Mục lục

CHƯƠNG 1 ĐẠI CƯƠNG VỀ CÔNG NGHỆ PHẦN MỀM 5

I. KHÁI QUÁT VỀ LỊCH SỬ LẬP TRÌNH 5

I.1. Lập trình tuyến tính 5

I.2. Lập trình có cấu trúc 6

I.3. Lập trình định hướng đối tượng (ĐHĐT) 6

I.4. Lập trình trực quan 7

I.5. Những tư tưởng cách mạng trong lập trình 7

II. CÁC PHƯƠNG DIỆN CỦA CÔNG NGHỆ PHẦN MỀM 8

II.1. Công nghệ phần mềm là gì? 8

II.2. Những yếu tố chất lượng bên ngoài và bên trong 8

II.3. Sản phẩm phần mềm là gì ? 9

III. NHỮNG NỘI DUNG CƠ BẢN CỦA CNPM 11

III.1. Tổng quan về công nghệ phần mềm 11

III.2. Chu kỳ sống của phần mềm 12

CHƯƠNG 2 THIẾT KẾ PHẦN MỀM 18

I. NỀN TẢNG CỦA THIẾT KẾ PHẦN MỀM 18

II. PHƯƠNG PHÁP LẬP TRÌNH CẤU TRÚC 20

II.1. Khái niệm về lập trình cấu trúc 22

II.2. Những ý tưởng cơ bản lập trình cấu trúc 22

II.3. Các cấu trúc điều khiển chuẩn 25

II.4. Một số ví dụ viết chương trình theo sơ đồ khối 28

III. CẤU TRÚC TỐI THIỂU 29

III.1. Các cấu trúc lồng nhau 31

IV. LẬP TRÌNH ĐƠN THỂ 32

IV.1. Khái niệm về đơn thể 32

IV.2. Mối liên hệ giữa các đơn thể 33

IV.2.1. Phân loại đơn thể 33

IV.2.2. Tổ chức một chương trình có cấu trúc đơn thể 33

V. PHÁT TRIỂN CHƯƠNG TRÌNH BẰNG TINH CHẾ TỪNG BƯỚC 35

V.1. Nội dung phương pháp 35

V.2. Ví dụ minh hoạ 36

V.2.1. Ví dụ 1 36

V.2.2. Bài toán 8 quân hậu 38

V.3. Sửa đổi chương trình 42
VI. PHỤ LỤC - ĐƠN VỊ TRONG TURBO PASCAL 50
VI.1. Giới thiệu Unit 50
VI.2. Cấu trúc của Unit 50
VI.3. Cách sử dụng Unit 52
VI.4. Ví dụ về Unit 53
VI.5. Bài tập 55
CHƯƠNG 3 HỢP THỨC HÓA PHẦN MỀM 57
I. XÁC MINH VÀ HỢP THỨC HÓA PHẦN MỀM 57
II. CHỨNG MINH SỰ ĐÚNG ĐẮN CỦA CHƯƠNG TRÌNH 58
II.1. Suy luận Toán học 59
II.1.1. Các quy tắc suy luận Toán học 59
II.1.2. Khái niệm về chứng minh tính đúng đắn của chương trình 60
II.1.3. Tiên đề và quy tắc suy diễn 61
II.1.4. Quy tắc điều kiện if B then P 62
II.1.5. Quy tắc điều kiện if B then P else Q 63
II.1.6. Quy tắc vòng lặp while 63
II.1.7. Các quy tắc khác 64
II.2. Phương pháp của C.A.R. Hoare 66
II.2.1. Phát biểu 66
II.2.2. Chứng minh tính đúng đắn từng phần của Div 66
II.3. Chứng minh dừng 69
II.3.1. Chứng minh dừng của một chương trình 69
II.3.2. Chứng minh dừng của Div 70
II.3.3. Đánh giá một chương trình lặp 71
III. XÂY DỰNG CHƯƠNG TRÌNH 72
III.1. Mở đầu 72
III.2. Bài toán cờ tam tài 73
III.2.1. Lời giải thứ nhất 74
III.2.2. Lời giải thứ hai 75
III.2.3. Chứng minh tính đúng đắn của chương trình (I) 76
III.3. In ra một danh sách theo thứ tự ngược 80
III.3.1. TILDA1 81
IV. CÁC TIÊN ĐỀ VÀ QUY TẮC SUY DIỄN 82
IV.1. Điều kiện trước yếu nhất và điều kiện sau mạnh nhất của một dãy lệnh 82
IV.1.1. Hàm fppre 83
IV.1.2. Hàm fppost 83
IV.1.3. Sử dụng điều kiện trước yếu nhất và điều kiện sau mạnh nhất để chứng minh tính đúng đắn của chương trình 84

IV.2. Các tiên đề gán ... 86
IV.2.1. Điều kiện trước yếu nhất và điều kiện sau mạnh nhất của lệnh gán ... 86
IV.2.2. Quy tắc tính toán điều kiện sau mạnh nhất của một phép gán ... 87
V. BÀI TẬP ... 89

CHƯƠNG 4 THỬ NGHIỆM CHƯƠNG TRÌNH ... 90
I. KHẢO SÁT PHẦN MỀM ... 90
II. CÁC PHƯƠNG PHÁP THỬ NGHIỆM ... 92
II.1. Định nghĩa và mục đích thử nghiệm ... 92
II.2. Thử nghiệm trong chu kỳ sống của phần mềm ... 94
II.2.1. Thử nghiệm đơn thể ... 94
II.2.2. Thử nghiệm tích hợp ... 95
II.2.3. Thử nghiệm hệ thống ... 96
II.2.4. Thử nghiệm hồi quy ... 97
II.3. Dẫn dắt các thử nghiệm ... 97
II.4. Thiết kế các phép thử phá hủy (Defect Testing) ... 98
II.4.1. Các phương pháp dựa trên chương trình ... 98
II.4.2. Các phương pháp dựa trên đặc tả ... 100
II.4.3. Kết luận ... 101
II.4.4. Các tiêu chuẩn kết thúc thử nghiệm ... 101
II.5. Các phép thử nghiệm thống kê ... 102
II.5.1. Mở đầu ... 102
II.5.2. Ước lượng độ ổn định của một phần mềm ... 104

CHƯƠNG 5 ĐẶC TẢ PHẦN MỀM ... 105
I. MỞ ĐẦU ĐẶC TẢ PHẦN MỀM ... 105
I.1. Khái niệm về đặc tả ... 105
I.1.1. Đặc tả là gì ? ... 105
I.1.2. Các phương pháp đặc tả ... 105
I.1.3. Các thí dụ minh họa ... 106
I.2. Đặc tả và lập trình ... 107
II. ĐẶC TẢ CẤU TRÚC DỮ LIỆU ... 109
II.1. Khái niệm về Cấu trúc dữ liệu cơ sở vectơ ... 109
II.1.1. Dẫn nhập ... 109
II.1.2. Đặc tả hình thức ... 110
II.2. Truy nhập một phần tử của vectơ ... 110
II.3. Các thuật toán xử lý vectơ ... 111
II.3.1. Truy tìm tuần tự một phần tử của vectơ (sequential search) ... 111
II.3.2. Tìm kiếm nhị phân (Binary search) ... 113
III. ĐẶC TẢ ĐẠI SỐ : MÔ HÌNH HÓA PHÁT TRIỂN PHẦN MỀM ... 117
III.1. Mở đầu ... 117
III.2. Phân loại các phép toán ... 119
III.3. Hạng và biến ... 120
III.4. Phép thế các hạng ... 120

III.5. Các thuộc tính của đặc tả 122
III.5.1. Mô hình lập trình (triển khai) *122*
III.5.2. Mô hình đặc biệt *123*
III.5.3. Mô hình đồng dư *123*
III.6. Phép chứng minh trong đặc tả đại số 123
III.6.1. Lý thuyếttương đương *124*
III.6.2. Khái niệm về lý thuyết quy nạp *125*
III.6.3. Chứng minh tự động bởi viết lại *126*
III.6.4. Phân cấp trong đặc tả đại số *128*
IV. ĐẶC TẢ HAY CÁCH CỤ THỂ HÓA SỰ TRỪU TƯỢNG 129
IV.1. Đặc tả phép thay đổi bộ nhớ 129
IV.2. Hàm 131
IV.3. Hợp thức hóa và phục hồi 134
IV.4. Bắt đầu triển khai thực tiễn 137
IV.5. Phép hợp thành (cấu tạo) 140
IV.6. Triển khai thứ hai 141
IV.7. Triển khai thực hiện lần thứ ba 146
IV.8. Đặc tả làm gì ? 149

CHƯƠNG 1

Đại cương về công nghệ phần mềm

I. Khái quát về lịch sử lập trình

Lập trình (programming), hay *lập chương trình cho máy tính điện tử* (MTĐT) là một ngành còn rất mới mẻ. MTĐT đầu tiên lập trình được mới chỉ xuất hiện cách đây hơn bốn mươi năm [1]. Suốt hơn bốn thập kỷ qua, lập trình không ngừng được cải tiến và phát triển, càng ngày càng hướng về nhu cầu của người lập trình.

Lập trình là một công việc nặng nhọc, năng suất thấp so với các hoạt động trí tuệ khác. Ví dụ nếu một sản phẩm phần mềm khoảng 2000 – 3000 dòng lệnh đòi hỏi 3 người lập trình chính trong vòng 6 tháng thì năng suất mỗi người chỉ dao động trong khoảng từ 5 đến 6 lệnh mỗi ngày (?!).

Chính vì các sản phẩm phần mềm khi tung ra thị trường chưa thực sự hoàn hảo ngay nên người ta thường dùng mẹo thương mại bằng cách gán cho sản phẩm một cái đuôi "phiên bản" (version) để nói rằng phiên bản ra sau đã khắc phục được những khiếm khuyết của phiên bản trước đó.

Ví dụ 1 :

Hệ điều hành MS–DOS đã có các phiên bản 1.0, 3.3, 5.0, 6.0, 7.0 v.v...

Microsoft Windows đã có các phiên bản 1.0, 2.0, 3.0, 3.1, 3.11.

Nay là Windows 95, 97, 98 v.v...

Turbo Psacal của hãng Borland Inc. đã có các phiên bản 5.0, 6.0, 7.0, 8.0 v.v...

I.1. Lập trình tuyến tính

Với những MTĐT đầu tiên, người ta sử dụng ngôn ngữ máy (machine language) hay ngôn ngữ bậc thấp (low level) để lập trình và dùng các khoá cơ khí để nạp chương trình vào máy. Theo đà phát triển của các thiết bị phần cứng, các ngôn ngữ bậc cao (high level) với các dòng lệnh tựa tiếng Anh bắt đầu được sử dụng. Máy sẽ dịch chương trình đó sang ngôn ngữ máy trước khi thực hiện.

Với những ngôn ngữ lập trình ban đầu, chương trình viết ra gồm những dòng lệnh có khuynh hướng nối nhau theo dãy dài, khó hiểu về mặt logic. Người ta sử

[1] ENIAC (Electronic Numerical Integrator and Computer) là chiếc MTĐT đầu tiên ra đời năm 1945 tại trường Đại học Tổng hợp Pensylvania, nước Mỹ.

dụng các lệnh nhảy (goto) để điều khiển chương trình một cách tuỳ tiện. Chương trình là một mớ rối rắm không khác gì món mì sợi (spaghetti) của nước Ý.

Các ngôn ngữ lập trình tuyến tính không kiểm soát được những sư thay đổi của dữ liệu. Mọi dữ liệu sử dụng trong chương trình đều có tính toàn cục và có thể bị thay đổi vào bất cứ lúc nào. Vào giai đoạn này, người ta xem việc lập trình như một hoạt động nghệ thuật nhuốm màu sắc tài nghệ cá nhân hơn là khoa học, với thuật ngữ "the art of programming".

I.2. Lập trình có cấu trúc

Vào cuối những năm 1960 và đầu 1970, khuynh hướng lập trình cấu trúc (structured programming) ra đời. Theo phương pháp này, một chương trình có cấu trúc được tổ chức theo các phép toán mà nó phải thực hiện. Chương trình bao gồm nhiều thủ tục, hay hàm, riêng rẽ. Các thủ tục hay hàm này độc lập với nhau, có dữ liệu riêng, giải quyết những vấn đề riêng, nhưng có thể trao đổi qua lại với nhau bằng các tham biến.

Lập trình cấu trúc làm cho việc kiểm soát chương trình dễ dàng hơn, và do vậy, giải quyết bài toán dễ dàng hơn. Tính hiệu quả của lập trình cấu trúc thể hiện ở khả năng trừu tượng hoá. Trong một chương trình có cấu trúc, người ta chỉ quan tâm về mặt chức năng : một thủ tục hay hàm nào đó có thực hiện được công việc đã cho hay không ? Còn việc thực hiện như thế nào là không quan trọng, chùng nào còn đủ tin cậy.

Mặc dù kỹ thuật thiết kế và lập trình cấu trúc được sử dụng rộng rãi nhưng vẫn bộc lộ những khiếm khuyết. Khi độ phức tạp tăng lên thì sự phụ thuộc của chương trình vào kiểu dữ liệu mà nó xử lý cũng tăng theo. Cấu trúc dữ liệu trong một chương trình có vai trò quan trọng cũng như các phép toán thực hiện trên chúng. Một khi có sự thay đổi trên một kiểu dữ liệu thì một thủ tục nào đó tác động lên kiểu dữ liệu này cũng phải thay đổi theo.

Khiếm khuyết trên cũng ảnh hưởng đến tính hợp tác giữa các thành viên lập trình. Một chương trình có cấu trúc được giao cho nhiều người thì khi có sự thay đổi về cấu trúc dữ liệu của một người sẽ ảnh hưởng đến công việc của những người khác.

I.3. Lập trình định hướng đối tượng (ĐHĐT)

Lập trình ĐHĐT (oriented-object programming) được xây dựng trên nền tảng của lập trình cấu trúc và trừu tượng hoá dữ liệu (data abstraction).

Chương trình ĐHĐT được thiết kế xung quanh dữ liệu mà nó thao tác chứ không bản thân các thao tác. Tính ĐHĐT làm rõ mối quan hệ giữa dữ liệu và thao tác trên dữ liệu.

Trừu tượng hoá dữ liệu là làm cho việc sử dụng các cấu trúc dữ liệu trở nên độc lập đối với việc cài đặt cụ thể. Ví dụ số dấu chấm động (floating point number) đã được trừu tượng hoá trong mọi ngôn ngữ lập trình. NSD thao tác trên các số dấu chấm động mà không quan tâm đến cách biểu diễn nhị phân trong máy của chúng như thế nào.

Lập trình ĐHĐT liên kết các cấu trúc dữ liệu với các phép toán. Một cấu trúc nào đó thì tương ứng, ta có những phép toán nào đó. Ví dụ : một bản ghi về nhân sự có thể được đọc, cập nhật sự thay đổi và được cất giữ, còn một số phức thì được dùng trong tính toán. Không thể viết số phức lên tệp như một bản ghi nhân sự, cũng không thể cộng trừ nhân chia hai bản ghi nhân sự với nhau như cách của số phức.

Lập trình ĐHĐT đưa vào nhiều thuật ngữ và khái niệm mới, chẳng hạn khái niệm lớp (class), khái niệm kế thừa (inheritence).

Ưu điểm của lập trình ĐHĐT là làm cho việc phát triển phần mềm nhanh chóng hơn với khả năng dùng lại các chương trình cũ. Một lớp mới được xem như lớp suy diễn, có thể được kế thừa cấu trúc dữ liệu và các phương pháp của lớp gốc hoặc lớp cơ sở.

Một trong những ngôn ngữ lập trình ĐHĐT được nói đến là SMALLTALK, được phát triển năm 1980 tại Xerox Palo Alto Recearch Center (PARC). Hiện nay, nhiều ngôn ngữ lập trình thông dụng cũng được trang bị thêm khả năng ĐHĐT, như là C++, Delphi, v.v...

I.4. Lập trình trực quan

Lập trình trực quan (visual programming) được phát triển trên nền tảng của lập trình ĐHĐT. Khi thiết kế chương trình, người lập trình nhìn thấy ngay kết quả qua từng thao tác và giao diện người dùng (user interface) khi chương trình được thực hiện. Người lập trình có thể dễ dàng chỉnh sửa về màu sắc, kích thước, hình dáng và các xử lý thích hợp lên các đối tượng có mặt trong giao diện.

Các ngôn ngữ lập trình trực quan thông dụng hiện nay thường được phát triển trong môi trường Microsoft Windows, như Visual Basic, Visual C++, Visual Foxpro, Java. v.v...

I.5. Những tư tưởng cách mạng trong lập trình

Lập trình là một trong những lĩnh vực khó nhất của toán học ứng dụng. Người ta coi lập trình là một khoa học nhằm đề xuất những nguyên lý và phương pháp để nâng cao năng suất lao động của lập trình viên. Năng suất ở đây được hiểu là tính đúng đắn của chương trình, tính dễ đọc, dễ sửa, tận dụng hết khả năng của thiết bị mà không phụ thuộc vào thiết bị.

Thực chất của quá trình lập trình là người ta không lập trình trên một ngôn ngữ cụ thể mà lập trình hướng tới nó. Chương trình phải được viết dưới dạng các thao tác có cấu trúc trên các đối tượng có cấu trúc và các mệnh đề nhằm khẳng định tính đúng đắn của kết quả.

Những tư tưởng cách mạng trong lập trình thể hiện ở hai điểm sau :

- Chương trình và lập trình viên trở thành đối tượng nghiên cứu của lý thuyết lập trình.
- Làm thế nào để làm chủ được sự phức tạp của hoạt động lập trình ?

II. Các phương diện của công nghệ phần mềm

II.1. Công nghệ phần mềm là gì?

Theo từ điển *Computer Dictionary* của Microsoft Press® (1994), Software Engineering : The design and development of sofware (computer program), from concept through execution and documentation.

Từ điển Larousse (1996) định nghĩa chi tiết hơn : *Công nghệ phần mềm là tập hợp các phương pháp, mô hình, kỹ thuật, công cụ và thủ tục liên quan đến các giai đoạn xây dựng một sản phẩm phần mềm*. Các giai đoạn đó là : *đặc tả* (specifiction), *thiết kế* (design), *lập trình* (programming), *thử nghiệm* (testing), *sửa sai* (debugging), *cài đặt* (setup) để đem vào *ứng dụng* (application), *bảo trì* (maintenance) và *lập hồ sơ* (documentation).

Mục đích chính của công nghệ phần mềm là để sản xuất ra những phần mềm có chất lượng. Chất lượng phần mềm không là một khái niệm đơn giản, bao gồm nhiều yếu tố. Chẳng hạn chương trình chạy nhanh, dễ sử dụng, có tính cấu trúc, dễ đọc dễ hiểu, v.v...

Người ta thường đánh giá theo hai kiểu chất lượng : những yếu tố chất lượng bên ngoài và những yếu tố chất lượng bên trong.

II.2. Những yếu tố chất lượng bên ngoài và bên trong

Những yếu tố chất lượng bên ngoài người dùng có thể nhận biết được, như tốc độ nhanh, chạy ổn định, tính dễ sử dụng, dễ thích nghi với những thay đổi (tính mở rộng), tính công thái học (ergonomy, human factor), v.v...

Những yếu tố chất lượng bên ngoài của một sản phẩm phần mềm là :

Tính đúng đắn	Khả năng thực hiện chính xác công việc đặt ra.
Tính bền vững	Có thể hoạt động trong những điều kiện bất thường.
Tính có thể mở rộng	Khả năng dễ sửa đổi để thích nghi với những thay đổi mới

Tính sử dụng lại	Khả năng sử dụng lại toàn bộ hay một phần của hệ thống cho những ứng dụng mới.
Tính tương thích	Có thể dễ dàng kết hợp với các sản phẩm phần mềm khác.
Các chất lượng khác	Hiệu quả đối với nguồn tài nguyên của MTĐT như bộ xử lý, bộ nhớ..., dễ chuyển đổi (không phụ thuộc vào cấu hình phần cứng), dễ kiểm chứng và an toàn (được bảo vệ quyền truy nhập), dễ sử dụng, v.v...

Những yếu tố chất lượng bên trong là là tính đơn thể, tính dễ đọc, dễ hiểu mà chỉ những người làm Tin học chuyên nghiệp mới biết được. Yếu tố chất lượng bên ngoài là mục đích cuối cùng nhưng yếu tố chất lượng bên trong lại là mấu chốt để đạt được những yếu tố chất lượng bên ngoài.

II.3. Sản phẩm phần mềm là gì ?

Mặc dù người ta không định nghĩa nhưng khái niệm sản phẩm phần mềm được hiểu như là một hệ thống chương trình thực hiện một nhiệm vụ tương đối độc lập nhằm phục vụ cho một ứng dụng cụ thể trong cuộc sống của con người (và có thể được thương mại hoá). Ví dụ các sản phẩm phần mềm :

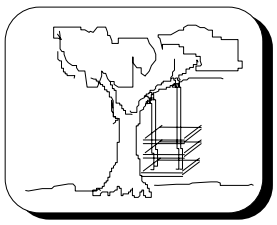
- Hệ điều hành : MS – DOS, OS/2, Unix, MAC OS...
- Hệ điều hành mạng máy tính : Unix, Novell Netware, Windows NT... và các ứng dụng trên mạng LAN, WAN, Internet/Intranet (các Browsers, các dịch vụ khai thác Internet...).
- Các ngôn ngữ lập trình (chương trình dịch) : Turbo Pascal, Turbo C, C^{++}...
- Hệ quản trị cơ sở dữ liệu : Microsoft Foxpro, Microsoft Access, Oracle, Paradox...
- Microsoft Windows và các ứng dụng trên Windows.
- Các trò chơi (games).
- Các phần mềm trợ giúp thiết kế (CAD, Designers...), trợ giúp giảng dạy...
- Các hệ chuyên gia, trí tuệ nhân tạo, người máy, v.v...
- Các chương trình phòng chống virus, v.v...

Dưới đây là bảng tóm tắt quá trình tiến hóa của sản phẩm phần mềm :

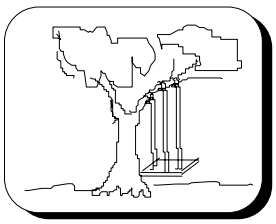
Thời kỳ đầu tiên 1950 – 1960	Xử lý theo lô (Batch processing) Phần mềm được viết theo đơn đặt hàng
Thời kỳ thứ hai 1960 – 1970	Đa người dùng (Multiusers) Thời gian thực (Real time) Cơ sở dữ liệu (Database) Phần mềm sản phẩm

Thời kỳ thứ ba 1970 – 1990	Hệ thống xử lý phân bổ (Distributed processing system) Thông minh (Intelligence) Phần cứng giá thành hạ Hiệu quả tiêu thụ
Thời kỳ thứ tư 1990 trở đi	Hệ thống để bàn (Desktop – Personal – Notebook computers) Lập trình hướng đối tượng (Object oriented programming) Lập trình trực quan (Visual programming) Hệ chuyên gia (Expert system) Mạng thông tin toàn cầu (Worldwide communication network) Xử lý song song (Paralell processing) ...

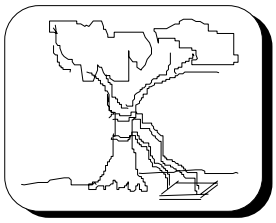
Sau đây là một tranh vui về quá trình tạo ra một sản phẩm phần mềm đã khá quen thuộc đối với những người làm Tin học từ hơn 20 năm nay (theo J. CLAVIER, "Diriger un projet informatique", Edition J. C. I. Inc, Canada 1993) :



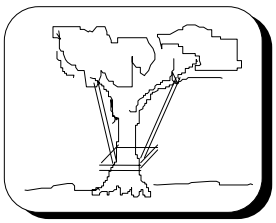
1. Người đặt hàng
Ví dụ : Công ty Công viên



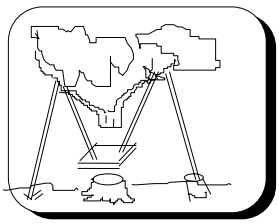
2. Thiết kế của chủ trì đề tài



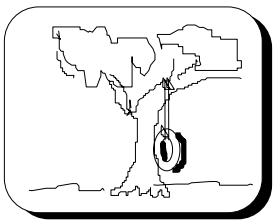
3. Sản phẩm của người lập trình



4. Sau khi sửa sai với nhiều sáng kiến cải tiến



5. Triển khai cho khách hàng



6. Ước mơ của người sử dụng !

Hình 1.1. Quá trình tạo ra một sản phẩm phần mềm

III. Những nội dung cơ bản của CNPM

III.1. Tổng quan về công nghệ phần mềm

Công nghệ phần mềm đặc trưng bởi tập hợp các phương pháp để phát triển một chương trình (phần mềm nói chung). Sự phát triển một chương trình, hay *tiến trình phần mềm* (software process), không chỉ nằm ở chỗ lập trình theo nghĩa hẹp mà còn là việc triển khai các giai đoạn dẫn đến lập trình. Tập hợp các giai đoạn này được gọi là *chu kỳ sống* (hay vòng đời) của phần mềm (life cycle).

Với một dự án Tin học lớn, nhiều người lập trình tham gia được chia thành nhóm, mỗi nhóm phụ trách giải quyết một phần của dự án. Người phụ trách dự án có nhiệm vụ phân bổ công việc cho từng nhóm, đảm bảo mối liên lạc giữa các nhóm, kiểm tra tiến trình phát triển của dự án, chất lượng của sản phẩm phần mềm khi hoàn tất.

Tiến trình phát triển phần mềm gồm 3 giai đoạn chính là *xác định*, *phát triển* và *bảo trì*, không phụ thuộc vào miến áp dụng, độ lớn và độ phức tạp của dự án phát triển, cũng như mô hình được lựa chọn.

Giai đoạn xác định :

Giai đoạn này trả lời câu hỏi *là cái gì* ? (What?) và khi nào (When?) về dữ liệu (thông tin) cần xử lý, mục đích chức năng và môi trường phát triển. Gồm 3 bước :

- Phân tích hệ thống.

- Lập kế hoạch dự án phần mềm.

- Phân tích yêu cầu thực tiễn.

Giai đoạn phát triển :

Giai đoạn này trả lời câu hỏi *làm như thế nào* ? (How?). Gồm 3 bước :

- Thiết kế phần mềm : Sử dụng các công cụ đặc tả và lập trình cấu trúc.

- Chọn công cụ hoặc các ngôn ngữ lập trình để tiến hành viết chương trình.

- Kiểm thử (phát hiện sai sót, nhầm lẫn...).

Giai đoạn bảo trì :

Giai đoạn này tập trung vào các thay đổi (Modify). Có 3 kiểu thay đổi :

- *Sửa đổi* : Dù phần mềm có chất lượng tốt, vẫn tồn tại những khiếm khuyết từ việc sử dụng của khách hàng (người sử dụng). Bảo trì sửa đổi làm thay đổi phần mềm, khắc phục khiếm khuyết.

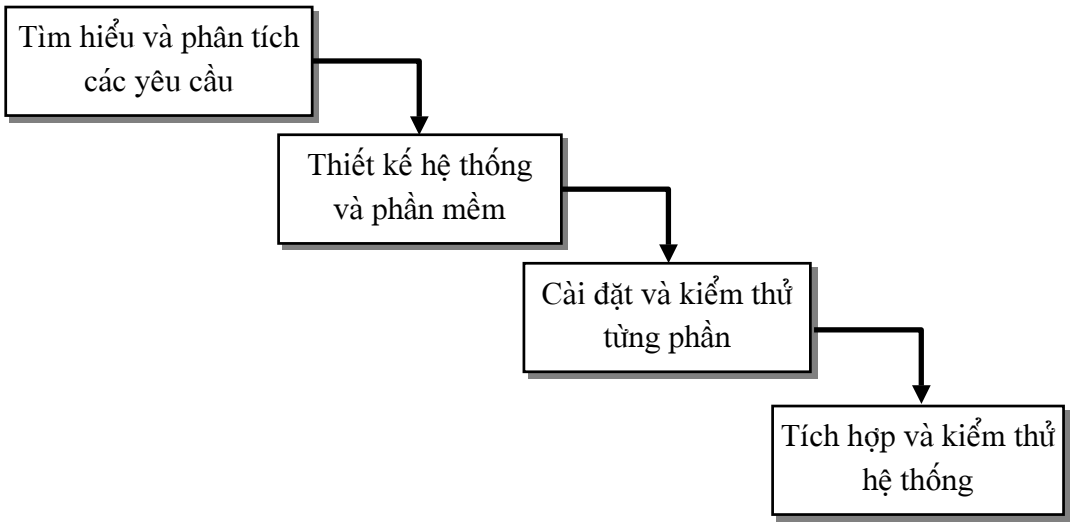
- *Thích nghi* : Nhằm làm phần mềm thích nghi với môi trường phần cứng, như CPU, OS, các thiết bị ngoại vi.

- *Nâng cao* : Khách hàng tìm ra những chức năng phụ của phần mềm. Bảo trì hoàn thiện để mở rộng phần mềm ra ngoài những chức năng vốn có.

III.2. Chu kỳ sống của phần mềm

Có nhiều mô hình khác nhau để thể hiện một chu kỳ sống (life cycle). Sau đây là một chu kỳ sống kiểu cổ điển theo mô hình thác nước ("waterfall" model) gồm các giai đoạn như sau :

- Tìm hiểu và phân tích các yêu cầu (RAD – Requirements analysis and definition)
- Thiết kế hệ thống và phần mềm (SSD – System and software design)
- Cài đặt và kiểm thử từng phần (IUT – Inplementtation and Unit testing)
- Tích hợp và kiểm thử hệ thống (IST – Integrgion and system testing)



Hình 1.2. Mô hình thác nước

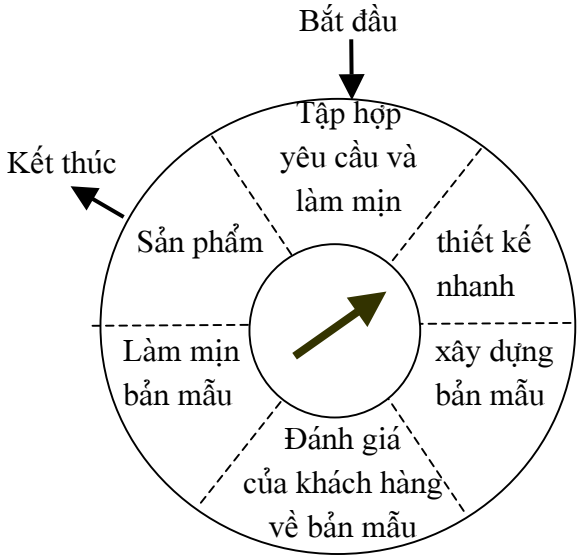
Dẫu rằng mô hình thác nước trên đây có ích lợi trong việc quản lý (management), lập kế hoạch và lập báo cáo tiến độ phát triển phần mềm nhưng chỉ thích hợp với một lớp hệ thống phần mềm nào đó mà thôi, không phù hợp với các hoạt động đã chỉ ra trong mô hình.

Tiến trình phần mềm gồm các hoạt động phức tạp và biến động mà không thể biểu diễn trên một mô hình đơn giản. Những mô hình tốt về tiến trình phần mềm vẫn còn là chứng chủ đề nghiên cứu. Hiện nay, các mô hình tổng quát khác nhau hay tính thực dụng của sự phát triển phần mềm, gắn bó chặt chẽ với nhau.

Mô hình thác nước nguyên thuỷ (original) là một trong những mô hình tổng quát mang tính thực dụng sâu sắc.

Sau đây là một số tiếp cận :

1. *Tiếp cận thác nước* (the waterfall approach) : Bao gồm các giai đoạn đặc tả yêu cầu, thiết kế phần mềm, cài đặt, kiểm thử, v.v..., sau mỗi giai đoạn là sự kết thúc (signed-off) và tiếp tục giai đoạn tiếp theo.
2. *Lập trình thăm dò* (exloratory programming) : Cho phép tăng nhanh quá trình để dẫn đến tính thỏa đáng của hệ thống. Lập trình thăm dò thường được áp dụng trong lĩnh vực trí tuệ nhân tạo, khi NSD không thể định hình được các đặc tả yêu cầu. NSD quan tâm đến tính thỏa đáng của kết quả hơn là tính chính xác.
3. *Bản mẫu* (prototyping) : Tương tự tiếp cận lập trình thăm dò. Pha đầu tiên bao gồm phát triển một chương trình cho phép thử nghiệm. Tuy nhiên, mục đích của phát triển là thiết lập các yêu cầu hệ thống. Sau đó là sự cài đặt lại phần mềm để đưa đến hệ thống chất lượng - sản phẩm.



Hình 1.3. Tiếp cận kiểu bản mẫu

4. *Biến đổi hình thức* (formal transformation) : Là sự biến đổi các đặc tả hình thức (formal specification) của hệ thống phần mềm đang xét để thành một chương trình khả thi nhưng bảo toàn được tính chính xác (correctness - preserving transformations).
5. *Lắp ráp hệ thống từ các thành phần dùng lại được* (system assembly from reusable components). Kỹ thuật này cho phép xây dựng hệ thống từ các thành phần đã có. Tiến trình phát triển hệ thống là sự lắp ráp hơn là sự sáng tạo.

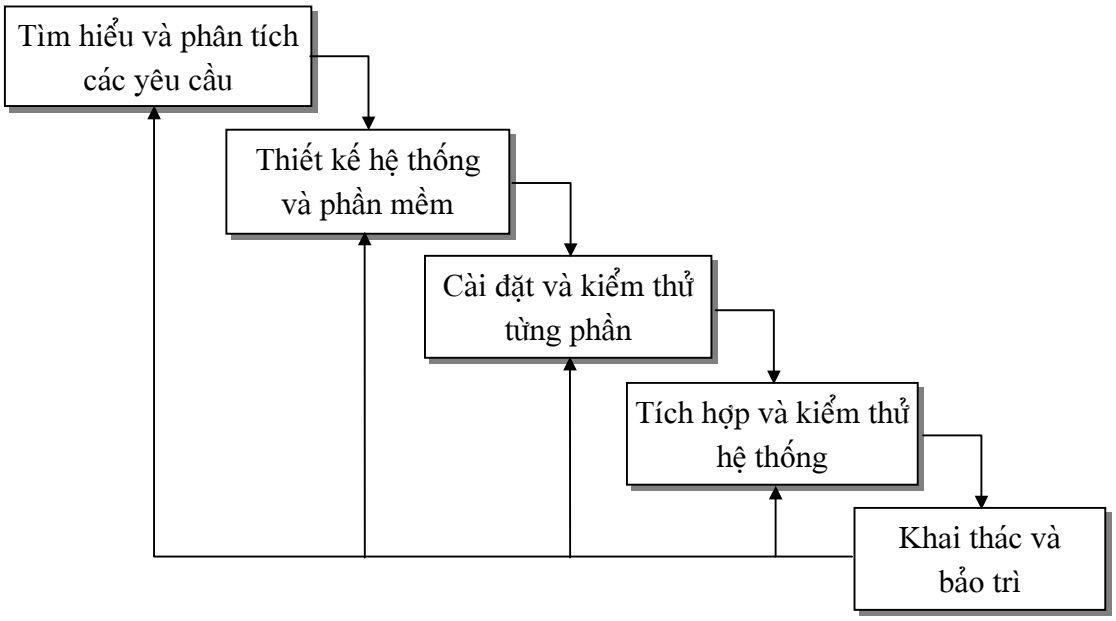
Hiện nay, các tiếp cận 1, 2, 3 được ứng dụng nhiều trong thực tiễn.

Trên thực tế, các giai đoạn phát triển phần mềm không phải rời riêng mà là gối lên nhau (overlap) và thông tin được cung cấp lẫn nhau.

Trong khi thiết kế, những vấn đề và các yêu cầu gắn bó với nhau, trong khi lập trình, những vấn đề thiết kế được tìm thấy, v.v... Lúc này, tiến trình phần mềm không đơn giản là một mô hình tuyến tính mà bao gồm một dãy các tương tác của các hoạt động phát triển.

Tuy nhiên, một mô hình chứa các vòng lặp sẽ làm khó khăn cho việc quản lý và báo cáo. Có nhiều dạng mô hình trong tiến trình phần mềm. Sau đây là một số mô hình :

1. Mô hình thác nước cải tiến

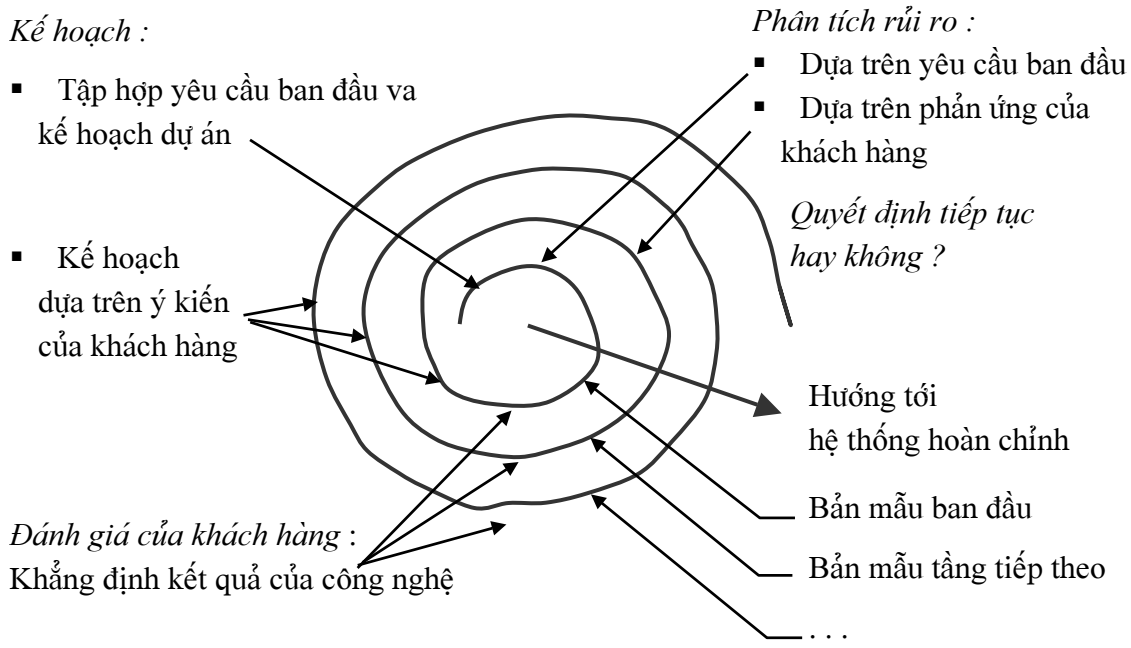


Hình 1.4. Mô hình thác nước cải tiến

1. *Tìm hiểu và phân tích các yêu cầu*: NSD hệ thống và người phát triển hệ thống bàn bạc, trao đổi (consultation) với nhau để thiết lập mục đích, ràng buộc và các dịch vu của hệ thống phần mềm, lĩnh hội được những đòi hỏi của bài toán.
2. *Thiết kế hệ thống và phần mềm* : Tiến trình thiết kế hệ thống phân chia các yêu cầu thành các hệ thống phần cứng, phần mềm và thiết lập một kiến trúc hệ thống toàn bộ (overall system architecture). Việc thiết kế phần mềm bao gồm việc thể hiện các chức năng hệ thống phần mềm (software system functions) để biến đổi thành các chương trình khả thi.
3. *Cài đặt và kiểm thử từng phần* : Trong giai đoạn này, các đơn vị chương trình hay tập hợp các chương trình được kiểm thử lần lượt sao cho thỏa mãn các đặc tả tương ứng.
4. *Tích hợp và kiểm thử hệ thống* : Các đơn vị chương trình được tích hợp và kiểm thử như là một hệ thống đầy đủ để đảm bảo các yêu cầu đặt ra ban đầu. Sau giai đoạn này, hệ thống phần mềm được giao cho khách hàng.
5. *Khai thác và bảo trì* (operation and maintenance) : Đây là một pha dài nhất của chu kỳ sống. Hệ thống được cài đặt và đưa vào sử dụng thực tế. Việc bảo trì bao gồm việc khắc phục những sai sót xảy ra đã không xuất hiện trong các giao đoạn trước đó của chu kỳ sống. Việc tối ưu hóa các dịch vụ của hệ thống được xem như là những yêu cầu mới được phát hiện.

2. Mô hình xoắn ốc

Phát triển trên tính ưu việt của vòng đời cổ điên và bản mẫu, bổ sung nhũng yếu tố còn thiếu và thêm các yếu tố mới, phân tích rủi ro.



Hình 1.5. Mô hình xoắn ốc

Ưu điểm :

Các phiên bản (hay sản phẩm) được hoàn thiện dần theo chiều xoáy ốc từ trong ra ngoài.

Nhược điểm :

- Khó đánh giá chính xác, nhất là khi gặp rủi ro, khó kiểm soát. Do đó khó thuyết phục được các khách hàng lớn
- Mô hình nàuy còn mới, chưa được kiểm nghiệm nhiều trong thực tiễn.

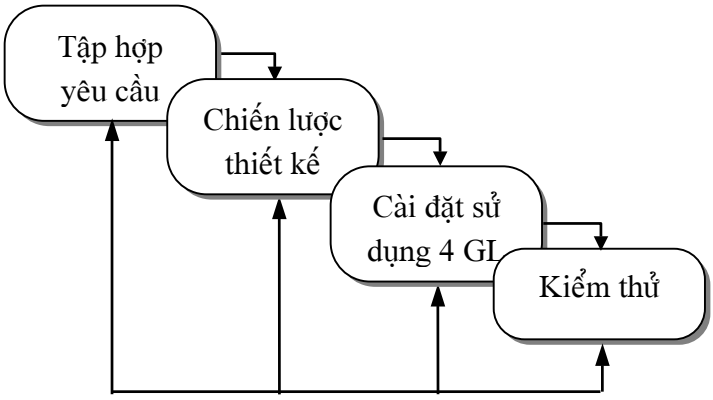
3. Kỹ thuật thế hệ 4 (4th Generation Technology)

Bao gồm các công cụ phần mềm trên cơ sở tự động sản sinh mã chương trình gốc theo nhu cầu của người phát triển :

- Ngôn ngữ phi thủ tục[2] (non procedural language) để truy cập cơ sở dữ liệu.
- Bộ sinh báo cáo.
- Bộ thao tác dữ liệu.

[2] là ngôn ngữ lập trình không tuân theo cách gọi thủ tục hay gọi chương trình con thông thường, không sử dụng các cấu trúc điều khiển, tuần tự, mà dựa trên tập hợp các yếu tố và quan hệ để dẫn về kết quả yêu cầu. Ví dụ ngôn ngữ vấn tin SQL thuộc loại này.

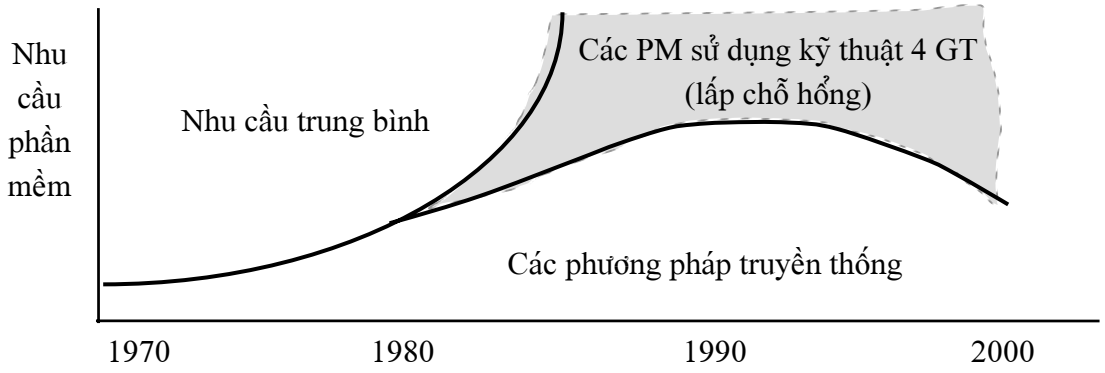
- Bộ tương tác và thiết kế màn hình.
- Bộ sinh chương trình.
- Bảng tính.
- Công cụ đồ họa.



Hình 1.6. Kỹ thuật thế hệ 4

Ưu điểm :

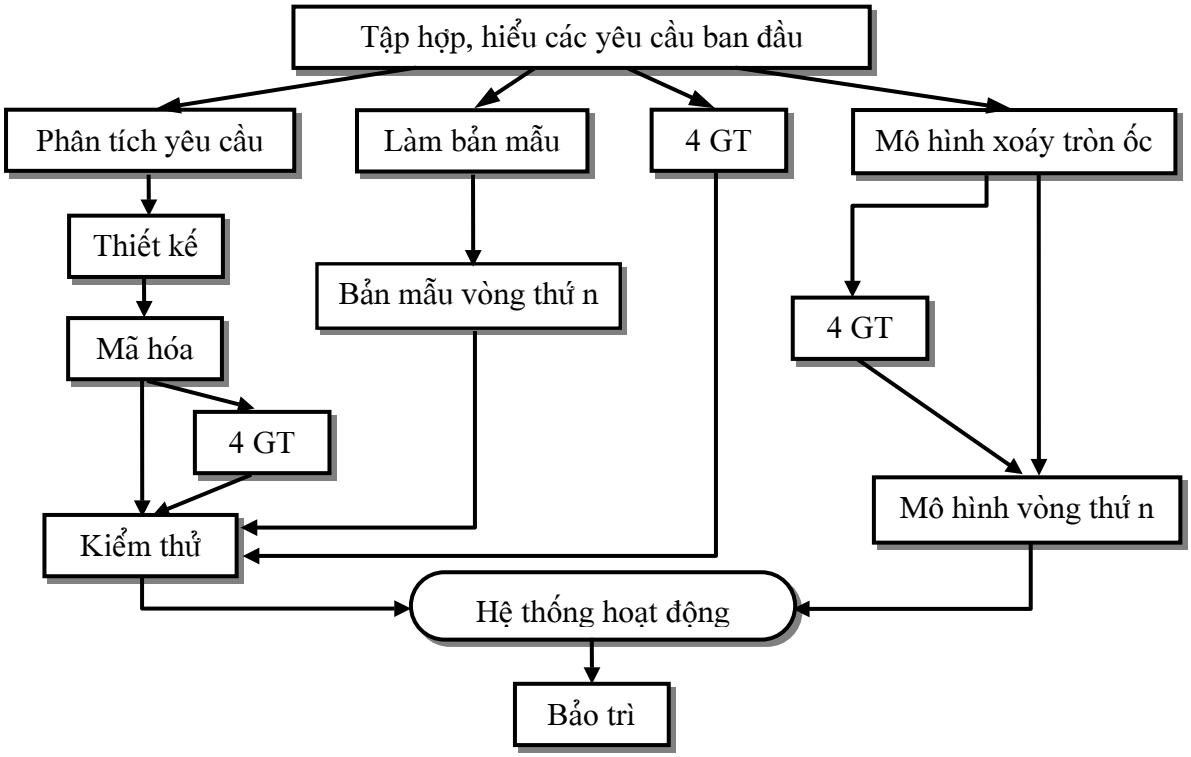
Thường được sử dụng để xây dựng các hệ thông tin và tương lai là các ứng dụng kỹ nghệ phát triển phần mềm thời gian thực.



Hình 1.7. Nhu cầu phần mềm

5. Tích hợp các kỹ thuật

Nhằm tăng cường tính tối ưu trong phát triển phần mềm, người ta có xu hướng tích hợp các kỹ thuật cổ điển, xoáy tròn ốc và 46T đã nêu.



Hình 1.8. Tích hợp các kỹ thuật

CHƯƠNG 2

Thiết kế phần mềm

I. Nền tảng của thiết kế phần mềm

Phương pháp lập trình cấu trúc 19

II. Phương pháp lập trình cấu trúc

Phương pháp lập trình cấu trúc 21

II.1. Khái niệm về lập trình cấu trúc

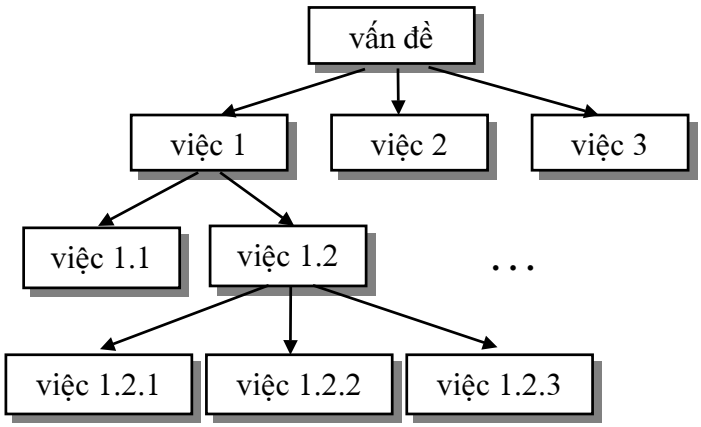
Lập trình cấu trúc (Structured Programming) là trường phái lập trình xuất hiện vào những năm 70 và được duy trì phát triển từ đó đến nay. Lập trình cấu trúc phản ánh quan niệm : lập trình là công việc sáng tạo nhưng có tính khoa học và có phương pháp, không phải là ngẫu hứng cá nhân.

Tính logic và trong sáng của chương trình đảm bảo độ tin cậy, dễ hiểu, dễ sửa và dễ thừa kế chương trình.

II.2. Những ý tưởng cơ bản lập trình cấu trúc

a) Chương trình là một hệ thống phân cấp từ trên xuống

Trong lập trình cấu trúc, chương trình là một hệ thống phân cấp từ trên xuống, trong đó các thành phần tương tác với nhau tối thiểu. Vấn đề cần giải quyết bao gồm các vấn đề nhỏ hơn, mỗi vấn đề đó lại bao gồm các vấn đề nhỏ hơn nữa, v.v... cho đến mức cuối cùng là những công việc đơn giản và dễ giải quyết hoặc đã giải quyết rồi.



Hình 2.1. Chương trình là một hệ thống phân cấp

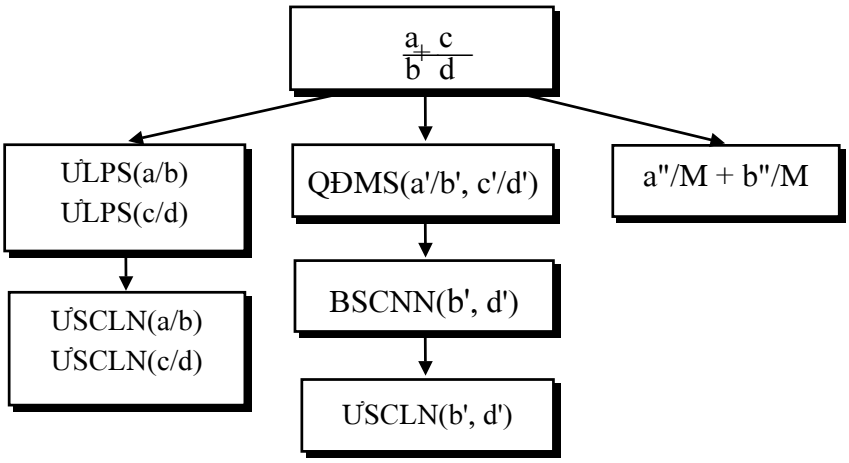
Ví dụ 2 :

Phân tích bài toán cộng hai phân số để đưa về bài toán tìm ước số chung lớn nhất.

Để cộng hai phân số, trước tiên cần ước lược chúng, tiếp đó quy đồng mẫu số để lấy mẫu số chung. Cuối cùng tiến hành cộng hai tử số của hai phân số đã có chung mẫu số. Việc ước lược phân số được đưa về tìm ước số chung lớn nhất của tử số và mẫu số (sử dụng thuật toán Euclide).

Để quy đồng mẫu số, cần tìm bội số chung nhỏ nhất. Việc tìm bội số chung nhỏ nhất của hai số lại được đưa về tìm ước số chung lớn nhất của chúng :

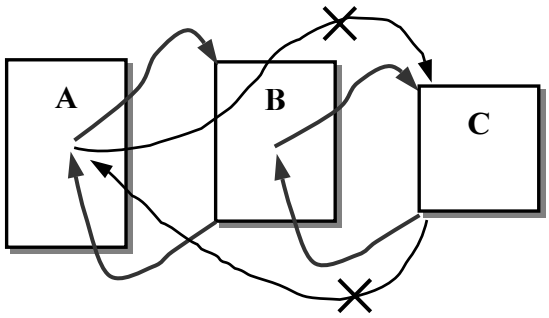
BSCNN(b', d') = b' * d' / ƯSCLN(b',d').



Hình 2.2. Phân tích bài toán cộng hai phân số

Như vậy, chương trình là một hệ thống gồm nhiều thành phần phân cấp, mỗi thành phần có nhiệm vụ giải quyết một vấn đề sơ cấp và có tính độc lập cao. Các thành phần nên tương tác với nhau tối thiểu. Giữa hai thành phần trong hệ thống chỉ nên có tối đa một đường tương tác là đường trao đổi thông tin để dễ quản lý và dễ kiểm soát.

- Giữa chương trình chính và chương trình con có đường giao tiếp là việc truyền tham biến. Không được gọi chương trình con theo kiểu vượt cấp.



Hình 2.3. A gọi B, B gọi C, nhưng A không gọi được C

- Hạn chế dùng biến toàn cục (global variables) trong chương trình con vì sẽ tạo thêm những đường giao tiếp khó quản lý. Chẳng hạn, một chương trình con nào đó làm thay đổi một biến toàn cục thì ở một nơi khác, trong một chương trình con khác hoặc ngay trong chương trình chính, cũng sẽ khó nhận biết sự thay đổi này.

b) Không sử dụng lệnh nhảy goto

Lệnh goto (jump statement) dùng để chuyển điều khiển đến một điểm khác trong chương trình. Lệnh goto làm khó quản lý và khó kiểm soát chương trình nên khó đọc, khó sửa sai (rối rắm như món mì sợi Spaghetti của Ý).

Các chương trình viết trên ngôn ngữ aassembly hoặc trên các ngôn ngữ bậc cao như Fortran, Algol, Cobol... thường sử dụng lệnh goto.

Ví dụ 3 :

Chương trình Algol sau đây sử dụng lệnh goto để điều khiển vòng lặp tính tổng các phần tử của mảng a gồm N số thực :

```
S := 0; I := 0;
Start : S := S + a[ I]; I := I + 1;
     if I <= N then goto Start;
...
```

c) Chương trình có tính cấu trúc

Chương trình chỉ sử dụng các cấu trúc điều kiện chuẩn, dễ hiểu, dễ thể hiện thuật toán. Cấu trúc của chương trình phản ánh được cấu trúc của vấn đề và cách giải quyết vấn đề (làm như thế nào ?). Phương pháp hay được sử dụng để thiết kế chương trình là *phân tích từ trên xuống* (Top-Down Analysis) và *tổng hợp từ dưới lên* (Bottom up Synthesis).

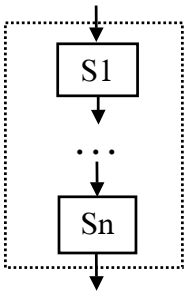
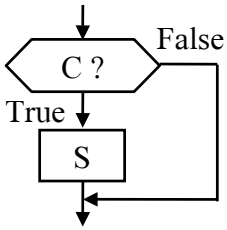
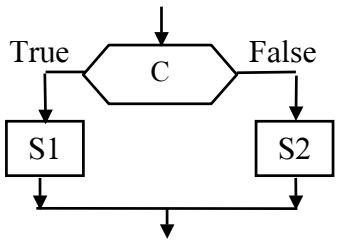
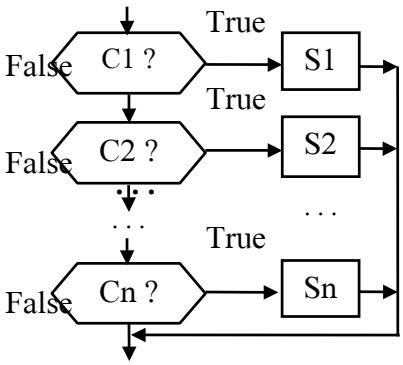
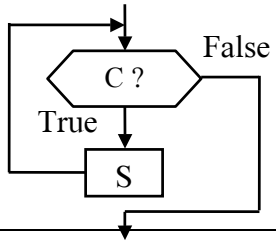
Nội dung phương pháp phân tích từ trên xuống là nhìn nhận xem xét tổng quát toàn bộ vấn đề, xuất phát từ mục tiêu (đỉnh) đi xuống các thành phần trong hệ thống, chia các thành phần thành các thành phần nhỏ hơn theo một cấu trúc phân cấp chặt chẽ.

Nội dung phương pháp tổng hợp từ dưới lên là xuất phát từ các vấn đề cụ thể và cách giải quyết cụ thể, sau đó tích hợp chúng lại thành vấn đề lớn hơn và cách giải tổng quát hơn, hướng từ dưới lên trên để nhận được vấn đề cần phải giải quyết ban đầu

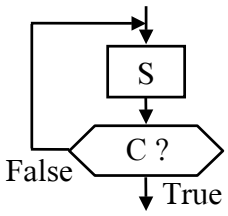
Cách thiết kế này gây khó khăn vì khó kiểm soát và dễ lạc hướng, khó đáp ứng đầy đủ các yêu cầu của vấn đề.

II.3. Các cấu trúc điều khiển chuẩn

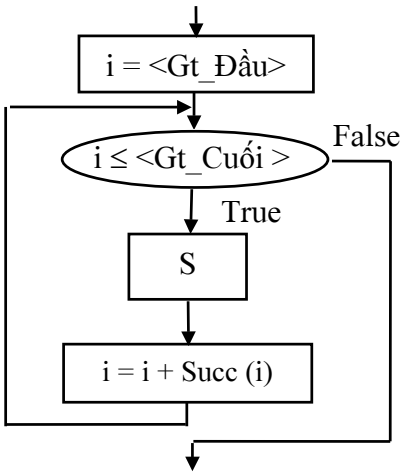
Trong chương trình, chỉ nên sử dụng 7 cấu trúc điều khiển sau đây với quy ước S là một *lệnh* (Statement) và C là một *biểu thức điều kiện* (Condition) nào đó :

Stt	*Cấu trúc điều khiển*	*Lưu đồ tương đương*	*Mô tả*
1	**Tuần tự** (Sequential) **begin** S1 . . . Sn **end**		Được coi như là một lệnh ghép (khối), thực hiện tuần tự các lệnh S1, S2, ..., Sn.
2	**Rẽ nhánh** (Branching) *a) Rẽ nhánh thiếu* **if** C **then** S		Nếu C thoả mãn (True) thì thực hiện S. Nếu C không thoả mãn (False) thì không làm gì cả.
3	*b) Rẽ nhánh đủ* **if** C **then** S1 **else** S2		Nếu C đúng thì thực hiện lệnh S1. Nếu C sai thì thực hiện S2.
4	**Lựa chọn** (Selection) **case** C1 : S1 C2 : S2 . . . Cn : Sn **endcase**		Nếu C1 đúng thì thực hiện S1. Nếu không, nếu C2 đúng thì thực hiện S2, v.v... Cuối cùng, nếu Cn đúng thì thực hiện Sn. Nếu không thì thôi.
5	Cấu trúc lặp (Iteration) Kiểm tra điều kiện trước khi thực hiện vòng lặp :		Khi C còn đúng thì còn thực hiện S. Khi C sai thì dừng.

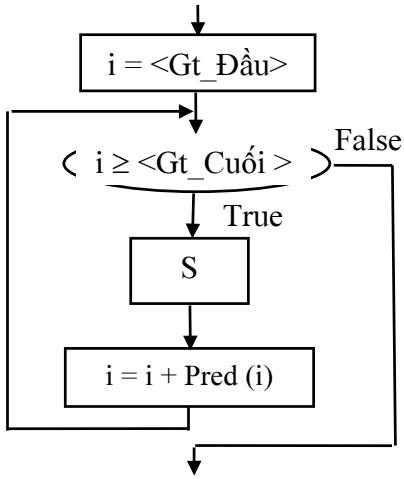
	while C **do** S		

Stt	*Cấu trúc điều khiển*	*Lưu đồ tương đương*	*Mô tả*
6	Lặp với kiểm tra điều kiện sau khi thực hiện xong thân vòng lặp : **do** S **until** C		Còn thực hiện S khi C còn chưa thoả mãn (sai). Ít nhất lặp được một lần. Dừng khi C đúng.

7 Lặp hết trước số lần (for)

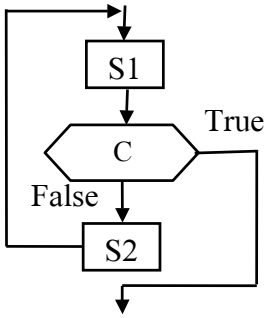


for I ← Gt_Đầu **to** Gt_Cuối **do** S



for I ← Gt_Đầu **downto** Gt_Cuối **do** S

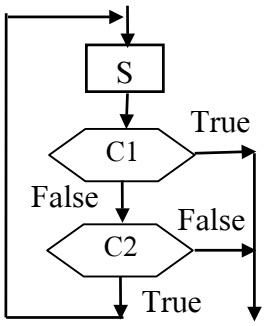
Ngoài các cấu trúc lặp hay gặp thông thường trên đây, người ta còn sử dụng các cấu trúc lặp có thoát (loop exit) như sau :




```
{ sử dụng khoá key để đánh dấu lối thoát,
  key không xuất hiện trong S và trong C }
key := False
While not key do begin
    S1
    if C then key := True else S2
End
```

II.4. Một số ví dụ viết chương trình theo sơ đồ khối

Ví dụ 4 :

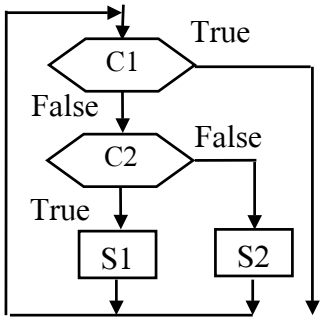


{ Vòng lặp này dùng **Repeat** }

Repeat
S
Until C1 **or not** C2

Chú ý :Có điều kiện cuối vòng lặp có thể dùng **Repeat**

Ví dụ 5 :

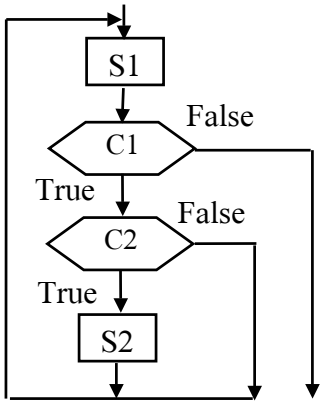


{ Vòng lặp này dùng **While** }

While C1 do
if C2 **then** S1 **else** S2

Chú ý :Có điều kiện trước vòng lặp có thể dùng **While**

Ví dụ 6 :



{ Đây là cấu trúc loop-exit, có thể dùng **While** như sau :}

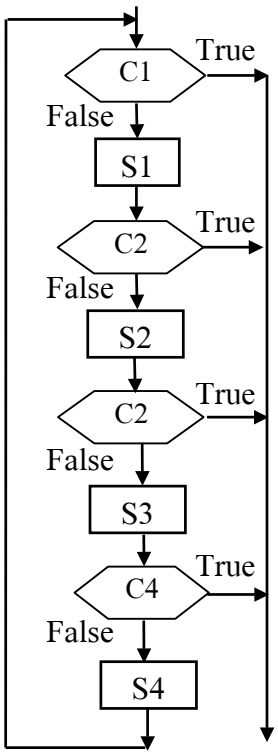
key **:= False**
While not key **do begin**
S1
if not C1 **then** key **:= True**
else if C2 **then** S2

End

Cấu trúc loop-exit trên đây có thể dùng **Repeat** như sau :

key **:= False**
Repeat
S1
if C1 **then** key **:= True**
else if C2 **then** S2
Until key

Ví dụ 7 :




```
{ Trường hợp Loop-exit mở rộng.
  Dùng khóa key để đánh dấu lối thoát như
  sau :}
key := False
Repeat
    If C1 then key := True
    Else begin
        S1
        If C2 then key := True
        Else begin
            S2
            If C3 then key := True
            Else begin
                S3
                If C4 then key := True
                Else S4
            End
        End
    End
Until key
```

III. Cấu trúc tối thiểu

Các cấu trúc điều khiển chuẩn là kết quả của những cố gắng lớn trong cuộc cách mạng về lập trình những năm 60. Những nhà tin học có tên tuổi đã đóng góp công sức là Bohm C. và Jacopini G., Dijkstra E.W. và Warier, v.v...

Để đảm bảo tính trong sáng, đơn giản và tự nhiên, người ta khuyên rằng chỉ nên xây dựng chương trình với 3 cấu trúc điều khiển cơ bản là *tuần tự, rẽ nhánh* và *lặp*.

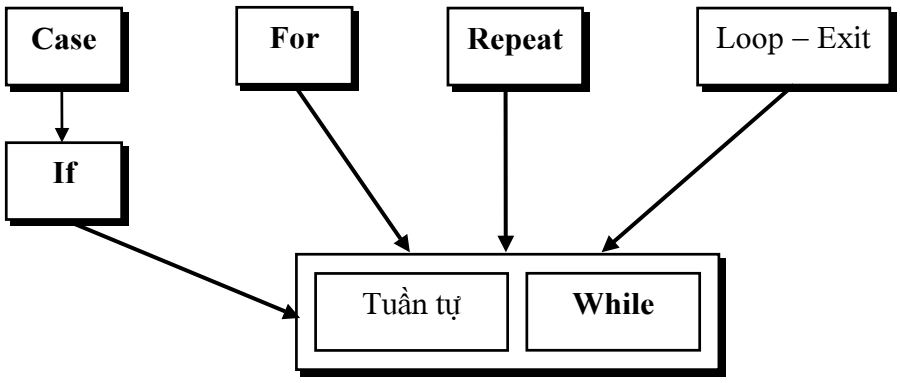
Tuy nhiên, Bohm và Jacopini đã chứng minh được rằng chỉ cần tối thiểu hai cấu trúc tuần tự và lặp là đủ.

Định lý Bohm và Jacopini 1986

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

- *Với mọi dữ liệu vào X thuộc miền xác định X, ta có P(x) = Q(x) : P và Q biến đổi những cái vào giống nhau thành các ra giống 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 chứa thêm một số biến logic.*
- *Q sử dụng hai cấu trúc điều khiển duy nhất là **tuần tự** và vòng lặp* **while** (SW: Sequence &While).

Sơ đồ chuyển cấu trúc như sau :



*Hình 2.4. Chuyển về cấu trúc **tuần tự** và lặp **while*** (SW)

Sau đây là cách chuyển đổi của các cấu trúc **Case**, **If**, **For**, **Repeat** và **Loop** – Exit :

1. **Case → if**

```
Case C of
    C1 : S1
     . . .
    Cn : Sn
End {Case}
```

→

```
If C = C1 then S1
else if C = C1 then S2
    else if . . .
    . . .
    else if C = Cn then Sn
```

2. **if → SW**

Dùng hai biến phụ kiểu logic để thực hiện vòng lặp **While** đúng một lần :

Var p, q : **Boolean**

```
If C then S
```

→

```
p := C
While p do begin
    S ; p := not p
End
```

```
If C then S1 else S2
```

→

```
p := C ; q := p
While p do begin
    S1 ; p := not p
```

		End; **While not** q **do begin** S2 **;** q **:= not** q **End**

3. **Repeat → SW**

Repeat S **until** C	→	S **; While not** C **do** S

4. **For → SW**

For I**:=** GtĐầu **to** GtCuối
do S
	→	I **:=** GtĐầu
		While i <= GtCuối **do**
		begin
		S **;** I **:=** Succ (I)
		End

For I**:=** GtĐầu **downto**
GtCuối
do S
	→	I **:=** GtĐầu
		While i >= GtCuối **do**
		begin
		S **;** I **:=** Pred (I)
		End

III.1. Các cấu trúc lồng nhau

Bản thân lệnh S trong mỗi cấu trúc điều khiển cơ bản lại có thể là một cấu trúc điều khiển khác.

Ví dụ 8 ***: Với cấu trúc điều kiện***

If C **then** S1 **else** S2

Tại S1 và S2, ta có thể đặt các cấu trúc điều khiển khác, chẳng hạn thế S1 bởi :

While C1 **do** S3

và thế S2 bởi :

Repeat S4 **until** C2

Ta có :

If C **then**

While C1 **do** S3

Else Repeat S4 **until** C2

Đến lượt S3 và S4 lại có thể thay thế bởi các cấu trúc khác, v.v...

Với cácc phép thế như vậy, cấu trúc của chương trình ngày càng phức tạp và dẫn đến khó hiểu và dễ sai sót. Chính vì vậy mà người ta chú trọng triển khai chương trình từ trên xuống và viết các cấu trúc theo từng khối.

Các khối có thể thụt vòa, thụt ra để phản ánh tính cấu trúc và mức độ lồng nhau của các cấu trúc.

Nguyên tắc : *Cấu trúc con được viết lọt vào trong (thụt vào) cấu trúc cha. Điểm vào và điểm ra của mỗi cấu trúc phải nằm trên cùng một hàng dọc.*

IV.Lập trình đơn thể

IV.1. Khái niệm về đơn thể

Ý tưởng cơ bản của lập trình cấu trúc là phân rã vấn đề lớn thành các vấn đề nhỏ hơn cho đến khi nhận được các vấn đề tương đối đơn giản, mỗi vấn đề này được giải quyết bởi một đơn thể chương trình (module). Mỗi đơn thể có các tính chất như sau :

a) Tính đơn thuần

- Chỉ giải quyết những đối tượng dữ liệu có liên hệ với nhau trong phạm vi của vấn đề.
- Có một lối vào và một lối ra, bên trong chỉ dùng những cấu trúc điều khiển chuẩn.
- Hoạt động chỉ phụ thuộc vào dữ liệu đưa vào chứ không phụ thuộc vào tình trạng trước đó của nó. Mỗi đơn thể là một hàm dữ liệu vào, kết quả tiên đoán được.

b) Tính chuyên biệt

- Chỉ thực hiện một chức năng, nhiệm vụ nhất định.
- Không quá dài hoặc quá ngắn (lý tưởng là mỗi đơn thể có từ 60 đến 70 dòng lệnh vừa nằm trọn trong một trang A4).
- Chỉ được khởi động bằng cách gọi.

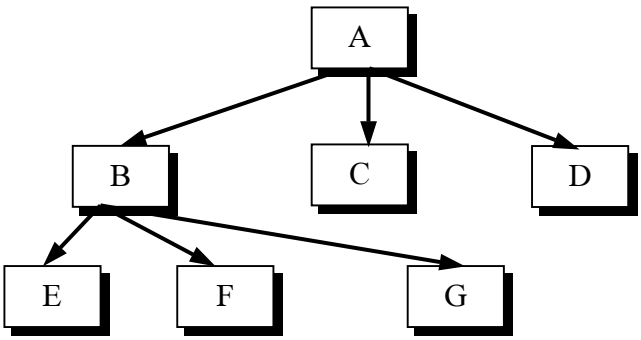
c) Tính độc lập

- Là một đơn vị biên dịch. Có thể viết và chạy thử độc lập.

Các ngôn ngữ lập trình bậc cao như Pascal (kỹ thuật dùng Unit), C, C++ (include các tệp chương trình) và hầu hệ các công cụ lập trình thường gặp hiện nay đều cho phép lập trình theo đơn thể.

IV.2. Mối liên hệ giữa các đơn thể

Các đơn thể nối kết với nhau thành chương trình, tổ chức phân cấp dạng cây (tree).



Hình 2.5. Mối liên hệ giữa các đơn thể

IV.2.1.Phân loại đơn thể

Có 4 loại đơn thể :

a) Đơn thể điều khiển

Đơn thể điều khiển (Director Module) có chức năng gọi các đơn thể khác xử lý.

b) Đơn thể xử lý

Đơn thể xử lý (Pcocessing Module) chuyên trách một nhiệm vụ nào đó trên vùng dữ liệu độc lập. Đơn thể xử lý được đơn thể điều khiển gọi tới và sau khi thực hiện xong chức năng, đơn thể xử lý trả quyền điều khiển trở lại cho đơn thể điều khiển.

c) Đơn thể vào/ra

Đơn thể vào/ra (IO Module) chuyên trách vào/ra dữ liệu, có sự kiểm tra và xử lý sai sót. Đơn thể cũng do đơn thể điều khiển gọi tới giống như hoạt động của đơn thể xử lý.

d) Đơn thể chương trình con

(Subroutine Module) nhằm giải quyết một nhiệm vụ trọn vẹn nhưng có quan hệ với các đơn thể khác. Đơn thể chương trình con được gọi thực hiện nhiều lần trong chương trình.

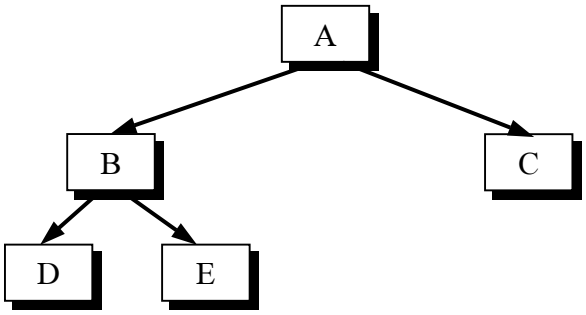
IV.2.2. Tổ chức một chương trình có cấu trúc đơn thể

Cấu trúc chương trình gồm : - Cấu trúc nội tại của các đơn thể.
- Mối liên hệ giữa các đơn thể.

Các đơn thể được tổ chức phân cấp dạng cây nhưng phải thỏa mãn tính *cục bộ tham chiếu* (locality of Reference) : chỉ có đơn thể mức cao hơn (cha) mới có quyền tham chiếu (gọi) đến đơn thể mức thấp hơn kề đó (con).

Những cấu trúc cây thỏa mãn tính cục bộ tham chiếu được gọi là *cấu trúc cây thuần túy* (Pure tree Structure).

Ví dụ 9 :



– D chỉ phục vụ B, chỉ có B mới có quyền điều khiển D, C không thể gọi D.

– Giữa B và D chỉ có một đường tương tác duy nhất là trao đổi tham biến.

Hình 2.6. Cấu trúc cây thuần túy

a) Đặc điểm của cấu trúc cây thuần túy

– Mỗi đơn thể chỉ được quyền điều khiển đơn thể con trực tiếp, giảm được tính phức tạp của chương trình.

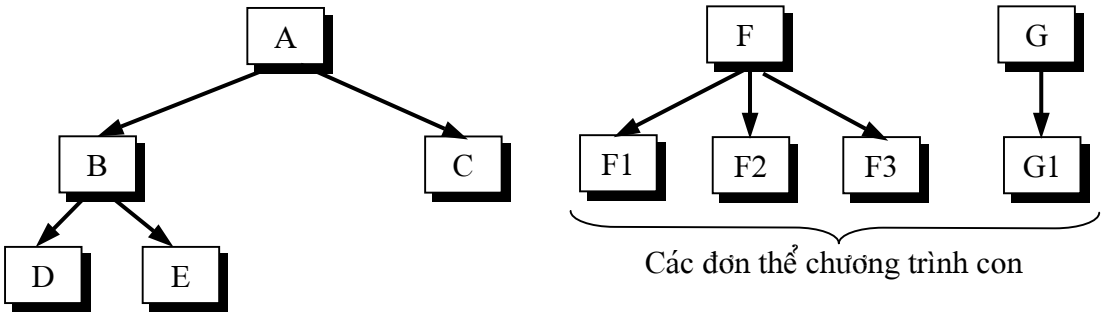
– Các nhánh hoàn toàn tách biệt nhau nên có tính tương tác tối thiểu.

Ngoại lệ : Nếu một công việc nào đó cần thực hiện nhiều lần, nhiều chỗ trong chương trình thì nên tổ chức thành một đơn thể chương trình con và vẽ riêng, không vẽ vào cấu trúc cây.

Như vậy, các đơn thể chương trình con chỉ đóng vai trò thư viện, một sự mở rộng của ngôn ngữ lập trình. Trường hợp đơn thể chương trình con phức tạp thì có thể tổ chức theo cấu trúc cây thuần túy.

Như vậy toàn bộ chương trình là một tập hợp các cấu trúc cây thuần túy.

Ví dụ 10 :



Hình 2.7. Cấu trúc cây thuần túy của chương trình và chương trình con

b) Thử nghiệm chương trình trên cấu trúc cây thuần túy

Quá trình thử nghiệm một chương trình :

– Thử nghiệm các đơn thể chương trình con trước.

– Thử nghiệm các đơn thể trong chương trình chính, từ dưới lên và riêng từng nhánh.

Cần phân biệt :

– Các đơn thể xử lý phụ thuộc vào ngữ cảnh (context) nào, là con của đơn thể nào ?

– Các đơn thể chương trình con độc lập với ngữ cảnh.

Để thử nghiệm chương trình cho trong hình vẽ trong ví dụ ở trên :

– Thử F và G trước (sau khi đã thử F1, F2, F3 và G1).

– Thử D và E rồi thử B.

– Thử C.

– Thử cả chương trình.

V. Phát triển chương trình bằng tinh chế từng bước

V.1. Nội dung phương pháp

Nguyên lý phát triển CHTR bằng tinh chế từng bước (hay thiết kế từ trên xuống) do Niclaus Wirth (tác giả của ngôn ngữ lập trình Pascal) đề xuất vào năm 1971, trong bài báo của mình "Program Development by Stepwise Refinement".

Ban đầu, CHTR là những câu được viết bằng ngôn ngữ tự nhiên (chẳng hạn tiếng Việt) thể hiện sự phân tích tổng thể của người lập trình.

Sau đó, tại mỗi bước, mỗi câu được phân tích chi tiết hơn thành những câu khác. Có nghĩa đã phân tích một công việc thành những công việc bé hơn.

- Mỗi câu được gọi là một đặc tả (Specification).

- Mỗi bước phân tích được gọi là đã tinh chế (refine) câu (công việc) đó.

Sự tinh chế được hướng về phía ngôn ngữ lập trình sẽ dùng. Nghĩa là càng ở bước sau, những câu chữ trên ngôn ngữ tự nhiên càng đơn giản dễ hiểu hơn và được thay thế bằng các câu lệnh của ngôn ngữ lập trình. Nếu câu còn tỏ ra phức tạp, có thể coi đó là một CHTR con và tiếp tục tinh chế nó.

Trong quá trình tinh chế, cần đưa ra các cấu trúc dữ liệu tương ứng với từng bước. Như vậy sự tinh chế các đặc tả CHTR và dữ liệu là song song.

Phương pháp tinh chế từng bước thể hiện tư duy giải quyết vấn đề từ trên xuống, trong đó sự phát triển của các bước là hướng về ngôn ngữ lập trình sẽ sử dụng. Đáy của sự đi xuống trong hoạt động phân tích là các câu lệnh và các mô tả dữ liệu viết bằng ngôn ngữ lập trình.

Ý nghĩa : Việc lập trình có sự định hướng và có sự ngăn nắp trên giấy nháp, tránh mò mẫm thử nghiệm mang tính trực giác.

V.2. Ví dụ minh hoạ

V.2.1. Ví dụ 1

Nhập vào dãy các ký hiệu liên tiếp từ bàn phím cho đến khi kí tự dấu chấm (.) được gõ. In ra số lượng từng chữ số từ 0..9 đã đọc.

Chẳng hạn, nếu nhập vào dãy :

Kiki1t2047655kp412.

thì in ra :

số chữ số 0 đã đọc = 1,
số chữ số 1 đã đọc = 2,
số chữ số 2 đã đọc = 2
...

1. Phác thảo lời giải

Cần in ra 10 giá trị ứng với các chữ số từ 0..9. Có thể dùng 10 biến đơn ZERO, MOT, HAI, BA... nhưng tốt nhất nên dùng một mảng có 10 phần tử :

Số ['0'] chứa kí tự '0' đã đọc;
Số ['1'] chứa kí tự '1' đã đọc;
v.v**...**

Ta mô tả như sau :

```
Type dãy = array [ '0'..'9'] of integer;
var số = dãy;
     c: Char; { ký tự được đọc }
```

Từ đó lời giải có thể được viết như sau :

```
Repeat
   đọc_một_kí_tự; { là ký tự c }
   if kí_tự_là_chữ_số then đếm_chữ_số_đó;
   { ví dụ, nếu đọc '2' thì tăng số ['2'] lên 1}
Until c = dấu chấm;
```

```
for c := '0' to '9' do
  writeln('số các chữ số',c,'đã đọc =',số [ c] :2);
```

Ta tinh chế bước *kí_tự_là_chữ_số* bằng cách chuyển ra dạng biểu thức Pascal như sau :

```
('0' < c) and (c < = '9')
```

Việc *đọc_một_kí_tự* được viết như sau : `Read (c);`

Dấu chấm có thể dùng hằng :

```
Const dấu chấm '=';
```

Ta thấy trước lúc đếm, các phần tử của mảng số phải bằng 0. Ta có :

```
for c := '0' to '9' do số [ c] := 0;
```

Bây giờ ta có chương trình hoàn chỉnh như sau :

```
Program Đếm chữ số;
Const dấu chấm = '.';
Type dãy = array [ '0'..'9'] of integer;
Var  số: dãy;
     c: char;
begin
   for c := '0' to '9' do số [ c] := 0;
   writeln ('Hãy gõ vào các kí tự');
   writeln ('và kết thúc bằng dấu chấm (.) :');
   Repeat
      Read (c);
      if ('0' < = c) and (c < = '9') then
      số [ c] := số [ c] + 1;
   Until c = dấu chấm;
   for c := '0' to '9' do
      writeln ('Số các chữ số', c, ' đã đọc =', số [ c] : 2)
   Readln
end.
```

Cho chạy chương trình ta được kết quả như sau :

```
Hãy gõ vào các kí tự
và kết thúc bằng dấu chấm (.) :
ytr7657g858450020820.
Số các chữ số 0_đã đọc = 4
Số các chữ số 1_đã đọc = 0
Số các chữ số 2_đã đọc = 2
Số các chữ số 3_đã đọc = 0
Số các chữ số 4_đã đọc = 1
Số các chữ số 5_đã đọc = 3
```

```
Số các chữ số 6_đã đọc = 1
Số các chữ số 7_đã đọc = 2
Số các chữ số 8_đã đọc = 3
Số các chữ số 9_đã đọc = 0
```

V.2.2. Bài toán 8 quân hậu

Hãy đặt 8 quân hậu lên bàn cờ vua (có 8 x 8 ô) sao cho không có quân nào ăn được quân nào ? Một quân hậu có thể ăn được bắt cứ quân nào nằm trên cùng cột, cùng hàng hay cùng đường chéo thuận nghịch với nó.

Bài toán này do Call Friedrich Gauss đưa ra vào năm 1850 nhưng không có lời giải hoàn toàn theo phương pháp giải tích. Lý do là loại bài toán này không phù hợp với các phương pháp giải tích mà phải tìm cách khác để giải trên MTĐT, có thể thử đi thử lại nhiều lần.

Niclaus Wirth trình bày phương pháp *thử-sai* (trial-and-error) như sau :

Đặt một quân hậu vào cột 1 (trên một hàng tuỳ ý);

Đặt tiếp một quân hậu thứ hai sao cho 2 quân không ăn nhau;

Tiếp tục đặt quân thứ 3, v.v...

Lời giải có dạng một vòng lặp như sau :

```
Xét-cột-đầu;
Repeat
   Thử_cột;
   if an_toàn then begin
      Đặt_quân_hậu_vào;
      Xét_cột_kế_tiếp;
   end
   else Quay_lại;
until Đã_xong_với_cột_cuối or Đã_quay_lại_quá_cột_đầu;
```

Các công việc được tinh chế dần dần bằng cách chọn các việc đơn giản, có cách giải ngay để tiến hành trước như sau :

Gọi bàn cờ vua 8 × 8 gồm các ô (i, j) ở cột j, hàng i với j=1..8 và i=1..8, ta có :

Xét_cột_đầu : Bắt đầu với cột j=1.

Xét_cột_kế_tiếp : Tức là chuyển qua xét cột kế tiếp và chuẩn bị xét hàng đầu tiên :

```
j:= j+1; i:= 0;
```

Đã_xong_với_cột_cuối : Lúc này đã xong cả 8 cột, quân hậu cuối cùng đã được đặt vào bàn cờ : thành công, ta có biểu thức :

```
j > 8
```

Đã_quay_lại_quá_cột_đầu: Tức là đã lùi quá cột đầu tiên : tình trạng bế tắc xảy ra : không tìm ra lời giải !

```
j < 1
```

Thử_cột : Tìm xem có thể đặt quân hậu tại hàng nào ở cột đang xét. Bước *Thử_cột* sẽ có dạng :

```
repeat
   Xét_một_hàng  ;  { là hàng thứ i }
   Kiểm_tra_an_toàn  ;  { khi đặt quân hậu vào hàng này }
Until An_toàn or Đã_xét_đến_hàng_cuối;
```

Lúc đầu I=0, việc *Xét_một_hàng* tức : `i:= i+1`

Từ đó ta có ngay *Đã_xét_đến_hàng_cuối* tức là : `i = 8`

Lúc này người ta tìm cách biểu diễn dữ liệu tương ứng vì các công việc đã có vẻ "mịn" rồi. Theo lời khuyên của Niclaus Wirth, sự biểu diễn dữ liệu càng trì hoãn lâu càng tốt (đến khi không thể trì hoãn được nữa) !

Vì bàn cờ có 8 x 8 ô nên có thể nghĩ ngay đến việc sử dụng một ma trận Boolean hai chiều để biểu diễn :

```
Var B : array [ 1..8, 1..8] of Boolean;
```

B [i, j] có giá trị true nếu có quân hậu ở hàng i, cột j.

Tuy nhiên, cách biểu diễn này gây khó khăn cho việc kiểm tra hai đường chéo có 2 quân hậu nào ăn nhau không theo luật cờ vua ?

Bây giờ ta dùng 3 dãy Boolean a, b, c với :

a [i] = true nếu không tồn tại quân hậu nào nằm trên hàng i.

b [k] = true nếu không tồn tại quân hậu nào nằm trên đường chéo thuận thứ k.

c [l] = true nếu không tồn tại quân hậu nào nằm trên đường chéo nghịch thứ l.

Với mỗi ô (i, j) hàng i cột j, ta có quan hệ như sau :

–đường chéo thuận thứ k thoả mãn i + j = k;

–đường chéo nghịch thứ l thoả mãn i - j = l;

Vì vậy, nếu : $1 \le i, j \le 8$ thì : $2 \le k \le 16$ và : $-7 \le l \le 7$.

Ta có các mảng a , b , c như sau :

```
var  a : array [ 1..8] of boolean;
     b : array [ 2..16] of boolean;
     c : array [ -7..7] of boolean;
```

Để biểu diễn sự kiện đặt quân hậu tại cột j vào hàng i, ta dùng dãy nguyên x sao cho x [j] = i nếu như có một quân hậu ở ô (i, j) :

```
var x : array [ 1..8] of integer;
```

Việc đặt quân hậu vào ô (i, j) sẽ làm cho :

a [i] = b [i+j] = c [i-j] = false

Kiểm_tra_an_toàn : Cho đến lúc này, chưa có hai quân hậu nào trong số những quân đã đặt lên bàn cờ có thể ăn lẫn nhau. Điều kiện *An_toàn* để đặt quân hậu vào ô (i, j) là :

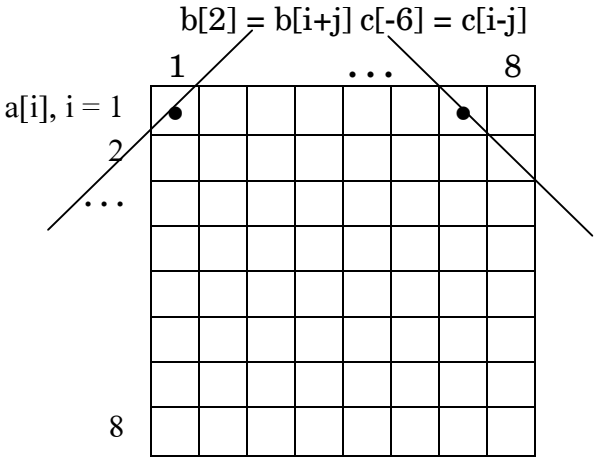
a [i] = b [i+j] = c [i-j] = true;

Bằng cách sử dụng một biến logic :

```
Var Antoan: Boolean;
```

Việc *Kiểm_tra_an_toàn* được dịch ra Pascal như sau :

```
An toàn := a [ i] and b [ i + j] and c [ i - j];
```



Hình 2.8. Bàn cờ vua cho bài toán tám quân hậu

Đặt quân hậu vào ô (i, j) *Đặt_quân_hậu_vào* sẽ là :

```
x[ j] := i;
a [ i] := false;
b [ i+j] := false;
c [ i-j] := false;
```

Tiếp tục tinh chế bước phức tạp nhất là *Quay_lại* :

Quay_lại : là quay lại một cột ở trước cột đang xét để đặt lại quân hậu cho cột đó khi tình thế hiện trạng là bế tắc.

Bước *Quay_lại* có dạng :

```
Xét_lại_cột_trước;
if not Đã_quay_lại_quá_cột_đầu then begin
   Bỏ_quân_hậu_ở_cột_đó; { tức cột trước cột đang xét, ô (i, j) }
   if Đang_ở_hàng_cuối_cùng then begin
```

```
        Xét_lại_cột_trước ;
        if not Đã_quay_lại_quá_cột_đầu then
           Bỏ_quân_hậu_ở_cột_đó
     end
  end;
```

Dễ dàng ta thấy *Xét_lại_cột_trước* tức là :

```
j = j - 1;
```

Còn *Đã_quay_lại_quá_cột_đầu* thì đã xét trước đây, tức là :

```
j < 1;
```

Thao tác *Bỏ_quân_hậu_ở_cột_đó* sẽ có dạng:

```
i:= x [ j] ; a [ i] := true; b[ i+j] := true; c[ i-j] := true;
```

Chương trình hoàn chỉnh như sau :

```
Program TamQuânHau;
Uses Crt;
Const Hau='Q ';ov='#';
Var x: array[ 1..8] of Integer;
    a: array[ 1..8] of Boolean;
    b: array[ 2..16] of Boolean;
    c: array[ -7..7] of Boolean;
    i,j: Integer;
    antoan: Boolean;
Begin
   for i:=1 to 8 do a[ i] :=true;
   for i:=2 to 16 do b[ i] :=true;
   for i:=-7 to 7 do c[ i] :=true;
   j:=1; i:=0;
   repeat
      repeat
         i:=i+1;
         antoan:=a[ i] and b[ i+j] and c[ i-j] ;
      until antoan or (i=8);
      if antoan then begin
         x[ j] :=i;
         a[ i] :=false; b[ i+j] :=false; c[ i-j] :=false;
         j:= j+1; i:= 0
      end else begin
         j:=j-1;
         if j>=1 then begin
            i:=x[ j] ;
            a[ i] :=true; b[ i+j] :=true; c[ i-j] :=true;
            if i=8 then begin
```

```
                j:=j-1;
                if j>=1 then begin
                   i:=x[ j] ;
                   a[ i] :=true; b[ i+j] :=true; c[ i-j] :=true
                end
             end
          end
       end
    until (j>8) or (j<1);
    if j<1 then writeln('Khong co loi giai!')
    else
    for i:=1 to 8 do begin
       for j:=1 to 8 do
       if x[ j] =i then write(Hau) else write(ov);
       writeln
    end;
    readln
end.
```

Kết quả chạy chương trình như sau :

```
Turbo Pascal  Version 7.0  Copyright (c) 1983,92 Borland International
Q # # # # # # #
# # # # # # Q #
# # # # Q # # #
# # # # # # # Q
# Q # # # # # #
# # # Q # # # #
# # # # # Q # #
# # Q # # # # #
```

V.3. Sửa đổi chương trình

Chương trình viết xong chạy tốt chưa có nghĩa quá trình lập trình đã xong. Do nhu cầu, có thể cần sửa đổi lại theo một cách nào đó cho phù hợp. Nhờ phương pháp tinh chế từng bước mà người lập trình có thể dễ dàng nhìn thấy những chỗ cần chỉnh sửa trong chương trình. Đây là khả năng duy trì (Maintainability) của phương pháp.

Một đặc tính khác của phương pháp tinh chế từng bước là tính phổ cập (portability)) của chương trình : ta dễ dàng chuyển đổi sang một môi trường khác, tức là chuyển sang một ngôn ngữ lập trình khác, hoặc một hệ thống máy tính khác. Để minh hoạ, ta xét bài toán 8 quân hậu tổng quát như sau :

Tìm tất cả các phương án có thể đặt 8 quân hậu lên bàn cờ sao cho không có hai quân nào ăn lẫn nhau.

Từ tinh chế lần 1 trong mục trước, ta cần có hai sửa đổi như sau :

- Khi đã đến cột cuối cùng và đặt quân hậu cuối cùng vào bàn cờ, ta in lời giải ra nhưng chưa kết thúc chương trình ngay mà tiếp tục quay trở lại để tìm lời giải khác.
- Chương trình ngừng khi sự quay lại đã quá cột đầu.

Lời giải có dạng phác thảo như sau :

```
Xét_cột_đầu;
repeat
   Thử_cột ;
   if An_toàn then begin
      Đặt_quân_hậu_vào ;
      Xét_cột_kế ;
      if Cột_kế_vượt_quá_cột_cuối_cùng then begin
         In_ra_lời_giải;
         Quay_lại
      end
   end else Quay_lại
Until Đã_quay_lại_quá_cột_đầu;
```

Từ đây, với các bước làm mịn đã giải quyết ở mục trước, ta có thể viết lại thành chương trình hoàn chỉnh.

Bài toán mã đi tuần

Cho bàn cờ n × n ô và một quân mã đang ở toạ độ x_0, y_0. Hãy tìm cách cho quân mã đi theo luật cờ vua để qua hết tất cả các ô của bàn cờ, mỗi ô đi qua đúng một lần ?

Cách giải quyết để quân mã đi qua hết $n^2 - 1$ ô của bàn cờ là tại mỗi ô mà quân mã đang đứng, hãy xác định xem có thể thực hiện một nước đi kế tiếp nữa hay không ? Như vậy thuật toán để tìm nước đi kế tiếp có thể viết thành thủ tục đệ quy dạng phác thảo như sau :

```
Procedure Thử_nước_đi_kế;
Begin
   Khởi_động_nước_đi_có_thể
   Repeat
      Chọn_một_nước_đi
      if OK then begin
         Thực_hiện_nước_đi
         if Chưa_hết_nước then begin
            Thử_nước_đi_kế;
            if NotOK then Xoá_nước_trước
         end else Thành_công
      end
   Until Đi_được or (Hết_nước_đi);
   Kết_thúc
End;
```

Bây giờ ta cần tìm cấu trúc dữ liệu để biểu diễn bàn cờ n × n ô, mỗi ô có toạ độ (i, j), với 1 ≤ i,j ≤ n. Dễ dàng ta tìm được mô tả như sau :

```
Type Idx = 1..n;
Var  H: Array[ Idx, Idx] of Integer;
```

Trong mô tả trên, thay vì sử dụng giá trị kiểu `Bollean` để đánh dấu ô đó đã được đi qua chưa, ta đưa vào giá trị kiểu `Integer` để dò theo quá trình di chuyển của quân mã theo quy ước như sau :

H[x, y] = 0 ô <x, y> chưa được quân mã đi qua

H[x, y] = i ô <x, y> đã được quân mã đi qua ở nước thứ i, $1 \le i \le n^2$

Để chỉ một nước đi có thành công hay không, ta sử dụng biến `Bollean q` với quy ước như sau :

`q = true` nước đi thành công

`q = false` không có nước đi

Ta thấy điều kiện *Chưa_hết_nước* được biểu diễn bởi biểu thức : $i \le n^2$

Giả sử gọi u, v là toạ độ nước đi kế tiếp của quân mã theo luật cờ vua thì điều kiện OK phải thoả mãn :

- Ô mới <u, v> phải thuộc vào bàn cờ, nghĩa là $1 \le u \le n$ và $1 \le v \le n$.
- Quân mã chưa đi qua ô <u, v>, nghĩa là H[u, v] = 0.

Bằng cách xây dựng tập hợp :

```
Var s: set of Idx;
```

biểu thức điều kiện OK bây giờ có thể viết :

```
(u in s) and (v in s) and H[ u, v] =0
```

Để ghi nhận nước đi hợp lệ *Thực_hiện_nước_đi*, ta sử dụng phép gán :

```
H[ u, v]  := i;
```

Từ đó, việc *Xoá_nước_trước* có thể sử dụng phép gán :

```
H[ u, v]  := 0
```

Để ghi nhận kết quả lời gọi đệ quy, ta sử dụng biến `Bollean q1` cho biểu thức điều kiện *Đi_được*. Như vậy, *Thành_công* sẽ là :

```
q1 := true
```

và *Kết_thúc* sẽ là :

```
q := q1
```

Bây giờ ta có lời giải mịn hơn như sau :

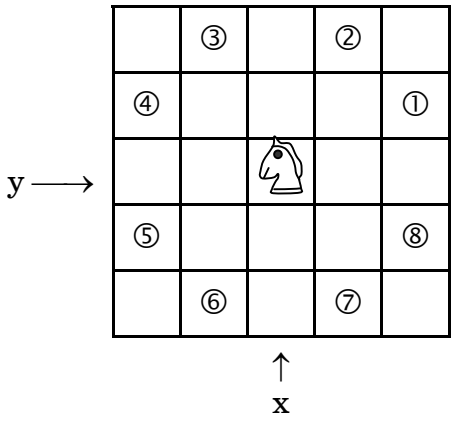
```
Procedure Try(i:Integer; x, y : Idx; Var q: Boolean);
```

```
Var  u, v:Integer;
     q1: Boolean;
Begin
   Khởi_động_nước_đi_có_thể
   Repeat
      Chọn_một_nước_đi
      if (u in s) and (v in s) and H[u, v]=0 then begin
         H[u, v] := i;
         if n < sqr(n) then begin
            Try(i+1, u, v, q1);
            if not q1 then H[ u, v] := 0
         end else q1:= true
      end
   Until q1 or (Hết_nước_đi);
   q := q1
End;
```

Cho đến lúc này, ta chưa xét đến luật đi của quân mã, nghĩa là chương trình xây dựng ở trên độc lập với luật cờ vua với chủ ý giảm nhẹ những chi tiết chưa cần thiết khi phát triển chương trình.

Như vậy ta vẫn còn hai việc chưa giải quyết là : *Khởi_động_nước_đi_có_thể* và *Chọn một_nước_đi*.

Cho trước một toạ độ bất kỳ <x, y> của quân mã trên bàn cờ, ta có thể có tám ô <u, v> được đánh số từ 1..8 (theo chiều ngược kim đồng hồ) mà quân mã có thể nhảy đến như hình dưới đây :



Hình 2.9. Các vị trí khác nhau của quân mã

Để có được <u, v>, từ <x, y>, ta cần xác định giá trị chênh lệch theo toạ độ. Ta sẽ dùng hai mảng một chiều a và b, mỗi mảng sẽ có kích thước 8 phần tử, để lưu giữ 8 giá trị chênh lệch theo toạ độ <x, y>, với quy ước chiều đi ↑ và → mang dấu +, chiều đi ← và ↓ mang dấu –. Ta có khai báo như sau :

```
Var a, b: Array[ 1..8] of integer;
```

Chẳng hạn nếu cho <x, y> = <x_0, y_0> với điều kiện n–2 ≥ x_0, y_0 ≥ 3 thì ta có thể có 8 cặp giá trị như sau :

```
① a[ 1] := 2;    b[ 1] := 1;
② a[ 2] := 1;    b[ 2] := 2;
③ a[ 3] := -1;   b[ 3] := 2;
④ a[ 4] := -2;   b[ 4] := 1;
⑤ a[ 5] := -2;   b[ 5] := -1;
⑥ a[ 6] := -1;   b[ 6] := -2;
⑦ a[ 7] := 1;    b[ 7] := -2;
⑧ a[ 8] := 2;    b[ 8] := -1;
```

Bây giờ, để đánh số các nước đi có thể, ta sử dụng một biến k nguyên, k sẽ nhận giá trị trong phạm vi 1..8. Như vậy, đầu thủ tục, việc *Khởi_động_nước_đi_có_thể* tương ứng với lệnh gán :

```
k:= 0;
```

Việc *Chọn một_nước_đi* tương ứng với các lệnh gán :

```
k:= k + 1;
q1:= false;
u:= x + a[ k] ; v:= y + b[ k] ;
```

Còn biểu thức điều kiện *Hết_nước_đi* sẽ tương ứng với : `k = 8`

Thủ tục đệ quy được bắt đầu bởi toạ độ <x_0, y_0> = <1,1>, kể từ nước đi k=2, các ô của bàn cờ đều có thể là đích của quân mã với khởi động :

```
for i:=1 to n do for j:=1 to n do H[ i, j] := 0;
```

Lời gọi thủ tục như sau :

```
H[ 1, 1] := 1; Try(2, 1, 1, q);
```

Cuối cùng là một thay đổi nhỏ bằng cách thêm biến nguyên `nsq` để tính `sqr(n) ngoài thủ tục`. Chú ý rằng n ≥ 5. Sau đây là chương trình hoàn chỉnh :

Chương trình mã đi tuần :

```
PROGRAM KnightsTour;
Uses CRT, Dos;
Const NMax=50;
Type Idx = 1..Nmax;
Var  i, j: Idx;
     N, Nsq: integer;
     q: Boolean;
     s: set of Idx;
     H: Array[Idx, Idx] of Integer;
     a, b: Array[1..8] of integer;
     gio, phut, giay, hund : Word; { Để tính thời gian }
Procedure Try(i:Integer; x, y : Idx; Var q: Boolean);
```

```
Var k, u, v:Integer; q1: Boolean;
Begin
k:= 0;
  Repeat
    k:= k + 1; q1:= false; u:= x + a[k]; v:= y + b[k];
    if (u in s) and (v in s) and (H[u, v]=0) then begin
       H[u, v] := i;
       if i < Nsq then begin
          Try(i+1, u, v, q1);
          if not q1 then H[u, v] := 0
       end else q1:= true
    end
  Until q1 or (k=8);
  q := q1
End { Try };
Begin { KnightsTour main }
   ClrScr;
   a[1]:= 2;   b[1]:= 1;
   a[2]:= 1;   b[2]:= 2;
   a[3]:= -1;  b[3]:= 2;
   a[4]:= -2;  b[4]:= 1;
   a[5]:= -2;  b[5]:= -1;
   a[6]:= -1;  b[6]:= -2;
   a[7]:= 1;   b[7]:= -2;
   a[8]:= 2;   b[8]:= -1;
   Write('Cho N = '); Readln(N);
   While (N>1) and (N<Nmax) do begin
      { Giờ bắt đầu tính }
      GetTime(gio, phut, giay, hund);
      Writeln('Bắt đầu =', gio:2,':', phut:2,':', giay:2,':',
      hund);
      nsq:= sqr(N);
      s:=[1..n];
      for i:=1 to n do for j:=1 to n do H[i, j]:= 0;
      H[1, 1]:= 1; Try(2, 1, 1, q);
      if q then
         for i:=1 to N do begin
             for j:=1 to N do
                write(h[i, j]:5);
             Writeln
         end
      else Writeln('Không có lời giải !');
      { Giờ bắt đầu tính }
      GetTime(gio, phut, giay, hund);
      Writeln('Kết thúc=', gio:2,':', phut:2,':', giay:2,':',
      hund);
      Write('Cho N = '); Readln(N);
   End {While}
End.
```

Sau đây là kết quả với N = 5 :

Cho N = 5

Bắt đầu = 5:59:57:30

1	6	15	10	21
14	9	20	5	16
19	2	7	22	11
8	13	24	17	4
25	18	3	12	23

Kết thúc = 5:59:57:36

Kết quả với N = 6 :

Cho N = 6

Bắt đầu = 6: 0:40:80

1	16	7	26	11	14
34	25	12	15	6	27
17	2	33	8	13	10
32	35	24	21	28	5
23	18	3	30	9	20
36	31	22	19	4	29

Kết thúc = 6: 0:41:79

VI. Phụ lục - Đơn vị trong Turbo Pascal

Đơn vị (Unit) trong Turbo Pascal thể hiện tính cấu trúc trong lập trình : cho phép chia chương trình lớn thành một hệ thống phân cấp gồm một chương trình chính và nhiều đơn vị chương trình con.

VI.1. Giới thiệu Unit

Unit là tập hợp khai báo các hằng, các kiểu dữ liệu, các biến, các thủ tục và hàm có quan hệ với nhau để đưa vào sử dụng trong chương trình chính.

Mỗi đơn vị, được cất giữ trên thiết bị nhớ phụ (đĩa từ) dưới dạng một tệp chương trình Pascal *.PAS và được dịch (compile) riêng rẽ. Kết quả dịch là một tệp mới có phần mở rộng là `*.TPU`.

Để gọi các Unit, trong phần đầu chương trình sử dụng lệnh :

```
USES <tên Unit>
```

trong phần đầu chương trình.

Thư viện các chương trình mẫu Turbo Pascal có 8 Unit chuẩn như sau :

System	chứa các hàm và thủ tục thư viện thông dụng mức hệ thống : xử lý tệp, xử lý chuỗi, tính các hàm toán học. System được gọi mặc nhiên mà không cần khai báo : **USES System**
Dos	cung cấp các chức năng của hệ điều hành MS-DOS
Crt	để điều khiển các thiết bị vào (bàn phím) và ra (màn hình) : Goto XY, Clrscr...
Graph	cung cấp cácc khả năng đồ họa (graphics) cho các loại màn hình khác nhau : Hercule, CGA, EGA, VGA...
Turbo3	để tương thích với các chương trình Turbo Pascal V3.0 đã có.
Graph3	thực hiện các chương trình con đồ hoạ theo kiểu con rùa (tortoise) của Turbo Pascal V3.0

VI.2. Cấu trúc của Unit

Unit do NSD tạo ra. Mỗi Unit được đặt trong một tệp chương trình, gồm những thành phần như sau :

a) Phần tên của Unit (Unit Heading)

```
Unit <tên Unit>
```

b) Phần giao tiếp (Interface Section)

```
Interface
        { Các khai báo sau đây không bắt buộc phải có }
        Uses <Ds các Units dùng cho Unit này>
        Const <Ds các hằng>
        Type <Ds các mô tả kiểu>
        Var <Ds các khai báo biến>
        <Ds các phần đầu của các thủ tục và hàm>
```

Các khai báo Const, Type và Var sau Interface cho Unit cũng dùng được tại nơi sử dụng đơn vị này. Ta nói chúng là "thấy được" (Visible).

Danh sách các tên thủ tục và hàm được các Unit khác dùng đến sẽ được khai báo đầy đủ trong phần tiếp theo :

c) Phần hiện thực (Implementation Section)

```
Implementation
        { Các khai báo sau đây không bắt buộc phải có }
        Uses <Ds các Uses sử dụng đến>
        Label <Ds các nhãn>
        Const <Ds các hằng>
        Type <Ds các mô tả kiểu>
        Var <Ds các khai báo biến>
        <Các mô tả hàm và/hoặc thủ tục>
```

Các mô tả hàm và/hoặc thủ tục trong phần này gồm có :

- Các hàm và /hoặc thủ tục đã mô tả ở phần giao tiếp.
- Các hàm và/hoặc thủ tục nội bộ dùng riêng trong phần hiện thực (không khai báo trong phần giao tiếp).

Các mô tả nhãn, hằng, kiểu dữ liệu, biến và các hàm, thủ tục nội bộ trong phần hiện thực của một Unit là không dùng được tại nơi sử dụng Unit này. Ta gọi chúng là "bị dấu" (Hidden).

d) Phần khởi động (Initialization section)

```
Begin
    <Các lệnh Pascal>
```

Phần này có thể vắng mặt nhưng **end.** phải có mặt !

```
end.
```

Khi nơi sử dụng một Unit có phần khởi động Initialization thì phần khởi động của Unit này sẽ được gọi chạy trước khi thân của nơi sử dụng chạy.

Nếu có một nơi sử dụng nhiều Unit thì phần khởi động của các Unit đó sẽ được chạy theo thứ tự xuất hiện của tên của các Unit trong khai báo Uses của nơi gọi.

Thường thì trong phần khởi động của một đơn vị, người ta làm các động tác chuẩn bị như khởi gán cho các biến, mở các tệp, thông báo chế độ chạy chương trình...

VI.3. Cách sử dụng Unit

a) Một chương trình hay một Unit có thể sử dụng nhiều Unit khác

Sử dụng một Unit có nghĩa là được quyền sử dụng các hằng, kiểu dữ liệu, biến, các hàm và/hoặc thủ tục đã được khai báo trong phần giao tiếp Interface của Unit đó.

Ví dụ : Để sử dụng các Units có tên lần lượt là U1, U2, ..., Un, dùng mệnh đề Uses đặt sau khai báo Program trong chương trình :

Uses U1, U2, . . .,Un ;

b) Khi có một sửa đổi nào đó trong phần Interface của một Unit

Khi đó, những (tệp) chương trình hay (tệp) Unit đó cần được dịch (Compile) lại.

Khi trong một Unit chỉ có sự sửa đổi ở phần Implementation hay ở phần Initialization thì không cần dịch lại những (tệp) chương trình hay (tệp) Unit sử dụng Unit đó.

Ví dụ 11 :

Giả sử chương trình P gọi U1 và U1 sử dụng U2 :

```
Program P ;
Uses U1 ;
Const
   a = b;
Begin
   writeln (a);
end.
```

P.PAS

⟶

```
Unit U1 ;
Interface
   Uses U2 ;
   Const b = c;
Implementation
end.
```

U1.PAS

⟶

```
Unit U2 ;
Interface
   Const c = 10;
Implementation
end.
```

U2.PAS

Nếu trong U2, đổi c=5 chẳng hạn thì cần phải dịch lại U2.PAS và U1.PAS (vì U1 sử dụng U2) và dịch lại P.PAS.

c) Trùng tên hằng, biến, kiểu dữ liệu và tên CT con (hàm, thủ tục)

Để phân biệt, người ta đặt trước tên trùng đó tên Unit có chứa tên này và cách một dấu chấm (.). Riêng chương trình chính (khai báo Program) thì không cần.

Ví dụ 12 :

Giả sử chương trình P sử dụng các Unit U1 và U2.

Nếu trong P, trong Interface của U1 và U2 đều có khai báo biến i thì để phân biệt i nào là của P, i nào là của U1 và U2 người ta viết :

i {i của P}
U1.i {i của U1}
U2.i {i của U2}

VI.4. Ví dụ về Unit

Viết chương trình tính nhiều lần diện tích hình tròn với bán kính nhập vào từ bàn phím, cho đến khi bán kính nhập vào là 0 thì dừng.

Gọi chương trình chính là HINHTRON và Unit sử dụng để tính diện tích hình tròn là DTHTRON, ta có :

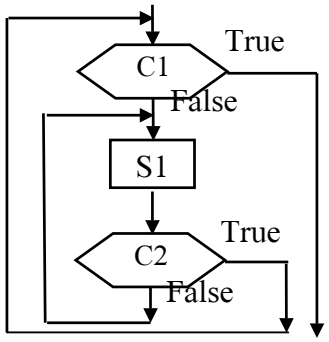
```
Program HINHTRON ; {Tệp HINHTRON.PAS}
Uses Crt, DTHTRON ;
Var bk : Real ;
Begin
   Write ('Số Pi = ', Pi) ; {hằng Pi khai báo trong Unit DTHTRON}
   Repeat
      Write (bán kính =) ; Readln (bk) ;
      if bk > 0 then
         Writeln ('Diện tích = ', Dientich (bk) ) ;
   Until bk = 0
End. { HINHTRON}

Unit DTHTRON ; {Tệp DTHTRON.PAS}
Interface
   Const Pi = 3.1415926535 ; {1/π = 0.318309886}
   Function Dientich (R : Real) : Real;
Implementation
   Function Dientich;
   Begin
      Dientich := Pi * Sqr (R)
   End;
Begin {phần khởi động Inilialization}
```

```
   Writeln ('trong chương trình chính sẽ sử dụng Unit DTHTRON !')
End.
```

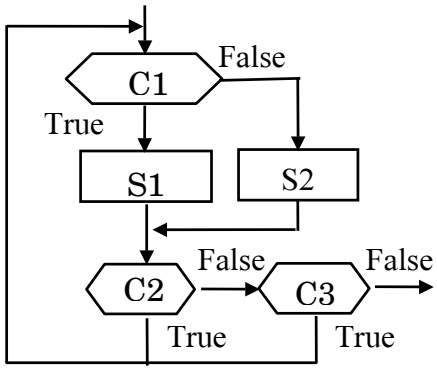
VI.5. Bài tập

1. Sử dụng các câu lệnh Pascal để viết chương trình theo sơ đồ khối dưới đây bằng cách chỉ sử dụng các cấu trúc điều khiển cơ bản. Sau đó hãy đổi về dạng chương trình chỉ sử dụng ba cấu trúc điều khiển là tuần tự, điều kiện **if then else** và lặp **While**.

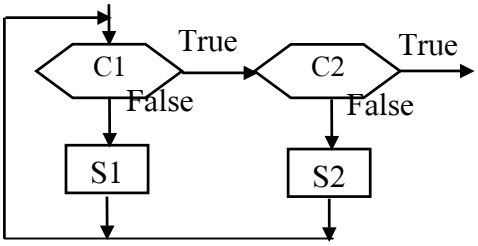
1.



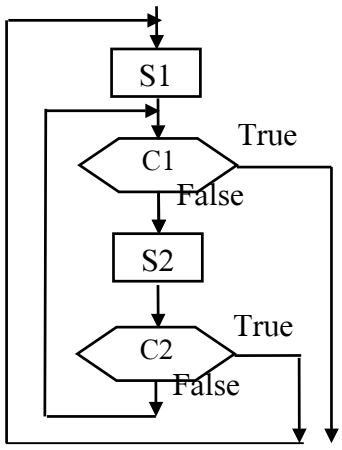
2.



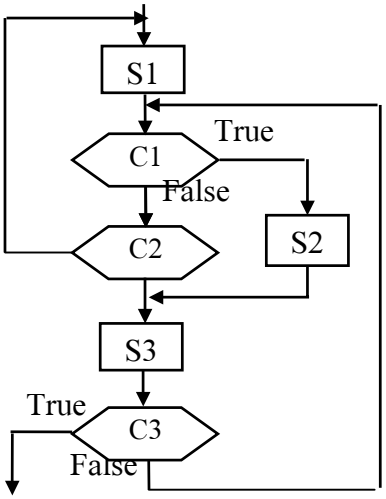
3.



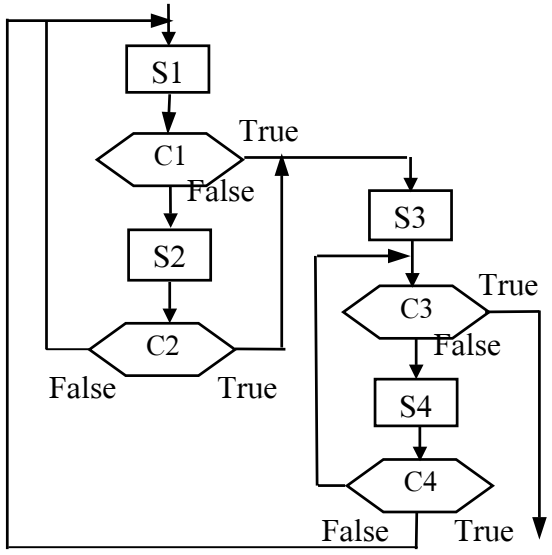
4.



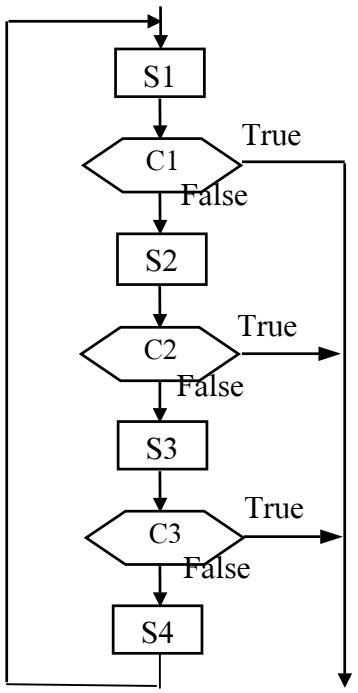
5.



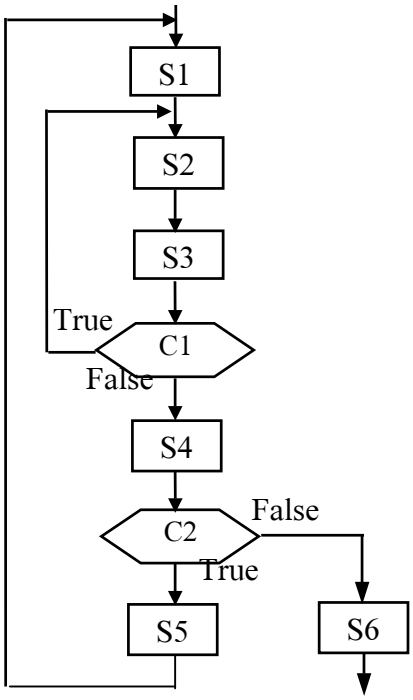
6.



7.



8.



2. **Xử lý chuỗi** (string) : Viết chương trình đọc một câu (kết thúc bởi Enter ↵) sau đó tiến hành các công việc :

- Thống kê số từ, số ký tự trong câu.
- Tách câu ra thành các từ cách nhau bởi dấu cách (space). In ra các từ này.
- Nén câu (bỏ các dấu cách giữa các từ). In kết quả.
- Thay thế những xuất hiện của câu con S1 thành câu con S2 và in kết quả. (S1 và S2 là các câu con nhập vào)

Yêu cầu sử dụng Unit cho mỗi việc. Chương trình chính dùng menu để gọi.

3. Viết chương trình xử lý ma trận vuông A cấp n x n dưới dạng menu, gồm các việc sau :

- Đọc vào ma trận vuông cấp n x n.
- Tính định thức của ma trận.
- Kiểm tra tính đối xứng qua đường chéo chính của ma trận.
- Xác định xem ma trận có dạng tam giác trên không ? (các phần tử phía dưới đường chéo chính đều = 0)
- Xác định xem ma trận có dạng tam giác dưới không ? (các phần tử phía trên đường chéo chính đều = 0)

Yêu cầu dùng Unit. Bốn việc sau cùng chỉ có hiệu lực khi ma trận đã được đọc.

CHƯƠNG 3

Hợp thức hóa phần mềm

I. Xác minh và hợp thức hóa phần mềm

Người ta thường phân biệt 2 yếu tố trong hoạt động sản xuất phần mềm :

1. Xây dựng đặc tả, lập trình và lập hồ sơ (viết các hướng dẫn sử dụng v.v ...) Yếu tố này tạo nên giai đoạn phát triển chương trình.
2. Xác minh (hay kiểm tra) và hợp thức hóa các sản phẩm phần mềm là việc so sánh các sản phẩm phần mềm đó với các đặc tả của chúng sao cho có quan hệ thỏa mãn.

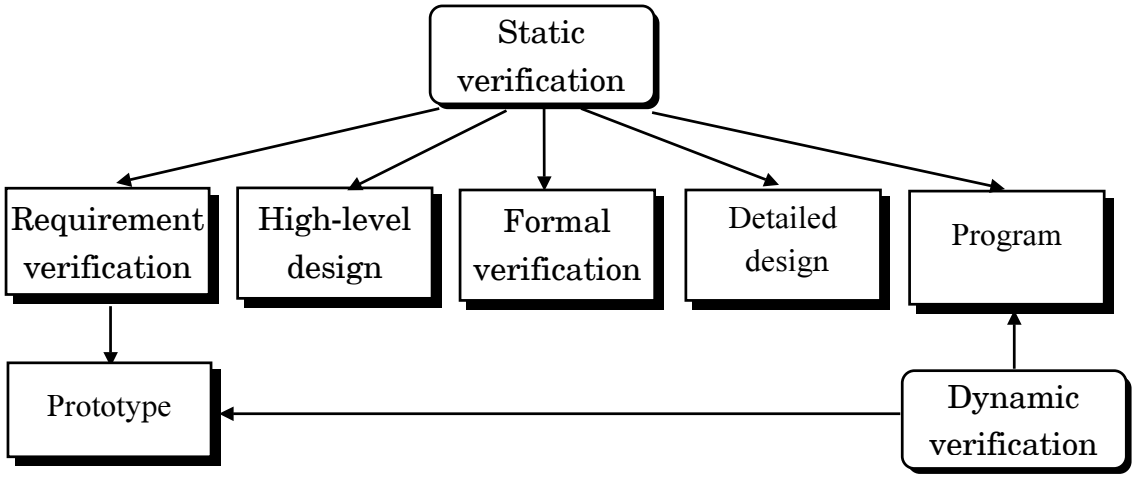
Người ta dùng thuật ngữ *xác minh* (verification) khi so sánh một sản phẩm với một đặc tả chặt chẽ (rigourous). Khi đặc tả là không hình thức (informal), người ta dùng thuật ngữ *hợp thứa hóa* (validation).

Giai đoạn xác minh và hợp thức hóa, gọi tắt là giai đoạn V & V (Verification and Validation) bao giờ cũng có mặt trong mọi dự án Tin học.

Sự khác nhau cơ bản giữa V & V được Boehm B.W. (1979) tóm tắt như sau :

Validation : *Are we building the right product?*

Verification : *Are we building the product right?*



Hình 3.1. Kỹ thuật tĩnh và kỹ thuật động của quá trình V&V

Để thực hiện quá trình V&V, người ta sử dụng các kỹ thuật tĩnh (static techniques) và động (dynamic techniques) để kiểm tra hệ thống.

Kỹ thuật tĩnh nhằm phân tích biểu diễn hệ thống qua việc phân tích yêu cầu, phân tích thiết kế và hiển thị (listing) chương trình. Kỹ thuật động là việc thử nghiệm(testing) chương trình

Kỹ thuật tĩnh bao gồm việc thanh tra (inspection) chương trình, phân tích và xác minh hình thức (formal verification) hay chứng minh sự đúng đắn (prouving) của chương trình.

II. Chứng minh sự đúng đắn của chương trình

Phương pháp chứng minh là sử dụng các định lý để minh họa tính đúng đắn của sản phẩm cần xác minh. Phương pháp này không có khả năng hợp thức hóa các đặc tả phi hình thức, bởi vì không thể chứng minh một cách toán học các tính chất không được định nghĩa chặt chẽ. Người ta phân biệt các phép chứng minh hình thức, được diễn tả trong lý thuyết logic, và các chứng minh phi hình thức nhưng chặt chẽ, như trong các cuốn sách về Toán học.

Mọi phép chứng minh hình thức tính đúng đắn của chương trình được xây dựng một cách tường minh từ các tiên đề và các quy tắc suy diễn logic. Thực tiễn cho thấy không thể xây dựng một phép chứng minh như vậy mà không sử dụng đến những công cụ như là các công cụ chứng minh định lý.

Những phép chứng minh hình thức cho các chương trình cỡ hàng ngàn dòng lệnh đã được thực hiện. Chúng cho phép khẳng định tính được phê phán của chương trình.

Trong phép chứng minh không hình thức, người ta định nghĩa kiến trúc tổng quan của phép chứng minh, xử lý những điểm khó khăn, để lại cho người đọc sự chăm sóc chi tiết đến các điểm khác.

Ví dụ để chứng minh không hình thức sự đúng đắn của một chương trình, người ta có thể diễn tả các tất biến của vòng lặp và của thủ tục đệ quy. Một phép chứng minh phi hình thức không cần thiết phải sử dụng các công cụ, vấn đề là người đọc sẽ tự kiểm chứng thông qua các cuộc trao đổi, thảo luận (chẳng hạn tổ chức thanh tra căn cứ trên việc xác minh).

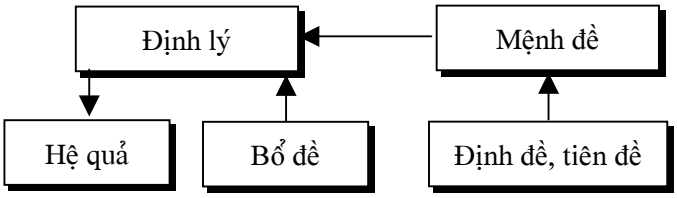
Người ta còn có thể thử chứng minh phi hình thức các tính chất đã phát biểu ít nhiều có tính chặt chẽ (chẳng hạn đề cập đến vấn đề hợp thức hóa), khái niệm chứng minh tính đúng đắn theo nghĩa Toán học được thay thế bởi khái niệm biện luận (reasonig - argumentation) nhằm thuyết phục các chuyên gia Tin học.

II.1. Suy luận Toán học

Trong lĩnh vực suy luận Toán học, người ta thường đặt ra hai vấn đề :

1. *Khi nào thì một suy luận là đúng ?*
2. *Có thể sử dụng những phương pháp nào để xây dựng các suy luận Toán học ?*

Suy luận Toán học là một hình thức tư duy mà từ một hay nhiều mệnh đề logic đã có (phán đoán) rút ra được một mệnh đề logic mới. Kết quả cuả một suy luận nào đó phải là đúng hoặc là sai. Trong Toán học, định lý là một phát biểu có thể chứng minh được là đúng. Người ta hay gặp mô hình chứng minh một định lý Toán học (là đúng) như sau :



Hình 3.2. Chứng minh một định lý Toán học

II.1.1. Các quy tắc suy luận Toán học

Để trình bày các quy tắc suy luận Toán học, chúng ta nhắc lại các phép toán logic sau :

- ¬ **not** (không)
- ∧ **and** (và)
- ∨ **or** (hoặc)
- → **implicate** (kéo theo)
- ~ **equivalence** (tương đương)

Chú ý : a → b tương đương với ¬a ∨ b, hay **if** a **then** b **else true**
a ~ b có nghĩa (a → b) ∧ (b → a).

Thứ tự ưu tiên của các phép toán logic là ¬, ∧, ∨, →, ~. Bảng logic như sau :

a	b	**¬a**	a ∧ b	a ∨ b	a → b	a ~ b
0	0	1	0	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	0	0
1	1	0	1	1	1	1

Để chứng minh một định lý, người ta sử dụng một số tiên đề và quy tắc suy luận, hay là các hằng đúng. Bảng dưới đây trình bày các quy tắc suy luận được sử dụng trong chứng minh tính đúng đắn của chương trình.

Stt	*Quy tắc suy luận*	*Tên gọi*	*Ví dụ*
1	$\dfrac{p}{\therefore p \vee q}$	Luật khẳng định	*Nhạc của Trịnh Công Sơn hay. Vậy nhạc của Trịnh Công Sơn hay hoặc ca sĩ Khánh Ly hát hay.*
2	$\dfrac{p \wedge q}{\therefore p}$	Luật rút gọn	*Tháng này trời nắng hạn và sông Đà thì cạn nước. Vậy trời nắng hạn.*
3	$\dfrac{p \to q \quad p}{\therefore q}$	Luật tách rời (Modus Ponens)	*Nếu cơm chín thì cần tắt lửa.* *Cơm đã chín.* *Vậy cần tắt lửa.*
4	$\dfrac{p \to q \quad \neg q}{\therefore \neg p}$	Luật phủ định (Modus Tollens)	*Nếu mặt trời ở đỉnh đầu thì bóng ngắn nhất.* *Bóng không ngắn nhất.* *Vậy mặt trời không ở đỉnh đầu*
5	$\dfrac{p \to q \quad q \to r}{\therefore p \to r}$	Tam đoạn luận giả định	*Nếu trời mưa thì đường HP bị ngập.* *Nếu đường HP bị ngập thì phải xuống xe dắt bộ. Vậy trời mưa thì phải xuống xe dắt bộ.*
6	$\dfrac{p \vee q \quad \neg p}{\therefore q}$	Tam đoạn luận chuyển	*Cu Tý thuộc bài hoặc là cu Tý ham chơi .* *Mà Cu Tý không thuộc bài.* *Vậy cu Tý ham chơi.*

Trong bảng trên, dấu $\therefore$ được đọc là *vậy thì*. Mỗi luật (cơ sở của phép suy luận), chẳng hạn luật tách rời (Modus Ponens), có thể viết dưới dạng hằng đúng :

$$(p \wedge (p \to q)) \to q$$

II.1.2. Khái niệm về chứng minh tính đúng đắn của chương trình

Một chương trình P xác định một thuật toán cho phép nhận vào một tập hợp dữ liệu L để đưa ra một tập hợp kết quả R. Nói cách khác, với mọi $d \in L$, chương trình P xác định hoặc một dãy hữu hạn các phép tính để cho ra một kết quả $P(d) \in R$, hoặc một dãy vô hạn các phép tính : chương trình bị "quẩn" với dữ liệu d.

Mặt khác, P được viết để tính một hàm f nào đó từ $D \subseteq L$ vào R. P đúng nếu và chỉ nếu P tính đúng hàm f, nghĩa là nếu $\forall d \in D$, P(d) xác định (P không quẩn với dữ liệu vào d) và bằng f(d).

Thông thường, để kiểm tra tính đúng đắn của chương trình, người ta dùng phương pháp thử (test) : người ta chọn một dãy các dữ liệu mẫu $d_1, d_2, ..., d_n$, rồi cho P chạy lần lượt với mỗi dữ liệu để kiểm tra rằng $P(d_1) = f(d_1)$, $P(d_2) = f(d_2)$, ..., $P(d_n) = f(d_n)$.

Tính không đầy đủ của phương pháp này thể hiện ở chỗ không thể thử hết mọi dữ liệu của D, dù D hữu hạn : có thể xảy ra P cho kết quả đúng với mọi dữ liệu mẫu d_i đã chọn nhưng với một dữ liệu $d \neq d_i \ \forall \ i$, P cho một kết quả sai.

Hơn nữa, phương pháp thử không bao giờ chứng minh được một chương trình là đúng đắn, chỉ có thể chứng minh được là không đúng, nếu với một giá trị d_i nào đó đã chọn, thì $P(d_1) \neq f(d_1)$.

Về mặt lý thuyết, phương pháp chứng minh tính đúng đắn của chương trình mang tính Toán học bằng cách chứng minh một định lý tương đương : $\forall d \in D, P(d) = f(d)$.

II.1.3. Tiên đề và quy tắc suy diễn

Một cách tổng quát, các phát biểu cần chứng minh có dạng E{P}S, trong đó E và S là các điều kiện, P là dãy các lệnh (hay là một chương trình) theo nghĩa rằng : nếu E đúng trước khi thực hiện P thì nếu P dừng, S đúng sau khi thực hiện P. Người ta gọi E là điều kiện trước (precondition) và S là điều kiện sau (postcondition) của chương trình P.

Các điều kiện trước E và điều kiện sau S được xây dựng từ các biểu thức logic có thể nhận giá trị đúng (1) hoặc sai (0), chúng là mối liên hệ giữa các biến của chương trình (ví dụ các biến a, b, c, p, q, r trong các biểu thức $a = bq + r$, $q \geq 0$, $b^2 - 4ac > 0$, v.v...) cùng các phép toán logic (nếu có).

Tính chất :

> Với mọi chương trình P và mọi điều kiện C, ta đều có :
> **false** {P} C và C {P} **true**

Việc chứng minh nếu một chương trình P dừng thì P sẽ cho kết quả đúng được gọi là chứng minh tính đúng đắn từng phần.

Trong các chứng minh tính đúng đắn từng phần, các tiên đề và định lý sẽ là các phát biểu có dạng E {P} S. Sau đây là danh sách các tiên đề và các quy tắc suy diễn cho phép chứng minh các định lý dạng E {P} S.

Để chứng minh các quan hệ giữa các điều kiện (ví dụ $E_1 \sim E_2$, $E_1 \rightarrow E_2$, v.v...), người ta sử dụng các tính chất của đại số Boole và miền xác định các biến chương trình (số nguyên trong trường hợp Div).

a) Tiên đề về phép gán

Cho phép gán **x := <bt>** và một điều kiện sau S, ta có tiên đề :

E **{x := <bt> }** S

trong đó E nhận được từ S bằng phép thế các biến x bởi biểu thức <bt>. E là *điều kiện yếu nhất* phải làm thoả mãn các biến trước khi thực hiện phép gán sao cho S là đúng sau đó.

Ví dụ 1

$(xy \geq 0)$ { **z := x*y** } $(z \geq 0)$

$(q + 1 \geq 0)$ { **q := q + 1** } $(q \geq 0)$

$((x+y)^2 = y)$ { **x := x + y** } $(x^2 = y)$

Chú ý quan trọng :

Nhờ quy tắc trên đây, người ta có thể chứng minh điều kiện trước của một lệnh gán, bằng cách sử dụng một điều kiện sau, nhưng ngược lại là không thể. Bởi vậy, để chứng minh tính đúng đắn của chương trình, người ta xuất phát từ điểm kết thúc (điều kiện sau) để tiến hành ngược lên điểm bắt đầu (điều kiện trước).

b) Quy tắc ";" hay tổ hợp các lệnh và lệnh ghép

Gọi E, F, S là các điều kiện, P và Q là các dãy lệnh, ta có :

$$\frac{E\ \{P\}\ F \qquad F\ \{Q\}\ S}{\therefore E\ \{P\ ;\ Q\}\ S} \qquad\qquad \frac{E\ \{P\}\ S}{\therefore E\ \{\textbf{begin}\ P\ \textbf{end}\}\ S}$$

Ví dụ 13 :

Chứng minh **(x=1) { y:= 2; Z:= x + y} (Z=3)**

Là đúng đắn với khẳng định đầu E ≡ (**x = 1)**

Và khẳng định cuối S ≡ (**z = 3)**

Giả sử S đúng, tức z = 3, khi đó nhận được x + y = 3, ta có :

(x+y=3) { Z:= x + y} (Z=3) (1)

Do y được gán giá trị 2, nên nhận được giá trị của x là 1, tức là :

(x=1) { y:= 2} (x+y=3) (2)

Vậy theo quy tắc ";", từ (1) và (2) ta có P kết thúc thì S đúng (đpcm).

II.1.4. Quy tắc điều kiện if B then P

$$\frac{E \wedge B\ \{P\}\ S \qquad E \wedge \neg B \rightarrow S}{\therefore E\ \{\textbf{if B then P}\}\ S}$$

Chú ý :

B phải được xem như một điều kiện sao cho có thể ứng dụng một trong những quy tắc được trình bày ở đây. Điều này có nghĩa rằng việc tính B không làm thay đổi các giá trị của các biến của chương trình.

Ví dụ 2 :

Chứng minh : E{**if x>y then y:=x**} (y ≥ x), với **E** là điều kiện đầu nào đó.

Khi E đúng và x > y đúng (điều kiện B) thì y có giá trị x, vậy y ≥ x (S) , ta có :

$$\mathbf{E} \wedge (x>y)\ \{\ \mathbf{y:=\ x}\ \}\ (y \geq x) \qquad (1)$$

Khi E đúng và x > y sai (¬B) có nghĩa x ≤ y, vậy y ≥ x (S), tức là :

$$\mathbf{E} \wedge (x \leq y) \rightarrow (y \geq x) \qquad (2)$$

Vậy từ (1) và (2) ta có nhận được đpcm.

II.1.5. Quy tắc điều kiện if B then P else Q

$$\frac{E \wedge B\ \{P\}\ S \qquad E \wedge \neg B\ \{Q\}\ S}{\therefore E\ \{\ \mathbf{if\ B\ then\ P\ else\ Q}\ \}\ S}$$

Ví dụ 3 :

Chứng minh : E {**if x < 0 then abs:= -x els abs:= x**}(abs = |x|) với **E** là điều kiện đầu nào đó.

Vậy chương trình là đúng với điều kiện đầu **E** và điều kiện sau abs = |x|.

Khi E đúng và x < 0 đúng (B) thì abs có giá trị –x, tức abs = –x = |x| và điều kiện sau S đúng, ta có :

$$\mathbf{E} \wedge x<0\ \{\ \mathbf{abs:=\ -x}\ \}(abs = |x|) \qquad (1)$$

Khi E đúng và có x < 0 sai (¬B) thì x ≥ 0, khi đó abs có giá trị x tức abs = x = |x| và điều kiện sau S đúng, tức là :

$$\mathbf{E} \wedge x<0\ \{\ \mathbf{abs:=\ x}\ \}(abs = |x|) \qquad (2)$$

Vậy từ (1) và (2) ta có nhận được đpcm.

II.1.6. Quy tắc vòng lặp while

$$\frac{E \wedge B\ \{\ P\ \}\ E}{\therefore E\ \{\ \mathbf{while\ B\ do\ P}\ \}\ E \wedge \neg B}$$

Ở đây, E và B là những điều kiện. Riêng điều kiện E được gọi là *bất biến* của vòng lặp.

Một trong những khó khăn của việc chứng minh tính đúng đắn của chương trình là tìm được bất biến cho mỗi vòng lặp (nghĩa là một bất biến cho phép chứng minh đúng cái yêu cầu). Thực tế *không tồn tại một phương pháp có tính hệ thống và tổng quan để tìm ra những bất biến như vậy*.

Ví dụ 4 :

Sử dụng bất biến của vòng lặp, chứng minh đoạn chương trình tính fac = n!, với n ∈ N sau đây :

```
i := 1; fac := 1;
while i < n do begin
   i := i + 1;
   fac := fac * i
end;
```

Gọi P ≡ {**begin i:= i + 1; fac := fac * i end**}

Giả sử điều kiện E ≡ *(fac = i!) ∧ (i ≤ n)*, ta cần chứng minh E là bất biến của vòng lặp.

Ta sẽ chứng minh bằng quy nạp :

E đúng trước khi vào vòng lặp, vì *i = 1, fac = 1 = 1!* Và *1 ≤ n*.

Giả sử E đúng với *i < n* sau khi thực hiện vòng lặp và sau đó, **while** còn được thực thi một lần nữa. Trước hết *i* được tăng thêm 1 (với lệnh gán `i:= i + 1`) và do vậy vẫn còn *i ≤ n*. Do giả thiết quy nạp *fac = (i – 1) !* trước khi vào vòng lặp nên *fac* sẽ có giá trị là :

$$fac = (i - 1)! * i = i!$$

Từ đó E quả thật là bất biến của vòng lặp và mệnh đề :

$$(E \wedge (i < n))\ \{P\}\ E \text{ đúng.}$$

Từ đó suy ra khẳng định :

$$E\ \{\ \textbf{while i < n do P}\ \}\ E \wedge (i \geq n) \text{ cũng đúng.}$$

Vì vòng lặp kết thúc sau khi lặp n – 1 lần, khi đó *i = n* và *fac = n!*.

II.1.7. Các quy tắc khác

Quy tắc điều kiện trước :

$$\frac{E\ \{P\}\ S \qquad E' \rightarrow E}{\therefore E'\ \{P\}\ S}$$

Quy tắc "và" :

$$\frac{E\ \{P\}\ S \qquad E\ \{P\}\ S'}{\therefore E\ \{P\}\ S \wedge S'}$$

Quy tắc điều kiện sau :

$$\frac{E\ \{P\}\ S \qquad S \rightarrow S'}{\therefore\ E\ \{P\}\ S'}$$

Quy tắc "hoặc" :

$$\frac{E\ \{P\}\ S \qquad E'\ \{P\}\ S}{\therefore\ E \vee E'\ \{P\}\ S \wedge S'}$$

Ví dụ5 :

Chứng minh chương trình con P tính tích hai số nguyên *m* và *n* là đúng :

```
P ≡ function product(m, n: Integer): Integer;
      begin
      {P1 ≡} if n < 0 then a:= –n else a:= n;
      {P2 ≡} k:= 0; x:= 0;
      {P3 ≡} while k < a do begin
                 x:= x + m;
                 k:= k + 1
             end;
      {P4 ≡} if n < 0 then product:= –x
             else product:= x
      end;
```

Ta sẽ chứng minh rằng sau khi thực hiện P thì hàm trả về giá trị là *mn*.

Ta chia P gồm bốn đoạn CT là {P1; P2; P3; P4} như trên.

Gọi E là điều kiện đầu E ≡ «*m, n nguyên*» và S1 ≡ E ∧ (a = |n|).

Khi đó có thể chỉ ra E {P1} S1 là đúng.

Gọi S2 ≡ S1 ∧ (k = *0*) ∧ (x = *0*). Dễ dàng kiểm tra rằng S1 {P2} S2 là đúng.

Ta cũng thấy điều kiện ($x = mk$) ∧ ($k \leq a$) là một bất biến trong vòng lặp P3 tương tự với lý luận quy nạp trong vòng lặp tính *n!*. Vòng lặp này kết thúc sau a bước lặp khi $k = a$, tức $x = ma$ tại điểm này.

Gọi S3 ≡ ($x = ma$) ∧ (a = [n]). Từ đó suy ra S2 {P3} S3 là đúng.

Cuối cùng có thể chỉ ra P4 là đúng với điều kiện đầu S3 và điều kiện cuối S, với S ≡ *product* = *mn*. Vậy S3 {S4} S đúng và hàm trả về giá trị là *mn*.

Từ các mệnh đề E {P1} S1, S1 {P2} S2, S2 {P3} S3 và S3 {S4} S là đúng, theo quy tắc hợp thành, ta có P cũng đúng, tức E { P } S đúng.

Ngoài ra do cả P1, P2, P3 và P4 đều dừng nên P cũng dừng $_{(qed)}$.

II.2. Phương pháp của C.A.R. Hoare

II.2.1. Phát biểu

Sau đây là một ví dụ sử dụng phương pháp của C.A.R. Hoare :

```
Div : r:=a; q:= 0;
      while r >= b do begin
                r:= r - b;
                q:= q + 1;
      end;
```

a, b, q, r là các biến nguyên, a và b được khởi gán giá trị đầu lần lượt là A và B. Với $A \in N$ và $B \in N^+$, hàm Div tính thương q và số dư r của phép chia A cho B. Như vậy với hàm Div ta có :

$D = N \times N^+$, $R = N \times N$ và f là hàm xác định trên $N \times N^+$ vào trong $N \times N$, nghĩa là từ cặp $(A, B) \in N \times N^+$, xác định cặp $(q, r) \in N \times N$ sao cho $A = Bq + r$ và $r < B$.

Bây giờ cần chứng minh Div tính đúng hàm f theo hai bước như sau :

1. *Chứng minh tính đúng đắn từng phần* : nếu Div dừng thì Div sẽ cho kết quả đúng.
2. *Chứng minh tính dừng* : Div dừng với mọi dữ liệu thuộc $N \times N^+$.

Trong trường hợp Div, ta cần chứng minh phát biểu sau :

Nếu trước khi thực hiện lệnh đầu tiên của Div, a và b được gán giá trị đầu $A \in N$ và $B \in N^+$. Sau khi thực hiện, q và r thoả mãn các điều kiện $A = Bq + r$, $r < B$, $r \geq 0$, $q \geq 0$. Ta có :

$(a=A) \wedge (b=B) \wedge (A\geq 0) \wedge (B>0)$ {Div} $(A=Bq+r) \wedge (q\geq 0) \wedge (r\geq 0) \wedge (r<B)$.

II.2.2. Chứng minh tính đúng đắn từng phần của Div

Ta cần chứng minh :

$(a=A) \wedge (b=B) \wedge (A\geq 0) \wedge (B>0)$ {Div} $(A=Bq+r) \wedge (q\geq 0) \wedge (r\geq 0) \wedge (r<B)$

Để dễ theo dõi, ta đặt tên các điều kiện như sau :

$P1 = (a=A) \wedge (b=B)$

$P2 = (A\geq 0) \wedge (B>0)$

$Q1 = (A=Bq+r) \wedge (q\geq 0) \wedge (r\geq 0)\ (r<B)$

Chú ý rằng :

$P1 \wedge P2 \rightarrow P1 \wedge (a\geq 0) \wedge (b>0)$ và

$P1 \wedge (a=bq+r) \wedge (q\geq 0) \wedge (r\geq 0) \wedge (r<b) \rightarrow Q1$

Theo các quy tắc điều kiện trước và điều kiện sau, chỉ cần chứng minh :

(I) $P1 \wedge (a\geq 0) \wedge (b>0)$ {Div} $P1 \wedge (a=bq+r) \wedge (q\geq 0) \wedge (r\geq 0) \wedge (r<b)$

Ta sẽ chứng minh hai giai đoạn :

1. P1 {Div} P1

2. $(a\geq 0) \wedge (b>0)$ {Div} $(a=bq+r) \wedge (q\geq 0) \wedge (r\geq 0) \wedge (r<b)$

❶ ***P1 {Div} P1***

Trước hết cần chỉ ra P1 là bất biến của vòng lặp. Theo quy tắc gán, ta có :

P1 { q :=q + 1 } P1

P1 { r := r – b } P1

như vậy, theo quy tắc tổ hợp :

P1 { r := r – b ; q := q + 1 } P1, nhưng :

$P1 \wedge (r\geq b) \rightarrow P1$

Theo quy tắc điều kiện trước :

$P1 \wedge (r\geq b)$ { r := r – b ; q :=q + 1 } P1

Theo quy tắc vòng lặp :

(1) P1 { while r≥b do begin r := r – b ; q :=q + 1 end } $P1 \wedge (r<b)$

Áp dụng lần nữa quy tắc gán và quy tắc tổ hợp, ta có :

(2) P1 { r := a ; q := 0 } P1

Từ (1) và (2), theo quy tắc tổ hợp, ta có :

P1 {Div} $P1 \wedge (r<b)$

Cuối cùng, theo quy tắc điều kiện sau :

P1 {Div} P1 (*đpcm*)

❷ ***$(a\geq 0) \wedge (b>0)$ {Div} $(a=bq+r) \wedge (q\geq 0) \wedge (r\geq 0) \wedge (r<b)$***

Đặt :

$P3 = (a=bq+r) \wedge (q\geq 0) \wedge (r\geq 0)$

$P4 = (a=b(q+1)+r) \wedge (q+1\geq 0) \wedge (r\geq 0)$ bằng cách thay q trong P3 bởi q+1

$P5 = (a=b(q+1)+r–b) \wedge (q+1\geq 0) \wedge (r–b \geq 0)$ bằng cách thay r trong P4 bởi r–b

(2.1.) Ta xem rằng P3 là một bất biến của vòng lặp. Theo quy tắc gán :

P4 { q := q + 1 } P3

và P5 { r := r – b } P4

Từ đó theo quy tắc tổ hợp :

(3) P5 { r := r – b ; q := q + 1 } P3

Xét điều kiện trước, ta có :

(i) P5 ~ (a=bq+r) ∧ (q≥ –1) ∧ (r≥b)

(ii) Vì rằng :

(q≥0) → (q≥ –1) và

(r≥0) ∧ (r≥b) → (r≥b), ta có :

P3 ∧ (r≥b) → (a=bq+r) ∧ (q≥ –1) ∧ (r≥b)

Theo quy tắc điều kiện trước, phát biểu (3) sẽ là :

P3 ∧ (r≥b) { r := r – b ; q := q + 1 } P3

Theo quy tắc vòng lặp dẫn đến :

(4) P3 { while r≥b do begin r := r – b ; q := q + 1 end } P3 ∧ (r<b)

(2.2.) Bây giờ sử dụng bất biến vòng lặp như là điều kiện sau cho những lệnh đầu tiên của dãy. Ta có quy tắc gán :

(a=b.0+r) ∧ (0≥0) ∧ (r≥0) { q := 0 } P3

và (a=b.0+r) ∧ (0≥0) ∧ (r≥0) ~ (a=r) ∧ (r≥0)

Áp dụng cho lần nữa quy tắc gán :

(a=a) ∧ (a≥0) { r := a } (a=r) ∧ (r≥0) và :

(a=a) ∧ (a≥0) ~ (a≥0)

Theo quy tắc tổ hợp, ta có :

(5) (a≥0) { r := a ; q := 0 } P3

Quy tắc tổ hợp áp dụng cho (4) và (5) cho ta :

(a≥0) {Div} P3 ∧ (r<b)

Quy tắc điều kiện sau dẫn đến kết luận :

(a≥0) ∧ (b>0) {Div} P3 ∧ (r<b)

Từ ❶ và ❷ đã chứng minh, suy ra (I) theo quy tắc "và", đồng thời kết thúc việc chứng minh tính đúng đắn từng phần của Div.

II.3. Chứng minh dừng

II.3.1. Chứng minh dừng của một chương trình

Một chương trình không là đệ quy, không chứa lệnh goto sẽ chỉ có thể một dãy vô hạn tính toán nếu nó thực hiện vô hạn lần thân một vòng lặp. Việc chứng minh một chương trình như vậy dừng với mọi dữ liệu d $\in$ D dẫn đến việc chứng minh rằng mỗi vòng lặp của chương trình chỉ có thể thực hiện một số hữu hạn lần $\forall$ d$\in$D.

Cho vòng lặp :

```
while B do P
```

với x_1, x_2, ..., x_n là các biến của chương trình. Gọi W_E là tập hợp các giá trị của vectơ :

$w = < x_1, x_2, ..., x_n >$

sao cho các biến thoả mãn điều kiện E.

Giả thiết rằng điều kiện E là bất biến với vòng lặp đang xét và thoả mãn trước khi thực hiện vòng lặp này.

Nếu $w = w_1 \in W_{E \wedge B}$ trước khi thực hiện vòng lặp, khi đó $w = w_2 \in W_E$ sau khi thực hiện thân vòng lặp P.

Nếu $w_2 \in W_{E \wedge \neg B}$, vòng lặp dừng ; nếu không $w = w_3 \in W_E$, sau khi thực hiện P, v.v...

Như vậy để chứng minh vòng lặp dừng, chỉ cần chỉ ra rằng mọi dãy w_1, w_2, w_3, ... đã xây dựng là hữu hạn.

Phương pháp để chứng minh là định nghĩa một hàm m từ $W_{E \wedge B}$ vào N sao cho có thể chỉ ra rằng :

$$E \wedge B \wedge (m(w) = m_0) \{ P \} \neg B \wedge (m(w) < m_0) \qquad \forall m_0 \in N$$

Chú ý :

Phát biểu trên có nghĩa rằng nếu $w = w_i$ trước khi thực hiện P, thì sau khi thực hiện P, hoặc vòng lặp dừng, hoặc $m(w_{i+1}) < m(w_i)$.

Trong trường hợp này (đã chứng minh được P dừng), ta có thể khẳng định rằng với mọi dãy w_1, w_2, w_3, ..., dãy $m(w_1)$, $m(w)_2$, $m(w)_3$ ... giảm dần thực sự trong N, là hữu hạn.

Điều đó chứng minh rằng dãy w_1, w_2, w_3, ... là hữu hạn, và vòng lặp dừng.

II.3.2. Chứng minh dừng của Div

Div chỉ chứa một vòng lặp :

while $r \geq b$ do begin r := r – b ; q := q +1 end

Vòng lặp này luôn luôn dừng.

Điều kiện { b=B } là một bất biến của vòng lặp này, được thoả mãn trước khi thực thể vòng lặp (xem chứng minh đúng từng phần mục ❶).

Tương tự như vậy đối với điều kiện ($r \geq 0$) (xem ❷).

Ta xét tập hợp $W_1 = W_{(b = B) \wedge (r \geq 0) \wedge (r \geq b)}$ và hàm $m : W_1 \rightarrow N$ sao cho m(w) = r.

Chú ý :

Việc chọn hàm này dẫn việc chứng minh dừng đến việc chứng minh rằng biến r nhận các giá trị giảm ngặt và liên tục.

Đặt :

$P6 = (b=B) \wedge (r \geq 0) \wedge (r \geq b)$

Tiếp theo, ta còn phải chứng minh rằng :

(1) $P6 \wedge (r=m_0)$ { r := r – b ; q := q + 1 } $(r<b) \wedge (r<m_0)$ $\forall\ m_0 \in N$

Theo quy tắc gán :

$(r-b < m_0)$ { r := r – b ; q := q + 1 } $(r<m_0)$ $\forall\ m_0 \in N$

áp dụng các quy tắc điều kiện trước và điều kiện sau, ta có :

(2) $P6 \wedge (r=m_0) \wedge (r-b<m_0)$ {r := r–b ; q := q+1} $(r<b) \wedge (r<m_0)$ $\forall\ m_0 \in N$

Mặt khác :

$(b=B) \wedge (r=m_0) \wedge (r-b<m_0) \sim (b=B) \wedge (r=m_0) \wedge (m_0-B<m_0)$

hay :

$(b=B) \wedge (r=m_0) \wedge (r-b<m_0) \sim (b=B) \wedge (r=m_0) \wedge (B>0)$

Điều kiện B>0 luôn luôn đúng vì rằng $(A, B) \in D = N \times N^+$, như vậy :

$(b=B) \wedge (r=m_0) \wedge (r-b<m_0) \sim (b=B) \wedge (r=m_0)$

Từ đó dẫn đến (1) xuất phát từ (2). Việc chứng minh Div dừng đã xong.

Chú ý :

Để chứng minh Div dừng với $\forall\ d \in D$, nói chung không áp dụng được cho $\forall d \in L$.

Ví dụ nếu $L = Z \times Z$, Div có thể quẩn với B = 0.

II.3.3. Đánh giá một chương trình lặp

Đánh giá một chương trình P là xác định thời gian và kích thước bộ nhớ cần thiết để thực hiện chương trình P :

- một hàm $T_P : L \to R \cup \{+\infty\}$ sao cho, $\forall\ d \in L$, $T_P(d)$ là thời gian thực hiện P đối với dữ liệu d ($T_P(d) = +\infty$ nếu chương trình quẩn).
- một hàm $N_P : L \to N \cup \{+\infty\}$ sao cho, $\forall\ d \in L$, $N_P(d)$ là số đơn vị bộ nhớ cần thiết để thực hiện P đối với dữ liệu d ($N_P(d)$ có thể vô hạn nếu chương trình quẩn).

Ở đây, ta chỉ quan tâm đến việc xác định thời gian thực hiện chương trình gồm các lệnh cơ sở như lệnh gán (quy tắc ";"), lệnh điều kiện và vòng lặp while.

Ta sẽ gọi f_P là hàm riêng phần (partial function) của L trong tập hợp các kết quả R, được tính bởi chương trình P, và W là tập hợp W_{true} các giá trị có thể của các biến trong P.

Hàm f_P có thể mở rộng thành một hàm $W \to W$. Tương tự, nếu P' là một dãy các lệnh của P, hàm $f_P$dc tính bởi P có thể được định nghĩa như là một hàm của

$W \to W$.

Ví dụ 14 :

Trong trường hợp Div, nếu L = D, $W = N \times N^+ \times N \times N$ = tập hợp các giá trị có thể của a, b, r, q.

f_{Div} (a, b, r, q) = (a, b, thương của a và b, phần dư của a chia cho b).

Nếu xét lệnh q := q + 1 của Div :

$f_{q := q+1}$ (a, b, r, q) = (a, b, q+1, r).

Thời gian thực hiện một chương trình có thể được định nghĩa như sau :

1. Mỗi lệnh cơ sở và mỗi điều kiện có một thời gian thực hiện độc lập với giá trị của các biến. Như vậy, $\forall\ w \in W$:

   Với mọi phép gán x := y, $T_{x := y}(w)$ = constant. Ta viết $T_{x := y}$.

   Với mọi điều kiện B, $T_B(w)$ = constant. Ta viết T_B.

2. $T_{P\,;\,Q}(w) = T_P(w) + T_Q(f_p(w))$

3. $T_{\text{if B then P}}(w) = \text{if B then } (T_B + T_P(w)) \text{ else } T_B$

   $T_{\text{if B then P else Q}}(w) = \text{if B then } (T_B + T_P(w)) \text{ else } (T_B + T_Q(w))$

4. $T_{\text{while B do P}}(w) = (n(w) + 1)\, T_B + \sum_{i=0}^{n(w)-1} T_P(f_P^i(w))$

trong đó, n là hàm $W \to N \cup \{ +\infty \}$, sao cho n(w) là số lần thực hiện thân vòng lặp P với giá trị đầu w của các biến.

Đánh giá thời gian thực hiện của Div :

Số lần lặp của Div là giá trị cuối của biến q, giả sử là $\left\lfloor \frac{A}{B} \right\rfloor$, phần nguyên của A chia cho B, trong đó, A và B là các dữ liệu của Div.

Mặt khác, thời gian thực hiện thân vòng lặp độc lập với các dữ liệu và bằng :

$T_{r := r - b} + T_{q := q + 1}$

Ta có :

$$T_{Div}(A, B) = T_{r := a} + T_{q := 0} + \left(\left\lfloor \frac{A}{B} \right\rfloor + 1\right) T_{r \geq b} + \left\lfloor \frac{A}{B} \right\rfloor (T_{r := r - b} + T_{q := q + 1})$$

Như vậy, $T_{Div}(A, B)$ có dạng $K_1 \left\lfloor \frac{A}{B} \right\rfloor + K_2$, trong đó K_1 và K_2 độc lập với các dữ liệu A và B.

$T_{Div}(A, B)$ là bậc (order) của $\left\lfloor \frac{A}{B} \right\rfloor$.

Chú ý : Người ta thường quan tâm đến bậc của hàm TP thay vì bản thân hàm TP, vì TP phụ thuộc vào phần cứng và phần mềm sử dụng để thực thi chương trình P. Như vậy chỉ cần xác định số lần thực hiện dãy các lệnh hay thực hiện nhất của chương trình. Điều này có thể nhận được bằng cách đếm số lần lặp trong mỗi vòng lặp.

III. Xây dựng chương trình

III.1. Mở đầu

Trong mục trước, ta thấy việc chứng minh tính đúng đắn của một chương trình, dù chỉ là một chương trình rất ngắn và đơn giản, rất khó khăn và mệt mỏi. Để khắc phục, người ta đưa ra một phương pháp hiệu quả hơn là *vừa thiết kế vừa chứng minh tính đúng đắn của chương trình*.

Ví dụ, đặc tả một bất biến cho một vòng lặp trước khi viết, kiểm tra tính dừng của vòng lặp này với điều kiện sau, rồi viết thân của vòng lặp sao cho giữ được bất biến và chỉ thực hiện một số hữu hạn lần. Để tính ƯSCLN của hai số nguyên dương, ta sử dụng các tính chất số học như sau :

Nếu a = b ƯSCLN(a, b) = a = b
Nếu a < b ƯSCLN(a, b) = ƯSCLN(a, b – a)
Nếu a > b ƯSCLN(a, b) = ƯSCLN(a – b, b)

Giả sử a và b là hai biến của chương trình nhận giá trị hai dữ liệu lần lượt A và B. Sau khi gán, điều kiện ƯSCLN(a, b) = ƯSCLN(A, B) thoả mãn.

Nếu thân chương trình chứa một vòng lặp của bất biến ƯSCLN(a, b) = ƯSCLN(A, B) với điều kiện kiểm tra dừng là a = b, thì ta có :

ƯSCLN(a, b) = a = b = ƯSCLN(A, B)

sau khi thực hiện vòng lặp này. Kết quả tìm được là một trong hai biến a và b.

Cần xác định thân của vòng lặp này sao cho, một mặt, bất biến được thoả mãn, mặt khác, vòng lặp chỉ thực hiện một số hữu hạn lần. Theo tính chất của ƯSCLN, nếu a ≠ b, lệnh :

```
if a > b then a := a – b else b := b – a
```

làm cho điều kiện ƯSCLN(a, b) = ƯSCLN(A, B) trở nên bất biến. Chỉ còn phải chứng minh rằng vòng lặp đã viết chỉ thực hiện một số hữu hạn lần.

Ta thấy quan hệ : $(a > 0) \wedge (b > 0)$

là bất biến trong vòng lặp (điều kiện của vòng lặp là a ≠ b, lệnh b := b – a chỉ được thực hiện nếu b > a).

Vậy max(a, b) cũng là bất biến. Hơn nữa max(a, b) giảm ngặt mỗi lần thực hiện thân vòng lặp, và vòng lặp luôn luôn dừng.

Như vậy ta đã viết và chứng minh không hình thức chương trình sau đây :

```
P : while a <> b do
      if a > b then a := a – b else b := b – a ;
      ketqua := a ;
```

III.2. Bài toán cờ tam tài

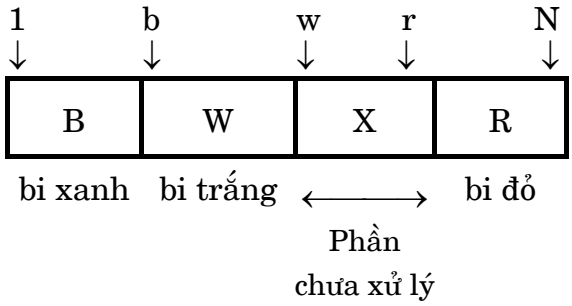
Cho trước :

1. Một mảng các hòn bi đánh số từ 1 đến N, mỗi hòn bi mang một màu hoặc xanh, hoặc trắng, hoặc đỏ.
2. Các vị từ B(i), W(i) và R(i) là đúng nếu và chỉ nếu hòn bi thứ i ($1 \le i \le N$) là xanh, trắng và đỏ tương ứng.
3. Cặp hoán vị (i, j) để đặt hòn bi thứ i thành j, hòn bi thứ j thành i, $\forall$ i, j $\in$ 1..N, không loại trừ trường hợp i = j.

Ta cần sắp xếp các hòn bi theo thứ tự "xanh, trắng, đỏ", mỗi vị từ B, W và R chỉ được tính đến một1 lần cho mỗi hòn bi đã cho. Hơn nữa, các hoán vị phải càng ít càng tốt.

III.2.1. Lời giải thứ nhất

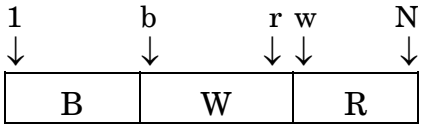
Ta sẽ sử dụng một vòng lặp while với bất biến như hình dưới đây và điều kiện kiểm tra dừng là "vùng X rỗng".



Cần có 3 chỉ số b, w và r để phân cách 4 vùng của mảng. Ta thực hiện phép chọn như sau : các chỉ số b và w đứng ngay sau các vùng B và W tương ứng của các viên bi. Chỉ số r là của viên bi đứng ngay trước vùng R.

Ta có : $P_{b, w, r} = (1 \le \alpha < b \rightarrow B(\alpha)) \wedge (b \le \alpha < w \rightarrow W(\alpha)) \wedge (r < \alpha \le N \rightarrow R(\alpha))$

Điều kiện sau của chương trình như sau :



hay có thể viết : $P_{b, w, r} \wedge (w = r + 1)$. Chương trình sẽ có dạng như sau :

{ Khởi gán : $P_{b, w, r}$ true }

while (vùng_X_không_rỗng) **do**

{ Thân vòng lặp của bất biến $P_{b, w, r}$ }

Sau khi thực hiện chương trình này, ta kiểm tra được rằng : $P_{b, w, r} \wedge$ (vùng_X_rỗng)
và mảng các viên bi đã được sắp xếp.

Chương trình sau đây chứa biến n nhận giá trị N :

```
        w:= 1; b := 1; r := n;
        while w <= r do
(I)        if W(w) then w:= w+1
           else if B(w) then
              begin HoánVị(b, w); b:= b+1; w:= w+1 end
              else
              begin HoánVị(r, w); r:= r-1 end
```

Ta nhận thấy rằng chương trình này thoả mãn các vị từ B, W và R.

Phân tích :

Số lần lặp = N

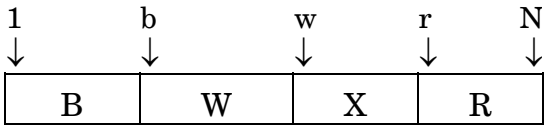
Số lần hoán vị = ne_1 = #B + #R (#B và #R lần lượt là số viên bi xanh và đỏ)

Kết quả này chưa là tối ưu, ta có thể làm giảm số lần hoán vị các viên bi đỏ như sau :

```
(I^bis)   w:= 1; b := 1; r := n;
          while w <= r do
            if W(w) then w:= w+1
            else if B(w) then
              begin HoánVị(b, w); b:= b+1; w:= w+1 end
              else
              begin while R(r) and w < r then r:= r-1;
                    HoánVị(r, w); r:= r-1
              end
```

III.2.2. Lời giải thứ hai

Ta sử dụng bất biến biểu diễn như sau :



Từ đó đưa đến chương trình (tạm thời chấp nhận việc chứng minh không hình thức) :

```
(II)      w:= 1; b := 1; r := n;
          while r <= n do
            if R(r) then r:= r-1
            else if W(r) then
              begin HoánVị(w, r); w:= w+1; r:= r-1 end
              else
              begin HoánVị(w, r); r:= r-1;
                    HoánVị(b, w); b:= b+1; w:= w+1
              end
```

Phân tích :

Số lần lặp = N

Số lần hoán vị = ne_2 = #W + 2#B

So sánh với chương trình (I), ta có : $ne_2 - ne_1$ = #B + #W – #R

Chương trình (I) sẽ hiệu quả hơn (II), nếu như ít ra nửa số viên bi là đỏ.

Bài tập :

Từ bất biến trên đây, ta có thể nhận được chương trình như sau :

```
        w:= 1; b := 1; r := n;
        while r <= n do
           if R(r) then r:= r-1
(II^bis)   else if W(r) then
              begin HoánVị(w, r); w:= w+1; r:= r+1 end
              else
              begin HoánVị(b, r); b:= b+1;
                    HoánVị(w, r); w:= w+1; r:= r+1
              end
```

Chương trình trên là sai. Hãy tìm một cách phần bố ban đầu của mảng hai viên bi để thấy sai.

III.2.3. Chứng minh tính đúng đắn của chương trình (I)

a) Các định nghĩa và ký hiệu

Ta đã xây dựng chương trình (I) bằng cách sử dụng bất biến :

$P_{b, w, r} = (1 \le \alpha < b \rightarrow B(\alpha)) \wedge (b \le \alpha < w \rightarrow W(\alpha)) \wedge (r < \alpha \le N \rightarrow R(\alpha))$

Để chứng minh tính đúng đắn của chương trình này, ta sử dụng ký hiệu đơn giản hoá :

$x \le \alpha < y \rightarrow A(\alpha)$ là ký hiệu viết tắt của điều kiện :

$$\begin{cases} A(x) \wedge A(x+1) \wedge \ldots \wedge A(y) & \text{nếu } x \le y \\ \text{true} & \text{nếu không} \end{cases}$$

hay : $(x \le y) \rightarrow A(x) \wedge \ldots \wedge A(y)$

Bây giờ ta xem xét các điều kiện đặc trưng của bài toán .

a. Các điều kiện (1) và (2) được biểu diễn bởi :

$E_1 = (1 \le \alpha \le N) \rightarrow (B(\alpha) \vee W(\alpha) \vee R(\alpha))$

b. Định nghĩa (3) của hoán vị (i, j) đưa đến tiên đề sau đây :

"Cho điều kiện sau S, ta có :

$E \wedge (1 \le i \le N) \wedge (1 \le j \le N)$ { hoán vị (i, j) } S

nếu E nhận được từ S bằng cách thay thế :

– mọi vị từ B(bt), W(bt), R(bt), trong đó bt có giá trị i, bởi B(j), W(j), R(j)

– và mọi vị từ B(bt), W(bt), R(bt), trong đó bt có giá trị j, bởi B(i), W(i), R(i)".

c. Điều kiện sau của chương trình (sắp xếp mảng) được biểu diễn bởi :

$$S_1 = (1 \le b \le N+1) \wedge \quad (b-1 \le r \le N) \wedge (1 \le \alpha < b \rightarrow B(\alpha)) \wedge (b \le \alpha < r \rightarrow W(\alpha)) \wedge (r < \alpha \le N \rightarrow R(\alpha))$$

Bất biến của vòng lặp ta vừa sử dụng không hẳn là $P_{b, w, r}$ nhưng :

$$Q_{b,\,w,\,r} = E_1 \wedge (1 \le b \le N) \wedge (w-1 \le r \le N) \wedge P_{b,\,w,\,r}$$

Giả sử A_1 là vòng lặp while của chương trình (I) và A_2 là thân của vòng lặp này.

b) Chứng minh tính đúng đắn từng phần

Bổ đề : $x \le \alpha \le y \to A(\alpha)$ tương đương với $(x \le \alpha \le y-1 \to A(\alpha)) \wedge (x \le y \to A(y))$

Chứng minh :

$x \le \alpha \le y \to A(\alpha)$

$\sim (x \le y) \to A(x) \wedge \ldots \wedge A(y)$ theo định nghĩa

$\sim ((x < y) \to A(x) \wedge \ldots \wedge A(y)) \wedge ((x = y) \to A(y))$

$\sim ((x < y) \to A(x) \wedge \ldots \wedge A(y-1)) \wedge ((x < y) \to A(y)) \wedge ((x = y) \to A(y))$

$\sim x \le \alpha \le y-1 \to A(\alpha)) \wedge (x \le y \to A(y))$ đpcm

(α) Chứng minh của bất biến $Q_{b,\,w,\,r}$ trong A_2

Ta kiểm tra rằng $E_1 \wedge (1 \le i \le N) \wedge (1 \le j \le N)$ { hoán vị (i, j) } E_1

(i) Trường hợp W(w)

Rõ ràng ta có : $Q_{b,\,w+1,\,r}$ { w := w+1 } $Q_{b,\,w,\,r}$

Vả lại : $Q_{b,\,w,\,r} \wedge (w \le r) \wedge W(w) \sim E_1 \wedge (1 \le b \le w) \wedge (w \le r \le N) \wedge P_{b,\,w,\,r} \wedge W(w)$

và : $P_{b,\,w,\,r} \wedge W(w) \to P_{b,\,w+1,\,r}$

như vậy : $Q_{b,\,w,\,r} \wedge (w \le r) \wedge W(w) \to Q_{b,\,w+1,\,r}$

Điều này chứng minh : $\boxed{Q_{b,\,w,\,r} \wedge (w \le r) \wedge W(w) \{ w := w+1 \} Q_{b,\,w,\,r}}$

(ii) Trường hợp B(w)

Rõ ràng ta có : $Q_{b+1,\,w+1,\,r}$ { b := b+1 ; w := w+1 } $Q_{b,\,w,\,r}$

Vả lại :

$$P_{b+1,\,w+1,\,r} = (1 \le \alpha < b+1 \to B(\alpha)) \wedge (b+1 \le \alpha < w+1 \to W(\alpha)) \wedge (r \le \alpha < N \to R(\alpha))$$

Từ đó theo bổ đề :

$$P_{b+1,\,w+1,\,r} \sim (1 \le \alpha < b+1 \to B(\alpha)) \wedge (b+1 \le \alpha < w \to W(\alpha)) \wedge (r \le \alpha < N \to R(\alpha)) \wedge ((b \ge 1) \to B(b)) \wedge ((w \ge b+1) \to W(\alpha))$$

Áp dụng tiên đề định nghĩa bởi phép hoán vị cho $Q_{b+1,\,w+1,\,r}$:

$$F \{ \text{hoán vị}(b, w) \} Q_{b+1,\,w+1,\,r}$$

với :

$F = E_1 \wedge (1 \le b+1 \le w+1) \wedge (w \le r \le N)$

$\wedge (1 \le \alpha < b \rightarrow B(\alpha)) \wedge (b+1 \le \alpha < w \rightarrow W(\alpha))$

$\wedge (r < \alpha \le N \rightarrow R(\alpha)) \wedge ((b \ge 1) \rightarrow B(w))$

$\wedge ((w \ge b+1) \rightarrow W(b)) \wedge (1 \le b \le N) \wedge (1 \le w \le N)$

Nhưng :

$(b+1 \le \alpha < w \rightarrow W(\alpha)) \wedge ((w \ge b+1) \rightarrow W(b)) \sim (b \le \alpha < w \rightarrow W(\alpha))$

Như vậy :

$F \sim E_1 \wedge (1 \le b \le w) \wedge (w \le r \le N) \wedge B(w) \wedge P_{b, w, r}$

Từ đó suy ra :

$Q_{b, w, r} (w \le r) \wedge \neg W(w) \wedge B(w) \rightarrow F$

Điều này chứng minh :

$$Q_{b, w, r} \wedge (w \le r) \wedge \neg W(w) \wedge B(w) \{ \text{hoánvị}(b, w) ; b := b+1 ; w := w+1 \} Q_{b, w, r}$$

(iii) Trường hợp R(w)

Ta có :

$Q_{b, w, r-1} \{ r := r -1 \} Q_{b, w, r}$

Mặt khác theo bổ đề :

$P_{b, w, r-1} \sim P_{b, w, r} \wedge ((r \le N) \rightarrow R(r))$

Áp dụng tiên đề định nghĩa bởi phép hoán vị cho $Q_{b, w, r-1}$:

$$G \{ \text{hoánvị}(r, w) \} Q_{b, w, r-1}$$

với :

$G = E_1 \wedge (1 \le b \le w) \wedge (w-1 \le r-1 \le N) \wedge P_{b, w, r} \wedge ((r \le N) \rightarrow R(w))$
$\wedge (1 \le r \le N) \wedge (1 \le w \le N)$

Thực tế ta có :

$P_{b, w, r} \wedge (b \le w) \wedge (w \le r) \wedge (1 \le w \le N) \wedge (1 \le r \le N)$

$$\{ \text{hoánvị}(r, w) \} P_{b, w, r} \wedge (b \le w) \wedge (w \le r)$$

Ta có :

$G \sim E_1 \wedge (1 \le b \le w) \wedge (w \le r \le N) \wedge R(w) \wedge P_{b, w, r}$

Vả lại :

$E_1 \wedge \neg W(w) \wedge \neg B(w) \wedge (1 \le w \le N) \rightarrow R(w)$

Như vậy :

$Q_{b, w, r} \wedge (w \le r) \wedge \neg W(w) \wedge \neg B(w) \rightarrow G$

Điều này chứng minh :

$$Q_{b, w, r} \wedge (w \le r) \wedge \neg W(w) \wedge \neg B(w) \{ \text{hoánvị}(r, w) ; r := r - 1 \} Q_{b, w, r}$$

Từ 3 kết quả trên ta suy ra :

$$Q_{b, w, r} \wedge (w \le r) \{ A_2 \} Q_{b, w, r}$$

(β) Chứng minh tính đúng đắn của chương trình (I)

Ta đã có :

$$Q_{b, w, r} \{ A_2 \} Q_{b, w, r} \wedge (w > r)$$

(i) Điều kiện sau

Ta có :

$$Q_{b, w, r} \wedge (w > r) \rightarrow (1 \le b \le w) \wedge (w=r+1) \wedge (r \le N) \wedge P_{b, w, r}$$

và :

$P_{b, w, r} \wedge (w=r+1) \rightarrow (1 \le \alpha < b \rightarrow B(\alpha)) \wedge (b \le \alpha < r \rightarrow W(\alpha)) \wedge (r < \alpha \le N \rightarrow R(\alpha))$

Như vậy :

$$Q_{b, w, r} \wedge (w > r) \rightarrow S_1$$

trong đó :

$$Q_{b, w, r} \{ A_1 \} S_1$$

(ii) Khởi gán

Ta có :

$$E_1 \wedge (0 \le n \le N) \wedge (n < \alpha \le N \rightarrow R(\alpha)) \{ w := 1; b := 1; r := n \} Q_{b, w, r}$$

vả lại :

$$(n=N) \rightarrow (0 \le n \le N) \wedge (n < \alpha \le N \rightarrow R(\alpha)),$$

do đó :

$$E_1 \wedge (n=N) \{ \text{chương trình (I)} \} S_1$$

Như vậy ta đã chứng minh xong tính đúng đắn từng phần.

c) Chứng minh dừng

Chương trình (I) chỉ có một vòng lặp A_1 và thân của vòng lặp là A_2.

Giả sử : $W_{Q_{bwr} \wedge (w \le r)} = X_1$ và hàm $m : X_1 \rightarrow N$ sao cho $m(x) = r - w$.

Chú ý : Việc chứng minh dừng là chỉ ra rằng kích thước của vùng chưa xử lý r – w giảm dần mỗi lần thực hiện A_2.

Để chứng minh việc dừng của A_1, chỉ cần chỉ ra rằng :

$$Q_{b, w, r} \wedge (w \le r) \wedge (r - w = m_0) \{ A_2 \} (w > r) \vee (r - w < m_0) \quad \forall m_0 \in N$$

Chú ý rằng :

$(r - w = m_0) \{ w := w+1 \}$ $(r - w = m_0 - 1)$ trường hợp của W(w) và B(w)

và $(r - w = m_0) \{ r := r - 1 \}$ $(r - w = m_0 - 1)$ trường hợp của R(w)

Ta chứng minh được rằng :

$$Q_{b, w, r} \wedge (w \le r) \wedge (r - w = m_0) \{ A_2 \} (r - w = m_0 - 1) \quad \forall m_0 \in N$$

Từ đó chương trình (I) dừng.

d) Chứng minh chương trình (I) thoả mãn điều kiện của bài toán

Bây giờ chỉ còn phải chứng minh rằng điều kiện "mỗi vị từ B, W và R chỉ được xét tính nhiều nhất một lần cho mỗi viên bi" chưa xuất hiện trong S_1 cũng thoả mãn.

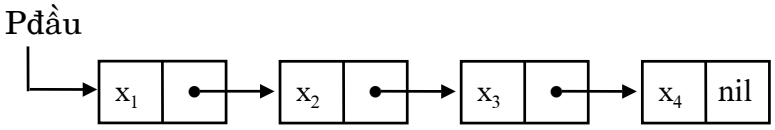
Thật vậy, ở trên, ta đã suy ra được rằng A_2 được thực hiện đúng N lần (giá trị của r –w giảm 1 mỗi lần, giảm từ N–1 đến 0). Mặt khác, việc thực hiện A_2 xét tính (nhiều nhất một lần) các vị từ liên quan đến chỉ một viên bi. Trong quá trình sắp xếp, màu của N viên bi là hoàn toàn xác định và vì có N lần thực hiện, mỗi thực hiện của A_2 chỉ liên quan đến 1 viên bi phân biệt.

III.3. In ra một danh sách theo thứ tự ngược

Giả sử ta có một danh sách tuyến tính gồm các phần tử có cấu trúc và một con trỏ chỉ đến phần tử đầu tiên của danh sách như sau :

```
const n = ...                { độ dài lớn nhất của danh sách }
type  KiểuPhầnTử = ...       { kiểu của các phần tử trong danh sách }
KiểuConTrỏ = 0..n;           { kiểu của con trỏ, 0 là giá trị nil }
PhầnTử = record
        NộiDung: KiểuPhầnTử;
        TiếpTheo: KiểuConTrỏ; {trỏ đến phần tử tiếp theo }
        end;
Danhsách = array[1..n] of PhầnTử;
var  Ds : Danhsách;          { danh sách các phần tử }
     Pđầu : KiểuConTrỏ;      { con trỏ chỉ đến phần tử đầu tiên của danh
     sách }
```

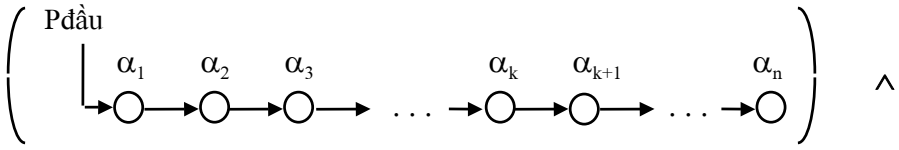
Giả thiết rằng danh sách chứa ít nhất một phần tử (đầu danh sách **PĐầu** ≠ **nil**). Bài toán đặt ra là in nội dung danh sách theo thứ tự ngược lại. Ví dụ, nếu ta có :



thì chương trình sẽ in ra như sau : x_4, x_3, x_2, x_1.

Ta sẽ đưa ra ba lời giải TILDA1, TILDA2 và TILDA3 cho danh sách có độ dài n phần tử. Đối với mỗi lời giải, ta sẽ phân tích thời gian và bộ nhớ cần thiết để thực hiện chương trình.

Cả ba lời giải đều sử dụng vòng lặp với bất biến có dạng sau :



(các trường nội dung các bản ghi α_{k+1}, ..., α_n được in theo chiều ngược lại)

Mỗi bước của vòng lặp là tìm phần tử α_k và viết nội dung (α_k). Ba chương trình phân biệt nhau cơ bản ở cách tiếp cận đến α_k.

III.3.1.TILDA1

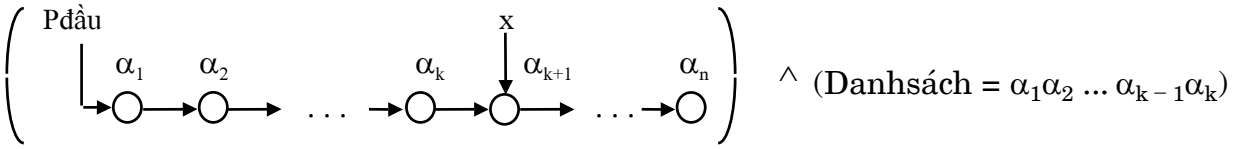
Ta sử dụng một danh sách con trỏ, đầu tiên danh sách ở trạng thái rỗng, sau đó hoạt động với hai thao tác như sau :

put(expr) đặt vào đỉnh danh sách giá trị của biểu thức expr.

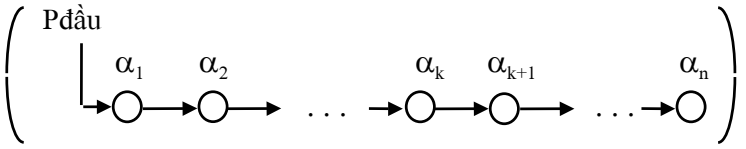
get(x) nếu danh sách khác rỗng, lấy giá trị ở đỉnh danh sách để gán cho biến x, nếu không, thao tác không xác định.

Chương trình TILDA1 gồm hai vòng lặp :

(a) Vòng lặp thứ nhất :



(b) Vòng lặp thứ hai :



$\wedge$ (Danhsách = $\alpha_1\alpha_2 \ldots \alpha_k$) (các trường nội dung các bản ghi α_{k+1}, ..., α_n đã được in theo chiều ngược lại)

Mỗi bước của vòng lặp là tìm phần tử α_k và viết nội dung (α_k). Ba chương trình phân biệt nhau cơ bản ở cách tiếp cận đến α_k.

IV.Các tiên đề và quy tắc suy diễn

Mục này sẽ nghiên cứu các vấn đề sau :

- – Khái niệm về điều kiện trước yếu nhất và điều kiện sau mạnh nhất của một dãy lệnh.
- – Các kiểu tiên đề gán khác nhau.
- – Các tiên đề và quy tắc suy diễn cho một số cấu trúc ngôn ngữ lập trình (khối, thủ tục).
- – Phân tích chương trình.

IV.1. Điều kiện trước yếu nhất và điều kiện sau mạnh nhất của một dãy lệnh

Trong mục trước, ta đã ký hiệu :

– W_E tập hợp các giá trị các biến của một chương trình thoả mãn điều kiện E,

– f_P là hàm tính được bởi dãy các lệnh P của một chương trình.

Nếu f_P được định nghĩa cho mọi giá trị của W_E , P không quẩn và không thực hiện các phép tính vô định (ví dụ chia cho 0) nếu điều kiện trước E thoả mãn. Tính chất này được ký hiệu là $term_EP$.

Bây giờ ta xét phát biểu E {P} S. Ta có các tính chất sau đây :

(i) E {P} S là true nếu và chỉ nếu $f_P(W_E) \subseteq W_S$

(ii) Nếu $term_EP$ thì :

E {P} S là true nếu và chỉ nếu $W_E \subseteq f_P^{-1}(W_S)$

Ví dụ 15 :

(1) Cho phát biểu (q > 0) { q := q+1 } (q > 0), trong đó q là biến duy nhất của chương trình. Ta có :

$$W_E = W_S = N^+ , \quad f_{q := q+1}(N^+) = N^+ - \{1\} \subseteq N^+ ,$$

và $N^+ \subseteq f^{-1}_{q:=q+1}(N^+) = N$

(2) Cho phát biểu $(q \geq 0) \wedge (y \geq 0)$ { q := q div y } $(q \geq 0) \wedge (y \geq 0)$, trong đó q và y là các biến duy nhất của chương trình. Ta có :

$$W_E = W_S = N \times N , \qquad f_{q:=q \text{ div } y}(N \times N) = N \times N^+ \subset W_S ,$$

nhưng $$f^{-1}_{q:=q \text{ div } y}(N \times N) = N \times N^+ \not\supseteq W_E$$

Trong trường hợp $W_E = f_P^{-1}(W_S)$, E là *điều kiện trước yếu nhất* (la plus faible précondition) phải được thoả mãn trước khi thực hiện P để cho S được thoả mãn sau đó.

Thực tế, nếu E' {P} S và $term_{E'}$ P thì $W_{E'} \subseteq f_P^{-1}(W_S) = W_E$ và như vậy E' → E.

Mặt khác, nếu $f_P(W_E) = W_S$, S là *điều kiện sau mạnh nhất* (la plus forte postcondition) phải được thoả mãn sau khi thực hiện P nếu E là đúng trước.

Thực tế, nếu E {P} S' thì $f_P(W_E) = W_S \subseteq W_{S'}$, như vậy S → S'.

Ta ký hiệu hai hàm *fppre* và *fppost* như sau :

– với một dãy lệnh và với một điều kiện sau, *fppre* trả về điều kiện trước yếu nhất tương ứng

– với một điều kiện trước và một dãy lệnh, *fppost* trả về điều kiện sau sau mạnh nhất tương ứng.

Chú ý rằng các hàm *fppre(P, S)* và *fppost(E, P)* được định nghĩa gần như tương đương.

Bây giờ ta sẽ trình bày các tính chất của các hàm fppre và fppost.

IV.1.1. Hàm fppre

Với mọi điều kiện S và dãy lệnh P, pfpost(pfpre(P, S), P) → S

Chứng minh :

Đặt E ~ pfpre(P, S).

Ta có $W_E = f_P^{-1}(W_S)$, từ đó suy ra $f_P(W_E) = f_P(f_P^{-1}(W_S)) \subseteq W_S$

Hay có thể nói E ~ pfpre(P, S), từ đó suy ra pfpost(E, P) → $S_{đpcm}$

IV.1.2. Hàm fppost

Với mọi điều kiện E và dãy lệnh P, E → pfpre(P, pfpost(E, P)).

Chứng minh :

Đặt S ~ pfpost(E, P).

Ta có $W_S = f_P(W_E)$, từ đó suy ra $f_P^{-1}(W_S) = f_P^{-1}(f_P(W_E)) \supseteq W_E$

Hay có thể nói S ~ pfpost(E, P), từ đó suy ra E → pfpre(P, S) $_{đpcm}$

Bài tập :

1. Cho điều kiện E và một dãy lệnh P, những điều kiện nào làm thoả mãn f_P sao cho :

$$E \sim pfpre(P, pfpost(E, P)) ?$$

IV.1.3. Sử dụng điều kiện trước yếu nhất và điều kiện sau mạnh nhất để chứng minh tính đúng đắn của chương trình

a) Trường hợp điều kiện trước yếu nhất

Sau khi định nghĩa hàm pfpre, với mọi điều kiện E và S, và mọi dãy lệnh P, ta có :

$$(E \{P\} S \wedge term_E P) \sim (E \rightarrow pfpre(P, S))$$

đặc biệt :

$$term_E P \sim (E \rightarrow pfpre(P, true))$$

Điều này có nghĩa rằng tập hợp các giá trị của pfpre(P, true) là miền xác định của hàm f_P (nghĩa là tập hợp các giá trị sao cho chương trình P đưa ra một kết quả).

Khả năng biểu diễn pfpre(P, S), ∀ P và ∀ S, cho phép chứng minh tính đúng đắn (CMTĐĐ) toàn cục của chương trình.

Một hệ thống CMTĐĐ toàn cục sử dụng điều kiện trước yếu nhất bao gồm :

(a) Với mỗi lệnh sơ cấp, các tiên đề cho điều kiện trước yếu nhất ứng với một điều kiện sau đã cho. Đối với phép gán, các tiên đề đã cho ở mục I nói chung sẽ thay thế vai trò này.

(b) Các quy tắc suy diễn cho phép xây dựng điều kiện trước yếu nhất của một lệnh không sơ cấp P, xuất phát từ các điều kiện trước yếu nhất của các lệnh trong P.

Ví dụ 16 :

Giả sử cần xác định : **`pfpre(if B then P else Q, S)`**

Nếu B cho một kết quả, ta có :

`pfpre(if B then P else Q, S)`

`~ (B ∧ pfpre(P, S) ∨ (¬B ∧ pfpre(Q, S))`

Nhưng việc tính B để cho một kết quả nếu và chỉ nếu `pfpre(if B then , true)` là đúng.

Như vậy trong trường hợp tổng quát :

```
pfpre(if B then P else Q, S)
        ~if pfpre(if B then , true)
        then (B ∧ pfpre(P, S) ∨ (¬B ∧ pfpre(Q, S)) else false
```

Không có quy tắc đơn giản cho vòng lặp while.

Độ phức tạp của các quy tắc đưa ra để xác định dkt của một vòng lặp while có thể được minh hoạ như sau :

– Vấn đề dừng của vòng lặp while trừu tượng không là quyết định được.
Vì vậy, điều kiện

```
pfpre(while B do P, true)
```

không tính được cho mọi B, mọi P.

– Việc giải quyết một số bài toán nổi tiếng đưa về việc chứng minh một chương trình là dừng. Ví dụ, người ta không biết nếu :

```
pfpre(while n<>1 do
   n:=if not odd(n) then n div 2 else 3*n+1, true)~(n >= 1)
```

Đây là sự giả định (conjecture) của Collatz.

Chú ý :Các tiên đề và các quy tắc suy diễn xét ở mục I cho phép chứng minh tính đúng đắn của các phát biểu E {P} S, trong đó E ≠ pfpre(P, S). Ví dụ, ta đã chứng minh tính đúng đắn từng phần rằng :

$$(a \geq 0)\ \{Div\}\ (a = bq + r) \wedge (q \geq 0) \wedge (0 \leq r < b)$$

b) Trường hợp điều kiện sau mạnh nhất

Theo định nghĩa của pfpost, ta có : E {P} S ~ (pfpost(E, P) → S)

Như vậy, một hệ thống chứng minh tính đúng đắn dựa trên việc tính toán các điều kiện sau mạnh nhất cho phép chứng minh tính đúng đắn *từng phần*.

Đối với lệnh điều kiện, quy tắc suy diễn được cho bởi tính chất :

```
pfpost(E, if B then P else Q)
~ pfpost(E ∧ B, P) ∧ pfpost(E ∧ ¬B,Q)
```

với giả trị rằng B luôn luôn cho một kết quả.

Trong trường hợp tổng quát :

```
pfpost(E, if B then P else Q)
   ~ pfpost(E ∧ (if pfpre(if B then, true) then B else false), P)
     ∨ pfpost(E ∧ (if pfpre(if B then, true) then ¬B else
   false), Q)
```

IV.2. Các tiên đề gán

IV.2.1. Điều kiện trước yếu nhất và điều kiện sau mạnh nhất của lệnh gán

Điều kiện trước E được tính toán theo quy tắc đã trình bày ở mục I để tạo ra các tiên đề gán là điều kiện trước yếu nhất của một lệnh gán x := <bt> và của một điều kiện sau S, khi đại lượng $term_E$(x := <bt>) có giá trị true. Trong trường hợp này, tiên đề được ký hiệu bởi :

pfpre(x := <biểu thức>, S) { x := <bt> } S

Trong trường hợp tổng quát, ta có :

pfpre(x := <bt>, S) ~ if pfpre(x := <bt>, true) then S(x/<bt>) else false.

Điều kiện pfpre(x := <bt>, true) thể hiện :

– x và các toán hạng của <bt> được định nghĩa :
  chỉ số của mảng nằm giữa cận dưới và cận trên, các con trỏ chỉ đến các biến khác nil, v.v...,

– mọi phép toán thực thi đều cho kết quả :
  có sự tương thích về kiểu của các toán hạng, không có phép chia cho 0, v.v...

Ở đây ta không thấy sự vi phạm về tính hợp thức của các tiên đề đã nêu trong mục I về tính đúng đắn từng phần : nếu $term_E$(x := <bt>) có giá trị false, phát biểu E { x := <bt> } S là true. Người ta có thể củng cố điều kiện trước của các tiên đề này bởi các điều kiện do pfpre(x := <bt>, true) đưa đến.

Ví dụ, nếu m và n lần lượt là cận dưới và cận trên của mảng a, thay vì :

$$(y > 0) \quad \{ x := a[j] \} \quad (y > 0) \text{ và}$$
$$(a[j] = y) \quad \{ x := a[j] \} \quad (x = y)$$

Ta có thể viết : $(m \le j \le n) \wedge (y > 0) \{ x := a[j] \} (y > 0)$

$$\text{if } (m \le j \le n) \text{ then } (a[j] = y) \text{ else false} \quad \{ x := a[j] \} \quad (x = y)$$

Theo mệnh đề III.1.1, nếu E (x := <bt>) S là một tiên đề, S không nhất thiết là điều kiện sau mạnh nhất ứng với E. Như vậy các tiên đề gán nói chung không thể ký hiệu :

E (x := <bt>) pfpost(E, x := <bt>)

Chẳng hạn ta có tiên đề : $(\frac{1}{x} \ge 0) \{ x := 1/x \} (x \ge 0)$

Ta có : $(\frac{1}{x} \ge 0) \sim (x > 0)$, nhưngpfpost$(x > 0, x := 1/x) \sim (x > 0)$

Bây giờ ta sẽ xét các quy tắc xây dựng các tiên đề từ điều kiện trước về điều kiện sau. Các tiên đề này sẽ luôn luôn được ký hiệu bởi :

E (x := <bt>) pfpost(E, x := <bt>)

nhưng không thể ký hiệu bởi : pfpre(x := <bt>, S) { x := <bt> } S

IV.2.2. Quy tắc tính toán điều kiện sau mạnh nhất của một phép gán

Với lệnh gán x := <bt>, chỉ có biến x bị thay đổi. Như vậy, $f_{x := <bt>}$ là một hàm từ tập hợp W các giá trị của các biến của chương trình vào chính nó, đặt đồng nhất các thành phần, trừ biến x.

Ta ký hiệu $\pi_x f_{x := <bt>}$ là phép chiếu của $f_{x := <bt>}$ vào thành phần này.

Ví dụ, với lệnh q := q+1 của chương trình Div ở mục I, ta có :

$f_{q := q+1}(a, b, q, r) = (a, b, q+1, r)$và $\pi_q f_{q := q+1}(a, b, q, r) = q+1$

Tương tự, quy tắc điều kiện trước về tiên đề gán đã cho ở mục I có thể viết :

$$E \sim S(x / \pi_x f_{x := <bt>})$$

Bây giờ ta sẽ định nghĩa hai quy tắc tính toán điều kiện sau mạnh nhất của một lệnh gán.

a) Trường hợp đặc biệt

Nếu hàm thu hẹp $f_{x := <bt>}$ vào W_E là toàn cục và đơn ánh, sẽ tồn tại hàm ngược $f^{-1}_{x := <bt>}$

Trong trường hợp này, có thể sử dụng quy tắc sau đây để tính pfpost :

$$pfpost(E, x := <bt>) \sim E(x / \pi_x f^{-1}_{x := <bt>})$$

nghĩa l1 ta nhận được điều kiện sau mạnh nhất của một điều kiện trước E và một phép gán x := <bt> bởi việc thay thế những nơi x xuất hiện trong điều kiện E bởi hàm ngược của $f_{x := <bt>}$.

Từ quy tắc tính pfpost, ta có tiên đề gán từ điều kiện trước và điều kiện sau như sau :

$$E \{ x := <bt> \} E(x / \pi_x f^{-1}_{x := <bt>})$$

Sư tồn tại hàm ngược của $f^{-1}_{x := <bt>}$ cho phép tìm được giá trị của x trước khi thực hiện phép gán, từ các giá trị các biến sau khi gán.

Chú ý : $pfpost(q \geq 0, q := q+1) \sim (q-1 \geq 0)$

$pfpost(q \geq 0, q := q*2) \sim (\frac{q}{2} \geq 0)$

$pfpost(q = 0, q := q+a) \sim (q-a = 0)$

Ta nhận được các tiên đề sau đây :

$(q \geq 0) \{ q := q+1 \} (q-1 \geq 0)$

$(q \geq 0) \{ q := q*2 \} (\frac{x}{2} \geq 0)$

$(q = 0) \{ q := q+a \} (q-a = 0)$

Quy tắc trên không dùng được cho các phép gán như q := 0, hoặc q := q div 2.

Ta có thể dùng quy tắc để chứng minh chương trình theo chiều bắt đầu – kết thúc.

Ví dụ, để chứng minh $(x > 0)\ \{ x := x*2 \}\ (x > 0)$:

- Xét quy tắc từ điều kiện sau về điều kiện trước :
  $(2x > 0)\ \{ x := x*2 \}\ (x > 0)$

Sử dụng quy tắc điều kiện trước, ta nhận được kết quả vì :
$(2x > 0) \rightarrow (x > 0)$

- Xét quy tắc từ điều kiện trước về điều kiện sau :

  $(x > 0)\ \{ x := x*2 \}\ (\frac{x}{2} > 0)$

  Sử dụng quy tắc điều kiện sau, ta nhận được kết quả vì : $(\frac{x}{2} > 0) \rightarrow (x > 0)$

Chú ý : Các tiên đề gán đã đưa ra trong các ví dụ trên có thể nhận được từ điều kiện sau bởi tiên đề gán ở mục I. Trong các ví dụ này, hàm $f_{x := <bt>} : W \rightarrow W$ là đơn ánh.

Ví dụ sau đây chỉ ra rằng nếu hàm $f_{x := <bt>} : W \rightarrow W$ không là đơn ánh, người ta không còn có tính chất này.

Ví dụ 17 :

Hàm $f_{x := x \text{ div } 2}$ là toàn thể, nhưng không đơn ánh. Thu hẹp của nó vào $W_{x = 2y}$ là đơn ánh.

Áp dụng quy tắc tính pfpost, ta có tiên đề : $(x = 2y)\ \{ x := x \text{ div } 2 \}\ (2x = 2y)$

tương đương với : $(x = 2y)\ \{ x := x \text{ div } 2 \}\ (x = y)$

Tuy nhiên, xuất phát từ điều kiện sau $(2x = 2y)$ và sử dụng tiên đề gán ở mục I, ta có tiên đề :

$$(2\left\lfloor\frac{x}{2}\right\rfloor = 2y)\ \{ x := x \text{ div } 2 \}\ (2x = 2y)$$

tương đương với : $(2\left\lfloor\frac{x}{2}\right\rfloor = 2y)\ \{ x := x \text{ div } 2 \}\ (x = y)$

b) Trường hợp tổng quát

Nếu không tồn tại hàm ngược của $f_{x := <bt>}$, sẽ không thể tìm được giá trị ban đầu của x, xuất phát từ giá trị các biến sau khi thực hiện phép gán.

V. Bài tập

Bài 1 : Sử dụng các quy tắc trên đây, chứng minh các tính chất sau :

1. Nếu E {P} S và E' {P} S' là các định lý, thì E $\wedge$ E' {P} S $\wedge$ S' và E $\vee$ E' {P} S $\vee$ S' cũng là các định lý.
2. Nếu E {P} F và G {Q} S là các định lý và nếu F $\rightarrow$ G, thì E {P ; Q} S là một định lý.
3. Sử dụng quy tắc ";", chứng minh rằng : (y = 2) { x := y+1 ; z :=x+y } (z = 5)
4. Sử dụng quy tắc điều kiện trước, chứng minh rằng : (q $\geq$ 0) { q := q + 1 } (q $\geq$ 0)
5. Phép thế sử dụng trong quy tắc trên đây có thể được viết E = S (x/<bt>). Trong điều kiện đặc biệt của một biểu thức điều kiện trước, phép thế này được định nghĩa bởi :

   S (x/if a then e_1 else e_2) $\stackrel{dn}{=}$ (a $\wedge$ S (x/e_1)) $\vee$ ($\neg$a $\wedge$ S (x/e_2))

   Sử dụng quy tắc trên đây để chứng minh :

   (z $\geq$ 0) { z := if b > 0 then z + b else z – b } (z $\geq$ 0)

Bài 2 : Chứng minh Div dừng :

Bằng cách sử dụng hàm m'(w) = $\left\lfloor \frac{A}{B} \right\rfloor$ – q, trong đó $\left\lfloor \frac{A}{B} \right\rfloor$ là phần nguyên của phép chia $\frac{A}{B}$.

Chứng minh sau khi ra khỏi vòng lặp, m'(w) = 0.

Bài 3 : Từ chương trình P trong mục II.2.1, hãy :

1. Chứng minh hình thức chương trình P.
2. Cho biết số lần tối đa thực hiện vòng lặp của P.
3. Sử dụng chương trình Div của mục trước để suy diễn từ P một chương trình tính ƯSCLN của hai số nguyên dương A và B bởi việc tính số dư liên tiếp.

Bài 4 : Từ chương trình (I^{bis}) trong mục II.2.2, hãy :

1. Chứng minh rằng chương trình (I^{bis}) thoả mãn các ràng buộc của bài toán.
2. Chứng minh rằng chương trình (I^{bis}) không còn thoả mãn nếu thay thế vòng lặp trong cùng bởi :

```
while R(r) and (w<=r) do r:=r-1;
if w<r then begin hoánvị(r, w); r:=r-1 end;
```

Chứng minh rằng chương trình này có thể dẫn đến tính R(0).