**CMM解释器说明文档**

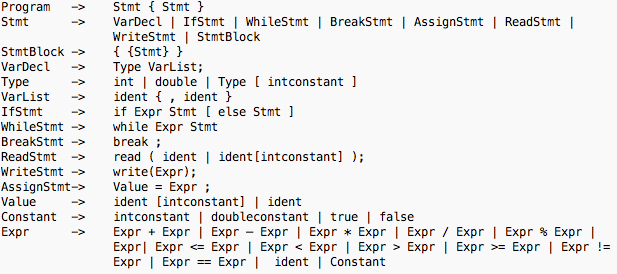
1. **编译方法说明**
2. **词法语法说明**

词法现象采用了附录1中的词法结构，如下

|  |  |
| --- | --- |
| 保留字 | If, else, while, read, write, int, double, true, false, break |
| 特殊符号 | + - \* / = , ; < <= > >= == != ( ) { } [ ] // /\* \*/ |
| 其他 | 十进制与十六进制（使用0x或者0X表示）的整数与实数（其中：**.11** 为非法, 但0.123 或者12. 或 12.0 是合法的）  标识符（由数字、字母和下划线组成的串，但必须以字母开头、且不能以下划线结尾的串；大小写敏感） |

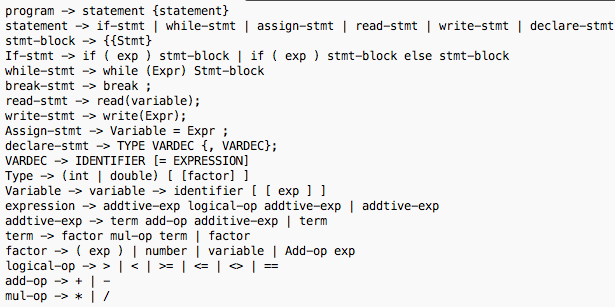
语言结构也如附录1所示。

语法采用了附录1中的CMM文法，在该基础上做了部分修改，修改后的文法规则如下：



1. **语法分析**

采用递归下降子程序法，先去除左递归，然后提取左因子，得到BNF语法如下：



为每一个文法规则申明一个函数，来判断输入的token序列是否符合文法规则，并且返回一个树节点对象。

然后使用词法分析器完成词法分析，将解析完成的token序列输入到语法分析器中，每遇到一个非终结符，则调用相应的分析子程序，然后在子程序中对符号一一进行匹配，如果匹配失败，进行错误处理，如果匹配成功，返回到上一个程序，进行下一非终结符的匹配，直到所有的非终结符匹配完成，将每一个匹配完成的终结符或非终结符都添加到一个树节点中，每个树节点都定义了其子节点的集合以及该节点的类型和值，通过递归调用，形成树节点的关联，构成一个语法树，即语法分析最后的结果

1. **语义分析**

语义分析中，对语法分析生成的语法树进行遍历操作，判断语法上下文关系，检查是否有语义错误，如类型检查等等。类似于递归下降法，为每种属性文法生成一个函数，然后为每一步操作编写一个四元式，包括临时变量的生成，值的计算，跳转等操作。

1. **中间代码**

自己定义了中间代码的形式为：(operation, arg1, arg2, address)，第一位为操作符，最后一位为地址（变量名），定义了如下几种类型的中间代码表达式

*(jmp, arg1, \_, index )*表明如果arg1为false需要进行跳转，跳转到下标为index的中间代码上

*(read,\_,\_,variable)*表明需要从控制台读取变量的值，并赋给variable

*(write,arg1,\_,variable)*表示需要输出，如果variable为一个数组，arg1就为它的下标，否则，arg1为\_

*(in,\_,\_,\_)*表示进入到一个新的符号层（之后在代码实现中解释）

*(out,\_,\_,\_)*表示退出当前符号层

*(assign, value, arg1, variable)*表示赋值操作，将value赋值给variable，如果涉及到数组下标的赋值，需要用的arg1

*(operation, arg1, arg2, variable)*表示需要进行运算操作，arg1 operation arg2，将结果赋值给variable

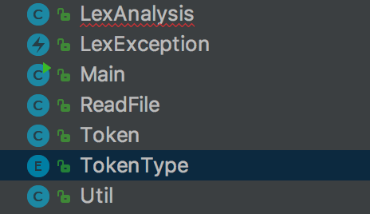
*(type,value,\_,variable)*表示需要申明一个变量，将value的值赋值给variable

*(\_, arrname, index, variable)*表示需要申明一个临时变量来存放数组中的某一个值

在语义分析过程中，产生中间代码数组，然后传入中间代码进行实现。

1. **其他语法**

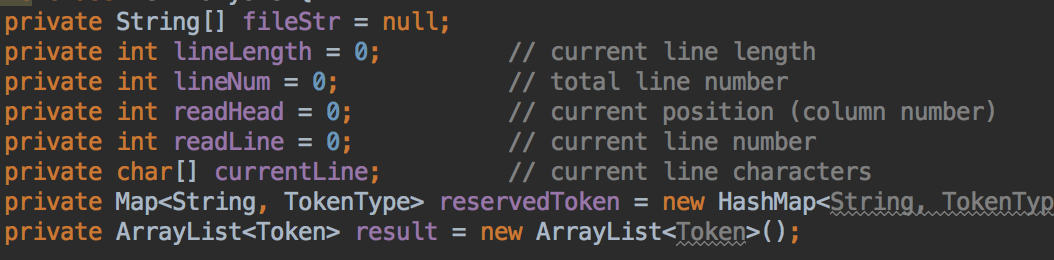
添加了for循环

1. **代码实现**
2. **词法**
3. **总体结构**：

程序包含上述几个文件，LexAnalysis为词法分析类，定义了词法分析的方法和词法分析中一些关键的数据结构；LexException是一个异常类，在词法分析中出现错误时，抛出LexException异常；Main是主函数，调用输入文件，解析，调用词法分析，输出等操作；ReadFile是用来读取源文件的类，将其建为类是为了方便之后的扩展；Token类是识别到的token，定义了token的类别及其位置；TokenType如上所述；Util定义了输出方式，虽然只有一种输出，但为了方便扩展将其建为单独的类。

1. **主要数据结构说明：**

LexAnalysis类中，定义了如下数据结构：



fileStr是读入源文件后将每一行解析出来的字符串数组

lineLength是当前行的长度

lineNum是该源文件的行数，即fileStr.length

readHead是当前读取头，即在当前行中的正在读取的位置

readLine是当前读取的行数（方便记录位置）

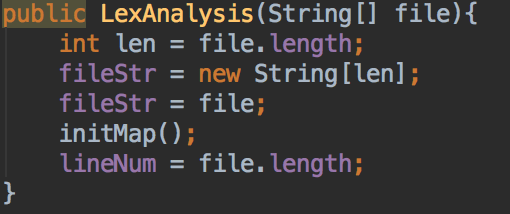
currentLine是当前正在读取解析的行的字符数组

reservedToken是保留字的集合，用于判断字符串是否为保留字

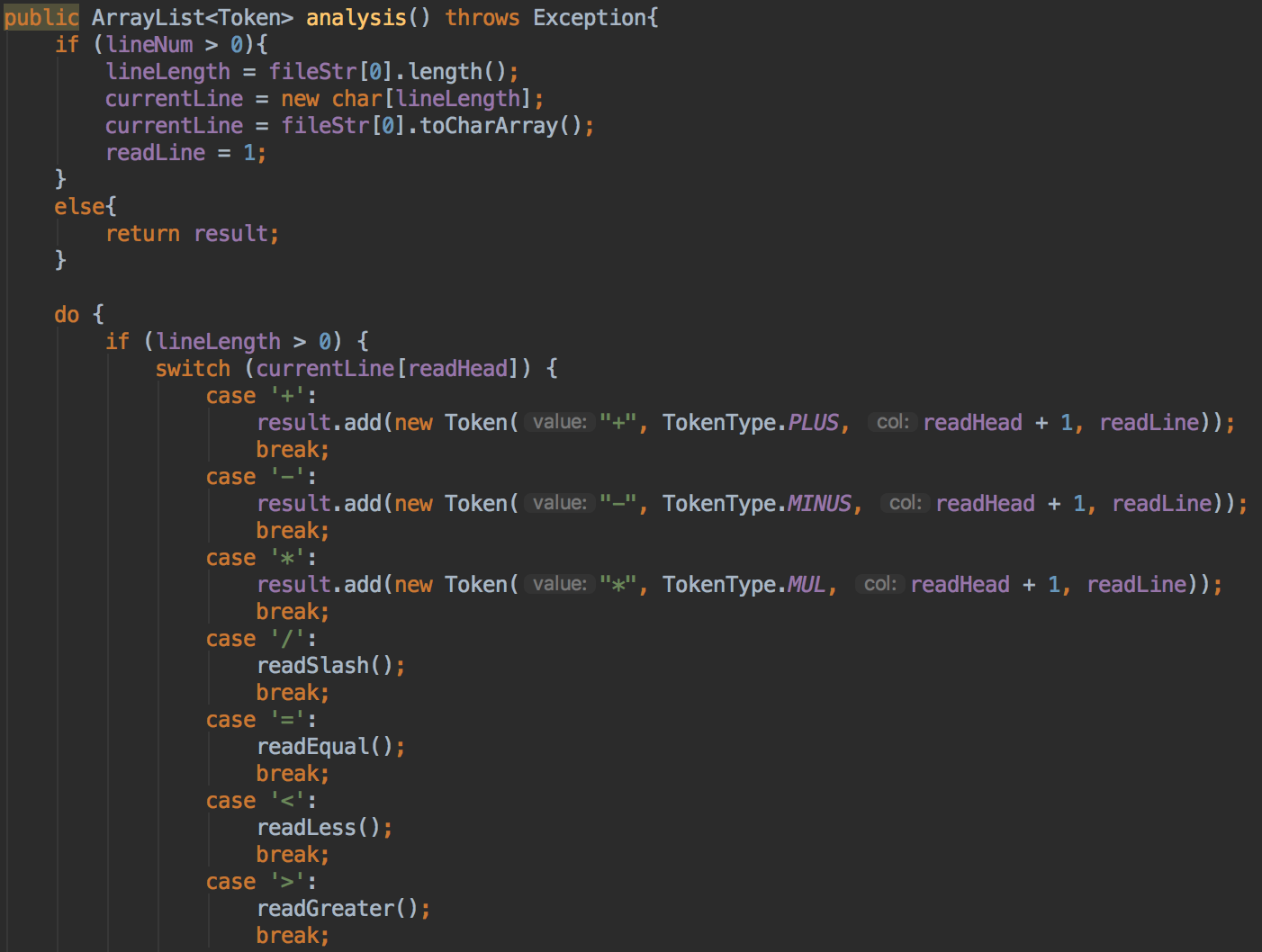
result是词法分析的结果数据结构

1. **算法说明：**

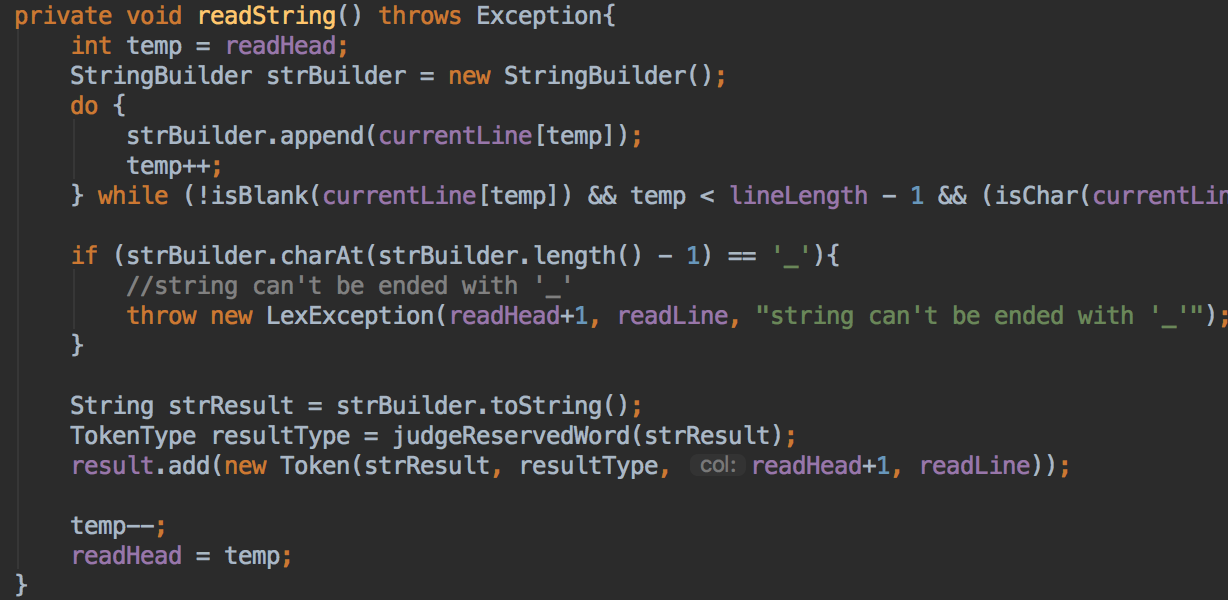
主要说明LexAnalysis中的算法：

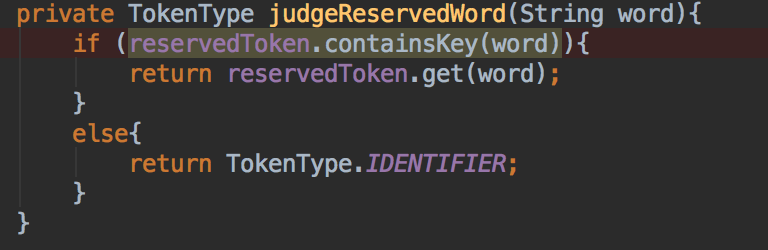


首先在创建词法分析器对象时，对一些数据结构进行初始化操作，给fileStr和lineNum赋值，将所有的保留字添加到reservedToken中



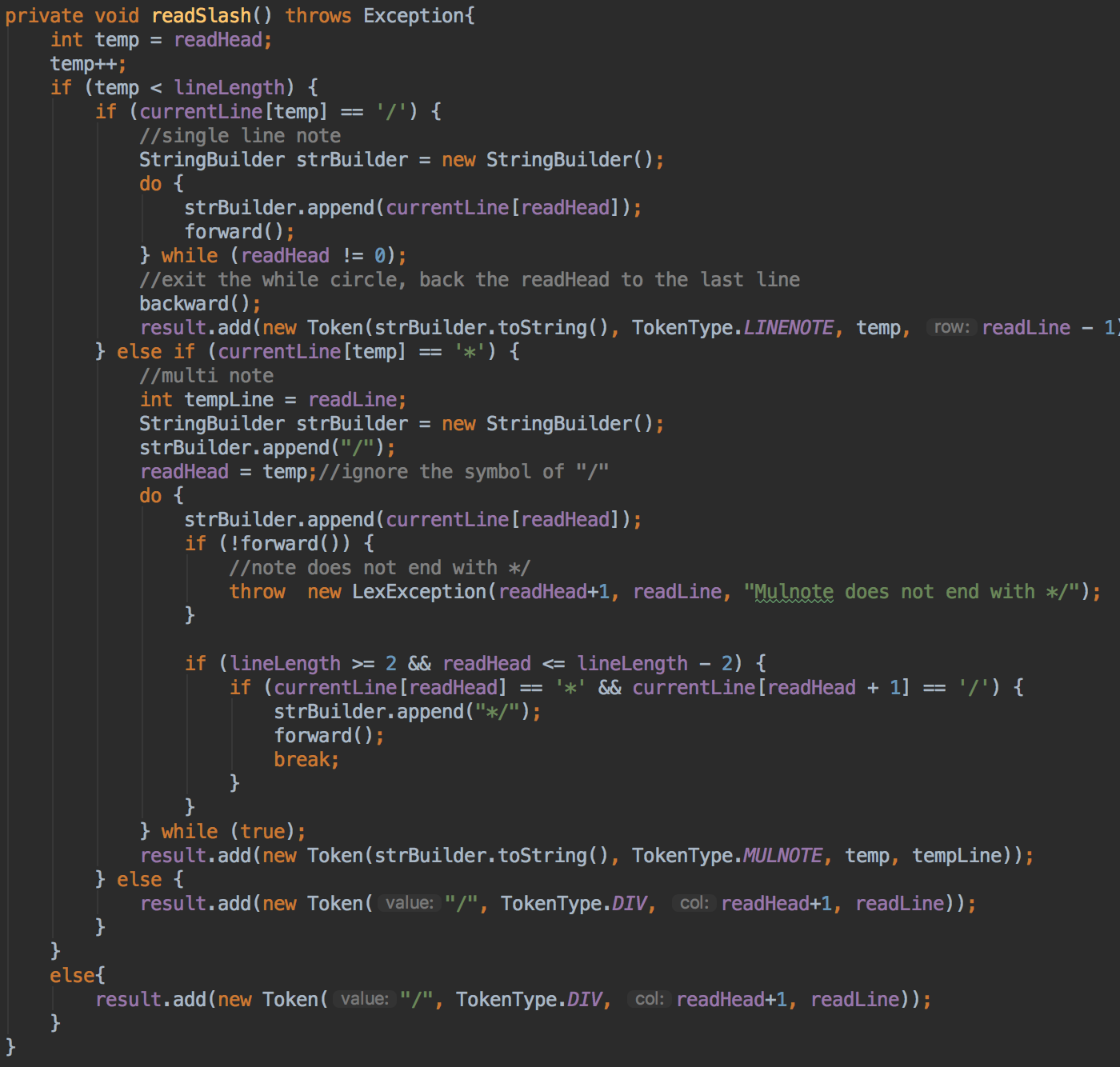
然后开始进行词法分析，首先判断源文件中有无字符，有的话将调整readHead和currentLine到第一行。然后开始逐字分析，如果单个字符即可确定其类型，例如’[’，’\*’等，直接声明Token对象，将其添加到result数组中，否则跳转到函数中进行处理。跳转的处理函数如下，以String解析为例：





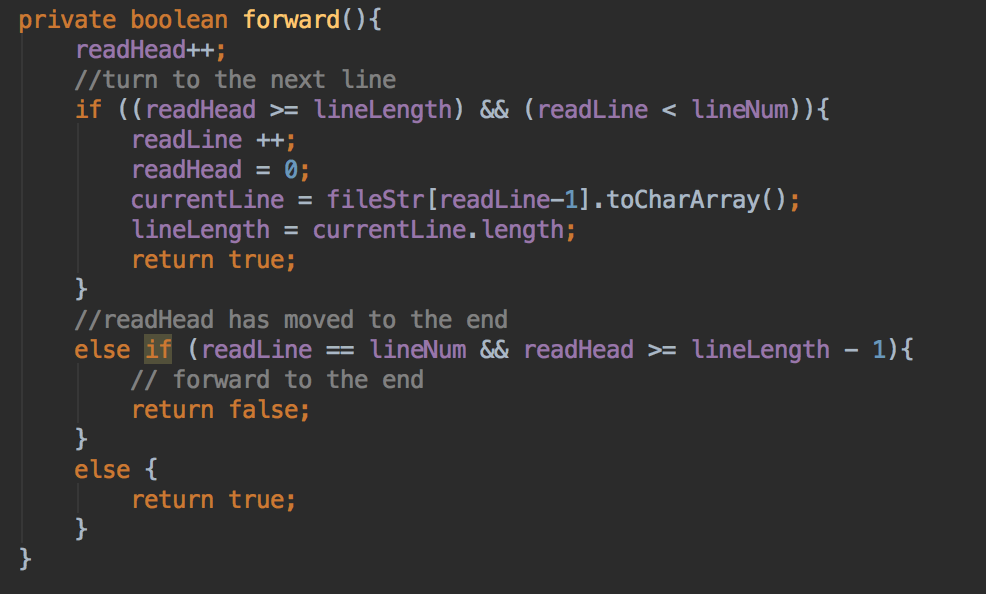
先声明一个temp整形变量，用来代替readHead进行解析，以防因错误导致readHead无法复位，使用temp向后移动，并将其加入到stringBuilder中，直到遇到空格或者到达行末。然后判断最后一位是否为’\_’，最后判断最终识别的字符串是否为保留字，然后将识别的字符添加到result中，并修改readHead的位置到最后一位读取字符所在的位置。

读取注释算法如下图：



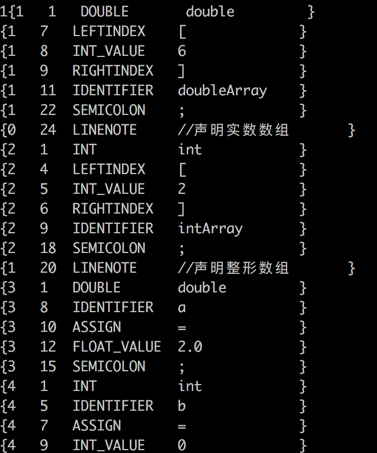
注释以/开头，所以要判断是除号，单行注释还是多行注释，所以从/开始解析。同上，声明temp变量来将指针向后移动，如果遇到/则为单行注释，循环添加识别到的字符直到到下一行；如果遇到\*则为多行注释，再声明一个tempLine来进行行的移动，循环添加字符，直到遇到\*/或者无法继续移动；如果遇到其他字符，则为除号，添加到result中。

接下来对forward函数进行说明：



forward即readHead向前移动的函数，分为三种情况：换行，移动，终止；（判断过程如上图，主要是要防止发生数组越界）分别对其情况进行判断，然后返回移动结果。

1. **最终结果**



1. **语法**

语法分析中关键节点为树节点，其中定义了nextLayer（下一层的所有树节点），type（此节点的类型，即非终结符），value（如果此节点类型为VALUE，即已经匹配到终结符，value的值为此终结符token的值）运用此树节点完成了一个链表结构，是构造语法分析树的关键数据结构。

具体方法：setNext：为下一层节点集赋值

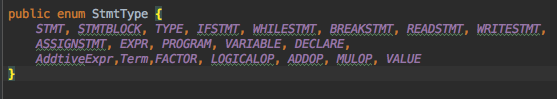
toString：输出此节点的值，非终结符或者匹配到的终结符

setValue：匹配到终结符时调用，将终结符token传入。



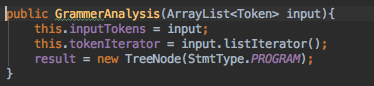
树节点的类型有如下几种：

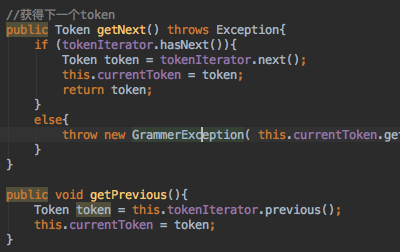
分别对应文法中的非终结符和一个匹配成功的终结符VALUE



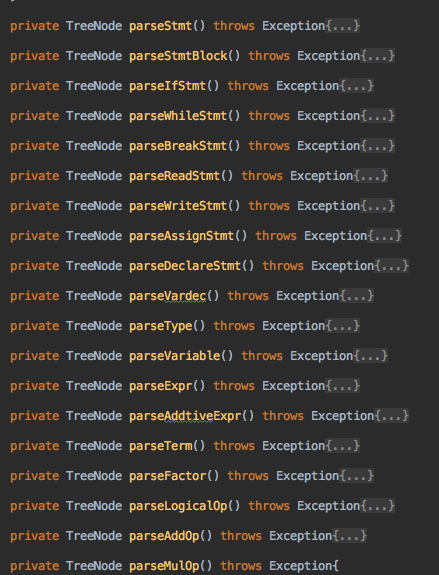
语法分析器初始化函数如下：

将输入的token序列保存，存储为迭代器类型，然后申明根节点，定义为PROGRAM类型，由此节点开始向下延伸。

  
语法分析器中，使用迭代器将输入的token数组进行迭代，迭代器识别下一个和回到上一个的操作函数如下图：

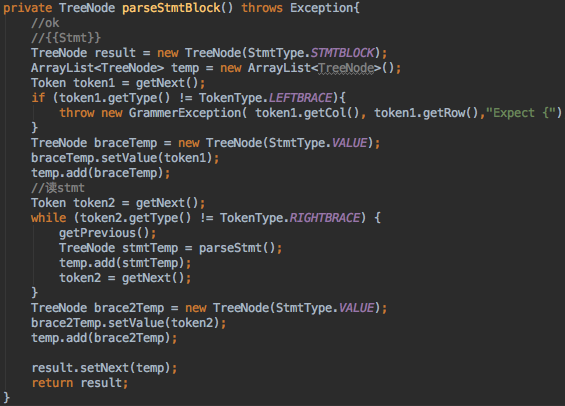


识别函数如下，为每一个文法规则建立一个函数进行识别：



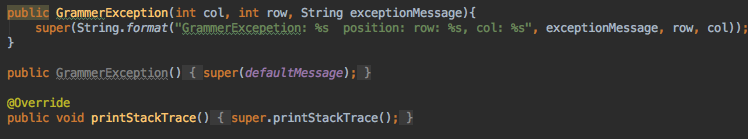
用识别stmtblock作为例子进行说明：

函数结构如下，当识别到 { token时，要进入stmtblock进行分析，首先要申明一个类型为STMTBLOCK的树节点，因为它不是终结符，所以要为它申明下一层节点的数组，然后进行识别，判断第一个token是否为 { ，如果不是要进行错误处理，然后继续识别stmt非终结符，调用parsestmt函数进行识别，然后返回识别完成的树节点，最后识别 } ，同样如果识别错误进行错误处理，如果识别到了，要将其存储为VALUE类型，即识别成功的终结符，一起存入下一层节点数组中，然后将下一层节点通过setNext函数赋值传给此节点。



错误处理函数如下：

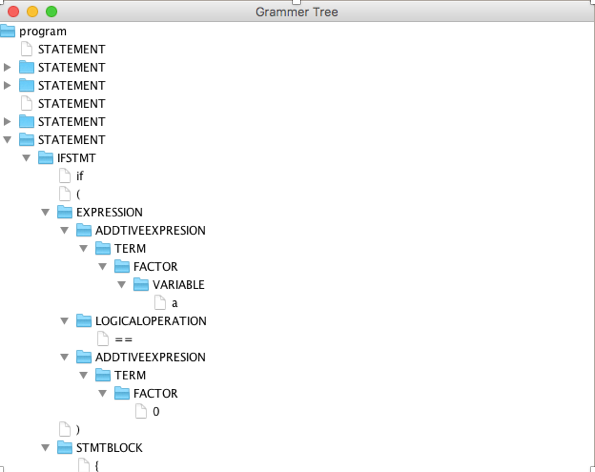
继承自Exception，传入错误的行列数和错误信息，然后抛出异常



输出函数：

本程序中调用了JTree类来显示最终的语法树，因为语法分析最终的结果是一个TreeNode的链表，所以要将其进行遍历，将每个节点输出到语法树中。这里采用的是深度遍历，递归调用DFS，最终遍历得到语法树。

输出结果如下图所示：

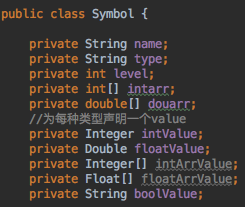


1. **语义**
2. **四元式的定义**



用一个类申明了中间代码的形式，内部有四个变量，都为String类型，在构造时就要进行填充；backfill函数用来对四元式进行回填操作；toString函数用来输出四元式为(a,b,c,d)的形式

1. **符号表的定义**

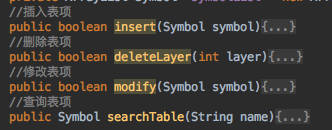


首先是符号的申明，每个符号都对应一个类型，一个符号名，一个值以及对应的层次，因为在进行构造的时候，传入的都是String类型，不方便进行之后的操作运算，所以为每种类型都申明了一个value变量，构造时对类型进行判断，然后赋值到对应的value中。



然后是符号表，使用一个ArrayList来存放所有符号，一个符号对应一个Symbol类型变量。

然后定义了如下几个方法对符号表进行操作：

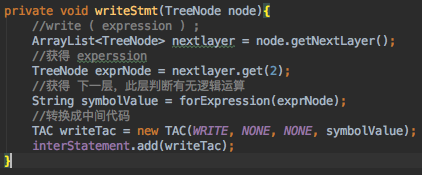


在这里主要阐述一下删除操作。删除操作传入的是int型变量，因为变量在申明之后不会释放，除非变量层发生了变化，比如说进入了while循环，其中申明的变量全部为临时变量，当跳出while循环时，刚刚申明的变量需要全部释放，这时候才需要调用删除函数，对所有该层以及该层以上的函数进行删除 ；删除时因为是在遍历符号表循环中进行，所以不能直接对符号表进行修改，要先记录下符号下标，然后从后往前进行逐个删除。

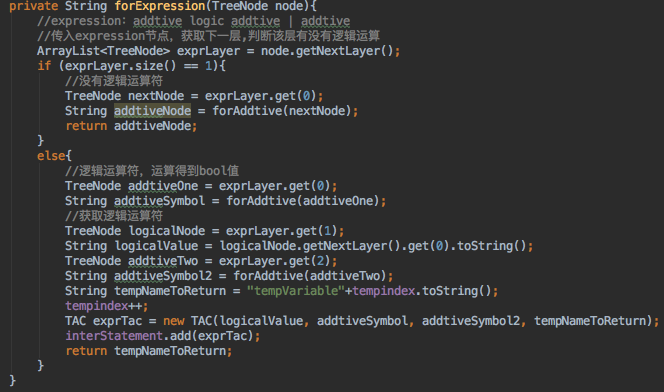
1. **语义分析**

为每一个语法分析产生的非终结符节点申明一个方法，用来进行语义分析，得到四元式。对需要获取非终结符值的节点，需要返回String变量，用来表示遍历得到的终结符或者计算得到的临时变量。

这里用write(1 + 2)；作为例子进行阐述，首先语法分析树中该句为一个Statement节点，进一步需要解析writeStatement节点，进入该函数：



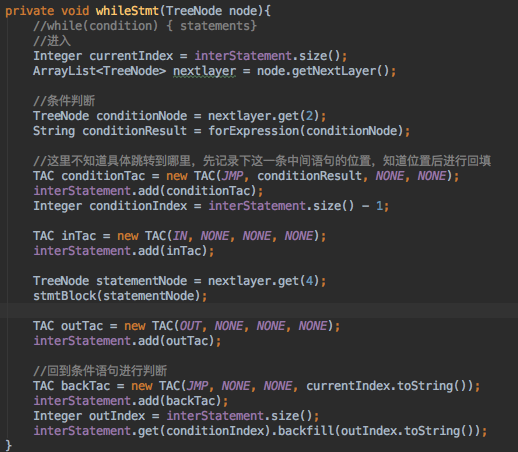
我们通过获得下一层的第3个节点，也就是expression节点，可以得到我们要进行打印的值，这是要调用forExpression方法，获取要打印的变量或者数字。



在该方法中，我们需要解析逻辑运算，然后跳转到foraddtive方法解析加减运算得到term + term，然后跳转到forTerm方法中解析乘除运算，最后跳转到forFactor方法中得到我们所要的1 和2，然后回到forAddtive方法中计算1+2的值并赋值给临时变量，添加四元式为(+,1,2, tempvariable0)，然后将tempvariable0返回给forexpression进而返回给writeStatement，添加四元式(write,\_,\_,tempvariable0)。

在这里解释一下临时变量的赋值操作，定义了一个int型私有变量，用来控制临时变量的不重复，每次申明之后变量加一，通过字符串拼接得到一个临时变量值。

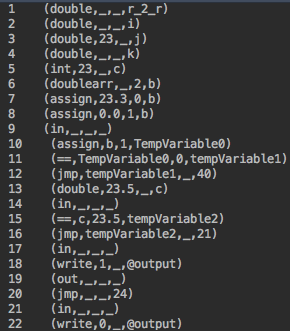
另外需要说明一下回填过程，在while和if语句解析的过程当中，需要需要加入跳转，如果不满足条件，就要跳转到while循环之外，但是我们在语义分析过程中暂时不知道该位置在哪，所以先将address是申明为NONE，在while循环语义分析结束时再对address进行回填得到完整的四元式，代码如下：



对于运算也需进行一些说明，在运算操作时，例如3+5\*4/5-9，得到的四元式顺序是/,\*,-,+计算结果与实际不符，是因为语义分析中按照计算的优先级顺序进行解析（逻辑，加减，乘除），且同等级是从右向左进行解析（因为语法规则中消除了左递归）所以造成了上述错误，所以我们需要对forTerm，forAddtive函数进行改造，直接在函数中进行循环，而不是递归调用，将运算顺序改为从左到右，函数如下：



语义分析得到的最终结果为：



1. **中间代码解释执行**

中间代码实现部分接受语义分析产生的中间代码的list作为唯一参数，对list中的中间代码依次进行读取，根据其操作符的不同，调用不同的方法。最终正确输出了给出的所有测试用例的结果。对输入的cmm代码进行包括类型检查；变量的作用域与声明检查；除数不为0；数组下标检查；语句中的布尔表达式的类型检查等操作，在错误时报错，在正确时将其存入符号表中。

1. **设计思想**

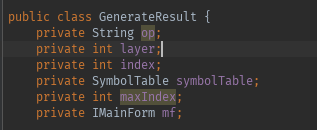
在完成中间代码的实现部分时，按照以下的设计思路来进行，首先确定与语义分析的联系，完成语义分析的同学提供了三个类，为Symbol，SymbolTable和TAC。其中Symbol类为定义一个新的symbol，也就是符号，包括了这个符号的类型，名字，值以及层数等等。SymbolTable主要定义了一个符号表，并编写了对于符号表的增删改等操作，最后是TAC，就是传输的中间代码。

中间代码实现的工作就是，获取tac并判断其类型，根据类型的不同调用不同的方法，并在一些方法中，如intStmt，assignStmt实例化Symbol对象并赋值，最终调用symbolTable的方法将其存入符号表。具体的数据类型接下来进行详细阐述。

1. **主要数据结构和算法声明**

中间代码的实现部分，只需要一个类就可以完成，在这里命名为GenerateResult，就是将最终的结果展示出来。

首先为GenerateResult定义了一些私有变量，如下图。



op为获取到的中间代码的操作符；

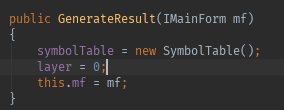
layer为层次，用来控制变量的层级关系，从而实现变量作用域的控制；

index为当前的数组下标，之所以定义为私有，是因为在存在判断条件时，需要跳转到指定的行数，此时需要一个私有变量来控制最外层的循环；

maxIndex也是类似，是为了防止跳转出现的越界的问题；

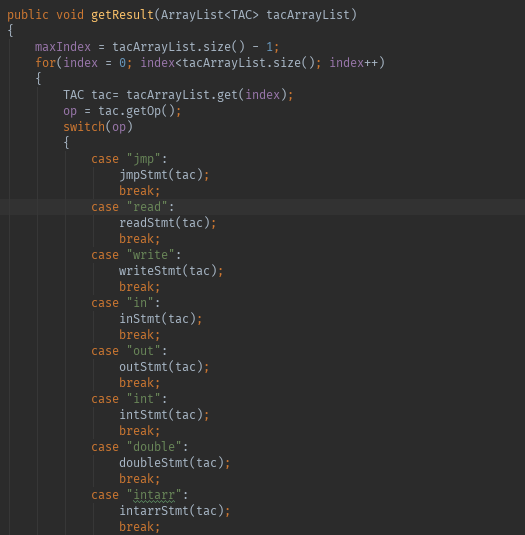
symbolTable则是为了调用符号表的方法，因此定义为私有变量；

IMainForm是GUI的接口，用来将打印的结果展示在GUI界面中。

构造函数

非常易懂，接受接口为参数，将GUI部分生成的窗体导入mf中。

接下来对函数进行介绍，首先是一个总体的函数。它接受TAC类的ArrayList为参数，对中间代码的每一行依次进行操作。采用switch case方法，在识别到不同的op时进行不同的操作。如下图。



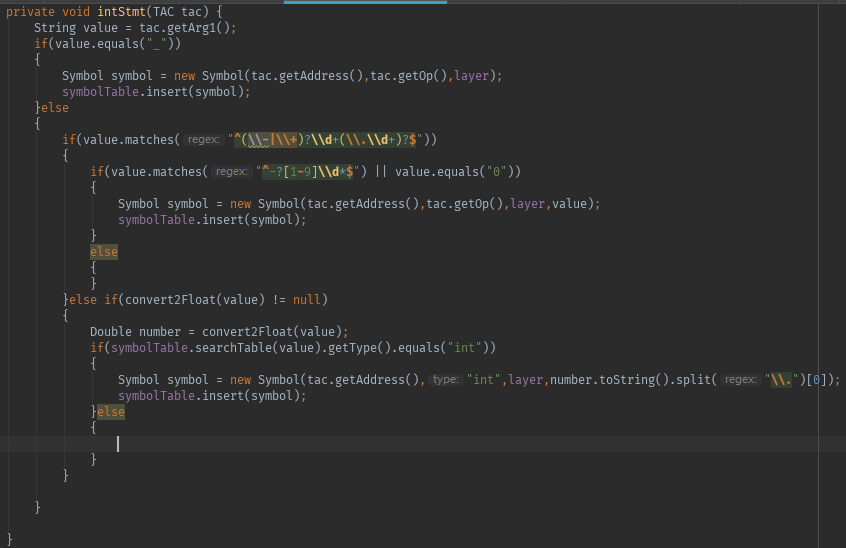
具体的实现函数介绍

在完成整个中间代码的实现后，总结一下发现实现函数可以如下几个类别来进行讲解

1. 声明部分

声明部分即字面意思，根据数据类型，声明变量，在cmm中只能声明为int，double，int[]，double[]四种类型。因此包括四个函数，intStmt()，doubleStmt()，intarrStmt()，doublearrStmt()。前两个是整型和浮点型，后两个为他们的数组形式。

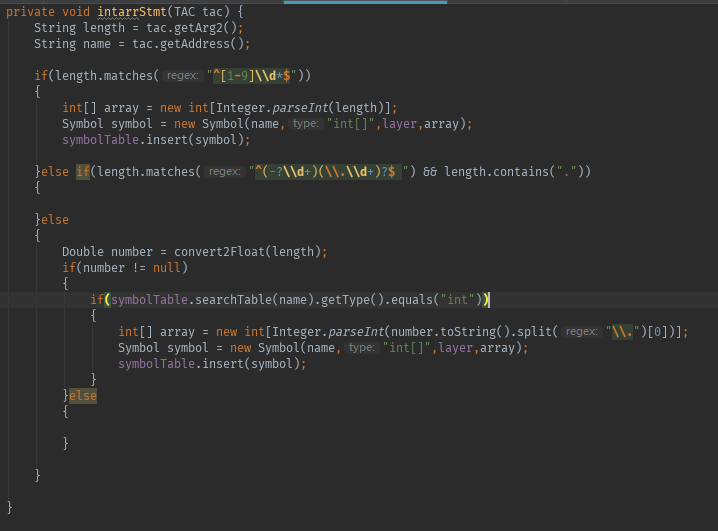
对于数组，cmm不支持在定义时就赋值的语法，因此只有前两种需要考虑赋了初值的情况。代码如下（int型，double型类似）



首先判断是否赋初值（即value.equal(“\_”)）,直接实例化Symbol，并添加到symbolTable中，如果赋值，判断是否为整数，如果是整数则实例化，如果是浮点数，就报错；接下来判断value为变量的情况，首先调用convert2float方法，如果返回值不为null说明存在这个变量且有值，判断这个变量的类型，是int时赋值，是double时报错。

对于double，赋的值无论是int还是double都是符合语法规则的。

再说数组的情况，代码如下（以int型数组为例）

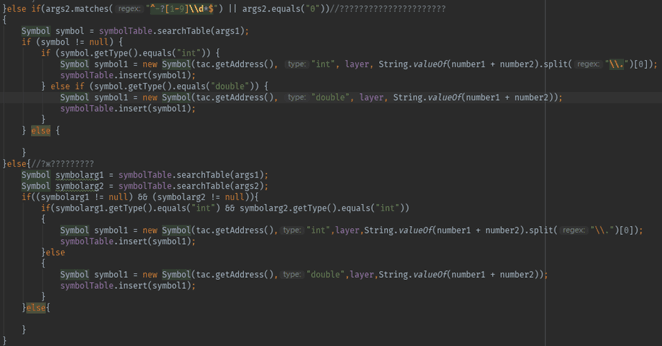
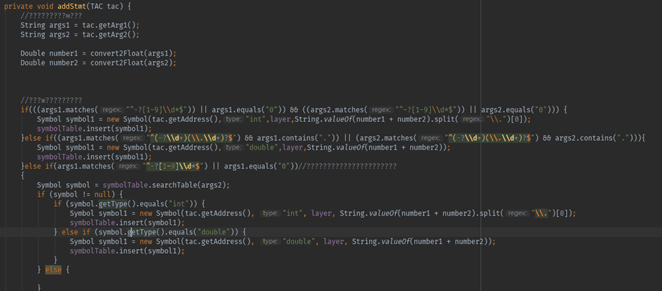


首先要判断长度是否为整数，或者其为变量，判断其类型是否为int，是的话就实例化symbol，数组的symbol构造函数不同，会为其新建一个名为array，长度为length的数组。

1. 加减乘除部分

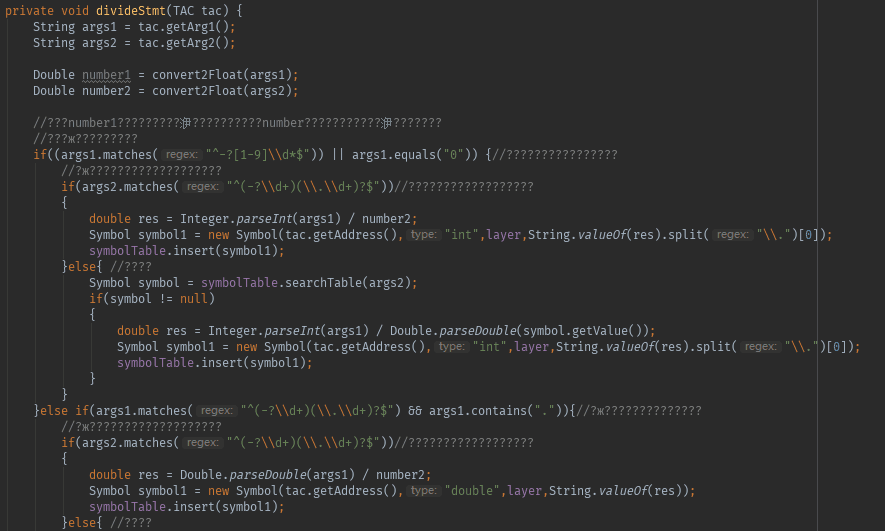
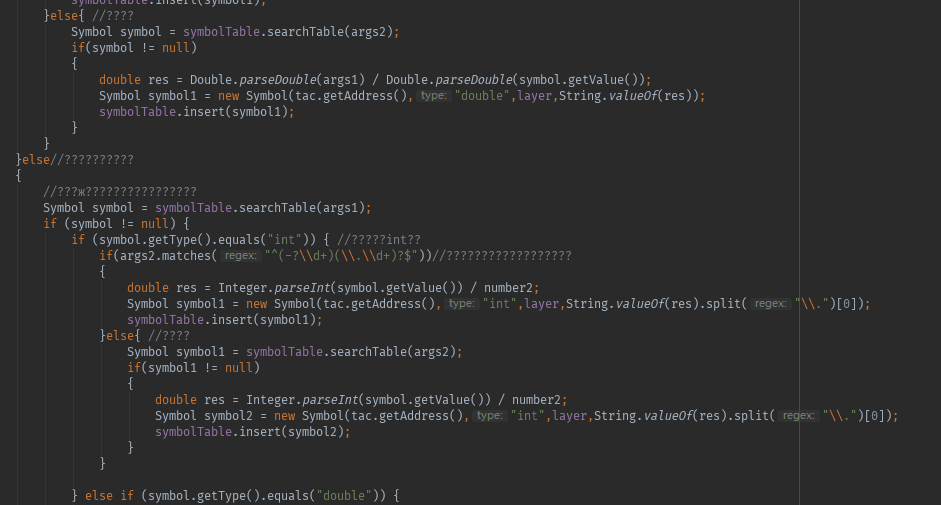
包括了addStmt()，subStmt()，mulStmt()，divideStmt()四个函数，其中比较类似的是add，sub和mul，因为他们是必须两个参数都是int，最终的结果才是int型，而div是如果被除数是int，结果就是int，被除数是double，结果就是double，与第二个参数无关。

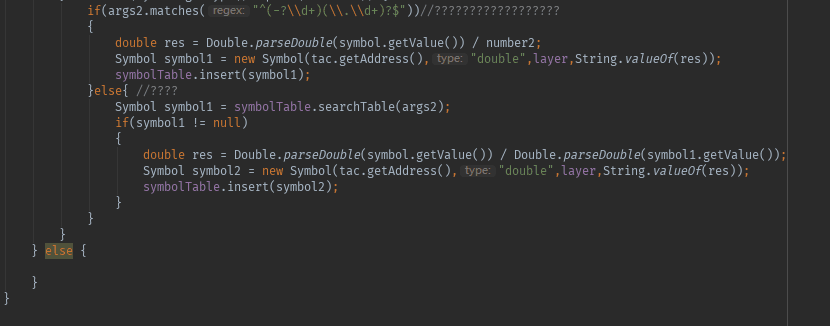
首先看addStmt(),sub和mul类似。



逻辑如下，首先判断两个参数是不是都是整数，如果是就实例化symbol，类型为int，再判断各自是不是double，如果有一个是就实例化symbol，类型为double。再判断第一个是整数，第二个是变量的情况。如果变量类型为int，就是int，如果变量类型为double，就是double；接下来是第一个是变量，第二个是整数，方法类似。最后是两个都是变量的情况，再次判断。

接下来看divideStmt()。

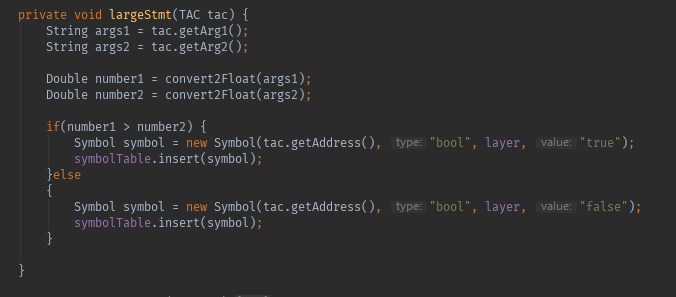
 



逻辑如下，首先判断第一个参数是否是整数，如果是，无论第二个参数是数字还是变量，将值提取出来，存为int型；再判断第一个是否为浮点数，同上，最后是第一个参数是变量，首先判断其类型，接下来同上。

1. 条件判断部分

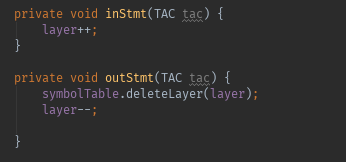
条件判断部分包括了大于，小于，等于，不等于，大于等于和小于等于这几种情况，方法类似，以largeStmt()为例子讲解。



当正确时，实例化一个类型为bool的symbol，值为true，反之为false。

1. 层次修改部分

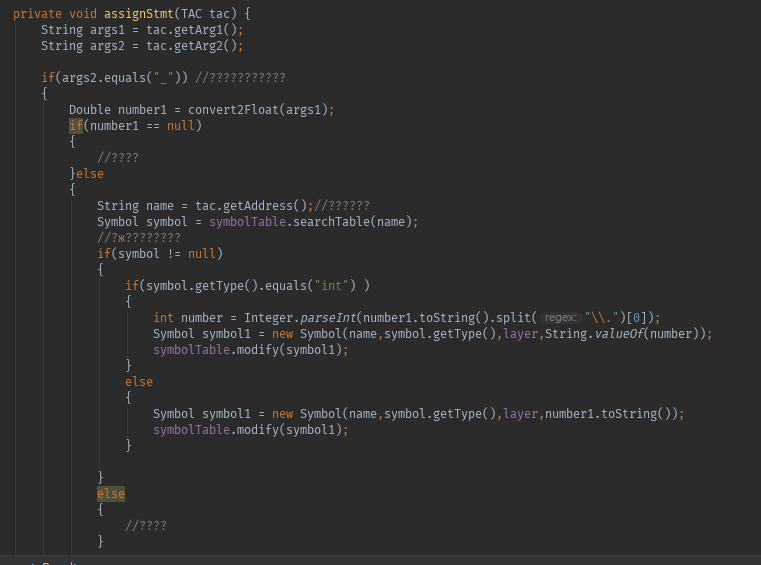
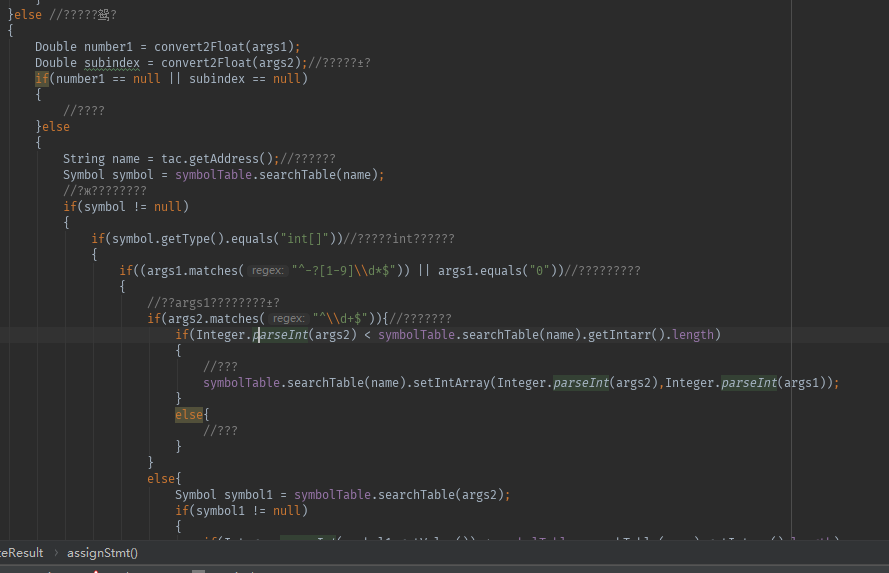
层次修改包括inStmt()，outStmt()。代码如下



当遇到｛时，调用inStmt，将layer++。遇到 ｝时，将当前layer的变量全部删除，再把layer--。

1. 赋值部分

赋值部分可以说是非常重要的部分。首先对于语义分析给出的中间代码，存在给临时变量赋值的情况，而这会导致无法判断是该变量究竟是未声明还是临时变量。因此为了避免这一问题，用op = “assign”代表存在变量，op = “\_”代表临时变量。代码如下

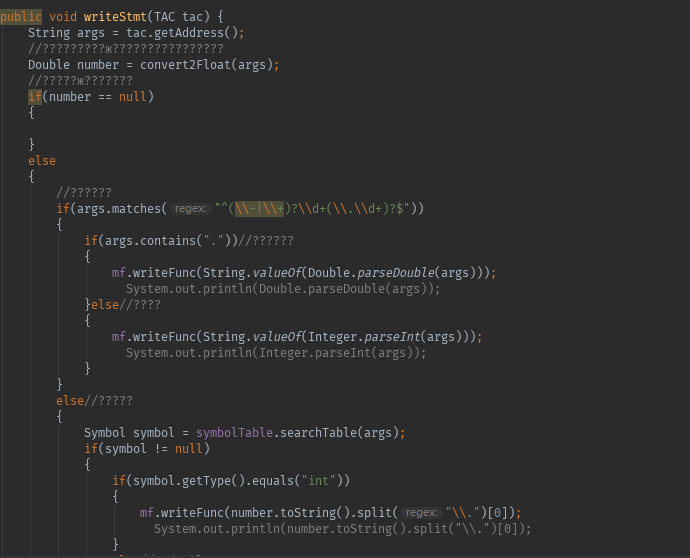
 

过长未完全展示

由于情况过多，且为了给其他的方法减小工作量，因此assign函数进行了大量的判断，大致逻辑如下，首先判断是否是给数组赋值，如果不是，判断value是数字还是变量，并且类型要一致；如果是，先判断是int数组还是double数组，接下来判断value是数字还是变量，数字和变量必须与数组的类型一致。

1. 读写部分

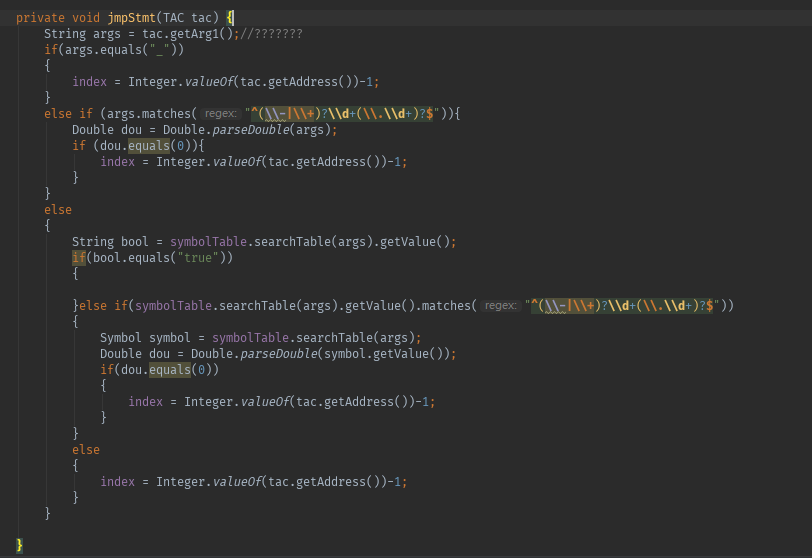
包括writeStmt()和readStmt()函数，代码如下（writeStmt为例）



如果是数字，判断是浮点还是整数，调用接口的writeFunc()打印结果。如果是参数，判断是int还是double，输出。

1. 跳转部分

只有jmpStmt函数来实现跳转的功能。代码如下



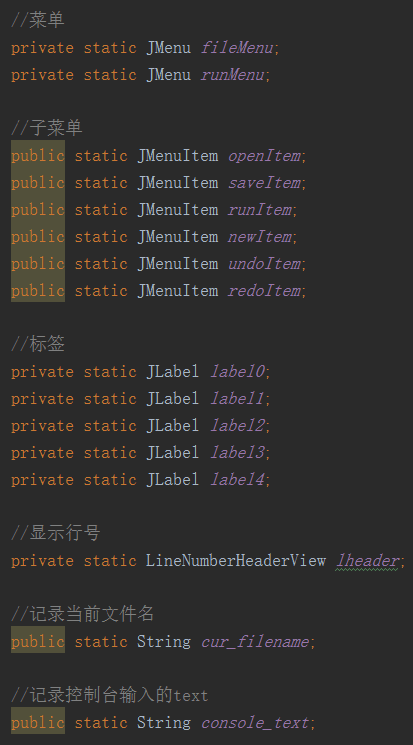
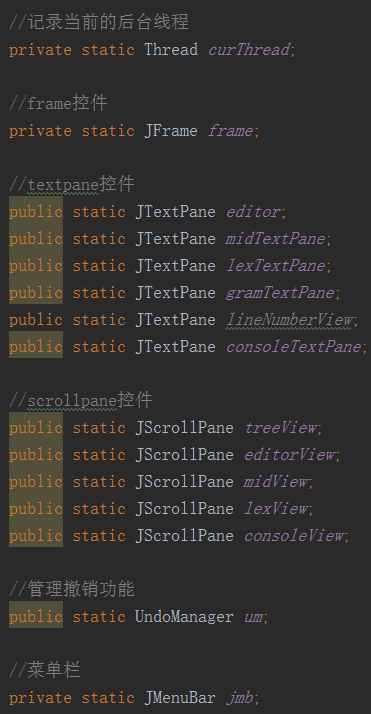
如果args.equals(“\_”)则无条件跳转，否则，判断如果是数字，分为0和非0两种情况。除此之外判断其类型，分为true和false两种情况。对index值进行修改从而达到跳转的目的。

1. **GUI**
2. **概述**

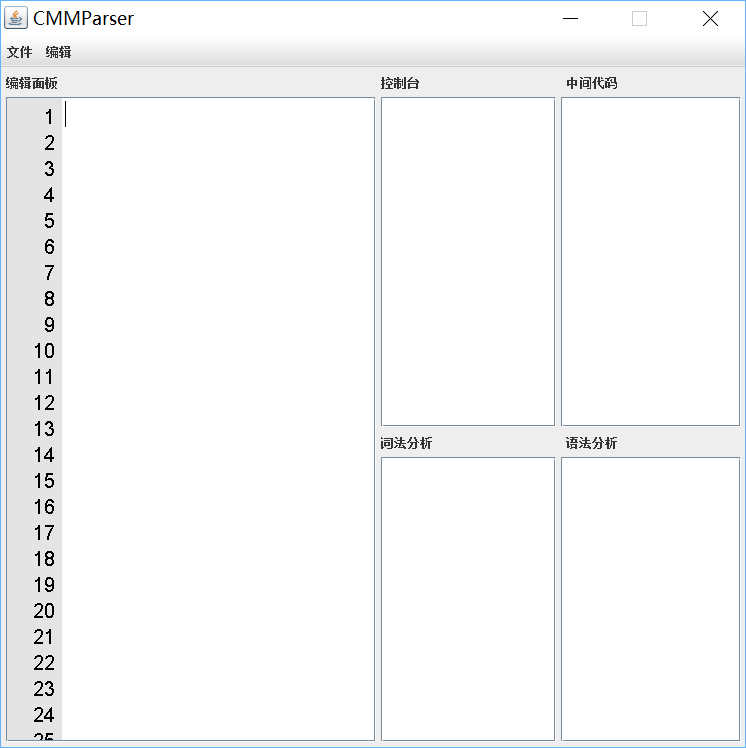
GUI部分包括：

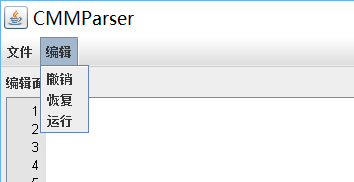
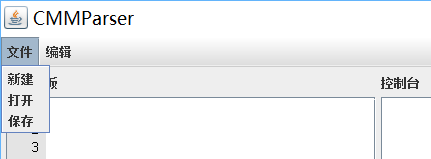
1. GUI的界面设计；
2. 界面显示行号；
3. 关键词高亮；
4. 新建、打开、保存文本文件功能，撤销和恢复文本功能，解决编码问题，统一使用GBK编码；
5. 运行功能；
6. 使用多线程分离复杂任务；
7. **具体内容**
8. **GUI的界面设计**

此次的GUI界面设计没有使用IDE提供的界面拖拽功能，完全由代码实现。新建一个MainForm类，类中包括以下属性：

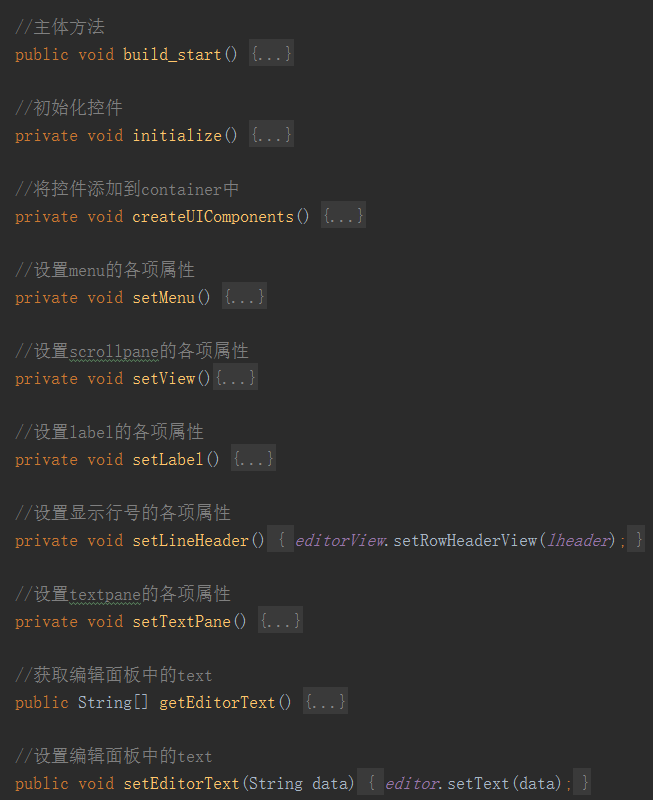


大部分为界面设计用到的控件，暂不详述。设计界面如下：





MainForm中方法如下：



其中：

1. 使用默认构造函数；
2. 主体方法为build\_start；
3. 在build\_start函数中调用initialize（初始化控件）和createUIComponents（将各个控件添加到container中）；
4. createUIComponents在添加控件之前调用setMenu，setView，setTextPane，setLineHeader，setLabel等函数具体初始化各个控件。
5. getEditorText获取编辑面板的内容，传出一个处理后的String数组；
6. setEditorText，setMidText，setLexText，setGramText用来设置编辑面板、中间代码、词法分析、语法分析面板的内容，包括添加Listener，设置位置大小等；
7. 语义分析遇到read时，调用readFunc函数阻塞后台线程，使用户在前端输入数据，监控用户输入，输入结束后（按下回车键）把输入内容传递给后台，继续运行后台线程；
8. 语义分析遇到write时，调用writeFunc函数在控制台输出结果；
9. CMM解析任一阶段出现问题，调用reportErr函数，在控制台输出结果；

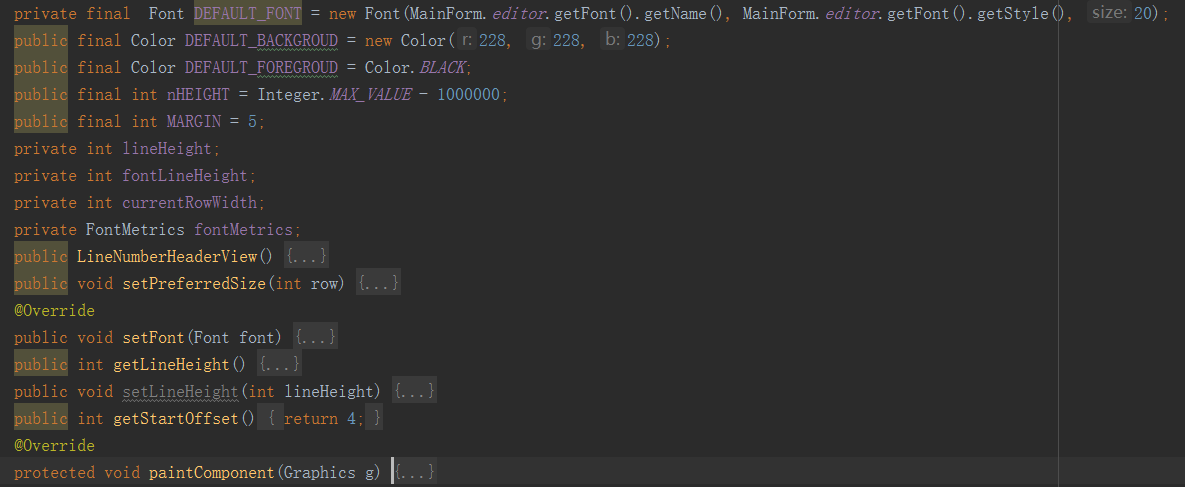
新建了一个IMainForm的接口类，提供以下方法：



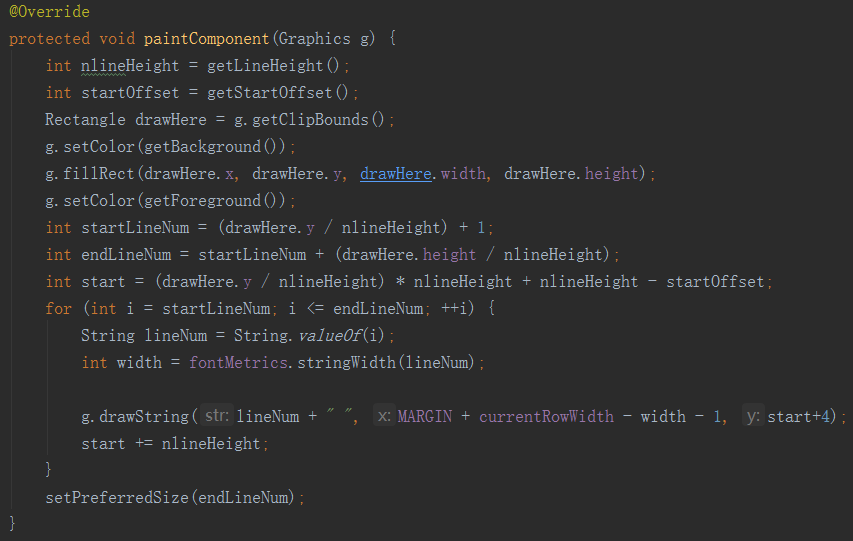
让之前创建的MainForm类实现这个接口，可以提高代码的封装性和安全性。

1. **界面显示行号**

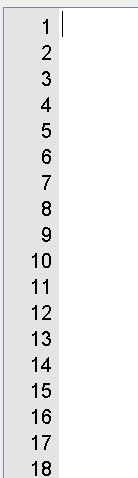
新建一个LineNumberHeader类，继承自Jcomponent，属性与方法如下：



主要是重写了其中的paintComponent，对组件进行绘制，具体代码如下：

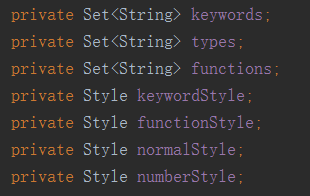


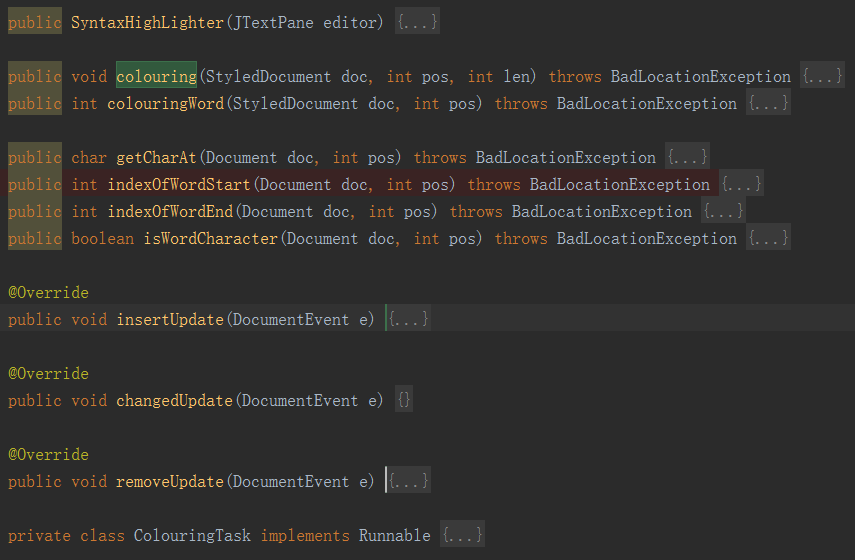
按照设定的行高、字体、颜色来绘制组件。

效果如下：

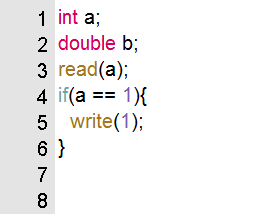
1. **关键词高亮**

新建SyntaxHighLighter类，来实现DocumentListener接口，属性与方法如下：





重写了DocumentListener接口中的insertUpdate，changedUpdate，removeUpdate方法，当编辑面板增加或删除字符时调用Listener中对应的监听方法，在监听方法中再调用colouring函数，判断已经输入的组合在一起的单词、数字等是否为关键字（if，else，while，for，read，write，for，数字等），若是则新建一个线程进行内容替换（把普通的颜色替换为关键字的颜色），若不是则不替换颜色。

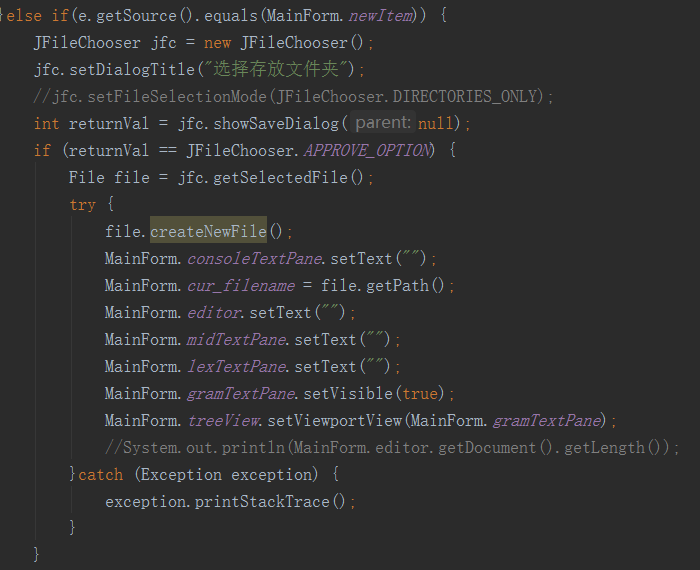
效果如下：

不仅仅是输入，打开已存在的文件时，高亮效果同样存在。

1. **新建、打开、保存文本文件功能，撤销和恢复文本功能，解决编码问题，统一使用GBK编码**

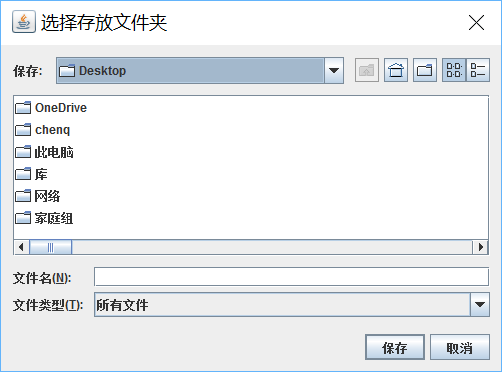
新建了一个MenuItemListener类，实现ActionListener接口，用来监控菜单项的点击：

监控“新建”菜单项点击的代码：

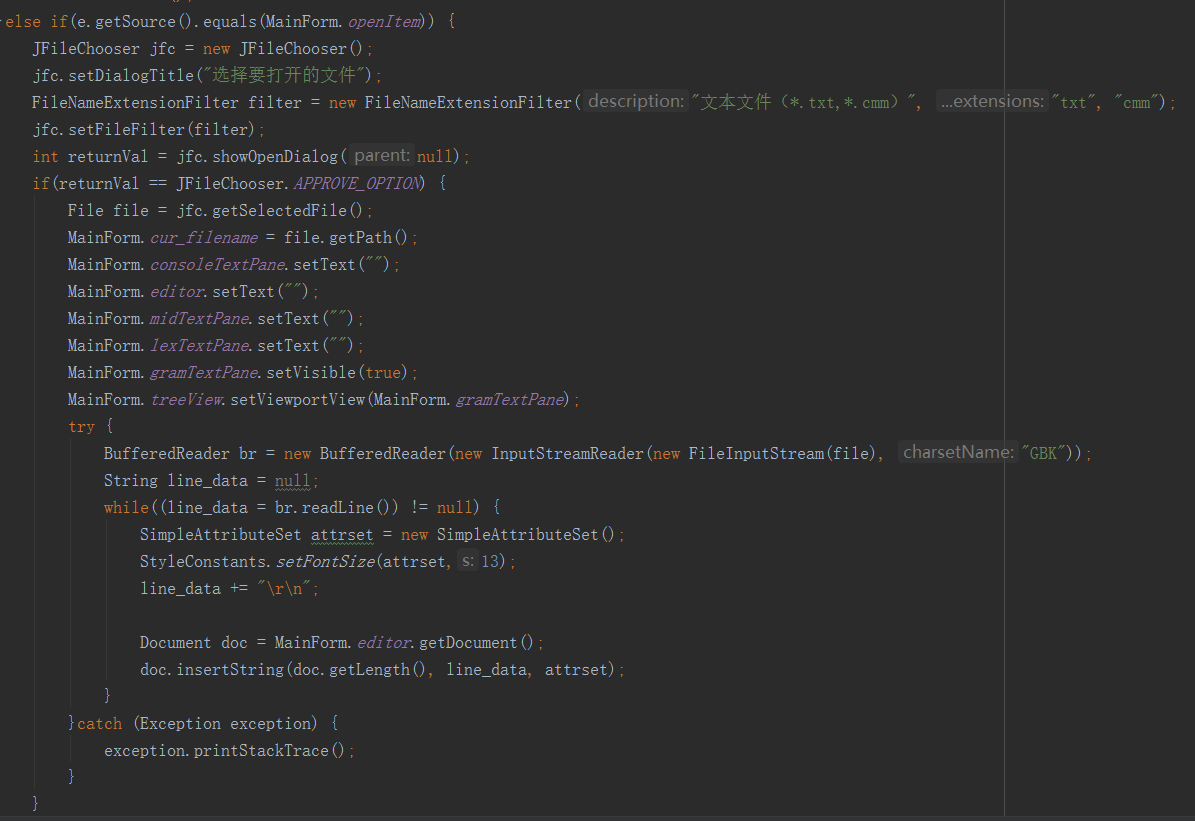


使用JfileChooser控件来实现此功能，当新建成功后初始化MainForm里的内容。

点击“新建”效果如下：

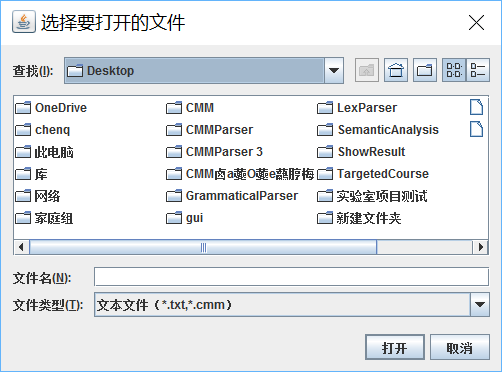


监控“打开”菜单项点击的代码：

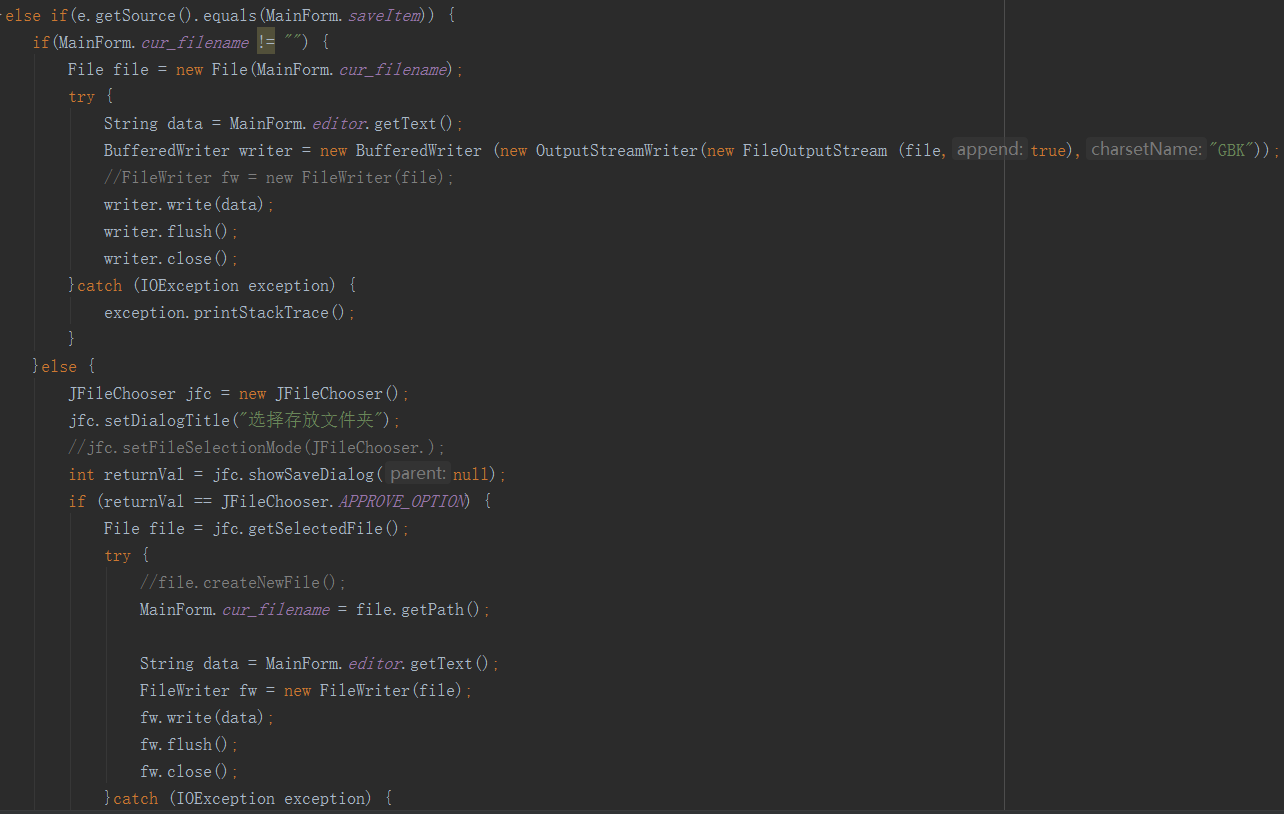


使用JfileChooser控件来实现此功能，当打开成功后初始化MainForm里的内容，设置被打开的文本的编码为GBK，防止中文乱码。

点击“打开”效果如下：

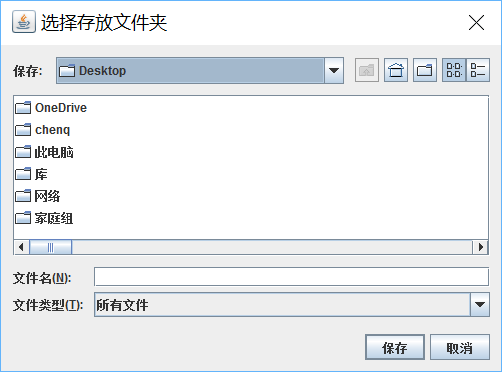


监控“保存”菜单项点击的代码：



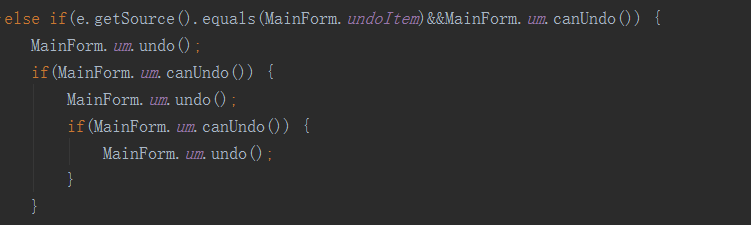
使用JfileChooser控件来实现此功能，当前已有打开的文件时，将编辑面板中全部内容写到原文件中，当前没有打开的文件时，新建一个文件，并存储编辑面板里的内容。保存时设置将要存储的内容编码为GBK，防止中文乱码。

当前无打开的文件时，点击“保存”效果如下：

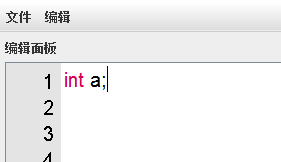
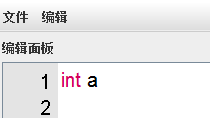


当前有打开的文件，点击“保存”直接保存。

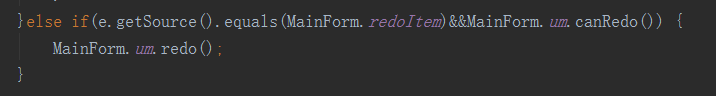
监控“撤销”菜单项点击的代码：

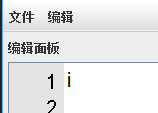
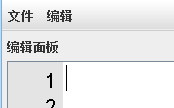
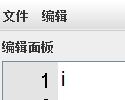


由于实现了着色功能，仅使用一次undo无法退回到输入字符之前的状态，只能退回到颜色、字体替换之前状态，所以需要三次undo。

实现效果如下：点击撤销后：

监控“恢复”菜单项点击的代码：

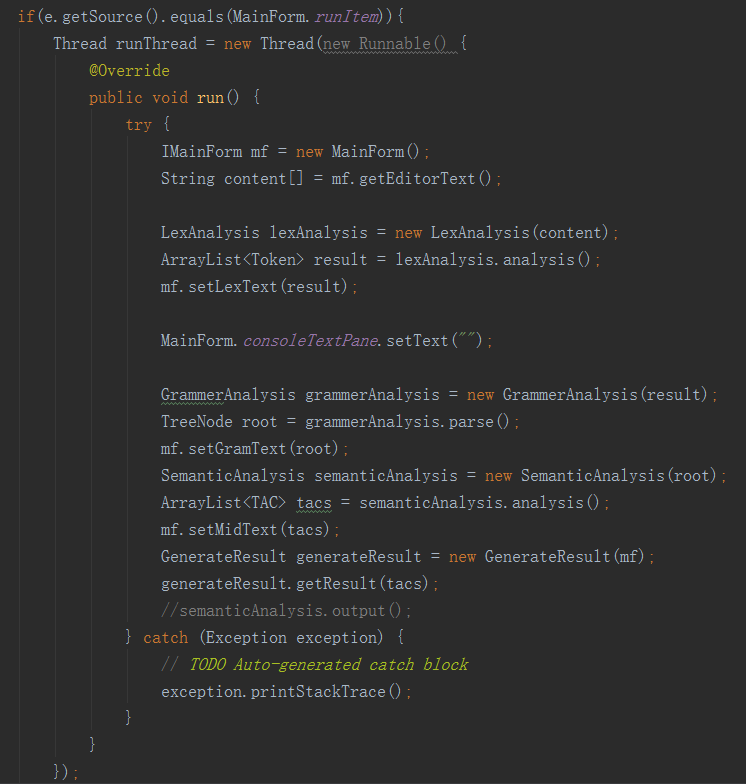


效果如下：点击撤销：点击恢复：

1. **运行功能**

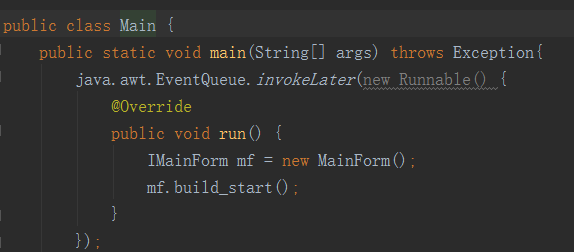
在MenuItemListener类中添加监控“运行”菜单项的功能：

点击“运行”菜单项时代码如下：

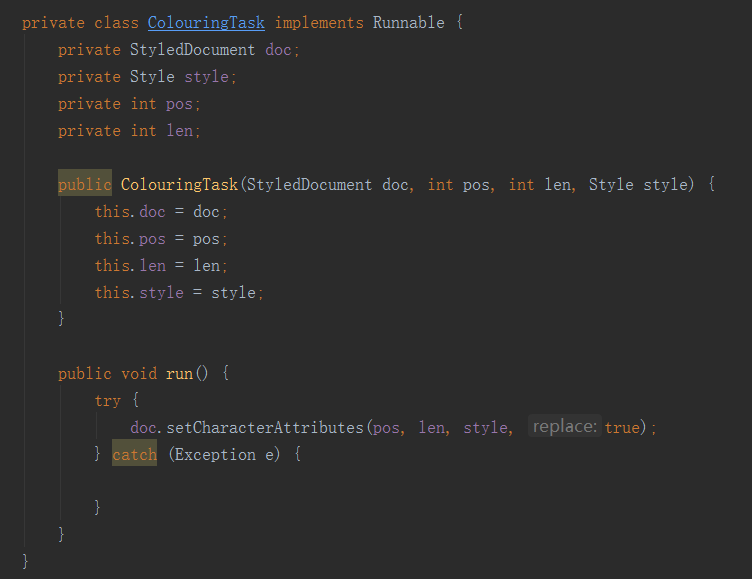


如图所示，新建了一个线程用来调用后台进行代码分析。

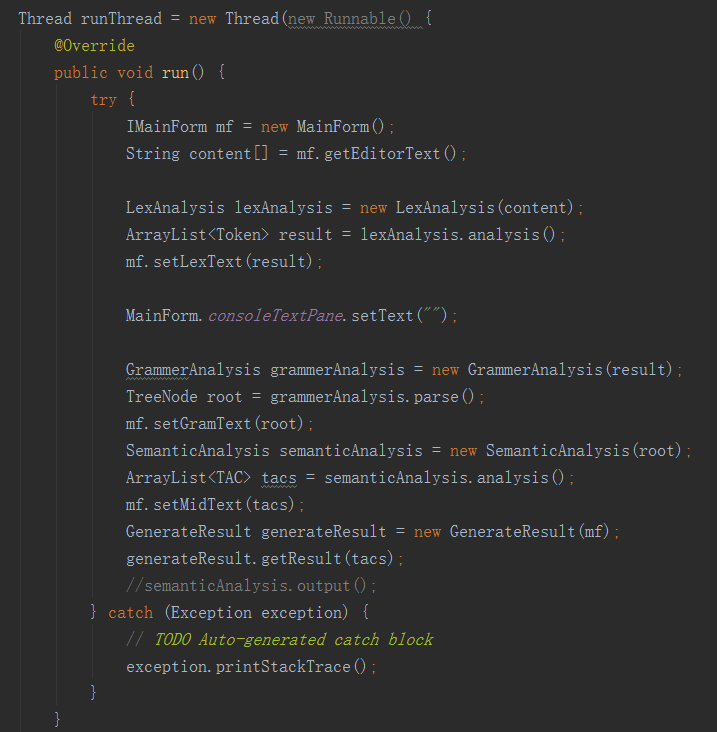
1. **使用多线程分离复杂任务**
2. 在整个项目的Main类中，初始新建一个线程用来启动前端GUI界面，代码如下：



1. 关键词着色时，判断需要着色后，新建一个线程执行关键的着色功能，代码如下：

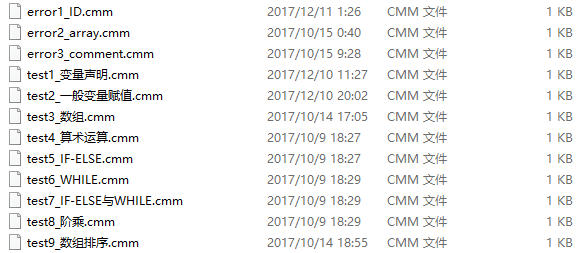


1. 点击“运行”菜单项后，新建一个线程进行代码分析，代码如下：



1. **调试**

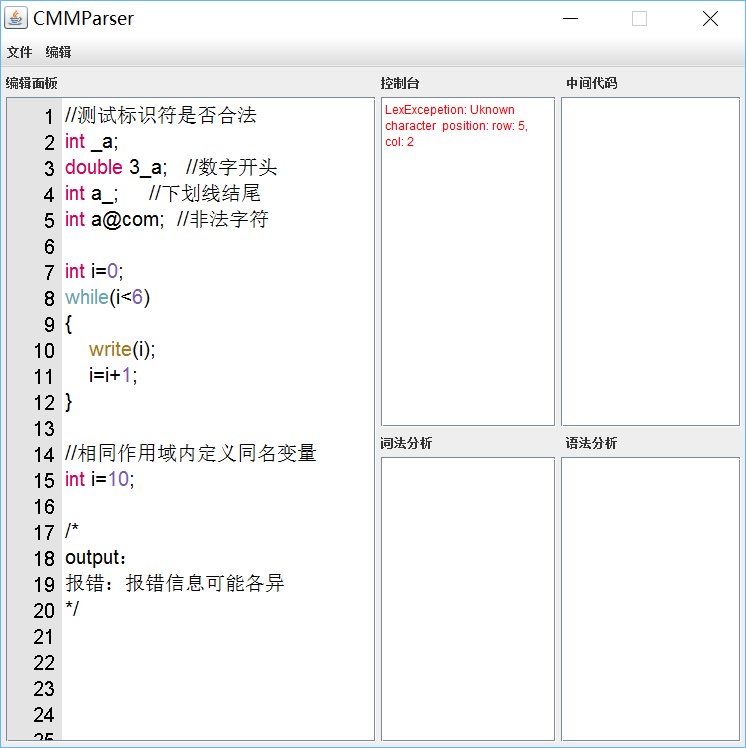
根据之前作业中助教提供的一些测试用例，来展示最终语义分析得到的结果。这些测试用例包括3个错误的用例和8个正确的用例。如下图



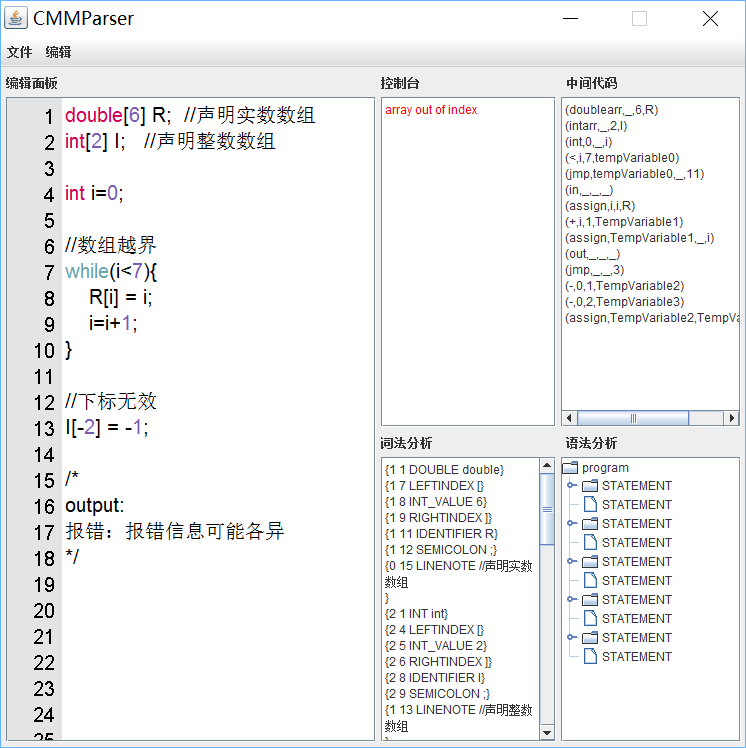
在GUI中选择路径打开对应文件，如下图

测试结果如下所示：

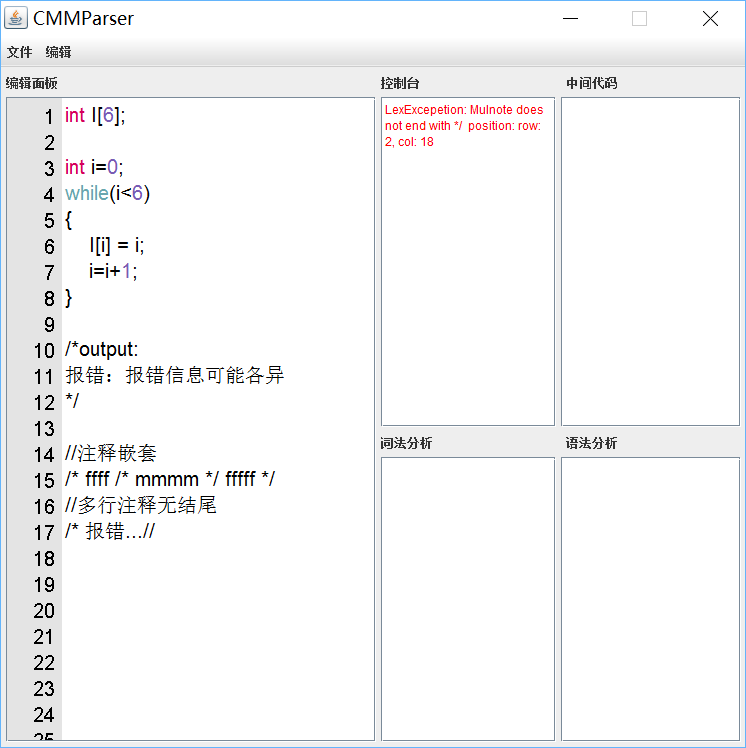
error1\_ID.cmm



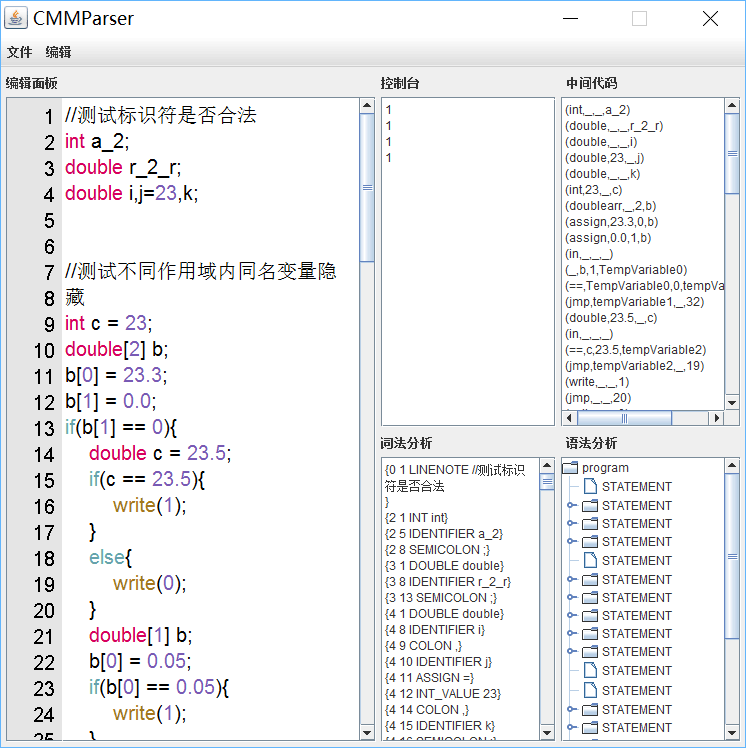
error2\_array.cmm



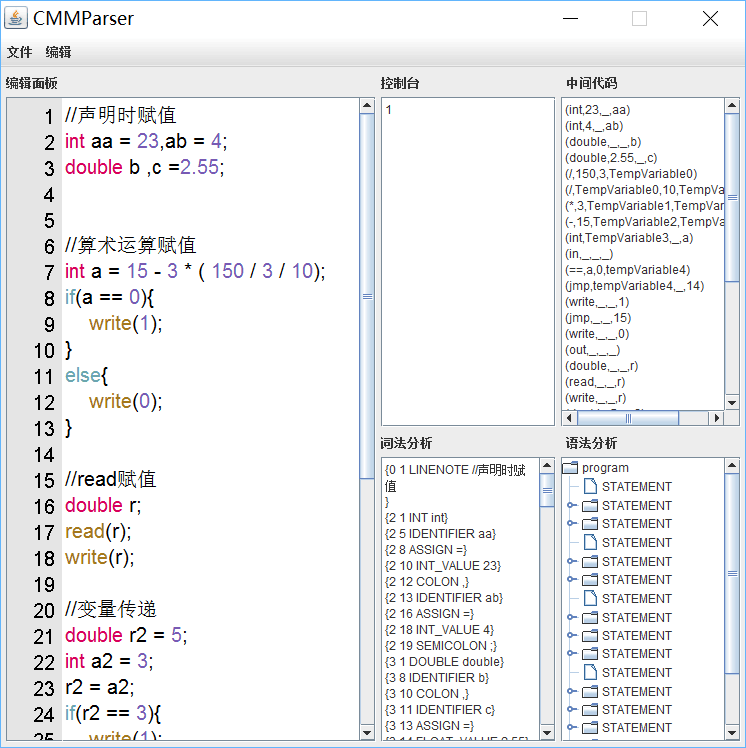
error3\_comment.cmm



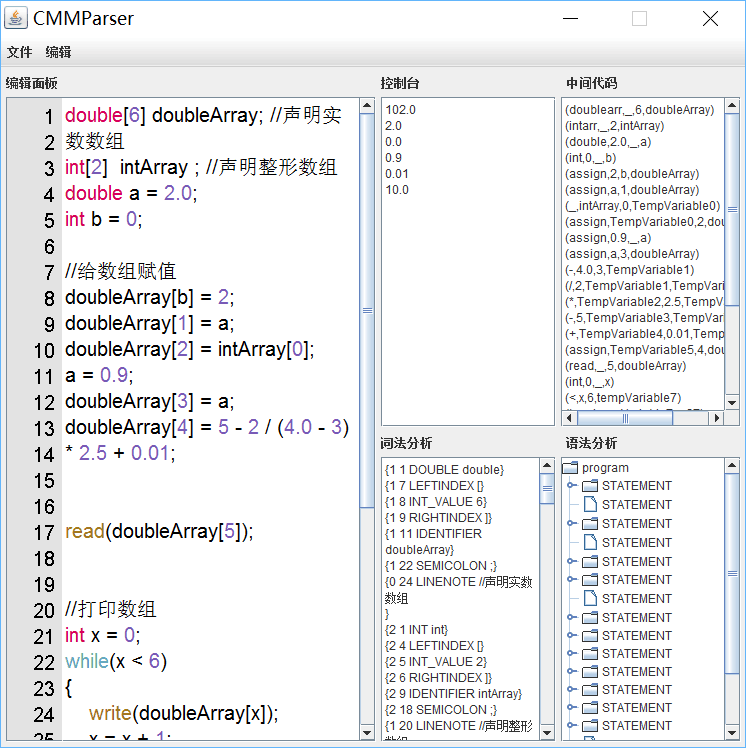
test1\_变量声明.cmm



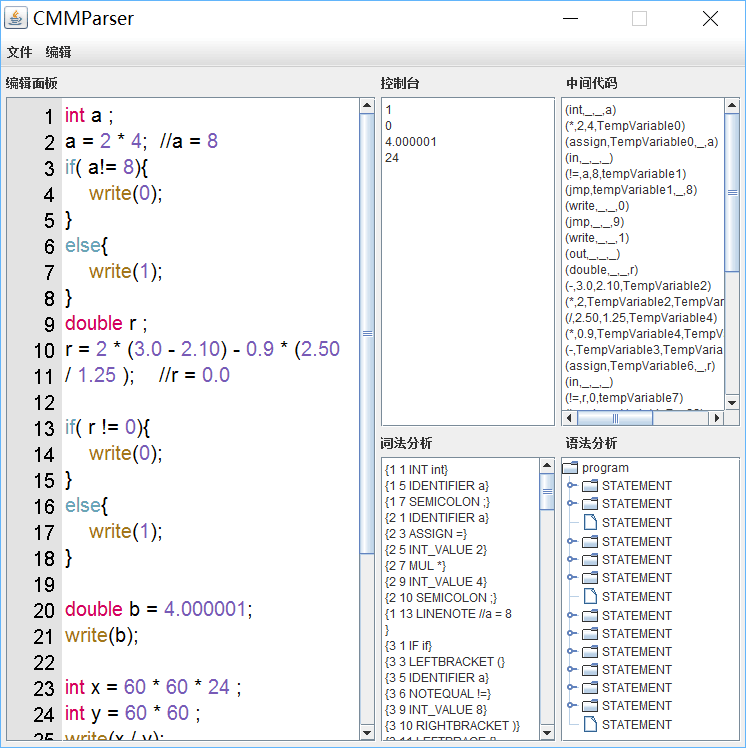
test2\_一般变量赋值.cmm



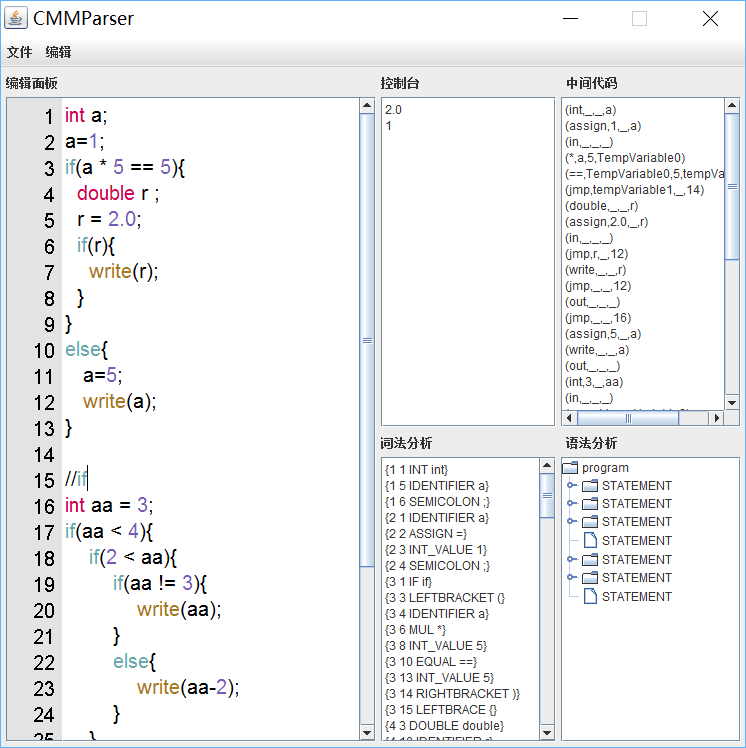
test3\_数组.cmm



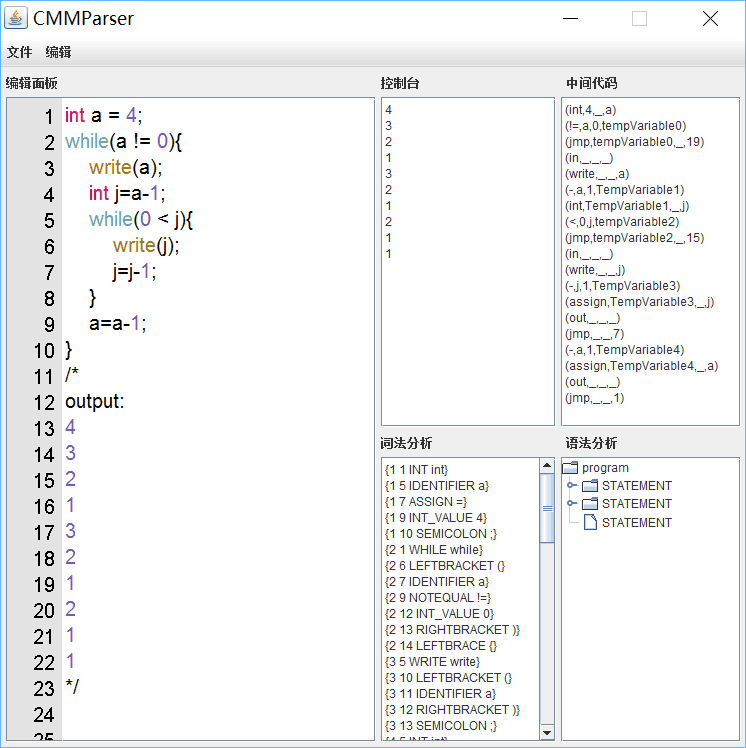
test4\_算术运算.cmm



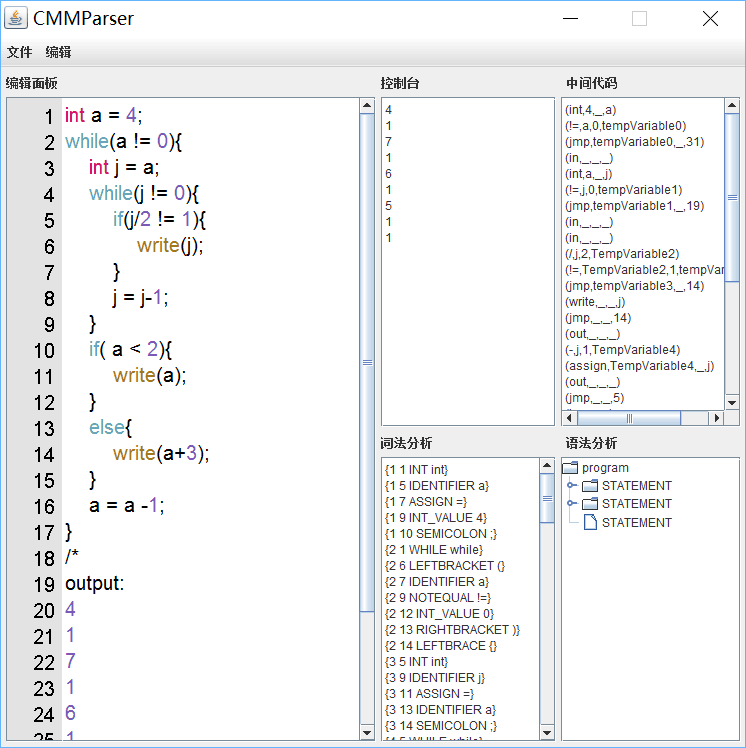
test5\_IF-ELSE.cmm



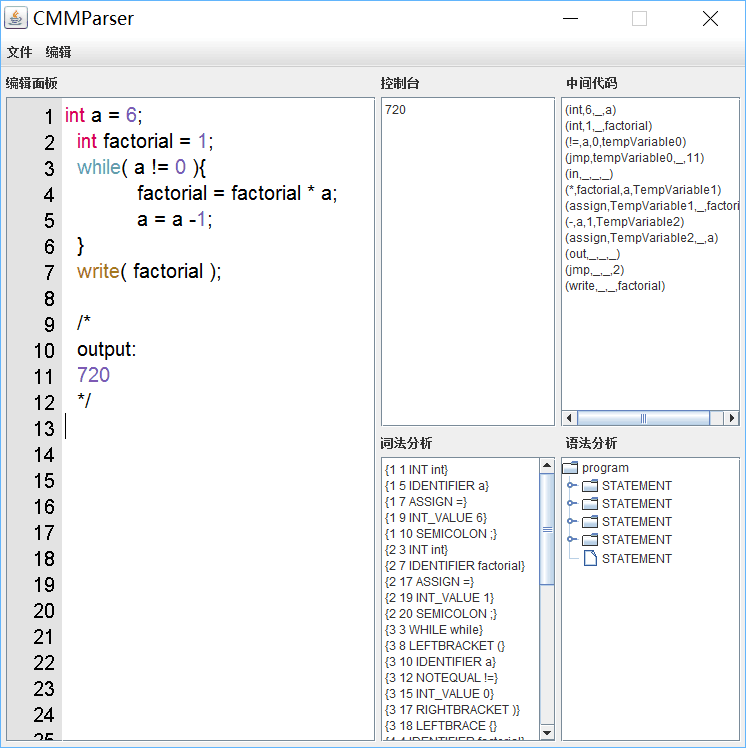
test6\_WHILE.cmm



test7\_IF-ELSE与WHILE.cmm



test8\_阶乘.cmm



test9\_数组排序.cmm

