

数据结构与算法 第一次实验

学号：22920212204392 姓名：黄勛

一、 实验目的

1. 了解线性表的基础实现方法与原理，理解线性表的基本操作的代码编写方式
2. 学会灵活按照线性表的存储内容自由编写线性表的存储结构
3. 在线性表的基础上进而理解链表的基础实现方法，理解链表的基本操作的代码编写方式
4. 通过实验探索线性表与链表的相似点与区别，发现二者在操作实现上的不同

二、 实验内容

1-1 设链表的存储结构如下：

```
typedef struct Node
{
    Type data;    //数据域; Type: 用户定义数据类型
    struct Node *next;    //指针域
} Node, *LinkList;
```

实现链表的基本操作。

在本次实验中实现的内容：

✓ 创建空链表

```
10 LinkList create() {
11     LinkList list=(LinkList)malloc(sizeof(Node));
12     if(list!=NULL) list->next=NULL;
13     else printf("Create failed!\n");
14     return(list);
15 }
```

✓ 判断链表是否为空

```
18 int isempty(LinkList list) {
19     return(list->next==NULL);
20 }
```

- ✓ 在链表中找某元素的存储位置

```
23 int find(LinkList list,int x) {
24     Node* p=list->next;
25     int i=1;
26     while(p->next!=NULL && p->data!=x){
27         p=p->next;
28         i++;
29     }
30     if(p->data!=x)i=0;
31     return i;
32 }
```

- ✓ 单链表的插入（链表末尾插入元素 x）

```
35 int insert(LinkList list,int x) {
36     Node *p=list,*q=(Node*)malloc(sizeof(Node));
37     while(p->next!=NULL){
38         p=p->next;
39     }
40     if(p==NULL) {
41         printf("Insert failed!\n");
42         return 0;
43     } else {
44         q->data=x;
45         q->next=NULL;
46         p->next=q;
47         return 1;
48     }
49 }
```

- ✓ 单链表的删除（删除第一个值为 x 的结点）

```
52 int delete_node(LinkList list,int x) {
53     Node *p,*q;
54     p=list;
55     if(p->next==NULL) return 0;
56     while(p->next!=NULL && p->next->data!=x)
57         p=p->next; //找值为x的结点的前驱结点的存储位置
58     if(p->next==NULL) { //没找到值为x的结点
59         printf("Not exist!\n");
60         return 0;
61     } else {
62         q=p->next;
63         p->next=q->next;
64         free(q);
65         return 1;
66     }
67 }
```

- ✓ 打印链表

```
70 void show(LinkList list){
71     Node *p=list->next;
72     while(p!=NULL) {
73         if(p->next==NULL)
74             printf("%d",p->data);
75         else
76             printf("%d->",p->data);
77         p=p->next;
78     }
79     printf("\n");
80 }
```

- ✓ Main 函数中对编写操作的具体应用
(编码具体内容见 1-1.cpp)

1-2 实现顺序表的分级查找算法。基本要求包括：

(1) 设计顺序表和索引表的存储结构。

- ① 对于顺序表，用常见的方法存储：

```
7 typedef struct Slist*List;
8 struct Slist {
9     int data[100];
10    int n;
11    int N;
12};
```

- ② 对于索引表 创建 key 值与 link 值一一对应建立索引

```
3 struct index {
4     int key;
5     int link;
6 } index_table[10];
```

(2) 根据顺序表创建索引表。

在实验过程中编写了如下函数创建索引表，其中 List L 为顺序表，通过循环将索引信息写入索引表 a 中，hierarchy 与 hierarchyum 为分级的层次数量。

```
22 void creatindex(int a[][10], int hierarchy, int hierarchyum[], List L) {
23     int i = 0, j = 0, k = 0;
24     for (i = 0; i < hierarchy; i++) {
25         for (j = 0; j < hierarchyum[i]; j++) {
26             a[i][j] = L->data[k];
27             k++;
28         }
29     }
30 }
```

(3) 设计分级查找算法。

设计：要实现分级查找算法，首先要对存储的数据进行分级，并针对用户的分级建立对应的索引表，最后针对创建的索引表创建查找算法。

具体实现：

- 1) 根据级别（层次）数量创建索引表：其中 hierarchy 为级别总数，hierarchyum 为每个级别中的元素个数

建立索引：

```
82 printf("请输入顺序表的储存量\n");
83 scanf("%d", &num);
84 printf("请输入%d个数\n", num);
85 L = creatSlist(num, L);
86 printf("请输入要分成几级\n");
87 scanf("%d", &hierarchy);
88 printf("请输入每级要分成几个\n");
89 for (int i = 0; i < hierarchy; i++) {
90     scanf("%d", &hierarchyum[i]);
91 }
```

建立索引表:

```
40 void findkey(struct index index_table[], int a[][10], int hierarchynum[], int hierarchy) {
41     int i;
42     for (i = 0; i < hierarchy; i++) {
43         index_table[i].key = a[i][hierarchynum[i] - 1];
44         index_table[i].link = i;
45     }
46 }
```

2) 接下来需要编写的就是实现分级查找的功能

① 首先遍历查找级数

```
47 int search_hierarchy(struct index index_table[], int hierarchy, int number) {
48     int i, flag = 0;
49     for (i = 0; i < hierarchy; i++) {
50         if (number <= index_table[i].key) {
51             flag = 1;
52             break;
53         }
54     }
55     if (flag == 1) return index_table[i].link;
56     else {
57         printf("这个数不存在\n");
58         return -1;
59     }
60 }
```

② 最后再在这一级寻找需要的数据

```
61 int searchpos(struct index index_table[], int hierarchy, int ind, int a[][10], int number, int hierarchynum[]) {
62     int flag = 0, j;
63     for (j = 0; j < hierarchynum[ind]; j++) {
64         if (a[ind][j] == number) {
65             flag = 1;
66             break;
67         }
68     }
69     if (flag == 1) {
70         return j;
71     } else {
72         printf("这个数不存在\n");
73         return -1;
74     }
75 }
```

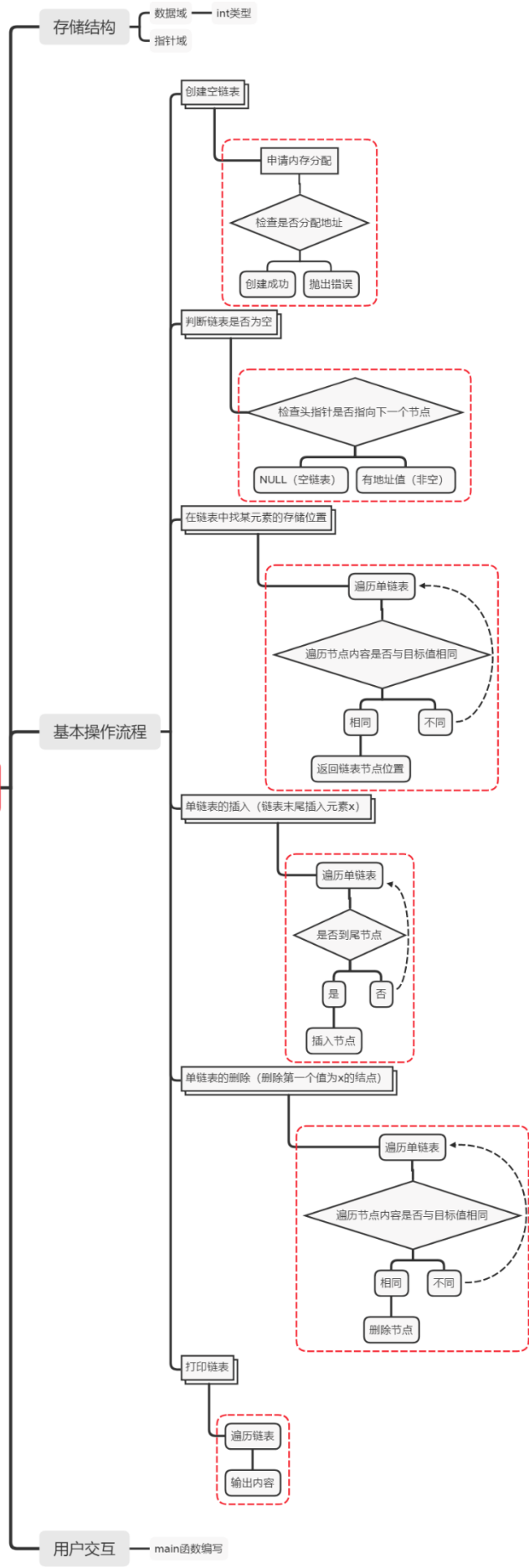
(4) 最后根据编写的功能写出能与用户交互的 main 函数

三、 主要算法流程图

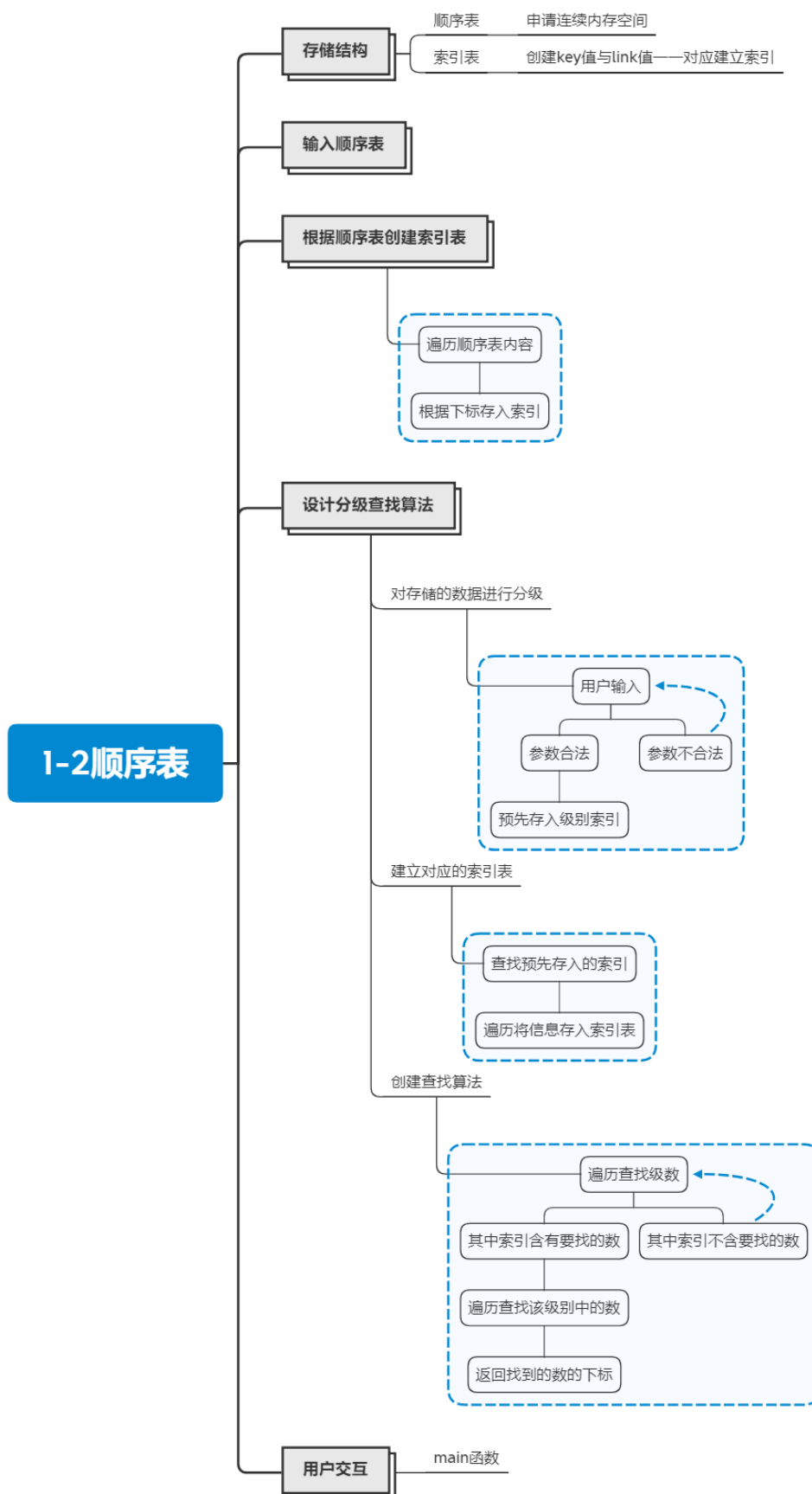
1-1 算法流程

(图片太小可见目录下的 1-1 链表.png)

1-1链表



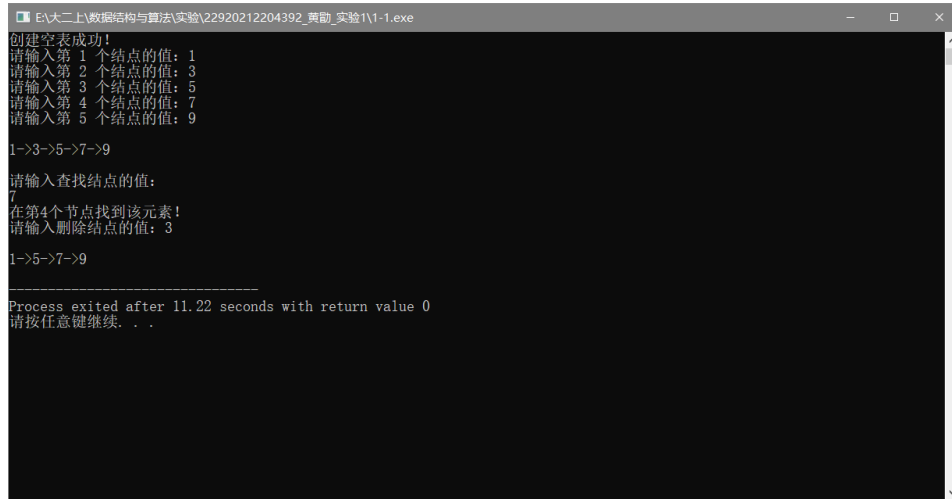
1-2 算法流程



四、 实验结果：

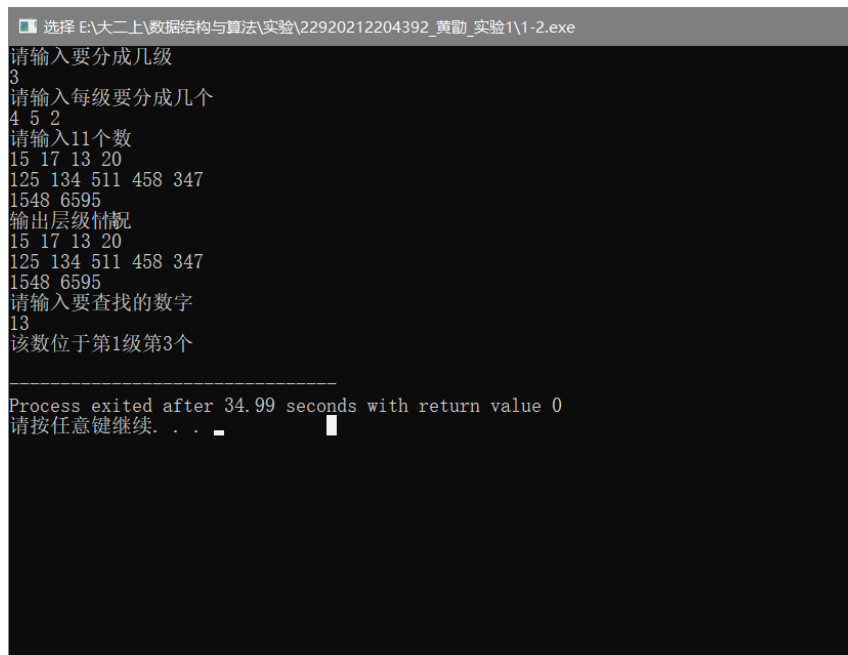
(结合截图说明算法的输入输出)

1、关于 1-1 的输入与输出：



在实际运行中，我创建了 1->3->5->7->9 的链表，在程序中能够得到良好展示，并且实现了查找结点和删除节点的功能。

2、关于 1-2 的输入与输出



在实际运行中，我创建了 15 17 13 20 125 134 511 458 347 1548 6595 的顺序表，并且分成了三级，通过编写的查找算法我成功找到了 13 位于第 1 级第 3 个，程序整个过程运行良好。

五、 实验小结（即总结本次实验所得到的经验与启发等）：

在本次实验中，我尝试具体运用顺序表以及链表，在实体机的实验中我能够更深刻地地理

解对这一部分数据结构的执行方式与特点，并且在编写代码的过程中，我通过不断的调试去寻找语句之间的问题和不足，在潜移默化中提高了我的代码编写能力，这是一次完成效果良好的实验！