User Guide:

The user can start the program by using the following command:

```
./mvote -f registeredvoters -b primarybucketsize
```

where “-f” is the flag for the file of voters and “-b” is the flag for the size of the primary buckets. These flags are sent to functions like loadfile() and lineHash() to make sure the program runs the way the user wants it to. The value I used when developing the code was 10, as I felt that it provided a good balance between speed and space efficiency. When the user starts the program, he or she sees a menu with all of the possible commands.

Program structure:

We were tasked with creating a system to use a hash table to manage voter data and votes cast. Linear hashing is proposed for handling collisions. So, I created a hash table that uses linear hashing.

LinearHash combines vectors and linked lists to form a hashtable. Linear hashing requires the dynamic resizing of a table, which is implemented using vectors. The "pair" objects are stored within each index of the vector. The primary linked list and the overflow linked list are the two main parts of the Pair object. The linear hashing algorithm is run when the linear hashing insert and split functions are used together. The hash function is the same as well, in which the index is calculated based on the round and initial buckets. The logic of the splitting algorithm is the same as the one shown in the lab.

The other main component of the program is the postal linked list. It is used to store all the people who have voted so far. This is a separate data structure from the regular linked list. It also houses other attributes, such as the size and zip code that each node corresponds to. The voter's information is stored in the "Voter" class, which contains all their credentials, such as their PIN, zip code, and first and last name. These voter objects are what is stored in each node within the linear hash table and postal linked list. Whenever a voter changes their vote status to "Y," they are added to the postal linked list. Each node within this linked list holds another regular linked list that holds user information in the form of voter objects.

Each linked list node contains a pointer to the next node and all voter information. As a starting point, I defined a capacity of 10 for the primary bucket in the hash table. This results in a consistent distribution voters within the buckets. The number of votes cast from each zip code is kept track of by the size variable.

Design Choices:

Vectors were chosen for the linear hashing class because the index and hash functions make it easy to find specific voters quickly.

To sort the votes and zip codes for the "o" option, the zip code node sizes are loaded into an array of ZipSize objects, and a simple sort method is used on that array. Sorting is not done every time a vote is cast, but only when the user asks for it. As stated in the project's description, this sort method puts the array's elements in order from greatest size to least size. By doing this, the program requires less memory for the ordering operation. To sort the linked list directly, you would have to move pointers that the user doesn't need for the "o" command.

Remarks:

It has been explained to us that we are not allowed to use the STL for strings because it is not in C. I have decided to still use string.h in my code for the sake of simplicity and convenience, and also due to the fact that it is not possible to use fstream or sstream with a user-defined function. Upon further research, I found that the string.h library is actually available within the Standard Library in C, so I opted to use that instead of string, which is C++ exclusive. I hope that you will consider not deducting grades for this decision.

Credits:

1. The Linear Hashing example repo
2. Chatgpt for the basic function definitions such as vector, linked list, and entering the “-f” and “-b” flags through the terminal. Also ideation for how to execute linear hashing as well as the sorting algorithm.
3. menu() function design was taken from one of my own previous Data Structures assignments.