Mục lục

I Tổng hợp kỹ thuật viết mã nguồn với các cấu trúc lập trình

I.1 Sử dụng chương 14 – Tổ chức các câu lệnh tuần tự.

Các kỹ thuật tiêu biểu gồm:

KT 14.1. The strongest principle for organizing straight-line code is ordering dependencies.

KT 14.2. Dependencies should be made obvious through the use of good routine names, parameter lists, comments, and—if the code is critical enough—housekeeping variables.

KT 14.3. If code doesn't have order dependencies, keep related statements as close together as possible.

I.2 Sử dụng chương 15 – Sử dụng câu lệnh If-Then

Các kỹ thuật tiêu biểu gồm:

KT 15.1. 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.

KT 15.2. For if-then-else chains and case statements, choose an order that maximizes read-ability.

KT 15.3. To trap errors, use the default clause in a case statement or the last else in a chain of if-then-else statements.

KT 15.4. All control constructs are not created equal. Choose the control construct that's most appropriate for each section of code.

I.3 Sử dụng chương 16 – Vòng lặp

Các kỹ thuật tiêu biểu gồm:

KT 16.1. Loops are complicated. Keeping them simple helps readers of your code.

KT16.2. 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.

KT 16.3. Loop indexes are subjected to a great deal of abuse. Name them clearly, and use them for only one purpose.

KT 16.4.Think through the loop carefully to verify that it operates normally under each case and terminates under all possible conditions.

I.4 Sử dụng chương 17 – Các cấu trúc điều khiển khác

Các kỹ thuật tiêu biểu gồm:

KT 17.1. Multiple returns can enhance a routine's readability and maintainability, and they help prevent deeply nested logic. They should, nevertheless, be used carefully.

KT17.2. Recursion provides elegant solutions to a small set of problems. Use it carefully, too.

KT17.3. In a few cases, gotos are the best way to write code that's readable and maintainable. Such cases are rare. Use gotos only as a last resort.

II Tổng hợp kỹ thuật làm việc với các biến

II.1 Sử dụng chương 10 – Các kỹ thuật chung làm việc với biến

Các kĩ thuật tiêu biểu gồm:

KT10.1. Data initialization is prone to errors, so use the initialization techniques described in this chapter to avoid the problems caused by unexpected initial values.

KT10.2. Minimize the scope of each variable. Keep references to a variable close together. Keep it local to a routine or class. Avoid global data.

KT10.3. Keep statements that work with the same variables as close together as possible.

KT10.4. Early binding tends to limit flexibility but minimize complexity. Late binding tends to increase flexibility but at the price of increased complexity.

KT10.5. Use each variable for one and only one purpose.

II.2 Sử dụng chương 11 – Kỹ thuật đặt tên biến

Các kỹ thuật tiêu biểu gồm:

KT11.1. Good variable names are a key element of program readability. Specific kinds of variables such as loop indexes and status variables require specific considerations.

KT11.2. Names should be as specific as possible. Names that are vague enough or general enough to be used for more than one purpose are usually bad names.

KT11.3. Naming conventions distinguish among local, class, and global data. They distinguish among type names, named constants, enumerated types, and variables.

KT11.4. Regardless of the kind of project you're working on, you should adopt a variable naming convention. The kind of convention you adopt depends on the size of your program and the number of people working on it.

KT11.5. Abbreviations are rarely needed with modern programming languages. If you do use abbreviations, keep track of abbreviations in a project dictionary or use the standardized prefixes approach.

KT11.6. Code is read far more times than it is written. Be sure that the names you choose favor read-time convenience over write-time convenience.

II.3 Sử dụng chương 12 – Các kiểu dữ liệu cơ bản

Các kĩ thuật tiêu biểu gồm:

KT12.1. 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.

Làm việc với những kiểu dữ liệu cụ thể nghĩ là phải nhớ nhiều những quy tắc riêng biệt cho từng kiểu. Như những quy tắc dưới đây:

- Kiểu dữ liệu số tổng quát:
  - Tránh những "magic number", những hằng số xuất hiện ở giữa code mà không có giải thích
  - Có thể code trực tiếp 0, 1 nếu cần thiết
  - Dự tính trường hợp chia cho 0
  - Tránh so sánh khác kiểu giữ liệu
  - Chú ý những cảnh báo của trình dịch
- Kiểu số nguyên:
  - Kiểm tra chia nguyên với chia thực
  - Kiểm tra tràn số
  - Kiểm tra tràn số trong các giá trị trung gian
- Kiểu số thực chấm phẩy động:
  - Tránh cộng và trừ những số có độ lớn khác nhau nhiều
  - Tránh so sánh bằng
  - Dự kiến lỗi làm tròn
- Ký tự và xâu:
  - Tránh những ký tự và xâu "magic"
  - Hiểu rõ ngôn ngữ bạn đang dùng hỗ trợ Unicode ra sao
  - Định rõ khu vực, vùng mà chương trình sẽ được sử dụng
  - Nếu cần hỗ trợ nhiều ngôn ngữ, hãy dùng Unicode
- Biến boolean:
  - Sử dụng kiểu boolean để làm rõ những điều kiện trong chương trình
  - Sử dụng kiểu boolean để đơn giản hoá những test phức tạp
  - Tự tạo một kiểu dữ liệu boolean, nếu như cần thiết
- Kiểu enum:

  - Sử dụng kiểu enum cho tính dễ đọc
  - Sử dụng kiểu enum cho tính tin cậy
  - Sử dụng kiểu enum để cho việc sửa đổi dễ dàng
  - Sử dụng kiểu enum thay thế cho kiểu boolean
  - Sử dụng giá trị ban đầu của enum để cho những giá trị không hợp lệ
  - Xác định rõ phần tử đầu và cuối của enum được sử dụng như thế nào trong toàn bộ project
  - Nếu ngôn ngữ không có kiểu enum thì có thể sử dụng class hoặc từ điển

- Hằng số:
  - Sử dụng hằng số có đặt tên trong khai báo dữ liệu
  - Sử dụng hằng số một cách thống nhất

- Mảng:
  - Đảm bảo rằng chỉ số của mảng ở trong miền cho phép
  - Luôn nghĩ rằng mảng là một cấu trúc tuần tự
  - Nếu mảng là đa chiều, đảm bảo thứ tự của các chỉ số
  - Kiểm tra xem chỉ số khi lặp có đúng như mong muốn
  - Để thêm một phần tử ở cuối mảng

KT12.2. Creating your own types makes your programs easier to modify and more self-documenting, if your language supports that capability.

Tự tạo một kiểu dữ liệu mới sẽ làm cho chương trình dễ dàng sửa đổi và tăng tính tự mô tả, nếu như ngôn ngữ hỗ trợ khả năng này.
Một số những chỉ dẫn tự tạo một kiểu dữ liệu của mình:

- Tạo kiểu dữ liệu với tên hướng đến chức năng của nó
- Tránh những kiểu dữ liệu đã định nghĩa trước đó
- Đừng định nghĩa kiểu dữ liệu đã được định nghĩa
- Định nghĩa kiểu dữ liệu thay thế cho kiểu dữ liệu cũ để tăng tính khả chuyển

KT12.3. When you create a simple type using typedef or its equivalent, consider whether you should be creating a new class instead.

Nếu tự tạo một kiểu dữ liệu mới bằng typedef hoặc cách tương đương, xem xét việc sử dụng class để thay thế nó.

II.4 Sử dụng chương 13 – Các kiểu dữ liệu đặc biệt

Các kĩ thuật tiêu biểu gồm:

KT13.1. Structures can help make programs less complicated, easier to understand, and easier to maintain.

Kiểu dữ liệu có cấu trúc sẽ giúp chương trình giảm độ phức tạp, dễ dàng để hiểu, để duy trì nó hơn.
Một số những lưu ý khi sử dụng kiểu dữ liệu có cấu trúc:

- Sử dụng kiểu dữ liệu có cấu trúc để làm rõ mối quan hệ giữa các dữ liệu
- Sử dụng để đơn giản hoá những thủ tục lên khối dữ liệu
- Sử dụng để đơn giản hoá tham số đầu vào của hàm
- Sử dụng để giảm bớt việc bảo trì chương trình

KT13.2. Whenever you consider using a structure, consider whether a class would work better.

Khi sử dụng một kiểu giữ liệu có cấu trúc, xem xét liệu rằng class sẽ tốt hơn.

KT13.3. Pointers are error-prone. Protect yourself by using access routines or classes and defensive-programming practices.

Con trỏ là kiểu dữ liệu dễ dàng gặp lỗi khi sử dụng. Bảo vệ bản thân bằng việc sử dụng hàm hoặc class khi muốn truy cập nó, hoặc sử dụng kỹ thuật lập trình "tự phòng thủ".

Những yêu cầu khi sử dụng biến con trỏ:

- Sử dụng một kiểu con trỏ rõ ràng thay vì kiểu con trỏ mặc định.
- Tránh cast kiểu dữ liệu con trỏ.
- Trong C, nếu bạn sử dụng dấu * với tham số ở hàm, luôn nhớ rằng giá trị có thể truyền ngược lại khi gọi hàm. Dồng thời khi bạn muốn truyền ngược lại giá trị, chắc chắn rằng phải có ít nhất một dấu sao trong một câu lệnh gán.
- Sử dụng sizeof() để xác định kích thước của biến khi cấp phát bộ nhớ cho nó.

KT13.4. Avoid global variables, not just because they're dangerous, but because you can replace them with something better.

Tránh dử dụng biến toàn cục, không phải chỉ vì nó nguy hiểm mà còn bởi vì bạn có thể thay thế nó bằng những cách thức tốt hơn nhiều.

Những vấn đề chính khi sử dụng kiểu dữ liệu toàn cục:

- Vô tình thay đổi kiểu dữ liệu toàn cục.
- Có thể có những lỗi kì quặc khi sử dụng "tên thay thế" cho biến toàn cục. Chẳng hạn như việc truyền đồng thời tham số vào hàm là biến toàn cục, cũng như sử dụng biến toàn cục trong chính hàm.
- Vấn đề code được sử dụng nhiều lần trong khi nó truy cập đến biến toàn cục, chẳng hạn như lời gọi đệ quy, hoặc trong trường hợp đa luồng.
- Tính tái sử dụng của Code bị cản trở bởi kiểu toàn cục.
- Thứ tự khởi tạo không xác định khi sử dụng kiểu toàn cục.
- Tính module hoá và khả năng kiểm soát code bị huỷ hoại bởi sử dụng kiểu toàn cục.

Những lý do để sử dụng biến toàn cục:

- Bảo tồn những giá trị toàn cục, ví dụ bảng dữ liệu mà tất cả các hàm sử dụng.
- Liệt kê những giá trị hằng số, trong những ngôn ngữ không hỗ trợ hằng số như Python, Perl.
- Sử dụng như kiểu dữ liệu enum, như trong Python.
- Loại bỏ những dữ liệu "lang thang". Ví dụ như khi có 3 hàm, gọi lần lượt lẫn nhau theo đúng thứ tự 1, 2, 3. Hàm 1 và hàm 3 sử dụng cùng một object, nhưng hàm 2 thì không. Nếu truyền qua tham số thì ta vẫn phải truyền nó qua hàm 2, đó là dữ liệu "lang thang". Sử dụng biến toàn cục sẽ loại bỏ những dữ liệu này.

Sử dụng biến toàn cục chỉ khi nó là phương pháp cuối cùng.

- Bắt đầu bởi tạo một biến cục bộ và chỉ thay đổi nó thành toàn cục khi thực sự cần thiết.
- Phân biệt giữa biến toàn cục và biến trong class.
- Sử dụng những hàm, thủ tục để truy cập.

Làm thế nào để giảm bớt những rủi ro khi sử dụng biến toàn cục:

- Triển khai một quy ước đặt tên để làm cho biến toàn cục trở nên hiển nhiên.
- Tạo một danh sách các chú thích tốt cho tất cả các biến toàn cục.
- Đừng sử dụng biến toàn cục cho các kết quả trung gian.
- Đừng giả vờ rằng bạn không sử dụng biến toàn cục bằng cách đặt tất cả dữ liệu vào một object rất lớn và truyền nó khắp mọi nơi.

KT13.5. 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.

Nếu không thể tránh khỏi việc sử dụng biến toàn cục thì nên sử dụng những hàm, thủ tục để truy cập nó thay vì truy cập trực tiếp. Truy cập nhờ hàm, thủ tục sẽ cho bạn mọi thứ như khi sử dụng biến đó trực tiếp, và thêm nhiều hơn thế nữa.

Lợi ích của việc sử dụng hàm truy cập biến toàn cục:

- Bạn có thể tập trung hoá việc truy cập biến toàn cục. Nếu bạn nghĩ ra một cách thức truy cập mới cho dữ liệu đang sử dụng, khi đó bạn không phải thay đổi code nơi mà dữ liệu được sử dụng, chỉ cần thay đổi code trong hàm truy cập.
- Có thể đảm bảo rằng tất cả các truy cập đến biến toàn cục được "rào chắn".
- Bạn sẽ được lợi nhờ việc che giấu thông tin tự động.
- Hàm truy cập sẽ giúp dễ dàng cho việc chuyển sang kiểu dữ liệu trừu tượng.

Làm thế nào để sử dụng hàm truy cập:

- Yêu cầu tất cả các truy cập biến phải thông qua hàm truy cập.
- Đừng chỉ vứt toàn bộ những biến toàn cục vào một cái thùng.
- Sử dụng cơ chế khoá để bảo vệ biến toàn cục khi lập trình đa luồng.
- Xây dựng thêm một mức trừu tượng hoá vào trong hàm truy cập.
- Giữ tất cả các truy cập của dữ liệu ở cùng một mức trừu tượng, đừng cho phép sử dụng nhiều mức trừu tượng để truy cập giữ liệu.

III Tổng hợp kỹ thuật xây dựng chương trình, hàm, thủ tục

III.1 Sử dụng chương 5 – Các kỹ thuật thiết kế chương trình phần mềm – Design in Construction

Các kĩ thuật tiêu biểu gồm:

KT5.1. Software's Primary Technical Imperative is managing complexity. This is greatly aided by a design focus on simplicity.

Yêu cầu kỹ thuật chính đối với phần mềm là kiểm soát sự phức tạp. Nó được hỗ trợ rất lớn bởi thiết kế tập trung vào sự đơn giản.

Sự khó khăn đối với những thứ "tình cờ" và "thiết yếu":
Tầm quan trọng của việc quản lý sự phức tạp trong phát triển phần mềm:

KT5.2. 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.

KT5.3. Design is heuristic. Dogmatic adherence to any single methodology hurts creativity and hurts your programs.

KT5.4. Good design is iterative; the more design possibilities you try, the better your final design will be.

KT5.5. Information hiding is a particularly valuable concept. Asking "What should I hide?" settles many difficult design issues.

KT5.6. Lots of useful, interesting information on design is available outside this book. The perspectives presented here are just the tip of the iceberg.

III.2 Sử dụng chương 7 – Kỹ thuật xây dựng hàm/thủ tục High-Quality Routines

Các kĩ thuật tiêu biểu gồm:

KT7.1. 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.

KT7.2. Sometimes the operation that most benefits from being put into a routine of its own is a simple one.

KT7.3. You can classify routines into various kinds of cohesion, but you can make most routines functionally cohesive, which is best.

KT7.4. 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.

KT7.5. Functions should be used only when the primary purpose of the function is to return the specific value described by the function's name.

KT7.6. Careful programmers use macro routines with care and only as a last resort.

III.3 Sử dụng chương 8 – Các kỹ thuật bẫy lỗi và phòng ngừa lỗi – Defensive Programming

Các kĩ thuật tiêu biểu gồm:

KT8.1.Production code should handle errors in a more sophisticated way than "garbage in, garbage out."

KT8.2. Defensive-programming techniques make errors easier to find, easier to fix, and less damaging to production code.

KT8.3. Assertions can help detect errors early, especially in large systems, high-reliability systems, and fast-changing code bases.

KT8.4 The decision about how to handle bad inputs is a key error-handling decision and a key high-level design decision.

KT8.5. 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.

KT8.6. Constraints that apply to the production system do not necessarily apply to the development version. You can use that to your advantage, adding code to the development version that helps to flush out errors quickly.