TÓM LƯỢC BÀI GIẢNG
NHẬP MÔN LẬP TRÌNH

(Vũ Quốc Hoàng, vqhoang@fit.hcmus.edu.vn, FIT-HCMUS, 2020)

BÀI 5B
CẤU TRÚC LẶP 2

Chủ đề

- Lưu đồ
- Cấu trúc điều khiển
- Cấu trúc lặp và lệnh lặp
- Duyệt chuỗi

Tài liệu

[1] Vũ Quốc Hoàng, *Bí kíp luyện Lập trình C (Quyển 1)*, hBook, 2017.

[2] Tony Gaddis, *Starting out with C++ From Control Structures through Objects*, Pearson, 8th edition, 2015.

[3] Vũ Quốc Hoàng, *Bí kíp luyện Lập trình nhập môn với Python*, hBook, 2020.

- Đọc kĩ: Bài 2.6 (trang 115-116), Bài 3.2 [1]
- Đọc thêm: Phần 5.7-5.10, 5.12 [2]; Bài 7 [3]

Kiến thức

- Khi thân của một lệnh lặp chứa một lệnh lặp khác, ta có cấu trúc **lặp lồng** (nested loop). Cấu trúc này thường mang lại cách xử lý mạnh mẽ.
- Việc chuyển các vòng lặp lồng về vòng lặp đơn (tức không lồng), thường được gọi là **"khử lồng"**. Xử lý không lồng thường hiệu quả nhưng phức tạp hơn so với xử lý lồng.
- Có 2 lệnh điều khiển thường được dùng để điều khiển chi tiết các vòng lặp là: `break` giúp bỏ qua các lệnh còn lại trong thân lặp và thoát khỏi vòng lặp; `continue` giúp bỏ qua các lệnh còn lại trong thân lặp và chuyển qua lần lặp kế tiếp.
- Lệnh `break` cũng thường được dùng trong các cấu trúc lặp **"giả vô hạn"** (pseudo-infinite loop):

```
while True:
    <S>
    if <C>:
        break
```

```
while(1) {
    <S>
    if(<C>)
        break;
}
```

  với chiến lược **đánh giá/lượng giá muộn** (lazy/deferred evaluation): không xác định điều kiện lặp từ đầu mà xác định khi cần.
- **Chuỗi** (string) là **dãy** (sequence) các **kí tự** (character). Số lượng kí tự trong chuỗi được gọi là **chiều dài** (length) của chuỗi. Các kí tự có thể được truy cập bằng **chỉ số** (index). Lệnh lặp `for` thường được dùng để **"duyệt"** qua các kí tự của chuỗi.

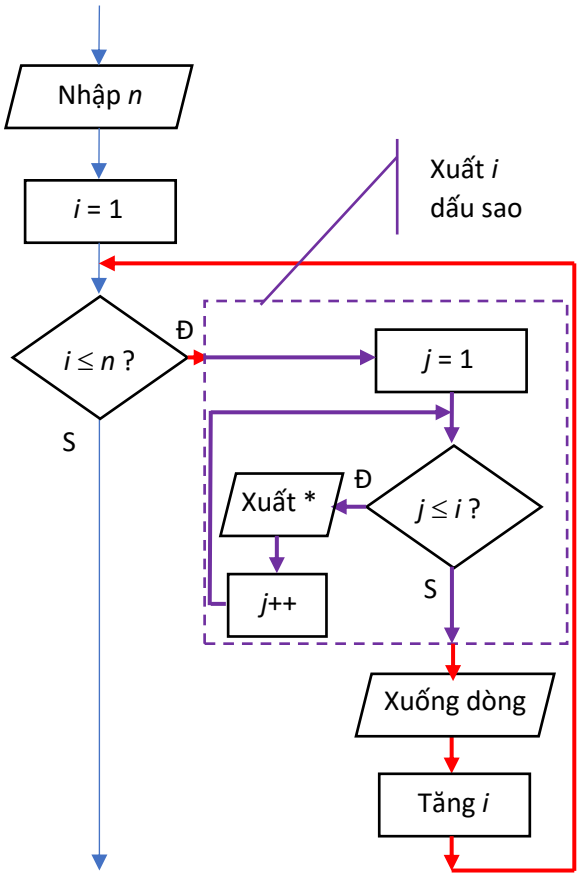
Kĩ năng

- Vẽ được và phân tích được lưu đồ cho các việc đơn giản
- Cài đặt được chương trình từ lưu đồ và vẽ được lưu đồ từ mã nguồn chương trình
- Thành thạo việc dùng vòng lặp lồng, khử lồng, lệnh `break`, `continue`, duyệt chuỗi
- Vận dụng được các cấu trúc lặp phức tạp để giải các bài toán
- Ý thức và rèn luyện được việc viết chương trình đẹp, ngăn nắp, rõ ràng

Mã nguồn minh họa

Mã nguồn 1. (Vòng lặp lồng)

Lưu đồ



Mã giả

Nhập kích thước *n*
Với dòng *i* = 1, 2, …, *n*:
- Xuất *i* dấu sao
- Xuống dòng

Python

```
n = int(input("n = ? "))
for i in range(1, n + 1):
    for j in range(i): # Xuất i dấu sao
        print("*", end="")
    print()
```

C/C++

```
#include <iostream>
using namespace std;

int main() {
    int n;

    cout << "n = ? ";
    cin >> n;

    for (int i = 1; i <= n; i++) {
        // Xuất i dấu sao
        for (int j = 1; j <= i; j++)
            cout << "*";
        cout << endl;
    }
}
```

Giải thích và bình luận thêm:

- Chiến lược thiết kế từ trên xuống giúp "nhìn" và viết các vòng lặp lồng dễ dàng: vòng lặp ngoài là để xuất ra các dòng *i* từ 1 đến *n*; sau đó, việc xuất ra dòng *i* được hiện thực bằng vòng lặp trong (xuất ra *i* dấu sao). Đây cũng nguyên lý thiết kế theo khối hay "hộp đen"
- Biến lặp của các vòng lặp lồng cần được đặt tên khác nhau (thường là `i`, `j`, `k`, ...)
- Trong Python, để xuất mà không xuống dòng, ta đặt giá trị cho tham số `end` của hàm `print` là chuỗi rỗng `""`
- Thử nghĩ ra tình huống dùng vòng lặp lồng 3 "cấp" hay 3 "mức"

Mã nguồn 2. (Vòng lặp lồng và khử lồng)

Mã giả

Cho x, n

$$S = 1 + x + x^2 + \cdots + x^n = \sum_{i=0}^{n} x^i$$

Python 3

```
x = float(input("x = ? "))
n = int(input("n = ? "))
S = sum([x**i for i in range(n + 1)])
print(S)
```

Python 2

```
x = float(input("x = ? "))
n = int(input("n = ? "))
term, S = 1, 1
for i in range(1, n + 1):
    term *= x
    S += term
print(S)
```

Python 1

```
x = float(input("x = ? "))
n = int(input("n = ? "))
S = 0
for i in range(0, n + 1):
    term = 1
    for j in range(i):
        term *= x
    S += term
print(S)
```

C/C++ 2

```
#include <iostream>
using namespace std;

int main(){
    double x; int n;
    cout << "x = ? "; cin >> x;
    cout << "n = ? "; cin >> n;

    double term = 1, S = 1;
    for (int i = 1; i <= n; i++) {
        term *= x;
        S += term;
    }


    cout << S;
}
```

C/C++ 1

```
#include <iostream>
using namespace std;

int main() {
    double x; int n;
    cout << "x = ? "; cin >> x;
    cout << "n = ? "; cin >> n;

    double S = 0;
    for (int i = 0; i <= n; i++) {
        double term = 1;
        for (int j = 1; j <= i; j++)
            term *= x;
        S += term;
    }
    cout << S;
}
```

Giải thích và bình luận thêm:

- Cấu trúc lặp hay được vận dụng để tính toán lặp mà điển hình là tính tổng lặp hữu hạn
- Đơn giản nhất, vòng lặp lồng được dùng để tính lặp, tuy nhiên thường kém hiệu quả (mã Python 1, C/C++ 1)
- Việc khử lồng hay dùng vòng lặp đơn thường hiệu quả hơn (mã Python 2, C/C++ 2) nhưng cũng phức tạp hơn. Chiến lược chung là tính số hạng sau của tổng dựa trên số hạng trước đó, ở trên thì số hạng thứ i (x^i) là tích của số hạng thứ $i-1$ với x ($x^i = x^{i-1} \times x$). Ta cũng thường phải tách riêng số hạng đầu ($x^0 = 1$)
- Python hỗ trợ cách viết rất gọn (mã Python 3) tương tự **kí pháp sigma** (Σ)
- Thật ra, tổng trên là tổng các số hạng của **cấp số nhân** (geometric progression) và có **công thức đóng** (closed form), tức là công thức chỉ dùng cố định các phép toán cơ bản là cộng, trừ, nhân, chia. Cụ thể:

$$S = 1 + x + x^2 + \cdots + x^n = \sum_{i=0}^{n} x^i = \frac{1 - x^{n+1}}{1 - x}$$

Dĩ nhiên, trong trường hợp này, tính bằng công thức đóng sẽ hiệu quả hơn (không minh họa ở trên). Tuy nhiên, nhiều tính toán lặp không có công thức đóng

- Trường hợp tổng vô hạn các số hạng (còn gọi là **chuỗi**, series), tức $n = \infty$, thì sao?

Mã nguồn 3. (Điều khiển chi tiết vòng lặp)

Tiếng Việt

Cho số nguyên *n* > 1, *n* được gọi là số nguyên tố (prime) nếu *n* không chia hết cho số nguyên nào trong khoảng [2, *n* – 1], ngược lại *n* được gọi là hợp số (composite).

Python

```python
n = int(input("Enter n > 1: "))

is_prime = True
for i in range(2, n):
    if n % i == 0: # n chia het cho i
        is_prime = False
        break

if is_prime:
    print("n is prime")
else:
    print("n is composite")
```

C

```c
#include <stdio.h>


int main() {
    int n;
    printf("Enter n > 1: ");
    scanf("%d", &n);

    int is_prime = 1; // true
    for (int i = 2; i < n; i++)
        if (n % i == 0) {
            // n chia het cho i
            is_prime = 0; // false
            break;
        }

    if (is_prime)
        printf("n is prime");
    else
        printf("n is composite");
}
```

C++

```cpp
#include <iostream>
using namespace std;

int main() {
    int n;
    cout << "Enter n > 1: ";
    cin >> n;

    bool is_prime = true;
    for (int i = 2; i < n; i++)
        if (n % i == 0) {
            // n chia het cho i
            is_prime = false;
            break;
        }

    if (is_prime)
        cout << "n is prime";
    else
        cout << "n is composite";
}
```

Giải thích và bình luận thêm:

- Vì chỉ cần chia hết cho chỉ 1 số trong phạm vi [2, *n* – 1] thì *n* là hợp số nên việc dùng lệnh `break` ở trên để thoát khỏi vòng lặp "thử chia hết cho" khi tìm thấy ước số sẽ giúp chương trình hiệu quả hơn nhiều (chạy nhanh hơn)
- Kiểu **luận lý** (boolean) chỉ có 2 giá trị là đúng/sai. Trong Python là `True`/`False`; trong C++ là kiểu `bool` với 2 giá trị `true`/`false`; C không hỗ trợ trực tiếp kiểu luận lý mà qui ước dùng số khác 0 (thường là 1) là đúng và số 0 là sai
- Biến luận lý (biến chứa giá trị luận lý) thường được gọi là **biến cờ** (flag variable). Khi dùng làm điều kiện chọn (hay lặp) ta chỉ cần dùng giá trị của biến mà không cần dùng toán tử so sánh. Chẳng hạn

ở trên, viết là `if(is_prime)` chứ không nên viết `if(is_prime == True)` hay `if(is_prime != 0)`

- Nếu thay "duyệt xuôi" (*i* chạy từ 2 đến *n* – 1) thành "duyệt ngược" (*i* chạy từ *n* – 1 về 2) thì sao?

Mã nguồn 4. (Lặp qua các kí tự của chuỗi)

Python 1

```
b = input("Nhập chuỗi số nhị phân: ")
n = 0
for i in range(len(b)):
    n = n*2 + int(b[i])
print("Số thập phân:", n)
```

Python 2

```
b = input("Nhập chuỗi số nhị phân: ")
n = 0
for d in b:
    n = n*2 + int(d)
print("Số thập phân:", n)
```

C

```
#include <stdio.h>
#include <string.h>


int main() {
    char b[100];
    printf("Nhap chuoi so nhi phan: ");
    scanf("%s", &b);

    int n = 0;
    int len = strlen(b);
    for (int i = 0; i < len; i++)
        n = n*2 + (b[i] - '0');
    printf("So thap phan: %d", n);
}
```

C++

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    string b;
    cout << "Nhap chuoi so nhi phan: ";
    cin >> b;

    int n = 0;
    int len = b.length();
    for (int i = 0; i < len; i++)
        n = n*2 + (b[i] - '0');
    cout << "So thap phan: " << n;
}
```

Giải thích và bình luận thêm:

- Để tính chiều dài của chuỗi, trong Python ta dùng hàm dựng sẵn `len`; trong C++, ta dùng **"phương thức"** `length`; trong C, ta dùng hàm `strlen` trong module `string.h`
- Các kí tự trong chuỗi có chỉ số từ 0 đến *len* – 1 với *len* là chiều dài chuỗi và được truy cập bằng cú pháp: `<chuỗi>[<chỉ số>]`
- Kí tự trong Python là chuỗi đặc biệt (chuỗi chỉ có 1 kí tự); kí tự trong C/C++ là mã ASCII của kí tự (số nguyên) và có thể được mô tả bằng hằng kí tự với dấu nháy đơn (`'<kí tự>'`)
- Việc duyệt qua các kí tự của chuỗi thường được cài đặt bằng lệnh `for` với biến lặp là chỉ số (mã Python 1, C, C++)
- Lệnh `for` trong Python thực ra giúp duyệt qua các phần tử của một dãy mà chuỗi chỉ là một trường hợp (mã Python 2)
- Việc "chuyển" kí số (là kí tự 0, 1, ...) thành số (số 0, 1, ...) có thể được thực hiện bằng hàm `int` trong Python hoặc bằng hiệu với mã ASCII của kí tự 0 (`'0'`) trong C/C++ (vì mã ASCII của các kí số được sắp liên tiếp)
- Mở rộng để nhập chuỗi số **thập lục phân** (hexadecimal)

Bài tập

1. Làm các bài tập sau theo C, C++: 2.5.1-2.5.4, 3.2.1-3.2.3 [1].
2. Làm thêm các bài tập sau bằng Python: Bài 7 [3].