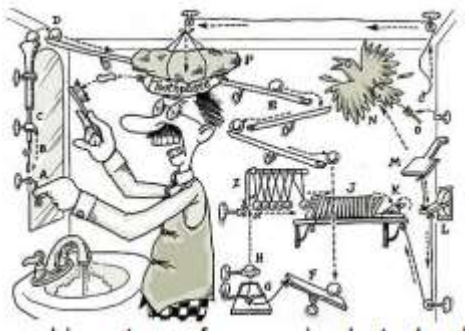
Introduction:

At this point you are familiar with Linked Lists, Queues, Stacks, and Trees in detail (you can also explore Hash Tables). This exercise is designed to give you some practical, hands-on practice in combining a few of these data structures.

What is a Rube Goldberg Machine?

In this assignment you will build a virtual Rube Goldberg machine to practice with some of the data structures we discussed in class thus far. A Rube Goldberg Machine is a complex device that performs simple tasks in indirect and convoluted ways Reuben Goldberg was an American engineer who changed his career to cartoonist. He was famous for his political cartoons and satire. However, the works which would lead to his lasting fame involved a character named Professor Lucifer Gorgonzola Butts shown in the cartoon on the left. In this series, Goldberg drew absurd inventions that would later bear his name: Rube Goldberg Machines. Rube Goldberg received the Pulitzer prize for his satire in 1948. Today there are several contests around the world known as Rube Goldberg contests, which challenge high school students to make complex machines to perform a simple tasks. These contests are not only fun, but they challenge students to practice fundamental engineering, physics, and mechanical principles as well as encourage creativity. They must utilize a specified number of various simple machines, using common junk and household items to perform absurdly simple tasks such as cracking open and egg, opening a door, ringing a bell, and so forth. You can check out a real Rube Goldberg machine at http://www.flixxy.com/best-rubegoldberg-machine.htm. Enjoy it!



Assignment Description:

Your job will be to create a virtual Rube Goldberg Machine with ADTs. The ADTs that you will use will include the queue, stack, binary tree, heap, and many of the associated operations on these ADTs. One way to approach this assignment is to think about the functionality from the user's perspective. Certainly this is a reasonable place to start, however you should think in terms of the application being composed of ADTs and operations upon them. Note that the ADTs will contain the data and with each ADT we studied there were lots of operations upon them. Think as if you were a small team and that you had a few engineers creating the ADTs, and others that would compose the application using these ADTs. Remember, "I don't want to know how to build a color TV set, I want to watch FRIENDS." This is the mind set we should have as the engineer responsible for composing the application from the ADTs. Those building the ADTs should worry about the details of the ADT structures and operations in isolation from other ADTs and in isolation from application specifics as much as possible. There should be as little dependency between the ADTs and the application as possible – you will have to use your judgment. In this assignment, you will play the role of ADT builder and application builder.

Your Virtual Rube Goldberg machine will do the following:

• You will read from a provided space-and-comma-delimited text file list of data in the following format:

- first name (30 characters)
- last name (30 characters)
- age

- day of birth (mm-dd-yy)

An example would be as follows: Mallika David, 49, 1960

• Your program should support any number of entries. As the data is read from a file, initially store the data in a queue.

• Dequeue each element from the queue, print each item, and requeue each item. Ask the user to press any key to continue the processing.

• Next, you will reverse the order of the data in the queue by dequeuing each element and pushing them onto a stack. Once all the data is dequeued from the queue and pushed on to the stack, pop off each element of the stack and re-queue each element back into a queue ADT, reversing their order in the queue. Once completed, dequeue each element from the queue and print each item for the user - ask the user to press any key to continue the processing. Make sure you requeue each item to preserve the queue.

• Next, dequeue the elements from the queue and place them into an unordered binary tree. Add the items into the tree ADT in the order they are dequeued, adhering to the binary tree shape property.

• Print the contents of the tree in pre-order and ask the user to press any key to continue the processing. Print the contents of the tree in post-order and ask the user to press any key to continue the processing. Ask the user to press any key to continue the processing.

• Move the data from the unordered binary tree into a linked list ADT using an in-order traversal of the tree. Print the contents of the list and ask the user to press any key to continue the processing.

• Sort the contents of the list using a quick sort in ascending order. Print the contents of the list and ask the user to press any key to continue the processing.

• Allow the user to interactively enter another name, age, and birthday. Add this to the list in the proper location to maintain the sorted order. Print the contents of the list and ask the user to press any key to continue the processing.

• At this point you are done with the processing.

• You will use C or C++ for this assignment – you may use any compiler you like. Please do not use any prepackaged, open-source, or commercial libraries for the list, stack, queue, or tree ADTS. Maintain a clean separation of these ADTs. Make sure that you handle all possible errors within the ADTs and the end-user application. Your code MUST BE WELL STRUCTURED AND DOCUMENTED! Ensure that you include headers and comments in your code, properly indent code blocks, and otherwise use good coding practices. If we can't read and understand your code, we will not grade it.

Of course your program must work and it must be easy to follow what the program is doing. The user must be given simple, easy to follow feedback in order to follow the progress of the operation. You will also be graded on

- how well structured your ADTs are,
- how well you have maintained a clean separation of concerns between the various ADTs and the application itself, and
- how the overall structure of the program.

Note that a working program alone is not enough to get a good grade on this assignment – structure and separation of concerns and code quality matter as much as functionality.

What to Turn In:

Post your source code and executable code (.c and .exe) on GCR– note that we will not recompile programs. We will not install any special environments to run your code. We expect to run your code from a simple command window and observe the operation. All I/O from the user to the application shall be from the keyboard. We will run your program and observe its operation – if we cannot run your program, you will lose points depending upon the reason for the problem. Assume that the target OS platform is the Windows XP or better.

Deliverables:

1. Report that includes:
   • Describe the design of your various ADTs, the data structures you selected, what information and operations you hid and your reasoning for your decisions.
   • Describe how to use your program.
   • Show the analysis of your application in terms of correctness and complexity (performance and memory) as described in class. You may utilize any of the notation and analysis techniques presented in the class.
2. Demonstration program.
   NOTE: Please be thorough, but as concise as possible in your discussion. Note that grammar, good style, and format counts. Any reasonable formatting style/structure is acceptable.

   Provide your source code, data files, and executable program.

   Put all of these materials (described above) into a zip file and turn it in electronically in GCR. Use Windows file compression or Winzip to compress your files. Include a "readme" file in your archive that explains the contents of your archive. Name your archive as follows: A3_TeamName.

   Grading: This assignment is worth 50 points as follows:
   - Report content: 20 points total for content.
   - Implementation: Correctness of the application for a total of 20 points. Note the design and structure of your API, application, and how well it matches your design is an important part of the evaluation.
   - Ease of use and write-up quality: Your application must be easy to use – easy to enter input, easy to read and understand your output (5 points). The write-up will also be evaluated in terms of grammar, format, and readability will also be considered (5 points).