Lab 2 – Arrays, Pointers, Structures, and Functions in C++

Computer Science 2413 – Data Structures – Fall 2019

Due by 11:59 pm CST on Wednesday, 04 September 2019

This lab is individual work. Each student must complete this assignment independently.

User Request:

"Create a simple program to read, store, and write drilling data."

Objectives:

1	Satisfy all requirements for input, input processing, and output.	*4 points*
2	Use arrays to store the data as it is read in.	*4 points*
3	Allocate and deallocate memory dynamically using `new` and `delete`.	*4 points*
4	Complete/use the given design elements, including two structures and one function (plus `main`).	*4 points*
5	Use proper design, coding style, documentation, and formatting.	*4 points*

Description:

This lab builds on Lab 1 by requiring the use of four important building blocks for C++ programs—arrays, pointers, structures, and functions.[1] Lab 2 does this by adding and modifying operational requirements and implementation requirements to those already present in Lab 1. These additional requirements define *Driller 0.2*. Other than these requirements, Driller 0.2 must satisfy all requirements of Driller 0.1 (as we will retroactively call the version of Driller from Lab 1).

Operational Issues:

In contrast to Driller 0.1, Driller 0.2 will read and store all data before sending any output to `cout`. Once the data is read in and checked for internal consistency (as described in Lab 1), the data will be sent to `cout` in reverse row order. Note that you do not have to sort the data. Whatever the order in which records appear in the file, Driller 0.2 will print them in reverse order. Also, note that the order of the data columns will remain unchanged. Note that all errors (e.g., messages regarding duplicate timestamps and invalid data) will be sent to `cout` before any valid records are sent there. Additionally, when Driller 0.2 outputs floating-point data, it will format it to always have a decimal point followed by two trailing digits, regardless of the format of the corresponding numeral in the input.

Implementation Issues:

In addition to the implementation requirements from Lab 1, you must use arrays, pointers, structures, and at least one function in this assignment. Arrays will be used to store the data and pointers will be used to refer to memory that is dynamically allocated. Note that the memory must be dynamically allocated, because in many cases it is not possible to know how much data will be found in a file at the time your code is written. Therefore, it is important to know how to dynamically allocate memory.

[1]Note that arrays, pointers, structures, and functions are part of C as well as C++. In fact, these building blocks are less important in C++ than they were in C, because C++ provides important alternatives to these, such as vectors, references, classes, and methods. However, for this lab we're focusing on understanding these basic building blocks.

Driller 0.2 must initially allocate enough space to hold the data for 10 drilling records. As it reads the file, if that allocation proves to be insufficient, Driller 0.2 will ask for a new allocation twice that large (using `new`), copy the old data to the new space, deallocate the old data space (using `delete`), set the old pointer to the deallocated memory to `NULL`, and continue reading from the file. If that allocation also proves to be insufficient, Driller 0.2 will repeat the doubling, copying, deallocating, and resuming process, continuing in this way until the entire file is read in. This process of dynamically doubling the array size as the data is read in must be modularized within a function. This function must be tested and shown to work correctly using unit tests that will be included in the zyLab test bench, as well as work in the final, integrated program. This function will be named "`doubleDrillingArray`" and its required signature follows:

```
drillingArray* doubleDrillingArray(drillingArray* currentDrillingArray)
```

The data for each record will be stored in a structure named "`drillingRecord`," which is defined as follows:

```
struct drillingRecord {
	double nums[16];		// holds the numeric data, in column order
	std::string strings[2];		// holds the string data, in column order
};
```

The array of records will be a part of a structure that also records the capacity of the array, defined as follows:

```
struct drillingArray {
	int capacity;			// maximum capacity, in records
	drillingRecord* data = NULL;	// pointer to array of records
};
```

Note that this lab does not require you to create any classes or to use object-oriented design at all. In fact, you don't even need to create any functions except the `main` function and `doubleDrillingArray`. Nonetheless, you must use good programming style and documentation, including making your code modular, using explanatory comments for each section of code, using meaningful variable and method names, using consistent indentation, etc.

You may write your program from scratch or may start from programs for which the source code is freely available on the web or through other sources (such as friends or student organizations). If you do not start from scratch, you must give a complete and accurate accounting of where all of your code came from and indicate which parts are original or changed, and which you got from which source(s). Failure to give credit where credit is due is academic fraud and will be dealt with accordingly.

You may use existing classes and data structures (such as vectors) in your code for reading and parsing the input data. However, the `doubleDrillingArray` function must be written from scratch and use only pointers and primitive C/C++ arrays.

Due Date:

You must submit an electronic copy of your code by **11:59 pm CST on Wednesday, 04 September 2019**.