CHƯƠNG 3

CẤU TRÚC ĐIỀU KHIỂN VÀ DỮ LIỆU KIỂU MẢNG

Cấu trúc rẽ nhánh
Cấu trúc lặp
Mảng dữ liệu
Mảng hai chiều

I. CẤU TRÚC RẼ NHÁNH

Nói chung việc thực hiện chương trình là hoạt động tuần tự, tức thực hiện từng lệnh một từ câu lệnh bắt đầu của chương trình cho đến câu lệnh cuối cùng. Tuy nhiên, để việc lập trình hiệu quả hơn hầu hết các NNLT bậc cao đều có các câu lệnh rẽ nhánh và các câu lệnh lặp cho phép thực hiện các câu lệnh của chương trình không theo trình tự tuần tự như trong văn bản.

Phần này chúng tôi sẽ trình bày các câu lệnh cho phép rẽ nhánh như vậy. Để thống nhất mỗi câu lệnh được trình bày về cú pháp (tức cách viết câu lệnh), cách sử dụng, đặc điểm, ví dụ minh hoạ và một vài điều cần chú ý khi sử dụng lệnh.

1. Câu lệnh điều kiện if

a. Ý nghĩa

Một câu lệnh **if** cho phép chương trình có thể thực hiện khối lệnh này hay khối lệnh khác phụ thuộc vào một điều kiện được viết trong câu lệnh là đúng hay sai. Nói cách khác câu lệnh **if** cho phép chương trình rẽ nhánh (chỉ thực hiện 1 trong 2 nhánh).

b. Cú pháp

– **if (điều kiện) { khối lệnh 1; } else { khối lệnh 2; }**

– **if (điều kiện) { khối lệnh 1; }**

Trong cú pháp trên câu lệnh if có hai dạng: có else và không có else. điều kiện là một biểu thức lôgic tức nó có giá trị đúng (khác 0) hoặc sai (bằng 0).

Khi chương trình thực hiện câu lệnh if nó sẽ tính biểu thức điều kiện. Nếu điều kiện đúng chương trình sẽ tiếp tục thực hiện các lệnh trong khối lệnh 1, ngược lại nếu

điều kiện sai chương trình sẽ thực hiện khối lệnh 2 (nếu có else) hoặc không làm gì (nếu không có else).

c. Đặc điểm

- Đặc điểm chung của các câu lệnh có cấu trúc là bản thân nó chứa các câu lệnh khác. Điều này cho phép các câu lệnh if có thể lồng nhau.
- Nếu nhiều câu lệnh if (có else và không else) lồng nhau việc hiểu if và else nào đi với nhau cần phải chú ý. Qui tắc là else sẽ đi với if gần nó nhất mà chưa được ghép cặp với else khác. Ví dụ câu lệnh

```
if (n>0) if (a>b) c = a;
else c = b;
```

là tương đương với

```
if (n>0) { if (a>b) c = a; else c = b;}
```

d. Ví dụ minh hoạ

Ví dụ 1 : Bằng phép toán gán có điều kiện có thể tìm số lớn nhất max trong 2 số a, b như sau: `max = (a > b) ? a: b ;`

hoặc max được tìm bởi dùng câu lệnh if:

```
if (a > b) max = a; else max = b;
```

Ví dụ 2 : Tính năm nhuận. Năm thứ n là nhuận nếu nó chia hết cho 4, nhưng không chia hết cho 100 hoặc chia hết 400. Chú ý: một số nguyên a là chia hết cho b nếu phần dư của phép chia bằng 0, tức a%b == 0.

```
#include <iostream.h>
void main()
{
        int nam;
        cout << “Nam = “ ; cin >> nam ;
        if (nam%4 == 0 && year%100 !=0 || nam%400 == 0)
                cout << nam << "la nam nhuan” ;
        else
                cout << nam << "la nam khong nhuan” ;
}
```

Ví dụ 3 : Giải phương trình bậc 2. Cho phương trình $ax^2 + bx + c = 0$ ($a \neq 0$), tìm x.

```
#include <iostream.h>                    // tệp chứa các phương thức vào/ra
#include <math.h>                        // tệp chứa các hàm toán học
void main()
{
      float a, b, c;                     // khai báo các hệ số
      float delta;
      float x1, x2;                      // 2 nghiem
      cout << “Nhap a, b, c:\n” ; cin >> a >> b >> c ;    // qui ước nhập a ≠ 0
      delta = b*b - 4*a*c ;
      if (delta < 0) cout << “ph. trình vô nghiệm\n” ;
      else if (delta==0) cout<<“ph. trình có nghiệm kép:" << -b/(2*a) << '\n';
      else
      {
            x1 = (-b+sqrt(delta))/(2*a);
            x2 = (-b-sqrt(delta))/(2*a);
            cout << “nghiem 1 = " << x1 << " và nghiem 2 = " << x2 ;
      }
}
```


Chú ý: do C++ quan niệm "đúng" là một giá trị khác 0 bất kỳ và "sai" là giá trị 0 nên thay vì viết if (x != 0) hoặc if (x == 0) ta có thể viết gọn thành if (x) hoặc if (!x) vì nếu (x != 0) đúng thì ta có x ≠ 0 và vì x ≠ 0 nên (x) cũng đúng. Ngược lại nếu (x) đúng thì x ≠ 0, từ đó (x != 0) cũng đúng. Tương tự ta dễ dàng thấy được (x == 0) là tương đương với (!x).

2. Câu lệnh lựa chọn switch

a. Ý nghĩa

Câu lệnh if cho ta khả năng được lựa chọn một trong hai nhánh để thực hiện, do đó nếu sử dụng nhiều lệnh if lồng nhau sẽ cung cấp khả năng được rẽ theo nhiều nhánh. Tuy nhiên trong trường hợp như vậy chương trình sẽ rất khó đọc, do vậy C++ còn cung cấp một câu lệnh cấu trúc khác cho phép chương trình có thể chọn một trong nhiều nhánh để thực hiện, đó là câu lệnh switch.

b. Cú pháp

```
switch (biểu thức điều khiển)
{
    case biểu_thức_1: dãy lệnh 1 ;
    case biểu_thức_2: dãy lệnh 2 ;
    case ……...: ..... ;
    case biểu_thức_n: dãy lệnh n ;
    default: dãy lệnh n+1;
}
```

- biểu thức điều khiển: phải có kiểu nguyên hoặc kí tự,
- các biểu_thức_i: được tạo từ các hằng nguyên hoặc kí tự,
- các dãy lệnh có thể rỗng. Không cần bao dãy lệnh bởi cặp dấu {},
- nhánh default có thể có hoặc không và vị trí của nó có thể nằm bất kỳ trong câu lệnh (giữa các nhánh case), không nhất thiết phải nằm cuối cùng.

c. Cách thực hiện

Để thực hiện câu lệnh **switch** đầu tiên chương trình tính giá trị của biểu thức điều khiển (btđk), sau đó so sánh kết quả của btđk với giá trị của các biểu_thức_i bên dưới lần lượt từ biểu thức đầu tiên (thứ nhất) cho đến biểu thức cuối cùng (thứ n), nếu giá trị của btđk bằng giá trị của biểu thức thứ i đầu tiên nào đó thì chương trình sẽ thực hiện dãy lệnh thứ i và tiếp tục thực hiện tất cả dãy lệnh còn lại (từ dãy lệnh thứ i+1) cho đến hết (gặp dấu ngoặc đóng } của lệnh switch). Nếu quá trình so sánh không gặp biểu thức (nhánh case) nào bằng với giá trị của btđk thì chương trình thực hiện dãy lệnh trong default và tiếp tục cho đến hết (sau default có thể còn những nhánh case khác). Trường hợp câu lệnh switch không có nhánh default và btđk không khớp với bất cứ nhánh case nào thì chương trình không làm gì, coi như đã thực hiện xong lệnh switch.

Nếu muốn lệnh switch chỉ thực hiện nhánh thứ i (khi btđk = biểu_thức_i) mà không phải thực hiện thêm các lệnh còn lại thì cuối dãy lệnh thứ i thông thường ta đặt thêm lệnh **break;** đây là lệnh cho phép thoát ra khỏi một lệnh cấu trúc bất kỳ.

d. Ví dụ minh hoạ

Ví dụ 1 : In số ngày của một tháng bất kỳ nào đó được nhập từ bàn phím.

```
int th;
cout << “Cho biết tháng cần tính: “ ; cin >> th ;
switch (th)
```

```
{
        case 1: case 3: case 5: case 7: case 8: case 10:
        case 12: cout << "tháng này có 31 ngày" ; break ;
        case 2: cout << "tháng này có 28 ngày" ; break;
        case 4: case 6: case 9:
        case 11: cout << "tháng này có 30 ngày" ; break;
        default: cout << "Bạn đã nhập sai tháng, không có tháng này" ;
}
```

Trong chương trình trên giả sử NSD nhập tháng là 5 thì chương trình bắt đầu thực hiện dãy lệnh sau case 5 (không có lệnh nào) sau đó tiếp tục thực hiện các lệnh còn lại, cụ thể là bắt đầu từ dãy lệnh trong case 7, đến case 12 chương trình gặp lệnh in kết quả "tháng này có 31 ngày", sau đó gặp lệnh break nên chương trình thoát ra khỏi câu lệnh switch (đã thực hiện xong). Việc giải thích cũng tương tự cho các trường hợp khác của tháng. Nếu NSD nhập sai tháng (ví dụ tháng nằm ngoài phạm vi 1..12), chương trình thấy th không khớp với bất kỳ nhánh case nào nên sẽ thực hiện câu lệnh trong default, in ra màn hình dòng chữ "Bạn đã nhập sai tháng, không có tháng này" và kết thúc lệnh.

Ví dụ 2 : Nhập 2 số a và b vào từ bàn phím. Nhập kí tự thể hiện một trong bốn phép toán: cộng, trừ, nhân, chia. In ra kết quả thực hiện phép toán đó trên 2 số a, b.

```
void main()
{
        float a, b, c ;                 // các toán hạng a, b và kết quả c
        char dau ;                      // phép toán được cho dưới dạng kí tự
        cout << "Hãy nhập 2 số a, b: " ; cin >> a >> b ;
        cout << "và dấu phép toán: " ; cin >> dau ;
        switch (dau)
        {
                case '+': c = a + b ; break ;
                case '–': c = a - b ; break ;
                case 'x': case '.': case '*': c = a * b ; break ;
                case ':': case '/': c = a / b ; break ;
        }
        cout << setiosflags(ios::showpoint) << setprecision(4) ;       // in 4 số lẻ
```

```
        cout << "Kết quả là: " << c ;
    }
```

Trong chương trình trên ta chấp nhận các kí tự x, ., * thể hiện cho phép toán nhân và :, / thể hiện phép toán chia.

3. Câu lệnh nhảy goto

a. Ý nghĩa

Một dạng khác của rẽ nhánh là câu lệnh nhảy **goto** cho phép chương trình chuyển đến thực hiện một đoạn lệnh khác bắt đầu từ một điểm được đánh dấu bởi một nhãn trong chương trình. Nhãn là một tên gọi do NSD tự đặt theo các qui tắt đặt tên gọi. Lệnh goto thường được sử dụng để tạo vòng lặp. Tuy nhiên việc xuất hiện nhiều lệnh goto dẫn đến việc khó theo dõi trình tự thực hiện chương trình, vì vậy lệnh này thường được sử dụng rất hạn chế.

b. Cú pháp

Goto <nhãn> ;

Vị trí chương trình chuyển đến thực hiện là đoạn lệnh đứng sau nhãn và dấu hai chấm (:).

c. Ví dụ minh hoạ

Ví dụ 3 : Nhân 2 số nguyên theo phương pháp Ấn độ.

Phương pháp Ấn độ cho phép nhân 2 số nguyên bằng cách chỉ dùng các phép toán nhân đôi, chia đôi và cộng. Các phép nhân đôi và chia đôi thực chất là phép toán dịch bit về bên trái (nhân) hoặc bên phải (chia) 1 bit. Đây là các phép toán cơ sở trong bộ xử lý, do vậy dùng phương pháp này sẽ làm cho việc nhân các số nguyên được thực hiện rất nhanh. Có thể tóm tắt phương pháp như sau: Giả sử cần nhân m với n. Kiểm tra m nếu lẻ thì cộng thêm n vào kq (đầu tiên kq được khởi tạo bằng 0), sau đó lấy m chia 2 và n nhân 2. Quay lại kiểm tra m và thực hiện như trên. Quá trình dừng khi không thể chia đôi m được nữa (m = 0), khi đó kq là kết quả cần tìm (tức kq = m*n). Để dễ hiểu phương pháp này chúng ta tiến hành tính trên ví dụ với các số m, n cụ thể. Giả sử m = 21 và n = 11. Các bước tiến hành được cho trong bảng dưới đây:

Bước	m (chia 2)	n (nhân 2)	kq (khởi tạo kq = 0)
1	21	11	m lẻ, cộng thêm 11 vào kq = 0 + 11 = 11
2	10	22	m chẵn, bỏ qua
3	5	44	m lẻ, cộng thêm 44 vào kq = 11 + 44 = 55

4	2	88	m chẵn, bỏ qua
5	1	176	m lẻ, cộng thêm 176 vào kq = 55 + 176 = 231
6	0		m = 0, dừng cho kết quả kq = 231

Sau đây là chương trình được viết với câu lệnh goto.

```
void main()
{
        long m, n, kq = 0;              // Các số cần nhân và kết quả kq
        cout << “Nhập m và n: “ ; cin >> m >> n ;
        lap:                            // đây là nhãn để chương trình quay lại
        if (m%2) kq += n;               // nếu m lẻ thì cộng thêm n vào kq
        m = m >> 1;                     // dịch m sang phải 1 bit tức m = m / 2
        n = n << 1;                     // dịch m sang trái 1 bit tức m = m * 2
        if (m) goto lap;                // quay lại nếu m ≠ 0
        cout << “m nhân n =” << kq ;
}
```


II. CẤU TRÚC LẶP

Một trong những cấu trúc quan trọng của lập trình cấu trúc là các câu lệnh cho phép lặp nhiều lần một đoạn lệnh nào đó của chương trình. Chẳng hạn trong ví dụ về bài toán nhân theo phương pháp Ấn độ, để lặp lại một đoạn lệnh chúng ta đã sử dụng câu lệnh goto. Tuy nhiên như đã lưu ý việc dùng nhiều câu lệnh này làm chương trình rất khó đọc. Do vậy cần có những câu lệnh khác trực quan hơn và thực hiện các phép lặp một cách trực tiếp. C++ cung cấp cho chúng ta 3 lệnh lặp như vậy. Về thực chất 3 lệnh này là tương đương (cũng như có thể dùng goto thay cho cả 3 lệnh lặp này), tuy nhiên để chương trình viết được sáng sủa, rõ ràng, C++ đã cung cấp nhiều phương án cho NSD lựa chọn câu lệnh khi viết chương trình phù hợp với tính chất lặp. Mỗi bài toán lặp có một đặc trưng riêng, ví dụ lặp cho đến khi đã đủ số lần định trước thì dừng hoặc lặp cho đến khi một điều kiện nào đó không còn thoả mãn nữa thì dừng … việc sử dụng câu lệnh lặp phù hợp sẽ làm cho chương trình dễ đọc và dễ bảo trì hơn. Đây là ý nghĩa chung của các câu lệnh lặp, do vậy trong các trình bày về câu lệnh tiếp theo sau đây chúng ta sẽ không cần phải trình bày lại ý nghĩa của chúng.

1. Lệnh lặp for

a. Cú pháp

for (dãy biểu thức 1 ; điều kiện lặp ; dãy biểu thức 2) { khối lệnh lặp; }

- Các biểu thức trong các dãy biểu thức 1, 2 cách nhau bởi dấu phảy (,). Có thể có nhiều biểu thức trong các dãy này hoặc dãy biểu thức cũng có thể trống.
- Điều kiện lặp: là biểu thức lôgic (có giá trị đúng, sai).
- Các dãy biểu thức và/hoặc điều kiện có thể trống tuy nhiên vẫn giữ lại các dấu chấm phảy (;) để ngăn cách các thành phần với nhau.

b. Cách thực hiện

Khi gặp câu lệnh **for** trình tự thực hiện của chương trình như sau:

- Thực hiện dãy biểu thức 1 (thông thường là các lệnh khởi tạo cho một số biến),
- Kiểm tra điều kiện lặp, nếu đúng thì thực hiện khối lệnh lặp → thực hiện dãy biểu thức 2 → quay lai kiểm tra điều kiện lặp và lặp lại quá trình trên cho đến bước nào đó việc kiểm tra điều kiện lặp cho kết quả sai thì dừng.

Tóm lại, biểu thức 1 sẽ được thực hiện 1 lần duy nhất ngay từ đầu quá trình lặp sau đó thực hiện các câu lệnh lặp và dãy biểu thức 2 cho đến khi nào không còn thoả điều kiện lặp nữa thì dừng.

c. Ví dụ minh hoạ

Ví dụ 1 : Nhân 2 số nguyên theo phương pháp Ấn độ

```
void main()
{
    long m, n, kq;                    // Các số cần nhân và kết quả kq
    cout << “Nhập m và n: “ ; cin >> m >> n ;
    for (kq = 0 ; m ; m >>= 1, n <<= 1) if (m%2) kq += n ;
    cout << “m nhân n =” << kq ;
}
```

So sánh ví dụ này với ví dụ dùng goto ta thấy chương trình được viết rất gọn. Để bạn đọc dễ hiểu câu lệnh for, một lần nữa chúng ta nhắc lại cách hoạt động của nó thông qua ví dụ này, trong đó các thành phần được viết trong cú pháp là như sau:

- Dãy biểu thức 1: kq = 0,
- Điều kiện lặp: m. Ở đây điều kiện là đúng nếu m ≠ 0 và sai nếu m = 0.