数 据 结 构
DATA STRUCTURE

软件与通信工程学院　郭美
18075531998

课程学习指导

课程特点： 内容抽象、概念性强、内容灵活、不易掌握

- 1. 提前预习、认真听课、按时完成书面及上机作业
- 2. 注意先修课程的知识准备

  离散数学、C语言
- 3. 注意循序渐进

  基本概念、基本思想、基本步骤、算法设计
- 4. 注意培养算法设计的能力

  理解所讲算法、对此多做思考：若问题要求不同，应如何选择数据结构，设计有效的算法

考核方式

期末成绩：

- 平时成绩： 20%

  作业、小测验、实验

  课堂纪律、回答问题

  无故迟到：－2/次

  无故旷课：－5/次

  缺交作业：－3/次

- 实验成绩： 10%

  出勤；实验完成情况

- 考试成绩： 70%（闭卷笔试）

第1章 绪论

教学目标

- 1. 了解数据结构研究的主要内容
- 2. 掌握数据结构中涉及的基本概念
- 3. 掌握算法的时间、空间复杂度及其分析的简易方法

教学内容

1.1 数据结构的研究内容

1.2 基本概念和术语

1.3 抽象数据类型的表示与实现

1.4 算法与算法分析

1.1 数据结构的研究内容

📖 电子计算机的主要用途：

☞早期：主要用于数值计算

（1）从具体问题抽象出一个适当的数学模型；
（2）设计解此数学模型的算法；
（3）编程，测试、调整直至得到最终解答。

☞后来：处理逐渐扩大到非数值计算领域，能处理多种复杂的具有一定结构关系的数据

数据结构是一门研究非数值计算的程序设计问题中的操作对象，以及它们之间的关系和操作等相关问题的学科。

应用举例1——学籍档案管理

假设一个学籍档案管理系统应包含如下表所示的学生信息。

学生基本情况

学　号	姓　名	性　别	出生年月	……
99070101	李　军	男	80．12	……
99070102	王颜霞	女	81．2	……
99070103	孙　涛	男	80．9	……
99070104	单晓宏	男	81．3	……
……	……	……	……	……

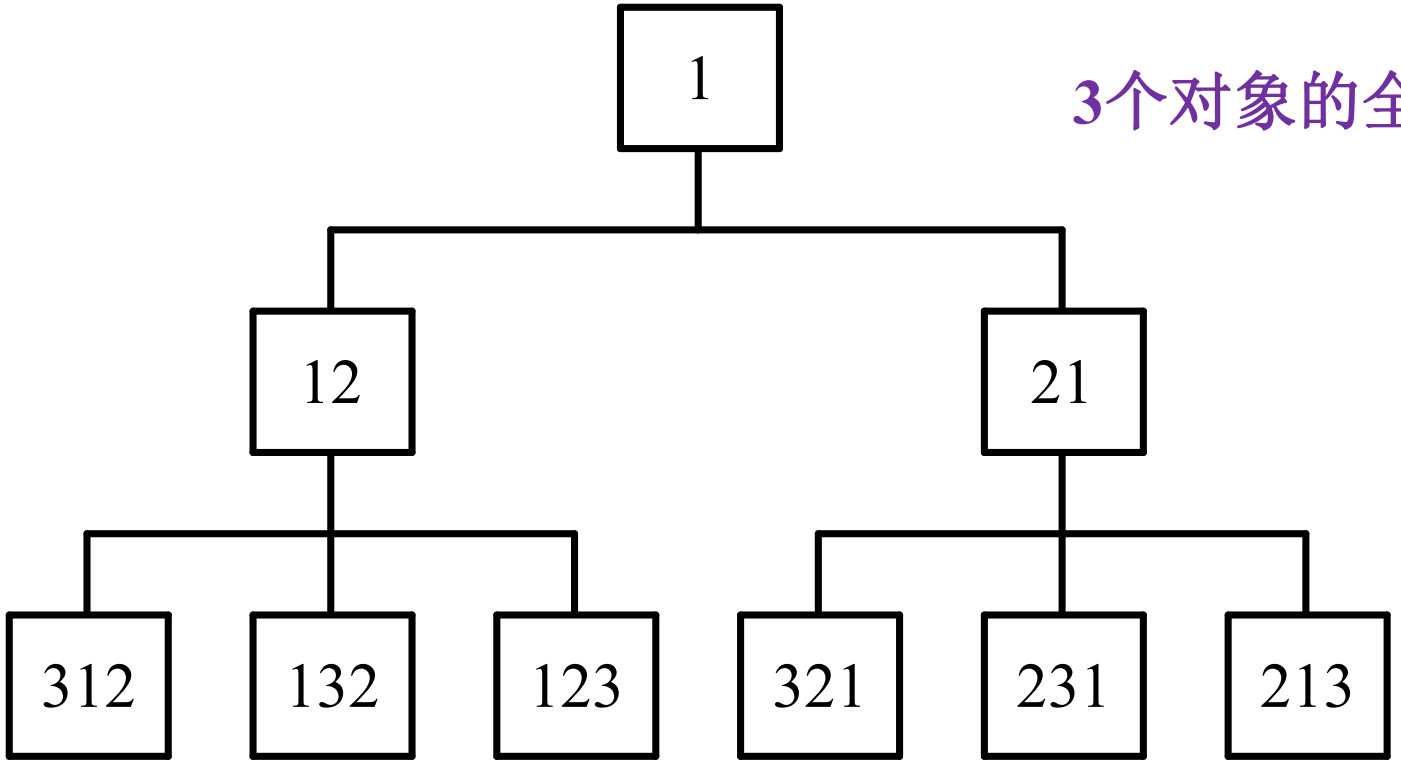
- 应用举例1——学籍档案管理

特点：

- 每个学生的信息占据一行，所有学生的信息按学号顺序依次排列构成一张表格；
- 表中每个学生的信息依据学号的大小存在着一种前后关系，这就是我们所说的线性结构；
- 对它的操作通常是插入某个学生的信息，删除某个学生的信息，更新某个学生的信息，按条件检索某个学生的信息等等。

- 应用举例2——输出n个对象的全排列

输出n个对象的全排列可以使用下图所示的形式描述。



3个对象的全排列过程

- 应用举例2——输出n个对象的全排列

特点：

- 在求解过程中，所处理的数据之间具有层次关系，这是我们所说的树形结构；
- 对它的操作有：建立树形结构，输出最低层结点内容等等。

应用举例3——制定教学计划

在制定教学计划时，需要考虑各门课程的开设顺序。有些课程需要先导课程，有些课程则不需要，而有些课程又是其他课程的先导课程。比如，计算机专业课程的开设情况如下表所示：

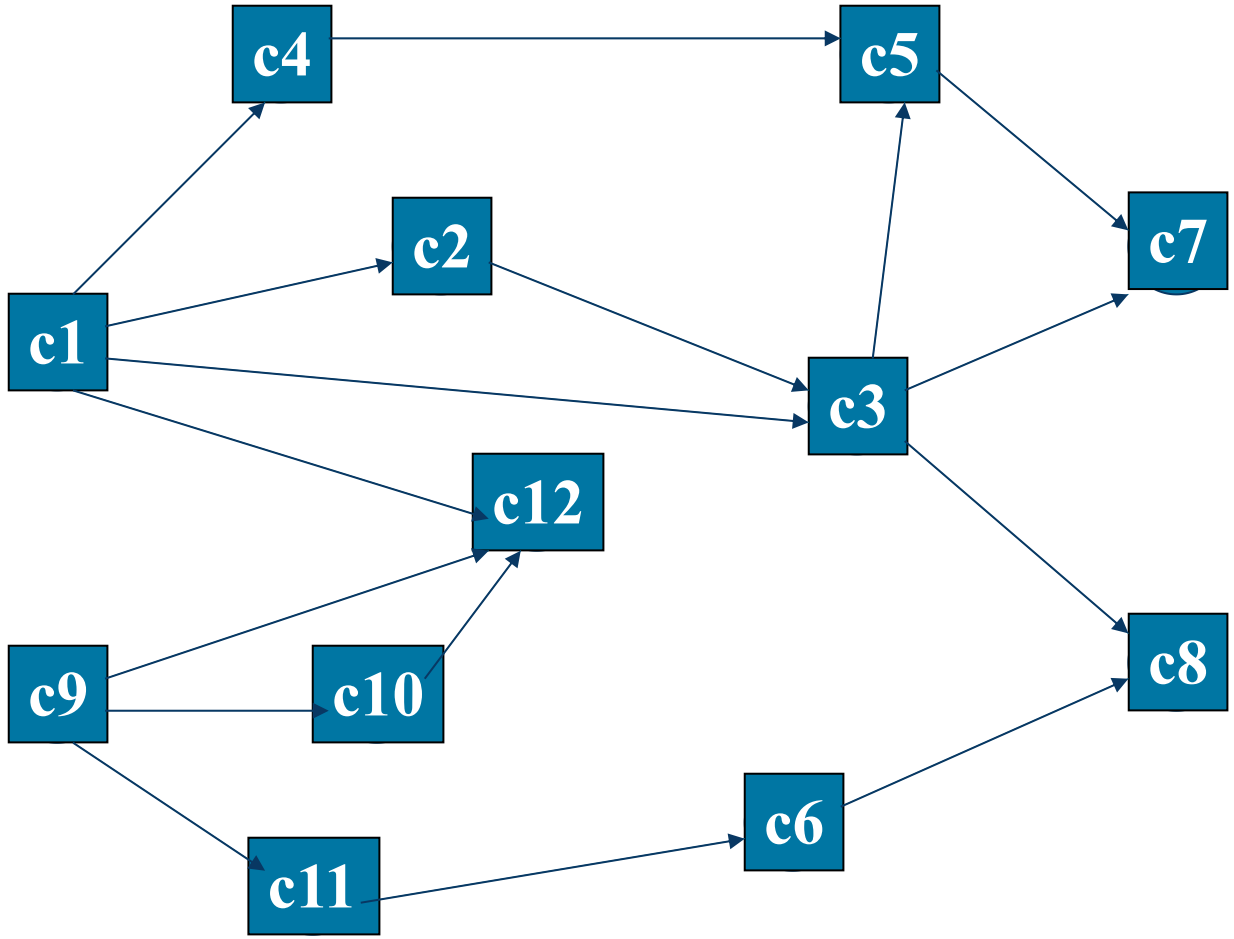
应用举例3——制定教学计划

计算机专业学生的必修课程

课程编号	课程名称	需要的先导课程编号
C1	程序设计基础	无
C2	离散数学	C1
C3	数据结构	C1，C2
C4	汇编语言	C1
C5	算法分析与设计	C3，C4
C6	计算机组成原理	C11
C7	编译原理	C5，C3
C8	操作系统	C3，C6
C9	高等数学	无
C10	线性代数	C9
C11	普通物理	C9
C12	数值分析	C9，C10，C1

- 应用举例3——制定教学计划

课程先后关系的图形描述形式：



- 应用举例3——学籍档案管理

特点：

- 课程之间的先后关系用图结构描述；
- 通过实施创建图结构，按要求将图结构中的顶点进行线性排序。

结论： 计算机的操作对象的关系更加复杂，操作形式不再是单纯的数值计算，而更多地是对这些具有一定关系的数据进行组织管理，我们将此称为**非数值性处理**。要使计算机能够更有效地进行这些非数值性处理，就必须弄清楚这些操作对象的特点，它们在计算机中的表示方式以及各个操作的具体实现手段。

1.1 数据结构的研究内容

📖1968年，高德纳（Donald Ervin Knuth）教授在其所写的《计算机程序设计技术》第一卷《基本算法中》较系统的阐述了数据的逻辑结构和存储结构及其操作，开创了数据结构的课程体系。

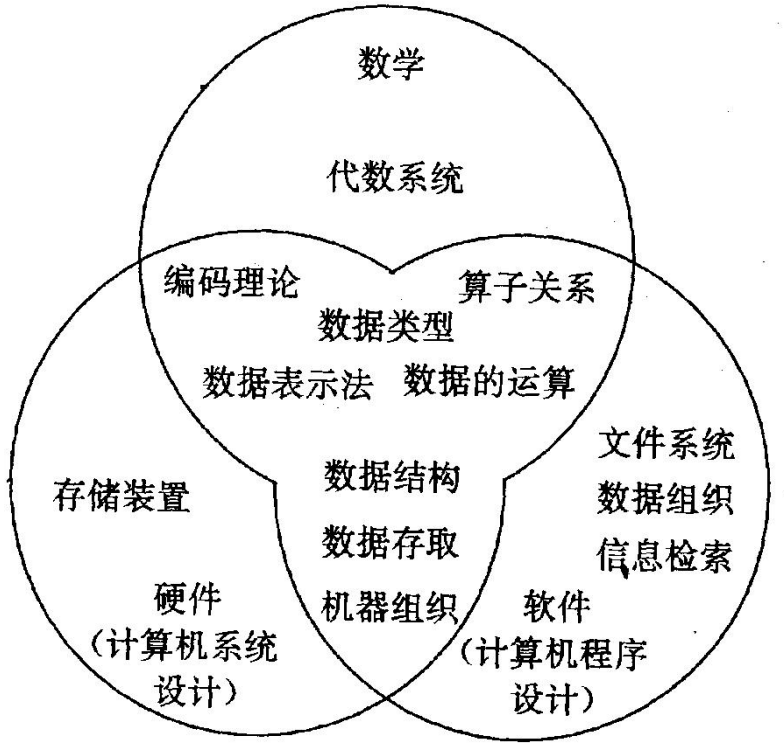
📖尼古拉斯.沃思（Niklaus Wirth)教授提出

程序=算法+数据结构

1.1 数据结构的研究内容

- 《数据结构》课程介绍

介于数学、计算机硬件和计算机软件三者之间的一门核心课程。



1.2 基本概念和术语

- 1、数据（Data）—所有能输入到计算机中去，能被计算机识别并处理的，描述客观事物的符号。
  - 数值型数据
  - 非数值型据（多媒体信息处理）
- 2、数据元素（Data Element）—数据的基本单位，也称结点（node）或记录（record）。
- 3、数据项（Data Item）—有独立含义的数据最小单位，也称域（field）。

三者之间的关系：数据 > 数据元素 > 数据项

例：学生表 > 个人记录 > 学号、姓名……

1.2 基本概念和术语

➢ 4、数据对象（Data Object）—相同特性数据元素的集合，是数据的一个子集。

- 整数数据对象

$$N = \{ 0, \pm 1, \pm 2, \cdots \}$$

- 学生数据对象

学生记录的集合

四者之间的关系：

数据 > 数据对象 > 数据元素 > 数据项

1.2 基本概念和术语

➢ 5、数据结构（Data Structure)—是相互之间存在一种或多种特定关系的数据元素的集合。

📖 数据结构的两个层次：

📖 逻辑结构---

数据元素间抽象化的相互关系，与数据的存储无关，独立于计算机，它是从具体问题抽象出来的数学模型。

📖 存储结构（物理结构）----

数据元素及其关系在计算机存储器中的存储方式。

- 逻辑结构

划分方法一

（1）线性结构

有且仅有一个开始和一个终端结点，并且所有结点都最多只有一个直接前趋和一个后继。

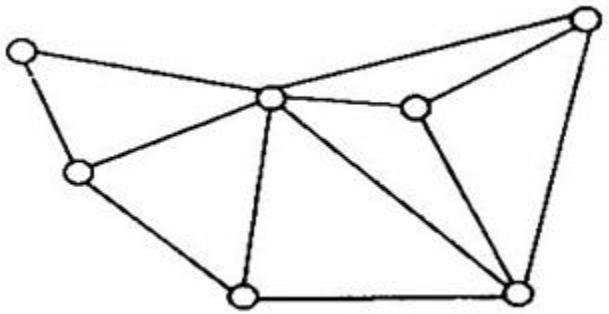
例如：线性表、栈、队列、串

（2）非线性结构

一个结点可能有多个直接前趋和直接后继。

例如：树、图

- 逻辑结构

图 

划分方法二

（1）集合----数据元素间除“同属于一个集合”外，无其它关系。

（2）线性结构----一对一（1:1），如线性表、栈、队列。

（3）树形结构----一对多（1:n），如树。

（4）图形结构----多对多（m:n），如图。

1.2 基本概念和术语

➢ 5、数据结构（Data Structure)—是相互之间存在一种或多种特定关系的数据元素的集合。

📖 数据结构的两个层次：

📖 逻辑结构---

数据元素间抽象化的相互关系，与数据的存储无关，独立于计算机，它是从具体问题抽象出来的数学模型。

📖 存储结构（物理结构）----

数据元素及其关系在计算机存储器中的存储方式。

存储结构

亦称**物理结构**，是数据的逻辑结构在计算机存储器内的表示（或映像）。它依赖于计算机。

（1）顺序存储结构

借助元素在存储器中的**相对位置**来表示数据元素间的逻辑关系。

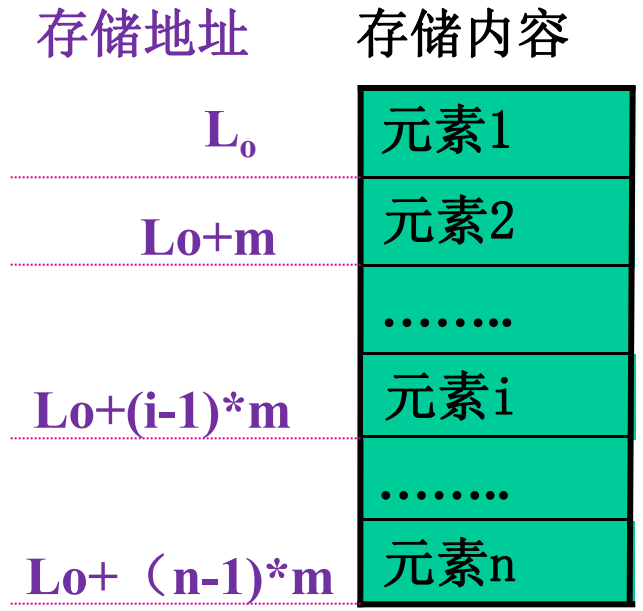
（2）链式存储结构

借助指示元素存储地址的**指针**表示数据元素间的逻辑关系。

顺序存储结构

常用于线性数据结构，将逻辑上相邻的数据元素存储在物理上相邻的存储单元里。

顺序存储



Loc(元素i)=Lo+（i-1)*m

存储结构

亦称**物理结构**，是数据的逻辑结构在计算机存储器内的表示（或映像）。它依赖于计算机。

（1）顺序存储结构

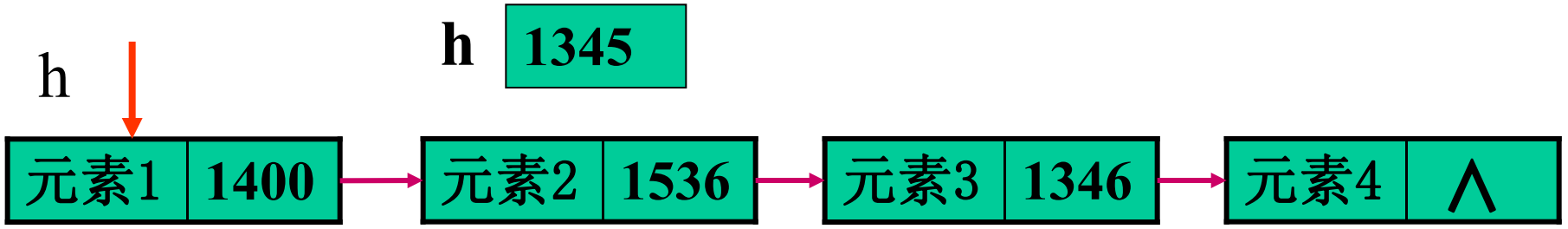
借助元素在存储器中的**相对位置**来表示数据元素间的逻辑关系。

（2）链式存储结构

借助指示元素存储地址的**指针**表示数据元素间的逻辑关系。

链式存储结构

把数据元素放在任意的存储单元。用一个指针存放数据元素的地址。



存储地址	存储内容	指针
1345	元素1	1400
1346	元素4	∧
…….	……..	…….
1400	元素2	1536
…….	……..	…….
1536	元素3	1346

1.2 基本概念和术语

- 6、数据类型（Data Type）—是一组性质相同的值的集合，以及定义于这个集合上的一组运算的总称。

> 在一种程序设计语言中，变量所具有的数据种类。

- FORTRAN语言：

  整型、实型、和复数型

- C语言：

  基本数据类型： char int float double void

  构造数据类型：数组、结构体、共用体、文件

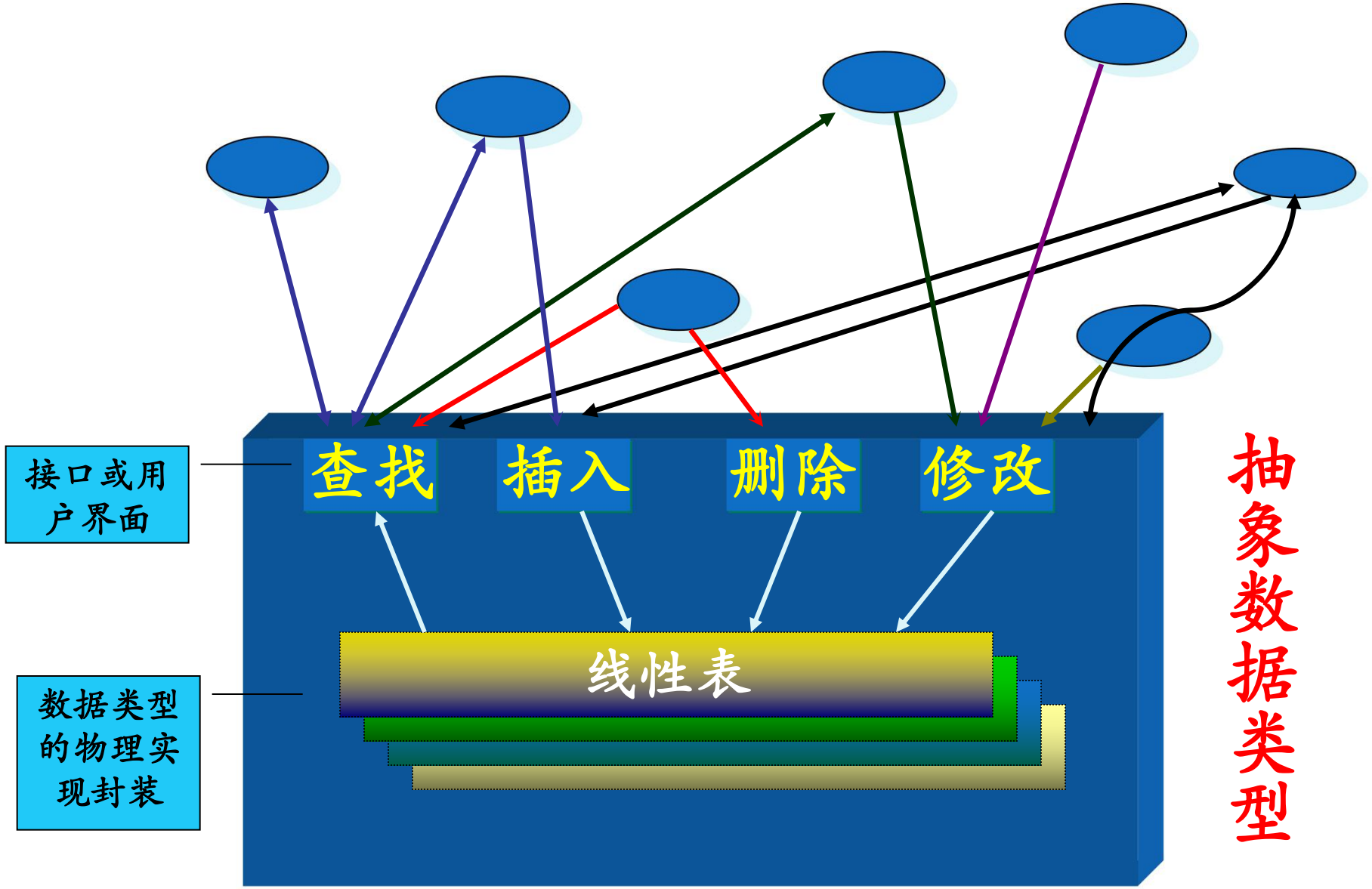
1.2 基本概念和术语

➢ 7、抽象数据类型（Abstract Data Type , ADT ）

—由用户定义，用以表示应用问题的 数据模型 。它由基本的数据类型构 成，并包括一组相关的服务（或称操作）。

抽象数据类型与数据类型实质上是一个概念，但其特征是使用与实现分离，实行封装和信息隐蔽（独立于计算机）。

信息隐蔽和数据封装，使用与实现相分离



抽象数据类型可以用以下的三元组来表示：

ADT = （D，S，P）

数据对象　D上的关系集　D上的操作集

ADT常用定义格式

```
ADT抽象数据类型名{
        数据对象：<数据对象的定义>
        数据关系：<数据关系的定义>
        基本操作：<基本操作的定义>
} ADT抽象数据类型名
```

1.3 抽象数据类型的表示与实现

- 1、可以通过固有的数据类型（如整型、实型、字符型等）来表示和实现。
- 2、有些类似C语言中的结构（struct）类型，但增加了相关的操作。

教材中用的是类C语言（介于伪码和C语言之间）作为描述工具

- 3、上机时要用具体语言实现，如C或C++等。

- 类C语言

（1）预定义常量及类型

借助元素在存储器中的相对位置来表示数据元素间的逻辑关系。

```
//函数结果状态代码
#define OK  1
#define ERROR  0
#define OVERFLOW  -2


// Status是函数返回值类型，其值是函数结果状态
typedef  int  Status;
```

- 类C语言

(2) 数据元素被约定为ElemType 类型，

用户需要根据具体情况，自行定义该数据类型。

(3) 算法描述为以下的函数形式：

```
函数类型 函数名（函数参数表）
{
        语句序列;
}
```

(4) 内存的动态分配与释放

使用new和delete动态分配和释放内存空间

分配空间　指针变量=new数据类型;

释放空间　delete指针变量;

- 类C语言

(5) 赋值语句

(6) 选择语句

(7) 循环语句

(8) 使用的结束语句形式有：

函数结束语句 return

循环结束语句 break;

异常结束语句 exit（异常代码）；

- 类C语言

(9) 输入输出语句的形式有:

输入语句　cin（scanf( )）

输出语句　cout (printf( ))

(10) 扩展函数有:

求最大值 max

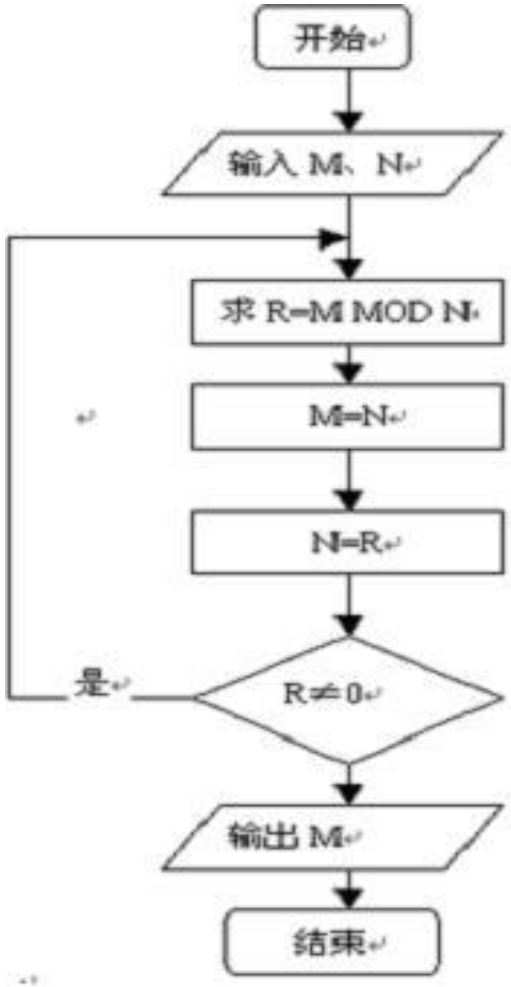
求最小值 min

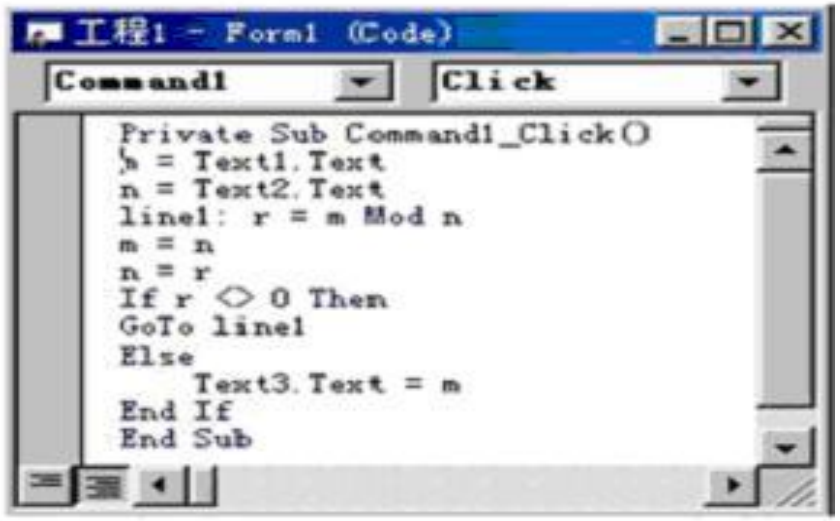
1.4 算法和算法分析

- 1、算法（Algorithm）—是解决特定问题求解步骤的描述，在计算机中表现为指令的有限序列，并且每条指令表示一个或多个操作。
  - 对于给定的问题，可以有多种算法来解决。

- 2、算法的描述
  - 自然语言
  - 流程图
  - 程序设计语言
  - 伪码

欧几里德算法：求两个自然数的最大公约数





1.4 算法和算法分析

➢ 3、算法的特性

- 输入　有0个或多个输入。
- 输出　有一个或多个输出(处理结果)
- 确定性　每步定义都是确切、无歧义的
- 有穷性　算法应在执行有穷步后结束
- 可行性　每一步运算都能够通过执行有效次数完成。

1.4 算法和算法分析

➢ 4、算法的评价

- **正确性**　算法至少应该具有输入、输出和加工处理无歧义性、能正确反映问题的需求、能够得到问题的正确答案。
- **可读性**　算法要便于阅读，有助于人们对算法的理解。
- **健壮性**　当输入非法数据时，也能正常作出反应和处理。
- **高效性**　对相同规模的问题，运行时间短、占用空间少。（时间代价和空间代价）

1.4 算法和算法分析

➢ 5、算法效率的度量

◆ 事后统计方法

主要是通过设计好的测试程序和数据，利用计算机计时器对不同算法编制的程序的运行时间进行比较，从而确定算法效率（时间效率）的高低。

缺陷：必须先运行依据算法编制的程序；时间的比较依赖计算机硬件和软件等因素；算法的测试数据设计困难。

基于事后统计方法的诸多缺陷，一般不予采纳。

1.4 算法和算法分析

➢ 5、算法效率的度量

◆ 事前分析估算方法

在计算机程序编制前，依据统计方法对算法进行估算。

依赖因素：

- ✓ 算法采用的策略、方法；（算法好坏的根本）
- ✓ 编译产生的代码质量；（由软件来支持）
- ✓ 问题的输入规模；（问题规模n）
- ✓ 机器执行指令的速度。（依赖于硬件性能）

比较两种求和的算法：

（1）第一种算法

```
int i,num=0,n=100;      //执行1次
for(i=1;i<=n;i++)       //执行n+1次
    sum=sum+i;          //执行n次
printf("%d",sum);       //执行1次
```

（2）第二种算法

```
int sum=0,n=100;        //执行1次
sum=(1+n)*n/2;          //执行1次
printf("%d",sum);       //执行1次
```

Back

算法的渐进时间复杂度

算法中基本语句重复执行的次数是问题规模n的某个函数f(n)，算法的时间量度记作：

$$T(n)=O(f(n))$$

- 算法中重复执行次数和算法的执行时间成正比的语句
- 对算法运行时间的贡献最大

n越大算法的执行时间越长

- *排序：n为记录数*
- *矩阵：n为矩阵的阶数*
- *多项式：n为多项式的项数*
- *集合：n为元素个数*
- *树：n为树的结点个数*
- *图：n为图的顶点数或边数*

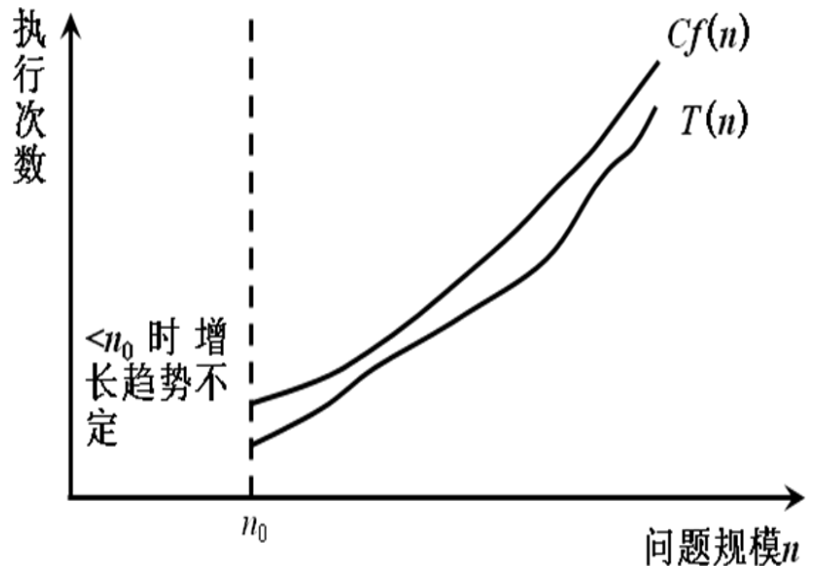
算法的渐进时间复杂度

算法中基本语句重复执行的次数是问题规模n的某个函数f(n)，算法的时间量度记作：

$$T(n)=O(f(n))$$

表示随着n的增大，算法执行的时间的增长率和f(n)的增长率相同，称渐近时间复杂度。

数学符号“O”的定义为：
若$T(n)$和$f(n)$是定义在正整数集合上的两个函数，则
$T(n)=O(f(n))$表示存在正的常数C和n_0，使得当$n \geqslant n_0$时都满足$0 \leqslant T(n) \leqslant Cf(n)$。



分析算法时间复杂度的基本方法

- 找出**语句频度最大**的那条语句作为**基本语句**
- 计算**基本语句**的频度得到问题规模*n*的某个函数f(n)
- 取其数量级用符号“O”表示（称之为大O记法）

```
x = 0;  y = 0;
for ( int k = 0; k < n; k ++ )
    x ++;
for ( int i = 0; i < n; i++ )
  for ( int j = 0; j < n; j++ )
    y ++;
```

$T(n) = O(n^2)$

$f(n)=n^2$

O（ ）为渐近符号，称之为大O记法。

推导大O阶方法

1、用常数1取代运行时间中所有的加法常数；

2、在修改后的运行次数函数中，只保留最高阶项；

3、如果最高阶项存在且不是1，则去除与这个项相乘的常数。

推导大O阶方法

➢ 1、常数阶：O(1)

顺序结构、单纯的分支结构（不包含在循环结构中）的时间复杂度。

例1：

```
int sum=0,n=100;          /*执行一次*/
sum=(1+n)*n/2;            /*执行一次*/
printf("%d",sum);         /*执行一次*/
```

以上算法的运行次数函数是**f(n)=3**，根据推导大O阶的方法，该算法时间复杂度为**O(1)**。

➢ 1、常数阶：O(1)

顺序结构、单纯的分支结构（不包含在循环结构中）的时间复杂度。

例2：

```
int sum=0,n=100;          /*执行一次*/
sum=(1+n)*n/2;            /*执行第一次*/
sum=(1+n)*n/2;            /*执行第二次*/
sum=(1+n)*n/2;            /*执行第三次*/
sum=(1+n)*n/2;            /*执行第四次*/
sum=(1+n)*n/2;            /*执行第五次*/
printf("%d",sum);         /*执行一次*/
```

这种与问题的大小无关（n的多少），执行时间恒定的算法，称之为O(1)的时间复杂度。

推导大O阶方法

➢ 2、线性阶：O(n)

循环结构的时间复杂度。关键要分析循环结构的运行情况。

例3：

```
int i;
for(i=0;i<n;i++) {
    /* 时间复杂度为O(1)的程序步骤序列 */
}
```

该算法时间复杂度为O(n)，因为循环体中的代码需要执行n次。

推导大O阶方法

➢ 3、对数阶：O(log n)

例4：下面这段代码，时间复杂度是多少？

```
int count=1;
while(count<n)  {
    count=count*2;
    /*  时间复杂度为O(1)的程序步骤序列 */
}
```

由$2^x=n$得到$x=\log_2 n$，该算法时间复杂度为O(log n)。

推导大O阶方法

- 4、平方阶：O(n2)、O(m*n)

例5：下面这段代码，时间复杂度是多少？

```
int i, j;
for(i=0;i<n;i++) {
    for(j=0;j<n;j++) {
    /* 时间复杂度为O(1)的程序步骤序列 */
    }
}
```

内循环的时间复杂度为O(n)，外循环使内循环语句再循环n次，该算法时间复杂度为$O(n^2)$。

➢ 4、平方阶：O(n2)、O(m*n)

例6：下面这段代码，时间复杂度是多少？

```
int i, j;
for(i=0;i<m;i++) {
    for(j=0;j<n;j++) {
    /* 时间复杂度为O(1)的程序步骤序列 */
    }
}
```

外循环的循环次数改为了m，时间复杂度为O(m*n)。

总结：循环的时间复杂度等于循环体的复杂度乘以该循环运行的次数。

➢ 4、平方阶：O(n2)、O(m*n)

例7：下面这段代码，时间复杂度是多少？

```
int i, j;
for(i=0;i<n;i++) {
    for(j=i;j<n;j++) {
    /* 时间复杂度为O(1)的程序步骤序列 */
    }
}
```

循环总的执行次数为：

$n+(n-1)+(n-2)+...+1=n(n+1)/2=n^2/2+n/2$，

复杂度为$O(n^2)$。

常见的时间复杂度

执行次数函数	阶	非正式术语
12	O(1)	常数阶
2n+3	O(*n*)	线性阶
$3n^2+2n+1$	$O(n^2)$	平方阶
$5\log_2 n+20$	O(log*n*)	对数阶
$2n+3n\log_2 n+19$	O(*n*log*n*)	nlogn 阶
$6n^3+2n^2+3n+4$	$O(n^3)$	立方阶
2^n	$O(2^n)$	指数阶

$O(1)<O(\log n)<O(n)<O(n\log n)<O(n2)<O(n^3)<O(2^n)<O(n!)<O(n^n)$

注2　空间复杂度S(n)按数量级递增顺序也与上表类同。

- 最坏时间复杂度与平均时间复杂度

以查找一个有n个随机数字数组中的某个字，最好情况是第一个数字就是，则算法的时间复杂度为O(1)；最坏情况是要找的这个数字在最后一个位置上，则算法的时间复杂度为O(n)。

假设这个要找的数字在每一个位置的可能性是相同的，它的平均的查找时间为n/2次。

对于算法的分析，有计算所有情况的平均值，即平均时间复杂度；也有计算最坏情况下的时间复杂度，即最坏时间复杂度。一般在没有特殊说明情况下，都是指最坏时间复杂度。

算法的渐进空间复杂度

空间复杂度——算法所需存储空间的度量，记作：

$$S(n)=O(f(n))$$

其中n为问题的规模(或大小)

一个上机执行的程序除了需要存储空间来寄存本身所用指令、常数、变量和输入数据外，也需要一些对数据进行操作的工作单元和存储一些为实现计算所需信息的辅助空间。

原地工作：若辅助空间相对于输入数据量来说是常数，则称此算法为原地工作，空间复杂度为O(1)。

若辅助空间所占空间量依赖于特定的输入， 则除特别指明外，均按最坏情况来分析。

例5：将一维数组a中的n个数逆序存放到原数组中。

S(n) = O(1)
原地工作

【算法1】

```
for(i=0;i<n/2;i++)
{   t=a[i];
    a[i]=a[n-i-1];
    a[n-i-1]=t;
}
```

S(n) = O(n)

【算法2】

```
for(i=0;i<n;i++)
    b[i]=a[n-i-1];
for(i=0;i<n;i++)
    a[i]=b[i];
```

小结

1、数据、数据元素、数据项、数据结构等基本概念

2、对数据结构的两个层次的理解

- 逻辑结构
- 存储结构

3、抽象数据类型的表示方法

4、算法、算法的时间复杂度及其分析的简易方法