Rolodex Program Reference Guide

A Reference Guide written by Project Group 1

David Allen, R. Scott Hosley II,
Myndert Papenhuyzen, William Warren

This document serves as the reference guide for the Rolodex Program, a program designed to parse and store contact information in a quick and organized fashion. This reference guide has sections providing an overview structure and program flow, a look at the various modules interactions of the program, followed by a detailed specification of what the program expects for input and produces as output.

Upon compilation using `make`, the program can be invoked from the command line as

```
Aprog <input_file_name> <output_file_name>
```

Overall Structure and Program Flow

Once the user specifies the input and output files in the command line, the program performs all execution without user interaction. See the "Input & Output Specifications" section later in the guide for further information regarding the I/O expectations.

The overall structure is laid out below. Further elaboration will be provided in the "Modules" section of this reference guide.

Beginning Execution/Getting Data

At the beginning of the program's execution the necessary variables are initiated and a check in the `main` function to validate the input and output file parameters is executed. If invalid, an error is thrown and the program exits. If valid, the data is streamed from the file into a `Rolo` object with the function `readRolo`.

Parsing Data

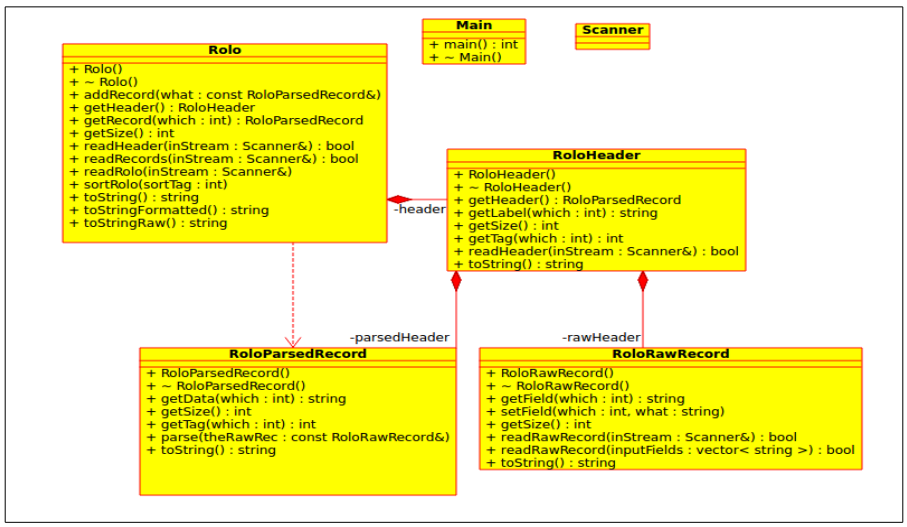
Once inside the `readRolo` function, the data is split into either the `readHeader` or `readRecords` functions of `Rolo`. This is where the program determines what is header information and what is record information as specified in the "Input/Output Specification" section. Once header information is inserted into the `readHeader` function it is passed into a `RoloHeader` object with its own `readHeader` function. Here the header data gets parsed by a `RoloRawRecord` object with the `readRawRecord` function and by a `RoloParsedRecord` object with the `parse` function and the tags are extracted and returned for sorting. Conversely, once the record information is inserted into the `readRecords` function it, too, is parsed by `RoloRawRecord` and `RoloParsedRecord` objects and is stored in a vector for sorting.

Sorting Data

The next step in `main` is calling the `sortRolo` function of the `Rolo` object and passing in an integer value that indicates which field to sort by. For the `sortRolo` function, a bubblesort algorithm is implemented that iterates over the vector of `RoloParsedRecord` objects and sorts them based on the appropriate field in ascending order.

Outputting Data/Ending Execution

After the parsing and sorting of the data the final step is the output. In `main`, the output file is written using an overloaded operator for the `Rolo` object. All of the classes have overloaded the stream output operator, `<<`, for their `toString` functions.



This is the UML Diagram for all of the classes in this program, showing how they interract and otherwise fit together.

Program Modules

As indicated in the description of the overall structure and the illustration of the UML, this program is divided up into four major modules: the `Rolo` class, the `RoloHeader` class, the `RoloRawRecord` class, and the `RoloParsedRecord` class.

Rolo Class

The `Rolo` class is the primary worker of this program, the “Do the Work” class as Professor Buell would say. It is in charge of delegating the input data to the appropriate classes so that the data can be parsed and read. Furthermore, this class sorts the data and is responsible for generating the program's output.

Primarily, the `Rolo` class makes use of the `vector` from C++'s Standard Template Library to store the list of `RoloParsedRecords` that hold the majority of the data. Additionally, the `Rolo`

class also directly stores an instance of the `RoloHeader` class to store the tag header information associated with the data.

RoloHeader Class

The `RoloHeader` class is used to store information pertaining to the header used to organize the record data. A header simply contains a list of tag identifiers and the associated descriptive labels to give meaning to data stored in a `RoloParsedRecord`. In fact, a `RoloHeader` instance is merely a special record that stores the tag information, making use of the `RoloRawRecord` and `RoloParsedRecord` classes to do so.

The Record Classes

In implementation, the `RoloRawRecord` class is merely used as a utility to initially read and store record information, as such, it only provide facilities for reading data from the input file. It makes use of a string vector to store the provided data for later processing by the `RoloParsedRecord` class.

Unlike the `RoloRawRecord` class, the `RoloParsedRecord` class meaningfully stores the record data. While it does not read any data, it is responsible for parsing a raw record, thereby interpreting each line to associate header information to the data through the use of an internally-defined field structure that stores the associated tag identifier and the data string.

Input & Output Specifications

The Rolodex Program requires only one input file and produces only one output file; these files are specified as command line arguments.

```
%00 Name
%02 HAdd2
%01 HAdd1
%04 WAddr
%06 WNum
%07 MNum
%03 HAdd3
%08 Email
%09 Notes
%05 HNum

%00 Spears, Britney
%02 Mandeville LA 70471
%01 70 Doublewide Lane
%05 985.432.1234

%00 Vitter, David
%01 P.O. Box 1234
%02 Mandeville LA 70470
%05 504.626.1010
%06 504.626.2020

%01 1 Hotsy Totsy Road
%02 Mandeville LA 70471
%00 Hotsy Totsy
%05 985.444.5555

```

A sample input file for the program. The header can be seen as the list of tags and labels in the first block. The blocks of text beneath the header are the various records.

Input File

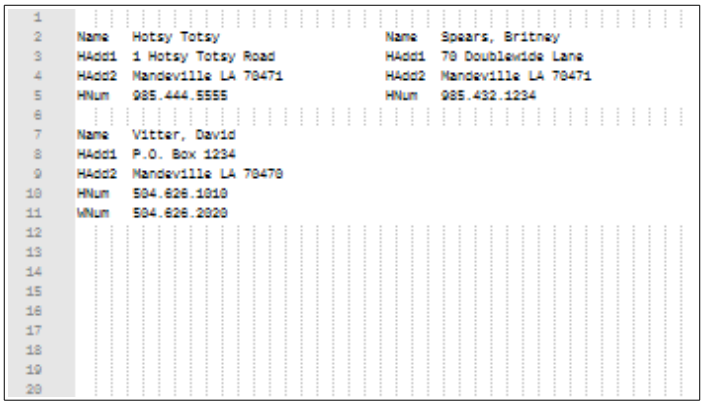
The input file can become complicated for large collections of data, but boils down to only a few simple guidelines. The file consists of at least two blocks of text, each separated by a blank line; no line of text within a block can be blank.

The first block of text in the file is assumed to be the header for the Rolodex records. The header defines the list of tags that will be used to describe every data point in the records. Each line in the header represents a tag and begins with a % symbol that is followed by a unique two-

digit identifier and the tag's label. These identifiers need not follow in sequential order, but doing makes the sorted results easier to predict.

All subsequent blocks in the input file are the records to be placed into the Rolodex. These blocks consist of a variable number of tags and their corresponding data. Once again, the tags do not have to be provided in sequential order with respect to their tag identifier; however, doing so may make the sorted results easier to predict. Also, if a block of data contains multiple tags with the same identifier, then the data stored in the program will be the data of the last tag with that identifier.

As an important note: the program's behavior is undefined in cases where the input is improperly formatted. Most of the time, however, the program will just produce garbled output resulting from incorrectly separated records.



A sample output file with three records, sorted by the data in the second line. Only the first twenty rows of output are shown, with vertical lines drawn to visualize the empty lines.

Output File

Under normal execution, the program produces an output file that contains the sorted list of all of the records provided to it by the user printed out as two columns, with the record order going left to right along a row. Be aware that the individual record lines are truncated to be 38 characters long, losing any text that exceeds the limit.

The output is generally split into pages consisting of 55 lines each with a form feed character separating pages. Although the number of records per row is set at two, there is a variable number of rows per page since the height of a row is dynamically set to be high enough to fit all of the data lines of the record with the most data. There is a check implemented in the program so that if the output of either record on a line were to exceed the 55-line limit, both records' output gets pushed onto the next page.

In addition to the output file, the program may also produce logging information that is prints out to the console window. This information is provided primarily for debugging purposes and can generally be ignored during execution.

Summary, in Short

As an implementation of a Rolodex, this program is tasked with receiving structured contact information from the user, then reading and parsing that data into records along with the header information used to organize the data within them. It can then sort the records and print out the resulting list to an output file, whose location is also to be provided by the user.