User guide:

The program facilitates the implementation of a vote monitoring system where the user has the options to look up, modify, or purge records, or access polling statistics.

The user will invoke the program using the following invocation and flags:

```
./mvote -f registeredvoters -m hashtablesize
```

where -f is the flag for file of voters and -m is the flag for size of hashtable. Providing valid arguments is essential for the program to run.

The program presents the user with a menu of available commands with their required parameters. Upon receiving the input of the command, the program executes the relevant operations and displays the output to the terminal. This process continues until the user decides to exit the program. Further, the user must provide valid inputs of commands and their parameters or else they will be presented with an error message.

Technical implementation:

The program keeps track of the records by using two different structures.

The first structure is a hash table with chained hashing, where the chains of each location in the hash table are implemented using a singly linked list. This structure stores the records of all the voters included in the registeredvoters file provided by the user.

The second structure stores the records of voters who have cast their vote by their zip code. The implementation is based on a singly linked list of singly linked lists where the outer linked list (postalCodeLinkedList) is a linked list of postal codes where votes have been cast. Each node in this linked list points to another linked list (voterLinkedList) where each node points to a voter's profile (or node) who has already voted in the hash table.

Thus, the structures allow the tracking of the poll's progress.

Further, the profile of a voter is stored in a struct called voterProfile to allow for easy access for a voter's information. Here, a Boolean data type for the voted status is used, where false indicates "no" and true indicates "yes".

The linked lists are somewhat similar in structure except for their node types. The different operations facilitated by the lists are search, removal, modification, and access to specific nodes using some *key* as well as typical linked list operations such as adding to and removing from the front of the linked list.

The hash table and the linked lists all use dynamically allocated memory, which is then deallocated appropriately using the class destructors.

Further, specifically selecting the return type of member functions of these classes also allows for checking if the command was implemented or not for some reason. This then allows for appropriately selecting further progress in the program, or giving an appropriate message to the user. Examples include the Boolean return type of modification and deletion functions in linked list classes.

The two structures are created in the main() function of the program and are passed as arguments to *menu* function by reference, where they are then modified according to the user input command.

Further, the user inputs from commands are always checked for validity of commands and the parameters, as well as missing parameters. If either of the aforementioned is the case, the user is presented with the appropriate message instead of executing the command. Only if the user provides a valid command and parameters will the command be executed.

Design choices:

The use of classes and functions allows for reusing of code. Furthermore, considering that most operations take O(n) in their worst case (such as look up operations), the sorting algorithm for the 'o' command was also chosen to be suitable in terms of its runtime. Here, a heapsort is implemented to display an ordered output of number of votes cast by post code so that the worst case outcome is O(nlogn) for sorting. Within the sorting algorithm, instead of sorting the linked list every time a new vote is cast, the sorting is performed only when demanded by the user. Further, the ordering is performed by storing the relevant information in a dynamic array of a struct called sortedStruct. This structure only contains information about the postcode and the number of votes cast in it. This way, less memory is required by the program for the ordering operation since sorting the linked list directly would require also moving around (possibly copies of) information regarding nodes such as pointers that are not required by the user for the particular command 'o'.

Credits:

1. The basic structure for the hashtable.h and hashtable.cpp has been adapted from the starter code of lab12 of the Data Structures course by Professor Sebti Foufou I took in Spring 2020, filename "lab12_part2.cpp".
2. The basic structure for the linked lists has been adapted from the starter code of lab 5 of the Data Structures course by Professor Sebti Foufou I took in Spring 2020, filename "lab5.1_sol.cpp".
3. The removal of a particular node from singly linked lists has been adapted from GeeksForGeeks. https://www.geeksforgeeks.org/linked-list-set-3-deleting-node/