TÀI LIỆU LẬP TRÌNH CƠ BẢN
C++ VÀ PYTHON

CHUYÊN ĐỀ 3
CẤU TRÚC LẶP – LỆNH LẶP for VÀ while

I. LÝ THUYẾT CẤU TRÚC LẶP for

1. Cấu trúc lặp có ý nghĩa gì?

Bài toán: Viết chương trình in ra màn hình 100 dấu *

Ta có thể sử dụng lệnh như sau:

C++	PYTHON
cout << "**********...";	print("*******...")

Nhận xét: Phải ghi chính xác 100 dấu *

Ý tưởng cải tiến: thực hiện 100 lần, mỗi lần in 1 dấu *

C++	PYTHON
for (int i=1; i<=100; i++) cout << "*";	for i in range(100): print("*", end="")

Chương trình hoàn chỉnh như sau:

C++	PYTHON

int main()
{
for (int i=1; i<=100; i++)
cout << "*";
return(0);
}
	for i in range(10):
	print("*", end="")

Trong chương trình trên, lệnh *for* nằm trước lệnh in ra màn hình, chính lệnh này đã giúp máy tính hiểu là cần phải thực hiện lệnh *cout/print* 100 lần.

 Lệnh lặp được dùng để thực thi một số việc giống nhau, được lặp đi lặp lại nhiều lần, kết thúc khi đủ số lần lặp theo yêu cầu của người dùng hoặc theo điều kiện kết thúc do người dùng quy định.

2. Cấu trúc lặp for trong C++

```
for (khởi tạo giá trị biến lặp ; điều kiện ; cập nhật biến lặp)
{
  <câu lệnh 1>;
  <câu lệnh 2>;
  …
}
```

Nếu chỉ cần lặp một dòng lệnh thì không cần phải có cặp dấu { }

Trong đó:

- **Khởi tạo giá trị biến lặp:** Gán giá trị xuất phát cho biến lặp.
- **Điều kiện để tiếp tục lặp:** Một phép so sánh, nếu điều kiện đúng thì sẽ tiếp tục vòng lặp.
- **Cập nhật biến lặp:** Biểu thức xác định lượng thay đổi giá trị của biến lặp sau mỗi vòng lặp.

Xét câu lệnh lặp sau:

```
for (int i = 1; i<=10 ; i++)
```

int i=1: Biến lặp **i** có kiểu là **int** và được gán giá trị ban đầu là **1**.

i <= 10: Điều kiện để vòng lặp được tiếp tục. Nếu điều kiện sai thì sẽ kết thúc vòng lặp. Trong trường hợp này, vòng lặp sẽ kết thúc khi điều kiện i<=10 bị sai (tức là lúc i=11)

Đôi khi trong lệnh ***for*** không có điều kiện. Việc dừng hay tiếp tục thực hiện vòng lặp tiếp theo phụ thuộc vào việc kiểm tra bên trong câu lệnh:

```
for (int i=1; i<=100; i++ )
{
    tong+=i;
    if (tong > 1000) break;
}
```

Trong ví dụ trên, lệnh **break** sử dụng để dừng vòng lặp khi lệnh này được gọi, tức là khi *tong* > *1000*, lệnh *break* sẽ được thực hiện, vòng lặp sẽ lập tức dừng lại mà không cần quan tâm đến điều kiện ban đầu. Nếu điều kiện tong > 1000 không bị sai, vòng lặp vẫn hoạt động bình thường cho đến khi điều kiện $i \leq 100$ không còn đúng nữa.

i++: sau khi thực hiện xong vòng lặp, giá trị của i sẽ tăng lên một đơn vị, rồi lại kiểm tra điều kiện, nếu vẫn đúng thì thực hiện vòng lặp, rồi lại tăng i lên 1 đơn vị, …

Việc thay đổi giá trị của biến đếm có nhiều kiểu khác nhau:

KIỂU THAY ĐỔI	LỆNH	VÍ DỤ
Tăng i thêm 1 đơn vị ở mỗi lần lặp lại	**i++** **hoặc i+=1**	`for (int i=1; i<=10; i++)` `for (int i=1; i<=10; i+=1)`
Tăng i thêm k đơn vị	**i+=k**	`for (int i=1; i<=10; i+=k)`
Giảm i đi 1 đơn vị	**i--**	`for (int i=10; i>=1; i--)`
Giảm i đi 2 đơn vị	**i-=2**	`for (int i=10; i>=1; i-=k)`

❖ Cách thực hiện của câu lệnh:

Xét câu lệnh

```
for (int i=1; i<=100; i++)
        cout << "*";
```

Đầu tiên, giá trị của i là 1 (do lệnh khởi tạo i=1)

Chương trình sẽ kiểm tra điều kiện i<=100, giá trị hiện tại của i là 1 nên i<=100 là đúng, chương trình sẽ thực hiện lệnh cout, sẽ có 1 dấu * được in ra trên màn hình.

Sau khi thực hiện lệnh cout, giá trị của biến i được tăng lên 1 (do lệnh cập nhật biến lặp i++)

Chương trình lại thực hiện kiểm tra điều kiện i<=100, và nếu đúng thì lại tiếp tục thực hiện lệnh cout (các dấu * sẽ tiếp tục được ghi ra), rồi lại tăng i, rồi lại kiểm tra điều kiện, …

Vòng lặp sẽ kết thúc khi điều kiện i<=100 bị sai (i tăng lên đến 101)

Có thể mô tả các bước thực hiện như sau:

Giá trị của i	Điều kiện	Kết quả kiểm tra	Hành động	Kết quả trên màn hình
1	1<=10	Đúng	Thực hiện lệnh rồi tăng i	*
2	2<=10	Đúng	Thực hiện lệnh rồi tăng i	**
3	3<=10	Đúng	Thực hiện lệnh rồi tăng i	***
….				
99	99<=100	Đúng	Thực hiện lệnh rồi tăng i	***… (99 dấu)
100	100<=100	Đúng	Thực hiện lệnh rồi tăng i	***… (100 dấu)
101	101<=100	Sai	Dừng lại	

3. Cấu trúc lặp trong Python

a) Cú pháp lệnh

```
for <biến lặp> in <đối tượng cần lặp qua> :
    <câu lệnh 1>
    <câu lệnh 2>
    …
```

- Đối tượng cần lặp: Là một tập hợp các phần tử, có thể là một đoạn giá trị liên tiếp từ L đến R, một dãy số, một xâu ký tự, ... Trong giới hạn tài liệu này (buổi thứ 3 cơ bản), ta chỉ nói đến đối tượng ***range***

- Biến lặp: Mang ý nghĩa đại diện cho một phần tử trong đối tượng cần lặp, ví dụ với đoạn liên tiếp từ 10 đến 20, biến lặp sẽ lần lượt mang giá trị là 10, 11, …, 20.

- Câu lệnh: Các lệnh sẽ được thực hiện tương ứng với mỗi phần tử.

b) Hàm range() trong Python

range (start, stop, step)

Hàm ***range()*** trong Python cho phép bạn tạo một chuỗi số bằng cách sử dụng các tham số ***start*** (Giá trị bắt đầu), ***stop*** (Giá trị mà tại đó sẽ kết thúc) và ***step*** (Độ tăng giảm của giá trị tiếp theo so với giá trị trước đó). Thông số start và step có thể không cần xuất hiện trong hàm.

Nếu chỉ có 1 thông số, thì thông số đó chính là ***stop***, hàm range sẽ tạo ra dãy số liên tục từ 0 đến stop-1, nghĩa là giá trị biến lặp sẽ lần lượt là 0, 1, …, stop-1.

Nếu có 2 thông số, thì ta có start và ***stop***, hàm sẽ tạo ra dãy số liên tiếp từ start đến stop-1, nghĩa là giá trị biến lặp sẽ lần lượt là start, start+1, start+2, …, stop-1.

Nếu có cả 3 thông số, hàm sẽ tạo ra dãy số từ start đến một số bé hơn hoặc bằng stop-1 (bé hơn stop), trong đó độ lệch giữa 2 giá trị liên tiếp trong dãy là step, nghĩa là giá trị biến lặp sẽ lần lượt là start, start + step, start + 2step, ….

Ví dụ:

CÚ PHÁP HÀM	Ý NGHĨA	KẾT QUẢ TẠO RA
range(5)	Số 5 là giá trị stop, hàm sẽ tạo ra chuỗi các số liên tiếp từ 0 đến 4	0 1 2 3 4
range(2, 5)	2 là start, 5 là stop, hàm sẽ tạo ra chuỗi các số liên tiếp từ 2 đến 4	2 3 4
range(2, 20,3)	Tạo ra các số từ 2 đến một giá trị bé hơn hoặc bằng 19, mỗi số cách nhau 3 đơn vị	2 5 8 11 14 17

c) Hàm range() trong câu lệnh for

Xét câu lệnh lặp in 100 dấu *:

```
for i in range(100):
    print("*", end="")
```

Trong câu lệnh trên, hàm range sẽ tạo ra một chuỗi số từ 0 đến 99. Khi đó giá trị của i sẽ lần lượt là 0, 1, 2, …, 99, với mỗi giá trị i, lệnh print sẽ được thực hiện 1 lần (với mỗi i sẽ in được 1 dấu *)

II. VÍ DỤ

1. Viết chương trình in ra màn hình như sau:

Lan lap thu 1

Lan lap thu 2

…

Lan lap thu 10

C++	PYTHON

int main()
{
for (int i=1; i<=10; i++)
cout << "Lan lap thu " << i << '\n';
return(0);
}
	for i in range(10):
	print("Lan lap thu" , i+1)

2. In ra màn hình từ 1 đến 100, mỗi số cách nhau 1 khoảng trắng:

C++	PYTHON

int main()
{
for (int i=1; i<=100; i++)
cout <<i<<' ';
return(0);
}
	for i in range(100):
	print(i+1, end=" ")

3. In ra màn hình từ -100 đến -1, mỗi số cách nhau 1 khoảng trắng:

C++	PYTHON
for (int i=-100; i<=-1; i++) cout <<i<<' ';	for i in range(-100,0):

	print(i, end=" ")

4. Chương trình in ra màn hình các số tự nhiên chẵn không quá 100, in các giá trị giảm dần (100 98 96 … 2)

C++	PYTHON
for (int i=100; i>=2; i-=2) cout <<i<<' ';	for i in range(100,0,-2): print(i, end=" ")

5. Chương trình tính tổng từ 1 đến n, với n nhập từ bàn phím:

C++	PYTHON

long long s=0;
int n;
cin>>n;
for (int i=1; i<=n;i++) s=s+i;
cout<<s;
	s=0
	n=int(input())
	for i in range(n): s=s+(i+1)
	print(s)

III. LỆNH LẶP while

Lệnh while cũng là lệnh lặp tương tự lệnh for, chỉ khác là lệnh while không

1. Cú pháp

C++	PYTHON

while (điều_kiện)
{
Các lệnh
Cập nhật lại điều kiện
}
	while điều_kiện:
	Các lệnh cần thực hiện
	Cập nhật điều kiện

2. Sử dụng lệnh lặp while trong chương trình - Mối liên hệ giữa for và while

Trong C++, lệnh ***for*** và lệnh ***while*** hoàn toàn có thể thay thế cho nhau. Trong Python thì tùy trường hợp. Tuy nhiên về bản chất, lệnh ***for*** sử dụng cho các trường hợp có thể xác định số vòng lặp rõ ràng, lệnh while lại thích hợp cho những trường hợp điều kiện chưa xác định được cần lặp bao nhiêu vòng lặp. Tùy trường hợp mà lựa chọn lệnh phù hợp và hiệu quả nhất.

Chương trình tính tổng các số từ 1 đến n có thể giải bằng lệnh ***while***:

C++	PYTHON

int n; long long s=0;
cin >>n;
int i=1;
	n=int(input())
	i=1
	s=0

while (i<=n)
{
s+=i;
i++;
}
cout<<s;
	while i<=n:
	s+=i
	i+=1
	print(s)

Chương trình đếm số chữ số của một số nguyên giải bằng lệnh ***while***:

C++	PYTHON

long long n; int dem=0;
cin>>n;
while(n>0)
{
n=n/10;
dem++;
}
cout<<dem;
	n=int(input())
	dem=0
	while n>0:
	n=n//10
	dem+=1
	print(dem)

Chương trình trên có thể sử dụng lệnh ***for*** để thay thế:

C++	PYTHON

long long n;
cin>>n;
int dem=0;
for (; n>0 ;)
{
n=n/10;
dem++;
}
cout<<dem;
	n=int(input())
	dem=0
	for i in range(20):
	n=n//10
	dem+=1
	if n==0:
	break
	print(dem)

Trong C++, lệnh for hoàn toàn có thể thay cho ***while*** và ngược lại, tuy nhiên cú pháp lệnh sẽ khá đặc thù, vẫn nên hạn chế sử dụng.	Trong trường hợp này ta phải dự trù biến i đến 20 vì chữ số nguyên thường vào khoảng 10^{18}. Chương trình trở nên phức tạp hơn.

IV. LỆNH break TRONG CÂU LỆNH LẶP

Lệnh **break** được dùng để thoát khỏi vòng lập ngay lập tức, sử dụng trong các trường hợp trong vòng lặp xảy ra những trường hợp đặc biệt cần phải dừng vòng lặp lại.

1. Ví dụ 1:

Nhập vào một số nguyên n. In ra ước lớn hơn 1 đầu tiên của n.

C++	PYTHON

int n;
cin >> n;
for (int i=1; i<=n; i++)
if (n%i==0) break;
cout << i;
	n=int(input())
	for i in range(2, n+1):
	if n%i==0: break
	print(i)

2. Ví dụ 2:

Viết chương trình nhập vào một số nguyên n, cho biết n có phải là số nguyên tố hay không? (Một số nguyên tố là số tự nhiên lớn hơn 1 và chỉ chia hết cho 1 và chính nó)

Để kiểm tra n có phải là số nguyên tố hay không, ta sẽ chia n cho các số từ 2 đến căn bậc 2 của n, nếu có trường hợp nào chia hết thì n không phải là số nguyên tố.

25 □ 1 5 25

20 □ 1 2 4 4.47123 5 10 20

i <= căn n

□ i*i <= căn n * căn n = n

Thuật toán:

- Đặt 1 biến kiểm tra **kq** kiểu bool, **kq=1** nghĩa là n là số nguyên tố, kq=0 nghĩa là n không phải số nguyên tố, đầu tiên ta gán giá trị kq=1, nghĩa là trước khi kiểm tra, ta giả định số n là số nguyên tố.

- Nếu n<2 gán lại kq=0

- Sử dụng vòng lặp, chia n cho các số từ 2 đến căn bậc 2 của n.

Nếu có một trường hợp i nào mà n%i==0 thì lập tức gán lại kq=0 và break (vì đã sai rồi nên không cần chạy tiếp tục vòng lặp nữa). Như vậy nếu không có trường hợp i nào mà n chia hết thì chạy đến hết vòng lặp vẫn không có lệnh gán **kq** nào xảy ra.

- Kiểm tra lại giá trị của kq và đưa ra kết luận.

C++	PYTHON

```
int n;
cin>>n;
bool kqkiemtra=1;
if (n<2) kqkiemtra=0;
for (int i=2; i*i<=n; i++)
   if (n %i==0)
   {
      kqkiemtra=0;
      break;
   }
if (kqkiemtra) cout<<"YES";
else cout<<"NO";
```

```
n=int(input())
kqkiemtra=1
if n<2: kqkiemtra=0
i=2
while i*i<=n:
   if n%i==0:
      kqkiemtra=0
      break
   i+=1
if kqkiemtra:
   print("YES")
else:
   print("NO")
```

V. LỆNH LẶP LỒNG TRONG LỆNH LẶP

V. BÀI TẬP

1. Viết chương trình in ra màn hình như sau:

1 voi 1 la 2
2 voi 2 la 4
…
10 voi 10 la 20

2. Viết chương trình nhập vào số N (2<=N<=9). In ra bảng cửu chương N

INPUT	OUTPUT
3	3 x 1 = 3 3 x 2 = 6 … 3 x 10 = 30

3. Viết chương trình in ra màn hình 5 dòng như sau

1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10

4. Viết chương trình in ra các số từ 1 đến 100 như sau:

01 2 3 4 5 6 7 8 9 10 (0*10+i)
11 12 13 14 15 16 17 18 19 20 (1*10+i
21 22 23 24 25 26 27 28 29 30
…
91 92 93 94 95 96 97 98 99 100

5. Viết chương trình nhập vào 2 số a, b (a bé hơn b) có 3 chữ số và thực hiện các chức năng sau:

- In ra màn hình các số từ a đến b
- Tính tổng từ a đến b
- Cho biết từ a đến b có bao nhiêu số chia hết cho cả 2 và 3
- Cho biết từ a đến b có bao nhiêu chữ số 1 được sử dụng.

INPUT	OUTPUT
110 115	110 111 112 113 114 115 675 1 13

6. Viết chương trình nhập vào một số nguyên a và một số k. Hãy cho biết chữ số thứ k của a là bao nhiêu (Sử dụng chia lấy nguyên và chia lấy dư để tách chữ số thứ k).

INPUT	OUTPUT
536814 4	8

7. Viết chương trình in ra bảng cửu chương từ 2-9 trên màn hình.

2 x 1 = 2 3 x 1 = 3 4 x 1 = 4