TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

BÀI GIẢNG PHƯƠNG PHÁP LUẬN LẬP TRÌNH

Người soạn: THs Phạm Thị Thương

Bộ môn: Công nghệ phần mềm.

Thái Nguyên, 2013

LỜI NÓI ĐẦU

Bài giảng này được biên soạn chủ yếu dựa trên tài liệu ***Công nghệ phần mềm*** của GS.TSKH Nguyễn Xuân Huy. Ngoài ra tác giả còn dựa trên một số cuốn sách tác giả có chỉ rõ nguồn gốc của các nội dung trích dẫn được trình bày trong sách này để bạn đọc thuận lợi cho việc theo dõi và đọc thêm các tài liệu liên quan.

Qua lần giảng dạy thứ 2 về môn học này, tác giả đã mạnh dạn biên soạn cuốn sách với mục đích làm tài liệu học tập và tham khảo cho các bạn sinh viên ngành Công nghệ thông tin và những người yêu thích hoạt động lập trình. Vì kinh nghiệm và thời gian có hạn, chắc chắn bài giảng không thể tránh khỏi các khiếm khuyết về mặt nội dung cũng như về hình thức trình bày. Tác giả rất mong nhận được các ý kiến đóng góp của các bạn đọc gần xa để cuốn sách ngày càng hoàn thiện hơn và trở thành cuốn tài liệu hay, thú vị đối với bạn đọc.

Mọi ý kiến đóng góp xin gửi về theo địa chỉ: tn.univer@gmail.com

Tác giả

CHƯƠNG 1 LỊCH SỬ HÌNH THÀNH VÀ PHÁT TRIỂN CỦA LẬP TRÌNH

1.1 Kỹ thuật lập trình giai đoạn thứ nhất của MTĐT

Lập trình, hay nói chính xác hơn là học lập trình là một công việc nặng nhọc, năng xuất thấp. Có thể nói năng xuất lập trình đứng cuối bảng so với các hoạt động trí tuệ khác. Cho đến nay, năng xuất của lập trình viên chỉ dao động trong khoảng 4-5 lệnh/ngày. Một sản phẩm phần mềm có thể được thiết kế và cài đặt trong khoảng 6 tháng với 3 lao động chính. Nhưng để kiểm tra và tiếp tục tìm lỗi, hoàn thiện sản phẩm đó phải mất thêm chừng 3 năm. Hiện tượng này là phổ biến trong tin học, người ta khắc phục nó bằng một mẹo nhỏ có tính chất thương mại như sau: Thay vì sửa sản phẩm, người ta công bố bản sửa đó dưới dạng một phiên bản mới.

> Ví dụ: Ta thấy hệ điều hành DOS 4.0, chỉ tồn tại một thời gian ngắn được thay bằng DOS 5.0, ...tương tự cho hệ điều hành Window.....

Trong thời kỳ đầu của tin học, khoảng những năm 50, người ta lập trình bằng các ngôn ngữ bậc thấp.

- Việc nạp và theo dõi hoạt động của chương trình một cách trực tiếp theo chế độ trực tuyến (on-line), tìm & diệt lỗi (debugging) như ta hay làm ngày nay là không thể được.
- => Lập trình viên ngày xưa làm việc thận trọng hơn ngày nay rất nhiều.

Trước những năm 60, người ta coi lập trình như một hoạt động nghệ thuật, nhuốm màu sắc tài nghệ cá nhân hơn là khoa học. Một người nắm được một ngôn ngữ lập trình và một số mẹo vặt tận dụng cấu hình phần cứng cụ thể của máy tính có thể được xem là chuyên gia nắm bắt được những bí mật của "nghệ thuật lập trình".

1.2 Cuộc khủng hoảng PM những năm 60

Những năm 60 đã bùng nổ " cuộc khủng hoảng về đảm bảo phần mềm" được đặc trưng bởi hai hiện tượng sau đây:

- Chi phí cho tin học quá lớn, trong đó chi phí cho phần mềm chiếm tỉ lệ cao và ngày càng tăng so với chi phí cho kỹ thuật tin học (phần cứng).
  - Năm 1965 tổng chi phí cho tin học trên Thế giới chiếm 250 tỉ Franc.
  - Rất nhiều đề án lớn nhằm ứng dụng tin học bị thất bại liên tiếp. Nguyên nhân thất bại chủ yếu là do phần đảm bảo sản phẩm
- Để giải quyết những vướng mắc trong kỹ thuật lập trình, các nhà tin học lý thuyết đã đi sâu vào nghiên cứu, tìm hiểu bản chất của ngôn ngữ, thuật toán và các hoạt động lập trình và nâng nội dung của nó lên thành nguyên lý khoa học.

Các kết quả nghiên cứu điển hình như:
+ Dijkstra trong nghiên cứu của mình đã chỉ ra rằng: "động thái của chương trình có thể được đánh giá một cách tường minh qua các cấu trúc lặp, rẽ nhành, gọi đề quy..." và rằng "... tay nghề của lập trình viên tỷ lệ nghịch với toán tử goto mà anh ta viết trong chương trình"
+ Gues trong bài báo của mình đã phân tích sâu sắc nguyên nhân dẫn đên tình trọng trong lập trình dùng goto bừa bãi sẽ biến chương trình thành một mớ rối rắm như món mì sợi.
+ Gries phê phán trong các trường đại học, người ta dạy ngôn ngữ lập trình chứ không dạy kỹ thuật lập trình, người ta dạy các mẹo để lập trình chứ không quan tâm đến các nguyên lý & phương pháp luận lập trình.

1.3 Những tư tưởng cách mạng trong lập trình

Tính cách mạng của những quan điểm khoa học nẩy nở trong giai đoạn này thể hiện ở những điểm sau đây:

- Chương trình máy tính và lập trình viên trở thành đối tượng nghiên cứu của lý thuyết lập trình
- Vấn đề cơ bản đặt ra đối với lý thuyết lập trình là "làm thế nào có thể làm chủ được sự phức tạp của hoạt động lập trình".

Do trí tuệ của từng cá nhân lập trình viên là có hạn mà các vấn để thực tiễn cần giải quyết bằng các công cụ tin học là lớn và phức tạp. Vấn đề đặt ra là liệu có thể phân bài toán lớn thành những bài toán nhỏ có độ phức tạp thấp để giải riêng, sau đó tổng hợp kết quả lại được không?

Các kết quả nghiên cứu đạt được:
+ Năm 1969, Hoere phát biểu các tiên đề phục vụ cho việc chứng minh tính đúng đắn của chương trình và phát hiện tính chất bất biến của vòng lặp. Sau đó Dijkstra và Hoere đã phát triển ngôn ngữ lập trình có cấu trúc.
+ Để triển khai được các nguyên lý lập trình Wirth đã thiết kế và cài đặt ngôn ngữ ALGOL W – một biến thể của ALGOL – 60. Sau này ALGOL W tiếp tục được hoàn thiện để trở thành ngôn ngữ lập trình Pascal. Đây là ngôn ngữ giản dị, trong sáng về cú pháp, dễ minh họa các tư tưởng của lập trình hiện đại => Rất phù hợp trong giảng dạy.
+ Năm 1978, Kernighan và Ritchie đã thiết kế ra ngôn ngữ lập trình C.

Cuộc cách mạng lập trình diễn ra những năm 60 – 70 đem lại cho chúng ta những nhận thức sau đây:

- Lập trình là một trong những lĩnh vực khó nhất của toán học ứng dụng. Có thể coi lập trình như một khoa học nhằm đề xuất các nguyên lý và phương pháp nâng cao hiệu xuất lao động cho lập trình viên. Năng xuất ở đây cần định hướng trước hết đến:
  - Tính đúng đắn của chương trình
  - Tính dễ đọc, dễ hiểu, dễ thực hiện của chương trình
  - Tính dễ sửa đổi của chương trình
  - Tận dụng tối đa khả năng của thiết bị mà vẫn không phụ thuộc vào thiết bị.
- Tóm lại: Kỹ thuật lập trình hay ở mức độ rộng hơn là CNPM nhằm hướng đến mục tiêu cuối cùng là "Sử dụng tối ưu sự phối hợp giữa người và máy".
- Người ta chỉ kiểm soát được tính đúng đắn của một đối tượng nào đó nếu nó được kiến trúc một cách đơn giản và trong sáng.
  - Lập trình viên phải thoát khỏi những ràng buộc cụ thể về văn phạm của ngôn ngữ lập trình, phải diễn đạt một cách trong sáng và đúng đắn các chỉ thị (chứ không phải biết bao nhiêu ngôn ngữ) – đây là tiêu chuẩn số 1.
  - Trong lập trình, một số lập trình viên mới hay bị cột chặt vào những ngôn ngữ lập trình cụ thể. Thực chất của quá trình lập trình là "người ta không lập trình trên một ngôn ngữ cụ thể mà lập trình hướng đến nó"

1.4 Triển khai chương trình theo sắc thái công nghệ

Công nghệ sản xuất một sản phẩm phần mềm nào đó thường bao gồm các giai đoạn sau:

1. *Tìm hiểu nhu cầu của khách hàng*

   Đây là bước hình thành nên bài toán
2. *Xác định các chức năng cần có của sản phẩm*
3. *Chia nhỏ các chức năng thành các nhóm độc lập tương đối với nhau.* Mỗi nhóm sẽ ứng với một bộ phần hợp thành của sản phẩm

   Ví dụ: Để sản xuất hộp bia, người ta phân chia công nghệ này thành các giai đoạn (nhóm các chức năng) sau đây:

   a. Chuẩn bị nguyên liệu
   b. Lên men
   c. Làm hộp

d. Ướp hương liệu
e. Đóng hộp
f. Đóng thùng
g. Nhập kho

Chú ý rằng, mỗi công đoạn bao gồm nhiều chức năng khác nhỏ hơn.

Ví dụ:

+ Công đoạn *a) chuẩn bị nguyên liệu* có thể bao gồm:

- Chuẩn bị lúa đại mạch,
- Chuẩn bị hoa bia,
- ....

+ Công đoạn *c) làm hộp* có thể gồm:

- Cắt tấm nhôm,
- Dập hộp,
- In nhãn.

4. *Giao việc thiết kế và sản xuất sản mỗi bộ phần của sản phẩm này cho từng người hoặc nhóm người.*
5. *Các nhóm triển khai công việc*: Thực hiện các bước thiết kế, sản xuất, thử nghiệm. Trong quá trình này, các nhóm thường xuyên liên hệ với nhau nhằm hạn chế tối đa các công việc trùng lặp và đảm bảo tính tương thích khi ghép nối các bộ phận.
6. *Ghép nối các bộ phận/chi tiết thành phẩm*
7. *Thử nghiệm sản phẩm, sửa nếu cần.*
8. *Bán và giao lô sản phẩm đầu tiên cho khách hàng*
9. *Thu thập thông tin phản hồi từ phía người sử dụng*. Quyết định sửa lại bản sản phẩm, cải tiến hoặc hủy bỏ việc sản xuất sản phẩm này.

Trên đây là quy trình sản xuất sản phẩm nói chung. Nhóm làm phần mềm cũng thực hiện ngần ấy nhiệm vụ ở các giai đoạn tương ứng. Trong nhóm các cách chuyên viên cho từng nhiệm vụ.

- Phân tích viên + lãnh đạo nhóm: Đảm nhận các nhiệm vụ từ 1- 4
    - Sản phẩm thu được sau giai đoạn 4 là phần đầu của hồ sơ phần mềm gồm:
        - Các đặc tả yêu cầu
        - Các đặc tả chức năng của phần mềm cũng như các module tương ứng.
- Giai đoạn 5: Dành cho các nhóm triển khai công việc

- Các nhóm trưởng lại tiếp tục phát triển bộ hồ sơ = cách bổ sung thêm các đặc tả chi tiết, tinh chế dần từng bước cho đến khi nhận được những chương trình con viết trên ngôn ngữ lập trình cụ thể
- Các lập trình viên cũng tham gia vào việc:
  - Phân tích các nhiệm vụ con
  - Đặc tả từng khối chức năng trong nhóm của mình, đặc tả tốt có thể giao nhiệm vụ mã hóa cho máy tính
  - Mã hóa:
    - Chuyển biểu diễn của thuật toán từ dạng đặc tả sang dạng mã nguồn (sử dụng ngôn ngữ lập trình cụ thể).
    - Khi mã hóa cũng nên tận dụng quỹ thuật toán và quỹ chương trình:
      - Vì thời gian giao nộp sản phẩm rất hạn hẹp, nên tìm và sử dụng lại những mô đun, thủ tục hoặc thuật toán đã có sẵn.
      - Để thực hiện được điều này không phải là dễ, nó đòi hỏi một ý thức trách nhiệm cao của mọi thành viên trong tập thể. Ở đây, nguyên tắc "mình vì mọi người, mọi người vì mình" được coi là đắc dụng. Khi viết một phần mềm nào đó, dù nhỏ, bạn luôn có ý thức rằng mình viết cho nhiều người dùng. Có ý thức đó, ngoài cái lợi là bạn sẽ thận trọng cho sản phẩm của mình, bạn còn thực hiện được đóng góp quan trọng vào quỹ thuật toán và quỹ chương trình, mà ở đó, bạn cũng có quyền khai thác các sản phẩm trí tuệ của người khác.
      - Có thể nói, không có lĩnh vực nào con người phung phí chất xám nhiều như trong tin học. Với năng xuất thảm hại là 4-5 lệnh/ngày, người ta làm ra hàng vạn, hàng triệu module, chương trình, hệ thống ... để thực hiện cùng một công việc.
  - Trao đổi với các nhóm khác để:
    - nhận được những thủ tục dùng chung và
    - thông hiểu lẫn nhau:
      - Muốn hiểu biết tốt nên:
        - Dùng cùng một thứ ngôn ngữ đặc tả, nhằm mô tả những yếu tố sau của chương trình:
          - Input, output,

- Các phép biến đổi
- Các kết quả cần đạt được ở mỗi điểm của chương trình.

▪ Hành động theo cùng một nguyên tắc: Nguyên tắc hành động đảm bảo tính trong sang, dễ hiểu là triển khai chương trình theo cấu trúc từ trên xuống dưới và tinh chế dần

▪ Hướng dẫn cách sử dụng các chi tiết sản phẩm do nhóm mình phát triển.

- Giai đoạn 6: Do chuyên viên trưởng về lập trình thực hiện, chuyên viên này cần:
  - Trực tiếp tham gia vào các hoạt động 1-4
  - Theo dõi tiến độ hoạt đông 5
  - Bám sát hoạt động của giai đoạn 7
- Nhiệm vụ 7: Do chuyên viên kiểm thử thực hiện
- Nhiệm vụ 8+9: Do một chuyên gia Marketing + lãnh đạo nhóm thực hiện. Chuyên gia này cần tham gia vào các hoạt động từ 1-4. Các nhóm cần có sự tư vấn của
  - Các chuyên gia đảm bảo toán học và các chuyên gia thuật toán.
  - Các cố vấn kỹ thuật: Tư vấn, trợ giúp nhóm lựa chọn thiết bị (phần cứng), thuật toán, phầm mềm trợ giúp...

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 1

CHƯƠNG 2 CÁC PHƯƠNG PHÁP LUẬN LẬP TRÌNH

2.1 Đặt vấn đề *[6]*

Mặc dù các nhà khoa học đã cố gắng rất nhiều để giới thiệu một nền tảng khoa học trong lập trình. Nhưng lập trình hầu như luôn được dạy một cách thủ công (craft):

- Nó thường được dạy theo ngữ cảnh của một (hoặc một vài) ngôn ngữ lập trình

- Có sự mập mờ giữa các công cụ, các khái niệm và các cách khác nhau về quan điểm lập trình/phương pháp lập trình.

=> Dạy lập trình theo cách này, cũng giống như dạy xây dựng các cây cầu (bridge). Một số người dạy cách xây dựng các cây cầu gỗ, một số người lại dạy cách xây dựng các cây cầu bằng sắt....mà không nghĩ đến việc kết hợp giữa gỗ và sắt. Do đó, dẫn đến chương trình phải chịu một bản thiết kế nghèo nàn (bị ràng buộc bởi những hạn chế của kỹ thuật như công cụ, ngôn ngữ, ...).

Khoa học giúp ta hiểu sâu sắc vấn đề và sự hiểu biết này tạo cho ta khả năng dự đoán và khái quát hóa. Ví dụ: Trong khoa học thiết kế, cho phép người thiết kế thiết kế mọi cây cầu (có thể tạo bằng bất kỳ chất liệu nào như sắt, gỗ, hoặc kết hợp cả hai..) và thiết kế chúng theo những khái niệm như sau:

- Sức lực (force)

- Năng lượng (energy)

- Sức ép (Stress)

- Sự căng (Strain)

- và các luật (Laws) mà chúng tuân theo.

Tương tự, trong khoa học lập trình, ta cũng cần thiết kế chương trình dựa trên các khái niệm chung về lập trình. Có thể xem lập trình là hoạt động nằm giữa 2 hoạt động là đặc tả hệ thống và chạy chương trình cài đặt bản đặc tả. Như vậy, lập trình gồm 2 bước:

- Thiết kế kiến trúc chương trình và các trừu tượng (a)

- Mã hóa bản thiết kế, sử dụng ngôn ngữ lập trình cụ thể (b)

Trong đó:

(a) Thiết kế kiến trúc chương trình và các trừu tượng

- Là những hoạt động độc lập với ngôn ngữ lập trình, và cũng là công việc khó nhất và quan trọng nhất trong lập trình nhằm đưa ra bản kiến trúc chương trình và các *thiết kế trừu tượng*. Các thiết kế trừu tượng được định nghĩa là các thiết bị/công cụ

để giải quyết một vấn đề thực tế. Đây là chìa khóa chính của các trừu tượng. Các trừu tượng có thể được phân loại theo thứ bậc, phụ thuộc vào cách chúng được đặc tả. Càng ở mức cao, sự trừu tượng càng có mức khái quát cao hơn. Sự trừu tượng cũng là một phần của cuộc sống hàng ngày mà ta thường lãng quên.

Ví dụ: Một số trừu tượng như:

- Những quyển sách
- Các tua vít,
- Cây bút chì
- Công cụ viết

...

Trong đó: *"Bút chì"* được đặc tả chi tiết hơn so với *"công cụ để viết"*, nhưng cả 2 đều là các trừu tượng.

- Để bản thiết kế chương trình mang tính độc lập, ta cần thiết kế chương trình dựa trên các khái niệm lập trình nói chung để xây dựng các trừu tượng. Có thể kết hợp nhiều phương pháp luận lập trình trong một chương trình. Ta có thể thấy dường như là sai lầm khi làm điều này, nhưng thực chất điều này không sao cả trong mọi tình huống, khá tự nhiên rằng một chương trình tốt là sử dụng nhiều phương pháp luận lập trình.

(b) Mã hóa bản đặc tả, sử dụng ngôn ngữ lập trình cụ thể

Sử dụng kỹ thuật mã hóa cụ thể (ngôn ngữ, công cụ, các chuẩn, ...) để chuyển bản đặc tả thiết kế chương trình thành chương trình cụ thể. Điều này giống như việc, từ bản thiết kế cây cầu, chúng ta đi xây dựng cây cầu bằng gỗ, bằng sắt, hoặc kết hợp gỗ và sắt, hoặc bằng chất liệu tùy ý.

<u>*Tóm lại*</u>

Hoạt động lập trình như định nghĩa ở trên gồm hai phần cơ bản:

- Kỹ thuật: Gồm các công cụ, các kỹ thuật thực hành, và các chuẩn cho phép ta lập trình
- Nền tảng khoa học của nó: Gồm các lý thuyết cho phép ta hiểu hoạt động lập trình. Khoa học giải thích kỹ thuật một cách trực tiếp, và giúp ích cho ta hiểu sâu sắc vấn để để từ đó có thể vận dụng, phân tích, tổng hợp và đưa ra các cải tiến hoặc để xuất các kỹ thuật mới.....

Khoa học và kỹ thuật phải kết hợp với nhau không thể tách rời để giải quyết trọn vẹn một vấn đề nào đó. Không có kỹ thuật, ta làm việc đơn thuần với các cơ chế. Không có khoa học, ta phải làm việc một cách thủ công, thiếu sự hiểu biết sâu sắc.

2.2 Phương pháp luận là gì

Phương pháp luận là một cách tiếp cận để giải quyết vấn đề nào đó. Phương pháp luận lập trình là cách tiếp cận để viết ra các chương trình (theo quan điểm lý thuyết). Theo quan điểm kỹ thuật, phương pháp luận lập trình còn được gọi là kỹ thuật lập trình **[1].**

2.3 Các phương pháp triển khai chương trình

2.3.1 Triển khai chương trình từ trên xuống dưới và từ dưới lên trên

Khi vận dụng *nguyên lý phân mức bài toán theo cấp độ trừu tượng hóa*, chúng ta làm quen với hai cách tiếp cận từ trên xuống và từ dưới lên.

Cách tiếp cận thứ nhất (Top – Down) được tác giải của ngôn ngữ Pascal đề xuất vào thập kỷ 70 và đặt tên là làm mịn dần. Điều quan trọng của quá trình làm mịn dần/hay chính xác dần chương trình là phải tiến hành đồng thời với chính xác hóa dữ liệu. Cận tiếp cận này giúp ta xuất phát từ *máy giải* trừu tượng đi dần đến máy giải cụ thể được trang bị bởi ngôn ngữ lập trình cụ thể.

Cách tiếp cận thứ 2 (Bottom - up) xuất phát từ những viên gạch đầu tiên để thiết kế nền móng, và từng tầng, từng tầng cho đến khi nhận được một kiến trúc hoàn chỉnh (*ví dụ được vận dụng trong giáo dục, đào tạo)*. Các tiếp cận này thường được vận dụng trong trường hợp chiến lược giải bài toán chưa được nghiên cứu.

Quá trình trừu tượng hóa được chia làm nhiều mức. Mỗi mức nói chung được xác định bởi 4 công cụ:

1. Ngôn ngữ
2. Cấu trúc dữ liệu
3. Các thao tác
4. Máy giải

Trong đó:

+ *Ngôn ngữ*: là công cụ dùng để mô tả CTDL và các thao tác cần thiết. Ngôn ngữ ở các mức trừu tượng ở mức trên có thể là ngôn ngữ tự nhiên hoặc ngôn ngữ tự nhiên có cấu trúc (phi hình thức hoặc bán hình thức)

+ *Máy giải*: Ở mức cao, máy giải là máy trừu tượng (máy giả định). Càng ở mức trên thì mức độ trừu tượng càng cao theo nguyên tắc “thấy cây chưa quan trọng bằng thấy rừng”.

Quá trình trừu tượng hóa được chính xác dần và mịn dần ở các mức dưới cho đến khi nhận được chương trình hoàn chỉnh viết trên ngôn ngữ lập trình cụ thể để chạy trên ngôn ngữ cụ thể.

Ví dụ: Triển khai chương trình Phanso.Pas (minh họa nguyên lý phân mức bài toán theo cấp độ trừu tượng hóa. Mục đích cuối cùng là ta thu được chương trình Pacal tên là PHANSO.PAS chạy trên máy tính IBM XT/AT.

Mức 0: <Mức xuất phát>

Chúng ta có một cặp (bộ đôi) α = <P, F>. Trong đó P là tập các phân số, F là tập các phép toán trên phân số, F = {+, - , *, /}.

+ Những bộ đôi như vậy được gọi là hệ đại số. Tổng quát thì hệ đại số α = <P, F> là một bộ đôi, trong đó:

- P là tập các phần tử

- F là tập các phép toán trên các phần tử của P. Mỗi phần tử f trong F là một ánh xạ: f: P^n -> P, trong đó P^n là ký hiệu biểu diễn cho tích Decartes bậc n:

$P^n = P \times P \times ....\times P$, mỗi phần tử của P^n có dạng $(x_1, x_{2, ...,}x_n)$, $x_i \in P$; i:1..n

F còn được gọi là phép toán n ngôi.

+ Trở lại với khái niệm phân số ở mức 0, ta tiếp tục mô tả P và F. Để mô tả một tập, người ta dùng 2 cách: Liệt kê mọi phần tử của tập hoặc chỉ ra các tính chất xác định của các phần tử thuộc tập.

- Với P ta có thể mô tả:

(1) Bằng ngôn ngữ tự nhiên như sau;

Tập các phân số P bao gồm các cặp *tử* - *mẫu*, trong đó tử là một số nguyên, mẫu là một số tự nhiên

(2) Bằng các ký hiệu toán học như sau:

$P = \{x/y | x \in Z, y \in N\}$ ***(1)***

Lưu ý: dấu , trong (1) biểu diễn quan hệ *"and"*

- Mô tả F

Ta định nghĩa các phép toán +, -, *. / trong F. Vì ký hiệu / đã được dùng để biểu diễn phân số, nên ta dùng ký hiệu : cho phép chia hai phân số. Cả 2 phép toán trên đều là các phép toán 2 ngôi:

$+ : P \times P \to P$

$- : P \times P \to P$

$* : P \times P \to P$

$: : P \times P \to P$

} ***(2)***

Ví dụ: Muốn cộng hai phan số, ta quy đồng mẫu số của chúng rồi cộng tử số, giữ nguyên mẫu số chung, kết quả sẽ được giảm ước.

Tóm lại: Ở mức 0 ta có:

- CTDL: được mô tả ở ***(1)***
- Các phép toán: được mô tả ở ***(2)***
- Ngôn ngữ: Toán học
- Máy giải: Con người

Mức 1 <Chi tiết hóa các phép toán trên phân số và cấu trúc dữ liệu>

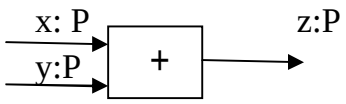
- Chi tiết hóa CTDL

  $P = \{x/y \mid x \in Z, y \in N\}$

  Nếu $p = x/y \in P$ thì *tử(p)*= x; *mẫu(p)* = y

- Chi tiết hóa các phép toán trên phân số
  - Mô tả phép cộng 2 phân số:

    $+ : P \times P \to P$



$z = x + y \overset{\Delta}{=}$ qđms (x,y);
tu(z) = tu(x) + tu(y);
mau(z) = mau(x);
rutgon(z);

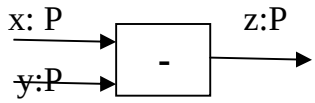
Thay hình vẽ về bộ biến đổi ta có thể ghi:

Input: x:P, y:P
Output: z:P

Máy giải ở đây còn trừu tượng ở chỗ coi nó thực hiện được 2 thao tác là *qđms* (quy đồng mẫu số 2 phân số) và *rutgon* (rút gọn một phân số)

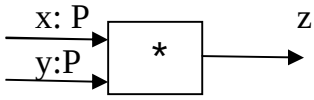
  - Tương tự cho các phép toán còn lại, ta có

- : P×P → P



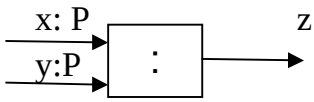
z = x - y $\overset{\Delta}{=}$ qđms (x,y);
tu(z) = tu(x) - tu(y);
mau(z) = mau(x);
rutgon(z);

* : P×P → P



z = x * y $\overset{\Delta}{=}$ tu(z) = tu(x) * tu(y);
mau(z) = mau(x)*mau(y);
rutgon(z);

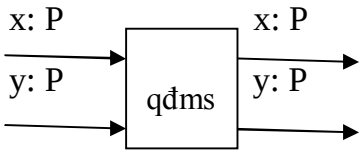
: : P×P → P



z = x : y $\overset{\Delta}{=}$ Yêu cầu: tu(y) ≠ 0; mau(z)>0
tu(z) = tu(x) * mau(x);
mau(z) = mau(x)*tu(y);
rutgon(z);

Mức 2 <Tiếp tục triển khai các thao tác *qđms* và r*utgon*>

qđms: P×P → P×P



qđms(x,y) $\overset{\Delta}{=}$ d= msc(mau(x), mau(y));
tspx = d:mau(x);
tspy= d:mau(y);
tu(x) = tu(x)*tspx;

mau(x) = d;

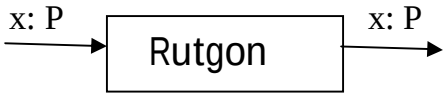
tu(y) = tu(y)*tspy;

mau(y) = d;

Trong đó: msc – mẫu số chung

tsp – thừa số phụ

Rutgon: $P \rightarrow P$



Rutgon(x) $\underset{=}{\Delta}$ d= ucln(tu(x), mau(x));

tu(x) = tu(x):d;

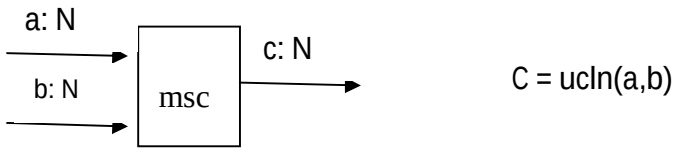
mau(x) = mau(x):d;

Mức 3 <Tiếp tục triển khai các thao tác lấy mẫu số chung *(msc)* và ước chung lớn nhất *(ucln)*>

msc: $N^2 \rightarrow N$

msc(a,b) bcnn(a,b);

ucln: $N^2 \rightarrow N$ $\underset{=}{\Delta}$



ucln(a,b) $\underset{=}{\Delta}$ khi b>0

r=du(a,b);

a=b;

b=r;

dừng khi b=0;

c =a;

Đây là thuật toán Euclide: Muốn tìm ước số chung của hai số tự nhiên a, b, ta chia nguyên a cho b để lấy số dư r. Sau đó, nếu $r \neq 0$ ta lại lấy b chia cho r để tìm số dư mới.

Đến đây, chúng ta hãy áp dụng ***nguyên lý nhất quán*** sau đây để thực hiện một vài sửa đổi:

> Nguyên lý nhất quán
>
> Dữ liệu được khai báo thế nào thì phải thao tác thế ấy. Cần sớm phát hiện những mâu thuẫn giữa CTDL và thao tác để khắc phục ngay

Ta định nghĩa phân số như sau:

$P = \{x/y | x \in Z, y \in N\}$ (3)

Trong thao tác rút gọn ta tính:

d=ucln(tu(x),mau(x)) (4)

Sau đó ta mô tả hàm ucln như là một ánh xạ $N \times N \rightarrow N$

Ucln: $N \times N \rightarrow N$ (5)

=> Với những phân số có tử là số âm, ví dụ -3/4 thì biểu thức ucln(-3,5) sẽ vi phạm mô tả 5. Ta nói (4) đã thiết lập nên mâu thuẫn giữa (3) và (5)

Để khắc phục mâu thuẫn này, ta có thể chọn một trong các biện pháp sau:

Biện pháp 1: Sửa lại mô tả ucln để hàm này có thể làm việc với các số âm:

Ucln: $Z \times Z \rightarrow N$

Sửa đổi này dĩ nhiên sẽ kéo theo một vài sửa đổi tương ứng trong hàm ucln

Biện pháp 2: Giữ nguyên định nghĩa cũ của ucln và sửa lại thao tác rút gọn.

Dưới đây ta chọn biện pháp 2: Việc sửa đổi này được phản ánh ở mức triển khai số 4 dưới đây.

Sau khi thêm bớt một vài chi tiết nhỏ và thêm thao tác khởi trị cho một phân số cũng như thủ tục quy định cái vào (input) và cái ra (output) chúng ta thu được một bản mô tả ở mức khá gần với ngôn ngữ lập trình truyền thống.

Lưu ý: Chú thích được viết trong cặp dấu (* và *)

Mức 4 <kết quả thu được là các đặc tả mịn, được giao cho nhân viên mã hóa để anh ta chuyển thành các đơn vị chương trình cụ thể, với một ngôn ngữ cụ thể>

```
(* 1-  Mô tả kiểu dl phân số *)
```

$P = \{ x/y \mid x \in Z, y \in N\}$

Nếu p = x/y ∈P thì

Tử(p)=x

Mẫu(p)=y

(*2 - Phép cộng 2 phân số*)

```
+: P×P →P
   (x:P, y: P): x+y: P
    input: x:P, y:P
    output: z:P (*z=x+y*)
```

```
    qđms(x,y);
    tu(z)=tu(x)+tu(y);
    mẫu(z) = mẫu(x);
    rutgon(z);
```

(*3 - Phép trừ 2 phân số*)

```
-: P×P →P
   (x:P, y: P): x-y: P
    input: x:P, y:P
    output: z:P (*z=x-y*)
```

```
    qđms(x,y);
    tu(z)=tu(x)-tu(y);
    mẫu(z) = mẫu(x);
    rutgon(z);
```

(* 4 - Phép nhân 2 phân số*)

***** : P×P →P (x:P, y: P): x ∗ y: P input: x:P, y:P output: z:P (* z=x ∗ y *)
tu(z)=tu(x)∗tu(y); mẫu(z) = mẫu(x)∗mẫu(y); rutgon(z);

(* 5 - Phép chia 2 phân số*)

: : P×P →P (x:P, y: P): x : y: P input: x:P, y:P output: z:P (* z=x : y *) điều kiện: tu(y) ≠0
Nếu tu(y)=0 Báo lỗi (‘chia cho 0’) Dừng chương trình Ngược lại tử(z)=tử(x)∗mẫu(y); mẫu(z) =mẫu(x)*tử(y); nếu mẫu(z)<0 {do tử(y)<0} tử(z) = - tử(z);

```
        mẫu(z) = - mẫu(z);
  rutgon(z);
```

(*6 - Rút gọn phân số*)

Rutgon : P →P

(x:P): x : P

input: x:P

output: x:P (* Rutgon(x) *)

```
nếu  tử(x)<0
     d=ucln(-tử(x),mẫu(x))
ngược lại
  nếu tu(x)=0 dung chtrinh
  nguoc lai
     d=ucln(tử(x), mẫu(x))
tử(x)=tử(x):d;
mẫu(x) = mẫu(x):d;
rutgon(z);
```

(*7- Quy đồng mẫu số 2 phân số*)

qđms : P×P →P×P

(x:P, y: P): (x,y): P×P

input: x:P, y:P

output: x:P; y∈P (* qđms(x,y) *)

Điều kiện:

mẫu(x.)=mẫu(y.)=bcnn(mẫu(x), mẫu(y))

x=x., y=y. (*x.,y. là đầu ra của x, y *)

---

d= bcnn(mẫu(x), mẫu(y));

tử(x)=tử(x)*(d/mẫu(x));

mẫu(x)=d;

tử(y) = tử(y)∗(d/mẫu(y));

mẫu(y)=d;

(*8 - Tìm bội số chung nhỏ nhất của 2 số tự nhiên*)

bcnn : N×N →N×N

(x:N, y: N): bcnn(x,y): N

input: x:N, y:N

output: d:N (* d= bcnn(x,y) *)

Điều kiện:

d⋮x, d⋮y (*chia hết*)

(z:N: z⋮x, z⋮y) => z ≥d

---

d= x*y/ucln(x,y);

(* 9 - Tìm ước chung lớn nhất của 2 số tự nhiên *)

ucln : N×N →N×N (x:N, y: N): ucln(x,y): N input: x:N, y:N output: d:N (* d= ucln(x,y) *) Điều kiện: x⋮d, y⋮d (*chia hết*) (z:N: x⋮z, y⋮z) => d ≥z

khi y>0
r=du(x,y) ;
x=y;
y=r;
dừng
d= x;

(*10 - In một phân số ra màn hình*)

print : p →Screen (*màn hình*)
print(p) = in(tư(p),'/',mẫu(p)) (*trong đó: Thủ tục in là thủ tục cơ bản mà máy trừu tượng ở mức 4 thực hiện*)

*(*trong đó thủ tục in là thủ tục cơ bản mà máy trừu tượng ở bước 4 thực hiện*)*

(*11- Khởi trị cho một phân số *)

Trị:	Z×N → P input: a: Z, n: N output: p: P (* p = a/b*)
	tử(p) = a; mẫu(p)=b;

Công việc còn lại là mã hóa các đặc tả trên theo các quy định của ngôn ngữ lập trình cụ thể

a) Với ngôn ngữ Pascal

Ta sử dụng phương pháp triển khai chương trình từ dưới – lên: bottom - up):

Phương pháp đi từ dưới – lên

(Bottom – up Method)

Đi từ cái chung đến cái riêng, từ đối tượng thành phần ở mức thấp lên các đối tượng mức cao, từ những đơn vị đã biết, lắp ráp thành các đơn vị mới

⇨ Bạn đọc tự viết chương trình theo phương pháp này

b) Với ngôn ngữ C

Ta áp dụng phương pháp triển khai chương trình từ trên xuống:

Phương pháp đi từ trên – xuống

(Top - down Method)

Đi từ cái chung đến cái riêng, từ kết luận đến cái đã biết, từ tổng thể đến đơn vị

Đây là phương pháp được dùng rộng rãi nhất trong quá trình thiết kế và cài đặt chương trình

⇨ Bạn đọc tự viết chương trình theo phương pháp này.

Lưu ý

Hai phương pháp đi xuống và đi lên trong thực tiễn ít khi được dùng một cách thuần túy. Tùy theo kinh nghiệm của mỗi lập trình viên, hai phương pháp này thường được trộn lẫn với nhau, được sử dụng đồng thời theo một tỷ lệ nào đó.

Ví dụ: Xét bài toán vận động sau

Hãy biểu diễn trên màn hình MT cảnh một người chạy đến đá quả bóng

Cách làm:

+ Do ít làm quen với đồ họa máy tính, nên mối lo lắng đầu tiên của chúng ta là làm thế nào để ***biểu diễn được người và bóng***?. Chỉ sau khi xem thư mục giới thiệu các hàm thư viện trong chế độ đồ họa, ta tìm được các hàm cơ bản để từ đó có thể vẽ người và bóng chúng ta mới tạm yên tâm giải tiếp. Giả sử các hàm đó là:

- Line(x1, y1, x2, y2,c): Vẽ một đoạn thẳng nối 2 điểm (x1,y1), (x2, y2) với mầu c
- Circle (x,y,r,c): Vẽ đường tròn tâm (x,y), bán kính r với mầu c
- Point(x,y,c): Chấm một điểm tại (x,y) với mầu c trên màn hình.
- Tag(obj1, obj2): Kiểm tra hai đối tượng có chạm nhau không.

+ Câu hỏi tiếp theo là: Làm thế nào để các ***đối tượng vận động*** được?

Sau khi tìm hiểu ta lại phát hiện ra cách thể hiện một đối tượng vận động một bước từ (x1, y1) $\rightarrow$ (x2, y2) như sau:

- Ve obj(1) tại (x1, y2) với mầu c
- Xóa obj(1) tại (x1, y1)
- Vẽ obj(2) tại (x2, y2) với mầu c

Để xóa một đối tượng, ta vẽ lại đối tượng = mầu nền.

⇨ Tóm lại: Chúng ta một cách tự nhiên đã xây dựng chương trình từ dưới lên

+ Những lập trình viên đã quen thuộc ít nhiều với kỹ thuật đồ họa sẽ giải bài toán bằng phương pháp từ trên – xuống như sau, chẳng hạn:

Mức 0:

- Người chạy đến quả bóng
- Người đá quả bóng

Mức 1:

- Khi (người chưa chạm bóng)

  Người chạy một bước

- Dừng
  - Người đá quả bóng;
  - Bóng bay đi;

  ....

2.3.2 Làm mịn dần (tinh chế dần từng bước)

Như đã nói, phương pháp triển khai chương trình bằng các bước làm mịn dần (successive refinements) do Wirth đề xuất chính là tiền thân của phương pháp thiết kế chương trình từ trên – xuống (top – down design).

Ta đã biết, năm 1956, Chuyên gia tâm lý học George Miller đã chỉ ra rằng: Tại một thời điểm, chúng ta chỉ có khả năng tập trung vào khoảng 7 vấn đề khó khăn cần giải quyết (7 chunk). Phát biểu này được gọi là luật của Miller (Miller's Law) [Miller, 1956]. Tuy nhiên, một vấn đề cần giải quyết lại chia thành nhiều hơn 7 vần đề con cần giải quyết, để hạn chế lượng thông tin/các vấn đề giải quyết tại một thời điểm ta sử dụng kỹ thuật *"tinh chế từng bước" –" stepwise refinement"* [2].

Tinh chế từng bước là một kỹ thuật giải quyết vấn đề có thể được định nghĩa như công cụ để cắt tỉa các quyết định chi tiết đến các giai đoạn sau để tập trung vào các vấn đề quan trọng/các vấn đề chính. Như luật của Miller đã đưa ra, tại một thời điểm ta chỉ có thể tập trung giải quyết khoảng 7 khó khăn.

Tinh chế từng bước được vận dụng trong nhiều kỹ thuật kỹ nghệ phần mềm như, các kỹ thuật phân tích, thiết kế, cài đặt, kiểm thử và tích hợp.

Ví du: Xét phương pháp tinh chế từng bước áp để thiết kế cho vấn đề nhỏ sau đây.

Vấn đề/bài toán [2]:

Thiết kế một sản phẩm cập nhật file chủ tuần tự, file này chứa tên và địa chỉ của dữ liệu cần quản lý hàng tháng, *True Life Software Disasters*.

Cập nhật là thao tác có trong hầu hết các ứng dụng thuộc các miền khác nhau. Có 3 kiểu giao dịch mà phần mềm cần hỗ trợ: Các phép chèn, các phép sửa đổi, và các phép xóa, với các module mã giao dịch là 1,2 và 3 tương ứng. Các kiểu giao dịch là:

Kiểu 1: INSERT (Thêm bản ghi mới vào file chủ)

Kiểu 2: MODIFY (Sửa một bản ghi đang tồn tại)

Kiểu 3: DELETE (Xóa một bản ghi đang tồn tại)

Các giao dịch được sắp xếp theo thức tự alpha theo tên gọi. Nếu nhiều hơn một giao dịch được thực hiện trên một bản ghi cho trước, các giao dịch trên bản ghi đó được sắp xếp sao cho các phép chèn xảy ra trước các phép sửa đổi và các sửa đổi xảy ra trước các phép xóa.

Bước đầu tiên là thiết kế một giải pháp để cung cấp một file cụ thể về các giao dịch đầu vào, như chỉ ra trong hình 2.1. File này chứa 5 bản ghi (ít khi thực hiện cả hai phép sửa đổi và phép xóa một bản ghi tương tự cùng một thời điểm chạy).

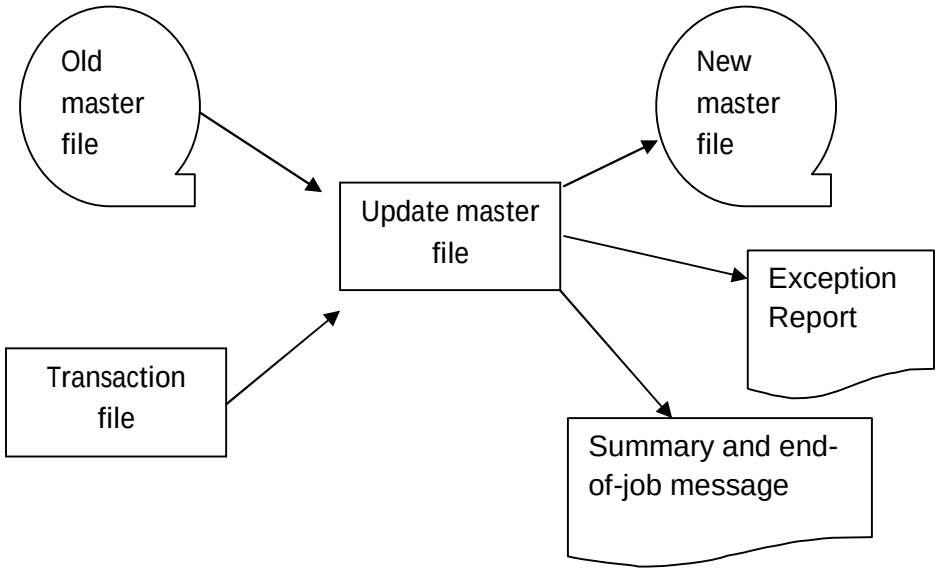
Hình 2.1: các bản ghi giao dịch đầu vào cho việc cập nhật file chủ tuần tự

Kiểu giao dịch	**Tên**	**Địa chỉ**
3	Brown	
1	Harris	2 Oak Lane, Townsville
2	Jones	Box 345, Tarrytown
3	Jones	
1	Smith	1304 Elm Avenue, Oak City

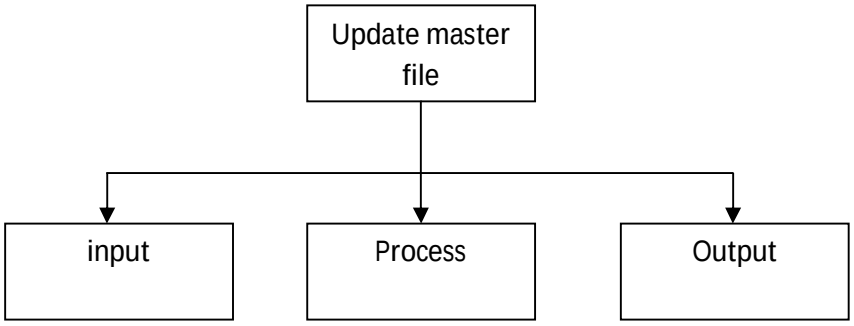
Vấn đề có thể được biểu diễn như trong hình 2.2. Có 2 file đầu vào:

1. Tên file chủ cũ và các bản ghi địa chỉ
2. File giao dịch

Hình 2.2: Sự biểu diễn của hoạt động cập nhất file chủ tuần tự/update sequential master file



Hình 2.3: bản tinh chế thiết kế đầu tiên

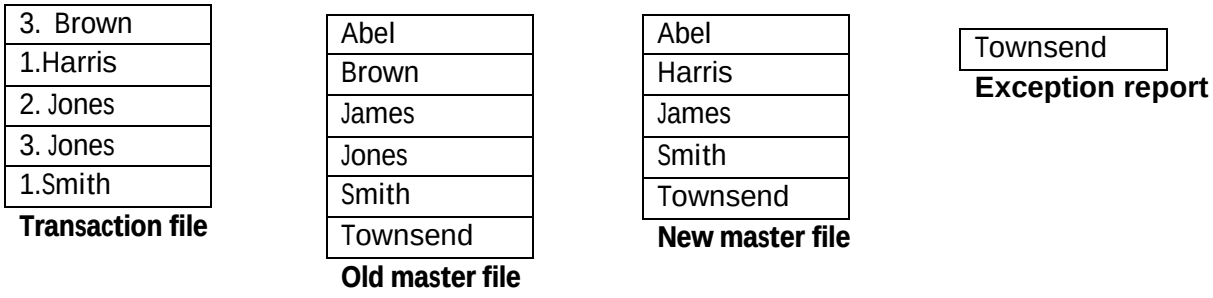


Và 3 file đầu ra:

1. Tên file chủ mới và các bản ghi địa chỉ
2. Báo cáo ngoại lệ
3. Tổng kết và thông điệp kết thúc công việc

Để bắt đầu tiến trình thiết kế, ta xuất phát từ hộp *update master file* chỉ ra trong hình 2.3, hộp này có thể được phân nhỏ thành 2 hộp con: input, process, và output. Với giả định các bản ghi dữ liệu đầu vào và đầu ra là đúng đắn và được tạo đúng thời điểm. Ta tập trung vào chức năng *process*, để xác định những gì cần làm, xét hình 2.4.

Hình 2.4: File giao dịch/ Transaction file; file chủ cũ/old master file, file chủ mới/new master file và thông báo ngoại lệ



Hoạt động xử lý *process* được tiến hành như sau:

Khóa của bản ghi giao dịch thứ nhất (Brown) được so sánh với khóa của bản ghi thứ nhất của file cũ (Abel), vì Brown đứng sau Abel, bản ghi Abel được viết vào file chủ mới, và bản ghi tiếp theo trong file chủ cũ được đọc (Brown). Trong trường hợp này, khóa của bản ghi giao dịch khớp với khóa của bản ghi trong file giao dịch, và vì kiểu của giao dịch là 3 (delete), do đó bản ghi Brown phải bị xóa. Điều này được thực hiện bằng cách không copy bảng ghi này sang file chủ mới. Bản ghi giao dịch tiếp theo được đọc (Harris) và bản ghi file chủ cũ (james) được đọc, Harris đứng trước James, do đó nó được

chèn vào file chủ mới. Bản ghi giao dịch tiếp theo Jones được đọc. Vì Jones đến sau James, James được chi vào file củ mới, và bản ghi tiếp theo trong file chủ cũ được đọc, đó là Jones. Từ file giao dịch, Jones được sửa sau đó bị xóa, bản ghi giao dịch tiếp theo được đọc (Smith) và bản ghi tiếp theo trong file chủ cũ được đọc (Smith. Không may thay, kiểu giao dịch là 1 (insert), nhưng Smith đã có trong file chủ. Do đó có một lỗi liên quan đến sắp xếp dữ liệu, và bản ghi Smith cho ra báo cáo ngoại lệ, bản ghi Smith trong file chủ cũ được ghi vào file chủ mới.

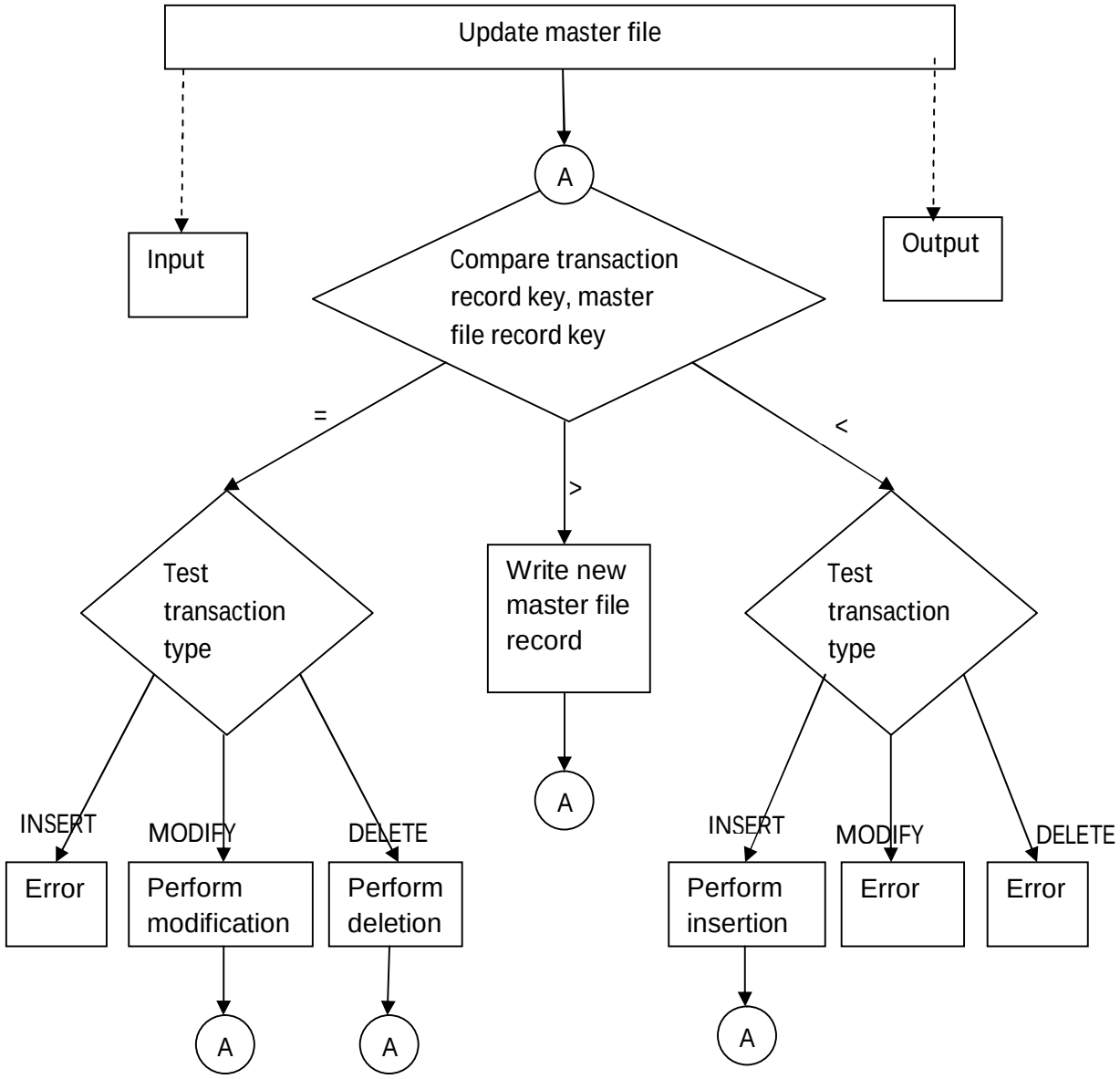
Đến đây *Process* đã được hiểu, và có thể được biểu diễn như trong hình 5.5.

Hình 2.5: Hình 2.5 Biểu diễn bằng biểu đồ của *process*

Khóa bản ghi giao dịch = khóa của bản ghi trong file giao dịch cũ	1. *Insert: In thông điệp lỗi* 2. *Modify: Thay đổi bản ghi file chủ* 3. *Delete: * Xóa bản ghi file chủ*
Khóa bản ghi giao dịch > khóa của bản ghi trong file chủ cũ	*Copy bản ghi file chủ cũ đến file chủ mới*
Khóa bản ghi giao dịch<khóa của bản ghi file chủ cũ	1. *Insert: Viết bản ghi giao dịch lên file chủ mới* 2. *Modify: In thông điệp lỗi* 3. *Delete: In thông điệp lỗi*
**Delete một bản ghi file chủ được triển khai bằng cách không copy bản ghi lên file chủ mới*	

Tiếp theo hộp *process* của hình 2.3 có thể được tinh chế, kết quả tinh chế lần thứ 2 được chỉ ra trong hình 2.6.

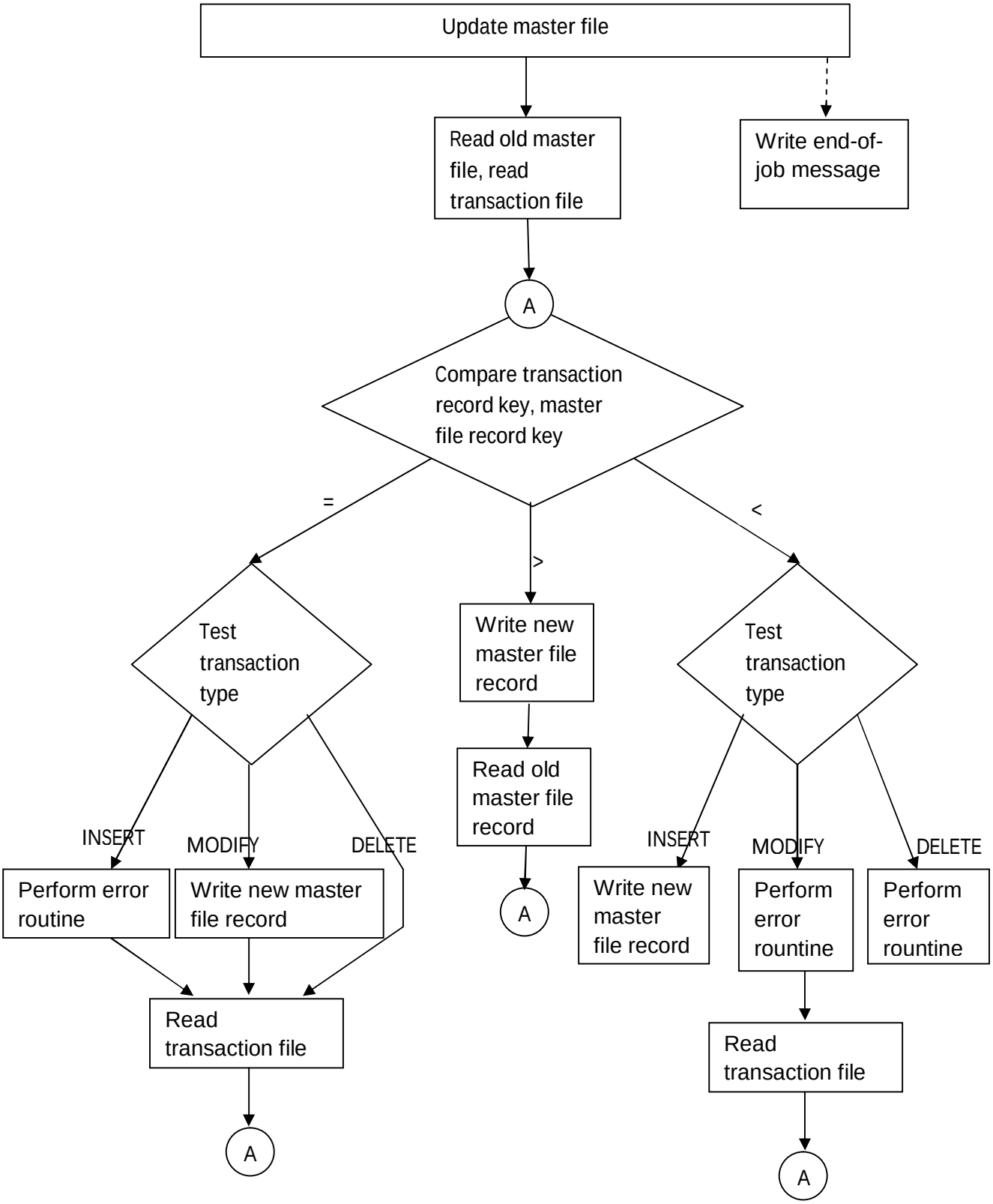
Hình 2.6: Kết quả tinh chế thiết kế lần 2



Các đường nét đứt đến các hộp input và output chỉ ra các quyết định làm thế nào để điều kiển đầu vào và đầu ra, các hộp này sẽ được trì hoãn đến lần tinh chế tiếp theo. Hình này ta tiến hành tinh chế hộp *Process* bằng cách đưa ra biểu đồ luồng công việc của nó. Trong biểu đồ chúng ta cũng chưa cung cấp điều kiện kết thúc file, cũng chưa đặc tả những gì cần làm khi gặp điều kiện lỗi. Điểm mạnh của tinh chế từng bước đó là các vấn đề này và các vấn đề tương tự có thể được giải quyết ở những lần tinh chế sau.

Bước tiếp theo ta tinh chế các hộp input và output của hình 2.6. Kết quả tinh chế được chỉ ra trong hình 2.7

Hình 2.7: Kết quả tinh chế bản thiết kế lần thứ 3 (bản thiết kế có một lỗi chủ điểm)



Các điều kiện kết thúc file, và việc viết thông điệp kết thúc công việc (end – of - job) vẫn chưa được điều khiển/xử lý. Nó lại được thực hiện ở lần tinh chế lặp lại sau. Bản tinh chế thiết kế ở hình 2.7 có một lỗi chủ điểm đó là. Xét tình huống dữ liệu ở hình 2. 5. Khi giao dịch hiện thời là **2 Jones**, hay sửa đổi Jones, và bản ghi file chủ cũ hiện thời là **Jones** . Trong thiết kế hình 5.7, vì khóa của bản ghi giao dịch là giống với khóa của bản ghi file chủ cũ, đường đi trái nhất của cây phân cấp được thực hiện để kiểm tra hộp quyết định *Test transaction type*. Vì kiểu giao dịch hiện thời là Modify, bản ghi file chủ cũ được sửa đổi và được viết vào file chủ mới, bản ghi tiếp theo được đọc. Bản ghi này là **3 Jones**, hay xóa Jones. Nhưng bản ghi Jones đã được sửa đổi đã được viết vào file chủ mới.

Người đọc có thể muốn biết, tại sao lại đưa ra bản tinh chế không đúng. Mục đích việc làm này là, khi sử dụng tinh chế từng bước, rất cần thiết phải thực hiện kiểm tra sự thành công của mỗi kết quả tinh chế trước khi thực hiện lần tinh chế tiếp theo. Nếu bản tinh chế có lỗi, cần phải thực hiện lại tiến trình từ đầu chứ không phải chỉ đơn thuần là quay lại bản tinh chế được đó và xử lý lại nó. Trong ví dụ này, bước tinh chế thứ 2 (hình 5.6) là đúng, vì thế nó được sử dụng làm cơ sở cho bước tinh chế thứ 3. Lúc này, bản thiết kế sử dụng level – 1 (nhìn ở trên), đó là bản ghi giao dịch chỉ được xử lý sau khi bản ghi giao dịch tiếp theo được phân tích, chi tiết về vấn này được chỉ ra bài tập 5.1[2]

Trong lần tinh chế thứ 4, các chi tiết bị lờ đi, bây giờ được quan tâm, như việc mở, đóng các file. Với tinh chế từng bước, các chi tiết được xử lý sau, sau đó sự logic của bản thiết kế được phát triển đầy đủ. Ta thấy, không thể chạy được sản phẩm nếu không thực hiện việc mở và đóng file. Điều quan trọng ở đây đó là trong từng trạng thái thiết kế, ta sẽ xử lý các chi tiết của việc mở và đóng file. Khi thiết kế đang được phát triển, chỉ khoảng 7 vấn đề mà người thiết kế có thể tập trung. Do đó việc mở và đóng file không được thực hiện khi thiết kế, mà chúng chỉ đơn thuần là các chi tiết cài đặt mà là một phần của bất kỳ thiết kế nào. Tuy nhiên, những lần tinh chế sau đó, việc mở và đóng file trở nên quan trọng. Theo các cách khác, bước tinh chế có thể được xem như một kỹ thuật vạch ra các ưu tiên của nhiều vấn đề biến đổi phải được giải quyết trong workflow. Tinh chế từng bước đảm bảo rằng mọi vấn đề được giải quyết và mỗi vấn đề được giải quyết ở thời điểm phù hợp, số lượng vấn đề được giải quyết ở mỗi bước không nhiều hơn 7+/- 2

Thuật ngữ tinh chế từng bước được giới thiệu đầu tiên bởi Wirth[1971]. Trong các case study trước, tinh chế từng bước được áp dụng cho biểu đồ luồng, ở đó Wirth đã áp dụng kỹ thuật này cho việc giả mã. Sự biểu diễn cụ thể mà sử dụng tinh chế từng bước là không quan trọng, tinh chế từng bước là kỹ thuật chung có thể sử dụng cho mọi luồng công việc và cho hầu hết mọi biểu diễn.

Sức mạnh của tinh chế từng bước đó là nó giúp kỹ sư phần mềm tập trung vào các khía cạnh liên quan của nhiệm vụ phát triển hiện thời và lờ đi các chi tiết, trì hoãn các chi

tiết này đến các bước tinh chế sau đó. Không giống như kỹ thuật chia để trị, ở đó, vấn đề tổng thể được chia thành các vấn đề con có mức độ quan trọng tương đượng nhau. Trong tinh chế từng bước, tầm quan trọng của một vấn đề thay đổi từ lần tinh chế này đến lần tinh chế khác.

2.4 Các phương pháp luận lập trình

2.4.1 Giới thiệu [6]

Phương pháp luận lập trình được biểu diễn bởi mô hình lập trình. Một mô hình lập trình thể hiện một quan điểm/một cách tiếp cận trong lập trình. Mỗi mô hình hỗ trợ một tập các khái niệm, các kỹ thuật và các nguyên tắc thiết kế. Các mô hình lập trình khác nhau có những mức độ diễn đạt khác nhau, các kỹ thuật lập trình khác nhau và các cách lập luận khác nhau về chúng.

Mọi mô hình lập trình đều có vị thế của nó, được thiết kế và sử dụng trong hoàn cảnh riêng. Một nguyên tắc quan trọng ta cần nhận biết đó là: Cần kết hợp giữa các mô hình để giải quyết triệt để một hệ thống ứng dụng phức tạp.

Một kiểu lập trình tốt yêu cầu sử dụng các khái niệm lập trình được kết hợp trong các mô hình khác nhau. Các ngôn ngữ lập trình chỉ cài đặt một mô hình tạo khó khăn cho việc tạo ra một chương trình tốt.

Ví dụ- Các ngôn ngữ hướng đối tượng: Khuyến khích sử dụng quá nhiều khái niệm về trạng thái và kế thừa => tạo sự rắc rối trong chương trình

- Các ngôn ngữ lập trình hàm: Khuyến khích sử dụng quá nhiều về lập trình thứ bậc cao hơn. Nó xâu mọi hàm trong chương trình thành một biểu thức. Điều này làm cho chương trình rắc rối, không đạt tính mô đun hóa.

- Các ngôn ngữ lập trình logic: Khuyến khích sử dụng nhiều cú pháp mệnh đề Horn và tìm kiếm. Các ngôn ngữ này định nghĩa chương trình như một tập các mệnh đề Horn (lắp ráp các mệnh đề logic theo kiểu if .. then). Điều này dẫn đến nhiều giải thuật trở nên rắc rối khi được viết theo kiểu này, tìm kiếm lưu vết luôn được sử dụng thậm chí một cách tự nhiên, hầu như không cần nó.

Hệ thống Mozart – OZ: là một phần mềm hỗ trợ cài đặt hiệu quả một số mô hình lập trình (Logic, Hàm, Hướng đối tượng, có cấu trúc, ràng buộc, phân tán, ...). Hệ thống này đã được phát triển trong một thời gian dài, bắt đầu từ năm 1991, nó được phát hành với đầy đủ mã nguồn mở. Các phát hành:

- Phiên bản đầu được phát hành năm 1995;
- Phiên bản hỗ trợ phân tán phát hành năm 1999

- Phiên bản Morart 1.3.0 phát hành năm 2003

- Hiện nay đã phát hành phiên bản 1.4.0

Bạn đọc có thể vào Website http://www.mozart-oz.org để biết thêm các thông tin chi tiết. Mozart – System là một môi trường phát triển tăng dần và cung cấp đầy đủ các công cụ trợ giúp phát triển ứng dụng. Mozart – OZ được cài đặt bằng ngôn ngữ OZ.

Các mô hình lập trình được liệt kê trong bảng sau:

Phương thức lập trình		
Mệnh lệnh	1	Lập trình hướng thiết bị - Gear Oriented Programming
	2	Lập trình hướng công tắc- Switch Oriented Programming
	3	Lập trình hướng thủ tục – Procedural/structured Programming
	4	Lập trình hướng đối tượng – Object Oriented Programming
	5	Lập trình hướng lát cắt – Aspect Oriented Programming
	6	Lập trình hướng cấu phần – Component Oriented Programming
	7	Lập trình hướng dịch vụ - Service Oriented Programming
	8	Điện toán đám mây – Clound Computing
Khai báo	1	Lập trình logic – Logic Programming
	2	Lập trình hàm – Functional Programming
	3	Lập trình CSDL – Database Programming

+ Với phương thức lập trình mệnh lệnh:

- Người lập trình phải tìm cách diễn đạt được thuật toán, Chỉ ra cách thức làm thế nào để giải quyết bài toán đã cho.
- *Ưu điểm:* Hiệu quả trong lập trình, vì người lập trình có thể tác động trực tiếp vào phần cứng
- *Hạn chế*: Chương trình không có khả năng suy đoán, không có trí tuệ.

+ Với phương thức lập trình khai báo:

- Người lập trình xây dựng cơ sở tri thức, các yêu cầu tính toán, truy vấn dựa trên các khai báo để giải quyết bài toán
- *Ưu điểm*: Chương trình có khả năng suy diễn
- *Hạn chế*: Khó cài đặt và vận hành hơn so với chương trình mệnh lệnh.

2.4.2 Lập trình hướng thiết bị

Lập trình để thay đổi thiết bị, các khớp và các chuyển động. Qua các cử động vật lý của thiết bị và giới hạn trong thiết bị này, một tính toán mới có thể được thực hiện. Phương thức lập trình này thường được sử dụng trong lĩnh vực điều khiển tự động. Thiết bị ở đây không nhất thiết phải là máy tính. Phương thức lập trình này đòi hỏi lập trình viên phải có khả năng vật lý tốt và phải có trí tuệ. Có thể dùng phương thức lập trình này trong việc xây dựng các phần mềm ứng dụng nhúng.

2.4.3 Lập trình hướng công tắc

Máy tính điện tử đầu tiên là ENIAC (Electronic Numerical Integrator And Computer), ra đời năm 1942. Lập trình trên máy tính có nghĩa là điều chỉnh các công tắc và lắp ráp lại toàn bộ hệ thống.

Ngôn ngữ lập trình là ngôn ngữ máy (machine language). Chương trình hoàn toàn phụ thuộc vào kiến trúc phần cứng máy tính và những quy ước khắt khe của nhà chế tạo phần cứng.

Để giảm nhẹ khó khăn cho hoạt động lập trình từ những năm 1950, ngôn ngữ hợp dịch/hợp ngữ (Assembly) còn gọi là ngôn ngữ biểu tượng (Symbolic) ra đời. Trong hợp ngữ, các mã lệnh và địa chỉ của các toán hạng được thay thế bằng các từ tiếng anh gợi nhớ như: ADD, SUB, MUL, DIV, JUMP, ... tương ứng với các phép toán +, - , *, /, chuyển điều khiển, ...Cũng từ những năm 50, người ta đưa vào kỹ thuật chương trình con (subprogram/sub-rountine) và xây dựng những thư viện chương trình con để khi cần thì gọi đến hoặc sử dụng lại những đoạn chương trình đã viết.

Các ngôn ngữ bậc thấp như ngôn ngữ máy, hợp ngữ thường chỉ được dùng để viết các chương trình điều khiển & kiểm tra thiết bị, các chương trình gỡ rối (debugger) hay các công cụ...

2.4.4 Lập trình có cấu trúc

Lập trình cấu trúc là nguyên lý chủ đạo trong CNPM. Theo nguyên lý này ta sử dụng rộng rãi khái niệm trừu tượng hóa nhằm mục đích phân rã bài toán thành những bài toán nhỏ hơn để dễ dàng triển khai và đảm bảo tính đúng đắn của chương trình.

Ví dụ: Triển khai chương trình phân số đã đề cập ở phần 2.3.1

Phương pháp này ra đời cùng với sự ra đời của các ngôn ngữ lập trình bậc cao như: Pascal, C, Basic, ...hỗ trợ cho phương pháp này. Tính cấu trúc của phương pháp và của ngôn ngữ thể hiện ở cấu trúc điều khiển, cấu trúc dữ liệu và cấu trúc của chương trình...

a. Cấu trúc lệnh, lệnh có cấu trúc

(1) Cấu trúc lệnh (Cấu trúc điều khiển)

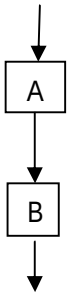
- Mỗi module chương trình máy tính về bản chất là một bản mã hóa thuật toán. Đến lượt mình, thuật toán được coi là một dãy hữu hạn các thao tác trên các đối tượng nhằm thu được một số kết quả sau một số hữu hạn bước

- Các thao tác trong một chương trình cụ thể được điều khiển bởi các lệnh hay các cấu trúc điều khiển, còn các đối tượng chịu sự tác động của thao tác thì được mô tả và được kiến trúc thông qua các Cấu trúc dữ liệu. Trong Pascal ta biết các cấu trúc điều khiển sau:

+ Cấu trúc tuần tự

A;

B;



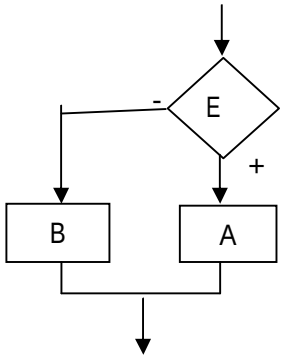
Sau khi thực hiện A thì thực hiện B

+ Cấu trúc rẽ nhánh dạng đầy đủ

If E

Then A

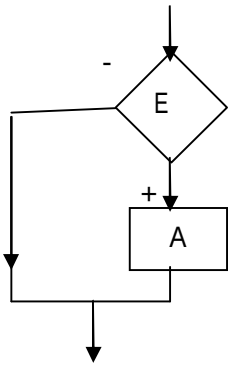
Else B;



+ Cấu trúc rẽ nhánh dạng khuyết

If E

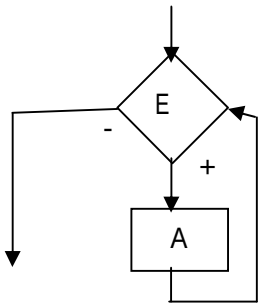
Then A



+ Cấu trúc lặp với điều kiện trước

While E do

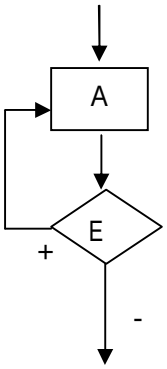
A;



+ Cấu trúc lặp với điều kiện sau

If E

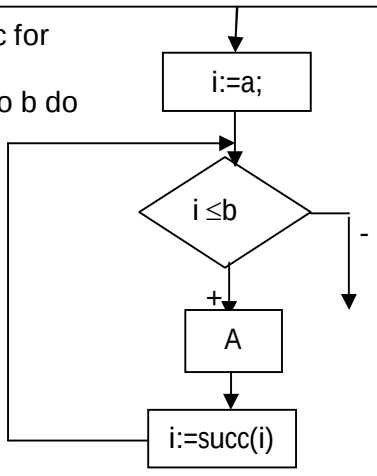
Then A



+ Cấu trúc for

For i:=a to b do

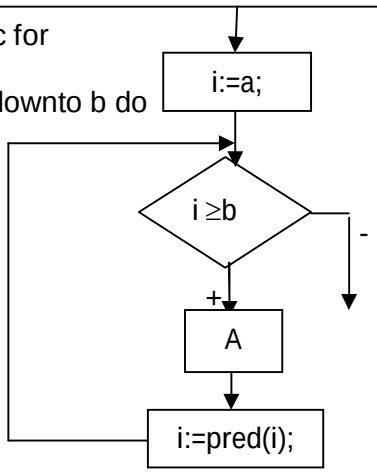
A;



+ Cấu trúc for

For i:=a downto b do

A;



Ngoài ra, còn một số cấu trúc khác như cấu trúc chọn (case), cấu trúc tạo khối (đặt các lệnh trong cặp Begin và end), cấu trúc gọi thủ tục và hàm, cấu trúc rẽ nhánh vô điều kiện (goto).

Các nhà lập trình có kinh nghiệm khuyên rằng có thể và nên xây dựng chương trình với 3 cấu trúc điều khiển cơ bản là *tuần tự, rẽ nhánh* và *lặp*. Vì chúng khá đơn giản, trong sáng và tự nhiên và dùng các cấu trúc này để kiểm soát được tính đúng đắn của chương trình.

⇨ Vậy nói chung thì cần bao nhiêu cấu trúc điều khiển là đủ thể hiện các thuật toán. Định lý sau đây cho biết chỉ cần 2 là đủ.

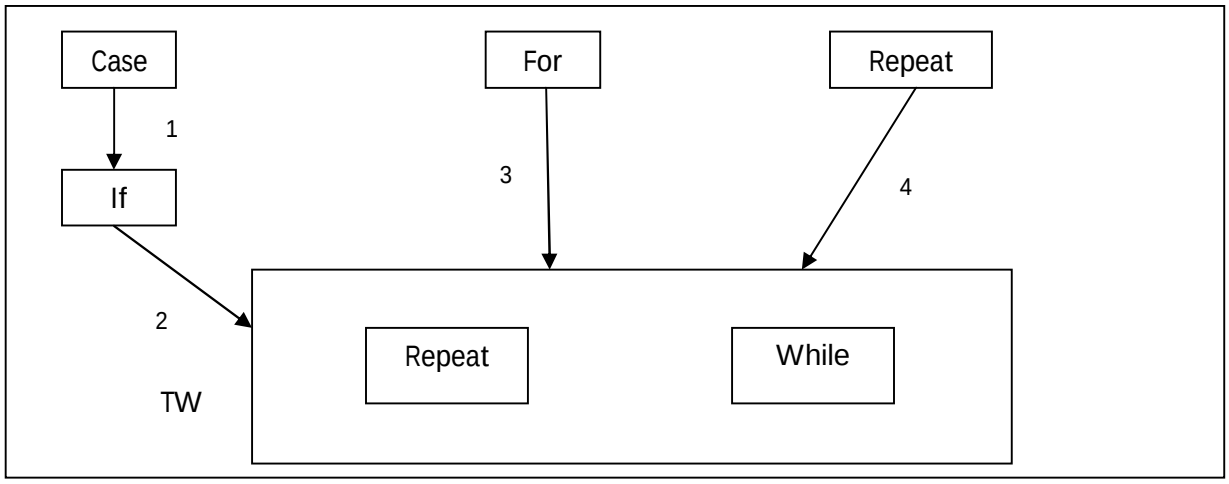
Định lý (Boehm C., Jacopini G., 1966)

Với mọi chương trình viết dưới dạng sơ đồ khối P (Flowchart) đều tồn tại một chương trình Q tương đương P theo nghĩa sau đây:

- *Với mọi cái vào x ở miền chấp nhận được (miền xác định) ta luôn có P(x)=Q(x). Nói cách khác 2 chương trình trên biến đổi những cái vào như nhau thành những cái ra như nhau*
- *Các thao tác trên các biến của Q là giống như của P*
- *Các biến của Q cũng là các biến của P, có thể Q thêm một vài biến logic*
- *Chương trình Q chỉ sử dụng duy nhất 2 cấu trúc điều khiển là lặp với điều kiện trước và tuần tự.*

Chúng ta sẽ minh họa khẳng định trên bằng cách chuyển các cấu trúc điều khiển của Pascal về 2 cấu trúc điều khiển là tuần tự và lặp với điều kiện trước. Riêng cấu trúc goto chúng ta không quan tâm vì lập trình viên hiện đại hoàn toàn không sử dụng đến nó.

Sơ đồ chuyển là như sau:



Case	⟹ *If*
Case i of *a: A;* *b: B;* *.....* *y: Y;* *End; {case}*	*If i=a then A* *else* *If i=B then B* *Else* *If.....* *Else if i=y then Y*

2. <u>Từ If → TW</u>

Chúng ta sẽ sử dụng hai biến phụ kiểu logic:

Var c, b : Bolean để thực hiện vòng lặp while một lần

- *If E then A* sẽ được chuyển thành:
  - *b:= E;*
    While b do
    Begin
    A;
    b:= NOT b;
    End;
- *If E then A else B* sẽ được chuyển thành:
  - *b:= E; c:=b;*
    while b do
    begin
    A;
    b:= not b;
    end;
    while not c do
    begin
    B;
    c:= not c;
    end;

3. <u>Từ For → TW</u>

- *For i:=a to b do A* sẽ được chuyển thành:
  - *i:= a;*
    While i<=b do
    Begin
    A;

```
        i:= succ(i);
    End;
```

- *For i:= b downto a do A* sẽ được chuyển thành:
  - ```
    i:= b;
    While i>= a do
        Begin
            A;
            i:= pred(i);
        End;
    ```

4. **Từ Repeat → TW**

   Repeat A until E sẽ được chuyển thành

   ```
   A;
   While not E do A;
   ```

Lưu ý:

Khi sử dụng các cấu trúc lặp, ta phải thận trọng đến tính hữu hạn của số lần lặp. Các thao tác trong vòng lặp có thể làm cho vòng lặp trở nên vô hạn. Sau đây là một vài tình huống vòng lặp không kết thúc.

Bài toán 1 (Niêu cơm Thạch Sanh)

Giả sử niêu cơm Thạch Sanh chứa n bát (n>0). Thạch Sanh chia cơm cho hàng binh theo quy tắc 3 bước sau đây:

Bước 1: Lấy từ trong nồi ra một nửa số cơm

Bước 2: Bớt trở lại một bát vào nồi

Bước 3: Chia số cơm đã lấy ra cho hàng binh

Tất cả các thao tác trên đều được thực hiện trên tập số nguyên không âm. Hãy tính xem, với mỗi n cho trước, sau bao lượt chia cơm thì số cơm trong nồi không đủ chia tiếp được nữa (không còn để chia nữa).

Cách làm:

Giả sử ta giải bài toán trên = một chương trình Pascal gồm 3 thủ tục:

- Nạp dữ liệu {n bát} (1)
- Chia cơm ; {Tính số lượt chia cơm: k} (2)
- Thông báo kết quả ; {số k tìm được} (3)

Thủ tục (1)&(3) là đơn giản. Xét thủ tục (2) – Chia cơm theo quy tắc 3 bước ở trên

Mức 0
   ```

k:= 0; {đếm số lượt chia cơm}

While (n còn chia được) do

Begin

k:= k+1;

Chia và tính số cơm còn lại trong nồi ghi vào n;

End;

Sau khi lấy một nửa số cơm trong nồi, ta còn phải bớt lại một bát, do đó n phải thỏa mãn điều kiện ***n div 2 >=1***. Giải ra ta được ***n>=2***. Đây chính là điều kiện lặp cho vòng while

Mức 1 *(Chia cơm mịn hơn)*

k:=0;

While n>=2 do

Begin

k:= k+1;

c:= (n div 2)-1; {số cơm lấy ra}

n:= n –c; {số cơm còn lại trong nồi}

End;

Mức 2 *(Đặc tả mịn)*

Đặc tả các thủ tục (1), (2), (3)

⇨ Viết chương trình chia cơm bằng Pascal để chạy

Nhận xét:

Chương trình này không dừng với mọi n>1. Như vậy Niêu cơm Thạch Sanh có thực trong đời thường. Ta thử thủ tục chia cơm với n = 40 bát lúc đầu. Bảng sau đây cho thấy n giảm dần từ 40 → 3, sau đó n giữ mãi giá trị 3 này. Như vậy, sau khi lấy ra được 37 bát cơm (sau 6 lần chia) thì số cơm trong nồi trở thành một số bất biến (n=3) vì n>2 nên vòng lặp while sẽ thực hiện mãi không dừng

Dĩ nhiên, nếu ta sửa lại điều kiện kết thúc cho while là n>2 thì mọi việc sẽ tốt đẹp. Điều này sẽ được chứng minh trong chương cuối.

k	***n***	***c***
0	40	19
1	21	9
2	12	5
3	7	2
4	5	1
5	4	1
6	3	0
7	3	0
8	3	0
...	....	...

Bài toán 2 (Chúc rượu)

Trên bàn tiệc có 2655 cái cốc, 1/5 trong số đó được đặt úp, số còn lại đặt ngửa. Năm người phục vụ, mỗi người cầm ngẫu nhiên 2 cái cốc và đảo cốc như sau: Cốc úp đảo thành ngửa, cốc ngửa đảo thành úp. Thời gian đảo một cốc là 15 giây. Hãy tính xem trong bao lâu thì toàn bộ số cốc sẽ được lật ngửa[1].

Bài tập này dành cho bạn đọc. Bạn đọc hãy phát hiện tính bất biến của vòng lặp khi những người phục vụ thực hiện đảo cốc là gì?

(2) Lệnh cấu trúc

Lệnh có cấu trúc là các cấu trúc điều khiển cho phép chứa các cấu trúc điều khiển khác bên trong nó. Khi tìm hiểu một cấu trúc điều khiển cần xác định rõ vị trí được phép đặt một cấu trúc điều khiển khác trong nó.

Ví dụ: Cấu trúc If trong Pascal có dạng:

If E then A else B;

Tại A, B có thể đặt các cấu trúc điều khiển khác. Ví dụ

```
If E then
        While E1 do C
Else
        Repeat
                D
        Until E2
```

Tại C, D lại có thể đặt các cấu trúc điều khiển khác.....

Bằng cách "thế" các cấu trúc như trên, chương trình sẽ ngày càng phức tạp và có nguy cơ trở thành khó đọc. Chính vì thế ta cần:

- Chú trọng triển khai chương trình từ trên xuống

- Viết các cấu trúc lệnh thành các khối "nhô – thụt" để nhấn mạnh phạm vi của cấu trúc & thể hiện đúng mức độ lồng nhau của các cấu trúc. Nguyên tắc viết chương trình theo cấu trúc là:

Nguyên tắc viết chương trình theo cấu trúc

Cấu trúc con phải được viết lọt vào trong cấu trúc cha. Điểm vào và điểm ra của cấu trúc phải nằm trên cùng một cột

b. Cấu trúc dữ liệu (Data Structures)

Chúng ta có thể đặt ra câu hỏi: Vì sao trong tin học lại xuất hiện CTDL & các cấu trúc điều khiển?

Tin học được xây dựng trên nền tảng của khái niệm toán học về thuật toán. Đến lượt mình, thuật toán quan tâm đến thao tác và các đối tượng. Có thể nêu vắn tắt những vấn đề nghiên cứu của lý thuyết thuật toán như sau:

1. Các thuật toán (máy giải, máy trừu tượng) xử lý những đối tượng nào (CTDL)?
2. Cách xử lý ra sao? (Các thao tác& các cấu trúc điều khiển thao tác)
3. Độ phức tạp của biểu diễn dữ liệu & thao tác là bao nhiêu (tốn bao nhiêu miền nhớ, miền nháp và kết quả trung gian, thời gian xử lý).

Như vậy:

Cấu trúc điều khiển được nẩy sinh từ nhu cầu diễn đạt thuật toán. Người ta có thể đặt ra nhiều cấu trúc điều khiển, tuy nhiên những người sáng tạo ra những ngôn ngữ lập trình thường chọn 3 cấu trúc điều khiển trong sáng, đơn giản là cấu trúc lặp, tuần tự và rẽ nhánh.

Cấu trúc dữ liệu đa dạng hơn cấu trúc điều khiển vì chúng nẩy sinh do mô phỏng các đối tượng thực tế. Ví dụ cây, đồ thị, tập hợp, ...

Tin học tiếp thu một số CTDL "kinh điển" của toán học như: Cây, đồ thị, ma trận...và cũng sáng tạo ra các CTDL hết sức độc đáo như: Ngăn xếp, hàng đợi, danh sách, tệp tin, ...Tệp tin và ngăn xếp là hai cấu trúc không thể thiếu được khi tổ chức chương trình dịch và thực hiện chương trình đã dịch. Khi giải một bài toán cụ thể, khéo léo chọn

cấu trúc dữ liệu sẽ giúp ta diễn đạt thuật toán một cách thuận lợi. Đó chính là nội dung của công thức nổi tiếng do Wirth – tác giả của ngôn ngữ lập trình Pascal đề xuất:

Program = Algorithms + Data Structures

Ý nghĩa chính của công thức này thể hiện ở chỗ: Thuật toán thực hiện các thao tác trên CTDL. CTDL và thuật toán phải tương thích với nhau

Ví dụ minh họa công thức trên. Ta xét bài toán sau

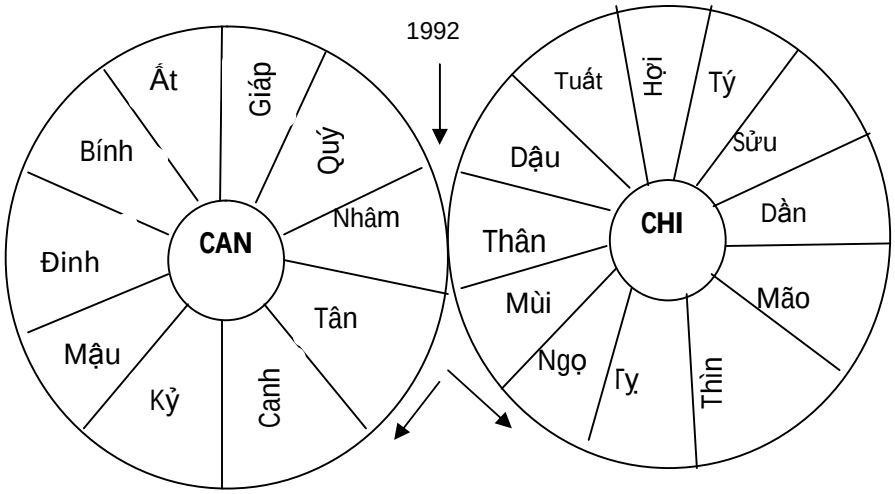
Bài toán số 3

Năm 1992 là năm Nhâm Thân theo Âm lịch. Hãy viết chương trình Pascal nhập vào một năm dương lịch và thông báo lên màn hình tên của năm âm lịch tương ứng.

Cách làm

Năm âm lịch được tính theo Can và Chi. Có 10 can là Giáp, Ất, Bính, Đinh, Mậu, Kỷ, Canh, Tân, Nhâm, Quý và 12 chi là: Tý, Sửu, Dần, Mão, Thìn, Tỵ, Ngọ, Mùi, Thân, Dậu, Tuất, Hợi ứng với 12 con giáp là Chuột, Trâu, Hổ, Mèo, Rồng, Rắn, Ngựa, Dê, Khỉ, Gà, Chó.

Khi tính năm âm lịch, người ta xếp Can và Chi theo vòng tròn. Mỗi năm quay đi một hình quạt. Xét 2 bánh xe Can và Chi như hình vẽ:



Nếu quay một bánh xe theo chiều mũi tên, thì bánh xe thứ 2 sẽ quay theo nhờ ma sát. Theo hình vẽ ta thấy, nếu năm 1992 là năm Nhâm Thân thì năm:

- 1993 là Quý Dậu
- 1994 là Giáp Tuất
- .....

Khi quay bánh xe theo chiều ngược lại ta tính được:

- Năm 1991 là năm Tân Mùi
- Năm 1990 là năm Canh ngọ
- .....

Quản lý dữ liệu vòng tròn được tổ chức tuyến tính và truy cập bằng hàm lấy số dư. Gọi Can[0..9] và Chi[0..11] là 2 mảng lưu trữ tương ứng cho Can & Chi, ta có thể gán như sau:

Can[0]:= "Nhâm"; Can[1] := "Quý"; Can[2]:= "Giáp"; .....Can[9]:= "Tân".

Chi[0]:= "Thân"; Chi[1]:= "Dậu"; Chi[2]:= "Tuất"; .....Chi[11]:= "Mùi".

Như vậy, năm *199*2 sẽ tương ứng với *Can[0]và Chi[0].* Từ đó, ta có công thức:

Năm *x* sẽ ứng với: *Can[Dư(x-1992,10)] và*

Chi[Dư(x-1992,12)]

Trong đó: Dư(a,b) là hàm lấy số dư của phép chia nguyên a cho b. Ví dụ Dư(18, 10)=8; Dư (-27, 5)=3.

⇨ Chương trình giải bài toán:

```
Begin
    Repeat
        Write("Năm dương lịch x= ");
        Readln(x);
        Writeln(Can[Du(x-1992,10)], ' ', Chi[Dư(x-1992,12)]);
    Until n=0;
End.
```

⇨ Lựa chọn cấu trúc dữ liệu phù hợp với giải thuật sẽ rất thuận lợi cho việc cài đặt giải thuật.

<u>*Tóm lại.*</u>

Triển khai chương trình một cách có cấu trúc, lắp ráp chương trình từ các khối, các đơn thể được mô tả trong sáng, có chức năng rõ ràng, đặt tên cho các biến và các thủ tục một cách tường minh (tự nhiên) kèm theo những chú giải xúc tích, nêu bật được hình trạng tại mỗi thời điểm thao tác, ta sẽ thu được một chương trình mà:

- Ai đọc cũng có thể hiểu

- Khi cần sửa chữa chức năng, nó giúp ta chọn đúng khối chức năng đó để sửa

- Khi gặp lỗi, giúp nhanh chóng xác định được lỗi đó phát sinh ở chỗ nào, khoanh vùng để nắm bắt và sửa nó.

- Chương trình được viết theo cấu trúc vừa phục vụ tốt cho việc xác định tính đúng đắn, vừa thuận lợi cho việc kiểm chứng tự động.

2.4.5 Lập trình hướng đối tượng

- Nguyên tắc chủ yếu để làm chủ độ phức tạp của chương trình là triển khai chương trình theo mức và nguyên tắc này được sử dụng rất rộng rãi

- Khi mô tả các kiểu dữ liệu phù hợp với các đối tượng của thế giới thực, người ta rút ra 3 nhận xét hết sức quan trọng sau đây:

Nhận xét 1

Không thể tách rời CTDL với các thao tác trên cấu trúc ấy. Dữ liệu được phát sinh để chịu sự biến đổi, để được xử lý, nghĩa là chúng là đối tượng của thao tác. Khi mô tả một đối tượng là liệt kê các tính chất của đối tượng. Các tính chất này được phân làm 2 loại:

Nhóm 1: Khuôn dạng, sự tổ chức, kiến trúc của đối tượng, tên, kích thước, xuất sứ, ...

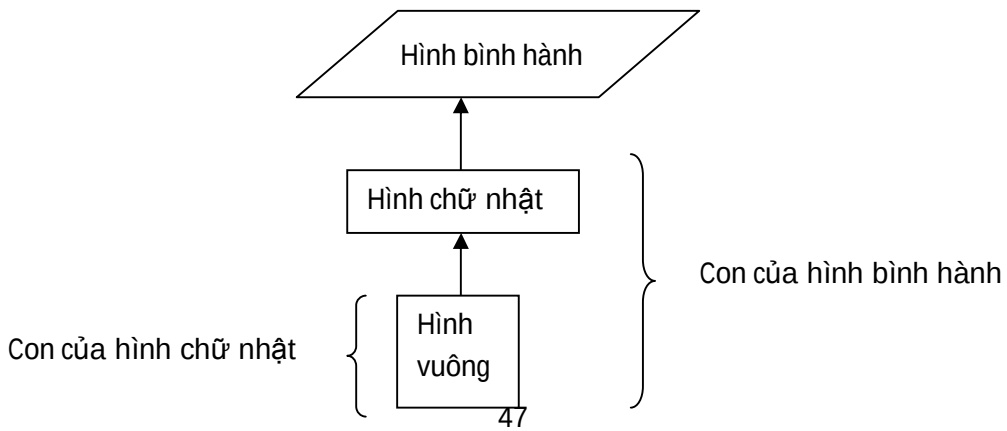
Nhóm 2: Sự vận động của đối tượng và các tương tác với các đối tượng khác.

Nhận xét 2

Nhiều kiểu dữ liệu thực ra chỉ là sự kế thừa và mở rộng từ các kiểu dữ liệu trước đó. Ví dụ:

- Kiểu hình chữ nhật (HCN) $\overset{\Delta}{=}$ là hình bình hành có một góc vuông
- Kiểu hình vuông (HV) $\overset{\Delta}{=}$ Là hình chữ nhật có 2 cạnh liên tiếp = nhau
- ....

=> HCN và HV là những kiểu con của hình bình hành. Trong đó, hình vuông lại là kiểu con của hình chữ nhật.



Trật tự phân cấp cha – con cho các kiểu, ngoài thuận lợi tiết kiệm cho việc mô tả, ta còn thấy một ưu thế tuyệt với đó là: Hình ảnh phân cấp làm giảm đáng kể tính phức tạp của vấn đề so với những mô tả lộn xộn, bình đẳng,

Nhận xét 3

Sự xuất hiện và mất đi của một đối tượng cần được mô tả một cách tường minh. Sẽ rất khó khăn cho việc quản lý các đối tượng nếu ta không biết trước được sự tồn tại của chúng

=> 3 nhận xét trên đã được thể hiện trong phương pháp lập trình mới – Lập trình hướng đối tượng.

Phương pháp hướng đối tượng là sự kế tục và phát triển tự nhiên của phương pháp lập trình có cấu trúc. Sự mở rộng được thể hiện như sau:

(1) Mô tả cấu trúc và các thao tác cho các phần tử của một kiểu phải được thực hiện tường minh vào lúc mô tả kiểu

(2) Các đối tượng con (lớp con) của một đối tượng (lớp cha) cho trước được mô tả bằng cách chỉ rõ cha của nó và các thao tác, các thuộc tính riêng của nó

(3) Mỗi đối tượng cần được tạo lập và hủy bỏ một cách tường minh. Việc truy cập tới các đối tượng & các thành phần của chúng được thực hiện giống như truy cập đến các biến bản ghi và các trường của nó.

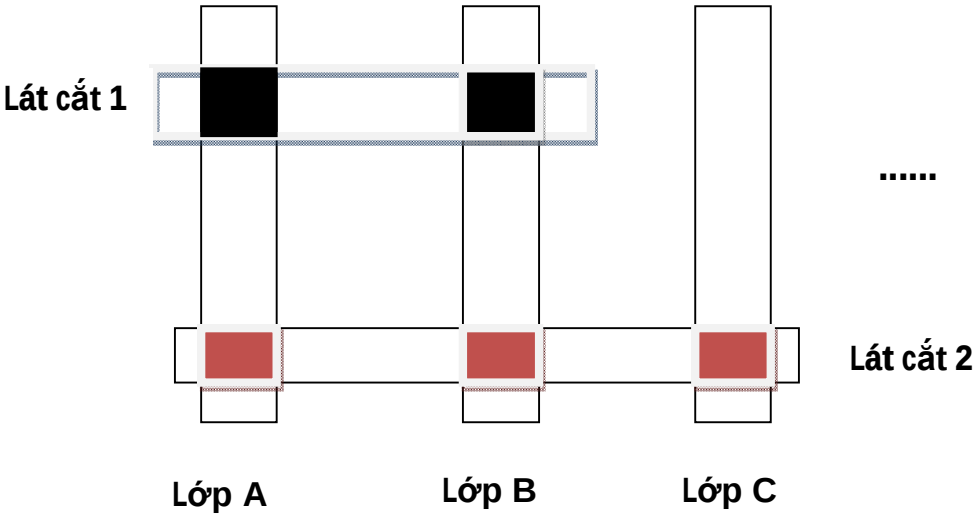
2.4.6 Lập trình hướng lát cắt

Khoảng 7 năm trở lại đây, một khuynh hướng lập trình mới xuất hiện được gọi là lập trình hướng lát cắt – AOP (Aspect Oriented Programming). AOP được xem là phương pháp bổ sung cho OOP – Chỗ mà OOP còn thiếu sót trong việc tạo ra những ứng dụng phức tạp.

OOP hiện là môt hình được lựa chọn hầu hết cho các dự án phát triển phần mềm. OOP rất hữu hiệu trong việc lập trình mô hình hóa hành vi chung của các đối tượng. Tuy nhiên nó không giải thích thỏa đáng những hành vi liên quan đến nhiều đối tượng. AOP giải quyết được vấn đề này và rất có thể là bước phát triển tiếp theo trong phương pháp lập trình.

Kỹ thuật OOP rất suất sắc trong việc đóng gói các hành vi vào trong các chủ thể (lớp đối tượng), miễn là chúng hoàn toàn tách biệt nhau. Tuy nhiên, trong các bài toán thực tế, thường có những hành vi đan xen nhau, liên quan đến nhiều lớp. Trong thực tế C++/Java đều không hỗ trợ đóng gói những hành vi đan nhau. Dẫn đến, các mã chương trình này có thể nằm lẫn lộn, rải rác và rất khó quản lý.

AOP là phương pháp lập trình mới, cho phép đóng gói những hành động liên quan đến nhiều lớp vào trong các khối. Mỗi khối này được gọi là một "aspect", tạm dịch là "lát cắt". AOP được xem là phương pháp bổ sung cho OOP, cho phép giải quyết những bài toán phức tạp tốt hơn, hiệu quả hơn. Các *lát cắt* của hệ thống có thể thay đổi, thêm hoặc xóa lúc biên dịch và có thể tái sử dụng.



AOP ıa ky thuật mới, đầy triển vọng, hứa hẹn đem lại nhiều lợi ích cho việc phát triển phần mềm:

- Mô đun hóa những vấn đề đan nhau: AOP xác định các vấn đề một cách tách biệt, hạn chế tối thiểu các nhập nhằng mã bằng cách cho phép mô đun hóa cả những vấn đề liên quan đến nhiều lớp.

- Dễ dàng phát triển hệ thống: Việc thêm các chức năng mới có thể thực hiện dễ dàng bằng cách tạo các *lát cắt* mới mà không cần quan tâm đến các vấn đề đan nhau khác. Khi thêm các mô đun mới vào hệ thống\ các lắt cắt cũ sẽ đan kết chúng và tạo nên sự phát triển chặt chẽ.

- Cho phép thiết kế tương lai: Một thiết kế tốt phải tính cả đến các yêu cầu hiện tại và tương lai. Việc các định các yêu cầu tương lai là một công việc khó khăn, nhưng nếu bỏ sót các yêu cầu tương lai có thể ta sẽ phải thay đổi hoặc làm lại nhiều phần hệ thống. Với AOP, người thiết kế hệ thống có thẻ để lại các quyết định thiết kế cho những yêu cầu tương lai nhờ thực hiện theo các *lát cắt* riêng biệt.

- Tái sử dụng mã tốt hơn: Các *lát cắt* là những mô đun riêng biệt, được kết hợp linh động. Đây chính là yếu tố quan trọng để tái sử dụng mã.

Bạn đọc có thể tìm hiểu về AOP kỹ hơn tại địa chỉ aosd.net

Ngôn ngữ cho AOP:

- AspectJ: Được xây dựng bởi Xerox PARC, là mở rộng từ ngôn ngữ Java
- Aspect Werkz
- Hyper/J
- JAC
- RROSE
- JMangler

.....

2.4.7 Lập trình hướng cấu phần

Lập trình hướng cấu phần là kiểu lập trình có xu hướng chia hệ thống phần mềm thành những thành phần giữ các chức năng khác nhau (mỗi thành phần này được gọi là một bộ phận hợp thành) mà khi kết hợp lại ta thu được một hệ thống phần mềm hoàn chỉnh.

Với COP, chương trình được xây dựng bằng cách lắp ráp các thành phần phần mềm có thể sử dụng lại, các khối tự chứa mã máy (hay còn gọi là các khối .thành phần thực hiện). Các thành phần này gồm các thành phần giao diện, các kết nối. COP nẩy sinh xuất phát từ thực tế rằng mọi thứ có cấu trúc đều được tạo nên từ các thành phần khác. Điển hình như trong nền công nghiệp tự động, các hệ thống được cấu tạo từ các thành phần. Ví dụ, để phát triển một chiếc ô tô là rất phức tạp.

Ô tô = {Các thành phần được ghép nối};

Các thành phần cấu tạo nên ô tô là thuộc nhiều loại khác nhau, kích cỡ khác nhau, chức năng khác nhau, được sản xuất bởi các nhà sản xuất khác nhau. Các thành phần này giới hạn từ các ốc vít rất nhỏ đến các hệ thống con phức tạp hơn như các động cơ, các bộ truyền phát nhanh, ...

Trong công nghiệp phần mềm. Sản phẩm vẫn làm bằng tay là chủ yếu. Điều đó dẫn đến tính năng của sản phẩm thấp, chất lượng không đảm bảo, khó tái sử dụng, ...

Trong công nghệ phần cứng. Mọi sản phẩm cũng được tạo ra dựa trên phương pháp hướng cấu phần. Thực tế cho thấy nền công nghiệp này phát triển rất nhanh, thu được nhiều lợi nhuận. Đây chính là lý do tại sao COP lại quan trọng.

COP sử dụng nhiều khái niệm của OOP nhưng hai phương pháp này là độc lập nhau. COP phát triển phần mềm bằng cách lắp ráp các thành phần trong khi OOP nhấn mạnh đến các lớp và các đối tượng. COP nhấn mạnh giao diện và kết cấu, trong khi OOP nhấn mạnh về cài đặt viết mã. COP không cần biết bất cứ kiến thức nào về cách thức một thành phần cài đặt giao diện của chúng, nó xem thành phần như một hộp đen (không bị ảnh hưởng bởi sự thay đổi trong cài đặt của giao diện thành phần), chỉ quan tâm đến đầu vào, đầu ra, chức năng nhiệm vụ của hộp đen đó. Ví dụ: Ốc vít dùng để làm gì, làm thế

nào để sử dụng nó mà không cần biết nó được làm như thế nào, sử dụng công cụ gì. COP lắp ráp các thành phần thông qua giao diện của các thành phần này

Sự khác nhau giữa 2 phương pháp lập trình OOP và COP được chỉ ra trong bảng sau:

COP	**OOP**
COP phát triển phần mềm bằng cách lắp ráp các thành phần	OOP nhấn mạnh cài đặt các lớp và các đối tượng
COP nhấn mạnh giao diện và kết cấu	OOP nhấn mạnh về đối tượng/lớp OOP hỗ trợ bao bọc, thừa kế, đa xạ nhưng chưa bao giờ đạt mục đích của nó vì kế thừa xâm phạm bao bọc, hơn nữa các đối tượng và các lớp không tự vận hành.
COP là kỹ thuật đóng gói và phân tán	OOP là kỹ thuật cài đặt
COP hỗ trợ sử dụng lại mức cao	OOP hỗ trợ sử dụng lại mức thấp
COP về nguyên tắc có thể viết bởi bất kỳ ngôn ngữ lập trình nào	OOP bị giới hạn bởi ngôn ngữ OO
COP gồm các thành phần gắn kết lỏng lẻo	OOP gồm các đối tượng phụ thuộc chặt chẽ hơn vào các đối tượng khác qua giao diện kế thừa (Cha thay đổi, con thay đổi theo)
COP có các thành phần hạt nhân từ lớn đến nhỏ	OOP gồm các đối tượng là các đơn vị của kiến trúc
COP hỗ trợ đa giao diện và thiết kế hướng giao diện	OOP không cung cấp mỗi quan hệ rõ ràng về giao diện giữa các lớp con và lớp cha

Các ngôn ngữ có thể được sử dụng trong lập trình hướng cấu phần:

- Visual Basic
- Delphi
- C#
- Java

Tóm lại

Kỹ nghệ phần mềm dựa trên cấu phần – CBSE (Component Based Software Engineering) gồm các hoạt động:

- COA (Component Oriented Analysis): Phân tích hướng cấu phần
- COD (Component Oriented Design): Thiết kế hướng cấu phần
- COP (Component Oriented Programming): Lập trình hướng cấu phần
- COM (Component Oriented Management): Quản lý hướng cấu phần

Từ cách nhìn của tiến trình kỹ nghệ, các thành phần có thể được phân làm 5 dạng khác nhau:

1. Thành phần đặc tả: Biểu diễn các đặc tả của một đơn vị phần mềm, mô tả một tập hành vi của các đối tượng thành phần.
2. Thành phần giao diện: Định nghĩa một tập hành vi có thể được yêu cầu bởi một đối tượng thành phần.
3. Thành phần cài đặt: Có thể vận hành độc lập (điều này không có nghĩa nó độc lập với các thành phần khác. Nó có thể có nhiều phụ thuộc với thành phần khác).
4. Thành phần đã cài đặt: Thành phần mã thực thi
5. Thành phần đối tượng: Lớp, gói, ...

CBSE hiện tại là một mô hình cho việc phát triển các hệ thống phần mềm lớn như:

- Các ứng dụng phân tán cho doanh nghiệp
- Các ứng dụng Web – N Tier
- Các dịch vụ Web (Web Services)

Có thể dùng các công nghệ của Java như:

- EJB: Enterprise Java Bean,
- COM: Component Object Model,
- DCOM: Distributed Component Object Model ,
- CORBA:

Hoặc công công nghệ .NET của Microsoft để phát triển các hệ thống phần mềm theo mô hình này.

2.4.8 Lập trình hướng dịch vụ

Phần mềm đang ngày càng trở nên phức tạp quá mức, những nguyên nhân chính khiến cho các hệ thống phần mềm có độ phức tạp tăng cao là:

- Sự xuất hiện của nhiều công nghệ mới => Tạo nên môi trường không đồng nhất;

- Nhu cầu trao đổi, chia sẻ thông tin, tương tác giữa các hệ thống ngày càng tăng.

- Lập trình dư thừa & không thể tái sử dụng, gây tốn kém rất nhiều, không những trong giai đoạn phát triển hệ thống mà trong cả các giai đoạn vận hành, bảo trì phần mềm.

Các câu hỏi đặt ra là làm thế nào để:

- Giảm chi phí đầu tư cơ sở hạ tầng,
- Khai thác hiệu quả các công nghệ có sẵn
- Phục vụ các yêu cầu của khách hàng tốt hơn. Đáp ứng tốt các thay đổi thường xuyên về mặt nghiệp vụ.
- Khả năng tích hợp cao với các hệ thống bên ngoài
- .....

⇨ SOA (Service Oriented Architecture) là một giải pháp cho việc trả lời các câu hỏi này. Giúp cho các ứng dụng có thể giao tiếp với nhau mà không cần biết các chi tiết kỹ thuật bên trong của từng ứng dụng. SOA đưa ra các chuẩn giao tiếp dùng để gọi dịch vụ độc lập với nền tảng hệ thống.

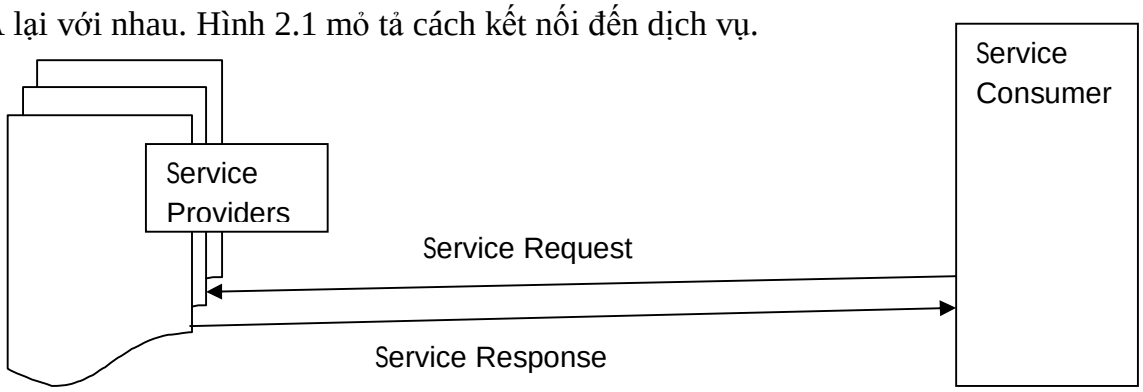
SOA phát triển từ đầu những năm 2000, được khởi xướng bởi IBM. Hiện nay đang được đầu tư và ứng dụng rộng rãi bởi nhiều tập đoàn lớn như IBM, Oracle, SAP. Microsoft, ...

SOA = {Các module};

Trong đó, các module không đơn thuần là các gói phần mềm, hay các bộ thư viện nào đó mà nó là các dịch vụ. Mỗi dịch vụ hỗ trợ thực hiện một quy trình nghiệp vụ nào đó, chúng nằm rải rác ở nhiều nơi khác nhau và có thể truy cập thông qua môi trường mạng. Chúng được tích hợp với nhau để cùng cộng tác thực hiện một công việc nào đó theo yêu cầu của khách hàng.

Ví dụ: Các dịch vụ dự báo thời tiết, Người dùng có thể sử dụng MT PC, Laptop, Mobile.. để truy cập sử dụng các dịch vụ này qua môi trường kết nối và trả cước cho việc truy cập sử dụng dịch vụ.

Để kết nối tới các dịch vụ, có nhiều cách, cách thông dụng nhất là dùng các giao thức mạng có sẵn (các dịch vụ Web) – Giúp cho việc kết nối các thành phần của hệ thống SOA lại với nhau. Hình 2.1 mỏ tả cách kết nối đến dịch vụ.

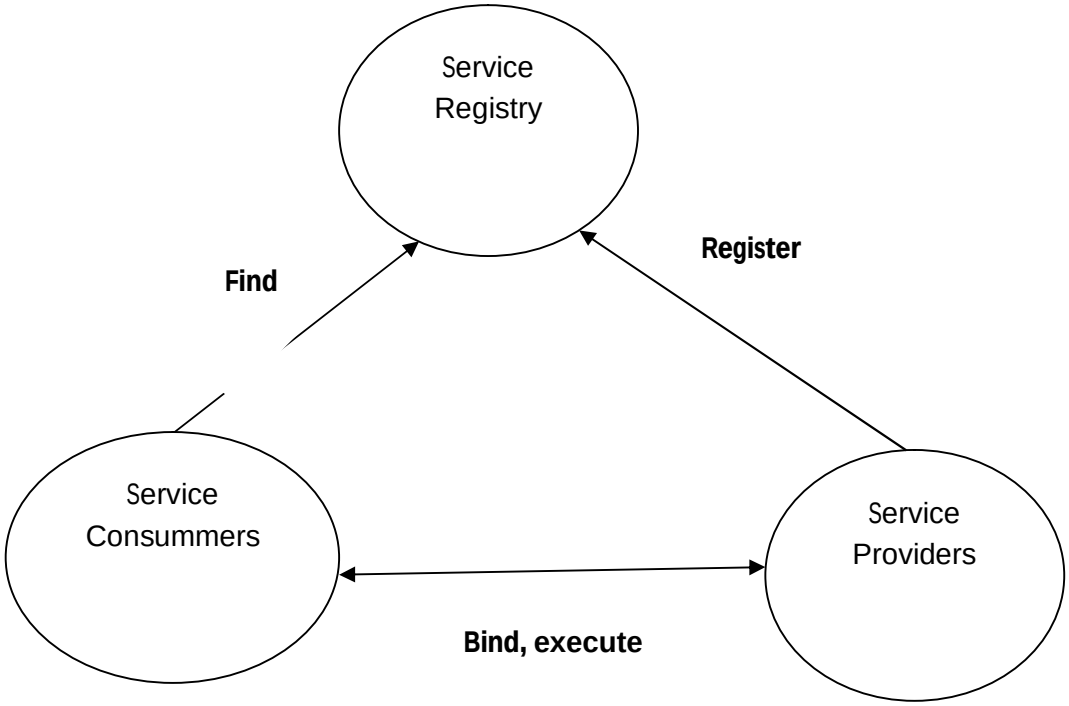


Service Providers - Các nhà cung cấp dịch vụ: Cung cấp dịch vụ phục vụ cho một nhu cầu nào đó của khách hàng. Các dịch vụ này được cung cấp qua môi trường mạng, được đăng ký ở một nơi nhất định trên mạng, nên chúng dễ dàng được tìm thấy và sử dụng lại.

Service Consumer - Người sử dụng dịch vụ: Gửi yêu cầu cung cấp dịch vụ đến nhà cung cấp và sử dụng dịch vụ mà không cần biết vị trí của dịch vụ cũng như dịch vụ được xây dựng như thế nào. Các dịch vụ này có thể có không gian địa chỉ bên ngoài ứng dụng, hoặc ở một máy khác so với máy chứa ứng dụng. SOA hướng đến kiến trúc ứng dụng phân tán.

SOA hỗ trợ khái niệm khai thác dịch vụ (Service Discovery) như sau: Người sử dụng cần dịch vụ nào, có thể tìm kiếm dịch vụ dựa trên một số tiêu chuẩn. Kết quả tìm kiếm là danh sách các dịch vụ thỏa mãn điều kiện, người sử dụng có thể chọn dịch vụ có phí giao dịch thấp nhất, kết nối đến nhà cung cấp dịch vụ để sử dụng dịch vụ.

Mối ràng buộc duy nhất giữa bên cung cấp dịch vụ và bên sử dụng dịch vụ là bản hợp đồng được cung cấp bộ đăng ký dịch vụ trung gian. Ràng buộc này là ràng buộc trong thời gian chạy. Tất cả các thông tin cần thiết về dịch vụ được lấy về và sử dụng trong khi chạy. Hình 2.2 Tổng quan về kiến trúc hướng dịch vụ - SOA

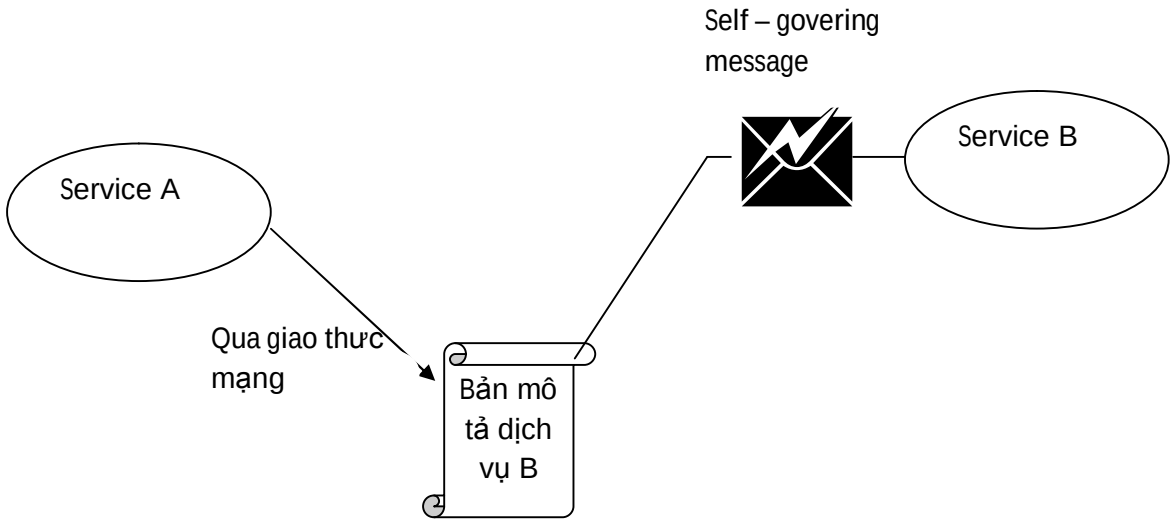


Service Registry: Là nơi lưu trữ các thông tin về các dịch vụ của các nhà cung cấp dịch vụ khác nhau. Người sử dụng dịch vụ dựa trên các thông tin này để tìm kiếm và lựa chọn nhà cung cấp dịch vụ phù hợp.

Các thành phần trong kiến trúc SOA giao tiếp với nhau sử dụng các thông điệp. Các thông điệp này dựa trên các giao thức đã được chuẩn hóa như HTTP. FTP, ...

=> Hệ thống SOA độc lập với nền, Các dịch vụ hoạt động trên các platform khác nhau vẫn có thể giao tiếp với nhau nhờ các giao diện đã được chuẩn hóa để cùng cộng tác thực hiện một tác vụ nào đó.

Hình 2.3 mô tả cách thức một một dịch vụ (dịch vụ A) truyền thông điệp để gọi dịch vụ khác (dịch vụ B)



2.4.9 Điện toán đám mây

Để quản lý tốt, hiệu quả dữ liệu của Công ty mình, của khách hàng, của đối tác. Các doanh nghiệp phải đầu tư, tính toán rất nhiều loại chi phí như phần cứng, phần mềm, mạng, quản trị viên, bảo trì, sửa chữa, nâng cấp, mở rộng, bảo mật dữ liệu .....Tử đây nẩy sinh câu hỏi liệu có một nơi tin cậy giúp các danh nghiệp quản lý tốt nguồn dữ liệu đó mà không phải bận tâm đến cơ sở hạ tầng, công nghệ sử dụng mà chỉ tập trung chính vào công việc kinh doanh của họ, như vậy, hiệu quả và lợi nhuận họ mang lại sẽ cao hơn? Thuật ngữ "điện toán đám mây – cloud computing" bắt nguồn từ đó.

Điện toán đám mây, về ý tưởng là tất cả mọi thứ như dữ liệu, phần mềm, công nghệ, tính toán, nền tảng, cơ sở hạ tầng, ... mọi thứ đều trên Internet. Từ đó, xuất hiện các *máy chủ ảo* tập trung ở trên mạng. Các máy chủ này cung cấp các dịch vụ giúp các

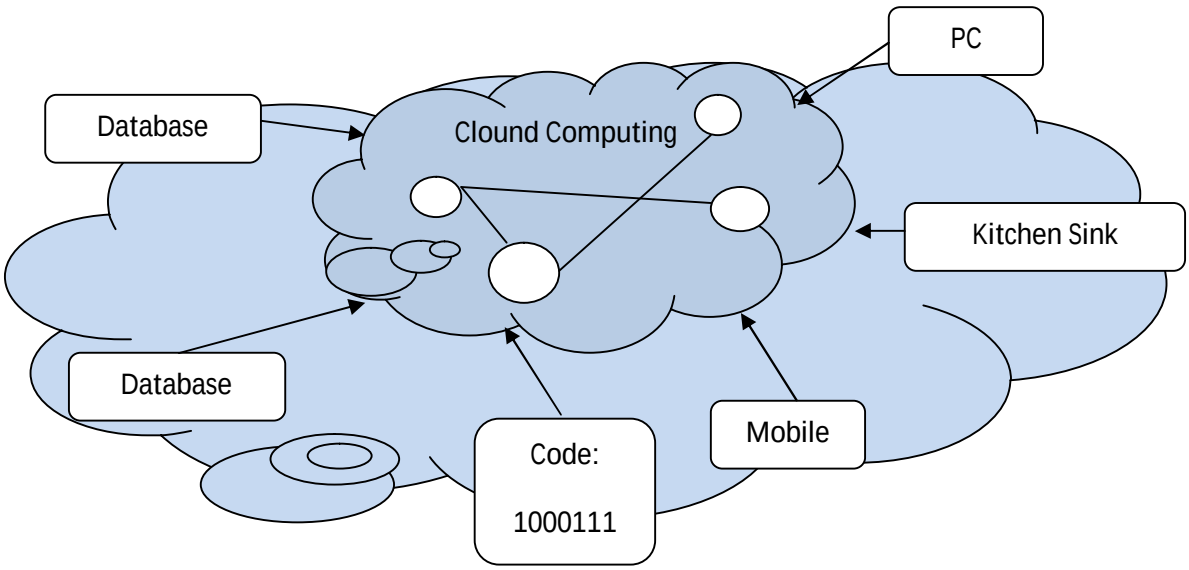
doanh nghiệp quản lý dữ liệu dễ dàng, họ sẽ chỉ trả chi phí cho dung lượng sử dụng của dịch vụ mà không cần phải đầu tư nhiều vào cơ sở hạ tầng cũng như không phải quan tâm nhiều đến công nghệ. Vậy "Clound Computing" là gì. Theo Wikimedia: "*Điện toán đám mây còn được gọi là điện toán máy chủ ảo, là mô hình điện toán sử dụng các công nghệ máy tính và phát triển dựa vào mạng intenet*".

Thuật ngữ đám mây là nối ẩn dụ chỉ mạng Internet (dựa vào cách bố trí của nó trong sơ đồ mạng máy tính) như một liên tưởng về độ phức tạp của các cơ sở hạ tầng chứa trong nó. Ở mô hình điện toán này, mọi khả năng liên quan đến công nghệ thông tin đều được cung cấp dưới dạng các dịch vụ. Nó cho phép người sử dụng truy cập các công nghệ từ một nhà cung cấp nào đó trong đám mây và không cần phải có các kiến thức, kinh nghiệm về công nghệ đó, cũng như không quan tâm đến cơ sở hạ tầng chứa nó.

Theo Gartner (http://www.buildingtheclound.co.uk):

"*Điện toán đám mây là một mô hình điện toán mà khả năng mở rộng và linh hoạt về công nghệ thông tin được cung cấp như một dịch vụ cho nhiều khách hàng đang sử dụng các công nghệ trên internet*".

Hình 2.4 mô tả tổng quan về điện toán đám mây



Các nguồn điện toán khổng lồ như phần mềm, dịch vụ, công nghệ, ... sẽ nằm rải rác tại các máy chủ (đám mây) trên Internet thay vì trong các máy tính gia đình và trong các máy tính văn phòng đặt dưới mặt đất, Mọi người cùng kết nối và sử dụng các dịch vụ trong đám mây mỗi khi họ cần.

=> Vấn đề đặt ra là cần tích hợp các đám mây thành "Sky Computing" và đưa ra các chuẩn chung để giải quyết các bài toán lớn của khách hàng.

Tóm lại

Điện toán đám mây nhằm giả quyết các vấn đề sau:

- *Vấn đề lưu trữ dữ liệu*: Dữ liệu lưu trữ tập trung ở các kho dữ liệu khổng lồ. Các tập đoàn lớn như Microsoft, Google, ... có hàng chục kho dữ liệu trung tâm nằm giải rác khắp nơi trên thế giới. Các tập đoàn này sẽ cung cấp các dịch vụ cho phép doanh nghiệp có thể lưu trữ và quản lý dữ liệu của họ trong các kho dữ liệu trung tâm trên mạng
- *Vấn đề về sức mạnh tính toán:* Có hai giải pháp là sử dụng các siêu máy tính để xử lý các tính toán và sử dụng các hệ thống tính toán //, phân tán, tính toán lưới...
- *Vấn đề về cung cấp tài nguyên phần mềm:* Cung cấp các dịch vụ như:
    - *IaaS (Infrastructure as a Service):* Cung cấp môi trường cơ sở hạ tầng để sử lý như các máy chủ ảo, lưu trữ, cân bằng tải, tường lửa, ...Những dịch vụ này có thể được thực hiện thông qua các công nghệ khác nhau, ảo hóa là một trong những công nghệ phổ biến nhất, ngoài ra còn có các công nghệ như tính toán lưới, tính toán chuỗi, ...
    - *PaasS (Platform as a Service):* Cung cấp môi trường để phát triển và chạy các ứng dụng, chứng thực, ủy quyền, quản lý phiên và siêu dữ liệu, ...
    - *SaaS (Software as a Service):* Là mô hình đám mây tiên tiến, phức tạp nhất. Các giải pháp hiện đang được cung cấp theo mô hình SaaS gồm:
        - *Doanh nghiệp thông minh*
        - *Hội nghị Web*
        - *Email*
        - *Bộ ứng dụng văn phòng*
        - ....

2.4.10 Lập trình hàm

Trong lập trình mệnh lệnh, một chương trình thường chứa 3 lời gọi đến các chương trình con liên quan đến quy trình:

- Đưa dữ liệu vào;

- Xử lý dữ liệu

- Xuất dữ liệu ra

Tương đương với chương trình gọi đến 3 chương trình con:

```
Begin
    getData(....);  {đưa dữ liệu vào}
```

processData(....); {xử lý dữ liệu}

outputData(.....); {xuất kết quả}

End.

Trong lập trình hàm, các lời gọi chương trình được viết thành biểu thức rất đơn giản:

(outputData(processData(getData(....))))

Các ngôn ngữ lập trình hàm cũng là những ngôn ngữ bậc cao, mang tính trừu tượng hơn so với các ngôn ngữ mệnh lệnh (còn gọi là ngôn ngữ bậc 4). Khi lập trình với các ngôn ngữ hàm, người lập trình phải định nghĩa các hàm toán học dễ suy luận, dễ hiểu và không cần quan tâm chúng được cài đặt trên máy như thế nào.

Hàm muốn nói ở đây là một ánh xạ toán học thuần túy: Từ tập nền → miền giá trị. Một hàm có thể không có hoặc có các tham số, sau khi tính toán, hàm trả về một giá trị nào đó thuộc miền giá trị.

Các ngôn ngữ hỗ trợ lập trình hàm như Lips, Scheme....Hạn chế của lập trình hàm là chương trình có thể rắc rối, không đạt được tính module hóa

2.4.11 Lập trình logic

Lập trình logic được ứng dụng chủ yếu và phổ biến trong lĩnh vực trí tuệ nhân tạo như:

- Công nghệ xử lý tri thức
- Máy học
- Hệ chuyên gia
- Xứ lý ngôn ngữ tự nhiên
- Trò chơi,
- ....

Nguyên lý của lập trình logic là dựa chủ yếu vào mệnh đề Horn. Mỗi mệnh đề Horn dùng để biểu diễn một sự kiện, một sự vật nào đó là đúng, không đúng, xảy ra hay không xảy ra .....trong thế giới thực.

1. Nếu một người là già và khôn ngoan thì người đó là hạnh phúc
2. Jonh là người hạnh phúc
3. Nếu X là cha/mẹ của Y, và Y là cha/mẹ của Z thì X là ông/bà của Z
4. Tom là ông của FAT
5. Tất cả mọi người đều chết (hoặc, nếu ai là người thì ai đó phải chết)
6. Socrat là người

Trong các mệnh đề Horn ở trên, các mệnh đề 1,3,5 được gọi là các luật/Rule. Còn các mệnh đề 2,4,6 gọi là các sự kiện.

Một chương trình logic có thể xem là một CSDL gồm các mệnh đề Horn hoặc dạng luật, hoặc dạng sự kiện, ...Người sử dụng gọi để chạy một chương trình logic bằng cách đặt ra các câu hỏi/truy vấn (Question/Query) truy vấn các CSDL này.

Ví dụ: Socrat có chết không? Tương đương với câu hỏi Socrat chết là đúng hay sai

Một hệ thống logic sẽ chạy chương trình theo cách suy luận – tìm kiếm trả lời dựa trên vồn hiểu biết đã có là chương trình, để minh chứng câu hỏi là khẳng định đúng hay sai..

Ngôn ngữ lập trình đại diện cho phương pháp lập trình này là Prolog. Prolog rất thích hợp để giải các bài toán liên quan đến các đối tượng và mối quan hệ giữa chúng. Một chương trình Prolog = 1 CSDL = {các mệnh đề}.

Mỗi *mệnh đề* có thể là luật, sự kiện, câu hỏi, nó được xây dựng từ các *vị từ* (predicate). Mỗi *vị từ* là một phát biểu nào đó về các đối tượng có giá trị chân lý là đúng hoặc sai. Một *vị từ* có thể có các đối số là các *nguyên logic (logic atom)/nguyên tử. Nguyên logic* biểu diễn một quan hệ giữa các *hạng (term*). Hạng được xem là những đối tượng dữ liệu trong môt chương trình logic. Một hạng có thể là sơ cấp hoặc hạng phức. Hạng sơ cấp gồm các hằng, biến, ...Hạng phức biểu diễn các đối tượng phức tạp của bài toán cần giải thuộc lĩnh vực đang xét. Hạng phức có thể là một hàm tử (function) chứa các đối số có dạng:

<Tên_hàm_tử> (đối soos1, đối số 2, ...).

Mỗi đối số có thể là biến, hạng sơ cấp, hạng phức hợp, ...

Một số kiểu hàm tử như kiểu bản ghi, kiểu danh sách, kiểu mảng,.....

Ví dụ: hàm tử "**.**" Biểu diễn cấu trúc List. Xét các danh sách sau (*)

f(5, a,b)**.**

Student (Lan, 1998, infor, 2, dc)**.**

[a,b,c]**.**

=> ta có:

f(5,a,b): là hạng phức

*f(5,a,b)***.**: là nguyên tử

*f(5.a,b)***.** *có 3 phần tử là danh sách ít*: là một vị từ

*Nếu đơn vị x có các đơn vị là các danh sách là * thì a là một tổ chức nhỏ*: là một mệnh đề

Hạng + mối quan hệ giữa các hạng tạo thành một mệnh đề

2.4.12 Lập trình CSDL

Trong lập trình CSDL, chương trình bao gồm một tập các khai báo truy xuất đến CSDL để trích rút ra các thông tin

Ví dụ: Xét chương trinh

Select .....

From .....

Where .....

Critia .....

Các ngôn ngữ hỗ trợ cho phương pháp lập trình này như SQL, SQL Server, ...

.

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 2

CHƯƠNG 3 PHONG CÁCH LẬP TRÌNH, GỠ RỐI VÀ TỐI ƯU CHƯƠNG TRÌNH

3.1 Phong cách lập trình

Phong cách lập trình bao hàm một triết lý về lập trình, nhấn mạnh tới tính đơn giản, rõ ràng. Viết một chương trình máy tính là viết một dãy các lệnh bằng một ngôn ngữ lập trình cụ thể. Cách thức của mỗi lệnh diễn tả trong một chừng mực nào đó sẽ xác định tính dễ hiểu của toàn bộ chương trình.

Mục đích của phong cách lập trình là làm cho chương trình dễ đọc hơn, một phong cách tốt quyết định đến việc lập trình tốt. Những nguyên tắc của phong cách lập trình dựa trên những cảm nhận chung bằng kinh nghiệm mà không dựa vào những quy luật hay những chỉ dẫn nào khác.

Chương trình phải được viết rõ ràng, đơn giản, logic, diễn đạt một trình tự tự nhiên, dễ hiểu, tên hàm, tên biến phải có nghĩa, định dạng rõ ràng và phải có phần chú thích....Tính nhất quán là cần thiết giúp người khác có thể hiểu được chương trình nếu mọi người đều có phong cách giống nhau. Do đó, người ta đưa ra các nguyên tắc/quy ước trong lập trình để tạo nên phong cách tốt, đảm bảo chương trình có mã nguồn dễ hiểu, dễ kiểm thử, dễ bảo trì và ít lỗi.

Nếu ta bắt đầu bảo trì chương trình với bộ mã nguồn tồi, ta sẽ mất chi phí lớn hơn gấp 10 lần để phát triển và cố định lỗi so với chương trình có mã tốt.

3.2 Các nguyên tắc lập trình

Trước khi tìm hiểu các nguyên tắc lập trình, chúng ta cần phân biệt giữa giải thuật và chương trình. Giải thuật bản chất là cách làm/phương pháp thực hiện cái gì đó. Chương trình tuân theo ngữ nghĩa của các luật ngôn ngữ. Sau đây chúng ta sẽ tìm hiểu các nguyên tắc trong lập

Nguyên tắc 1
“ Lập trình không chỉ là viết một chương trình mà máy tính hiểu, mà còn phải để mọi người hiểu” .

Là nguyên tắc quan trọng cần biết trong lập trình. Viết chương trình để Máy tính hiểu thì nó mới thực hiện đúng, để mọi người hiểu để bảo trì, sửa chữa, đánh giá. Đây là nguyên tắc kỹ nghệ phần mềm đầu tiên cần phải nghĩ đến. Viết chương trình mà người khác không thể đọc và hiểu là một kỹ nghệ tồi.

Nguyên tắc 2

" Cần tạo chú thích trong chương trình" .

Chú thích giúp người đọc hiểu chương trình:

- Chú thích giúp người đọc không phải bằng cách:
  - Nhắc lại những điều mà mã nguồn đã nêu rõ
  - Phủ nhận mã nguồn
  - Làm rối trí người đọc với cách trình bày phức tạp mà lại còn mắc lỗi.
- Chú thích tốt phải:
  - Chỉ ra một cách ngắn gọn, rõ ràng những chi tiết đáng chú ý
  - Bằng cách cung cấp một cái nhìn chi tiết hơn về tiến trình của chương trình
  - Tuân theo các nguyên lý tạo chú thích sau:

Một chương trình tốt phải có các chú thích sau:

- Có một chú thích ở đầu chương trình, tên file giải thích công việc mà chương trình làm.
  Ví dụ: Chiếu chương trình minh họa một chương trình tốt, một chương trình tồi
- Chú thích trong chương trình. Người lập trình cần lưu vết các mở rộng trong chương trình, nó sẽ giúp cho việc gỡ rối và bảo trì chương trình:
  - Chú thích cho mỗi phương thức: Giải thích những gì phương thức thực hiện, các điều kiện trước, điều kiện sau
    - Thường 1->3 dòng là tốt nhất cho mỗi phương thức
    - Ví dụ: Minh họa một chương trình tốt, một chương trình tồi (chỉ có mã, không có chú thích, không thụt cấp, không có ý niệm tiếp tục chương trình này).
  - Chú thích cho các biến toàn cục
  - Chú thích cho những thao tác phức tạp.
  - Chú thích cho các thuật toán phức tạp sử dụng trong chương trình:
    - Cho biết tài liệu tham khảo
    - Mô tả ngắn gọn thuật toán: Ý tưởng của thuật toán
    - Các dữ liệu được sử dụng, điều kiện thực thi thuật toán

Lưu ý:

- *Đừng chú thích cho những gì hiển nhiên*

- *Đừng chú thích phủ định mã nguồn: Các chú thích phải được cập nhật khi mã nguồn thay đổi*
- *Chú thích phải rõ ràng, không gây nhầm lẫn: Chú thích được coi là giúp người đọc hiểu những chỗ khó trong chương trình chứ không phải để gây thêm khó khăn cho họ.*
    - *Mã nguồn tốt cần ít chú thích hơn và ngược lại*
- *Chú thích không được lấn áp mã nguồn.*
- *Đừng chú thích cho những đoạn mã nguồn dở mà hãy viết lại nó.*

Nguyên tắc 3 " Nên nghĩ mỗi phương thức giải quyết một vấn đề" .

Một vấn đề có thể là mức cao: Đó là bài toán đơn, mức cao và phân rã xuống. Hầu hết các phương thức khoảng từ 1 -> 15 dòng mã.

Nguyên tắc 4 " Đăt các định danh nên gợi nhớ, ngắn gọn, giải thích những gì chương trình làm một cách chính xác " .

Nếu một phương thức rất ngắn, tên giải thích chính nó một cách chính xác, có các điều kiện trước, sau rõ ràng là một ý tưởng tốt. Tên không nên đặt quá dài vì nó tiềm ẩn lỗi.

Có sự khác nhau giữa lập trình cấu trúc là lập trình hướng đối tượng:

- Trong lập trình có cấu trúc: Chương trình được tổ chức dưới dạng cây phân cấp chức năng => Tên các chức năng là các động từ
- Trong lập trình hướng đối tượng: Chương trình được tổ chức dưới dạng cây phân cấp lớp đối tượng => Tên các lớp đối tượng là các danh từ, dang từ cũng được xem là phương thức (phương thức khởi tạo).
    - Ví dụ: Một đĩa CD được xem là phương thức, nó cũng có các tham biến của nó, và các câu lệnh của nó được ẩn đi. Phương thức này có thể được gọi khi tạo ra một chiếc đĩa CD.

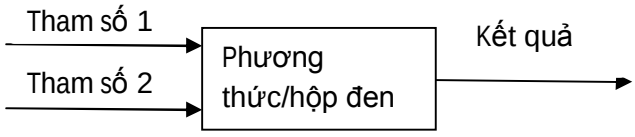
Nguyên tắc 5: Ẩn thông tin " Một phương thức nên che đi những gì diễn ra trong thân nó " .

Người dùng phương thức chỉ cần biết:

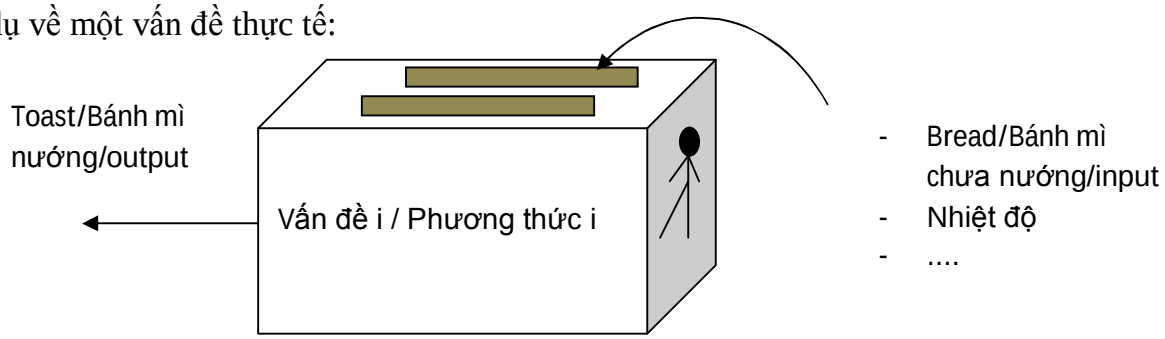
- Tên phương thức: Có thể là động từ hoặc danh từ tùy theo kiểu giá trị trả về của nó
- Phương thức dùng để làm gì: Làm thế nào thì không cần biết
- Các tham biến truyền vào nó là gì.

Ví dụ: Ta biết phương thức readInt(): Dùng để lấy thông của người dùng từ bàn phím, giá trị (số nguyên) trả về thông qua tên hàm, không có tham biến truyền vào mà không cần biết bên trong phương thức là gì.

Có thể xem một phương thức như một hộp đen:



Ví dụ về một vấn đề thực tế:



Trong lập trình hướng đối tượng, để tạo một chương trình tốt, người lập trình nên nghĩ về những thành phần nào của đối tượng nên để ở public (mọi chỗ trong hệ thông đều có thể truy cập làm thay đổi giá trị của chúng), thành phần nào nên để private (riêng của mình).

Tương tự việc chọn cách đặt tên để giúp người đọc chương trình hiểu rõ, việc viết các biểu thức cũng phải được thực hiện sao cho chúng có ý nghĩa càng tường minh càng tốt, mã nguồn càng sáng sủa càng tốt.

Nguyên tắc 6: " Các biểu thức điều kiện nên mang tính tự nhiên " .

Hãy viết các biểu thức như cách bạn đọc. Các biểu thức điều kiện phủ định luôn luôn khó hiểu.

Ví dụ:

If(!block_Id < actblks)|| !(block_Id>=unblocks))

{

.....

}.

Đảo lại các phát biểu phủ định sẽ là phát biểu dạng khẳng định:

If(block_Id >= actblks)||(block_Id<unblocks))

{

.....

}.

Nguyên tắc 7: “Dùng dấu ngoặc để tránh tối nghĩa” .

Dùng dấu ngoặc để đánh dấu một nhóm và có thể dùng để làm rõ nội dung ngay cả khi không cần dùng.

Ví dụ: Phát biểu sau đây sử dụng nguyên tắc 7

Các phép toán so sánh (<, <=, ==, !=, >=, >) có thứ tự ưu tiên lớn hơn các toán tử logic (&&, ||)

Nguyên tắc 8: “Phân tích những biểu thức phức tạp thành những biểu thức đơn giản hơn” .

C. C++. Java, ... có cú pháp và số lượng toán tử phong phú. Khi lập trình rất dễ bị cuốn vào việc thu gọn chương trình bằng cách nhồi nhét tất cả chỉ vào trong một phát biểu

Ví dụ: Biểu thức sau rất ngắn gọn song đã đưa quá nhiều phép toán vào trong một phát biểu:

*? *x+= (*xp = (2*k)<(n-m)?c[k+1]:d[k--]));*

Biểu thức này sẽ dễ hiểu hơn nếu ta chia nhỏ ra thành nhiều đoạn như sau:

*If (2*k<n-m)*

**xp=c[k+1]*

Else

**xp=d[k--]*

**x+=*xp;*

Nguyên tắc 9: Tính sáng sủa của biểu thức “Diến đạt các biểu thức sao cho rõ nghĩa nhất, tránh mã nguồn bí hiểm” .

Năng lực sáng tạo vô tận của các lập trình viên đôi khi được dùng để viết mã nguồn ngắn nhất có thể, hoặc để tìm các cách thông minh hơn để đạt được mục đích. Tuy nhiên đôi khi những kỹ năng này được áp dụng sai chỗ; vì mục tiêu là viết mã nguồn sao cho sáng sủa, dễ hiểu chứ không phải viết mã để thể hiện tài khéo léo.

Ví dụ: xét xem phép tính sau thực hiện gì?

Subkey =Subkey>>(bitoff – ((bitoff>>3)<=3));

Dịch bitoff sang phải 3 bít

Kết quả dịch sang trái 3 bít, thay 3 bít được dịch bằng số 0

Kết quả thu được 3 bít cuối của bitoff (111=7)

Ta thấy, rất mất thời gian giải đáp để hiểu biểu thức trên, do đó, nên thay biểu thức này bởi biểu thức tương đương:

Subkey = subkey >> (bitoff&0x7), hoặc

Subkey >>=bitoff&0x7;

Biểu thức sẽ ngắn gọn và dễ hiểu hơn rất nhiều.

Nguyên tắc 10: “Đặt tên cho các số tối nghĩa”

Số tối nghĩa là các hằng số như kích thước mảng, vị trí ký tự, hệ số chuyển đổi và các giá trị số khác xuất hiện trong chương trình. Về nguyên tắc chung, tất cả các giá trị số trừ 0 và 1 đều xem là tối nghĩa và nên đặt tên cho nó. Một con số trong mã nguồn sẽ không thể cho biết gì về tầm quan trọng cũng như xuất xứ của nó. Điều đó được xem là tối nghĩa và nó làm cho chương trình trở nên khó hiểu và khó sửa đổi

Ví dụ: Chiếu chương trình để minh họa

- Cách đặt tên cho các số tối nghĩa để làm chương trình dễ hiểu

- Sự tiện lợi trong việc sửa chương trình khi đặt tên cho các số tối nghĩa

Nguyên tắc 11:
"Dùng các hằng số dạng ký tự, không dùng các dạng nguyên khi làm việc với các biến ký tự"

Ví dụ:

Xét một phép toán kiểm tra điều kiện sau:

```
If (c>=65 && c<=90)
    ........
```

Phép toán này hoàn toàn phụ thuộc vào một bẳng mã ký tự đặc thù nào đó. Để tránh sự phụ thuộc này, tốt nhất ta nên dùng phép toán sau thay thế:

```
If (c>='A'&& c<='Z')
    ......
```

Nguyên tắc 12: Viết chương trình theo cấu trúc
"Cấu trúc con lọt trong cấu trúc cha. *Điểm vào và điểm ra của một cấu trúc phải trên cùng một cột*"

Tóm lại

Mã nguồn chương trình được viết tốt sẽ dễ đọc, dễ hiểu và hầu như chắc chắn là ít lỗi hơn, và có khả năng kích thước sẽ nhỏ hơn so với mã nguồn được chắp vá, cẩu thả và không hề chau chuốt.

Trong tình huống cấp bách cần hoàn thành chương trình cho kịp với một thời hạn nào đó, thường vấn đề phong cách bị gác sang một bên để giải quyết sau. Quyết định này có thể sẽ phải trả giá đắt. Mã nguồn tùy tiện cũng là mã nguồn dở. Nó không chỉ nguy hiểm và khó đọc mà còn có thể gây ra những hậu quả nghiêm trọng khác.

Quan điểm cơ bản là: Phong cách tốt phải là vấn đề thói quen. Nếu ngay từ đầu khi viết mã nguồn bạn đã nghĩ đến vấn đề phong cách và nếu bạn dành thời gian để duyệt lại và cải tiến chương trình, bạn sẽ tập cho mình một thói quen tốt. Một khi đã trở thành thói quen, tiềm thức của bạn sẽ giúp bạn trong nhiều chi tiết vụn vặt, và ngay cả khi bạn phải viết mã nguồn trong trạng thái chịu áp lực công việc cao, mã nguồn của bạn cũng sẽ tốt hơn.

3.3 Các chuẩn trong lập trình

Các chuẩn có thể được kiểm thử bằng máy tính. Một số chuẩn chung cho tất cả các ngôn ngữ lập trình:

“Mọi modul” chỉ gồm 35 ->50 phát biểu. Nếu ít hơn, hoặc lớn hơn cần hỏi ý kiến người quản lý.

“Các lệnh if không được lồng nhau quá 3 mức”.

“Nên tránh sử dụng các phát biểu goto, goto có thể dược sử dụng để xử lý lỗi – nhưng nên tối thiểu dùng”.

Mục đích của các chuẩn mã hóa là tăng tính dễ dùng cho hoạt động bảo trì, làm tăng chất lượng phần mềm, đây là một trong những mục đích chính của kỹ nghệ phần mềm.

3.4 Các lỗi có thể phát sinh trong quá trình thiết kế và cài đặt một sản phẩm phầm mềm

Ta hãy đơn giản hóa quá trình thiết kế và cài đặt một sản phẩm phần mềm theo một tiến trình gồm 5 giai đoạn đầu tiên (còn các giai đoạn khác như kiểm sửa, vận hành, bảo trì, ... tạm thời bỏ qua):

1. Phân tích các yêu cầu: thu thập, tìm hiểu các nhu cầu, hình thành bài toán
2. Thiết kế các chức năng: Xác định các chức năng cần có của sản phẩm+chia nhỏ các chức năng
3. Đặc tả các thao tác cơ sở
4. Cài đặt các chức năng: Mã hóa bản đặc tả mức thấp
5. Khớp nối các đơn thể.

Ta lần lượt xét các loại lỗi có thể nẩy sinh trong tiến trình thiêt kế và cài đặt sản phẩm phầm mềm trong các mục sau đây.

3.4.1 Ý đồ thiết kế sai

Đây là loại lỗi chiến lược, là loại lỗi nặng, nó là ta mất đi một khoản tiền lớn với các lao động trí tuệ lớn. Nó dẫn cả nhóm đến tình trạng thất bại. Kết quả mà cả một tập thể làm ròng rã trong nhiều năm tháng lại chẳng dùng vào việc gì. Tuy nhiên cần lưu ý đến lời bàn sau: Nếu ý đồ thiết kế sai, nhưng bản thân thiết kế đúng và chuẩn mực thì nhóm thiết kế là nhóm giỏi. Họ còn nhiều dịp để thi thố tài năng. Trong phát triển phần mềm, lỗi này liên quan đến việc lựa chọn sai quy trình phát triển phần mềm.

(~ quy trình thiết kế sai mục đích, hệ thông sai mục đích: dự án 122, ...)

3.4.2 Phân tích các yêu cầu không đầy đủ và lệnh lạc (xảy ra ở giai đoạn 1)

Cần lưu ý rằng, người đặt hàng vì không biết gì về tin học nên họ không thể phát biểu chính xác và đầy đủ các yêu cầu của họ. Chúng ta cũng không đủ hiểu biết về địa bàn và đối tượng mà ta định áp dụng tin học. Do đó, ta không thể thu thập đầy đủ và chính xác các thông tin của đối tượng.

Đây là mâu thuẫn cơ bản của quá trình thiết kế, thường xảy ra ở giai đoạn 1.

Ví dụ: Xét bài toán phân số, lỗi thuộc loại này là:

Người lập trình không thể diễn đạt chính xác khái niệm phân số, ví dụ về chia phân số. Bản thân ta cũng chưa bao giờ đụng chạm tới các phép toán về phần số.

3.4.3 Hiểu sai về chức năng

Lỗi này thường xảy ra ở giai đoạn 2. Hiểu sai về chức năng dẫn đến đặc tả chức năng sai.

Ví dụ: Trong bài toán phân số:

- Ta không định nghĩa chính xác được thế nào là phân số. Ví dụ coi -2/-3, 2/-3 cũng là phân số. Thực ra dấu – không đi với mẫu số vì sẽ gây nhiều rắc rối khi thực hiện các phép toán.
- Khi làm việc với phân số, bỏ qua toán tử tạo lập một phân số => vi phạm nguyên lý đối tượng.

Nguyên lý đối tượng
"Khi thao tác trên đối tượng nào, thì trước hết phải phát sinh ra nó"

- Không rút gọn phân số kết quả
- Quan niệm chia phân số giống nhân nghịch đảo mà quên không xét tử số = 0.

Điều đặc biệt là khi học ở phổ thông, ta có thể rất hiểu và thuộc các quy tắc thực hiện các phép toán trên phân số. Nhưng khi đặt bút mô tả chúng, thì ta lại quên những điểm rất cơ bản. Đó là nội dung của nguyên lý gián đoạn.

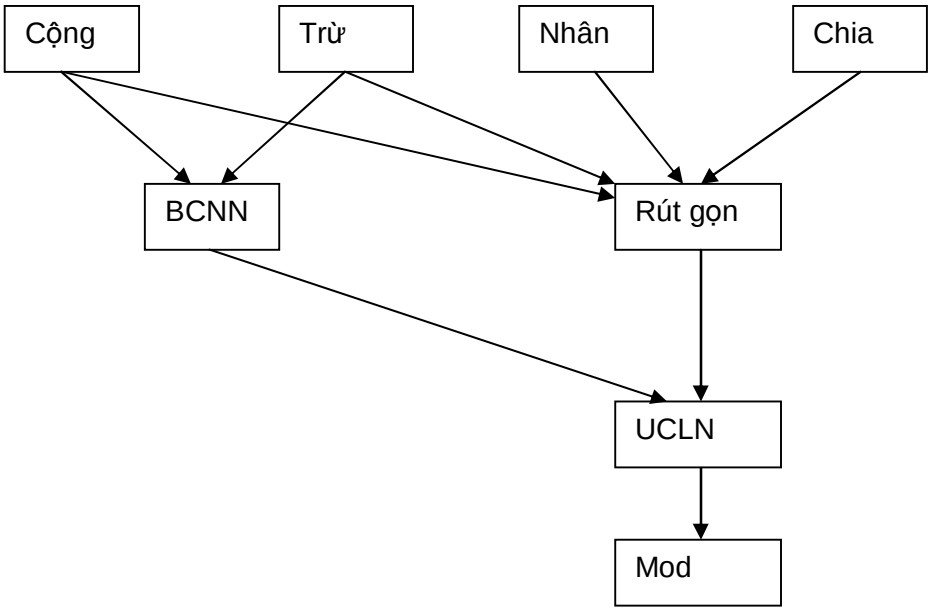
Nguyên lý gián đoạn
"Người lập trình không có ý thức thường trực về việc dùng những tri thức cơ bản của khoa học mà họ có sẵn"

3.4.4 Lỗi tại các đối tượng chịu tải

Thường xảy ra ở giai đoạn 3. Đây là các lỗi nằm ở các thủ tục/thao tác mức thấp, có mặt trong bộ phận hợp thành của hầu hết các thủ tục khác. Đây là những lỗi nặng.

Ví dụ 1: Tưởng tượng rằng, một seri các bóng bán dẫn sai quy cách do dây chuyền sản xuất chính có trục trặc. Các bóng bán dẫn này lại được dùng để lắp ráp thành các thiết bị điện tử đủ loại như đài thu thanh, máy ghi âm, Ti vi, Đồng hồ, ...=> Hậu quả của loại lỗi này rõ ràng là rất lớn.

Ví dụ 2: Xét bài toán phân số



Cả 4 phép toán Cộng, trừ, nhân, chia đều gọi thủ tục rút gọn. Rút gọn là thủ tục chịu tải, nếu sai do đặc tả/cài đặt thì mọi phép toán trên phân số đều sai.

Tiếp tục triển khai các phép toán BCNN, Rút gọn, 2 thủ tục này gọi thủ tục UCLN, UCLN là thủ tục chụi tải mức thấp hơn, các lỗi của nó còn nặng hơn lỗi tại thủ tục rút gọn.

UCLN lại gọi thủ tục lấy phần dư Mod. Mod là thủ tục chịu tải mức thấp hơn nữa. Thủ tục BCNN chỉ dùng cho hai phép toán Cộng, trừ thông qua thủ tục quy đồng mẫu số (qđms) => Mức chịu tải của BCNN trong thiết kế này là nhỏ hơn mức chịu tải của thủ tục UCLN, Rút gọi và Mod.

Càng đi xuống, mức chụi tải càng lớn, lỗi phát sinh ở mức dưới được xem là nặng hơn ở mức trên. Chúng ta cần nắm vững nguyên lý mức độ lỗi.

Nguyên lý mức độ lỗi

"Lỗi nặng nhất nằm ở mức cao nhất (ý đồ thiết kế) và ở mức thấp nhất (thủ tục có mức chịu tải lớn nhất)"

Nguyên lý mức độ lỗi chỉ đạo chúng ta tập trung chú ý khi xây dựng mục đích thiết kế & khi triển khai đặc tả và cài đặt các thao tác ở mức thấp.

3.4.5 Lỗi lây lan

Lỗi lây lan là những lỗi được truyền từ chương trình này sang chương trình khác. Lỗi này được định nghĩa như sau:

Giả sử ta có 2 chương trình A và B. A chứa lỗi E, B không chứa lỗi E. Nếu đến một thời điểm nào đó B cũng nhiễm E một cách trực tiếp hoặc gián tiếp từ A thì ta nói E là lỗi nây lan. E là đoạn chương trình do người ta cố lập ra và cài sẵn vào chương trình (vào A chẳng hạn), rồi tìm cách truyền cho những chương trình lành khác. Từ cách nhìn này ta thấy, mỗi con visus (chương trình v) thường có hai thành phần:

v=(T, E).

Trong đó:

- T: là cơ chế (thủ tục) truyền bệnh, thường là truyền chính v sang những tệp chương trình lành khác
- E: Lỗi gặp phải

Nghiên cứu Visus máy tính là vấn đề khá hay. Ta không đi sâu vào cơ chế visus mà dành vấn đề này cho những nhà visus học.

3.4.6 Lỗi cú pháp

Những lỗi đã phân tích ở trên (trừ lỗi do visus gây ra) đều là những lỗi thực hiện (Run Errors). Các chương trình đều được viết đúng về mặt cú pháp, tức là đúng với những quy định về văn phạm và cấu trúc của ngôn ngữ. Ngoài những loại lỗi trên, còn có một loại lỗi nữa đó là lỗi cú pháp. Nói chung các lỗi cú pháp do chương trình dịch phát hiện và thông báo theo nguyên lý an toàn.

Nguyên lý an toàn

"Mọi lỗi, dù nhỏ, phải được phát hiện ở một bước nào đó của chương trình. Việc phát hiện lỗi phải được thực hiện trước khi lỗi đó hoành hành"

Để tìm và sửa được các lỗi cú pháp trong chương trình của mình, lập trình viên phải đọc và hiểu được các thông báo lỗi (thường được viết bằng ngôn ngữ Tiếng Anh).

Cũng cần lưu ý: Do độ phức tạp của chương trình nên không phải lỗi nào cũng được chỉ ra một cách tường minh ở đúng chỗ nó xuất hiện. Loại lỗi này thường xảy ra ở giai đoạn 4.

3.4.7 Hiệu ứng phụ (Side Effect)

Đây là hiện tượng xảy ra khi một đơn vị chương trình/module làm thay đổi giá trị của một biến ngoài ý muốn của lập trình viên.

Ví dụ: Viết một chương trình Pascal tính giá trị của biểu thức:

f = 2*(x+y*2-1);

Với x, y là một số nguyên nhập vào từ bàn phím.

Chương trình sẽ gồm các bước sau:

- Nạp x;
- Nạp y;
- Tính f;
- Thông báo kết quả;

⇨ Chương trình

```
Program Bieuthuc;
Uses crt;
Var x,y,z: integer;
Fuction f():integer;
Begin
        y:= 2*y+1;
        f:=2*(x+y);
End;
BEGIN
    Write('x='); readln(x); {Nạp x}
    Write('y='); readln(y); {Nạp y}
    z:=f();
    writeln(2, '*(',x,'+2*',y,'+1)=',z);
```

END.

Cho chạy chương trình và nạp x=3; y=4. Kết quả xuất hiện trên màn hình 2*(3+2***9**+1)=24. Nhưng điều ta mong đợi là hiển thị lên thông báo:2*(3+2***4**+1)=24.

Đáng lẽ phải in giá trị 4 của y như đã nạp, chườn trình lại in giá trị 9. Xem lại f ta thấy lệnh y:=2*y+1 đã làm thay đổi giá trị của biến y từ 4 thành 9. Vì y là biến toàn cục cho nên mọi điểm của chương trình nói chung và trong hàm f nói riêng đều có thể truy cập và làm thay đổi giá trị của y.

Để tránh hiệu ứng phụ, ta nên áp dụng nguyên lý địa phương sau đây:

Nguyên lý địa phương

- Các biến địa phương (dù trùng tên với biến toàn cục) không làm thay đổi giá trị của biến toàn cục;
- Mọi tham biến hình thức loại truyền theo trị trong chương trình con đều là các biến địa phương;
- Các biến khai báo trong chương trình con đều là các biến địa phương.
- Khi sử dụng biến phụ thì nên dùng biến địa phương.

Sử

```
Program Bieuthuc;
Uses crt;
Var x,y,z: integer;
Fuction f():integer;
Var y: integer; {hoặc sửa hàm f thành: Fuction f(x,y:integer):integer;}
Begin
    y:= 2*y+1;
    f:=2*(x+y);
End;
BEGIN
   Write('x='); readln(x); {Nạp x}
   Write('y='); readln(y); {Nạp y}
   z:=f();
   writeln(2, '*(',x,'+2*',y,'+1)=',z);
END.
```

3.5 Một số vấn đề trong cải tiến hiệu xuất chương trình

Ta sẽ quan tâm đến vấn đề giảm thời gian chạy và chi phí bộ nhớ cho chương trình. Để minh họa cho các luận cứ được nêu, ta chỉ ra các vấn đề cần quan tâm thông qua các ví dụ minh họa.

3.5.1 Tốc độ xử lý

Trong hầu hết các trường hợp, tốc độ của chương trình là quan trọng, đặc biệt với các ứng dụng thời gian thực, ứng dụng về xử lý trên CSDL lớn, ...Để một ứng dụng có tốc độ xử lý nhanh, người lập trình chúng ta phải quan tâm tới các yếu tố như:

- Thuật toán sử dụng,
- Lựa chọn CTDL,
- Tinh chế mã,
- ....

a.Thuật toán sử dụng

<u>* Xác định bài toán</u>

Trước khi bắt tay vào giải bài toán, cần tìm hiểu kỹ các yêu cầu mà bài toán đặt ra và tận dụng mọi điều đã biết từ bài toán.

Ví dụ: Cho một mảng gồm 1.000.000 phần tử, các giá trị nằm trong khoảng từ 1→10 một cách ngẫu nhiên. Hãy sắp xếp để được mảng có thứ tự giảm dần.

Ta thấy, bài toán trên, theo cách giải tổng quát là bài toán sắp xếp. Do đó, ta có thể sử dụng các thuật toán sắp xếp như sắp xếp chèn, lựa chọn, sắp xếp nhanh, ... để giải, và độ phức tạp trong cách giải này là $O(n^2)$ hoặc O(nlogn) nếu là thuật toán sắp xếp nhanh. Tuy nhiên, ta đã bỏ qua một tính chất của bài toán đó là các giá trị nằm trong khoảng từ 0 → 10 một cách ngẫu nhiên. Sau khi nghiên cứu bài toán, tận dụng thêm điều đã biết này từ bài toán, ta quyết định sử dụng thuật toán đếm để sắp xếp bài toán như sau;

- Khởi tạo mảng gồm 10 biến nguyên với các giá trị 0 (i = 1..10)

- Thực hiện rải trị cho mảng 10 biến nguyên ứng với số lần là giá trị của biến i như sau: Với mỗi giá trị i trong mảng ban đầu đã cho (gồm 1.000.000 phần tử), tăng giá trị của ô nhớ i lên một đơn vị

=> Như vậy, chi phí về độ phức tạp của bài toán là O(n).

<u>* Sức mạnh của thuật toán:</u> Lựa chọn thuật toán phù hợp

Việc nghiên cứu thuật toán rất hữu ích, chúng có ảnh hưởng quan trọng đến các hệ thống phần mềm và đặc biệt là chúng giúp tăng tốc độ vận hành của hệ thống

Ví dụ: xét bài toán Quay mảng một chiều chứa N phần tử về bên trái I vị trí

Các thuật toán có thể sử dụng là:

Thuật toán 1:

- Sao I phần tử đầu tiên của mảng sang mảng tạm
- Dịch chuyển N-I phần tử cuối của mảng về bên phải I vị trí
- Sao I phần tử của mảng tạm vào cuối mảng.
  - ⇨ Với I và N lớn, mảng tạm khá tốn bộ nhớ. Xét trường hợp bộ nhớ không rồi rào thì giải quyết thế nào?

Thuật toán 2:

- Viết thủ tục con quay mảng X sang trái 1 vị trí => giải quyết được vấn đề bộ nhớ vì cần dùng một biến phụ
- Thực hiện thủ tục trên I lần
  - ⇨ Tốn thời gian vì nó tỷ lệ với N. Thời gian thực hiện là O(I*N)

Thuật toán 3:

Để ý rằng, khi quay phần tử X gồm N phần tử sang trái I vị trí, thì

Phần tử X[I+1] sẽ là phần tử X’[1]

Phần tử X[2I+1] là phần tử X’[I+1]

...

Và cứ tiếp tục như vậy

=> Thuật toán 3 sẽ như sau (giả sử mảng X đánh chỉ số từ 0):

- Dịch chuyển X[1] đến một biến tạm T

- Dịch chuyển X[i+1] đến X[1];

- Dịch chuyển X[2i+1] đến X[i+1];

- Dịch chuyển X[3i+1] đến X[2i+1];

- Dịch chuyển X[4i+1] đến X[3i]+1;

- ....

Quá trình cứ tiếp tục, chỉ số của X được tính theo mod của N, tức là khi vượt quá N sẽ lấy phần dư của phép chia. Quá trình lặp lại cho đến khi gặp phần tử dịch chuyển là X[i] và lúc này dùng giá trị từ biến T để dịch chuyển đến ô nhớ tương ứng và chấm dứt quá trình.

Ví dụ:

Xét N=8; I=3; X= ABCDEFGH

Quá trình thực hiện thuật toán diễn ra như sau:

X= ABCDEFGH; N=8; I=3; T=A

- Dịch chuyển X[i+1]: X[4] đến X[1];

- Dịch chuyển X[2i+1]: X[7] đến X[4];

- Dịch chuyển X[3i+1]: X[10] =X[2]đến X[7];

- Dịch chuyển X[4i+1]: X[13]=X[] đến X[10]=X[2];

-....

Quá trình trên được tóm tắt lại như sau:

T=A; N=8; I=3; X=ABCDEFGH

4 → 1:	DBCDEFGH
7 → 4:	DBCGEFGH
(0) 2 → 7:	DBCGEFBH
5 → 2:	DECGEFBH
8→ 5:	DECGHFBH
(11)3→ 8:	DECGHFBC
6→ 3(11):	DEFGHFBC
9(1)→ 6:	DEFGHABC
T	=> DỪNG

Đây là thuật toán có chi phí vùng nhớ không lớn và thời gian chạy chấp nhận được: O(n) như Thuật toán 1 nhưng không tốn không gian.

b.Lựa chọn CTDL

Song song với thuật toán, việc lựa chọn CTDL ảnh hưởng lớn đến hiệu xuất chương trình và nó tác động đến bản thân thuật toán vì CTDL và thuật toán gắn bó mật thiết với nhau. Việc lựa chọn đúng đắn CTDL có tác dụng:

- Làm giảm không gian bộ nhớ;

- Giảm thời gian chạy;

- Tăng tính khả chuyển và dễ bảo trì chương trình.

Ví dụ: lựa chọn cấu trúc dữ liệu phù hợp với thuật toán => phát huy hai tác dụng trên.

Xét bài toán 3 <Luyện thi cho đội tuyển tin học quốc gia 1992>

Cho n số khác 0 là a1, a2, a3, ..., an (n≥1). Tìm số tự nhiên nhỏ nhất khác mọi phần từ của dãy trên.

Giải

Gọi A là tập số đã cho, N là tập số tự nhiên. Ta cần tìm min của tập N-A.

Ví dụ: Nếu A = {8,9,4,1,3} => số cần tìm là 2

Nếu A = {1,2,3,4,5} => số cần tìm là 6

Một cách tự nhiên, nhờ sắp xếp lại dãy đã cho theo trật tự tăng dần rồi tìm ra lỗ hổng đầu tiên, đó chính là số cần tìm.

Ví dụ: Ta sắp xếp A = {1,3,4,8,9} và 2 chính là lỗ hổng,

Còn nếu A = {1,2,3,4,5} thì lỗ hổng đầu tiên là 6, nằm ngoài dãy

Cần lưu ý: Sự khác nhau giữa dãy số và tập số là trong dãy số có thể có những phần tử trùng nhau.

Để tìm lỗ hỏng trong dãy, ta áp dụng kỹ thuật thẻ ghi như sau: Dùng một biến m (thẻ ghi) đoán trước số nhỏ nhất (khác mọi phần tử của A) rồi làm chính xác m tại mỗi bước duyệt A. Ta có phương án sau:

Phương án 1

- Sắp xếp dãy A theo chiều tăng dần
  {a1<=a2<=a3.....<=an}
- Đặt thẻ ghi số (số dự đoán):

```
m:=1;
for i:=1 to n do
    if ai=m then m:=m+1;
{m là số cần tìm khi kết thuc vòng for}
```

Nhận xét:

- Độ phức tạp tính toán chủ yếu do thuật toán sắp xếp gánh chịu. Trường hợp tốt nhất ta cần $n\log_2 n$ đơn vị thời gian.

- Nếu n=1024 = 2^n thì phương án 1 đòi hỏi thời gian là:

  T1 $=2^{10}\log_2 2^{10}$ (sắp xếp) + 2^{10} (duyệt)

  $= 11.2^{10} = 11.n.$

Để cải tiến phương pháp, ta tìm cách trộn lẫn quá trình sắp xếp với quá trình tìm kiếm. Mỗi lần tìm được phần tử min trong số các phần tử cần duyệt thì so sánh với m để chỉnh sửa cho phù hợp. Ta có phương án 2

Phương án 2

```
m:=1; {ghi thẻ số dự đoán cần tìm}
{sắp xếp A theo thuật toán tìm min – thuật toán lựa chọn}
For i:=1 to n do
Begin
    j:=i;
    for k:=i+1 to n do
        if a[j]>a[k] then j:=k;
    {đổi chỗ a[i]&a[j]}
    x:=a[i];
    a[i]:=a[j];
    a[j]:=x;
    {sửa lại thẻ m}
    If a[i]=m then m:=m+1;
End; {for}
```

Các biến trong chương trình được khai báo như sau:

```
Const  n=100;
Var    a: array[1..n] of integer;
       I,k,x,m,j: integer;
```

Nhận xét

Phương án này tồi hơn phương án 1 vì để sắp xếp mảng A, ta cần thời gian n^2, với $n = 2^{10}$ ta tính được :

$T_2 = 2^{10}*2^{10} = 2^{10}*n.$

⇨ Chúng ta hướng sự cải tiến vào CTDL.

Dễ thấy, nếu ta xây dựng được mảng B = b1, b2, b3, ...,bn sao cho i = 1..n,

Bi = 1 nếu ∃ aj=i; = 0 nếu ngược lại. (3)

⇨ Phần tử khác 1 đầu tiên trong B sẽ có chỉ số là số cần tìm. Ví dụ:
A= {5,2,1,1,8,2,6}, n= 7 thì ta sắp xếp được mảng B = (1,1,0,0,1,1,0). Ta viết lại 2 dãy A &B với các chỉ số ở hàng 1 như sau:

Chỉ số:	1	2	3	4	5	6	7
A	(5	2	1	1	8	2	6)
B	(1	1	0	0	1	1	0)

Ta thấy rõ bản chất của hệ thức (3)

$$\begin{cases} b[ai]=1 \text{ nếu } ai \in [1..n], \\ b[i]=0 \text{ nếu ngược lại, } i = 1..n \end{cases} \qquad (4)$$

Kỹ thuật này còn có tên gọi vui là "ai về nhà nấy", nghĩa là ai nếu có đăng ký số nhà trong dãy 1..n thì được cấp phát số nhà là b[ai]. Phương án 3 khi đó sẽ khá trong sáng.

Phương án 3

```
Const   n = 100; n1= n+1;
Var     a:array [1..n]of integer;
        b: array [1..n1] of Boolean;
        i: integer;
Begin
        {Khởi trị cho b}
        For i:= 1 to n1 do b[i]:= false;
        {duyệt A để ai về nhà nấy}
        For i:= 1 to n do
                If(a[i] in [1..n]) and (a[i]>=1) and (a[i]<=n) then
                        b[a[i]]:= true;
        {Tìm lỗ hổng đầu tiên trong b mà b[i]=false}
        m:=1;
```

```
        while b[m] do m:= m+1;
    End;
```

Nhận xét

- Ta khai báo b nhiều hơn a một phần tử (n1 = n+1) để dễ thao tác trong mọi trường hợp, mọi phần tử của A đều có giá trị trong đoạn [1..n]. Khi đó ∀i=1..n mà b[i]= true, giá trị đầu tiên đạt false của b là b[i] với i-n+1, kỹ thuật này được gọi là kỹ thuật đặt lính canh. Nó được dùng rộng rải trong lập trình.

Kỹ thuật đặt lính canh:

Nếu có thể hãy đặt trước các giá trị ở đầu hoặc cuối (mảng, tệp) để làm ranh giới cho việc tìm kiếm. Giá trị của lính canh thường làm cho vòng lặp được tinh giảm sảng sủa.

- Với phương án 3, ta cần thời gian là $T_3 = 3n+2$ cho 3 vòng lặp:
  - 2 vòng for: n1+n = 2n+1
  - 1 vòng while: n1=n+1

  ⇨ Đây chính là thuật toán đòi hỏi thời gian tuyến tính cỡ n. Thay vì thế, ta chịu tốn miền nhớ cho mảng b. Tuy nhiên bằng tính toán nhỏ dưới đây, ta thấy sự tốn kém là không đáng kể:

  Trong hầu hết mọi máy tính, lưu trữ một số nguyên cần 2 byte = 16 bit, lưu trữ một giá trị logic trong chương trình dịch tốt, chỉ cần 1 bit. Như vậy, để có phương án 3 ta chỉ cần dành miền nhớ = 1/16 của a để tổ chức mảng b.

Bảo toàn độ phức tạp?

Thêm miền nhớ, có thể thêm thời gian và ngược lại có thể tiết kiệm miền nhớ thông qua việc tăng thời gian tính toán

Đây không phải là nguyên lý hay định luật = > không nên định hướng theo cách này.

? Có thể bỏ mảng b mà vẫn giữ nguyên thời gian tính toán được chăng? Câu trả lời là có thể được. Bạn đọc suy nghĩ để tìm lời giải này. Gợi ý: Lấy a đóng vai trò là mảng b

Ta sẽ tìm hiểu chi tiết hơn về việc lựa chọn CTDL khi xét về không gian bộ nhớ chương trình.

c.Tinh chế mã

Thông thường để tăng tính hiệu quả của chương trình, người ta thường bàn về các cách tiếp cận bậc cao như:

- Định nghĩa bài toán;

- Kiến trúc hệ thống;
- Thiết kế thuật toán;
- Chọn lựa CTDL.

Tuy nhiên, các cách tiếp cận bậc thấp như thường được thực hiện ở những phần "tốn kém – điểm nóng" của chương trình để cải tiến hiệu xuất. Mặc dù đây là phương pháp không phải lúc nào cũng cần thiết nhưng đôi lúc nó tạo ra sự khác biệt lớn trong hiệu xuất chương trình.

Khi thực hiện tinh chế mã, cần xác định ở đâu là "điểm nóng" của chương trình và hãy tập trung vào điểm nóng ấy. Có nhiều phương pháp để cài tiến hiệu xuất chương trình, hãy dùng phương pháp này sau cùng, phương pháp này còn được dùng để giảm không gian chiếm bởi chương trình.

"điểm nóng" là một đoạn mã chiếm phần lớn thời gian chạy của chương trình. Một số kỹ thuật tinh chế mã này như sau:

1. *Tập hợp những biểu thức chung*: Một phép tính tốn nhiều thời gian xuất hiện nhiều lần, hãy thực hiện chỉ một lần và lưu lại kết quả.
   Ví dụ:
   + Xét lệnh sau:
   Sqrt(dx*dx+dy*dy)+((sqrt(dx*dx+dy*dy)>0)?.....
   ⇨ Hàm sqrt gọi 2 lần trong một biểu thức, hãy tính sqrt một lần và dùng giá trị đó ở 2 nơi.

   + Nếu một phép tính thực hiện bên trong vòng lặp nhưng không phụ thuộc vào những thay đổi bên trong vòng lặp thì hãy đưa nó ra ngoài

   Ví dụ: Xét lệnh sau:

   ```
   For(i=0;i<nstarting(c); i++){
   }
   ```

   Nên thay bằng các lệnh:

   ```
   n= nstarting(c);
   For(i=0;i<n; i++){
   }
   ```

2. *Thay các phép tính chạy chậm bằng các phép tính chạy nhanh hơn*: Thường thay phép nhân thành bằng phép cộng/phép dịch bit
3. *Khử hay loại bỏ các vòng lặp*: Nếu thân vòng lặp không quá dài và không có nhiều bước lặp, có thể sẽ hiệu quả hơn khi viết rõ.

Ví dụ:

```
For (i=0;i<3;i++)
a[i]=b[i]+c[i];
```

⇨ Phí tổn khi thực hiện lệnh này vì nó làm chạm đến các bộ xử lý hiện đại bằng cách ngắt dòng thực thi chương trình, đặc biệt là trường hợp rẽ nhánh.

⇨ Nên thay lệnh trên bằng 3 lệnh tuần tự sau:

```
a[0]=b[0]+c[0];
a[1]=b[1]+c[1];
a[2]=b[2]+c[2];
```

4. *Lưu lại các giá trị thường dùng, tránh tính lại*
5. *Sử dụng cấp phát bộ nhớ đặc biệt*:

   Trong chương trình thường có nhiều lệnh cấp phát ô nhớ (cùng kích thước) bằng toán tử new/mallocc => chiếm phần lớn thời gian chạy. Nên cấp phát một lần đủ số ô nhớ cần thiết, nếu có thể nên có thể dùng lại các ô nhớ vừa cấp phát.

6. *Viết lại bằng ngôn ngữ cấp thấp hơn*:
   Chương trình có thể tăng tốc rất nhiều nếu dùng các đoạn mã lệnh có tính chất phụ thuộc phần cứng. Tuy nhiên cần cân nhắc cách này vì nó sẽ phá hủy sự linh động của chương trình và làm cho việc bảo trì, sửa chữa trong tương lai sẽ trở nên khó khăn hơn nhiều.

3.5.2 Không gian bộ nhớ

Trong những ngày đầu của ngành kỹ thuật máy tính, các nhà lập trình bị hạn chế bởi nhiều bởi bộ nhớ máy tính nhỏ. Ngày nay vấn đề này không còn là điểm nóng nữa. Tuy vậy, khi thiết kế chương trình không phải lúc nào ta cũng có đủ bộ nhớ để sử dụng bởi nhiều lý do khác nhau.

a.**Không gian dữ liệu**

Các nguyên tắc để làm giảm không gian lưu trữ dữ liệu là:

- Đảm bảo tính đơn giản của dữ liệu

- Trong một số trường hợp, đừng lưu trữ, hãy tính lại khi cần thiết

- Nén dữ liệu sau đó giải nén khi cần dùng

- Sử dụng các nguyên tắc cấp phát bộ nhớ động

- Dùng kiểu dữ liệu có kích thước nhỏ nhất có thể.

Ví dụ: Xét bài toán

Trên một bản đồ chứa 2000 điểm (bản đồ quân sự) được đánh số từ 1 -> 2000. Một vị trí trên bản đồ được xác định bằng một cặp tọa độ (x,y) với x là một số nguyên nằm trong khoảng 1..200, y là số nguyên nằm trong khoảng từ 1..150. Chương trình dùng cặp (x,y) để xác định điểm nào được chọn trong 2000 điểm đã cho.

Lời giải

* Không gian lưu trữ 1

Một cách hiển nhiên để lưu bản đồ là dùng mảng 2 chiều 200*150. Ứng với mỗi x.y sẽ chứa một giá trị trong khoảng từ 1..2000 hoặc chứa giá trị 0.

200									57					
...														
127														
...														
6	538				965									
5														
4				320			32		7					
3	17													
2		89						15						
1	2	3	4	5	6	7	8	...						150

Nhận xét

Việc dùng bảng này, thời gian truy cập nhanh, nhưng nó chiếm đến 200*150 = 30.000 ô nhớ, và giả sử để lưu dữ liệu (ở đây là các số thuộc 1..2.000) cần 2 byte thì cần 30.000*2byte = 60.000 ô nhớ

Tuy nhiên trong mảng trên thì đa số là các giá trị 0 (giá trị không dùng). Do vậy, nếu ta dùng cái nào thì lưu giá trị đó (ở đây là 2.000 giá trị) thì việc chiếm bộ nhớ sẽ giảm đi đáng kể

* Không gian lưu trữ 2

Thay vì dùng một mảng 2 chiều, ta dùng 3 mảng 1 chiều như sau:

- Mảng Value: Dùng để chứa các giá trị (ở đây là từ 1..2.000) theo từng cột, như vậy cần 2.000 ô nhớ

- Mảng Row: Chứa chỉ số hàng tương ứng với các giá trị ở mảng value => Cần 2.000 ô nhớ

- Mảng Col: Chứa chỉ số cột tương ứng với các giá trị ở mảng Value = > Cần 2000 ô nhớ

=> Tổng cộng cần 2.000* 3= 6.000 ô nhớ <30.000 ô nhớ rất nhiều

* Không gian lưu trữ 3

Bạn đọc cải tiến không gian lưu trữ 2 để tránh trùng lặp các chỉ số hàng, cột của các value trong mảng value.

b. **Không gian chương trình**

Trong một số chương trình, đôi lúc kích thước của chính bản thân nó là vấn đề. Hãy định nghĩa các chương trình con, sử dụng các bộ thông dịch chuyên dụng để làm chương trình đơn giản, trong sáng và rõ ràng hơn, dễ bảo trì hơn.

3.6 Case Tools hỗ trợ trong cài đặt

Các CASE tools hỗ trợ hỗ trợ cho việc cài đặt các thực thể mã (code artifacts, còn gọi là các thành phần/component) như:

- Các công cụ quản lý phiên bản, tích hợp,
- Các công cụ xây dựng (build Tools),
- Các công cụ quản lý cấu hình (Configuration management Tools),
- .... là rất cần thiết.

Các workbenches điều khiển cấu hình bán trên thị trường như:

- PVCS & Source Safe
- Công cụ điều khiển cấu hình mã nguồn mở thông dụng là CVS.

Sự tích hợp của các workbenches tạo thành môi trường/Environments

3.7 Các ngôn ngữ lập trình thế hệ thứ 4

Máy tính đầu tiên không có các trình thông dịch, cũng không có các trình biên dịch. Chúng được lập trình theo mã nhị phân hoặc được cứng hóa với các bản mạch được cắm trực tiếp vào máy tính.

Mã máy là ngôn ngữ thế hệ thứ nhất. Ngôn ngữ này rất khó sử dụng, đòi hỏi người lập trình phải hiểu biết sâu về phần cứng máy tính.

Các ngôn ngữ thế hệ thứ 2 là Assembler, được phát triển sau những năm 1940 và trước những năm 1950s. Nhìn chung mỗi chỉ thị Assembler được chuyển dịch thành một chỉ thị mã máy. Mã nguồn Assembler có chiều dài tương tự mã máy. Ngôn ngữ này dễ sử dụng hơn ngôn ngữ máy, bởi nó hướng đến cú pháp của ngôn ngữ tự nhiên.

Ngôn ngữ thế hệ thứ 3 (hoặc ngôn ngữ mức cao) như C, C++, Pascal, Java, ... Mỗi chỉ thị của ngôn ngữ bậc cao được dịch thành 5/10 chỉ thị máy. Chiều dài mã ngôn ngữ mức cao ngắn hơn mã Assembler tương ứng. Các ngôn ngữ bậc 3 đơn giản hơn, dễ hiểu và dễ bảo trì hơn so với các ngôn ngữ thế hệ trước.

Sau những năm 1970, mục tiêu chính của việc thiết kế ngôn ngữ thế hệ 4 (4 GL) là mỗi phát biểu 4 GL tương ứng với 30 -> 50 chỉ thị mã máy. Các sản phẩm được viết = 4 GL như Focus/Natural là ngắn hơn, dễ phát triển và dễ bảo trì hơn.

Đặc biệt, nhiều ngôn ngữ 4 GL là các ngôn ngữ phi thủ tục (nonprocedural). Ví dụ, xét đoạn mã sau:

For every *survegor*

If *rating* ***is*** *excellent*

Add *500$* ***to*** *salary*

Trình biên dịch của 4 GL dịch chỉ thị phi thủ tục này thành một chuỗi các chỉ thị mã máy mà có thể được thực hiện một cách có thủ tục.

4 GL được hỗ trợ bởi các CASE Workbenches & Environments mạnh mẽ. Các Workbenches và Environments cả 2 đều có những điểm mạnh và yếu, không tránh khỏi việc đưa một số lượng lớn các CASE vào với mức độ thuần thục thấp. Các thất bại của 4 GL thường gắn với CASE Environment hơn là từ chính mục đích của 4 GL.

Các 4 GL gồm DB2, Oracle, PowerBuilder, ...

Nhiều tổ chức sử dụng 4 GL trên 3 năm đã cảm nhận được các lợi ích mà 4 GL mang lại là vượt xa so với các chi phí đối với nó.

Mục tiêu thứ 2 của 4 GL là hướng người dùng vào công việc lập trình và bảo trì sản phẩm phần mềm (*end – user programming*).

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 3

CHƯƠNG 4 CHỨNG MINH TÍNH ĐÚNG ĐẮN CỦA CHƯƠNG TRÌNH

4.1 Tính đúng đắn của sản phẩm

Tính đúng đắn của chương trình thể hiện ở chỗ, chương trình thực hiện chính xác những mục tiêu và chức năng đã được đề xuất ở giai đoạn thiết kế

Ví dụ:

Nếu chức năng của chương trình là sắp xếp một tệp có số lượng bản ghi tùy ý theo 1 trường tùy ý với chiều tăng, giảm tùy ý thì tình huống sau đây sẽ vi phạm tính đúng đắn của chương trình:

+ Không thực hiện (treo máy) được với những tệp trống (không chứa bản ghi nào)

+ Không chạy hoặc chạy sai với những tệp có trên 100 trường hoặc trên 10^4 bản ghi.

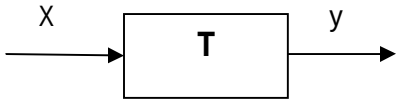
+ Không chạy hoặc chạy sai với những tệp tin có chứa trường có chiều dài >125 byte trong trường hợp có yêu cầu sắp xếp theo trật tự giảm dần.

Tính đúng đắn của sản phẩm phần mềm được xác minh qua những căn cứ sau đây:

- Tính đúng đắn của thuật toán
- Tính tương đương của chương trình với thuật toán: Thuật toán có thể đúng nhưng chương trình lập ra không tương đương với thuật toán nên dẫn đến kết quả sai. Trong trường hợp này người ta gọi là việc mã hóa bị sai
- Tính đúng đắn của chương trình có thể chứng minh trực tiếp trong văn bản chương trình
- Tính đúng đắn của chương trình có thể được khẳng định dần dần qua việc kiểm thử, qua việc áp dụng chương trình trong một khoảng thời gian đủ lớn, trên một diện khá rộng với tần xuất sử dụng cao.
- Các tác giả sản phẩm cần cung cấp đầy đủ những luận cứ và kết quả xác nhận tính đúng đắn của sản phẩm. Những tư liệu đó bao gồm:
    - Chương trình (listing) có kèm theo các luận đề phục vụ cho việc chứng minh
    - Các phương pháp và kỹ thuật kiểm thử chương trình
    - Các kết quả chạy thử
    - Các giấy chứng nhận, nhận xét của cá nhân và cơ sở đã khảo sát và áp dụng chương trình.

4.2 Khái niệm chung, cách đặt vấn đề chứng minh

Chương trình được đặc tả như một bộ biến đổi tuần tự T chuyển cái vào x thành cái ra y. Khi lập trình ta đã xác định mục tiêu rất rõ ràng, tức là chúng ta đã biết cái vào x và cái ra y. Việc cần làm trong khi lập trình là xác định dãy các thao tác tuần tự T sao cho:



{ vào x }

T?

{ ra y }

Ví dụ1: Đặc tả đầu vào/ra của chương trình tìm ước chung lớn nhất của 2 số tự nhiên

Đặc tả phi hình thức	***Đặc tả hình thức***
{vào: a:N, b:N} *T?* *{Ra: tìm d∈N sao cho:* *a chia hết cho d và* *b chia hết cho d và* *nếu có c để a chia hết cho c và b chia hết cho c, thì c≤d}*	*{P: a:N, b:N}* *T?* *{Q:d:N|a ⋮ d, b ⋮d, (∃c|a ⋮ c, b ⋮ c) => c ≤ d)}* *Chú ý: Liên từ "và" được thay bằng dấu phẩy*

Ví dụ 2: Đặc tả đầu vào, ra của chương trình tráo 2 cốc nước x và y cho nhau

Đặc tả phi hình thức	***Đặc tả hình thức***
{vào: x chứa lượng nước a. *Y: chứa lượng nước b}* *T?* *{Ra: x chứa lượng nước b,* *Y chứa lượng nước a}*	*{P: x=a, y=b}* *T?* *{Q:x=b, y=a}*

Trong đó:

P: cái vào, là mệnh đề nói về cái ta có/miền xác định của chương trình T

Q: Cái ra, là mệnh đề nói về cái ta muốn có/miền giá trị của T

Nếu xem T như một ánh xạ thì ta có thể viết:

T: P $\rightarrow$ Q

T biến thiên trên miền P và cho ra giá trị trên miền Q. Ta biết T là một dãy liên tiếp các thao tác:

T = T1, T2, ...,Tn

Vậy bằng cách nào đó ta khẳng định được liên tiếp các mệnh đề sau đây:

- Thao tác T1 biến đổi P thành P1
- Thao tác T2 biến đổi P1 thành P2
- Thao tác T3 biến đổi P2 thành P3
- ...
- Thao tác Ti biến đổi Pi-1 thành Pi
- ...
- Thao tác Tn biến đổi Pn-1 thành Pn/Pn chính là Q

Thì ta có thể đảm bảo rằng chương trình T là đúng đắn. Việc làm trên được gọi là chứng minh tính đúng đắn của chương trình T. Trong chương trình T, các khẳng định được viết dưới dạng các chú thích như sau:

{P}

T1;

{P1}

T2;

...

Ti;

{Pi}

..

Tn

{Pn => Q}

Ví dụ: Ta thử chứng minh chương trình tráo 2 cốc nước x và y

{P: x=a, y=b (giả sử ban đầu x chứa lượng nước a, y chứa lượng nước b)}

{mượn thêm cốc z}

Đổ x sang z;

{P1: ta có x=0 (cốc rỗng), y=b, x=a}

Đổ y sang x;

{P2: ta có x=b, y=0, z=a}

Đổ z sang y;

{P3: ta có x=b, y=a} => đây chính là mệnh đề Q cần tìm

Ví dụ 2: Cho hai biến nguyên x và y chứa các giá trị trong khoảng -1000 đến +1000. Hãy tráo trị của chúng mà không dùng biến phụ.

Đoạn chương trình sau vừa thực hiện, vừa chứng minh tính đúng đắn của kết quả khi thực hiện.

Lưu ý:

Khi thực hiện phép gán thì biến trao chỉ sao chép trị của nó sang biến nhận. Sau khi gán, giá trị của biến trao vẫn được bảo toàn.

{P: x=a, y=b}

x:= x+y;

{P1: x=a+b, y=b}

y:=x-y;

{P2: x=a+b, y=a}

x:=x-y;

{P3: x=b, y=a} => đây chính là mệnh đề Q cần tìm.

4.3 Hệ tiền đề của Hoare

Ta đã biết rằng, một chương trình T bất kỳ là một dãy liên tiếp các cấu trúc điều khiển & các lệnh T1, T2, ...,, Tn. Boehm và Jacopini đã chứng minh rằng T có thể viết dưới dạng một dãy liên tiếp các cấu trúc tuần tự và cấu trúc điều kiện lặp với điều kiện trước.

Năm 1969, Hoare đã đề xuất ra một hệ tiền đề (*bao gồm 4 tiền đề H1->H4*) giúp cho việc chứng minh tính đúng đắn của chương trình. Hệ tiền đề đã nêu rõ quan hệ giữa mệnh đề trước {P} và mệnh đề sau {Q} khi chịu tác động của một cấu trúc điều khiển hoặc một lệnh gán Ti:

{P} Ti {Q} *Nếu ta có P là đúng, và nếu ta thực hiện thao tác Ti thì sẽ thu được Q là đúng.*	*Ta thường nói vắn tắt là: Nếu P và Ti thì Q*

Chúng ta sẽ tìm hiểu hệ tiền đề Hoare dưới dạng tính chất của phép gán & các cấu trúc điều khiển của Pascal.

H1: Tính chất của phép gán

{P} x:= E {Q}

{P=>Q[x/E]}

Điều kiện đủ để Q đúng sau khi thực hiện lệnh gán x:=E (E là một biểu thức) là trước khi thực hiện phép gán nói trên ta phải có Q[x/E] đúng (Q[x/E] là mệnh đề thu được từ Q bằng cách thay thế ∀ xuất hiện x trong Q bằng E).

Một dạng phát biểu khác của tính chất trên là:

Điều kiện đủ để có Q sau khi thực hiện lệnh gán x:=E là trước đó ta phải có P suy dẫn được ra Q[x/E].

Ví dụ:

1. {x>n} x:= x+1 {x>n+1}
   Thật vậy: Đặt Q: {x>n+1} ta có Q[x/(x+1)]: {x+1>n+1} hay {x>n}=>điều phải chứng minh (vì trùng với giả thiết P)
2. {x(x-1)<0} x:= 1/x+y {x>y+1}

   Thật vậy: Đặt Q: {x>y+1} ta có Q [x/(1/x+y)]: 1/x+y>y+1} hay {1/x>1} (1), mặt khác từ giả thiết P: {x(x-1)<0} theo định lý về dấu của tam thức bậc 2 => 0<x<1, từ đây ta có 1/x>1 (2).

   Từ (1) và (2) => điều phải chứng minh.

H2: Tính chất của dãy thao tác

Nếu {P} T1 {Q}

Và {Q} T2 {R}

Thì {P} T1; T2 {R}

Ví dụ:

Chứng minh:

{0<x<1}

x:=1/x +y;

x:=x+1;

{x>y+2}.

Đặt R: {x>y+2}, ta có R[x/(x+1)]: {x+1>y+2} hay {x>y+1}. Từ đây ta có:

{x>y+1}

x:=x+1;

{x>y+2}.

Ta còn phải chứng minh:

{0<x<1}

x:=1/x+y;

{x>y+1}.

Đặt Q: {x>y+1}, ta có Q[x/(1/x+y)]:{1/x+y>y+1} hay {1/x>1} (1)

Theo giả thiết P: {0<x<1} => {1/x>1} (2)

Từ (1) và (2) => điều phải chứng minh.

H3: Tính chất của cấu trúc rẽ nhánh	
<u>Dạng 1</u> *Nếu {P, C} A {Q}* *Và {P, !C} => {Q}* *Thì {P} if C then A {Q}*	<u>Dạng 2</u> *Nếu {P, C} A {Q}* *Và {P, !C} B {Q}* *Thì {P} if C then A* *Else B {Q}*

Ví dụ:

1. **Chứng minh**

{x<y}

If x<-2 then

Begin

```
            y:=x*x;
            x:=-x;
        End
    Else
        y:=x+y+3;
    {y>x+1}
```

Ta phải chứng minh hai khẳng định sau:

1.a)	*1.b)*

{x<y, x<-2}
y:= x*x;
x:=-x;
{y>x+1}
	{x<y, x≥-2}
	y:= x+y+3;
	{y>x+1}

Chứng minh khẳng định 1a)

Đặt Q: {y>x+1}, ta có Q[x/-x]: {y>-x+1}

Đặt R: {y>-x+1}, ta có R[y/x*x]: {x^2 >-x+1} hay {x^2 +x - 1>0}, theo định lý về dấu của tam thức bậc 2, muốn có {x^2+x-1>0} thì ta phải có (-1-$\sqrt{5}$)/2>x $\vee$ (-1+$\sqrt{5}$)/2<x (1)

Theo giả thiết P: x<-2 nên x<(-1-$\sqrt{5}$)/2 (2)

Từ (1) và (2) => điều phải chứng minh.

Chứng minh khẳng định 1b)

Đặt Q: {y>x+1}, ta có Q[y/(x+y+3)]: {x+y+3>x+1} hay {y>-2} (1)

Từ giả thiết P: {x<y, x≥-2}=> y>-2 (2)

Từ (1)&(2) => điều phải chứng minh.

2. Chứng minh

```
{x>0}
If odd(x) then x:=x+1;   {if x là lẻ thi....}
y:= x div 2;
{x=2y}
```

- Đặt R: {x=2y}, ta có R[y/(x div 2)]: {x=2(x div 2)} hay x là chẵn.
- Tiếp theo ta chứng minh rằng:

{x>0}

If odd(x) then x:=x+1;

{x chẵn}.

Ta phải chứng minh 2 khẳng định sau

1.a)	*1.b)*
{x>0, x lẻ} x:=x+1; {x chẵn}	{x>0, x chẵn} {x chẵn} ⇨ Luôn đúng

Chứng minh 1a)

Thật vậy, Nếu đặt R: {x=2k}, ta có R[x/(x+1)]:{x+1=2k} hay {x=2k+1} => x là một số lẻ với k nguyên bất kỳ.

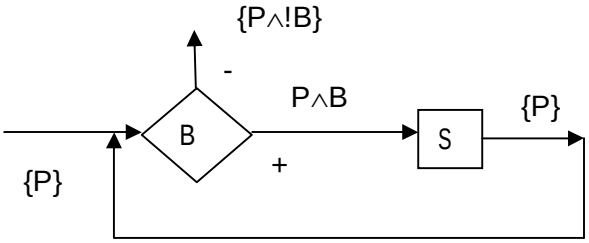
⇨ Tổng hợp lại ta có:
{x>0}
If odd(x) then x:=x+1;
{x=2k}
y:= x div 2;
{x=2y}

⇨ Điều phải chứng minh.

H4: Tính chất của vòng lặp với điều kiện trước	
Dạng 1 ***While B do S*** ***{!B}*** *Trong đó:* !B là phủ định của B	Dạng 2 ***Nếu {P và B} S {P}*** ***Thì :*** ***{P}*** ***While B do S {P}*** ***{P và !B}***

Để ý đến dạng thứ 2 của H4, đó chính là tính chất hết sức quan trọng của vòng lặp. Nếu {P} là một bất biến đối với S theo điều kiện B (nghĩa là nếu P đúng và B đúng và sau khi thực hiện S ta thu được P đúng) thì P sẽ bất biến đối với chính vòng lặp.

Ví dụ:



1.Chứng minh

{P: x⋮ $d, y \vdots d, x > 0, y > 0$}

While x>=y do

x:=x-y;

{Q: x⋮ $d, y \vdots d$}

Thật vậy:

Ta có, Q[x/(x-y)]: {(x-y)⋮ $d, y \vdots d$} (1)

Từ giả thiết P ta có:

x=ad, y=bd, do đó x-y=(a-b)d => x-y⋮ d , y⋮ d *theo giả thiết* (2)

Từ (1)&(2)=> điều phải chứng minh.

Đoạn chương trình trên được viết lại như sau;

{P: x⋮ $d, y \vdots d, x > 0, y > 0$}

While x>=y do

{M: x⋮ $d, y \vdots d, x \geq y, x > 0, y > 0$}

x:=x-y;

{Q: x⋮ $d, y \vdots d, y > 0, x \leq 0$}

Đoạn chương trình này cho ta biết:

Ước số chung (x,y) = Ước số chung (Dư(x,y),y) và

{x⋮ $d, y \vdots d$} chính là bất biến của vòng lặp.

2.Chứng minh đoạn lệnh - Tìm kiếm một phần tử có trong mảng.

Cho mảng a[1..n], n≥1, và một giá trị x có mặt trong mảng a nhưng không biết là phần tử thứ mấy. Hãy xác định chỉ số i đầu tiên thỏa mãn điều kiện a[i]=x.

Chứng minh

i:=1;

{P: a[i]=x∨a[i]<>x}

While a[i]<>x do

{Q: a[i]<>x}

i:=i+1;

{R: a[i]=x ∨ a[i]<>x}

{M: a[i]=x}

Thật vậy:

Với i=1, ta chưa thể nói gì về a[i]. Do đó ta có {P: a[i]=x ∨ a[i]<>x} (dâu ∨ là dấu hoặc logic). Sau khi kiểm tra điều kiện của while là a[i]<>x, ta có {Q: a[i]<>x}. Khi tăng i thêm một đơn vị, dĩ nhiên ta không thể biết gì về a[i] có bằng x hay không, do đó ta có mệnh đề R, đây chính là bất biến của vòng lặp. Khi vòng lặp kết thúc, tức a[i]=x/là điều kiện sai của vòng while. Ta có mệnh đề:

R∧{a[i]=x} => Mệnh đề {M: a[i]=x}.

3.Tìm kiếm một phần tử trong mảng.

Cho mảng a[1..n], n>=1 và một giá trị x. Hãy cho biết x có trong mảng hay không. Nếu có thì xác định chỉ số i đầu tiên để a[i]=x, nếu không có thì cho i = 0.

Xét đoạn mã sau:

i:=1;

while (i<=n) and (a[i]<>x) do i:=i+1;

Ta viết lại đoạn mã trên với các chứng minh đi kèm:

i:=1;

{P: a[i]=x ∨ a[i]<>x}

While (i<=n) and (a[i]<>x) do

{Q: (i<=n)∧(a[i]<>x)}

i:=i+1;

{R: ((i>n)∧(∀k=1..n|a[k]<>x)) ∨

((a[i]=x∧(1<=i<=n))}

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 4

TÀI LIỆU THAM KHẢO

[1] Nguyễn Xuân Huy, 1993, *Công nghệ phần mềm*.

[2] Stephen R. Schach, 2005, *Object – Oriented & Classical Software Engineering*, ISBN 0 -07-286551-2, Mc Graw Hill.

[3] Ngô Trung Việt; *Các nguyên lý lập trình.*

[4]. COMP 248, 2006, *Programming Methodology*.

[5] Don Knuth, 1998, *The Art of Computer Programming*; Volume 3, 2nd Edition, Addison Wesley.

[6] Peter Van Roy and Self Haridi, 2004, *Concepts, Techniques, and Models of Computer Program*, The MIT Press.