CS61B, Summer 2002 **Project #1: InvisiCalc** **Barath Raghavan**
UC Berkeley

Autograder Feedback Due Date: July 12, 2002 11:59 pm
Due Date: July 14, 2002 11:59 pm
Group Size: 1
Submission Command: `submit proj1`
Weight: 20 points

1 Introduction

It is often said that the first widely popular real-world application ("killer-app") for early personal computers was the spreadsheet. With its availability and ease of use, harnessing the power (by those standards anyway) of the desktop computer, it allowed people to do more powerful number crunching. VisiCalc is often considered to be the first of these revolutionary applications for the personal computer.

Your task is to implement a spreadsheet program with power similar to classic spreadsheet applications, and in some ways, different capabilities; no Apple][memory restrictions here! Note that there is no graphical user interface - all I/O is text based. The aim of this project is to familiarize you with writing a relatively large program in Java, with deciding between alternate implementations of methods, and with realizing the tradeoffs between code complexity, memory space, and running time.

2 Definitions

Before we describe the input format, we must first define cells, expressions, types, and their significance.

2.1 Cell Identifiers

Spreadsheet cells are uniquely identified as a sequence of one or more uppercase letters immediately followed by a positive integer value. The alphabetic portion of the cell identifier indicates the column; the number portion indicates the row. For example, `A1`, `Q13`, and `FAT537` are valid cell identifiers, whereas `D0` and `61B` are not. In the identifier `FAT537`, `FAT` is the column and `537` is the row. Note that after column `Z` is column `AA`, after `AZ` is `BA`, and so on. There is no limitation on the length of a cell identifier.

Cell ranges are sets of vertically or horizontally contiguous cells of the same type used for various commands and functions (for example, `A1-A7` or `A1-D1`). Cell windows are rectangular cell regions that are indicated in the same manner as cell ranges, except the bounding cells do not have to be on the same row or column (for example, `A1-B2` includes the cells `A1`, `A2`, `B1`, and `B2`). Cell ranges and windows are not ordered – `A1-B2` is the same as `B2-A1`.

2.2 Types

Each cell in the spreadsheet has one of the following types: *Empty*, *Text*, *Real*, or *Date*. All cells begin as *Empty* cells and return to *Empty* when the `clear` command is used. A cell is of type *Text*, *Real*, or *Date* if it receives an expression of that type. *Text* cells represent simple character strings that are input enclosed in double-quotes but are printed without them. *Real* cells represent real-valued floating point numbers of arbitrary precision (see the package `java.math`). *Date* cells represent a date in the format year-month-day with positive year value, and are printed with leading zeroes for month and day values less than 10.

2.3 Expressions

Expressions allow you define what to store in spreadsheet cells by use of *Values*, *Operators*, *Functions*, and *Cell References*. Expressions can be composed of any or all of the previous and parentheses to indicate precedence of evaluation, but all values, operators, functions, and cell references must be separated by whitespace from other input (for example, `(3+5)` is input as `( 3 + 5 )` to make processing easier).

Values simply represent one of the non-empty types as a literal string. For example, `"moose"` is a *Text* literal, `8.00` is a *Real* literal, and `2002-06-24` is a *Date* literal.

Operators are the standard arithmetic operators you are used to: *+*, -, *, and */*. The operators have no precedence rules; they are evaluated left to right unless parenthesized otherwise. Their behavior is defined in the usual sense on *Real* values and is undefined and disallowed on *Text* and *Empty* cells. The operators *+* and - are allowed on *Date* values; their behaviors are repectively to add or subtract a given integer number of days from the given date (for example, `2002-06-24 - 20` yields `2002-06-04`). The value being added to a *Date* value could be a real, and since it doesn't make sense to add a floating point value to a date, only the integer part of the real value should be added.

Functions are either requests to *sum* a range of cells or to *avg* (average) a range of cells. These functions are only allowed on *Real* values. Each function takes a cell range as a single argument and respectively returns the sum or average of the cells given. For example, `sum(A1-A5)` and `avg(A1-A5)` (with no spaces, as shown) evaluate to the sum and average respectively or the given cell range.

Cell References are values that are retrieved from other cells upon computation and evaluation of an expression. Cell references are simple cell identifiers used as parts of expressions. When such an expression is assigned to a cell, that cell must be recomputed whenever any of the cells it references are modified (and those in turn must be recomputed whenever the cells they reference are modfied, and so on). Circular references cannot be computed, and thus are disallowed – it is erroneous for an input to create a circular reference.

3 Commands

Input will be given to your application using standard input, either by a human (you, your reader, ...) or an automatic testing program. Blank lines should be ignored. The commands are as follows:

`#`	indicates a comment. The `#` character and everything on the same line of input is ignored. The `#` must be at the beginning of the line.
Cell `=` *Expression*	indicates an assignment to a cell. The expression *Expression* is assigned to the cell. *Cell* is a single cell identifier.
`display` *Window*	indicates a request to display the given cell window. If only one cell is being displayed, the window is simply a single cell identifier. If no window is specified, the smallest possible window that includes all non-empty cells should be displayed.
`clear` *Window*	indicates a request to clear the given cell window, or all cells if no window is specified. If only one cell is being cleared, the window is simply a single cell identifier. Cells that are cleared are set to type *Empty*.
`sort` *Ordering Window* `by` *Range*	indicates a request to sort the elements in the given cell window using the cells in the given cell range as keys to sort by. Ordering is either `ascending` or `descending`. The sorting should be stable (items with the same value should retain their original ordering).
`quit`	quits the application.

4 Output

Output from the display command must adhere to a strict format. Column headings are to be on the first line (shifted one column to the right) and row headings in the first column for each subsequent line (much like in a graphical spreadsheet). Each cell has a fixed width – either 10 characters or the width of the widest row or column heading, whichever is greater. There is a 1 space gap between columns. All cell values and headers are to be left justified. Cell values that are longer than the fixed width computed are to be truncated to the maximum width. Examples of this output format are shown in the examples section.

5 Getting Started

The directory `~cs61b/hw/proj1/` contains some files you should use as a framework for this project. Copy them into a new `proj1` directory in your home directory. Use the makefile provided to compile and test your project. We have included a few test cases for you, but you need to add your own to test your program more thoroughly.

Input to your program will, in part, come from fallible humans, who may input improper commands or other erroneous data. Therefore, your program should be able to handle improper input and not simply crash. After erroneous input, the internal state of your program is up to you (you may save the value of the operation), but you should output the word `ERROR` followed by a reasonable error message (of your choice) on one line. Your choice of recovery is less important than being sure that it actually does recover. After an error, your program should still be usable (it should continue to accept input and process commands as usual).

Our testing of your projects (but not our grading!) will be automated. The testing program will be finicky, so be sure that:

- Your makefile is set up to compile your project with the command `gmake` and to run with all your tests with the command `gmake test`. The makefile provided to you already does this, but you will need to modify it if you add more files or tests.
- Your main function must be in a class called `invisicalc`, as it is in the provided files.
- The first line of output of your program identifies the program. It may contain anything.
- Before reading the first command and on reading each subsequent end-of-line, your program must print the prompt '`>␣`' (greater than followed by a blank).
- Your program should exit immediately upon a `quit` command or an end-of-file in the input.
- Your submission should not produce any debugging output – it may confuse the automated tester.

6 Advice

In doing this project you will find many useful classes in the Java standard libraries. In particular, you should look at `java.util.Calendar`, `java.math.BigDecimal`, and various classes in `java.util`. You may also, depending on your implementation, find various methods in `java.util.Collections` or `java.util.Arrays` to be useful.

In addition, there are several (perhaps subtle) parts of this project that you should think about carefully:

- How cells and their identifiers are stored
- How cell references are stored, evaluated, and checked for circularity
- How to store possibly infinite cell-space (due to unrestricted cell identifiers)

- How to process expressions
- How to handle cell windows and ranges, especially during output

7 Examples

User input is underlined.

```
$ java invisicalc
Invisicalc, Version 0.99
> # This is a comment
> A7 = 5
> B4 = 3 + A7
> A5 = "antidisestablishmentarianism"
> display
          A          B
4                    8
5         antidisest
6
7         5

> quit
```

```
$ java invisicalc
Invisicalc, Version 0.99
> YUK1 = 0
> YUK2 = 0
> YUK3 = YUK1 + YUK2
> YUK4 = YUK2 + YUK3
> YUK5 = YUK3 + YUK4
> YUK1 = 1
> YUK2 = 1
> YUL5 = sum(YUK1-YUK5)
> display
          YUK        YUL
1         1
2         1
3         2
4         3
5         5          12

> quit
```

```
$ java invisicalc
Invisicalc, Version 0.99
> ABRACADABRA99 = 2002-01-01
> display
           ABRACADABRA
99         2002-01-01

> quit
```