

Chương 3: Cấu trúc điều khiển

Giảng viên: Ph.D Nguyễn Văn Hòa
Khoa KT-CN-MT – ĐH An Giang

Giới thiệu

- Dữ liệu và tác vụ là 2 yếu tố cơ bản của CT
- Mọi sự kết hợp của chúng gắn liền với cấu trúc điều khiển
- Cấu trúc điều khiển là tập hợp các quy tắc xác định thứ tự thực hiện chương trình
- Xét về cấu trúc thì có 3 loại điều khiển
 - Điều khiển trong biểu thức
 - Điều khiển giữa các lệnh (phát biểu): như cấu trúc điều kiện hay cấu trúc lặp
 - Điều khiển trong chương trình con: gọi trả về hay đệ qui

Giới thiệu (tt)

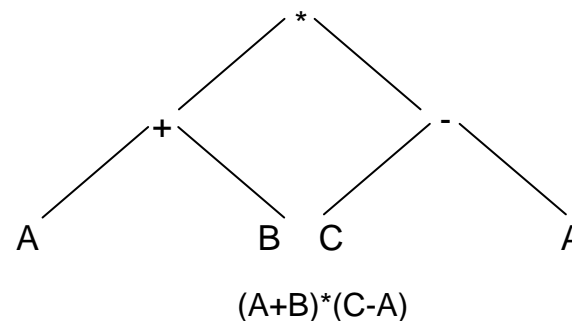
- Xét về thiết kế ngôn ngữ thì có 2 loại điều khiển
 - Điều khiển ngầm: được thiết kế trong ngôn ngữ LT, VD qui tắc ưu tiên của các toán tử
 - Điều khiển tường minh: được xác định bởi programmer
- Hai cấu trúc điều khiển
 - Điều khiển tuần tự
 - Điều khiển cạnh tranh (concurrency) : 2 hoặc nhiều hơn đoạn chương trình được thực thi song song
- Cấu trúc điều khiển tốt
 - Dễ viết
 - Dễ đọc

Nội dung chính của chương

- Điều khiển trong biểu thức
- Lệnh lựa chọn hay điều kiện
- Lệnh lặp
- Rẽ nhánh không điều kiện
- Luồng điều khiển không tuần tự

Điều khiển trong biểu thức

- Cơ chế điều khiển trong biểu thức là sự chồng chất hàm (functional composition)
 - Tác vụ hay phép toán
 - Các đối số hay toán hạng
- Toán hạng: hằng, các kết quả của các phép tham khảo dữ liệu (biến) hoặc kết quả của các phép toán khác
- Cơ chế chồng được biểu diễn bởi một cấu trúc cây



Điều khiển trong biểu thức (tt)

- Thứ tự ưu tiên của các toán tử
- Các toán tử cùng độ ưu tiên
- Thứ tự các toán hạng
 - Hiệu ứng lề
- Tính đa năng của toán tử

Thứ tự ưu tiên các toán tử

- Kết quả của biểu thức $3 + 5 * 2 ?? \rightarrow$ phụ thuộc thứ tự thực hiện các toán tử
- Thứ tự thực hiện các toán tử trong NNLT

FORTRAN 95

**

*, /

-âm, +dương

+, -

C

postfix ++, --

prefix ++,

-âm, +dương

*, /, %

+, -

Ada

**, abs

*, /, mod

rem

-âm, +dương

+, -

Các toán tử cùng ưu tiên (associativity)

- Các phép toán có cùng ưu tiên, + và -, cần có 1 qui luật để xác định thứ tự thực hiện
- VD $a-b+c-d$
- Các>NNLT qui định thứ tự thực hiện
 - Từ trái sang phải
 - Trừ Fortran (từ phải sang trái)
- Ada bắt buộc phải có dấu ngoặc $((a-b)+c)-d$
- Nếu có ngẫu ngoặc thì thứ tự ưu tiên của các toán tử bị mất đi và thực hiện theo dấu ngoặc

Biểu thức điều kiện

- Toán tử «?» là toán tử tam phân (ternary)
- Biểu thức điều kiện chỉ có trong ngôn ngữ C
- VD `average = (count == 0)? 0 : sum / count`
- Có thể dùng câu lệnh `if-then-else` thay thế

```
if (count == 0)
    average = 0;
else
    average = sum / count;
```

Biểu thức quan hệ

- Biểu thức quan hệ: 1 toán tử quan hệ và 2 toán hạng có nhiều kiểu khác nhau
- Trị của biểu thức quan hệ: đúng hoặc sai
- VD toán tử quan hệ


	FORTRAN 95	C	Ada
Bằng	.EQ. ==	==	=
Không Bằng	.NE. <>	!=	/=
Lớn hơn	.GT. >	>	>
Nhỏ hơn	.LT. <	<	<
Lớn hơn or bằng	.GN. >=	>=	>=
Nhỏ hơn or bằng	.LN. <=	<=	<=

Biểu thức logic

- Biểu thức logic: 1 toán tử logic và 2 toán hạng có kiểu logic
- Trị của biểu thức logic: đúng hoặc sai
- VD các toán tử logic

FORTRAN 77	FORTRAN 90	C	Ada
.AND.	and	&&	and
.OR.	or		or
.NOT.	not	!	not
			xor

Thứ tự thực hiện các toán tử : C

- Hậu tố (++ , --)
 - Tiền tố (++ , -- , !) âm dương (+ , -)
 - *, / , %
 - + , -
 - < , > , <= , >=
 - == , !=
 - &&
 - ||
- 

Hiệu ứng lề

- Hiệu ứng lề là 1 phép toán trả về kết quả ẩn
- Hàm hiệu ứng lề là hàm thay đổi biến không cục bộ hoặc có truyền quy chiếu

```
int a = 5;  
int fun1() { a = 17; return 3; }  
void fun2() { a = a + fun1(); }  
void main() {  
    fun2();  
}
```

- Kết quả của a là 8 hay 20

Hiệu ứng lè

- Giải pháp để giải quyết hiệu ứng lè
 - NNLT không cho phép hàm tham chiếu các biến không cục bộ và truyền quy chiếu
 - Ưu điểm : dễ thực hiện
 - Khuyết điểm : không linh hoạt
 - NNLT phải qui định thứ tự ưu tiên của các toán hạng
 - Khuyết điểm : giảm khả năng tối ưu code của trình biên dịch
 - Ngôn ngữ C trả về giá trị của `a` là 20

Cú pháp của biểu thức

- **Dạng trung tố (infex)**
 - Phổ biến và tự nhiên nhất: kí hiệu phép toán được viết giữa 2 toán hạng
 - VD $(A + B) * (C - A)$
- **Dạng tiền tố (prefix)**
 - Kí hiệu phép toán được viết trước các toán hạng
 - VD $* (+ (A B)) - (C A)$
- **Dạng hậu tố (posfix)**
 - Kí hiệu phép toán được viết sau các toán hạng
 - VD $((A B) +) (C A) -$ *

Toán tử đa năng hóa (overloaded)

- Một toán tử có thực hiện nhiều phép toán → toán tử đa năng hóa
- Phép toán + với kiểu số nguyên và kiểu số thực
- Phép toán &: lấy địa chỉ và phép toán And (bit)
 - `int x, y, z ;`
 - `x = &y //` trả về địa chỉ ô nhớ của y cho x
 - `x = y&z //` trả về giá trị của phép toán And trên y , z
- Phép toán *
 - Trả về trị của ô nhớ mà pointer trỏ đến hoặc phép toán nhân

Lệnh điều khiển

- Ba loại cấu trúc điều khiển cơ bản
 - Lệnh tuần tự
 - Lệnh gán, lệnh gọi chương trình con
 - Lệnh xuất / nhập
 - Lệnh lựa chọn hay điều kiện
 - Lệnh lặp
- Lệnh rẽ nhánh không điều kiện

Lệnh lựa chọn hay điều kiện

- Lệnh điều kiện là một lệnh biểu thị sự lựa chọn của hai hoặc đa nhánh để thực hiện
- Chia làm 2 loại
 - Chỉ có 2 lựa chọn (lệnh IF)
 - Nhiều lựa chọn (lệnh CASE)

Lệnh lựa chọn : 2 lựa chọn

- Dạng phổ biến :

```
if control_expression then clause  
else clause
```

- Các yếu tố trong lệnh 2 lựa chọn

- ❑ Dạng và kiểu của biểu thức lựa chọn như thế nào : quan hệ, toán và logic?
- ❑ Các câu lệnh gì sau then và sau else?
- ❑ Lệnh lựa chọn có lồng nhau hay không?

Biểu thức lựa chọn (2 lựa chọn)

- ALGOL 60 : chỉ dùng biểu thức logic
if (boolean_expr) then *stmt*
else *stmt*
- C (89 trở về trước) : chỉ dùng biểu thức logic
- C (99) và C++ : biểu thức toán và logic
 - VD `if(5-3) printf(«A») else printf(«B»);`
 - C/C++/Java: `if (expr) stmt else stmt`
- Ada, Java và C# chỉ cho phép dùng biểu thức logic

Sự lựa chọn lòng nhau

- Thí dụ trong Java

```
if (sum == 0)
    if (count == 0)
        result = 0;
    else result = 1;
```

- Lệnh `if` nào sẽ đi cùng với lệnh `else`?
- NN Java qui định lệnh `if` và lệnh `else` gần nhau nhất sẽ đi cùng nhau

- C, C++, C# yêu cầu dùng `{ }` để phân định
- Perl yêu cầu câu lệnh ghép (`if else` đầy đủ)

Sự lựa chọn lòng nhau

■ C, C++, C#

```
if (sum == 0){  
    if (count == 0)  
        result = 0;  
}else result = 1;
```

■ Ada

```
if (sum == 0) then  
    if (count == 0)then  
        result = 0;  
    end if  
else  
    result = 1;  
end if
```

```
if (sum == 0) then  
    if (count == 0)then  
        result = 0;  
    else  
        result = 1;  
    end if  
end if
```

Lệnh lựa chọn đa nhánh

- Cho phép lựa chọn 1 nhánh trong số nhiều nhánh lệnh để thực hiện
- Các yếu tố trong lệnh lựa chọn đa nhánh
 - Kiểu và dạng của biểu thức điều khiển là gì?
 - Công việc của từng nhánh lệnh là gì?
 - Có nhánh lệnh không thỏa điều kiện không? Nếu có nó thực hiện lệnh gì?

Mô hình switch-case

```
switch (expr) {  
    case const_expr_1 : stmt_1;  
    ...  
    case const_expr_n : stmt_n;  
    [default: stmt_{n+1};]  
}
```

- Switch-case trong C, C++, Java
 - Biểu thức điều khiển phải là kiểu nguyên
 - Câu lệnh được chọn có thể 1 câu lệnh đơn hoặc lệnh hợp thành
 - Default : được chọn nếu không có giá trị nào thỏa expr

Mô hình switch-case

- Switch-case trong Ada

```
case expression is
    when choice list => stmt_sequence;
    ...
    when choice list => stmt_sequence;
    when others => stmt_sequence; ]
end case;
```

- Dễ đọc hơn switch-case của C

- C# chỉ cho phép thực hiện 1 lệnh đơn trong trường hợp được chọn

Chọn đa nhánh với lệnh `if`

- Lệnh chọn đa nhánh có thể chuyển thành lệnh chọn 2 nhánh với `else-if`

```
if (expr == const_expr_1 ) stmt_1;  
else if (expr == const_expr_2) stmt_2
```

...

```
Else if (expr == const_expr_n) stmt_n;  
else stmt_n+1;
```

- ❑ Phải kết hợp với lệnh nhảy (`goto`)
- ❑ Code rất nghèo nàn

Lệnh lặp

- Lệnh lặp là để thực hiện một số lần lệnh đơn hay lệnh hợp thành
- Các yếu tố trong lệnh lặp
 - Làm thế nào để kiểm soát lặp?
 - Kiểm soát lặp xuất hiện ở đâu trong vòng lặp?

Lệnh lặp với bộ đếm

- Lệnh có 1 biến đếm, giá trị của biến này từ giá trị bắt đầu (*initial*) đến giá trị kết thúc (*terminal*) và giá trị của bước nhảy (*stepsize*)
- Các yếu tố trong lệnh lặp :
 - Biến lặp có kiểu gì và phạm vi nào?
 - Giá trị của biến lặp khi vòng lặp kết thúc là bao nhiêu?
 - Giá trị của biến lặp có được thay đổi trong thân vòng lặp không? Nếu có thì có ảnh hưởng đến vòng lặp không?

VD - lệnh lặp với bộ đếm

■ Cú pháp của Fortran 90

- ❑ **do** *bien_lap* = *tri_bat_dau*, *tri_ket_thuc* [, *buoc_nhay*]
- ❑ Trị của bước nhảy là bất kỳ (trừ 0), mặc định 1
- ❑ *tri_bat_dau*, *tri_ket_thuc* có thể là biểu thức
- ❑ Kiểu của biến lặp phải là kiểu số nguyên
- ❑ Biến lặp không được thay đổi trong thân vòng lặp

■ Fortran 95

- ❑ **do** *bien_lap* = *tri_bat_dau*, *tri_ket_thuc* [, *buoc_nhay*]
...
end do

VD - lệnh lặp với bộ đếm (tt)

■ Cú pháp lệnh `for` của Pascal

```
for bien_lap := bat_dau (to | downto)  
    ket_thuc do ...
```

- ❑ `bien_lap` có kiểu số nguyên
- ❑ Sau khi kết thúc vòng lặp, giá trị của `bien_lap` là không xác định
- ❑ Giá trị của `bien_lap` không thể thay đổi trong thân vòng lặp;
- ❑ `bat_dau`, `ket_thuc` có thể là biểu thức, nhưng các tham số có thể thay đổi trong vòng lặp và không ảnh hưởng đến vòng lặp

VD - lệnh lặp với bộ đếm (tt)

- Cú pháp lệnh for trong Ada

- ❑ **for** `bien_lap` **in** [`reverse`] `day_roi_rac` **loop**

- ...

- end loop**

- ❑ `day_roi_rac` : miền con số nguyên, 1..10, hoặc kiểu liệt kê `monday..friday`

- ❑ Phạm vi của biến có bao gồm vòng lặp

- ❑ Giá trị của biến lặp là không xác định sau khi vòng lặp kết thúc

VD - lệnh lặp với bộ đếm (tt)

■ Cú pháp của `for` trong C

```
for ([expr_1] ; [expr_2] ; [expr_3] )  
    statement
```

- Mọi thứ có thể thay đổi trong thân vòng lặp
- Biểu thức *expr_1* được định lượng 1 lần (trước khi thực hiện vòng lặp), *expr_2* và *expr_3* được định lượng ở mọi lần lặp

■ C++ vs C

- C++ : cho phép khai báo kiểu trong *expr_1*
 - `For(int count =0; count<len; count++) {...}`

Lệnh lặp có điều kiện

- Lệnh lặp chỉ được thực hiện khi điều kiện đúng
- Các yếu tố trong lệnh lặp có điều kiện
 - Kiểm tra điều kiện trước hay sau
 - Có phải lệnh lặp có điều kiện là trường hợp đặc biệt của lệnh lặp với bộ đếm
- Cú pháp

□ <i>While (ctrl_expr)</i>	<i>do</i>
<i>loop body</i>	<i>loop body</i>
	<i>while (ctrl_expr)</i>

Lệnh lặp có điều kiện (tt)

- Pascal phân chia rõ ràng kiểm tra điều kiện trước và sau : `while-do` và `repeat-until`
- C và C++ dùng cả (`while-do` and `do-while`); kiểm tra trước là lặp nếu điều kiện đúng nhưng kiểm tra điều kiện sau là lặp đến khi điều kiện sai
- Cũng giống như C, nhưng biểu thức điều kiện của Java có kiểu `boolean`
- Ada chỉ có lặp kiểm tra trước

Rẽ nhánh không điều kiện

- Lệnh rẽ nhánh không điều kiện được đưa ra 1960s
- Cho phép thay đổi thứ tự thực hiện chương trình
- Cơ chế phổ biến nhất là lệnh: `goto`
- Một số NNLT không hỗ trợ lệnh `goto`
- C# cung cấp lệnh `goto`, có thể dùng trong `switch-case`
- Lệnh `goto` làm cho CT trở nên khó đọc và khó bảo trì

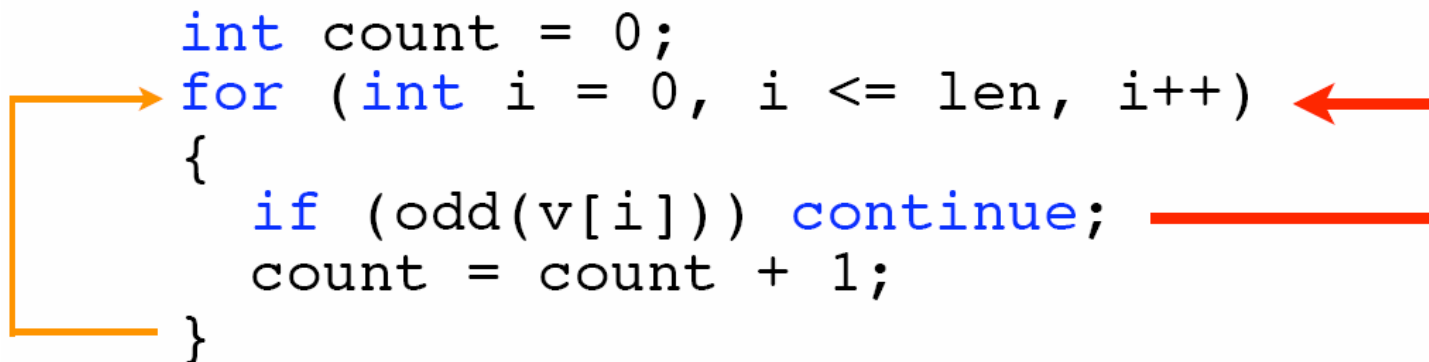
Luồng điều khiển không từng tự (Nonlocal control flow)

- Cho phép thoát khỏi luồng điều khiển thông thường
- Ứng dụng
 - Thoát sớm trong các cấu trúc lặp
 - Trả về (return) sớm trong các hàm
 - Bắt các ngoại lệ
- VD
 - Continue & break
 - Return
 - Try/catch

Lệnh `continue` & `break`

- *Lệnh `continue`* sẽ kết thúc vòng lặp hiện hành và chuyển đến vòng lặp tiếp theo
- Khi gặp lệnh này, các câu lệnh còn lại trong thân của vòng lặp sẽ được bỏ qua
- VD

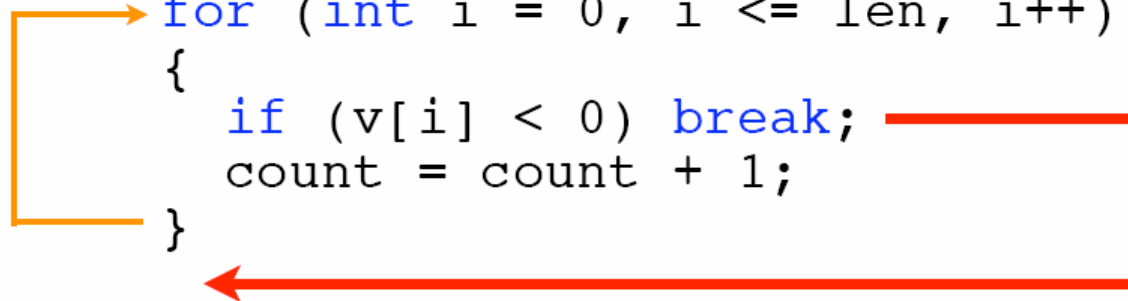
```
int count = 0;
for (int i = 0, i <= len, i++)
{
    if (odd(v[i])) continue;
    count = count + 1;
}
```



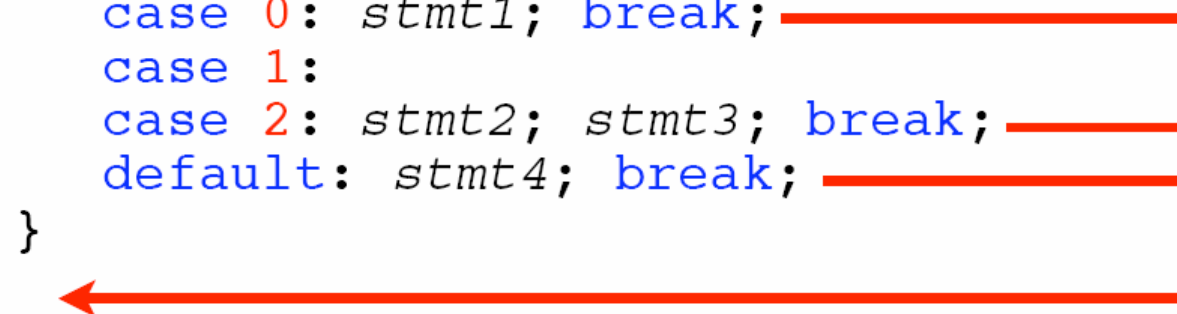
Lệnh `continue` & `break` (tt)

- Lệnh *break* để thoát khỏi các cấu trúc lặp hoặc *switch*

```
int count = 0;
for (int i = 0, i <= len, i++)
{
    if (v[i] < 0) break;
    count = count + 1;
}
```



```
switch (expr)
{
    case 0: stmt1; break;
    case 1:
    case 2: stmt2; stmt3; break;
    default: stmt4; break;
}
```



Lệnh return

```
int F(int v[], int len)
{
    int count = 0;
    for (int i = 0, i <= len, i++)
    {
        if (v[i] < 0) return -1;
        count = count + 1;
    }
    return count;
}

int main()
{
    ...
    c = F(a, 10);
    ...
}
```

The diagram illustrates the execution flow of the `return` statement. An orange arrow points from the `for` loop to the `return count;` statement. A red arrow points from the `return -1;` statement to the `return count;` statement. Another red arrow points from the `return -1;` statement to the `c = F(a, 10);` statement in the `main` function.

Lệnh continue và break (Java)

```
block1:
while (XXX) {
    ...
    while (XXX) {
        ...
        if (XXX) break block1;
    }
}
loop1:
while (XXX) {
    ...
    while (XXX) {
        ...
        if (XXX) continue loop1;
    }
}
```


Try/catch

- Cú pháp

```
try{  
    // đoạn mã có khả năng gây ra ngoại lệ  
}  
catch(Exception e1){  
    // Nếu các lệnh trong khối 'try' tạo ra ngoại lệ có loại e1  
}  
...  
catch(Exception eN){  
    ...  
}  
Finally {  
    // khối lệnh nay luôn được thực hiện cho dù ngoại lệ có xảy ra hay  
    không  
}
```

Try/catch

■ VD

```
try {  
    x = f(x,y);  
}  
catch (ArithmeticException) {  
    System.out.println("An arithmetic error  
occurred!");  
    System.exit(0);  
}
```

- ❑ Nếu hàm $f(x,y)$ không có lỗi, tiến trình vẫn tiếp tục
- ❑ Nếu hàm $f(x,y)$ lỗi, `ArithmeticException` được chuyển tới JVM