Program 5 - ADT Sequence Structures

CS 580U - Fall 2017

Due Date: 5:00 p.m., November 27, 2017

Any changes made to the assignment after posting will be in red (Last Updated 11/21 3:36 p.m.)
All programs will be tested on the machines in the Q22 lab. If your code does not run on the system in this lab, it is considered non-functioning EVEN IF IT RUNS ON YOUR PERSONAL COMPUTER. You can write your code anywhere, but always check that your code runs on the lab machines before submitting.

Driver Code and Test Input Files

- **Provided Files**
  - program5.c //Driver Code

Grading Rubric

TOTAL: 25 points

- **Part A (10 points):**
  - [2 pts] Test #1 Passed: newVector function creates a vector with properly initialized values
  - [2 pts] Test #2-4 Passed: inserting into the vector passes all tests
  - [2 pts] Test #5-7 Passed: reading from vector passes all tests
  - [2 pts] Test #8-9 Passed: removing from vector passes tests
  - [2 pts] Test #10 Passed: deleting vector does not result in memory leak
- **Part B (10 points):**
  - [2 pts] Test #11 Passed: newList function creates a list with properly initialized values
  - [2 pts] Test #12 Passed: inserting into list at index passes test
  - [2 pts] Test #13 Passed: reading from list at index passes test
  - [4 pts] Test #14 Passed: removing and deleting from list at index passes test
- **Part C (5 points):**
  - [5 pts] implements profileInsert, profileRead, and profileRemove as described
- **Style Guidelines and Memory Leaks**
  - *You will lose significant points for the following:*
    - Makefile does not have requested format and labels (-5 points)
    - Does not pass Valgrind Tests (-5 points)
    - Does not follow requested program structure and submission format (-5 points)

Guidelines

This is a pair programming assignment. You and a partner can divide up the work. Although both of you may not work on all parts of the program you should understand and be able to fully explain every portion of the code. Outside of your team, it is permissible to consult with classmates to ask general questions about the assignment, to help discover and fix specific bugs, and to talk about high level approaches in general terms. It is not permissible to give or receive answers or solution details from fellow students.

You may research online for additional resources; however, you may not use code that was written specifically *to* solve the problem you have been given, and you may not have anyone else help you or your partner write the code or solve the problem. You may use code snippets found online, providing that they are appropriately and clearly cited, within your submitted code.

If you or your partner are found to have plagiarized any part of the assignment, both will receive a 0 and be reported.

By submitting this assignment, you agree that you have followed the above guidelines regarding collaboration and research.

For the next program you and partner are going to build both a dynamic array (vector) and a linked list. Each data structure will be tested to ensure the validity of its operations. Once that is complete and you know your data structures are working, we are going to test them against each other for performance (this is the fun part).

To make our code more portable, we are going to wrap our data in a Data struct, and build all our operations around this data type. In a header file, data.h, create a struct, Data, that contains the following:

- a single integer called, 'value'

Since we are going to be passing by value, we will not need a constructor or destructor.

Part A: Vectors

- You must break up your code into vector.h and vector.c according to the conventions we discussed in class.
- Create a dynamic array data structure, Vector. You must create your struct and internal array on the heap (using malloc). Your dynamic array should have, at minimum, the following:
    - *data*: A pointer to a Data struct array
    - *current_size*: an unsigned integer containing the current size
    - *max_size*: an unsigned integer containing the maximum capacity
    - *void (*insert)(Vector * array, int index, Data value):* a function pointer to an insert function
    - *Data * (*read)(Vector * array, int index):* a function pointer to an read function
    - *void (*remove)(Vector * array, int index):* a function pointer to an delete function
    - *void (*delete)(Vector * array):* a function pointer to a destructor
  - You may add additional attributes if you need them
- You must create the following functions for your Vector
  - Constructor - initializes the vector struct attributes and returns a pointer to a Vector struct created on the heap
    - Vector * newVector()

  NOTE: The remaining function names are suggestions since the driver code calls them all via function pointers

  - Insert - inserts an element at the specified index. Use the 2n+1 geometric expansion formula to increase the size of your list if the index is out of the current bounds.
    - void insertVector(Vector * array, int index, Data value);
  - Remove - deletes an element from the list at the specified index.
    - void removeVector(Vector * array, int index);
      - You must implement true deletion, which will reduce the size of the vector by 1

    - Read - return ***the pointer to the data struct*** from the specified index, return NULL if the index is out of bounds, and a data struct with the value set to -1 if the index has not been initialized
        - Data * readVector(Vector * array, int index);
    - Destructor - upon exiting, be sure all memory has been freed by calling
        - void * deleteVector which free's all struct memory
            - You should return a NULL pointer from any delete procedure. This is just a convention.

Part B: Linked Lists

- You must break up your code into list.h and list.c according to the conventions we discussed in class.
  Your node struct must have the following:
    - *next/prev:* A pointer to the next and previous nodes
    - *data:* A data object **(note: not a pointer)**

  Your list struct must have the following:
    - *head/tail:* A pointer to nodes at the head and tail
    - *void (*insert)(List *, int, Data):* a function pointer to an insert function
    - *Data * (*read)(List *, int):* a function pointer to a read function
    - *void (*remove)(List *, int):* a function pointer to an delete function
    - *void (*delete)(List *):* a function pointer to a destructor
- Create a doubly linked list using a list and node structs. You must create your linked list on the heap (using malloc). Your linked list should have the following operations:
    - Constructor - initializes the linked list struct:
        - List * newList()
            - A pointer to a head and tail node, both initialized to NULL
            - set function pointers to the appropriate functions
            - returns a pointer to a List struct created on the heap
    - Insert - inserts an element at a specified index in the list.
        - void insertList(List * list, int index, Data value);
            - Adds the Data to the specified index
            - If the index is out of bounds, adds the data to the end of your list.
    - Delete - deletes an element from a specified index in the list.
        - void removeData(List * list, int index);
            - If the index is out of bounds, you should just return without doing anything.
    - Read - returns a pointer to the data element stored in the list
        - Data * readData(List * list, int index);
            - If the index is out of bounds, return a NULL pointer

Part C: Profiling Your Code

- Create another file called profile.c/.h. Inside this file you should implement 3 functions that profile your vector and list data structures. You must ensure that only the relevant code is being profiled. Example profiling code is provided:
    - ```
      #include <time.h>
      #include <sys/time.h>/* timeval, gettimeofday() */
      ...
      struct timeval start, stop;
      gettimeofday(&start, NULL);

      //code to be profiled…
      gettimeofday(&stop, NULL);
      time_t start_time = (start.tv_sec* 1000000) + start.tv_usec;
      time_t stop_time = (stop.tv_sec* 1000000) + stop.tv_usec;
      float profile_time = stop_time - start_time;
      ```
- Print the results to stdout. The description of each function is below.
    - void profileInsert(Vector *, List *)
        - Insert 1000 data objects. The data objects should contain integers in increasing value, i.e. 1,2,3, etc.
        - Profile the insertion of the data objects for both the vector and the list separately, then print the results.
    - void profileRead(Vector *, List *)
        - Search for 100 random data objects in your sequences. Try to use as similar search algorithms as possible for your search. Your search algorithm can assume unordered data.
        - Profile the read time of the data objects for both the vector and the list separately, then print the results.
    - void profileRemove(Vector *, List *)
        - Remove for 100 random data objects in your sequences.
        - Profile the read time of the data objects for both the vector and the list separately, then print the results.

Part D: Submission

- Required code organization:
    - program5.c //Provided Driver Code
    - profile.h/c
        - profileInsert(Vector *, List *)
        - profileRead(Vector *, List *)
        - profileRemove(Vector *, List *)
    - data.h
        - Data struct
            - value (int)
    - vector.h/c - Your header file should have the following function declarations:
      ```

        - Vector struct
            - data (Data)
            - current_size (int)
            - int max_size (int)
            - void (*insert)(struct Vector *, int, Data);
            - Data * (*read)(struct Vector *, int);
            - void (*remove)(struct Vector *, int);
            - void * (*delete)(struct Vector *);
        - Vector * newVector();
    - list.h/c - Your header file should have (at minimum) the following function declarations:
        - Node struct
            - data (Data)
            - next (Node *)
            - prev (Node *)
        - List struct
            - head (Node *)
            - tail (Node *)
            - void (*insert)(struct List * , int , Data);
            - Data * (*read)(struct List * , int);
            - void (*remove)(struct List * , int);
            - void (*delete)(struct List *);
                - deletes the entire list from memory
        - List * newList();
    - makefile
        - *You must have the following labels in your makefile:*
            - all - to compile all your code to an executable called '**program5**' (no extension). **Do not run**.
            - run - to compile if necessary and run
            - checkmem - to compile and run with valgrind
            - clean - to remove all executables and object files
- While inside your program 5 folder, create a zip archive with the following command
    - zip -r <username>_program5 <list of required files>
        - This creates an archive of all file and folders in the current directory called <username>_program5.zip
        - **Do not zip the folder itself, only the files required for the lab**
- Upload the archive to Blackboard under Program 5.