

Problem

Implement hash table.

your hash table will consist of English language words - 8 characters or less. Start with a table size of 19 and insert words one at a time. When you have inserted 10 words, your program must give a message "load factor $> .5$ ". Then the program automatically increases the table size to double the current size and round up to the next higher prime number. Insert a few more words after that.

You may select a library hash function. But you must implement your own collision resolution scheme (quadratic probing).

Keep track of number of collisions for a given word. So if you try to insert a word and there is a collision then you select the next location based on quadratic probing. If this second location also gives a collision then you have 2 collisions and so on. Keep track of the highest collision for a given word and print it.

Run your program to insert words and then show a screen shot of its execution to

1. Find a word that has been added.
2. Try to find a word that is not in the table - your program should print a message saying it is not in the dictionary.

Approach

What I did is I considered a mapping from words of eight characters or less to the unit interval. Then I mapped the unit interval to the indexes of 1 to 19 (0 to 18 in the program). The map took two things into consideration:

1. Length of Word
2. Starting Character

I did not differentiate from upper case and lower case.

Code

Code 1: Hash_Functions.h

```
#ifndef Hash_Functions
#define Hash_Functions

#include <iostream>

// Since we are using words, we need this library.
#include <string>

// Since it is easier to change the size of vectors than arrays,
// I used vectors.
#include <vector>

// I used this to hold words while i change the size of the Hash Table.
#include <queue>

// For isblank.
#include <ctype.h>

using namespace std;

// This function will determine whether a number is prime or not.
bool Prime(int n) {
    int m = 2;
    while (m < n / 2) {
        if (n % m == 0) {
            return false;
        }
        m++;
    }
    return true;
}

// Given a number n, this function will find the next highest prime
// number, if n is prime, then it will return n.
int Next_Prime(int n) {
    if (Prime(n)) {
        return n;
    }
    else return Next_Prime(n + 1);
}

// This function will take a string of at most 8 characters and
// assign a number between 0 and 1 based on two things:
// 1. The size of the string
// 2. The starting character
// Note that we do not differentiate between lower case and
```

```
// upper case.
double String_Value(string s) {
    double value = 0;

    // l is the raw size of the string.
    // tl is the true size of the string, which ommits blank spaces.
    int l = s.size();
    int tl = 0;
    int m = 0;
    for (int i = 0; i < l; i++) {
        if (!isblank(s[i])) {
            tl++;
        }
    }

    // Assing the first letter of the string a value.
    // Note that I do not differentiate between lower
    // case and upper case.
    if (!isblank(s[0])) {
        m = (int)s[0];
        if (m > 64 && m < 91) {
            m = m - 64;
        }
        else if (m > 96 && m < 123) {
            m = m - 96;
        }
    }

    // Here we calculate the value of the word based on two things:
    // 1. Length of word
    // 2. Starting letter
    double dm = (double)m / 26;
    double dtl = (double)tl / 8;
    value = (dm + dtl)/2;
    return value;
}

// Here we will map the value of the word to an index from 0 to
// size - 1.
int Hash_Map(int size, double val) {
    double m = val * (double)size;
    return (int)m - 1;
}

// Here we have the Hash Class containing the Hash Table.
class Hash {
public:
    int size;
    double load;
    vector<string> Hash_Table;
};
```

```
vector<int> Collisions;
void Change_Size(int);
void add(string);
void find(string);
void Print();
};

//This function changes the size of the Hash Table.
void Hash::Change_Size(int n) {
    Hash_Table.clear();
    Collisions.clear();
    size = n;
    for (int i = 0; i < size; i++) {
        Hash_Table.push_back("_");
        Collisions.push_back(0);
    }
}

// This function will add words to the hash table.
void Hash::add(string s) {
    cout << "The_Word_added_is:_" << s << endl;
    cout << "Word_Value_is:_" << String_Value(s) << endl;

    if (load > .5) {
        cout << "Load_Factor_>.5" << endl << endl;
        int n = Next_Prime(Hash_Table.size() * 2);
        cout << "Changing_Hash_Table_size_to_" << n << "."
            <<endl << endl;

        // Save the current words in the Hash Table to a temporary
        // queue while we change the size of the Table.
        queue<string> temp;
        for (int i = 0; i < size; i++) {
            if (String_Value(Hash_Table[i]) != 0) {
                temp.push(Hash_Table[i]);
            }
        }
        cout << "Now_adding_all_the_words_back_in_including_the_"
            << "new_word:_" << endl << endl;
        // Now lets add the new element to this temp vector.
        temp.push(s);
        Change_Size(n);
        load = 0;
        int sizetemp = temp.size();
        for (int i = 0; i < sizetemp; i++) {
            add(temp.front());
            temp.pop();
        }
    }
    else {
```

```
// We calculate the position that the word should be in.
double value = String_Value(s);
int i = Hash_Map(size, value);
string temp = Hash_Table[i];

// If the position is empty, then we place it there.
if (String_Value(temp) == 0) {
    Hash_Table[i] = s;
    load = load + 1 / ((double)size);
    cout << "The Load is: " << load << endl;
}
// Otherwise we have collisions and must act accordingly.
else
{
    Collisions[i]++;
    int p = 1;
    int I = (i + p) % size;
    temp = Hash_Table[I];
    while (String_Value(temp) != 0) {
        Collisions[I]++;
        p = p * 2;
        I = (i + p) % size;
        temp = Hash_Table[I];
    }
    Hash_Table[I] = s;
    load = load + 1 / ((double)size);
    cout << "The Load is: " << load << endl;
}

cout << endl;
}

// This function will compare two strings to see if they are the same or not.
bool Compare(string a, string b) {
    int la = a.size();
    int lb = b.size();
    if (la != lb) {
        return false;
    }
    for (int i = 0; i < la; i++) {
        if (a[i] != b[i]) {
            return false;
        }
    }
    return true;
}

// This function will take a string and see if it is in the Hash Table.
void Hash::find(string w) {
```

```
    cout << "Finding_the_word:_ " << w << endl;

    // Calculating the position of the word in the Hash Table.
    double val = String_Value(w);
    int i = Hash_Map(size, val);
    int I = 0;
    string s = Hash_Table[i];

    // If we find it immediatly, then we found it.
    if (Compare(w, s)) {
        cout << "The_word_was_found_at_index_" << i + 1 << endl << endl;
        return;
    }

    // If it is not in its correct spot but there are collisions in this
    // spot, then we will traverse through the collisions until we either
    // find the word or run out of collisions.
    if (Collisions[i] > 0) {
        int p = 1;
        I = (i + p) % size;
        s = Hash_Table[I];
        while (!Compare(w, s) && (Collisions[I] > 0)) {
            p = p * 2;
            I = (i + p) % size;
            s = Hash_Table[I];
        }
    }

    // Once we go through all the collisions, we check and see if the
    // word is there.
    if (Compare(w, s)) {
        cout << "The_word_was_found_at_index_" << I + 1 << endl << endl;
        return;
    }
    else {
        cout << "The_word_was_not_found." << endl << endl;
        return;
    }
}

// For formatting the table.
void Word_Print(string s) {
    int l = s.size();
    for (int i = 0; i < 8; i++) {
        if (i < l) {
            cout << s[i];
        }
        else {
            cout << "_";
        }
    }
}
```

```
    }  
}  
  
// This function will print the Hash Table.  
void Hash::Print() {  
    for (int i = 0; i < size; i++) {  
        string w = Hash_Table[i];  
        cout << i + 1 << " : ";  
        Word_Print(w);  
        cout << " " << "Collisions : " << Collisions[i] << endl;  
    }  
    cout << endl;  
}  
#endif Hash_Functions
```

Code 2: Assignment_6_Hash_Table.cpp

```
#include "Hash_Functions.h"

using namespace std;

int main() {
    // Initialise the Hash Table and make it to be size 19.
    Hash Words;
    Words.Change_Size(19);

    // Add the following words.
    Words.add("It");
    Words.add("Would");
    Words.add("Be");
    Words.add("Nice");
    Words.add("If");
    Words.add("You");
    Words.add("Give");
    Words.add("Me");
    Words.add("An");
    Words.add("A");

    // Print the current table.
    Words.Print();

    // Find the following words.
    Words.find("Nice");
    Words.find("Me");
    Words.find("Triangle");

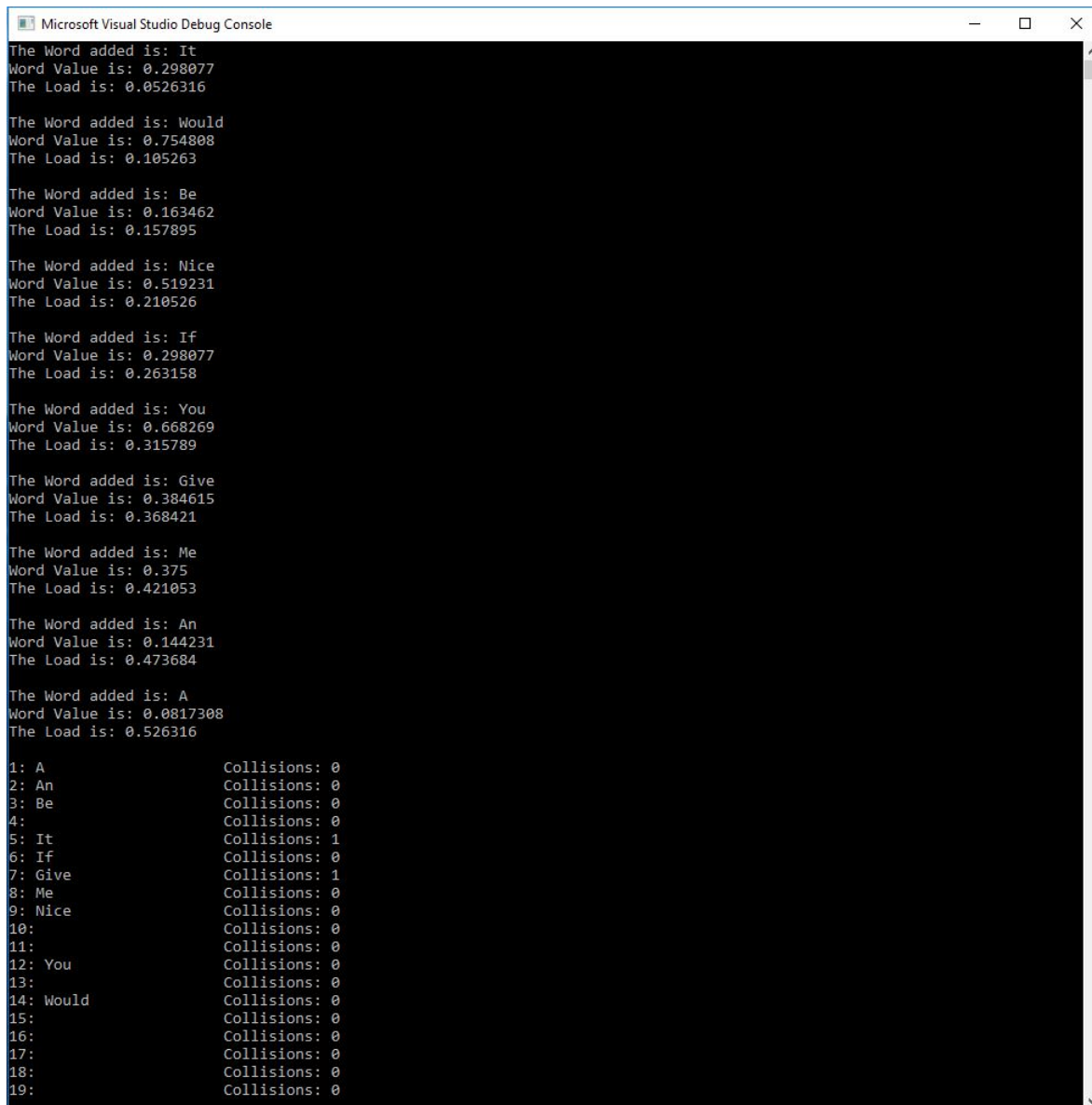
    // Now let us add more words but here we pass the load threshold.
    Words.add("In");
    Words.add("This");
    Words.add("Class");

    // Print the final table.
    Words.Print();

    // Find these following words in the final table.
    Words.find("In");
    Words.find("Nice");
    Words.find("Me");
    Words.find("Banana");
}
```

Results

The following is the run I did.



```
Microsoft Visual Studio Debug Console

The Word added is: It
Word Value is: 0.298077
The Load is: 0.0526316

The Word added is: Would
Word Value is: 0.754808
The Load is: 0.105263

The Word added is: Be
Word Value is: 0.163462
The Load is: 0.157895

The Word added is: Nice
Word Value is: 0.519231
The Load is: 0.210526

The Word added is: If
Word Value is: 0.298077
The Load is: 0.263158

The Word added is: You
Word Value is: 0.668269
The Load is: 0.315789

The Word added is: Give
Word Value is: 0.384615
The Load is: 0.368421

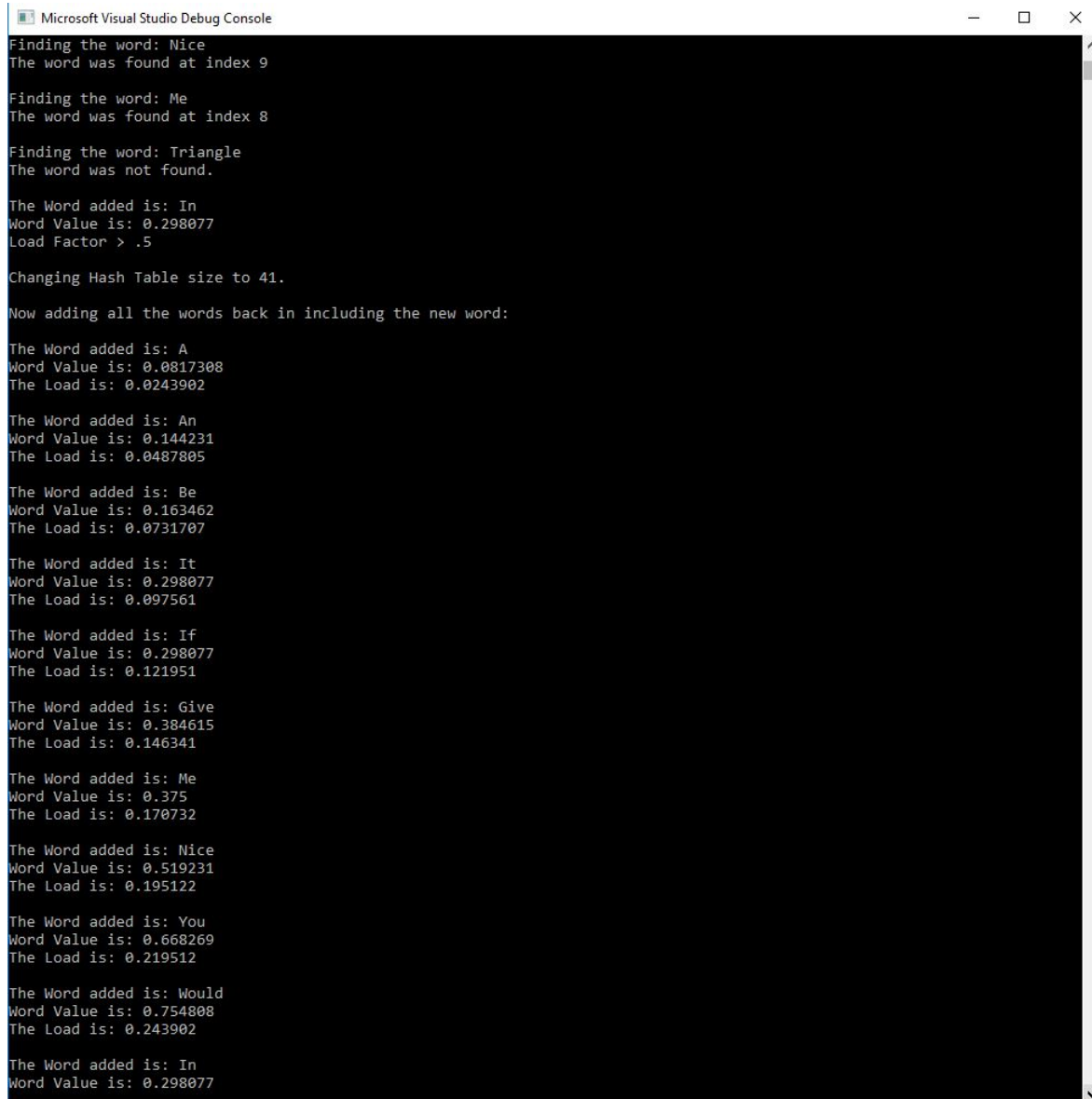
The Word added is: Me
Word Value is: 0.375
The Load is: 0.421053

The Word added is: An
Word Value is: 0.144231
The Load is: 0.473684

The Word added is: A
Word Value is: 0.0817308
The Load is: 0.526316

1: A           Collisions: 0
2: An          Collisions: 0
3: Be          Collisions: 0
4:             Collisions: 0
5: It          Collisions: 1
6: If          Collisions: 0
7: Give        Collisions: 1
8: Me          Collisions: 0
9: Nice        Collisions: 0
10:            Collisions: 0
11:            Collisions: 0
12: You        Collisions: 0
13:            Collisions: 0
14: Would      Collisions: 0
15:            Collisions: 0
16:            Collisions: 0
17:            Collisions: 0
18:            Collisions: 0
19:            Collisions: 0
```

Figure 1: Results



```
Microsoft Visual Studio Debug Console
Finding the word: Nice
The word was found at index 9

Finding the word: Me
The word was found at index 8

Finding the word: Triangle
The word was not found.

The Word added is: In
Word Value is: 0.298077
Load Factor > .5

Changing Hash Table size to 41.

Now adding all the words back in including the new word:

The Word added is: A
Word Value is: 0.0817308
The Load is: 0.0243902

The Word added is: An
Word Value is: 0.144231
The Load is: 0.0487805

The Word added is: Be
Word Value is: 0.163462
The Load is: 0.0731707

The Word added is: It
Word Value is: 0.298077
The Load is: 0.097561

The Word added is: If
Word Value is: 0.298077
The Load is: 0.121951

The Word added is: Give
Word Value is: 0.384615
The Load is: 0.146341

The Word added is: Me
Word Value is: 0.375
The Load is: 0.170732

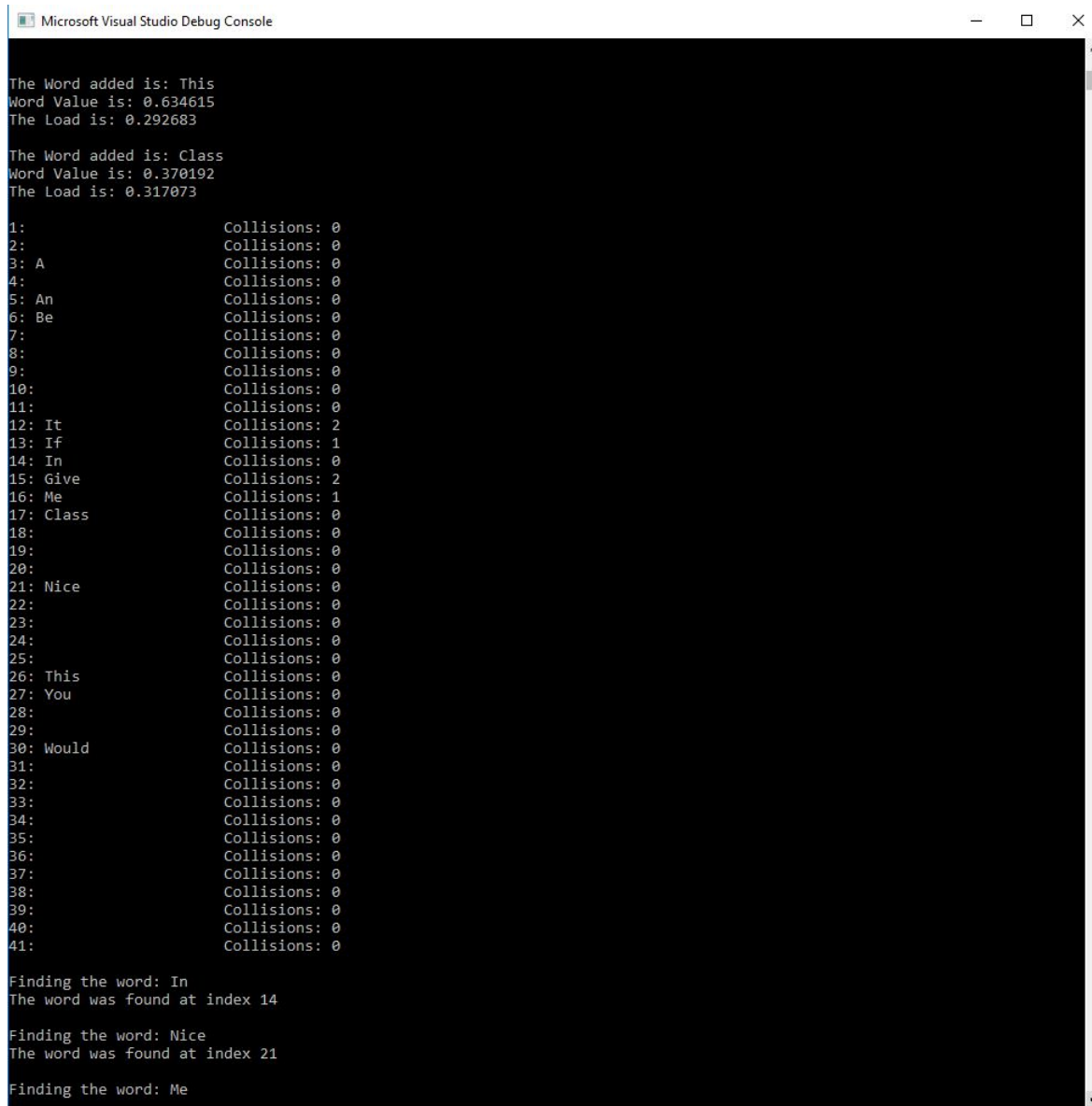
The Word added is: Nice
Word Value is: 0.519231
The Load is: 0.195122

The Word added is: You
Word Value is: 0.668269
The Load is: 0.219512

The Word added is: Would
Word Value is: 0.754808
The Load is: 0.243902

The Word added is: In
Word Value is: 0.298077
```

Figure 2: Results



```
Microsoft Visual Studio Debug Console

The Word added is: This
Word Value is: 0.634615
The Load is: 0.292683

The Word added is: Class
Word Value is: 0.370192
The Load is: 0.317073

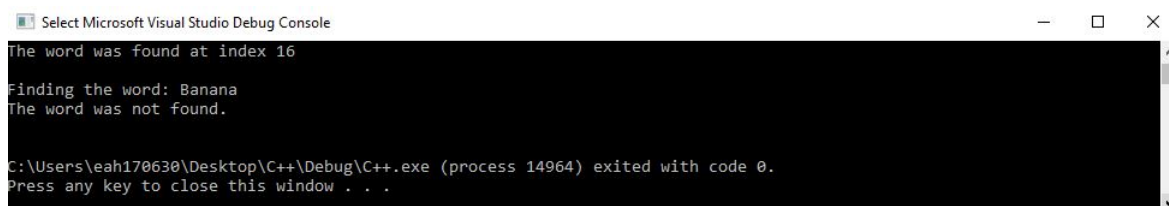
1:          Collisions: 0
2:          Collisions: 0
3: A        Collisions: 0
4:          Collisions: 0
5: An       Collisions: 0
6: Be       Collisions: 0
7:          Collisions: 0
8:          Collisions: 0
9:          Collisions: 0
10:         Collisions: 0
11:         Collisions: 0
12: It       Collisions: 2
13: If       Collisions: 1
14: In       Collisions: 0
15: Give     Collisions: 2
16: Me       Collisions: 1
17: Class    Collisions: 0
18:         Collisions: 0
19:         Collisions: 0
20:         Collisions: 0
21: Nice     Collisions: 0
22:         Collisions: 0
23:         Collisions: 0
24:         Collisions: 0
25:         Collisions: 0
26: This     Collisions: 0
27: You      Collisions: 0
28:         Collisions: 0
29:         Collisions: 0
30: Would    Collisions: 0
31:         Collisions: 0
32:         Collisions: 0
33:         Collisions: 0
34:         Collisions: 0
35:         Collisions: 0
36:         Collisions: 0
37:         Collisions: 0
38:         Collisions: 0
39:         Collisions: 0
40:         Collisions: 0
41:         Collisions: 0

Finding the word: In
The word was found at index 14

Finding the word: Nice
The word was found at index 21

Finding the word: Me
```

Figure 3: Results



```
Select Microsoft Visual Studio Debug Console

The word was found at index 16

Finding the word: Banana
The word was not found.

C:\Users\eah170630\Desktop\C++\Debug\C++.exe (process 14964) exited with code 0.
Press any key to close this window . . .
```

Figure 4: Results