南京信息工程大学 数据结构 I 实验(实习)报告

实验（实习）名称 **哈希表的创建与查找** 日期 2024.12. 得分_____ 指导教师_____

学院 计院 专业 计科

一、实验目的

1、理解并熟练掌握哈希表和哈希函数的概念；
2、掌握各种哈希函数和冲突解决办法；
3、熟练掌握哈希表的构造和查找。

二、实验内容与步骤

1、根据从键盘输入的数列 {23,5,17,12,26,31,13,4,6}，设计一个表长为 13 的哈希表，完成该哈希表的构造。哈希函数采用除留余数法：H(key)= key MOD 13，并用线性探测再散列的方法处理哈希地址冲突。功能包括：

（1）输出哈希表中的全部内容（包括哈希地址和相应存储的元素值），如果某个地址存储的元素为空，可以用空格表示。

（2）针对从键盘输入的给定值进行查询，判断该值在哈希表中是否存在？如果存在，输出相应的哈希地址；否则，输出不存在。

（3）输出该哈希表的平均查找长度 ASL（假定每个元素被查找的概率都是相等的）。

提示：本次实验需要使用的存储结构和操作模块主要有

（1）结构体定义：

```
typedef struct Elem{
    int key; //元素的值
    int compareTimes; //查找到该元素需要比较的次数
    int flag; //用来标识该位置是否已经存有数据
};
typedef struct HashList{
    Elem *base;
    int length;
};
```

（2）建立哈希表：void CreateHashList(HashList &HL)；

（3）显示哈希表：void PrintHashList(HashList HL)；

（4）查找：void IndexHashList(HashList HL)；

（5）计算平均查找长度 ASL：void CalcuASL(HashList H)；

（6）主函数：void main()

报告要求：

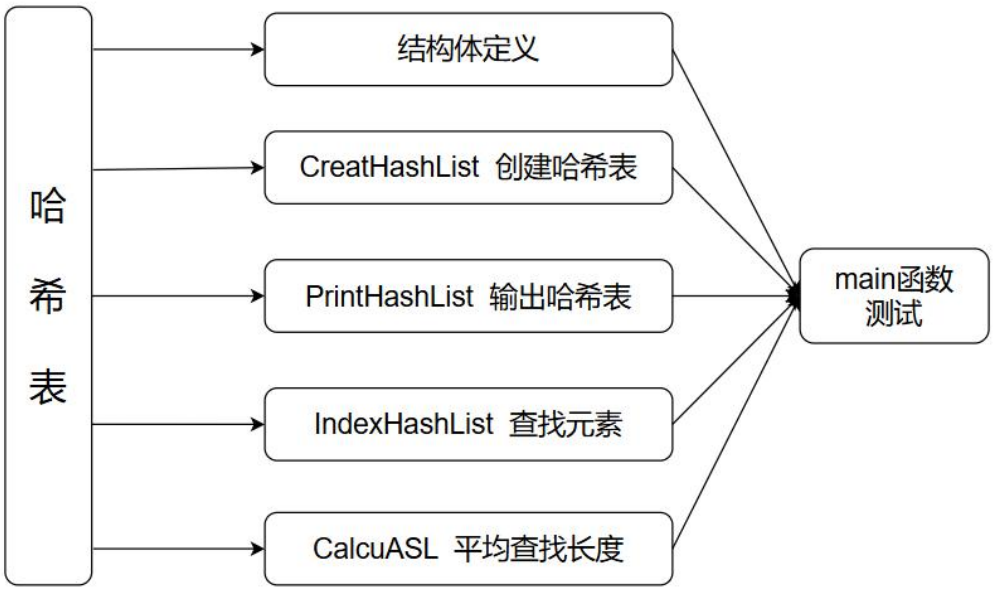
（1）回答使用哈希表的目的？

用哈希表来存储和快速检索整数集合。通过使用除留余数法作为哈希函数，同时采用线性探测再散列法解决哈希冲突。哈希表的每个元素包含一个整数键、一个记录比较次数的字段和一个标志位，以指示该位置是否被占用。这种数据结构的设计允许快速访问和检索，同时动态内存分配使得哈希表可以根据需要调整大小。此外，还计算了平均查找长度（ASL），以评估哈希表的性能。

哈希表因其高效的数据访问特性，在实际应用中被广泛用于数据库、缓存、去重和快速

访问等多种场景。它提供了更快的查找、插入和删除操作，尤其是在处理大量数据时，哈希表的平均时间复杂度接近 O(1)，使得这些操作更加高效。此外，哈希表能够动态调整大小以适应数据量的增减，优化空间利用率，并且可以处理数据冲突，使得在存储和检索具有相同键值的数据时更加灵活。

（2）画出整个程序的结构图；



（3）实现代码（文本）和测试结果（图片）

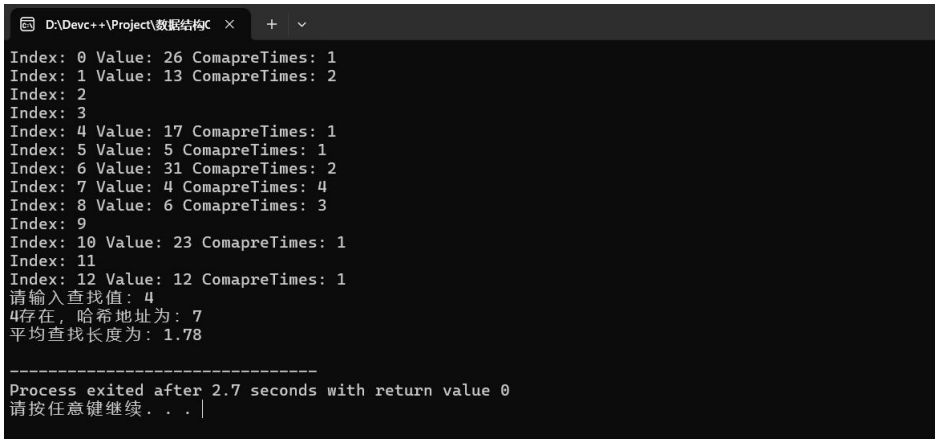
实验代码：

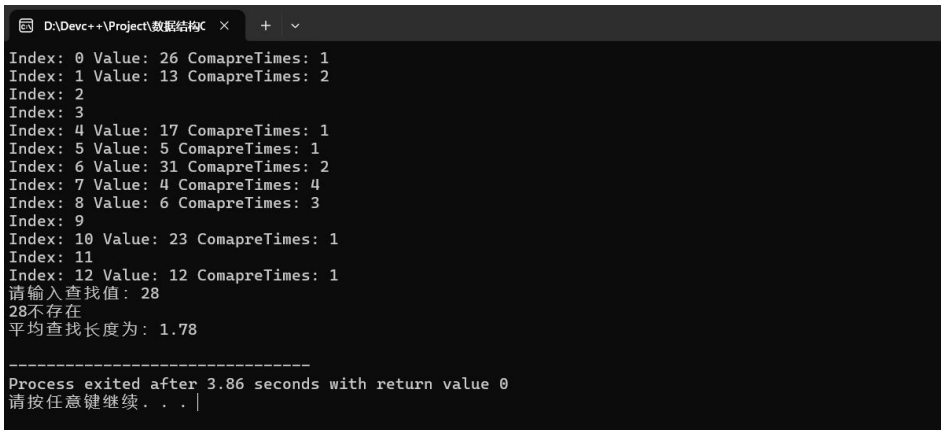
```
#include<stdio.h>
#include<stdlib.h>
typedef struct Elem{
    int key;   // 元素的值
    int compareTimes;   // 查找到该元素需要的比较次数
    int flag;   // 用来表示该位置是否已存有数据
}Elem;
typedef struct HashList{
    Elem *base;
    int length;
}HashList;
// 建立哈希表
void CreatHashList(HashList *HL,int arr[],int n){
    int i;
    HL->length=13;
    HL->base=(Elem*)malloc(HL->length*sizeof(Elem));
    for(i=0;i<HL->length;i++){
        HL->base[i].key=0;
        HL->base[i].compareTimes=0;
```

```
            HL->base[i].flag=0;
        }
        for(i=0;i<n;i++){
            int comparetimes=1;
            int index=arr[i]%13;
            while(HL->base[index].flag){
                index=(index+1)%13;
                comparetimes++;
            }
            HL->base[index].key=arr[i];
            HL->base[index].flag=1;
            HL->base[index].compareTimes=comparetimes;
        }
}
// 输出哈希表
void PrintHashList(HashList HL){
        int i;
        for(i=0;i<HL.length;i++){
            if(HL.base[i].flag){
        printf("Index:%d Value:%dComapreTimes: %d\n",i,HL.base[i].key,HL.base[i].compareTimes);
            }else{printf("Index: %d\n",i);}
        }
}
// 查找元素
void IndexHashList(HashList HL,int key){
        int index=key%13;
        int count=0;
        while(HL.base[index].flag&&HL.base[index].key!=key){
            index=(index+1)%13;
            count++;
        }
        if(HL.base[index].flag&&HL.base[index].key==key){
            printf("%d 存在，哈希地址为：%d\n",key,index);
        }else{
            printf("%d 不存在\n",key);
        }
}
// 平均查找长度
void CalcuASL(HashList HL){
        int i;
        int totalcount = 0;       // 总比较次数
        int filledSlots = 0;
        for(i = 0; i < HL.length; i++){
            if(HL.base[i].flag){ // 如果位置已被填充
```

```
                totalcount += HL.base[i].compareTimes; //  累加比较次数
                filledSlots++; //  增加已填充位置的计数
            }
        }
        double average = (double)totalcount / filledSlots; //  计算平均查找长度
        printf("平均查找长度为：%.2f\n", average);
}
int main(){
        int arr[]={23,5,17,12,26,31,13,4,6};
        int n=sizeof(arr)/sizeof(arr[0]);
        HashList HL;
        CreatHashList(&HL,arr,n);
        PrintHashList(HL);
        int key;
        printf("请输入查找值：");
        scanf("%d",&key);
        IndexHashList(HL,key);
        CalcuASL(HL);
        free(HL.base);
        return 0;
}
```

运行结果：





三、实验心得（必写）

本次实验的目的是实现一个简单的哈希表，并使用线性探测再散列法解决哈希冲突。通过编写 C 语言程序，我深入理解了哈希表的工作原理和哈希函数的设计，以及如何处理哈希表中的数据冲突。

在实验中，我首先定义了两个结构体 Elem 和 HashList 来存储哈希表的元素和整体结构。接着，我实现了 CreatHashList 函数来创建哈希表，初始化所有元素，并使用线性探测法插入数据。PrintHashList 函数用于打印哈希表的内容，IndexHashList 函数用于查找特定元素，CalcuASL 函数用于计算平均查找长度（ASL）。

通过本次实验，我加深了对哈希表工作原理的理解，包括哈希函数的设计、冲突解决策略以及性能评估。在查找外部资料的过程中，我也认识到了在实际应用中哈希表的应用需要考虑的多种因素，如哈希函数的选择、冲突解决策略的效率以及内存管理的重要性。本次实验让我受益匪浅。