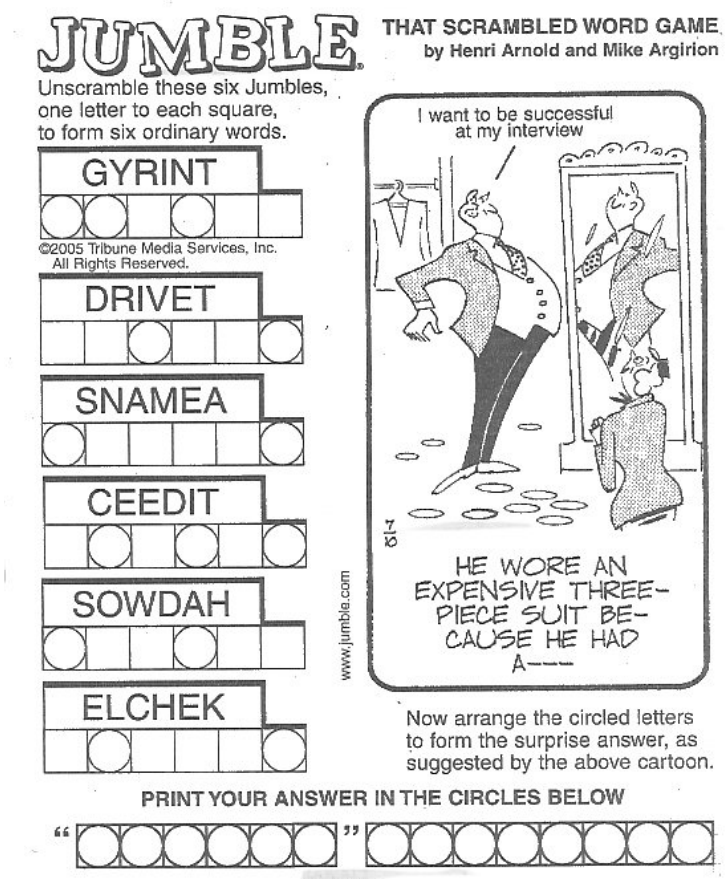
CS251 - Jumble Project
Due Monday April 8 at 11am.

In this project you will use a clever application of hash tables to solve "word jumble" puzzles. You have probably seen these puzzles in the newspaper. An example is below.



You are given a sequence of letters and you are supposed to rearrange them to form a real English word. That's about it!

Overview of Program Behavior

Your program will first initialize its internal data structures (primarily a hash table of your own design) from a dictionary of English words which has been provided to you. You will implement se a **"separate chaining"** policy for resolving collisions,
After initialization your program will **report some statistics** on your

hash table (things like number of entries, number of non-empty buckets, average list length among non-empty buckets, longest list length; details below).

Then your program enters an **interactive loop** in which it asks the user to enter a jumbled string and then reports a list of all English words which can be formed by rearranging the letters (if any).

The program terminates when the user enters a single period "." instead of a jumbled sequence of letters.

Details

So, what should you store in your hash table? Would storing the words themselves in the hash table help us in this application? It doesn't seem like it -- we aren't given a word to lookup, but the jumbled version of a word. Here is the idea:

Each word in the dictionary is a collection of letters in a specific order. Suppose we re-arrange the letters so they are in sorted order?

For example the sorted reording of "steak" is "aekst". Similarly, the sorted reordering of "stake" is also "aekst." A trivial observation is that one string **s1** can be rearranged to be equal to another string **s2** if and only if **s1** and **s2** have exactly the same sorted representation.

So instead of storing actual words in your hash table you will store sorted re-orderings (as the "key" and a list of all words that have the same sorted ordering (as the "value"). In other words, the table is implementing a function from sorted strings to sets of English words that are formed from exactly those letters.

When the user enters a jumbled sequence of letters, you then just sort the string and do a lookup on it to retrieve all words that can be formed by rearranging the letters.

For example, if the user enters "ketsa", you would use the sorted version "aekst" as the key for a hash table lookup which would return the English words "steak", "stake" and "takes" (could be others I

suppose).

Design

For this assignment, you will have more freedom in your design choices than in previous assignments. However, you must decompose your code into a hash table module and an application module. So you will have the following files:

The hash table header file: **ht.h**

The hash table implementation file **ht.c**

The jumble application file (containing main): **jumble.c**

Your design should employ data abstraction -- for example, to the degree possible, your hash table module should be general purpose and not intrinsically tied to the jumble application program.

Requirements:

Dynamic resizing/load factor. Your hash table should automatically resize itself to twice its current size when an insertion makes the table exceed its maximum load factor. The load factor is a constant which you will choose, but something around 0.75 is typical.

Hash function options. You will implement two hash functions. The first (hash function "zero") will be "naive" and simply add the numerical values of the letters and take the modulus with respect to the table size.

Hash function "one" will be more sophisticated. It will weight the individual string positions differently as we have discussed in class. The exact coefficients you use are up to you, but the general form should
be something like this (before taking the modulus):

$$\text{hash(s)} = \sum_{i=0}^{n-1} c^{(n-i-1)} \times s_i$$

for some coefficient c.

Suggestion: Since your program can specify either hash function, you should probably use **function pointers** to make this simple. Simply assign a hash function pointer variable to one of the two hash functions and from that point onward, just invoke the right hash function through the function pointer variable. This saves the trouble of a bunch of tedious conditionals all over the place checking which hash function you are supposed to be using.

Command line args. Your program will be called jumble will have one required command line argument -- the dictionary file name -- and one optional argument -- a hash function selector. If not hash function selector is specified, your program should default to using the "smart" hash function (hash function one). Examples:

```
jumble dict.txt 0

jumble dict.txt 1
```

The dictionary file will simply contain one word per line for easy parsing.

Submission

You will submit a single archive file containing the following:

ht.h
ht.c
jumble.c
readme.txt or **readme.pdf**

We've discussed all of them except the readme file. In your readme file include the following:

1. An explanation of your "smart" hash function.
2. Discussion of the relative performance of the two hash functions in terms of
    a. expected time for lookups,
    b. worst-case time for lookups
    c. table usage
3. A discussion of your hash table module interface -- e.g., what is visible to a client, what is hidden; how general purpose is the table -- can it easily be used for other hash table applications etc.

Submissions will be made through blackboard. A sample dictionary file will be made available.