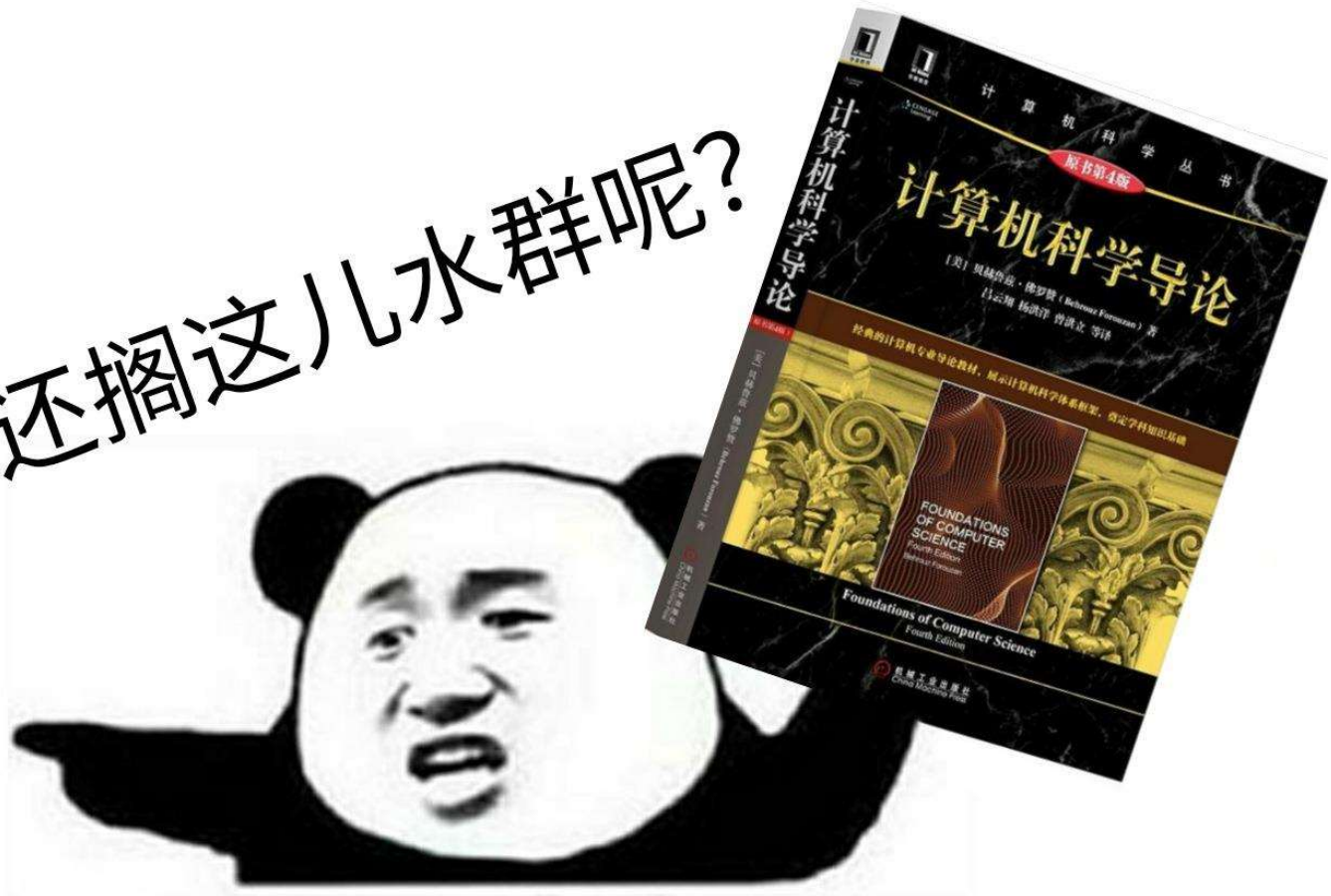
还搁这儿水群呢?



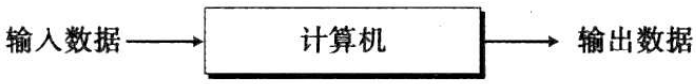
计导会了吗?

第一章 绪 论

一、图灵模型

图灵认为所有的计算都可以在一种特殊的机器上执行。

1 数据处理器



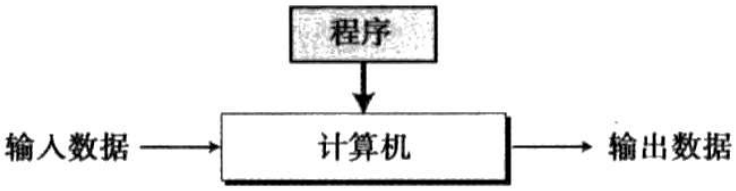
在讨论图灵模型之前，将计算机定义为数据处理器。

计算机是一个接收输入数据、处理数据并产生输出数据的黑盒。

该模型过于宽泛，按照该模型定义，计算器也可以算做一种计算机。

2 可编程数据处理器

图灵模型是一种适用于通用计算机的模型。



该模型增加了额外的元素：程序。

程序是用来告诉计算机对数据进行处理的指令集合。

输出数据依赖于两方面因素，即输入数据和程序。

3 通用图灵机

通用图灵机是对现代计算机的首次描述。

只要提供了合适的程序，该机器就能做任何运算。

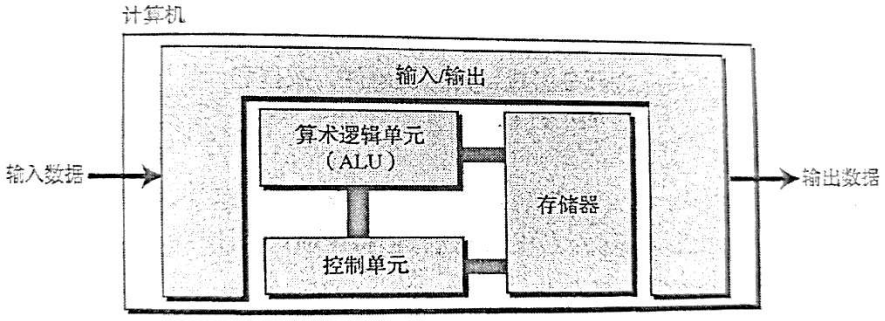
可以证明：通用图灵机和一台功能强大的计算机能够进行同样的运算。

通用图灵机能做任何可计算的运算。

二、冯·诺依曼模型

1 四个子系统

冯·诺依曼模型定义了四个子系统：存储器、算术逻辑单元、控制单元、输入/输出。



存储器在计算机处理过程中用来存储数据和程序。

算术逻辑单元（ALU）是用来进行算术和逻辑运算的地方。

控制单元控制存储器、ALU 和输入/输出子系统。

现代计算机通常将 ALU 和控制单元集成在 CPU 中。

输入子系统负责从计算机外部接收输入数据和程序，输出子系统负责将计算机的处理结果输出到计算机外部。

2 **存储程序的概念**

冯·诺依曼模型要求程序必须存储在存储器中。

早期的计算机只将数据存储在存储器中，执行程序通过操作开关或改变配线完成。

现代计算机的存储器用来存储程序及其响应数据。

程序和数据都以位模式（0 和 1 的序列）存储在存储器中。

3 **指令的顺序执行**

冯·诺依曼模型中的一段程序是由一组数量有限的指令组成

控制单元从内存中提取指令、解释指令、执行指令；指令按照顺序执行

一条指令可能会请求跳转到前面或后面的某个地方去执行，跳转后仍然会顺序执行。

三、计算机组成部分

计算机系统由 3 大部分构成：计算机硬件、数据、计算机软件。

1 **计算机硬件**

计算机硬件基于冯·诺伊曼模型，包含四部分。

2 **数据**

冯·诺依曼模型将计算机定义为数据处理机，接收输入数据，处理并输出相应结果。

存储数据

冯·诺伊曼模型没有定义数据如何存储在计算机中。

对于电子计算机，最好的存储方式应该是电子信号（出现与消失），计算机可以以两种状态之一的形式来存储数据（0 或 1）

文本、图像、声音等数据不能直接存储到计算机内部，必须将它们转换成合适的形式(0、1 序列)才能存储到计算机中

组织数据

数据并不是无序组织的，数据被组织成许多小的单元，再由这些小的单元组织成更大的单元。

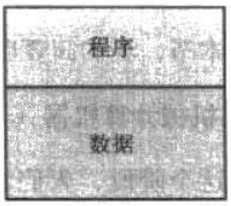
3 **计算机软件**

冯·诺依曼模型的主要特征是程序的概念。早期的计算机没有使用这一模型，但也使用了程序的概念，编程体现为开关的开闭和配线的改变。

冯·诺依曼模型改变了编程的概念。基本的两个方面是存储程序和程序由指令序列构成。

程序必须是存储的　存储器中不仅要存储数据，还要存储程序。

指令的序列　程序必须是有序的指令集。



存储器

算法

编程：了解每条指令能完成的任务，知道将这些指令结合起来完成特定任务。

引入算法：以循序渐进的方式理解问题、分解问题、寻找解决问题的方法，

算法研究的是逐步解决问题的方法。

语言

早期的“编程”直接写指令的二进制模式，难以编写大规模程序，编程非常困难。

计算机科学家们提出用符号代表二进制模式，这样计算机语言的概念诞生了。

软件工程

冯·诺依曼模型并没有定义软件工程。软件工程是指结构化程序的设计和编写。现在，软件工程不仅用来描述完成某一任务的应用程序，还包括程序设计中所要严格遵循的原理和规则。

操作系统

在程序设计过程中，有一些指令序列对所有程序都是公用的、通用的。

早期的操作系统是为程序访问计算机部件提供方便的一种通用管理程序。

现代操作系统已经成为管理计算机软硬件及资源的系统软件。

四、历　史

机械计算机器（1930 年之前）

Pascaline、莱布尼茨之轮、提花织机、分析引擎、具有编程能力的机器

电子计算机的诞生（1930～1950）

ABC、Z1、Mark I、巨人、ENIAC

计算机的诞生（1950 年至今）

第一代计算机、第二代计算机、第三代计算机、第四代计算机、第五代计算机

随着计算机的发展，诞生了新的学科：计算机科学。

复 习 题

Q1-1.定义一个基于图灵模型的计算机。

答：图灵认为所有的计算都可以在一种特殊的机器上执行。他将该模型建立在人们进行计算过程的行为上，并将这些行为抽象到用于计算的机器的模型中，这才真正改变了世界。

Q1-2.定义一个基于冯·诺伊曼模型的计算机。

答：冯·诺伊曼模型定义了计算机的组成，即存储器、算术逻辑单元（ALU）、控制单元和输入/输出四个子系统。

Q1-3.在基于图灵模型的计算机中，程序的作用是什么？

答：基于图灵模型，程序是用来告诉计算机对数据进行处理的指令集合。

Q1-4.在基于冯·诺伊曼模型的计算机中，程序的作用是什么？

答：冯·诺伊曼模型要求程序必须存储在存储器中。现代计算机的存储器用来存储程序及其响应数据。

Q1-5.计算机中有哪些子系统？

答：冯·诺伊曼模型的子系统是存储器、算术逻辑单元（ALU）、控制单元和输入/输出。

Q1-6.计算机中存储器子系统的功能是什么？

答：存储器是用来存储程序和数据的区域。

Q1-7.计算机中 ALU 子系统的功能是什么？

答：算术逻辑单元（ALU）是用来进行算术和逻辑运算的地方。

Q1-8.计算机中控制单元子系统的功能是什么？

答：控制单元控制存储器、ALU 和输入/输出子系统。

Q1-9.计算机中输入/输出子系统的功能是什么？

答：输入子系统负责从计算机外部接收输入数据和程序，输出子系统负责将计算机的处理结果输出到计算机外部。

Q1-10.简述 5 个时代的计算机？

答：第一代计算机（约 1950～1959 年）以商用计算机的出现为特征，只有专家们才能使用。第二代计算机（约 1959～1965 年）使用晶体管替代了真空管。第三代计算机（约 1965～1975 年）开始于集成电路的发明，更加减少了计算机的成本和大小。第四代计算机（约 1975～1985 年）出现了微型计算机。第五代计算机始于 1985 年，见证了笔记本电脑和掌上电脑的诞生、辅助存储媒体（CD-ROM、DVD 等）的改进、多媒体的应用以及虚拟现实的实现。

练 习 题

P1-1.解释为什么计算机不能解决那些计算机外部世界无解决方法的问题。

答：为了解决问题，计算机要遵循一组称为程序的指令，这组指令是根据用纸笔来解题的方法编写的。如果在计算机外部世界没有解决问题的方法，我们就无法编写这样的程序。

P1-2.如果一台小的便宜的计算机可以做大型昂贵的计算机能做的同样事情，为什么人们需要大的呢？

答：根据图灵的观点，任何大型计算机可以解决的问题也可以由小型计算机解决，但大型计算机可能更快地解决这个问题。

P1-3.研究 Pascaline 计算器，看看它是否符合图灵模型。

答：在图灵模型中，计算机由输入数据、输出数据和程序组成。Pascaline 计算器是加减法机，按照图灵模型，它不是计算机，因为它缺少程序部分。

P1-4.研究莱布尼茨之轮，看看它是否符合图灵模型。

答：在图灵模型中，计算机由输入数据、输出数据和程序组成。按照图灵模型，莱布尼茨之轮不是计算机，因为它缺少程序部分。

P1-5.研究雅卡尔提花织机，看看它是否符合图灵模型。

答：在图灵模型中，计算机由输入数据、输出数据和程序组成。在雅卡尔提花织机中，程序（穿孔卡）控制输出（提花织机编织的图案）。因此，它是一台基于图灵模型的计算机。

P1-6.研究查尔斯·巴比奇分析引擎，看看它是否符合冯·诺依曼模型。

答：分析引擎具有冯·诺依曼模型的全部四个组成部分：制造场（ALU）、存储单元（存储器）、操作者（控制单元）和输出单元（输入/输出），但程序没有存储在存储器中。因此，按照冯·诺依曼模型，它不是计算机。

P1-7.研究 ABC 计算机，看看它是否符合冯·诺依曼模型。

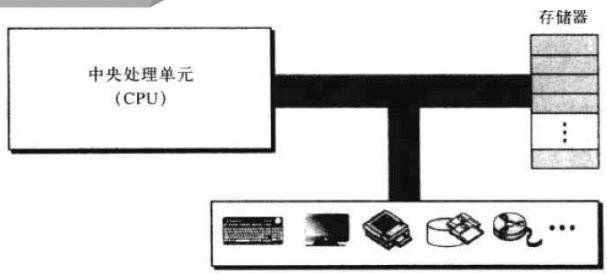
答：第一台基于冯·诺依曼模型的计算机最初被认为是 ENVAC（1950 年制造），但一直存在争议和法庭之争，1973 年地方法院宣布 ENIAC 的专利无效，并认定 ABC（1950 年制造）是第一台计算机。

P1-8.研究并找出键盘起源于哪一代计算机。

答：1964 年，第一台键盘出现，其具有分时多用户系统。这是第二代计算机的结束，第三代计算机的开始。

第五章 计算机组成

一、三大子系统



二、中央处理单元

中央处理单元（CPU）用于数据的运算，包括算术逻辑单元（ALU）、控制单元和寄存器组。

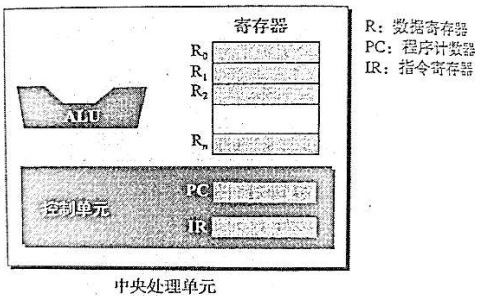
1 **算数逻辑单元**

算术逻辑单元 ALU 对数据进行逻辑、移位和算术运算。

逻辑运算　与、或、非、异或

移位运算　逻辑移位和算术移位

算术运算　整数和实数的算术运算



中央处理单元

2 **寄存器**

寄存器是用来临时存放数据的高速独立的存储单元。

数据寄存器　保存运算的中间结果，可以提高运算速度。

指令寄存器 IR　CPU 负责从内存中逐条取出指令，暂存在指令寄存器中并解释执行。

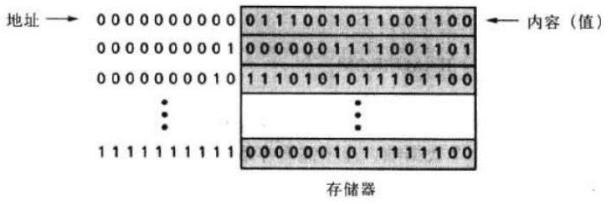
程序计数器 PC　程序计数器保存当前正在执行指令的地址，当前指令执行完成后，自动增 1，指向下一条指令的地址。

3 **控制单元**

控制单元控制各个子系统的操作，控制单元通过向各子系统发送信号来实现控制。

三、主存储器

主存储器是存储单元的集合。每个存储单元用唯一的标识符进行标识，称为**地址**。数据以字为单位在存储器中传入和传出。不同的机器，字可以取 8 位、16 位、32 位或 64 位。如果字是 8 位，称为字节。



1 **地址空间**

在存储器中存取每个字都需要有相应的标识符。程序员通过命名来标识（变量名字），在硬件层次上，每个字都是通过地址来标识。

所有在存储器中标识的独立的地址单元的总数称为地址空间。例如，一个 64KB、字长为 1 字节的内存的地址空间的范围为 0～65535。

作为位模式的地址　地址用无符号二进制整数定义。

【例】一台计算机有 32MB 内存，需要多少位来寻址内存中的任意一个字节？

解　32MB＝2^{25}B，需要 25 位来标识每一个字节。

【例】一台计算机有 128MB 内存，计算机字长为 8 字节，需要多少位来寻址内存中的任意一个单字？

解　128MB＝2^{27}B＝2^{24}×8B，需要 24 位来标识每一个字。

2 存储器的类型

随机存取存储器（RAM）是主存的主要组成部分。RAM 具有可随机读写、易失性特点。

静态 RAM（SRAM），使用触发器门电路保存数据，速度快但价格昂贵。

动态 RAM（DRAM），使用电容存储数据，充电为 1，放电状态为 0。需要周期性刷新，速度慢但价格便宜。

只读存储器（ROM）的内容由制造商写入，具有只读、非易失性特点。

可编程只读存储器（PROM）出厂时空白，借助特殊设备可一次写入内容。

可擦除可编程只读存储器（EPROM），借助紫外线装置可擦除内容，然后可重新写入。

电可擦除可编程只读存储器（EEPROM），使用电子脉冲即可擦除和编程写入。

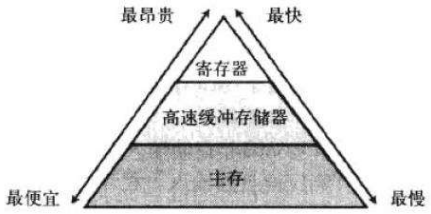
3 存储器的层次结构

存取速度快的存储器价格昂贵，通常采用层次配置：

对速度要求苛刻的场合，配置少量高速存储器，如 CPU 中的寄存器；

用适量中速的存储器存储需要经常访问的数据；

用大量低速存储器存储大量不经常访问的数据。



4 高速缓冲存储器

高速缓冲存储器的存取速度比主存快，但比 CPU 及其内部的寄存器要慢，置于 CPU 和内存之间。

基本原理：CPU 要存取主存中的一个字时，首先检查高速缓存。如果存在，CPU 直接存取；如果不存在，CPU 将从主存中复制一份从需要读取的字开始的数据块覆盖高速缓存中的内容，CPU 从高速缓存中存取数据。

高速缓存可以提高整体访问性能。

四、输入/输出子系统

1 非存储设备

非存储设备提供 CPU 与外界的通信，但不能存储信息，如键盘、监视器、打印机等。

2 存储设备

存储设备可存储大量信息以备后用，速度比主存慢但价格便宜。存储的信息不易丢失，称为辅助存储设备。

磁介质存储设备　使用磁性来存储数据，磁性介质的磁化表示 1，消磁表示 0

磁盘由多张盘片叠加而成，每个盘片上的读/写磁头控制数据的读写。

数据被存储在磁盘表面。

每个盘面被划分为磁道，磁道之间有内部间隔。

每个磁道被划分为若干扇区，扇区之间也有内部间隔。

磁盘是随机存储设备。

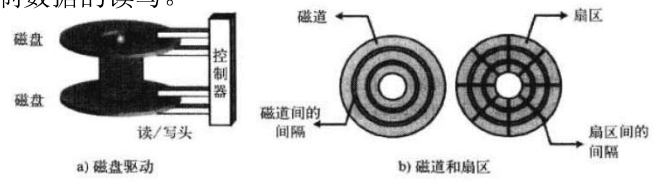


图5-6　磁盘

磁带表面划分为 9 个磁道，每道上每个点存储 1 个位。垂直的切面 9 个点存储一个字节，1 位用于错误检测。

磁带是顺序存取设备，虽然磁带表面可能会分成若干块，但缺乏寻址机制；要想读取指定的块，必须顺序通过其前面的所有块。

磁带速度比磁盘慢，但非常便宜，常用于大量数据的备份。

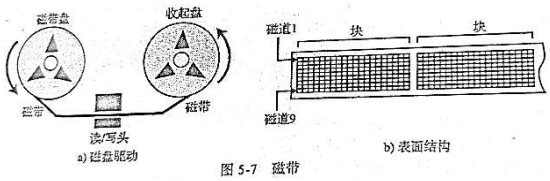


图 5-7　磁带

光存储设备　采用光（激光）技术来存储和读取数据。

使用 CD 主要用来保存音频信息现在，同样的技术用于存储计算机信息。

包括 CD-ROM（只读光盘）、CD-R（可刻录光盘）、CD-RW（可重写光盘）、DVD（多功能光盘）。

五、子系统的互连

1 CPU 和存储器的连接

图5-12　使用三种总线连接CPU和存储器

数据总线　用于传送数据。线的根数由字长决定，如字长为 4 个字节，需要 32 根数据总线。

地址总线　要访问主存中的某个字，首先通过地址总线传送地址。线的根数取决于存储空间的大小，如存储器容量为2^n个字，需要 n 根地址总线。

控制总线　用于发送控制命令。线的根数取决于计算机所需要的控制命令总数，如计算机有2^m条控制命令，需要 m 根控制总线。

2 I/O 设备的连接

CPU 与内存通过总线连接，而输入/输出设备由于速度慢，不能直接接入总线。输入/输出设备借助控制器或接口连接到总线上。

串行控制器只有一根线连接到设备上，每次只能传输一位数据；并行控制器有数根线连接到设备上，一次能传输多个位。

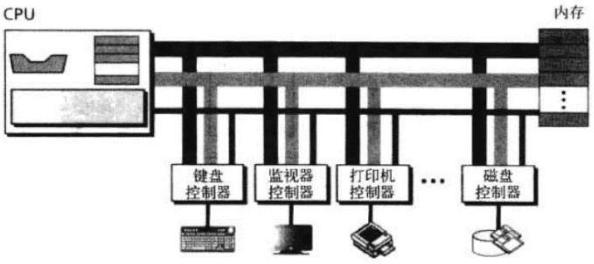


图5-13　I/O设备与总线的连接

SCSI（小型计算机系统接口）　8、16 或 32 线的并行接口。提供菊花链连接，两端需要终结器。

火线　是一种高速串行接口，通过菊花链或树形连接可连接多达 63 个设备。

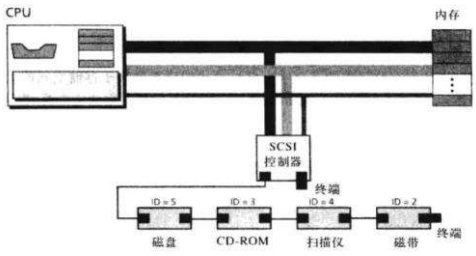


图5-14　SCSI控制器

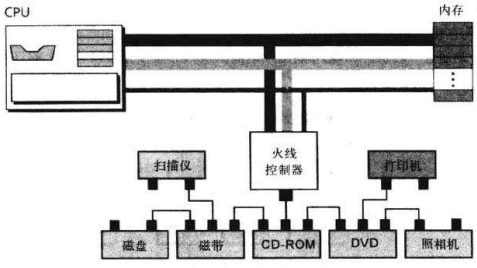


图5-15　火线控制器

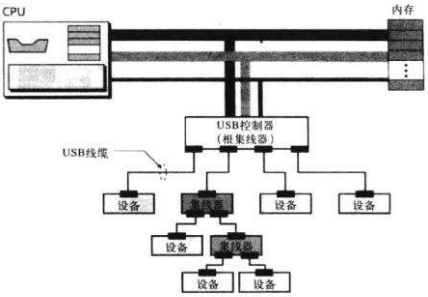


图5-16　USB控制器

USB（通用串行总线）　是一种串行控制器，用来连接低速和高速设备。支持多达 127 个设备接入到 USB 控制器上，支持热交换（不用关机即可插拔）。

3 输入/输出设备的寻址

通常 CPU 使用相同的总线在主存和输入/输出设备之间读写数据。唯一的区别在于使用不同的指令。

如果指令涉及主存中的字，数据在主存和 CPU 之间传送；如果指令涉及输入/输出设备，则数据在 CPU 和 I/O 设备之间传送。

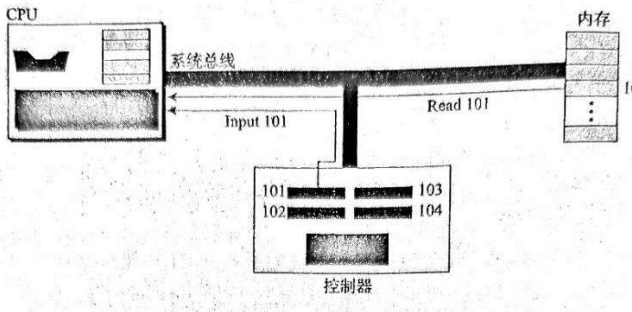


图 5-17　I/O 独立寻址

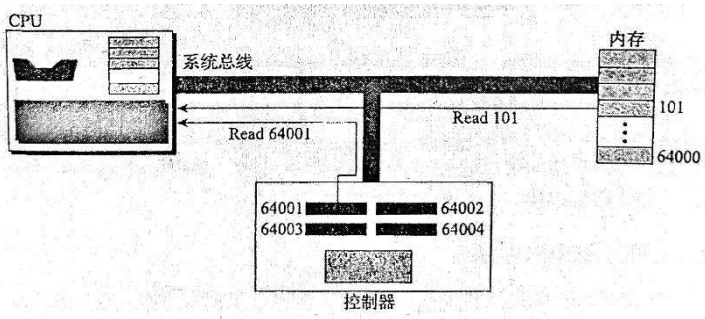


图 5-18　I/O 存储器映射寻址

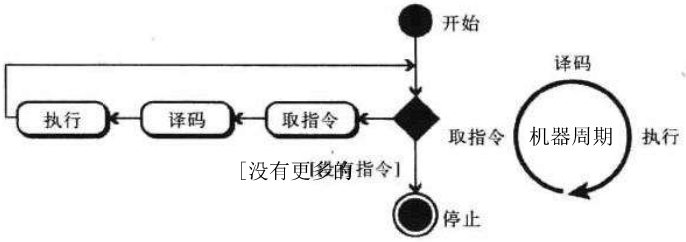
I/O 独立寻址　读写主存和读写输入/输出设备使用不同的指令。每个输入/输出设备有自己的地址。

I/O 存储器映射寻址　将输入/输出设备中的每个寄存器看成主存中的字，统一寻址，使用相同的指令。

六、程序执行

1 机器周期

CPU 利用重复的机器周期执行指令，简化的周期包括取指令、译码、执行。



取指令　由控制单元将下一条要执行的指令复制到指令寄存器中，程序计数器自动加 1 指向内存中的下一条指令。

译码　控制单元对指令寄存器中的指令进行译码，产生系统可以执行的操作码。

执行　控制单元发送命令到某个部件完成操作。如将两个寄存器中的内容相加并将结果保存到输出寄存器中；从内存中加载（读）数据项。

2 输入/输出操作

计算机需要通过命令把数据从 I/O 设备传输到 CPU 和内存。因为输入/输出设备的运行速度比 CPU 要慢很多，CPU 的操作必须和输入/输出设备同步(CPU 等待输入/输出设备)。有三种基本的同步方法：

程序控制输入/输出　CPU 等待 I/O 设备；CPU 查询设备状态，效率非常低。

中断控制输入/输出　I/O 设备开始工作后，CPU 转向其它工作，I/O 完成后，通知(中断)CPU。

直接存储器存取（DMA）　适合高速 I/O 设备(如磁盘)和主存之间直接传输大量数据块(不需要通过 CPU 的数据传输)。由 DMA 控制器负责数据的传输。

七、不同的体系结构

1 CISC（复杂指令集计算机）

设计策略是使用大量的指令，包括复杂指令。

2 RISC（精简指令集计算机）

设计策略是体系结构的设计策略是使用少量的指令完成最少的简单操作。

3 流水线

一条指令完成取指令阶段，进入译码阶段，下一条指令即可开始取指令。

通过流水线技术可以提高系统的吞吐量（单位时间内完成的指令总数）。

问题：遇到转移指令时，预取并执行的指令将作废。

4 并行处理

传统计算机有单个控制单元、单个算术逻辑单元、单个内存单元；并行处理系统可以具有多个控制单元、多个算术逻辑单元和多个内存单元。根据数据流和指令流，并行系统划分为 4 类：

SISD（单指令流单数据流）　有一个控制单元，一个算术逻辑单元，一个内存单元。指令顺序执行，每条指令可以存取数据流中的一个或多个数据项。

SIMD（单指令流多数据流）　有一个控制单元，多个算术逻辑单元，一个内存单元。所有处理器单元从控制单元接收相同的指令，但处理不同的数据项。

MISD（多指令流单数据流）　没有真正实现过。

MIMD（多指令流多数据流）　同时执行多个任务，真正意义上的并行处理。

八、简单计算机

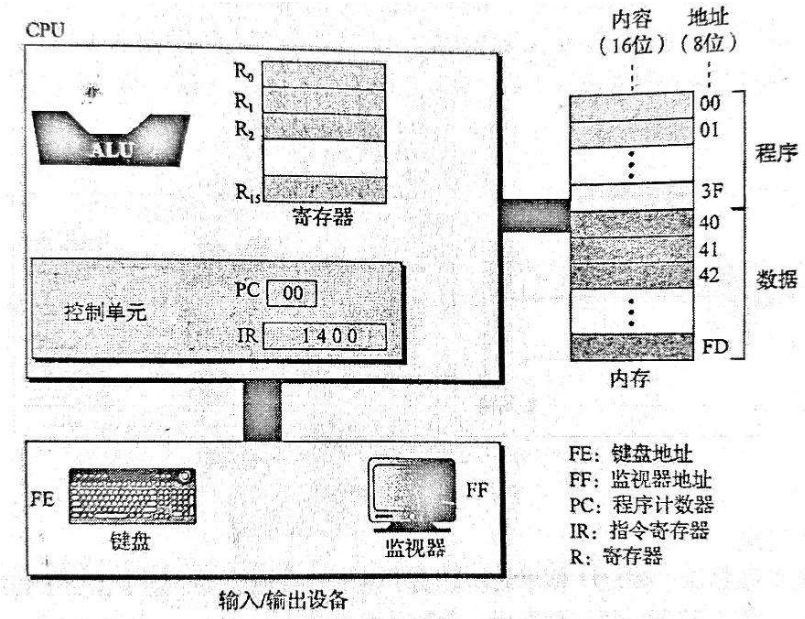


图 5-30　简单计算机的组成

1 CPU

数据寄存器、算术逻辑单元和控制单元。

16 个 16 位数据寄存器，地址为 0～F。指令寄存器 IR，16 位。程序计数器 PC，8 位。

2 主存

256 个 16 位（字长为 2 个字节）存储单元，地址 00～FF。00～3F 存储指令，40～FD 存储数据。

3 输入/输出子系统

键盘和监视器，统一编址，键盘为 FE，监视器 FF。

4 指令集

简单计算机具备 16 条指令，每个指令由操作码和操作数构成。

操作码指明执行的操作类型（用 4 位表示）

操作数按 4 位划分，包含操作数或操作数的地址（寄存器地址、内存地址、输入输出设备地址）

不是所有指令都需要 3 个操作数，不需要的操作数填充为 0。

5 处理指令

简单计算机使用机器周期：取指令、译码和执行。

复 习 题

Q5-1.计算机由哪三个子系统组成？

答：中央处理单元（CPU）、主存储器和输入/输出。

Q5-2.CPU 由哪几个部分组成？

答：算术逻辑单元（ALU）、控制单元和寄存器组。

Q5-3.ALU 的功能是什么？

答：对数据进行逻辑、移位和算术运算。

Q5-4.控制单元的功能是什么？

答：控制各个子系统的操作。

Q5-5.主存的功能是什么？

答：在程序被执行时存储数据和程序。

Q5-6.定义 RAM、ROM、SRAM、DRAM、PROM、EPROM 和 EEPROM。

答：RAM 是随机存取存储器，可以由用户读写。

ROM 是只读存储器。其内容是由制造商写进去的，用户只能读但不能写。

SRAM 是静态 RAM，使用触发器门电路来保存数据。

DRAM 是动态 RAM，使用电容来保存数据。

PROM 是可编程只读存储器，可由用户使用专用设备进行编程。

EPROM 是可擦除 PROM，用户得用一种可以发出紫外光的特殊仪器对其擦写。

EEPROM 是电可擦除 PROM，对它的编程和擦除无须从计算机上拆下来。

Q5-7.高速缓冲存储器的作用是什么？

答：高速缓冲存储器使 CPU 能快速访问主存中的部分数据。

Q5-8.描述一下磁盘的物理组成。

答：磁盘由一个或多个带有薄磁膜的盘片和每个盘片表面的读/写头组成。

Q5-9.磁盘和磁带表面是怎样组织的？

答：磁盘的表面被划分成圆环形的磁道，每个磁道又分成若干个扇区。

磁带的宽度分为 9 个磁道，长度可以分为若干块。

Q5-10.比较分析 CD-R、CD-RW 和 DVD。

答：用户可以把数据存储在 CD-R 和 CD-RW 上，CD-RW 的优势是可以擦写。比起 CD-ROM（650MB），DVD 使用更小的坑和纹间表面来存储更多的数据（4.7GB 到 17GB）。

Q5-11.比较分析 SCSI、火线和 USB 控制器。

答：SCSI 是一种提供设备和总线之间菊花链连接的并行接口。

火线是一种用包的形式传送数据的高速串行接口，可以使用菊花链或树形连接。

多个设备可以连接到一个 USB 控制器上。

Q5-12.比较分析两种 I/O 设备寻址的方法。

答：I/O 独立寻址　　使用不同的指令访问内存和 I/O 设备。

I/O 存储器映射寻址使用相同的指令访问内存和 I/O 设备。

Q5-13.比较分析三种同步 CPU 和 I/O 设备的方法。

答：在程序控制输入/输出中，CPU 等待 I/O 设备；CPU 浪费大量时间查询设备状态。

在中断控制输入/输出中，　I/O 设备通过中断通知 CPU。

在直接存储器存取（DMA）中，CPU 将其 I/O 请求发送到负责管理整个传送过程的 DMA 控制器。

Q5-14.比较分析 CISC 体系结构和 RISC 体系结构。

答：CISC（复杂指令集计算机）用大量指令来执行机器层面的命令。这使得 CPU 和控制单元的电路非常复杂。

RISC（精简指令集计算机）使用少量指令，复杂指令用简单指令集来完成。

Q5-15.描述流水线及其作用。

答：流水线允许不同周期的不同阶段同时完成。流水线可以增加计算机的吞吐量。

Q5-16.描述并行处理及其作用。

答：一台计算机可以有多个控制单元、多个 ALU 和多个存储单元来并行执行多条指令。并行处理增加了计算机的吞吐量。

练 习 题

P5-1 一台计算机有 64 MB 的内存，每个字长为 4 字节。那么在存储器中对每个字寻址需要多少位？

解：64 MB＝16×4 MB＝16 M 字 ＝2^{24} 字，故需要 24 位对每个字寻址。

P5-2 如果屏幕有 24 行， 每行 80 个字符，则需要多少字节的内存用于存储全屏的数据？假定系统使用 ASCII 码，每个 ASCII 字符占 1 字节。

解：需要 24×80＝1920 字节。

P5-3 假如一台计算机有 16 个数据寄存器(R0～R15)、 1024 个字的存储空间以及 16 种不同的指令(如 add、subtract 等)，那么下面这条指令最少需要占多少位空间？add M R2

解：需要 4 位来确定指令($2^4 = 16$)，4 位来寻址寄存器($2^4 = 16$)，10 位来在内存中寻址一个字 ($2^{10} = 1024$)，

故指令需要占 4+4+10＝18 位空间。

P5-4 在 P5-3 题中，如果数据和指令使用相同的字长，那么每个数据寄存器大小是多少？解：18 位。

P5-5 在 P5-3 题中，计算机中的指令寄存器大小是多少？解：18 位。

P5-6 在 P5-3 题中，计算机中的程序计数器大小是多少？解：10 位。

P5-7 在 P5-3 题中，数据总线为多少位？解： 18 位。

P5-8 在 P5-3 题中，地址总线为多少位？解：10 位。

P5-9 在 P5-3 题中，控制总线最少需要多少位？解：4 位。

P5-10 计算机使用 I/O 独立寻址。内存为 1024 个字。如果每个控制器包括 16 个寄存器，那么计算机可以存取多少个控制器？

解：内存为 1024 字，可以用 $\log_2 1024 = 10$ 位寻址。由于计算机使用 I/O 独立寻址，最多可用 10 位来寻址 I/O 寄存器，即最多可以有 $2^{10} = 1024$ 个寄存器。每个控制器包括 16 个寄存器，则在这个系统中最多可以存取 1024÷16=64 个控制器。

P5-11 计算机使用 I/O 存储器映射寻址。地址总线为 10 条(10 位)。如果内存为 1000 个字，那么计算机可以存取多少个四位寄存器控制器？

解：地址总线有 10 条，说明其可以寻址 $2^{10} = 1024$ 字。而内存只有 1000 字，由于计算机使用 I/O 存储器映射寻址，故控制器可用 1024－1000＝24 字。每个控制器包括 4 个寄存器，则在这个系统中最多可以存取 24÷4=6 个控制器。

P5-12 使用 5.8 节中的简单计算机指令集，编写执行下列计算的程序代码：

D←A+B+C

A、B、C 和 D 是二进制补码格式的整数，用户输入 A、B 和 C 的值，D 的值显示在监视器上。

解：第一列给出指令地址供参考，不是代码的一部分。

$(00)_{16}$ $(1FFE)_{16}$ // **RF ← MFE, Input A from keyboard to RF**
$(01)_{16}$ $(240F)_{16}$ // **M40 ← RF, Store A in M40**
$(02)_{16}$ $(1FFE)_{16}$ // **RF ← MFE, Input B from keyboard to RF**
$(03)_{16}$ $(241F)_{16}$ // **M41 ← RF, Store B in M41**
$(04)_{16}$ $(1FFE)_{16}$ // **RF ← MFE, Input C from keyboard to RF**
$(05)_{16}$ $(242F)_{16}$ // **M42 ← RF, Store C in M42**
$(06)_{16}$ $(1040)_{16}$ // **M40 ← R0, Load A from M40 to R0**
$(07)_{16}$ $(1141)_{16}$ // **M41 ← R1, Load B from M41 to R1**
$(08)_{16}$ $(1242)_{16}$ // **M42 ← R2, Load C from M42 to R2**
$(09)_{16}$ $(3310)_{16}$ // **R3 ← R1 + R0, Add A to B and store in R3**
$(0A)_{16}$ $(3432)_{16}$ // **R4 ← R3 + R2, Add C to (A + B) and store in R4**
$(0B)_{16}$ $(2434)_{16}$ // **M43 ← R4, Store The result in M43**
$(0C)_{16}$ $(1F43)_{16}$ // **RF ← M43, Load the result to RF**
$(0D)_{16}$ $(2FFF)_{16}$ // **MFF ← RF, Send the result to the monitor**
$(0E)_{16}$ $(0000)_{16}$ // **Halt**

P5-13 使用 5.8 节中的简单计算机指令集，编写编写执行下列计算的程序代码：

B←A+3

A 和 3 是二进制补码格式的整数，用户输入 A 的值，B 的值显示在监视器上。(提示：使用加 1 指令)

解：第一列给出指令地址供参考，不是代码的一部分。

$(00)_{16}$ $(1FFE)_{16}$ // **RF ← MFE, Input A from keyboard to RF**
$(01)_{16}$ $(240F)_{16}$ // **M40 ← RF, Store A in M40**
$(02)_{16}$ $(1040)_{16}$ // **M40 ← R0, Load A from M40 to R0**
$(03)_{16}$ $(A000)_{16}$ // **R0 ← R0 + 1, Increment A**
$(04)_{16}$ $(A000)_{16}$ // **R0 ← R0 + 1, Increment A**
$(05)_{16}$ $(A000)_{16}$ // **R0 ← R0 + 1, Increment A**

$(06)_{16}$ $(2410)_{16}$ // **M41 ← R0, Store The result in M41**
$(07)_{16}$ $(1F41)_{16}$ // **RF ← M41, Load the result to RF**
$(08)_{16}$ $(2FFF)_{16}$ // **MFF ← RF, Send the result to the monitor**
$(09)_{16}$ $(0000)_{16}$ // **Halt**

P5-14 使用 5.8 节中的简单计算机指令集，编写执行下列计算的程序代码：

B←A－2

A 和 2 是二进制补码格式的整数，用户输人 A 的值，B 的值显示在监视器上。(提示：使用减 1 指令)

解：第一列给出指令地址供参考，不是代码的一部分。

$(00)_{16}$ $(1FFE)_{16}$ // **RF ← MFE, Input A from keyboard to RF**
$(01)_{16}$ $(240F)_{16}$ // **M40 ← RF, Store A in M40**
$(02)_{16}$ $(1040)_{16}$ // **M40 ← R0, Load A from M40 to R0**
$(03)_{16}$ $(B000)_{16}$ // **R0 ← R0 − 1, Decrement A**
$(04)_{16}$ $(A000)_{16}$ // **R0 ← R0 − 1, Decrement A**
$(05)_{16}$ $(2410)_{16}$ // **M41 ← R0, Store The result in M41**
$(06)_{16}$ $(1F41)_{16}$ // **RF ← M41, Load the result to RF**
$(07)_{16}$ $(2FFF)_{16}$ // **MFF ← RF, Send the result to the monitor**
$(08)_{16}$ $(0000)_{16}$ // **Halt**

P5-15 使用 5.8 节中的简单计算机指令集，为程序编写代码，该程序完成 *n* 个从键盘输入的整数的相加，并显示它们的和。你首先需要输入 *n* 的值。(提示：使用减 1 指令和跳转指令，重复 *n* 次加运算)

解：第一列给出指令地址供参考，不是代码的一部分。

$(00)_{16}$ $(10FE)_{16}$ // **RF ← MFE, Input 0 from keyboard to R0**
$(01)_{16}$ $(11FE)_{16}$ // **RF ← MFE, Input n from keyboard to R1**
$(02)_{16}$ $(12FE)_{16}$ // **RF ← MFE, Input the first number to R2**
$(03)_{16}$ $(B100)_{16}$ // **R1 ← R1 − 1 Decrement R1**
$(04)_{16}$ $(B100)_{16}$ // **R1 ← R1 − 1 Decrement R1**
$(05)_{16}$ $(13FE)_{16}$ // **RF ← MFE, Input the next number to R3**
$(06)_{16}$ $(3223)_{16}$ // **R2 ← R2 + R3 Add R3 to R2 and store in R2**
$(07)_{16}$ $(D104)_{16}$ // **If R0 ≠ R1 then PC = 04, otherwise continue**
$(08)_{16}$ $(2FF2)_{16}$ // **MFF ← R2, Send the result to the monitor**
$(09)_{16}$ $(0000)_{16}$ // **Halt**

P5-16 使用 5.8 节中的简单计算机指令集，为程序编写代码，该程序接收从键盘来的两个整数，如果第一个整数为 0，程序把第二个整数加 1；如果第一个整数是 1，程序把第二个整数减 1。第一个整数必须是 0 或 1，否则程序失败。程序显示加 1 或减 1 的结果。

解：第一列给出指令地址供参考，不是代码的一部分。

$(00)_{16}$ $(10FE)_{16}$ // **RF ← MFE, Input 0 from keyboard to R0**
$(01)_{16}$ $(11FE)_{16}$ // **RF ← MFE, Input the first integer to R1**
$(02)_{16}$ $(12FE)_{16}$ // **RF ← MFE, Input the second integer to R2**
$(03)_{16}$ $(D108)_{16}$ // **If R0 ≠ R1 then PC = 08, otherwise continue**
$(04)_{16}$ $(A200)_{16}$ // **R2 ← R2 + 1**
$(05)_{16}$ $(2FF2)_{16}$ // **MFF ← R2, Send the result to the monitor**
$(06)_{16}$ $(A100)_{16}$ // **R1 ← R1 + 1**
$(07)_{16}$ $(D10A)_{16}$ // **If R0 ≠ R1 then PC = 0A, otherwise continue**
$(08)_{16}$ $(B200)_{16}$ // **R2 ← R2 − 1**
$(09)_{16}$ $(2FF2)_{16}$ // **MFF ← R2, Send the result to the monitor**
$(0A)_{16}$ $(0000)_{16}$ // **Halt**

第七章 操作系统

一、引 言

1 操作系统

操作系统是介于计算机**硬件和用户**(程序和人)之间的**接口**，它使得其他程序更加方便有效地运行，并能方便地对计算机硬件和软件资源进行访问。

操作系统的两个主要设计目标：有效地使用硬件、容易地使用资源。

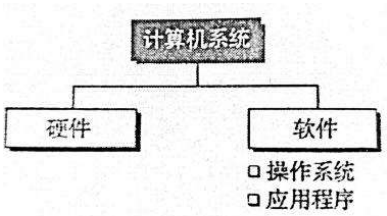


图 7-1 计算机系统

2 自举过程

只在 ROM 中存放一小部分程序（自举程序），加电后执行该程序，该程序的职责是负责将操作系统装入 RAM 中，载入完成后修改程序计数器并执行真正的操作系统部分。

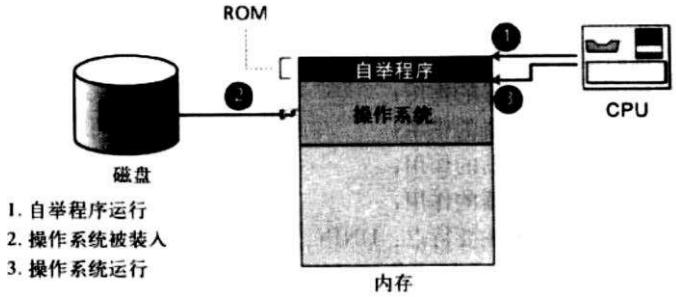


图7-2 自举过程

二、演 化

1 批处理系统

设计于上世纪 50 年代，目的是控制大型计算机。

用穿孔卡片输入数据，用行式打印机输出结果，用磁带作为辅助存储介质。

运行的程序称为作业。

2 分时系统

多道程序：多个作业同时装入内存。

分时技术：资源可以被多个作业共享，每个作业分到一段时间轮流使用资源。

采用分时技术的多道程序极大地改进了计算机使用效率。依然是串行处理的

多道和分时要求操作系统可以调度：给不同的程序分配资源并决定哪个程序什么时候使用哪一种资源。

在内存中等待分配资源的程序称为进程。

3 个人系统

适合个人计算机的操作系统，如 DOS。

4 并行系统

在同一计算机中安装多个 CPU，每个 CPU 可以执行一个程序或一个程序的一部分。

可达到更高的速度和效率。多个任务可以并行处理

5 分布式系统

作业可以由远隔千里的多台计算机共同完成。

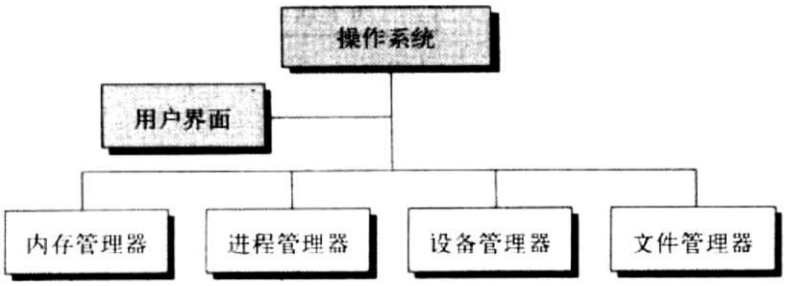
程序可以在一台计算机上运行一部分，而在另一台计算机上运行另一部分，它们通过互联网连接。

6 实时系统

实时系统指在特定时间限制内完成任务。

常应用在交通控制、病人监控或军事控制系统中等领域。

三、组成部分



1 用户界面

每个操作系统都有用户界面，用户界面是用来接收用户（进程）的输入并向操作系统解释这些请求的程序。

一些操作系统（如 UNIX）的用户界面被称为命令解释程序；

另外一些操作系统的用户界面称为窗口，是由菜单驱动的 GUI（图形用户界面）。

2 内存管理器

内存管理是操作系统的一个重要职责。

单道程序 一小部分内存用来装载操作系统，大多数内存专用于装载单一的程序。

整个程序被装入内存运行，运行结束后由下一个程序取代。

问题：大程序无法加载；

单道执行，一个程序运行时，其它程序不能运行；

CPU 等待 I/O 操作，CPU 利用率低。

多道程序 同一时刻可以装入多个程序并同时执行，CPU 轮流为它们服务。多个程序在内存中并存。

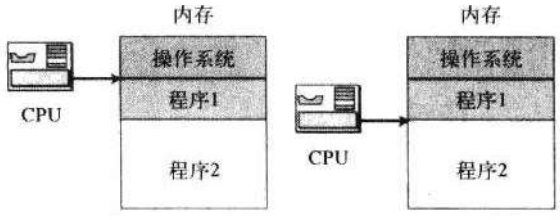
分区调度 内存预先被划分为不定长的若干分区，每个分区保存一个程序。

每个程序完全载入内存，占用连续的地址。

问题：分区大小预先决定，难以决断；

随着程序的运行、退出，空闲区会增加；

对空闲区的管理将增加系统的负担。



a) CPU开始执行程序1　　b) CPU开始执行程序2

图7-7　分区调度

分页调度 内存预先被划分为不定长的若干分区，每个分区保存一个程序。

每个程序完全载入内存，占用连续的地址。

内存被分成大小相等的若干帧，程序被划分为大小相等的页。

加载程序时，页被载入到帧中，程序可以占用内存中不连续的帧。

问题：要求把程序整体载入内存中，大程序无法加载。

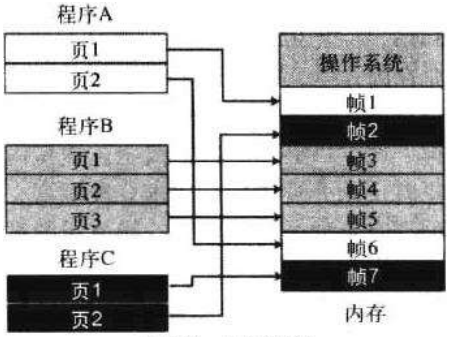


图7-8　分页调度

请求分页调度　程序被分成页，但是页可以依次载入内存、运行，然后被另一个页代替。

请求分页调度可加载执行大程序，甚至加载比内存还大的程序。

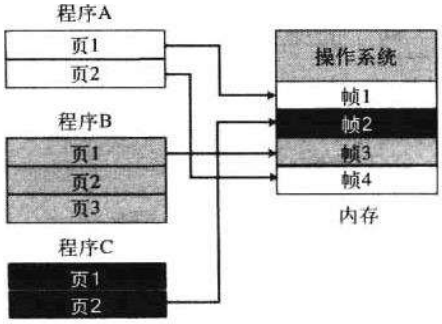


图7-9　请求分页调度

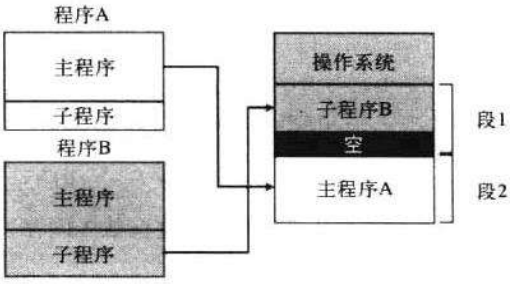


图7-10　请求分段调度

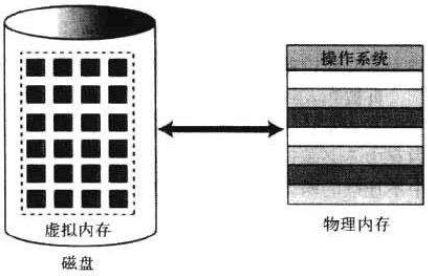
请求分段调度　程序按模块划分成段，它们载入内存中、执行，然后被其他段替代。

请求分页和分段调度　先将程序按模块分段，将每个段再细分为页，每个页再装入到不连续的帧中。

虚拟内存　在请求分页和请求分段调度中，一部分程序驻留内存，一部分则放在磁盘中。

由于采用交换技术，虽然只有部分程序加载到内存中，但用户感觉到整个程序“已经”加载到内存中运行，这个感觉上的更大的内存就是虚拟内存。

当今几乎所有的操作系统都使用了虚拟内存技术。



3 进程管理器

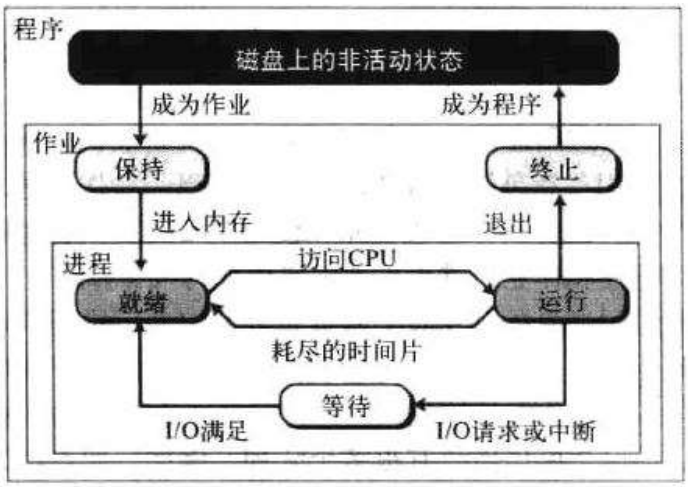
程序、作业和进程

程序　由程序员编写的一组稳定的指令，存储在磁盘上，可能会也可能不会成为作业。

作业　从被选中执行，到运行结束并再次成为程序的过程中的程序称为作业。

进程　执行中的程序，是驻留在内存中的作业，是从众多等待作业中选取出来并装入内存的作业。

状态图



调度器

作业调度器　将作业从保持状态转入就绪状态，或从运行状态转入终止状态。负责创建和终止进程。

进程调度器　调度进程在就绪、运行、等待状态之间转换。

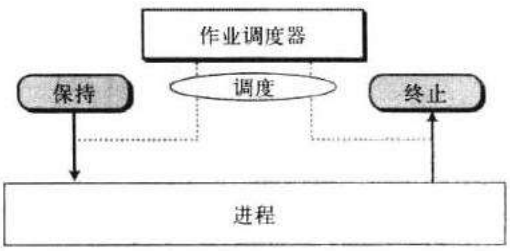


图7-13　作业调度器

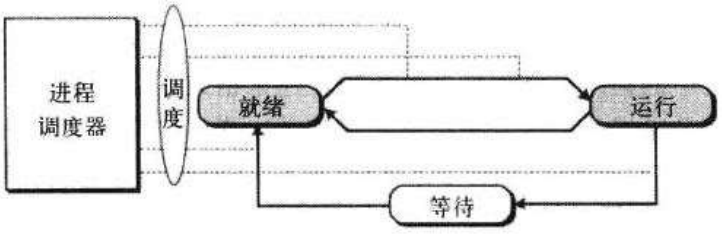
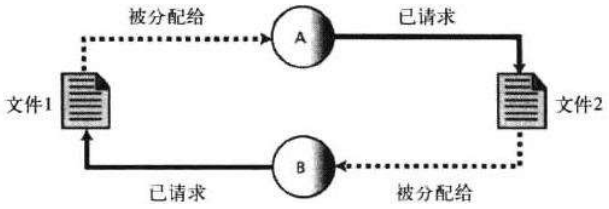


图7-14　进程调度器

队列 多个进程或作业相互竞争计算机资源。操作系统使用作业控制块和进程控制块描述作业和进程信息，将等待同一种资源的作业或进程的控制块存储到队列中。进程管理器可以用多种策略从队列中选择下一个作业或进程。

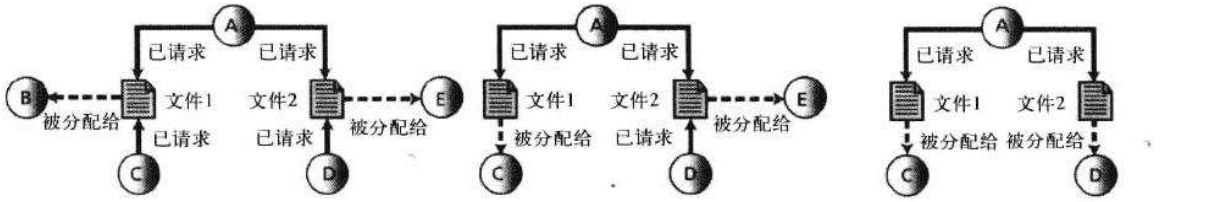
进程同步

死锁 当操作系统没有对进程的资源进行限制时将会发生死锁。



死锁的四个必要条件：互斥、资源占有、抢先、循环等待。

饥饿（饿死） 当操作系统对进程分配资源有太多限制的时候，进程长期得不到资源，称为饥饿。



a) 进程A需要文件1和文件2两者　b) 进程A还需要文件1和文件2两者　c) 进程A还需要文件1和文件2两者（饿死）

设备管理器 设备管理器负责有效使用输入/输出设备。

不停监视所有输入/输出设备的状态；为每一个设备维护一个队列；控制用于访问输入/输出设备的不同策略。

4 文件管理器

操作系统使用文件管理器控制对文件的访问。

控制对文件的访问；管理对文件的创建、删除和修改；为文件命名；管理文件的存储；负责归档和备份。

四、主流操作系统

UNIX、Linux、Windows。

复 习 题

Q7-1 应用程序和操作系统的不同点是什么？

答：操作系统是一种用来使得其他程序更加方便有效运行的程序。

Q7-2 操作系统的组成是什么？

答：用户界面、内存管理器、进程管理器、设备管理器和文件管理器。

Q7-3 单道程序和多道程序之间有何区别？

答：单道程序下，内存中只有一个程序。

多道程序下，多个程序同时存储装入内存，但是计算机的资源仅分配给正在运行的程序。

Q7-4 分页调度与分区调度有什么差别？

答：在分区调度中，内存被分为不定长的几个部分，每个部分保存一个完整的程序。

在分页调度中，程序和内存都被分为大小相等的若干部分。程序在内存中不必是连续的。

Q7-5 为什么请求分页调度比常规页面调度更有效率？

答：在分页调度中，程序必须整体载入内存中才能运行。而在请求分页调度中，一个程序只有部分页载入内存中，这意味着更多的程序可以在任何给定时间使用计算机的资源。

Q7-6 程序和作业之间有何联系？作业和进程之间有何联系？程序和进程之间的联系又如何？

答：程序是存储在磁盘上的一组稳定的指令，程序在被选中执行时成为作业。

作业是计划执行的程序，作业在真正装入内存、开始执行时成为进程。

Q7-7 程序驻留在哪里？作业驻留在哪里？进程驻留在哪里？

答：进程至少有一部分驻留在主存中，程序和作业驻留在磁盘上。

Q7-8 作业调度器和进程调度器有什么区别？

答：作业调度器将作业从保持状态转入就绪状态，或从运行状态转入终止状态。

进程调度器调度进程在不同状态之间的转换。

Q7-9 为什么操作系统需要队列？

答：因为可能同时有许多作业和进程。为了共享所有资源，需要队列来确保所有作业和进程都能访问所需资源。

Q7-10 死锁和饥饿有何区别？

答：当进程都在等待其他进程占有的资源时，就会发生死锁：它们都在等待对方。当操作系统没有对进程的资源进行限制时将会发生死锁。

当操作系统对进程分配资源有太多限制时，就会发生饥饿。如果进程永远等不到开始执行所需的资源，它可能永远也不会开始执行。

练 习 题

P7-1 一个计算机装有一个单道程序的操作系统。如果内存容量为 64MB，操作系统需要 4MB 内存，那么该计算机执行一个程序可用的最大内存为多少？

解：64 − 4 = 60 MB

P7-2 若操作系统自动分配 10MB 内存给数据，重做第 P7-1 题。

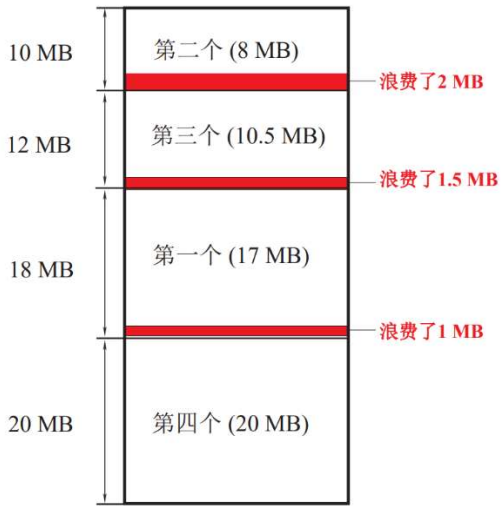
解：64 − (10 + 4) = 50 MB

P7-3 一个单道程序的操作系统执行程序时平均访问 CPU 要 10 微秒，访问 I/O 设备要 70 微秒，CPU 空闲时间为百分之多少？

解：$\frac{70}{70+10} \times 100\% = 87.5\%$

P7-4 一个多道程序的操作系统用一个适当的分配计划把 60MB 内存分为 10MB、12MB、18MB、20MB。第一个程序运行需要 17MB 内存，使用了第三分区。第二个程序运行需要 8MB 内存，使用了第一分区。第三个程序运行需要 10.5MB，使用了第二分区。最后，第四个程序运行需要 20MB 内存，使用了第四分区。那么总共使用了多少内存？总共浪费了多少内存？内存的浪费率是多少？

解：



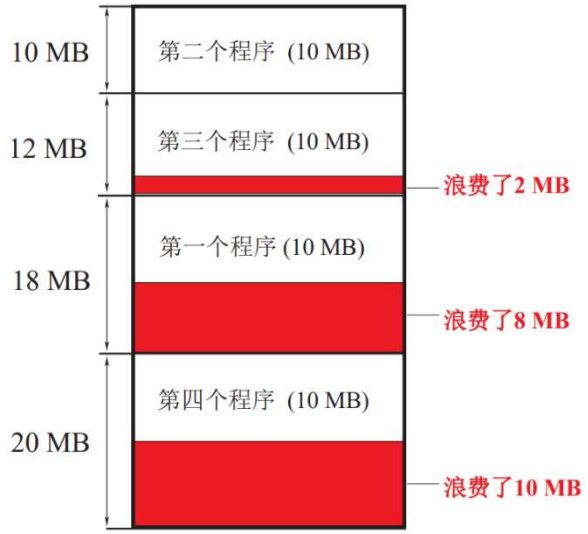
使用的总内存 = 17 + 8 + 10.5 + 20 = 55.5 MB

浪费的总内存 = 2 + 1.5 +1 = 4.5 MB

内存的浪费率 = $\frac{4.5}{60} \times 100\% = 7.5\%$

P7-5 如果所有的程序都需要 10MB 内存，重做第 P7-4 题。

解：



使用的总内存 $= 10 + 10 + 10 + 10 = 40$ MB

浪费的总内存 $= 2 + 8 + 10 = 20$ MB

内存的浪费率 $= \frac{20}{60} \times 100\% = 33.3\%$

P7-6 一个多道程序的操作系统使用分页调度。可用内存为 60MB，分为 15 个帧，每一帧大小为 4MB。第一个程序需要 13MB，第二个程序需要 12MB，第三个程序需要 27MB。

a.第一个程序需要用到多少帧？

b.第二个程序需要用到多少帧？

c.第三个程序需要用到多少帧？

d.有多少个帧没有用到？

e.总共浪费的内存是多少？

f.内存的浪费率是多少？

解：**a.** $13 \div 4 = 3.25 \rightarrow 4$ 页，故需要 4 帧

b. $12 \div 4 = 3$ 页，故需要 3 帧

c. $27 \div 4 = 6.75 \rightarrow 7$ 页，故需要 7 帧

d. $15 - (4 + 3 + 7) = 1$ 帧

e. 不考虑每帧内部浪费的内存，共有 1 帧→4 MB 浪费掉。

f. $\frac{4}{60} \times 100\% = 6.67\%$

P7-7 一个操作系统使用的虚拟内存，但执行的时候需要所有的程序驻留在物理内存中(没有分页调度或分段调度)。物理内存大小为 100MB，虚拟内存为 1GB。有多少 10MB 大小的程序可以同时运行？它们之中有多少可以随时驻留在内存中？多少则必须要存在磁盘里？

解：物理内存和虚拟内存共 $100 + 1024 = 1124$ MB，$1124 \div 10 = 112.4 \rightarrow$故可以同时运行 112 个程序。

其中 $100 \div 10 = 10$ 个可以随时驻留在内存中，$[1024 \div 10] = 102$ 个必须存在磁盘里。

P7-8 进程在下面的情况下处于什么状态？

a.进程在使用 CPU

b.进程结束打印，等待 CPU 再次调用

c.进程因为时间片用尽而被终止

d.进程从键盘读取数据

e.进程打印数据

答：**a.**运行

b.就绪

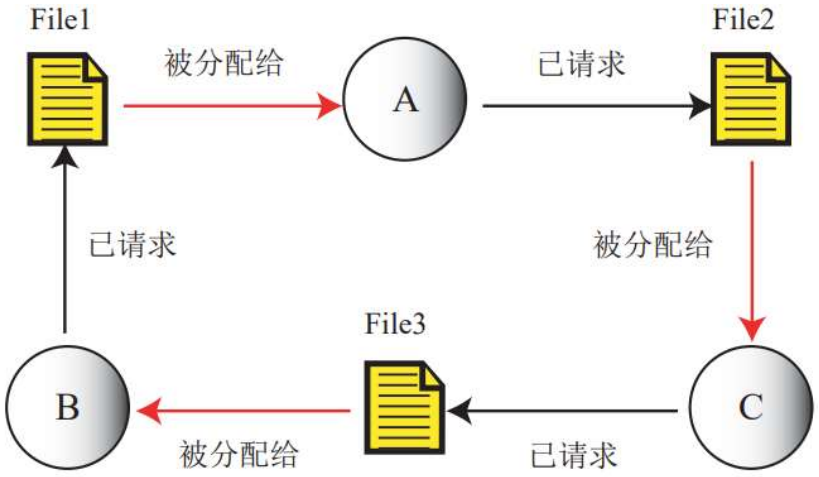
c.就绪

d.等待

e.等待

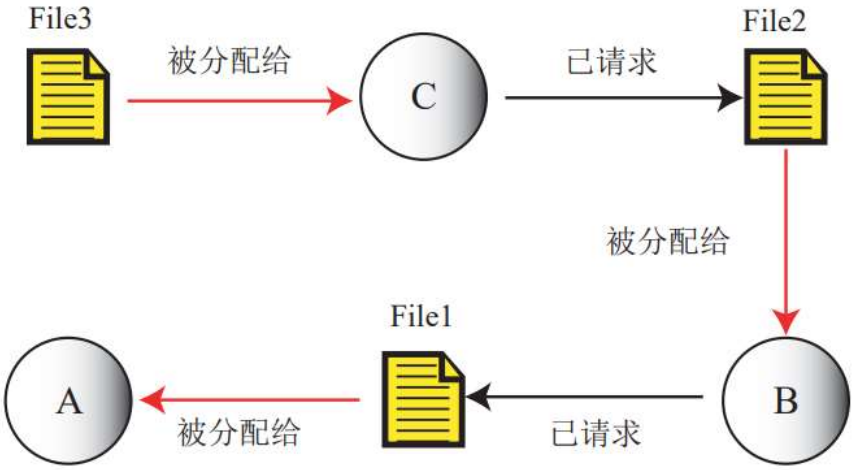
P7-9 三个进程(A、B 和 C)同时运行，进程 A 占用 File 1 但需要 File 2。进程 B 占用 File 3 但需要 File 1。进程 C 占用 File 2 但需要 File 3。为这几个进程画一个框图。这种情况是不是死锁？

解：是死锁，因为满足其四个必要条件：互斥、资源占有、抢先、循环等待。



P7-10 三个进程(A、B 和 C)同时运行，进程 A 占有 File 1，进程 B 占有 File 2 但需要 File 1，进程 C 占有 File 3 但需要 File 2。为这几个进程画一个框图。这种情况是不是死锁？如果不是，说明进程怎样最后完成它们的任务。

解：不是死锁，因为不满足其四个必要条件之一的循环等待。

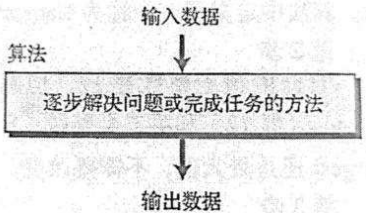


第八章 算 法

一、概 念

非正式定义

算法是一种逐步解决问题或者完成任务的方法。



二、三种结构

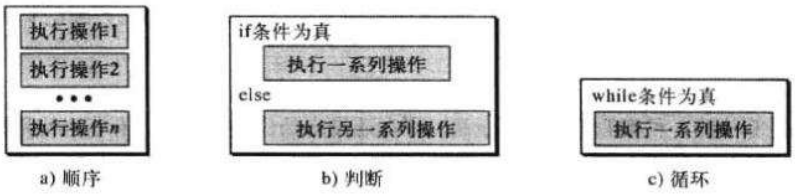


图8-6 三种结构

1 顺序

算法（最终编为程序）都是指令序列，可以是一个简单指令或是其他两种结构之一。

2 判断（选择）

需要检测条件，如果检测结果为真，执行一个指令序列；如果为假，执行另一个指令序列。

3 循环

在一些问题中，相同的指令序列必须重复执行。通过循环结构来解决重复的问题。

是一组明确步骤的有序集合，它产生结果并在有限的时间内终止。

三、算法的表示

1 UML

统一建模语言（UML）是算法的图形表示法。隐藏了算法的所有细节，只显示算法如何从开始运行到结尾。

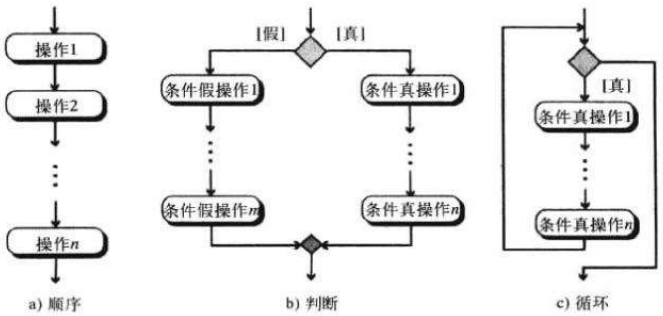


图8-7 三种结构的UML

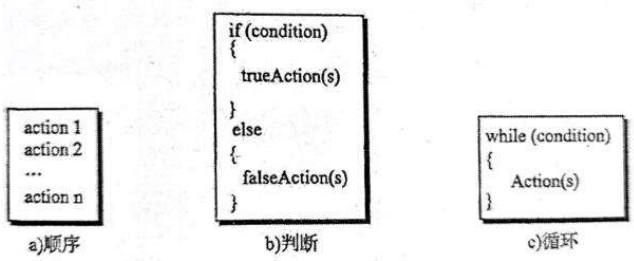


图 8-8 用伪代码表示三种结构

2 伪代码

伪代码是算法的一种类似英语的表示法。伪代码没有统一的标准。

四、更正式的定义

正式定义

算法是一组明确步骤的有序集合，它产生结果并在有限的时间内终止。

五、基本算法

1 求和

在循环中使用加法操作。

2 乘积

在循环中使用乘法操作。

3 最大和最小

找最小值的算法中，先用一个很大的数初始化，再循环执行判断两个数的较小值。

找最大值的算法同理。

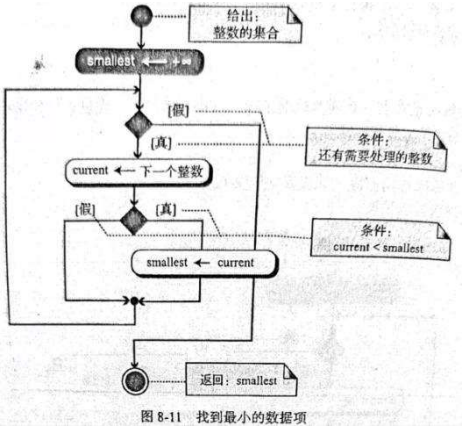
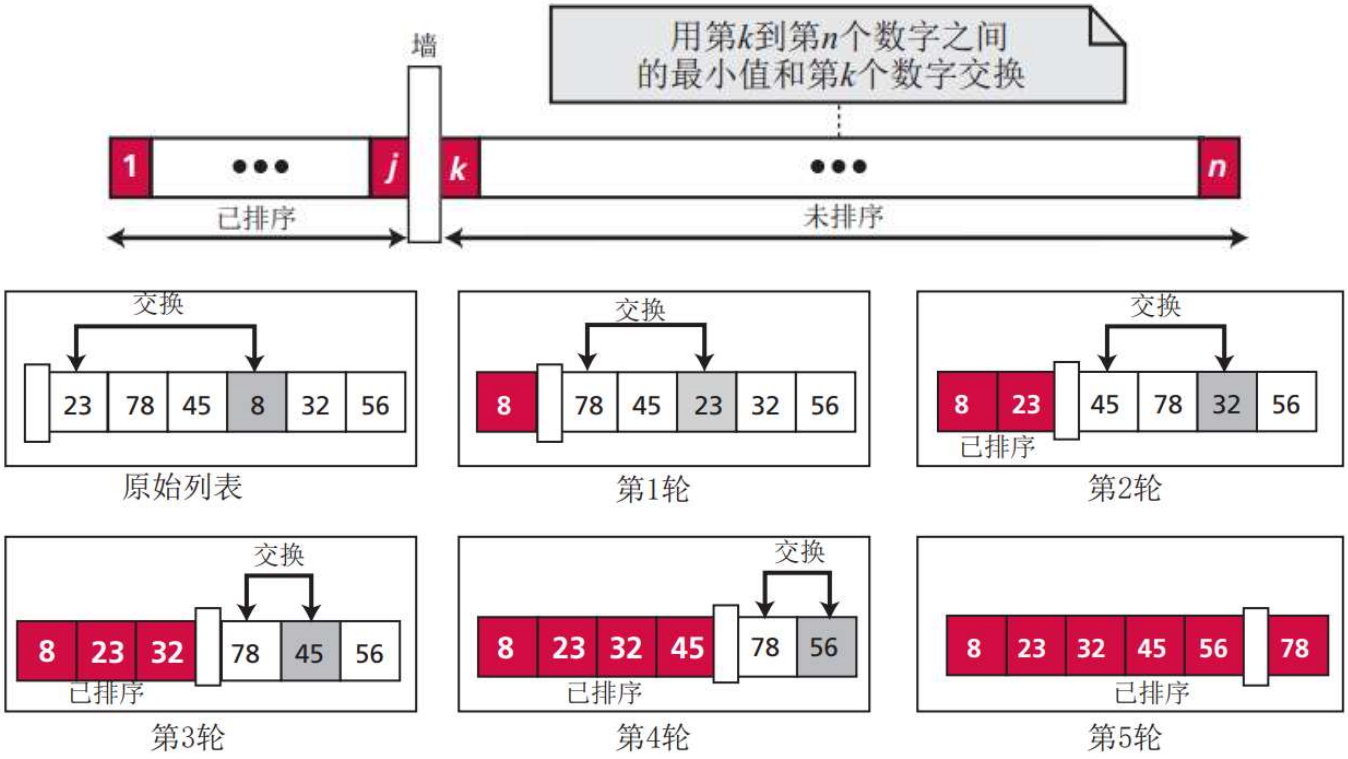


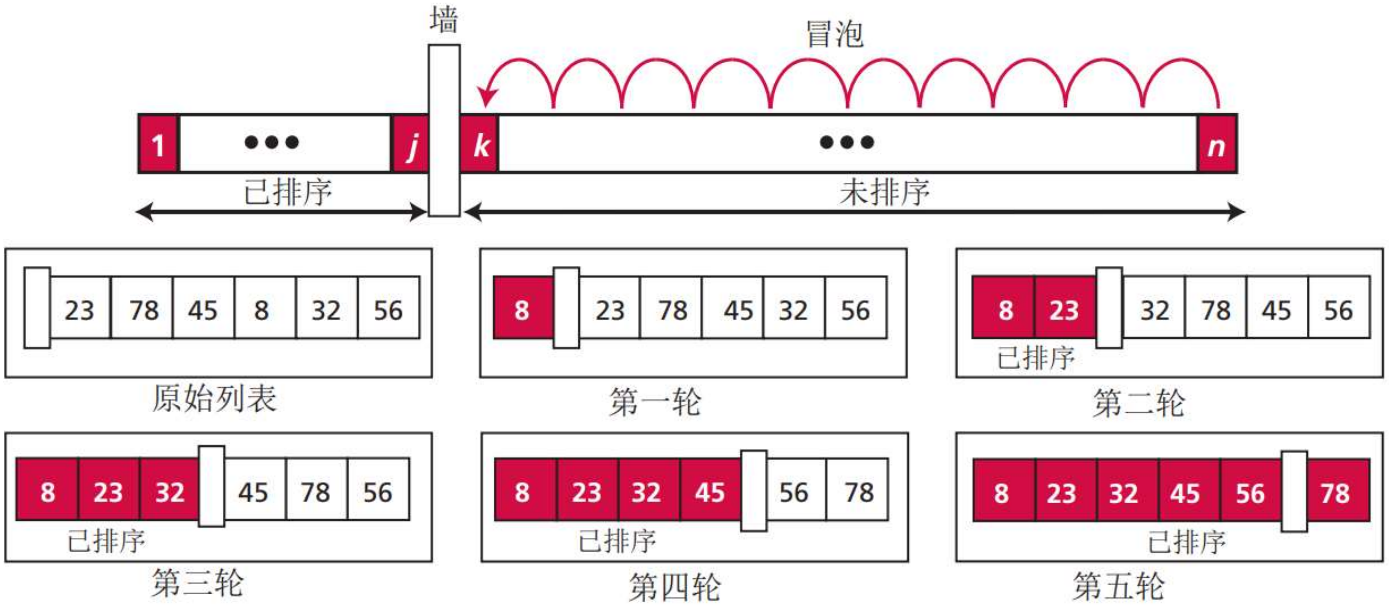
图 8-11 找到最小的数据项

4 排序

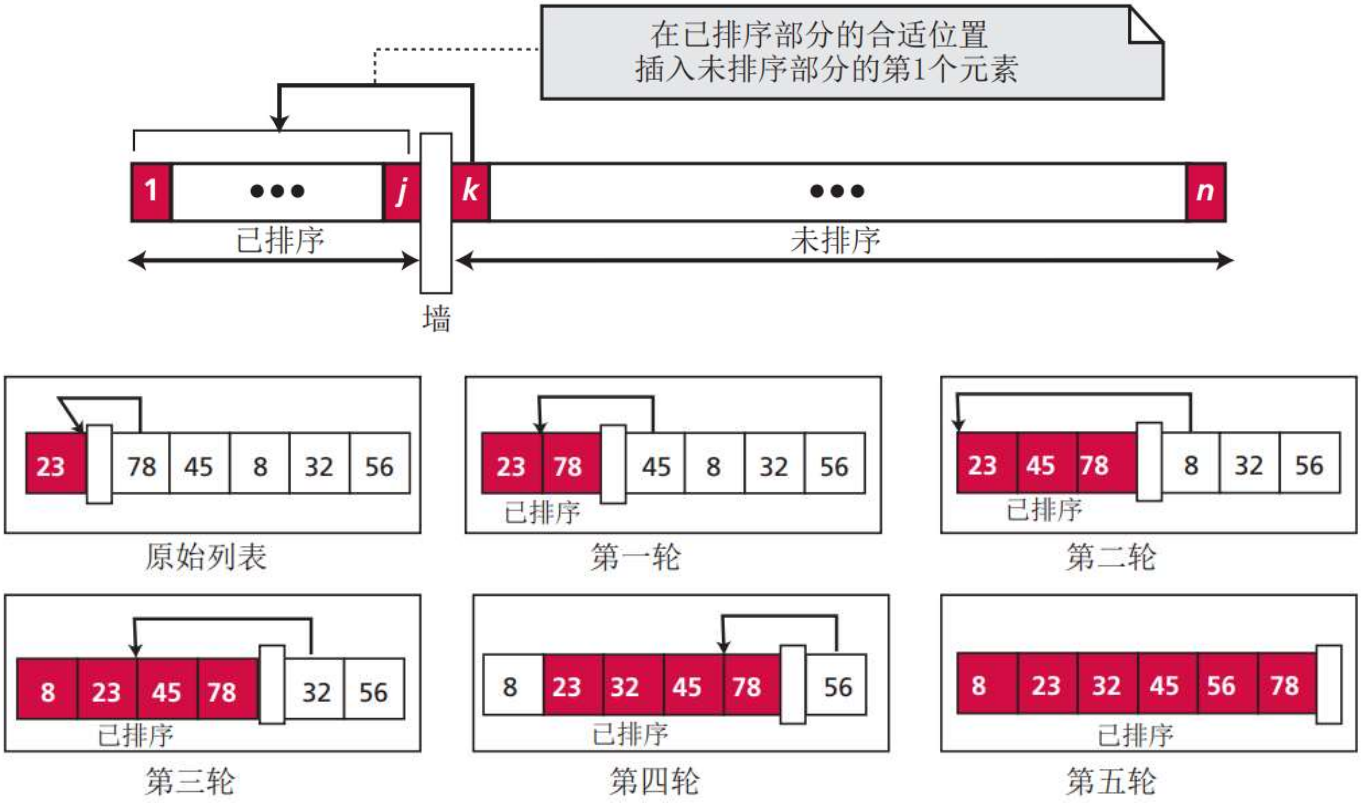
选择排序



冒泡排序



插入排序



其他排序算法

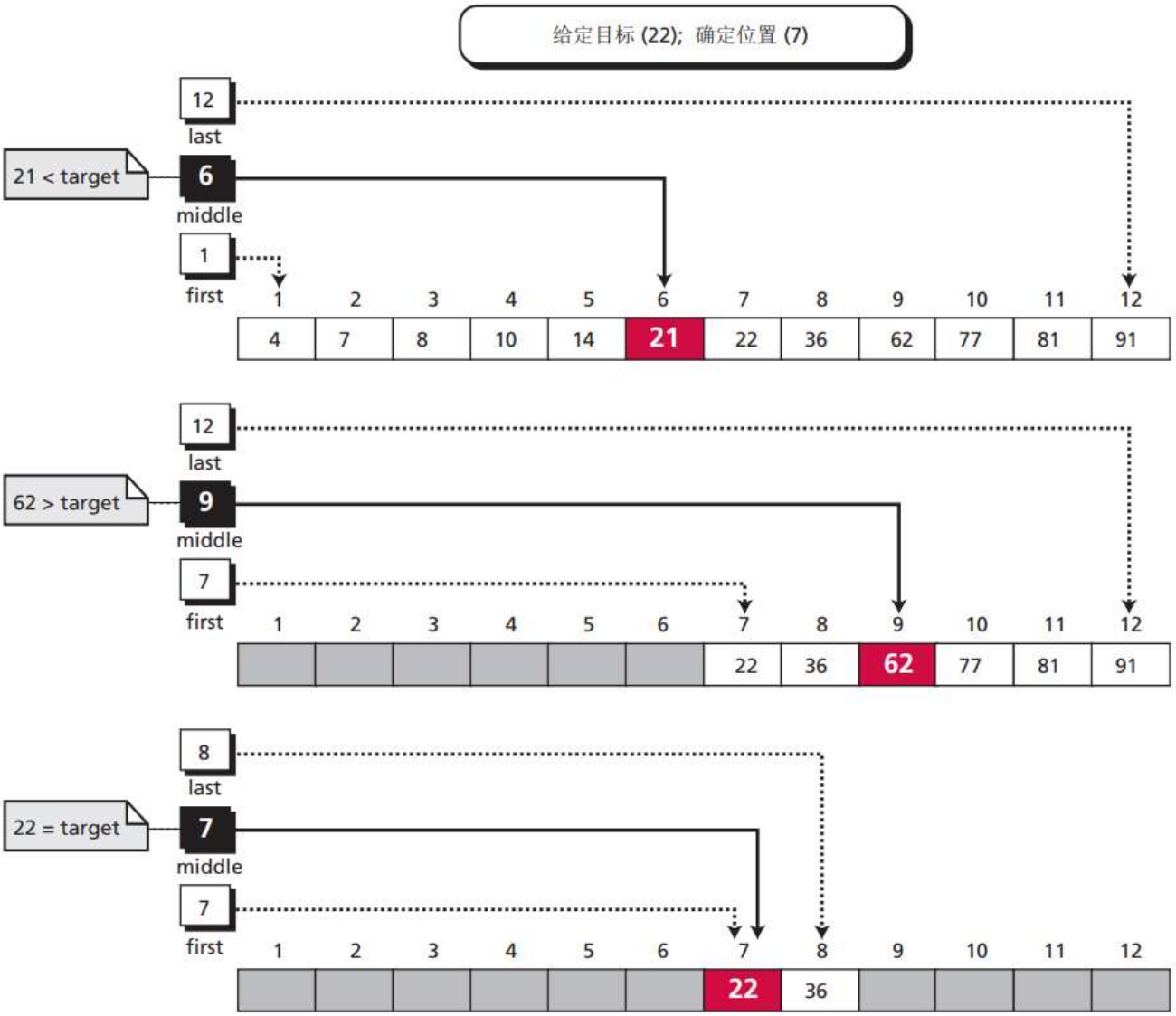
以上三种算法效率很低，它们是最简单的算法，是更高效算法的基础

还有其他算法，快速排序、堆排序、Shell 排序、桶排序、合并排序和基排序等。

5 **查找**

顺序查找 用于在无序列表中查找。

折半查找（二分查找） 对有序列表更有效率的查找方法。



六、子算法

结构化编程的原则要求将算法分成几个单元，称为子算法。

子算法的优点：程序更容易理解；子算法可在主算法的不同地方调用，而无需重写。

七、递　归

有两种途径可用于编写解决问题的算法。一种使用迭代，另一种使用递归。递归就是算法调用自身的过程。

1 迭代的定义

如果算法的定义不涉及算法本身，则算法是迭代的。

$$\text{Factorial}(n) = \begin{cases} 1 & \text{当 } n = 0\text{时} \\ n \times (n-1) \times (n-2) \cdots 3 \times 2 \times 1 & \text{当 } n > 0\text{时} \end{cases}$$

←阶乘的迭代定义

2 递归的定义

一个算法出现在它本身的定义中，该算法就是递归定义的。

$$\text{Factorial}(n) = \begin{cases} 1 & \text{当 } n = 0\text{时} \\ n \times \text{Factorial}(n-1) & \text{当 } n > 0\text{时} \end{cases}$$

←阶乘的递归定义

复 习 题

Q8-1 算法的正式定义是什么？

答：算法是一组明确步骤的有序集合，它产生结果并在有限的时间内终止。

Q8-2 给出用于结构化程序设计中的三种结构的定义。

答：**a.** 顺序：算法是指令序列，可以是一个简单指令或是其他两种结构之一。

b. 判断（选择）：检测条件，如果检测结果为真，执行一个指令序列；如果为假，执行另一个指令序列。

c. 循环：重复相同的指令序列。

Q8-3 UML 图与算法有何关系？

答：统一建模语言（UML）是算法的图形表示法。它使用“大图”的形式掩盖了算法的所有细节，只显示算法从开始到结束的整个流程。

Q8-4 伪代码与算法有何关系？

答：伪代码是算法的一种类似英语的表示法。

Q8-5 排序算法的用途是什么？

答：排序算法根据数据的值对其进行排列。

Q8-6 本章有哪三种基本排序算法？

答：选择排序、冒泡排序、插入排序。

Q8-7 查找算法的用途是什么？

答：在列表中确定目标所在位置。

Q8-8 本章讨论的基本查找算法主要有哪两种？

答：顺序查找和折半查找。前者常用于查找无序列表，后者用于查找有序列表。

Q8-9 给出迭代过程的定义和一个例子。

答：如果算法使用循环结构来执行重复命令，则该算法是迭代的。

Q8-10 给出递归过程的定义和一个例子。

答：如果一个算法调用它本身，即出现在它本身的定义中，则该算法是递归的。

练 习 题

P8-1 使用求和算法， 画一张表，显示下面的列表中每个整数被处理后的和值。

20 12 70 81 45 13 81

解：每次迭代后 **Sum** 的值如下表所示：

迭代	数据项	Sum= 0
1	20	**Sum= 0** + 20 = 20
2	12	**Sum= 20** + 12 = 32
3	70	**Sum= 32** + 70 = 102
4	81	**Sum= 102** + 81 = 183
5	45	**Sum= 183** + 45 = 228
6	13	**Sum= 228** + 13 = 241
7	81	**Sum= 241** + 81 = 322
退出循环后		**Sum= 322**

P8-2 使用乘积算法， 画一张表，显示下面的列表中每个整数被处理后的乘积值。

2 12 8 11 10 5 20

解：每次迭代后 **Product** 的值如下表所示：

迭代	数据项	Product= 1
1 个	2	**Product = 1** × 2 = 2
2	12	**Product= 2** × 12 = 24
3	8	**Product= 24** × 8 = 192
4	11	**Product= 192** × 11 = 2112
5	10	**Product= 2112** × 10 = 21120
6	5	**Product= 21120** × 5 = 105600
7	20	**Product = 105600** × 20 = 2112000
退出循环后		**Product= 2112000**

P8-3 使用 FindLargest 算法， 画一张表，显示下面的列表中每个整数被处理后的 Largest 的值。

18 12 8 20 10 32 5

解：每次迭代后 **Largest** 的值如下表所示：

迭代	数据项	Largest=－∞
1	18	**Largest= 18**
2	12	**Largest= 18**
3	8	**Largest= 18**
4	20	**Largest= 20**
5	10	**Largest= 10**
6	32	**Largest= 32**
7	5	**Largest= 32**
退出循环后		**Largest= 32**

P8-4 使用 FindSmallest 算法， 画一张表，显示下面的列表中每个整数被处理后的 Smallest 的值。

18 3 11 8 20 1 2

解：每次迭代后 **Smallest** 的值如下表所示：

迭代	数据项	Smallest=＋∞
1	18	**Smallest= 18**
2	3	**Smallest= 3**
3	11	**Smallest= 3**
4	8	**Smallest= 3**
5	20	**Smallest= 3**
6	1 个	**Smallest= 1**
7	2	**Smallest= 1**
退出循环后		**Smallest= 1**

P8-5 使用选择排序算法， 手工排序下列数据列表并借助表给出每轮所做的工作。

14 7 23 31 40 56 78 9 2

解：使用选择排序算法，每轮过后列表的状态和墙的位置如下表所示：

轮	列表								
	14	7	23	31	40	56	78	9	2
1	***2***	7	23	31	40	56	78	9	14
2	***2***	***7***	23	31	40	56	78	9	14
3	***2***	***7***	***9***	31	40	56	78	23	14
4	***2***	***7***	***9***	***14***	40	56	78	23	31
5	***2***	***7***	***9***	***14***	***23***	56	78	40	31
6	***2***	***7***	***9***	***14***	***23***	***31***	78	40	56
7	***2***	***7***	***9***	***14***	***23***	***78***	***40***	78	56
8	***2***	***7***	***9***	***14***	***23***	***78***	***40***	***56***	78

P8-6 使用冒泡排序算法， 手工排序下列数据列表并借助表给出每轮所做的工作。

14 7 23 31 40 56 78 9 2

解：使用冒泡排序算法，每轮过后列表的状态和墙的位置如下表所示：

轮	列表								
	14	7	23	31	40	56	78	9	2
1	***2***	14	7	23	31	40	56	78	9
2	***2***	***7***	14	9	23	31	40	56	78
3	***2***	***7***	***9***	14	23	31	40	56	78
4	***2***	***7***	***9***	***14***	23	31	40	56	78
5	***2***	***7***	***9***	***14***	***23***	31	40	56	78
6	***2***	***7***	***9***	***14***	***23***	***31***	40	56	78
7	***2***	***7***	***9***	***14***	***23***	***31***	***40***	56	78
8	***2***	***7***	***9***	***14***	***23***	***31***	***40***	***56***	78

P8-7 使用插入排序算法，手工排序下列数据列表并借助表给出每轮所做的工作。

7 23 31 40 56 78 9 2

解：使用插入排序算法，每轮过后列表的状态和墙的位置如下表所示：

轮	列表								
	14	7	23	31	40	56	78	9	2
1	***7***	***14***	23	31	40	56	78	9	2
2	***7***	***14***	***23***	31	40	56	78	9	2
3	***7***	***14***	***23***	***31***	40	56	78	9	2
4	***7***	***14***	***23***	***31***	***40***	56	78	9	2
5	***7***	***14***	***23***	***31***	***40***	***56***	78	9	2
6	***7***	***14***	***23***	***31***	***40***	***56***	***78***	9	2
7	***7***	***9***	***14***	***23***	***31***	***40***	***56***	***78***	2
8	***2***	***7***	***9***	***14***	***23***	***31***	***40***	***56***	***78***

P8-8 一个列表包含以下元素。前两个元素已经使用选择排序算法排好序了，那么在进行了选择排序的三轮后列表中的元素排序结果如何？

7 8 26 44 13 23 98 57

解：经过三轮选择排序后，列表的状态和墙的位置如下表所示：

轮	列表							
	7	**8**	26	44	13	23	98	57
1	**7**	**8**	**13**	44	26	23	98	57
2	**7**	**8**	**13**	**23**	26	44	98	57
3	**7**	**8**	**13**	**23**	**26**	44	98	57

P8-9 一个列表包含以下元素。前两个元素已经使用冒泡排序算法排好序了，那么在进行了冒泡排序的三轮后列表中的元素排序结果如何？

7 8 26 44 13 23 57 98

解：经过三轮冒泡排序后，列表的状态和墙的位置如下表所示：

轮	列表							
	7	**8**	26	44	13	23	57	98
1	**7**	**8**	**13**	26	44	23	57	98
2	**7**	**8**	**13**	**23**	26	44	57	98
3	**7**	**8**	**13**	**23**	**26**	44	57	98

P8-10 一个列表包含以下元素。前两个元素已经使用插入排序算法排好序了，那么在进行插入排序的三轮后列表中的元素排序结果如何？

3 13 7 26 44 23 98 57

解：经过三轮插入排序后，列表的状态和墙的位置如下表所示：

迭代	列表							
	3	**13**	7	26	44	23	98	57
1	**3**	**7**	**13**	26	44	23	98	57
2	**3**	**7**	**13**	**26**	44	23	98	57
3	**3**	**7**	**13**	**26**	**44**	23	98	57

P8-11 一个列表包含以下元素。使用折半查找算法，跟踪查找 88 的步骤，要求给出每一步中 first、mid 和 last 的值。

8 13 17 26 44 56 88 97

解：目标（88）的位置为 7，步骤如下表所示：

first	*last*	*mid*
1	8	4
5	8	6
7	8	7

1	2	3	4	5	6	7	8	
8	13	17	26	44	56	88	97	target > 44
				44	56	88	97	target > 56
						88	97	target = 88

P8-12 一个列表包含以下元素。使用折半查找算法，跟踪查找 20 的步骤，要求给出每一步中 first、mid 和 last 的值。

17 26 44 56 88 97

解：目标（20）未找到，步骤如下表所示：

first	*last*	*mid*
1	6	3
1	2	1
2	2	2
2	1	1

1	2	3	4	5	6	
17	26	44	56	88	97	target < 44
17	26					target > 17
	26					target < 26

first > *last* → 目标不在列表中

P8-13 使用 8.5.1 节中的图 8-19（顺序查找），显示查找目标 11（不在列表中）的所有步骤。

解：目标（11）未找到，步骤如下表所示：

迭代	1	2	3	4	5	6	7	8	9	10	11	12	
	4	21	36	14	62	91	8	22	7	81	77	10	
1	**4**	21	36	14	62	91	8	22	7	81	77	10	target ≠ 4
2	4	**21**	36	14	62	91	8	22	7	81	77	10	target ≠ 21
3	4	21	**36**	14	62	91	8	22	7	81	77	10	target ≠ 36
4	4	21	36	**14**	62	91	8	22	7	81	77	10	target ≠ 14
5	4	21	36	14	**62**	91	8	22	7	81	77	10	target ≠ 62
6	4	21	36	14	62	**91**	8	22	7	81	77	10	target ≠ 91
7	4	21	36	14	62	91	**8**	22	7	81	77	10	target ≠ 8
8	4	21	36	14	62	91	8	**22**	7	81	77	10	target ≠ 22
9	4	21	36	14	62	91	8	22	**7**	81	77	10	target ≠ 7
10	4	21	36	14	62	91	8	22	7	**81**	77	10	target ≠ 81
11	4	21	36	14	62	91	8	22	7	81	**77**	10	target ≠ 77
12	4	21	36	14	62	91	8	22	7	81	77	**10**	target ≠ 10
	目标不在列表中。												

P8-14 使用 8.5.5 节中的图 8-20（折半查找），显示查找目标 17（不在列表中）的所有步骤。

解：目标（17）未找到，步骤如下表所示：

first	last	mid	1	2	3	4	5	6	7	8	9	10	11	12	
1	12	6	4	7	8	10	14	21	22	36	62	77	81	91	target < 21
1	5	3	4	7	8	10	14								target > 8
4	5	4				10	14								target > 10
5	5	5					14								target > 14
6	5	5	*first* > *last* → 目标不在列表中												

P8-15 应用阶乘算法的迭代定义，当求 6!（6 的阶乘）的值时，显示每一步中 F 的值。

解：6!＝720 用迭代算法计算的步骤如下表所示：

i	*Factorial*
1	F = **1**
2	F = **1** × 2 = 2
3	F = **2** × 3 = 6
4	F = **6** × 4 = 24
5	F = **24** × 5 = 120
6	F = **120** × 6 = 720
退出循环后	F = **720**

P8-16 应用阶乘算法的递归定义，当求 6!的值时，显示每一步中 F 的值。

解：6!＝720 用递归算法计算的步骤如下表所示：

↓	6! = 6 × 5!	6! = 6 × 5! = 6 × **120** = **720**	
↓	5! = 5 × 4!	5! = 5 × 4! = 5 × **24** = **120**	↑
↓	4! = 4 × 3!	4! = 4 × 3! = 4 × **6** = **24**	↑
↓	3! = 3 × 2!	3! = 3 × 2! = 3 × **2** = **6**	↑
↓	2! = 2 × 1!	2! = 2 × 1! = 2 × **1** = **2**	↑
↓	1! = 1 × 0!	1! = 1 × 0! = 1 × **1** = **1**	↑
	0! = **1**		↑

P8-17 用伪代码写出一个递归算法，使用右图中的定义，求两整数的最大公约数（gcd）。在这个定义中，表达式“$x \bmod y$”意思是 x 除以 y，取余数作为操作的值。

$$\gcd(x, y) = \begin{cases} x & \text{当 } y = 0 \text{时} \\ \gcd(y, x \bmod y) & \text{其他情况} \end{cases}$$

解：计算最大公约数的伪代码如下：

```
算法：gcd(x, y)
目的：求两个数的最大公约数
前提：x, y
后续：无
返回：gcd(x, y)
{
    if (y = 0)
        return x
    else
        return gcd (y, x mod y)
}
```

P8-18 使用上题图中的定义，求下列值：

a. gcd(7,41)　**b.** gcd(12,100)　**c.** gcd(80,4)　**d.** gcd(17,29)

解：

a. 下表展示了**gcd** (7, 41) = 7的计算过程：

x	y	$x \bmod y$	结果
7	41	7	gcd (7, 41) = gcd (41, 7)
41	7	0	gcd (41, 7) = gcd (7, 0)
7	0		gcd (7, 0) = 7

b. 下表展示了 **gcd** (12, 100) = 4的计算过程：

x	y	$x \bmod y$	结果
12	100	12	gcd (12, 100) = gcd (100, 12)
100	12	4	gcd (100, 12) = gcd (12, 4)
12	4	0	gcd (12, 4) = gcd (4, 0)
4	0		**gcd** (4, 0) = 4

c. 下表展示了**gcd** (80, 4)= 4的计算过程：

x	y	$x \bmod y$	结果
80	4	0	gcd (80, 4) = gcd (4, 0)
4	0		**gcd** (4, 0) = 4

d. 下表展示了**gcd** (17, 29) = 1的计算过程：

x	y	$x \bmod y$	结果
17	29	17	**gcd** (17, 29) = **gcd** (29, 17)
29	17	12	**gcd** (29, 17) = **gcd** (17, 12)
17	12	5	**gcd** (17, 12) = **gcd** (12, 5)
12	5	2	**gcd** (12, 5) = **gcd** (5, 2)
5	2	1	**gcd** (5, 2) = **gcd** (2, 1)
2	1	0	**gcd** (2, 1) = **gcd** (1, 0)
1	0		**gcd** (1, 0) = 1

P8-19 用伪代码写一递归算法，使用下图中的定义，求一次从 n 个对象中取 k 个对象的组合。

$$C(n, k) = \begin{cases} 1 & \text{当} k = 0 \text{或 } n = k \\ C(n-1, k) + C(n-1, k-1) & \text{当 } n > k > 0 \end{cases}$$

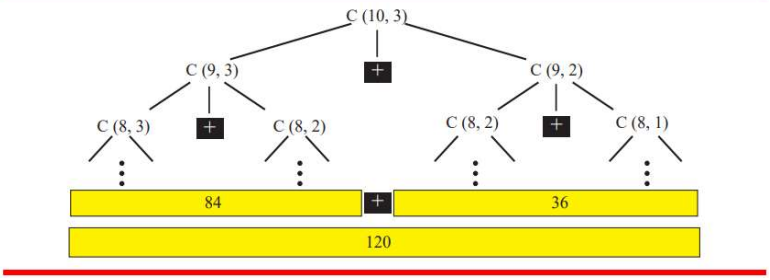
解：计算组合数的伪代码如下：

```
算法：Combination(x, y)
目的：求一次从 n 个对象中取 k 个对象的组合数C(n,k)
前提：n, k
后续：无
返回：C(n, k)
{
    if (k = 0 or n = k)
        return 1
    else
        return C(n - 1, k) + C(n - 1, k - 1)
}
```

P8-20 使用上题图中的定义，求下列值：

a. C(3,2)　**b.** C(5,5)　**c.** C(2,7)　**d.** C(4, 3)

解：**a.** **图 P8-20** C(10, 3)的计算过程



b. C(5, 5) = 1

c. C(2, 7) 不存在

d. C(4, 3) = C(3, 3) + C(3, 2) = 1 + [C(2, 2) + C(2, 1)]= 1 + [1 + C(1, 1) + C(1, 0)] = 1 + [1 + 1 + 1] = 4

P8-21 斐波那契序列（Fib(n)）被用在科学和数学上，如下图所示。用伪代码写一递归算法，求 Fib(n)的值。

$$\text{Fib}(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ \text{Fib}(n) = \text{Fib}(n-1) + \text{Fib}(n-2) & \text{if } n > 1 \end{cases}$$

解：计算斐波那契数列的伪代码如下：

```
算法：Fibonacci(n)
目的：求斐波那契数列中的元素
前提：n
后续：无
返回：Fib(n)
{
    if (n = 0)
        return 0
    if (n = 1)
        return 1
    else
        return Fib(n - 1) + Fib(n - 2)
}
```

P8-22 使用上题图中的定义，求下列值：

a. Fib(2) **b.** Fib(3) **c.** Fib(4) **d.** Fib(5)

解：**a.** Fib(2) = Fib(1) + Fib(0) = 1 + 0 = 1

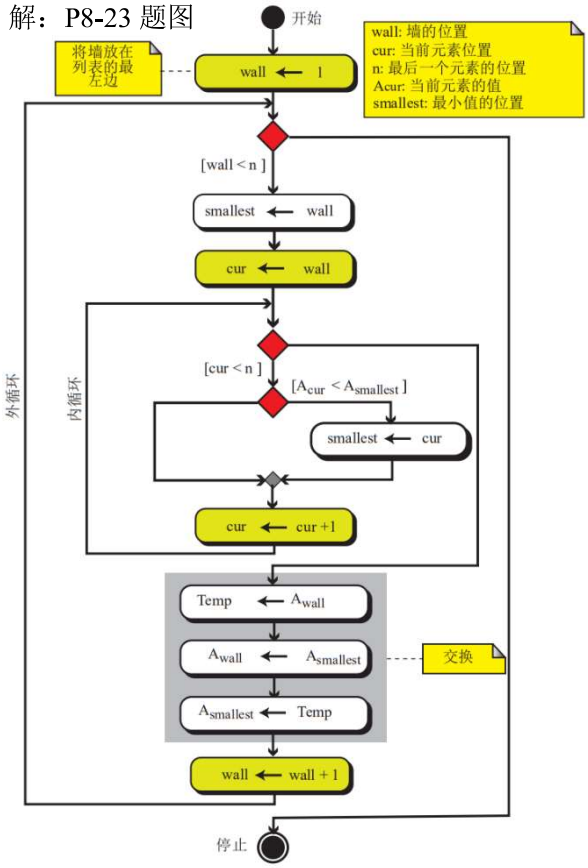
b. Fib(3) = Fib(2) + Fib(1) = [Fib(1) + Fib(0)] + Fib(1) = [1 + 0] + 1 = 2

c. Fib(4) = Fib(3) + Fib(2) = 2 + 1 = 3

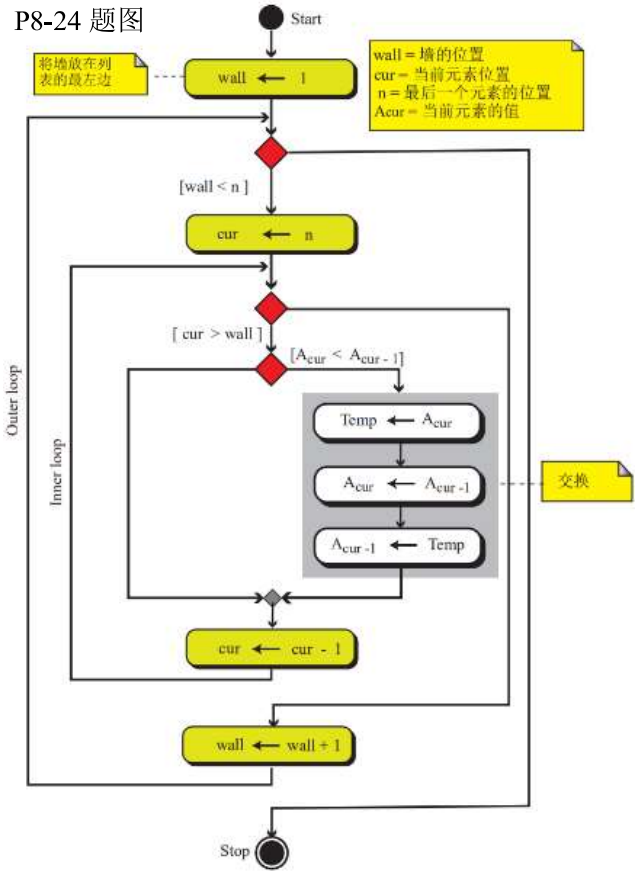
d. Fib(5) = Fib(4) + Fib(3) = 3 + 2 = 5

P8-23 画出使用两个循环的选择排序算法的 UML 图。嵌套循环用来在未排序的子列表中找出最小的元素。

解：P8-23 题图



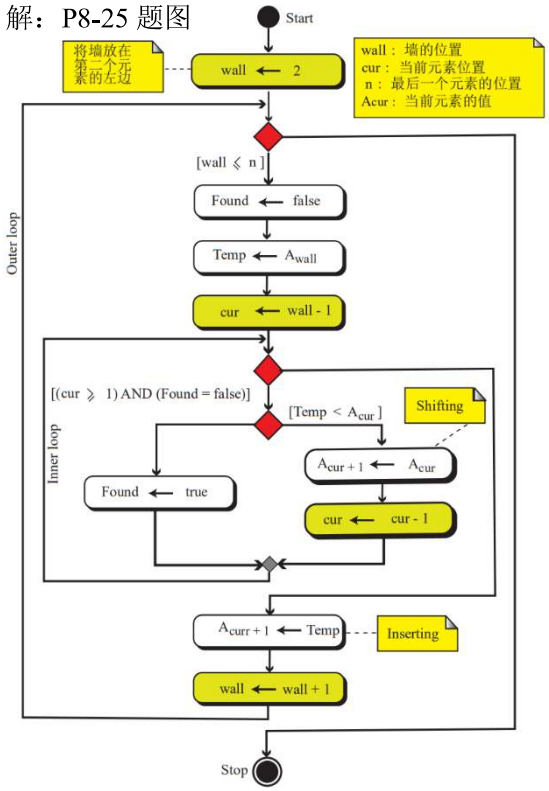
P8-24 题图



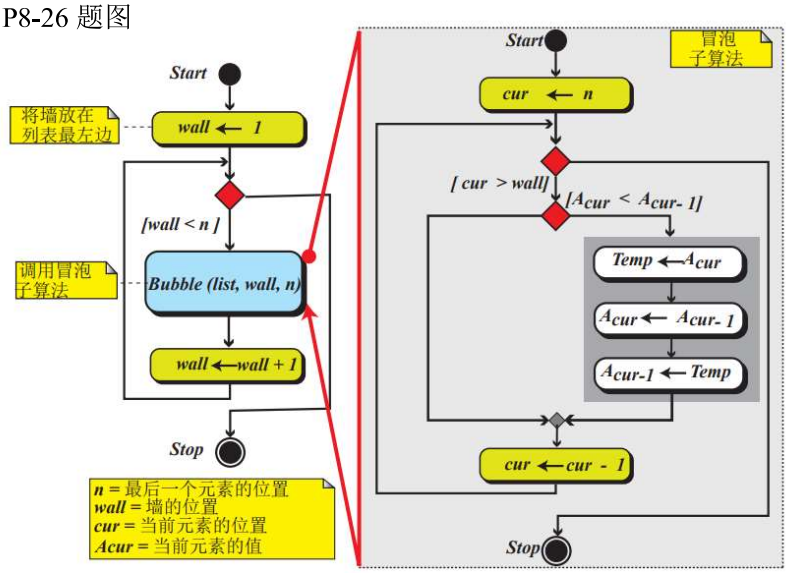
P8-24 画出使用两个循环的冒泡排序算法的 UML 图。嵌套循环用来在未排序的子列表中交换相邻的数据项。

P8-25 画出使用两个循环的插入排序算法的 UML 图。嵌套循环用来在排序的子列表中做插入工作。

解：P8-25 题图



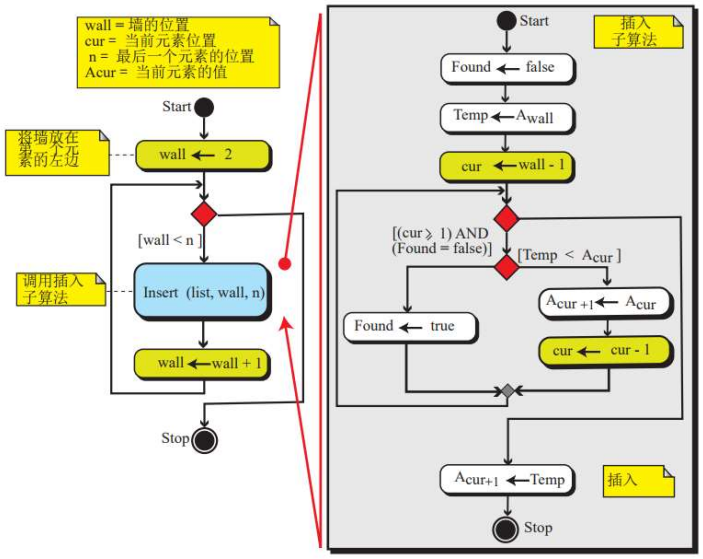
P8-26 题图



P8-26 画出使用子算法的冒泡排序算法的 UML 图。子算法对未排序的子列表进行冒泡排序。

P8-27 画出使用子算法的插入排序算法的 UML 图。子算法对排序的子表做插入工作。

解：



P8-28 用伪代码写出 8.5.1 节图 8-9 中的 UML 图的算法。

解：求和算法的伪代码如下：

```
算法：Summation (list)
目的：求一组整数的和
前提：一组整数
后续：无
返回：该组整数的和
{
  sum ← 0
  while (more integer to add)
  {
    get next integer
    sum ← sum + (next integer)
```

```
  }
  return sum
}
```

P8-29 用伪代码写出 8.5.5 节图 8-10 中的 UMI.图的算法。

解：乘积算法的伪代码如下：

```
算法：Product (list)
目的：求一组整数的积
前提：一组整数
后续：无
返回：该组整数的积
{
  product ← 1
  while (more integer to multiply)
  {
    get next integer
    product ← product × (next integer)
  }
  return product
}
```

P8-30 用伪代码写出使用两个嵌套循环的选择排序算法。

解：选择排序算法的伪代码如下：

```
算法：SelectionSort (list, n)
目的：用选择排序给一组数排序
前提：一组数
后续：无
返回：
{
  wall ← 1                              // 将墙放在列表的最左边
  while (wall < n)                      // 外层循环
  {
    smallest ← wall
    cur ← wall                          // 当前元素是墙左边的元素
    while (cur < n)                     // 内层循环
    {
      if (A_cur < A_smallest)
        smallest ← cur
      cur ← cur + 1                     // 移动当前元素
    }
    Temp ← A_wall                       // 下面三行执行交换
    A_wall ← A_smallest
    A_smallest ← Temp
    wall ← wall + 1                     // 将墙向右移动一个元素
  }
}
```

P8-31 用伪代码写出使用子算法的选择排序算法，子算法是在未排序的子列表中求最小的整数。

解：选择排序主算法的伪代码如下：

```
算法：SelectionSort (list, n)
目的：用选择排序给一组数排序
```

```
前提：一组数
后续：无
返回：
{
  wall ← 1                                    // 将墙放在列表的最左边
  while (wall < n)
  {
    smallest ← FindSmallest(list, wall, n) //调用 FindSmallest
    Temp ← A_wall                             // 下面三行执行交换
    A_wall ← A_smallest
    A_smallest ← Temp
    wall ← wall + 1                           // 将墙向右移动一个元素
  }
  return SortedList
}
```

其中最小值子算法的伪代码如下：

```
算法：FindSmallest (list, wall, n)
目的：找出无序列表中的最小值
前提：一组数
后续：无
返回：无序列表中最小值的位置 Algorithm: (list, wall, n)
{
    smallest ← wall                    // 假定第一个元素最小
    cur ← wall                         // 当前元素是墙左边的元素
    while (cur < n)                    // 内层循环
    {
      if (A_cur < A_smallest)
        smallest ← cur
      cur ← cur + 1                    // 移动当前元素
    }
}
```

P8-32 用伪代码写出使用两个嵌套循环的冒泡排序算法。

解：冒泡排序算法的伪代码如下：

```
算法：BubbleSort (list, n)
目的：用冒泡排序给一组数排序
前提：一组数
后续：无
返回：
{
  wall ← 1                             // 将墙放在列表的最左边
  while (wall < n)                     // 外层循环
  {
    cur ← n                            // 从列表的最后开始
    while (cur > wall)                 // 从列将最小的元素冒到左边的列表
    {
       if (A_cur < A_cur - 1)          // 向左冒一个泡
       {
         Temp ← A_cur
         A_cur ← Ac_ur-1
         A_cur-1 ← Temp
       }
```

```
      cur ← cur - 1
    }
    wall ← wall + 1                    // 将墙向右移动一个元素
  }
}
```

P8-33 用伪代码写出使用子算法的冒泡排序算法，子算法是在未排序的子列表中做冒泡工作。

解：冒泡排序主算法的伪代码如下：

```
算法：BubbleSort (list, n)
目的：用冒泡排序给一组数排序
前提：一组数
后续：无
返回：
{
  wall ← 1                             // 将墙放在列表的最左边
  while (wall < n)
  {
    Bubble(list, wall, n)
    wall ← wall + 1                    // 将墙向右移动一个元素
  }
return SortedList
}
```

其中冒泡子算法的伪代码如下：

```
算法：Bubble (list, wall, n)
目的：从无序列表中冒泡
前提：一组数、数的个数和墙的位置
后续：无
返回：
{
    cur ← n                            // 从列表的最后开始
    while (cur > wall)                 // 从列将最小的元素冒到左边的列表
    {
        if (Acur < Acur - 1)           // 向左冒一个泡
        {
          Temp ← Acur
          Acur ← Acur-1
          Acur-1 ← Temp
        }
        cur ← cur - 1
      }
}
```

P8-34 用伪代码写出使用两个嵌套循环的插入排序算法。

解：插入排序算法的伪代码如下：

```
算法：InsertSort (list, n)
目的：用插入排序给一组数排序
前提：一组数
后续：无
返回：
{
  wall ← 2                             // 将墙放在第二个元素的左边
  while (wall ≤ N)
```

```
  {
    Found ← false
    Temp ← A_wall
    cur ←wall - 1                    // 将墙左边的元素设置为当前元素
    while ((cur ≥ 1) AND Found = false))
    {
      if (Temp < A_cur)              // 向左交换一个位置
      {
        A_cur + 1 ← A_cur
        cur ← cur - 1
      }
      else Found ← true
    }
    A_cur +1 ← Temp                  // 插入
    wall ← wall + 1                  // 将墙向右移动一个元素
  }
}
```

P8-35 用伪代码写出使用子算法的插入排序算法，子算法是在未排序的子列表中做插入工作。

解：插入排序主算法的伪代码如下：

```
算法：InsertSort (list, n)
目的：用插入排序给一组数排序
前提：一组数
后续：无
返回：
{
  wall ← 2
  while (wall ≤ N)
  {
    Insert(list, wall, n)          // 调用 Insert
    wall ← wall + 1
  }
}
```

其中插入子算法的伪代码如下：

```
算法：InsertSort (list, n)
目的：在已排序列表中插入元素
前提：一组数和墙的位置
后续：无
返回：
{
    Found ← false
    Temp ← A_wall
    cur ←wall - 1                    // 将墙左边的元素设置为当前元素
    while ((cur ≥ 1) AND Found = false))
    {
      if (Temp < A_cur)              // 向左交换一个位置
      {
        A_cur + 1 ← A_cur
        cur ← cur - 1
      }
      else Found ← true
```

```
    }
    A_{cur+1} ← Temp                    // 插入
}
```

P8-36 用伪代码写出顺序查找算法，包含如果目标找到或找不到时算法的终止条件。

解：顺序查找算法的伪代码如下：

```
算法: SequentialSearch (list, target, n)
目的: 对无序列表应用顺序查找
前提: list, target, n
后续: 无
返回: flag, i                                   // flag 表示查找状态
{
  flag ← false
  i ← 1
  while ((i < n + 1) or (flag = false))
  {
    if (A_i = target)
      flag ← true                               // A_i 是列表中第 i 个数
    i ← i + 1
  }
  return (flag, i)                              // 若 flag 为真，i 为要找的元素的位置。
}
```

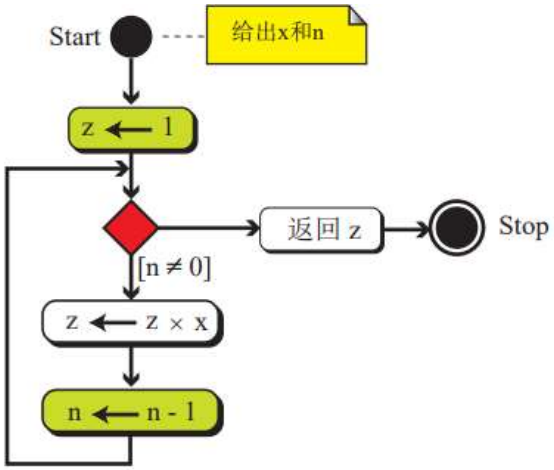
P8-37 用伪代码写出折半查找算法，包含如果目标找到或找不到时算法的终止条件。

解：折半查找算法的伪代码如下：

```
算法: BinarySearch (list, target, n)
目的: 对有序列表应用折半查找
前提: list, target, n
后续: 无
返回: flag, i                                   // flag 表示查找状态
{
  flag ← false
  first ← 1
  last ← n
  while (first ≤ last)
  {
    mid = (first + last) / 2
    if (target < A_mid)
      Last ← mid - 1                            // A_i 是列表中第 i 个数
    if (target > A_mid)
      first ← mid + 1
    if (target = A_mid)                         // 找到 target
      first ← Last + 1
  }
  if (target > A_mid)
    i = mid + 1
  if (x ≤ A_mid)
    i = mid
  if (x = A_mid)
    flag ← true
  return (flag, i)
}
```

P8-38 使用乘积算法的 UML 图，画图计算 x^n 的值，x 和 n 是两个给定的整数。

解：



P8-39 用伪代码写一算法，求 x^n 的值，x 和 n 是两个给定的整数。

解：求整数的幂算法的伪代码如下：

```
算法：Power (x, n)
目的：求 x^n 的值，其中 x 和 n 为整数
前提：x 和 n
后续：无
返回：x^n
{
  z ← 1
  while (n ≠ 1)
  {
    z ← z × x
    n ← n − 1
  }
  return z
}
```

第十章 软件工程

一、软件生命周期

软件开发后，要经过用户使用，如果发现错误、设计规则或需求方本身发生变化，需要修改。使用和修改这两个步骤一直进行下去直到软件过时。“过时”意味着因效率低下、语言过时、用户需求的重大变化或其它因素而导致软件失去它的有效性。

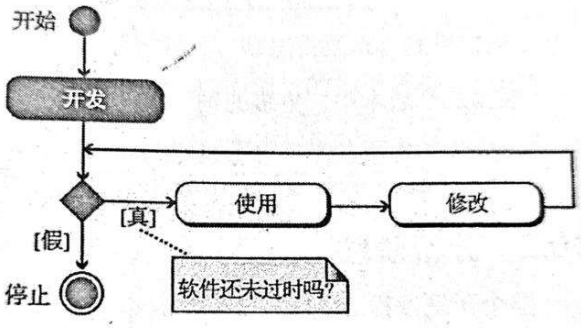


图 10-1 软件生命周期

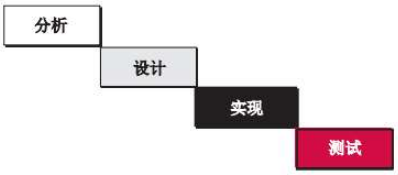
开发过程模型

在软件生命周期中，开发过程包括 4 个阶段：分析、设计、实现和测试。

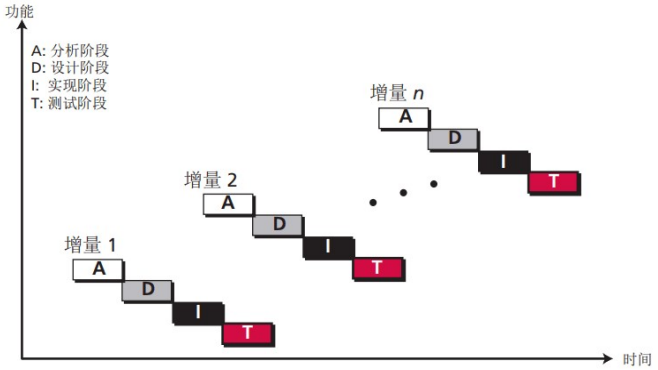
瀑布模型 在这种模型中，开发过程只有一个方向的流动。这意味着前一个阶段不结束，后一个阶段不能开始。

优点：下一阶段开始前，之前的阶段都已完成。

缺点：难以定位问题，如果过程的一部分有问题，必须检查整个过程。



增量模型 在增量模型中，软件的开发要经历一系列步骤。开发者首先完成系统的一个简化版本；在第二个版本中，更多的细节被加入，这样一直继续下去。



二、分析阶段

整个开发过程的开始，这个阶段生成规格说明文档，说明软件要做什么，而没有说明如何去做。

1 面向过程分析

如果实现阶段使用过程式语言，那么面向过程分析（也称为结构化分析或经典分析）就是分析阶段使用的方法。

数据流图 显示系统中数据的流动。

实体关系图 我们不讨论

状态图 通常用于当系统中的实体状态在响应事件时将会改变的情况下。

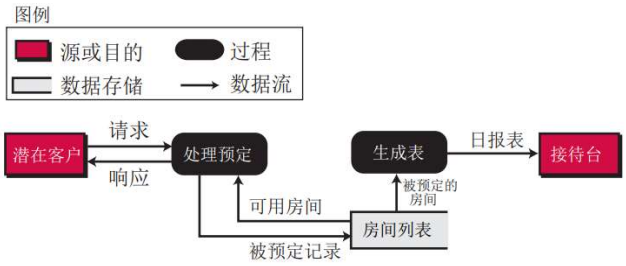


图 10-4 数据流图的一个例子

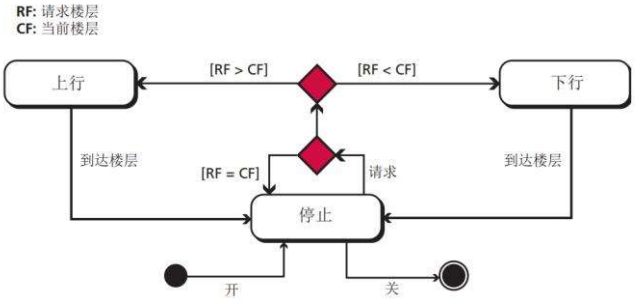


图 10-5 状态图的一个例子

2 面向对象分析

如果实现使用面向对象语言，那么面向对象分析就是分析过程使用的。

用例图 给出了系统的用户视图，显示了用户与系统间的交互。

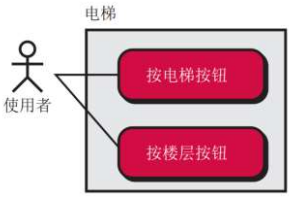


图 10-6 用例图的一个例子

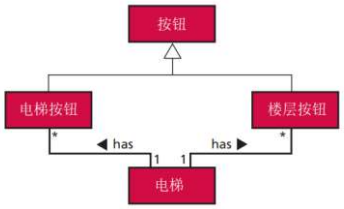


图 10-7 类图的一个例子

类图 需要考虑系统涉及的实体。

状态图 与面向过程分析中状态图作用相同。

三、设计阶段

设计阶段定义系统如何完成在分析阶段所定义的需求。在设计阶段，系统所有的组成部分都被定义。

1 面向过程设计

在**面向过程**设计中，整个系统被分解成一组过程或模块。我们既要设计过程，也要设计数据。

结构图 在面向过程设计中，说明模块间关系的常用工具是结构图。

模块化 将大项目分解成较小的部分，以便能够容易理解和处理。

耦合：对两个模块互相绑定紧密程度的度量。

软件系统中模块间的**耦合必须最小化。**

内聚：程序中处理过程相关紧密程度的度量。

软件系统模块间的**内聚必须最大化。**

2 面向对象设计

在**面向对象**设计中，设计阶段通过详细描述类的细节来继续。

类是由一组变量（属性）和一组方法组成。面向对象设计阶段列出这些**属性**和方法的细节。

四、实现阶段

程序员为面向过程设计中的模块编写程序或者编写程序单元，实现面向对象设计中的类。

1 语言的选择

在面向过程开发中，工程团队需要从 Fortran、Cobol、Pascal、C 等面向过程语言中选择一个或一组语言。

2 软件质量

在实现阶段创建的软件质量是一个非常重要的问题。

软件质量因素 **软件质量**能够划分成三个广义的度量：**可操作性**、**可维护性**和**可迁移性**。

五、测试阶段

测试阶段的目标就是发现错误，这就意味着良好的测试策略能发现最多的错误。

1 白盒测试

白盒测试（或玻璃盒测试）基于知道软件内部结构的测试，目标是检查软件所有的部分是否全部设计出来。白

盒测试假定测试者知道有关软件的一切。

四个标准：

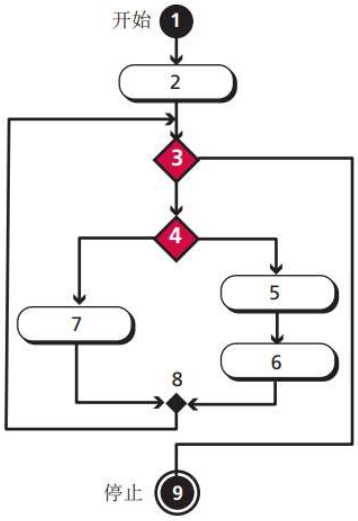
每个模块中的所有独立的路径至少被测试过一次。

所有的判断结构（两路的或多路的）每个分支都被测试。

每个循环被测试。

所有数据结构都被测试。

基本路径测试　一种软件中每条语句至少被执行一次的方法。例：



独立路径

路径1: (1, 2, 3, 9)
路径2: (1, 2, 3, 4, 5, 6, 8, 3, 9)
路径3: (1, 2, 3, 4, 7, 8, 3, 9)

控制结构测试　包含基本路径测试，包括条件测试、数据流测试、循环测试。

2 **黑盒测试**

在不知道程序的内部也不知道程序是怎样工作的情况下测试程序。

穷尽测试　用输入域中的所有可能的值去测试软件。常常不现实。

随机测试　选择输入域的值的子集来测试。

边界值测试　当遇到边界值时，错误经常发生，用边界值来测试非常重要。

六、文档

文档是一个持续的过程。

软件的正确使用和有效维护离不开文档。通常软件有三种独立的文档：**用户文档**、**系统文档**和**技术文档**。

1 **用户文档**

为了软件包正常运行，传统上称为用户手册的文档对用户是必不可少的。

2 **系统文档**

系统文档定义软件本身。在系统开发的 4 个阶段都应该存在。

3 **技术文档**

技术文档描述了软件系统的安装和服务。

复 习 题

Q10-1 定义“软件生命周期”。

答：软件和其他的产品一样，周期性地重复着一些阶段。

Q10-2 区分瀑布模型和增量开发模型。

答：在瀑布模型中，开发过程只有一个方向的流动。这意味着前一个阶段不结束，后一个阶段不能开始。

在增量模型中，软件的开发要经历一系列步骤。开发者首先完成系统的一个简化版本，这个版本表示了整个系统但不包括具体的细节。

Q10-3 软件开发的 4 个阶段是什么?

答：分析、设计、实现、测试。

Q10-4 说明分析阶段的目标，描述此阶段中的两种趋势。

答：分析阶段生成规格说明文档，这个文档说明了软件要做什么，而没有说明如何去做。

两种趋势是面向过程分析、面向对象分析。

Q10-5 说明设计阶段的目标，描述此阶段中的两种趋势。

答：设计阶段定义系统如何完成在分析阶段所定义的需求。在设计阶段，系统所有的组成部分都被定义。

两种趋势是面向过程设计、面向对象设计。

Q10-6 描述模块化，说出与模块化有关的两个问题。

答：模块化意味着将大项目分解成较小的部分，以便能够容易理解和处理。换言之，模块化意味着将大程序分解成能互相通信的小程序。

当系统被分解成模块时的两个重要问题是耦合和内聚。

Q10-7 描述耦合和内聚之间的区别。

答：耦合是对两个模块互相绑定紧密程度的度量。内聚是程序中处理过程相关紧密程度的度量。

Q10-8 说明实现阶段的目标，描述此阶段中的质量问题。

答：在实现阶段，程序员为面向过程设计中的模块编写程序，或者编写程序单元实现面向对象设计中的类。

在实现阶段创建的软件质量是一个非常重要的问题。

Q10-9 说明测试阶段的目标，列出两类测试。

答：测试阶段的目标就是发现错误。两类测试是白盒（或玻璃盒）测试和黑盒测试。

Q10-10 描述白盒测试和黑盒测试之间的区别。

答：白盒测试是检查系统的所有功能，由程序员来完成。

黑盒测试是在不知道程序内部的情况下测试程序，由系统测试工程师和用户完成。

练 习 题

P10-1 在第 9 章中，我们解释了常量的使用比字面值更受欢迎。这种偏好对软件生命周期的影响是什么？

P10-2 在第 9 章中，我们说明了模块间的通信可以通过传值或传引用来进行。哪一种提供了模块间更少的耦合？

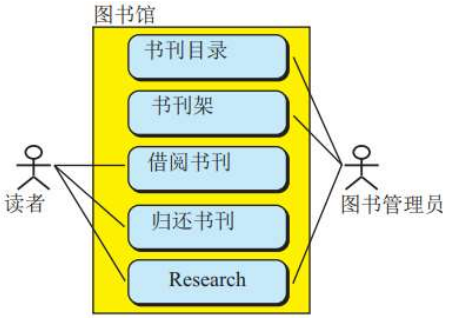
P10.3 在第 9 章中，我们说明了模块间的通信可以通过传值或传引用来进行。哪一种提供了模块间更多的内聚？

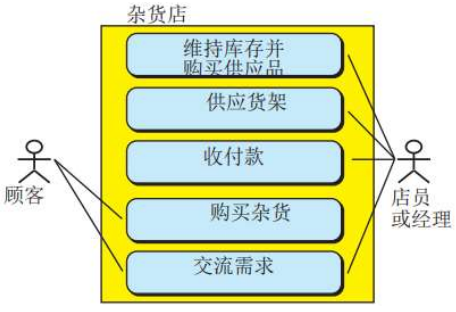
答：第 9 章没学。

P10-4 画出一个简单图书馆的用例图。

P10-5 画出一个小杂货店的用例图。

解：

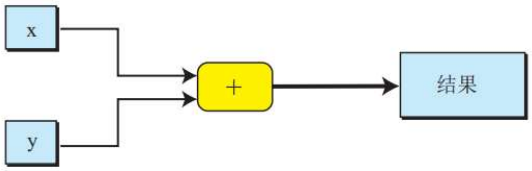


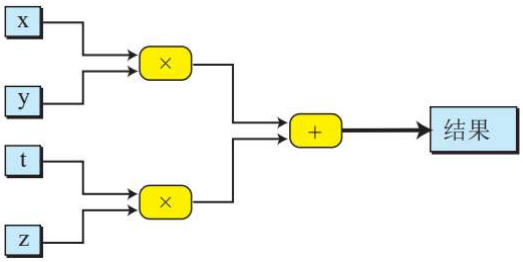


P10-6 显示简单数学公式 $x+y$ 的数据流图。

P10-7 显示简单数学公式 $x \times y + z \times t$ 的数据流图。

解：

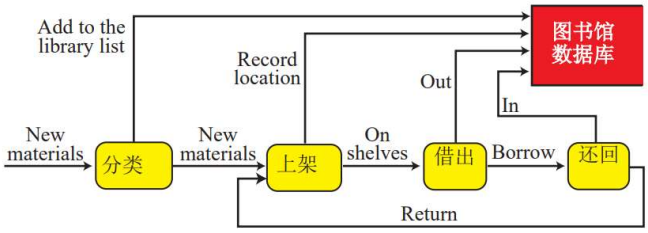


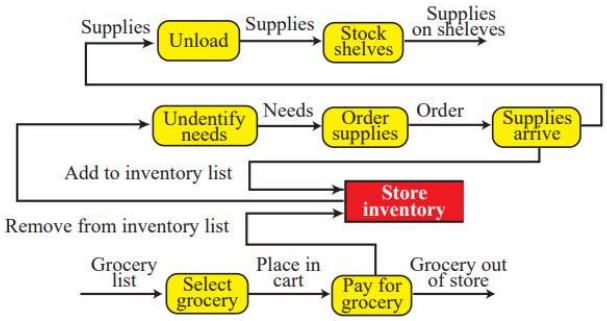


P10-8 显示图书馆的数据流图。

P10-9 显示小杂货店的数据流图。

解：

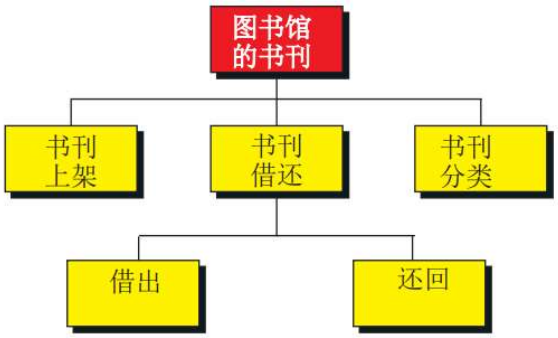


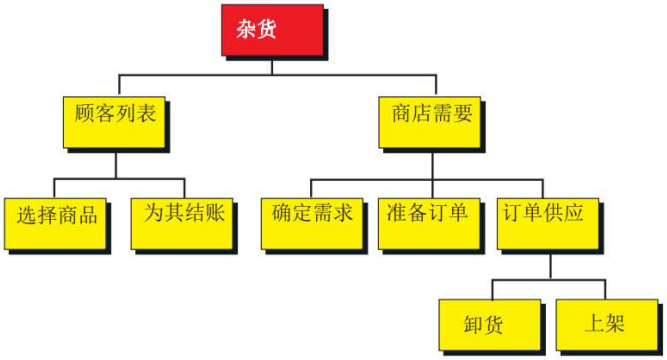


P10-10 创建练习题 P10-8 的结构图。

P10-11 创建练习题 P10-9 的结构图。

解：





P10-12 显示固定容量的堆栈(参见第 12 章)的状态图。

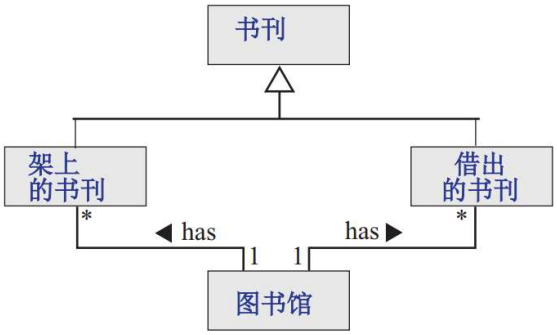
P10-13 显示固定容量的队列(参见第 12 章)的状态图。

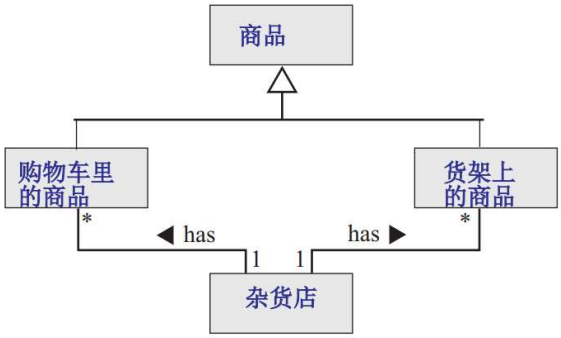
解：第 12 章不考。

P10-14 创建图书馆的类图。

P10-15 创建小杂货店的类图。

解：

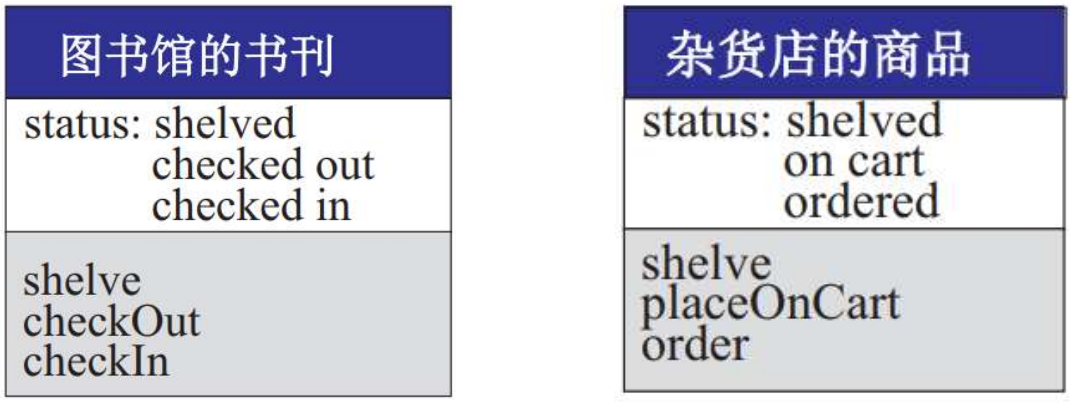




P10-16 显示练习题 P10-14 中类的细节。

P10-17 显示练习题 P10-15 中类的细节。

解：



P10-18 一个程序的输入由 1000 到 1999 范围（包含）中的三个整数构成。求出测试这些数字的所有组合的穷尽测试的数目。

解：$1000^3 = 10^9$

P10-19 列出练习题 P10-18 中所需要的边界值测试。

解：下表是推荐的边界值：

第一个整数	第二个整数	第三个整数
1000	1000	1000
1000	1000	1999
1000	1999	1000
1000	1999	1999
1999	1000	1000
1999	1000	1999
1999	1999	1000
1999	1999	1999

P10-20 一个随机数生成器能生成 0 到 0.999 间的一个数。该随机数生成器是如何用来做练习题 P10-18 中所描述的系统的随机测试的？

解：每次测试按照以下步骤：

a. 首先生成一个 0 到 0.999 间的一个数；

b. 然后将其乘 1000，得到 0 到 999 之间的一个数；

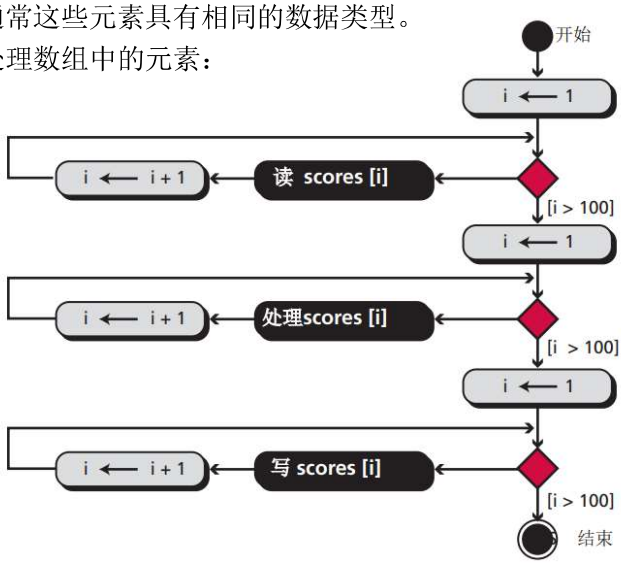
c. 最后将其加 1000 ，得到的数就是在 1000 到 1999 之间的整数。

第十一章 数据结构

一、数 组

数组是元素的顺序集合，通常这些元素具有相同的数据类型。

我们可以用循环来读写、处理数组中的元素：



1 数组名与元素名

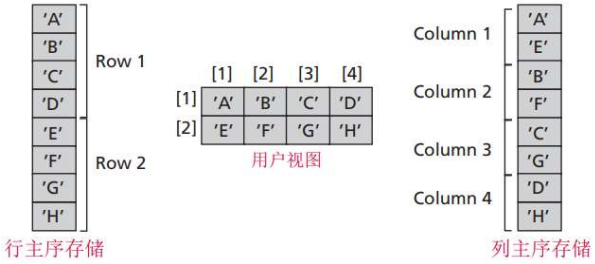
scores 是数组名，scores[1]、scores[2]是元素名。

2 多维数组

包含行和列的表格称为二维数组。

3 存储配置

大多数计算机使用行主序存储，也可以使用列主序存储。



4 数组操作

查找元素 我们可以对未排序数组使用顺序查找，已排序数组使用折半查找。

元素的插入 找到插入的位置后，插入新的元素，后面的元素向数组尾部移动一个元素。

元素的删除 删除元素后，后面的元素向数组的开始位置移动一个位置。

检索元素 随意地存取一个元素。

数组的遍历 指被应用于数组中每个元素上的操作。

5 字符串 是字符的集合。

6 数组的应用

当需要进行的插入和删除数目较少，而需要大量的查找和检索操作时，数组是合适的结构。

二、记 录

记录是一组相关元素的集合，它们可能是不同的类型，但一个记录有一个名称。记录中的每个元素称为域。

记录中的元素可以是相同类型或不同类型，但记录中的所有元素必须是关联的。

1 记录名与域名

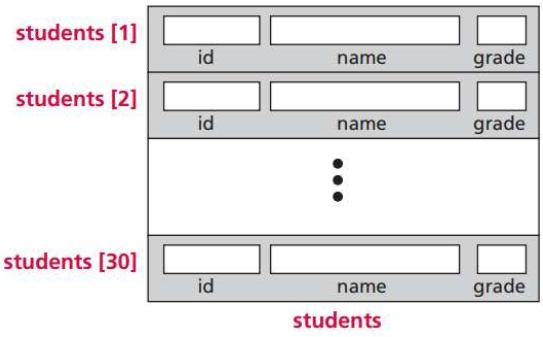
student 是记录名，student.id、student.name 是域名。

2 记录与数组的比较

数组可以定义一个班级的学生，而记录定义了学生的不同属性。

3 记录数组

记录数组中，首先定义元素（数组的名字），然后才能定义元素的部分。例：



第三个学生的学号：(student[3]).id。

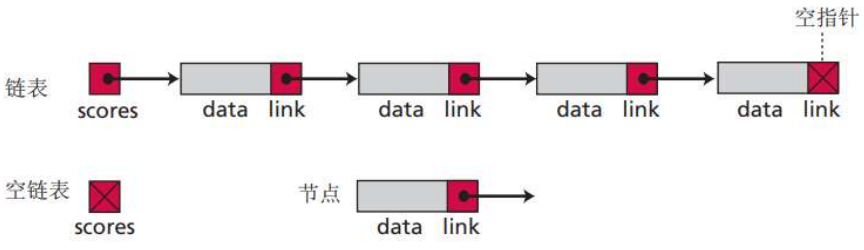
4 数组与记录数组

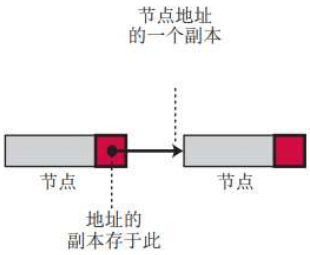
都表示数据项的列表，数组可看成记录数组的一种特例，其中每个元素只带一个域的记录。

三、链　表

链表是一个数据的集合，其中每个元素包含下一个元素的地址；即每个元素包含两部分：数据和链。

数据部分包含可用的信息，并被处理。链则将数据连在一起，它包含一个指明列表中下一个元素的指针（地址）。

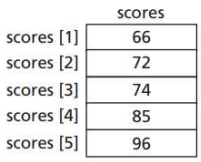




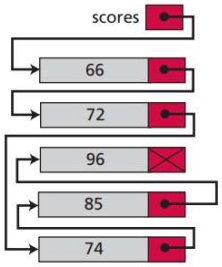
1 数组与链表

都能表示内存中的数据项列表，区别在于数据项连接在一起的方式。

记录数组中，连接工具是索引。链表中，连接工具是指向下一元素的链（指针或下一元素的地址）。



a. 数组表示

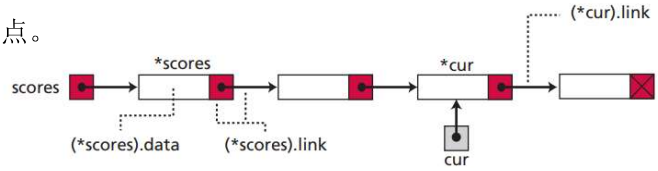


b. 链表表示

2 链表名与节点名

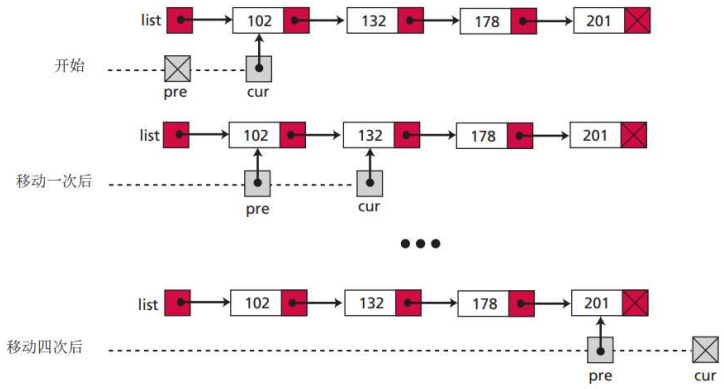
链表名是头指针的名字，该头指针指向表中的第一个节点。

如果指向节点的指针称为 p，我们称节点为*p。



3 链表操作

查找链表　链表的查找只能是顺序的，我们使用两个指针 pre 和 cur。



每一步移动中有

pre ← cur 和 cur ← (*cur).link

插入节点 首先使用查找算法，如果查找算法返回的标记为假，将允许插入。

插入空表 如果表是空（list=null），新数据项被作为第一个元素插入。

list ← new

开始处插入 如果查找算法返回的标记为假，pre 指针的值为空，那数据就需要插在表的开始处。

(*new).link ← cur 和 list ← new

末尾处插入 如果查找算法返回的标记为假，cur 指针的值为空，那数据就需要插在表的末尾。

(*pre).link ← new 和(*new).link ← null

中间插入 如果查找算法返回的标记为假，两个指针的值都不为空，那数据就需要插在表的中间。

(*new).link ← cur 和(*pre).link ← new

删除节点 首先使用查找算法，如果查找算法返回的标记为真，将允许删除。

删除首节点 如果 pre 指针为空，首节点将被删除。

list ←(*cur).link

删除中间或末尾节点 如果两个指针都不为空，那要删除的节点是中间节点或末尾节点。

(*pre).link ←(*cur).link

检索节点 为了检查或复制节点中所含数据而随机访问节点。检索只使用 cur 指针，它指向被查找算法找到的节点。

遍历链表 需要一个“步行”指针，当元素被处理时，它从一个节点移到另一个节点。使用循环。

4 链表的应用

如果需要大量的插入和删除，那么链表是合适的，但查找一个链表比查找一个数组要慢。

复 习 题

Q11-1 给出数据结构的三种类型名称。

答：数组、记录、链表。

Q11-2 数组元素和记录元素的区别是什么？

答：大多数语言中数组的所有元素都是相同类型的。

记录的元素可以是相同的或不同的类型，但所有元素必须相互关联。

Q11-3 数组元素和链表元素的区别是什么？

答：数组的元素在内存中是连续的，可以通过使用索引访问。

链表的元素存储在可能分散在整个内存中的节点中，只能通过链表的访问函数访问。

Q11-4 为什么用索引而不是下标来标注数组元素？

答：用中括号而不是下标来索引，可以我们让在索引中使用变量。

Q11-5 数组元素在内存中如何存储？

答：数组连续地存储在内存中。大多数计算机使用行主序存储来存储二维数组。

Q11-6 记录中域的定义是什么？

答：域是具有含义的最小命名数据。

Q11-7 在链表中节点的域是什么？

答：数据和下一个节点的指针。

Q11-8 链表中指针的功能是什么？

答：标识链表中的下一个元素。

Q11-9 如何指向链表中的第一个节点？

答：头指针指向表中第一个节点。

Q11-10 链表中最后一个节点的指针指向什么？

答：链表中最后一个节点的指针是空指针，什么也不指向。

练 习 题

P1.1 两个数组 A 和 B，各有 10 个整数， 要求测试数组 A 中每个元素是否与数组 B 中对应元素相等，写出该算法。

解：比较两个数组的伪代码如下：

```
算法：Compares array A with array B
目的：测试两个数组中对应元素是否相等
前提：包含 10 个整数的数组 A 和 B
后续：无
返回：true 或 false
{
  i ← 1
  while (i ≤ 10)
  {
    if A[i] ≠ B[i]          // A 不等于 B
      return false
    i ← i + 1
  }
  return true               // A 等于 B
}
```

P11-2 写出一个算法，要求倒序排列数组中元素，即最后一个变成第一个，倒数第二个成为第一，以此类推。

解：倒序排列数组的伪代码如下：

```
算法：ReverseArray (A, n)
目的：倒序排列数组中元素
前提：包含 n 个元素的数组 A
后续：数组 A 元素倒序
返回：
{
  i ← 1
  j ← n
  while (i < j)
  {
    Temp ← A[j]
    A[j] ← A[i]
    A[i] ← Temp
    i ← i + 1
    j ← j - 1
  }
}
```

P11-3 写出一个算法，要求打印出具有 R 行和 C 列的二维数组中的内容。

解：打印二位数组元素的伪代码如下：

```
算法：PrintArray (A, r, c)
目的：打印二维数组的内容
前提：数组 A
后续：打印的元素
返回：
{
```

```
  i ← 1
  while (i ≤ r)
  {
    j ← 1
    while (j ≤ c)
    {
      print A[i][j]
      j ← j + 1
    }
    i ← i + 1
  }
}
```

P11-4 写出一个算法，在具有 N 个元素的数组上应用顺序查找。

解：下面是在无序数组上应用顺序查找的伪代码。如果在数组中找到目标，则将 flag 设置为 true，跳出循环，返回 *i* 为目标的位置，flag 的值为 true。如果没有找到目标，就跳出循环，返回 *i* 的值为 *n*+1，flag 的值为 false。

```
算法：SequentialSearchArray (A, n, x)
目的：在具有 n 个元素的数组 A 上应用顺序查找
前提：A, n, x                                  // x 为要找的目标
后续：无
返回：flag, i
{
  flag ← false
  i ← 1
  while ((i ≤ n) or (flag = false))
  {
    if (A[i] = x)
      flag ← true
    i ← i + 1
  }
  return (flag, i)                             //若 x 未找到，flag 为 false
}
```

P11-5 写出一个算法，在具有 N 个元素的数组上应用折半查找。

解：下面是在有序数组上应用折半查找的伪代码。若 flag 为 true，表示 *x* 被找到，位置为 *i*；若 flag 为 false，表示 *x* 没有被找到，*i* 是 target 应该所在的位置。

```
算法：BinarySearchArray (A, n, x)
目的：在具有 n 个元素的数组 A 上应用折半查找
前提：A, n, x                                  // x 为要找的目标
后续：无
返回：flag, i
{
  flag ← false
  first ← 1
  last ← n
  while (first ≤ last)
  {
    mid = (first + last) / 2
    if (x < A[mid])
      Last ← mid - 1
    if (x > A[mid])
```

```
      first ← mid + 1
    if (x = A[mid])
      first ← Last + 1                    // x 被找到
  }
  if (x > A[mid])
    i = mid + 1
  if (x ≤ A[mid])
    i = mid
  if (x = A[mid])
    flag ← true
  return (flag, i)
}
```

P11-6 写出一个算法，在有序的数组中插入一个元素，算法必须调用查找算法找到插入的位置。

解：在有序数组中插入一个元素的主算法伪代码如下：

```
算法: InsertSortedArray (A, n, x)
目的: 在有序数组中插入一个元素
前提: A, n, x                              // x 为要插入的值
后续: 无
返回: A
{
  {flag, i} ← BinarySearch (A, n, x)      // 调用折半查找算法
  if (flag = true)                         // x 已经在 A 中
  {
    print (x 已经在 A 中)
    return
  }
  ShiftDown (n, A, i)                      // 调用后移算法
  A[i] ← x
  return
}
```

其中的子算法 ShiftDown 的伪代码如下：

```
算法: ShiftDown (n, A, i)
目的: 将索引 i 及其后的所有元素后移一位
前提: A, n, i
后续: 无
返回: A
{
  j ← n
  while ( j > i - 1)
  {
    A[j + 1] ← A[j]
    j ← j - 1
  }
}
```

P11-7 写出一个算法，在有序的数组中删除一个元素，算法必须调用查找算法找到删除的位置。

解：从有序数组中删除一个元素的主算法伪代码如下：

```
算法: DeleteSortedArray (A, n, x)
目的: 从有序数组中删除一个元素
前提: A, n, x                              // x 为要删除的值
```

```
后续：无
返回：
{
  {flag, i} ← BinarySearch (A, n, x)      // 调用折半查找算法
  if (flag = false)                        // x 不在 A 中
  {
    print (x 不在 A 中)
    return
  }
  ShiftUp (A, n, i)                        // 调用前移算法
return
}
```

其中的子算法 ShiftUp 的伪代码如下：

```
算法：ShiftUp (A, n, i)
目的：将索引 i 及其后的所有元素前移一位
前提：A, n, i
后续：无
返回：A
{
  j ← i
  while ( j ≤ n + 1)
  {
    A[j] ← A[j + 1]
    j ← j + 1
  }
  return
}
```

P11-8 写出一个算法，给数组中的每个元素乘以一个常数。

解：给数组中的每个元素乘以一个常数的算法伪代码如下：

```
算法：MultiplyConstant (A, n, C)
目的：给数组中的每个元素乘以一个常数
前提：A, n, C                              // C 为常数
后续：无
返回：
{
  i ← 1
  while (i ≤ n)
  {
    A[i] ← C × A[i]
    i ← i + 1
  }
}
```

P11-9 写出一个算法， 要求将一分数 （Fr1）加到另一分数（Fr2）上。

解：分数加法算法的伪代码如下：

```
算法：AddFraction (Fr1, Fr2)
目的：求两个分数的和
前提：Fr1, Fr2                             // 假定分母不为 0
后续：无
返回：Fr3
{
```

```
  x ← gcd (Fr1.denom, Fr2.denom)            // 调用最大公约数算法 gcd（见第八章）
  y ← (Fr1.denom × Fr2.denom) / x           // y 为最小公分母
  Fr3.num ← (y / Fr1.denom) × Fr1.num + (y / Fr2.denom) × Fr2.num
  Fr3.denom ← y
  z ← gcd (Fr3.num, Fr3.denom)              // 约分
  Fr3.num ← Fr3.num / z
  Fr3.denom ← Fr3.denom / z
  return (Fr3)
}
```

P11-10 写出一个算法，要求从一分数（Fr2）中减去另一分数（Fr1）。

解：下面是分数减法算法的伪代码。先取 Fr1 的相反数，再调用 AddFraction 把它与 Fr2 相加。

```
算法：SubtractFraction (Fr1, Fr2)
目的：求两个分数的差（Fr2 - Fr1）
前提：Fr1, Fr2                              // 假定分母不为 0
后续：无
返回：Fr3
{
  Fr1.num ← - Fr1.num                       // 取 Fr1 的相反数
  Fr3 ← AddFraction (Fr1, Fr2)              // 调用分数加法算法 AddFraction
  return (Fr3)
}
```

P11-11 写出一个算法，要求一分数（Fr1）乘以另一分数（Fr2）。

解：分数乘法算法的伪代码如下：

```
算法：MultiplyFraction (Fr1, Fr2)
目的：求两个分数的乘积
前提：Fr1, Fr2                              // 假定分母不为 0
后续：无
返回：Fr3
{
  Fr3.num ← Fr1.num × Fr2.num
  Fr3.denom ← Fr1.denom × Fr2.denom
  z ← gcd (Fr3.num, Fr3.denom)              // 约分
  Fr3.num ← Fr3.num / z
  Fr3.denom ← Fr3.denom / z
  return (Fr3)
}
```

P11-12 写出一个算法，要求用分数（Fr2）除分数（Fr1）。

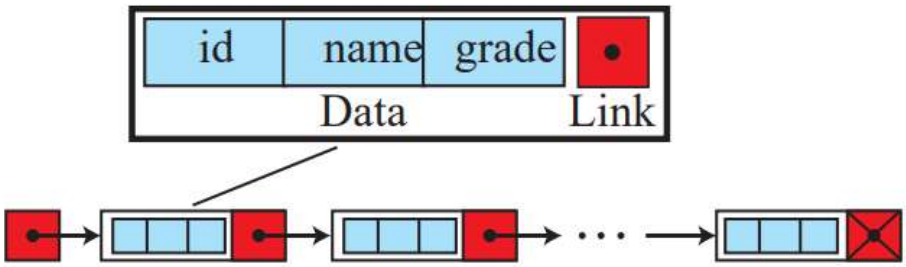
解：下面是分数除法算法的伪代码。先取 Fr2 的倒数，再调用 MultiplyFraction 把它与 Fr1 相乘。

```
算法：DivideFraction (Fr1, Fr2)
目的：求两个分数的商（Fr1 ÷ Fr2）
前提：Fr1, Fr2                              // 假定分母不为 0
后续：无
返回：Fr3
{
  Temp ← Fr2.denom
  Fr2.denom ← Fr2.num
  Fr2.num ← Temp
  Fr3 ← MultiplyFraction (Fr1, Fr2)         // 调用分数乘法算法 MultiplyFraction
```

```
  return (Fr3)
}
```

P11-13 画出一个示意图，显示数据部分是学生记录的链表，记录中有标识（id）、姓名（name）和成绩（grade）。

解：



P11-14 显示链表的删除算法（11.3.3 节中的算法 11.4）是如何删除链表中唯一的一个节点的。

解：**SearchLinkedList** 算法返回 **pre** = *null*，**cur** 指向唯一节点，**flag** = *true*。由于 **pre** = *null*，所以动作是 `list ← (*cur).link`，相当于 `list ← null`。结果是一个空表，如下图所示：

P11-15 显示链表的插入算法 （11.3.3 节中的算法 11.3）是如何在空链表中插入一个节点的。

解：因为 *list* = *null*，所以 **SearchLinkedList** 算法执行 `new ← list`，这就创建了一个只有一个节点的表。

P11-16 显示我们如何使用插入算法（11.3.3 节中的算法 11.3） 从零开始建立一链表。

解：下面是建立链表算法的伪代码。这里调用了 InsertLinkedList 算法。

```
算法：BuildLinkedList (data records)
目的：从零开始建立一链表
前提：给定的数据记录列表
后续：无
返回：第一个节点有指针指向的链表
{
  list ← null
  while (more data records)
  {
    InsertLinkedList (list, next record.key, next record)
  }
  return (list)
}
```

P11-17 写一个算法，求出数字链表中数字的平均数。

解：求数字链表中数字平均数算法的伪代码如下：

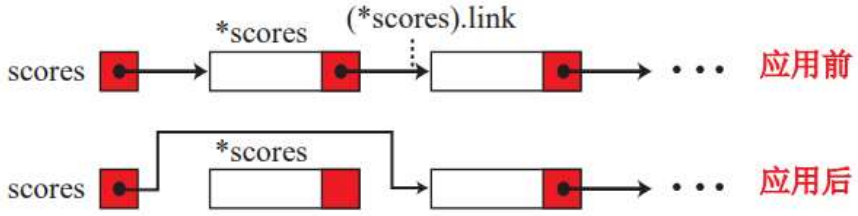
```
算法：LinkedListAverage (list)
目的：求出链表中数字的平均数
前提：链表
后续：无
返回：平均数
```

```
{
  counter ← 1
  sum ← 0
  walker ← list
  while (walker ≠ null)
  {
    sum ← sum + (*walker).data
    walker ← (*walker).link
    counter ← counter + 1
  }
  average ← sum / counter
  return average
}
```

P11-18 如果我们对图 11-9 的链表应用下列语句，将会发生什么？

`scores ← (*scores).link`

解：下图为应用这条语句前后的链表。该语句使第一个节点脱落了。本题表明，我们永远不应该移动头指针，否则表中数据将会丢失。



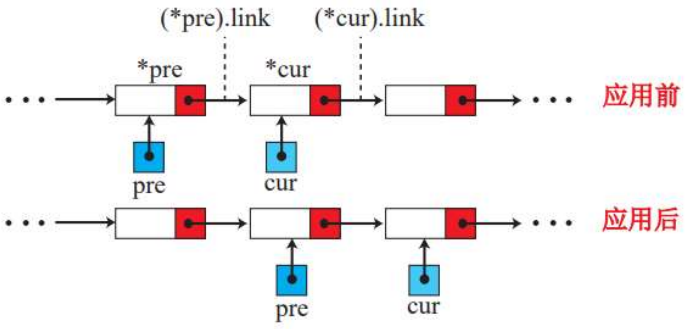
P11-19 如果我们对图 1-13 的链表应用下列语句，将会发生什么？

`cur ←(*cur).link` 和 `pre ←(*pre).link`

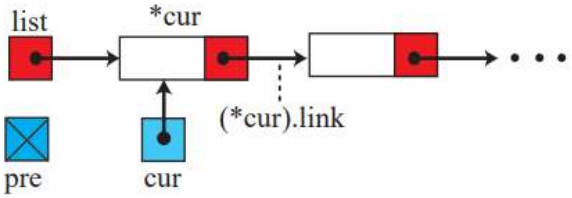
解：**a.** 如下图 a，如果 **pre** 非空，这两条语句将两个指针一起向右移动。在这种情况下，这两个语句与我们在课文中讨论的语句 `pre ← cur` 和 `cur ← (*cur).link` 是等价的。

b. 然而，当 **pre** 为空时，语句 `pre ←(*pre).link` 不起作用，因为在这种情况下，`(*pre).link` 不存在（如下图 **b**）。因此，我们应该避免使用这种方法。

。



(a)



(b)

哈尔滨工程大学本科生考试试卷（A 卷）

（2018-2019 年 第一 学期）

课程编号： 201406101 201406120 **课程名称：** 计算机科学与技术专业导论

1. 请解答：

（1）操作系统的组成是什么？ （5 分）

（2）应用程序和操作系统的不同点是什么？ （5 分）

2. 使用选择排序算法，手工排序下列数据列表，给出每轮排序后的结果以及“墙”的位置。（10 分）

14 7 23 31 40 56 78 9 2

3. 请解答：

（1）比较和对照计算机语言中的过程式模式与面向对象模式。 (6 分)

（2）汇编语言与机器语言有哪些区别？ （4 分）

4. 请解答：

（1）软件开发的 4 个阶段是什么？ （4 分）

（2）描述白盒测试与黑盒测试的区别。 （6 分）

5. 请解答：

（1）数组元素和链表元素的区别是什么？ （6 分）

（2）一个二叉树有 10 个节点。树的中序遍历和前序遍历如下。画出这棵树。　　（6 分）
前序：JCBADEFIGH
中序：ABCEDFJGIH

6. 请解答：

（1）在索引文件中，索引是如何关联数据文件的？　　(5 分)

（2）文本文件和二进制文件的区别是什么？　　(5 分)

7. 在关系数据库中，什么叫关系？什么叫属性？什么叫元组？　　（6 分）

8. 对下面给定频率的字符进行赫夫曼编码，给出计算过程。　　(10 分)

A(12), B(8), C(9), D(20), E(31), F(14), G(8)

9. 请解答：

（1） 对称密钥密码术和非对称密钥密码术有什么不同？　　（6 分）

（2） 在非对称密钥密码中，那个密钥用于加密？那个密钥用于解密？　　（4 分）

10. 请解答：

（1）说明图灵机的组成和每一部件的功能。　　（6 分）

（2）描述图灵测试。　　（6 分）