**CMM解释器项目文档**

### 一、背景

小组的基本信息：

|  |  |  |
| --- | --- | --- |
| **姓名** | **学号** | **分工内容** |
| **袁浩** | **2013302580115** | **组长,负责代码实现** |
| **蒋凯迪** | **2013302580144** | **组员.** |
| **钟敏** | **2013302580142** | **组员.** |

**班级:软工三班**

**年级:2013级**

**专业:软件工程**

### 二、文法描述

请参见cmm.jj文件

### 三、系统分析和设计

开发平台:Windows 10 Pro build 10586 ,Ubuntu 14.04.3 LTS

开发工具:Eclipse Mars.1,Intellij IDEA 15 Ultimate,Visual Studio 2015 Enterprise

1. 我们的CMM语言特色
2. 支持int,real,boolean类型 (boolean类型为我加的功能)
3. 因为支持了boolean类型,所以支持了位运算,所以干脆支持了大部分操作符(优先级如下,由低到高)

=,+=,-=,\*=,/=,%=,|=,&=,^=,<<=,>>=

||(短路或),

&&(短路与),

|(按位或),

&(按位与),

^(异或),

==,!=(相等与不等)

>,<,>=,<=(大小比较)

<<(左移),>>(右移),

+(加),-(减),

\*,/,%(乘法,除法,求余)

~(按位取反),!(逻辑非),+(正),-(负)

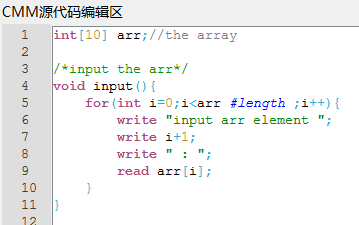
++,--(递增,前置或者后置)

函数调用(),数组下边运算[],嵌套表达式(),取变量地址#adress(预处理),取数组长度#length(预处理)

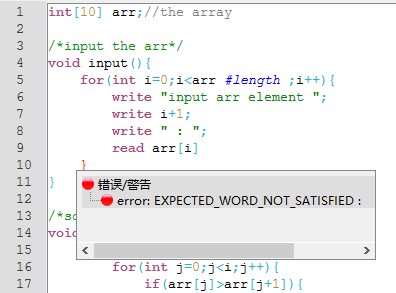
1. 支持函数(函数功能是我加的功能)
2. 函数支持参数和返回值
3. 函数支持递归
4. 函数支持先声明后定义
5. 支持的控制结构有:顺序结构-if-else结构,while循环,for循环 (for循环是我加的功能)
6. 支持break,continue和return (是我加的功能)
7. 支持数组
8. 变量默认初始化
9. 支持静态变量
10. 从main函数开始执行(规定)

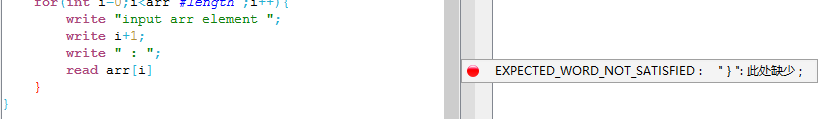
注:另请参见 CMM语言说明.txt

1. CMM解释器IDE功能
2. 语法高亮

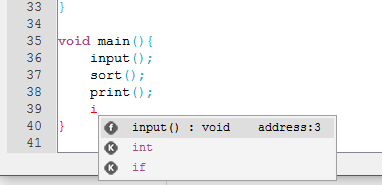


1. 错误处理

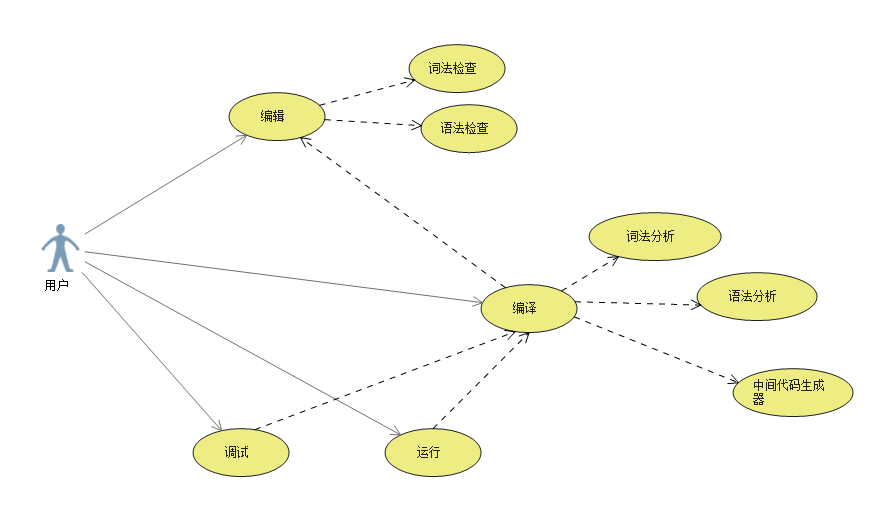




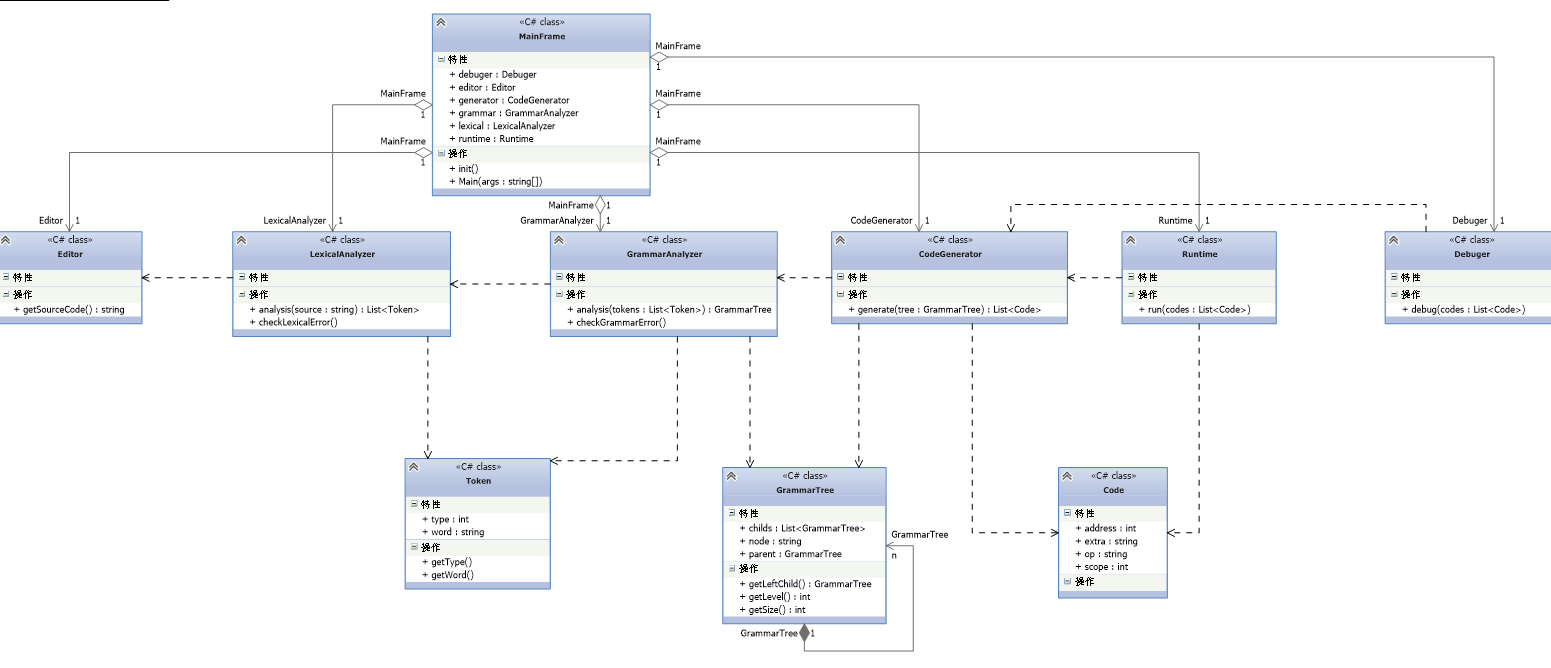
1. 代码提示,自动完成



1. javacc集成,词法分析,语法分析,中间代码生成,语法树
2. 代码自动格式化
3. 支持保存中间代码和加载中间代码
4. 运行
5. 调试
6. 调试时支持查看寄存器,支持查看变量,支持查看函数帧,支持在调试时修改相关变量的值,支持动态添加,删除,启用,禁用断点
7. 支持的调试类型有:执行下一条指令,执行到下一行,执行到下一个断点,执行到函数返回
8. 系统概要设计
9. 系统用例图(Visual Studio):



1. 基本类图(Visual Studio):



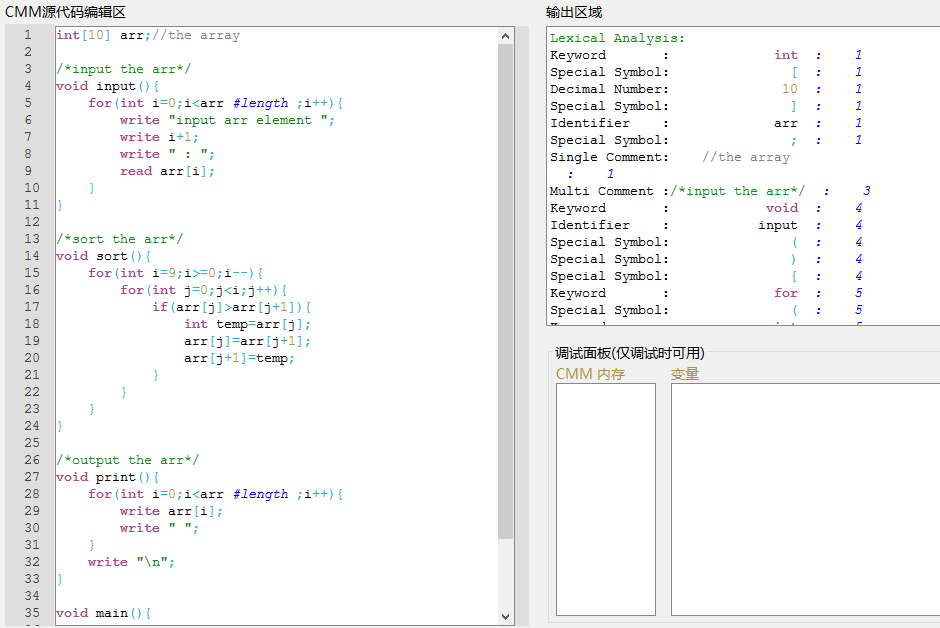
1. 详细设计

由于前几次实验已经介绍了词法分析和语法分析,这一次就主要介绍运行和调试

1. 词法分析

使用DFA,由于较为复杂,请参看实验二报告和源代码,或者直接看程序的输出

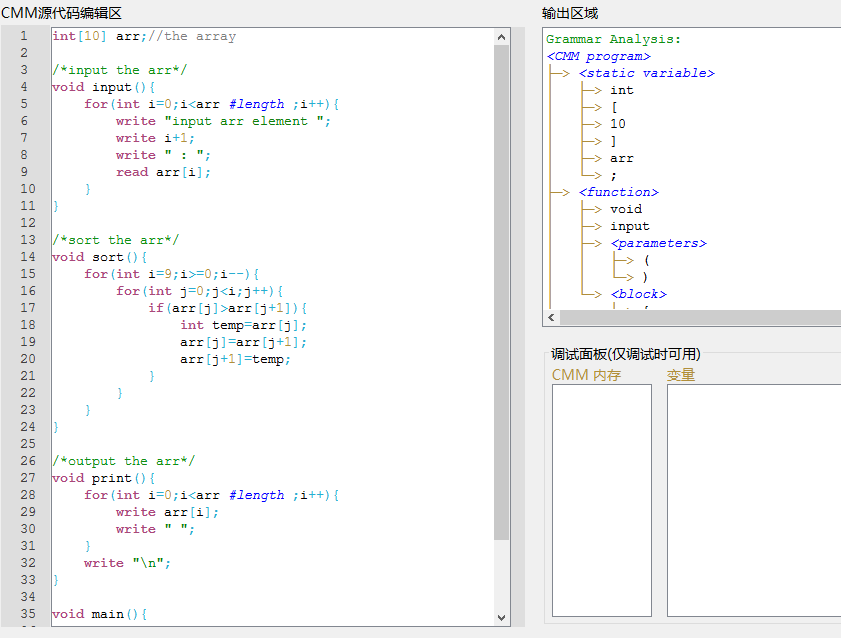
如下图



1. 语法分析

同上,请参考源代码,实验三和程序输出

如下图

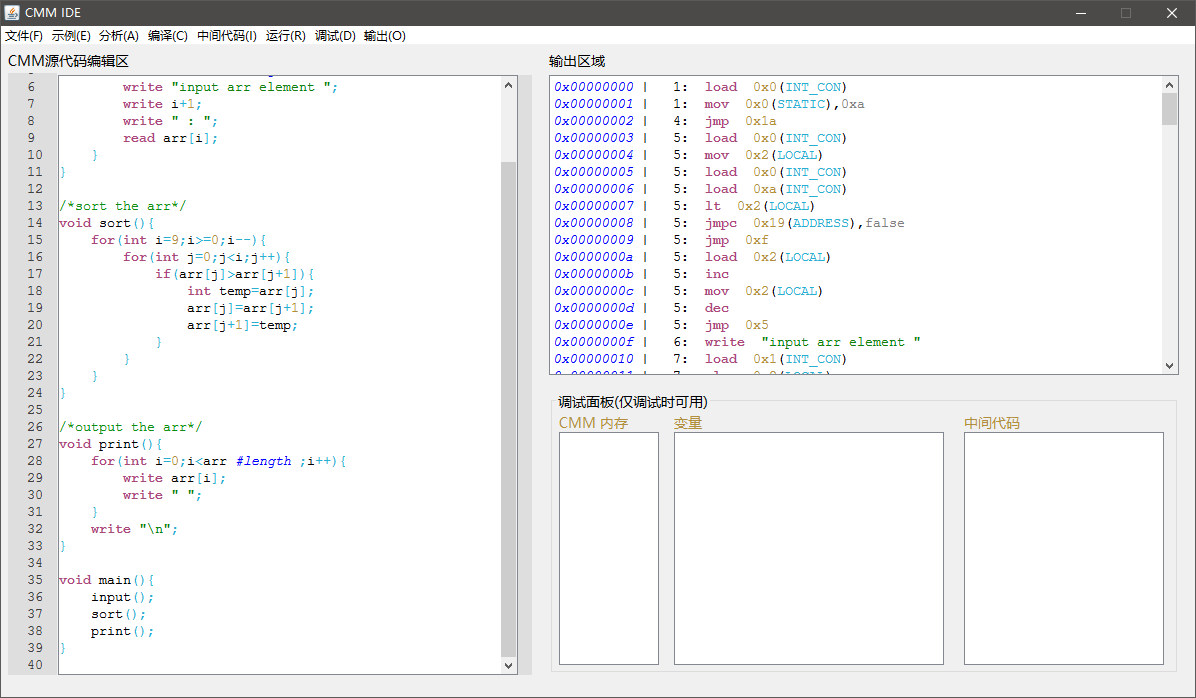


1. 运行

运行的时候先要编译,编译的过程会生成中间代码下面先对中间代码做介绍

1. 中间代码

典型的中间代码如下



蓝色字体是中间代码的顺序,黑色字体是中间代码对应的源代码的行数

冒号后面才是真正的中间代码

第一个参数是操作符,操作符定义如下(Intellij IDEA 15)



第二个参数是地址或者立即数,第二个参数的具体含义与第三个参数有

第三个参数是寻址方式,分为三种,立即寻址,直接寻址,间接寻址

如果第三个参数是立即寻址,那么第二个参数就是立即数

如果第三个参数是直接寻址,那么第二个参数就是操作数地址

如果第三个参数是间接寻址,那么第二个参数就是操作数的地址的地址,这是为了支持数组运算

第四个参数是额外的参数

1. 运行原理

真个运行环境其实就是一个CMM语言的虚拟机,运行时,程序计数器会自动递增,每加载一条指令,就会拿去执行,执行完毕再加载下一条指令,关键的结构如下(Eclipse Mars.1),其实就是一个巨大的switch-case结构



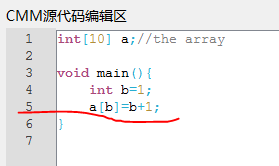
1. 寄存器介绍

CMM虚拟机的寄存器结构如下



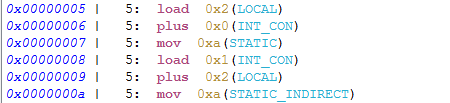
这些寄存器基本上满足了运行和调试的需求,需要重点提到的是ac寄存器,程序中所有的运算都发生在ac上面

比如这段代码



中的第五行a[b]=b+1;

会被编译为如下中间代码



第五条:就是把b加载到ac寄存器

第六条:就是把a的首地址(a的首地址在编译时确定,这里为0)加到ac上

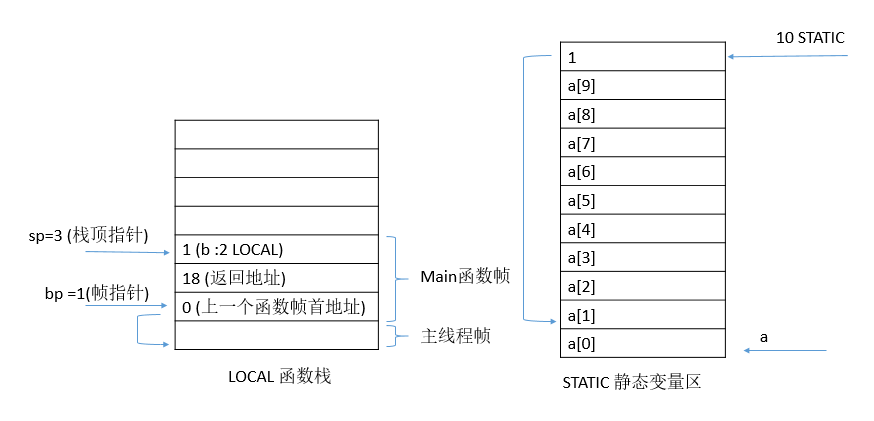
-这两条语句其实就是在求a[b]的地址

第七条:就是把ac当前的值(也就是a[b]的地址)放到了STATIC区10的位置

第八条:就是加载常数1到ac

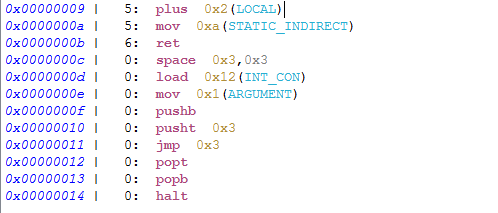
第九条:就是把b与ac的值(就是1)相加

第十条:就是把ac的值(就是b+1)放到某个地址(这个地址存储在 STATIC区10位置,其实就是a[b]的位置存储在STATIC区的10位置),如下所示



1. 内存分布

如上图所示,运行时除了的内存分布主要就是函数栈和静态变量区,其中函数栈是由一个一个的函数帧组成的,除了主线程帧之外,其余的帧至少有两个单元,分别存放上一个帧的首地址和返回地址,如上图中main函数的上一个帧就是主线程帧,其帧首地址0,而返回地址为18代表返回后执行第18条指令



如图,第0x12条和第0x13条指令为main函数退栈,第0x14条指令结束程序

除了这两个单元之外,接下来就是参数和局部变量,由于main函数没有参数,所以接下来就是唯一一个局部变量b的地址,b的地址在中间代码中为2 LOCAL,还要加上bp的值才是真实的地址.

1. 寻址方式

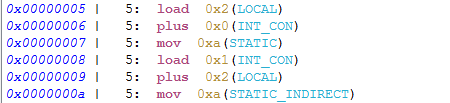
前面已经说过,有三种寻址方式

立即寻址:中间代码第二个参数就是操作数

直接寻址:中间代码第二个参数是操作数地址

间接寻址:中间代码第二个参数是操作数地址的地址

还是以第5-10条指令为例



第五条:直接寻址

第六条:立即寻址

第七条:直接寻址

第八条:立即寻址

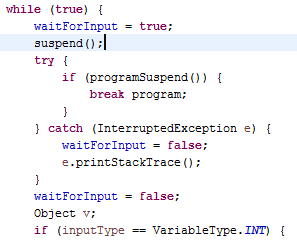
第九条:直接寻址

第十条:间接寻址

1. 模拟终端输入的实现

输入时等待其实是使用了多线程技术,关键代码如下

当需要等待输入的时候,将线程挂起:

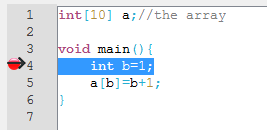


suspend函数和programSuspend函数如下:

suspend:

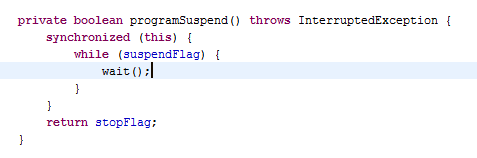


在程序挂起的时候,如果是调试模式,还要更新界面,让用户查看内存情况,如下



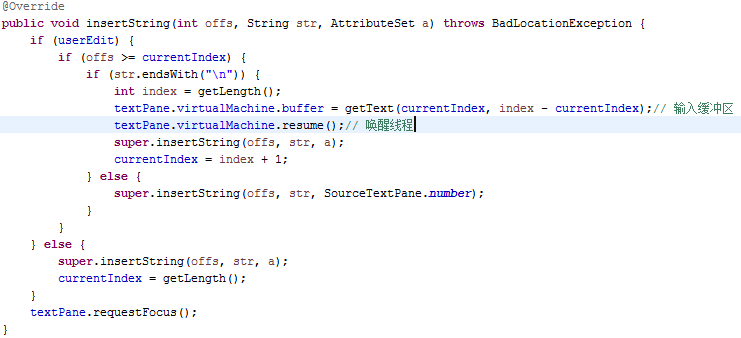


programSuspend:



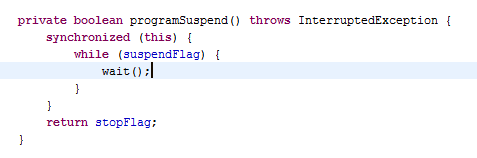
用到了多线程技术

在输入完成后,会将输入存储在输入缓冲区,并且唤醒线程



1. 多线程调试的应用

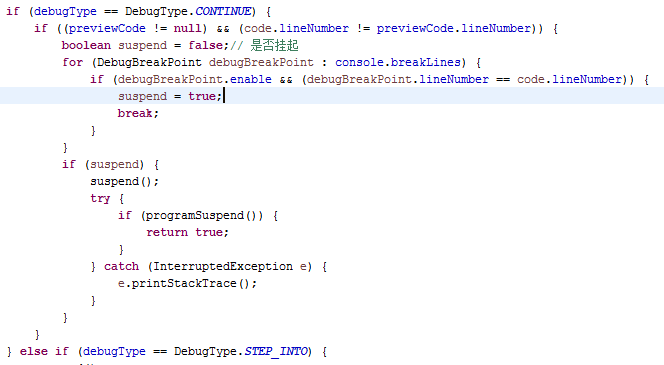
如何实现程序挂起:



如何唤醒:



如何断点调试:



1. 核心算法

前面其实已经提到了很多的核心算法(包括前面两次实验也提到了很多),这里还有一些其他的核心算法.

1. 代码提示与自动完成

在我的程序中有一个关键字列表和预处理列表,在语法分析器中有一个符号表,代码提示功能是这样实现的:

1. 检测当前输入的位置,将输入位置前面的字符取出,作为前导字符串.
2. 在关键字列表,预处理列表和符号表中查找以前导字符串开头的关键字,预处理和符号,将匹配的结果构造一个弹出菜单
3. 当点击弹出菜单或者时,将剩余的字符串插入输入的地方
4. 错误处理的思想

我们知道,每一个语法结构都有自己的开始符号集合,后跟符号集合,结束符号集合.

所谓错误处理,就是当我们的语法分析程序需要某个特定的输入符号而该符号却没有出现的时候所采取的策略.

我的策略如下,假如我需要特定的输入符号{A},但是{A}并没有出现,分四种情况考虑:

1. 遇到了其余符号,标记之后,马上跳过
2. 遇到了后跟符号,则假设用户漏写了{A},于是用一个默认的{A'}顶替(或者其他方案),并给出错误提示.
3. 遇到了结束符号集合(比如分号或者右大括号),则假设当前语法结构已经结束,给出错误提示并且返回.
4. 遇到了{A},说明在{A}之前用户可能误输入了一些其余的符号,将其忽略掉, 给出错误提示并从{A}处继续分析.



1. 返填的思想
2. 为了支持函数先声明后定义,为了支持函数的递归,在遇到函数调用语句时不会马上填写函数地址或者函数帧的大小,这个地址或者函数帧大小会在函数分析完成后返填
3. 当分析到break;语句的时候,还没有分析到循环结尾,所以并不知道循环在何处结束,所以break语句的地址需要等到循环结束的时候进行返填
4. 当分析到while语句的时候,并不知道while语句结尾地址是多少,需要返填,if-else语句也是如此,for循环也一样,只是更为复杂,需要多次返填
5. ...... (更多请参看源代码)
6. 撤销(Undo)和重做(Redo)的实现

撤销和重做的功能本来系统提供了一个UndoManager类,但是有些小问题,所以自己实现了一个,首先要实现一个队列ListQueue,然后在队列之上实现Undo和Redo的功能.核心代码如下:

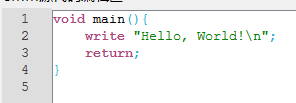


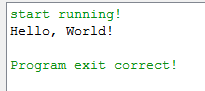
1. 参照前两次实验
2. ......(更多请参看源代码)

### 四、测试用例和测试结果

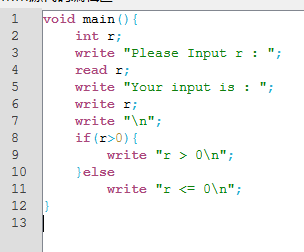
测试数据参见 示例 文件夹,也可以在程序的 菜单->示例 里面看到

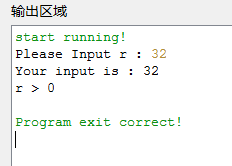
hello\_world示例



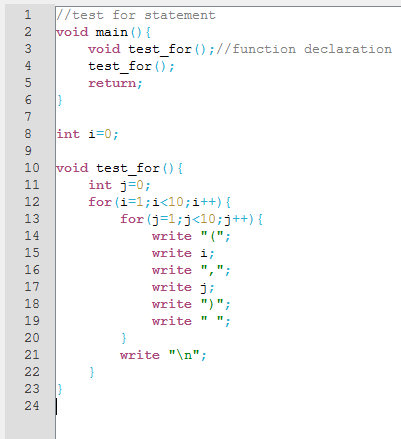


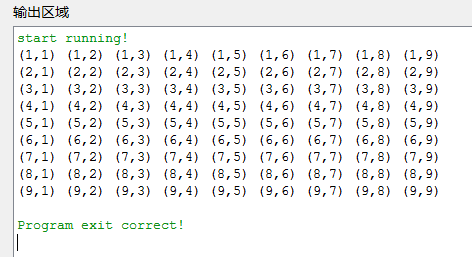
test\_input示例



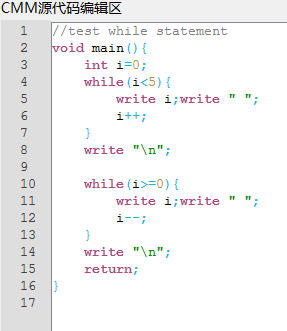


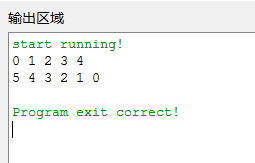
test\_for示例



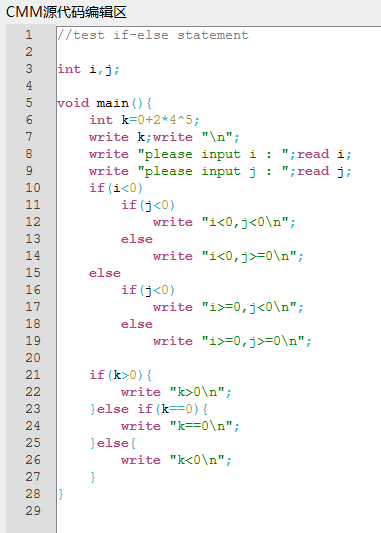


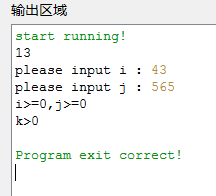
test\_while示例



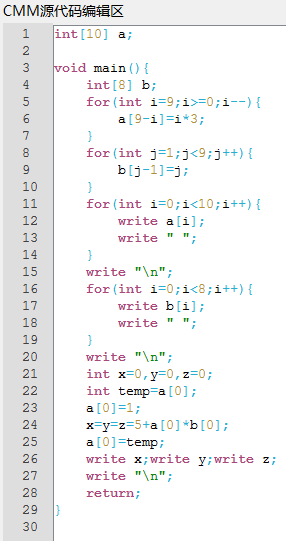


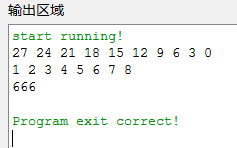
test\_if\_else示例



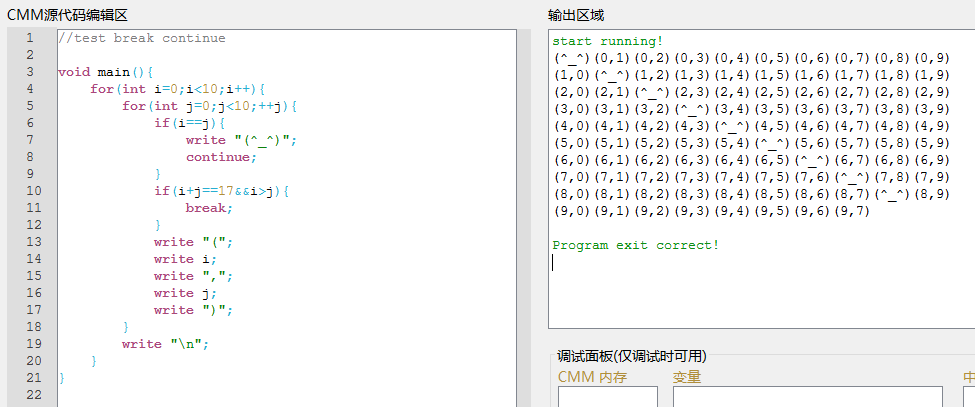


test\_arr示例

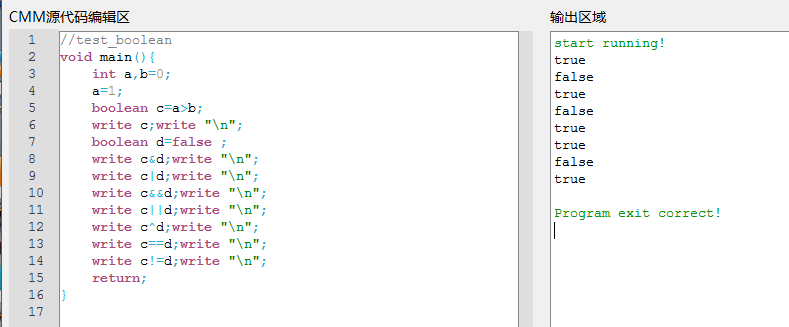




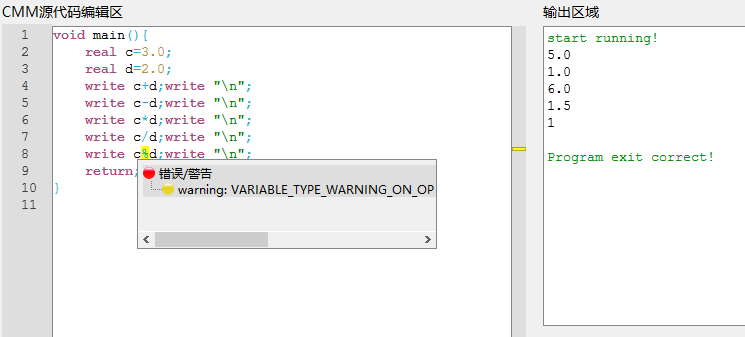
test\_break\_continue示例



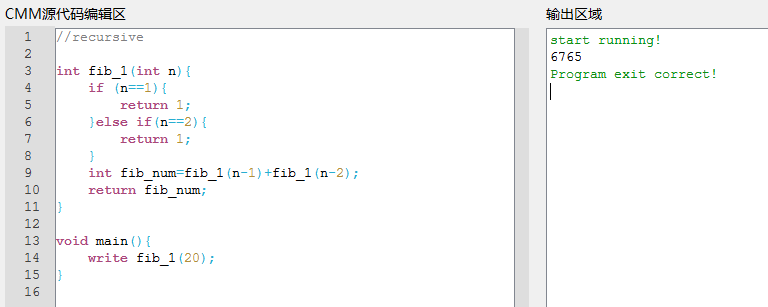
test\_boolean示例



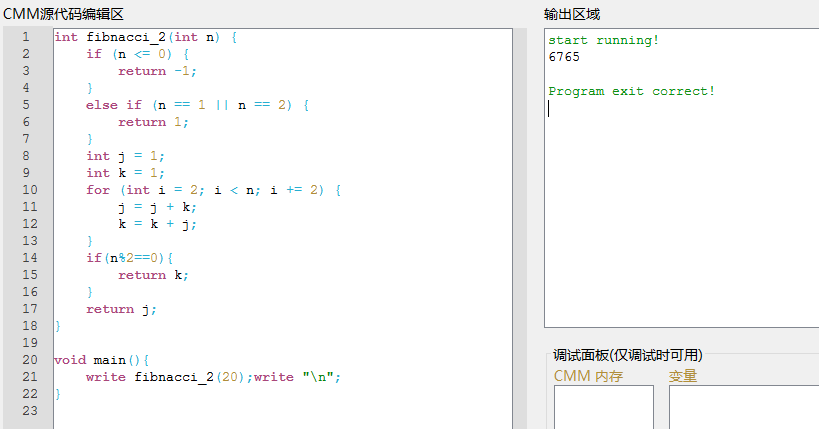
test\_real示例



recursive示例(递归函数支持)



fib2示例(综合示例1,斐波那契数列)



sort示例(综合示例2,冒泡排序算法的CMM实现)



### 五、使用说明（可选）

1. 需要使用JDK1.7及以上版本才能编译运行

2. 主要的功能都在程序的菜单上面可以看到

3. 程序提供了一部分示例,在示例菜单中

关于断点和撤销重做以及其他功能说明,具体请参见 使用说明.txt

### 六、小结

1.遇到的问题和解决方法

1. 函数支持的问题

解决方法:对内存进行按帧分配,增加bp与sp指针

1. 数组元素操作的问题

解决方法:增加间接寻址方式

1. 调试支持

解决方法:引入多线程,增加线程挂起标志和线程运行标志,增加调试标志寄存器,增加指令寄存器和前一条指令寄存器

1. 静态变量支持

解决方法:增加静态变量区,与函数栈区分离

1. 支持伪终端输入

解决方法:增加输入缓冲区

1. 控制终端编辑区域

解决方法:增加用户编辑标志

1. 撤销与重做功能异常

解决方法:移除系统接口,改为手动实现

1. 系统不支持界面垂直流布局

解决方法:手动扩展布局类

2.得到的经验

1. 要提前做好设计
2. 要有规范的代码格式
3. 要有一定的注释
4. 思维缜密,少留漏洞
5. 不要害怕手动实现,不要害怕手动控制,不要过于依赖系统提供的接口或者函数

### 七、参考资料

<<编译原理>>

<<编译原理及实践>>