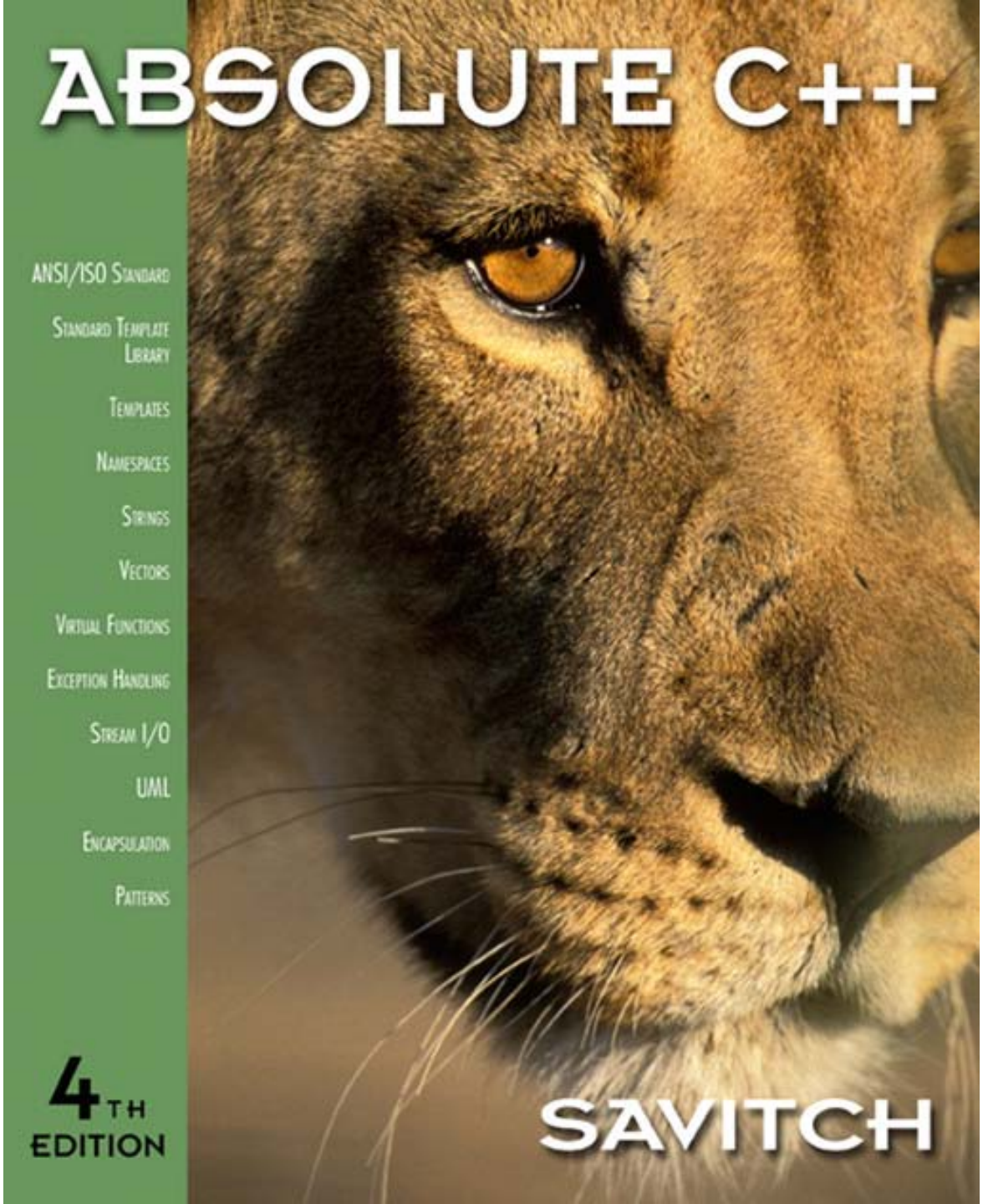
Bài 2: Luồng điều khiển

Giảng viên: Hoàng Thị Điệp
Khoa Công nghệ Thông tin – ĐH Công Nghệ



Chapter 2

Flow of Control

PEARSON Addison Wesley

Mục tiêu bài học

- Biểu thức logic
  - Lập biểu thức, Tính giá trị & Các luật ưu tiên
- Các cơ chế rẽ nhánh
  - if-else
  - switch
  - if-else lồng nhau
- Lặp
  - while, do-while, for
  - Các vòng lặp lồng nhau

Biểu thức logic:
Display 2.1 Các phép toán so sánh

- Các phép toán logic
  - Phép AND logic (&&)
  - Phép OR logic (||)

Display 2.1 Comparison Operators

MATH SYMBOL	ENGLISH	C++ NOTATION	C++ SAMPLE	MATH EQUIVALENT
=	Equal to	==	x + 7 == 2*y	x + 7 = 2y
≠	Not equal to	!=	ans != 'n'	*ans* ≠ *'n'*
<	Less than	<	count < m + 3	*count* < m + 3
≤	Less than or equal to	<=	time <= limit	*time* ≤ *limit*
>	Greater than	>	time > limit	*time* > *limit*
≥	Greater than or equal to	>=	age >= 21	*age* ≥ 21

Tính giá trị biểu thức logic

- Kiểu dữ liệu bool
  - Trả về true hoặc false
  - true, false là các hằng định nghĩa sẵn trong thư viện
- Bảng giá trị chân lý
  - Display 2.2 trong slide sau

Tính giá trị biểu thức logic:
Display 2.2 Bảng giá trị chân lý

Display 2.2 Truth Tables

AND

Exp_1	*Exp_2*	*Exp_1* && *Exp_2*
true	true	true
true	false	false
false	true	false
false	false	false

OR

Exp_1	*Exp_2*	*Exp_1* || *Exp_2*
true	true	true
true	false	true
false	true	true
false	false	false

NOT

Exp	!(*Exp*)
true	false
false	true

Display 2.3
Thứ tự ưu tiên các phép toán (1/4)

Display 2.3 Precedence of Operators

`::`	Scope resolution operator
`.`	Dot operator
`->`	Member selection
`[]`	Array indexing
`( )`	Function call
`++`	Postfix increment operator (placed after the variable)
`--`	Postfix decrement operator (placed after the variable)
`++`	Prefix increment operator (placed before the variable)
`--`	Prefix decrement operator (placed before the variable)
`!`	Not
`-`	Unary minus
`+`	Unary plus
`*`	Dereference
`&`	Address of
`new`	Create (allocate memory)
`delete`	Destroy (deallocate)
`delete[]`	Destroy array (deallocate)
`sizeof`	Size of object
`( )`	Type cast

Highest precedence (done first)

Display 2.3
Thứ tự ưu tiên các phép toán (2/4)

*	Multiply
/	Divide
%	Remainder (modulo)
+	Addition
–	Subtraction
<<	Insertion operator (console output)
>>	Extraction operator (console input)

Lower precedence (done later)

Display 2.3
Thứ tự ưu tiên các phép toán (3/4)

Display 2.3 Precedence of Operators

All operators in part 2 are of lower precedence than those in part 1.

<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal
!=	Not equal
&&	And
||	Or

Display 2.3
Thứ tự ưu tiên các phép toán (4/4)

=	Assignment
+=	Add and assign
−=	Subtract and assign
*=	Multiply and assign
/=	Divide and assign
%=	Modulo and assign
? :	Conditional operator
throw	Throw an exception
,	Comma operator

Lowest precedence (done last)

Ví dụ về thứ tự ưu tiên

- Số học tính trước logic
  - x + 1 > 2 || x + 1 < -3 có nghĩa là:
    - (x + 1) > 2 || (x + 1) < -3
- Tính đoản mạch biểu thức logic
  - Short-circuit evaluation
  - (x >= 0) && (y > 1)
  - Cẩn thận với toán tử tự tăng!
    - (x > 1) && (y++)
- Dùng số nguyên như giá trị logic
  - Số khác 0 → true
  - 0 → false

Các cơ chế rẽ nhánh

- Lệnh if-else
  - Lựa chọn giữa 2 lệnh dựa trên biểu thức điều kiện
  - Ví dụ:

```
if (hrs > 40)
  grossPay = rate*40 + 1.5*rate*(hrs-40);
else
  grossPay = rate*hrs;
```

Cú pháp lệnh if-else

- Cú pháp hình thức:
  if (<boolean_expression>)
  <yes_statement>
  else
  <no_statement>
- Chú ý là mỗi lựa chọn chỉ là MỘT lệnh!
- Để có nhiều lệnh thực hiện trong 1 nhánh → hãy dùng lệnh gộp

Lệnh gộp/tạo khối

- Chỉ được thực thi 1 lệnh ở mỗi nhánh
- Ta phải dùng lệnh gộp { } cho 1 nhóm lệnh
  - Còn được gọi là lệnh tạo khối
- Mỗi khối cần có 1 lệnh tạo khối
  - Ngay cả khi khối chỉ có 1 lệnh
  - Làm chương trình dễ đọc hơn

Ví dụ lệnh tạo khối

- Chú ý cách lùi đầu dòng trong ví dụ:

```
if (myScore > yourScore)
{
        cout << "I win!\n";
        wager = wager + 100;
}
else
{
        cout << "I wish these were golf scores.\n";
        wager = 0;
}
```

Lỗi thường gặp

- Nhầm lẫn phép "=" và phép "=="
- Một là “phép gán” (=)
- Một là “phép so sánh bằng” (==)
  - Rất khác nhau trong C++!
  - Ví dụ:

```
if (x = 12)  ←Chú ý phép toán sử dụng!
    Do_Something
else
    Do_Something_Else
```

else là tùy chọn

- Về else là tùy chọn
  - Nếu trong nhánh false (else) bạn chẳng muốn làm gì thì có thể lược bớt nhánh này
  - Ví dụ:

```
if (sales >= minimum)
    salary = salary + bonus;
cout << "Salary = " << salary;
```

  - Chương trình sẽ tiếp tục thực thi lệnh cout

Các lệnh lồng nhau

- Lệnh if-else có thể chứa lệnh nhỏ hơn
  - Lệnh bao ngoài có thể là lệnh kép hoặc lệnh đơn (như ta vừa thấy)
  - Lệnh bên trong có thể là bất cứ lệnh gì, kể cả là một lệnh if-else khác!
  - Ví dụ:

```
if (speed > 55)
    if (speed > 80)
        cout << "You're really speeding!";
    else
        cout << "You're speeding.";
```

    - Chú ý lùi đầu dòng hợp lý!

Lệnh if-else nhiều nhánh

- Không mới, chỉ lùi đầu dòng là khác
- Tránh được lùi đầu dòng "quá nhiều"
  - Cú pháp:

Multiway if-else Statement

SYNTAX

```
if (Boolean_Expression_1)
     Statement_1
else if (Boolean_Expression_2)
     Statement_2
           .
           .
           .
else if (Boolean_Expression_n)
     Statement_n
else
     Statement_For_All_Other_Possibilities
```

Ví dụ lệnh if-else nhiều nhánh

EXAMPLE

```
if ((temperature < -10) && (day == SUNDAY))
    cout << "Stay home.";
else if (temperature < -10) //and day != SUNDAY
    cout << "Stay home, but call work.";
else if (temperature <= 0) //and temperature >= -10
    cout << "Dress warm.";
else //temperature > 0
    cout << "Work hard and play hard.";
```

The Boolean expressions are checked in order until the first `true` Boolean expression is encountered, and then the corresponding statement is executed. If none of the Boolean expressions is *true*, then the *Statement_For_All_Other_Possibilities* is executed.

Lệnh switch

- Là một lệnh khác để điều khiển rẽ nhiều nhánh
- Sử dụng biểu thức điều khiển có giá trị trả về kiểu bool (true hoặc false)
- Cú pháp:
  - Slide sau

Cú pháp lệnh switch

switch Statement

SYNTAX

```
switch (Controlling_Expression)
{
    case Constant_1:
        Statement_Sequence_1
        break;
    case Constant_2:
        Statement_Sequence_2
        break;
                    .
                    .
                    .
    case Constant_n:
        Statement_Sequence_n
        break;
    default:
        Default_Statement_Sequence
}
```

You need not place a `break` *statement in each case. If you omit a* `break`*, that case continues until a* `break` *(or the end of the* `switch` *statement) is reached.*

Ví dụ lệnh switch

EXAMPLE

```
int vehicleClass;
double toll;
cout << "Enter vehicle class: ";
cin >> vehicleClass;

switch (vehicleClass)
{
    case 1:
        cout << "Passenger car.";
        toll = 0.50;
        break;
    case 2:
        cout << "Bus.";
        toll = 1.50;
        break;
    case 3:
        cout << "Truck.";
        toll = 2.00;
        break;
    default:
        cout << "Unknown vehicle class!";
  }
```

If you forget this **break**, *then passenger cars will pay $1.50.*

Lệnh switch: nhiều nhãn case

- Chương trình sẽ thực thi switch tới khi gặp lệnh break
  - switch cung cấp một “lối vào”
  - Example:

```
case "A":
case "a":
    cout << "Excellent: you got an "A"!\n";
    break;
case "B":
case "b":
    cout << "Good: you got a "B"!\n";
    break;
```

  - Chú ý là có thể nhiều nhãn trỏ tới cùng “lối vào”

Lỗi thường gặp với switch

- Quên lệnh break;
  - Đây không phải lỗi biên dịch
  - Chương trình đơn thuần thực thi cả các nhãn case phía sau tới khi gặp được 1 lệnh break;
- Ứng dụng hay gặp nhất: TẠO MENU
  - Cho bạn một hình dung rõ ràng về “bức tranh toàn cảnh”
  - Thể hiện hiệu quả cấu trúc menu
  - Mỗi nhánh là một lựa chọn của menu

Ví dụ menu dùng switch

- Lệnh switch và menu là “cặp đôi hoàn hảo” :

```
switch (response)
{
        case "1":
                // Execute menu option 1
                break;
        case "2":
                // Execute menu option 2
                break;
        case 3":
                // Execute menu option 3
                break;
        default:
                cout << "Please enter valid response.";
}
```

Toán tử điều kiện

- Còn được gọi là “toán tử tam nguyên”
  - Cho phép nhúng các điều kiện vào biểu thức
  - Về cơ bản đây là toán tử viết tắt của “if-else”
  - Ví dụ:
    ```
    if (n1 > n2)
        max = n1;
    else
        max = n2;
    ```
  - Có thể viết thành:
    ```
    max = (n1 > n2) ? n1 : n2;
    ```
    - "?" và ":" tạo thành toán tử tam nguyên này

Lặp

- 3 kiểu lặp trong C++
  - while
    - Linh hoạt nhất
    - Không “giới hạn”
  - do-while
    - Kém linh hoạt nhất
    - Luôn thực thi thân vòng lặp ít nhất 1 lần
  - for
    - Là phép lặp “đếm” tự nhiên

Cú pháp lệnh lặp while

Syntax for `while` and `do-while` Statements

A `while` STATEMENT WITH A SINGLE STATEMENT BODY

```
while (Boolean_Expression)
    Statement
```

A `while` STATEMENT WITH A MULTISTATEMENT BODY

```
while (Boolean_Expression)
{
    Statement_1
    Statement_2
      .
      .
      .
    Statement_Last
}
```

Ví dụ lệnh lặp while

- Xem xét đoạn mã:

```
count = 0;                  // Khởi tạo
while (count < 3)           // Điều kiện lặp
{
      cout << "Hi ";        // Thân vòng lặp
      count++;                        // Biểu thức cập nhật
}
```

  - Thân vòng lặp được thực hiện bao nhiêu lần?

Cú pháp lệnh lặp do-while

A do-while STATEMENT WITH A SINGLE-STATEMENT BODY

```
do
    Statement
while (Boolean_Expression);
```

A do-while STATEMENT WITH A MULTISTATEMENT BODY

```
do
{
    Statement_1
    Statement_2
      .
      .
      .
    Statement_Last
} while (Boolean_Expression);
```

Do not forget the final semicolon.

Ví dụ lệnh lặp do-while

- ```
  count = 0;            // Khởi tạo
  do
  {
      cout << "Hi "; // Thân vòng lặp
      count++;              // Biểu thức cập nhật
  } while (count < 3);      // Điều kiện lặp
  ```
  - Thân vòng lặp được thực hiện bao nhiêu lần?
  - Lệnh lặp do-while luôn thực thi thân vòng lặp ít nhất 1 lần!
  ```

So sánh while và do-while

- Rất giống nhau nhưng…
  - Có một điểm khác biệt quan trọng
    - Vấn đề là "KHI NÀO" biểu thức logic được kiểm tra
    - while: kiểm tra TRƯỚC khi thân lặp được thực hiện
    - do-while: kiểm tra SAU khi thân lặp được thực hiện
- Ngoài khác biệt này, về cơ bản chúng giống hệt nhau!
- while phổ biến hơn vì nó linh hoạt hơn

Toán tử dấu phẩy

- Tính một danh sách các biểu thức, trả về giá trị của biểu thức cuối cùng
- Được dùng nhiều nhất trong vòng lặp for
- Ví dụ:
  first = (first = 2, second = first + 1);
  - first được gán giá trị bằng 3
  - second được gán giá trị bằng 3
- Không nói chắc được thứ tự tính các biểu thức

Cú pháp lệnh lặp for

for (Khởi_tạo; Biểu_thức_logic; Cập_nhật)
    Thân_vòng_lặp

- Giống như if-else, Thân_vòng_lặp có thể là một khối lệnh
  - Thật ra ta thường gặp dạng khối lệnh hơn

Ví dụ vòng lặp for

- for (count=0;count<3;count++)
  {
  cout << "Hi "; // Thân vòng lặp
  }
- Thân vòng lặp được thực hiện bao nhiêu lần?
- Khởi_tạo, Điều_kiện_lặp và Cập_nhật đều được đưa vào cấu trúc lệnh lặp for.
- Bản chất lệnh lặp này dựa trên đếm

Các vấn đề liên quan đến lệnh lặp

- Biểu thức điều kiện của lệnh lặp có thể là bất cứ biểu thức logic nào
- Ví dụ:

```
while (count<3 && done!=0)
{
    // Do something
}
for (index=0;index<10 && entry!=-99)
{
    // Do something
}
```

Lỗi thường gặp: Đặt dấu ; nhầm chỗ

- Hãy cẩn thận vì bạn có thể đặt dấu ; nhầm chỗ
  - Ví dụ:

```
while (response != 0) ;←
{
    cout << "Enter val: ";
    cin >> response;
}
```

  - Chú ý dấu “;” phía sau điều kiện của while!
- Kết quả của ví dụ trên là: Lặp vô hạn!

Lỗi thường gặp: Lặp vô hạn

- Điều kiện lặp phải cho giá trị false ở lần lặp nào đó
  - Nếu không → lặp vô hạn.
  - Ví dụ:

```
while (1)
{
    cout << "Hello ";
}
```

  - Đây là một lệnh lặp hoàn toàn hợp lệ trong C++ → luôn vô hạn!
- Lặp vô hạn đôi khi là có chủ ý
  - Ví dụ: trong các hệ thống nhúng

Lệnh break và continue

- Luồng điều khiển
  - Các lệnh lặp cho phép ta điều khiển "uyển chuyển" việc ra/vào luồng chương trình
  - Trong số ít trường hợp, ta có thể thay đổi luồng tự nhiên
- break;
  - Buộc lệnh lặp dừng ngay lập tức.
- continue;
  - Bỏ qua phần còn lại của thân vòng lặp
- Những lệnh này ảnh hưởng tới luồng tự nhiên
  - Chỉ dùng khi thực sự cần thiết!

Lệnh lặp lồng nhau

- Nhắc lại: Ta có thể đặt bất cứ lệnh C++ hợp lệ nào vào thân vòng lặp
- Điều đó có nghĩa thân vòng lặp có thể là một lệnh lặp khác!
  - Gọi là các lệnh lặp lồng nhau
- Đòi hỏi bạn phải lùi đầu dòng cẩn thận:

```
for (outer=0; outer<5; outer++)
    for (inner=7; inner>2; inner--)
        cout << outer << inner;
```

  - Chú ý là ví dụ này không cần { } vì mỗi thân lặp chỉ có một lệnh
  - Phong cách lập trình tốt: thêm { } cho code dễ đọc

Tóm tắt 1

- Biểu thức logic
  - Tương tự như biểu thức số học → kết quả là true hoặc false
- Các lệnh rẽ nhánh của C++
  - if-else, switch
  - Lệnh switch nên được dùng khi muốn tạo menu
- Các lệnh lặp của C++
  - while
  - do-while
  - for

Tóm tắt 2

- Lệnh lặp do-while
  - Thực thi thân vòng lặp ít nhất 1 lần
- Lệnh lặp for
  - Về bản chất là phép lặp dựa trên đếm
- Có thể thoát sớm các lệnh lặp
  - Lệnh break
  - Lệnh continue
  - Phong cách lập trình: Nên hạn chế dùng 2 lệnh này

Chuẩn bị bài tới

- Đọc chương 3 giáo trình.