数据结构

课程综合训练

队名： S.P.

成员： 白靖 2151601024 软件 52

宋士鹏 2151601032 软件 52

提交日期： 2017 年 2 月 26 日

联系电话： 13588681972（白靖）

15529277507（宋士鹏）

目录

数据结构与算法分析课程综合训练

此报告作为数据结构与算法分析课程综合训练，内容包括实验名称，需求和规格说明，设计，调试报告，运行结果展示和实验总结。程序代码采用 Java 语言实现并在 Windows 环境下测试，运行环境参数如下：

处理器及内存	CPU：Intel(R) Core(TM) i7-4710HQ @2.50GHz RAM：16.00GB (DDR3L 1600MHz)
运行操作系统	MS Windows 10 Professional x64 (Version: 10.0.14393)
Java 运行环境	Java(TM) SE Runtime Environment (build 1.8.0_102-b14)

(一) 报告正文

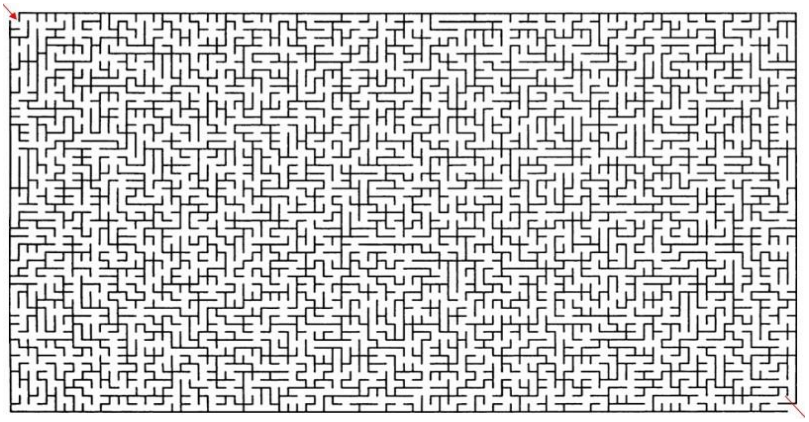
1 实验名称　实验一：迷宫

2 需求和规格说明

2.1 引言

使用 Union-Find 数据结构完成一个迷宫的生成。迷宫的入口点位于左上角，出口点是在图的右下角

如该图所示的迷宫是由 50*88 个单元组成的矩形，在该矩形中，左上角的单元被连通到右下角的单元，而且这些单元与相邻的单元通过墙壁分离开来。



2.2 需求概述

.输入：

输入两个整数，代表所生成迷宫对应矩阵的大小

所有测试代码详见百度云：http://pan.baidu.com/s/1c1E0kCw

密码：kdku

2.输出：

1）用合理的方式展现生成的迷宫；

2）给出迷宫可以走通的路径。以 SEN…（代表向南，然后向东，然后再向北，等等） 的形式给出输出结果。

2.3 接口需求

键盘接口输入迷宫行列。其余无特殊接口要求。

2.4 需求分析

该迷宫需要实现的功能有能按用户输入的两个整数表示任意 m*n 大小迷宫，迷宫之间用墙隔开，并且还要用指定的方向符号表示出可从入口走到出口的一组答案。

3 设计

3.1 设计思想

迷宫表示：

(1)由于用 m*n 矩阵表示迷宫的大小。

(2)迷宫路径初始化：使用 Union-Find 数据结构开辟迷宫路径。

(3)查找路径：用深度优先搜索 DFS 查找一条通路。

(4)打印路径：用 N、E、W、S 打印路径。

3.2 设计表示

Functions 接口：定义 Maze 类使用的函数包含以下功能的函数：

初始化迷宫	随机生成迷宫
解决迷宫	初始化并查集
把二维数组"房间"下标转为适合并查集的下标	并查集：查找集合
并查集：合并集合	判断两个集合（元素、点）是否连通
打印迷宫	打印迷宫路径
恢复迷宫初始状态	深度优先查找迷宫路径

Maze 类：迷宫的存储和操作类，实现了 Functions 接口，以给定的行数与列数进行初始化。

Test 类：测试类。

3.3 实现注释

实现了项目要求的：

（1） 输入两个整数创建 m*n 大小迷宫

（2） 合理方式表示迷宫。用’+’表示墙角，用’-’和’|’表示墙壁,’ #’(空格)表通路

（3） 给出迷宫可以走通的路径。用 N、S、W、E 表示了迷宫可通过的路径。

额外实现了：

以’#’代表路径给出路径的形象解法并打印。

3.4 详细设计表示

（1）迷宫表示：由于用 m*n 矩阵表示迷宫的大小，故以二维字符数组存储迷宫。但是由于设计思想是为每个矩阵方格用墙角符号’+’和墙壁符号’-’或’|’来表示，即，形象地以 ┆┆ 来表示一个房间（3 行 3 列）所以二维数组实际的行数和列数为 2m+1 和 2n+1

（2）迷宫路径初始化：用二维字符数组初始化迷宫完成以后，此时的迷宫除入口和出口以外是全封闭的。使用 Union-Find 数据结构开辟迷宫路径以生成迷宫，设计思想为从出口开始随机查找一面为’-’或’|’的墙，如果该墙所连接的迷宫方格之间不连通，则拆掉这面墙，反之如果连通选取另一面墙，直到出口与入口是连通的，这样就保证了迷宫至少有一条通路。对应在 Union-Find 数据结构上的操作就是先将各个房间视作独立的一棵树，且树只有一个结点。在房间组成的森林中，任意选取两颗树，若这两棵树不在一颗树中，则将小树（高度小）的父节点直接连接在大树（高度大）的父节点下，直至出口与入口房间对应的树的父节点是同一个结点，在寻找父节点的过程中压缩路径，即直接将子孙结点连接在树的根节点上。

（3）查找打印路径：用深度优先搜索 DFS 查找一条通路，遇到墙则视为不连通，并用符号’#’表示路径的走法(即赋值所遍历过的房间 Room)，并每探寻一个结点就讲 NSWE 中的一个压入动态数组中，最后反序输出即为路径。

4 调试报告

4.1 时空分析

存储迷宫时用到了二维数组这种数据结构，访问一个元素的时间开销仅仅为$O(1)$，查找一个元素在无索引的情况下最坏的时间开销为$O((2m+1)(2n+1))$，最好情况为$O(1)$。

另外在随机生成迷宫时使用了并查集数据结构，空间复杂度为$O((2m+1)(2n+1))$，建立一个集合的时间复杂度为$O(1)$，n 次合并 m 查找的时间复杂度为$O(mAlpha(n))$，其中 Alpha 是 Ackerman 函数的某个反函数。

给出迷宫的路径是用到了深度优先算法（DFS），由于采用图的存储方式是邻接矩阵（适合于稠密图），故空间开销为$O((2m+1)(2n+1))$，时间开销也大致为$O((2m+1)(2n+1))$（$(2m+1)(2n+1)$为矩阵中元素的个数）。

4.2 回顾分析

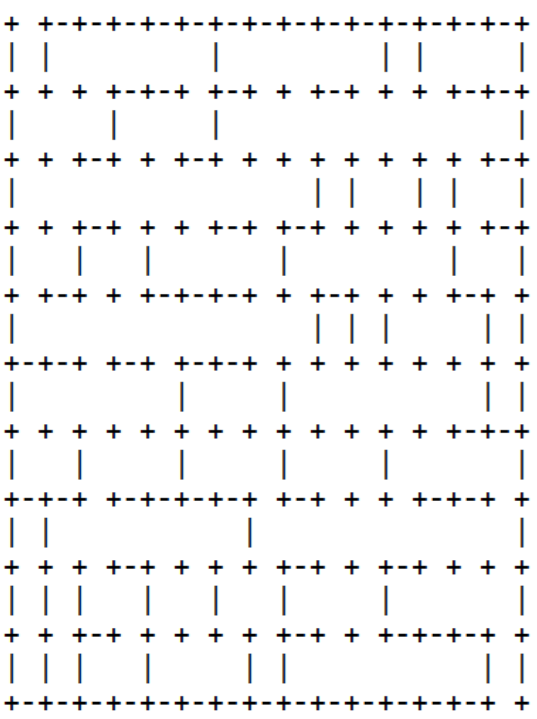
程序实现前学习并查集后，迷宫的随机生成变得不是很困难，在生成随机生成迷宫的过程中，使用了种子随机数，即利用 System.currentTimeMillis()获得当前时间的毫秒数后，利用 Random()讲时间作为种子参数传入生成随机数，保证同一线程中使用不同的随机数，是生成的迷宫更具有随机性。另外在打印路径中遇到了输出错误，经过调试检查发现需要在 DFS 中先存储走过的路径后，存入动态数组后倒序输出，调用了库 ArrayList<String> ，并使用库函数 add()直接在动态 String 类数组中存入路线结果（NSWE），最后倒序输出。最后在测试类中，为了矫正用户的输入，使用了 try-catch 模块应用 nextLine()函数确保持续输入。

4.3 改进意见

希望能将迷宫的表示改进为 OpenGL 编写的图形界面，迷宫会比控制台上打印的更直观，并且希望引入新的迷宫参数 complexity，通过 DFS 返回数字生成难度不一的迷宫。

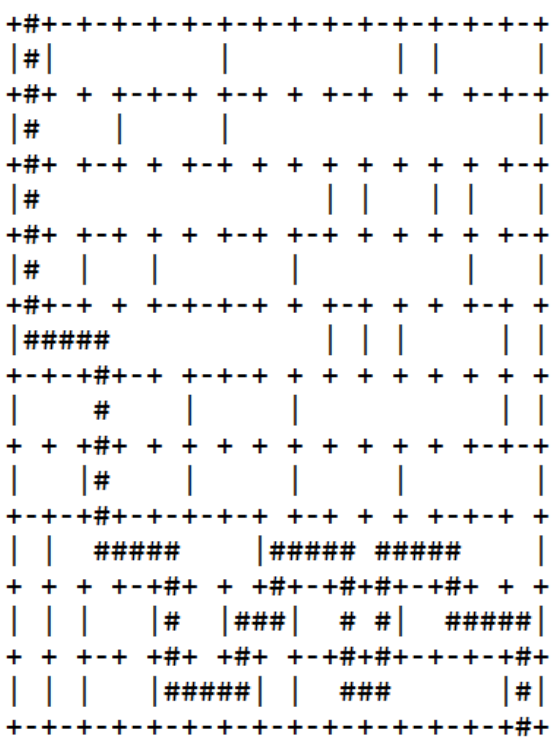
5 运行结果展示

```
请在输入迷宫的宽度：
a
输入宽度错误，请重新输入：
-1
输入宽度错误，请重新输入：
15
请在输入迷宫的高度：
10
迷宫生成成功！
```



路径表示及图示如下（上下左右分别以NSWE表示）：

SSSSSSSSSEEEESSSSSSEEEESSSEEEEEEEEEEEEESSSSEEEEEEEEEESSEEEES



1 实验名称　实验二：微型编程语言解释器

2 需求和规格说明

2.1 引言

解释器（英语：Interpreter），又译为直译器，是一种电脑程序，能够把高级编程语言一行一行直接转译运行。解释器不会一次把整个程序转译出来，只像一位“中间人”，每次运行程序时都要先转成另一种语言再作运行，因此解释器的程序运行速度比较缓慢。它每转译一行程序叙述就立刻运行，然后再转译下一行，再运行，如此不停地进行下去。该题目要求实现的是一个能够解释执行具有赋值语句、函数定义语句以及函数执行语句的小型解释器。

2.2 需求概述

（一） 输入

输入由一组赋值语句、函数定义语句以及函数调用语句构成。

赋值语句具有如下的定义形式：ASSIGN <variable> <expression>。variable 是用一个仅包含英文字母所构成的长度不超过 8 个字符的字符串构成；expression 是一个算术四则混合运算表达式，其包含的运算符就是+、-、*、/ 四个算术运算法，其操作数要么是正数（只需要考虑整数形式）要么是已定

义的 variable。表达式中的运算符和操作数使用空格的形式进行分割。一个赋值语句只占用一行，具体举例如下：

ASSIGN X 1

ASSIGN X X + 2

ASSIGN Number (X + 2) * (X – 2)

函数调用语句的定义形式：CALL <function> 。function 即为函数名字，函数名字和 variable 名字的定义要求是一样的。函数调用语句也只占用一行，具体举例如下：

CALL TryThis

函数定义语句是由一组语句组成，具体要求如下：

1) 第一行必须是 DEFINE <function>
2) 最后一行必须是 END
3) 除第一行和最后一行之外的所有行只能是赋值语句或者函数调用语句

具体举例如下：

DEFINE IncrementX

ASSIGN X X+1

END

DEFINE FF

CALL IncrementX

ASSIGN Y X*X

END

备注：输入的源代码中可以包含空行，但这些空行在解释执行过程中都应该被忽略。

（二） 输出

解释器执行后，其输出内容应该就是对每一条语句执行结果的显示。具体解释如下：

1) 每当执行一条 ASSIGN 语句，解释器都应该输出以下格式的内容：
   “Assigning<value> to <variable>”
2) 每当执行到函数定义语句，解释器都应该输出如下格式的内容
   “Defining function<function name>”
3) 每当执行到函数调用语句，解释器都应该输出如下格式的内容：
   “Calling function<function name>”

假设一个源代码如下所示：

ASSIGN X 1

ASSIGN Y 1

DEFINE Fib

ASSIGN TMP Y

ASSIGN Y X+Y

ASSIGN X TMP

END

CALL Fib

ASSIGN W X

CALL Fib

ASSIGN Z W * Y – X * X

在上面的输入下，解释器的执行结果应该如下：

Assigning 1 to X
Assigning 1 to Y
Defining Fib
Calling Fib
Assigning 1 to TMP
Assigning 2 to Y
Assigning 1 to X
Assigning 1 to W
Calling Fib
Assigning 2 to TMP
Assigning 3 to Y
Assigning 2 to X
Assigning -1 to Z

解释器可以忽略如下检查：

1) 输入的源代码格式总是正确的；
2) 每个变量在访问之前总是有值的；
3) 每个函数在输入的源代码中只会被定义一次；
4) 每个函数在调用之前总是被定义好的。
5) 不支持递归函数的调用，不管是直接的递归调用还是间接递归调用。

（三） 拓展需求

实现 FOR 循环，具体要求如下：

具体的循环语句格式为：FOR <expression> <statement>循环语句将根据 expression 计算的结果值执行多次 statement 语句。其中，statement 要么是一条赋值语句，要么是一个函数调用语句；expression 只有在进入循环时才会计算，而且仅被计算这一次。举例如下：

FOR 10 ASSIGN N N + N
FOR / N 2 ASSIGN X X + 1
FOR 8 CALL Fib

当执行循环语句时，输出的要求只具体到输出循环语句中对赋值语句或者函数调用语句
的输出，而不用专门写对循环语句的输出内容。

2.3 接口需求

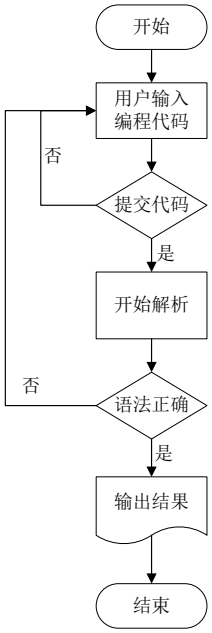
为方便用户使用该微型编程语言解释器，需要提供便于输入的键盘接口，而考虑到实际使用时，多行的编程不易于在简单的控制台中输入，该微型编程语言解释器提供用户图形界面（GUI）接口。

2.4 需求分析

该微型编程语言解释器主要需要实现的功能：接受用户输入的既定语法的编程语言；存储输入的变量及函数；解析输入的程序语句，实现变量的定义，赋值，运算，函数的定义，调用，for 循环结构的实现；判断输入程序的正确性；给出正确的输出并展示给用户。

2.5 建模论述

注：因为本程序功能较为简单，故只建立系统流程图进行论述



说明：程序开始，用户输入编程代码，点击“开始解析”提交代码，程序读取用户输入并进行解析，如果发现错误，则不给出输出结果，用户可继续修改代码。若程序正确，则输出解析结果，程序结束。

3 设计

3.1 设计思想

用户的输入采用哈希表（散列表）进行存储，具体实现的数据类型是数组和链表，其中哈希表的构造算法使用直接定址法，即直接使用输入的首字母定位元素在表中的位置。

表达式的运算采用栈进行实现。

解决变量名，函数名存储冲突的算法采用开散列法。

用户图形界面使用 Java 的 JFrame 进行实现。

3.2 设计表示

Node 类：结点类，用于存储输入的变量名及其对应值；函数名及其对于的函数体；各类关键字，以及下一个结点。是设计中存储的基本单位。提供以函数名，下一节点为参数

的构造函数，以下一个结点微餐送的构造函数，返回下个结点的函数，设置和获取各类属性的函数。

ListFunctions 接口：规定了链表类 List 的行为包括下列实现功能的函数：

初始化链表	销毁整个链表
当前指针指向头结点	指向下一个结点
指向前一个结点	设置当前结点的 item 值
获得当前结点的 item 值	设置当前结点的 value 值
获得当前结点的 value 值	判断链表是否为空
判断是否在链表中	指定位置寻找结点
获得链表的长度	在尾部追加一个结点
在当前位置插入一个结点	打印链表
移除并返回当前元素	

List 类：链表类。包括三个 Node 类型的成员变量，分辨起到指向头结点，指向尾结点，指向当前结点的功能，实现了 ListFunctions 接口。

OpenHashing 类：哈希表类。包括一个 List 类型数组作为成员变量，构造函数开辟了 26 个存储空间分别对应英文字母表的 26 个字母。有实现打印哈希表，检查变量格式（由不超过 8 位的英文字母组成），向哈希表中添加结点，根据首字母寻址元素在哈希表中地址的各种函数。

Calculation 类：用于表达式的计算类。包括实现将中缀表达式转化为后缀表达式并存储的函数，计算后缀表达式值的函数。

Test 类：测试类。继承于 JFrame 类用于生成用户图形界面，有用于将变量替换为其对应值的替换函数，识别程序中的关键字的解释函数，整个程序入口的 main 函数。

3.3 实现注释

实现了题目要求的赋值操作，函数定义与调用操作。额外实现了 For 循环结构的解释。

3.4 详细设计表示

（一） 获取输入

在 Test 类生成的 JPanel 的 TestArea 中输入，并存入一个字符串数组中。

（二） 输入解释

利用 String 类型的 split()函数，先以”\n”即换行符为参数，将输入的字符串以行分割，然后以” ”即空格作为参数调用 split()函数，分解语句块，因为语句首字母只有有限情况，故进入分支结构：

1) 首字母为 A，进入赋值模块（ASSIGN）；
2) 首字母为 D，进入定义模块（DEFINE）；
3) 首字母为 C，进入调用模块（CALL）；
4) 首字母为 F，进入循环模块（FOR）；

（三） 赋值模块（ASSIGN）

探查整个字符串，定位置第八个字符（变量名的起始位置），循环遍历变量名至空格停止。读取变量名后利用哈希函数存储至相应的位置并赋值 item

值，并继续查看变量名之后的字符串，为赋值表达式，讲复制表达式中的数字转换为整型数，新建加算器对象，调用计算函数，利用栈计算表达式的值并存入变量名同一结点的 value 值。

（四） 定义模块（DEFINE）

寻找函数名：定位置第八个字符（函数名的起始位置），循环遍历变量名至空格停止。继续遍历，以分行符为分隔，将函数体中的语句以字符串数组的形式存储在函数名对应的结点中。

（五） 调用模块（CALL）

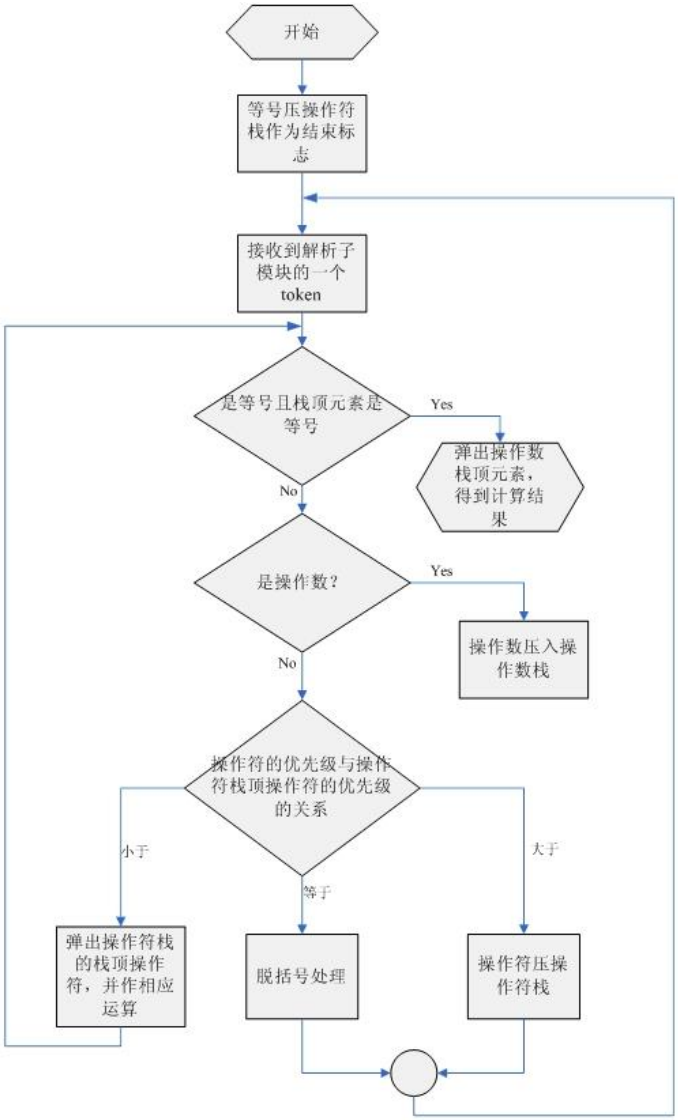
遍历存储函数的结点对应的对象数组，定位函数体，并将函数体作为参数调用解释函数。

（六） 循环模块（FOR）

读取循环次数（第五个字符），若数字只有一位，则直接传递为循环参数，若数字不止一位，则调用 String 类型的 append()函数直接在尾部追加后利用 ValueOf()再转换成数值传递循环参数，单循环参数次调用解释函数。

（七） 计算模块（CAL）

算法使用系统流程图描述如下：



说明：

1.自左向右扫描表达式，凡是遇到操作数一律进操作数栈。

2.当遇到运算符时，如果他的优先级比运算符栈栈顶元素的优先级高就栈。反之，取出栈顶运算符和操作数栈顶的两个连续操作数运算，并将结果存入操作数栈，然后继续比较该运算符与栈顶的运算符的优先级。

3.左括号一律进运算符栈，右括号一律不进运算符栈，取出栈顶运算符和操作数栈顶的两个连续操作数运算，并将结果存入操作数栈，直到取出左括号为止。

4 调试报告

4.1 时空分析

具体实现计算器类时使用了顺序栈数据结构，对于规模为 n 的输入，其空间开销为$O(n)$，对于一个不确定的表达式，频繁的入栈与出栈操作虽然消耗了大量的时间，但时间开销始终在$O(n)$中，因此该算法是有时间保障的。

实现函数名存储时使用了哈希表，解决冲突使用了开散列法，并且以首字母作为关键码 Key，因此平均成功、失败查找长度都是有保障的。

4.2 回顾分析

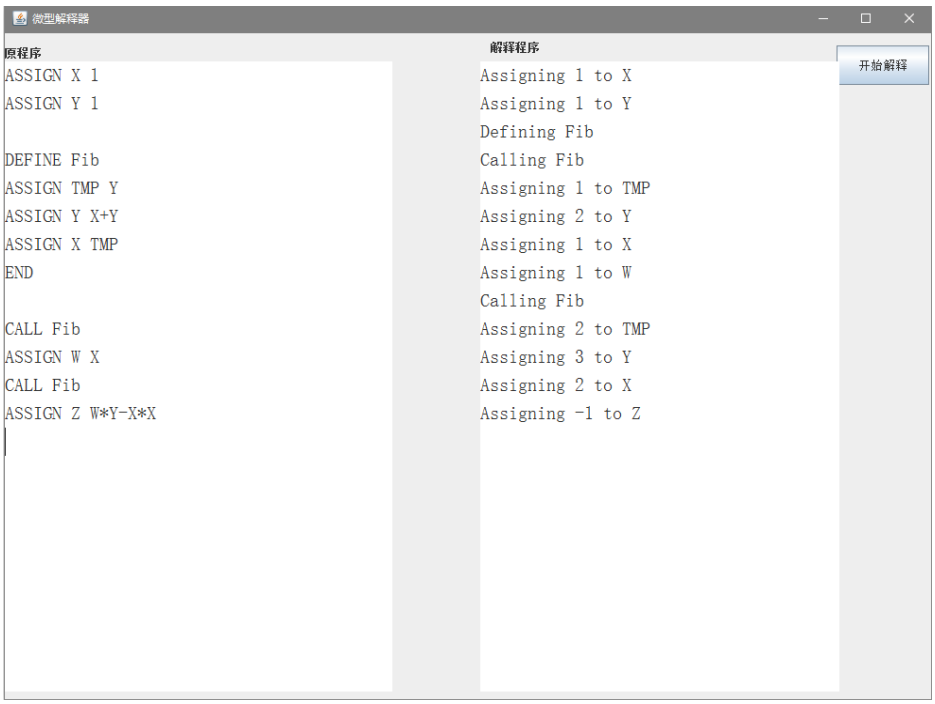
程序实现过程中算法思路是清晰的，但是到具体细节的实现与处理上遇到了不少的困难，查阅了大量资料，这些具体实现的细节大部分都是调用 Java 内置的库函数解决的。例如计算器读取表达式时，先是以字符串的形式存储表达式，在遍历字符串的过程中遇到数值则压入数字字符串中，而由于数字可能不止占一个字符，这就要求字符串长度可变，所以找到了 Java 中的 StringBuffer 类，该类型可以随时调整大小与内容，并且调用了库函数 append()，读取多位数字时，直接利用该函数在尾部追加即可。还有将字符串和整型相互转换的 ValueOf()，toString()函数等等。另外导入了 Java 中的 util.Stack 类，使用 peek()函数直接查看栈顶元素以及利用已经由库定义好的 pop()，push()函数进行出入栈操作等等。

4.3 改进意见

希望能将表达式的运算的类型扩展到浮点型而不仅仅是整型，而且支持指对数运算，另外在函数解析的阶段可以利用栈实现函数的递归调用。

5 运行结果展示

样例输入：



For 循环测试

(二) 实验总结

通过这次实验我们巩固和加深了对数据结构的理解，提高综合运用本课程所学知识的能力。并且对课程上没有充分理解的 Union-Find 数据结构有了充分的认识和了解，对常用的数据结构如链表、树、散列表等的运用有了更深刻的认识。

我们不但认识了对于具体程序选择合适的数据结构的重要性，并且会在几种数据结构中进行优化选择。如在题目一中通过分析发现使用栈这种数据结构也能实现迷宫的生成，但加大了程序的时间和空间开销，增加了程序的复杂性。没有用树形结构的并查集实现简便。

题目二中解释器的设计使我们对解释性语言和编译型语言以及计算机的编译原理有了初步的了解。

通过这次实验认识到了数据结构这门课是一门实践性很强的课程，实验是对所学知识进行巩固总结必不可少的一项环节，使我们能够更灵活的掌握所学知识。

(三) 参考文献

[1] 张桂珠，刘丽，陈爱国 .Java 面向对象程序设计（第 2 版）[M].北京:邮电大学出版社 ，2005.

[2] 苗春义．Java 项目开发全程实录．北京：清华大学出版社，2008.6

[3] Weiss, MA. 数据结构与算法分析：Java 语言描述[M]. 第 2 版. 机械工业

[4] Sesh Venugopal 数据结构从应用到实现（Java 版） 机械工业出版社

[5] Hanan Samel 多维与度量数据结构基础 清华大学出版社

[6] 张海藩. 软件工程导论. 北京：清华大学出版社， 2003.

[7] 周之英．现代软件工程〔M〕．北京：科学出版社，2000.