编译原理实验三实验报告

191220163 计算机科学与技术系 张木子苗

完成的实验要求

在本次实验中，我完成了**全部的必做任务**，代码**可以通过所有的必做内容样例。**没有完成选做内容。

提供的测试方法

在Test目录下，我一共准备了4个代码文件用于测试：

`test1.cmm` 到 `test4.cmm` ，对应实验指导文件里的必做样例1-2和选做样例1-2

为了方便测试，我为我的代码写了个简单的脚本文件，`Code/Test.sh`，在**Code目录下命令行界面输入**：

```
./Test.sh n
```

即可自动编译并且执行对源文件 `testn.cmm` 的中间代码生成，并且将结果写入到文件 `temp.ir` 中。注意，**文件 `temp.ir` 在 `irisim` 目录下!**

如果需要测试其它用例，**请将新的测试用例放在Test目录下，命名格式为 testn.cmm（n不要为1-4）。在Code目录下命令行界面输入**：

```
./Test.sh n
```

即可自动编译并且执行对该测试用例的中间代码生成，并且输出到文件中。

```
njucs@njucs-VirtualBox:~/Compile_Code/Lab3$ cd Code
njucs@njucs-VirtualBox:~/Compile_Code/Lab3/Code$ ./Test.sh 1
njucs@njucs-VirtualBox:~/Compile_Code/Lab3/Code$ ./Test.sh 2
njucs@njucs-VirtualBox:~/Compile_Code/Lab3/Code$ ./Test.sh 3
Cannot translate: Code contains variables or parameters of structure type.
njucs@njucs-VirtualBox:~/Compile_Code/Lab3/Code$ ./Test.sh 4
Cannot translate: Code contains variables or parameters of multi-dimensional array type.
njucs@njucs-VirtualBox:~/Compile_Code/Lab3/Code$
```

代码的实现思路

数据结构

构造两个结构体：`struct Operand_` 和 `struct InterCode`

其中 `Operand` 代表的是中间代码中的操作数，`InterCode` 代表的是一行中间代码。`InterCode` 中的 kind 表示中间代码的类型，即 Label，Assign等等。由于指令类型不同，对应的操作数数量也不同，再利用联合体 union u，根据中间代码的 kind 来决定操作数。

另一个结构体 `struct Arg_List_` 用于处理函数调用时传参。

具体的数据结构实现参见 `intermediate.h`。

实现思路

对于符号表的初始化，我们在 `semantic.c` 中加上一个 `init_table()` 函数，并且在main函数中调用。

翻译中间代码时，我们依旧时通过遍历语法分析树，生成中间代码。

具体思路是：对于不同的非终结符，分别编写其对应的 Translate 函数，在碰到包含该非终结符的产生式时调用。

Translate 函数的编写大致参考了实验指导，此处不再赘述。

所有的中间代码被放在了一个类型为 `InterCode` 的大数组中，采用的是固定分配，上限是10万行。当中间代码生成步骤结束之后，根据数组中的内容，将结果打印到文件 `temp.ir` 中。

对于#ifdef的说明

在做实验过程中碰到了不少bug，为了定位bug所在的地方，我在进入每个translate函数的最开始处添加了一句输出语句，打印该函数的函数名。当在 `intermediate.h` 的最前面 #define debug 时，会输出调试信息。在提交前我已经讲这句 #define 注释掉了，可以忽略。