Programming Technique

Nhóm 8

February 2019

1 Chương 14: Tổ chức câu lệnh tuần tự

1.1 The strongest principle for organizing straight-line code is ordering dependencies.

Nguyên tắc quan trọng nhất của tổ chức câu lệnh tuần tự là sắp xếp câu lệnh các câu lệnh phụ thuộc nhau. Ví dụ: Trong insertsort.py

- Trong file mẫu có 2 lỗi chính tả đã được sửa.
- Trong hàm insertion−Sort, trong vòng lặp for có 3 cụm câu lệnh được tách ra, giúp rõ ràng hơn. Các câu lệnh trong mỗi cụm được viết liền thể hiện sự phụ thuộc.
- Ở dưới hàm $insertion - Sort$, các câu lệnh được viết liền với nhau,sắp xếp hợp lí thể hiện sự phụ thuộc.

1.2 Dependencies should be made obvious through the use of good routine names, parameter lists, comments, and—if the code is critical enough—housekeeping variables.

Câu lệnh cần được thể hiện rõ ràng bằng cách sử dụng tốt tên đoạn chương trình, danh sách tham số, chú thích, và nếu chương trình cần sự chặt chẽ, ta có thể sử dụng các biến trạng thái.

Ví dụ: quick−sort.py

- Code mẫu có vài lỗi cú pháp.
- Thứ tự các hàm được sắp xếp phù hợp.
- Tên các biến được đặt liên quan tới chức năng, các tham số để điều khiển được ứng dụng.

1.3 If code doesn't have order dependencies, keep related statements as close together as possible

Nếu đoạn mã không có sự sắp xếp câu lệnh, các câu lệnh liên quan đến nhau có thể chồng chéo lên nhau khiến người khác đọc code cảm thấy khó hình dung.

2 Chương 15: Sử dụng câu lệnh if-then

2.1 For simple if-else statements, pay attention to the order of the if and else clauses, especially if they process a lot of errors. Make sure the nominal case is clear.

Đối với câu đơn if-else, chú ý đến thứ tự của if và else, đặc biệt là nếu chúng xử lý rất nhiều lỗi. Hãy chắc chắn rằng các trường hợp danh nghĩa là rõ ràng. Ví dụ: bubble−sort.py

- Code không có lỗi nào.
- Các câu lệnh sau lệnh điều kiện if phù hợp.

2.2 For if-then-else chains and case statements, choose an order that maximizes read-ability.

Cần đặt thứ tự sao cho chuỗi if-then-else là dễ đọc nhất. Để câu lệnh dễ đọc, trước hết phải tối giản các điều kiện trong mệnh đề if. Ví dụ: selection−sort.py

2.3 To trap errors, use the default clause in a case statement or the last else in a chain of if-then-else statements.

Bẫy lỗi, sử dụng các mệnh đề ngầm định trong một mệnh đề lệnh hoặc cuối trong một chuỗi các câu lệnh if-then-else. Điều này đảm bảo chương trình có thể ứng phó với mọi giá trị có thể có của biến.

Ví dụ: 15−3.py.

2.4 All control constructs are not created equal. Choose the control construct that's most appropriate for each section of code.

Tất cả các cấu trúc điều khiển không được tạo ra với vai trò như nhau. Cần chọn cấu trúc điều khiển mà là thích hợp nhất.

3 Chương 16: Vòng lặp

3.1 Loops are complicated. Keeping them simple helps readers of your code.

Vòng lặp khá phức tạp, giữ cho vòng lặp đơn giản để giúp người đọc dễ đọc code hơn. Ví dụ: 16-1.py

- Theo em tìm hiểu thì map, và filter không trả về list trong python 3, mình phải ép kiểu list cho nó để có thể dùng được hàm len

3.2 Techniques for keeping loops simple include avoiding exotic kinds of loops, minimizing nesting, making entries and exits clear, and keeping housekeeping code in one place.

Kĩ thuật để giữ vòng lặp đơn giản bao gồm tránh các loại vòng lặp khác lạ, tối giản hóa các vòng lặp lồng nhau, làm cho khởi tạo và thoát vòng lặp rõ rang và giữa cho housekeeping code ở một ví trí:
• Minimizing nesting: Ví dụ: 16-2-nesting.py

- Có lỗi về code về kết thúc vòng lặp while bên trong, lỗi tab đầu dòng của các câu lệnh điều kiện phía dưới.

- Making entries clearly:

Ví dụ: 16-2-entries.py

- Có một vài lỗi cú pháp như it, dấu ngoặc của lệnh print, dấu nháy ‘ cần thay bằng “

- Making exits clearly:

Ví dụ: 16-2-housekeeping.py

- Trong khai báo không có a
- Không có điểm bắt đầu của k

3.3 Loop indexes are subjected to a great deal of abuse. Name them clearly, and use them for only one purpose.

Chỉ số vòng lặp bị lạm dụng rất nhiều. Đặt tên chúng rõ ràng và chỉ dùng cho một mục đích duy nhất.
Ví dụ: 16-2-nesting.py
Ví dụ về cách đặt tên rõ ràng:

Ví dụ :

- Việc dùng tên mang nghĩa rõ ràng ngoài việc giúp dễ đọc còn giúp tránh nhầm lẫn việc dùng lại 1 chỉ số 2 lần.

- Giới hạn miền làm việc của biến chỉ số chỉ thực hiện trong vòng lặp của nó, hay là mỗi biến chỉ số chỉ phục vụ cho một mục đích

16-2-loopHousekeeping.py

3.4 Think through the loop carefully to verify that it operates normally under each case and terminates under all possible conditions.

Nghĩ xuyên suốt vòng lặp thật kỹ càng để kiểm chứng rằng nó đã hoạt động bình thường tổng mọi trường hợp kết thúc nó dưới mọi điều kiện có thể.

4 Chương 17: Các cấu trúc điều khiển

4.1 Multiple returns can enhance a routine's readability and maintainability, and they help prevent deeply nested logic. They should, nevertheless, be used carefully.

Chương trình con có nhiều lần trả về (return) có thể nâng cao khả năng đọc và bảo trì của chương trình và giúp ngăn ngừa sai sót trong những đoạn lồng nhau. Dù vậy, chúng nên được sử dụng cẩn thận.
Ví dụ: binary$_s$*earch.p*

4.2 Multiple returns can enhance a routine's readability and maintainability

- Lệnh return sẽ giúp chương trình ngắn gọn, dễ đọc.
- Trong khi kiểm tra các trường hợp trả về của hàm, cố gắng đơn giản hóa việc kiểm tra điều kiện bằng cách kết hợp các câu lện rẽ nhánh với return một cách hiệu quả, thay vì các lệnh if lồng nhau kiểm tra các điều kiện thì có thể return ngay khi một điều kiện không được thỏa mãn.
- Vì khi gặp lệnh return, chương trình con sẽ dừng lại và trả về kết quả, các câu lệnh sau đó sẽ không được thực thi, vì vậy phải kiểm tra xem vị trí đặt câu lện return có chính xác hay không, đôi khi hay bị mắc phải các trường hợp là do return sớm nên kiểm tra chưa đủ các trường hợp, ...
- Không được quá lạm dụng return, đôi khi việc quá lạm dụng sẽ làm rối chương trình, khó kiểm soát khi debug.

Code mẫu của đề bài bị lỗi do vị chí đặt return không hợp lí, và thiếu trường hợp trả về khi tìm được kết quả của bài toán.. Sửa lại bằng cách ta đặt return – 1 tức là không tìm thấy ra ngoài vòng while, điều chỉnh lại điều kiện của thuật binary_search để hàm chạy đúng.

4.3 Recursion provides elegant solutions to small sets of problems.

Trong chương 17.2 đã lưu ý cho chúng ta các phương diện chung nhất của thuật toán đệ quy. Ngoài ra, khi sử dụng đệ quy, cần phải lưu ý:

- Kiểm tra xem bài toán cần giải quyết có thể phân hoạch thành các bài toán con với cùng vấn đề được hay không.
- Kiểm tra xem các bài toán con có phải là một tập phân hoạch của vấn đề hay không.
- Phải tìm được đại lượng đơn biến của quá trình đệ quy. Giống như vòng lặp while(), để kiểm soát được vòng lặp ta phải có một chỉ số chắc chắn sẽ thay đổi hướng đến trạng thái kết thúc vồng lặp sau hữu hạn vòng lặp.

Code mẫu của đề bài bị lỗi, do khi đệ quy hàm MergeSort, ta chia thành hai phần nhỏ để đệ qiu tiếp tục, nhưng hai hàm MergeSort đệ quy đó chưa phải là tập phân hoạch của bài toán gọi đệ quy hai hàm đó, do thiếu trường hợp nên bị sai. Sửa lại bằng cách thêm trường hợp thiếu vào để hoàn thiện hàm đệ quy.

4.4 Câu lệnh goto

Câu lệnh goto có đặc điểm là thực hiện lại chương trình từ vị trí được đánh dấu nào đó, vì vậy đôi khi code sẽ trở lên rất loạn do quá trình chạy chương trình các câu lệnh tiếp theo sẽ là vị trí được đánh dấu. Vì vậy chỉ dùng goto khi nó là cách thức cuối cùng.

5 Chương 10: Techniques for tackling with variable

Trong tài liệu đã đề cập rất chi tiết các kĩ thuật khi làm việc với biến, sau đây sẽ là một số điểm cần được lưu ý:

- Nên giới hạn phạm vi ảnh hưởng của biến nhất có thể, như vậy sẽ giúp người lập trình và người đọc dễ kiểm soát từng phần của chương trình hơn.
- Tối giảm hóa thời gian sống của biến, mỗi biến sẽ có một nhiệm vụ riêng biệt và thôi ảnh hưởng đến chương trình khi nhiệm vụ của chúng đã kết thúc. Khi bảo trì hoặc nâng cấp chương trình thì mỗi sự thay đổi sẽ được kiểm soát dễ dàng hơn.
- Tên biến nên được đặt để thể hiện được vai trò của nó trong chương trình, thể hiện được sự phân biệt vai trò của các biến trong trường hợp chúng là các biến có cùng ý nghĩa, ví dụ như temporary, index,..., có thể dùng các hậu tố, tiền tố để tăng tính thống kê khi sử dụng biến , hay còn phân biệt vai trò, phạm vi,.... Cố gắng đặt tên biến ngắn gọn nhất có thể .

6 Chương 12: Các kiểu dữ liệu cơ bản

6.1 Working with specific data types means remembering many individual rules for each type. Use this chapter's checklist to make sure that you've considered the common problems.

- Tránh việc sử dụng trực tiếp số trong chương trình mà thay vào đó ên sử dụng Hằng số (Const).
- Tránh việc sử dụng so sánh bằng trong chương trình vì có thể phạm vào lỗi dấu phẩy động. Ví dụ: float.py

6.2 Creating your own types makes your programs easier to modify and more self-documenting, if your language supports that capability.

- Có thể tự tạo kiểu dữ liệu của riêng mình để giúp đơn giản hóa chương trình và dễ dàng sửa lỗi nếu ngôn ngữ lập trình hỗ trợ
  - Tránh việc định nghĩa lại các kiểu dữ liệu đã định nghĩa.
  - Tạo kiểu dữ liệu mới nên hướng đến chức năng của nó.

Ví dụ: coordinate.py
Đây là một kiểu dữ liệu mới lưu lại tỏa đọ một điểm trên mặt phẳng Oxy và có chức năng (method) như khởi tạo, in ra, tính khoảng cách giữa 2 điểm trên mặt phẳng. Chúng ta có thể thêm nhiều chức năng khác vào như: tính hệ số góc, điểm có thuộc đường thẳng được đưa vào không, ...

6.3 When you create a simple type using typedef or its equivalent, consider whether you should be creating a new class instead

Trong c/c++ có thể tạo các kiểu dữ liệu cơ bản bằng typedef hoặc struct. Tuy nhiên trong Java/Python thì tạo một kiểu dữ liệu đồng nghĩa với việc tạo một class mới.

7 Chương 13: Các kiểu dữ liệu đặc biệt

7.1 Structures can help make programs less complicated, easier to understand, and easier to maintain

Trong Python không có kiểu struct như c/c++. Tuy nhiên các kiểu dữ liệu như Tuples, Lists hoặc Dictionaries lại vô cùng mạnh mẽ do các đối tượng được lưu đều là Object. Việc này giúp cho code nhìn đơn giản hơn giúp người khác dễ đọc và dễ bảo trì hơn.

7.2 Whenever you consider using a structure, consider whether a class would work better.

Sử dụng class sẽ dễ dàng cải tiến hơn là sử dụng kiểu struct. Trong Java/Python thì chỉ có class

7.3 Pointers are error-prone. Protect yourself by using access routines or classes and defensive-programming practices.

Dùng con trỏ dễ gây lỗi.
Ví dụ: 13−3.py
Trong Python thì là khái niệm tham chiếu. Luyện tập việc dùng các chương trình con trả về giá trị hoặc là dùng các câu lệnh clone() do trong Python ta làm việc với các Object, còn trong Java ta vẫn có thể dùng các câu lệnh gán với các kiểu dữ liệu primitive như: int, double, ...

7.4 Avoid global variables, not just because they're dangerous, but because you can replace them with something better.

Hạn chế sử dụng biến toàn cục sẽ giúp giảm nhầm lẫn với các biến trong các chương trình con hoặc vô tình thay đổi biến toàn cục. Khi làm việc với các chương trình lớn, nếu thay đổi biến toàn cục sẽ phải thay đổi tất cả biến đó trong chương trình.
Ví dụ: 13−4.py

7.5 If you can't avoid global variables, work with them through access routines. Access routines give you everything that global variables give you, and more.

Ví dụ: $13_5.py$ Trong code sử dụng chương trình con để thực hiện thao tác xóa các phần tử trong L1 mà trùng với L2 thay vì viết ngay trên chương trình chính và chỉ ra một lỗi về tham chiếu trong python mà chúng ta dễ mắc phải nếu không để ý.

8 Chương 5: Các kĩ thuật thiết kế chương trình phần mềm

8.1 If you can't avoid global variables, work with them through access routines. Access routines give you everything that global variables give you, and more.

Ví dụ: *heapsort.py*

- Chia thành nhiều chương trình con giúp dễ quản lí hơn.
- Sử dụng các chương trình con thay vì dùng trực tiếp biến
- Dùng class để tạo kiểu dữ liệu mới riêng biệt
- Trong hàm $max_heapify$ dùng biến cục bộ thay vì dùng biến toàn cục

8.2 Simplicity is achieved in two general ways: minimizing the amount of essential complexity that anyone's brain has to deal with at any one time, and keeping accidental complexity from proliferating needlessly.

- Giảm độ phức tạp
- Có khả năng tái sử dụng
- Tín hiệu cao
- Tính linh động
- Sự phân lớp

Trong heapsort.py được viết thành nhiều hàm giúp giảm độ phức tạp, dễ đọc và có khả năng đem sử dụng ở nơi khác

8.3 Good design is iterative; the more design possibilities you try, the better your final design will be.

Code là quá trình lặp đi lặp lại. Code nhiều sẽ giúp tích lũy kinh nghiệm và gia tăng trình độ.

8.4 Information hiding is a particularly valuable concept. Asking, "What should I hide?" settles many difficult design issues.

Thông tin ẩn là một tính chất vô cùng quan trọng trong mỗi chương trình. Ví dụ như là các thông tin mà Sinh Viên được phép xem và và không được phép xem, hoặc các thông tin mà khách hàng được phép thấy trong một cửa hàng online.
Trong Java có thể dùng kiểu private để ẩn thông tin hàm và biến.
Trong *heapsort.py* hàm $max_heapify$ bị ẩn đi.

9 Chương 7: Kỹ thuật xây dựng hàm, thủ tục

9.1 The most important reason for creating a routine is to improve the intellectual manageability of a program, and you can create a routine for many other good reasons. Saving space is a minor reason; improved readability, reliability, and modifiability are better reasons.

Ví dụ: $7 - 1.py$
Đoạn code trên sử dụng hàm $bisection_cuberoot_approx()$ để tính toán căn bậc ba của một số thực x với độ chính xác epsilon xác định. Người dùng sau chỉ cần sử dụng lại hàm này để trả lại kết quả tương ứng mà không cần quan tâm tới cài đặt cụ thể của hàm. Điều này giúp giảm sự phức tạp của code và tăng khả năng quản lý của lập trình viên.

9.2 Sometimes the operation that most benefits from being put into a routine of its own is a simple one.

Ví dụ: even.py
Đoạn code trên đưa việc kiểm tra tính chẵn - lẻ của một số nguyên i vào trong hàm $is_even()$. Tuy hàm $is_even()$ có ý nghĩa rất đơn giản, nội dung của nó lại không trực quan với người đọc. Sử dụng tên hàm $is_even()$ giúp code dễ đọc, dễ hiểu hơn.

9.3 The name of a routine is an indication of its quality. If the name is bad and it's accurate, the routine might be poorly designed. If the name is bad and it's inaccurate, it's not telling you what the program does. Either way, a bad name means that the program needs to be changed.

Ví dụ:
Đoạn code trên có các hàm $isPalindrome()$, $isPalin()$ (để kiểm tra tính đối xứng của xâu kí tự) và $toChars()$ (để chuyển chữ hoa trong xâu thành chữ thường và loại bỏ các kí tự không phải chữ cái ra khỏi xâu). Lập trình viên có thể dễ dàng suy ra được giá trị trả về từ tên của hàm. Ta có thể cải thiện tên hàm $isPalin()$ được sử dụng nội bộ để phân biệt thành $_isPalin()$ để với hàm $isPalindrome()$ có chức năng gần giống. Ví dụ:
Class Person được khởi tạo với các hàm gán giá trị ($set_name()$), lấy giá trị ($get_name()$, $get_age()$, vv...), chỉnh sửa dữ liệu ($add_friend()$) và tính toán ($age_diff()$). Công dụng của các hàm này có thể suy ra một cách dễ dàng từ tên của hàm.

9.4 Functions should be used only when the primary purpose of the function is to return the specific value described by the function's name.

Ví dụ:
Đoạn code trên đã được sửa lại so với đoạn code được cung cấp trong đề bài: tên hàm $Tower()$ không thể hiện được tính chất của hàm là một thủ tục dùng để giải bài toán Tháp Hà Nội, và có thể gây hiểu nhầm về kiểu dữ liệu trả về. Đổi tên hàm thành solveHanoiTower() giúp giải quyết vấn đề này.

Ví dụ:
Đoạn code trên trả lại số Fibonacci thứ x. Tên hàm fib() thể hiện rõ nội dung của hàm và giá trị trả lại.

10 Chương 3: Các kỹ thuật bẫy lỗi và phòng ngừa lỗi

10.1 Assertions can help detect errors early, especially in large systems, high-reliability systems, and fast-changing code bases.

Ví dụ:
Đoạn code trên sử dụng assert để phát hiện ra trường hợp độ dài mảng grades bằng không, tránh việc thực hiện chia cho 0 xảy ra sau đó. Assert là một công cụ tốt trong quá trình viết code. Trong thực tế, trường hợp này cần được xử lý riêng, hoặc dữ liệu đầu vào cần được đảm bảo thỏa mãn không nảy sinh ra lỗi.

10.2 The decision about how to handle bad inputs is a key error-handling decision and a key high-level design decision.

Ví dụ:
Đoạn code trên có 2 phần. Phần thứ nhất không xử lý các trường hợp gây lỗi (như dữ liệu đầu vào không phải kiểu số, chia cho 0, vv...) và có thể dẫn tới việc chương trình dừng hoạt động. Đoạn code thứ hai sử dụng try/except để bắt và xử lý các lỗi trên một cách đầy đủ.

10.3 Exceptions provide a means of handling errors that operates in a different dimension from the normal flow of the code. They are a valuable addition to the programmer's intellectual toolbox when used with care, and they should be weighed against other error-processing techniques.

Ví dụ:
Đoạn code trên chính là đoạn code tính điểm trung bình trong KT 8.3, với assert được thay thế bằng try/except. Trường hợp chia cho 0 là một trường hợp đơn giản, không cần thiết phải sử dụng tới exception. Một đoạn lệnh rẽ nhánh là đủ để giải quyết được trường hợp này.