福州大学至诚学院
FUZHOU UNIVERSITY ZHICHENG COLLEGE

高级语言程序设计（C语言与数据结构）

杨雄

83789047@qq.com

课程考核

1

期末卷面考试

70%

2

作业+小测

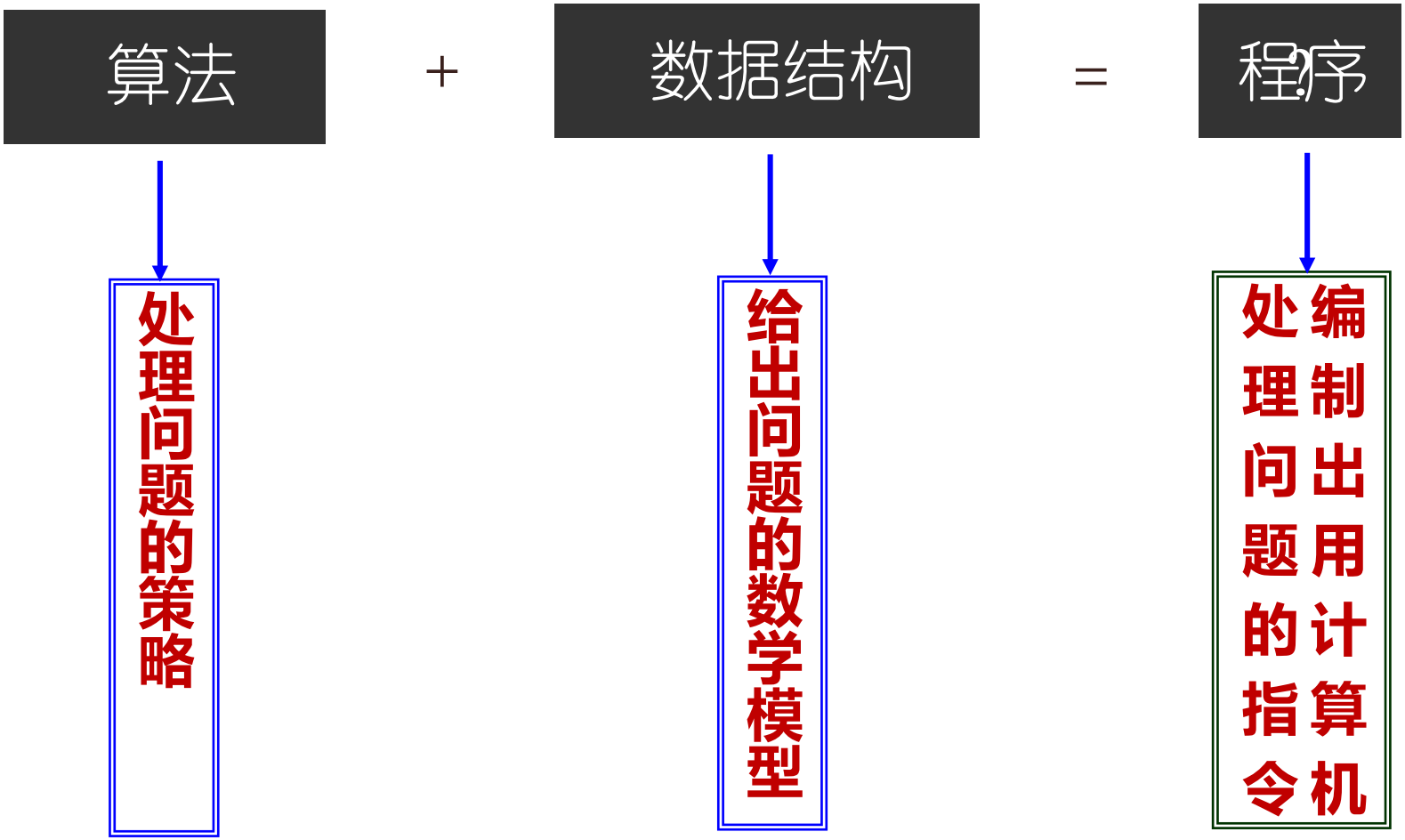
20%

3

出勤+课堂参与度

10%

开场白



课程内容

线性表（栈和队列）

基本数据结构

图

复杂数据结构

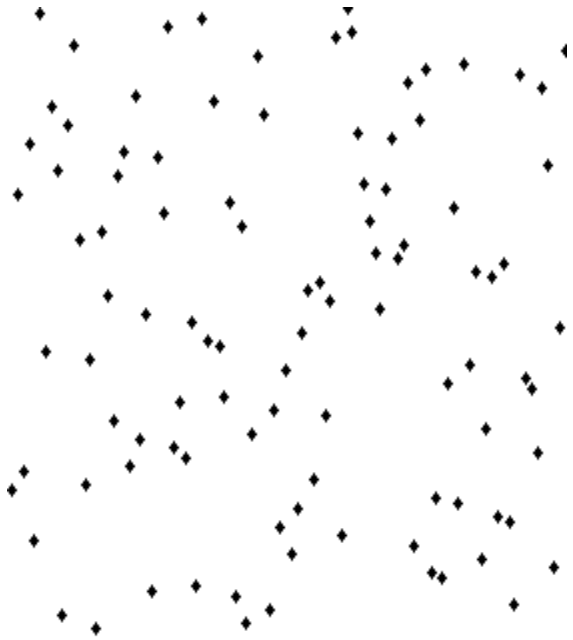
树和二叉树

复杂数据结构

查找与内部排序

算法
/数据结构的应用

冒泡排序



01 定义

“多次扫描一个数组，交换遇到的每一对相邻的、顺序反了的数字；当不再发生交换时，数组已完成排序”

02 代码实现

```
void bubble_sort(int a[], int n)
{
    int i, j, temp;
    for (j = 0; j < n - 1; j++)
        for (i = 0; i < n - 1 - j; i++)
        {
            if(a[i] > a[i + 1])
            {
                temp = a[i];
                a[i] = a[i + 1];
                a[i + 1] = temp;
            }
        }
}
```

1. 绪 论

01 什么是数据结构

02 基本概念与术语

03 抽象数据类型

04 算法与算法分析

学习目的

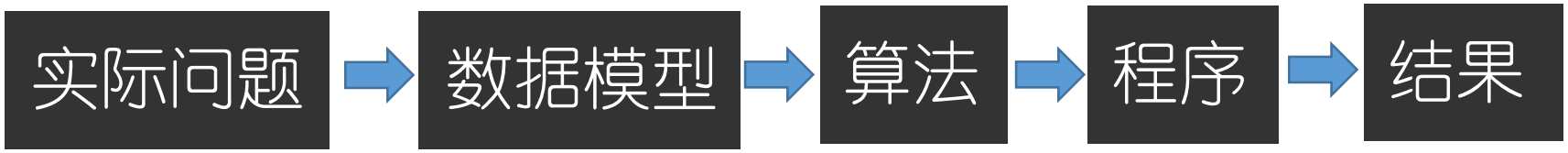
- 掌握数据结构的相关概念和术语
  掌握四类基本的数据关系
- 理解抽象数据类型的表示及实现
- 对算法、算法要求、算法效率的度量进行有效的分析

Part.1

1.1 什么是数据结构？

1.1 什么是数据结构？

- 计算机解决问题的步骤



- 描述非数值计算问题的模型是---
  - 如表、树、图之类的数据结构
- 数据结构是---
  - 研究计算机的操作对象(数据)以及它们之间的关系和操作等的学科。

1.1 什么是数据结构？

◆ 数据结构的例子

例1：电话号码查询系统

设有一个电话号码薄，它记录了N个人的名字和其相应的电话号码，假定按如下形式安排：(a_1, b_1)，(a_2, b_2)，…(a_n, b_n)，其中a_i, b_i(i=1，2…n) 分别表示某人的名字和电话号码。

姓名	电话号码
陈海	13612345588
李四锋	13056112345
。。。	。。。

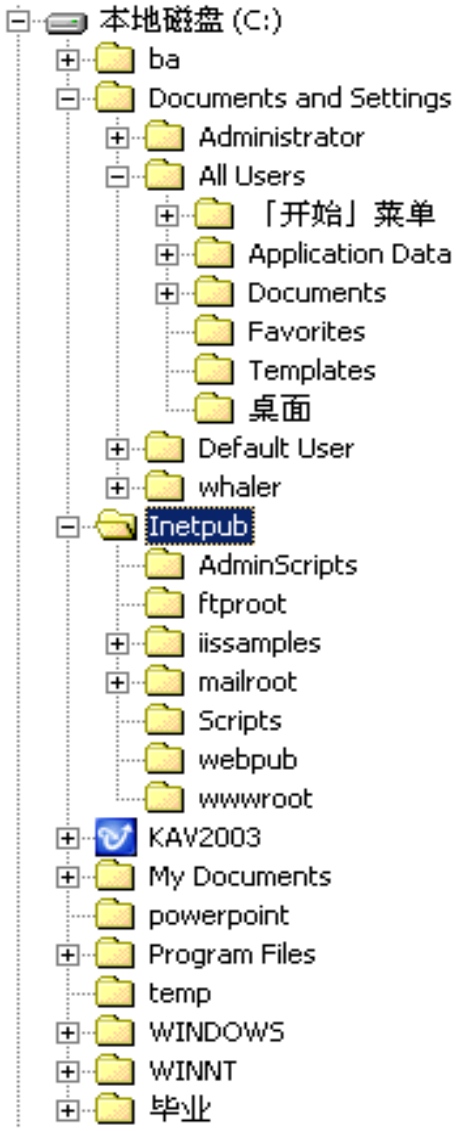
数据与数据成简单的一对一的线性关系。

1.1 什么是数据结构？

◆ 数据结构的例子

例2：磁盘目录文件系统

磁盘根目录下有很多子目录及文件，每个子目录里又可以包含多个子目录及文件，但每个子目录只有一个父目录，依此类推。

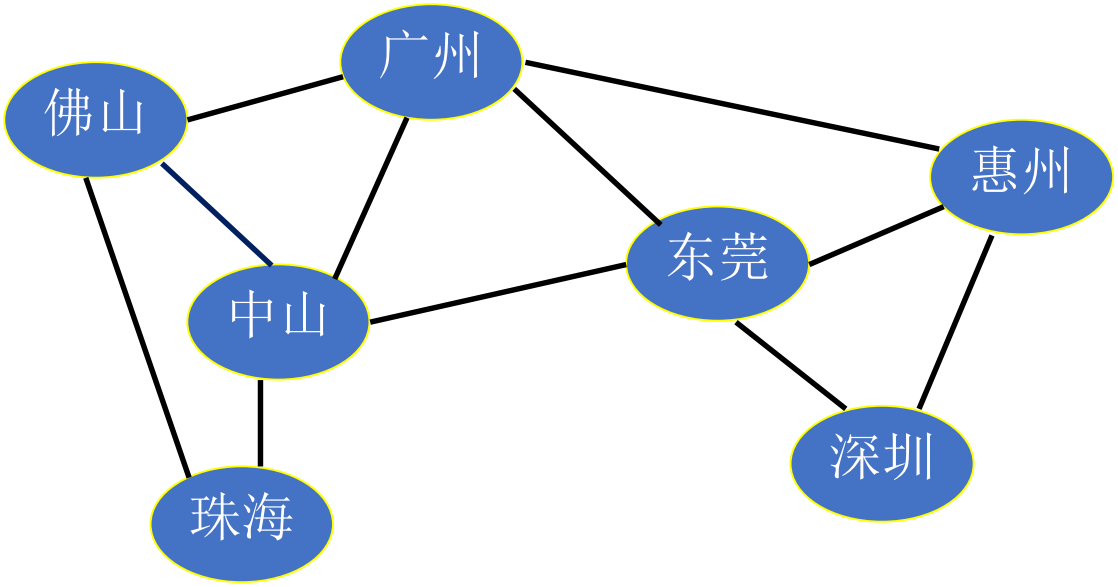


数据与数据成一对多的关系，是一种典型的非线性关系结构—树形结构。

◆ 数据结构的例子

例3：交通网络图

从一个地方到另外一个地方可以有多条路径。



数据与数据成多对多的关系，是一种非线性关系结构。

Part.2

1.2 基本概念和术语

1.2 基本概念和术语

数据：是客观事物的符号表示。在计算机科学中指的是所有能输入到计算机中并被计算机程序处理的符号的总称。

- 数值性数据
- 非数值性数据

1.2 基本概念和术语

数据元素：是组成数据的、有一定意义的基本单位，在计算机中通常作为一个整体来进行考虑和处理。

数据项：一个数据元素可由若干个数据项组成。

★数据项是数据的不可分割的最小单位。

1.2 基本概念和术语

数据对象：是性质相同的数据元素的集合，是数据的一个子集。

如：

字符集合C={‘A’,’B’,’C,…} 。

整数数据对象 N = { 0, ±1, ±2, … }

1.2 基本概念和术语

在现实世界中，不同数据元素之间不是独立的，而是存在特定的关系，我们将这些关系称为结构。

数据结构：是指相互之间存在一种或多种特定关系的数据元素的集合。

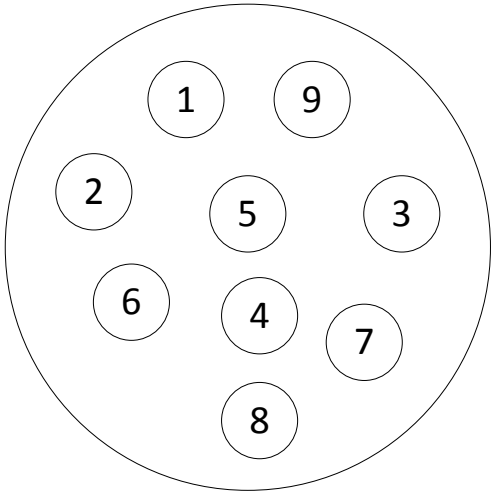
1.2 基本概念和术语

- 逻辑结构和物理结构
  - 逻辑结构：**是指数据对象中数据元素之间的相互关系。**

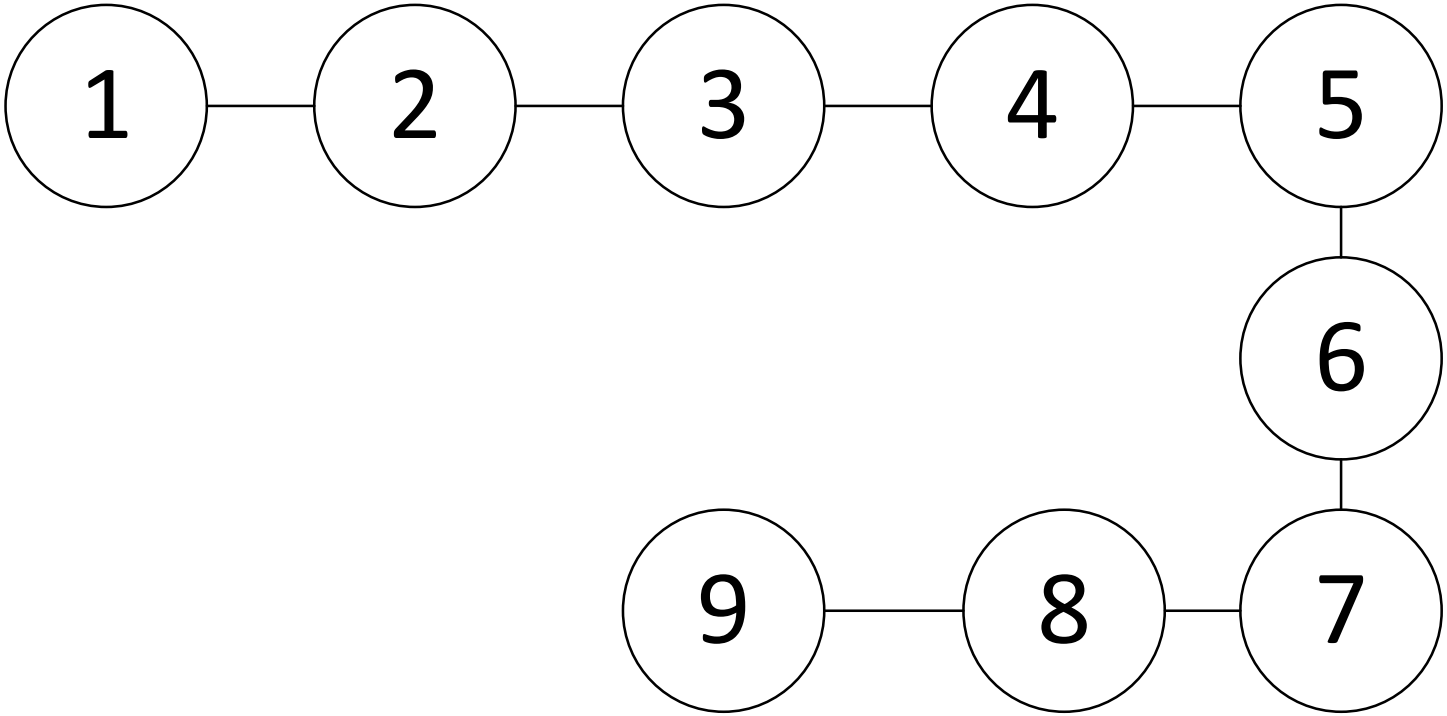
逻辑结构分为以下四种：

① **集合**：

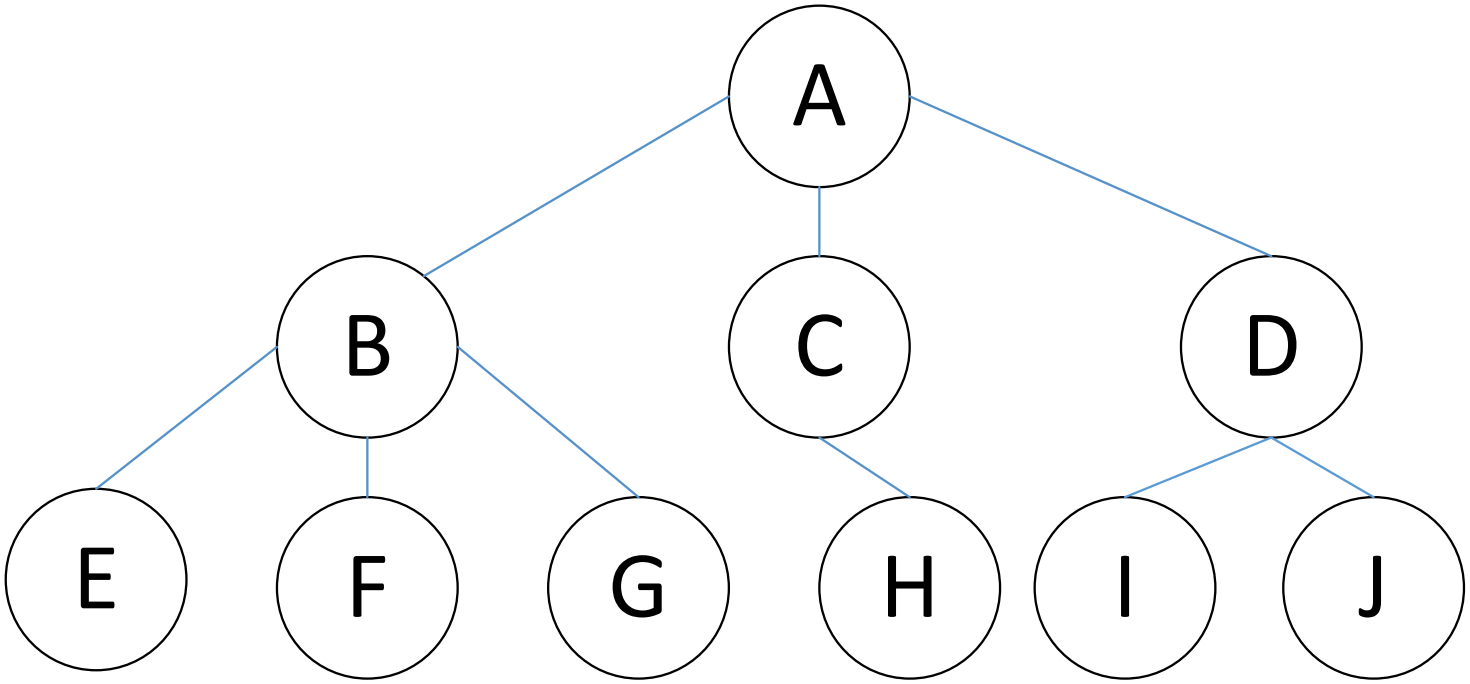
结构中的数据元素除了“同属于一个集合”外，没有其它关系。



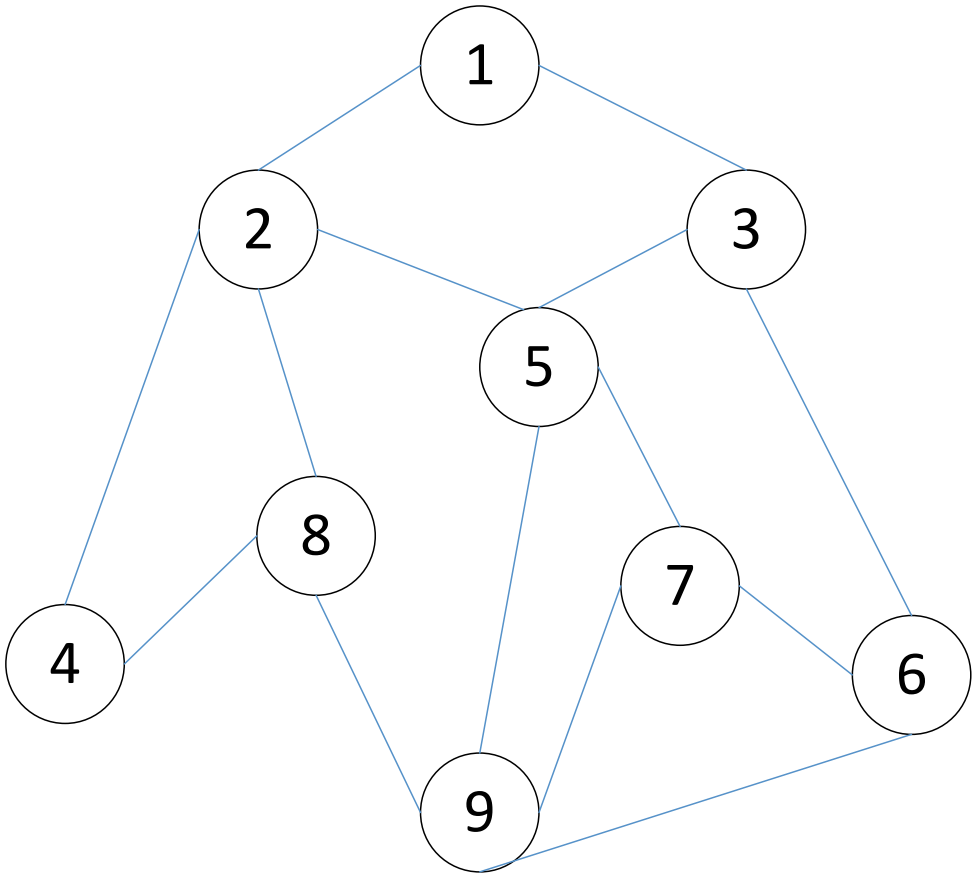
② 线性结构：数据元素之间存在一对一的关系。



③ 树型结构：树形结构中的数据元素之间存在一对多的关系。



④ 图形结构：图形结构中的数据元素之间存在多对多的关系。



1.2 基本概念和术语

- 逻辑结构：**是指数据对象中数据元素之间的相互关系。**

①线性结构：除第一个元素和最后一个元素之外，其他元素都有且仅有一个直接前驱，有且仅有一个直接后继；

②非线性结构：其逻辑特征是一个结点可能有多个直接前驱和直接后继；

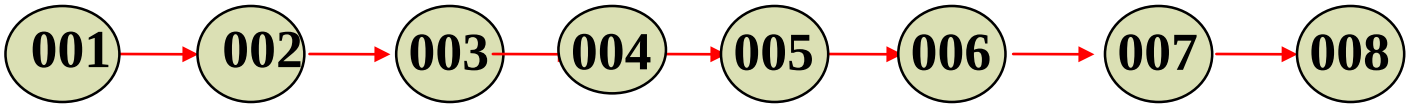
1.2 基本概念和术语

例

学生基本情况登记表，记录了每个学生的学号、姓名、专业、政治、面貌，表中的记录是按学生的学号顺序排列的。

学生间学号顺序关系是一种线性结构关系

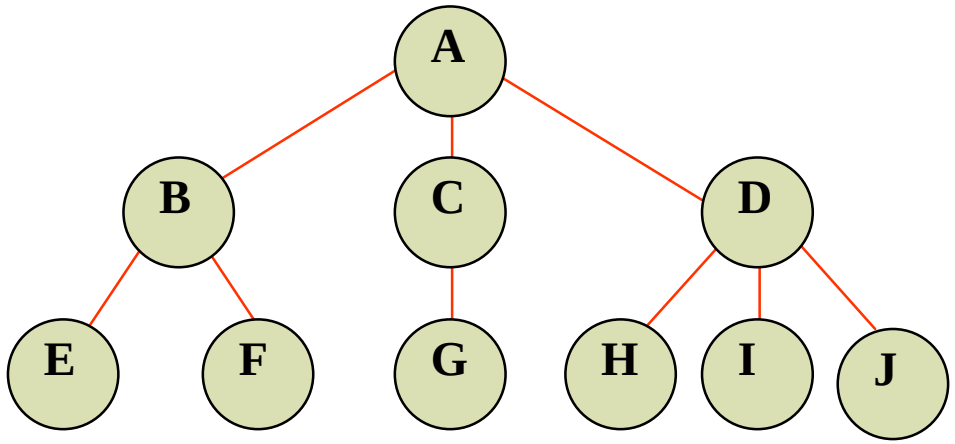
学号	姓名	专业	政治面貌
001	王洪	计算机	党员
002	孙文	计算机	团员
003	谢军	计算机	团员
004	李辉	计算机	团员
005	沈祥福	计算机	党员
006	余斌	计算机	团员
007	巩力	计算机	团员
008	孔令辉	计算机	团员



1.2 基本概念和术语

例

家族的族谱:假设某家族有10个成员A， B， C， D， E， F，G， H，I， J，他们之间的血缘关系可以用如下图表示。



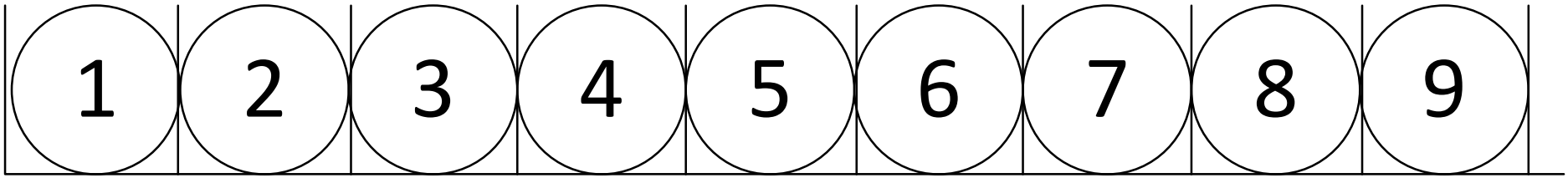
1.2 基本概念和术语

- 逻辑结构和物理结构

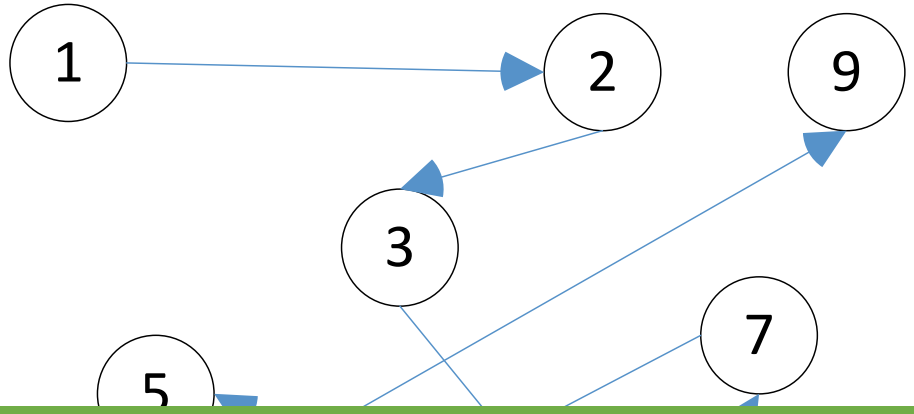
  ➢物理结构：是指数据的逻辑结构在计算机中的存储形式。

  物理结构主要有两种：

  ① **顺序存储结构：把数据元素存放在地址连续的存储单元里，其数据间的逻辑关系和物理关系是一致的。**



② **链式存储结构**：是把数据元素存放在**任意的存储单元**里，这组存储单元**可以是连续的，也可以是不连续的。**



逻辑结构是面向问题的

而物理结构是面向计算机的，其基本的目标就是将数据其及逻辑关系存储到计算机的内存中

1.2 基本概念和术语

数据的逻辑结构和物理结构是密不可分的两个方面，一个算法的设计取决于所选定的逻辑结构，而算法的实现依赖于所采用的存储结构。

在C语言中，用一维数组表示顺序存储结构；用结构体类型表示链式存储结构。

1.2 基本概念和术语

对每种数据结构，主要讨论如下三方面的问题：

①数据的逻辑结构

数据元素之间的逻辑关系，是具体关系的抽象。

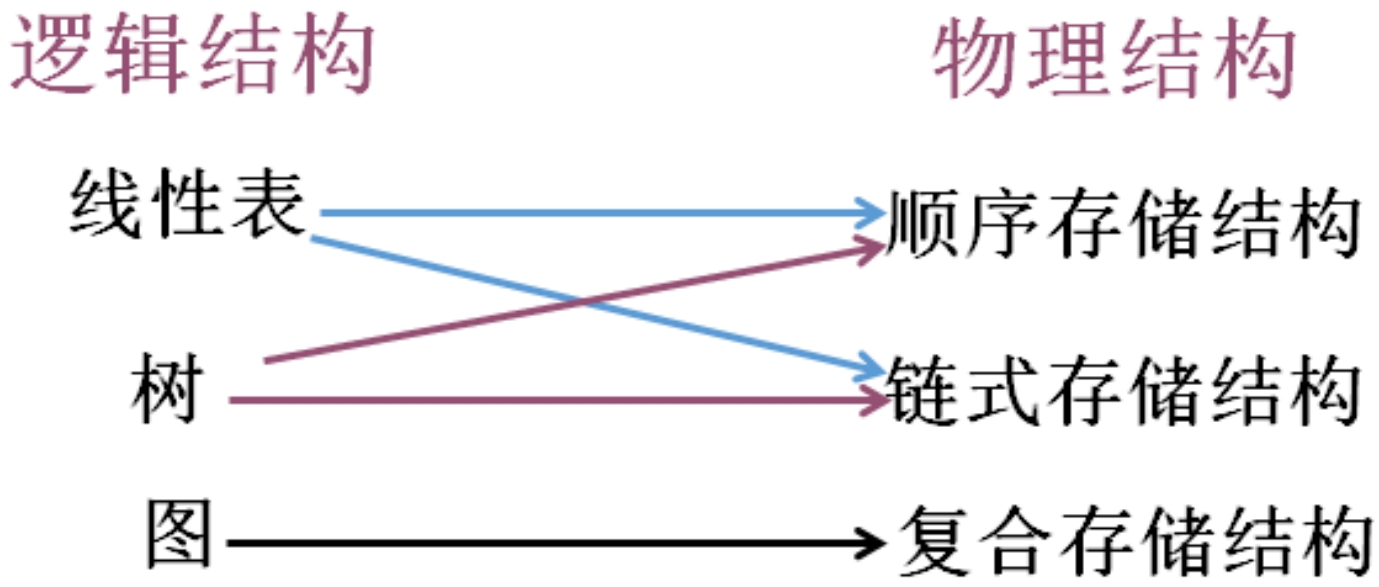
②数据的存储结构(物理结构)：

数据元素及其关系在计算机内存中的表示；

③数据的运算

即对数据施加的操作。定义在数据的逻辑结构上的抽象的操作。

1.2 基本概念和术语



逻辑结构与所采用的存储结构

Part.3

1.3 抽象数据类型

1.3 抽象数据类型

数据类型：指的是一组性质相同的值的集合和定义在此集合上的一些操作的总称。

例:C语言中的数据类型

■ 基本数据类型：是不可以再分解的数据类型，包括整形、字符型等。

■ 结构类型：由若干个类型组合而成，是可以再分解的。如整形数组。

抽象是指抽取出事物具有的普遍性的本质。

1.3 抽象数据类型

抽象数据类型(Abstract Data Type ，简称ADT)：是指一个数学模型以及定义在该模型上的一组操作。

ADT的定义仅是一组逻辑特性描述， 与其在计算机内的表示和实现无关。因此，不论ADT的内部结构如何变化，只要其数学特性不变，都不影响其外部使用。

1.3 抽象数据类型

ADT的一般定义形式是：

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>

ADT特点：

- 降低了软件设计的复杂性；
- 提高了程序的可读性和可维护性；
- 程序的正确性容易保证

其中数据对象和数据关系的定义用伪码描述。

基本操作的定义是：

<基本操作名>(<参数表>)

初始条件： <初始条件描述>

操作结果： <操作结果描述>

- **初始条件：描述操作执行之前数据结构和参数应满足的条件;若不满足，则操作失败，返回相应的出错信息。**
- **操作结果：描述操作正常完成之后，数据结构的变化状况和应返回的结果。**

1.3 抽象数据类型

- ADT Triplet{

  数据对象:D={e1,e2,e3|e1,e2,e3属于集合}

  数据关系:R1={ < e1,e2 > | < e2,e3 > }

  基本操作：

  - InitTriplet(&T,v1,v2,v3)

    初始条件:

    操作结果: 构造三元组T,元素e1,e2和e3分别被赋予参数v1,v2和v3的值。

  - DestroyTriplet(&T)

    初始条件: 三元组T已经存在。

    操作结果: 销毁三元组T

Part.4

1.4 算法与算法分析

1.4 算法与算法分析

一个求1+2+3+……+100结果的程序，你应该怎么写呢？

```
int i, sum = 0, n = 100;
for(i = 1; i <= n; i++)
{
    sum += i;
}
printf(“ %d “, sum);
```

1.4 算法与算法分析

一个求1+2+3+……+100结果的程序，你应该怎么写呢？

sum = 1 + 2 + 3+ …… + 99 + 100

sum = 100 + 99 + 98+ …… + 2 + 1

2xsum=101 + 101 + 101 + …… + 101 + 101

100个

所以 sum = 5050

用程序来实现如下：

```
int sum = 0, n = 100;

sum = (1 + n) * n / 2;

printf(“ %d “ , sum);
```

1.4 算法与算法分析

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。

算法具有以下五个特性

① 输入： 有零个或多个输入，这些输入取自于某个特定的对象集合。

② 输出： 有一个或多个输出，这些输出是同输入有着某些特定关系的量。

③ 有穷性： 必须总是在执行有穷步之后结束，且每一步都在有穷时间内完成。

算法具有以下五个特性

④ 确定性：每一条指令必须有确切的含义。不存在二义性。且算法只有一个入口和一个出口。

⑤ 可行性： 算法是能行的。即算法描述的操作都可以通过已经实现的基本运算执行有限次来实现。

1.4 算法与算法分析

一个算法可以用多种方法描述，主要有：使用自然语言描述；使用形式语言描述；使用计算机程序设计语言描述。

在本门课程的学习、作业练习、上机实践等环节，算法都用C语言来描述。在上机实践时，为了检查算法是否正确，应编写成完整的C语言程序。

1.4 算法与算法分析

评价一个好的算法有以下几个标准：

① 正确性(Correctness)：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。

- **算法程序没有语法错误**
- **算法程序对于合法的输入数据能够产生满足要求的输出结果**
- **算法程序对于非法的输入数据能够得出满足规格说明的结果**
- **算法程序对于精心选择的，甚至刁难的测试数据都有满足要求的输出结果**

② **可读性**(Readability)： 算法应容易供人阅读和交流。可读性好的算法有助于对算法的理解和修改。

③ **健壮性**(Robustness)： 算法应具有容错处理。当输入非法或错误数据时，算法应能适当地作出反应或进行处理，而不会产生莫名其妙的输出结果。

④ **通用性**(Generality)： 算法应具有一般性 ，即算法的处理结果对于一般的数据集合都成立。

⑤ **时间效率与存储量需求**： 效率指的是算法执行的时间；存储量需求指算法执行过程中所需要的最大存储空间。一般地，这两者与问题的规模有关。

1.4 算法与算法分析

算法执行时间需通过依据该算法编制的程序**在计算机上运行所消耗的时间**来度量。其方法通常有两种：

事后统计：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，从而确定算法效率的高低。

不采纳

问题：

- 必须先运行依据算法编制的程序；
- 依赖软硬件环境，容易掩盖算法本身的优劣；
- 算法的测试数据设计困难，并用程序的运行时间往往还与测试数据的规律有很大关系。

1.4 算法与算法分析

事前分析：在计算机程序编制前，**依据统计方法**对算法进行估算。

与此相关的因素有：

- **问题的输入规模**
- 对源程序进行编译所需要时间
- 机器执行指令的速度
- **程序中指令重复执行的次数**

频度：算法执行一次，某一语句实际被执行的次数，叫该语句在此算法中的频度。

1.4 算法与算法分析

第一种算法：

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

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 */
```

1.4 算法与算法分析

```
int i, j, x = 0, sum = 0, n = 100;      /* 执行1次 */
for(i = 1; i<= n; i++)
{
   for(j = 1; j<= n; j++)
   {
         x++;                                  ?
         sum += x;
   }
}
printf("%d", sum);                      /* 执行1次 */
```

- **算法的时间复杂度**

算法的计算时间 = ∑ 语句频度，记做 T(n)

1.4 算法与算法分析

某个算法，随着n的增大，它会越来越优于另一算法，或者越来越差于另一算法。

这其实是**事前分析估算方法**的理论依据，通过**算法时间复杂度**来估算算法时间效率。

1.4 算法与算法分析

- 算法的时间复杂度，只依赖于问题的规模(通常用n表示)，或者说它是问题规模的函数：

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

其中f(n)是问题规模n的某个函数

1.4 算法与算法分析

推导大O阶：

1.用**常数1**取代运行时间中的所有**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**。

得到的结果就是大O阶。

例1：

```
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 */
```

常数阶（顺序结构、分支结构）：O(1)

1.4 算法与算法分析

```
int sum = 0, n = 100;          /* 执行1次 */
sum = (1 + n) * n / 2;         /* 执行1次 */
sum = (1 + n) * n / 2;         /* 执行2次 */
sum = (1 + n) * n / 2;         /* 执行3次 */
sum = (1 + n) * n / 2;         /* 执行4次 */
sum = (1 + n) * n / 2;         /* 执行5次 */
printf(“ %d “ , sum);          /* 执行1次 */
```

例2：

```
int i, sum=0;
for(i = 0; i < n; i++)
{
   sum += i;
}
```

线性阶：O（n）

例4：

```
int i, j, sum;
for (i = 0; i < n; i++)
{
   for (j = 0; j < n; j++)
   {
         sum = sum + i + j;
   }
}
```

平方阶：O（n^2）

1.4 算法与算法分析

常见时间复杂度

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(logn)	对数阶
$2n+3n\log_2 n+19$	O(nlogn)	nlogn阶
$6n^3+2n^2+3n+4$	$O(n^3)$	立方阶

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

Part.5

总结

总 结

1. 熟悉各名词、术语的含义，掌握基本概念，特别是数据的逻辑结构和存储结构之间的关系。分清哪些是逻辑结构的性质，哪些是存储结构的性质。
2. 了解抽象数据类型的定义、表示和实现方法。
3. 了解算法五个要素的确切含义：①有穷性（能执行结束）；②确定性（对于相同的输入执行相同的路径）；③有输入；④有输出；⑤可行性（用以描述算法的操作都是足够基本的）。
4. 掌握计算语句频度和估算算法时间复杂度的方法。