Assignment #5

Computer Science I – COP 3502

Objectives

1. To learn how to use hash tables
2. To practice implementing known algorithms

Problem: Game Cabinet (cabinet.c)

This program is designed to organize a collection of games. A hash table will be used to simulate a storage closet for various board and card games.

You will implement a hash table to store the names of these games and then answer some queries about if and where the games are located. You are required to implement the linear chaining hashing technique, which stores an array of linked lists. Each linked list node should simply store a string (no longer than 29 characters) and a pointer to another node. The size of the array will be specified in the input file, thus, the array needs to be dynamically allocated.

Hash Function

The input to our hash function will be a string that represents the name of a game to be stored. The hash function will need to convert this string to a valid index of the hash table using the following function:

$hash(c_{n-1}c_{n-2}...c_1c_0) = [ascii(c_{n-1}) + ascii(c_{n-1}) + \ldots + ascii(c_1) + ascii(c_0)] \% tablesize$

where n is the length of the string and c_x is an individual character of the string, $0 <= x < n$.

Recommended Data Structures

```
struct game {
        char name[30];
        struct game * next;
};

struct game ** cabinet;
```

Specifications

These specifications must be followed exactly.

There are two input files to use, one of games to be stored in the hash table and one of queries to be executed.

Input File 1:

1. The input location will be an input file titled “gamelist.txt”.
2. The first line of the file will be an integer, n, that represents the number of games in the file. This number is **also** the table size for the hash table.
3. All games will be UPPERCASE and contain valid ascii characters without whitespace

Input File 2:

1. The input location will be an input file titled “query.txt”.
2. The first line of the file will be an integer, n, that represents the number of queries in the file.
3. All queries and games will be UPPERCASE and contain valid ascii characters without whitespace

Query Commands:

There are two types of queries specified in the file:

1) Whether or not a game is stored in the game cabinet/hash table. (And if it is, where it is stored.)
2) The number of games in the game cabinet/hash table stored at the same location as a particular game would be stored.

Each query starts with a single number (either 1 or 2), specifying the type of query as noted above. This is followed by a space and a string to query.

For a query of type #2, first calculate the hash value of the given game. Then determine the number of games stored in the game cabinet/hash table at that location.

Output:

1. Output location must be an output file titled “queryresults.out”
2. For each query, print a header and the results of the query:
    a. Query #c
    b. Type 1 Queries
        i. <Game> was find in location x.
        ii. <Game> was not found in the game cabinet.
    c. Type 2 Queries
        i. There are z games stored at location x.

Sample Input 1 (gamelist.txt)

```
11
APPLES_TO_APPLES
BOGGLE
CANDYLAND
CHESS
GO_FISH
MONOPOLY
POKER
SCRABBLE
TWISTER
UNO
WAR
```

Sample Input 2 (query.txt)
4
1 TWISTER
2 POKER
1 DOMINOES
2 GO_FISH

Sample Output File:
Query #1: TWISTER was found in location 1.

Query #2: There are 3games stored at location 0.

Query #3: DOMINOES was not found in the game cabinet.

Query #4: There are 1 gamesat location 4.

Program:

1. You must implement the hash table as a dynamically allocated array of linked lists
2. You must design, implement, and use at least 2 functions in your solution

Deliverables
One source file – *cabinet.c* – is to be submitted over WebCourses.

Restrictions
Your program must be able to be compiled and executed in a standard C Development Environment. Yourprogram should include a header comment with the following information: your name, course number, section number, assignment title, and date. Also, make sure youinclude comments throughout your code describing the major steps in solving the problem.

Grading Details
Your programs will be graded upon the following criteria:

1) Your correctness

2) Your programmingstyleand use of white space. Even if you have a plan and your program works perfectly, if your programming style is poor or your use of white space is poor, you could get 10% or 15% deducted from your grade.

3) Your comments. Each function must have appropriate pre and post conditions. Internal comments should be used liberally – both in main and in any functions.

4) Compatibility – You must submit C source files that can be compiled and executed in a standard C Development Environment. If your program does not compile, you will get a sizable deduction from your grade.