CS302

Assignment 7: Vote Counter

Description

Election season. A time when people gather to decide which public figure will represent a group of people at the regional or national level. The issue is that there are many people who are voting and using an automated system for voting would be ideal to count the ballots quickly. Using a fast search structure would be even more ideal. A hash map does seem to be the fastest search structure. For this assignment you will need to use a filestream to read in candidate info and voting ballots to determine who wins the election, you would need to create your own custom hash map type. Below is the structure you will need to implement. This will be a hybrid of chaining and linear probing, as in each index in the table can store up to 2 entries, if an index contains 2 entries and no match, a linear probe is done to check the next index in the able.

```
template <typename t1, typename t2>
class hashMap
{
public:
  struct hashPair
  {
    t1 key;
    t2 value;
    hashPair * link;
  };

  struct iteratorPair
  {
    t1 * key;
    t2 * value;
    iteratorPair * link;
  };

  class iterator
  {
  public:
    friend class hashMap;
    iterator();
    const iterator& operator++(int);
    bool operator==(const iterator&) const;
    bool operator!=(const iterator&) const;
    t1 first();
```

```
    t2 second();
  private:
    iterator(iteratorPair*);
    iteratorPair * element;
  };

  hashMap();
  ~hashMap();
  t2& operator[](t1);
  iterator begin() const;
  iterator end() const;
private:
  void resize();
  int h(std::string) const;
  int items;
  int size;
  hashPair ** table;
  iteratorPair * head;
};
```

Each member of the hashMap class contains/performs the following

- `struct hashPair` - contains the (key, value) pair for each entry in the hash table
- `struct iteratorPair` - contains pointer to an existing (key, value) pair (this will be used in a linked list of all entries currently stored in the hash table)
- `int items` - maintains a count of how many full elements exist in the table, i.e. if an element in the hash table contains 2 entries then you would increment the items counter
- `int size` - denotes the size of the hash table
- `hashPair ** table` - is the hash table, it's an array of linked lists
- `iteratorPair * head` - is the head pointer of the linked list that maintains all the existing entries in the hash table
- `hashMap<t1, t2>::hashMap()` - default constructor, sets the size to 5, items to 0, sets head with `NULL`, allocates the hash table `table = new hashPair*[size]` and then sets each element of table with `NULL`
- `hashMap<t1, t2>::~hashMap()` - destructor, deallocate the hash table (you first need to deallocate each linked list within table and then deallocate table), and then dellocate the linked list that head points to
- `t2& hashMap<t1, t2>::operator[](t1 key)` - the bracket operator that performs the insert/find for the hash map, you will perform the following steps
  1. Check the load factor, if the load factor is 50% or larger, then call the resize() function
  2. Call the hash function and store the result into an integer
     `int x = h(key);`
  3. Check location `table[x]`
     - Either this entry is empty, or it only contains 1 node, or it can contain 2 nodes
     - You will need to determine whether the key exists in this entry or if the key needs to be inserted into this entry
     - If the key exists (somewhere in this linked list), then return its value field

- If the key does not exist (in the linked list) and there is a vacancy (this location only contains 1 node), create the entry and create an entry in linked list of entries that head points to (if table[x] is now full i.e. it contains 2 entries increment items by 1) and then return the value field of the entry
- If a collision occurs (location contains 2 nodes and neither matches the key), then update x
  `x = (x + 1) % size;`
  and try step 3 again

- `typename hashMap<t1, t2>::iterator hashMap<t1, t2>::begin() const` - returns an iterator that points to the first node in the linked list that head points to
- `typename hashMap<t1, t2>::iterator hashMap<t1, t2>::end() const` - returns a null iterator
- `void hashMap<t1, t2>::resize()` - doubles the size of the hash table, needs to remap all the elements from the original table to a larger table
- `int hashMap<t1, t2>::h(std::string key) const` - hash function specific for when the key is of type `std::string`, this function adds up all the ASCII values of the string then mods it by the size and returns this value

Each member of the `iterator` class contains/performs the following

- `iteratorPair * element` - a pointer that points to a node in the linked list of entries that exist in the hash table
- `hashMap<t1, t2>::iterator::iterator()` - default constructor that sets `elements` with `NULL`
- `hashMap<t1, t2>::iterator::iterator(iteratorPair * p)` - constructor that sets `elements` with the address in `p`
- `const typename hashMap<t1, t2>::iterator& hashMap<t1, t2>::iterator::operator++(int)` - postfix operator that moves the iterator over to the next node in the linked list
- `bool hashMap<t1, t2>::iterator::operator==(const hashMap<t1, t2>::iterator& rhs) const` - compares if two iterators point to the same node, compares `this->element` with `rhs.element`, then it returns true if they are the same address or false otherwise
- `bool hashMap<t1, t2>::iterator::operator!=(const hashMap<t1, t2>::iterator& rhs) const` - compares if two iterators point to the same node, compares `this->element` with `rhs.element`, then it returns false if they are the same address or true otherwise
- `t1 hashMap<t1, t2>::iterator::first()` - returns the key field of the node, since each node in the linked list contains a pointer to a key in the existing table, the function would contain this single line of code: `return *(element->key);`
- `t2 hashMap<t1, t2>::iterator::second()` - returns the value field of the node, since each node in the linked list contains a pointer to a key in the existing table, the function would contain this single line of code: `return *(element->value);`

Contents of Main

Similar to the last assignment, your main file will need to include a set of `hashMap` objects to lookup elements. Once again all searches need to be done using the operator[] search/insert function, the iterator can be used to cycle through the elements when outputting at the end. You can first thing of a brute force approach to solve this problem and then modify the linear searches into hash map lookups. You can also use `unordered_map` at first just to see if your main algorithm is at least correct before implementing the `hashMap` class.

Input

The input begins with a single positive integer on a line by itself indicating the number of the cases following, each of them as described below. This line is followed by a blank line, and there is also a blank line between two consecutive inputs.

The first line of input contains a positive integer n, the number of candidates in the riding. Then n pair of lines: the first pair (line) contains the name of the candidate, then the second pair (line) contains the party name. No candidate name is repeated and no party name is repeated in the input (except for 'independent' since that is not a party affiliation). No lines contain leading or trailing blanks.

The next line contains a positive integer m and is followed by m lines each indicating the name of a candidate for which a ballot is cast. Any names not in the list of candidates should be ignored.

Output

For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line (or any whitespace separated). Output consists of a single line containing one of:

- The name of the party with whom the winning candidate is associated, if there is a winning candidate and that candidate is associated with a party.
- The word 'independent' if there is a winning candidate and that candidate is not associated with a party.
- The word 'tie' if there is no winner; that is, if no candidate receives more votes than every other candidate.

Specifications

- No linear or any searching allowed other than the use of a hash map object
- Make sure your code is memory leak free

Sample Run

```
$ ./a.out

Enter filename: input01.txt
Case 1 results: Rhinoceros

$ ./a.out

Enter filename: input02.txt
Case 1 results: Tie

$ ./a.out

Enter filename: input03.txt
Case 1 results: Green Party

$ ./a.out

Enter filename: input04.txt
Case 1 results: Alliance Party
```

```
$ ./a.out

Enter filename: input05.txt
Case 1 results: Libertarian Party
Case 2 results: Republican Party
Case 3 results: independent
```

Submission

Submit the source files to code grade by the deadline

References

- Supplemental Video https://youtu.be/lPOEV_q3hwA
- Link to the top image can be found at https://clipartmag.com/download-clipart-image#election-cliparts-35.png