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

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

过程调用和函数调用的中间代码

- call f(E1...En)的中间代码结构：

  E_1 的中间代码

  ……

  E_n 的中间代码

  (VarACT / ValACT, 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：

- (ADDI ,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> → id #CallHead#
(<ParamList>) #CallTail#

<ParamList> → ε | <ExpList>

<ExpList> → E #ActParam# <NextList>

<NextList> → ε | , <ExpList>

其中

- CallHead:当遇到过程 ／ 函数名id时，将其符号表地址压入Sem栈，令实参计数器为0 。
- ActParam:对每个实参 E_i ：产生它的中间代码，将结果的类型和语义信息压入Sem栈，实参计数器加1 。

<ProcFunCall> → id #CallHead#
(<ParamList>) #CallTail#
<ParamList> → ε | <ExpList>
<ExpList> → E #ActParam#
<NextList>
<NextList> → ε | , <ExpList>

- CallTail:
- 取出id的所有语义信息。
- 检查形、实参个数是否一致，检查形
  容；
- 产生送实参信息到形参信息的VaiAct
- 根据f是实在过程(函数)名或形式过程
  的CALL代码；
- 删除当前过程 ／ 函数调用语句所占的语义栈单元，如果f是函数，则把返回值的类型和语义信息压入Sem栈。

Sem

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

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。

f ( ) ⇒

H(10) ⇒
- (ValACT, 10, $Offset_1$, 1)
- (CALL, H, false, t_1)

g(Y) ⇒
- (VarACT, Y, $Offset_1$, 1)
- (CALL, g, true, t_2)

- (ValACT, t_1, $Offset_1$,1)
- (ValACT, t_2, $Offset_1$+1, 1)
- (CALL, f, true, t_3)

x + f( ) (ADDI, x, t_3, t_4)

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

有些程序设计语言的标号是用说明语句来声明的（如Pascal），而大多数程序设计语言的标号则是直接在语句前面使用。这两种情形的中间代码生成过程有所不同。

对于用说明语句来定义标号的情形

标号声明格式：LABELL L_1, L_2, …, L_n

转向语句：　goto　L_i

标号定位：　L_i ： S

标号的语义信息:所标识的代码地址,或是指其内部标号 （为其分配的一个存储单元，用来存储它所标识的代码地址）。

标号的处理原则：设立标号表，其类似于符号表。当进入一个局部化单位时，建立本层标号表，把本层的标号及其语义信息填入表中；当结束一个局部化单位时，删除本层标号表。

（1） 标号声明格式：

LABEL L_1 , L_2 , …… , L_n ;

当扫描标号声明部分时用语义子程序 NewLabel 给每个 L_i 分配一个内部标号 LL_i （ L_i, LL_i）填入标号表.

（2）对转向语句“goto L_i ”，产生中间代码 （ JMP , — , — , LL_i ）；

（3）对标号定位“ L_i : …”，产生中间代码 （ LABEL , — , — , LL_i ）.

对于没有说明语句来定义标号的情形

转向语句：　goto　L_i

标号定位：　L_i ： S

在处理所有整个程序之前，需要建立一个数组ArrayL来记录当前遇到的所有标号及其语义信息（内部标号及一个用于表示该标号是否已经定位了的标志），初始时为空。

ArrayL结构：

标号名	定位与否标志	地址/语义信息/内部标号

标号名	定位与否标志	地址/语义信息/内部标号

（1）每当遇到转向语句“goto L_i ”，先查一下ArrayL，

如果没有查到该标号：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（作为L_i的语义信息）以及表示该标号为未定位的标记，添加到ArrayL 。

若查到标号L_i:

①L_i是已经定位的了，则从ArrayL中取出它的地址LL_i，然后产生中间代码 ：

（ JMP ， — ， — ， LL_i ）；

②L_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。

（2）每当遇到标号定位“ L_i ：…” ，首先给每个 L_i分配一个内部标号 LL_i，产生中间代码：（ LABEL ， — ， — ， LL_i ）；然后查ArrayL，如果没有标号L_i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  L1;
....
20 goto  L1;
....
30 goto  L1;
....
40    L1: S;
....
50 goto  L1;
```

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	m, LL_1, p

(m) (JMP , — , — , LL_1)

……

(n) (JMP , — , — , LL_1)

……

(p) (JMP , — , — , LL_1)

……

(q) (LABEL , — , — , LL_1)

……

(w) (JMP, — , — , LL_1)

(m) (JMP , — , — , LL_1)

(n) (JMP , — , — , LL_1)

(p) (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.FORM，—，—）作用：产生条件转移

S1的四元式

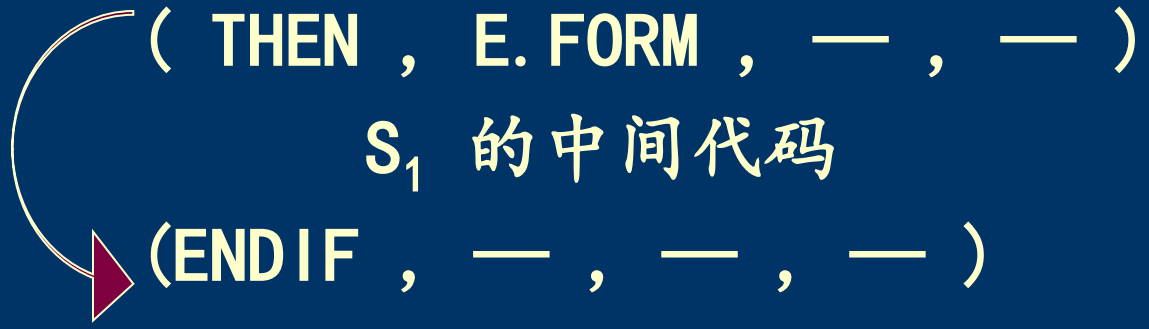
(ELSE，—，—，—)作用：转移、定位

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：



- 条件语句生成中间代码的动作文法：

<S>→if <E> then #ThenIf# <S>
<ElsePart> #EndIf#

< ElsePart > → else #ElseIf# <S>

< ElsePart > → ε

- ThenIf:当遇到关键字then时，该语句的条件表达式计算结果的语义信息已经在语义栈的栈顶，因此主要完成以下工作

1）根据Sem[top].FORM的值，检查它的类型是否为boolean类型，

2）产生中间代码：（ THEN ， Sem [top] ， — ， — ）；

3） E弹栈：pop(1)

- Endlf:遇条件语句结束符：产生中间代码:
  （ENDIF，—，—，—）
- Elself:当遇到关键字else时，产生中间代码：
  （ELSE，—，—，—）

While语句的中间代码

- While语句形式为：

  S → 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 ，— ，— ）条件转移

S的中间代码

(ENDWHILE ， — ， — ， —)

定位、转移

- while语句的中间代码生成动作文法：

〈S〉 → while #StartWhile# 〈E〉 do
#DoWhile# 〈S〉 #EndWhile#

其中动作子程序

◆StartWhile:遇 while 时：产生中间代码:
(WHILE , — , — , —)

◆DoWhile:遇 do 时（表达式E处理完，其类型和地址在Sem[top]）：
(1) 类型检查：检查E是否为boolean类型
(2) 产生中间代码： (DO , Sem[top].FORM , — , —)
(3) E弹栈：pop(1)

◆EndWhile:遇循环结束符时：产生中间代码 ：
(ENDWHILE , — , — , —)

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

(ASSIG, 1,—,a)
(WHILE , — , — , —)
(<= , a, 10, t_0)
(DO , t_0 , — , —)
(<> , a,b,t_1)
(THEN , t_1 , — , —)
(SUBI ,a , 1 , t_2)
(MULTI , t_2 , 1 , t_3)
(AADD, A , t_3, t_4)
(SUBI ,b , 1 , t_5)
(MULTI , t_5 , 1 , t_6)
(AADD, A, t_6, t_7)
(ADDI, t_7 , 2, t_8)

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

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

- FunDec →

Function id (ParamList) : Type; Declaration #Entry# 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, $Label_f$, $Size_f$, L_f)
(ADDF, k, k, t_0)
(ASSIG, t_0, -, f)
(ENDFUNC, _ , _ , _)
(ENTRY, $Label_Q$, $Size_Q$, L_Q)
(FLOAT, 50, _, t_1)
(VALACT, t_1, off_x, 2)
(CALL, $Label_f$, true, t_2)
(ASSIG, t_2, -, u)
(MULTF, u, x, t_3)
(ASSIG, t_3, -, y)
(ENDPROC, _ , _ , _)