# Collaboration Policy

By submitting this assignment, you are acknowledging you have read the collaboration policy in the syllabus for projects. This project should be done individually. You may not receive any assistance from anyone outside of the CS 251 teaching staff.

# Late Policy

You are given the grace period between the deadline and the first lecture of the following day (9am). You do not need to let anyone know you are using this grace period. Beyond this period, no late submissions will be accepted.

# What to Submit

*main.cpp* and *ourvectorAnalysis.txt*

Your ourvectorAnalysis.txt should include your analysis of your usage of vectors.

**Do not submit any additional files.** We have all the files that we provided to you and you should not be changing those files.

# Where to Submit

zyBooks - main.cpp (Implementation Scores)

Gradescope both main.cpp and ourvectorAnalysis.txt (Style and Vector Analysis scores)

# Table of Contents

[Project Summary](#_ez5exh7trh5i)

[Example Execution](#_42g7cqgfxvcg)

[Data Files](#_53wg4mh8xs38)

[Congressional District Data: districts.txt](#_z74i5tj5egyv)

[Number of eligible voters in each state: eligible\_voters.txt](#_ltgbh3x519td)

[Calculating Gerrymandering](#_od20re11kmyj)

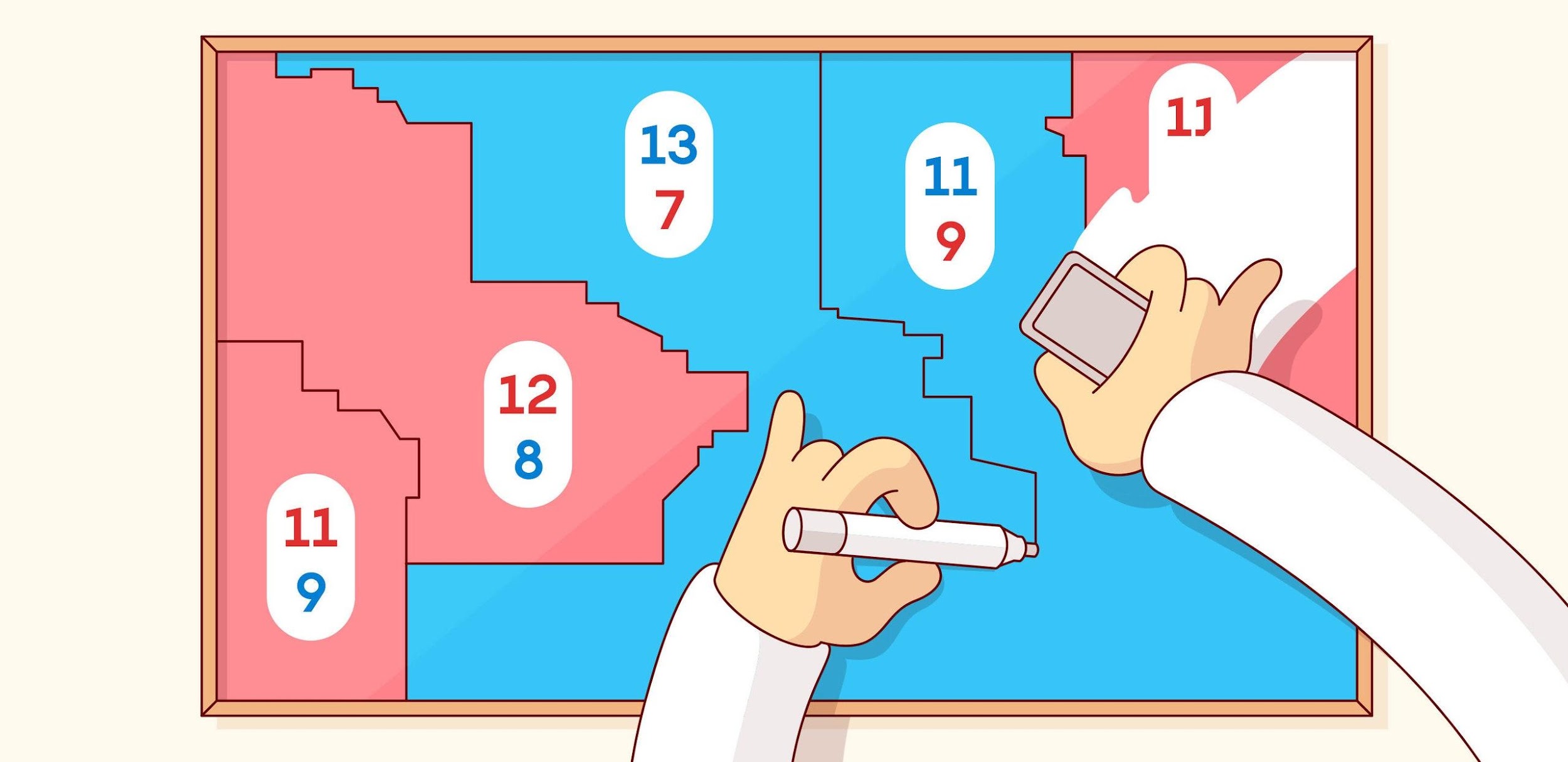
[More Details about plot command](#_h0iv5izcpi3)

[Ourvector Analysis](#_20j9rleepq7o)

[String Library in C++](#_oa2p4flcyiv4)

[Programming Environment and Standard Version](#_weqwkb89tvkv)

[Requirements Recap](#_84y3dp8mkrl7)



**Image credits:** <https://www.wired.com/story/the-math-behind-gerrymandering-and-wasted-votes/>

# Project Summary

Recently gerrymandering has been a popular topic in the news and the subject of an ongoing Supreme Court case. Gerrymandering is creating voting districts in such a way that gives one party an unfair advantage over the other. Political parties do this to try to help themselves stay in power. Recently, a research group came up with a mathematical definition for what constitutes gerrymandering. For this project, you will write a code that determines whether the state’s districts are gerrymandered, according to the researchers’ definition. The input data about states’ districts and total voters comes from two input files. Even if you disagree with this definition of gerrymandering, it is interesting to understand it better as the courts are currently debating it.

| You are going to write a program that can:   1. ***load*** in voting data (and display it to the screen), 2. ***search*** the data for a particular state, 3. ***plot*** all the districts’ votes of that particular state, 4. display the calculated ***stats*** for that particular state. | [Starter Code with Example Input & Output](https://drive.google.com/drive/folders/15SXKvgfs2WiV3kNi1OCPBSDMH1kVNPmo?usp=share_link)  Full potential output and some edge cases: [Sample Output Text](https://drive.google.com/file/d/1c7uLU-8Oa6ZEJym1okuO_LIT6aUALfXi/view?usp=share_link) |
| --- | --- |

Pay close attention to the formatting of the output, it will need to be exact to pass the test cases and receive credit. The best way to do this is through the test cases when they are available. These will highlight differences between your output and the expected output. To get started an example input and output text file are provided. Note that states that are gerrymandered have more stats printed when the “stats” command is called. See details in test cases.

***Only Read on Load***

You may only read each file once and file reading is only allowed when load is called. The data should all be stored in vector(s) (using **ourvector** only) to be used for the remaining commands. You will need to submit an analysis of how many vectors are created and justify how you minimized the amounts. See the ourvector analysis section for details.

From the examples, you will notice that searching for the state should be case insensitive. Therefore, you will need to be consistent with capitalization during storage and search.

# Example Execution

Here is a quick tour of one execution of the program (console input is in **red**):

Welcome to the Gerrymandering App!

Data loaded? No

State: N/A

Enter command: **load districts.txt eligible\_voters.txt**

-----------------------------

Reading: districts.txt

...Maryland...8 districts total

...

...Pennsylvania...18 districts total

Reading: eligible\_voters.txt

...Alabama...3606103 eligible voters

...

...Wyoming...431011 eligible voters

Data loaded? Yes

State: N/A

Enter command: **search illinois**

-----------------------------

Data loaded? Yes

State: Illinois

Enter command: **stats**

-----------------------------

Gerrymandered: No

Wasted Democratic votes: 921521

Wasted Republican votes: 850785

Eligible voters: 8983758

Data loaded? Yes

State: Illinois

Enter command: **plot**

-----------------------------

District: 1

DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDRRRRRRRRRRRRRRRRRRRRRRRRRRR

…

District: 18

DDDDDDDDDDDDDDDDDDDDDDDDDRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR

Data loaded? Yes

State: Illinois

Enter command: **exit**

-----------------------------

# Data Files

## Congressional District Data: districts.txt

Each line of districts.txt contains a state name followed by district information in groups of three. The first of the three is the district number, the second is the democratic votes in that district and the third the republican votes in that district. Some states do not number the districts. For example, any state with one district has a district “number” AL. Search the text file to see for yourself. To keep things simple, you should store this single district with the number 1 instead of AL, which is also how it will print in the console. See test cases for details. Depending on the size of the state, there will be a different number of districts. For example:

Arkansas,1,63555,124139,2,103477,123073,3,0,151630,4,87742,110789

Alaska,AL,114596,142566

Rhode Island,1,87060,58877,2,105716,63844

Notice that the data is separated with commas (“,”) instead of the spaces. We are using commas because some state names contain spaces and we want to preserve those spaces. You should use C++ strings and C++’s string library functions. You should also use C++ streams for file reading. It is recommended to use **getline** instead of the extraction operator (>>) for the file reading. See lecture notes and lab for file reading template code. The district numbers and the vote counts will always be integers.

## Number of eligible voters in each state: eligible\_voters.txt

Every state name in districts.txt is also in eligible\_voters.txt, so you do not need to worry about a state having district data but no eligible voter data. If there is a state in eligible\_voters.txt, but not contained in district.txt, that state should be ignored (should not be stored in your vector, should not be found when search is called, and should not be printed out during load). Each line of eligible\_voters.txt contains a state name, followed by the eligible voter count for that state. For example:

Alabama,3606103

Alaska,519501

Arizona,4738332

Arkansas,2148441

California,25278803

**Note**: Your code will be tested on other files of the same format. Do not assume that the files only have some number of lines, only have US state names, or anything else about the files other than what has been described here about their format. Two files will always be provided to loading, the first districts and the second eligible voters but the filenames may change from the examples (e.g. load 1.txt 2.txt would contain districts in 1.txt and voters in 2.txt)

**Note 2:** Think about how you are going to store this data. Think about it…now. Don’t get too deep into file reading and string parsing and calculations before reading through the rest of the project and deciding where you want to put all this. One ourvector? Multiple ourvectors? Structs? The choice is yours. No abstractions or data structures are allowed other than ourvector. A little planning will go a long way!

# Calculating Gerrymandering

You can determine gerrymandering for states with three or more districts by counting up and comparing the wasted votes cast for each party. For states with less than three districts, you cannot determine gerrymandering. We will define a wasted vote as any vote not needed to win the election. That means all votes for the party that loses the district seat are wasted as well as all votes for the winning party other than the “over half” they need to win the majority. The “over half” is defined as the least number of votes that is over half of the total number of votes in the district. For example, if there were 10 total votes, then 6 would be over half. If there were 9 total votes, then 5 would be over half.

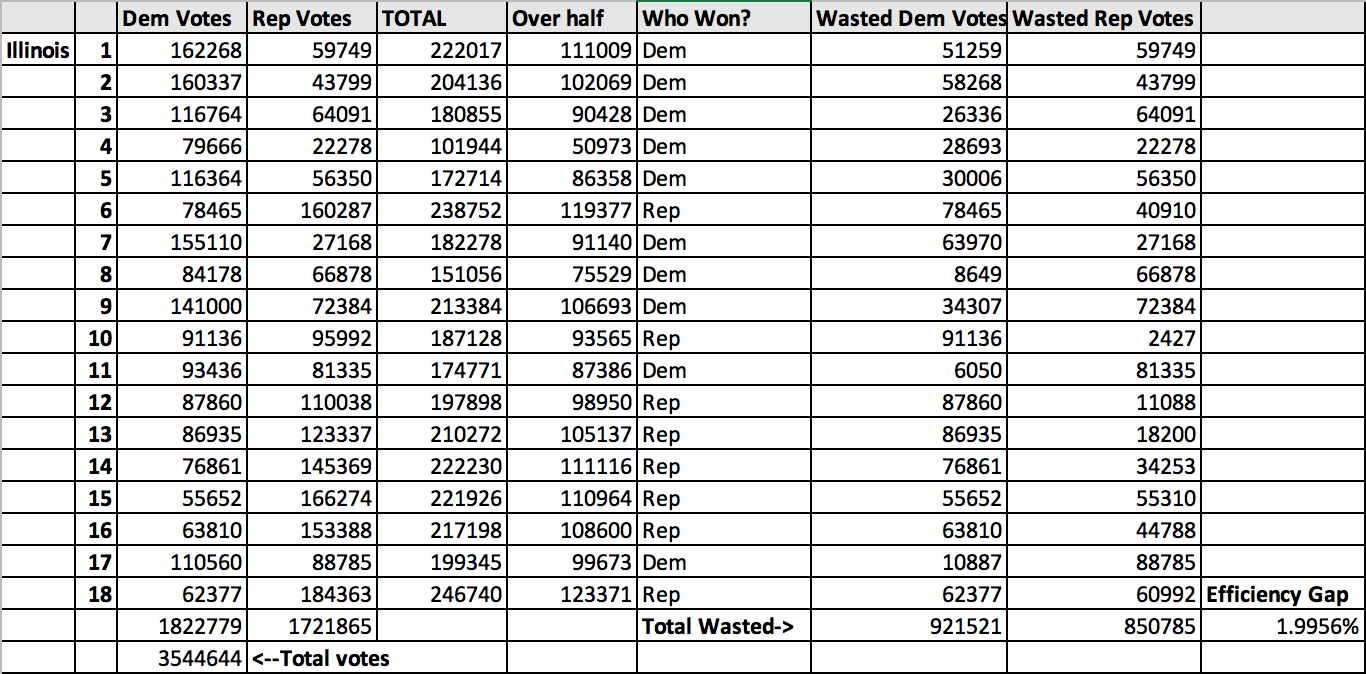
Having calculated this data, we can sum up the wasted votes for each district. It is impossible to make voting districts exactly fair and so we shouldn’t expect the wasted vote counts to be equal. We define the **efficiency gap** as the difference in the totals of the wasted votes for each party, expressed as a percentage of total votes cast.

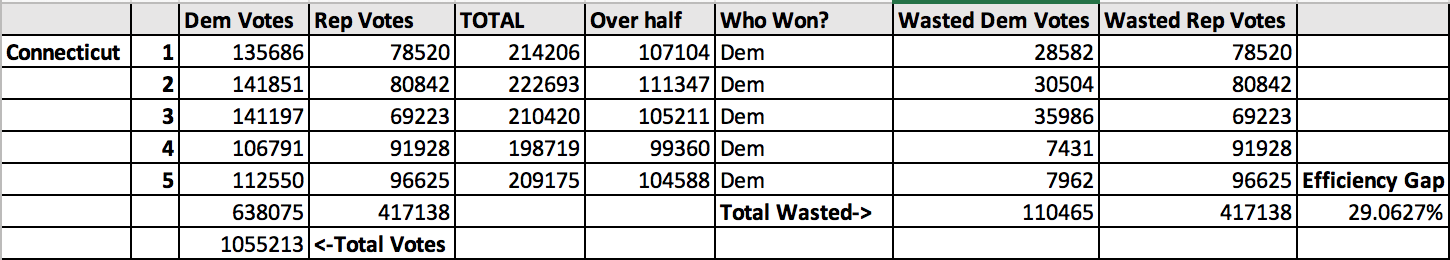
*NOTE: We subtract the smaller number from the larger when possible, to ensure a nonnegative efficiency gap. We could also take the absolute value of the difference.*

*NOTE2: Total votes cast is not the same as eligible voters per state. The total votes cast must be calculated by adding all democratic and republican votes in all of a states’ districts.*

Researchers have discovered that it is almost impossible for the disadvantaged party to recover if the efficiency gap is greater than or equal to 7 percent. Therefore, the researchers, as well as us for the purposes of this assignment, will consider a state gerrymandered when there is a 7% or greater efficiency gap. States with less than three districts cannot be gerrymandered.

To check your work and calculations (and to further understand the two paragraphs above), please see the following tables for intermediate calculations for two sample states (Illinois and Connecticut). This will help you verify your calculations are being done the same as the solution your code is tested against. If you want to read more about this, [read this article](https://www.wired.com/story/the-math-behind-gerrymandering-and-wasted-votes/).





# More Details about plot command

The “plot” command will visualize the votes for all districts in a state. As a reminder from above, it looks like this:

[Sample Output Text](https://drive.google.com/file/d/1c7uLU-8Oa6ZEJym1okuO_LIT6aUALfXi/view?usp=share_link)

For each district, you are going to print 100 letters (either D’s or R’s). The D’s will be on the left and the R’s will be on the right. The plot is intended to look like a text-based histogram. The way you will determine how many D’s and how many R’s will be described with an example. For Illinois, District 1 has 162268 democratic votes and has 59749 republican votes. This means that 73% (162268/(162268+59749)= 0.7308809686) of District 1 are democratic votes and 27% of District 1 are republican votes. Your plot then should print 73 D’s followed by 27 R’s for District 1 (which is what is shown above). In order to stay consistent with the solution, calculate the percentage of Democratic votes and always round *down* to the nearest integer. Then, give the remaining percentage of the votes to Republicans. *NOTE: I realize rounding properly makes more sense, but this is just easier.*

# Ourvector Analysis

Open ourvector.h and go to the member function \_stats. Make sure this portion of the code is not commented out:

cerr << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cerr << "ourvector<" << name << "> stats:" << endl;

cerr << " # of vectors created: " << Vectors << endl;

cerr << " # of elements inserted: " << Inserts << endl;

cerr << " # of elements accessed: " << Accesses << endl;

cerr << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

Run your code (make sure to run all commands) with this version of ourvector.h, and you should see some vector stats print at the bottom of your code:

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**ourvector<T> stats:**

**# of vectors created: <X>**

**# of elements inserted: <Y>**

**# of elements accessed: <Z>**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

Submit a file named “ourvectorAnalysis.txt”. Copy and paste what you get for the report above into your text file (include all input, output, and the ourvector report). In it you must write approximately 250 words to justify why, when, and how your code has created X number of vectors, has inserted Y elements, and accessed Z elements. Your analysis should be specific and quantitative. For the vectors created, you should identify all line numbers of your code where those vectors are created. You should be specific enough that your counts add up to exactly the amount of vectors. For the number of elements inserted and number of elements accessed, you can estimate more approximately based on the sizes of the vectors and the types of operations you are doing.

Your code should minimize the number of vectors created, elements inserted, and elements accessed. One way to do this is to make sure you are passing by reference. Another way is avoid searching too often (you should only search during load and search). Each piece of state data should only be stored exactly once, and you should justify that this is the case. You may have more than one ourvector, for example, if you want to store each state’s district data in its own ourvector. Structs are encouraged!

# String Library in C++

You’ll need some string processing, namely finding characters within a string, and extracting a substring. While you can certainly write these functions yourself, it’s expected that you’ll use the .find() and .substr() functions built into the **string** class provided by C++: <http://www.cplusplus.com/reference/string/string/>.

**Don’t forget to #include <string>.**

Your solution is required to store all data in a **vector<T>** class --- to be precise, in a vector-compatible implementation we are providing: **ourvector<T>**. For the purposes of this assignment, always start with an empty vector, and then add data to the vector by calling the **push\_back()** member function. When you need to access an element of the vector, use the **.at()** function, or the more convenient and familiar **[ ]** syntax. To empty a vector, use the **.clear()** function. For more info on vector<T>: <http://www.cplusplus.com/reference/vector/vector/>. Don’t forget to #include “ourvector.h” (it is already in the starter code).

Finally, to work with files, #include <fstream>. To read from a file, use an **ifstream** object, and use the **>>** operator when inputting a single value (e.g. integer or single word). When you need to input 1 or more words into a single string variable, use **getline(infile, variable)**.

# Programming Environment and Standard Version

We are compiling via **g++** with **-std=C++20** or **-std=c++2a**. Do not ask us to change the C++ version; we are compiling against C++ 2020 version within the automatic testing areas.

# Requirements Recap

1. You must use ourvector<T> (“ourvector.h”) for storing all data. No other data structures may be used.
2. You are allowed to use and add other libraries (make sure to include them at the top of your file). Some you might find useful: <iostream>, <sstream>, <string>, <fstream>, <math.h>, <algorithm>, etc. You may not need some of these, there are lots of ways to solve this problem.
3. Each input file may be opened and read exactly once; store the data in ourvector<T> if need be. The file reading must occur only when load is called.
4. The algorithm for searching for a state inside your vector must be written by you, no library functions allowed.
5. You should be able to identify and count all vectors created. The # of inserts and the # of accesses must be reasonable. You should determine that yourself and justify in your submission (in “ourvectorAnalysis.txt”).
6. Your main.cpp program file must have a header comment with your name and a program overview. Reference the lecture and live coding examples for comment examples. Each function must have a header comment above the function, explaining the function’s purpose, parameters, and return value (if any). Inline comments should be supplied as appropriate; comments like “declares variable” or “increments counter” are useless. Comments that explain non-obvious assumptions or behavior **are** appropriate.
7. Each command (load, search, stats, plot) must be implemented using a function; this implies a complete solution must have at least 5 functions (4+main). However, a good solution will have many more than 5 functions to decompose your code properly.
8. No global variables; use proper parameter passing and proper function return values.
9. The cyclomatic complexity (CCN) of any function should be minimized. In short, cyclomatic complexity is a representation of code complexity --- e.g. nesting of loops, nesting of if-statements, etc. You should strive to keep code as simple as possible, which generally means encapsulating complexity within functions (and calling those functions) instead of explicitly nesting code.

| **Here’s an example of with low CCN:**  while (…) {  if(searchFunctionFindsWhatWeNeed(…))  doSomething();  next value;  } | **Here’s an example with high CCN:**  **while** (…) {  **for** (…) { // loop to search  if { (search condition met)  **for** { (…)  // now do something:  …  }  }  }  next value;  } |
| --- | --- |

As a general principle, if we see code that has **more than 2 levels** of explicit looping --- an example of which is shown above --- that score will receive grade penalties. The solution is to move one or more loops into a function, and call the function.

**Citations/Resources**

Assignment is inspired by Allison Obourn (University of Arizona), Marty Stepp (Stanford University), and Shanon Reckinger (University of Illinois Chicago).

More information about Gerrymandering calculation:

<https://www.wired.com/story/the-math-behind-gerrymandering-and-wasted-votes/>

**Copyright 2023 Adam T Koehler, PhD - University of Illinois Chicago**

This assignment description is protected by [U.S. copyright law](https://en.wikipedia.org/wiki/Digital_Millennium_Copyright_Act). Reproduction and distribution of this work, including posting or sharing through any medium, such as to websites like chegg.com is explicitly prohibited by law and also violates [UIC's Student Disciplinary Policy](https://dos.uic.edu/community-standards/academic-integrity/student-academic-resources/)(A2-c. Unauthorized Collaboration; and A2-e3. Participation in Academically Dishonest Activities: Material Distribution).

Material posted on [any third party](http://guides.dml.georgetown.edu/c.php?g=904530&p=6510961) sites in violation of this copyright and the website terms will be removed. Your user information will be released to the author.