顺序表

7-1 顺序表的建立及遍历

读入n值及n个整数，建立顺序表并遍历输出。

输入格式:

读入n及n个整数

输出格式:

输出n个整数，以空格分隔（最后一个数的后面没有空格）。

输入样例:

在这里给出一组输入。例如：

```
4
-3 10 20 78
```

输出样例:

在这里给出相应的输出。例如：

```
-3 10 20 78
```

代码：

```
#include<stdio.h>

int main() {
    int n;
    scanf("%d", &n);

    int a[100];
    for (int i = 0; i < n; i++)
        scanf("%d", &a[i]);

    for (int i = 0; i < n; i++) {
        if (i == n - 1)
            printf("%d", a[i]);
        else
            printf("%d ", a[i]); // 我在这里添加了一个空格，这样元素之间就会有空格分隔
    }

    return 0;
}
```

7-2 递增有序顺序表的插入

实验目的：1、掌握线性表的基本知识 2、深入理解、掌握并灵活运用线性表。3、熟练掌握线性表的存储结构及主要运算的实现
已知顺序表L递增有序，将X插入到线性表的适当位置上，保证线性表有序。。

输入格式:

第1行输入顺序表长度，第2行输入递增有序的顺序表，第3行输入要插入的数据元素X。

输出格式:

对每一组输入，在一行中输出插入X后的递增的顺序表。

输入样例:

在这里给出一组输入。例如：

```
5
1 3 5 7 9
6
```

输出样例:

在这里给出相应的输出。例如：

```
1,3,5,6,7,9,
```

代码：

```
#include<stdio.h>
int main()
{
    int a,b[100];
    scanf("%d",&a);
    for(int i=0;i<a;i++)
    {
        scanf("%d",&b[i]);
    }
    int c;
    scanf("%d",&c);
    if(a==0) b[0]=c;
    else{
        for(int i=0;i<a;i++)
    {
        if(c<b[i])
        {
            for(int j=a;j>i;j--)
                b[j]=b[j-1];
            b[i]=c;
            break;
        }
    }
```

```
    }
    for(int i=0;i<a+1;i++)
    {
        printf("%d,",b[i]);
    }
}
```

7-3 顺序表（删除）

已知一组数据，采用顺序存储结构存储，其中所有的元素为整数。设计一个算法，删除元素值在[x,y]之间的所有元素

输入格式:

输入包含三行数据，第一行是表中元素个数，第二行是顺序表的各个元素，第三行是区间x和y。

输出格式:

删除元素值在[x,y]之间的所有元素后，输出新的顺序表。(最后无空格)

输入样例:

在这里给出一组输入。例如：

```
10
55 11 9 15 67 12 18 33 6 22
10 20
```

输出样例:

在这里给出相应的输出。例如：

```
55 9 67 33 6 22
```

代码：

```
#include<stdio.h>
int main()
{
    int n,a[100],b[100],x,y,j=0;
    scanf("%d",&n);
    for(int i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    scanf ("%d %d",&x,&y);
    for(int i=0;i<n;i++)
    {
        if(a[i]<x||a[i]>y)
        {
            b[j]=a[i];
            j++;
        }
```

```
    }
    for(int i=0;i<j-1;i++)
        printf("%d ",b[i]);
    printf("%d",b[j-1]);
    return 0;
}
```

7-4 最大子列和问题

给定*K*个整数组成的序列{ *N*1, *N*2, ..., *N***K* }，“连续子列”被定义为{ *N***i*, *N***i*+1, ..., *N***j* }，其中1≤*i*≤*j*≤*K*。“最大子列和”则被定义为所有连续子列元素的和中最大者。例如给定序列{ -2, 11, -4, 13, -5, -2 }，其连续子列{ 11, -4, 13 }有最大的和20。现要求你编写程序，计算给定整数序列的最大子列和。

本题旨在测试各种不同的算法在各种数据情况下的表现。各组测试数据特点如下：

- 数据1：与样例等价，测试基本正确性；
- 数据2：102个随机整数；
- 数据3：103个随机整数；
- 数据4：104个随机整数；
- 数据5：105个随机整数；

输入格式:

输入第1行给出正整数*K* (≤100000)；第2行给出*K*个整数，其间以空格分隔。

输出格式:

在一行中输出最大子列和。如果序列中所有整数皆为负数，则输出0。

输入样例:

```
6
-2 11 -4 13 -5 -2
```

输出样例:

```
20
```

代码:

```
#include<stdio.h>
int main(){
    int a,b[100000],ThisSum=0,MaxSum=0;
    scanf("%d",&a);
    for(int i=0;i<a;i++)
    {
        scanf("%d",&b[i]);
    }
    for(int i=0;i<a;i++)
    {
        ThisSum=ThisSum+b[i];
```

```
        if(ThisSum>MaxSum)
            MaxSum=ThisSum;
        if(ThisSum<0)
            ThisSum=0;
    }
    printf("%d",MaxSum);
    return 0;
}
```

7-5 数组元素循环右移n位

从键盘接收两个整数m和n，分别表示一维整型数组的元素个数，和要向移动的位数。已知0<m<=100，以及n>0。

在用户输入m和n后，第二行输入相应个数的数组元素。

程序要实现的功能是，**让数组元素往右移动n位。**

例如，数组的5个元素是：1,2,3,4,5。

往右移动1位后：5,1,2,3,4

往右移动2位后：4,5,1,2,3

输入格式:

第一行输入两个整数，第二行输入数组元素。

输出格式:

移动后，数组的每一个元素，注意每个数组元素后有且仅有一个空格。

输入样例:

第一行的数据5和2，表示数组容量为5，让数组元素往右移动2个位置。

第二行是数组的每一个元素的值。

```
5 2
1 2 3 4 5
```

输出样例:

输出移动后的数组元素值，注意每个元素后**有且仅有**一个空格。

```
4 5 1 2 3
```

代码:

```
#include<stdio.h>
int main()
{
    int x,y,a[100];
    scanf("%d %d",&x,&y);
    int n;
```

```
    n = y % x ;
    for(int i = 0;i < x;i++)
    {
        scanf("%d",&a[i]);
    }
    for(int j=1;j<=n;j++)
        {
            int b=a[x-1];
            for(int i=x-1;i>0;i--)
                a[i]=a[i-1];
            a[0]=b;
        }
    for(int i = 0;i < x;i++)
    {
        printf("%d ",a[i]);
    }
    return 0;
}
```

7-6 单词逆置

输入一个可能包含若干（至少1个）单词的句子（可以假设每个单词之间有且仅有一个空格，标点符号视为单词的组成部分），输出每个单词逆置后的英文句子（参看样例输出）。

输入格式:

首先输入一个正整数T，表示测试数据的组数，然后是T组测试数据。每组测试数据输入一个字符串（长度不超过80），表示英文句子。

输出格式:

对于每组测试，输出每个单词逆置后的英文句子。

输入样例:

```
1
emoclew era uoY
```

输出样例:

```
welcome are You
```

代码:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_SIZE 80

typedef struct {
    char items[MAX_SIZE]; // 栈的存储空间，使用字符数组
```

```
    int top;               // 栈顶位置的索引
    int stackLength;     // 栈的最大长度
} Sqstack;

// 初始化栈
void initStack(Sqstack *s) {
    s->top = -1;           // 初始化栈顶索引为-1，表示栈为空
    s->stackLength = MAX_SIZE; // 设置栈的最大长度
}

// 压栈操作
int push(Sqstack *s, char ch) {
    if (s->top >= s->stackLength - 1) {
        return 0; // 如果栈满，返回0
    }
    s->items[++s->top] = ch; // 将元素压入栈顶
    return 1;
}

// 弹栈操作
int pop(Sqstack *s, char *e) {
    if (s->top == -1) {
        return 0; // 如果栈空，返回0
    }
    *e = s->items[s->top--]; // 弹出栈顶元素
    return 1;
}

// 逆置字符串中的每个单词
void rotate(char arr[]) {
    char temp[MAX_SIZE]; // 临时数组，用于存储逆置的单词
    int i, j = 0;
    Sqstack s;
    initStack(&s); // 初始化栈

    // 逆置每个单词并输出
    for (i = 0; arr[i] != '\0'; i++) {
        if (arr[i] != ' ') {
            if (!push(&s, arr[i])) {
                break; // 如果栈满，退出循环
            }
        } else {
            // 当遇到空格时，逆置并输出栈中的单词
            while (pop(&s, &temp[j++]) == 1) {
                printf("%c", temp[j - 1]);
            }
            printf("%c", arr[i]); // 输出空格
            j = 0; // 重置临时数组的索引
        }
    }
    // 逆置并输出最后一个单词（如果有的话）
    while (pop(&s, &temp[j++]) == 1) {
        printf("%c", temp[j - 1]);
    }
    printf("\n"); // 每组测试数据后输出换行符
}
```

```
int main() {
    int T;
    char arr[MAX_SIZE];
    scanf("%d", &T);
    getchar(); // 读取并忽略换行符
    while (T--) {
        gets(arr); // 读取字符串
        rotate(arr); // 逆置并输出单词
    }
    return 0;
}
```

7-7 一元多项式的乘法与加法运算

设计函数分别求两个一元多项式的乘积与和。

输入格式:

输入分2行，每行分别先给出多项式非零项的个数，再以指数递降方式输入一个多项式非零项系数和指数（绝对值均为不超过1000的整数）。数字间以空格分隔。

输出格式:

输出分2行，分别以指数递降方式输出乘积多项式以及和多项式非零项的系数和指数。数字间以空格分隔，但结尾不能有多余空格。零多项式应输出`0 0`。

输入样例:

```
4 3 4 -5 2  6 1  -2 0
3 5 20  -7 4  3 1
```

输出样例:

```
15 24 -25 22 30 21 -10 20 -21 8 35 6 -33 5 14 4 -15 3 18 2 -6 1
5 20 -4 4 -5 2 9 1 -2 0
```

代码:

```
#include<stdio.h>
int main()
{
  //指数相加的最大值为2000;
  int j,h[10000]={0},h1[10000]={0},m1[10000]={0},m[10000]={0},k=0,k1=0,t1[2000]=
{0},t[2000]={0};
  int  n,m3,a[10000]={0},a1[10000]={0},b[10000]={0},b1[10000]={0},c[10000]=
{0},c1[10000]={0},d[10000]={0},d1[2000]={0},i;
  scanf("%d",&n);
  for(i=0;i<n;i++)
  {
      scanf("%d %d",&a[i],&a1[i]);
      d[a1[i]]=d[a1[i]]+a[i];
```

```
  }
 scanf("%d",&m3);
  for(i=0;i<m3;i++)
  {
    scanf("%d %d",&b[i],&b1[i]);
    d[b1[i]]=d[b1[i]]+b[i];
  }
  k=0;
  for(i=0;i<n;i++)
  {
    for(j=0;j<m3;j++)//不可以此处写k++;
    {
      c[k]=a[i]*b[j];
      c1[k]=a1[i]+b1[j];//多项式相乘，系数相乘，指数相加;
      h[c1[k]]=h[c1[k]]+c[k];
      k++;//易错点：否则当j取到m3时跳出循环;
      //就不再执行k++;就少算了;
    }
  }
  k1=0;
  for(i=2000;i>=0;i--)//乘法运算;
  {
    if(h[i]!=0)
    {
      m[k1]=h[i];//m数组用来储存乘法运算结果的系数;
      m1[k1]=i;//m1来储存乘法运算结果后的指数;
      k1++;
    }
  }
  for(i=0;i<k1;i++)
  {
    printf("%d %d",m[i],m1[i]);
    if(i<k1-1) printf(" ");
  }
  if(k1==0)
  {
    printf("0 0");//输出零多项式;
  }
  printf("\n");
  k=0;
  for(i=2000;i>=0;i--)
  {
    if(d[i]!=0)
    {
      t[k]=d[i];
      t1[k]=i;
      k++;
    }
  }
  for(i=0;i<k;i++)
  {
    printf("%d %d",t[i],t1[i]);
    if(i<k-1) printf(" ");
  }
  if(k==0) printf("0 0");//有零多项式;
  return 0;
```

```
}
```

7-8 一元多项式的加法

设计程序求两个一元多项式的和。

输入格式:

输入分2行，每行分别先给出多项式非零项的个数，再以指数递降方式输入一个多项式非零项系数和指数。数字间以空格分隔。

输出格式:

输出1行，以指数递降方式输出和多项式非零项的系数和指数（保证不超过整数的表示范围）。数字间以空格分隔，但结尾不能有多余空格。零多项式应输出0 0。

输入样例:

```
4 3 4 -5 2  6 1  -2 0
3 5 20  -7 4  3 1
```

输出样例:

```
5 20 -4 4 -5 2 9 1 -2 0
```

代码:

```
#include<stdio.h>
#include<stdlib.h>

struct node
{
    int coef;
    int index;
    struct node *next;
};

struct node *Read()
{
    struct node *head, *tail, *p;
    int i,n;
    head=(struct node *)malloc(sizeof(struct node));
    head.next=NULL;
    tail=head;
    scanf("%d",&n);
    for(i=1; i<=n; i++)
    {
        p=(struct node *)malloc(sizeof(struct node));
        scanf("%d %d",&p->coef,&p->index);
        p->next=NULL;
        tail->next=p;
```

```
        tail=p;
    }
    tail->next = NULL;
    return head;
}

struct node *addition(struct node *L1,struct node *L2)
{
    struct node *t1,*t2,*L3,*p,*head;
    t1 = L1->next;
    t2 = L2->next;
    head = (struct node*)malloc(sizeof(struct node));
    head->next = NULL;
    L3 = head;
    while(t1 && t2)
    {
        p = (struct node*)malloc(sizeof(struct node));
        if(t1->index == t2->index)
        {
            p->coef = t1->coef + t2->coef;
            p->index = t1->index;
            t1 = t1->next;
            t2 = t2->next;
        }
        else if(t1->index < t2->index)
        {
            p->coef = t2->coef;
            p->index = t2->index;
            t2 = t2->next;
        }
        else if(t1->index > t2->index)
        {
            p->coef = t1->coef;
            p->index = t1->index;
            t1 = t1->next;
        }
        L3->next = p;
        L3 = L3->next;
    }
    if(t1)
        L3->next = t1;
    else if(t2)
        L3->next = t2;
    return head;
}

void Print(struct node *L)
{
    struct node *p = L->next;
    int flag = 1;
    // for(; p; p = p->next)
    while (p)
    {
        if(flag == 0 && p->coef)
        {
            printf(" ");
```

```
        }
        if(p->coef)
        {
            printf("%d %d",p->coef,p->index);
            flag = 0;
        }
        p = p->next;
    }
    if(flag == 1)
        printf("0 0");
    printf("\n");
}

int main()
{
    struct node *L1,*L2,*head;
    L1 = Read();
    L2 = Read();
    head = addition(L1,L2);
    Print(head);
    return 0;
}
```

7-9 合并有序数组

给定2个非降序序列，要求把他们合并成1个非降序序列。假设所有元素个数为N，要求算法的时间复杂度为O(N)。

输入格式:

输入有4行。
第1行是一个正整数m，表示第2行有m个整数，这些整数构成一个非降序序列，每个整数之间以空格隔开。第3行是一个正整数n，表示第4行有n个整数，这些整数也构成一个非降序序列，每个整数之间以空格隔开。

输出格式:

把第2行的m个整数和第4行的n个整数合并成一个非降序序列，输出这个整数序列。每个数之间隔1个空格。

输入样例:

```
6
1 3 6 6 8 9
4
2 4 5 7
```

输出样例:

```
1 2 3 4 5 6 6 7 8 9
```

代码：

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

typedef struct node
{
 int data;
 struct node*next;
}node,*listnode;

listnode creative(int n)
{
 listnode head;
 head=(node*)malloc(sizeof(struct node));
 head->next=NULL;
 node*p,*rear;
 p=rear=head;
 for(int i=0;i<n;i++)
 {
  p=(node*)malloc(sizeof(struct node));
  scanf("%d",&p->data);

  rear->next=p;
  rear=p;
 }
 rear->next=NULL;
 return head;
}

listnode marge(listnode s1,listnode s2)
{
    if(s1==NULL)
    {
        return s2;
    }
    else if(s2==NULL)
    {
        return s1;
    }
    else
    {
        if(s1->data < s2->data)
        {
            s1->next=marge(s1->next,s2);
            return s1;
        }
        else
        {
            s2->next=marge(s2->next,s1);
            return s2;
        }
    }
}
```

```
void print(node*l1)
{
  l1=l1->next;
 while(l1->next!=NULL)
 {
  l1=l1->next;
  printf("%d ",l1->data);
 }
 printf("\n");
}

int main()
{
 int n,m;

 scanf("%d",&n);

 listnode s1;

 s1=(node*)malloc(sizeof(struct node));

 s1=creative(n);

 scanf("%d",&m);

 listnode s2;

 s2=(node*)malloc(sizeof(struct node));

 s2=creative(m);

 listnode l1=NULL;
 l1=marge(s1,s2);


   print(l1);
}
```