编译原理lab3 实验报告

221220070 刁涟承

一、功能实现

- 在词法分析和语法分析程序的基础上，实现语义分析和类型检查；
- 无错误时应没有错误输出，有语法错误时准确报告错误类型、错误信息。
- **选做三**：将结构体间的类型等价机制由名等价改为**结构等价（*Structural Equivalence*）**

二、实现思路

操作数类型表示

在设计操作数类型结构体时，用`struct Operand`表示操作数类型，分为`VAR`，`TEMP`，`LABEL`，`FUNC`等多种类型，并用整形或浮点数、字符串标识唯一的操作数值。其中`VAR`，`TEMP`类型可以附加前缀。

```
typedef struct _Operand Operand;
struct _Operand {
    enum {
        _UNDEF,
        _VAR,       // v: 源代码中命名过的变量
        _TEMP,      // t: 源代码中未命名，但是IR中需要临时使用的变量或地址
        _CONST_INT, // 常数，`value`即值
        _CONST_FLO,
        _FUNC,      // 函数: 用函数名表示
        _LABEL      // label: 跳转标签
    } vtype;
    enum {
        _NOTHING,
        _GET_ADDR,  // &v
        _GET_VAL    // *v
    } prefix;
    union {
        int vint;
        float vflo;
        char* vfunc;
    } value;
};
```

IR赋值代码

IR中可能出现的赋值操作用`struct InterCode`表示，分类为`ASSIGN`，`ADD`，`SUB`，`MUL`，`DIV`等等，用于代表中间代码中的所有赋值运算操作。结构体信息可在`lib/nodes.h`中找到。

IR代码节点

生成文件中的每一句中间代码都是一个`IRNode`节点，可以用`struct IRNode`这个结构体表示。分类为`LABEL`，`JMP`，`COND_JMP`，`RET`，`DEC`，`ARG`等等，每个分类都包含1-3个操作数等细分信息；除此之外，为构造线性的中间代码，我使用双向链表节点构造`IRNode`。结构体信息可在`lib/nodes.h`中找到。

翻译EXP

在`transl_Exp()`函数中包含两个参数，除了当前语法树节点`LevelNode* Exp`外，还有一个左值操作数`Operand lvar`，当前表达式若作为语句中赋值号右边部分，需要将其计算结果赋给传入的左值，并将该计算结果作为`Operand`返回给上一级。在`transl_exp_assign()`方法中，负责将左值传入右边表达式；而其它方法中若需要表达式计算，向更深一级表达式中传入临时变量（`TEMP`）。

```
Operand transl_exp_assign(LevelNode* Exp, Operand lvar) {
    Exp = Exp->childhead;
    Operand def = transl_Exp(Exp, newTEMP(-1)); // 用`vint = -1`表示担任左值
    Operand use;
    if (def.prefix == _GET_VAL) {
        use = transl_Exp(Exp->next->next, alloc_TEMP());
        addIRNode(mkIR_ASSIGN(genASSIGN(def, use)));
    }
    else use = transl_Exp(Exp->next->next, def);
    if (!isundef(lvar)) {
        addIRNode(mkIR_ASSIGN(genASSIGN(lvar, def)));
    }
    return use; // 赋值语句: 返回"="右边的
}
```

选做一：结构体变量和结构体参数

在处理`Exp DOT ID`时，我按照以下步骤进行：

1. 计算结构体内部成员`ID`偏移量，取到地址后再取值返回给上一级
2. 对左`Exp`调用`handle_Exp()`得到类型值，调用`getSzieOf()`获得`id`偏移量
3. 注意，`*t`不能直接作为右操作数，只有在复制赋值语句中可以直接用
4. `S[i].id`情况, 特判`IR`代码末尾：若出现对取过值的临时变量取地址，则删除前两行，直接对原地址操作

三、编译方式

可以使用`Code/Makefile`结合命令行进行编译：

```
make clean: # 清除所有生成文件
make t-%: # 用某个测试样例测试
make io-%: # 用某个样例测试并输出到out.txt
make c-%: #用某个样例测试并输出到out.txt，与预期输出文件比较
```

四、编译环境

操作系统：GNU Linux Release: Ubuntu 22.04 LTS, Kernel version 5.15.153.1-2；
编译器：GCC version 7.5.0；
词法分析工具：GNU Flex version 2.6.4；
语法分析工具：GNU Bison version 3.0.4。