Assignment 4: Cuckoo Hashing algorithm

Name: Sangyong Jin, Danny Navarro, Shelley Pham

INPUT: an input file containing strings of characters, one string per line

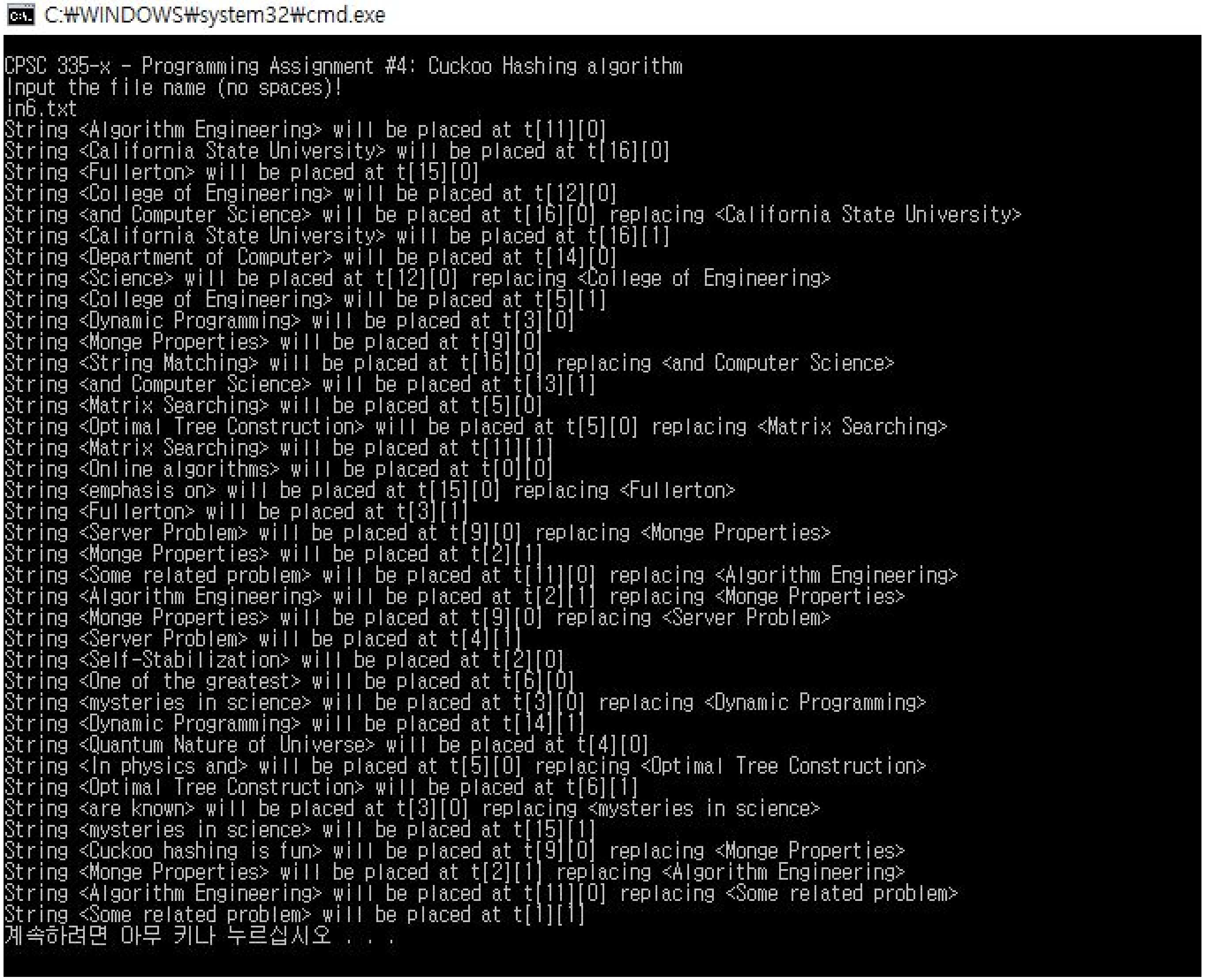
OUTPUT: a detailed list of where the strings are inserted.

Contents

	Table T1	Table T2
[0]	Online Algorithms	
[1]		Some related problem
[2]	Self-Stabilization	Monge Properties
[3]	Are known	Fullerton
[4]	Quantum Nature of Universe	Server Problem
[5]	In physics and	College of Engineering
[6]	One of the greatest	Optimal Tree Construction
[7]		
[8]		
[9]	Cuckoo Hashing is fun	
[10]		
[11]	Algorithm Engineering	Matrix Searching
[12]	Science	
[13]		And Computer Science

[14]	Department of Computer	Dynamic Programming
[15]	Emphasis on	Mysteries in science
[16]	String Matching	California State University

Output



```
CPSC 335-x - Programming Assignment #4: Cuckoo Hashing algorithm
Input the file name (no spaces)!
in6.txt
String <Algorithm Engineering> will be placed at t[11][0]
String <California State University> will be placed at t[16][0]
String <Fullerton> will be placed at t[15][0]
String <College of Engineering> will be placed at t[12][0]
String <and Computer Science> will be placed at t[16][0] replacing <California State University>
String <California State University> will be placed at t[16][1]
String <Department of Computer> will be placed at t[14][0]
String <Science> will be placed at t[12][0] replacing <College of Engineering>
String <College of Engineering> will be placed at t[5][1]
String <Dynamic Programming> will be placed at t[3][0]
String <Monge Properties> will be placed at t[9][0]
String <String Matching> will be placed at t[16][0] replacing <and Computer Science>
String <and Computer Science> will be placed at t[13][1]
String <Matrix Searching> will be placed at t[5][0]
String <Optimal Tree Construction> will be placed at t[5][0] replacing <Matrix Searching>
String <Matrix Searching> will be placed at t[11][1]
String <Online algorithms> will be placed at t[0][0]
String <emphasis on> will be placed at t[15][0] replacing <Fullerton>
String <Fullerton> will be placed at t[3][1]
String <Server Problem> will be placed at t[9][0] replacing <Monge Properties>
String <Monge Properties> will be placed at t[2][1]
String <Some related problem> will be placed at t[11][0] replacing <Algorithm Engineering>
String <Algorithm Engineering> will be placed at t[2][1] replacing <Monge Properties>
String <Monge Properties> will be placed at t[9][0] replacing <Server Problem>
String <Server Problem> will be placed at t[4][1]
String <Self-Stabilization> will be placed at t[2][0]
String <One of the greatest> will be placed at t[6][0]
String <mysteries in science> will be placed at t[3][0] replacing <Dynamic Programming>
String <Dynamic Programming> will be placed at t[14][1]
String <Quantum Nature of Universe> will be placed at t[4][0]
String <In physics and> will be placed at t[5][0] replacing <Optimal Tree Construction>
String <Optimal Tree Construction> will be placed at t[6][1]
String <are known> will be placed at t[3][0] replacing <mysteries in science>
String <mysteries in science> will be placed at t[15][1]
String <Cuckoo hashing is fun> will be placed at t[9][0] replacing <Monge Properties>
String <Monge Properties> will be placed at t[2][1] replacing <Algorithm Engineering>
String <Algorithm Engineering> will be placed at t[11][0] replacing <Some related problem>
String <Some related problem> will be placed at t[1][1]
계속하려면 아무 키나 누르십시오 . . .
```

Source Code

```
#include <iostream>
#include <cstring>
#include <stdio.h>


using namespace std;


// cuckoo tables' size
const int tablesize = 17;
// combine the two 1-dimensional table into one 2-dimensional table
char  t[tablesize][2][255];


// compute the hash functions
size_t f(char*, size_t);


// place a string in one of the hash tables
bool place_in_hash_tables(char*);


int main() {


      // the strings to be stored in the hash tables
      char s[255] = "";
      char null_st[] = "";
      size_t i, len;
      bool placed;
```

```
// clear the tables
for (i = 0; i< tablesize; i++) {
     strcpy(t[i][0], null_st);
     strcpy(t[i][1], null_st);
}


char filename[255] = "";


// display the header
cout << endl << "CPSC 335-x - Programming Assignment #4: ";
cout << "Cuckoo Hashing algorithm" << endl;

// read the strings from a file
cout << "Input the file name (no spaces)!" << endl;
cin >> filename;


// open the file for reading
FILE *file = fopen(filename, "r");
if (file != NULL)
{
     /* read line by line from the file */
     while (fgets(s, 255, file) != NULL) {
          // place null character at the end of the line instead of <return>
          len = strlen(s);
          s[len - 1] = '\0';
          // insert the string in the cuckoo table
```

```
                placed = place_in_hash_tables(s);
                // check whether the placement was successful
                if (!placed) {
                     cout << "Placement has failed" << endl;
                     return -1;
                }
           }
           fclose(file);
      }
      else
      {
           perror(filename); /* why didn't the file open? */
      }

      return 0;

}




bool place_in_hash_tables(char *s) {

      bool placed;
      size_t pos;
      int index;
      char temp_s[255], temp[255];

      strcpy(temp_s, s);

      // use a counter to detect loops
```

```
     int counter = 0;

     // start with table T1
     index = 0;

     placed = false;


     pos = f(temp_s, index);


     while ((!placed) && (counter < 2 * tablesize)) {


           if (strcmp(t[pos][index], "") == 0) {
                // the entry at index <pos> in the <index> hash table is available so store the
string <temp_s> there
                cout << "String <" << temp_s << "> will be placed at";
                cout << " t[" << pos << "][" << index << "]" << endl;
                strcpy(t[pos][index], temp_s);
                placed = true;
                return placed;
           }
           else {
                // the entry at index <pos> in the <index> hash table is not available so
                // obtain the string stored over there in variable <temp> and store the string
<temp_s> there
                // now the string <temp> needs to be placed in the other table
                cout << "String <" << temp_s << "> will be placed at" << " t[" << pos;
                cout << "][" << index << "]" << " replacing <" << t[pos][index] << ">";
                cout << endl;
```

```
               // YOU NEED TO WRITE THE CODE TO STORE IN temp THE STRING STORED AT
               // t[pos][index] AND STORE IN t[pos][index] THE STRING temp_s
               strcpy(temp, t[pos][index]);
               strcpy(t[pos][index], temp_s);
               // NOW temp_s CONTAINING THE EVICTED STRING NEEDS TO BE STORED
               strcpy(temp_s, temp);
               // IN THE OTHER TABLE
               // WRITE THE CODE TO SET index TO INDICATE THE OTHER TABLE
               if (index == 0)
                    index = 1;
               else if (index == 1)
                    index = 0;
               // WRITE THE CODE TO CALCULATE IN pos THE HASH VALUE FOR temp_s
               pos = f(temp_s, index);
               counter++;

          }
     }
     return placed;
};
```

```
size_t f(char *s, size_t index) {
// compute the hash functions
     // s is the string (the key) to which we apply the hash function
     // index indicates which hash function will be used
     // index == 0 means the first hash function
     // index == 1 means the second hash function
```

```
size_t po, len;
int i, val = 0, temp;
po = 1;


len = strlen(s);


if (index == 0) {

     val = s[0];
     val = val % tablesize;
     if (val < 0) val += tablesize;


     if (len == 1)
          return val;

     for (i = 1; i < len; i++)
     {
          temp = s[i];
          po *= 31;


          po = po % tablesize;
          if (po < 0) po += tablesize;

          val += temp * po;
          val = val % tablesize;
```

```
            if (val < 0) val += tablesize;
      }
      return val;
}
else
{
      // YOU NEED TO IMPLEMENT THE STEPS TO CALCULATE THE SECOND
      // HASH FUNCTION
      val = s[len - 1];
      val = val % tablesize;
      if (val < 0) val += tablesize;

      if (len == 1)
            return val;

      for (i = 1; i < len; i++)
      {
            temp = s[len - i - 1];
            po *= 31;

            po = po % tablesize;
            if (po < 0) po += tablesize;

            val += temp * po;
            val = val % tablesize;

            if (val < 0) val += tablesize;
      }

      return val;
}}
```