

Project 1 – Gerrymander

CS 251, Fall 2020, Reckinger

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 a total of 5 late days to use at your discretion (for all projects, not each project). You may use the late days in 24-hour chunks (either 1 day at a time or all five at once). You do not need to alert the instructor to ask permission to use late days. You can manage your use of late days on Mimir directly.

What to submit: `main.cpp` and `ourvectorAnalysis.txt`. Your `main.cpp` should include your own Creative Component. Your `ourvectorAnalysis.txt` should include your analysis of your usage of vectors. Please note there have been a few changes to the data files since this was released so you might want to redownload the data files from starter code linked below. If you submit your folder, your code will not run. If you submit only `program.exe`, your code will not run. If you submit all the files in your folder, the autograder will be ignoring all files except `main.cpp`, but will run.

Project Summary

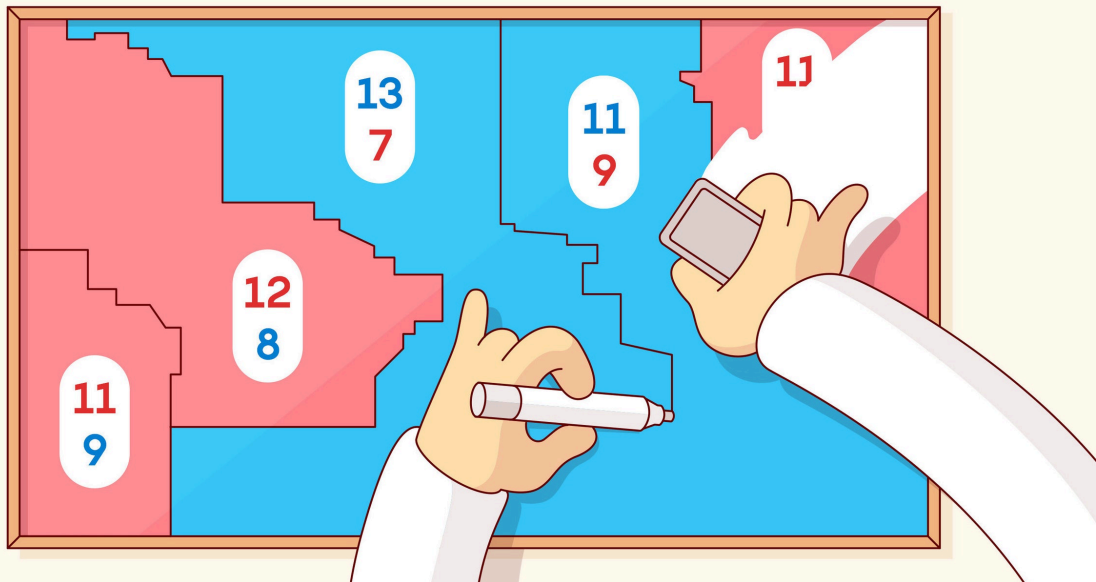


Image credits: <https://www.wired.com/story/the-math-behind-gerrymandering-and-wasted-votes/>

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),

- Please see the test cases to see the formatting of the console input/output. Here is a quick tour of approximately what your program should look like (console input is in **red**, the rest is console output):

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 created (and justify that it is minimized). See below for details.

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

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 insertion 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 ignore (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.

Note 2: Think about how you are going to store this data. Think about it...now. Don’t get too deep in 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 ourvecotor. 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 this tables below 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](#).

		Dem Votes	Rep Votes	TOTAL	Over half	Who Won?	Wasted Dem Votes	Wasted Rep Votes	
Illinois	1	162268	59749	222017	111009	Dem	51259	59749	
	2	160337	43799	204136	102069	Dem	58268	43799	
	3	116764	64091	180855	90428	Dem	26336	64091	
	4	79666	22278	101944	50973	Dem	28693	22278	
	5	116364	56350	172714	86358	Dem	30006	56350	
	6	78465	160287	238752	119377	Rep	78465	40910	
	7	155110	27168	182278	91140	Dem	63970	27168	
	8	84178	66878	151056	75529	Dem	8649	66878	
	9	141000	72384	213384	106693	Dem	34307	72384	
	10	91136	95992	187128	93565	Rep	91136	2427	
	11	93436	81335	174771	87386	Dem	6050	81335	
	12	87860	110038	197898	98950	Rep	87860	11088	
	13	86935	123337	210272	105137	Rep	86935	18200	
	14	76861	145369	222230	111116	Rep	76861	34253	
	15	55652	166274	221926	110964	Rep	55652	55310	
	16	63810	153388	217198	108600	Rep	63810	44788	
	17	110560	88785	199345	99673	Dem	10887	88785	
	18	62377	184363	246740	123371	Rep	62377	60992	Efficiency Gap
		1822779	1721865			Total Wasted->	921521	850785	1.9956%
		3544644	<--Total votes						

		Dem Votes	Rep Votes	TOTAL	Over half	Who Won?	Wasted Dem Votes	Wasted Rep Votes	
Connecticut	1	135686	78520	214206	107104	Dem	28582	78520	
	2	141851	80842	222693	111347	Dem	30504	80842	
	3	141197	69223	210420	105211	Dem	35986	69223	
	4	106791	91928	198719	99360	Dem	7431	91928	
	5	112550	96625	209175	104588	Dem	7962	96625	Efficiency Gap
		638075	417138			Total Wasted->	110465	417138	29.0627%
		1055213	<--Total 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](#)

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.*

Creative Component

As a final functionality requirement, you must write your own command of your choice. Therefore, in addition to your code responding to exit, load, search, stats, and plot, you must develop your own command and its functionality. This open-ended piece will be graded using the rubric item in Mimir. For full credit: (1) you must have instructions in the header comment at the top of your file on how to use your command; (2) must use data provided in two data files given (no additional files allowed); (3) must be either a calculation(s) or a visualization of data; (4) must be relevant to the project and interesting; (5) the output must look nice and be well thought out; (6) may be per state or not per state; (7) must be original and your own work, your work should not be similar to others; (8) extraordinary work may receive up to 5% extra credit with instructor approval.

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 5 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 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!

Make sure when you submit your code that your ourvector.h has the cerr ... lines commented back out.

Learning 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

You are required to program your project in Mimir environment. Mimir allows us to all work in the same space and allows all teaching staff to best assist you. It also minimizes messing with cross platform problems. Mimir tracks your progress as you write the code, which will assist us when grading your work.

Mimir is set up to automatically use your highest scoring submission.

On Mimir, we are compiling via g++ with **-std=C++11**. Do not ask us to change the C++ version; we are compiling against C++ 11.

Requirements

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. 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, <yourCommand>) must be implemented using a function; this implies a complete solution must have at least 5 functions. However, a good solution will have many more than 5 functions to decompose your code properly.
8. No global variables; use parameter passing and function return.
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 function) instead of explicitly nesting code. Here’s an example of simpler code with low CCN:

```
while (...) {
    if (searchFunctionFindsWhatWeNeed(...))
        doSomething();

    next value;
}
```

Here’s an example of complex code with high CCN:

```
while (...) {
    for (...) { // loop to do search
        if { (search condition is met)
            for { (...) // now do something:
                ...
            }
        }
    }

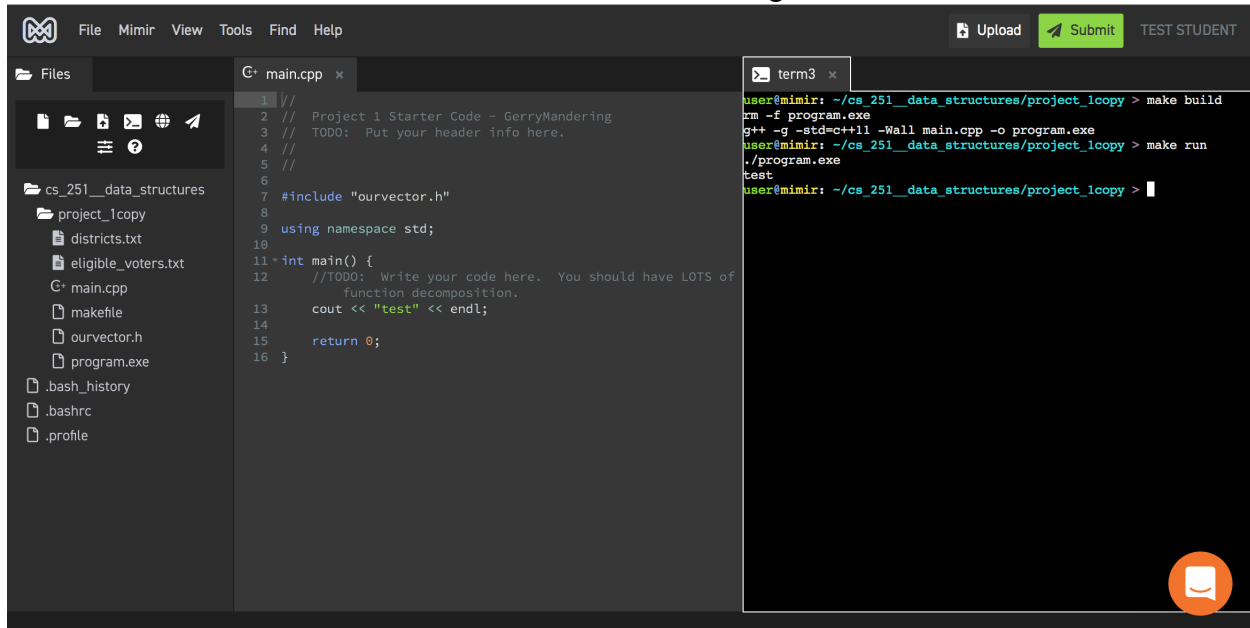
    next value;
}
```

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.

Getting Started in Mimir

Click on “Open in Mimir IDE”.

Click on the course, then the project folder (yours will be called project1_gerrymandering, below it shows project1_copy). Inside of the Project 1 folder, up the main.cpp file and add a few cout statements to the code. Look at how the terminal below AND the directory on the left from which we opened main are the same. Don't forget to save your code, it does not save automatically. To compile, type “make build” in command window. To run, type “make run”. You will need to save the file, compile, and run each time you make changes to your code. To test your code against the test cases and submit, click submit. Make sure to click the correct code to submit and the correct test cases to submit against.



```
1 //
2 // Project 1 Starter Code - GerryMandering
3 // TODO: Put your header info here.
4 //
5 //
6
7 #include "ourvector.h"
8
9 using namespace std;
10
11 int main() {
12     //TODO: Write your code here. You should have LOTS of
13     //function decomposition.
14     cout << "test" << endl;
15     return 0;
16 }
```

```
user@mimir: ~/cs_251__data_structures/project1_copy > make build
rm -f program.exe
g++ -g -std=c++11 -Wall main.cpp -o program.exe
user@mimir: ~/cs_251__data_structures/project1_copy > make run
./program.exe
test
user@mimir: ~/cs_251__data_structures/project1_copy >
```

NOTE: Once you have the IDE open, you may want to adjust the settings. Click File->Settings. We recommend unchecking the box, “use spaces instead of tabs”.

TIP: Ctrl+C will cancel a run command if you are stuck in a stream.

*solution.exe

The link above links to the executable associated with the solution code. Download the file to your local machine. Upload it to your Mimir project1 folder. Inside the terminal, type "chmod a+x solution.exe". Then type "./solution.exe" to run the solution. You can now use this if you want to see the behavior of the solution code (rather than reading the sample text or using the test cases).

Citations/Resources

Assignment is inspired by Allison Obourn (University of Arizona), Marty Stepp (Stanford University).

More information about Gerrymandering calculation:

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

Copyright 2020 Shanon Reckinger.

This assignment description is protected by [U.S. copyright law](#). 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](#) (A2-c. Unauthorized Collaboration; and A2-e3. Participation in Academically Dishonest Activities: Material Distribution).

Material posted on [any third party](#) sites in violation of this copyright and the website terms will be removed. Your user information will be released to the author.