List

Zibin Zheng（郑子彬）

School of Data and Computer Science， SYSU

http://www.inpluslab.com

课程主页：http://inpluslab.sysu.edu.cn/dsa2016/

逻辑结构

数据元素之间的关系是什么？

26个英文字母的字母表：
（A,B,C,…,Z）

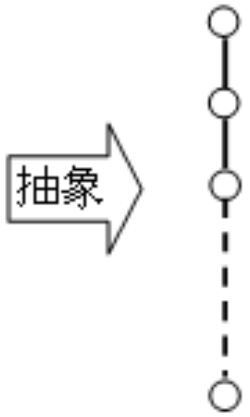
某校从1978年到1983年各种型号的计算机拥有量的变化情况：
（6，17，28，50，92，188）

逻辑结构

数据元素之间的关系是什么？

学　号	姓　名	性　别	出生日期	政治面貌
0001	陆　宇	男	1986/09/02	团员
0002	李　明	男	1985/12/25	党员
0003	汤晓影	女	1986/03/26	团员
⋮	⋮	⋮	⋮	⋮

(a)　学生学籍登记表



(b)　线性结构

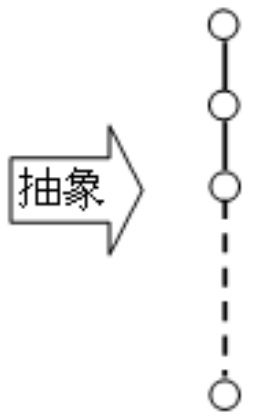
常把数据元素称为**记录（record）**
含有大量记录的线性表称为**文件（file）**

逻辑结构

数据元素之间的关系是什么？

职工号	姓　名	性　别	基本工资	岗位津贴	业绩津贴
000826	王一梅	女	1200	900	600
000235	李　明	男	1800	1200	900
000973	郑　浩	男	800	500	
⋮	⋮		⋮	⋮	⋮

(a)　职工工资表



(b)　线性结构

现实生活中，许多问题抽象出的数据模型是线性表，如何存储这种线性结构并实现插入、删除、查找等基本操作呢？

Definition

线性表的定义

- 线性表：简称表，是n（$n\geq0$）个具有相同类型的数据元素的有限序列。
- 线性表的长度：线性表中数据元素的个数。
- 空表：长度等于零的线性表，记为：$L=(\ )$。
- 非空表记为：$L=(a_1, a_2, \ldots, a_{i-1}, a_i, \ldots, a_n)$

其中，a_i（$1\leq i\leq n$）称为数据元素；
下角标 i 表示该元素在线性表中的位置或序号 。

Characteristics of List

线性表的特性

1. 有限性：线性表中数据元素的个数是有穷的。
2. 相同性：线性表中数据元素的类型是同一的。
3. 顺序性：线性表中相邻的数据元素a_{i-1}和a_i之间存在序偶关系(a_{i-1}, a_i)，即a_{i-1}是a_i的前驱，a_i是a_{i-1}的后继；a_1无前驱，a_n无后继，其它每个元素有且仅有一个前驱和一个后继。

Characteristics of List

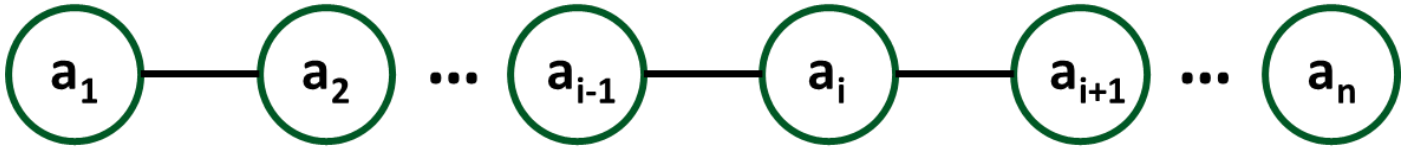
- 线性表（N，r）

（a）结点集N中有一个**唯一**的开始结点，它没有前驱，但有一个唯一的后继；

（b）对于有限集N, 它存在一个**唯一**的终止结点，该结点有一个唯一的前驱而没有后继；

（c）其它的结点皆称为内部结点，每一个内部结点既有一个唯一的前驱，也有一个唯一的后继；

（d）线性表所包含的结点个数称为线性表的长度，它是线性表的一个重要参数；长度为0的线性表称为空表；



Cases

- 星座表是不是线性表？

- 公司组织结构，总经理管几个总监，总监管几个主管，主管管几个员工？

- 班级同学之间的友谊关系？

- 班级同学的名单？

Operations

- LenList(L)：求表L的长度
- GetElem(L, i)：取表L中第i个数据元素
- SearchElem(L, e)：按值查找
- InsertElem(&L, i, e) ：在表L中第i个位置插入新的数据元素e
- DeleteElem(&L, i)：删除表L中第i个数据元素

ADT for List

ADT Sqlist

Data

$D=\{a_i, a_i \in \textbf{ElementType}, i=1, 2, \ldots, n, n\geq 0\}$

$R=\{<a_{i-1}, a_i> | a_{i-1}, a_i \in D, i=1, 2, \ldots, n\}$

Operations

求表的长度

LenList(L)

初始条件：线性表L已存在

操作结果：返回L中数据元素个数

ADT for List

取表L中第i个数据元素

GetElem(L, i)

初始条件：线性表L已存在， $1 \le i \le LenList(L)$

操作结果：返回L中第i个数据元素的值

按值查找

SearchElem(L, e)

初始条件：线性表已存在

操作结果：若e存在表中，返回第一个e的位置，否则返回0

ADT for List

在L中第i个位置插入新的数据元素e

InsertElem(&L, i, e)

初始条件：线性表L已存在， 1≤i≤LenList(L)+1

操作结果：在L中第i个位置插入新的数据元素e，表长加1

删除表中第i个数据元素

DeleteElem(&L, i)

初始条件：线性表已存在且非空，1≤i≤LenList(L)

操作结果：删除L中第i个位置的数据元素，表长减1 。

End ADT

ADT for List

说明:

（1）线性表的基本操作是根据实际应用而定

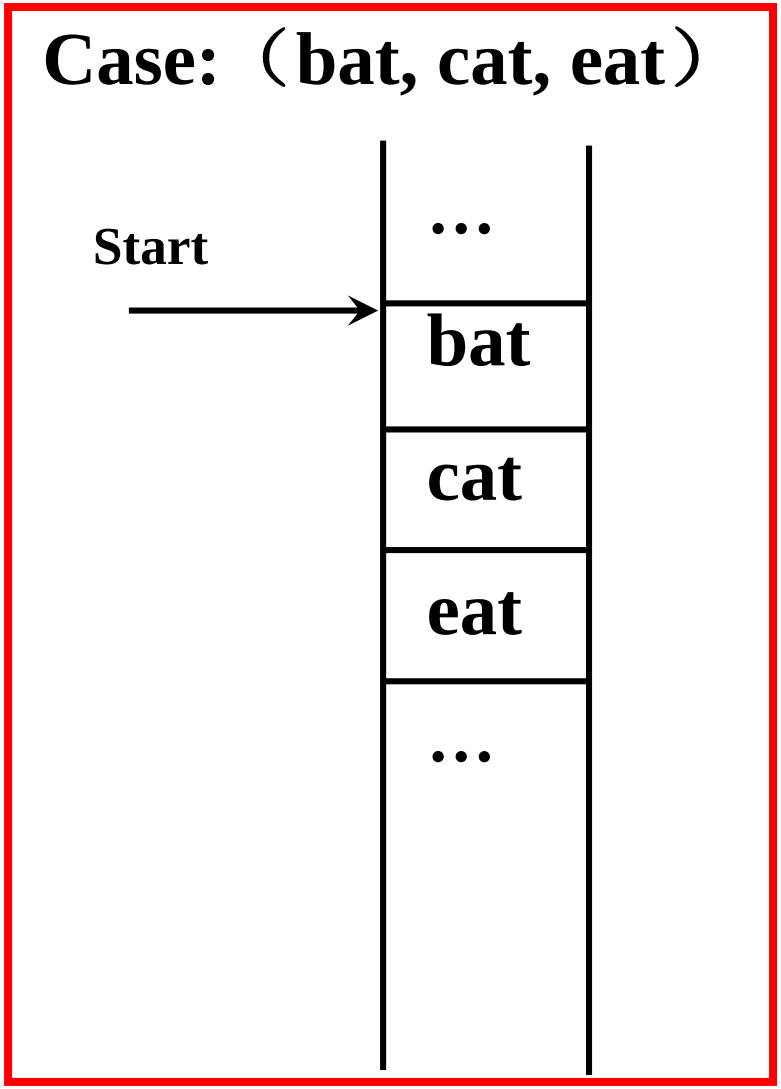
（2）复杂的操作可以通过基本操作的组合来实现

（3）对不同的应用，操作的接口可能不同

Two types of physical structure

（1）Sequence（顺序存储结构）：

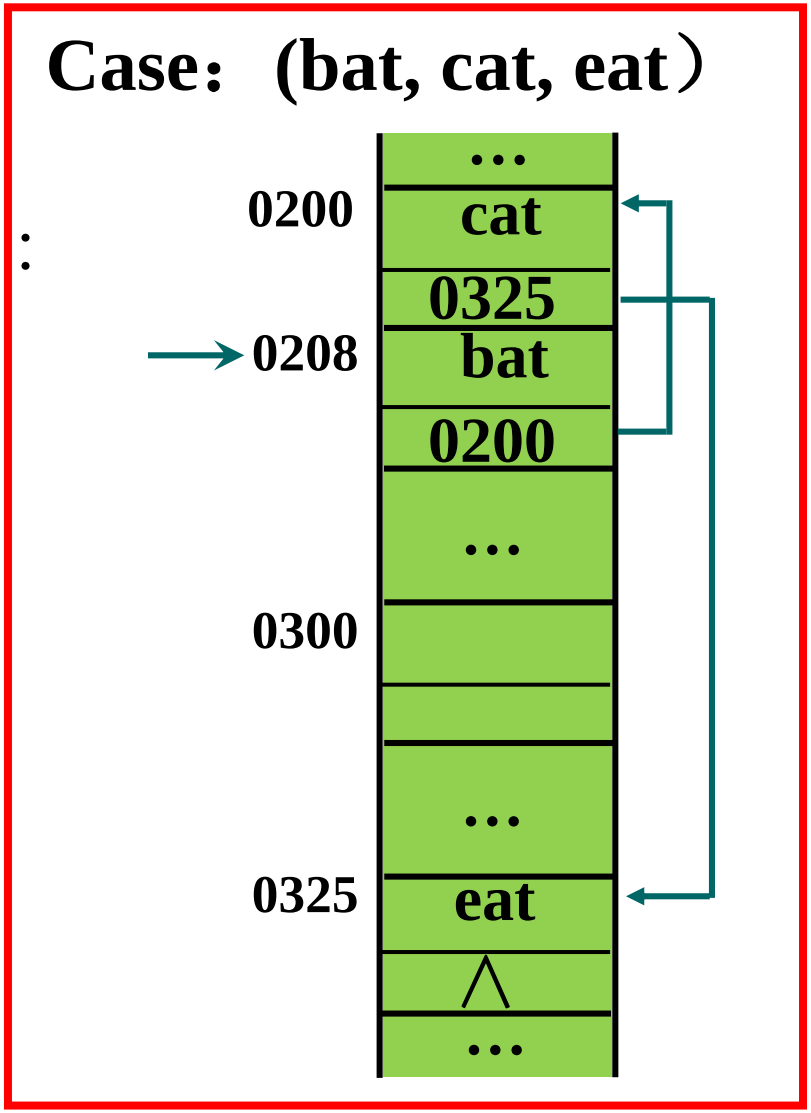
- ✓ Storage cell with continuous location address
- ✓ Logical relationship is described by the function of storage location



Two types of physical structure

（2）Linked（链式存储结构）:

- ✓ Storage cell
- ✓ Logical relationship is described by point



Implementation

- Two main types:

★Sequence List (Array)

  - 线性表的顺序存储结构，指的是用一段地址连续的存储单元依次存储线性表的数据元素

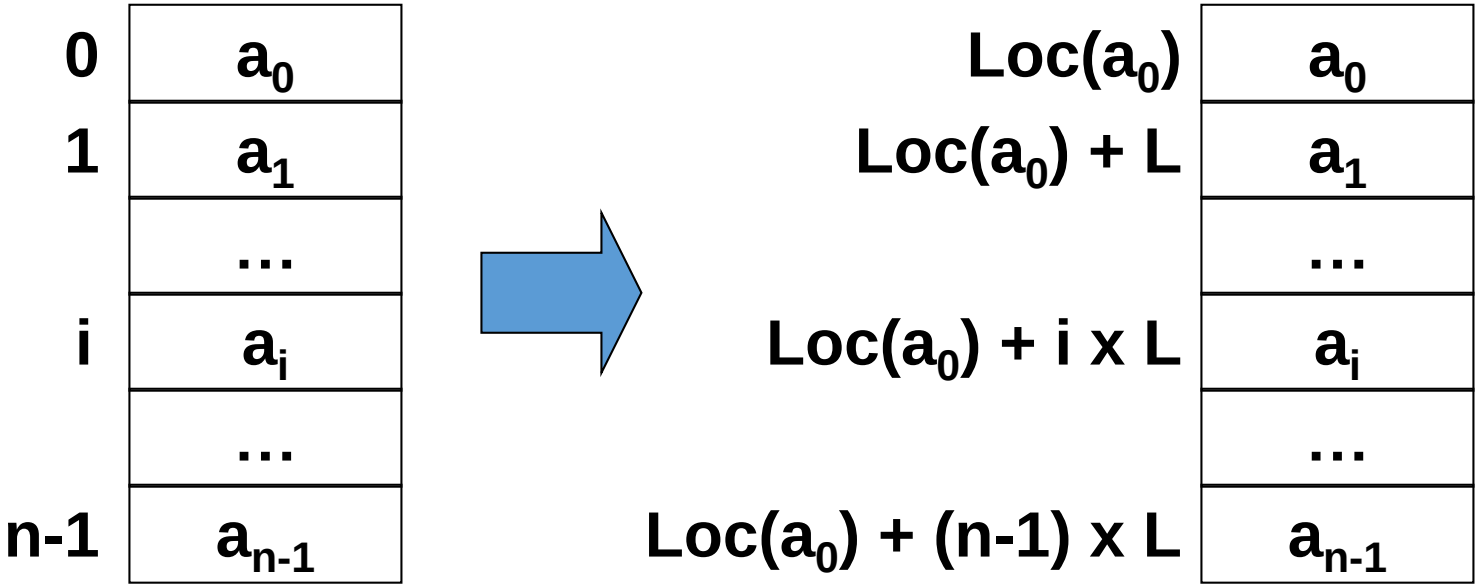
- Linked List

  - 线性表的链式存储结构，指的是用一组任意的存储单元存储线性表的数据元素，这组存储单元可以是连续的，也可以是不连续的。

Sequence List

- **Array implementation**

$$LOC(a_i) = LOC(a_0) + i \times L \qquad 1 <= i <= n$$



Characters

- **How to express the logical structure in the computer**

?

逻辑上相邻的元素，在物理位置上也相邻。

C++ Implementation

```
class arrList : public List<T>  {                    // 顺序表，向量
private:                                             // 线性表的取值类型和取值空间
      T  *aList ;                                    // 私有变量，存储顺序表的实例
      int  maxSize;                                          // 私有变量，顺序表实例的最大长度
      int  curLen;                                   // 私有变量，顺序表实例的当前长度
      int  position;                                 // 私有变量，当前处理位置
public:                                              // 顺序表的运算集
      arrList(const int size) {                      // 创建一个新的顺序表，参数为表实例的最大长度
            maxSize = size;  aList = new T[maxSize]; curLen = position = 0;
      }
      ~arrList() {                                   // 析构函数，用于消除该表实例
            delete [] aList;
      }
      void clear() {                                 // 将顺序表存储的内容清除，成为空表
            delete [] aList; curLen = position = 0;
            aList = new T[maxSize];
      }}
      int length();                                          // 返回此顺序表的当前实际长度
      bool append(const T value);                    // 在表尾添加一个元素value，表的长度增1
      bool insert(const int p, const T value);  // 在位置p上插入一个元素value，表的长度增1
      bool delete(const int p);                      // 删除位置p上的元素，表的长度减 1
      bool setValue(const int p, const T value);             // 用value修改位置p的元素值
      bool getValue(const int p, T& value);      // 把位置p的元素值返回到变量value中
      bool getPos(int & p. const T value);       // 查找值为value的元素，并返回第1次出现的位置
};
```

Operations

- 插入元素运算
  - InsertElem(&L, i, e)
- 删除元素运算
  - DeleteElem(&L, i)
- 查找元素运算
  - SearchElem(L, e)　//按值查找
  - GetElem(L, i)　　//按位置查找

Insert element e at location i

InsertElem(&L, i, e)

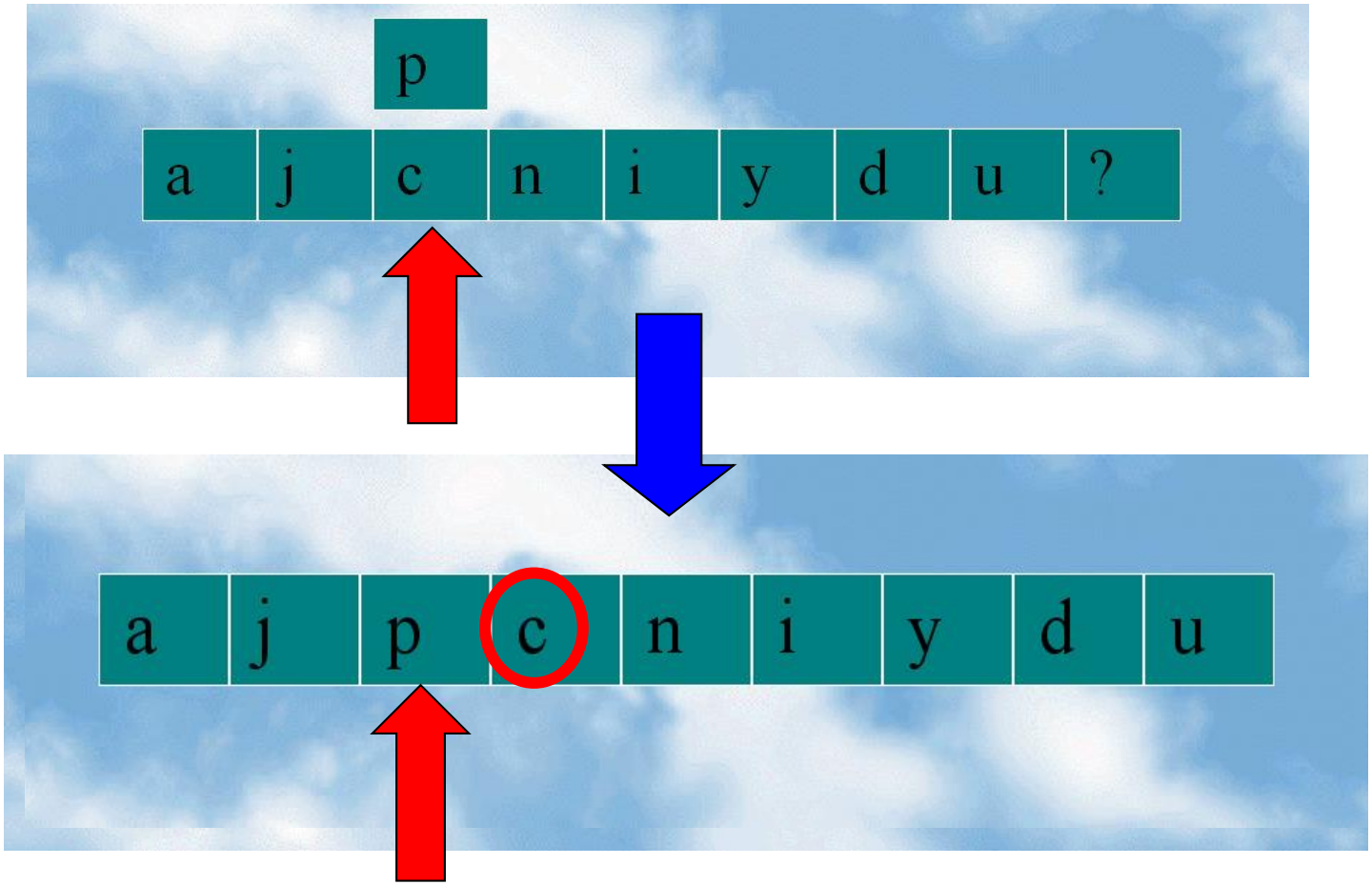
Input: **i: location, e : ElementType;**

Pre-condition: **1≤i≤L.length+1**

Post-condition: **Insert element e at location i and add 1 to Length of L**

Insert element e at location i

i = 3, e = 'P'



Insert element e at location i

```
int InsertElem(sqlist *L, int i, DataType e)
{ int j;
```

Pre-condition

Move element

```
}
```

Insert element e at location i

```
int InsertElem(sqlist *L, int i, DataType e)
{ int j;
  if (((*L).length)>=maxsize)
    { printf("overflow\n"); return NULL;} \\溢出
  else
    if((i<1)||(i>((*L).length)＋1)
      { printf("error\n"); return NULL;}  \\非法位置
```

Move element

```
}
```

Insert element e at location i

```
int InsertElem(sqlist *L, int i, DataType e)
{ int j;
  if (((*L).length)>=maxsize)
    { printf("overflow\n"); return NULL;} \\溢出
  else
    if((i<1)||(i>((*L).length)＋1)
      { printf("error\n"); return NULL;}  \\非法位置
    else
      {   for (j=(*L).length; j>=i; j--)
            (*L).data[j+1]＝(*L).data[j];
          (*L).data[i]＝e;
          (*L).length＝(*L).length +1;
        }
    return(1);
 }
```

Delete element at location i

DeleteElem(&L, i)

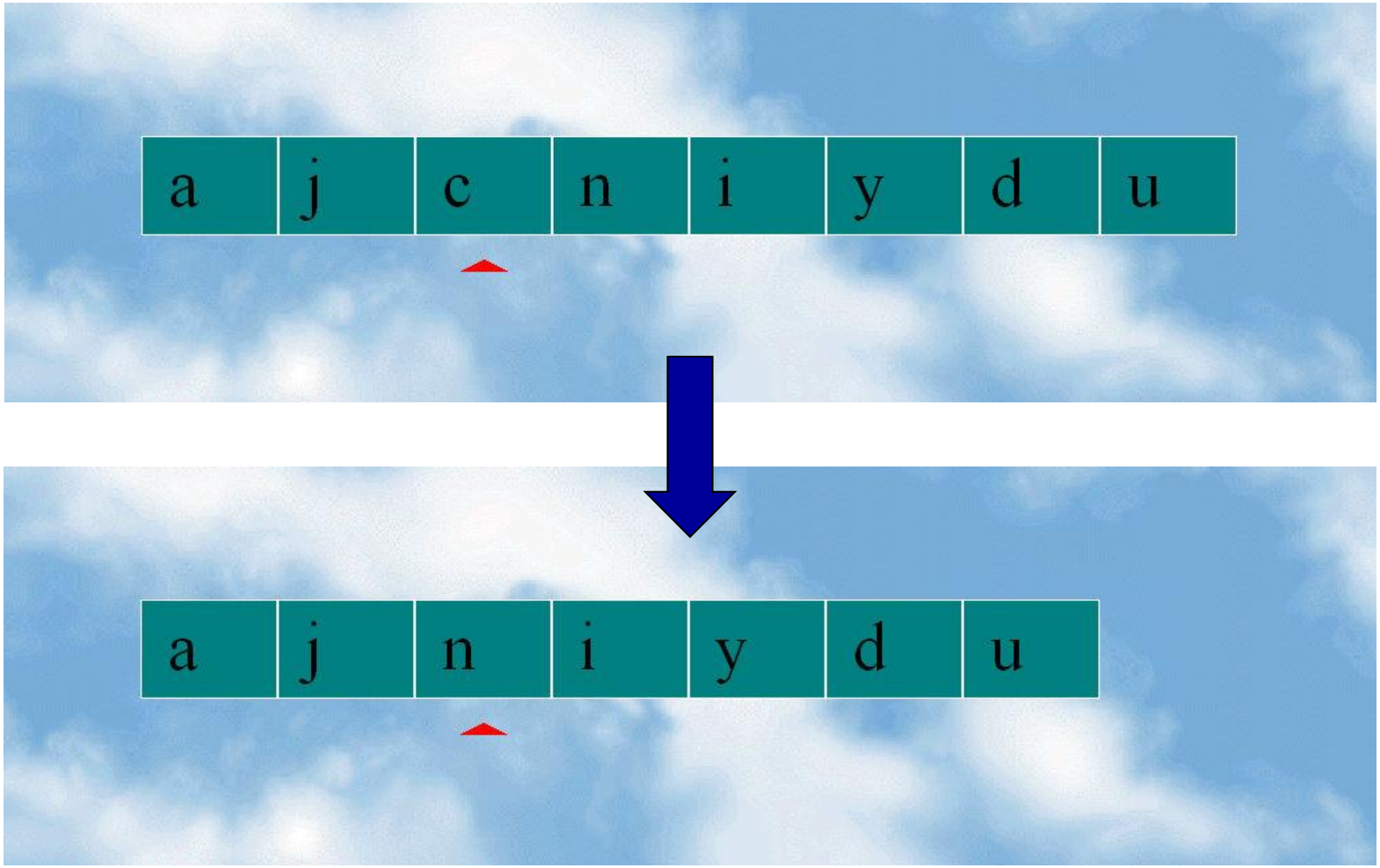
Input：location i；

Pre-condition：1≤i≤L.length

Post-condition：delete the element at location i.

Delete element at location i

i = 3



Delete element at location i

```
int  DeleteElem(sqlist *L, int i)
{ int j;
```

Pre-condition
Move element
Post-condition

```
}
```

Delete element at location i

```
int  DeleteElem(sqlist *L, int i)
{ int j;
   if ((i<1)||(i>(*L).length))
     { printf("error\n"); return NULL;}  \\非法位置
  else
    {   for(j=i+1; j<=(*L).length; j++)
            (*L).data[j-1]＝(*L).data[j];
        (*L).length--;              \\表长减1
    }
  return(1);
}
```

Analysis of operations (Insert and delete)

- Relate to the list length and location.
- **Insertion：**
  - **最坏：i=1，移动次数为n**
  - **最好：i=表长+1，移动次数为0**
  - **平均：等概率情况下，平均移动次数n/2**
- **Deletion：**
  - **最坏：i=1，移动次数为n-1**
  - **最好：i=表长，移动次数为0**
  - **平均：等概率情况下，平均移动次数(n-1)/2**

Operations

- 插入元素运算
  - InsertElem(&L, i, e)
- 删除元素运算
  - DeleteElem(&L, i)
- 查找元素运算
  - SearchElem(L, e)　//按值查找
  - GetElem(L, i)　　//按位置查找

按值查找

- 查找元素e

SearchElem(L, e)

Input: element e;

Output: location of element e;

Pre-condition: L is not empty;

Post-condition: If e is in List L then return location i.

按值查找

```
int SearchElement (SqList L, DataType e)
{ i=1;
  while ( i<=L.length && e != L.data[i-1])
        ++i;
  if (i<=L.length) return i;
        else return 0;
}
```

按位置查找

- 查找位置 i

GetElem(L, i)

Input: location i;

Output: element e;

Pre-condition: 1≤i≤L.length

Post-condition: return the element at location i.

按位置查找

```
DataType GetElem(SqList L, int i)
{
  If ((i<1)||(i>((*L).length))
    { printf(“error\n”);
     return NULL;}  \\非法位置
   return (*L).data[i-1];
 }
```

Analysis of implementation in array

1）优点

- 顺序表的结构简单
- 顺序表的存储效率高，是紧凑结构,无须为表示节点间的逻辑关系而增加额外的存储空间
- 顺序表是一个随机存储结构（直接存取结构）

2）缺点

- 在顺序表中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。

课堂作业（**请写上学号，姓名**）

1. 写出下面2个程序段的时间复杂度

在下面的程序段中，对 x 的赋值语句的频度为

```
FOR i:=1  TO  n  DO
    FOR j:=1  TO  n  DO
      x:=x+1;
```

程序段 FOR i:=n-1 DOWNTO 1 DO

```
    FOR j:=1 TOi DO
        IF A[j]>A[j+1]
            THEN  A[j]与 A[j+1]对换；
```

其中 n 为正整数，则最后一行的语句频度在最坏情况下是

2. 什么是逻辑结构？什么是存储结构？它们的关系是什么？

谢谢！