

CHƯƠNG VII

MÔI TRƯỜNG THỜI GIAN THỰC HIỆN

Nội dung chính:

Trước khi xem xét vấn đề sinh mã được trình bày ở các chương sau, chương này trình bày một số vấn đề liên quan đến việc *gọi thực hiện chương trình con*, các *chiến lược cấp phát bộ nhớ* và *quản lý bảng ký hiệu*. Cùng một tên trong chương trình nguồn có thể biểu thị cho nhiều *đối tượng dữ liệu* trong chương trình đích. Sự biểu diễn của các đối tượng dữ liệu tại thời gian thực thi được xác định bởi kiểu của nó. Sự cấp phát và thu hồi các đối tượng dữ liệu được quản lý bởi một tập các *chương trình con ở dạng mã đích*. Việc thiết kế các chương trình con này được xác định bởi ngữ nghĩa của chương trình nguồn. Mỗi sự thực thi của một chương trình con được gọi là một *mẫu tin kích hoạt*. Nếu một chương trình con đệ quy, một số mẫu tin kích hoạt có thể tồn tại cùng một thời điểm. Mỗi ngôn ngữ lập trình đều có *quy tắc tầm vực* để xác định việc xử lý khi tham khảo đến các tên không cục bộ. Tùy vào ngôn ngữ, nó cho phép một *chương trình chứa các chương trình con lồng nhau hoặc không lồng nhau*; Cho phép *gọi đệ quy hoặc không đệ quy*; Cho phép *truyền tham số* bằng giá trị hay tham chiếu ... Vì thế, khi thiết kế một chương trình ở dạng mã đích ta cần chú ý đến các yếu tố này.

Mục tiêu cần đạt:

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

- Cách gọi và thực thi một chương trình.
- Cách tổ chức bộ nhớ và các chiến lược cấp phát – thu hồi bộ nhớ.

Kiến thức cơ bản:

Sinh viên phải biết một số ngôn ngữ lập trình cấp cao như Pascal, C++, Java, v.v hoặc đã được học môn ngôn ngữ lập trình (phần đề cập đến các chương trình con).

Tài liệu tham khảo:

[1] **Compilers : Principles, Technique and Tools** - Alfred V.Aho, Jeffrey D.Ullman - Addison - Wesley Publishing Company, 1986.

[2] **Modern Compiler Implementation in C** - Andrew W. Appel - Cambridge University Press, 1997.

I. CHƯƠNG TRÌNH CON

1. Định nghĩa chương trình con

Định nghĩa chương trình con là một sự khai báo nó. Dạng đơn giản nhất là sự kết hợp giữa tên chương trình con và thân của nó.

Ví dụ 7.1: Chương trình Pascal đọc và sắp xếp các số nguyên

```

(1)  program sort(input, output)
(2)    var a: array[0..10] of integer;
(3)  procedure readarray;
(4)    var i: integer;
(5)    begin
(6)      for i=1 to 9 do read(a[i]);
(7)    end;
(8)  function partition(y,z:integer): integer;
(9)    var i,j,x,v: integer;
(10)   begin...
(11)  end;
(12) procedure quicksort(m,n:integer);
(13)   var i: integer;
(14)   begin;
(15)     if (n>m) then begin
(16)       i:= partition(m,n);
(17)       quicksort(m,i-1);
(18)       quicksort(i+1,n);
(19)     end;
(20)   end;
(21)   begin
(22)     a[0]:= -9999, a[10]:= 9999;
(23)     readarray;
(24)     quicksort(1,9);
(25)   end.

```

Hình 7.1- Chương trình Pascal đọc và sắp xếp các số nguyên

Chương trình trên chứa các định nghĩa chương trình con

- Chương trình con readarray từ dòng 3 - 7, thân của nó từ 5 - 7
- Chương trình con partition từ dòng 8 - 11, thân của nó từ 10 - 11.
- Chương trình con quicksort từ dòng 12 - 20, thân của nó từ 14 - 20.

Chương trình chính cũng được xem như là một chương trình con có thân từ dòng 21 - 25

Khi tên chương trình con xuất hiện trong phần thân của một chương trình con ta nói chương trình con được gọi tại điểm đó.

2. Cây hoạt động

Trong quá trình thực hiện chương trình thì:

1. Dòng điều khiển là tuần tự: tức là việc thực hiện chương trình bao gồm một chuỗi các bước. Tại mỗi bước đều có một sự điều khiển xác định.

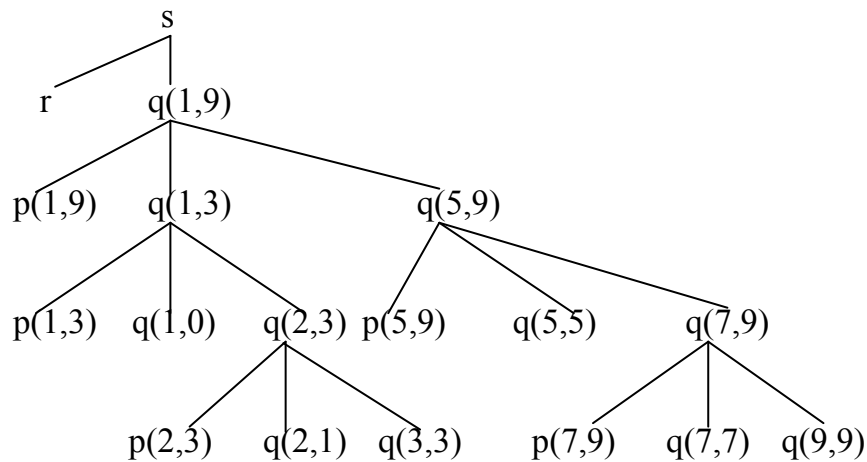
2. Việc thực hiện chương trình con bắt đầu tại điểm bắt đầu của thân chương trình con và trả điều khiển về cho chương trình gọi tại điểm nằm sau lời gọi khi việc thực hiện chương trình con kết thúc.

- Thời gian tồn tại của một chương trình con p là một chuỗi các bước giữa bước đầu tiên và bước cuối cùng trong sự thực hiện thân chương trình con bao gồm cả thời gian thực hiện các chương trình con được gọi bởi p .
- Nếu a và b là hai sự hoạt động của hai chương trình con tương ứng thì thời gian tồn tại của chúng tách biệt nhau hoặc lồng nhau.
- Một chương trình con là đệ quy nếu một hoạt động mới có thể bắt đầu trước khi một hoạt động trước đó của chương trình con đó kết thúc.
- Để đặc tả cách thức điều khiển vào ra mỗi hoạt động của chương trình con ta dùng cấu trúc cây gọi là cây hoạt động.
 1. Mỗi nút biểu diễn cho một hoạt động của một chương trình con.
 2. Nút gốc biểu diễn cho hoạt động của chương trình chính.
 3. Nút a là cha của b nếu và chỉ nếu dòng điều khiển sự hoạt động đó từ a sang b
 4. Nút a ở bên trái của nút b nếu thời gian tồn tại của a xuất hiện trước thời gian tồn tại của b .

Ví dụ 7.2: Xét chương trình sort nói trên

- Bắt đầu thực hiện chương trình.
- Vào readarray.
- Ra khỏi readarray.
- Vào quicksort(1,9).
- Vào partition(1,9)
- Ra khỏi partition(1,9) // giả sử trả về 4
- Vào quicksort(1,3)
-
- Ra khỏi quicksort(1,3).
- Vào quicksort(5,9);
-
- Ra khỏi quicksort(5,9).
- Sự thực hiện kết thúc.

Hình 7.2 - Xuất các mẫu tin hoạt động đề nghị của chương trình trong hình 7.1
Ta có cây hoạt động tương ứng.



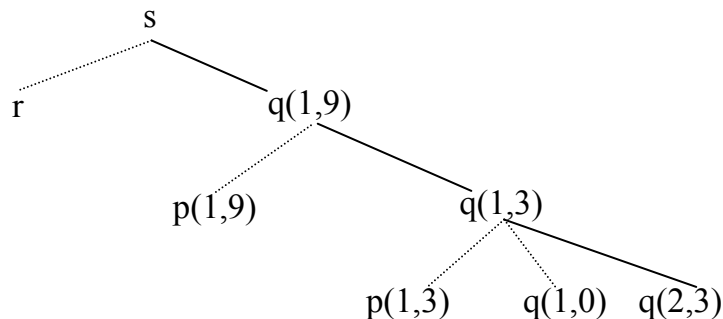
Hình 7.3- Cây hoạt động tương ứng với phần xuất trong hình 7.2

3. Ngăn xếp điều khiển

Dòng điều khiển một chương trình tương ứng với phép duyệt theo chiều sâu của cây hoạt động. Bắt đầu từ nút gốc, thăm một nút trước các con của nó và thăm các con một cách đệ quy tại mỗi nút từ trái sang phải.

Chúng ta có thể dùng một Stack, gọi là Stack điều khiển, để lưu trữ sự hoạt động của chương trình con. Khi sự hoạt động của một chương trình con bắt đầu thì đẩy nút tương ứng với sự hoạt động đó lên đỉnh Stack. Khi sự hoạt động kết thúc thì pop nút đó ra khỏi Stack. Nội dung của Stack thể hiện đường dẫn đến nút gốc của cây hoạt động. Khi nút n nằm trên đỉnh Stack thì Stack chứa các nút nằm trên đường từ n đến gốc.

Ví dụ 7.3: Hình sau trình bày nội dung của Stack đang lưu trữ đường đi từ nút $q(2, 3)$ đến nút gốc. Các cạnh nét đứt thể hiện một nút đã pop ra khỏi Stack.



Hình 7.4 - Stack điều khiển chứa các nút dẫn tới nút gốc

Tại thời điểm này thì đường dẫn tới gốc là: s q(1, 9) q(1, 3) q(2, 3) (q(2, 3) nằm trên đỉnh Stack)

4. Tầm vực của sự khai báo

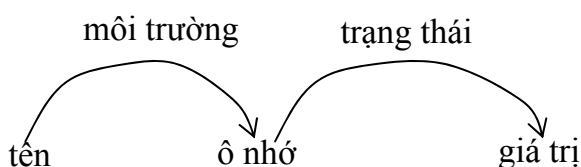
Đoạn chương trình chịu ảnh hưởng của một sự khai báo gọi là tầm vực của khai báo đó.

Trong một chương trình có thể có nhiều sự khai báo trùng tên ví dụ biến i trong chương trình `sort`. Các khai báo này độc lập với nhau và chịu sự chi phối bởi quy tắc tầm của ngôn ngữ.

Sự xuất hiện của một tên trong một chương trình con được gọi là cục bộ (local) trong chương trình con ấy nếu tầm vực của sự khai báo nằm trong chương trình con, ngược lại được gọi là không cục bộ (nonlocal).

5. Liên kết tên

Trong ngôn ngữ của ngôn ngữ lập trình, thuật ngữ môi trường (environment) để chỉ một ánh xạ từ một tên đến một vị trí ô nhớ và thuật ngữ trạng thái (state) để chỉ một ánh xạ từ vị trí ô nhớ tới giá trị lưu trữ trong đó



Hình 7.5 - Hai ánh xạ từ tên tới giá trị

Môi trường khác trạng thái: một lệnh gán làm thay đổi trạng thái nhưng không thay đổi môi trường.

Khi một môi trường kết hợp vị trí ô nhớ s với một tên x ta nói rằng x được liên kết tới s . Sự kết hợp đó được gọi là mối liên kết của x .

Liên kết là một bản sao động (dynamic counterpart) của sự khai báo.

Chúng ta có sự tương ứng giữa các ký hiệu động và tĩnh:

Ký hiệu tĩnh	Bản sao động
Định nghĩa chương trình con	Sự hoạt động của chương trình con
Sự khai báo tên	Liên kết của tên
Tầm vực của sự khai báo	Thời gian tồn tại của liên kết

Hình 7.6 - Sự tương ứng giữa ký hiệu động và tĩnh

6. Các vấn đề cần quan tâm khi làm chương trình dịch

Các vấn đề cần đặt ra khi tổ chức lưu trữ và liên kết tên:

1. Chương trình con có thể đệ quy không?
2. Điều gì xảy ra cho giá trị của các tên cục bộ khi trả điều khiển từ hoạt động của một chương trình con.

3. Một chương trình con có thể tham khảo tới một tên cục bộ không?
4. Các tham số được truyền như thế nào khi gọi chương trình con.
5. Một chương trình con có thể được truyền như một tham số?
6. Một chương trình con có thể được trả về như một kết quả?
7. Bộ nhớ có được cấp phát động không?
8. Bộ nhớ có phải giải phóng một cách tường minh?

II. TỔ CHỨC BỘ NHỚ

Tổ chức bộ nhớ trong thời gian thực hiện ở đây có thể sử dụng cho các ngôn ngữ Fortran, Pascal và C.

1. Phân chia bộ nhớ trong thời gian thực hiện

Bộ nhớ có thể chia ra để lưu trữ các phần:

1. Mã đích.
2. Đối tượng dữ liệu.
3. Bản sao của Stack điều khiển để lưu trữ hoạt động của chương trình con.

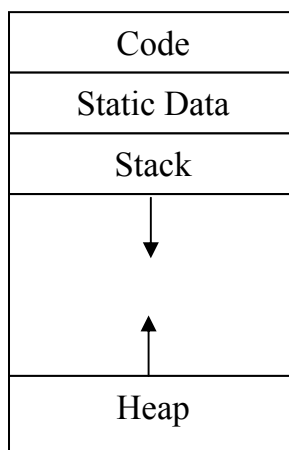
Trong đó kích thước của mã đích được xác định tại thời gian dịch cho nên nó được cấp phát tĩnh tại vùng thấp của bộ nhớ. Tương tự kích thước của một số đối tượng dữ liệu cũng có thể biết tại thời gian dịch cho nên nó cũng được cấp phát tĩnh.

Cài đặt các ngôn ngữ như Pascal, C dùng sự mở rộng của Stack điều khiển để quản lý sự hoạt động của chương trình con.

Khi có một lời gọi chương trình con, sự thể hiện của một hoạt động bị ngắt và thông tin về tình trạng của máy, chẳng hạn như giá trị bộ đếm chương trình (program counter) và thanh ghi được lưu vào trong Stack. Khi điều khiển trả về từ lời gọi, hoạt động này được tiếp tục sau khi khôi phục lại giá trị của thanh ghi và đặt bộ đếm chương trình vào ngay sau lời gọi.

Đối tượng dữ liệu mà thời gian tồn tại của nó được chứa trong một hoạt động được lưu trong Stack.

Một vùng khác của bộ nhớ được gọi là Heap lưu trữ tất cả các thông tin khác.



Hình 7.7 - Phân chia bộ nhớ trong thời gian thực hiện cho mã đích và các vùng dữ liệu khác

2. Mẫu tin hoạt động

Thông tin cần thiết để thực hiện một chương trình con được quản lý bằng cách dùng một mẫu tin hoạt động bao gồm một số trường như sau :

Giá trị trả về
Các tham số thực tế
Liên kết điều khiển
Liên kết truy nhập
Trạng thái máy
Dữ liệu cục bộ
Giá trị tạm thời

Hình 7.8 - Mẫu tin hoạt động tổng quát

Ý nghĩa các trường như sau:

1. Giá trị tạm thời: được lưu giữ trong quá trình đánh giá biểu thức.
2. Dữ liệu cục bộ: Lưu trữ dữ liệu cục bộ trong khi thực hiện chương trình con.
3. Trạng thái máy: lưu giữ thông tin về trạng thái của máy trước khi một chương trình con được gọi. Thông tin máy bao gồm bộ đếm chương trình và thanh ghi lệnh mà nó sẽ phục hồi khi điều khiển trả về từ chương trình con
4. Liên kết truy nhập: tham khảo tới dữ liệu không cục bộ được lưu trong các mẫu tin hoạt động khác.
5. Liên kết điều khiển: trở tới mẫu tin hoạt động của chương trình gọi.
6. Các tham số thực tế: được sử dụng bởi các chương trình gọi để cho chương trình được gọi. Thông thường các tham số được lưu trong thanh ghi chứ không phải trong mẫu tin hoạt động.
7. Giá trị trả về: được dùng bởi chương trình được gọi để trả về cho chương trình gọi một giá trị. Trong thực tế giá trị này thường được trả về trong thanh ghi.

III. CHIẾN LƯỢC CẤP PHÁT BỘ NHỚ

Đối với các vùng nhớ khác nhau trong tổ chức bộ nhớ, ta có các chiến lược cấp phát khác nhau :

1. Cấp phát tĩnh cho tất cả các đối tượng dữ liệu tại thời gian dịch.

2. Cấp phát sử dụng Stack cho bộ nhớ trong thời gian thực hiện.
3. Đối với vùng dữ liệu Heap sử dụng cấp phát và thu hồi dạng Heap.

1. Cấp phát tĩnh


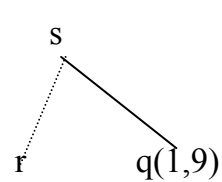
Trong cấp phát tĩnh, tên được liên kết với vùng nhớ lúc chương trình được dịch. Vì mỗi liên kết không thay đổi tại thời gian chạy nên mọi lần một chương trình con được kích hoạt, tên của nó được liên kết với cùng một vùng nhớ. Tính chất này cho phép giá trị của các tên cục bộ được giữ lại thông qua hoạt động của các chương trình con. Từ kiểu của tên, trình biên dịch xác định kích thước bộ nhớ của nó. Do đó trình biên dịch xác định được vị trí của mẫu tin kích hoạt giữa đoạn mã chương trình và các mẫu tin kích hoạt khác. Trong thời gian biên dịch, chúng ta có thể điền vào đoạn của các địa chỉ mà mã lệnh có thể tìm đến để truy xuất dữ liệu. Tương tự địa chỉ các vùng lưu trữ thông tin khi chương trình con được gọi đều được xác định tại thời gian dịch. Tuy nhiên cấp phát tĩnh cũng có một số hạn chế sau:

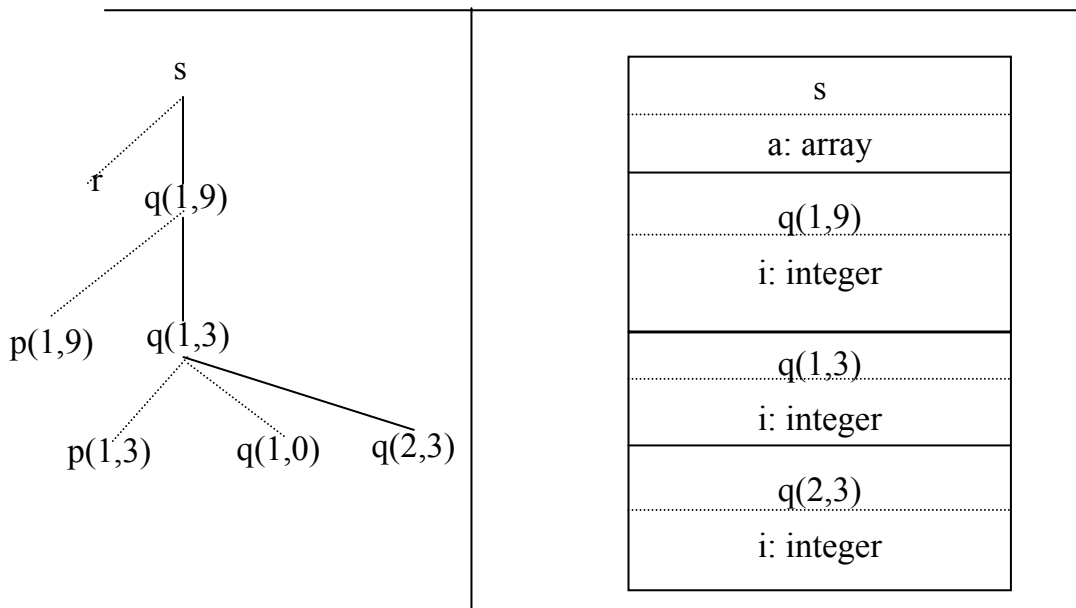
1. Kích thước và vị trí của đối tượng dữ liệu trong bộ nhớ phải được xác định tại thời gian dịch.
2. Không cho phép gọi đệ quy vì tất cả các kích hoạt của một chương trình con đều dùng chung một liên kết đối với tên cục bộ.
3. Cấu trúc dữ liệu không thể được cấp phát động vì không có cơ chế để cấp phát tại thời gian thực hiện.

2. Cấp phát ô nhớ sử dụng Stack

Bộ nhớ được tổ chức như là một Stack. Các mẫu tin kích hoạt được push vào Stack khi hoạt động bắt đầu và pop ra khỏi Stack khi hoạt động kết thúc.

Ví dụ 7.4: Chúng ta sẽ minh họa việc cấp phát và loại bỏ mẫu tin kích hoạt tương ứng với cây hoạt động của chương trình sort:

Cây hoạt động	Mẫu tin kích hoạt trong Stack
s	<div style="border: 1px solid black; padding: 5px; margin: 0 auto; width: 100px;"> <div style="border-bottom: 1px dotted black; padding-bottom: 2px; text-align: center;">s</div> <div style="padding-bottom: 2px; text-align: center;">a: array</div> </div>
	<div style="border: 1px solid black; padding: 5px; margin: 0 auto; width: 100px;"> <div style="border-bottom: 1px dotted black; padding-bottom: 2px; text-align: center;">s</div> <div style="border-bottom: 1px dotted black; padding-bottom: 2px; text-align: center;">a: array</div> <div style="border-bottom: 1px dotted black; padding-bottom: 2px; text-align: center;">r</div> <div style="padding-bottom: 2px; text-align: center;">i: integer</div> </div>
	<div style="border: 1px solid black; padding: 5px; margin: 0 auto; width: 100px;"> <div style="border-bottom: 1px dotted black; padding-bottom: 2px; text-align: center;">s</div> <div style="border-bottom: 1px dotted black; padding-bottom: 2px; text-align: center;">a: array</div> <div style="border-bottom: 1px dotted black; padding-bottom: 2px; text-align: center;">q(1,9)</div> <div style="padding-bottom: 2px; text-align: center;">i: integer</div> </div>



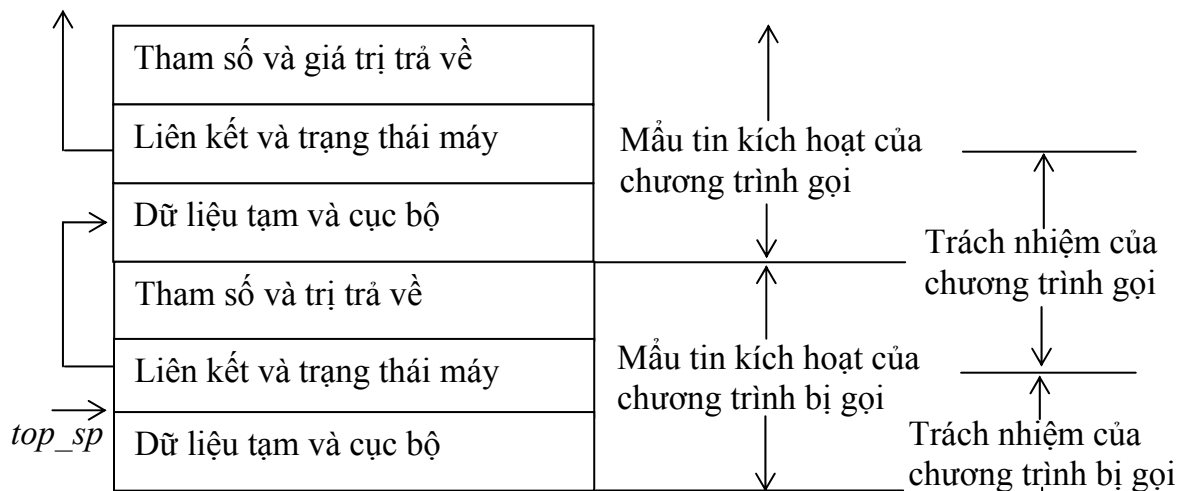
Hình 7.9 - Sự cấp phát và loại bỏ các mẫu tin kích hoạt

Bộ nhớ cho dữ liệu cục bộ trong mỗi lần gọi một chương trình con được chứa trong mẫu tin kích hoạt cho lần gọi đó. Như vậy các tên cục bộ được liên kết với bộ nhớ trong mỗi một hoạt động, bởi vì một mẫu tin kích hoạt được push vào Stack khi có một lời gọi chương trình con. Dữ liệu của các biến cục bộ sẽ bị xóa bỏ khi sự thực hiện chương trình con kết thúc.

Giả sử thanh ghi top đánh dấu đỉnh của Stack. Tại thời gian thực hiện một mẫu tin kích hoạt có thể được cấp phát hoặc thu hồi bằng cách tăng hoặc giảm thanh ghi top bằng kích thước của mẫu tin kích hoạt.

Gọi thực hiện chương trình con

Gọi chương trình con được thực hiện bởi lệnh gọi trong mã đích - lệnh gọi cấp phát một mẫu tin kích hoạt và đưa thông tin vào cho các trường - lệnh trở về sẽ phục hồi các trạng thái máy để chương trình gọi tiếp tục thực hiện



Hình 7.10 - Phân chia công việc giữa chương trình gọi và chương trình bị gọi

Hình trên mô tả mối quan hệ giữa mẫu tin kích hoạt của chương trình gọi và chương trình bị gọi. Mỗi mẫu tin như vậy có ba trường chủ yếu: các tham số thực tế và trị trả về, các mối liên kết và trạng thái máy và cuối cùng là trường dữ liệu tạm và cục bộ.

Thanh ghi `top.sp` chỉ đến cuối trường các mối liên kết và trạng thái máy. Vị trí này được biết bởi chương trình gọi. Đoạn mã cho chương trình bị gọi có thể truy xuất dữ liệu tạm và cục bộ của nó bằng cách sử dụng độ dời (offsets) từ `top.sp`.

Lệnh gọi thực hiện các công việc sau :

1. Chương trình gọi đánh giá các tham số thực tế.
2. Chương trình gọi lưu địa chỉ trả về và giá trị cũ của `top.sp` vào trong mẫu tin kích hoạt của chương trình bị gọi. Sau đó tăng giá trị của `top.sp`.
3. Chương trình được gọi lưu giá trị thanh ghi và các thông tin trạng thái khác
4. Chương trình được gọi khởi tạo dữ liệu cục bộ của nó và bắt đầu thực hiện.

Lệnh trả về thực hiện các công việc sau:

1. Chương trình bị gọi gửi giá trị trả về vào mẫu tin kích hoạt của chương trình gọi.
2. Căn cứ vào thông tin trong trường trạng thái, chương trình bị gọi khôi phục `top_sp` cũng như giá trị các thanh ghi và truyền tới địa chỉ trả về trong mã của chương trình gọi.
3. Mặc dù `top_sp` đã bị giảm, chương trình gọi cần sao chép giá trị trả về vào trong mẫu tin kích hoạt của nó để sử dụng cho việc tính toán biểu thức.

Dữ liệu có kích thước thay đổi

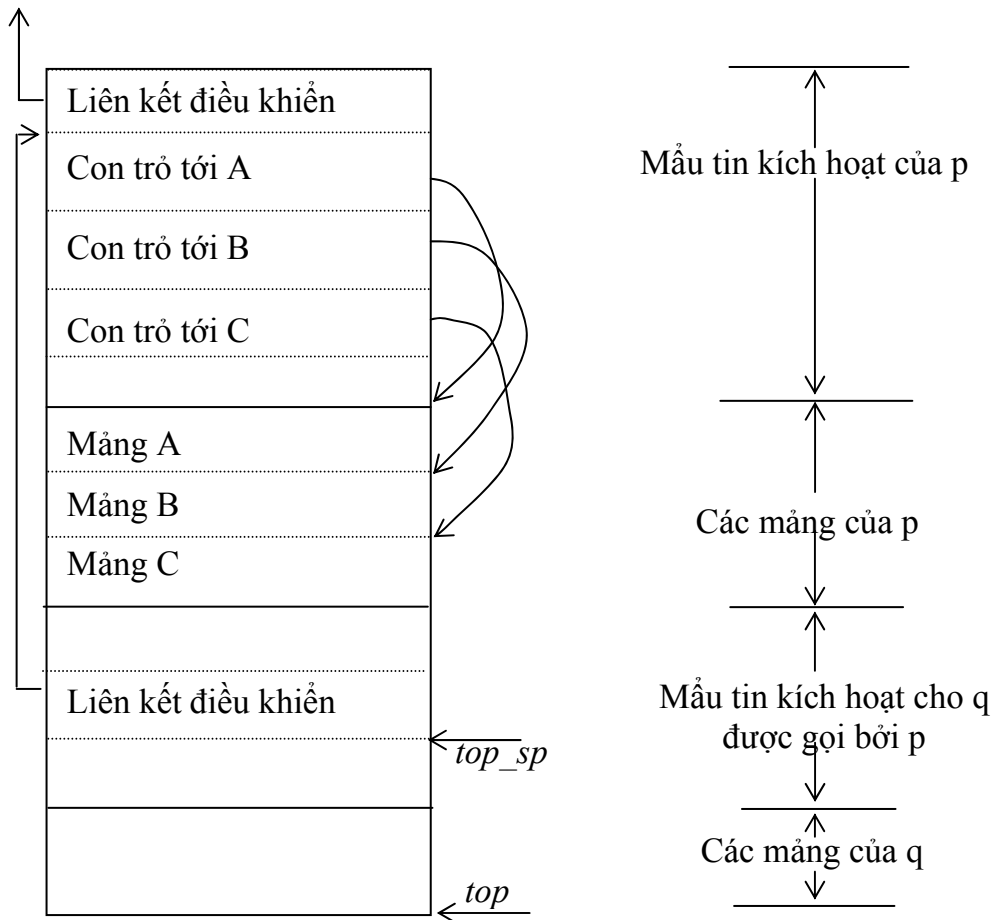
Một số ngôn ngữ cho phép dữ liệu có kích thước thay đổi.

Chẳng hạn chương trình con `p` có 3 mảng có kích thước thay đổi, các mảng này được lưu trữ ngoài mẫu tin kích hoạt của `p`. Trong mẫu tin kích hoạt của `p` chỉ chứa các con trỏ tới điểm bắt đầu của mỗi một mảng. Địa chỉ tương đối của các con trỏ này được biết tại thời gian dịch nên mã đích có thể truy nhập tới các phần tử mảng thông qua con trỏ.

Hình sau trình bày chương trình con `q` được gọi bởi `p`. Mẫu tin kích hoạt của `q` nằm sau các mảng của `p`. Truy nhập vào dữ liệu trong Stack thông qua hai con trỏ **`top`**, **`top.sp`**:

`top` chỉ đỉnh Stack nơi một mẫu tin kích hoạt mới có thể bắt đầu.

`top_sp` dùng để tìm dữ liệu cục bộ



Hình 7.11 - Truy xuất các mảng được cấp phát động

3. Cấp phát Heap

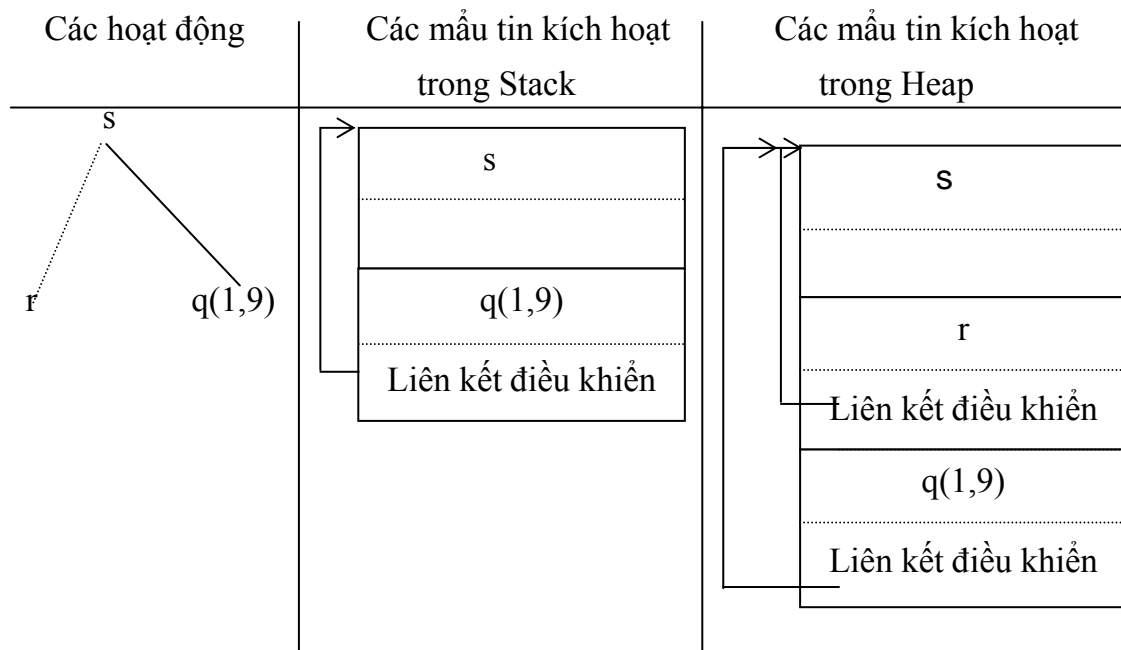
Chiến thuật cấp phát sử dụng Stack không đáp ứng được các yêu cầu sau:

1. Giá trị của tên cục bộ được giữ lại khi hoạt động của chương trình con kết thúc.
2. Hoạt động của chương trình bị gọi tồn tại sau chương trình gọi.

Các yêu cầu trên đều không thể cấp phát và thu hồi theo cơ chế LIFO (Last - In, First - Out) tức là tổ chức theo Stack.

Heap là khối ô nhớ liên tục được chia nhỏ để có thể cấp phát cho các mẫu tin kích hoạt hoặc các đối tượng dữ liệu khác.

Sự khác nhau giữa cấp phát Stack và Heap là ở chỗ mẫu tin cho một hoạt động được giữ lại khi hoạt động đó kết thúc.



Hình 7.12 - Mẫu tin kích hoạt được giữ lại trong Heap

Về mặt vật lý, mẫu tin kích hoạt cho $q(1,9)$ không phụ thuộc mẫu tin kích hoạt cho r . Khi mẫu tin kích hoạt cho r bị giải phóng thì bộ quản lý Heap có thể dùng vùng nhớ tự do này để cấp phát cho mẫu tin khác. Một số vấn đề thuộc quản lý hiệu quả một Heap sẽ được trình bày trong mục VIII.

IV. TRUY XUẤT TÊN KHÔNG CỤC BỘ

1. Quy tắc tầm vực

Quy tắc tầm vực của ngôn ngữ sẽ xác định việc xử lý khi tham khảo đến các tên không cục bộ.

Quy tắc tầm vực bao gồm hai loại: **Quy tắc tầm tĩnh** và **quy tắc tầm động**.

Quy tắc tầm tĩnh (static - scope rule): Xác định sự khai báo áp dụng cho một tên bằng cách kiểm tra văn bản chương trình nguồn. Các ngôn ngữ Pascal, C và Ada sử dụng quy tắc tầm tĩnh với một quy định bổ sung: “tầm gần nhất”.

Quy tắc tầm động (dynamic- scope rule): Xác định sự khai báo có thể áp dụng cho một tên tại thời gian thực hiện bằng cách xem xét hoạt động hiện hành. Các ngôn ngữ Lisp, APL và Snobol sử dụng quy tắc tầm động.

2. Cấu trúc khối

Một khối bắt đầu bởi một tập hợp các khai báo cho tên (khai báo biến, định nghĩa kiểu, định nghĩa hằng...) sau đó là một tập hợp các lệnh mà trong đó các tên có thể được tham khảo.

Cấu trúc khối thường được sử dụng trong các ngôn ngữ cấu trúc như Pascal, Ada, PL/1. Trong đó chương trình hay chương trình con được tổ chức thành các khối lồng nhau.

Ngôn ngữ cấu trúc khối sử dụng quy tắc tầm tĩnh. Tầm của một khai báo được cho bởi **quy tắc tầm gần nhất** (most closely nested).

1. Một khai báo tại đầu một khối xác định một tên cục bộ trong khối đó. Bất kỳ một tham khảo tới tên trong thân khối được xem xét như là một tham khảo tới dữ liệu cục bộ trong khối nếu nó tồn tại.

2. Nếu một tên x được tham khảo trong thân một khối B và x không được khai báo trong B thì x được xem như là một sự tham khảo tới sự khai báo trong B' là khối nhỏ nhất chứa B. Nếu trong B' không có một khai báo cho x thì lại tham khảo tới B'' là khối nhỏ nhất chứa B'.

3. Nếu một khối chứa định nghĩa các khối khác thì mọi khai báo trong các khối con hoàn toàn bị che dấu đối với khối ngoài.

Cấu trúc khối có thể cài đặt bằng cách sử dụng cơ chế cấp phát Stack. Khi điều khiển đi vào một khối thì ô nhớ cho các tên được cấp phát và chúng bị thu hồi khi điều khiển rời khỏi khối.

3. Tầm tĩnh với các chương trình con không lồng nhau

Quy tắc tầm tĩnh của ngôn ngữ C đơn giản hơn so với Pascal và các định nghĩa chương trình con trong C không lồng nhau. Một chương trình C là một chuỗi các khai báo biến và hàm. Nếu có một sự tham khảo không cục bộ đến tên a trong một hàm nào đó thì a phải được tham khảo bên ngoài tất cả các hàm. Tất cả các tên khai báo bên ngoài hàm đều có thể được cấp phát tĩnh. Vị trí các ô nhớ này được biết tại thời gian dịch do đó một tham khảo tới tên không cục bộ trong thân hàm được xác định bằng địa chỉ tuyệt đối. Các tên cục bộ trong hàm nằm trong mẫu tin hoạt động trên đỉnh Stack và có thể xác định bằng cách sử dụng địa chỉ tương đối.

4. Tầm tĩnh với các chương trình con lồng nhau.

Trong ngôn ngữ Pascal các chương trình con có thể lồng nhau nhiều cấp.

Ví dụ 7.5: Xét chương trình

- (1) **program** sort(input, output);
- (2) var a: array [0...10] of integer;
- (3) x : integer;
- (4) **procedure** readarray;
- (5) var y : integer;
- (6) begin ... a... end; {readarray}
- (7) **procedure** exchange(i,j:integer);
- (8) begin
- (9) x:= a[i]; a[i] := a[j]; a[j] := x;
- (10) end; {exchange}
- (11) **procedure** quicksort(m,n:integer);
- (12) var k,v: integer;

- (13) function partition(y,z: integer) : integer;
- (14) var i,j : integer;
- (15) begin...a...
- (16) ...v...
- (17) ...exchange(i,j)...
- (18) end; {partition}
- (19) begin...end; {quicksort}
- (20) begin...end; {sort}

Hình 7.13 - Một chương trình Pascal với các chương trình con lồng nhau

Xét chương trình con partition, trong đó tham khảo đến các tên không cục bộ như:

- a: Khai báo trong chương trình chính.
- v: khai báo trong quicksort;
- exchange:khai báo trong chương trình chính.

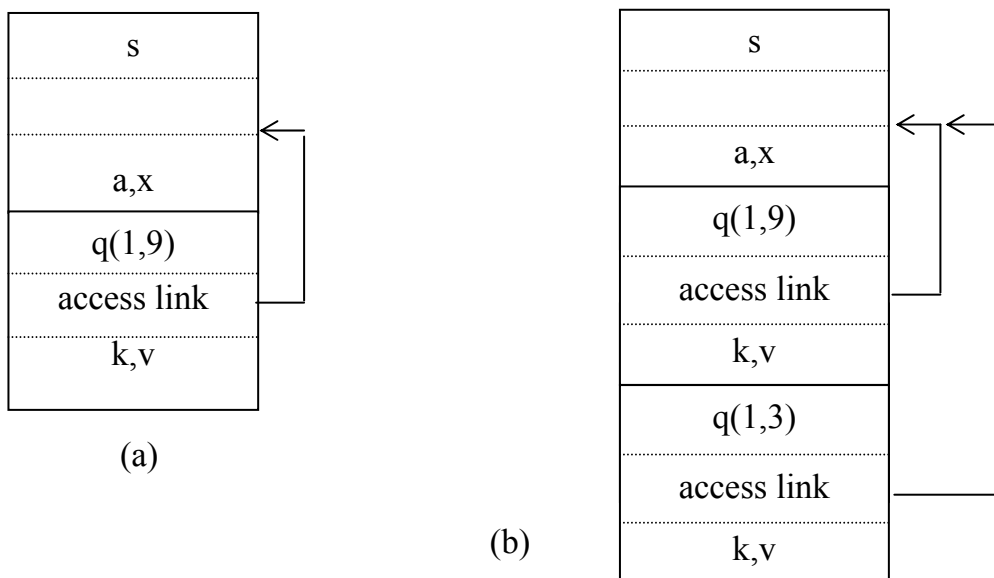
Độ sâu của sự lồng nhau

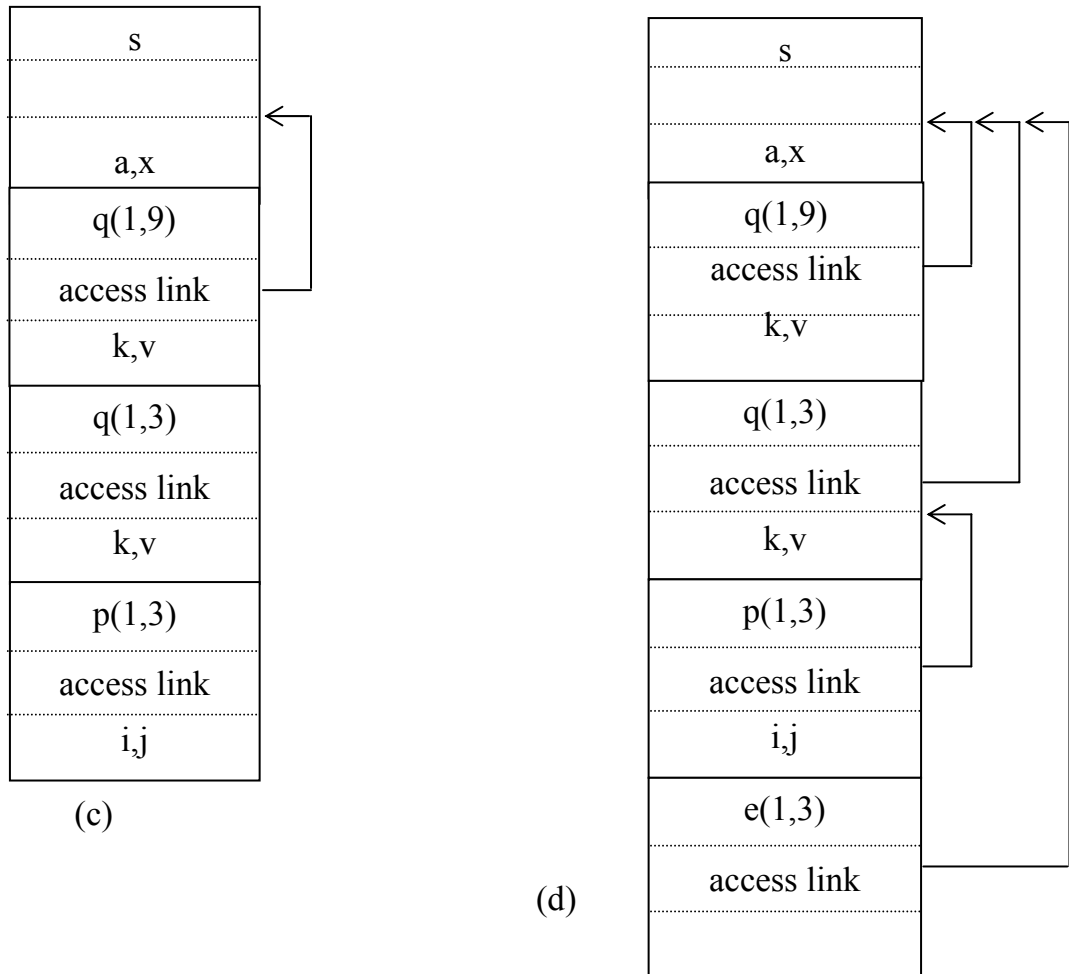
Chúng ta sử dụng thuật ngữ độ lồng sâu để chỉ tầm tĩnh. Tên của chương trình chính có độ sâu cấp một và chúng ta tăng thêm một khi đi từ một chương trình con vào một chương trình con được bao (khai báo) trong nó. Như vậy trong chương trình con partition, a có độ sâu cấp 1, v có độ sâu cấp 2, i có độ sâu cấp 3. Quicksort có độ sâu cấp 2, partition có độ sâu cấp 3, exchange có độ sâu cấp 2.

Liên kết truy xuất

Để cài đặt tầm tĩnh cho các chương trình con lồng nhau ta dùng con trỏ liên kết truy xuất trong mỗi mẫu tin kích hoạt. Nếu chương trình con p được lồng trực tiếp trong q thì liên kết trong mẫu tin kích hoạt của p trỏ tới liên kết truy xuất của mẫu tin kích hoạt hiện hành của q. Hình sau mô tả nội dung Stack trong khi thực hiện chương trình sort trong ví dụ trên

Ví dụ 7.6:





Hình 7.14 - Liên kết truy xuất cho phép tìm kiếm ô nhớ của các tên không cục bộ

Liên kết truy xuất của s rỗng vì s không có bao đóng.

Liên kết truy xuất của một mẫu tin kích hoạt của một chương trình con bất kỳ đều trở đến mẫu tin kích hoạt của bao đóng của nó.

Giả sử chương trình con p có độ lồng sâu np tham khảo tới một tên không cục bộ a có độ lồng sâu $na \leq np$. Việc tìm đến địa chỉ của a được tiến hành như sau:

- Khi chương trình con p được gọi thì một mẫu tin kích hoạt của p nằm trên đỉnh Stack. Tính giá trị $np - na$. Giá trị này được tính tại thời gian dịch.
- Đi xuống $np - na$ mức theo liên kết truy xuất ta tìm đến được mẫu tin kích hoạt của chương trình con trong đó a được khai báo. Tại đây địa chỉ của a được xác định bằng cách lấy địa chỉ của mẫu tin cộng với độ dời của a (địa chỉ tương đối của a).

Ví dụ 7.7 (ứng với hình 7.14c) : Hàm partition có độ lồng sâu là $np = 3$ tham khảo tới biến a có độ lồng sâu $na = 1$ và biến v có độ lồng sâu $nv = 2$.

Để xác định a cần tính $np - na = 3 - 1 = 2 \Rightarrow$ cần hạ hai cấp

Từ p(1,3) hạ một cấp đến q(1,3) theo liên kết truy xuất.

Từ q(1,3) hạ một cấp đến s theo liên kết truy xuất đến s là nơi a được khai báo.

Để xác định v cần tính $np - nv = 3 - 2 = 1 \Rightarrow$ cần hạ một cấp xuống $q(1,3)$ là nơi v được khai báo.

Giả sử chương trình con p có độ lồng sâu np gọi chương trình con e ở độ lồng sâu ne . Đoạn mã để thiết lập liên kết truy xuất phụ thuộc vào việc chương trình được gọi có được định nghĩa trong chương trình gọi hay không?

Trường hợp 1: $np < ne$: Chương trình con e có độ lồng sâu lớn hơn chương trình con p do đó hoặc e được lồng trong p hoặc p không thể tham khảo đến e (e bị che dấu khỏi p). Ví dụ `sort` gọi `quickort`, `quicksort` gọi `partition`.

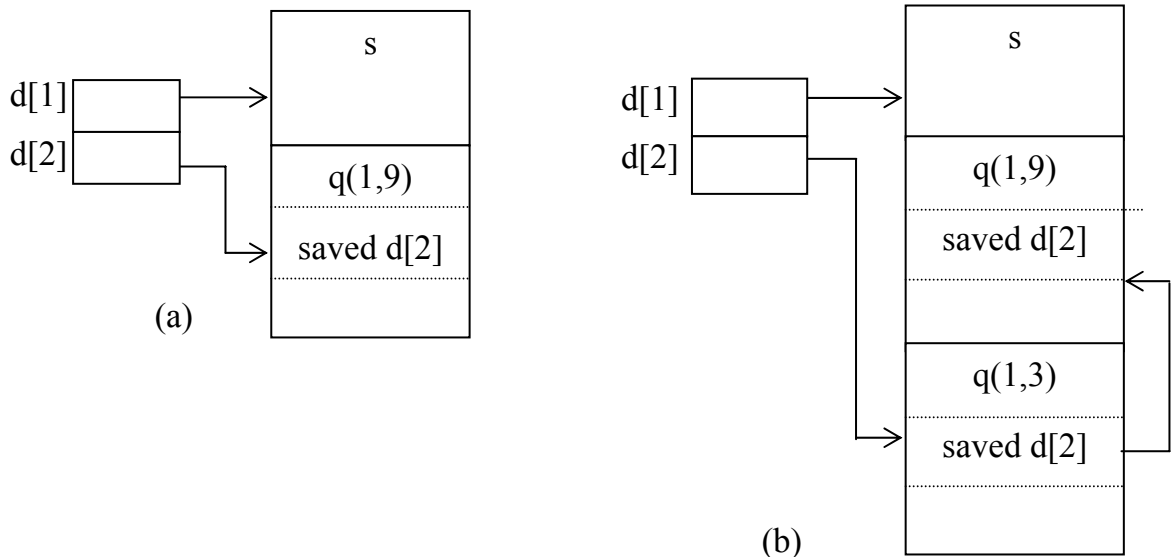
Trường hợp 2: $np \geq ne$: chương trình con e có độ lồng sâu nhỏ hơn hoặc bằng độ lồng sâu của chương trình con p . Theo quy tắc tầm tĩnh thì p có thể tham khảo e . Ví dụ `quicksort` gọi chính nó, `partition` gọi `exchange`. Từ chương trình gọi $np - ne + 1$ bước làm theo liên kết truy nhập ta tìm được mẫu tin kích hoạt của bao đóng gần nhất chứa cả chương trình gọi và chương trình được gọi. Chẳng hạn $p(1,3)$ gọi $e(1,3)$, $np = 3$, $ne = 2$. Ta phải làm $3 - 2 + 1$ bằng hai bước theo liên kết truy xuất từ p đến s .

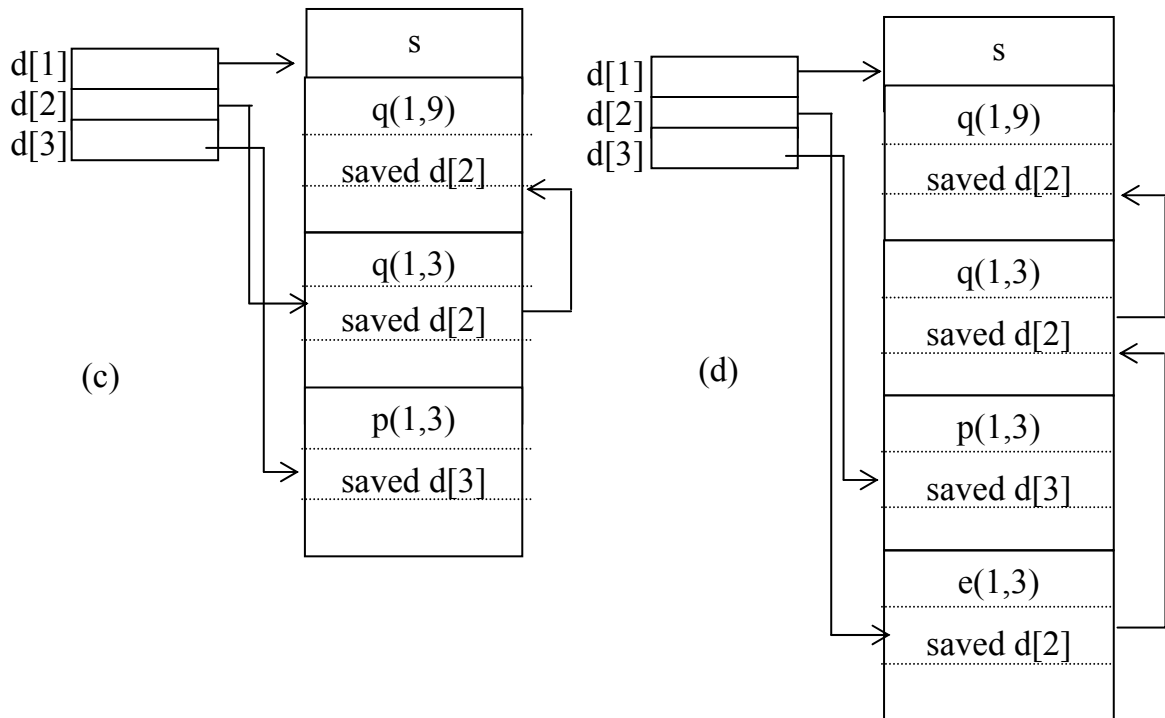
Display: để truy xuất nhanh các tên không cục bộ người ta dùng một mảng d các con trỏ tới các mẫu tin kích hoạt mảng này gọi là `display`.

Giả sử điều khiển nằm trong hoạt động của chương trình con t có độ lồng sâu j thì $j-1$ phần tử của `display` trỏ tới các mẫu tin kích hoạt của các bao đóng gần nhất của p và $d[j]$ trỏ tới kích hoạt của p .

Một tên không cục bộ a có độ sâu i nằm trong mẫu tin kích hoạt được trỏ bởi $d[i]$.

Ví dụ 7.8:





Hình 7.15 - Sử dụng display khi các chương trình con không được truyền như các tham số

(a): Tình trạng trước khi $q(1,3)$ bắt đầu, quicksort có độ lồng sâu cấp 2, $d[2]$ được gửi cho mẫu tin kích hoạt của quicksort khi nó bắt đầu. Giá trị của $d[2]$ được lưu trong mẫu tin kích hoạt của $q(1,9)$.

(b): Khi $q(1,3)$ bắt đầu $d[2]$ trở tới mẫu tin kích hoạt mức ứng với $q(1,3)$, giá trị của $d[2]$ lại được lưu trong mẫu tin này. Giá trị này là cần thiết để phục hồi display cũ khi điều khiển trả về cho $q(1,9)$. Như vậy khi một mẫu tin kích hoạt mới được đẩy vào Stack thì:

- Lưu giá trị của $d[i]$ vào mẫu tin đó.
- Đặt $d[i]$ trở tới mẫu tin đó.

Khi một mẫu tin được pop khỏi Stack thì $d[i]$ được phục hồi.

Giả sử một chương trình con có độ lồng sâu cấp j gọi một chương trình con có độ lồng sâu cấp i . Có hai trường hợp xảy ra phụ thuộc chương trình con được gọi có được định nghĩa trong chương trình gọi hay không.

Trường hợp 1: $j < i \Rightarrow i = j+1$: thêm ô nhớ $d[i]$, cấp phát mẫu tin kích hoạt cho chương trình con i , ghi $d[i]$ vào trong đó và đặt $d[i]$ trở tới nó (ví dụ 7.8a, 7.8c)

Trường hợp 2: $j \geq i$: Ghi giá trị cũ của $d[i]$ vào mẫu tin kích hoạt mới và đặt $d[i]$ trở vào mẫu tin cuối. (ví dụ 7.8b và 7.8d)

5. Tầm động

Với khái niệm tầm động, một hoạt động mới kế thừa sự liên kết đã tồn tại của một tên không cục bộ. Tên không cục bộ a trong hoạt động của chương trình được gọi tham khảo đến cùng một ô nhớ như trong hoạt động của chương trình gọi. Đối với tên cục bộ thì một liên kết mới được thiết lập tới ô nhớ trong mẫu tin hoạt động mới.

Ví dụ 7.9: Xét chương trình:

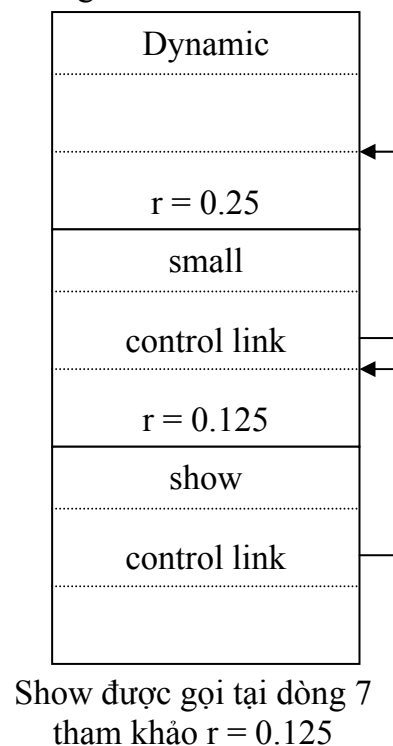
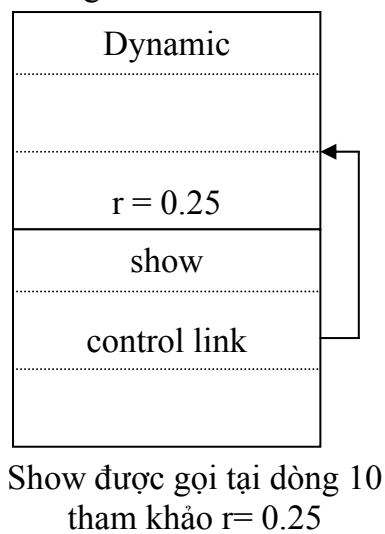
- (1) **program** dynamic (input, output);
- (2) var r : real;
- (3) **procedure** show;
- (4) begin write(r : 5 : 3); end;
- (5) **procedure** small;
- (6) var r : real;
- (7) begin r := 0.125; show; end;
- (8) begin
- (9) r := 0.25;
- (10) show, small, writeln;
- (11) end;

Hình 7.16 - Kết quả chương trình tùy thuộc vào tầm động hay tầm tĩnh được sử dụng

Kết quả thực hiện chương trình:

- Dưới tầm tĩnh;
0.250 0.250
- Dưới tầm động:
0.250 0.125

Khi show được gọi tại dòng 10 trong chương trình chính thì 0.250 được in ra vì r của chương trình chính được sử dụng. Tuy nhiên khi show được gọi tại dòng 7 trong small thì 0.125 được in ra vì r của chương trình con small được sử dụng. Cơ chế tầm động sử dụng liên kết điều khiển để tham khảo tên không cục bộ.



Hình 7.17 - Sử dụng liên kết điều khiển để tham khảo các tên không cục bộ

V. TRUYỀN THAM SỐ

Khi một chương trình con gọi một chương trình con khác thì phương pháp thông thường để giao tiếp giữa chúng là thông qua tên không cục bộ và thông qua các tham số của chương trình được gọi.

Ví dụ 7.10: Để đổi hai giá trị $a[i]$ và $a[j]$ cho nhau ta dùng

- (1) **procedure** exchange(i, j : integer);
- (2) var x : integer;
- (3) begin
- (4) $x := a[i]$; $a[i] := a[j]$; $a[j] := x$;
- (5) end;

trong đó mảng a là tên không cục bộ và i, j là các tham số.

Có rất nhiều phương pháp truyền tham số như:

- Truyền bằng giá trị (Transmission by value, call-by-value)
- Truyền bằng tham khảo (Transmission by name, call-by-name)...

Ở đây chúng ta xét hai phương pháp phổ biến nhất:

1. Truyền bằng giá trị

Là phương pháp đơn giản nhất của truyền tham số được sử dụng trong C và Pascal. Truyền bằng giá trị được xử lý như sau:

1. Tham số hình thức được xem như là tên cục bộ do đó ô nhớ của các tham số hình thức nằm trong mẫu tin kích hoạt của chương trình được gọi.
2. Chương trình gọi đánh giá các tham số thực tế và đặt các giá trị của chúng vào trong ô nhớ của tham số hình thức.

2. Truyền tham chiếu (truyền địa chỉ hay truyền vị trí)

Chương trình gọi truyền cho chương trình được gọi con trỏ tới địa chỉ của mỗi một tham số thực tế.

Ví dụ 7.11:

- (1) **program** reference (input, output)
- (2) var i : integer;
- (3) a : array[0...10] of integer;
- (4) **procedure** swap(var x, y : integer);
- (5) var $temp$: integer;
- (6) begin
- (7) $temp := x$;

```

(8)    x := y;
(9)    y := temp;
(10)   end;
(11)   begin
(12)    i := 1;  a[1] := 2;
(13)    swap(i,a[1]);
(14)   end;

```

Hình 7.18 - Chương trình Pascal với thủ tục swap

Với lời gọi tại dòng (13) ta có các bước sau:

1. Copy địa chỉ của i và $a[i]$ vào trong mẫu tin hoạt động của swap thành $arg1$, $arg2$ tương ứng với x , y .
2. Đặt temp bằng nội dung của vị trí được trả về bởi $arg1$ tức là $temp := 1$. Bước này tương ứng lệnh $temp := x$ trong dòng (7) của swap.
3. Đặt nội dung của vị trí được trả về bởi $arg1$ bởi giá trị của vị trí được trả về bởi $arg2$, tức là $i := a[1]$. Bước này tương ứng lệnh $x := y$ trong dòng (8) của swap.
4. Đặt nội dung của vị trí được trả về bởi $arg2$ bởi giá trị của temp. Tức là $a[1] := i$. Bước này tương ứng lệnh $y := temp$.

VI. BẢNG KÝ HIỆU

Chương trình dịch sẽ sử dụng bảng ký hiệu để lưu trữ thông tin về tầm vực và mối liên kết của các tên. Bảng ký hiệu được truy xuất nhiều lần mỗi khi một tên xuất hiện trong chương trình nguồn.

Có hai cơ chế tổ chức bảng ký hiệu là danh sách tuyến tính và bảng băm.

1. Cấu trúc một ô của bảng ký hiệu

Mỗi ô trong bảng ký hiệu tương ứng với một tên. Định dạng của các ô này thường không giống nhau vì thông tin lưu trữ về một tên phụ thuộc vào việc sử dụng tên đó. Thông thường một ô được cài đặt bởi một mẫu tin. Nếu muốn có được sự đồng nhất của các mẫu tin ta có thể lưu thông tin bên ngoài bảng ký hiệu, trong mỗi ô của bảng chỉ chứa các con trỏ tới thông tin đó,

Trong bảng ký hiệu cũng có thể có lưu các từ khóa của ngôn ngữ. Nếu vậy thì chúng phải được đưa vào bảng ký hiệu trước khi bộ phân tích từ vựng bắt đầu.

2. Vấn đề lưu trữ lexeme của danh biểu

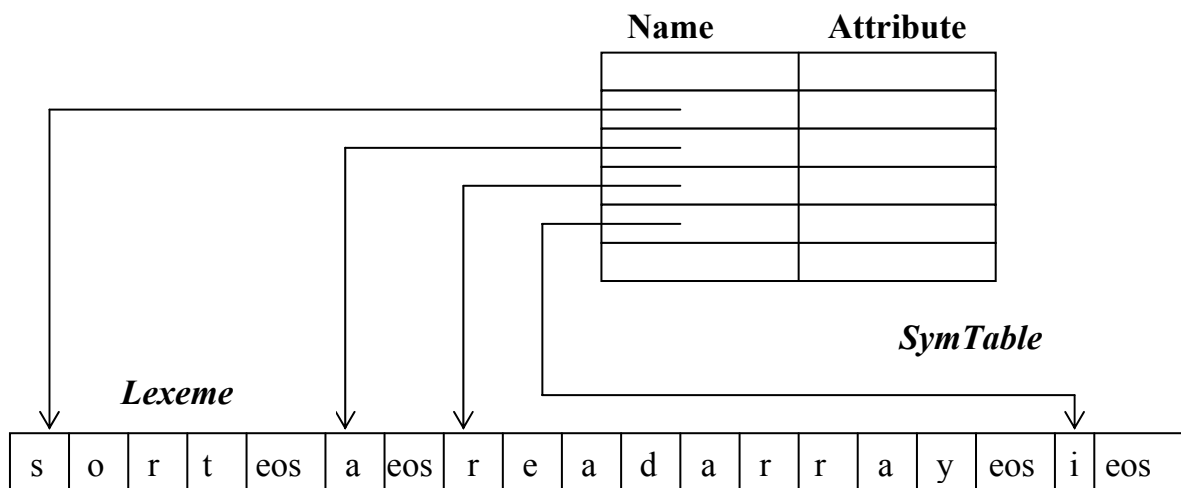
Các danh biểu trong các ngôn ngữ lập trình thường có hai loại: Một số ngôn ngữ quy định độ dài của danh biểu không được vượt quá một giới hạn nào đó. Một số khác không giới hạn về độ dài.

Trường hợp danh biểu bị giới hạn về độ dài thì chuỗi các ký tự tạo nên danh biểu được lưu trữ trong bảng ký hiệu.

Name									Attribute
s	o	r	t						
a									
r	e	a	d	a	r	r	a	y	
i									

Hình 7.19 - Bảng ký hiệu lưu giữ các tên bị giới hạn độ dài

Trường hợp độ dài tên không bị giới hạn thì các Lexeme được lưu trong một mảng riêng và bảng ký hiệu chỉ giữ các con trỏ tới đầu mỗi Lexeme



Hình 7.20 - Bảng ký hiệu lưu giữ các tên không bị giới hạn độ dài

3. Tổ chức bảng ký hiệu bằng danh sách tuyến tính

Cấu trúc đơn giản, dễ cài đặt nhất cho bảng ký hiệu là danh sách tuyến tính của các mẫu tin. Ta dùng một mảng hoặc nhiều mảng tương đương để lưu trữ tên và các thông tin kết hợp với chúng. Các tên mới được đưa vào trong danh sách theo thứ tự mà chúng được phát hiện. Vị trí của mảng được đánh dấu bởi con trỏ available chỉ ra một ô mới của bảng sẽ được tạo ra.

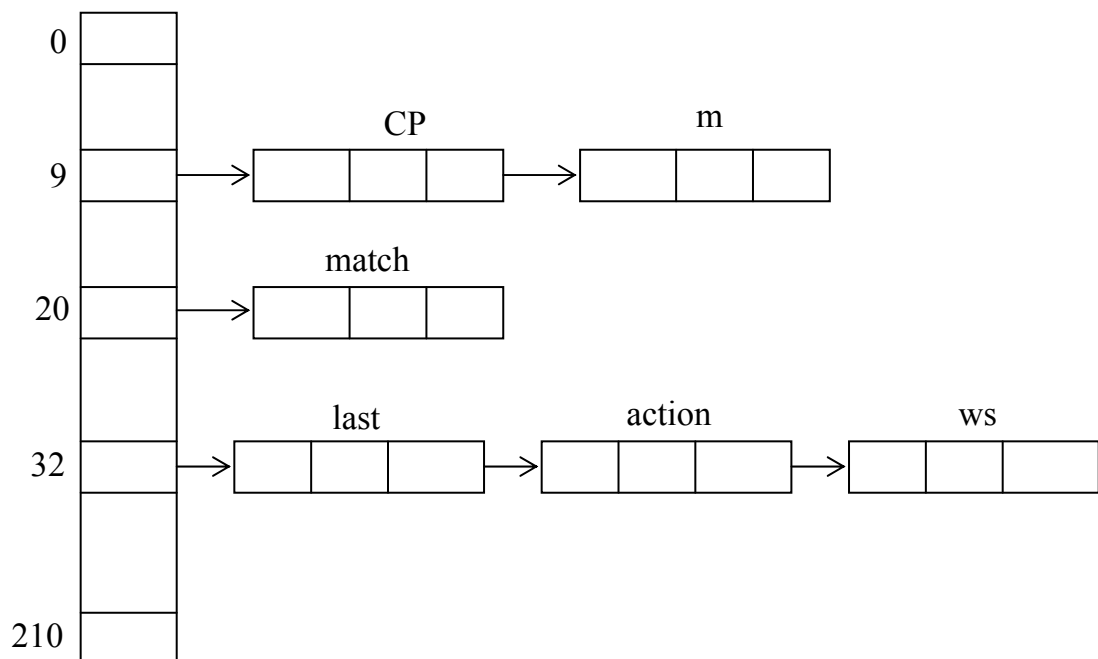
Việc tìm kiếm một tên trong bảng ký hiệu được bắt đầu từ available đến đầu bảng. Trong các ngôn ngữ cấu trúc khối sử dụng quy tắc tầm tĩnh. Thông tin kết hợp với tên có thể bao gồm cả thông tin về độ sâu của tên. Bằng cách tìm kiếm từ available trở về đầu mảng chúng ta đảm bảo rằng sẽ tìm thấy tên trong tầng gần nhất.

id1
info 1
id2
info2
...

Hình 7.21 - Danh sách tuyến tính các mẫu tin

4. Tổ chức bảng ký hiệu bằng bảng băm

Kỹ thuật sử dụng bảng băm để cài đặt bảng ký hiệu thường được sử dụng vì tính hiệu quả của nó. Cấu tạo bao gồm hai phần; bảng băm và các danh sách liên kết.



Hình 7.22 - Bảng băm có kích thước 211

1. Bảng băm là một mảng bao gồm m con trỏ.
2. Bảng danh biểu được chia thành m danh sách liên kết, mỗi danh sách liên kết được trỏ bởi một phần tử trong bảng băm.

Việc phân bổ các danh biểu vào danh sách liên kết nào do hàm băm (hash function) quy định. Giả sử s là chuỗi ký tự xác định danh biểu, hàm băm h tác động lên s trả về một giá trị nằm giữa 0 và m-1 $h(s) = t \Rightarrow$ Danh biểu s được đưa vào trong danh sách liên kết được trỏ bởi phần tử t của bảng băm.

Có nhiều phương pháp để xác định hàm băm.

Phương pháp đơn giản nhất như sau:

1. Giả sử s bao gồm các ký tự $c_1, c_2, c_3, \dots, c_k$. Mỗi ký tự cho ứng với một số nguyên dương $n_1, n_2, n_3, \dots, n_k$; lấy $h = n_1 + n_2 + \dots + n_k$.
2. Xác định $h(s) = h \bmod m$

BÀI TẬP CHƯƠNG VII

7.1. Hãy dùng quy tắc tầm vực của ngôn ngữ Pascal để xác định tầm vực ý nghĩa của các khai báo cho mỗi lần xuất hiện tên a, b trong chương trình sau. Output của chương trình là các số nguyên từ 1 đến 4.

```
Program    a ( input, output);  
Procedure  b ( u, v, x, y : integer);  
    Var    a : record    a, b : integer end;  
          b : record    a, b : integer end;  
  
    begin  
        With a do begin    a := u ; b := v  end;  
        With b do begin    a := x ; b := y  end;  
        Writeln ( a.a, a.b, b.a, b.b );  
  
    end;  
  
Begin  
    B ( 1, 2, 3, 4)  
  
End.
```

7.2. Chương trình sau sẽ in ra giá trị như thế nào nếu giả sử thông số được truyền bằng:

- a) trị
- b) quy chiếu
- c) trị - kết quả
- d) tên

```
Program    main ( input, output);  
Procedure  p ( x, y, z );  
    begin  
        y := y + 1;  
        z := z + x;  
  
    end;  
  
Begin  
    a := 2 ;  
    b := 3 ;  
    p ( a + b ; a, a )  
    print a
```

End.

7.3. Cho đoạn chương trình trong **Algol** như sau :

```
begin ...  
Procedure A ( px); procedure px      { tham số hình thức px là thủ tục }  
begin  
    procedure B ( pz); procedure pz    { tham số hình thức pz là thủ tục }  
        begin  
            ....  
            pz;  
            ....  
        end;  
    B (px);  
end;  
procedure C;  
    begin  
        procedure D;  
            begin ... end;  
        A(D);  
    end;  
C;  
end.
```

Hãy giải thích quá trình thực thi của chương trình trên, các bước truyền tham số (giải thích bằng hình ảnh của Stack).

7.4. Cho đoạn chương trình sau:

```
var a, b : integer;  
Procedure AB  
    Var a, c : real;  
        k, l : integer;  
    procedure AC  
        Var x, y : real;  
            b : array [ 1 .. 10] of integer;  
    begin
```



```

        ....
        end;
    begin
        ....
        end;
begin
    ....
end.

```

Hãy xây dựng bảng ký hiệu thao các phương pháp sau:

a) Danh sách tuyến tính

b) Băm (hash), nếu giả sử ta có kết quả của hàm biến đổi băm như sau:

```

a = 3;      b = 4;      c = 4;      k = 2;      l = 3;
x = 4;      y = 5;      AB = 2;     AC = 6;

```

7.5. Cho đoạn chương trình sau:

```

Program   baitap;
Var    a : real;
        procedure   sub1 ;
            Var    x, y : real;
            begin
                ....
            end;
        procedure   sub2 (t :integer);
            Var    k : integer;
            procedure   sub3 ;
                Var    m : real;
                begin
                    ....
                end;
            procedure   t;
                Var    x, y : real;
                begin
                    ....
                end;

```

begin

....

end.

Hãy vẽ bảng ký hiệu cho từng chương trình con có con trỏ trỏ đến bảng ký hiệu của chương trình bị gọi và có con trỏ trỏ ngược lại bảng ký hiệu của chương trình gọi nó.