中间代码代码生成实验报告

181250086 刘佳月 181250086@smail.nju.edu.cn

实现功能

将通过词法分析、语法分析、语义分析的 C—源代码翻译为中间代码（三地址代码），并以线性结构输出。

关键数据结构

1. Operand_表示操作数

```
struct Operand_{
    enum{TMP_VAR, VARIABLE, CONSTANT, LABEL, NONE}kind;
    union{
        int intVal;//标签序号
        string strVal;//函数名称 常数的字符串 变量名 临时变量名t4
    }u;//操作数的属性不同
};
```

2.

```
struct InterCode_{//单条中间代码。将中间代码划分为19种。
    enum{W_LABEL, W_FUNCTION, W_ASSIGN, W_ADD, W_SUB, W_MUL, W_DIV,
    W_GET_ADDR, W_GET_VAL, W_VAL_GOT,
    W_GOTO, W_IFGOTO, W_RETURN, W_DEC, W_ARG, W_CALL, W_PARAM, W_READ, W_WRITE}kind;
    union{
        struct{Operand op;}Single;
        struct{Operand op1, op2, result;}Double;
        struct{Operand x,y,label; string relop;}Three;
        struct{Operand left,right;}Assign;
        struct{Operand op;int size;}Dec;
    }u;//不同类型的中间代码有不同个数和种类的操作数
    InterCode prev;
    InterCode next;
};
```

3. 用双向链表存储中间代码，便于用 interInsert(InterCode)函数来插入生成的中间代码。

实现步骤

根据产生式，为主要的非终结符编写函数。语句和条件表达式、函数调用、函数参数的翻译按照书上不再赘述，基本表达式 Exp 的翻译要注意传 place。翻译数组时，分两种情况， Exp1 ASSIGNOP Exp2 的 Exp1，和 Exp->Exp LB Exp RB，将中间代码类型按需设为 W_GET_ADDR, W_GET_VAL, W_VAL_GOT 即可。

优化

- 对于常数 a，直接用#a 来表示，而不是 t1:=#a，再用 t1
- 对于图中所示的这种多个 LABEL 指示的是同一位置的情况，我们仅保留了第一个 label，将下面的 label 都删除，并且将 GOTO 目标是下面几条 label 的 goto 指令的 GOTO 目标改为第一个 label。

```
WRITE #31
LABEL label33 :
LABEL label30 :
LABEL label21 :
LABEL label3 :
RETURN #0
```

印象深刻的 bug

对于 Stmt->Exp SEMI 产生式，书上写的是 trans_Exp(Exp,NULL)。我一开始就传了 NULL，但造成了很多奇怪数不清搞不明白的段错误，所以就增加 Operand_::NONE 类型，立马 AC