BÀI 2
CÁC CẤU TRÚC ĐIỀU KHIỂN

I. Cấu trúc rẽ nhánh và cấu trúc chọn

I.1. Cấu trúc rẽ nhánh

Trong thực tế, khi giải quyết một công việc thường ta phải lựa chọn nhiều phương án giải quyết khác nhau. Người ta thường biểu diễn vấn đề này bằng 2 mệnh đề logic sau:

- [1]. **Nếu … thì …;**
- [2]. **Nếu … thì … ngược lại thì …**

Để mô phỏng hai mệnh đề đó, trong ngôn ngữ lập trình C++ đưa ra cấu trúc rẽ nhánh.

Cú pháp:

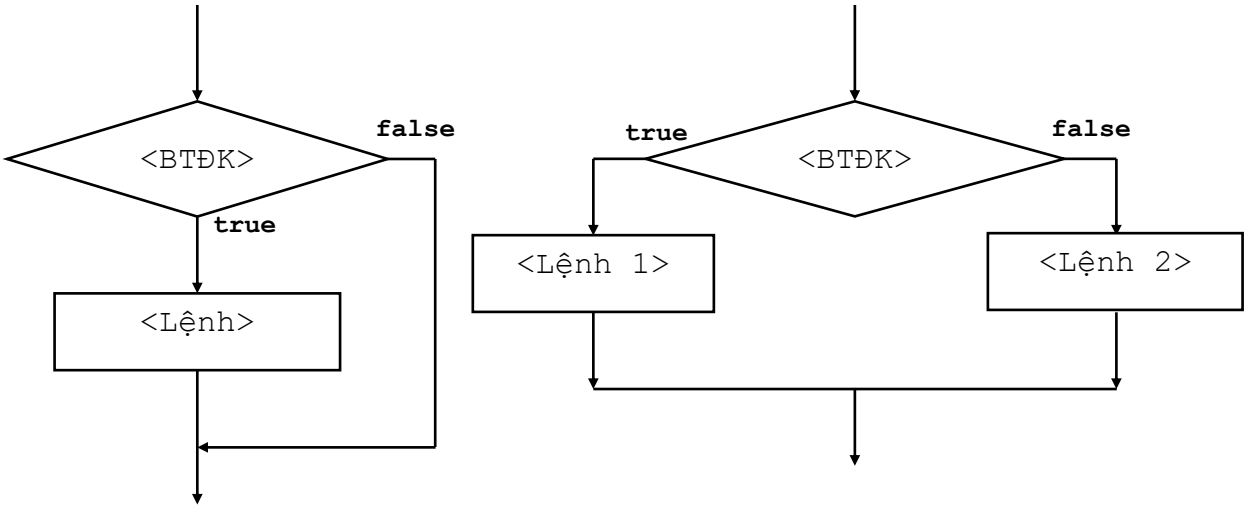
```
if (<biểu thức điều kiện>) <Lệnh 1>;
[else <Lệnh 2>;]
```

Ý nghĩa:

Nếu <biểu thức điều kiện> nhận giá trị đúng (**true**), sẽ thực hiện <Lệnh 1>, ngược lại, nếu <biểu thức điều kiện> nhận giá trị sai (**false**) sẽ thực hiện <Lệnh 2>; (nếu có thành phần **[else <Lệnh 2>;])**.

- <Lệnh 1> và <Lệnh 2> có thể là một lệnh, một khối lệnh hoặc một, một khối các cấu trúc điều khiển. Các khối lệnh hoặc khối cấu trúc điều khiển được đặt trong hai dấu { }.

Cấu trúc rẽ nhánh có hai dạng (tuỳ thuộc vào sự có hay không có thành phần **[else <Lệnh 2>;])** như trong sơ đồ khối dưới đây.



a). Mô tả mệnh đề [1] *b) Mô tả mệnh đề [2]*

Ví dụ: Lập chương trình nhập vào một số nguyên. Kiểm tra tính chẵn lẻ của số đó và thông báo ra màn hình.

```
#include <conio.h>
#include <stdio.h>
#include <iostream>
using namespace std;
int main()
{
        int a;
        cout<< “a =  ”;
        cin>>a;
        if (a%2 = = 0)
                cout<<”so “<<a<<” chan”;
        else
                cout<<”so “<<a<<” le”;
}
```

Các lệnh if có thể lồng nhau theo nghĩa: Các câu lệnh bên trong một mệnh đề if lại có thể là các mệnh đề if.

Mỗi lệnh *if* đầy đủ sẽ cho phép lựa chọn 2 khả năng để thực hiện. Trong trường hợp có n khả năng lựa chọn và các khả năng loại trừ nhau, ta có thể sử dụng n-1 lệnh *if* đầy đủ lồng nhau.

Ví dụ: Viết chương trình thực hiện việc nhập vào số tiền phải trả của khách hàng. Nếu số tiền nhập vào từ 300 tới 400, khuyến mại 20% số tiền phải trả. Nếu số tiền từ 400 trở lên, khuyến mại 30%. Các trường hợp khác không được khuyến mại. Tính và in số tiền khuyến mại của khách lên màn hình.

```
#include <conio.h>
#include <stdio.h>
#include <iostream>
using namespace std;
int   main()
{
        int T, km;
        cout<<" Nhap so tien "; cin>>T;
        if (T>=300 && T <=400)      km = T*0.2;
        else
                if (T>400) km = T*0.3;
        else    km = 0;

        cout<<"So tien khuyen mai "<<km;
}
```

Nếu n khả năng là loại trừ nhau thì khi đó có thể sử dụng n-1 lệnh if lồng nhau hoặc có thể sử dụng n lệnh if rời nhau cho n khả năng lựa chọn. Trường hợp ngược lại, ta nên sử dụng các lệnh if rời nhau.

Ví dụ: Viết chương trình nhập vào điểm tổng kết và xếp loại đạo đức của một sinh viên. Sau đó tính số tiền học bổng cho sinh viên đó như sau:

Nếu tổng kết >=7.00 thì được 300000.

Nếu điểm tổng kết >=9.00 và đạo đức = "T" thì được cộng thêm 100000.

Xét đoạn trình sau:

```
#include <conio.h>
#include <stdio.h>
#include <iostream>
using namespace std;

int main()
{
        float tk; char hk; long T;
        cout<<"Nhap diem tong ket: ";    cin>>tk;
        cout<<"Nhap hanh kiem: ";        cin>>hk;
        T=0;
        if (tk >= 7)      T = 300000;
        else  if (tk>=9 && hk = = 'T')     T += 100000;
        cout<<"Học bong: " <<T;
}
```

Đoạn trình trên sẽ cho kết quả sai trong trường hợp sinh viên tổng kết >=9.0 và đạo đức tốt. Lý do là sử dụng hai lệnh *if* lồng nhau khi các khả năng không loại trừ nhau.

Đoạn trình trên có thể được viết lại như sau:

```
#include <conio.h>
#include <stdio.h>
#include <iostream>
using namespace std;

int main()
{
        float tk; char  hk; long T;
        cout<<"Nhap diem tong ket: ";    cin>>tk;
        cout<<"Nhap hanh kiem: ";        cin>>hk;
        T=0;
        if (tk >= 7)      T = 300000;
        if (tk>=9 && hk = = 'T')     T += 100000;
        cout<<"Hoc bong " <<T;
 }
```

I.2. Cấu trúc chọn

Trong trường hợp có quá nhiều khả năng lựa chọn và các khả năng loại trừ nhau, nếu sử dụng nhiều lệnh *if* lồng nhau sẽ làm cho chương trình phức tạp, khó kiểm soát. Vì vậy C++ cung cấp một cấu trúc điều khiển khác sử dụng trong trường hợp này, đó là cấu trúc chọn.

Cú pháp:

```
switch (⟨Biến nguyên⟩)
{
        case <GT 1>:  <Lệnh 1;>  break;
        case <GT 2>:  <Lệnh 2;>     break;
        …
        case <GT n>:  <Lệnh n;>     break;
        [default:          <Lệnh mặc định>;]
}
```

Ý nghĩa: Kiểm tra giá trị của ⟨Biến nguyên⟩:

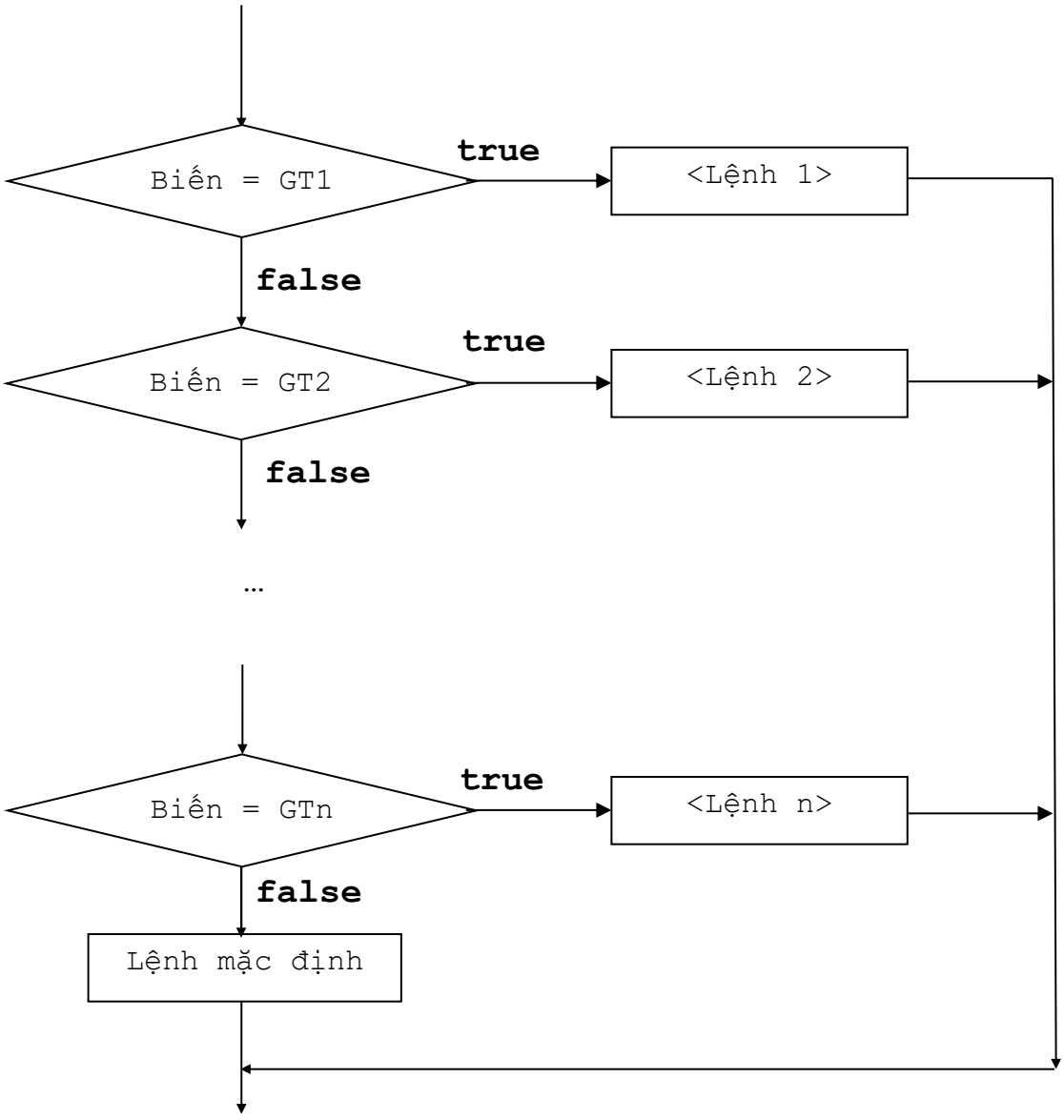
Nếu ⟨Biến nguyên⟩ nhận giá trị <GT 1>, thực hiện <Lệnh 1>

Nếu ⟨Biến nguyên⟩nhận giá trị <GT 2>, thực hiện <Lệnh 2>…

Nếu ⟨Biến nguyên⟩ nhận giá trị <GT n>, thực hiện <Lệnh n>

Nếu có thành phần [default: …], thực hiện <lệnh mặc định> khi biến nguyên không nhận giá trị nào trong n giá trị ở trên.

Sơ đồ khối:



- Lệnh ***switch*** chỉ thực hiện trên biến nguyên hoặc tương đương (cí dụ biến kiểu char). Các câu lệnh <Lệnh 1>, <Lệnh 2>… có thể là một khối lệnh hoặc khối các cấu trúc điều khiển (tức nhiều lệnh, nhiều cấu trúc điều khiển đặt giữa hai ký tự { và }). Sau đó phải có lệnh break; nếu không, sau khi thực hiện lệnh tương ứng của một case, lệnh switch sẽ thực hiện tới case tiếp theo và sẽ sai ý nghĩa của nó.

- Thành phần [default: …] là không bắt buộc. Nếu có thành phần này, <Lệnh mặc định> sẽ được được thực hiện sau khi tất cả các trường hợp *case* đều không thỏa mãn.

Ví dụ 1: Viết chương trình nhập vào mã học vị (là một số nguyên) của một nhân viên. In ra học vị tương ứng với quy định:

Mã =1: Cử nhân.

Mã =2: Kỹ sư.

Mã =3: Thạc sỹ.

Mã =4: Tiến sỹ.

Các mã khác: Không xếp loại học vị.

```
int main()
{
   int M;
   cout<<"Nhap ma hoc vi: ";        cin>>M;
   switch (M)
   {
   case 1:      cout<<"Cu nhan";          break;
   case 2:      cout<<"Ky su";                  break;
   case 3:      cout<<"Thac sy";          break;
   case 4:      cout<<"Tien sy";          break;
   default:     cout<<"Khong xep loai hoc vi";
   }
 }
```

Ví dụ 2: Viết chương trình nhập vào một tháng của một năm nào đó. In số ngày của tháng đó ra màn hình.

Để đơn giản, ta tạm thời coi tháng 2 sẽ có 28 ngày, trừ các năm nhuận nó sẽ có 29 ngày. Năm nhuận được xác định là các năm chia hết 4 (điều này chỉ là tạm thời quy định như vậy, để tính chính xác năm nhuận, sẽ phức tạp hơn).

```
#include <conio.h>
#include <stdio.h>
#include <iostream>
 using namespace std;
int main()
{
   int T, N;
   cout<<"Nhập tháng"; cin>>T;
   cout<<"Nhập năm ";  cin>>N;
   switch (T)
   {
          case 1:
```

```
            case 3:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12:      cout<<"Thang co 31 ngay"; break;
            case 4:
            case 6:
            case 9:
            case 11:      cout<<"Thang co 30 ngay";        break;
            case 2: if (N%4 ! = 0)   cout<<"Thang co 28 ngay";
                          else              cout<<"Thang co 29 ngay";
                          break;
            default:      cout<<"Thang khong hop le";
        }
    }
```

Chuyển đổi giữa cấu trúc chọn và rẽ nhánh:

Với cấu trúc rẽ nhánh, các biến trong biểu thức điều kiện có thể có kiểu bất kỳ. Ngược lại, với cấu trúc chọn, chỉ lựa chọn các trường hợp của biến nguyên. Do vậy, việc chuyển đổi từ cấu trúc chọn sang cấu trúc rẽ nhánh bao giờ cũng thực hiện được một cách dễ dàng, điều ngược lại không đúng.

Để chuyển đổi một cấu trúc rẽ nhánh mà biểu thức điều kiện có các biến không phải kiểu nguyên sang cấu trúc chọn cần sử dụng thêm một biến nguyên để mã hoá các trường hợp của nó, sau đó ta áp dụng cấu trúc chọn trên biến nguyên này.

II. Cấu trúc lặp

II.1. Vòng lặp với số lần lặp xác định

Giả sử cần thực hiện một vòng lặp với n lần lặp for(int i=0; i<n; i++). Khi thực hiện lệnh lặp này, máy tính phải thực hiện tuần tự các công việc sau: (1) Gán i:=0; (2) Kiểm tra xem i có còn nhỏ hơn n hay không, tức kiểm tra giá trị của biểu thức: i < n; và nếu i < n, (3) Tăng giá trị của i lên 1 đơn vị sau mỗi lần lặp: i:=i+1;

Như vậy, dễ thấy máy tính cần thực hiện 3 biểu thức của vòng lặp: i:=0, i<n, i++;. Mỗi vòng lặp, ta cần phải xác định cho được 3 biểu thức này. Ta tạm gọi chúng là các biểu thức <BT1>, <BT2>, <BT3>.

Trong C++, vòng lặp xác định được viết với 3 biểu thức dạng như trên. Cú pháp như sau:

```
for (<BT1>; <BT2>; <BT3)
    <Lệnh lặp>;
```

Trong đó, <BT1> thường nhận nhiệm vụ khởi gán giá trị ban đầu cho biến điều khiển (hay còn gọi là biến chạy). <BT2> là một biểu thức logic được dùng làm điều kiện dừng cho vòng lặp. Vòng lặp sẽ dừng khi <BT2> nhận giá trị sai (**false**); <BT3> được dùng để thay đổi giá trị của biến chạy sau mỗi lần lặp.

<Lệnh lặp> có thể là một lệnh, một khối lệnh hoặc một, một khối các cấu trúc điều khiển.

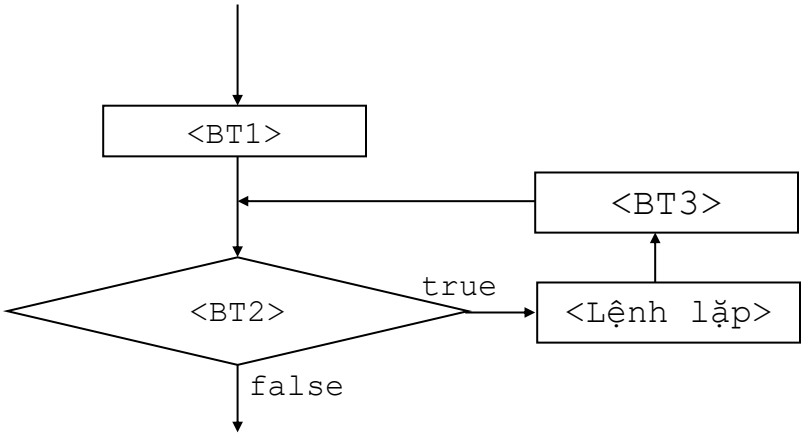
Quy trình thực hiện:

B1: Thực hiện <BT1>

B2: Kiểm tra <BT2>. Nếu sai, thoát khỏi vòng for và chuyển tới lệnh tiếp theo sau for (nếu có). Ngược lại, sang B3.

B3: Thực hiện <Lệnh lặp>, thực hiện <BT3>, quay lại B2.

Sơ đồ khối:



Các trường hợp đặc biệt của vòng lặp for

- **Trường hợp 1:** Các biểu thức <BT1>, <BT2>, <BT3> có thể khuyết nhưng các dấu ";" phải được giữ nguyên.

Ví dụ: Viết chương trình tính n! (n nguyên)

Cách 1: Sử dụng vòng lặp for thông thường

```
int main()
{
        int n;   long GT=1;
        cout<<"Nhap n ";              cin>>n;
        for (int i=2;i<=n; i++)GT*=i;
                cout<<n<<" Giai thua : "<<GT;
        return 0;
}
```

Cách 2: Vòng for khuyết <BT1>.

```
int main()
{
        int n; long GT=1;
        cout<<"Nhap n "; cin>>n;
        int i=2;
        for (;i<=n; i++)         GT*=i;
                cout<<n<<" Giai thua : "<<GT;
        return 0;
}
```

Cách 3: Vòng for khuyết <BT1> và <BT3>

```
int main()
{
        int n; long GT=1;
        cout<<"Nhap n "; cin>>n;
        int i=2;
        for (; i<=n; )
        {
                GT*=i;
                i++;
        }
        cout<<n<<" Giai thua : "<<GT;
        return 0;
}
```

Cách 4: Vòng for khuyết cả 3 biểu thức. Trong trường hợp này, vòng for không thể dừng một cách tự nhiên được (do thiếu điều kiện dừng là <BT2>). Khi đó ta cần thoát khỏi vòng lặp một cách có chủ định.

Các cách thoát khỏi vòng lặp for khi thiếu <BT2>

+ Cách 1: sử dụng lệnh break;

Khi gặp lệnh break; trong thân vòng for, chương trình sẽ lập tức thoát khỏi vòng lặp for và chuyển tới lệnh tiếp theo bất kể <Biểu thức 2> vẫn nhận giá trị đúng hoặc khuyết <BT2>.

+ Cách 2: sử dụng lệnh goto;

Lệnh goto có dạng: goto <Nhãn>; . Trong đó,<Nhãn> có dạng:

<Tên nhãn> :

<Tên nhãn> tuỳ ý đặt theo quy ước đặt tên trong C.

Khi gặp lệnh goto <Nhãn>;, chương trình sẽ nhảy tới vị trí đặt nhãn. Nếu nhãn đặt sau vòng for, chương trình sẽ thoát khỏi vòng for.

Cần chú ý trong trường hợp 2 lệnh for lồng nhau, khi đó lệnh break chỉ làm cho chương trình thoát khỏi vòng for gần nhất chứa lệnh break. Do vậy, để thoát khỏi cả 2 vòng for lồng nhau, ta có thể sử dụng lệnh goto.

Thoát khỏi for sử dụng break:

```
int main()
{
        int n; long GT=1;
        cout<<"Nhap n "; cin>>n;
        int i=2;
        for (; ;)
        {
                if (i==n) break;
                GT*=i;
                i++;
        }
        cout<<n<<" Giai thua : "<<GT;
        return 0;
}
```

Thoát khỏi for, sử dụng goto:

```
int main()
{
	int n; long GT=1;
	cout<<"Nhap n "; cin>>n;
	int i=2;
	for (; ;)
	{
		if (i==n) goto Ex; // nhãn là Ex, tên tự đặt.
		GT*=i;
		i++;
	}
	Ex:
	cout<<n<<" Giai thua : "<<GT;
	return 0;
}
```

Vì độ "rắc rối" của chương trình (đối với chương trình dịch) tỷ lệ thuận với số lệnh goto sử dụng trong chương trình, vì vậy nên hạn chế sử dụng goto.

- **Trường hợp 2:** Các <BT1>, <BT3> có thể là các biểu thức phức hợp (tức là gồm nhiều biểu thức con). Khi đó, các biểu thức con được đặt cách nhau bởi dấu phảy.

Ví dụ 2: Cho dãy số nguyên x[] = { 1, 4, 5, 7, 3, 2};. Viết chương trình đảo ngược dãy số trên và in kết quả lên màn hình.

Để giải quyết bài toán trên, có thể có nhiều cách. Cách giải sau minh hoạ cách viết khác của vòng for với <BT1> và <BT3> là các biểu thức phức hợp.

```
#include "conio.h"
#include "iostream"
#include "stdio.h"
using namespace std;
int main()
{
	int x[ ] = {1, 4, 5, 7, 3, 2}, n;
	n=sizeof(x)/ sizeof(int); //n là số phần tử của x (6)
	for (int i=0, j=n-1; i<n/2; i++, j--)
	{
		int tg = x[i];
```

```
            x[i]=x[j];
            x[j]=tg;
        }
        for (int i=0; i<n; i ++)
            cout<<x[i]<<"   ";
    }
```

Thậm chí, có thể viết lại lời giải trên bằng cách sử dụng vòng for với các lệnh thân vòng for được đưa vào <BT3> như sau:

```
int main()
{
    int x[ ] = {1, 4, 5, 7, 3, 2}, n;
    n=sizeof(x)/ sizeof(int);
    int tg;

    for (int i=0, j=n-1; i<n/2; tg=x[i], x[i]=x[j], x[j]=tg, i++, j--);

    for (int i=0; i<n; i ++)
        cout<<x[i]<<"   ";
    return 0;
}
```

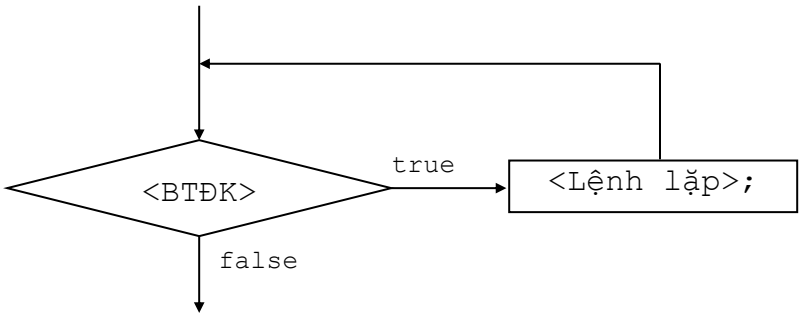
II.2. Vòng lặp với số lần lặp không xác định

Trong C++, ta chia vòng lặp với số lần lặp không xác định ra làm hai loại:

a. Lặp kiểm tra điều kiện trước:

Trước tiên, kiểm tra biểu thức điều kiện. Nếu biểu thức điều kiện còn đúng, sẽ thực hiện lệnh lặp. Nếu biểu thức điều kiện sai, ra khỏi vòng lặp.

Sơ đồ khối:



Như vậy: Lệnh lặp có thể không được thực hiện lần nào nếu <BTĐK> sai ngay từ đầu hoặc cũng có thể lặp vô hạn nếu <BTĐK> luôn đúng.

Cú pháp:

```
while (<BTĐK>)
        <Lệnh lặp>;
```

- <Lệnh lặp> có thể là một lệnh, một khối lệnh hoặc một, một khối cấu trúc điều khiển.

Ý nghĩa:

B1: Kiểm tra biểu thức điều kiện <BTĐK>. Nếu biểu thức điều kiện sai, thoát ra khỏi vòng lặp và chuyển tới lệnh tiếp theo sau vòng lặp (nếu có). Nếu biểu thức điều kiện đúng, chuyển qua bước 2.

B2: Thực hiện <Lệnh lặp>. Quay lại B1.

Ví dụ 1. Viết chương trình tìm số lũy thừa 2 đầu tiên lớn hơn 1000.

```
#include <conio.h>
#include <stdio.h>
#include <iostream>
using namespace std;
int main()
{
        int So=1;
        while (So <=1000)
                So *=2;
        cout<<"So can tim la: "<<So;
}
```

Ví dụ 2. Viết chương trình nhập vào điểm đạo đức của một học sinh (thang điểm 100, nguyên) và số ngày đi học muộn của học sinh đó. Nếu học sinh đi học muộn 1 ngày, trừ 1 điểm. Tính và in số điểm còn lại của học sinh.

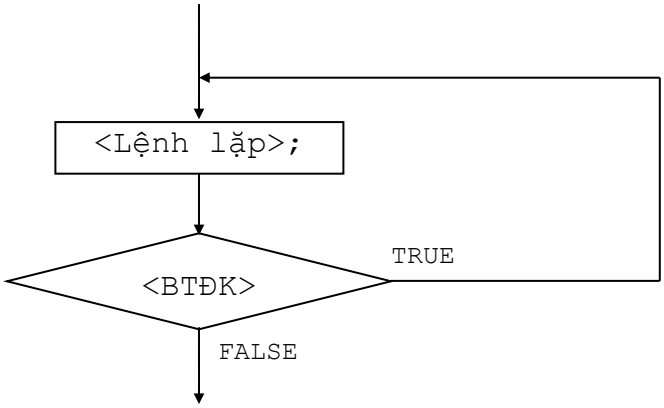
```
#include <conio.h>
#include <stdio.h>
#include <iostream>
using namespace std;
int main()
{
```

```
        int D, HM;
        cout<<"Nhap diem dao duc: ";          cin>>D;
        cout<<"Nhap so ngay di hoc muon: ";   cin>>HM;
        while (HM>0)
        {
            D - -;
            HM - -;
        }
        cout<<"Diem con lai "<<D;
}
```

b. Lặp kiểm tra điều kiện sau:

Tương tự như vòng lặp kiểm tra điều kiện trước, vòng lặp kiểm tra điều kiện sau chỉ khác ở chỗ biểu thức điều kiện được kiểm tra mỗi khi đã thực hiện lệnh lặp. Như vậy, lệnh lặp luôn được thực hiện ít nhất một lần.

Sơ đồ khối:



Cú pháp:

```
do
    <Lệnh lặp>;
while (<BTĐK>);
```

- <Lệnh lặp> có thể là một lệnh, một khối lệnh hoặc một, một khối cấu trúc điều khiển.

Ý nghĩa:

B1: Thực hiện <Lệnh lặp>;

B2: Kiểm tra biểu thức điều kiện <BTĐK>. Nếu biểu thức điều kiện sai, thoát ra khỏi vòng lặp và chuyển tới lệnh tiếp theo sau vòng lặp (nếu có). Nếu biểu thức điều kiện đúng, quay lại bước 1.

Ví dụ 1: Viết chương trình tìm số nguyên x đầu tiên (nhỏ nhất) lớn hơn 5 mà thỏa mãn sin(x) = 1

```
int main()
{
        int x=0;
        do
        {
                x +=1;
        }
        while (x <=5 | | sin(x) !=1);
        cout<<"So can tim la: "<<x;
}
```

Ví dụ 2: Viết chương trình nhập vào một số nguyên x. Kiểm tra xem số đó đã lớn hơn 10 hay chưa. Nếu chưa, yêu cầu nhập lại cho tới khi số nhập vào lớn hơn 10. Kiểm tra xem số đó có phải là số nguyên tố không, in kết luận lên màn hình.

```
int main()
{
    int So;
    do
    {
        cout<<"Nhap mot so nguyen: ";  cin>>So;
        if (So <=10)  cout<<"So khong hop le. Hay nhap lai !";
    }
    while (So <=10);
    //Kiểm tra xem So có phải là số nguyên tố không
    bool kt = true; int d =2;
    do
    {
        if (So % d ==0) kt = false;
        d++;
    }
    while (d<So);
```

```
    if (kt)        cout<<"So "<<So<<" La so nguyen to";
    else           cout<<"So "<<So<<" Khong la so nguyen to";
}
```

Để thoát khỏi vòng lặp không xác định, ta cũng có thể sử dụng break; hoặc goto. Tuy nhiên, các trường hợp này ít xảy ra do đặc thù của vòng lặp.

Chuyển đổi giữa các cấu trúc lặp:

Từ vòng lặp xác định (for), ta có thể chuyển sang vòng lặp không xác định (while hoặc do/ while) bằng cách sử dụng thêm một biến đếm kiểu nguyên trong thân vòng lặp không xác định. Trước khi lặp ta khởi gán giá trị của biến đếm này bằng 0. Mỗi khi thực hiện một lần lặp, ta tăng giá trị của biến đếm này lên 1. Điều kiện dừng là khi số lần lặp đã đủ.

Để chuyển đổi ngược lại (từ cấu trúc lặp không xác định sang cấu trúc lặp xác định) thì nói chung, ta phải sử dụng các vòng for khuyết biểu thức 2, dưới dạng:

for (<Biểu thức 1>; ; <biểu thức 3>)

bởi vì ta không biết chắc số lần lặp của cấu trúc. Khi đó ta có thể sử dụng lệnh break hoặc goto trong thân vòng for để thoát có chủ đích.

II.3. Các ví dụ minh hoạ sử dụng vòng lặp

Ví dụ 1. Viết chương trình nhập vào một số nguyên n, sau đó tính tổng các số nguyên tố thuộc đoạn [1..n]. Cho biết có bao nhiêu số nguyên tố thuộc đoạn trên?

```
int main()
{
        int n, Tong, Dem;
       cout<<"Nhap so nguyen: "; cin>>n;
        Tong =Dem=0;
        for (int i=1; i<=n; i++)
        {
    bool Check = true;
            for (int j=2; j< i; j++)
                        if (i%j==0) Check = false;
            if (Check)
            {
                        Tong +=i;       Dem++;
            }
```

```
        }
        cout<<"Tong cac so nguyen to: "<<Tong;
        cout<<"Co "<<Dem<< " so nguyen to trong doan 1..n";
}
```

Ví dụ 2. Viết chương trình nhập vào một số nguyên n và một số thực x, sau đó tính giá trị biểu thức:

$$F = \begin{cases} x + \frac{x^2}{3} + \frac{x^3}{3^2} + .. + \frac{x^n}{3^{n-1}} & \text{Nếu n chẵn} \\ 0 & \text{Nếu n lẻ} \end{cases}$$

```
int main()
{
    int n; float x, F, ts, ms;
    cout<<"Nhap x"; cin>>x;
    cout<<"Nhap n"; cin>>n;
    F=0;
    if (n%2==0)
    {
            F=x; ts=x; ms=1;
            for(int i=1; i<n; i++)
            {
                    ts*=x;          ms*=3;
                    F+=ts/ms;
            }
    }
    cout<<"Gia tri cua bieu thuc F= "<<F;
}
```