编译原理实验三报告

周羽萱 211220074 1813156367@qq.com

一. 实现功能

所有的必做+第 2 个选做功能，具体如下：

(1) 在实验一、二词法分析、语法分析和语义分析均正确后，将 C—代码转化为中间代码，使用三地址代码的形式存储中间代码。

(2) 完成第 2 个选做功能：允许一维数组类型的变量作为函数参数，允许出现高维数组类型的变量。

亮点：

对中间代码进行了优化

(1)Stmt -> IF LP Exp RP Stmt ELSE Stmt

如果第一个 Stmt 最后是 return 语句，就不用生成 go 语句

```
InterCode ic_right = code_tail->code;
if(ic_right->kind != IC_RETURN){
    //优化，判断正确的末尾是不是return语句，如果是，就不用生成GOTO语句
    InterCode ic2 = (InterCode)malloc(sizeof(struct InterCode_));
    ic2->kind = IC_GOTO;
    ic2->u.one_op.op = label3;
    printf("Stmt -> IF LP Exp RP Stmt ELSE Stmt label_id:%d\n",label3->u.varno);
    add_codelist(ic2);
}
```

(2)Exp -> ID | Exp -> INT

直接将对应的变量或常数作为返回值，而不是生成赋值语句

(3)Exp -> Exp = Exp

直接将右值作为返回值，而不是生成 place:=…的赋值语句

二. 程序编译

1. gcc -g -w main.c syntax.tab.c common.c semantic.c intermediate_code.c symbol_tab.c -lfl -ly -o parser
2. ./parser test.cmm

三. 代码实现

1. 数据结构

操作符和中间代码结构如下：

```
struct Operand_{
    enum{
        EM_VAR = 1, // 变量 (var)
        EM_CONSTANT = 2, // 常量 (#1)
        EM_ADDRESS = 3, // 地址 (*var)
        EM_LABEL = 4, // 标签(LABEL label1:)
        EM_ARR = 5, // 数组 (arr[3])
        EM_STRUCT = 6, // 结构体 (struct Point p)
        EM_TEMP = 7, // 临时变量 (t1)
        EM_FUNC = 8,// 函数名 (Func)
        EM_GETADDR = 9// 取地址 (&p)
    } kind;
    union{
        int varno; // 变量定义的序号
        int labelno; // 标签序号
        int val; // 操作数的值
        int tempno; // 临时变量序号 (唯一的)
        char* funcname; // 函数名
    } u;
    char* symbolname;//符号的名字
    Type type; // 计算数组, 结构体占用size
    int para; // 标识函数参数
};
```

```
struct InterCode_{
    enum{
        IC_ASSIGN = 1,
        IC_LABEL = 2,
        IC_PLUS = 3,
        IC_SUB = 4,
        IC_MUL = 5,
        IC_DIV = 6,
        IC_DEC = 7,
        IC_FUNC = 8,
        IC_CALL = 9,
        IC_PARAM = 10,
        IC_ARG = 11,
        IC_GOTO = 12,
        IC_IFGOTO = 13,
        IC_ADDR = 14,
        IC_READ = 15,
        IC_WRITE = 16,
        IC_RETURN = 17,
        IC_GETADDR = 18,
        IC_GETVALUE = 19
    } kind;
    union{
        char* func; // 函数名
        struct{Operand right, left; } assign; // 二维表代码
        struct{Operand result, op1, op2; } three_op; // 三地址代码
        struct{Operand op; int size;} dec; // 数组, 结构体的定义
        struct{Operand op; } one_op; // 单目运算符, 适用于IC_LABEL,IC_FUNC,IC_PARAM,IC_ARG,IC_GOTO,IC_READ,IC_WRITE,IC_RETURN
        struct{Operand op1, op2, label; char* relop;} ifgoto; // 条件跳转
    }u;
};
```

其余数据结构同 ppt

2. 中间代码存储：

本次实验我使用链表存储中间代码，没有像 ppt 把 Intercode 作为函数返回值（因为我觉得传递过程中比较容易出错），每生成一个中间代码就把它添加到存储中间代码的链表上。

```
void add_codelist(InterCode code){
    CodeList c = (CodeList)malloc(sizeof(struct CodeList_));
    c->code = code;
    c->next = NULL;
    if(code_head == NULL){
        code_head = c;
        code_tail = c;
    }
    else{
        code_tail->next = c;
        c->prev = code_tail;
        code_tail = c;
    }
}
```

3. 同一符号的存储：

```
struct FieldList_
{
    char* name; // 域的名字
    Type type; // 域的类型
    FieldList tail; // 下一个域
    int var_no;//序号，用于中间代码生成
};
```

我给符号的域添加了一个元素：var_no，记录中间代码中这个符号的序号。根据我的代码逻辑，对于变量、临时变量和数组，它们的序号都不相同，只是在输出时有差别（例如变量输出 v1,临时变量 t1,数组 a1）。当出现一个符号时，我会先查找该符号有无关联的变量/临时变量/数组，如果没有，就新建，并在域中存储；如果有，使用这个变量/临时变量/数组即可。

4. 之前代码遗留问题：

a. Relop 没有存入具体的符号，修改 lab1 中的内容记录

b. 在语义分析中添加 read 和 write 函数

四． 总结与感悟

1. 本次实验中我认为难度最大的是数组处理，需要好好思考一下，想明白了每步逻辑和处理过程就能写出代码。
2. 依然出现了很多 segmentation fault,我用 print 定位出的 bug 位置不对，最后还是乖乖使用 gdb 了，不能偷懒。

感谢助教哥哥批改！