

CMM 解释器说明文档

第 2 组

刘晓宇 2015302580121

胡婷婷 2015302580144

谢添 2015302580086

目录

(一) 框架设计	2
文法结构设计	2
代码结构设计	3
(二) 开发设计	4
项目需求分析划分优先级	4
开发模式	5
(三) 实现思路	5
(四) 特色功能	6
(五) 运行演示	7
程序主界面	7
使用介绍	8
特色功能演示	12
常规功能演示	15
(六) 项目总结	22

（一）框架设计

文法设计

cmm 文法

```
Program    → Stmt { Stmt }
Stmt       → VarDecl | IfStmt | WhileStmt | BreakStmt | AssignStmt |
            ReadStmt | WriteStmt | StmtBlock
StmtBlock  → { {Stmt} }
VarDecl    → Type VarList;
Type       → int | double | Type [ intconstant ] |string
VarList    → ident { , ident } | AssignStmt
IfStmt     → if (Expr) Stmt[ else Stmt ]
WhileStmt  → while (Expr) Stmt
BreakStmt  → break ;
ReadStmt   → read ( ident | ident[intconstant] );
WriteStmt  → write(Expr);
AssignStmt → Value = Expr ;
Value      → ident [intconstant] | ident
Constant   → intconstant | doubleconstant | true | false | stringconstant
Expr       → Expr + Expr | Expr - Expr | Expr * Expr | Expr / Expr |
            Expr % Expr | - Expr | Expr <= Expr | Expr < Expr |
            Expr > Expr | Expr >= Expr | Expr != Expr | Expr == Expr |
            ident | Constant
```

新增特色语法：

字符串字面常量及字符串变量、数组

单行输出 PrintStmt → print(Expr);











for 循环 forStmt → for(JudgeStmt) Stmt

JudgeStmt → ValDecl; Expr ; Expr

代码结构设计

采用 MVC 结构










Model 包含实体类

- ▼  model
 - >  FourElementType.java
 - >  Symbol.java
 - >  SymbolType.java
 - >  Token.java
 - >  TokenType.java
 - >  TreeNode.java
 - >  TreeNodeType.java
 - >  TreeNodeValue.java
 - >  Value.java

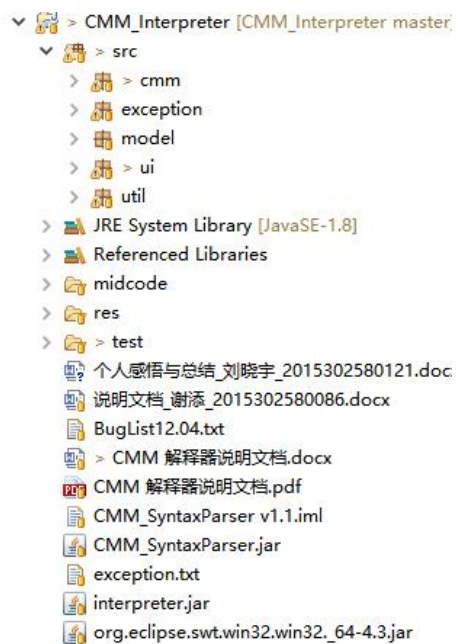
View 包含 UI 界面，处理数据显示

- ▼  > ui
 - >  AboutInterface.java
 - >  Display_Tree.java
 - >  > JavaLineStyleer.java
 - >  Main.java
 - >  > TreeControl.java

Controller 包含解释器核心三步处理，从视图读取数据，控制用户输入，并向模型发送数据。

- ▼  > cmm
 - >  GenerateMidCode.java
 - >  > Interpreter.java
 - >  Lexer.java
 - >  Main.java
 - >  SymbolTable.java
 - >  > SyntaxParser.java
 - >  test.java
- ▼  util
 - >  Util.java

总代码树



（二）开发设计

项目需求分析划分优先级

经过需求分析，我们认为首要任务是能够在期限内产出一个版本，鉴于代码前后关联度、难度、代码量三个方面的考量，我们将整个项目划分为三部分，并划分了优先级：

（1）**第一优先级**：在期限内保证能够产出一个版本，所以要完成解释器基本功能 + UI 界面。此部分分配给刘晓宇。

（2）**第二优先级**：加入工具特色——debug 功能，此项任务分配给胡婷婷。

（3）**第三优先级**：加入 IR 特色——生成字节码等 IR，此项任务分配给谢添。

项目需求	优先级	难度	代码量	负责人
词法分析	1	0	500	刘晓宇
语法分析	2	1	800	刘晓宇
语义分析	3	2	700	刘晓宇
UI 界面	4	2	1300	刘晓宇
语法特色	5	2	300	胡婷婷、刘晓宇
工 具 特 色 (debugger)	6	3	400	胡婷婷
工具特色（语法 高亮、语法树展 示）	6	3	300	刘晓宇
IR 特色(四元式)	7	3	500	谢添

（三）实现思路

词法分析器：从流中读取字符，分析，并产生一个含有词法单元的链表。

语法分析器：获得词法分析器产生的 token 列表，根据文法，采用自顶向下递归子程序法，构造语法树，返回语法树森林链表。在构造语法树的同时，会进行语法错误检查，并抛出语法错误异常。

语义分析器：使用单例模式实现一个符号表，自顶向下递归遍历语法树结点，遇见变量声明或赋值等及时更新符号表，同时计算并保存结果到语法树结点中。在遍历语法树的同时，会进行语义错误检查，并抛出语义错误异常。语法树结点需要保存信息，这些信息包括：结点类型、结点字面值、结点数据值、结点数据类型、结点字符串字面值、结点布尔值、是否为中断结点（debug 使用）以及结点的左中右子树，语义分析主要是围绕这些信息，在遍历时计算并保存。

UI 界面：

使用 SWT GUI 框架。

包含打开文件、词法分析、语法分析、语义分析、调试、退出等按钮，代码区、输入区、输出区、调试信息区等文本框这些基本组件。

Debugger 特色功能模块：

断点功能

在需要增加断点或去掉断点的所在行上双击，设置其所在行的背景色，如果背景为浅蓝，则为断点行，如果背景为 null，即默认为白色，则不为断点行。在增加或去掉断点的同时，对代码所生成的树进行遍历，遍历到所在行的第一个树节点时改变其节点的 isInterrupt 值，有断点时 isInterrupt 值为 true，没有时则为 false。

调试运行功能

点击调试按钮后，开始对树进行语义分析，当判断当前节点的 isInterrupt 值为 true，即有断点时，暂停语义分析，并且在 Variable 栏中将所有变量的值打印出来。

继续调试功能

点击继续调试按钮后，从之前所停的断点节点继续分析，直到遇到新的断点或者运行结束。

语法高亮特色功能模块：

借鉴 Java 语法高亮库，继承 LineStyleListener 类。

其基本思路为获取 styledtext 文本框的文本内容后，传入 LineStyle 类中处理，对于文本内容中的字符串、数字、标识符、关键字等不同 token 设置不同颜色与字体粗细。

语法树图形化展示功能模块：

此部分是为了方便语法分析及语义分析 debug 的工作，可以清晰看出语法树的结构。

实现思路为遍历语法树森林，自上而下递归遍历语法树、遇到结点时，使用画笔在 Panel 上画圆及结点类型或结点字面值，同时判断其是否存在子结点，若存在子结点，在父节点和子节点间连线，然后显示以该子节点为根节点的子树。其中值得注意的部分是，在不同层次的结点显示，需要不同的结点间距。

IR 特色模块：

通过在词法分析，语法分析和语义分析程序的基础上，将 CMM 源代码翻译成中间代码，认识中间代码的表示形式和生成中间代码的原理和技巧，掌握对简单赋值语句的翻译过程，从而达到对编译器的编译原理有更深入的理解，提高代码能力和代码修养。

而我们这里会选择四元式——一种普遍采用的中间代码形式作为我们的中间代码，其中

每条”指令“包含操作符和三个地址，两个是为运算对象的，一个是为结果的。

（四）特色功能

语法特色：单行输出

```
print (1) ;
```

字符串字面常量

```
write("CMM Interpreter");
```

字符串类型变量

```
string s = "We love CMM Interpreter."  
write(s);
```

字符串数组

```
string[3] s;  
s[0] = "Alice";  
s[1] = "Sylvia";  
s[2] = "Sophie";
```

For 循环结构

```
for(int i = 0 ; i < 8; i = i+1){  
    write(i);  
}
```

功能特色：

调试功能

可以在代码行双击加断点， 分步执行调试。详细使用将在下面的运行演示中。

语法高亮

在代码显示区可以语法高亮

语法树图形化展示

将语法树以三叉树的形式展示在面板上

IR 特色：

可以在 midcode 文件夹中找到相应测试案例的四元式 txt 文件，生成指令集。

比如 JMP 指令：

(`jmp`, 条件, `null`, 目标)
当条件为假的时候, 跳转到目标.
又如 `ASSIGN` 指令:
(`assign`, 变量, `null`, 值)
将值赋给变量.
`INT/REAL` 指令:
(`int/real`, `null`, 值/`null`, 变量名)
声明某个名称为变量, 如果第三个元素不为 `null`, 代表是声明该长度的数组.
如果第二个元素不为 `null`, 说明在声明的同时给变量赋值, 此时声明的一定是单个变量。

（五）运行演示

程序主界面

CMM_Interpreter

文件 工具 调试 帮助

+

📄

🔍

⏸

🔧

▶

🔌

Input box

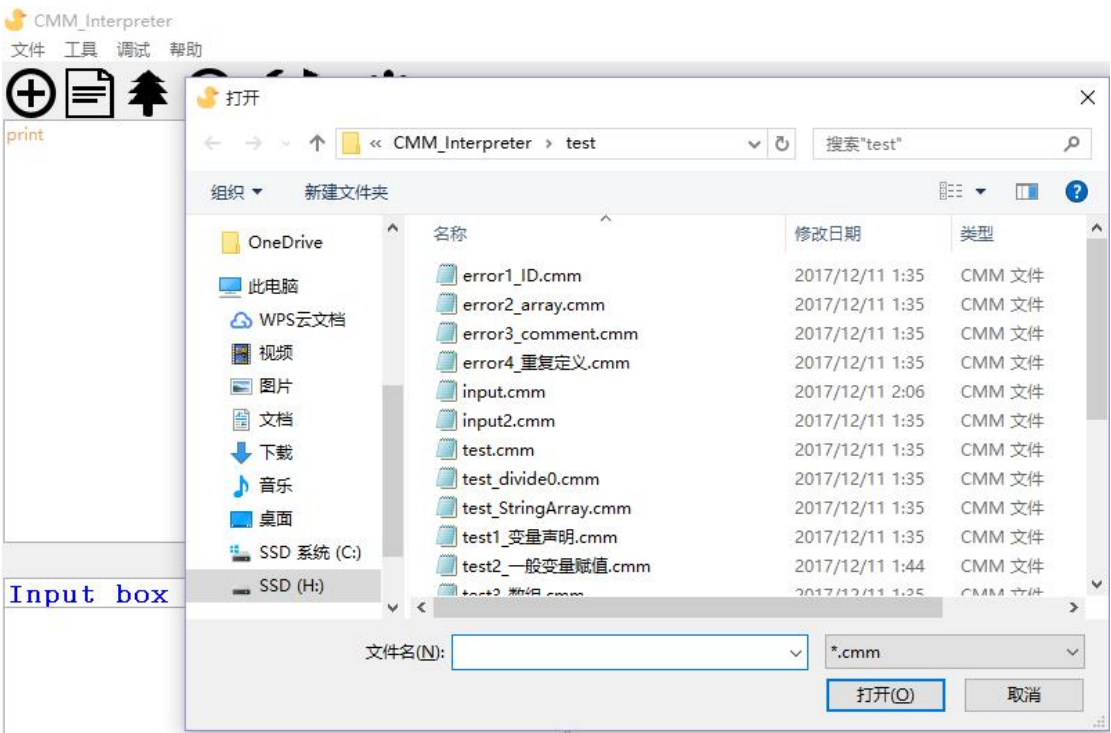
Output box

Variable

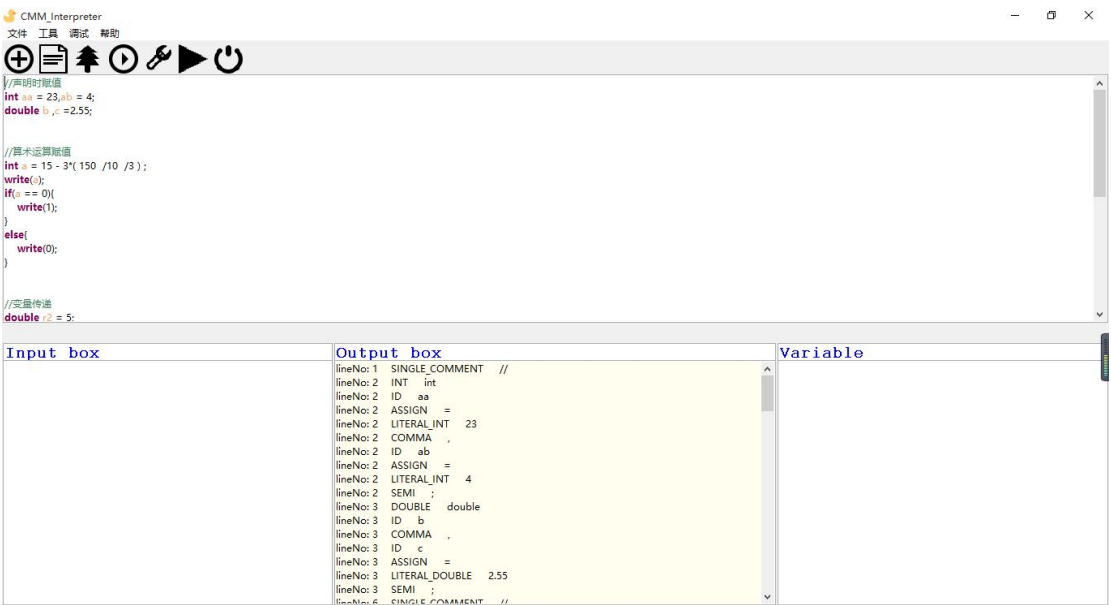
文件选项	打开、保存、退出选项
工具选项	词法分析、语法分析、解释执行选项
调试选项	调试执行、继续调试选项
帮助选项	关于选项
工具栏图标从左到右依次是	打开、词法分析、语法分析、解释执行、调试开始、继续调试、退出按钮
上方文本框	代码显示区
下方文本框从左到右依次是	输入框、输出框、调试信息框

使用介绍

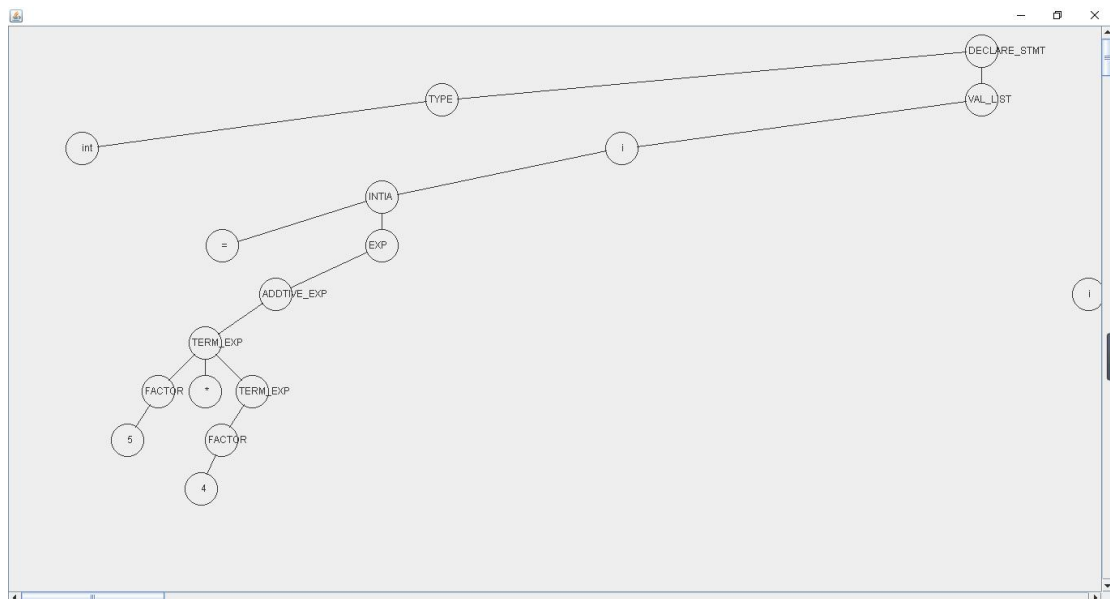
第一步，打开 cmm 代码文件



第二步，点击词法分析按钮，进行词法分析



第三步，点击语法分析按钮，生成语法树



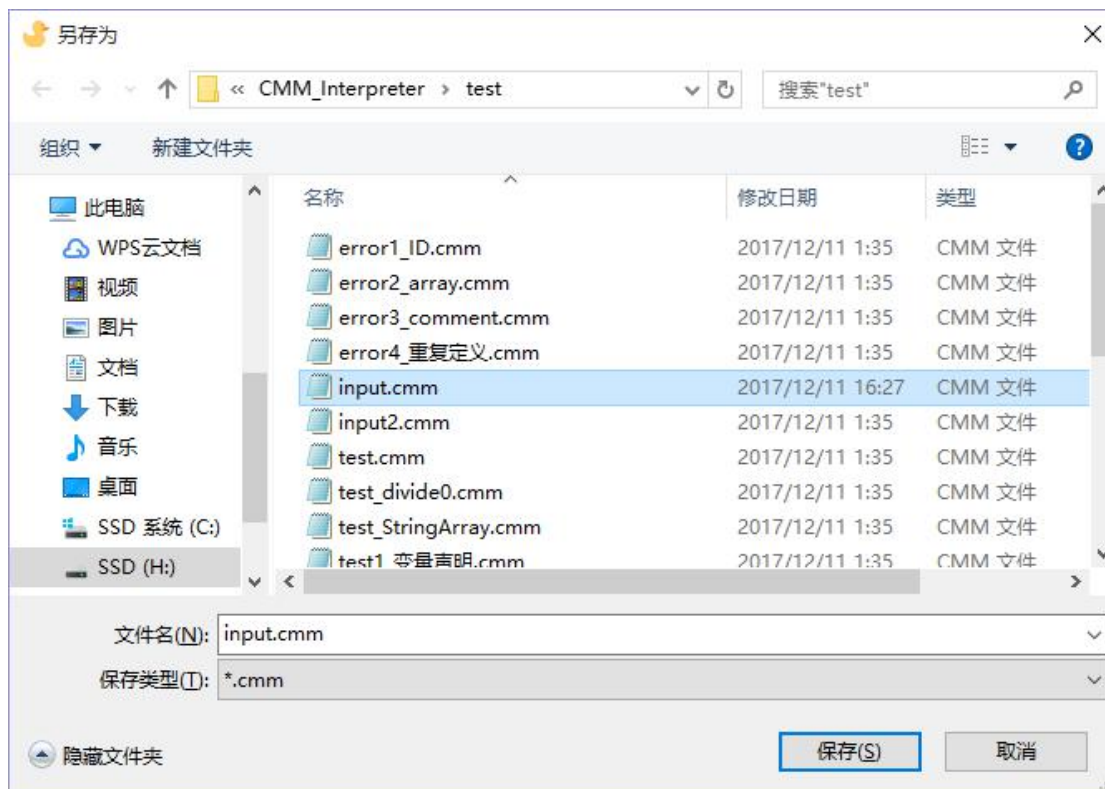
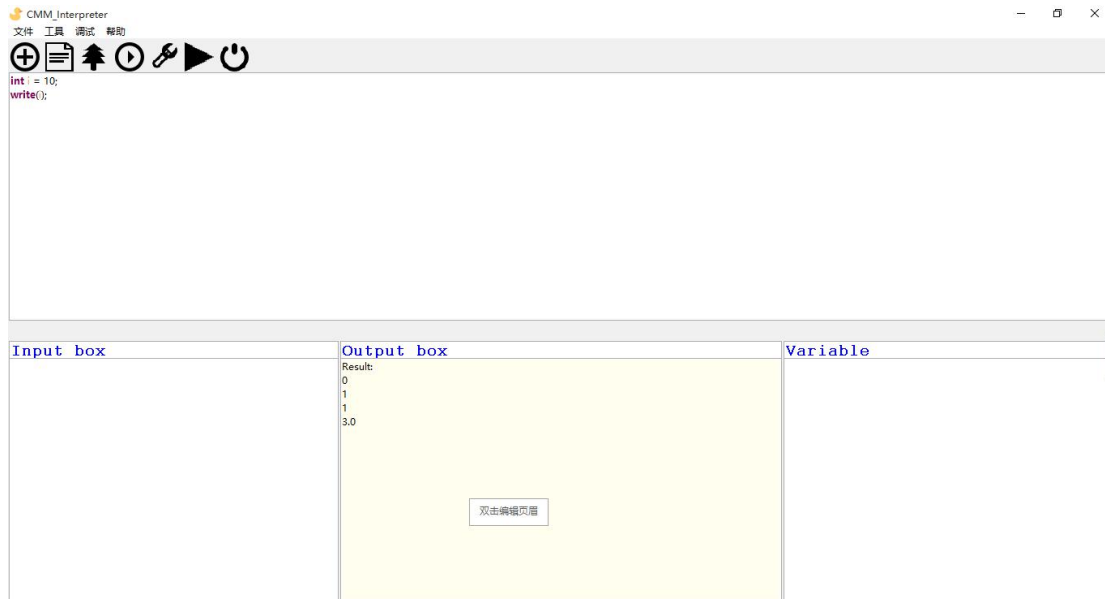
第四步，点击解释执行按钮，得到 cmm 程序结果

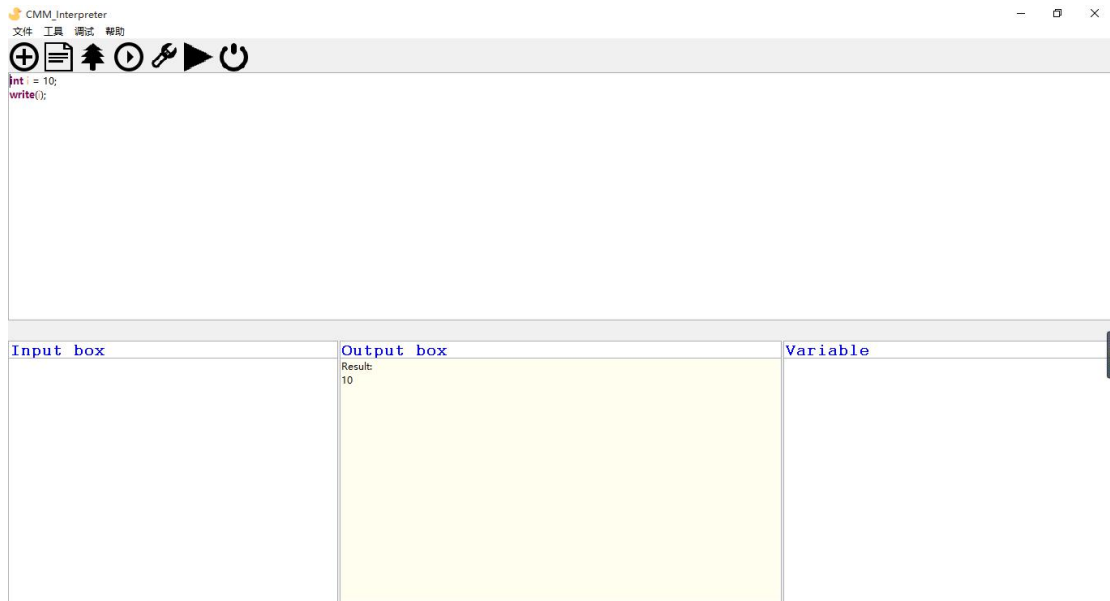
```
int a = 3;
r2 = a2;
if(r2 == 3){
  write(1);
}
else{
  write(0);
}
r2 = a2;
write(-2);
/*
output:
1
1
3.0
*/
```

Input box	Output box	Variable
	Result: 0 1 1 3.0	

第五步，在代码区可以直接进行代码编辑，例如我们测试一个简单的 demo

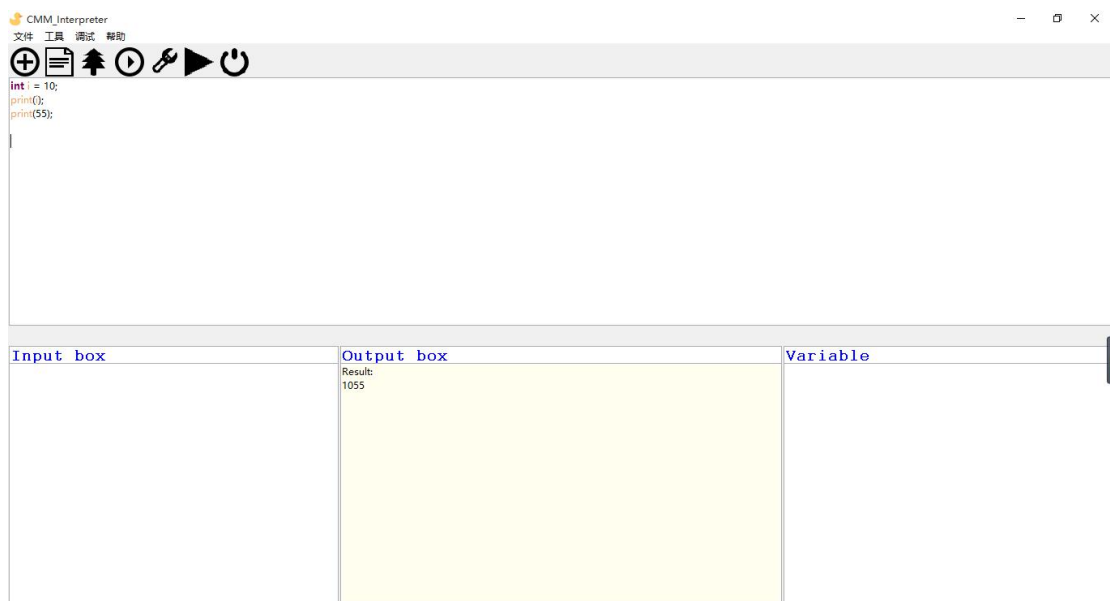
在代码区编辑好代码后，点击保存，或者使用快捷键：Ctrl + Shift + S 保存一下，即可进行分析运行。



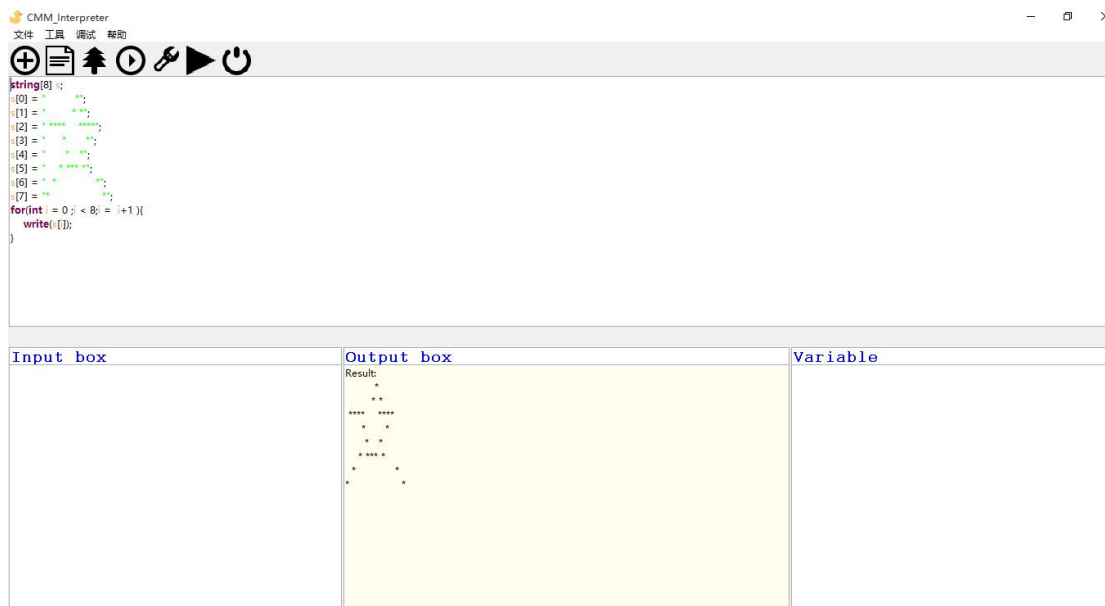


特色功能演示

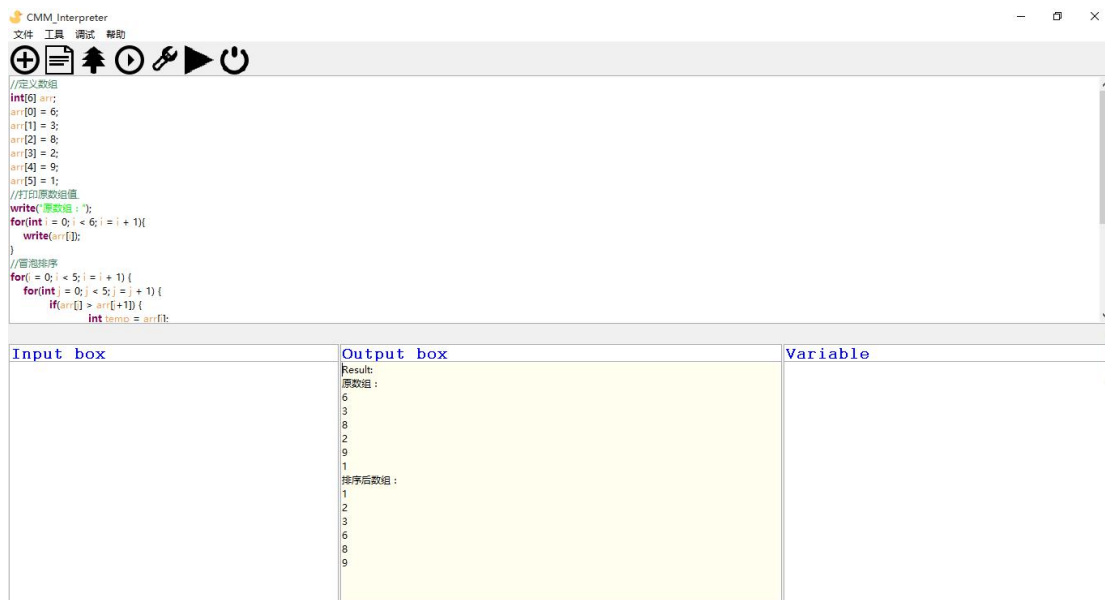
单行输出



字符串字面常量值 （打印五角星 **demo**）

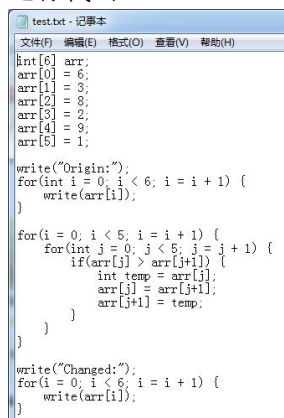


for 循环 （冒泡排序 demo）



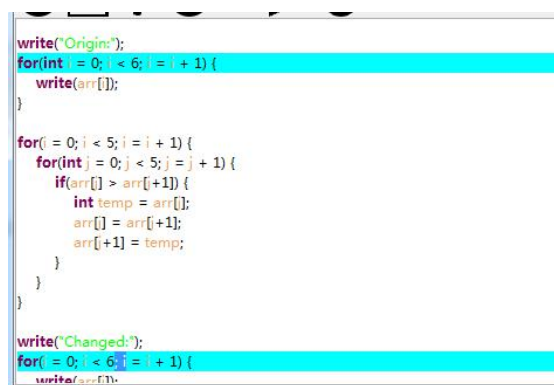
debug

运行代码



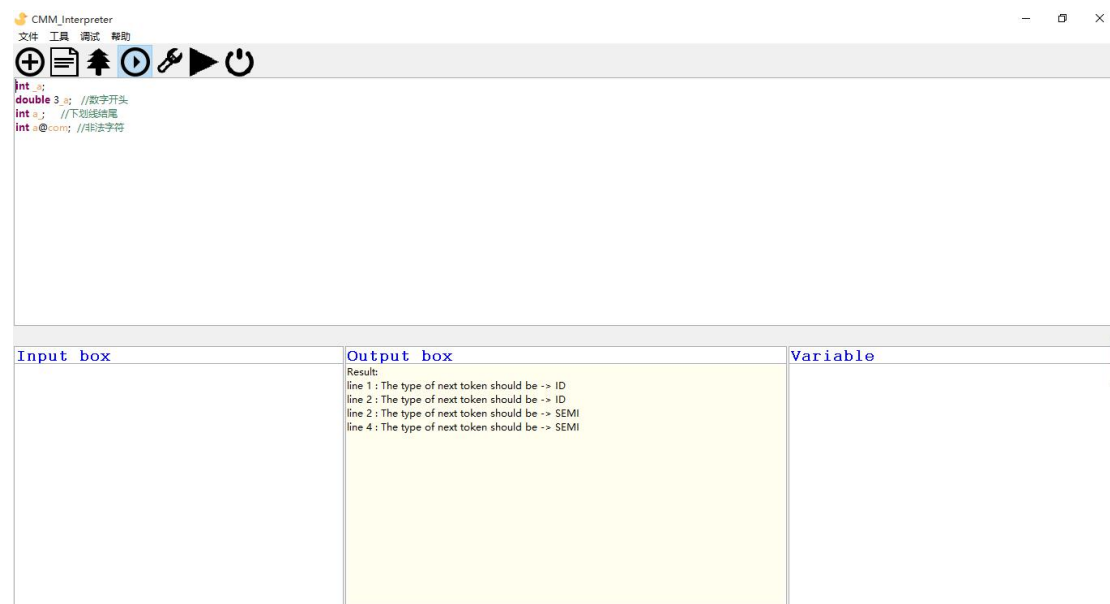
点击调试运行按钮:

在如图所示处添加断点:

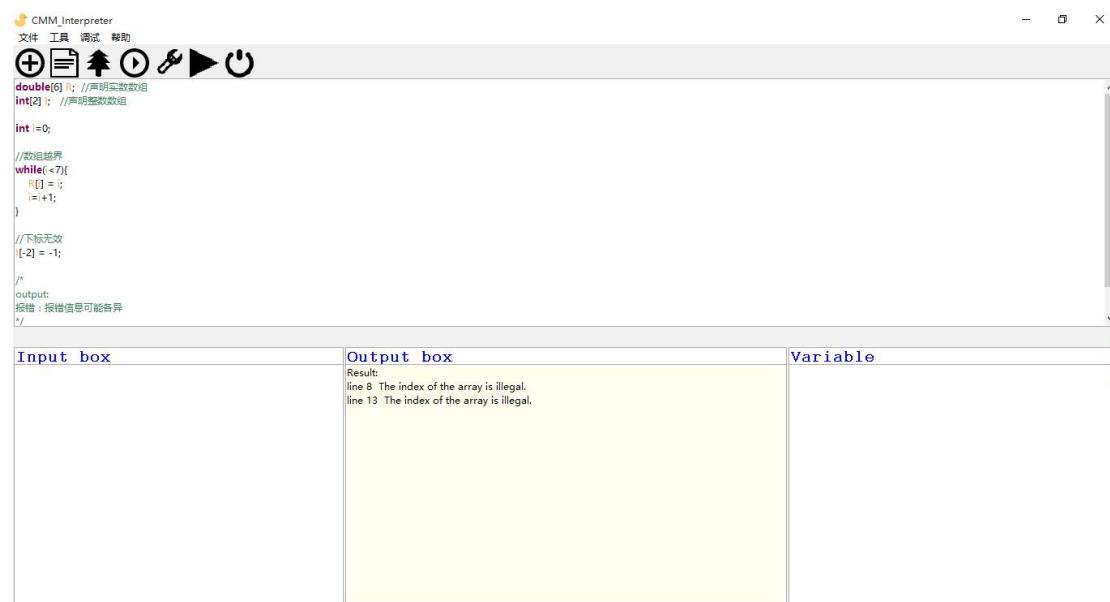


常规功能演示

变量名错误测试



数组越界错误测试



注释嵌套错误测试

CMM Interpreter

文件 工具 调试 帮助

```

int i=0;
while(<6)
{
    [] = ;
    i=i+1;
}

/*output:
报错：报错信息可能各异
*/

//注释嵌套
/* ffff /* mmmm */ ffff */
//多行注释结束
/* 报错-//

```

Input box

Output box

Variable

Result

line 15 : The type of next token should be -> ASSIGN

line 16 : The type of next token should be -> ID

line 17 : The type of next token should be -> SEMI

line -1 : The type of next token should be -> RCOMMENT

变量 <ffff> 不存在

重复定义错误测试

CMM Interpreter

文件 工具 调试 帮助

```

int i=0;
while(<6)
{
    write();
    i=i+1;
}

//相同作用域内定义同名变量
int i=10;

/*
output :
报错：报错信息可能各异
*/

```

Input box

Output box

Variable

Result

line 9 变量 <i> 重复声明

0

1

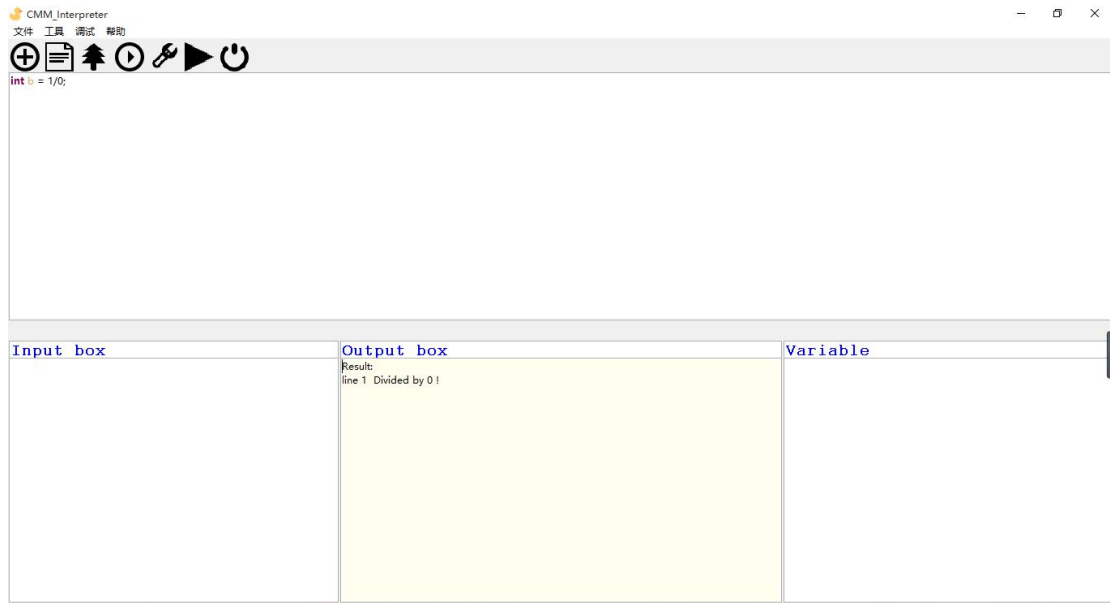
2

3

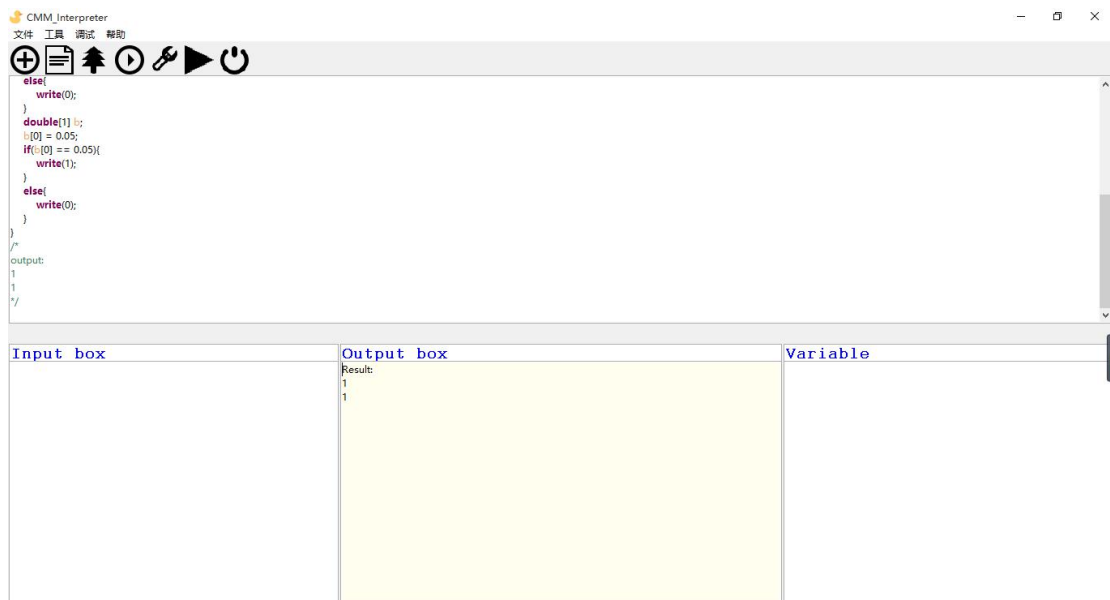
4

5

除零错误测试



变量声明测试



一般变量赋值测试

CMM_Interpreter

文件 工具 调试 帮助

⊕ 📄 ⬆ ⏪ 🔧 ▶ ⏻

```
int a2 = 3;
/*
 * 2 = a2;
 */
if(a2 == 3){
    write(1);
}
else{
    write(0);
}

/*
 * 2 = a2;
 */
write(0);
/*
output:
0
1
1
3.0
*/
```

Input box	Output box	Variable
	Result: 0 1 1 3.0	

数组测试

CMM_Interpreter

文件 工具 调试 帮助

⊕ 📄 ⬆ ⏪ 🔧 ▶ ⏻

```
int i = 0;
while(i < 6)
{
    write(doubleArray[i]);
    x = x + 1;
}

/*
output:
23.33 #这里是输入的数字
2.0
2.0
0.0
0.9
0.01
23.33
*/
```

Input box	Output box	Variable
23.33	Result: 2.0 2.0 0.0 0.9 0.00999999999999999787 23.33	

算数运算测试

CMM Interpreter

文件 工具 调试 帮助

+

📄

🌲

⏸

🔧

▶

🔌

```
write(1);
}

double i = 4.000001;
write(i);

int x = 60 * 60 * 24;
int y = 60 * 60;
write(x / y);

/*
output:
1
0
4.000001
24
*/
```

Input box	Output box	Variable
	Result: 1 0 4.000001 24	

IF_ELSE 测试

CMM Interpreter

文件 工具 调试 帮助

+

📄

🌲

⏸

🔧

▶

🔌

```
int aa = 3;
if(aa < 4){
    if(2 < aa){
        if(aa != 3){
            write(aa);
        }
        else{
            write(aa-2);
        }
    }
}







/*
output:
2.0
1
*/
```

Input box	Output box	Variable
	Result: 2.0 1	

WHILE 测试

CMM Interpreter

文件 工具 调试 帮助

```

int a = 4;
while(a != 0){
    write(a);
    int i = a-1;
    while(i < 0){
        write(i);
        i = i-1;
    }
    a = a-1;
}
/*
output:
4
3
2
1
3
2
1
2
1
1
2







```

Input box	Output box	Variable
	Result: 4 3 2 1 3 2 1 2 1 1 2	

IF_ELSE 与 WHILE 综合测试

CMM Interpreter

文件 工具 调试 帮助

```

int a = 4;
while(a != 0){
    int j = a;
    while(i != 0){
        if(2 != 1){
            write(i);
        }
        i = i-1;
    }
    if(a < 2){
        write(a);
    }
    else{
        write(a+3);
    }
    a = a-1;
}
/*

```

Input box	Output box	Variable
	Result: 4 1 7 1 6 1 5 1 1	

阶乘测试

CMM Interpreter

文件 工具 调试 帮助

```
int a = 6;
int factorial = 1;
while(a != 0){
    factorial = factorial * a;
    a = a - 1;
}
write(factorial);

/*
output:
720
*/
```

Input box	Output box	Variable
	Result: 720	

数组排序测试

CMM Interpreter

文件 工具 调试 帮助

```
double[] a = {
    -1.0, -0.99,
    -1.0, 5,
    4.01, 3.0
};

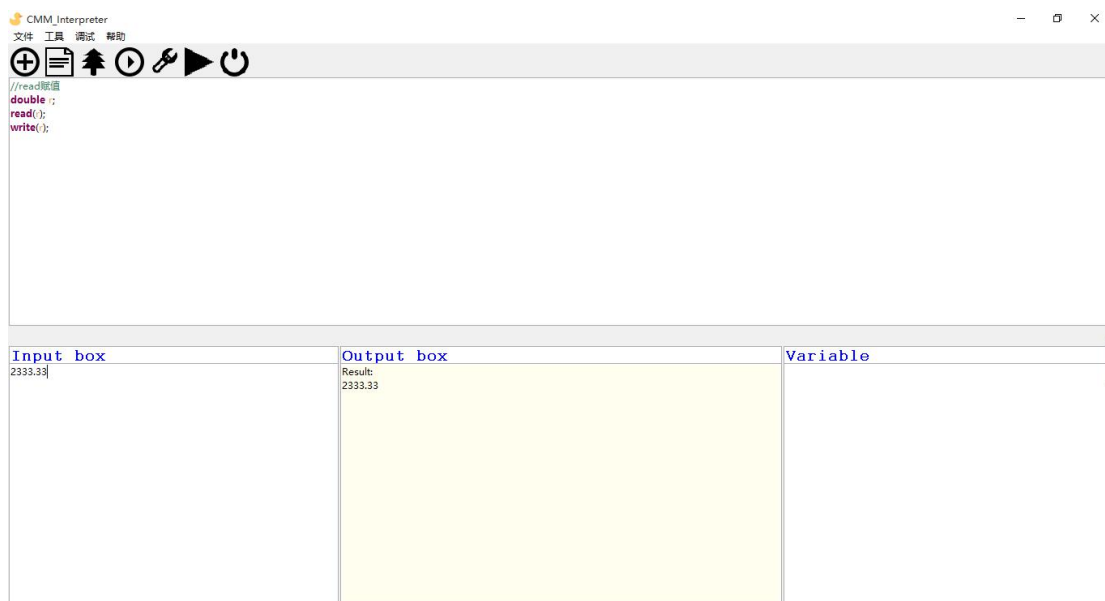
int i = 6;
int j;
int swap;

while(i != 1){
    swap = 0;

    j = 1;
    while(j < i-1){
        if(a[j+1] < a[j]){
            swap = 1;
            // swap logic
        }
        j++;
    }
    i--;
}
```

Input box	Output box	Variable
	Result: -1.0 -0.99 3.0 4.01 5.0	

READ 测试



（六）项目总结

这次解释器项目在编译原理、解释语言的基础学习上，制作一个“玩具版”eclipse,我们不仅实现了解释器的主要功能：词法分析、语法分析、语义分析，也为CMM语言加入了for循环等语法特色，还加入了UI界面中的debug调试功能、代码语法高亮、语法树图形化展示等工具特色，此外，我们还对IR功能进行了学习和探究，虽然没有时间和精力实现CMM代码转为字节码，但我们加入了较之简单一些的四元式，使得我们的简易版eclipse更加充实，为可移植性做了一定的准备。总之，这次实践结果上可能有些许瑕疵，过为简单，但根据实践的过程来讲是一次相对成功的实践，让我们组员能够按照项目需求划分任务优先级，进而合理分配，并使用git工具协同开发，体会到了多人并肩作战做项目的无穷乐趣。