第一章　概论

考点一：基本概念

1、从宏观上看，数据、数据元素和数据项反映了数据组织的三个层次。

（1）数据：所有被计算机存储，处理的对象。

（2）数据元素：是数据的基本单位，在程序中作为一个整体加以考虑和处理。又称元素，顶点，结点，记录。

（3）数据项：　数据项组成数据元素，又称字段或域，是数据的不可分割的最小标识单位。

考点二：数据结构

1、数据结构主要研究数据的逻辑结构、数据的存储结构以及数据的基本运算。

2、根据数据元素之间的关系，通常有四类基本的逻辑结构：集合、线性结构、树形结构、图结构。

（1）集合中任何两个结点间没有邻接关系，组织形式松散。

（2）线性结构中结点按逻辑关系依次排列成一条“链”，结点之间一个一个依次相邻接。

（3）树形结构中具有分支、层次特性，形态像自然界中的树，上层的结点可以和下层多个结点相邻接，但下层结点只能和上层的一个结点相邻接。

（4）图结构最复杂，其中任何两个结点都可以邻接。

3、数据的存储结构包括：存储数据元素；数据元素之间的关联方式。

4、四种基本存储方式：

（1）顺序存储方式：所有存储结点存放在一个连续的存储区。利用节点在存储器中的相对位置来表示数据元素之间的逻辑关系。

（2）链式存储方式：每个存储结点不仅含有一个数据元素，还包含一组指针。每个指针指向一个与本结点有逻辑关系的结点，用指针表示数据元素之间的逻辑关系。

（3）索引存储方式

（4）散列存储方式

考点三：算法

1、评价算法好坏的因素：

（1）正确性：能正确地实现预定的功能，满足具体问题的需要。

（2）易读性：易于阅读、理解和交流，便于调试、修改和扩充。

（3）健壮性：即使输入非法数据，算法也能适当地做出反应或进行处理，不会产生预料不到的运行结果。

（4）时空性：指该算法的时间性能(或时间效率)和空间性能(或空间效率)，前者是算法包含的计算量，后者是算法需要的存储量。

2、常见时间复杂性量级：

（1）常数阶：O（1）即算法的时间复杂性与输入规模 n 无关或 n 恒为常数。

（2）对数阶：O（$\log_2 n$）

（3）线性阶：O(n)

（4）多项式阶：O（n^2），常见的多项式阶 O（n^2）和 O（n^2）

（5）指数阶：O（C^n），常见的指数阶 O（2^n）

通常认为指数阶量级的算法实际是不可计算的，而量级低于平方阶的算法是高效率的

3、一个算法对具有相同输入数据量的不同输入数据，时间复杂度可能会不同。通常还可以用最坏时间复杂度和平均时间复杂度来度量算法的性能。

（1）最坏时间复杂性：对相同输入数据量的不同输入数据，算法时间用量的最大值。

（2）平均时间复杂性：对所有相同输入数据量的不同输入数据，算法时间用量的平均值。

4、空间复杂度是对一个算法在运行过程中临时占用存储空间大小的量度。在估算算法空间复杂度时，一般只需要分析辅助变量所占用的空间。辅助变量占用的空间由算法决定，有的需要占用随问题规模 n 增大而增大的临时空间，有的不随问题的规模而改变。

第二章　线性表

考点一：线性表的基本概念

1、线性表是一种线性结构，它是由 n（n≥0）个数据元素组成的有穷序列，数据元素又称为结点。结点个数 n 称为表长。当 n=0 时，线性表不含任何数据元素，称为空表，记为（）。当 n＞0 时，线性表通常表示成（a_1，a_2，…，a_n），a_1 称为起始节点，a_n 称为终端结点。对任意一对相邻结点 a_i 和 a_i+1（1≤i＜n），a_i 称为 a_i+1 的直接前驱，a_i+1 称为 a_i 的直接后继。

2、线性结构的基本特征：若至少含有一个结点，则除起始节点没有直接前驱外，其他结点有且只有一个直接前驱，除终端结点没有直接后继外，其他结点有且只有一个直接后继。

3、线性表的基本运算：

（1）初始化 Initiate（L）：建立一个空表 L=（）。

（2）求表长 Length（L）：返回线性表 L 的长度。

（3）读表元素 Get（L，i）：返回线性表第 i 个元素，当不满足（1≤i≤Length（L））时，返回一特殊值。

（4）定位 Locate（L，X）：查找线性表中数据元素等于 x 的结点序号，若有多个数据元素值与 x 相等，运算结果为这些节点中序号的最小值，若找不到该结点，则运算结果为 0。

（5）插入 Insert(L，X，i)：在线性表 L 的第 i 个位置上插入一个值为 x 的新数据元素，参数 i 的合法取值范围是 1≤i≤n+1。

（6）删除 Delete（L，i）：删除 L 的第 i 个结点 a_i，i 的合法取值范围是 1≤i≤n。

考点二：线性表的顺序存储

1、线性表顺序存储的方法是：将表中的结点依次存放在计算机内存中的一组连续的存储单元中，数据元素在线性表中的邻接关系决定它们在存储空间中的存储位置，即逻辑结构中相邻的结点其存储位置也相邻。

2、用顺序存储实现的线性表称为顺序表，定义顺序表要预分配存储空间。

3、顺序表实现的算法分析：

操作	最坏情况下移动次数	平均移动次数	时间复杂度
插入	n-i+1	n/2	O（n）
删除	n-1	n-1/2	O（n）
定位	/	/	O（ n）
求表长，读表元	/	/	O（1）

考点三：线性表的链接存储——单链表

1、单链表表示法的基本思想：用指针表示结点间的逻辑关系。

2、一个存储结点包含两部分：

data 部分 ：称为数据域，用于存储线性表的一个数据元素。

Next 部分：称为指针域或链域，用于存放一个指针，指向本结点所含数据元素的直接后继所在的结点

3、终端结点的指针：

NULL 称为空指针，不指向任何结点，只起标志作用。

Head 称为头指针变量，指向单链表的第一个结点的，称为头指针。

头指针具有标识单链表的作用，故常用头指针变量来命名单链表。

4、线性表的基本运算在单链表上的实现：

（1）初始化

```
LinkList InitiateLinklist（）                    // 建立一个空的单链表
{    LinkList head;                              //头指针
     head=malloc(sizeof(node));                  动态构建一结点，它是头结点
     head->next=NULL;
     return head;
}
```

变量 head 是链表的头指针，它指向新创建的结点，即头结点。一个空单链表仅有一个头结点，它的指针域为 NULL

（2）求表长

```
Int lengthLinklist(Linklist head)                //求表 head 的长度
{    Node *p=head;                               //p 是工作指针，初始时 p 指向头结点
     int cnt=0;                                  //判断是否为尾结点
     while (p ->next!=NULL)                  //指针移动到下一个结点
     {    p=p->next;
          cnt++;
     }
     return cnt;                                 //返回表长
}
```

（3）读表元素

```
Node * GetLinklist(LinkList head，int i)
//在单链表 head 中查找第 i 个元素结点，若找到，则返回指向该结点的指针；否则返回 NULL
{
     Node * p;                                   //p 是工作指针
          p=head->next;                          //初始时，p 指向首结点
          int c=1;
          while((c<i)&&(p!=NULL))                //当未到第 i 结点且未到尾结点时继续后移
               {p=p->next c++i}
               if(i==c) return p;                     //找到第 i 个结点
          else return NULL;                           //i<1 或 i>n，i 值不合法，查找失败
}
```

注意:p!=NUL 在这里作为判断是否已遍历整个链表的条件，不可遗漏

（4）定位

```
int LocateLinklist(Linklist head，DataTypex)
    //求表 head 中第一个值等于 x 的结点的序号，若不存在这种结点，返回结果为 0
    {
    Node*p=head;                    //P 是工作指针
        p=p->next;                  //初始时 p 指向首结点
        int i=0;                    //1 代表结点的序号，这里置初值为 0
        while(p!=NULL&&p->data!=x)  //访问链表
        {   i++;
            p=p->next;
    }
        if(p!=NULL)     return i+1;
        else return 0;
}
```

（5）删除

```
void InsertLinklist ( LinkList head， DataType x， int i)
    //在表 head 的第 1 个数据元素结点之前插入一个以 x 为值的新结点
    {Node *p，*q;
    if (i==1) q=head;
    else q=GetLinklist(head，i-1);          //找第 1-1 个数据元素结点
    if(q==NULL)                             //第 1-1 个结点不存在
        exit("找不到插入的位置");
    else
        {   p=malloc(sizeof(Node));     p->data=x;      //生成新结点
            p->next=q->next;                //新结点链域指向*q 的后继结点
            q->next=p;                      //修改*q 的链域
    }
}
```

链接操作 p->next=q->next 和 q->next=p 两条语句的顺序不能颠倒，否则结点*q 的链域值（即指向原表第 i 个结点的指针）将丢失。

（6）删除

```
void DeleteLinklist(LinkList head， int i)          //删除表 head 的第 1 个结点
    {   Node * q;
        if(i==1)        q=head;
        else    q=GetLinklist(head，i-1);           //先找待删结点的直接前驱
        if(q!==NULL && q->next!=NULL)       //若直接前驱存在且待删结点存在
        {
        p=q->next;                          //p 指向待删结点
```

```
        q->next=p->next;                    //移出待删结点
        free(p);                            //释放已移出结点 P 的空间
    }
    else exit（“找不到要删除的结点”）;          //结点不存在
```

考点四：循环链表

1、带有头结点的单链表的头指针为 head，判断该单链表为非空的条件是 head->next!=NULL。

2、在带有头结点的循环链表中，尾指针为 rear，判断指针 P 所指结点为首结点的条件是 p==rear->next->next。

3、在带有头结点的循环链表中，头指针为 head，判断指针 p 所指结点为首结点的条件是 p==head->next。

4、非空的单循环链表的头指针为 head，尾指针为 rear，则 rear->next=head。

考点五：双向循环链表

1、双向循环链表的对称性可以用 p=prior->next=p->next->prior

2、在双向循环链表中，设 p 指向待删结点，删除*p 的正确语句为：

（1）p->prior->next=p->next;　　//p 前驱结点的后链指向 p 的后继结点

（2）p->next->prior=p->prior;　　//p 后继结点的前链指向 p 的前驱结点

（3）free（p）；　　//释放*p 空间

3、在 p 所指结点的后面插入一个新结点*t，需要修改四个指针，

（1）t->prior=p;

（2） t->next=p->next;

（3）p->next->prior=t;

（4） p->next=t;