CS3016/4012 Project 2011

Andrew Butterfield and Glenn Strong

November 18, 2011

Contents

1 Introduction

- Objective: implement a program that reads a specific spreadsheet data file and allows the user to make queries and fix up the data.
- Deadline: Monday, December 19, 2011

The program purpose is to enable a user to query and clean-up a spreadsheet, made available using the comma-separated values format (CSV). The specific spreadsheet of interest is one that contains data about maps, and the program must work satisfactorily with this data.
The program will have a command-line interface, so the user can: load spreadsheet files; perform simple queries; invoke some data cleanup; save cleaned-up files; and exit the program.

1.1 Challenges

Key challenges include:

- Being able to handle data entered in fields using a variety of ad-hoc styles, that are human readable, but tricky for machines to handle. Decisions to be made will include what the "official" format should be, and what to do about missing or loosely defined data, e.g., how does something like "summer 2006" get turned into a standardised data format?
- The management of program state, and how this interacts with the way Haskell manages I/O state (i.e., the `IO` monad).
- The internal representation of the spreadsheet once parsed:

  `[[String]]` each field is represented as a `String`.

  `[[Field]]` each field is represented as a `data` type variant

  `[Record]` each row is represented as a bespoke record, tailored for the specific spreadsheet being handled.

  `[Map String Field]` each row is a mapping(?) from column names to `Field` values.

  ... or whatever.
- SS students only: How the program can be implemented in a generic fashion, with configuration data, so that it can be used with other spreadsheets with different kinds of data. Note that this is closely tied to the decisions made as part of the previous challenge.

Note: *we reserve the right to use plagiarism detection tools to check submitted work.*

2 Requirements

2.1 Requirements List

CLI Command Line Interface — an interactive command line interface must be implemented, with a facility to exit the program (A.8).

SYN Command Syntax — The commands (syntax and function) are described in A, and the syntax as presented must be the one implemented.
(SS students only) The command parser will allow users to type command and arguments abbreviations where possible (e.q. `q` should be enough for `quit`).

CSV CSV Parsing/Rendering — The program will be capable of parsing and writing CSV format files

QRY Queries — The program will support the queries as described in A.2, A.3.
(SS students only) The program will support query output redirection (A.4).

UPD Updates — The program will support the updates as described in A.5, A.6, A.7.

ERR Error Handling — error handling should be implemented so that nothing the user types can cause the program to crash.
(SS students only) Haskell Exception Handling should be used.

HLP Help — the user should be able to get useful help (A.8).

2.2 Command List

- `load` — load CSV spreadsheet
- `save` — save spreadsheet as CSV
- `report` — run builtin report
- `count` — count records satisfying a condition
- `list` — show records satisfying a condition
- `distinct` — report distinct items in a column
- `output` — redirect output to file
- `nooutput` — redirect output to console
- `date-fix` — fix date data
- `grid-fix` — fix grid-reference data
- `reformat` — reformat column data
- `sort` — sort spreadsheet
- `select` — select sheet rows
- `show` — show selected rows
- `update` — update field
- `delete` — delete row
- `insert` — insert new row
- `help` — help
- `quit` — exit the program

A Command syntax

Commands take the general form: `keyword arg1 arg2 arg3 ...` ⟨*newline*⟩
Arguments are separated by spaces, and must be quoted if they contain spaces, using the same quotation rules as the CSV format. When a command can determine that a required argument is missing it should prompt for the relevant data.

A.1 File manipulation

- `load` - one required argument, which is a filename to be read. The contents will become the current record set. The first line in the file is assumed to contain the column names, the following lines are the records.
- `save` - one required argument, the filename to be written. Contents will be replaced with the current record set. The column names are written as the first line, followed by the records.

Example:

```
> load "map-register.csv"
1 header line (10 named fields), 85 records
skipped 14 blank records
>
```

A.2 Built in reports

- `report` followed by the name of one of the built-in reports from the following list:
  - `registrations` - reports the number of maps each club has registered
  - `completions` - reports the number of maps which should be complete at the current date

Examples

```
> report registrations
3Roc, 2
AJAX, 3
BOC, 32
...
> report completions
Bull Island
Featherbeds
...
```

A.3 General queries

- `count` followed by a list of conditions will show the number of records which meet all the conditions
- `list` followed by a list of conditions will show the records that match all the conditions
- `distinct` followed by a single column will report the number of unique values contained in that column

  A column is specified by the column name (as found on the first line of the input file); quotation marks are required around the name if it contains spaces and are optional otherwise. As a convenience when dealing with very long column names the user may also specify columns in the form `$1`, `$2` and so on to (to mean the first column, the second, and so on)[1]

  A condition consists of a column name, an equality symbol, and a quoted string. For the condition to match the contents of the string must match the contents of the column. Glob style matching is used, so the symbols `*` (to match zero or more of any character) and `?` to match any single character are permitted.

  Where several conditions are desired they should be listed separated by spaces; all conditions must be true for a record to be included in the result.

(3b) generic ones like "how many different values" in this column

```
> distinct $3
81 distinct values for "Nearest Town"
> count $3="*Cork*" $1="*B*"
14 values for "Nearest Town"="*Cork*" "Club"="*B*"
```

A.4 Saving query results

(SS students only) By default the result of a query are displayed at the console. To save the results of a query to a file the user issues the `output` command:

- `output` - followed by a filename. The results of all subsequent queries are stored in this file
- `nooutput` - re-directs the output of subsequent queries to the console.

```
> set output "club-regs.txt"
output file set
> report registrations
14 records written
> nooutput
output directed to console
>
```

[1] We assume that no such name appears in the input file; the form |$1— is always interpreted as referring to columns by number

Subsequent commands allow the user to modify the spreadsheet data, so a `save` command needs to be provided:

- `save` - followed by a filename. The current spreadsheet is written in CSV format to the specified file.

A.5 Transformations

Data in the input files are often in a mix of formats. Specifically, individual records may differ as to the formatting of dates, grid references, and so on. This section contains some simple commands to allow the user to specify how the data should be written out in a uniform way. Once these commands are issued the formatting should be applied both to the results of queries and to the data written out by the `save` command.

- `date-fix` followed by a column and a date format will rewrite the contents of that column accordingly. It is an error to apply this to a column which does not contain dates (as determined during the initial loading of the file)
- `grid-fix` followed by a column and a grid format will rewrite the contents of the column accordingly. It is an error to apply this to a column which does not contain a grid reference (as determined during the initial loading of the file)
- `reformat` followed by a column name and one of the following reformatting instructions will rewrite the contents of the column accordingly.
  - `uppercase` - convert text to uppercase
  - `capitalize` - capitalize the first letter of each word, convert others to lowercase
  - `lowercase` - convert text to lowercase
  - `trim` - remove excess whitespace from the column

Date formats are strings as understood by the Haskell `Data.Time.TimeFormat.formatTime` function; however, the strings should be checked to ensure that they contain exactly: year, month, and day information (our time data is accurate only to the day, and the new formats must not discard any information).
Grid formats consist of the number `4` or the number `6`, to indicate the two standard formats (4-digit or 6-digit with a leading letter). Fields that were incorrect in the input (e.g. 6-digit with trailing letter) should be rewritten correctly during this transformation where feasible[2].

```
> date-fix $7 "YYYY-MM-DD"
84 records adjusted
> reformat-dates $7 "YYYY"
Not applying, format too simple
> reformat $1 uppercase
4 records adjusted
```

[2] i.e. when a suitable parse can be found

A.6 Sorting

Setting the sort order for output (once set this should apply both to queries and to the contents of saved files).

- `sort` followed by a list of field names, and orderings. An ordering is either the word `ascending` or `descending`.

```
> order by $7 ascending $1 descending $2 descending $5 ascending
84 records resorted
>
```

A.7 User editing

The penultimate set of commands allow the user to edit the contents of individual cells. The user needs to specify a single row and column, and supply replacement contents. To simplify this we introduce the notion of a *selection*

- `select` followed by a list of conditions (as described in the section on queries), or the word `all` (which selects all rows)
- `show` prints out the currently selected rows in a user friendly format. Each row is preceded by the row number, columns should be formatted and include the column header.

To edit a cell the `update` command is used

- `update` followed by a row number, then a column specification (as described in the section on queries) and a new value to be stored in the column

example:

```
> select $1="AJA*"
2 Records selected
> show
   Club   Map Name       Nearest Town        Terrain        Map grade      ...
3  AJAX   Bull Island    Clontarf Dublin     Sand dunes     championship   ...
4  AJAX   Carrickshouk   Laragh Co. Wicklow  forest         championship   ...
5  AJAX   Featherbeds    South Dublin        open mountain                 ...
> update 5 "Map grade" "championship"
1 cell updated
>
```

In addition we have facilities to delete and insert rows:

- `delete` followed by a row number deletes that row.
- `insert` followed by entries of the form `$1="..." $2="..."` inserts a new row with the specified data, prompting for missing fields.

A.8 Quit and Help

The final set of commands allow the user get help and quit the program

- `help` optionally followed by a command should give appropriate help, as shoud the command `?`.
- `quit` exits the program, but should prompt the user if there is a modified spreadsheet that has not been saved.