Th.s. NGUYỄN VĂN LINH

NGÔN NGỮ LẬP TRÌNH

Được biên soạn trong khuôn khổ dự án ASVIET002CNTT
"Tăng cường hiệu quả đào tạo và năng lực tự đào tạo của sinh viên khoa Công nghệ Thông tin - Đại học Cần thơ"

ĐẠI HỌC CẦN THƠ - 12/2003

TỔNG QUAN

MỤC ĐÍCH YÊU CẦU

Mục đích của môn học Ngôn ngữ lập trình là cung cấp cho sinh viên một khối lượng kiến thức tương đối hoàn chỉnh về nguyên lí của ngôn ngữ lập trình. Cùng với môn học Tin học lí thuyết, Ngôn ngữ lập trình sẽ là môn học tiên quyết để học môn Trình biên dịch. Sau khi học xong môn học này, sinh viên cần:

- Nắm được các khái niệm về đối tượng dữ liệu và kiểu dữ liệu. Các khía cạnh cần nghiên cứu khi đặc tả và cài đặt một kiểu dữ liệu. Vấn đề kiểm tra kiểu và chuyển đổi kiểu cũng cần được quan tâm.
- Nắm được các kiểu dữ liệu sơ cấp và có cấu trúc. Với mỗi kiểu dữ liệu cần nắm định nghĩa, đặc tả và cách cài đặt kiểu dữ liệu.
- Nắm được khái niệm trừu tượng hoá trong lập trình thể hiện trên hai khía cạnh là trừu tượng hoá dữ liệu bằng cách sử dụng các kiểu dữ liệu tự định nghĩa và trừu tượng hoá chương trình bằng cách chia chương trình thành các chương trình con. Vấn đề truyền tham số cho chương trình con cũng cần được lưu tâm.
- Nắm được khái niệm điều khiển tuần tự, nguyên tắc điều khiển tuần tự trong biểu thức và giữa các câu lệnh.

ĐỐI TƯỢNG SỬ DỤNG

Môn học ngôn ngữ lập trình được dùng để giảng dạy cho các sinh viên năm thứ 4 chuyên ngành Tin học.

NỘI DUNG CỐT LÕI

Trong khuôn khổ 45 tiết, giáo trình được cấu trúc thành 9 chương

Chương 1: Mở đầu. Chương này trình bày khái niệm về ngôn ngữ lập trình, lợi ích của việc nghiên cứu ngôn ngữ lập trình và các tiêu chuẩn để đánh giá một ngôn ngữ lập trình tốt.

Chương 2: Kiểu dữ liệu. Chương này trình bày các khái niệm về đối tượng dữ liệu và kiểu dữ liệu; các phương pháp kiểm tra kiểu và chuyển đổi kiểu; Phép gán trị cho biến và sự khởi tạo biến.

Chương 3: Kiểu dữ liệu sơ cấp. Chương này trình bày khái niệm về kiểu dữ liệu sơ cấp, sự đặc tả và nguyên tắc cài đặt một kiểu dữ liệu sơ cấp nói chung. Phần chủ yếu của chương trình bày một số kiểu dữ liệu sơ cấp phổ biến như các kiểu số, kiểu miền con, kiểu liệt kê, kiểu kí tự và kiểu logic.

Chương 4: Kiểu dữ liệu có cấu trúc. Chương này trình bày khái niệm về kiểu dữ liệu có cấu trúc, sự đặc tả các thuộc tính, đặc tả phép toán, đặc biệt là phép toán lựa chọn một phần tử; các phương pháp lưu trữ một cấu trúc dữ liệu trong bộ nhớ và phương pháp lựa chọn phần tử. Nội dung chủ yếu của chương trình bày các cấu trúc cụ thể như mảng, mẩu tin, chuỗi ký tự, tập hợp…

Chương 5: Kiểu dữ liệu tự định nghĩa. Chương này trình bày về sự trừu tượng hoá, định nghĩa kiểu dữ liệu và sự tương đương của các kiểu dữ liệu được định nghã.

Chương 6: Chương trình con. Chương này trìn bày về sự định nghĩa và cơ chế gọi thực hiện chương trình con, các phương pháp truyền tham số cho chương trình con.

Chương 7: Điều khiển tuần tự. Chương này trình bày các loại điều khiển tuần tự và vấn đề xử lý ngoại lệ.

Chương 8: Lập trình hàm. Chương này trình bày khái niệm, bản chất của lập trình hàm và giới thiệu một ngôn ngữ lập trình hàm điển hình là LISP.

Chương 9: Lập trình logic. Chương này trình bày khái niệm, bản chất của lập trình logic và giới thiệu một ngôn ngữ lập trình hàm điển hình là PROLOG.

KIẾN THỨC TIÊN QUYẾT

Để học tốt môn học ngôn ngữ lập trình cần phải có các kiến thức và kĩ năng lập trình căn bản.

DANH MỤC TÀI LIỆU THAM KHẢO

[1] **Terrence W. Pratt, Marvin V. Zelkowitz;** *Programming Languages: Design and Implementation;* Prentice-Hall, 2000.

[2] **Doris Appleby, Julius J. VandeKopple**; *Programming Languages*; McGraw-Hill; 1997.

[3] **Ryan Stensifer**; *The Study of Programming Languages*; Prentice Hall, 1995.

[4] Maryse CONDILLAC; *Prolog fondements et applications*; BORDAS, Paris 1986.

[5] Website về XLISP

http://webmaker.web.cern.ch/WebMaker/examples/xlisp/www/cldoc_1.html

[6] Website về Turbo Prolog

http://www.csupomona.edu/%7Ejrfisher/www/prolog_tutorial/contents.html

CHƯƠNG 1: MỞ ĐẦU

1.1 TỔNG QUAN

1.1.1 Mục tiêu

Sau khi học xong chương này, sinh viên cần phải nắm:

- Khái niệm và phân loại các ngôn ngữ lập trình.
- Vai trò của ngôn ngữ lập trình trong công nghệ phần mềm.
- Lợi ích của việc nghiên cứu ngôn ngữ lập trình.
- Các tiêu chuẩn để đánh giá ngôn ngữ lập trình.

1.1.2 Nội dung cốt lõi

- Khái niệm về ngôn ngữ lập trình.
- Vai trò của ngôn ngữ lập trình.
- Lợi ích của việc nghiên cứu ngôn ngữ lập trình.
- Các tiêu chuẩn để đánh giá một ngôn ngữ lập trình tốt.

1.1.3 Kiến thức cơ bản cần thiết

Kiến thức và kĩ năng lập trình căn bản

1.2 KHÁI NIỆM VỀ NGÔN NGỮ LẬP TRÌNH

Như chúng ta đã biết, máy tính bao gồm phần cứng là các thiết bị điện tử trong đó thông tin được biểu diễn dưới dạng số nhị phân và phần mềm bao gồm các chương trình được tạo ra bằng cách sử dụng các ngôn ngữ lập trình.

Như vậy ngôn ngữ lập trình (NNLT) là ngôn ngữ dùng để viết các chương trình cho máy tính. Cũng như các ngôn ngữ thông thường, NNLT cũng có từ vựng, cú pháp và ngữ nghĩa. Theo tiến trình lịch sử phát triển, ngôn ngữ lập trình có thể được chia ra làm ba loại chủ yếu như sau:

Ngôn ngữ máy (machine language): Là các chỉ thị dưới dạng nhị phân, can thiệp trực tiếp vào trong các mạch điện tử. Chương trình được viết bằng ngôn ngữ máy thì có thể được thực hiện ngay không cần qua bước trung gian nào. Tuy nhiên chương trình viết bằng ngôn ngữ máy dễ sai sót, cồng kềnh và khó đọc, khó hiểu vì toàn những con số 0 và 1.

Hợp ngữ (assembly language):

Hợp ngữ là một bước tiến vượt bậc đưa ngôn ngữ lập trình thoát ra khỏi ngôn ngữ máy khó hiểu. Ngôn ngữ này xuất hiện vào những năm 1950, nó được thiết kế để máy tính trở nên thân thiện hơn với người sử dụng. Hợp ngữ đưa ra khái niệm biến (variable), nhờ đó mà ta có thể gán một ký hiệu cho một vị trí nào đó trong bộ nhớ mà không phải viết lại địa chỉ này dưới dạng nhị phân mỗi lần sử dụng. Hợp ngữ cũng chứa vài "phép toán giả", tức là ta có thể biểu biễn mã phép toán dưới dạng phát biểu (hay còn gọi là câu lệnh) thay vì dưới dạng nhị phân. Các câu lệnh bao gồm hai phần: phần mã lệnh

(viết tựa tiếng Anh) chỉ phép toán cần thực hiện và phần tên biến chỉ địa chỉ chứa toán hạng của phép toán đó.

Để máy thực hiện được một chương trình viết bằng hợp ngữ thì chương trình đó phải được dịch sang ngôn ngữ máy. Công cụ thực hiện việc dịch đó được gọi là Assembler.

Ngôn ngữ cấp cao (High level language): Là ngôn ngữ được tạo ra và phát triển nhằm phản ánh cách thức người lập trình nghĩ và làm. Ngôn ngữ cấp cao rất gần với ngôn ngữ con người (Anh ngữ) nhưng chính xác như ngôn ngữ toán học. Nhờ ngôn ngữ cấp cao mà lĩnh vực lập trình trở nên phổ biến, rất nhiều người có thể viết được chương trình, và nhờ thế mà các phần mềm phát triển như vũ bão, phục vụ nhiều lĩnh vực của cuộc sống. Cùng với sự phát triển của các thế hệ máy tính, ngôn ngữ lập trình cấp cao cũng được phát triển rất đa dạng và phong phú, việc lập trình cho máy tính vì thế mà cũng có nhiều khuynh hướng khác nhau: lập trình cấu trúc, lập trình hướng đối tượng, lập trình logic, lập trình hàm... Một chương trình viết bằng ngôn ngữ cấp cao được gọi là chương trình nguồn (source programs). Để máy tính "hiểu" và thực hiện được các lệnh trong chương trình nguồn thì phải có một chương trình dịch để dịch chương trình nguồn (viết bằng ngôn ngữ cấp cao) thành chương trình đích.

Trong khuôn khổ tài liệu này, thuật ngữ ngôn ngữ lập trình dùng để chỉ ngôn ngữ lập trình cấp cao.

1.3 VAI TRÒ CỦA NGÔN NGỮ LẬP TRÌNH

Để thấy rõ vai trò của ngôn ngữ lập trình trong công nghệ phần mềm chúng ta hãy xét các giai đoạn chủ yếu để xây dựng một phần mềm. Các giai đoạn đó bao gồm:

- Xác định: Mục tiêu của giai đoạn xác định là để hiểu rõ các yêu cầu của khách hàng. Kết quả của giai đoạn này là mô hình thế giới thực được phản ánh thông qua một tài liệu đặc tả yêu cầu.
- Phân tích: Mục tiêu của giai đoạn này là xác định chính xác hệ thống sẽ làm những gì theo quan điểm của người sử dụng. Kết quả của giai đoạn phân tích là một tài liệu đặc tả chức năng mô tả hệ thống sẽ có những chức năng gì.
- Thiết kế: Mục tiêu của giai đoạn thiết kế là xác định chính xác hệ thống sẽ làm việc như thế nào. Kết quả của giai đoạn này là một tài liệu đặc tả thiết kế. Đây là một tài liệu kỹ thuật mà những người thực hiện sẽ căn cứ vào đó mà tạo ra phần mềm.
- Cài đặt: Là việc thực hiện cách giải quyết vấn đề đã được đề xuất bởi người thiết kế bằng một NNLT. Kết quả của giai đoạn này là một hệ chương trình máy tính.
- Tích hợp và kiểm thử hệ thống: Do các chuyên viên tin học thực hiện nhằm ghép nối các bộ phận của hệ thống và kiểm tra xem hệ thống có được thực hiện đúng theo thiết kế không.
- Chấp nhận: Do các chuyên viên tin học cùng với khách hàng tiến hành nhằm xác nhận hệ thống chương trình bảo đảm các yêu cầu của người sử dụng.
- Vận hành khai thác: Hệ thống được triển khai để sử dụng.

Ở trên chỉ trình bày một mô hình làm phần mềm, gọi là mô hình thác nước (water fall), ngoài ra còn có nhiều mô hình khác. Tuy nhiên trong tất cả các mô hình ấy đều phải có giai đoạn **cài đặt**. Trong đó NNLT đóng vai trò là một **công cụ** giúp con người thực hiện bước cài đặt này. Công cụ đó ngày càng được cải tiến hoàn thiện và có thể nói mọi tiến bộ trong tin học đều thể hiện ra trong NNLT. NNLT vừa là công cụ giúp các nhà tin học giải quyết các vấn đề thực tế nhưng đồng thời cũng là nơi mà những nghiên cứu mới nhất của tin học được đưa vào. Lĩnh vực này vừa mang tính truyền thống vừa mang tính hiện đại.

1.4 LỢI ÍCH CỦA VIỆC NGHIÊN CỨU NNLT

Trước khi nghiên cứu về NNLT, chúng ta cần thảo luận xem vì sao các sinh viên tin học và các nhà lập trình chuyên nghiệp cần phải nắm các khái niệm tổng quát về NNLT. Việc nghiên cứu tốt NNLT sẽ đạt được các lợi ích như sau:

1.4.1 Cho phép lựa chọn một NNLT phù hợp với dự án thực tế

Hiện nay có rất nhiều dự án công nghệ thông tin ứng dụng vào nhiều lĩnh vực khác nhau của cuộc sống. Do tính chất của từng dự án mà phần mềm có thể được cài đặt bằng các NNLT khác nhau. Với một vốn kiến thức rộng về NNLT, những người làm dự án có thể lựa chọn nhanh chóng một NNLT phù hợp với đề án thực tế. Chẳng hạn có thể lựa chọn ngôn ngữ lập trình Java cho các dự án lập trình truyền thông, hay hướng lập trình logic cho các dự án về trí tuệ nhân tạo.

1.4.2 Sử dụng một cách có hiệu quả các công cụ của ngôn ngữ

Các ngôn ngữ nói chung đều cung cấp những công cụ đặc biệt để tạo ra các tiện ích cho lập trình viên, nhưng khi sử dụng chúng không đúng đắn có thể sẽ gây ra những sai lầm lớn. Một ví dụ điển hình là phép đệ quy (recursion) - một công cụ lập trình đặc biệt có hiệu lực trong nhiều ngôn ngữ. Khi sử dụng đệ quy một cách đúng đắn thì có thể cài đặt một giải thuật đẹp đẽ và có hiệu quả. Nhưng trong trường hợp khác nó có thể gây ra một sự lãng phí thời gian chạy máy rất lớn cho một giải thuật đơn giản. Điều này có thể tránh được nếu như lập trình viên có một sự hiểu biết sâu sắc về ngôn ngữ lập trình và các cài đặt bên trong nó.

1.4.3 Làm tăng vốn kinh nghiệm khi xây dựng các chương trình

Nếu người lập trình đã có sự nghiên cứu một cách rộng rãi nhiều ngôn ngữ mà một trong chúng có cài đặt sẵn những công cụ nào đó thì anh ta có thể tự thiết lập những công cụ tương tự khi phải viết chương trình bởi một ngôn ngữ mà trong đó các công cụ như thế chưa được cài đặt.

1.4.4 Tạo sự dễ dàng để học một ngôn ngữ mới

Mặc dù có nhiều NNLT khác nhau nhưng chúng đều có những nguyên tắc chung của NNLT. Rất nhiều ngôn ngữ có chung cú pháp (sai khác nhau chút ít về cách viết), có chung các kiểu dữ liệu (sai khác nhau chút ít về tên gọi). Việc nắm vững các nguyên lý cơ bản của NNLT sẽ là một điều kiện thuận lợi lớn để tiếp cận một cách nhanh chóng với một ngôn ngữ lập trình cụ thể mới. Thực tế cho thấy rằng với những người nắm vững NNLT, khi gặp một ngôn ngữ lập trình cụ thể mới, họ có thể vừa nghiên cứu ngôn ngữ mới này vừa áp dụng để lập trình giải quyết một bài toán theo yêu cầu.

1.4.5 Tạo tiền đề để thiết kế một ngôn ngữ mới

Việc thiết kế ngôn ngữ mới là một đòi hỏi của khoa học phát triển NNLT. Nếu chúng ta không nghiên cứu về NNLT thì không thể nào có kiến thức để xây dựng một ngôn ngữ mới.

1.5 CÁC TIÊU CHUẨN ĐÁNH GIÁ MỘT NGÔN NGỮ LẬP TRÌNH TỐT

Những yếu tố sau tạo nên một ngôn ngữ tốt, nó cũng là những tiêu chuẩn để người lập trình đánh giá ngôn ngữ này tốt hơn ngôn ngữ kia khi lựa chọn một ngôn ngữ để sử dụng. Ngoài ra khi thiết kế một ngôn ngữ lập trình mới, ta cũng phải quan tâm đến các tiêu chuẩn này để có được một ngôn ngữ tốt.

1.5.1 Tính dễ đọc

Tính dễ đọc của một NNLT là sự dễ dàng đọc hiểu một chương trình được viết bằng ngôn ngữ đó. Tính dễ đọc được đặc trưng bởi các thuộc tính sau:

1.- **Sự giản dị**. Một ngôn ngữ được gọi là có tính giản dị nếu ngôn ngữ đó có ít các thành phần cơ sở, tức là ít các yếu tố được định nghĩa trước. Các ngôn ngữ mà chúng ta có thể đạt được một phép toán bằng nhiều cách khác nhau thì không phải là một ngôn ngữ giản dị. Chẳng hạn trong ngôn ngữ C để tăng thêm một đơn vị cho biến count ta có thể sử dụng nhiều cách như count = count + 1, count += 1, count++ hoặc ++count. Các phép toán chồng (overload) cũng làm cho ngôn ngữ trở nên phức tạp. Chẳng hạn toán tử + có thể hiểu là cộng hai số nguyên, cộng hai số thực, hợp hai tập hợp hay ghép nối hai chuỗi ký tự...

2.- **Cấu trúc điều khiển**. Các lệnh có cấu trúc cho phép viết các chương trình sáng sủa, dễ đọc, dễ hiểu. Chúng ta có thể nhận thấy điều này trong các ngôn ngữ thuộc thập niên 1960 như BASIC, FORTRAN trong đó do thiếu các cấu trúc điều khiển nên chương trình phải sử dụng nhiều lệnh GOTO, rất khó theo dõi để hiểu chương trình. Ta hãy so sánh hai đoạn chương trình in ra màn hình 10 số tự nhiên đầu tiên được viết bằng ngôn ngữ BASIC (không có lệnh cấu trúc FOR) và ngôn ngữ Pascal.

Viết bằng BASIC

```
10  i=1;
20  IF i>10 THEN GOTO 60;
30  PRINT i ;
40  i=i+1;
50  GOTO 20;
60  PRINT "In xong";
```

Viết bằng Pascal

```
FOR i:=1 TO 10 DO
   Writeln(i);
Writeln('In xong');
```

3.- **Kiểu dữ liệu và cấu trúc dữ liệu**. Xem xét kiểu dữ liệu và cấu trúc dữ liệu của một ngôn ngữ cũng góp phần đánh giá một ngôn ngữ có dễ đọc hay không. Chẳng hạn trong các ngôn ngữ không có kiểu dữ liệu logic thì phải sử dụng kiểu số để thay thế và do đó mà chương trình trở nên khó đọc. Ví dụ ta hay sử dụng biến found trong các chương trình tìm kiếm một phần tử x trong một mảng a gồm n phần tử. Nếu ngôn ngữ sử dụng có kiểu logic thì ta có thể gán cho found giá trị TRUE hoặc FALSE để biểu diễn trạng thái tìm thấy phần tử cần tìm hay không, ngược lại đối với các ngôn ngữ không có kiểu logic thì ta phải dùng kiểu số và gán cho found giá trị 1 hoặc 0. Ta hãy so sánh hai đoạn chương trình sau để xem đoạn chương trình nào dễ hiểu hơn.

```
found := 0;
i := 1;
While (i<=n)and (found=0) do
  IF a[i]=x THEN found := 1
  ELSE i := i+1;
```

```
found := FALSE;
i := 1;
While(i<=n)and(NOT found) do
  IF a[i]=x THEN found:= TRUE
  ELSE i:=i+1;
```

4.- **Cú pháp**. Cú pháp của ngôn ngữ có ảnh hưởng lớn đến sự dễ đọc hiểu của chương trình. Chúng ta xét một số thí dụ sau để thấy rõ vấn đề này.

- Một số ngôn ngữ quy định độ dài tối đa của danh biểu quá ngắn, chẳng hạn trong FORTRAN 77 độ dài tối đa của danh biểu là 6, do đó tên biến nhiều khi phải viết tắt nên khó đọc hiểu.
- Việc sử dụng từ khóa cũng góp phần làm cho ngôn ngữ trở nên dễ đọc. Chẳng hạn trong ngôn ngữ Pascal chỉ sử dụng một từ khóa end để kết thúc một khối, kết thúc một lệnh case hay kết thúc một lệnh hợp thành do đó chương trình trở nên khó đọc, trong khi Ada dùng các từ khóa end if để kết thúc lệnh if, end loop để kết thúc lệnh vòng lặp thì chương trình dễ đọc hơn.

1.5.2 Tính dễ viết

Tính dễ viết của một ngôn ngữ là khả năng sử dụng ngôn ngữ đó để viết một chương trình cho một vấn đề nào đó một cách dễ dàng hay không. Thông thường các ngôn ngữ dễ đọc thì đều dễ viết. Tính dễ viết phải được xem xét trong ngữ cảnh của vấn đề mà ngôn ngữ được sử dụng để giải quyết. Theo đó không thể so sánh tính dễ viết của hai ngôn ngữ cho cùng một bài toán mà một trong hai được thiết kế để dành riêng giải quyết bài toán đó. Ví dụ để giải quyết bài toán quản trị dữ liệu, chúng ta không thể so sánh Pascal với một hệ quản trị cơ sở dữ liệu như Foxpro, Access hay Oracle.

Sau đây là một số yếu tố quan trọng nhất ảnh hưởng tới tính dễ viết của ngôn ngữ.

1.- **Sự giản dị**. Nếu một ngôn ngữ có quá nhiều cấu trúc thì một số người lập trình sẽ không quen sử dụng hết tất cả chúng. Tốt nhất là có một số nhỏ các cấu trúc ban đầu và một quy tắc để kết hợp chúng thành các cấu trúc phức tạp hơn.

2.- **Hỗ trợ cho trừu tượng**. Một cách ngắn gọn, trừu tượng (abstraction) là khả năng để định nghĩa và sử dụng các cấu trúc hoặc các phép toán phức tạp theo cách thức mà nó cho phép bỏ qua các chi tiết. Một ví dụ về trừu tượng là chương trình con, từ chương trình gọi, chúng ta gọi chương trình con để thực hiện một tác vụ nào đó mà không cần biết các cài đặt chi tiết bên trong chương trình con đó. Thực chất trừu tượng hóa chính là làm cho chương trình sáng sủa hơn.

3.- **Khả năng diễn đạt**. Là những công cụ của ngôn ngữ mà người lập trình có thể sử dụng để diễn đạt giải thuật một cách dễ dàng. Nói cách khác, một ngôn ngữ có khả năng diễn đạt là ngôn ngữ cung cấp cho người lập trình những công cụ sao cho người lập trình có thể nghĩ sao thì viết chương trình như vậy. Chẳng hạn lệnh lặp FOR trong Pascal dễ sử dụng cho cấu trúc lặp với số lần lặp xác định hơn là lệnh WHILE.

1.5.3 Độ tin cậy

Độ tin cậy của một ngôn ngữ lập trình là khả năng của ngôn ngữ hỗ trợ người lập trình tạo ra các chương trình đúng đắn. Độ tin cậy được thể hiện bởi các đặc trưng sau:

1.- **Kiểm tra kiểu**. Là kiểm tra lỗi về kiểu của chương trình trong giai đoạn dịch hoặc trong khi thực hiện. Kiểm tra kiểu là một yếu tố quan trọng đảm bảo độ tin cậy của ngôn ngữ. Kiểm tra kiểu sẽ báo cho người lập trình biết các lỗi về kiểu và yêu cầu họ có các sửa chữa cần thiết để có một chương trình đúng.

2.- **Xử lý ngoại lệ** (Exception Handing). Là một công cụ cho phép chương trình phát hiện các lỗi trong thời gian thực hiện, tạo khả năng để sửa chữa chúng và sau đó tiếp tục thực hiện mà không phải dừng chương trình.

3.- **Sự lắm tên** (Aliasing): Khi có hai hay nhiều tên cùng liên kết tới một ô nhớ ta gọi là sự lắm tên. Chẳng hạn các biến con trỏ trong ngôn ngữ Pascal cùng trỏ đến một ô nhớ. Sự lắm tên có thể làm giảm độ tin cậy do người lập trình không kiểm soát được giá trị được lưu trữ trong ô nhớ. Hãy xét ví dụ sau trong Pascal

```
Var p, q: ^integer;
Begin
     New(p);
     p^ := 50;
     q:= p;  {Cả q và p cùng trỏ đến một ô nhớ}
     writeln(p^, ' và ', q^);
     q^ := 20;
     writeln(p^, ' và ', q^);
end;
```

Kết quả thực hiện đoạn chương trình này là in ra hai dòng:
50 và 50
20 và 20

Trong khi nhiều người lầm tưởng hai dòng sẽ in ra là:
50 và 50
50 và 20

1.5.4 Chi phí

Chi phí của một ngôn ngữ cũng thường được quan tâm như là một tiêu chuẩn để đánh giá ngôn ngữ. Chi phí ở đây phải được hiểu là cả tiền bạc và thời gian. Chi phí này bao gồm:

- Chi phí đào tạo lập trình viên sử dụng ngôn ngữ. Chi phí này phụ thuộc vào sự giản dị của ngôn ngữ.
- Chi phí cài đặt chương trình. Chi phí này phụ thuộc vào tính dễ viết của ngôn ngữ.
- Chi phí dịch chương trình.
- Chi phí thực hiện chương trình.
- Chi phí bảo trì chương trình.

- Chi phí mua trình biên dịch

1.6 CÂU HỎI ÔN TẬP

1. Vai trò của ngôn ngữ lập trình trong công nghệ phần mềm là gì?
2. Nêu các lợi ích của việc nghiên cứu ngôn ngữ lập trình.
3. Nêu tên các tiêu chuẩn để đánh giá một ngôn ngữ lập trình tốt.
4. Nêu tên các yếu tố ảnh hưởng đến tính dễ đọc.
5. Nêu tên các yếu tố ảnh hưởng đến tính dễ viết.
6. Nêu tên các yếu tố ảnh hưởng đến độ tin cậy.
7. Thế nào là sự lắm tên?
8. Chi phí của ngôn ngữ lập trình bao gồm những chi phí nào?

CHƯƠNG 2: KIỂU DỮ LIỆU

2.1 TỔNG QUAN

2.1.1 Mục tiêu

Sau khi học xong chương này, sinh viên cần phải nắm:

- Khái niệm về đối tượng dữ liệu, biến, hằng.
- Khái niệm về kiểu dữ liệu.
- Các phương pháp kiểm tra kiểu và biến đổi kiểu.

2.1.2 Nội dung cốt lõi

- Các khái niệm về đối tượng dữ liệu, kiểu dữ liệu.
- Sự khai báo các đối tượng dữ liệu trong chương trình.
- Kiểm tra kiểu, biến đổi kiểu dữ liệu.
- Vấn đề gán giá trị và khởi tạo biến.

2.1.3 Kiến thức cơ bản cần thiết

Kiến thức và kĩ năng lập trình căn bản

2.2 ĐỐI TƯỢNG DỮ LIỆU

2.2.1 Khái niệm đối tượng dữ liệu

Trong máy tính thực dữ liệu được lưu trữ ở bộ nhớ trong và bộ nhớ ngoài. Trong đó dữ liệu được tổ chức thành các *bit*, các *byte* hoặc *word*. Tuy nhiên trong máy tính ảo của một NNLT nào đó, dữ liệu có tổ chức phức tạp hơn với các mảng, ngăn xếp, số, chuỗi ký tự ...

*Người ta sử dụng thuật ngữ **đối tượng dữ liệu** (ĐTDL) để chỉ một nhóm của một hoặc nhiều mẩu dữ liệu trong máy tính ảo.*

Khác với tính chất tĩnh tương đối của các vùng nhớ trong máy tính thực, các ĐTDL và các mối liên hệ nội tại của chúng lại thay đổi một cách động trong quá trình thực hiện chương trình.

2.2.2 Các loại ĐTDL

Xét về mặt cấu trúc thì người ta phân ĐTDL làm hai loại là **ĐTDL sơ cấp** và **ĐTDL có cấu trúc** hay cấu trúc dữ liệu.

***ĐTDL sơ cấp** là một ĐTDL chỉ chứa một giá trị dữ liệu đơn.* Hằng hạn như một số, một kí tự,…

***ĐTDL có cấu trúc** hay cấu trúc dữ liệu là một tích hợp của các ĐTDL khác*. Mỗi ĐTDL thành phần của ĐTDL có cấu trúc được gọi là một **phần tử**. Mỗi phần tử của cấu trúc dữ liệu có thể là một ĐTDL sơ cấp hay cũng có thể là một ĐTDL có cấu trúc khác. Ví dụ một chuỗi kí tự, một tập hợp các số, một véctơ, một ma trận,…đều là các ĐTDL có cấu trúc.

Xét về mặt nguồn gốc thì có thể phân ĐTDL làm hai loại: **ĐTDL tường minh** và **ĐTDL ẩn**.

ĐTDL tường minh *là một ĐTDL do người lập trình tạo ra* chẳng hạn như các biến, các hằng,… được người lập trình viết ra trong chương trình.

ĐTDL ẩn *là một ĐTDL được định nghĩa bởi hệ thống* như các ngăn xếp lưu trữ các giá trị trung gian, các mẩu tin kích hoạt chương trình con, các ô nhớ đệm của tập tin... Các ĐTDL này được phát sinh một cách tự động khi cần thiết trong quá trình thực hiện chương trình và người lập trình không thể truy cập đến chúng được.

2.2.3 Thuộc tính của ĐTDL

Thuộc tính của một ĐTDL là một tính chất đặc trưng của ĐTDL đó.

Mỗi ĐTDL có một tập hợp các thuộc tính để phân biệt ĐTDL này với ĐTDL khác.

Các ĐTDL sơ cấp chỉ có một thuộc tính duy nhất là kiểu dữ liệu của đối tượng đó. Các ĐTDL có cấu trúc có thêm các thuộc tính nhằm xác định số lượng, kiểu dữ liệu của các phần tử và các thuộc tính khác.

2.2.4 Giá trị dữ liệu

Giá trị dữ liệu (GTDL) của một ĐTDL sơ cấp có thể là một số, một ký tự hoặc là một giá trị logic tùy thuộc vào kiểu của ĐTDL đó.

Mỗi GTDL thường được biểu diễn bởi một dãy các *bit* trong bộ nhớ của máy tính.

Cần phân biệt hai khái niệm ĐTDL và GTDL. Một ĐTDL luôn luôn được biểu diễn bởi một khối ô nhớ trong bộ nhớ của máy tính trong khi một GTDL được biểu diễn bởi một dãy các *bit*. Khi nói rằng một ĐTDL A chứa một GTDL B có nghĩa là: khối ô nhớ biểu diễn cho A chứa dãy *bit* biểu diễn cho B.

GTDL của một ĐTDL có cấu trúc là một tập hợp các GTDL của các phần tử của ĐTDL có cấu trúc đó.

2.2.5 Thời gian tồn tại

Thời gian tồn tại (lifetime) của một ĐTDL là khoảng thời gian ĐTDL chiếm giữ bộ nhớ của máy tính. Thời gian này được tính từ khi ĐTDL được tạo ra cho đến khi nó bị hủy bỏ trong quá trình thực hiện chương trình.

2.2.6 Các mối liên kết

Một ĐTDL có thể tham gia vào nhiều mối liên kết trong thời gian tồn tại của nó. Các liên kết quan trọng nhất là:

- Sự liên kết của ĐTDL với một hoặc nhiều giá trị. Sự liên kết này có thể bị thay đổi bởi phép gán trị.
- Sự liên kết của một ĐTDL với một hoặc nhiều tên được tham chiếu trong quá trình thực hiện chương trình. Các liên kết này được thiết lập bởi sự khai báo và thay đổi bởi việc gọi và trả chương trình con.

- Sự liên kết của một ĐTDL với một số ĐTDL khác gọi là các hợp thành (component). Các liên kết này thường được biểu diễn bởi giá trị con trỏ và nó có thể bị thay đổi bởi việc thay đổi con trỏ.
- Sự liên kết của một ĐTDL với ô nhớ trong bộ nhớ. Sự liên kết này thường không thể thay đổi một cách trực tiếp bởi người lập trình mà nó được thiết lập và có thể bị thay đổi bởi các thường trình (routine) quản lý bộ nhớ của máy tính ảo.

2.3 BIẾN VÀ HẰNG

2.3.1 Biến

Biến là một ĐTDL được người lập trình định nghĩa và đặt tên một cách tường minh trong chương trình. Giá trị của biến có thể bị thay đổi trong thời gian tồn tại của nó.

Tên biến được dùng để xác định và tham khảo tới biến. Trong các NNLT, tên biến thường được quy định dưới dạng một dãy các chữ cái, dấu gạch dưới và các chữ số, bắt đầu bằng một chữ cái và có chiều dài hữu hạn.

2.3.2 Hằng

Hằng là một ĐTDL có tên và giá trị của hằng không thay đổi trong thời gian tồn tại của nó.

Hằng trực kiện (literal constant) là một hằng mà tên của nó là sự mô tả giá trị của nó (chẳng hạn "27" là sự mô tả số thập phân của ĐTDL giá trị 27). Chú ý sự khác biệt giữa 2 giá trị 27. Một cái là một số nguyên được biểu diễn thành một dãy các *bit* trong bộ nhớ trong quá trình thực hiện chương trình và cái tên "27" là một chuỗi 2 ký tự "2" và "7" mô tả một số nguyên như nó được viết trong chương trình.

2.4 KIỂU DỮ LIỆU

2.4.1 Định nghĩa kiểu dữ liệu

Kiểu dữ liệu là một tập hợp các ĐTDL và tập hợp các phép toán thao tác trên các ĐTDL đó.

Mọi NNLT đều xây dựng cho mình một tập các kiểu dữ liệu nguyên thuỷ. Chẳng hạn ngôn ngữ LISP, kiểu dữ liệu chính là các cây nhị phân với các phép toán CAR, CDR và CONS còn đối với các ngôn ngữ cấp cao khác thì các kiểu dữ liệu nguyên thủy thường là: *integer*, *real*, *character* và *boolean*. Hơn nữa các ngôn ngữ còn cung cấp phương tiện cho phép người lập trình định nghĩa các kiểu dữ liệu mới.

Kiểu dữ liệu trong ngôn ngữ được nghiên cứu trên hai phương diện khác nhau: **Sự đặc tả** và **sự cài đặt** kiểu dữ liệu.

2.4.2 Sự đặc tả kiểu dữ liệu

Khi đặc tả một kiểu dữ liệu chúng ta thường quan tâm đến các thành phần cơ bản sau:

- Các thuộc tính nhằm phân biệt các ĐTDL của kiểu.
- Các giá trị mà các ĐTDL của kiểu có thể có.

- Các phép toán có thể thao tác trên các ĐTDL của kiểu.

Ví dụ, xét sự đặc tả kiểu dữ liệu mảng ta thấy:

1.- Các thuộc tính có thể bao gồm: số chiều, miền xác định của chỉ số đối với mỗi chiều và kiểu dữ liệu của các phần tử.

2.- Các giá trị có thể nhận của các phần tử mảng.

3.- Các phép toán có thể bao gồm: phép lựa chọn một phần tử mảng thông qua việc sử dụng chỉ số của phần tử đó, phép gán một mảng cho một mảng khác…

Phép toán

Các phép toán thao tác trên các ĐTDL là một **bộ phận không thể thiếu** của kiểu dữ liệu. Khi nói đến kiểu dữ liệu mà chúng ta không quan tâm đến các phép toán là chưa hiểu đầy đủ về kiểu dữ liệu đó. Mà dường như khiếm khuyết này lại hay xảy ra. Ví dụ khi nói đến kiểu integer trong ngôn ngữ Pascal, chúng ta chỉ nghĩ rằng đó là kiểu số nguyên, có các giá trị từ -32768 đến 32767, mà ít khi quan tâm đến các phép toán như +, -, *, … hay nói chính xác hơn chúng ta cứ nghĩ các phép toán này là mặc nhiên phải có. Trong tin học không có cái gì tự nhiên mà có cả, mọi cái hoặc do chúng ta tự tạo ra hoặc sử dụng cái có sẵn do người khác đã tạo ra. Nhấn mạnh việc có mặt các phép toán trong kiểu dữ liệu là để lưu ý chúng ta khi định nghĩa một kiểu dữ liệu mới, phải trang bị cho nó các phép toán cần thiết.

Có hai loại phép toán là các **phép toán nguyên thủy** được ngôn ngữ định nghĩa và các **phép toán do người lập trình định nghĩa** như là các chương trình con.

Phép toán trong NNLT về phương diện lôgic là một hàm toán học: đối với một đối số (argument) đã cho nó có một kết quả duy nhất và xác định.

Mỗi một phép toán có một miền xác định (domain) là tập hợp các đối số và một miền giá trị (range) là tập hợp các kết quả có thể tạo ra. Hoạt động của phép toán xác định kết quả được tạo ra đối với tập hợp bất kỳ các đối số đã cho. Giải thuật chỉ rõ làm thế nào để xác định kết quả đối với tập hợp bất kỳ các đối số đã cho là phương pháp phổ biến để xác định hoạt động của phép toán. Ngoài ra còn có những cách xác định khác chẳng hạn để xác định hoạt động của phép toán nhân chúng ta có thể cho một "bảng nhân" thay vì cho giải thuật của phép nhân hai số.

Để chỉ rõ miền xác định của phép toán, số lượng, thứ tự và kiểu dữ liệu của các đối số, tương tự miền giá trị, số lượng, thứ tự và kiểu dữ liệu của các kết quả người ta thường sử dụng các ký hiệu toán học.

Tên phép toán: Miền xác định -> Miền giá trị

Trong đó **Miền xác định = Kiểu đối số X Kiểu đối số X…**

(Miền xác định là tập tích Đề-các của các kiểu đối số)

Miền giá trị = Kiểu kết quả X Kiểu kết quả X ...

(Miền giá trị là tập tích Đề-các của các kiểu kết quả)

Khi nghiên cứu các phép toán trên các kiểu dữ liệu chúng ta cần lưu ý các vấn đề sau:

1.- Các phép toán không được xác định đầu vào một cách chắc chắn.

Một phép toán được xác định trên nhiều hơn một miền xác định thường chứa đựng các lỗi. Ví dụ các phép toán số học có thể xác định trên nhiều tập hợp số khác nhau có thể gây ra sự tràn số hoặc một kết quả sai lệch mà ta không thể kiểm soát được.

Ví dụ trong ngôn ngữ Pascal, phép cộng có thể xác định trên nhiều miền xác định khác nhau như *integer*, *real*,… nên có thể có những kết quả sai lệch như trong ví dụ sau:

```
var a, b : integer;
begin
        {1}     a:= 32767;
        {2}     b:= 30000;
        {3}     writeln(32767+30000);
        {4}     writeln(a+b);
end.
```

Kết quả của chương trình trên là 62767 và -2769.

Trong đó 62767 là kết quả của phép cộng 32767+30000. Đây là một kết quả đúng, do máy tính "hiểu" các số 32767 và 30000 là các số thực (real) và phép "+" trong lệnh {3} là "phép cộng các số thực". Ngược lại -2769 là kết quả sai của phép toán a+b. Về mặt toán học thì kết quả của a+b là 62767, nhưng kết quả của chương trình máy tính lại là -2769! Sở dĩ chương trình máy tính (ngôn ngữ Pascal) lại có kết quả này là do hai biến a và b được khai báo là các biến thuộc kiểu integer nên phép "+" trong lệnh {4} được hiểu là "phép cộng các số nguyên". Về nguyên tắc thì tổng a+b phải có giá trị thuộc kiểu integer nhưng do tập giá trị của kiểu integer là các số nguyên từ -32768 đến 32767 nên mới "sinh chuyện".

2.- Các đối số ẩn

Các phép toán trong chương trình thông thường sẽ được gọi với một tập hợp các đối số tường minh (explicit arguments). Tuy nhiên các phép toán có thể truy cập đến những đối số ẩn (implicit arguments) thông qua việc sử dụng các biến toàn cục hoặc tham chiếu các biến không cục bộ khác. Những đối số ẩn như thế sẽ gây khó khăn cho việc kiểm soát giá trị dữ liệu và do đó có thể ảnh hưởng đến kết quả của chương trình.

Ví dụ:

```
Var x: Integer;
Procedure P;
Begin
        x:= 0;
End;
Begin
        {1}      x:=10;
        {2}     P;
        {3}     Writeln(x);
End.
```

Trong ví dụ trên, chương trình con P thực hiện việc giá trị 0 cho biến toàn cục x. Trong chương trình chính, mặc dù ta mới gán 10 cho x (lệnh 1), nhưng sau khi gọi thủ tục P (lệnh 2) thì ở lệnh 3, x lại có giá trị 0. Việc chương trình con sử dụng biến không cục bộ như vậy sẽ dễ gây ngộ nhận cho người lập trình rằng x có giá trị 10, đặc biệt khi thủ tục P được định nghĩa ở một đoạn nào đó, xa đoạn chương trình chính.

3.- Hiệu ứng lề

Một phép toán có thể trả về một kết quả ẩn, và các kết quả ẩn như vậy sẽ gây ra hiệu ứng lề (side effect) làm thay đổi giá trị được lưu trữ của các ĐTDL khác mà người lập trình khó lòng kiểm soát. Các phép toán có thể gây nên hiệu ứng lề là phép gán (có trả về một giá trị) và các chương trình con mà tham số được truyền bằng quy chiếu. Chẳng hạn xét ví dụ sau trong Pascal:

```
var m,n: integer;
function f(var a: integer): integer;
begin
    a := 2*a;
    f  := 5;
end;
begin
    m := 10;
    n  := m + f(m);
    writeln(n);
    readln;
end.
```

Với mọi số *integer* a hàm f luôn trả về một kết quả tường minh là 5 và một kết quả ẩn là 2a, chính kết quả ẩn này làm thay đổi giá trị của ĐTDL m do đó n sẽ có giá trị là 25 chứ không phải là 15 như chúng ta lầm tưởng.

2.4.3 Sự cài đặt kiểu dữ liệu

Khi xét sự cài đặt kiểu dữ liệu ta phải quan tâm đến hai yếu tố sau:

- Tổ chức lưu trữ giá trị dữ liệu của kiểu dữ liệu trong bộ nhớ của máy tính hay còn gọi là sự biểu diễn trong bộ nhớ.
- Giải thuật thực hiện các phép toán thao tác trên các giá trị dữ liệu của kiểu.

Hai yếu tố này liên quan chặt chẽ đến nhau, nói chính xác hơn là tuỳ thuộc vào cách thức tổ chức lưu trữ mà có các giải thuật thao tác tương ứng.

2.5 SỰ KHAI BÁO

2.5.1 Khái niệm khai báo

Khai báo là một lệnh trong chương trình dùng để chuyển tới bộ dịch, thông tin về số lượng và kiểu của ĐTDL cần thiết trong quá trình thực hiện chương trình.

Nhờ vị trí của khai báo trong chương trình, chẳng hạn đầu chương trình con, sự khai báo có thể chỉ rõ thời gian tồn tại của ĐTDL.

Sự khai báo còn xác định sự liên kết của các ĐTDL với các tên của nó.

Có hai loại khai báo là khai báo tường minh và khai báo ẩn. Khai báo tường minh là sự khai báo do người lập trình viết ra trong chương trình, như trong các khai báo của Pascal. Khai báo ẩn như trong trường hợp các ĐTDL được dùng một cách mặc nhiên mà không cần một sự khai báo tường minh nào. Ví dụ trong ngôn ngữ FORTRAN biến INDEX có thể dùng mà không cần khai báo tường minh và nó được trình biên dịch FORTRAN hiểu một cách mặc nhiên là một biến nguyên bởi vì tên của nó được bắt đầu bởi một trong các chữ cái từ I đến N.

Ngôn ngữ lập trình được chia làm hai loại: **ngôn ngữ khai báo**, trong đó các ĐTDL phải được khai báo trước khi sử dụng và **ngôn ngữ không khai báo**, trong đó ĐTDL có thể sử dụng mà không cần phải khai báo. Với ngôn ngữ khai báo, ĐTDL sau khi đã khai báo phải sử dụng đúng như nó đã được khai báo, trong khi đối với ngôn ngữ không khai báo, một ĐTDL có thể sử dụng một cách tuỳ thích. Đây là một trong những lý do làm cho ngôn ngữ không khai báo trở nên mềm dẻo hơn.

2.5.2 Mục đích của sự khai báo

Việc khai báo có các mục đích quan trọng sau:

- Chọn một tổ chức lưu trữ tốt nhất cho ĐTDL. Chẳng hạn trong ngôn ngữ Pascal để lưu trữ ngày trong tháng ta có thể khai báo biến *ngay* có kiểu là *integer* được lưu trữ trong bộ nhớ bởi 2 *byte*. Tuy nhiên trong một tháng chỉ có tối đa 31 ngày nên ta có thể khai báo biến *ngay* có kiểu miền con 1..31 được lưu trữ trong bộ nhớ chỉ với 1 *byte*.
- Quản lý bộ nhớ: Sự khai báo cho phép xác định thời gian tồn tại của ĐTDL mà các chương trình quản lý bộ nhớ sử dụng để cấp phát và giải phóng bộ nhớ cho ĐTDL.
- Các phép toán chung. Hầu hết các ngôn ngữ đều dùng các ký hiệu đặc biệt như "+" để chỉ một phép toán nào đó phụ thuộc vào kiểu dữ liệu của đối số. Ví dụ trong Pascal, "A+B" có nghĩa là "phép cọng các số nguyên" nếu A và B thuộc kiểu *Integer*, "phép cọng các số thực" nếu A và B thuộc kiểu *real* và là "phép hợp" nếu A và B thuộc kiểu tập hợp. Các phép toán như thế được gọi là các phép toán chung bởi vì nó không chỉ rõ một phép toán nhất định nào. Sự khai báo cho phép bộ dịch xác định một phép toán cụ thể được chỉ định bởi ký hiệu phép toán chung. Ví dụ trong Pascal, từ sự khai báo hai biến A và B, trình biên dịch sẽ xác định được phép toán cụ thể trong ba phép toán, theo đó nếu A, B là các biến *integer* thì "A+B" là phép cộng hai số nguyên, nếu A, B là hai biến *real* thì "A+B" là phép cộng hai số thực… Ngược lại trong SNOBOL4 vì không có khai báo kiểu cho biến nên sự xác định phép "+" nào để thực hiện phải được làm tại thời điểm mà một phép "+" bị bắt gặp trong quá trình thực hiện chương trình.
- Kiểm tra kiểu. Mục đích quan trọng nhất của việc khai báo là chúng cho phép kiểm tra kiểu của biến. Vì tính chất quan trọng của việc kiểm tra kiểu nên chúng ta sẽ xem xét nó trong mục sau.

2.6 KIỂM TRA KIỂU VÀ BIẾN ĐỔI KIỂU

2.6.1 Khái niệm kiểm tra kiểu

Kiểm tra kiểu là kiểm tra xem kiểu thực nhận được của các đối số trong một phép toán có đúng với kiểu dữ liệu mà các đối số đó cần có hay không.

Ví dụ trước khi thực hiện lệnh gán X := A * B việc kiểm tra phải được xác định đối với 2 phép toán nhân và phép gán. Trước hết phép nhân phải nhận được 2 tham số A, B có kiểu số, nếu cả A và B đúng là có kiểu số (chẳng hạn số nguyên) thì tiếp tục kiểm tra cho phép toán gán. Tích A*B sẽ là một số nguyên nên X cũng phải là một biến thuộc kiểu nguyên, nếu không đúng như vậy thì có sự sai kiểu.

Kiểm tra kiểu có thể được tiến hành trong lúc chạy chương trình (kiểm tra kiểu động) hoặc trong lúc biên dịch chương trình (kiểm tra kiểu tĩnh).

2.6.2 Kiểm tra kiểu động

Khái niệm:

Kiểm tra kiểu động là kiểm tra kiểu được thực hiện ***trong khi thực hiện chương trình.***

Thông thường kiểm tra kiểu động được thực hiện một cách tức thì trước khi thực hiện một phép toán.

Phương pháp thực hiện:

Để kiểm tra kiểu động người ta phải lưu trữ thông tin về kiểu của mỗi một ĐTDL cùng với ĐTDL đó. Trước khi thực hiện một phép toán thông tin về kiểu của mỗi một đối số được kiểm tra. Nếu kiểu của các đối số là đúng thì phép toán sẽ được thực hiện và kiểu của kết quả sẽ được ghi lại để dùng kiểm tra cho các phép toán sau, ngược lại sẽ có một thông báo lỗi về kiểu .

Ngôn ngữ sử dụng:

Kiểm tra kiểu động được sử dụng trong các **ngôn ngữ không khai báo** như SNOBOL4, LISP, APL. Trong các ngôn ngữ này không có sự khai báo kiểu cho biến. Kiểu dữ liệu của các biến A và B trong biểu thức "A+B" có thể thay đổi trong quá trình thực hiện chương trình. Trong những trường hợp như vậy, kiểu của A và B phải được kiểm tra động tại mỗi lần phép cộng được gọi thực hiện. Trong các ngôn ngữ không khai báo, các biến đôi khi được gọi là không định kiểu vì chúng không có kiểu cố định.

Ưu điểm:

Ưu điểm chủ yếu của kiểm tra kiểu động là tính mềm dẻo trong khi viết chương trình: không yêu cầu khai báo kiểu và kiểu của ĐTDL có thể thay đổi trong quá trình thực hiện chương trình. Người lập trình không phải lo lắng về kiểu dữ liệu.

Nhược điểm:

Tuy nhiên kiểm tra kiểu động cũng có một số yếu điểm như sau:

- Có khả năng bỏ sót lỗi về kiểu. Bởi vì việc kiểm tra động chỉ kiểm tra tại thời điểm thực hiện phép toán do đó các phép toán nằm trong nhánh chương trình không được thực hiện thì sẽ không được kiểm tra. Bất kỳ một nhánh chưa được kiểm tra nào đều có thể chứa các đối số có lỗi về kiểu và do đó các lỗi này có thể xuất hiện tại thời điểm sau đó. Ví dụ ta có một đoạn chương trình sau được viết trong một ngôn ngữ kiểm tra kiểu động:

  Nhập số a từ bàn phím;

  Nhập số b từ bàn phím;

  Nếu a > b Thì x := a + b

  Ngược lại x := a + “titi”;

  Nếu khi thực hiện đoạn chương trình này, người sử dụng luôn luôn nhập số a lớn hơn số b thì điều kiện a>b luôn luôn đúng nên không bao giờ chương trình

thực hiện lệnh x := a + "titi" do đó không bao giờ phát hiện lỗi về kiểu: a là một số, không thể cộng với "titi" là một chuỗi.

- Kiểm tra kiểu động đòi hỏi thông tin về kiểu phải được lưu giữ cho mỗi một ĐTDL trong quá trình thực hiện chương trình do đó yêu cầu về bộ nhớ phải lớn.
- Kiểm tra kiểu phải được tiến hành tức thì trước mỗi khi thực hiện một phép toán nên tốc độ thực hiện chương trình chậm.

2.6.3 Kiểm tra kiểu tĩnh

Khái niệm:

Kiểm tra kiểu tĩnh là sự kiểm tra kiểu ***được thực hiện trong quá trình dịch chương trình****.*

Phương pháp thực hiện:

Theo nguyên tắc kiểm tra kiểu tĩnh, thông tin về kiểu của ĐTDL phải được cung cấp cho bộ dịch. Thông tin này một phần được cung cấp bởi phép khai báo của người lập trình và một phần bởi ngôn ngữ .

Các thông tin bao gồm:

- Đối với mỗi một phép toán thì đó là số lượng, thứ tự và kiểu dữ liệu của đối số và kiểu của kết quả. Đối với các phép toán nguyên thuỷ thì việc định nghĩa ngôn ngữ sẽ cung cấp các thông tin này còn đối với chương trình con thì người lập trình phải xác định một cách tường minh.
- Đối với mỗi một biến thì đó là kiểu của biến.
- Đối với mỗi một hằng, thì đó là kiểu của đối tượng dữ liệu hằng. Ngữ nghĩa của một hằng trực kiện sẽ chỉ ra kiểu của nó, chẳng hạn "2" là một số nguyên, "2.3" là một số thực.

Kiểm tra kiểu tĩnh được thực hiện như sau: Thông qua đoạn đầu của chương trình, bộ biên dịch tập hợp thông tin từ sự khai báo trong chương trình vào trong bảng danh biểu (symbol table) nơi chứa thông tin về kiểu của các biến và chương trình con. Bộ biên dịch cũng sẽ có thông tin về các phép toán nguyên thuỷ được định nghĩa bởi ngôn ngữ, các hằng...Khi gặp một phép toán thì phải tra trong bảng danh biểu để xác định kiểu của mỗi một đối số có hợp lệ hay không. Chú ý rằng nếu phép toán là phép toán chung như đã nói ở trên thì có thể có nhiều kiểu hợp lệ cho một đối số. Nếu kiểu của đối số là hợp lệ thì kiểu kết quả được xác định và bộ biên dịch ghi lại thông tin này để kiểm tra các phép toán sau.

Ngôn ngữ sử dụng:

Kiểm tra kiểu tĩnh thường được sử dụng trong các **ngôn ngữ khai báo** tức là khi viết chương trình, các biến phải được khai báo kiểu trước khi sử dụng như Pascal, C…

Ưu điểm:

- Do phép kiểm tra kiểu tĩnh kiểm tra tất cả các phép toán có thể xuất hiện trong bất kỳ một lệnh nào của chương trình, tất cả các nhánh của chương trình đều được kiểm tra nên không thể có sự sót lỗi vê kiểu.

- Mặt khác thông tin về kiểu không gắn với ĐTDL tại thời điểm thực hiện chương trình nên tiết kiệm được bộ nhớ và tăng tốc độ thực hiện chương trình.

Nhược điểm:

Yếu điểm chủ yếu của kiểm tra kiểu tĩnh là chương trình không mềm dẻo, người lập trình luôn phải lo lắng về việc sử dụng biến không đúng kiểu.

2.7 CHUYỂN ĐỔI KIỂU

Trong quá trình kiểm tra kiểu, nếu có sự không tương thích giữa kiểu thực của đối số và kiểu đang được monng đợi của phép toán ấy thì có hai lựa chọn có thể:

- Sự không tương thích kiểu bị báo lỗi hoặc
- Một sự chuyển đổi kiểu tự động được thi hành để đổi kiểu của đối số thực tế thành kiểu đúng với yêu cầu.

Chuyển đổi kiểu là một phép toán được định nghĩa như sau:

Sự chuyển đổi: Kiểu1 -> Kiểu2 nghĩa là sự chuyển đổi lấy ĐTDL của một kiểu và sản sinh ra một ĐTDL "tương ứng" của một kiểu khác. Hầu hết các ngôn ngữ đều cung cấp hai phương pháp chuyển đổi kiểu:

- Trang bị một tập hợp các hàm đã được xây dựng mà người lập trình có thể gọi trong chương trình để tạo ra sự chuyển đổi kiểu. Ví dụ Pascal trang bị hàm ROUND để đổi một ĐTDL số thực thành một đối tượng dữ liệu nguyên với giá trị bằng phần nguyên của số thực.
- Như là một sự chuyển đổi tự động (còn gọi là ép kiểu) do ngôn ngữ thực hiện trong một số trường hợp không tương thích kiểu nào đó. Ví dụ trong Pascal các đối số của phép toán số học "+" có lẫn số thực và số nguyên hoặc khi gán một số nguyên cho một biến số thực thì số nguyên phải được đổi một cách tự động thành kiểu thực.

Đối với kiểm tra kiểu động thì sự chuyển đổi kiểu tự động được diễn ra tại điểm mà sự không tương thích kiểu được tìm thấy trong quá trình thực hiện chương trình. Đối với sự kiểm tra kiểu tĩnh thì một mã phụ sẽ được xen vào trong chương trình đích dùng để gọi tới hàm biến đổi kiểu tại điểm thích hợp trong quá trình thực hiện.

Chuyển đổi kiểu tự động giúp người lập trình khỏi mọi lo lắng về sự sai kiểu và tránh việc gọi tới một số lượng lớn các phép biến đổi kiểu tường minh trong chương trình. Tuy nhiên chúng ta nên tránh việc chuyển đổi kiểu bằng cách viết các phép toán đúng kiểu. Chẳng hạn trong lập trình thay vì viết lệnh x := 1 (với x là biến số thực) ta nên viết x := 1.0, với lệnh trước thì khi thực hiện phải có một sự chuyển đổi kiểu tự động còn với lệnh sau thì không cần nên thời gian thực hiện sẽ nhanh hơn.

2.8 GÁN VÀ KHỞI TẠO

2.8.1 Phép gán

Gán trị cho biến là sự lưu trữ giá trị dữ liệu vào trong ô nhớ của biến đó.

Gán trị là một phép toán cơ bản trong các NNLT. Nó dùng để thay đổi sự liên kết của giá trị với ĐTDL.

Nói chung các ngôn ngữ khác nhau thì phép gán cũng khác nhau.

Sự khác nhau đầu tiên là khác nhau về cú pháp, chẳng hạn ta có một số cú pháp lệnh gán như sau:

A := B	(Pascal hay Ada)
A = B	(C, C++, Fortran, PL/1 và SNOBOL4)
MOVE B TO A	(COBOL)
A <- B	(APL)
(SETQ A B)	(LISP)

Sự khác nhau thứ hai là kết quả trả về của phép gán trị. Nói chung trong các ngôn ngữ, lệnh gán trị không trả về kết quả. Chẳng hạn trong Pascal, đặc tả phép gán là **Phép gán (:=) Type1 x Type2 -> Void** với sự hoạt động: Đặt giá trị được chứa trong đối tượng dữ liệu Type1 thành bản sao của giá trị được chứa trong đối tượng dữ liệu Type2 và trả về một kết quả có kiểu void (có thể hiểu là không có kết quả trả về).

Trong một số ngôn ngữ như C, C++ và LISP, phép gán trả về trực tiếp một kết quả là một bản sao của giá trị được gán. Chẳng hạn trong C, sự đặc tả phép gán là

Phép gán (=) Type1 x Type2 -> Type3 với sự hoạt động: Đặt giá trị được chứa trong đối tượng dữ liệu Type1 thành bản sao của giá trị được chứa trong đối tượng dữ liệu Type2 và tạo ra một ĐTDL mới Type3 chứa bản sao giá trị của Type2, trả về Type3 như là một kết quả.

Vì phép gán trong Pascal không trả về một kết quả nên chúng ta chỉ sử dụng chức năng "gán trị" của nó mà thôi. Vì không có kết quả trả về nên mỗi một lệnh ta chỉ có thể viết một phép gán, chẳng hạn để gán giá trị 10 cho hai biến A và B ta phải viết hai lệnh B := 10; A := B; hoặc A := 10; B := 10;

Ngược lại khi lập trình bằng ngôn ngữ C, vì phép gán có trả về một kết quả nên ta có thể viết: A = B = 10; Trong đó phép gán B =10 vừa thực hiện chức năng "gán trị" giá trị 10 cho B vừa trả về 1 giá trị để gán tiếp cho A. Giá trị được trả về như là một kết quả của phép gán B cho A bị bỏ qua vì lệnh không chứa một phép toán nào sau đó nữa.

Phép gán trong ngôn ngữ C++ cũng có cùng cơ chế như trong C, vì vậy khi thiết kế toán tử gán cho một đối tượng nào đó (Overloading toán tử = trong khi xây dựng một lớp nào đó) ta phải viết:

```
<tên lớp> & operator= (const <tên lớp> & Obj)
{
  // Thực hiện việc gán dữ liệu;
  return *this;
}
```

Trong đó tên lớp là tên của lớp chúng ta đang định nghĩa. Phương thức này nhận vào một đối tượng Obj, thực hiện việc gán Obj cho đối tượng hiện hành và trả đối tượng hiện hành này về như một kết quả.

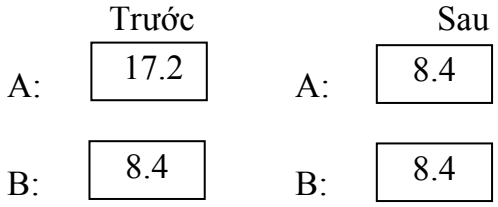
Ví dụ ta tạo một class có tên là point để biểu diễn cho một điểm trong mặt phẳng được đặc trưng bởi hai tọa độ x và y. Trong chương trình ta muốn gán các điểm cho nhau, nên trong khi định nghĩa class point, ta phải định nghĩa toán tử gán. Cụ thể như sau:

```
class point {
```

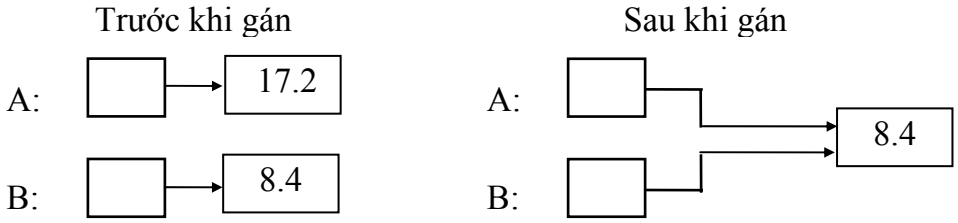
```
        float x;
        float y;
public:
        point() {x=0.0 ; y=0.0;}  // Phương thức xây dựng mặc nhiên
        point (float a, float b)  {x=a; y=b;} // Phương thức xây dựng bình thường
        point & operator= (const point & p ) //  Định nghĩa toán tử gán
        {
                x = p.x; y = p.y;  // Gán dữ liệu
                return * this;
        }
 }; // term
```

Sự khác nhau cuối cùng của phép gán là ở cách thức tiến hành gán trị. Xét lệnh gán của Pascal "A := B", ở Pascal cũng như một số ngôn ngữ khác, điều này có nghĩa là: "Gán bản sao của giá trị của biến B cho biến A". Bây giờ ta lại xét lệnh gán "A = B" của SNOBOL4. Trong SNOBOL4 thì nó có nghĩa là: "Tạo một biến tên A tham chiếu tới ĐTDL mà B đã tham chiếu". Trong SNOBOL4 cả A và B cùng trỏ tới một ĐTDL.

Pascal A := B (Sao chép ĐTDL khi gán)



SNOBOL4 A = B (Sao chép sự trỏ đến ĐTDL khi gán)



Cách thực hiện lệnh gán của SNOBOL4 rõ ràng là đã tạo ra một sự lắm tên.

2.8.2 Sự khởi tạo biến

Khởi tạo một biến là gán cho biến đó một giá trị đầu tiên.

Một biến khi được tạo ra thì sẽ được cấp phát ô nhớ nhưng nó vẫn chưa được khởi tạo. Khi nó được gán một giá trị đầu tiên thì mới được khởi tạo.

Các biến chưa được khởi tạo là nguồn gốc của các lỗi lập trình. Khi một biến được cấp phát ô nhớ mà chưa được khởi tạo thì trong ô nhớ của nó cũng có một giá trị ngẫu nhiên nào đó. Thường là một **giá trị rác** (Khi một ĐTDL nào trước đó đã bị hủy bỏ nhưng giá trị của ĐTDL này trong ô nhớ vẫn còn, giá trị này gọi là giá trị rác). Điều nguy hiểm là giá trị rác này vẫn là một giá trị hợp lệ. Vì thế chương trình có thể xử lý trên giá trị rác này một cách bình thường và chúng ta không thể kiểm sóat được kết quả xử lý đó.

Vì tính chất nghiêm trọng như đã nói trên của biến chưa được khởi tạo, các ngôn ngữ lập trình có thể sử dụng các giải pháp sau để khắc phục:

1.- **Nếu biến chưa được khở tạo thì sẽ có giá trị NULL**: Khi một biến mới được tạo ra, ô nhớ cấp phát cho nó phải chứa một dãy các bit biểu diễn cho một giá trị "NULL". Tùy thuộc vào kiểu của biến mà giá trị NULL này sẽ có một giá trị cụ thể, ví dụ nếu là biến số thì NULL là 0, nếu là biến chuỗi kí tự thì NULL là chuỗi rỗng, nếu biến là logic thì NULL là FALSE...

2.- **Khởi tạo biến ngay sau khi nó vừa được tạo ra** là một cách lập trình tốt và trong một số ngôn ngữ mới đều cung cấp phương tiện để làm điều này một cách dễ dàng. Trong ngôn ngữ Pascal một biến được khởi tạo đồng thời với việc khai báo được gọi là **biến có giá trị đầu** hay còn gọi là **hằng định kiểu.**

Ví dụ:

```
const i:integer=10;
    a: ARRAY[1..3,1..2] Of Integer = ((11, 12), (21, 22), (31, 32));
var j:integer;
begin
 writeln(i);  i:= i+1;  writeln(i);
 for i:=1 to 3 do begin
    for j:=1 to 2 do write(a[i,j]:5);
     writeln;
 end; end.
```

2.9 CÂU HỎI ÔN TẬP

1. Xét về mặt cấu trúc thì có các loại đối tượng dữ liệu nào?
2. Thế nào là một đối tượng dữ liệu sơ cấp?
3. Thế nào là một đối tượng dữ liệu có cấu trúc?
4. Đối tượng dữ liệu tường minh là gì?
5. Đối tượng dữ liệu ẩn là gì?
6. Kể tên các mối liên kết của đối tượng dữ liệu.
7. Thế nào là một biến?
8. Thế nào là một hằng?
9. Kiểu dữ liệu là gì?
10. Khi đặc tả một kiểu dữ liệu, chúng ta phải đặc tả những điều gì?
11. Cho ví dụ về một phép toán gây ra hiệu ứng lề.
12. Khi cài đặt một kiểu dữ liệu, chúng ta phải chỉ rõ những điều gì?
13. Mục đích của sự khai báo là gì?
14. Thế nào là kiểm tra kiểu?
15. Khi có sự không tương thích về kiểu thì chương trình dịch phải làm gì?
16. Kể tên các phương pháp kiểm tra kiểu.

17. Kiểm tra kiểu tĩnh được tiến hành trong lúc nào?
18. Kiểm tra kiểu động được tiến hành trong lúc nào?
19. Nêu các điểm mạnh của kiểm tra kiểu tĩnh.
20. Nêu các điểm yếu của kiểm tra kiểu tĩnh.
21. Nêu các điểm mạnh của kiểm tra kiểu động.
22. Nêu các điểm yếu của kiểm tra kiểu động.
23. Thông tin về kiểu trong kiểm tra kiểu tĩnh được lưu trữ ở đâu?
24. Thông tin về kiểu trong kiểm tra kiểu động được lưu trữ ở đâu?
25. Kiểm tra kiểu động được thực hiện trong ngôn ngữ nào?
26. Kiểm tra kiểu tĩnh được thực hiện trong ngôn ngữ nào?
27. Phép gán trị có trả về một kết quả không?
28. Thế nào là khởi tạo một biến?
29. Nếu trong một biểu thức có sử dụng một biến chưa được khởi tạo thì có thể đánh giá (định trị) được biểu thức đó không?

CHƯƠNG 3: KIỂU DỮ LIỆU SƠ CẤP

3.1 TỔNG QUAN

3.1.1 Mục tiêu

Sau khi học xong chương này, sinh viên cần phải nắm:

- Khái niệm về kiểu dữ liệu sơ cấp.
- Đặc tả và phương pháp cài đặt kiểu dữ liệu sơ cấp trong các ngôn ngữ lập trình.
- Một số kiểu dữ liệu sơ cấp cụ thể như: kiểu số, ký tự, logic…

3.1.2 Nội dung cốt lõi

- Kiến thức tổng quan về kiểu dữ liệu sơ cấp.
- Một vài kiểu dữ liệu sơ cấp: kiểu số, liệt kê, logic, ký tự.

3.1.3 Kiến thức cơ bản cần thiết

Kiến thức và kĩ năng lập trình căn bản, kiến thức chương 2.

3.2 ĐỊNH NGHĨA KIỂU DỮ LIỆU SƠ CẤP

Kiểu dữ liệu sơ cấp là một kiểu dữ liệu mà các ĐTDL của nó là các ĐTDL sơ cấp.

Nói chung các ngôn ngữ lập trình đều có các kiểu dữ liệu sơ cấp sau: số nguyên (*integer, int…*), số thực (*real, float, double…*), ký tự (*char, character…*), logic (*bool, boolean…*) và kiểu liệt kê.

3.3 SỰ ĐẶC TẢ CÁC KIỂU DỮ LIỆU SƠ CẤP

3.3.1 Thuộc tính của kiểu dữ liệu sơ cấp

Thuộc tính cơ bản nhất của bất kỳ một ĐTDL sơ cấp nào chính là kiểu dữ liệu của nó. Đối với một số kiểu dữ liệu cụ thể thì có thể có thêm các thuộc tính bổ sung để đặc trưng cho kiểu đó.

3.3.2 Giá trị của kiểu dữ liệu sơ cấp

Tập hợp các giá trị của một kiểu dữ liệu sơ cấp luôn là một **tập hợp có thứ tự** và có một **giá trị nhỏ nhất** và một **giá trị lớn nhất**.

Chính nhờ tính chất có thứ tự của tập giá trị sơ cấp nên trong thao tác sắp xếp dữ liệu, khóa sắp xếp thường thuộc kiểu dữ liệu sơ cấp.

Ví dụ kiểu dữ liệu *integer* là một tập hợp hữu hạn các số nguyên (dĩ nhiên là có thứ tự), từ một số nguyên nhỏ nhất đến một số nguyên lớn nhất. Số nguyên nhỏ nhất và số nguyên lớn nhất là các số nguyên tương ứng với các giá trị nguyên nhỏ nhất và lớn nhất có thể biểu diễn một cách thuận tiện trong bộ nhớ của máy tính.

3.3.3 Phép toán trên kiểu dữ liệu sơ cấp

Do tập giá trị sơ cấp có thứ tự, nên trong tất cả các kiểu dữ liệu sơ cấp đều có các phép toán quan hệ. Ngoài ra còn có các phép toán nhận vào một số đối số thuộc kiểu sơ cấp và trả về một giá trị sơ cấp cùng kiểu. Tuy nhiên cần hết sức lưu ý rằng tập các giá trị sơ cấp có giá trị nhỏ nhất và giá trị lớn nhất, cho nên đôi khi giá trị trả về của phép toán không nằm trong giới hạn của tập giá trị sơ cấp, điều này sẽ gây ra sự sai sót trong chương trình.

3.4 CÀI ĐẶT CÁC KIỂU DỮ LIỆU SƠ CẤP

3.4.1 Tổ chức dữ liệu trong bộ nhớ

Người ta thường sử dụng việc tổ chức dữ liệu dưới phần cứng của máy tính để biểu diễn cho các giá trị dữ liệu của kiểu dữ liệu sơ cấp.

Lý do của việc lựa chọn này rất đơn giản: Nếu biểu diễn bộ nhớ của phần cứng được sử dụng thì các phép toán cơ bản trên dữ liệu của kiểu này có thể được thực hiện bởi các phép toán do phần cứng cung cấp. Mà các phép toán được thiết kế bởi phần cứng sẽ có tốc độ thực hiện nhanh. Ngược lại, nếu ta sử dụng sự biểu diễn bởi phần mềm thì phải sử dụng các phép toán mô phỏng bởi phần mềm mà tốc độ thực hiện sẽ chậm hơn.

Tuy nhiên, việc sử dụng biểu diễn bởi phần cứng cũng có yếu điểm là tập các giá trị sẽ bị hạn chế.

Ví dụ để biểu diễn một số nguyên trong bộ nhớ, ta có thể sử dụng hai phương pháp:

1.- Sử dụng cách biểu diễn một số nguyên của phần cứng, chẳng hạn sử dụng 16 bit để biểu diễn cho một số nguyên. Với phương pháp này thì ta có thể sử dụng luôn các phép tính số học trên số nguyên (+, -, *, DIV, MOD) đã được thiết kế cho phần cứng. Ưu điểm của phương pháp này là các phép tính số học có tốc độ thực hiện nhanh. Nhược điểm của phương pháp là tập giá trị các số nguyên chỉ có 65535 số (từ -32768 đến 32767).

2.- Sử dụng một cấu trúc dữ liệu nào đó để biểu diễn cho một số nguyên, chẳng hạn sử dụng một chuỗi kí tự, trong đó mỗi kí tự lưu trữ môt chữ số. Ưu điểm của phương pháp là tập các giá trị nguyên sẽ rất lớn (số các chữ số trong một nguyên có thể bằng chiều dài của chuỗi kí tự biểu diễn cho nó). Nhược điểm của phương pháp là chúng ta phải xây dựng các chương trình con để thực hiện các phép tính số học và dĩ nhiên tốc độ thực hiện của các chương trình con này sẽ chậm hơn các phép tính được xây dựng trong phần cứng.

Các thuộc tính (chủ yếu là kiểu dữ liệu) của ĐTDL sơ cấp được xử lý bằng 2 cách chính như sau:

1.- Các thuộc tính của ĐTDL có thể được xác định trong khi biên dịch bởi trình biên dịch. Các thuộc tính này sẽ được lưu trữ trong bộ dịch của ngôn ngữ (chẳng hạn bảng danh biểu) và khi cần sẽ tìm lại các thuộc tính này để sử dụng. Đó là phương pháp thông dụng trong các ngôn ngữ biên dịch như FORTRAN, C và Pascal, nơi mà tính hiệu quả của việc sử dụng bộ nhớ và tốc độ thực hiện chương trình là những mục tiêu trên hết.

2.- Các thuộc tính có thể được lưu trữ trong bộ mô tả như là một phần của ĐTDL tại thời gian thực hiện. Đây là phương pháp thông dụng trong các ngôn ngữ thông dịch như LISP và SNOBOL4, nơi mà tính linh hoạt mềm dẻo là mục tiêu trước hết chứ không phải là tính hiệu quả.

3.4.2 Cài đặt phép toán

Mỗi một phép toán thao tác trên các ĐTDL của một kiểu dữ liệu sơ cấp đã cho có thể được cài đặt bằng một trong 3 cách như sau:

1.- Như là một phép toán phần cứng trực tiếp, nếu sự biểu diễn bộ nhớ của ĐTDL là sự biểu diễn của phần cứng. Ví dụ nếu các số nguyên được lưu trữ bằng cách dùng biểu diễn phần cứng cho số nguyên thì các phép toán như phép cộng, trừ và các phép toán số học khác của số nguyên có thể được thực hiện bằng cách dùng các phép toán số học cho số nguyên đã được xây dựng trong phần cứng.

2.- Như là một thủ tục hoặc hàm thực hiện các phép toán. Ví dụ phép toán lấy căn bậc hai thông thường không được cung cấp một cách trực tiếp như là một phép toán trong phần cứng ngay cả khi các số được biểu diễn bằng sự biểu diễn của phần cứng và vì vậy nó được cài đặt như là một chương trình con tính căn bậc hai. Nếu các ĐTDL không được biểu diễn bằng sự biểu diễn xác định bởi phần cứng thì tất cả các phép toán phải được mô phỏng bởi phần mềm.

3.- Như là một chuỗi các dòng mã lệnh dùng để thực hiện phép toán như là một dãy các phép toán phần cứng. Ví dụ hàm lấy trị tuyệt đối của một số được định nghĩa là:

$$ABS(x) = \begin{cases} x \text{ nêu } x \geq 0 \\ -x \text{ nêu } x < 0 \end{cases}$$ thường được cài đặt như là một chuỗi các mã lệnh:

1.- Nhận giá trị x từ bộ nhớ

2.- Nếu x>=0 thì bỏ qua chỉ thị kế tiếp

3.- Đặt x = -x

4.- Lưu x vào bộ nhớ

Trong đó mỗi một dòng mã lệnh được thực hiện bởi một phép toán trong phần cứng.

3.5 KIỂU DỮ LIỆU SỐ

Hầu hết các ngôn ngữ lập trình đều có các kiểu dữ liệu số, nhưng các chi tiết của sự đặc tả và phép cài đặt các kiểu này có nhiều điểm khác nhau. Kiểu số nguyên và kiểu số thực là phổ biến nhất bởi vì chúng dựa một cách trực tiếp vào phần cứng của máy tính.

3.5.1 Số nguyên

Sự đặc tả

Đặc tả các thuộc tính: Một ĐTDL của kiểu số nguyên không có thuộc tính nào khác ngoài kiểu của nó.

Đặc tả giá trị: Tập hợp các giá trị nguyên được xác định theo dạng là một tập hợp con có thứ tự hữu hạn của tập vô hạn các số nguyên đã được nghiên cứu trong toán học.

Giá trị nguyên lớn nhất đôi khi được biểu diễn như là một hằng xác định. Ví dụ trong Pascal là hằng MaxInt. Miền giá trị của kiểu số nguyên là tập các số nguyên từ -MaxInt đến MaxInt. Giá trị MaxInt được lựa chọn phản ánh giá trị nguyên lớn nhất có thể biểu diễn được trong phần cứng.

Đặc tả các phép toán: Trên ĐTDL nguyên thường có các nhóm phép toán chính như sau:

1.- Các phép tính số học

Các phép tính số học hai ngôi thường được định nghĩa là:

Bin_Op: Integer x Integer -> Integer.

Ở đây Bin_Op có thể là cộng (+), trừ (-), nhân (*), chia (/ hoặc DIV), lấy phần dư (MOD) hoặc một số phép toán tương tự khác.

Các phép tính số học một ngôi được định nghĩa: Unary_Op : Integer -> Integer

Ở đây Unary_Op có thể là âm (-), dương (+).

Các phép toán số học phổ biến khác thường được chứa trong thư viện chương trình con.

2.- Các phép toán quan hệ

Các phép toán quan hệ được định nghĩa là: Rel_Op : Integer x Integer -> Boolean

Ở đây Rel_Op có thể là bằng, khác, nhỏ hơn, lớn hơn, nhỏ hơn hoặc bằng, lớn hơn hoặc bằng. Phép toán quan hệ so sánh hai giá trị dữ liệu đối số và trả về kết quả là một đối tượng dữ liệu logic (đúng hoặc sai).

3.- Gán trị

Cũng như phép gán tổng quát, phép gán của số nguyên có thể trả về (với định nghĩa: **Assignment : Intger x Integer -> Integer)** hoặc không trả về một giá trị (với đinh nghĩa: **Assignment : Integer x Intger -> Void**) .

Sự cài đặt

Kiểu dữ liệu nguyên hầu hết được cài đặt một cách trực tiếp bằng cách dùng sự biểu diễn bộ nhớ được xác định bởi phần cứng và tập hợp các phép tính số học, các phép toán quan hệ nguyên thuỷ trong phần cứng cho các số nguyên. Thông thường sự biểu diễn này sử dụng một từ trong bộ nhớ hoặc một dãy các bytes để lưu trữ một số nguyên. Chẳng hạn ngôn ngữ Pascal đã sử dụng biểu diễn số nguyên bởi 1 từ (word) trong phần cứng của máy tính để biểu diễn cho một số integer.

3.5.2 Miền con của số nguyên

Sự đặc tả

Kiểu miền con của kiểu dữ liệu nguyên là một kiểu dữ liệu mà tập các giá trị của nó là một dãy các giá trị nguyên trong một khoảng giới hạn đã định.

Các dạng khai báo sau thường được sử dụng:

A : 1..10 (Pascal)

A : Integer Range 1..10 (Ada)

Như vậy về thuộc tính, kiểu miền con của kiểu số nguyên, có thuộc tính của kiểu số nguyên. Về giá trị, tập các giá trị của kiểu miền con được xác định rõ trong phép khai báo và cuối cùng, kiểu miền con cho phép sử dụng tập hợp phép toán như trong kiểu số nguyên bình thường.

Sự cài đặt

Kiểu miền con được cài đặt tương tự như cài đặt kiểu số nguyên.

Lợi ích của việc sử dụng kiểu miền con

Kiểu miền con có một ưu điểm nổi bật đó là kiểm tra kiểu tốt hơn kiểu số nguyên. Việc khai báo một biến kiểu miền con cho phép kiểm tra kiểu một cách nghiêm ngặt hơn khi thực hiện lệnh gán trị cho biến. Ví dụ để lưu trữ các tháng trong một năm ta có thể sử dung biến MONTH với khai báo kiểu miền con là MONTH: 1..12 thì lệnh gán MONTH := 0 là không hợp lệ và lỗi đó được tự động tìm thấy khi biên dịch. Nhưng nếu MONTH được khai báo là Integer thì lệnh gán trên là hợp lệ và lỗi chỉ có thể được tìm ra bởi người lập trình trong quá trình chạy thử.

3.5.3 Số thực dấu chấm động

Sự đặc tả

Tập hợp các giá trị thực dấu chấm động được xác định là một dãy số có thứ tự từ một số âm nhỏ nhất tới một số lớn nhất được xác định trong phần cứng của máy tính, nhưng các giá trị không được phân bố rời rạc đều trong giới hạn này.

Các phép tính số học, các phép toán quan hệ, phép gán đối với số thực cũng giống như đối với số nguyên. Một số phép toán khác cũng được các ngôn ngữ trang bị như là các hàm, chẳng hạn:

SIN : Real -> Real (Hàm SIN)

COS : Real -> Real (Hàm COSIN)

SQRT: Real -> Real (Hàm lấy căn bậc hai)

MAX : Real x Real -> Real (Hàm lấy giá trị lớn nhất)

Sự cài đặt

Sự biểu diễn bộ nhớ cho kiểu dữ liệu thực dấu chấm động dựa trên cơ sở biểu diễn phần cứng trong đó một ô nhớ được chia thành một phần định trị (mantissa) và một số mũ (exponent).

Các phép tính số học và các phép toán quan hệ trên kiểu số thực được hỗ trợ bởi phần cứng. Các phép toán khác phải được ngôn ngữ cài đặt như là các chương trình con.

3.6 KIỂU LIỆT KÊ

3.6.1 Đặt vấn đề

Trong lập trình, có một điều phổ biến là một biến có thể lấy một hoặc một số nhỏ các giá trị. Chẳng hạn biến NGAY_TRONG_TUAN chỉ lấy 7 giá trị biểu diễn cho "chủ nhật", "thứ hai", "thứ ba",..."thứ bảy". Tương tự biến GIOI_TINH chỉ có hai giá trị biểu diễn là "nam" và "nữ". Trong các ngôn ngữ như FORTRAN hay COBOL một biến như vậy phải có kiểu số nguyên và các giá trị được biểu diễn bởi các số nguyên chẳng hạn "chủ nhật" được biểu diễn bởi số 1, "thứ hai" được biểu diễn bởi số 2,... "nam" được biểu diễn bởi số 0 và "nữ" được biểu diễn bởi số 1.

Chương trình sử dụng các giá trị này như là các số nguyên và người lập trình phải nhớ sự tương ứng giữa các giá trị nguyên với "nghĩa" của chúng trong ứng dụng. Quả thực đây là một điều bất tiện và dễ gây ra sai sót.

Nhiều ngôn ngữ mới như Pascal hay Ada cho phép người lập trình tự đặt ra một kiểu dữ liệu bằng cách liệt kê ra một danh sách các giá trị của kiểu đó. Kiểu này gọi là kiểu liệt kê.

3.6.2 Sự đặc tả

Người lập trình định nghĩa kiểu liệt kê bằng cách liệt kê ra một danh sách các tên trực kiện thông qua sự khai báo. Các tên trực kiện trong danh sách là các giá trị của kiểu và thứ tự của chúng cũng được xác định nhờ thứ tự chúng xuất hiện trong danh sách. Chẳng hạn, ta có khai báo biến trong Pascal:

```
VAR
        NGAY_TRONG_TUAN : (Chu_nhat, Hai, Ba, Tu, Nam, Sau, Bay);
```

Vì có nhiều biến có cùng một kiểu liệt kê được dùng trong một chương trình nên người ta thường định nghĩa một liệt kê như là một kiểu có tên, sau đó sử dụng nó để xác định kiểu cho nhiều biến như trong Pascal:

```
TYPE
        NGAY = (Chu_nhat, Hai, Ba, Tu, Nam, Sau, Bay);
```

sau đó khai báo biến:

```
VAR
        NGAY_TRONG_TUAN, NGAY_LAM_VIEC: NGAY;
```

Trong sự khai báo trên, các tên trực kiện như Chu_nhat, Hai, Ba,… chính là các giá trị của kiểu và các giá trị này được sắp thứ tự như chúng đã được ghi ra, tức là Chu_nhat < Hai < Ba < … < Bay.

Chú ý rằng kiểu NGAY đã được định nghĩa thì có thể được dùng như một tên kiểu nguyên thuỷ (Integer chẳng hạn) và các hằng trực kiện như Chu_nhat, Hai, Ba, Tu,.. cũng được sử dụng như là các hằng trực kiện nguyên thuỷ (chẳng hạn "27"). Vì thế ta có thể viết:

```
        IF NGAY_TRONG_TUAN  = Hai THEN ...
```

Các phép toán cơ bản trong kiểu liệt kê là các phép toán quan hệ (bằng, nhỏ hơn, lớn hơn...), phép gán trị, phép toán cho giá trị đứng sau và giá trị đứng trước một giá trị trong dãy các hằng trực kiện của liệt kê.

3.6.3 Sự cài đặt

Biểu diễn bộ nhớ cho một ĐTDL kiểu liệt kê thường là không phức tạp. Mỗi giá trị trong liệt kê được biểu diễn bằng một số nguyên 0, 1, 2,... Ví dụ kiểu NGAY ở trên chỉ cần sử dụng 7 giá trị từ 0 đến 6, trong đó 0 biểu diễn cho Chu_nhat, 1 biểu diễn cho Hai, 2 biểu diễn cho Ba,...

Sự cài đặt các phép toán cơ bản cũng không phức tạp. Các phép quan hệ được cài đặt bằng cách sử dụng các phép toán quan hệ dưới phần cứng cho số nguyên như "=", "<", ">",... Phép toán lấy giá trị đứng sau một giá trị được cài đặt bằng cách lấy số nguyên biểu diễn cho giá trị đó cộng thêm 1 và có sự kiểm tra để thấy được kết quả không vượt quá giới hạn cho phép. Chẳng hạn để xác định giá trị sau Hai, ta lấy 1 (biểu diễn cho Hai) cộng với 1 được 2, mà 2 biểu diễn cho Ba, nên sau Hai là Ba, nhưng sau Bay thì không có giá trị nào vì tổng của 6 (biểu diễn cho Bay) với 1 được 7, vượt quá giới hạn cho phép của kiểu. Tương tự cho phép toán lấy giá trị đứng trước của một giá trị.

3.6.4 Lợi ích của việc sử dụng kiểu liệt kê

Kiểu liệt kê được đưa vào trong ngôn ngữ lập trình nhằm để giải quyết vấn đề được nêu ra trong phần đặt vấn đề. Từ đó ta có thể thấy rõ việc sử dụng kiểu liệt kê làm cho chương trình sáng sủa, trực quan, người lập trình không còn phải nhớ "nghĩa" của giá trị số và do vậy chương trình sẽ có độ chính xác cao hơn. Nói cách khác, kiểu liệt kê làm tăng tính dễ đọc, tính dễ viết và độ tin cậy của ngôn ngữ.

3.7 KIỂU LOGIC

Kiểu logic (*bool, boolean* hoặc *logical*) là kiểu dữ liệu phổ biến trong hầu hết các ngôn ngữ.

3.7.1 Sự đặc tả

Kiểu dữ liệu logic gồm các ĐTDL có một trong hai giá trị đúng hoặc sai. Trong Pascal và Ada, kiểu dữ liệu logic được xem một cách đơn giản như là một liệt kê được định nghĩa bởi ngôn ngữ. BOOLEAN = (FALSE, TRUE) trong đó xác định các tên "FALSE" và "TRUE" cho các giá trị của kiểu và xác định thứ tự FALSE < TRUE. Các phép toán phổ biến trên kiểu logic gồm có:

AND: Boolean X Boolean -> Boolean

OR: Boolean X Boolean -> Boolean

NOT: Boolean -> Boolean

3.7.2 Phép cài đặt

Chỉ cần một bit của bộ nhớ để lưu trữ một đố tượng dữ liệu logic. Tuy nhiên vì các bit đơn có thể không có địa chỉ riêng trong bộ nhớ nên ta phải sử dụng một đơn vị nhớ có địa chỉ như là byte hoặc word do đó các giá trị FALSE và TRUE có thể được biểu diễn bằng hai cách khác nhau:

1.- **Bit đặc trưng** (thông thường là bit dấu của sự biểu diễn số) với 0 biểu diễn cho FALSE và 1 biểu diễn cho TRUE, các bits còn lại trong byte hoặc word sẽ bị bỏ qua.

2.- **Sử dụng toàn bộ đơn vị nhớ** để ghi giá trị zero (tất cả các bit đề bằng 0) biểu diễn cho FALSE còn giá trị khác zero biểu diễn cho TRUE.

3.8 KIỂU KÝ TỰ

Hầu hết dữ liệu xuất và nhập đều có dạng ký tự và có sự chuyển đổi sang dạng dữ liệu khác trong quá trình nhập xuất. Chẳng hạn khi ta nhập một chữ số (hoặc một chuỗi chữ số) từ bàn phím vào một biến số trong chương trình thì đã có một sự chuyển đổi chữ số (chuỗi chữ số) thành số. Hay khi ta ghi một số ra máy in hoặc ra một tập tin văn bản thì đã có một sự chuyển đổi từ số thành chữ số (chuỗi chữ số). Tuy nhiên việc xử lý một số dữ liệu trực tiếp dưới dạng ký tự cũng cần thiết trong một số ứng dụng nào đó, chẳng hạn trong xử lý văn bản. Dữ liệu chuỗi ký tự có thể được cung cấp một cách trực tiếp thông qua kiểu chuỗi ký tự (như trong SNOBOL4, PL/1 và các ngôn ngữ mới khác) hoặc thông qua kiểu ký tự và chuỗi ký tự được xem như là một mảng các ký tự (như trong APL, Pascal và Ada. Chú ý Turbo Pascal đã có kiểu chuỗi ký tự).

3.8.1 Sự đặc tả

Kiểu ký tự là một liệt kê được định nghĩa bởi ngôn ngữ tương ứng với một tập hợp ký tự chuẩn được cho bởi phần cứng và hệ điều hành như tập các ký tự ASCII chẳng hạn.

Các phép toán trên dữ liệu ký tự bao gồm: các phép toán quan hệ, phép gán, và đôi khi có phép kiểm tra xem một ký tự có thuộc một lớp đặc biệt "chữ cái", "chữ số" hoặc lớp ký tự xác định nào đó.

3.8.2 Phép cài đặt

Các giá trị dữ liệu hầu như luôn được cài đặt một cách trực tiếp bởi phần cứng và hệ điều hành. Do đó các phép toán quan hệ cũng được biểu diễn một cách trực tiếp bởi phần cứng.

3.9 CÂU HỎI ÔN TẬP

1. Nêu định nghĩa kiểu dữ liệu sơ cấp.
2. Tập các giá trị của một kiểu sơ cấp có đặc điểm gì?
3. Có phải các ngôn ngữ lập trình thường sử dụng biểu diễn trong phần cứng để biểu diễn cho kiểu số nguyên?
4. Để cài đặt các phép toán số học trên kiểu dữ liệu số nguyên, có phải người ta phải thiết lập các chương trình con trong ngôn ngữ?
5. Tại sao người ta lại sử dụng kiểu liệt kê?
6. Tại sao người ta lại sử dụng kiểu miền con?

CHƯƠNG 4: KIỂU DỮ LIỆU CÓ CẤU TRÚC

4.1 TỔNG QUAN

4.1.1 Mục tiêu

Sau khi học xong chương này, sinh viên cần phải nắm:

- Khái niệm về kiểu dữ liệu có cấu trúc.
- Đặc tả và phương pháp cài đặt kiểu dữ liệu có cấu trúc.
- Các đặc tả, phương pháp tổ chức lưu trữ, cài đặt các phép toán của một số kiểu dữ liệu có cấu trúc như: vecto, mảng nhiều chiều, mẩu tin, chuỗi ký tự…

4.1.2 Nội dung cốt lõi

- Kiểu dữ liệu có cấu trúc.
- Các đặc tả, phương pháp lưu trữ, hình thức truy xuất, cài đặt các phép toán của một số kiểu dữ liệu có cấu trúc.

4.1.3 Kiến thức cơ bản cần thiết

Kiến thức và kĩ năng lập trình căn bản, kiến thức chương 2.

4.2 ĐỊNH NGHĨA KIỂU DỮ LIỆU CÓ CẤU TRÚC

Kiểu dữ liệu có cấu trúc hay còn gọi là cấu trúc dữ liệu (CTDL) là một kiểu dữ liệu mà các ĐTDL của nó là các ĐTDL có cấu trúc.

Như vậy CTDL là một tập hợp các ĐTDL có cấu trúc cùng với tập hợp các phép toán thao tác trên các ĐTDL đó. Các kiểu dữ liệu như mảng, mẩu tin, chuỗi, ngăn xếp (stacks), danh sách, con trỏ, tập hợp và tập tin là các CTDL.

4.3 SỰ ĐẶC TẢ KIỂU CẤU TRÚC DỮ LIỆU

4.3.1 Sự đặc tả các thuộc tính

Các thuộc tính chủ yếu của CTDL bao gồm:

Số lượng phần tử

Số lượng các phần tử của một CTDL cho biết kích thước của CTDL, số lượng này có thể cố định hoặc thay đổi tuỳ loại CTDL.

*Một CTDL được gọi là có **kích thước cố định** nếu số lượng các phần tử không thay đổi trong thời gian tồn tại của nó.*

Ví dụ các kiểu mảng, mẩu tin là các CTDL có kích thước cố định.

*Một CTDL được gọi là có **kích thước thay đổi** nếu số lượng các phần tử thay đổi một cách động trong thời gian tồn tại của nó.*

Ví dụ ngăn xếp, danh sách, tập hợp, chuỗi ký tự và tập tin là các CTDL có kích thước thay đổi. Các phép toán cho phép thêm hoặc bớt các phần tử của cấu trúc làm thay đổi kích thước của cấu trúc.

Kiểu của mỗi một phần tử

Mỗi một phần tử của CTDL có một kiểu dữ liệu nào đó, ta gọi là kiểu phần tử. Kiểu phần tử có thể là một kiểu dữ liệu sơ cấp hoặc một CTDL. Các phần tử trong cùng một CTDL có thể có kiểu phần tử giống nhau hoặc khác nhau.

*Một CTDL được gọi là **đồng nhất** nếu tất cả các phần tử của nó đều có **cùng một kiểu.***

Ví dụ mảng, chuỗi ký tự, tập hợp và tập tin là các CTDL đồng nhất .

*Một CTDL được gọi là **không đồng nhất** nếu các phần tử của nó có kiểu **khác nhau.***

Ví dụ mẩu tin là CTDL không đồng nhất.

Tên để dùng cho các phần tử được lựa chọn

Để lựa chọn một phần tử của CTDL cho một xử lý nào đó người ta thường sử dụng một tên. Đối với cấu trúc mảng, tên có thể là một chỉ số nguyên hoặc một dãy chỉ số; đối với mẩu tin, tên là một tên trường. Một số kiểu cấu trúc dữ liệu như ngăn xếp và tập tin cho phép truy nhập đến chỉ một phần tử đặc biệt (phần tử đầu tiên hoặc phần tử hiện hành).

Số lượng lớn nhất các phần tử

Đối với CTDL có kích thước thay đổi như chuỗi ký tự hoặc ngăn xếp, đôi khi người ta quy định thuộc tính kích thước tối đa của cấu trúc để giới hạn số lượng các phần tử của cấu trúc.

Tổ chức cấu trúc

Tổ chức phổ biến nhất là một dãy tuần tự của các phần tử. Vector (mảng một chiều), mẩu tin, chuỗi ký tự, ngăn xếp, danh sách và tập tin là các CTDL có tổ chức kiểu này.

Một số cấu trúc còn được mở rộng thành dạng "nhiều chiều" ví dụ mảng nhiều chiều, mẩu tin mà các phần tử của nó là các mẩu tin, danh sách mà các phần tử của nó là danh sách.

4.3.2 Các phép toán trên cấu trúc dữ liệu

Một số các phép toán đặc thù của CTDL:

Phép toán lựa chọn phần tử của cấu trúc

Phép toán lựa chọn một phần tử là phép toán truy nhập đến một phần tử của CTDL và làm cho nó có thể được xử lý bởi một phép toán khác.

Có hai loại lựa chọn:

***Lựa chọn ngẫu nhiên** (hay còn gọi là lựa chọn trực tiếp) là sự lựa chọn một phần tử tùy ý của cấu trúc dữ liệu được truy nhập thông qua một cái tên.*

Ví dụ để lựa chọn một phần tử nào đó của mảng, ta chỉ ra chỉ số của phần tử đó, để lựa chọn một phần tử của mẩu tin ta sử dụng tên của phần tử đó.

Lựa chọn tuần tự *là sự lựa chọn trong đó phần tử được lựa chọn là một phần tử đứng sau các phần tử đã được lựa chọn khác theo tuần tự của việc xử lý hoặc là lựa chọn một phần tử đặc biệt nào đó.*

Ví dụ lựa chọn tuần tự các phần tử trong một tập tin hay lựa chọn một phần tử trên đỉnh của ngăn xếp.

Các phép toán thao tác trên toàn bộ cấu trúc dữ liệu

Là các phép toán có thể nhận các CTDL làm các đối số và sản sinh ra kết quả là các CTDL mới. Chẳng hạn phép gán một mẩu tin cho một mẩu tin khác hoặc phép hợp hai tập hợp.

Thêm / bớt các phần tử

Là các phép toán cho phép thêm vào CTDL hoặc loại bỏ khỏi CTDL một số phần tử. Các phép toán này sẽ làm thay đổi số lượng các phần tử trong một CTDL. Việc thêm vào hay loại bỏ một phần tử thường phải chỉ định một vị trí nào đó.

Tạo / hủy CTDL

Là các phép toán tạo ra hoặc xóa bỏ một CTDL.

4.4 SỰ CÀI ĐẶT CÁC CẤU TRÚC DỮ LIỆU

4.4.1 Biểu diễn bộ nhớ

Sự biểu diễn bộ nhớ cho một CTDL bao gồm:

- Bộ nhớ cho các phần tử của cấu trúc.

- Bộ mô tả để lưu trữ một số hoặc tất cả các thuộc tính của cấu trúc.

Có hai phương pháp để biểu diễn bộ nhớ là:

Biểu diễn tuần tự

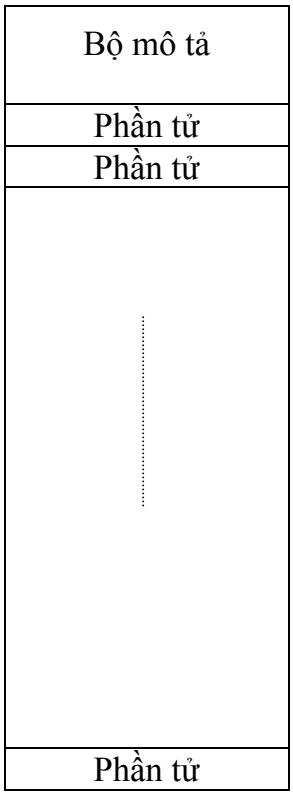
Biểu diễn tuần tự là sự biểu diễn, trong đó CTDL được lưu trữ như một khối các ô nhớ liên tiếp nhau, bắt đầu bằng bộ mô tả sau đó là các phần tử.

Đây là phương pháp được dùng cho các CTDL có kích thước cố định hoặc có kích thước thay đổi nhưng đồng nhất. Chẳng hạn có thể dùng biểu diễn tuần tự để biểu diễn cho mảng, mẩu tin,…

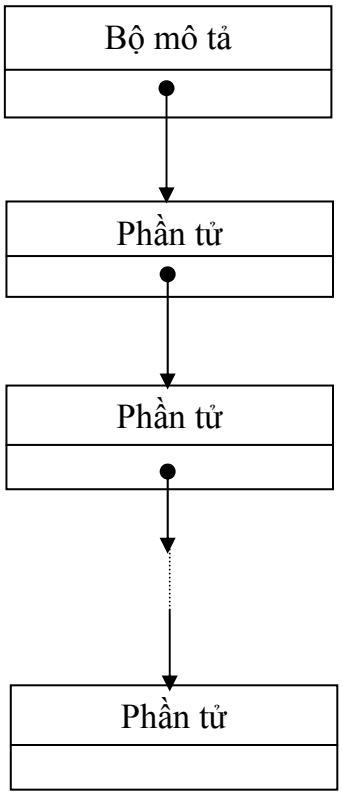
Biểu diễn liên kết

Biểu diễn liên kết là sự biểu diễn, trong đó CTDL được lưu trữ trong nhiều khối ô nhớ tại các vị trí khác nhau trong bộ nhớ, mỗi khối liên kết với khối khác thông qua một con trỏ gọi là con trỏ liên kết.

Phương pháp này thường được sử dụng cho các CTDL có kích thước thay đổi. Chẳng hạn có thể dùng biểu diễn liên kết để biểu diễn cho danh sách, ngăn xếp,…



Biểu diễn tuần tự



Biểu diễn liên kết

4.4.2 Cài đặt các phép toán trên cấu trúc dữ liệu

Phép toán lựa chọn một phần tử là phép toán cơ bản nhất trong các phép toán trên CTDL. Như trên đã trình bày, có hai cách lựa chọn là lựa chọn ngẫu nhiên và lựa chọn tuần tự và hai cách biểu diễn bộ nhớ là biểu diễn tuần tự và biêu diễn liên kết. Vì vậy ở đây chúng ta sẽ xét cách thực hiện mỗi một phương pháp lựa chọn với mỗi một phương pháp biểu diễn bộ nhớ.

Đối với biểu diễn tuần tự

Như trên đã trình bày, trong cách biểu diễn tuần tự, một khối ô nhớ liên tục sẽ được cấp phát để lưu trữ tòan bộ CTDL. Trong đó, vị trí đầu tiên của khối ô nhớ được gọi là **địa chỉ cơ sở**. Khoảng cách từ địa chỉ cơ sở đến vị trí của phần tử cần lựa chọn được gọi là **độ dời** của phần tử.

Cách thức truy xuất, được cho bởi tên hoặc chỉ số của phần tử (chẳng hạn chỉ số của một phần tử của mảng), sẽ xác định cách tính độ dời của phần tử như thế nào.

Để lựa chọn **ngẫu nhiên** một phần tử cần phải xác định vị trí thực của phần tử (tức là địa chỉ của ô nhớ lưu trữ phần tử đó) theo công thức:

Vị trí thực của phần tử = Địa chỉ cơ sở + độ dời của phần tử.

Lựa chọn **tuần tự** một dãy các phần tử của cấu trúc có thể theo các bước:

- Để chọn phần tử đầu tiên ta dùng cách tính địa chỉ cơ sở cộng với độ dời như đã nói ở trên.

- Đối với các phần tử tiếp theo trong dãy, cộng kích thước của phần tử hiện hành với vị trí của phần tử hiện hành để được vị trí của phần tử kế tiếp.

Đối với biểu diễn liên kết

Như trên đã trình bày, các khối ô nhớ trong biểu diễn liên kết được bố trí rời rạc nhau, khối này nối với khối kia bằng con trỏ và lúc đầu chỉ nắm được con trỏ tới khối đầu tiên. Do đó việc đi đến các khối luôn phải xuất phát từ khối đầu tiên.

Để lựa chọn **ngẫu nhiên** một phần tử trong cấu trúc liên kết cần phải duyệt một dãy các khối, từ khối đầu tiên đến khối cần lựa chọn.

Lựa chọn **tuần tự** một dãy các phần tử được thực hiện bằng cách lựa chọn phần tử đầu tiên như đã nói ở trên và sau đó từ phần tử hiện hành, duyệt theo con trỏ để đến phần tử kế tiếp.

4.5 VÉCTƠ

4.5.1 Định nghĩa véctơ

*Véctơ (còn gọi là mảng một chiều) là một CTDL bao gồm một số **cố định** các phần tử có kiểu **giống nhau** được tổ chức thành một dãy tuần tự các phần tử.*

Như vậy véctơ là một CTDL có kích thước cố định và đồng nhất.

4.5.2 Sự đặc tả và cú pháp

Đặc tả thuộc tính của véctơ

Các thuộc tính của một véctơ là:

- **Số lượng các phần tử**, luôn được chỉ rõ bằng cách cho tập chỉ số. Tập chỉ số này thông thường được cho bởi một miền con các số nguyên, trong trường hợp đó, **số lượng** các phần tử bằng **số nguyên cuối cùng - số nguyên đầu tiên + 1**. Một cách tổng quát thì tập chỉ số có thể là kiểu liệt kê nào đó, trong trường hợp này, số lượng phần tử bằng số giá trị trong kiểu liệt kê. Cũng có những ngôn ngữ chỉ định rõ số lượng các phần tử như ngôn ngữ C chẳng hạn.

- **Kiểu dữ liệu của mỗi một phần tử,** thường được viết rõ trong khai báo.

- **Chỉ số được sử dụng để lựa chọn mỗi một phần tử**. Nếu tập chỉ số được cho bởi một miền con của tập các số nguyên thì số nguyên đầu tiên chỉ định phần tử đầu tiên số nguyên thứ 2 chỉ định phần tử thứ 2 ...Nếu tập chỉ số là một liệt kê thì giá trị đầu tiên trong liệt kê là chỉ số của phần tử đầu tiên. Nếu ngôn ngữ chỉ định rõ số lượng các phần tử thì 0 là chỉ số của phần tử đầu tiên.

Khai báo véctơ trong Pascal là **ARRAY [<tập chỉ số>] OF <kiểu phần tử>.**

Ví dụ VAR a: ARRAY[1..10] OF real;

Khai báo này xác định 1 véctơ a có 10 phân tử là các số real. Các phần tử này được lựa chọn bởi các chỉ số từ 1 đến 10.

Miền giá trị của chỉ số không nhất thiết bắt đầu từ 1, ví dụ

Var b: ARRAY [-5..10] OF integer; Với khai báo này thì b là một véctơ có 16 phần tử (10 – (-5) + 1 = 16). Các phần tử được lựa chọn nhờ các chỉ số từ -5 đến 10.

Miền giá trị của chỉ số không nhất thiết là miền con của số nguyên, nó có thể là một liệt kê bất kỳ (hoặc 1 miền con của một liệt kê). Ví dụ:

```
Type
        Ngay = (Chu_nhat, Hai, Ba, Tu, Nam, Sau, Bay);
var
        c : ARRAY [Ngay] OF Integer ;
```

Khai báo này xác đinh véctơ c có 7 phần tử là các số integer, các phần tử của c được lựa chọn nhờ các "chỉ số" từ Chu_nhat đến Bay.

Khai báo véctơ trong ngôn ngữ C là **<kiểu phần tử> <tên biến> [<số lượng phần tử>]**.

Ví dụ int d[10];

Khai báo này xác định véctơ d có 10 phần tử các số int, các phần tử này được lựa chọn nhờ các chỉ số từ 0 đến 9.

Đặc tả các phép toán trên véctơ

Các phép toán trên véctơ bao gồm:

Phép toán **lựa chọn một phần tử** của véctơ là **phép lấy chỉ số**, được viết bằng tên của véctơ theo sau là chỉ số của phần tử được lựa chọn đặt trong cặp dấu []. Như vậy phép lựa chọn một phần tử của véctơ là phép **lựa chọn trực tiếp**.

Ví dụ, với các khai báo trong các ví dụ thuộc phần đặc tả thuộc tính nói trên,

Các phần tử của véctơ a được lựa chọn bằng cách viết a[1], a[2], …, a[10].

Các phần tử của véctơ b được lựa chọn bằng cách viết b[-5], b[-4], …, b[10].

Các phần tử của véctơ c được lựa chọn bằng cách viết c[Chu_nhat], c[Hai], …, c[Bay].

Các phần tử của véctơ d được lựa chọn bằng cách viết d[0], d[1], …, d[9].

Chỉ số có thể là một hằng hoặc một biến (nói chung là một biểu thức), ví dụ a[i] hay a[i+2]. Nhờ chỉ số là một biểu thức nên việc lập trình trở nên đơn giản hơn nhiều nhờ tính khái quát của chỉ số.

Ví dụ để in ra giá trị của 10 phần tử trong véctơ a, thay vì ta phải viết 10 lệnh in các phần tử cụ thể theo kiểu writeln(a[1]); writeln(a[2]); writeln(a[3]); … ta chỉ cần viết một lệnh for i:=1 to 10 do writeln(a[i]);

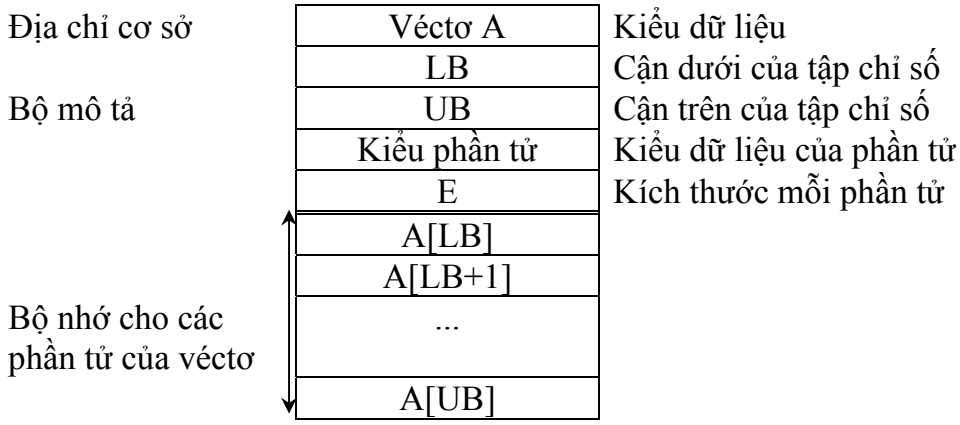
Các phép toán khác trên véctơ bao gồm các phép toán tạo và hủy bỏ véctơ, gán hai véctơ cho nhau và các phép toán thực hiện như các phép toán số học trên từng cặp 2 véctơ có cùng kích thước. Chẳng hạn phép cộng 2 véctơ (cộng các phần tử tương ứng). Tùy thuộc vào ngôn ngữ mà các phép toán này có hoặc không có.

4.5.3 Cài đặt một véctơ

Biểu diễn bộ nhớ

Biểu diễn bộ nhớ **tuần tự** được sử dụng để biểu diễn cho một véctơ.

Mô hình sau minh họa cho sự biểu diễn bộ nhớ của véctơ A : ARRAY[LB..UB] OF <kiểu phần tử>.



Khối ô nhớ để lưu trữ một véctơ có hai phần: **bộ mô tả** và **bộ nhớ dành cho các phần tử** của véctơ. Trong bộ mô tả lưu trữ kiểu dữ liệu của cấu trúc (véctơ A), cận dưới của tập chỉ số (LB - Lower Bound), cận trên của tập chỉ số (UB - Upper Bound), kiểu dữ liệu của phần tử và kích thước mỗi phần tử (E). Bộ nhớ dành cho các phần tử của véctơ lưu trữ liên tiếp các phần tử, từ phần tử đầu tiên (A[LB]) cho đến phần tử cuối cùng (A[UB]). Do các phần tử có cùng một kiểu nên các ô nhớ dành cho các phần tử có kích thước bằng nahu.

Địa chỉ của ô nhớ đầu tiên trong khối gọi là địa chỉ cơ sở.

Giải thuật thực hiện các phép toán

Phép toán lựa chọn một phần tử được thực hiện bằng cách **tính vị trí của phần tử** cần lựa chọn theo công thức:

$$\textbf{Vị trí của phần tử thứ i} = \propto + D + (i - LB) * E$$

Trong đó i là chỉ số của phần tử cần lựa chọn, $\propto$ là địa chỉ cơ sở của khối ô nhớ (địa chỉ word hoặc byte đầu tiên của khối ô nhớ dành cho véctơ) D là kích thước của bộ mô tả, LB là cận dưới của tập chỉ số và E là kích thước của mỗi một đối tượng dữ liệu thành phần (số word hoặc byte cần thiết để lưu trữ một phần tử).

Nếu chỉ số là một giá trị của kiểu liệt kê chứ không phải số nguyên thì hiệu i-LB phải được tính toán một cách thích hợp (chẳng hạn sử dụng hiệu của hai số thứ tự tương ứng của i và LB trong liệt kê).

Phép gán một véctơ cho một véctơ khác có cùng thuộc tính được thực hiện bằng cách sao chép nội dung trong khối ô nhớ biểu diễn véctơ thứ nhất sang khối ô nhớ biểu diễn véctơ thứ hai.

Các phép toán trên toàn bộ véctơ được thực hiện bằng cách sử dụng các vòng lặp xử lý tuần tự các phần tử của véctơ.

4.6 MẢNG NHIỀU CHIỀU

Ma trận (mảng hai chiều) được xem như là một véctơ của các véctơ. Một mảng 3 chiều được xem như là một véctơ của các ma trận...

4.6.1 Sự đặc tả và cú pháp

Đặc tả thuộc tính

Mảng nhiều chiều tương tự như véctơ nhưng chỉ có một thuộc tính khác véctơ là mỗi một chiều phải có một tập chỉ số tương ứng.

Chẳng hạn khai báo cho một mảng hai chiều có thể đươc viết dưới dạng

ARRAY[LB1..UB1, LB2..UB2] OF <Kiểu phần tử>

Trong đó tập chỉ số 1 có các giá trị từ LB1 đến UB1, tập chỉ số 2 có các giá trị từ LB2 đến UB2.

Như vậy số lượng các phần tử của mảng hai chiều sẽ là **(UB1-LB1+1)*(UB2-LB2+1)**

Ví dụ sự khai báo của Pascal:

M= array [1..3, -1..2] of Integer;

Sự khai báo này cho ta thấy mảng M có hai chiều, chiều thứ nhất được xác định bởi tập chỉ số 1..3 và chiều thứ hai được xác định bởi tập chỉ số -1..2. Có thể xem đây là một ma trận có 3 dòng và 4 cột, như vậy sẽ có 12 phần tử, mỗi phần tử có thể lưu trữ một số integer.

Đối với các mảng có số chiều nhiều hơn hai thì cách làm cũng tương tự như mảng hai chiều.

Đặc tả phép toán

Phép lựa chọn một phần tử được thực hiện bằng cách chỉ ra tên mảng và chỉ số của mỗi một chiều.

Chẳng hạn để lựa chọn một phân tử của ma trận ta viết tên ma trận, theo sau là cặp chỉ số dòng, cột phân cách nhau bởi dấu phẩy và đặt trong cặp dấu [], ví dụ M[2,0].

Như vậy phép lựa chọn một phần tử của mảng nhiều chiều là phép **lựa chọn trực tiếp**.

4.6.2 Sự cài đặt

Sự biểu diễn bộ nhớ

Sự biểu diễn bộ nhớ đối với mảng nhiều chiều tương tự như sự biểu diễn bộ nhớ đối với véctơ. Nghĩa là cũng sử dụng sự **biểu diễn tuần tự** và khố ô nhớ được chia làm hai phần: bộ mô tả và bộ nhớ cho các phần tử. Bộ mô tả của mảng giống bộ mô tả của véctơ ngoại trừ mỗi một chiều có một cận dưới và cận trên của tập chỉ số của chiều đó. Trong bộ nhớ dành cho các phần tử ta cũng lưu trữ liên tiếp các phần tử theo một trật tự nào đó.

Với ma trận, về mặt logic thì ma trận là một bảng gồm m dòng và n côt, mỗi một ô là một phần tử, nhưng bộ nhớ lại chỉ gồm các ô liên tiếp nhau, vì thế ta phải lưu trữ ma trận theo trật tự dòng hoặc theo trật tự cột.

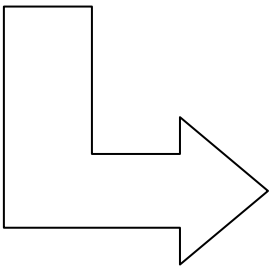
Lưu trữ theo **trật tự dòng** có nghĩa là trong bộ nhớ dành cho các phần tử ta lưu trữ tuần tự các phần tử trong dòng thứ nhất, tiếp đến là các phần tử trong dòng thứ hai... cho đên dòng cuối cùng.

Lưu trữ theo **trật tự cột** nghĩa là trong bộ nhớ dành cho các phần tử ta lưu trữ tuần tự các phần tử trong cột thứ nhất, tiếp đến là các phần tử trong cột thứ hai... cho đến cột cuối cùng.

Chẳng hạn với khai báo M: ARRAY [1..3,-1..2] OF Integer; ta có hình ảnh biểu diễn trong bộ nhớ như các hình sau:

Cấu trúc logic của ma trận M

M[1,-1]	M[1,0]	M[1,1]	M[1,2]
M[2,-1]	M[2,0]	M[2,1]	M[2,2]
M[3,-1]	M[3,0]	M[3,1]	M[3,2]

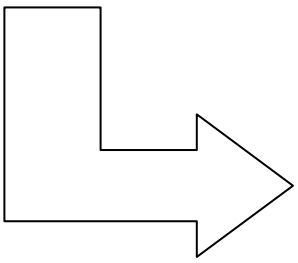


Lưu trữ ma trận M theo **trật tự dòng**

Bộ mô tả		
	Ma trận M	Kiểu dữ liệu
	LB1 (= 1)	Cận dưới của tập chỉ số thứ nhất
	UB1 (= 3)	Cận trên của tập chỉ số thứ nhất
	LB2 (= -1)	Cận dưới của tập chỉ số thứ hai
	UB2 (= 2)	Cận trên của tập chỉ số thứ hai
Bộ nhớ cho Các phần tử	M[1,-1]	
	M[1,0]	Dòng thứ nhất
	M[1,1]	
	M[1,2]	
	M[2,-1]	Dòng thứ hai
	M[2,0]	
	...	.....
	M[3,2]	

Cấu trúc logic của ma trận M

M[1,-1]	M[1,0]	M[1,1]	M[1,2]
M[2,-1]	M[2,0]	M[2,1]	M[2,2]
M[3,-1]	M[3,0]	M[3,1]	M[3,2]



Lưu trữ ma trận M theo **trật tự cột**

Bộ mô tả		
	Ma trận M	Kiểu dữ liệu
	LB1 (= 1)	Cận dưới của tập chỉ số thứ nhất
	UB1 (= 3)	Cận trên của tập chỉ số thứ nhất
	LB2 (= -1)	Cận dưới của tập chỉ số thứ hai
	UB2 (= 2)	Cận trên của tập chỉ số thứ hai
Bộ nhớ cho Các phần tử	M[1,-1]	
	M[2,-1]	Cột thứ nhất
	M[3,-1]	
	M[1,0]	
	M[2,0]	Cột thứ hai
	M[3,0]	
	...	.....
	M[3,2]	

Giải thuật thực hiện phép toán

Để thực hiện phép toán lựa chọn phần tử, ta sử dụng công thức tính vị trí của phần tử trong bộ nhớ.

Với cách lưu trữ theo trật tự dòng của ma trận M, để tính vị trí của M[i,j], đầu tiên ta xác định số dòng cần nhảy qua: (i-LB1) nhân với độ dài của mỗi dòng để xác định vị trí bắt đầu của dòng thứ i và sau đó tìm vị trí thứ J trong dòng này như đối với 1 véctơ. Như vậy, vị trí của phần tử M[i,j] được tính bởi:

$$\text{Vị trí của M [i,j]} = \propto + D + (i-LB1) \times S + (j-LB2) \times E$$

Trong đó: $\propto$ là địa chỉ cơ sở.
D là độ lớn của bộ mô tả.
S là độ lớn của mỗi dòng = (UB2 - LB2 +1) x E.
LB1 là cận dưới của chỉ số thứ nhất.
LB2,UB2 tương ứng là cận dưới và cận trên của chỉ số thứ hai.

Tương tự ta có thể thành lập công thức tính vị trí của phần tử M[i,j] trong trường hợp ma trận M được tổ chức lưu trữ theo trật tự cột.

Tổng quát hóa công thức này cho mảng nhiều chiều hơn là một điều đơn giản.

4.7 MẪU TIN

4.7.1 Định nghĩa mẩu tin

Mẩu tin là một CTDL bao gồm một số cố định các phần tử có kiểu khác nhau.

Như vậy, mẩu tin là một CTDL có **kích thước cố định** và **không đồng nhất**. Các phần tử của mẩu tin được gọi là các trường.

4.7.2 Sự đặc tả và cú pháp

Đặc tả thuộc tính

Các thuộc tính của một mẩu tin phải được chỉ rõ trong phép khai báo, chúng bao gồm:

1. Số lượng các phần tử.

2. Kiểu dữ liệu của các phần tử (Các phần tử có thể có kiểu khác nhau).

3. Mỗi phần tử được cho bởi tên phần tử (tên trường).

Cú pháp khai báo mẩu tin của Pascal:

```
Nhan_vien: RECORD
             Ma: Integer;  {Mã nhân viên}
             Ho_ten: String[25];
             Tuoi: Integer;  {Tuổi}
             Luong: Real;  {Hệ số lương}
           END
```

Việc khai báo này đặc tả một mẩu tin có 4 phần tử của các kiểu Integer, Real và String. Mỗi phần tử có một tên: Ma, Ho_ten, Tuoi và Luong. Để chọn một phần tử của mẩu tin ta sử dụng tên của phần tử (trường) đó, chẳng hạn trong Pascal, Nhan_vien.Luong là để truy xuất tới phần tử Luong của mẩu tin Nhan_vien.

Đặc tả phép toán

Lựa chọn một phần tử là phép toán cơ bản cuả mẩu tin. Phép toán này được thực hiện bằng cách chỉ ra tên trực kiện của phần tử.

Ví dụ để lựa chọn phần tử thứ 4 của mẩu tin Nhan_vien ta viết: Nhan_vien.Luong.

Phép toán lựa chọn một phần tử của mẩu tin là sự **lựa chọn trực tiếp**.

Mặc dù đều là lựa chọn trực tiếp, nhưng có khác biệt so với cách lựa chọn phần tử của véctơ. Điểm khác biệt ở đây là: đối với véctơ, ta có thể sử dụng giá trị của một biểu thức làm chỉ số, chẳng hạn VECTO[i+1], còn đối với mẩu tin thì bắt buộc phải chỉ rõ tên trực kiện, chứ không thể là biểu thức.

Ngoài phép toán lựa chọn phần tử, phép gán các mẩu tin có cùng cấu trúc là một phép toán phổ biến được các ngôn ngữ đưa vào. Chẳng hạn Nhan_vien := InputRec trong đó InputRec có các thuộc tính giống hệt Nhan_vien.

4.7.3 Sự cài đặt

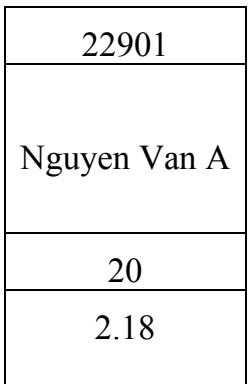
Biểu diễn bộ nhớ

Biểu diễn bộ nhớ **tuần tự** được sử dụng để lưu trữ một mẩu tin. Một khối liên tục các ô nhớ được dùng để lưu trữ cho một mẩu tin, trong khối đó, mỗi ô biểu diễn cho một trường. Có thể cũng cần sử dụng bộ mô tả riêng cho từng trường để lưu trữ thuộc tính của các trường đó. Do các trường có kiểu khác nhau nên ô nhớ dành cho chúng cũng có kích thước khác nhau.

Giải thuật thực hiên phép toán

Việc lựa chọn phần tử được thực hiện một cách dễ dàng vì tên trường được biết đến thông qua việc dịch chứ không phải được tính toán thông qua việc thực hiện như đối với véctơ. Việc khai báo mẩu tin còn cho phép xác định kích thước và vị trí của nó trong ô nhớ thông qua việc dịch. Kết quả là độ dời của phần tử bất kỳ có thể được tính thông qua việc dịch.

Chẳng hạn với mẩu tin Nhan_vien, các phần tử của nó được lưu trữ trong bộ nhớ như sau:



Vị trí của một phần tử bất kỳ được tính một cách dễ dàng. Chẳng hạn

Vị trí của Tuoi = α + Kích thước của Ma + Kích thước của Ho_ten.

Trong đó α là địa chỉ cơ sở của khối ô nhớ biểu diễn cho Nhan_vien.

Phép toán gán toàn bộ một mẩu tin cho một mẩu tin khác có cùng cấu trúc được thực hiện một cách đơn giản là copy nội dung khối ô nhớ biểu diễn cho mẩu tin thứ nhất sang khối ô nhớ biểu diễn cho mẩu tin thứ 2.

4.8 MẪU TIN CÓ CẤU TRÚC THAY ĐỔI

4.8.1 Đặc tả và khai báo

Trước hết ta xét ví dụ sau:

Giả sử trong một xí nghiệp có hai loại công nhân là công nhân trong biên chế và công nhân hợp đồng. Đối với công nhân trong biên chế thì lương sẽ được tính bằng số ngày công * mức lương tối thiểu * hệ số /20, những ngày nghỉ bảo hiểm xã hội, họ được trả lương bảo hiểm xã hội. Ngược lại công nhân hợp đồng chỉ được trả lương bằng số ngày công * đơn giá công nhật và họ không được trả lương bảo hiểm xã hội.

Ta thấy, hai loại công nhân này có những thông tin chung là họ tên, số ngày công, tiền lương và loại công nhân (biên chế hay hợp đồng). Mỗi loại công nhân lại có các thông tin riêng. Đối với công nhân trong biên chế, ta cần thêm các thông tin: hệ số lương và số ngày nghỉ bảo hiểm xã hội. Đối với công nhân hợp đồng, ta cần thêm thông tin về đơn giá công nhật.

Nếu sử dụng mẩu tin bình thường để lưu trữ thông tin về hai loại công nhân này, ta cần tất cả 7 trường để lưu trữ 4 thông tin chung và 3 thông tin riêng. Khối ô nhớ cần cấp phát phải đủ để lưu trữ cả 7 trường nhưng việc sử dụng khối ô nhớ lại bị dư, do đối với công nhân biên chế ta chỉ cần 6 trường, đối với công nhân hợp đồng ta chỉ cần 5 trường!

Đặc tả thuộc tính

Để giải quyết vấn đề lãng phí bộ nhớ, trong một số ngôn ngữ lập trình có một loại CTDL gọi là mẩu tin có cấu trúc thay đổi.

Mỗi một cấu trúc sẽ có một số trường giống nhau cho mọi loại mẩu tin và một số trường khác nhau cho từng loại mẩu tin. Các trường giống nhau gọi là phần chung hay phần tĩnh, các trường khác nhau này gọi là phần động hay phần thay đổi của mẩu tin.

Chẳng hạn đối với bài toán nêu trên thì mỗi công nhân được lưu trong một mẩu tin, có các trường thuộc phần chung đó là Ho_Ten, Ngay_Cong, Tien_Luong. Ngoài ra tùy thuộc vào loại công nhân là biên chế hay hợp đồng mà có các trường riêng. Đối với công nhân trong biên chế ta cần thêm các trường He_So và Nghi_Bhxh để lưu trữ hệ số lương và số ngày nghỉ bảo hiểm xã hội. Đối với công nhân hợp đồng ta chỉ cần thêm một trường là Gia_Cong_Nhat để lưu trữ giá công nhật cho mỗi người.

Khai báo trong Pascal như sau:

```
TYPE
    loai_cong_nhan = (bien_che,hop_dong);
VAR
   Cong_Nhan : RECORD
       ho_ten: String[20];
       ngay_cong: Real;
       luong: Real;
       CASE loai: loai_cong_nhan OF
            bien_che:
                 (he_so: Real;
                 nghi_bhxh:Real);
            hop_dong:
```

```
                    (gia_cong_nhat: Real);
    END;
```

Khai báo trên định nghĩa một mẩu tin có cấu trúc thay đổi. Mẩu tin luôn luôn có các trường Ho_Ten, Ngay_Cong, Luong và Loai. Khi giá trị của Loai = "bien_che" thì mẩu tin còn có các trường He_So và Nghi_Bhxh, trong khi đó nếu giá trị của Loai = "hop_dong" thì nó lại có trường Gia_Cong_Nhat.

Đặc tả phép toán

Phép toán lựa chọn các phần tử của mẩu tin có cấu trúc thay đổi cũng giống như mẩu tin bình thường. Chẳng hạn ta có thể sử dụng Cong_Nhan.Luong, Cong_Nhan.He_So hay Cong_Nhan.Gia_Cong_Nhat. Tuy nhiên các trường thuộc phần động chỉ tồn tại trong một thời điểm nhất định do đó khi chúng ta truy xuất tới một tên trường mà nó không tồn tại thì sẽ bị lỗi. Trường Loai trong ví dụ trên là rất quan trọng vì nó chỉ ra phần động nào của mẩu tin được sử dụng trong quá trình thực hiện chương trình. Người đọc có thể tham khảo ví dụ tương đối hoàn chỉnh viết bằng Pascal.

```
uses crt;
Const luong_toi_thieu = 290000;
Type
  Loai_cong_nhan = (bien_che, hop_dong);
  Cong_nhan = Record
            ho_ten : String[20];
            Ngay_cong : real;
            luong : real;
            Case loai: Loai_cong_nhan of
                 bien_che: (He_so, so_ngay_nghi_BHXH : real);
                 hop_dong: (don_gia: real);
  end;
  danh_sach_cong_nhan = Array[1..10] of cong_nhan;
Var
  n : integer; ho_so : danh_sach_cong_nhan;

{Nhập danh sách công nhân, và các thông tin liên quan đến lao động}
Procedure Nhap (var ho_so: danh_sach_cong_nhan; var n: integer);
Var
  i: integer;
  loaicn : char;
Begin
  write('So cong nhan: '); readln(n);
  For i:=1 to n do with ho_so[i] do begin
      Writeln('Cong nhan ',i);
      Write('Ho va Ten: '); readln(ho_ten);
      Write('Loai cong nhan: A la bien che, B la hop dong ');
```

```
      readln(loaicn);
      If Upcase(loaicn) ='A' then loai := bien_che else loai := hop_dong;
      write('So ngay cong: '); readln(ngay_cong);
      if loai = bien_che then begin
        write('He so: '); readln(he_so);
        write('So ngay nghi bao hiem: '); readln(so_ngay_nghi_BHXH);
      end else begin
        write('Don gia hop dong: '); readln(don_gia);
      end;
  end; { with Ho_so[i] }
end; {nhap}
{Tính lương cho từng công nhân, theo công thức của từng loại công nhân}
Procedure Tinh_luong (var ho_so: danh_sach_cong_nhan; n: integer);
Var
  i : integer; luong_binh_quan: real;
begin
  for i:=1 to n do with ho_so[i] do begin
      if loai = bien_che then begin {tính lương của công nhân biên chế}
        luong_binh_quan := he_so * luong_toi_thieu/20;
        luong :=  ngay_cong * luong_binh_quan +
                  so_ngay_nghi_BHXH * luong_binh_quan*0.80;
      end else {tính lương của công nhân hợp đồng}
        luong := ngay_cong * don_gia;
   end;  { with Ho_so[i] }
end; {Tinh_luong }

Procedure In_luong (ho_so: danh_sach_cong_nhan; n: integer);
Var
  i : integer;
begin
  for i:=1 to n do with ho_so[i] do begin
     Write(ho_ten:25);
     If loai = bien_che then write('Bien che':10)
     else write('Hop dong':10);
     write(ngay_cong:5:1);
     if loai = bien_che then begin
        write(he_so:5:1);
        write(so_ngay_nghi_BHXH:5:1);
     end else
        write(don_gia:10:2);
     writeln(luong:10:2);
```

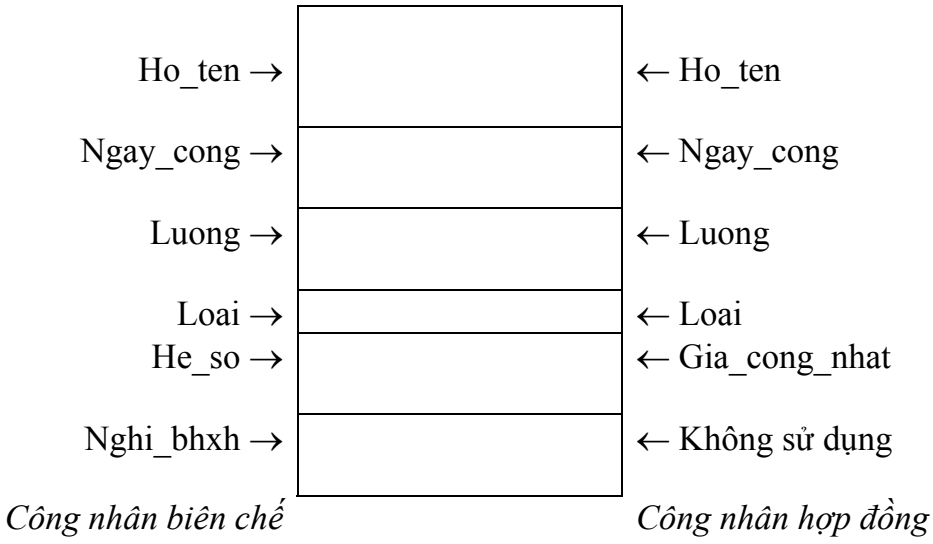
```
   end; { with Ho_so[i] }
   end; { In_luong }
begin {Chuong trinh chinh}
  nhap(ho_so,n);
  tinh_luong(ho_so,n);
  in_luong(ho_so,n);
  readln;
end.
```

4.8.2 Cài đặt mẩu tin có cấu trúc thay đổi

Biểu diễn bộ nhớ

Biểu diễn tuần tự sẽ được sử dụng để biểu diễn cho một mẩu tin có cấu trúc thay đổi.

Thông qua việc dịch, tổng bộ nhớ cần để lưu các phần tử của mỗi một phần động được xác định và bộ nhớ được cấp phát đủ để lưu trữ mẩu tin với phần động lớn nhất. Chẳng hạn với mẩu tin cong_nhan ta có mô hình lưu trữ như trong hình vẽ sau:



Vì khối ô nhớ đủ lớn để lưu trữ phần động lớn nhất nên có đủ chỗ cho bất kỳ một phần động nào nhưng đối với những phần động nhỏ hơn sẽ không sử dụng tới một số ô nhớ đã được cấp phát.

Với mẩu tin có cấu trúc thay đổi, rõ ràng ta đã tiết kiệm được một số ô nhớ so với mẩu tin bình thường.

Giải thuật thực hiện phép toán

Lựa chọn một phần tử của phần động cũng giống như lựa chọn một phần tử bình thường, qua việc dịch thì độ dời của phần tử được lựa chọn sẽ được tính toán và qua việc thực hiện thì độ dời được cọng vào địa chỉ cơ sở của khối để xác định vị trí của phần tử.

4.9 CHUỖI KÝ TỰ:

Chuỗi ký tự là cấu trúc dữ liệu bao gồm một dãy các ký tự.

Như vậy, kiểu chuỗi ký tự là một kiểu đồng nhất, còn về kích thước thì có thể cố định hoặc thay đổi tùy theo ngôn ngữ. Kiểu dữ liệu chuỗi ký tự là một kiểu quan trọng mà hầu hết các ngôn ngữ đều có.

4.9.1 Đặc tả và cú pháp:

Đặc tả thuộc tính

Tùy ngôn ngữ, có thể có 3 cách đặc tả đối với kiểu chuỗi ký tự:

a/ Độ dài được khai báo cố định: Chuỗi ký tự có thể có độ dài (kích thước) **cố định** được khai báo trong chương trình. Mọi giá trị được gán cho đối tượng dữ liệu chuỗi đều có cùng độ dài như vậy. Khi một chuỗi thực được gán cho đối tượng dữ liệu mà độ dài của chuỗi thực khác độ dài được khai báo thì sẽ có sự điều chỉnh độ dài của chuỗi thực bằng cách cắt bớt các ký tự dư hoặc thêm vào các ký tự trắng để có được một chuỗi có độ dài đúng như khai báo.

Đây là kỹ thuật cơ bản được dùng trong COBOL trong đó từ khóa PICTURE được dùng để xác định số lượng ký tự, ví dụ: Last_Name PICTURE X(20) khai báo biến chuỗi ký tự Last_Name chứa một chuỗi 20 ký tự.

Trong Pascal (chuẩn) kiểu dữ liệu chuỗi ký tự không có. Thay vào đó kiểu chuỗi ký tự được biểu diễn như là một véctơ của các ký tự Last_Name: PACKED ARRAY [1..20] OF Char.

b/ Độ dài thay đổi trong một giới hạn đã được khai báo: Chuỗi ký tự có thể có độ dài cực đại được khai báo trước trong chương trình nhưng giá trị thực của đối tượng dữ liệu được lưu trữ có thể là chuỗi có độ dài ngắn hơn, thậm chí có thể là chuỗi rỗng. Trong quá trình thực hiện độ dài của giá trị chuỗi của đối tượng dữ liệu có thể thay đổi, nó cũng sẽ bị cắt nếu vượt giới hạn đã khai báo.

Đây là kỹ thuật được dùng trong PL/1 (và cả trong Turbo Pascal).

c/ Độ dài không giới hạn: Chuỗi ký tự có thể có **độ dài bất kỳ** và độ dài có thể thay đổi một cách động thông qua quá trình thực hiện.

Đây là kỹ thuật được dùng trong SNOBOL4.

Trong ba phương pháp nói trên thì hai phương pháp đầu cho phép cấp phát bộ nhớ cho mỗi một đối tượng dữ liệu chuỗi được xác định tại thời gian dịch. Đối vơi phương pháp thứ ba thì sử dụng cấp phát bộ nhớ động tại thời gian thực hiện. Các phương pháp khác nhau cũng đòi hỏi các phép toán khác nhau trên chuỗi. Sau đây là một số phép toán chủ yếu.

Đặc tả phép toán

Trên chuỗi ký tự, thường có các phép toán sau:

a/ Phép ghép nối (concatennation)

Ghép là phép toán nhập hai chuỗi ký tự tạo ra một chuỗi mới ví dụ nếu "//" là ký hiệu của phép ghép thì "BLOCK"//"HEAD" cho ra "BLOCKHEAD". Turbo Pascal sử dụng toán tử "+" cho phép toán ghép chuỗi.

b/ Các phép toán quan hệ trên chuỗi

Các phép toán quan hệ thông thường như bằng, nhỏ hơn, lớn hơn... trên kiểu ký tự có thể được mở rộng cho chuỗi ký tự. Tập hợp các ký tự cơ bản luôn luôn có một thứ tự. Mở rộng thứ tự này cho chuỗi ký tự thành thứ tự alphabe trong đó chuỗi A nhỏ hơn chuỗi B nếu ký tự đầu tiên của A nhỏ hơn ký tự đầu tiên của B hoặc hai ký tự đầu tiên tương ứng của chúng bằng nhau và ký tự thư hai của A nhỏ hơn ký tự thứ hai của B... Nếu chuỗi A ngắn hơn chuỗi B thì A được mở rộng bằng cách thêm vào các ký tự trắng cho dài bằng B để so sánh.

c/ Chọn chuỗi con dùng chỉ số chỉ vị trí của ký tự

Nhiều ngôn ngữ cung cấp một phép toán chọn chuỗi con của một chuỗi bằng cách cho vị trí của ký tự đầu tiên và ký tự cuối cùng của nó (hoặc vị trí của ký tự đầu tiên và chiều dài của chuỗi con). Ví dụ trong FORTRAN, lệnh NEXT = STR(6:10) là gán 5 ký tự, bắt đầu từ vị trí thứ 6 đến vị trị thứ 10 của chuỗi STR cho biến chuỗi NEXT.

d/ Định dạng nhập - xuất

Định dạng nhập xuất là phép toán dùng để thay đổi dạng nhập vào hoặc xuất ra của các chuỗi ký tự. Nhập xuất có định dạng là nét nổi bật của FORTRAN và PL/1.

e/ Chọn chuỗi con dùng so mẫu

Thông thường người ta không biết vị trí của một chuỗi con cần chọn trong một chuỗi lớn hơn nhưng quan hệ của nó với một chuỗi con khác thì có thể biết. Ví dụ chuỗi các chữ số sau dấu chấm thập phân hay chuỗi đứng sau một khoảng trống. Phép so mẫu có một đối số thứ nhất để xác định dạng của chuỗi con cần chọn (chẳng hạn độ dài của nó). Đối số thứ hai của phép toán so mẫu là chuỗi ký tự dùng để tìm trong chuỗi (chẳng hạn dấu chấm thập phân). Như vậy kết quả của phép toán so mẫu là chọn được một chuỗi con bắt đầu từ sau dấu chấm thập phân và có độ dài đã cho.

4.9.2 Cài đặt

Biểu diễn bộ nhớ

Mỗi một phương pháp đặc tả chuỗi có một cách biểu diễn bộ nhớ tương ứng.

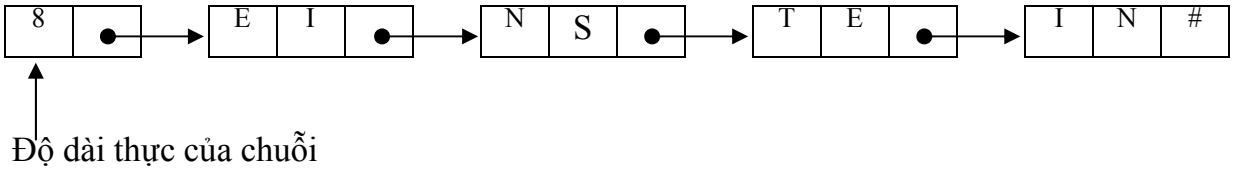
Đối với chuỗi có độ dài được khai báo cố định thì dùng véctơ của các ký tự. Ví dụ chuỗi được khai báo có độ dài 8 và được dùng để lưu trữ chuỗi EINSTEIN (cũng có 8 ký tự):

E	I	N	S	T	E	I	N

Đối với chuỗi có độ dài thay đổi trong một giới hạn đã được khai báo thì vẫn dùng véctơ của các ký tự, trong đó sử dụng hai ô làm bộ mô tả chứa giá trị thể hiện độ dài lớn nhất đã được khai báo và độ dài hiện hành của chuỗi. Ví dụ chuỗi được khai báo có độ dài 12 và được dùng để lưu trữ chuỗi EINSTEIN (có 8 ký tự):

12	8	E	I	N	S	T	E	I	N				
Độ dài khai báo	Độ dài thực											Các ô dư không sử dụng	

Đối với chuỗi có độ dài không giới hạn thì sử dụng biểu diễn bộ nhớ liên kết với bộ mô tả chứa độ dài hiện tại của chuỗi.



Độ dài thực của chuỗi

Giải thuật thực hiện các phép toán

Thông thường phần cứng hỗ trợ cho việc biểu diễn chuỗi có độ dài cố định nhưng đối với các biểu diễn khác cho chuỗi thì phải được mô phỏng bởi phần mềm. Các phép toán ghép, chọn chuỗi con và so mẫu phải mô phỏng bởi phần mềm.

4.10 CẤU TRÚC DỮ LIỆU CÓ KÍCH THƯỚC THAY ĐỔI

CTDL có kích thước thay đổi là một cấu trúc mà trong đó số lượng các phần tử có thể thay đổi một cách động trong quá trình thực hiện chương trình.

Một số kiểu chủ yếu của cấu trúc dữ liệu có kích thước thay đổi là:

4.10.1 Danh sách và cấu trúc danh sách

Danh sách là một CTDL tuyến tính với số lượng thay đổi của các phần tử có kiểu giống nhau.

Nếu các phần tử của một danh sách lại là một danh sách thì được gọi là cấu trúc danh sách (list structures).

Các phần tử có thể được thêm vào hoặc xóa khỏi một danh sách. Các phần tử có thể được lựa chọn từ một danh sách nhưng vì vị trí của phần tử trong danh sách có thể bị thay đổi do phép thêm và xóa các phần tử nên không thể sử dụng chỉ số để xác định phần tử. Thay vào đó, việc lựa chọn dựa trên cơ sở của mối quan hệ của vị trí của phần tử với danh sách chẳng hạn phần tử đầu, hai, ba, kế hặc cuối. Biểu diễn bộ nhớ liên kết cho danh sách và cấu trúc danh sách được dùng một cách phổ biến để phù hợp với sự thay đổi số lượng các phần tử.

4.10.2 Ngăn xếp và hàng đợi

Ngăn xếp là một danh sách mà trong đó việc lựa chọn, thêm, xóa phần tử được thực hiện ở 1 đầu của danh sách.

Do việc thêm, xóa phần tử chỉ được thực hiện ở một đầu của ngăn xếp, nên phần tử được **đưa vào sau**, sẽ được **lấy ra trước**. Do vậy ngăn xếp còn được gọi là cấu trúc dữ liệu kiểu LIFO (Last In, First Out).

Hàng đợi là một danh sách mà trong đó việc lựa chọn, và xóa phần tử được thực hiện ở một đầu còn việc thêm phần tử được thực hiện ở đầu khác của danh sách.

Do việc xóa phần tử được thực hiện ở một đầu (đầu của hàng) còn việc thêm phần tử được thực hiện ở cuối hàng, nên phần tử được **đưa vào trước**, sẽ được **lấy ra trước**. Do vậy hàng đợi còn được gọi là cấu trúc dữ liệu kiểu FIFO (First In, First Out).

Cả hai phương pháp biểu diễn tuần tự và liên kết đều được dùng cho ngăn xếp và hàng đợi.

4.11 CON TRỎ

4.11.1 Cấp phát tĩnh, cấp phát động và con trỏ

Cấp phát bộ nhớ (gọi tắt là cấp phát) là sự dành riêng các ô nhớ của bộ nhớ cho chương trình sử dụng.

Thông thường các ô nhớ được cấp phát để lưu trữ giá trị dữ liệu của biến. Có hai phương pháp cấp phát là **cấp phát tĩnh** và **cấp phát động**.

Cấp phát tĩnh là sự cấp phát ô nhớ cho ĐTDL được thực hiện trong quá trình dịch.

Trong khi biên dịch, thông qua sự khai báo biến, bộ dịch xác định được kiểu dữ liệu của ĐTDL nên sẽ dành sẵn một khối ô nhớ đủ lớn để lưu trữ ĐTDL của kiểu này.

Người lập trình sử dụng ô nhớ được cấp phát thông qua tên biến.

Khi khối chương trình, nơi khai báo biến kết thúc thì ô nhớ đã được cấp phát sẽ được tự động giải phóng.

Ưu điểm

Ưu điểm của cấp phát tĩnh là người lập trình **dễ sử dụng**, cụ thể là người lập trình chỉ cần khai báo biến, chương trình dịch sẽ tự động cấp phát và sau đó tự động giải phóng.

Nhược điểm

Nhược điểm của cấp phát tĩnh là việc sử dụng bộ nhớ **không tối ưu**, cụ thể là có thể cấp phát nhiều ô nhớ nhưng sử dụng không hết hoặc cấp phát thiếu.

Cấp phát động là sự cấp phát trong khi thực hiện chương trình.

Người lập trình phải viết lệnh cấp phát trong chương trình, khi thực hiện lệnh này thì bộ nhớ mới được cấp phát.

Sử dụng cấp phát động, người lập trình có thể ra lệnh giải phóng để thu hồi ô nhớ.

Để có thể cấp phát động, ta cần có một biến con trỏ hay còn gọi là biến kiểu tham chiếu. Biến con trỏ là một ĐTDL sơ cấp chứa địa chỉ của khối ô nhớ được cấp phát.

Người lập trình sử dụng ô nhớ được cấp phát thông qua biến con trỏ.

Ưu điểm

Ưu điểm nổi bật của cấp phát động là **sử dụng bộ nhớ một cách tối ưu**.

Nhược điểm

Nhược điểm của cấp phát động là **sự lắm tên**, có thể có nhiều tên biến con trỏ cùng tham chiếu đến một ô nhớ và do vậy làm giảm độ tin cậy của chương trình. Ngoài ra cũng gặp khó khăn khi sử dụng cấp phát động.

4.11.2 Sự đặc tả

Đặc tả thuộc tính

Có hai loại con trỏ khác nhau:

Con trỏ chỉ có thể tham chiếu tới các ĐTDL cùng kiểu

Đây là phương pháp được dùng trong Pascal và Ada.

Ví dụ trong Pascal:

Var p: ^integer chỉ ra rằng p là một biến con trỏ chứa địa chỉ của ô nhớ lưu trữ được một số integer.

Var q: ^VECT chỉ ra rằng q là một biến con trỏ chứa địa chỉ của khối ô nhớ của ĐTDL thuộc kiểu véctơ VECT nào đó.

Con trỏ có thể tham chiếu tới các ĐTDL khác kiểu nhau

Đây là cách được dùng trong các ngôn ngữ như SNOBOL4, nơi mà đối tượng dữ liệu mang bộ mô tả kiểu trong quá trình thực hiện và phép kiểm tra kiểu động được sử dụng.

Đặc tả phép toán

Các phép toán bao gồm:

Phép toán cấp phát ô nhớ động: Phép toán này dùng để cấp phát ô nhớ cho đối tượng dữ liệu mới và trả địa chỉ của ô nhớ đó về trong biến con trỏ. Đây là phép toán quan trọng nhất của kiểu con trỏ. Phép toán này có hai điểm khác biệt với việc tạo ra đối tượng dữ liệu tĩnh (bằng cách khai báo) là: Đối tượng dữ liệu được tạo ra không cần có tên vì nó được truy xuất thông qua con trỏ và đối tượng dữ liệu có thể được tạo ra một cách động trong quá trình thực hiện chương trình. Trong Pascal và Ada thì phép toán này có tên là NEW. Ví dụ NEW(p).

Phép toán truy xuất ô nhớ được cấp phát động: Để truy xuất đến giá trị dữ liệu lưu trong khối ô nhớ cấp phát động ta phải sử dụng địa chỉ của khối ô nhớ thông qua tên con trỏ (vì khối ô nhớ này không có tên). Ví dụ q^[5] là phần tử thứ 5 của véctơ Vect được trỏ bởi q.

Phép toán thu hồi ô nhớ được cấp phát động: Phép toán này cho phép giải phóng ô nhớ đã cấp phát. Trong Pascal, dùng phép toán DISPOSE.

Ví dụ sau trong Pascal minh hoạ tổng hợp các điều nói trên:

```
Type
Vect = ARRAY[1..10] of Integer;
{Lúc này bộ nhớ cho Vect chưa được cấp phát}
VAR
p: ^Vect;
{Khai báo p là một biến con trỏ chứa địa chỉ của khối ô
nhớ lưu trữ ĐTDL thuộc kiểu véctơ Vect. Khi dịch đến đây
thì ô nhớ cho p sẽ được cấp phát}
Begin
```

```
     NEW(p);
{Cấp phát ô nhớ cho véctơ và trả địa chỉ của ô nhớ này cho
biến con trỏ p (hay còn nói p trỏ tới khối ô nhớ này)}
     p^[5] := 20; {Truy xuất đến phần tử thứ 5 của véctơ}
     writeln(p^[5]);
     Dispose(p);  {Giải phóng ô nhớ đã cấp cho véctơ}
End.
```

4.11.3 Sự cài đặt

Có hai phương pháp biểu diễn bộ nhớ được sử dụng để biểu diễn cho một giá trị con trỏ:

Địa chỉ tuyệt đối

Giá trị con trỏ là địa chỉ ô nhớ thực của khối ô nhớ của ĐTDL.

Phương pháp này rất hiệu quả, bởi vì giá trị con trỏ tự nó quy định sự truy xuất trực tiếp tới đối tượng dữ liệu bằng cách dùng phép toán truy xuất bộ nhớ của phần cứng.

Địa chỉ tương đối

Đây là phương pháp cấp phát một vùng nhớ rộng với địa chỉ cơ sở của nó. Giá trị con trỏ là độ dời của ĐTDL. Địa chỉ của ĐTDL được tính bằng cách lấy địa chỉ cơ sở + độ dời của ĐTDL (tức là giá trị của con trỏ).

Phương pháp này thuận tiện cho việc quản lý bộ nhớ nhưng truy xuất đến ĐTDL chậm vì phải tính địa chỉ của khối ô nhớ biểu diễn cho ĐTDL.

4.12 TẬP HỢP

4.12.1 Đặc tả

Đặc tả thuộc tính

Tập hợp là một cấu trúc dữ liệu đồng nhất và có kích thước thay đổi.

Trong một tập hợp người ta không quan tâm đến thứ tự của các phần tử; giá trị các phần tử khác nhau.

Đặc tả phép toán

Các phép toán cơ bản trên tập hợp là:

1/ Kiểm tra sự tồn tại của một phần tử

Phép toán này dùng để xác định xem một giá trị X nào đó có phải là một phần tử của tập hợp S hay không.

2/ Thêm và bớt các phần tử cho tập hợp

Thêm giá trị X vào trong tập S, với điều kiện nó chưa là một phần tử của tập hợp. Xóa một giá trị dữ liệu X của tập S nếu nó là một phần tử của S. Hai phép toán này sẽ làm thay đổ kích thước của tập hợp.

3/ Phép hợp, giao và hiệu của 2 tập hợp

Đây là các phép toán được định nghĩa tương tự như trong toán học.

4.12.2 Cài đặt

Để cài đặt một tập hợp, ta có thể sử dụng một trong hai phương pháp sau:

Véctơ bit

Biểu diễn bộ nhớ

Tập hợp được biểu diễn bởi một chuỗi các bit. Cách tiếp cận này phù hợp cho một không gian nhỏ. Chẳng hạn ta có một không gian gồm n phần tử được đánh số thứ tự e_1, e_2, ... e_n. Một tập hợp các phần tử được chọn từ không gian này được biểu diễn bởi một véctơ có n bit, trong đó nếu bit thứ i có giá trị 1 thì phần tử e_i thuộc vào tập hợp, ngược lại bit thứ i có giá trị 0 thì e_i không thuộc tập hợp.

Giải thuật thực hiện các phép toán

Với cách biểu diễn này, việc thêm một phần tử vào trong tập hợp được thực hiện bằng cách cho bit tương ứng giá trị bằng 1. Việc xóa một phần tử trong tập hợp được thực hiện bằng cách cho bit tương ứng giá trị bằng 0. Phép kiểm tra một phần tử có thuộc tập hợp hay không được thực hiện bằng cách kiểm tra bit tương ứng có giá trị là 1 hay 0. Phép hợp của hai tập hợp tương ứng với phép toán logic OR của hai véctơ bit. Phép giao của hai tập hợp tương ứng với phép toán logic AND của hai véctơ bit. Hiệu của hai tập hợp tương ứng với phép toán logic AND của véctơ bit thứ nhất với phần bù của véctơ bit thứ hai. Các phép toán logic trên các véctơ bit đều được hỗ trợ bởi phần cứng.

Ví dụ Ta có một không gian bao gồm 5 phần tử 1,2,3,4,5. Khi đó

Tập hợp A = {1,2,4,5} được biểu diễn bởi véctơ (1,1,0,1,1)

Tập hợp B = {2,3,4} được biểu diễn bởi véctơ (0,1,1,1,0)

Do đó $A \cup B$ sẽ là tập {1,2,3,4,5} bởi vì (1,1,0,1,1) OR (0,1,1,1,0) = (1,1,1,1,1)

$A \cap B$ sẽ là tập hợp {2,4} bởi vì (1,1,0,1,1) AND (0,1,1,1,0) = (0,1,0,1,0)

A\B sẽ là tập hợp {1,5} bởi vì phần bù của (0,1,1,1,0) là (1,0,0,0,1) và

(1,1,0,1,1) AND (1,0,0,0,1) = (1,0,0,0,1)

Ưu điểm

Dễ dàng cài đặt các phép toán trên tập hợp với tốc độ thực hiện nhanh nhờ sử dụng các phép toán của phần cứng.

Nhược điểm

Không thể biểu diễn cho tập hợp mà các phần tử của nó có thể lấy từ một không gian lớn, có số lượng các phần tử bất kỳ.

Bảng băm

Biểu diễn bộ nhớ

Phương pháp này thích hợp cho các không gian lớn. Theo đó mỗi tập hợp được biểu diễn bởi một bảng băm (bảng băm mở). Mỗi phần tử của tập hợp được lưu trữ trong

các lô (bucket) của bảng băm nhờ vào hàm băm (mỗi lô là một danh sách liên kết, mỗi phần tử của danh sách chứa môt phần tử của tập hợp).

Giải thuật thực hiện các phép toán

Phép toán kiểm tra sự tồn tại của một phần tử trong tập hợp được thực hiện bằng cách sử dụng phép tìm kiếm một phần tử trong bảng băm.

Các phép toán thêm và bớt một phần tử của tập hợp được thực hiện bằng cách sử dụng các phép toán tương ứng là xen và xoá một phần tử của bảng băm.

Các phép toán hợp, giao và hiệu của hai tập hợp đòi hỏi phải có một sự cài đặt công phu hơn.

Ưu điểm

Có thể biểu diễn cho tập hợp bất kỳ, không giới hạn về kích thước.Các phép toán kiểm tra một phần tử thuộc tập hợp, thêm và bớt một phần tử thực hiện dễ dàng và khá hiệu quả.

Nhược điểm

Khó khăn trong việc cài đặt các phép toán hợp, giao và hiệu của hai tập hợp.

4.13 TẬP TIN

Tập tin là một CTDL có 2 tính chất đặc biệt.

1/ Lưu trữ trong bộ nhớ ngoài như đĩa hay băng từ do đó có thể lớn hơn hầu hết các CTDL khác.

2/ Thời gian tồn tại của nó lâu dài.

Tập tin tuần tự là một kiểu phổ biến nhất của tập tin nhưng nhiều ngôn ngữ còn cung cấp tập tin truy xuất trực tiếp và tập tin tuần tự có chỉ mục.

4.13.1 Tập tin tuần tự

Sự đặc tả

Tập tin tuần tự là một CTDL bao gồm một dãy tuyến tính các phần tử có cùng kiểu. Độ dài của tập tin là không giới hạn. Kiểu phần tử có thể là kiểu sơ cấp hoặc kiểu cấu trúc có kích thước cố định như mảng hoặc mẩu tin. Kiểu cấu trúc có kích tước thay đổi thông thường không thể là phần tử của tập tin (do đó không có tập tin của tập tin hay tập tin của ngăn xếp).

Một cách phổ biến, tập tin có thể được truy nhập theo một trong hai mode: READ hoặc WRITE. Trong cả hai mode này đều có một con trỏ tập tin (file position pointer) dùng để xác định vị trí của phần tử nào đó hoặc sau phần tử cuối cùng. Trong mode WRITE, con trỏ tập tin luôn luôn chỉ vào sau phần tử cuối cùng và phép toán duy nhất có thể là ghi một phần tử mới vào vị trí đó. Trong mode READ, con trỏ tập tin có thể chỉ vào bất kỳ vị trí nào trong tập tin và phép toán duy nhất là đọc phần tử đó. Trong cả hai mode, phép toán READ hoặc WRITE đều di chuyển con trỏ tập tin đến phần tử kế tiếp. Nếu con trỏ tập tin chỉ tới sau phần tử cuối cùng của tập tin thì tập tin được gọi là được chỉ tới cuối tập tin (end-of-file).

Các phép toán chủ yếu đối với tập tin tuần tự là:

1/ OPEN

Thông thường một tập tin phải được mở trước khi sử dụng. Phép toán OPEN chỉ ra tên của tập tin và mode truy xuất tập tin (READ hoặc WRITE). Nếu mode là READ thì tập tin phải chắc chắn là đã tồn tại. Hệ diều hành cung cấp đặc tính của tập tin, cấp phát ô nhớ cần thiết cho vùng nhớ đệm và đặt con trỏ tập tin vào phần tử đầu tiên. Nếu mode là WRITE thì hệ điều hành tạo một tập tin rỗng, nếu tập tin đã tồn tại thì xóa tất cả các phần tử của tập tin để nó rỗng, con trỏ tập tin chỉ vào vị trí đầu tập tin rỗng.

Ví dụ trong Pascal thủ tục RESET mở một tập tin để READ và thủ tục REWRITE mở một tập tin để WRITE.

2/ READ

Phép toán READ chuyển nội dung của phần tử hiện hành của tập tin (được chỉ định bởi con trỏ tập tin) vào biến được chỉ định trong chương trình.

3/ WRITE

Phép toán WRITE tạo ra một phần tử mới của tập tin tại vị trí hiện hành và chuyển nội dung của biến chương trình được chỉ định vào phần tử mới.

4/ Kiểm tra cuối tập tin

Là phép toán xác định xem vị trí của con trỏ tập tin có nằm sau phần tử cuối cùng của tập tin hay không.

5/ CLOSE

Khi việc xử lý tập tin đã hoàn tất thì nó phải được đóng lại. Thông thường tập tin được đóng một cách tự động khi chương trình kết thúc. Tuy nhiên nếu muốn thay đổi mode truy nhập tập tin từ WRITE sang READ hoặc ngược lại thì tập tin phải được đóng một cách tường minh bằng phép toán CLOSE và sau đó mở lại cho mode mới.

Phép cài đặt

Trong hầu hết các hệ máy tính, thì hệ điều hành chịu trách nhiệm chủ yếu về việc cài đặt tập tin bởi vì tập tin được tạo ra và sử dụng bởi nhiều ngôn ngữ lập trình khác nhau. Ngôn ngữ lập trình chỉ làm một việc là cung cấp những cấu trúc dữ liệu cần thiết để giao diện với hệ điều hành.

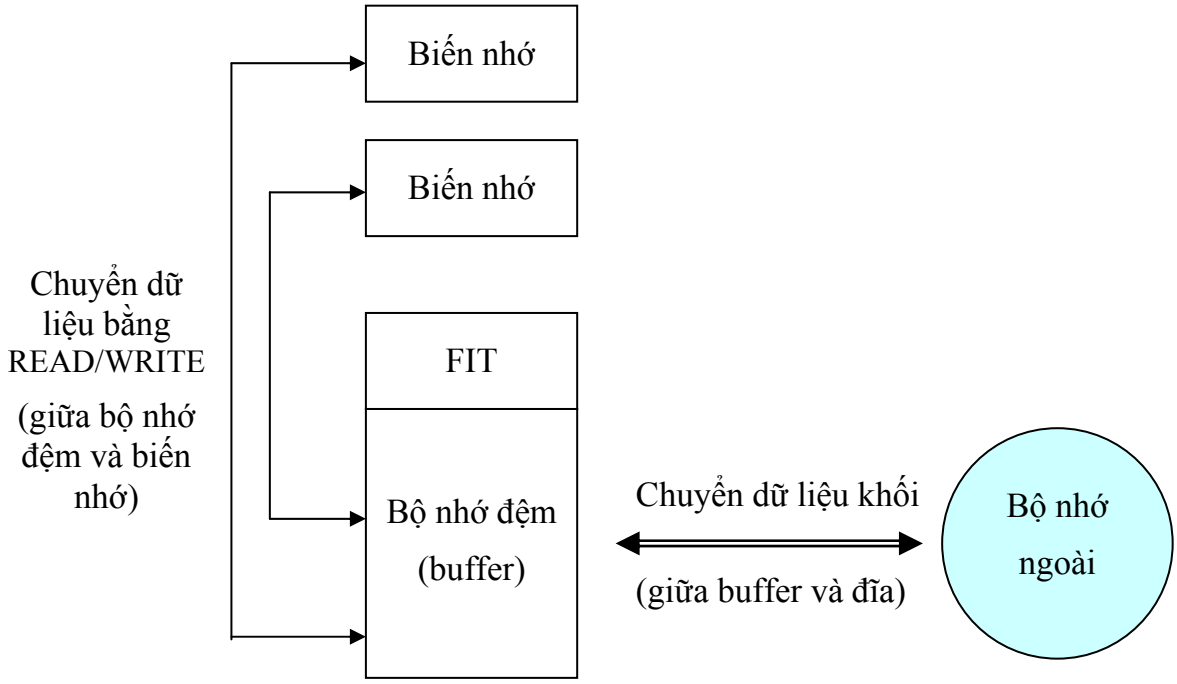
Các phép toán trên tập tin được cài đặt một cách chủ yếu bằng cách gọi các phép toán của hệ điều hành.

Khi chương trình mở một tập tin, thì bộ nhớ lưu trữ một bảng thông tin về tập tin (FIT) (File Information Table) và một bộ nhớ đệm (buffer) được cung cấp. Phép toán OPEN của hệ diều hành sẽ lưu trữ thông tin về vị trí và các đặc tính của tập tin vào trong bảng FIT.

Nếu tập tin được mở để ghi thì khi phép toán WRITE chuyển một phần tử để nối vào cuối tập tin, thì dữ liệu được gửi cho phép toán WRITE của hệ điều hành. Phép toán WRITE của hệ điều hành sẽ lưu dữ liệu vào trong vị trí có thể của bộ nhớ đệm. Khi trong bộ nhớ đệm đã tích lũy được một khối các phần tử thì khối đó sẽ được chuyển sang bộ nhớ ngoài (đĩa hoặc băng từ). Quá trình tiếp tục của phép toán WRITE được

thực hiện bằng cách lấp đầy bộ nhớ đệm cho đến khi một khối có thể được chuyển ra bộ nhớ ngoài.

Đối với READ thì ngược lại, một khối các phần tử của tập tin sẽ được chuyển sang bộ nhớ đệm và mỗi một phép toán READ được thực hiện bởi chương trình lại chuyển một phần tử từ bộ nhớ đệm sang biến chương trình cho đến khi bộ nhớ đệm trở thành rỗng thì một khối lại được chuyển từ bộ nhớ ngoài vào bộ nhớ đệm.



4.13.2 Tập tin văn bản

Tập tin văn bản là một tập tin của các ký tự. Đây là một loại tập tin rất thông dụng vì nó được sử dụng một cách dễ dàng trong tất cả các ngôn ngữ lập trình và các công cụ khác (Các loại tập tin khác không có được đặc điểm này). Tập tin văn bản cũng là một tập tin tuần tự nên các thao tác trên nó cũng tương tự như trên tập tin tuần tự. Ngoài ra còn có các phép toán đặc biệt khác cho phép chuyển đổi dữ liệu khác thành ký tự và ngược lại khi đọc hoặc ghi trên tập tin văn bản.

4.13.3 Tập tin truy xuất trực tiếp

Tập tin truy xuất trực tiếp là một tập tin được tổ chức sao cho bất kỳ một phần tử nào cũng được truy xuất một cách ngẫu nhiên. Để làm được điều đó mỗi một phần tử của nó phải có một khóa chẳng hạn khóa của mỗi phần tử là số thứ tự của nó trong tập tin. Để truy xuất phần tử bất kỳ, trước hết con trỏ của tập tin phải được di chuyển tới phần tử có khóa được chỉ định, sau đó phép toán READ hoặc WRITE mới được thực hiện. Phép toán WRITE có thể thay đổi nội dung đã có trong một phần tử đã tồn tại.

4.14 CÂU HỎI ÔN TẬP

1. Nêu định nghĩa kiểu dữ liệu có cấu trúc.

2. Nêu tên các thuộc tính của cấu trúc dữ liệu?
3. Thế nào là cấu trúc dữ liệu đồng nhất?
4. Thế nào là cấu trúc dữ liệu không đồng nhất?
5. Thế nào là cấu trúc dữ liệu có kích thước cố định?
6. Thế nào là cấu trúc dữ liệu có kích thước thay đổi?
7. Cho ví dụ về một cấu trúc dữ liệu đồng nhất.
8. Cho ví dụ về một cấu trúc dữ liệu không đồng nhất.
9. Cho ví dụ về một cấu trúc dữ có kích thước cố định.
10. Cho ví dụ về một cấu trúc dữ có kích thước không cố định.
11. Trên cấu trúc dữ liệu thường có các phép toán nào?
12. Kể tên các phương pháp lựa chọn một phần tử của cấu trúc dữ liêu?
13. Nêu tên các phương pháp biểu diễn cấu trúc dữ liệu trong bộ nhớ?
14. Phép toán lựa chọn trực tiếp (ngẫu nhiên) một phần tử của cấu trúc dữ liệu được biểu diễn tuần tự được thực hiện bằng cách nào?
15. Có phải kiểu véctơ (mảng một chiều) là một cấu trúc dữ liệu có kích thước cố định?
16. Cho biết công thức xác định số phần tử của một vectơ.
17. Cho biết công thức xác định địa chỉ (vị trí) của phần tử V[i] của véctơ V.
18. Có phải kiểu véctơ (mảng một chiều) là một cấu trúc dữ liệu có kích thước không cố định?
19. Có phải kiểu véctơ (mảng một chiều) là một cấu trúc dữ liệu đồng nhất?
20. Có phải kiểu véctơ (mảng một chiều) là một cấu trúc dữ liệu không đồng nhất?
21. Để lưu trữ một véctơ trong bộ nhớ, người ta thường sử dụng biểu diễn tuần tự hay biểu diễn liên kết?
22. Cho biết công thức xác định số phần tử của một ma trận M[LB1..UB1, LB2..UB2] (mảng hai chiều).
23. Cho biết công thức xác định địa chỉ (vị trí) của phần tử M[i,j] của ma trận M[LB1..UB1, LB2..UB2]. Biết rằng các phần tử được lưu trữ theo trật tự dòng.
24. Cho biết công thức xác định địa chỉ (vị trí) của phần tử M[i,j] của ma trận M[LB1..UB1, LB2..UB2]. Biết rằng các phần tử được lưu trữ theo trật tự cột.
25. Giả sử có khai báo VAR A:array[0..3, 1..3] of integer; Các phần tử của ma trận A được lưu trữ trong bộ nhớ theo phương pháp khai triển theo cột (trật tự cột). Hãy vẽ mô hình biểu diễn sự lưu trữ này.
26. Giả sử có khai báo VAR A:array[0..3, 1..3] of integer; Các phần tử của ma trận A được lưu trữ trong bộ nhớ theo phương pháp khai triển theo dòngt (trật tự dòng). Hãy vẽ mô hình biểu diễn sự lưu trữ này.

27. Giả sử có khai báo VAR A:array[0..3, 1..3] of integer; Các phần tử của ma trận A được lưu trữ trong bộ nhớ theo phương pháp khai triển theo dòng (trật tự dòng), giả sử địa chỉ cơ sở của khối ô nhớ là ∝, kích thước bộ mô tả là D, kích thước mỗi phần tử là E. Hãy tính địa chỉ (vị trí) của phân tử A[1,2].
28. Giả sử có khai báo VAR A:array[0..3, 1..3] of integer; Các phần tử của ma trận A được lưu trữ trong bộ nhớ theo phương pháp khai triển theo cột (trật tự cột), giả sử địa chỉ cơ sở của khối ô nhớ là ∝, kích thước bộ mô tả là D, kích thước mỗi phần tử là E. Hãy tính địa chỉ (vị trí) của phân tử A[1,2].
29. Nêu tên các thuộc tính của kiểu mẩu tin.
30. Có phải mẩu tin là một cấu trúc dữ liệu có kích thước cố định?
31. Có phải mẩu tin là một cấu trúc dữ liệu có kích thước không cố định?
32. Có phải mẩu tin là một cấu trúc dữ liệu đồng nhất?
33. Có phải mẩu tin là một cấu trúc dữ liệu không đồng nhất?
34. Để lưu trữ một mẩu tin trong bộ nhớ, người ta thường sử dụng biểu diễn tuần tự hay biểu diễn liên kết?
35. Việc lựa chọn một phần tử của mẩu tin được thực hiện bởi sự lựa chọn tuần tự hay trực tiếp?
36. Có phải mẩu tin có cấu trúc thay đổi là một cấu trúc dữ liệu có kích thước cố định?
37. Có phải mẩu tin có cấu trúc thay đổi là một cấu trúc dữ liệu có kích thước thay đổi?
38. Nêu tên các phương pháp đặc tả chuỗi ký tự.
39. Nêu tên các phép toán thường có trên kiểu chuỗi ký tự.
40. Cấp phát tĩnh được thực hiện vào lúc nào?
41. Cấp phát động được thực hiện vào lúc nào?
42. Cho biết các ưu nhược điểm của cấp phát động.
43. Sử dụng cấp phát tĩnh, người lập trình có thể chủ động giải phóng ô nhớ không?
44. Sử dụng cấp phát động, người lập trình có thể chủ động giải phóng ô nhớ không?
45. Biến con trỏ được cấp phát động hay cấp phát tĩnh?
46. Có những loại con trỏ nào?
47. Nêu tên các phép toán thường có trên tập hợp.
48. Nêu tên các phương pháp để biểu diễn một tập hợp.
49. Giả sử một tập hợp được biểu diễn bởi một vectơ bit, hãy cho biết giải thuật để thực hiện các phép toán Hợp, Giao và Hiệu hai tập hợp.
50. Sử dụng véctơ bit để biểu diễn cho một tập hợp thì có những ưu, nhược điểm gì?

51. Sử dụng bảng băm để biểu diễn cho một tập hợp thì có những ưu, nhược điểm gì?
52. Giả sử một không gian có 5 phần tử e1, e2, e3, e4, e5> Tập hợp { e2, e1, e5, e4} được biểu diễn bởi vector bit nào?
53. Giả sử có ba tập hợp A, B, C được biiểu diễn bởi ba vector bit tương ứng là (1, 0, 1, 1, 1); (1, 0, 1, 0, 1) và (1, 1, 1, 0, 1). Cho biết biểu thức liên hệ giữa các tập A,B và C?
54. Kể tên các phép toán thường có trên tập tin tuần tự.
55. Trong tập tin tuần tự, chúng ta có thể nhảy đến một phần tử bất kỳ để truy xuất nó hay không?
56. Trong tập tin truy xuất trực tiếp, chúng ta có thể nhảy đến một phần tử bất kỳ để truy xuất nó hay không?
57. Trong tập tin truy xuất trực tiếp, chúng ta có thể truy xuất các phần tử một cách tuần tự từ đầu đến cuối tập tin hay không?

CHƯƠNG 5: KIỂU DO NGƯỜI DÙNG ĐỊNH NGHĨA

5.1 TỔNG QUAN

5.1.1 Mục tiêu

Sau khi học xong chương này, sinh viên cần phải nắm:

- Khái niệm về trừu tượng hóa, kiểu dữ liệu do người lập trình định nghĩa.
- Mục đích của việc định nghĩa kiểu dữ liệu.
- Phân biệt các hình thức xác định sự tương đương giữa các kiểu dữ liệu

5.1.2 Nội dung cốt lõi

- Trừu tượng hoá
- Kiểu dữ liệu do người dùng định nghĩa.
- Xác định sự tương đương giữa các kiểu dữ liệu

5.1.3 Kiến thức cơ bản cần thiết

Kiến thức và kĩ năng lập trình căn bản

5.2 SỰ PHÁT TRIỂN CỦA KHÁI NIỆM KIỂU DỮ LIỆU

Sự phát triển của khái niệm kiểu dữ liệu là sự phát triển chủ yếu của ngôn ngữ lập trình trong những năm 70. Trong những ngôn ngữ cũ như FORTRAN và COBOL đã bắt đầu có khái niệm về kiểu.

Ý niệm đầu tiên về sự định nghĩa kiểu là một tập hợp các giá trị mà một biến có thể nhận. Kiểu dữ liệu trong các ngôn ngữ cũ này luôn luôn gắn liền với các biến riêng lẻ, do đó mỗi một phép khai báo biến phải đặt tên cho một biến và định nghĩa kiểu của nó. Do đó nếu một chương trình sử dụng nhiều biến có kiểu giống nhau thì mỗi một biến phải được khai báo riêng.

Bước tiếp theo của sự phát triển khái niệm kiểu được nghiên cứu trong Pascal. Trong đó cho phép đặt tên cho một kiểu, tức là một tập giá trị nào đó. Phép khai báo biến chỉ cần tên biến và tên kiểu đã định nghĩa chứ không cần định nghĩa lại kiểu.

Bước cuối cùng của sự phát triển khái niệm kiểu là: Kiểu không chỉ là một tập hợp các đối tượng dữ liệu mà còn là một tập hợp các phép toán có thể thao tác trên các đối tượng dữ liệu này.

5.3 TRỪU TƯỢNG HÓA

5.3.1 Khái niệm trừu tượng hóa

Trừu tượng hóa là một phương pháp giúp người lập trình biết cách tập trung vào những vấn đề, những thuộc tính bản chất của chương trình mà bỏ qua các thuộc tính không cần thiết. Nó là một vũ khí chống lại độ phức tạp của chương trình, mục đích của nó là đơn giản hóa quá trình lập trình.

Có hai loại trừu tượng hóa cơ bản trong ngôn ngữ lập trình là trừu tượng hóa quá trình và trừu tượng hóa dữ liệu.

5.3.2 Trừu tượng hóa quá trình

Trừu tượng hóa quá trình là việc phân chia chương trình thành những chương trình con. Mỗi chương trình con đảm nhiệm một tác vụ nào đó và được đặc trưng bởi một cái tên.

Ở cấp độ chương trình chính chúng ta chỉ gọi thực hiện các chương trình con, thông qua các tên chương trình con, để thực hiện các tác vụ mà chương trình con đó đảm trách. Như vậy, ở chương trình chính, chúng ta chỉ quan tâm đến kết quả của chương trình con mang lại mà không cần biết chi tiết cài đặt bên trong chương trình con đó.

Ví dụ để viết một chương trình quản lý, ta có thể viết theo hai cách, cách thứ nhất không phân chia thành các chương trình con và cách thứ hai có sử dụng chương trình con.

```
Program Quan_ly;
Begin
   {
   ----
   Đoạn chương trình dùng cho việc
   nhập dữ liệu
   ----
   }
   {
   ----
   Đoạn chương trình dùng cho việc xử
   lý dữ liệu
   ----
   }
   {
   ----
   Đoạn chương trình dùng cho việc
   xuất dữ liệu
   ----
   }
end.
```

Chương trình không có chương trình con

```
Program Quan_ly;
Prcedure nhap_du_lieu;
   {
   Chương trình con nhập dữ liệu
   }
Prcedure xu_ly_du_lieu;
   {
   Chương trình con xử lý dữ liệu
   }
Prcedure xu_ly_du_lieu;
   {
   Chương trình con xuất dữ liệu
   }

Begin  {Chương trình chính}
      nhap_du_lieu;
      xu_ly_du_lieu;
      xuat_du_lieu;
end.
```

Chương trình với chương trình con

Đối với phương pháp thứ nhất, ta thấy toàn bộ chương trình được viết trong chương trình chính, điều này làm cho chương trình chính rất rườm rà, khó đọc hiểu, khó kiểm soát, khó sửa lỗi,...

Đối với phương pháp thứ hai, trong chương trình chính ta chỉ thấy tên các chương trình con (nhap_du_lieu, xu_ly_du_lieu, xuat_du_lieu) và thông qua các tên này ta biết rõ chương trình chính làm những việc gì còn bản thân các việc ấy được làm như thế nào thì ta không cần biết.

Ưu điểm của trừu tượng hoá quá trình

Việc phân chia chương trình thành các chương trình con có các ưu điểm nổi bật như sau:

- Ở chương trình chính, cái tổng thể được làm nổi bật, các chi tiết bị che dấu nên chương trình sáng sủa, dễ đọc hiểu.

- Một chương trình con đã được thiết kế thì có thể gọi thực hiện nhiều lần mà không phải viết lại. Với việc truyền tham số cho chương trình con, ta nhận được các kết quả khác nhau ở các lần gọi khác nhau.

- Khi xây dựng chương trình con ta có thể kiểm thử nó một cách độc lập, nên việc phát hiện và sửa lỗi dễ dàng hơn.

- Do chương trình được chia thành nhiều chương trình con, mỗi chương trình con có thể giao cho một hoặc một nhóm lập trình viên thực hiện nên tăng khả năng làm việc theo nhóm.

5.3.3 Trừu tượng hóa dữ liệu

Trừu tượng hoá dữ liệu là việc tạo ra kiểu dữ liệu trừu tượng. Kiểu dữ liệu trừu tượng là một tập hợp các ĐTDL và tập hợp các phép toán, thao tác trên các ĐTDL đó.

Ngày nay, khi ta nói kiểu dữ liệu thực chất là kiểu dữ liệu trừu tượng.

Kiểu dữ liệu trừu tượng có thể được định nghĩa bởi ngôn ngữ hoặc do người lập trình định nghĩa.

Ví dụ về kiểu dữ liệu trừu tượng do ngôn ngữ định nghĩa:

Kiểu integer trong Pascal hay kiểu int trong C là một kiểu dữ liệu trừu tượng do ngôn ngữ định nghĩa. Trong đó tập các ĐTDL là tập các số nguyên từ -32768 đến 32767; tập hợp các phép toán bao gồm các phép toán một ngôi (+, -), các phép toán hai ngôi (+, -, *, DIV, MOD), các phép toán quan hệ (<, <=, =, <>, >=, >).

Ví dụ về kiểu dữ liệu trừu tượng do người lập trình định nghĩa:

Trong môn học cấu trúc dữ liệu, chúng ta đã biết một loạt các kiểu dữ liệu trừu tượng do người lập trình định nghĩa như danh sách, ngăn xếp, hàng đợi, cây,...

Chẳng hạn kiểu dữ liệu trừu tượng danh sách là một dãy các phần tử với tập hợp các phép toán như tạo danh sách rỗng, kiểm tra danh sách rỗng, xen một phần tử vào danh sách, xoá một phần tử khỏi danh sách, ...

Sau đây ta sẽ nghiên cứu kỹ hơn về kiểu dữ liệu trừu tượng do người dùng định nghĩa.

5.4 ĐỊNH NGHĨA KIỂU

5.4.1 Khái niệm

Ngoài các kiểu nguyên thuỷ được định nghĩa bởi ngôn ngữ, người lập trình còn có thể định nghĩa các kiểu của riêng mình. Định nghĩa một kiểu dữ liệu mới bao gồm việc xác định các yếu tố sau:

- Tên của kiểu.
- Sự biểu diễn bộ nhớ cho các đối tượng dữ liệu của kiểu.
- Tập hợp các phép toán (các chương trình con) thao tác trên các đối tượng dữ liệu của kiểu.

Ví dụ trong Pascal ta xét định nghĩa kiểu như sau:

TYPE

```
    RealVect = ARRAY[1..10] OF real;
```

Sau đó ta có thể dùng phép khai báo biến:

```
VAR
    A: RealVect;
    B,C:RealVect;
```

Ưu điểm của định nghĩa kiểu:

- Làm cho việc viết chương trinh trở nên ngắn gọn, sáng sủa hơn.
- Khi cần thay đổi cấu trúc dữ liệu, chỉ cần thay đổi một lần ở mức định nghĩa kiểu chứ không cần phải thay đổi nhiều lần ở mức khai báo từng biến riêng biệt.

Chúng ta thấy rằng kiểu do người dùng định nghĩa chính là một kiểu dữ liệu trừu tượng.

5.4.2 Tính tương đương của các kiểu định nghĩa

Kiểm tra kiểu dẫn tới sự so sánh giữa kiểu dữ liệu của đối số thực đã được cho của một phép toán và kiểu dữ liệu của đối số mà phép toán đó cần đến. Nếu kiểu giống nhau thì đối số được chấp nhận và phép toán được tiến hành, nếu kiểu khác nhau, thì một lỗi được xem xét hoặc một sự cưỡng bức chuyển đổi kiểu được dùng để đổi kiểu của đối số thực thành kiểu thích hợp.

Vấn đề ở đây là cần phải xác định hai kiểu như thế nào thì được coi là "giống nhau" hay tương đương. Xét ví dụ sau đây:

```
TYPE        Vect1 = ARRAY[1..10] OF REAL;
            Vect2 = ARRAY[1..10] OF REAL;
VAR   x,z : Vect1;
      y : Vect2;
PROCEDURE Sub(a:Vect1);
    .....
END;   { Sub }
BEGIN   { Chương trình chính }
    x := y;
    Sub(y);
    ......
END.
```

Vấn đề ở đây là các biến x, y và a có cùng kiểu do đó lệnh gán x := y và lời gọi chương trình con Sub(y) là đúng hay chúng có khác kiểu.

Có hai cách giải quyết cho vấn đề này: tương đương tên và tương đương cấu trúc.

1/ Tương đương tên

H ai kiểu dữ liệu được xem là tương đương chỉ khi chúng có tên giống nhau. Như vậy

các kiểu Vect1 và Vect2 ở trên là khác kiểu mặc dù đối tượng dữ liệu có chung một cấu trúc. Lệnh gán x := y và lời gọi chuong trình con Sub(y) là không hợp lệ. Tương đương tên là phương pháp được dùng trong Ada và Pascal. Tương đương tên có một điểm yếu là khi một kiểu không có tên như trong khai báo trực tiếp:

VAR w : ARRAY[1..10] OF REAL;

Biến w có kiểu riêng nhưng là kiểu không có tên. Như vậy w không thể được dùng như là một đối số cho một phép toán mà phép toán đó đòi hỏi một đối số của một kiểu có tên.

2/ Tương đương cấu trúc

Hai kiểu dữ liệu được xem là tương đương nếu chúng xác định các đối tượng dữ liệu có cấu trúc bên trong giống nhau. Thông thường thuật ngữ "cấu trúc bên trong giống nhau" có nghĩa là giống nhau về sự biểu diễn bộ nhớ được dùng cho cả hai lớp đối tượng dữ liệu. Ví dụ Vect1 và Vect2 là tương đương cấu trúc bởi vì mỗi một đối tượng dữ liệu của kiểu Vect1 và mỗi một đối tượng dữ liệu của kiểu Vect2 có chung số phần tử có kiểu tương đương.

Quản lý bộ nhớ đối với các đối tượng dữ liệu của cả hai kiểu này là giống nhau, do đó công thức truy nhập giống nhau có thể được sử dụng để lựa chọn các phần tử và nói chung sự cài đặt tại thời gian thực hiện của các kiểu dữ liệu là giống hệt nhau.

Tương đương cấu trúc không có các bất tiện như tương đương tên nhưng nó lại có những vấn đề khác, chẳng hạn như hai biến có thể tương đương cấu trúc một cách không cố ý mặc dù người lập trình đã khai báo chúng một cách tách biệt như trong ví dụ sau:

```
TYPE Meters = INTEGER;
     Liters = INTEGER;
VAR  Len : Meters;
     Vol : Liters;
```

Các biến Len và Vol có kiểu tương đương cấu trúc và do đó một lỗi như phép cộng Len + Vol sẽ không được tìm thấy bởi phép kiểm tra kiểu tĩnh. Khi có nhiều lập trình viên làm việc chung trong một chương trình thì tương đương kiểu không cố ý có thể gây nên các lỗi rất nghiêm trọng như trong ví dụ nói trên.

5.5 CÂU HỎI ÔN TẬP

1. Thế nào là trừu tượng hoá quá trình?
2. Thế nào là trừu tượng hoá dữ liệu?
3. Lập trình theo kiểu trừu tượng hoá quá trình có những ưu điểm nào?
4. Định nghĩa kiểu dữ liệu có những ưu điểm nào?
5. Có những phương pháp nào để xác định sự tương đương của các kiểu dữ liệu.

CHƯƠNG 6: CHƯƠNG TRÌNH CON

6.1 TỔNG QUAN

6.1.1 Mục tiêu

Sau khi học xong chương này, sinh viên cần phải nắm:

- Khái niệm về chương trình con.
- Cơ chế hoạt động khi chương trình con được gọi.
- Các cách truyền tham số cho chương trình con.

6.1.2 Nội dung cốt lõi

- Cơ chế hoạt động của chương trình con.
- Các loại tham số của chương trình con.
- Các cách truyền tham số cho chương trình con.

6.1.3 Kiến thức cơ bản cần thiết

Kiến thức và kĩ năng lập trình căn bản

6.2 ĐỊNH NGHĨA CHƯƠNG TRÌNH CON

Chương trình con là một phép toán trừu tượng được định nghĩa bởi người lập trình. Chương trình con có các đặc tính như sau:

- Mỗi chương trình con có một điểm vào duy nhất.

Chương trình gọi tạm ngừng việc thực hiện trong quá trình thực hiện chương trình con.

- Điều khiển luôn được trả về chương trình gọi khi sự thực hiện chương trình con kết thúc.

Khi nói đến chương trình con, chúng ta quan tâm đến hai khía cạnh: sự định nghĩa chương trình con và lời gọi thực hiện chương trình con.

Định nghĩa chương trình con là một đoạn chương trình nguồn được viết ra bởi người lập trình.

6.2.1 Đặc tả chương trình con

Sự đặc tả chương trình con bao gồm:

- Tên của chương trình con
- Số lượng các tham số, thứ tự của chúng và kiểu dữ liệu của mỗi một tham số .
- Số lượng các kết quả, thứ tự của chúng và kiểu dữ liệu của mỗi một kết quả.
- Hoạt động được thực hiện bởi chương trình con.

Chương trình con biểu diễn một hàm toán học, là một ánh xạ từ tập hợp các tham số đến tập hợp các kết quả. Chương trình con trả về một kết quả duy nhất trong tên chương trình con thường được gọi là một hàm. Cú pháp điển hình đặc tả hàm được quy định trong ngôn ngữ lập trình Pascal:

Function Ten_ham(Danh sách các tham số cùng với kiểu dữ liệu tương ứng): Kiểu kết quả trả về

Ví dụ Function FN(x:real; y:integer) : real

Đặc tả này xác định hàm FN : REAL x INTEGER -> REAL

Nếu chương trình con trả về nhiều hơn một kết quả hoặc không có kết quả trả về trong tên chương trình con thường được gọi là thủ tục (procedure hoặc subroutine). Cú pháp điển hình đặc tả thủ tục được quy định trong ngôn ngữ lập trình Pascal:

Procedure Ten_thu_tuc(Danh sách các tham sốcùng với kiểu dữ liệu tương ứng)

Ví dụ Procedure SUB(X:real; Y:Integer; Var Z:Real; Var U:boolean);

Trong sự đặc tả này, tham số có tên đứng sau VAR biểu thị một kết quả hoặc một tham số có thể bị thay đổi. Cú pháp của sự đặc tả này trong Ada là:

Procedure SUB(X: IN Real; Y: IN Integer; Z: IN OUT Real; U: OUT Boolean)

Thủ tục này khai báo một chương trình con với sự xác định:

SUB : Real x Integer x Real -> Real x Boolean

Các từ IN, OUT và IN OUT phân biệt ba trường hợp sau đây: IN chỉ định một tham số không thể bị thay đổi bởi cương trình con, IN OUT chỉ định một tham số có thể bị thay đổi và OUT chỉ định một kết quả.

Mặc dù chương trình con biểu diễn một hàm toán học nhưng nó cũng có các vấn đề tương tự như đối với các phép toán nguyên thuỷ:

- Chương trình con có thể có các tham số ẩn trong dạng biến không địa phương mà nó tham chiếu.

- Chương trình con có thể có kết quả ẩn (hiệu ứng lề) được trả về thông qua sự thay đổi các biến không địa phương hoặc thông qua việc thay đổi các tham số IN-OUT của nó.

- Chương trình con có thể nhạy cảm với tiền sử (tự sửa đổi), vì vậy kết qủa của nó không chỉ phụ thuộc vào tham số được cho tại lần gọi đó mà còn phụ thuộc vào toàn bộ lịch sử các lần gọi trước đó. Nhạy cảm với tiền sử có thể do dữ liệu địa phương vẫn còn giữ lại giữa các lần gọi của chương trình con hoặc thông qua sự thay đổi mã riêng của nó (ít phổ biến hơn).

6.2.2 Cài đặt chương trình con

Các phép toán nguyên thuỷ được cài đặt bằng cách dùng cấu trúc dữ liệu và các phép toán được cung cấp bởi máy tính ảo bên dưới ngôn ngữ lập trình. Chương trình con biểu diễn một phép toán được xây dựng bởi người lập trình và do đó chương trình con được cài đặt bằng cách dùng cấu trúc dữ liệu và các phép toán được cung cấp bởi chính bản thân ngôn ngữ lập trình đó. Sự cài đặt được xác định bởi thân chương trình con, bao gồm cả việc khai báo dữ liệu cục bộ xác định cấu trúc dữ liệu được dùng cho chương trình con và các lệnh xác định hành động sẽ làm khi chương trình con thực hiện.

Sự khai báo và các lệnh thường được bao gói, người sử dụng chương trình con không thể truy xuất được tới dữ liệu cục bộ và các lệnh bên trong chương trình con. Người sử

dụng chỉ có thể gọi chương trình con với một tập hợp các tham số và nhận lại các kết quả đã được tính toán. Cú pháp của Pascal đối với chương trình con là điển hình:

```
FUNCTION FN(x:REAL; y:INTEGER):REAL ;        (Mô tả)
VAR
   m : ARRAY[1..10] OF REAL;   (Khai báo các đối
   n : INTEGER;                 tượng dữ liệu cục bộ)
BEGIN
  .                            (Dãy các lệnh xác định
  .                             hành động của chương
  .                             trình con)
END;
```

Thông thường trong chương trình con có thể có các chương trình con khác biểu thị các phép toán được định nghĩa bởi người lập trình dùng cho chỉ chương trình con chứa chúng. Những chương trình con "cục bộ" này được bao gói, nghĩa là chúng không thể được gọi tới từ bên ngoài chương trình con chứa chúng.

Kiểm tra kiểu cũng là một vấn đề quan trọng đối với chương trình con. Mỗi lần gọi chương trình con đòi hỏi các tham số có kiểu đúng như đã được xác định trong sự đặc tả chương trình con. Kiểu của kết quả được trả về của chương trình con cũng phải được biết đến. Vấn đề kiểm tra kiểu tương tự như đối với các phép toán nguyên thuỷ. Kiểm tra kiểu có thể được thực hiện một cách tĩnh trong quá trình dịch, nếu đã có sự khai báo kiểu cho các tham số và kết quả của mỗi một chương trình con. Mặt khác kiểm tra kiểu có thể là động trong quá trình thực hiện chương trình. Sự chuyển đổi kiểu ẩn các tham số để đổi chúng thành các kiểu đúng cũng có thể được cung cấp một cách tự động bởi sự cài đặt ngôn ngữ.

6.3 CƠ CHẾ GỌI CHƯƠNG TRÌNH CON

6.3.1 Định nghĩa và kích hoạt chương trình con

Người lập trình viết một định nghĩa chương trình con trong chương trình. Thông qua việc thực hiện chương trình, nếu chương trình con được gọi thì một kích hoạt chương trình con (subprogram activation) được tạo ra. Khi sự thực hiện chương trình con kết thúc thì kích hoạt bị phá huỷ. Nếu có một lần gọi khác thì một kích hoạt mới sẽ được tạo ra. Như vậy từ một định nghĩa chương trình con, có nhiều kích hoạt có thể được tạo trong quá trình thực hiện chương trình. Sự định nghĩa dùng làm khuôn mẫu (template) cho việc tạo kích hoạt trong quá trình thực hiện.

Chú ý rằng sự định nghĩa là một sự diễn tả trong chương trình như nó đã được viết ra và do đó nó được biết đến trong quá trình dịch. Kích hoạt chương trình con chỉ có thể có trong quá trình thực hiện. Trong quá trình thực hiện, sự định nghĩa chỉ tồn tại trong dạng khuôn mẫu mà từ đó các kích hoạt có thể được tạo ra.

6.3.2 Cài đặt định nghĩa và kích hoạt chương trình con

Xét lại định nghĩa chương trình con trong Pascal:

```
FUNCTION FN(x : REAL; y : INTEGER) : REAL;
CONST
     max = 20;
```

```
VAR
      m : ARRAY[1..max] OF REAL;
      n : INTEGER;
BEGIN
     n := MAX;
     x := 2 * x + m[5];
    ......
END;
```

Sự định nghĩa này xác định các thành phần cần thiết cho một kích hoạt chương trình con:

1/ Bộ nhớ đối với các tham số, đối tượng dữ liệu x và y.

2/ Bộ nhớ cho kết quả hàm, đối tượng dữ liệu của kiểu REAL;

3/ Bộ nhớ cho biến cục bộ, mảng m và biến n.

4/ Bộ nhớ cho các hằng trực kiện và các hằng, 20, 2 và 5.

5/ Bộ nhớ cho mã có thể thực hiện phát sinh từ các lệnh trong thân chương trình con.

Định nghĩa chương trình con cho phép các vùng nhớ này được tổ chức và các mã có thể thực hiện được xác định thông qua việc dịch. Kết quả của việc dịch là khuôn mẫu dùng để xây dựng mỗi một kích hoạt riêng tại thời gian chương trình con được gọi trong quá trình thực hiện.

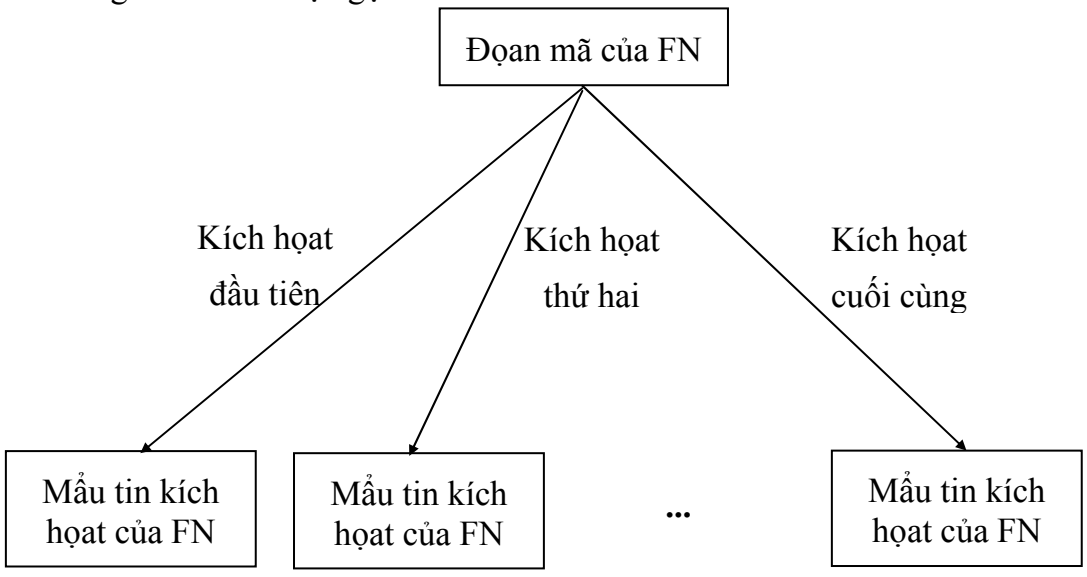
Để xây dựng một kích hoạt chương trình con từ khuôn mẫu của nó, cái khuôn mẫu ban đầu có thể phải được sao chép vào trong vùng mới của bộ nhớ. Tuy nhiên thay vì sao chép hoàn toàn, nó được tách ra thành hai phần:

1/ Phần tĩnh, gọi là đoạn mã (code segment) bao gồm các mục 4 (các hằng) và 5 (các mã có thể thực hiện được) đã nói ở trên. Phần tĩnh không thay đổi trong quá trình thực hiện chương trình con và do thế một bản sao có thể được dùng cho tất cả các kích hoạt.

2/ Phần động, gọi là mẩu tin kích hoạt (Activation record) bao gồm các mục 1 (Các tham số), 2 (Kết quả hàm) và 3 (dữ liệu cục bộ) và cộng thêm nhiều mục khác của dữ liệu ẩn như vùng nhớ tạm, điểm trở về, và sự tham khảo các biến không cục bộ. Phần động có cấu trúc giống nhau cho tất cả các kích hoạt nhưng nó chứa các giá trị dữ liệu khác nhau cho mỗi một kích họat. Do đó mỗi một kích hoạt cần thiết phải có một bản sao mẩu tin kích hoạt riêng của nó. Hình vẽ sau trình bày cấu trúc của một kích hoạt chương trình con của hàm FN nói trên

	Mở đầu	
	Mã lệnh có thể thực hiện	Khối mã có thể thực hiện
	Kết thúc	
Đoạn mã của FN	20 2 5 1	Các hằng
		Điểm trở về
	FN	Kết quả của hàm
	x	Các tham số
Mẩu tin kích hoạt của FN	y	
	m : : : n	Các biến cục bộ

Đối với mỗi chương trình con, một đoạn mã tồn tại thông qua sự thực hiện chương trình. Các mẩu tin kích hoạt được tạo ra và huỷ bỏ một cách động thông qua thực hiện mỗi lần chương trình con được gọi và mỗi lần nó kết thúc bởi RETURN.



Kích thước và cấu trúc của mẩu tin kích hoạt của chương trình con thông thường được xác định trong quá trình dịch, đó là trình biên dịch (compile) có thể xác định mẩu tin kích hoạt lớn như thế nào và vị trí của mỗi một phần tử trong đó. Để truy xuất đến các phần tử có thể sử dụng công thức tính địa chỉ cơ sở cộng độ dời như đã trình bày đối với mẩu tin bình thường.

6.4 CHƯƠNG TRÌNH CON CHUNG

Sự đặc tả chương trình con thông thường liệt kê số lượng, thứ tự và kiểu dữ liệu của các tham số. Chương trình con chung (generic subprogram) là một chương trình con có một tên nhưng có nhiều định nghĩa khác nhau, được phân biệt bởi số lượng, thứ tự và kiểu dữ liệu của các tham số. Khái niệm về phép toán chung đã được đề cập trong chương II (phần mục đích của sự khai báo). Một khi phép toán hay chương trình con chung xuất hiện, vấn đề cơ bản đối với trình biên dịch là làm sao xác định được đúng ý nghĩa của nó trong một tập hợp nhiều ý nghĩa có thể có. Thông tin về đối số của phép toán hoặc tham số của chương trình con giúp chương trình dịch nhận biết ý nghĩa đích thực của chương trình con chung.

6.5 TRUYỀN THAM SỐ CHO CHƯƠNG TRÌNH CON

6.5.1 Khái niệm truyền tham số

Truyền tham số là phương pháp chuyển giao dữ liệu giữa các đơn vị chương trình nhằm đạt được các kết quả khác nhau khi gọi thực hiện chương trình con. Tham số có hai loại là tham số hình thức và tham số thực tế.

Tham số hình thức là một loại đặc biệt của ĐTDL cục bộ trong chương trình con. Nó được xác định lúc định nghĩa chương trình con. Khi định nghĩa chương trình con, phải xác định một danh sách các tham số hình thức cùng với khai báo kiểu tương ứng.

Tham số thực tế là một ĐTDL được chia sẻ cho chương trình con bằng cách truyền trong lời gọi thực hiện chương trình con.

6.5.2 Sự tương ứng giữa tham số tham số thực tế và tham số hình thức

Khi một chương trình con được gọi cùng với một danh sách các tham số thực tế thì đòi hỏi phải có một sự tương ứng giữa các tham số hình thức và tham số thực tế. Sự tương ứng này nhằm để xác định tham số thực tế nào được truyền cho tham số hình thức nào. Có hai phương pháp để xác định sự tương ứng này:

Tương ứng vị trí: Sự tương ứng giữa các cặp tham số thực tế và tham số hình thức đặt cơ sở trên sự tương ứng vị trí của chúng trong danh sách các tham số thực tế lúc gọi và danh sách các tham số hình thức trong định nghĩa chương trình con.

Hầu hết các ngôn ngữ đều dùng tương ứng vị trí và số lượng các tham số hình thức và tham số thực tế là bằng nhau do đó các cặp tương ứng là duy nhất. Tuy nhiên cũng có những ngôn ngữ cho phép lời gọi chương trình con không cần cung cấp đủ số lượng tham số thực tế.

Tương ứng tên: Tham số hình thức bắt cặp với tham số thực tế được chỉ định trong lời gọi chương trình con. Ví dụ trong Ada có lời gọi chương trình con như sau:

SUB(y => B, x => 27).

Trong đó tham số thực tế B bắt cặp với tham số hình thức y và tham số thực tế 27 bắt cặp với tham số hình thức x.

6.5.3 Các phương pháp truyền tham số tham số

Nói chung một ngôn ngữ cung cấp nhiều phương pháp truyền tham số mà người lập trình có thể lựa chọn để xác định khai báo tham số hình thức lúc định nghĩa chương trình con và cung cấp các tham số thực tế lúc gọi thực hiện chương trình con. Các phương pháp truyền tham số chủ yếu bao gồm:

Truyền bằng giá trị (transmission by value)

- Tham số hình thức là tham số chỉ vào (IN-only parameters), tức là chỉ nhận giá trị vào cho chương trình con, không có nghĩa vụ trả kết quả về cho chương trình gọi. Tham số hình thức được xem như là một **biến cục bộ** của chương trình con và được cấp phát ô nhớ riêng.

- Tham số thực tế là một **biểu thức** (là một biến, một hằng, một hàm hoặc là một biểu thức thực sự).

- Phương pháp thực hiện: Tại thời điểm gọi, **giá trị** của tham số thực tế được sao chép vào trong ô nhớ của tham số hình thức. Trong quá trình thực hiện chương trình con, mọi thao tác trên tham số hình thức là sự thao tác trên ô nhớ riêng của nó, không ảnh hưởng đến tham số thực tế.

- Khi chương trình con kết thúc, sự thay đổi giá trị của tham số hình thức, **không làm ảnh hưởng** đến giá trị của tham số thực tế.

Truyền tham chiếu (transmission by reference)

- Tham số hình thức là tham số vào ra (IN-OUT parameters), tức là nó có nghĩa vụ nhận giá trị vào cho chương trình con và trả kết quả về cho chương trình gọi. Tham số hình thức là một **con trỏ**.

- Tham số thực tế phải là **một biến**, tức là một ĐTDL có ô nhớ.

- Phương pháp thực hiện: Tại thời điểm gọi, con trỏ của tham số thực tế được sao chép cho tham số hình thức. Trong quá trình thực hiện chương trình con, mọi thao tác trên tham số hình thức là sự thao tác trên **ô nhớ của tham số thực tế**.

- Khi chương trình con kết thúc, mọi thay đổi giá trị của tham số hình thức đều làm giá trị của tham số thực tế **thay đổi** theo.

Truyền bằng giá trị-kết quả (transmission by value-result)

- Tham số hình thức là tham số vào ra (IN-OUT parameters) nhưng là một biến cục bộ của chương trình con và được cấp phát ô nhớ riêng.

- Tham số thực tế phải là **một biến**, tức là một ĐTDL có ô nhớ.

- Phương pháp thực hiện: Tại thời điểm gọi, **giá trị** của tham số thực tế được sao chép vào trong ô nhớ của tham số hình thức. Trong quá trình thực hiện chương trình con, mọi thao tác trên tham số hình thức là sự thao tác trên ô nhớ riêng của nó, không ảnh hưởng đến tham số thực tế.

- Khi chương trình con kết thúc, giá trị cuối cùng của tham số hình thức được sao chép vào ô nhớ của tham số thực tế.

Truyền bằng kết quả (transmission by result)

- Tham số hình thức là tham số chỉ ra (OUT-only parameters), tức là chỉ trả kết quả về cho chương trình gọi, không có nghĩa vụ nhận giá trị vào cho chương trình con. Tham số hình thức được xem như là một **biến cục bộ** của chương trình con và được cấp phát ô nhớ riêng.
- Tham số thực tế phải là **một biến**, tức là một ĐTDL có ô nhớ.
- Phương pháp thực hiện: Giá trị của tham số thực tế **không được sử dụng** trong chương trình con. Tham số hình thức có thể được gán trị như đối với một biến cục bộ. Trong quá trình thực hiện chương trình con, mọi thao tác trên tham số hình thức là sự thao tác trên ô nhớ riêng của nó, không ảnh hưởng đến tham số thực tế.
- Khi chương trình con kết thúc, giá trị cuối cùng của tham số hình thức được sao chép vào ô nhớ của tham số thực tế.

Ví dụ viết bằng ngôn ngữ giả

```
Var m:integer;
Procedure P(a:integer);
Begin
     a:= 20;
     writeln(m);
end;
begin
     m:=10;
     P(m);
     writeln(m);
end.
```

Kết quả thực hiện chương trình đối với các phương pháp truyền tham số

Truyền bằng giá trị	Truyền tham chiếu	Truyền bằng giá trị-kết quả	Truyền bằng kết quả
10	20	10	10
10	20	20	20

6.6 CÂU HỎI ÔN TẬP

1. Nêu tên các phương pháp tương ứng giữa tham số thực tế và tham số hình thức khi thực hiện việc truyền tham số cho chương trình con.
2. Nêu tên các phương pháp truyền tham số cho chương trình con.
3. Cho biết sự khác nhau và giống nhau giữa các phương pháp truyền tham số .

CHƯƠNG 7: ĐIỀU KHIỂN TUẦN TỰ

7.1 TỔNG QUAN

7.1.1 Mục tiêu

Sau khi học xong chương này, sinh viên cần phải nắm:

- Khái niệm về điều khiển tuần tự.
- Các thứ tự thực hiện chương trình trong biểu thức, trong câu lệnh..
- Khái niệm về ngoại lệ, xử lý ngoại lệ.

7.1.2 Nội dung cốt lõi

- Điều khiển tuần tự trong biểu thức.
- Điều khiển tuần tự trong câu lệnh.
- Ngoại lệ và xử lý ngoại lệ.

7.1.3 Kiến thức cơ bản cần thiết

Kiến thức về cấu trúc dữ liệu và kĩ năng lập trình căn bản

7.2 KHÁI NIỆM ĐIỀU KHIỂN TUẦN TỰ

Điều khiển tuần tự là tập hợp quy tắc xác định thứ tự thực hiện trong chương trình.

Xét về mặt cấu trúc thì có ba loại điều khiển:

- Điều khiển trong biểu thức.
- Điều khiển giữa các lệnh.
- Điều khiển giữa các chương trình con.

Xét về mặt thiết kế ngôn ngữ thì có hai loại điều khiển là:

- Điều khiển ẩn được thiết kế trong ngôn ngữ chẳng hạn quy tắc ưu tiên của các toán tử trong biểu thức.
- Điều khiển tường minh do người lập trình viết trong chương trình chẳng hạn sử dụng các câu lệnh điều khiển như rẽ nhánh, lặp lại ...

7.3 ĐIỀU KHIỂN TUẦN TỰ TRONG BIỂU THỨC

7.3.1 Đặt vấn đề

Xét công thức nghiệm của phương trình bậc hai $x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$

Công thức đơn giản này bao gồm ít nhất 15 phép toán khác nhau. Mã hoá trong hợp ngữ hoặc ngôn ngữ máy, có thể đòi hỏi ít nhất 15 lệnh. Hơn thế, người lập trình phải quy định bộ nhớ cho 5 đến 10 kết quả trung gian sẽ phát sinh. Người lập trình cũng sẽ phải quan tâm đến việc tối ưu như các phép toán sẽ được thực hiện theo thứ tự như thế nào để bộ nhớ tạm là nhỏ nhất ...

Trong ngôn ngữ cấp cao như FORTRAN, công thức này được viết như một biểu thức

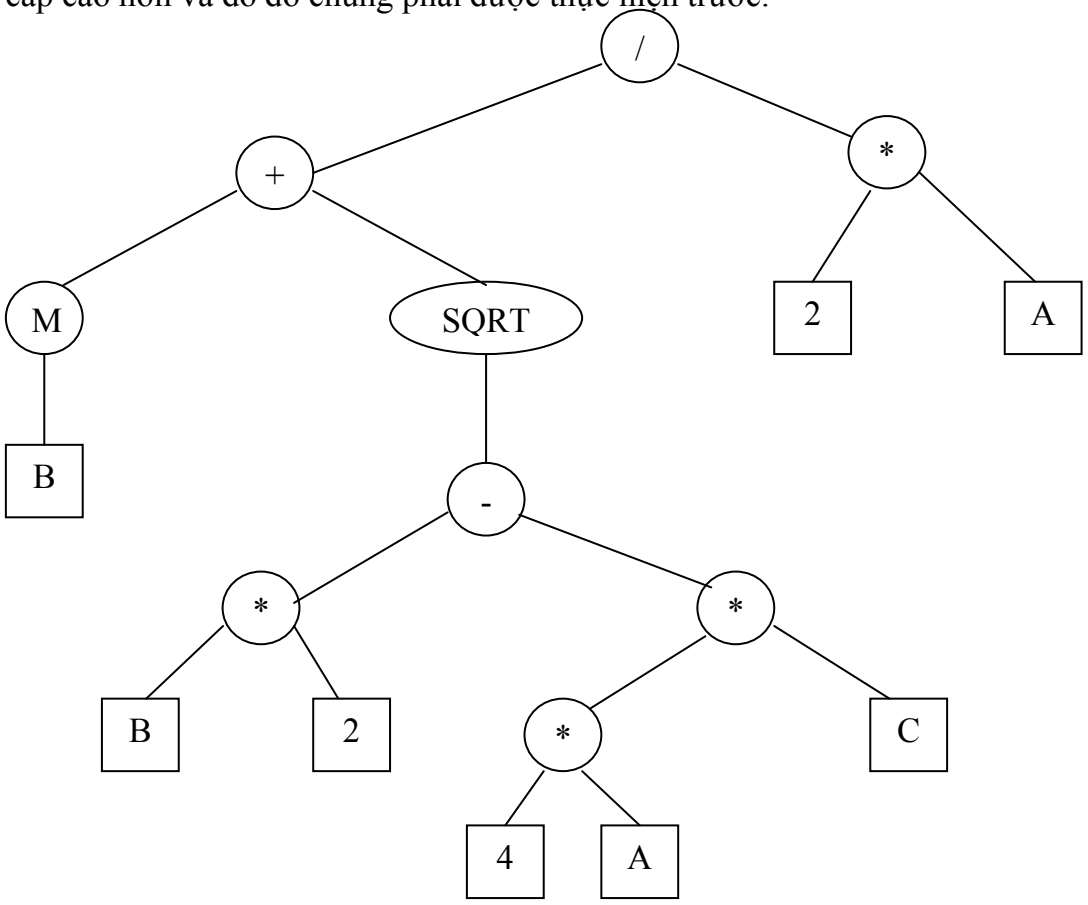
x = (-b + SQRT(b**2 - 4*a*c))/(2*a)

Biểu thức là một phương tiện tự nhiên và mạnh mẽ cho việc biểu diễn dãy các phép toán, tuy vậy chúng nảy sinh các vấn đề mới chẳng hạn như thứ tự thực hiện các toán tử.

7.3.2 Sự biểu diễn theo cấu trúc cây của biểu thức

Cơ chế điều khiển tuần tự cơ bản trong biểu thức là phép lấy hàm hợp: Một phép toán chính và các toán hạng của nó. Trong đó các toán hạng có thể là các hằng, biến hoặc các phép toán khác mà các toán hạng của chúng lại có thể là các hằng, biến hoặc các phép toán khác... Như vậy có thể xem biểu thức là một cấu trúc cây, trong đó nút gốc của cây biểu diễn cho phép toán chính, các nút giữa gốc và lá biểu diễn cho các phép toán trung gian và các nút lá biểu diễn các biến và các hằng. Ví dụ biểu thức nghiệm phương trình bậc hai được biểu diễn theo cấu trúc cây như sau (dùng M để biểu diễn cho phép toán một ngôi lấy số đối):

Sự biểu diễn cây làm sáng sủa cấu trúc điều khiển của biểu thức. Rõ ràng là các kết quả của biến hoặc phép toán ở cấp thấp trong cây được coi như là toán hạng của phép toán ở cấp cao hơn và do đó chúng phải được thực hiện trước.



7.3.3 Cú pháp của biểu thức

Nếu chúng ta xem biểu thức được biểu diễn bởi cây thì để dùng biểu thức trong chương trình, cây phải được tuyến tính hóa chẳng hạn phải có quy định để viết cây như là một dãy tuyến tính các ký hiệu. Chúng ta hãy xem các ký hiệu phổ biến nhất:

Ký hiệu tiền tố (prefix)

Theo ký hiệu Prefix, phép toán viết trước, sau đó là các toán hạng theo thứ tự từ trái sang phải. Nếu một toán hạng lại là một phép toán thì cũng theo quy tắc tương tự. Có ba loại ký hiệu prèix là ordinary, Polish, và Cambridge Polish.

Ký hiệu ordinary prefix sử dụng các dấu ngoặc để bao quanh các toán hạng và dấu phẩy để phân biệt các toán hạng. Ví dụ cấu trúc cây trong hình trên sẽ trở thành:

/(+M(B),SQRT(-(^(B,),*(*(4,A),C)))),*(2,A))

Một biến thể của ký hiệu này được dùng trong ngôn ngữ LISP đôi khi được gọi là Cambridge Polish. Theo ký hiệu Cambridge Polish thì các dấu ngoặc bên trái đứng sau một toán tử được chuyển ra trước toán tử đó và dấu phẩy ngăn cách các toán hạng bị xóa đi. Cấu trúc cây trên trở thành: (/(+(M B)(SQRT(-(^ B 2)(*(* 4 A)C)))) (* 2 A))

Biến thể thứ hai được gọi là ký hiệu Polish, cho phép bỏ hẳn các dấu ngoặc. Nếu chúng ta giả sử rằng số lượng các toán hạng của mỗi một phép toán là đã biết và cố định thì các dấu ngoặc là không cần thiết. Cấu trúc cây trên sẽ trở thành: / + M B SQRT - ^ B 2 * * 4 A C * 2 A

Bởi vì nhà toán học Ba lan Lukasiewiez đã phát minh ra ký hiệu không dấu ngoặc này nên thuật ngữ "Polish" được dùng cho ký hiệu này và các biến thể của nó.

Thực tế hiển nhiên là các biểu thức kiểu này rất khó giải. Trong thực tế, chúng ta không thể giải biểu thức dạng Polish. Các dạng ordinary prefix và Cambridge Polish đòi hỏi quá nhiều dấu ngoặc và dĩ nhiên là các ký hiệu này không gần gũi với những ký hiệu đã trở thành thói quen của chúng ta. Tuy nhiên ký hiệu ordinary prefix là một ký hiệu toán học chuẩn cho hầu hết các phép toán khác các phép toán số học và logic, chẳng hạn f(x,y,z) được viết theo ký hiệu prefix. Điều quan trọng hơn là ký hiệu prefix được dùng để biểu diễn một phép toán với số lượng toán hạng bất kỳ và do đó nói chung chỉ cần học một quy tắc để viết các biểu thức bất kỳ.

Ký hiệu hậu tố (postfix)

Ký hiệu postfix tương tự như ký hiệu Prefix ngoại trừ ký hiệu phép toán đứng sau danh sách các toán hạng. Ví dụ ((A,B)+,(C,A)-)* Hoặc A B + C A - *

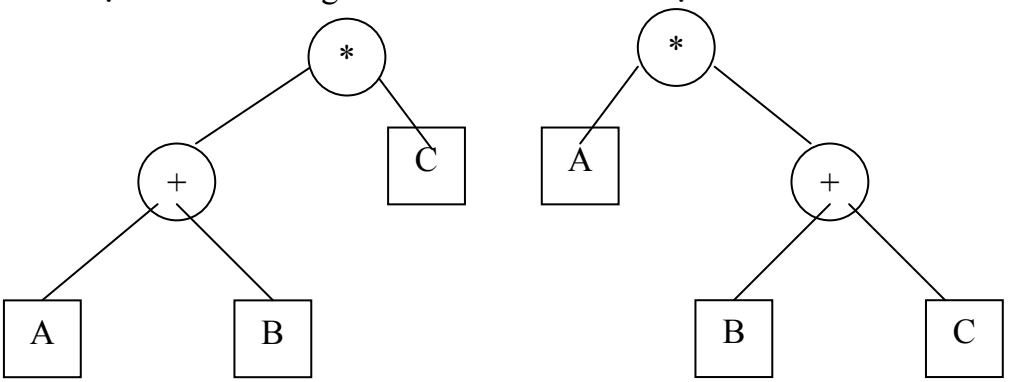
Postfix không phải là sự biểu diễn phổ biến cho biểu thức trong ngôn ngữ lập trình nhưng nó có tầm quan trọng như là cơ sở của sự biểu diễn tại thời gian thực hiện của biểu thức.

Kí hiệu trung tố (infix)

Ký hiệu trung tố thích hợp với phép toán hai ngôi tức là phép toán có hai toán hạng. Trong ký hiệu trung tố, ký hiệu phép toán được viết giữa hai toán hạng. Vì ký hiệu trung tố dùng cho các phép tính số học cơ bản, phép toán quan hệ và các phép toán logic trong toán hoc thông thường nên nó cũng được chọn để dùng một cách rộng rãi trong ngôn ngữ lập trình cho các phép toán đó và trong một số trường hợp còn được mở rộng cho các phép toán khác. Mặc dù ký hiệu trung tố được dùng một cách phổ biến, nhưng việc dùng nó trong ngôn ngữ lập trình cũng gây ra một số vấn đề nhất định:

1/ Vì ký hiệu trung tố chỉ thích hợp đối với phép toán hai ngôi nên một ngôn ngữ không chỉ dùng ký hiệu trung tố mà còn kết hợp với ký hiệu Prexfix hoặc Postfix. Điều này làm cho việc dịch trở nên phức tạp hơn.

2/ Khi có nhiều hơn một toán tử trung tố xuất hiện trong một biểu thức thì có thể xẩy ra tình trạng mập mờ, nghĩa là một biểu thức có thể biểu diễn bằng nhiều cây biểu thức. Ví dụ biểu thức trung tố: A * B + C có thể được biểu diễn thành hai cây như sau:



Dấu ngoặc có thể được dùng để chia các toán tử và toán hạng thành các nhóm, như (A * B) + C hoặc A * (B + C), nhưng trong các biểu thức phức tạp thì các dấu ngoặc lồng nhiều lớp là một trở ngại lớn cho người lập trình. Vì lý do này các ngôn ngữ thường sử dụng quy tắc điều khiển ẩn mà việc dùng dấu ngoặc là không cần thiết. Hai quy tắc ẩn phổ biến là:

a/ Quy tắc ưu tiên trước: Các phép toán xuất hiện trong biểu thức được sắp xếp theo một thứ bậc hoặc một thứ tự ưu tiên trước. Trong một biểu thức có nhiều phép toán, thứ bậc theo quy tắc ẩn là phép toán nào có bậc ưu tiên cao hơn sẽ được thực hiện trước. Ví dụ trong biểu thức A * B + C, phép nhân ưu tiên trước phép cộng nên sẽ được thực hiện trước.

b/ Quy tắc kết hợp: Trong một biểu thức có nhiều phép toán cùng cấp theo thứ tự ưu tiên thì nguyên tắc kết hợp là cần thiết để hoàn thiện việc xác định thứ tự các phép toán. Ví dụ trong biểu thức: A - B - C thì phép toán trừ thứ nhất hay phép trừ thứ hai được thực hiện trước?. Kết hợp trái (thực hiện từ trái qua phải) là nguyên tắc phổ biến nhất cho các phép toán số học, do đó A - B - C được xử lý như (A - B) - C. Tuy nhiên, có một số phép toán lại đòi hỏi sự kết hợp phải, chẳng hạn phép gán trong ngôn ngữ C. Trong ngôn ngữ C ta có thể viết a = b = 10, và thứ tự thực hiện là gán 10 cho b trước, kết quả trả về của phép gán này là 10 sẽ được gán tiếp cho a.

7.3.4 Dịch biểu thức thành biểu diễn cây

Dịch một biểu thức từ sự biểu diễn cú pháp của nó trong văn bản chương trình thành dạng có thể thực hiện là một qúa trình hai giai đoạn. Trước hết biểu thức được dịch thành biểu diễn cây của nó và sau đó cây được dịch thành một dãy các lệnh có thể thực hiện được. Giai đoạn 1 thông thường chỉ liên quan tới sự thành lập cấu trúc điều khiển cây cơ bản của biểu thức, lợi dụng quy tắc ẩn về ưu tiên trước và kết hợp khi biểu thức dùng ký hiệu trung tố. Giai đoạn thứ hai có những quyết định cụ thể liên quan tới thủ tục của sự định giá (evalution) được tạo ra bao gồm cả sự tối ưu hóa quá trình định giá.

7.3.5 Biểu diễn trong thời gian thực hiện của biểu thức

Nhiều sự biểu diễn thời gian thực của biểu thức được dùng trong cài đặt ngôn ngữ. Sau đây là một số sự lựa chọn được dùng:

1/ Dãy mã máy. Kỹ thuật phổ biến nhất là dịch các biểu thức thành dạng mã máy. Thứ tự các lệnh phản ánh cấu trúc điều khiển tuần tự của biến thức ban đầu. Biểu diễn mã máy cho phép dùng trình thông dịch của phần cứng nên thực hiện rất nhanh.

2/ Cấu trúc cây. Biểu thức có thể được thực hiện một cách trực tiếp trong biểu diễn cấu trúc cây tự nhiên của chúng, sử dụng trình thông dịch mềm. Đây là kỹ thuật cơ bản được dùng trong LISP nơi mà toàn bộ chương trình được biểu diễn như là một cấu trúc cây trong quá trình thực hiện.

3/ Dạng frefix hoặc Postfix. Biểu thức trong dạng prefix hoặc postfix có thể được thực hiện bằng giải thuật thông dịch mà nó quét biểu thức từ trái qua phải. Biểu diễn postfix có một lợi ích đặc biệt ở đây, là thứ tự của các ký hiệu trong biểu diễn postfix tương ứng với thứ tự trong đó các phép toán khác nhau phải được thực hiện. Biểu diễn prefix là dạng có thể thực hiện của chương trình trong SNOBOL4.

Các chiến lược định gia biểu thức sẽ được trình bày trong lý thuyết chương trình dịch.

7.4 ĐIỀU KHIỂN TUẦN TỰ GIỮA CÁC LỆNH

7.4.1 Các lệnh cơ bản

Lệnh cơ bản là lệnh mà trong đó không chứa các lệnh khác. Các lệnh cơ bản bao gồm lệnh gán, lời gọi chương trình con, các lệnh nhập, xuất, lệnh nhảy goto. Trong một lệnh cơ bản có thể chứa các biểu thức mà cấu trúc điều khiển đã được trình bày ở phần trên.

Các cấu trúc trong chương trình thường có là: hợp thành, lựa chọn và lặp lại. Các ngôn ngữ khác nhau cài đặt các cấu trúc này một cáh khác nhau.

7.4.2 Điều khiển tuần tự dùng nhãn lệnh và lệnh GOTO

Cơ chế ban đầu của điều khiển tuần tự trong hầu hết các ngôn ngữ là ghi nhãn lệnh và chuyển điều khiển tới lệnh có nhãn từ chỗ này sang chỗ khác trong chương trình. Việc chuyển điều khiển thường được thực hiện bằng lệnh GOTO. Có hai dạng của lệnh GOTO là:

1/ GOTO không điều kiện. Trong một chuỗi các lệnh, một lệnh GOTO không điều kiện như GOTO NEXT chuyển điều khiển tới lệnh có nhãn là NEXT. Lệnh đứng sau GOTO sẽ không được thực hiện.

2/ GOTO có điều kiện. Trong một chuỗi lệnh, một lệnh GOTO có điều kiện như IF A = 0 then GOTO NEXT chuyển điều khiển tới lệnh có nhãn là NEXT chỉ khi điều kiện sau IF đúng.

Sử dụng hai dạng GOTO này, chúng ta dễ dàng biểu diễn các dạng điều khiển cơ bản như sau

Lệnh hợp thành	Lệnh lựa chọn	Lệnh lặp lại
S0 GOTO L1 L2: S2 GOTO L3 L1: S1 GOTO L2 L3 : S3	S0 IF A=0 THEN GOTO L1 S1 GOTO L2 L1: S2 L2: S3	S0 L1: IF A=0 THEN GOTO L2 S1 GOTO L1 L2: S2
Chuỗi lệnh thực hiện S0 S1 S2 S3	Chuỗi lệnh thực hiện S0 S1 S3 Hoặc S0 S2 S3	Chuỗi lệnh thực hiện S0 S2 Hoặc S0 S1 S2 Hoặc S0 S1 S1 S2 Hoặc S0 S1 S1 S1 S2

Lệnh GOTO có thuận tiện là dễ dùng, và có hiệu quả trong thực hiện vì nó phản ánh cấu trúc cơ bản của máy tính quy ước (conventional computers), trong đó mỗi một từ lệnh hoặc byte lệnh đều có địa chỉ, và trong phần cứng có các lệnh nhảy được xây dựng để chuyển điều kkhiển đến địa chỉ được chỉ định. Lệnh GOTO biểu thị một cấu trúc điều khiển tự nhiên để người lập trình chuyển ngôn ngữ cấp cao sang hợp ngữ. Hầu hết các ngôn ngữ cũ đều có cả lệnh GOTO cơ bản và nhiều dạng cải tiến đặt nền móng cho việc dùng nhãn như là dữ liệu. Trong các ngôn ngữ mới như Pascal điều khiển tuần tự trên cơ sở lệnh GOTO ít quan trong hơn mặc dù vẫn còn lệnh đó.

Trong một số ngôn ngữ mới, lệnh GOTO đã bị loại bỏ hoàn toàn. Vì sử dụng nhãn và lệnh GOTO thì chương trình trở nên rất khó đọc vì không có cấu trúc tổng thể và thứ tự các lệnh trong văn bản chương trình nguồn không tương ứng với thứ tự các lệnh khi thực hiện.

7.4.3 Các lệnh cấu trúc

Một lệnh có cấu trúc là một lệnh chứa các lệnh khác. Các lệnh thành phần của một lệnh có cấu trúc có thể là một lệnh cơ bản hoặc một lệnh có cấu trúc. Hầu hết ngôn ngữ cung cấp một tập hợp các lệnh có cấu trúc biểu thị các dạng điều khiển cơ bản (hợp thành, lựa chọn và lặp lại) mà không cần dùng lệnh GOTO.

Lệnh hợp thành (Compound Statements)

Lệnh hợp thành là một chuỗi các lệnh được đặt vào trong một cặp ký hiệu thể hiện sự mở đầu và kết thúc của chuỗi đó. Chẳng hạn trong Pascal, lệnh hợp thành là chuỗi các lệnh được đặt trong cặp từ khóa begin và end như sau:

```
Begin
        Lệnh 1;
```

```
        Lênh 2;
        .....
        Lệnh n
End
```

Cấu trúc lệnh hợp thành cho phép một tập hợp các lênh được trừu tượng hóa thành một lệnh đơn.

Lệnh hợp thành được cài đặt trong máy tính ảo bằng cách thiết lập một khối các mã lệnh có thể thực hiện được biểu diễn cho mỗi một lệnh của chuỗi lệnh trong bộ nhớ. Thứ tự mà chúng xuất hiện trong bộ nhớ xác định thứ tự trong đó chúng được thực hiện.

Lệnh điều kiện (Conditional Statements)

Lệnh điều kiện là một lệnh biểu thị sự lựa chọn của hai hoặc nhiều lệnh. Việc lựa chọn được điều khiển bằng cách kiểm tra một số điều kiện thường được viết trong dạng biểu thức của các phép toán quan hệ và logic. Các lệnh điều kiện phổ biến là lệnh IF và lệnh CASE.

Lệnh IF được cụ thể hóa thành các dạng IF một nhánh, IF hai nhánh và IF đa nhánh.

Chọn thực hiện một lệnh được biểu thị là IF một nhánh: IF <điều kiện> THEN <Lệnh> ENDIF

Chọn một trong hai dùng IF hai nhánh: IF <điều kiện> THEN <Lệnh1> ELSE <Lệnh2> ENDIF

Chọn một trong nhiều dùng các IF nối tiếp nhau hoặc dùng IF đa nhánh:

```
IF <điều kiện1> THEN <Lệnh1>
ELSIF<điều kiện2> THEN <Lệnh2>
   .
   .
   .
ELSIF <điều kiệnN> THEN <LệnhN>
ELSE <LệnhN+1> ENDIF
```

Lệnh CASE

Điều kiện trong lệnh If đa nhánh thường phải lặp lại việc kiểm tra giá trị của một biến, ví dụ:

```
IF TAG = 0 THEN
    <Lệnh 0>
ELSIF TAG = 1 THEN
    <Lệnh 1>
ELSIF TAG = 2 THEN
    <Lệnh 2>
ELSE
    <Lệnh 3>
ENDIF
```

Cấu trúc phổ biến này được biểu diễn một cách súc tích hơn bằng lệnh CASE

```
CASE TAG OF
    0: <Lệnh 0>
```

1: <Lệnh 1>
2: <Lệnh 2>
OTHERS: <Lệnh3>
ENDCASE

Cài đặt lệnh điều kiện

Lệnh IF được cài đặt bằng cách dùng lệnh rẽ nhánh và lệnh nhảy có điều kiện hoặc không có điều kiện trong phần cứng. Kết quả tương tự như đã mô tả trong phần 7.3.2.

Lệnh lặp (Interation Statements)

Lặp lại đơn, kiểu đơn giản nhất của lệnh lặp xác định phần thân (của lệnh) được thực hiện một số cố định lần. Lệnh PERFORM của COBOL là một điển hình: PERFORM <thân> k TIMES

Lặp lại khi điều kiện đúng: WHILE < test > DO <thân>

Lặp lại trong khi tăng một sự đếm: FOR i:=1 STEP 2 UNTIL 30 DO <thân>

Lặp không xác định, trong đó điều kiện để thoát khỏi vòng lặp không đặt tại đầu vòng, như trong Ada:

LOOP

.....

EXIT WHEN <điều kiện>

.....

END LOOP;

Hoặc trong Pascal sử dụng vòng lặp WHILE với điều kiện luôn luôn đúng: WHILE true DO BEGIN .... END;

Cài đặt các lệnh lặp dùng các chỉ thị rẽ nhánh/ nhảy của phần cứng.

7.5 SỰ NGOẠI LỆ VÀ XỬ LÝ NGOẠI LỆ

7.5.1 Một số khái niệm

Trong quá trình thực hiện chương trình thường xẩy ra một số sự kiện đặc biệt hoặc các lỗi như sự tràn số, truy xuất đến chỉ số mảng nằm ngoài tập chỉ số, thực hiện lệnh đọc một phần tử cuối tập tin... Các sự kiện đó được gọi là ngoại lệ (exception). Thay vì tiếp tục thực hiện chương trình bình thường, một chương trình con sẽ được gọi để thực hiện một vài xử lý đặc biệt nào đó gọi là xử lý ngoại lệ. Hành động chú ý đến ngoại lệ, ngắt sự thực hiện chương trình và chuyển điều khiển đến xử lý ngoại lệ được gọi là đề xuất ngoại lệ (raising the exception)

7.5.2 Xử lý ngoại lệ

Thông thường các ngoại lệ đã được định nghĩa trước bởi ngôn ngữ, chẳng hạn như ZERO_DIVIDE chỉ sự kiện chia cho một số không, END_OF_FILE: hết tập tin , OVERFLOW: tràn số, hay tràn stack ... Xử lý ngoại lệ là một hành vi xử lý tương ứng khi một ngoại lệ có thể diễn ra. Ví dụ

void example () {

```
        ......
        average = sum/total;
        ...
        return ;
        when zero_divide {
                average = 0;
                printf(" error: cannot compute average, total is zero\n");
        }
        ......
} /** function example **/
```

7.5.3 Đề xuất một ngoại lệ

Một ngoại lệ có thể bị đề xuất bằng phép toán nguyên thuỷ được định nghĩa bởi ngôn ngữ chẳng hạn phép cộng, phép nhân có thể đề xuất ngoại lệ OVERFLOW. Ngoài ra, một ngoại lệ có thể bị đề xuất một cách tường minh bởi người lập trình bằng cách dùng một lệnh được cung cấp cho mục đích đó. Chẳng hạn trong Ada: raise BAD_DATA_VALUE;

Lệnh này có thể được thực hiện trong một chương trình con sau khi xác định một biến riêng hoặc tập tin nhập chứa giá trị không đúng.

7.5.4 Lan truyền một ngoại lệ (Propagating an exception)

Thông thường, khi xây dựng chương trình thì vị trí mà một ngoại lệ xuất hiện không phải là vị trí tốt nhất để xử lý nó. Khi một ngoại lệ được xử lý trong một chương trình con khác chứ không phải trong chương trình con mà nó được đề xuất thì ngoại lệ đó được gọi là được truyền (propagated) từ điểm mà tại đó nó được đề xuất đến điểm mà nó được xử lý.

Quy tắc để xác định việc xử lý một ngoại lệ đặc thù thường được gọi là chuỗi động (dynamic chain) của các kích hoạt chương trình con hướng tới chương trình con mà nó đề xuất ngoại lệ. Khi một ngoại lệ P được đề xuất trong chương trình con C, thì P được xử lý bởi một xử lý được định nghĩa trong C nếu có một cái xử lý như thế. Nếu không có thì C kết thúc. Nếu chương trình con B gọi C thì ngoại lệ được truyền đến B và một lần nữa được đề xuất tại điểm trong B nơi mà B gọi C. Nếu B không cung cấp một xử lý cho P thì B bị kết thúc và ngoại lệ lại được truyền tới chương trình gọi B vân vân... Nếu các chương trình con và bản thân chương trình chính không có xử lý cho P thì toàn bộ chương trình kết thúc và xử lý chuẩn được định nghĩa bởi ngôn ngữ sẽ được gọi tới.

Một hiệu quả quan trọng của quy tắc này đối với việc truyền các ngoại lệ là nó cho phép một chương trình con kế thừa (remain) như là một phép toán trừu tượng được định nghĩa bởi người lập trình ngay cả trong việc xử lý ngoại lệ. Một phép toán nguyên thuỷ có thể bất ngờ ngắt quá trình bình thường của nó và đề xuất một ngoại lệ. Tương tự, thông qua việc thực hiện lệnh RAISE, một chương trình con có thể bất ngờ ngắt quá trình bình thường của nó và đề xuất một ngoại lệ. Đến chương trình gọi thì

hiệu quả của đề xuất ngoại lệ của chương trình con cũng giống như hiệu quả đề xuất của phép toán nguyên thủy, nếu chương trình con tự nó không có một xử lý ngoại lệ. Nếu ngoại lệ được xử lý trong chương trình con thì chương trình con trở về một cách bình thường và chương trình gọi nó không bao giờ biết được rằng một ngoại lệ đã được đề xuất.

7.5.5 Sau khi một ngoại lệ được xử lý

Sau khi một xử lý đã hoàn thành việc xử lý một ngoại lệ và xử lý đó đã kết thúc thì có một vấn đề đặt ra là quyền điều khiển được chuyển tới chỗ nào? Điều khiển nên được chuyển tới chỗ mà ngoại lệ được đề xuất? Điều khiển nên chuyển về lệnh trong chương trình con chứa xử lý nơi mà ngoại lệ được đề xuất sau khi được truyền tới? Chương trình con chứa xử lý tự kết thúc một cách bình thường và nó xuất hiện tại chương trình gọi như là không có gì xẩy ra. Đây là những lựa chọn khi thiết kế ngôn ngữ.

7.6 CÂU HỎI ÔN TẬP

1. Thế nào là điều khiển tuần tự?
2. Xét về mặt cấu trúc thì có những loại điều khiển tuần tự nào?
3. Xét về mặt thiết kế ngôn ngữ thì có những loại điều khiển tuần tự nào?
4. Trong biểu diễn trung tố một biểu thức, để khắc phục tình trạng một biểu thức có thể có nhiều cây biểu thức (tình trạng mập mờ), người ta thường sử dụng các quy tắc gì?

CHƯƠNG 8: LẬP TRÌNH HÀM

8.1 TỔNG QUAN

8.1.1 Mục tiêu

Sau khi học xong chương này, sinh viên cần phải nắm:

- Khái niệm về lập trình hàm.
- Kỹ thuật lập trình đệ qui.
- Các cấu trúc cơ bản của ngôn ngữ LISP

8.1.2 Nội dung cốt lõi

- Lập trình hàm.
- Căn bản về ngôn ngữ lập trình LISP.

8.1.3 Kiến thức cơ bản cần thiết

Kiến thức và kĩ năng lập trình căn bản.

8.2 NGÔN NGỮ LẬP TRÌNH HÀM

8.2.1 Giới thiệu

Hầu hết các ngôn ngữ lập trình từ trước đến nay được xây dựng dựa trên nguyên lý kiến trúc máy tính Von Neumann. Lớp chủ yếu trong các ngôn ngữ đó là các ngôn ngữ ra lệnh. Đơn vị làm việc trong một chương trình là câu lệnh. Kết quả của từng câu lệnh được tổ hợp lại thành kết quả của cả chương trình. Các ngôn ngữ này bao gồm: FORTRAN, COBOL, Pasacl, Ada... Mặc dù ngôn ngữ ra lệnh đã được hầu hết người lập trình chấp nhận nhưng sự liên hệ chặt chẽ với kiến trúc máy tính là một hạn chế đến việc phát triển phần mềm.

Ngôn ngữ lập trình hàm được thiết kế dựa trên các hàm toán học là một trong những ngôn ngữ không ra lệnh quan trọng nhất. Trong đó LISP là một ngôn ngữ tiêu biểu.

8.2.2 Hàm toán học

Hàm là một sự tương ứng giữa các phần tử của một tập hợp (miền xác định) với các phần tử của một tập hợp khác (miền giá trị). Định nghĩa hàm xác định miền xác định, miền giá trị và quy tắc tương ứng giữa các phần tử của miền xác định với các phần tử của miền giá trị. Thông thường sự tương ứng được mô tả bởi một biểu thức. Hàm toán học có hai đặc trưng cơ bản là:

- Thứ tự đánh giá biểu thức được điều khiển bởi sự đệ quy và biểu thức điều kiện chứ không phải bằng cách lặp lại và liên tiếp như trong các ngôn ngữ ra lệnh.
- Hàm toán học không có hiệu ứng lề cho nên với cùng một tập đối số, hàm toán học luôn cho cùng một kết quả.

Định nghĩa hàm thường được viết bởi tên hàm, danh sách các tham số nằm trong cặp dấu ngoặc và sau đó là biểu thức, ví dụ: lap_phuong(x) ≡ x*x*x trong đó x là một số thực. Miền xác định, miền giá trị là các tập số thực.

Lúc áp dụng, một phần tử cụ thể của miền xác định gọi là đối sẽ thay thế cho tham số trong định nghĩa hàm. Kết quả hàm thu được bằng cách đánh giá biểu thức hàm. Ví dụ lap_phuong(2.0) cho giá trị là 8.0. Trong định nghĩa hàm, x đại diện cho mọi phần tử của miền xác định. Trong lúc áp dụng, nó được cho một giá trị (chẳng hạn 2.0), giá trị của nó không thay đổi sau đó. Điều này trái ngược với biến trong lập trình có thể nhận các giá trị khác nhau trong quá trình thực hiện chương trình.

Trong định nghĩa hàm, ta bắt cặp tên hàm với biểu thức x*x*x. Đôi khi người ta sử dụng hàm không tên, trong trường hợp đó người ta sử dụng biểu thức lambda. Giá trị của biểu thức lambda chính là hàm của nó. Ví dụ λ(x)x*x*x. Tham số trong biểu thức lambda được gọi là biến kết ghép. Khi biểu thức lambda được đánh giá đối với một tham số đã cho, người ta nói rằng biểu thức được áp dụng cho tham số đó.

8.2.3 Dạng hàm

Dạng hàm là sự tổ hợp của các hàm. Dạng hàm phổ biến nhất là hàm hợp. Nếu f được định nghĩa là hàm hợp của g và h, được viết là f ≡ g.h thì việc áp dụng f được định nghĩa là sự áp dụng h sau đó áp dụng g lên kết quả.

Xây dựng (construction) là một dạng hàm mà các tham số của chúng là những hàm. Người ta ký hiệu một xây dựng bằng cách để các hàm tham số vào trong cặp dấu ngoặc vuông. Khi áp dụng vào một đối số thì các hàm tham số sẽ được áp dụng vào đối đó và tập hợp các kết quả vào trong một danh sách. Ví dụ: G(x) ≡ x*x, H(x) ≡ 2*x và I(x) ≡ x/2 thì [G,H,I](4) có kết quả là (16,8,2).

Áp dụng cho tất cả là một dạng hàm mà nó lấy một hàm đơn như là một tham số. Áp dụng cho tất cả được ký hiệu là ∝. Nếu áp dụng vào một danh sách các đối thì áp dụng cho tất cả sẽ áp dụng hàm tham số cho mỗi một giá trị và tập hợp các kết quả vào trong một danh sách. Ví dụ

Cho h(x) ≡ x*x thì ∝(h, (2,3,4)) có kết quả là (4,9,16)

8.2.4 Bản chất của ngôn ngữ lập trình hàm

Mục đich của việc thiết kế ngôn ngữ lập trình hàm là mô phỏng các hàm toán học một cách nhiều nhất có thể được. Trong ngôn ngữ ra lệnh, một biểu thức được đánh giá và kết quả của nó được lưu trữ trong ô nhớ được biểu diễn bởi một biến trong chương trình. Ngược lại, trong ngôn ngữ lập trình hàm không sử dụng biến và do đó không cần lệnh gán. Điều này giải phóng người lập trình khỏi mối quan tâm về ô nhớ của máy tính trong khi thực hiên chương trình. Không có biến cho nên không có cấu trúc lặp (vì cấu trúc lặp được điều khiển bởi biến). Các lệnh lặp lại sẽ được xử lý bằng giải pháp đệ quy. Chương trình là các định nghĩa hàm và các áp dụng hàm. Sự thực hiện là việc đánh giá các áp dụng hàm. Sự thực hiện một hàm luôn cho cùng một kết quả khi ta cho nó cùng một đối số. Điều này gọi là trong suốt tham khảo (referential transparancy). Nó cho thấy rằng ngữ nghĩa của ngôn ngữ lập trình hàm đơn giản hơn ngữ nghĩa của ngôn ngữ lập trình ra lệnh và ngôn ngữ hàm bao gồm cả những nét đặc biệt của ngôn ngữ ra lệnh.

Ngôn ngữ hàm cung cấp một tập hợp các hàm nguyên thủy, một tập các dạng hàm để xây dựng các hàm phức tạp từ các hàm đã có. Ngôn ngữ cũng cung cấp một phép toán áp dụng hàm và các cấu trúc lưu trữ dữ liệu. Một ngôn ngữ hàm được thiết kế tốt là

một ngôn ngữ có tập hợp nhỏ các hàm nguyên thủy. Phần sau chúng ta làm quen với một ngôn ngữ lập trình hàm khá nổi tiếng là ngôn ngữ LISP.

8.3 NGÔN NGỮ LISP

8.3.1 Giới thiệu:

Được J. MAC CARTHY viết năm 1958, LISP là một trong những ngôn ngữ lập trình sớm nhất. Đầu năm những năm 80, LISP được phát triển mạnh nhờ những áp dụng trong lĩnh vực trí tuệ nhân tạo. LISP có các ưu điểm chính như sau:

- Cú pháp đơn giản. Trong LISP chỉ có một cấu trúc duy nhất là cấu trúc danh sách (LISP là ngôn ngữ xử lý danh sách: LISP = LISt Processing language), không có lệnh, không có từ khóa, tất cả các hàm đều được viết dưới dạng danh sách.
- Là một ngôn ngữ mạnh nhờ tính tương đương giữa dữ liệu và chương trình: dữ liệu và chương trình đều là danh sách, chúng có thể thao tác nhờ chung một công cụ.
- Mềm dẻo và dễ phát triển.

8.3.2 Các khái niệm cơ bản

Nguyên tử (atom)

Nguyên tử là một đối tượng cơ bản của LISP, nguyên tử có thể là số hoặc ký hiệu.

- **Số**. Dữ liệu số trong LISP cũng giống như trong một số ngôn ngữ lập trình khác như Pascal, C…

Ví dụ về các hằng số: 5, -17, 5.35, 3/4, 118.2E+5,...

- **Ký hiệu (symbol)** là một chuỗi các ký tự (trừ các ký tự đặc biệt, dấu ngoặc và khoảng trống). Các hằng ký hiệu được viết mở đầu bằng dấu nháy đơn ‘.

Ví dụ về các hằng ký hiệu: ‘a, ‘anh, ‘anh_ba,...

Một số ký hiệu được định nghĩa trước như: T (về mặt logic, được hiểu là TRUE), NIL (về mặt logic, được hiểu là FALSE).

Hằng ký hiệu số được xem như là một số, chẳng hạn ‘5 = 5.

Danh sách

Danh sách là một dãy có phân biệt thứ tự của các phần tử cách nhau ít nhất một khoảng trắng và đặt nằm trong cặp dấu ngoặc đơn ().

Phần tử của danh sách có thể là một nguyên tử hoặc là một danh sách.

Hằng danh sách được mở đầu bằng dấu nháy đơn ‘.

Ví dụ về các hằng danh sách:

- ‘() Danh sách rỗng, tương đương ký hiệu NIL.
- ‘(a 5 c) Danh sách gồm 3 phần tử.
- ‘(3 (b c) d (e (f g))) Danh sách gồm 4 phần tử, trong đó phần tử thứ 2 và phần tử thứ 4 lại là các danh sách.

Biểu thức

Biểu thức là một nguyên tử hoặc một danh sách. Biểu thức luôn có một giá trị mà việc định trị nó theo nguyên tắc sau:

- Nếu biểu thức là một số, thì giá trị của biểu thức là giá trị của số đó.

  Ví dụ:

  > 25

  = 25

- Nếu biểu thức là một ký hiệu thì giá trị của biểu thức có thể là
  - Được xác định trước bởi LISP (chẳng hạn t có giá trị là T (TRUE) và nil có giá trị là NIL một danh sách rỗng) hoặc
  - Một giá trị dữ liệu của người sử dụng hoặc trong chương trình được gán cho một biến. Biến không cần phải khai báo.

  Ví du:

  > (setq a 3) ; Gán số 3 cho biến có tên a

  = 3

  > a ; hỏi giá trị của ký hiệu "a"

  = 3

- Nếu biểu thức là một danh sách có dạng (E_0 E_1 ... E_n) thì giá trị của biểu thức được xác định theo cách sau đây:
  - Phần tử đầu tiên E_0 phải là một hàm đã được LISP nhận biết.
  - Các phần tử E_1, E_2, ..., E_n được định trị tuần tự từ trái sang phải. Giả sử ta có các giá trị tương ứng là V_1, V_2, ..., V_n
  - Hàm E_0 được áp dụng cho các đối V_1, V_2, ..., V_n. Giá trị của hàm E_0 chính là giá trị của biểu thức.

  Ví dụ

  > (+ 5 3 6)

  = 14

  > (+ 4 (+ 3 5))

  = 12

  - **Chú ý:** Nếu biểu thức dùng hàm QUOTE hoặc dấu nháy đơn sẽ không được đánh giá

  Ví dụ:

  > '(+ 1 2)

  = (+ 1 2)

8.3.3 Các hàm

Một chương trình của LISP là một hàm hoặc một hàm hợp. Các hàm có thể do LISP định nghĩa trước hoặc do lập trình viên tự định nghĩa.

Một số hàm định nghĩa trước

- **Các hàm số học: +, -, *, /, 1+, 1-, MOD, SQRT tác động lên các biểu thức số và cho kết quả là một số.**

Ví dụ:

```
> (+  5  6  2)
= 13
> (-  8  3)
= 5
> (-  8  3  1)
= 4
>(1+  5)  ; Tương đương (+  5  1)
= 6
> (1-  5)   ; Tương đương (-  5  1)
= 4
>(MOD  14  3)
= 2
>(sqrt 9) ; Lấy căn bậc hai của 9
= 3
```

- **Các hàm so sánh các số <, >, <=, >=, = và /=, cho kết quả là T hoặc NIL**

  Ví dụ:

```
>(<  4  5)
= T
>(>  4  (*  2  3))
= NIL
```

- **(EQ s1 s2) so sánh xem hai ký hiệu s1 và s2 có giống nhau hay không?**

  Ví dụ:

```
>(eq  ‘tuong  ‘tuong)
= T
>(eq  ‘tuong  ‘duong)
= NIL
>(eq  ‘5  5  )
```

= T

- **(EQUAL o1 o2) so sánh xem đối tượng bất kỳ o1 và o2 có giống nhau hay không?**

Ví dụ:

>(equal ‘(a b c) ‘(a b c))

= T

>(equal ‘(a b c) ‘(b a c))

= NIL

>(equal ‘a ‘a)

= T

- **Các hàm thao tác trên danh sách: CAR, CDR, CONS và LIST**
  - (CAR L) nhận vào danh sách L, trả về phần tử đầu tiên của L.

Ví du:

> (CAR '(1 2 3))

= 1

> (CAR 3)

Error: bad argument type - 3

>(CAR nil)

= NIL

> (CAR '((a b) 1 2 3))

= (A B)

  - (CDR L) nhận vào danh sách L, trả về một danh sách bằng phần còn lại của danh sách L sau khi bỏ đi phần tử đầu tiên.

Ví dụ:

>(cdr '(1 2 3))

= (2 3)

>(cdr 3)

Error: bad argument type - 3

>(cdr nil)

= NIL

>(cdr '(1))

= NIL

>(CAR (CDR ‘(a b c)))

= B

- **Viết gộp các hàm: T**a có thể dùng hàm C..A/D..R để kết hợp nhiều CAR và CDR (có thể thay thế việc lồng nhau tới 4 cấp)

Ví du:

(CADR '(a b c))

= B

- (CONS x L) nhận vào phần tử x và danh sách L, trả về một danh sách, có được bằng cách thêm phần tử x vào đầu danh sách L

Ví du:

>(CONS 3 '(1 2 3))

= (3 1 2 3)

>(CONS 3 nil)

= (3)

>(CONS '(a b) '(1 2 3))

= ((A B) 1 2 3)

- (LIST E_1 E_2 ... E_n) nhận vào n biểu thức E_1, E_2, ..., E_n, trả về danh sách bao gồm n phần tử V_1, V_2, ..., V_n, trong đó V_i là giá trị của biểu thức E_i (i=1..n) .

Ví du:

>(list 1 2)

= (1 2)

>(list 'a 'b)

= (A B)

>(list 'a 'b (+ 2 3 5))

= (A B 10)

- **Các vị từ kiểm tra**
  - (ATOM a) xét xem a có phải là một nguyên tử.
  - (NUMBERP n) xét xem n có phải là một số.
  - (LISTP L) xét xem L có phải là một danh sách.
  - (SYMBOLP S) xét xem S có phải là một ký hiệu.
  - (NULL L) nhận vào 1 danh sách L. Nếu L rỗng thì trả về kết quả là T, ngược lại thì trả về kết quả là NIL.

Ví du:

>(atom 'a)

= T

>(numberp 4)

= T

```
>(symbolp 'a)
= T
>(listp '(1  2))
= T
>(symbolp NIL)
= T
>(listp NIL)
= T
>(null NIL)
= T
>(null ‘(a  b))
= NIL
>(null 10)
= NIL
```

- **Các hàm logic AND, OR và NOT**
  - (AND E_1 E_2... E_n) nhận vào n biểu thức E_1, E_2,... E_n. Hàm AND định trị các biểu thức E_1 E_2... E_n từ trái sang phải. Nếu gặp một biểu thức là NIL thì dừng và trả về kết quả là NIL. Nếu tất cả các biểu thức đều khác NIL thì trả về giá trị của biểu thức E_n.

Ví dụ:

```
>(AND (>  3  2) (=  3  2) (+  3  2))
= NIL
>(AND (>  3  2) (- 3  2) (+ 3  2))
= 5
```

  - (OR E_1 E_2 ... E_n) nhận vào n biểu thức E_1, E_2,... E_n. Hàm OR định giá các biểu thức E_1 E_2... E_n từ trái sang phải. Nếu gặp một biểu thức khác NIL thì dừng và trả về kết quả là giá trị của biểu thức đó. Nếu tất cả các biểu thức đều là NIL thì trả về kết quả là NIL.

Ví du:

```
>(OR (=  3  2) (+  2  1) (list  1  2))
= 3
>(OR  (=  2  1)  (Cdr ‘(a) )  (listp 3 ))
= NIL
```

  - (NOT E) nhận vào biểu thức E. Nếu E khác NIL thì trả về kết quả là NIL, ngược lại thì trả về kết quả là T.

- **Các hàm điều khiển**
  - (IF E_1 E_2 E_3) nhận vào 3 biểu thức E_1, E_2 và E_3. Nếu E_1 khác NIL thì hàm trả về giá trị của E_2 ngược lại trả về giá trị của E_3
  - (IF E_1 E_2) tương đương (IF E_1 E_2 NIL)
  - Nếu E_2 khác NIL thì (IF E_1 E_2 E_3) tương đương (OR (AND E_1 E_2) E_3)
  - (COND ($ĐK_1$ E_1)
    ($ĐK_2$ E_2)
    .....
    ($ĐK_n$ E_n)
    [(T E_{n+1})]
)

    Nếu $ĐK_1$ khác NIL thì trả về kết quả là giá trị của E_1, ngược lại sẽ xét $ĐK_2$. Nếu $ĐK_2$ khác NIL thì trả về kết quả là giá trị của E_2, ngược lại sẽ xét $ĐK_3$...

    .....

    Nếu $ĐK_n$ khác NIL thì trả về kết quả là giá trị của E_n, ngược lại sẽ trả về NIL hoặc trả về kết quả là giá trị của E_{n+1} (trong trường hợp ta sử dụng (T E_{n+1}))
  - (PROGN E_1 E_2 ... E_n) nhận vào n biểu thức E_1, E_2,... E_n. Hàm định trị các biểu thức E_1, E_2,... E_n từ trái sang phải và trả về kết quả là giá trị của biểu thức E_n.
  - (PROG1 E_1 E_2 ... E_n) nhận vào n biểu thức E_1, E_2,... E_n. Hàm định trị các biểu thức E_1, E_2,... E_n từ trái sang phải và trả về kết quả là giá trị của biểu thức E1.

Hàm do người lập trình định nghĩa

Cú pháp định nghĩa hàm là:

```
(defun <tên hàm> <danh sách các tham số hình thức>
        <biểu thức>
)
```

Ví dụ 1: Định nghĩa hàm lấy bình phương của số a

```
(defun binh_phuong (a)
    (* a  a)
)
```

Sau khi nạp hàm này cho LISP, ta có thể sử dụng như các hàm đã được định nghĩa trước.

```
>(binh_phuong 5)
= 25
>(binh_phuong (+ 5 2))
= 49
```

Ví dụ 2: Định nghĩa hàm DIV chia số a cho số b, lấy phần nguyên.

Trước hết ta có: a DIV b = (a – a MOD b)/b

```
(defun DIV (a  b)
        (/  (-  a  (MOD a  b)) b)
)
```

8.3.4 Đệ quy

Một hàm đệ quy là một hàm có lời gọi chính nó trong biểu thức định nghĩa hàm. Mô tả một đệ quy bao gồm:

- Có ít nhất một trường hợp "dừng" để kết thúc việc gọi đệ quy.
- Lời gọi đệ quy phải bao hàm yếu tố dẫn đến các trường hợp "dừng".

Ví dụ 1: Viết hàm tính n giai thừa

Công thức đệ quy tính n giai thừa là $n! = \begin{cases} 1 \text{ neu } n = 0 \\ n * (n-1)! \end{cases}$

Hàm (giai_thua N) viết bằng ngôn ngữ LISP:

```
(defun giai_thua (n)
        (if (=  n  0) 1 ; trường hợp "dừng"
           (*  n  (giai_thua (1-  n)));  n-1 là yếu tố dẫn đến trường hợp dừng
         ) ; If
)
```

Ví dụ 2: Viết hàm DIV chia a cho b lấy phần nguyên, viết bằng đệ quy.

Công thức đệ quy: $a \text{ DIV } b = \begin{cases} 0 \text{ neu } a < b \\ 1 + (a-b) \text{ DIV } b \end{cases}$

Hàm (DIV a b) viết bằng LISP:

```
(defun DIV (a b)
        (if (<  a  b) 0 ; Trường hợp "dừng"
           (1+    (DIV  (-  a  b) b)); a-b là yếu tố dẫn đến trường hợp dừng
         ) ; If
)
```

Ví dụ 3: Viết hàm (phan_tu i L), nhận vào số nguyên dương i và danh sách L. Hàm trả về phần tử thứ i trong danh sách L hoặc thông báo "không tồn tại".

Công thức đệ quy:

$$\text{Phan tu thu i trong DS L} = \begin{cases} \text{"Khong ton tai" neu DS L rong} \\ \text{Phan tu dau tien cua L neu } i = 1 \\ \text{Phan tu thu } (i-1) \text{ trong DS "duoi" cua L} \end{cases}$$

Hàm (phan_tu i L) viết bằng LISP:

```
(defun phan_tu(i  L)
        (cond
            ((Null  L) "Khong ton tai")
            ((=  i  1) (car  L)); trường hợp dừng thứ hai
            (T (phan_tu (1-   i) (cdr   L)))
```

```
            ) ; cond
    )
```

Trong chương trình trên, (null L) là trường hợp "dừng" thứ nhất; (= i 1) là trường hợp "dừng" thứ hai; (cdr L) là yếu tố dẫn đến trường hợp "dừng" thứ nhất và (1- i) yếu tố dẫn đến trường hợp "dừng" thứ hai.

8.3.5 Các hàm nhập xuất

- **(LOAD <Tên tập tin>)**

Nạp một tập tin vào cho LISP và trả về T nếu việc nạp thành công, ngược lại trả về NIL. Tên tập tin là một chuỗi kí tự có thể bao gồm cả đường dẫn đến nơi lưu trữ tập tin đó. Tên tập tin theo quy tắc của DOS, nghĩa là chỉ có tối đa 8 ký tự trong phần tên và 3 ký tự phần mở rộng và không chứa các ký tự đặc biệt.

Ta có thể sử dụng LOAD để nạp một tập tin chương trình của LISP trước khi gọi thực hiện các hàm đã được định nghĩa trong tập tin đó.

Ví dụ:

```
>(Load "D:\btlisp\bai1.lsp")
```

- **(READ)**

Đọc dữ liệu từ bàn phím cho đến khi gõ phím Enter, trả về kết quả là dữ liệu được nhập từ bàn phím.

- **(PRINT E)**

In ra màn hình giá trị của biểu thức E, xuống dòng và trả về giá trị của E.

- **(PRINC E)**

In ra màn hình giá trị của biểu thức E (không xuống dòng) và trả về giá trị của E.

- **(TERPRI)**

Đưa con trỏ xuống dòng và trả về NIL.

8.3.6 Biến toàn cục và biến cục bộ

Biến toàn cục

Biến toàn cục (global variables) là biến mà phạm vi của nó là tất cả các hàm. Biến toàn cục sẽ tự động giải phóng khi chương trình dịch LISP kết thúc.

- **Hàm (SETQ <tên biến> <biểu thức>)**

Gán trị của <biểu thức> cho <tên biến> và trả về kết quả là giá trị của <biểu thức>.

Ví dụ:

```
>(setq  x (*  2  3))
= 6
> x  ; biến x vẫn còn tồn tại và có giá trị là 6
```

= 6

Biến cục bộ

Biến cục bộ (local variables) là biến mà phạm vi của nó chỉ nằm trong hàm mà nó được tạo ra. Biến cục bộ sẽ tự động giải phóng hàm tạo ra nó kết thúc.

- **(LET ((var1 E1) (var2 E2) ... (vark Ek)) Ek+1 ... En)**

Ta thấy hàm này có 2 phần: phần gán trị cho các biến và phần định trị các biểu thức.

Gán trị của biểu thức E_i cho biến cục bộ var_i tương ứng và thực hiện (PROGN E_{k+1} ... E_n).

Ví dụ:

>(Let ((a 3) (b 5)) (* a b) (+ a b))

= 8

> a ; biến a lúc này đã được giải phóng nên LISP sẽ thông báo lỗi

error: unbound variable - A

Biến cục bộ che biến toàn cục

Trong lập trình hàm, người ta rất hạn chế sử dụng biến, nếu thật sự cần thiết thì nên sử dụng biến cục bộ. Tuy nhiên việc khai báo biến cục bộ trong hàm LET gây khó khăn cho việc viết chương trình hơn là sử dụng biến toàn cục. Để khắc phục tình trạng này, ta sẽ kết hợp cả hai hàm LET và SETQ để sử dụng biến cục bộ che biến toàn cục. Cách làm như sau:

- Trong phần gán trị cho biến của LET ta tạo ra một biến và gán cho nó một giá trị bất kỳ, chẳng hạn số 0.
- Trong phần định trị các biểu thức, ta có thể sử dụng SETQ để gán trị cho biến đã tạo ra ở trên, biến này sẽ là một biến cục bộ chứ không còn là toàn cục nữa.
- Cụ thể chúng ta có thể viết:

```
(LET  ( (var   E1)…..)
            …….
            (SETQ  var  E2)
            ……
)
```

Với cách làm này thì biến var trong hàm SETQ sẽ trở thành biến cục bộ.

Ví dụ: Giả sử ta đã định nghĩa được hàm (ptb2 a b c), giải phương trình bậc hai $ax^2+bx+c = 0$. Bây giờ ta viết hàm (giai_ptb2) cho phép nhập các hệ số a, b, c từ bàn phím và gọi hàm (ptb2 a b c) để thực hiện việc giải phương trình. Có hai phương pháp để viết hàm này.

Phương pháp 1: dùng các biến toàn cục a, b, c

```
(defun giai_ptb2 ()
        (progn
                (print "Chương trình giải phương trình bậc hai")
```

```
                (princ "Nhập hệ số a: ") (setq  a (read))
                (princ "Nhập hệ số b: ") (setq  b (read))
                (princ "Nhập hệ số c: ") (setq  c (read))
                (ptb2  a  b  c)
        )
    )
```

Sau khi thực hiện chương trình này, thì các biến toàn cục a, b và c vẫn còn.

Phương pháp 2: dùng các biến cục bộ d, e, f

```
    (defun giai_ptb2 ()
        (let ((d  0) (e  0) (f  0))
                (print "Chương trình giải phương trình bậc hai")
                (princ "Nhập hệ số a: ") (setq  d (read))
                (princ "Nhập hệ số b: ") (setq  e (read))
                (princ "Nhập hệ số c: ") (setq  f (read))
                (ptb2  d  e  f)
        )
    )
```

Sau khi thực hiện chương trình này, thì các biến cục bộ d, e và f được giải phóng.

8.3.7 Hướng dẫn sử dụng LISP

Sử dụng XLISP

XLISP là một trình thông dịch, chạy dưới hệ điều hành Windows. Chỉ cần chép tập tin thực thi XLISP.EXE có dung lượng 288Kb vào máy tính của bạn là có thể thực hiện được.

Để thực hiện các hàm, chỉ cần gõ trực tiếp hàm đó vào sau dấu chờ lệnh (>) của XLISP. Trong trường hợp không có dấu chờ lệnh, hãy dùng menu Run/Top level hoặc Ctrl-C để làm xuất hiện dấu chờ lệnh.

Việc định nghĩa một hàm cũng có thể gõ trực tiếp vào sau dấu chờ lệnh. Tuy nhiên cách làm này sẽ khó sửa chữa hàm đó và do vậy ta thường định nghĩa các hàm trong một tập tin chương trình, sau đó nạp vào cho XLISP để sử dụng.

Ta có thể lưu trữ lại tình trạng làm việc hiện hành vào trong tập tin .WKS bằng cách dùng menu File/Save workspace và sau đó có thể khôi phục lại bằng cách dùng menu File/Restore workspace.

Soạn thảo tập tin chương trình

Do XLISP không có công cụ để soạn thảo chương trình nên ta có thể sử dụng Notepad để soạn thảo tập tin chương trình.

Trong một tập tin chương trình ta có thể định nghĩa nhiều hàm.

Lưu tập tin chương trình có tên theo quy định của DOS (8.3) với phần mở rộng .LSP và để trong cặp dấu nháy kép.

Nạp hàm tự định nghĩa cho XLISP

Có hai phương pháp để nạp các hàm tự định nghĩa cho XLISP:

- Phương pháp 1: Copy và dán khối
  - Trong Notepad, đánh dấu khối một hàm tự định nghĩa và copy khối (Edit/Copy hoặc Ctrl-C).
  - Trong XLISP, dán khối tại dấu chờ lệnh (Edit/Paste hoặc Ctrl-Ins).
  - Với phương pháp này thì khi viết các hàm, không nên viết một dòng lệnh quá dài.
  - Nếu khối hàm dán vào không có lỗi thì tên hàm sẽ xuất hiện và ta có thể sử dụng được hàm đó.
  - Phương pháp này rất phù hợp với việc kiểm thử từng hàm.
- Phương pháp 2: Mở tập tin chương trình
  - Trong XLISP, sử dụng menu File-Open/Load để mở tập tin chương trình chứa các hàm đã được viết và lưu trữ bởi Notepad. Chúng ta cũng có thể sử dụng hàm (LOAD <tên tập tin>) để mở tập tin chương trình.
  - Nếu việc mở thành công thì có thể gọi thực hiện bất kỳ hàm nào đã có trong tập tin chương trình.
  - Nếu có một hàm viết sai dấu ngoặc thì việc mở tập tin sẽ thất bại và do đó ta không thể dùng bất kỳ hàm nào trong tập tin đó.
  - Phương pháp này thích hợp với việc nạp nhiều hàm đã được kiểm chứng trong một tập tin chương trình để sử dụng.

Một số thông báo lỗi thường gặp

- Unbound function: Hàm không có.
- Bad function: Hàm sai.
- Too many arguments: Thừa tham số.
- Too few arguments: Thiếu tham số.
- Misplaced close paren: Thừa dấu ngoặc đóng/ Thiếu dấu ngoặc mở.
- EOF reached beore expression end: Thừa dấu ngoặc mở/ Thiếu dấu ngoặc đóng.
- Not a number: Đối số của hàm phải là một số.
- Bad argument type: Kiểu của tham số sai.

CHƯƠNG 9: LẬP TRÌNH LOGIC

9.1 TỔNG QUAN

9.1.1 Mục tiêu

Sau khi học xong chương này, sinh viên cần phải nắm:

- Khái niệm về lập trình logic.
- Các nguyên tắc trong lập trình logic.
- Viết chương trình đơn giản bằng ngôn ngữ Prolog.

9.1.2 Nội dung cốt lõi

- Lập trình logic.
- Căn bản về ngôn ngữ lập trình Prolog.

9.1.3 Kiến thức cơ bản cần thiết

Kiến thức và kĩ năng lập trình căn bản

9.2 GIỚI THIỆU VỀ LẬP TRÌNH LOGIC

Trong lập trình logic, ta có thể sử dụng các vị từ để định nghĩa các khái niệm của tất cả các môn khoa học khác.

Ví dụ định nghĩa một số nguyên tố:

Số nguyên tố N là một số nguyên lớn hơn 1, chỉ chia hết cho 1 và chính nó.

Để xét xem số N có phải là số nguyên tố hay không, người ta thường sử dụng dấu hiệu nhận biết: Số nguyên tố là một số nguyên dương, không chia hết cho mọi số nguyên tố nhỏ hơn nó và 2 là số nguyên tố nhỏ nhất.

Dấu hiệu này có thể mô tả bằng các vị từ như sau:

- 2 là một số nguyên tố.
- N là một số nguyên tố nếu N>0, M là số nguyên tố nào đó, M<N và N không chia hết cho M.

Khi mô tả bài toán dưới dạng logic vị từ, ta có thể yêu cầu hệ thống tìm kiếm các lời giải liên quan đến các khai báo đó. Bài toán cần giải được xem là "mục tiêu" mà hệ thống phải chứng minh trên cơ sở các tri thức đã được khai báo.

Như thế, toàn bộ các ký hiệu của ngôn ngữ lập trình suy về một công thức đặc biệt:

- Phát sinh từ một yêu cầu.
- Nhằm chứng minh một mục tiêu. Để trả lời cho câu hỏi đó hệ thống xem nó như là một "đích" và cố chứng minh "đích" đó bằng cách tạo những suy diễn trên cơ sở các tri thức đã khai báo.

Một ngôn ngữ logic có thể được dùng trong giai đoạn đặc tả yêu cầu của quy trình xây dựng một sản phẩm phần mềm. Hơn thế nữa, logic vị từ cho phép biểu diễn hầu hết các khái niệm và các định lý trong các bộ môn khoa học.

Một trong những ngôn ngữ lập trình logic có hỗ trợ rất nhiều cho lĩnh vực trí tuệ nhân tạo mà ta xét đến ở đây đó là ngôn ngữ Prolog.

9.3 NGÔN NGỮ PROLOG

9.3.1 Giới thiệu

Prolog là một ngôn ngữ cấp cao, có đặc điểm gần với ngôn ngữ tự nhiên, từ những người mới học đến những lập trình viên chuyên nghiệp đều có thể tiếp cận một cách nhanh chóng, viết ra một chương trình ứng dụng hữu ích.

Prolog ra đời vào năm 1973 do C.Camerauer (Đại học Marseilles, Pháp) và nhóm đồng sự phát triển. Từ đó đến nay, qua nhiều lần cải tiến, đặc biệt hãng Borland cho ra đời phần mềm TURBO PROLOG với nhiều ưu điểm, thuận tiện cho người sử dụng. Để giải quyết một số vấn đề, ta nhận thấy sử dụng ngôn ngữ Prolog cho ta chương trình gọn nhẹ hơn nhiều so với các ngôn ngữ khác.

Khác với những ngôn ngữ cấu trúc như Pascal, hay C mà ta đã làm quen, Prolog là một ngôn ngữ mô tả, với một số sự kiện và quy luật suy diễn đã mô tả, Prolog sẽ suy luận cho ta các kết quả.

9.3.2 Các yếu tố cơ bản của Turbo Prolog

Trong một chương trình Prolog, ta cần khai báo các yếu tố sau đây: đối tượng, quan hệ giữa các đối tượng, sự kiện và các luật.

Đối tượng

Gồm có các hằng và biến. Hằng mang giá trị cho sẵn ở đầu chương trình hoặc trong quá trình viết ta đưa vào; Các biến có giá trị thay đổi sẽ được gán giá trị khi chạy chương trình. Tên biến là một ký tự hoa hoặc một chuỗi ký tự, bắt đầu bằng một ký tự hoa.

Có một loại biến đặc biệt gọi là biến tự do, biến này không có tên và người ta dùng ký hiệu _ (dấu gạch dưới) thay cho tên biến.

Quan hệ giữa các đối tượng

Quan hệ giữa các đối tượng được dùng dưới hình thức vị từ.

Ví dụ: Thich(X,Y) là vị từ diễn tả câu “X thích Y” trong ngôn ngữ tự nhiên.

Blue(car) là vị từ diễn tả câu “Car is blue”.

Như vậy các vị từ sẽ bao gồm tên của vị từ và các đối số của nó. Các đối số được đặt trong ngoặc và phân cách nhau bởi dấu phẩy.

Sự kiện và luật

Sự kiện là một vị từ diễn tả một sự thật.

Ví dụ: “2 là một số nguyên tố” là một sự kiện vì nó diễn tả sự thật 2 là một số nguyên tố.

Luật là vị từ diễn tả quy luật suy diễn mà ta công nhận đúng. Luật được trình bày dưới dạng một mệnh đề.

Ví dụ để suy diễn số nguyên N bất kỳ là một số nguyên tố ta viết:

"N là một số nguyên tố nếu N>0, M là số nguyên tố nào đó, M<N và N không chia hết cho M".

9.3.3 Cấu trúc của một chương trình Prolog

Một chương trình Prolog thường gồm có 3 hoặc 4 đoạn cơ bản: clauses, predicates, domains và goal. Phần goal có thể bỏ đi, nếu ta không thiết kế goal trong chương trình, thì khi thực hiện, hệ thống sẽ yêu cầu ta nhập goal vào.

Phần Domains

Đây là phần định nghĩa kiểu mới dựa vào các kiểu đã biết. Các kiểu được định nghĩa ở đây sẽ được sử dụng cho các đối số trong các vị từ. Nếu các vị từ sử dụng đối số có kiểu cơ bản thì có thể không cần phải định nghĩa lại các kiểu đó. Tuy nhiên để cho chương trình sáng sủa, người ta sẽ định nghĩa lại cả các kiểu cơ bản.

Cú pháp: <**danh sách kiểu mới**> = <**kiểu đã biết**> hoặc <**danh sách kiểu mới**> = <**danh sách kiểu đã biết**>

Trong đó các kiểu mới phân cách nhau bởi dấu phẩy, còn các kiểu đã biết phân cách nhau bởi dấu chấm phẩy.

Ví dụ:

```
Domains
        ten, tac_gia, nha_xb, dia_chi = string
        nam, thang, so_luong = integer
        dien_tich = real
        nam_xb = nxb(thang, nam)
        do_vat = sach(tac_gia, ten, nha_xb, nam_xb);  xe(ten, so_luong); nha(dia_chi,
        dien_tich)
```

Trong ví dụ trên, ta đã định nghĩa các kiểu mới, trong đó các kiểu mới *ten, tac_gia, nha_xb, dia_chi* dựa vào cùng một kiểu đã biết là *string*; các kiểu mới *nam, thang, so_luong* dựa vào cùng một kiểu đã biết là *integer*; kiểu mới *dien_tich* dựa vào kiểu đã biết là *real*; kiểu mới *năm_xb* dựa vào kiểu nxb được xây dựng từ các kiểu đã biết là *thang, nam*; còn kiểu *do_vat* lại dựa vào các kiểu *sach, xe, nha* mà các kiểu này lại dựa vào các kiểu đã biết.

Phần Predicates

Đây là phần bắt buộc phải có. Trong phần này chúng ta cần phải khai báo đầy đủ các vị từ sử dụng trong phần Clauses, ngoại trừ các vị từ mà Turbo Prolog đã xây dựng sẵn.

Cú pháp: <**Tên vị từ**> (<**danh sách các kiểu**>)

Các kiểu là các kiểu cơ bản hoặc là các kiểu đã được định nghĩa trong phần domains và được viết phân cách nhau bơi dấu phẩy.

Ví dụ:

```
Predicates
        so_huu (ten, do_vat)
```

so_nguyen_to(integer)

Trong ví dụ trên ta khai báo hai vị từ. Trong đó vị từ *so_huu (ten, do_vat)* để chỉ một người có tên là *ten* sẽ sở hữu môt *do_vat* nào đó. Còn vị từ *so_nguyen_to(integer)* để xét xem một số *integer* nào đó có phải là số nguyên tố hay không.

Phần Clauses

Đây là phần bắt buộc phải có dùng để mô tả các sự kiện và các luật, sử dụng các vị từ đã khai báo trong phần predicates.

Cú pháp:

<**Tên vị từ**>(<**danh sách các tham số**>) <**kí hiệu**>
<**Tên vị từ 1**>(<**danh sách các tham số 1**>) <**kí hiệu**>
… … …
<**Tên vị từ N**>(<**danh sách các tham số N**>) <**kí hiệu**>

Trong đó: *Tên vị từ* phải là các *tên vị từ* đã được khai báo trong phần predicates. Các *tham số* có thể là các hằng hoặc biến có kiểu tương thích với các kiểu tương ứng đã được khai báo trong các vị từ ở trong phần predicates; các tham số được viết cách nhau bởi dấu phẩy. Các *kí hiệu* bao gồm:

:- (điều kiện nếu).

, (điều kiện và).

; (điều kiện hoặc).

. (kết thúc vị từ)

Ví dụ:

```
Clauses
        so_nguyen_to(2):-   !.
        so_nguyen_to(N):-   N>0,
                            so_nguyen_to(M),
                            M<N,
                            N MOD M <>0.
        so_huu("Nguyen Van A", sach("Do Xuan Loi", "Cau truc DL", "Khoa hoc Ky thuat", nxb(8,1985))).
```

Chú ý: Nếu trong các tham số của một vị từ có biến thì biến này phải xuất hiện ít nhất 2 lần trong vị từ đó hoặc trong các vị từ dùng để suy diễn ra vị từ đó. Nếu chỉ xuất hiện một lần thì bắt buộc phải dùng biến tự do.

Ví dụ: Để diễn tả sự kiện: Tổ hợp chập 0 của N (N bất kỳ) bằng 1, ta không thể viết *Tohop(N,0,1)* vì biến N chỉ xuất hiện đúng một lần trong vị từ này, do đó ta phải viết *Tohop(_,0,1)* .

Phần Goal

Bao gồm các mục tiêu mà ta yêu cầu Turbo Prolog xác định và tìm kết quả. Đây là phần không bắt buộc phải có. Nếu ta viết sẵn trong chương trình thì đó gọi là goal nội; Nếu không, khi chạy chương trình Turbo Prolog sẽ yêu cầu ta nhập goal vào, lúc này gọi là goal ngoại.

Cú pháp phần goal giống như cú pháp phần clauses. Tức là ta đưa vào một hoặc một số các vị từ.

Nếu tất cả các tham số của vị từ là hằng thì kết quả nhận được là Yes (đúng) hoặc No (sai). Nếu trong các tham số của vị từ có biến thì kết quả trả về sẽ là các giá trị của biến.

Ngoài các phần chủ yếu nói trên, ta có thể đưa vào các phần liên quan đến khai báo hằng, các tập tin liên quan hoặc chỉ thị dịch.

Ví dụ:

Constants

Pi = 3.141592653

Một số ví dụ về chương trình prolog

Ví dụ 1: Xét xem một số N có phải là số nguyên tố hay không.

```
domains
        so_nguyen = integer
predicates
        so_nguyen_to(so_nguyen)
Clauses
        so_nguyen_to(2):-   !.
        so_nguyen_to(N):-   N>0,
                            so_nguyen_to(M),
                            M<N,
                            N MOD M <>0.
goal
        so_nguyen_to(13).
```

Ví dụ 2: Giả sử ta có bảng số liệu như sau:

Tên người	giới tính	Đặc điểm	Tiêu chuẩn kết bạn
lan	nữ	đẹp, khoẻ, tốt,	khoẻ, thông minh, đẹp
hồng	nữ	đẹp, thông minh, giàu	khoẻ, thông minh, giàu
thuỷ	nữ	tốt, khoẻ, giàu	đẹp, khoẻ, thông minh
anh	nam	khoẻ, giàu, thông minh	đẹp, thông minh, tốt
bình	nam	đẹp, khoẻ, thông minh	đẹp, khoẻ
hùng	nam	giàu, thông minh, khoẻ	tốt, thông minh, khoẻ

Tiêu chuẩn kết bạn là hai người khác phái, người này hội đủ các tiêu chuẩn của người kia và ngược lại. Hãy viết chương trình để tìm ra các cặp có thể kết bạn với nhau.

```
domains
  ten, g_tinh = symbol
predicates
  gioi_tinh(ten, g_tinh)
  dep(ten)
  tot(ten)
  giau(ten)
  thong_minh(ten)
```

```
  khoe(ten)
  thich(ten,ten)
  ket_ban(ten,ten)
clauses
  gioi_tinh(lan,nu).
  gioi_tinh(hong,nu).
  gioi_tinh(thuy,nu).
  gioi_tinh(anh,nam).
  gioi_tinh(binh,nam).
  gioi_tinh(hung,nam).

  dep(lan).
  dep(hong).
  dep(binh).

  khoe(thuy).
  khoe(lan).
  khoe(binh).
  khoe(anh).
  khoe(hung).

  tot(lan).
  tot(thuy).

  thong_minh(hong).
  thong_minh(anh).
  thong_minh(hung).
  thong_minh(binh).

  giau(hong).
  giau(thuy).
  giau(hung).

  thich(lan,X):-khoe(X), dep(X), thong_minh(X).
  thich(hong,X):-khoe(X), thong_minh(X), giau(X).
  thich(thuy,X):-khoe(X), dep(X), thong_minh(X).
  thich(ann,X):-dep(X), tot(X), thong_minh(X).
  thich(binh,X):-dep(X), khoe(X).
  thich(hung,X):-khoe(X), tot(X), thong_minh(X).

  ket_ban(X,Y):- gioi_tinh(X,M),
                 gioi_tinh(Y,N),
                 M<>N,
                 thich(X,Y),
                 thich(Y,X).
```

9.3.4 Các nguyên tắc của ngôn ngữ Prolog

Việc giải quyết vấn đề trong ngôn ngữ Prolog chủ yếu dựa vào hai nguyên tắc sau: Đồng nhất, quay lui.

Đồng nhất

Một quan hệ có thể đồng nhất với một quan hệ nào đó cùng tên, cùng số lượng tham số, các đại lượng con cũng đồng nhất theo từng cặp.

Một hằng có thể đồng nhất với một hằng.

Một biến có thể đồng nhất với một hằng nào đó và có thể nhận luôn giá trị hằng đó.

Chẳng hạn trong ví dụ 2 nói trên nếu ta sử dụng goal dep(lan) thì có kết quả là Yes. Nếu ta dùng goal dep(X) thì sẽ có 3 kết quả: X=lan, X=hong và X=binh.

Khi ta dùng goal dep(lan) thì dep(lan) sẽ đồng nhất với sự kiện dep(lan) trong phần clauses và do hai vị từ đồng nhất với nhau và hai đối số hằng đồng nhất nhau nên kết quả là Yes.

Khi dùng goal dep(X) thì dep sẽ được đồng nhất với dep và biến X đồng nhất với hằng lan, do đó ta có kết quả X=lan. Tương tự X=hong và X=binh.

Quay lui

Giả sử hệ thống đang chứng minh goal g, trong đó g được mô tả như sau:

$g :- g_1, g_2, \ldots, g_{j-1}, g_j, \ldots, g_n.$

Khi các g_i kiểm chứng từ trái sang phải, đến g_j là sai thì hệ thống sẽ quay lui lại g_{j-1} để tìm lời giải khác.

Chẳng hạn trong ví dụ 2 nói trên, khi ta yêu cầu Goal: thich(lan,X), ta được X=binh.
Vị từ thich(lan,X) sẽ được đồng nhất với thich(lan,X) trong phần clauses, theo đó hệ thống phải chứng minh thich(lan,X):-khoe(X), dep(X), thong_minh(X).

- Trước hết đồng nhất khoe(X) với khoe(thuy) => X=thuy.
- Do dep(thuy) sai nên quay lui đồng nhất khoe(X) với khoe(lan) => X=lan.
- Do dep(lan) đúng nên tiếp tục kiểm tra thong_minh(lan).
- Do thong_minh(lan) sai nên quay lui để đồng nhất khoe(X) với khoe(binh) để có X=binh, sau đó kiểm tra thấy dep(binh) và thong_minh(binh) đều đúng nên X=binh là một nghiệm.

9.3.5 Bộ ký tự, từ khoá

Prolog dùng bộ ký tự sau: các chữ cái và chữ số (A – Z, a – z, 0 – 9); các toán tử (+, -, *, /, <, =, >) và các ký hiệu đặc biệt.

Một số từ khoá:

a. **Trace**: Khi có từ khoá này ở đầu chương trình, thì chương trình được thực hiện từng bước để theo dõi; dùng phím F10 để tiếp tục.

b. **Fail**: Khi ta dùng goal nội, chương trình chỉ cho ta một kết quả (mặc dù có thể còn những kết quả khác), để nhận về tất cả các kết quả khi chạy goal nội, ta dùng toán tử Fail.

c. **!** hay còn gọi là nhát cắt, goal ngoại luôn cho ta mọi kết quả, muốn nhận chỉ một kết quả từ goal ngoại, ta dùng ký hiệu !.

9.3.6 Các kiểu dữ liệu

Trong prolog có kiểu dữ liệu chuẩn và kiểu do người lập trình định nghĩa.

Kiểu dữ liệu chuẩn

Là kiểu dữ liệu do prolog định nghĩa sẵn. Prolog cung cấp các kiểu dữ liệu chuẩn là: char, integer, real, string và symbol.

a. **Char**: Là kiểu ký tự. Hằng ký tự phải nằm giữa hai dấu nháy đơn.

Ví dụ: ‘a’, ‘#’.

b. **Integer:** Là kiểu số nguyên, tập giá trị bao gồm các số nguyên từ -32768 đến 32767.

c. **Real:** Là kiểu số thực, tập giá trị bao gồm các số thực thuộc hai đoạn: đoạn các số âm từ -10307 đến -10-307 và đoạn số dương từ 10-307 đến 10307.

d. **String:** Là kiểu chuỗi ký tự. Hằng chuỗi ký tự phải nằm giữa hai dấu nháy kép.

Ví dụ: “Turbo prolog 2.0”

e. **Symbol:** Là một kiểu sơ cấp, có hình thức giống chuỗi ký tự. Hằng symbol có hai dạng: Dãy các chữ, số và dấu gạch dưới viết liên tiếp, ký tự đầu phải viết thường (chẳng hạn: telephone_number); Dãy các ký tự ở giữa một cặp hai nháy kép (giống như chỗi ký tự)

f. **Một số phép toán của các kiểu**

Phép toán số học

Phép toán	Ý nghĩa	Kiểu của đối số	Kiểu kết quả
+	Cộng hai số	Integer, real	giống kiểu đối số
-	Trừ hai số	Integer, real	giống kiểu đối số
*	Nhân hai số	Integer, real	giống kiểu đối số
/	Chia hai số	Integer, real	giống kiểu đối số
Mod	Phép chia lấy phần dư	Integer	Integer
Div	Phép chia lấy phần nguyên	Integer	Integer

Phép toán quan hệ

Phép toán	Ý nghĩa	Kiểu của đối số	Kết quả
<	Nhỏ hơn	Char, integer, real, string	Yes hoặc No
<=	Nhỏ hơn hay bằng	Char, integer, real, string	Yes hoặc No
=	Bằng	Char, integer, real, string	Yes hoặc No

>	Lớn hơn	Char, integer, real, string	Yes hoặc No
>=	Lớn hơn hay bằng	Char, integer, real, string	Yes hoặc No
<> hay ><	Khác	Char, integer, real, string	Yes hoặc No

Các vị từ như các hàm toán học

Vị từ	Ý nghĩa	Kiểu của đối số	Kiểu kết quả	Ví dụ
Sin(X)	Tính sin của X	real	real	
Tan(X)	Tính tang của X	real	real	
Arctan(X)	Tính arctang của X	real	real	
Exp(X)	Tính e^X	real	real	
Ln(X)	Tính logarit cơ số e của X	real	real	
Log(X)	Tính Logarit cơ số 10 của X	real	real	
SQRT(X)	Tính căn bậc hai của X	real	real	
ROUND(X)	Cho ta số nguyên là số X được làm tròn, dấu là dấu của X	real	integer	round(2.3)=2 round(2.5)=3 round(-2.5)=-2 round(-2.6)=-3
TRUNC(X)	Cho phần nguyên của số X, dấu là dấu của X	real	integer	trunc(2.5)=2 trunc(-2.6)=-2
ABS(X)	Cho ta trị tuyệt đối của X	real	real	
Random(X)	Cho ta số thực X nằm trong khoảng [0, 1)	real	real	
Random(Y, X)	Cho ta số nguyên X nằm trong khoảng [0, Y)	real	integer	

Toán tử NOT(X) : Nếu X là Yes thì NOT(X) là No và ngược lại.

Các kiểu dữ liệu do người lập trình định nghĩa

a. **Kiểu mẩu tin:**

Cú pháp: **<tên kiểu mẩu tin> = tên mẩu tin (danh sách các kiểu phần tử)**

Ví dụ:

```
Domains
        ten, tac_gia, nha_xb, dia_chi = string
        nam, thang, so_luong = integer
        dien_tich = real
        nam_xb = nxb(thang, nam)
        do_vat = sach(tac_gia, ten, nha_xb, nam_xb);  xe(ten, so_luong); nha(dia_chi,
        dien_tich)
predicates
        so_huu(ten,do_vat)
clauses
        so_huu("Nguyen Van A", sach("Do Xuan Loi", "Cau truc DL", "Khoa hoc Ky
thuat", nxb(8,1985))).
```

so_huu("Le thi B", xe("Dream II", 2)).
so_huu("Nguyen Huu C", nha("3/1 Ly Tu Trong, tp Can Tho", 100.5))

b. Kiểu danh sách

Cú pháp: **<tên kiểu danh sách> = <tên kiểu phần tử>***

Ví dụ:

Domains

intlist = integer*

Một danh sách là một dãy các phần tử phân cách nhau bởi dấu phẩy và đặt trong cặp dấu ngoặc vuông.

Ví dụ:

[] % Danh sách rỗng

[1,2,3] % Danh sách gồm ba số nguyên 1, 2 và 3.

Cấu trúc của danh sách bao gồm hai phần: Phần đầu là phần tử đầu tiên của danh sách và phần đuôi là một danh sách của các phần tử còn lại.

Danh sách được viết theo dạng [X|Y] thì X là phần tử đầu và Y là danh sách đuôi. Chẳng hạn trong danh sách [1,2,3] thì đầu là số nguyên 1 và đuôi là danh sách [2,3].

Trong danh sách cũng có thể dùng biến tự do, chẳng hạn ta có thể viết [_|Y] để chỉ một danh sách có đầu là một phần tử nào đó và có đuôi là danh sách Y.

9.3.7 Các hàm xuất nhập chuẩn

Xuất ra màn hình

a. **Write(Arg1, Arg2, … ,Argn)** in ra màn hình giá trị của các đối số.

b. **Writef(Formatstring, Arg1, Arg2, … ,Argn)** in ra màn hình giá trị của các đối số theo định dạng được chỉ định trong Formastring.

Trong đó Formastring là một chuỗi có thể là:

- "%d": In số thập phân bình thường; đối số phải là char hoặc integer.
- "%c": Đối số là một số integer, in ký tự có mã Ascci là đối số đó, chẳng hạn writef("%c",65) được A.
- "%e": In số thực dưới dạng lũy thừa của 10.
- "%x": In số Hexa; đối số phải là char hoặc integer.
- "%s": In một chuỗi hoặc một symbol.

Nhập vào từ bàn phím

a. **Readln(X)**: Nhập một chuỗi ký tự vào biến X.

b. **ReadInt(X)**: Nhập một số nguyên vào biến X.

c. **ReadReal(X)**: Nhập một số thực vào biến X.

d. **ReadChar(X)**: Nhập vào một ký tự vào biến X.

9.3.8 Kỹ thuật đệ quy

Đệ quy là kỹ thuật lập trình được sử dụng trong nhiều ngôn ngữ. Trong Turbo Prolog ta sử dụng đệ quy khi một vị từ được định nghĩa nhờ vào chính vị từ đó.

Như đã nói trong chương lập trình hàm, trong chương trình đệ quy phải có ít nhất một trường hợp dừng và lời gọi đệ quy phải chứa yếu tố dẫn đến trường hợp dừng. Trong Prolog, trường hợp dừng được thể hiện bằng một sự kiện, yếu tố dẫn đến trường hợp dừng thể hiện bằng một biến, liên hệ với biến ban đầu bởi một công thức.

Ví dụ 1: Tính n giai thừa.

```
Predicates
        Facto (integer, integer)
Clauses
        Facto(0,1):- !.
        Facto(N, FactN) :- N > 0, M = N – 1, facto(M, factM), factN = N*factM.
```

Ở ví dụ trên ta đã định nghĩa một vị từ dùng để tính giá trị giai thừa của một số tự nhiên, đối số thứ nhất là số cần tính giai thừa và đối số thứ hai dùng để nhận giá trị trả về.

Trường hợp dừng ở đây được xác đinh bởi sự kiện 0 giai thừa là 1.

Để tính N! ta tính M! với M= N-1. Yếu tố dẫn đến trường hợp dừng là biến M có giá trị bằng N-1.

Ví dụ 2: Xác định một phần tử trong danh sách các symbol

```
domains
        symbol_list = symbol*
predicates
        element1(integer,symbol_list,symbol)
        element  (integer,symbol_list,symbol)
clauses
  % element1 không suy diễn ngược được
        element1(1,[X|_],X).
        element1(N,[_|L],Y):-      M=N-1,
                                   element1(M,L,Y).
  % element có thể suy diễn ngược
        element(1,[X|_],X).
        element(N,[_|L],Y):-       element(M,L,Y),
                                   N=M+1.
```

Sự suy diễn thuận chiều là cho danh sách và vị trí, tìm được phần tử tại vị trí đó, chẳng hạn, nếu ta đưa vào goal element(2,[a,b,c,d],X) ta được X=b.

Sự suy diễn ngược ở đây là cho danh sách và phần tử, tìm được vị trí của phần tử đó, chẳng hạn, nếu ta đưa vào goal element(N,[a,b,c,d], b) ta được N=2.

Ví dụ 3: Sắp xếp một danh sách các số nguyên

```
domains
        list=integer*
predicates
```

```
        insert(integer,list,list)
        sort(list,list)
clauses
        insert(E,[],[E]).
        insert(E,[A|B],[E,A|B]):-    E<=A.
        insert(E,[A|B],[A|C]):-      E>A,insert(E,B,C).

        sort([],[]).
        sort([X|R1],L):-      sort(R1,R),
                              insert(X,R,L).
```