Final Project Description

Read all of this document. If you do not read these instructions you may lose points for failing to follow requirements.

For your final project you will be writing a very basic text spreadsheet with cells that store strings, numbers, and functions whose result is determined by the data in other cells. You will be provided with several testing scripts to help you test your program. 85% of your grade will be based upon the result of those testing scripts, 5% will be based upon proper memory destruction with delete, and 10% of your grade will be based upon your code quality. Code quality means that your code is nicely formatted and indented, you have comments before functions, loops, if statements, variable declarations, etc.

If your program does not pass all of the test scripts then you can schedule time after the due date for 10 minute in-person code review. We will go over your work and you can receive partial credit for the features that you did not finish implementing.

1 Starting the Program

Your program will be invoked with two arguments specifying the starting width (argv[1]) and height (argv[2]) of the spreadsheet. If both arguments are not positive numbers then the program prints an error message and quits with a return value of 1.

If your program was named final.exe then this would run the program with a spreadsheet with 2 columns and 5 rows:

final.exe 2 5

2 Interactive Commands

Once running, your program should support several commands to set the contents of cells, add rows, and delete rows. To simplify the data structures and operations and you will need to use, this project will only have row operations.

2.1 Adding or Removing Rows and Columns

The commands to add a row is:

addrow x

This command inserts the new row before position x, so to add a row before the first row the user would type:

addrow 0

The command to remove the row at position x is:

removerow x

Whenever you add a new row it should have the same number of entries as the current number of columns. If the user attempts to add a row at an index that is outside of the range of the spreadsheet, or the user attempts to delete a row outside of the range of the spreadsheet, print out the message "Error: row out of range" using std::cout and do not alter the spreadsheet.

2.2 Setting Cell Contents

The following command sets the contents of the cell at column x, row y:

set x y <type> <contents>

0, 0 is the upper left cell. If the user specifies a cell that does not exist then print out the error message: "Error: cell out of range" using std::cout and do not alter the spreadsheet.

The contents of a cell may be a string, a number, or a function. The program will determine which one is appropriate based the type string that the user provides. The program will interpret the type string as follows:

If type is "number" then interpret the rest of the string as a single double value. For example, the user could set the cell at column 2, row 3 to 3.14 with the following command:

set 2 3 number 3.14

If the input value is non a number then you should print out the error message "Error: Bad input for set number" using std::cout and leave the spreadsheet unchanged. You can detect if reading from a stream into a number succeeded with code like this:

```
if (in_stream >> x) {
  //number was read successfully
}
else {
  //Input was not a number, ignore it
  std::cout<<"Error: Bad input for set number\n";
```

```
}
```

If the type string is "string" then the contents string should be interpreted as a string and the contents of the cell should be set to the entire user provided contents string. For example, the user could set the cell at column 2, row 3 to "this is a string" with this:

set 2 3 string this is a string

If type string is "mean", "min", or "max" then the cell is a function cell. A function cell will calculate a certain function across a range of cells; for simplicity this range will only be across rows, not columns. Since the range will be across a row, the next value specifies the index of the row. The next two words are numbers specifying the beginning and end (inclusive) offset into the row or column. To set a cell's contents equal to the mean of cells from row 0 offset 1 to row 0 offset 3 the user could type the command:

set 2 3 mean 0 1 3

Function cells will display as a number unless there is an error in the user's command; for example if the user specified a negative range, a range outside of the spreadsheet, or did not provide enough arguments. In that case, the program will print out the error message "Error: function incorrect" and will not modify the contents of the cell.

The results of function cells should update if the number in a cell is updated. Function cells do not treat strings as numbers, and should just ignore non-numeric cells.

2.3 The Print Command

The print command shows the cells of the spreadsheet. The name of this command is simply "print" and will cause your program to print the spreadsheet using std::cout one row per line with commas and white space characters in between cells in the same row. Print out an extra new line after the last row so that the user's next command is on a line of its own.

2.4 Quitting

If the user enters the command "quit" then the program should exit with return code 0. All memory allocated with the *new* operator must be deleted with the *delete* operator.

3 Implementation Details

3.1 Processing Input

Your program should process data one line at a time, meaning that a full line should be consumed whenever the user types enter. The simplest way to do this is by reading an entire line into a string and then converting that string into an input stream (an input stream can be used like std::cin). This guarantees that the entire line is always consumed so that an error in the user's input on one line does not affect the user's next line of code.

```
//sstream has the std::istringstream class
#include <sstream>
//String has std::getline and std::string
#include <string>
//Always consume all input on a line
std::string line;

int main(int argc, char** argv) {
        ...
        //Read in the entire input line
        std::string line;
        while (std::getline(std::cin, line)) {
                //Convert the string into an input stream and read in the command
                std::istringstream ss(line);
                std::string command;
                ss >> command;
                //Process the command
                ...
        }
}
```

If an unrecognized command is entered the program prints out the error "Error: unknown command". The program will consume any other input on that line.

3.2 Polymorphic Cells

Your cell class should be polymorphic so that you spreadsheet can be represented as a list of list of cells: LList<LList<Cell*>>. Every cell must have a way to represent its contents as a string, so it makes sense for the base class to have a function that returns a string as a virtual function. The Cell class could be as simple as this:

```
class Cell {
        public:
                //String representation of data in this cell
                virtual std::string toString();
};
```

The base Cell class should return an empty string when printed.

The numeric cell should store values as double variables. The function cells are most easily created by inheriting from the numeric cell class since the function cells must hold their results as doubles as well. Function cells should also remember which function they represent and which cells they use to calculate their values so that they can update their value if those cells change.

3.3 Function Cells

The function cells are probably the most difficult part of this project. They must derive their value from the values of other cells and should update their value when the spreadsheet is modified. I suggest that you break down this functionality into several parts.

- First, function cells should remember which cell they are supposed to calculate their value from.
- Second, the function cell should have an update function that accepts begin and end iterators. This will allow you to write code inside of the function cell class that is independent of all of the spredsheet code.
- Third, inside of the update function write code to use the remembered locations to find iterators to the cells that the function cell uses to calculate its value.
- Finally, write functions for min, max, and mean that accept two LList<Cell*>::iterator values.

If you follow those steps then your class functionality will be neatly decoupled from the rest of the program, and the functions will be easier to think about. That being said, there is one more important detail about function cells: what happens when the cells they are using to calculate a value are all empty or only contain strings? In this case the function cells should store a special value indicating that they do not have a number. That value an be obtained as follows:

```
#include <limits>
double m = std::numeric_limits<double>::quiet_NaN();
```

3.4 Non-numeric Values in argv

You have two options to deal with this. Either you can use atoi and check if the result is 0 (calling atoi(argv[1]) when the first argument is a string will return 0) or you can do exception handling like this:

```
//Initialize the width and height of the spreadsheet
int width;
int height;
try {
  //Attempt to set width and height from the user provided arguments
  width = std::stoi(std::string(argv[1]));
  height = std::stoi(std::string(argv[2]));
}
catch (std::exception e) {
  //Handle failure of the stoi function
  std::cout<<"Got bad values for width and height!\n";
  return 1;
}
```

4 Scoring Rubrics

You *must* use a version of the LList class placed on sakai and implemented in class. You may not use any other containers. You should store a linked list of linked lists of Cell pointers, e.g. LList<LList<Cell*>>, to represent the spreadsheet. Cell must be a polymorphic class, and you must inherit from it to implement the string, numeric, and function cells. Numeric cell values should be represented as doubles.

10% Program runs and the "quit" command works. Bad arguments detected.

10% An empty spreadsheet of the specified size is created at startup. The "print" command works, bad commands are ignored.

10% "set x y string" command works. Strings can be placed into the spreadsheet and printed.

10% "set x y number" command works. Numbers can be placed into the spreadsheet and printed.

10% "addrow" command works. New rows are created and existing cells retain their values.

10% "removerow" command works. Rows can be deleted and cells in other rows remain unaffected.

5% "min" function cells work.

5% "max" function cells work.

5% "mean" function cells work.

10% Function cells update their values when the appropriate non-string cells change their values.

5% Any memory that was allocated is deleted. This will be checked manually.

10% Code quality: formatting, readability, and comments.

The following can be implemented for extra credit *only if all features for regular credit are complete*:

5% Function cells continue using the same cells after addrow commands move their position or the position of the row the function cell's value is based upon.

5% Function cells continue using the same cells after removerow commands move their position or the position of the row the function cell's value is based upon. If the row being used for source values is deleted then the function cell's value becomes NaN (not a number).

5 Example Program Execution

```
>Project4.exe 3 3
print
,       ,
,       ,
,       ,
set 0 0 test!
Error: unknown command
set 0 0 string test 2!
print
 test 2!,       ,
,       ,
,       ,
set 0 1 number 3.14159
print
 test 2!,       ,
3.141590,       ,
,       ,
addrow 2
print
 test 2!,       ,
3.141590,       ,
,       ,
,       ,
removerow 2
print
 test 2!,       ,
3.141590,       ,
,       ,
set 0 2 min 1 0 2
print
 test 2!,       ,
3.141590,       ,
3.141590,       ,
set 1 1 number 1
print
 test 2!,       ,
3.141590,       1.000000,
1.000000,       ,
```

```
set 1 2 max 0 0 2
print
 test 2!,       ,
3.141590,       1.000000,
1.000000,       1.#QNAN0,
set 1 2 max 1 0 2
print
 test 2!,       ,
3.141590,       1.000000,
1.000000,       3.141590,
set 2 2 mean 2 0 1
print
 test 2!,       ,
3.141590,       1.000000,
1.000000,       3.141590,       2.070795
set 2 1 number 0.5
print
 test 2!,       ,
3.141590,       1.000000,       0.500000
0.500000,       3.141590,       1.820795
quit
```

6 Changelog

November 19, 2014 Fixed some garbage characters where LList<LList<Cell*>> should appear. Also made the first two goals more clear. Added explanation of detecting numeric command line arguments (Section 3.4). Added instruction about how to handle incorrect input for the "set x y number" command.