CPE-2600 FALL 2024

Systems Programming

Lab 7 Vector Calculator Updates

Introduction

The purpose of this assignment is to update our vector calculator program with additional features.

Work on the assignment is to be done ***individually***. You are welcome to collaborate with class members, but the project must be your own work.

Project Description

This project will add two features to your vector calculator program - the ability to load and save vectors to and from files, and the ability to store “unlimited” vectors through the use of dynamic memory.

Dynamic Memory

The initial specification called for a hardcoded-limit of storing 10 vectors. This would be accomplished by simply having an array with 10 structures and those structures would be assigned as needed. Of course, this is quite limiting. You would have a hard time predicting how many vectors future users of the program might actually use. Guessing too few would limit the usefulness of the program. Guessing too high would be excess memory usage and may limit performance. The solution is, of course, to allocate memory as needed - aka dynamic memory. So, add dynamic storage to your program. You may accomplish this in a variety of fashions. Both examples in lecture may work well.

The first example in lecture depicted a dynamic array. You would usually start with a rather small array (dynamically allocated), then, when that array is full, reallocate the space to increase the size of the allocated block of memory. Of course, you have to copy over the data to the new array before deallocating the old space. You can do that manually or let `realloc()` do that for you.

The second example in lecture was a linked list. With this technique new memory is allocated with each and every new vector that your program needs to store. Each new “node” is linked to the previous one as each node is not necessarily stored in continuous memory.

From a data-structures standpoint, each technique may have benefits over the other. For our program, I do not think there is a compelling reason to pick one over the other. I

suspect the dynamic array might be a little easier to implement than a linked list, especially since your initial version used a fixed array. However, the choice is yours.

In any case, adding this feature should not change the outward appearance of your program at all. Instead of having the limit of 10 vectors, your program should happily continue to store vectors until heap memory is exhausted.

If you chose to "name" vectors with a single character name in your previous version, you may be limited to 52 vectors total (A-Z,a-z). No need to change the naming scheme and your implementation will be allowed to have that limitation.

File IO

We saw a variety of examples of file input/output in lecture with the main dichotomy being text vs binary files. A text file has the virtue of being human readable and the possibility of interchanging with other programs. A binary file is likely more compact (not always!) and is simple to write and read - no need for parsing or tokenizing. For this application I think the best experience will be to support loading and saving "CSV" files. Comma-separated value files are simple to parse and offer easy interchange. In fact, you will be able to export CSV from Excel to easily load a bunch of vectors for testing your dynamic memory facilities. One CSV file, generated from Excel, is provided for testing. The format will simply be a vector's name in the first field followed by the three vector magnitudes. Be sure to tolerate either a single "\n" or |"\r\n" for line endings as well as the potential for empty lines at the end of the file.

An example CSV file content:

```
a,1,2,3
b,4.4,5.5,6.6
c,8.8,11,13.2
d,17.6,22,26.4
```

Implementation Requirements

1. Dynamic Memory Features
    - Modify vector storage such that dynamic memory is used to allow essentially unlimited storage. You may implement any data structure you would like. Allocated memory should expand as new vectors are added - either in blocks as in the dynamic array example or for each vector as in the linked list example.
    - Your implementation must release dynamic memory when the `clear` command is issued. This makes sense for linked-list implementations. It might not make sense for dynamic array implementations, but it is a requirement nonetheless.
    - Your implementation must release all memory when exiting. While not absolutely required, consider using `atexit()` to register an exit handler

and/or a signal handler for `SIGINT` to trap CTRL-C and exit gracefully. Your demo will require running with `valgrind` to prove no memory leaks at exit.

2. File I/O
    - Add a command to load a CSV file. the command will simply be `load <fname>` where `<fname>` is the file to be loaded. It should be noted that `<fname>` cannot contain spaces, but relative and absolute paths should work (i.e. `../data1.csv` or `/home/user/data/f1.csv` or `myData.txt`). Do not add `.csv` to the entered filename programatically, and do not necessarily expect the filename entered by the user to have a `.csv` extension.
        - You must gracefully handle a missing (or mistyped) file.
        - You must gracefully handle a mis-formatted CSV file.
        - Should you clear vector storage before loading a file or merge the contents of the file with vectors already in memory, possibly overwriting some vectors? This is completely up to you in your design.
    - Add a command to save all current vectors in memory to a CSV file. The file should only include actual vectors (no empty or null vectors). This file should be formatted such that it can be read in via the load command as well as opened by Excel. It does not make sense to append to an existing file, so it should overwrite a file should it already exists. The command should simply be `save <fname>` and the same behavior for relative and absolute paths apply as noted above.
3. **OPTIONAL** - It might be nice to exercise your dynamic memory capabilities. Add a `fill` function to automatically generate a large number of vectors.

Deliverables

Be sure to do all work for this lab in the git repository you started during lab in Week 6. You may work on the main branch and should periodically commit changes throughout the period of time you are working on this project.

For submission, you will push the updated code to the git repository along with a tag marking the “release” of new features. You will also provide a README.md file with an overview of the project. Details and instructions for the tagging and README.md file will be provided during lab in Week 8 and will be part of the Week 8 in-lab activities.

The general demo script will be:

- make clean
- make // project builds with no warnings
- run program via valgrind
- attempt to load non-existing csv file // program reports non-existing file
- load provided csv file // observe loaded vectors

- load provided csv file again // observe same set of vectors
- do an operation resulting in a new vector
- save to new csv file
- exit program // observe valgrind not report errors or leaks
- run program
- load file saved // observe all vectors present
- observe saved file in editor or Excel