PHENIKAA UNIVERSITY
Realizing your potential

CHƯƠNG TRÌNH DỊCH

Chương 6. Sinh mã trung gian

TS. Phạm Văn Cảnh
Khoa Công nghệ thông tin

Email: canh.phamvan@phenikaa-uni.edu.vn

Nội dung

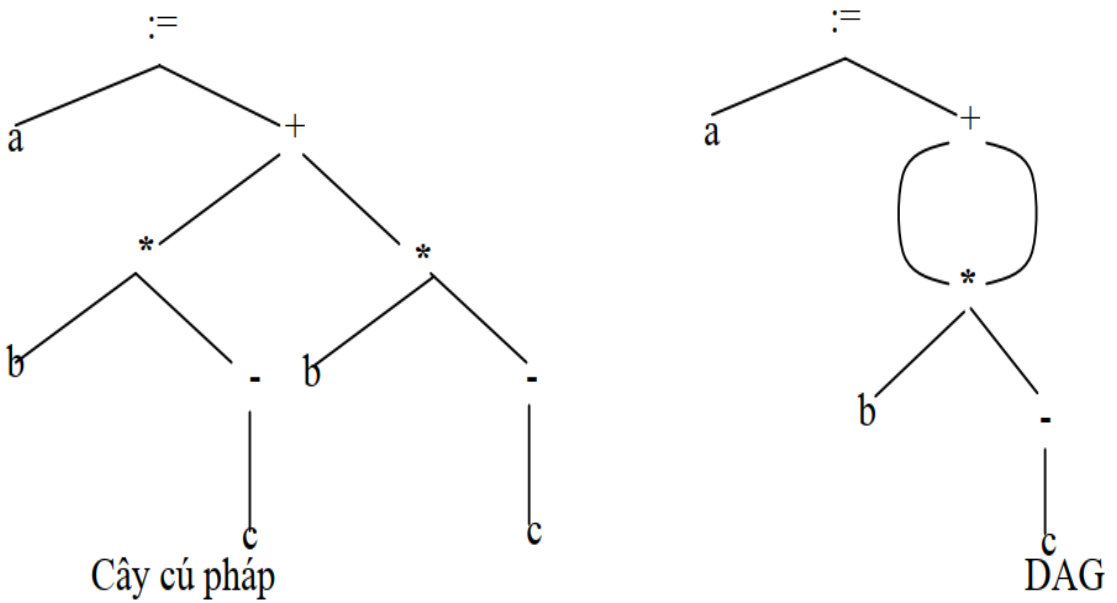
1. Ngôn ngữ trung gian
2. Khai báo
3. Lệnh gán
4. Biểu thức logic
5. Lệnh Case
6. Bài tập

1. Ngôn ngữ trung gian

- Thay vì một chương trình nguồn được dịch trực tiếp sang mã đích, nó nên được dịch sang dạng mã trung gian bởi kỳ trước trước khi được tiếp tục dịch sang mã đích bởi kỳ sau vì một số tiện ích:
  - Thuận tiện khi muốn thay đổi cách biểu diễn chương trình đích;
  - Giảm thời gian thực thi chương trình đích vì mã trung gian có thể được tối ưu.
- Chương này giới thiệu các dạng biểu diễn trung gian đặc biệt là dạng mã ba địa chỉ.

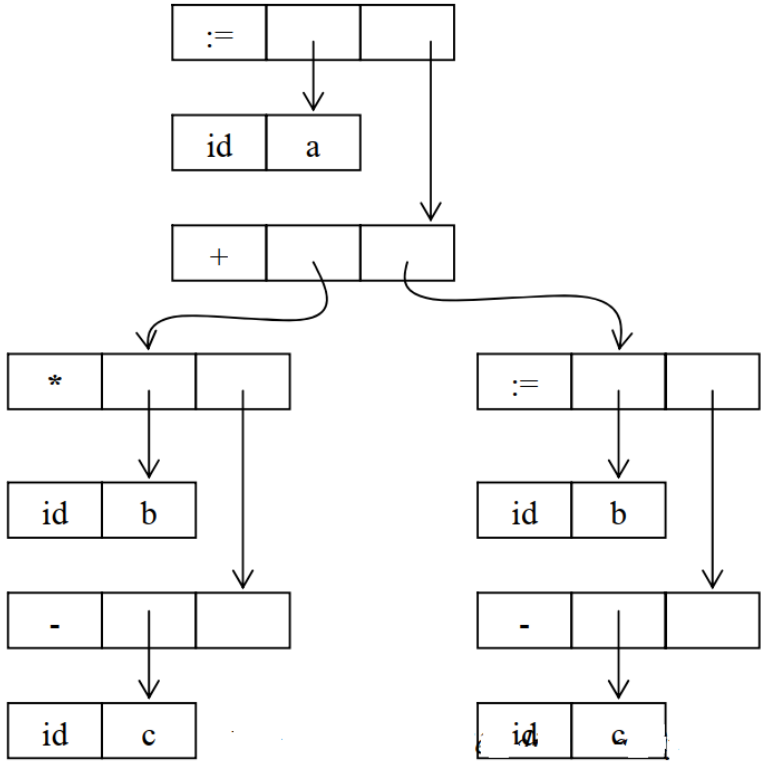
1. Ngôn ngữ trung gian: Biểu diễn đồ thị

- Cây cú pháp mô tả cấu trúc phân cấp tự nhiên của chương trình nguồn.
- DAG cho ta cùng lượng thông tin nhưng bằng cách biểu diễn ngắn gọn hơn trong đó các biểu thức con không được biểu diễn lặp lại.
- Ký pháp hậu tố là một biểu diễn tuyến tính của cây cú pháp. Nó là một danh sách các nút của cây, trong đó một nút xuất hiện ngay sau con của nó.
- Ví dụ: a b c - * b c - * + := là biểu diễn hậu tố của cây cú pháp hình trên.



1. Ngôn ngữ trung gian: Biểu diễn đồ thị

- Cây cú pháp có thể được cài đặt bằng một trong 2 phương pháp:
  - Mỗi nút được biểu diễn bởi một mẫu tin, với một trường cho toán tử và các trường khác trỏ đến con của nó.
  - Một mảng các mẫu tin, trong đó chỉ số của phần tử mảng đóng vai trò như là con trỏ của một nút.



0	id	b	
1	id	c	
2	-	1	
3	*	0	2
4	id	b	
5	id	c	
6	-	5	
7	*	4	6
8	+	3	7
9	id	a	
10	:=	9	8
11	....	....	.....

1. Ngôn ngữ trung gian: Mã 3 địa chỉ

- Mã lệnh 3 địa chỉ là một chuỗi các lệnh có dạng tổng quát là x :=y op z.
  - Trong đó x,y,z là tên, hằng hoặc dữ liệu tạm sinh ra trong khi dịch, op là một toán tử số học hoặc logic.
- Chú ý rằng không được có quá một toán tử ở vế phải của mỗi lệnh. Do đó biểu thức x + y* z phải được dịch thành chuỗi:

```
t1 := y * z
t2 := x + t1
```

  - Trong đó t1, t2 là những tên tạm sinh ra trong khi dịch.

1. Ngôn ngữ trung gian: Mã 3 địa chỉ

- Mã lệnh 3 địa chỉ là một chuỗi các lệnh có dạng tổng quát là x :=y op z.
- Ví dụ:
  - Mã lệnh 3 địa chỉ là một biểu diễn tuyến tính của cây cú pháp hoặc DAG, trong đó các tên tường minh biểu diễn cho các nút trong trên đồ thị.

```
t1 := - c
t2 := b * t1
t3 := - c
t4 := b * t3
t5 := t2 + t4
a := t5
```

```
t1 := -c
t2 := b * t1
t3 := t2 + t2
a := t3
```

Mã lệnh 3 địa chỉ của cây cú pháp và mã lệnh 3 địa chỉ của DAG

1. Ngôn ngữ trung gian: các dạng mã 3 địa chỉ

1. Lệnh gán dạng **x := y op z**, trong đó op là toán tử 2 ngôi số học hoặc logic.
2. Lệnh gán dạng **x := op y**, trong đó op là toán tử một ngôi. Chẳng hạn, phép lấy số đối, toán tử NOT, các toán tử SHIFT, các toán tử chuyển đổi.
3. Lệnh COPY dạng **x :=y**, trong đó giá trị của y được gán cho x.
4. Lệnh nhảy không điều kiện **goto L**. Lệnh 3 địa chỉ có nhãn L là lệnh tiếp theo thực hiện.
5. Các lệnh nhảy có điều kiện như **if x relop y goto L**.
   - Lệnh này áp dụng toán tử quan hệ relop (<, =, >=, ...) vào x và y.
   - Nếu x và y thỏa quan hệ relop thì lệnh nhảy với nhãn L sẽ được thực hiện, ngược lại lệnh đứng sau IF x relop y goto L sẽ được thực hiện.

1. Ngôn ngữ trung gian: các dạng mã 3 địa chỉ

6. param x và call p, n cho lời gọi chương trình con và return y. Trong đó, y biểu diễn giá trị trả về có thể lựa chọn. Cách sử dụng phổ biến là một chuỗi lệnh 3 địa chỉ.

```
param     x1
param     x2
.. .. ..
param     xn
call      p, n
```

được sinh ra như là một phần của lời gọi chương trình con p (x1,x2,.. .., xn).

7. Lệnh gán dạng x := y[i] và x[i] := y.

- Lệnh đầu lấy giá trị của vị trí nhớ của y được xác định bởi giá trị ô nhớ i gán cho x.
- Lệnh thứ 2 lấy giá trị của y gán cho ô nhớ x được xác định bởi i.

1. Ngôn ngữ trung gian: các dạng mã 3 địa chỉ

8. Lệnh gán địa chỉ và con trỏ dạng **x :=&y , x := *y và *x := y.**

- Trong đó, x := &y đặt giá trị của x bởi vị trí của y.
- Câu lệnh x := *y với y là một con trỏ mà r_value của nó là một vị trí, r_value của x đặt bằng nội dung của vị trí này.
- Cuối cùng *x := y đặt r_value của đối tượng được trỏ bởi x bằng r_value của y

1. Ngôn ngữ trung gian:

Dịch trực tiếp cú pháp thành mã 3 địa chỉ

PHENIKAA UNIVERSITY
Realizing your potential

❑ Ví dụ: Định nghĩa S _ thuộc tính sinh mã lệnh địa chỉ cho lệnh gán:

Luật sinh	Luật ngữ nghĩa
S→ id := E	*S.code := E.code || gen (id.place ':=' E.place)*
E→ E_1 + E_2	*E.place := newtemp;* *E.code := E_1.code || E_2.code ||* *gen (E.place ':=' E_1.place '+' E_2.place)*
E→ E_1 * E_2	*E.place := newtemp;* *E.code := E_1.code || E_2.code ||* *gen (E.place ':=' E_1.place '*' E_2.place)*
E→ - E_1	*E.place := newtemp;* *E.code := E_1.code|| gen (E.place ':=' 'uminus' E_1.place)*
E→ (E_1)	*E.place := id.place;* *E.code := ''*
E→ id	

- Với chuỗi nhập a = b * - c + b * - c:
- Thuộc tính tổng hợp S.code biểu diễn mã 3 địa chỉ cho lệnh gán S.
- Ký hiệu chưa kết thúc E có 2 thuộc tính
  - E.place là giá trị của E
  - E.code là chuỗi lệnh 3 địa chỉ để đánh giá E.

  a := t5
  t1 := -c
  *t2 := b * t1*
  t3 := - c
  *t4 := b * t3*
  t5 := t2 + t4
- Khi mã lệnh 3 địa chỉ được sinh, tên tạm được tạo ra cho mỗi nút trong trên cây cú pháp.

1. Ngôn ngữ trung gian:

Dịch trực tiếp cú pháp thành mã 3 địa chỉ

- Ví dụ: Định nghĩa trực tiếp cú pháp sinh mã lệnh ba địa chỉ cho câu lệnh while
  - Lệnh S → while E do S1 được sinh ra bằng cách dùng các thuộc tính S.begin và S.after để đánh dấu lệnh đầu tiên trong đoạn mã lệnh của E và lệnh sau đoạn mã lệnh của S.
  - Hàm newlabel trả về một nhãn mới tại mỗi lần được gọi

Luật sinh	Luật ngữ nghĩa
S→ while E do S_1	*S.begin := newlabel;* *S.after := newlabel;* S.code := gen(S.begin ':') || E.code || *gen('if' E.place '=' 0 'goto' S.after)* || *S_1.code* || *gen('goto' S.begin)* || *gen(S.after ':')*

1. Ngôn ngữ trung gian:

Cài đặt mã lệnh 3 địa chỉ

- Lệnh 3 địa chỉ là một dạng trừu tượng của mã lệnh trung gian.
- Trong chương trình dịch, các mã lệnh này có thể cài đặt bằng một mẩu tin với các trường cho toán tử và toán hạng.
- Có 3 cách biểu diễn là **bộ tứ, bộ tam và bộ tam gián tiếp.**

1. Ngôn ngữ trung gian:

Cài đặt mã lệnh 3 địa chỉ

- Bộ tứ (quadruples) là một cấu trúc mẩu tin có 4 trường ta gọi là op, arg1, arg2 và result.
  - Trường op chứa toán tử. Lệnh 3 địa chỉ x := y op z được biểu diễn bằng cách thay thế y bởi arg1, z bởi arg2 và x bởi result.
  - Các lệnh với toán tử một ngôi như x := -y hay x := y thì không sử dụng arg2. Các toán tử như param không sử dụng cả arg2 lẫn result.
  - Các lệnh nhảy có điều kiện và không điều kiện đặt nhãn đích trong result.
  - Nội dung các trường arg1, arg2 và result trỏ tới ô trong bảng ký hiệu đối với các tên biểu diễn bởi các trường này.
  - Các tên tạm phải được đưa vào bảng ký hiệu khi chúng được tạo ra.

- Ví dụ: Bộ tứ cho lệnh a := b * -c + b * -c

	op	arg1	arg2	result
(0)	uminus	c		t1
(1)	*	b	t1	t2
(2)	uminus	c		t3
(3)	*	b	t3	t4
(4)	+	t2	t4	t5
(5)	:=	t5		a

1. Ngôn ngữ trung gian:

Cài đặt mã lệnh 3 địa chỉ

- **Bộ tam (quadruples)**
  - Để tránh phải lưu tên tạm trong bảng ký hiệu; chúng ta có thể tham khảo tới giá trị tạm bằng vị trí của lệnh tính ra nó.
  - Để làm điều đó ta sử dụng bộ tam (triples) là một mẩu tin có 3 trường op, arg1 và arg2.
    - Trong đó, arg1 và arg2 trỏ tới bảng ký hiệu (đối với tên hoặc hằng do người lập trình định nghĩa) hoặc trỏ tới một phần tử trong bộ tam (đối với giá trị tạm).
    - Ví dụ: Biểu diễn bộ tam cho các lệnh ba địa chỉ. Các lệnh như x[i]:=y và x:=y[i] sử dụng 2 ô trong cấu trúc bộ tam.

	op	arg1	arg2
(0)	uminus	c	
(1)	*	b	(0)
(2)	uminus	c	
(3)	*	b	(2)
(4)	+	(1)	(3)
(5)	:=	a	(4)

	op	arg1	arg2
(0)	[ ]	x	i
(2)	:=	(0)	y

	op	arg1	arg2
(0)	[ ]	y	i
(2)	:=	x	(0)

1. Ngôn ngữ trung gian:

Cài đặt mã lệnh 3 địa chỉ

- **Bộ ba gián tiếp:** Một cách biểu diễn khác của bộ tam là thay vì liệt kê các bộ tam trực tiếp ta sử dụng.
- Ví dụ: Biểu diễn bộ tam gián tiếp cho các lệnh ba địa chỉ

	statements
(0)	(14)
(1)	(15)
(2)	(16)
(3)	(17)
(4)	(18)
(5)	(19)

	op	arg1	arg2
(14)	uminus	c	
(15)	*	b	(14)
(16)	uminus	c	
(17)	*	b	(16)
(18)	+	(15)	(17)
(19)	:=	a	(18)

2. Khai báo: Khai báo trong chương trình con

- Các tên cục bộ trong chương trình con được truy xuất đến thông qua địa chỉ tương đối của nó. Gọi là offset
- Ví dụ: Xét lược đồ dịch cho việc khai báo biến

P →	{*offset:= 0* } D
D → D ; D	
D → id : T	{*enter(id.name, T,type, offset); offset := offset + T.width* }
T → integer	{ *T.type:= integer; T.width := 4* }
T → real	{ *T.type:= real ; T.width := 8* }
T → array[num] of T1	{ *T.type:= array(num.val, T1.type);* *T.width := num.val * T1.width* }
T→ ↑ T1	{ *T.type := pointer(T1.type) ; T.width := 4* }

- Trong ví dụ trên, ký hiệu chưa kết thúc P sinh ra một chuỗi các khai báo dạng id:T.
- Trước khi khai báo đầu tiên được xét thì offset = 0. Khi mỗi khai báo được tìm thấy tên và giá trị của offset hiện tại được đưa vào trong bảng ký hiệu, sau đó offset được tăng lên một khoảng bằng kích thước của đối tượng dữ liệu được cho bởi tên đó.
- Thủ tục **enter(name, type, offset)** tạo một ô trong bảng ký hiệu với tên, kiểu và địa chỉ tương đối của vùng dữ liệu của nó. Ta sử dụng các thuộc tính tổng hợp type và width để chỉ ra kiểu và kích thước (số đơn vị nhớ) của kiểu đó

2. Khai báo: Lưu trữ thông tin về tầm

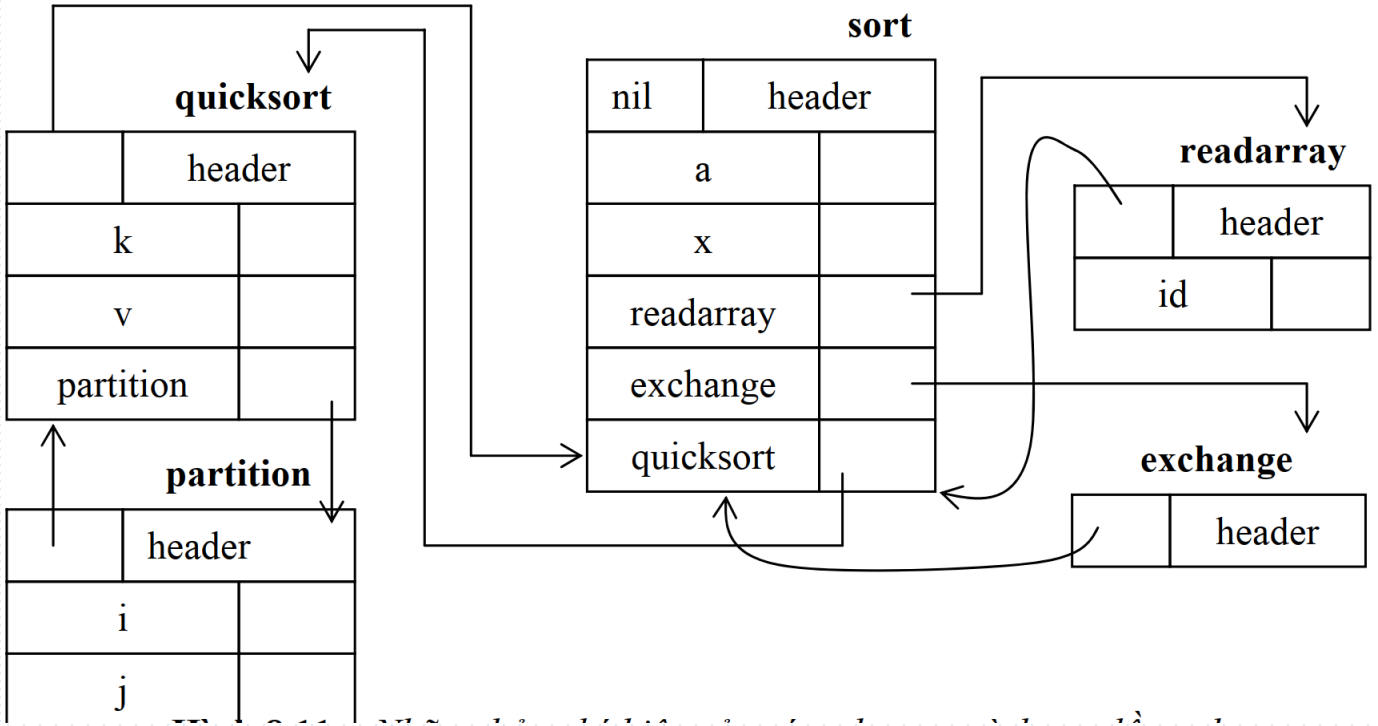
- Trong một ngôn ngữ mà chương trình con được phép khai báo lồng nhau. Khi một chương trình con được tìm thấy thì quá trình khai báo của chương trình con bao bị tạm dừng.
- Văn phạm cho sự khai báo đó là;

  P → D

  D → D ; D | id: T | proc id ; D; S
- Để đơn giản chúng ta tạo ra một bảng ký hiệu riêng cho mỗi chương trình con.
- Khi một khai báo chương trình con D → proc id D1 ; S được tạo ra thì các khai báo trong D1 được lưu trữ trong bảng ký hiệu mới.

2. Khai báo: Lưu trữ thông tin về tầm

- Ví dụ: Chương trình Sort có bốn chương trình con lồng nhau readarray, exchange, quicksort và partition. Ta có năm bảng ký hiệu tương ứng



2. Khai báo: Lưu trữ thông tin về tầm

❑ Lược đồ dịch cho chương trình con lồng nhau

$P \rightarrow M\ D$	*{ addwidth(top(tblptr), top(offset)); pop(tblptr); pop(offset) }*
$M \rightarrow \varepsilon$	*{ t:=mktable(nil); push(t, tblptr) ; push(0,offset) }*
$D \rightarrow D_1\ ;\ D_2$	
$D \rightarrow$ proc id ; N D_1 ; S	*{t:= top(tblptr); addwidth(t, top(offset)); pop(tblptr; pop(offset); enterproc(top(tblptr), id_name,t) }*
$D \rightarrow$ id : T	*{enter(top(tblptr), id_name, T.type, top(offset)); top(offset):= top(offset) + T.width }*
$N \rightarrow \varepsilon$	*{t:=mktable(top(tblptr)); push(t, tblptr); push(0,offset) }*

- Ta dùng Stack **tblptr** để giữ các con trỏ bảng ký hiệu.
- **offset** là một Stack khác để lưu trữ offset.
- Với lược đồ *A→ BC {action A}* thì tất cả các hành vi của các cây con B và C được thực hiện trước A.
- Ký hiệu chưa kết thúc N đóng vai trò tương tự như M khi một khai báo chương trình con xuất hiện. Nó dùng **mktable(top(tblptr))** để tạo ra một bảng mới.
- Tham số **top(tblptr)** cho giá trị con trỏ tới bảng lại được push vào đỉnh Stack tblptr và 0 được push vào Stack offset.

2. Khai báo: Xử lý với mẩu tin

Khai báo một mẩu tin được cho bởi luật sinh

T→ **record** D **end**

Luật dịch tương ứng

T→ **record** L D **end** { *T.type := record(top(tblptr));*
T.width := top(offset); pop(tblptr) ; pop(offset) }

L→ ε { *t:= mktable(nil); push(t,tblptr) ; push(0,offset)* }

- Sau khi từ khóa record được tìm thấy thì hành vi kết hợp với L tạo một bảng ký hiệu mới cho các tên trường.
- Các hành vi của D → id : T đưa thông tin về tên trường id vào trong bảng ký hiệu cho mẩu tin.

3. Lệnh gán: Tên trong bảng ký hiệu

PHENIKAA UNIVERSITY
Realizing your potential

Xét lược đồ dịch để sinh ra mã lệnh 3 địa chỉ cho lệnh gán

S→ id := E {*p:=lookup(id.name);*
if p <> nil then emit(p ':=' E.place) else error }

E→ E_1 + E_2 { *E.place := newtemp;*
emit(E.place ':=' E_1.place '+' E_2.place) }

E→ E_1 * E_2 { *E.place := newtemp;*
emit(E.place ':=' E_1.place '' E_2.place)* }

E→ - E_1 { *E.place := newtemp;*
emit(E.place ':=' 'unimus' E_1.place) }

E→ (E_1) { *E.place:=E_1.place)* }

E→ id { *p:=lookup(id.name);*
if p <> nil then E.place := p else error }

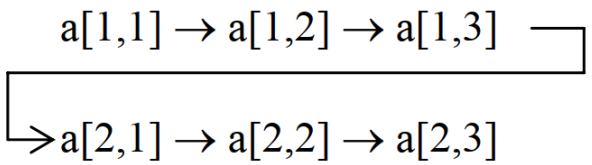
- Hàm lookup tìm trong bảng ký hiệu xem có hay không một tên được cho bởi id.name.
- Nếu có thì trả về con trỏ của ô, nếu không trả về nil.
  Xét luật sinh D → proc id ; ND1 ; S
  Hành vi kết hợp với ký hiệu chưa kết thúc N cho phép con trỏ của bảng ký hiệu cho chương trình con đang nằm trên đỉnh Stack tblptr.
- Các tên trong lệnh gán sinh ra bởi ký hiệu chưa kết thúc S sẽ được khai báo trong chương trình con này hoặc trong bao của nó.
- Khi tham khảo tới một tên thì trước hết hàm lookkup sẽ tìm xem có tên đó trong bảng ký hiệu hiện hành hay không.
- Nếu tên không được tìm thấy trong tất cả các mức thì lookup trả về nil

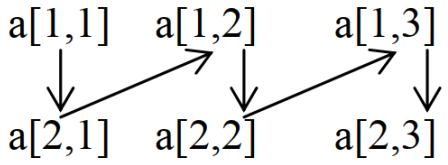
3. Lệnh gán: Địa chỉ hóa các phần tử của mảng

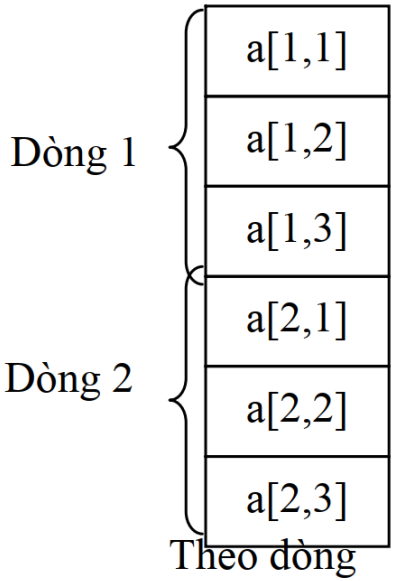
- Các phần tử của mảng có thể truy xuất nhanh nếu chúng được liền trong một khối các ô nhớ kết tiếp nhau.
- Trong mảng một chiều nếu kích thước của một phần tử là w thì địa chỉ tương đối phần tử thứ i của mảng A được tính theo công thức.
  - Địa chỉ tương đối của A[i] = base + (i-low) * w
  - Trong đó
    - low: là cận dưới tập chỉ số
    - base: là địa chỉ tương đối của ô nhớ cấp phát cho mảng tức là địa chỉ tương đối của A[low]
  - Biến đổi một chút ta được
    - Địa chỉ tương đối của A[i]= i * w + (base -low * w)
    - Trong đó: c=base - low * w có thể tính được tại thời gian dịch và lưu trong bảng ký hiệu. Do đó địa chỉ tương đối A[i] = i * w +c.

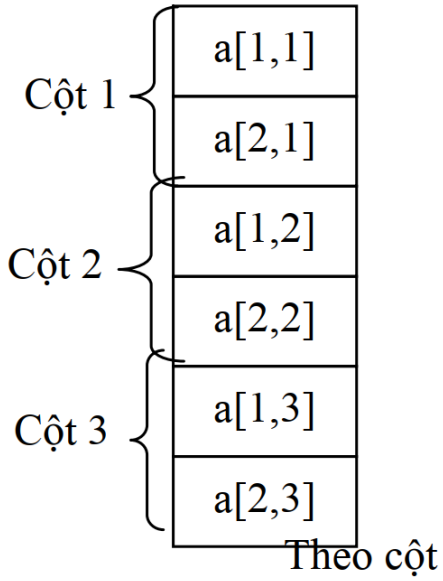
3. Lệnh gán: Địa chỉ hóa các phần tử của mảng

- Mảng hai chiều có thể xem như là một mảng theo một trong hai dạng: theo dòng (row_major) hoặc theo cột (colum_major)









3. Lệnh gán: Địa chỉ hóa các phần tử của mảng

- Trong trường hợp lưu trữ theo dòng, địa chỉ tương đối của phần tử a[i1, j2] có thể tính theo công thức
- Địa chỉ tương đối của A[i1, j2] = base + ((i1- low1) * n2 +j2 -low2) * w
  - Trong đó low1 và low2 là cận dưới của hai tập chỉ số.
  - n2 : là số các phần tử trong một dòng.

3. Lệnh gán: Biến đổi kiểu trong lệnh gán

- Giả sử chúng ta có 2 kiểu là integer và real; integer phải đổi thành real khi cần thiết.
- Ta có, các hành vi ngữ nghĩa kết hợp với luật sinh E → E1 + E2

```
E.place := newtemp
if E1.type= integer and E2.type = integer then begin
        emit(E.place ':=' E1.place 'int + ' E2.place);
        E.type:= integer;
end
else if E1.type=real and E2.type =real then begin
        emit(E.place ':=' E1.place 'real + ' E2.place);
        E.type:= real;
end
else if E1.type=integer and E2.type =real then begin
        u:=newtemp;      emit(u ':='  'intoreal' E1.place);
        emit(E.place ':=' u 'real +' E2.place);
        E.type:= real;
end
else if E1.type=real and E2.type =integer then begin
        u:=newtemp;      emit(u ':=' 'intoreal' E2.place);
        emit(E.place ':= ' E1.place 'real +' u);
        E.type:= real;
  end
  else E.type := type_error;
end
```

3. Lệnh gán: Biến đổi kiểu trong lệnh gán

- Ví dụ: Với lệnh gán x := y + i * j trong đó x,y được khai báo là real; i , j được khai báo là integer. Mã lệnh 3 địa chỉ xuất ra là:
  - t1 := i int * j
  - t3 := into real t1
  - t2 := y real + t3
  - x := t2

4. Biểu thức logic

- Biểu thức logic được sinh ra bởi văn phạm sau:

  E→ E or E | E and E | not E | (E) | id relop id | true | false

  - Trong đó or và and kết hợp trái; or có độ ưu tiên thấp nhất, kế tiếp là and và sau cùng là not
- Thông thường có 2 phương pháp chính để biểu diễn giá trị logic.
- **Phương pháp 1:** Mã hóa true và false bằng các số và việc đánh giá biểu thức được thực hiện tương tự như đối với biểu thức số học, có thể biểu diễn true bởi 1 , false bởi 0; hoặc các số khác không biểu diễn true, số không biểu diễn false...
- **Phương pháp 2:** Biểu diễn giá trị của biểu thức logic bởi một vị trí đến trong chương trình. Phương pháp này rất thuận lợi để cài đặt biểu thức logic trong các điều khiển.

4. Biểu thức logic

- Phương pháp 1: sử dụng 1 để biểu diễn true và 0 để biểu diễn false. Biểu thức được đánh giá từ trái sang phải theo cách tương tự biểu thức số học.
- Với biểu thức a or b and not c, ta có dãy lệnh 3 địa chỉ:

```
t1 := not c
t2 := b and t1
t3 := a or t2
```

- Biểu thức quan hệ a<b tương đương lệnh điều kiện if a<b then 1 else 0. dãy lênh 3 địa chỉ tương ứng là:

```
100 : if a<b goto 103
101 : t := 0
102 : goto 104
103 : t :=1
104 :
```

4. Biểu thức logic

- Ta có, lược đồ dịch để sinh ra mã lệnh 3 địa chỉ đối với biểu thức logic:

$E \rightarrow E_1$ or E_2 {*E.place:= newtemp; emit(E.place ':=' E_1.place 'or' E_2.place)* }

$E \rightarrow E_1$ and E_2 { *E.place:= newtemp; emit(E.place ':=' E_1.place 'and' E_2.place)*}

$E \rightarrow$ not E_1 {*E.place:= newtemp; emit(E.place ':=' 'not' E_1.place)* }

$E \rightarrow id_1$ relop id_2 { *E.place:= newtemp;*

emit('if' id_1.place relop.op id_2.place 'goto' nextstat +3);

emit(E.place ':=' '0'); emit('goto' nextstat +2);

emit(E.place ':=' '1') }

$E \rightarrow$ true { *E.place:= newtemp; emit(E.place ':=' '1')* }

$E \rightarrow$ false { *E.place:= newtemp; emit(E.place ':=' '0')* }

4. Biểu thức logic

- Với biểu thức a < b or c< d and e < f, nó sẽ sinh ra lệnh địa chỉ như sau:

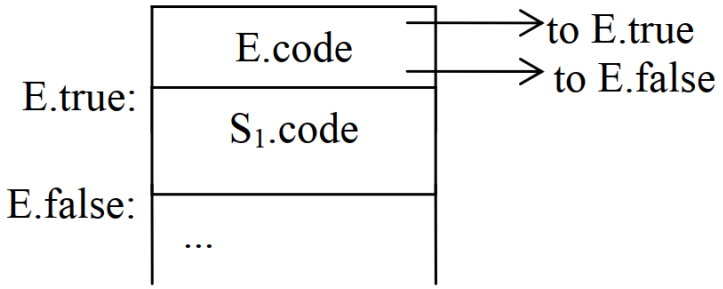
```
100   : if a<b goto 103
101   : t1 := 0
102   : goto 104
103   : t1 := 1
104   : if c<d goto 107
105   : t2 := 0
106   : goto 108
107   : t2 := 1
108   : if e<f goto 111
109   : t3 := 0
110   : goto 112
111   : t3 := 1
112   : t4 := t2 and t3
113   : t5 := t1 or t4
```

4. Biểu thức logic: Các lệnh điều khiển

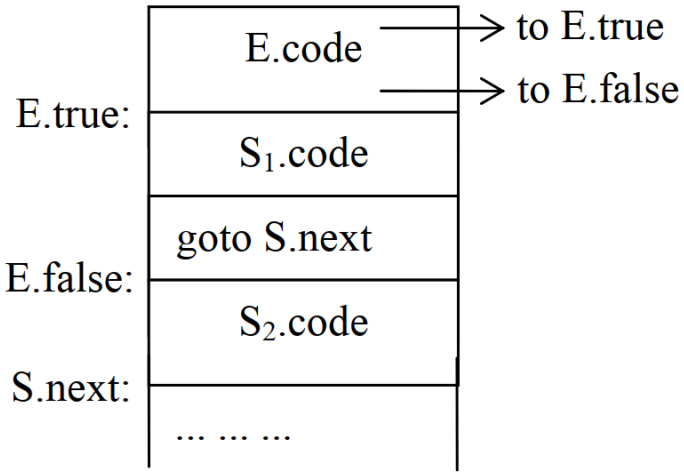
- Văn phạm: S→ if E then S1 | if E then S1 else S2 | while E do S1
- Với mỗi biểu thức logic E, chúng ta kết hợp với 2 nhãn
  - E.true : Nhãn của dòng điều khiển nếu E là true.
  - E.false : Nhãn của dòng điều khiển nếu E là false.
  - S.code : Mã lệnh 3 địa chỉ được sinh ra bởi S.
  - S.next : Là nhãn mà lệnh 3 địa chỉ đầu tiên sẽ thực hiện sau mã lệnh của S.
  - S.begin : Nhãn chỉ định lệnh đầu tiên được sinh ra cho S.

4. Biểu thức logic: Các lệnh điều khiển

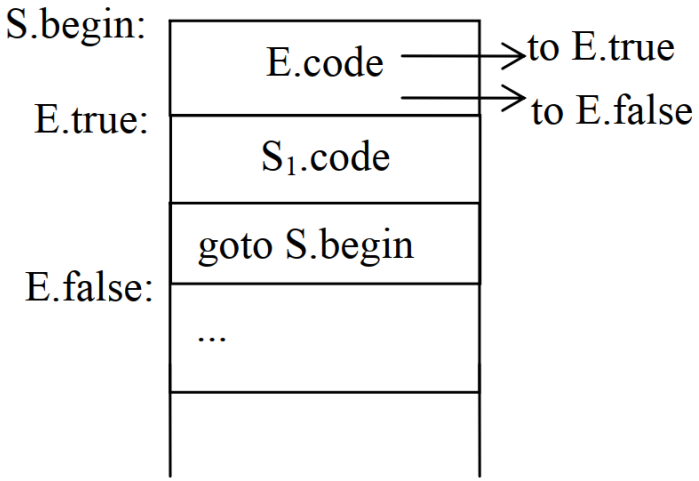
- Ví dụ: Mã lệnh của các lệnh if-then, if-then-else, và while-do



(a) if -then



(b) if -then-else



(c) while-do

4. Biểu thức logic: Các lệnh điều khiển

- Ta có định nghĩa trực tiếp cú pháp cho các lệnh điều khiển:

Luật sinh	Luật ngữ nghĩa
S→ if E then S_1	*E.true := newlabel;* *E.false := S.next;* *S_1.next := S.next;* *S.code := E.code || gen(E.true ':') || S_1.code*
S→ if E then S_1 else S_2	*E.true := newlabel;* *E.false := newlabel;* *S_1.next := S.next;* *S_2.next := S.next;* *S.code := E.code || gen(E.true ':') || S_1.code || gen('goto' S.next) || gen(E.false ':') || S_2.code*
S→ while E do S_1	*S.begin := newlabel;* *E.true := newlabel;* *E.fasle := S.next;* *S_1.next := S.begin;* *S.code:= gen(S.begin':') || E.code || gen(E.true ':') || S_1.code || gen('goto' S.begin)*

4. Biểu thức logic:
Dịch biểu thức logic trong các lệnh điều khiển

- Ta có định nghĩa trực tiếp cú pháp cho các lệnh điều khiển:
- Nếu E có dạng a<b thì mã lệnh sinh ra có dạng
  - if a<b goto E.true
  - goto E.false
- Nếu E có dạng E1 or E2. Nếu E1 là true thì E là true. Nếu E1 là flase thì phải đánh giá E2. Do đó E1.false là nhãn của lệnh đầu tiên của E2. E sẽ true hay false phụ thuộc vào E2 là true hay false.
- Tương tự cho E1 and E2.
- Nếu E có dạng not E1 thì E1 là true thì E là false và ngược lại.

4. Biểu thức logic:
Biểu thức logic và biểu thức số học

- Trong thực tế biểu thức logic thường chứa những biểu thức số học như (a+b) < c.
- Trong các ngôn ngữ mà false có giá trị số là 0 và true có giá trị số là 1 thì (a<b) + (b<a) có thể được xem như là một biểu thức số học có giá trị 0 nếu a = b và có giá trị 1 nếu a <> b.
- Phương pháp biểu diễn biểu thức logic bằng mã lệnh nhảy có thể vẫn còn được sử dụng
- Xét văn phạm E → E+ E | E and E | E relop E | id
  - Trong đú, E and E đũi hỏi hai đối số phải l logic. Trong khi + v relop cú cỏc đối số l biểu thức logic hoặc/v số học.
  - Để sinh mã lệnh trong trường hợp này, chúng ta dùng thuộc tính tổng hợp E.type có thể là arith hoặc bool.
  - E sẽ có các thuộc tính kế thừa E.true và E.false đối với biểu thức số học

4. Biểu thức logic: Biểu thức logic và biểu thức số học

- Ta có luật ngữ nghĩa kết hợp với E → E1 + E2 như sau

E.type := arith;

if E_1*.type = arith* **and** E_2*.type = arith* **then begin**

/ phép cộng số học bình thường */*

E.place := newtemp;

E.code := E_1.code || E_2.code || gen(E.place ':=' E_1.place '+' E_2.place)

end

else if E_1*.type = arith* **and** E_2*.type = bool* **then begin**

E.place := newtemp;

E_2.true := newlabel;

E_2.false := newlabel;

E.code := E_1.code || E_2.code || gen(E_2.true ':' E.place ':= ' E_1.place +1) ||

gen('goto' nextstat +1) || gen(E_2.false ':' E.place ':= ' E_1.place)

else if ...

4. Biểu thức logic:
Biểu thức logic và biểu thức số học

- Trong trường hợp nếu có biểu thức logic nào có biểu thức số học, chúng ta sinh mã lệnh cho E1, E2 bởi các lệnh

  E2.true : E.place := E1.place +1

  goto nextstat +1

  E2.false : E.place := E1.place

5. Lệnh Case

- Lệnh CASE hoặc SWITCH thường được sử dụng trong các ngôn ngữ lập trình
- Cú pháp của lệnh SWITCH/ CASE

```
SWITCH E
begin
        case V1 : S1
        case V2 : S2
        ....
        case Vn-1 : Sn-1
        default: Sn
end
```

5. Lệnh Case: dịch trực tiếp cú pháp Case

- 1. Đánh giá biểu thức.
- 2. Tùy một giá trị trong danh sách các case bằng giá trị của biểu thức. Nếu không tìm thấy thì giá trị default của biểu thức được xác định.
- 3. Thực hiện các lệnh kết hợp với giá trị tìm được để cài đặt

5. Lệnh Case: dịch trực tiếp cú pháp Case

- Ta có phương pháp cài đặt như sau

```
        mã lệnh để đánh giá biểu thức E vào t
        goto test
L1 :    mã lệnh của S1
        goto next
L2:     mã lệnh của S2
        goto next
.....
Ln-1 :  mã lệnh của Sn-1
        goto next
Ln :    mã lệnh của Sn
        goto next
test :  if t=V1 goto L1
        if t=V2 goto L2
        .. .. .. ..
        if t=Vn-1 goto Ln-1
        else  goto Ln
next:
```

5. Lệnh Case: dịch trực tiếp cú pháp Case

PHENIKAA UNIVERSITY
Realizing your potential

- Một phương pháp khác để cài đặt lệnh SWITCH là

mã lệnh để đánh giá biểu thức E vào t
if t <> V1 goto L1
mã lệnh của S1
goto next
L_1 : if t <> V2 goto L_2
mã lệnh của S2
goto next
L_2:.....
L_{n-2} : if t <> V_{n-1} goto L_{n-1}
mã lệnh của Sn-1
goto next
Ln-1 : mã lệnh của Sn
next:

6. Bài tập

❑ 1. Dịch biểu thức : a * - (b + c) thành các dạng:

a) Cây cú pháp.

b) Ký pháp hậu tố.

c) Mã lệnh máy 3 - địa chỉ.

❑ 2. Trình bày cấu trúc lưu trữ biểu thức - (a + b) * (c + d) + (a + b + c) ở các dạng:

a) Bộ tứ .

b) Bộ tam.

c) Bộ tam gián tiếp

6. Bài tập

- 3. Sinh mã trung gian (dạng mã máy 3 - địa chỉ) cho các biểu thức C đơn giản sau:

a) x = 1

b) x = y

c) x = x + 1

d) x = a + b * c

e) x = a / (b + c) - d * (e + f)

- 4. Sinh mã trung gian (dạng mã máy 3 - địa chỉ) cho các biểu thức C sau:

a) x = a [i] + 11

b) a [i] = b [c[j]]

c) a [i][j] = b [i][k] * c [k][j]

d) a[i] = a[i] + b[j]

e) a[i] + = b[j]