第五章：中间代码生成(3)

过程调用和函数调用的中间代码

- 形参种类:

  值参、变量参数、函数参数

- 需要的信息:

  形参的种类、传送的内容、

  偏移、传送的个数、函数类型(实在函数、形式函数)

- 过程调用和函数调用的语法形式

  $ProcFunCall \rightarrow id\ (E_1, \ldots\ldots, E_n)$

过程调用和函数调用的中间代码

- call id(E1...En)的中间代码结构：

  E_1 的中间代码

  ……

  E_n 的中间代码

  (VarACT变参/ValACT值参,t_1,$Offset_1$,$Size_1$)

  ……

  (VarACT / ValACT , t_n ,$Offset_n$,$Size_n$)

  实参的计算结果传递到相应的形参变量

  (CALL , <f> ,true/false ,Result)

  调用过程/函数体执行

注：true静态确定转向地址；false:动态确定转向地址 （id为形参函数）；

过程调用和函数调用的中间代码

（ E_n . typ ， E_n 语义信息 ）
……
（ E_1 . typ ， E_1 语义信息）
（ — ，id ）

要检查的语义错误:
【1】id是不是函数名
【2】每个实参Ei和形参Xi的类型和种类方面是否匹配
【3】实参个数和形参个数是否相同

例：假设有实在函数调用 f(X+1,Y),并且 x 是一般整型变量，Y 是变参整型变量，f函数名，同时假定 f 的两个形参第一个是值参、整数类型，第二个是变参、整数类型，则对应的中间代码如下：

- (+ ,X ,1 ,t_1)
- (ValACT ,t_1 ,$Offset_1$,1)
- (VarACT ,Y ,$Offset_1$+1 ,1)
- (CALL ,f ,true ,t_2)

注：其中$Offset_1$和$Offset_1$+1分别示表函数f的第1、2个参数的偏移量。

过程调用和函数调用的动作文法

<ProcFunCall> → **M(List) #send_turn**

M → id #push1

List → E #nextlist

List → List,E #nextlist

#push1的语义动作：将id压入Sem栈，参数计数器i为0

#nextlist的语义动作：E已经在栈中，参数计数器累加，i++

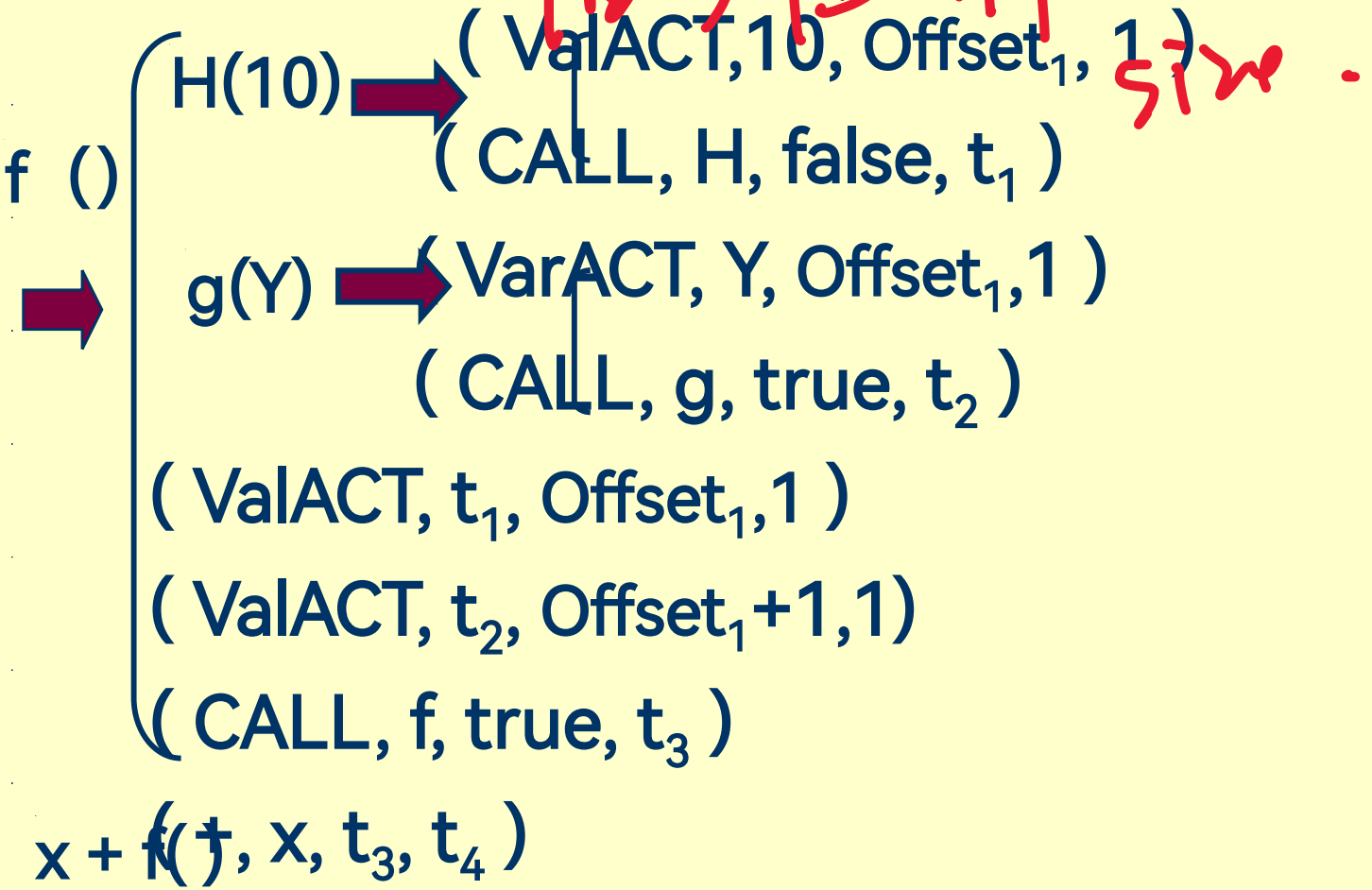
过程调用和函数调用的动作文法

#send_turn的语义动作

- 取出id:sem(s-i-1)的所有语义信息。
- 依此检查形、实参个数是否一致，检查形、实参类型是否相容;
- 产生送实参信息到形参信息的ValAct/VarAct中间代码;
- 根据f是实在过程(函数)名或形式过程(函数)名产生相应的CALL代码;
- 删除当前过程／函数调用语句所占的i+1个语义栈单元，如果f是函数，则把返回值的类型和语义信息压入Sem栈。

例：x + f (H(10), g(Y))

其中：x是整型变量，H为返回值是整型的形参函数名，H的形参为整型值参，f,g为返回值是整型的实在函数名，f的参数均为整型值参，g的参数为变参。



注：其中$Offset_1$表示函数H，g及f的第1个参数的偏移量。

控制语句的中间代码生成

GOTO语句和标号定位的中间代码

条件语句的中间代码

While语句的中间代码

GOTO语句和标号定位的中间代码

- goto L的中间代码(JMP,-,-,L)
- L:S的中间代码（Label,-,-,L）S.code

- 动作文法、代码生成算法:

S → goto L #goto

S → DL: S

DL→ L #label

GOTO语句和标号定位的中间代码

- 对于标号表有定义情况:

#goto 的语义动作

L是指向标号表中对应的位置。

Gen(goto,-,-,sem(s-1)) pop(1)

#label 的语义动作

Gen(label,-,-,sem(s-1)) pop(1)

GOTO语句和标号定位的中间代码

- 对于标号表没有定义的情况、且标号无需声明的语言，要在过程中创建标号表ArrayL:

#goto 的语义动作

（1）查ArrayL，如果没有则产生一条缺欠（需回填）转移地址的中间代码:（JMP，—，—，—），并把标号L_i、该四元式的地址以及表示该标号为未定位的标记，添加到ArrayL 。

若有则： L_i是已经定位的了，从ArrayL中取出它的地址LL_i，然后产生中间代码：

（JMP，—，—，LL_i）；

L_i是未定位的，则从ArrayL中取出它的地址LL_i，然后产生需回填转移地址的中间代码：

（JMP，—，—，LL_i）；

ArrayL（L_i）的地址填入上述中间代码编号。

GOTO语句和标号定位的中间代码

- #label的语义动作：

  产生中间代码：(LABEL , — , — , L)；

  然后查ArrayL

如果没有标号L则把该标号及其相应的语义信息加入中ArrayL，并且标记为已定位；

如果有标号L_i并标为未定位，则往对应的所有四元式回填地址。

GOTO语句和标号定位中间代码的示例:

例如有下列程序:

....
10 goto L_1;
....
20 goto L_1;
....
30 goto L_1;
....
40 L_1: S;
....
50 goto L_1;

ArrayL结构:

标号名	定位与否标志	地址/语义信息/内部标号
L_1	未定位	LL_1 p

(m) (JMP , — , — , LL_1) (m) (JMP , — , — , LL_1)

.....

(n) (JMP , — , — , LL_1) (n) (JMP , — , — , LL_1)

.....

(p) (JMP , — , — , LL_1) (p) (JMP , — , — , LL_1)

.....

(q) (LABEL , — , — , LL_1)

.....

(w) (JMP, — , — , LL_1)

条件语句的中间代码

- S →if （E） then S1 else S2(1)
- S →if （E） then S2(2)

（1）的结构:

E的四元式

（then,E,-,-） 作用：产生条件转移

S1的四元式

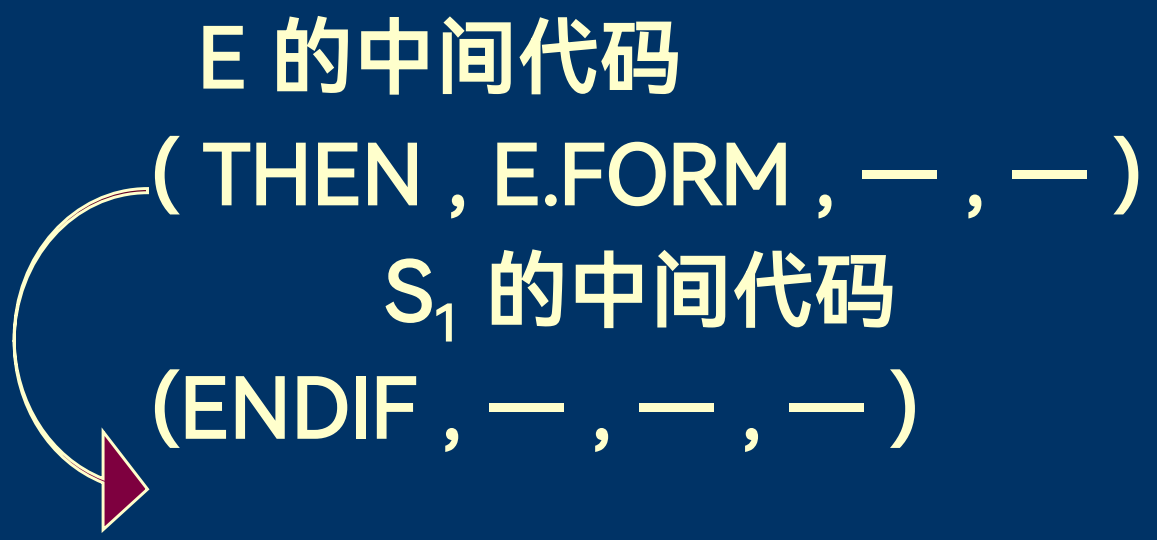
(else,-,-,-) 作用：转移、定位

S2的四元式

(endif,-,-,-) 作用：定位

条件语句的中间代码

- if E then S_1 （2）中间代码结构:



条件语句的动作文法

- S →if (E) then M S #endif
- S →if (E) then M1 S else M2 S#endif
- M →ε **#then**
- M1→ε **#then**
- M2→ε **#else**

条件语句的动作文法

- #then的语义动作

检查sem(s-1)是不是逻辑值

Gen(THEN,sem(s-1),-,-) pop(1)

#else的语义动作

Gen(ELSE,-,-,-)

#endif的语义动作

Gen(ENDIF,-,-,-)

While语句的中间代码

- While语句形式为：

  $S \rightarrow$ while （E） do S

- While语句的中间代码结构：

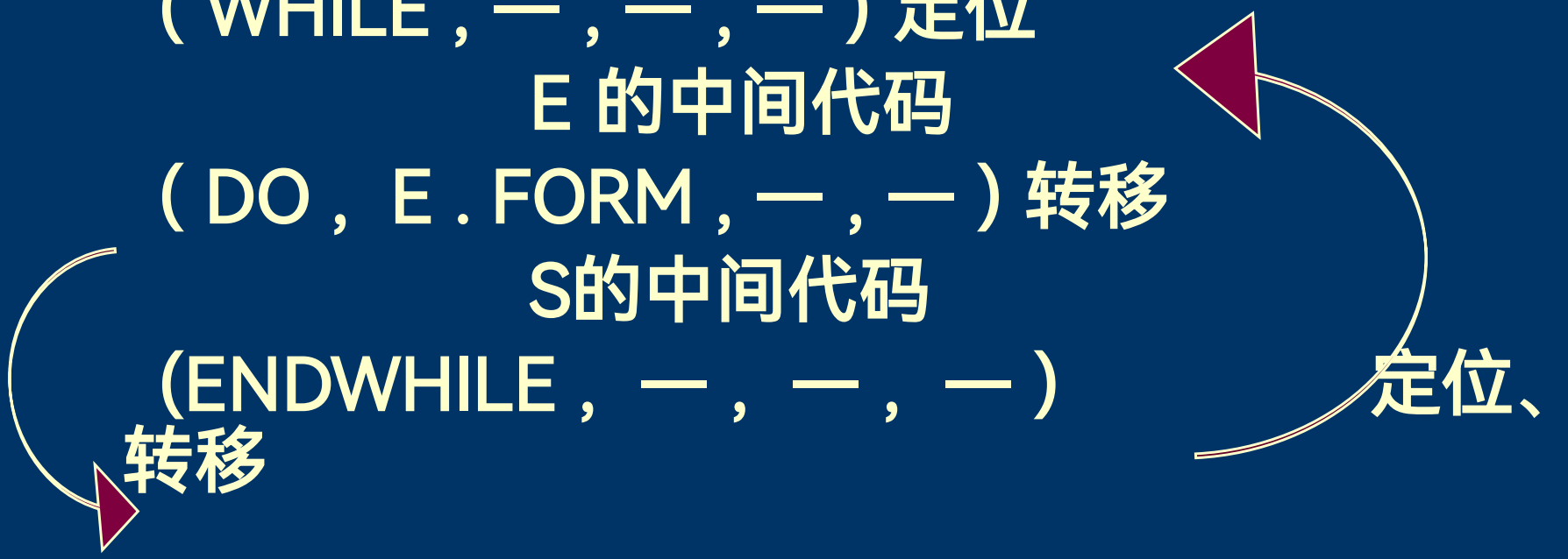
（WHILE，—，—，—）定位

E 的中间代码

（DO，E . FORM，—，—）转移

S的中间代码

（ENDWHILE，—，—，—）



While语句的中间代码

S → WHILE（E） DO S #Whileend

WHILE → while #While

DO → do #Do

#While动作： Gen(While,-,-,-)

#Do动作： E的类型检查

Gen(DO,Sem(s-1),-,-) pop(1)

#Whileend动作： Gen(Whileend,-,-,-)

过程／函数声明的中间代码生成

- 对应过程／函数Q声明的中间代码有:

（ENTRY，$Label_Q$，$Size_Q$，$Level_Q$）

或(ENTRY,Q,-,-)

Q函数/过程体的中间代码

（ENDPROC，—，—，—）或

（ENDFUNC，—，—，—）

过程／函数声明的中间代码生成

- 过程／函数声明的形式可定义为:

ProcFunDec → ProcDec | FunDec

ProcDec → Procedure id (ParamList)
Declaration
ProgramBody

FunDec → Function id (ParamList) : Type
Declaration
ProgramBody

其中ParamList对应参数声明，Declaration对应整个声明部分，ProgramBody对应程序体，而Type对应类型定义。

动作文法为:

- ProcFunDec → ProcDec | FunDec

ProcDec →

Procedure id#Entry#(ParamList) ; Declaration ProgramBody #EndProc#

- FunDec →

Function id#Entry#(ParamList) : Type; Declaration ProgramBody#EndFunc#

- Entry:

给子程序Q分配新标号$Label_Q$，并将它填到Q的符号表项中；产生入口中间代码:

(ENTRY ,$Label_Q$,$Size_Q$,$Level_Q$) 或(ENTRY,Q,-,-)

- EndProc和EndFunc:

产生出口中间代码:

(ENDPROC , — , — , —)

或 (ENDFUNC , — , — , —)

例: 过程声明的中间代码(不保存符号表)

```
procedure  Q( x: real )
    ;
    var  u : real ;
    function f( k: real ):real;
      begin
            f := k +k
      end;
begin
  u := f(50);
  y:= u * x
end;
```

(ENTRY, $Label_Q$, $Size_Q$, L_Q)

(ENTRY, $Label_f$, $Size_f$,L_f)

(+, k, k, t_0)

(=, t_0,-,f)

(ENDFUNC,_ ,_ ,_)

(FLOAT, 50,_, t_1)

(VALACT, 50,off_x,2)

(CALL, $Label_f$,true,t_2)

(=, t_2,-, u)

(*, u,x, t_3)

(=, t_3,-, y)

(ENDPROC, _ ,_ ,_)

```
P138.3 a[1..10]of integer;
a:=1;
while a<=10 do
        begin  if a<>b then
                                A[a]:
=A[b]+2
                            else  a:=a+1;
                    b:=b+1;
                end
```

(=, 1,—,a)
(WHILE , — , — , —)
(<= , a,10,t0)
(DO , t_0 , — , —)
(<> , a,b,t_1)
(THEN , t_1 , — , —)
(- ,a , 1 , t_2)
(* , t_2 , 1 , t_3)
([], A , t_3, t_4)
(- ,b , 1 , t_5)
(*, t_5 , 1 , t_6)
([], A , t_6, t_7)
(+, t_7 , 2, t_8)

(=, t_8,—,t_4)
(ELSE , — , — , —)
(+, a , 1, t_9)
(=, t_9, —,a)
(ENDIF , — , — , —)
(+, b , 1, t_{10})
(=, t_{10}, —, b)
(ENDWHILE ,— , — , —)