Computer Science 271
Project 6
Due Friday, November 19

Complete this project with a partner. Both individuals are expected to contribute equally to all parts of the project.

1. Implement a hash table (with chaining) template class using a `List` template class (possibly modified) that you wrote in CS 173. (If you need one, see me.) Your template class definition will look like this:

```
template <class KeyType>
class HashTable
{
   public:
      HashTable(int numSlots);
      HashTable(const HashTable<KeyType>& h);
      ~HashTable();

      KeyType* get(const KeyType& k) const;
      void insert(KeyType *k);
      void remove(const KeyType& k);

      HashTable<KeyType>& operator=(const HashTable<KeyType>& h);

      std::string toString(int slot) const;

   private:
      int slots;
      List<KeyType> *table;  // an array of List<KeyType>'s
};
```

   Notes:

   - Notice that the `insert` and `get` methods insert and return *pointers* to `KeyType` objects. This means that your `List` class needs to store pointers as well, i.e., each node needs to contain a *pointer* to an item. Also, the `List` methods that insert new items need to take pointers as parameters, and the methods that return items need to return pointers. In other words, the `List` interface should look like this:

```
template <class T>
class Node
{
   public:
      T *item;
      Node<T> *next;

      // constructors here...
};

template <class T>
class List
{
   public:
      // constructors here...
```

```
        void append(T *item);
        int index(const T& item) const;
        void insert(int index, T *item);
        T *pop(int index);
        void remove(const T& item);

        // other methods...

    private:
        Node<T> *head;
        int count;

        // private methods...
};
```

- `KeyType` represents the type of data being stored in the hash table. Your implementation should assume that
  - the `KeyType` class contains a key field and that the comparison operators for `KeyType` have been overloaded so that they return the result of comparisons between keys,
  - the stream insertion operator (`<<`) has been overloaded for `KeyType`, and
  - `KeyType` includes a public method named

    ```
    int hash()
    ```

    that returns a non-negative integer based on the value of the key. (Note that this value must fit in the space allocated to an `int`.) When your hash function methods use a `hash` method to compute a slot number, they will need to mod the value by the number of slots, i.e.,

    ```
    slot = k->hash() % slots;
    ```
- The parameter of the `get` and `remove` methods is an object in which the internal key field is set to the value of the key for which to search. (Any other data fields in the object will be ignored.) The `get` method should return a pointer to the object in the hash table that contains the desired key.
- Include suitable preconditions and postconditions in the comments before each method.
- Your `get` and `remove` methods should throw `KeyError` exceptions when the parameter is not found.
- The `toString` method should return a string representation of the items in the given `slot`.
- Include unit tests (using `assert` and the `toString` method) for each of your methods. When you are testing the `get` method, make sure you also test that the method is returning the correct *pointer* value, not just the correct key value.

2. Develop a good hash function mapping strings to integers in $\{0, 1, \ldots, m-1\}$. Develop this hash function on your own, without consulting external sources. Evaluate your hash function by writing a small program that calls your hash function on every word in the dictionary file `/usr/share/dict/words`. (Note that you do not need to actually insert the words into a hash table.) Create a bar graph that displays how evenly your hash function assigns words to slots when $m = 1000$ (x-axis is slot number, y-axis is chain length). Also compute the minimum and maximum number of words assigned to a slot, and the standard deviation. The standard deviation of a sample is defined to be

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$$

where n is the number of values (i.e., slots), x_i is the i-th value (i.e., number of items assigned to slot i), and μ is the mean of the values (i.e., total number of items / number of slots). In Python, the standard deviation would be computed as follows:

```
def stdDev(data):
    n = len(data)
    mean = sum(data) / n
    total = 0
    for x in data:
        total += (x - mean) ** 2
    return math.sqrt(total / n)
```

Hand in a document (LaTeX ⇒ PDF) containing a description of your hash function and your analysis of it.

3. Implement a dictionary ADT as a template class named `Dictionary` that inherits from your hash table template class. Your class should include same methods as your `Dictionary` template class in the previous project. Remember that all keys must be unique; your `insert` method should throw a `KeyError` exception if a duplicate key is inserted. Include unit tests for all four methods.

4. Recompile your `query_movies.cpp` program from the previous project using your new `Dictionary` template class. (It should work exactly the same!) Since the key in this application (a movie title) is a string, you should implement the `hash` method for your `Movie` class using the algorithm you developed in part 2.

   Using the time functions introduced in Project 3, compare how long it takes to insert all of the movies if the program uses the hash table implementation to how long it takes if the program uses the binary search tree implementation. Explain your results.

To submit your project, submit

1. the following source files:

   - `hash.h`, `hash.cpp`, and `test_hash.cpp`
   - `dict.h`, `dict.cpp`, and `test_dict.cpp`
   - `movie.h` and `query_movies.cpp`

2. and a **single** PDF named `proj7_yournames.pdf` that combines a PDF of the above source files (created using `enscript`) with your answer to part 2.

   You can join together two PDF files using the `pdfjoin` program. If your source code PDF is named `proj7_yournames_code.pdf` and your answer to part 2 is named `proj7_yournames_hash.pdf`, then join them together into a single PDF named `proj7_yournames.pdf` with

   ```
   pdfjoin -o proj7_yournames.pdf proj7_yournames_code.pdf proj7_yournames_hash.pdf
   ```

Be sure to indicate the names of both members on all of your submitted files.