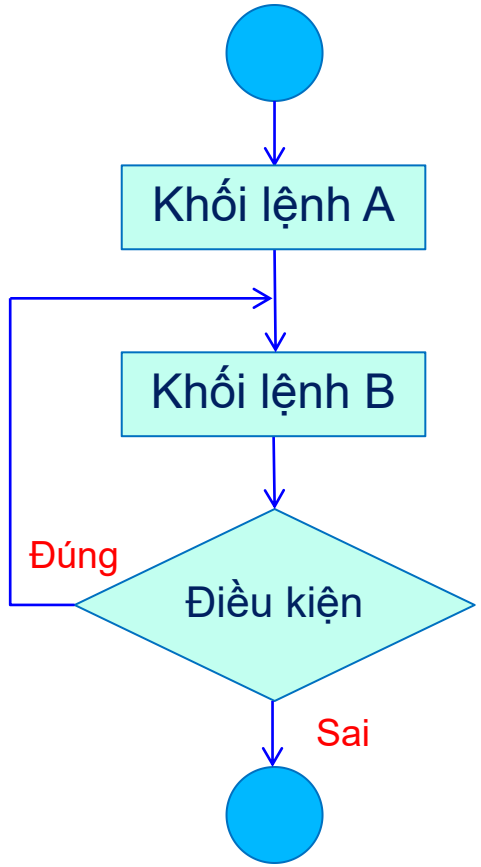
KỸ THUẬT LẬP TRÌNH C/C++

Chương 5:
CẤU TRÚC ĐIỀU KHIỂN LẶP

Nội dung

1. Vòng lặp for
2. Vòng lặp while
3. Vòng lặp do-while
4. Lệnh break
5. Lệnh continue
6. Lệnh goto
7. Hàm exit()



1. Vòng lặp for

1. Vòng lặp for

Vòng lặp for trong C++ là một cấu trúc điều khiển lặp được sử dụng để thực thi số lần lặp cụ thể.

Cú pháp:

```
for (<khởi tạo biến>; <điều kiện lặp>; <bước nhảy>)
{
    // khối lệnh trong vòng lặp for;
}
```

<khởi tạo biến>: khởi tạo giá trị của biến và chỉ thực thi một lần duy nhất.

<điều kiện lặp>: kiểm tra biến đã khởi tạo với điều kiện lặp, nếu <điều kiện lặp> đúng (true) thì thực thi khối lệnh trong vòng lặp for, nếu <điều kiện lặp> sai (false) thì kết thúc vòng lặp for.

<bước nhảy>: sẽ thay đổi giá trị của biến lúc khởi tạo ban đầu (có thể tăng hoặc giảm). Giá trị của biến này sẽ được kiểm tra lại với <điều kiện lặp> sau mỗi lần lặp.

1. Vòng lặp for

Ví dụ 1: Viết chương trình in ra màn hình năm số tăng dần từ 1 đến 5.

```
#include <iostream>
using namespace std;
int main(){
	for (int i = 1; i <= 5; i++)
	{
		cout << i << endl;
	}
	return 0;
}
```

Lần lặp	Biến i	Điều kiện lặp i<=5	Thực thi
1	i = 1	true	In ra 1 và tăng i lên 2
2	i = 2	true	In ra 2 và tăng i lên 3
3	i = 3	true	In ra 3 và tăng i lên 4
4	i = 4	true	In ra 4 và tăng i lên 5
5	i = 5	true	In ra 5 và tăng i lên 6
6	i = 6	false	Kết thúc vòng lặp for

1. Vòng lặp for

Ví dụ 2: Viết chương trình in ra màn hình năm số giảm dần từ 5 đến 1.

```
#include <iostream>
using namespace std;
int main(){
	for (int i = 5; i > 0; i--)
	{
		cout << i << endl;
	}
	return 0;
}
```

Lần lặp	Biến i	Điều kiện lặp i > 0	Thực thi
1	i = 5	true	In ra 5 và giảm i còn 4
2	i = 4	true	In ra 4 và giảm i còn 3
3	i = 3	true	In ra 3 và giảm i còn 2
4	i = 2	true	In ra 2 và giảm i còn 1
5	i = 1	true	In ra 1 và giảm i còn 0
6	i = 0	false	Kết thúc vòng lặp for

1. Vòng lặp for

Ví dụ 3: Viết chương trình in ra màn hình năm số chẵn tăng dần từ 0 đến 10.

```
#include <iostream>
using namespace std;
int main(){
	for (int i = 0; i <= 10; i = i + 2)
	{
		cout << i << endl;
	}
	return 0;
}
```

Kết quả:

```
0
2
4
6
8
10
```

1. Vòng lặp for

❖ **Một số lưu ý khi sử dụng vòng lặp for**

➢ Vòng lặp for có thể lồng nhau.

Ví dụ 4:

```
#include <iostream>
using namespace std;
int main() {
	int n = 3, m = 4;
	if (n < 10 && m < 20) {
		for (int i = 0; i < n; i++)	{
			for (int j = 0; j < m; j++) {
				cout << i + j << endl;
			}
		}
	}
	return 0;
}
```

Kết quả:

?

1. Vòng lặp for

❖ Một số lưu ý khi sử dụng vòng lặp for

➢ Vòng lặp for có thể lồng nhau.

Ví dụ 4:

```
#include <iostream>
using namespace std;
int main() {
        int n = 3, m = 4;
        if (n < 10 && m < 20) {
                for (int i = 0; i < n; i++)     {
                        for (int j = 0; j < m; j++) {
                                cout << i + j << endl;
                        }
                }
        }
        return 0;
}
```

Kết quả:

```
i = 0; j = 0 =>      0
i = 0; j = 1 =>      1
i = 0; j = 2 =>      2
i = 0; j = 3 =>      3

i = 1; j = 0 =>      1
i = 1; j = 1 =>      2
i = 1; j = 2 =>      3
i = 1; j = 3 =>      4

i = 2; j = 0 =>      2
i = 2; j = 1 =>      3
i = 2; j = 2 =>      4
i = 2; j = 3 =>      5
```

1. Vòng lặp for

❖ **Một số lưu ý khi sử dụng vòng lặp for (tt)**

➢ Trong vòng lặp for, có thể sẽ không có phần <khởi tạo biến>, vì biến đã được khởi tạo bên ngoài lệnh for.

Ví dụ 5 (a):

```
#include <iostream>
using namespace std;
int main(){
	for (int i = 1; i <= 5; i++)
	{
		cout << i << endl;
	}
	return 0;
}
```

Ví dụ 5 (b):

```
#include <iostream>
using namespace std;
int main(){
	int i = 1;
	for (; i <= 5; i++)
	{
		cout << i << endl;
	}
	return 0;
}
```

1. Vòng lặp for

❖ **Một số lưu ý khi sử dụng vòng lặp for (tt)**

➢ Trong vòng lặp for, có thể sẽ không có phần <bước nhảy>, vì nó được đặt khối lệnh của for.

Ví dụ 6 (a):

```
#include <iostream>
using namespace std;
int main(){
    int i;
    for (i = 0; i < 10; i++) {
        cout << i << endl;
    }
    return 0;
}
```

Ví dụ 6 (b):

```
#include <iostream>
using namespace std;
int main(){
    int i;
    for (i = 0; i < 10; ) {
        cout<< i << endl;
        i++;
    }
    return 0;
}
```

1. Vòng lặp for

❖ **Một số lưu ý khi sử dụng vòng lặp for (tt)**

➢ Trong vòng lặp for, có thể sẽ không có phần <điều kiện lặp>, <điều kiện lặp> được đặt trong khối lệnh của for.

Ví dụ 7 (a):

```
#include <iostream>
using namespace std;
int main(){
    int i;
    for (i = 0; i <=10; i++){
        cout<<i<<endl;
    }
    return 0;
}
```

Ví dụ 7 (b):

```
#include <iostream>
using namespace std;
int main(){
    int i;
    for (i = 0; ; i++){
        cout<<i<<endl;
    }
    return 0;
}
```

Ví dụ 7 (c):

```
#include <iostream>
using namespace std;
int main(){
    int i;
    for (i = 0; ; i++){
        if (i >= 10)
        break;
        cout<<i<<endl;
    }
    return 0;
}
```

1. Vòng lặp for

❖ Một số lưu ý khi sử dụng vòng lặp for (tt)

➢ Trong vòng lặp for, các thành phần chính gồm <khởi tạo biến> , <điều kiện lặp>, <bước nhảy> cách nhau bởi dấu ; Nếu có nhiều thành phần con trong mỗi thành phần chính thì được cách nhau bởi dấu ,

Ví dụ 8:

```
#include <iostream>
using namespace std;
int main(){
        for (int i = 1, j = 2; i + j < 10; i++, j += 2) {
                cout<< i + j <<endl;
        }
        return 0;
}
```

Kết quả:

?

2. Vòng lặp while

2. Vòng lặp while

Vòng lặp while trong C++ là một cấu trúc điều khiển lặp được sử dụng để lặp một phần của chương trình một vài lần. Nếu số lần lặp không được xác định trước thì vòng lặp while được khuyến khích sử dụng trong trường hợp này.

Cú pháp:

```
while(<điều kiện lặp>) {
    // Khối lệnh được lặp lại cho đến khi điều kiện = False
}
```

Trong đó <điều kiện lặp> thường là biểu thức với các toán tử quan hệ, kết quả trả về là true (1) hoặc false (0).

Nguyên lý hoạt động của vòng lặp while: Kiểm tra <điều kiện lặp>, nếu <điều kiện lặp> đúng (true) thì khối lệnh trong vòng lặp while được thực thi, nếu <điều kiện lặp> là sai (false) thì sẽ thoát khỏi vòng lặp while và chuyển đến dòng lệnh bên ngoài vòng lặp while.

2. Vòng lặp while

Ví dụ 9: In ra màn hình các số từ 1 đến 5 sử dụng vòng lặp while.

```
#include <iostream>
using namespace std;
int main() {
  int i = 1;
  while (i <= 5) {
      cout << i << endl;
      i++;
  }
  return 0;
}
```

Lần lặp	Biến i	Điều kiện lặp i<=5	Thực thi
1	i = 1	true	In ra 1 và tăng i lên 2
2	i = 2	true	In ra 2 và tăng i lên 3
3	i = 3	true	In ra 3 và tăng i lên 4
4	i = 4	true	In ra 4 và tăng i lên 5
5	i = 5	true	In ra 5 và tăng i lên 6
6	i = 6	false	Kết thúc vòng lặp while

2. Vòng lặp while

❖ Một số lưu ý khi sử dụng vòng lặp while.

➢ Vòng lặp while có thể lồng nhau.

Ví dụ 10:

```
#include <iostream>
using namespace std;
int main() {
  int n = 2, m = 5;
  if (n < 10 && m < 20) {
        while (n >= 1) {
              while (m >= 1) {
                    cout << m << endl;
                    m--;
              }
              n--;
        }
  }
  return 0;
}
```

Kết quả:

?

2. Vòng lặp while

❖ Một số lưu ý khi sử dụng vòng lặp while (tt).

➢ Vòng lặp while có thể không thực hiện lần nào do điều kiện lặp ngay từ lần đầu đã không thỏa mãn.

Ví dụ 11:

```
#include <iostream>
using namespace std;
int main() {
  int n = 1;
  while (n > 10)  {
        cout << n <<endl;
        n--;
  }
  return 0;
}
```

2. Vòng lặp while

❖ Một số lưu ý khi sử dụng vòng lặp while (tt).

➢ Vòng lặp while có thể bị lặp vô hạn.

Ví dụ 12:

```
#include <iostream>
using namespace std;
int main() {
  int n = 1;
  while (1)  {
       cout << n <<endl;
       n++;
  }
  return 0;
}
```

3. Vòng lặp do-while

3. Vòng lặp do-while

Vòng lặp do-while là một biến thể của vòng lặp while. Vòng lặp này sẽ thực thi khối lệnh một lần, trước khi kiểm tra xem điều kiện có đúng không, sau đó nó sẽ lặp lại vòng lặp nếu điều kiện là đúng.

```
do {
   // Khối lệnh
}
while (<điều kiện lặp>);
```

Trong đó <điều kiện lặp> thường là biểu thức với các toán tử quan hệ, kết quả trả về là true (0) hoặc false (0).

Nguyên lý hoạt động của vòng lặp do-while: Thực thi dòng lệnh trong do-while trước. Sau đó, kiểm tra <điều kiện lặp>, nếu <điều kiện lặp> đúng (true) thì thực thi dòng lệnh trong do-while lần nữa. Quá trình kiểm tra <điều kiện lặp> và thực thi dòng lệnh trong do-while sẽ chấm dứt cho đến khi <điều kiện lặp> sai (false).

Lưu ý: Vòng lặp do-while kết thúc bằng dấu chấm phẩy.

3. Vòng lặp do-while

Ví dụ 13: In ra màn hình các số từ 1 đến 5 sử dụng vòng lặp do-while.

```
#include <iostream>
using namespace std;
int main() {
	int i = 1;
	do {
		cout << i << endl;
		++i;
	}
	while (i <= 5);
	return 0;
}
```

Lần lặp	Biến i	Điều kiện lặp i<=5	Thực thi
	i = 1	Không kiểm tra	In ra 1 và tăng i lên 2
1	i = 2	true	In ra 2 và tăng i lên 3
2	i = 3	true	In ra 3 và tăng i lên 4
3	i = 4	true	In ra 4 và tăng i lên 5
4	i = 5	true	In ra 5 và tăng i lên 6
5	i = 6	false	Kết thúc vòng lặp do-while

3. Vòng lặp do-while

❖ **Sự khác nhau giữa vòng lặp while và do-while:**

Ví dụ 14 (a):

```
#include <iostream>
using namespace std;
int main() {
	int n = 1;
	while (n > 10) {
		cout<<n<<endl;
		n--;
	}
	return 0;
}
```

➢ Kết quả: ??

Ví dụ 14 (b):

```
#include <iostream>
using namespace std;
int main() {
	int n = 1;
	do
	{
		cout<<n<<endl;
		n--;
	} while (n > 10);
	return 0;
}
```

➢ Kết quả: ??

3. Vòng lặp do-while

❖ **Sự khác nhau giữa vòng lặp while và do-while:**

Ví dụ 14 (a):

```
#include <iostream>
using namespace std;
int main() {
	int n = 1;
	while (n > 10) {
		cout<<n<<endl;
		n--;
	}
	return 0;
}
```

➢ Kết quả: không in gì ra màn hình

Ví dụ 14 (b):

```
#include <iostream>
using namespace std;
int main() {
	int n = 1;
	do
	{
		cout<<n<<endl;
		n--;
	} while (n > 10);
	return 0;
}
```

➢ Kết quả: 1

4. Lệnh break

4. Lệnh break

Câu lệnh **break** trong C++ có hai cách sử dụng như sau:

- Khi gặp câu lệnh break trong một vòng lặp, vòng lặp sẽ kết thúc ngay lập tức và câu lệnh kế tiếp sau vòng lặp được thực thi.
- Lệnh break có thể được sử dụng để kết thúc một case trong mệnh đề switch case.

Nếu bạn sử dụng vòng lặp lồng nhau, câu lệnh break sẽ dừng việc thực hiện vòng lặp trong cùng và bắt đầu thực hiện câu lệnh kế tiếp sau vòng lặp trong cùng.

4. Lệnh break

Ví dụ 15: In ra màn hình giá trị của biến i trong mỗi vòng lặp, khi điều kiện (i == 5) thỏa mãn thì vòng lặp kết thúc.

```
#include <iostream>
using namespace std;
int main() {
    for (int i = 0; i < 10; i++) {
        if (i == 5) {
            break;
        }
        cout << i << endl;
    }
    return 0;
}
```

Kết quả:

```
0
1
2
3
4
```

5. Lệnh continue

5. Lệnh continue

Câu lệnh **continue** trong C++ hoạt động giống như câu lệnh break. Thay vì buộc kết thúc vòng lặp, nó buộc trở về kiểm tra điều kiện để thực hiện vòng lặp tiếp theo và bỏ qua các lệnh bên trong vòng lặp hiện tại sau lệnh continue.

Khi gặp lệnh continue, chương trình sẽ kết thúc vòng lặp hiện tại (hay bỏ qua một lần lặp), còn break thì kết thúc luôn vòng lặp.

Đối với vòng lặp for, câu lệnh continue làm cho điều khiển chương trình tăng hoặc giảm biến đếm của vòng lặp. Đối với vòng lặp while và do-while, câu lệnh continue làm cho điều khiển chương trình quay về đầu vòng lặp và kiểm tra điều kiện của vòng lặp.

5. Lệnh continue

Ví dụ 16: In ra màn hình giá trị của biến i trong mỗi vòng lặp, khi điều kiện (i == 3) thỏa mãn thì bỏ qua lần lặp với (i == 3) rồi bắt đầu lần lặp mới và in ra tiếp 4 và 5.

```
#include <iostream>
using namespace std;
int main() {
    for (int i = 1; i <= 5; i++) {
        if (i == 3) {
            continue;
        }
        cout << i << endl;
    }
    return 0;
}
```

Kết quả:

```
1
2
4
5
```

6. Lệnh goto

6. Lệnh goto

Lệnh **goto** trong C++ cho phép nhảy vô điều kiện đến bất kỳ điểm nào trong chương trình thông qua nhãn (label) được khai báo trong chương trình.

Cú pháp:

```
<tên nhãn>:
    // Các lệnh C++
goto <tên nhãn>
```

Lưu ý: Quy tắt đặt tên nhãn cũng tương tự như đặt tên biến. Sau tên nhãn phải có dấu ":"

7. Hàm exit()

7. Hàm exit()

Hàm **exit()** trong C++ được sử dụng để thoát khỏi chương trình. Hàm này, khi được gọi sẽ ngay lập tức kết thúc chương trình và chuyển quyền điều khiển cho hệ điều hành.

Cú pháp:

```
exit (int mã_trả_về);
```

Trong đó:

mã_trả_về thường là số 0; exit(0) sẽ xác định việc kết thúc chương trình một cách bình thường.

Tuy nhiên có một vài trường hợp mã_trả_về là những số khác 0 để xác định một vài loại lỗi.