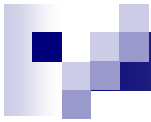




# Bài 11

## Sinh mã trung gian



# Nội dung

- Mã ba địa chỉ
- Sinh mã cho lệnh gán
- Sinh mã cho các biểu thức logic
- Sinh mã cho các cấu trúc lập trình



# Mã trung gian

- Một chương trình với mã nguồn được chuyển sang chương trình tương đương trong ngôn ngữ trung gian bằng bộ sinh mã trung gian.
- Ngôn ngữ trung gian được người thiết kế trình biên dịch quyết định, có thể là:
  - Cây cú pháp
  - Ký pháp Ba Lan sau (hậu tố)
  - Mã 3 địa chỉ ...



# Mã trung gian

- Được sản sinh dưới dạng một chương trình cho một máy trừu tượng
- Mã trung gian thường dùng : mã ba địa chỉ, tương tự mã assembly
- Chương trình là một dãy các lệnh. Mỗi lệnh gồm tối đa 3 định danh.
- Tồn tại nhiều nhất một toán tử ở vế phải cộng thêm một toán tử gán
- x,y,z là các địa chỉ , tức là tên, hằng hay các tên trung gian do trình biên dịch sinh ra
  - Tên trung gian phải được sinh để thực hiện các phép toán trung gian
  - Các địa chỉ được thực hiện thường là con trỏ tới lỗi vào của nó trong bảng ký hiệu



# Mã trung gian của $t_2 = x + y * z$

- $t_1 := y * z$
- $t_2 := x + t_1$



# Tập mã lệnh ba địa chỉ điển hình

- Mã 3 địa chỉ tương tự mã Assembly: lệnh có thể có nhãn, có những lệnh chuyển điều khiển cho các cấu trúc lập trình.
  1. *Lệnh gán  $x := y \text{ op } z$ .*
  2. *Lệnh gán với phép toán 1 ngôi :  $x := \text{op } y$ .*
  3. *Lệnh sao chép:  $x := y$ .*
  4. *Lệnh nhảy không điều kiện:  $\text{goto } L$ ,  $L$  là nhãn của một lệnh*
  5. *Lệnh nhảy có điều kiện  $x \text{ relop } y \text{ goto } L$ .*



# Tập mã lệnh ba địa chỉ điển hình

6. Lời gọi thủ tục param x và call p,n để gọi thủ tục p với n tham số . Return y là giá trị thủ tục trả về

param  $x_1$

param  $x_2$

...

param  $x_n$

Call p,n

7. Lệnh gán có chỉ số  $x:=y[i]$  hay  $x[i]:=y$



# Sinh mã trực tiếp từ ĐNTCP

- Thuộc tính tổng hợp S.code biểu diễn mã ba địa chỉ của lệnh
- Các tên trung gian được sinh ra cho các tính toán trung gian
- Các biểu thức được liên hệ với hai thuộc tính tổng hợp
  - E.place chứa địa chỉ chứa giá trị của E
  - E.code mã ba địa chỉ để đánh giá E
- Hàm newtemp sinh ra các tên trung gian t1, t2, . .
- Hàm gen sinh mã ba địa chỉ
- Trong thực tế, code được gửi vào file thay cho thuộc tính code



# Dịch trực tiếp cú pháp thành mã 3 địa chỉ

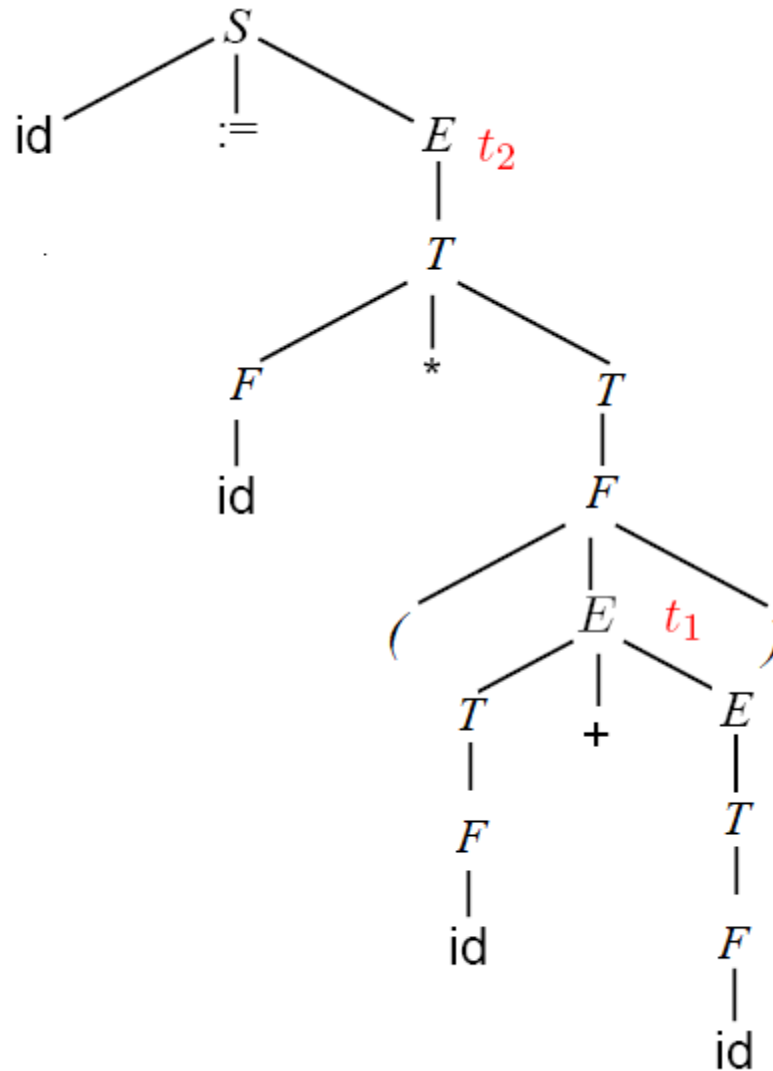
Sản xuất	Quy tắc ngữ nghĩa
$S \rightarrow id := E$	$\{S.code = E.code \parallel gen(id.place := E.place)\}$
$E \rightarrow T + E_1$	$\{E.place = newtemp ; E.code = T.code \parallel E_1.code \parallel gen(E.place := T.place + E_1.place) \}$
$E \rightarrow T$	$\{E.place = T.place ; E.code = T.code\}$
$T \rightarrow F * T_1$	$\{T.place = newtemp ; T.code = F.code \parallel T_1.code \parallel gen(T.place := F.place * T_1.place) \}$
$T \rightarrow F$	$\{T.place = F.place ; T.code = F.code\}$
$F \rightarrow (E)$	$\{F.place = E.place ; F.code = E.code\}$
$F \rightarrow id$	$\{F.place = id.place ; F.code = " \}$

Hàm ***newtemp*** trả về một dãy các tên khác nhau  $t_1, t_2, \dots$  cho lời gọi kế tiếp.

$E.place$ : là tên sẽ giữ giá trị của  $E$

$E.code$ : là dãy các câu lệnh 3 địa chỉ dùng để ước lượng  $E$

# Mã cho lệnh gán $a := b * (c + d)$



$t_1 := c + d$

$t_2 := b * t_1$

$a := t_2$

# Cài đặt câu lệnh 3 địa chỉ

## Bộ bốn (Quadruples)

$t_1 := c + d$

$t_2 := b * t_1$

$a := t_2$

	<i>op</i>	<i>arg1</i>	<i>arg2</i>	<i>result</i>
(0)	+	c	d	$t_1$
(1)	*	b	$t_1$	$t_2$
(2)	:=	$t_2$		a

**Tên tạm phải được thêm vào bảng kí hiệu khi chúng được tạo ra.**

## Cài đặt câu lệnh 3 địa chỉ

### ■ Bộ ba (Triples)

$t_1 := c + d$

$t_2 := b * t_1$

$a := t_2$

	<i>op</i>	<i>arg1</i>	<i>arg2</i>
(0)	+	c	d
(1)	*	b	(0)
(2)	assign	a	(1)

**Tên tạm không được thêm vào trong bảng kí hiệu.**

## Các dạng khác của câu lệnh 3 địa chỉ

- Ví dụ:  
 $x[i] := y$                        $x := y[i]$
- Sử dụng 2 cấu trúc bộ ba

	<i>op</i>	<i>arg1</i>	<i>arg2</i>
(0)	[ ]	x	i
(1)	:=	(0)	y

	<i>op</i>	<i>arg1</i>	<i>arg2</i>
(0)	[ ]	y	i
(1)	:=	x	(0)

# Cài đặt câu lệnh 3 địa chỉ

- Bộ 3 gián tiếp: sử dụng một danh sách các con trỏ các bộ 3

	<i>op</i>		<i>op</i>	<i>arg1</i>	<i>arg2</i>
(0)	(14)	(14)	uminus	c	
(1)	(15)	(15)	*	b	(14)
(2)	(16)	(16)	uminus	c	
(3)	(17)	(17)	*	b	(16)
(4)	(18)	(18)	+	(15)	(17)
(5)	(19)	(19)	assign	a	(18)

# Sinh mã cho khai báo

Sử dụng biến toàn cục *offset*.

Các tên cục bộ trong chương trình con được truy xuất thông qua địa chỉ tương đối *offset*.

<u>Sản xuất</u>	<u>Quy tắc ngữ nghĩa</u>
$P \rightarrow M D$	$\{ \}$
$M \rightarrow \varepsilon$	$\{offset:=0 \}$
$D \rightarrow D; D$	
$D \rightarrow id : T$	$\{ enter(id.name, T.type, offset)$ $offset:=offset + T.width \}$
$T \rightarrow integer$	$\{T.type = integer; T.width = 4 \}$
$T \rightarrow char$	$\{T.type = char; T.width = 1 \}$
$T \rightarrow array [ num ] of T_1$	$\{T.type=array(1..num.val, T_1.type)$ $T.width = num.val * T_1.width \}$

# Lưu trữ thông tin về phạm vi

- Trong một ngôn ngữ mà chương trình con được phép khai báo lồng nhau, mỗi khi tìm thấy một CTC thì quá trình khai báo của chương trình bao nó bị tạm dừng.

- Văn phạm của khai báo này:

$P \rightarrow D$

$D \rightarrow D; D \mid id : T \mid \text{proc } id ; D ; S$

- Khi một khai báo chương trình con  $D \rightarrow \text{proc } id D1; S$  được tạo ra thì các khai báo trong  $D1$  được lưu trong bảng kí hiệu mới.



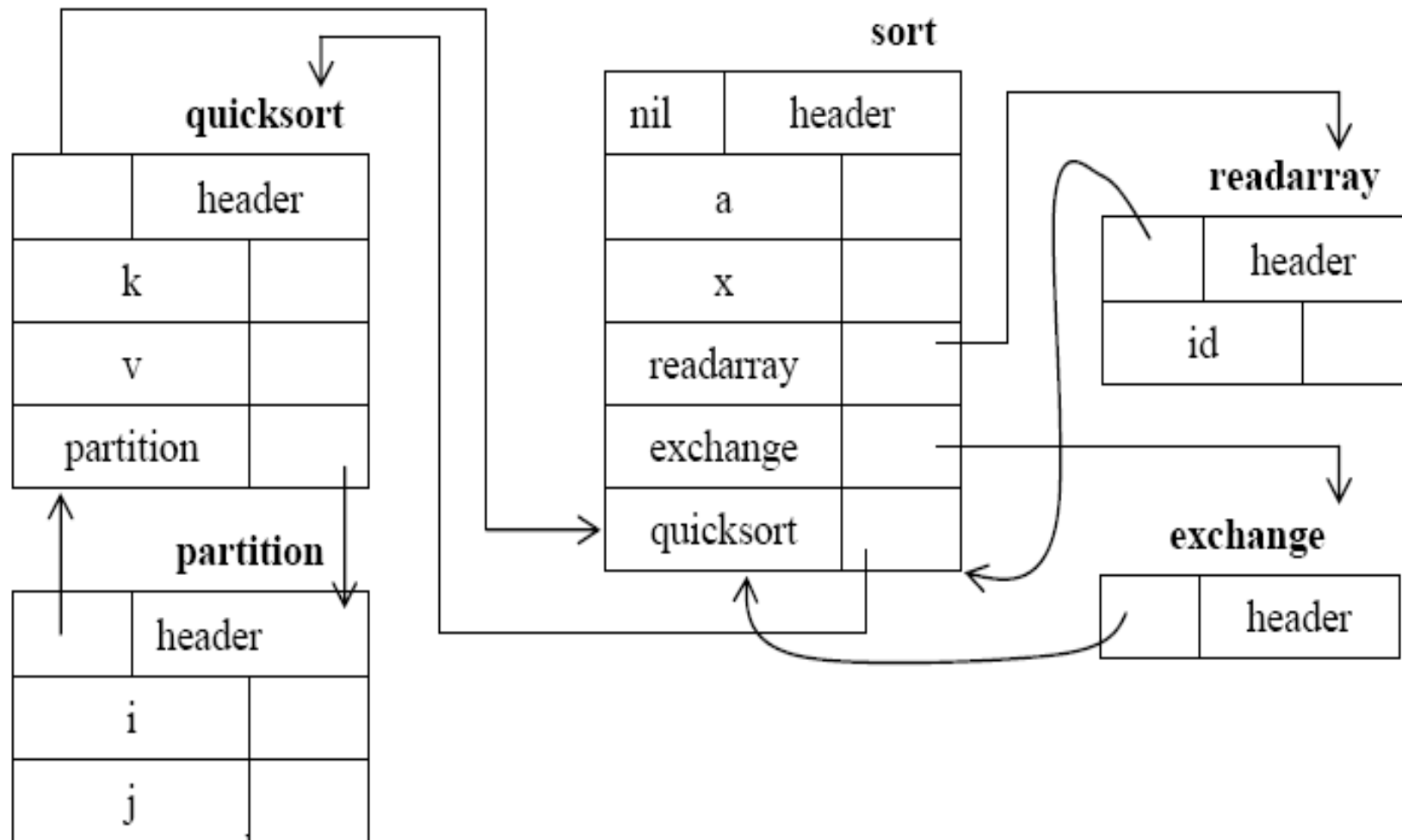


# Khai báo chương trình con lồng nhau

- Ví dụ chương trình:

- 1) Program sort;
- 2) *Var a: array[0..10] of integer;*
- 3) *x: integer;*
- 4) Procedure readarray;
- 5) *Var i: integer;*
- 6) *Begin ...a... end {readarray};*
- 7) Procedure exchange(i, j: integer);
- 8) *Begin {exchange} end;*
- 9) Procedure quicksort(m, n: integer);
- 10) *Var k, v: integer;*
- 11) Function partition(y,z: integer): integer;
- 12) *Begin ..a..v..exchange(i,j) end; {partition}*
- 13) *Begin ... end; {quicksort}*
- 14) *Begin ... end; {sort}*

# Năm hằng kí hiệu của Sort





# Các thủ tục trong tập quy tắc ngữ nghĩa S

**mktable(previous)** – tạo một bảng kí hiệu mới, bảng này có *previous* chỉ đến bảng cha của bảng kí hiệu mới này.

**enter(table,name,type,offset)** – tạo ra một ô mới có tên *name* trong bảng kí hiệu được chỉ ra bởi *table* và đặt kiểu *type*, địa chỉ tương đối *offset* vào các trường bên trong ô đó.

**enterproc(table,name,newtable)** – tạo ra một ô mới cho tên chương trình con vào *table*, *newtable* trở tới bảng kí hiệu của chương trình con này.

**addwidth(table,width)** – ghi tổng kích thước của tất cả các ô trong bảng kí hiệu vào header của bảng đó.

Xử lý các khai báo trong những chương trình con lồng nhau

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

- **tblptr** – để giữ con trỏ bảng kí hiệu.
- **offset** – lưu trữ địa chỉ offset hiện tại của bảng kí hiệu trong **tblptr**.

## Xử lý các khai báo trong những chương trình con lồng nhau

- Với sản xuất  $A \rightarrow BC \{action_A\}$  thì các hoạt động trong cây con B, C được thực hiện trước A.
- Sản xuất  $M \rightarrow \varepsilon$  khởi tạo stack **tblptr** với một bảng kí hiệu cho phạm vi ngoài cùng (chương trình sort) bằng lệnh **mktable(nil)** đồng thời đặt offset = 0.
- N đóng vai trò tương tự M khi một khai báo chương trình con xuất hiện, nó dùng lệnh **mktable(top(tblptr))** để tạo ra một bảng mới, tham số **top(tblptr)** cho giá trị con trỏ tới bảng lại được đẩy vào đỉnh stack tblptr và 0 được đẩy vào stack offset.
- Với mỗi khai báo **id: T** một ô mới được tạo ra cho id trong bảng kí hiệu hiện hành, stack tblptr không đổi, giá trị top(offset) được tăng lên bởi T.width.
- Khi  $D \rightarrow \text{proc id ; } N D_1 \text{ ; } S$  diễn ra thì kích thước của tất cả các đối tượng dữ liệu khai báo trong  $D_1$  sẽ nằm trên đỉnh stack offset. Nó được lưu trữ bằng cách dùng Addwidth, các stack tblptr và offset bị đẩy và chúng ta thao tác trên các khai báo của chương trình con.

# ĐNTCP để sinh ra mã lệnh 3 địa chỉ cho lệnh gán

$S \rightarrow id := E$        $\{ p := \text{lookup}(id.name);$   
                                  $\text{if } p \neq \text{nil then emit}(p := E.place) \text{ else error} \}$

$E \rightarrow E_1 + E_2$        $\{ E.place := \text{newtemp};$   
                                  $\text{emit}(E.place := E_1.place + E_2.place) \}$

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

$E \rightarrow - E_1$        $\{ E.place := \text{newtemp};$   
                                  $\text{emit}(E.place := \text{'unimus' } E_1.place) \}$

$E \rightarrow ( E_1 )$        $\{ E.place := E_1.place \}$

$E \rightarrow id$        $\{ p := \text{lookup}(id.name);$   
                                  $\text{if } p \neq \text{nil then } E.place := p \text{ else error} \}$



# Tên trong bảng kí hiệu

- Hàm lookup sẽ tìm trong bảng kí hiệu xem có hay không một tên được cho bởi *id.name*. Nếu có thì trả về con trỏ của ô, nếu không thì trả về nil.
- Thủ tục emit để đưa mã 3 địa chỉ vào một tập tin output chứ không xây dựng thuộc tính code cho các kí hiệu chưa kết thúc như gen. Quá trình dịch thực hiện bằng cách đưa ra một tập tin output nếu thuộc tính code của kí hiệu không kết thúc trong vế trái sản xuất được tạo ra bằng cách nối thuộc tính code của kí hiệu không kết thúc trong vế phải theo đúng thứ tự xuất hiện của các kí hiệu chưa kết thúc ở vế phải.



# Tên trong bảng kí hiệu

- Xét sản xuất  $D \rightarrow \text{proc id}; ND_1; S$
- Các tên trong lệnh gán sinh ra bởi kí hiệu không kết thúc S sẽ được khai báo trong chương trình con này hoặc trong chương trình chứa nó.
- Khi khai báo tới một tên thì trước hết hàm lookup sẽ tìm xem tên đó có trong bảng kí hiệu hiện hành hay không, nếu không thì dùng con trỏ trong header của bảng để tìm bảng kí hiệu bao nó và tìm trong đó, nếu không tìm thấy trong tất cả các mức thì lookup trả về nil.



# Địa chỉ hóa các phần tử của mảng

- Các phần tử của mảng có thể truy xuất nhanh nếu chúng được lưu trữ trong một khối ô nhớ kế 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:
- $A[i] = \text{base} + (i - \text{low}) * w$
- Trong đó:
  - Low: cận dưới tập chỉ số
  - Base: địa chỉ tương đối của ô nhớ cấp phát cho mảng (địa chỉ tương đối của  $A[\text{low}]$ )
- Tương đương  $A[i] = i * w + (\text{base} - \text{low} * w)$
- Trong đó:
  - $c = \text{base} - \text{low} * w$  có thể được tính tại thời gian dịch và lưu trong bảng kí hiệu
- $\Rightarrow A[i] = i * w + c$

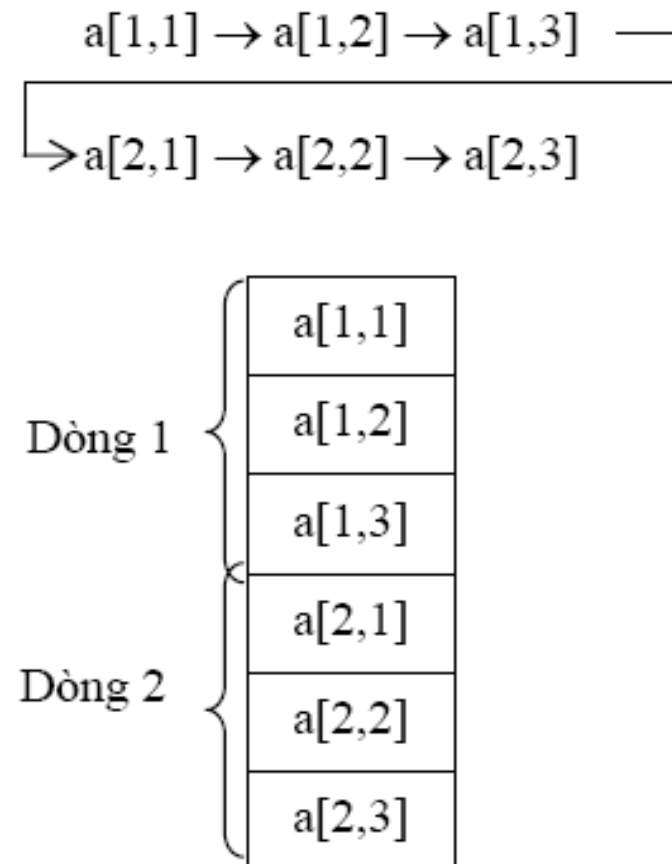
# Địa chỉ hóa các phần tử của mảng 2 chiều

## ■ Theo dòng

Địa chỉ tương đối của  $A[i_1, i_2] =$   
 $\text{base} + ((i_1 - \text{low}_1) * n_2 + i_2 - \text{low}_2) * w$

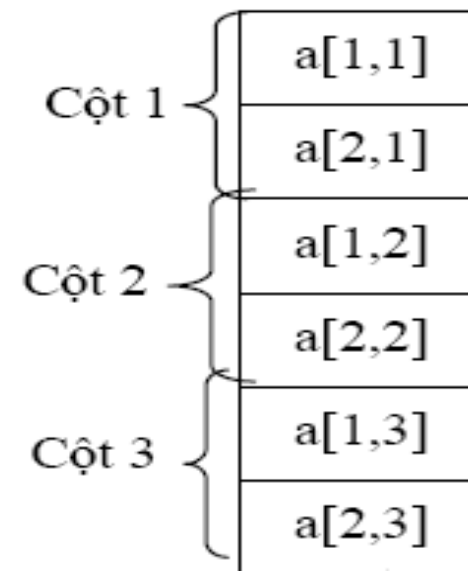
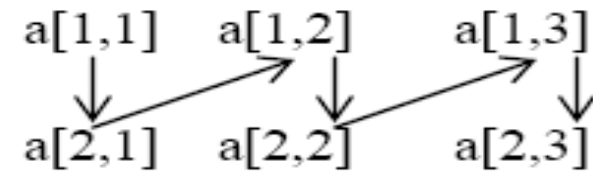
$\text{low}_1, \text{low}_2$ : cận dưới cho  $i_1$  và  $i_2$


$n_2$ : số lượng các giá trị mà  $i_2$  có thể nhận. Nếu  $\text{high}_2$  là cận trên của  $i_2$  thì  $n_2 = \text{high}_2 - \text{low}_2 + 1$



# Địa chỉ hóa các phần tử của mảng 2 chiều

□ Theo cột





# Sinh mã biểu thức Logic

- Biểu thức logic được sinh bởi văn phạm sau:  
 $E \rightarrow E \text{ or } E \mid E \text{ and } E \mid \text{not } E \mid (E) \mid \text{id relop id} \mid \text{true} \mid \text{false}$
- Trong đó:
  - Or và And kết hợp trái
  - Or có độ ưu tiên thấp nhất tiếp theo là And, và Not

# Biểu diễn bằng số

- Mã hóa true và false bằng các số và ước lượng một biểu thức boole tương tự như đối với biểu thức số học
- Có thể biểu diễn true là 1; false là 0
- Hoặc các số khác 0 là true, 0 là false

Ví dụ: biểu thức  $a \text{ or } b \text{ and not } c$

- Mã 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. Mã 3 địa chỉ tương ứng:  
100: if  $a < b$  goto 103  
101: t:=0  
102: goto 104  
103: t:= 1  
104:

# ĐNTCP dùng số để biểu diễn các giá trị logic

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

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

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

$E \rightarrow id_1 \text{ relop } id_2 \quad \{ E.place := newtemp;$

Nextstat cho biết  
nhãn của câu  
lệnh 3 địa chỉ  
tiếp theo.

Emit: đặt câu lệnh  
3 địa chỉ vào tập  
tin, emit làm tăng  
nextstat sau khi  
thực hiện

$(\text{if } id_1.place \text{ relop.op } id_2.place \text{ 'goto' nextstat} + 3);$

$(E.place := '0'); \quad \text{emit}(\text{'goto' nextstat} + 2);$

$(E.place := '1') \}$

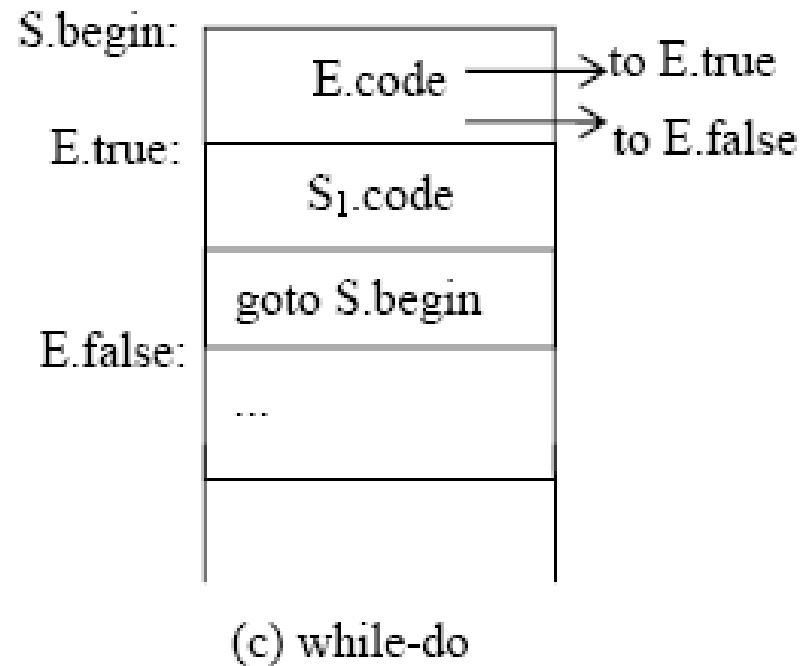
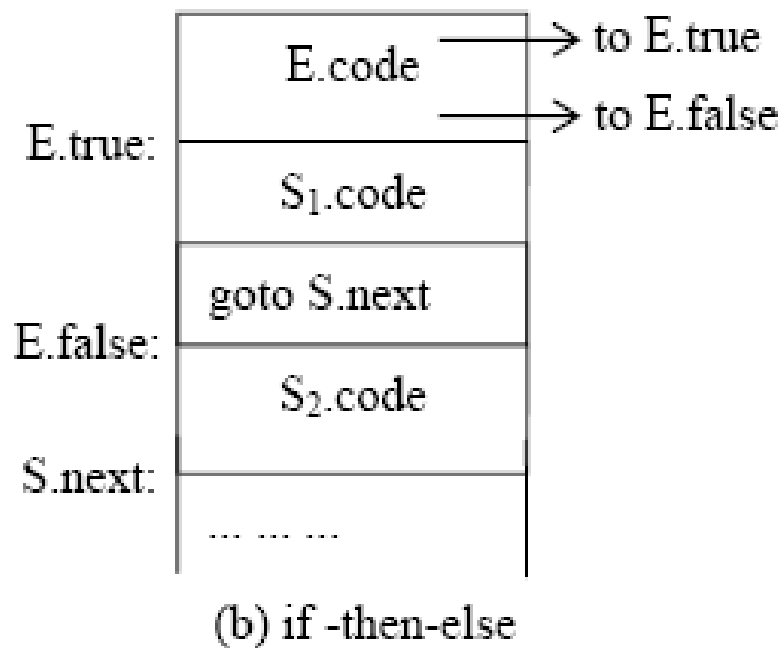
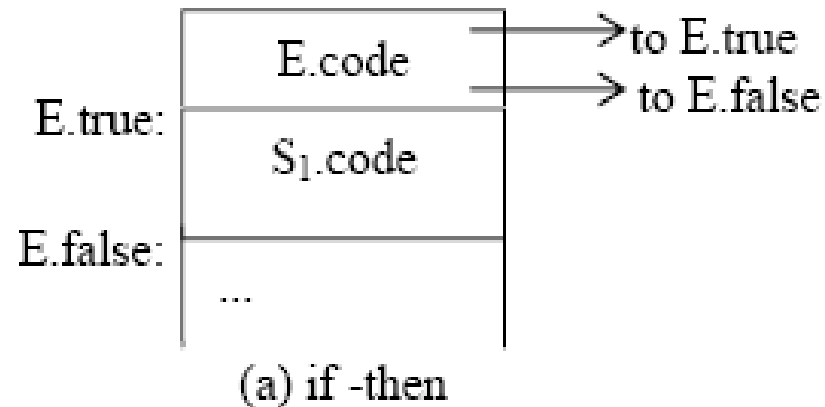
$E \rightarrow \text{true} \quad \{ E.place := newtemp; \text{ emit}(E.place := '1') \}$

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

# Sinh mã cho các cấu trúc lập trình

- Biểu diễn các giá trị của biểu thức Boole bằng biểu thức đã đến được trong một chương trình.
- Ví dụ: cho câu lệnh sau
- $S \rightarrow \text{if } E \text{ then } S_1 \mid \text{if } E \text{ then } S_1 \text{ else } S_2 \mid \text{while } E \text{ do } S_1$
- Với mỗi biểu thức 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 mã lệnh 3 địa chỉ đầu tiên sẽ thực hiện sau mã lệnh của S
  - S.begin: nhãn địa chỉ lệnh đầu tiên được sinh ra cho S

## Mã lệnh của các lệnh if-then, if-then-else, while-do





# ĐNTCP cho các cấu trúc lập trình

PRODUCTION	SEMANTIC RULES
$P \rightarrow S$	$S.next = newlabel()$ $P.code = S.code \parallel label(S.next)$
$S \rightarrow \text{assign}$	$S.code = \text{assign.code}$
$S \rightarrow \text{if } B \text{ then } S_1$	$B.true = newlabel()$ $B.false = S_1.next = S.next$ $S.code = B.code \parallel label(B.true) \parallel S_1.code$
$S \rightarrow \text{if } B \text{ then } S_1 \text{ else } S_2$	$B.true = newlabel()$ $B.false = newlabel()$ $S_1.next = S_2.next = S.next$ $S.code = B.code$ $\parallel label(B.true) \parallel S_1.code$ $\parallel gen('goto' S.next)$ $\parallel label(B.false) \parallel S_2.code$
$S \rightarrow \text{while } B \text{ do } S_1$	$begin = newlabel()$ $B.true = newlabel()$ $B.false = S.next$ $S_1.next = begin$ $S.code = label(begin) \parallel B.code$ $\parallel label(B.true) \parallel S_1.code$ $\parallel gen('goto' begin)$
$S \rightarrow S_1 S_2$	$S_1.next = newlabel()$ $S_2.next = S.next$ $S.code = S_1.code \parallel label(S_1.next) \parallel S_2.code$



## Dịch biểu thức logic trong các cấu trúc lập trình

- Nếu E có dạng:  $a < b$  thì mã lệnh sinh ra có dạng

If  $a < b$  then goto E.true else goto E.false

- Nếu E có dạng: E1 or E2 thì
  - Nếu E1 là true thì E cũng là true
  - Nếu E1 là false thì phải đánh giá E2; E sẽ là true hay false phụ thuộc E2
- Tương tự với E1 and E2



Dịch  
biểu  
thức  
logic  
trong  
các  
cấu  
trúc  
lập  
trình

SẢN XUẤT	QUY TẮC NGŨ NGHĨA
$E \rightarrow E_1 \text{ or } E_2$	$E_1.true := E.true;$ $E_1.false := newlabel;$ $E_2.true := E.true;$ $E_2.false := E.false;$ $E.code := E_1.code \parallel gen(E.false ':') \parallel E_2.code$
$E \rightarrow E_1 \text{ and } E_2$	$E_1.true := newlabel;$ $E_1.false := E.false;$ $E_2.true := E.true;$ $E_2.false := E.false;$ $E.code := E_1.code \parallel gen(E.true ':') \parallel E_2.code$
$E \rightarrow \text{not } E_1$	$E_1.true := E.false;$ $E_1.false := E.true;$ $E.code := E_1.code$
$E \rightarrow (E_1)$	$E_1.true := E.true;$ $E_1.false := E.false;$ $E.code := E_1.code$
$E \rightarrow id_1 \text{ relop } id_2$	$E.code := gen('if' id_1.place \text{ relop } op id_2.place$ $\quad 'goto' E.true) \parallel gen('goto' E.false)$
$E \rightarrow \text{true}$	$E.code := gen('goto' E.true)$
$E \rightarrow \text{false}$	$E.code := gen('goto' E.false)$

# Biểu thức logic ở dạng hỗn hợp

- Thực tế, các biểu thức logic thường chứa các biểu thức số học như trong  $(a+b)<c$

Xét văn phạm  $E \rightarrow E + E \mid E \text{ and } E \mid E \text{ relop } E \mid id$

Trong đó,  $E \text{ and } E$  đòi hỏi hai đối số phải là logic. Trong khi  $+$  và  $\text{relop}$  có các đối số là biểu thức logic hoặc/và số học.

Để sinh mã lệnh trong trường hợp này, chúng ta dùng thuộc tính tổng hợp  $E.\text{type}$  có thể là  $\text{arith}$  hoặc  $\text{bool}$ .  $E$  sẽ có các thuộc tính kế thừa  $E.\text{true}$  và  $E.\text{false}$  đối với biểu thức số học.

# Biểu thức logic ở dạng hỗn hợp

QT ngữ nghĩa kết hợp với  $E \rightarrow E_1 + E_2$

*E.type := arith;*

**if** *E<sub>1</sub>.type = arith and E<sub>2</sub>.type = arith* **then begin**

*/\* phép cộng số học bình thường \*/*

*E.place := newtemp;*

*E.code := E<sub>1</sub>.code || E<sub>2</sub>.code || gen(E.place := ' E<sub>1</sub>.place +' E<sub>2</sub>.place)*

**end**

**else if** *E<sub>1</sub>.type = arith and E<sub>2</sub>.type = bool* **then begin**

*E.place := newtemp;*


*E<sub>2</sub>.true := newlabel;*

*E<sub>2</sub>.false := newlabel;*

*E.code := E<sub>1</sub>.code || E<sub>2</sub>.code || gen(E<sub>2</sub>.true := ' E.place := ' E<sub>1</sub>.place + 1) ||*

*gen('goto' nextstat + 1) || gen(E<sub>2</sub>.false := ' E.place := ' E<sub>1</sub>.place)*

**else if ...**



# Biểu thức logic ở dạng hỗn hợp

nếu có biểu thức logic nào có biểu thức số học, sinh mã lệnh cho E1, E2 bởi

$E_2.\text{true} : E.\text{place} := E_1.\text{place} + 1$

$\text{goto nextstat} + 1$

$E_2.\text{false} : E.\text{place} := E_1.\text{place}$



# Ví dụ

- Sinh mã trung gian cho lệnh sau:

while  $a < b$  do

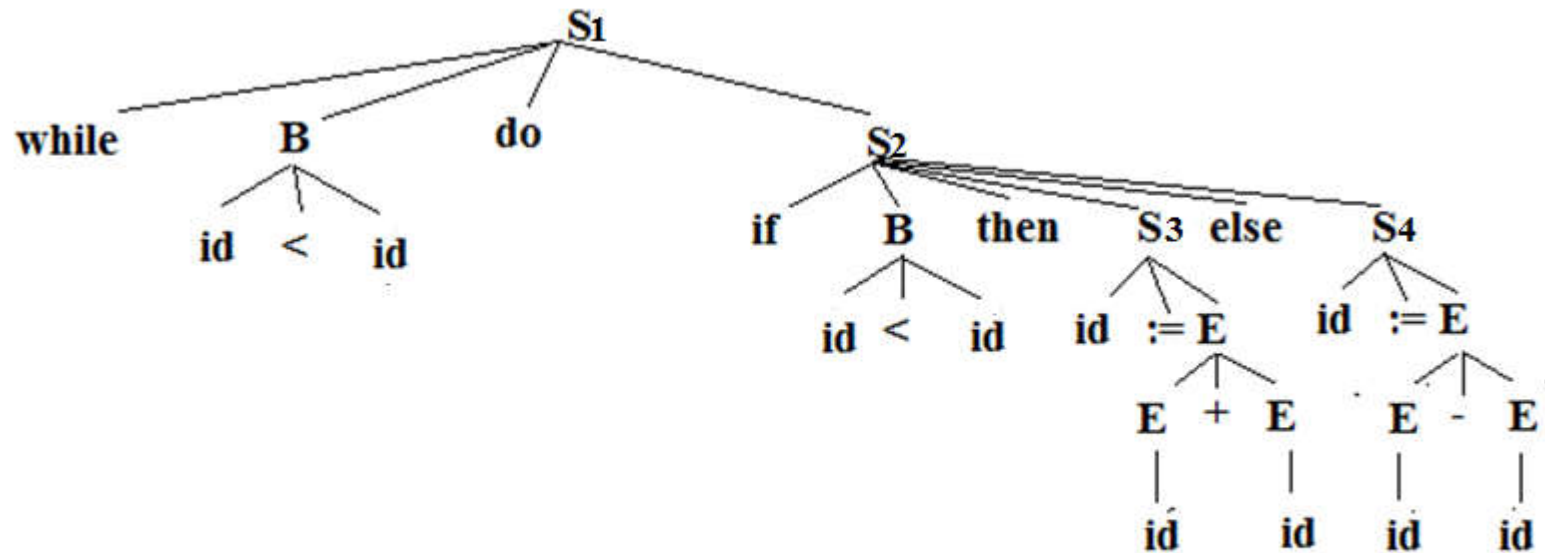
if  $c < d$  then

$x := y + z$

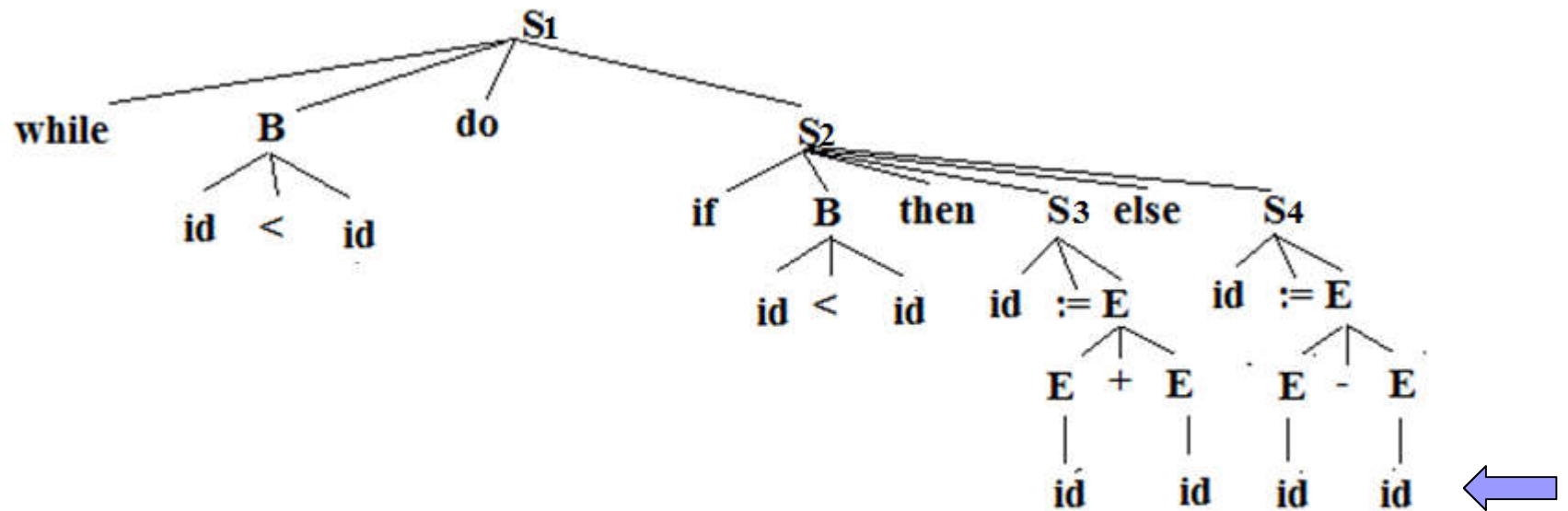
else

$x := y - z$

# Cây phân tích cú pháp của lệnh

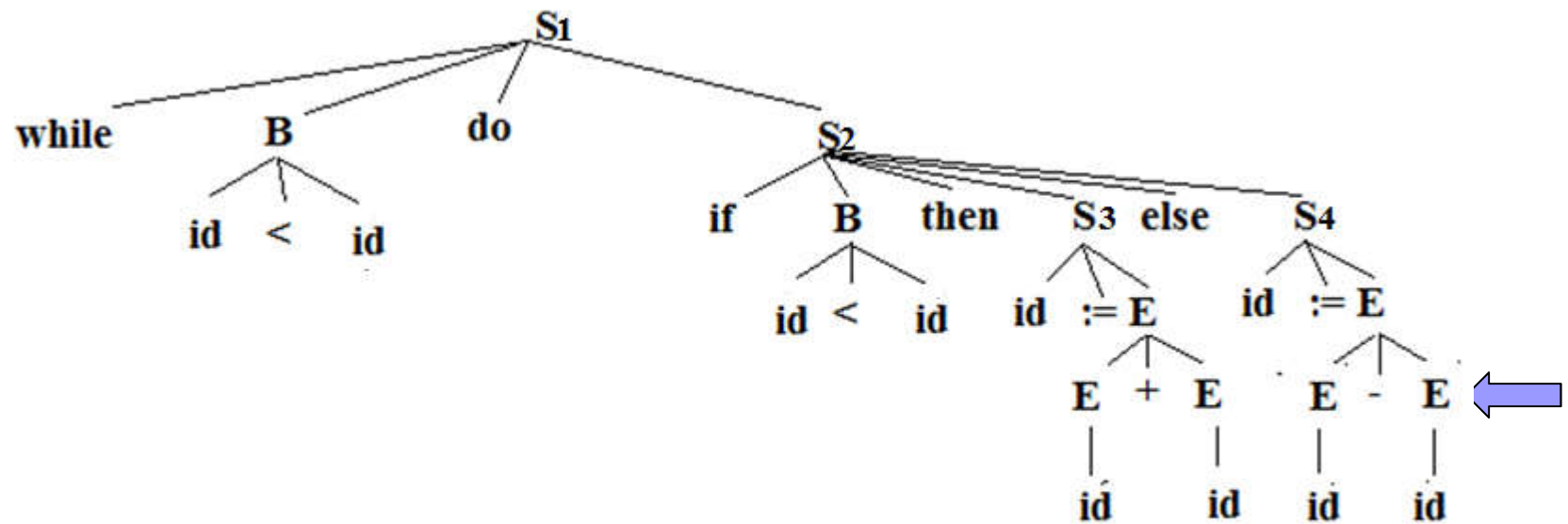






**Thuộc tính**

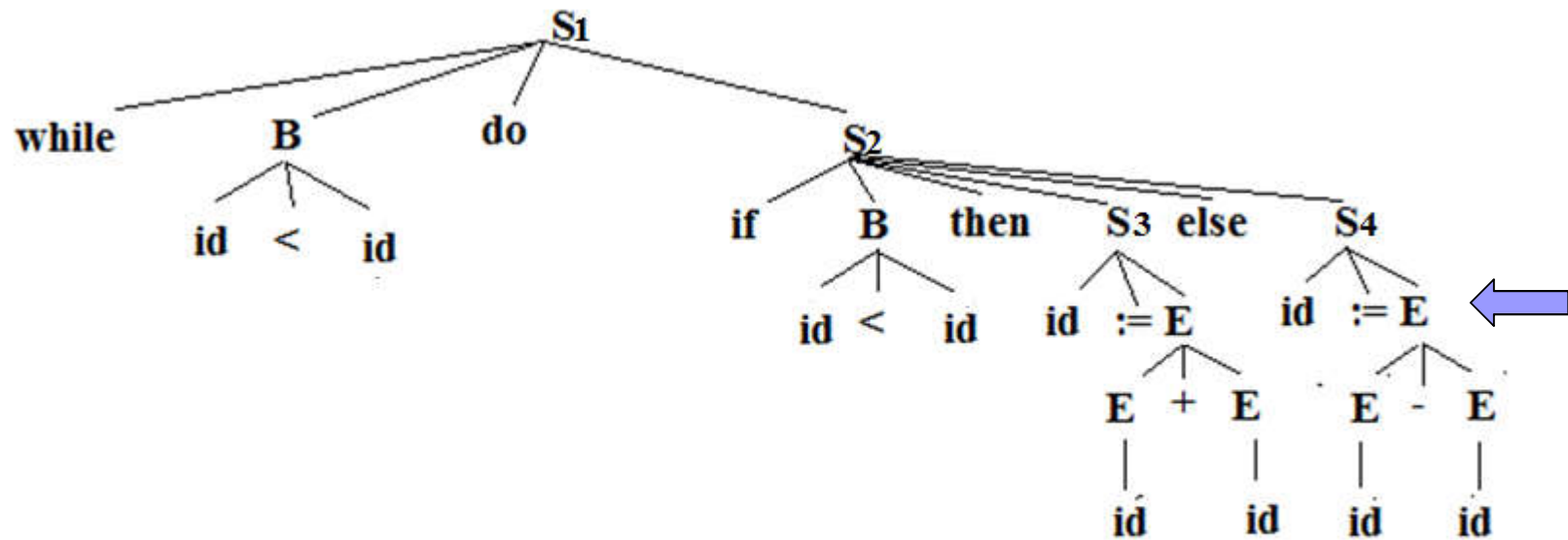
id.place = z



### Thuộc tính

E.place = z

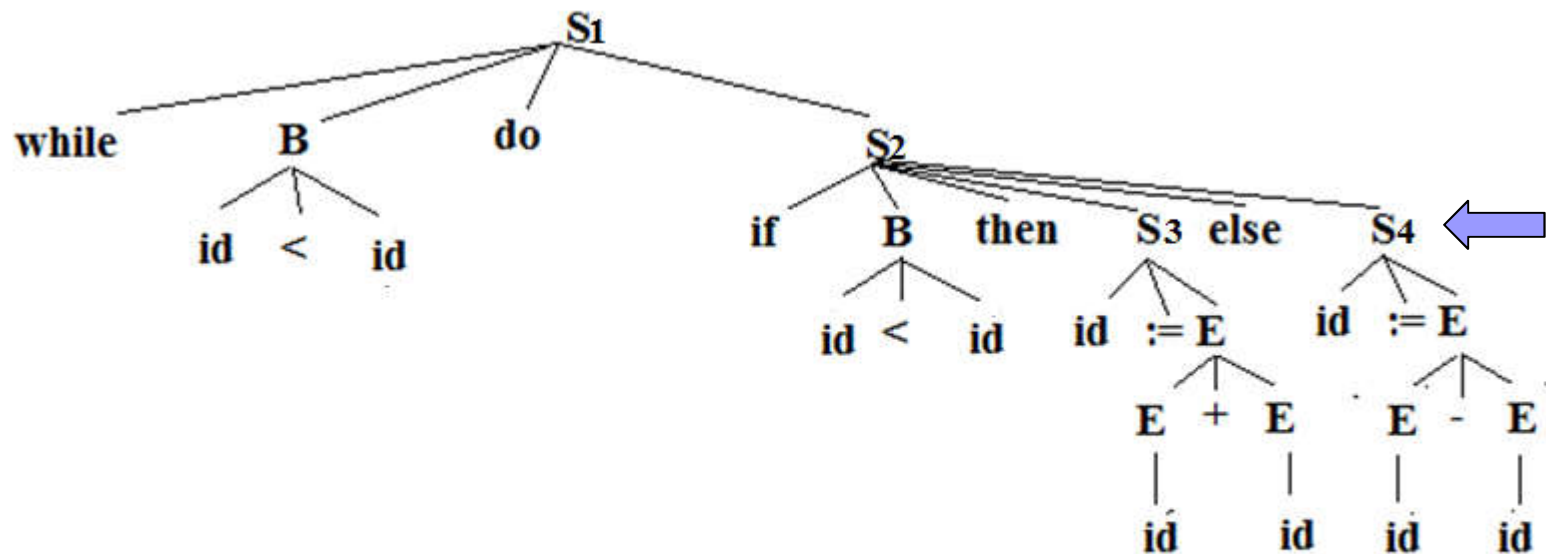
E.code = ""



### Thuộc tính

E.place = newtemp() t1

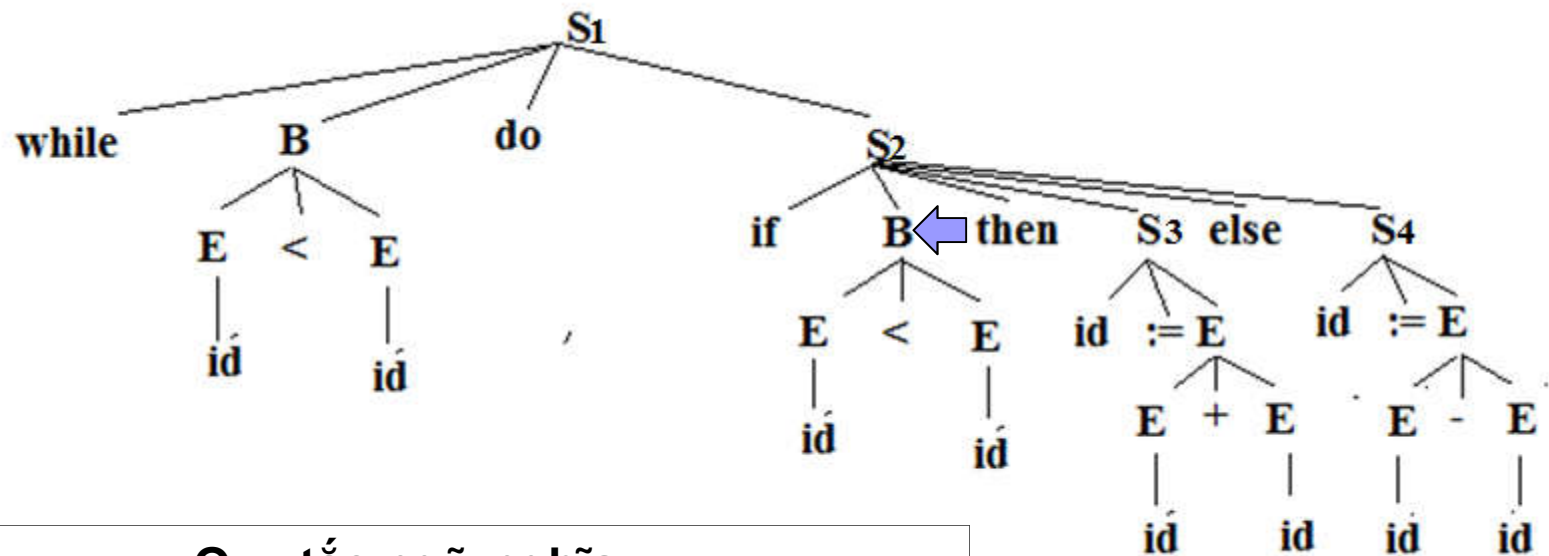
E.code= "t1:=y-z"



$S \rightarrow id := E$ 
**Quy tắc ngữ nghĩa**  
 $\{ S.code = E.code || gen(id.place := E.place) \}$

### Thuộc tính

$id.place = x$   
 $S.code = "t1 := y - z$   
 $\quad x := t1"$



### Quy tắc ngữ nghĩa

$E \rightarrow id_1 \text{ relop } id_2$

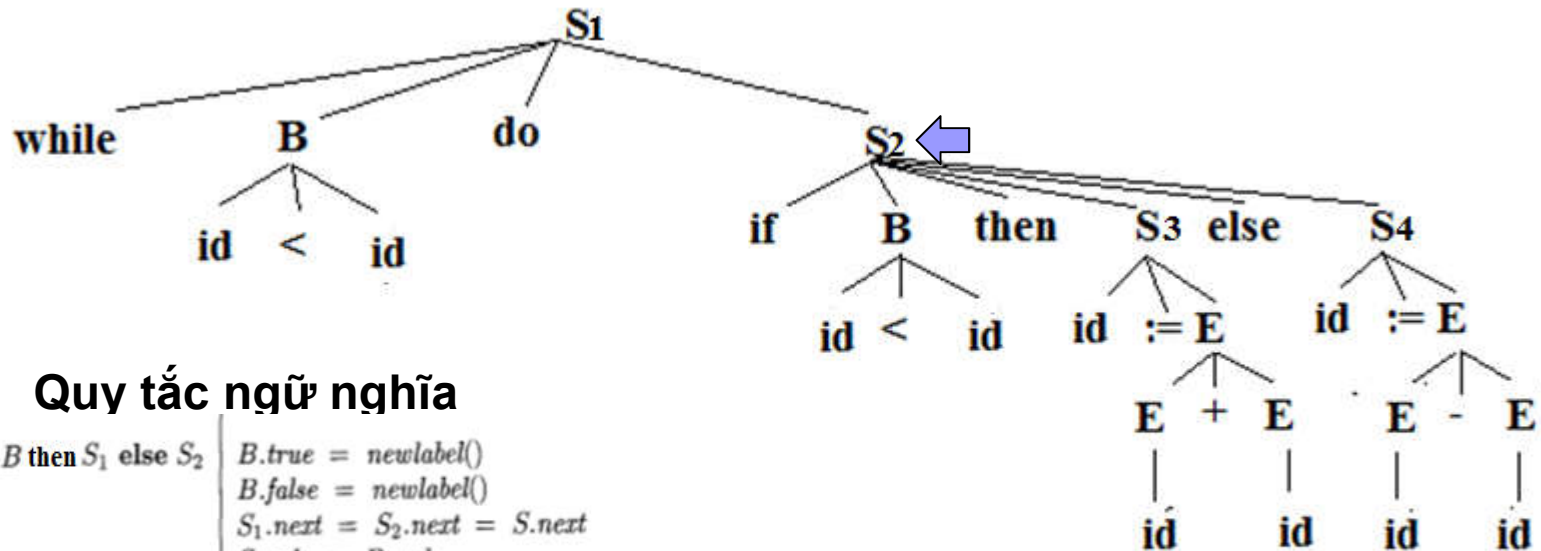
$E.code := gen('if' id_1.place \text{ relop.op } id_2.place$   
 $'goto' E.true) || gen('goto' E.false)$

### Thuộc tính

.code

if c < d goto L1

goto L2



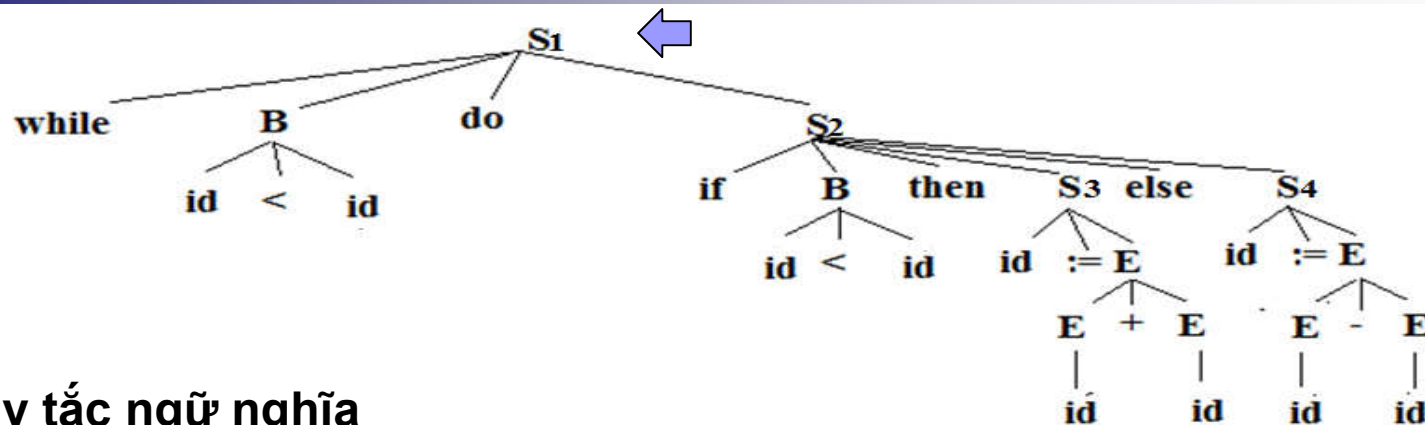
### Quy tắc ngữ nghĩa

$S \rightarrow \text{if } B \text{ then } S_1 \text{ else } S_2$	<pre> B.true = newlabel() B.false = newlabel() S1.next = S2.next = S.next S.code = B.code    label(B.true)    S1.code    gen('goto' S.next)    label(B.false)    S2.code           </pre>
--	---

### Thuộc tính

```

B.true = newlabel() -> L1
B.false = newlabel() -> L2
S2.next = S3.next = S4.next
S2.code
  if c < d goto L1
  goto L2
L1: t1 = y + z
  x = t1
  goto S.next
L2: t2 = y - z
  x = t1
  
```



## Quy tắc ngữ nghĩa

$S \rightarrow \text{while } B \text{ do } S_1$

```

begin = newlabel()
B.true = newlabel()
B.false = S.next
S1.next = begin
S.code = label(begin) || B.code
        || label(B.true) || S1.code
        || gen('goto' begin)
  
```

## Thuộc tính

begin = newlabel()  $\rightarrow$  L3  
 B.true = newlabel()  $\rightarrow$  L4  
 B.False = S.next  
 S1.next = begin = L3  $\Rightarrow$  S2.next = S3.next = L3  
 S.code  
 L3: if a < b goto L4  
     goto L0  
 L4: if c < d goto L1  
     goto L2  
 L1: t1 = y + z  
     x = t1  
     goto L3  
 L2: t2 = y - z  
     x = t1  
     goto L3

Nhãn L0 sẽ xuất hiện trong chương trình khi sử dụng các quy tắc ngữ nghĩa của luật  $S \rightarrow S_1 S_2$