

HW1：线性表 HW1 实验报告

姓名：钱子贤 学号：2054170 日期：2021 年 10 月 17 日

实验报告格式按照模板来即可，对字体大小、缩进、颜色等不做要求

实验报告要求在文字简洁的同时将内容表示清楚

1. 涉及数据结构和相关背景

顺序表

顺序表是指采用顺序存储结构的线性表，它利用内存中的一片连续存储区域存放表中的所有元素。可以根据需要对表中的所有数据进行访问，元素的插入和删除可以在表中的任何位置进行。顺序表的基本操作，包括顺序表的创建，第 i 个位置插入一个新的元素、删除第 i 个元素、查找某元素、顺序表的销毁。

2. 实验内容

2.1 1-1 学生信息管理

2.1.1 问题描述

顺序表的基本操作

描述

本题 定义一个包含学生信息（学号，姓名）的的顺序表，使其具有如下功能：(1) 根据指定学生个数，逐个输入学生信息；(2) 给定一个学生信息，插入到表中指定的位置；(3) 删除指定位置的学生记录；(4) 分别根据姓名和学号进行查找，返回此学生的信息；(5) 统计表中学生个数。

2.1.2 基本要求

输入输出要求

第 1 行是学生总数 n

接下来 n 行是对学生信息的描述，每行是一名学生的学号、姓名，用空格分割；
(学号、姓名均用字符串表示)

接下来是 p 行对顺序表的操作：（每行内容之间用空格分隔）

insert i 学号 姓名：表示在第 i 个位置插入学生信息，若 i 位置不合法，输出-1，否则输出 0

remove j:表示删除第 j 个元素，若元素位置不合适，输出-1，否则输出 0

check name 姓名 y: 查找姓名 y 在顺序表中是否存在，若存在，输出其位置序号及学号、姓名，若不存在，输出-1。

check no 学号 x: 查找学号 x 在顺序表中是否存在，若存在，输出其位置序号及学号、姓名，若不存在，输出-1。

end: 操作结束，输出学生总人数，退出程序。

注：全部数值 ≤ 10000 ，元素位置从 1 开始。 学生信息有重复数据（输入时未做检查），查找时只需返回找到的第一个。

每个操作都在上一个操作的基础上完成。

2.1.3 数据结构设计

```
#include<cstdlib>
#include<cstring>
#include<cmath>
#include<algorithm>
#include<iostream>

#include <string.h>
#define max 200
using namespace std;
//顺序表数据结构
typedef struct
{
    int no[max]; //顺序表元素
    char name[max][10];
    int length;
    //顺序表当前长度
}student;

//初始化顺序表函数，构造一个空的顺序表
int initial(student& p)
{
    memset(p.no, 0, sizeof(p.no));
    memset(p.name, 0, sizeof(p.name));
    p.length = 0; //初始化长度为0
    return 0;
}

//创建顺序表函数 初始化前n个数据
bool create1(student& p, int n)
{
    for (int i = 0; i < n; i++)
```

```

    {
        cin >> p.no[i];
        cin >> p.name[i];
        p.length++;
    }
    return true;
}
//创建顺序表函数
void create(student& p)
{
    int n;
    bool flag;
    p.length = 0;
    cin >> n;
    flag = createl(p, n);
}
//查找函数 按位置从小到大查找元素 并返回位置
int search1(student p, char name[10])
{
    for (int i = 0; i < p.length; i++)//从低位置查找
    {
        if (*p.name[i] == *name)
            return i + 1;
    }
    return 0;
}
//查找功能函数 调用search1查找元素
void search(student p)
{
    char name[10]; int flag;
    cin >> name;
    flag = search1(p, name);
    if (flag)
    {
        cout << flag << " " << p.no[flag - 1] << " " << p.name[flag - 1] << endl;
    }
    else
        cout << -1 << endl;
}
//查找函数 查找位置 并返回位置
int searchloc1(student p, int no1)
{
    for (int i = 0; i < p.length; i++)//从低位置查找

```

```

    {
        if (p.no[i] == no1)
            return i + 1;
    }
    return 0;
}

void searchloc(student p)
{
    int no1;
    cin >> no1;
    searchloc1(p, no1);
    int flag = searchloc1(p, no1);
    if (flag)
    {
        cout << flag << " " << p.no[flag - 1] << " " << p.name[flag - 1] << endl;
    }
    else
        cout << -1 << endl;
}

```

//插入函数 位置i插入数据 i及之后元素后移 1=<i<=length+1

```

bool insert1(student& p, int i, int no1, char name[10])
{
    if (i<1 || i>p.length + 1) //判断位置是否有效
    {
        return false;
    }
    if (p.length >= max) //判断存储空间是否已满
    {
        return false;
    }
    for (int j = p.length; j >= i; j--) //位置i及之后元素后移
    {
        p.no[j] = p.no[j - 1];
        *p.name[j] = *p.name[j - 1];
    }
    p.no[i - 1] = no1;
    *p.name[i - 1] = *name;
    p.length++;
    return true;
}

//插入功能函数 调用insert1完成顺序表元素插入 调用printit函数显示插入成功后的结果
void insert(student& p)
{

```

```

    int place;
    int nol;
    char name[20];
    bool flag;
    cin >> place;
    cin >> nol;
    cin >> name;
    flag = insert1(p, place, nol, name);
    if (flag)
    {
        cout << 0 << endl;
    }
    else
        cout << -1 << endl;
}

//删除函数 删除位置i的元素 i之后的元素依次前移
bool deletes1(student& p, int i)
{
    if (i<1 || i>p.length)
    {
        return false;
    }
    for (int j = i; j <= p.length - 1; j++)//位置i之后元素依次前移覆盖
    {
        p.no[j - 1] = p.no[j];
        *p.name[j - 1] = *p.name[j];
    }
    p.length--;
    return true;
}

void deletes(student& p)
{
    int place;
    bool flag;
    cin >> place;
    flag = deletes1(p, place);
    if (flag)
    {
        cout << 0 << endl;
    }
    else
        cout << -1 << endl;
}

```

```

int main()
{

    student p;
    initial(p);
    create(p);
    char order[10];
    char end[10] = "end";
    char insert1[10] = "insert";
    char remove[10] = "remove";
    char check[10] = "check";
    char name[10] = "name";
    char no[10] = "no";
    char next[10];
    while (1)
    {
        cin >> order;
        if (strcmp(order, end)==0)
        {
            cout <<p.length << endl;
            return 0;
        }
        else if (strcmp(order, insert1)==0)
        {
            insert(p);
        }
        else if (strcmp(order, remove)==0)
        {
            deletes(p);
        }
        else if (strcmp(order, check)==0)
        {
            cin >> next;
            if (strcmp(next, name)==0)
                search(p);
            else if (strcmp(next , no)==0)

```

```

        {
            searchloc(p);
        }

    }

}

return 0;

```

2.1.5 调试分析（遇到的问题和解决方法）

```

4
1650400 叶肯
1650800 张一
1652501 李红
1653010 王豪
insert 1 1654660 张三
0
remove 1
0
check name 李红
3 1652501 李红
check no 1650400
1 1650400 叶肯
end
4

```

出现数组越界的情况，需要加大空间。

2.1.6 总结和体会

在只循环一次的情况下，这题有两种解法

一种是用数组存储每次遍历的链表，这种比较简单但浪费一定空间。

第二种是快慢指针，快指针先移动 n 步，然后快慢指针一起移动直到快指针到表尾，此时慢指针的地址为所要删除节点的前一个节点。

注意：需要考虑边界条件，此题的边界条件是当 n 为链表的长度时，此时删除的节点为头节点，需要返回 `head->next`。

2.2 题目二

1-2 有序表的合并

描述

已知线性表 LA 和 LB 中的元素按值非递减有序排列，现将 LA 和 LB 归并成一个新的有序表 LC，且 LC 中的数据元素仍按值非递减有序排列。现请你依次输入该两个有序表，输入时可以是非有序的，输出合并后的结果。

输入

第 1 行 n 个整数, 当为 0 时结束，表示 LA 中的元素。

第 2 行 m 个整数，为 0 表示结束，表示 LB 中的元素。

假定全部数值 ≤ 10000

输出

1 行，合并后的结果

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

int main()
{
    int a[2000] = {0};
    int i = 0;
    int judge = 0;
    while (1)
    {
        cin >> a[i];
        if (a[i] == 0)
        {
            judge++;
            i = i - 1;
        }
        i++;
    }
```



```

        if (judge == 2)
            break;
    }

    for (int j = 0; j < i - 1; j++)
    {
        for (int k = 0; k < i - 1 - j; k++)
        {
            if (a[k] > a[k + 1])
            {
                int temp;
                temp = a[k];
                a[k] = a[k + 1];
                a[k + 1] = temp;
            }
        }
    }
    for (int j = 0; j < 2000; j++)
    {
        if (a[j] != 0)
            cout << a[j] << " ";
        else
            break;
    }
    return 0;
}

```

```

10 20 40 30 0
5 10 15 0
5 10 10 15 20 30 40

```

算法步骤

创建一个空表 **Lc**

依次从 **La** 或 **Lb** 中“摘取”元素值较小的结点插入到 **Lc** 表的最后，直至其中一个表变空为止

继续将 **La** 或 **Lb** 其中一个表的剩余结点插入在 **Lc** 表的最后

2.3 题目三 1-3 扑克牌游戏

题目描述

扑克牌有 4 种花色：黑桃(Spade)、红心(Heart)、梅花(Club)、方块(Diamond)。每种花色有 13 张牌，编号从小到大为：A,2,3,4,5,6,7,8,9,10,J,Q,K。

对于一个扑克牌堆，定义以下 4 种操作命令：

1) 添加(Append)：添加一张扑克牌到牌堆的底部。如命令“Append Club Q”表示添加一张梅花 Q 到牌堆的底部。

2) 抽取(Extract)：从牌堆中抽取某种花色的所有牌，按照编号从小到大进行排序，并放到牌堆的顶部。如命令“Extract Heart”表示抽取所有红心牌，排序之后放到牌堆的顶部。

3) 反转(Revert)：使整个牌堆逆序。

4) 弹出(Pop)：如果牌堆非空，则除去牌堆顶部的第一张牌，并打印该牌的花色和数字；如果牌堆为空，则打印 NULL。

初始时牌堆为空。输入 n 个操作命令 ($1 \leq n \leq 200$)，执行对应指令。所有指令执行完毕后打印牌堆中所有牌花色和数字（从牌堆顶到牌堆底），如果牌堆为空，则打印 NULL

注意：每种花色和编号的牌数量不限。

对于 20% 的数据， $n \leq 20$ ，有 Append、Pop 指令

对于 40% 的数据， $n \leq 50$ ，有 Append、Pop、Revert 指令

对于 100% 的数据， $n \leq 200$ ，有 Append、Pop、Revert、Extract 指令

从右上方下载 p98.py 并运行以生成随机测试数据

输入描述

第一行输入一个整数 n ，表示命令的数量。

接下来的 n 行，每一行输入一个命令。

输出描述

输出若干行，每次收到 Pop 指令后输出一行（花色和数子或 NULL），最后将牌堆中的牌从牌堆顶到牌堆底逐一输出（花色和数字），若牌堆为空则输出 NULL

pop () 是移除堆栈顶部的元素并且返回它的值

push () 是把对象压入堆栈的顶部

这里的堆栈不是特指栈, 是 LinkedList 中特有的方法, LinkedHashMap 和 LinkedHashMap, ArrayList 中没有此方法

进栈 push()

入栈：在栈顶（数组的尾部）添加指定的元素，并返回新数组的长度。

出栈 pop()

出栈：删除栈顶（数组的尾部）的一个元素，并返回删除的元素。

其他

shift():删除数组头部的第一个元素，并返回删除的元素。

unshift():在数组头部的第一个元素前添加一个元素，并返回新数组的长度。

2.4 题目四 1-4 一元多项式的相加和相乘

描述

一元多项式是有序线性表的典型应用，用一个长度为 m 且每个元素有两个数据项（系数项和指数项）的线性表 $((p_1, e_1), (p_2, e_2), \dots, (p_m, e_m))$ 可以唯一地表示一个多项式。本题实现多项式的相加和相乘运算。本题输入不保证有序。

对于%15的数据，有 $1 \leq n, m \leq 15$

对于%33的数据，有 $1 \leq n, m \leq 50$

对于%66的数据，有 $1 \leq n, m \leq 100$

对于 100% 的数据，有 $1 \leq n, m \leq 2000$

本题总分 150 分，得到 100 分或以上可视为满分

尚未测试极限数据

从右上角下载 `p77_data.cpp`，编译并运行生成随机测试数据

输入

第 1 行一个整数 m ，表示第一个一元多项式的长度

第 2 行有 $2m$ 项， $p_1 \ e_1 \ p_2 \ e_2 \ \dots$ ，中间以空格分割，表示第 1 个多项式系数和指数

第 3 行一个整数 n ，表示第二个一元多项式的项数

第 4 行有 $2n$ 项， $p_1 \ e_1 \ p_2 \ e_2 \ \dots$ ，中间以空格分割，表示第 2 个多项式系数和指数

第 5 行一个整数，若为 0，执行加法运算并输出结果，若为 1，执行乘法运算并输出结果；若为 2，输出一行加法结果和一行乘法的结果。

输出

运算后的多项式链表，要求按指数从小到大排列

当运算结果为 0 0 时，不输出。

```

#include <iostream>
using namespace std;
#define OK 0
#define ERROR -1
class LNode {
public:
    float data;
    int expn;
    LNode* next;
    LNode() {
        next = NULL;
    }
};
class LList {
public:
    LNode* head;
    LList() {
        head = new LNode;
        head->next = NULL;
    }
    int LL_insert(int i, float e, int _expn) {
        if (i < 1) return ERROR;
        int j = 1;
        LNode* p = head->next, * q = head;
        while (p && j < i) {
            q = p;
            p = p->next;
            j++;
        }
        if (!p && j < i) return ERROR;
        LNode* node = new LNode;
        node->data = e;
        node->expn = _expn;
        q->next = node;
        node->next = p;
        return OK;
    }
    int LL_delete(int i) {
        if (i < 1) return ERROR;
        int j = 1;
        LNode* p = head->next, * q = head;
        while (!p && j < i) {
            q = p;
            p = p->next;

```

```

        j++;
    }
    if (j <= i) return ERROR;
    q->next = p->next;
    return OK;
}

void LL_display() {
    LNode* p = head->next;
    while (p) {
        if (!p->data) {
            p = p->next;
            continue;
        }
        if (p != head->next) {
            cout << " ";
        }
        if (p->data > 0 || p == head->next) {

            cout << p->data<<" ";
        }
        else
            if (p->data != 0)
                cout << " " << p->data << " ";

        cout << p->expn << " ";
        p = p->next;
    }
    cout << endl;
}

float LL_circle() {
    LNode* p = head->next, * q;
    while (p->next != p) {
        q = p->next->next;
        p->next->next = q->next;
        p = p->next->next;
    }
    return p->data;
}

void LL_AddPolyn(LList& LLb);
LList* LL_MulPolyn(LList& LLb);
void LL_DeletePolyn();
};

void LList::LL_AddPolyn(LList& LLb) {
    LNode* pa = head->next; LNode* pb = LLb.head->next;

```

`LNode* ha = new LNode; LNode* pha = ha;` //ha作和链表的头节点, pha指向和链表末尾节点

```
ha->expn = -1; ha->data = 0;
while (pa && pb) {
    if (pa->expn > pb->expn) {
        pha->next = pb;
        pb = pb->next;
        pha = pha->next;
    }
    else if (pa->expn < pb->expn) {
        pha->next = pa;
        pa = pa->next;
        pha = pha->next;
    }
    else {
        float sum = pa->data + pb->data;
        if (sum != 0.0) {
            pa->data = sum;
            pha->next = pa;
            pa = pa->next;
            pha = pha->next;
        }
        else {
            LNode* tempa = pa;
            pa = pa->next;
            delete tempa;
        }
        LNode* tempb = pb;
        pb = pb->next;
        delete tempb;
    }
}
while (pa) {
    pha->next = pa;
    pa = pa->next;
    pha = pha->next;
}
while (pb) {
    pha->next = pb;
    pb = pb->next;
    pha = pha->next;
}
pha->next = NULL;
head = ha;
```

```

}
LList* LList::LL_MulPolyn(LList& LLb) {
    LNode* pa = head->next;
    LList* AList = new LList;           //累加
    AList->head->data = 0; AList->head->expn = -1;
    while (pa) {
        LList MList;                   //逐项相乘
        int i = 1;
        LNode* pb = LLb.head->next;
        while (pb) {
            float m_data;
            int m_expn;
            m_data = pa->data * pb->data;
            m_expn = pa->expn + pb->expn;
            MList.LL_insert(i++, m_data, m_expn);
            if (pa == head->next)
                AList->LL_insert(i - 1, 0, 0);
            pb = pb->next;
        }
        AList->LL_AddPolyn(MList);
        pa = pa->next;
    }
    return AList;
}

void LList::LL_DeletePolyn() {
    LNode* p = head->next;
    while (p) {
        LNode* q = p;
        p = p->next;
        delete q;
    }
}

int main() {
    LList list;
    int n;
    cin >> n;
    for (int i = 0; i < n; i++) {
        float data;
        int expn;
        cin >> data >> expn;
        list.LL_insert(i + 1, data, expn);
    }
    LList listb;
    cin >> n;

```



```

for (int i = 0; i < n; i++) {
    float data;
    int expn;
    cin >> data >> expn;
    listb.LL_insert(i + 1, data, expn);
}

LList* mulList = list.LL_MulPolyn(listb);
//cout << "["; list.LL_display(); cout << "]"; cout << "*"; cout << "[";
listb.LL_display(); cout << "]"; cout << "=";

int tt;
cin >> tt;
if (tt == 0)
{
    list.LL_AddPolyn(listb);
    list.LL_display();

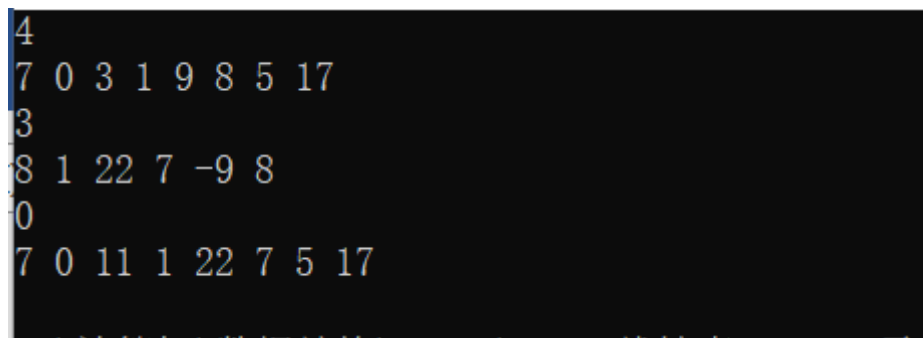
}
else if(tt == 1)
{
    mulList->LL_display();

}
else if (tt == 2)
{
    list.LL_AddPolyn(listb);
    list.LL_display();
    mulList->LL_display();

}

list.LL_DeletePolyn(); //删除整一条和链表
return 0;
}

```



```

4
7 0 3 1 9 8 5 17
3
8 1 22 7 -9 8
7 0 11 1 22 7 5 17

```

我们设计的程序为一元多项式的表示及相加，这是使用 C++ 语言写成。我们使用计算机处理的对象之间通常存在着的是一种最简单的线性关系，这类数学模型可称为线性的数据结

构。而数据存储结构有两种：顺序存储结构和链式存储结构。线性表是最常用且最简单的一种数据结构。所以我们做的是——一元多项式的表示及相加，其过程其实是对线性表的标操作。实验结构和链接存储结构上的运算以及熟练运用掌握的线性表的操作，实现一元 n 次多项式的目的是掌握线性表的基本操作，插入、删除、查找，以及线性表合并等运算在顺序存储的加法运算。学习实现一元 n 次多项式的加法是符号多项式的操作，是表处理的典型用例，需要注意的是：顺序存储结构指的是用数组方法，使用数组方法实现时，在插入和删除的方面，数组不如链表灵活，方法复杂，删除其中一个需要将其后的数组元素改变位置，使其数组保持原有的顺序结构，在查找方面较链表简单，只需要知道其下标就可以知道。链接存储结构指的是用链表方法，值得注意的是，删除和插入较为灵活，不需要变动大多数元素，但是查找过程相对于数组这种顺序存储结构来说较为复杂，耗时巨大。在这里我们在一元多项式的表示与相加实验中，采用的是链式存储结构。

2.5 题目五 1-5 级数相加

级数求和

输入：

若干行，在每一行中给出整数 N 和 A 的值，（ $1 \leq N \leq 150$ ， $0 \leq A \leq 15$ ）

对于 20% 的数据，有 $1 \leq N \leq 12$ ， $0 \leq A \leq 5$

对于 40% 的数据，有 $1 \leq N \leq 18$ ， $0 \leq A \leq 9$

对于 100% 的数据，有 $1 \leq N \leq 150$ ， $0 \leq A \leq 15$

从右上方下载 p95.py 并运行以生成随机测试数据

输出：

对于每一行，在一行中输出级数 $A + 2A^2 + 3A^3 + \dots + NA^N = \sum_{i=1}^N iA^i$ 的整数值

```
#include <iostream>
#include <cstdio>
```

```

#include <string>
#include <algorithm>
using namespace std;
int main()
{
    int n, t;
    int a[10000];
    int sum[10000] = { 0 };//开一个足够大的数组

    cin >> n; //t组测试数据
    cin >> t; //输入想要输出n的多少次方

    for (int i = 1; i <= n; i++)
    {
        for (int qq = 0; qq < n*n; qq++)
            a[qq] = 0;//先把数组每位数初始化为0
        a[0] = 1;//让数的最后一位为1（想想为什么u）
        for (int j = 0; j < i; j++)
        {

            for (int k = 0; k < n * n; k++)
                a[k] *= t; //每次遍历，数组中的每一位数都要乘以2
            for (int k = 0; k < n * n; k++)
            {
                if (a[k] > 9)
                {
                    int kk = a[k] / 10;
                    a[k + 1] += kk; //如果数组中的一位数超过了9，需要进位
                    a[k] = a[k] % 10; //进位后自身取余
                }
            }
        }
    }

    for (int j = 0; j < n*n; j++)
        a[j] *= i; //每次遍历，数组中的每一位数都要乘以2
    for (int j = 0; j < n * n; j++)
    {
        if (a[j] > 9)
        {
            int kk = a[j] / 10;
            a[j + 1] += kk; //如果数组中的一位数超过了9，需要进位
            a[j] = a[j] % 10; //进位后自身取余
        }
    }
}

```

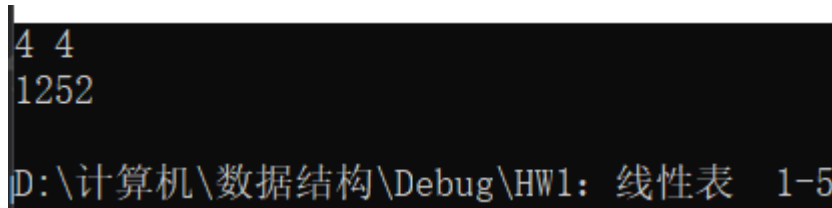
```

for (int j = 0; j < n * n; j++)
    sum[j] += a[j]; //每次遍历，数组中的每一位数都要乘以2
for (int j = 0; j < n * n; j++)
{
    if (sum[j] > 9)
    {
        int kk = sum[j] / 10;
        sum[j + 1] += kk; //如果数组中的一位数超过了9，需要进位
        sum[j] = sum[j] % 10; //进位后自身取余
    }
}

for (int p = n*n; p >= 0; p--) //由于在数组中存数时是从最后一位开始存的，所以要逆序输出
{
    if (sum[p] > 0) //数组中可能有很多位都没用上，值为0，这些0不能输出，找出第一个不是0的位置再输出
    {
        for (int q = p; q >= 0; q--)
            printf("%d", sum[q]);
        break;
    }
}

if (t == 0) printf("1"); //特判一下，如果n==0，2的0次方=1
printf("\n");
return 0;
}

```



```

4 4
1252
D:\计算机\数据结构\Debug\HW1: 线性表 1-5

```

大整数的存储使用数组即可。例如定义 `int` 型数组 `d[1000]`，那么这个数组中的每一位就代表了存放的整数的每一位。如将整数 235813 存储到数组中，则有 `d[0]=3`, `d[1]=1`, `d[2]=8`, `d[3]=5`, `d[4]=3`, `d[5]=2`，即整数的高位存储在数组的高位，整数的低位存储在数组的低位。不反过来存储的原因是，在进行运算的时候都是从整数的低位到高位进行枚举，顺位存储和这种思维相合。因此在读入之后需要在另存为至 `d[]` 数组的时候反转一下。

3.实验总结

这一章的作业都是线性顺序表, 线性表的顺序存储是指用一组地址连续的存储单元依次存储线性表中的各个元素, 使得线性表在逻辑结构上相邻的元素存储在连续的物理存储单元中, 即: 通过数据元素物理存储的连续性来反应元素之间逻辑上的相邻关系。采用顺序存储结构存储的线性表通常简称为顺序表。

顺序存储的线性表的特点:

线性表的逻辑顺序与物理顺序一致;

数据元素之间的关系是以元素在计算机内“物理位置相邻”来体现。

在数据表的第 i 个位置插入元素, 在顺序表的第 i 个位置插入元素 e , 首先将顺序表第 i 个位置的元素依次向后移动一个位置, 然后将元素 e 插入第 i 个位置, 移动元素要从后往前移动元素, 即: 先移动最后一个元素, 在移动倒数第二个元素, 依次类推; 插入元素之前要判断插入的位置是否合法, 顺序表是否已满, 在插入元素之后要将表长 $L \rightarrow \text{length}++$;

顺序表的长度就是就顺序表中的元素的个数, 由于在插入和删除操作中都有对数据表的长度进行修改, 所以求表长只需返回 length 的值即可;

查找顺序表中第 i 个元素的值 (按序号查找), 如果找到, 将将该元素值赋给 e 。查找第 i 个元素的值时, 首先要判断查找的序号是否合法, 如果合法, 返回第 i 个元素对应的值。

删除表中的第 i 个元素 e , 删除数据表中的第 i 个元素, 需要将表中第 i 个元素之后的元素依次向前移动一位, 将前面的元素覆盖掉。移动元素时要想将第 $i+1$ 个元素移动到第 i 个位置, 在将第 $i+2$ 个元素移动 $i+1$ 的位置, 直到将最后一个元素移动到它的前一个位置, 进行删除操作之前要判断顺序表是否为空, 删除元素之后, 将表长 $L \rightarrow \text{length}--$;