Lab 04 – Part A: Arrays on the Heap

Assigned: 2019-02-12 10:00:00

Due: 2019-02-14 23:59:00, concurrent with Part A of this lab.

Instructions:

- Written portions of this assignment are submitted via Canvas. Unless specified otherwise, the written portion of the assignment is to be completed using LaTeX. All derivations, images, graphs, and tables are to be included in this document. Handwritten solutions will receive zero credit.
- Code portions of this assignment are submitted via `code.vt.edu`. Source code must be in the private repository, to which the CMDA 3634 instructors must have access.

Deliverables: For this assignment, you are to submit the following:

1. (Canvas) `<pid>_Lab_04.pdf`: A PDF file, renderd by `pdflatex` (the file generated by Overleaf is sufficient) containing the answers to the questions requiring written answers. Use the template provided in the project repository. Your submission should include both Part A and Part B of this lab.
2. (`code.vt.edu`) The source files required to compile and run your solutions to the lab and the tex and image files for your report, in the appropriate directories.

Collaboration: This assignment is to be completed by yourself, however, you make seek assistance from your classmates. In your submission you must indicate from whom you received assistance.

Honor Code: By submitting this assignment, you acknowledge that you have adhered to the Virginia Tech Honor Code and attest to the following:

> I have neither given nor received unauthorized assistance on this assignment. The work I am presenting is ultimately my own.

Resources

- More on Git:
  - `https://www.taniarascia.com/getting-started-with-git/`
  - `http://rogerdudler.github.io/git-guide/`
- More on LaTeX:
  - `https://www.overleaf.com/learn`
- More on C programming:
  - `https://www.geeksforgeeks.org/c-programming-language/`
  - Recommended texts
- Arrays on the Heap:
  - malloc/free: `https://en.cppreference.com/w/c/memory`
  - stack vs heap: `https://www.gribblelab.org/CBootCamp/7_Memory_Stack_vs_Heap.html`
  - stack vs heap: `https://medium.com/fhinkel/confused-about-stack-and-heap-2cf3e6adb771`

Tasks

Warning: I have indicated where you should be running commands in a terminal with the > character. This character is **not** part of the command!

1. **Setup** your coding environment.
   (a) Pull the lab materials from the upstream repository.
   (b) Examine the history to see what new files have been added by the instructors.
   (c) Push your local master branch to the origin repository to add the new changes.
2. **Implement** the following requirements in C. Read the readme file in the `labs/lab04/code/` directory. Be sure to use git to commit your code regularly. Also be sure to push your code to `code.vt.edu` regularly.
   (a) A solution to `Vector3D` from Lab 03a is provided in `labs/lab04/code/VectorND`. We will build from these solution files to convert `Vector3D` to work for N-dimensional vectors, using the heap.
   (b) Copy the files supporting 3-vectors and name the new files to indicate that they will support N dimensions. Use the names `vectorNd.h` and `vectorNd.c`. Add and commit the new files immediately, before making any changes to them.
   (c) Modify the contents of `vectorNd.h` and `vectorNd.c`. These changes must not be in `vector3d.h` or `vector3d.c`.
      i. Update the "include once" directives in the header and the filename in source file's include block.
      ii. Modify the `Vector3D` type to be a `VectorND` type. This new type should have only two members:
         - an array on the heap, represented by a pointer to a floating-point number, called `data`
         - an integer called `dimension`.
      iii. Create an allocator function (`allocate_VectorND`) with the following requirements:
         - the declaration is in `vectorNd.h` and the definition is in `vectorNd.c`
         - the first argument is a pointer to a `VectorND` instance
         - the second argument is the desired dimension
         - use `malloc()` to allocate the `data` member of the `VectorND` argument
         - use `memset()` to initialize the `data` member of the `VectorND` argument to `0`

- the `dimension` member is initialized to the desired value
- the function returns an error code, with 0 indicating success and 1 representing failure
- the allocation fails if `malloc()` fails (returns a null pointer)

iv. Create a deallocator function (`deallocate_VectorND`) with the following requirements:
- the declaration is in `vectorNd.h` and the definition is in `vectorNd.c`
- the first argument is a pointer to a `VectorND` instance
- use `free()` to deallocate the `data` array
- there is no error code, as `free()` has no standard failure mechanism
- the `data` pointer should point to `NULL` when this call is complete
- the `dimension` member should be set to 0

v. Convert the `norm`, `normalize`, `axpy`, `inner_product` routines to operate on N-dimensional vectors, using `VectorND`, with the following requirements:
- the declarations are in `vectorNd.h` and the definitions are in `vectorNd.c`
- all type instances are passed-by-pointer
- `norm` should return the norm of the vector, not an error code
- `normalize` should return an error code, with 0 indicating success
- `normalize` must handle the case of the zero-vector. Note, direct comparison with between two floating point numbers is dangerous. We will discuss this in more detail in the future, but for now, testing if $|x - \hat{x}| < 10^{-5}$ is an acceptable substitute.
- `axpy` and `inner_product` must return an error code, with 0 indicating success and 1 representing failure
- `axpy` and `inner_product` fail if the input vectors do not have the same length
- `inner_product` must now return the length of the vector through a new, third argument to the function. Remember, use a pointer!
- Use a `for` loop to handle element-wise operations on the N-dimensional vectors.

(d) Verify that your implementation works correctly by running the `make test` rule and running the `vectorNd_test` program. Verify that you pass the tests successfully. If not, debug until you have passed the tests. You may consider testing these routines incrementally, by commenting out ones that you have not completed.

(e) Redirect the output of `vectorNd_test` to a text file named `output_pt_a_vector.txt`.

```
> ./vectorNd_test > output_pt_a_vector.txt
```

Be sure to include this output file in your final git submission.

3. **Answer** the questions listed below. The following workflow uses Overleaf, but you are welcome to use a local tex installation if you prefer.

(a) In Overleaf, create a Lab 04 project.

(b) Copy the files `lab04_report.tex` and `lab04_report_a.tex` from the `labs/lab04/report/` directory to Overleaf.

(c) Answer the questions and upload data files where necessary. The template is in `lab04_report_a.tex`, which is automatically included in the main tex file.

(d) In your VM, copy your report tex source and output files into the `labs/lab04/report/` directory. Overleaf allows you to download projects in `.zip` format. Download the zip file, move it to the report directory, and unzip it.

```
> unzip <project_name>.zip .
```

(e) Add them to git, and commit them.

4. **Submit** your results.

(a) Push your source code and latex files to `code.vt.edu`. From anywhere in your local projects repository

```
> git push origin master
```

(b) Examine your assignment repository on `code.vt.edu` to be sure that all of your materials have been correctly submitted.

(c) When you have completed Part B of this lab, upload a PDF of your report to Canvas.

Questions

Answer the following questions using the latex template provided in the `lab04/report` directory of the assignment repository.

1. Use the `listings` package to include your output (`output_pt_a_vector.txt`) in your pdf. You will need to copy `output_pt_a_vector.txt` to the reports directory.
2. For each of the following use-cases, indicate if the specified array should be allocated on the stack, the heap, or either. Explain your selection.
   (a) An array of integers length 10 in a function that is called a small number of times.
   (b) An array of doubles of length 3, where $\sim 10^3$ instances exist and frequently used in the program.
   (c) An array of doubles of length 3, where $\sim 10^4$ instances exist and frequently used in the program.
   (d) An array of doubles of length 3, where $\sim 10^5$ instances exist and frequently used in the program.
   (e) An array of doubles of length 3, where $\sim 10^6$ instances exist and frequently used in the program.
   (f) An array of doubles of length 3, where $\sim 10^8$ instances exist and frequently used in the program.
   (g) An array of floats of length 10,000, to be used throughout the whole program.
   (h) An array of floats of length 10,000, to be used in a single function.
3. In C, there is no mechanism to see if a pointer points to heap memory that has already been allocated, so we cannot be sure that we do not re-allocate an array. How can we code defensively to ensure that this does not happen?
4. Other than the instructor or TAs, who did you receive assistance from on this assignment?