Assignment 8

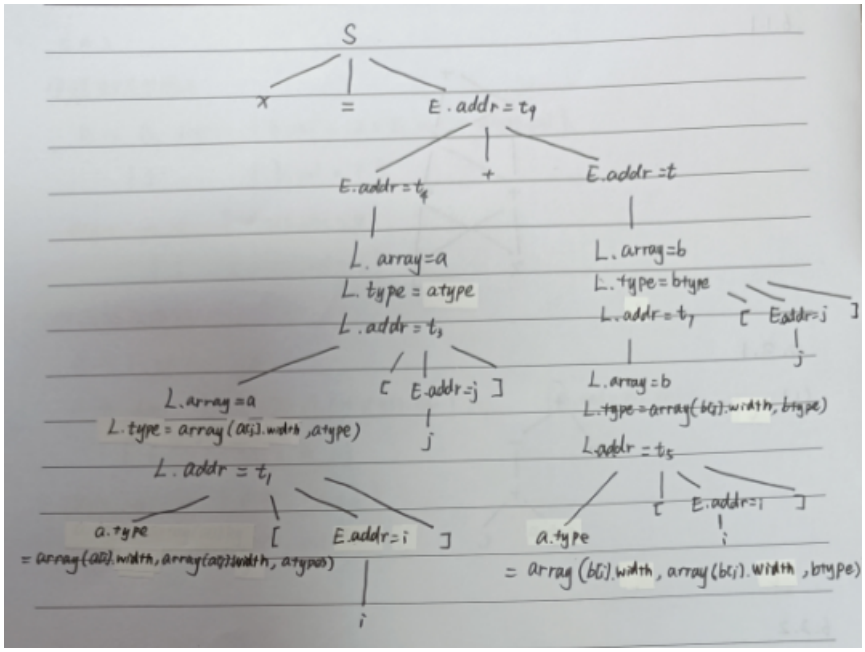
6.4.3(2)

练习**6.4.3**：使用图6-22所示的翻译方案来翻译下列赋值语句：

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

答：

语法分析树：



三地址代码：

```
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
```

6.4.6(2)

练习**6.4.6**：一个按行存放的整数数组A［i，j］的下标i的范围为1～10，下标j的范围为1～20。每个整数占4个字节。假设数组A从0字节开始存放，请给出下列元素的位置：

2）A［10，8］

答：

((10 - 1) * 20 + (8 - 1)) * 4 = 748

6.4.8(2)

练习**6.4.8**：一个按行存放的实数型数组A［i，j，k］的下标i的范围为1～4，下标j的范围为0～4，且下标k的范围为5～10。每个实数占8个字节。假设数组A从0字节开始存放。计算下列元素的位置。

2）A［1，2，7］

答:

((1 - 1) * 5 * 6 + (2 - 0) * 6 + (7 - 5)) * 8 = 112

6.5.2

练习**6.5.2**：像Ada中那样，我们假设每个表达式必须具有唯一的类型，但是我们根据一个子表达式本身只能推导出一个可能类型的集合。也就是说，将函数E_1应用于参数E_2（其文法产生式为$E \rightarrow E_1(E_2)$）有如下规则：

$$E.type = \{t \mid 对E_2.type中的某个s，s \rightarrow t在E_1.type中\}$$

描述一个可以确定每个子表达式的唯一类型的语法制导定义（SDD）。它首先使用属性type，按照自底向上的方式综合得到一个可能类型的集合。在确定了整个表达式的唯一类型之后，自顶向下地确定属性unique的值，这个属性表示各个子表达式的类型。

- **本题文法：S→id=E E→E1+E2 | -E1 | (E1) | id | E1(E2)**

答:

```
S -> id = E          gen(top.get(id.lexeme) '=' E.addr);

E -> E1 + E2         E.type = max(E1.type, E2.type);
                     a1 = widen(E1.addr, E1.type, E.type);
                     a2 = widen(E2.addr, E2.type, E.type);
                     E.addr = new Temp();
                     gen(E.addr '=' a1 '+' a2);

    | -E1            E.addr = new Temp();
                     gen(E.addr '=' 'minus' E1.addr);

    | (E1)           E.addr = E1.addr;

    | id             E.addr = top.get(id.lexeme);

    | E1(E2)         E.type = {t|对E2.type中的某个s，s→t在E1.type中}
                     E.addr = new Temp();
                     gen(E.addr '=' E1.addr '(' E2.addr ')');
```

6.6.1

练习**6.6.1**：在图6-36的语法制导定义中添加处理下列控制流构造的规则：

1）一个repeat语句，**repeat** S **while** B。

！2）一个for循环语句，**for**（S_1；B；S_2）S_3。

答:

1)

```
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
```

2)

```
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)
```