CS302

Assignment 5: Word Classification (Using a custom hash map)

Description

For this assignment, you need to solve the same problem like in the previous assignment but this time you need to create a custom class, it can be seen below

```
template <class t1, class t2>
class hashMap
{
private:
  struct node
  {
    t1 key;
    t2 value;
    node * link;
  };

  std::size_t items1;
  std::size_t items2;

  std::size_t capacity;

  node ** table1;
  node ** table2;

  void resize(std::size_t);
  void copyMap(const hashMap<t1, t2>&);
  void deallocateMap();

  std::size_t h1(const std::string&) const;
  std::size_t h2(const std::string&) const;

public:
  hashMap() : items1(0), items2(0), capacity(5)
  {
    table1 = new node*[capacity];
```

```
    table2 = new node*[capacity];

    for (int i = 0; i < capacity; i++)
    table1[i] = table2[i] = nullptr;
  }

  const hashMap<t1, t2>& operator=(const hashMap<t1, t2>&);
  t2& operator[](t1);

  hashMap(const hashMap<t1, t2>& map) { copyMap(map); }
  ~hashMap() { deallocateMap(); }
};
```

Each member contains/performs the following

- `struct node` - each key value pair is stored in a node of a linked list
- `node ** table1` - a hash table that contains an array of linked lists, each linked list cannot exceed two nodes
- `node ** table2` - a hash table that contains an array of linked lists, each linked list cannot exceed two nodes
- `std::size_t items1` - denotes the amount of full lists in table1 (a full list is a linked list that contains 2 nodes)
- `std::size_t items2` - denotes the amount of full lists in table2 (a full list is a linked list that contains 2 nodes)
- `std::size_t capacity` - denotes the amount of elements in table1 and table2
- `hashMap<t1, t2>::hashMap()` - default constructor that initializes an empty `hashMap` object (is written for you in the header file provided)
- `hashMap<t1, t2>::hashMap(const hashMap<t1, t2>& copy)` - copy constructor, calls the `copyMap` function (the copy constructor is provided for you in the header file)
- `hashMap<t1, t2>::~hashMap()` - destructor that calls `deallocateMap`, the destructor code is provided for you in the header file
- `const hashMap<t1, t2>& hashMap<t1, t2>::operator=(const hashMap<t1, t2>& rhs)` - assignment operator, must deallocate the `*this` object and then deep copies `rhs` object into `*this` object, you can call `deallocateMap` and `copyMap` within the function
- `void hashMap<t1, t2>::copyMap(const hashMap<t1, t2>& map)` - performs a deep copy of `map` object into the `*this` object
- `void hashMap<t1, t2>::deallocateMap()` - deallocates the `*this` object, needs to deallocate all the nodes in table1 and table2 and then deallocate the table1 and table2 pointers
- `size_t hashMap<t1, t2>::h1(const string& s) const` - hash function that returns the following sum

$$\sum_{i=0}^{\text{str.length()}-1} 10^i * \text{str[i]}$$

- `size_t hashMap<t1, t2>::h2(const string& s) const` - hash function that returns the following sum

$$\sum_{i=0}^{\text{str.length}()-1} 10^i * \text{str}[\text{str.length}() - 1 - i]$$

- `t2& hashMap<t1, t2>::operator[](t1 key)` - the operator that overloads the insert/find function for the `hashMap` class, this function finds a node with a matching key and returns its value, if the key does not exist, the function creates a new entry in one of the linked lists and returns the value field of this newly created node, the steps below explain how this function should work
  1. if the load factor of table1 or table2 is equal to or over 20%, call the resize function (pass in the capacity for the actual parameter)
  2. Compute index1 and index2 by calling hash1 and hash2 and store their results into index1 and index2 respectively mod by capacity
  3. Check table1[index1] linked list, if a node in this list has a matching key, return the value field of this node
  4. If table1[index1] does not contain a matching key, insert a new node in the front or end of the linked list as long as there is room in table1[index1] linked list (if table1[index1] is empty or only contains 1 node then there is room for a new node), if the table1[index1] linked list becomes full (contains 2 nodes) after inserting the new node, increment items1, set this node's key field with the key parameter and set this node's value field with `t2()` which is the default value of whatever type `t2` is, then return this node's value field
  5. If a node with a matching key was not found in table1, then perform the same steps as the above step but on the linked list in table2[index2]
  6. If table2 is also full and no matching key, update `index1 = h1(key) + i * h2(key)` and `index2 = h2(key) + i * h1(key)` where `i` is the collision counter (and is set to 1 initially since there was a collision in both tables), then go to step 3
- `void hashMap<t1, t2>::resize(size_t amount)` - hash function that remaps everything from table1 and table2 to two larger tables of lengths capacity + amount, once everything is remapped, you set table1 and table2 to the larger tables (but you need to deallocate the original table1 and table2 lists)

Input

Check the previous assignment for the input

Output

Check the previous assignment for the output

Contents of Main

Same idea as in the last assignment but you replace `unordered_map` objects with `hashMap` objects, `vector` objects are still ok to use

Specifications

- Must use a `hashMap` object for any searching
- Program must work with the input provided on code grade

- No linear searching of any kind is allowed (all searching must use a hash map)

Sample Run

```
$ ./a.out < input01.txt

Test case 1
SQF Problem

$ ./a.out < input02.txt

Test case 1
Math
Physics
CompSci

$ ./a.out < input03.txt

Test case 1
SQF Problem

Test case 2
Geometrical

Test case 3
SQF Problem
```

Submission

Submit your hashMap.cpp and main.cpp to code grade by or before the deadline

References

- Supplemental Video https://youtu.be/BYOgZBCgIDU
- Link to the top image can be found at $https://www.pngpix.com/download/pile-paper-png-image$