

# 计算机科学导论

## 第4章 程序设计基础

杨老师邮箱: [fdteachers@163.com](mailto:fdteachers@163.com)

# 第4章 程序设计基础

## 学习目标

了解程序设计的基础知识、程序设计风格的重要性、基本的查找和排序方法

掌握结构化程序设计方法和面向对象程序设计方法的思想、几种基本的数据结构

## 4.1 程序设计基础

程序设计是指用计算机语言对所要解决的问题中的数据以及处理问题的方法和步骤所做的完整而准确的描述的过程。程序设计步骤如下：

- (1) 确定要解决的问题。
- (2) 分析问题。在着手解决问题之前，应该通过分析，充分理解问题，明确原始数据、解题要求、需要输出的数据及形式等。
- (3) 选择计算方法。
- (4) 确定数据结构和算法。算法是解题的过程。首先集中精力于算法的总体规划，然后逐层降低问题的抽象性，逐步充实细节，直到最终把抽象的问题具体化成可用程序语句表达的算法。这是一个自上而下、逐步细化的过程。

## 4.1 程序设计基础

- (5) 绘制流程图。
- (6) 编写程序。利用程序设计语言表示算法，编写代码。
- (7) 调试并测试程序。调试程序包括编译和连接等操作。程序员还要对程序执行的结果进行分析，只有能够得到正确结果的程序才是所需的程序。
- (8) 整理资料，交付使用。
- 高质量程序设计目标是结构化程度高、可读性好、效率高、可靠性高、便于维护。



## 4.2 程序设计方法

- 程序设计初期，由于计算机硬件条件的限制，运算速度与存储空间都迫使程序员追求高效率，编写程序成为一种技巧与艺术，而程序的可理解性、可扩充性等要素则次之。
- 随着计算机硬件与通信技术的发展，计算机应用领域越来越广泛，应用规模也越来越大，程序设计不再是一两个程序员就可以完成的任务。在这种情况下，编写程序不再片面追求高效率，而是综合考虑程序的可靠性、可扩充性、可重用性和可理解性等要素。

## 4.2.1 结构化程序设计方法

- 1965年，艾兹格·W·迪科斯彻（Edsger Wybe Dijkstra，1930~2002）提出了结构化的概念，它是软件发展的一个重要的里程碑。结构化程序设计是以模块功能和处理过程设计为主的详细设计的基本原则，采用自顶向下、逐步求精及模块化的程序设计方法，使用三种基本控制结构构造程序，任何程序都可由顺序、选择、循环三种基本控制结构构造。
- 结构化程序设计思想：
- 采用自顶向下、逐步求精的设计方法和单入口单出口的控制结构。



## 4.2.1 结构化程序设计方法

- 20世纪60年代末到70年代初，当时采用结构程序设计方法的两个最著名项目是：
- ①纽约时报信息库管理系统，含8.3万行源代码，只花了11人·年，第一年使用过程中，只发生过一次使系统失效的软件故障；
- ②美国宇航局空间实验室的模拟系统，含40万行源代码，只用两年时间就全部完成。

## 4.2.1 结构化程序设计方法

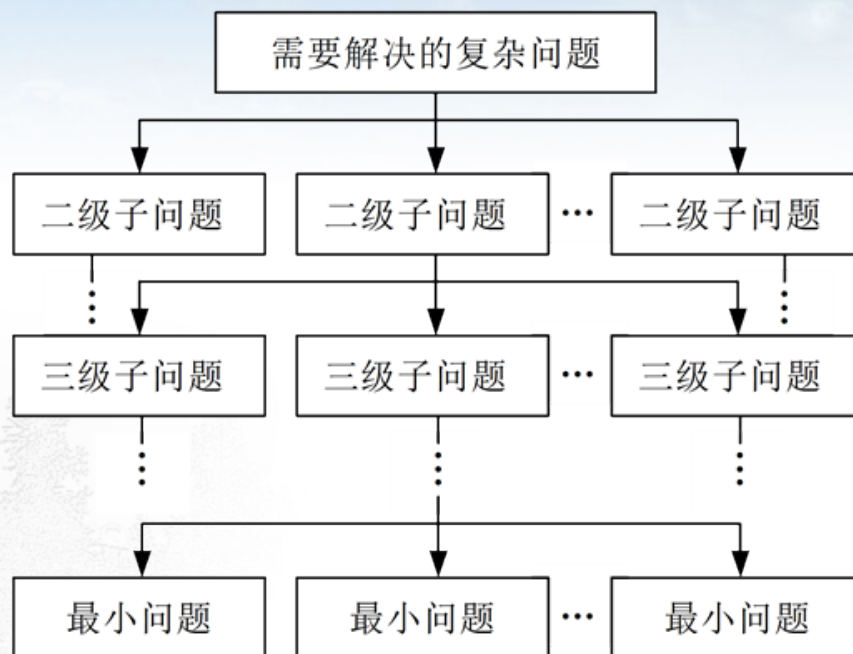
### 自上而下与自下而上

- 先将一个大问题分解成若干个子问题，把比较复杂的子问题继续分解成更加简单的二级子问题，直至每个子问题都有显而易见的解决办法，然后在实现时采用自下而上的方法，逐一编写解决各个子问题的程序。设计程序时采用自上而下的方法比采用自下而上的方法效率要高得多。



## 4.2.1 结构化程序设计方法

➤ 采用自上而下解决问题的思路如图:



## 4.2.1 结构化程序设计方法

### 结构化方法

- 结构化方法有助于在正式编写程序之前充分理解问题的实质和实现方法，并且可以在具体编码过程中提供指导。

## 4.2.1 结构化程序设计方法

结构化方法通常遵循以下原则:

- (1) 用户参与的原则
- (3) 自上而下的原则
- (4) 阶段成果文档化

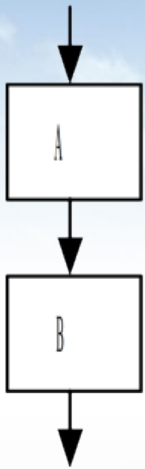
## 4.2.1 结构化程序设计方法

### 结构化程序设计方法

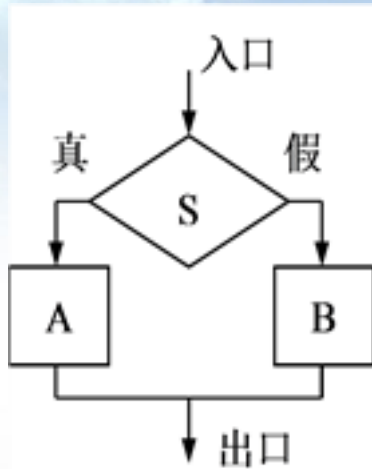
- 结构化程序设计的原则是：
- (1) 使用顺序、选择、循环3种基本控制结构表示程序逻辑。
- (2) 程序语句组织成容易识别的语句模块，每个模块都是单入口、单出口。
- (3) 严格控制GOTO语句的使用。



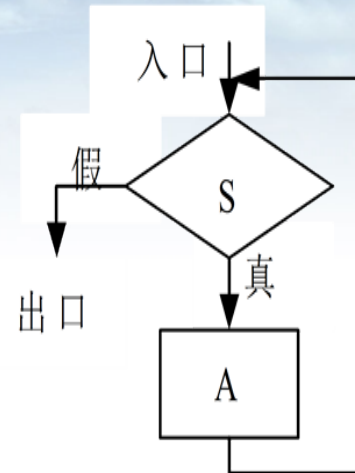
## 4.2.1 结构化程序设计方法



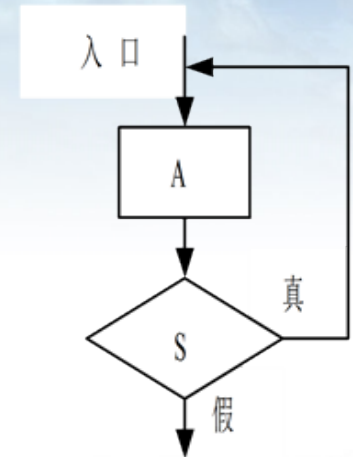
(a) 顺序结构



(b) 选择结构



(c) while循环



(d) do-while循环

## 4.2.1 结构化程序设计方法

### 模块化方法

- 一个复杂的问题可以划分为多个简单问题的组合。
- 在自顶向下、逐步细化的过程中，把复杂问题分解成一个个简单问题的最基本方法就是模块化。
- 模块化便于问题的分析，模块体现了信息隐藏的概念。模块常用子程序加以实现。

## 4.2.2 面向对象的程序设计方法

### 1. 面向对象的思想

- OO(Object Oriented, 面向对象)的程序设计把客观事物看作具有属性和行为的对象, 通过抽象找出同一类对象的共同属性(静态特征)和行为(动态特征), 形成类。
- 阿伦(Alan Kay, 1940~)设计出了名震业界的Smalltalk语言, 被业界公认为“面向对象编程系列语言”的代表作品。



## 4.2.2 面向对象的程序设计方法

### 2. 对象、消息传递和类

#### 对象(object)

是对现实问题的高度概括、分类和抽象。每个对象都只有自己的数据和相应的处理函数，整个程序是由一系列相互作用的对象来构成，不同对象之间是通过发送消息来实现相互联系、相互作用。



## 4.2.2 面向对象的程序设计方法

### ➤ 消息传递

消息是对象之间进行通信的一种机制。发送给某个对象的一个消息包含着要求接收对象完成某些活动的信息。接收到消息的对象经过解释，然后予以响应。这个通信机制叫做消息传递。

发送消息的对象并不需要知道接收消息的对象如何对请求予以响应。

## 4.2.2 面向对象的程序设计方法

### ➤ 类(class)

定义了一组大体上相似的对象。一个类所包含的方法和数据描述一组对象的共同行为和属性。对象则是类的具体化，是类的实例。

类通过派生可以有子类，同样也可以有父类，形成层次结构。

## 4.2.2 面向对象的程序设计方法

### 3 . 抽象(abstract)

- 是对具体事物(即对象)进行概括，即忽略事物的非本质特征，只注意那些与当前目标有关的本质特征，从而抽象出一类对象的共性并加以描述。
- 对一个问题的抽象一般来讲应该包括两个方面：数据抽象和代码抽象(或称为行为抽象)。

## 4.2.2 面向对象的程序设计方法

- 例如，对一批书籍进行归纳、抽象。读者可能关心的是书籍的学科种类、价格、出版社信息，而销售这批书籍的书店经销商希望开发一个书籍的进货、销售情况的管理软件，则他所关心的属性除了以上这些信息以外，更关心可能是书籍的销售量和销售速度。
- 由此可以看出，同一个研究对象，由于所研究问题的侧重点不同，就可能产生不同的抽象结果；即使对于同一个问题，解决问题的要求不同，也可能产生不同的抽象结果，这些结果的不同就表现在不同的数据和方法上。



## 4.2.2 面向对象的程序设计方法

### 4 . 封装性

#### ➤ 封装的两个含义

第一是，将抽象得到的数据成员和代码成员相结合，形成一个不可分割的整体，即对象，这种数据及行为的有机结合也就是封装。

第二个含义称为信息隐蔽，即尽可能隐蔽对象的内部细节。在隐蔽对象内部细节的同时，对象需要提供与外部世界进行交流的接口，并实现对数据访问权限的合理控制，把整个程序中不同部分的相互影响减少到最低限度。

## 4.2.2 面向对象的程序设计方法

### 5 . 继承性

- 是父类和子类之间共享数据和方法的机制。在定义一个类的时候，可以以一个已经存在的类为基础，并把这个已经存在的类所包含的属性和方法作为自身的一部分，然后加入新的属性和方法以区别于原来的类。
- 原有的类称为**基类或父类**，产生的新类称为派生类。

## 4.2.2 面向对象的程序设计方法

### 6. 多态性

- 在收到外部消息时，对象通常要予以响应。不同的对象收到同一消息可能产生完全不同的结果。

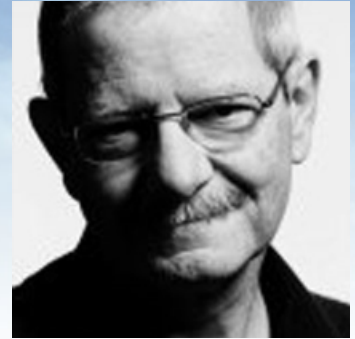
## 4.2.3 函数程序设计

- 函数程序设计语言使用非常简单的计算模型或者程序观点，一个程序是输入集合到输出集合的数学函数，执行一个程序便是计算一个函数在给定输入的输出值。
- 函数程序的特点是清晰、简洁和易读等，这些特点使得大型程序的开发更高效，维护更容易。
- 函数程序设计语言因其简单的基本理论，使现代程序设计的基本思想，如抽象、数据抽象、多态和重载等都得到了最清楚的体现。因此，函数程序设计不仅是学习现代程序设计思想的理想语言，而且为传统的命令式和面向对象的程序设计语言提供了很有意义的视角。



## 4.2.3 函数程序设计

- 1987年Joe Armstrong在Prolog基础上创建了Erlang，经过了十年发展，于1998年发布开源版本。Erlang是运行于虚拟机的解释性语言，但是现在也包含有乌普萨拉大学高性能Erlang计划（HiPE）开发的本地代码编译器，自R11B-4版本开始，Erlang也开始支持脚本式解释器。在编程范型上，Erlang属于多重范型编程语言，涵盖函数式、并发式及分布式。顺序执行的Erlang是一个及早求值，单次赋值和动态类型的函数式编程语言。



## 4.2.4 程序设计风格

- 程序设计风格指一个人编制程序时所表现出来的特点、习惯、逻辑思路等。

## 4.2.4 程序设计风格

### 1. 源程序文档化

(1) 标识符应按意取名。

(2) 程序应加注释。注释是程序员与读者之间通信的重要工具，用自然语言或伪码描述。它说明了程序的功能，特别是在维护阶段，对理解程序提供了明确指导。注释分序言性注释和功能性注释。序言性注释应置于每个模块的起始部分。

## 4.2.4 程序设计风格

### 2. 数据说明

➤ 为了使数据定义更易于理解和维护，有以下原则。

(1) 数据说明顺序应规范，使数据的属性更易于查找，从而有利于测试、纠错与维护。如常量说明、类型说明、全局变量说明、局部变量说明等。

(2) 一个语句说明多个变量时，各变量名按字典顺序排列。

(3) 对于复杂的数据结构，要加注释，说明在程序实现时的特点。



## 4.2.4 程序设计风格

### 3 . 语句构造

- 语句构造的原则是简单、直接，不能为了追求效率而使代码复杂化。
- 为了便于阅读和理解，一行只写一条语句；
- 不同层次的语句采用缩进形式，使程序的逻辑结构和功能特征更加清晰。
- 要避免复杂的判定条件，避免多重的循环嵌套；
- 表达式中使用括号以提高运算次序的清晰度等。

## 4.2.4 程序设计风格

### 4. 在编写输入和输出语句时应考虑原则

- (1) 输入操作步骤和输入格式尽量简单。
- (2) 应检查输入数据的合法性、有效性，报告必要的输入状态信息及错误信息。
- (3) 输入一批数据时，使用数据或文件结束标志，而不要用计数来控制。
- (4) 交互式输入时，提供可用的选择和边界值。
- (5) 当程序设计语言有严格的格式要求时，应保持输入格式的一致性。
- (6) 输出数据表格化、图形化。

输入、输出风格还受其他因素的影响，如输入/输出设备、用户经验及通信环境等。

## 4.2.4 程序设计风格

### 5 . 效率

➤ 效率是指处理机时间和存储空间的使用。

(1) 效率是一个性能要求，目标在需求分析时给出。

(2) 追求效率要建立在不损害程序可读性和可靠性基础上，要在确保程序正确、清晰的情况下提高效率。

(3) 提高程序效率的根本途径在于选择良好的设计方法、良好的数据结构，而不是靠编程时对程序语句做调整。

## 4.2.5 程序设计举例

例4.1 输入三角形的3个边长 $a$ ,  $b$ 和 $c$ , 求三角形面积。

$$area = \sqrt{s(s-a)(s-b)(s-c)} \quad s = (a+b+c)/2$$

则计算该三角形的面积的C语言源程序核心代码如下:

```
#include <math.h>
main()
{
    float a,b,c,s,area;
    scanf( "%f,%f,%f" ,&a,&b,&c);
    s=1.0/2*(a+b+c);
    area=sqrt(s*(s-a)*(s-b)*(s-c));
    printf( "a=%7.2f,b=%7.2f,c=%7.2f,s=%7.2f\n" ,a,b,c,s);
    printf( "area=%7.2f\n" ,area);
}
```



## 4.2.5 程序设计举例

例4.2 求  $ax^2 + bx + c = 0$  方程的根,  $a, b, c$  由键盘输入, 设  $b^2 - 4ac \geq 0$ , 根据数学知识, 可以求得方程的根为:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2} \quad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2}$$

$$\text{设: } p = \frac{-b}{2a}, \quad q = \frac{\sqrt{b^2 - 4ac}}{2a}$$

则方程的根可以改写为:

$$x_1 = p + q \quad x_2 = p - q$$

## 4.2.5 程序设计举例

计算该方程的根的原程序核心代码如下：

```
#include <math.h>
main()
{
    float a,b,c,disc,x1,x2,p,q;
    scanf( "a=%f,b=%f,c=%f" ,&a,&b,&c);
    disc=b*b-4*a*c;
    p=-b/(2*a);
    q=sqrt(disc)/(2*a);
    x1=p+q;x2=p-q;
    printf( "\nx1=%5.2f\nx2=%5.2f\n" ,x1,x2);
}
```

## 4.3 基本数据结构

- 数据结构(Data Structure)是系统设计和程序开发的重要基础。

## 4.3.1 基本概念

### 1 . 数据、数据类型

- 数据是对客观事物的符号表示。在计算机系统内，数据通常是指能够输入到计算机中并被计算机进行处理的符号的集合。例如，数字、字母、汉字、图形、图像、声音等信息在计算机内部的表示都是数据，可以是数值数据，也可以是非数值数据。
- 数据类型是指具有相同取值范围和可以实施同种操作的数据的集合。例如，在程序设计语言中，通常定义了字符型、整数型、数组等多种数据类型。



## 4.3.1 基本概念

### 2 . 数据元素、数据项、数据对象

- 能够独立并完整地描述客观世界实体的基本数据单元称为数据元素，它是组成数据的基本单位。在不同的应用环境中，数据元素有时可以称为节点、记录等。
- 数据项是组成数据元素的不可分割的最小单位。最简单的数据元素是由一个数据项构成的。
- 同类数据元素的集合称为数据对象。

## 4.3.1 基本概念

### 3 . 数据结构

➤ **数据结构**是指数据元素之间的相互关系的集合，包括了数据的逻辑结构、物理结构以及数据的运算。

#### (1)数据的逻辑结构

数据的逻辑结构是指数据元素之间的逻辑关系。数据之间可以根据不同的关系组成不同的数据结构。

## 4.3.1 基本概念

### 线性结构

数据结构中，如果数据元素之间存在着前后顺序的关系，即除了第一个数据元素和最后一个元素外，其他每个元素都有惟一的一个前驱和一个后继元素，这样的数据元素之间的关系被称为线性结构。

### 树形结构

数据结构中，如果数据元素之间有顺序关系，且除了一个被称为根节点的元素外，每个元素都有一个前驱节点，并且可以有多个后继节点，这种逻辑结构称为树形结构。

### 图形结构

数据结构中，如果任何一个数据元素都可以有多个前驱节点和多个后继节点，这种逻辑结构称为图形结构。

## 4.3.1 基本概念

### (2) 数据的物理结构

数据的物理结构是指逻辑结构在计算机存储器中的表示。数据的物理结构不仅要存储数据本身，还要存储表示数据间的逻辑关系。

数据的物理结构主要有四种，分别是顺序结构、链表结构、索引结构及散列结构。



## 4.3.1 基本概念

### ①顺序结构

把所有元素存放在一片连续的存储单元中，逻辑上相邻的元素存储在物理位置相邻的存储单元中，由此得到的存储表示称为顺序存储结构。

顺序存储结构常借助于程序设计语言中的数组来实现。优点是使用方法简单，缺点是必须预先分析出所需定义数组的大小。

## 4.3.1 基本概念

### ②链表结构

对逻辑上相邻的元素不要求其物理位置相邻，元素间的逻辑关系通过附设的指针域来实现，由此得到的存储表示称为链式存储结构。

链式存储结构通常借助于程序设计语言中的指针来实现。

## 4.3.1 基本概念

### ③索引结构

针对每个数据结构建立一张所谓的索引表，每个数据元素占用表中的一项，每个表项包含一个能够惟一识别一个元素的关键字和用以指示该元素的地址指针。

### ④散列结构

通过构造相应的散列函数，由散列函数的值来确定元素存放的地址。

## 4.3.1 基本概念

### (3) 数据运算

数据操作的集合。常见的数据操作有数据的插入、删除、查找、遍历等。

数据操作通常由计算机程序加以实现，通常也叫**算法实现**。

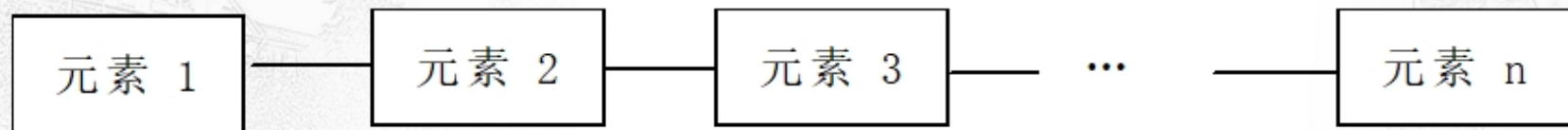


## 4.3.2 几种典型的数据结构

### 1. 线性表

#### (1) 定义

**线性表**是由有限个相同的数据元素构成的序列，元素之间是一对一的线性关系，除了第一个元素只有直接后继、最后一个元素只有直接前驱外，其余数据元素都有一个直接前驱和一个直接后继，如图：



## 4.3.2 几种典型的数据结构

### (2)运算和实现

线性表通常采用顺序和链表两种物理实现。对于经常变化的表，通常采取链表结构。线性表常用的运算主要有：插入、删除、查询和遍历。

## 4.3.2 几种典型的数据结构

### ①插入

在保持原有的存储结构的前提下，根据插入要求，在适当的位置插入一个元素。插入操作要求线性表要有足够的存放新元素的空间，如果空间不足，插入操作无法进行，线性表会溢出。

### ②删除

在线性表中，找到满足条件的数据元素，并删除。如果线性表为空，删除就会失败。

## 4.3.2 几种典型的数据结构

### ③查询

在线性表中，按照查询条件，定位数据元素的过程就是查询。查询的条件一般根据数据元素中的关键字进行。实际上，数据的插入和删除都需要首先定位数据元素。对于空的线性表是无法查询的。

### ④遍历

是指按照某种方式，逐一访问线性表中的每一个数据元素，并执行相同处理的操作。这里的处理可以是读、写、或查询等。



## 4.3.2 几种典型的数据结构

### 2. 栈

#### (1)定义

对于由N个数据元素构成的一个线性序列，如果只允许在其固定的一端位置插入和删除一个数据元素，那么这种逻辑结构的数据结构称为**堆栈或栈(stack)**。

允许插入或删除的这一端称为**栈顶(top)**，另一个固定端称为**栈底(bottom)**。当表中没有元素时称为**空栈**。

## 4.3.2 几种典型的数据结构

### (2) 运算和实现

栈的基本运算主要有：入栈、出栈和判断。

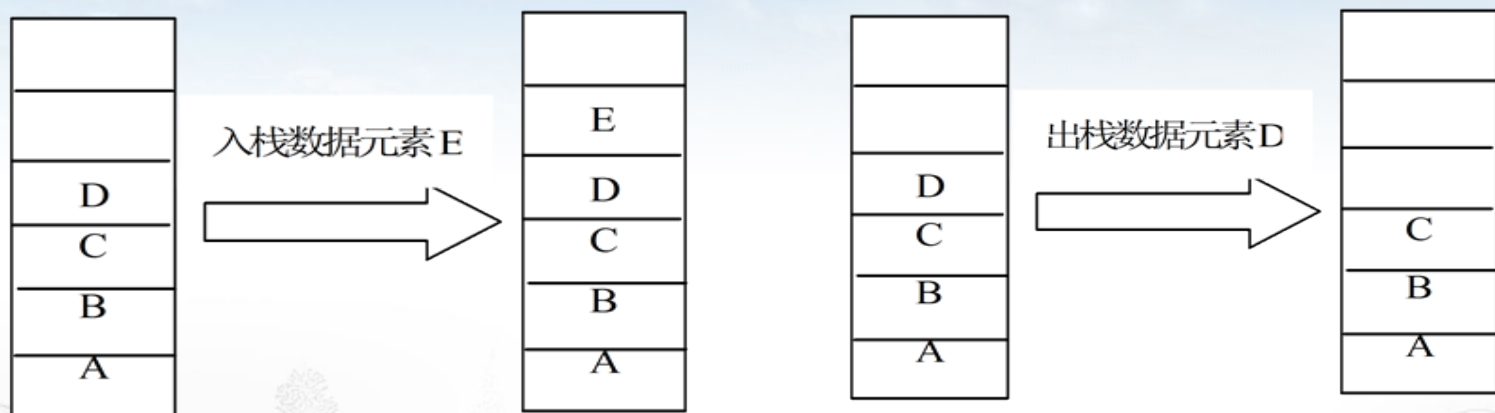
#### ①入栈

入栈也叫压栈，是在栈顶添加新元素的操作，新的元素入栈后成为新的栈顶元素。

#### ②出栈

出栈也叫退栈或弹栈，是将栈顶元素从栈中退出并传递给用户程序的操作。

## 4.3.2 几种典型的数据结构



## 4.3.2 几种典型的数据结构

### ③判断

判断操作用来检查栈内数据是否为空，返回结果是一个逻辑值：真或假。如果栈顶和栈底重合，说明堆栈为空。



## 4.3.2 几种典型的数据结构

### 3. 队列

#### (1) 定义

对于由 $N$ 个数据元素构成的一个线性序列，如果在其固定的一端只允许插入数据元素，且在另一端只允许删除数据元素，这种逻辑结构称为**队列(Queue)**。只允许插入的一端称为**队尾(Rear)**，只允许删除的一端称为**队首(Front)**。

队列是一种“**先进先出**”的数据结构，在操作系统的进程调度管理、网络数据包的存储转发等多种领域中被广泛使用。

## 4.3.2 几种典型的数据结构

### (2) 运算

队列的基本运算主要有：入队、出队和判断。

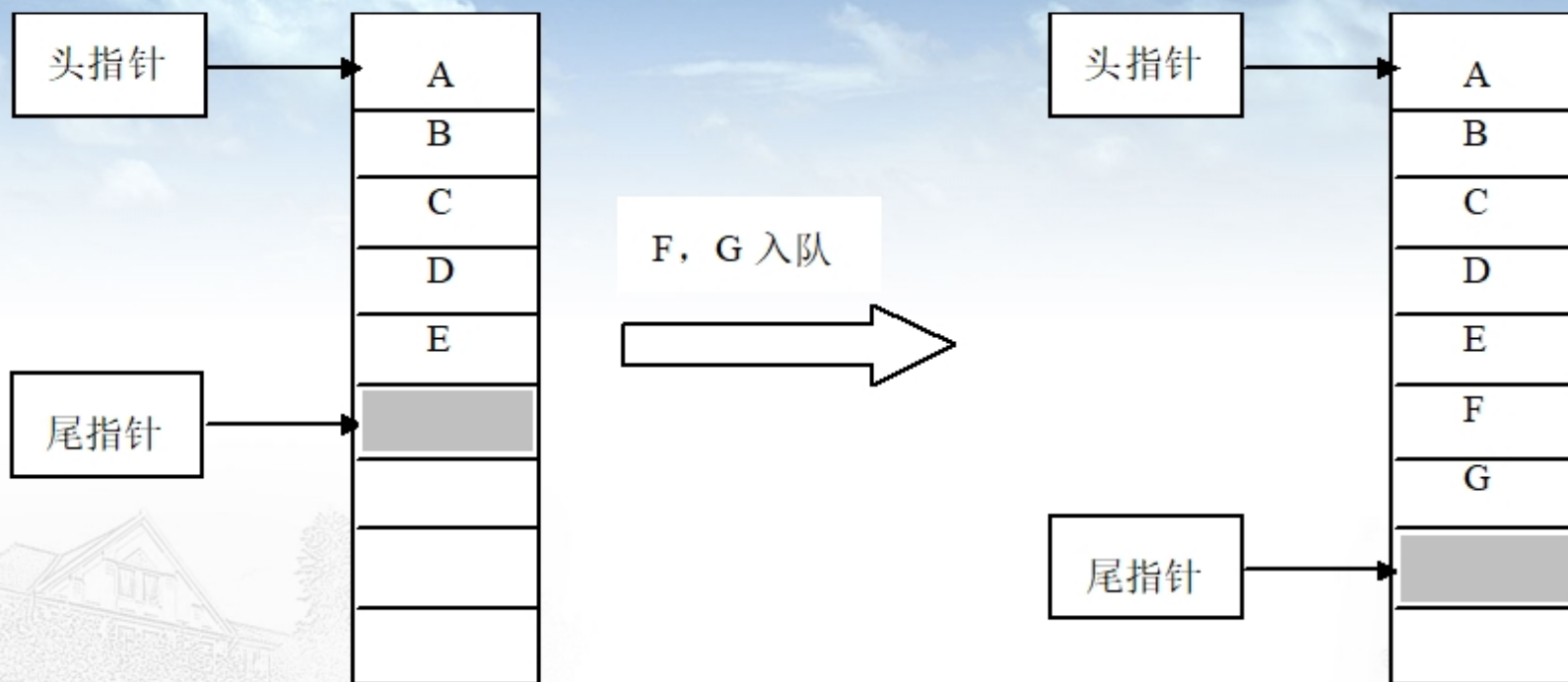
#### ①入队

入队是在队列中插入一个新数据元素的过程，插入在队尾进行，新的元素成为队尾。

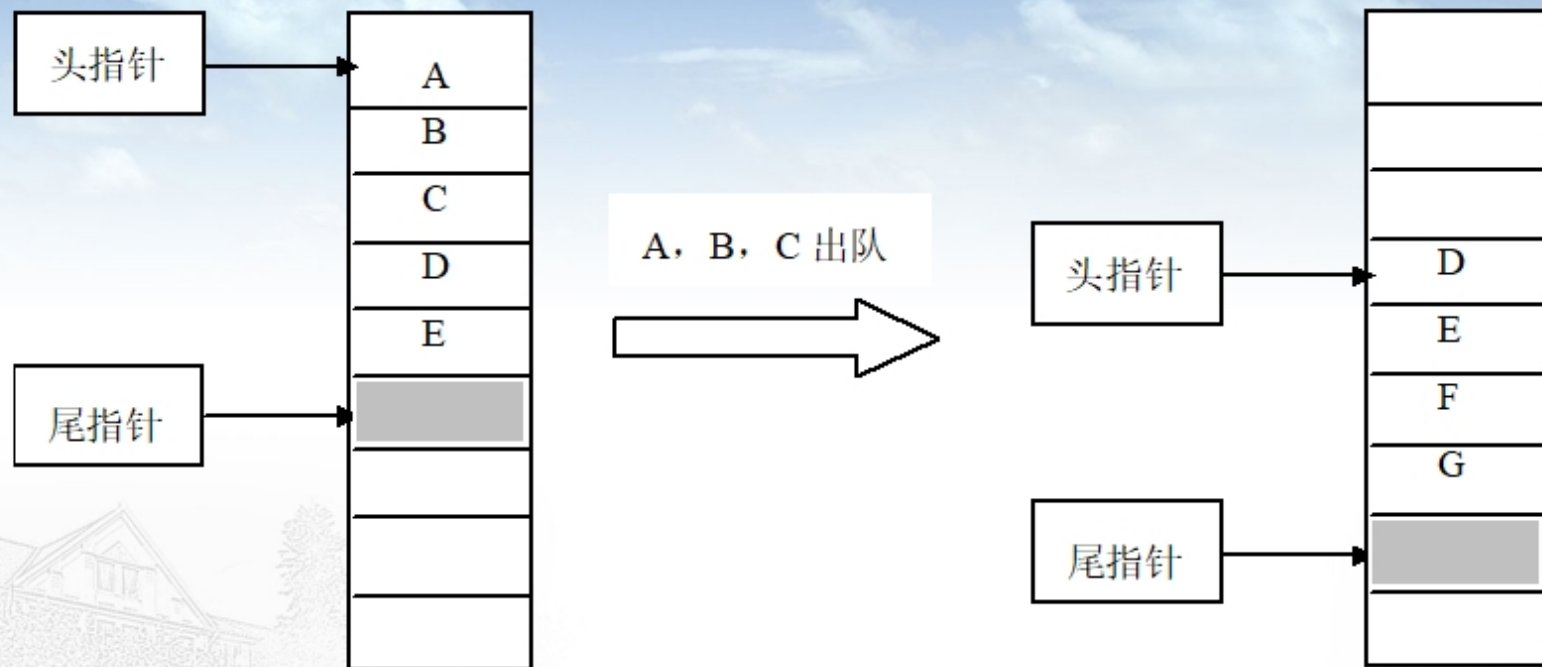
#### ②出队

出队是在队列中删除一个数据元素的过程，删除在队首进行并把出来的数据传递给用户程序。

## 4.3.2 几种典型的数据结构



## 4.3.2 几种典型的数据结构

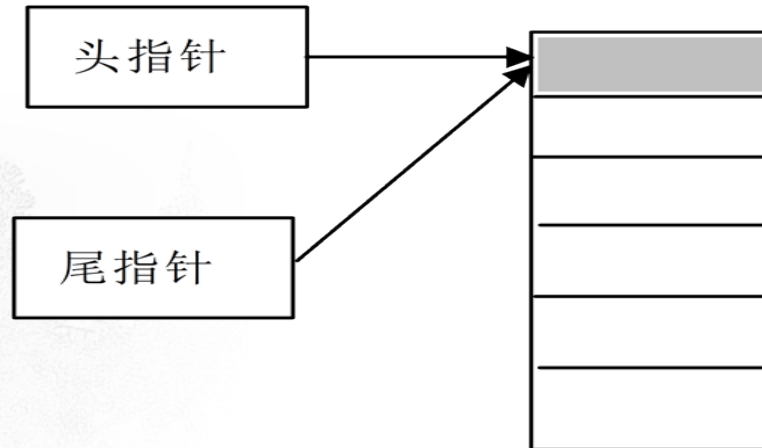




## 4.3.2 几种典型的数据结构

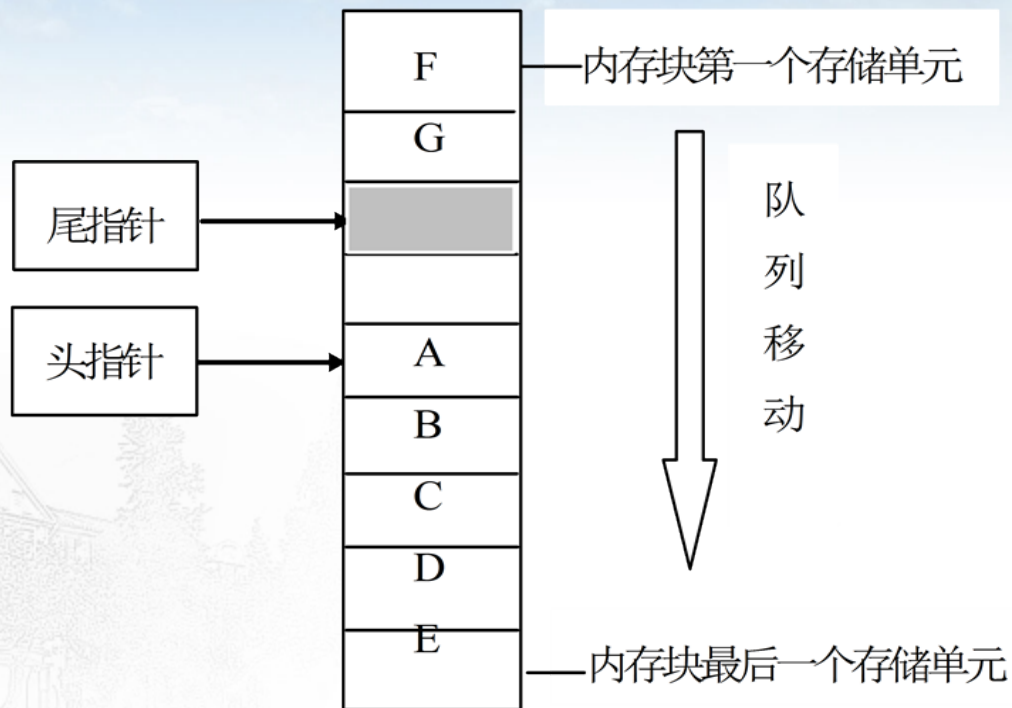
### ③判断

判断操作用来检查队列是否为空，返回结果是一个逻辑值：真或假，如图：



## 4.3.2 几种典型的数据结构

### (3) 循环队列的实现



## 4.3.2 几种典型的数据结构

### 4. 树

#### (1) 定义

在树型结构中，每个数据元素称为一个结点，除了唯一的根结点外，其他每个结点都有且仅有一个父结点，每个元素可以有多个子结点。

树型结构是一种非常重要的非线性数据结构，可以用来描述客观世界中广泛存在的以分支关系定义的层次结构，如各种各样的社会组织结构关系。在计算机领域中，树型结构可以用于大型列表的搜索、源程序的语法结构、人工智能系统等诸多问题。

## 4.3.2 几种典型的数据结构

### (2) 运算

树常见的基本运算有：插入、删除和遍历。

#### ①插入

在树中合适的位置，添加一个节点，通常插入新的节点后，仍然应该保持该树本身所具有的性质。

#### ②删除

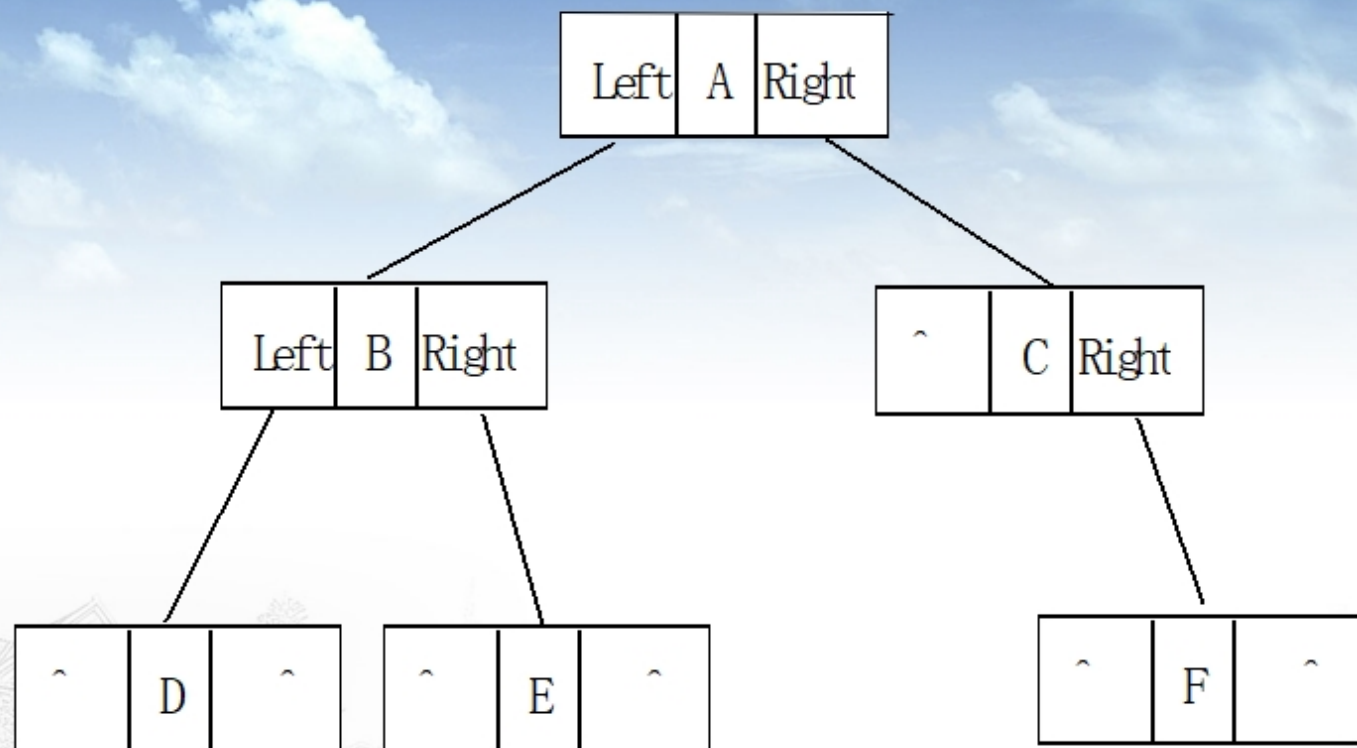
在树中找到满足条件的节点并删除。通常删除节点后，也要保持该树本身所具有的性质。

#### ③遍历

按照某种顺序或规则，对树中的每个节点逐一进行访问的过程。



## 4.3.2 几种典型的数据结构



## 4.3.2 几种典型的数据结构

### 5. 图

#### (1) 定义

- 图形结构是一种比树型结构更复杂的非线性结构。在图形结构中，每个数据元素称为一个顶点，任意两个顶点之间都可能相关，这种相关性用一条边来表示，顶点之间的邻接关系可以是任意的。
- 图在计算机领域有着广泛的应用，可以用来描述计算机网络的拓扑结构，以及图论中的最小生成树等问题。除此以外，图在自然科学、社会科学和人文科学等许多领域也都有着非常广泛的应用。

## 4.3.2 几种典型的数据结构

### (2) 运算

常见的基本运算有：添加顶点、删除顶点、添加边、删除边和遍历图。

#### ①添加顶点

在图中添加新的顶点，新添加的顶点通常是孤立的节点，还没有边连接。

#### ②删除顶点

在图中去掉一个顶点，显然，在去掉一个顶点的同时还应该删除与该顶点所连接的边。

## 4.3.2 几种典型的数据结构

### ③添加边

根据指定的顶点，添加相应的边。

### ④删除边

根据指定的顶点，删除相应的边。

### ⑤遍历图

按照一定的规则，对图中的每个数据顶点逐一进行访问。

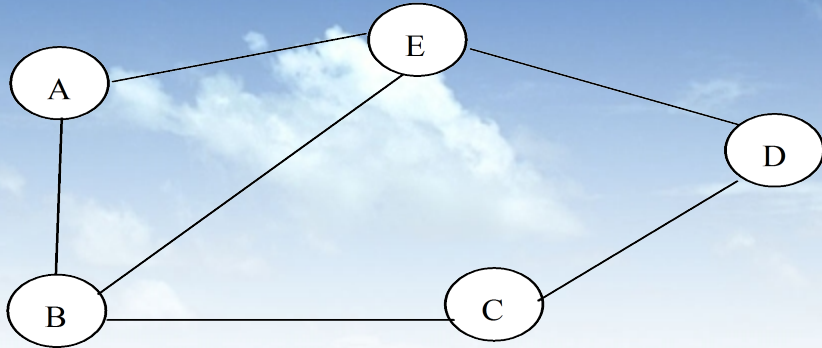


## 4.3.2 几种典型的数据结构

### (3) 实现

图通常用数组和链表两种结构加以实现。对于各个顶点和顶点之间的关系分别采用邻接矩阵和邻接列表来进行描述。

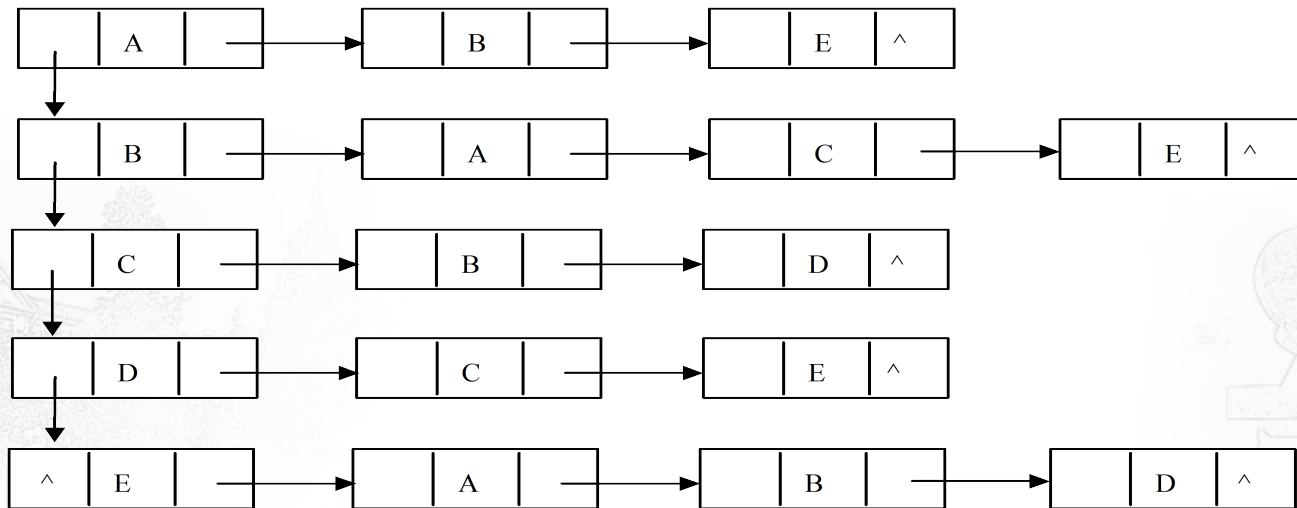
## 4.3.2 几种典型的数据结构



	A	B	C	D	E
A	0	1	0	0	1
B	1	0	1	0	1
C	0	1	0	1	0
D	0	0	1	0	1
E	1	1	0	1	0

(a)

(b)



(c)

### 4.3.3 查找

- 查找是指根据给定的某个值，在查找表中确定一个其关键字等于给定值的记录或数据元素。若表中存在这样的一个记录，则称查找是成功的，此时查找的结果为给出整个记录的信息，或指示该记录在查找表中的位置；若表中不存在关键字等于给定值的记录，则称查找失败，此时查找的结果可给出一个“空”记录或“空”指针。

## 4.3.3 查找

查找的方法主要有顺序查找、二分查找、分块查找、数表的动态查找（二叉排序树查找、平衡二叉树AVL树、B树、B+树）、哈希查找等。

### 1. 顺序查找

- 顺序查找是在一个队列中找出与给定关键字相同数值的具体位置。原理是让关键字与队列中的数从第一个开始逐个比较，直到找出与给定关键字相同的数值为止。

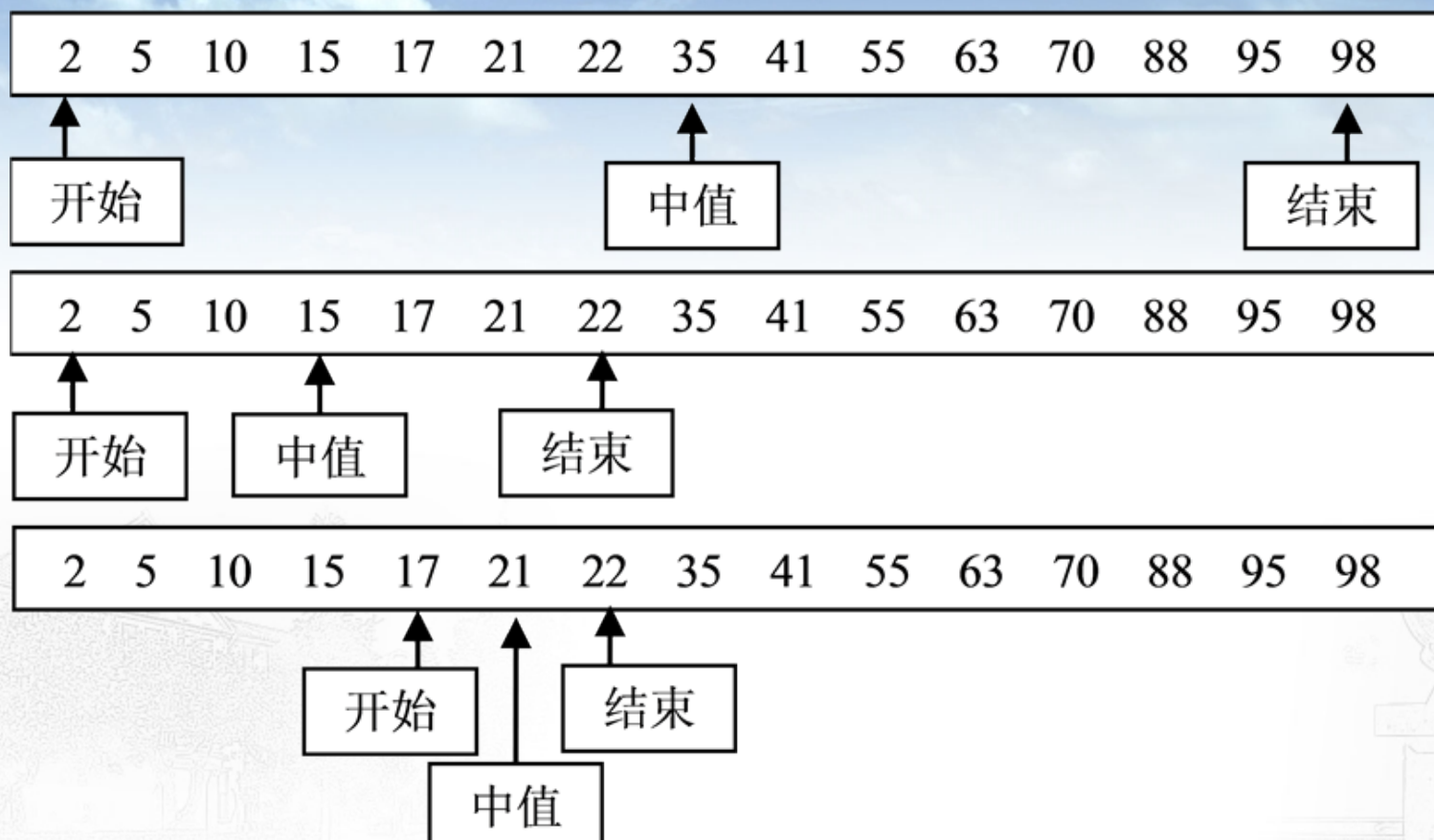


## 4.3.3 查找

### 2. 二分查找

- 二分查找又称折半查找，它是一种效率较高的查找方法。但二分查找必须采用顺序存储结构，且必须按关键字大小有序对给定队列进行排列。
- 二分查找算法的思想是：将表中间位置记录的关键字与查找关键字进行比较，如果两者相等，则查找成功；否则利用中间位置记录将表分成前、后两个子表，如果中间位置记录的关键字小于查找关键字，则进一步查找前一子表（假定队列是从小到大排列），否则进一步查找后一子表。重复以上过程，直至找到满足条件的记录，使查找成功，或直至子表不存在为止，此时查找失败。

### 4.3.3 查找



### 4.3.3 查找

- 优、缺点：二分查找法的优点是比较次数少，查找速度快，平均性能好；其缺点是要求待查表为有序表，且插入、删除困难。因此，二分查找方法适用于不经常变动而查找频繁的有序列表。

## 4.3.3 查找

### 3. 分块查找

- 分块查找又称索引顺序查找，它是顺序查找的一种改进方法。
- 分块的原则是将 $n$ 个数据元素“按块有序”划分为 $m$ 块（ $m \leq n$ ）。每一块中的结点不必有序，但块与块之间必须“按块有序”；即第1块中任一元素的关键字都必须小于第2块中任一元素的关键字；而第2块中任一元素又都必须小于第3块中的任一元素等。
- 分块查找是首先选取各块中的最大关键字构成一个索引表；然后查找分两个部分：先对索引表进行二分查找或已确定待查记录在哪一块中；最后在已确定的块中用顺序法进行查找



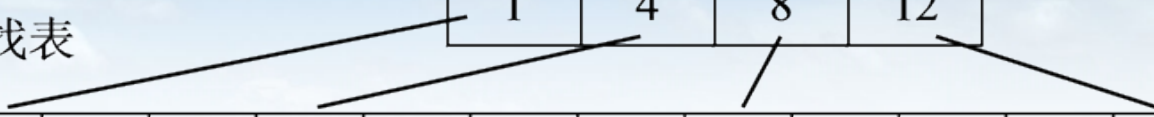
## 4.3.3 查找

索引表

21	59	78	99
1	4	8	12

查找表

3	21	11	24	46	33	59	65	60	78	66	86	99	80	91
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15



## 4.3.4 排序

- 排序是计算机程序设计中的一种重要操作。简单地说，排序就是要整理文件中的记录，使之按关键字递增(或递减)次序排列起来。
- 排序的方法很多，但就其全面性能而言，很难提出一种被认为是最好的方法，每一种方法都有各自的优、缺点，适合在不同的环境(如记录的初始排列状态等)中使用。如果按排序过程中依据的不同原则对内部排序方法进行分类，则可分为直接插入排序、冒泡排序、快速排序等。

## 4.3.4 排序

### 1. 直接插入排序

**直接插入排序**是一种最简单的排序方法，它的基本操作是将一个记录插入到已排好序的有序表中，从而得到一个新的、记录数增1的有序表。

# 直接插入排序

步骤	插入数字	6	30	20	45	15
1	初始值 11					
2	6 11					
3	6 11 30					
4	6 11 20 30					
5	6 11 20 30 45					
6	6 11 15 20 30 45					



## 4.3.4 排序

### 2. 冒泡排序

**冒泡排序**是通过交换两个元素实现的，它的思想是：设有数组 $A[n+1]$ ( $n$ 为序列中元素个数)，第一趟在序列( $A[0]-A[n-1]$ )中从前往后进行两个元素的比较，如后者小，则交换，比较 $n-1$ 次；第一趟排序结束，最大元素被交换到 $A[n-1]$ 中(即沉底)，下一趟排序只要在子序列( $A[]-A[n-2]$ )中进行；如果在某一趟排序中未交换元素，说明子序列已经有序，则不再进行下一趟排序。

# 冒泡排序

初始值: 40 30 15 22 9 50

第一趟: 40 30 15 22 9 50

30 40 15 22 9 50

30 15 40 22 9 50

30 15 22 40 9 50

30 15 22 9 40 50

30 15 22 9 40 **50**

第二趟: 30 15 22 9 40 50

15 30 22 9 40 50

15 22 30 9 40 50

15 22 9 30 40 50

15 22 9 30 **40 50**

第三趟: 15 22 9 30 40 50

15 22 9 30 40 50

15 9 22 30 40 50

15 9 22 **30 40 50**

第四趟: 15 9 22 30 40 50

9 15 22 30 40 50

9 15 **22 30 40 50**

第五趟: 9 15 22 30 40 50

结果: **9 15 22 30 40 50**

## 4.3.4 排序

### 3 . 快速排序

**快速排序**的基本思想是：对任意给定的系列中存在元素  $R$ ，经过一趟排序后，将原序列分割成两个子序列  $(R_{p(0)}, R_{p(1)}, \dots, R_{p(s-1)})$  和  $(R_{p(s+1)}, R_{p(s+2)}, \dots, R_{p(n-1)})$ ，其中前一个子序列中的所有元素的关键字均小于或等于该元素的关键字值  $K_{p(s)}$ ，后一个子序列中元素的关键字均大于或等于  $K_{p(s)}$ 。称该元素  $R = R_{p(s)}$  为分割元素，子序列  $(R_{p(0)}, R_{p(1)}, \dots, R_{p(s-1)})$  为其低端序列， $(R_{p(s+1)}, R_{p(s+2)}, \dots, R_{p(n-1)})$  为其高端序列；很明显，以后只需要对低端和高端分别进行快速排序，直到子序列为空或只有一个元素时结束，最后得到有序序列。

# 快速排序

初始值: 55 67 45 30 20 70 88 11 (将 55 作为关键字)

第一次比较: 11 67 45 30 20 70 88 **55**

第二次比较: 11 45 30 20 70 88 67 **55**

第三次比较: 11 20 45 30 70 88 67 **55**

第一次排序结果: **【11 20 45 30】 55 【70 88 67】**

第二次排序结果: **11 【20 45 30】 55 【67】 70 【88】**

第三次排序结果: **11 20 【45 30】 55 67 70 88**

第四次排序结果: **11 20 30 45 55 67 70 88**



## 4.4 事件驱动程序设计

- 事件驱动程序设计（Event-Driven Programming）是一种程序设计模型，这种模型的程序执行流程是由使用者的动作（如鼠标的按键、键盘的按键动作）或者是由其他程序的信息来决定的。
- 计算机操作系统是事件驱动程序的典型范例。在操作系统的最底层，中断处理器的动作就像是硬件事件的直接处理器，搭配着CPU 执行分配事件规则动作。对软件处理程序而言，基本上操作系统可视为一个事件分配器，传送数据和软件中断给用户自己写的软件处理程序。

# 本章小结

- 本章介绍了程序设计基础知识，包括结构化、面向对象和函数式程序设计方法，数据结构的基本概念和应用实例，事件驱动程序设计。在计算机领域中，程序设计的语言类型和方法技术更新很快，为了紧跟技术发展趋势，必须理解和掌握程序设计的核心思想和算法设计与分析的基本方法。
- 通过本章的学习，读者应该了解结构化程序设计方法和面向对象程序设计方法；熟悉程序设计中常用的几种数据结构，如线性表、栈、队列、树、图等，初步了解事件驱动程序设计的基本步骤。

谢谢大家！