miền cụ thể của ứng dụng.
- Cách nạp chồng phương thức (methods overloading).
- Phương thức đệ quy (recursive methods) là gì?
- Sự khác nhau giữa truyền thông số theo giá trị và theo tham chiếu.

4. Phương thức (Method)

4.1 Đóng gói (Packaging Code) trong C#

- Modules
  - Namespace
  - Class
  - Method
- Cho phép sử dụng các lớp và các phương thức mà không cần biết bên trong chúng làm việc thế nào, chỉ cần biết chúng làm được gì
- .NET Framework Class Library (FCL)
  - Giúp tăng khả năng sử dụng lại
  - **`Console`**
  - **`MessageBox`**

4.1 Đóng gói (Packaging Code) trong C#



4.2 static Methods, static Variables và Class Math

- **Math** class
  - Cho phép người sử dụng thực hiện các phép tính toán học thông dụng
  - Các hằng số
    - **Math.PI** = 3.1415926535…
    - **Math.E** = 2.7182818285…

Method	Description	Example

`Abs( x )`	absolute value of *x*	`Abs( 23.7 )` is `23.7`
		`Abs( 0 )` is `0`
		`Abs( -23.7 )` is `23.7`

`Ceiling( x )`	rounds *x* to the smallest integer not less than *x*	`Ceiling( 9.2 )` is `10.0` `Ceiling( -9.8 )` is `-9.0`
`Cos( x )`	trigonometric cosine of *x* (*x* in radians)	`Cos( 0.0 )` is `1.0`
`Exp( x )`	exponential method e^x	`Exp( 1.0 )` is approximately `2.7182818284590451` `Exp( 2.0 )` is approximately `7.3890560989306504`
`Floor( x )`	rounds *x* to the largest integer not greater than *x*	`Floor( 9.2 )` is `9.0` `Floor( -9.8 )` is `-10.0`
`Log( x )`	natural logarithm of *x* (base *e*)	`Log( 2.7182818284590451 )` is approximately `1.0` `Log( 7.3890560989306504 )` is approximately `2.0`
`Max( x, y )`	larger value of *x* and *y* (also has versions for `float`, `int` and `long` values)	`Max( 2.3, 12.7 )` is `12.7` `Max( -2.3, -12.7 )` is `-2.3`
`Min( x, y )`	smaller value of *x* and *y* (also has versions for `float`, `int` and `long` values)	`Min( 2.3, 12.7 )` is `2.3` `Min( -2.3, -12.7 )` is `-12.7`
`Pow( x, y )`	*x* raised to power *y* (x^y)	`Pow( 2.0, 7.0 )` is `128.0` `Pow( 9.0, .5 )` is `3.0`
`Sin( x )`	trigonometric sine of *x* (*x* in radians)	`Sin( 0.0 )` is `0.0`
`Sqrt( x )`	square root of *x*	`Sqrt( 900.0 )` is `30.0` `Sqrt( 9.0 )` is `3.0`
`Tan( x )`	trigonometric tangent of *x* (*x* in radians)	`Tan( 0.0 )` is `0.0`

4.2 static Methods, static Variables và Class Math

- **Tại sao phương thức Main được khai báo là static?**
  - Trong thời gian ứng dụng khởi động, lúc không có đối tượng của lớp được tạo, phương thức Main phải được gọi để bắt đầu thực thi chương trình.
  - Phương thức Main đôi khi được gọi tại một mục nhập (entry point) của ứng dụng. Khai báo Main là static cho phép môi trường thực thi triệu gọi phương thức Main mà không cần tạo một thực thể của lớp.

```
public static void Main( string args[] )
```


4.3 Khai báo phương thức với nhiều thông số

- Variables
  - Khai báo bên trong một method = local variables
  - Khai báo bên ngoài một method = global variables
  - Chỉ có phương thức định nghĩa các local variables mới biết chúng tồn tại
    - Gửi các thông số (parameters) để liên lạc với các phương thức khác
- Lý do sử dụng
  - Chia để trị (Divide and conquer)
  - Khả năng sử dụng lại (Reusability)
    - Sử dụng các lớp và phương thức khi xây dựng các khối cho những cái mới
  - Giảm việc lặp lại
    - Các phương thức có thể được gọi tại bất kỳ nơi nào của chương trình.

4.3 Khai báo phương thức với nhiều thông số

- Cách viết một phương thức
  - Phần đầu (Header)

    ReturnType Properties *Name*(*Param1*, *Param2*,…)
  - Phần thân (Body)
    - Chứa các đoạn mã về những gì mà phương thức làm
    - Chứa giá trị trả về nếu cần thiết
  - Dùng khi gọi tại một nơi nào khác trong chương trình
    - Truyền các thông số (parameters) nếu cần thiết
  - Tất cả các phương thức phải được khai báo bên trong một lớp
- Ví dụ: Fig. 7.3, Fig. 7.4

4.4 Lưu ý trong khai báo và sử dụng phương thức

- Có 3 cách để gọi phương thức:
  - Sử dụng tên phương thức để gọi phương thức trong cùng một lớp

    `Maximum(number1, number2, number3)`
  - Sử dụng một biến chứa tham chiếu đến một đối tượng, theo sau là toán tử (.) và tên phương thức để gọi một phương thức non-static method của đối tượng được tham chiếu:

    `maximumFinder.DetermineMaximum()`
  - Sử dụng tên lớp và toán tử (.) để gọi một phương thức static của lớp đó

    `Math.Sqrt(900.0)`
- Lưu ý: một phương thức static chỉ có thể gọi các phương thức static khác trong cùng lớp và chỉ thao tác được trên các static variables trong cùng lớp đó.

4.4 Lưu ý trong khai báo và sử dụng phương thức

- Nếu phương thức không trả về một giá trị nào đó, điều khiển được trả về khi luồng chương trình đi đến cuối phương thức hoặc khi câu lệnh **`return;`** được thực thi.
- Nếu phương thức trả về một giá trị, câu lệnh **`return expression;`** sẽ đánh giá expression, sau đó trả về giá trị và điều khiển về cho nơi gọi.

4.5 Ngăn xếp gọi phương thức (Method Call Stack) và bảng ghi hoạt động (Activation Records)

- Stack là một cấu trúc dữ liệu dạng last-in-first-out (LIFO).
- Khi một ứng dụng gọi một phương thức, phương thức được gọi phải biết cách trả về nơi gọi nó, vì thế địa chỉ trả về của việc gọi phương thức được đưa vào trong stack thực thi chương trình (method call stack). Nếu thực hiện việc gọi chuỗi các phương thức, các địa chỉ trả về lần lượt được đưa vào method call stack theo trật tự last-in-first-out.

4.5 Ngăn xếp gọi phương thức (Method Call Stack) và bảng ghi hoạt động (Activation Records)

- Stack thực thi chương trình cũng chứa vùng nhớ của các local variables sử dụng trong mỗi lần gọi phương thức trong quá trình thực thi một ứng dụng.
  - Dữ liệu này được lưu trữ trong một phần của stack thực thi chương trình, gọi là bảng ghi hoạt động (Activation Records).
  - Khi việc gọi một phương thức được thực hiện, bảng ghi hoạt động tương ứng được đưa vào stack thực thi chương trình.
  - Khi phương thức trả về nơi gọi, bảng ghi hoạt động tương ứng được lấy ra khỏi stack, các local variables tương ứng không còn được biết đến nữa, nếu một local variable chứa một tham chiếu đến một đối tượng thì đối tượng đó không được truy xuất tới nữa và được xóa khỏi bộ nhớ bởi bộ don rác (garbage collection).
- Do sức chứa trong bộ nhớ là giới hạn, nếu có nhiều việc gọi phương thức được thực hiện, kéo theo có nhiều bảng ghi hoạt động tương ứng được lưu trữ trong stack thực thi chương trình , có thể xảy ra trường hợp stack bị tràn (stack overflow).

4.6 Sự tăng cấp tham số và khuôn mẫu (Argument Promotion and Casting)

- Chuyển đổi tường minh (Implicit Conversion)
  - Đối tượng được chuyển đổi sang dạng thích hợp một cách tường minh
  - Chỉ thực hiện khi trình biên dịch biết rằng không có dữ liệu bị mất
- Chuyển đổi không tường minh (Explicit Conversion)
  - Đối tượng được chuyển đổi bằng tay (manually)
  - Được yêu cầu nếu có sự mất mát dữ liệu
  - Mở rộng ra
    - Chuyển đổi tượng sang dạng của lớp dẫn xuất (derived class) và sẽ phức tạp hơn.
  - Làm nhỏ lại
    - Chuyển đổi tượng sang dạng của lớp cơ sở (base class) và là nguyên nhân của việc mất dữ liệu.

Type	Can be Converted to Type(s)
bool	**object**
byte	**decimal**, **double**, **float**, **int**, **uint**, **long**, **ulong**, **object**, **short** or **ushort**
sbyte	**decimal**, **double**, **float**, **int**, **long**, **object** or **short**
char	**decimal**, **double**, **float**, **int**, **uint**, **long**, **ulong**, **object** or **ushort**
decimal	**object**
double	**object**
float	**double** or **object**
int	**decimal**, **double**, **float**, **long** or **object**
uint	**decimal**, **double**, **float**, **long**, **ulong**, or **object**
long	**decimal**, **double**, **float** or **object**
ulong	**decimal**, **double**, **float** or **object**
object	None
short	**decimal**, **double**, **float**, **int**, **long** or **object**
ushort	**decimal**, **double**, **float**, **int**, **uint**, **long**, **ulong** or **object**
string	**object**

4.7 Framework Class Library

- Namespace
  - Một nhóm các lớp và phương thức của chúng
  - FCL được hình thành từ các namespaces
  - Namespaces được tập hợp, lưu trữ trong các tập tin .dll
  - .NET Framework: thư viện tập hợp tất cả các namespaces
  - Được khai báo trong chương trình bằng từ khóa **`using`**

Namespace	Description
System	Contains essential classes and data types (such as **int**, **double**, **char**, etc.). Implicitly referenced by all C# programs.
System.Data	Contains classes that form ADO .NET, used for database access and manipulation.
System.Drawing	Contains classes used for drawing and graphics.
System.IO	Contains classes for the input and output of data, such as with files.
System.Threading	Contains classes for multithreading, used to run multiple parts of a program simultaneously.
System.Windows.Forms	Contains classes used to create graphical user interfaces.
System.Xml	Contains classes used to process XML data.

4.8 Bộ sinh số ngẫu nhiên (Random-Number Generation)

- Class **Random**
  - Nằm bên trong namespace **System**
  - Thật sự ngẫu nhiên (Truly random)
    - Các số được sinh ra bằng một phương trình (equations) với một hạt mầm (seed)
  - **randomObject.Next**()
    - Trả về một số nằm trong khoảng từ 0 đến **Int32.MaxValue**
      - Int32.MaxValue = 2,147,483,647
  - **randomObject.Next**(*x*)
    - Trả về một số nằm trong khoảng từ 0 đến **Int32.MaxValue** nhưng không có số *x*
  - **randomObject.Next**(*x*, *y*)
    - Trả về một số nằm trong khoảng từ *x* đến **Int32.MaxValue** nhưng không có số *y*
- *Ví dụ: Fig. 7.7, Fig. 7.8*

4.9 Giới thiệu kiểu liệt kê

- Phương án thay thế hẳng là enumeration (liệt kê), gồm một tập hợp những hằng được đặt tên.
- Chúng ta định nghĩa một enumeration giống như sau:

```
public enum TimeOfDay
{
   Morning = 0,
   Afternoon = 1,
   Evening = 2
}
```

- *Ví dụ: Fig. 7.9, Fig. 7.10*

4.10 Phạm vi khai báo (Scope of Declarations)

- Thời hiệu (Duration)
  - Là thời gian để một định danh (identifier) tồn tại trong bộ nhớ
- Phạm vi (Scope)
  - Là một bộ phận chương trình mà trong đó đối tượng được tham chiếu đến
- Biến cục bộ (Local variables)
  - Được tạo ra khi có khai báo
  - Được hủy khi thoát ra khỏi khối chương trình
  - Trường hợp không có giá trị khởi tạo
    - Thông thường -> 0
    - **`bool`** -> **`false`**
    - reference variables -> **`null`**

4.10 Phạm vi khai báo (Scope of Declarations)

- Quy tắc cơ bản như sau:
  - Phạm vi của một thông số trong phần thân một phương thức là tại nơi khai báo đó xuất hiện.
  - Phạm vi của một local-variable là từ vi trí khai báo đó xuất hiện đến kết thúc khối chứa khai báo đó.
  - Phạm vi của một method, property hoặc field của một class là toàn bộ phần thân class đó.

4.10 Phạm vi khai báo (Scope of Declarations)

- Phạm vi (Scope)
  - Phạm vi của lớp (Class scope)
    - Từ lúc khai báo ({) đến lúc kết thúc (})
    - Toàn cục cho tất cả các phương thức trong lớp
      - Thay đổi trực tiếp
    - Có thể lặp lại tên
  - Phạm vi của khối (Block scope)
    - Từ lúc khai báo ({) đến lúc kết thúc (})
    - Chỉ sử dụng bên trong khối đó
      - Phải được truyền và thay đổi gián tiếp
    - Không được lặp lại tên biến
- *Ví dụ: Fig. 7.11, Fig. 7.12*

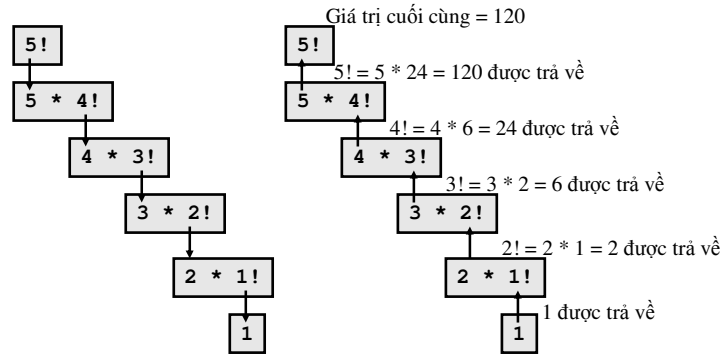
4.11 Nạp chồng phương thức (Method Overloading)

- Các phương thức có thể có cùng tên
  - Có thể có cùng tên nhưng cần có các đối số khác biệt (different arguments)
    - Các biến được truyền phải khác nhau
      - Cả trong kiểu nhận và trật tự được truyền đi
  - Thường thực hiện cùng một công việc
    - Trên các kiểu dữ liệu khác nhau
- *Ví dụ: Fig. 7.13, Fig. 7.14, Fig. 7.15*

4.12 Đệ quy (Recursion)

- Các phương thức đệ quy
  - Là các phương thức có thể gọi chính nó
    - Trực tiếp (Directly)
    - Gián tiếp (Indirectly)
      - Gọi thông qua phương thức khác
  - Liên tục chia bài toán sang dạng đơn giản hơn
  - Phải hội tụ để kết thúc đệ quy
- *Ví dụ: Fig. 7.17*

4.12 Đệ quy (Recursion)

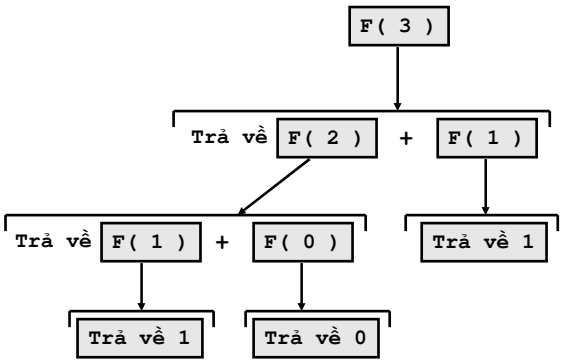


(a) Quá trình gọi đệ quy. (b) Giá trị trả về mỗi lần gọi đệ quy.

4.12 Đệ quy (Recursion)

- Chuỗi số Fibonacci
  - F(0) = 0
  - F(1) = 1
  - F(n) = F(n - 1) + F(n - 2)
  - Đệ quy được sử dụng để tính F(n)
- Complexity theory
  - How hard computers need to work to perform algorithms

4.12 Đệ quy (Recursion)



4.12 Đệ quy (Recursion)

- Sự lặp lại (Iteration)
  - Sử dụng các cấu trúc lặp
    - **`while`**, **`do/while`**, **`for`**, **`foreach`**
  - Tiếp tục cho đến khi không thõa điều kiện lặp
- Đệ quy (Recursion)
  - Sử dụng các cấu trúc chọn lựa
    - **`if`**, **`if/else`**, **`switch`**
  - Lặp thông qua việc gọi phương thức
  - Tiếp tục cho đến khi đạt đến trường hợp cơ sở (base case)
  - Tạo ra một bản sao (duplicate) của các biến
    - Có thể làm lãng phí bộ nhớ và tốc độ xử lý

4.13 Truyền thông số thông số theo trị và theo tham chiếu (Passing Arguments: Pass-by-Value vs. Pass-by-Reference)

- Truyền thông số thông số theo trị (Passing by value)
  - Gửi đến một bản sao của đối tượng
  - Luôn trả về giá trị
  - Thiết lập giá trị theo mặc định
- Truyền thông số thông số theo tham chiếu (Passing by reference)
  - Gửi đến một điểm tham chiếu thật sự
    - Nguyên nhân các biến bị thay đổi trong suốt chương trình
  - Luôn trả về bằng tham chiếu
  - Từ khóa **`ref`** được xác định bằng tham chiếu
  - Từ khóa **`out`** có nghĩa là phương thức gọi sẽ thực hiện khởi tạo nó
- *Ví dụ: Fig. 7.18, Fig. 7.19*

Câu hỏi

- ?
- ?
- ?

5. Dãy (Array)

Trong phần này chúng ta sẽ học:

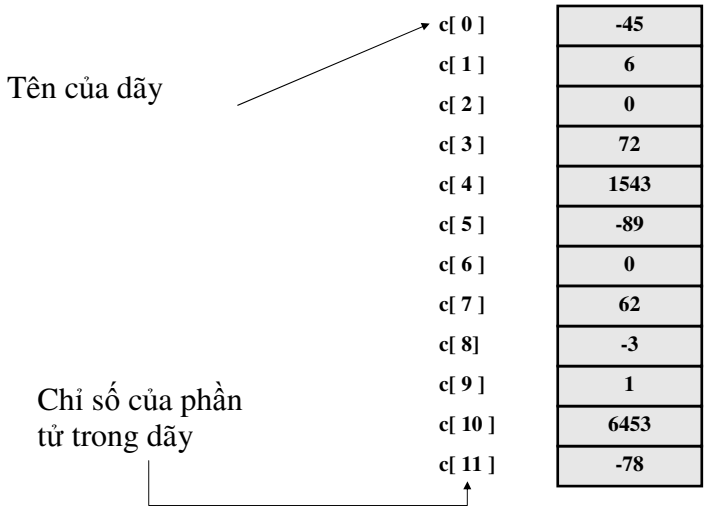
- Dãy (arrays) là gì?
- Sử dụng dãy đử lưu trữ và truy lục dữ liệu từ danh sách và bảng giá trị.
- Khai báo, khởi tạo dãy và đề cập đến các phần tử riêng biệt của dãy.
- Sử dụng câu lệnh **foreach** lặp đi lặp lại thông qua dãy.
- Truyền dãy vào phương thức.
- Khai báo và thao tác trên dãy đa chiều.
- Viết phương thức sử dụng danh sách đối số chiếu dài thay đổi được.
- Cách đọc các đối số command-line vào ứng dụng.

5. Dãy (Array)

5.1 Dãy (Arrays)

- Là nhóm các vùng nhớ liên tiếp nhau
  - Cùng tên
  - Cùng loại
- Các phần tử trong dãy được đánh chỉ số
- Phần tử đầu tiên của dãy được đánh chỉ số bắt đầu là 0
  - Ví dụ: Phần tử đầu tiên của dãy **c** là **c[0]**

5.1 Dãy (Arrays)



Operators	Associativity	Type
`() [] . ++ --`	left to right	highest (unary postfix)
`++ -- + - ! (type)`	right to left	unary (unary prefix)
`* / %`	left to right	multiplicative
`+ -`	left to right	additive
`< <= > >=`	left to right	relational
`== !=`	left to right	equality
`&`	left to right	boolean logical AND
`^`	left to right	boolean logical exclusive OR
`|`	left to right	boolean logical inclusive OR
`&&`	left to right	logical AND
`||`	left to right	logical OR
`?:`	right to left	conditional
`= += -= *= /= %=`	right to left	assignment

5.2 Khai báo và tạo dãy

- Lập trình viên phải xác định kiểu dữ liệu của các phần tử trong dãy
- Toán tử **new** dùng để cấp phát linh động số phần tử trong dãy
- Việc khai báo và khởi tạo dãy không nhất thiết phải trong cùng một câu lệnh
- Trong các dãy của kiểu giá trị, mỗi phần tử chứa một giá trị của kiểu khai báo
- Trong các dãy của kiểu tham chiếu, mỗi phần tử làm một tham chiếu đến một đối tượng của kiểu dữ liệu khai báo

5.2 Khai báo và tạo dãy

- Toán tử **new** có thể được dùng để xác định có bao nhiêu phần tử mà một dãy sẽ chứa
- Dãy có thể được khởi tạo từ các danh sách khởi tạo
  - Cấp phát vùng nhớ cho dãy – số lượng phần tử trong danh sách khởi tạo sẽ xác định độ lớn của dãy
  - Các phần tử trong dãy được khởi tạo với các giá trị trong danh sách khởi tạo

5.3 Ví dụ về sử dụng dãy

- Khai báo và khởi tạo dãy (Fig. 8.2)
- Sử dụng bộ khởi tạo dãy (Fig. 8.3)
- Tính giá trị được lưu trong mỗi phần tử của dãy (Fig. 8.4)
- Tính tổng các phần tử của dãy (Fig. 8.5)
- Sử dụng Bar Chart để hiển thị dữ liệu của dãy ở dạng đồ họa (Fig. 8.6)
- Sử dụng phần tử của dãy làm biến đếm (Fig. 8.7)
- Sử dụng dãy để phân tích kết quả khảo sát (Fig. 8.8)

5.4 Câu lệnh `foreach`

- Cấu trúc lặp **`foreach`** được sử dụng để lặp đi lặp lại thông qua các giá trị trong các cấu trúc dữ liệu như dãy
- Không có biến đếm
- Có một biến được dùng để biểu diễn cho giá trị của mỗi phần tử
- *Ví dụ: Fig. 8.12*

5.5 Truyền dãy và phần tử của dãy vào phương thức

- Việc truyền dãy như là thông số cho các phương thức bằng cách xác định tên dãy
- Dãy được truyền theo kiểu tham chiếu
- Các phần tử riêng biệt của dãy được truyền theo giá trị
- Các biến lưu trữ đối tượng thật ra là tham chiếu đến các đối tượng đó
- Mỗi tham chiếu là một vùng nhớ trong đó bản thân đối tượng được lưu trữ
- *Ví dụ: Fig. 8.13*

5.6 Truyền dãy bằng trị và bằng tham chiếu

- Truyền tham trị cho phương thức
  - Tạo ra một phiên bản của biến
  - Bất kỳ sự thay đổi có liên quan đến phiên bản này đều không tác động đến biến gốc
- Truyền tham chiếu cho phương thức
  - Tạo ra một phiên bản của tham chiếu đến đối tượng
  - Bất kỳ sự thay đổi có liên quan đến tham chiếu này đều không tác động đến biến gốc
  - Bất kỳ sự thay đổi về nôi dung của đối tượng bên trong phương thức đều tác động đến đối tượng gốc

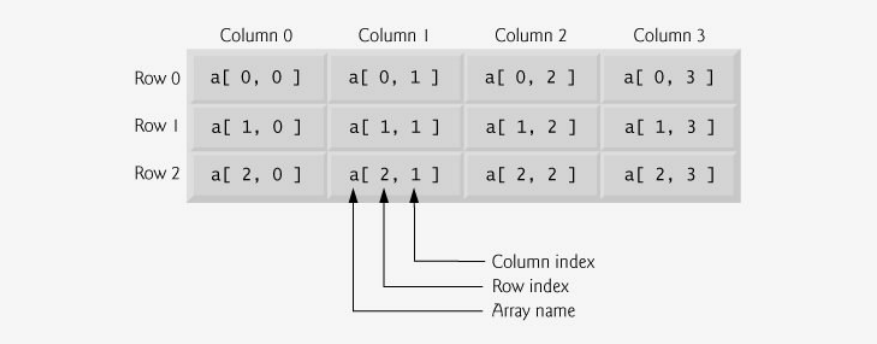
5.6 Truyền dãy bằng trị và bằng tham chiếu

- Từ khóa **ref** được dùng để truyền tham số cho phương thức theo tham chiếu
  - Kiểu giá trị của biến không được nhân bản – thay đổi của biến bên trong phương thức sẽ thay đổi đến biến bên ngoài phương thức
  - Tham chiếu đến đối tượng không được nhân bản – thay đổi của tham chiếu bên trong phương thức sẽ thay đổi đến tham chiếu bên ngoài phương thức
- Các lập trình viên cẩn thận khi sử dụng **ref**
  - Có thể tham chiếu đến null
  - Có thể dẫn đến việc phương thức làm thay đổi biến giá trị và biến tham chiếu theo cách không mong muốn
- *Ví dụ: Fig. 8.14*

5.7 Dãy nhiều chiều

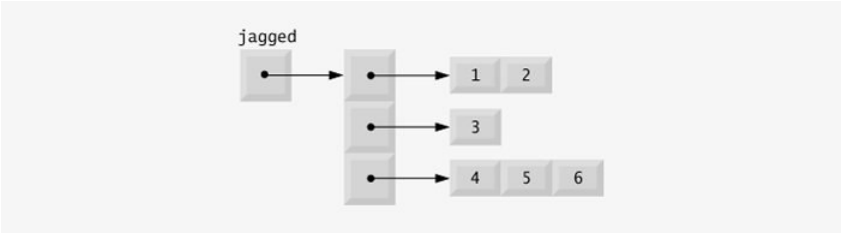
- Cần hai hay nhiều chỉ số để xác định một phần tử cụ thể
- Dãy cần hai chỉ số để xác định một phần tử gọi là dãy hai chiều
- Dãy hình chữ nhật
  - Thường biểu diễn các bảng trong đó các hàng có cùng độ lớn và các cột cũng có cùng độ lớn
  - Chỉ số thứ nhất xác định vị trí hàng và chỉ số thứ hai xác định chỉ số cột của phần tử
- Dãy không đồng nhất (Jagged Arrays)
  - Dãy của dãy (Arrays of arrays)
  - Các dãy tạo thành jagged arrays có thể có độ dài khác nhau

Dãy hình chữ nhật



Ví dụ: Fig. 8.19

Dãy không đồng nhất (Jagged Arrays)



5.8 Danh sách đối số chiều dài thay đổi (Variable-Length Argument Lists)

- Danh sách đối số chiều dài thay đổi cho phép tạo các phương thức nhận số lượng thông số tùy ý.
- Ví dụ: Fig. 8.22

5.9 Sử dụng đối số Command-Line (Using Command-Line Arguments)

- Trong nhiều hệ thống, khả năng truyền thông số từ command line vào một ứng dụng bằng cách sử dụng **`string[]`** (một dãy các chuỗi - array of strings) trong danh sách thông số của phương thức **`Main`**
- Ví dụ: Fig. 8.23

Câu hỏi

- ?
- ?
- ?

LẬP TRÌNH GIAO DIỆN

Chương 3

C# và lập trình hướng đối tượng

1

Chương 3

C# và lập trình hướng đối tượng

Đóng gói và che giấu thông tin
(Encapsulation and data hiding)

2

Mục tiêu

Trong phần này chúng ta sẽ biết:

- Khái niệm đóng gói (encapsulation) và che giấu thông tin (data hiding).
- Khái niệm về sự trừu tượng hóa dữ liệu và các kiểu dữ liệu trừu tượng (ADTs - abstract data types).
- Sử dụng từ khóa **`this`**.
- Sử dụng chỉ mục (indexers) để truy xuất đến các thành phần của một lớp.
- Sử dụng **static variables** và **static methods**.
- Sử dụng **readonly fields**.
- Các vấn đề về quản lý bộ nhớ trong C#.
- Cách tạo một **class library**.
- Sử dụng **internal access modifier**.

Nội dung chính

1. Khái niệm đóng gói (encapsulation)
2. Kiểu dữ liệu trừu tượng (ADTs - abstract data types)
3. Phạm vi của lớp (Class Scope)
4. Điều khiển việc truy xuất đến các thành phần của lớp (Controlling Access to Members)
5. Khởi tạo các đối tượng của lớp: Constructors
6. Public properties
7. Sự cấu thành (Composition)
8. Sử dụng tham chiếu **`this`**
9. Bộ "thu dọn rác" Garbage Collection và Destructors
10. static Class Members
11. const và readonly Members
12. Chỉ mục (Indexers)
13. Data Abstraction và Encapsulation
14. Khả năng sử dụng lại phần mềm (Software Reusability)
15. Truy xuất nội (Internal Access)
16. Class View và Object Browser
17. Cách tạo class library

1. Khái niệm đóng gói (encapsulation)

- Các lớp đối tượng đóng gói dữ liệu (data) và các phương thức (methods)
- Các đối tượng có thể che giấu việc hiện thực đối với các đối tượng khác (che giấu thông tin - information hiding)
- Methods: các khối trong chương trình
- User-defined type: class được viết bởi lập trình viên
- Class chứa:
  - Data members: member variable hoặc instance variables
  - Methods: thao tác trên data members

2. Kiểu dữ liệu trừu tượng (ADTs - abstract data types)

- Abstract Data Types – che giấu hiện thực từ những Object khác
- Các biến (variables) được định nghĩa bên trong Class nhưng không phải là Method được gọi là biến thực thể (*instance variables)*
- *Member Access Modifiers*
  - *public*: thành phần được truy xuất bất cứ nơi nào trong thực thể của đối tượng hiện có
  - *private :* thành phần chỉ được truy xuất bên trong định nghĩa Class

2. Kiểu dữ liệu trừu tượng (ADTs - abstract data types)

- Access methods: đọc hay cập nhật dữ liệu
- Predicate methods: kiểm tra các điều kiện
- Phương thức xây dựng lớp (Constructor)
  - Xây dựng Object của Class
  - Có thể có đối số
  - Không trả về giá trị
  - Có thể có nhiều constructor trong một class
- Toán tử **new** được sử dụng để khởi tạo thực thể của Class
- **Project < Add Class**: thêm một Class mới vào Project

3. Phạm vi của lớp (Class Scope)

- Tất cả các thành phần (members) của lớp đều có thể được truy xuất bởi các phương thức bên trong lớp đó và có thể được tham chiếu theo tên
- Ở bên ngoài lớp, các thành phần (members) không được tham chiếu theo tên, các public members có thể được tham chiếu theo cách: *referenceName.memberName*
- Che giấu các biến thực thể (instance variables)
  - Instance variables có thể được truy xuất bằng cách sử dụng từ khóa **this**
    Ví dụ: ***this****.hour*.
- Ví dụ: Fig. 9.1

4. Điều khiển việc truy xuất đến các thành phần của lớp (Controlling Access to Members)

- Các **public** methods trong một lớp thể hiện các chức năng mà lớp đó cung cấp cho các client bên ngoài.
- Các Methods chỉ thực hiện một công việc
  - Nếu một method cần thực hiện công việc khác thì phải gọi phương thức khác trợ giúp
  - Client không thể truy xuất đến các phương thức trợ giúp này, do đó chúng được khai báo ở dạng **private**
- Các Data members phải được khai báo dưới dạng **private** với các **public properties** tương ứng giúp cho việc truy xuất dữ liệu được an toàn hơn.
- Ví dụ: Fig. 9.3

5. Khởi tạo các đối tượng của lớp: Constructors

- Các thực thể (Instances) của lớp được khởi tạo bởi các phương thức xây dựng lớp (constructors)
- Constructors sẽ khởi tạo các instance variables của các đối tượng (objects) tương ứng
- Các phương thức xây dựng lớp khác nhau (Overloaded constructors) được sử dụng để cung cấp các cách khởi tạo đối tượng khác nhau
- Nếu constructor không rõ ràng, tất cả data members được khởi tạo theo nguyên tắc:
  - Primitive numeric types -> 0
  - Boolean types -> false
  - Reference types -> null
- Nếu lớp không khai báo constructor, một constructor mặc định sẽ được cung cấp
  - constructor mặc định này không có đối số và cũng không có code
- Ví dụ: Fig. 9.7

6. Public properties

- Các Public properties cho phép các client thực hiện việc:
  - Lấy giá trị từ các private data
  - Thiết lập giá trị cho các private data
- Get accessor
  - Lấy giá trị
  - Điều khiển việc định dạng dữ liệu
- Set accessor
  - Thiết lập giá trị
  - Phải chắc chắn rằng giá trị mới là phù hợp với data member

7. Sự cấu thành (Composition)

- Các đối tượng được tham chiếu như là một Instance Variables của các lớp khác
- Sử dụng lại (Software Reuse) – tham chiếu đến các đối tượng có sẵn thì dễ dàng và nhanh hơn khi viết code lại
- Sử dụng các kiểu dữ liệu người dùng định nghĩa (user-defined types) như là các instance variables
- Ví dụ: Fig. 9.9

8. Sử dụng tham chiếu `this`

- Tất cả các đối tượng có thể tham chiếu đến chính nó bằng cách sử dụng từ khóa **this**.
- Thường được sử dụng để phân biệt giữa một biến trong một phương thức với thực thể (instance variables) của một đối tượng.
- Ví dụ: Fig. 9.4

9. Bộ "thu dọn rác" Garbage Collection và Destructors

- Toán tử **new** được sử dụng để cấp phát vùng nhớ khi tạo một đối tượng mới
- Khi các đối tượng không còn được tham chiếu đến nữa, CLR sẽ thực hiện việc "thu dọn rác" (garbage collection)
- Garbage collection giúp tránh rò rỉ bộ nhớ (do các vùng nhớ không còn sử dụng nữa chưa được phục hồi)
- Việc quản lý và cấp phát các tài nguyên (kết nối đến cơ sở dữ liệu, truy xuất tập tin,…) phải được các lập trình viên kiểm soát rõ ràng.

9. Bộ "thu dọn rác" Garbage Collection và Destructors

- Sử dụng *finalizer*s kết hợp với garbage collector để giải phóng các tài nguyên và bộ nhớ.
- Trước khi garbage collector phục hồi vùng nhớ của một đối tượng, nó phải gọi finalizer của đối tượng đó
- Mỗi một lớp chỉ có một finalizer (còn được gọi là phương thức hủy đối tượng destructor)
- Tên của destructor bắt đầu bằng ký tự ~, sau đó là tên lớp tương ứng
- Destructors không có đối số

10. static Class Members

- Mỗi đối tượng của một lớp có một bản sao của các instance variables tương ứng
- Đôi khi hữu dụng đối với tất cả các thực thể (instances) của lớp đó khi dùng chung một bản sao của biến
- Khai báo biến có sử dụng từ khóa **static** chỉ để tạo một bản sao của biến đó tại một thời điểm (dùng chung cho tất cả các đối tượng cùng loại)
- Có thể sử dụng các khai báo phạm vi cho **static** variables (**public**, **private**,...)
- Ví dụ: Fig. 9.12

11. const và readonly Members

- Khai báo các thành phần constant members (giá trị không thay đổi) bằng cách sử dụng từ khóa **const**
- Thành phần được khai báo **const** thì hoàn toàn **static**
- **const** members phải được khởi tạo ngay khi khai báo
- Sử dụng từ khóa **readonly** để khai báo các thành phần sẽ được khởi tạo trong constructor nhưng không thay đổi sau đó.
- Ví dụ: Fig. 9.14

12. Chỉ mục (Indexers)

- Đôi khi một lớp đóng gói dữ liệu như là một danh sách các phần tử
- Indexers là các đặc tính (special properties) cho phép truy xuất đến các dữ liệu trong một lớp dưới dạng dãy (array-style)
- Indexers có thể được định nghĩa để chấp nhận cả hai dạng chỉ số integer và non-integer
- Được định nghĩa bằng từ khóa **this**
- Khi sử dụng indexers, các lập trình viên thực hiện giống như với arrays, với các **get** và **set** accessors
- Ví dụ: Fig. 9.5

13. Data Abstraction và Information Hiding

- Các lớp phải che giấu các chi tiết hiện thực bên trong
- Stacks
  - Last-in, first-out (LIFO)
- Queues
  - Giống như hàng đợi
  - First-in, first-out (FIFO)

14. Khả năng sử dụng lại phần mềm (Software Reusability)

- Framework Class Library (FCL) chứa hàng nghìn lớp tiền định nghĩa (predefined classes)
- Các lớp FCL có thể được sử dụng bất kỳ khi nào
  - Không có lỗi
  - Tối ưu
  - Tài liệu hướng dẫn tốt
- Thuận lợi trong việc sử dụng lại mã trong mô hình phát triển nhanh ứng dụng (RAD - Rapid Application Development)

15. Truy xuất nội (Internal Access)

- Các lớp có thể được khai báo ở 2 dạng access modifiers là **`public`** và **`internal`**.
- Nếu không có khai báo về access modifier trong khai báo lớp thì theo mắc định sẽ là **`internal access`**. Điều này cho phép lớp chỉ được sử dụng bởi tất cả các đoạn mã trong cùng một bộ phận (**`assembly`**).
- Bên trong của cùng một bộ phận như class thì được xem là **`public access`**. Tuy nhiên, nếu một class library được tham chiếu bởi một ứng dụng thì các lớp bên trong thư viện này sẽ không thể truy cập được từ các đoạn mã của ứng dụng.
- Tương tự như vậy đối với các methods, instance variables và các members khác của class.
- Ví dụ: Fig. 9.20

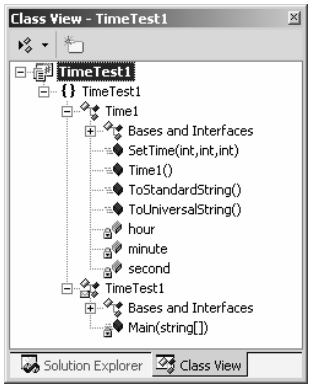
16. Class View và Object Browser

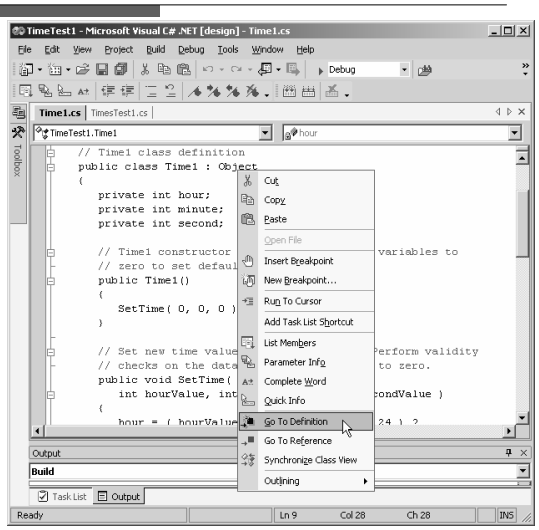
- **Class View** và **Object Browser** là các chức năng của Visual Studio nhằm làm thuận tiện việc thiết kế các ứng dụng hướng đối tượng
- **Class View**
  - Hiển thị tất cả các variables và methods của tất cả các class trong project
  - Hiển thị dưới dạng cấu trúc hình cây
  - + tại các node cho phép mở rộng
  - - tại các node cho phép thu gọn
  - Chọn trên thực đơn lệnh để hiển thị: **View < Class View**

16 Class View và Object Browser

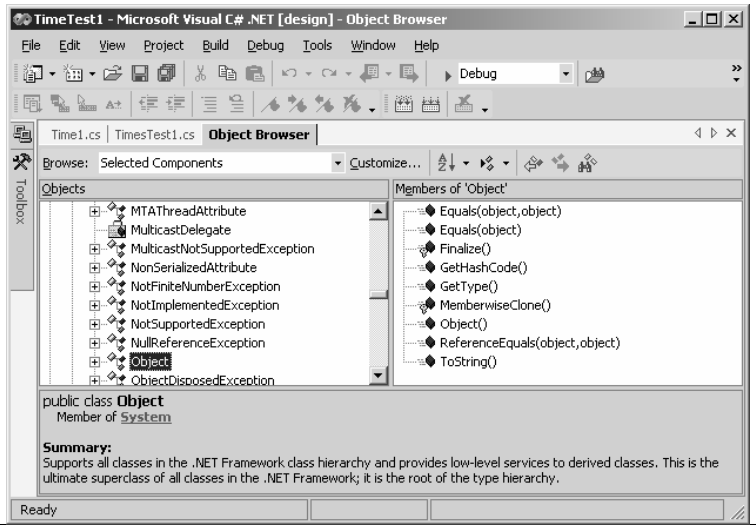
- **Object Browser**
  - Hiển thị danh sách các class trong một library
  - Giúp những người phát triển hệ thống (developer) nghiên cứu về các chức năng của một class cụ thể
  - Để hiển thị **Object Browser**, chọn bất kỳ .NET FCL method và chọn **Go To Definition**

16. Class View và Object Browser





16. Class View và Object Browser



17. Cách tạo class library

- Các bước tạo class library để sử dụng lại như sau :
  1. Khai báo một **`public`** class. Nếu class không phải **`public`**, nó chỉ được sử dụng bởi các lớp trong cùng bộ phận (assembly).
  2. Chọn một tên namespace và khai báo **`namespace`** trong tập tin chứa source-code để khai báo lớp có thể sử dụng lại.
  3. Biên dịch class vào trong một class library.
  4. Thêm một tham chiếu đến class library trong một ứng dụng.
  5. Chỉ định **`using`** cho namespace.
- Ví dụ: Fig. 9.16

Câu hỏi

- ?
- ?
- ?

Chương 3

C# và lập trình hướng đối tượng

Thừa kế (Inheritance)

Mục tiêu

Trong phần này chúng ta sẽ biết:

- Cách để thừa kế giúp tăng khả năng sử dụng lại phần mềm.
- Khái niệm lớp cơ sở (base classes) và lớp dẫn xuất (derived classes).
- Tạo một derived class thừa kế các attributes và behaviors từ một base class.
- Sử dụng access modifier **`protected`** để cho các methods của derived class truy xuất được các members của base class.
- Truy xuất đến các members của base class bằng từ khóa **`base`**.
- Cách sử dụng constructors trong cơ chế thừa kế.
- Các phương thức của class **`Object`**, base class trực tiếp hay gián tiếp của tất cả các lớp.

Nội dung chính

1. Khái niệm thừa kế
2. Base Classes và Derived Classes
3. **`protected`** và **`internal`** Members
4. Mối quan hệ giữa Base Classes và Derived Classes
5. Constructors và Destructors trong Derived Classes
6. Công nghệ phần mềm với thừa kế
7. Class **`object`**

1. Khái niệm thừa kế

- Các class được tạo ra từ việc hấp thu các methods và variables của một class sẵn có. Điều này cho phép các đối tượng chia sẻ hay mở rộng các đặc tính sẵn có mà không phải tiến hành định nghĩa lại.
- Lớp kế thừa được gọi là lớp con hay còn gọi là lớp dẫn xuất (derived class) vì nó kế thừa các methods và variables từ lớp cha hay con được gọi là lớp cơ sở (base class)
- Lớp kế thừa có thể hiện thực thêm các methods và variables để gia tăng khả năng của chính nó.
- Các đối tượng của derived class là đối tượng của base class, nhưng đối tượng của base class không phải là đối tượng của derived class.
- Quan hệ "Is a": đối tượng của derived class có thể được đối xử như đối tượng của base class
- Quan hệ "Has a": đối tượng của lớp này là một thành phần tham chiếu đối tượng của lớp khác
- Một derived class chỉ có thể truy xuất các thành phần non-private của base class trừ khi nó thừa kế các accessor methods

2. Base Classes và Derived Classes

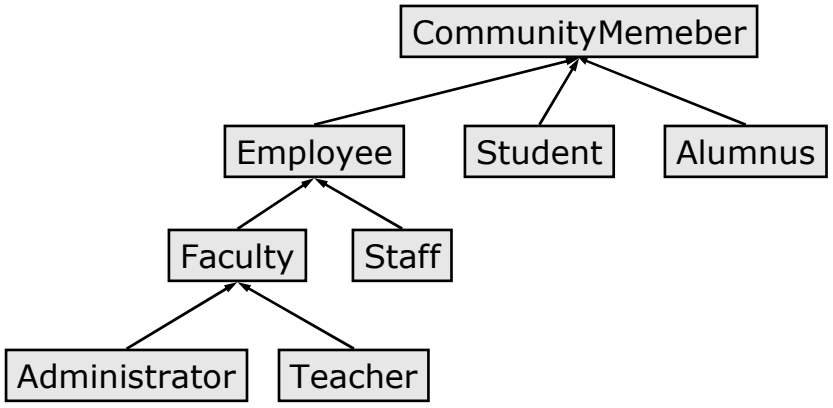
- Một object thường là object của lớp khác
- Mỗi derived-class là một object của base class của nó
- Hình thức thừa kế thể hiện dưới dạng một cây phân cấp
- Để xác định class one được dẫn xuất từ class two:
  - **class one : two**
- Sự cấu thành (Composition):
  - Thể hiện bằng mối quan hệ "has a"
- Các Constructors không được thừa kế

2. Base Classes và Derived Classes

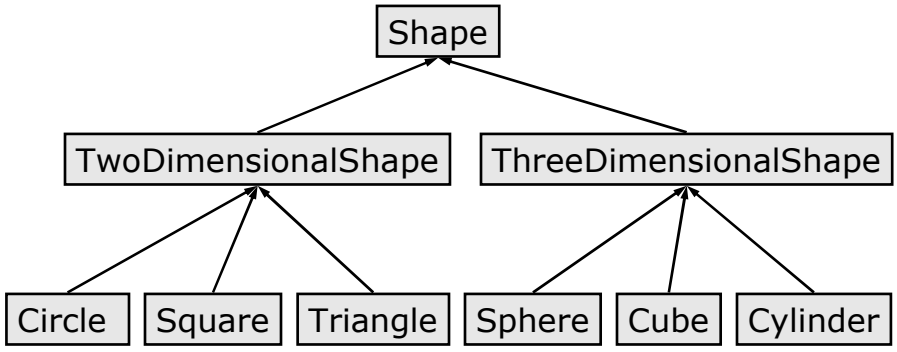
Base class	Derived classes
Student	GraduateStudent UndergraduateStudent
Shape	Circle Triangle Rectangle
Loan	CarLoan HomeImprovementLoan MortgageLoan
Employee	FacultyMember StaffMember
Account	CheckingAccount SavingsAccount

Fig. 9.1 Inheritance examples.

2. Base Classes và Derived Classes



2. Base Classes và Derived Classes



Ví dụ: Fig. 10.4 -> Fig. 10.15

3. `protected` và `internal` Members

- **`protected`** members
  - Chỉ có thể được truy xuất bởi base class hoặc bất kỳ lớp được dẫn xuất từ base class
- **`internal`** members:
  - Chỉ có thể được truy xuất bởi lớp khai báo chúng trong cùng bộ phận (assembly)
- Các thành phần Overridden của base class có thể được truy xuất bằng:
  - **base.member**

4. Mối quan hệ giữa Base Classes và Derived Classes

- Sử dụng cây phân cấp để thể hiện mối quan hệ giữa base class và derived classes
- Việc đầu tiên mà một derived class làm là gọi constructor của base class
- Từ khóa **`override`** được dùng để khai báo một method của derived-class hiện thực lại (override) một method của base-class
- Nếu một method của base class sẽ hiện thực lại (override) bởi derived-class, nó phải được khai báo bằng từ khóa **`virtual`**

5. Constructors và Destructors trong Derived Classes

- Việc tạo thực thể của một derived class là do constructor của base class được gọi
- Khi một destructor được gọi, nó thực hiện các công việc của nó và sau đó triệu gọi destructor của base class của nó.
- Ví dụ: Fig. 10.16 -> Fig. 10.18

6. Công nghệ phần mềm với thừa kế

- Có thể tùy biến các derived classes để đạt được yêu cầu bằng cách:
  - Tạo các member variables mới
  - Tạo các methods mới
  - Override base-class members
- .NET Framework Class Library(FCL) cho phép sử dụng lại hoàn toàn thông qua thừa kế

7. Class `object`

- Các method của class **`object`** được tất cả các class thừa kế trực tiếp hoặc gián tiếp

Method	Mô tả
Equals	So sánh 2 đối tượng có tương đương với nhau hay không
Finalize	Thực hiện việc hủy đối tượng
GetHashCode	Trả về giá trị theo hashtable
GetType	Trả về kiểu của lớp đối tượng
MemberwiseClone	Tạo bản sao shallow của thực thể
ReferenceEquals	So sánh 2 đối tượng tham chiếu có tương đương với nhau hay không
ToString	Trả về một chuỗi

Câu hỏi

- ?
- ?
- ?

Chương 3

C# và lập trình hướng đối tượng

Đa hình, giao tiếp & toán tử nạp chồng
(Polymorphism, Interfaces & Operator Overloading)

Mục tiêu

Trong phần này chúng ta sẽ biết:

- Khái niệm về đa hình (polymorphism)
- Sử dụng overridden methods để thực hiện đa hình (polymorphism).
- Phân biệt giữa abstract classes và concrete classes.
- Khai báo abstract methods để tạo abstract classes.
- Cách mà đa hình (polymorphism) làm cho hệ thống có khả năng mở rộng và bảo trì.
- Xác định kiểu của đối tượng (object's type) tại thời điểm thực thi.
- Tạo sealed methods và sealed classes.
- Khai báo và hiện thực các giao tiếp (interfaces).
- Sử dụng overload operators để thao tác trên các objects.

Nội dung chính

1. Khái niệm đa hình (Polymorphism)
2. Ví dụ về đa hình (Polymorphism)
3. Giải thích hành vi đa hình (Demonstrating Polymorphic Behavior)
4. Abstract Classes và Abstract Methods
5. Case Study: Payroll System Using Polymorphism
6. **`sealed`** Methods và **`sealed`** Classes
7. Delegate
8. Case Study: Creating and Using Interfaces
9. Operator Overloading

1. Khái niệm đa hình (Polymorphism)

Cơ chế đa hình (Polymorphism) cho phép:

- Viết các chương trình điều khiển một số lượng lớn các class có liên quan trong một cách hành xử chung.
- Hệ thống dễ dàng mở rộng

1. Khái niệm đa hình (Polymorphism)

- Hệ thống class theo thứ bậc (Class hierarchies)
  - Có thể gán đối tượng của derived-class cho base-class tham chiếu
  - Có thể dễ dàng chuyển đổi (cast) các kiểu với nhau trong một class hierarchy
- Một object của một derived-class có thể được đối xử như một object của base-class của chính nó
  - Dãy các tham chiếu của base-class đề cập đến các đối tượng của nhiều kiểu derived-class
  - Đối tượng của Base-class **không** là đối tượng của bất kỳ derived classes nào của nó

1. Khái niệm đa hình (Polymorphism)

- Sử dụng câu lệnh **`switch`** để xác định kiểu của đối tượng
  - Phân biệt các đối tượng với nhau, thực hiện các hành động thích hợp dựa vào kiểu đối tượng
- Các vấn đề khi sử dụng câu lệnh **`switch`**
  - Lập trình viên quên kiểm tra kiểu đối tượng
  - Lập trình viên quên kiểm tra tất cả các trường hợp có thể xảy ra
  - Khi có kiểu đối tượng mới, lập trình viên quên cập nhật lại cấu trúc **switch**
  - Mỗi khi thêm hoặc xóa trong một lớp đòi hỏi phải cập nhật mỗi câu lệnh **switch**

2. Ví dụ về đa hình (Polymorphism)

base-class **Quadrilateral**

- derived-class **Rectangle**
- derived-class **Square**
- derived-class **Parallelogram**
- derived-class **Trapezoid**
- Method **`perimeter`** cần được triệu gọi trên tất cả các lớp
- Bằng một tham chiếu **Quadrilateral**, C# chọn overriding method thích hợp trong derived-class từ đối tượng được khởi tạo một cách đa hình

(Demonstrating Polymorphic Behavior)

- base-class **SpaceObject** – chứa method **DrawYourself**
  - derived-class **Martian** (hiện thực **DrawYourself**)
  - derived-class **Venutian** (hiện thực **DrawYourself**)
  - derived-class **Plutonian** (hiện thực **DrawYourself**)
  - derived-class **SpaceShip** (hiện thực **DrawYourself**)
- Một screen-manager program chứa một dãy **SpaceObject** chứa các tham chiếu đến các của các đối tượng được dẫn xuất từ **SpaceObject**
- Để làm mới screen, screen-manager gọi method **DrawYourself** trên mỗi đối tượng của dãy trên
- Một chương trình đa hình gọi phiên bản thích hợp của **DrawYourself** trên mỗi đối tượng một cách đa hình dựa trên kiểu của đối tượng

4. Abstract Classes và Abstract Methods

- Abstract classes
  - Không tạo thực thể
  - Được sử dụng dưới dạng base classes
  - Các định nghĩa lớp không trọn vẹn –> derived classes phải định nghĩa các phần bị thiếu này
  - Có thể chứa các abstract methods và/hoặc abstract properties
    - Không có hiện thực
    - Derived classes phải override các abstract methods và abstract properties thừa kế để cho phép tạo thực thể

4. Abstract Classes và Abstract Methods

- Abstract classes được sử dụng để cung cấp các base classes thích hợp cho các lớp cụ thể (concrete classes) thừa kế
- Abstract base classes thường rất chung chung trong việc định nghĩa đối tượng
- Để định nghĩa một abstract class, sử dụng từ khóa **abstract** trong dòng khai báo class
- Để khai báo một method hoặc property là abstract, sử dụng từ khóa **abstract** trong dòng khai báo; abstract methods và abstract properties không có sự hiện thực

4. Abstract Classes và Abstract Methods

- Concrete classes sử dụng từ khóa **override** để cung cấp sự hiện thực cho tất cả các abstract methods và abstract properties của base-class
- Bất kỳ class nào có chứa một abstract method hoặc abstract property thì phải được khai báo **abstract**
- Dù cho abstract classes không tạo được thực thể, chúng ta có thể sử dụng tham chiếu abstract class để giao việc tạo thực thể cho bất kỳ concrete class nào được dẫn xuất từ abstract class đó.

5. Case Study: Payroll System Using Polymorphism

- Base-class **Employee**
  - **abstract**
  - **abstract** method **Earnings**
- Các Class dẫn xuất từ Employee
  - **Boss**
  - **CommissionWorker**
  - **PieceWorker**
  - **HourlyWorker**
- Tất cả các derived-class đều hiện thực phương thức **Earnings**
- Chương trình sử dụng tham chiếu **Employee** để giao việc tạo thực thể cho các Class dẫn xuất
- Cơ chế đa hình gọi đúng phiên bản của **Earnings**

6. `sealed` Classes và `sealed` Methods

- `sealed` là một từ khóa trong C#
- `sealed` methods không cho phép overridden trong derived class
  - Methods được khai báo **static** và **private**, hiển nhiên là `sealed`
- **sealed** classes không có bất kỳ derived-classes nào
- Việc tạo **sealed** classes có thể cho phép tối ưu thời gian thực thi (runtime optimizations)
  - Ví dụ: việc gọi **virtual** method có thể được chuyển thành việc goị non-**virtual** method

7. Case Study: Tạo và sử dụng giao tiếp (Interfaces)

- Interfaces xác định các dịch vụ chung (methods và properties) để cho các class phải hiện thực
- Interfaces cung cấp sự hiện thực không theo mặc định (khác với abstract classes) cho phép cung cấp nhiều kiểu hiện thực (thừa kế) hơn
- Interfaces được định nghĩa bằng từ khóa **interface** với cú pháp giống như khai báo thừa kế (*ClassName* : *InterfaceName*)

7. Case Study: Tạo và sử dụng giao tiếp (Interfaces)

- Các class có thể hiện thực nhiều interface (mỗi interface được cách nhau bằng một dấu phẩy)
- Khi class hiện thực một interface, nó phải hiện thực mọi method và property trong định nghĩa interface
- Ví dụ: interface **IAge** trả về thông tin về tuổi của đối tượng (object's age)
  - Có thể sử dụng cho các đối tượng như: people, cars, trees…

8. Delegates

- Đôi khi cần phải truyền các method như là các đối số cho các method khác
- Delegates là một tập hợp các tham chiếu đến các method
- Các Delegate object có thể được truyền cho các method; sau đó các method này có thể triệu gọi các method do delegate objects chuyển đến
- Delegates chứa một method gọi là *singlecast delegates* và được tạo và dẫn xuất từ class **Delegate**
- Delegates chứa nhiều method gọi là *multicast delegates* và được tạo và dẫn xuất từ class **MulticastDelegate**

8. Delegates

- Delegates phải được khai báo trước khi sử dụng
- Một khai báo delegate phải xác định các thông số và kiểu trả về của các method mà delegate có thể chuyển đến
- Các method có thể được chuyển đến bởi một delegate phải có cùng chữ ký (signature) như delegate
- Delegate instances có thể được tạo khi chuyển đến method(s)
- Mỗi lần delegate instance được tạo, method mà nó chuyển đến phải được gọi

9. Operator Overloading

- C# có chứa nhiều operators (+, -, *, /) định nghĩa cho một số primitive types
- Hữu dụng khi sử dụng các operator với các user-defined types (đối với một số class phức tạp)
- Ký hiệu operator thường quá trực giác (intuitive) khi gọi method
- C# cho phép các lập trình viên nạp chồng toán tử (overload operators) để làm cho chúng thích hợp hơn trong ngữ cảnh sử dụng

9. Operator Overloading

- Các method định nghĩa các hành động cho overloaded operator
- Chúng thể hiện dưới dạng:
  public static *ReturnType* **operator** *operator-to-be-overloaded(arguments)*
- Các method này phải được khai báo là **public** và **static**
- Kiểu trả về như là kết quả của việc ước lượng của operation
- Từ khóa **operator** theo sau kiểu trả về nhằm xác định rằng method này định nghĩa một operator overload
- The last piece of information is the operator to be overloaded (such as +, -, *, etc.)
- If the operator is unary, one argument must be specified, if the operator is binary, then two, etc.

Câu hỏi

- ?
- ?
- ?