PROJECT 1: VECTOR

OVERVIEW

This assignment will explore the Vector Abstract Data Type (ADT). Remember that an ADT is a user-defined data type that includes a variety of operations to manipulate data, access or set values in the data structure of the ADT. A Vector is similar to an array in C in that it stores values of the same type. However, unlike an array a Vector can grow and shrink in size dynamically to fit the number of elements that need to be stored by the Vector while at the same time not waste memory resources unnecessarily. It can keep track of how many elements the Vector contains, and perform several other functions. You are required to:

1. Implement two different implementations of a Vector ADT based on different data structures.
2. Provide test programs to run the different implementations.

THE PROGRAMS

Write two separate C programs named *vectorADTArray* and *vectorADTLL* that implement and evaluates respectively two different versions of a Vector ADT (also known as an ArrayList in Java), one based on an array data structure, the other on a linked list, as discussed in class. Regardless of the implementation, the Vector ADT must grow and shrink dynamically in size to accommodate an unknown number of elements with only memory being the limiting factor for growth. The two types of implementations are described in details below. The header files `vectorArray.h` and `vectorLL.h` uploaded in *eLearning*, provide the interface for the VectorADT.

ARRAY-BASED IMPLEMENTATION

For the array-based implementation, you must choose a fixed size array for the Vector elements. If the array is not sufficient to hold more elements, then the program must allocate a new and larger data structure to hold all elements and copy the data from the old into the new data structure before freeing the memory of the old data structure. Likewise, if the array is too large after elements have been removed from the Vector, then the program must allocate a new and smaller data structure to hold fewer elements. Similarly, to above it must copy the data from the old into the new data structure before freeing the memory for the old data structure. You must choose a load factor to decide when to shrink the array. For example, a load factor of 50% would imply that if less than 50% of the Vector's capacity is used to store elements, then the array shrinks to a new size slightly above 50%, to allow for subsequent minor growth.

For this implementation you are to write an implementation file, `vectorArray.c`, which implements all of the functions included in `vectorArray.h`, also listed below. You must use `nodeArray.h` and `nodeArray.c`, uploaded to *eLearning*. These files contain the node data structure, which is used in the implementation of Vector.

LINKED-LIST-BASED IMPLEMENTATION

Next, you must use a linked-list data structure to dynamically grow and shrink the Vector as needed. There is no need for creating a structure that can only hold a fixed number of elements as with the implementation above. Your structure will grow with every element added and shrink with every element removed. Note that the implementation for capacity in this Vector is identical with the size of the data structure as the capacity to hold

elements is identical with the length of a linked-list. Use the header file `vectorLL.h` that includes the same functions as those in `vectorArray.h`.

For this implementation you are to write an implementation file, `vectorLL.c`, which implements all of the functions included in the header file, also listed below. You must also use `nodeLL.h` and `nodeLL.c`, also uploaded to *eLearning*, to develop your solution. Again, these files contain the node data structure, which is used in this implementation of Vector.

FUNCTIONS IN VECTOR ADT

Here is a list of the functions that both versions of the Vector ADT must implement:

```
newVector     vadd         vget         vset         vremove
vcontains     vindexOf     vclear       vdelete      vsize        vcapacity
```

EVALUATION & EXPERIMENTS

The two versions of Vector ADT need to be evaluated using two test programs named *vectorADTArray* and *vectorADTLL* implemented by the file `testVectorArray.c` and `testVectorLL.c`. Both test programs share the same test code. The difference between both programs is that one includes `vectorArray.h` and is compiled with `vectorArray.c`, `vectorArray.h`, `nodeArray.h` and `nodeArray.c` and the other includes `vectorLL.h` and is compiled with `vectorLL.c`, `vectorLL.h`, `nodeLL.h`, and `nodeLL.c` respectively. Follow the outline below to write your test program.

1. Capture the time using the real-time clock.
2. Create a Vector.
3. Add 500,000 elements to your vector read from standard input.
4. Check for the location of 100 elements of the vector read from standard input.
5. Remove up to 300,000 elements read from standard input. Some may not be removable as they don't exist in the Vector.
6. Check for the location of 100 elements of the Vector read from standard input.
7. Clear the Vector.
8. Delete the Vector.
9. Capture the time using the real-time clock and print out the elapsed time.

Both test programs will be executed using a test file that includes the necessary elements to run the program. The program will read the test data from a file by redirecting standard input to the file.

IMPLEMENTATION SUGGESTIONS

Although copying and pasting is highly error prone, one easy way to make sure that you have not missed any function to implement in each ADT is to simply copy the contents of `vectorArray.h` into `vectorArray.c` and `vectorLL.h` into `vectorLL.c` respectively and then implement the functions in each file accordingly.

Your two programs must have the following files:

- header files including `vectorArray.h`, `vectorLL.h`, `nodeArray.h`, `nodeLL.h`,
- implementation files including `vectorArray.c`, `vectorArray.c`, `nodeArray.c`, `nodeLL.c`, and `testVectorArray.c` and `testVectorLL.c`.

Look back at the lecture of the first weeks to get hints on how to start coding the different files. Be sure not to modify any of the files you were given as they determine what you are to implement.

PROJECT OUTCOMES

- To develop and compile two different implementations of an ADT along with test programs.
- To use dynamic memory allocation for memory management.
- To evaluate empirically the data structures.
- To use a Makefile for compiling source code into an executable.

DELIVERABLES & EVALUATION

Submit your complete solution as individual files containing A) source code, B) a single Makefile to compile the two programs (sample Makefile is posted in eLearning), and C) a README file (if applicable) to the corresponding Dropbox in *eLearning*. In addition, submit the source code and the Makefile as individual files to your own directory on the shared drive under `/shared/dropbox/treichherzer/cop5990`. Be sure to follow the Project Submission Instructions and Code Documentation and Structuring Guidelines posted by your instructor in *eLearning* under Course Materials to avoid unnecessary deductions.

Finally, be sure that your code is well organized. Functions should be refactored so that they can be easily reviewed and maintained. This means that any large function (functions with a body of about 20 or more lines of code) must be broken down into several smaller functions. You should refactor code such as the body of a loop or an if-else-statement when it exceeds a large number of lines of code or when the statement contains additional nested if-else or loop statements.

DUE DATE

The project is due on the date shown in the syllabus and *eLearning* calendar. Late submissions will not be accepted and the grader will not accept any emailed solutions. The syllabus contains details in regards to late submissions.

IMPORTANT NOTES:

1. Projects will be graded on whether they correctly solve the problem and whether they adhere to good programming practices.

2. Projects must be submitted by the time specified on the due date. Projects submitted after that time will get a grade of zero.

3. Please review UWF's academic conduct policy that was described in the syllabus. Note that viewing another student's solution, whether in whole or in part, is considered academic dishonesty. Also note that submitting code obtained through the Internet or other sources, whether in whole or in part, is considered academic dishonesty. All programs submitted will be reviewed for evidence of academic dishonesty, and all violations will be handled accordingly.

GRADING

This project is worth 100 points in total. The rubric used by the grader is included below. Keep in mind that there will be a substantial deduction if your code does not compile.

Project Submission	Perfect	Deficient		
eLearning	5 points individual files have been uploaded	0 points files are missing		
shared drive	5 points individual files have been uploaded	0 points files are missing		
Compilation	**Perfect**	**Good**	**Attempted**	**Deficient**
Makefile	5 points make file works; includes clean rule	3 points missing clean rule	2 points missing rules; doesn't compile project	0 points make file is missing
compilation	10 points no errrors	7 points some warnings	3 points some errors	0 points many errors
Documentation & Program Structure	**Perfect**	**Good**	**Attempted**	**Deficient**
documentation & program structure	5 points follows documentation and code structure guidelines	3 points follows mostly documentation and code structure guidelines; minor deviations	2 points some documentation and/or code structure lacks consistency	0 points missing or insufficient documentation and/or code structure is poor; review sample code and guidelines
VectorADT	**Perfect**	**Good**	**Attempted**	**Deficient**
Vector ADT Array impl.	25 points correct, completed	18 points minor errors	7 points incomplete	0 points missing
Vector ADT Linked List impl.	25 points correct, completed	18 points minor errors	7 points incomplete	0 points missing
test program given	20 points correct, completed	14 points minor errors	6 points incomplete	0 points missing

I will evaluate your solution as attempted or insufficient if your code does not compile. This means, if you submit your solution according to my instructions, document and structure your code properly, provide a makefile but the submit code does not compile or crashes immediately you can expect at most 40 out of 100 points. So be sure your code compiles and executes.