**编译原理实验 lab3**

1. 你的程序实现了哪些功能？简要说明如何实现这些功能

本次实验最主要的功能是实现语法树到中间代码的转化。

中间代码的转换过程，大体基于lab2语义分析的遍历语法树规范，在省去类型判断的优势下，同样需要进行符号表的建立与使用。中间代码的生成与语义分析的最大区别，一方面是语法树的类型规范性已得到保证，从而只需要根据中间代码规则生成对映代码即可；另一方面，则是在仅实现INT类型的前提下，对类型判断的结果进行进一步展开，生成对映的中间代码。

在实现中间代码生成时，我新创立的了translate.cc文件，大体继承symbol\_test.cc文件的框架，对语法树进行二次遍历；对于语法树中节点，不再进行测试类型比对操作，而是根据节点类型，如EXP,IDENT,FUNC\_DEF等等，生成相应的中间代码，并直接将中间代码结果记录于返回。

在新一轮的遍历过程中，全局变量的定义需要区别于一般变量的定义，因为全局变量的实质成为了一个LABEL而不再是一个简单的IDENT，从而我建立的新的函数global\_def()来处理此类情况；另外，对于函数的使用，特别是返回值为INT的函数以及参数的传递使用，也需要进行更为细致的处理，具体实现与分析将在第三部分详细展开。

2.你的程序应该如何被编译？请详细说明应该如何编译你的程序。无法顺利编译将导致助教无法对你的程序所实现的功能进行任何测试，从而丢失相应的分数

解压相应文件后，在最外层文件夹使用 “make compiler”命令即可获得compiler二进制文件，区别于lab1、lab2的compiler文件，其不仅接受一个文件的输入，同时输出一个具有中间代码的文件。

若使用“python3 test.py ./compiler lab3”测试命令，只可判断compiler的通过性正确率；使用“./compiler xxx.sy xxx.txt”测试命令，则可以获得词法分析器（每一个token）的输出结果；语法分析树（有缩进结构）的打印结果；语义分析阶段的正确与否，分别显示“Successfully！”与“Failed!”及相应错误原因以及经编译SysY语言后产生的IR中间代码文件。

1. 所以实验报告中需要重点描述的是你的程序中的亮点，是你认为最个性化、最具独创性的内容，尤其要避免大段地向报告里贴代码

3.1 词法分析的参数遗留问题

在中间代码的生成过程中，我遇到了词法分析时遗留的函数参数问题。在词法分析时，我们仅仅判断使用函数时，函数的参数是否符合词法，并直接记录其类型。这导致了数组a与INT变量a在词法分析器看来均为INT IDENT结构，没有INT变量与数组之分，导致在嵌套使用exp\_ir函数时，实参会因为此类情况进入错误的函数条件分支，对中间代码的生成带来一定的干扰。通过减弱了对映函数条件分支内部的判断条件，并将数组与全局变量归为一类（本质都是LABEL的传递），便可以较优的解决改问题。

3.2 函数返回值使用

在中间代码生成的过程中，函数的返回值，特别是返回值为INT的函数，给程序带来了极大的干扰。首先是STMT模块中的函数调用，在这一部分，函数的调用既可以使用返回值（赋值操作），又可以不使用返回值（仅仅调用函数产生副作用），但因为传递过程明确，可以通过建立函数传递参数（是否为首次调用）的方法进行解决。

但在return，if及while等模块中，exp\_ir存在较为深层的多次嵌套调用情况，函数（是否为首次调用）参数不再具有调控函数是否需要为返回值建立临时变量的通用性，因而我引入了return\_fun这一新的全局变量。在上述模块的进行中，若检测到即将进行基于函数调用的exp\_ir函数使用时，主动将return\_func全局变量置1，从而肯定函数返回值的需要性。

3.3 地址与数值传递

在lab2中，虽然我们涉及了类型判断，但因为只实现了INT类型（数组也由INT类型构成），因而在表达式的两侧，只要符号使用合法，我们便可以默认两侧已经类型一致（对于INT而言，数组类型需要进行额外的判断，往往引导判断的完全性）。但在生成中间代码，以及使用解释器进行计算，需要得到确切的答案时，我们需要进行更为细致的解读，特别是对于数组与全局变量的使用。

我们假设a是一个一维数组，我们可以在运算中，直接给出a，也可以给出a[1]，这在类型判断时，不需要进行额外的观察，因为a这一地址也是一个INT类型，但在生成中间代码时，为了程序的正确性，这不得不进入我们的考虑范畴。从而我们要在语义分析的基础上，对语法树进行更深一步的解读。另外，数组的赋值更是要基于地址的偏移进行填值，在使用中，也需要根据a[1]与a[i]数值还是变量的索引不同，决定是否需要生成更多的临时变量。

3.4 if、while的实现

在词法分析，语法分析甚至是语义分析时，条件判断跳转一直是较为简单的部分，类型检测往往只需要递归调用加上调用test\_exp函数即可。但在中间代码的生成部分，我们需要更为整体的考虑LABEL及其跳转的设置以及对OR/AND等条件的一步步跳转检验。

还是沿用递归的方法，为了不改变已有的函数接口，我在此基础之上，又引入了true\_design和false\_design两项全局变量，分别对映于判断条件为true或false时程序的LABEL设置及跳转方向。对于if-else语句而言，我只需要在递归的基础上，在true跳转模块的结尾加上一条无条件跳转语句，并在else模块后增添一项新的LABEL即可。同理，对于while循环，只需增加一项跳转至循环体头部的GOTO语句即可。