Ngôn ngữ Lập trình C++

Chương I - Giới thiệu ngôn ngữ C++

Nội dung chính

- Mã máy, Hợp ngữ, và ngôn ngữ bậc cao
- Một số ngôn ngữ lập trình bậc cao
- Lịch sử C và C++
- Hệ thống và môi trường lập trình C++
- Giới thiệu về C++
  - ví dụ về chương trình C++ đơn giản
  - khái niệm biến
  - vào ra dữ liệu
  - các phép toán số học
  - ra quyết định - các phép toán quan hệ

1.1 Mã máy, Hợp ngữ, và Ngôn ngữ bậc cao

1. Mã máy (machine language)

- Là ngôn ngữ duy nhất máy tính trực tiếp hiểu được, là "ngôn ngữ tự nhiên" của máy tính
- Được định nghĩa bởi thiết kế phần cứng, phụ thuộc phần cứng
- Gồm các chuỗi số, => chuỗi các số 0 và 1
- Dùng để lệnh cho máy tính thực hiện các thao tác cơ bản, mỗi lần một thao tác
- Nặng nề, khó đọc đối với con người
- Ví dụ:

```
+1300042774
+1400593419
+1200274027
```


1.1 Mã máy, Hợp ngữ, và Ngôn ngữ bậc cao

2. Hợp ngữ (assembly)

- Những từ viết tắt kiểu tiếng Anh, đại diện cho các thao tác cơ bản của máy tính
- Dễ hiểu hơn đối với con người
- Máy tính không hiểu
  - Cần đến các chương trình dịch hợp ngữ (assembler) để chuyển từ hợp ngữ sang mã máy
- Ví dụ:

```
LOAD    BASEPAY
ADD     OVERPAY
STORE   GROSSPAY
```

1.1 Mã máy, Hợp ngữ, và Ngôn ngữ bậc cao

3. Các ngôn ngữ bậc cao (high-level languages)
   - Tương tự với tiếng Anh, sử dụng các ký hiện toán học thông dụng
   - Một lệnh thực hiện được một công việc mà hợp ngữ cần nhiều lệnh để thực hiện được.
   - Ví dụ:

```
grossPay = basePay + overTimePay
```

   - Các chương trình dịch (compiler) để chuyển sang mã máy
   - Các chương trình thông dịch (interpreter program) trực tiếp chạy các chương trình viết bằng ngôn ngữ bậc cao.
     - Chậm hơn
     - Thuận tiện khi đang phát triển chương trình

1.2 Một số ngôn ngữ lập trình bậc cao

- **FORTRAN**
  - FORmula TRANslator (1954-1957: IBM)
  - Tính toán toán học phức tạp, thường dùng trong các ứng dụng khoa học và kỹ thuật
- **COBOL**
  - COmmon Business Oriented Language (1959)
  - Thao tác chính xác và hiệu quả đối với các khối lượng dữ liệu lớn,
    - Các ứng dụng thương mại
- **Pascal**
  - Tác giả: Niklaus Wirth
  - Dùng trong trường học.
- **Java**
  - Tác giả: Sun Microsystems (1991)
  - Ngôn ngữ điều khiển theo sự kiện (event-driven), hoàn toàn hướng đối tượng, tính khả chuyển (portable) rất cao.
  - Các trang Web với nội dung tương tác động
  - Phát triển các ứng dụng quy mô lớn

1.2 Một số ngôn ngữ lập trình bậc cao

- BASIC
  - Beginner's All-Purpose Symbolic Instruction Code
  - Từ giữa những năm1960
- Visual Basic
  - GUI, xử lý sự kiện (event handling), sử dụng Win32 API, lập trình hướng đối tượng (object-oriented programming), bắt lỗi (error handling)
- Visual C++
  - C++ của Microsoft và mở rộng
    - Thư viện của Microsoft (Microsoft Foundation Classes -MFC)
    - Thư viện chung
      - GUI, đồ họa, lập trình mạng, đa luồng (multithreading), …
      - Dùng chung giữa Visual Basic, Visual C++, C#
- C#
  - Bắt nguồn từ C, C++ và Java
  - Ngôn ngữ điều khiển theo sự kiện (event-driven), hoàn toàn hướng đối tượng, ngôn ngữ lập trình trực quan (visual programming language)

1.3 Lịch sử ngôn ngữ C và C++

- C
  - Dennis Ritchie (Bell Laboratories)
  - Là ngôn ngữ phát triển của hệ điều hành UNIX
  - Độc lập phần cứng => có thể viết các chương trình khả chuyển
  - Chuẩn hóa năm 1990 – ANSI C
  - Kernighan & Ritchie "The C Programming Language", 2nd, 1988
- C++
  - Là mở rộng của C
  - Đầu những năm 1980: Bjarne Stroustrup (phòng thí nghiệm Bell)
  - Cung cấp khả năng lập trình hướng đối tượng.
  - Ngôn ngữ lai
    - Lập trình cấu trúc kiểu C
    - Lập trình hướng đối tượng
    - Cả hai
- Có cần biết C trước khi học C++?

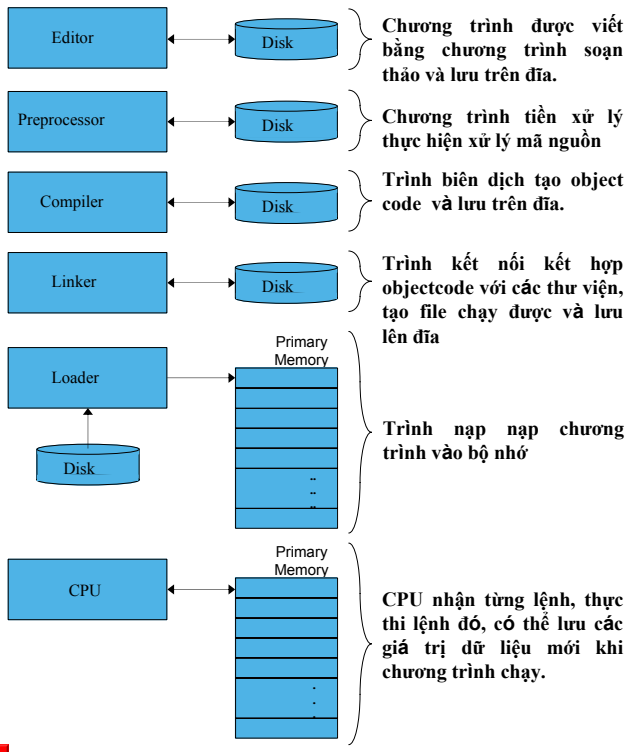
1.4 Hệ thống C++

- Môi trường phát triển chương trình (Program-development environment)
- Ngôn ngữ
- Thư viện chuẩn (C++ Standard Library)

1.4 Môi trường cơ bản cho lập trình C++

Các giai đoạn của chương trình C++:

1. Soạn thảo - Edit
2. Tiền xử lý - Preprocess
3. Biên dịch - Compile
4. Liên kết - Link
5. Nạp - Load
6. Chạy - Execute



1.4 Môi trường cơ bản cho lập trình C++

- Soạn thảo
  - File có kiểu mở rộng *.cpp, *cxx, *.cc, *.C
  - Unix/Linux: vi, emacs
  - MS.Windows: các môi trường soạn thảo tích hợp: Dev-cpp, Microssoft Visual C++, Borland C++ Builder, …
    - Chú ý mức độ hỗ trợ C++ chuẩn – ANSI/ISO C++

Ví dụ 1: Hello World!

```
1   /* A first program in C++.
2        Print a line of text to standard output */
3   #include <iostream>
4
5   // function main begins program execution
6   int main()
7   {
8      std::cout << "Hello World!\n";
9
10     return 0;   // indicate that program ended successfully
11
12  } // end function m
```

```
Hello World!
```

Chú thích

hàm **main** trả về một giá trị kiểu số nguyên.

Định hướng tiền xử lý (preprocessor directive) để khai báo sử dụng thư viện ra/vào chuẩn **<iostream>**.

Viết một dòng ra output chuẩn (màn hình)

hàm **main** xuất hiện đúng một lần trong mỗi chương trình C++.

Ngoặc trái **{** bắt đầu thân hàm.

Các lệnh kết thúc bằng dấu chấm phẩy **;**

Tương ứng, ngoặc phải **}** kết thúc thân hàm.

Từ khóa **return** là một cách thoát khỏi hàm; giá trị **0** được trả về có nghĩa chương trình kết thúc thành công.

1.5 Các thành phần cơ bản
Chú thích và định hướng tiền xử lý

- Chú thích - comment

```
// A first program in C++.
```

  - Làm tài liệu cho các chương trình
  - Làm chương trình dễ đọc dễ hiểu hơn
  - được trình biên dịch (compiler) bỏ qua
  - 1 dòng chú thích bắt đầu với **//**

- Các định hướng tiền xử lý - directive

```
#include <iostream>
```

  - Được xử lý ngay trước khi biên dịch
  - Bắt đầu bằng #

Ví dụ 1 - mở rộng 1

fig01_04.cpp

```
1   // Fig. 1.4: fig01_04.cpp
2   // Printing a line with multiple statements.
3   #include <iostream>
4
5   // function main begins program execution
6   int main()
7   {
8      std::cout << "Welcome ";
9      std::cout << "to C++!\n";
10
11     return 0;   // indicate that program ended successfully
12
13  } // end function main
```

Nhiều dòng lệnh tạo output trên một dòng.

```
Welcome to C++!
```

Ví dụ 1 - mở rộng 2

```
1   // Fig. 1.5: fig01_05.cpp
2   // Printing multiple lines with a single statement
3   #include <iostream>
4
5   // function main begins program execution
6   int main()
7   {
8      std::cout << "Welcome\nto\n\nC++!\n";
9
10     return 0;   // indicate that program ended successfully
11
12  } // end function main
```

Dùng ký tự dòng mới **\n** để in trên nhiều dòng.

```
Welcome
to

C++!
```


Ví dụ 2:
Chương trình tính tổng hai số nguyên

```
1   // Fig. 1.6: fig01_06.cpp
2   // Addition program.
3   #include <iostream>
4
5   // function main begins program execution
6   int main()
7   {
8      int integer1;  // first number to be input by user
9      int integer2;  // second number
10     int sum;       // variable in wh
11
12     std::cout << "Enter first integer\n";  // prompt
13     std::cin >> integer1;                  // read an int
14
15     std::cout << "Enter second integer\n"; // prompt
16     std::cin >> integer2;
17
18     sum = integer1 + integer2;
19
20     std::cout << "Sum is " << sum << endl; // print sum
21
22     return 0;   // indicate that program ended successfully
23
24  } // end function main
```

Khai báo các biến nguyên.

Nhập một số nguyên từ input chuẩn, ghi vào biến integer1

endl cho kết quả là một dòng trống.

Tính toán có thể được thực hiện trong lệnh output: Thay cho các dòng 18 và 20:

```
cout << "Sum is " << integer1 + integer2 << endl;
```

```
Enter first integer
45
Enter second integer
72
Sum is 117
```

1.5 Các thành phần cơ bản
Biến chương trình

- Biến - variable: Một nơi trong bộ nhớ, có thể lưu các giá trị thuộc một kiểu nào đó.
- Các kiểu dữ liệu cơ bản
  - **`int`** - số nguyên
  - **`char`** – ký tự
  - **`double`** - số chấm động
  - **`bool`** – các giá trị logic true hoặc false
- Các biến phải được khai báo tên và kiểu trước khi sử dụng

```
int integer1;
int integer2;
int sum;
```

- Có thể khai báo nhiều biến thuộc cùng một kiểu dữ liệu trong một dòng khai báo biến.

```
int integer1, integer2, sum;
```


1.5 Biến chương trình

- Quy tắc đặt tên biến
  - Chuỗi ký tự (chữ cái a..z, A..Z, chữ số 0..9, dấu gạch dưới _)
  - Không được bắt đầu bằng chữ số
  - Phân biệt chữ hoa chữ thường.

  Ví dụ:

  Tên biến hợp lệ: h678h_m2, _adh2, taxPayment…

  Không hợp lệ: áadàn, so chia, 2n, …

1.5 Biến chương trình

- Các khái niệm về bộ nhớ (memory)
  - Mỗi biến tương ứng với một khu trong bộ nhớ máy tính
  - Mỗi biến có tên, kiểu, kích thước, và giá trị
  - Khi biến được gán một giá trị mới, giá trị cũ bị ghi đè
  - Đọc giá trị của các biến trong bộ nhớ không làm thay đổi các biến trong bộ nhớ.

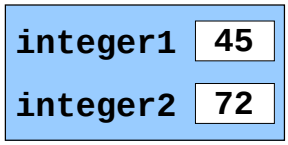
1.5 Biến chương trình

```
std::cin >> integer1;
```

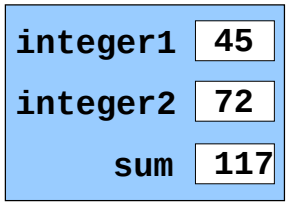
– giả sử người dùng nhập 45


```
std::cin >> integer2;
```

– giả sử người dùng nhập 72




```
sum = integer1 + integer2;
```



1.6 Vào ra dữ liệu

Các đối tượng vào/ra cơ bản

- **`cin`**
  - dòng dữ liệu vào chuẩn - Standard input stream
  - thường là từ bàn phím
- **`cout`**
  - dòng dữ liệu ra chuẩn - Standard output stream
  - thường là màn hình máy tính
- **`cerr`**
  - dòng báo lỗi chuẩn - Standard error stream
  - hiện các thông báo lỗi

1.6 Vào ra dữ liệu

In dòng văn bản ra màn hình

```
std::cout << "Enter first integer\n";  // prompt
```

- Đối tượng ra chuẩn - Standard output stream object
  - **`std::cout`**
  - "nối" với màn hình
  - **`<<`**
    - toán tử chèn vào dòng dữ liệu ra – stream insert operator
    - giá trị bên phải (right operand) được chèn vào dòng dữ liệu ra
- Không gian tên - Namespace
  - **`std::`** có nghĩa là sử dụng tên thuộc "namespace" **`std`**
  - **`std::`** được bỏ qua nếu dùng các khai báo **`using`**
- Escape characters **`\`**
  - đánh dấu các ký tự đặc biệt
    - ví dụ \\, \', \n, \t

1.6 Vào ra dữ liệu

Các chuỗi escape

Chuỗi Escape	Mô tả
`\n`	Dòng mới. Đặt con trỏ màn hình tại đầu dòng tiếp theo.
`\t`	Tab. Di chuyển con trỏ đến điểm dừng tab tiếp theo.
`\r`	Về đầu dòng. Chuyển con trỏ màn hình tới đầu dòng hiện tại; không xuống dòng mới.
`\a`	Chuông. Bật chuông hệ thống.
`\\`	Chéo ngược. Dùng để in một đấu chéo ngược.
`\"`	Nháy kép. Dùng để in một dấu nháy kép.

1.6 Vào ra dữ liệu

Nhập dữ liệu từ thiết bị vào chuẩn

```
std::cin >> integer1;  // read an integer
```

- Đối tượng dòng dữ liệu vào - Input stream object
  - **>>** (toán tử đọc từ dòng dữ liệu vào)
    - được sử dụng với **`std::cin`**
    - đợi người dùng nhập giá trị, rồi gõ phím *Enter* (Return)
    - lưu giá trị vào biến ở bên phải toán tử
      - đổi giá trị được nhập sang kiểu dữ liệu của biến
- **=** (toán tử gán)
  - gán giá trị cho biến
  - toán tử hai ngôi - Binary operator
  - Ví dụ:

```
sum = variable1 + variable2;
```

1.7 Tính toán số học

- Các phép toán số học
  - `*` Phép nhân
  - `/` Phép chia
    - Phép chia với số nguyên lấy thương là số nguyên và bỏ phần dư
      - `7 / 5` cho kết quả `1`
    - Phép chia với số thực cho kết quả là số thực
      - `7.0 / 5.0` cho kết quả `1.4`
  - `%` Phép lấy số dư
    - `7 % 5` cho kết quả `2`

1.7 Tính toán số học

- Các quy tắc ưu tiên - Rules of operator precedence
  - Các phép toán trong ngoặc được tính trước
    - ngoặc lồng nhau
      - các phép toán ở bên trong nhất được tính trước nhất
  - tiếp theo là các phép nhân, chia, và phép lấy số dư
    - các phép toán được tính từ trái sang phải
  - cộng và trừ được tính cuối cùng
    - các phép toán được tính từ trái sang phải

```
1  // Fig. 1.14: fig01_14.cpp
2  // Using if statements, relational
3  // operators, and equality operators.
4  #include <iostream>
5
6  using std::cout;  // program uses cout
7  using std::cin;   // program uses cin
8  using std::endl;  // program uses endl
9
10 // function main begins program execution
11 int main()
12 {
13    int num1;  // first number to
14    int num2;  // second number t
15
16    cout << "Enter two integers, and I will tell you\n"
17         << "the relationships they satisfy: ";
18    cin >> num1 >> num2;   // read two integers
19
20    if ( num1 == num2 )
21       cout << num1 << " is equal to " << num2 << endl;
22
23    if ( num1 != num2 )
24       cout << num1 << " is not equal to " << num2 << endl;
25
```

fig01_14.cpp (1 of 2)

khai báo **using** để sau đó không cần dùng tiền tố **std::**

Khai báo biến.

Có thể viết **cout** và **cin** mà không cần tiền tố **std::**

lệnh **if** kiểm tra xem các giá trị của **num1** và **num2** có bằng nhau không.

Nếu điều kiện là đúng (nghĩa là hai giá trị bằng nhau) thì thực hiện lệnh này.

lệnh **if** kiểm tra xem các giá trị của **num1** và **num2** có khác nhau không.

Nếu điều kiện là đúng (nghĩa là hai giá trị khác nhau) thì thực hiện lệnh này.


```
26    if ( num1 < num2 )
27       cout << num1 << " is less than " << num2 << endl;
28
29    if ( num1 > num2 )
30       cout << num1 << " is greater than " << num2 << end
31
32    if ( num1 <= num2 )
33       cout << num1 << " is less than or equal to "
34            << num2 << endl;
35
36    if ( num1 >= num2 )
37       cout << num1 << " is greater than or equal to "
38            << num2 << endl;
39
40    return 0;   // indicate that program ended successfully
41
42 } // end function main
```

Một lệnh có thể được tách thành nhiều dòng.

fig01_14.cpp output (1 of 2)

```
Enter two integers, and I will tell you
the relationships they satisfy: 22 12
22 is not equal to 12
22 is greater than 12
22 is greater than or equal to 12
```

```
Enter two integers, and I will tell you
the relationships they satisfy: 7 7
7 is equal to 7
7 is less than or equal to 7
7 is greater than or equal to 7
```

fig01_14.cpp
output (2 of 2)

1.8 Ra quyết định: Các phép toán quan hệ

Ký hiệu toán học	Toán tử của C++	Ví dụ điều kiện C++	Ý nghĩa của điều kiện
>	`>`	`x > y`	**x** lớn hơn **y**
<	`<`	`x < y`	**x** nhỏ hơn **y**
≥	`>=`	`x >= y`	**x** lớn hơn hoặc bằng **y**
≤	`<=`	`x <= y`	**x** nhỏ hơn hoặc bằng **y**
=	`==`	`x == y`	**x** bằng **y**
≠	`!=`	`x != y`	**x** khác **y**

1.8 Ra quyết định: Các phép toán quan hệ

- cấu trúc **`if`**
  - Đưa ra quyết định dựa vào kết quả đúng hoặc sai của điều kiện
    - Nếu điều kiện thỏa mãn thì thực hiện tập lệnh S
    - nếu không, tập lệnh S không được thực hiện

```
if ( num1 == num2 )
     cout << num1 << " is equal to " << num2 <<
  endl;
```


1.9 Khai báo using

- Khai báo sử dụng toàn bộ không gian tên
  - using namespace std;
  - Để không cần tiền tố std:: cho mọi tên trong std

```
1   // Fig. 1.4: fig01_04.cpp
2   // Printing a line with multiple statements.
3   #include <iostream>
4
5   using namespace std;
6   // function main begins program execution
7   int main()
8   {
9      cout << "Welcome ";
10     std::cout << "to C++!\n";
11
12     return 0;
13
14  } // end function main
```

1.9 Khai báo using

- Khai báo sử dụng từng tên

```
using std::cout;  // program uses cout
using std::cin;   // program uses cin
using std::endl;  // program uses endl
...
cout << "No need to write std::";
cin >> somevariable;
...
```

Ngôn ngữ lập trình C++

Chương 2 – Các kiểu dữ liệu cơ bản
Các cấu trúc điều khiển

Tài liệu đọc thêm

- Tài liệu đọc thêm cho chương này:
  - Section 2.1. Complete C++ Language Tutorial (CCLT)
  - Day 7. Teach Yourself C++ in 21 Days (TY21)
  - Namespace (Sec.5-2.CCLT) (Không bắt buộc)

Chương 2 – Kiểu dữ liệu và phép toán cơ bản Cấu trúc điều khiển và cấu trúc chương trình

Đề mục

Chương 2 – Kiểu dữ liệu và phép toán cơ bản Cấu trúc điều khiển và cấu trúc chương trình

Đề mục (tiếp theo)

2.1 Các kiểu dữ liệu cơ bản

char	ký tự hoặc số nguyên 8 bit
short	số nguyên 16 bit
long	số nguyên 32 bit
int	số nguyên độ dài bằng 1 word (16 bit hoặc 32 bit)
float	số chấm động 4 byte
double	số chấm động 8 byte
long double	số chấm động 10 byte
bool	giá trị Boolean, true hoặc false
wchar_t	ký tự 2 byte, lưu bảng chữ cái quốc tế

2.2 Các phép toán cơ bản

- phép gán – assignation (=)

```
x = 5;          //x: lvalue, 5: rvalue
```

  - là biểu thức có giá trị là giá trị được gán
- các phép toán số học - Arithmetic operators **(+, -, *, /, %)**
- các phép gán kép - Compound assignation operators **(+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=)**
- phép tăng và phép giảm (++, **--**)

2.2 Các phép toán cơ bản

- các phép quan hệ - relational operators
  (==, !=, >, <, >=, <=)
- các phép toán logic - Logic operators (**!, &&,** ||)
- phép điều kiện - Conditional operator (**?**).
  (7 == 5 ? 4 : 3) cho kết quả **3** do **7** khác **5**.
- các toán tử bit - Bitwise Operators
  (**&,** |, ^, ~, <<, >>).

2.2 Các phép gán tắt

- Các biểu thức gán tắt - Assignment expression abbreviations
  - Phép gán cộng
    `c = c + 3;` viết tắt thành `c += 3;`
- Các lệnh có dạng

```
variable = variable operator expression;
```

có thể được viết lại thành

```
variable operator= expression;
```

- Các phép gán khác

```
d -= 4      (d = d - 4)
e *= 5      (e = e * 5)
f /= 3      (f = f / 3)
g %= 9      (g = g % 9)
```

2.2 Các phép tăng và giảm

- Phép tăng - Increment operator (**++**)
  - có thể được dùng thay cho **`c += 1`**
- Phép giảm - Decrement operator (**--**)
  - có thể được dùng thay cho **`c -= 1`**
- Tăng/giảm trước – Preincrement/Predecrement
  - **`++c`** hoặc **`--c`**
  - Giá trị của biến bị thay đổi, sau đó biểu thức chứa nó được tính giá trị.
  - Biểu thức có giá trị là giá trị của biến sau khi tăng/giảm
- Tăng/giảm sau - Postincrement/Predecrement
  - **`c++`** hoặc **`c--`**
  - Biểu thức chứa biến được thực hiện, sau đó biến được thay đổi.
  - Biểu thức có giá trị là giá trị của biến trước khi tăng/giảm

2.2 Các phép tăng và giảm

- Ví dụ: nếu **`c = 5`**
  - **`cout << ++c;`**
    - **`c`** nhận giá trị **`6`**, rồi được in ra
  - **`cout << c++;`**
    - in giá trị **`5`** (**`cout`** được chạy trước phép tăng).
    - sau đó, **`c`** nhận giá trị **`6`**
- Khi biến không nằm trong biểu thức
  - Tăng trước và tăng sau có kết quả như nhau

```
++c;
cout << c;
```

và

```
c++;
cout << c;
```

là như nhau

```
1  // Fig. 2.14: fig02_14.cpp
2  // Preincrementing and postincrementing.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  // function main begins program execution
9  int main()
10 {
11    int c;                    // declare variable
12
13    // demonstrate postincrement
14    c = 5;                    // assign 5 to c
15    cout << c << endl;        // print 5
16    cout << c++ << endl;      // print 5 then postincrement
17    cout << c << endl << endl; // print 6
18
19    // demonstrate preincrement
20    c = 5;                    // assign 5 to c
21    cout << c << endl;        // print 5
22    cout << ++c << endl;      // preincrement then print 6
23    cout << c << endl;        // print 6
24
25    return 0;   // indicate successful termination
26
27 } // end function main
```

```
5
5
6

5
6
6
```

fig02_14.cpp
(1 of 2)

2.3 Các phép toán logic

- được dùng làm điều kiện trong các vòng lặp và lệnh if
- **&&** (logical **AND**)
  - **true** nếu cả hai điều kiện là **true**

```
if ( gender == 1 && age >= 65 )
   ++seniorFemales;
```

- **||** (logical **OR**)
  - **true** nếu ít nhất một trong hai điều kiện là **true**

```
if ( semesterAverage >= 90 || finalExam >= 90 )
   cout << "Student grade is A" << endl;
```

2.3 Các phép toán logic

- **!** (logical **NOT**, phủ định logic – logical negation)
  - trả về giá trị **true** khi điều kiện là **false**, và ngược lại

```
if ( !( grade == sentinelValue ) )
   cout << "The next grade is " << grade << endl;
```

tương đương với:

```
if ( grade != sentinelValue )
   cout << "The next grade is " << grade << endl;
```


Nhầm lẫn giữa phép so sánh bằng (==) và phép gán (=)

- Lỗi thường gặp
  - Thường không tạo lỗi cú pháp (syntax error)
- Các khía cạnh của vấn đề
  - biểu thức có giá trị có thể được dùng làm điều kiện
    - bằng không = false, khác không = true
  - Các lệnh gán cũng tạo giá trị (giá trị được gán)

Nhầm lẫn giữa phép so sánh bằng (==) và phép gán (=)

- Ví dụ

```
if ( 4 == payCode )
   cout << "You get a bonus!" << endl;
```

  - Nếu mã tiền lương (paycode) là 4 thì thưởng

- Nếu == bị thay bởi =

```
if ( payCode = 4 )
   cout << "You get a bonus!" << endl;
```

  - Paycode được gán giá trị 4 (không cần biết giá trị của paycode trước đó)
  - lệnh gán cho giá trị true (vì 4 khác 0)
  - trường hợp nào cũng được thưởng

Nhầm lẫn giữa phép so sánh bằng (==) và phép gán (=)

- Lvalue
  - là biểu thức có thể xuất hiện tại vế trái của phép gán
  - xác định một vùng nhớ có thể được gán trị (i.e, các biến)
    - `x = 4;`
- Rvalue
  - chỉ xuất hiện bên phải phép gán
  - hằng, các giá trị (literal)
    - không thể viết `4 = x;`
- Lvalue có thể được dùng như các rvalue, nhưng chiều ngược lại là không thể

Viết chương trình

- Trước khi viết chương trình
  - Hiểu kỹ bài toán
  - Lập kế hoạch giải quyết bài toán
- Trong khi viết chương trình
  - Biết lời giải có sẵn cho các bài toán con
  - Sử dụng các nguyên lý lập trình tốt

Thuật toán - Algorithm

- Các bài toán tin học
  - được giải bằng cách thực hiện một chuỗi hành động theo một thứ tự cụ thể
- Thuật toán: một quy trình quyết định
  - Các hành động cần thực hiện
  - Thứ tự thực hiện
  - Ví dụ: cách nấu một món ăn
- Điều khiển của chương trình – Program Control
  - Chỉ ra thứ tự thực hiện các lệnh

Mã giả - Pseudocode

- Mã giả: ngôn ngữ không chính thức được dùng để mô tả thuật toán
  - tương tự với ngôn ngữ hàng ngày
- Không chạy được trên máy tính
  - dùng để mô tả chương trình trước khi viết chương trình
    - dễ chuyển thành chương trình C++
  - chỉ gồm các lệnh chạy
    - không cần khai báo biến

```
Ví dụ:
    tìm số nhỏ hơn trong hai số


1.  nhập 2 số x,y
2.  nếu x>y thì in y ra màn hình
3.  nếu không, in x ra màn hình
```


Các cấu trúc điều khiển - Control Structures Khái niệm

- Thực thi tuần tự - Sequential execution
  - Các lệnh được thực hiện theo thứ tự tuần tự
- Chuyển điều khiển - Transfer of control
  - Lệnh tiếp theo được thực thi *không phải* lệnh tiếp theo trong chuỗi lệnh.
- 3 cấu trúc điều khiển
  - Cấu trúc tuần tự - Sequence structure
    - theo mặc định, chương trình chạy tuần tự từng lệnh
  - Các cấu trúc chọn lựa - Selection structures
    - **`if`**, **`if/else`**, **`switch`**
  - Các cấu trúc lặp - Repetition structures
    - **`while`**, **`do/while`**, **`for`**

Các cấu trúc điều khiển

- Các từ khóa của C++
  - Không thể dùng làm tên biến hoặc tên hàm

C++ Keywords

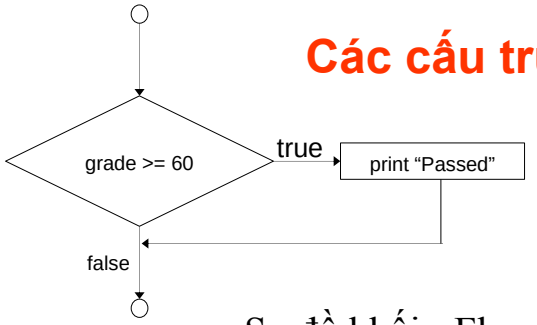
Keywords common to the C and C++ programming languages

auto	break	case	char	const
continue	default	do	double	else
enum	extern	float	for	goto
if	int	long	register	return
short	signed	sizeof	static	struct
switch	typedef	union	unsigned	void
volatile	while			

C++ only keywords

asm	bool	catch	class	const_cast
delete	dynamic_cast	explicit	false	friend
inline	mutable	namespace	new	operator
private	protected	public	reinterpret_cast	
static_cast	template	this	throw	true
try	typeid	typename	using	virtual
wchar_t				

Các cấu trúc điều khiển



- Sơ đồ khối - Flowchart
  - mô tả thuật toán bằng hình vẽ
  - gồm các ký hiệu đặc biệt được nối bằng các mũi tên (flowlines)
  - Hình chữ nhật (ký hiệu hành động)
    - kiểu hành động bất kỳ
  - ký hiệu oval
    - Bắt đầu hoặc kết thúc một chương trình, hoặc một đoạn mã (hình tròn)
- Các cấu trúc điều khiển có đúng 1 đầu vào, 1 đầu ra
  - Kết nối đầu ra của một cấu trúc điều khiển với đầu vào của cấu trúc tiếp theo
  - xếp chồng các cấu trúc điều khiển

Cấu trúc lựa chọn if

- Cấu trúc lựa chọn - Selection structure
  - chọn giữa các tuyến hành động khác nhau
  - ví dụ bằng mã giả:

    If student's grade is greater than or equal to 60

    Print "Passed"
  - Nếu điều kiện thỏa mãn (có giá trị **`true`**)
    - lệnh Print được thực hiện, chương trình chạy tiếp lệnh tiếp theo
  - Nếu điều kiện không thỏa mãn (có giá trị **`false`**)
    - lệnh Print bị bỏ qua, chương trình chạy tiếp
  - Cách viết thụt đầu dòng làm chương trình dễ đọc hơn
    - C++ bỏ qua các ký tự trắng (tab, space, etc.)

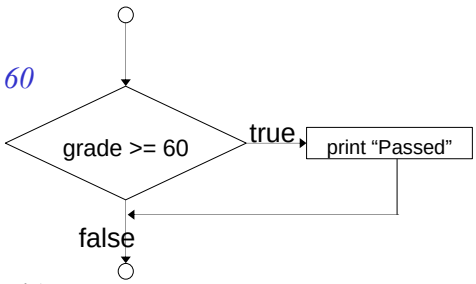
Cấu trúc lựa chọn if

- Dịch sang C++

  If student's grade is greater than or equal to 60

  Print "Passed"

```
if ( grade >= 60 )
   cout << "Passed";
```



- ký hiệu hình thoi (ký hiệu quyết định)
  - đánh đấu chọn lựa cần thực hiện
  - chứa một biểu thức có giá trị true hoặc false
    - kiểm tra điều kiện, đi theo đường thích hợp
- cấu trúc **`if`**
  - Single-entry/single-exit

Một biểu thức bất kỳ đều có thể được sử dụng làm điều kiện cho lựa chọn.

bằng 0 - **`false`**

khác 0 - **`true`**

Ví dụ:

`3 - 4` có giá trị **`true`**

Cấu trúc chọn lựa `if/else`

- **`if`**
  - Thực hiện hành động nếu điều kiện thỏa mãn
- **`if/else`**
  - thực hiện những hành động khác nhau tùy theo điều kiện được thỏa mãn hay không
- mã giả

  if student's grade is greater than or equal to 60
  print "Passed"
  else
  print "Failed"
- mã C++

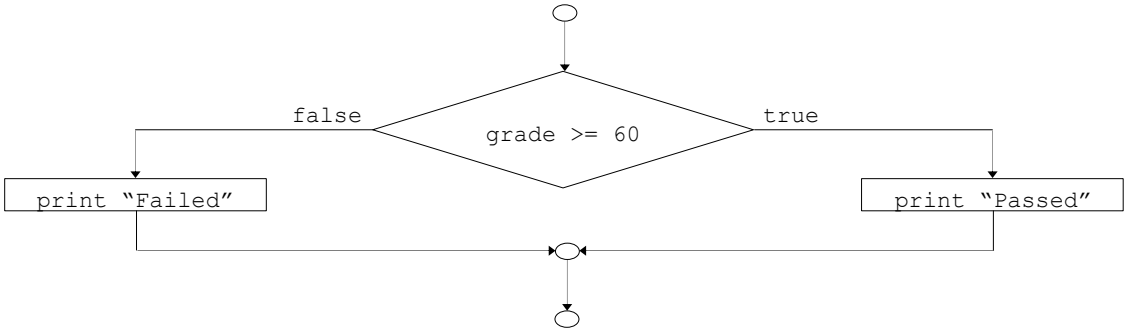
```
if ( grade >= 60 )
   cout << "Passed";
else
   cout << "Failed";
```


Cấu trúc chọn lựa `if/else`

- phép toán điều kiện 3 ngôi (**`?:`**)
  - ba tham số (điều kiện, giá trị nếu **`true`**, giá trị nếu **`false`**)
- mã có thể được viết:

```
cout << ( grade >= 60 ? "Passed" : "Failed" );
```

Condition **Value if true** **Value if false**



Cấu trúc chọn lựa `if/else`

- Các cấu trúc **`if/else`** lồng nhau
  - lệnh này nằm trong lệnh kia, kiểm tra nhiều trường hợp
  - Một khi điều kiện thỏa mãn, các lệnh khác bị bỏ qua

```
if student's grade is greater than or equal to 90
      Print "A"
else
      if student's grade is greater than or equal to 80
        Print "B"
      else
        if student's grade is greater than or equal to 70
          Print "C"
        else
          if student's grade is greater than or equal to 60
            Print "D"
          else
                Print "F"
```


Cấu trúc chọn lựa `if/else`

- Ví dụ

```
if ( grade >= 90 )          // 90 and above
   cout << "A";
else if ( grade >= 80 )     // 80-89
   cout << "B";
else if ( grade >= 70 )     // 70-79
   cout << "C";
else if ( grade >= 60 )     // 60-69
   cout << "D";
else                        // less than 60
   cout << "F";
```

Cấu trúc chọn lựa `if/else`

- lệnh phức – compound statement
  - tập lệnh bên trong một cặp ngoặc

```
if ( grade >= 60 )
   cout << "Passed.\n";
else {
   cout << "Failed.\n";
   cout << "You must take this course again.\n";
}
```

  - nếu không có ngoặc,

```
cout << "You must take this course again.\n";
```

  sẽ luôn được thực hiện
- Khối chương trình - Block
  - tập lệnh bên trong một cặp ngoặc

Cấu trúc lặp while

- Cấu trúc lặp - Repetition structure
  - hành động được lặp đi lặp lại trong khi một điều kiện nào đó còn được thỏa mãn
  - mã giả

    Trong khi vẫn còn tên hàng trong danh sách đi chợ của tôi

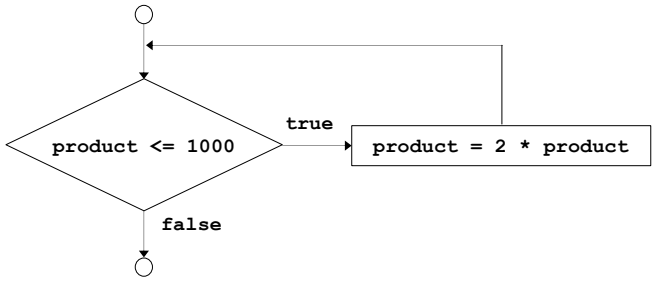
    Mua mặt hàng tiếp theo và gạch tên nó ra khỏi danh sách
  - vòng **`while`** lặp đi lặp lại cho đến khi điều kiện không thỏa mãn

Cấu trúc lặp while

- Ví dụ

```
int product = 2;
while ( product <= 1000 )
   product = 2 * product;
```

- Sơ đồ khối của vòng **while**



Thiết lập thuật toán (Điều khiển lặp bằng con đếm)

- Vòng lặp được điều khiển bằng con đếm (counter)
  - Lặp đến khi con đếm đạt đến giá trị nào đó
- Lặp hữu hạn - Definite repetition
  - số lần lặp biết trước
- Ví dụ

  Một lớp gồm 10 sinh viên làm một bài thi. Cho biết các điểm thi (số nguyên trong khoảng từ 0 đến 100). Tính trung bình điểm thi của lớp.

Thiết lập thuật toán (Điều khiển lặp bằng con đếm)

- Mã giả cho ví dụ:

Đặt tổng bằng 0
Đặt con đếm bằng 1
Trong khi con đếm nhỏ hơn hoặc bằng 10
Nhập điểm tiếp theo
Cộng điểm đó vào tổng
Thêm 1 vào con đến
Đặt trung bình lớp bằng tổng chia cho 10
In trung bình lớp

- Tiếp theo: Mã C++ cho ví dụ trên

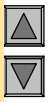


fig02_07.cpp (1 of 2)

```
1  // Fig. 2.7: fig02_07.cpp
2  // Class average program with counter-controlled repetition.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  // function main begins program execution
10 int main()
11 {
12    int total;        // sum of grades input by user
13    int gradeCounter; // number of grade to be entered next
14    int grade;        // grade value
15    int average;      // average of grades
16
17    // initialization phase
18    total = 0;          // initialize total
19    gradeCounter = 1;   // initialize loop counter
20
```

```
21     // processing phase
22     while ( gradeCounter <= 10 ) {       // loop 10 times
23        cout << "Enter grade: ";          // prompt for input
24        cin >> grade;                     // read grade from user
25        total = total + grade;            // add grade to total
26        gradeCounter = gradeCounter + 1;  // increment counter
27     }
28
29     // termination phase
30     average = total / 10;
31
32     // display result
33     cout << "Class average is " << average << endl;
34
35     return 0;   // indicate program ended successfully
36
37  } // end function main
```

fig02_07.cpp (2 of 2)

fig02_07.cpp output (1 of 1)

Con đếm được tăng thêm 1 mỗi lần vòng lặp chạy. Cuối cùng, con đếm làm vòng lặp kết thúc.

```
Enter grade: 98
Enter grade: 76
Enter grade: 71
Enter grade: 87
Enter grade: 83
Enter grade: 90
Enter grade: 57
Enter grade: 79
Enter grade: 82
Enter grade: 94
Class average is 81
```


Thiết lập thuật toán (Điều khiển lặp bằng lính canh)

- Giả sử bài toán trở thành:

  Viết một chương trình tính điểm trung bình của lớp, chương trình sẽ xử lý một số lượng điểm tùy ý mỗi khi chạy chương trình.

  – Số sinh viên chưa biết
  – Chương trình sẽ làm thế nào để biết khi nào thì kết thúc?
- Giá trị canh
  – Ký hiệu "Kết thúc của dữ liệu vào"
  – Vòng lặp kết thúc khi nhập canh
  – Canh được chọn để không bị lẫn với dữ liệu vào thông thường
    - trong trường hợp này là -1

Thiết lập thuật toán (Điều khiển lặp bằng lính canh)

- Thiết kế từ trên xuống, làm mịn từng bước
  - Bắt đầu bằng mã giả cho mức cao nhất

    Tính trung bình điểm thi của lớp
  - Chia thành các nhiệm vụ nhỏ hơn, liệt kê theo thứ tự

    Khởi tạo các biến

    Nhập, tính tổng, và đếm các điểm thi

    Tính và in trung bình điểm thi

Thiết lập thuật toán (Điều khiển lặp bằng lính canh)

- Nhiều chương trình có 3 pha
  - Khởi tạo - Initialization
    - Khởi tạo các biến chương trình
  - Xử lý - Processing
    - Nhập dữ liệu, điều chỉnh các biến trong chương trình
  - Kết thúc - Termination
    - Tính và in kết quả cuối cùng
  - Giúp việc chia nhỏ chương trình để làm mịn từ trên xuống

Thiết lập thuật toán (Điều khiển lặp bằng lính canh)

- Làm mịn pha khởi tạo

  Khởi tạo các biến

  thành

  Khởi tạo tổng bằng 0

  Khởi tạo biến đếm bằng 0

- Xử lý

  Nhập, tính tổng, và đếm các điểm thi

  thành

  Nhập điểm đầu tiên (có thể là canh)

  Trong khi người dùng còn chưa nhập canh

  Cộng điểm vừa nhập vào tổng

  Cộng thêm 1 vào biến đếm điểm

  Nhập điểm tiếp theo (có thể là canh)

Thiết lập thuật toán (Điều khiển lặp bằng lính canh)

- Kết thúc

  Tính và in trung bình điểm thi

  thành

  Nếu con đếm khác 0

  Đặt trung bình bằng tổng chia cho con đếm

  In giá trị trung bình

  Nếu không

  In "Không nhập điểm nào"

- Tiếp theo: chương trình C++

fig02_09.cpp
(1 of 3)

```
1  // Fig. 2.9: fig02_09.cpp
2  // Class average program with sentinel-controlled repetition.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8  using std::fixed;
9
10 #include <iomanip>        // parameterized stream manipulators
11
12 using std::setprecision;  // sets numeric output precision
13
14 // function main begins program execution
15 int main()
16 {
17    int total;          // sum of
18    int gradeCounter;   // number
19    int grade;          // grade value
20
21    double average;     // number with decimal point for average
22
23    // initialization phase
24    total = 0;          // initialize total
25    gradeCounter = 0;   // initialize loop counter
```

Dữ liệu kiểu **double** dùng để biểu diễn số thập phân.

fig02_09.cpp
(2 of 3)

```
26
27    // processing phase
28    // get first grade from user
29    cout << "Enter grade, -1 to end: ";  // prompt for input
30    cin >> grade;                        // read grade from user
31
32    // loop until sentinel value read from user
33    while ( grade != -1 ) {
34       total = total + grade;            // add grade to total
35       gradeCounter = gradeCounter + 1;  // increment counter
36
37       cout << "Enter grade, -1 to end: ";  // prompt for input
38       cin >> grade;                        // read next grade
39
40    } // end while
41
42    // termination phase
43    // if user entered at least one grade ...
44    if ( gradeCounter != 0 ) {
45
46       // calculate average of all grades entered
47       average = static_cast< double >( total ) / gradeCounter;
48
```

static_cast<double>() coi **total** như một **double** tạm thời (casting).
Cần thiết vì phép chia số nguyên bỏ qua phần dư.
gradeCounter là một biến **int**, nhưng nó được nâng lên kiểu **double**.

```
49        // display average with two digits of precision
50        cout << "Class average is " << setprecision( 2 )
51             << fixed << average << endl;
52
53     } // end if part of if/else
54
55     else // if no grades were entered, output appropriate message
56        cout << "No grades were entered" << endl;
57
58     return 0;   // indicate program ended successfully
59
60  } // end function main
```

fig02_09.cpp (3 of 3)

fig02_09.cpp output (1 of 1)

```
Enter grade, -1 to end: 75
Enter grade, -1 to end: 94
Enter grade, -1 to end: 97
Enter grade, -1 to end: 88
Enter grade, -1 to end: 70
Enter grade, -1 to end: 64
Enter grade, -1 to end: 83
Enter grade, -1 to end: 89
Enter grade, -1 to end: -1
Class average is 82.50
```

`fixed` làm số liệu ra được in theo dạng thông thường (không phải dạng ký hiệu khoa học); qui định in cả các chữ số 0 ở sau và in dấu chấm thập phân.

Include **`<iostream>`**

`setprecision(2)` in hai chữ số sau dấu phảy (làm tròn theo độ chính xác quy định).

Các chương trình dùng hàm này phải include **`<iomanip>`**

Các cấu trúc điều khiển lồng nhau

- Phát biểu bài toán

  Một trường có danh sách kết quả thi (1 = đỗ, 2 = trượt) của 10 sinh viên. Viết một chương trình phân tích kết quả thi. Nếu có nhiều hơn 8 sinh viên đỗ thì in ra màn hình dòng chữ *"Tăng tiền học phí".*

- Lưu ý
  - Chương trình xử lý 10 kết quả thi
    - số lần lặp cố định, sử dụng vòng lặp điều khiển bằng biến đếm
  - Có thể sử dụng hai con đếm
    - Một con đếm để đếm số lượng đỗ
    - Một con đếm khác đếm số lương trượt
  - Mỗi kết quả thi chỉ là 1 hoặc 2
    - Nếu không phải 1 thì coi là 2

Các cấu trúc điều khiển lồng nhau

- Phác thảo mức cao nhất - Top level outline

  Analyze exam results and decide if tuition should be raised

- Làm mịn lần một - First refinement

  Initialize variables

  Input the ten quiz grades and count passes and failures

  Print a summary of the exam results and decide if tuition should be raised

- Làm mịn - Refine

  Initialize variables

  to

  Initialize passes to zero

  Initialize failures to zero

  Initialize student counter to one

Các cấu trúc điều khiển lồng nhau

- Refine

  Input the ten quiz grades and count passes and failures

  to

  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

Các cấu trúc điều khiển lồng nhau

- tiếp tục làm mịn

  Print a summary of the exam results and decide if tuition should be raised

  to

  Print the number of passes

  Print the number of failures

  If more than eight students passed

  Print "Raise tuition"

- Program next

fig02_11.cpp (1 of 2)

```
1  // Fig. 2.11: fig02_11.cpp
2  // Analysis of examination results.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  // function main begins program execution
10 int main()
11 {
12    // initialize variables in declarations
13    int passes = 0;           // number of passes
14    int failures = 0;         // number of failures
15    int studentCounter = 1;   // student counter
16    int result;               // one exam result
17
18    // process 10 students using counter-controlled loop
19    while ( studentCounter <= 10 ) {
20
21       // prompt user for input and obtain value from user
22       cout << "Enter result (1 = pass, 2 = fail): ";
23       cin >> result;
24
```

```
25       // if result 1, increment passes; if/else nested in while
26       if ( result == 1 )        // if/else nested in while
27          passes = passes + 1;
28
29       else  // if result not 1, increment failures
30          failures = failures + 1;
31
32       // increment studentCounter so loop eventually terminates
33       studentCounter = studentCounter + 1;
34
35    } // end while
36
37    // termination phase; display number of passes and failures
38    cout << "Passed " << passes << endl;
39    cout << "Failed " << failures << endl;
40
41    // if more than eight students passed, print "raise tuition"
42    if ( passes > 8 )
43       cout << "Raise tuition " << endl;
44
45    return 0;   // successful termination
46
47 } // end function main
```

fig02_11.cpp
(2 of 2)

```
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Passed 6
Failed 4

Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Passed 9
Failed 1
Raise tuition
```

fig02_11.cpp
output (1 of 1)

Những điểm quan trọng về vòng lặp điều khiển bằng con đếm

- vòng lặp điều khiển bằng con đếm đòi hỏi
  - Tên của biến điều khiển(control variable) hay biến đếm (loop counter)
  - Giá trị khởi tạo của biến điều khiển
  - Điều kiện kiểm tra giá trị cuối cùng
  - Tăng/giảm biến đếm khi thực hiện vòng lặp

```
int counter = 1;                // initialization

while ( counter <= 10 ) {       // repetition condition
   cout << counter << endl;     // display counter
   ++counter;                   // increment
}
```


fig02_16.cpp
(1 of 1)

```
1  // Fig. 2.16: fig02_16.cpp
2  // Counter-controlled repetition.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  // function main begins program execution
9  int main()
10 {
11    int counter = 1;              // initialization
12
13    while ( counter <= 10 ) {     // repetition condition
14       cout << counter << endl;   // display counter
15       ++counter;                 // increment
16
17    } // end while
18
19    return 0;   // indicate successful termination
20
21 } // end function main
```

```
1
2
3
4
5
6
7
8
9
10
```

Cấu trúc vòng lặp for

- Dạng tổng quát của vòng **for**

```
for ( khởi_tạo; điều_kiện_lặp; tăng/giảm )
   lệnh
```

- Ví dụ

```
for( int counter = 1; counter <= 10; counter++ )
   cout << counter << endl;
```

  – In các số nguyên từ 1 đến 10

Không có dấu ; ở cuối

fig02_17.cpp
(1 of 1)

```
1  // Fig. 2.17: fig02_17.cpp
2  // Counter-controlled repetition with the for structure.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  // function main begins program execution
9  int main()
10 {
11    // Initialization, repetition condition and incrementing
12    // are all included in the for structure header.
13
14    for ( int counter = 1; counter <= 10; counter++ )
15       cout << counter << endl;
16
17    return 0;   // indicate successful termination
18
19 } // end function main
```

```
1
2
3
4
5
6
7
8
9
10
```

Cấu trúc vòng lặp for

- vòng **for** thường có thể viết được thành vòng **while** tương đương

```
khởi_tạo;
while ( điều_kiện_lặp){
   lệnh
   tăng/giảm biến đếm;
}
```

- Khởi tạo và tăng biến đếm
  - nếu sử dụng nhiều biến đếm, sử dụng dấu phảy để tách

```
for (int i = 0, j = 0;  j + i <= 10; j++, i++)
   cout << j + i << endl;
```



```
1  // Fig. 2.20: fig02_20.cpp
2  // Summation with for.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  // function main begins program execution
9  int main()
10 {
11    int sum = 0;                      // initialize sum
12
13    // sum even integers from 2 through 100
14    for ( int number = 2; number <= 100; number += 2 )
15       sum += number;                 // add number to sum
16
17    cout << "Sum is " << sum << endl;  // output sum
18    return 0;                         // successful termination
19
20 } // end function main
```

```
Sum is 2550
```

Ví dụ sử dụng vòng for

- Chương trình tính lãi kép (compound interest)
- *Một người đầu tư $1000.00 vào một tài khoản tiết kiệm với lãi suất 5%. Giả sử tiền lãi được gộp với vốn trong tài khoảng, tính và in ra số tiền trong tài khoản vào cuối mỗi năm trong vòng 10 năm. Sử dụng công thức sau để tính các khoản tiền đó:*

$$a = p(1+r)^n$$

- p : khoản đầu tư ban đầu (i.e., the principal),
  r : lãi suất hàng năm, (interest rate)
  n : số năm, và
  a : lượng tiền có trong tài khoản (amount on deposit) vào cuối năm thứ n

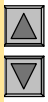


fig02_21.cpp
(1 of 2)

```
1  // Fig. 2.21: fig02_21.cpp
2  // Calculating compound interest.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7  using std::ios;
8  using std::fixed;
9
10 #include <iomanip>
11
12 using std::setw;
13 using std::setprecision;
14
15 #include <cmath>  // enables program to use function pow
16
17 // function main begins program execution
18 int main()
19 {
20    double amount;              // amount on deposit
21    double principal = 1000.0;  // starting principal
22    double rate = .05;          // interest rate
23
```

<cmath> header cần cho hàm **pow** (chương trình sẽ không dịch nếu không có khai báo này).

```
24     // output table column heads
25     cout << "Year" << setw( 21 ) << "Amount on deposit" << endl;
26
27     // set floating-point number format
28     cout << fixed << setprecision( 2 );
29
30     // calculate amount on deposit for each of ten years
31     for ( int year = 1; year <= 10; year++ ) {
32
33        // calculate new amount for specified year
34        amount = principal * pow( 1.0 + rate, year );
35
36        // output one table row
37        cout << setw( 4 ) << year
38             << setw( 21 ) << amount << endl;
39
40     } // end for
41
42     return 0;   // indicate successful termination
43
44  } // end function main
```

Đặt độ rộng của output ít nhất 21 ký tự. Nếu output ít hơn 21 ký tự thì căn phải.

pow(x,y) = x mũ y


```
Year    Amount on deposit
   1              1050.00
   2              1102.50
   3              1157.63
   4              1215.51
   5              1276.28
   6              1340.10
   7              1407.10
   8              1477.46
   9              1551.33
  10              1628.89
```

fig02_21.cpp
output (1 of 1)

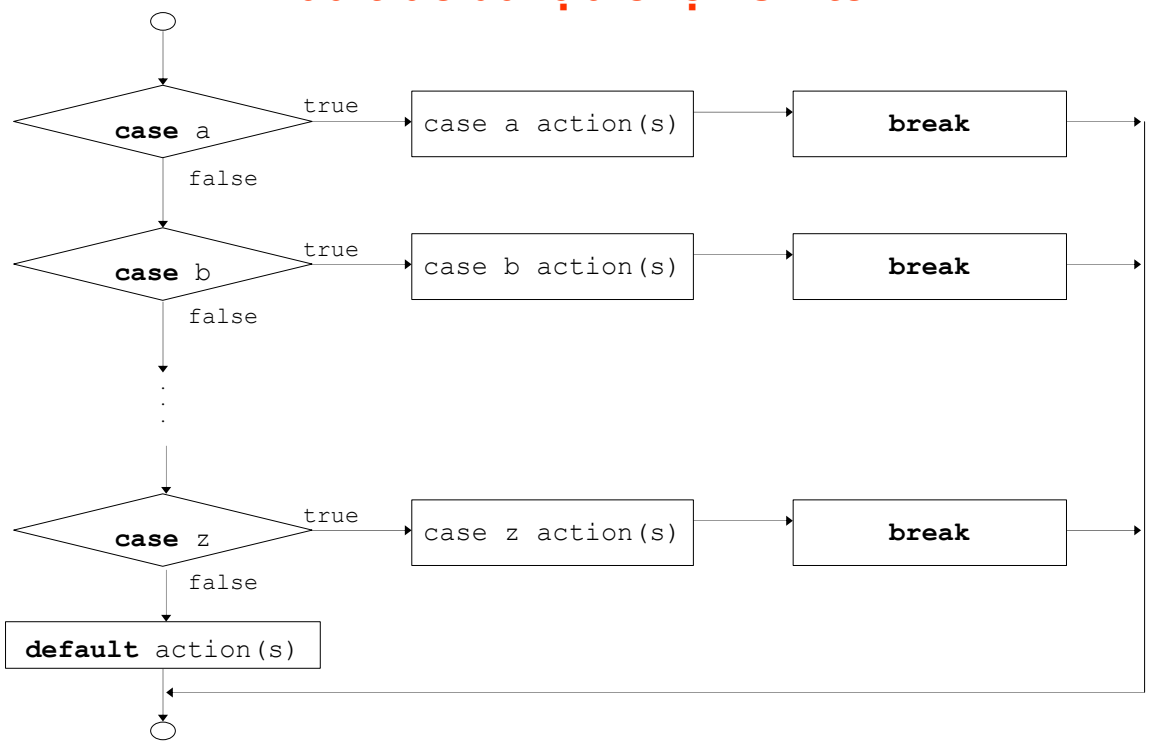
Các số được căn phải do các lệnh setw (với tham số có giá trị 4 và 21).

Cấu trúc đa lựa chọn switch

- **switch**
  - Test biến với nhiều giá trị
  - chuỗi các nhãn **case**
  - trường hợp **default** không bắt buộc

```
switch ( variable ) {
     case value1:         // taken if variable == value1
     statements
     break;              // necessary to exit switch

     case value2:
     case value3:       // taken if variable == value2 or == value3
     statements
     break;

     default:          // taken if variable matches no other cases
     statements
     break;
}
```


Cấu trúc đa lựa chọn switch



Cấu trúc đa lựa chọn switch

- Ví dụ sắp tới
  - Chương trình đọc xếp loại điểm (A-F)
  - Hiện số lượng mỗi xếp loại được nhập
- Chi tiết về các ký tự
  - Các ký tự đơn thường được lưu bằng kiểu dữ liệu **`char`**
    - **`char:`** số nguyên 1-byte, →có thể được lưu dưới dạng các giá trị **`int`**
  - Có thể coi ký tự là **`int`** hoặc **`char`**
    - 97 là biểu diễn dạng số của chữ 'a' thường (ASCII)
    - dùng cặp nháy đơn để lấy biểu diễn chữ của ký tự

```
cout << "The character (" << 'a' << ") has the value "
       << static_cast< int > ( 'a' ) << endl;
```

In ra dòng:

```
The character (a) has the value 97
```


fig02_22.cpp
(1 of 4)

```
1  // Fig. 2.22: fig02_22.cpp
2  // Counting letter grades.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  // function main begins program execution
10 int main()
11 {
12    int grade;       // one grade
13    int aCount = 0;  // number of As
14    int bCount = 0;  // number of Bs
15    int cCount = 0;  // number of Cs
16    int dCount = 0;  // number of Ds
17    int fCount = 0;  // number of Fs
18
19    cout << "Enter the letter grades." << endl
20         << "Enter the EOF character to end input." << endl;
21
```

```
22    // loop until user types end-of-file key sequence
23    while ( ( grade = cin.get() ) != EOF )
24
25       // determine which grade was input
26       switch ( grade ) {  // switch structure nested in while
27
28          case 'A':        // grade was uppercase A
29          case 'a':        // or lowercase a
30             ++aCount;     // increment aCount
31             break;        // necessary to exit swit
32
33          case 'B':                                   B
34          case 'b':
35             ++bCount;
36             break;
37
38          case 'C':                                   C
39          case 'c':
40             ++cCount;
41             break;
42
```

break kết thúc lệnh **switch** và chương trình chạy tiếp tại lệnh đầu tiên sau cấu trúc **switch**.

fig02_22.cpp
(2 of 4)

cin.get() sử dụng dot notation (ký hiệu kiểu dấu chấm). Hàm này đọc một ký tự từ bàn phím (sau khi nhấn *Enter*), và gán giá trị đó cho biến **grade**.

cin.get() trả về EOF (end-of-file), sau khi ký tự EOF được nhập, để đánh dấu kết thúc của dữ liệu vào. EOF có thể là ctrl-d hoặc ctrl-z, tùy theo hệ điều hành. (MS-Windows: ctrl-z, Unix/Linux: ctrl-d)

Các lệnh gán là biểu thức có giá trị bằng biến bên trái dấu gán =. Giá trị của lệnh này bằng giá trị trả về bởi hàm **cin.get()**.

Đặc điểm này còn được sử dụng để khởi tạo nhiều biến một lúc:
a = b = c = 0;

So sánh **grade** (một biến **int**) với biểu diễn số của **A** và **a**.


```
43          case 'D':        // grade was uppercase D
44          case 'd':        // or lowercase d
45             ++dCount;     // increment dCount
46             break;        // exit switch
47
48          case 'F':        // grade was up
49          case 'f':        // or lowercase
50             ++fCount;     // increment fC
51             break;        // exit switch
52
53          case '\n':       // ignore newlines,
54          case '\t':       // tabs,
55          case ' ':        // and spaces in input
56             break;        // exit switch
57
58          default:         // catch all other characters
59             cout << "Incorrect letter grade entered."
60                  << " Enter a new grade." << endl;
61             break;        // optional; will exit switch anyway
62
63       } // end switch
64
65    } // end while
66
```

fig02_22.cpp

Kiểm tra này là cần thiết vì *Enter* được nhấn sau mỗi chữ cái xếp loại được nhập. Việc nhấn *Enter* tạo một ký tự xuống dòng cần được loại bỏ. Cũng như vậy, ta muốn bỏ qua các ký tự trắng.

Lưu ý trường hợp **default** bao gồm tất cả các trường hợp còn lại (chưa xét đến).

```
67     // output summary of results
68     cout << "\n\nTotals for each letter grade are:"
69          << "\nA: " << aCount   // display number of A grades
70          << "\nB: " << bCount   // display number of B grades
71          << "\nC: " << cCount   // display number of C grades
72          << "\nD: " << dCount   // display number of D grades
73          << "\nF: " << fCount   // display number of F grades
74          << endl;
75
76     return 0;  // indicate successful termination
77
78  } // end function main
```

fig02_22.cpp (4 of 4)


```
Enter the letter grades.
Enter the EOF character to end input.
a
B
c
C
A
d
f
C
E
Incorrect letter grade entered. Enter a new grade.
D
A
b
^Z


Totals for each letter grade are:
A: 3
B: 2
C: 3
D: 2
F: 1
```

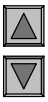
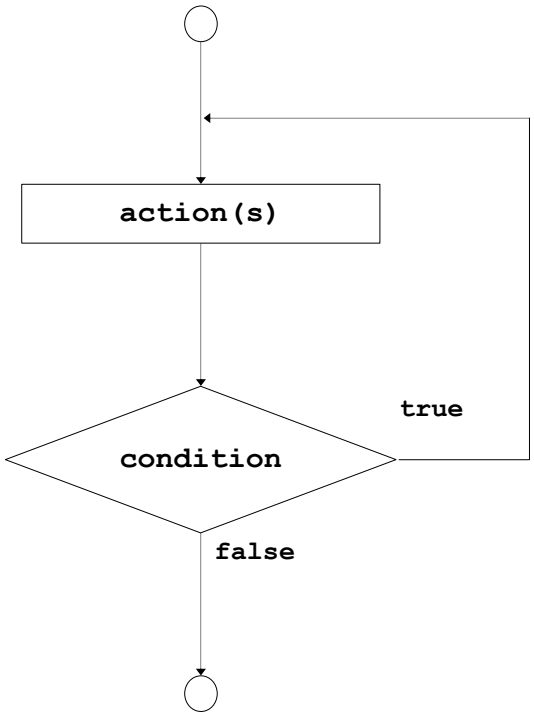


fig02_22.cpp output (1 of 1)

Cấu trúc lặp do/while

- Tương tự cấu trúc **while**
  - Kiểm tra điều kiện tiếp tục lặp ở cuối, không kiểm tra ở đầu
  - Thân vòng lặp chạy ít nhất một lần
- Công thức

```
do {
   statements
} while ( condition );
```




```
1  // Fig. 2.24: fig02_24.cpp
2  // Using the do/while repetition structure.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  // function main begins program
9  int main()
10 {
11    int counter = 1;              // initialize counter
12
13    do {
14       cout << counter << "  ";   // display counter
15    } while ( counter++ <= 10 );  // end do/while
16
17    cout << endl;
18
19    return 0;   // indicate successful termination
20
21 } // end function main
```

Chú ý phép tăng trước (preincrement) trong phần kiểm tra điều kiện lặp.

```
1  2  3  4  5  6  7  8  9  10
```

fig02_24.cpp (1 of 1)

fig02_24.cpp output (1 of 1)

Các lệnh break và continue

- **`break`**
  - Thoát ngay ra khỏi các cấu trúc **`while`**, **`for`**, **`do/while`**, **`switch`**
  - Chương trình tiếp tục chạy tại lệnh đầu tiên ngay sau cấu trúc
- thường được sử dụng để
  - Thoát khỏi vòng lặp sớm hơn bình thường
  - bỏ qua phần còn lại của **`switch`**


```
1  // Fig. 2.26: fig02_26.cpp
2  // Using the break statement in a for structure.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  // function main begins program execution
9  int main()
10 {
11
12    int x;  // x declared here so it can be used after the loop
13
14    // loop 10 times
15    for ( x = 1; x <= 10; x++ ) {
16
17       // if x is 5, terminate loop
18       if ( x == 5 )
19          break;           // break loop only if x is 5
20
21       cout << x << " ";   // display value of x
22
23    } // end for
24
25    cout << "\nBroke out of loop when x became " << x << endl;
26
27    return 0;   // indicate successful termination
28
29 } // end function main
```

```
1 2 3 4
Broke out of loop when x became 5
```

Thoát khỏi vòng **`for`** khi **`break`** được thực thi.

Các lệnh break và continue

- **`continue`**
  - được dùng trong **`while`**, **`for`**, **`do/while`**
  - bỏ qua phần còn lại của thân vòng lặp
  - chạy tiếp lần lặp tiếp theo
- với các vòng **`while`** và **`do/while`**
  - thực hiện kiểm tra điều kiện lặp ngay sau lệnh **`continue`**
- với vòng **`for`**
  - biểu thức tăng/giảm biến đếm được thực hiện
  - sau đó, điều kiện lặp được kiểm tra


```
1  // Fig. 2.27: fig02_27.cpp
2  // Using the continue statement in a for structure.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  // function main begins program execution
9  int main()
10 {
11    // loop 10 times
12    for ( int x = 1; x <= 10; x++ ) {
13
14       // if x is 5, continue with next iteration of loop
15       if ( x == 5 )
16          continue;        // skip remaining code in loop body
17
18       cout << x << " ";   // display value of x
19
20    } // end for structure
21
22    cout << "\nUsed continue to skip printing the value 5"
23         << endl;
24
25    return 0;              // indicate successful termination
26
27 } // end function main
```

```
1 2 3 4 6 7 8 9 10
Used continue to skip printing the value 5
```

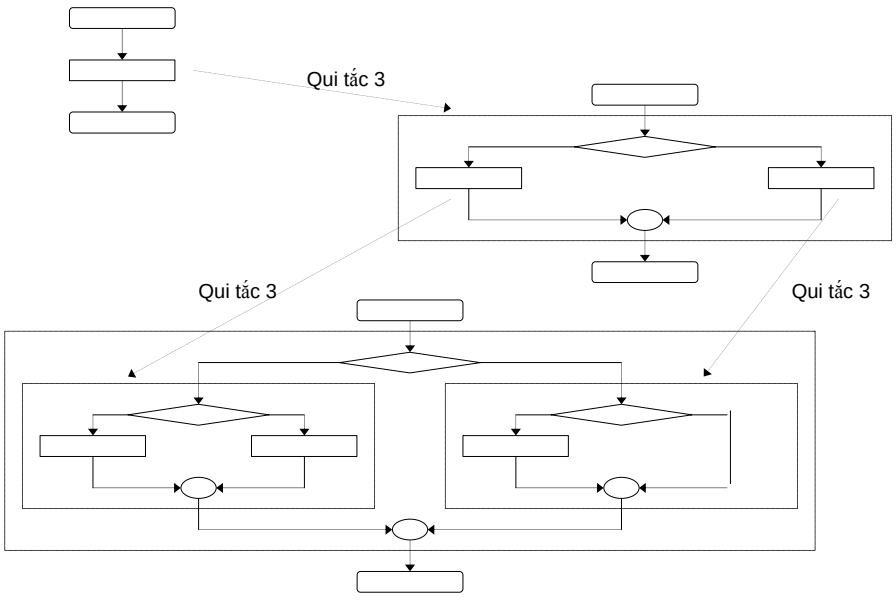
Bỏ qua phần còn lại của thân vòng **`for`**, nhảy đến lần lặp tiếp theo.

Lập trình cấu trúc
Structured-Programming

- Lập trình cấu trúc – Structured programming
  - Chương trình dễ hiểu, test, tìm lỗi (debug) và dễ sửa đổi hơn
- Các quy tắc lập trình cấu trúc
  - Chỉ sử dụng các cấu trúc điều khiển một đầu vào một đầu ra
  - Quy tắc
    1) Bắt đầu bằng một sơ đồ khối đơn giản nhất
    2) Mỗi hình chữ nhật (hành động) có thể được thay bằng một chuỗi gồm 2 hình chữ nhật khác
    3) Mỗi hình chữ nhật (hành động) có thể được thay bằng một cấu trúc điều khiển tùy ý (tuần tự, if, if/else, switch, while, do/while hoặc for)
    4) Các quy tắc 2 và 3 có thể được áp dụng nhiều lần và theo thứ tự tùy ý

Lập trình cấu trúc
Structured-Programming

Mô tả quy tắc 3 (thay một hình chữ nhật tùy ý bằng một cấu trúc điều khiển)



Ngôn ngữ lập trình C++

Chương 3 – Hàm

Chương 3 - Hàm

Đề mục

Chương 3 - Hàm

Đề mục

3.1 Giới thiệu

- Chia để trị - Divide and conquer
  - Xây dựng một chương trình từ các thành phần (component) nhỏ hơn
  - Quản lý từng thành phần dễ quản lý hơn quản lý chương trình ban đầu

3.2 Các thành phần của chương trình C++

- Các module: các hàm(function) và lớp(class)
- Các chương trình sử dụng các module mới và đóng gói sẵn ("prepackaged")
  - Mới: các hàm và lớp do lập trình viên tự định nghĩa
  - Đóng gói sẵn: các hàm và lớp từ thư viện chuẩn
- lời gọi hàm - function call
  - tên hàm và các thông tin (các đối số - arguments) mà nó cần
- định nghĩa hàm - function definition
  - chỉ viết một lần
  - được che khỏi các hàm khác
- tương tự
  - Một ông chủ (hàm gọi - the calling function or caller) đề nghị một công nhân (hàm được gọi - the called function) thực hiện một nhiệm vụ và trả lại (báo cáo lại) kết quả khi nhiệm vụ hoàn thành.

3.3 Các hàm trong thư viện toán học

- Thực hiện các tính toán toán học thông thường
  - Include header file **<cmath>** (hoặc **<math.h>**)
- Cách gọi hàm
  - tên_hàm (đối_số); **hoặc**
  - tên_hàm(đối_số_1, đối_số_2, …);
- Ví dụ

```
cout << sqrt( 900.0 );
```

  - Mọi hàm trong thư viện toán đều trả về giá trị kiểu **double**
- các đối số (argument) cho hàm có thể là
  - hằng - Constants
    - `sqrt( 4 );`
  - biến - Variables
    - `sqrt( x );`
  - biểu thức - Expressions
    - `sqrt( sqrt( x ) ) ;`
    - `sqrt( 3 - 6x );`

Method	Description	Example
`ceil( x )`	làm tròn *x* tới số nguyên nhỏ nhất không nhỏ hơn *x*	`ceil( 9.2 )` is `10.0` `ceil( -9.8 )` is `-9.0`
`cos( x )`	cos của *x* (lượng giác) (*x* tính theo đơn vị radian)	`cos( 0.0 )` is `1.0`
`exp( x )`	hàm mũ: *e mũ x*	`exp( 1.0 )` is `2.71828` `exp( 2.0 )` is `7.38906`

`fabs( x )`	giá trị tuyệt đối của *x*	`fabs( 5.1 )` is `5.1`
		`fabs( 0.0 )` is `0.0`
		`fabs( -8.76 )` is `8.76`

`floor( x )`	làm tròn *x* xuống số nguyên lớn nhất không lớn hơn *x*	`floor( 9.2 )` is `9.0` `floor( -9.8 )` is `-10.0`
`fmod( x, y )`	phần dư của phép chia *x/y* , tính bằng kiểu số thực	`fmod( 13.657, 2.333 )` is `1.992`
`log( x )`	loga tự nhiên của x (cơ số *e*)	`log( 2.718282 )` is `1.0` `log( 7.389056 )` is `2.0`
`log10( x )`	loga cơ số 10 của *x*	`log10( 10.0 )` is `1.0` `log10( 100.0 )` is `2.0`
`pow( x, y )`	*x* mũ *y*	`pow( 2, 7 )` is `128` `pow( 9, .5 )` is `3`
`sin( x )`	sin *x* (lượng giác) (*x* tính theo radian)	`sin( 0.0 )` is `0`
`sqrt( x )`	căn bậc hai của *x*	`sqrt( 900.0 )` is `30.0` `sqrt( 9.0 )` is `3.0`
`tan( x )`	tang *x* (lượng giác) (*x* tính theo radian)	`tan( 0.0 )` is `0`

Fig. 3.2 Math library functions.

3.4 Hàm - function

- Chương trình con
  - Module hóa một chương trình
  - khả năng tái sử dụng phần mềm – Software reusability
    - gọi hàm nhiều lần
- Các biến địa phương – Local variables
  - khai báo trong hàm nào thì chỉ được biết đến bên trong hàm đó
  - biến được khai báo bên trong định hàm là biến địa phương
- Các tham số – Parameters
  - là các biến địa phương với giá trị được truyền vào hàm khi hàm được gọi
  - cung cấp thông tin về bên ngoài hàm

```
2  // Creating and using a programmer-defined function.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int square( int );   // function prototype
9
10 int main()
11 {
12    // loop 10 times and calculate and output
13    // square of x each time
14    for ( int x = 1; x <= 10; x++ )
15       cout << square( x ) << "  ";  // function call
16
17    cout << endl;
18
19    return 0;  // indicates successful termination
20
21 } // end main
22
23 // square function definition returns square of an integer
24 int square( int y )  // y is a copy of argument to function
25 {
26    return y * y;      // returns square of
27
28 } // end function square
```

Function prototype: chỉ rõ kiểu dữ liệu của đối số và giá trị trả về. **square** cần một số **int**, và trả về **int**.

Cặp ngoặc **()** dùng khi gọi hàm. Khi chạy xong, hàm trả kết quả.

```
1   4   9   16   25   36   49   64   81   100
```

Định nghĩa hàm **square**. **y** là một bản sao của đối số được truyền vào. Hàm trả về **y * y**, hoặc **y** bình phương.

fig03_03.cpp (1 of 2)


```
1  // Fig. 3.4: fig03_04.cpp
2  // Finding the maximum of three floating-point numbers.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  double maximum( double, double, double ); // function prototype
10
11 int main()
12 {
13    double number1;
14    double number2;
15    double number3;
16
17    cout << "Enter three floating-point numbers: ";
18    cin >> number1 >> number2 >> number3;
19
20    // number1, number2 and number3 are arguments to
21    // the maximum function call
22    cout << "Maximum is: "
23         << maximum( number1, number2, number3 ) << endl;
24
25    return 0;  // indicates successful termination
```

Hàm **maximum** lấy 3 tham số (cả 3 là **double**) và trả về một **double**.

3_04.cpp 2)

```
26
27 } // end main
28
29 // function maximum definition;
30 // x, y and z are parameters
31 double maximum( double x, double y, double z )
32 {
33    double max = x;   // assume x is largest
34
35    if ( y > max )    // if y is larger,
36       max = y;       // assign y to max
37
38    if ( z > max )    // if z is larger,
39       max = z;       // assign z to max
40
41    return max;       // max is largest value
42
43 } // end function maximum
```

dấu phảy phân tách các tham số.

fig03_04.cpp (2 of 2)

fig03_04.cpp output (1 of 1)

```
Enter three floating-point numbers: 99.32 37.3 27.1928
Maximum is: 99.32

Enter three floating-point numbers: 1.1 3.333 2.22
Maximum is: 3.333

Enter three floating-point numbers: 27.9 14.31 88.99
```

hâu.

3.4 Hàm

- Nguyên mẫu hàm - Function prototype
  - Cho trình biên dịch biết kiểu dữ liệu của đối số và kiểu giá trị trả về của hàm
    `int square( int );`
    - Hàm lấy một giá trị **`int`** và trả về một giá trị **`int`**
  - Sẽ giới thiệu kỹ hơn sau

- Gọi hàm
  `square(x);`
  - Cặp ngoặc đơn là toán tử dùng để gọi hàm
    - Truyền đối số x
    - Hàm nhận được bản sao các đối số cho riêng mình
  - Sau khi kết thúc, hàm trả kết quả về cho nơi gọi hàm

3.5 Định nghĩa hàm – function definition

- định nghĩa hàm

  return-value-type function-name **(** *parameter-list* **)**
  {
  declarations and statements
  }

  – danh sách tham số – Parameter list
    - dấu phảy tách các tham số
      – mỗi tham số cần cho biết kiểu dữ liệu của tham số đó
    - Nếu không có đối số, sử dụng **`void`** hoặc để trống
  – giá trị trả về – Return-value-type
    - kiểu của giá trị trả về (sử dụng **`void`** nếu không trả về giá trị gì)

3.5 Định nghĩa hàm

Ví dụ về hàm

```
int square( int y )
{
   return y * y;
}

int main()
{
   ...
   cout << square(x);
   ...
}
```

- Từ khóa **`return`**
  – trả dữ liệu về, và trả điều khiển lại cho nơi gọi (caller)
    - nếu không trả về, sử dụng **`return;`**
  – hàm kết thúc khi chạy đến ngoặc phải (**}**)
    - điều khiển cũng được trả về cho nơi gọi
- Không thể định nghĩa một hàm bên trong một hàm khác

3.6 Nguyên mẫu hàm - Function Prototype

- Function prototype bao gồm
  - Tên hàm
  - Các tham số (số lượng và kiểu dữ liệu)
  - Kiểu trả về (**`void`** nếu không trả về giá trị gì)
- Function prototype chỉ cần đến nếu định nghĩa hàm đặt sau lời gọi hàm (function call)
- Prototype phải khớp với định nghĩa hàm
  - Function prototype

```
double maximum( double, double, double );
```

  - Function definition

```
double maximum( double x, double y, double z )
{
  …
}
```


3.6 Function Prototype

- Chữ ký của hàm - Function signature
  - Phần prototype chứa tên và các tham số của hàm
    - `double maximum( double, double, double );`

Function signature

- Ép kiểu đối số – Argument Coercion
  - Ép các đối số thành các kiểu dữ liệu thích hợp
    - đổi **`int`** (4) thành **`double`** (4.0)

```
cout << sqrt(4)
```

  - các quy tắc biến đổi
    - các đối số thường được tự động đổi kiểu
    - đổi từ **`double`** sang **`int`** có thể làm tròn dữ liệu
      - 3.4 thành 3
  - các kiểu hỗn hợp được nâng lên kiểu cao nhất
    - **`int * double`**

3.6 Function Prototype

Data types	
`long double`	
`double`	
`float`	
`unsigned long int`	(synonymous with `unsigned long`)
`long int`	(synonymous with `long`)
`unsigned int`	(synonymous with `unsigned`)
`int`	
`unsigned short int`	(synonymous with `unsigned short`)
`short int`	(synonymous with `short`)
`unsigned char`	
`char`	
`bool`	(`false` becomes 0, `true` becomes 1)

Fig. 3.5 Promotion hierarchy for built-in data types.

3.7 Header File

- Các file header chứa
  - các function prototype
  - định nghĩa của các kiểu dữ liệu và các hằng
- Các file header kết thúc bằng .h
  - các file header do lập trình viên định nghĩa

```
#include "myheader.h"
```

- Các file header của thư viện

```
#include <cmath>
```

  - chú ý:
    - <cmath> tương đương với <math.h> (kiểu cũ, trước ANSI C++)
    - <iostream> tương đương với <iostream.h> (kiểu cũ, trước ANSI C++)

3.8 Sinh số ngẫu nhiên
Random Number Generation

- Hàm **rand** (thuộc **<cstdlib>**)
  - **i = rand();**
  - Sinh một số nguyên không âm trong đoạn từ 0 đến RAND_MAX (thường là 32767)
- Lấy tỷ lệ và dịch (scaling and shifting)
  - phép đồng dư (lấy số dư) – Modulus (remainder) operator: **%**
    - **10 % 3** bằng **1**
    - **x % y** nằm giữa **0** và **y – 1**
  - Ví dụ

    i = rand() % 6 + 1;
    - "**Rand() % 6**" sinh một số trong khoảng từ **0** đến **5** (lấy tỷ lệ)
    - "**+ 1**" tạo khoảng từ 1 đến 6 (dịch)
  - Tiếp theo: chương trình thả súc sắc

fig03_07.cpp

```
// Fig. 3.7: fig03_07.cpp
// Shifted, scaled integers produced by 1 + rand() % 6.
#include <iostream>

using std::cout;
using std::endl;

#include <iomanip>

using std::setw;

#include <cstdlib>   // contains function prototype for rand

int main()
{
   // loop 20 times
   for ( int counter = 1; counter <= 20; counter++ )

      // pick random number from 1 to 6 and output it
      cout << setw( 10 ) << ( 1 + rand() % 6 );

      // if counter divisible by 5, begin new line of output
      if ( counter % 5 == 0 )
         cout << endl;

   } // end for structure

   return 0;  // indicates successful termination

} // end main
```

```
    6     6     5     5     6
    5     1     1     5     3
    6     6     2     4     2
    6     2     3     4     1
```

Kết quả của **rand()** được lấy tỷ lệ và dịch thành một số trong khoảng từ 1 đến 6.

3.8 Sinh số ngẫu nhiên

- Tiếp theo
  - Chương trình biểu diễn phân bố (distribution) của hàm **`rand()`**
  - Giả lập 6000 lần thả súc sắc
  - In số lượng các giá trị 1, 2, 3, v.v.... thả được
  - số lượng đếm được của mỗi giá trị phải xấp xỉ 1000


```
1  // Fig. 3.8: fig03_08.cpp
2  // Roll a six-sided die 6000 times.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <iomanip>
9
10 using std::setw;
11
12 #include <cstdlib>   // contains function prototype for rand
13
14 int main()
15 {
16    int frequency1 = 0;
17    int frequency2 = 0;
18    int frequency3 = 0;
19    int frequency4 = 0;
20    int frequency5 = 0;
21    int frequency6 = 0;
22    int face;  // represents one roll of the die
23
```

fig03_08.cpp (1 of 3)

```
24    // loop 6000 times and summarize results
25    for ( int roll = 1; roll <= 6000; roll++ ) {
26       face = 1 + rand() % 6;  // random number from 1 to 6
27
28       // determine face value and increment appropriate counter
29       switch ( face ) {
30
31          case 1:          // rolled 1
32             ++frequency1;
33             break;
34
35          case 2:          // rolled 2
36             ++frequency2;
37             break;
38
39          case 3:          // rolled 3
40             ++frequency3;
41             break;
42
43          case 4:          // rolled 4
44             ++frequency4;
45             break;
46
47          case 5:          // rolled 5
48             ++frequency5;
49             break;
```

fig03_08.cpp
(2 of 3)

```
50
51          case 6:          // rolled 6
52             ++frequency6;
53             break;
54
55          default:         // invalid value
56             cout << "Program should never get here!";
57
58       } // end switch
59
60    } // end for
61
62    // display results in tabular format
63    cout << "Face" << setw( 13 ) << "Frequency"
64         << "\n   1" << setw( 13 ) << frequency1
65         << "\n   2" << setw( 13 ) << frequency2
66         << "\n   3" << setw( 13 ) << frequency3
67         << "\n   4" << setw( 13 ) << frequency4
68         << "\n   5" << setw( 13 ) << frequency5
69         << "\n   6" << setw( 13 ) << frequency6 << endl;
70
71    return 0;  // indicates successful termination
72
73 } // end main
```

Trường hợp mặc định được xét đến, ngay cả khi nó không bao giờ xảy ra. Đây là một nét của phong cách lập trình tốt

```
Face      Frequency
   1           1003
   2           1017
   3            983
   4            994
   5           1004
   6            999
```

fig03_08.cpp
(3 of 3)

3.8 Sinh số ngẫu nhiên

- Gọi rand() lặp đi lặp lại
  - cho kết quả là cùng một chuỗi số
- Các số giả ngẫu nhiên (pseudorandom numbers)
  - chuỗi các số "ngẫu nhiên" được định sẵn
  - chương trình chạy lần nào cũng sinh cùng một chuỗi
- Để được các chuỗi ngẫu nhiên khác nhau
  - Cung cấp một giá trị hạt giống
    - điểm xuất phát cho việc sinh chuỗi ngẫu nhiên
    - hạt giống giống nhau sẽ cho cùng một chuỗi ngẫu nhiên
  - **`srand(seed);`**
    - **`<cstdlib>`**
    - sử dụng trước **`rand()`** để đặt hạt giống

fig03_09.cpp
(1 of 2)

```
1  // Fig. 3.9: fig03_09.cpp
2  // Randomizing die-rolling program.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  #include <iomanip>
10
11 using std::setw;
12
13 // contains prototypes for functions srand and rand
14 #include <cstdlib>
15
16 // main function begins program execution
17 int main()
18 {
19    unsigned seed;
20
21    cout << "Enter seed: ";
22    cin >> seed;
23    srand( seed );  // seed random number generator
24
```

Đặt hạt giống bằng **`srand()`**.

```
25     // loop 10 times
26     for ( int counter = 1; counter <= 10; counter++ ) {
27
28        // pick random number from 1 to 6 and output it
29        cout << setw( 10 ) << ( 1 + rand() % 6 );
30
31        // if counter divisible by 5, begin new line of output
32        if ( counter % 5 == 0 )
33           cout << endl;
34
35     } // end for
36
37     return 0;  // indicates successful termination
38
39  } // end main
```

rand() sinh cùng một chuỗi ngẫu nhiên nếu dùng cùng một hạt giống

```
Enter seed: 67
         6         1         4         6         2
         1         6         1         6         4

Enter seed: 432
         4         6         3         1         6
         3         1         5         4         2

Enter seed: 67
         6         1         4         6         2
         1         6         1         6         4
```

fig03_09.cpp (2 of 2)

fig03_09.cpp output (1 of 1)

3.8 Sinh số ngẫu nhiên

- Có thể sử dụng thời gian hiện tại để làm hạt giống
  - không cần phải đặt hạt giống mỗi lần sinh 1 số ngẫu nhiên
  - **`srand( time( 0 ) );`**
  - **`time( 0 );`**
    - **`<ctime>`**
    - trả về thời gian hiện tại, tính bằng giây
- Tổng quát về dịnh và lấy tỷ lệ
  - *Number* **=** *shiftingValue* **+ `rand()` %** *scalingFactor*
  - shiftingValue = số đầu tiên của khoảng mong muốn
  - scalingFactor = độ rộng của khoảng mong muốn

3.9 Ví dụ: Trò chơi may rủi và Giới thiệu về kiểu enum

- Kiểu liệt kê - Enumeration
  - tập hợp các số tự nhiên được đặt tên

    enum *typeName* **{***constant1*, *constant2*…**};**
  - Các hằng số là các số nguyên bắt đầu từ 0 (mặc định), tăng dần, mỗi lần thêm 1 đơn vị.
  - Các hằng phải có tên riêng
  - Không thể gán giá trị kiểu nguyên cho biến kiểu liệt kê
    - Phải dùng một giá trị thuộc cùng kiểu liệt kê đã được định nghĩa
- Ví dụ

```
enum Status {CONTINUE, WON, LOST};
enum Foo {Zero, One, Two};
Status enumVar;
enumVar = WON; // cannot do enumVar = 1 or enumVar=One
```


3.9 Ví dụ: Trò chơi may rủi và Giới thiệu về kiểu enum

- Các hằng kiểu liệt kê có thể có giá trị đặt trước

```
enum Months { JAN = 1, FEB, MAR, APR, MAY,
  JUN, JUL, AUG, SEP, OCT, NOV, DEC};
```

  - bắt đầu tại 1, tăng dần mỗi lần thêm 1
- Tiếp theo: giả lập trò chơi gieo súc sắc
  - Gieo 2 con súc sắc, được kết quả là tổng hai giá trị gieo được
  - 7 hoặc 11 tại lần gieo đầu tiên: người chơi thắng
  - 2, 3, hoặc 12 tại lần gieo đầu tiên: người chơi thua
  - 4, 5, 6, 8, 9, 10
    - giá trị gieo được trở thành "điểm" (point) của người chơi
    - người chơi phải gieo được số điểm của mình trước khi gieo được 7 để thắng cuộc

fig03_10.cpp
(1 of 5)

```
1  // Fig. 3.10: fig03_10.cpp
2  // Craps.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  // contains function prototypes for functions srand and rand
9  #include <cstdlib>
10
11 #include <ctime>   // contains prototype for function time
12
13 int rollDice( void );  // function prototype
14
15 int main()
16 {
17    // enumeration constants represent game status
18    enum Status { CONTINUE, WON, LOST };
19
20    int sum;
21    int myPoint;
22
23    Status gameStatus;  // can contain CONTINUE, WON or LOST
24
```

Hàm gieo 2 con súc sắc và trả về kết quả là 1 giá trị kiểu **int**.

Kiểu liệt kê để ghi trạng thái của ván chơi hiện tại.

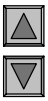


fig03_10.cpp
(2 of 5)

```
25    // randomize random number generator using current time
26    srand( time( 0 ) );
27
28    sum = rollDice();   // first roll of the dice
29
30    // determine game status and point based on sum of dice
31    switch ( sum ) {
32
33       // win on first roll
34       case 7:
35       case 11:
36          gameStatus = WON;
37          break;
38
39       // lose on first roll
40       case 2:
41       case 3:
42       case 12:
43          gameStatus = LOST;
44          break;
45
```

lệnh **switch** quyết định kết cục ván chơi, dựa vào kết quả gieo súc sắc.

```
46       // remember point
47       default:
48          gameStatus = CONTINUE;
49          myPoint = sum;
50          cout << "Point is " << myPoint << endl;
51          break;                // optional
52
53    } // end switch
54
55    // while game not complete ...
56    while ( gameStatus == CONTINUE ) {
57       sum = rollDice();          // roll dice again
58
59       // determine game status
60       if ( sum == myPoint )      // win by making point
61          gameStatus = WON;
62       else
63          if ( sum == 7 )         // lose by rolling 7
64             gameStatus = LOST;
65
66    } // end while
67
```

fig03_10.cpp
(3 of 5)


```
68    // display won or lost message
69    if ( gameStatus == WON )
70       cout << "Player wins" << endl;
71    else
72       cout << "Player loses" << endl;
73
74    return 0;  // indicates successful termination
75
76 } // end main
77
78 // roll dice, calculate sum and display results
79 int rollDice( void )
80 {
81    int die1;
82    int die2;
83    int workSum;
84
85    die1 = 1 + rand() % 6;  // pick random die1 value
86    die2 = 1 + rand() % 6;  // pick random die2 value
87    workSum = die1 + die2;  // sum die1 and die2
88
```

Hàm **rollDice** không lấy đối số, nên nó có từ khóa **void** tại danh sách tham số.

fig03_10.cpp
(4 of 5)

```
89     // display results of this roll
90     cout << "Player rolled " << die1 << " + " << die2
91          << " = " << workSum << endl;
92
93     return workSum;          // return sum of dice
94
95  } // end function rollDice
```

fig03_10.cpp (5 of 5)

fig03_10.cpp output (1 of 2)

```
Player rolled 2 + 5 = 7
Player wins

Player rolled 6 + 6 = 12
Player loses

Player rolled 3 + 3 = 6
Point is 6
Player rolled 5 + 3 = 8
Player rolled 4 + 5 = 9
Player rolled 2 + 1 = 3
Player rolled 1 + 5 = 6
Player wins
```

```
Player rolled 1 + 3 = 4
Point is 4
Player rolled 4 + 6 = 10
Player rolled 2 + 4 = 6
Player rolled 6 + 4 = 10
Player rolled 2 + 3 = 5
Player rolled 2 + 4 = 6
Player rolled 1 + 1 = 2
Player rolled 4 + 4 = 8
Player rolled 4 + 3 = 7
Player loses
```


3.10 Các kiểu lưu trữ – Storage Classes

- biến có các thuộc tính
  - đã biết: tên, kiểu, kích thước, giá trị
  - kiểu lưu trữ – Storage class
    - biến tồn tại bao lâu trong bộ nhớ
  - Phạm vi – Scope
    - biến có thể được sử dụng tại những nơi nào trong chương trình
  - Liên kết – Linkage
    - Đối với những chương trình gồm nhiều file (multiple-file program) – (xem chương 6), những file nào có thể sử dụng biến đó

3.10 Các kiểu lưu trữ – Storage Classes

- loại biến tự động – Automatic storage class
  - biến được tạo khi chương trình chạy vào một khối chương trình (block)
  - và bị hủy bỏ khi chương trình ra khỏi block
  - Chỉ có các biến địa phương của các hàm mới có thể là biến tự động
    - mặc định là tự động
    - từ khóa **`auto`** dùng để khai báo biến tự động
  - từ khóa **`register`**
    - gợi ý đặt biến vào thanh ghi tốc độ cao
    - có lợi cho các biến thường xuyên được sử dụng (con đếm vòng lặp)
    - Thường là không cần thiết, trình biên dịch tự tối ưu hóa
  - Chỉ dùng một trong hai từ **`register`** hoặc **`auto`**.
    - **`register int counter = 1;`**

3.10 Các kiểu lưu trữ

- loại biến tĩnh – Static storage class
  - Biến tồn tại trong suốt chương trình
  - Có thể không phải nơi nào cũng dùng được, do áp dụng quy tắc phạm vi (scope rules)
- từ khóa **`static`**
  - dành cho biến địa phương bên trong hàm
  - giữ giá trị giữa các lần gọi hàm
  - chỉ được biết đến trong hàm của biến đó
- từ khóa **`extern`**
  - mặc định với các biến/hàm toàn cục (global variables/functions)
    - toàn cục: được định nghĩa bên ngoài các hàm
  - được biết đến tại mọi hàm nằm sau biến đó

3.11 Các quy tắc phạm vi – Scope Rules

- Phạm vi – Scope
  - Phạm vi của một định danh (tên) là phần chương trình nơi có thể sử dụng định danh đó
- Phạm vi file – File scope
  - được định nghĩa bên ngoài một hàm và được biết đến tại mọi hàm trong file
  - các biến toàn cục (global variable), định nghĩa và prototype của các hàm.
- Phạm vi hàm – Function scope
  - chỉ có thể được dùng đến bên trong hàm chứa định nghĩa
  - Chỉ áp dụng cho các nhãn (label), ví dụ: các định danh đi kèm một dấu hai chấm (`case:`)

3.11 Các quy tắc phạm vi

- Phạm vi khối – Block scope
  - Bắt đầu tại nơi khai báo, kết thúc tại ngoặc phải `}`
    - chỉ có thể được dùng trong khoảng này
  - Các biến địa phương, các tham số hàm
  - các biến `static` cũng có phạm vi khối
    - loại lưu trữ độc lập với phạm vi
- Function-prototype scope
  - danh sách tham số của function prototype
  - không bắt buộc phải chỉ rõ các tên trong prototype
    - Trình biên dịch bỏ qua
  - Trong một prototype, mỗi tên chỉ được dùng một lần

fig03_12.cpp
(1 of 5)

```
1  // Fig. 3.12: fig03_12.cpp
2  // A scoping example.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  void useLocal( void );
9  void useStaticLocal( void );
10 void useGlobal( void );        // function prototype
11
12 int x = 1;       // global variable
13
14 int main()
15 {
16    int x = 5;    // local variable to main
17
18    cout << "local x in main's outer scope is " << x << endl;
19
20    { // start new scope
21
22       int x = 7;
23
24       cout << "local x in main's inner scope is " << x << endl;
25
26    } // end new scope
```

được khai báo bên ngoài hàm; là biến toàn cục với phạm vi file.

Biến địa phương với phạm vi hàm.

Tạo một khối, cho **x** phạm vi khối. Khi khối kết thúc, **x** sẽ bị hủy bỏ.

g03_12.cpp
2 of 5)

```
27
28    cout << "local x in main's outer scope is " << x << endl;
29
30    useLocal();       // useLocal has local x
31    useStaticLocal(); // useStaticLocal has static local x
32    useGlobal();      // useGlobal uses global x
33    useLocal();       // useLocal reinitializes its local x
34    useStaticLocal(); // static local x retains its prior value
35    useGlobal();      // global x also retains its value
36
37    cout << "\nlocal x in main is " << x << endl;
38
39    return 0;   // indicates successful termination
40
41 } // end main
42
```

```
43 // useLocal reinitializes local variable x during each call
44 void useLocal( void )
45 {
46    int x = 25; // initialized each time useLocal is called
47
48    cout << endl << "local x is
49         << " on entering useLo
50    ++x;
51    cout << "local x is " << x
52         << " on exiting useLocal" << endl;
53
54 } // end function useLocal
55
```

Biến tự động (biến địa phương của hàm). Biến này sẽ bị hủy khi hàm kết thúc, và được khởi tạo lại khi hàm bắt đầu.

fig03_12.cpp (3 of 5)


```
56 // useStaticLocal initializes static local variable x only the
57 // first time the function is called; value of x is saved
58 // between calls to this function
59 void useStaticLocal( void )
60 {
61    // initialized only first time useStaticLocal is called
62    static int x = 50;
63
64    cout << endl << "local static x is " << x
65         << " on entering useStaticLocal" << endl;
66    ++x;
67    cout << "local static x is " << x
68         << " on exiting useStaticLocal" << endl
69
70 } // end function useStaticLocal
71
```

Biến tĩnh địa phương của hàm; nó được khởi tạo đúng một lần và giữ nguyên giá trị giữa các lần gọi hàm.

fig03_12.cpp (4 of 5)

```
72  // useGlobal modifies global variable x during each call
73  void useGlobal( void )
74  {
75     cout << endl << "global x is " << x
76          << " on entering useGlobal" << endl;
77     x *= 10;
78     cout << "global x is " << x
79          << " on exiting useGlobal" << endl;
80
81  } // end function useGlobal
```

Hàm này không khai báo biến nào. Nó sử dụng biến toàn cục **x** đã được khai báo tại đầu chương trình.

fig03_12.cpp (5 of 5)

fig03_12.cpp output (1 of 2)

```
local x in main's outer scope is 5
local x in main's inner scope is 7
local x in main's outer scope is 5

local x is 25 on entering useLocal
local x is 26 on exiting useLocal

local static x is 50 on entering useStaticLocal
local static x is 51 on exiting useStaticLocal

global x is 1 on entering useGlobal
global x is 10 on exiting useGlobal
```


fig03_12.cpp output (2 of 2)

```
local x is 25 on entering useLocal
local x is 26 on exiting useLocal

local static x is 51 on entering useStaticLocal
local static x is 52 on exiting useStaticLocal

global x is 10 on entering useGlobal
global x is 100 on exiting useGlobal

local x in main is 5
```

3.12 Đệ quy – Recursion

- Các hàm đệ quy – Recursive functions
  - các hàm tự gọi chính mình
  - chỉ giải quyết một trường hợp cơ bản (base case)
- Nếu không phải trường hợp cơ bản
  - Chia bài toán thành các bài toán nhỏ hơn
  - Gọi bản sao mới của hàm để giải quyết vấn đề nhỏ hơn (gọi đệ quy (recursive call) hoặc bước đệ quy(recursive step))
    - hội tụ dần dần về trường hợp cơ bản
    - hàm gọi chính nó tại lệnh return
  - Cuối cùng, trường hợp cơ bản được giải quyết
    - câu trả lời đi ngược lên, giải quyết toàn bộ bài toán

3.12 Đệ quy

- Ví dụ: tính giai thừa (factorial)

  $n! = n * (n – 1) * (n – 2) * ... * 1$

  - Quan hệ đệ quy (n! = n * (n – 1)!)

    $5! = 5 * 4!$

    $4! = 4 * 3!...$

  - Trường hợp cơ bản (1! = 0! = 1)

```
1  // Fig. 3.14: fig03_14.cpp
2  // Recursive factorial function.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <iomanip>
9
10 using std::setw;
11
12 unsigned long factorial( unsigned long ); // function prototype
13
14 int main()
15 {
16    // Loop 10 times. During each iteration, calculate
17    // factorial( i ) and display result.
18    for ( int i = 0; i <= 10; i++ )
19       cout << setw( 2 ) << i << "! = "
20            << factorial( i ) << endl;
21
22    return 0;  // indicates successful termination
23
24 } // end main
```

Kiểu dữ liệu **unsigned long** có thể lưu số nguyên trong khoảng từ 0 đến 4 tỷ.

fig03_14.cpp (1 of 2)


```
25
26 // recursive definition of function factorial
27 unsigned long factorial( unsigned long number )
28 {
29    // base case
30    if ( number <= 1 )
31       return 1;
32
33    // recursive step
34    else
35       return number * factorial( number - 1 );
36
37 } // end function factorial
```

Trường hợp cơ bản xảy ra khi ta có **0!** hoặc **1!**.

Mọi trường hợp khác phải được chia nhỏ (bước đệ qui).

fig03_14.cpp (2 of 2)

fig03_14.cpp output (1 of 1)

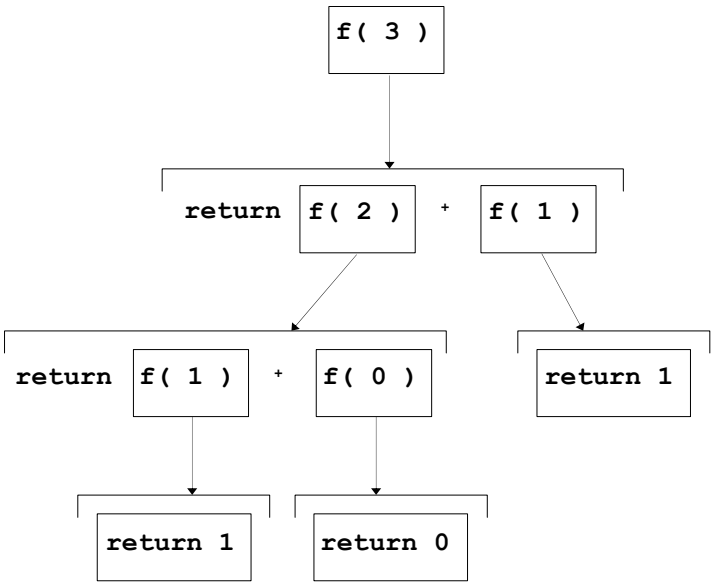
```
 0! = 1
 1! = 1
 2! = 2
 3! = 6
 4! = 24
 5! = 120
 6! = 720
 7! = 5040
 8! = 40320
 9! = 362880
10! = 3628800
```

3.13 Ví dụ sử dụng đệ quy: chuỗi Fibonacci

- Chuỗi Fibonacci: 0, 1, 1, 2, 3, 5, 8...
  - Mỗi số là tổng của hai số đứng liền trước
  - Ví dụ một công thức đệ quy:
    - *fib(n) = fib(n-1) + fib(n-2)*
- Mã C++ cho hàm Fibonacci

```
long fibonacci( long n )
{
   if ( n == 0 || n == 1 )  // base case
       return n;
  else
   return fibonacci( n - 1 ) +
          fibonacci( n - 2 );
}
```


3.13 Ví dụ sử dụng đệ quy: chuỗi Fibonacci



3.13 Ví dụ sử dụng đệ quy: chuỗi Fibonacci

- Thứ tự thực hiện
  - `return fibonacci( n - 1 ) + fibonacci( n - 2 );`
- Không xác định hàm nào được thực hiện trước
  - C++ không qui định
  - Chỉ có các phép **`&&, ||`** và **`?:`** đảm bảo thứ tự thực hiện từ trái sang phải
- Các lời gọi hàm đệ quy
  - Mỗi tầng đệ quy nhân đôi số lần gọi hàm
    - số thứ 30 cần 2^30 ~ 4 tỷ lời gọi hàm
  - Độ phức tạp lũy thừa (Exponential complexity)

fig03_15.cpp (1 of 2)

```
1  // Fig. 3.15: fig03_15.cpp
2  // Recursive fibonacci function.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  unsigned long fibonacci( unsigned long ); // func
10
11 int main()
12 {
13    unsigned long result, number;
14
15    // obtain integer from user
16    cout << "Enter an integer: ";
17    cin >> number;
18
19    // calculate fibonacci value for number input by user
20    result = fibonacci( number );
21
22    // display result
23    cout << "Fibonacci(" << number << ") = " << result << endl;
24
25    return 0;  // indicates successful termination
```

Các số Fibonacci tăng rất nhanh và đều là số không âm. Do đó, ta dùng kiểu **`unsigned long`**.

```
26
27 } // end main
28
29 // recursive definition of function fibonacci
30 unsigned long fibonacci( unsigned long n )
31 {
32    // base case
33    if ( n == 0 || n == 1 )
34       return n;
35
36    // recursive step
37    else
38       return fibonacci( n - 1 ) + fibonacci( n - 2 );
39
40 } // end function fibonacci
```

fig03_15.cpp (2 of 2)

fig03_15.cpp output (1 of 2)

```
Enter an integer: 0
Fibonacci(0) = 0

Enter an integer: 1
Fibonacci(1) = 1

Enter an integer: 2
Fibonacci(2) = 1

Enter an integer: 3
Fibonacci(3) = 2
```



```
Enter an integer: 4
Fibonacci(4) = 3

Enter an integer: 5
Fibonacci(5) = 5

Enter an integer: 6
Fibonacci(6) = 8

Enter an integer: 10
Fibonacci(10) = 55

Enter an integer: 20
Fibonacci(20) = 6765

Enter an integer: 30
Fibonacci(30) = 832040

Enter an integer: 35
Fibonacci(35) = 9227465
```

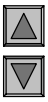


fig03_15.cpp output (2 of 2)

3.14 So sánh Đệ quy và Vòng lặp

- Lặp
  - Vòng lặp (Iteration): lặp tường minh
  - Đệ quy: các lời gọi hàm được lặp đi lặp lại
- Kết thúc
  - Vòng lặp: điều kiện lặp thất bại
  - Đệ quy: gặp trường hợp cơ bản
- Cả hai đều có thể lặp vô tận
- Cân đối giữa hiệu quả chương trình (vòng lặp) và công nghệ phần mềm tốt (đệ quy)
  - vòng lặp chạy nhanh hơn
  - đệ quy trong sáng hơn

3.15 Hàm với danh sách tham số rỗng

- Danh sách tham số rỗng
  - dùng từ khóa **`void`** hoặc để danh sách rỗng
  - dành cho hàm không lấy đối số
  - thí dụ: hàm **`print`** không lấy đối số và không trả về giá trị nào
    - **`void print();`**
    - **`void print( void );`**

```
1  // Fig. 3.18: fig03_18.cpp
2  // Functions that take no arguments.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  void function1();        // function prototype
9  void function2( void );  // function prototype
10
11 int main()
12 {
13    function1();  // call function1 with no arguments
14    function2();  // call function2 with no arguments
15
16    return 0;     // indicates successful termination
17
18 } // end main
19
```

fig03_18.cpp (1 of 2)


```
20 // function1 uses an empty parameter list to specify that
21 // the function receives no arguments
22 void function1()
23 {
24    cout << "function1 takes no arguments" << endl;
25
26 } // end function1
27
28 // function2 uses a void parameter list to specify that
29 // the function receives no arguments
30 void function2( void )
31 {
32    cout << "function2 also takes no arguments" << endl;
33
34 } // end function2
```

fig03_18.cpp (2 of 2)

fig03_18.cpp output (1 of 1)

```
function1 takes no arguments
function2 also takes no arguments
```

3.16 Hàm Inline

- Hàm inline
  - Từ khóa **`inline`** đặt trước hàm
  - Yêu cầu trình biên dịch sao chép mã vào chương trình thay cho việc tạo lời gọi hàm
    - Giảm chi phí gọi hàm (function-call overhead)
    - Trình biên dịch có thể bỏ qua **`inline`**
  - Tốt đối với các hàm nhỏ, hay dùng
- Ví dụ

```
inline double cube( const double s )
   { return s * s * s; }
```

  - **`const`** cho trình biên dịch biết rằng hàm không sửa đổi **`s`**
    - được nói đến trong các chương 6-7


```
1  // Fig. 3.19: fig03_19.cpp
2  // Using an inline function to calculate.
3  // the volume of a cube.
4  #include <iostream>
5
6  using std::cout;
7  using std::cin;
8  using std::endl;
9
10 // Definition of inline function cube. Definition of function
11 // appears before function is called, so a function prototype
12 // is not required. First line of function definition acts as
13 // the prototype.
14 inline double cube( const double side )
15 {
16    return side * side * side;  // calculate cube
17
18 } // end function cube
19
```

fig03_19.cpp (1 of 2)

```
20 int main()
21 {
22    cout << "Enter the side length of your cube: ";
23
24    double sideValue;
25
26    cin >> sideValue;
27
28    // calculate cube of sideValue and display result
29    cout << "Volume of cube with side "
30         << sideValue << " is " << cube( sideValue ) << endl;
31
32    return 0;  // indicates successful termination
33
34 } // end main
```

```
Enter the side length of your cube: 3.5
Volume of cube with side 3.5 is 42.875
```

fig03_19.cpp (2 of 2)

fig03_19.cpp output (1 of 1)

3.17 Tham chiếu và Tham số là tham chiếu

- Các tham chiếu là các biệt danh (alias) của các biến khác
  - chỉ tới cùng một biến
  - có thể được dùng bên trong một hàm

```
int count = 1; // khai báo biến nguyên count
int &cRef = count; // tạo cRef là một biệt danh của count
++cRef; // tăng count (sử dụng biệt danh của count)
```

- Các tham chiếu phải được khởi tạo khi khai báo
  - Nếu không, trình biên dịch báo lỗi
  - Tham chiếu lạc (Dangling reference)
    - tham chiếu tới biến không xác định

```
1  // Fig. 3.21: fig03_21.cpp
2  // References must be initialized.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int main()
9  {
10    int x = 3;
11
12    // y refers to (is an alias for) x
13    int &y = x;
14
15    cout << "x = " << x << endl << "y = " << y << endl;
16    y = 7;
17    cout << "x = " << x << endl << "y = " << y << endl;
18
19    return 0;  // indicates successful termination
20
21 } // end main
```

y được khai báo là một tham chiếu tới **x**.

```
x = 3
y = 3
x = 7
y = 7
```

fig03_21.cpp (1 of 1)

fig03_21.cpp output (1 of 1)


```
1  // Fig. 3.22: fig03_22.cpp
2  // References must be initialized.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int main()
9  {
10    int x = 3;
11    int &y;      // Error: y must be initialized
12
13    cout << "x = " << x << endl << "y = " << y << endl;
14    y = 7;
15    cout << "x = " << x << endl << "y = " << y << endl;
16
17    return 0;  // indicates successful termination
18
19 } // end main
```

Lỗi biên dịch – tham chiếu không được khởi tạo.

```
Borland C++ command-line compiler error message:
 Error E2304 Fig03_22.cpp 11: Reference variable 'y' must be
   initialized- in function main()

Microsoft Visual C++ compiler error message:
 D:\cpphtp4_examples\ch03\Fig03_22.cpp(11) : error C2530: 'y' :
   references must be initialized
```

fig03_22.cpp (1 of 1)

fig03_22.cpp (1 of 1)

3.17 Tham chiếu và Tham số là tham chiếu

- Gọi bằng giá trị - Call by value
  - Bản sao của dữ liệu được truyền cho hàm
  - Thay đổi đối với bản sao không ảnh hưởng tới dữ liệu gốc
  - Ngăn chặn các hiệu ứng phụ không mong muốn
- Gọi bằng tham chiếu - Call by reference
  - Hàm có thể truy nhập trực tiếp tới dữ liệu gốc
  - Các thay đổi thể hiện tại dữ liệu gốc

3.17 Tham chiếu và Tham số là tham chiếu

- Tham số tham chiếu - Reference parameter
  - Ý nghĩa: Là biệt danh (alias) của biến được truyền vào lời gọi hàm
    - 'truyền tham số bằng tham chiếu' hay 'truyền tham chiếu'
  - Cú pháp: Đặt ký hiệu **&** sau kiểu dữ liệu tại prototype của hàm
    - **`void myFunction( int &data )`**
    - có nghĩa "**`data`** là một tham chiếu tới một biến kiểu **`int`**"
  - dạng của lời gọi hàm không thay đổi
    - tuy nhiên dữ liệu gốc khi được truyền bằng tham chiếu có thể bị sửa đổi
- Con trỏ (chương 5)
  - Một cách truyền tham chiếu khác

```
1  // Fig. 3.20: fig03_20.cpp
2  // Comparing pass-by-value and pass-by-reference
3  // with references.
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8
9  int squareByValue( int );       // func
10 void squareByReference( int & );  // function prototype
11
12 int main()
13 {
14    int x = 2;
15    int z = 4;
16
17    // demonstrate squareByValue
18    cout << "x = " << x << " before squareByValue\n";
19    cout << "Value returned by squareByValue: "
20         << squareByValue( x ) << endl;
21    cout << "x = " << x << " after squareByValue\n" << endl;
22
```

Lưu ý ký hiệu **&** có nghĩa truyền tham chiếu (pass-by-reference).

fig03_20.cpp (1 of 2)


```
23    // demonstrate squareByReference
24    cout << "z = " << z << " before squareByReference" << endl;
25    squareByReference( z );
26    cout << "z = " << z << " after squareByReference" << endl;
27
28    return 0;  // indicates successful termination
29 } // end main
30
31 // squareByValue multiplies number by itself, stores the
32 // result in number and returns the new value of number
33 int squareByValue( int number )
34 {
35    return number *= number;  // caller's argument not modified
36
37 } // end function squareByValue
38
39 // squareByReference multiplies numberRef by itself and
40 // stores the result in the variable to which numberRef
41 // refers in function main
42 void squareByReference( int &numberRef )
43 {
44    numberRef *= numberRef;   // caller's argument modified
45
46 } // end function squareByReference
```

thay đổi **number**, nhưng đối số gốc (**x**) không bị thay đổi.

thay đổi **numberRef**, một biệt danh của đối số gốc. Do đó, **z** bị thay đổi.

fig03_20.cpp (2 of 2)

```
x = 2 before squareByValue
Value returned by squareByValue: 4
x = 2 after squareByValue

z = 4 before squareByReference
z = 16 after squareByReference
```

fig03_20.cpp
output (1 of 1)

3.18 Các đối số mặc định

- Lời gọi hàm với các tham số được bỏ qua
  - Nếu không đủ số tham số, các vị trí ở bên phải nhất sẽ được nhận giá trị mặc định của chúng
  - Các giá trị mặc định
    - Có thể là hằng, biến toàn cục, hoặc các lời gọi hàm
- Đặt các giá trị mặc định tại function prototype

```
int myFunction( int x = 1, int y = 2, int z = 3 );
```

  - **`myFunction(3)`**
    - **`x = 3`**, **`y`** và **`z`** nhận giá trị mặc định (bên phải nhất)
  - **`myFunction(3, 5)`**
    - **`x = 3`**, **`y = 5`** còn **`z`** nhận giá trị mặc định

```
1  // Fig. 3.23: fig03_23.cpp
2  // Using default arguments.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  // function prototype that specifies default arguments
9  int boxVolume( int length = 1, int width = 1, int height = 1 );
10
11 int main()
12 {
13    // no arguments--use default values for all dimensions
14    cout << "The default box volume is: " << boxVolume();
15
16    // specify length; default width and height
17    cout << "\n\nThe volume of a box with length 10,\n"
18         << "width 1 and height 1 is: " << boxVolume( 10 );
19
20    // specify length and width; default height
21    cout << "\n\nThe volume of a box with length 10,\n"
22         << "width 5 and height 1 is: " << boxVolume( 10, 5 );
23
```

Các giá trị mặc định được đặt trong function prototype.

Các lời gọi hàm thiếu một số đối số – Các đối số bên phải nhất nhận giá trị mặc định.

fig03_23.cpp (1 of 2)


```
24    // specify all arguments
25    cout << "\n\nThe volume of a box with length 10,\n"
26         << "width 5 and height 2 is: " << boxVolume( 10, 5, 2 )
27         << endl;
28
29    return 0;  // indicates successful termination
30
31 } // end main
32
33 // function boxVolume calculates the volume of a box
34 int boxVolume( int length, int width, int height )
35 {
36    return length * width * height;
37
38 } // end function boxVolume
```

fig03_23.cpp (2 of 2)

fig03_23.cpp output (1 of 1)

```
The default box volume is: 1

The volume of a box with length 10,
width 1 and height 1 is: 10

The volume of a box with length 10,
width 5 and height 1 is: 50

The volume of a box with length 10,
width 5 and height 2 is: 100
```

3.19 Toán tử phạm vi đơn

- Toán tử phạm vi đơn (`::`)
  Unitary Scope Resolution Operator
  - Dùng để truy nhập biến toàn cục nếu biến địa phương có cùng tên
  - Không cần thiết nếu các tên biến khác nhau
  - Cách dùng **`::tên_biến`**

    ```
    y = ::x + 3;
    ```
  - Nên tránh dùng các tên giống nhau cho các biến địa phương và toàn cục

fig03_24.cpp (1 of 2)

```
1  // Fig. 3.24: fig03_24.cpp
2  // Using the unary scope resolution operator.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <iomanip>
9
10 using std::setprecision;
11
12 // define global constant PI
13 const double PI = 3.14159265358979;
14
15 int main()
16 {
17    // define local constant PI
18    const float PI = static_cast< float >( ::PI );
19
20    // display values of local and global PI constants
21    cout << setprecision( 20 )
22         << "  Local float value of PI = " << PI
23         << "\nGlobal double value of PI = " << ::PI << endl;
24
25    return 0;  // indicates successful termination
```

Truy nhập **`PI`** toàn cục với **`::PI`**.

Chuyển đổi **`PI`** toàn cục thành một giá trị **`float`** cho **`PI`** địa phương. Ví dụ này cho thấy sự khác nhau giữa **`float`** và **`double`**.

```
26
27  } // end main
```

```
Borland C++ command-line compiler output:
  Local float value of PI = 3.141592741012573242
Global double value of PI = 3.141592653589790007

Microsoft Visual C++ compiler output:
  Local float value of PI = 3.1415927410125732
Global double value of PI = 3.14159265358979
```

fig03_24.cpp (2 of 2)

fig03_24.cpp output (1 of 1)

3.20 Chồng hàm

- Chồng hàm - Function overloading
  - Các hàm có cùng tên nhưng khác nhau về tham số
  - Nên thực hiện các nhiệm vụ tương tự
    - ví dụ, hàm tính bình phương cho **int** và hàm tính bình phương cho **float**

```
int square( int x) {return x * x;}
float square(float x) { return x * x; }
```

- Các hàm chồng phân biệt nhau bởi chữ ký
  - Dựa vào tên và kiểu tham số (xét cả thứ tự)
  - Trình biên dịch đảm bảo gọi đúng hàm chồng được yêu cầu

```
1  // Fig. 3.25: fig03_25.cpp
2  // Using overloaded functions.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  // function square for int values
9  int square( int x )
10 {
11    cout << "Called square with int argument: " << x << endl;
12    return x * x;
13
14 } // end int version of function square
15
16 // function square for double values
17 double square( double y )
18 {
19    cout << "Called square with double argument: " << y << endl;
20    return y * y;
21
22 } // end double version of function square
23
```

Các hàm chồng có cùng tên nhưng có tham số khác nhau

fig03_25.cpp (1 of 2)


```
24 int main()
25 {
26    int intResult = square( 7 );         // calls int version
27    double doubleResult = square( 7.5 ); // calls double version
28
29    cout << "\nThe square of integer 7 is " << intResult
30         << "\nThe square of double 7.5 is " << doubleResult
31         << endl;
32
33    return 0;  // indicates successful termination
34
35 } // end main
```

Tùy theo đối số được truyền vào (**int** hoặc **double**) để gọi hàm thích hợp.

fig03_25.cpp (2 of 2)

fig03_25.cpp output (1 of 1)

```
Called square with int argument: 7
Called square with double argument: 7.5

The square of integer 7 is 49
The square of double 7.5 is 56.25
```

3.21 Khuôn mẫu hàm - Function Template

- Cách ngắn gọn để tạo các hàm chồng
  - Sinh các hàm riêng biệt cho các kiểu dữ liệu khác nhau
- Cú pháp
  - Bắt đầu bằng từ khóa **`template`**
  - các tham số kiểu hình thức trong cặp ngoặc **<>**
    - **`typename`** hoặc **`class`** (đồng nghĩa) đặt trước mỗi tham số kiểu
    - là đại diện cho các kiểu cài sẵn (ví dụ **`int`**) hoặc các kiểu dữ liệu người dùng
    - chỉ ra các kiểu dữ liệu cho đối số hàm, giá trị trả về, biến địa phương
  - Hàm được định nghĩa như bình thường, ngoại trừ việc sử dụng các kiểu hình thức

3.21 Khuôn mẫu hàm

- Ví dụ

```
template < class T > // or template< typename T >
T square( T value1 )
{
   return value1 * value1;
}
```

  - **`T`** là một kiểu hình thức, được dùng làm tham số kiểu
    - hàm trên trả về giá trị thuộc cùng kiểu với tham số
  - Tại lời gọi hàm, T được thay bằng kiểu dữ liệu thực
    - Nếu là **`int`**, mọi **`T`** trở thành **`int`**

```
int x;
int y = square(x);
```

```
1  // Fig. 3.27: fig03_27.cpp
2  // Using a function template.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  // definition of function template maximum
10 template < class T >  // or template < typename T >
11 T maximum( T value1, T value2, T value3 )
12 {
13    T max = value1;
14
15    if ( value2 > max )
16       max = value2;
17
18    if ( value3 > max )
19       max = value3;
20
21    return max;
22
23 } // end function template maximum
24
```

Tham số kiểu hình thức **T** là đại diện của kiểu dữ liệu được kiểm tra trong hàm **maximum**.

maximum mong đợi mọi tham số đều thuộc cùng một kiểu dữ liệu.

fig03_27.cpp (1 of 3)

```
25 int main()
26 {
27    // demonstrate maximum with int values
28    int int1, int2, int3;
29
30    cout << "Input three integer values: ";
31    cin >> int1 >> int2 >> int3;
32
33    // invoke int version of maximum
34    cout << "The maximum integer value is: "
35         << maximum( int1, int2, int3 );
36
37    // demonstrate maximum with double values
38    double double1, double2, double3;
39
40    cout << "\n\nInput three double values: ";
41    cin >> double1 >> double2 >> double3;
42
43    // invoke double version of maximum
44    cout << "The maximum double value is: "
45         << maximum( double1, double2, double3 );
46
```

maximum được gọi với nhiều kiểu dữ liệu.

fig03_27.cpp (2 of 3)

```
47     // demonstrate maximum with char values
48     char char1, char2, char3;
49
50     cout << "\n\nInput three characters: ";
51     cin >> char1 >> char2 >> char3;
52
53     // invoke char version of maximum
54     cout << "The maximum character value is: "
55          << maximum( char1, char2, char3 )
56          << endl;
57
58     return 0;  // indicates successful termination
59
60  } // end main
```

fig03_27.cpp (3 of 3)

fig03_27.cpp output (1 of 1)

```
Input three integer values: 1 2 3
The maximum integer value is: 3

Input three double values: 3.3 2.2 1.1
The maximum double value is: 3.3

Input three characters: A C B
The maximum character value is: C
```

Ngôn ngữ lập trình C++

Chương 4 – Mảng

Chương 4 – Mảng

Đề mục

4.1 Giới thiệu

- Mảng (array)
  - Cấu trúc của những phần tử dữ liệu có liên quan
  - Thực thể tĩnh (giữ nguyên kích thước trong suốt chương trình)
- Một vài loại mảng
  - mảng dựa vào con trỏ (Pointer-based arrays) (C-like)
  - mảng là đối tượng (Arrays as objects) (C++)

4.2 Mảng

- Mảng
  - Tập hợp các vùng nhớ liên tiếp
  - Cùng tên, cùng kiểu (**`int`**, **`char`**, ...)
- Truy nhập đến 1 phần tử
  - Chỉ ra tên mảng và vị trí - position (chỉ số - index)
  - Cú pháp: **`tên_mảng[ chỉ_số ]`**
  - Phần tử đầu tiên ở vị trí 0
- Mảng c có n phần tử

  `c[ 0 ], c[ 1 ]` ... **`c[ n - 1 ]`**
  - Phần tử thứ N ở vị trí thứ N-1

4.2 Mảng

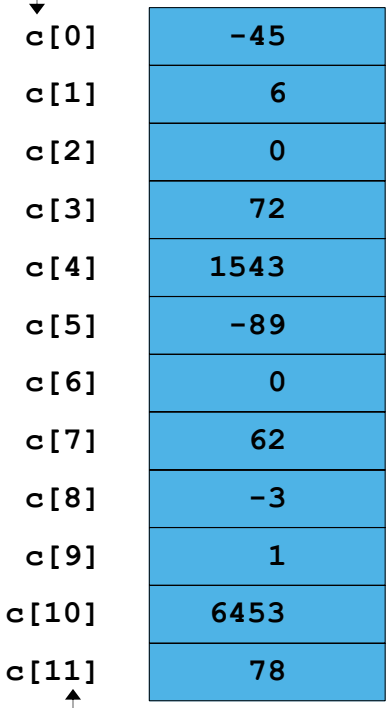
- Phần tử của mảng cũng như các biến khác
  - Gán giá trị và in mảng số nguyên **c**

```
c[ 0 ] =  3;
cout << c[ 0 ];
```

- Có thể sử dụng các phép toán trong cặp ngoặc vuông

`c[ 5 - 2 ]` cũng giống `c[3]`

Tên mảng
(Lưu ý rằng mọi phần tử của mảng này đều có cùng tên, **c**)



Chỉ số của phần tử trong mảng **c**

4.3 Khai báo mảng

- Khi khai báo mảng, chỉ rõ
  - Tên
  - Kiểu của mảng
    - Bất cứ kiểu dữ liệu nào
  - Số phần tử
  - *type arrayName* **[** *arraySize* **];**

```
int c[ 10 ];  // mảng của 10 số nguyên
float d[ 3284 ]; // mảng của 3284 số thực
```

- Khai báo nhiều mảng cùng kiểu
  - Sử dụng dấu phẩy như với các biến bình thường

```
int b[ 100 ], x[ 27 ];
```


4.4 Ví dụ về sử dụng mảng

- Khởi tạo mảng
  - Dùng vòng lặp khởi tạo từng phần tử
  - Khởi tạo cả danh sách
    - Chỉ rõ từng phần tử khi khai báo mảng

```
int n[ 5 ] = { 1, 2, 3, 4, 5 };
```

    - Nếu trong danh sách không có đủ số giá trị khởi tạo, các phần tử ở bên phải nhất sẽ nhận giá trị 0
    - Nếu danh sách thừa sẽ gây lỗi cú pháp
  - Khởi tạo giá trị bằng 0 cho tất cả các phần tử

```
int n[ 5 ] = { 0 };
```

  - Nếu không khai báo kích thước mảng, kích thước của danh sách các giá trị khởi tạo sẽ quyết định kích thước mảng

```
int n[] = { 1, 2, 3, 4, 5 };
```

    - Có 5 giá trị khởi tạo, do đó mảng có 5 phần tử
    - Nếu không khai báo kích thước mảng thì phải khởi tạo khi khai báo

fig04_03.cpp
(1 of 2)

```
1  // Fig. 4.3: fig04_03.cpp
2  // Initializing an array.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <iomanip>
9
10 using std::setw;
11
12 int main()
13 {
14    int n[ 10 ];  // n is an array of 10 intege
15
16    // initialize elements of array n to 0
17    for ( int i = 0; i < 10; i++ )
18       n[ i ] = 0;   // set element at location i to 0
19
20    cout << "Element" << setw( 13 ) << "Value" << endl;
21
22    // output contents of array n in tabular format
23    for ( int j = 0; j < 10; j++ )
24       cout << setw( 7 ) << j << setw( 13 ) << n[ j ] << endl;
25
```

Khai báo mảng 10 phần tử số nguyên.

Khởi tạo mảng bằng vòng lặp **for**. Chú ý rằng mảng gồm các phần tử từ **n[0]** đến **n[9]**.

```
26     return 0;  // indicates successful termination
27
28  } // end main
```

```
Element         Value
      0             0
      1             0
      2             0
      3             0
      4             0
      5             0
      6             0
      7             0
      8             0
      9             0
```

fig04_03.cpp (2 of 2)

fig04_03.cpp output (1 of 1)

fig04_04.cpp
(1 of 1)

```
1  // Fig. 4.4: fig04_04.cpp
2  // Initializing an array with a declaration.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <iomanip>
9
10 using std::setw;
11
12 int main()
13 {
14    // use initializer list to initialize array n
15    int n[ 10 ] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
16
17    cout << "Element" << setw( 13 ) << "Value" << endl;
18
19    // output contents of array n in tabular format
20    for ( int i = 0; i < 10; i++ )
21       cout << setw( 7 ) << i << setw( 13 ) << n[ i ] << endl;
22
23    return 0;  // indicates successful termination
24
25 } // end main
```

Lưu ý cách dùng danh sách khởi tạo cho mảng.

```
Element         Value
      0            32
      1            27
      2            64
      3            18
      4            95
      5            14
      6            90
      7            70
      8            60
      9            37
```

fig04_04.cpp
output (1 of 1)

4.4 Ví dụ về sử dụng mảng

- Kích thước của mảng
  - Có thể được xác định bằng hằng số (**`const`**)
    - **`const int size = 20;`**
  - Hằng số không thể thay đổi
  - Hằng phải được khởi tạo khi khai báo
  - Còn được gọi là "named constant" (giá trị được đặt tên) hoặc "read-only variable" (biến chỉ đọc)

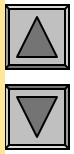


fig04_05.cpp
(1 of 2)

```
1  // Fig. 4.5: fig04_05.cpp
2  // Initialize array s to the even integers from 2 to 20.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <iomanip>
9
10 using std::setw;
11
12 int main()
13 {
14    // constant variable can be used
15    const int arraySize = 10;
16
17    int s[ arraySize ];  // array s has 10 elements
18
19    for ( int i = 0; i < arraySize; i++ )  // s
20       s[ i ] = 2 + 2 * i;
21
22    cout << "Element" << setw( 13 ) << "Value"
23
```

Chú ý từ khoá **const**. Chỉ có các biến **const** được dùng để khai báo kích thước mảng.

Chương trình dễ thay đổi hơn khi ta dùng hằng (**const**) cho kích thước của mảng.
Ta có thể thay đổi **arraySize**, và tất cả các vòng lặp vẫn hoạt động bình thường (nếu không, ta phải sửa mọi vòng lặp trong chương trình).

```
24     // output contents of array s in tabular format
25     for ( int j = 0; j < arraySize; j++ )
26        cout << setw( 7 ) << j << setw( 13 ) << s[ j ] << endl;
27
28     return 0;  // indicates successful termination
29
30  } // end main
```

fig04_05.cpp
(2 of 2)

fig04_05.cpp
output (1 of 1)

```
Element         Value
      0             2
      1             4
      2             6
      3             8
      4            10
      5            12
      6            14
      7            16
      8            18
      9            20
```

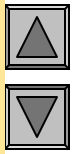


fig04_06.cpp
(1 of 1)

fig04_06.cpp
output (1 of 1)

```
1   // Fig. 4.6: fig04_06.cpp
2   // Using a properly initialized constant variable.
3   #include <iostream>
4
5   using std::cout;
6   using std::endl;
7
8   int main()
9   {
10     const int x = 7;  // initialized constant variable
11
12     cout << "The value of constant variable x is: "
13          << x << endl;
14
15     return 0;  // indicates successful termination
16
17  } // end main
```

Khởi tạo hằng

```
The value of constant variable x is: 7
```

```
1   // Fig. 4.7: fig04_07.cpp
2   // A const object must be initialized.
3
4   int main()
5   {
6      const int x; // Error: x must be initialized
7
8      x = 7;       // Error: cannot modify a const variable
9
10     return 0;    // indicates successful termination
11
12  } // end main
```

Lỗi cú pháp do không khởi tạo hằng.
Sửa giá trị của hằng cũng là một lỗi.

fig04_07.cpp
(1 of 1)

fig04_07.cpp
output (1 of 1)

```
d:\cpphtp4_examples\ch04\Fig04_07.cpp(6) : error C2734: 'x' :
   const object must be initialized if not extern
d:\cpphtp4_examples\ch04\Fig04_07.cpp(8) : error C2166:
   l-value specifies const object
```

```
1  // Fig. 4.8: fig04_08.cpp
2  // Compute the sum of the elements of the array.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int main()
9  {
10    const int arraySize = 10;
11
12    int a[ arraySize ] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
13
14    int total = 0;
15
16    // sum contents of array a
17    for ( int i = 0; i < arraySize; i++ )
18       total += a[ i ];
19
20    cout << "Total of array element values is " << total << endl;
21
22    return 0;  // indicates successful termination
23
24 } // end main
```

```
Total of array element values is 55
```

)4_08.cpp
)f 1)

)4_08.cpp
)put (1 of 1)

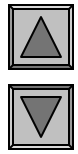


fig04_09.cpp
(1 of 2)

```
1  // Fig. 4.9: fig04_09.cpp
2  // Histogram printing program.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <iomanip>
9
10 using std::setw;
11
12 int main()
13 {
14    const int arraySize = 10;
15    int n[ arraySize ] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
16
17    cout << "Element" << setw( 13 ) << "Value"
18         << setw( 17 ) << "Histogram" << endl;
19
```

```
Element       Value         Histogram
      0          19         ********************
      1           3         ***
      2          15         ***************
      3           7         *******
      4          11         ***********
      5           9         *********
      6          13         *************
      7           5         *****
      8          17         *****************
      9           1         *
```

```
20    // for each element of array n, output a bar in histogram
21    for ( int i = 0; i < arraySize; i++ ) {
22       cout << setw( 7 ) << i << setw( 13 )
23            << n[ i ] << setw( 9 );
24
25       for ( int j = 0; j < n[ i ]; j++ )   // print one bar
26          cout << '*';
27
28       cout << endl;  // start next line of output
29
30    } // end outer for structure
31
32    return 0;  // indicates successful termination
33
34 } // end main
```

In số dấu sao (*) tương ứng với giá trị của phần tử **n[i]**.

fig04_09.cpp
output (1 of 1)

```
Element         Value         Histogram
      0            19         ********************
      1             3         ***
      2            15         ***************
      3             7         *******
      4            11         ***********
      5             9         *********
      6            13         *************
      7             5         *****
      8            17         *****************
      9             1         *
```

fig04_10.cpp
(1 of 2)

```
1  // Fig. 4.10: fig04_10.cpp
2  // Roll a six-sided die 6000 times.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <iomanip>
9
10 using std::setw;
11
12 #include <cstdlib>
13 #include <ctime>
14
15 int main()
16 {
17    const int arraySize = 7;
18    int frequency[ arraySize ] = { 0 };
19
20    srand( time( 0 ) );  // seed random-number gene
21
22    // roll die 6000 times
23    for ( int roll = 1; roll <= 6000; roll++ )
24       ++frequency[ 1 + rand() % 6 ]; // replaces 20-line switch
25                                      // of Fig. 3.8
```

Viết lại một chương trình cũ. Một mảng được sử dụng thay cho 6 biến thường, và các phần tử dễ dàng cập nhật hơn (không cần sử dụng **`switch`**).

Dòng lệnh này tạo ra một số trong khoảng 1 đến 6 và tăng phần tử **`frequency[]`** có chỉ số đó.

```
26
27     cout << "Face" << setw( 13 ) << "Frequency" << endl;
28
29     // output frequency elements 1-6 in tabular format
30     for ( int face = 1; face < arraySize; face++ )
31        cout << setw( 4 ) << face
32             << setw( 13 ) << frequency[ face ] << endl;
33
34     return 0;  // indicates successful termination
35
36  } // end main
```

fig04_10.cpp
(2 of 2)

fig04_10.cpp
output (1 of 1)

```
Face    Frequency
   1         1003
   2         1004
   3          999
   4          980
   5         1013
   6         1001
```

g04_11.cpp
of 2)

```
1   // Fig. 4.11: fig04_11.cpp   ***modified***
2   // Student mark statistic program.
3   #include <iostream>
4
5   using std::cout;
6   using std::endl;
7
8   #include <iomanip>
9
10  using std::setw;
11
12  int main()
13  {
14     // define array sizes
15     const int markSize = 40;   // size of array of marks
16     const int frequencySize = 11;  // size of array frequency
17
18     // place student marks in array of marks
19     int marks[ markSize ] = { 1, 2, 6, 4, 8, 5, 9, 7, 8,
20         10, 1, 6, 3, 8, 6, 10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7,
21         5, 6, 6, 5, 6, 7, 5, 6, 4, 8, 6, 8, 10 };
22
23     // initialize frequency counters to 0
24     int frequency[ frequencySize ] = { 0 };
25
```

4_11.cpp
2)

```
26     // for each student's mark, select value of an element of array
27     // responses and use that value as subscript in array
28     // frequency to determine element to increment
29     for ( int student = 0; student < markSize; student++ )
30        ++frequency[ marks[student] ];
31
32     // display results
33     cout << "Rating" << setw( 17 ) << "Frequency" << endl;
34
35     // output frequencies in tabular format
36     for ( int rating = 1; rating < frequencySize; rating++ )
37        cout << setw( 6 ) << rating
38             << setw( 17 ) << frequency[ rating ] << endl;
39
40     return 0;  // indicates successful termination
41
42  } // end main
```

marks[student] là điểm (từ 1 đến 10). Giá trị này quyết định chỉ số của phần tử **frequency[]** cần tăng.

Rating	Frequency
1	2
2	2
3	2
4	2
5	5
6	11
7	5
8	7
9	1
10	3

4.4 Ví dụ về sử dụng mảng

- Xâu - string (xem thêm ở chương 5)
  - Mảng của các ký tự
  - Mọi xâu đều kết thúc với ký tự **`null`** (**`'\0'`**)
  - Ví dụ
    - **`char string1[] = "hello";`**
      - Ký tự **`null`** tự động được thêm vào, xâu có 6 phần tử
    - **`char string1[] = { 'h', 'e', 'l', 'l', 'o', '\0' };`**
  - Chỉ số cũng giống như đối với mảng

    `String1[ 0 ]` bằng **`'h'`**

    `string1[ 2 ]` bằng **`'l'`**

4.4 Ví dụ về sử dụng mảng

- Nhập từ bàn phím bằng **`cin`**

```
char string2[ 10 ];
cin >> string2;
```

  - Ghi dữ liệu vào của người dùng vào xâu
    - Dừng lại ở ký tự trắng đầu tiên (tab, newline, blank…)
    - Thêm vào ký tự **`null`**
  - Nếu nhập quá nhiều, dữ liệu sẽ tràn mảng
    - Ta cần phải tránh điều này (mục 5.12 sẽ giải thích phương pháp)
- In xâu
  - **`cout << string2 << endl;`**
    - Không sử dụng được với các mảng có kiểu dữ liệu khác
  - In các ký tự cho đến khi gặp **`null`**

04_12.cpp
of 2)

```
1   // Fig. 4_12: fig04_12.cpp
2   // Treating character arrays as strings.
3   #include <iostream>
4
5   using std::cout;
6   using std::cin;
7   using std::endl;
8
9   int main()
10  {
11     char string1[ 20 ],                  // reserves 20 characters
12     char string2[] = "string literal"; // reserves 15 characters
13
14     // read string from user into array string2
15     cout << "Enter the string \"hello there\": ";
16     cin >> string1;  // reads "hello" [space terminates input]
17
18     // output strings
19     cout << "string1 is: " << string1
20          << "\nstring2 is: " << string2;
21
22     cout << "\nstring1 with spaces between characters is:\n";
23
```

Hai cách khác nhau để khai báo xâu. **string2** được khởi tạo và kích thước được xác định tự động.

Ví dụ về đọc xâu từ bàn phím và in ra.

```
24     // output characters until null character is reached
25     for ( int i = 0; string1[ i ] != '\0'; i++ )
26        cout << string1[ i ] << ' ';
27
28     cin >> string1;  // reads "there"
29     cout << "\nstring1 is: " << string1 << endl;
30
31     return 0;  // indicates successful termination
32
33  } // end main
```

Có thể truy nhập xâu giống như đối với mảng. Vòng lặp kết thúc khi gặp ký tự **null**.

p

fig04_12.cpp
output (1 of 1)

```
Enter the string "hello there": hello there
string1 is: hello
string2 is: string literal
string1 with spaces between characters is:
h e l l o
string1 is: there
```

4.4 Ví dụ về sử dụng mảng

- Kiểu lưu trữ tĩnh – static storage (chương 3)
  - Nếu là **`static`**, các biến địa phương lưu lại giá trị giữa các lần gọi hàm
  - chỉ được nhìn thấy trong thân hàm
  - Có thể khai báo mảng địa phương là static
    - được khởi tạo về 0

```
static int array[3];
```

- Nếu không phải static
  - Được tạo (và huỷ) tại mỗi lần gọi hàm


fig04_13.cpp
(1 of 3)

```
1  // Fig. 4.13: fig04_13.cpp
2  // Static arrays are initialized to zero.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  void staticArrayInit( void );      // function prototype
9  void automaticArrayInit( void );   // function prototype
10
11 int main()
12 {
13    cout << "First call to each function:\n";
14    staticArrayInit();
15    automaticArrayInit();
16
17    cout << "\n\nSecond call to each function:\n";
18    staticArrayInit();
19    automaticArrayInit();
20    cout << endl;
21
22    return 0;  // indicates successful termination
23
24 } // end main
25
```

```
26 // function to demonstrate a static local array
27 void staticArrayInit( void )
28 {
29    // initializes elements to 0 first time function is called
30    static int array1[ 3 ];
31
32    cout << "\nValues on entering staticArrayInit:\n";
33
34    // output contents of array1
35    for ( int i = 0; i < 3; i++ )
36       cout << "array1[" << i << "] = " << array1[ i ] << "  ";
37
38    cout << "\nValues on exiting staticArrayInit:\n";
39
40    // modify and output contents of array1
41    for ( int j = 0; j < 3; j++ )
42       cout << "array1[" << j << "] = "
43            << ( array1[ j ] += 5 ) << "  ";
44
45 } // end function staticArrayInit
46
```

Mảng static, khởi tạo về 0 tại lần gọi hàm đầu tiên.

Dữ liệu trong mảng bị thay đổi, các thay đổi được bảo toàn.

fig04_13.cpp (2 of 3)

```
47 // function to demonstrate an automatic local array
48 void automaticArrayInit( void )
49 {
50    // initializes elements each time function is called
51    int array2[ 3 ] = { 1, 2, 3 };
52
53    cout << "\n\nValues on entering automaticArrayInit:\n";
54
55    // output contents of array2
56    for ( int i = 0; i < 3; i++ )
57       cout << "array2[" << i << "] = " << array2[ i ] << "   ";
58
59    cout << "\nValues on exiting automaticArrayIn
60
61    // modify and output contents of array2
62    for ( int j = 0; j < 3; j++ )
63       cout << "array2[" << j << "] = "
64            << ( array2[ j ] += 5 ) << "   ";
65
66 } // end function automaticArrayInit
```

Mảng automatic, được tạo lại tại mỗi lần gọi hàm.

Tuy mảng bị thay đổi, nó sẽ bị huỷ khi hàm kết thúc và thayt đổi trong dữ liệu sẽ bị mất.

fig04_13.cpp (3 of 3)

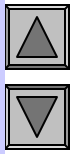


fig04_13.cpp output (1 of 1)

```
First call to each function:

Values on entering staticArrayInit:
array1[0] = 0  array1[1] = 0  array1[2] = 0
Values on exiting staticArrayInit:
array1[0] = 5  array1[1] = 5  array1[2] = 5

Values on entering automaticArrayInit:
array2[0] = 1  array2[1] = 2  array2[2] = 3
Values on exiting automaticArrayInit:
array2[0] = 6  array2[1] = 7  array2[2] = 8

Second call to each function:

Values on entering staticArrayInit:
array1[0] = 5  array1[1] = 5  array1[2] = 5
Values on exiting staticArrayInit:
array1[0] = 10  array1[1] = 10  array1[2] = 10

Values on entering automaticArrayInit:
array2[0] = 1  array2[1] = 2  array2[2] = 3
Values on exiting automaticArrayInit:
array2[0] = 6  array2[1] = 7  array2[2] = 8
```

4.5 Truyền tham số cho hàm

- Dùng tên mảng, bỏ cặp ngoặc vuông
  - Truyền mảng **myArray** cho hàm **myFunction**

```
int myArray[ 24 ];
myFunction( myArray, 24 );
```

  - Kích thước mảng thường được truyền, nhưng không nhất thiết
    - Có ích khi dùng để duyệt tất cả các phần tử
- Mảng được truyền bằng tham chiếu (passed-by-reference)
  - Hàm có thể thay đổi dữ liệu gốc của mảng
  - Tên mảng có giá trị bằng địa chỉ của phần tử đầu tiên
    - Hàm biết mảng được lưu ở đâu.
    - Hàm có thể sửa đổi dữ liệu ghi trong mảng
- Các phần tử mảng được truyền bằng giá trị (passed-by-value)
  - Như các biến thông thường
  - **square(myArray[3]);**

4.5 Truyền tham số cho hàm

- Các hàm dùng mảng làm đối số
  - Function prototype
    - `void modifyArray( int b[], int arraySize );`
    - `void modifyArray( int [], int );`
      - Trong prototype, tên không bắt buộc
    - cả hai hàm lấy đối số là một mảng số nguyên và 1 số nguyên
  - Không ghi cần kích thước mảng trong cặp ngoặc
    - Trình biên dịch bỏ qua
  - Nếu khai báo 1 tham số là **`const`**
    - đối số đó sẽ không thể bị thay đổi (chương trình dịch báo lỗi)
    - `void doNotModify( const int [] );`

ig04_14.cpp
1 of 3)

```
1  // Fig. 4.14: fig04_14.cpp
2  // Passing arrays and individual array elements to functions.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <iomanip>
9
10 using std::setw;
11
12 void modifyArray( int [], int );  // appears strange
13 void modifyElement( int );
14
15 int main()
16 {
17    const int arraySize = 5;                 // size of array a
18    int a[ arraySize ] = { 0, 1, 2, 3, 4 };  // initialize a
19
20    cout << "Effects of passing entire array by reference:"
21         << "\n\nThe values of the original array are:\n";
22
23    // output original array
24    for ( int i = 0; i < arraySize; i++ )
25       cout << setw( 3 ) << a[ i ];
```

Cú pháp cho mảng trong danh sách tham số

```
27     cout << endl;
28
29     // pass array a to modifyArray by reference
30     modifyArray( a, arraySize );
31
32     cout << "The values of the modified array are:\n";
33
34     // output modified array
35     for ( int j = 0; j < arraySize; j++ )
36        cout << setw( 3 ) << a[ j ];
37
38     // output value of a[ 3 ]
39     cout << "\n\n\n"
40          << "Effects of passing array element by value:"
41          << "\n\nThe value of a[3] is " << a[ 3 ] << '\n';
42
43     // pass array element a[ 3 ] by value
44     modifyElement( a[ 3 ] );
45
46     // output value of a[ 3 ]
47     cout << "The value of a[3] is " << a[ 3 ] << endl;
48
49     return 0;  // indicates successful termination
50
51  } // end main
```

Truyền tên mảng (**a**) và kích thước cho hàm. Mảng truyền bằng tham chiếu

1 phần tử mảng được truyền bằng giá trị; giá trị phần tử gốc không thể bị thay đổi.

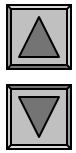
fig04_14.cpp
(2 of 3)

```
52
53 // in function modifyArray, "b" points to
54 // the original array "a" in memory
55 void modifyArray( int b[], int sizeOfArray )
56 {
57    // multiply each array element by 2
58    for ( int k = 0; k < sizeOfArray; k++ )
59       b[ k ] *= 2;
60
61 } // end function modifyArray
62
63 // in function modifyElement, "e" is a local copy of
64 // array element a[ 3 ] passed from main
65 void modifyElement( int e )
66 {
67    // multiply parameter by 2
68    cout << "Value in modifyElement is "
69         << ( e *= 2 ) << endl;
70
71 } // end function modifyElement
```

Tuy đặt tên là **b**, khi được gọi, mảng chỉ đến mảng **a**, nên hàm có thể thay đổi dữ liệu của **a**.

fig04_14.cpp (3 of 3)

Các phần tử đơn lẻ của mảng được truyền bằng giá trị, và các giá trị gốc không thể bị thay đổi.



```
Effects of passing entire array by reference:

The values of the original array are:
  0  1  2  3  4
The values of the modified array are:
  0  2  4  6  8


Effects of passing array element by value:

The value of a[3] is 6
Value in modifyElement is 12
The value of a[3] is 6
```

fig04_14.cpp
output (1 of 1)

```
1   // Fig. 4.15: fig04_15.cpp
2   // Demonstrating the const type qualifier.
3   #include <iostream>
4
5   using std::cout;
6   using std::endl;
7
8   void tryToModifyArray( const int [] );  // function prototype
9
10  int main()
11  {
12     int a[] = { 10, 20, 30 };
13
14     tryToModifyArray( a );
15
16     cout << a[ 0 ] << ' ' << a[ 1 ] << ' ' << a[ 2 ] << '\n';
17
18     return 0;  // indicates successful termination
19
20  } // end main
21
```

Tham số mảng được khai báo là **const**. Mảng không thể bị sửa đổi, kể cả khi nó được truyền bằng tham chiếu.

4_15.cpp
2)

```
22 // In function tryToModifyArray, "b" cannot be used
23 // to modify the original array "a" in main.
24 void tryToModifyArray( const int b[] )
25 {
26    b[ 0 ] /= 2;    // error
27    b[ 1 ] /= 2;    // error
28    b[ 2 ] /= 2;    // error
29
30 } // end function tryToModifyArray
```

fig04_15.cpp
(2 of 2)

4.6 Sắp xếp mảng

- Sắp xếp dữ liệu
  - Là một ứng dụng quan trọng
  - Hầu hết mọi cơ quan/tổ chức đều phải sắp xếp dữ liệu
    - Một khối lượng khổng lồ dữ liệu cần được sắp xếp
- Xếp nổi bọt (Bubble sort)
  - Duyệt mảng vài lần
  - So sánh cặp phần tử liên tiếp
    - Nếu thứ tự tăng (hoặc bằng nhau), không thay đổi gì
    - Nếu thứ tự giảm, tráo đổi hai phần tử
  - Lặp lại các bước trên cho mọi phần tử

4.6 Sắp xếp mảng

- Ví dụ:
  - Đi từ trái sang phải, và tráo các phần tử khi cần thiết
    - Một lần duyệt cho mỗi phần tử
  - Dãy gốc: 3 4 2 7 6
  - Lần duyệt 1: 3 2 4 6 7 (tráo đổi phần tử)
  - Lần duyệt 2: 2 3 4 6 7
  - Lần duyệt 3: 2 3 4 6 7 (không cần thay đổi)
  - Lần duyệt 4: 2 3 4 6 7
  - Lần duyệt 5: 2 3 4 6 7
  - Phần tử nhỏ "nổi" lên trên (như số 2 trong ví dụ)

4.6 Sắp xếp mảng

- Tráo đổi các biến

```
int x = 3, y = 4;
y = x;
x = y;
```

- Cái gì xảy ra?
  - Cả x và y đều là 3!
  - Cần có biến tạm
- Giải pháp

```
int x = 3, y = 4, temp = 0;
temp = x;  // temp là 3
x = y;     // x là 4
y = temp;  // y là 3
```


ig04_16.cpp
1 of 3)

```
1   // Fig. 4.16: fig04_16.cpp
2   // This program sorts an array's values into ascending order.
3   #include <iostream>
4
5   using std::cout;
6   using std::endl;
7
8   #include <iomanip>
9
10  using std::setw;
11
12  int main()
13  {
14     const int arraySize = 10;  // size of array a
15     int a[ arraySize ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
16     int hold;  // temporary location used to swap array elements
17
18     cout << "Data items in original order\n";
19
20     // output original array
21     for ( int i = 0; i < arraySize; i++ )
22        cout << setw( 4 ) << a[ i ];
23
```

```
24    // bubble sort
25    // loop to control number of passes
26    for ( int pass = 0; pass < arraySize - 1; pass++ )
27
28       // loop to control number of comparisons per pass
29       for ( int j = 0; j < arraySize - 1; j++ )
30
31          // compare side-by-side elements and swap them if
32          // first element is greater than second element
33          if ( a[ j ] > a[ j + 1 ] ) {
34             hold = a[ j ];
35             a[ j ] = a[ j + 1 ];
36             a[ j + 1 ] = hold;
37
38          } // end if
39
```

Duyệt 1 lần cho mỗi phần tử của mảng.

Nếu phần tử bên trái (chỉ số `j`) lớn hơn phần tử bên phải (chỉ số `j + 1`), thì ta tráo đổi chúng. Nhớ sử dụng biến tạm.

fig04_16.cpp (2 of 3)

```
40     cout << "\nData items in ascending order\n";
41
42     // output sorted array
43     for ( int k = 0; k < arraySize; k++ )
44        cout << setw( 4 ) << a[ k ];
45
46     cout << endl;
47
48     return 0;  // indicates successful termination
49
50  } // end main
```

fig04_16.cpp (3 of 3)

fig04_16.cpp output (1 of 1)

```
Data items in original order
   2   6   4   8  10  12  89  68  45  37
Data items in ascending order
   2   4   6   8  10  12  37  45  68  89
```

4.7 Ví dụ: sử dụng mảng để tính Mean, Median và Mode

- Mean
  - Giá trị trung bình (tổng/số phần tử)
- Median
  - Giá trị ở giữa dãy đã được sắp xếp
  - 1, 2, 3, 4, 5 (3 là median)
  - Nếu số phần tử là số chẵn, lấy trung bình của 2 số giữa
- Mode
  - Giá trị xuất hiện nhiều nhất
  - 1, 1, 1, 2, 3, 3, 4, 5 (1 là mode)

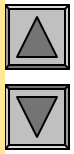
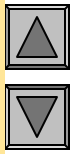


fig04_17.cpp
(1 of 8)

```
1  // Fig. 4.17: fig04_17.cpp
2  // This program introduces the topic of survey data analysis.
3  // It computes the mean, median, and mode of the data.
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8  using std::fixed;
9  using std::showpoint;
10
11 #include <iomanip>
12
13 using std::setw;
14 using std::setprecision;
15
16 void mean( const int [], int );
17 void median( int [], int );
18 void mode( int [], int [], int );
19 void bubbleSort( int[], int );
20 void printArray( const int[], int );
21
22 int main()
23 {
24    const int responseSize = 99;  // size of array responses
25
```

```
26    int frequency[ 10 ] = { 0 };  // initialize array frequency
27
28    // initialize array responses
29    int response[ responseSize ] =
30           { 6, 7, 8, 9, 8, 7, 8, 9, 8, 9,
31             7, 8, 9, 5, 9, 8, 7, 8, 7, 8,
32             6, 7, 8, 9, 3, 9, 8, 7, 8, 7,
33             7, 8, 9, 8, 9, 8, 9, 7, 8, 9,
34             6, 7, 8, 7, 8, 7, 9, 8, 9, 2,
35             7, 8, 9, 8, 9, 8, 9, 7, 5, 3,
36             5, 6, 7, 2, 5, 3, 9, 4, 6, 4,
37             7, 8, 9, 6, 8, 7, 8, 9, 7, 8,
38             7, 4, 4, 2, 5, 3, 8, 7, 5, 6,
39             4, 5, 6, 1, 6, 5, 7, 8, 7 };
40
41    // process responses
42    mean( response, responseSize );
43    median( response, responseSize );
44    mode( frequency, response, responseSize );
45
46    return 0;  // indicates successful termination
47
48 } // end main
49
```

)4_17.cpp
)f 8)



```
50 // calculate average of all response values
51 void mean( const int answer[], int arraySize )
52 {
53    int total = 0;
54
55    cout << "********\n  Mean\n********\n";
56
57    // total response values
58    for ( int i = 0; i < arraySize; i++ )
59       total += answer[ i ];
60
61    // format and output results
62    cout << fixed << setprecision( 4 );
63
64    cout << "The mean is the average value of the data\n"
65         << "items. The mean is equal to the total of\n"
66         << "all the data items divided by the number\n"
67         << "of data items (" << arraySize
68         << "). The mean value for\nthis run is: "
69         << total << " / " << arraySize << " = "
70         << static_cast< double >( total ) / arraySize
71         << "\n\n";
72
73 } // end function mean
74
```

fig04_17.cpp
(3 of 8)

Đổi sang **double** để được giá trị trung bình bằng số thực (thay vì giá trị nguyên).

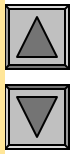
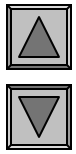


fig04_17.cpp
(4 of 8)

```
75  // sort array and determine median element's value
76  void median( int answer[], int size )
77  {
78     cout << "\n********\n Median\n********\n"
79          << "The unsorted array of responses is";
80
81     printArray( answer, size );  // output unsorted arr
82
83     bubbleSort( answer, size );  // sort array
84
85     cout << "\n\nThe sorted array is";
86     printArray( answer, size );  // output sorted array
87
88     // display median element
89     cout << "\n\nThe median is element " << size / 2
90          << " of\nthe sorted " << size
91          << " element array.\nFor this run the median is "
92          << answer[ size / 2 ] << "\n\n";
93
94  } // end function median
95
```

Sắp xếp mảng bằng cách truyền nó cho một hàm. Bảo vệ tính modun của chương trình



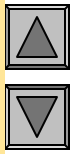
```
96  // determine most frequent response
97  void mode( int freq[], int answer[], int size )
98  {
99     int largest = 0;    // represents largest frequency
100    int modeValue = 0;  // represents most frequent response
101
102    cout << "\n********\n  Mode\n********\n";
103
104    // initialize frequencies to 0
105    for ( int i = 1; i <= 9; i++ )
106       freq[ i ] = 0;
107
108    // summarize frequencies
109    for ( int j = 0; j < size; j++ )
110       ++freq[ answer[ j ] ];
111
112    // output headers for result columns
113    cout << "Response" << setw( 11 ) << "Frequency"
114         << setw( 19 ) << "Histogram\n\n" << setw( 55 )
115         << "1    1    2    2\n" << setw( 56 )
116         << "5    0    5    0    5\n\n";
117
```

fig04_17.cpp
(5 of 8)

```
118    // output results
119    for ( int rating = 1; rating <= 9; rating++ ) {
120       cout << setw( 8 ) << rating << setw( 11 )
121            << freq[ rating ] << "           ";
122
123       // keep track of mode value and largest fequency value
124       if ( freq[ rating ] > largest ) {
125          largest = freq[ rating ];
126          modeValue = rating;
127
128       } // end if
129
130       // output histogram bar representing frequency value
131       for ( int k = 1; k <= freq[ rating ]; k++ )
132          cout << '*';
133
134       cout << '\n';  // begin new line of output
135
136    } // end outer for
137
138    // display the mode value
139    cout << "The mode is the most frequent value.\n"
140         << "For this run the mode is " << modeValue
141         << " which occurred " << largest << " times." << endl;
142
143 } // end function mode
```

fig04_17.cpp (6 of 8)

mode là giá trị xuất hiện nhiều nhất (có giá trị cao nhất trong mảng **freq**).



```
144
145 // function that sorts an array with bubble sort algorithm
146 void bubbleSort( int a[], int size )
147 {
148    int hold;  // temporary location used to swap elements
149
150    // loop to control number of passes
151    for ( int pass = 1; pass < size; pass++ )
152
153       // loop to control number of comparisons per pass
154       for ( int j = 0; j < size - 1; j++ )
155
156          // swap elements if out of order
157          if ( a[ j ] > a[ j + 1 ] ) {
158             hold = a[ j ];
159             a[ j ] = a[ j + 1 ];
160             a[ j + 1 ] = hold;
161
162          } // end if
163
164 } // end function bubbleSort
165
```

fig04_17.cpp
(7 of 8)


fig04_17.cpp
(8 of 8)

```
166 // output array contents (20 values per row)
167 void printArray( const int a[], int size )
168 {
169    for ( int i = 0; i < size; i++ ) {
170
171       if ( i % 20 == 0 )  // begin new line every 20 values
172          cout << endl;
173
174       cout << setw( 2 ) << a[ i ];
175
176    } // end for
177
178 } // end function printArray
```

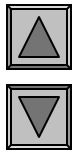


fig04_17.cpp
output (1 of 2)

```
********
  Mean
********
The mean is the average value of the data
items. The mean is equal to the total of
all the data items divided by the number
of data items (99). The mean value for
this run is: 681 / 99 = 6.8788
********
 Median
********
The unsorted array of responses is
 6 7 8 9 8 7 8 9 8 9 7 8 9 5 9 8 7 8 7 8
 6 7 8 9 3 9 8 7 8 7 7 8 9 8 9 8 9 7 8 9
 6 7 8 7 8 7 9 8 9 2 7 8 9 8 9 8 9 7 5 3
 5 6 7 2 5 3 9 4 6 4 7 8 9 6 8 7 8 9 7 8
 7 4 4 2 5 3 8 7 5 6 4 5 6 1 6 5 7 8 7

The sorted array is
 1 2 2 2 3 3 3 3 4 4 4 4 4 5 5 5 5 5 5 5
 5 6 6 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7 7
 7 7 7 7 7 7 7 7 7 7 7 7 7 8 8 8 8 8 8 8
 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9

The median is element 49 of
the sorted 99 element array.
For this run the median is 7
```

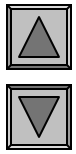


fig04_17.cpp
output (2 of 2)

```
********
  Mode
********
Response   Frequency         Histogram

                                    1    1    2    2
                               5    0    5    0    5

       1           1           *
       2           3           ***
       3           4           ****
       4           5           *****
       5           8           ********
       6           9           *********
       7          23           ***********************
       8          27           ***************************
       9          19           *******************
The mode is the most frequent value.
For this run the mode is 8 which occurred 27 times.
```

4.8 Tìm kiếm trên mảng: Tìm kiếm Tuyến tính và tìm kiếm Nhị phân

- Tìm một giá trị khoá (key value) trên mảng
- Tìm kiếm tuyến tính
  - So sánh từng phần tử của mảng với key
    - Bắt đầu từ một đầu, đi đến đầu kia của mảng
  - Hữu dụng cho mảng nhỏ và chưa sắp xếp
    - Không hiệu quả
    - Nếu giá trị cần tìm không có trong mảng thì phải kiểm tra tất cả các phần tử

4.8 Tìm kiếm trên mảng: Tìm kiếm Tuyến tính và tìm kiếm Nhị phân

- Tìm kiếm nhị phân
  - Chỉ sử dụng cho mảng đã sắp xếp
  - So sánh phần tử ở giữa (middle) với key
    - Nếu bằng, tìm thấy
    - Nếu key < middle
      - Lặp lại ở nửa đầu của mảng
    - Nếu key > middle
      - Lặp lại ở nửa cuối
  - Rất nhanh
    - Nhiều nhất là N bước với 2^N > số phần tử của mảng
    - mảng 30 phần tử cần nhiều nhất 5 bước
      $2^5 > 30$

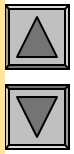


fig04_19.cpp (1 of 2)

```
1  // Fig. 4.19: fig04_19.cpp
2  // Linear search of an array.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  int linearSearch( const int [], int, int );  // prototype
10
11 int main()
12 {
13    const int arraySize = 100;  // size of array a
14    int a[ arraySize ];         // create array a
15    int searchKey;              // value to locate in a
16
17    for ( int i = 0; i < arraySize; i++ )  // create some data
18       a[ i ] = 2 * i;
19
20    cout << "Enter integer search key: ";
21    cin >> searchKey;
22
23    // attempt to locate searchKey in array a
24    int element = linearSearch( a, searchKey, arraySize );
25
```

Lấy đối số là một mảng, khoá cần tìm, và kích thước mảng.

```
26    // display results
27    if ( element != -1 )
28       cout << "Found value in element " << element << endl;
29    else
30       cout << "Value not found" << endl;
31
32    return 0;  // indicates successful termination
33
34 } // end main
35
36 // compare key to every element of array until location is
37 // found or until end of array is reached; return subscript of
38 // element if key or -1 if key not found
39 int linearSearch( const int array[], int key, int sizeOfArray )
40 {
41    for ( int j = 0; j < sizeOfArray; j++ )
42
43       if ( array[ j ] == key )  // if found,
44          return j;              // return location of key
45
46    return -1;  // key not found
47
48 } // end function linearSearch
```

fig04_19.cpp

```
Enter integer search key: 36
Found value in element 18

Enter integer search key: 37
Value not found
```

fig04_20.cpp
(1 of 6)

```
1  // Fig. 4.20: fig04_20.cpp
2  // Binary search of an array.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  #include <iomanip>
10
11 using std::setw;
12
13 // function prototypes
14 int binarySearch( const int [], int, int, int, int );
15 void printHeader( int );
16 void printRow( const int [], int, int, int, int );
17
18 int main()
19 {
20    const int arraySize = 15;  // size of array a
21    int a[ arraySize ];        // create array a
22    int key;                   // value to locate in a
23
24    for ( int i = 0; i < arraySize; i++ )  // create some data
25       a[ i ] = 2 * i;
26
```

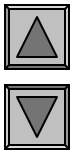


fig04_20.cpp
(2 of 6)

```
27     cout << "Enter a number between 0 and 28: ";
28     cin >> key;
29
30     printHeader( arraySize );
31
32     // search for key in array a
33     int result =
34        binarySearch( a, key, 0, arraySize - 1, arraySize );
35
36     // display results
37     if ( result != -1 )
38        cout << '\n' << key << " found in array element "
39             << result << endl;
40     else
41        cout << '\n' << key << " not found" << endl;
42
43     return 0;  // indicates successful termination
44
45  } // end main
46
```

```
47  // function to perform binary search of an array
48  int binarySearch( const int b[], int searchKey, int low,
49     int high, int size )
50  {
51     int middle;
52
53     // loop until low subscript is greater than high subscript
54     while ( low <= high ) {
55
56        // determine middle element of subarray being searched
57        middle = ( low + high ) / 2;
58
59        // display subarray used in this loop iteration
60        printRow( b, low, middle, high, size );
61
```

Xác định phần tử ở giữa

fig04_20.cpp
(3 of 6)

```
62          // if searchKey matches middle element, return middle
63          if ( searchKey == b[ middle ] )  // match
64             return middle;
65
66          else
67
68             // if searchKey less than middle element,
69             // set new high element
70             if ( searchKey < b[ middle ] )
71                high = middle - 1;  // search low end of array
72
73             // if searchKey greater than middle element,
74             // set new low element
75             else
76                low = middle + 1;   // search high end of array
77       }
78
79       return -1;  // searchKey not found
80
81    } // end function binarySearch
```

Sử dụng tìm Nhị phân:
Nếu key bằng middle, tìm thấy

Nếu nhỏ hơn, tìm nửa thấp

Nếu lớn hơn, tìm nửa cao

Vòng lặp tạo low, middle và high tự động. Nếu tìm nửa cao, thì phần tử low mới sẽ cao hơn middle.

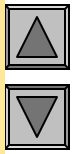


fig04_20.cpp
(5 of 6)

```
82
83 // print header for output
84 void printHeader( int size )
85 {
86    cout << "\nSubscripts:\n";
87
88    // output column heads
89    for ( int j = 0; j < size; j++ )
90       cout << setw( 3 ) << j << ' ';
91
92    cout << '\n';  // start new line of output
93
94    // output line of - characters
95    for ( int k = 1; k <= 4 * size; k++ )
96       cout << '-';
97
98    cout << endl;  // start new line of output
99
100 } // end function printHeader
101
```

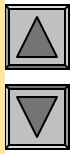


fig04_20.cpp
(6 of 6)

```
102 // print one row of output showing the current
103 // part of the array being processed
104 void printRow( const int b[], int low, int mid,
105    int high, int size )
106 {
107    // loop through entire array
108    for ( int m = 0; m < size; m++ )
109
110       // display spaces if outside current subarray range
111       if ( m < low || m > high )
112          cout << "    ";
113
114       // display middle element marked with a *
115       else
116
117          if ( m == mid )           // mark middle value
118             cout << setw( 3 ) << b[ m ] << '*';
119
120          // display other elements in subarray
121          else
122             cout << setw( 3 ) << b[ m ] << ' ';
123
124    cout << endl;  // start new line of output
125
126 } // end function printRow
```

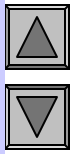


fig04_20.cpp
output (1 of 2)

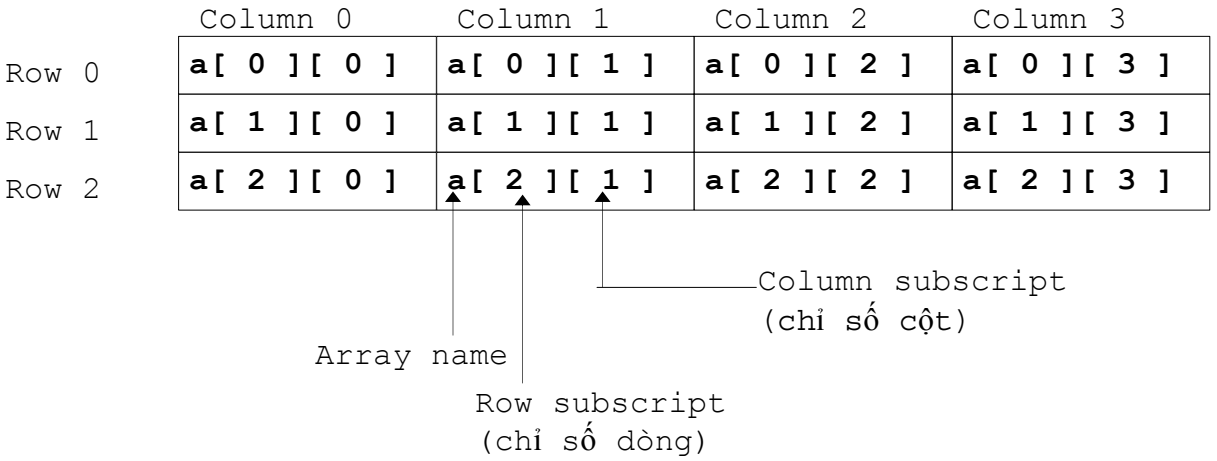
```
Enter a number between 0 and 28: 6

Subscripts:
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14
-------------------------------------------------------------
  0   2   4   6   8  10  12  14* 16  18  20  22  24  26  28
  0   2   4   6*  8  10  12

6 found in array element 3



Enter a number between 0 and 28: 25

Subscripts:
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14
-------------------------------------------------------------
  0   2   4   6   8  10  12  14* 16  18  20  22  24  26  28
                              16  18  20  22* 24  26  28
                                          24  26* 28
                                          24*

25 not found
```

```
Enter a number between 0 and 28: 8

Subscripts:
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14
---------------------------------------------------------------
  0   2   4   6   8  10  12  14* 16  18  20  22  24  26  28
  0   2   4   6*  8  10  12
                 8  10* 12
                 8*

8 found in array element 4
```

fig04_20.cpp output (2 of 2)

4.9 Mảng nhiều chiều

- Đa chỉ số
  - int a[3][4];
  - **a[i][j]**
  - Các bảng có dòng và cột
  - Dòng trước, cột sau
  - "Mảng của mảng"
    - **a[0]** là một mảng 4 phần tử
    - **a[0][0]** là phần tử đầu tiên của mảng



4.9 Mảng nhiều chiều

- Khởi tạo
  - Mặc định là **0**
  - Khởi tạo, mỗi dòng trong 1 cặp ngoặc

```
int b[ 2 ][ 2 ] = { { 1, 2 }, { 3, 4 } };
                    Row 0      Row 1
```

1	2
3	4

```
int b[ 2 ][ 2 ] = { { 1 }, { 3, 4 } };
```

1	0
3	4

4.9 Mảng nhiều chiều

- Truy nhập đến như bình thường

```
cout << b[ 0 ][ 1 ];
```

1	0
3	4

  - In ra **0**
  - Không sử dụng dấu phẩy (,)

```
cout << b[ 0, 1 ];
```

    - Lỗi cú pháp

- Function prototype
  - Phải chỉ rõ kích thước của các chỉ số
    - Không đòi hỏi kích thước cho chỉ số đầu tiên, cũng như mảng 1 chiều
  - `void printArray( int [][ 3 ] );`

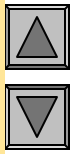


fig04_22.cpp
(1 of 2)

```
1  // Fig. 4.22: fig04_22.cpp
2  // Initializing multidimensional arrays.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  void printArray( int [][ 3 ] );
9
10 int main()
11 {
12    int array1[ 2 ][ 3 ] = { { 1, 2, 3 }, { 4, 5, 6 } };
13    int array2[ 2 ][ 3 ] = { 1, 2, 3, 4, 5};
14    int array3[ 2 ][ 3 ] = { { 1, 2 }, { 4 } };
15
16    cout << "Values in array1 by row are:" << endl;
17    printArray( array1 );
18
19    cout << "Values in array2 by row are:" << endl;
20    printArray( array2 );
21
22    cout << "Values in array3 by row are:" << endl;
23    printArray( array3 );
24
25    return 0;  // indicates successful termination
26 } // end main
```

Chú ý cấu trúc của prototype.

Chú ý nhiều cách khởi tạo. Các phần tử trong **array2** được gán từ dòng thứ nhất rồi đến dòng thứ hai.

```
28
29 // function to output array with two rows and three columns
30 void printArray( int a[][ 3 ] )
31 {
32    for ( int i = 0; i < 2; i++ ) {    // for each row
33
34       for ( int j = 0; j < 3; j++ )   // output column values
35          cout << a[ i ][ j ] << ' ';
36
37       cout << endl;  // start new line of output
38
39    } // end outer for structure
40
41 } // end function printArray
```

Vòng lặp for thường được dùng để quét qua mảng. Sử dụng vòng lặp lồng nhau cho mảng nhiều chiều.

ng04_22.cpp
output (1 of 1)

```
Values in array1 by row are:
1 2 3
4 5 6
Values in array2 by row are:
1 2 3
4 5 0
Values in array3 by row are:
1 2 0
4 0 0
```

4.9 Mảng nhiều chiều

- Tiếp theo: chương trình ví dụ về khởi tạo mảng
  - Chương trình lưu trữ điểm của sinh viên
  - Mảng nhiều chiều (bảng)
  - Dòng là sinh viên
  - Cột là điểm

	Quiz1	Quiz2
Student0	95	85
Student1	89	80

fig04_23.cpp
(1 of 6)

```
1  // Fig. 4.23: fig04_23.cpp
2  // Double-subscripted array example.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7  using std::fixed;
8  using std::left;
9
10 #include <iomanip>
11
12 using std::setw;
13 using std::setprecision;
14
15 const int students = 3;   // number of students
16 const int exams = 4;      // number of exams
17
18 // function prototypes
19 int minimum( int [][ exams ], int, int );
20 int maximum( int [][ exams ], int, int );
21 double average( int [], int );
22 void printArray( int [][ exams ], int, int );
23
```

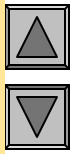


fig04_23.cpp
(2 of 6)

```
24 int main()
25 {
26    // initialize student grades for three students (rows)
27    int studentGrades[ students ][ exams ] =
28       { { 77, 68, 86, 73 },
29         { 96, 87, 89, 78 },
30         { 70, 90, 86, 81 } };
31
32    // output array studentGrades
33    cout << "The array is:\n";
34    printArray( studentGrades, students, exams );
35
36    // determine smallest and largest grade values
37    cout << "\n\nLowest grade: "
38         << minimum( studentGrades, students, exams )
39         << "\nHighest grade: "
40         << maximum( studentGrades, students, exams ) << '\n';
41
42    cout << fixed << setprecision( 2 );
43
```

```
44    // calculate average grade for each student
45    for ( int person = 0; person < students; person++ )
46       cout << "The average grade for student " << person
47            << " is "
48            << average( studentGrades[ person ], exams )
49            << endl;
50
51    return 0;  // indicates successful termination
52
53 } // end main
54
55 // find minimum grade
56 int minimum( int grades[][ exams ], int pupils, int tests )
57 {
58    int lowGrade = 100; // initialize to highest possible grade
59
60    for ( int i = 0; i < pupils; i++ )
61
62       for ( int j = 0; j < tests; j++ )
63
64          if ( grades[ i ][ j ] < lowGrade )
65             lowGrade = grades[ i ][ j ];
66
67    return lowGrade;
68
69 } // end function minimum
```

fig04_23.cpp (3 of 6)

Tính điểm trung bình cho sinh viên. Ta truyền dòng chứa điểm của sinh viên vào hàm. Chú ý: **`studentGrades[0]`** cũng là một mảng.

```
70
71 // find maximum grade
72 int maximum( int grades[][ exams ], int pupils, int tests )
73 {
74    int highGrade = 0;  // initialize to lowest possible grade
75
76    for ( int i = 0; i < pupils; i++ )
77
78       for ( int j = 0; j < tests; j++ )
79
80          if ( grades[ i ][ j ] > highGrade )
81             highGrade = grades[ i ][ j ];
82
83    return highGrade;
84
85 } // end function maximum
86
```

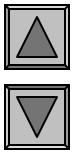
fig04_23.cpp
(4 of 6)

```
87  // determine average grade for particular student
88  double average( int setOfGrades[], int tests )
89  {
90     int total = 0;
91
92     // total all grades for one student
93     for ( int i = 0; i < tests; i++ )
94        total += setOfGrades[ i ];
95
96     return static_cast< double >( total ) / tests;  // average
97
98  } // end function maximum
```

fig04_23.cpp
(5 of 6)

```
99
100 // Print the array
101 void printArray( int grades[][ exams ], int pupils, int tests )
102 {
103    // set left justification and output column heads
104    cout << left << "                [0]  [1]  [2]  [3]";
105
106    // output grades in tabular format
107    for ( int i = 0; i < pupils; i++ ) {
108
109       // output label for row
110       cout << "\nstudentGrades[" << i << "] ";
111
112       // output one grades for one student
113       for ( int j = 0; j < tests; j++ )
114          cout << setw( 5 ) << grades[ i ][ j ];
115
116    } // end outer for
117
118 } // end function printArray
```

fig04_23.cpp
(6 of 6)



```
The array is:
                 [0]  [1]  [2]  [3]
studentGrades[0] 77   68   86   73
studentGrades[1] 96   87   89   78
studentGrades[2] 70   90   86   81

Lowest grade: 68
Highest grade: 96
The average grade for student 0 is 76.00
The average grade for student 1 is 87.50
The average grade for student 2 is 81.75
```

fig04_23.cpp
output (1 of 1)

Ngôn ngữ lập trình C++

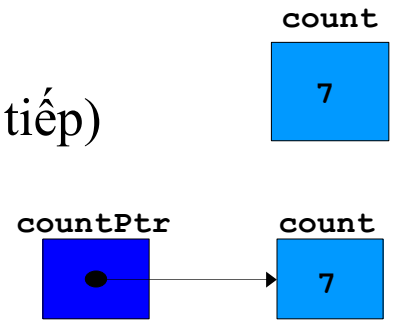
Chương 5 – Con trỏ và Xâu ký tự

Chương 5 – Con trỏ và Xâu ký tự

Đề mục

5.1 Giới thiệu

- Con trỏ (Pointer)
  - Mạnh, nhưng khó làm chủ
  - Có tác dụng như truyền tham chiếu (pass-by-reference)
  - Có liên quan chặt chẽ đến mảng và xâu
- Biến con trỏ (Pointer variable)
  - Chứa địa chỉ vùng nhớ thay vì chứa giá trị
  - Thông thường, biến chứa giá trị (tham chiếu trực tiếp)
  - Con trỏ chứa địa chỉ của biến mang giá trị cụ thể (tham chiếu gián tiếp)



5.2 Khai báo và khởi tạo biến con trỏ

- Khai báo con trỏ
  - ***** cho biết biến là con trỏ

```
int *myPtr;
```

    dữ liệu kiểu **int** có địa chỉ là **myPtr**, con trỏ kiểu **int ***
  - Mỗi con trỏ cần một dấu sao

```
int *myPtr1, *myPtr2;
```

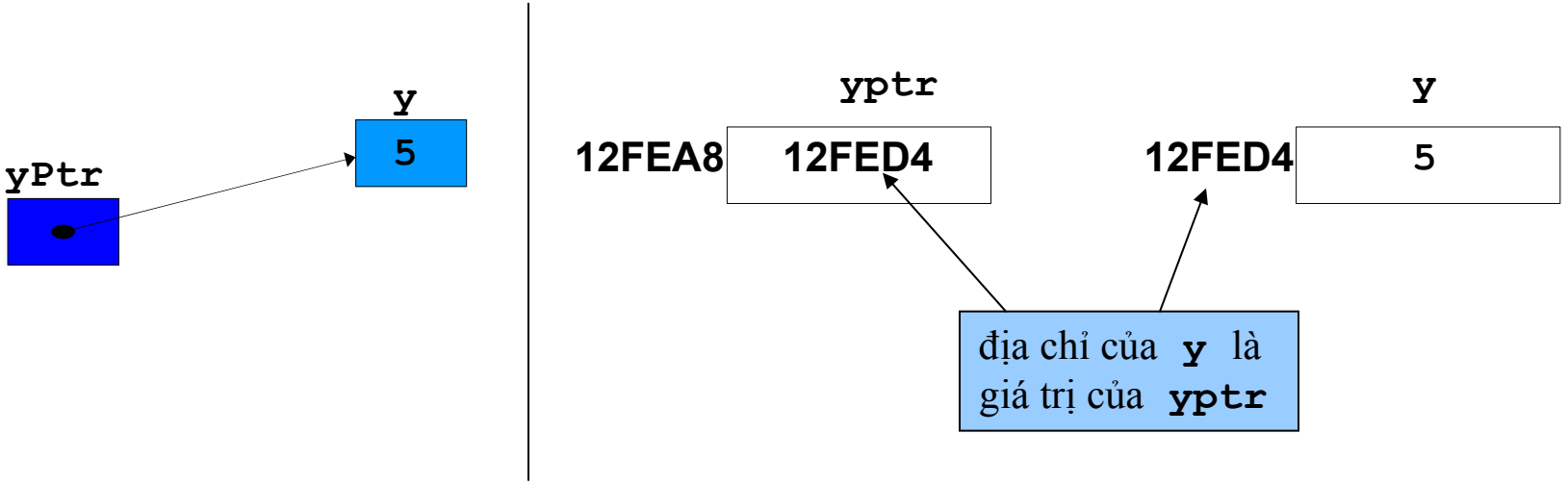
  - Có thể khai báo con trỏ tới bất cứ kiểu dữ liệu nào
- Khởi tạo con trỏ (Pointer initialization)
  - Khởi tạo về **0**, **NULL**, hoặc địa chỉ
    - **0** hoặc **NULL** không trỏ đến đâu cả

5.3 Các thao tác đối với con trỏ

- **&** Toán tử địa chỉ (address operator)
  - Trả về địa chỉ vùng nhớ của toán hạng
  - Ví dụ

```
int y = 5;
int *yPtr;
yPtr = &y;    // yPtr chứa địa chỉ của y
```

  - **yPtr** "trỏ đến" **y**



5.3 Các thao tác đối với con trỏ

- **`*`** phép thâm nhập (indirection/dereferencing)
  - Trả về đối tượng mà con trỏ trỏ tới
  - **`*yPtr`** trả về **`y`** (vì **`yPtr`** trỏ đến **`y`**).
  - con trỏ khi bị thâm nhập (dereferenced) là giá trị trái (lvalue)

```
*yptr = 9;       // assigns 9 to y
```

- **`*`** và **`&`** ngược nhau

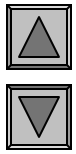


fig05_04.cpp
(1 of 2)

```
1   // Fig. 5.4: fig05_04.cpp
2   // Using the & and * operators.
3   #include <iostream>
4
5   using std::cout;
6   using std::endl;
7
8   int main()
9   {
10     int a;      // a is an integer
11     int *aPtr;  // aPtr is a pointer to an integer
12
13     a = 7;
14     aPtr = &a;  // aPtr assigned address of a
15
16     cout << "The address of a is " << &a
17          << "\nThe value of aPtr is " << aPtr;
18
19     cout << "\n\nThe value of a is " << a
20          << "\nThe value of *aPtr is " << *aPtr;
21
22     cout << "\n\nShowing that * and & are inverses of "
23          << "each other.\n&*aPtr = " << &*aPtr
24          << "\n*&aPtr = " << *&aPtr << endl;
25
```

***** và **&** ngược nhau

```
26     return 0;  // indicates successful termination
27
28  } // end main
```

fig05_04.cpp (2 of 2)

fig05_04.cpp output (1 of 1)

```
The address of a is 0012FED4
The value of aPtr is 0012FED4

The value of a is 7
The value of *aPtr is 7

Showing that * and & are inverses of each other.
&*aPtr = 0012FED4
*&aPtr = 0012FED4
```

***** và **&** ngược nhau; cùng kết quả khi cùng sử dụng cả 2 với **aPtr**

5.4 Gọi hàm bằng tham chiếu

- 3 cách truyền tham số cho hàm
  - Truyền giá trị (Pass-by-value)
  - Truyền tham chiếu với đối số là tham chiếu (Pass-by-reference with reference arguments)
  - Truyền tham chiếu với đối số là con trỏ (Pass-by-reference with pointer arguments)

5.4 Gọi hàm bằng tham chiếu

- Truyền tham chiếu với đối số là tham chiếu
  - Thay đổi giá trị gốc của tham số
  - hàm có thể "trả về" nhiều hơn một giá trị
- Truyền tham chiếu bằng đối số là con trỏ
  - Tương tự pass-by-reference
    - Sử dụng con trỏ và toán tử `*`
  - Truyền địa chỉ của đối số bằng toán tử `&`
  - Truyền mảng không cần toán tử `&` vì tên mảng chính là con trỏ
  - Toán tử thâm nhập `*` được dùng cùng con trỏ để tạo một tên khác cho biến được truyền vào

g05_06.cpp
of 2)

```
1  // Fig. 5.6: fig05_06.cpp
2  // Cube a variable using pass-by-value.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int cubeByValue( int );   // prototype
9
10 int main()
11 {
12    int number = 5;
13
14    cout << "The original value of number is "
15
16    // pass number by value to cubeByValue
17    number = cubeByValue( number );
18
19    cout << "\nThe new value of number is " << number << endl;
20
21    return 0;  // indicates successful termination
22
23 } // end main
24
```

Truyền number bằng giá trị; kết quả được trả về bởi **cubeByValue**

```
25  // calculate and return cube of integer argument
26  int cubeByValue( int n )
27  {
28     return n * n * n; // cube local variable n
29
30  } // end function cubeByValue
```

cubeByValue nhận tham số passed-by-value

Tính lập phương và trả về biến địa phương (local variable) **n**

ig05_06.cpp
2 of 2)

ig05_06.cpp
utput (1 of 1)

```
The original value of number is 5
The new value of number is 125
```

ig05_07.cpp
1 of 2)

```
1  // Fig. 5.7: fig05_07.cpp
2  // Cube a variable using pass-by-reference
3  // with a pointer argument.
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8
9  void cubeByReference( int * );   // prototype
10
11 int main()
12 {
13    int number = 5;
14
15    cout << "The original value of number is " << number;
16
17    // pass address of number to cubeByReference
18    cubeByReference( &number );
19
20    cout << "\nThe new value of number is " << number << endl;
21
22    return 0;  // indicates successful termination
23
24 } // end main
25
```

Prototype cho biết tham số là con trỏ trỏ đến dữ liệu kiểu **int**

Dùng toán tử địa chỉ **&** để truyền địa chỉ của **number** tới **cubeByReference**

cubeByReference thay đổi biến **number**

```
26  // calculate cube of *nPtr; modifies variable number in main
27  void cubeByReference( int *nPtr )
28  {
29     *nPtr = *nPtr * *nPtr * *nPtr;  // cu
30
31  } // end function cubeByReference
```

cubeByReference nhận địa chỉ của biến kiểu **int**, tức là con trỏ trỏ đến một số **int**

7.cpp

7.cpp

output (1 of 1)

```
The original value of number is 5
The new value of number is 125
```

Thay đổi và truy nhập biến kiểu **int** sử dụng toán tử thâm nhập ***

5.5 Sử dụng `const` với con trỏ

- Tính chất của **`const`**
  - Giá trị của biến không thay đổi
  - **`const`** được sử dụng cho một biến khi hàm không cần thay đổi biến đó.
- Nguyên tắc quyền ưu tiên tối thiểu
  - Chỉ cho hàm đủ quyền truy nhập để thực hiện nhiệm vụ của mình, không cho nhiều quyền hơn.
- Bốn cách truyền con trỏ cho hàm
  - Con trỏ thường trỏ đến dữ liệu thường
    - Khả năng truy cập cao nhất
  - Con trỏ thường trỏ đến hằng dữ liệu
  - Hằng con trỏ trỏ đến dữ liệu thường
  - Hằng con trỏ trỏ đến hằng dữ liệu
    - Ít quyền truy cập nhất

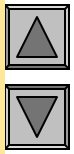


fig05_10.cpp
(1 of 2)

```
1  // Fig. 5.10: fig05_10.cpp
2  // Converting lowercase letters to uppercase letters
3  // using a non-constant pointer to non-constant data.
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8
9  #include <cctype>    // prototypes for islower and toupper
10
11 void convertToUppercase( char * );
12
13 int main()
14 {
15    char phrase[] = "characters and $32.98";
16
17    cout << "The phrase before conversion is: " << phrase;
18    convertToUppercase( phrase );
19    cout << "\nThe phrase after conversion is:  "
20         << phrase << endl;
21
22    return 0;  // indicates successful termination
23
24 } // end main
25
```

Con trỏ thường
đến dữ liệu thường

convertToUppercase
thay đổi biến **phrase**

```
26 // convert string to uppercase letters
27 void convertToUppercase( char *sPtr )
28 {
29    while ( *sPtr != '\0' ) {   // current charact
30
31       if ( islower( *sPtr ) )  // if character is lowercase,
32          *sPtr = toupper( *sPtr );
33
34       ++sPtr;  // move sPtr to next character in string
35
36    } // end while
37
38 } // end function convertToUppercase
```

sPtr là con trỏ thường trỏ đến dữ liệu thường

Hàm **islower** trả về **true** nếu ký tự là chữ thường

Hàm **toupper** trả về chữ hoa nếu ký tự ban đầu là chữ thường; nếu không **toupper** trả về ký tự đó (chữ hoa)

Khi dùng toán tử **++** cho con trỏ trỏ đến mảng, địa chỉ vùng nhớ lưu trong con trỏ sẽ được sửa để con trỏ trỏ đến phần tử tiếp theo của mảng.

cpp
(2 of 2)

fig05_10.cpp
output (1 of 1)

```
The phrase before conversion is: characters and $32.98
The phrase after conversion is:  CHARACTERS AND $32.98
```

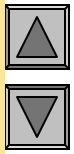


fig05_11.cpp
(1 of 2)

```
1   // Fig. 5.11: fig05_11.cpp
2   // Printing a string one character at a time using
3   // a non-constant pointer to constant data.
4   #include <iostream>
5
6   using std::cout;
7   using std::endl;
8
9   void printCharacters( const char * );
10
11  int main()
12  {
13     char phrase[] = "print characters of a string";
14
15     cout << "The string is:\n";
16     printCharacters( phrase );
17     cout << endl;
18
19     return 0;  // indicates successful termination
20
21  } // end main
22
```

Tham số là con trỏ thường trỏ đến hằng dữ liệu

Truyền con trỏ **phrase** cho hàm **printCharacters**.

```
23  // sPtr cannot modify the character to which it points,
24  // i.e., sPtr is a "read-only" pointer
25  void printCharacters( const char *sPtr )
26  {
27     for ( ; *sPtr != '\0'; sPtr++ )   // no initialization
28        cout << *sPtr;
29
30  } // end function printCharacters
```

fig05_11.cpp
(2 of 2)

sPtr là con trỏ thường trỏ đến hằng dữ liệu; không thể thay đổi ký tự mà **sPtr** trỏ đến.

Tăng **sPtr** để trỏ đến ký tự tiếp theo.

```
The string is:
print characters of a string
```

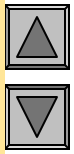


fig05_12.cpp (1 of 1)

fig05_12.cpp output (1 of 1)

```
1  // Fig. 5.12: fig05_12.cpp
2  // Attempting to modify data through a
3  // non-constant pointer to constant data.
4
5  void f( const int * );  // prototype
6
7  int main()
8  {
9     int y;
10
11    f( &y );   // f attempts illegal modification
12
13    return 0;  // indicates successful
14
15 } // end main
16
17 // xPtr cannot modify the value of the variable
18 // to which it points
19 void f( const int *xPtr )
20 {
21    *xPtr = 100;  // error: cannot modify a const object
22
23 } // end function f
```

Tham số là con trỏ thường trỏ đến hằng dữ liệu.

Truyền địa chỉ của biến **y** để thử thay đổi một cách không hợp lệ.

Cố thay đổi đối tượng hằng (const object) mà **xPtr** trỏ đến.

Lỗi sinh ra khi biên dịch.

```
d:\cpphtp4_examples\ch05\Fig05_12.cpp(21) : error C2166:
   l-value specifies const object
```

5.5 Sử dụng `const` với con trỏ

- **`const`** pointers - hằng con trỏ
  - Luôn trỏ đến vùng nhớ cố định
  - là mặc định cho tên mảng
  - Phải được khởi tạo khi khai báo

_13.cpp
1)

_13.cpp
ıt (1 of 1)

```
1  // Fig. 5.13: fig05_13.cpp
2  // Attempting to modify a constant pointer to
3  // non-constant data.
4
5  int main()
6  {
7     int x, y;
8
9     // ptr is a constant                    at can
10    // be modified throu                    ints to the
11    // same memory location.
12    int * const ptr = &x;
13
14    *ptr = 7;  // allowed: *ptr is not const
15    ptr = &y;  // error: ptr is const; cannot assign new address
16
17    return 0;  // indicates successful termination
18
19 } // end main
```

ptr là hằng con trỏ trỏ tới số nguyên.

Có thể thay đổi **x** (trỏ bởi **ptr**) vì **x** không phải là hằng

Không thể cho **ptr** trỏ đến địa chỉ mới vì **ptr** là hằng

Dòng 15 sinh ra lỗi biên dịch vì thay đổi địa chỉ mới cho constant pointer.

```
d:\cpphtp4_examples\ch05\Fig05_13.cpp(15) : error C2166:
   l-value specifies const object
```

fig05_14.cpp (1 of 1)

```
1  // Fig. 5.14: fig05_14.cpp
2  // Attempting to modify a constant pointer to constant data.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int main()
9  {
10    int x = 5, y;
11
12    // ptr is a constant pointer to a constant integer.
13    // ptr always points to the same location; the integer
14    // at that location cannot be modified.
15    const int *const ptr = &x;
16
17    cout << *ptr << endl;
18
19    *ptr = 7;  // error: *ptr is const; cannot assign new value
20    ptr = &y;  // error: ptr is const; cannot assign new address
21
22    return 0;  // indicates successful termination
23
24 } // end main
```

ptr là hằng con trỏ trỏ tới hằng số nguyên.

Không thể thay đổi **x** (trỏ bởi **ptr**) vì khai báo ***ptr** là hằng.

Không thể cho **ptr** trỏ đến địa chỉ mới vì **ptr** được khai báo là hằng.

5.6 Sắp xếp nổi bọt sử dụng truyền tham chiếu

- **`bubbleSort`** dùng con trỏ
  - Hàm **`swap`** truy nhập các phần tử của mảng
    - Các phần tử đơn của mảng: dữ liệu vô hướng (scalars)
      - Mặc định là pass by value
    - Truyền tham chiếu bằng toán tử địa chỉ **`&`**

```
1   // Fig. 5.15: fig05_15.cpp
2   // This program puts values into an array, sorts the values into
3   // ascending order, and prints the resulting array.
4   #include <iostream>
5
6   using std::cout;
7   using std::endl;
8
9   #include <iomanip>
10
11  using std::setw;
12
13  void bubbleSort( int *, const int );    // prototype
14  void swap( int * const, int * const );  // prototype
15
16  int main()
17  {
18     const int arraySize = 10;
19     int a[ arraySize ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
20
21     cout << "Data items in original order\n";
22
23     for ( int i = 0; i < arraySize; i++ )
24        cout << setw( 4 ) << a[ i ];
25
```

)5_15.cpp
of 3)

ần Minh Châu.
. VNU.

```
26    bubbleSort( a, arraySize );  // sort the array
27
28    cout << "\nData items in ascending order\n";
29
30    for ( int j = 0; j < arraySize; j++ )
31       cout << setw( 4 ) << a[ j ];
32
33    cout << endl;
34
35    return 0;  // indicates successful t
36
37 } // end main
38
39 // sort an array of integers using bubble sort algo
40 void bubbleSort( int *array, const int size )
41 {
42    // loop to control passes
43    for ( int pass = 0; pass < size - 1; pass++ )
44
45       // loop to control comparisons during each pass
46       for ( int k = 0; k < size - 1; k++ )
47
48          // swap adjacent elements if they are out of order
49          if ( array[ k ] > array[ k + 1 ] )
50             swap( &array[ k ], &array[ k + 1 ] );
```

fig05_15.cpp
(2 of 3)

Khai báo là **`int *array`** (thay vì **`int array[]`**) để cho hàm **`bubbleSort`** nhận mảng 1 chiều. Hai cách khai báo này là như nhau.

Nhận tham số kích thước của mảng; khai báo là **`const`** để chắc chắn rằng **`size`** sẽ không bị thay đổi.

```
51
52  } // end function bubbleSort
53
54  // swap values at memory locations to which
55  // element1Ptr and element2Ptr point
56  void swap( int * const element1Ptr, int * const element2Ptr )
57  {
58     int hold = *element1Ptr;
59     *element1Ptr = *element2Ptr;
60     *element2Ptr = hold;
61
62  } // end function swap
```

fig05_15.cpp (3 of 3)

fig05_15.cpp output (1 of 1)

Truyền tham chiếu, cho phép hàm tráo giá trị tại vùng nhớ.

```
Data items in original order
   2   6   4   8  10  12  89  68  45  37
Data items in ascending order
   2   4   6   8  10  12  37  45  68  89
```

5.6 Sắp xếp nổi bọt sử dụng truyền tham chiếu

- **`sizeof`**
  - Toán tử trả về kích thước byte của toán hạng
  - Với mảng, **`sizeof`** trả về giá trị

    (kích thước 1 phần tử) * (số phần tử)
  - Nếu **`sizeof( int ) = 4`**, thì

```
int myArray[10];
cout << sizeof(myArray);
```

    sẽ in ra 40
- **`sizeof`** có thể dùng với
  - Tên biến — `cout << "sizeof c = " << sizeof c`
  - Tên kiểu dữ liệu — `cout << sizeof( char )`
  - Hằng số

```
1   // Fig. 5.16: fig05_16.cpp
2   // Sizeof operator when used on an array name
3   // returns the number of bytes in the array.
4   #include <iostream>
5
6   using std::cout;
7   using std::endl;
8
9   size_t getSize( double * );  // prototype
10
11  int main()
12  {
13     double array[ 20 ];
14
15     cout << "The number of bytes in the array is "
16          << sizeof( array );
17
18     cout << "\nThe number of bytes returned by getSize is "
19          << getSize( array ) << endl;
20
21     return 0;  // indicates successful termination
22
23  } // end main
24
```

fig05_16.cpp
(1 of 2)

sizeof trả về tổng số byte của mảng.

Hàm **getSize** trả về số byte được dùng để lưu địa chỉ mảng **array**.

```
25  // return size of ptr
26  size_t getSize( double *ptr )
27  {
28     return sizeof( ptr );
29
30  } // end function getSize
```

sizeof trả về số byte của con trỏ.

fig05_16.cpp (2 of 2)

fig05_16.cpp output (1 of 1)

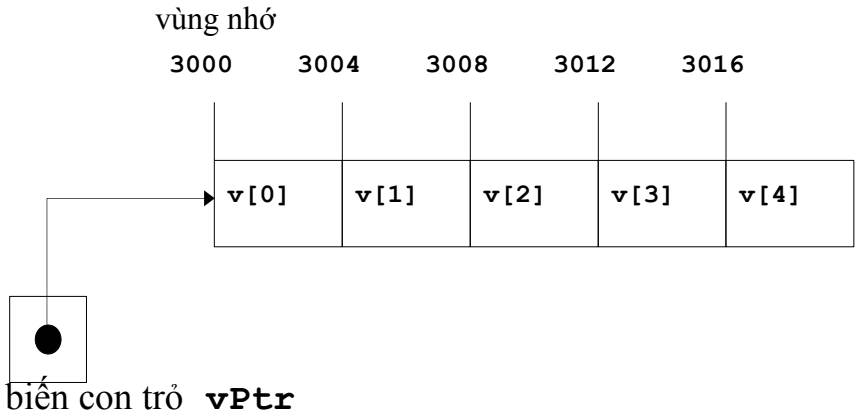
```
The number of bytes in the array is 160
The number of bytes returned by getSize is 4
```

5.7 Các phép toán đối với con trỏ

- Các phép toán con trỏ
  - Tăng/giảm con trỏ (**++** hoặc **--**)
  - Cộng/trừ 1 số nguyên với 1 con trỏ (**+** hoặc **+=** , **-** hoặc **-=**)
  - Con trỏ có thể trừ lẫn nhau
  - Cộng trừ với con trỏ là vô nghĩa trừ khi dùng cho con trỏ mảng
- Ví dụ: Mảng 5 phần tử **int** trên máy dùng kiểu **int** 4 byte
  - **vPtr** trỏ đến phần tử thứ nhất **v[0]**, tại địa chỉ 3000

    vPtr = 3000
  - **vPtr += 2**; trỏ **vPtr** tới **3008**

    vPtr trỏ tới **v[2]**



5.7 Các phép toán đối với con trỏ

- Trừ con trỏ (Subtracting pointers)
  - Trả về số phần tử giữa 2 địa chỉ

```
vPtr2 = v[ 2 ];
vPtr = v[ 0 ];
vPtr2 - vPtr == 2
```

- Gán con trỏ (Pointer assignment)
  - Một con trỏ có thể được gán cho con trỏ khác nếu cả hai cùng kiểu
  - Nếu không cùng kiểu thì phải đổi kiểu (cast)
  - Ngoại lệ: con trỏ tới **`void`** (kiểu **`void *`**)
    - con trỏ tổng quát, đại diện cho kiểu bất kỳ
    - không cần đổi kiểu để chuyển sang con trỏ sang dạng **`void`** pointer
    - Không thể (dùng *) lấy dữ liệu của con trỏ kiểu **`void`**

5.7 Các phép toán đối với con trỏ

- So sánh con trỏ (Pointer comparison)
  - Sử dụng các toán tử quan hệ để so sánh địa chỉ chứa trong con trỏ
  - Ví dụ: có hai con trỏ trỏ đến hai phần tử của một mảng, chỉ ra con trỏ trỏ đến phần tử được đánh số thứ tự cao
  - So sánh là vô nghĩa trừ khi các con trỏ trỏ đến các phần tử của cùng một mảng
  - Thường dùng để xác định khi con trỏ có giá trị bằng 0 (null) (không trỏ đến đâu cả)

5.8 Quan hệ giữa Con trỏ và Mảng

- Mảng và con trỏ có quan hệ chặt chẽ
  - Tên mảng cũng như hằng con trỏ (constant pointer)
  - Có thể dùng chỉ số đối với các con trỏ
- Dùng con trỏ để truy nhập các phần tử mảng
  - Phần tử `b[ n ]` có thể truy nhập bởi `*( bPtr + n )`
    - ký hiệu pointer/offset
  - Địa chỉ
    - `&b[ 3 ]` tương đương `bPtr + 3`
  - Tên mảng có thể coi như con trỏ
    - `b[ 3 ]` tương đương `*( b + 3 )`
  - Con trỏ có thể viết với cặp ngoặc vuông (ký hiệu pointer/subscript)
    - `bPtr[ 3 ]` tương đương `b[ 3 ]`

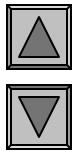


fig05_20.cpp
(1 of 2)

```
1  // Fig. 5.20: fig05_20.cpp
2  // Using subscripting and pointer notations with arrays.
3
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8
9  int main()
10 {
11    int b[] = { 10, 20, 30, 40 };
12    int *bPtr = b;   // set bPtr to point to array b
13
14    // output array b using array subscript notation
15    cout << "Array b printed with:\n"
16         << "Array subscript notation\n";
17
18    for ( int i = 0; i < 4; i++ )
19       cout << "b[" << i << "] = " << b[ i ] << '\n';
20
21    // output array b using the array name and
22    // pointer/offset notation
23    cout << "\nPointer/offset notation where "
24         << "the pointer is the array name\n";
25
```

Sử dụng ký hiệu chỉ số mảng.

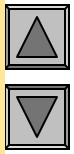


fig05_20.cpp
(2 of 2)

```
26     for ( int offset1 = 0; offset1 < 4; offset1++ )
27        cout << "*(b + " << offset1 << ") = "
28             << *( b + offset1 ) << '\n';
29
30     // output array b using bPtr and array subscript notation
31     cout << "\nPointer subscript notation\n";
32
33     for ( int j = 0; j < 4; j++ )
34        cout << "bPtr[" << j << "] = " << bPtr[ j ] << '\n';
35
36     cout << "\nPointer/offset notation\n";
37
38     // output array b using bPtr and pointer/offset notation
39     for ( int offset2 = 0; offset2 < 4; offset2++ )
40        cout << "*(bPtr + " << offset2 << ") = "
41             << *( bPtr + offset2 ) << '\n';
42
43     return 0;  // indicates successful termination
44
45  } // end main
```

Sử dụng tên mảng và ký hiệu pointer/offset.

Sử dụng ký hiệu chỉ số cho con trỏ.

Sử dụng **bPtr** và ký hiệu pointer/offset.

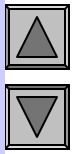


fig05_20.cpp
output (1 of 1)

```
Array b printed with:

Array subscript notation
b[0] = 10
b[1] = 20
b[2] = 30
b[3] = 40

Pointer/offset notation where the pointer is the array name
*(b + 0) = 10
*(b + 1) = 20
*(b + 2) = 30
*(b + 3) = 40

Pointer subscript notation
bPtr[0] = 10
bPtr[1] = 20
bPtr[2] = 30
bPtr[3] = 40

Pointer/offset notation
*(bPtr + 0) = 10
*(bPtr + 1) = 20
*(bPtr + 2) = 30
*(bPtr + 3) = 40
```

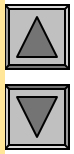


fig05_21.cpp
(1 of 2)

```
1   // Fig. 5.21: fig05_21.cpp
2   // Copying a string using array notation
3   // and pointer notation.
4   #include <iostream>
5
6   using std::cout;
7   using std::endl;
8
9   void copy1( char *, const char * );  // prototype
10  void copy2( char *, const char * );  // prototype
11
12  int main()
13  {
14     char string1[ 10 ];
15     char *string2 = "Hello";
16     char string3[ 10 ];
17     char string4[] = "Good Bye";
18
19     copy1( string1, string2 );
20     cout << "string1 = " << string1 << endl;
21
22     copy2( string3, string4 );
23     cout << "string3 = " << string3 << endl;
24
25     return 0;  // indicates successful termination
```

```
26
27  } // end main
28
29  // copy s2 to s1 using array notation
30  void copy1( char *s1, const char *s2 )
31  {
32     for ( int i = 0; ( s1[ i ] = s2[ i ] ) != '\0'; i++ )
33        ;   // do nothing in body
34
35  } // end function copy1
36
37  // copy s2 to s1 using pointer notation
38  void copy2( char *s1, const char *s2 )
39  {
40     for ( ; ( *s1 = *s2 ) != '\0'; s1++, s2++ )
41        ;   // do nothing in body
42
43  } // end function copy2
```

Sử dụng chỉ số mảng để copy xâu tại **s2** vào mảng ký tự **s1**.

Sử dụng ký hiệu con trỏ để copy xâu tại **s2** vào mảng ký tự **s1**.

Tăng cả hai con trỏ để trỏ đến phần tử tiếp theo trong mảng tương ứng.

5_21.cpp
(2 of 2)

fig05_21.cpp
output (1 of 1)

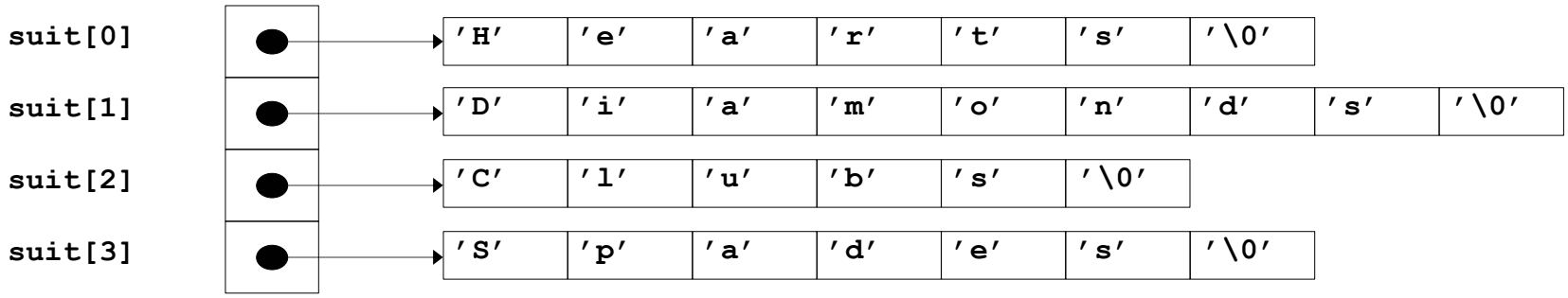
```
string1 = Hello
string3 = Good Bye
```

5.9 Mảng con trỏ

- Mảng chứa con trỏ
  - Thường dùng để lưu mảng của xâu

```
char *suit[ 4 ] = {"Hearts", "Diamonds",
                   "Clubs", "Spades" };
```

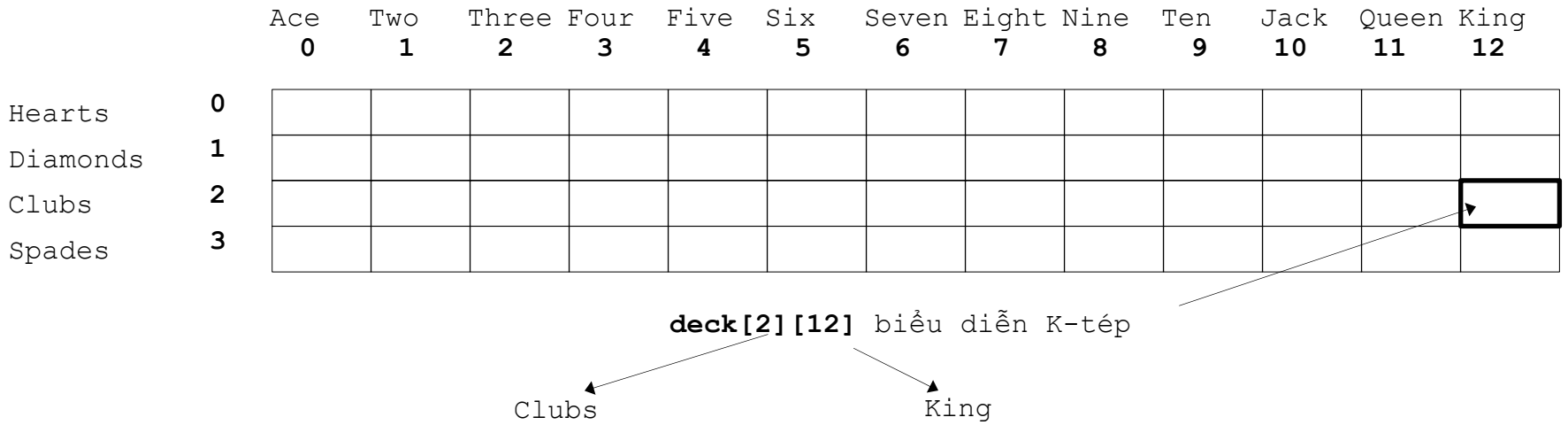
  - Mỗi phần tử của **suit** trỏ đến **char *** (1 xâu)
  - Mảng không chứa xâu, chỉ trỏ đến xâu



  - Mảng **suit** có kích thước cố định, nhưng xâu thì không

5.10 Ví dụ: Giả lập tráo bài và chia bài Tú-lơ-khơ

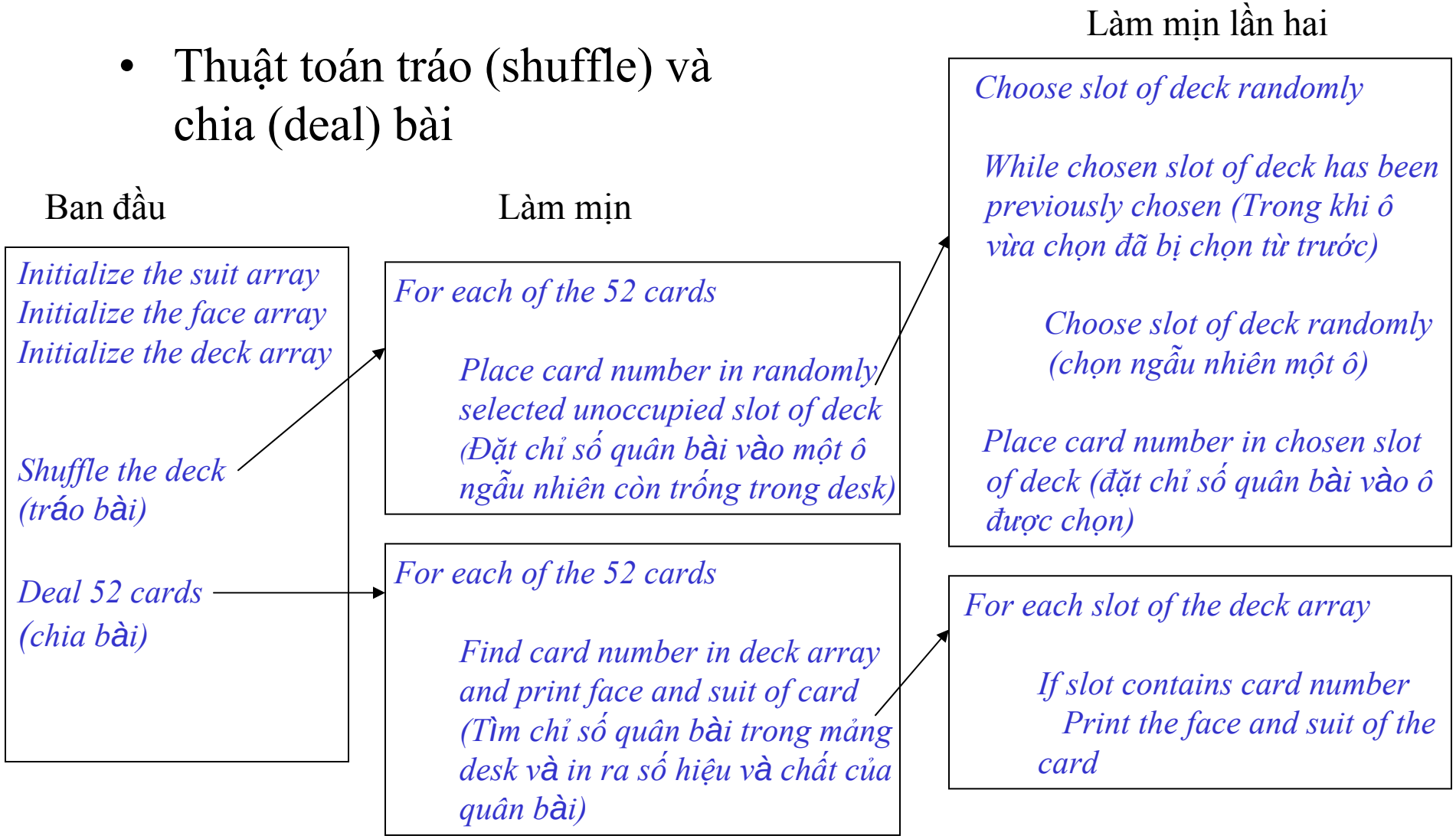
- Chương trình tráo bài (Card shuffling program)
  - Dùng một mảng gồm các con trỏ trỏ đến xâu để lưu trữ tên các chất (suit), i.e. cơ (hearts), rô (diamonds), pích (spades), tép (clubs)
  - Sử dụng một mảng hai chiều (hàng: chất, cột: giá trị)



  - Ghi các số từ 1-52 vào mảng để làm thứ tự chia các con bài

5.10 Ví dụ: Giả lập tráo bài và chia bài Tú-lơ-khơ

- Thuật toán tráo (shuffle) và chia (deal) bài



```
1   // Fig. 5.24: fig05_24.cpp
2   // Card shuffling dealing program.
3   #include <iostream>
4
5   using std::cout;
6   using std::left;
7   using std::right;
8
9   #include <iomanip>
10
11  using std::setw;
12
13  #include <cstdlib>  // prototypes for rand and srand
14  #include <ctime>    // prototype for time
15
16  // prototypes
17  void shuffle( int [][ 13 ] );
18  void deal( const int [][ 13 ], const char *[], const char *[] );
19
20  int main()
21  {
22     // initialize suit array
23     const char *suit[ 4 ] =
24        { "Hearts", "Diamonds", "Clubs", "Spades" };
25
```

mảng **suit** chứa các con trỏ trỏ đến các mảng **char**.

fig05_24.cpp
(1 of 4)

```
26    // initialize face array
27    const char *face[ 13 ] =
28       { "Ace", "Deuce", "Three", "Four",
29         "Five", "Six", "Seven", "Eight",
30         "Nine", "Ten", "Jack", "Queen", "King" };
31
32    // initialize deck array
33    int deck[ 4 ][ 13 ] = { 0 };
34
35    srand( time( 0 ) );         // seed random number generator
36
37    shuffle( deck );
38    deal( deck, face, suit );
39
40    return 0;  // indicates successful termination
41
42 } // end main
43
```

fig05_24.cpp (2 of 4)

mảng **face** chứa các con trỏ trỏ đến các mảng **char**.

```
44 // shuffle cards in deck
45 void shuffle( int wDeck[][ 13 ] )
46 {
47    int row;
48    int column;
49
50    // for each of the 52 cards, choose slot of deck randomly
51    for ( int card = 1; card <= 52; card++ ) {
52
53       // choose new random location until unoccupied slot found
54       do {
55          row = rand() % 4;
56          column = rand() % 13;
57       } while ( wDeck[ row ][ column ] != 0 ); // end do/while
58
59       // place card number in chosen slot of deck
60       wDeck[ row ][ column ] = card;
61
62    } // end for
63
64 } // end function shuffle
65
```

fig05_24.cpp
(3 of 4)

Vị trí hiện tại có dòng và cột được chọn ngẫu nhiên.

```
66 // deal cards in deck
67 void deal( const int wDeck[][ 13 ], const char *wFace[],
68            const char *wSuit[] )
69 {
70    // for each of the 52 cards
71    for ( int card = 1; card <= 52; card++ )
72
73       // loop through rows of wDeck
74       for ( int row = 0; row <= 3; row++ )
75
76          // loop through columns of wDeck for current ro
77          for ( int column = 0; column <= 12; column++ )
78
79             // if slot contains current card, display card
80             if ( wDeck[ row ][ column ] == card ) {
81                cout << setw( 5 ) << right << wFace[ column ]
82                     << " of " << setw( 8 ) << left
83                     << wSuit[ row ]
84                     << ( card % 2 == 0 ? '\n' : '\t' );
85
86             } // end if
87
88 } // end function deal
```

fig05_24.cpp
(4 of 4)

Căn lề phải trong một vùng gồm 5 ký tự.

Căn lề trái trong một vùng gồm 8 ký tự.

```
 Nine of Spades         Seven of Clubs
 Five of Spades         Eight of Clubs
Queen of Diamonds       Three of Hearts
 Jack of Spades          Five of Diamonds
 Jack of Diamonds       Three of Diamonds
Three of Clubs            Six of Clubs
  Ten of Clubs           Nine of Diamonds
  Ace of Hearts         Queen of Hearts
Seven of Spades         Deuce of Spades
  Six of Hearts         Deuce of Clubs
  Ace of Clubs          Deuce of Diamonds
 Nine of Hearts         Seven of Diamonds
  Six of Spades         Eight of Diamonds
  Ten of Spades          King of Hearts
 Four of Clubs            Ace of Spades
  Ten of Hearts          Four of Spades
Eight of Hearts         Eight of Spades
 Jack of Hearts           Ten of Diamonds
 Four of Diamonds        King of Diamonds
Seven of Hearts          King of Spades
Queen of Spades          Four of Hearts
 Nine of Clubs            Six of Diamonds
Deuce of Hearts          Jack of Clubs
 King of Clubs          Three of Spades
Queen of Clubs           Five of Clubs
 Five of Hearts           Ace of Diamonds
```

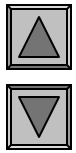


fig05_24.cpp
output (1 of 1)

5.11 Con trỏ tới hàm (Function Pointer)

- Con trỏ tới hàm
  - chứa địa chỉ của hàm
  - Tên mảng có giá trị là địa chỉ của phần tử đầu tiên của mảng
  - Tương tự, tên hàm có giá trị là địa chỉ bắt đầu của đoạn mã định nghĩa hàm
- Các con trỏ tới hàm có thể
  - được truyền vào trong hàm
  - được trả về từ hàm
  - được lưu trong mảng
  - được gán cho các con trỏ hàm khác

5.11 Con trỏ tới hàm

- Gọi hàm bằng con trỏ tới hàm
  - giả sử compare được khai báo là con trỏ tới hàm có kiểu tham số và kiểu trả về như sau:
    - `bool ( *compare ) ( int, int )`
  - gọi hàm bằng một trong hai cách
    - `( *compare ) ( int1, int2 )`
      - thâm nhập con trỏ để chạy hàm được con trỏ trỏ tới

    HOẶC
    - `compare( int1, int2 )`
      - dễ nhầm lẫn
        - người dùng có thể tưởng **`compare`** là tên của hàm thực trong chương trình

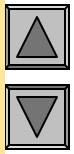


fig05_25.cpp
(1 of 5)

```
1  // Fig. 5.25: fig05_25.cpp
2  // Multipurpose sorting program using function pointers.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  #include <iomanip>
10
11 using std::setw;
12
13 // prototypes
14 void bubble( int [], const int, bool (*)( int, int ) );
15 void swap( int * const, int * const );
16 bool ascending( int, int );
17 bool descending( int, int );
18
19 int main()
20 {
21    const int arraySize = 10;
22    int order;
23    int counter;
24    int a[ arraySize ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37
};
25
```

Tham số thứ ba là con trỏ tới một hàm nhận 2 tham số **int** và trả về kết quả kiểu **bool**.

```
26     cout << "Enter 1 to sort in ascending order,\n"
27          << "Enter 2 to sort in descending order: ";
28     cin >> order;
29     cout << "\nData items in original order\n";
30
31     // output original array
32     for ( counter = 0; counter < arraySize; counter++ )
33        cout << setw( 4 ) << a[ counter ];
34
35     // sort array in ascending order; pass function ascending
36     // as an argument to specify ascending sorting order
37     if ( order == 1 ) {
38        bubble( a, arraySize, ascending );
39        cout << "\nData items in ascending order\n";
40     }
41
42     // sort array in descending order; pass function descending
43     // as an agrument to specify descending sorting order
44     else {
45        bubble( a, arraySize, descending );
46        cout << "\nData items in descending order\n";
47     }
48
```

fig05_25.cpp
(2 of 5)

```
49    // output sorted array
50    for ( counter = 0; counter < arraySize; counter++ )
51       cout << setw( 4 ) << a[ counter ];
52
53    cout << endl;
54
55    return 0;  // indicates successful termi
56
57 } // end main
58
59 // multipurpose bubble sort; parameter compare is a pointer to
60 // the comparison function that determines sorting order
61 void bubble( int work[], const int size,
62              bool (*compare)( int, int ) )
63 {
64    // loop to control passes
65    for ( int pass = 1; pass < size; pass++ )
66
67       // loop to control number of comparisons per pass
68       for ( int count = 0; count < size - 1; count++ )
69
70          // if adjacent elements are out of order, swap them
71          if ( (*compare)( work[ count ], work[ count + 1 ] ) )
72             swap( &work[ count ], &work[ count + 1 ] );
73
74 } // end function bubble
```

fig05_25.cpp (3 of 5)

compare là con trỏ tới một hàm nhận 2 tham số kiểu **int** và trả về giá trị kiểu **bool**.

Dùng ngoặc để chỉ rõ đây là con trỏ tới hàm

gọi hàm **compare** được truyền vào; thâm nhập con trỏ để chạy hàm.

```
75
76 // swap values at memory locations to which
77 // element1Ptr and element2Ptr point
78 void swap( int * const element1Ptr, int * const element2Ptr )
79 {
80    int hold = *element1Ptr;
81    *element1Ptr = *element2Ptr;
82    *element2Ptr = hold;
83
84 } // end function swap
85
86 // determine whether elements are out of order
87 // for an ascending order sort
88 bool ascending( int a, int b )
89 {
90    return b < a;// swap if b is less than a
91
92 } // end function ascending
93
94 // determine whether elements are out of order
95 // for a descending order sort
96 bool descending( int a, int b )
97 {
98    return b > a;   // swap if b is greater than a
99
100 } // end function descending
```

fig05_25.cpp

```
Enter 1 to sort in ascending order,
Enter 2 to sort in descending order: 1

Data items in original order
   2   6   4   8  10  12  89  68  45  37
Data items in ascending order
   2   4   6   8  10  12  37  45  68  89
```

```
Enter 1 to sort in ascending order,
Enter 2 to sort in descending order: 2

Data items in original order
   2   6   4   8  10  12  89  68  45  37
Data items in descending order
  89  68  45  37  12  10   8   6   4   2
```

5.11 Con trỏ tới hàm

- Mảng gồm các con trỏ hàm
  - Thường dùng cho các hệ thống điều khiển bằng thực đơn (menu-driven system)
  - Các con trỏ đến từng hàm được lưu trong mảng con trỏ hàm
    - các hàm đều phải có kiểu dữ liệu trả về giống nhau, và kiểu dữ liệu của tham số như nhau
  - Ánh xạ
    (lựa chọn thực đơn → chỉ số trong mảng con trỏ tới hàm)

```
1   // Fig. 5.26: fig05_26.cpp
2   // Demonstrating an array of pointers to functions.
3   #include <iostream>
4
5   using std::cout;
6   using std::cin;
7   using std::endl;
8
9   // function prototypes
10  void function1( int );
11  void function2( int );
12  void function3( int );
13
14  int main()
15  {
16     // initialize array of 3 pointers to functions that each
17     // take an int argument and return void
18     void (*f[ 3 ])( int ) = { function1, function2, function3 };
19
20     int choice;
21
22     cout << "Enter a number between 0 and 2, 3 to end: ";
23     cin >> choice;
24
```

fig05_26.cpp
(1 of 3)

Mảng được khởi tạo với tên của ba hàm, tên của hàm chính là con trỏ.

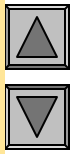
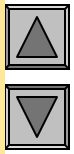


fig05_26.cpp
(2 of 3)

```
25    // process user's choice
26    while ( choice >= 0 && choice < 3 ) {
27
28       // invoke function at location choice in array f
29       // and pass choice as an argument
30       (*f[ choice ])( choice );
31
32       cout << "Enter a number between 0 and 2, 3 to end: ";
33       cin >> choice;
34    }
35
36    cout << "Program execution completed." << endl;
37
38    return 0;  // indicates successful termination
39
40 } // end main
41
42 void function1( int a )
43 {
44    cout << "You entered " << a
45         << " so function1 was called\n\n";
46
47 } // end function1
48
```

Gọi hàm được chọn bằng cách thâm nhập vào (dereferencing) phần tử tương ứng trong mảng.



```
49  void function2( int b )
50  {
51     cout << "You entered " << b
52          << " so function2 was called\n\n";
53
54  } // end function2
55
56  void function3( int c )
57  {
58     cout << "You entered " << c
59          << " so function3 was called\n\n";
60
61  } // end function3
```

fig05_26.cpp (3 of 3)

fig05_26.cpp output (1 of 1)

```
Enter a number between 0 and 2, 3 to end: 0
You entered 0 so function1 was called

Enter a number between 0 and 2, 3 to end: 1
You entered 1 so function2 was called

Enter a number between 0 and 2, 3 to end: 2
You entered 2 so function3 was called

Enter a number between 0 and 2, 3 to end: 3
Program execution completed.
```

5.12.1 Tổng kết về ký tự và xâu ký tự

- Hằng ký tự - Character constant
  - Giá trị nguyên biểu diễn dưới dạng một ký tự viết trong 2 dấu nháy
  - `'z'` là giá trị nguyên của ký tự `z`
    - Mã `122` trong bảng mã ASCII
- Xâu ký tự - String
  - Chuỗi các ký tự được coi như là một single unit
  - Có thể bao gồm chữ cái, chữ số, ký tự đặc biệt `+`, `-`, `*` ...
  - Hằng xâu ký tự - String literal (string constants)
    - Viết trong cặp nháy kép, ví dụ: `"I like C++"`
  - Mảng của các ký tự, kết thúc với ký tự rỗng (null character) `'\0'`
  - Xâu là một hằng con trỏ (constant pointer)
    - Trỏ đến ký tự đầu tiên của xâu
      - Giống như với mảng

5.12.1 Tổng kết về ký tự và xâu ký tự

- Gán giá trị cho xâu - String assignment
  - Mảng của ký tự
    - **`char color[] = "blue";`**
      - Tạo mảng **`color`** 5 phần tử kiểu **`char`**
        - phần tử cuối cùng là **`'\0'`**
  - Biến kiểu **`char *`**
    - **`char *colorPtr = "blue";`**
      - Tạo con trỏ **`colorPtr`** trỏ đến chữ **`b`** trong xâu **`"blue"`**
        - **`"blue"`** ở đâu đó trong bộ nhớ
  - Một cách khác cho mảng ký tự
    - **`char color[] = { 'b', 'l', 'u', 'e', '\0' };`**

5.12.1 Tổng kết về ký tự và xâu ký tự

- Đọc xâu
  - Đọc dữ liệu cho mảng ký tự **`word[ 20 ]`**

    `cin >> word`
    - Đọc các ký tự cho đến khi gặp ký tự trắng hoặc EOF
    - Xâu có thể vượt quá kích thước mảng

      `cin >> setw( 20 ) >> word;`
    - Đọc 19 ký tự (để lại chỗ cho **`'\0'`**)
- **`cin.getline`**
  - Đọc 1 dòng văn bản
  - **`cin.getline( array, size, delimiter );`**
  - Lưu input vào mảng **`array`** đến khi xảy ra một trong hai trường hợp
    - Kích thước dữ liệu đạt đến **`size`** – 1
    - Ký tự **`delimiter`** được nhập vào
  - Ví dụ

```
char sentence[ 80 ];
cin.getline( sentence, 80, '\n' );
```


5.12.2 Các hàm xử lý xâu ký tự

- Thư viện xử lý xâu **<cstring>** cung cấp các hàm
  - thao tác với dữ liệu kiểu xâu
  - so sánh xâu
  - tìm kiếm trên xâu các ký tự hoặc xâu khác
  - chia xâu thành các từ tố (tokenize strings)

5.12.2 Các hàm xử lý xâu ký tự

`char *strcpy( char *s1, const char *s2 );`	Copy xâu **s2** vào xâu **s1**. Trả về giá trị của **s1**.
`char *strncpy( char *s1, const char *s2, size_t n );`	Copy nhiều nhất **n** ký tự của xâu **s2** vào xâu **s1**. Trả về giá trị của **s1**.
`char *strcat( char *s1, const char *s2 );`	Thêm xâu **s2** vào sau xâu **s1**. Ký tự đầu tiên của **s2** ghi đè lên ký tự null của **s1**. Trả về giá trị của **s1**.
`char *strncat( char *s1, const char *s2, size_t n );`	Thêm xâu nhiều nhất là **n** ký tự của **s2** vào sau xâu **s1**. Ký tự đầu tiên của **s2** ghi đè lên ký tự null của **s1**. Trả về giá trị của **s1**.
`int strcmp( const char *s1, const char *s2 );`	So sánh xâu **s1** và xâu **s2**. Hàm trả về giá trị 0, nhỏ hơn 0, hoặc lớn hơn 0 nếu **s1** bằng, nhỏ hơn hoặc lớn hơn **s2**.

5.12.2 Các hàm xử lý xâu ký tự

`int strncmp( const char *s1, const char *s2, size_t n );`	So sánh **n** ký tự xâu **s1** và xâu **s2**. Hàm trả về giá trị 0, nhỏ hơn 0 hoặc lớn hơn 0 nếu **s1** bằng, nhỏ hơn hoặc lớn hơn **s2**.
`char *strtok( char *s1, const char *s2 );`	Một chuỗi lời gọi đến **strtok** chia xâu **s1** thành các "tokens"—từ tố, chẳng hạn các từ trong một dòng văn bản—phân tách nhau bởi các ký tự chứa trong xâu **s2**. Lời gọi đầu tiên lấy **s1** làm tham số thứ nhất, các lời gọi tiếp sau (với **NULL** là tham số thứ nhất) tiếp tục lấy các từ tố từ chính xâu đó. Mỗi lời gọi trả về một con trỏ tới từ tố vừa nhận được. Nếu không còn từ tố nào, hàm sẽ trả về giá trị **NULL**.
`size_t strlen( const char *s );`	Xác định độ dài của xâu **s**. Trả về số ký tự của xâu (không tính ký tự null).

5.12.2 Các hàm xử lý xâu ký tự

- Copy xâu
  - **`char *strcpy( char *s1, const char *s2 )`**
    - Copy tham số thứ hai vào tham số thứ nhất
      - Tham số thứ nhất phải có kích thước đủ lớn để chứa xâu và ký tự null
  - **`char *strncpy( char *s1, const char *s2, size_t n )`**
    - Xác định rõ số ký tự được copy từ xâu vào mảng
    - Không nhất thiết copy ký tự null

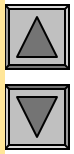


fig05_28.cpp
(1 of 2)

```
1  // Fig. 5.28: fig05_28.cpp
2  // Using strcpy and strncpy.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <cstring>   // prototypes for strcpy and strncpy
9
10 int main()
11 {
12    char x[] = "Happy Birthday to You";
13    char y[ 25 ];
14    char z[ 15 ];
15
16    strcpy( y, x );  // copy contents of x into y
17
18    cout << "The string in array x is: " << x
19         << "\nThe string in array y is: " << y << '\n';
20
21    // copy first 14 characters of x into z
22    strncpy( z, x, 14 );  // does not copy null character
23    z[ 14 ] = '\0';       // append '\0' to z's contents
24
25    cout << "The string in array z is: " << z << endl;
```

<cstring> chứa prototype cho **strcpy** và **strncpy**.

Copy toàn bộ xâu trong mảng **x** vào mảng **y**.

Copy 14 ký tự đầu tiên của mảng **x** vào mảng **y**. Chú ý rằng lệnh này không viết ký tự null.

Thêm ký tự null.

```
26
27     return 0;  // indicates successful termination
28
29  } // end main
```

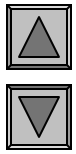
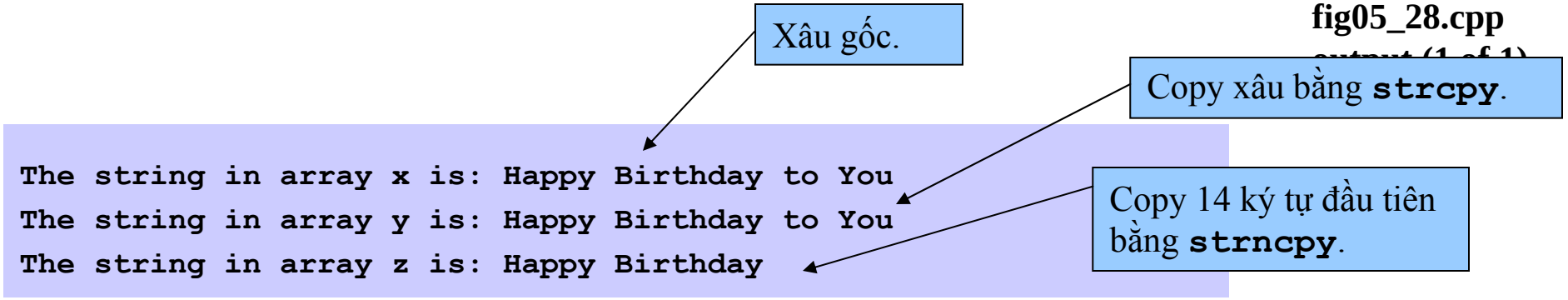


fig05_28.cpp
(2 of 2)

fig05_28.cpp



5.12.2 Các hàm xử lý xâu ký tự

- Nối xâu - Concatenating strings
  - **`char *strcat( char *s1, const char *s2 )`**
    - Nối xâu thứ hai vào sau xâu thứ nhất
    - Ký tự đầu tiên của tham số thứ hai thay thế ký tự null của tham số thứ nhất
    - Phải chắc chắn rằng tham số thứ nhất có kích thước đủ lớn để chứa thêm phần nối vào và ký tự null kết thúc xâu.
  - **`char *strncat( char *s1, const char *s2, size_t n )`**
    - Thêm n ký tự của tham số thứ hai vào sau tham số thứ nhất
    - Thêm ký tự null vào kết quả

fig05_29.cpp (1 of 2)

```
1   // Fig. 5.29: fig05_29.cpp
2   // Using strcat and strncat.
3   #include <iostream>
4
5   using std::cout;
6   using std::endl;
7
8   #include <cstring>   // prototypes for strcat and strncat
9
10  int main()
11  {
12     char s1[ 20 ] = "Happy ";
13     char s2[] = "New Year ";
14     char s3[ 40 ] = "";
15
16     cout << "s1 = " << s1 << "\ns2 = " << s2;
17
18     strcat( s1, s2 );  // concatenate s2 to s1
19
20     cout << "\n\nAfter strcat(s1, s2):\ns1 = " << s1
21          << "\ns2 = " << s2;
22
23     // concatenate first 6 characters of s1 to s3
24     strncat( s3, s1, 6 );  // places '\0' after last character
25
```

<cstring> chứa prototype cho **strcat** và **strncat**.

Thêm **s2** vào sau **s1**.

Thêm 6 ký tự đầu tiên của **s1** vào sau **s3**.

```
26     cout << "\n\nAfter strncat(s3, s1, 6):\ns1 = " << s1
27          << "\ns3 = " << s3;
28
29     strcat( s3, s1 );  // concatenate s1 to s3
30     cout << "\n\nAfter strcat(s3, s1):\ns1 = " << s1
31          << "\ns3 = " << s3 << endl;
32
33     return 0;  // indicates successful termination
34
35  } // end main
```

Thêm **s1** vào sau **s3**.

fig05_29.cpp
(2 of 2)

fig05_29.cpp
output (1 of 1)

```
s1 = Happy
s2 = New Year

After strcat(s1, s2):
s1 = Happy New Year
s2 = New Year

After strncat(s3, s1, 6):
s1 = Happy New Year
s3 = Happy

After strcat(s3, s1):
s1 = Happy New Year
s3 = Happy Happy New Year
```

5.12.2 Các hàm xử lý xâu ký tự

- So sánh xâu - Comparing strings
  - Các ký tự được biểu diễn bằng mã dạng số (numeric code)
    - các mã đó được dùng để so sánh các xâu ký tự
  - Các bộ mã ký tự (Character codes / character sets)
    - ASCII "American Standard Code for Information Interchage"
    - EBCDIC "Extended Binary Coded Decimal Interchange Code"
- Các hàm so sánh xâu
  - `int strcmp( const char *s1, const char *s2 )`
    - So sánh từng ký tự một, theo thứ tự từ điển
    - Trả về
      - 0 nếu xâu bằng nhau
      - Giá trị âm nếu xâu thứ nhất nhỏ hơn xâu thứ hai
      - Giá trị dương nếu xâu thứ nhất lớn hơn xâu thứ hai
  - `int strncmp( const char *s1, const char *s2, size_t n )`
    - So sánh n ký tự đầu tiên
    - Dừng so sánh nếu gặp ký tự null của 1 trong 2 tham số

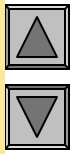


fig05_30.cpp
(1 of 2)

```
1  // Fig. 5.30: fig05_30.cpp
2  // Using strcmp and strncmp.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <iomanip>
9
10 using std::setw;
11
12 #include <cstring>  // prototypes for strcmp and strncmp
13
14 int main()
15 {
16    char *s1 = "Happy New Year";
17    char *s2 = "Happy New Year";
18    char *s3 = "Happy Holidays";
19
20    cout << "s1 = " << s1 << "\ns2 = " << s2
21         << "\ns3 = " << s3 << "\n\nstrcmp(s1, s2) = "
22         << setw( 2 ) << strcmp( s1, s2 )
23         << "\nstrcmp(s1, s3) = " << setw( 2 )
24         << strcmp( s1, s3 ) << "\nstrcmp(s3, s1) = "
25         << setw( 2 ) << strcmp( s3, s1 );
```

<cstring> chứa prototype cho **strcmp** và **strncmp**.

So sánh **s1** với **s2**.

So sánh **s1** với **s3**.

So sánh **s3** với **s1**.

```
26
27     cout << "\n\nstrncmp(s1, s3, 6) = " << setw( 2 )
28          << strncmp( s1, s3, 6 ) << "\nstrncmp(s1, s3, 7) = "
29          << setw( 2 ) << strncmp( s1, s3, 7 )
30          << "\nstrncmp(s3, s1, 7) = "
31          << setw( 2 ) << strncmp( s3, s1, 7 ) << endl;
32
33     return 0;  // indicates successful termination
34
35  } // end main
```

So sánh 6 ký tự đầu tiên của **s1** với **s3**.

So sánh 7 ký tự đầu tiên của **s1** với **s3**.

So sánh 7 ký tự đầu tiên của **s3** với **s1**.

fig05_30.cpp (2 of 2)

fig05_30.cpp (1 of 1)

```
s1 = Happy New Year
s2 = Happy New Year
s3 = Happy Holidays

strcmp(s1, s2) =  0
strcmp(s1, s3) =  1
strcmp(s3, s1) = -1

strncmp(s1, s3, 6) =  0
strncmp(s1, s3, 7) =  1
strncmp(s3, s1, 7) = -1
```

5.12.2 Các hàm xử lý xâu ký tự

- Phân tích từ tố - Tokenizing
  - Chia xâu thành các từ tố, phân tách bởi các ký tự ngăn cách (delimiting character)
  - Các từ tố thường là các đơn vị logic (logical units), chẳng hạn các từ (tách nhau bởi các dấu trống)
  - **`"This is my string"`** có 4 từ tố (tách nhau bởi các dấu trống)
  - **`char *strtok( char *s1, const char *s2 )`**
    - Cần gọi nhiều lần
      - Lần gọi đầu cần 2 tham số, xâu cần phân tích từ tố và xâu chứa các ký tự ngăn cách
        - Tìm ký tự ngăn cách tiếp theo và thay bằng ký tự null
      - Những lời gọi tiếp theo tiếp tục phân tích từ tố trên xâu đó
        - Gọi hàm với tham số thứ nhất là **`NULL`**

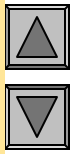


fig05_31.cpp
(1 of 2)

```
1   // Fig. 5.31: fig05_31.cpp
2   // Using strtok.
3   #include <iostream>
4
5   using std::cout;
6   using std::endl;
7
8   #include <cstring>  // prototype for strtok
9
10  int main()
11  {
12     char sentence[] = "This is a sentence with 7 tokens";
13     char *tokenPtr;
14
15     cout << "The string to be tokenized is:\n" << sentence
16          << "\n\nThe tokens are:\n\n";
17
18     // begin tokenization of sentence
19     tokenPtr = strtok( sentence, " " );
20
```

<cstring> chứa prototype cho **strtok**.

Lời gọi **strtok** đầu tiên khởi đầu việc phân tích từ tố.

```
21     // continue tokenizing sentence until tokenPtr becomes NULL
22     while ( tokenPtr != NULL ) {
23        cout << tokenPtr << '\n';
24        tokenPtr = strtok( NULL, " " );  // get next token
25
26     } // end while
27
28     cout << "\nAfter strtok, sentence = " << sentence << endl;
29
30     return 0;  // indicates successful termination
31
32  } // end main
```

fig05_31.cpp (2 of 2)

Các lời gọi **strtok** tiếp sau với **NULL** là tham số đầu để tiếp tục việc phân tích từ tố trên xâu **sentence**.

```
The string to be tokenized is:
This is a sentence with 7 tokens

The tokens are:

This
is
a
sentence
with
7
tokens

After strtok, sentence = This
```

5.12.2 Các hàm xử lý xâu ký tự

- Xác định độ dài xâu
  - `size_t strlen( const char *s )`
    - Trả về số ký tự của xâu
      - Không tính đến ký tự null

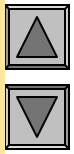


fig05_32.cpp
(1 of 1)

```
1  // Fig. 5.32: fig05_32.cpp
2  // Using strlen.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <cstring>  // prototype for strlen
9
10 int main()
11 {
12    char *string1 = "abcdefghijklmnopqrstuvwxyz";
13    char *string2 = "four";
14    char *string3 = "Boston";
15
16    cout << "The length of \"" << string1
17         << "\" is " << strlen( string1 )
18         << "\nThe length of \"" << string2
19         << "\" is " << strlen( string2 )
20         << "\nThe length of \"" << string3
21         << "\" is " << strlen( string3 ) << endl;
22
23    return 0;  // indicates success
24
25 } // end main
```

<cstring> chứa prototype cho **strlen**.

Sử dụng **strlen** để xác định độ dài xâu.

```
The length of "abcdefghijklmnopqrstuvwxyz" is 26
The length of "four" is 4
The length of "Boston" is 6
```

Ngôn ngữ lập trình C++

Chương 6 – Cấu trúc dữ liệu trừu tượng

Chương 6: Cấu trúc dữ liệu trừu tượng

Đề mục

Tài liệu đọc thêm

- Day 6. TY21 (lập trình cơ bản)
- Chap 4,5. Introduction to OOP Using C++ (IOOP) (khái niệm hướng đối tượng)

6.1 Giới thiệu

- các kiểu dữ liệu phức hợp cấu tạo từ các thành phần thuộc các kiểu dữ liệu khác
  - tạo kiểu dữ liệu mới - kiểu dữ liệu người dùng tự định nghĩa (user-defined data type)
- bản ghi
  - gồm nhiều trường, mỗi trường lưu trữ một thành viên dữ liệu thuộc một kiểu dữ liệu cài sẵn hoặc một kiểu dữ liệu người dùng khác.
- ví dụ
  - Thời gian(giờ, phút, giây) 17:10:02, 04:23:12,...
  - Họ tên (họ, đệm, tên) (Nguyễn, Văn, An), (Lê, Thị, Bình),...

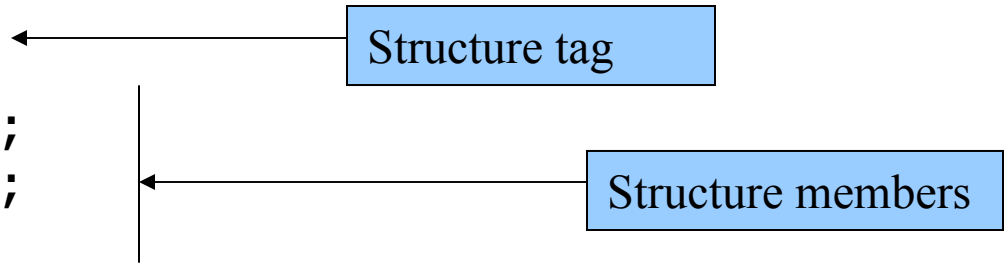
6.1 Giới thiệu

- C++:
  - struct và class - kiểu bản ghi
  - đối tượng (một thể hiện của một kiểu struct hay class nào đó) - bản ghi
  - thành viên dữ liệu - trường
  - hàm thành viên/phương thức - thao tác trên các thành viên dữ liệu

6.2 Cấu trúc - struct

- **`struct`** definition

```
struct Time {
      int hour;
      int minute;
      int second;
};
```



- quy tắc đặt tên cho các thành viên của cấu trúc
  - trong cùng **`struct`**: không thể trùng tên
  - trong các **`struct`** khác nhau: có thể trùng tên
- định nghĩa **`struct`** phải kết thúc bằng dấu chấm phảy.
  - Các biến kiểu cấu trúc được khai báo như các biến thuộc các loại khác
  - Ví dụ: khai báo biến đơn, mảng, con trỏ, tham chiếu...
    - **`Time timeObject;`**
    - **`Time timeArray[ 10 ];`**
    - **`Time *timePtr;`**
    - **`Time &timeRef = timeObject;`**

6.2 Cấu trúc - `struct`

- Self-referential structure - cấu trúc đệ quy
  - thành viên của một cấu trúc không thể thuộc kiểu cấu trúc đó
  - thành viên của một cấu trúc có thể là con trỏ đến kiểu cấu trúc đó (self-referential structure - cấu trúc đệ quy)
    - sử dụng cho danh sách liên kết (linked list), hàng đợi (queue), ngăn xếp (stack), và cây (tree)

```
struct Node {
        int data;
        Node* next;
  };
```

6.3 Truy nhập các thành viên của `struct`

- các toán tử truy nhập thành viên (member access operator)
  - Toán tử dấu chấm (`.`) truy nhập trực tiếp đến các thành viên của cấu trúc/lớp
  - Toán tử mũi tên (`->`) truy nhập các thành viên qua con trỏ đến đối tượng
  - Ví dụ: in thành viên **`hour`** của đối tượng **`timeObject`**:

```
cout << timeObject.hour;
```

hoặc

```
timePtr = &timeObject;
cout << timePtr->hour;
```

  - **`timePtr->hour`** tương đương **`( *timePtr ).hour`**
    - Cần có cặp ngoặc do `*` không được ưu tiên bằng `.`

fig06_01.cpp
(1 of 3)

```
1   // Fig. 6.1: fig06_01.cpp
2   // Create a structure, set its members, and print it.
3   #include <iostream>
4
5   using std::cout;
6   using std::endl;
7
8   #include <iomanip>
9
10  using std::setfill;
11  using std::setw;
12
13  // structure definition
14  struct Time {
15     int hour;     // 0-23 (24-hour clock format)
16     int minute;   // 0-59
17     int second;   // 0-59
18
19  }; // end struct Time
20
21  void printUniversal( const Time & );  // prototype
22  void printStandard( const Time & );   // prototype
23
```

Định nghĩa kiểu cấu trúc **`Time`** với 3 thành viên là số nguyên.

Truyền tham chiếu tới hằng **`Time`** để tránh sao chép tham số.

```
24 int main()
25 {
26    Time dinnerTime;          // variable o
27
28    dinnerTime.hour = 18;   // set hour member of dinnerTime
29    dinnerTime.minute = 30; // set minute member of dinnerTime
30    dinnerTime.second = 0;  // set second member of dinnerTime
31
32    cout << "Dinner will be held at ";
33    printUniversal( dinnerTime );
34    cout << " universal time,\nwhich is ";
35    printStandard( dinnerTime );
36    cout << " standard time.\n";
37
38    dinnerTime.hour = 29;   // set hour to invalid value
39    dinnerTime.minute = 73;  // set minute to invalid value
40
41    cout << "\nTime with invalid values: ";
42    printUniversal( dinnerTime );
43    cout << endl;
44
45    return 0;
46
47 } // end main
48
```

Sử dụng ký hiệu dấu chấm để khởi tạo các thành viên cấu trúc.

Quyền truy nhập trực tiếp tới dữ liệu cho phép gán các giá trị không hợp lệ.

6_01.cpp
(2 of 3)

```
49  // print time in universal-time format
50  void printUniversal( const Time &t )
51  {
52     cout << setfill( '0' ) << setw( 2 ) << t.hour << ":"
53          << setw( 2 ) << t.minute << ":"
54          << setw( 2 ) << t.second;
55
56  } // end function printUniversal
57
58  // print time in standard-time format
59  void printStandard( const Time &t )
60  {
61     cout << ( ( t.hour == 0 || t.hour == 12 ) ?
62               12 : t.hour % 12 ) << ":" << setfill( '0' )
63          << setw( 2 ) << t.minute << ":"
64          << setw( 2 ) << t.second
65          << ( t.hour < 12 ? " AM" : " PM" );
66
67  } // end function printStandard
```

fig06_01.cpp
(3 of 3)

Sử dụng manipulator **`setfill`**.

Dùng dấu chấm để truy nhập các thành viên dữ liệu.

```
Dinner will be held at 18:30:00 universal time,
which is 6:30:00 PM standard time.

Time with invalid values: 29:73:00
```

6.4 Cài đặt kiểu dữ liệu người dùng `Time` bằng `struct`

- Truyền tham số:
  - Mặc định **`struct`** được truyền bằng giá trị
  - Nên truyền **`struct`** bằng tham chiếu để tránh được việc phải sao chép cấu trúc

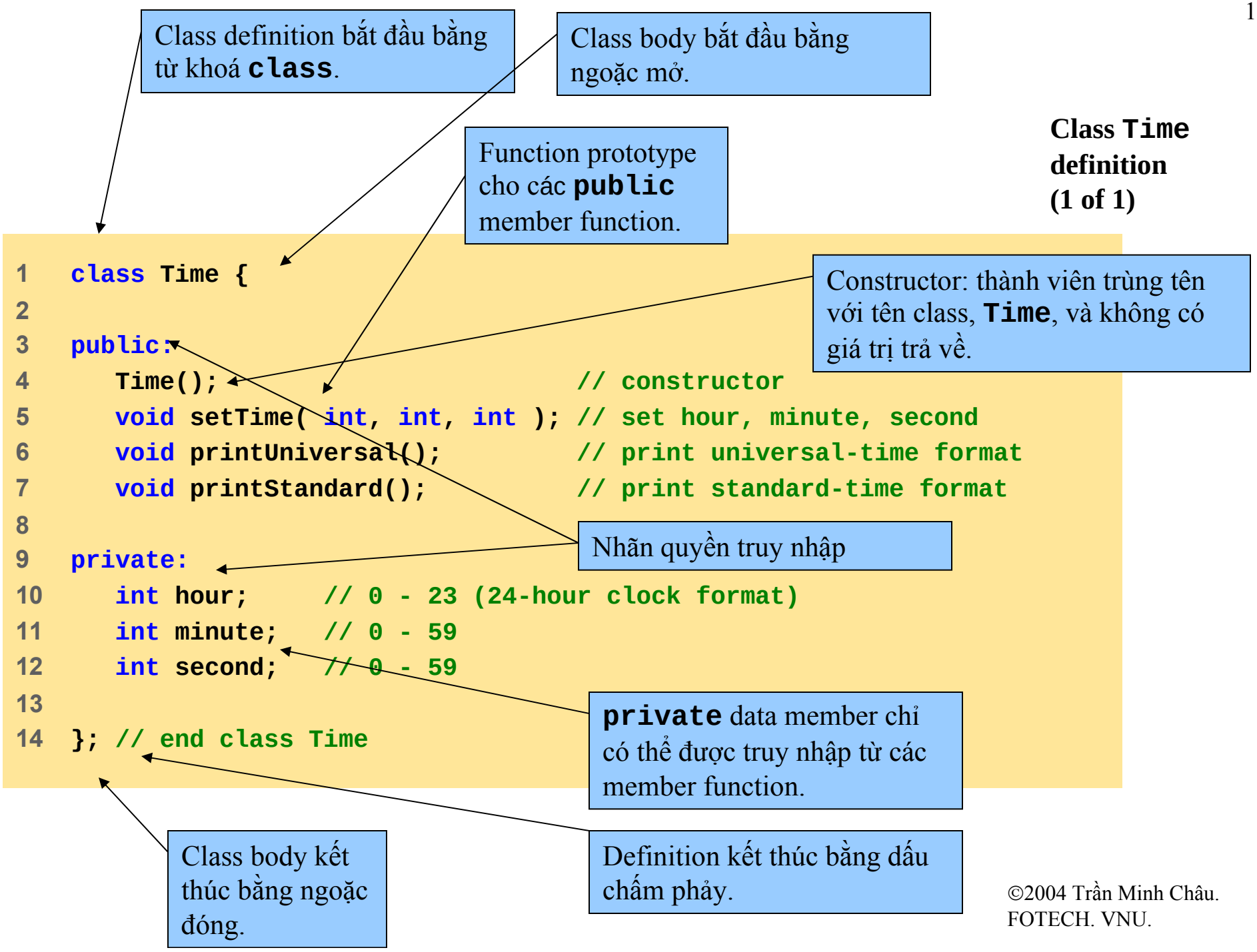
6.4 Cài đặt kiểu dữ liệu người dùng `Time` bằng `struct`

- **`struct`** kiểu C
  - không có giao diện giữa bên trong và bên ngoài cấu trúc
    - Nếu cài đặt thay đổi, mọi chương trình sử dụng **`struct`** đó phải được sửa đổi theo
  - không thể in ra như là một biến đơn
    - Phải in/định dạng cho từng thành viên
  - không thể so sánh hai **`struct`** theo kiểu thông thường
    - Phải so sánh từng thành viên
- **`struct`** kiểu C++
  - C++ mở rộng: **`struct`** có chức năng như **`class`**
  - thông lệ: **`struct`** chỉ được dùng cho các cấu trúc chỉ gồm dữ liệu; **`class`** dùng cho các lớp có cả dữ liệu và hàm thành viên.

6.5 Cài đặt một kiểu dữ liệu trừu tượng `Time` bằng một lớp - `class`

- Các lớp - Classes
  - mô hình các đối tượng
    - Thuộc tính - Attributes (data members)
    - Hành vi - Behaviors (member functions)
  - từ khoá **`class`**
  - các hàm thành viên – member functions
    - còn được gọi là các phương thức - method
    - được gọi để trả lời các thông điệp

Class `Time` definition (1 of 1)




```
1   class Time {
2
3   public:
4      Time();                        // constructor
5      void setTime( int, int, int ); // set hour, minute, second
6      void printUniversal();         // print universal-time format
7      void printStandard();          // print standard-time format
8
9   private:
10     int hour;     // 0 - 23 (24-hour clock format)
11     int minute;   // 0 - 59
12     int second;   // 0 - 59
13
14  }; // end class Time
```

6.5 Cài đặt một kiểu dữ liệu trừu tượng `Time` bằng một lớp - `class`

- Nhãn quyền truy nhập – Member access specifiers
  - quy định quyền truy nhập các thành viên của lớp từ các đoạn trình bên ngoài định nghĩa lớp
  - **`public:`**
    - thành viên có thể được truy nhập từ trong toàn bộ phạm vi của đối tượng
  - **`private:`**
    - thành viên chỉ có thể được truy nhập từ các hàm thành viên của chính lớp đó
  - **`protected:`**
    - dùng cho quan hệ thừa kế

6.5 Cài đặt một kiểu dữ liệu trừu tượng `Time` bằng một lớp - `class`

- Constructor – phương thức khởi tạo
  - hàm thành viên đặc biệt
    - khởi tạo các thành viên dữ liệu
    - trùng tên với tên lớp
  - được gọi khi đối tượng được tạo, ví dụ khi biến được khai báo
  - có thể có vài constructor
    - hoạt động theo nguyên tắc hàm gọi chồng
  - không có giá trị trả về và không có kiểu giá trị trả về

```
class Time {
   public:
      Time();
...
};
```

```
...
Time::Time()
{
    hour = minute = second = 0;
} // end Time constructor
```


6.5 Cài đặt một kiểu dữ liệu trừu tượng `Time` bằng một lớp - `class`

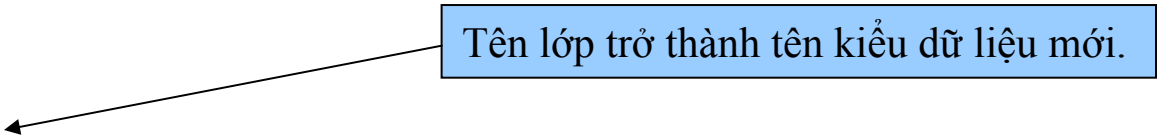
- Destructor – phương thức hủy
  - trùng tên với tên lớp
    - bắt đầu bằng dấu (~)
  - không có tham số
  - tối đa 1 destructor, không thể bị gọi chồng
  - dành cho việc dọn dẹp, chẳng hạn bộ nhớ

```
class Time {
   public:
      Time();
      ~Time();
...
};
```

```
...
Time::~Time()
{
    //empty
} // end Time destructor
```


6.5 Cài đặt một kiểu dữ liệu trừu tượng `Time` bằng một lớp - `class`

- các đối tượng của một lớp
  - Kể từ sau class definition
    - tên lớp trở thành tên kiểu mới - type specifier
      - C++ là ngôn ngữ mở rộng được
    - có thể khai báo đối tượng, mảng đối tượng, con trỏ và tham chiếu tới đối tượng
  - Ví dụ:




```
Time sunset;                    // object of type Time
Time arrayOfTimes[ 5 ];         // array of Time objects
Time *pointerToTime;            // pointer to a Time object
Time &dinnerTime = sunset;      // reference to a Time object
```

6.5 Cài đặt một kiểu dữ liệu trừu tượng `Time` bằng một lớp - `class`

- Các hàm thành viên được định nghĩa bên ngoài lớp
  - toán tử phạm vi (`::`)
    - gắn tên thành viên với tên lớp
    - xác định duy nhất các hàm của một lớp nào đó
    - các lớp khác nhau có thể có các hàm thành viên trùng tên
  - Công thức định nghĩa hàm thành viên

```
ReturnType ClassName::MemberFunctionName( )
{
  …
}
```

  - như nhau đối với hàm public hay private

6.5 Cài đặt một kiểu dữ liệu trừu tượng `Time` bằng một lớp - `class`

- Các hàm thành viên được định nghĩa bên trong lớp
  - Không cần toán tử phạm vi (::) và tên lớp
  - Trình biên dịch sẽ chuyển thành hàm **`inline`** nếu có thể
    - Bên ngoài lớp, các hàm inline cần từ khoá **`inline`**

```
1   // Fig. 6.3: fig06_03.cpp
2   // Time class.
3   #include <iostream>
4
5   using std::cout;
6   using std::endl;
7
8   #include <iomanip>
9
10  using std::setfill;
11  using std::setw;
12
13  // Time abstract data type (ADT) definition
14  class Time {
15
16  public:
17     Time();                          // constructor
18     void setTime( int, int, int ); // set hour, minute, second
19     void printUniversal();           // print universal-time format
20     void printStandard();            // print standard-time format
21
```

fig06_03.cpp
(1 of 5)

Định nghĩa lớp **`Time`**.

```
22 private:
23    int hour;     // 0 - 23 (24-hour clock format)
24    int minute;   // 0 - 59
25    int second;   // 0 - 59
26
27 }; // end class Time
28
29 // Time constructor initializes each data member to zero and
30 // ensures all Time objects start in a consistent state
31 Time::Time()
32 {
33    hour = minute = second = 0;
34
35 } // end Time constructor
36
37 // set new Time value using universal time, perform validity
38 // checks on the data values and set invalid values to zero
39 void Time::setTime( int h, int m, int s )
40 {
41    hour = ( h >= 0 && h < 24 ) ? h : 0;
42    minute = ( m >= 0 && m < 60 ) ? m : 0;
43    second = ( s >= 0 && s < 60 ) ? s : 0;
44
45 } // end function setTime
46
```

fig06_03.cpp
(2 of 5)

Constructor khởi tạo các thành viên dữ liệu **`private`** về **`0`**.

Hàm thành viên **`public`** kiểm tra tính hợp lệ của giá trị các đối số trước khi gán trị cho các thành viên dữ liệu **`private`**

```
47 // print Time in universal format
48 void Time::printUniversal()
49 {
50    cout << setfill( '0' ) << setw( 2 ) << hour << ":"
51         << setw( 2 ) << minute << ":"
52         << setw( 2 ) << second;
53
54 } // end function printUniversal
55
56 // print Time in standard format
57 void Time::printStandard()
58 {
59    cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
60         << ":" << setfill( '0' ) << setw( 2 ) << minute
61         << ":" << setw( 2 ) << second
62         << ( hour < 12 ? " AM" : " PM" );
63
64 } // end function printStandard
65
66 int main()
67 {
68    Time t;  // instantiate object t of class Time
69
```

Không có tham số (ngầm hiểu mục đích là in các thành viên dữ liệu); lời gọi hàm thành viên ngắn gọn hơn lời gọi hàm thường.

Khai báo biến **t** là đối tượng thuộc lớp **Time**.

fig06_03.cpp (3 of 5)

```
70     // output Time object t's initial values
71     cout << "The initial universal time is ";
72     t.printUniversal();   // 00:00:00
73
74     cout << "\nThe initial standard time is ";
75     t.printStandard();    // 12:00:00 AM
76
77     t.setTime( 13, 27, 6 ); // change time
78
79     // output Time object t's new values
80     cout << "\n\nUniversal time after setTime is ";
81     t.printUniversal();   // 13:27:06
82
83     cout << "\nStandard time after setTime is ";
84     t.printStandard();    // 1:27:06 PM
85
86     t.setTime( 99, 99, 99 );  // attempt invalid settings
87
88     // output t's values after specifying invalid values
89     cout << "\n\nAfter attempting invalid settings:"
90          << "\nUniversal time: ";
91     t.printUniversal();   // 00:00:00
92
```

fig06_03.cpp

Gọi các hàm thành viên **public** để in thời gian.

Dùng hàm thành viên **public** để gán trị cho các thành viên dữ liệu.

Thử gán các giá trị không hợp lệ cho các thành viên dữ liệu bằng cách sử dụng hàm thành viên **public**

```
93     cout << "\nStandard time: ";
94     t.printStandard();    // 12:00:00 AM
95     cout << endl;
96
97     return 0;
98
99  } // end main
```

fig06_03.cpp
(5 of 5)

fig06_03.cpp
output (1 of 1)

```
The initial universal time is 00:00:00
The initial standard time is 12:00:00 AM

Universal time after setTime is 13:27:06
Standard time after setTime is 1:27:06 PM

After attempting invalid settings:
Universal time: 00:00:00
Standard time: 12:00:00 AM
```

Các thành viên dữ liệu được gán về **0** sau khi thử các giá trị không hợp lệ.

6.5 Cài đặt một kiểu dữ liệu trừu tượng `Time` bằng một lớp - `class`

- lợi ích khi dùng lớp
  - đơn giản hóa việc lập trình
  - các giao diện – Interfaces
    - che dấu phần cài đặt – Hide implementation
  - tái sử dụng phần mềm – Software reuse
    - khả năng tích hợp – Composition (aggregation)
      - các thành viên của một lớp có thể là đối tượng thuộc lớp khác
    - thừa kế - Inheritance
      - các lớp mới được tạo từ lớp cũ

6.6 Phạm vi lớp và truy nhập các thành viên của lớp

- phạm vi lớp – Class scope
  - gồm thành viên dữ liệu và hàm thành viên của lớp
  - bên trong phạm vi lớp
    - Các thành viên của lớp
      - có thể được truy nhập thẳng từ mọi hàm thành viên
      - gọi bằng tên
  - bên ngoài phạm vi lớp
    - được gọi đến bằng tên đối tượng, tham chiếu/con trỏ tới đối tượng
      - `objectTime.printStandard()`

6.6 Phạm vi lớp và truy nhập các thành viên của lớp

- Phạm vi file - File scope
  - áp dụng cho các hàm không phải thành viên
- Phạm vi hàm – Function scope
  - Gồm các biến được khai báo trong hàm thành viên
  - chỉ được biết đến trong hàm đó
  - bị hủy khi hàm kết thúc
  - các biến trùng tên với biến thuộc phạm vi lớp
    - biến thuộc phạm vi lớp (class-scope variable) bị che (“hidden”)
      - truy nhập bằng toán tử phạm vi (`::`)

```
ClassName::classVariableName
```


6.6 Phạm vi lớp và truy nhập các thành viên của lớp

- Các toán tử để truy nhập các thành viên của đối tượng
  - giống các toán tử dành cho **`struct`**
  - toán tử (`.`) dùng cho
    - đối tượng
    - tham chiếu đến đối tượng
  - toán tử (`->`) dùng cho
    - các con trỏ tới đối tượng

```
1   // Fig. 6.4: fig06_04.cpp
2   // Demonstrating the class member access operators . and ->
3   //
4   // CAUTION: IN FUTURE EXAMPLES WE AVOID PUBLIC DATA!
5   #include <iostream>
6
7   using std::cout;
8   using std::endl;
9
10  // class Count definition
11  class Count {
12
13  public:
14     int x;
15
16     void print()
17     {
18        cout << x << endl;
19     }
20
21  }; // end class Count
22
```

Thành viên dữ liệu **`public x`** minh họa các toán tử truy nhập; thông thường các thành viên dữ liệu đều là **`private`**.

fig06_04.cpp
(1 of 2)

```
23  int main()
24  {
25     Count counter;                  // create counter object
26     Count *counterPtr = &counter;
27     Count &counterRef = counter;
28
29     cout << "Assign 1 to x and print using the object's name: ";
30     counter.x = 1;          // assign 1 to data member x
31     counter.print();        // call me
32
33     cout << "Assign 2 to x and print using a reference: ";
34     counterRef.x = 2;       // assig
35     counterRef.print();     // call
36
37     cout << "Assign 3 to x and print using a pointer: ";
38     counterPtr->x = 3;      // assign 3 to data member x
39     counterPtr->print(); // call member function print
40
41     return 0;
42
43  } // end main
```

Sử dụng dấu chấm cho đối tượng **counter**.

Sử dụng dấu chấm cho **counterRef** là tham chiếu đến đối tượng.

Sử dụng mũi tên cho **counterPtr** là con trỏ tới đối tượng.

fig06_04.cpp (2 of 2)

fig06_04.cpp output (1 of 1)

```
Assign 1 to x and print using the object's name: 1
Assign 2 to x and print using a reference: 2
Assign 3 to x and print using a pointer: 3
```

6.7 Tách giao diện ra khỏi cài đặt

- Tách giao diện khỏi cài đặt
  - ích lợi
    - dễ sửa đổi chương trình
  - bất lợi
    - phải tạo các file header gồm
      - một phần của cài đặt
        - Inline member functions – các hàm inline
      - gợi ý về phần khác của cài đặt
        - private members

6.7 Tách giao diện ra khỏi cài đặt

- Các file header
  - chứa các định nghĩa lớp và các nguyên mẫu hàm
  - được include trong mỗi file sử dụng lớp đó
    - `#include`
  - mở rộng của file `.h`
- Các file mã nguồn – Source-code files
  - chứa định nghĩa của các hàm thành viên
  - trùng tên file với file header tương ứng (không kể phần mở rộng)
    - đây chỉ là thông lệ, không bắt buộc
  - được biên dịch và liên kết với file chương trình chính

time1.h (1 of 1)

```
1   // Fig. 6.5: time1.h
2   // Declaration of class Time.
3   // Member functions are defined in time1.cpp
4
5   // prevent multiple inclusions of header file
6   #ifndef TIME1_H
7   #define TIME1_H
8
9   // Time abstract data type definit
10  class Time {
11
12  public:
13     Time();                             // constructor
14     void setTime( int, int, int );      // set hour, minute, second
15     void printUniversal();              // print universal-time format
16     void printStandard();               // print standard-time format
17
18  private:
19     int hour;      // 0 - 23 (24-hour clock format)
20     int minute;    // 0 - 59
21     int second;    // 0 - 59
22
23  }; // end class Time
24
25  #endif
```

Mã tiền xử lý để tránh việc file bị include nhiều lần.

"If not defined"

Mã giữa hai định hướng này không được include nếu tên **TIME1_H** đã được định nghĩa.

Định hướng tiền xử lý định nghĩa tên **TIME1_H**.

Thông lệ đặt tên: tên header file với dấu gạch dưới thay cho dấu chấm.

time1.cpp (1 of 3)

```
1   // Fig. 6.6: time1.cpp
2   // Member-function definitions for class Time.
3   #include <iostream>
4
5   using std::cout;
6
7   #include <iomanip>
8
9   using std::setfill;
10  using std::setw;
11
12  // include definition of class Time from time1.h
13  #include "time1.h"
14
15  // Time constructor initializes each data member to zero.
16  // Ensures all Time objects start in a consistent state.
17  Time::Time()
18  {
19     hour = minute = second = 0;
20
21  } // end Time constructor
22
```

Include header file **time1.h**.

Tên của header file đặt trong ngoặc kép; cặp ngoặc nhọn làm trình biên dịch cho rằng đó là một phần của thư viện chuẩn C++ (C++ Standard Library).

time1.cpp (2 of 3)

```
23 // Set new Time value using universal time. Perform validity
24 // checks on the data values. Set invalid values to zero.
25 void Time::setTime( int h, int m, int s )
26 {
27    hour = ( h >= 0 && h < 24 ) ? h : 0;
28    minute = ( m >= 0 && m < 60 ) ? m : 0;
29    second = ( s >= 0 && s < 60 ) ? s : 0;
30
31 } // end function setTime
32
33 // print Time in universal format
34 void Time::printUniversal()
35 {
36    cout << setfill( '0' ) << setw( 2 ) << hour << ":"
37         << setw( 2 ) << minute << ":"
38         << setw( 2 ) << second;
39
40 } // end function printUniversal
41
```

```
42  // print Time in standard format
43  void Time::printStandard()
44  {
45     cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
46          << ":" << setfill( '0' ) << setw( 2 ) << minute
47          << ":" << setw( 2 ) << second
48          << ( hour < 12 ? " AM" : " PM" );
49
50  } // end function printStandard
```

time1.cpp (3 of 3)

fig06_07.cpp (1 of 2)

```
1   // Fig. 6.7: fig06_07.cpp
2   // Program to test class Time.
3   // NOTE: This file must be compiled with time1.cpp.
4   #include <iostream>
5
6   using std::cout;
7   using std::endl;
8
9   // include definition of class Time from time1.h
10  #include "time1.h"
11
12  int main()
13  {
14     Time t;  // instantiate object t of class Time
15
16     // output Time object t's initial values
17     cout << "The initial universal time is ";
18     t.printUniversal();   // 00:00:00
19     cout << "\nThe initial standard time is ";
20     t.printStandard();    // 12:00:00 AM
21
22     t.setTime( 13, 27, 6 );   // change time
23
```

Include **time1.h** để đảm bảo tạo đúng và để tính kích thước đối tượng thuộc lớp **Time**.

```
24     // output Time object t's new values
25     cout << "\n\nUniversal time after setTime is ";
26     t.printUniversal();   // 13:27:06
27     cout << "\nStandard time after setTime is ";
28     t.printStandard();    // 1:27:06 PM
29
30     t.setTime( 99, 99, 99 );  // attempt invalid settings
31
32     // output t's values after specifying invalid values
33     cout << "\n\nAfter attempting invalid settings:"
34          << "\nUniversal time: ";
35     t.printUniversal();   // 00:00:00
36     cout << "\nStandard time: ";
37     t.printStandard();    // 12:00:00 AM
38     cout << endl;
39
40     return 0;
41
42  } // end main
```

fig06_07.cpp
(2 of 2)

fig06_07.cpp
output (1 of 1)

```
The initial universal time is 00:00:00
The initial standard time is 12:00:00 AM

Universal time after setTime is 13:27:06
Standard time after setTime is 1:27:06 PM
```

6.8 Quản lý quyền truy nhập thành viên

- các kiểu truy nhập – Access
  - **`private`**
    - kiểu mặc định - Default access mode
    - chỉ có các hàm thành viên và các hàm **`friend`** là có thể truy nhập các thành viên **`private`**
  - **`public`**
    - truy nhập được từ mọi hàm trong chương trình.
  - **`protected`**
    - dành cho quan hệ thừa kế, hiện tại chưa nói đến

```
1   // Fig. 6.8: fig06_08.cpp
2   // Demonstrate errors resulting from attempts
3   // to access private class members.
4   #include <iostream>
5
6   using std::cout;
7
8   // include definition of class Time from time1.h
9   #include "time1.h"
10
11  int main()
12  {
13     Time t;  // create Time object
14
15
16     t.hour = 7;  // error: 'Time::hour' is not accessible
17
18     // error: 'Time::minute' is not accessible
19     cout << "minute = " << t.minute;
20
21     return 0;
22
23  } // end main
```

fig06_08.cpp
(1 of 1)

hour là thành viên **private**;
truy nhập các thành viên **private** sẽ gây lỗi.

minute cũng là **private**;

6.8 Quản lý quyền truy nhập thành viên

- quyền truy nhập các thành viên của **`class`**
  - mặc định **`private`**
  - phải đặt tường minh **`public`**, **`protected`**
- quyền truy nhập các thành viên của **`struct`**
  - mặc định **`public`**
  - phải đặt tường minh **`private`**, **`protected`**
- truy nhập dữ liệu **`private`** của lớp
  - các hàm truy nhập (accessor method)
    - Get function – hàm đọc dữ liệu
      - đọc dữ liệu **`private`**
    - Set function – hàm ghi dữ liệu
      - ghi dữ liệu **`private`**

6.9 Các hàm truy nhập và các hàm tiện ích

- Các hàm truy nhập – Access functions
  - **`public`**
  - các hàm đọc và hiển thị dữ liệu
  - các hàm ghi dữ liệu (kèm kiểm tra tính hợp lệ)
  - các hàm mệnh đề – Predicate functions
    - kiểm tra các điều kiện
- Các hàm tiện ích – Utility functions
  - **`private`**
  - chỉ hỗ trợ hoạt động của các hàm thành viên kiểu **`public`**
  - không nhằm mục đích để cho client trực tiếp sử dụng

salesp.h (1 of 1)

```
1   // Fig. 6.9: salesp.h
2   // SalesPerson class definition.
3   // Member functions defined in salesp.cpp.
4   #ifndef SALESP_H
5   #define SALESP_H
6
7   class SalesPerson {
8
9   public:
10     SalesPerson();                  // constructor
11     void getSalesFromUser();        // input sales from keyboard
12     void setSales( int, double ); // set sales for a month
13     void printAnnualSales();        // summarize and print sales
14
15  private:
16     double totalAnnualSales();    // utility function
17     double sales[ 12 ];           // 12 monthly sales figures
18
19  }; // end class SalesPerson
20
21  #endif
```

hàm ghi dữ liệu thực hiện việc kiểm tra tính hợp lệ của dữ liệu (validity checks).

hàm tiện ích **private**

salesp.cpp (1 of 3)

```
1  // Fig. 6.10: salesp.cpp
2  // Member functions for class SalesPerson.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8  using std::fixed;
9
10 #include <iomanip>
11
12 using std::setprecision;
13
14 // include SalesPerson class definition from salesp.h
15 #include "salesp.h"
16
17 // initialize elements of array sales to 0.0
18 SalesPerson::SalesPerson()
19 {
20    for ( int i = 0; i < 12; i++ )
21       sales[ i ] = 0.0;
22
23 } // end SalesPerson constructor
24
```

lesp.cpp (2 of 3)

```
25 // get 12 sales figures from the user at the keyboard
26 void SalesPerson::getSalesFromUser()
27 {
28    double salesFigure;
29
30    for ( int i = 1; i <= 12; i++ ) {
31       cout << "Enter sales amount for month " << i << ": ";
32       cin >> salesFigure;
33       setSales( i, salesFigure );
34
35    } // end for
36
37 } // end function getSalesFromUser
38
39 // set one of the 12 monthly sales figures; function subtracts
40 // one from month value for proper subscript in sales array
41 void SalesPerson::setSales( int month, double amount )
42 {
43    // test for valid month and amount values
44    if ( month >= 1 && month <= 12 && amount > 0 )
45       sales[ month - 1 ] = amount; // adjust for subscripts 0-11
46
47    else // invalid month or amount value
48       cout << "Invalid month or sales figure" << endl;
```

hàm ghi dữ liệu thực hiện việc kiểm tra tính hợp lệ của dữ liệu (validity checks).

lesp.cpp (3 of 3)

```
49
50 } // end function setSales
51
52 // print total annual sales (with help of utility function)
53 void SalesPerson::printAnnualSales()
54 {
55    cout << setprecision( 2 ) << fixed
56         << "\nThe total annual sales are: $"
57         << totalAnnualSales() << endl; // call utility function
58
59 } // end function printAnnualSales
60
61 // private utility function to total annual sales
62 double SalesPerson::totalAnnualSales()
63 {
64    double total = 0.0;              // initialize total
65
66    for ( int i = 0; i < 12; i++ )  // summarize sales results
67       total += sales[ i ];
68
69    return total;
70
71 } // end function totalAnnualSales
```

Hàm tiện ích **private** phục vụ hàm **printAnnualSales**; đóng gói thao tác trên mảng **sales**.

fig06_11.cpp
(1 of 1)

```
1   // Fig. 6.11: fig06_11.cpp
2   // Demonstrating a utility function.
3   // Compile this program with salesp.cpp
4
5   // include SalesPerson class definition from salesp.h
6   #include "salesp.h"
7
8   int main()
9   {
10     SalesPerson s;          // create SalesPerson object s
11
12     s.getSalesFromUser();  // note simple sequential code; no
13     s.printAnnualSales();  // control structures in main
14
15     return 0;
16
17  } // end main
```

Chuỗi gọi hàm đơn giản;
logic chương trình được đóng gói trong các hàm thành viên.

```
Enter sales amount for month 1: 5314.76
Enter sales amount for month 2: 4292.38
Enter sales amount for month 3: 4589.83
Enter sales amount for month 4: 5534.03
Enter sales amount for month 5: 4376.34
Enter sales amount for month 6: 5698.45
Enter sales amount for month 7: 4439.22
Enter sales amount for month 8: 5893.57
Enter sales amount for month 9: 4909.67
Enter sales amount for month 10: 5123.45
Enter sales amount for month 11: 4024.97
Enter sales amount for month 12: 5923.92

The total annual sales are: $60120.59
```

fig06_11.cpp
output (1 of 1)

6.10 Khởi tạo các đối tượng: Constructor

- Constructors
  - khởi tạo các thành viên dữ liệu
    - hoặc có thể gán trị cho các thành viên dữ liệu sau
  - trùng tên với tên lớp
  - không có kiểu trả về
- Các giá trị khởi tạo – Initializers
  - được truyền dưới dạng đối số cho constructor
  - khi khai báo biến: đặt trong cặp ngoặc đơn trước dấu chấm phảy

```
class Time {
public:
   Time( int, int, int);
...
}; // end class Time
...
int main()
{
   Time t( 27, 74, 99 );
...
```

Class-type ObjectName(value1,value2,…);

6.11 Sử dụng các đối số mặc định với constructor

- có thể chỉ định các đối số mặc định
  - tương tự đối số mặc định của hàm thông thường
- constructor mặc định:
  - có thể gọi không cần tham số
    - `Time t;`
  - Tất cả các đối số là mặc định HOẶC thực sự không nhận tham số
    - `Time(int = 0, int = 0, int = 0);`

      hoặc
    - `Time();`
  - mỗi lớp chỉ được có tối đa một constructor mặc định

time2.h (1 of 1)

```
1   // Fig. 6.12: time2.h
2   // Declaration of class Time.
3   // Member functions defined in time2.cpp.
4
5   // prevent multiple inclusions of header file
6   #ifndef TIME2_H
7   #define TIME2_H
8
9   // Time abstract data type definition
10  class Time {
11
12  public:
13     Time( int = 0, int = 0, int = 0); // default constructor
14     void setTime( int, int, int ); // set hour, minute, second
15     void printUniversal();         // print universal-time format
16     void printStandard();          // print standard-time format
17
18  private:
19     int hour;     // 0 - 23 (24-hour clock format)
20     int minute;   // 0 - 59
21     int second;   // 0 - 59
22
23  }; // end class Time
24
25  #endif
```

Default constructor chỉ định giá trị mặc định cho mọi đối số.

time2.cpp (1 of 2)

```
1  // Fig. 6.13: time2.cpp
2  // Member-function definitions for class Time.
3  #include <iostream>
4
5  using std::cout;
6
7  #include <iomanip>
8
9  using std::setfill;
10 using std::setw;
11
12 // include definition of class Time from time2.h
13 #include "time2.h"
14
15 // Time constructor initializes each data member to zero;
16 // ensures all Time objects start in a consistent state
17 Time::Time( int hr, int min, int sec )
18 {
19    setTime( hr, min, sec );  // validate and set time
20
21 } // end Time constructor
22
```

Constructor gọi **`setTime`** để kiểm tra các giá trị được truyền vào (hoặc mặc định).

time2.cpp (2 of 2)

```
23 // set new Time value using universal time, perform validity
24 // checks on the data values and set invalid values to zero
25 void Time::setTime( int h, int m, int s )
26 {
27    hour = ( h >= 0 && h < 24 ) ? h : 0;
28    minute = ( m >= 0 && m < 60 ) ? m : 0;
29    second = ( s >= 0 && s < 60 ) ? s : 0;
30
31 } // end function setTime
32
33 // print Time in universal format
34 void Time::printUniversal()
35 {
36    cout << setfill( '0' ) << setw( 2 ) << hour << ":"
37         << setw( 2 ) << minute << ":"
38         << setw( 2 ) << second;
39
40 } // end function printUniversal
41
42 // print Time in standard format
43 void Time::printStandard()
44 {
45    cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
46         << ":" << setfill( '0' ) << setw( 2 ) << minute
47         << ":" << setw( 2 ) << second
48         << ( hour < 12 ? " AM" : " PM" );
49
50 } // end function printStandard
```

fig06_14.cpp
(1 of 2)

```
1  // Fig. 6.14: fig06_14.cpp
2  // Demonstrating a default constructor for class Time.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  // include definition of class Time from time2.h
9  #include "time2.h"
10
11 int main()
12 {
13    Time t1;           // all arguments defaulted
14    Time t2( 2 );      // minute and second defaulted
15    Time t3( 21, 34 ); // second defaulted
16    Time t4( 12, 25, 42 ); // all values specified
17    Time t5( 27, 74, 99 ); // all bad values specified
18
19    cout << "Constructed with:\n\n"
20         << "all default arguments:\n  ";
21    t1.printUniversal();  // 00:00:00
22    cout << "\n  ";
23    t1.printStandard();   // 12:00:00 AM
24
```

Khởi tạo các đối tượng **Time** sử dụng các tham số mặc định.

Khởi tạo đối tượng **Time** với các giá trị không hợp lệ; khâu kiểm tra tính hợp lệ sẽ gán các giá trị về **0**.

```
25     cout << "\n\nhour specified; default minute and second:\n  ";
26     t2.printUniversal();  // 02:00:00
27     cout << "\n  ";
28     t2.printStandard();   // 2:00:00 AM
29
30     cout << "\n\nhour and minute specified; default second:\n  ";
31     t3.printUniversal();  // 21:34:00
32     cout << "\n  ";
33     t3.printStandard();   // 9:34:00 PM
34
35     cout << "\n\nhour, minute, and second specified:\n  ";
36     t4.printUniversal();  // 12:25:42
37     cout << "\n  ";
38     t4.printStandard();   // 12:25:42 PM
39
40     cout << "\n\nall invalid values specified:\n  ";
41     t5.printUniversal();  // 00:00:00
42     cout << "\n  ";
43     t5.printStandard();   // 12:00:00 AM
44     cout << endl;
45
46     return 0;
47
48 } // end main
```

fig06_14.cpp
(2 of 2)

t5 được xây dựng bằng các đối số không hợp lệ, các giá trị được gán về **0**.

6.12 Destructor – hàm hủy

- Destructor – hàm thành viên tự hủy của đối tượng
  - hàm thành viên đặc biệt
  - trùng tên với tên lớp
    - bắt đầu bằng dấu ngã (~)
  - không nhận đối số
  - không có giá trị trả về
  - không thể bị gọi chồng
  - thực hiện việc dọn dẹp
    - trước khi hệ thống lấy lại phần bộ nhớ của đối tượng
      - tái sử dụng cho đối tượng mới
  - nếu không có destructor được định nghĩa tường minh
    - trình biên dịch tự tạo destructor "rỗng" – không làm gì hết

6.13 Khi nào Constructor và Destructor được gọi

- các constructor và destructor
  - được gọi ngầm bởi trình biên dịch
- thứ tự gọi hàm
  - phụ thuộc vào thứ tự thực thi chương trình
    - khi chương trình vào và ra khỏi phạm vi của các đối tượng
  - các đối tượng cũng là các biến thông thường,
    - biến được khởi tạo – constructor được gọi – tại thời điểm bắt đầu tồn tại / phạm vi
    - biến bị hủy – destructor được gọi – khi kết thúc sự tồn tại / ra khỏi phạm vi
  - thông thường, các lời gọi destructor theo thứ tự ngược lại với thứ tự gọi các constructor

6.13 Khi nào Constructor và Destructor được gọi

- Thứ tự các lời gọi constructor, destructor
  - đối với các đối tượng/biến phạm vi toàn cục (global scope objects)
    - Constructor
      - được gọi trước mọi hàm khác (kể cả **`main`**)
    - Destructor
      - được gọi khi **`main`** kết thúc (hoặc khi hàm **`exit`** được gọi)
      - không được gọi nếu chương trình kết thúc bằng hàm **`abort`**

6.13 Khi nào Constructor và Destructor được gọi

- Thứ tự các lời gọi constructor, destructor
  - đối với các đối tượng/biến địa phương (automatic local objects)
    - Constructor
      - được gọi khi đối tượng được định nghĩa
        - mỗi khi chương trình vào phạm vi của đối tượng
    - Destructor
      - được gọi khi đối tượng ra khỏi phạm vi
        - chương trình ra khỏi khối nơi đối tượng được định nghĩa
      - không được gọi nếu chương trình kết thúc bằng **`exit`** hay **`abort`**

6.13 Khi nào Constructor và Destructor được gọi

- Thứ tự các lời gọi constructor, destructor
  - các đối tượng tĩnh địa phương (**`static`** local objects)
    - Constructor
      - đúng một lần
      - khi chương trình chạy đến chỗ đối tượng được định nghĩa
    - Destructor
      - khi hàm **`main`** kết thúc hoặc khi hàm **`exit`** được gọi
      - không được gọi nếu chương trình kết thúc bằng hàm **`abort`**

create.h (1 of 1)

```
1   // Fig. 6.15: create.h
2   // Definition of class CreateAndDestroy.
3   // Member functions defined in create.cpp.
4   #ifndef CREATE_H
5   #define CREATE_H
6
7   class CreateAndDestroy {
8
9   public:
10     CreateAndDestroy( int, char * );  // constructor
11     ~CreateAndDestroy();              // destructor
12
13  private:
14     int objectID;
15     char *message;
16
17  }; // end class CreateAndDestroy
18
19  #endif
```

Các hàm thành viên constructor và destructor

Các thành viên **private** để minh họa thứ tự các lời gọi constructor và destructor

create.cpp (1 of 2)

```
1  // Fig. 6.16: create.cpp
2  // Member-function definitions for class CreateAndDestroy
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  // include CreateAndDestroy class definition from create.h
9  #include "create.h"
10
11 // constructor
12 CreateAndDestroy::CreateAndDestroy(
13    int objectNumber, char *messagePtr )
14 {
15    objectID = objectNumber;
16    message = messagePtr;
17
18    cout << "Object " << objectID << "   constructor runs   "
19         << message << endl;
20
21 } // end CreateAndDestroy constructor
22
```

Output message để thể hiện thời gian của các lời gọi hàm constructor.

```
23  // destructor
24  CreateAndDestroy::~CreateAndDestroy()
25  {
26     // the following line is for pedagogic purposes only
27     cout << ( objectID == 1 || objectID == 6 ? "\n" : "" );
28
29     cout << "Object " << objectID << "   destructor runs    "
30          << message << endl;
31
32  } // end ~CreateAndDestroy destructor
```

Output message để thể hiện thời gian của các lời gọi hàm destructor

create.cpp (2 of 2)

fig06_17.cpp (1 of 2)

```
1  // Fig. 6.17: fig06_17.cpp
2  // Demonstrating the order in which constructors and
3  // destructors are called.
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8
9  // include CreateAndDestroy class definition from create.h
10 #include "create.h"
11
12 void create( void );   // prototype
13
14 // global object
15 CreateAndDestroy first( 1, "(global before main)" );
16
17 int main()
18 {
19    cout << "\nMAIN FUNCTION: EXECUTION BEGINS" << endl;
20
21    CreateAndDestroy second( 2, "(local automatic in main)" );
22
23    static CreateAndDestroy third( 3, "(local static in main)" );
24
25    create();  // call function to create objects
26
```

tạo đối tượng có phạm vi toàn cục

Tạo đối tượng tự động địa phương.

Tạo đối tượng địa phương **static**.

Tạo các đối tượng tự động địa phương.

```
28    cout << "\nMAIN FUNCTION: EXECUTION RESUMES" << endl;
29
30    CreateAndDestroy fourth( 4, "(local automatic in main)" );
31
32    cout << "\nMAIN FUNCTION: EXECUTION ENDS" << endl;
33
34    return 0;
35
36 } // end main
37
38 // function to create objects
39 void create( void )
40 {
41    cout << "\nCREATE FUNCTION: EXECUTION BEGINS" << endl;
42
43    CreateAndDestroy fifth( 5, "(local automatic in create)" );
44
45    static CreateAndDestroy sixth(
46       6, "(local static in create)" );
47
48    CreateAndDestroy seventh(
49       7, "(local automatic in create)" );
50
51    cout << "\nCREATE FUNCTION: EXECUTION ENDS\" << endl;
52
53 } // end function create
```

fig06_17.cpp (2 of 2)

Tạo đối tượng tự động địa phương.

Tạo đối tượng tự động địa phương bên trong hàm.

Tạo đối tượng địa phương **static** bên trong hàm.

Tạo đối tượng tự động địa phương bên trong hàm.

```
Object 1   constructor runs   (global before main)

MAIN FUNCTION: EXECUTION BEGINS
Object 2   constructor runs   (local automatic in main)
Object 3   constructor runs   (local static in main)

CREATE FUNCTION: EXECUTION BEGINS
Object 5   constructor runs   (local automatic in create)
Object 6   constructor runs   (local static in create)
Object 7   constructor runs   (local automatic in create)

CREATE FUNCTION: EXECUTION ENDS
Object 7   destructor runs    (local automatic in create)
Object 5   destructor runs    (local automatic in create)

MAIN FUNCTION: EXECUTION RESUMES
Object 4   constructor runs   (local automatic in main)

MAIN FUNCTION: EXECUTION ENDS
Object 4   destructor runs    (local automatic in main)
Object 2   destructor runs    (local automatic in main)
Object 6   destructor runs    (local static in create)
Object 3   destructor runs    (local static in main)

Object 1   destructor runs    (global before main)
```

fig06_17.cpp
output (1 of 1)

đối tượng toàn cục được tạo

đối tượng **`static`** địa phương được tạo tại lời gọi hàm đầu tiên và hủy sau khi hàm **`main`** kết thúc.

6.14 Sử dụng các hàm truy nhập

- Set functions – các hàm ghi
  - kiểm tra tính hợp lệ trước khi sửa đổi dữ liệu **`private`**
  - thông báo nếu các giá trị là không hợp lệ
  - thông báo qua các giá trị trả về
- Get functions – các hàm đọc
  - các hàm truy vấn – "Query" functions
  - quản lý định dạng của dữ liệu trả về

time3.h (1 of 2)

```
1   // Fig. 6.18: time3.h
2   // Declaration of class Time.
3   // Member functions defined in time3.cpp
4
5   // prevent multiple inclusions of header file
6   #ifndef TIME3_H
7   #define TIME3_H
8
9   class Time {
10
11  public:
12     Time( int = 0, int = 0, int = 0 );  // default constructor
13
14     // set functions
15     void setTime( int, int, int );  // set hour, minute, second
16     void setHour( int );   // set hour
17     void setMinute( int ); // set minute
18     void setSecond( int ); // set second
19
20     // get functions
21     int getHour();         // return hour
22     int getMinute();       // return minute
23     int getSecond();       // return second
24
```

Các hàm ghi

các hàm đọc

time3.h (2 of 2)

```
25     void printUniversal(); // output universal-time format
26     void printStandard();  // output standard-time format
27
28  private:
29     int hour;              // 0 - 23 (24-hour clock format)
30     int minute;            // 0 - 59
31     int second;            // 0 - 59
32
33  }; // end clas Time
34
35  #endif
```

time3.cpp (1 of 4)

```
1  // Fig. 6.19: time3.cpp
2  // Member-function definitions for Time class.
3  #include <iostream>
4
5  using std::cout;
6
7  #include <iomanip>
8
9  using std::setfill;
10 using std::setw;
11
12 // include definition of class Time from time3.h
13 #include "time3.h"
14
15 // constructor function to initialize private data;
16 // calls member function setTime to set variables;
17 // default values are 0 (see class definition)
18 Time::Time( int hr, int min, int sec )
19 {
20    setTime( hr, min, sec );
21
22 } // end Time constructor
23
```

time3.cpp (2 of 4)

```
24  // set hour, minute and second values
25  void Time::setTime( int h, int m, int s )
26  {
27     setHour( h );
28     setMinute( m );
29     setSecond( s );
30
31  } // end function setTime
32
33  // set hour value
34  void Time::setHour( int h )
35  {
36     hour = ( h >= 0 && h < 24 ) ? h : 0;
37
38  } // end function setHour
39
40  // set minute value
41  void Time::setMinute( int m )
42  {
43     minute = ( m >= 0 && m < 60 ) ? m : 0;
44
45  } // end function setMinute
46
```

Gọi các hàm set để kiểm tra tính hợp lệ.

Các hàm set kiểm tra tính hợp lệ trước khi sửa đổi dữ liệu.

time3.cpp (3 of 4)

```
47  // set second value
48  void Time::setSecond( int s )
49  {
50     second = ( s >= 0 && s < 60 ) ? s : 0;
51
52  } // end function setSecond
53
54  // return hour value
55  int Time::getHour()
56  {
57     return hour;
58
59  } // end function getHour
60
61  // return minute value
62  int Time::getMinute()
63  {
64     return minute;
65
66  } // end function getMinute
67
```

Các hàm set kiểm tra tính hợp lệ trước khi sửa đổi dữ liệu.

Các hàm get cho client đọc dữ liệu

time3.cpp (4 of 4)

```
68  // return second value
69  int Time::getSecond()
70  {
71     return second;
72
73  } // end function getSecond
74
75  // print Time in universal format
76  void Time::printUniversal()
77  {
78     cout << setfill( '0' ) << setw( 2 ) << hour << ":"
79          << setw( 2 ) << minute << ":"
80          << setw( 2 ) << second;
81
82  } // end function printUniversal
83
84  // print Time in standard format
85  void Time::printStandard()
86  {
87     cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
88          << ":" << setfill( '0' ) << setw( 2 ) << minute
89          << ":" << setw( 2 ) << second
90          << ( hour < 12 ? " AM" : " PM" );
91
92  } // end function printStandard
```

Hàm get cho client đọc dữ liệu

fig06_20.cpp
(1 of 3)

```
1   // Fig. 6.20: fig06_20.cpp
2   // Demonstrating the Time class set and get functions
3   #include <iostream>
4
5   using std::cout;
6   using std::endl;
7
8   // include definition of class Time from time3.h
9   #include "time3.h"
10
11  void incrementMinutes( Time &, const int );  // prototype
12
13  int main()
14  {
15     Time t;              // create Time object
16
17     // set time using individual set functions
18     t.setHour( 17 );     // set hour to valid value
19     t.setMinute( 34 );   // set minute to valid value
20     t.setSecond( 25 );   // set second to valid value
21
```

Gọi các hàm set để gán các giá trị hợp lệ.

fig06_20.cpp
(3)

```
22    // use get functions to obtain hour, minute and second
23    cout << "Result of setting all valid values:\n"
24         << "  Hour: " << t.getHour()
25         << "  Minute: " << t.getMinute()
26         << "  Second: " << t.getSecond();
27
28    // set time using individual set functions
29    t.setHour( 234 );    // invalid hour set to 0
30    t.setMinute( 43 );   // set minute to valid value
31    t.setSecond( 6373 ); // invalid second set to 0
32
33    // display hour, minute and second after setting
34    // invalid hour and second values
35    cout << "\n\nResult of attempting to set invalid hour and"
36         << " second:\n  Hour: " << t.getHour()
37         << "  Minute: " << t.getMinute()
38         << "  Second: " << t.getSecond() << "\n\n";
39
40    t.setTime( 11, 58, 0 );    // set time
41    incrementMinutes( t, 3 );  // increment t's minute by 3
42
43    return 0;
44
45 } // end main
46
```

Cố dùng các hàm set để gán các giá trị không hợp lệ.

các giá trị không hợp lệ làm các data member bị gán về **0**.

Sửa đổi data member bằng hàm **setTime**.

```
47  // add specified number of minutes to a Time object
48  void incrementMinutes( Time &tt, const int count )
49  {
50     cout << "Incrementing minute " << count
51          << " times:\nStart time: ";
52     tt.printStandard();
53
54     for ( int i = 0; i < count; i++ ) {
55        tt.setMinute( ( tt.getMinute() + 1 ) % 60 );
56
57        if ( tt.getMinute() == 0 )
58           tt.setHour( ( tt.getHour() + 1 ) % 24);
59
60        cout << "\nminute + 1: ";
61        tt.printStandard();
62
63     } // end for
64
65     cout << endl;
66
67  } // end function increment
```

fig06_20.cpp
(3 of 3)

Dùng các hàm get để đọc và các hàm set để sửa dữ liệu.

```
Result of setting all valid values:
  Hour: 17  Minute: 34  Second: 25

Result of attempting to set invalid hour and second:
  Hour: 0  Minute: 43  Second: 0

Incrementing minute 3 times:
Start time: 11:58:00 AM
minute + 1: 11:59:00 AM
minute + 1: 12:00:00 PM
minute + 1: 12:01:00 PM
```

Cố gắng gán các giá trị không hợp lệ cho các thành viên dữ liệu, kết quả là thông báo lỗi và các thành viên bị gán về **0**.

6.15 Phép gán mặc định

- Gán đối tượng cho đối tượng
  - Phép gán (=)
    - có thể gán một đối tượng cho một đối tượng khác thuộc cùng kiểu
    - Mặc định: gán theo từng thành viên (memberwise assignment)
      - Mỗi thành viên của đối tượng vế phải được gán cho thành viên tương ứng tại vế trái
- được ngầm thực hiện khi
  - truyền tham số là đối tượng
  - trả về đối tượng
- Đối tượng có thể được truyền làm tham số cho hàm
  - Đối tượng có thể được hàm trả về
  - Mặc định: pass-by-value
    - Bản sao của đối tượng được truyền, trả về
      - sử dụng 'copy constructor'
        - sao chép các giá trị gốc vào đối tượng mới

```
1   // Fig. 6.24: fig06_24.cpp
2   // Demonstrating that class objects can be assigned
3   // to each other using default memberwise assignment.
4   #include <iostream>
5
6   using std::cout;
7   using std::endl;
8
9   // class Date definition
10  class Date {
11
12  public:
13     Date( int = 1, int = 1, int = 1990 ); // default constructor
14     void print();
15
16  private:
17     int month;
18     int day;
19     int year;
20
21  }; // end class Date
22
```

fig06_24.cpp
(1 of 3)

```
23  // Date constructor with no range checking
24  Date::Date( int m, int d, int y )
25  {
26     month = m;
27     day = d;
28     year = y;
29
30  } // end Date constructor
31
32  // print Date in the format mm-dd-yyyy
33  void Date::print()
34  {
35     cout << month << '-' << day << '-' << year;
36
37  } // end function print
38
39  int main()
40  {
41     Date date1( 7, 4, 2002 );
42     Date date2;  // date2 defaults to 1/1/1990
43
```

fig06_24.cpp
(2 of 3)

```
44     cout << "date1 = ";
45     date1.print();
46     cout << "\ndate2 = ";
47     date2.print();
48
49     date2 = date1;   // default memberwise assignment
50
51     cout << "\n\nAfter default memberwise assignment, date2 = ";
52     date2.print();
53     cout << endl;
54
55     return 0;
56
57  } // end main
```

phép gán mặc định gán từng thành viên của **date1** cho thành viên tương ứng của **date2**.

fig06_24.cpp (3 of 3)

fig06_24.cpp output (1 of 1)

```
date1 = 7-4-2002
date2 = 1-1-1990

After default memberwise assignment, date2 = 7-4-2002
```

Ngôn ngữ lập trình C++

Chương 7 – Ra vào dữ liệu

Chương 7 : Ra vào dữ liệu

Đề mục

Chương 7 : Ra vào dữ liệu

Đề mục (tiếp)

7.1 Giới thiệu

- C++ I/O
  - Hướng đối tượng
    - sử dụng tham chiếu, chồng hàm, chồng toán tử
  - An toàn về các kiểu dữ liệu
    - nhạy cảm với kiểu dữ liệu
    - báo lỗi nếu kiểu không khớp
  - có thể dùng cho cả kiểu người dùng tự định nghĩa và các kiểu chuẩn
    - làm cho C++ có khả năng mở rộng

7.2 Dòng - Stream

- Stream – dòng:
  - chuỗi byte, kết thúc bởi ký hiệu *end_of_file*
  - Input: từ bàn phím, đĩa... vào bộ nhớ
  - Output: từ bộ nhớ ra màn hình, máy in...
  - file cũng được coi là một dòng
- Các dòng cổ điển
  - vào/ra **`char`** (1 byte)
  - các ký tự giới hạn bảng mã ASCII
- Các thư viện dòng chuẩn
  - Một số ngôn ngữ cần các bảng chữ cái đặc biệt
  - Unicode
    - kiểu ký tự **`wchar_t`**
  - Có thể thực hiện I/O với các ký tự Unicode

7.2.2 Các file header thư viện iostream

- thư viện **`iostream`**
  - có các header file với hàng trăm chức năng vào/ra
  - **`<iostream.h>`**
    - vào chuẩn – Standard input (**`cin`**)
    - ra chuẩn – Standard output (**`cout`**)
    - dòng báo lỗi không có bộ nhớ đệm – Unbuffered error (**`cerr`**)
    - dòng báo lỗi có dùng bộ nhớ đệm – Buffered error (**`clog`**)
  - **`<iomanip.h>`**
    - các stream manipulator (có tham số) để định dạng I/O
  - **`<fstream.h>`**
    - các thao tác xử lý file

7.2.3 Các đối tượng và các lớp I/O

- **<<** và **>>**
  - các toán tử chèn và tách dòng
- **cin**
  - đối tượng **istream**
  - nối với input chuẩn (thường là bàn phím)
  - **cin >> grade;**
    - trình biên dịch tự xác định kiểu của **grade**
    - gọi toán tử thích hợp (đã được định nghĩa chồng)
    - không cần thông tin thêm về kiểu dữ liệu

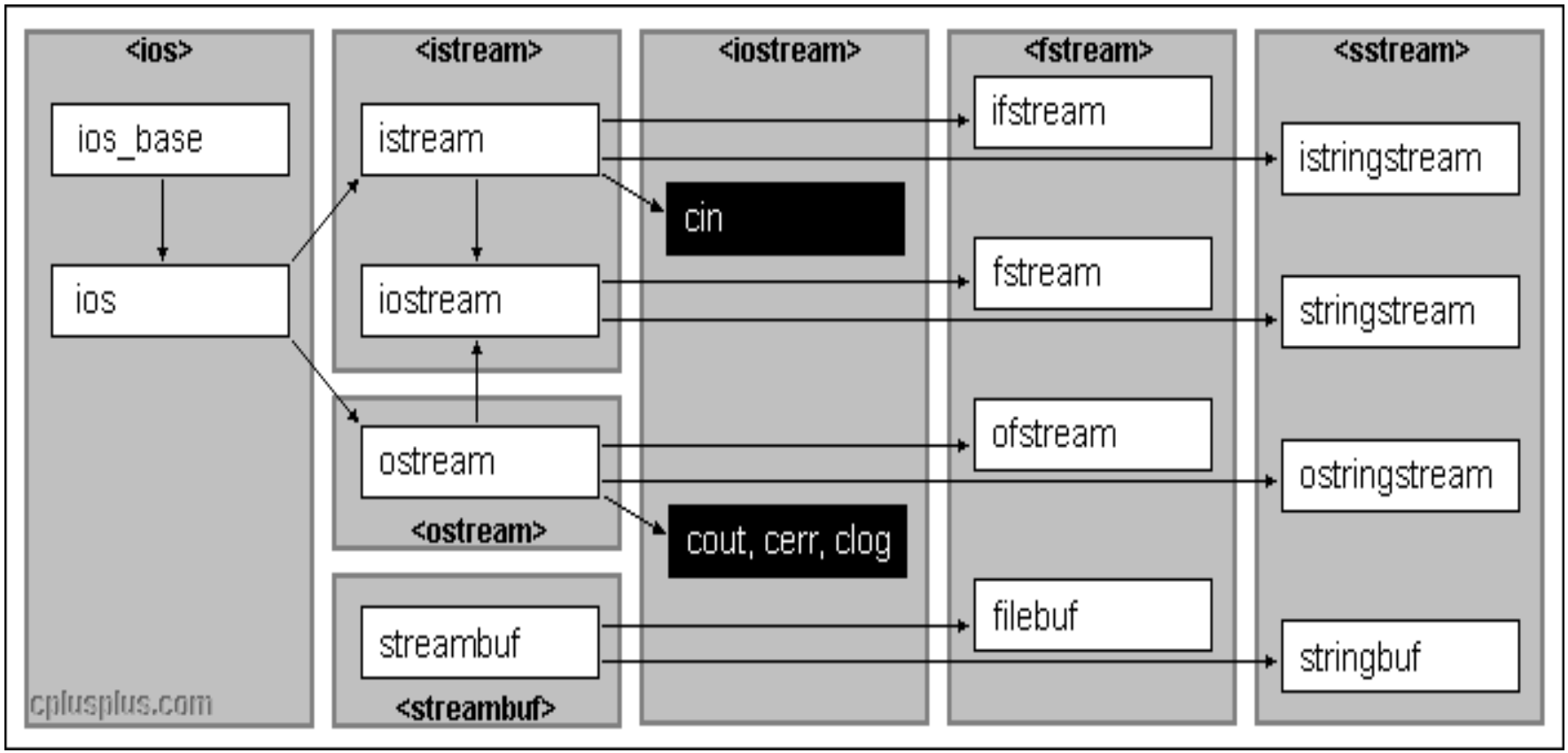
7.2.3 Các đối tượng và các lớp I/O

- **`cout`**
  - đối tượng **`ostream`**
  - nối với output chuẩn (thường là màn hình)
  - **`cin << grade;`**
    - cũng như với **`cin`**, không cần thêm thông tin về kiểu
- **`cerr`**, **`clog`**
  - các đối tượng **`ostream`**
  - nối với thiết bị báo lỗi chuẩn
  - **`cerr`** xuất ngay lập tức
  - **`clog`** sử dụng bộ nhớ đệm trước khi xuất
    - xuất khi bộ nhớ đệm đầy hoặc khi được xả (flushed)
    - ưu điểm hiệu năng (giải thích tại môn Hệ điều hành)

7.2.3 Các đối tượng và các lớp I/O

- C++ xử lý file tương tự
  - Các kiểu đối tượng dành cho xuất nhập **`char`**
    - **`ifstream`** (file input)
    - **`ofstream`** (file output)
    - **`fstream`** (file I/O)

7.2.3 Các đối tượng và các lớp I/O



7.3 Xuất theo dòng

- Output
  - sử dụng **`ostream`**
  - định dạng và không định dạng dữ liệu xuất
  - dành cho các kiểu dữ liệu chuẩn (**<<**)
    - các ký tự (hàm **`put`**)
    - các kiểu số nguyên (thập phân, bát phân, cơ số 16)
    - các số chấm động
      - quy định độ chính xác, vị trí dấu chấm, ký hiệu khoa học
  - dữ liệu được căn lề, chèn ký tự trống
  - điều khiển chữ hoa/chữ thường

7.3.1 Xuất các biến kiểu char *

- C++ tự động xác định kiểu dữ liệu
  - in giá trị của một **`char *`**
    - địa chỉ bộ nhớ của ký tự đầu tiên
- Rắc rối
  - **toán tử** << được định nghĩa chồng để in xâu kết thúc bằng null
  - cách giải quyết: đổi thành **`void *`**
    - sử dụng khi in giá trị của một con trỏ
    - in dưới dạng một số cơ số 16

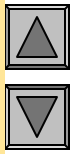


fig12_03.cpp (1 of 1)

fig12_03.cpp output (1 of 1)

```
1   // Fig. 12.3: fig12_03.cpp
2   // Printing the address stored in a char * variable.
3   #include <iostream>
4
5   using std::cout;
6   using std::endl;
7
8   int main()
9   {
10     char *word = "test";
11
12     // display value of char *, then display value of char *
13     // static_cast to void *
14     cout << "Value of word is: " << word << endl
15          << "Value of static_cast< void * >( word ) is: "
16          << static_cast< void * >( word ) << endl;
17
18     return 0;
19
20  } // end main
```

Để in giá trị của con trỏ, ta phải đổi sang kiểu **`void *`**. Nếu không, chương trình sẽ in xâu ký tự.

```
Value of word is: test
Value of static_cast< void *>( word ) is: 0046C070
```

7.3.2 Xuất ký tự bằng hàm thành viên put

- hàm **put**
  - in các ký tự
    - **cout.put('A');**
  - Có thể gọi liền
    - **cout.put('A').put('\n');**
    - Toán tử dấu chấm (.) được tính từ trái sang phải
  - Có thể sử dụng giá trị bằng số (mã ASCII)
    - **cout.put(65);**
    - in ký tự **'A'**

7.4 Nhập theo dòng

- dữ liệu vào có định dạng và không định dạng
  - **`istream`**
- toán tử **`>>`**
  - Thường bỏ qua các ký tự trắng (blank, tab, newline)
    - Có thể thay đổi
  - Trả về **`0`** khi gặp EOF
    - nếu không, trả về tham chiếu tới **`istream`**
    - **`cin >> grade`**
  - các bit trạng thái được bật nếu xảy ra lỗi
    - chi tiết sẽ nói đến sau

7.4.1 Các hàm thành viên get và getline

- hàm **`get`**
  - **`cin.get()`**
  - trả về một ký tự từ dòng (kể cả ký tự trắng)
    - trả về **`EOF`** nếu gặp end-of-file
- End-of-file
  - đánh dấu kết thúc dữ liệu vào
    - *ctrl-z* tại DOS/Windows
    - *ctrl-d* tại UNIX và Mac
  - **`cin.eof()`**
    - trả về **`1`** (**`true`**) nếu đã gặp EOF

fig12_04.cpp (1 of 2)

```
1   // Fig. 12.4: fig12_04.cpp
2   // Using member functions get, put and eof.
3   #include <iostream>
4
5   using std::cout;
6   using std::cin;
7   using std::endl;
8
9   int main()
10  {
11     int character; // use int, because char cannot represent EOF
12
13     // prompt user to enter line of text
14     cout << "Before input, cin.eof() is " << cin.eof() << endl
15          << "Enter a sentence followed by end-of-file:" << endl;
16
17     // use get to read each character; use put to display it
18     while ( ( character = cin.get() ) != EOF )
19        cout.put( character );
20
21     // display end-of-file character
22     cout << "\nEOF in this system is: " << character << endl;
23     cout << "After input, cin.eof() is " << cin.eof() << endl;
24
25     return 0;
```

Hàm **get** (không có đối số) trả về đúng một ký tự nhập vào, trừ khi gặp **EOF**.

```
26
27  } // end main
```

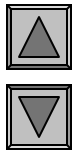


fig12_04.cpp (2 of 2)

fig12_04.cpp output (1 of 1)

```
Before input, cin.eof() is 0
Enter a sentence followed by end-of-file:
Testing the get and put member functions
Testing the get and put member functions
^Z

EOF in this system is: -1
After input cin.eof() is 1
```

7.4.1 Các hàm thành viên get và getline

- **`get(charRef)`**
  - đối số là tham chiếu ký tự
  - đọc một ký tự, lưu vào **`charRef`**
    - trả về tham chiếu tới **`istream`**
    - nếu hết file, trả về **`-1`**
- **`get(charArray, size, delimiter)`**
  - đọc cho đến khi được **`size-1`** ký tự, hoặc đến khi gặp ký tự phân cách
    - phân cách mặc định **`'\n'`**
    - ký tự phân cách được để lại dòng nhập
      - có thể loại bỏ bằng **`cin.get()`** hoặc **`cin.ignore()`**
  - tự động thêm null vào cuối để kết thúc mảng

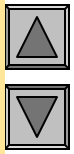


fig12_05.cpp
(1 of 2)

```
1   // Fig. 12.5: fig12_05.cpp
2   // Contrasting input of a string via cin and cin.get.
3   #include <iostream>
4
5   using std::cout;
6   using std::cin;
7   using std::endl;
8
9   int main()
10  {
11     // create two char arrays, each with 80 elements
12     const int SIZE = 80;
13     char buffer1[ SIZE ];
14     char buffer2[ SIZE ];
15
16     // use cin to input characters into buffer1
17     cout << "Enter a sentence:" << endl;
18     cin >> buffer1;
19
20     // display buffer1 contents
21     cout << "\nThe string read with cin was:" << endl
22          << buffer1 << endl << endl;
23
24     // use cin.get to input characters into buffer2
25     cin.get( buffer2, SIZE );
```

cin sẽ chỉ đọc cho đến ký tự trắng đầu tiên.

Không chỉ ra ký tự phân cách, do đó sẽ sử dụng phân cách mặc định (**\n**).

```
26
27     // display buffer2 contents
28     cout << "The string read with cin.get was:" << endl
29          << buffer2 << endl;
30
31     return 0;
32
33  } // end main
```

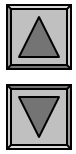


fig12_05.cpp (2 of 2)

fig12_05.cpp output (1 of 1)

```
Enter a sentence:
Contrasting string input with cin and cin.get


The string read with cin was:
Contrasting


The string read with cin.get was:
 string input with cin and cin.get
```

7.4.1 Các hàm thành viên get và getline

- **`getline(array, size, delimiter)`**
  - như phiên bản 3 tham số của **`get`**
  - đọc **`size-1`** ký tự, hoặc cho đến khi thấy ký tự phân cách
    - mặc định **`\n`**
  - loại bỏ ký tự phân cách khỏi dòng vào
  - đặt ký tự null vào cuối mảng

```
1   // Fig. 12.6: fig12_06.cpp
2   // Inputting characters using cin member function getline.
3   #include <iostream>
4
5   using std::cout;
6   using std::cin;
7   using std::endl;
8
9   int main()
10  {
11     const int SIZE = 80;
12     char buffer[ SIZE ]; // create array of 80 characters
13
14     // input characters in buffer via cin function getline
15     cout << "Enter a sentence:" << endl;
16     cin.getline( buffer, SIZE );
17
18     // display buffer contents
19     cout << "\nThe sentence entered is:" << endl << buffer << endl;
20
21     return 0;
22
23  } // end main
```

```
Enter a sentence:
Using the getline member function

The sentence entered is:
Using the getline member function
```

p

7.4.2 Các hàm thành viên peek, putback và ignore của istream

- **`ignore()`**
  - lấy các ký tự khỏi dòng (mặc định là 1 ký tự)
  - dừng khi gặp ký tự phân cách
    - phân cách mặc định là **`EOF`**
- **`putback()`**
  - đẩy ký tự vừa đọc được bằng **`get()`** trở lại dòng
- **`peek()`**
  - trả về ký tự tiếp theo trong dòng nhưng không lấy ra khỏi dòng

7.5 I/O không định dạng sử dụng read, write và gcount

- I/O không định dạng
  - **`read`** (hàm thành viên của **`istream`**)
    - đọc các byte thô vào mảng char
    - nếu đọc được không đủ số ký tự, đặt **`failbit`**
    - **`gcount()`** trả về số ký tự đã đọc được tại lần gọi gần nhất
  - **`write`** (hàm thành viên của **`ostream`**)
    - xuất các byte từ mảng char
      - dừng khi gặp ký tự null

```
char buffer[] = "HAPPY BIRTHDAY";
cout.write( buffer, 10 );
```

      - xuất 10 char đầu tiên

12_07.cpp
of 1)

```
1   // Fig. 12.7: fig12_07.cpp
2   // Unformatted I/O using read, gcount and write.
3   #include <iostream>
4
5   using std::cout;
6   using std::cin;
7   using std::endl;
8
9   int main()
10  {
11     const int SIZE = 80;
12     char buffer[ SIZE ]; // create array of 80 characters
13
14     // use function read to input characte
15     cout << "Enter a sentence:" << endl;
16     cin.read( buffer, 20 );
17
18     // use functions write and gcount to display buffer characters
19     cout << endl << "The sentence entered was:" << endl;
20     cout.write( buffer, cin.gcount() );
21     cout << endl;
22
23     return 0;
24
25  } // end main
```

đọc 20 ký tự từ dòng vào.
Hiển thị số ký tự đọc được,
sử dụng **write** và **gcount**.

```
Enter a sentence:
Using the read, write, and gcount member functions
The sentence entered was:
Using the read, writ
```

7.6 Giới thiệu về các Stream Manipulator

- Stream manipulator thực hiện việc định dạng
  - đổi hệ cơ số (hex, oct, dec, setbase)
  - độ rộng (ký tự) in ra dành cho dữ liệu xuất (setw)
  - đặt số chữ số sau dấu phảy (setprecision)
  - in/không in phần sau dấu phảy của số nguyên (showpoint/noshowpoint)
  - căn trái/phải/giữa (left/right/internal)
  - ký tự chèn vào các vị trí còn trống (setfill)
  - định dạng khoa học/dấu chấm động (scientific/fixed)
  - in các giá trị bool dạng chữ(true,false)/số (boolalpha/noboolalpha)
  - ...

7.7 Các trạng thái lỗi của dòng

- Kiểm tra trạng thái dòng bằng qua các bit trạng thái
  - **`eofbit`** được bật khi gặp EOF
    - hàm **`eof`** trả về **`true`** nếu **`eofbit`** được bật
    - **`cin.eof()`**
  - **`failbit`** được bật khi dòng xảy ra lỗi
    - dữ liệu không mất, lỗi có thể khôi phục được
    - hàm **`fail`** trả về **`true`** nếu bit được bật
  - **`badbit`** được bật khi mất dữ liệu
    - thường là không khôi phục được
    - hàm **`bad`**
  - **`goodbit`** bật khi **`badbit`**, **`failbit`** và **`eofbit`** tắt
    - hàm **`good`**

7.7 Các trạng thái lỗi của dòng

- Các hàm thành viên
  - **`rdstate()`**
    - trả về trạng thái lỗi của dòng
    - có thể dùng để kiểm tra **`goodbit`**, **`badbit`**, v.v...
    - sử dụng **`good()`**, **`bad()`** thì hơn
  - **`clear()`**
    - đối số mặc định là **`goodbit`**
    - đặt dòng trở về trạng thái tốt để có thể tiếp tục I/O
    - có thể truyền các giá trị khác
      - **`cin.clear( ios::failbit )`**
      - bật **`failbit`**

fig12_22.cpp
(1 of 2)

```
1  // Fig. 12.22: fig12_22.cpp
2  // Testing error states.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7  using std::cin;
8
9  int main()
10 {
11    int integerValue;
12
13    // display results of cin functions
14    cout << "Before a bad input operation:"
15         << "\ncin.rdstate(): " << cin.rdstate()
16         << "\n    cin.eof(): " << cin.eof()
17         << "\n   cin.fail(): " << cin.fail()
18         << "\n    cin.bad(): " << cin.bad()
19         << "\n   cin.good(): " << cin.good()
20         << "\n\nExpects an integer, but enter a character: ";
21
22    cin >> integerValue; // enter character value
23    cout << endl;
24
```

in trạng thái ban đầu, sử dụng các hàm thành viên

```
25     // display results of cin functions after bad input
26     cout << "After a bad input operation:"
27          << "\ncin.rdstate(): " << cin.rdstate()
28          << "\n    cin.eof(): " << cin.eof()
29          << "\n   cin.fail(): " << cin.fail()
30          << "\n    cin.bad(): " << cin.bad()
31          << "\n   cin.good(): " << cin.good() << endl << endl;
32
33     cin.clear(); // clear stream
34
35     // display results of cin functions after clearing cin
36     cout << "After cin.clear()"
37          << "\ncin.fail(): " << cin.fail()
38          << "\ncin.good(): " << cin.good() << endl;
39
40     return 0;
41
42  } // end main
```

Gọi hàm **clear**.

fig12_22.cpp
(2 of 2)

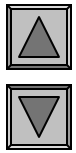


fig12_22.cpp
output (1 of 1)

```
Before a bad input operation:
cin.rdstate(): 0
    cin.eof(): 0
   cin.fail(): 0
    cin.bad(): 0
   cin.good(): 1

Expects an integer, but enter a character: A

After a bad input operation:
cin.rdstate(): 2
    cin.eof(): 0
   cin.fail(): 1
    cin.bad(): 0
   cin.good(): 0

After cin.clear()
cin.fail(): 0
cin.good(): 1
```

7.8 Đồng bộ một dòng ra và một dòng vào

- Rắc rối với output có bộ nhớ đệm
  - chương trình tương tác (hỏi người sử dụng, người sử dụng trả lời)
  - lời yêu cầu cần hiện ra trước khi nhập
    - output trong bộ nhớ đệm chỉ hiện ra khi bộ nhớ đệm đầy hoặc được xả (flushed)
- hàm thành viên **`tie`**
  - đồng bộ hóa các dòng
  - Output hiện ra trước các input tiếp theo
  - được thực hiện tự động với **`cin`** và **`cout`**, nhưng có thể viết
    - **`cin.tie( &cout )`**
  - cần thực hiện tường minh đối với các cặp I/O khác
  - để bỏ đồng bộ
    - **`inputStream.tie( 0 )`**

7.9 File và dòng

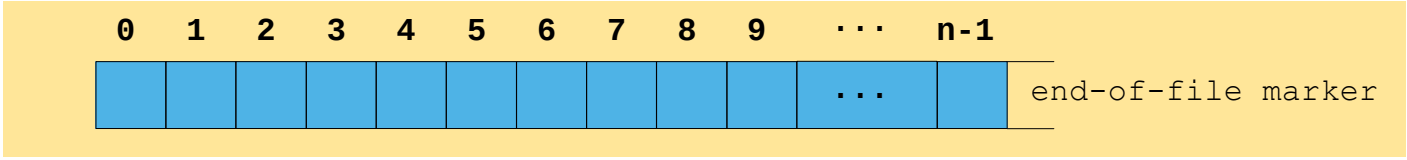
- Lưu trữ dữ liệu
  - Mảng, biến là dạng lưu trữ tạm thời
  - File là dạng lưu trữ bền vững
    - đĩa từ - magnetic disk, đĩa quang - optical disk, băng từ - tape
- trong chương này
  - tạo, cập nhật, xử lý file
  - truy nhập tuần tự (sequential access) và truy nhập ngẫu nhiên (random access)
  - xử lý có định dạng và xử lý thô (formatted and raw processing)

7.9 File và dòng

- Từ nhỏ nhất tới lớn nhất
  - Bit (binary digit)
  - Byte: 8 bits
    - Có thể lưu trữ 1 ký tự (**`char`**)
    - còn dùng để lưu Unicode dành cho bộ ký tự lớn hơn (**`wchar_t`**)
  - Trường - Field: nhóm ký tự có nghĩa
    - tên
  - Bản ghi - Record: nhóm các trường có liên quan
    - **`struct`** hoặc **`class`** trong C++
    - trong hệ thống trả lương (payroll system): tên, mã, địa chỉ, lương
    - mỗi trường liên quan đến cùng một nhân viên.
    - Khóa của bản ghi - Record key: trường dùng để xác định duy nhất bản ghi
  - File: nhóm các bản ghi có liên quan
    - danh sách lương cho cả công ty
  - Cơ sở dữ liệu - Database: nhóm các file có liên quan
    - danh sách lương, các tài khoản, …

7.9 File và dòng

- C++ coi file là một chuỗi byte - stream
  - Kết thúc bằng ký hiệu *end-of-file*



- Khi file mở
  - một đối tượng được tạo và kết nối với một dòng
  - tạo "đường liên lạc" từ đối tượng tới file
  - **`cin`**, **`cout`**, v.v... được tạo khi **`<iostream>`** được include
    - liên lạc giữa chương trình và file/thiết bị

7.10 File truy nhập tuần tự (sequential-access file)

- C++ không quy định cấu trúc file
  - Khái niệm "bản ghi" phải được cài đặt bởi lập trình viên
- Mở file
  - tạo đối tượng từ các lớp
    - **`ifstream`** (input only - chỉ đọc)
    - **`ofstream`** (output only - chỉ ghi)
    - **`fstream`** (I/O – file vừa đọc vừa ghi)
  - Constructor lấy tên file và kiểu mở file

```
ofstream outClientFile( "filename", fileOpenMode );
```

  - Hoặc, tạo object trước rồi gắn với một file sau

```
ofstream outClientFile;
outClientFile.open( "filename", fileOpenMode);
```


7.10 File truy nhập tuần tự

- Các kiểu mở file - File-open modes

Mode	Description
ios::app	Viết tiếp output vào cuối file.
ios::ate	Mở một file để ghi và di chuyển đến cuối file (thường dùng để nối dữ liệu vào file). Dữ liệu có thể được viết vào vị trí tùy ý trong file.
ios::in	Mở file để đọc
ios::out	Mở file để ghi.
ios::trunc	Loại bỏ nội dung file nếu nó tồn tại (mặc định đối với **ios::out**)
ios::binary	Mở file nhị phân (i.e., không phải file text) để đọc hoặc ghi.

  - theo mặc định, **`ofstream`** mở để ghi
    - **`ofstream outClientFile( "clients.dat", ios::out );`**
    - **`ofstream outClientFile( "clients.dat");`**

7.10 File truy nhập tuần tự

- Các phép toán
  - Overloaded **`operator!`**
    - **`!outClientFile`** hoặc **`!inClientfile`**
    - Trả về nonzero (true) nếu **`badbit`** hoặc **`failbit`** bật
      - mở file không tồn tại để đọc, không có quyền mở
  - Overloaded **`operator void*`**
    - chuyển đổi đối tượng dòng thành con trỏ
    - **`0`** khi **`failbit`** hoặc **`badbit`** được bật, nếu không: nonzero
      - **`failbit`** bật khi gặp EOF
    - **`while ( inClientFile >> myVariable )`**
      - lặp cho đến khi gặp EOF
  - Ghi/đọc file (như **`cout, cin`**)
    - **`outClientFile << myVariable`**
    - **`inClientFile >> myVariable`**
  - Đóng file
    - **`outClientFile.close()`**
    - đóng tự động khi destructor được gọi

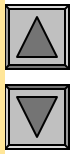


fig14_04.cpp
(1 of 2)

```
1   // Fig. 14.4: fig14_04.cpp
2   // Create a sequential file.
3   #include <iostream>
.....................
7   using std::ios;
8   using std::cerr;
9   using std::endl;
10
11  #include <fstream>
12
13  using std::ofstream;
14
15  #include <cstdlib>  // exit prototype
16
17  int main()
18  {
19     // ofstream constructor opens file
20     ofstream outClientFile( "clients.dat", ios::out );
21
22     // exit program if unable to create file
23     if ( !outClientFile ) {  // overloaded ! operator
24        cerr << "File could not be opened" << endl;
25        exit( 1 );
26
27     } // end if
```

Lưu ý các header file cần cho file I/O.

ofstream object được tạo và dùng để mở file **clients.dat"**. Nếu file chưa tồn tại, nó sẽ được tạo.

! operator dùng để kiểm tra xem có xảy ra lỗi khi mở file không.

```
28
29     cout << "Enter the account, name, and balance." << endl
30          << "Enter end-of-file to end input.\n? ";
31
32     int account;
33     char name[ 30 ];
34     double balance;
35
36     // read account, name and balance from cin, then place in file
37     while ( cin >> account >> name >> balance ) {
38        outClientFile << account << ' ' << name << ' ' << balance
39                      << endl;
40        cout << "? ";
41
42     } // end while
43
44     return 0;  // ofstream destructor closes file
45
46  }
```

fig14_04.cpp
(2 of 2)

cin được ngầm đổi thành 1 pointer.
Khi gặp EOF, nó trả về 0 và vòng lặp dừng.

Ghi dữ liệu ra file như ghi ra một dòng chuẩn.

File đóng khi destructor của object được gọi.
Có thể đóng một cách tường minh bằng cách gọi **close()**.

```
Enter the account, name, and balance.
Enter end-of-file to end input.
? 100 Jones 24.98
? 200 Doe 345.67
? 300 White 0.00
? 400 Stone -42.16
? 500 Rich 224.62
? ^Z
```

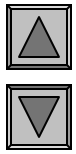


fig14_07.cpp
(1 of 3)

```
1  // Fig. 14.7: fig14_07.cpp
2  // Reading and printing a sequential file.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::ios;
8  using std::cerr;
9  using std::endl;
10 using std::left;
11 using std::right;
12 using std::fixed;
13 using std::showpoint;
14
15 #include <fstream>
16
17 using std::ifstream;
18
19 #include <iomanip>
20
21 using std::setw;
22 using std::setprecision;
23
24 #include <cstdlib> // exit prototype
25
26 void outputLine( int, const char * const, double );
27
```

```
28 int main()
29 {
30    // ifstream constructor opens the file
31    ifstream inClientFile( "clients.dat", ios::in );
32
33    // exit program if ifstream could not open file
34    if ( !inClientFile ) {
35       cerr << "File could not be opened" << endl;
36       exit( 1 );
37
38    } // end if
39
40    int account;
41    char name[ 30 ];
42    double balance;
43
44    cout << left << setw( 10 ) << "Account" << setw( 13 )
45         << "Name" << "Balance" << endl << fixed << showpoint;
46
47    // display each record in file
48    while ( inClientFile >> account >> name >> balance )
49       outputLine( account, name, balance );
50
51    return 0; // ifstream destructor closes the file
52
53 } // end main
```

mở file để đọc và kiểm tra.

Đọc từ file đến khi gặp EOF.

fig14_07.cpp
(2 of 3)

```
54
55 // display single record from file
56 void outputLine( int account, const char * const name,
57    double balance )
58 {
59    cout << left << setw( 10 ) << account << setw( 13 ) << name
60         << setw( 7 ) << setprecision( 2 ) << right << balance
61         << endl;
62
63 } // end function outputLine
```

fig14_07.cpp (3 of 3)

fig14_07.cpp output (1 of 1)

```
Account   Name         Balance
100       Jones          24.98
200       Doe           345.67
300       White           0.00
400       Stone         -42.16
500       Rich          224.62
```

7.11 Các hàm định vị cho file tuần tự

- con trỏ vị trí ghi số thứ tự của byte tiếp theo để đọc/ghi
- các hàm đặt lại vị trí của con trỏ:
  - **`seekg`** (đặt vị trí đọc cho lớp **`istream`**)
  - **`seekp`** (đặt vị trí ghi cho **`ostream`**)
  - **`seekg`** và **`seekp`** lấy các đối số là *offset* và *mốc*
    - Offset: số byte tương đối kể từ mốc
    - Mốc (**`ios::beg`** mặc định)
      - **`ios::beg`** - đầu file
      - **`ios::cur`** - vị trí hiện tại
      - **`ios::end`** - cuối file
- các hàm lấy vị trí hiện tại của con trỏ:
  - **`tellg`** và **`tellp`**

7.11 Các hàm định vị cho file tuần tự

- Ví dụ
  - **`fileObject.seekg(0)`**
    - đến đầu file (vị trí **`0`**), mặc định đối số thứ hai là **`ios::beg`**
  - **`fileObject.seekg(n)`**
    - đến byte thứ n kể từ đầu file
  - **`fileObject.seekg(n, ios::cur)`**
    - tiến n byte
  - **`fileObject.seekg(y, ios::end)`**
    - lùi y byte kể từ cuối file
  - **`fileObject.seekg(0, ios::cur)`**
    - đến cuối file
  - **`seekp`** tương tự
  - **`location = fileObject.tellg()`**
    - lấy vị trí đọc hiện tại của **`fileObject`**

7.11 Các hàm định vị cho file tuần tự

- Ví dụ:
  - chương trình quản lý tài khoản ngân hàng - Credit manager program
  - dữ liệu: file clients.dat
  - các chức năng:
    1. in danh sách các tài khoản rỗng (account with zero balance)
    2. in danh sách các tài khoản âm (account with credit)
    3. in danh sách các tài khoản dương (account with debit)
  - hoạt động của chương trình
    1. menu cho phép người dùng chọn một chức năng hoặc chọn dừng chương trình
    2. thực hiện chức năng đã chọn và in kết quả
    3. quay lại menu

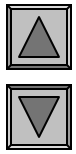


fig14_08.cpp
(1 of 6)

```
1  // Fig. 14.8: fig14_08.cpp
2  // Credit-inquiry program.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::ios;
8  using std::cerr;
9  using std::endl;
10 using std::fixed;
11 using std::showpoint;
12 using std::left;
13 using std::right;
14
15 #include <fstream>
16
17 using std::ifstream;
18
19 #include <iomanip>
20
21 using std::setw;
22 using std::setprecision;
23
24 #include <cstdlib>
25
```

```
26  enum RequestType { ZERO_BALANCE = 1, CREDIT_BALANCE,
27     DEBIT_BALANCE, END };
28  int getRequest();
29  bool shouldDisplay( int, double );
30  void outputLine( int, const char * const, double );
31
32  int main()
33  {
34     // ifstream constructor opens the file
35     ifstream inClientFile( "clients.dat", ios::in );
36
37     // exit program if ifstream could not open file
38     if ( !inClientFile ) {
39        cerr << "File could not be opened" << endl;
40        exit( 1 );
41
42     } // end if
43
44     int request;
45     int account;
46     char name[ 30 ];
47     double balance;
```

fig14_08.cpp
(2 of 6)

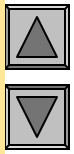
```
49     // get user's request (e.g., zero, credit or debit balance)
50     request = getRequest();
51
52     // process user's request
53     while ( request != END ) {
54
55        switch ( request ) {
56
57           case ZERO_BALANCE:
58              cout << "\nAccounts with zero balances:\n";
59              break;
60
61           case CREDIT_BALANCE:
62              cout << "\nAccounts with credit balances:\n";
63              break;
64
65           case DEBIT_BALANCE:
66              cout << "\nAccounts with debit balances:\n";
67              break;
68
69        } // end switch
70
```

fig14_08.cpp (3 of 6)

g14_08.cpp
of 6)

```
71          // read account, name and balance from file
72          inClientFile >> account >> name >> balance;
73
74          // display file contents (until eof)
75          while ( !inClientFile.eof() ) {
76
77             // display record
78             if ( shouldDisplay( request, balance ) )
79                outputLine( account, name, balance );
80
81             // read account, name and balance from file
82             inClientFile >> account >> name >> balance;
83
84          } // end inner while
85
86          inClientFile.clear();    // reset eof for next input
87          inClientFile.seekg( 0 ); // move to beginning of file
88          request = getRequest();  // get additional request from user
89
90       } // end outer while
91
92       cout << "End of run." << endl;
93
94       return 0; // ifstream destructor closes the file
95
96    } // end main
```

Dùng **clear** để bỏ cờ **eof**. Dùng **seekg** để đặt con trỏ định vị file về đầu file.



```
97
98  // obtain request from user
99  int getRequest()
100 {
101    int request;
102
103    // display request options
104    cout << "\nEnter request" << endl
105         << " 1 - List accounts with zero balances" << endl
106         << " 2 - List accounts with credit balances" << endl
107         << " 3 - List accounts with debit balances" << endl
108         << " 4 - End of run" << fixed << showpoint;
109
110    // input user request
111    do {
112       cout << "\n? ";
113       cin >> request;
114
115    } while ( request < ZERO_BALANCE && request > END );
116
117    return request;
118
119 } // end function getRequest
120
```

fig14_08.cpp
(5 of 6)

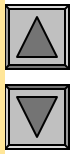


fig14_08.cpp
(6 of 6)

```
121 // determine whether to display given record
122 bool shouldDisplay( int type, double balance )
123 {
124    // determine whether to display credit balances
125    if ( type == CREDIT_BALANCE && balance < 0 )
126       return true;
127
128    // determine whether to display debit balances
129    if ( type == DEBIT_BALANCE && balance > 0 )
130       return true;
131
132    // determine whether to display zero balances
133    if ( type == ZERO_BALANCE && balance == 0 )
134       return true;
135
136    return false;
137
138 } // end function shouldDisplay
139
140 // display single record from file
141 void outputLine( int account, const char * const name,
142    double balance )
143 {
144    cout << left << setw( 10 ) << account << setw( 13 ) << name
145         << setw( 7 ) << setprecision( 2 ) << right << balance
146         << endl;
147
148 } // end function outputLine
```

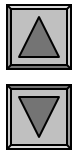


fig14_08.cpp
output (1 of 2)

```
Enter request
 1 - List accounts with zero balances
 2 - List accounts with credit balances
 3 - List accounts with debit balances
 4 - End of run
? 1

Accounts with zero balances:
300       White           0.00

Enter request
 1 - List accounts with zero balances
 2 - List accounts with credit balances
 3 - List accounts with debit balances
 4 - End of run
? 2
Accounts with credit balances:
400       Stone         -42.16
```

```
Enter request
 1 - List accounts with zero balances
 2 - List accounts with credit balances
 3 - List accounts with debit balances
 4 - End of run
? 3

Accounts with debit balances:
100       Jones          24.98
200       Doe           345.67
500       Rich          224.62

Enter request
 1 - List accounts with zero balances
 2 - List accounts with credit balances
 3 - List accounts with debit balances
 4 - End of run
? 4
End of run.
```

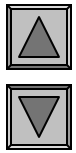


fig14_08.cpp
output (2 of 2)

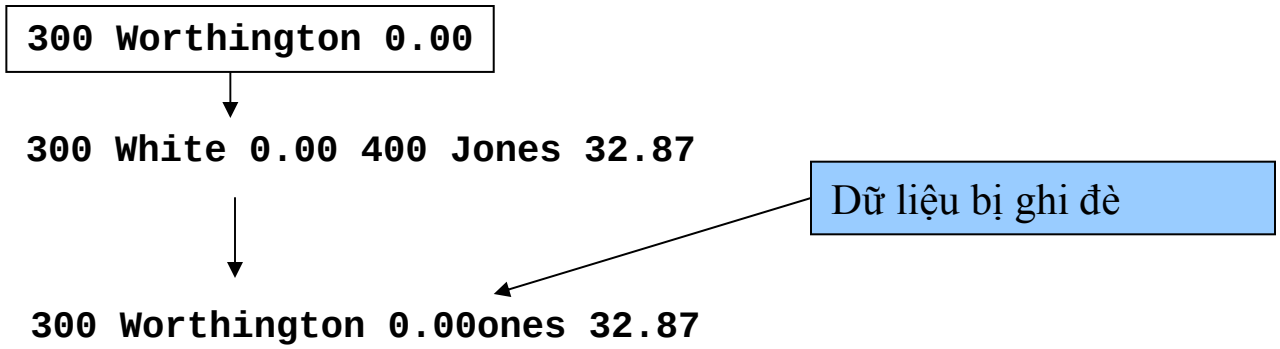
```
Enter request
 1 - List accounts with zero balances
 2 - List accounts with credit balances
 3 - List accounts with debit balances
 4 - End of run
? 3

Accounts with debit balances:
100       Jones          24.98
200       Doe           345.67
500       Rich          224.62

Enter request
 1 - List accounts with zero balances
 2 - List accounts with credit balances
 3 - List accounts with debit balances
 4 - End of run
? 4
End of run.
```

7.12 Các rắc rối khi cập nhật file truy nhập tuần tự

- cập nhật các file truy nhập tuần tự
  - Rủi ro: ghi đè các dữ liệu khác
  - Ví dụ: đổi tên từ "White" thành "Worthington"
    - Dữ liệu cũ

```
300 White 0.00 400 Jones 32.87
```

    - Chèn dữ liệu mới



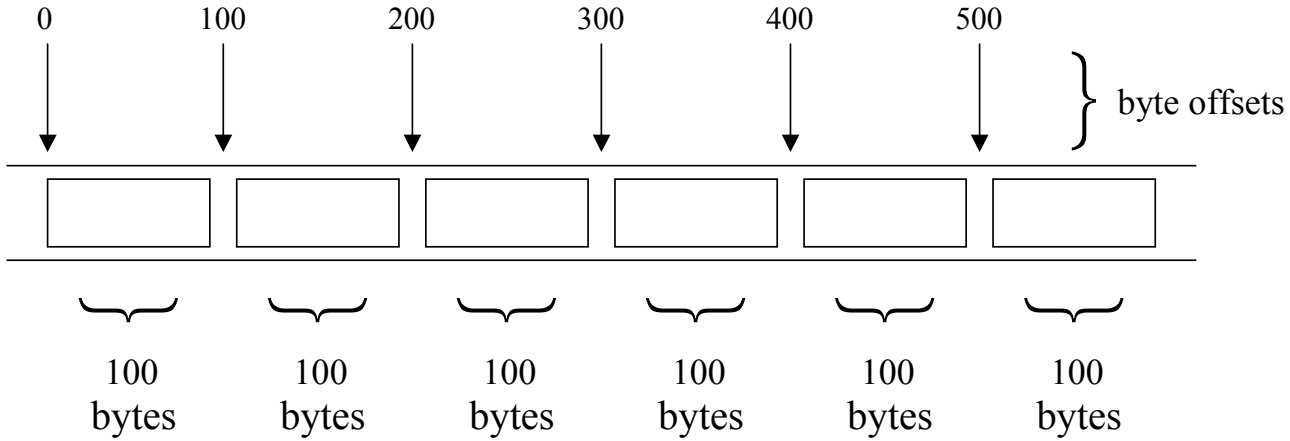
  - Định dạng bị rối loạn
  - Vấn đề có thể tránh được, nhưng biện pháp không hay.

7.13 Random-Access Files (các file truy nhập ngẫu nhiên)

- Truy nhập tức thời - Instant access
  - muốn định vị bản ghi một cách nhanh chóng
    - các hệ thống đặt vé máy bay (airline reservations), máy rút tiền tự động (ATM)
  - các file tuần tự phải duyệt qua từng bản ghi một
- Giải pháp: các file truy nhập ngẫu nhiên
  - khả năng truy nhập tức thời
  - chèn bản ghi mà không phá các dữ liệu khác
  - cập nhật/xóa một phần tử dữ liệu mà không làm thay đổi các dữ liệu khác

7.13 File truy nhập ngẫu nhiên (random-access file)

- C++ không quy định quy cách file
  - lập trình viên phải tự tạo quy cách cho các file truy nhập ngẫu nhiên
  - cách đơn giản nhất: các bản ghi độ dài cố định
    - tính toán được vị trí trong file từ kích thước bản ghi và khóa



7.13.1 Dữ liệu thô và dữ liệu định dạng

- **Ví dụ: `"1234567"` (`char *`) và `1234567` (`int`)**
  - định dạng: `char *` cần 8 byte (1 byte cho mỗi ký tự + null)
  - thô: `int` lấy một số cố định byte (có thể là 4)
    - 123 có cùng kích thước theo byte với 1234567
- các phép toán `<<` và `>>` dành cho dữ liệu định dạng
  - `outFile << number`
    - ghi `number` (`int`) dưới dạng `char *`
    - số lượng byte không cố định
- hàm `write()` và `read()` dành cho dữ liệu thô
  - `outFile.write( const char *, size );`
    - ghi ra các byte dạng thô
    - lấy tham số là con trỏ tới địa chỉ bộ nhớ, số byte cần ghi
      - sao chép dữ liệu trực tiếp từ bộ nhớ sang file
      - Không đổi thành `char *`

7.13.2 Ghi file truy nhập nhẫu nhiên

- Ví dụ hàm **`write()`**

```
outFile.write( reinterpret_cast<const char *>(&number),
               sizeof( number ) );
```

  - **`&number`** là **`int *`**
    - đổi thành **`const char *`** bằng **`reinterpret_cast`**
  - **`sizeof(number)`**
    - kích thước của **`number`** (một số **`int`**) tính theo byte
  - tương tự đối với hàm **`read`** (more later)
  - Chú ý:
    - chỉ dùng **`write`**/**`read`** giữa các máy tương thích
  - mở file kiểu **`ios::binary`** để đọc/ghi thô
- thường dùng để ghi toàn bộ một **`struct`** hoặc một đối tượng ra file

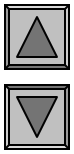
7.13.2 Ghi file truy nhập ngẫu nhiên

- Bài toán
  - chương trình quản lý tài khoản
  - Lưu trữ tối đa 100 bản ghi kích thước cố định
  - Bản ghi
    - Mã tài khoản - Account number (khóa)
    - Họ và tên - First and last name
    - Số tiền hiện có trong tài khoản - Balance
  - Các thao tác:
    - cập nhật, tạo mới, xóa, liệt kê tất cả các tài khoản ra một file
- Tiếp theo: chương trình tạo file chứa 100 bản ghi rỗng

ientData.h
of 2)

```
1   // Fig. 14.10: clientData.h
2   // Class ClientData definition used in Fig. 14.12–Fig. 14.15.
3   #ifndef CLIENTDATA_H
4   #define CLIENTDATA_H
5
6   #include <iostream>
7
8   using std::string;
9
10  class ClientData {
11
12  public:
13
14     // default ClientData constructor
15     ClientData( int = 0, string = "", string = "", double = 0.0 );
16
17     // accessor functions for accountNumber
18     void setAccountNumber( int );
19     int getAccountNumber() const;
20
21     // accessor functions for lastName
22     void setLastName( string );
23     string getLastName() const;
24
```

Class **ClientData** lưu thông tin về từng người. 100 đối tượng **ClientData** rỗng sẽ được ghi ra 1 file.



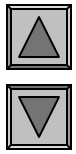
clientData.h
(2 of 2)

```
25    // accessor functions for firstName
26    void setFirstName( string );
27    string getFirstName() const;
28
29    // accessor functions for balance
30    void setBalance( double );
31    double getBalance() const;
32
33 private:
34    int accountNumber;
35    char lastName[ 15 ];
36    char firstName[ 10 ];
37    double balance;
38
39 }; // end class ClientData
40
41 #endif
```

Đặt giới hạn kích thước tên và họ. **`accountNumber`** (một số **`int`**) và **`balance`** (**`double`**) đã có kích thước cố định.

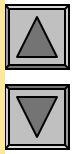

ClientData.cpp
(1 of 4)

```
1  // Fig. 14.11: ClientData.cpp
2  // Class ClientData stores customer's credit information.
3  #include <iostream>
4
5  using std::string;
6
7  #include <cstring>
8  #include "clientData.h"
9
10 // default ClientData constructor
11 ClientData::ClientData( int accountNumberValue,
12    string lastNameValue, string firstNameValue,
13    double balanceValue )
14 {
15    setAccountNumber( accountNumberValue );
16    setLastName( lastNameValue );
17    setFirstName( firstNameValue );
18    setBalance( balanceValue );
19
20 } // end ClientData constructor
21
22 // get account-number value
23 int ClientData::getAccountNumber() const
24 {
25    return accountNumber;
26
27 } // end function getAccountNumber
```



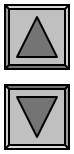
ClientData.cpp
(2 of 4)

```
28
29 // set account-number value
30 void ClientData::setAccountNumber( int accountNumberValue )
31 {
32    accountNumber = accountNumberValue;
33
34 } // end function setAccountNumber
35
36 // get last-name value
37 string ClientData::getLastName() const
38 {
39    return lastName;
40
41 } // end function getLastName
42
43 // set last-name value
44 void ClientData::setLastName( string lastNameString )
45 {
46    // copy at most 15 characters from string to lastName
47    const char *lastNameValue = lastNameString.data();
48    int length = strlen( lastNameValue );
49    length = ( length < 15 ? length : 14 );
50    strncpy( lastName, lastNameValue, length );
51
52    // append null character to lastName
53    lastName[ length ] = '\0';
```



ClientData.cpp
(3 of 4)

```
54
55 } // end function setLastName
56
57 // get first-name value
58 string ClientData::getFirstName() const
59 {
60    return firstName;
61
62 } // end function getFirstName
63
64 // set first-name value
65 void ClientData::setFirstName( string firstNameString )
66 {
67    // copy at most 10 characters from string to firstName
68    const char *firstNameValue = firstNameString.data();
69    int length = strlen( firstNameValue );
70    length = ( length < 10 ? length : 9 );
71    strncpy( firstName, firstNameValue, length );
72
73    // append new-line character to firstName
74    firstName[ length ] = '\0';
75
76 } // end function setFirstName
77
```



ClientData.cpp
(4 of 4)

```
78  // get balance value
79  double ClientData::getBalance() const
80  {
81     return balance;
82
83  } // end function getBalance
84
85  // set balance value
86  void ClientData::setBalance( double balanceValue )
87  {
88     balance = balanceValue;
89
90  } // end function setBalance
```

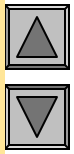


fig14_12.cpp
(1 of 2)

```
1  // Fig. 14.12: fig14_12.cpp
2  // Creating a randomly accessed file.
3  #include <iostream>
4
5  using std::cerr;
6  using std::endl;
7  using std::ios;
8
9  #include <fstream>
10
11 using std::ofstream;
12
13 #include <cstdlib>
14 #include "clientData.h"  // ClientData class definition
15
16 int main()
17 {
18    ofstream outCredit( "credit.dat", ios::binary );
19
20    // exit program if ofstream could not open file
21    if ( !outCredit ) {
22       cerr << "File could not be opened." << endl;
23       exit( 1 );
24
25    } // end if
```

Mở 1 file để ghi thô,
sử dụng một đối tượng **`ofstream`**
và **`ios::binary`**.

14_12.cpp
of 2)

```
26
27    // create ClientData with no information
28    ClientData blankClient;
29
30    // output 100 blank records to file
31    for ( int i = 0; i < 100; i++ )
32       outCredit.write(
33          reinterpret_cast< const char * >( &blankClient ),
34          sizeof( ClientData ) );
35
36    return 0;
37
38 } // end main
```

Tạo một đối tượng rỗng. Dùng **`write`** để ghi dữ liệu thô ra 1 file (truyền tham số là địa chỉ đối tượng và kích thước đối tượng).

7.13.3 Ghi dữ liệu vào vị trí tùy ý trong file truy nhập ngẫu nhiên

- Dùng **seekp** để ghi vào vị trí chính xác trong file
  - Bản ghi đầu tiên bắt đầu từ đâu?
    - Byte 0
  - Bản ghi thứ hai?
    - Byte 0 + sizeof(object)
  - Bản ghi bất kỳ?
    - (Recordnum - 1) * sizeof(object)

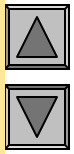


fig14_13.cpp
(1 of 3)

```
1   // Fig. 14.13: fig14_13.cpp
2   // Writing to a random access file.
3   #include <iostream>
.....
19  #include <cstdlib>
20  #include "clientData.h"  // ClientData class definition
21
22  int main()
23  {
24     int accountNumber;
25     char lastName[ 15 ];
26     char firstName[ 10 ];
27     double balance;
28
29     ofstream outCredit( "credit.dat", ios::binary );
30
31     // exit program if ofstream cannot open file
32     if ( !outCredit ) {
33        cerr << "File could not be opened." << endl;
34        exit( 1 );
35
36     } // end if
```

Mở file để ghi thô (binary writing).

```
38     cout << "Enter account number "
39          << "(1 to 100, 0 to end input)\n? ";
40
41     // require user to specify account number
42     ClientData client;
43     cin >> accountNumber;
44     client.setAccountNumber( accountNumber );
45
46     // user enters information, which is copied into file
47     while ( client.getAccountNumber() > 0 &&
48        client.getAccountNumber() <= 100 ) {
49
50        // user enters last name, first name and balance
51        cout << "Enter lastname, firstname, balance\n? ";
52        cin >> setw( 15 ) >> lastName;
53        cin >> setw( 10 ) >> firstName;
54        cin >> balance;
55
56        // set record lastName, firstName and balance values
57        client.setLastName( lastName );
58        client.setFirstName( firstName );
59        client.setBalance( balance );
```

Nhập account number, ghi vào đối tượng. Nó chưa được viết ra file.

fig14_13.cpp
(2 of 3)

Đặt **outCredit** vào vị trí thích hợp trong file (dựa vào account number).

fig14_13.cpp (3 of 3)

```
60
61       // seek position in file of user-specified record
62       outCredit.seekp( ( client.getAccountNumber() - 1 ) *
63          sizeof( ClientData ) );
64
65       // write user-specified information in file
66       outCredit.write(
67          reinterpret_cast< const char * >( &client ),
68          sizeof( ClientData ) );
69
70       // enable user to specify another account number
71       cout << "Enter account number\n? ";
72       cin >> accountNumber;
73       client.setAccountNumber( accountNumber );
74
75    } // end while
76
77    return 0;
78
79 } // end main
```

Ghi đối tượng **ClientData** vào file tại vị trí đó.

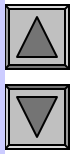


fig14_13.cpp
output (1 of 1)

```
Enter account number (1 to 100, 0 to end input)
? 37
Enter lastname, firstname, balance
? Barker Doug 0.00
Enter account number
? 29
Enter lastname, firstname, balance
? Brown Nancy -24.54
Enter account number
? 96
Enter lastname, firstname, balance
? Stone Sam 34.98
Enter account number
? 88
Enter lastname, firstname, balance
? Smith Dave 258.34
Enter account number
? 33
Enter lastname, firstname, balance
? Dunn Stacey 314.33
Enter account number
? 0
```

Lưu ý các account có thể được tạo theo thứ tự tùy ý.

7.13.4 Đọc tuần tự dữ liệu từ file truy nhập ngẫu nhiên

- **`read`** - tương tự **`write`**
  - Đọc các byte thô từ file vào bộ nhớ
  - **`inFile.read( reinterpret_cast<char *>( &number ), sizeof( int ) );`**
    - **`&number`**: địa chỉ để lưu dữ liệu
    - **`sizeof(int)`**: số byte cần đọc
  - Không dùng **`inFile >> number`** cho dữ liệu thô - nhị phân
    - **`>>`** nhận **`char *`**
- Chương trình tiếp theo
  - lấy dữ liệu từ một file random-access
  - duyệt tuần tự qua từng bản ghi
    - If no data (accountNumber == 0) then skip

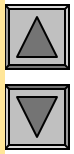


fig14_14.cpp
(1 of 2)

```
1   // Fig. 14.14: fig14_14.cpp
2   // Reading a random access file.
......
25  #include "clientData.h"  // ClientData class definition
26
27  void outputLine( ostream&, const ClientData & );
28
29  int main()
30  {
31     ifstream inCredit( "credit.dat", ios::in );
32
33     // exit program if ifstream cannot open file
34     if ( !inCredit ) {
35        cerr << "File could not be opened." << endl;
36        exit( 1 );
37
38     } // end if
39
40     cout << left << setw( 10 ) << "Account" << setw( 16 )
41          << "Last Name" << setw( 11 ) << "First Name" << left
42          << setw( 10 ) << right << "Balance" << endl;
43
44     ClientData client; // create record
45
46     // read first record from file
47     inCredit.read( reinterpret_cast< char * >( &client ),
48        sizeof( ClientData ) );
```

Đọc **`sizeof(ClientData)`** byte và ghi vào đối tượng **`client`**. Đây có thể là một bản ghi rỗng.

```
50    // read all records from file
51    while ( inCredit && !inCredit.eof() ) {
52
53       // display record
54       if ( client.getAccountNumber() != 0 )
55          outputLine( cout, client );
56
57       // read next from file
58       inCredit.read( reinterpret_cast< char * >( &client ),
59          sizeof( ClientData ) );
60
61    } // end while
62
63    return 0;
64
65 } // end main
66
67 // display single record
68 void outputLine( ostream &output, const ClientData &record )
69 {
70    output << left << setw( 10 ) << record.getAccountNumber()
71           << setw( 16 ) << record.getLastName().data()
72           << setw( 11 ) << record.getFirstName().data()
73           << setw( 10 ) << setprecision( 2 ) << right << fixed
74           << showpoint << record.getBalance() << endl;
75
76 } // end outputLine
```

Vòng lặp dừng khi có lỗi đọc (**`inCredit == 0`**) hoặc gặp EOF (**`inCredit.eof() == 1`**)

Output non-empty accounts. Lưu ý **`outputLine`** lấy 1 tham số kiểu **`ostream`**. Ta có thể dễ dàng output ra một file khác (mở bằngmột **`ofstream`** object, là dẫn xuất của **`ostream`**).

```
Account   Last Name       First Name    Balance
29        Brown           Nancy          -24.54
33        Dunn            Stacey         314.33
37        Barker          Doug             0.00
88        Smith           Dave           258.34
96        Stone           Sam             34.98
```

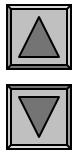


fig14_14.cpp output (1 of 1)

7.14 Ví dụ: Chương trình xử lý giao dịch

- Bài toán:
  - chương trình quản lý các tài khoản ngân hàng, cho phép truy nhập trực tiếp từng tài khoản
  - dữ liệu: file truy nhập ngẫu nhiên **`credit.dat`**
- Các chức năng cho người dùng (các lựa chọn cho menu)
  - Lựa chọn 1: ghi các account ra file **`print.txt`**

```
Account   Last Name       First Name    Balance
29        Brown           Nancy          -24.54
33        Dunn            Stacey         314.33
37        Barker          Doug             0.00
88        Smith           Dave           258.34
96        Stone           Sam             34.98
```

  - Lựa chọn 2: cập nhật bản ghi

```
Enter account to update (1 - 100): 37
37        Barker          Doug             0.00

Enter charge (+) or payment (-): +87.99
37        Barker          Doug            87.99
```

7.14 Ví dụ: Chương trình xử lý giao dịch

- Các chức năng (tiếp)
  - Lựa chọn 3: thêm bản ghi

```
Enter new account number (1 - 100): 22
Enter lastname, firstname, balance
? Johnston Sarah 247.45
```

  - Lựa chọn 4: xóa bản ghi

```
Enter account to delete (1 - 100): 29
Account #29 deleted.
```

- Mở file vừa đọc vừa ghi
  - Dùng **`fstream`** object
  - nhiều file-open mode cùng lúc

```
fstream inOutCredit( "credit.dat", ios::in | ios::out );
```

```
1   // Fig. 14.15: fig14_15.cpp
2   // This program reads a random access file sequentially, updates
3   // data previously written to the file, creates data to be placed
4   // in the file, and deletes data previously in the file.
5   #include <iostream>
6
7   using std::cout;
...
15  using std::showpoint;
16
17  #include <fstream>
18
19  using std::ofstream;
20  using std::ostream;
21  using std::fstream;
22
23  #include <iomanip>
24
25  using std::setw;
26  using std::setprecision;
27
28  #include <cstdlib>       // exit prototype
29  #include "clientData.h"  // ClientData class definition
```

fig14_15.cpp (1 of 14)

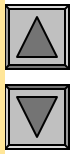


fig14_15.cpp
(2 of 14)

```
30
31 int enterChoice();
32 void printRecord( fstream& );
33 void updateRecord( fstream& );
34 void newRecord( fstream& );
35 void deleteRecord( fstream& );
36 void outputLine( ostream&, const ClientData & );
37 int getAccount( const char * const );
38
39 enum Choices { PRINT = 1, UPDATE, NEW, DELETE, END };
40
41 int main()
42 {
43    // open file for reading and writing
44    fstream inOutCredit( "credit.dat", ios::in | ios::out );
45
46    // exit program if fstream cannot open file
47    if ( !inOutCredit ) {
48       cerr << "File could not be opened." << endl;
49       exit ( 1 );
50
51    } // end if
52
```

Mở file để đọc và ghi (cần **`fstream`** object).

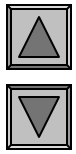


fig14_15.cpp
4)

```
53    int choice;
54
55    // enable user to specify action
56    while ( ( choice = enterChoice() ) != END ) {
57
58       switch ( choice ) {
59
60          // create text file from record file
61          case PRINT:
62             printRecord( inOutCredit );
63             break;
64
65          // update record
66          case UPDATE:
67             updateRecord( inOutCredit );
68             break;
69
70          // create record
71          case NEW:
72             newRecord( inOutCredit );
73             break;
74
75          // delete existing record
76          case DELETE:
77             deleteRecord( inOutCredit );
78             break;
79
```

Hiện menu và trả về lựa chọn người dùng.

```
80          // display error if user does not select valid choice
81          default:
82             cerr << "Incorrect choice" << endl;
83             break;
84
85       } // end switch
86
87       inOutCredit.clear(); // reset end-of-file indicator
88
89    } // end while
90
91    return 0;
92
93 } // end main
94
95 // enable user to input menu choice
96 int enterChoice()
97 {
98    // display available options
99    cout << "\nEnter your choice" << endl
100         << "1 - store a formatted text file of accounts" << endl
101         << "    called \"print.txt\" for printing" << endl
102         << "2 - update an account" << endl
103         << "3 - add a new account" << endl
104         << "4 - delete an account" << endl
105         << "5 - end program\n? ";
```

fig14_15.cpp (5 of 14)

```
106
107    int menuChoice;
108    cin >> menuChoice; // receive choice from user
109
110    return menuChoice;
111
112 } // end function enterChoice
113
114 // create formatted text file for printing
115 void printRecord( fstream &readFromFile )
116 {
117    // create text file
118    ofstream outPrintFile( "print.txt", ios::out );
119
120    // exit program if ofstream cannot create file
121    if ( !outPrintFile ) {
122       cerr << "File could not be created." << endl;
123       exit( 1 );
124
125    } // end if
126
127    outPrintFile << left << setw( 10 ) << "Account" << setw( 16 )
128        << "Last Name" << setw( 11 ) << "First Name" << right
129        << setw( 10 ) << "Balance" << endl;
130
```

In ra **print.txt**. Trước tiên, in header của bảng.

fig14_15.cpp (6 of 14)

```
131    // set file-position pointer to beginning of record file
132    readFromFile.seekg( 0 );
133
134    // read first record from record file
135    ClientData client;
136    readFromFile.read( reinterpret_cast< char * >( &client ),
137       sizeof( ClientData ) );
138
139    // copy all records from record file into text file
140    while ( !readFromFile.eof() ) {
141
142       // write single record to text file
143       if ( client.getAccountNumber() != 0 )
144          outputLine( outPrintFile, client );
145
146       // read next record from record file
147       readFromFile.read( reinterpret_cast< char * >( &client ),
148          sizeof( ClientData ) );
149
150    } // end while
151
152 } // end function printRecord
153
```

fig14_15.cpp

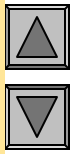
Đến đầu file, đọc dữ liệu về tài khoản, và in bản ghi nếu nó không rỗng.

Lưu ý **outputLine** lấy đối số là đối tượng **ostream** object (lớp cơ sở của **ofstream**). Nó có thể ghi ra file (như trong trường hợp này) hoặc **cout**.

```
154 // update balance in record
155 void updateRecord( fstream &updateFile )
156 {
157    // obtain number of account to update
158    int accountNumber = getAccount( "Enter account to update" );
159
160    // move file-position pointer to correct record in file
161    updateFile.seekg(
162       ( accountNumber - 1 ) * sizeof( ClientData ) );
163
164    // read first record from file
165    ClientData client;
166    updateFile.read( reinterpret_cast< char * >( &client ),
167       sizeof( ClientData ) );
168
169    // update record
170    if ( client.getAccountNumber() != 0 ) {
171       outputLine( cout, client );
172
173       // request user to specify transaction
174       cout << "\nEnter charge (+) or payment (-): ";
175       double transaction; // charge or payment
176       cin >> transaction;
```

fig14_15.cpp (8 of 14)

Đây là **`fstream`** (I/O) vì ta phải đọc balance cũ, cập nhật nó, và ghi balance mới.



```
177
178       // update record balance
179       double oldBalance = client.getBalance();
180       client.setBalance( oldBalance + transaction );
181       outputLine( cout, client );
182
183       // move file-position pointer to correct record in file
184       updateFile.seekp(
185          ( accountNumber - 1 ) * sizeof( ClientData ) );
186
187       // write updated record over old record in file
188       updateFile.write(
189          reinterpret_cast< const char * >( &client ),
190          sizeof( ClientData ) );
191
192    } // end if
193
194    // display error if account does not exist
195    else
196       cerr << "Account #" << accountNumber
197          << " has no information." << endl;
198
199 } // end function updateRecord
200
```

fig14_15.cpp
(9 of 14)

```
201 // create and insert record
202 void newRecord( fstream &insertInFile )
203 {
204    // obtain number of account to create
205    int accountNumber = getAccount( "Enter new account number" );
206
207    // move file-position pointer to correct record in file
208    insertInFile.seekg(
209       ( accountNumber - 1 ) * sizeof( ClientData ) );
210
211    // read record from file
212    ClientData client;
213    insertInFile.read( reinterpret_cast< char * >( &client ),
214       sizeof( ClientData ) );
215
216    // create record, if record does not previously exist
217    if ( client.getAccountNumber() == 0 ) {
218
219       char lastName[ 15 ];
220       char firstName[ 10 ];
221       double balance;
```

fig14_15.cpp (10 of 14)

Đây là **`fstream`** vì ta đọc thử để xem đã có sẵn một bản ghi rỗng hay chưa, nếu chưa, ta ghi một bản ghi mới.

```
222
223       // user enters last name, first name and balance
224       cout << "Enter lastname, firstname, balance\n? ";
225       cin >> setw( 15 ) >> lastName;
226       cin >> setw( 10 ) >> firstName;
227       cin >> balance;
228
229       // use values to populate account values
230       client.setLastName( lastName );
231       client.setFirstName( firstName );
232       client.setBalance( balance );
233       client.setAccountNumber( accountNumber );
234
235       // move file-position pointer to correct record in file
236       insertInFile.seekp( ( accountNumber - 1 ) *
237          sizeof( ClientData ) );
238
239       // insert record in file
240       insertInFile.write(
241          reinterpret_cast< const char * >( &client ),
242          sizeof( ClientData ) );
243
244    } // end if
245
```

fig14_15.cpp
(11 of 14)

```
246    // display error if account previously exists
247    else
248       cerr << "Account #" << accountNumber
249            << " already contains information." << endl;
250
251 } // end function newRecord
252
253 // delete an existing record
254 void deleteRecord( fstream &deleteFromFile )
255 {
256    // obtain number of account to delete
257    int accountNumber = getAccount( "Enter account to delete" );
258
259    // move file-position pointer to correct record in file
260    deleteFromFile.seekg(
261       ( accountNumber - 1 ) * sizeof( ClientData ) );
262
263    // read record from file
264    ClientData client;
265    deleteFromFile.read( reinterpret_cast< char * >( &client ),
266       sizeof( ClientData ) );
267
```

fig14_15.cpp (12 of 14)

là **fstream** vì ta đọc để kiểm tra xem account có tồn tại không. Nếu có, ta ghi dữ liệu rỗng (xóa nó). Nếu không, không cần xóa.

```
268    // delete record, if record exists in file
269    if ( client.getAccountNumber() != 0 ) {
270       ClientData blankClient;
271
272       // move file-position pointer to correct record in file
273       deleteFromFile.seekp( ( accountNumber - 1 ) *
274          sizeof( ClientData ) );
275
276       // replace existing record with blank record
277       deleteFromFile.write(
278          reinterpret_cast< const char * >( &blankClient ),
279          sizeof( ClientData ) );
280
281       cout << "Account #" << accountNumber << " deleted.\n";
282
283    } // end if
284
285    // display error if record does not exist
286    else
287       cerr << "Account #" << accountNumber << " is empty.\n";
288
289 } // end deleteRecord
290
```

fig14_15.cpp
(13 of 14)

```
291 // display single record
292 void outputLine( ostream &output, const ClientData &record )
293 {
294    output << left << setw( 10 ) << record.getAccountNumber()
295           << setw( 16 ) << record.getLastName().data()
296           << setw( 11 ) << record.getFirstName().data()
297           << setw( 10 ) << setprecision( 2 ) << right << fixed
298           << showpoint << record.getBalance() << endl;
299
300 } // end function outputLine
301
302 // obtain account-number value from user
303 int getAccount( const char * const prompt )
304 {
305    int accountNumber;
306
307    // obtain account-number value
308    do {
309       cout << prompt << " (1 - 100): ";
310       cin >> accountNumber;
311
312    } while ( accountNumber < 1 || accountNumber > 100 );
313
314    return accountNumber;
315
316 } // end function getAccount
```

fig14_15.cpp (14 of 14)

`outputLine` rất mềm dẻo, và có thể ghi ra **`ostream`** object bất kỳ (chẳng hạn 1 file hoặc **`cout`**).