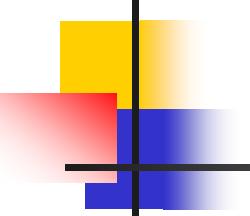


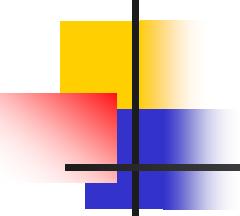
7.1 中间代码的形式

- **中间代码的作用**
 - 过渡：经过语义分析被译成中间代码序列
- **中间代码的形式**
 - 中间语言的语句
- **中间代码的优点**
 - 形式简单、语义明确、独立于目标语言
 - 便于编译系统的实现、移植、代码优化
- **常用的中间代码**
 - 语法树(6.3.5节)
 - 逆波兰表示、三地址码(三元式和四元式)、DAG图表示



7.1.1 逆波兰表示

- 中缀表达式的计算顺序不是运算符出现的自然顺序，而是根据运算符间的优先关系来确定的，因此，从中缀表达式直接生成目标代码一般比较麻烦。
- 波兰逻辑学家J. Lukasiewicz于1929年提出了后缀表示法，其优点为：表达式的运算顺序就是运算符出现的顺序，它不需要使用括号来指示运算顺序。



7.1.1 逆波兰表示

- 例7.1 下面给出的是一些表达式的中缀、前缀和后缀表示。

中缀表示

$a+b$

$a*(b+c)$

$(a+b)*(c+d)$

$a:=a*b+c*d$

前缀表示

$+ab$

$*a+bc$

$*+ab+cd$

$:=a+*ab*c d$

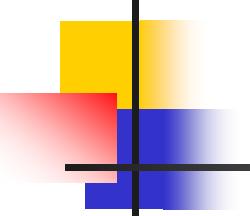
后缀表示

$ab+$

$abc+*$

$ab+cd+*$

$abc*bd*+:=$



7.1.2 三地址码

- 所谓三地址码，是指这种代码的每条指令最多只能包含三个地址，即两个操作数地址和一个结果地址。
- 如 $x+y*z$ 三地址码为： $t_1 := y*z$ $t_2 := x+t_1$
- 三地址码中地址的形式：
 - 名字、常量、编译器生成的临时变量。

7.1.2 三地址码

- 例7.2 赋值语句 $a := (-b) * (c+d) - (c+d)$ 的三地址码
如图7.1所示

```
t1 := minus b
t2 := c+d
t3 := t1*t2
t4 := c+d
t5 := t3-t4
a := t5
```

图7.1 $a := (-b) * (c+d) - (c+d)$ 的三地址码

四元式

- 四元式是一种比较常用的中间代码形式，它由四个域组成，分别称为op、arg1、arg2和result。op是一个一元或二元运算符，arg1和arg2分别是op的两个运算对象，它们可以是变量、常量或编译器生成的临时变量，运算结果则放入result中。

	op	arg ₁	arg ₂	result
0	minus	b		t ₁
1	+	c	d	t ₂
2	*	t ₁	t ₂	t ₃
3	+	c	d	t ₄
4	-	t ₃	t ₄	t ₅
5	assign	t ₅		a
		...		

三元式

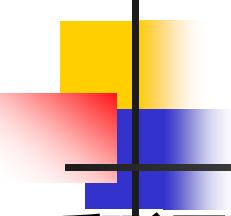
- 为了节省临时变量的开销，有时也可以使用只有三个域的三元式来表示三地址码。三元式的三个域分别称为op, arg₁和arg₂, op, arg₁和arg₂的含义与四元式类似，区别只是arg₁和arg₂可以是某个三元式的编号(图7.2(b)中用圆括号括起来的数字)，表示用该三元式的运算结果作为运算对象。

	op	arg ₁	arg ₂
0	minus	b	
1	+	c	d
2	*	(0)	(1)
3	+	c	d
4	-	(2)	(3)
5	assign	a	(4)
		...	

生成三地址码的语法制导定义

产生式	语义规则
$S \rightarrow id := E$	$S.code := E.code \parallel \text{gencode}(id.addr' := 'E.addr)$
$E \rightarrow E_1 + E_2$	$E.addr := \text{newtemp}; E.code := E_1.code \parallel E_2.code \parallel \text{gencode}(E.addr' := 'E_1.addr' + ' E_2.addr)$
$E \rightarrow E_1 * E_2$	$E.addr := \text{newtemp}; E.code := E_1.code \parallel E_2.code \parallel \text{gencode}(E.addr' := 'E_1.addr' * ' E_2.addr)$
$E \rightarrow -E_1$	$E.addr := \text{newtemp}; E.code := E_1.code \parallel \text{genode}(E.addr' := ''uminus' E_1.addr)$
$E \rightarrow (E_1)$	$E.addr := E_1.addr; E.code := E_1.code$
$E \rightarrow id$	$E.addr := id.addr; E.code := ''$
$E \rightarrow num$	$E.addr := num.val; E.code := ''$

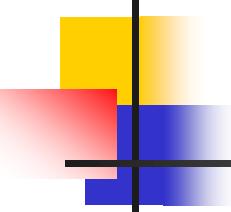
属性 `code` 表示生成的代码



7.3 赋值语句的翻译

翻译的需求

- 充分了解各种语言现象的语义
 - 包括：控制结构、数据结构、单词
 - 充分了解它们的实现方法
- 目标语言的语义
 - 了解中间代码的语义
 - 了解运行环境



辅助子程序与语义属性设置

■ 辅助子程序

- `gencode(code)`, `emit(code)`: 产生一条中间代码
- `newtemp`: 产生新的临时变量
- `lookup`: 检查符号表中是否出现某名字

■ 语义属性设置

- 中间代码序列: `code`
- 地址: `addr`
- 下一条四元式序号: `nextquad`

7.3.1 简单赋值语句的翻译

$S \rightarrow id := E$

```
{p := lookup(id.name);  
if p ≠ nil then  
    gencode( p, ':=', E.addr)  
else error }
```

$E \rightarrow E_1 + E_2 \{E.addr := newtemp;
emit(E.addr, ':=', E_1.addr, '+', E_2.addr)$

$E \rightarrow -E_1 \{E.addr := newtemp;
emit(E.addr, ':=', 'uminus', E_1.addr)$

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

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

x:=-a*b+c翻译为以下三地址码序列:
t1:=minus a
t2:=t1 * b
t3:=t2 + c
t4:=t3