编译原理实验 3 实验报告

161220143 吴御洲

一、概述

在实验 1 对 c—代码进行词法分析和语法分析，实验 2 对 c—代码进行了语义分析的基础上，生成了中间代码，以线性结构输出，同时在虚拟机中进行测试。

此外选作部分实现了选作 3.2，可以处理高维数组，同时可以将一维数组作为函数参数。

二、文件结构

lexical.l syntax.y syntaxtree.h syntaxtree.c 文件功能与之前实验一致；

semantic.h semantic.c 文件中在实验 2 的基础上，在对应的产生式处理函数中增加了翻译功能，同时增加了一个 Cond 函数，处理条件语句。

IR.h IR.c 文件中实现了中间代码的管理和新的数据结构的定义。

三、具体实现

```
typedef struct Operand_* Operand;
typedef struct Operand  {

typedef struct InterCode_* InterCode;
typedef struct InterCode  {
```

这是 IR.h 中定义的数据结构，同时利用一头一尾两个指针来动态存储中间代码

Operand 中包括了临时变量、变量、常数、地址、标号、函数这几种类型，根据类型不同存储的可能是一个记录第几个的整形数、一个字符串或者一个 Operand

InterCode 中也有不少类型，然后可以具体分为单操作数的 unaryOP，双操作数的 binaryOP，三操作数的 ternaryOP 和特殊的 ifgotoOP、decOP

在翻译函数调用的时候，会去 Args 函数中处理，注意在获取下一个的时候，会把后面的插到最前面，这样在传入参数的时候顺序即可正常。

在翻译数组的时候，首先获取基地址，然后读取第一个下标。如果是非零的数，就需要计算后面的大小，如 a[i][j][k]，若 i 不为 0，就需要计算[j][k]的大小，然后乘上 i 加到基地址上。此外如果下一层的数组存储类型是 int，则回填的是地址指向的位置，反之还是地址。

获取大小的 getSize 函数，如果传入类型是 BASIC，则返回 4，反之如果是 ARRAY，则取出数组的大小和类型，返回大小和 getSize(类型)的乘积。

此外，在.y 文件对涉及到 STRUCT 的处理的地方添加了一个 errorNum++，从而在出现结构体的时候可以顺利报错

四、如何运行

所有文件放在/Code 下，在/Code 输入 make parse 后生成可执行文件，之后将 xxx.cmm 测试文件也放在/Code 下，输入./parse xxx.cmm xxx.ir，即可将中间代码线性打印到 xxx.ir 文件中。然后在虚拟机小程序中处理，查看结果。

五、优化

在 Exp 的处理中，如果发现其是一个 INT 类型的，则不会先产生一个临时变量获取值然后再赋值，二是直接处理为一个 CONSTANT_OP 类型的 Operand，可以减少一条中间代码。

六、效果

设计了测试赋值、加减乘除运算、选择条件语句的代码，没有错误。
设计了循环的测试代码，没有错误。
设计了简单的函数调用 add，没有错误。
用排序测试了数组的处理，没有错误。
设计了二维数组的运算，没有错误。