

Mục lục

I	Tổng hợp kỹ thuật viết mã nguồn với các cấu trúc lập trình	2
I.1	Sử dụng chương 14 – Tổ chức các câu lệnh tuần tự.	2
I.2	Sử dụng chương 15 – Sử dụng câu lệnh If-Then	2
I.3	Sử dụng chương 16 – Vòng lặp	2
I.4	Sử dụng chương 17 – Các cấu trúc điều khiển khác	3
II	Tổng hợp kỹ thuật làm việc với các biến	3
II.1	Sử dụng chương 10 – Các kỹ thuật chung làm việc với biến	3
II.2	Sử dụng chương 11 – Kỹ thuật đặt tên biến	3
II.3	Sử dụng chương 12 – Các kiểu dữ liệu cơ bản	4
II.4	Sử dụng chương 13 – Các kiểu dữ liệu đặc biệt	6
III	Tổng hợp kỹ thuật xây dựng chương trình, hàm, thủ tục	6
III.1	Sử dụng chương 5 – Các kỹ thuật thiết kế chương trình phần mềm – Design in Construction	6
III.2	Sử dụng chương 7 – Kỹ thuật xây dựng hàm/thủ tục High-Quality Routines	7
III.3	Sử dụng chương 8 – Các kỹ thuật bắt lỗi và phòng ngừa lỗi – Defensive Programming	8

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 kỹ 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 kỹ 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 dữ 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 sẽ làm cho chương trình dễ dàng sửa đổi và tăng tính tự mô tả, nếu như ngôn ngữ hỗ trợ khả năng này.

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

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

Các kỹ thuật tiêu biểu gồm:

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

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

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

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

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.

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 kỹ thuật tiêu biểu gồm:

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

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 kỹ 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 kỹ 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.