**编译原理实验 lab1**

1.你的程序实现了哪些功能？简要说明如何实现这些功能

在Lab1中我主要实现了词法分析器、语法分析器以及语法分析树的建立。

其中，我先编写了lex代码，用正则表达式定义了各类token的字符组成；其中对于跨行注释的处理，同样没有借助lex状态功能实现；再借助flex工具，自动生成了解析判断这些token的C语言编写的词法分析器。

接着，我使用了上下文无关语法，用yacc的代码形式定义了简化版SysY语言的语法，重定义了一些常用符号的token表达，并引入语法分析树的node结构，最后借助Yacc工具，自动生成了由C语言编写的语法分析器。在本次编写与实现上下文无关语言语法时，我尝试了不使用空字符转换的语法表示。

在保证词法分析器与语法分析器正确的基础上，我建立了SysY语言的语法分析树，使对映程序的词法/语法解析可以得到更加直观的结果。语法分析器的实现全由我自己编写，因为主体使用了C语言（而非C++），从而可能尚存一定的重复代码。在语法分析树的建立过程中，我并没有借助union与enum的联合表达形式，而是采用统一的node形式记录不同类型的节点。在每一节点中，我记录了对映token的type，但并不是每一种token都对应一个节点，如“[”“]”就没有相应的节点对映（只出现于array的表达之中）；enum变量记录type的kind类型，主要为string（对映于变量名的记录），int（对映了各种进制整数与int const）以及noe（对映于不需要记录语义值的提示性token，如stmt、block等）；在此之外，我还基于kind类型分别记录name/value值，便于语法分析树的输出；最后记录了行号信息，便于后续的使用。

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。

若使用“python3 test.py ./compiler lab1”测试命令，只可判断compiler的通过性正确率；使用“./compiler xxx.sy”测试命令，则可以获得词法分析器（每一个token）的输出结果以及语法分析树（有缩进结构）的打印结果。

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
   1. 使用正则语言表示跨行注释

在本次词法分析器的实现中，我并没有使用状态的方法，来解决词法分析器对跨行注释的判断，而是使用了巧妙的正则表达式。如下式：

"/\*"[^\*]\*"\*"+([^\*/][^\*]\*"\*"+)\*"/"

对于跨行注释，首先读入左侧的/\*，表示跨行注释的开始；接着读入若干个非\*字符，即注释的内容，直到遇到下一个\*；接着读入至少一个连续的\*号，因为我必须抓捕\*/的存在，所以需要保证始终记录到最后一次出现\*的位次是不是上一个字符；若\*后紧跟/，则表示跨行注释结束，跳过([^\*/][^\*]\*"\*"+)\*；若不是/，则我必然可以读入至少一个非\*非/的符号；在此之后，其实已等价于我刚读入/\*的状态，因而此时只需记录最后一次出现\*的记录是不是上一个字符的状态即可。

* 1. 上下文无关语法中没有使用空字符串的转换

因为在SysY的定义语法中，存在大量的[]（出现0次或1次）{}（出现0次或至少一次）符号，但在Yacc的语法中，[]不被接受，{}则被接受为当前归约发生后动作；就一般操作而言，这样会产生大量形如 A：ε的语法规则，在实际的语法检测中，同样会产生一定的影响。因而我将部分语法规则进行拼合或枚举（如{}与{block\_item}），进而消除了转化为空字符串的所有干扰。

* 1. 对归约/归约冲突的解决

若按一般情况，仿造SysY定义语法编写简化SysY语法检测器Yacc代码时，会遇到一个reduce/reduce conflict；这是因为在定义中变量类型定义和函数类型定义的转化规则，均有可能以int IDENT为开头，这就导致了在一般的情况下，Yacc生成的语法分析状态机在已经接受到int，并下一个接受任意IDENT的时候，不知道向变量定义状态还是函数定义状态进行转变，从而产生两个reduce之间的conflict。在实现的细节中，我将int 与IDENT两个状态合并为一体，即不再接受IDENT时决定定义类型的走向，而是通过IDENT后的下一个字符，来决定归约方向：若接收到“；”/“=”，归约则向变量定义方向转变；如果接收到“(”，归约则向函数定义方向转变。

3.4语法分析树采用了左儿子右兄弟的策略

在生成语法分析树时，我采用了左儿子右兄弟的方式；且在打印语法分析树时，仅对左儿子添加缩进，右兄弟则为同样缩进的照常输出。这样的设定可以使我语法分析树的每一个节点都是二叉的，使不同类型的节点间更具一致性；另外，虽然在具体实现中这样的设定会增加代码的实现复杂度，但在语法树输出时，并列的变量定义，赋值语句的表现等会有同样的缩进，虽然树的深度没有实际减少，但在实际输出的可视化判断时，这会拥有更加良好的层次化表示；也可能有利于后续的语义分析。