Homework #2 - Dynamic Arrays in C

Objective: To implement a **dynamic array** "abstract data type" as a C module and to get familiar with the use of memory allocation and deallocation functions in the standard library **stdlib**. Every call to malloc(), calloc(), realloc(), fopen(), fread() or equivalent status returning function MUST include error handling.

1. Upload the file **hw2_files.tar** (available in MyCourses) to the **hw2** working directory. The tarball contains:

   the header file **DynamicArrays.h and ClassErrors.h**

   text file **us-eng-words.txt, us-eng-short.txt**, and **longLongTest.txt** for testing

   a binary called **debugTst**

   and a copyright note.

2. Using the class notes as a reference, examine **debugTst** using **gdb** and **valgrind**.
   Normally when you run the debugger you can see all the source code, but the source was not included, so you can only see the line numbers in this case. You can still set break points and print variable names.

   Run the program in the debugger passing the command line option "divide":
   1. What line does the program crash on?
   2. Stop the debugger just before the crash location, what is the value of 'number'?

3. Run the program using the following:
   valgrind --tool=memcheck --leak-check=yes ./debugTst memory
   1. How much memory "leaks"?
   2. Rerun with "good", how much memory is used and returned?

Be sure to include a description of how you used gdb and valgrind to answers the above questions in your Analysis.txt file. Your analysis.txt file must be formatted so that it can be viewed on an78 character wide display. (e.g. Format the final file by adding a hard CR/LF at column 78 or before).

4. Write the implementation of your *dynamic array module* (**DynamicArrays.c**) using the interface specification in the file **DynamicArrays.h**, where the elements of the array are variables of type **Data** containing an integer and a character–string field. Use the pseudo-code given in the Lecture Notes for reference.

5. A skeleton testing program called **TestDarray_hw.c** is provided. Add the necessary code to test the module using the file **us-eng-words.txt** as input. Each array element should store the word along with its position (index) in the file (that is what the integer field in the structure **Data** is for).

   Programs Behavior:

   1. The test program should get its *input from a file* whose name is passed in the *command-line*.
   2. The words data file will contain at least 6 words. Each word will typically be less than 255 bytes each. Your code should detect any words that violate the 255 byte rule, print a warning message to STDERR and ignore the word. This warning will not change the return codes.
   3. The test program programs should *print the first and last 6 elements* of the data set, and the *total number of words* in the input file to standard output **stdout** (*e.g.*, the screen). The last 6 elements must be in file order (e.g. as if you printed the entire list out and looked at the last 6)
   4. If called with a wrong number of arguments, the program should print a usage message to **stderr** and return an error code equal to 1. The usage message must be clear and must include a very brief description of what the program does and a very brief description of what all the options are and what they do.
   5. If the input file could not be opened, your programs should print an error message to **stderr** and return an error code of 2. The error message must clearly state what file could not be opened.
   6. Your code should check for all important return codes (e.g. fopen(), malloc(), realloc(), calloc(), etc) and exit or return error codes as defined in the header blocks. All error messages should be written to STDERR and should include the source code file name and line number.
   7. If the programs completes successfully it should return an error code of 0.

6. Write a make file, with test, clean, mem, all and help sections. Make all should build your dynamic array and test code binary. Make test should run the functional test with "us-eng-words.txt" and place the output in **out.txt**. Make mem should run the same test using valgrind and place the valgrind results in mem.txt. Note, valgrind will s will take 2 minutes to run. Your make file should also include clean and help options, doing the normal things.

   Write an **analysis.txt** summarizing your implementation and results formatted to 78 columns.

   Create a tarball **lastName_hw2.tar** (lastName is your last name) with all relevant files and submit it.

Grading Criteria

1. (10 points) Correct analysis for debugTst
2. (25 points) Correct implementation of Dynamic Array.
3. (25 points) Correct program behavior (error codes, error messages, checking for all important return codes, etc)
4. (25 points) Correct results reported.
5. (5 points) No memory leaks (you returned all memory back to the heap)
6. (10 points) for your Makefile .

Notes:

1. Start work on this homework early, especially if you do not have experience using pointers or programming in C .
2. It should be clear that the implementation must be your own (not copied from some remote source in the Internet)