CMM中间代码实现说明文档

张洹锦

2015302580241

**前言**

在第四次实验中，我主要负责CMM的中间代码实现部分，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。

**设计思想**

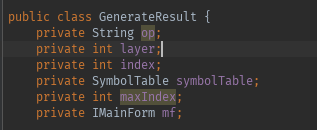
在完成中间代码的实现部分时，按照以下的设计思路来进行，首先确定与语义分析的联系，完成语义分析的同学提供了三个类，为Symbol，SymbolTable和TAC。其中Symbol类为定义一个新的symbol，也就是符号，包括了这个符号的类型，名字，值以及层数等等。SymbolTable主要定义了一个符号表，并编写了对于符号表的增删改等操作，最后是TAC，就是传输的中间代码。

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。

**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；

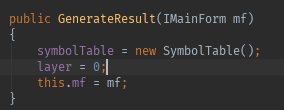
index为当前的数组下标，之所以定义为私有，是因为在存在判断条件时，需要跳转到指定的行数，此时需要一个私有变量来控制最外层的循环；

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；

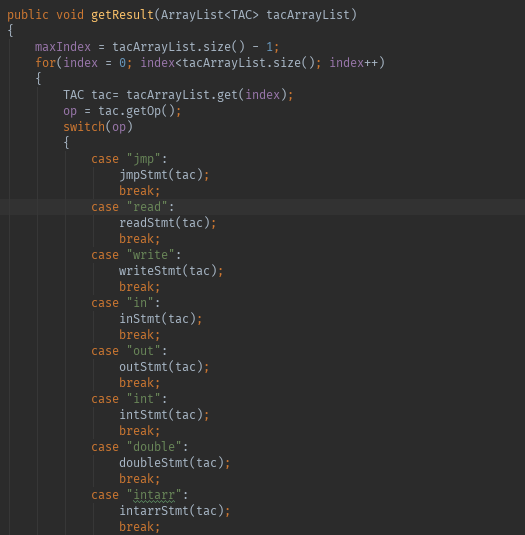
IMainForm是GUI的接口，用来将打印的结果展示在GUI界面中。

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。



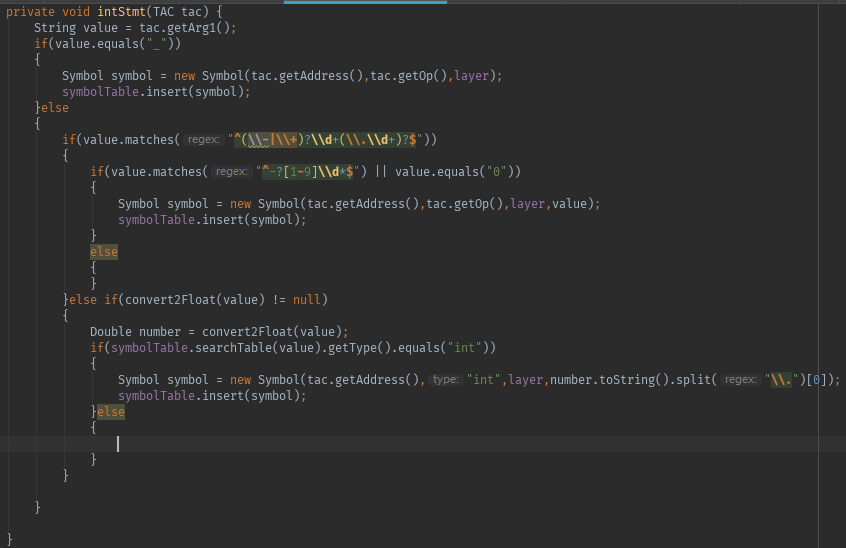
具体的实现函数介绍

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。

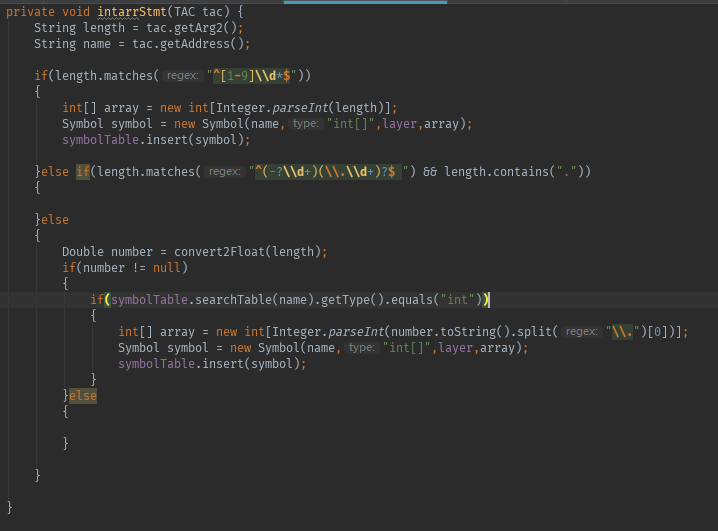
对于数组，cmm不支持在定义时就赋值的语法，因此只有前两种需要考虑赋了初值的情况。代码如下（int型，double型类似）



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）

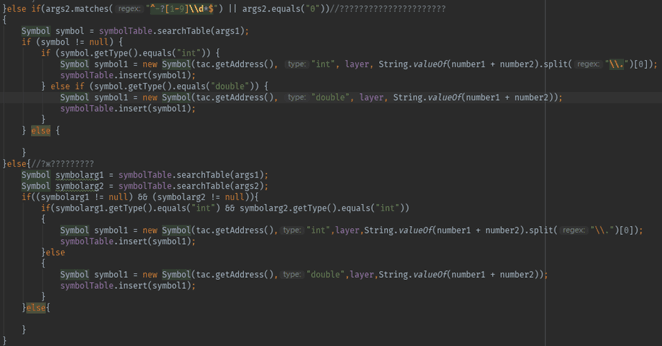
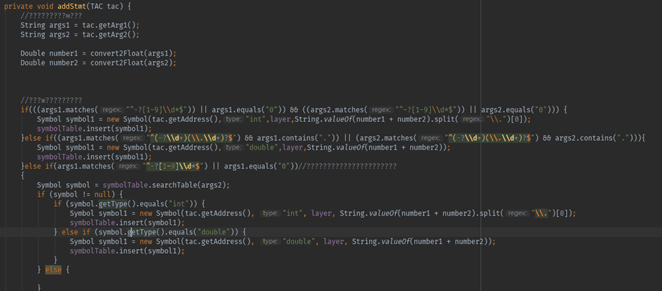


首先要判断长度是否为整数，或者其为变量，判断其类型是否为int，是的话就实例化symbol，数组的symbol构造函数不同，会为其新建一个名为array，长度为length的数组。

1. 加减乘除部分

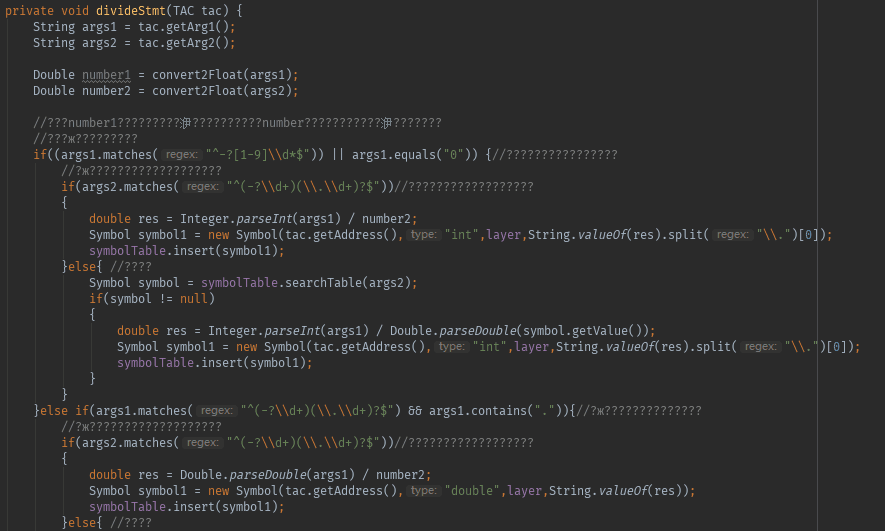
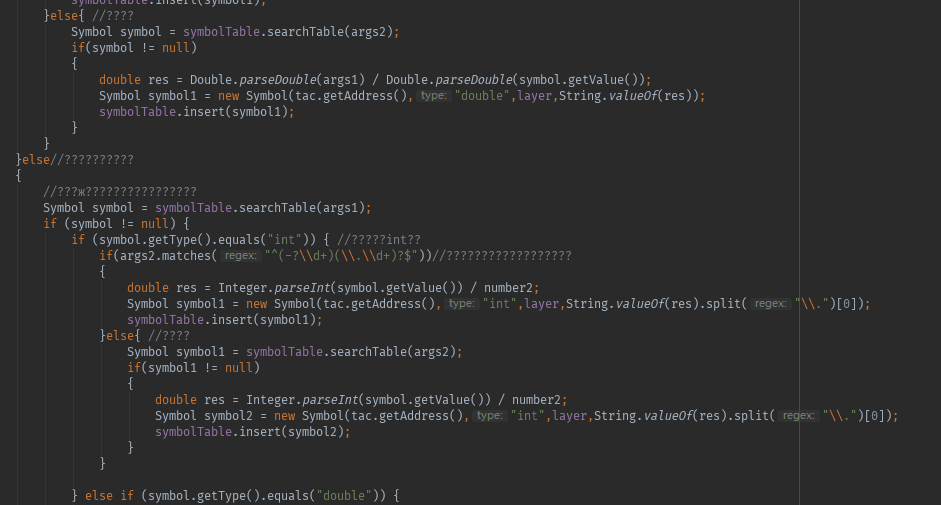
包括了addStmt()，subStmt()，mulStmt()，divideStmt()四个函数，其中比较类似的是add，sub和mul，因为他们是必须两个参数都是int，最终的结果才是int型，而div是如果被除数是int，结果就是int，被除数是double，结果就是double，与第二个参数无关。

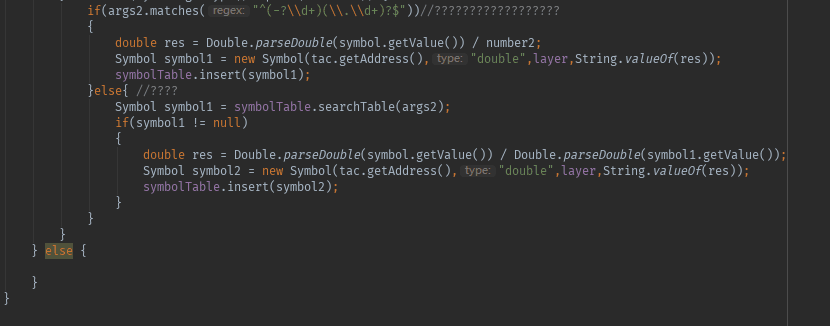
首先看addStmt(),sub和mul类似。



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()。

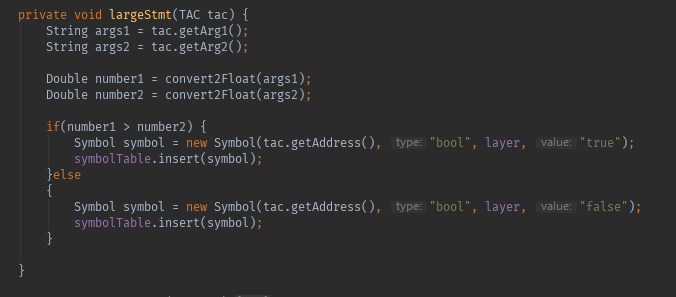
 



逻辑如下，首先判断第一个参数是否是整数，如果是，无论第二个参数是数字还是变量，将值提取出来，存为int型；再判断第一个是否为浮点数，同上，最后是第一个参数是变量，首先判断其类型，接下来同上。

1. 条件判断部分

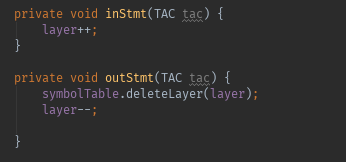
条件判断部分包括了大于，小于，等于，不等于，大于等于和小于等于这几种情况，方法类似，以largeStmt()为例子讲解。



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

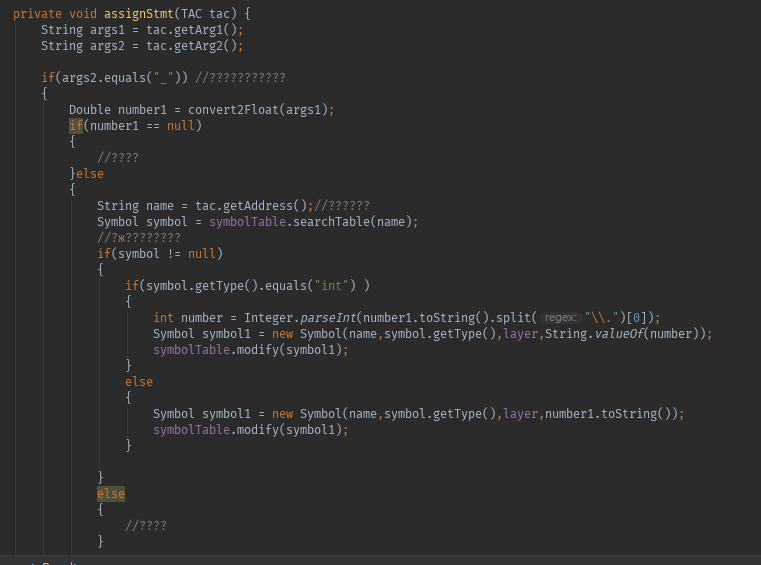
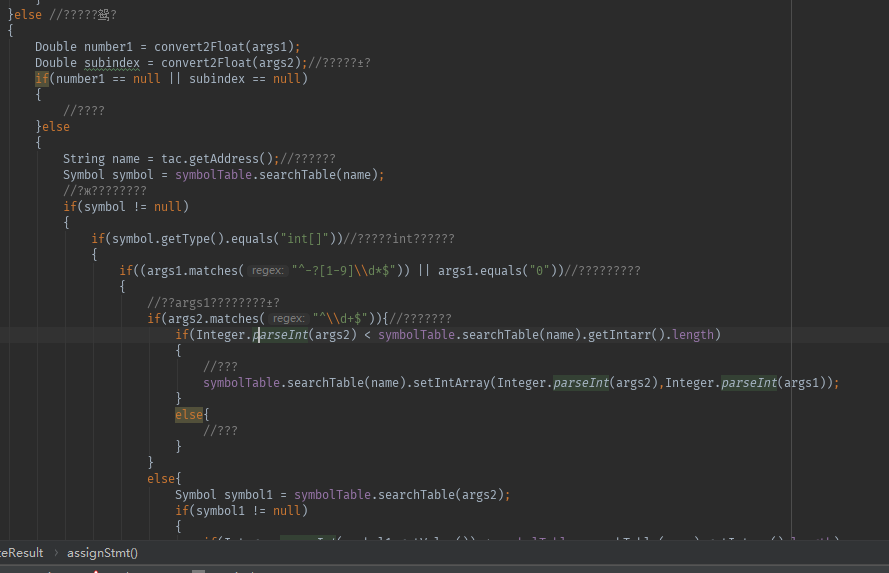
层次修改包括inStmt()，outStmt()。代码如下

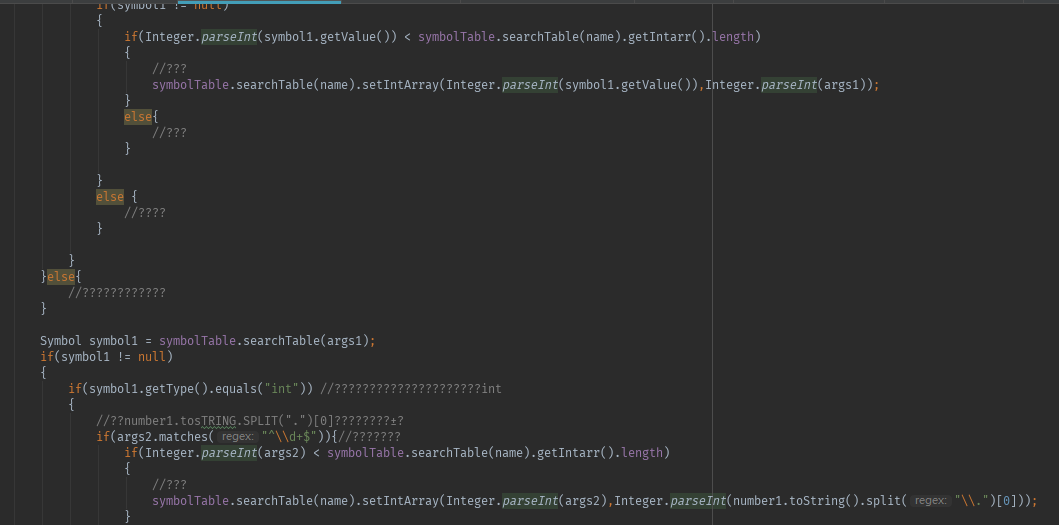
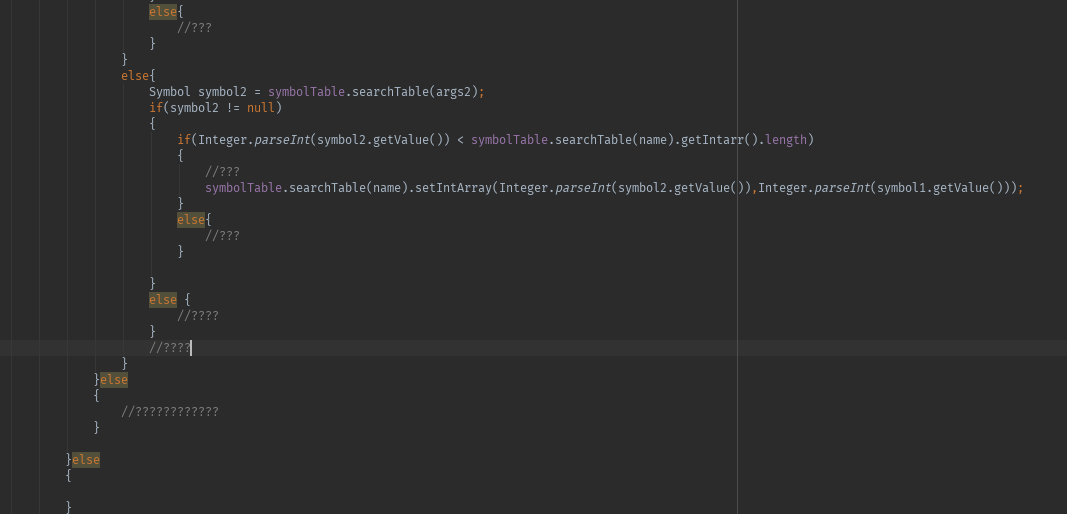


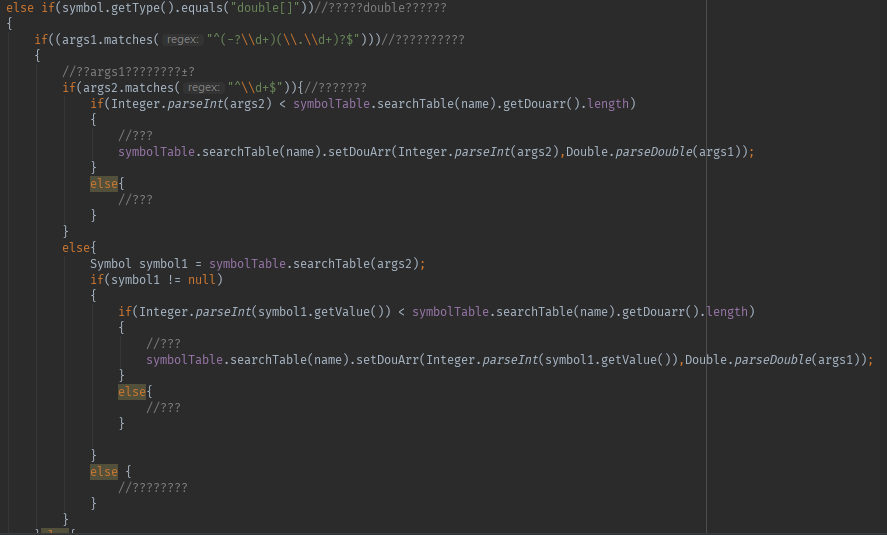
当遇到｛时，调用inStmt，将layer++。遇到 ｝时，将当前layer的变量全部删除，再把layer--。

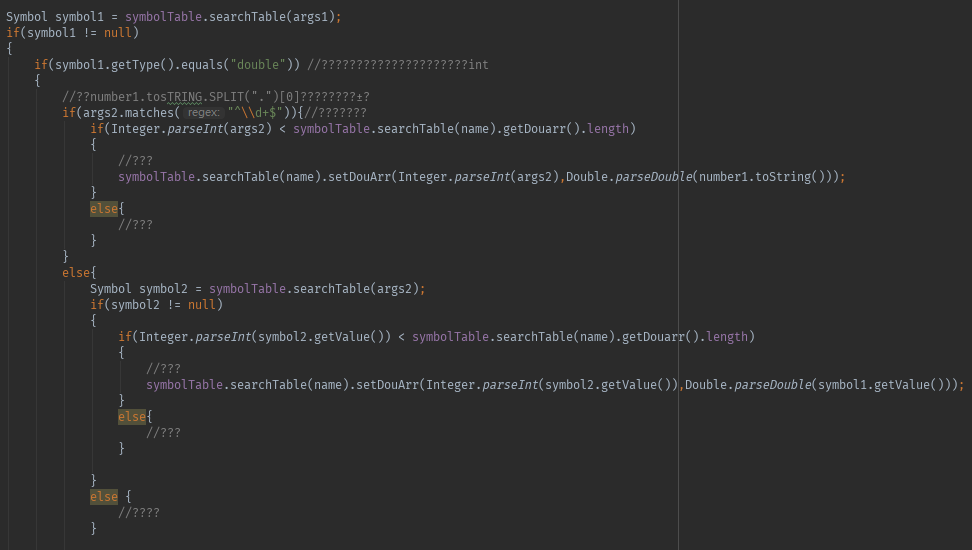
1. 赋值部分

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

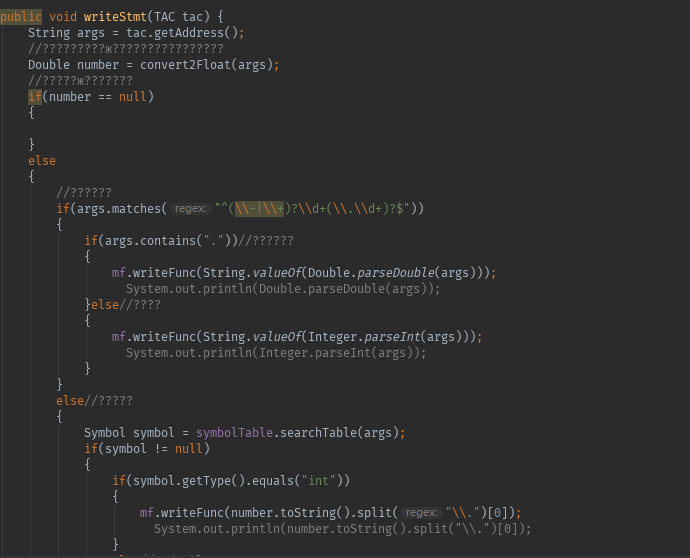




由于情况过多，且为了给其他的方法减小工作量，因此assign函数进行了大量的判断，大致逻辑如下，首先判断是否是给数组赋值，如果不是，判断value是数字还是变量，并且类型要一致；如果是，先判断是int数组还是double数组，接下来判断value是数字还是变量，数字和变量必须与数组的类型一致。

1. 读写部分

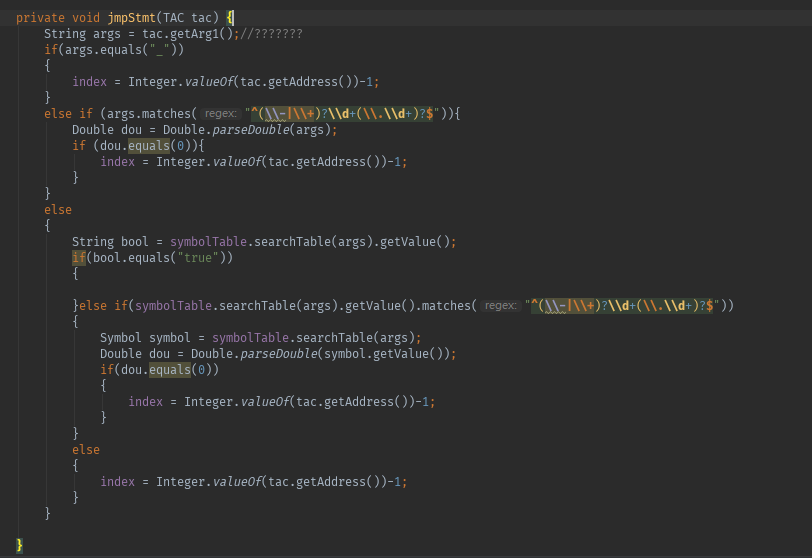
包括writeStmt()和readStmt()函数，代码如下（writeStmt为例）



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

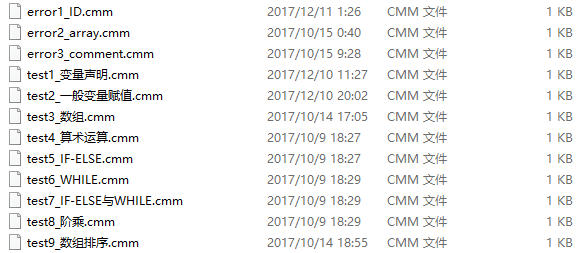
只有jmpStmt函数来实现跳转的功能。代码如下



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。

**测试用例**

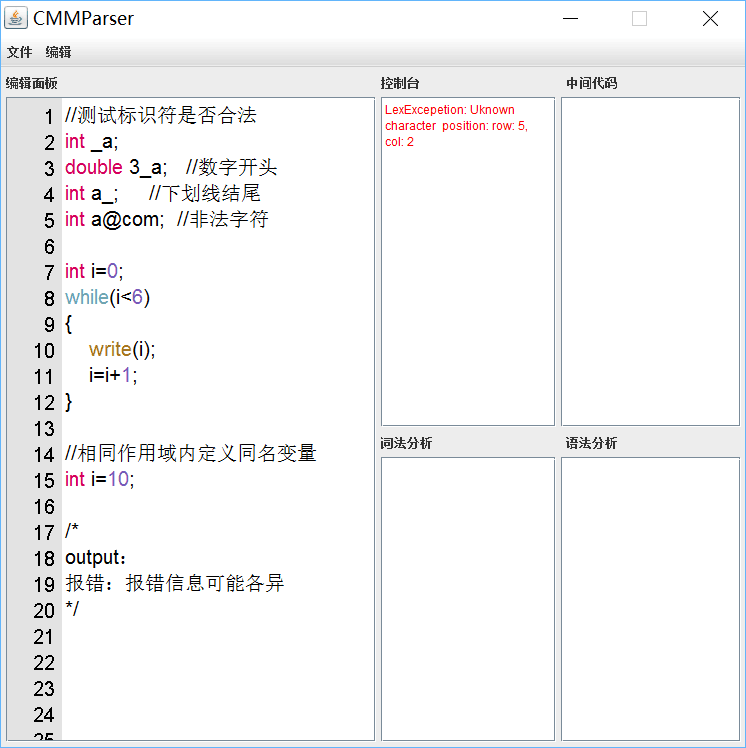
根据之前作业中助教提供的一些测试用例，来展示最终语义分析得到的结果。这些测试用例包括1个错误的用例和8个正确的用例。如下图



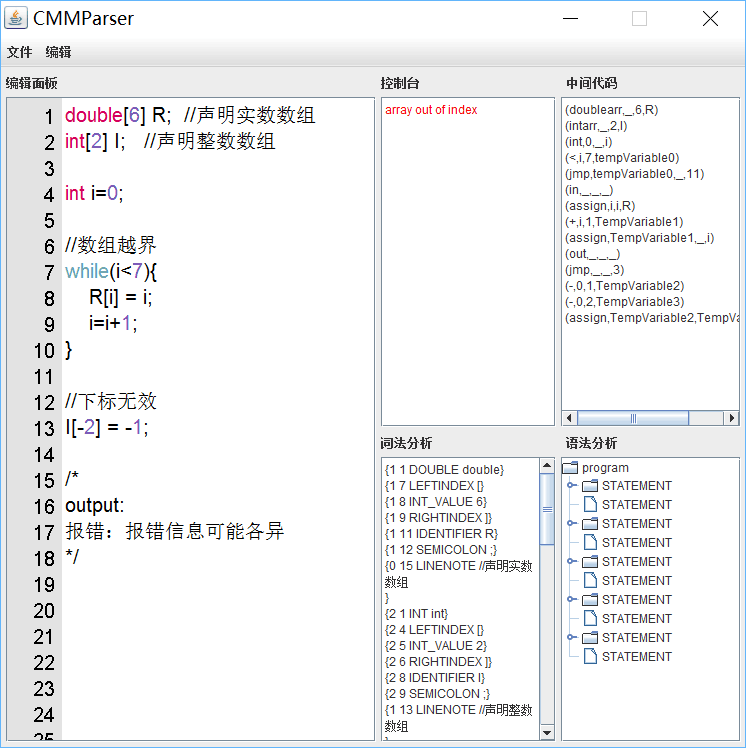
在GUI中选择路径打开对应文件，如下图

测试结果如下所示：

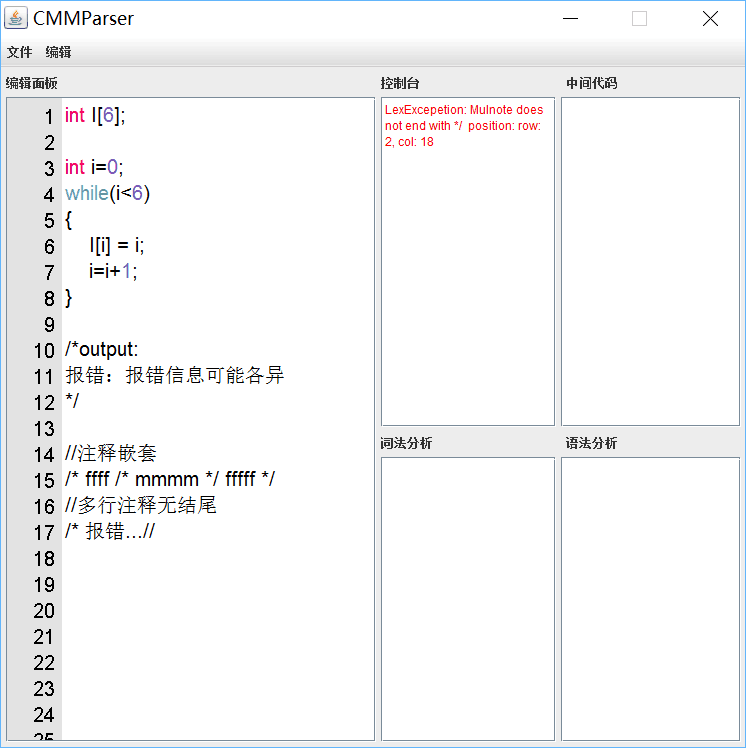
1. error1\_ID.cmm



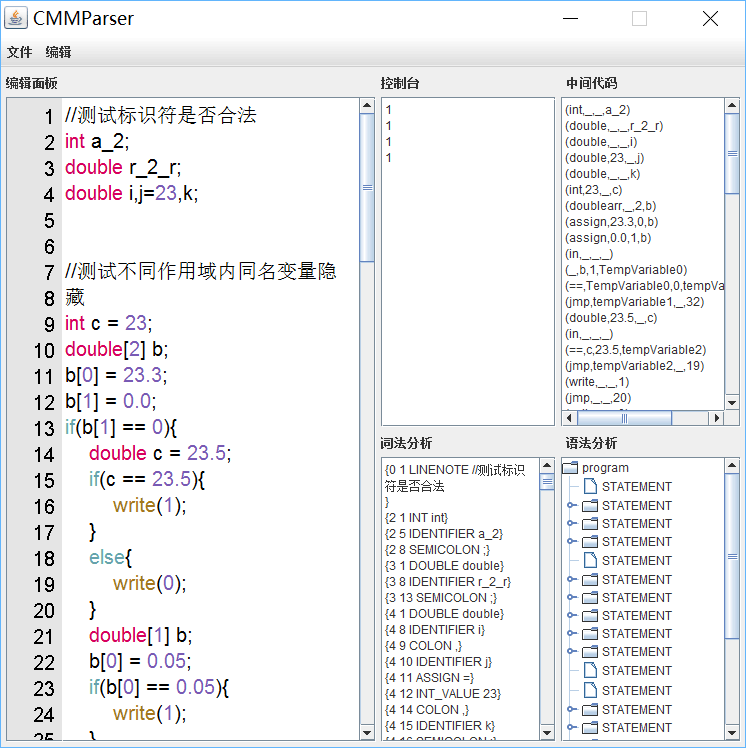
1. error2\_array.cmm



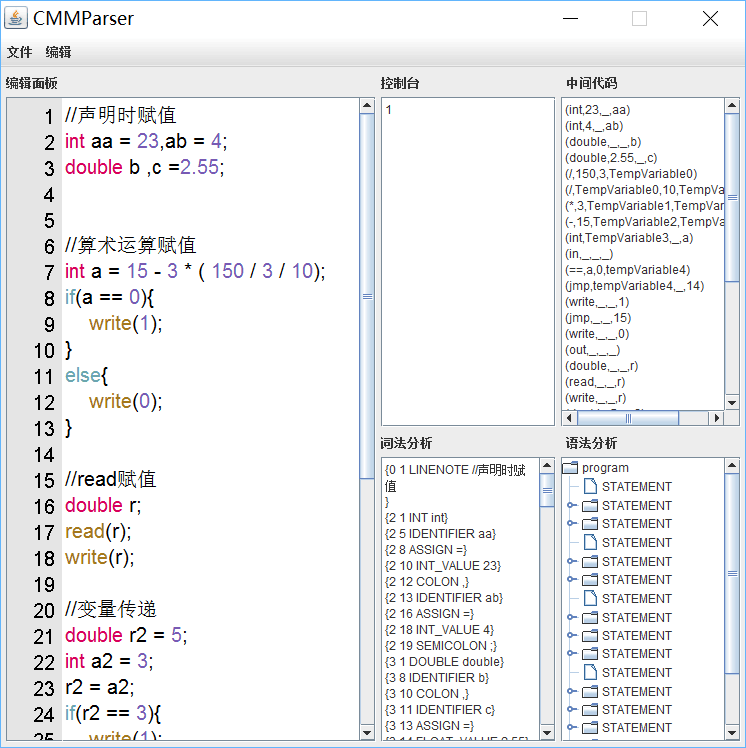
1. error3\_comment.cmm



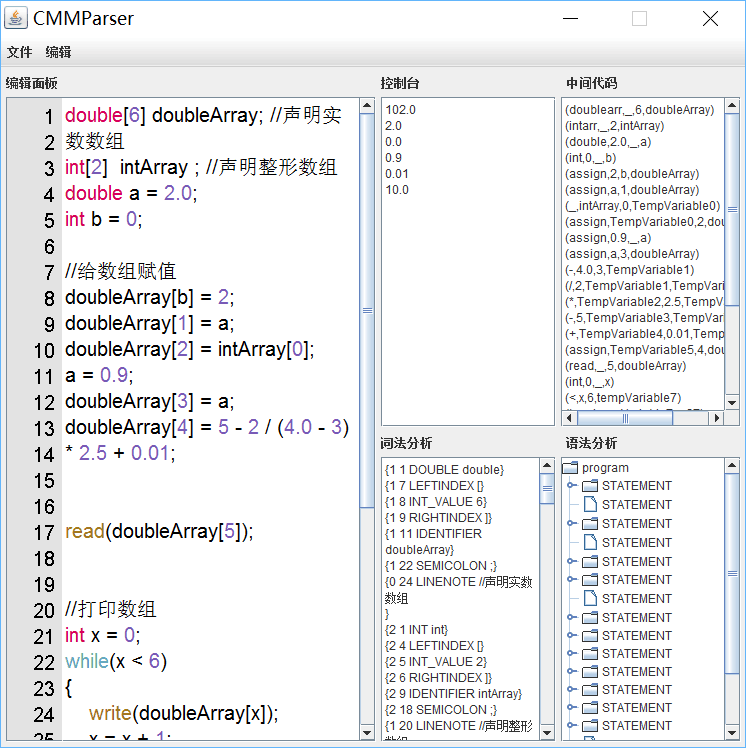
1. test1\_变量声明.cmm



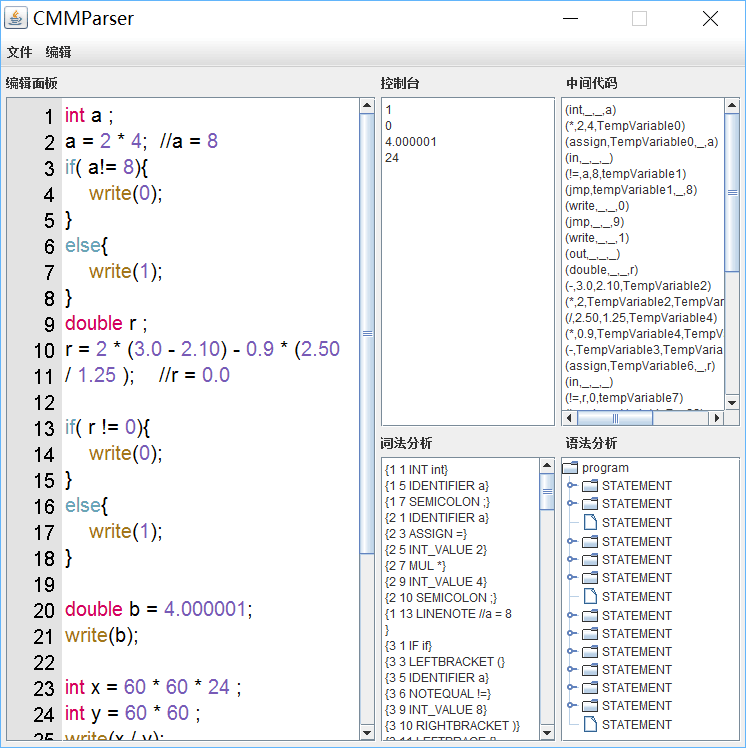
1. test2\_一般变量赋值.cmm



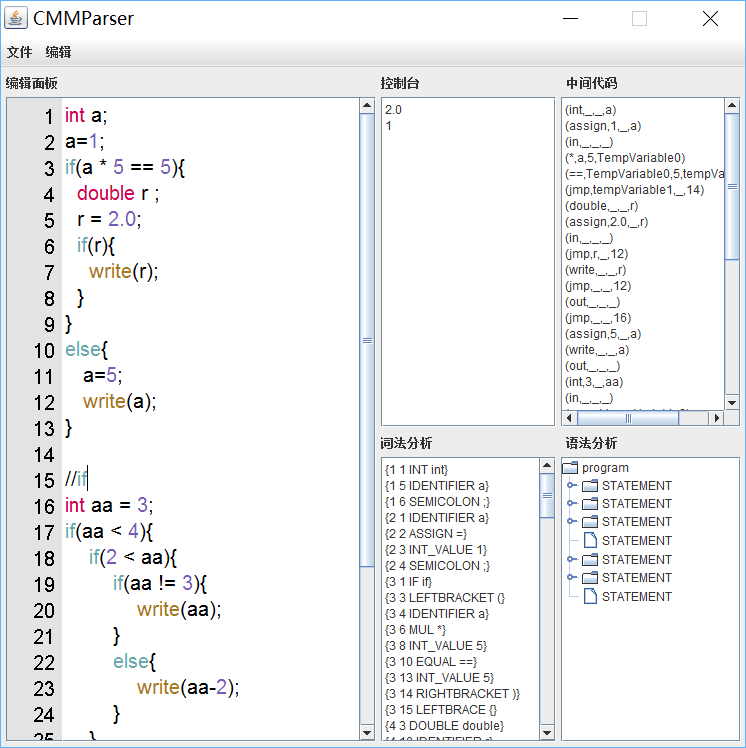
1. test3\_数组.cmm



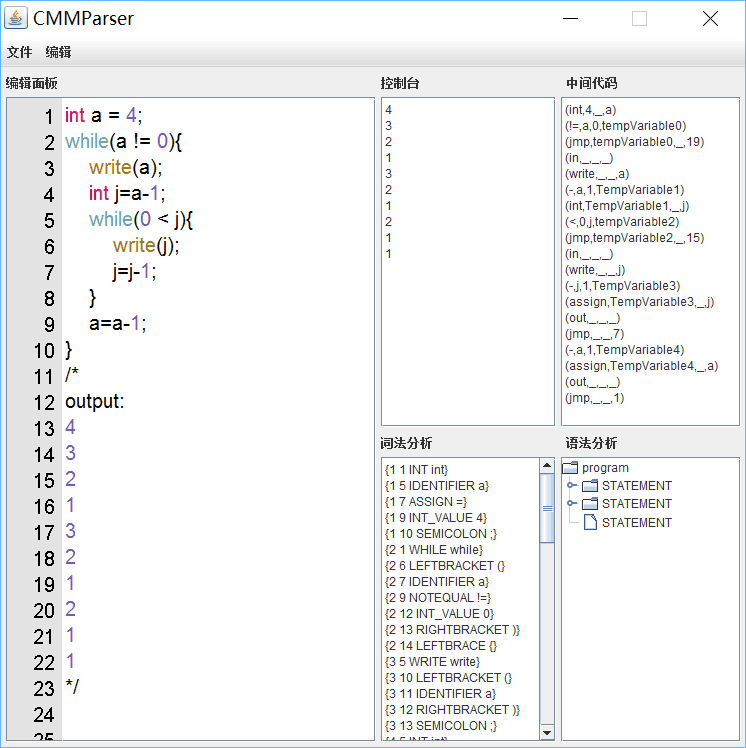
1. test4\_算术运算.cmm



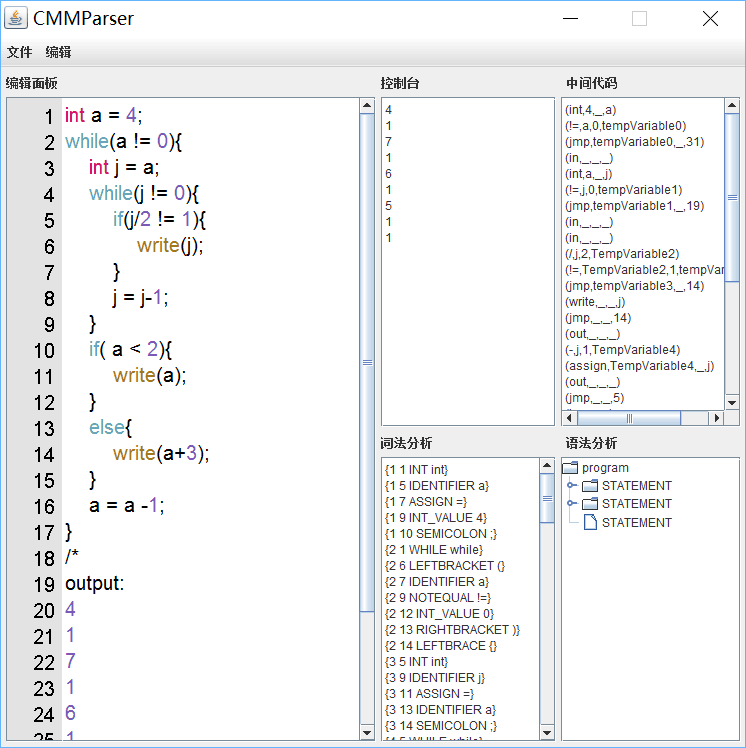
1. test5\_IF-ELSE.cmm



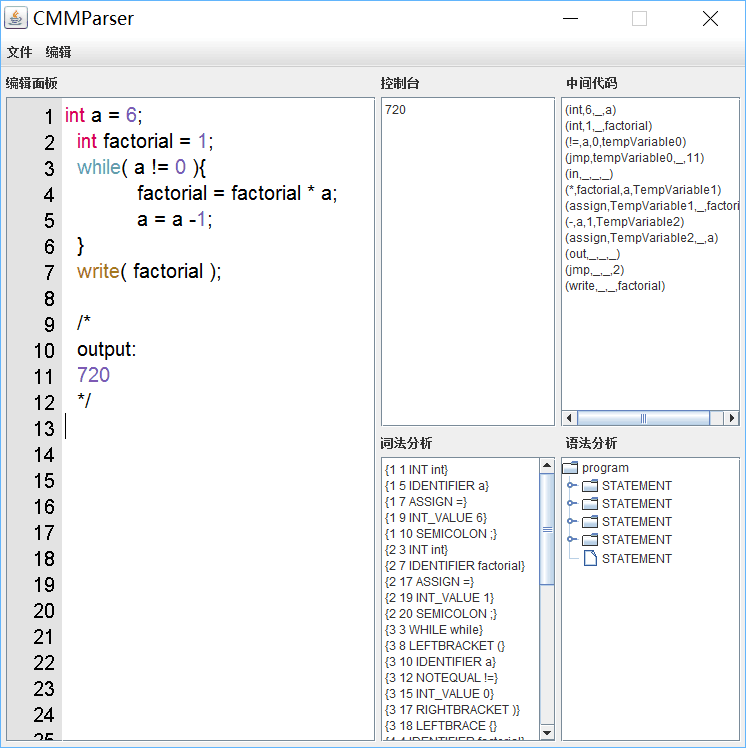
1. test6\_WHILE.cmm



1. test7\_IF-ELSE与WHILE.cmm



1. test8\_阶乘.cmm



1. test9\_数组排序.cmm

