实验 4 语义分析和中间代码生成器-赋值表达式

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

要求

参考课本 6.4.3 和 7.3，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

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；
- 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。在实验 4 中，因不处理数组，则 X.inArray 为存放 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

文法	语义动作
$stmts \longrightarrow stmt$ $rest0$	
$rest0 \longrightarrow stmt$ $rest0_1$	

rest0 ⟶ **ε**	

stmt⟶*loc*=*expr* ;	{
	emit('=,' expr.place ', - ,' loc.place);
	}

stmt⟶**if(***bool***)** $stmt_1$ **else** $stmt_2$	
stmt⟶ **while(** *bool***)** $stmt_1$	
loc⟶**id** *resta*	{resta.inArray=id.place} {loc.place=resta.place; }
resta ⟶ [*elist*]	
resta ⟶ ε	{resta.place=resta.inArray; }
elist ⟶ *expr* *rest1*	

rest1 ⟶ Ɛ	
bool ⟶ *equality*	
equality ⟶ *rel* *rest4*	
rest4 ⟶ == *rel* $rest4_1$	
rest4 ⟶ != *rel* $rest4_1$	
rest4 ⟶ **Ɛ**	
rel ⟶ *expr* *rop_expr*	
rop_expr ⟶ <*expr*	
rop_expr ⟶ <=*expr*	
rop_expr ⟶ >*expr*	
rop_expr ⟶ >=*expr*	
rop_expr ⟶ **Ɛ**	

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*	{
	factor.place = loc.place
	
	}

factor ⟶**num**	{factor.place = num.value}