编译原理第三次实验报告

201220069 周心同

实验内容

实现内容：包含所有的必做要求和选做要求，并通过了OJ所有样例。

本次实验是根据实验一的语法树并借助实验二已构造出的语法表生成中间代码，并通过一个虚拟机小程序检验生成的中间代码的正确性。

该实验难度很大，尤其是结构体和数组的翻译模式，同时debug非常困难。

编译方式

实验环境：与实验要求相同。

在Code目录下make即可。

程序亮点

1.为了较好的模块性，也防止改动对实验二造成的影响，本次实验内容全放于其他单独的文件中。

2.代码对照指导书中表的翻译模式设计了相对应的体系，能够与表中内容完美对照起来。

对于中间代码的结构，采用线性的动态数组，相比于链表，动态数组能够较简单的实现线性结构，相比于静态数组，动态数组能有效利用空间，实际上在实验过程中，没有一定要对中间代码进行插入、删除以及调换位置的操作，全部操作均在尾部追加中间代码，所以用单向链表其实就可以了。

举例：

Exp_1 RELOP Exp_2	t1 = new_temp() t2 = new_temp() code1 = translate_Exp(Exp_1, sym_table, t1) code2 = translate_Exp(Exp_2, sym_table, t2) op = get_relop(RELOP); code3 = [IF t1 op t2 GOTO label_true] return code1 + code2 + code3 + [GOTO label_false]

```
Operand t1 = new_temp();
Operand t2 = new_temp();
tExp(node->child[0],t1);//code1
tExp(node->child[2],t2);//code2
char * op = node->child[1]->yytext;
InterCode code3 = (InterCode)malloc(sizeof(InterCode_));
code3->kind = IF_GOTO_IR;
code3->operands[0] = t1;
code3->operands[1] = t2;
code3->operands[2] = label_true;
strcpy(code3->relop,op);
insertCode(code3);//code3

InterCode code4 = (InterCode)malloc(sizeof(InterCode_));
code4->kind = GOTO_IR;
code4->operands[0] = label_false;
insertCode(code4);
```

由于时间原因，加之InterCode种类繁多但每种数量不多，没有将IR的生成包装成函数，只是对OP进行了一些包装处理。

实验建议和感想

实验难度很大，建议多给一点框架，而不是从零起步，或者举一句翻译中间代码的例子，或者给出数组和结构体其中一个的翻译模式，这两个卡了我估计有10h，而且本实验debug手段非常有限，同时又很容易出现段错误。

还有做这个实验的一些诀窍：

1.首先不要有过于繁复的架构，我的架构虽然比较规整能和表对应，但有时也比较麻烦，当然这也有c不能用string进行拼接的原因。

2.如果碰到段错误，那么九成九就是空指针，而且非常容易出现，建议在每一次用指针（尤其是place这个不一定赋值的）的时候前面就加一个NULL的判断，并定位好bug位置。debug时可以结合语法树，gdb和二分注释法。

3.如果不考虑不同作用域下的同名变量，那么无需按照语法树的顺序去翻译，因为实验而给出了符号表，所以只需要对语法树按照实验一的遍历顺序一句句翻译即可

4.遇到困难一定要多打tag，并从一些其他的地方加入tag的信息，而不是通过复杂的方法求得，比如我从实验二中加入对变量是不是参数的判断，还有在操作数op中加入该操作数的type的值而不是想办法去求得该操作数的type。

5.临时变量和变量从正确性上看完全可以看成一类，没必要分开来给自己加了很多麻烦。