实验三 中间代码生成 实验报告

匡亚明学院 陈劭源 161240004

文件夹结构

注意：**Makefile**在根目录下（而不是在**Code**文件夹内）

```
.
├── Code                    // 源代码文件
│   ├── ast                 // 抽象语法树相关代码
│   │   └── ...
│   ├── error.c             // 错误处理代码
│   ├── lexical.l           // flex词法文件
│   ├── memory.c            // 内存管理代码
│   ├── symtbl.c            // 符号表代码
│   ├── main.c              // 主程序
│   └── syntax.y            // bison语法文件
├── Include                 // 头文件
│   ├── ast                 // 抽象语法树相关
│   │   └── ...
│   ├── cmm.h
│   ├── memory.h
│   ├── symtbl.h
│   ├── container           // 包含链表等数据结构
│   │   └── ...
│   ├── cst.h
│   ├── ir.h                // 中间代码相关
│   ├── location.h
│   ├── option.h
│   ├── utility.h
│   └── error.h
├── Makefile                // Makefile文件
├── parser                  // 语法分析器可执行文件
├── README.txt              // README文件
├── report.md               // 本实验报告的源代码
├── report.pdf              // 本实验报告
├── Test
│   ├── sample              // 提供的测试用例
│   │   └── ...
│   └── secret              // 自行构造的测试用例
│       └── ...
└── testrun.sh              // 测试用例运行脚本
```

编译和运行方法

编译环境

- OS: Ubuntu 18.04.2 LTS
- gcc: gcc 7.3.0
- flex: flex 2.6.4
- bison: GNU Bison 3.0.4
- make: GNU Make 4.1
- shell: GNU bash 4.4.19

编译方法

切换到根目录，输入

```
make
```

即可从源代码生成可执行文件parser（位于根目录）。

运行方法

输入

```
./parser
```

或

```
make run
```

即可运行语法分析器。语法分析器默认从标准输入读入c--源代码，可以通过参数指定从文件读入：

```
./parser source_file
```

运行

```
make clean
```

可以清除所有中间文件和目标文件。

完成的功能点

1. 将没有语义错误的c--源代码翻译成中间代码；
2. （选做）允许定义结构体类型的变量，并且可以将结构体类型作为函数参数，但是
    - 结构体不允许作为函数的返回值类型，也不允许结构体之间互相赋值；
    - 结构体的等价方式采用名等价；
    - 结构体作为函数参数时，遵循按值传递规则（即函数内修改结构体不会影响调用者中结构体的值）。
3. （选做）允许定义任意维数组，并且数组可以作为函数参数，但是
    - 数组不能作为函数返回值类型；
    - 任何情况下均不允许数组之间互相赋值（即使它们的维数和每维大小都相同）；
    - 数组作为参数传递时，必须确保数组的维数和每维的大小都匹配；

- 数组作为函数参数时，遵循按值传递规则（即函数内修改数组不会影响调用者中数组的值）。这一点与**C**和**C++**的规定并不一致，请特别注意。

实现方法

本次实验在上次实验构建的抽象语法树上进行。具体来说，根据抽象语法树节点的类型，生成不同的中间代码语句。

对于表达式而言，中间代码由下表生成：

Type	**IR Code**
INT	**tmp** := #INT
a *binary_op* b	**tmp** := a *binary_op* b
unary_op a	**tmp** := *unary_op* a
a = b	**a** := b
a *relop* b	**tmp** := 1 IF a *relop* b GOTO l1 **tmp** := 0 l1:
a && b	**tmp** := 0 IF a == 0 GOTO l1 IF b == 0 GOTO l1 **tmp** := 1 l1:
a || b	同上，但0,1互换
! a	**tmp** := 0 IF a == 0 GOTO l1 **tmp** := 1 l1:

对于语句，中间代码生成方式如下：

Type	IR Code
RETURN a;	(compute a) RETURN a
if (a) s;	(compute a) IF a == 0 GOTO l1 s l1:
if (a) s; else t;	(compute a) IF a == 0 GOTO l1 s GOTO l2 l1: t l2:
while (a) s;	l0: (compute a) if a == 0 GOTO l1 s GOTO l0
func(a1, a2, ...)	... (compute a2) ARG a2 (compute a1) ARG a1 CALL func

实验总结

本次实验在上一次实验的基础上进行，通过遍历抽象语法树并按照翻译模式进行翻译，将代码翻译成中间表示。