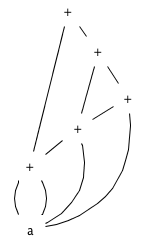
## 第六章作业

### 练习6.1.2

3) a + a + ( a + a + a + ( a + a + a + a ) )



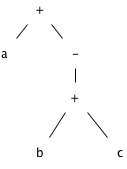
子表达式的值编码如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | id | a |  |
| 2 | + | 1 | 1 |
| 3 | + | 2 | 1 |
| 4 | + | 3 | 1 |
| 5 | + | 3 | 4 |
| 6 | + | 2 | 5 |

### 练习6.2.1

**a + - ( b + c )**

1. 抽象语法树



1. 四元式序列

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | op | arg1 | arg2 | result |
| 0 | + | b | c | t1 |
| 1 | uminus | t1 |  | t2 |
| 2 | + | a | t2 | t3 |

1. 三元式序列

|  |  |  |  |
| --- | --- | --- | --- |
|  | op | arg1 | arg2 |
| 0 | + | b | c |
| 1 | uminus | (0) |  |
| 2 | + | a | (1) |

1. 间接三元式序列

|  |  |
| --- | --- |
|  | statment |
| 100 | (0) |
| 101 | (1) |
| 102 | (2) |

|  |  |  |  |
| --- | --- | --- | --- |
|  | op | arg1 | arg2 |
| 0 | + | b | c |
| 1 | uminus | (0) |  |
| 2 | + | a | (1) |

### 练习6.3.1

SDT

S -> {top = new Evn(); offset = 0;}

D

D -> T id; {top.put(id.lexeme, T.type, offset);

offset += T.width}

D1

D -> ε

T -> int {T.type = interget; T.width = 4;}

T -> float {T.type = float; T.width = 8;}

T -> record '{'

{Evn.push(top), top = new Evn();

Stack.push(offset), offset = 0;}

D '}' {T.type = record(top); T.width = offset;

top = Evn.top(); offset = Stack.pop();}

标识符类型和相对地址

line id type offset Evn

1) x float 0 1

2) x float 0 2

2) y float 8 2

2) p record() 8 1

3) tag int 0 3

3) x float 4 3

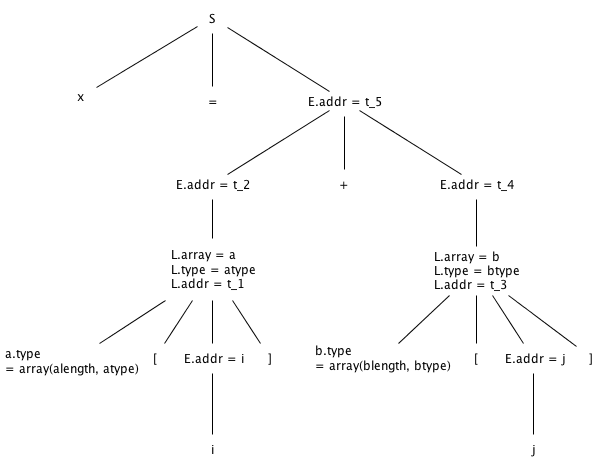
3) y float 12 3

3) q record() 24 1

### 练习6.4.3

1) x = a[i] + b[j]

语法分析树如下：



三地址代码如下：

t\_1 = i \* awidth

t\_2 = a[t\_1]

t\_3 = j \* bwidth

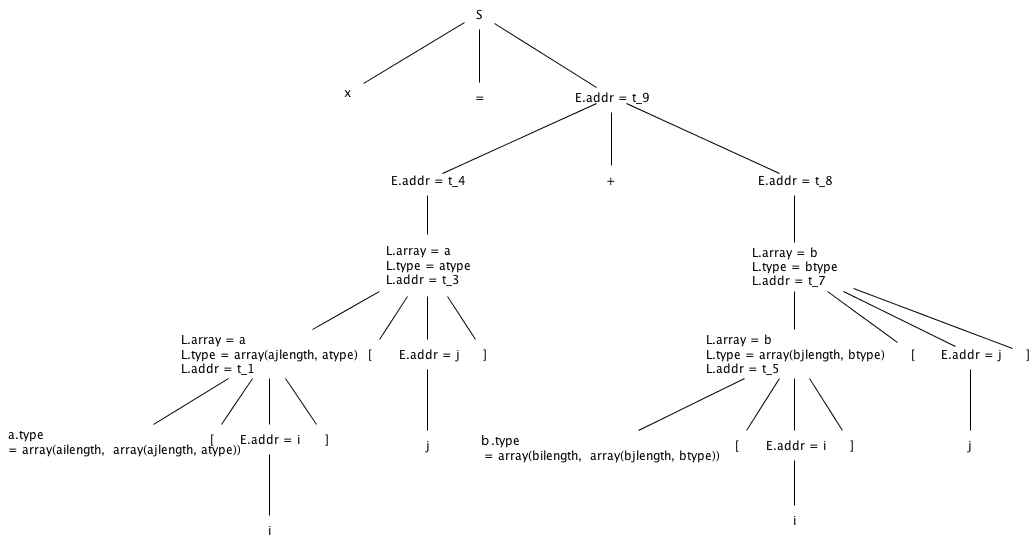
t\_4 = b[t\_3]

t\_5 = t\_2 + t\_4

x = t\_5

2) x = a[i][j] + b[i][j]

语法分析树如下：



三地址代码如下：

t\_1 = i \* ai\_width

t\_2 = j \* aj\_width

t\_3 = t\_1 + t\_2

t\_4 = a[t\_3]

t\_5 = i \* bi\_width

t\_6 = j \* bj\_width

t\_7 = t\_5 + t\_6

t\_8 = b[t\_7]

t\_9 = t\_4 + t\_8

x = t\_9

### 练习6.5.1

1) x = s + c

t1 = (int) s

t2 = (int) c

t3 = t1 + t2

x = (float) t3

2) i = s + c

t1 = (int) s

t2 = (int) c

i = t1 + t2

3) x = ( s + c ) \* ( t + d )

t1 = (int) s

t2 = (int) c

t3 = t1 + t2

t4 = (int) t

t5 = (int) d

t6 = t4 + t5

t7 = t3 \* t6

x = (float) t7

### 练习6.6.1

产生式 语义规则

S -> repeat S1 while B S1.next = newlabel()

B.true = newlabel()

B.false = S.next

S.code = label(B.true) || S1.code|| label(S1.next) || B.code

S -> for (S1; B; S2) S3 S1.next = newlabel()

B.true = newlabel()

B.false = S.next

S2.next = S1.next

S3.next = newlabel()

S.code = S1.code|| lable(S1.next) || B.code|| lable(B.true) || S3.code

|| label(S3.next) || S2.code|| gen('goto', S1.next)

### 练习6.7.1

1) a == b && ( c == d || e == f )

各个子表达式的truelist和falselist如下图所示：



具体分析过程如下：

首先，对于a==b按照E → id1 relop id2的语义动作进行规约，产生如下指令：

100: if a==b goto -

101: goto -

对于E→ E1 and M E2中的M记录了E2的入口，所以M.i={102}；

对于c==d按照E → id1 relop id2的语义动作进行规约，产生如下指令：

102: if c==d goto -

103: goto -

对于E→ E1 or M E2中的M记录了E2的入口，所以M.i={104}

对于e==f按照E → id1 relop id2的语义动作进行规约，产生如下指令：

104: if e==f goto -

105: goto -

接着，用产生式E→ E1 or M E2进行规约，执行以下动作：

{

backpatch( E1.falselist, M.quad );

E.truelist := merge(E1.truelist , E2.truelist );

E.falselist := E2.falselist

}

先执行backpatch( E1.falselist, M.quad )，即backpatch( {103}, 104 )，所以有

100: if a==b goto -

101: goto -

102: if c==d goto -

103: goto 104

104: if e==f goto -

105: goto -

接着执行E.truelist := merge(E1.truelist , E2.truelist )，所以E.t={102,104}，

接着执行E.falselist := E2.falselist，所以E.f={105}

然后用产生式E→( E1 )进行规约，执行以下动作：

{

E.truelist := E1.truelist;

E.falselist := E1.falselist

}

直接把各自的truelist和falselist拷贝过去，所以有E.t={102,104}，E.f={105}

然后用产生式E→E1 and M E2进行规约，执行以下动作：

{

backpatch( E1.truelist, M.quad );

E.truelist := E2.truelist;

E.falselist := merge( E1.falselist , E2.falselist )

}

先执行backpatch( E1.truelist, M.quad )即backpatch({100},102 )，所以有：

100: if a==b goto 102

101: goto -

102: if c==d goto -

103: goto 104

104: if e==f goto -

105: goto -

接着执行E.truelist := E2.truelist，所以E.t={102,104}

接着执行E.falselist := merge( E1.falselist , E2.falselist )，所以E.f={101,105}

如果分别用L1和L2表示整个表达式的真、假两个出口，则最终生成的指令如下：

100: if a==b goto 102

101: goto L2

102: if c==d goto L1

103: goto 104

104: if e==f goto L1

105: goto L2

2) ( a == b || c == d ) || e == f

与1)类似，可以得到各个子表达式的truelist和falselist如下：



如果分别用L1和L2表示整个表达式的真、假两个出口，则最终生成的指令如下：

100: if a==b goto L1

101: goto 102

102: if c==d goto L1

103: goto 104

104: if e==f goto L1

105: goto L2

3) ( a == b && c == d ) && e == f

与1)类似，可以得到各个子表达式的truelist和falselist如下：



如果分别用L1和L2表示整个表达式的真、假两个出口，则最终生成的指令如下：

100: if a==b goto 102

101: goto L2

102: if c==d goto 104

103: goto L2

104: if e==f goto L1

105: goto L2