实验 5 语义分析和中间代码生成器

语义分析器分两部分，第一部分为赋值表达式-实验 4（必做），第二部分为数组、布尔表达式和控制语句-实验 5（选做）。

要求

参考课本 6.4.3 和 7.3、7.4、7.5，实现递归下降翻译器。

注意

数据结构：

四元式：数组

跳转语句的四元式的第 4 个域需回填，请考虑如何存储每个四元式。

翻译模式与步骤

说明：下列翻译模式中，黑色字体为赋值表达式的翻译，与实验 4 相同，即第 1 步与实验 4 相同。

按以下顺序完成语义分析

1. 赋值语句的翻译

说明：

设文法符号为 X，其属性如下：

- X.place：存放 X 值的变量的名字；
- 函数 emit( )：将生成的四元式语句发送到输出文件中；
- 函数 newtemp( ): 生成一个临时变量的名字，如 t1。

测试：

输入： a=6/b+5*c-d;

输出：

0: /, 6, b, t1
1: *, 5, c, t2
2: +, t1, t2, t3
3: -, t3, d, t4
4: =, t4, -, a

2. 数组的翻译

说明：

设文法符号为 X，其属性如下：

X.inArray：指向符号表中相应数组名字表项的指针，若不使用符号表，则 X.inArray 即为数组的名字。

X.inNdim：下标表达式的个数，及维数

X.inPlace：存放由 Elist 中的下标表达式计算出来的值

X.array：指向符号表中相应数组名字表项的指针

X.place：若 X 为简单名字，X.place 为指向符号表中相应此名字表项的指针；若 X 为数组名字，X.place 为

数组地址中常量部分

X.offset：若 X 为简单名字，X.offset 为 null；若 X 为数组名字，X.offset 为数组地址中变量部分

limit(array, j)：返回 nj，即 array 数组的第 j 维长度，如 10、20 等。本实验中，就用字符串 nj 表示，如 n1、n2、n3 等

测试 1：

输入： x=A[i];

输出：

```
0:  -, A, C, t1
1:  *, i, w, t2
2:  =[], t1[t2], -, t3
3:  =, t3, -, x
```

测试 2：

输入： x=A[i, j];

输出：

```
0:  *, i, n2, t1
1:  +, t1, j, t1
2:  -, A, C, t2
3:  *, t1, w, t3
4:  =[], t2[t3], -, t4
5:  =, t4, -, x
```

3. 布尔表达式的翻译

测试：

输入：

```
while(a<b)
    if(c)
        x=y+z;
    else
        x=y-z;
a=y;
```

输出：

```
0:  j<, a, b, -
1:  j, -, -, -
2:  jnz, c, -, -
3:  j, -, -, -
4:  +, y, z, t1
5:  =, t1, -, x
6:  j, -, -, -
7:  -, y, z, t2
8:  =, t2, -, x
9:  j, -, -, -
10: =, y, -, a
```

4. 控制语句的翻译

说明：

merge(p1, p2)：把以 p1 和 p2 为链首的两条链合并为一，将 p2 的链尾的第 4 区段改为 p1，合并后的链首为 p2，回送合并后的链首

测试：

输入：

```
while(a<b)
    if(c)
        x=y+z;
    else
        x=y-z;
a=y;
```

输出：

```
0:   j<, a, b, 2
1:   j, -, -, 10
2:   jnz, c, -, 4
3:   j, -, -, 7
4:   +, y, z, t1
5:   =, t1, -, x
6:   j, -, -, 0
7:   -, y, z, t2
8:   =, t2, -, x
9:   j, -, -, 0
10:  =, y, -, a
```

以下语义规则中：

黑色字体：赋值表达式

蓝色字体：数组

绿色字体：布尔表达式

红色字体：控制语句

消除左递归文法	
stmts ⟶ *stmt* *rest0*	{rest0.inNextlist=stmt.nextlist} {stmts.nextlist=rest0.nextlist}
rest0 ⟶ *m stmt* *rest0*$_1$	{backpatch(rest0.inNextlist, m.quad); rest0$_1$.inNextlist=stmt.nextlist} {rest0.nextlist=rest0$_1$.nextlist}
rest0 ⟶ **ε**	{rest0.nextlist=rest0.inNextlist}

stmt⟶*loc*=*expr* ;	{if(loc.offset=null)
	emit('=,' expr.place ', - ,' loc.place);
	else
	emit('[]=,' expr.place ', - ,' loc.place '[' loc.offset ']');
	stmt.nextlist=makelist()}

stmt⟶**if(***bool***)** m_1 $stmt_1$ n **else** m_2 $stmt_2$	{backpatch(bool.truelist, m_1.quad); backpatch(bool.falselist, m_2.quad); stmt.nextlist= merge($stmt_1$.nextlist, n.nextlist, m_2.nextlist)}

stmt⟶ **while(**m_1 *bool***)** m_2 $stmt_1$	{backpatch($stmt_1$.nextlist, m_1.quad);
	backpatch(bool.truelist, m_2.quad);
	stmt.nextlist=bool.falselist;
	emit('j, -, -, ' m_1.quad)}

m⟶ε	{m.quad=nextquad}
n⟶ε	{n.nextlist=makelist(nextquad); emit('j, -, -, 0')}
loc⟶**id** *resta*	{resta.inArray=id.place} {loc.place=resta.place; loc.offset=resta.offset}

resta ⟶ [
elist
]
	{elist.inArray=resta.inArray}
	
	{resta.place=newtemp();
	emit('-,' elist.arry ',' C ',' resta.place);
	resta.offset=newtemp();
	emit('*, ' w ',' elist.offset ',' resta.offset);
	}

resta ⟶ ε	{resta.place=resta.inArray; resta.offset=null}
elist ⟶ *expr* *rest1*	{rest1.inArray=elist.inArray; rest1.inNdim=1; rest1.inPlace=expr.place} {elist.array=rest1.array; elist.offset=rest1.offset}

rest1⟶ ,
expr
 	
rest1$_1$
	{t=newtemp();
	m=rest1.inNdim+1;
	emit('*,' rest1.inPlace ',' limit(rest1.inarray,m) ',' t);
	emit('+,' t ',' expr.place ',' t);
	rest1$_1$.inArray=rest1.inArray;
	rest1$_1$.inNdim=m;
	rest1$_1$.inNplace=t}
	{rest1.array=rest1$_1$.array;

	rest1.offset=rest1$_1$.offset}
rest1 ⟶ Ɛ	{rest1.array=rest1.inArray; rest1.offset=rest1.inPlace}
bool ⟶ *equality*	{bool.truelist=equality.truelist bool.falselist=equality.falselist }
equality ⟶ *rel* *rest4*	{rest4.inTruelist=rel.truelist rest4.inFalselist=rel.falselist} {equality.truelist=rest4.truelist equality.falselist=rest4.falselist}
rest4 ⟶ == *rel rest4*$_1$	
rest4 ⟶ != *rel rest4*$_1$	
rest4 ⟶ Ɛ	{rest4.truelist=rest4.inTruelist rest4.falselist=rest4.inFalselist}
rel ⟶ *expr* *rop_expr*	{rop_expr.inPlace=expr.place} {rel.truelist=rop_expr.truelist rel.falselist=rop_expr.falselist}

rop_expr ⟶ <*expr*	{rop_expr.truelist=makelist(nextquad);
	rop_expr.falselist=makelist(nextquad+1);
	emit('j<,' rop_expr.inPlace ',' expr.place ', -');
	emit('j, -, -, -')}

rop_expr ⟶ <=*expr*	{rop_expr.truelist=makelist(nextquad);
	rop_expr.falselist=makelist(nextquad+1);
	emit('j<=,' rop_expr.inPlace ',' expr.place ', -');
	emit('j, -, -, -')}

rop_expr ⟶ >*expr*	{rop_expr.truelist=makelist(nextquad);
	rop_expr.falselist=makelist(nextquad+1);
	emit('j>,' rop_expr.inPlace ',' expr.place ', -');
	emit('j, -, -, -')}

rop_expr ⟶ >=*expr*	{rop_expr.truelist=makelist(nextquad);
	rop_expr.falselist=makelist(nextquad+1);
	emit('j>=,' rop_expr.inPlace ',' expr.place ', -');
	emit('j, -, -, -')}

rop_expr ⟶ Ɛ	{rop_expr.truelist=makelist(nextquad);
	rop_expr.falselist=makelist(nextquad+1);
	emit('jnz,' rop_expr.inPlace ', -, -');
	emit('j, -, -, -')}

expr ⟶ *term* *rest5*	{rest5.in=term.place} {expr.place=rest5.place}
rest5 ⟶ +*term* *rest5*$_1$	{rest5$_1$.in=newtemp(); emit('+,' rest5.in ',' term.place ',' rest5$_1$.in)} {rest5.place =rest5$_1$ .place}

rest5 ⟶ **-***term* *rest5*$_1$	{rest5$_1$.in=newtemp(); emit('-,' rest5.in ',' term.place ',' rest5$_1$.in)} {rest5.place =rest5$_1$ .place}
rest5 ⟶ **ε**	{rest5.place = rest5.in}
term⟶ *unary* *rest6*	{rest6.in = unary.place} {term.place = rest6.place}
rest6 ⟶ ***unary*** *rest6*$_1$	{rest6$_1$.in=newtemp(); emit('*,' rest6.in ',' unary.place ',' rest6$_1$.in)} {rest6.place = rest6$_1$ .place}
rest6 ⟶ **/***unary* *rest6*$_1$	{rest6$_1$.in=newtemp(); emit('/,' rest6.in ',' unary.place ',' rest61.in)} {rest6.place = rest6$_1$ .place}
rest6 ⟶ **ε**	{rest6.place = rest6.in}
unary ⟶ *factor*	{unary.place = factor.place}
factor⟶ **(***expr***)**	{unary.place = expr.place}
factor⟶*loc*	{if(loc.offset=null) factor.place = loc.place else {factor.place=newtemp(); emit('=[],' loc.place '[' loc.offset ']' ', -,' factor.place)}}
factor⟶**num**	{factor.place = num.value}