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

Computer Science 2413 – Data Structures – Fall 2017

Due by 11:59 pm CST on Tuesday, 12 September 2017

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

User Request:

"Create a simple program to read and write eclipse data with additional error checking."

Objectives:

1	Satisfy all requirements of Lab 1.	*5 points*
2	Use arrays to store the data as it is read in.	*5 points*
3	Use pointers to reference space that is dynamically allocated for storing the data.	*5 points*
4	Develop and use an appropriate design, including the use of at least one function.	*5 points*
5	Use proper coding style, documentation, formatting, and unit testing.	*5 points*

Description:

This lab builds on Lab 1 by requiring the use of three important building blocks for C++ programs—arrays, pointers, and functions.[1] Lab 2 does this by adding two operational requirements and four implementation requirements to those already present in Lab 1. These additional requirements define *EclipseR 0.2*. Other than these requirements, EclipseR 0.2 must satisfy all requirements of *EclipseR 0.1*, as we are retrospectively calling your program from Lab 1.

Operational Issues:

EclipseR 0.2 will make the same correctness checks as EclipseR 0.1, plus more.

First, as with EclipseR 1.0, it will check to ensure that there are the correct number of columns in each data row, based on eclipse type. Note that you are guaranteed that there will be at least 10 columns in each data row and that column 10 will start with "P" (for partial) if there should be 16 columns and will start with some other character if there should be 18 columns. If this check fails, EclipseR must send the following error message on its own line to standard error:

`Error in data row r: c columns found. Should be s.`

where `r` is the data row (starting from the first data row; that is, not counting the header rows and indexed from 1), `c` is the number of columns found (indexed from 1), and `s` is either 16 or 18, as appropriate, depending on whether or not the eclipse is partial. If this column count error check fails, EclipseR 2.0 will skip the rest of this data row with no further checks.

Second, if the data row passes the column count check, EclipseR 2.0 must check to ensure that each entry in the following columns is a whole number (i.e., could be represented with some integer type, such as `int`, `long`, or `long long`): 1, 2, 3, 5, 7, 8, 9, 15, 16, and (if present) 17. It must also check to ensure that each entry in columns 11 and 12 is a decimal number (i.e., could be represented by some floating point type, such as `float`, `double`, or `long double`). Note that you do not necessarily need to

1 Note that arrays, pointers, 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, and methods. However, we're focusing on understanding the basic building blocks from which more complex structures can be built.

store the data from these columns in numeric types, although you may elect to do so. You are simply required to perform checks to determine whether these data *could be* stored in numeric types. Choose a type that looks appropriate to you, based on the data in the data file given. (The values tested will not fall outside those found in the data file.) If EclipseR 0.2 encounters an error in a column type, it must provide an error message indicating at which row and column the error was found as well as the type of error found, and move on to the next column of data. The error message must match the following:

`Error in data row r: Column c is not a t number.`

where *r* is the data row, *c* is the column number where the error was found, and the possible options for *t* are the type of number, either `whole` or `decimal`. Each error message must appear on its own line and is sent to standard error. If any of these conversion checks fail, EclipseR 2.0 will skip the third type of check, below.

Third, if the data row passes the column count and column conversion checks, EclipseR 2.0 checks to ensure that the catalog number of each eclipse is unique (not repeated), as did EclipseR 1.0. If it encounters a duplicate catalog number, it will send to standard error an error message with the following format:

`Error in data row r: Duplicate catalog number n.`

where *r* is the data row and *n* is the duplicated catalog number found. If a duplicate catalog number is found, this line is ignored. Otherwise, it is added to the set of eclipses in EclipseR 2.0.

In addition, in contrast to EclipseR 1.0, EclipseR 2.0 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 above), 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 eclipses appear in the file, EclipseR 2.0 will print them in reverse order. Also, note that the order of the data columns will remain unchanged.

Implementation Issues:

In addition to the implementation requirements from Lab 1 (using Eclipse, etc.), you must use arrays, pointers, functions, and unit tests 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.

EclipseR 0.2 must initially allocate enough space to hold the data for 10 eclipses. As it reads the file, if that allocation proves to be insufficient, EclipseR 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`), and continue reading from the file. If that allocation also proves to be insufficient, EclipseR 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 you must also develop and code, as well as work in the final, integrated program.

In addition to these specific requirements, you must carefully consider how to organize the data such that it is easy to manipulate (e.g., using looping constructs), easy to understand (by treating similar data in similar ways), and helps to preserve data integrity (e.g., by grouping related data together in structured ways). **You will create simple design document that explains and justifies your design choices.** Please make this a PDF file and name it "**design.pdf**" in your submission.

The only libraries you may use for this assignment are `iostream`, `iomanip`, and `string` (`#include <iostream>`, `#include <iomanip>`, `#include <string>`).

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 the array size doubling function. 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 must follow C++ conventions for compilation modules, including using a header (`.h`) file for constant variables, prototypes, etc. and a source (`.cpp`) file for implementations of functions, methods, 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 other source. Failure to give credit where credit is due is academic fraud and will be dealt with accordingly.

Due Date:

You must submit an electronic copy of your Eclipse project **and your design document** to the appropriate dropbox in Canvas by **11:59 pm CST on Tuesday, 12 September 2017**.