I have to develop an application termed mvote (monitoring a vote), that lets a manager monitor a poll. In this, a number of qualified votes can participate. I will have to create structures that facilitate the fast access of information pertinent to the poll. My application will be based on both hashing and linked-lists that jointly help rapidly assess the progress of a poll.

Things my program should do:

1 - manage a number of qualified poll-participants. For simplicity, assume that the information we have for every voter is her first and last name, RIN (resident ID number), the postal code of residence and whether she has voted thus far. Initially, all participants are marked as not having voted.
2 - create a hash-table based on chained-hashing that allows the poll-monitor to store, manipulated and access information about the status of voting based on the RIN of every individual. The hash-table is dynamic for it can arbitrarily expand as voters are inserted into the structure.
3- maintain a dynamic structure that helps quantify the poll progress through the computation of statistics especially aggregate information provided on a per-zip-code fashion.
allow the monitor of the poll to enter, modify and even purge participants and their respective information from an ongoing session.
4- gracefully release all main-memory space dynamically allocated before the execution of mvote terminates.
Operations including look-ups, insertions/deletions of participants, change of voting status, and derivation of statistics are all launched through a prompt that your application should provide.

Features of mvote:

Your program should comply with the following basic characteristics:
1- Data available for poll participants entail: RIN, last and first name, zip code of residence.

2- The registry of the voters can be provided in terms of an ASCII file at commend line. Every line
contains the information of a single participant and the size of the file is not known a priori.

3- The monitor overseeing the poll can also manually at prompt enter and delete records of participants and record whether an individual has voted.

4- The RIN is used as key to enter participant data into the the hash-table used by mvote. Respectively, the zip-code numbers are used as key in maintaining record of which exactly votes have successfully casted her vote in each zip-code area.

5-Refrain from using static arrays and/or structures in your mvote implementation.

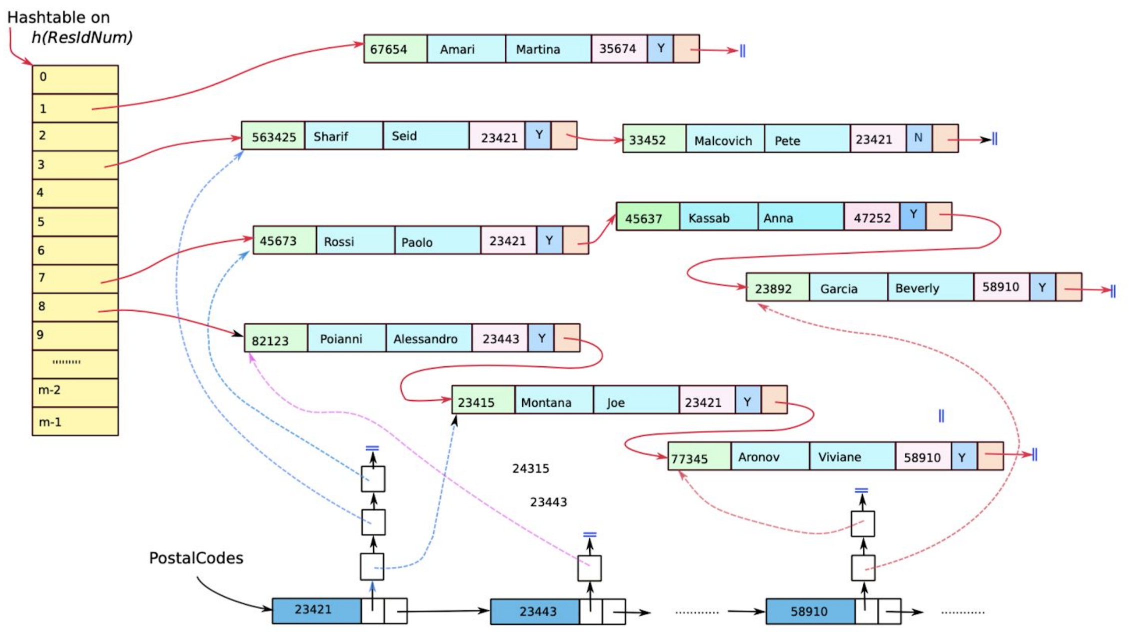
Hashtable on
h(ResIdNum)
0
1
2
3
4
5
6
7
8
9
''''''''
m-2
m-1
67654 Amari Martina 35674 Y
563425 Sharif Seid 23421 Y
33452 Malcovich Pete 23421 N
45637 Kassab Anna 47252 Y
45673 Rossi Paolo 23421 Y
23892 Garcia Beverly 58910 Y
82123 Poianni Alessandro 23443 Y
23415 Montana Joe 23421 Y
77345 Aronov Viviane 58910 Y
24315
23443
PostalCodes
23421
23443
58910
....
....

Figure 1: A sketch of the structures used by `mvote`

Figure 1:

Description of mvote Structures:
Figure 1 depicts the key features of the structure to be used in your program. On the top-left, you see the basic hash-table structure based on chained hashing that uses a function h() of your choice. Based on a careful selection of its size m, this table offers real quick look-ups. The structure nodes are created on-the-fly as we input records of information with each such record being uniquely identified by an RIN. Solid lines represent links between voters that hash in the same entry and are chained together. For example, records for Rossi, Kassab and Garcia all hash at value 7 and are chained off the respective element of the table.
In the lower part of Figure 1, a second structure appears that realizes groups of participants who have already casted their vote per zip-code. Dotted-lines depict the pointers that “connect” every single zip-code in which voters have casted their vote. If for whatever reason the manager decides to purge a specific voter, if there already exists a respective pointer in this linked-list structure of Figure 1 to just-deleted voter, the list should be properly cleared up. For example in zip-code 58910, Aronov and Garcia have voted already. The latter is indicated by a “Y”(yes) element found on the participant record. Those who have not voted so far but are legitimate participants have their respective status element marked as “N”(no).

How Your Application Should Be Invoked:
Your mvote application can use one or more flag options (or in-line prameters) that you may deem required. A possible invocation of your application would be:
mymachine-promt >> ./mvote -f registeredvoters -m hashtablesize
– mvote is the executable of the program that you will create,

– flag -f indicates the text-file named registeredvoters, containing an initial set of qualified poll participants,
– flag -m designates the size of the hash-table hashtablesize you will use as the main structure to host
your data.
Feel free to enhance your interface with additional flags that you may deemed useful in your implementation.

Evidently, parameters can be listed at invocation in a random order.
Once your application gets initialized, it then presents the poll manager with a prompt. Every time the user of the program (i.e., manager) types in a command, mvote should reciprocate by taking a required action. At prompt, your program should be able to handle the following commands:
1- l <key>
lookup the hash-table for a voter with id: <key>. If found, print the pertinent record out on the tty; otherwise, print an error indication.

2- i <rin> <lname> <fname> <zip>
inserts all information for a specific voter whose ID is <rin>, last and first names are <lname> and <fname> respectively, and she/he resides in <zip>. All inserted voters have their voting status initially set to NO. If <rin> already exists, abort and present an error message.

3- d <rin>
delete the voter with ID <rin>. If there is no such a voter on the structure, take no farther action and print a corresponding message.

4- r <rin>
register the voter with ID <rin> as having already voted by changing her status to YES. If <rin> is already marked as YES, no farther action is needed and a appropriate message is displayed.

5- bv <fileofkeys>
bulk-vote for all the keys (i.e., <rin>s) that appears in file fileofkeys>. For each attempted vote marking, the operation should have a similar behavior to that of “r <rin>” just above.

6- v
present the number of people marked as having voted so far.

perc
display the percentage of people whose vote has been as YES over the number of poll participants who are part of the hast-table.

7- z <zipcode>
print the number of all poll participants marked as having voted YES and list their ids one id per line.

8- o
produce a output list of zipcodes in decreasing order of the number of people who live in each zipcode and have marked as Y(es). One zipcode along with the number of voters marked as YES is displayed per line.

9- exit
terminate the program but before gracefully release all dynamically allocated memory.

10 - Any other command that may facilitate the development of the program. For example, you may find useful to print content structures in a meaningful way while developing mvote.

Please also provide a short write-up about the design choices you have taken in ordered to design your program(s); 1-2 pages in ASCII-text would be more than enough.

1 - You have to use separate compilation in the development of your program.

2- If you decide to use C++ instead of plain C, you should not use STL/templates.

3- It would be convenient to set the size of the hash-table m according to the size of (i.e., number of lines appearing in) file <registeredvoters> as this is offered in the invocation of your program. In this case, as flag -m has no purpose, it can be omitted and you have to describe how you actually determine the size m on the fly in your write up.