# Toureter Interpreter

——**Project Document**

**Written by Xiaoran Zhan**

# Produced by Toureter Group

**All Rights Reserved, Copy Reserved**

**CMM解释器项目文档**

### 一、背景

完成小组的基本信息：

|  |  |  |
| --- | --- | --- |
| **姓名** | **学号** | **分工内容** |
| 贾道远 | 2010302580176 | 小组组长  后台逻辑开发 |
| 徐亚渤 | 2010302580285 | 后台逻辑开发 |
| 徐洁斌 | 2010302580165 | 前台组组长  前台设计和实现 |
| 詹翛然 | 2010302580224 | 前台美工、逻辑 |

### 二、文法描述

#### 2.1 词法分析的对象是CMM语言的源程序，源程序的叙述如下：

CMM语言为C语言的一个子集：

* + 语言结构：顺序结构（赋值语句、输入、输出）、选择语句（if-else）、循环结构（while）。这些语句结构和C语言的结构一样，允许嵌套。
  + 表达式局限于关系表达式和算术表达式，运算的优先级为：算术运算、关系运算，并服从左结合规则。
  + 算术表达式包括整数和实数上的运算、变量以及“（）”、“\*”、“+”、“-”、“/”，运算符的优先级顺序为：“（）”大于“\*”和“/”大于“+”和“-”。
  + 关系运算符包括：“<”、“==”、“<>”。
  + 一条语句以“；”结束；程序由一条语句或者由“{”和“}”嵌套表达的复合语句。
  + 支持多行注释（使用“/\*”和“ \*/ ”）
  + 支持数组运算，数组的下标必须是正整数，使用“[”和“]”表示数组下标。
  + 变量的使用之前需要先声明，声明的方式和C语言一样。

|  |  |  |
| --- | --- | --- |
| 保留字 | 特殊符号 | 其他 |
| if | + | 十进制的整数与实数 |
| else | - |
| while | \* |
| read | / | 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） |
| write | = |
| int | < |
| real | == |  |
| struct | <> |  |
| class | ( |  |
|  | ) |  |
|  | ; |  |
|  | { |  |
|  | } |  |
|  | /\* |  |
|  | \*/ |  |
|  | [ |  |
|  | ] |  |
|  | /\* |  |
|  | \*/ |  |
|  | && |  |
|  | || |  |
|  | ! |  |

经过词法分析，得到的符号类型简介如下：

|  |  |
| --- | --- |
| 符号类型 | 符号 |
| Keyword | if，else，while，int，real，read，write |
| Sign | +，-，\*，/，/\*，\*/，=，==，<，<>，[，]，(，)，{，}，;，,， |
| int | 十进制整数，由数字串组成 |
| real | 十进制实数，由数字和小数点组成 |
| Identifier | 由字母开头，由数字、字母和\_组成，并且不以下划线结尾的字符串 |

#### 2.2 程序

程序由class和struct组成

Program ->( class|struct)\*

* + 1. **类**

CLASS-> class identifier { (DECLARE\_STMT |FUNCTION)\* } //类定义由ID和声明语句,函数组合而成

**2.2.2 结构体**

STRUCT-> struct identifier { (DECLARE\_STMT)\* } //结构体由ID和声明语句组成

* + 1. **函数**

FUNCTION-> TYPE identifier (( DECLARE\_STMT)\*) STMT\_SEQUENCE

* + 1. **语句序列**

STMT\_SEQUENCE -> { STATEMENT}\* // 语句序列由多条语句组成

* + 1. **语句**

STATEMENT-> ASSGIN\_STMT|

WRITE\_STMT |

READ\_STMT |

IF\_STMT |

WHILE\_STMT |

DECLARE\_STMT|

CALL\_STMT

* + 1. **循环语句**

WHILE\_STMT -> while ( EXPR\* ) ({ STMT\_SEQUENCE})|(STATEMENT) //while语句

* + 1. **赋值语句**

ASSGIN\_STMT -> identifier = EXPR

identifier -> identifier (“[“ EXPR“]” )\*

* + 1. **条件语句**

If\_STMT -> if ( EXPR\* ) ({ STMT\_SEQUENCE})|(STMT) (else ({STMT\_SEQUENCE})|(STATEMENT))?

* + 1. **读语句**

READ\_STMT -> read identifier; //读语句

* + 1. **写语句**

WRITE\_STMT -> write EXPR; //写语句

* + 1. **表达式语句**

EXPR->LOGIC\_AND\_EXPR

LOGIC\_AND\_EXPR -> LOGIC\_OR\_EXPR (&& LOGIC\_OR\_EPXR)?

LOGIC\_OR\_EXPR ->LOGIC\_NOT\_EXPR (|| LOGIC\_NOT\_EXPR)?

LOGIC\_NOT\_EXPR ->(!)? LOGIC\_ELEMENT\_EXPR

LOGIC\_ELEMENT\_EXPR-> true|false|EQUALITY\_EXPR

EQUALITY\_EXPR -> ARITHM\_EXPR (<|<=|>|>=|==|<> ARITHM\_EXPR)?

ARTHM\_EXPR ->TERM\_EXPR (+|- TERM\_EXPR)?

TERM\_EXPR ->UNARY\_EXPR (\*|/|% UNARY\_EXPR)?

UNARY\_EXPR ->(-)? ELMENT

ELMENT -> const|CALL\_STMT|identifier|(EXPR)

详见 参照文档《语法分析设计说明文档.pdf》

### 三、系统分析和设计

包括系统的概要设计、详细设计和核心算法说明及分析，系统所采用的开

发工具等。

* 1. **系统设计**

整个解释器大体分为三个模块：词法分析模块，语法分析模块和语义分析及解释执行模块。

词法分析模块定义了空白符、单行及多行注释符号、CMM语言保留字、整数及实数、标识符、分隔符以及运算符。

语法分析实现了对CMM语言基本语法规则的定义，并生成抽象语法树。

语义分析及解释执行模块实现了程序的翻译及执行，在语法分析生成的抽象语法树的基础上对节点逐层分析并作相应处理。

* 1. **核心算法分析及说明**

主要的代码编写工作集中在对语义的分析及最终的解释执行上，另外也包括了部分生成代码编写以及前后台连接。

语义分析及解释执行模块的代码编写主要思想是基于对语法分析生成的抽象语法树的节点解释，针对于不同的节点编写不同规则进行解释，父节点和子节点逐级调用，最终完成解释执行。

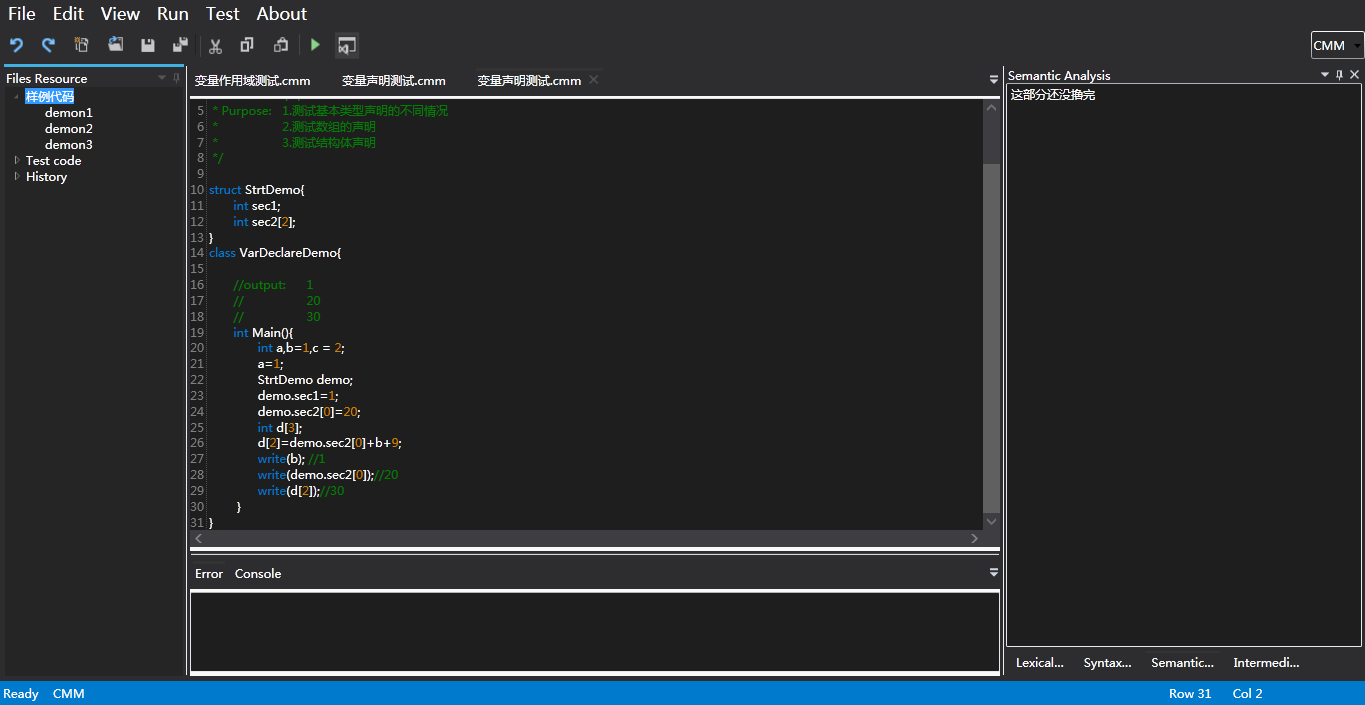
* 1. **系统采用的开发工具**

VS2012 ,Notpad

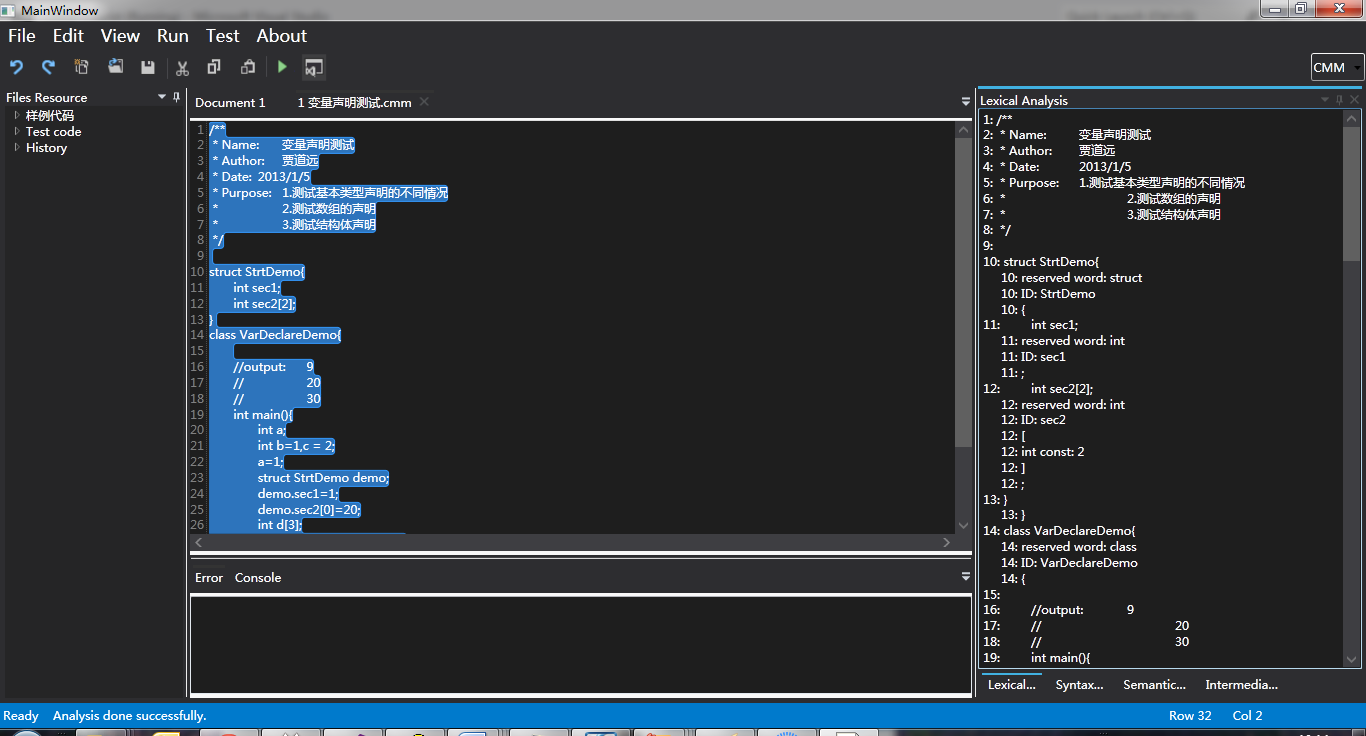
### 四、测试用例和测试结果

#### 4.1 无参数类展示

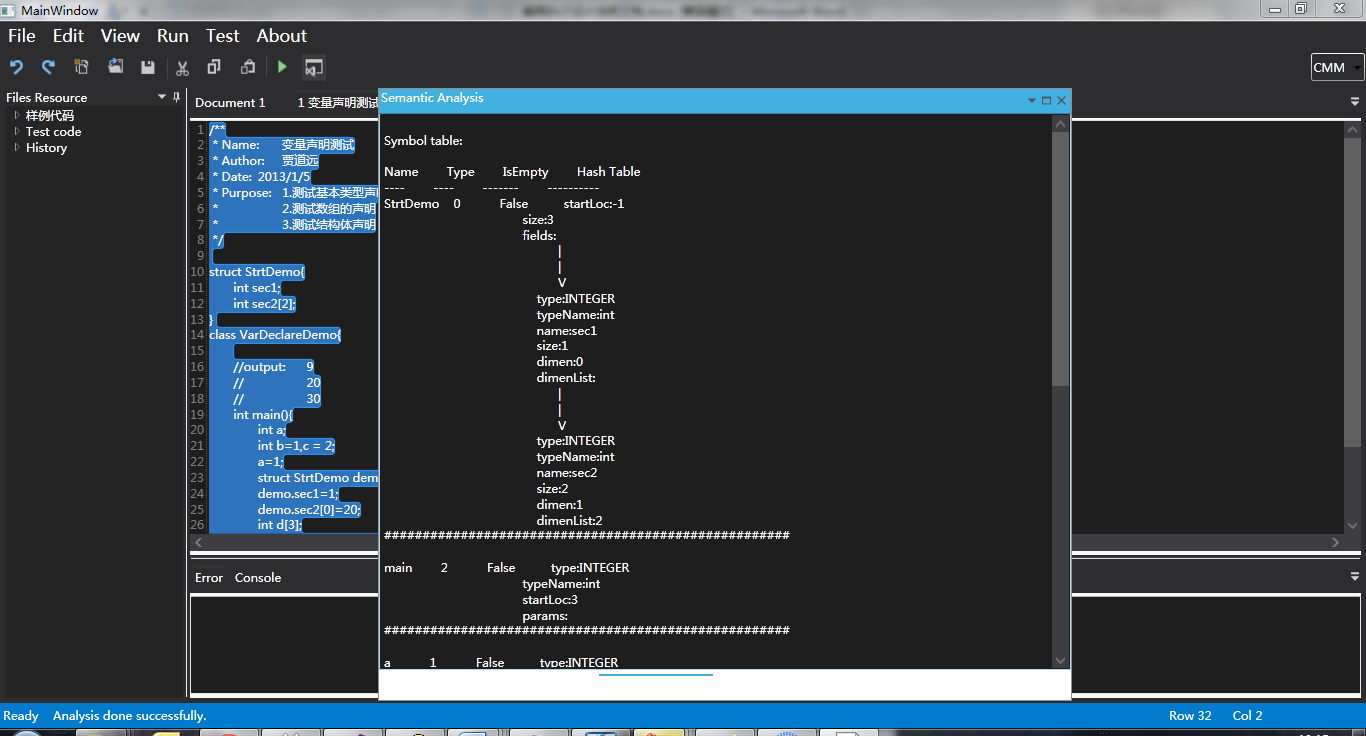
输入以下内容：



得到如下结果，完成分析过程：

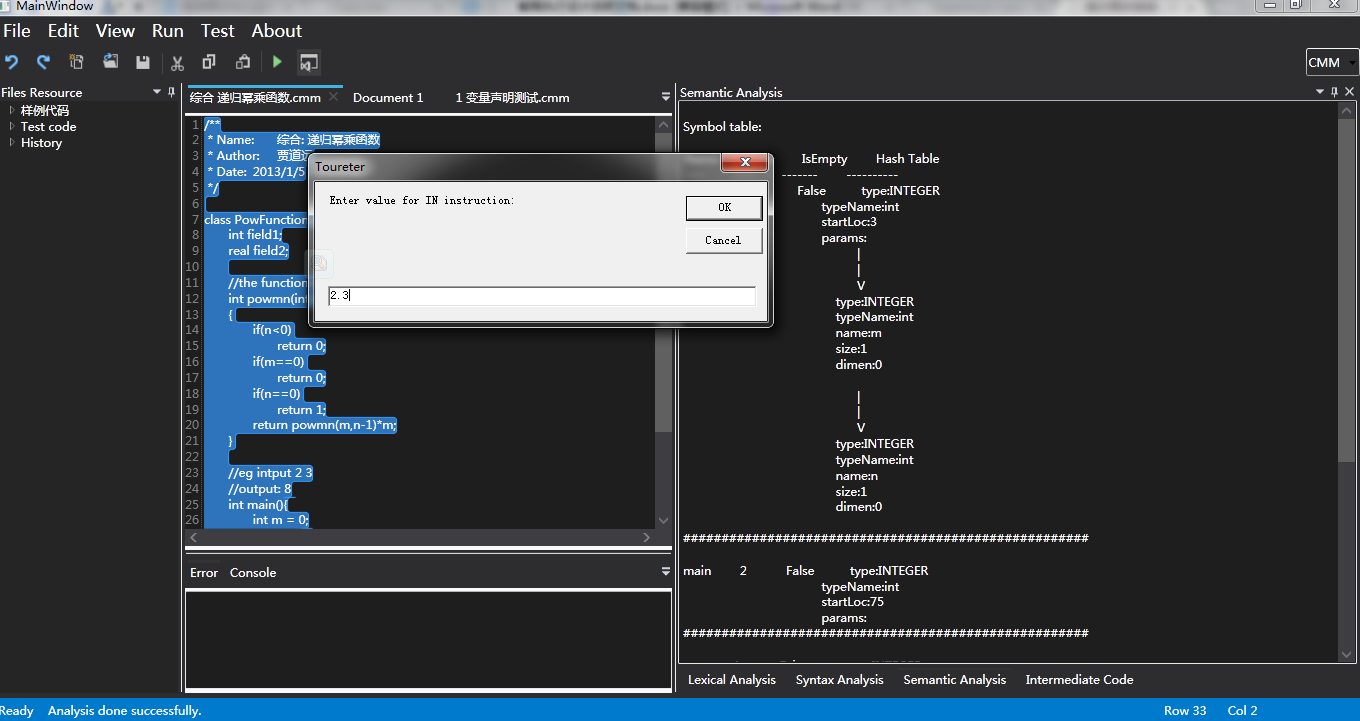


所得到的符号表是：

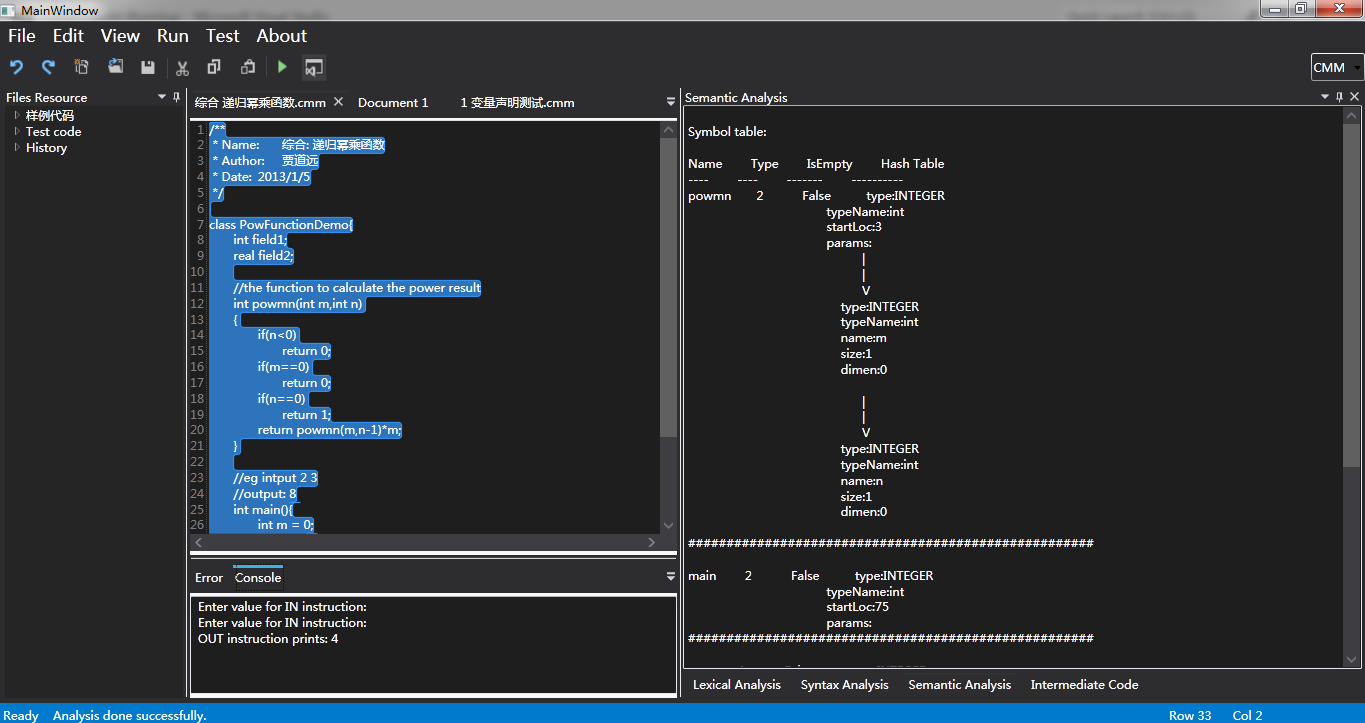


#### 4.2 有参数类展示

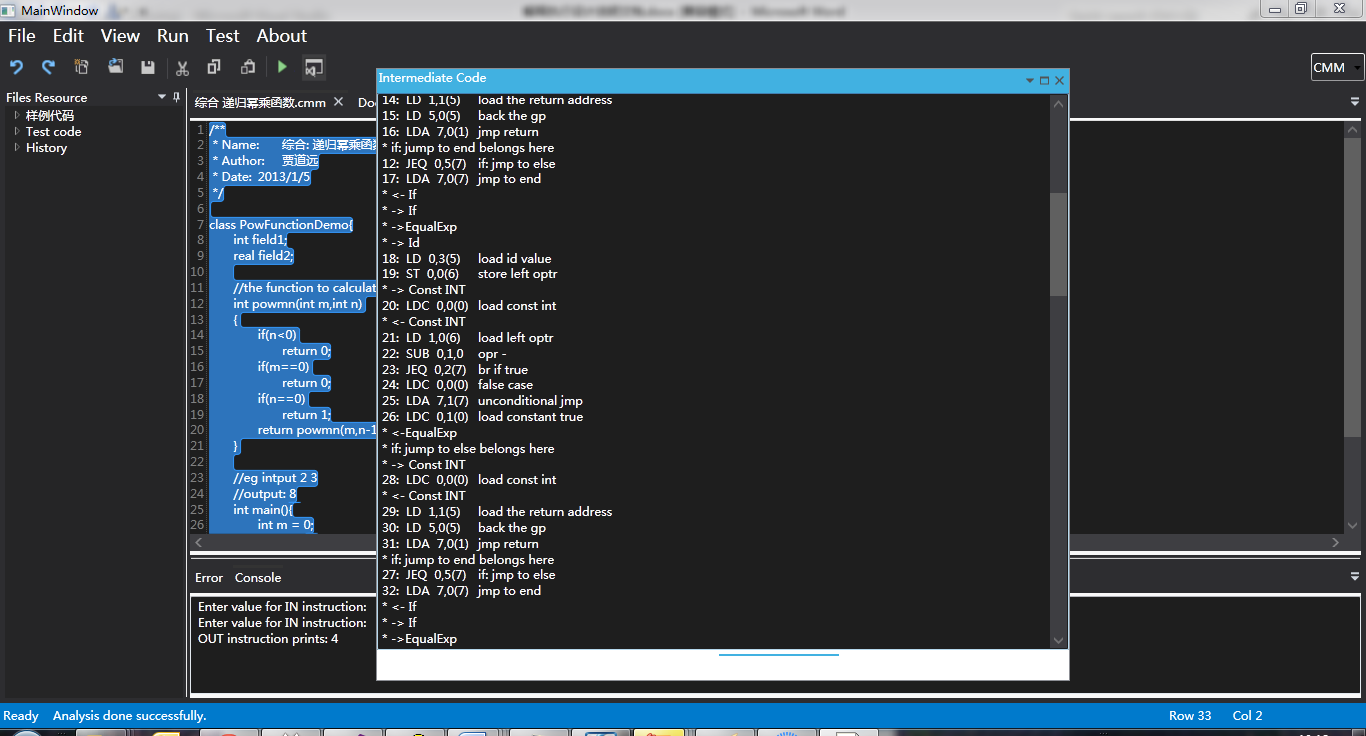
输入以下内容，得到输入内容和参数输入截图如下：



执行结果如下图：



中间代码如下图：



### 五、使用说明（可选）

WIN 7环境

对运行所需要的环境及基本使用方法进行简要说明。

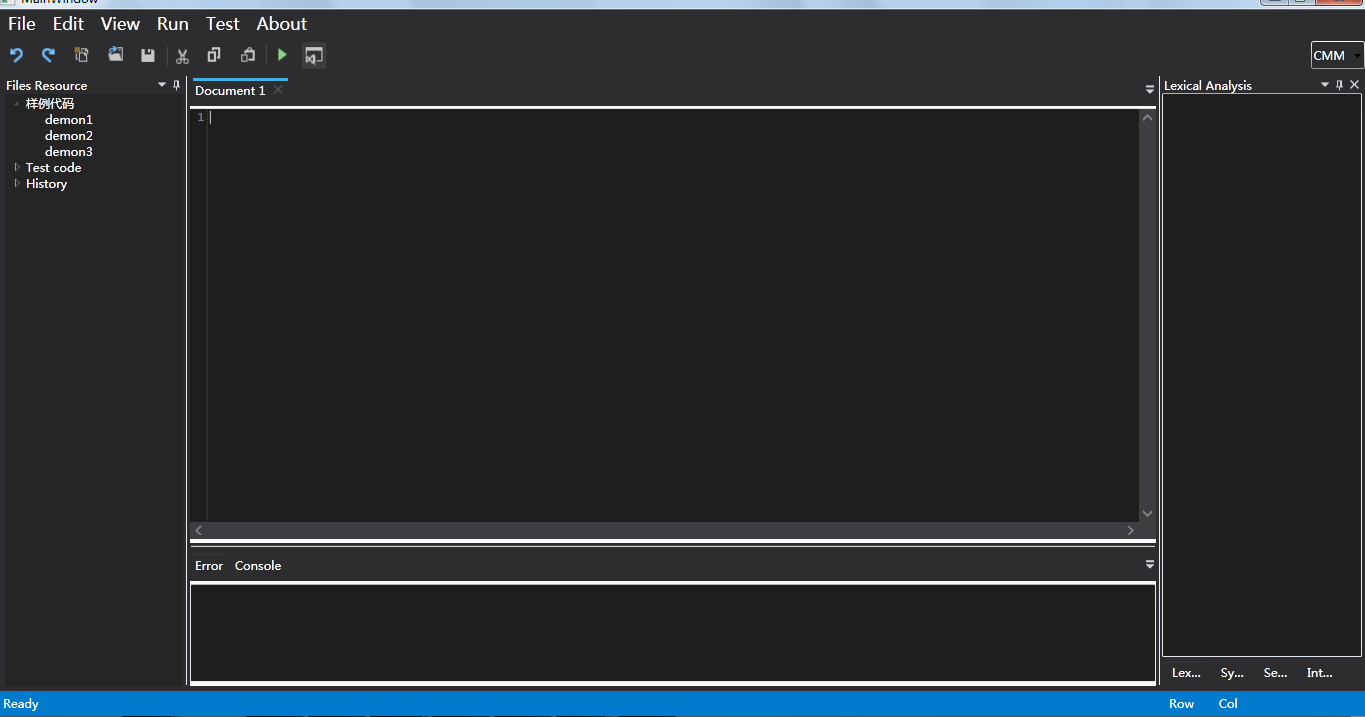


图1

左侧可以打开历史打开文件。或者点击BUTTON里的打开按钮来打开文件

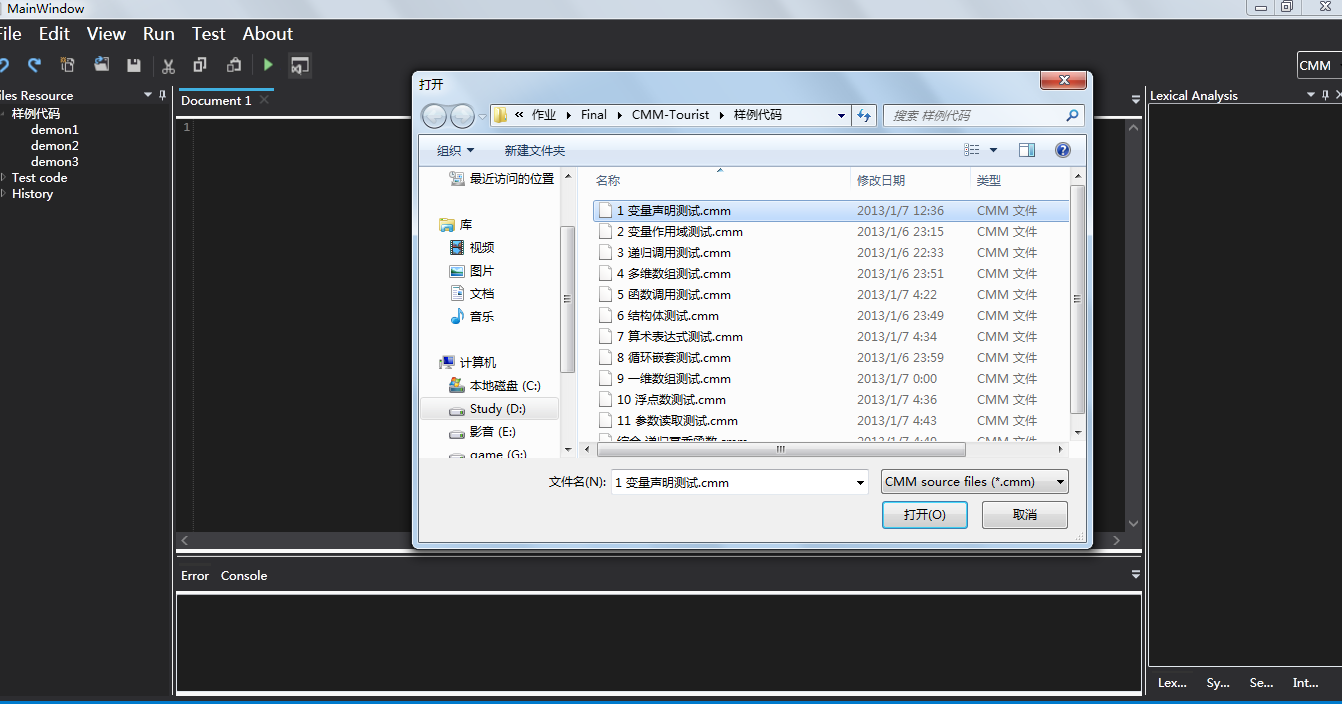


图2 点击了打开文件

点击运行，右侧默认选择显示LEXICAL Analysis结果

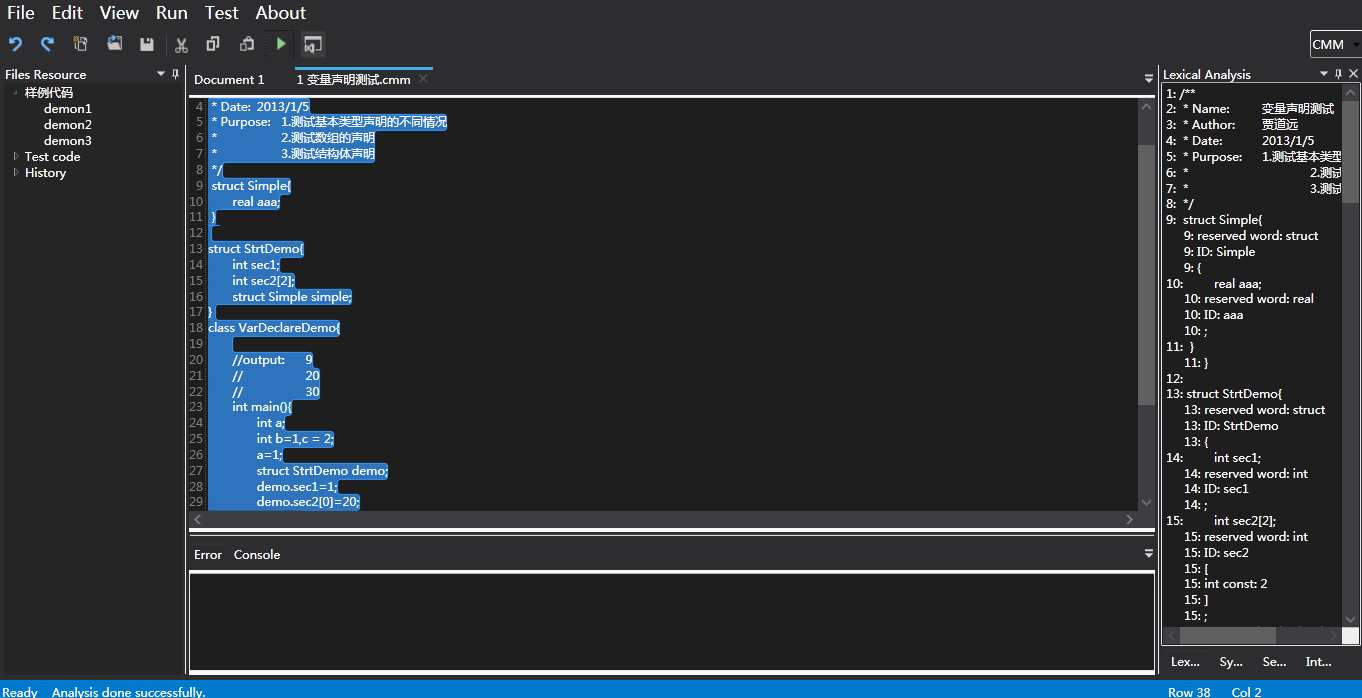


图3 运行显示分析结果

或者在VIEW视图里面选择要词法还是语法还是语义分析或者是中间代码的窗口。

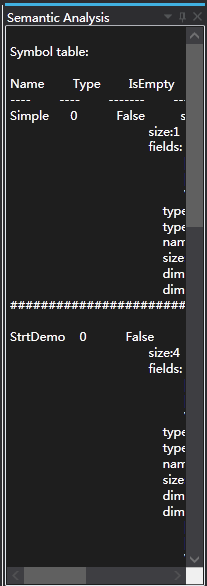
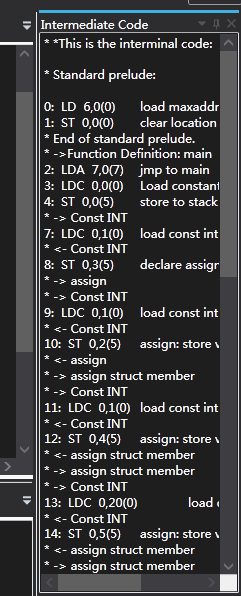
 

图4语义分析 图5 中间代码

进行拖拽窗口可以放入想放的位置

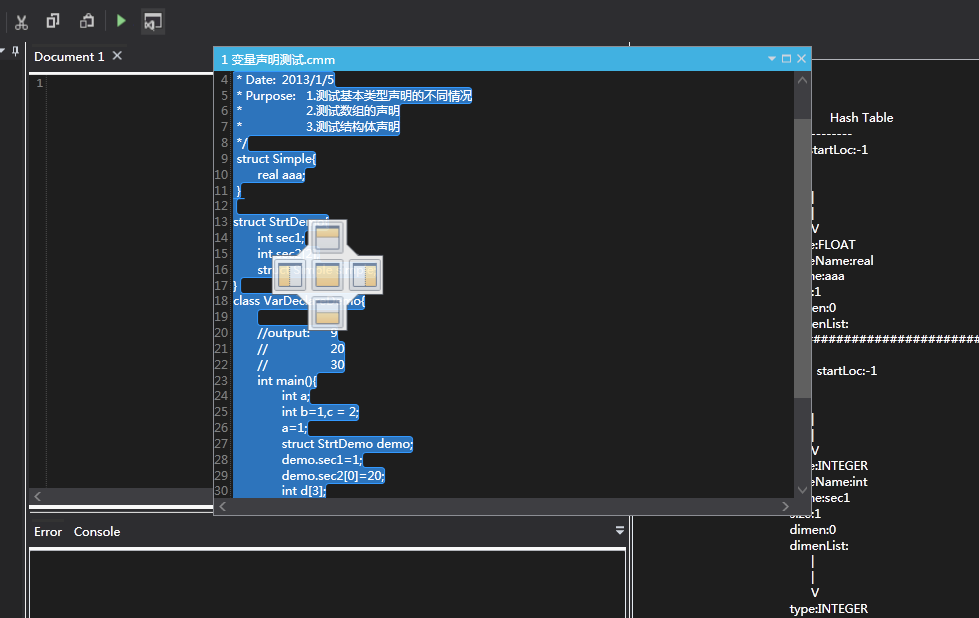


图 6 移动

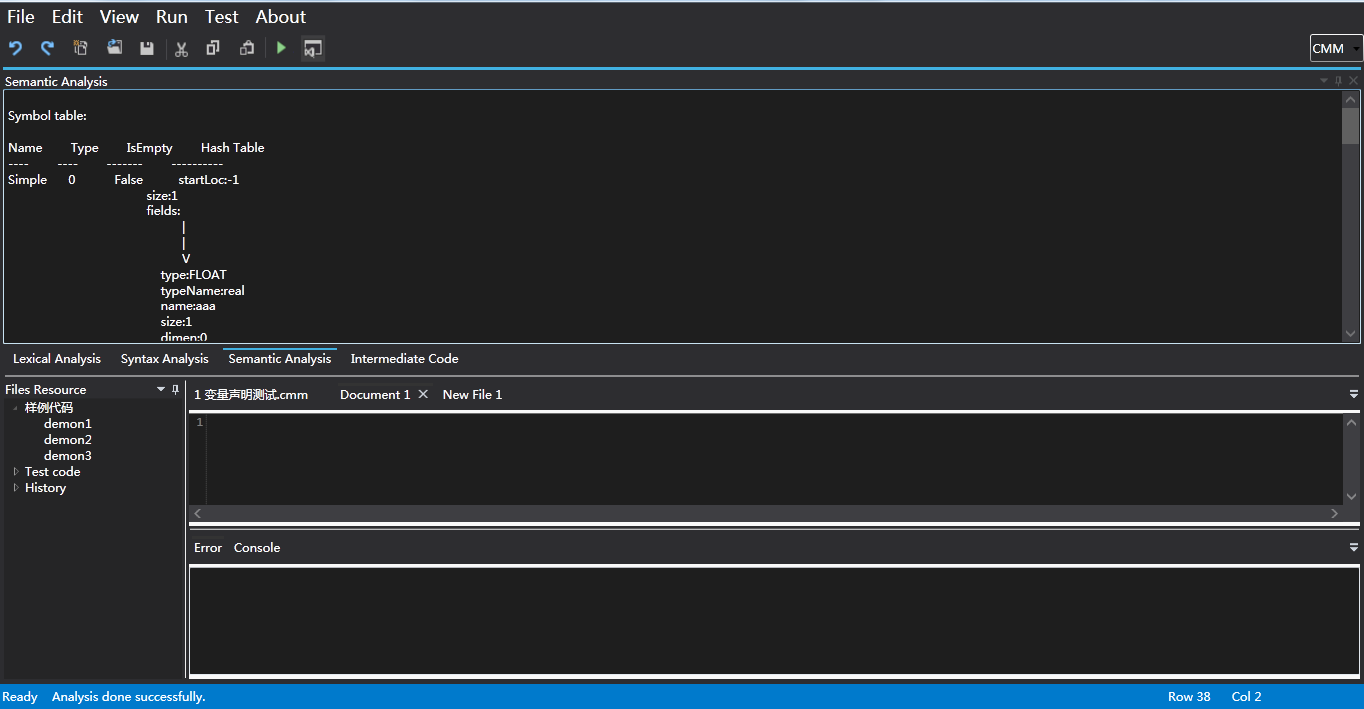


图7移动

New可以新建一个未命名的窗口

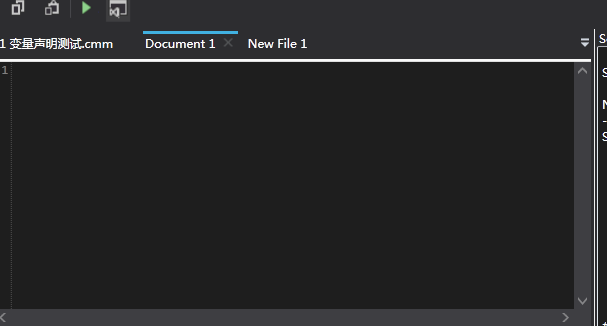


图 8 新建

详见 参照文档《作品使用手册.pdf》

### 六、小结

1. 总结本次工作，包括对完成的解释器的性特色分析、功能分析、测试结果的分析等；

2. 在完成项目中所遇到的问题及解决方法；

3. 设计实现的经验和体会。

本次工作我们小组第一次接触并使用WPF，也是第一次自己着手来编写编译器这类的软件。在先前的三次作业中，我们单独完成了编译器的词法与语法分析，对于.NET和编译器本身也有了一定的理解。在经过一番对比和讨论过后，我们决定以贾道远同学的代码为基础来完成整个编译器。本以为有了前两步的铺垫和磨练，我们小组一定能很快拿下这个工作。但当我们真正开始去做时，我们遭遇的困难超乎想象，可谓步步维艰。

由于在开发前期，大家都对相关知识的认知与考虑并没有达到一个合理的高度，所以造成文法有漏洞也是在所难免的。词法分析虽然很干脆简单，毕竟其要做的事很少，只是单纯的扫描和记录Tokens以及一些相关信息。当然，词法分析因为是站在开发的前列，我们开始对后期开发无法有足够的考虑，而且后面的分析又相当依赖于其分析结果，所以在整个开发过程中，词法分析都在根据整个设计与架构的要求进行相应的调整与完善，平白浪费了许多时间和造成大量的重复工作。

在分析与总结了贾道远同学的语法树的特点之后，我们决定使用递归的方法来解释执行。因为在前期工作上的不顺，这次我们做的很谨慎，也尽量留出更多的空挡来思考设计的问题。这不但包括整个语义分析程序执行的流程演绎，还具体到了哪个栈应该有哪些元素来填充。事实证明，我们的这个谨慎是正确的，在反复仔细的研究分析思路之后，我们发现了很多问题，避免了很多弯路，考虑的也变的全面很多。而在这些设计确定基本没有问题之后，后期的实现，也就水到渠成了。虽然没有实现附加的功能，但是在任务要求说明书里面的功能基本都实现了，这也算是给我们付出的一种肯定吧。

总的来说在完成这次解释器构造的项目过程之中，我们加深了对.NET 和WPF，也加强了对可执行界面文件的编写能力，能够将学到的编译原理理论运用于实际。同时也发现了不少问题，例如在做界面的过程中由于诸多不懂使得界面不够完善，bug很多；还有在设计前必须对程序的结构有大体的把握跟预计，每次盲目的起手编码导致后续工作的各种不便，甚至有时要推倒重来等等，在以后编码过程中我们一定会多注意这几点，并加以改善。除此之外，更是意识到了团队的重要性。在整个开发过程中，每个人的水平和优点是不同的，良好的分工与合作加上及时的沟通与交流是成功的关键。

### 七、参考资料

MSDN

CNBLOG

《编译原理(第2版)》(清华大学出版社)

**详细见 参照文档压缩包《Toureter Interpreter 参照文档.zip》**