اصول طراحی کامپایلر

درس‌نامه‌ی ۱۷

کاظم فولادی
http://kazim.fouladi.ir

ویراست اول: ۱۳۸۸
ویراست دوم: ۱۳۹۳

فهرست مطالب

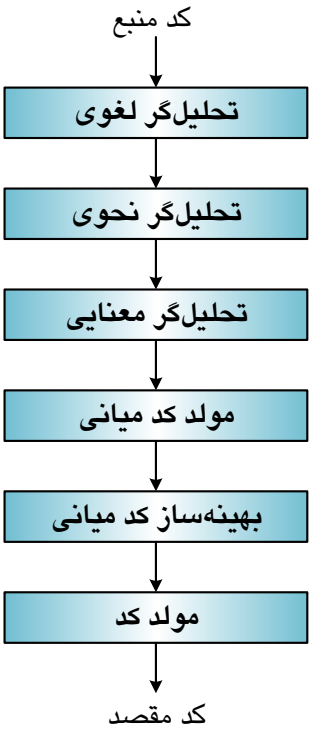
۱۷

تولید کد میانی

INTERMEDIATE CODE GENERATION

۱-۱۷ مقدمه

تولید کد میانی، چهارمین مرحله‌ی فرآیند کامپایل است.



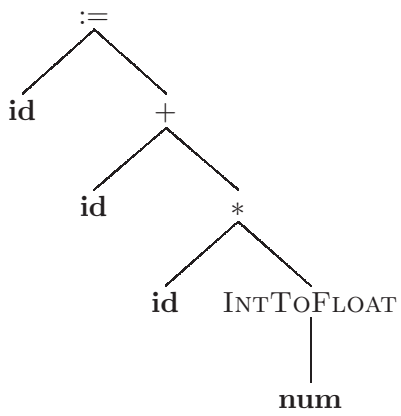
شکل ۱۷-۱: تولید کد میانی به عنوان چهارمین مرحله‌ی فرایند کامپایل

- در این مرحله، کد میانی به عنوان برنامه‌ای برای یک ماشین انتزاعی تولید می‌شود.
- کد میانی باید به گونه‌ای باشد که بسادگی به کد مقصد تولید شود (مانند کد سه‌آدرسی).

شکل ۱۷-۲: مرحله‌ی تولید کد میانی

مثال

نمونه‌ای ساده از تولید کد میانی:



ساختار نحوی انتزاعی

↓

کد میانی سه‌آدرسی

```
temp1 := inttofloat(60)
temp2 := id3 * temp1
temp3 := id2 + temp2
id1 := temp3
```

۲-۱۷ کد سه‌آدرسی

- **Assignment**
  - **Binary:** $x := y$ **op** z
  - **Unary:** $x :=$ **op** y
  - **"op" can be any reasonable arithmetic or logical operator.**
- **Copy**
  - **Simple:** $x := y$
  - **Indexed:** $x := y[i]$ **or** $x[i] := y$
  - **Address and pointer manipulation:**
    - $x := \&y$
    - $x := *y$
    - $*x := y$
- **Jump**
  - **Unconditional: goto** L
  - **Conditional: if** x $relop$ y **goto** L_1 **[else goto** L_2**, where** $relop$ **is** $<, =, >, \geq, \leq$ **or** $\neq$**.**
- **Procedure call**
  - **Call procedure** $P(X1, X2, \ldots, Xn)$

```
PARAM X1
PARAM X2
...
PARAM Xn
CALL P,n
```

۱-۲-۱۷ پیاده‌سازی کد سه‌آدرسی: چهارتایی‌ها

Quadruples

	op	arg1	arg2	result
(0)	uminus	c		t_1
(1)	*	b	t_1	t_2
(2)	uminus	c		t_3
(3)	*	b	t_3	t_4
(4)	+	t_2	t_4	t_5
(5)	:=	t_5		a

۲-۲-۱۷ پیاده‌سازی کد سه‌آدرسی: سه‌تایی‌ها

Triples

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

۳-۲-۱۷ پیاده‌سازی کد سه‌آدرسی: سه‌تایی‌های غیرمستقیم

Indirect triples

	statement
(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)	assign	a	(18)

۳-۱۷ کد برای دستور انتساب

یک تعریف S-Attributed برای تولید کد دستور انتساب (assignment) می‌تواند به صورت زیر باشد (|| به معنای عمل الحاق رشته‌هاست.)

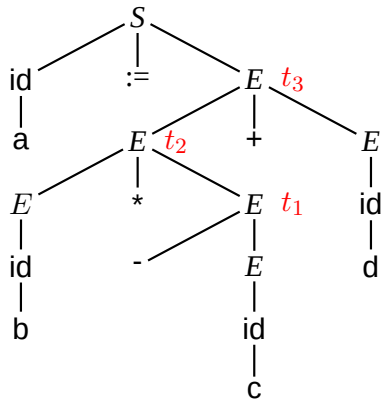
PRODUCTION	SEMANTIC RULES
$S \rightarrow \mathsf{id} := E$	$S.code := E.code \parallel gen(\mathsf{id}.place\ ':='\ E.place)$
$E \rightarrow E_1 + E_2$	$E.place := newtemp;$ $E.code := E_1.code \parallel E_2.code \parallel$ $gen(E.place\ ':='\ E_1.place\ '+'\ E_2.place)$
$E \rightarrow E_1 * E_2$	$E.place := newtemp;$ $E.code := E_1.code \parallel E_2.code \parallel$ $gen(E.place\ ':='\ E_1.place\ '*'\ E_2.place)$
$E \rightarrow -E_1$	$E.place := newtemp;$ $E.code := E_1.code \parallel gen(E.place\ ':='\ '\mathsf{uminus}'\ E_1.place)$
$E \rightarrow (E_1)$	$E.place := E_1.place; E.code := E_1.code$
$E \rightarrow \mathsf{id}$	$E.place := \mathsf{id}.place; E.code := ''$

- خصیصه‌ی ترکیبی $S.code$ کد سه‌آدرسی را برای دستور انتساب نشان می‌دهد.
- نام‌های موقتی برای محاسبات میانی تولید می‌شوند.
- برای عبارت‌ها، دو خصیصه‌ی ترکیبی زیر را در نظر گرفته‌ایم:
  ۱) $E.place$ حاوی نامی برای متغیر موقتی شامل مقدار E
  ۲) $E.code$ حاوی کد سه‌آدرسی برای ارزیابی E
- تابع $newtemp$ متغیرهای موقتی را با نام‌های متمایز t_1، t_2، ... تولید می‌کند.
- تابع gen کد سه‌آدرسی را به صورت زیر تولید می‌کند:
  ۱) هر چیزی که داخل علامت نقل قول قرار دارد، به عنوان لیترال رشته‌ای در نظر گرفته می‌شود.
  ۲) سایر اجزا ارزیابی می‌شوند.
- در عمل می‌توان کد حاصل را بجای قرار دادن در خصیصه‌ی $code$، مستقیماً به خروجی فرستاد.

مثال

نمونه‌ی یک کد برای دستور انتساب:

Given the assignment a := b ∗ −c + d the code generated by the above grammar is:



$$
\begin{aligned}
t_1 &:= \text{uminus } c \\
t_2 &:= b * t_1 \\
t_3 &:= t_2 + d \\
a &:= t_3
\end{aligned}
$$

۱۷-۳-۱ انتساب و جدول نمادها

- فرض می‌کنیم هر نام به جای اشاره‌گری به مدخل آن نام در جدول نمادها قرار گرفته باشد، زیرا سایراطلاعات آن نام نیز برای تولید کد نهایی لازم است (مانند آدرس متغیرها).
- با فرض فوق، هرگاه یک نام موقتی توسط تابع $newtemp$ ایجاد می‌شود، آن را نیز باید وارد جدول نمادها کرد.
- تابع $lookup(\mathbf{id}.name)$ مقدار nil برمی‌گرداند اگر مدخل مربوط به **id** پیدا نشود و در غیر این صورت اشاره‌گری به مدخل آن برمی‌گرداند.
- تابع $lookup(\mathbf{id}.name)$ را می‌توان به گونه‌ای پیاده‌سازی کرد که حوزه‌ی دید متغیرها را نیز در نظر بگیرد: اگر $name$ در جدول نماد فعلی ظاهر نشده بود، جدول نماد احاطه‌کننده‌ی آن بررسی می‌شود.
- بجای استفاده از خصیصه‌ی $code$، کد سه‌آدرسی را به فایل خروجی $emit$ می‌کنیم.
این کار در صورتی ممکن است که $code$ ناپایانه‌ی سمت چپ، همیشه از الحاق $code$ ناپایانه‌های سمت راست با همان ترتیب به دست آید (البته ممکن است بین آنها تعدادی رشته‌ی اضافی هم موجود باشد).

انتساب‌ها و جدول نمادها: ترجمه

PRODUCTION	SEMANTIC RULES
$S \rightarrow \mathsf{id} := E$	$p := lookup(\mathsf{id}.name)$; if $p \neq nil$ then $emit(p\,' :='\ E.place)$ else $error$
$E \rightarrow E_1 + E_2$	$E.place := newtemp$; $emit(E.place\,' :='\ E_1.place\,' +'\ E_2.place)$
$E \rightarrow E_1 * E_2$	$E.place := newtemp$; $emit(E.place\,' :='\ E_1.place\,' *'\ E_2.place)$
$E \rightarrow -E_1$	$E.place := newtemp$; $emit(E.place\,' :='\ '\mathsf{uminus}'\ E_1.place)$
$E \rightarrow (E_1)$	$E.place := E_1.place$
$E \rightarrow \mathsf{id}$	$p := lookup(\mathsf{id}.name)$; if $p \neq nil$ then $E.place := p$ else $error$

۴-۱۷ عبارت‌های بولی

- مقادیر منطقی
- عبارت‌های شرطی در دستورات کنترل برنامه

گرامر:

$$E \rightarrow E \textbf{ or } E \mid E \textbf{ and } E \mid \textbf{not } E \mid (E) \mid \textbf{id relop id} \mid \textbf{true} \mid \textbf{false}$$

دور روش برای ارزیابی‌عبارت‌های بولی وجود دارد:

۱) **بازنمایی عددی (Numerial representation)**:
true را با '۱' و **false** را با '۰'کد می‌کنیم و مشابه عبارت‌های محاسباتی با آنها برخورد می‌نماییم.

۲) **کد پرش ـ مدار کوتاه (Jump code / Short circuit)**:
مقدار یک عبارت بولی را با حرکت در طول برنامه محاسبه می‌کنیم.

مثال

ارزیابی عبارت‌های بولی:

- عبارت بولی از سمت چپ به راست ارزیابی می‌شود.
- **and** و **or** شرکت‌پذیر از چپ هستند.
- پایین‌ترین اولویت مربوط به **or** و سپس **and** و آنگاه **not** دارای بالاترین اولویت است.

- **Example 1.** The translation for a or (b and (not c)) is:

  $t_1 :=$ not c

  $t_2 :=$ b and t_1

  $t_3 :=$ a or t_2

- **Example 2.** A relational expression such as a<b is equivalent to the conditional statement if a<b then 1 else 0 and its translation involves *jumps to labeled statements*:

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

۱۷-۴-۱ عبارت‌های بولی (بازنمایی عددی): ترجمه

- متغیر سراسری $nextstat$ اندیس دستورالعمل بعدی درکد سه‌آدرسی را نشان می‌دهد و مقدار آن توسط $emit$ افزایش می‌یابد.
- خصیصه‌ی op برای تعیین عملگر رابطه‌ای مورد استفاده **relop** به کار می‌رود.

PRODUCTION	SEMANTIC RULES
$E \rightarrow E_1$ 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	$E.place := newtemp;$ $emit(E.place\ ':='\ 'not'\ E_1.place)$
$E \rightarrow (E_1)$	$E.place := E_1.place$
$E \rightarrow$ 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	$E.place := newtemp;\ emit(E.place\ ':='\ '0')$

مثال

عبارت‌های بولی (بازنمایی عددی):

- $E \rightarrow id_1\ relop\ id_2$
  - **{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");}**
- **Example for translating ($a < b$ or $c < d$ and $e < f$):**

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

۲-۴-۱۷ کد پرشی برای عبارت‌های بولی

عبارت‌های بولی به دنباله‌ای از پرش‌های شرطی و غیرشرطی به $E.true$ و $E.false$ ترجمه می‌شوند.

- a $<$ b. The code is of the form:
  if a $<$ b then goto *E.true*
  goto *E.false*
- E_1orE_2. If E_1 is true then E is true, so $E_1.true = E.true$. Otherwise, E_2 must be evaluated, so $E_1.false$ is set to the label of the first statement in the code for E_2.
- E_1andE_2. Analogous considerations apply.
- not E_1. We just interchange the true and false with that for E.

تذکر: خصیصه‌های $true$ و $false$ موروثی هستند و ترجمه باید برای تعریف L-Attributed انجام شود.

مثال

عبارت‌های بولی (کد پرشی): ترجمه

PRODUCTION	SEMANTIC RULES
$E \rightarrow E_1$ or E_2	$E_1.true := E.true$; $E_1.false := newlabel$; $E_2.true := E.true$; $E_2.false := E.false$; $E.code := E_1.code \parallel gen(E_1.false\ ':') \parallel E_2.code$
$E \rightarrow E_1$ and E_2	$E_1.true := newlabel$; $E_1.false := E.false$; $E_2.true := E.true$; $E_2.false := E.false$; $E.code := E_1.code \parallel gen(E_1.true\ ':') \parallel E_2.code$
$E \rightarrow$ 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$_2$	$gen('$if$'$ id$_1.place$ relop.op id$_2.place$ $'$goto$'$ $E.true) \parallel$ $gen('$goto$'$ $E.false)$
$E \rightarrow$ true	$E.code := gen('$goto$'$ $E.true)$
$E \rightarrow$ false	$E.code := gen('$goto$'$ $E.false)$

۳-۴-۱۷ عبارت‌های بولی: کد پرشی ـ ترجمه‌ی کامل‌تر

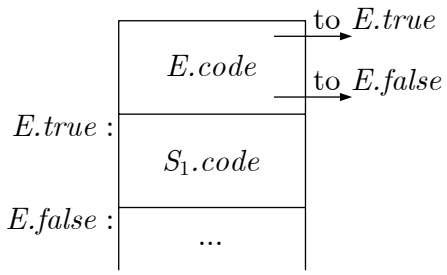
PRODUCTION	SEMANTIC RULES
$E \rightarrow E_1$ **or** E_2	$E_1.true := E.true$; $E_1.false := newlabel$; $E_2.true := E.true$; $E_2.false := E.false$; $E.code := E_1.code \| \| gen(E_1.false' :') \| \| E_2.code$;
$E \rightarrow E_1$ **and** E_2	$E_1.true := newlabel$; $E_1.false := E.false$; $E_2.true := E.true$; $E_2.false := E.false$; $E.code := E_1.code \| \| gen(E_1.true' :') \| \| E_2.code$;
$E \rightarrow$ **not** E_1	$E_1.true := E.false$; $E_1.false := E.true$; $E.code := E_1.code$;
$E \rightarrow E_1$ **relop** E_2	$E.code := E_1.code \| \| E_2.code$ $\| \| gen('$`if`$'$ $E_1.place$ **relop**.op $E_2.place$ $'$`goto`$'$ $E.true)$ $\| \| gen('$`goto`$'$ $E.false)$
$E \rightarrow$ **true**	$E.code := gen('$`goto`$'$ $E.true)$
$E \rightarrow$ **false**	$E.code := gen('$`goto`$'$ $E.false)$

۱۷-۵ دستورات کنترل جریان برنامه

- در ترجمه، فرض می‌کنیم که یک دستورالعمل کد سه‌آدرسی می‌تواند برچسب نمادین (symbolic label) داشته باشد و تابع $newlabel$ چنین برچسب‌های نمادینی را تولید می‌کند.
- به E دو برچسب نسبت داده می‌شود:
  - $E.true$، برچسب به شرط درست بودن E است.
  - $E.false$، برچسب به شرط نادرست بودن E است.
- به S خصیصه‌ی موروثی $S.next$ نسبت داده می‌شود که برچسب متصل شده به نخستین دستور پس از کد S را نشان می‌دهد.

> این روش تولید برچسب‌های وابسته به $S.next$ منجر به تولید تعداد بسیار زیادی برچسب می‌شود. روش Backpatching برچسب‌ها را تنها در صورت نیاز تولید می‌کند.

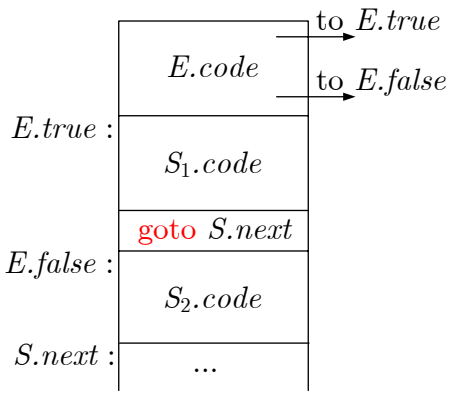
۱۷-۵-۱ دستور شرطی (if-then)



$S \rightarrow \text{if } E \text{ then } S_1$

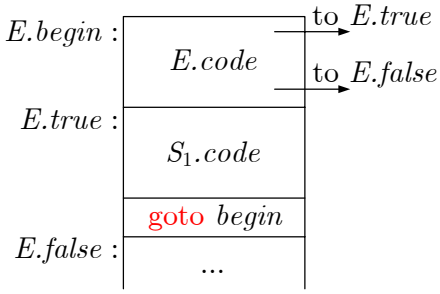
$E.true := newlabel;\ \ E.false := S.next;$
$S_1.next := S.next;$
$S.code := E.code \parallel gen(E.true\ ':') \parallel S_1.code$

۱۷-۵-۲ دستور شرطی (if-then-else)



$S \rightarrow$ if E then S_1 else S_2	$E.true := newlabel;\ \ E.false := newlabel;$ $S_1.next := S.next;\ \ S_2.next := S.next;$ $S.code := E.code \parallel gen(E.true\ ' :') \parallel S_1.code \parallel$ $gen('\text{goto}'\ S.next) \parallel$ $gen(E.false\ ' :') \parallel S_2.code$

۳-۵-۱۷ دستور تکرار (while-do)



$S \rightarrow$ while E doS_1	$E.begin := newlabel; \parallel$ $E.true := newlabel;\ \ E.false := S.next; \parallel$ $S_1.next := E.begin; \parallel$ $S.code := gen(E.begin\ ' :') \parallel E.code \parallel$ $gen(E.true\ ' :') \parallel S_1.code \parallel$ $gen('\text{goto}'\ E.begin)$

۴-۵-۱۷ دستورات جریان کنترل: ترجمه

PRODUCTION	SEMANTIC RULES
$S \rightarrow$ if E then S_1	$E.true := newlabel;\ \ E.false := S.next;$ $S_1.next := S.next;$ $S.code := E.code \parallel gen(E.true\ ' :') \parallel S_1.code$
$S \rightarrow$ if E then S_1 else S_2	$E.true := newlabel;\ \ E.false := newlabel;$ $S_1.next := S.next;\ \ S_2.next := S.next;$ $S.code := E.code \parallel gen(E.true\ ' :') \parallel S_1.code \parallel$ $gen('\text{goto}'\ S.next) \parallel$ $gen(E.false\ ' :') \parallel S_2.code$
$S \rightarrow$ while E doS_1	$E.begin := newlabel; \parallel$ $E.true := newlabel;\ \ E.false := S.next; \parallel$ $S_1.next := E.begin; \parallel$ $S.code := gen(E.begin\ ' :') \parallel E.code \parallel$ $gen(E.true\ ' :') \parallel S_1.code \parallel$ $gen('\text{goto}'\ E.begin)$

۵-۵-۱۷ دستور سوئیچ (Switch/Case)

switch expr{
case V_1**:** S_1
...
case V_k**:** S_k
default: S_d
}

دنباله‌ی ترجمه:

- ارزیابی عبارت $expr$
- یافتن مقدار تطابق‌یافته در لیست با مقدار $expr$
  (اگر هیچ تطابقی وجود نداشت، تطابق با default انجام می‌شود.)
- اجرای دستور متناظر با مقدار تطابق‌یافته

چگونگی یافتن مقدار تطابق‌یافته: آزمون ترتیبی گزینه‌ها، استفاده از یک جدول look-up، جدول درهم‌سازی (hash)، Backpatching

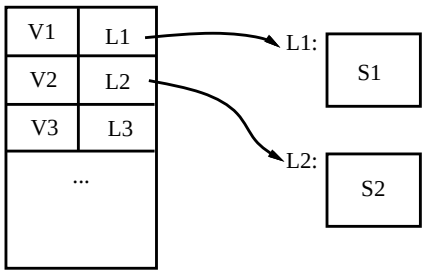
پیاده‌سازی ترجمه‌ی دستور سوئیچ (Switch/Case)

دو روش مختلف برای ترجمه:

```
code to evaluate E into t
goto test
L1: code for S1
    goto next
    ...
Lk: code for Sk
    goto next
Ld: code for Sd
    goto next
test:
    if t = V1 goto L1
    ...
    if t = Vk goto Lk
    goto Ld
next:
    ...
```

```
code to evaluate E into t
    if t <> V1 goto L1
    code for S1
    goto next
L1: if t <> V2 goto L2
    code for S2
    goto next
    ...
L(k-1): if t <> Vk goto Lk
    code for Sk
    goto next
Lk: code for Sd
next:
```

- **استفاده از یک جدول یافتن آدرس پرش به کمک یک حلقه:**



- **جدول درهم‌سازی**:
وقتی بیش از ۱۰ مدخل وجود داشت، باید از یک جدول درهم‌سازی برای یافتن مدخل مناسب جدول استفاده کرد.
- **Backpatching**:
  - یک سری از دستورات پرش که مقصد پرش آنها موقتاً نامشخص باقی گذاشته می‌شود، ایجاد می‌شود.
  - جدول برچسب‌هایی که باید مشخص شوند: هر مدخل حاوی لیستی از مکان‌هاست باید backpatch شوند.[۱]

۶-۱۷ عبارت‌های بولی در حالت مخلوط (Mixed-mode)

- عبارت‌های بولی معمولاً حاوی زیرعبارت‌های محاسباتی هستند: $(a + b) < c$
- از طرفی، اگر $true = ۱$ و $false = ۰$ باشد، آنگاه $(a < b) + (b < a)$ یک عبارت محاسباتی است.

گرامر:

$$E \rightarrow E + E \mid E \textbf{ and } E \mid E \textbf{ relop } E \mid \textbf{id}$$

- $E + E$ حاصل محاسباتی دارد، اما آرگومان‌هایش می‌توانند مخلوط باشند.
- E **and** E حاصل بولی دارد و آرگومان‌هایش باید بولی باشند.
- E **relop** E حاصل بولی دارد و آرگومان‌هایش می‌توانند مخلوط باشند.
- **id** از نوع محاسباتی فرض می‌شود.
- برای تولید کد از خصیصه‌ی ترکیبی $E.type$ استفاده می‌کنیم که مقدار آن $arith$ یا $bool$ است.
- عبارت‌های بولی خصیصه‌های موروثی $E.true$ و $E.false$ را دارند که برای کد پرش مناسب است.
- عبارت‌های محاسباتی خصیصه‌ی ترکیبی $E.place$ را خواهند داشت که برای نگهداری مقدار (موقتی) E استفاده می‌شود.
- متغیر سراسری $nextstat$ اندیس کد سه‌آدرسی بعدی را نشان می‌دهد که مقدار آن توسط gen افزایش می‌یابد.

[۱] از همین روش می‌توان برای پیاده‌سازی دستور Goto هم استفاده کرد.

۱۷-۶-۱ عبارت‌های مخلوط: ترجمه

$E \rightarrow E_1 + E_2$	$E.type := arith;$ if $E_1.type := arith$ and $E_2.type := arith$ then begin $E.place := newtemp;$ $E.code := E_1.code \parallel E_2.code \parallel$ $gen(E.place\,' :=' \; E_1.place\,' +' \; E_2.place)$ end esle if $E_1.type := arith$ and $E_2.type := bool$ then begin $E.place := newtemp;$ $E_2.true := newlabel; E_2.false := newlabel;$ $E.code := E_1.code \parallel E_2.code \parallel$ $gen(E_2.true' :' \; E.place\,' :=' \; E_1.place + 1) \parallel$ $gen('\text{goto}' nextstat + 1) \parallel$ $gen(E_2.false' :' \; E.place\,' :=' \; E_1.place)$ end else if ...

۱۷-۷ تولید کد برای آرایه‌ها

- **1-D array: $A[i]$. Assume**
  - **lower bound in address =** low
  - **element data width =** w
  - **starting address =** $base$
- **Address for $A[i]$**
  - $= base + (i - low) * w$
  - $= i * w + (base - low * w)$
  - **The value of $(base - low * w)$ can be computed at compile time.**
- **2-D array $A[i_1, i_2]$.**
  - **Row major:**
    - $A[1,1], A[1,2], A[1,3], A[2,1], A[2,2], \ldots$
    - ***Advantage: A[i,j] = A[i][j].***
    - ***A[1] means the first row.***
  - **Column major:**
    - $A[1,1], A[2,1], A[1,2], A[2,2], A[1,3], \ldots$

- **Address for** $A[i_1, i_2]$
  - $= base + ((i_1 - low_1) * n_2 + (i_2 - low_2)) * w$
  - $= (i_1 * n_2 + i_2) * w + (base - low_1 * n_2 * w - low_2 * w)$
  - n_2 **is the number of elements in a row.**
  - low_2 **is the lower bound of the second coordinate.**
  - **The value** $(base - low_1 * n_2 * w - low_2 * w)$ **can be computed at compiler time.**
- **Similar method for multi-dimensional arrays. Address for** $A[i_1, i_2, \ldots, i_k]$
  - $= (i_1 * \Pi_{i=2}^{k} n_i i_2 * \Pi_{i=3}^{k} n_i + \cdots + i_k) * w + (base - low_1 * \Pi_{i=2}^{k} n_i * w - \cdots - low_k * w)$
  - n_i **is the number of elements in the** i**th coordinate.**
  - low_i **is the lower of the** i**th coordinate.**
  - **The values** $\Pi_{i=j}^{k}$, $2 \leq j \leq k-1$, **and** $(base - low_1 * \Pi_{i=2}^{k} n_i * w - \cdots - low_k * w)$ **can be computed at compile time.**

- $L \rightarrow Elist\ ]$
  - **{L.place := newtemp();**
  - **L.offset := newtemp();**
  - **emit(L.offset, ":=",Elist.elesize, "*",Elist.place);}**
- $L \rightarrow id$
  - **{L.place := id.place; L.offset := null}**
- $Elist \rightarrow Elist_1, E$
  - **{ t := newtemp();**
  - **m :=** $Elist_1$**.ndim + 1;**
  - **emit(t, ":=",**$EList_1.place$**, "*",limit(**$Elist_1$**.array,m));**
  - **emit(t,":=",t,"+",E.place);**
  - **Elist.array :=** $Elist_1$**.array;**
  - **Elist.place := t;**
  - **EList.ndim := m; }**
- $Elist \rightarrow id\ [\ E$
  - **{Elist.place := E.place;**
  - **Elist.ndim := 1; p := lookup(id.name,symbol_table);**
  - **Elist.elesize := p.size;**
  - **EList.array := p.place}**

۸-۱۷ تولید کد برای فراخوانی روال‌ها

- باید فضایی برای رکورد فعالیت روال فراخوانی‌شده تخصیص داده شود.
- آرگومان‌ها ارزیابی می‌شوند و در یک مکان معلوم برای روال فراخوانی‌شده آماده می‌شوند.
- وضعیت فعلی ماشین ذخیره می‌شود.
- وقتی روال برمی‌گردد،
  - مقدار بازگشتی در یک فضای معلوم قرار داده می‌شود.
  - رکورد فعالیت بازیابی می‌شود.

مثال

تولید کد برای فراخوانی روال‌ها

- **Example:**
  - $S \rightarrow$ **call** $id(Elist)$
    - ▷ ***{for each item p on the queue Elist.queue do***
    - ▷ ***emit("PARAM", q);***
    - ▷ ***emit("call", id.place);}***
  - $Elist \rightarrow Elist, E$
    - ▷ ***{append E.place to the end of Elist.queue}***
  - $Elist \rightarrow E$
    - ▷ ***{initialize $Elist.queue$ to contain only E.place}***
- **Idea:**
  - **Use a queue to hold parameters, then generate codes for parameters.**
  - **May have code like:**
    - ▷ ***code for E_1, store in t_1***
    - ▷ $\cdots$
    - ▷ ***code for E_k, store in t_k***
    - ▷ ***PARAM t_1***
    - ▷ $\cdots$
    - ▷ ***PARAM t_k***
    - ▷ ***call p***

تمرین

۱. با توجه به کنش‌های معنایی نوشته شده برای دستورات یک برنامه در کلاس، برای دستور **repeat** S **while** E کنش‌های معنایی لازم را بنویسید.

۲. کد سه‌آدرسی را برای عبارت زیر تولید کنید:

$$\textbf{if } a = b \textbf{ and } c = d \textbf{ or } e = f \textbf{ then } x = z$$