作业 HW1 试验报告

姓名: 范潇　学号: 2254298

2023年 10月 15日

1　涉及数据结构和相关背景

本次实验涉及到的数据结构为线性表的顺序存储结构，即顺序表，以及链表。

在计算机内存中,顺序表的存储结构为一段连续的存储单元.

顺序表的基本操作有：

- 顺序表的创建
- 在第i个位置插入元素
- 删除第i个位置的元素
- 查找某个元素
- 顺序表的销毁

顺序表的特点在于用物理位置的相邻及前后关系来表示数据元素之间的逻辑关系.

其优势有:

- 结构简单
- 存储效率高,是紧凑结构
- 为随机存储结构,即如果每个元素占用的存储单位固定,只需要知道顺序表的起始位置(或称为基地址) 以及元素的位序,便可以直接算出该元素所在位置.

缺点有:

- 进行插入和删除操作时,需要移动数据元素,算法效率较低
- 对于长度变化较大的线性表,或者要预先分配较大空间或者要经常扩充线性表,给操作带来不方便.

链表则定义为在内存中用一组任意的存储单元来存储线性表的数据元素，用每个数据元素所带的指针来确定其后继元素的存储位置。这两部分信息组成数据元素的存储映像，称作结点。即每个结点由数据域与指针域(链域),n个结点链接成一个链表.

链表又可以进一步分为单链表、循环链表和双向链表。其中单链表也称为线性链表，含有头结点的单链表较为常用；循环链表则是首尾相接的链表，通常在循环链表中加入表头结点；双向链表则是能够从前驱和后继两个方向遍历的线性链表，常采用带特殊结点的循环链表形式。

2 实验内容

2.1 轮转数组

2.1.1 问题描述

题目： 给定一个整数顺序表nums，将顺序表中的元素向右轮转 k 个位置，其中 k 是非负数。

数据范围：

- $1 \leq$nums.length$\leq 10^5$
- $-2^{31} \leq$nums[i]$\leq 2^{31}-1$
- $0 \leq$k$\leq 10^5$

2.1.2 基本要求

输入要求： 第一行两个整数n和k，分别表示nums的元素个数n，和向右轮转k个位置；第二行包括n个整数，为顺序表nums中的元素。

输出要求： 轮转后的顺序表中的元素,用空格隔开。

2.1.3 数据结构设计

```
typedef struct{
    ElemType *elem;
    int length;//current length
    int listsize;//current allocated size
}SqList;//顺序表
SqList ans;
```

2.1.4 功能说明

Algorithm 1 初始化顺序表

输入： SqList &L待初始化的顺序表， int n置零长度

输出： void

1: **function** InitList_SqList(SqList &L,int n)
2: 申请内存空间并判断是否成功
3: 将前 n个元素置零
4: **end function**

Algorithm 2 将数据写入顺序表中的指定位置

输入： SqList &L已初始化的顺序表， int i写入位序， ElemType e写入值

输出： void

1: **function** InitList_SqList(SqList &L,int n)
2: 判断写入位序 i是否合法
3: 将 e写入该位序对应的数据中//*(L.elem+i-1)=e;
4: **end function**

Algorithm 3 读取顺序表中指定位置的数据

输入： SqList &L已初始化的顺序表， int i读取位序， ElemType e用于存储值的变量

输出： void

1: **function** InitList_SqList(SqList &L,int n)
2: 判断读取位序 i是否合法
3: 将该位序对应的数据写入 e中

4: **end function**

```
InitList_SqList(ans,n);
    for(int i = 0;i<n;i++){
        scanf("%d",&temp);
      ListWrite_sq(ans,(i+k)%n+1,temp);//在写入时体现“向右轮转”
    }
    for(int i =0;i<n;i++){
        GetElem(ans,i+1,temp);//在读取时从第一个元素开始顺序读取
        printf("%d␣",temp);
    }
```

2.1.5 调试分析

一开始尝试着使用在写入时从第一个元素开始，而在读取时体现“向右轮转”，但是这样实现的程序能够通过后面6个测试样例，却没办法通过前4个测试样例，最后我改变策略，使用了上述的程序逻辑，并成功通过所有测试样例。

2.1.6 总结和体会

在确定元素在顺序中的位序时，可以利用模运算将位序限定在合法范围内，从而避免了条件语句的使用。

顺序表可以用于模拟数组。本题中我利用了线性表随机存取的特性，提高了算法的效率。

2.2 学生信息管理

2.2.1 问题描述

题目： 设计一个学生信息管理系统，要求能够实现以下功能：

- 根据指定学生个数，逐个输入学生信息
- 给定一个学生信息，插入到表中指定的位置
- 删除指定位置的学生记录
- 分别根据学生和学号进行查找，返回此学生的信息
- 统计表中学生个数

2.2.2 基本要求

输入要求： 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,字符串长度 <100) 接下来是若干行对顺序表的操作：(每行内容之间用空格分隔)

- insert i：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个位置的学生信息, 若j位置不合法，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：表示操作结束，输出学生总人数，退出程序。

输出要求： 输出每次操作的结果，每个结果占一行。

2.2.3 数据结构设计

```
typedef struct{
    char no[INFO_MAX_LENGTH];
    char name[INFO_MAX_LENGTH];
} ElemType;
typedef struct{
    ElemType *elem;
    int length;//current length
    int listsize;//current allocated size
}SqList;
SqList L;//学生信息列表
char opt[OPT_MAX_LENGTH];//用于存储输入操作
```

2.2.4 功能说明

Algorithm 4 插入学生信息

输入: SqList &L学生信息列表， int i插入位序， ElemType待插入学生信息

输出: 操作状态 *Status*

```
1: function ListInsert_sq(SqList &L,int i,ElemType e)
2:     判断插入位序是否合法
3:     判断是否需要扩容
4:     从表末尾至插入位置依次向后平移一位
5:     在指定位序上插入学生信息
6:     return OK
7: end function
```

Algorithm 5 删除学生信息

输入: SqList &L学生信息列表， int i删除位序

输出: 操作状态 *Status*

```
1: function ListDelete_sq(SqList &L,int i)
2:     判断删除位序是否合法
3:     从删除元素的后一位至表末依次向前平移一位
4:     return OK
5: end function
```

Algorithm 6 查询学生信息

输入: SqList L学生信息列表，char info[]指定信息

输出: void

```
1: function Locate_By_info(SqList L,char info[])
2:     指针遍历顺序表，直到超出范围或所指学生信息符合要求
3:     if 所指学生信息不符合要求 then
4:         打印错误信息
5:     end if
6:     打印所指学生信息
7: end function
```

Algorithm 7 创建学生信息列表

输入: *SqList &L*空学生信息列表，*int n*学生数量

输出: 操作状态

```
1: function CreateList_Sq(SqList L,int n)
```

2: 申请内存空间，并判断是否成功
3: 循环判断是否要增加申请的空间，直至超过学生数量
4: 设置表长
5: 循环读取学生信息，并依次写入顺序表中
6: **return** OK
7: **end function**

2.2.5 调试分析

在一开始调试该程序时，后面几个测试点显示runtime error，因此推断是因为存储信息的数组大小不够导致的。于是把存储信息的数组行数加至10010，但是仍然显示runtime error，后来老师说题目所给的信息有误，最后将行数加至20000便成功通过了。

2.2.6 总结和体会

在顺序表中插入数据时，应从后往前遍历并后移，这样可以避免数据的覆盖；删除数据时，应从前往后遍历并前移，这样可以避免数据的覆盖。

在解答这类信息存储和比较的题目时，要给存储信息的数组以适当的冗余。同时，像在这道题中，学号可以有几十位，此时应该将它视为字符串来处理，尽管它是一串数字。

为了避免由于申请的内存空间不足而导致的运行时错误，可以利用realloc函数来解决，不过这样可能会导致时间增加，对于单个题目而言，可以事先通过数据范围来确定需要申请的内存空间大小。

2.3 一元多项式的相加和相乘

2.3.1 问题描述

题目：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((p1,e1),(p2,e2),...,(pm,em))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。本题输入保证是按照指数项递增有序的。

2.3.2 基本要求

输入要求：

- 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，p1 e1 p2 e2 ...，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，p1 e1 p2 e2 ...，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。

输出要求： 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。

2.3.3 数据结构设计

```
typedef struct {
    int co;
    int deg;
}ElemType,term;
typedef int Status;
//因为这里结构体内部包含自身，无法省去LNode
typedef struct LNode{//普通结点
    ElemType data;
    struct LNode *next;
}LNode;
typedef struct {//特殊结点
    LNode* front;//指向头结点
    LNode* rear;//因为要采用尾插法，所以增加了指向表尾的指针
}LinkList,Poly;
Poly poly1,poly2,ans;//分别存储两个给定的多项式以及答案多项式
```

2.3.4 功能说明

Algorithm 8 创建多项式

输入: LinkList&L多项式，int n 非零项个数

输出: 操作状态

```
1: function CreateList_L(LinkList&L,int n)
2:     为特殊结点申请内存空间并判断是否成功
3:     while 读入个数小于 n  do
4:         为普通结点申请内存空间并判断是否成功
5:         将新结点连接至链表尾部
6:         //L.rear− >next=node;L.rear=node;node− >next=NULL
7:         将读取数据写入新结点中
8:     end while
9:     return OK;
10: end function
```

Algorithm 9 初始化多项式

输入: LinkList&L多项式

输出: 操作状态

```
1: function InitList(LinkList &L)
2:     为特殊结点申请内存空间并判断是否成功
3:     return OK
4: end function
```

Algorithm 10 尾插法

输入: LinkList&L多项式，ElemType e待插入项

输出: 操作状态

```
1: function ListInsert(LinkList &L,ElemType e)
2:     为普通结点申请内存空间，并判断是否成功
3:     将待插入项写入新结点
4:     将新结点连接至链表尾部//为了保证存储的多项式的次数递增
5:     //L.rear− >next=node;L.rear=node;node− >next=NULL
6:     return OK
7: end function
```

Algorithm 11 遍历多项式

输入: LinkList&L多项式

输出: 操作状态

```
1: function TraversePoly(LinkList L)
2:     判断是否多项式是否为空
3:     将工作指针指向开始结点
4:     while 工作指针不指向表尾 do
5:         打印所指项
6:         指向下一个结点
7:     end while
8:     打印当前所指元素
9:     return OK
10: end function
```

Algorithm 12 销毁多项式

输入: LinkList&L多项式

输出: 操作状态

```
1: function DestroyPoly(LinkList L)
2:     工作指针 p, q同时指向头结点
3:     将工作指针指向开始结点
4:     while 工作指针不指向表尾 do
5:         释放当前项所在空间
6:         指向下一项
7:     end while
8:     return OK
9: end function
```

Algorithm 13 多项式相加

输入: LinkList La多项式a，LinkList Lb多项式b，&Lc多项式c

输出: 操作状态

```
1: function AddPoly(LinkList La,LinkList Lb,LinkList &Lc)
2:     将两个工作指针分别指向相加的两个多项式的开始元素
3:     while 两个工作指针都不指向NULL do
4:         if 指向的非零项的次数相同并且系数相加不为零 then
5:             将指向的两个项相加并添加进Lc
6:             两个指针同时指向下一项
```

```
        else
            将两个指针指向的次数更大的那一项添加进Lc
            将那个指针指向下一项
        end if
    end while
    遍历多项式
    return OK
end function
```

Algorithm 14 非零项乘以多项式

输入: Poly L被乘多项式，Poly &ans答案多项式，term operand非零项

输出: 操作状态

```
function AddPoly(Poly L,Poly&ans,term operand)
    工作指针指向被乘多项式开始结点
    while 工作指针不为NULL do
        工作指针所指项与非零项operand系数相乘，次数相加，然后加入答案多项式
        工作指针指向下一项
    end while
    return OK
end function
```

Algorithm 15 多项式乘法

输入: Poly La多项式，Poly Lb多项式

输出: 操作状态

```
function MultiPoly(Linklist La,LinkList Lb)
    初始化答案多项式
    工作指针指向第一个多项式的开始结点
    while 工作指针不为NULL do
        工作指针指向的非零项与第二个多项式相乘，然后加至答案多项式
        工作指针指向下一项
    end while
    遍历答案多项式
    return OK
```

10: **end function**

2.3.5 调试分析

通过打印中间变量来进行调试。

2.3.6 总结和体会

为了使得保存的多项式的系数是递增的，需要使用尾插法来创建链表。链表的优势在多项式的次数分布较为稀疏时体现出来，能够减少内存空间的浪费。

2.4 求级数

2.4.1 问题描述

题目： 求级数

$$\sum_{i=1}^{N} iA^i$$

2.4.2 基本要求

输入要求： 若干行，在每一行中给出整数 N和 A的值

输出要求： 对每一行输入，在一行中输出结果。

2.4.3 数据结构设计

在该题中，我用一维数组来模拟高精度运算，最终的结果也用一维数组来存储，其中第1个元素是个位数，第2个元素是十位数，以此类推。

```
typedef struct{
    ElemType elem[LIST_SIZE];
    int listsize;
    int length;
}num;
num sum,temp;//分别用于存储求和值和中间值
```

2.4.4 功能说明

```
    //转换为高精度数
Status CreateNum(num &L,int num){
    L.length = 0;
    L.listsize = LIST_SIZE;
    if(num<10)
    L.elem[L.length++]=num;
    if(num>=10){//利用了题目中所给的数据范围
        L.length =2;
        L.elem[0]=num-10;
        L.elem[1]=1;
    }
    return OK;
}
```

```
    //打印高精度数
Status PrintNum(num La){
    for(int i =La.length-1;i>=0;i--)//从后向前打印
      printf("%d",La.elem[i]);
    printf("\n");
    return OK;
}
```

```
    //高精度相加
Status Add(num src,num &dst){
    int i = 0;
    int remainder=0;
    while(i<dst.length){//遍历所有目标高精度中的项
        dst.elem[i]+=remainder;
        if(i<src.length)//当起始高精度还有剩余元素时，将其加至目标高精度中
          dst.elem[i]+=src.elem[i];
        remainder= dst.elem[i]/10;//求余数
        dst.elem[i]%=10;//模运算
        i++;
    }
    while(i<src.length||remainder){
        //当余数不为零或者起始高精度还有剩余项时，继续添加至目标高精度中
        dst.elem[dst.length++]=remainder;
        if(i<src.length)
```

```
17          dst.elem[i]+=src.elem[i];
18        remainder=dst.elem[i]/10;//求余数
19        dst.elem[i]%=10;//模运算
20        i++;
21    }
22    return OK;
23 }
```

Algorithm 16 常数与高精度数相乘

输入: int n,常数，num &operand高精度数

输出: 操作状态

1: **function** MULTI(int c,num &operand)
2: **if** c¿=10 **then**
3: 通过高精度数中所有元素后移一位来得到高精度数乘以10
4: 求c-10乘以高精度数
5: 两个高精度数相加得到所求高精度数
6: **else**
7: **while** 遍历高精度数 **do**
8: 当前项乘以常数，再加上余数
9: 除以十得到余数
10: 模10得到最终值
11: **end while**
12: **end if**
13: **end function**

```
1 //求和
2 CreateNum(sum,0);
3       for(int i =1;i<=N;i++){
4           CreateNum(temp,i);
5           for(int j =1;j<=i;j++)
6             Multi(A,temp);//求幂
7           Add(temp,sum);//累加
8       }
9       PrintNum(sum,visit);
```

2.4.5 调试分析

一开始进行测试时，发现程序会返回一些绝对值很大的正数或者负数，并且同样的输入输入不一样，推断是出现了内存非法访问，于是对程序进行了针对性的检查，最后调试成功。

2.4.6 总结和体会

尽管这个级数求和能够给出准确的求和公式，但是由于数值很大，只能用高精度来求解。

利用顺序表的来模拟高精度时，可以选择像用数组一样，先将元素全部置零，每次进行加法和乘法时都是遍历整个表。但是这样会导致效率降低。如果选择只遍历必要的元素，则需要注意边界值的处理，避免表的长度比实际的要大，这会导致打印高精度数时出错。

2.5 扑克牌游戏

2.5.1 问题描述

题目：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。
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（$1 \leqslant n \leqslant 200$），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。

2.5.2 基本要求

输入要求： 第1行一个整数n，表示操作命令的个数。接下来n行，每行一个操作命令。

输出要求： 输出若干行，每次收到Pop指令后输出一行（花色和数字或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。

2.5.3 数据结构设计

```
typedef int Status;
typedef struct{//普通结点
    int suit;
    int point;
}card,ElemType;
typedef struct Card{//双向链表特殊结点,
    Card* prior;//从底部指向顶部
    Card* next;//从顶部指向底部
    ElemType card;
}Card,*Deck;//因为操作涉及大量的删除和添加操作，所以选用链表Extract
Deck deck;//又因为操作，所以选择使用双向链表Revert
//-----------------------------------------
// prior <- bottom Head top -> next
//-----------------------------------------
const char *suits[]={"Spade","Heart","Club","Diamond","ERROR"};
const char *points[]={"ERROR","A","2","3","4","5","6","7","8","9","10","J","
    Q","K"};
```

2.5.4 功能说明

Algorithm 17 创建排队

输入： Deck&L牌堆

输出： 操作状态

1: **function** InitDeck(Deck &L)
2: 为特殊结点申请内存空间并判断是否成功
3: **return** OK

4: **end function**

Algorithm 18 打印牌堆

输入: Deck &L牌堆

输出: 操作状态

1: **function** PrintDeck(Deck L)
2: 判断牌堆是否为空
3: 工作指针指向开始元素
4: **while** 工作指针不指向头结点 **do**
5: 打印所指数据
6: 工作指针指向下一个结点
7: **end while**
8: **return** OK
9: **end function**

```
//从底部插入
Status AppendCard(Deck &L,card in){
    Card * p = (Card*)malloc(sizeof(Card));
    if(!p){
printf("OVERFLOW\n");
      return OVERFLOW;
    }
    p->card=in;
    p->next = L;
    p->prior = L->prior;
    L->prior->next=p;
    L->prior=p;
    return OK;
}
```

```
Status PopCard(Deck &L){//从顶部删除
    if(L->next==L){
        printf("NULL\n");
        return ERROR;
    }
    Card*p = L->next;//记录待删除的结点
    L->next=L->next->next;
    L->next->prior=L;
    printf("%s␣%s\n",suits[p->card.suit],points[p->card.point]);
```

```
    free(p);
    return OK;
}
```

```
    //反转
Status RevertDeck(Deck &L){
    Card* p = L,*temp;
    temp=p->next;//将头结点的两个指针互换
    p->next = p->prior;
    p->prior=temp;
    p=p->prior;//按照原来的从顶到底的方向移动，由于之前已经互换，所以
        是p=p->prior
    while(p!=L){//遍历其他结点，并将它们的两个指针互换
        temp=p->next;
        p->next = p->prior;
        p->prior=temp;
        p=p->prior;
    }
    //PrintDeck(deck);
    return OK;
}
```

Algorithm 19 抽出指定花色的牌，排序后再放回牌堆顶部

输入: Deck&L牌堆，int suit指定花色

function EXTRACT(Deck&L,int suit)
    $Sort(suit\_list, pt\_list, suit, top)$
    检查牌堆是否为空
    初始化备用牌堆
    while i遍历1到13 **do**
        while 工作指针遍历L **do**
            if 所指牌花色为指定花色，且点数为i **then**
                将该牌从原牌堆删去，尾插至备用牌堆//确保点数递增
            end if
        end while
    end while
    if 备用牌堆不为空 **then**
        将备用牌堆插入原牌堆顶
    end if

15: **end function**

2.5.5 调试分析

在调试本程序时，我输出了大量的中间变量来辅助判断错误。同时我还将测试成功的输入与测试失败的相比较，来找出是什么输出导致了错误。

2.5.6 总结和体会

本题的易错点之一在于点数的比较，不能够简单地用ASCII码来来比较。我的处理方法是将格式不统一的点数以及花色统一转化为数字。这种处理一是能够方便比较，二是配合数组便于输出打印。

这道题的数据结构的设计也十分重要，如果选择顺序表，则Extract函数的效率将会大大减少。同时，双向链表的使用也使得Revert函数的实现变得简单。

3 实验总结

在编写程序时，不仅仅要确保其正确性，还要确保其可读性，因为这样能够大大增加程序的可维护性，并且减少了调试程序所需要的时间。通过这次实验，我体会到选取合适的数据结构不仅仅能够提高程序运行的效率，对于功能实现的难易程度也有很大的影响。在设计数据结构时，要综合考虑数据的形态，例如大小、分布情况等等，以及对于数据的操作。