IT4073:NGÔN NGỮ và PHƯƠNG PHÁP DỊCH

THÀNH CÔNG

Phạm Đăng Hải

haipd@soict.hut.edu.vn

Chương 5: Sinh mã

1. Sinh mã trung gian
2. Sinh mã đích
3. Tối ưu mã

Giới thiệu

- Bộ sinh mã trung gian chuyển chương trình nguồn sang chương trình tương đương trong ngôn ngữ trung gian
  - Chương trình trung gian là một chương trình cho một máy trừu tượng
- 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 …

Nội dung

- Chương trình dịch định hướng cú pháp
- Cây cú pháp
- Ký pháp Ba lan sau
- Mã 3 địa chỉ
  - Các dạng mã
  - Dịch trực tiếp cú pháp thành mã 3 địa chỉ
  - Sinh mã cho khai báo
  - Sinh mã cho lệnh gán
  - Sinh mã cho các biểu thức logic
  - Sinh mã cho các cấu trúc lập trình

Chương trình dịch định hướng cú pháp

Mỗi ký hiệu VP liên kết với một tập thuộc tính:

- **Thuộc tính tổng hợp**:
  - Giá trị của thuộc tính tại một nút trong cây được xác định từ giá trị của các nút con của nó.
- **Thuộc tính kế thừa**:
  - Giá trị của thuộc tính được định nghĩa dựa vào giá trị nút cha và/hoặc các nút anh em của nó.

- Tồn tại một tập luật ngữ nghĩa dùng để tính giá trị thuộc tính

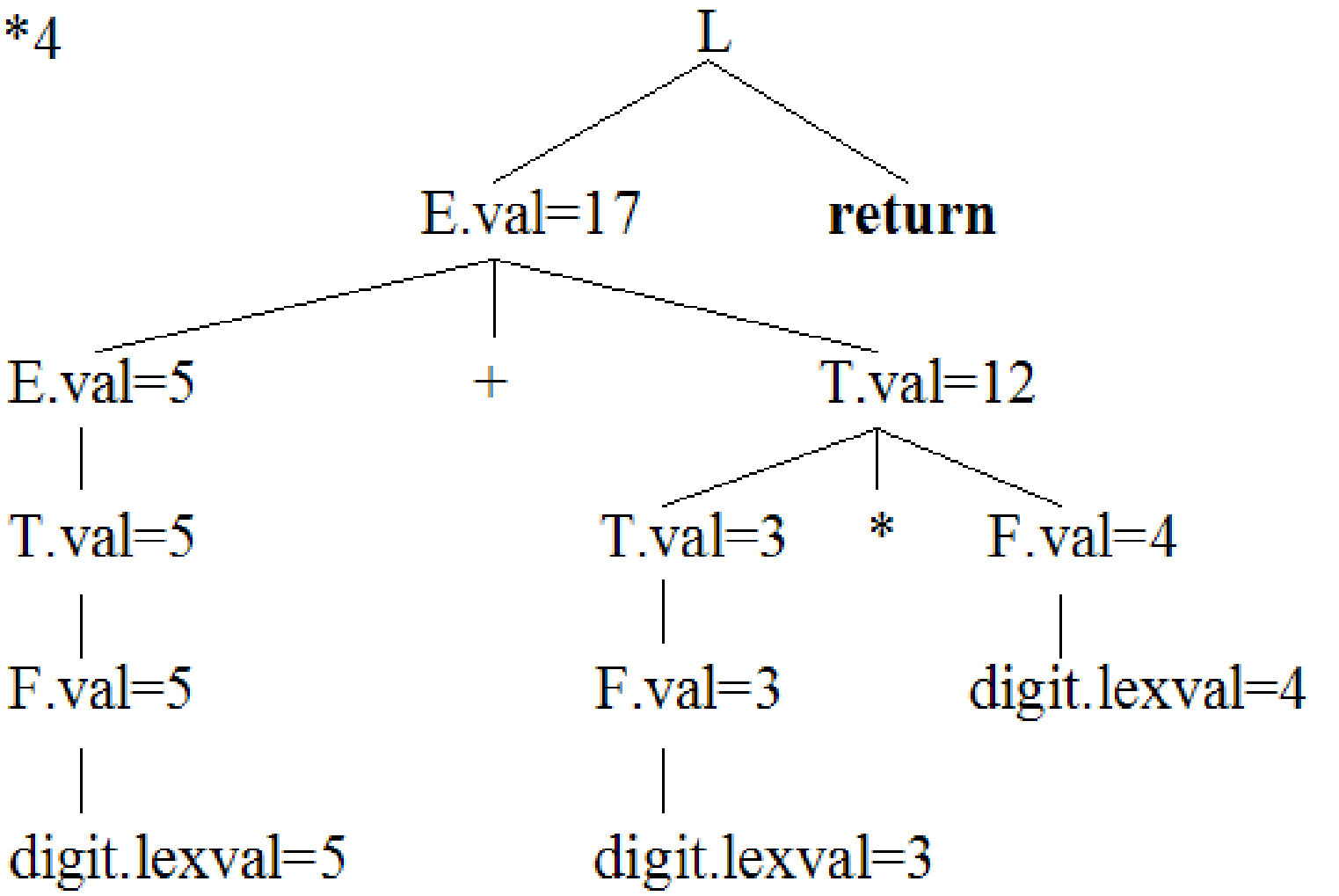
Ví dụ

Sản xuất	Quy tắc ngữ nghĩa
L → E **return**	Print (E.*val*)
E → E_1+T	E.*val* = E_1.*val* + T.*val*
E → T	E.*val* = T.*val*
T → T_1 * F	T.*val* = T_1.*val* * F.*val*
T → F	T.*val* = F.*val*
F → (E)	F.*val* = E.*val*
F → **digit**	F.*val* = digit.*lexval*

- *Các ký hiệu **E, T, F** có thuộc tính tổng hợp **val***
- *Từ tố **digit** có thuộc tính tổng hợp **lexval** (Được bộ phân tích từ vựng đưa ra)*

Chú giải cây suy dẫn

Input: 5+3*4



Nội dung

- Chương trình dịch định hướng cú pháp
- Cây cú pháp
- Ký pháp Ba lan sau
- Mã 3 địa chỉ
  - Các dạng mã
  - Dịch trực tiếp cú pháp thành mã 3 địa chỉ
  - Sinh mã cho khai báo
  - Sinh mã cho lệnh gán
  - Sinh mã cho các biểu thức logic
  - Sinh mã cho các cấu trúc lập trình

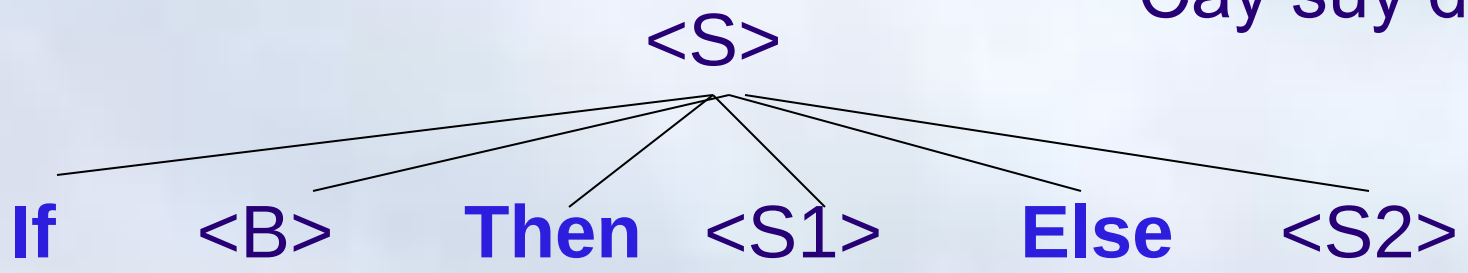
Cây cú pháp (Syntax tree)

- Cây cú pháp (*syntax tree*) là dạng thu gọn của cây phân tích (*parse tree*) dùng để biểu diễn cấu trúc của ngôn ngữ
- Trong cây cú pháp các toán tử và từ khóa không xuất hiện ở các nút lá mà đưa vào các nút trong.
  - Cha của các nút lá là các toán hạng tương ứng
- Cây cú pháp có ý nghĩa dụng trong cài đặt
  - Cây phân tích (cú pháp) chỉ ý nghĩa về mặt logic

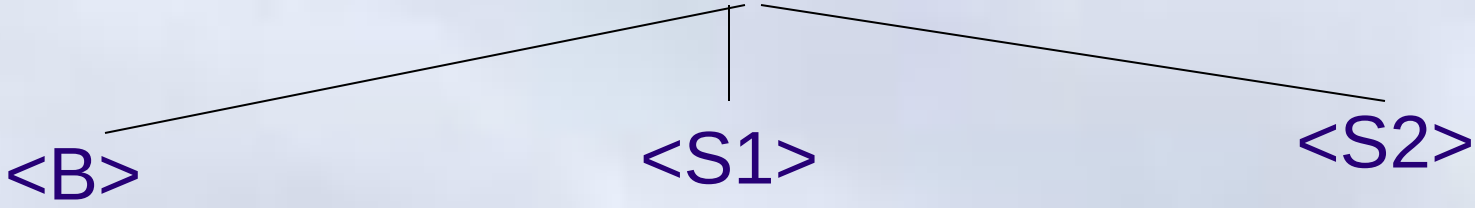
Cây cú pháp →Ví dụ 1

Ví dụ: S → **If** B **Then** S1 **Else** S2

Cây suy dẫn



If Then Else Cây cú pháp



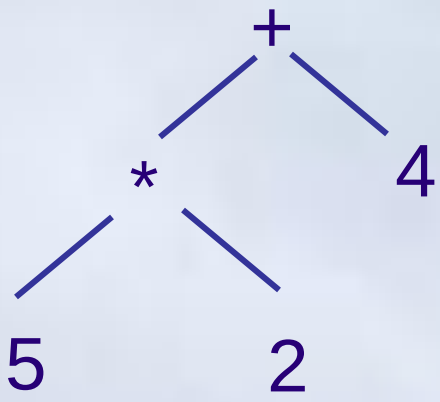
Xây dựng cây cú pháp →Ví dụ 2

E→ E+T | T

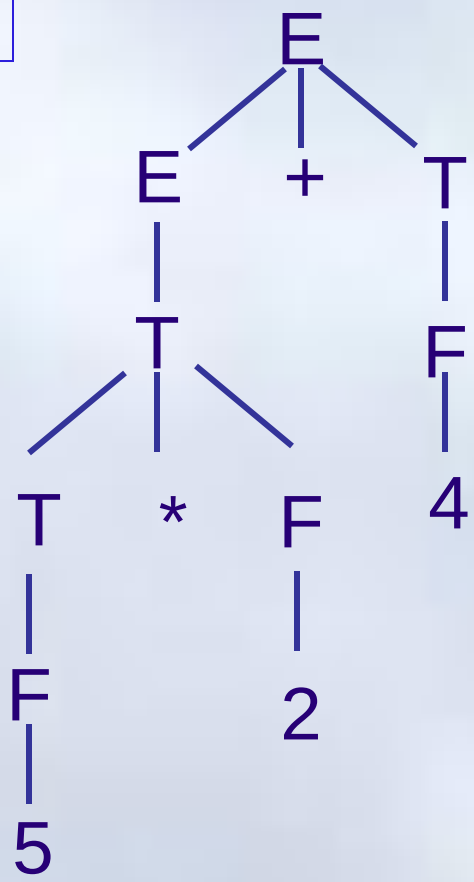
T→ T*F | F

F →(E) | num

5 * 2 + 4



Cây cú pháp

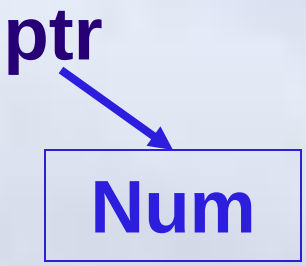


Cây phân tích

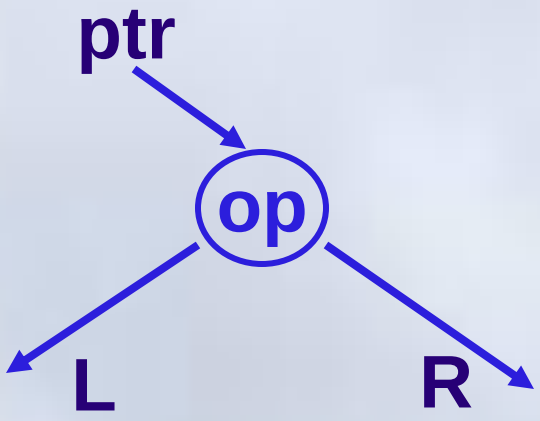
Xây dựng cây cú pháp cho biểu thức

- Các ký hiệu không kết thúc có thuộc tính tổng hợp ***link*** để lưu con trỏ, trỏ tới một nút trên cây cú pháp
- Sử dụng các hàm

ptr = mkLeaf(Num)



ptr = mkNode(op, L, R)



Cây cú pháp (Syntax tree)

Sản xuất	Luật ngữ nghĩa
$E \rightarrow E_1 + T$	E.link := mkNode(+,E_1.link,T.link)
$E \rightarrow T$	E.link := T.link
$T \rightarrow T_1 * F$	T.link := mkNode(*,T_1.link,F.link)
$T \rightarrow F$	T.link := F.link
$F \rightarrow (E)$	F.link := E.link
$F \rightarrow$ **num**	F.link := mkLeaf(num)

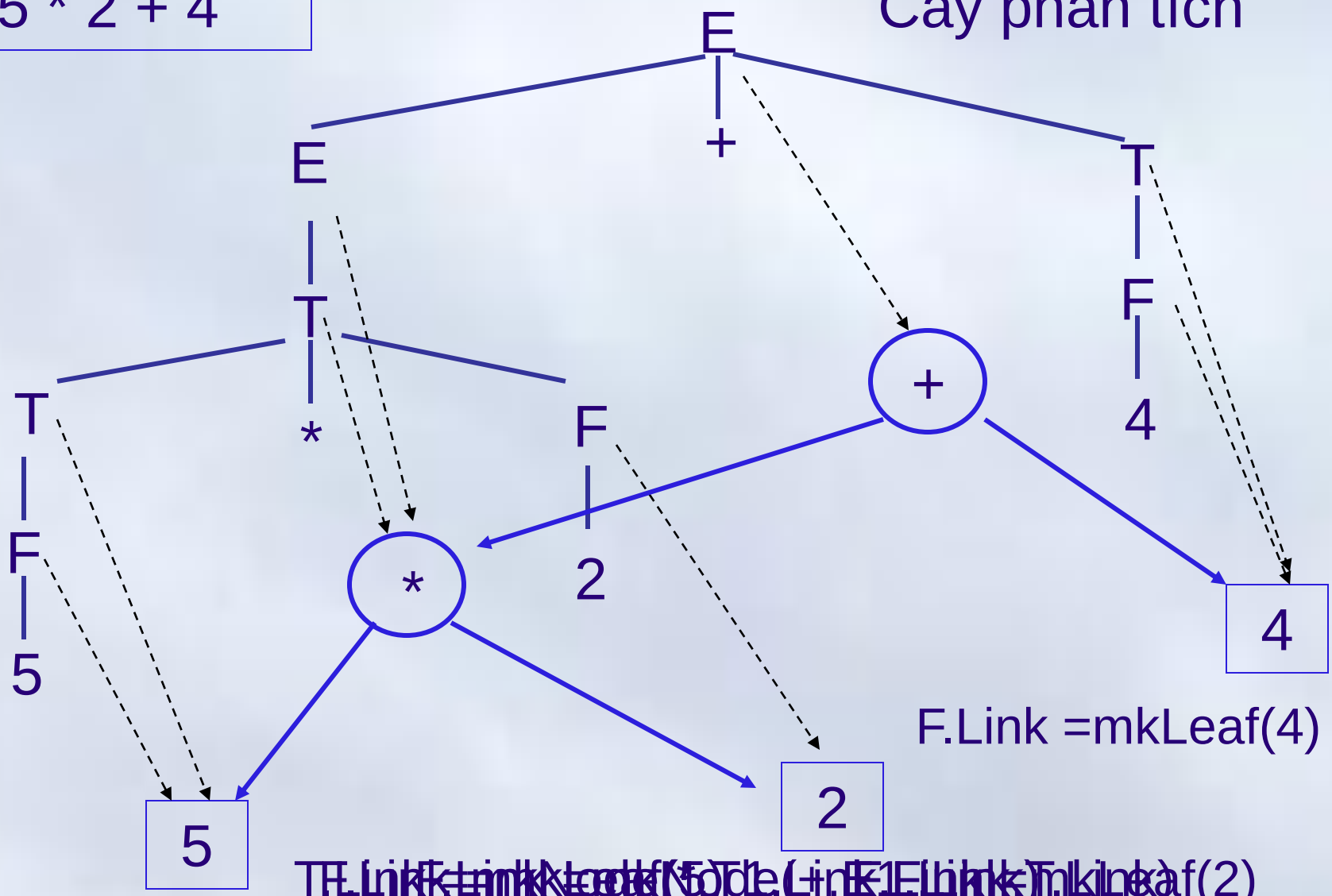
Cây cú pháp được xây dựng từ dưới lên trên

- Sau khi phân tích xong một sản xuất mới gọi luật ngữ nghĩa tương ứng (*duyệt thứ tự sau*)

Xây dựng cây cú pháp →Ví dụ

5 * 2 + 4

Cây phân tích



Nội dung

- Chương trình dịch định hướng cú pháp
- Cây cú pháp
- Ký pháp Ba lan sau
- Mã 3 địa chỉ
  - Các dạng mã
  - Dịch trực tiếp cú pháp thành mã 3 địa chỉ
  - Sinh mã cho khai báo
  - Sinh mã cho lệnh gán
  - Sinh mã cho các biểu thức logic
  - Sinh mã cho các cấu trúc lập trình

Ký pháp Ba lan sau (Reverse Polish notation)

Ký pháp Ba lan: Là một ký hiệu toán học trong đó dấu đặt trước/ sau toán hạng

- Ký pháp thông thường (giữa): 5 + 6
- Ký pháp Ba lan trước: + 5 6
- Ký pháp Ba lan sau (ngược): 5 6 +

Mục đích:

- Giảm thiểu bộ nhớ và dùng stack để tính toán

Sử dung:

- Trong một số loại máy tính tay

Quy tắc dịch dạng trung tố → dạng hậu tố

Sản xuất	Ký pháp hậu tố	Ký pháp tiền tố
E → E+T	E → E T +	E → + E T
E → T	E → T	E → T
T → T * F	T → T F *	T → * T F
T → F	T → F	T → F
F → (E)	F → E	F → E
F → **digit**	F → **digit**	F → **digit**

Ký pháp Ba lan sau → Ví dụ 1

a+b*c+d

$E \Rightarrow E + T$	$\Leftrightarrow$	$E \Rightarrow E\ T\ +$
$\Rightarrow E + T + T$	$\Leftrightarrow$	$\Rightarrow E\ T\ +\ T\ +$
$\Rightarrow T + T + T$	$\Leftrightarrow$	$\Rightarrow T\ T\ +\ T\ +$
$\Rightarrow F + T + T$	$\Leftrightarrow$	$\Rightarrow F\ T\ +\ T\ +$
$\Rightarrow a + T + T$	$\Leftrightarrow$	$\Rightarrow a\ T\ +\ T\ +$
$\Rightarrow a + T * F + T$	$\Leftrightarrow$	$\Rightarrow a\ T\ F\ *\ +\ T\ +$
$\Rightarrow a + F * F + T$	$\Leftrightarrow$	$\Rightarrow a\ F\ F\ *\ +\ T\ +$
$\Rightarrow a + b * F + T$	$\Leftrightarrow$	$\Rightarrow a\ b\ F\ *\ +\ T\ +$
$\Rightarrow a + b * c + T$	$\Leftrightarrow$	$\Rightarrow a\ b\ c\ *\ +\ T\ +$
$\Rightarrow a + b * c + F$	$\Leftrightarrow$	$\Rightarrow a\ b\ c\ *\ +\ F\ +$
$\Rightarrow a + b * c + d$	$\Leftrightarrow$	$\Rightarrow a\ b\ c\ *\ +\ d\ +$

Ký pháp Ba lan sau → Ví dụ 2

(a+b) * (c+d)

E ⇒ T	⇔	E ⇒ T +
⇒ T * F	⇔	⇒ T F *
⇒ F * F	⇔	⇒ F F *
⇒ (E) * F	⇔	⇒ E F *
⇒ (T + F) * F	⇔	⇒ T F + F *
⇒ (F + F) * F	⇔	⇒ F F + F *
⇒ (a + F) * F	⇔	⇒ a F + F *
⇒ (a + b) * F	⇔	⇒ a b + F *
⇒ (a + b) * (E)	⇔	⇒ a b + E *
⇒ (a + b) * (T + F)	⇔	⇒ a b + T F + *
⇒ (a + b) * (F + F)	⇔	⇒ a b + F F + *
⇒ (a + b) * (c + F)	⇔	⇒ a b + c F + *
⇒ (a + b) * (c + d)	⇔	⇒ a b + c d + *

Nội dung

- Chương trình dịch định hướng cú pháp
- Cây cú pháp
- Ký pháp Ba lan sau
- Mã 3 địa chỉ
  - Các dạng mã
  - Dịch trực tiếp cú pháp thành mã 3 địa chỉ
  - Sinh mã cho khai báo
  - Sinh mã cho lệnh gán
  - Sinh mã cho các biểu thức logic
  - Sinh mã cho các cấu trúc lập trình

Mã 3 địa chỉ

- Là loại mã trung gian thường dùng, tương tự mã assembly
- Chương trình trung gian là một dãy các lệnh thuộc kiểu mã 3 địa chỉ
  - Mỗi lệnh gồm tối đa 3 toán hạng
  - 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 như con trỏ tới phần tử tương ứng của nó trong bảng ký hiệu

Mã 3 địa chỉ → Ví dụ

- Câu lệnh
  - A = x + y * z
- Chuyển thành mã 3 địa chỉ

  T = y * z

  A = x + T

T là tên trung gian

  - Được bộ sinh mã trung gian sinh ra cho các toán tử trung gian

Mã 3 địa chỉ → Các dạng phổ biến

- Mã 3 địa chỉ tương tự mã Assembly:
  - Lệnh có thể có nhãn,
  - Tồn tại những lệnh chuyển điều khiển cho các cấu trúc lập trình.
- Các dạng lệnh
  - *Lệnh gán* **x := y *op* z**.
  - *Lệnh gán với phép toán 1 ngôi* : **x := *op* y**.
  - *Lệnh sao chép*: **x := y.**
  - *Lệnh gán có chỉ số* ***X := y[i] hoặc x[i]= y***

Mã 3 địa chỉ → Các dạng phổ biến

- *Lệnh gán địa chỉ và con trỏ*

  x = &y; x = * y; *x = y

- *Lệnh nhảy không điều kiện*: **goto L**,
  - L là nhãn của một lệnh
- Lệnh nhảy có điều kiện **IF x *relop* y goto L**.
  - *Nếu thỏa mãn quan hệ relop (>,>=,<,..) thì thực hiện lệnh tại nhãn L,*
  - *Nếu không thỏa mãn, thực hiện câu lệnh ngay tiếp theo lệnh **IF***

Mã 3 địa chỉ → Các dạng phổ biến

- *Gọi thủ tục với n tham số* ***call p, n.***

  Khai báo tham số ***param x***

  Trả về giá trị ***return y***

Thường dung với chuỗi lệnh 3 địa chỉ

– *Lời gọi chương trình con* ***Call p(X_1, X_2,…x_n)*** *sinh ra*

param x_1

param x_2

param x_n

Call p, n

Dịch trực tiếp cú pháp thành mã 3 địa chỉ

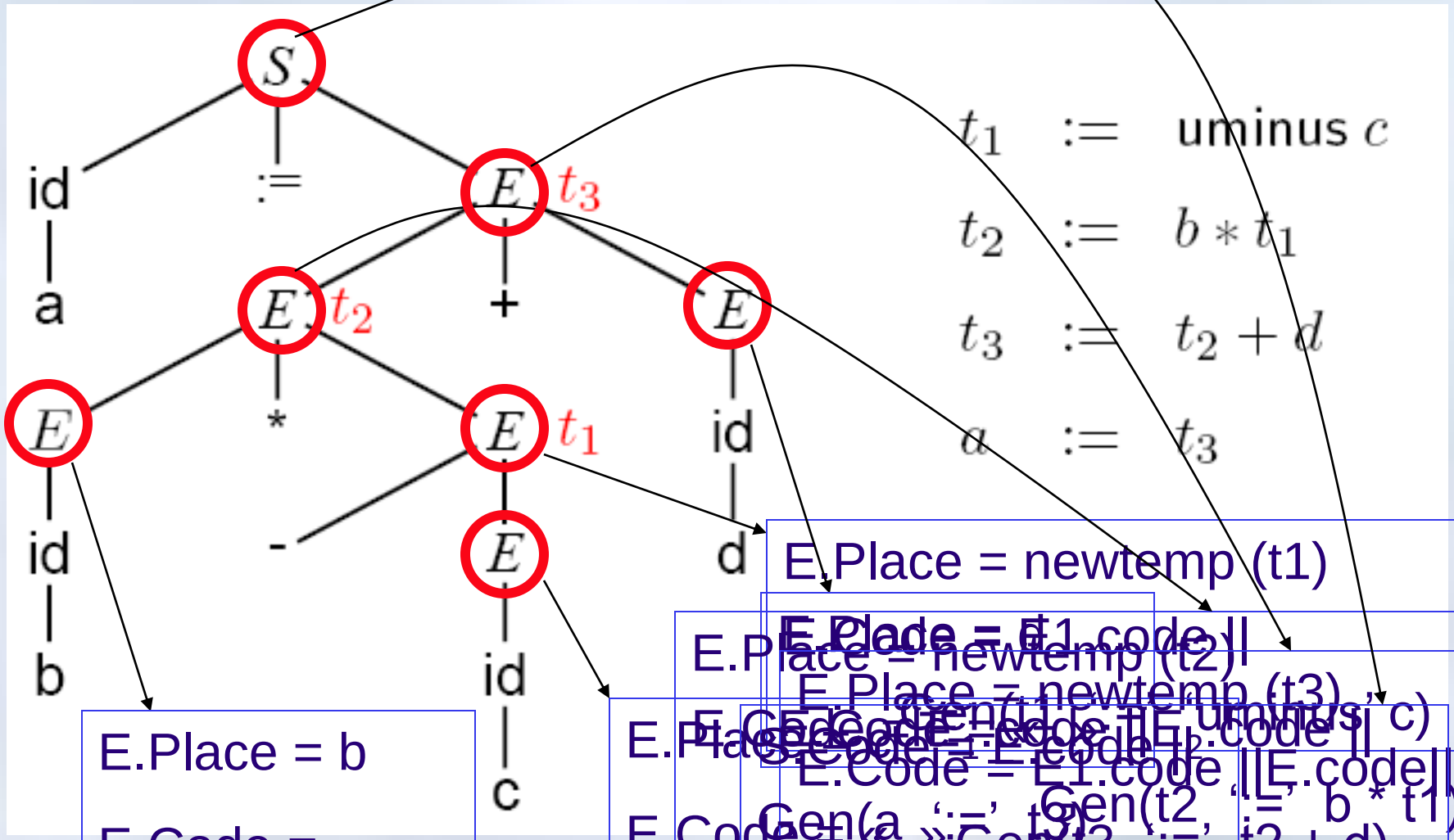
- Thuộc tính tổng hợp **S.code** biểu diễn mã ba địa chỉ của lệnh **S**
- Các tên trung gian được sinh ra cho các tính toán trung gian
- Các ký hiệu không kết thúc E có 2 thuộc tính
  - E.place: Thuộc tính địa chỉ/tên chứa giá trị của ký hiệu E
  - E.code: Chứa chuỗi mã lệnh địa chỉ để đánh giá E
- Hàm **newtemp()** sinh ra các tên trung gian t1, t2,. .
- Sử dụng hàm **gen(x ':=' y '+' z)** thể hiện mã 3 địa chỉ câu lệnh x := y + z
  - Các biểu thức ở các vị trí của x, y, z được đánh giá khi truyền vào hàm gen()

Dịch trực tiếp cú pháp thành mã 3 địa chỉ

Sản xuất	Quy tắc 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)	E.*place*= E_1.*place* ; E.*code* = E_1.*code*
E → **Id**	E.*place* = id.*place* ; E.*code* = ''

Dịch trực tiếp cú pháp thành mã 3 địa chỉ

Câu lệnh gán: a := b * -c + d



E.Place = b

E.Code = « »

E.Place = newtemp (t1)

E.Place = t1 code ||

E.Place = newtemp (t2)

E.Place = newtemp (t3)

E.Code = E1.code || E.uminus || c)

E.Code = E1.code || E2.code ||

E.Code = E1.code ||E.code||

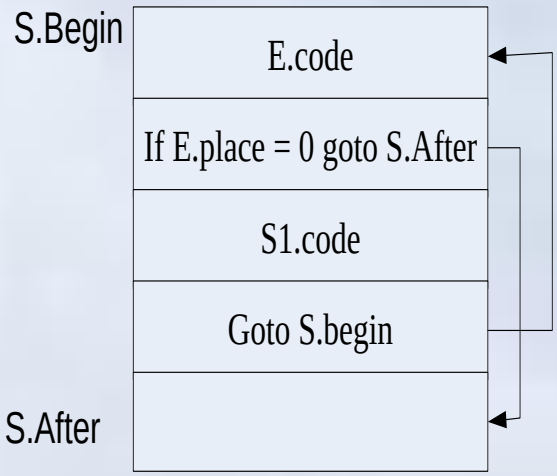
Gen(t2, ‘:=’ b * t1)

Gen(a ‘:=’ t3)

Gen(t3 ‘:=’ t2 + d)

Dịch trực tiếp cú pháp thành mã 3 địa chỉ

Ví dụ: Câu lệnh lặp while

Sản xuất	**Quy tắc ngữ nghĩa**
S → while E do S1 	S.Begin = newLabel S.After = newLabel S.Code = /*Sinh mã cho lệnh while gồm*/ gen(S.begin ':') || E.code|| /*Sinh mã cho lệnh đánh giá E*/ Gen('if' E.place '=' 0 'goto' S.After)|| S1.code|| /*Sinh mã cho lệnh S1*/ gen('goto' S.Begin)|| /*Sinh mã cho goto*/ Gen(S.After ':') /*Sinh mã cho nhãn mới*/

Hàm newLabel: Sinh ra một nhãn mới

Cài đặt lệnh 3 địa chỉ→Biểu diễn bộ bốn

- Sử dụng cấu trúc gồm 4 trường: Op, Arg1, Arg2, Result
  - **Op:** Chứa mã nội bộ của toán tử
  - Các trường Arg1, Arg2, Result trỏ tới các ô trong bảng ký hiệu ứng với các tên tương ứng
- Câu lệnh dạng **a:= b Op c**
  - Đặt b vào Arg1, C vào Arg2 và a vào Result
- Câu lệnh một ngôi: **a:= b**; **a:=-b**
  - Không sử dụng Arg2

Cài đặt lệnh 3 địa chỉ→Biểu diễn bộ bốn

Ví dụ lệnh a = -b * (c+d)

Lệnh 3 địa chỉ

t1 := -b

t2 := c + d;

t3 := t1 * t2;

a := t3

Biểu diễn bởi dãy các bộ 4

	Op	Arg1	Arg2	Result
0	uminus	b		t1
1	+	c	d	t2
2	*	t1	t2	t3
3	:=	t3		a

Các tên tạm phải được đưa vào bảng ký hiệu

Cài đặt lệnh 3 địa chỉ→Biểu diễn bộ ba

- Mục đích để trách đưa tên tạm vào bảng ký hiệu
- Tham khảo tới giá trị tạm thời bằng vị trí lệnh sử dụng tính ra giá trị này
- Bỏ trường **Result**, Các trường **Arg1, Arg2** trỏ tới phần tử tương ứng trong bảng ký hiệu hoặc câu lệnh tương ứng

	Op	Arg1	Arg2
0	uminus	b	
1	+	c	d
2	*	(0)	(2)
3	:=	a	(2)

Sinh mã cho khai báo

- Sử dụng biến toàn cục offset
  - Trước khi bắt đầu khai báo: offset = 0
  - Với mỗi khai báo biến sẽ đưa tên đối tượng, kiểu và giá trị của offset vào bảng ký hiệu
  - Tăng offset lên bằng kích thước của dữ liệu
- Các tên trong chương trình con được truy xuất thông qua địa chỉ tương đối offset
  - Khi gặp tên đối tượng (biến), dựa vào trường offset để biết vị trí trong vùng dữ liệu

Sinh mã cho khai báo

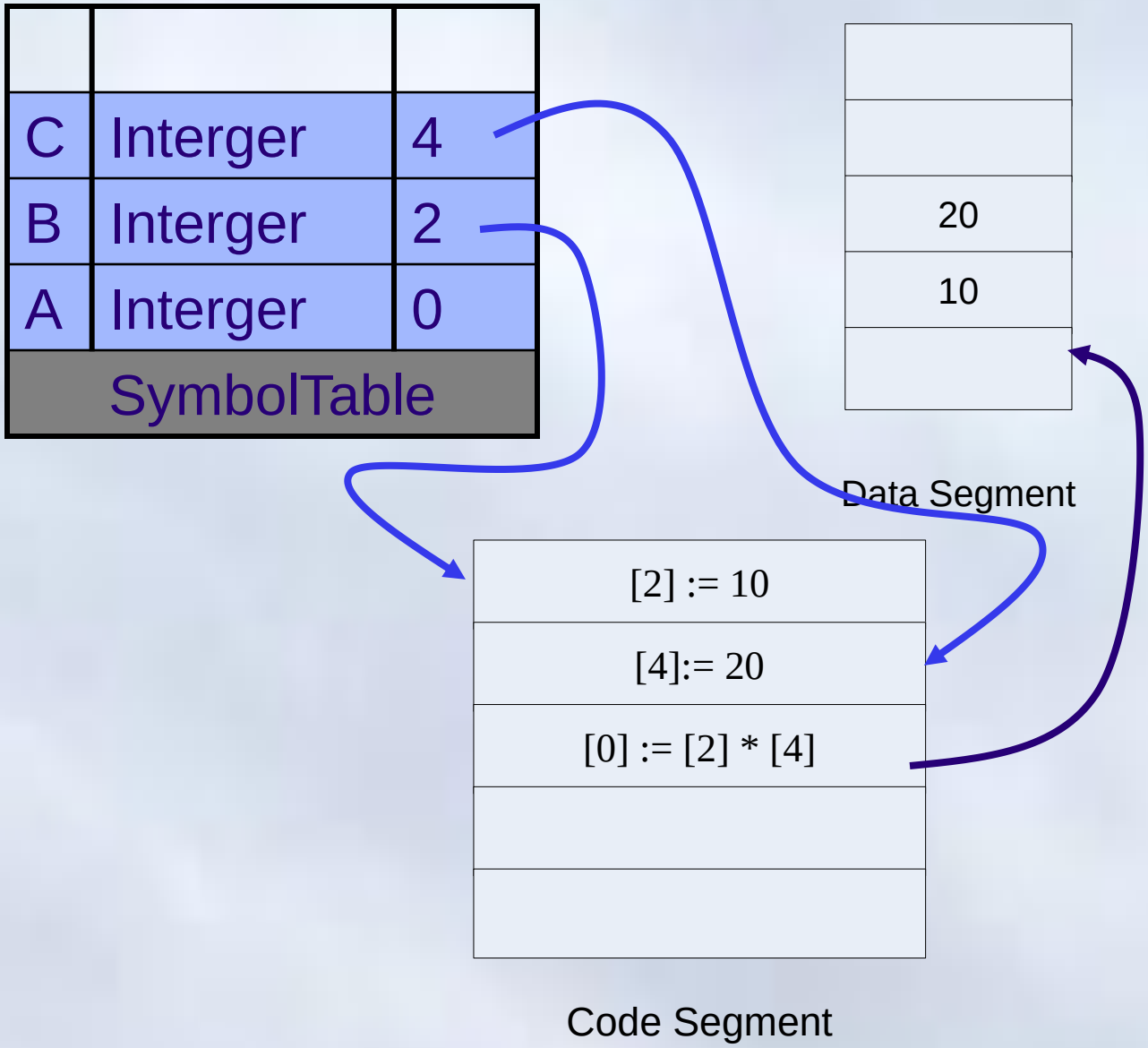
Sản xuất	Quy tắc ngữ nghĩa
P→ MD	{}
M → ε	{*Offset = 0*}
D → D ; D	
D → Id : T	*enter*(id.*name*, T.*type*, *offset*) *Offset =Offset +T.Width*
T → interger	T.*type = Interger; T.width = 2*
T → real	T.*type =real*; T.*width = 4*
T →array[num] of T_1	T.*type=array*(1..num.*val*,T_1.*type*) T.*width* = num.val * T_1.*width*

Hàm **Enter(name, type, offset)** thêm một đối tượng vào bảng ký hiệu với tên (name), kiểu(type) và địa chỉ tương đối (offset) của vùng dữ liệu của nó.

Sinh mã cho khai báo→Ví dụ

A: Interger;
B: Interger;
C:Interger;

B :=10
C:=20
A:=B*C



Lưu trữ thông tin về phạm vi

- Văn phạm cho phép các chương trình con bao nhau
  - Khi bắt đầu phân tích chương trình con, phần khai báo của chương trình bao tạm dừng
  - Dùng một bảng ký hiệu riêng co mỗi chương trình con
- Văn phạm của khai báo này:

  P → D

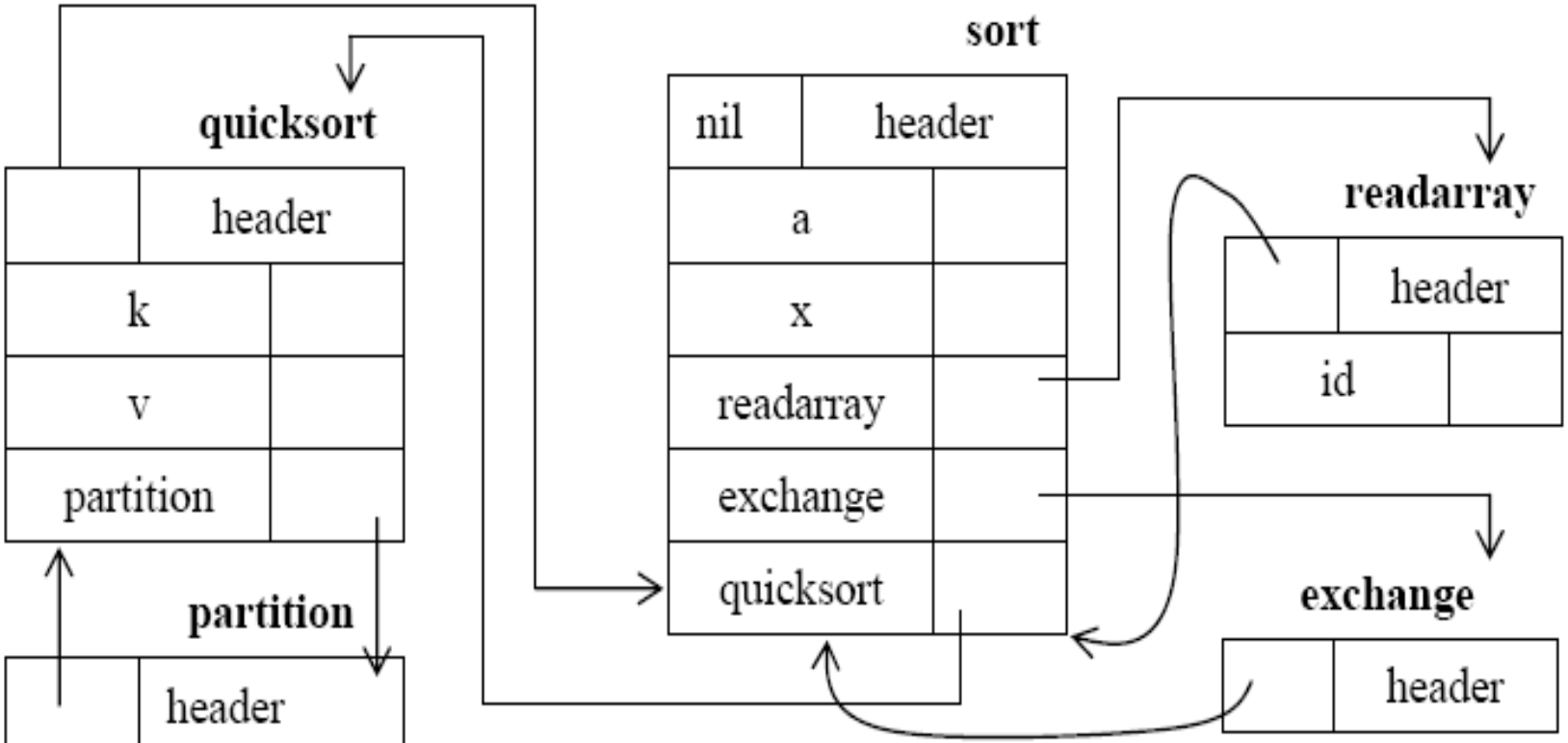
  D → D; D | id : T | proc id ; D ; S
- Khi khai báo chương trình con D → **proc** id D1; S được phân tích thì các khai báo trong D1 được lưu trong bảng kí hiệu mới.

Lưu trữ thông tin về phạm vi→Ví dụ

```
1) Program sort;      //Chương trình Quicksort
2) Var  a: array[0..10] of integer;
3)        x: integer;
4)      Procedure readarray;
5)         Var i: integer;
6)         Begin …… 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 ……exchange(i,j) end; {partition}
13)    Begin … end; {quicksort}
14)Begin … end; {sort}
```

Lưu trữ thông tin về phạm vi→Ví dụ

- Các bảng ký hiệu của chương trình sort



Quy tắc ngữ nghĩa →Các thao tác

- **mktable(previous)** – tạo một bảng kí hiệu mới, bảng này có *previous* chỉ đến bảng cha của nó.
- **enter(table,name,type,offset)** – thêm một đối tượng mới có tên *name* vào bảng kí hiệu được chỉ ra bởi *table* và đặt kiểu là *type* và địa chỉ tương đối là *offset* vào các trường tương ứng.
- **enterproc(table,name,newbtable)** – tạo một phần tử mới trong bảng table cho chương trình con name, newtable trỏ tới bảng kí hiệu của CTC này.
- **addwidth(table,width)** – ghi tổng kích thước của tất cả các p/tử trong bảng kí hiệu vào header của bảng.

Khai báo trong chương trình con

Sản xuất	Quy tắc ngữ nghĩa
P→ MD	*addwidth*(*top*(tblptr), *top*(offset)); *pop*(tblptr); *pop*(offset)
M → ε	t:=*mktable*(null); *push*(t, tblptr); *push*(0, offset)
D → D ; D	
D →**proc id**; ND_1;S	t:=*top*(tblpr); *addwidth*(t,*top*(offset)); *pop*(tblptr); *pop*(offset); *enterproc*(*top*(tblptr), id.name, t)
N → ε	t:=*mktable*(*top*(tblptr)); *push*(t,tblptr); *push*(0,offset);
D → id : T	*enter*(*top*(tblptr),id.name,T.*type*, *top*(offset); *top*(offset):=*top*(offset) + T.*width*

Tblptr: là Stack dùng chứa các con trỏ trỏ tới bảng ký hiệu

Offset: Là Stack dùng lưu trữ các Offset

Xử lý các khai báo trong chương trình con

- Sản xuất: **P→MD** :
  - Hoạt động của cây con M được thực hiện trước
- Sản xuất: **M → ε:**
  - Tạo bảng ký hiệu cho phạm vi ngoài cùng (*chương trình sort*) bằng lệnh ***mktable(nil)*** ***//Không có SymTab cha***
  - Khởi tạo stack ***tblptr*** với bảng ký hiệu vừa tạo ra
  - Đặt offset = 0.
- Sản xuất: **N → ε:**
  - Tạo ra một bảng mới ***mktable(top(tblptr))***
    - Tham số ***top*(tblptr)** cho giá trị con trỏ tới bảng cha
  - Thêm bảng mới vào đỉnh stack **tblptr** //push(t,tblptr)
  - 0 được đẩy vào stack ***offset*** //push(0,Offset)

N đóng vai trò tương tự M khi một khai báo CTC xuất hiện

Xử lý các khai báo trong chương trình con

- Với mỗi khai báo **id: T**
  - một phần tử mới được tạo ra cho id trong bảng kí hiệu hiện hành (*top(tblptr)*)
  - Stack tblptr không đổi,
  - Giá trị **top(offset)** được tăng lên bởi T.width.
- Khi **D→ proc id ; N D_1 ; S** diễn ra
  - Kích thước của tất cả các đối tượng dữ liệu khai báo trong D_1 sẽ nằm trên đỉnh stack offset.
  - Kích thước này được lưu trữ bằng cách dùng Addwidth(),
  - Các stack tblptr và offset bị lấy mất phần tử trên cùng (pop())
  - Thao tác thực hiện trên các khai báo của chương trình con.

Sinh mã cho lệnh gán → Các hàm

- Hàm lookup()
  - Tìm trong bảng kí hiệu xem một tên (*id.name*) đã tồn tại
    - Tìm trong bảng ký hiệu hiện thời (top(tblptr))
    - Nếu không có, tìm trong các bảng ký mức cha (*con trỏ trong phần header của bảng ký hiệu*)
  - Nếu tồn tại, trả về con trỏ tới vị trí; ngược lại, trả về nil.
- Thủ tục emit()
  - Ghi mã 3 địa chỉ vào một tập tin output
    - gen() xây dựng thuộc tính code cho các kí hiệu chưa kết thúc
  - Khi 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, sẽ ghi ra tập tin bên ngoài

Sinh mã cho lệnh gán

Sản xuất	Quy tắc ngữ nghĩa
$S \rightarrow Id:=E$	p := lookup(id.name) if p<>nil then emit(p ':=' E.place) else error()
$E \rightarrow E_1+E_2$	E.*Place* = newTemp() emit(E.*place* ':=' E_1.*place* '+' E_2.*place*)
$E \rightarrow E_1*E_2$	E.*Place* = newTemp() emit(E.*place* ':=' E_1.*place* '*' E_2.*place*)
$E \rightarrow -E_1$	E.*place*= *newtemp()*; *emit*(E.*place* ':=' 'uminus' E_1.*place*)
$E \rightarrow (E)$	E.*place*= E_1.*place* ;
$E \rightarrow$ **Id**	p := lookup(id.name) if p<>nil then E.place := p else error()

Địa chỉ hóa các phần tử của mảng

- Các phần tử của mảng được lưu trữ trong một khối ô nhớ kế tiếp nhau để truy xuất nhanh
- **Mảng một chiều:** nếu kích thước một phần tử là w
  $\Rightarrow$ địa chỉ tương đối phần tử thứ ***i*** của mảng A là

$$\mathbf{A[i] = base + (i\text{-}low)*w}$$

$$A[i] = i*w + (base - low*w)$$

  - *Low*: cận dưới tập chỉ số. Một số ngôn ngữ, low = 0
  - *Base*: địa chỉ tương đối của ô nhớ cấp phát cho mảng(địa chỉ tương đối của phần tử A[low])
  - c = base – low*w có thể được tính tại thời gian dịch và lưu trong bảng kí hiệu. Vậy A[i] = i*w + c
- **Mảng 2 chiều:** mảng của mảng 1 chiều

Sinh mã cho biểu thức logic

- Biểu thức logic được sỉnh 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 tiếp theo là And, và Not (*Văn phạm trên nhập nhằng*)
- Mã hóa giá trị logic true/false
  - Mã hóa bằng số; đánh giá một biểu thức logic như một biểu thức số học
  - Biểu diễn số **1: true, 0: false**

Sinh mã cho biểu thức logic→Ví dụ

- Biểu thức **a or b and not c**
  - Mã 3 địa chỉ:

    t1 = not c

    t2 = b and t1

    t3 = a or t2
- Biểu thức **a < b**
  - Tương đương lệnh **if a<b then 1 else 0**.
  - Mã 3 địa chỉ tương ứng (*g/thiết lệnh bắt đầu 100*)

    100: if a<b goto 103

    101: t:=0

    102: goto 104

    103: t:= 1

    104:

Sinh mã cho biểu thức logic: biểu diễn số

Sản xuất	Quy tắc ngữ nghĩa
$E \rightarrow E_1$ or E_2	E.Place =newTemp(); Emit(E.place '**:=**' E1.place '**or**' E2.place)
$E \rightarrow E_1$ and E_2	E.Place =newTemp(); Emit(E.place '**:=**' E1.place '**and**' E2.place)
$E \rightarrow$ not E_1	E.Place =newTemp(); Emit(E.place '**:=**' '**not**' E1.place)
$E \rightarrow Id_1$ relop Id_2	E.Place =newTemp(); Emit('**if**' id1.place relop id2.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**')
Nextstat (*next statement*) cho biết chỉ số của câu lệnh 3 địa chỉ tiếp theo	

Sinh mã cho biểu thức logic→Ví dụ

- Biểu thức a< b AND c > d
  - E→E and E → Id <Id and E → Id<Id and Id > Id

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

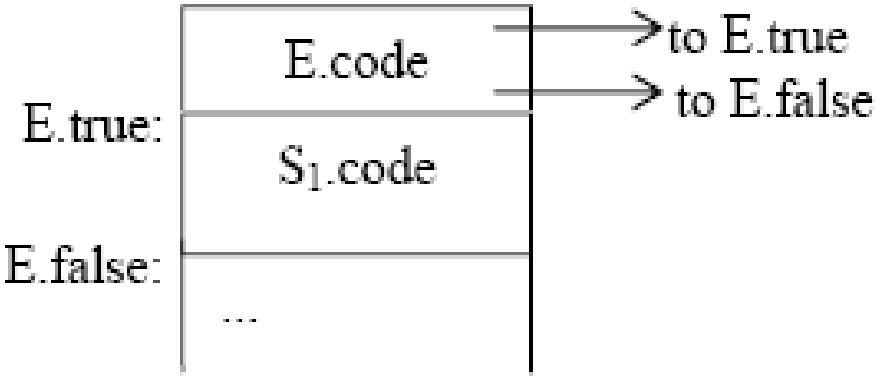
107: t2 := 1;

108: t3 := t1 and t2

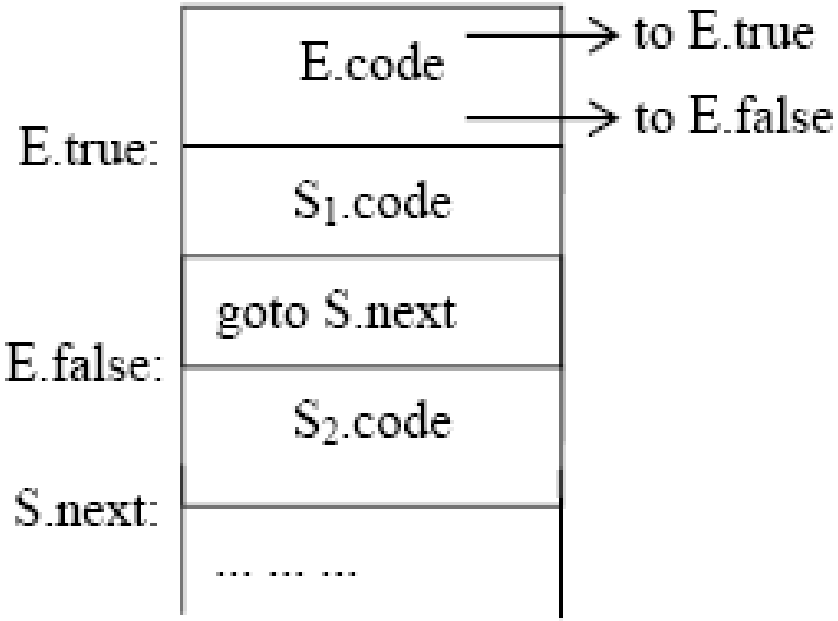
Sinh mã cho các cấu trúc lập trình

- Cấu trúc: S → if E then S_1 |
  if E then S_1 else S_2 |
  while E do S_1
- **E** là biểu thức logic. E có 2 nhãn
  - **E.true**: nhãn của dòng điều khiển nếu E là true
  - **E.false**: nhãn của dòng điều khiển nếu E là false
- **E.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

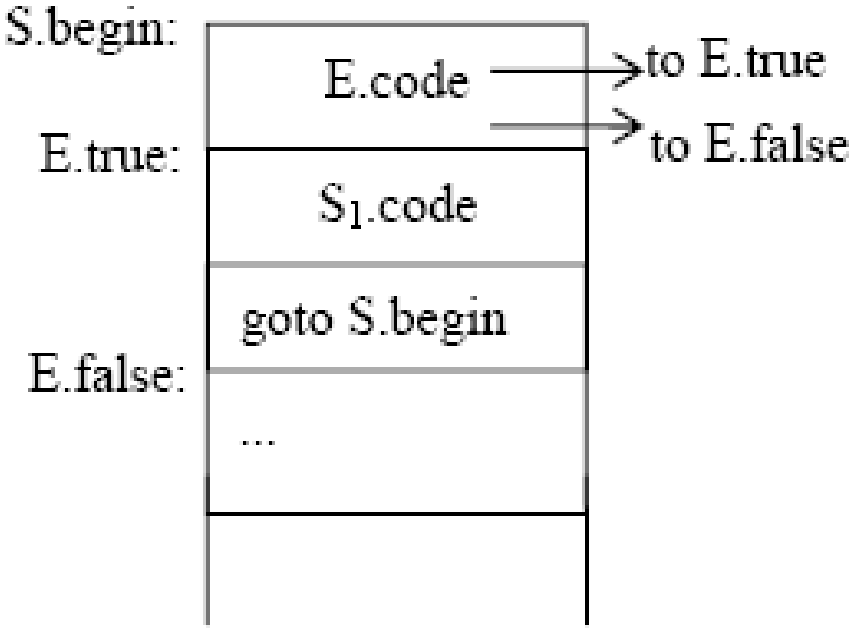
Sinh mã cho các cấu trúc lập trình



(a) if -then



(b) if -then-else



(c) while-do

Dịch trực tiếp cú pháp cho các cấu trúc lập trình

Sản xuất	Quy tắc ngữ nghĩa
$S \rightarrow$ 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
$S \rightarrow$ 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 ' : ') || S1.code || gen('goto' S.next)|| gen(E.false ' : ') || S2.code
$S \rightarrow$ while E do S_1	S.Begin=newLabel(); E.True = newLabel(); E.False = S.next; S_1.next = S.Begin S.Code = gen(S.begin ':') ||E.code ||gen(E.true ':') S_1.code || gen('**goto** ' S.Begin);

Sinh mã cho biểu thức logic trong cấu trúc lập trình

- Nếu E có dạng: a < b
  - Mã lệnh sinh ra có dạng

    If a<b then goto E.true

    goto E.false

E.Code sinh ra như thế nào?

- Nếu E có dạng: E_1 or E_2 thì
  - Nếu E_1 là true thì E cũng là true
  - Nếu E_1 là false thì phải đánh giá E_2; E sẽ là true hay false phụ thuộc E2
- Tương tự với E1 and E2
- Nếu E có dạng not E_1
  - Nếu E_1 là true, E là false và ngược lại

Sinh mã cho biểu thức logic trong cấu trúc lập trình

Sản xuất	Quy tắc ngữ nghĩa
$E \rightarrow E_1$ 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 || gen (E_1.false ‘: ‘) || E_2.code
$E \rightarrow E_1$ 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 || gen (E_1.true ‘: ‘) || E_2.code
Chú ý: E.True và E.false là các thuộc tính kế thừa	

Sinh mã cho biểu thức logic trong cấu trúc lập trình

Sản xuất	Quy tắc ngữ nghĩa
$E \rightarrow \text{not } E_1$	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
$E \rightarrow Id_1$ relop Id_2	E.Code := gen('**if**' id1.place relop id2.place '**goto**' E.true) gen('goto' E.false);
E→True	E.Code := gen('goto' E.true)
E→False	E.Code := gen('goto' E.false)

Sinh mã cho biểu thức logic → Ví dụ 1

a < b or c < d and e < f

- Giả thiết Ltrue và Lfalse là nhãn đi đến ứng với các giá trị true và false của biểu thức
- Dựa trên quy tắc ngữ nghĩa, sinh ra

```
        if a < b goto Ltrue
        goto L1:
L1:     if c < d goto L2
        goto Lfalse
L2:     if e < f goto Ltrue
        goto Lfalse
```

Sinh mã cho biểu thức logic → Ví dụ 1

- **E→a < b**

  E.code = if a < b goto E.true goto E.false

- **E→E_1 or E_2**
  - E_1.true := E.true ⇒ E_1.true = Ltrue
    - Ltrue là nhãn đi tới nếu biểu thức là true
    - Lfalse là nhãn đi tới nếu biểu thức là false
  - E_1.false := L1; //E1.False = newLabel()
  - E_2.true := E.true= Ltrue; E_2.false := E.false = Lfalse
  - E.Code = E_1.code || gen (E_1.false ‘: ‘) || E_2.code

    if a < b goto E_1.true goto E_1.false || E_1.false || E_2.code

    if a < b goto Ltrue goto L1 L1: || E_2.code (1)

Sinh mã cho biểu thức logic → Ví dụ 1

- **E→c < d** E.code = if c < d goto E.true goto E.false
- **E→e < f** E.code = if e < f goto E.true goto E.false
- **E→E_1 and E_2**
  - E_1.true := L2
  - E_1.false := E.false = LFalse;
  - E_2.true := E.true= Ltrue; E_2.false := E.false = Lfalse
  - E.Code = E_1.code || gen (E_1.true ‘: ‘) || E_2.code

  if c < d goto E_1.true goto E_1.false || E_1.true : || E_2.code

  if c<d goto L2 goto Lfalse L2: if e < f goto E_2.true goto E_2.false

  if c<d goto L2 goto Lfalse L2: if e < f goto Lttue goto Lfalse (2)

Sinh mã cho biểu thức logic → Ví dụ 2

```
While a ≠ b do
        If a > b Then
                a := a – b //  (S1)
        Else
                b := b – a  //  (S2)
```

$S \Rightarrow Id := E \Rightarrow Id := E_1 - E_2 \Rightarrow Id := Id - Id$

(A)

```
S1.Code
    t1 := a – b
    a := t1
```

```
S2.Code
    t2 := b – a
    a := t2
```

(B)

Sinh mã cho biểu thức logic → Ví dụ 2

S → While E do S_1

S.Begin = L1 // S.Begin=newLabel();

E.True = L2 // E.True = newLabel();

E.False = Next // E.False = S.next();

S_1.next = L1 // S_1.next = S.Begin

S.Code = L1: || E.code || L2 || S_1.code || goto L1 (1)

E → a ≠ b

E.Code := **if** a ≠ b **goto** L2 goto Next (2)

E.Code:=gen('**if**' id1.place relop id2.place '**goto**' E.true)
gen('goto' E.false);

Sinh mã cho biểu thức logic → Ví dụ 2

$S \rightarrow$ if E then S_1 else S_2

E.True = L3 // E.True = newLabel();

E.False = L4 // E.False = newLabel();

S_1.next = L1 // S_1.next = S.next;

S_2.next = L1 // S_2.next = S.next (3)

S.Code = E.code || L3 : || S1.code || goto L1 L4 : || S2.code

$E \rightarrow$ a > b

E.Code := **if** a > b **goto** L3 goto L4 (4)

E.Code:=gen('**if**' id1.place relop id2.place '**goto**' E.true)
gen('goto' E.false);

Sinh mã cho biểu thức logic → Ví dụ 2

1 + 2 + + 4 + A + B

```
L1 :   if  a  ≠  b goto L2
       goto Next
L2 :   if  a > b goto L3
       goto L4
L3 :   t1 := a – b
       a := t1
       goto L1
L4:    t2 := b – a
       a := t2
       goto L1
Next:
```

Biểu thức hỗn hợp

- Thực tế, các biểu thức logic thường chứa các biểu thức số học
  - (a+b)<c
- Xét văn phạm

  E →E + E | E and E | E relop E| Id

  E and E đòi hỏi 2 đối số phải là logic

  E + E, E relop E: Các đối số là biểu thức toán học
- Để sinh mã trong trường hợp phức hợp
  - Dùng thuôc tính tổng hợp E.Type cho biết kiểu là *arith* hay *logic*.

Chương 5: Sinh mã

1. Sinh mã trung gian
2. Sinh mã đích
3. Tối ưu mã

Nội dung

1. Giới thiệu
2. Môi trường thực hiện: Máy ngăn xếp
   - Bộ thông dịch cho máy ngăn xếp
3. Sinh mã đích từ mã trung gian
   - Mã trung gian là cây cú pháp
   - Mã trung gian là ký pháp Ba lan sau
   - Mã trung gian là mã 3 địa chỉ
4. Sinh mã đích từ mã nguồn
   - Xây dựng bảng ký hiệu
   - Sinh mã cho câu lệnh

1. Giới thiệu

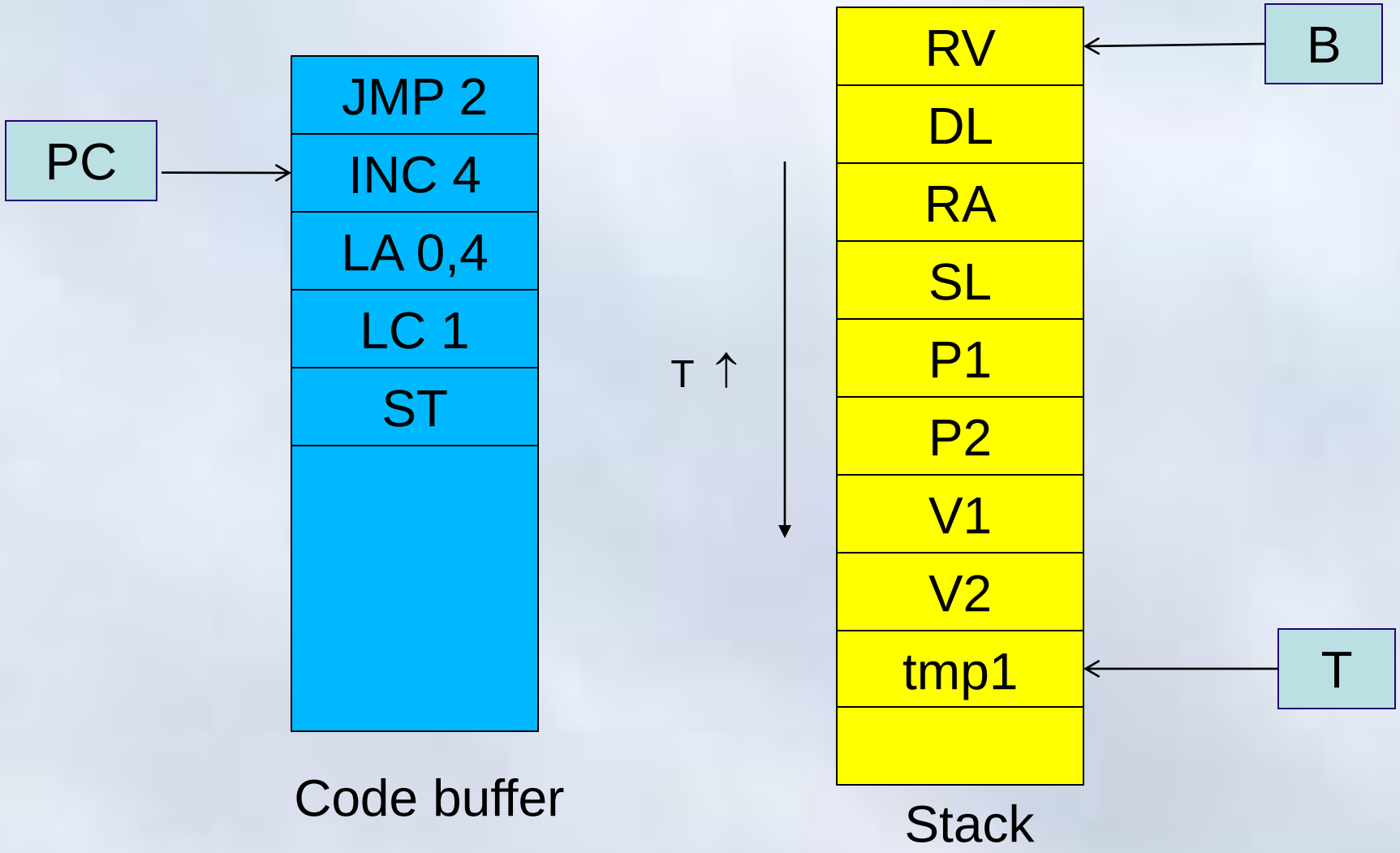
- Chương trình đich viết trên một ngôn ngữ trung gian
- Là dạng Assembly của máy giả định
  - Máy ảo (*Máy ảo làm việc với bộ nhớ stack*)
- Việc thực hiện chương trình thông qua một bộ thông dich *interpreter*
  - *Interpreter* mô phỏng hành động của máy ảo
  - Thực hiện tập lệnh assembly của nó
- Chương trình đích được dịch từ
  - Mã trung gian
  - Mã nguồn

2. Môi trường thực hiện

- Sử dụng máy ảo là máy ngăn xếp
- Máy ngăn xếp là một hệ thống tính toán
  - Sử dụng ngăn xếp để lưu trữ các kết quả trung gian của quá trình tính toán
  - Kiến trúc đơn giản
  - Bộ lệnh đơn giản
- Máy ngăn xếp có hai vùng bộ nhớ chính
  - **Khối lệnh:**
    - Chứa mã thực thi của chương trình
  - **Ngăn xếp:**
    - Lưu trữ các kết quả trung gian

2. Máy ngăn xếp

PC, B, T là các thanh ghi của máy



2. Máy ngăn xếp → Thanh ghi

- **PC (program counter):**
  - Con trỏ lệnh trỏ tới lệnh hiện tại đang thực thi trên bộ đệm chương trình
- **B (base):**
  - Con trỏ trỏ tới địa chỉ cơ sở của **vùng nhớ cục bộ**. Các biến cục bộ được truy xuất gián tiếp qua con trỏ này
- **T (top);**
  - Con trỏ, trỏ tới đỉnh của ngăn xếp

2. Máy ngăn xếp → Bản hoạt động

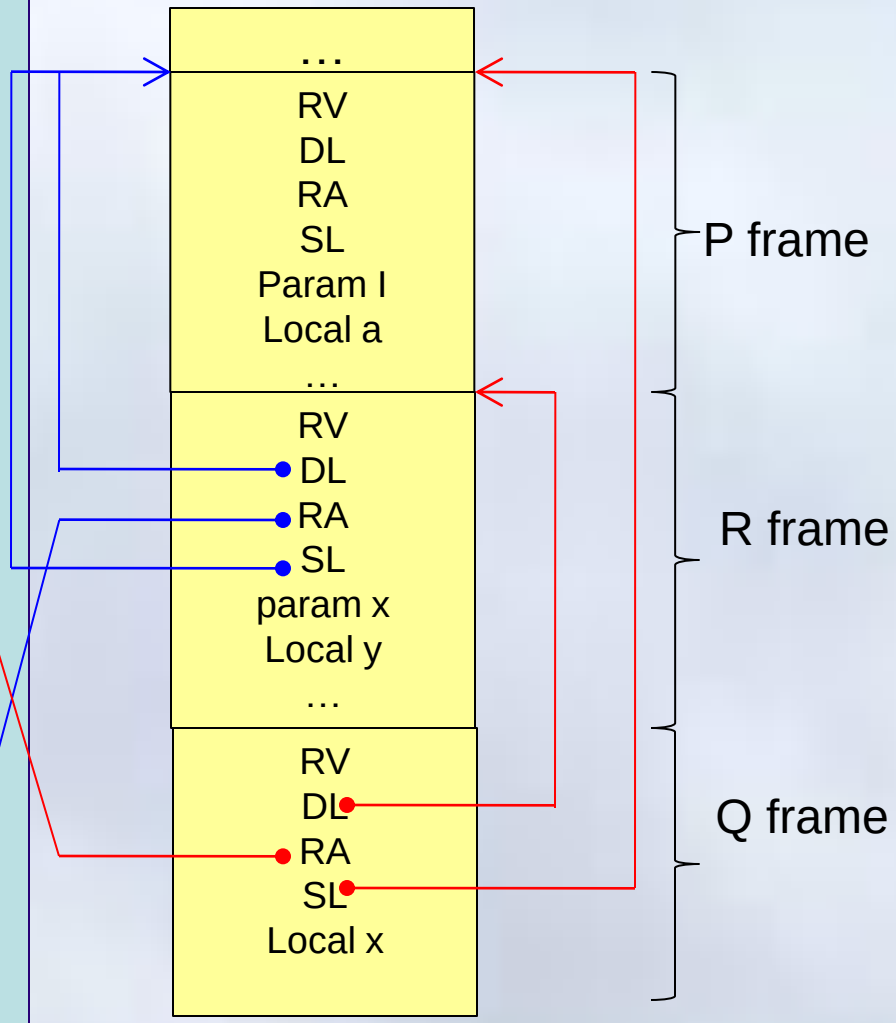
- Không gian nhớ cấp phát cho mỗi chương trình con (*hàm/thủ tục/chương trình chính*) khi chúng được kích hoạt
  - Lưu giá trị tham số
  - Lưu giá trị biến cục bộ
  - Lưu các thông tin quan trọng khác:
    - RV, DL, RA, SL
- Một chương trình con có thể có nhiều bản hoạt động

2. Máy ngăn xếp → Bản hoạt động (stack frame)

- RV (*return value*):
  - Lưu trữ giá trị trả về cho mỗi hàm
- DL (*dynamic link*):
  - Địa chỉ cơ sở của bản hoạt động của chương trình con gọi tới nó (caller).
  - Được sử dụng để hồi phục ngữ cảnh của chương trình gọi (caller) khi chương trình được gọi (called) kết thúc
- RA (*return address*):
  - Địa chỉ lệnh quay về khi kết thúc chương trình con
  - Sử dụng để tìm tới lệnh tiếp theo của caller khi called kết thúc
- SL (*static link*):
  - Địa chỉ cơ sở của bản hoạt động của chương trình con bao ngoài
  - Sử dụng để truy nhập các biến phi cục bộ

2. Máy ngăn xếp → Bản hoạt động → Ví dụ

```
Procedure P(I : integer);
  Var a : integer;
  Function Q;
    Var x : char;
    Begin
        …
      return
    End;
  Procedure R(X: integer);
    Var y : char;
    Begin
      …
      y = Call Q;
      …
    End;
  Begin
    …
    Call R(1);
    …
  End;
```

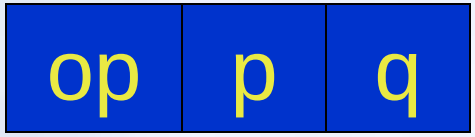


2. Máy ngăn xếp → Lệnh

- Lệnh máy có dạng : **Op p q**
  - Op : Mã lệnh
  - p, q : Các toán hạng.
    - Các toán hạng có thể tồn tại đầy đủ, có thể chỉ có 1 toán hạng, có thể không tồn tại
    - Ví dụ

```
J 1        % Nhảy đến địa chỉ 1
LA 0, 4    % Nạp địa chỉ từ số 0+4 lên đỉnh stack
HT         %Kết thúc chương trình
```

2. Máy ngăn xếp → Bộ lệnh (1/5)



LA	Load Address	t:=t+1; s[t]:=base(p)+q;
LV	Load Value	t:=t+1; s[t]:=s[base(p)+q];
LC	Load Constant	t:=t+1; s[t]:=q;
LI	Load Indirect	s[t]:=s[s[t]];
INT	Increment T	t:=t+q;
DCT	Decrement T	t:=t-q;

2. Máy ngăn xếp → Bộ lệnh (2/5)

op	p	q

J	Jump	pc:=q;
FJ	False Jump	if s[t]=0 then pc:=q; t:=t-1;
HL	Halt	Halt
ST	Store	s[s[t-1]]:=s[t]; t:=t-2;
CALL	Call	s[t+2]:=b; s[t+3]:=pc; s[t+4]:=base(p); b:=t+1; pc:=q;
EP	Exit Procedure	t:=b-1; pc:=s[b+2]; b:=s[b+1];
EF	Exit Function	t:=b; pc:=s[b+2]; b:=s[b+1];

2. Máy ngăn xếp → Bộ lệnh (3/5)

op	p	q

RC	Read Character	read one character into s[s[t]]; t:=t-1;
RI	Read Integer	read integer to s[s[t]]; t:=t-1;
WRC	Write Character	write one character from s[t]; t:=t-1;
WRI	Write Integer	write integer from s[t]; t:=t-1;
WLN	New Line	CR & LF

2. Máy ngăn xếp → Bộ lệnh (4/5)

op	p	q

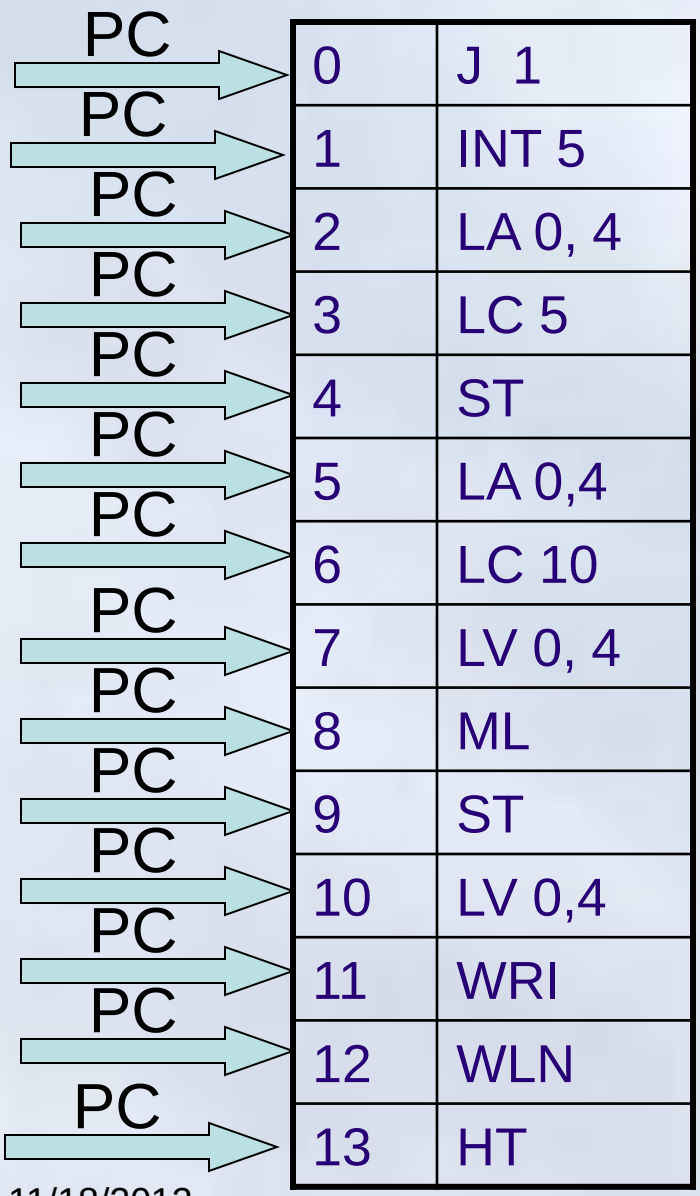
AD	Add	t:=t-1; s[t]:=s[t]+s[t+1];
SB	Subtract	t:=t-1; s[t]:=s[t]-s[t+1];
ML	Multiply	t:=t-1; s[t]:=s[t]*s[t+1];
DV	Divide	t:=t-1; s[t]:=s[t]/s[t+1];
NEG	Negative	s[t]:=-s[t];
CV	Copy Top of Stack	s[t+1]:=s[t]; t:=t+1;

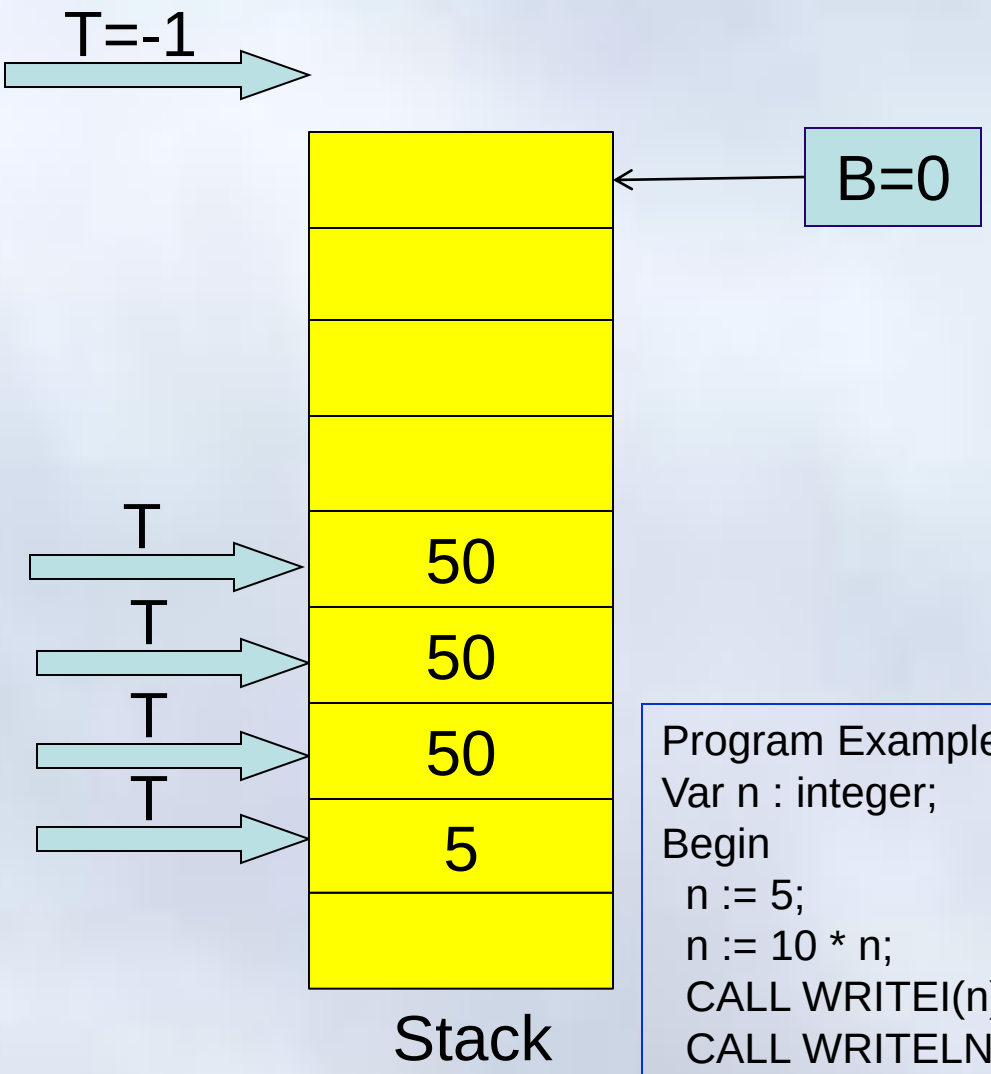
2. Máy ngăn xếp → Bộ lệnh (5/5)

op	p	q

EQ	Equal	t:=t-1; if s[t] = s[t+1] then s[t]:=1 else s[t]:=0;
NE	Not Equal	t:=t-1; if s[t] != s[t+1] then s[t]:=1 else s[t]:=0;
GT	Greater Than	t:=t-1; if s[t] > s[t+1] then s[t]:=1 else s[t]:=0;
LT	Less Than	t:=t-1; if s[t] < s[t+1] then s[t]:=1 else s[t]:=0;
GE	Greater or Equal	t:=t-1; if s[t] >= s[t+1] then s[t]:=1 else s[t]:=0;
LE	Less or Equal	t:=t-1; if s[t] <= s[t+1] then s[t]:=1 else s[t]:=0;

2. Máy ngăn xếp → Ví dụ






```
Program Example1;
Var n : integer;
Begin
  n := 5;
  n := 10 * n;
  CALL WRITEI(n);
  CALL WRITELN;
End.
```