

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 thông nhất mỗi câu lệnh được trình bày về cú pháp (tức cách viết câu lệnh), cách sử dụng, đặc điểm, ví dụ minh họa 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 khỏi 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 họa

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

hoặc max được tìm bởi dùng câu lệnh if:

if ($a > b$) $\text{max} = a$; else $\text{max} = b$;

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 nghiệm
    cout << "Nhập a, b, c:\n" ; cin >> a >> b >> c ; // qui ước nhập  $a \neq 0$ 
    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 << "nghiệm 1 = " << x1 << " và nghiệm 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 ($x \neq 0$) hoặc if ($x == 0$) ta có thể viết gọn thành if (x) hoặc if ($!x$) vì nếu ($x \neq 0$) đúng thì ta có $x \neq 0$ và vì $x \neq 0$ nên (x) cũng đúng. Ngược lại nếu (x) đúng thì $x \neq 0$, từ đó ($x \neq 0$) 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 rõ 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 họa

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 tắc 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 họa

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

Phương pháp Án 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 Ân 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 phẩy (,). 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 phẩy (;) 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 lại 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 thỏa điều kiện lặp nữa thì dừng.

c. Ví dụ minh họa

Ví dụ 1 : Nhân 2 số nguyên theo phương pháp Ân 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 $m \neq 0$ và sai nếu $m = 0$.

- Dãy biểu thức 2: $m >>= 1$ và $n <<= 1$. 2 biểu thức này có nghĩa $m = m >> 1$ (tương đương với $m = m / 2$) và $n = n << 1$ (tương đương với $n = n * 2$).
- Khối lệnh lặp: chỉ có một lệnh duy nhất if ($m \% 2$) kq $+= n$; (nếu phần dư của m chia 2 là khác 0, tức m lẻ thì cộng thêm n vào kq).

Cách thực hiện của chương trình như sau:

- Đầu tiên thực hiện biểu thức 1 tức gán $kq = 0$. Chú ý rằng nếu kq đã được khởi tạo trước bằng 0 trong khi khai báo (giống như trong ví dụ 6) thì thành phần biểu thức 1 ở đây có thể để trống (nhưng vẫn giữ lại dấu ; để phân biệt với các thành phần khác).
- Kiểm tra điều kiện: giả sử $m \neq 0$ (tức điều kiện đúng) for sẽ thực hiện lệnh lặp tức kiểm tra nếu m lẻ thì cộng thêm n vào cho kq.
- Quay lại thực hiện các biểu thức 2 tức chia đôi m và nhân đôi n và vòng lặp được tiếp tục lại bắt đầu bằng việc kiểm tra m ...
- Đến một bước lặp nào đó m sẽ bằng 0 (vì bị chia đôi liên tiếp), điều kiện không thoả, vòng lặp dừng và cho ta kết quả là kq.

Ví dụ 2 : Tính tổng của dãy các số từ 1 đến 100.

Chương trình dùng một biến đếm i được khởi tạo từ 1, và một biến kq để chứa tổng. Mỗi bước lặp chương trình cộng i vào kq và sau đó tăng i lên 1 đơn vị. Chương trình còn lặp khi nào i còn chưa vượt qua 100. Khi i lớn hơn 100 chương trình dừng. Sau đây là văn bản chương trình.

```
void main()
{
    int i, kq = 0;
    for (i = 1 ; i <= 100 ; i++) kq += i ;
    cout << "Tổng = " << kq;
}
```

Ví dụ 3 : In ra màn hình dãy số lẻ bé hơn một số n nào đó được nhập vào từ bàn phím.

Chương trình dùng một biến đếm i được khởi tạo từ 1, mỗi bước lặp chương trình sẽ in i sau đó tăng i lên 2 đơn vị. Chương trình còn lặp khi nào i còn chưa vượt qua n. Khi i lớn hơn n chương trình dừng. Sau đây là văn bản chương trình.

```
void main()
{
    int n, i ;
```

```
cout << "Hãy nhập n = " ; cin >> n ;
for (i = 1 ; i < n ; i += 2) cout << i << '\n' ;
}
```

d. Đặc điểm

Thông qua phần giải thích cách hoạt động của câu lệnh for trong ví dụ 7 có thể thấy các thành phần của for có thể để trống, tuy nhiên các dấu chấm phẩy vẫn giữ lại để ngăn cách các thành phần với nhau. Ví dụ câu lệnh for (kq = 0 ; m ; m >= 1, n <= 1) if (m%2) kq += n ; trong ví dụ 7 có thể được viết lại như sau:

```
kq = 0;
for ( ; m ; ) { if (m%2) kq += n; m >= 1; n <= 1; }
```

Tương tự, câu lệnh for (i = 1 ; i <= 100 ; i++) kq += i ; trong ví dụ 8 cũng có thể được viết lại như sau:

```
i = 1;
for ( ; i <= 100 ; ) kq += i ++;
```

(câu lệnh kq += i++; được thực hiện theo 2 bước: cộng i vào kq và tăng i (tăng sau)).

Trong trường hợp điều kiện trong for cũng để trống chương trình sẽ ngầm định là điều kiện luôn luôn đúng, tức vòng lặp sẽ lặp vô hạn lần (!). Trong trường hợp này để dừng vòng lặp trong khối lệnh cần có câu lệnh kiểm tra dừng và câu lệnh break.

Ví dụ câu lệnh for (i = 1 ; i <= 100 ; i++) kq += i ; được viết lại như sau:

```
i = 1;
for ( ; ; )
{
    kq += i++;
    if (i > 100) break;
}
```

Tóm lại, việc sử dụng dạng viết nào của for phụ thuộc vào thói quen của NSD, tuy nhiên việc viết đầy đủ các thành phần của for làm cho việc đọc chương trình trở nên dễ dàng hơn.

e. Lệnh for lồng nhau

Trong dãy lệnh lặp có thể chứa cả lệnh for, tức các lệnh for cũng được phép lồng nhau như các câu lệnh có cấu trúc khác.

Ví dụ 4 : Bài toán cỗ: vừa gà vừa chó bó lại cho tròn đếm đủ 100 chân. Hỏi có mấy gà

và mấy con chó, biết tổng số con là 36.

Để giải bài toán này ta gọi g là số gà và c là số chó. Theo điều kiện bài toán ta thấy g có thể đi từ 0 (không có con nào) và đến tối đa là 50 (vì chỉ có 100 chân), tương tự c có thể đi từ 0 đến 25. Như vậy ta có thể cho g chạy từ 0 đến 50 và với mỗi giá trị cụ thể của g lại cho c chạy từ 0 đến 25, lần lượt với mỗi cặp (g, c) cụ thể đó ta kiểm tra 2 điều kiện: $g + c == 36$? (số con) và $2g + 4c == 100$? (số chân). Nếu cả 2 điều kiện đều thoả thì cặp (g, c) cụ thể đó chính là nghiệm cần tìm. Từ đó ta có chương trình với 2 vòng for lồng nhau, một vòng for cho g và một vòng cho c.

```
void main()
{
    int g, c ;
    for (g = 0 ; g <= 50 ; g++)
        for (c = 0 ; c <= 25 ; c++)
            if (g+c == 36 && 2*g+4*c == 100) cout << "gà=" << g << ", chó=" << c ;
}
```

Chương trình trên có thể được giải thích một cách ngắn gọn như sau: Đầu tiên cho $g = 0$, thực hiện lệnh for bên trong tức lần lượt cho $c = 0, 1, \dots, 25$, với $c=0$ và $g=0$ kiểm tra điều kiện, nếu thoả thì in kết quả nếu không thì bỏ qua, quay lại tăng c , cho đến khi nào $c>25$ thì kết thúc vòng lặp trong quay về vòng lặp ngoài tăng g lên 1, lại thực hiện vòng lặp trong với $g=1$ này (tức lại cho c chạy từ 0 đến 25). Khi g của vòng lặp ngoài vượt quá 50 thì dừng. Từ đó ta thấy số vòng lặp của chương trình là $50 \times 25 = 1000$ lần lặp.

Chú ý: Có thể giảm bớt số lần lặp bằng nhận xét số gà không thể vượt quá 36 (vì tổng số con là 36). Một vài nhận xét khác cũng có thể làm giảm số vòng lặp, tiết kiệm thời gian chạy của chương trình. Bạn đọc tự nghĩ thêm các phương án giải khác để giảm số vòng lặp đến ít nhất.

Ví dụ 5 : Tìm tất cả các phương án để có 100đ từ các tờ giấy bạc loại 10đ, 20đ và 50đ.

```
main()
{
    int t10, t20, t50;                                // số tờ 10đ, 20đ, 50đ
    sopa = 0;                                         // số phương án
    for (t10 = 0 ; t10 <= 10 ; t10++)
        for (t20 = 0 ; t20 <= 5 ; t20++)
            for (t50 = 0 ; t50 <= 2 ; t50++)
```

```
if (t10*10 + t20*20 + t50*50 == 100)           // nếu thỏa thì
{
    sopa++;
    if (t10) cout << t10 << "tờ 10đ " ;
    if (t20) cout << "+" << t20 << "tờ 20đ " ;
    if (t50) cout << "+" << t50 << "tờ 50đ " ;
    cout << '\n' ;
}
cout << "Tổng số phương án = " << sopa ;
}
```

2. Lệnh lặp while

a. Cú pháp

while (điều kiện) { khối lệnh lặp ; }

b. Thực hiện

Khi gặp lệnh while chương trình thực hiện như sau: đầu tiên chương trình sẽ kiểm tra điều kiện, nếu đúng thì thực hiện khối lệnh lặp, sau đó quay lại kiểm tra điều kiện và tiếp tục. Nếu điều kiện sai thì dừng vòng lặp. Tóm lại có thể mô tả một cách ngắn gọn về câu lệnh while như sau: *lặp lại các lệnh trong khi điều kiện vẫn còn đúng.*

c. Đặc điểm

- Khối lệnh lặp có thể không được thực hiện lần nào nếu điều kiện sai ngay từ đầu.
- Để vòng lặp không lặp vô hạn thì trong khối lệnh thông thường phải có ít nhất một câu lệnh nào đó gây ảnh hưởng đến kết quả của điều kiện, ví dụ làm cho điều kiện đang đúng trở thành sai.
- Nếu điều kiện luôn luôn nhận giá trị đúng (ví dụ biểu thức điều kiện là 1) thì trong khối lệnh lặp phải có câu lệnh kiểm tra dừng và lệnh break.

d. Ví dụ minh họa

Ví dụ 1 : Nhập 2 số nguyên theo phương pháp Ân độ

```
void main()
{
```

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

}

Trong chương trình trên câu lệnh while (m) ... được đọc là "trong khi m còn khác 0 thực hiện ...", ta thấy trong khối lệnh lặp có lệnh $m \geq 1$, lệnh này sẽ ảnh hưởng đến điều kiện (m), đến lúc nào đó m bằng 0 tức (m) là sai và chương trình sẽ dừng lặp.

Câu lệnh while (m) ... cũng có thể được thay bằng while (1) ... như sau:

```
void main()
{
    long m, n, kq; // Các số cần nhân và kết quả kq
    cout << "Nhập m và n: " ; cin >> m >> n ;
    kq = 0 ;
    while (1) {
        if (m%2) kq += n ;
        m >>= 1;
        n <<= 1;
        if (!m) break ; // nếu m = 0 thì thoát khỏi vòng lặp
    }
    cout << "m nhân n =" << kq ;
```

}

Ví dụ 2 : Bài toán cổ: vừa gà vừa chó bó lại cho tròn đếm đủ 100 chân. Hỏi có mấy gà và mấy con chó, biết tổng số con là 36.

```
void main()
```

```
{  
    int g, c ;  
    g = 0 ;  
    while (g <= 36) {  
        c = 0 ;  
        while (c <= 50) {  
            if (g + c == 36 && 2*g + 4*c == 100) cout << g << c ;  
            c++;  
        }  
        g++;  
    }  
}
```

Ví dụ 3 : Tìm ước chung lớn nhất (UCLN) của 2 số nguyên m và n.

Áp dụng thuật toán Euclide bằng cách liên tiếp lấy số lớn trừ đi số nhỏ khi nào 2 số bằng nhau thì đó là UCLN. Trong chương trình ta qui ước m là số lớn và n là số nhỏ. Thêm biến phụ r để tính hiệu của 2 số. Sau đó đặt lại m hoặc n bằng r sao cho m > n và lặp lại. Vòng lặp dừng khi m = n.

```
void main()  
{  
    int m, n, r;  
    cout << "Nhập m, n: " ; cin >> m >> n ;  
    if (m < n) { int t = m; m = n; n = t; } // nếu m < n thì đổi vai trò hai số  
    while (m != n) {  
        r = m - n ;  
        if (r > n) m = r; else { m = n ; n = r ; }  
    }  
    cout << "UCLN = " << m ;  
}
```

Ví dụ 4 : Tìm nghiệm xấp xỉ của phương trình $e^x - 1.5 = 0$, trên đoạn $[0, 1]$ với độ chính xác 10^{-6} bằng phương pháp chia đôi.

Để viết chương trình này chúng ta nhắc lại phương pháp chia đôi. Cho hàm $f(x)$ liên tục và đổi dấu trên một đoạn $[a, b]$ nào đó (tức $f(a), f(b)$ trái dấu nhau hay $f(a)*f(b)$

< 0). Ta đã biết với điều kiện này chắc chắn đồ thị của hàm $f(x)$ sẽ cắt trục hoành tại một điểm x_0 nào đó trong đoạn $[a, b]$, tức x_0 là nghiệm của phương trình $f(x) = 0$. Tuy nhiên việc tìm chính xác x_0 là khó, vì vậy ta có thể tìm xấp xỉ x' của nó sao cho x' càng gần x_0 càng tốt. Lấy c là điểm giữa của đoạn $[a, b]$, c sẽ chia đoạn $[a, b]$ thành 2 đoạn con $[a, c]$ và $[c, b]$ và do $f(a), f(b)$ trái dấu nên chắc chắn một trong hai đoạn con cũng phải trái dấu, tức nghiệm x_0 sẽ nằm trong đoạn này. Tiếp tục quá trình bằng cách chia đôi đoạn vừa tìm được ... cho đến khi ta nhận được một đoạn con (trái dấu, chứa x_0) sao cho độ dài của đoạn con này bé hơn độ xấp xỉ cho trước thì dừng. Khi đó lấy bất kỳ điểm nào trên đoạn con này (ví dụ hai điểm mút hoặc điểm giữa của a và b) thì chắc chắn khoảng cách của nó đến x_0 cũng bé hơn độ xấp xỉ cho trước, tức có thể lấy điểm này làm nghiệm xấp xỉ của phương trình $f(x) = 0$.

Trong ví dụ này hàm $f(x)$ chính là $e^x - 1.5$ và độ xấp xỉ là 10^{-6} . Đây là hàm liên tục trên toàn trực số và đổi dấu trên đoạn $[0, 1]$ (vì $f(0) = 1 - 1.5 < 0$ còn $f(1) = e - 1.5 > 0$). Sau đây là chương trình.

```
void main()
{
    float a = 0, b = 1, c;           // các điểm mút a, b và điểm giữa c
    float fa, fc;                  // giá trị của f(x) tại các điểm a, c
    while (b-a > 1.0e-6)          // trong khi độ dài đoạn còn lớn hơn ε
    {
        c = (a + b)/2;             // tìm điểm c giữa đoạn [a,b]
        fa = exp(a) - 1.5; fc = exp(c) - 1.5; // tính f(a) và f(c)
        if (fa*fc == 0) break;      // f(c) = 0 tức c là nghiệm
        if (fa*fc > 0) a = c; else b = c;
    }
    cout << "Nghiệm xấp xỉ của phương trình = " << c ;
}
```

Trong chương trình trên câu lệnh `if (fa*fc > 0) a = c; else b = c;` dùng để kiểm tra $f(a)$ và $f(c)$, nếu cùng dấu ($f(a)*f(c) > 0$) thì hàm $f(x)$ phải trái dấu trên đoạn con $[c, b]$ do đó đặt lại đoạn này là $[a, b]$ (để quay lại vòng lặp) tức đặt $a = c$ và b giữ nguyên, ngược lại nếu hàm $f(x)$ trái dấu trên đoạn con $[a, c]$ thì đặt lại $b = c$ còn a giữ nguyên. Sau đó vòng lặp quay lại kiểm tra độ dài đoạn $[a, b]$ (mới) nếu đã bé hơn độ xấp xỉ thì dừng và lấy c làm nghiệm xấp xỉ, nếu không thì tính lại c và tiếp tục quá trình.

Để tính $f(a)$ và $f(c)$ chương trình đã sử dụng hàm $\exp(x)$, đây là hàm cho lại kết quả e^x , để dùng hàm này hoặc các hàm toán học nói chung, cần khai báo file nguyên

mẫu math.h.

3. Lệnh lặp do ... while

a. Cú pháp

do { khối lệnh lặp } while (điều kiện) ;

b. Thực hiện

Đầu tiên chương trình sẽ thực hiện khối lệnh lặp, tiếp theo kiểm tra điều kiện, nếu điều kiện còn đúng thì quay lại thực hiện khối lệnh và quá trình tiếp tục cho đến khi điều kiện trở thành sai thì dừng.

c. Đặc điểm

Các đặc điểm của câu lệnh do ... while cũng giống với câu lệnh lặp while trừ điểm khác biệt, đó là khối lệnh trong do ... while sẽ được thực hiện ít nhất một lần, trong khi trong câu lệnh while có thể không được thực hiện lần nào (vì lệnh while phải kiểm tra điều kiện trước khi thực hiện khối lệnh, do đó nếu điều kiện sai ngay từ đầu thì lệnh sẽ dừng, khối lệnh không được thực hiện lần nào. Trong khi đó lệnh do ... while sẽ thực hiện khối lệnh rồi mới kiểm tra điều kiện lặp để cho phép thực hiện tiếp hoặc dừng).

d. Ví dụ minh họa

Ví dụ 1 : Tính xấp xỉ số pi theo công thức Euler $\frac{\pi^2}{6} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \dots + \frac{1}{n^2}$, với $\frac{1}{n^2} < 10^{-6}$.

```
void main()
{
    int n = 1; float S = 0;
    do S += 1.0/(n*n) while 1.0/(n*n) < 1.0e-6;
    float pi = sqrt(6*S);
    cout << "pi = " << pi ;
}
```

Ví dụ 2 : Kiểm tra một số n có là số nguyên tố.

Để kiểm tra một số n > 3 có phải là số nguyên tố ta lần lượt chia n cho các số i đi

từ 2 đến một nửa của n. Nếu có i sao cho n chia hết cho i thì n là hợp số ngược lại n là số nguyên tố.

```
void main()
{
    int i, n;                                // n: số cần kiểm tra
    cout << "Cho biết số cần kiểm tra: " ; cin >> n ;
    i = 2 ;
    do {
        if (n%i == 0) {
            cout << n << "là hợp số" ;
            return ;                            // dừng chương trình
        }
        i++;
    } while (i <= n/2);
    cout << n << "là số nguyên tố" ;
}
```

Ví dụ 3 : Nhập dãy kí tự và thống kê các loại chữ hoa, thường, chữ số và các loại khác còn lại đến khi gặp ENTER thì dừng.

```
void main()
{
    char c;                                  // kí tự dùng cho nhập
    int n1, n2, n3, n4 ;                    // số lượng các loại kí tự
    n1 = n2 = n3 = n4 = 0;
    cout << "Hãy nhập dãy kí tự: \n" ;
    do
    {
        cin >> c;
        if ('a' <= c && c <= 'z') n1++;      // nếu c là chữ thường thì tăng n1
        else if ('A' <= c && c <= 'Z') n2++;  // chữ hoa, tăng n2
        else if ('0' <= c && c <= '9') n3++;  // chữ số, tăng n3
        else n4++;                           // loại khác, tăng n4
    }
```

```
cout << n1 << n2 << n3 << n4 ;      // in kết quả  
} while (c != 10) ;                      // còn lặp khi c còn khác kí tự ↴  
}
```

4. Lối ra của vòng lặp: break, continue

a. Lệnh break

Công dụng của lệnh dùng để thoát ra khỏi (chấm dứt) các câu lệnh cấu trúc, chương trình sẽ tiếp tục thực hiện các câu lệnh tiếp sau câu lệnh vừa thoát. Các ví dụ minh họa bạn đọc có thể xem lại trong các ví dụ về câu lệnh switch, for, while.

b. Lệnh continue

Lệnh dùng để quay lại đầu vòng lặp mà không chờ thực hiện hết các lệnh trong khối lệnh lặp.

Ví dụ 1 : Giả sử với mỗi i từ 1 đến 100 ta cần thực hiện một loạt các lệnh nào đó trừ những số i là số chính phương. Như vậy để tiết kiệm thời gian, vòng lặp sẽ kiểm tra nếu i là số chính phương thì sẽ quay lại ngay từ đầu để thực hiện với i tiếp theo.

```
int i ;  
for (i = 1; i <= 100; i++) {  
    if (i là số chính phương) continue;  
    {  
        // dãy lệnh khác  
        .  
        .  
    }  
}
```

(Để kiểm tra i có là số chính phương chúng ta so sánh căn bậc hai của i với phần nguyên của nó. Nếu hai số này bằng nhau thì i là số chính phương. Cụ thể nếu $\sqrt{i} = \text{int}(\sqrt{i})$ thì i là số chính phương. Ở đây \sqrt{x} là hàm trả lại căn bậc hai của x. Để sử dụng hàm này cần phải khai báo file nguyên mẫu math.h.)

5. So sánh cách dùng các câu lệnh lặp

Thông qua các ví dụ đã trình bày bạn đọc có thể thấy rằng về mặt thực chất để tổ chức một vòng lặp chúng ta có thể chọn một trong các câu lệnh goto, for, while, do ... while, có nghĩa là mặt khả năng thực hiện các câu lệnh này là như nhau. Tuy nhiên,

trong một ngữ cảnh cụ thể việc sử dụng câu lệnh phù hợp trong chúng làm cho chương trình sáng sủa, rõ ràng và tăng độ tin cậy lên cao hơn. Theo thói quen lập trình trong một số ngôn ngữ có trước và dựa trên đặc trưng riêng của từng câu lệnh, các lệnh lặp thường được dùng trong các ngữ cảnh cụ thể như sau:

- FOR thường được sử dụng trong những vòng lặp mà số lần lặp được biết trước, nghĩa là vòng lặp thường được tổ chức dưới dạng một (hoặc nhiều) biến đếm chạy từ một giá trị nào đó và đến khi đạt được đến một giá trị khác cho trước thì dừng. Ví dụ dạng thường dùng của câu lệnh for là như sau:
 - `for (i = gt1 ; i <= gt2 ; i++)` ... tức i tăng từ gt1 đến gt2 hoặc
 - `for (i = gt2 ; i >= gt1 ; i--)` ... tức i giảm từ gt2 xuống gt1
- Ngược lại với FOR, WHILE và DO ... WHILE thường dùng trong các vòng lặp mà số lần lặp không biết trước, chúng thường được sử dụng khi việc lặp hay dừng phụ thuộc vào một biểu thức logic.
- WHILE được sử dụng khi khả năng thực hiện khối lặp không xảy ra lần nào, tức nếu điều kiện lặp có giá trị sai ngay từ đầu, trong khi đó DO ... WHILE được sử dụng khi ta biết chắc chắn khối lệnh lặp phải được thực hiện ít nhất một lần.

BÀI TẬP

Lệnh rẽ nhánh

1. Nhập một kí tự. Cho biết kí tự đó có phải là chữ cái hay không.
2. Nhập vào một số nguyên. Trả lời số nguyên đó: âm hay dương, chẵn hay lẻ ?
3. Cho $n = x = y$ và bằng: a. 1 b. 2 c. 3 d. 4

Hãy cho biết giá trị của x , y sau khi chạy xong câu lệnh:

```
if (n % 2 == 0) if (x > 3) x = 0;  
else y = 0;
```

4. Tính giá trị hàm

a. $f(x) = \begin{cases} 3x + \sqrt{x} & , x > 0 \\ e^x + 4 & , x \leq 0 \end{cases}$

b. $f(x) = \begin{cases} \sqrt{x^2 + 1} & , x \geq 1 \\ 3x + 5 & , -1 < x < 1 \\ x^2 + 2x - 1 & , x \leq -1 \end{cases}$

5. Viết chương trình giải hệ phương trình bậc nhất 2 ẩn: $\begin{cases} ax + by = c \\ dx + ey = f \end{cases}$
6. Nhập 2 số a, b. In ra max, min của 2 số đó. Mở rộng với 3 số, 4 số ?
7. Nhập 3 số a, b, c. Hãy cho biết 3 số trên có thể là độ dài 3 cạnh của một tam giác ? Nếu là một tam giác thì đó là tam giác gì: vuông, đều, cân, vuông cân hay tam giác thường ?
8. Nhập vào một số, in ra thứ tương ứng với số đó (qui ước 2 là thứ hai, ..., 8 là chủ nhật).
9. Nhập 2 số biểu thị tháng và năm. In ra số ngày của tháng năm đó (có kiểm tra năm nhuận).
10. Lấy ngày tháng hiện tại làm chuẩn. Hãy nhập một ngày bất kỳ trong tháng. Cho biết thứ của ngày vừa nhập ?

Lệnh lặp

11. Giá trị của i bằng bao nhiêu sau khi thực hiện cấu trúc for sau:

for (*i* = 0; *i* < 100; *i*++);

12. Giá trị của x bằng bao nhiêu sau khi thực hiện cấu trúc for sau:

for (*x* = 2; *i* < 10; *x*+=3) ;

13. Bạn bổ sung gì vào lệnh for sau:

for (; *nam* < 1997 ;) ;

để khi kết thúc nam có giá trị 2000.

14. Bao nhiêu ký tự ‘X’ được in ra màn hình khi thực hiện đoạn chương trình sau:

for (*x* = 0; *x* < 10; *x*++) for (*y* = 5; *y* > 0; *y*--) cout << ‘X’;

15. Nhập vào tuổi cha và tuổi con hiện nay sao cho tuổi cha lớn hơn 2 lần tuổi con. Tìm xem bao nhiêu năm nữa tuổi cha sẽ bằng đúng 2 lần tuổi con (ví dụ 30 và 12, sau 6 năm nữa tuổi cha là 36 gấp đôi tuổi con là 18).

16. Nhập số nguyên dương N. Tính:

$$a. \quad S_1 = \frac{1+2+3+\dots+N}{N}$$

$$b. \quad S_2 = \sqrt{1^2 + 2^2 + 3^2 + \dots + N^2}$$

17. Nhập số nguyên dương n. Tính:

$$a. \quad S_1 = \sqrt{3 + \sqrt{3 + \sqrt{3 + \dots + \sqrt{3}}}} \quad n \text{ dấu căn}$$

$$b. \quad S_2 = \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{\dots + \cfrac{1}{2}}}}} \quad n \text{ dấu chia}$$

18. Nhập số tự nhiên n. In ra màn hình biểu diễn của n ở dạng nhị phân.

19. In ra màn hình các số có 2 chữ số sao cho tích của 2 chữ số này bằng 2 lần tổng của 2 chữ số đó (ví dụ số 36 có tích $3*6 = 18$ gấp 2 lần tổng của nó là $3 + 6 = 9$).

20. Số *hoàn chỉnh* là số bằng tổng mọi ước của nó (không kể chính nó). Ví dụ $6 = 1 + 2 + 3$ là một số hoàn chỉnh. Hãy in ra màn hình tất cả các số hoàn chỉnh < 1000 .

21. Các số *sinh đôi* là các số nguyên tố mà khoảng cách giữa chúng là 2. Hãy in tất cả cặp số sinh đôi < 1000 .

22. Nhập dãy kí tự đến khi gặp kí tự ‘.’ thì dừng. Thống kê số chữ cái viết hoa, viết thường, số chữ số và tổng số các kí tự khác đã nhập. Loại kí tự nào nhiều nhất?
23. Tìm số nguyên dương n lớn nhất thoả mãn điều kiện:
- $1 + \frac{1}{3} + \frac{1}{5} + \dots + \frac{1}{2n-1} < 2.101999$.
 - $e^n - 1999 \log_{10} n < 2000$.
24. Cho $\varepsilon = 1e-6$. Tính gần đúng các số sau:
- Số pi theo công thức Euler: $\frac{\pi^2}{6} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \dots + \frac{1}{n^2}$ dùng lặp khi $\frac{1}{n^2} < 10^{-6}$.
 - e^x theo công thức: $e^x = 1 + \frac{x^1}{1!} + \frac{x^2}{2!} + \dots + \frac{x^n}{n!}$ dùng lặp khi $\left| \frac{x^n}{n!} \right| < 10^{-6}$.
 - $\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \dots + (-1)^n \frac{x^{2n+1}}{(2n+1)!}$, dùng lặp khi $\left| \frac{x^{2n+1}}{(2n+1)!} \right| < 10^{-6}$.
 - \sqrt{a} ($a > 0$) theo công thức: $s_n = \begin{cases} a & n=0 \\ (s_{n-1}^2 + a)/2s_{n-1} & n>0 \end{cases}$, dùng khi $|s_n - s_{n-1}| < 10^{-6}$.
25. In ra mã của phím bất kỳ được nhấn. Chương trình lặp cho đến khi nhấn ESC để thoát.
26. Bằng phương pháp chia đôi, hãy tìm nghiệm xấp xỉ ($\text{độ chính xác } 10^{-6}$) của các phương trình sau:
- $e^x - 1.5 = 0$, trên đoạn $[0, 1]$.
 - $x2^x - 1 = 0$, trên đoạn $[0, 1]$.
 - $a_0x^n + a_1x^{n-1} + \dots + a_n = 0$, trên đoạn $[a, b]$. Các số thực a_i, a, b được nhập từ bàn phím sao cho $f(a) \neq f(b)$ trái dấu.