Joshua Fox, M.S.
Kaji Rashad, ET

Last Updated: 09/29/18

Lab 5 – File I/O

Overview

For this assignment, you are going to load a series of files containing data and search the loaded data for specific criteria. That might sound a bit dry, but the files contain information about LEGO sets, and everyone loves LEGO! The structure of this assignment is a bit open-ended; you can solve this problem in any way that you see fit. That is the first part of the lab, which is worth **20** points. You will submit this portion on **zyBooks**, just like the previous assignments.

The second part of the lab involves interacting with the terminal, and providing **command-line arguments** to your application. You are going to use the *filename* as **command-line argument** of *main()*. To achieve this, you will modify the **main.cpp** file. This portion will be worth **5** points, and you will submit this via Canvas.

Description

First things first, the files: There are **3** main files that you will be loading in this assignment:

- *lego1.csv*
- *lego2.csv*
- *lego3.csv*

In addition, there are **3** sample files included with this document. Use those to test your loading process. The 3 larger files (on zyBooks, without "SAMPLE_" in the filename) are in the exact same format, the sample files have just been reduced to a handful of entries, so you could easily test as you go.

The data that you will be loading is information about a LEGO set:

1. *Its set number*
2. *The theme it comes from (City, Technic, Star Wars, etc.)*
3. *The name of the set*
4. *How many parts and mini-figures it contains (if any)*
5. *Its price in US dollars*

Your goal is to read this from 1 of those 3 files (or all of them at once), store it, and then search it based on a few different criteria.

Main.cpp is only required for this assignment, *but* you are free to write any class/functions that you see fit to assist you solve this problem. Main.cpp has some structure to it already to help you get started. Take some time to ***think*** about how you might go about this before diving in.

Searches

The different searches you will perform will be based on a *menu* that might look like this:

1. Most expensive
2. Largest piece count
3. Search for set name containing…
4. Search themes...
5. Part count information
6. Price information
7. Minifigure information
8. If you bought one of everything...

Most Expensive From the sets that were loaded, which is the most expensive?	The most expensive set is: *Name:* Super Awesome Building Set *Number:* 99923 *Theme:* City *Price:* $21.99 *Minifigures:* 4 *Piece count:* 286
Largest piece count From the sets that were loaded, which has the most parts?	The set with the highest parts count: *Name:* Really Big Set *Number:* 22231 *Theme:* Technic *Price:* $249.99 *Minifigures:* 0 *Piece count:* 5211
Search for set names containing… Get a string as input from the user. Then search all sets and their names to see if they contain the search term. There could be a lot of sets matching the search term, so show them in a more concise, list format with the *set ID*, *name*, and *price*. If no sets are found, report that as well.	Sets matching "Fire Station": 49281 Fire Station $19.99 9381 Big Fire Station $49.99 **-OR-** No sets found matching that search term
Search for set themes containing… Ditto[1], but for the theme of the set instead	Sets matching "City": 1234 Police Station $29.99 49281 Fire Station $19.99 // And TONS more… LEGO City is huge!!
Part count information	Average part count for 601 sets: 492

[1] A similar thing; a duplicate

Show the average parts for all the loaded sets which have non-zero values, and show the largest/smallest sets as well	Set with the smallest part count: [Set data goes here] Set with the largest part count: [Set data goes here]
Price information Ditto, but for prices: Average, Minimum & Maximum	Average price information for 601 sets: $500 Set with the minimum price: [Set data goes here] Set with the maximum price: [Set data goes here]
Mini-figure information Ditto, but for mini-figures	
If you bought one of everything… How much would it costs? How many parts and mini-figures would you have?	If you bought one of everything... It would cost: $9999.99 You would have 200207 pieces in your collection You would have an army of 3000 mini-figures!

Reading Files

When reading a text file, all of the data typically gets read in as a string. If the final storage variable isn't a string (such as a person's age, or the price of something), you must convert it. In the *<string>* header file, there are a number of functions to help you convert. These function converts a STRING_TO_SOMETHING, and are named like *stoi* (string to integer), *stof* (string to float), etc.

The implementation of some of these functions may throw an exception of type "invalid_argument" if the conversion process fails, so you may want to encapsulate these operations in *try/catch* blocks, and use a default value if you catch an exception—if the number can't be converted, (in this case) it's because a value wasn't there, so what would be a good value to use in the absence of anything else? Refer back to the section on Exceptions in your textbook if you need to.

Terminal

The reasoning behind interacting with the terminal is to increase your flexibility and exposure. It's a great idea to utilize an IDE for enormous projects. However, there will be times where you will be required to use the terminal as a developer. You will live in the terminal world while enrolled in Operating Systems. The below definition gives a brief synopsis:

"The command-line shell is a text-based user interface for your operating system. Unlike a GUI shell, which uses a graphical representation of the system's services and resources, the command-line shell uses only text. It is the fundamental interface between you and the operating system, and in many cases offer you more direct control over the system processes."

Your goal in this portion of the lab is to pass in the *filename* when running your program. To achieve this, you can do the following:

1. Open up terminal and navigate to your project directory to compile it using *g++*
   a. Refer to the *Windows Subsystem for Linux (WSL)* **page** in Canvas
2. Run your program and pass in the filename as a parameter
   a. I.E. "*./main.out* **lab5_sample.txt**"
      i. You should *throw -1* if the file doesn't exist.

From this, you should observe that testing is much faster. Make sure to refer to the *Useful Resources* section as it contains helpful articles.

Submission

zyBooks: Follow the normal procedure as you've been for these previous labs.

Canvas: Submit **only** your source codes in a zip folder (i.e. don't include any .out/.exe files). The zip folder name should be: **lastname_firstname_L5** (i.e. rashad_kaji_L5). Within the folder, you should include a snapshot of successfully compiling and running the program. You can run a test case having a short output to fit it on your screen. In the snapshot, write your name using any clipart editor. The ETs will test by compiling your code with g++ then passing in a filename. We will use any of the sample or actual test files. We will test it with a non-existent filename in the directory.

Tips

1. Choices you make at the start of a program can have a big impact on how the rest of the program gets developed. Think about how you want to store the information retrieved from the file, and how you could easily pass that data to various functions you might write.
2. If you have a process for easily loading and accessing the data, the rest of the functionality should be a lot easier to write. Make sure the loading process is all taken care of before worrying about anything else.
3. The code to load 1 file containing 1 piece of data (no matter how complex that data is) should not be much different than loading 100 files, each containing 100 elements. Start by thinking about just 1 element from the file first. Do the values you read match the values in the file? What about 2 entries, does everything add up? Then 3, 4, etc.
4. When passing containers of data, make sure you pass them by **REFERENCE**, not by value. Creating copies of massive data sets is generally a bad, bad thing… unless you specifically need to duplicate the data. If not? Then pass by reference (in C++ that means either by pointer or the reference data type)
5. There may be a fair amount of repetition in a program like this. Think about where you can create helper functions to reduce the number of times you write the exact same (or just slightly different) code.

Useful Resources

1. Linux Shell Tutorial: https://www.computerhope.com/issues/chshell.htm
2. Mac Terminal Cheat-sheet: https://github.com/0nn0/terminal-mac-cheatsheet
3. Windows Terminal Cheat-sheet: http://simplyadvanced.net/blog/cheat-sheet-for-windows-command-prompt/
4. Command Line Arguments in C/C++: https://www.geeksforgeeks.org/command-line-arguments-in-c-cpp/