

**Project No.7: Hash Application on Student Information System**

Name	Student No.	Email	Responsibility
李志容	14346009	978282388@qq.com	Parts of codes, parts of the reports
谭笑	14346022	296309711@qq.com	Parts of codes, parts of the reports

Data: 2016.6.3

**1. Introduction**

This project is building a student information administration system. Students' information includes name, age, gender and ID, while the ID is used as a key. To build an efficient and useful system, we use hash tables to store information since hash tables are an efficient implementation of a keyed array data structure, a structure sometimes known as an associative array or map. A hash table is made up of two parts: an array and a mapping function, known as a hash function. In this project, we use mod function as hash function and solve the hash collision by two methods, which are separate chaining and open addressing. In addition, our program has a good user interface.

**2. Analysis and Design****1) Store information with hash function****a) Data structures**

To store the information, we use the data structure of Vector, because it can quickly find an element by index and when the size is fixed, it is very stable. Hash address value is important in storing the information, so if we calculate the hash address, we can directly operate on corresponding position in the vector.

In separate chaining method, we use linked-list to store the element that has the same hash address because it is easy to insert or remove an element on the list.

In open addressing method, we only use vector, but we also set a symbol for each position, to check whether this position is ACTIVE, DELETED or EMPTY.

**b) Important algorithms**

The most important algorithm is calculating the hash address value for every inserted information, given the ID. Firstly, we get the last 6 numbers of the ID string. Then turn the string of last 6 digits to integer value. At last, take its mod with the whole vector size, which is set in our project is 107 initially. The codes are follow:

```
template <typename HashObj>
int HashTable<HashObj>::hash(const string & ID) const{
    //according to the last 6 ID numbers, calculate the hash address
    string key = ID.substr(ID.length() - 6);
    int hashval = 0;
    for (int i = 0; i < key.length(); i++) {
        //calculate the hash address
        hashval = 10 * hashval + key[i];
    }
    hashval %= VList.size();
    if (hashval < 0)
        hashval += VList.size();
    return hashval;
}
```

### c) Justification

We only consider the last 6 digits assuming that it is nearly impossible that two different ID that have the same last 6 digits, which is more efficient.

### d) Time complexity

In the process of turning the string to integer, time costing is  $O(6)$ . Other time cost in calculating. So the overall time complexity is  $O(\text{constant})$ .

## 2) Resolve collision by two methods

### a) Separate chaining

#### i. Data structures

Separate chaining is to keep a list of all elements that hash to the same value. We can use the standard library list implementation. Since a doubled linked list is efficient in deletion and insertion, it is preferable in this project.

#### ii. Important algorithms

The first important algorithm is the **FIND** operation. To perform a find operation, we use the hash function to determine which list to traverse. Then search the appropriate list.

```

template <typename HashObj>
void HashTable<HashObj>::findKey(const string & s) {
    //find hash address of s
    int hashval = hash(s);
    //get the list according to coressponding hash address of s
    list<HashObj> &whichList = VList[hashval];
    list<HashObj>::iterator itr = whichList.begin();
    bool isContained = false;
    //traverse the corresponding list
    for ( ; itr != whichList.end(); itr++) {
        if ((*itr).getID() == s) {
            isContained = true;
            //print the information that is found
            cout << "Found successfully: " << endl;
            cout << "Name: " << (*itr).getName() << endl;
            cout << "ID: " << (*itr).getID() << endl;
            cout << "Age: " << (*itr).getAge() << endl;
            cout << "Gender: " << ((*itr).getSex() == 0? "Male":"Female") << endl;
        }
    }
    if (!isContained) cout << s << " This ID is not contained in the system" << endl;
}

```

The second important algorithm is the **INSERT** operation. To perform an insert, we check the appropriate list to see whether the element is already in place. If the element turns out to be new, it can be inserted at the front of the list, since it is convenient and also because frequently it happens that recently inserted elements are the most likely to be accessed in the near future.

```

template <typename HashObj>
bool HashTable<HashObj>::insert(const HashObj & x) {
    //get the list according to coressponding hash address of the object x
    list<HashObj> &whichList = VList[myhash(x)];
    //if it isn't stored, store it
    if (!contains(x)) {
        whichList.push_back(x);
    }
    return true;
}

```

### iii. Time complexity

Suppose that the average length for each list is  $\beta$ , then the FIND operation need to use  $O(\text{constant})$ , which is used to calculate the hash value and  $O(\beta)$ , the average time use to find the correct position. So the total time is  $O(\text{constant})$ .

The INSERT operation is the same.

## b) Open addressing

### i. Data structures

Open addressing is to resolve collisions by the method of probing, which means searching for alternative positions in the array to place the key. There are three main types of open addressing——linear

probing, quadratic probing and double probing. In this project, we choose to use linear probing which searches the table for the nearest following free position to insert the new key.

## ii. Important algorithms

The first important algorithm is the **FIND** operation. This function finds the hash address given a key string and output the corresponding information of that position. First, find the original position by hash function. Then resolve the collisions if any. If the key exists in the table, output the information corresponding to the key. Otherwise output the error information.

```
void findKey(const string& x) {
    int currentPos = hash(x); //original position
    while (array[currentPos].info != EMPTY && array[currentPos].element.getID() != x) {
        //terminate when the position is found or x already exists in the table
        currentPos += 1; //linear probing
        if (currentPos >= array.size()) //mod operation
            currentPos -= array.size();
    }
    if (isActive(currentPos)) {
        cout << "Found successfully: " << endl; //print the information that is found
        cout << "Name: " << array[currentPos].element.getName() << endl;
        cout << "ID: " << array[currentPos].element.getID() << endl;
        cout << "Age: " << array[currentPos].element.getAge() << endl;
        cout << "Gender: " << (array[currentPos].element.getSex() == 0? "Male": "Female") << endl;
    }
    else
        cout << "Fail" << endl;
}
```

The second important algorithm is the **INSERT** operation. This function inserts a key into the hash table. If the insertion succeeds, return true. If it fails which means the key already exists in the table, return false. First, find the position to insert the key. If that position is marked as active, it means the insertion fails. If not, insert the key to that position and mark it as active. Then if the current size is larger than half the array size, double the array size because we need the load factor to be less than 0.5.

```
bool insert(const HashObj& x) { //if succeed, return 1
    int currentPos = findPos(x);
    if (isActive(currentPos)) //x already existed in the table
        return false;
    array[currentPos] = HashEntry(x, ACTIVE); //insert the entry
    if (++currentSize > array.size() / 2) //if current size is larger than half the array size
        rehash(); //enlarge the size of the array
    return true;
}
```

## iii. Time complexity

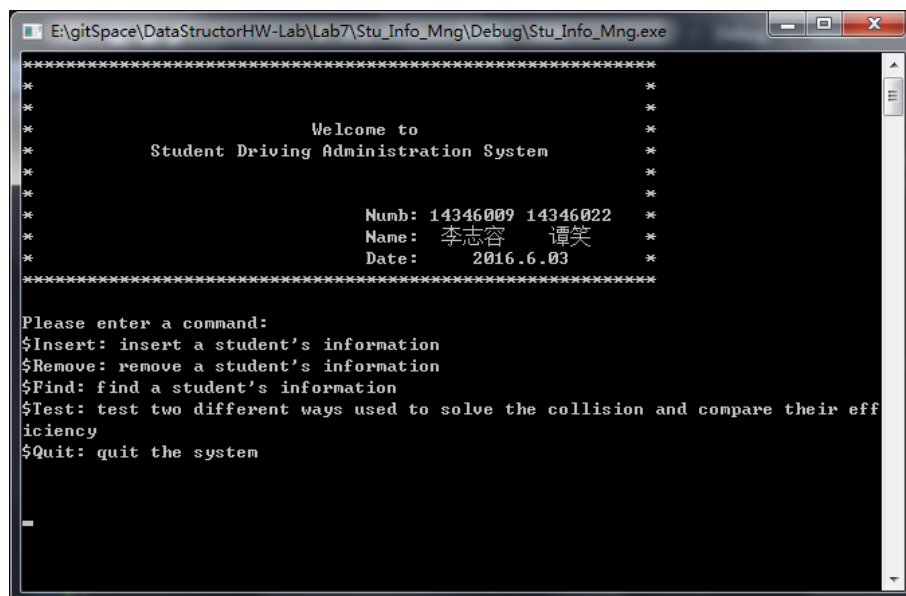
The time complexities for both **FIND** and **INSERT** operations are  $O(\text{constant})$ .

### c) Comparison

These two methods differ mainly in data structures. Separate chaining uses linked list. The table size can be smaller than the number of keys. Deletion is convenient. Open addressing finds alternative position. It's more efficient for a small number of keys. But it can easily cause clustering.

## 3. Test

### a) Initial interface:

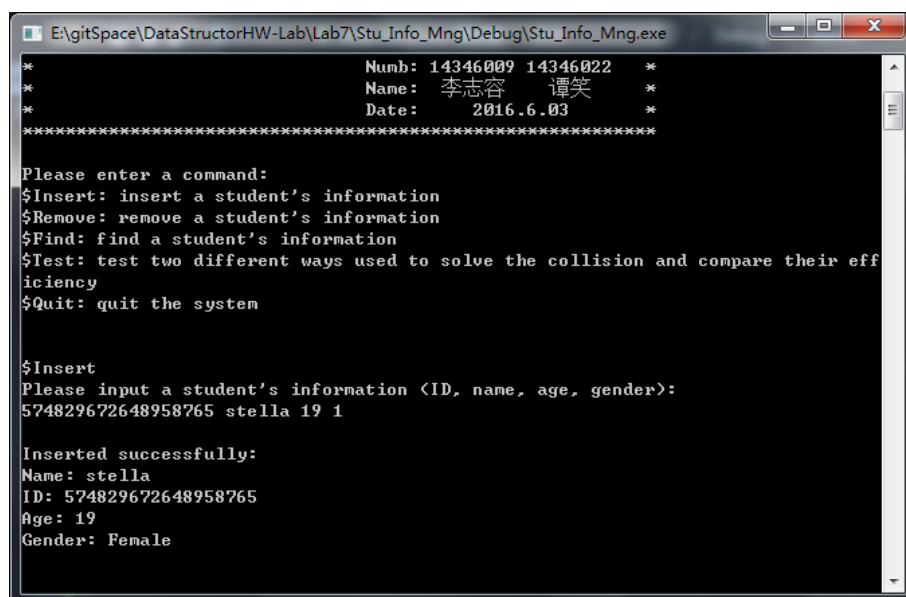


```
E:\gitSpace\DataStructorHW-Lab\Lab7\Stu_Info_Mng\Debug\Stu_Info_Mng.exe
*****
*                               *
*                               *
*      Welcome to              *
*      Student Driving Administration System
*                               *
*                               *
*      Numb: 14346009 14346022 *
*      Name: 李志容 谭笑      *
*      Date: 2016.6.03        *
*                               *
*****

Please enter a command:
$Insert: insert a student's information
$Remove: remove a student's information
$Find: find a student's information
$Test: test two different ways used to solve the collision and compare their efficiency
$Quit: quit the system

-
```

### b) Insert a student's information:

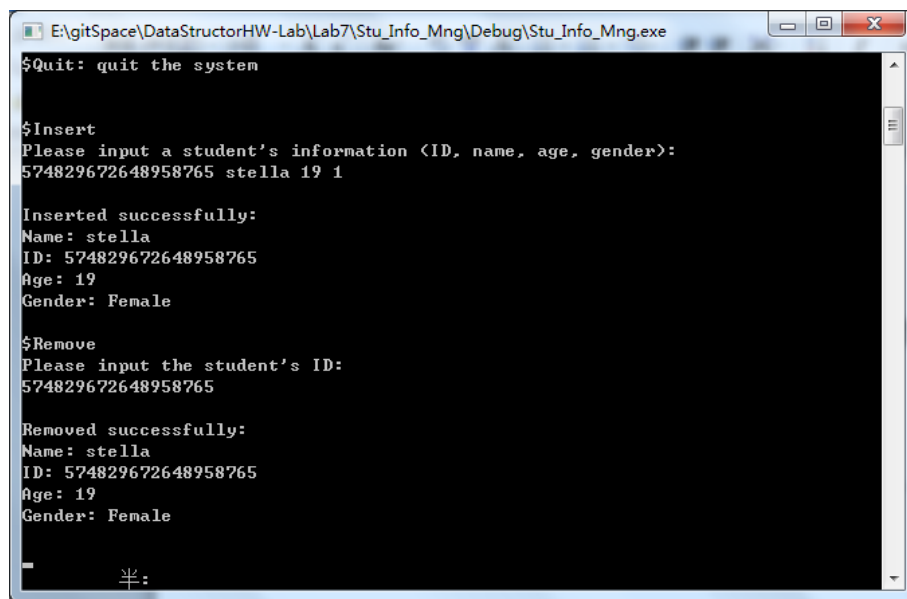


```
E:\gitSpace\DataStructorHW-Lab\Lab7\Stu_Info_Mng\Debug\Stu_Info_Mng.exe
*****
*                               *
*                               *
*      Welcome to              *
*      Student Driving Administration System
*                               *
*                               *
*      Numb: 14346009 14346022 *
*      Name: 李志容 谭笑      *
*      Date: 2016.6.03        *
*                               *
*****

Please enter a command:
$Insert: insert a student's information
$Remove: remove a student's information
$Find: find a student's information
$Test: test two different ways used to solve the collision and compare their efficiency
$Quit: quit the system

$Insert
Please input a student's information (ID, name, age, gender):
574829672648958765 stella 19 1

Inserted successfully:
Name: stella
ID: 574829672648958765
Age: 19
Gender: Female
```

**c) Remove a student's information:**

```
E:\gitSpace\DataStructorHW-Lab\Lab7\Stu_Info_Mng\Debug\Stu_Info_Mng.exe
$Quit: quit the system

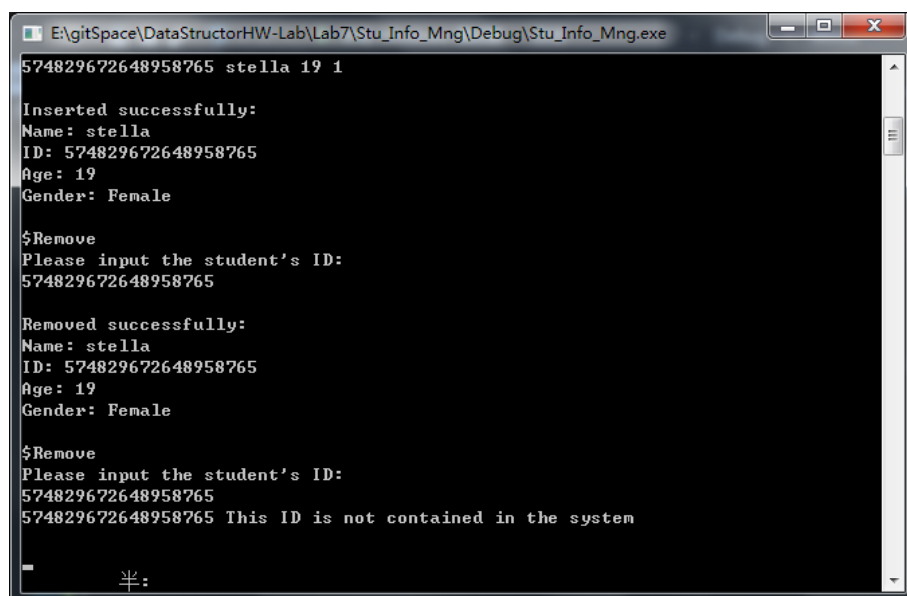
$Insert
Please input a student's information (ID, name, age, gender):
574829672648958765 stella 19 1

Inserted successfully:
Name: stella
ID: 574829672648958765
Age: 19
Gender: Female

$Remove
Please input the student's ID:
574829672648958765

Removed successfully:
Name: stella
ID: 574829672648958765
Age: 19
Gender: Female

半:
```

**d) Remove a non-existed information:**

```
E:\gitSpace\DataStructorHW-Lab\Lab7\Stu_Info_Mng\Debug\Stu_Info_Mng.exe
574829672648958765 stella 19 1

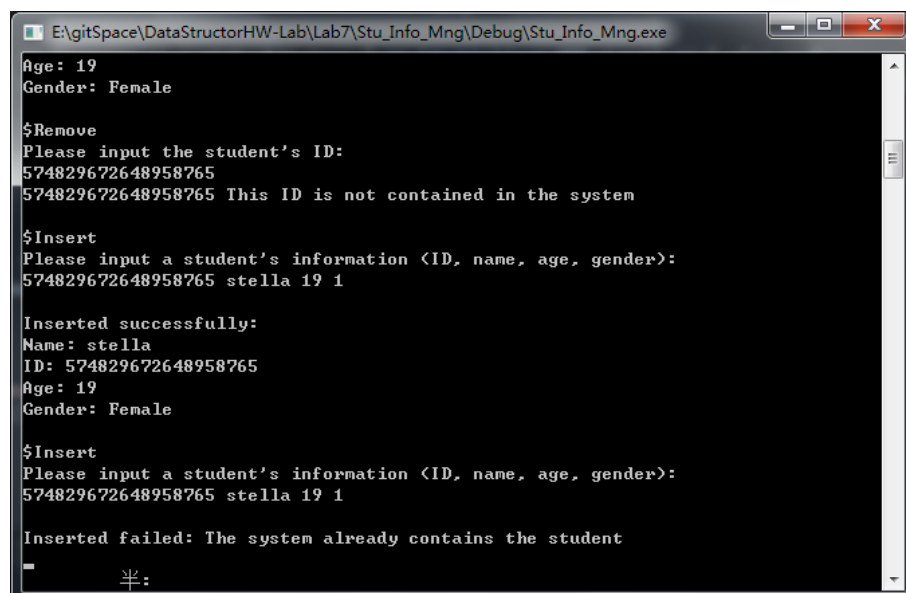
Inserted successfully:
Name: stella
ID: 574829672648958765
Age: 19
Gender: Female

$Remove
Please input the student's ID:
574829672648958765

Removed successfully:
Name: stella
ID: 574829672648958765
Age: 19
Gender: Female

$Remove
Please input the student's ID:
574829672648958765
574829672648958765 This ID is not contained in the system

半:
```

**e) Insert an existed information:**

```
E:\gitSpace\DataStructorHW-Lab\Lab7\Stu_Info_Mng\Debug\Stu_Info_Mng.exe
Age: 19
Gender: Female

$Remove
Please input the student's ID:
574829672648958765
574829672648958765 This ID is not contained in the system

$Insert
Please input a student's information (ID, name, age, gender):
574829672648958765 stella 19 1

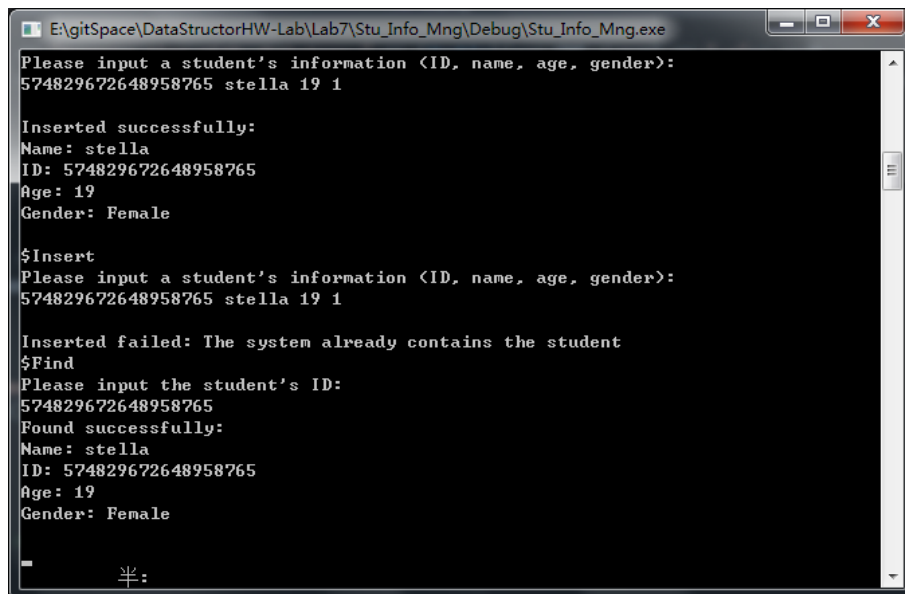
Inserted successfully:
Name: stella
ID: 574829672648958765
Age: 19
Gender: Female

$Insert
Please input a student's information (ID, name, age, gender):
574829672648958765 stella 19 1

Inserted failed: The system already contains the student

半:
```

## f) Find the information given an ID:



```

E:\gitSpace\DataStructorHW-Lab\Lab7\Stu_Info_Mng\Debug\Stu_Info_Mng.exe
Please input a student's information (ID, name, age, gender):
574829672648958765 stella 19 1

Inserted successfully:
Name: stella
ID: 574829672648958765
Age: 19
Gender: Female

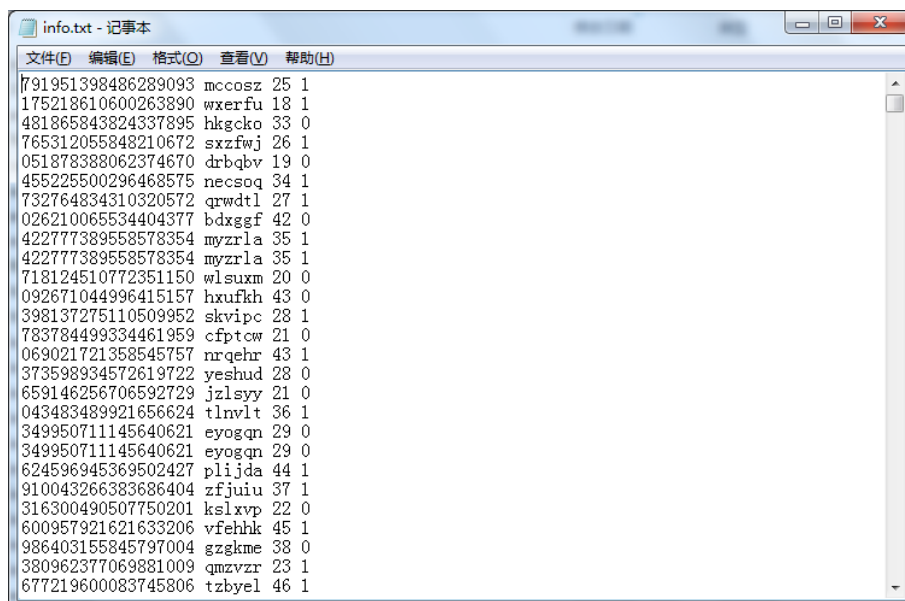
$Insert
Please input a student's information (ID, name, age, gender):
574829672648958765 stella 19 1

Inserted failed: The system already contains the student
$Find
Please input the student's ID:
574829672648958765
Found successfully:
Name: stella
ID: 574829672648958765
Age: 19
Gender: Female

```

## g) Test the finding time that using chaining and open addressing:

Firstly, input the information from the file:



```

info.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
791951398486289093 mcosz 25 1
175218610600263890 wxerfu 18 1
481865843824337895 hkgcko 33 0
765312055848210672 sxzfwj 26 1
051878388062374670 drbqbv 19 0
455225500296468575 necsoq 34 1
732764834310320572 qrwtdl 27 1
026210065534404377 bdxggf 42 0
422777389558578354 myzrla 35 1
422777389558578354 myzrla 35 1
718124510772351150 wlsuxm 20 0
092671044996415157 hxufkh 43 0
398137275110509952 skvipc 28 1
783784499334461959 cfptcw 21 0
069021721358545757 nrqehr 43 1
373598934572619722 yeshud 28 0
659146256706592729 jzlsyy 21 0
043483489921656624 tlnvlt 36 1
349950711145640621 eyogqn 29 0
349950711145640621 eyogqn 29 0
624596945369502427 plijda 44 1
910043266383686404 zfjuuu 37 1
316300490507750201 kslxvp 22 0
600957921621633206 vfehkh 45 1
986403155845797004 gzkme 38 0
380962377069881009 qmzvzr 23 1
677219600083745806 tzbyel 46 1

```

Then randomly find the information given a random ID. Test for 30 times and calculate average time:

When the number of data in the input file (info.txt) is 2000:

```

D:\tao\file\data structures and algorithms\Lab7\Lab7\Stu_Info_Mng\Debug\Stu_Info_Mng...
$Test
Please wait for inputing the information from the file.
Input finished
Total numbers of student: 2000
chaining:      0 s   open addressing:  0.001 s
chaining:      0 s   open addressing:  0.002 s
chaining:      0 s   open addressing:  0.003 s
chaining:      0 s   open addressing:  0.002 s
chaining:      0 s   open addressing:  0.002 s
chaining:      0 s   open addressing:  0.003 s
chaining:      0 s   open addressing:  0.002 s
chaining:      0 s   open addressing:  0.002 s
chaining:      0 s   open addressing:  0.003 s
chaining:      0 s   open addressing:  0.001 s
chaining:      0 s   open addressing:  0.003 s
chaining:      0 s   open addressing:  0.002 s
chaining:      0 s   open addressing:  0.001 s
chaining:      0 s   open addressing:  0.003 s
chaining:      0 s   open addressing:  0.002 s
chaining:      0.001 s open addressing:  0.002 s
chaining:      0 s   open addressing:  0.003 s
chaining:      0.001 s open addressing:  0.001 s
chaining:      0 s   open addressing:  0.003 s
chaining:      0 s   open addressing:  0.003 s
chaining:      0 s   open addressing:  0.004 s
chaining:      0 s   open addressing:      0 s
chaining:      0 s   open addressing:  0.001 s
chaining:      0 s   open addressing:  0.002 s
chaining:      0 s   open addressing:  0.003 s
chaining:      0 s   open addressing:  0.001 s
chaining:      0 s   open addressing:  0.002 s
chaining:      0 s   open addressing:  0.003 s
chaining:      0 s   open addressing:  0.002 s
chaining:      0 s   open addressing:  0.001 s
Average time with chaining: 0 s
Average time with open addressing: 0.00186667 s

```

When the number of data in the input file (info.txt) is more than 7000:

```

$I
Please wait for inputing the information from the file.
Input finished
Total numbers of student: 7388
chaining:      0 s   open addressing:  0.002 s
chaining:      0 s   open addressing:  0.006 s
chaining:      0 s   open addressing:  0.004 s
chaining:      0.001 s open addressing:  0.003 s
chaining:      0 s   open addressing:  0.004 s
chaining:      0.001 s open addressing:  0.004 s
chaining:      0.001 s open addressing:  0.004 s
chaining:      0.001 s open addressing:  0.003 s
chaining:      0 s   open addressing:  0.003 s
chaining:      0.001 s open addressing:  0.002 s
chaining:      0 s   open addressing:  0.005 s
chaining:      0 s   open addressing:  0.008 s
chaining:      0 s   open addressing:  0.006 s
chaining:      0.001 s open addressing:  0.006 s
chaining:      0 s   open addressing:  0.012 s
chaining:      0 s   open addressing:  0.006 s
chaining:      0 s   open addressing:  0.003 s
chaining:      0 s   open addressing:  0.003 s
chaining:      0 s   open addressing:  0.002 s
chaining:      0 s   open addressing:  0.002 s
chaining:      0 s   open addressing:  0.01 s
chaining:      0 s   open addressing:  0.013 s
chaining:      0 s   open addressing:  0.009 s
chaining:      0 s   open addressing:  0.005 s
chaining:      0 s   open addressing:  0.006 s
chaining:      0.001 s open addressing:  0.007 s
chaining:      0 s   open addressing:  0.01 s
chaining:      0 s   open addressing:  0.008 s
chaining:      0.001 s open addressing:  0.006 s
chaining:      0 s   open addressing:  0.005 s
Average time with chaining: 0.0001 s
Average time with open addressing: 0.00603333 s

```



## 4. Conclusion and Discussion

### Conclusion:

From the result above, we find that the more information that stored in the file, the more time it needs to find specific information. In addition, whether the capacity of information that stored in the file is large or not, the cost time using separating chaining is much less than using open addressing.

### Achieved:

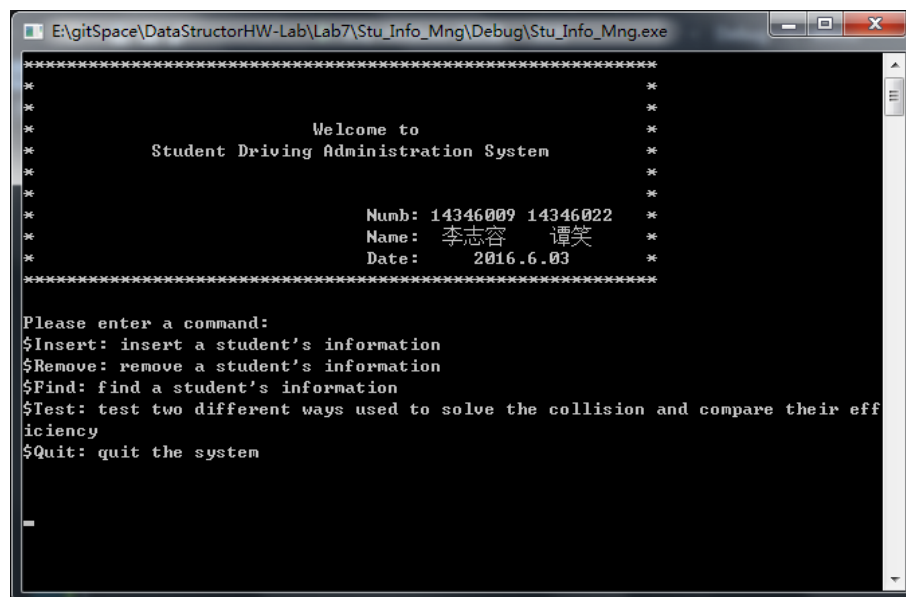
We have achieved to manage the examinees' information by using hash table and resolve the collisions by two methods——separate chaining and open addressing.

### Not achieved:

We didn't use two methods to insert and remove information in the main function. We only use separate chaining.

### Highlights:

- a) We have designed a user-friendly interface which allows the users to choose any operations they want.



```
*****
*                                     *
*                                     *
*           Welcome to               *
*   Student Driving Administration   *
*                                     *
*                                     *
*           Numb: 14346009 14346022  *
*           Name: 李志容 谭笑        *
*           Date: 2016.6.03          *
*                                     *
*****

Please enter a command:
$Insert: insert a student's information
$Remove: remove a student's information
$Find: find a student's information
$Test: test two different ways used to solve the collision and compare their efficiency
$Quit: quit the system
```

- b) We have randomly generated a database of more than 7000 examinees. We read it when comparing the efficiencies of the two methods. Only if we have a large amount of data can we compare them more clearly.

## 5. Appendices

Refer to "codes-hash.pdf".

## 6. References

- a) 《数据结构与算法实验实践教程》——乔海燕、蒋爱军、高集荣、刘晓铭
- b) 《Data Structures and Algorithm Analysis in C++》——Mark Allen Weiss
- c) 《Introduction to Programming with C++》——Y. Daniel Liang
- d) [https://en.wikipedia.org/wiki/Open\\_addressing](https://en.wikipedia.org/wiki/Open_addressing)
- e) [https://en.wikipedia.org/wiki/Linear\\_probing](https://en.wikipedia.org/wiki/Linear_probing)
- f) [http://soj.sysu.edu.cn/show\\_problem.php?pid=1007&cid=569](http://soj.sysu.edu.cn/show_problem.php?pid=1007&cid=569)