Assignment 5 - ADT Dynamic Arrays

Due Date: 5:00 p.m., Nov 11 (Fri)

All programs will be tested on the machines in the Q22 lab. It is required that your code run on the computers in the lab (or via ssh).

Any changes made to the assignment after posting will be in red

- *All structs and function declarations should go into a header file*
- *You must use separate compilation for this lab, with a header file and source file for your data types*

Part 1: Vectors

Part A: Dynamic Arrays

- Create a struct, Data, that contains a single integer called, 'value'
- 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
  - You may add additional attributes if you need them
- You must create the following functions for your Vector
  - initVector - initializes the vector struct attributes and returns a pointer to a Vector struct created on the heap
    - Vector * initVector()
  - 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 vectorInsert(Vector * array, int index, Data value);
  - Delete - deletes an element from the list at the specified index.
    - void vectorRemove(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 * vectorRead(Vector * array, int index);
- 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.

- **EXTRA CREDIT**
  - Another method of deleting without reducing the size of the array is to have an additional index array. Instead of deleting elements from the array, you keep track of valid indexes in another array, and use the second array to iterate. Implement this form of deletion instead of reducing the size of the array for extra credit. Below is a visual example.

index array

0	1	2	3	4

data array

6	7	5	3	2

After deleting index 3, we can find the valid 3rd index by going first to the index array, then to the data array. *notice the value in index 3 isn't removed.* No need to delete when you can just overwrite later.

index array

0	1	2	4	-1

data array

6	7	5	3	2

You must still reduce current_size as needed. This implementation should be transparent to the driver code.

Part B: Profiling Your Code

- Create another version of your insert function called, vectorInsertIncremental, that uses incremental expansion (only add 1 to your array size when more space is needed).
- When we test your code, we will multiply the ratio of unused space to used space in your vector with your total time to get your total efficiency.
  - In other words, if your vector has a current size of 20 and capacity of 30, you would multiply your total time by 1.5.

- Part C: Designing Your Code
    - Create one more version of your insert function called, vectorInsertMine, that uses an expansion algorithm of your own design.

Part 2 - Submission

- Required code organization:
    - program5.c //Driver Code
    - vector.h
        - Your header file should have the following function declarations:
            - Vector * initVector();
            - void vectorInsert(Vector * array, int index, Data value);
            - void vectorInsertIncremental(Vector * array, int index, Data value);
            - void vectorInsertMine(Vector * array, int index, Data value);
            - Data * vectorRead(Vector * array, int index);
            - void vectorRemove(Vector * array, int index);
            - void * deleteVector(Vector *);
    - vector.c
    - makefile
        - *You must have the following labels in your makefile:*
            - all - to compile all your code to an executable called '**program4**' (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 program5 *
        - This creates an archive of all file and folders in the current directory called program5.zip
        - **Do not zip the folder itself, only the files required for the lab**
- Upload the archive to Blackboard under Program 5.
- You may demo your lab by downloading your archive from Blackboard. Extract your archive, then run your code, show your source, and answer any questions the TA may have.

Grading Guidelines

Total: 25 points

- **Part A:**
  - Data Constructor returns a pointer to a Data struct that encapsulates value *(1 point)*
  - Passes Initialization test 1 *(3 points)*
  - Passes Insert tests 2-4 *(2 points each)*
  - Passes Read tests 5-7 *(2 points each)*
  - Passes Delete 8-10 *(2 points each)*
  - EXTRA CREDIT - Must be fully working (5 points)
- **Part B & C**
  - Passes test 11: Incremental, Geometric, and own Expansion Algorithms implemented *(3 points)*
- **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 (-10 points)
    - Does not follow requested program structure and submission format (-10 points)
    - Does not follow formatting guidelines (-5 points)