1. 题目分析

说明程序设计的任务，强调的是程序要做什么，此外列出各成员分工

- 任务：针对某个集体中人名设计一个哈希表，使得平均查找长度不超过 R，并完成相应的建表和查找程序。假设人名为中国人姓名的汉语拼音形式。待填入哈希表的人名共有 30 个，取平均查找长度的上限为 2。哈希函数用除留余数法构造。用伪随机探测再散列法处理冲突。
- 成员分工：

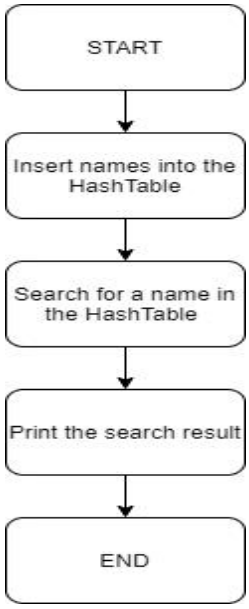
2. 数据结构设计

该程序主要用到的数据结构是**自定义的哈希表**，该数据结构包含以下成员：

1. **Person**：人名节点结构体，包含一个 name 字符数组用于存储人名。
2. **HashTable**：哈希表结构体，包含以下成员：

   data：一个 Person 类型的指针，用于存储哈希表中的数据。
   flags：一个整数数组，用于标记哈希表中的位置是否为空。
   当一个位置有人名时，对应位置的标记为 1；否则，标记为 0。
   size：哈希表的大小，表示哈希表可以容纳的最大元素数量。

主程序的流程如下：



3. 程序设计

实现概要设计中的数据类型，对主程序、模块及主要操作写出伪代码，画出函数的调用关系

各模块伪代码如下：

初始化哈希表：

```
// 初始化哈希表

void initHashTable(HashTable *table, int size)

{

    // 为哈希表分配空间

    // 为标记数组分配空间

    // 设置哈希表的大小

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

    {

    // 初始化人名为空

    // 初始化标记数组为 0

    }

}
```

除留余数法：

```
// 哈希函数：除留余数法

int hashFunction(char *name, int size)

{

    int sum = 0;
```

```
    for (int i = 0; i < strlen(name); i++)
    {
        // 将人名中每个字符的 ASCII 码相加
    }
    return sum % size;
}
```

插入人名到哈希表中：

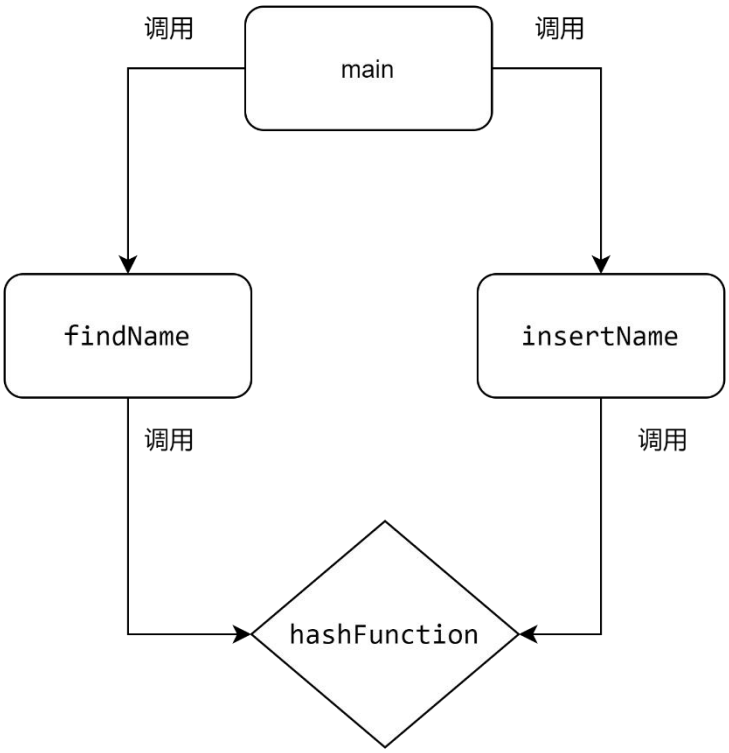
```
// 插入人名到哈希表中
void insertName(HashTable *table, char *name)
{
    // 计算人名在哈希表中的位置
    int i = 0;
    // 记录发生冲突的次数
    while // 人名发生冲突
    {
        // 发生冲突时，使用伪随机探测再散列法处理
    }
    // 将人名填入哈希表中
    // 标记数组中的位置为 1，表示该位置已经有人名
}
```

查找人名在哈希表中：

```
// 查找人名在哈希表中的位置
int findName(HashTable *table, char *name)
{
    // 计算人名在哈希表中的位置
    int i = 0;
    while // 人名发生冲突
```

```
    {
        if// 比较人名是否相同
        {
            return index; // 找到了人名，返回索引位置
        }
        // 发生冲突时，使用伪随机探测再散列法处理
    }
    return NOT_FOUND; // 未找到人名
}
```

各函数层次关系图：



4. 调试分析

时间复杂度：

1. 初始化哈希表时间复杂度 = O(size，其中 size 是哈希表的大小)
2. 插入人名到哈希表中的时间复杂度：平均情况下 O(1)，最坏情况下为 O(size)。

   Explain：这是因为哈希函数的散列操作通常具有 O(1) 的复杂度，但在发生冲突时可能需要执行一系列探测再散列操作，导致最坏情况下的复杂度为 O(size)。
3. 查找人名在哈希表中的位置的时间复杂度：平均情况下为 O(1)，最坏情况下为 O(size)。

   Explain： 与插入操作类似，平均情况下的复杂度是常数级别，但最坏情况下需要遍历整个哈希表才能找到人名或确定其不存在。

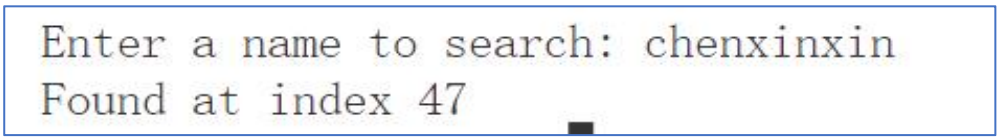
空间复杂度：O(size)

Explain：哈希表数据和标记数组：O(size)，其中 size 是哈希表的大小。哈希表数据占用的空间是 O(size)，标记数组占用的空间也是 O(size)。

5. 测试结果

Input：chenxinxin；
Output：Found at index 47；

说明：
输入姓名集里面的姓名，
将会输出该姓名在哈希表中的位置。



6. 用户使用说明

用户运行程序后系统会自动打印各个姓名插入到哈希表中冲突的次数，随后可以输入姓名集合里面的的一个姓名，系统会输出其在哈希表中的位置。

注：姓名集是在源文件中一直写进去的，而不是 txt 或者文档读入的，故若想修改姓名集合，则需要重写 main 函数里面的数组。

7. 选作内容

实现了（1）、（2）、（3）

（1）从教科书上介绍的几种哈希函数构造方法中选出适用者并设计几个不同的哈希函数，比较它们的地址冲突率(可以用更大的名字集合作试验)。

解：将书本上的平方去中法与上述的除留取余法进行地址冲突比较，发现平方去中法的冲突率非常高，达到了 31.22%。

除留余数法：
总冲突次数 12
平均冲突次数 0.4
冲突率 0.013333

```
Inserted chencaiyi with 0 conflicts.
Inserted chenxinxin with 0 conflicts.
Inserted fangsongjie with 0 conflicts.
Inserted huangjiawei with 0 conflicts.
Inserted huangjunlin with 0 conflicts.
Inserted leiyang with 0 conflicts.
Inserted luwei with 0 conflicts.
Inserted wangzhanqi with 0 conflicts.
Inserted wuweidong with 0 conflicts.
Inserted longjiajing with 1 conflicts.
Inserted huangweiqin with 0 conflicts.
Inserted chenshangming with 1 conflicts.
Inserted huangtao with 1 conflicts.
Inserted maixupeng with 0 conflicts.
Inserted chenying with 0 conflicts.
Inserted liujunhui with 1 conflicts.
Inserted chenqingdong with 0 conflicts.
Inserted guoziming with 1 conflicts.
Inserted chenyilong with 0 conflicts.
Inserted libaiyang with 1 conflicts.
Inserted liangdeguang with 0 conflicts.
Inserted zhangyuxing with 0 conflicts.
Inserted huangyibin with 0 conflicts.
Inserted liuwenlong with 0 conflicts.
```

```
Inserted liuwenlong with 0 conflicts.
Inserted zengzihao with 1 conflicts.
Inserted wangshulian with 1 conflicts.
Inserted zhuzixian with 2 conflicts.
Inserted lidingkun with 0 conflicts.
Inserted caofu with 0 conflicts.
Inserted yuemingju with 2 conflicts.
```

平方取中法：
总冲突次数 281
平均冲突次数 9.366
冲突率 0.312222

```
Inserted chencaiyi with 0 conflicts.
Inserted chenxinxin with 0 conflicts.
Inserted fangsongjie with 1 conflicts.
Inserted huangjiawei with 2 conflicts.
Inserted huangjunlin with 3 conflicts.
Inserted leiyang with 2 conflicts.
Inserted luwei with 3 conflicts.
Inserted wangzhanqi with 4 conflicts.
Inserted wuweidong with 4 conflicts.
Inserted longjiajing with 5 conflicts.
Inserted huangweiqin with 7 conflicts.
Inserted chenshangming with 8 conflicts
Inserted huangtao with 5 conflicts.
Inserted maixupeng with 7 conflicts.
Inserted chenying with 8 conflicts.
Inserted liujunhui with 9 conflicts.
Inserted chenqingdong with 9 conflicts.
Inserted guoziming with 12 conflicts.
Inserted chenyilong with 10 conflicts.
Inserted libaiyang with 13 conflicts.
Inserted liangdeguang with 13 conflicts
Inserted zhangyuxing with 16 conflicts.
Inserted huangyibin with 17 conflicts.
Inserted liuwenlong with 18 conflicts.
Inserted zengzihao with 14 conflicts.
Inserted wangshulian with 20 conflicts.
Inserted zhuzixian with 16 conflicts.
Inserted lidingkun with 17 conflicts.
Inserted caofu with 18 conflicts.
Inserted yuemingju with 20 conflicts.
```

（2）研究这 30 个人名的特点，努力找一个哈希函数，使得对于不同的拼音名一定不发生地址冲突。
解：根据以下名字集合，总结了三个特点，针对这三个特点实现了一个自定义哈希函数。

```
------------------------------------------------------------------
------
//名字集合
"chencaiyi", "chenxinxin", "fangsongjie", "huangjiawei", "huangjunlin", "leiyang",
"luwei", "wangzhanqi", "wuweidong", "longjiajing", "huangweiqin", "chenshangming",
"huangtao",
"maixupeng", "chenying", "liujunhui", "chenqingdong", "guoziming", "chenyilong",
"libaiyang", "liangdeguang",
"zhangyuxing", "huangyibin", "liuwenlong", "zengzihao", "wangshulian", "zhuzixian",
"lidingkun", "caofu", "yuemingju"
------------------------------------------------------------------
------
```

1. 大多数名字**遵循“姓+名”或“姓+中间名+名字”的模式。**
2. 有些名字有**重复的字符**，如 chencaiyi,chenxinxni,huangjunlin。
3. 名称的**长度各不相同**，从 5 到 12 个字符不等。

因此设计了一个自定义哈希函数

```
// 哈希函数：自定义哈希函数

int hashFunction(char *name, int size)

{

    int hash = 0;

    int nameLength = strlen(name);

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

    {

        hash = (hash * 31 + name[i]) % size;

    }

    return hash;

}
```

(3) 在哈希函数确定的前题下尝试各种不同处理冲突的方法，考查平均查找长度的变化和造好的哈希表中关键字的聚簇性。
解：在确定使用（2）所说的自定义哈希函数后，将采用单独链接法来处理冲突的情况。

单独链接法：在此方法中，哈希表中的每个插槽都包含一个链表或其他数据结构，用于存储散列到同一索引的多个元素。发生冲突时，新元素将添加到该索引处的链接列表中。要搜索元素，请在相应的索引处遍历链表，直到找到匹配项。

下面是改动最大的函数`findName`。

```
// 查找人名在哈希表中的位置

int findName(HashTable *table, char *name)

{

    int index = hashFunction(name, table->size); // 计算人名在哈希表中的位置

    // 遍历链表查找人名

    PersonNode *currentNode = table->data[index];

    while (currentNode != NULL)

    {

        if (strcmp(currentNode->name, name) == 0)

        {

            return index; // 找到了人名，返回索引位置

        }

        currentNode = currentNode->next;

    }

    return NOT_FOUND; // 未找到人名

}
```

8. 附录

附录说明：

一共有四个.cpp 文件，分别为`HashMap.cpp`、`HashMap(1).cpp`、`HashMap(2).cpp`、`HashMap(3).cpp `。

1. `HashMap.cpp `：使用求留取余法的哈希表。
2. `HashMap(1).cpp `：使用平方取中法的哈希表。
3. `HashMap(2).cpp `：使用自定义哈希方法来改善求留取余法的哈希表。
4. `HashMap(3).cpp `：使用单链接法来处理冲突情况来改善 HashMap(2).cpp 中的程序。

Word 直接复制粘贴代码有点难看，可以直接打开 Code 文件里面相应的.cpp 文件

HashMap.cpp 程序

```
/*

 * @Author: hiddenSharp429 z404878860@163.com

 * @Date: 2023-06-13 16:38:25

 * @LastEditors: hiddenSharp429 z404878860@163.com

 * @LastEditTime: 2023-06-14 17:28:58

 * @FilePath: \appe:\C OR C++\code\HashMap.cpp

 * @Description: 这是默认设置,请设置`customMade`, 打开koroFileHeader查看配置 进行

设置: https://github.com/OBKoro1/koro1FileHeader/wiki/%E9%85%8D%E7%BD%AE

 */

#include <stdio.h>

#include <stdlib.h>

#include <string.h>



#define HashTABLE_SIZE 61  // 哈希表的大小，选择一个较大的素数

#define MAX_NAME_LENGTH 20 // 人名的最大长度

#define NOT_FOUND -1       // 未找到的标志
```

```
typedef struct
{
    char name[MAX_NAME_LENGTH]; // 人名
} Person;

typedef struct
{
    Person *data; // 哈希表的数据
    int *flags;    // 标记哈希表中的位置是否为空
    int size;      // 哈希表的大小
} HashTable;

// 哈希函数：除留余数法
int hashFunction(char *name, int size)
{
    int sum = 0;
    for (int i = 0; i < strlen(name); i++)
    {
        sum += name[i]; // 将人名中每个字符的 ASCII 码相加
    }
    return sum % size;
```

```
}


// 初始化哈希表
void initHashTable(HashTable *table, int size)
{
    table->data = (Person *)malloc(sizeof(Person) * size); // 为哈希表分配空间
    table->flags = (int *)malloc(sizeof(int) * size);      // 为标记数组分配空间
    table->size = size;                                    // 设置哈希表的大小
    for (int i = 0; i < size; i++)
    {
        strcpy(table->data[i].name, ""); // 初始化人名为空
        table->flags[i] = 0;             // 初始化标记数组为 0
    }
}
// 插入人名到哈希表中
void insertName(HashTable *table, char *name)
{
    int index = hashFunction(name, table->size); // 计算人名在哈希表中的位置
    int i = 0;
    int conflicts = 0;                  // 记录发生冲突的次数
    while (table->flags[index] == 1) // 人名发生冲突
    {
```

```
        // 发生冲突时，使用伪随机探测再散列法处理
        i++;
        index = (index + i * i) % table->size;
        conflicts++;
    }
    strcpy(table->data[index].name, name); // 将人名填入哈希表中
    table->flags[index] = 1;                    // 标记数组中的位置为 1，表示该位置已经有人名

    printf("Inserted %s with %d conflicts.\n", name, conflicts);
}

// 查找人名在哈希表中的位置
int findName(HashTable *table, char *name)
{
    int index = hashFunction(name, table->size); // 计算人名在哈希表中的位置
    int i = 0;
    while (table->flags[index] != 0) // 人名发生冲突
    {
        if (strcmp(table->data[index].name, name) == 0) // 比较人名是否相同
        {
            return index; // 找到了人名，返回索引位置
        }
```

```
        i++;
        index = (index + i * i) % table->size; // 发生冲突时，使用伪随机探测再散列法处理
    }
    return NOT_FOUND; // 未找到人名
}

int main()
{
    HashTable table;
    initHashTable(&table, HashTABLE_SIZE); // 初始化哈希表

    // 待填入哈希表的人名
    char names[30][MAX_NAME_LENGTH] = {
        "chencaiyi", "chenxinxin", "fangsongjie", "huangjiawei", "huangjunlin", "leiyang",
        "luwei", "wangzhanqi", "wuweidong", "longjiajing", "huangweiqin",
"chenshangming", "huangtao",
        "maixupeng", "chenying", "liujunhui", "chenqingdong", "guoziming", "chenyilong",
"libaiyang", "liangdeguang",
        "zhangyuxing", "huangyibin", "liuwenlong", "zengzihao", "wangshulian",
"zhuzixian", "lidingkun", "caofu", "yuemingju"};
```

```
// 建立哈希表
for (int i = 0; i < 30; i++)
{
    insertName(&table, names[i]);
}

// 查找程序
char searchName[MAX_NAME_LENGTH];
printf("Enter a name to search: "); // 输入要查找的人名
scanf("%s", searchName);              // 读取人名

int index = findName(&table, searchName); // 查找人名在哈希表中的位置
if (index != NOT_FOUND)                   // 找到了人名
{
    printf("Found at index %d\n", index);
}
else // 未找到人名
{
    printf("Not found\n");
}

return 0;
```

```
}
```

HashMap(1).cpp 程序

```
/*
 * @Author: hiddenSharp429 z404878860@163.com
 * @Date: 2023-06-14 17:27:57
 * @LastEditors: hiddenSharp429 z404878860@163.com
 * @LastEditTime: 2023-06-14 17:28:14
 * @FilePath: \appe:\C OR C++\code\HashMap(1).cpp
 * @Description: 这是默认设置,请设置`customMade`, 打开koroFileHeader查看配置 进行
设置: https://github.com/OBKoro1/koro1FileHeader/wiki/%E9%85%8D%E7%BD%AE
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define HashTABLE_SIZE 61  // 哈希表的大小，选择一个较大的素数
#define MAX_NAME_LENGTH 20 // 人名的最大长度
#define NOT_FOUND -1       // 未找到的标志

typedef struct
{
    char name[MAX_NAME_LENGTH]; // 人名
} Person;
```

```
typedef struct
{
    Person *data; // 哈希表的数据
    int *flags;    // 标记哈希表中的位置是否为空
    int size;      // 哈希表的大小
} HashTable;

// 哈希函数：平方取中法
int hashFunction(char *name, int size)
{
    int nameLength = strlen(name);
    int square = nameLength * nameLength;
    int midDigits = (square / 100) % 100; // 取中间两位数

    return midDigits % size;
}

// 初始化哈希表
void initHashTable(HashTable *table, int size)
{
    table->data = (Person *)malloc(sizeof(Person) * size); // 为哈希表分配空间
```

```
    table->flags = (int *)malloc(sizeof(int) * size);       // 为标记数组分配空间
    table->size = size;                                     // 设置哈希表的大小
    for (int i = 0; i < size; i++)
    {
        strcpy(table->data[i].name, ""); // 初始化人名为空
        table->flags[i] = 0;                // 初始化标记数组为 0
    }
}

// 插入人名到哈希表中
void insertName(HashTable *table, char *name)
{
    int index = hashFunction(name, table->size); // 计算人名在哈希表中的位置
    int i = 0;
    int conflicts = 0; // 记录发生冲突的次数
    while (table->flags[index] == 1) // 人名发生冲突
    {
        // 发生冲突时，使用平方探测再散列法处理
        i++;
        index = (index + i * i) % table->size;
        conflicts++;
    }
```

```
    strcpy(table->data[index].name, name); // 将人名填入哈希表中
    table->flags[index] = 1;                // 标记数组中的位置为 1，表示该位置已经有人名

    printf("Inserted %s with %d conflicts.\n", name, conflicts);
}

// 查找人名在哈希表中的位置
int findName(HashTable *table, char *name)
{
    int index = hashFunction(name, table->size); // 计算人名在哈希表中的位置
    int i = 0;
    while (table->flags[index] != 0) // 人名发生冲突
    {
        if (strcmp(table->data[index].name, name) == 0) // 比较人名是否相同
        {
            return index; // 找到了人名，返回索引位置
        }
        i++;
        index = (index + i * i) % table->size; // 发生冲突时，使用平方探测再散列法处理
    }
    return NOT_FOUND; // 未找到人名
}
```

```
int main()
{
    HashTable table;
    initHashTable(&table, HashTABLE_SIZE);

    // 待填入哈希表的人名
    char names[30][MAX_NAME_LENGTH] = {
        "chencaiyi", "chenxinxin", "fangsongjie", "huangjiawei", "huangjunlin", "leiyang",
        "luwei", "wangzhanqi", "wuweidong", "longjiajing", "huangweiqin",
"chenshangming", "huangtao",
        "maixupeng", "chenying", "liujunhui", "chenqingdong", "guoziming", "chenyilong",
"libaiyang", "liangdeguang",
        "zhangyuxing", "huangyibin", "liuwenlong", "zengzihao", "wangshulian",
"zhuzixian", "lidingkun", "caofu", "yuemingju"};

    // 建立哈希表
    for (int i = 0; i < 30; i++)
    {
        insertName(&table, names[i]);
    }
```

```
    // 查找程序
    char searchName[MAX_NAME_LENGTH];
    printf("Enter a name to search: ");
    scanf("%s", searchName);

    int index = findName(&table, searchName);
    if (index != NOT_FOUND)
    {
        printf("Found at index %d\n", index);
    }
    else
    {
        printf("Not found\n");
    }

    return 0;
}
```

HashMap(2).cpp 程序

```
/*
 * @Author: hiddenSharp429 z404878860@163.com
 * @Date: 2023-06-14 17:29:44
```

```
 * @LastEditors: hiddenSharp429 z404878860@163.com
 * @LastEditTime: 2023-06-14 17:45:32
 * @FilePath: \appe:\C OR C++\code\HashMap(2).cpp
 * @Description: 这是默认设置,请设置`customMade`, 打开koroFileHeader查看配置 进行设置: https://github.com/OBKoro1/koro1FileHeader/wiki/%E9%85%8D%E7%BD%AE
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define HashTABLE_SIZE 61  // 哈希表的大小，选择一个较大的素数
#define MAX_NAME_LENGTH 20 // 人名的最大长度
#define NOT_FOUND -1       // 未找到的标志

typedef struct
{
    char name[MAX_NAME_LENGTH]; // 人名
} Person;

typedef struct
{
    Person *data; // 哈希表的数据
```

```
    int *flags;    // 标记哈希表中的位置是否为空
    int size;      // 哈希表的大小
} HashTable;


// 哈希函数：自定义哈希函数
int hashFunction(char *name, int size)
{
    int hash = 0;
    int nameLength = strlen(name);

    for (int i = 0; i < nameLength; i++)
    {
        hash = (hash * 31 + name[i]) % size;
    }

    return hash;
}


// 初始化哈希表
void initHashTable(HashTable *table, int size)
{
    table->data = (Person *)malloc(sizeof(Person) * size); // 为哈希表分配空间
```

```
    table->flags = (int *)malloc(sizeof(int) * size);       // 为标记数组分配空间
    table->size = size;                                     // 设置哈希表的大小
    for (int i = 0; i < size; i++)
    {
        strcpy(table->data[i].name, ""); // 初始化人名为空
        table->flags[i] = 0;                // 初始化标记数组为 0
    }
}


// 插入人名到哈希表中
void insertName(HashTable *table, char *name)
{
    int index = hashFunction(name, table->size); // 计算人名在哈希表中的位置
    int i = 0;
    int conflicts = 0; // 记录发生冲突的次数
    while (table->flags[index] == 1) // 人名发生冲突
    {
        // 发生冲突时，使用平方探测再散列法处理
        i++;
        index = (index + i * i) % table->size;
        conflicts++;
    }
```

```
    strcpy(table->data[index].name, name); // 将人名填入哈希表中
    table->flags[index] = 1;                // 标记数组中的位置为 1，表示该位置已经有人名

    printf("Inserted %s with %d conflicts.\n", name, conflicts);
}

// 查找人名在哈希表中的位置
int findName(HashTable *table, char *name)
{
    int index = hashFunction(name, table->size); // 计算人名在哈希表中的位置
    int i = 0;
    while (table->flags[index] != 0) // 人名发生冲突
    {
        if (strcmp(table->data[index].name, name) == 0) // 比较人名是否相同
        {
            return index; // 找到了人名，返回索引位置
        }
        i++;
        index = (index + i * i) % table->size; // 发生冲突时，使用平方探测再散列法处理
    }
    return NOT_FOUND; // 未找到人名
}
```

```
int main()
{
    HashTable table;
    initHashTable(&table, HashTABLE_SIZE);

    // 待填入哈希表的人名
    char names[30][MAX_NAME_LENGTH] = {
        "chencaiyi", "chenxinxin", "fangsongjie", "huangjiawei", "huangjunlin", "leiyang",
        "luwei", "wangzhanqi", "wuweidong", "longjiajing", "huangweiqin",
"chenshangming", "huangtao",
        "maixupeng", "chenying", "liujunhui", "chenqingdong", "guoziming", "chenyilong",
"libaiyang", "liangdeguang",
        "zhangyuxing", "huangyibin", "liuwenlong", "zengzihao", "wangshulian",
"zhuzixian", "lidingkun", "caofu", "yuemingju"};

    // 建立哈希表
    for (int i = 0; i < 30; i++)
    {
        insertName(&table, names[i]);
    }
```

```
    // 查找程序
    char searchName[MAX_NAME_LENGTH];
    printf("Enter a name to search: ");
    scanf("%s", searchName);

    int index = findName(&table, searchName);
    if (index != NOT_FOUND)
    {
        printf("Found at index %d\n", index);
    }
    else
    {
        printf("Not found\n");
    }

    return 0;
}
```

HashMap(3).cpp 程序

```
/*
 * @Author: hiddenSharp429 z404878860@163.com
 * @Date: 2023-06-14 17:28:46
```

```
 * @LastEditors: hiddenSharp429 z404878860@163.com
 * @LastEditTime: 2023-06-14 17:46:22
 * @FilePath: \appe:\C OR C++\code\HashMap(3).cpp
 * @Description: 这是默认设置,请设置`customMade`, 打开koroFileHeader查看配置 进行设置: https://github.com/OBKoro1/koro1FileHeader/wiki/%E9%85%8D%E7%BD%AE
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define HashTABLE_SIZE 61  // 哈希表的大小，选择一个较大的素数
#define MAX_NAME_LENGTH 20 // 人名的最大长度
#define NOT_FOUND -1       // 未找到的标志

typedef struct PersonNode
{
    char name[MAX_NAME_LENGTH]; // 人名
    struct PersonNode *next;    // 指向下一个节点的指针
} PersonNode;

typedef struct
{
```

```
    PersonNode **data; // 哈希表的数据
    int size;          // 哈希表的大小
} HashTable;


// 哈希函数：自定义哈希函数
int hashFunction(char *name, int size)
{
    int hash = 0;
    int nameLength = strlen(name);

    for (int i = 0; i < nameLength; i++)
    {
        hash = (hash * 31 + name[i]) % size;
    }

    return hash;
}


// 初始化哈希表
void initHashTable(HashTable *table, int size)
{
```

```
    table->data = (PersonNode **)malloc(sizeof(PersonNode *) * size); // 为哈希表分配
空间
    table->size = size;                                                // 设置哈希表的大小
    for (int i = 0; i < size; i++)
    {
        table->data[i] = NULL; // 初始化每个槽位为空
    }
}

// 插入人名到哈希表中
void insertName(HashTable *table, char *name)
{
    int index = hashFunction(name, table->size); // 计算人名在哈希表中的位置

    // 创建新节点
    PersonNode *newNode = (PersonNode *)malloc(sizeof(PersonNode));
    strcpy(newNode->name, name);
    newNode->next = NULL;

    if (table->data[index] == NULL)
    {
        // 槽位为空，直接插入新节点
```

```
        table->data[index] = newNode;
    }
    else
    {
        // 槽位非空，遍历链表找到尾节点并插入新节点
        PersonNode *currentNode = table->data[index];
        while (currentNode->next != NULL)
        {
            currentNode = currentNode->next;
        }
        currentNode->next = newNode;
    }

    printf("Inserted %s\n", name);
}

// 查找人名在哈希表中的位置
int findName(HashTable *table, char *name)
{
    int index = hashFunction(name, table->size); // 计算人名在哈希表中的位置

    // 遍历链表查找人名
```

```
    PersonNode *currentNode = table->data[index];
    while (currentNode != NULL)
    {
        if (strcmp(currentNode->name, name) == 0)
        {
            return index; // 找到了人名，返回索引位置
        }
        currentNode = currentNode->next;
    }

    return NOT_FOUND; // 未找到人名
}

int main()
{
    HashTable table;
    initHashTable(&table, HashTABLE_SIZE);

    // 待填入哈希表的人名
    char names[30][MAX_NAME_LENGTH] = {
        "chencaiyi", "chenxinxin", "fangsongjie", "huangjiawei", "huangjunlin", "leiyang",
```

```
        "luwei", "wangzhanqi", "wuweidong", "longjiajing", "huangweiqin",
"chenshangming", "huangtao",
        "maixupeng", "chenying", "liujunhui", "chenqingdong", "guoziming", "chenyilong",
"libaiyang", "liangdeguang",
        "zhangyuxing", "huangyibin", "liuwenlong", "zengzihao", "wangshulian",
"zhuzixian", "lidingkun", "caofu", "yuemingju"};


    // 建立哈希表
    for (int i = 0; i < 30; i++)
    {
        insertName(&table, names[i]);
    }


    // 查找程序
    char searchName[MAX_NAME_LENGTH];
    printf("Enter a name to search: ");
    scanf("%s", searchName);


    int index = findName(&table, searchName);
    if (index != NOT_FOUND)
    {
        printf("Found at index %d\n", index);
```

```
    }
    else
    {
        printf("Not found\n");
    }

    return 0;
}
```