

# 数据结构课程设计报告

设计题目：散列表的设计与实现

学生姓名：朱铭宇

专 业：信息安全

班 级：20-1班

学 号：2020210574

指导教师：胡学钢

完成日期：2021.6.20

# （一）需求和规格说明

**问题描述：**设计散列表实现电话号码查找系统。

**编程任务：**

1) 设每个记录有下列数据项：用户名、电话号码、地址；

2) 从键盘输入各记录，以用户名（汉语拼音形式）为关键字建立散列表；3) 采用一定的方法解决冲突；

4) 查找并显示给定电话号码的记录；

5) （选作）设计不同的散列函数，比较冲突率；

6) （选作）在散列函数确定的前提下，尝试各种不同类型处理冲突的方法，

考察平均查找长度的变化。

# （二）设计

## 1．设计思想

本题主要考察散列表的构建与实现，并且实现相应的查找功能。选到这个题目之后，我的第一想法是采用开放地址法实现散列表，然后就开始学习清华大学邓俊辉老师的书，并根据他的实现方式，构建了自己的散列表。首先是选用了位图(Bitmap)这一结构，通过散列函数获取散列码的过程中需要一个素数，而每次构建的时候指定一个素数未免有些麻烦，我需要快速获得一个素数，并且要求占用的空间尽可能小，所以选择了位图来存储素数。然后要以用户名为关键字简历散列表，所以选择适当的针对字符串string的散列函数尤为重要。解决冲突的方式就是线性探查法，在哈希因子大于散列表容量一半的时候就进行重散列rehash操作。

## 2. 设计表示

1. 位图(Bitmap)结构：

|  |  |  |  |
| --- | --- | --- | --- |
| **结构体名称** | **成员数据类型** | **成员名** | **描述** |
| Bitmap | char\* | M | 位图元素 |
| int | N | 位图容量 |
|  |  | Bitmap() | 调用init()函数 |
|  |  | Bitmap(char\* file,int n=8) | 按指定规模或默认规模，从指定文件中读入位图 |
|  | Void | Init() | 初始化位图 |
|  |  | ~Bitmap() | 析构，释放位图空间 |
|  | Void | Expand() | 若被访问的bitmap[k]越界，则需要扩容 |
|  | Void | Eratosthenes(int n,char\* file) | 将0~n的数中为合数的数进行标记，并写入外部文件 |

1. 类：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类名** | **成员类别** | **类型** | **成员名** | **描述** |
| Entry | 函数 | int | length() | 链表长度（结点数） |
| string | get\_word(int i) | 取第i个结点的数据 |
|
|
|
|
|
|
| 数据 | K | key | 关键码 |
| V | Value | 值 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类名** | **成员类别** | **类型** | **成员名** | **描述** |
| hashtable | 函数 | int | Probe4hit() | 沿关键码k对应的试探链，找到词条匹配的桶 |
| Int | Probe4free() | 沿关键码k对应的试探链，找到首个可用空桶 |
| Void | Rehash() | 重散列 |
| Int | Size() | 获取词条数量 |
| Bool | Put() | 对散列表进行插入操作 |
| V | Get() | 对散列表进行查找操作 |
| node\* | get\_rear() | 取尾指针 |
| 数据 | Entry<K,V>\*\* | Ht | 桶数组，存放词条数组 |
| Int | M | 桶数组容量 |
| Int | N | 词条数量 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类名** | **成员类别** | **类型** | **成员名** | **描述** |
| TelephoneNumber | 函数 |  | TelephoneNumber() | 从指定文件中读取用户信息，并且存入散列表中 |
| Void | Showmenu() | 展示菜单 |
| Void | Exitsystem() | 退出程序 |
| Void | Push() | 调用散列表的插入操作 |
| Void | Find() | 调用散列表的查找操作 |
| 数据 | Hashtable<string,long long> | H | 内部存放的散列表 |

1. 函数：

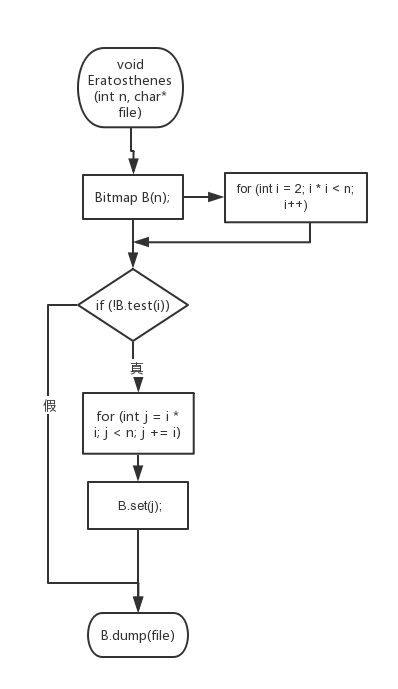
|  |  |  |
| --- | --- | --- |
| **数据类型** | **函数名称** | **描述** |
| Void | Eratosthenes(int n,char\* file) | 将0~n的数中为合数的数进行标记，并写入外部文件 |
| void | Push\_in\_file() | 将散列表中的信息写入到文件中 |
| Int | primeNLT() | 根据file文件中的记录，在c到n内取最小的苏舒 |
| Size\_t | Hashcode() | 对字符串对象进行计算，算出它的哈希码 |
| void | DisplayWord(list l) | 显示所有单词。用到wordlist类中print()函数 |
| void | AddWord(list l) | 添加单词。用到wordlist类中的insert()函数和insert\_file(string str)函数，其中前者是从键盘键入数据，后者是从文件中读取数据 |
| void | DeleteWord(list l) | 删除单词。用到wordlist类中的delete\_word()函数。 |

## 核心算法

该程序所设计的散列表核心算法较多，在此一一展开。

第一步构建位图(Bitmap)，位图是一种特殊的序列结构，可以动态的表示一组整数构成的集合，其长度无限、且每个元素的类型都是bool型。这里使用了一段动态申请的连续空间M[]，并依次将其中的各比特位与位图集合中的各整数一一对应：若集合中包含整数k，则该段空间中的第k个比特位为1；否则该比特位为0。在实现上述一一对应关系时，这里借助了高效的整数移位和位运算。鉴于每个字节通常包含8个比特，故通过移位运算：k >> 3即可确定对应的比特位所属字节的秩；通过逻辑位与运算：k & 0x07即可确定该比特位在此字节中的位置；通过移位操作：0x80 >> (k & 0x07)即可得到该比特位在此字节中对应的数值掩码（mask）。如此，只需在局部将此字节与上述掩码做逻辑或运算，即可将整数k所对应的比特位设置为1；将此字节与上述掩码做逻辑与运算，即可测试该比特位的状态；将此字节与上述掩码的反码做逻辑位与运算，即可将该比特位设置为0。这里还提供了一个dump()接口，可以将位图整体导出至指定的文件，以便对此后的新位图批量初始化。例如在后面9.3节实现高效的散列表结构时，经常需要快速地找出不小于某一整数的最小素数。为此，可以借助Eratosthenes算法，事先以位图形式筛选出足够多个候选素数，并通过dump()函数将此集合保存至文件。此后在使用散列表时，可一次性地读入该文件，即可按照需要反复地快速确定合适的素数。

第二步是在位图的基础上构建散列表。在构造散列表时，需要一个初始的素数，为了加速素数的选取，通过Eratosthenes() 和primeNLT()函数事先计算出不超过1048576的所有素数，并存放于prime-1048576-bitmap.txt文件中备查。于是在创建散列表或者重散列的时候，对于在此范围内的任意给定长度下限c都能迅速找到不小于c的最小素数M作为散列表的长度，并以此为据申请相应数量的空桶，同时创建一个同样长度的位图结构，作为懒惰删除标志表。懒惰删除，即不在要删除的位置删除这个元素，而是给这个元素一个标记，当访问的时候读到这个标记的时候，便会跳过这个元素，采用这个办法的原因主要是对于散列表来说，删除的操作有些繁琐，会导致连锁反应。接着进行查找和插入操作，probe4Hit()函数用来确认散列表是否包含目标词条，首先采用除余法确定首个试探的桶单元，然后按照线性探查法沿查找链逐个试探；插入操作通过调用probe4Free()，采用除余法确定起始桶单元之后，沿查找链依次检查，直到发现空桶，在此实现插入。再然后是重散列rehash()。当装填因子大于0.5时就进行重散列操作，将原散列表中的所有词条整天迁移至容量至少加倍的新散列表中。

用户在使用该程序的时候，会先行展示一个菜单，提供用户使用的接口。在开始的时候系统会自动初始化，在内部构建一个散列表，并且通过读文件的方式将用户名和电话号码读入散列表。文件中的信息采用了特殊化处理的手段，用在每一个电话号码前加一个#作为标识符号，在文件结尾加上%作为标识符号，这种操作的原因是，如果将标识符#设置在一条信息后，那么在一个用户的信息读取结束后，采用getchar()函数读取标识符，虽然会正确读取到#，但是还会有一个换行符\n没有读取到，这样会造成缓冲区溢出，程序会非正常终止。在插入的时候通过push()函数调用push\_in\_file()，通过seekp()函数，将输入定位到文件末尾的%前面一个字符，然后读取一个字符，检查是否是换行符，如果是换行符那么继续读取一个字符，如果是#就跳出循环，退出插入操作，如果不是#则通过ungetc()将这个字符退还给文件。

# （三）用户手册

程序运行显示一级菜单，提示选择退出管理程序/增加用户信息/查找用户信息 0/1/2。

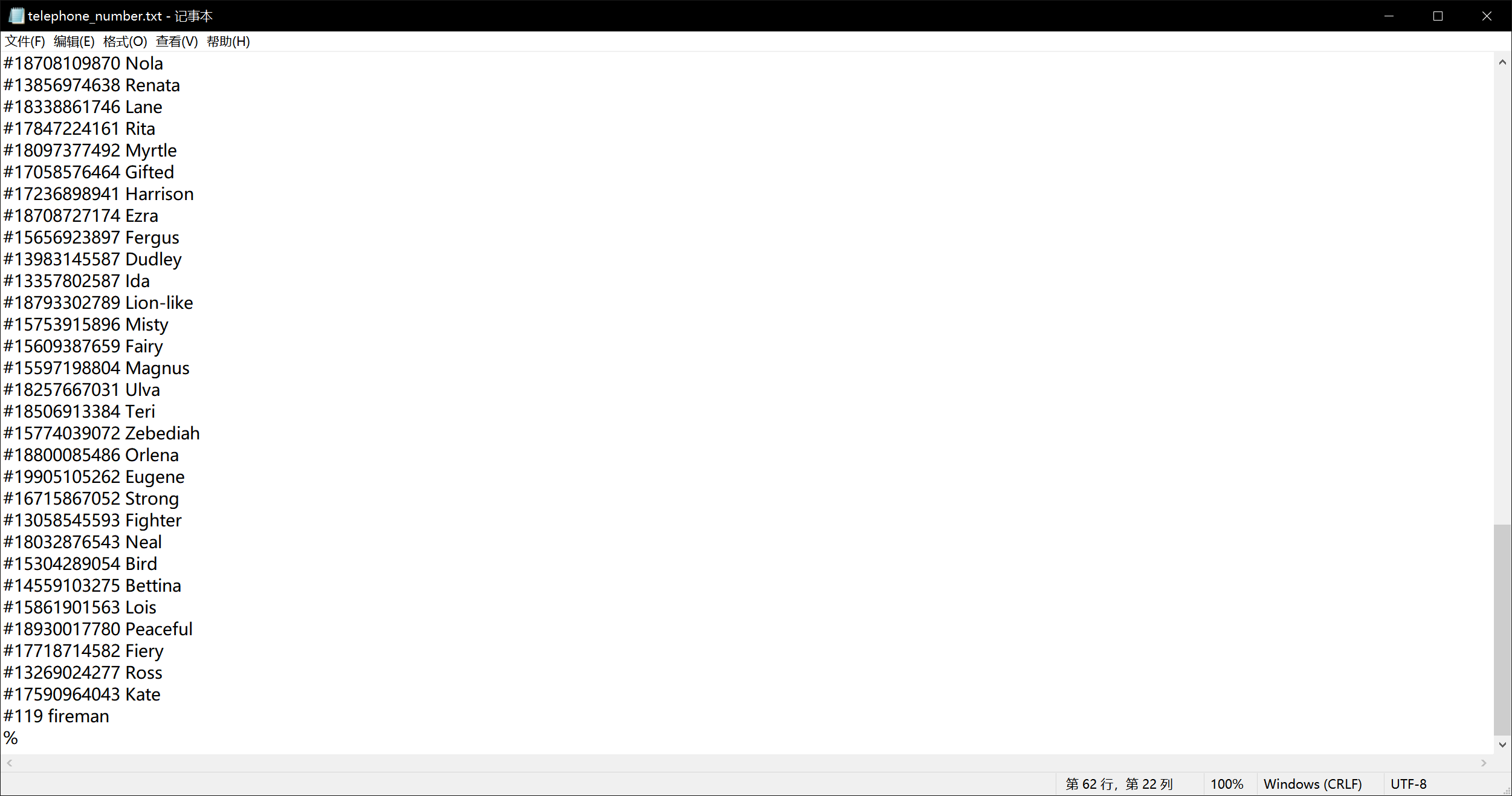
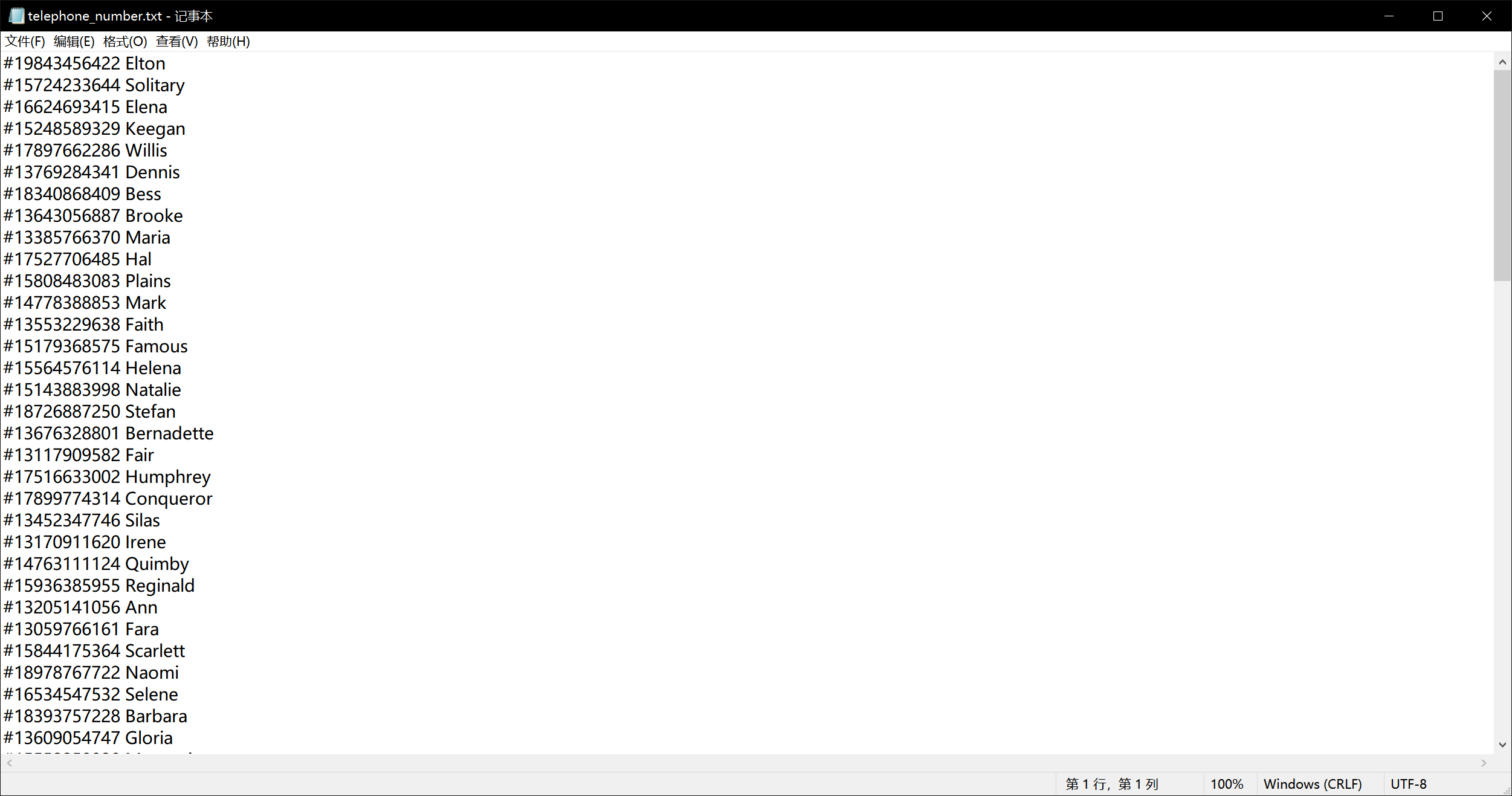
选择增加用户信息进入二级菜单，提示依次输入电话号码和用户名(若输入完毕输入#)并按回车键提交，提交后会插入成功提示。随后按任意键返回主菜单。

选择查找用户信息进入二级菜单，输入待查询的用户名，即可展示该用户的电话号码。

选择退出程序，通过exit(0)退出程序。

# （四）调试及测试

## 1. 测试数据：



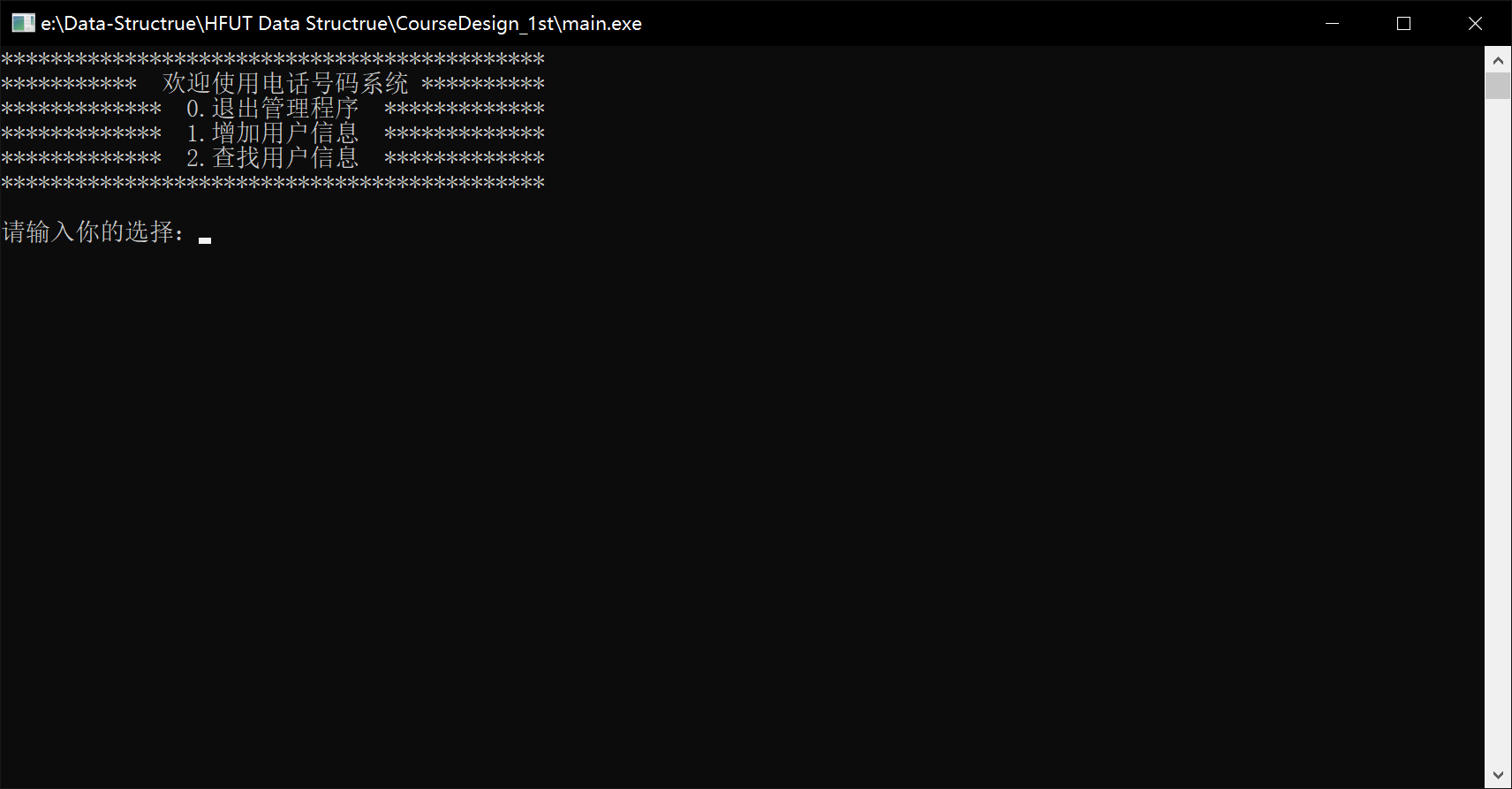
**用户信息库**

添加用户：110 police

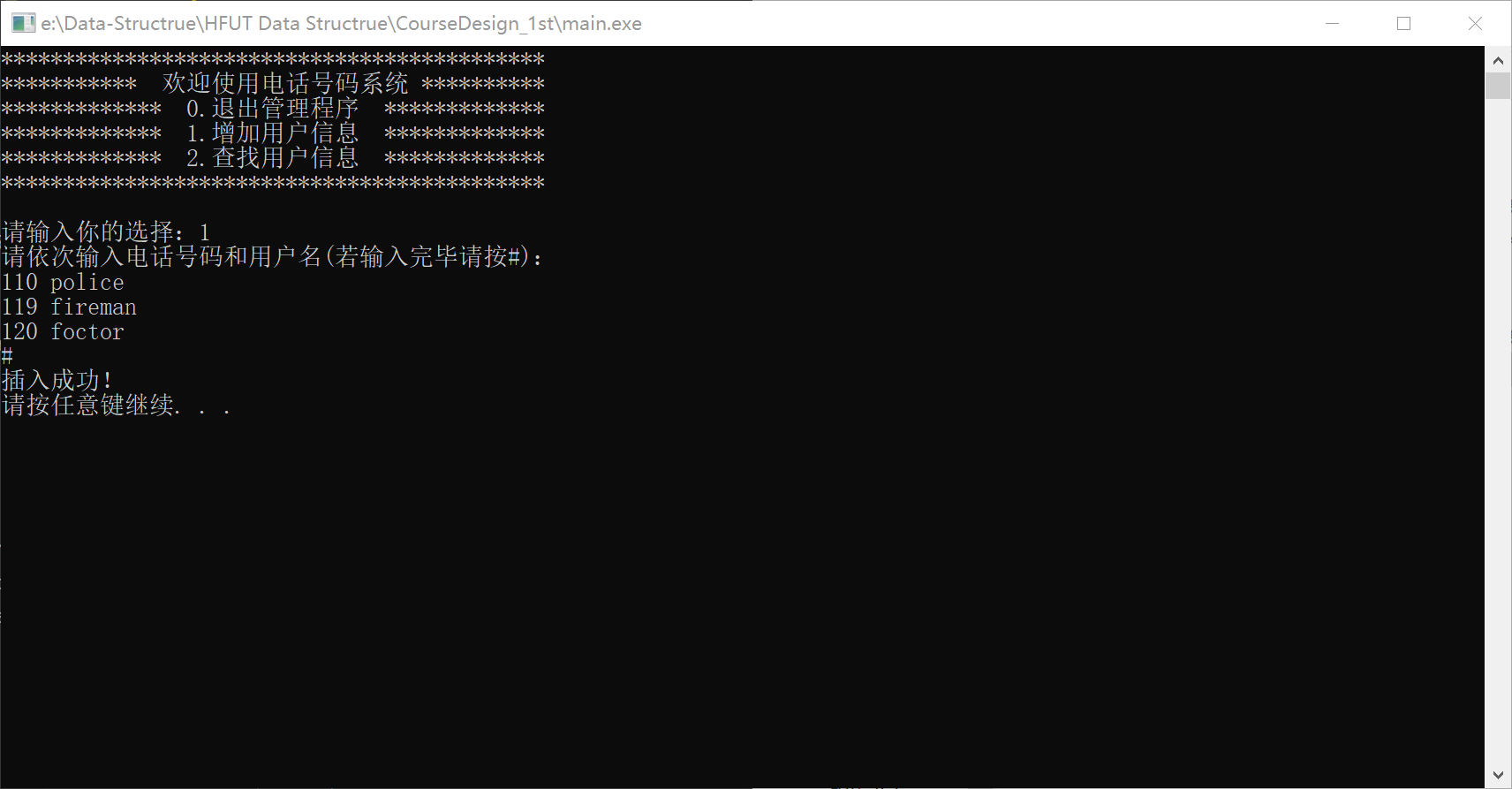
119 fireman

120 doctor

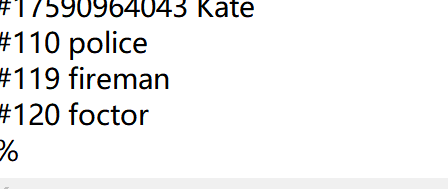
## 2. 测试结果：

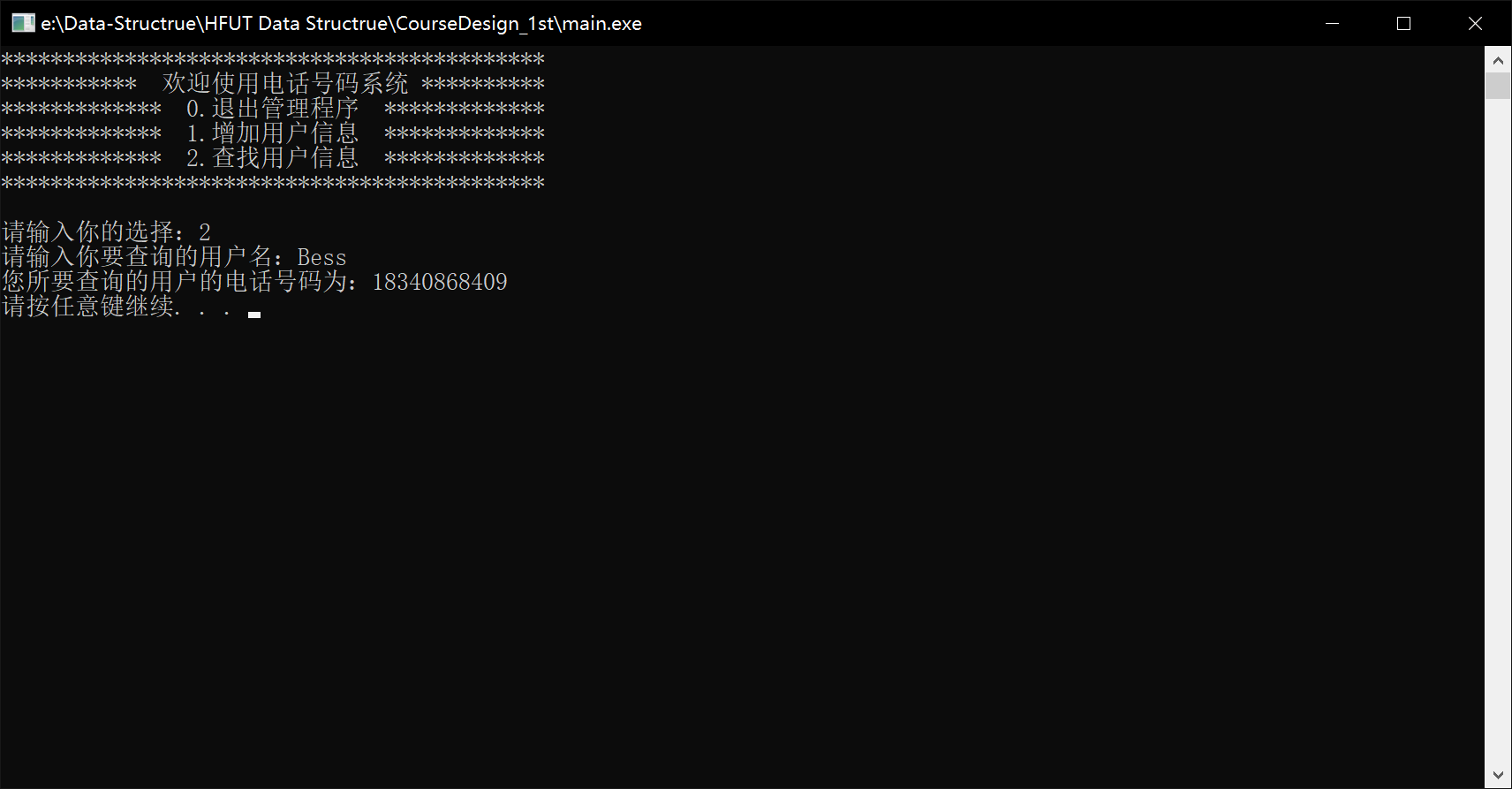
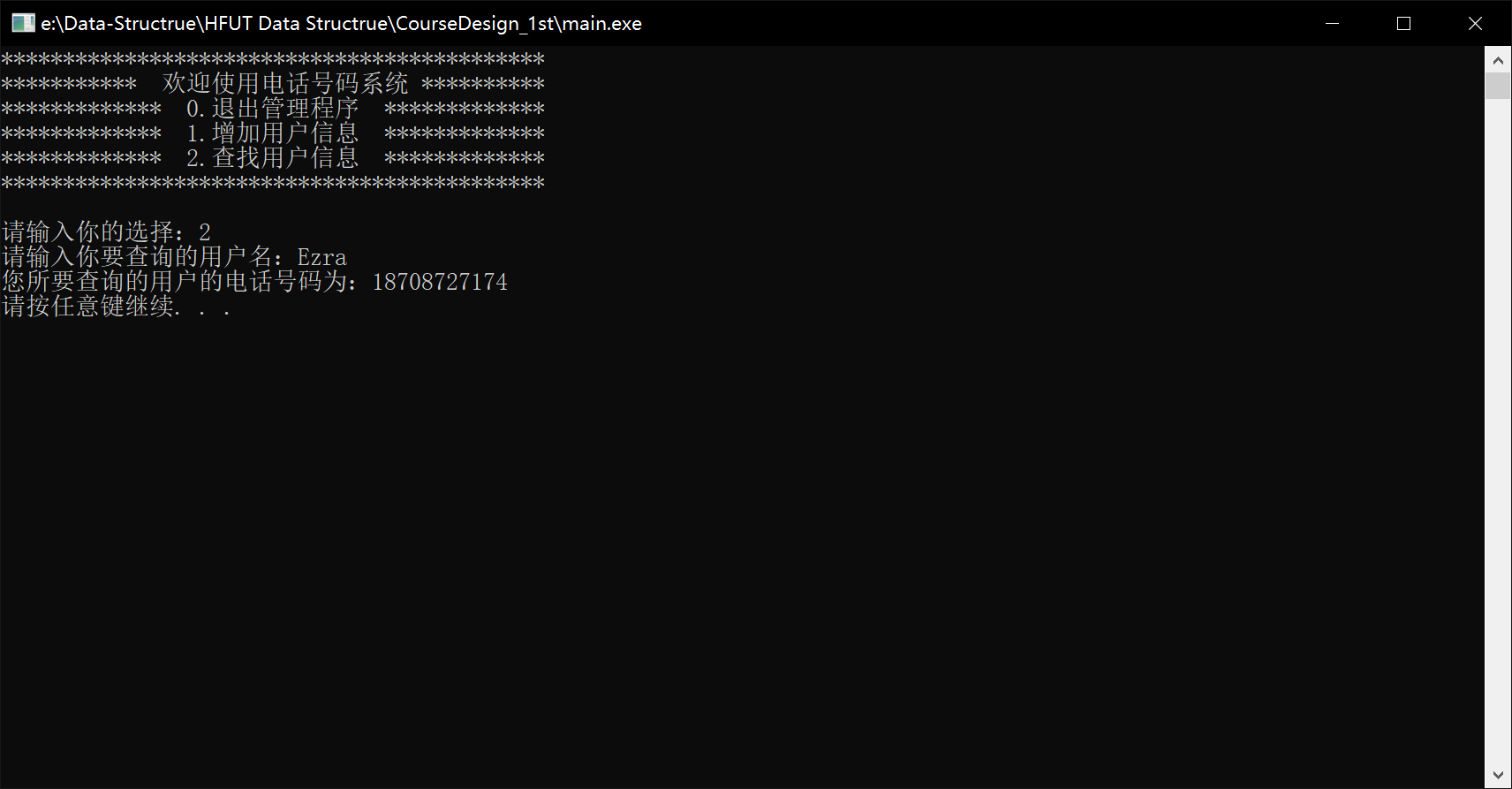
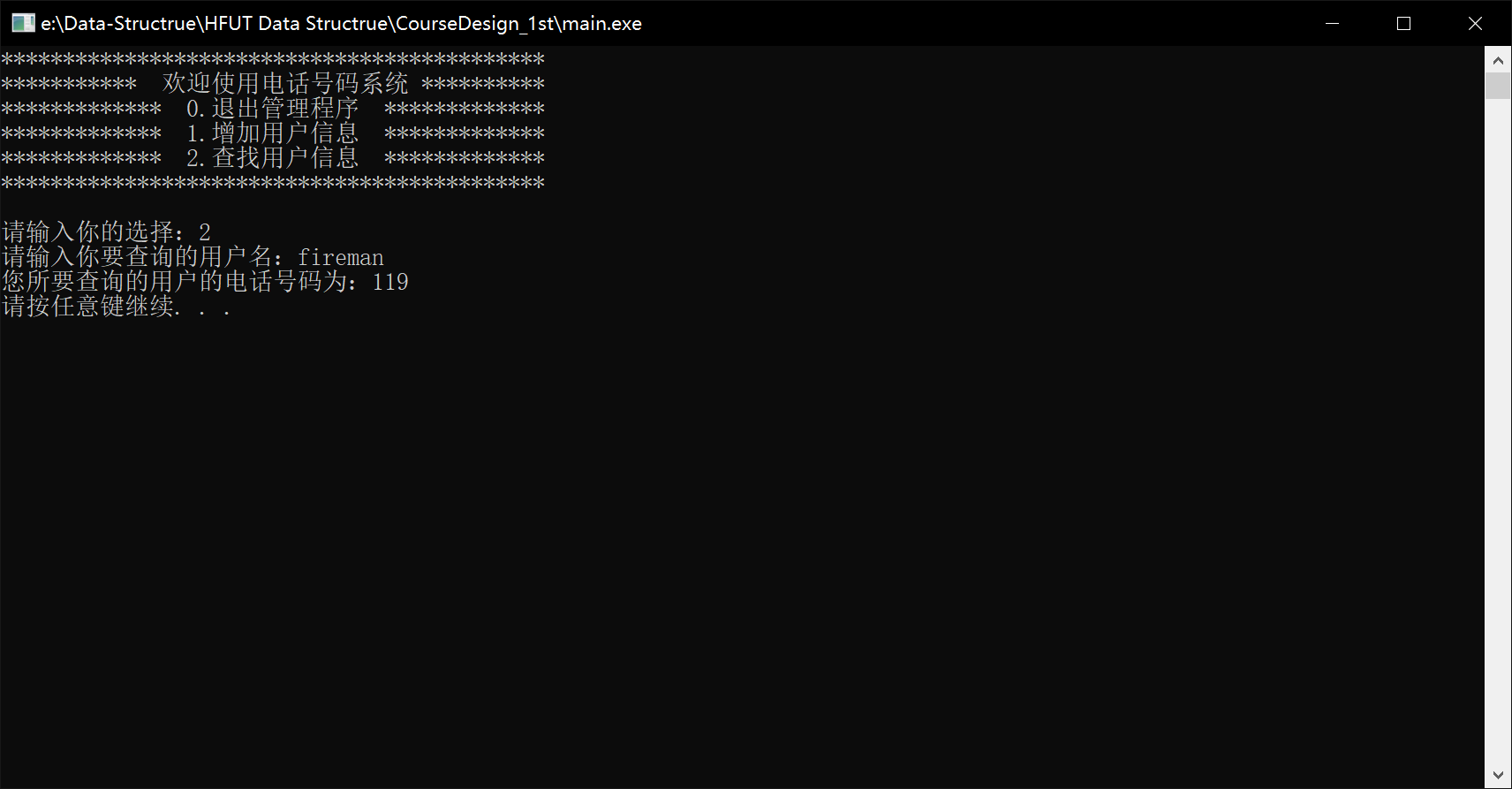
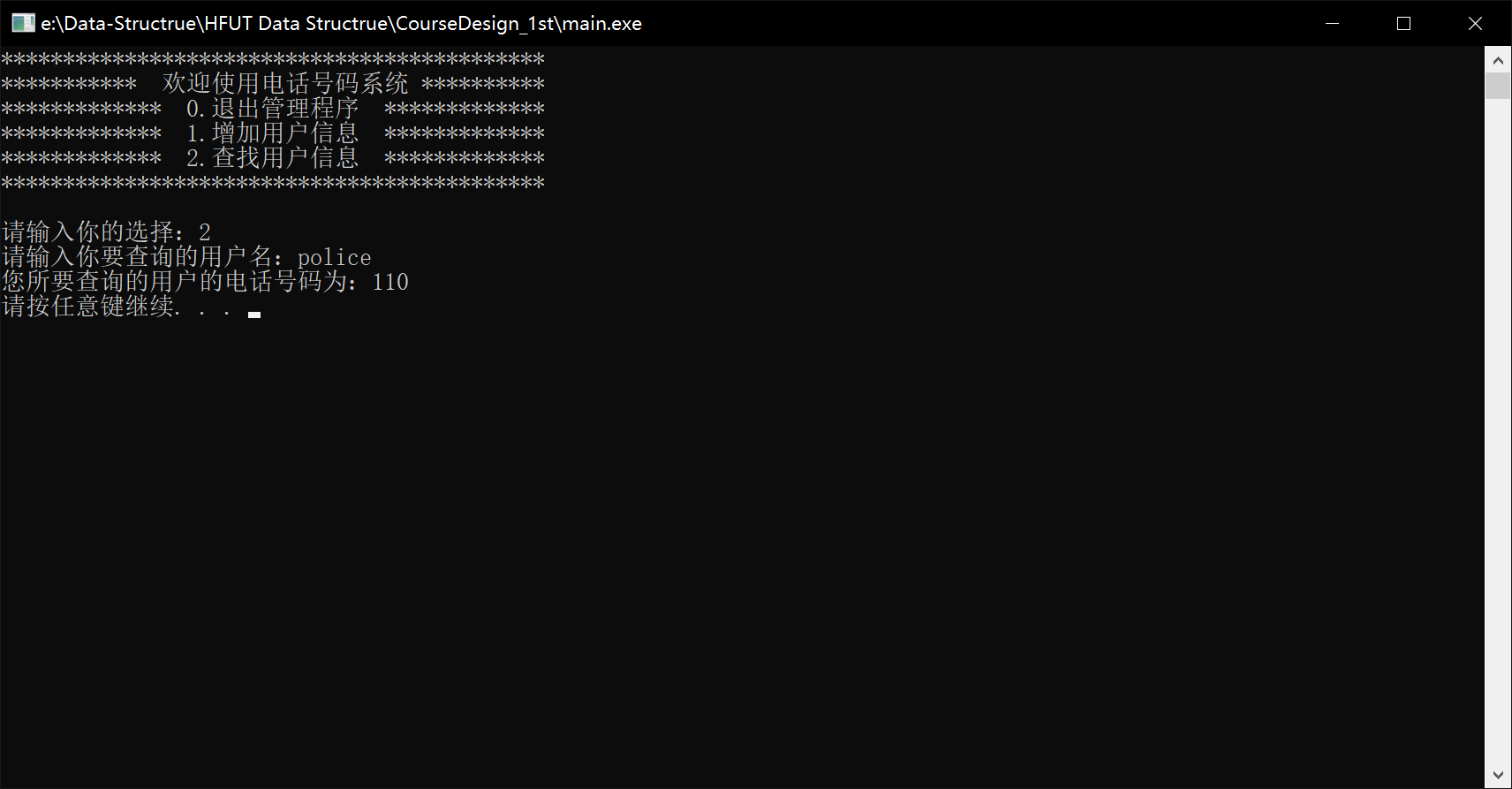


**主菜单**



**添加用户**





查找用户

## 3. 进一步改进：

（1）散列表即为哈希表，C++11发布的STL库中的unordered\_map就是基于哈希表实现的，有一点可惜的地方就是这种数据结构不支持重复的关键码，所以限制了它的一些应用面。

（2）没有在程序中展示出查找链长度，查找次数这些过程量，不阅读源码的话就不知道该程序使用散列表实现的。

（3）在对文件进行读写操作的时候代码实现的不够优美，总觉得有更好的解决方案，但是我自己的方法较为复杂，需要getchar()两次，然后通过判断来决定是否将读取的字符退还给文件。

# (五) 感想

从寒假到这学期一开始就是跟着清华大学邓俊辉老师学习的数据结构，在很长一段时间里都被他晦涩难懂的描述和代码风格所困扰，很多时候都是一知半解，但是好在全部跟着敲了一遍，把书上的数据结构都实现了一遍、并且自己优化了一下，其实也不能说是优化，只能说是近人可读一些。他的代码在我看来有一部分是把简单问题复杂化，但是这也是出于工程的严谨与全面性。

这道题如果要做出一个成品并不难，构建一个散列表，只需要一个比较简单的散列函数。而我学习了邓俊辉老师的构建方式，虽然复杂繁琐，但是收获颇多。一是素数的选择，不必在初始化时指定，而是自主实现一个筛选素数的算法，这里采用了埃氏筛法，效率更高；二是海量数据处理方面的知识，在传统处理方法中一个整型数字int占用4个字节，如果数据量较大，则空间开销很大，而这里采用位图(Bitmap)结构，用一个比特位存储一个数，将下标是合数的元素标记出来。这样既节省空间，又可以加快素数的选取。三是懒惰删除法，在一些数据结构和程序之中，真正意义上的删除可能工程量较大，因为这涉及到局部结构和整体结构的调整，一但要执行删除操作，就可能要进行全局调整；懒惰删除法的意思即为将待删除元素进行标识，在程序实际运行时，先行检测标识，如果检测到标识，那么就直接跳过该元素，实现程序意义上的删除。最后就是散列函数的选择，如果单纯使用除余法，那么会导致空间利用率不高，从而出现大量聚集的现象，这一方面的知识很难靠自己去参悟出来，只能学习并借鉴前人的经验，邓俊辉老师给出的针对字符串的散列函数十分优雅，首先对每个字符进行位运算，然后通过迭代，实现整个字符串的散列计算，使得关键码能够更加均匀地分布。

我觉得遗憾的点就是不能对汉字用户名进行操作，在上一个学期的C++程序设计基础的课程设计的时候，我就选择了一道关于字符串的题目，那个程序我最后实现出来也是无法对中文字符进行操作，最后和老师交流的时候才知道中文字符算作两个字符，很多针对英文字符串的算法在对中文字符进行操作的时候需要在逻辑层面进行修改。希望以后能涉猎一些对于这方面的学习，从而从容地解决这些问题。

# 附录：

1. 头文件Bitmap.h：

#pragma once

#include <cstdio>

#include <cstdlib>

#include <iostream>

#include <memory.h>

using namespace std;

class Bitmap { //位图Bitmap类

private:

char\* M;

int N; //比特图所存放的空间M[]，容量为N\*sizeof(char)\*8比特

protected:

void init(int n)

{

M = new char[N = (n + 7) / 8];

memset(M, 0, N);

}

public:

Bitmap(int n = 8) { init(n); } //按指定或默认规模创建比特图（为测试暂时选用较小的默认值）

Bitmap(char\* file, int n = 8) //按指定或默认规模，从指定文件中读取比特图

{

init(n);

FILE\* fp = fopen(file, "r");

fread(M, sizeof(char), N, fp);

fclose(fp);

}

~Bitmap()

{

delete[] M;

M = NULL;

} //析构时释放比特图空间

void set(int k)

{

expand(k);

M[k >> 3] |= (1 << (k % 8));

}

void clear(int k)

{

expand(k);

M[k >> 3] &= ~(1 << (k % 8));

}

bool test(int k)

{

expand(k);

return M[k >> 3] & (1 << (k % 8));

}

void dump(char\* file) //将位图整体导出至指定的文件，以便对此后的新位图批量初始化

{

FILE\* fp = fopen(file, "w");

fwrite(M, sizeof(char), N, fp);

fclose(fp);

}

char\* bits2string(int n)

{ //将前n位转换为字符串——

expand(n - 1); //此时可能被访问的最高位为bitmap[n - 1]

char\* s = new char[n + 1];

s[n] = '\0'; //字符串所占空间，由上层调用者负责释放

for (int i = 0; i < n; i++)

s[i] = test(i) ? '1' : '0';

return s; //返回字符串位置

}

void expand(int k)

{ //若被访问的Bitmap[k]已出界，则需扩容

if (k < 8 \* N)

return; //仍在界内，无需扩容

int oldN = N;

char\* oldM = M;

init(2 \* k); //与向量类似，加倍策略

memcpy\_s(M, N, oldM, oldN);

delete[] oldM; //原数据转移至新空间

}

};

void Eratosthenes(int n, char\* file)

{

Bitmap B(n);

B.set(0);

B.set(1); //0和1都不是素数

for (int i = 2; i \* i < n; i++) //逐个地

if (!B.test(i)) //确认下一个素数i

for (int j = i \* i; j < n; j += i) //并将一系列能被i整除的

B.set(j); //把合数标记了

B.dump(file); //将筛选标记统一存入指定文件，以便日后直接导入

}

头文件Dictionary.h：

#pragma once

template<typename K,typename V>

struct Dictionary{

virtual int size() const=0;

virtual bool put(K,V)=0;

virtual V\* get(K k)=0;

virtual bool remove (K k)=0;

};

头文件hashtable.h:

#pragma once

#include "Bitmap.h"

#include "Dictionary.h"

#include <cstring>

using namespace std;

template <typename K, typename V>

struct Entry

{

K key;

V value;

Entry(K k = K(), V v = V())

: key(k), value(v){};

Entry(Entry<K, V> const &e)

: key(e.key), value(e.value){};

bool operator<(Entry<K, V> const &e) { return key < e.key; }

bool operator>(Entry<K, V> const &e) { return key > e.key; }

bool operator==(Entry<K, V> const &e) { return key == e.key; }

bool operator!=(Entry<K, V> const &e) { return key != e.key; }

};

template <typename K, typename V>

class Hashtable : public Dictionary<K, V>

{

private:

Entry<K, V> \*\*ht; //桶数组，存放词条数组

int M; //桶数组容量

int N; //词条数量

Bitmap \*lazyRemoval; //懒惰删除标记

#define lazilyRemoved(x) (lazyRemoval->test(x))

#define markAsRemoved(x) (lazyRemoval->set(x))

protected:

int probe4Hit(const K &k); //沿关键码k对应的试探链，找到词条匹配的桶

int probe4Free(const K &k); //沿关键码k对应的试探链，找到首个可用空桶

void rehash(); //重散列算法：扩充桶数组，保证装填因子在警戒线以下

public:

Hashtable(int c = 2); //创建一个容量不小于c的散列表

~Hashtable();

int size() const { return N; }

bool put(K, V);

V \*get(K k);

bool remove(K k);

};

//根据file文件中的记录，在[c,n)内取最小的素数

int primeNLT(int c, int n, char \*file)

{

Eratosthenes(n, file);

Bitmap B(file, n);

while (c < n)

if (B.test(c))

c++;

else

return c; //返回首个发现的素数

return c;

}

//字符

static size\_t hashCode(char c)

{

return (size\_t)c;

}

//整数以及long long int

static size\_t hashCode(int k)

{

return (size\_t)k;

}

static size\_t hashCode(long long i)

{

return (size\_t)((i >> 32) + (int)i);

}

//字符串

static size\_t hashCode(string s)

{

int h = 0;

for (size\_t n = s.size(), i = 0; i < n; ++i)

{

h = (h << 5) | ((h >> 27));

h += (int)s[i];

}

return (size\_t)h;

}

template <typename K, typename V>

Hashtable<K, V>::Hashtable(int c)

{

string s="E:/Data-Structrue/HFUT Data Structrue/CourseDesign/prime-1048576-bitmap.txt";

const char\* x=s.c\_str();

char\* t=const\_cast<char\*>(x);

M = primeNLT(c, 1048576,t );

N = 0;

//开辟桶数组

ht = new Entry<K, V> \*[M];

memset(ht, 0, sizeof(Entry<K, V> \*) \* M);

lazyRemoval = new Bitmap(M);

}

template <typename K, typename V>

Hashtable<K, V>::~Hashtable()

{

for (int i = 0; i < M; i++)

if (ht[i])

delete ht[i];

delete ht;

delete lazyRemoval;

}

//词条的查找算法

template <typename K, typename V>

V \*Hashtable<K, V>::get(K k)

{

int r = probe4Hit(k);

//禁止词条的key值雷同

return ht[r] ? &(ht[r]->value) : NULL;

}

//沿关键码k对应的查找链，找到与之匹配的桶

template <typename K, typename V>

int Hashtable<K, V>::probe4Hit(const K &k)

{

int r = hashCode(k) % M; //从起始桶触发

while ((ht[r] && (k != ht[r]->key)) || (!ht[r] && lazilyRemoved(r)))

r = (r + 1) % M; //沿查找链线性试探：跳过所有冲突的桶以及带懒惰删除标记的桶

return r;

}

template <typename K, typename V>

bool Hashtable<K, V>::remove(K k)

{

int r = probe4Hit(k);

if (!ht[r])

return false;

delete ht[r];

ht[r] = NULL;

markAsRemoved(r);

N--;

return true;

}

template <typename K, typename V>

int Hashtable<K, V>::probe4Free(const K &k)

{

int r = hashCode(k) % M;

while (ht[r])

r = (r + 1) % M;

return r;

}

template <typename K, typename V>

bool Hashtable<K, V>::put(K k, V v)

{

if (ht[probe4Hit(k)])

return false;

int r = probe4Free(k);

ht[r] = new Entry<K, V>(k, v);

++N;

if (N \* 2 > M)

rehash();

return true;

}

template <typename K, typename V>

void Hashtable<K, V>::rehash()

{

int old\_capacity = M;

Entry<K, V> \*\*old\_ht = ht;

string s= "E:/Data-Structrue/HFUT Data Structrue/CourseDesign/prime-1048576-bitmap.txt";

const char\* c=s.c\_str();

char\* t=const\_cast<char\*>(c);

M = primeNLT(2 \* M, 1048576, t);

N = 0;

ht = new Entry<K, V> \*[M];

memset(ht, 0, sizeof(Entry<K, V> \*) \* M);

delete lazyRemoval;

lazyRemoval = new Bitmap(M);

for (int i = 0; i < old\_capacity; ++i)

{

if (old\_ht[i])

put(old\_ht[i]->key, old\_ht[i]->value);

}

delete old\_ht;

}

头文件TelephoneNumber.h:

#pragma once

#include <fstream>

using namespace std;

#include "hashtable.h"

class TelephoneNumber {

public:

TelephoneNumber();

void showmenu();

void exitsystem();

void push();

void find();

private:

Hashtable<string, long long> h;

};

TelephoneNumber::TelephoneNumber()

{

fstream txtfile("E:/Data-Structrue/HFUT Data Structrue/CourseDesign\_1st/telephone\_number.txt");

string id;

long long number;

char flag;

txtfile >> flag;

do {

txtfile >> number >> id;

h.put(id, number);

txtfile >> flag;

} while (flag == '#');

txtfile.clear();

txtfile.close();

}

void TelephoneNumber::showmenu()

{

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\* 欢迎使用电话号码系统 \*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\* 0.退出管理程序 \*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\* 1.增加用户信息 \*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\* 2.查找用户信息 \*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << endl;

}

void TelephoneNumber::exitsystem()

{

cout << "确认退出";

system("pause");

exit(0);

}

void push\_in\_file(const string& id, long long number)

{

fstream txtfile("E:/Data-Structrue/HFUT Data Structrue/CourseDesign\_1st/telephone\_number.txt");

txtfile.seekp(-1, ios::end);

txtfile << '#' << number << ' ' << id << endl

<< '%';

txtfile.close();

}

void TelephoneNumber::push()

{

string id;

long long number;

cout << "请依次输入电话号码和用户名(若输入完毕请按#)：" << endl;

do {

cin >> number >> id;

h.put(id, number);

push\_in\_file(id, number);

char c = getchar();

if (c == '\n') {

char b=getchar();

if (b == '#')

break;

else ungetc(b,stdin);

}

} while (true);

cin.clear();

cout << "插入成功！" << endl;

system("pause");

system("cls");

}

void TelephoneNumber::find()

{

cout << "请输入你要查询的用户名：";

string s;

cin >> s;

if (h.get(s))

cout << "您所要查询的用户的电话号码为：" << \*h.get(s) << endl;

else

cout << "您索要查找的用户不存在！" << endl;

system("pause");

system("cls");

}

源文件 main.cpp

/\*设计散列表实现电话号码查找系统。

1) 设每个记录有下列数据项：用户名、电话号码、地址；

2) 从键盘输入各记录，以用户名（汉语拼音形式）为关键字建立散列表；

3) 采用一定的方法解决冲突；

4) 查找并显示给定电话号码的记录； \*/

#include "TelephoneNumber.h"

#include <iostream>

#include <string>

using namespace std;

int main(void)

{

TelephoneNumber tn;

char choice = 0;

while (true) {

tn.showmenu();

cout << "请输入你的选择：";

cin >> choice;

switch (choice) {

case '0':

tn.exitsystem();

break;

case '1':

cin.clear();

tn.push();

break;

case '2':

cin.clear();

tn.find();

break;

default:

cout << "输入格式有误！请重新输入。" << endl;

cin.clear();

cin.sync();

system("pause");

system("cls");

continue;

}

}

system("pause");

return 0;

}

测试用例：

#19843456422 Elton

#15724233644 Solitary

#16624693415 Elena

#15248589329 Keegan

#17897662286 Willis

#13769284341 Dennis

#18340868409 Bess

#13643056887 Brooke

#13385766370 Maria

#17527706485 Hal

#15808483083 Plains

#14778388853 Mark

#13553229638 Faith

#15179368575 Famous

#15564576114 Helena

#15143883998 Natalie

#18726887250 Stefan

#13676328801 Bernadette

#13117909582 Fair

#17516633002 Humphrey

#17899774314 Conqueror

#13452347746 Silas

#13170911620 Irene

#14763111124 Quimby

#15936385955 Reginald

#13205141056 Ann

#13059766161 Fara

#15844175364 Scarlett

#18978767722 Naomi

#16534547532 Selene

#18393757228 Barbara

#13609054747 Gloria

#15552258920 Mountain

#13511951064 Emmett

#17100977455 Hadwin

#15507462611 Myrrh

#19805859086 Wayne

#13878456272 Timothea

#15189517515 Ryan

#17893204025 Halbert

#13466731285 Miles

#17680108249 Happy

#18354353662 Kane

#19103582224 Octavia

#18740628974 Meadow

#18007986232 Godfrey

#19824411986 Tristan

#15648668312 Eddie

#15590381428 Victorious

#13917364924 Abigail

#17013988857 Lawyer

#18922729180 Tara

#13416186820 Geneva

#15278147599 Homer

#13576827811 Thomas

#14553857621 Veleda

#13912245489 Faye

#18416431652 Jimmy

#18316566616 Trix

#18552572534 Queenly

#18502771500 Powerful

#19163237100 Princess

#13202861748 Rory

#15034729078 Alice

#19179618109 Ward

#17588438827 Beloved

#17899607970 Island

#17607992649 Samantha

#18883145612 Annette

#18708109870 Nola

#13856974638 Renata

#18338861746 Lane

#17847224161 Rita

#18097377492 Myrtle

#17058576464 Gifted

#17236898941 Harrison

#18708727174 Ezra

#15656923897 Fergus

#13983145587 Dudley

#13357802587 Ida

#18793302789 Lion-like

#15753915896 Misty

#15609387659 Fairy

#15597198804 Magnus

#18257667031 Ulva

#18506913384 Teri

#15774039072 Zebediah

#18800085486 Orlena

#19905105262 Eugene

#16715867052 Strong

#13058545593 Fighter

#18032876543 Neal

#15304289054 Bird

#14559103275 Bettina

#15861901563 Lois

#18930017780 Peaceful

#17718714582 Fiery

#13269024277 Ross

#17590964043 Kate

#110 police

#119 fireman

#120 foctor

%