**编译PA1：词法分析 报告**

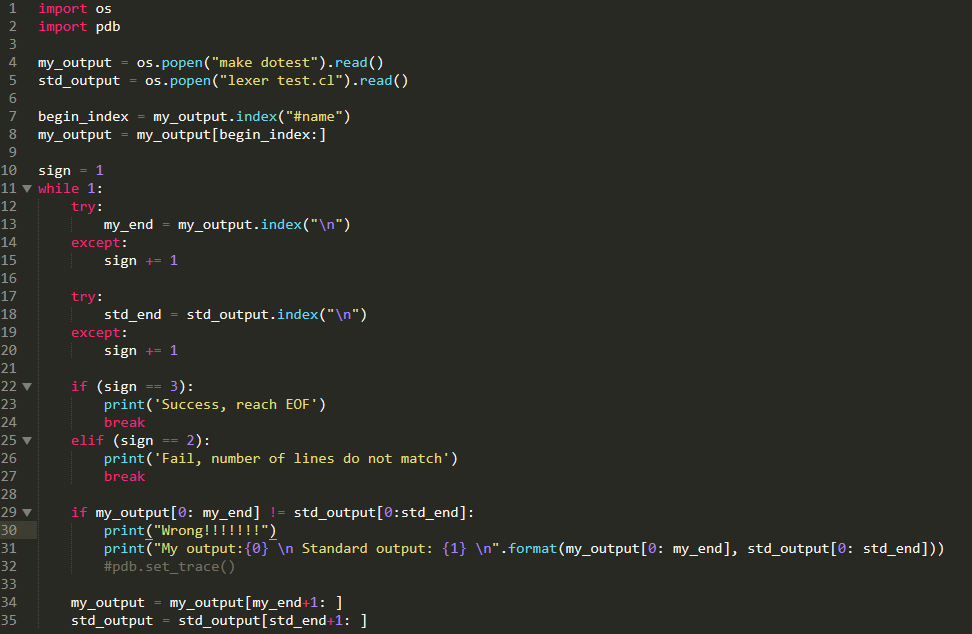
17307130015 王鑫涛

**一、背景知识**

本次实验为PA1，但实际上对应的文件夹为assignments/PA2，主要修改其下的cool.flex。flex是词法解析器的生成器。我们可以通过运行make lexer在当前目录下产生可执行文件lexer。运行这个lexer并传给他一个文件路径(e.g. lexer test.cl)，就可以让他对test.cl文件进行词法分析，并得到输出。

项目提供了测试用例test.cl以及作为参考物的完整的lexer（位于bin目录下）。通过比较我们自己写的lexer的输出和完整lexer的输出，即可知道我们是否正确实现了生成器。

参考《CS143：编译原理｜PA2：正则表达式和词法分析》一文，在实验开始前，先实现了一个用于比对我们写的lexer的输出和完整lexer输出的python程序。



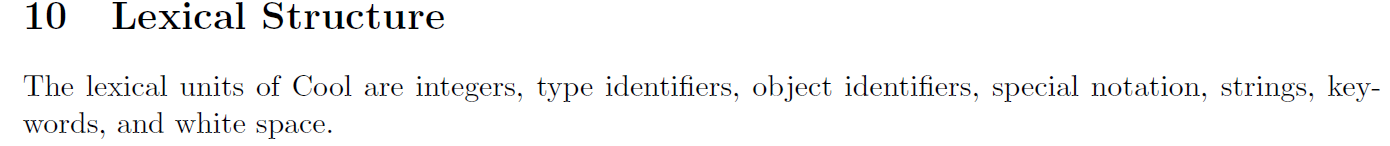
通过执行make dotest，可以编译lexer（这一步也可以通过make lexer实现），并使用lexer对test.cl进行词法分析。

cool\_flex的代码结构在pa1\_intro.pdf里解释得比较详细。在Definition部分，有两种代码：C代码和flex代码。要写C代码，可以使用缩写或用%{、%}括号包起来。%{和%}之间包含的内容会被原封不动拷贝到输出文件C代码中。

值得注意的是，在Rules的定义中，如果没有声明<condition>，则默认为仅在<INITIAL>状态下生效，而不是对所有状态均生效。

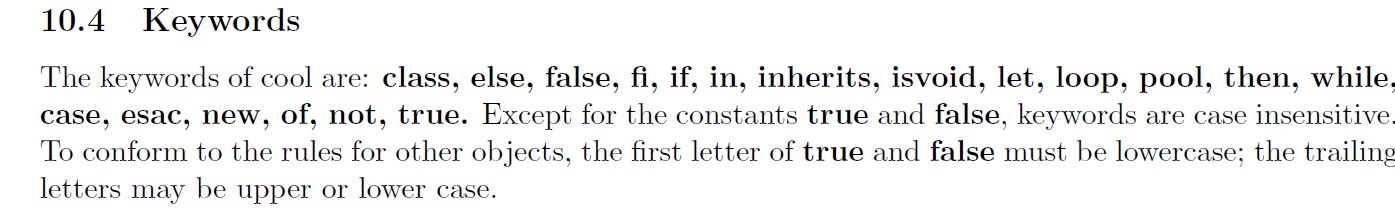
Definition和Rules部分的其他细节可见pa1\_intro.pdf。

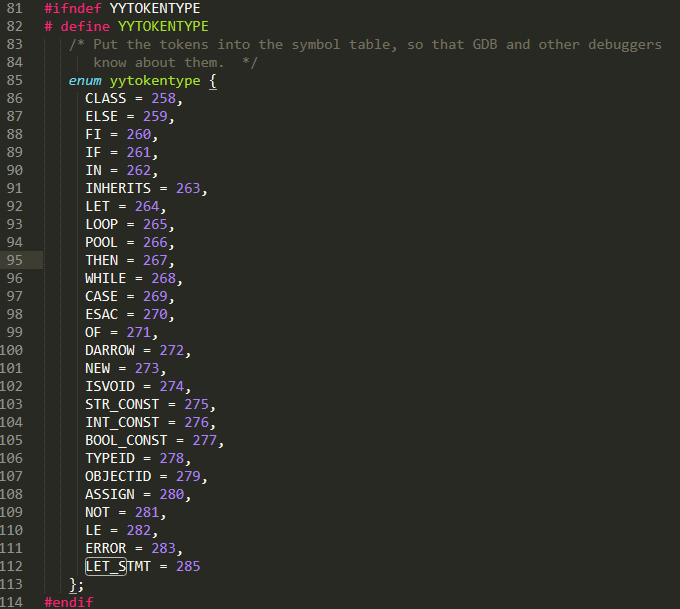
所需要实现的许多规则在cool-manual.pdf中的10 Lexical Structure中。主要是整数、标识符、字符串、注释、关键字和空白符。



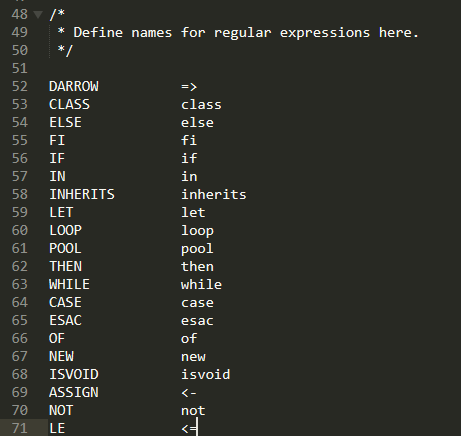
**二、关键字**

Cool语言中的关键字是固定的，定义在include/PA2/cool-parse.h中。STR\_CONST、INT\_CONST、BOOL\_CONST、TYPEID、OBJECTID并不是关键字。LET\_STMT暂时忽略。 这在cool-manual.pdf中的10.4也可以看出。

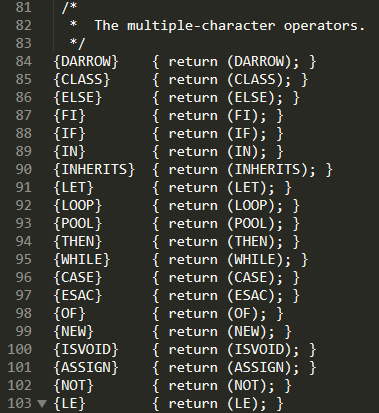




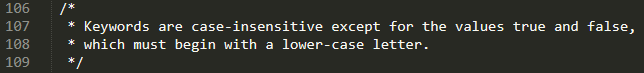
在Definition区，定义这些关键字。



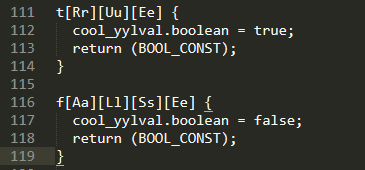
在Rules区，定义相应的行为。



根据代码中的提示，关键字中，true和false两个常量关键字要求首字母必须小写。



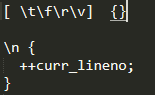
这2个常量的信息将进入全局变量cool\_yylval。同时，需要返回BOOL\_CONST类型。



**三、单字符**

使用.可以匹配到合法字符。直接返回字符yytext[0]即可。需要注意的是，如果将.定义在代码中较前的位置，由于.任意匹配的能力，会覆盖所有单字符的匹配。因此，最好将任意字符的匹配放到最后。

然后处理空白字符和换行符。空白字符（包括空格！）不需要任何的操作，而换行符需要增加当前行数。



**四、注释**

接下来实现注释。注释可分为单行注释和多行注释。

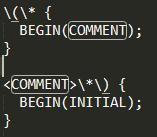


单行注释以两个-开头，之后匹配任意字符都会被认为是注释。单行注释只需屏蔽掉这一行，即不产生任何效果。

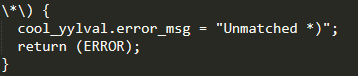
在设计多行注释中，由于(\*标识了注释的开始，\*)标识了注释的结束，这里需要用到状态变量，具体可见pa1\_intro.pdf内的start condition部分。



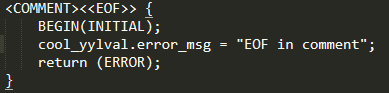
首先，在Definition区，声明状态变量COMMENT。



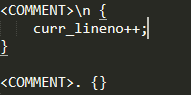
理想的情况下，我们遇到(\*，就激活状态COMMENT。（在COMMENT状态下）遇到\*)，就表示注释的结束，返回初始状态。



如果我们遇到了\*)，但此时不在COMMENT状态下，即没有出现过(\*，那么这时显然是错误的。报错信息参考lexer的输出。



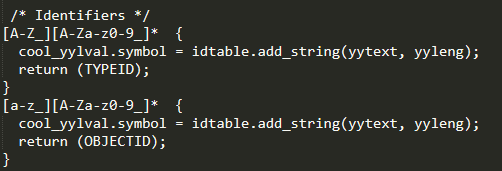
同样，如果出现了(\*，但只到文件结束都没有\*)，也是不合理的，也需要报错。



遇到其他字符，我们都不生成token。但是如果遇到换行符的时候，还是需要增加行数的。

**五、标识符**

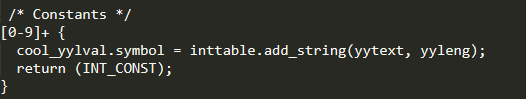
在cool中，变量名的定义同c，而类名则要求以大写字母开头。当我们识别到标识符的时候，要将其相关信息写入符号表中。



对于以大写开头的类别，需要返回TYPEID；对于以小写开头的变量，需要返回OBJECTID。符号表有方法add\_string，我们可以用这个方法将匹配到的内容yytext保存到其中，然后将其写在全局变量cool\_yylval的symbol当中。

**六、常量**

整数常量可以直接使用数字的正则表达式进行匹配。常量也需要记录在符号表当中。区别在于，需要返回INT\_CONST。



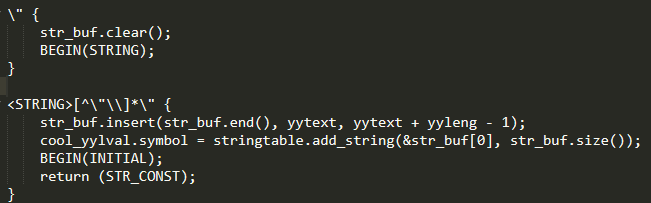
对于字符串常量来说，字符串常量是用双引号定义的。这里类似于上述的注释定义，我们需要使用一个条件变量，来标识字符串定义的开始 -- 也就是说出现了第一个双引号。而当出现第二个双引号时，则标识着字符串的结束。另一方面，在字符串定义中，我们还需要处理转义符号，因此需要使用一个变量来标识当前是否在转义状态下。因此，定义两个新的条件变量。

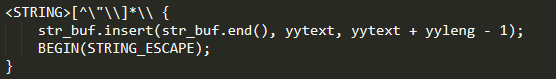


定义一个变量来存储字符串。考虑到如果使用字符数组来存储，将会有长度的问题，因此索性使用c++中的string。为了与事先定义好的字符数组string\_buf区别开，就叫它str\_buf。

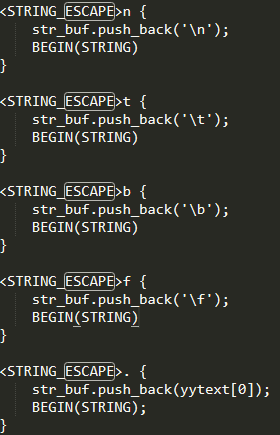




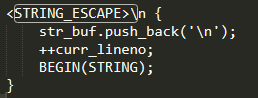
正常情况下，我们以第一个引号的出现，标识识别字符串变量的开始，激活状态STRING，并清空缓冲区。在STRING状态下，我们识别除了转义字符和引号以外的字符，直到出现另一个引号。我们忽略最后一个引号，将识别到的其他内容插入到str\_buf中，然后在符号表中记录这个变量，返回初始状态，即可。

遇到转义符时的处理大致是一样的，都忽略到最后识别到的字符（转义符），将其他的字符先插入其中。遇到转义符时，进入转义状态STRING\_ESCAPE。

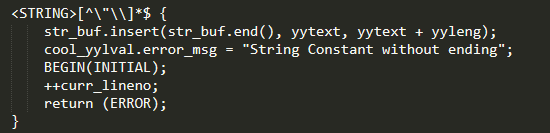
只需要处理\n \t \b \f这几个转义即可。其他情况的转义符则被忽视。



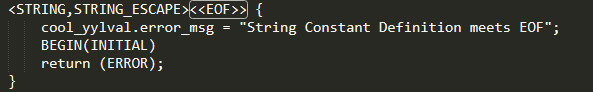
如果遇到[\\n](file:///\\n)，应作为换行符处理。



如果字符串常量定义的结尾没有另一个引号或者转义符，是错误的情况。

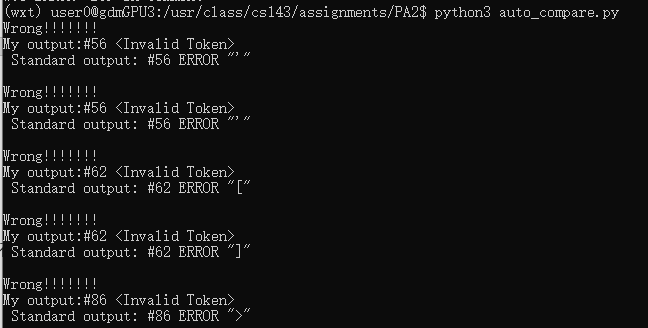


同理，我们也不期望在定义字符串常量时遇到EOF。

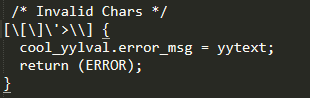


**七、非法字符**

在定义完以上部分之后，进行python auto\_compare.py时，我发现test.cl中还存在非法字符。

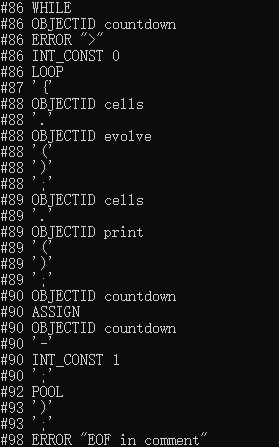


我们需要识别出这些非法字符，并返回ERROR。



**八、最终测试结果**

通过使用make dotest，可以查看自己生成的词法分析器对test.cl的分析结果。



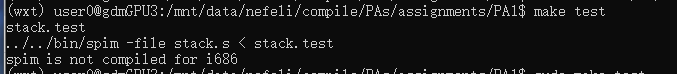
通过使用上述定义的python文件对我们写的cool\_flex生成的词法分析器和参考词法分析器对test.cl分析结果进行比较，发现没有报错，两者生成的结果是一致的。



**九、心得体会**

本PA作为编译课的第一个PA，难度本身还好，不过配环境就花了我比较多的时间。由于对linux系统不够熟练，遇到了以下问题，后来发现才是64位系统对32位文件的不支持。而在处理这个问题的过程中，由于环境下的apt-get出了一些问题，多次换源失败，使得我花了很多时间，大感受挫。





一开始接触cool语言时，也感到了比较大的困惑。通过百度和cool-manual.pdf，才了解cool语言的语法。然而，关于cool和生成器的很多细节，还是在实验中，不断比对正确输出和生成的输出，才能发现的，比如不声明condition不代表匹配任何condition，比如cool中的非法字符。