Separate Chaining:

HashTable.h

```
#ifndef HASHTABLE_H
#define HASHTABLE_H

#include "Student.h"
#include <iostream>
#include <vector>
#include <list>
#include <ctime>
using namespace std;

template <typename HashObj>
class HashTable { //seperate chaining
public:
	explicit HashTable(int size = 107);	//constructor

	void makeEmpty();	//set the information of all elements to be EMPTY
	void print();	//print all the information that is stored

	const vector<list<HashObj> > & getList() const{	//get the whole vector that stored the information
		return VList;
	}

	bool contains(const HashObj & x) const;	//check if the vector contains the object x
	bool insert(const HashObj & x);	//insert the object x
	void findKey(const string & s);	//given key value find its whole information
	void remove(const string & s);	//given key value remove its whole information
	double cal_time(const string & s);	//calculate the time that use to find a key
private:
	vector<list<HashObj> > VList;	//Use vector to store the students' information
	int hash(const string & key) const;	//given key value find its hash address
	int myhash(const HashObj & x) const;	//given hashObject find its hash address
};

template <typename HashObj>
```

```
HashTable<HashObj>::HashTable(int size = 107) {    //constructor
	VList.resize(size);
	makeEmpty();
}

template <typename HashObj>
void HashTable<HashObj>::makeEmpty() {        //clear the contain
	for (int i = 0; i < VList.size(); i++) {
		VList[i].clear();
	}

}

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

template <typename HashObj>
void HashTable<HashObj>::print() {        //print all the information that is stored
	const vector<list<HashObj> > V = getList();
	for (int i = 0; i < V.size(); i++) {
		list<HashObj> l = V[i];
		if (!l.empty()) {

			cout << i << " ";
			for (list<HashObj>::iterator j = l.begin(); j != l.end(); j++) {
				if ((*j).getName() != "")
					cout << (*j).getName() << " ";
			}
				cout << endl;
		}
	}
}
```

```
template <typename HashObj>
int HashTable<HashObj>::myhash(const HashObj & x) const {      //given hashObject
find its hash address
      string ID = x.getID();      //get the ID from hash object
      int hashval = hash(ID);      //find its hash address
      return hashval;
}

template <typename HashObj>
bool HashTable<HashObj>::contains(const HashObj & x) const {
      const list<HashObj> &whichList = VList[myhash(x)];      //get the list according to
coressponding hash address of the object x
      return (find(whichList.begin(), whichList.end(), x) != whichList.end());
}


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

template <typename HashObj>
void HashTable<HashObj>::findKey(const string & s) { //find the hash position given a
key and output the information
      int hashval = hash(s);       //find hash address of s
      list<HashObj> &whichList = VList[hashval];    //get the list according to
coressponding hash address of s
      list<HashObj>::iterator itr = whichList.begin();
      bool isContained = false;
      for ( ; itr != whichList.end(); itr++) {    //traverse the corresponding list
            if ((*itr).getID() == s) {
                  isContained = true;
                  cout << "Found successfully: " << endl;    //print the information that is
found
                  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;
}

template <typename HashObj>
double HashTable<HashObj>::cal_time(const string & s) { //calculate time used to
search for the key in the table

        DWORD start, end;
        start = timeGetTime();
        int hashval = hash(s);        //find hash address of s
        list<HashObj> &whichList = VList[hashval];    //get the list according to
coressponding hash address of s
        list<HashObj>::iterator itr = whichList.begin();
        bool isContained = false;
        for ( ; itr != whichList.end(); itr++) {    //traverse the corresponding list
            if ((*itr).getID() == s)
                isContained = true;
        }
        if (!isContained) cout << s <<" This ID is not contained in the system" << endl;
        end = timeGetTime();
        return (double)((double)(end - start) / 1000);
}

template <typename HashObj>
void HashTable<HashObj>::remove(const string & s) {
        int hashval = hash(s);
        list<HashObj> &whichList = VList[hashval];
        list<HashObj>::iterator itr = whichList.begin();
        bool isContained = false;
        for ( ; itr != whichList.end(); itr++) {      //traverse the corresponding list
            if ((*itr).getID() == s) {
                isContained = true;
                cout << "\nRemoved successfully: " << endl;    //print the information
that is removed
                cout << "Name: " << (*itr).getName() << endl;
                cout << "ID: " << (*itr).getID() << endl;
                cout << "Age: " << (*itr).getAge() << endl;
                cout << "Gender: " << ((*itr).getSex() == 0? "Male":"Female") <<    endl;
                whichList.erase(itr);
                break;
```

```
            }
        }
        if (!isContained) cout << s <<" This ID is not contained in the system" << endl;

}

#endif HASHTABLE_H
```

Open Addressing:

HashTable2.h

```
#include <vector>
#include <cmath>
#include <string>
#include <ctime>
using namespace std;

template <typename HashObj> //int id
class HashTable2 { //open addressing
public:
    explicit HashTable2(int size = 101) { //constructor
        array.resize(nextPrime(size)); //set the array size to be the next prime
number after size
        makeEmpty();
    }

    bool contains(const HashObj& x) const { //if the table contains the object x
        return isActive(findPos(x)); //if the position of x is active
    }

    void makeEmpty() { //clear the table
        currentSize = 0;
        for (int i = 0; i < array.size(); i++) //set the information of all elements to be
EMPTY
            array[i].info = EMPTY;
    }

    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;
        }

        bool remove(const string& x) { //if succeed, return 1
            int currentPos = hash(x); //original position
            while (array[currentPos].info != EMPTY && array[currentPos].element.id !=
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))
                array[currentPos].info = DELETED; //set the information to be DELETED
            else
                return false;
        }

        void findKey(const string& x) { //find the hash position given a key and output
the information
            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;
                //output the information
            }

            else //output error information
                cout << "Fail" << endl;
        }
```

```
    double cal_time(const string& x) {
        DWORD start, end;
        start = timeGetTime();
        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 << "Fail" << endl;

        end = timeGetTime();
        //cout << (double)((double)(end - start) / 1000) << endl;
        return (double)((double)(end - start) / 1000);
    }

    enum EntryType {ACTIVE, EMPTY, DELETED}; //three state of entry
    struct HashEntry {
        HashObj element;
        EntryType info;

        HashEntry(const HashObj& e = HashObj(), EntryType i = EMPTY)
{ //constructor of HashEntry
            element = e;
            info = i;
        }
    };

    vector<HashEntry> &getArray() {
        return array;
    }

private:
    vector<HashEntry> array;
    int currentSize;

    bool isActive(int currentPos) const { //if the current position is ACTIVE
        return array[currentPos].info == ACTIVE;
    }
```

```
    int findPos(const HashObj& x) const { //find the position in the table
corresponding to x
        int currentPos = myhash(x); //original position
        while (array[currentPos].info != EMPTY && array[currentPos].element != 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();
        }
        return currentPos;
    }

    void rehash() { //double the size of the array
        vector<HashEntry> oldArray = array; //to store the old array
        array.resize(nextPrime(2 * oldArray.size())); //set the array size to be the
next prime number after twice old size
        for (int j = 0; j < array.size(); j++) //set the information of all elements of the
new array to be EMPTY
            array[j].info = EMPTY;
        currentSize = 0;
        for (int i = 0; i < oldArray.size(); i++) //insert the elements in the old array
into the new array
            if (oldArray[i].info == ACTIVE)
                insert(oldArray[i].element);
    }

    int hash(const string& x) const { //hash function with argument of string type
        int xVal = 0; //to store the value of the string
        for (int i = 0; i < 3; i++) //last 4 bits since the array size is 101
            xVal = xVal + pow(10, i) * (x[x.size() - 1 - i] - '0');
        int currentPos = xVal % array.size(); //mod operation
        return currentPos;
    }

    int myhash(const HashObj& x) const { //hash function with argument of HashObj
type
        string xStr = x.getID(); //get the key string of the object
        return hash(xStr);
    }
};

bool isPrime(int n) { //if n is a prime number
    for (int i = 2; i <= n / 2; i++)
        if (n % i == 0) //if n has any factor no less than 2
```

```
            return false;
    return true;
}

int nextPrime(int n) { //to find the next prime number after n
    while (!isPrime(n)) //end loop when n is prime
        n++;
    return n;
}
```

Student.h

```
#ifndef   STUDENT_H
#define STUDENT_H

#include <string>
#include <iostream>
using namespace std;

class Student {
public:
    Student(string id="", string n="", int a=0, bool s=0) {
        ID = id;
        name = n;
        age = a;
         gender = s;
    }

    const string & getID() const{
        return ID;
    }

    const string & getName() const{
        return name;
    }

    const int & getAge() const{
        return age;
    }

    const bool & getSex() const{
        return gender;
    }
```

```
    bool operator == (const Student & stu) const{
        return (getID() == stu.getID());
    }

    bool operator != (const Student & stu) const {
        return !(*this == stu);
    }


private:
    string ID;      //student's ID nember
    string name;    //student's name
    int age;        //student's age
    bool gender; //0 stands for male, 1 stands for female

};

#endif STUDENT_H
```

源.cpp

```
#include "HashTable.h"
#include "HashOpenAddress.h"
#include "HashTable2.h"
#include <iostream>
#include <ctime>
#include <cstring>
#include <fstream>
#include <ostream>
#include <windows.h>
#include <sstream>
#include <iomanip>
#include <Mmsystem.h>
#pragma comment(lib, "Winmm.lib")
using namespace std;
long long totalNumber;

void create_stu() { //create a database of students' information
    std::ofstream outfile;
    outfile.open("E://info.txt", ios::ate);
    for (int i = 0; i < 3000; i++) {
        Sleep(900); //the period is 0.9s
```

```
        srand(time(0));
        string id = "";
        string name = "";
        int age;
        int sex;

        for (int i = 0; i < 18; i++) { //id is a 18-bit string
            id += '0'+ rand() % 10;
        }
        for (int i = 0; i < 6; i++) { //name is a 6-bit string
            name += 'a' + rand() % 26;
        }

        age = rand() % 30 + 18; //age ranges from 0 to 47
        sex = rand() % 2; //0 stands for male while 1 stands for female
        outfile << id << " " << name << " " << age << " " << sex << endl;
    }
    outfile.close();

    Student stu("", "", 0, 0);
}

int str_to_i(string s) { //convert string to integer
    int res = 0; //result
    for (int i = 0; i < s.length(); i++) //digit by digit
        res = res * 10 + (s[i]-'0');
    return res;
}


void read_stu(HashTable<Student> &h, HashTable2<Student> & h2) { //read
students' information from the database generated
    std::ifstream infile;
    infile.open("E://info.txt", ios::in);
    string line;

    while(getline(infile, line)) { //read line by line
        string id = "";
        string name = "";
        int age;
        int sex;
        string temp = "";
        int len = line.length();
        int i;
```

```
            id = line.substr(0, 18);
            name = line.substr(19, 6);

            temp = line.substr(26, 2);
            age = str_to_i(temp);

            temp = line[29];
            if (temp[0] == '0') sex = 0;
            else sex = 1;

            Student stu(id, name, age, sex);
            Student info(id, name, age, sex);
            //create two objects

            h.insert(stu);
            h2.insert(info);
            //insert them into two hash tables respectively

            totalNumber++;
            if (infile.eof()) {
                  break;
            }
      }

      infile.close();
}

void welcome() { //welcome interface
      cout
<<"**********************************************************" <<
endl;
      cout <<"*
*" << endl;
      cout <<"*
*" << endl;
      cout <<"*                                  Welcome to
*" << endl;
      cout <<"*              Student Driving Administration System              *" <<
endl;
      cout <<"*
*" << endl;
      cout <<"*
*" << endl;
      cout <<"*                                  Numb: 14346009 14346022
```

```
*" << endl;
	cout <<"*                                        Name:    李志容       谭笑
*" << endl;
	cout <<"*                                        Date:        2016.6.05          *"
<< endl;
	cout
<<"************************************************************" <<
endl;
	cout << endl;
}

void command_tips() { //command interface
	cout << "Please enter a command:" << endl;
	cout << "$Insert: insert a student's information" << endl;
	cout << "$Remove: remove a student's information" << endl;
	cout << "$Find: find a student's information" << endl;
	cout << "$Test: test two different ways used to solve the collision and compare
their efficiency" << endl;
	cout << "$Quit: quit the system" << endl;
	cout << endl;
}

int main() {
	//create_stu();
	welcome();
	command_tips();
	cout << endl;
	srand(time(0));

	HashTable<Student> h;
	HashTable2<Student> table;
	//create two hash tables

	//read_stu(h, table);

	string command;
	while(cin >> command) {
		if (command[0] == '$') { //a command starts with a '$"
			if (command[1] == 'Q') break; //$Quit
			switch(command[1]) {
			case 'I': //$Insert
				{
					cout << "Please input a student's information (ID, name, age,
gender):" << endl;
```

```
                    string id, name;
                    int age;
                    bool sex;
                    cin >> id >> name >> age >> sex;
                    Student student(id, name, age, sex);
                    if (h.insert(student)) {
                        cout << "\nInserted successfully: " << endl;    //print the
information that is removed
                        cout << "Name: " << student.getName() << endl;
                        cout << "ID: " << student.getID() << endl;
                        cout << "Age: " << student.getAge() << endl;
                        cout << "Gender: " << (student.getSex() == 0?
"Male":"Female") << endl <<    endl;
                    } else {
                        cout << "\nInserted failed: The system already contains the
student" << endl;
                    }

                }
                break;
            case 'R': //$Remove
                {
                    cout << "Please input the student's ID:" << endl;
                    string s2;
                    cin >> s2;
                    h.remove(s2);
                    cout << endl;
                }
                break;
            case 'F': //$Find
                {
                    cout << "Please input the student's ID:" << endl;
                    string s2;
                    cin >> s2;
                    h.findKey(s2);
                    cout << endl;
                }
                break;
            case 'T': //$Test
                {
                    cout << "Please wait for inputing the information from the file."
<< endl;
                    read_stu(h, table); //It cost some time since there is a large
amount of information
```

```
                        cout << "Input finished" << endl;
                        cout << "Tatal numbers of student: " << totalNumber << endl;
                        vector<list<Student> > ve = h.getList();
                        double ave_CHI = 0, ave_OPEN = 0; //average time used for
two methods
                        int num = 30; //number to search
                        for (int i = 0; i < num; i++) {
                            list<Student>::iterator p = ve[rand() % ve.size()].end();
                            string s1 = (*(--p)).getID();
                            double CHAIN, OPEN; //time used for two methods
                            ave_CHI += (double)h.cal_time(s1) ;
                            ave_OPEN += (double)table.cal_time(s1);
                            cout << "chaning: " << setw(8) << (double)h.cal_time(s1)
<< " s" << "      open addressing: " << setw(8) <<
                                    (double)table.cal_time(s1) << " s" << endl;
                            Sleep(100); //the period is 0.1s
                        }
                        cout << "Average time with chaining: " << (double)(ave_CHI /
num) << " s"<< endl;
                        cout << "Average time with open addressing: " <<
(double)(ave_OPEN / num) << " s" << endl;


                        cout << endl;
                    } break;
                default:
                    {
                        cout << "Command error! Please enter a right command!" <<
endl;
                        command_tips();
                    } break;
                }
            } else { //not a command
                cout << "Command error! Please enter a right command!" << endl;
                command_tips();
            }
        }

        //574829672648958765 stella 19 1
        system("pause");
        return 0;
}
```