第 4 章　控制系统应用程序设计

任何一个计算机控制系统，都包括一个相应的软件支持系统。如何组织这个软件支持系统，是一个复杂的问题。为把软件系统组织并设计好，就需要掌握程序结构、数据结构、软件的组织方法等方面的内容。

4.1　应用程序设计的基本原则与方法

控制系统中控制任务的实现最终是靠程序的执行来完成的。应用程序设计得如何，将决定整个控制系统的效率和它的优劣。

4.1.1　应用程序设计的基本步骤

图 4-1 给出了应用软件设计的流程图，它描述了应用程序设计的基本任务和设计过程。

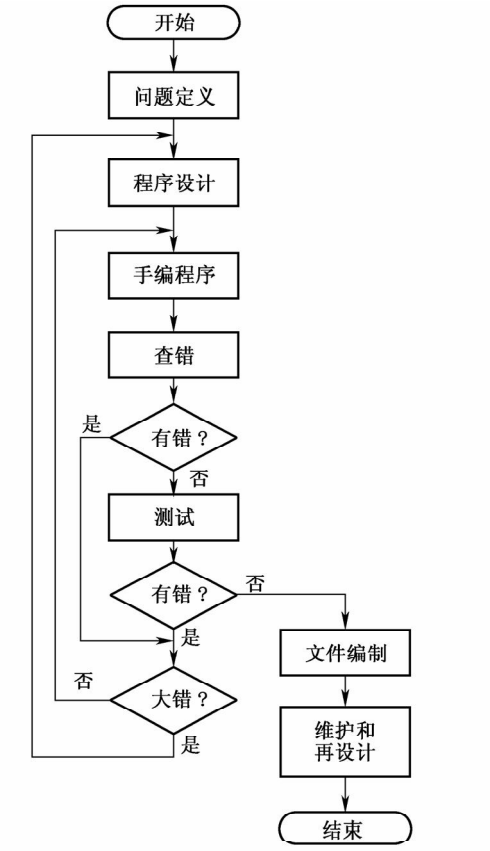


图 4-1　应用软件设计流程图

问题的定义是确定控制任务对微型机控制系统的要求，它包括定义输入和输出、处理要求、系统具体指标（如执行时间、精度、响应时间等）、出错处理方法等。程序设计是指把所定义的问题用程序的方式对控制任务进行描述。这一步要用到流程图和模块程序、自顶向下设计、结构程序等程序设计技术。手编程序是把设计框图变成微型机能接受的指令。实时控制中通常是用汇编语言编写程序，然后汇编成机器语言。查错是用来发现编程中的错误。在查错阶段可以利用诸如查错程序，断点、跟踪、模拟程序，逻辑分析器、联机仿真器等手段。测试也称程序正确性确认，通过测试保证程序正确完成要求的任务。在测试这一步要注意选择正确的测试数据和测试方法。文件编制用流程图、注释、存储器分配说明等方法来描述程序并形成文件，以便于用户和操作人员了解。维护和再设计是对程序进行维护、改进和扩充，以解决现场设备发生的问题，有时还要有特殊的诊断手段（或程序）及维护手段，有时为满足新的要求和处理任务，可能需要改进或扩充程序。

4.1.2 应用程序设计的目标及原则

1. 设计目标

（1）可靠性。软件可靠性指在规格制定、设计、实现和测试等过程中排除和避免各种将来可能发生的故障，以及在运行中一旦发生故障时可采取的各种解脱措施。软件可靠性与硬件可靠性在本质上是有区别的：硬件故障一般是由于器件物理机理的病变和衰老所致，而软件故障往往是由于设计错误和实现差错所致。因此，在一定程度上，软件可靠性必须在设计阶段就确定，到以后生产和运行阶段再考虑就晚了，也困难了。

（2）可修改性。软件可修改性是指需要有文档完备的、方法科学的、表达可理解的设计，这种设计结构化良好，系统性能易于调整。

（3）可理解性。设计可理解性是软件可靠性与可修改性的前提条件。可理解性并不单纯是文档清晰可读的问题，很大程度上取决于设计者的创造性和洞察力、对设计对象本身是否透彻掌握，同时还依赖于设计工具和方法论的应用。

（4）可测试性。在限定的成本范围内，设计一个适当的测试数据集合，该数据集合能对所建立的系统进行测试，并应保证设计能得到全面的检验。

（5）效率。指软件设计完成后，在一种计算机上实现，生成可执行代码及其运行的效率如何。效率一般可用程序执行时间和存储需求来度量。

另外，还需要考虑软件设计质量方面的目标等。

2. 软件设计原则

一个良好的设计，要求模块分解清晰、层次结构分明，并具有高度的模块独立性。为了获得良好的软件设计，应遵循的原则主要有：

（1）模块结构性原则。模块性是指将软件组织的数据流与控制流分解为适当的模块及其接口，它对设计目标的实现起着很大的作用。对于可理解性，是由于模块性提供了良好的结构性；对于可靠性，是由于模块性提供了错误发生的局部性；对于可修改性，是由于模块性提供了模块的相对独立性。对模块的基本要求是：

- 模块应当足够小且简单，单个模块所包含的功能关联程度高。
- 模块之间的接口应当足够小且简单，即模块间的互连或交互性应当很低。
- 不必知道模块的内部结构，便能够方便地将它们组合成一个大的系统。

尽管模块性思想似乎很合理，但在实践中往往并不成功。例如，模块过小会使编译开销很大。同时，当将许多模块组合起来时，往往会出现困难，模块之间的接口往往需要修改，一些模块甚至需要重写。结构化程序设计思想出现以后，模块性思想被赋予结构化的新含义。于是便出现了模块结构性的设计思想，这种软件设计的基本思想可以概括如下：一个软件系统应能分解为各个组成部分，而这些部分之间是以一定的层次结构组织起来的；在这种分解的每一层次上只能具有规定的、有限制的结构形式（称为构件），如顺序、循环、选择等。

（2）抽象性原则。抽象有助于软件设计具有良好的层次结构，每一层代表一定程度的抽象。层次上升，抽象性提高；层次下降，具体性提高。抽象性原则对大而复杂的系统设计尤为重要。

（3）局部性原则。彼此关系密切的元素集中在一起，某一个概念（如数据、过程、子程序）仅在一处被说明和定义，其他地方需要时引用即可。这些都是局部性原则的应用。

（4）信息隐藏原则。这一原则与抽象原则密切相关。高层只考虑较抽象的属性，较细致的属性暂不考虑，隐藏起来，留给下层考虑。上下关系如此，内外关系也是如此。模块内部的结构对外是隐藏的，外界只见到它的外在表现，只通过接口调用它的功能。这些都是信息隐藏原则的应用。

4.1.3　应用程序设计的方法

1. 模块化程序设计

模块化程序设计的出发点是把一个复杂的系统软件分解为若干个功能模块，每个模块执行单一的功能，并且具有单入口单出口结构。模块化程序设计的传统方法是建立在把系统功能分解为程序过程或宏指令的基础上。但在很大的系统里，这种分解往往导致大量过程，这些过程虽然容易理解，但却有复杂的内部依赖关系，因此带来一些问题不好解决。

（1）自底向上模块化设计。首先对最低层模块进行编码、测试和调试。这些模块正常工作后，就可以用它们来开发较高层的模块。例如，在编主程序前，先开发各个子程序，然后用一个测试用的主程序来测试每一个子程序。这种方法是汇编语言设计常用的方法。

（2）自顶向下模块化设计。首先对最高层进行编码、测试和调试。为了测试这些最高层模块，可以用“节点”来代替还未编码的较低层模块，这些“节点”的输入和输出满足程序说明部分的要求，但功能少得多。该方法一般适合用高级语言来设计程序。

上述两种方法各有优缺点。在自底向上开发中，高层模块设计中的根本错误也许要很晚才能发现。在自顶向下开发中，程序的大小和性能往往要在开发关键性的低层模块时才会表现出来。实际工作中，最好将两种方法结合起来。先开发高层模块和关键性低层模块，并用“节点”来代替以后开发的不太重要的模块。

2. 结构化程序设计

结构化程序设计的概念最早由 E.W.Dijkstra 提出。1965 年他在一次会议上指出：“可以从高级语言中取消 GO TO 语句”“程序的质量与程序中所包含的 GO TO 语句的数量成反比”。1966 年 C.Böhm 和 G.Jacopini 证明了只用 3 种基本控制结构就能实现任何单入口单出口的程序。这 3 种基本控制结构是顺序结构、选择结构、循环结构，它们的流程图如图 4-2 所示。

实际上用顺序结构和循环结构（又称 DO-WHILE 结构）完全可以实现选择结构（又称 IF-THEN-ELSE 结构），因此理论上最基本的控制结构只有两种。Böhm 和 Jacopini 的证明为结构化程序设计技术奠定了理论基础。

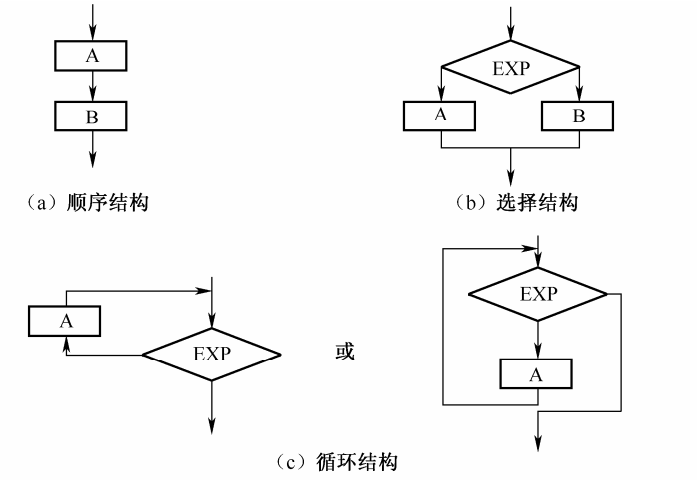


图 4-2　程序的基本控制结构流程图

结构化程序设计是一种程序设计技术，它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。关于逐步求精方法 Niklaus Wirth 曾做过如下说明：“我们对付复杂问题的最重要的办法是抽象，因此，对一个复杂的问题不应该立即用计算机指令、数字和逻辑符号来表示，而应该用较自然的抽象语句来表示，从而得出抽象程序。抽象程序对抽象的数据进行某些特定的运算并用某些合适的记号（可能是自然语言）来表示。对抽象程序作进一步的分解，并进入下一个抽象层次，这样的精细化过程一直进行下去，直到程序能被计算机接受为止。这时的程序可能是用某种高级语言或机器指令书写的。”

在总体设计阶段采用自顶向下逐步求精的方法，可以把一个复杂问题的解法分解和细化成一个由许多模块组成的层次结构的软件系统。在详细设计或编码阶段采用自顶向下逐步求精的方法，可以把一个模块的功能逐步分解细化为一系列具体的处理步骤或某种高级语言的语句。

3. 应用软件的设计

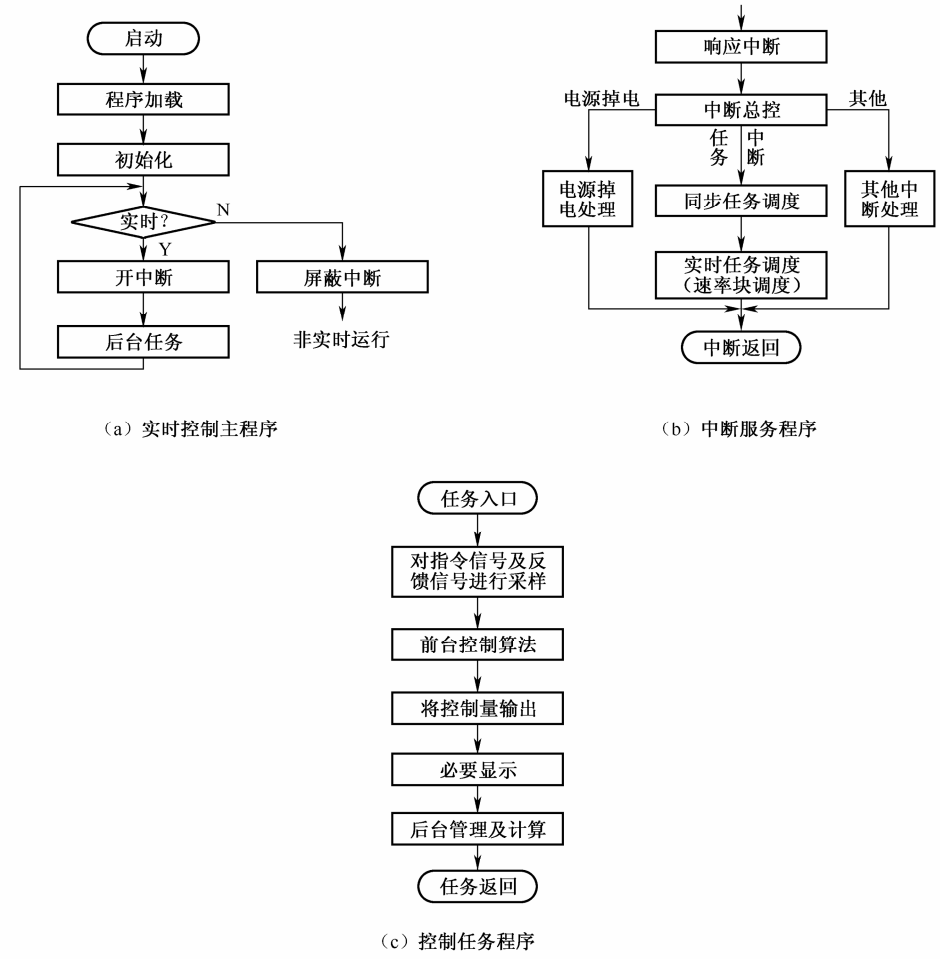
微机控制系统的应用软件是用户为解决实时控制问题而提出的，一般由用户自行设计和编制。根据应用程序的功能，可将应用程序分为：控制程序、数据采集及处理程序、巡回检测程序、数据管理程序。

（1）控制软件的设计。对于微机控制系统来说，实时控制软件基本包括：实时管理软件和过程监视及控制算法计算软件两大部分。

1）实时管理软件。实时管理软件是对整个控制系统进行管理用的程序，包括对应用控制程序的调度、I/O 管理、中断管理、实时管理等。实时管理软件相当于整个微机控制系统的主程序。

2）过程监视及控制算法计算软件。过程监视及控制算法计算软件主要是根据采集的信息、输入的指令以及所设计的控制算法，产生不同的控制指令的计算程序，主要包括：数据变换处理程序（如数字滤波、单位换算、数据合理性检查、数据补偿校正等）、控制指令生成程序（如控制器算法计算、系统状态控制、控制指令输出等）、事故处理程序（如对系统不同故障的处理指令生成等）、信息管理程序（如数据存储、输出、打印、显示、文件管理等）。

典型微机控制系统的控制软件流程如图 4-3 所示。

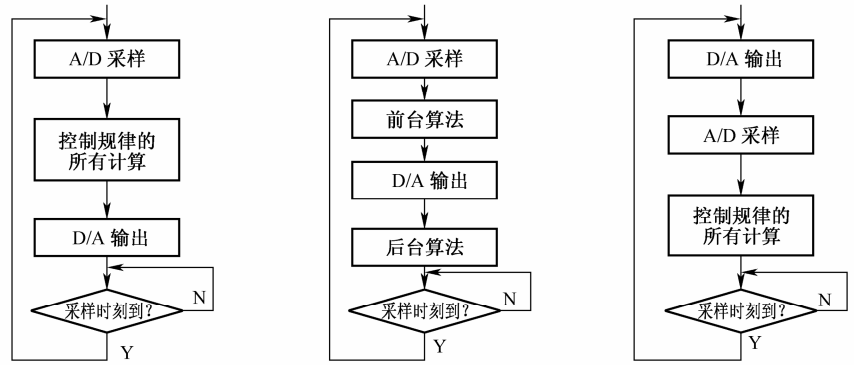


（a）实时控制主程序　　（b）中断服务程序

（c）控制任务程序

图 4-3　典型微机控制系统的控制软件流程图

（2）控制算法设计中减少计算延时的方法。通常控制算法实现时有 3 种输入输出方法，这 3 种控制算法的对应流程框图如图 4-4 所示。



（a）控制算法末端输出　（b）控制算法中间输出　（c）下一采样时刻输出

图 4-4　3 种控制算法的流程框图

4. 工业控制组态软件

目前，越来越多的控制工程师已不再采用芯片→电路设计→模块制作→系统组成调试→……的传统模式来研制计算机控制系统，而是采用组态模式。计算机控制系统的组态功能可分为两个主要方面：硬件组态和软件组态。

硬件组态常以总线式工业控制机（PC 总线或 STD 总线）为主进行选择和配置。总线式工业控制机具有小型化、模型化、标准化、组合化、结构开放的特点，因此在硬件上可以根据不同的控制对象选择相应的功能模板，组成各种不同的应用系统，使硬件工作量几乎接近于零，只需按要求对各种功能模板安装与接线即可。

软件组态常以工业控制组态软件为主来实现。工业控制组态软件是标准化、规模化、商品化的通用过程控制软件。控制工程师在不必了解计算机硬件和程序的情况下，在 CRT 屏幕上采用菜单方式，用填表的方法对输入、输出信号用“仪表组态”方法进行软连接。这种通用树形填空语言有简单明了、使用方便等特点，十分适合控制工程师掌握应用，大大减少了重复性、低层次、低水平应用软件的开发，提高了软件的使用效率和价值，提高了控制的可靠性，缩短了应用软件的开发周期。因此，工业控制组态软件是性能优良的软件产品。

然而，以往计算机控制系统的软件功能（如实时数据库、历史数据库、数据点的生成、图形、控制回路、报表功能的实现）是靠软件人员通过编程实现的，工作量大得惊人。这样设计出来的软件通用性极差，对于每个不同的应用对象都要重新设计或修改程序，这种方法实现的软件功能可靠性也较低。近几年来，工业控制组态软件得到了广泛的重视和迅速的发展。目前，我国已开发出很多成功的组态软件，而且技术发展很快，与国际水平相差不大，如组态王组态软件、力控组态软件等，当然有些国外的组态软件已开始了汉化工作。大多数的组态功能是离线进行实现的，即在应用系统的设计开发阶段仔细地完成系统的组态和配置。控制系统的软件组态是生成整个系统的重要技术，对每一个控制回路分别依照其控制回路图进行。组态工作是在组态软件支持下进行的，组态软件主要包括：控制组态、图形生成系统、显示组态、I/O 通道登记、单位名称登记、趋势曲线登记、报警系统登记、报表生成系统共 8 个方面的内容。有些系统可根据特殊要求而进行一些特殊的组态工作。控制工程师利用工程师键盘，以人机会话方式完成组态操作，系统组态结果存入磁盘存储器中，以作为运行时使用。下面对上述 8 种组态功能进行简单的介绍。

（1）控制组态。在工业控制组态软件中，一般有 PID 等几十种基本算法。控制算法的组态生成在软件上可以分为两种实现方式：一种方式是采用模块宏的方式，即一个控制规律模块（如 PID 运算）对应一个宏命令（子程序），在组态生成时，每用到一个控制模块，则组态生成控制算法，产生的执行文件中就将该宏所对应的算法换入执行文件；另一种常用的方式是将各控制算法编成各个独立的可以反复调用的功能模块，对应每一模块有一个数据结构，该数据结构定义了该控制算法所需要的各个参数，因此只要这些参数定义了，控制规律就定了，有了这些算法模块，就可以生成绝大多数的控制功能。

（2）图形生成系统。计算机控制系统的人机界面越来越多地采用图形显示技术。图形画面主要是用来监视生产过程的状况，并可通过对画面上对象的操作实现对生产过程的控制。图形画面一般有两种：静态画面（或背景画面）和动态画面。静态画面一般用来反映监视对象的环境和相互关系，它的显示是不随时间而变化的。动态画面一般用以反映被监视对象和被控对象的状态与数值等，它在显示过程中是随现场被监控对象的变化而变化的。在生成图形画面时，

不但要有静态画面，而且还要有“活”的部分，即动态画面。

（3）显示组态。计算机控制系统的画面显示一般分为三级，即总貌画面、组貌画面、回路画面。若想构成这些画面，就要进行显示组态操作。显示组态操作包括选择模拟显示表、定义显示表、显示登记方法等操作。

1）选择模拟显示表。由于计算机控制系统显示画面常采用各种模拟显示表来显示测量值、设定值和输出值，因此显示组态一般可用 6 种模拟显示表，即调节控制表、报警显示表、阀位操作表、监视操作表、比率设定表、流量累计表。

2）定义模拟显示表。选择了回路的模拟显示表后，尚需对显示表的每一个参数进行确定，并在画面上设定相应的值。

3）显示登记法。显示登记法是进入系统显示登记画面，选择过程控制站站号及工作方式；登记控制组号、组名、该组员的回路号，进行分组登记操作；显示表登记（登记每一个控制回路所用的模拟显示表）；将显示登记文件存入后备文件或打印。

（4）I/O 通道登记。计算机控制系统能支持多种类型的信号输入和输出。从生产过程来看，每一输入输出都有不同的名称和意义，因此需要将输入输出定义成特定的含义，这就是 I/O 通道登记。I/O 通道主要是模拟量 I/O 和开关量 I/O 等通道。

（5）单位名称登记。对系统各种画面中需要显示的工程单位名称采用登记的方法，可使用中英文一切符号登记生成自己特有的单位名称，主要登记编号和单位名。

（6）趋势曲线登记。趋势曲线显示在控制系统中很重要，为了完成这种功能需要对趋势曲线进行登记。系统的硬盘中保存有 3 种趋势曲线数据，即当天的数据、昨天的数据和历史的数据。当天的趋势曲线数据，系统以一定的周期将其保存起来。到第二天就将当天的数据覆盖昨天的数据。历史数据是当你需要某天的数据时，从硬盘拷贝到软盘保存起来。

趋势曲线的规格主要有：趋势曲线幅数、趋势曲线每幅条数、每条时间、显示精度。趋势曲线登记表的内容主要有：幅号、幅名、编号、颜色、曲线名称、来源、工程量上限和下限。

（7）报警系统登记。报警显示画面分成三级，即报警概况画面、报警信息画面、报警画面。报警概况画面是第一级，它显示系统中所有报警点的名称和报警次数；报警信息画面是第二级，它是第一级画面的展开与细化，调出相应的报警信息画面，即可观察到报警时间、消警时间、报警点名称、报警原因等；报警画面是第三级，可以调出与报警点相应的各显示画面，包括总貌画面、组画面、回路画面、趋势曲线画面等。

为了完成报警登记，要填写登记表，内容包括：编号、名称、原因类型、原因参数、画面类型、画面参数。

（8）报表生成系统。报表生成系统用于系统的报表及打印输出。因而报表系统的主要功能是定义各种报表的数据来源、运算方式，以及报表的打印格式和时间特性。

4.2　数据结构及其应用

在计算机控制系统中，除了数值计算和数据的输入输出外，还经常遇到非数值运算。为了设计高质量的程序，设计者不但要掌握编程技术，还要研究程序所加工的对象，即研究数据的格式、特性、数据元素间的相互关系。下面介绍常用的数据结构基本知识，包括线性表、数组、堆栈、队列、链表和树，以及数据查找和数据排序方法。

4.2.1 基本术语

数据（Data）是描述客观事物的数、字符，以及所有能输入到计算机中并被计算机程序处理的符号的集合。在计算机领域，它是程序加工的“原料”，能够被计算机输入、存储、处理和输出的一切信息都叫数据。

数据元素（Data Element）是一个数据整体中相对独立的基本单位，即数据这个集合中的一个个体（客体）。对于一个字符串来说，每个字符就是它的数据元素；对于一个数组来说，每一个成分就是它的数据元素。有时一个数据元素可由若干个数据项（Data Item）组成，数据项是数据的最小单位。

数据对象（data object）是具有相同特性的数据元素的集合，是数据的一个子集。例如，整数的数据对象是集合 N={0，±1，±2，…}，字母字符的数据对象是集合 C={A，B，…，Z}。

数据结构（data structure）简单说来是指数据以及数据之间的联系，是带有结构的数据元素的集合。被计算机加工的数据元素都不是孤立的，在它们之间存在着某种联系，这种相互之间的关系通常称为结构。

为了更确切地描述数据结构，数据结构通常采用二元组表示：

$$\text{Data-Structure}=(K, R)$$

K 是数据元素的集合，R 是 K 上二元关系的集合，其中：

$$K=\{k_i \mid 1\leqslant i\leqslant n，\ n\geqslant 0\}$$

$$R=\{r_j \mid 1\leqslant j\leqslant m，\ m\geqslant 1\}$$

k_i 表示第 i 个数据元素，n 为数据结构中数据元素的个数。若 n=0，则 K 是个空集。r_j 表示第 j 个二元关系，m 为 K 上关系的个数。

数据结构所研究的内容是数据元素之间的逻辑关系，即所谓数据的逻辑结构。而数据元素在计算机内的存储方式即是所谓数据的物理结构或存储结构。数据元素在计算机中有顺序、链接、索引、散列等多种，但可以概括为两种不同的存储结构，即顺序存储结构和非顺序存储结构（又称链式存储结构）。

4.2.2 数据结构的类型

1. 顺序结构

顺序结构就是把数据存放在从某个存储地址开始的连续存储单元中。顺序结构包括线性表、数组、堆栈和队列，其中前两种为静态顺序结构，后两种为动态顺序结构。

（1）线性表。线性表是一组有序的数据元素，可表示为

$$(a_1，\ a_2，\ \cdots，\ a_n) \qquad (4\text{-}1)$$

其中，a_i（i=1，2，…，n）是数据元素，其下标 i 表示元素的序号，代表元素在线性表中的位置。n 是表中元素的个数，定义为表的长度。当 n=0 时，则为空表。

线性表中每个数据元素的位置是固定的，元素之间的相对位置是线性的。比如线性表(4-1)中，a_1 是第一个元素，a_n 是最后一个元素。当 1<i<n 时，a_i 的前一个元素（称为直接前趋）是 a_{i-1}，后一个元素（称为直接后继）是 a_{i+1}。

在计算机中，用一组连续的存储单元依次存储线性表的数据元素，假设每个元素占用 L 个存储单元，则第 i 个元素 a_i 的存储地址 LOC(a_i)为

$$LOC(a_i)=LOC(a_1)+(i-1)\times L \tag{4-2}$$

其中，$LOC(a_1)$是线性表中第一个元素的存储地址，也称线性表首址。线性表的这种机内表示法叫做线性表的顺序映象。

对线性表可能进行的运算有以下几种：

①确定线性表的长度 n。

②存取线性表的第 i 个数据元素，检查或修改数据值。

③删除第 i 个元素。

④在第 i 个和第 i+1 个数据元素之间插入一个新的数据元素。

⑤把一个线性表拆成两个或两个以上的线性表。

⑥将两个或两个以上的线性表合并成一个线性表。

⑦对线性表中的数据元素按某个数据值递增（或递减）的顺序进行重新排序。

⑧按某个数据值查找数据元素。

并非每个表都必须进行所有这些运算，一般情况下只需进行其中一部分运算即可。运算和运算规则是构成一个数据结构不可缺少的部分，常用的运算是上述③④和⑦⑧。

（2）数组。数组（Array）是线性表的推广，其中每个元素是由一个数值和一组下标组成的。例如，一个 m×n 的矩阵式

$$A=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}=\left[a_{ij}\right] \tag{4-3}$$

可以看成一个二维数组，其中每个元素 a_{ij} 都和一个二维空间的数(i, j)（i=1，2，…，m，j=1，2，…，n）相对应。

由此可见，数组是线性表的简单推广；反之，线性表是数组的一种特例。例如，线性表（4-1）相当于数组（4-3）中的一行或一列元素。

对于数组，通常有以下两种运算：

①给定一组下标，查找与其对应的数据元素。

②给定一组下标，存取或修改与其对应的数据元素。

（3）堆栈。堆栈（Stack）是一种特殊结构的线性表，限定只能在表的一端进行插入或删除（包括存取），如图 4-5 所示。表中元素以 a_1，a_2，…，a_n 的顺序进栈，以相反的顺序 a_n，…，a_2，a_1 出栈。也就是说，后进入的元素先退出。因此，堆栈又称后进先出表 LIFO。允许插入或删除的一端称为栈顶（top），其相反的另一端则称为栈底（bottom）。有人把堆栈比喻成子弹夹，进栈的过程相当于压入（push）子弹，出栈的过程相当于弹出（pop）子弹。

设堆栈的长度为 n，用栈顶指针 sp 来描述栈。当 sp=0 时，称之为空栈；当 sp=n 时，称之为满栈。每次进栈，sp 加 1；反之，每次退栈，sp 减 1。空栈（sp=0）时想退栈，称之为下溢，满栈（sp=n）时想进栈，称之为上溢。

堆栈在程序设计中的应用十分普遍。例如，子程序的调用及其返回处理、断点保存和恢复、数据的暂存等。

对于堆栈，通常有以下 3 种运算：

①在栈顶插入一个新的数据元素（sp 加 1）。

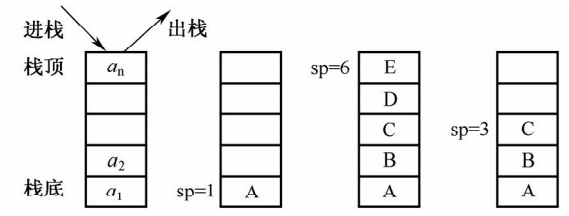


图 4-5　栈的示意图

②删去栈顶的一个数据元素（sp 减 1）。

③读栈顶的一个数据元素。

（4）队列。队列（queue）也是一种特殊的线性表，和堆栈相反，队列是先进先出表 FIFO。表中元素以 a_1，a_2，…，a_n 的顺序进入，还以相同的顺序出去，如图 4-6 所示。

允许插入的一端称为队尾（rear），允许删除的一端称为队头（front）。

设队列的长度为 n，用头指针（front）和尾指针（rear）来描述队列，如图 4-7 所示是顺时针循环队列。每次进队，rear 加 1；反之，每次出队，front 减 1。采用模 n 运算，图 4-7 所示队列的模为 6。为了正确地判断队列的溢出，避免出现假溢出，我们规定头指针 front 总是队列中实际的队头小 1 个位置，并且规定尾指针 rear 加 1 后等于头指针 front 作为队满（或队溢出）标志。

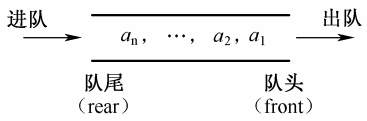


图 4-6　队的示意图

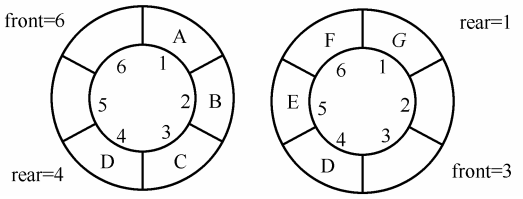


图 4-7　循环示意图

对于队列，通常有以下两种运算：

①在队列的队尾插入一个新的数据元素（rear 加 1）。

②在队列的队头删除一个数据元素（front 减 1）。

2. 链形结构

前面介绍的线性表、数组、堆栈和队列的共同特点是要求连续的存储单元来顺序存放数据元素。从而也就带来了共同的特点，一是进行插入或删除操作时，要移动大量的数据元素，并浪费时间；二是不易扩展，有时为了留有余地，将会浪费存储空间。为了克服这些缺点而采用链形结构，简称链表，如图 4-8 所示。

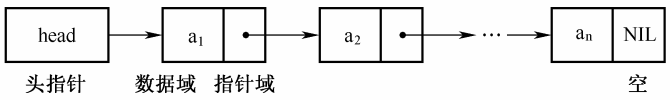


图 4-8　链表示意图

链表由若干个节点组成，每个节点有两个域：一个是数据域，用来存放数据元素；另一个是指针域，用来存放下一个节点的数据域首址。通过指针域把各节点按要求的顺序连接起来组成一个首尾相连的表，由于其组成像一根链条，故取名为链表。

为了确定链表中第一个节点的数据域首址，设置了头指针（head）；为了标识链表中的最

后一个节点，将其指针域设置为“空”（NIL），如图 4-8 所示。

对于链表这种结构，在逻辑上是有序的，用指针域指明各节点（或数据元素）之间的关系；而在物理上则可能是无序的，各节点在存储器中的物理位置可以任意配置。在使用链表时，只着眼于它的逻辑顺序，而往往不注意它的实际存储位置。

链表的插入或删除，只需要改变节点的指针域，而不必变更其物理位置。例如，图 4-9 中要在节点 a_1 和 a_2 之间插入一个新的节点 b，只需要把节点 a_1 的指针域改成指向节点 b，而把新节点 b 的指针域指向节点 a_2，a_2 以后的各节点都不必变更，至于新节点 b 的物理位置可以任意配置。又例如，图 4-10 中要删除节点 a_2，只需要把节点 a_1 的指针域改成指向节点 a_3，其余都不必变更。由此可见，链表的插入和删除非常方便，不必像上述线性表那样，插入或删除一个数据元素要变更其后所有元素的物理位置。

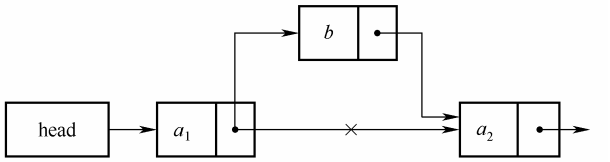


图 4-9　插入节点 b

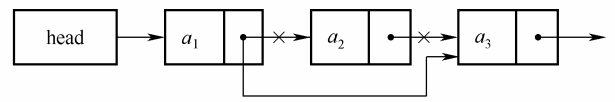


图 4-10　删除节点 a_2

如果给出链表中某节点的存储地址，便可从头指针（head）开始顺序查找到该节点。

对于图 4-11（a）所示的单链表，只能从头指针开始查找，查找时间长。为了加快查找速度，可以采用图 4-11（b）和（c）所示的循环链表和双重链表。

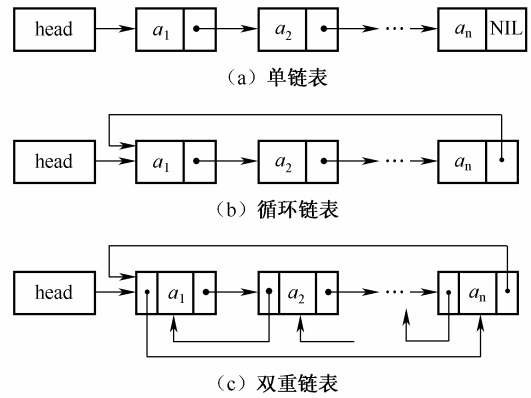


图 4-11　3 种链表结构示例

如果把单链表中最后一个节点的空指针域改成指向第一个节点的首址，那么就构成了循环链表。循环链表的优点是可以从任何一个节点开始查找所需节点。但是，单向链表和循环链表都只能单向查找。

为了能够双向查找，可以采用双重链表。双重链表中每个节点有 3 个域：左指针域、数据域和右指针域。左指针域用来链接其前趋节点，右指针域用来链接其后继节点。

链表可以用来存放大量的数据和信息。例如，计算机控制系统中众多的控制模块、运算模块、顺序逻辑模块所对应的数据区（或线性表），以及大量的操作信息、历史记录信息等，都可以以链表方式存储。

链表的运算通常有 3 种：插入节点、删除节点和查找节点。

3. 树形结构

计算机所管理的数据、信息和文件有时具有层次关系或上下级关系。例如图 4-12 所示的 7 个数据记录，每个记录有 4 个域：记录名、数据、左指针和右指针。如果把记录抽象为一个节点，则可以得到如图 4-13 所示的结构形式。其形状很像一棵倒长的树，称为树形结构，或简称树（Tree）。

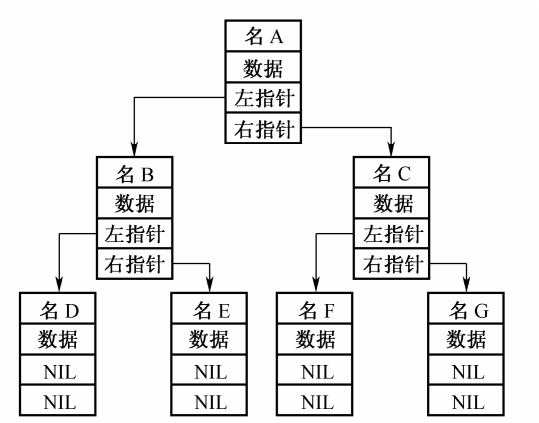


图 4-12　树形结构示例

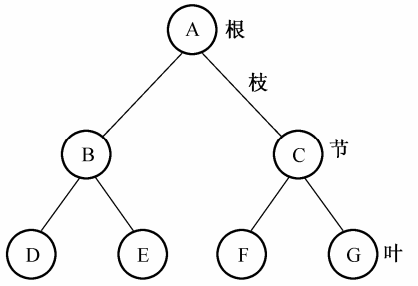


图 4-13　倒长的树

为了形象地描述树中各节点之间的层次关系，在图 4-12 中，把最高节点 A 称为树“根”，节点之间的连线称为树“枝”，具有下枝的节点称为树“节”，不具有下枝的节点称为树“叶”。

为了形象地描述树中各节点之间的层次关系，也常常用家族术语来描述。例如，在图 4-13 中，节点 A 是节点 D、E、F、G 的祖父，节点 B 是节点 D、E 的父亲，节点 C 是节点 F、G 的父亲，节点 A 又是节点 B、C 的父亲；反之，还可以用儿子、孙子等术语来描述。

4.2.3　数据查找技术

在利用线性表、数组、堆栈、队列、链表和树这些数据结构解决实际问题时，还经常遇到数据查找，例如从线性表中查找某个数据元素，就会考虑到选择何种查找方法才能节省查找时间。

在研究数据查找方法时，会用到关键字这个术语。关键字是唯一标识数据元素、节点和记录的数值（或名字）。比如，每个人的身份证号码可以作为公民的关键字，因为利用身份证号码，可以查找到这个人的性别、年龄、住址、单位、职业等。

数据查找的过程就是将待查关键字与实际关键字比较的过程。最简单的数据查找就是查表法。所谓查表法，就是把事先计算或测得的数据按一定顺序编制成表格，查表程序的任务就

是根据被测参数的值或者中间结果查出最终所需要的结果。查表是一种非数值计算方法，利用这种方法可以完成数据补偿、计算、转换等各种工作，它具有程序简单、执行速度快等优点。

1. 顺序查找

顺序查找是一种最简单的查找方法，对数据表的结构无任何要求。查找过程如下：从数据表头开始，依次取出每个记录的关键字，再与待查记录的关键字比较；如果两者相符，则表明查到了记录。如果整个表查找完毕仍未找到所需记录，则查找失败。

顺序查找速度较慢。设有 n 个记录组成的表，平均查找次数为(n+1)/2。顺序查找虽然比较“笨”，但对无序表或较短表而言，仍是一种比较常用的方法。顺序查找适用于数据记录个数较少的情况。

2. 折半查找

折半查找也叫对分查找，是一种在实际应用中最常使用的方法。对于那些表格比较长，按关键字大小顺序排列的数据表，可以采用折半查找。设有一个按关键字从小到大顺序排列的表，若待查记录的关键字为 k_i，折半查找过程如下：首先选取表中间的一个记录的关键字与 k_i 比较，如果 k_i 大于该关键字，那就再取表的后半部中间的记录，比较其关键字；如果 k_i 小于该关键字，那就再取表的前半部中间的记录，比较其关键字。这样重复进行，直至找到所需要的记录。如果没有，则查找失败。

设 8 个关键字的排列顺序为

11　13　25　27　39　41　43　45

并作符号 L、H 和 M 分别表示查找段首、尾和中间关键字序号。

设要查找关键字 41，查找过程如下：

第一次：	11	13	25	27	39	41	43	45
	↑			↑				↑
	L=1			M=4				H=8
第二次：	11	13	25	27	39	41	43	45
					↑	↑		↑
					L=5	M=6		H=8

其中，M=INT((L+H)/2)，INT 表示取整数。经过两次比较就找到了关键字 41。由此可见，折半查找速度比顺序查找速度快，但前提是事先应按关键字大小顺序排列好。

3. 分块查找

分块查找是介于顺序查找和折半查找之间的一种折衷方法。将一组关键字均匀地分成若干块，块间按大小排序，块内关键字不排序，如图 4-14 所示。

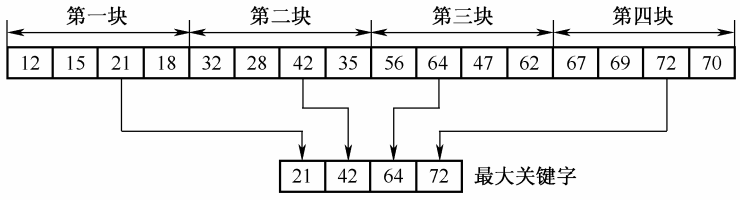


图 4-14　分块查找示意图

该图中 16 个关键字分成 4 块，第一块中的关键字都小于第二块中的关键字，第二块中的关键字又都小于第三块中的关键字，依此类推。另外，建立了一个各块中的最大关键字表。

设待查记录的关键字为 k_i（56），分块查找分两步进行：首先用折半查找法查找最大关键字表，确定 k_i 在哪一块（第三块）；然后用顺序查找法查找 k_i 所在块（第三块），从而查到所需记录。

4. 直接查找

如果记录的关键字与存储地址之间符合某一函数关系，则可以通过函数运算直接求得关键字的所在地址，以便找到相应的记录。例如，某计算机控制系统中数据采集点记录的关键字 K 与存储地址 D 之间的函数式为

$$D=K\times M+F \tag{4-4}$$

其中，M 是每个记录的字节数，F 是数据表（记录）的首地址。

采用直接查找法的数据结构应满足以下条件：

（1）关键字 K 与存储地址 D 之间应满足某个函数式 D(K)。

（2）关键字数值分散性不大，否则一块内存区将被占用得十分零碎，浪费存储空间。

4.2.4 数据排序技术

数据排序和数据查找一样，也是数据结构的辅助性运算，排序还可以促进查找方法的改进，提高查找的速度。折半查找比顺序查找的速度快，其前提是要事先进行排序。排序的目的就是把无序的数据表按关键字值大小顺序排列，变成有序的数据表。下面介绍几种常用的数据排序的方法：插入排序、希尔排序、选择排序和快速排序。

1. 插入排序

插入排序的方法是每次把第 i 个关键字与前 i-1 个逐个进行比较，一旦找到合适的位置就进行插入。这个方法类似于玩纸牌时，按大小顺序整理手中的纸牌。图 4-15 给出了一个具体实例，描述插入排序的过程。

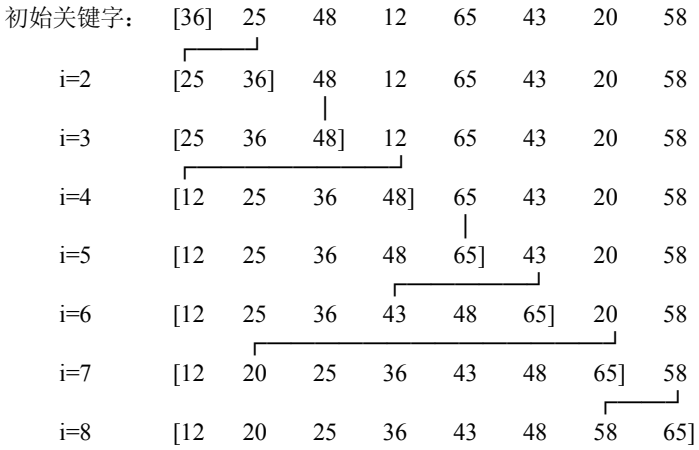


图 4-15 插入排序示例

如果有 N 个关键字，则需要比较 N-1 遍。另外，每遍比较最多要移动 N-1 个数据。插入排序的原理是最简单的，缺点是比较次数多，移动数据次数也多，因此排序速度比较慢。

2. 希尔排序

希尔排序的过程如图 4-16 所示。首先反复比较两个相距 d_1 的关键字，按大小排序；然后取 $d_2<d_1$，再反复比较两个相距 d_2 的关键字，按大小排序；然后取 $d_2<d_1$，再反复比较两个相距 d_2 的关键字，按大小排序。依此类推，直至 $d_i=1$ 为止。这时，全部记录便按关键字大小的次序排好了。

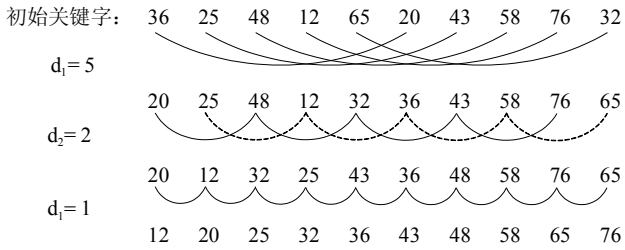


图 4-16　希尔排序示例

该方法是对插入排序的改进，每一遍以不同的增量进行插入排序。比如在图 4-16 中，第一遍增量为 4，第二遍增量为 2，第三遍增量为 1。增量为 1 时，便是插入排序。经过前面几遍的跳跃式排序，所有记录已几乎有序了，所以最后一遍作插入排序时，数据移动量较小。由于在前面几遍的排序中不需要逐项进行比较，从而减少了数据移动，提高了排序的速度。

3. 选择排序

选择排序的过程如图 4-17 所示。设共有 N 个关键字，首先找出数据表中 N 个关键字的最小项，将其与表中第一个关键字大于它的项对换；然后再在其余 N-1 个关键字中找出最小的，将其与表中第一个关键字大于它的项对换；依此类推，从 N-1 个逐步（每次减少一个）减到一个（即最大关键字）。这样，即把关键字从小到大排序完毕。

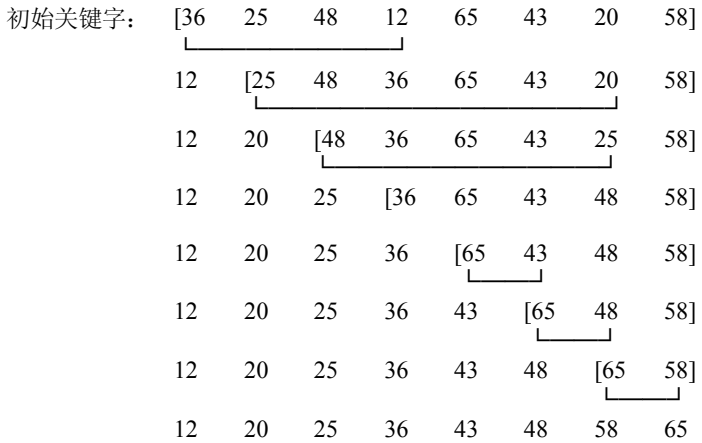


图 4-17　选择排序示例

4. 快速排序

顾名思义，快速排序法是目前内部排序（指全部要排序的记录都在内存中）中速度较快的一种排序方法。在快速排序中，记录的比较和交换是从两端向中间进行的，排序码较大的记

录一次就能够交换到后面单元，排序码较小的记录一次就能够交换到前面单元，记录每次移动的距离较远，因而总的比较和移动次数较少。它的基本原理是，首先取表中第一个关键字 K_1 作为控制关键字，从最末项 j 开始往前与 K_1 比较，找到 $K_{j-d}<K_1$ 就交换（d≥0）；再从第二个关键字 K_2 开始往后与 K_{i-d} 比较，找到 $K_i>K_{j-d}$ 再交换（i≥2）。继续此过程，直至把控制关键字 K_1 放在表中某个合适的位置 m，记成 $K_1(m)$。使得它前面的所有关键字都小于它，而它后面的所有关键字都大于它。这样，整个表以 $K_1(m)$为界而分成左、右两部分。这叫做第一遍排序。再分别对此两部分进行排序，又把此两部分各分成更小的两部分。这样继续下去，直至每部分只剩下一项为止。

设初始关键字为：

43	52	10	39	91	02	14	67
↑						↑	↑
i=1						j=1	j=n

首先取控制关键字 K_1=43，排序过程为：

	14	52	10	39	91	02	43	67
		↑						
		i=2						
	14	43	10	39	91	02	52	67
	14	02	10	39	91	43	52	67
第一遍结束→	[14	02	10	39]	43	[91	52	67]
	↑			↑		↑		↑
	i_1			j_1		i_2		j_2

这时把整个表以控制关键字 43 为界分成[i_1，j_1]和[i_2，j_2]两部分，再分别对这两部分继续进行排序。先对[i_1，j_1]进行排序，取其控制关键字 K_1=14，排序过程为：

	[10	02]	14	39	43	[91	52	67]
第二遍结束→	02	10	14	39	43	[91	52	67]
						↑		↑
						i_2		j_2

再对[i_2，j_2]进行排序，取其控制关键字 K_1=91，排序过程为：

	02	10	14	39	43	[67	52]	91
排序完毕→	02	10	14	39	43	52	67	91

4.3 测量数据预处理技术

在计算机控制系统中，经常需要对生产过程的各种信号进行测量。测量时，一般先用传感器把生产过程的信号转换成电信号，然后用 A/D 转换器把模拟信号变成数字信号，读入计算机中。对于这样得到的数据，一般要进行一些预处理，其中最基本的为线性化处理、标度变换和系统误差的自动校准。

4.3.1 系统误差的自动校准

系统误差是指在相同条件下，经过多次测量，误差的数值（包括大小符号）保持恒定或按某种已知的规律变化。这种误差的特点是，在一定的测量条件下，其变化规律是可以掌握的，

产生误差的原因一般也是知道的。因此，原则上讲，系统误差是可以通过适当的技术途径来确定并加以校正的。在系统的测量输入通道中，一般均存在零点偏移和漂移，产生放大电路的增益误差及器件参数的不稳定等现象，它们会影响测量数据的准确性，这些误差都属于系统误差。有时必须对这些系统误差进行自动校准。其中偏移校准在实际中应用最多，并且常采用程序来实现，称为数字调零。调零电路如图 4-18 所示。

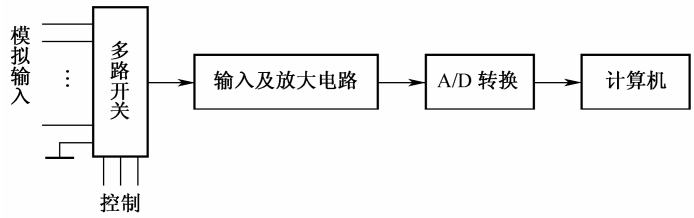


图 4-18　数字调零电路

在测量时，先把多路输入接到所需测量的一组输入电压上进行测量，测出这时的输入值为 x_1，然后把多路开关的输入接地，测出零输入时 A/D 转换器的输出为 x_0，用 x_1 减去 x_0 即为实际输入电压 x。采用这种方法，可以去掉输入电路、放大电路及 A/D 转换器本身的偏移及随时间和温度而发生的各种漂移的影响，从而大大降低对这些电路器件的偏移值的要求，简化硬件成本。

除了数字调零外，还可以采用偏移和增益误差的自动校准。自动校准的基本思想是在系统开机后或每隔一定时间自动测量基准参数，如数字电压表中的基准参数为基准电压和零电压，然后计算误差模型，获得并存储误差补偿因子。在正式测量时，根据测量结果和误差补偿因子计算校准方程，从而消除误差。自动校准技术有多种，下面介绍两种比较常用的方法。

1. 全自动校准

全自动校准由系统自动完成，不需要人的介入，其电路结构如图 4-19 所示。该电路的输入部分加有一个多路开关。系统在刚上电时或每隔一定时间自动进行一次校准。这时，先把开关接地，测出这时的输入值 x_0，然后把开关接 V_R，测出输入值 x_1，并存放 x_1、x_0，在正式测量时，如测出的输入值为 x，则这时的 V 可用下式计算得出：

$$V = \left(\frac{x - x_0}{x_1 - x_0} \right) \times V_R \tag{4-5}$$

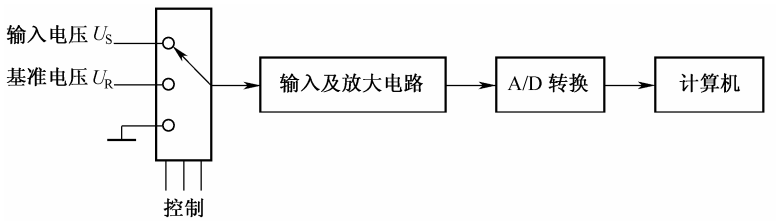


图 4-19　全自动校准电路

采用这种方法测得的 U 与放大器的漂移和增益变化无关，与 U_R 的精度也无关。这样可大大提高测量精度，降低对电路器件的要求。

2. 人工自动校准

全自动校准只适于基准参数是电信号的场合，并且它不能校正由传感器引入的误差。为了克服这种缺点，可以采用人工自动校准。

人工自动校准的原理与全自动校准差不多，只是现在不是自动定时进行校准，而是由人工在需要时接入标准的参数进行校准测量，把测得的数据存储起来，供后使用。一般人工自动校准只测一个标准输入信号 y_R，零信号的补偿由数字调零来完成。设数字调零后测出的数据分别为 x_R（接校准输入 y_R 时）和 x（接被测输入 y 时），则可按下式来计算 y：

$$y = \frac{y_R}{x_R} \times x \tag{4-6}$$

如果在校准时，计算并存放y_R/x_R的值，则测量校准时，只需要进行一次乘法即可。

有时校准输入信号 y_R 不容易得到，这时可采用现时的输入信号 y_i。校准时，计算机测出这时的对应输入 x_i，而人采用其他的高精度仪器测出这时的 y_i，并输入计算机中，然后计算机计算并存放 y_i/x_i 的值，代替前面的 y_R/x_R 来作校准系数。

人工自动校准特别适合于传感器特性随时间会发生变化的场合。

4.3.2 标度变换

控制系统在读入被测模拟信号并转换成数字量后，往往还要转换成操作人员所熟悉的工程值。这是因为生产现场的各种工艺参数量纲不同。例如，压力的单位为 Pa，流量的单位为 m^3/h，温度的单位为℃。这些参数 A/D 转换后得到一系列的数码，这些数码的值并不一定等于原来带有量纲的参数值，它仅代表参数值的大小，因此必须把它转换成带有量纲的数值后才能运算、显示或打印输出。这种转换称为工程量转换，也称为标度变换。标度变换有各种不同类型，它取决于被测参数测量传感器的类型，设计时应根据实际情况选择适当的标度变换方法。

1. 线性参数标度变换

线性参数标度变换是最常用的标度变换方法，其前提条件是被测参数值与 A/D 转换结果为线性关系。线性标度变换公式为：

$$A_x = (A_m - A_0)\frac{N_x - N_0}{N_m - N_0} + A_0 \tag{4-7}$$

式中，A_0 为一次测量仪表的下限，A_m 为一次测量仪表的上限，A_x 为实际测量值（工程量），N_0 为仪表下限所对应的数字量，N_m 为仪表上限所对应的数字量，N_x 为测量值所对应的数字量。

式（4-7）为线性标度变换的通用公式，其中 A_m、A_0、N_m、N_0 对于某一个固定的被测参数来说都是常数，不同的参数有着不同的值。为了使程序设计简单，一般把一次测量仪表的下限 A_0 所对应的 A/D 转换值置为 0，即 N_0=0。这样式（4-7）可以写成

$$A_x = (A_m - A_0)\frac{N_x}{N_m} + A_0 \tag{4-8}$$

在很多测量系统中，仪表下限值 A_0=0，此时对应的 N_0=0，式（4-8）可以进一步简化为

$$A_x = A_m \frac{N_x}{N_m} \tag{4-9}$$

式（4-7）至式（4-9）是在不同情况下的线性刻度仪表测量参数的标度变换公式。

所谓计算机标度变换程序，就是根据上述 3 个公式编写的计算程序。为此，可分别把 3 种情况设计成不同的子程序。设计时，可以采用定点运算，也可以采用浮点运算，根据需要进行选用。为编程方便，3 个公式可分别写成如下形式：

$$A_{x1} = a_1 N_x + b_1 \tag{4-10}$$

$$A_{x2} = a_2 N_x + A_0 \tag{4-11}$$

$$A_{x3} = a_3 N_x \tag{4-12}$$

其中，$a_1 = \dfrac{A_m - A_0}{N_m - N_0}$，$b_1 = A_0 - \dfrac{A_m - A_0}{N_m - N_0} N_0$，$a_2 = \dfrac{A_m - A_0}{N_m}$，$a_3 = \dfrac{A_m}{N_m}$。

根据式（4-10）至式（4-12）可以求出不同情况下被测参数的标度变换值。

2. 非线性参数标度变换

必须指出，前面讲的标度变换公式只适用于线性变化的参量。如果被测参量为非线性的，则上述的 3 个公式不再适用，需要重新建立标度变换公式。

一般而言，非线性参数的变化规律各不相同，故标度变换公式亦需要根据各自的具体情况建立。

（1）公式变换法。

例如，在流量测量中，流量与差压间的关系式为：

$$Q = K\sqrt{\Delta P} \tag{4-13}$$

式中，Q 为流量，K 为刻度系数，与流体的性质及节流装置的尺寸有关，ΔP 为节流装置前后的差压。

由此可见，流体的流量与被测流体流过节流装置前后生成的压力差的平方根成正比，于是得到测量流量时的标度变换公式为：

$$Q_x = (Q_m - Q_0)\sqrt{\frac{N_x - N_0}{N_m - N_0}} + Q_0 \tag{4-14}$$

式中，Q_x 为被测流体的流量值，Q_m 为流量仪表的上限值，Q_0 为流量仪表的下限值，N_x 为差压变送器所测得的差压值（数字量），N_m 为差压变送器上限所对应的数字量，N_0 为差压变送器下限所对应的数字量。

对于流量仪表，一般下限皆为 0，即 $Q_0=0$，所以式（4-14）可以简化为

$$Q_x = Q_m\sqrt{\frac{N_x - N_0}{N_m - N_0}} \tag{4-15}$$

若取流量表下限对应的数字量 $N_0=0$，更可进一步简化公式为

$$Q_x = Q_m\sqrt{\frac{N_x}{N_m}} \tag{4-16}$$

式（4-14）至式（4-16）即为不同初始条件下的流量标度变换公式。

与线性刻度变换公式一样，由于 Q_m、Q_0、N_m、N_0 都是常数，所以式（4-14）至式（4-16）

可分别记作

$$Q_{x1} = K_1\sqrt{N_x - N_0} + Q_0 \tag{4-17}$$

$$Q_{x2} = K_2\sqrt{N_x - N_0} \tag{4-18}$$

$$Q_{x3} = K_3\sqrt{N_x} \tag{4-19}$$

其中，$K_1 = \dfrac{Q_m - Q_0}{\sqrt{N_m - N_0}}$，$K_2 = \dfrac{Q_m}{\sqrt{N_m - N_0}}$，$K_3 = \dfrac{Q_m}{\sqrt{N_m}}$。

式（4-17）至式（4-19）即为各种不同条件下的流量标度变换公式，根据这些公式可以设计出各种条件下的流量标度变换程序。

（2）其他标度变换法。

许多非线性传感器并不像上面讲的流量传感器那样，可以写出一个简单的公式；或者虽然能够写出公式，但计算相当困难。这时可以采用多项式插值法，也可以用线性插值法或查表法进行标度变换。

4.3.3 线性化处理

在许多控制系统及智能化仪器中，一些参量往往是非线性参量，常常不便计算和处理，有时甚至很难找出明显的数学表达式，需要根据实际检测值或采用一些特殊的方法来确定其与自变量之间的函数关系式；在某些时候，即使有较明显的解析表达式，但计算起来也相当麻烦。而在实际测量和控制系统中，都允许有一定范围的误差。

例如，在温度测量中，热电阻及热电偶与温度之间的关系即为非线性关系，很难用一个简单的解析式来表达。在流量测量中，流量孔板的差压信号与流量之间也是非线性关系，即使能够用公式$Q = K\sqrt{\Delta P}$计算，但开方运算不但复杂，而且误差也比较大。另外，在一些精度及实时性要求比较高的仪表及测量系统中，传感器的分散性、温度的漂移、机械滞后等引起的误差在很大程度上都是不能允许的。诸如此类的问题，在模拟仪表及测量系统中，解决起来相当麻烦，有时甚至是不可能解决的。而采用计算机后，则可以用软件补偿的办法进行校正。这样，不仅能节省大量的硬件开支，而且精度也大为提高。

1. 线性插值算法

用计算机处理非线性函数应用最多的方法是线性插值法。线性插值法是代数插值法中最简单的形式。假设变量 y 和自变量 x 的关系如图 4-20 所示。

已知 y 在点 x_0 和 x_1 的对应值分别为 y_0 和 y_1，现在用直线 $\overline{AB}$ 代替弧线 $\overset{\frown}{AB}$，由此可得直线方程

$$g(x) = ax + b \tag{4-20}$$

根据插值条件，应满足

$$\begin{cases} y_0 = ax_0 + b \\ y_1 = ax_1 + b \end{cases} \tag{4-21}$$

解方程组（4-21），可以求出直线方程 g(x)的参数 a 和 b。由此可以求出该直线方程的表达式为

$$g(x)=\frac{(y_1-y_0)}{(x_1-x_0)}(x-x_0)+y_0 \\ =k(x-x_0)+y_0 \tag{4-22}$$

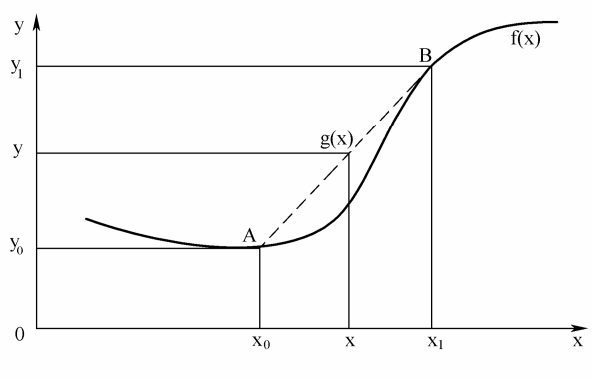


图 4-20　线性插值法示意图

或

$$g(x)=y_0\left(\frac{x_1-x}{x_1-x_0}\right)+y_1\left(\frac{x_0-x}{x_0-x1}\right) \tag{4-23}$$

其中，$k=\frac{(y_1-y_0)}{(x_1-x_0)}$，称为直线方程 g(x)的斜率。

式（4-22）为点斜式直线方程，式（4-23）为两点式直线方程。

由图 4-20 可以看出，插值点 x_0 和 x_1 之间的间距越小，那么在这一区间 g(x)和 f(x)之间的误差越小。因此，在实际应用中，为了提高精度，经常采用折线来代替曲线，此方法称为分段插值法。

2. 分段插值算法程序的设计方法

分段插值法的基本思想是将被逼近的函数（或测量结果）根据变化情况分成几段，为了提高精度及缩短运算时间，各段可根据精度要求采用不同的逼近公式。最常用的是线性插值和抛物线插值。在这种情况下，分段插值的分段点的选取可按实际曲线的情况灵活决定。关于抛物线插值算法，读者可参阅相关文献。

分段插值法程序设计的步骤如下：

（1）用实验法测量出传感器的输出变化曲线 y=f(x)（或各插值节点的值(x_i,y_i)，i=0，1，2，…，n）。为使测量结果更接近实际值，要反复进行测量，以便求出一个比较精确的输入输出曲线。

（2）将上述曲线进行分段，选取各插值基点。为了使基点的选取更合理，可根据不同的方法分段。主要有以下两种方法：

- 等距分段法。等距分段法即沿 x 轴等距离地选取插值基点。这种方法的主要优点是使 $x_{i+1}-x_i$ 为常数，从而简化计算过程。但是，当函数的曲率或斜率变化比较大时将会产生一定的误差。要想减小误差，必须把基点分得很细，但这样势必占用更多的内存，并使计算机的开销加大。
- 非等距分段法。这种方法的特点是函数基点的分段不是等距的，而是根据函数曲线形

状的变化率的大小来修正插值点间的距离。曲率变化大的部位，插值距离取小一点。也可以使常用刻度范围插值距离取小一点，而在曲线较平缓和非常用刻度区域距离取大一点，但是非等距插值点选取比较麻烦。

（3）根据各插值基点的(x_i-y_i)值，使用相应的插值公式，求出模拟 y=f(x)的近似表达式 $P_n(x)$。

（4）根据 $P_n(x)$编写出应用程序。

用式（4-22）进行计算比较简便，只需要进行一次减法、一次乘法和一次加法运算即可。

在用分段法进行程序设计之前，必须首先判断输入值 x_i 处于哪一段。为此，需要将 x_i 与各分点值进行比较，以确定出该点所在的区间。然后，转到相应段逼近公式进行计算。

值得说明的是，分段插值法总的来讲光滑度都不太高，这对于某些应用是有缺陷的。但是，就大多数工程要求而言，也能基本满足需要。在这种局部化的方法中，要提高光滑度，就必采用更高阶的导数值，多项式的次数也需要相应增高。

4.3.4 越限报警处理

为了实现安全生产，在计算机控制系统中，对于重要的参数和系统部位，都要设置紧急状态报警系统，越限报警处理就是其中的一种。由采样读入的数据或经计算机处理后的数据是否超过工艺参数的范围计算机要加以判别，如果超越了规定数值，则需要通知操作人员采取相应的措施，确保生产的安全。在控制系统中常用的报警方式是声、光及语言报警，常用的最简单的报警程序是越限报警。越限报警分为上限报警、下限报警、上下限报警。

报警处理程序一般都需要根据系统的要求编写，虽然不同系统的报警处理程序是不一样的，但报警程序设计的基本思想是相同的。报警程序主要由以下几个步骤组成：

（1）采样被测参数。

（2）比较采样值和给定的上下限。

（3）根据比较结果执行相应的处理程序。

如果需要报警的参数是 X_n，该参数的上下限值分别是 X_{max} 和 X_{min}，则上下限报警的规则定义如下：

（1）上限报警：若 $X_n>X_{max}$，则上限报警，否则继续执行原定操作。

（2）下限报警：若 $X_n<X_{min}$，则下限报警，否则继续执行原定操作。

（3）上下限报警：若 $X_n>X_{max}$，则上限报警，否则判断 $X_n<X_{min}$ 否？若是，则下限报警，否则继续原定操作。