哈尔滨工业大学（深圳）2021 年春《数据结构》
第一次作业 线性结构

学号		姓名		成绩	

1、简答题

1.1 简述线性链表头指针,头结点,首元结点(第一个元素结点) 三个概念的区别。

【参考答案】头指针是指向链表中第一个结点的指针。首元结点是指链表中存储第一个数据元素的结点。头结点是在首元结点之前附设的一个结点,该结点不存储数据元素,其指针域指向首元结点,其作用主要是为了方便对链表的操作。它可以对空表、非空表以及首元结点的操作进行统一处理。

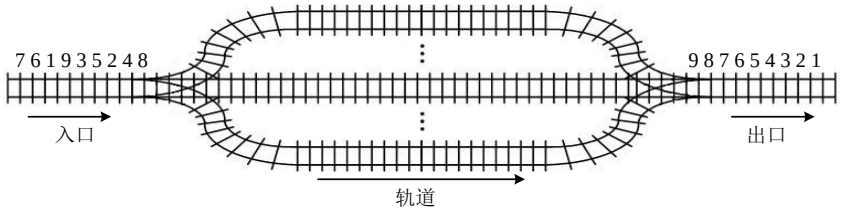
1.2 在什么情况下用顺序表比链表好?

【参考答案】当线性表的数据元素在物理位置上是连续存储的时候,用顺序表比用链表好,其特点是可以进行随机存取。

1.3 简述队列和栈这两种数据类型的相同点和差异处

【参考答案】栈和队列相同出是都是线性结构，是线性表的特殊形式，属于限定性的线性表。但与线性表可以在任何位置插入或删除元素不同，栈只允许在表尾（或表的一端）做插入和删除操作，形成后进先出结构；队列的插入和删除操作分别限定在表的两端进行（表头删除，表尾插入），形成先进先出的结构。

1.4 设有如下图所示的火车车轨，入口到出口之间有 n 条轨道，列车的行进方向均为从左至右，列车可驶入任意一条轨道。现有编号为 1～9 的 9 列列车，驶入的次序依次是 8,4,2,5,3,9,1,6,7。若期望驶出的次序依次为 1 至 9，则 n 至少是多少？



【参考答案】n>=4

1.5 现有队列 Q 与栈 S,初始时队列 Q 中的元素依次是 1,2,3,4,5,6(1 在队头),栈 S 为空。若仅允许下列 3 种操作：

① 出队并输出出队元素；

② 出队并将出队元素入栈；

③ 出栈并输出出栈元素。

请分析是否能得到 1,2,5,6,4,3 和 3,4,5,6,1,2 两个输出序列,为什么？

【参考答案】可以得到 1,2,5,6,4,3 序列，但得不到 3,4,5,6,1,2 序列。简单说明过程出队、进栈、退栈、输出过程即可

1.6 假设按低下标优先存储整数数组 A(-3:8,3:5,-4:0,0:7)时，第一个元素的字节存储地址是 100，每个整数占 4 个字节。问：A(0,4,-2,5)的存储地址是什么？请简要说明计算方法。

【参考答案】

A(-3:8,3:5,-4:0,0:7)对应规范化的数组为：A(0:11,0:2,0:4,0:7)

元素 A(0,4,-2,5)　相当于 A(3,1,2,5)

所以有：L(3,1,2,5)=100+4(3*3*5*8+1*5*8+2*8+5)=1784

可以理解成一本书：

书的每行字数＝8 个

书的每页字数＝5*8＝40 个

每本书的字数＝3*40＝120 个

2、数据结构设计及算法描述

给出一个停车场需求如下：

设停车场是一个可以停放 n 辆汽车的狭长通道，且只有一个大门可供汽车进出。汽车在停车场内按车辆到达时间的先后顺序，依次由北向南排列（大门在最南端，最先到达的第一辆车停放在车场的最北端），若车场内已停满 n 辆车，那么后来的车只能在门外的便道上等候，一旦有车开走，则排在便道上的第一辆车即可开入；当停车场内某辆车要离开时，在它之后进入的车辆必须先退出车场为它让路，待该辆车开出大门外，其他车辆再按原次序进入车场，每辆停放在车场的车在它离开停车场时必须按它停留的时间长短交纳费用。

试为停车场管理系统实现设计所需的数据结构；完成该管理系统需要哪些基本操作？文字描述算法基本思想。

【参考答案】

以栈模拟停车场，以队列模拟车场外的便道。每一组输入数据包括三个数据项：汽车“到达”或“离去”信息、汽车牌照号码以及到达或离去的时刻。对每一组输入数据进行操作后的输出信息为：若是车辆到达，则输出汽车在停车场内或便道上的停车位置；若是车辆离去，则输出汽车在停车场内停留的时间和应交纳的费用（在便道上停车不收费）。栈以顺序存储结构实现，队列以链表结构实

现。

基本操作：

1) int InitStack(StackCar *S) //初始化栈

2) int InitQueue(QueueCar *Q) //初始化队列

3) int Push(StackCar *S,CarNode *p) //进栈操作

4) int Pop(StackCar S,CarNode (&p)) //出栈操作

5) int EnQueue(QueueCar *Q,CarNode *p) //进队列操作

6) int DeQueue(QueueCar Q,CarNode (&q)) //出队列操作

7) void PriceCal(CarNode *p,int Location) //价格计算

8) void InCarPark(StackCar *Enter,QueueCar *Q) //进停车站

9) void OutCarPark(StackCar *Out,StackCar *Temp,QueueCar *Q)

10) void DisplayStack(StackCar *S) //显示停车站的停车情况

11) void DisplayQueue(QueueCar *Q) //显示便道的停车情况

或主要给出 3）--7）即可。

算法思想：

建立两个栈一个作为停车场，另一个作为临时栈，建立一个队列。使用栈和队列的配合使用，进行停车场的建立，栈以顺序存储结构实现，队列以链表结构实现。首先在栈空的时候，使车辆进栈，保存车辆的进栈信息，以及输入时间和代码，当栈的指针指向栈顶，及大于 stack[MAX+1]时，那么以后的车就先进入便道，当停车场的车有离开的时候，那么在该车以后的车在 p++的引导下，相继开出停车场并进入临时栈，当该车开出的时候，那么刚才出去的车在按照先前的次序依次进栈，再把先前停在便道上的车按次序进栈。定义栈的顺序存储结构和队列的链式存储结构，同时定义栈顶指针 top 和栈底指针 base。然后初始化栈，同时定义在停车时的停车时间 time 和所需要的费用 mony 的关系当车进入的时候栈顶元素+1，在当车进来的时候在+1，直到空间不足，当车进来的时候显示停车场已经停满，要停到便道上，便道元素+1；当停车场的车出来一辆之后，便道上的车才能进入停车场，队列元素减 1，栈元素加 1。而离开的车辆则输出离开的时间，调用计费函数，进行停车费的计算。

3、算法设计

针对本部分的每一道题，要求：

（1） 采用 C 或 C++语言设计数据结构；

（2） 给出算法的基本设计思想；

（3） 根据设计思想，采用 C 或 C++语言描述算法，关键之处给出注释。

（4） 说明你所设计算法的时间复杂度和空间复杂度。

3.1 已知单向链表 L 是一个递增有序表,试写一高效算法,删除表中值大于 min 且小于 max 的 结点(若表中有这样的结点),同时释放被删结点的空间,这里 min 和 max 是两个给定的参数。

【参考答案】

```
struet node {
     datatype data ;
     struct node * next;
     } ListNode ;
typedef ListNode    *LinkList ;
void DeleteList(LinkList L , DataType    min , DataType max )
{ ListNode    *p，*q，*h ;
   p = L->next ; //采用代表头结点的单链表
   while( p&&p->data <= min) //找比 min 大的前一个元素位置
   q=p; p=p->next ;
   p=q ;   //保存这个元素位置
   while(q&&q->data < max)    //找 max 小的最后一个元素位置
   q=q->next ;
   while (p->next! =q)
   { h = p->next;
      p=p 一>next;
      free ( h) ;   //释放空间
   }
    p->next = q;   //把断点链上
 }
```

3.2 一个长度为 L（L≥1）的升序序列 S，处在第 L/2 个位置的数称为 S 的中位数。例如，若序列 S1=(11, 13, 15, 17, 19)，则 S1 的中位数是 15。两个序列的中位数是含它们所有元素的升序序列的中位数。例如，若 S2=(2, 4, 6, 8, 20)，则 S1 和 S2 的中位数是 11。现有两个等长升序序列 A 和 B，试设计一个在时间和空间两方面都尽可能高效的算法，找出两个序列 A 和 B 的中位数。

【答案要点】

（1）给出算法的基本设计思想

分别求两个升序序列 A、B 的中位数，设为 a 和 b。若 a=b，则 a 或 b 即为所求的中位数；否则，舍弃 a、b 中较小者所在序列之较小一半，同时舍弃较大者所在序列之较大一半,要求两次舍弃的元素个数相同。在保留的两个升序序列中，重复上述过程，直到两个序列中均只含一个元素时为止，则较小者即为所求的中位数。

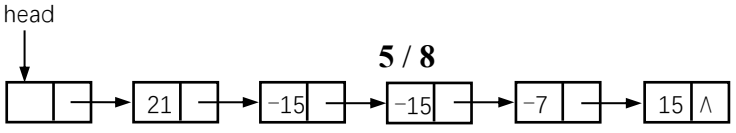
（2）算法实现

```
int M_Search ( int A[], int B[], int n )
```

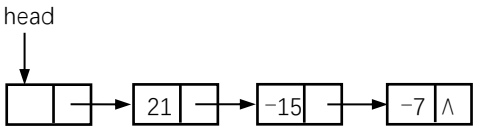
```
{   int start1, end1, mid1, start2, end2, mid2;
    start1 = 0;    end1 = n-1;
    start2 = 0;    end2 = n-1;
    while ( start1 != end1 || start2 != end2 )
    {   mid1 = (start1 + end1) / 2;
        mid2 = (start2 + end2) / 2;
        if ( A[ mid1 ] == B[ mid2 ] )
            return A[ mid1 ];
        if ( A[ mid1 ] < B[ mid2 ] )
        {   // 分别考虑奇数和偶数，保持两个子数组元素个数相等
            if ( ( start1 + end1 ) % 2 == 0 )
            {   // 若元素为奇数个
                start1 = mid1;    // 舍弃 A 中间点以前的部分且保留中间点
                end2 = mid2;      // 舍弃 B 中间点以后的部分且保留中间点
            }
            else
            {  // 若元素为偶数个
                start1 = mid1 + 1;   // 舍弃 A 的前半部分
                end2 = mid2;      // 舍弃 B 的后半部分
            }
        }
        else
        {   if ( ( start1 + end1 ) % 2 == 0 )
            {   // 若元素为奇数个
                end1 = mid1;  //舍弃 A 中间点以后的部分且保留中间点
                start2 = mid2;//舍弃 B 中间点以前的部分且保留中间点
            }
            else
            {  // 若元素为偶数个
                end1 = mid1;      // 舍弃 A 的后半部分
                start2 = mid2+1; // 舍弃 B 的前半部分
            }
        }
    }
    return A[start1] < B[start2] ? A[start1] : B[start2];
}
```

（3）上述所给算法的时间、空间复杂度分别是 O(log2n)和 O(1)。

3.3 用单向链表保存 m 个整数，结点的结构为：(data, next)，且|data|≤n（n 为正整数)。现要求设计一个时间复杂度尽可能高效的算法,对于链表中 data 的绝对值相等的结点，仅保留第一次出现的结点而删除其余绝对值相等的结点。例如，若给定的单链表 head 如下：



则删除结点后的 head 为：



【参考答案】

（1）算法的基本设计思想

算法的核心思想是用空间换时间。使用辅助数组记录链表中已出现的数值，从而只需对链表进行一趟扫描。

因为|data|≤n，故辅助数组 q 的大小为 n+1，各元素的初值均为 0。依次扫描链表中的各结点，同时检查 q[|data|]的值，如果为 0，则保留该结点，并令 q[|data|]=1；否则，将该结点从链表中删除。

（2）使用 C 语言描述的单链表结点的数据类型定义

```
typedef struct node {
    int        data;
    struct node   *next;
} NODE;
typedef NODE *PNODE;
```

（3）算法实现

```
void func( PNODE h, int n )
{   PNODE p=h, r;
    int *q, m;
    q=(int *) malloc(sizeof(int)*(n+1));//申请 n+1 个位置的辅助空间
    for ( int i=0; i<n+1; i++ )            // 数组元素初值置 0
       *(q+i) = 0;
    while ( p->next != NULL )
    {   m = p->next->data>0 ? p->next->data : -p->next->data;
       if ( *(q+m) == 0 )          // 判断该结点的 data 是否已出现过
       {   *(q+m) = 1;             // 首次出现
          p = p->next;             // 保留
       }
       else                        // 重复出现
       {   r = p->next;            // 删除
          p->next = r->next;
          free( r );
       }
    }
    free( q );
}
```

3.4 设有一个双向链表,每个结点中除有 pred、data 和 next 这 3 个域外,还有一个访问频度域 freq,在链表被启用之前,其值均初始化为零。每当在链表进行一次 LocateNode(L,x)运算时,令元素值为 x 的结点中 freq 域的值加 1,并调整表中结点的次序,使其按访问频度的递减序排列,以便使频繁访问的结点总是靠近表头。试写一符合上述要求的 LocateNode 运算的算法。

【参考答案】

数据结构定义 DLIST:

```
struct    NODE  {
        ElementType   data ;
        int freq;
        struct NODE   *next，*pred;
};
typedef   NODE *DLIST;

DList locate ( DList L, ElemType x)
// L 是带头结点的按访问频度递减的双向链表
{ DList  p=L->next,q;  //p 为 L 表的工作指针,q 为 p 的前驱,用于查找插入位置
  while (p&&p-> data! = x)   p=p->next; //查找值为 x 的结点
  if(!p) { printf("不存在所查结点\n" ) ; exit (0) ; }
  else {  p->freq++; //令元素值为 x 的结点的 freq 域加 1
          p->next->pred =p->pred; //将 p 结点从链表上摘下
          p->pred->next=p->next;
          q=p->pred;  //以下查找 p 结点的插入位置
          while (q!= L && q->freq<p->freq) q = q->pred ;
          p->next=q->next; q->next->pred=p; //将 P 结点插入
          p->pred=q; q->next=p;
        }
   return ( p) ;  //返回值为 x 的结点的指针
  }//算法结束
```

在算法中先查找数据 *x*，查找成功时结点的访问频度域增 *1*，最后将该结点按频度递减插入链表中。

T(n)=O(n),S(n)=(1).

3.5 线性表中元素存放在数组 A(1..n)中,元素是整型数。分别写出非递归和递归算法求出数组 A 中的最大和最小元素，分析时间和空间复杂度。

【参考答案】

递归:

```
int MinMaxValue(int A [ ], int n, int *max, int *min)
//一维数组 A 中存放有 n 个整型数，本算法递归地求出其中的最小数
{  if (n>0)
```

```
    {  if(*max < A[n])  *max=A[n];
       if(*min > A[n])  *min=A[n] ;
       MinMaxValue(A, n-1, max, min) ;
    }  //算法结束
```

此问题考查的知识点是递归算法的编写。主要特点是数组的划分及递归结束的条件。本算法函数调用的格式是：*MinMaxValue (arr, n, &max, &min)* ;其中 *arr* 是具有 *n* 个整数的一维数组，*max=-32768* 是最大数的初值，*min=32767* 是最小数的初值。

T(n)=O(n),S(n)=O(1)和 O(n)