Contents

Chapter 1

前言

今天，计算机科学一词是一个非常广泛的概念。尽管如此，在本书里，我们将其定义为“和计算机有关的问题”。本章首先阐述什么是计算机，接着探索和计算机直接相关的一些问题。首先我们将计算机看成一个图灵模型，这是从数学上和哲学上对计算机的定义。然后，阐述当今的计算机是如何建立在冯·诺伊曼模型基础上。最后介绍计算机这一改变文明的装置的简明历史。

Alan Turing 在 1937 年首次提出了一个通用计算设备的设想。他设想所有的计算都能在一个中特殊的机器上执行，这就是现在所说的图灵机。尽管图灵对这样一种机器进行了数学上的描述，但他还是更有兴趣关注计算的哲学定义，而不是建造一台真是的机器。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 i 抽象到用于计算的机器的模型中，这才真正改变了世界。

在讨论图灵模型之前，让我们把计算机定义成一个数据处理器。依照这种定义，计算机就可以被看作是一个接受输入数据、处理数据并产生输出数据的黑盒。尽管这个模型能够体现现代计算机的功能，但是他的定义还是太宽泛。按照这种定义，也可以认为便携式计算器是计算机。

另一个问题是这个模型并没有说明它处理的类型以及是否可以处理一种以上的类型。换句话说，它并没有清楚地说明基于这个模型的机器能够完成操作的类型和数量。它是专用机器还是通用机器呢？

这种模型可以表示为一种设计用来完成特定人物的专用计算机，比如用来控制建筑物温度或汽车油料使用。尽管如此，计算机作为一个当今使用的术语，是一种通用的机器，它可以完成各种不同的工作。这表明我们需要该模型改变为图灵模型来反映当今计算机的实现。

图灵模型是一个适用与通用计算机的更好模型。该模型添加了一个额外的元素——程序到不同的计算机器中。程序是用来告诉计算机对数据进行处理的指令集合。

在这个图灵模型中，输入数据是依赖来两个方面因素的结合作用：输入数据和程序。对于相同的数据输入，如果改变程序，则可以产生不同的输出。类此地，对于同样的程序，如果改变输入数据，其输出结果也将不同。最后，如果输入数据和程序保持不变，输出结果也将不变。让我们看看下面三个示例。

通用图灵集是对现代计算机的首次描述，该机器只要提供了和是的程序就能做任何运算。可以证明，一台很强大的计算机和通用图灵机一样能进行同样的运算。我们所需要的仅仅是为这两者提供数据以及用于描述如何做运算的程序。

实际上。通用图灵机能做任何可计算的运算。

基于通用图灵机建造的计算机都是在存储器中存储数据。在 1944～1945 年期间，冯·诺依曼指出，鉴于程序和数据在逻辑上是相同的，因此程序也能存储在计算机的存储器中。

基于冯·诺依曼模型建造的计算机分为 4 个子系统：存储器、算术逻辑单元、控制单元和输入/输出单元。

存储器是用来存储的区域，在计算机的处理过程中存储器用来存储数据和程序，我们将在这一章后边讨论存储数据和程序的原因。

算术逻辑单元是用来进行计算和逻辑运算的地方。如果是一台数据处理计算机，它应该能够对数据进行算术运算。当然它也应该可以对数据进行一系列逻辑运算，正如我们将在第 4 章看到的那样。

控制单元是对存储器、算术逻辑单元、输入/输出等子系统进行控制的单元。

输入子系统负责从计算机外部接收输入数据和程序；输入子系统负责将计算机的处理结果输出到计算机外部。输入/输出子系统的定义相当广泛，它们还包含辅助存储设备，例如，用来存储处理所需的程序和数据的磁盘和磁带等。当一个磁盘用于存储处理后的输出结果，我们一般就可以认为它是输出设备，如果从该磁盘上读取数据，则该磁盘就被认为是输入设备。

冯·诺依曼模型中要求程序必须存储在内存中。这和早期只有数据才存储在存储器中的计算机结构完全不同。完成某一任务的程序是通过操作一系列的开关或改变其配线来实现。

现代计算机的存储单元用来存储程序及其响应数据。这意味着数据和程序应该具有相同的格式，这是因为他们都存储在存储器中。实际上它们都是以位模式存储在内存中的。

冯·诺依曼模型中的一段程序是由一组有限的指令组成。按照这个模型，控制单元从内存中提取一条指令，解释指令，接着执行指令。换句话说，指令就一条接着一条地顺序执行。当然，一条指令可能会请求控制单元以便跳转到其前面或后面的指令去执行，但是这并不意味着指令没有按照顺序来执行。指令的顺序执行是基于冯·诺依曼模型的计算机初始条件。当今的计算机以最高的顺序来执行程序。

我们可以认为计算机由三大部分组成：计算机硬件、数据和计算机软件。

冯·诺依曼模型清楚地将一台计算机定义为一台数据处理机。它接受输入数据，处理并输出相应的结果。

冯·诺依曼模型并没有定义数据如何存储在计算机中。如果一台计算机是一台电子设备，最好的数据存储方式应该是电子信号，例如以电子信号的出现和消失的的定方式来存储数据，这意味着一台计算机可以以两种状态之一的形式存储数据。

显然，在日常使用的数据并不是以两种状态之一的形式存在，例如，我们数字系统中使用的数字可以是 0～9 十种状态中的任何一个。但是你不能将这类信息存储到计算机内部，除非将这类信息转换成另一种只使用两种状态的系统。同样，你也需要处理其他类型的数据，它们同样也不能直接存储到计算机中，除非将它们转变成合适的形式。

尽管数据只能以一种形式存储在计算机内部，但在计算机外部却可以表现为不同的形式。另外，计算机开创了一门新兴的研究领域——数据组织。在将数据存储到计算机之前，能否有效地将数据组织成不同的实体和格式？如今，数据并不是按照杂乱无章的次序来组织信息的。数据被组织成许多小的单元，再由这些小的单元组成更大的单元，等等。

图灵或冯·诺依曼模型的主要特征是程序的概念。尽管早期的计算机并没有计算机存储器中存储程序，但它们还是使用了程序的概念。编程在早期的计算机中体现为系列开关的打开或关闭以及配线的改变。编程在数据世纪开始处理之前是由操作员或工程师完成的一项工作。

在冯·诺依曼模型中，这些程序被存储在计算机的存储器中，存储器中不仅要存储数据，还要存储程序。

这个模型还要求程序必须是有序的指令集。每一条指令操作一个或者多个数据项。因此，一条指令可以改变它前面指令的作用。

也许我们会问为什么程序必须由不同的指令集组成，答案是重用性。如今，计算机完成成千上万的任务，如果每一项任务的程序都是相对独立而且和其他程序之间没有任何公用段，编程将会变成意见很困难的事情。图灵模型和冯·诺依曼模型通过仔细地定义计算机可以使用的不同指令集，从而使得编程变得相对简单。程序员通过组合这些不同的指令创建任意数量的程序。每个程序可以是不同指令的不同组合。

要求程序包含一系列指令是的编程变得可能，但也带来了另外一些使用计算机方面的问题。程序员不仅要了解每条指令所完成的任务，还要知道怎样将这些指令结合起来完成一些特定的任务。对于一些不同的问题，程序员首先应该以循序渐进的方式来解决问题，接着尽量找到合适的指令来解决问题。这种按照步骤解决问题的方法就是所谓的算法。算法在计算机科学中起到了重要的作用。

在冯·诺依曼模型中没有定义软件工程，软件工程是指结构化程序的设计和编写。今天，它不仅仅是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。

在计算机发展的演变过程中，科学家们发现有一系列指令对所有程序来说是公用的。如果这些指令只编写一次就可以用于所有程序，那么效率将会大大提高。这样，就出现了操作系统。计算机操作系统最初是为了程序访问计算机部件提供方便的一种管理程序。今天，操作系统所完成的工作远不止这些。

Chapter 2

计算机组成

本章我们将讨论计算机的组成。讲解计算机是如何由三个子系统组成的。我们还介绍了简单假想的计算机，它能运行简单程序，完成基本的算术或逻辑运算。

计算机的组成部件可以分为三大类：中央处理单元、主存储器和输入/输出子系统。接下来的三个部分将讨论这些子系统以及如何将这些子系统组成一台计算机。

中央处理器用于数据的运算。在大多数体系结构中，它有三个组成部分：算术逻辑单元、控制单元、寄存器组、快速存储定位。

算术逻辑单元对数据进行逻辑、位移和算术运算

在第 4 章中，我们讨论了几种逻辑运算，如：非、与、或和抑或。这些运算大输入数据作为二进制位模式，运算的结果也是二进制模式。

在第 4 张中，我们讨论了数据的两种位移运算：逻辑移位和算术移位运算。逻辑移位运算用来对二进制位模式进行向左或右的移位，而算术运算被应用于整数。它们的主要用途是用 2 除或乘一个整数。

寄存器是用来存放临时数据的高速独立的存储单元。CPU 的运算离不开大量寄存器的使用。

在过去，计算机只有几个数据寄存器用来存储输入数据和运算结果。现在，由于越来越多的复杂运算改由硬件设备实现，所以计算机在 CPU 中使用几十个寄存器来提高运算速度，并且需要一些寄存器来保存这些运算的中间结果。

现在，计算机存储的不仅是数据，还有存储在内存中相对应的程序。CPU 的主要职责是：从内存中逐条取出指令，并将取出的指令存储在指令寄存器中，解释并执行指令。

CPU 中另一个通用寄存器是程序计数器。程序计数器中保存这当前正在执行的指令。当前的指令执行完后，计数器敬将自动加 1，指向下一条指令的内存地址。

CPU 的第三个部分是控制单元，控制单元控制各个子系统的操作。控制是通过从控制单元到其他子系统的信号进行的。

主存储器是计算机内的第二个子系统。它是存储单元的集合，每一个存储单元都有唯一的标识，称为地址。数据以成为字的位组形式在内存中传入和传出。字可以是 8 位、16 位、32 位，甚至有时是 64 位，如果字是 8 位，一般称为 1

字节。术语字节在计算机科学中使用相当普遍，因此有时成为 16 位位 2 字节，32 位位 4 字节。

在存储器中存取每个字都需要有相应的标识符。尽管程序员使用命名的方式来区分，但在硬件层次上，每个子都是通过地址来标识的。所以在内存中表示的独立的地址单元的总数称为地址空间。

随机存取存储器是计算机中主存的主要组成部分。在随即存取设备中，可以使用存储单元地址来随机存取一个数据项，而不需要存取位于它前面的所有数据项。该术语有时因 ROM 也能随机存取而与 ROM 混淆，

计算机用户需要许多存储器，尤其是速度快而且价格低廉的存储器。但这种要求并不总能得到满足。存取数度快的存储器通常都不便宜。因此需要寻找一种折中的方法。解决的办法是采用存储器的层次结构。

Chapter 3

计算机网络和因特网

个人计算机的发展带动了商业、工业、科学和教育的巨大改变。网络也发生了类型的变革。技术的进步使得通信线路能传送更多、更快的信号。而不断发展的服务使得我们能够使用这些扩展的能力。计算机网络的研究导致了新技术的产生——在全球各个地方交换文本、音频和视频等数据，在任何时候快速、准确地下载或上载信息。

虽然本章的目标是讨论因特网，一个将世界上几十亿台计算机互相连接的系统，我们对互联因特网的认识不应该是一个单独的网络，而是一个网络结合体，一个互联网络。因此，我们的旅程将从定义一个网络开始。然后我们将要展示如何通过网络连接来建造小型的互联网络。最终我们会展示因特网的结构并且在本章的以下部分开启研究因特网的大门。

网络是一系列可用于通信的设备相互连接构成的。在这个定义里面，一个设备可以是一台主机，比如一台大型计算机、台式机、便携式计算机、工作站、手机或安全系统。在这种定义中，设备也可以是一个连接设备，比如用来将一个网络与另一个网络相连接的路由器，一个将不同设备连接在一起的交换机，或者一个用于改变数据形式的调制器，等等。在一个网络中，这些设备都通过有线或无线传输媒介互相连接。当我们在家通过即插即用路由器连接两台计算机时，虽然规模很小，但已经建造了一个网络。

局域网通常是与单个办公室、建筑或校园内的几个主机相连的私有网络。基于机构的需求，一个局域网既可以简单到某人家庭办公室中的两台个人计算机和一台打印机，也可以扩大至一个公司范围，并包括音频和视频设备。在一个局域网中的每一台主机都作为这台主机在局域网中唯一定义的一个标识和一个地址。一台主机向另一台主机发送的数据包中包括源主机和目标主机的地址。

广域网也是通信设备互相连接构成的。但是广域网与局域网之间有一些差别。局域网的大小通常是受限的，跨越一个办公室、一座大楼或一个校园；而广域网的地理跨度更大，可以跨越一个城镇、一个州、一个国家，甚至横跨世界。局域网将主机互联，广域网则将交换机、路由器或调制解调器之类的连接设备互连。通常，局域网为机构私有，广域网则由通信公司创建并运营，并且租给使用它的机构。我们可以看到广域网的两种截然不同的案例：点对点广域网和交换广域网。

点对点广域网是通过传输媒介连接两个通信设备的网络。

交换广域网是一个有至少两个端的网络。就像我们很快就会看到的那样，交

换广域网用于今天全球通信的骨干网。我们也可以这么说，交换广域网是一个点对点广域网通过开关连接产生的结合体。

现在很难看见独立存在的局域网或广域网，它们现在都是互相连接的。当两个或多个网络相互连接时，它们构成一个互联网络，或者说网际网。

正如我们之前讨论过的，一个网际网是来嗯个或多个可以互相通信的网络。最值得注意的网际网是因特网，它由成千上万个互连的网络组成。

我们已经展示了因特网通过连接设备将大大小小的网络相互交织在一起构成的基本结构。然而，如果仅仅将这些部分连接在一起，很明显什么都不会发生。为了产生沟通，即需要硬件也需要软件设备。这就像当进行一个复杂的计算时，我们同时需要计算机和程序。

当讨论因特网时，有一个词我们总是会听见，这个词就是协议。协议定义了发送器、接收器以及所有中间设备必须遵守以保证有效地通信的规则。简单的通信可能只需要一条简单的协议，当通信便得复杂时，可能需要将任务分配到不同的协议层中，在这种情况写，我们在每一个协议层都需要一个协议，或者协议分层。

为了更好地理解协议分层的必要，首先我们开发一个简单的场景。假定 Ann 和 Maria 是有很多共同想法的邻居，她们每次都会为了一个何时退休的计划互相见面和进行沟通。突然，Ann 所在的公司为她提供一个升职的机会，但是同时要求她搬到离 Maria 很远的一个城市中的分部去住。由于她们想出了一个具有创新型的计划——在退休后开始做新的生意，这两个朋友仍然向继续它们的通信并且就这个新计划交换想法。她们决定通过到邮局使用普通邮件通信来继续她们的对话，但是如果邮件被拦截了，她们不想别人知晓它们的想法。...

Chapter 4

操作系统

计算机系统是由两个主题部分组成：硬件和软件。硬件是计算机的物理设备。软件则是使得硬件能国正常工作的程序的集合。计算机软件分为两大类：操作系统和应用程序。应用程序使用计算机硬件来解决用户的问题。另一方面，操作系统则控制计算机用户对硬件的访问。

操作系统是一个非常复杂的系统，因此很难给予它一个普遍认同的简单定义。在这里列举一些常见的定义：

操作系统是介于计算机硬件和用户之间的接口。

操作系统时一种用来使得其他程序更加方便有效运行的程序。

操作系统作为通用管理程序管理着计算机系统中每个部件的活动，并确保计算机系统中的硬件和软件资源能够更加有效地使用。当资源使用出现冲突时，操作系统应能国及时处理，排除冲突。

操作系统时计算机硬件和用户的一个接口，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。

基于上面的定义，操作系统位其他程序提供支持。例如，它负责把其他程序装入内存以便运行。但是，操作系统本身也是程序，它需要被装入内存和运行，这个困难如何解决呢？如果使用 ROM 技术把操作系统存储在内存中，这个问题就你能解决。CPU 的程序计数器可以被设置位这个 ROM 的开始处。当计算机被 i 加电时，CPU 从 ROM 中读取指令，执行它们。但这种解决方案是非常低效的，因为内存的很大一部分需要由 ROM 构成，而不能被其他程序使用。如今的技术是仅需要分配小部分的内存给部分操作系统。

如今使用的解决方案采用两个阶段过程。很小一部分内存用 ROM 构成，其中存有成为自举程序的小程序。当计算机被加电时，CPU 计数器被设置位自举程序的第一条指令，并执行程序中的指令，操作系统就被执行。

操作系统已经经历了很长的一段发展历程，我们将在下面加以总结。

批处理操作系统设计于 20 世纪 60 年代，目的是控制大型计算机。当时计算机十分庞大。用穿孔卡片进行输入数据，用行式打印机输出结果，用磁带设备作为辅助存储介质。

每个运行的程序叫一个作业。想要运行程序的程序员通过穿孔卡片将程序和数据输入计算机，并向控制器发出作业请求。穿孔卡片由操作员处理。如果程序运行成功，打印结果将传给程序员，如果不成功，则报错。

这个时代的操作系统非常简单：它们只保证计算机所有资源被从一个作业转换到另一个作业。

为了有效的使用计算机资源，多道程序的概念被引进。它可以将多个作业同时装进内存，并且当且仅当资源可用时分配给需要它的作业。例如，当一个程序正使用输入/输出设备时，CPU 则处于空闲状态，并可以供其他程序使用。我们将在本章后面详细介绍多道操作系统。

多道程序带类了分时的概念：资源可以被不同的作业分享。每个作业可以分到一段时间来使用资源。因为计算机运行速度很快，所以分时系统对于用户是隐藏的，每个用户都感觉整个系统在为自己服务。

最终利用分时技术的多道程序极大地改进了计算机的使用效率。但是，它们需要一个更复杂的操作系统，它必须可以调度：给不同的程序分配资源并决定哪一个程序什么时候使用哪一个资源。在这个时代中用户和计算机的关系也改变了。用户可以直接与操作系统进行交互而不必通过操作员。一个新的术语也随之产生：进程。一个作业是一个要运行的程序，一个进程则是在内存中等待分配资源的程序。

个人计算机产生后，需要有一类适合这类计算机的操作系统。于是，单用户操作系统就应运动而生了。

人们对更快和更有效的需求导致了并行系统的设计：在统一计算机中安装多个 CPU，每个 CPU 可以处理一个程序或一个程序的一部分。意味着很多任务可以并行地处理而不是在串行处理。当然这种操作系统要比单 CPU 的操作系统复杂得多。

网络化和网络互联的发展，扩大了操作系统的内涵。一个以往必须在一台计算机上运行的作业现在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，只要它们通过网络连接即可。资源可以是分布式的，一个程序需要的文件可能分布在世界的不同地方。分布式系统结合了以往系统的特点和新的功能，例如安全控制。

实时系统是指在特定时间限制内完成任务。它们被用在实时应用程序中，这些应用程序监控、响应或控制外部过程或环境。在交通控制、病人监控或军事控制系统中可以找到实时系统的例子。

现在的操作系统十分复杂，它必须可以管理系统中的不同资源。它像是一个有多个上层部门经理的管理机构，每个部门经理负责自己的部门管理，并且互相协调。现代操作系统至少具有以下 4 中功能：存储管理、进程管理、设备管理、文件管理。就像很多组织由一个部门不归任何经理管理一样，操作系统也有这样一个部分，被称为用户界面或命令解释程序，它负责操作系统与外界通信。

每个操作系统都有用户界面，即指用来接收用户的输入并向操作系统解释这些请求的程序。一些操作系统的用户界面，被称为命令解释程序。在其他操作系统中，则被称为窗口，以指明它是一个由菜单驱动的并有着 GUI 的部件。

喜爱年在计算机操作系统的一个重要职责是内存管理。计算机中存储器的容量近年来得到激增，同样所处理的程序和数据也越来越大。内存分配必须进行管理一面“内存溢出”的错误。操作系统按照存储管理可以分为两大类：单道程序和多道程序。

单道程序属于过去，但它还是值得学习，因为它有助于理解多道程序。在单道程序中，大多数内存用来转载单一的程序，仅仅一小部分用来装载操作系统。在这种配置下，整个程序装入内存运行，运行结束后，程序区域由其他程序取代。

这里内存管理器的工作是简单明了的，即将程序载入内存、运行它、再装入新程序。但是，在技术方面仍然有很多问题：

程序必须能够载入内存。如果内存容量比程序小，程序就无法运行。

当一个程序正在运行时，其他程序不能运行。一个程序在执行过程中经常需要从输入设备得到数据，并且把数据发送至输出设备。但输入/输出设备的速度远远小于 CPU，所以当输入/输出设备运行时，CPU 处于空闲状态。而此时由于其他程序不再内存中，CPU 不能为其服务。这种情况下 CPU 和内存的使用效率很低。

在多道程序下，同一时刻可以装载多个程序并且能够同时被执行。CPU 轮流为其服务。

从 20 世纪 60 年代开始，多道程序已经经过了一系列的改进。

有两种技术属于非交换范畴，这意味着程序在运行期间始终驻留在内存中。另外两种技术属于交换范畴。也就是说，在运行过程中，程序可以在内存和硬盘之间多次交换数据。

多道程序的第一种技术称为分区调度。在这种模式中，内存被分为不定长的几个分区。那个部分或分区保存一个程序。CPU 在各个程序之间交替服务。它由一个程序开始，执行一些指令，直到有输入/输出操作或者分配给程序的时限到达为止。CPU 保存最近使用的指令所分配的内存地址后转入下一个程序。对下一个程序采用同样的步骤反复执行下去。当所有程序服务完毕后，再转回第一个程序。当然，CPU 可以进行优先级管理，用于控制分配给每个 CPU 时间。

在这种技术下，每个程序完全载入内存，并占用连续的地址。分区调度改进了 CPU 的使用效率，但仍然有以下一些问题：

分区的大小必须由内存管理器预先决定。如果分区小了，由的程序就不能载入内存。如果分区大了，就会出现空闲区。

即使分区在刚开始时比较合适，但随着新程序的交换载入内存后有可能出现空闲区。

当空闲区过多时，内存管理器能够紧缩分区并删除空闲区和创建新区，但这将会增加系统额外开销。

分页调度提高了分区调度的效率。在分页调度下，内存被分成大小相等的若干部分，称为帧。程序被分为大小相等的部分，称为页。页和帧的大小通常时一样的，并且与系统用于从存储设备中提取信息的块的大小相同。

页被载入内存中的帧。如果一个程序有 3 页，它就在内存中占用 3 个帧。在这种技术下，程序在内存中不必时连续的：来嗯个连续的页可以占哟哦嗯内存中不连续的两个帧。分页调度对分区调度的优势在于，一个需要 6 个帧的程序可以代替两个各占有不连续的 3 个帧的程序。而不必等到有 6 个连续的帧出现后再载入内存。

分页调度在一定程度上提高了效率，但整个程序仍需要在运行前全部载入内存。这意味着只有 4 个不连续帧时，一个需要 6 个空闲帧的程序不能载入。

分页调度不需要程序转载在连续的内存中，但仍需要程序载入内存中运行。请求分页调度该表了后一种限制。在请求分页调度中，程序被分为页，但是页可以依次载入内存、运行，然后被另一个页替代。换句话说，内存可以同时载入多个程序的页。此外，来自同一个程序的连续页可以不必载入同一个帧，一个页可以载入任何空闲帧。

类此于分页调度的技术时分段调度。在分页调度中，不像程序员以模块来考虑程序，程序实际是分为大小相同的页。你将在后面的章节中看到，程序通称

由主程序和子程序组成，在请求分段调度中，程序将按照程序员的角度划分成段，它们载入内存中、执行，然后被来自统一程序或其他程序的模块代替。

请求分页和分段调度结合了两者的有点以提高系统效率。一个段也许太大而不能载入内存中的空闲区。内存可以分成很多帧，一个模块可以份额成很多页，一次装入内存运行。

请求分页调度和请求分段调度移位这当程序运行时，一部分程序驻留在内存中个一部分则放在硬盘上。这就意味着...。虚拟内存。

操作系统的第二个功能是进程管理，在介绍该概念之前，我们先定义一些术语。

现代操作系统关于指令集有三个术语：程序、作业和进程。尽管这些术语比较模糊，并且不同操作系统对于它们的定义不一致，我们还是可以作出非正式的定义。

程序是由程序员编写的一组稳定的指令，存在磁盘上，它可能会也肯能不会成为作业。

从一个程序被选中执行，到其运行结束并再次成为一个程序的这段过程中，该程序称为作业。在整个过程中，作业可能会或不会被执行，或者驻留在磁盘上等待调入内存，或者在内存中等待 CPU 执行，或者驻留在硬盘或内存中等待一个输入/输出事件，或者在内存中等待知道被 CPU 运行。在所有这些情况下程序才称为作业。当一个作业执行完毕，它又变成程序代码并再次留在硬盘中，操作系统不再支配该程序。

进程是一个运行中的程序。该程序开始运行但还未结束。换句话说，进程是一个驻留在内存中运行的作业，它是从众多等待作业中选取出来并转入内存中的作业。一个进程可以处于运行或者的等待 CPU 调用。只要作业装入内存就称为一个进程。

将一个作业或者进程从一个状态改变为另一个状态，进程管理器使用了两个调度器：作业调度器和进程调度器。

作业调度器将一个作业从保持状态转入就绪状态，或是从运行状态转入终止状态。换句话说，作业调度其负责从作业中创建一个进程或终止一个进程。

进程调度器将一个进程从一个状态转入另一个状态。当一个进程等待某事件发生时，它使这一进程从运行状态进入等待状态。当事件发生时，进程将从等待状态进入就绪状态。

设备管理器负责访问输入/输出设备。在计算机系统中输入/输出设备存在这数量和速度的限制。由于这些设备与 CPU 和内存比起来速度要慢很多，多以当一个进程访问输入/输出设备时，在该段是建安内这些设备对其他进程而言是不可用的。设备管理器负责让输入/输出设备使用起来更有效。

对设备管理器的细节的讨论需要掌握有关操作系统原理的高级知识，这些都不在本书讨论之列。但是我们可以在这里简要地列出设备管理器的功能。

现今的操作系统使用文件管理器来控制对文件的访问。对文件管理器细节的讨论同样需要掌握有关操作系统原理和文件访问的高度概念，这些超出了本出的讨论范围。我们将在地 13 章讨论一些有关文件访问的问题，但是对于了解文件管理器实际的操作还是不够。下面简述一下文件管理器的功能：

文件管理器控制文件的访问。只有那些获得允许的应用程序才能够访问，访问方式也不同。例如，一个进程也许可以读取文件，但却不允许写操作。另一个进程也许被允许执行文件和进程，但却不允许读取文件的内容。

文件管理器管理文件的创建、删除和修改。

文件管理器可以给文件命名。

文件管理器管理文件的存储：怎样存储，存在哪里等。

文件管理器负责归档和备份。

在这一节，我们将介绍一些常用的操作系统，以促进将来的学习。我们选择三种计算机用户熟悉的操作系统：UNIX、Linux 和 Windows。

UNIX 是由贝尔实验室的计算机科学研究小组的 Thomson 和 Ritchie 在 1969 年首先开发出来的。从那时起，UNIX 经历了许多版本。它是一个在程序员和计算机科学家中较为流行的操作系统。它可以不经较大的改动而方便地从一个平台移植到另一个平台。原因是它主要是由 C 语言编写的。第二，UNIX 拥有一套功能强大的工具，它们能够组合起来去解决许多问题，而这一工作在其他操作系统则需要通过编程来完成。第三，它具有设备无关性，因为操作系统本身就包含了设备驱动程序，这意味着它可以方便地配置来运行任何设备。

UNIX 是多用户、多道程序、可移植的操作系统，它被设计来方便编程、文本处理、通信和其他许多希望操作系统来完成的任务。它包含几百个简单、单一目的的函数，这些函数能组合起来完成任何可以想象的处理任务。它的灵活性通过它可以在三种不同的计算机环境中而得到证明，这三种环境为：单机个人环境、分时系统和客户、服务器系统。

UNIX 由 4 个主要部分构成：内核、命令解释器、一组标准工具和应用程序。这些组成部分显示在图中。

内核是 UNIX 系统的心脏。它包含操作系统最基本的部分：内存管理、进程管理、设备管理和文件管理。系统所有其他部分均调用内核来执行这些服务。

命令解释器是 UNIX 中用户最可见的部分。它接收和解释用户输入的命令。在许多方面，这使它成为 UNIX 结构的最重要的组成部分。它肯定也是用户最知道的部分。为了在系统做任何事情，我们必须向命令解释器输入命令。如果命令需要一个工具，命令解释器将请求内核执行该工具。如果命令需要一个应用程序，命令解释器需要内核运行它。有些操作系统有几种不同的命令解释器。

UNIX 中有几百个工具。工具是 UNIX 标准程序，它为用户提供支撑过程。常用的三个工具是：文本编辑器、搜索程序和排序程序。

许多系统工具实际上复杂的应用程序。例如，UNIX 的电子邮件系统被看成一个工具，就像三种常见的文本编辑器：vi、emacs 和 pico。所有这 4 个工具本身都是很大的系统。其他工具是简短函数。例如，list 工具显示磁盘目录中的文件。

UNIX 的应用是指一些程序，它们不是操作系统发布中的标准部分。它们是由系统管理员、专职程序员或用户编写的，提供了对系统的扩展能力。事实上，许多标准工具自多年前都是作为应用出现的，后来被证明非常有用，现在就成了系统的一部分。

在 1991 年，芬兰 Helsinki 大学的学生 Linux Torvalds 开发了一个新的操作系统，这就是如今所知的 Linux。初始内核如今成长为全面的操作系统。1997 年发布的 Linus2.0 内核成为商业操作系统，它具有传统 UNIX 的所有特性。

内核负责处理所有属于内核的职责。

系统库含有一组被应用程序使用的函数，用于与内核交互。

系统工具是使用系统库提供的服务，执行管理任务的各个程序。

Linux 的安全机制提供了传统上为 UNIX 定义的安全特性。如身份验证和访问控制。

Chapter 5

算法

算法一种非正式定义如下：算法是一种逐步解决问题或完成任务的方法。

按照这种定义，算法完全独立于计算机体系。更特别的是，还应该记住算法接受一组输入数据，同时产生一组输出数据。

下面用一个例子来对这种简单的定义进行分析。我们要生成从一组正整数中找到最大整数的一个算法。这个算法应该能从一组任意整数中找出其最大值。这个算法必须具有通用性并与整数的个数无关。

很明显，要完成从许多整数中找到最大值的这个任务不可可能只要一步完成。算法必须一个个地测试每一个整数。

要解决这个问题，可以用一种直接的方法。先用一组少量的整数，然后将这种解决方法扩大到任意多的整数。其实对 5 个整数所采取的解决方法的原理和约束条件与对 1 000 个或 1 000 000 个整数采取的是一样的。可以假设，即使是 5 个整数的例子，算法也必须一个接一个地处理那些整数。看到第一个整数，并不知道剩下的整数的值。等处理完第一个整数，算法才开始处理第二个整数，依次进行。

我们称这个算法为求最大值算法。每个算法都有自己不同于其他算法的名字。这个算法接收一组 5 个整数作为输入，然后输出其中的最大值。

细化。为了使算法能在所有的程序中应用，还需要进行细化。现在有两个问题，首先，第一步中的动作与其他步骤中不一样。其次第二步到第五步中的程序描述语言不通。我们只要很简单地改进一下算法就可以解决以上两个问题。把第二步到第五步的程序段都写成“如果当前整数大于 Largest，那么当前整数就称为 Largest”。第一步不通于其他步是因为那时 Largest 还没有初始化。如果开始就把 Largest 初始化为负无穷，那么第一步就可写成和其他步一样，所以。增加一个新的步骤，可称为第 0 步，也就是表明它要在处理任何其他整数之前完成。

可以把这个算法泛化吗？假设要从 n 个正整数中找到最大值，n 的值可能是 1 000 或 1 000 000，或者更多。当然，可以按照图所示那样重复每一步。

计算机专家为结构化程序或算法定义了三种结构。这种想法认为程序必定是由顺序、判断（选择）和循环这三种结构组成。已经证实其他结构都是不必要的。仅仅使用这三种结构就可以是程序或算法容易理解、调试或修改。

第一种结构称为顺序结构。算法都是指令序列。它可以是一简单指令或其他两种结构之一。

有些问题只用简单的指令序列是不能够解决的。有时候需要检测一个条件是否满足。假如测试的结果为真，则可以继续顺序往下执行指令；假如结果为假，程序将从另外一个顺序结构的指令继续执行。这就是所谓的判断（选择）结构。

有些问题中，相同指令序列需要重复。可以用重复或循环结构来解决这个问题。

到目前为止，我们已经使用图来表示算法的基本概念。在最近几十年中，还出现了其他几种用来表示算法的工具。这里讲介绍 UML 和伪代码这两种工具。

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，它只显示算法从开始到结束的整个流程。

伪代码是算法的一种类似英语的表示法。现在还没有伪代码的标准。有些人使用得过细，有些人则使用得过粗。有些人用一种很像英语的代码，有些则用和 Pascal 编程语言相似的语法。

既然我们讨论了算法的概念并且给出了它的表示，下面给出算法更为正式的定义。

算法是一组明确步骤的有序集合，它产生结果并在有限时间内终止。

算法必须是一组定义良好的有序的指令集合。

算法的每一步都必须有清晰、明白的定义。如某一步是将两整数相加，那么必须定义相加的两个整数和加法符号，相同的符号不能再某处用作加法符号，而在其他地方用作乘法符号。

算法必须产生结果，否则该算法就没有意义。结果可以是返回给调用算法的数据或其他效果（如打印）。

算法必须能够终止（停机）。如果不能，说明不是算法。

有一些算法在计算机科学中应用的非常普遍，我们称之为“基本”算法。这里讲讨论一些最常用的算法。讨论只是概括性的，具体的实现则取决于采用何种语言。

计算机科学中经常用到的一种算法是求和。你可以容易地实现两个或三个整数的相加，但是怎样才能实现一系列整数相加呢？答案很简单：在循环中使用加法操作。

求和算法可以分为三个逻辑部分：

1）将和（sum）初始化。

2）循环，在每次迭代总将一个新数加到和（sum）上。

3）推出循环后返回结果。

另一个常用算法是求出一系列整数的乘积。方法也很简单：在循环中使用乘法操作。乘法算法有三个逻辑部分：

1）将乘积（product）初始化。

2）循环，在每次迭代中将一个新数与乘积相乘。

3）推出循环后返回结果。

在本章开头讨论了求一组整数中最大值的算法。它的思想是通过一个判断结构求到两个数中的较大值。如果是把这个结构放在循环中，就可以求出一组数中的最大值。

求一组整数中的最小值和上面的方法相似，只有两个小小的不同。

计算机科学总的一个最普遍应用的是排序，即根据数据的值对它们进行排列。人们的周围充满了数据，如果这些数据都是无序的，可能会花很多时间去查找一条简单信息。

本节将介绍三种排序算法：选择排序、冒泡排序、插入排序。这三种方法是当今计算机科学中使用的快速排序的基础。

在选择排序中，数字列表可分为两个字列表，它们通过假想的一堵墙分开。求未排序子列表中最小的元素并把它和未排序子列表中第一个元素进行交换，进过每次选择和交换，两个子列表中假想的这堵墙向前移动一个元素，这样每次排序列表中将增加一个元素而未排序列表中将减少一个元素，每次把一个元素从未排序列表一道已排序列表就完成了一轮排序。一个还有 n 个元素的数字列表需要 $n-1$ 轮排序来完成数据的重新排序。

根据在 8.2 节描述的三种编程结构，可以为每一个可解的问题创建算法。结构化编程的原则要求将算法分成几个单元，称为子算法。每个子算法依次又分为更小的子算法。

使用子算法至少有两个优点：

程序更容易理解。

子算法可在主算法中不同地方调用，而无需重写。

程序员使用的另一个编程工具就是结构图。结构图是一种高级设计工具，它显示了算法和子算法之间的关系。它一般在设计阶段使用，而不是在编程阶段。

通常，有两种途径用于编写解决问题的算法。一种使用迭代，另一种使用递归。递归是算法的自我调用过程。

学习一个简单的例子，考虑一个阶乘的计算。阶乘的因子是从 1 到该数的整数。迭代的定义如图所示。如果算法的定义不涉及算法本身，则算法是迭代的。

每一个算法出现在它本身定义中，该算法就是递归定义的。

Chapter 6

程序设计语言

对计算机而言，要编写程序就必须使用计算机语言。计算机语言是指编写程序时，根据事先定义的规则而写出的预定义语句集合。计算机语言经过多年的发展已经从机器语言演化到高级语言。

在计算机发展的早期，唯一的程序设计语言是机器语言。每台计算机有其自己的机器语言，这种机器语言由“0”和“1”序列组成。在第 5 章中，我们看到在一台原始假想的计算机中，我们需要用 11 行代码去读两个整数、把它们相加并输出结果。当用机器语言来写时，这些代码就成了 11 行二进制代码，每行 16 位，如表所示。

机器语言是计算机硬件唯一能理解的语言，它由具有两种状态的电子开关构成：关（表示 0）或开（表示 1）。

计算机唯一识别的语言是机器语言。

虽然用机器语言编写真实地表示了数据是如何被计算机操纵的。但它至少有两个缺点：首先，它依赖于计算机。如果使用不同的硬件，那么一台计算机的机器语言与另一台计算机的机器语言就不同。其次，用这种语言编写程序时非常单调乏味的，而且很难发现错误。现在我们将机器语言时代称为编程语言的第一代。

编程语言中接下来的演化是伴随着带符号或助记符的指令和地址代替二进制码而发生的。因为他们使用符号，所以这些语言首先被称为符号语言。这些助记符语言后来就被称为汇编语言。假想计算机用于替代机器语言的汇编语言显示在程序中。

尽管汇编语言大大提高了变成效率，但任然需要程序员在所使用的硬件上花费大部分精力。用符号语言编程也很枯燥，因为每条机器指令都必须单独编码。为了提高程序员效率以及从关注计算机转到关注解决的问题，促进了高级语言的发展。

高级语言可移植到许多不同的计算机，使得程序员能够将精力集中在应用程序上，而不是计算机结构的复杂性上。高级语言旨在使程序员摆脱汇编语言繁琐的细节。高级语言同汇编语言有一个共性：它们必须被转化为机器语言，这个转化过程被称为解释或编译。

数年来，人们开发了各种各样的语言，最著名的有 BASIC、COBOL、Pascal、Ada、C、C++ 和 Java。

当今程序通常是同一种高级语言来编写。为了在计算机上运行程序，程序需

要被翻译成它要运行在其他的计算机的机器语言。高级语言程序被称为源程序。被翻译成的机器语言程序被称为目标程序。有两种方法用于翻译：编译和解释。

编译程序通常把整个源程序翻译成目标程序。

有些计算机语言使用解释器把源程序翻译成目标程序。解释是把源程序中的每一行翻译成目标程序中相应的行，并执行它的过程。但是，我们需要意识到在解释中的两种趋势：在 java 语言之前被有些程序使用的和 java 使用的解释。

在 java 语言之前的有些解释式语言使用一种被称为解释的第一种方法的解释过程，因为缺少其他任何名字，说以称为解释的第一种方法。在这种解释中，程序的每一行被翻译成其使用的计算机上的机器语言，该行机器语言被立即执行。如果在翻译或执行中有任何错误，过程就显示消息，其余的过程就被终止。程序需要被改正，再次从头解释和执行。第一种方法被看成是一种慢的过程，这就是大多数语言使用编译而不是解释的原因。

随着 java 的到来，一种新的解释过程就被引入了。java 语言能向任何计算机移植。为了取得可移植性，源程序到目标程序的翻译分成两步进行：编译和解释。java 源程序首先被编译，创建 java 的字节代码，字节代码看起来像机器语言中的代码，但不是任何特定计算机的目标代码，它是一种虚拟机的目标代码，该虚拟机称为 java 虚拟机或 JVM。字节代码然后能被任何运次 JVM 模拟器的计算机编译或解释，也就是运行字节代码的计算机只需要 JVM 模拟器，而不是 java 编译器。

编译和解释的不同在于，编译在执行前翻译整个源代码，而解释一次只翻译和执行源代码中的一行。但是，两种方法都遵循图中显示的相同的翻译过程。

词法分析器一个符号一个符号地读取源代码，创建源语言中的助记符表。

词法分析器分析一组助记符，找出指令。

语义分析器检查语法分析器创建的句子，确保它们不含有二义性。计算机语言通常是无二义性的，这意味着这一步骤或者是在翻译器中被省略，或者其责任被最小化。

在无二义性指令被语义分析器创建之后，每条指令将转化为一组程序将要在其上运行的计算机的机器语言。这个是由代码生成器完成的。

当今计算机语言按照它们使用的解决问题的方法分类。因此，模式是计算机语言看待要解决问题的一种方式。计算机语言课分成 4 中模式：过程性、面向对象、函数式和声说式。

在过程模式中我们把程序看成是操纵被动对象的主动主体。我们日常生活中遇到许多被动对象：石头、书、灯等。一个被动对象本身不能发出一个动作，但它能从主动主体接收动作。

过程式模式下的程序就是主动主体，该主体使用称为数据或数据项的被动对象。作为被动对象的数据项存储在计算机的内存中，程序操作他们。为了操纵数据，主动主体发出动作，称之为过程。

我们需要把过程和程序触发区分开。程序不定义过程，它只触发或调用过程。过程必须已经存在。

当使用过程式高级语言时，程序仅由许多过程调用构成，除此之外没有任何东西。这不是显而易见的，但即使使用像加法运算符这样的简单数学运算符时，我们也是正在使用一个过程调用一个已经编写的过程。

如果我们考虑过程和被作用于的对象，那么过程式模式的概念就变得更为简单，而且容易理解。这种模式的程序由三部分构成：对象创建部分、一组过程调用和每个过程的一组代码。有些过程在语言本身中已经被定义。通过组合这些代码，开发者可以建立新的过程。

面向对象模式处理活动对象，而不是被动对象。我们在日常生活中遇到许多活动对象：汽车、自动门、洗盘机。在这些对象上执行的动作都包含在这些对象那中：对象只需要接收合适的外部刺激来执行其中一个动作。

总体上，方法的格式与有些过程式语言中用的函数非常相似。每个方法有它的头、局部变量和语句。这就意味着我们对过程式语言所讨论的大多数特性都可以应用在为面向对象程序所写的方法上。换言之，我们可以认为面向对象语言实际上是带有新的理念和新的特性的过程式语言的扩展。

在面向对象模式中，作为本质，一个对象能从另一个对象继承。这个概念被称为继承性。当一般类被定义后，我们可以定义继承了一般类中一些特性的更具体的类，同时这些类具有一些新特性。

多态性因为他们使用符号，所以这些语言首先被称为符号语言。这些助记符语言后来就被称为汇编语言。假想计算机用于替代机器语言的汇编语言显示在程序中。

尽管汇编语言大大提高了变成效率，但任然需要程序员在所使用的硬件上花费大部分精力。用符号语言编程也很枯燥，因为每条机器指令都必须单独编码。为了提高程序员效率以及从关注计算机转到关注解决的问题，促进了高级语言的发展。

高级语言可移植到许多不同的计算机，使得程序员能够将精力集中在应用程序上，而不是计算机结构的复杂性上。高级语言旨在使程序员摆脱汇编语言繁琐的细节。高级语言同汇编语言有一个共性：它们必须被转化为机器语言，这个转化过程被称为解释或编译。

数年来，人们开发了各种各样的语言，最著名的有 BASIC、COBOL、Pascal、Ada、C、C++ 和 Java。

当今程序通常是同一种高级语言来编写。为了在计算机上运行程序，程序需要被翻译成它要运行在其他的计算机的机器语言。高级语言程序被称为源程序。被翻译成的机器语言程序被称为目标程序。有两种方法用于翻译：编译和解释。

编译程序通常把整个源程序翻译成目标程序。

有些计算机语言使用解释器把源程序翻译成目标程序。解释是把源程序中的每一行翻译成目标程序中相应的行，并执行它的过程。但是，我们需要意识到在解释中的两种趋势：在 java 语言之前被有些程序使用的和 java 使用的解释。

在 java 语言之前的有些解释式语言使用一种被称为解释的第一种方法的解释过程，因为缺少其他任何名字，说以称为解释的第一种方法。在这种解释中，程序的每一行被翻译成其使用的计算机上的机器语言，该行机器语言被立即执行。如果在翻译或执行中有任何错误，过程就显示消息，其余的过程就被终止。程序需要被改正，再次从头解释和执行。第一种方法被看成是一种慢的过程，这就是大多数语言使用编译而不是解释的原因。

随着 java 的到来，一种新的解释过程就被引入了。java 语言能向任何计算机移植。为了取得可移植性，源程序到目标程序的翻译分成两步进行：编译和解释。java 源程序首先被编译，创建 java 的字节代码，字节代码看起来像机器语言中的代码，但不是任何特定计算机的目标代码，它是一种虚拟机的目标代码，该虚拟机称为 java 虚拟机或 JVM。字节代码然后能被任何运次 JVM 模拟器的计算机编译或解释，也就是运行字节代码的计算机只需要 JVM 模拟器，而不是 java 编译器。

编译和解释的不同在于，编译在执行前翻译整个源代码，而解释一次只翻译和执行源代码中的一行。但是，两种方法都遵循图中显示的相同的翻译过程。

词法分析器一个符号一个符号地读取源代码，创建源语言中的助记符表。

词法分析器分析一组助记符，找出指令。

语义分析器检查语法分析器创建的句子，确保它们不含有二义性。计算机语言通常是无二义性的，这意味着这一步骤或者是在翻译器中被省略，或者其责任被最小化。

在无二义性指令被语义分析器创建之后，每条指令将转化为一组程序将要在其上运行的计算机的机器语言。这个是由代码生成器完成的。

当今计算机语言按照它们使用的解决问题的方法分类。因此，模式是计算机语言看待要解决问题的一种方式。计算机语言课分成 4 中模式：过程性、面向对象、函数式和声说式。

在过程模式中我们把程序看成是操纵被动对象的主动主体。我们日常生活中遇到许多被动对象：石头、书、灯等。一个被动对象本身不能发出一个动作，但它能从主动主体接收动作。

过程式模式下的程序就是主动主体，该主体使用称为数据或数据项的被动对象。作为被动对象的数据项存储在计算机的内存中，程序操作他们。为了操纵数据，主动主体发出动作，称之为过程。

我们需要把过程和程序触发区分开。程序不定义过程，它只触发或调用过程。过程必须已经存在。

当使用过程式高级语言时，程序仅由许多过程调用构成，除此之外没有任何东西。这不是显而易见的，但即使使用像加法运算符这样的简单数学运算符时，我们也是正在使用一个过程调用一个已经编写的过程。

如果我们考虑过程和被作用于的对象，那么过程式模式的概念就变得更为简单，而且容易理解。这种模式的程序由三部分构成：对象创建部分、一组过程调用和每个过程的一组代码。有些过程在语言本身中已经被定义。通过组合这些代码，开发者可以建立新的过程。

面向对象模式处理活动对象，而不是被动对象。我们在日常生活中遇到许多活动对象：汽车、自动门、洗盘机。在这些对象上执行的动作都包含在这些对象那中：对象只需要接收合适的外部刺激来执行其中一个动作。

总体上，方法的格式与有些过程式语言中用的函数非常相似。每个方法有它的头、局部变量和语句。这就意味着我们对过程式语言所讨论的大多数特性都可以应用在为面向对象程序所写的方法上。换言之，我们可以认为面向对象语言实际上是带有新的理念和新的特性的过程式语言的扩展。

在面向对象模式中，作为本质，一个对象能从另一个对象继承。这个概念被称为继承性。当一般类被定义后，我们可以定义继承了一般类中一些特性的更具体的类，同时这些类具有一些新特性。

多态性因为他们使用符号，所以这些语言首先被称为符号语言。这些助记符语言后来就被称为汇编语言。假想计算机用于替代机器语言的汇编语言显示在程序中。

尽管汇编语言大大提高了变成效率，但任然需要程序员在所使用的硬件上花费大部分精力。用符号语言编程也很枯燥，因为每条机器指令都必须单独编码。为了提高程序员效率以及从关注计算机转到关注解决的问题，促进了高级语言的发展。

高级语言可移植到许多不同的计算机，使得程序员能够将精力集中在应用程序上，而不是计算机结构的复杂性上。高级语言旨在使程序员摆脱汇编语言繁琐的细节。高级语言同汇编语言有一个共性：它们必须被转化为机器语言，这个转化过程被称为解释或编译。

数年来，人们开发了各种各样的语言，最著名的有 BASIC、COBOL、Pascal、Ada、C、C++ 和 Java。

当今程序通常是同一种高级语言来编写。为了在计算机上运行程序，程序需要被翻译成它要运行在其他的计算机的机器语言。高级语言程序被称为源程序。被翻译成的机器语言程序被称为目标程序。有两种方法用于翻译：编译和解释。

编译程序通常把整个源程序翻译成目标程序。

有些计算机语言使用解释器把源程序翻译成目标程序。解释是把源程序中的每一行翻译成目标程序中相应的行，并执行它的过程。但是，我们需要意识到在解释中的两种趋势：在 java 语言之前被有些程序使用的和 java 使用的解释。

在 java 语言之前的有些解释式语言使用一种被称为解释的第一种方法的解释过程，因为缺少其他任何名字，说以称为解释的第一种方法。在这种解释中，程序的每一行被翻译成其使用的计算机上的机器语言，该行机器语言被立即执行。如果在翻译或执行中有任何错误，过程就显示消息，其余的过程就被终止。程序需要被改正，再次从头解释和执行。第一种方法被看成是一种慢的过程，这就是大多数语言使用编译而不是解释的原因。

随着 java 的到来，一种新的解释过程就被引入了。java 语言能向任何计算机移植。为了取得可移植性，源程序到目标程序的翻译分成两步进行：编译和解释。java 源程序首先被编译，创建 java 的字节代码，字节代码看起来像机器语言中的代码，但不是任何特定计算机的目标代码，它是一种虚拟机的目标代码，该虚拟机称为 java 虚拟机或 JVM。字节代码然后能被任何运次 JVM 模拟器的计算机编译或解释，也就是运行字节代码的计算机只需要 JVM 模拟器，而不是 java 编译器。

编译和解释的不同在于，编译在执行前翻译整个源代码，而解释一次只翻译和执行源代码中的一行。但是，两种方法都遵循图中显示的相同的翻译过程。

词法分析器一个符号一个符号地读取源代码，创建源语言中的助记符表。

词法分析器分析一组助记符，找出指令。

语义分析器检查语法分析器创建的句子，确保它们不含有二义性。计算机语言通常是无二义性的，这意味着这一步骤或者是在翻译器中被省略，或者其责任被最小化。

在无二义性指令被语义分析器创建之后，每条指令将转化为一组程序将要在其上运行的计算机的机器语言。这个是由代码生成器完成的。

当今计算机语言按照它们使用的解决问题的方法分类。因此，模式是计算机语言看待要解决问题的一种方式。计算机语言课分成 4 中模式：过程性、面向对象、函数式和声说式。

在过程模式中我们把程序看成是操纵被动对象的主动主体。我们日常生活中遇到许多被动对象：石头、书、灯等。一个被动对象本身不能发出一个动作，但它能从主动主体接收动作。

过程式模式下的程序就是主动主体，该主体使用称为数据或数据项的被动对象。作为被动对象的数据项存储在计算机的内存中，程序操作他们。为了操纵数据，主动主体发出动作，称之为过程。

我们需要把过程和程序触发区分开。程序不定义过程，它只触发或调用过程。过程必须已经存在。

当使用过程式高级语言时，程序仅由许多过程调用构成，除此之外没有任何东西。这不是显而易见的，但即使使用像加法运算符这样的简单数学运算符时，我们也是正在使用一个过程调用一个已经编写的过程。

如果我们考虑过程和被作用于的对象，那么过程式模式的概念就变得更为简单，而且容易理解。这种模式的程序由三部分构成：对象创建部分、一组过程调用和每个过程的一组代码。有些过程在语言本身中已经被定义。通过组合这些代码，开发者可以建立新的过程。

面向对象模式处理活动对象，而不是被动对象。我们在日常生活中遇到许多活动对象：汽车、自动门、洗盘机。在这些对象上执行的动作都包含在这些对象那中：对象只需要接收合适的外部刺激来执行其中一个动作。

总体上，方法的格式与有些过程式语言中用的函数非常相似。每个方法有它的头、局部变量和语句。这就意味着我们对过程式语言所讨论的大多数特性都可以应用在为面向对象程序所写的方法上。换言之，我们可以认为面向对象语言实际上是带有新的理念和新的特性的过程式语言的扩展。

在面向对象模式中，作为本质，一个对象能从另一个对象继承。这个概念被称为继承性。当一般类被定义后，我们可以定义继承了一般类中一些特性的更具体的类，同时这些类具有一些新特性。

多态性.

人们已经发明了一些面向对象语言。我们简要讨论其中两种语言的特性：C++ 和 java。

Java 是由 Sun Microsystems 公司开发的，它在 C 和 C++ 的基础上发展而来，但是 C++ 的一些特性从语言中被移除，从而使 java 更健壮。另外，改语言是完全面向类操作的。在 C++ 中，你甚至可以不用定义类就能解决问题。而在 java 中，每个数据项都属于一个类。

java 中的程序可以是一个应用程序也可以是一个小程序。应用程序是指一个可以完全对运行的程序。小程序则是嵌入超文本标记语言中的程序，存储在服务器上并由浏览器运行。浏览器也可以把它从服务器下载到本地运行。

在 java 中，应用程序（或小程序）是类以及类实例的集合。java 自带的丰富类库是它的有趣特征之一。尽管 C++ 也提供类库，但是 java 中用户可以在提供的类库基础上构建新类。

在 java 中，程序的执行也是独具特色的。构建一个类并把它传给编译器。由编译器来调用类的方法。java 的另一大有趣的特点是多线程。线程是指按顺序执行的动作序列。C++ 只允许单线程执行，但是 java 允许多线程执行。

在这一节，我们通过对一些过程式语言的快速浏览，发现共同的概念。这些概念中的一些读大多数面向对象也适用，这是因为当创建方法时面向对象模式使用过程式模式。

所有计算机语言的共同特点之一就是具有标识符，即对象的名称。标识符允许给程序对象命名。

数据类型定义了一系列值及应用于这些值的一系列操作。每种数据类型值的集合被称为数据类型的域。大多数语言都定义了两类数据类型：简单数据类型和复合数据类型。

简单数据类型是不能分解成更小数据类型的数据类型。

复合数据类型是一组元素，其中每个元素都是简单数据类型或复合数据类型。

变量是存储单元的名字。

字面值是程序中使用的预定义的值。

常量

输入和输出

表达式是由一系列操作数和运算符简化后的一个单一数值。运算符。操作数。

每条语句都使程序执行一个相应动作。它被直接翻译成一条或多条计算机可执行指令。

Chapter 7

软件工程

软件生命周期是软件工程中的一个基础概念。软件和其他的产品一样，周期性地重复着一些阶段。

软件最初由开发小组开发。通常，在它需要修改之前会使用一段时间。由于软件会发现错误、设计改变规则或公司本身发生变化，这些都导致需要经常修改软件。为长久使用考虑软件应该被修改。使用和修改，这两个步骤移植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他因素而导致软件失去它的有效性。

虽然软件工程及图中的所有三个过程，但本章我们只讨论开发过程，它在图中处于循环流程之外。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。开发过程有多种模型，这里我们讨论常见的两种：瀑布模型和增量模型。

软件开发过程中一种非常流行的模型就是众所周知的瀑布模型。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。

例如，整个工程的分析阶段应该在设计阶段开始前完成。整个设计阶段应该在是现阶段前完成。

瀑布模型既有优点，也有缺点。优点之一就是在下一个阶段开始前每个阶段已经完成。但是，瀑布模型的缺点是难以定位问题：如果过程的一部分有问题，必需检验整个过程。

在增量模型中，软件的开发要历经一些列步骤。开发者首先完成整个系统的一个简化版本，这个版本表示整个系统，但不包括具体的细节。

在第二个阶段中，更多的细节被加入，而有些还没完成，然后再次测试系统。如果这时有问题，开发者知道问题出于新功能。知道现有系统工作正确后，它们才增加新的功能。这样过程一直继续下去，直到要求的功能全部加入。

整个开发过程始于分析阶段，这个阶段生成规范说明文档，这个文档说明明了软件要做什么，而没有说明如何去做。分析阶段可以使用两种独立的方法，它们依赖于是现阶段使用的是过程编程语言，还是面向对象语言。

如果是现阶段使用过程语言，那么面向过程分析就是分析阶段使用的方法。这种情况下的规格说明可以使用多种建模工具，但是我们只讨论其中少数几个。

数据流图显示了系统中数据的流动。它们使用 4 中符号：方形盒表示数据源或数据目的，带圆角的矩形表示过程。末端开口的矩形标识数据存储的地方，箭

头表示数据流。

分析阶段使用的另一个建模工具是实体关系图。因为这个图也用于数据库的设计，所以我们将在 12 章中讨论。

状态图提供了另外一种有用的工具，它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。作为状态图的一个例子，我们显示了单人电梯的操作。当楼层按钮被按，电梯将按要求的方向移动，在到达目的地之前，它不会响应其他任何请求。

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。这种情况下的规格说明文档可以使用多种工具，但我们只讨论其中少数几种。

在瀑布模型中，设计阶段完成后，实现阶段就可以开始了。在这个阶段，程序员为面向过程设计中的模块编写程序或编写程序单元，实现面向对象设计中的类。在每种情况中，都有一些我们需要提及的问题。

在面向过程开发中，工程团队需要从第 9 章所讨论的面向过程语言中选择一个或一组语言。虽然有些语言被看成既是面向过程的，又是面向对象的语言，但通常实现只用纯过程语言，如 C 语言。在面向对象的情况下，C++ 和 java 的使用都很普遍。

在实现阶段创建的软件质量是一个非常重要的问题。具有高质量的软件系统是一个能满足用户需求、符合组织操作标准和能高效运行在为其开发的硬件上的一个软件。但是，如果我们需要取得高质量的软件系统，那么必须能定义质量的一些属性。

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。每个度量如图所示还可以扩展。

可操作性涉及系统的基本操作。就像图中的显示一样，可操作性有多种度量方法：准确性、高效性、可靠性、安全、及时性和适用性。

不准确的系统比没有系统更糟糕。任何被开发的系统都必须经过系统测试工程师和用户检测。准确性能够通过诸如故障平均时间、每千行代码错误数以及用户请求变更数这样的测量指标来度量。

高效性大体上是个主管的术语。在一些实例中，用具将指定性指标，例如实时响应必须在 1 秒之内接收到，成功率在 95

可靠性实际上综合了其他各种因素。如果用户指望系统完成工作并对其有信心，这时它就是可靠的。另外，一些度量直接说明了系统的可靠性，最显著的故障平均时间。

一个系统的安全性是以未经授权的人得到系统数据的难易程度为参照。尽管这时一个主观的领域，但仍然有可核查的清单帮助评估系统的安全性。例如，系统有并且需要密码来验证用户吗？

在软件工程中，及时性意味着几件不同的事情。系统及时传递它的输出了吗？对于在线系统，响应时间满足用户的需求了吗？

适用性是另一个很主观的领域。对适用性的最好度量方法是观察用户，看他们是如何使用这个系统的。用户访谈能够常常发现系统适用性上的问题。

可维护性以保湿系统正常运行并及时更新为参照。很多系统需要经常修改，这不是因为它们不能拿很好地运行而是因为外部因素的改变。

可变性是个主观因素。

可修正性的只用度量是回复正常的平均时间。

可迁移性是指把数据和系统从一个平台移动到另一个平台并重用代码的能力。在很多情况下，这不是一个很重要的因素。另一方面，如果你编写具有通用性的软件，那么可迁移性就很关键了。

测试阶段的目的就是发现错误，这就意味着良好的测试策略能发现最多的错误。有两种测试：白盒测试和黑盒测试。

白盒测试是基于知道软件内部结构的。测试的目的是检验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。在这种情况下，程序就像玻璃盒子，其中的每件事都是可见的。白盒测试由软件工程师或一个专门的团队来完成。使用软件的白盒测试需要保证至少满足下面 4 条标准：

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。

在过去，已经有多正测试方法，我们只讨论其中的两种：基本路径测试和控制结构测试。

基本路径测试是由 Tom McCabe 提出。这种方法创建一组测试用例，这些用例执行软件中的每条语句至少一次。

基本路径测试使用图论和圈复杂性找到必须被走过的独立路径，从而保证每条语句至少被执行一次。

控制结构测试比基本路径测试更容易理解并且包含基本路径测试，这种方法使用下面将要简要讨论的不同类的测试。

（1）条件测试

条件测试应用于模块中的条件表达式，简单条件是关系表达式，而复合条件是简单条件和逻辑运算符的组合。条件测试用来检查是否所有的条件都被正确设置。

（2）数据流测试

数据流测试是基于通过模块的数据流。这种测试选择测试用例，这些用例涉及检查被用在赋值语句左边的变量的值。

（3）循环测试

循环测试使用测试用例检查循环的正确性。所有类型的循环都仔细测试。

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。换言之，程序就像看不见内部的黑盒。黑盒测试按照软件应该完成的功能来测试软件，如它的输入和输出。下面介绍几种黑盒测试方法。

最好的黑盒测试方法就是用输入域中的所有可能的值去测试软件。但是，在复杂的软件中，输入域是如此巨大，这样做常常不现实。

在随机测试中，选择输入域的值的子集来测试，子集选择的方式是非常重要的。在这种情况下，随机数生成器是非常有用的。

当遇到边界值时，错误经常发生。例如，一个模块定义它的输入必须大于或等于 100，那这个模块用边界值 100 来写实就非常重要。如果模块在边界值出错，那有可能就是模块代码中的有些条件，例如，$x > 100$ 被写成 $x > 100$。

软件的正确使用和有效维护离不开文档。通常软件有三个独立的文档：用户文档、系统文档和技术文档。注意文档是一个持续的过程。如果软件在发布之后有问题，也必须写文档。如果软件被修改，那么所有的修改和与原软件包间的关系都要被写进文档。只有当软件包过时后，编写文档才停止。

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。它告诉用户如何一步步地使用软件包。它通常包含一个教程知道用户熟悉软件包的各项特征。

一个好的用户手册能够称为一个功能强大的营销工具。用户文档在营销中的重要性再强调也不过分。手册应该面向新手和专业用户。配有好的用户文档必定有利于软件的销售。

系统文档定义软件本身。转系系统文档的目的是为了让原始开发人员之外的人能够维护和修改软件包。系统文档在系统开发的所有 4 个阶段都应该存在。

在分析阶段，收集的信息应该仔细地用文档记录。另外，系统分析员应该定义信息的来源。需求和选用的方法必须用基于它们的推论来清楚表述。

在设计阶段，最终版本中用到的工具必须记录在文档中。例如，如果结构图修改了多次，那么最终的版本要用完整的注释记录在案。

在实现阶段，代码的每个模块都应记录在文档中。另外，代码应该使用注释和描述头尽可能详细地形成自文档。

最后，开发人员必须仔细地形成测试阶段的文档。对最终产品使用的每种测试，连同它的结果都要记录在文档中。甚至令人不快的结果和产生它们的数据也要记录在案。

技术文档描述了软件系统的安装和服务。安装文档描述了软件如何按章在每个计算机上，如服务器和客户端。服务文档描述了如果需要，系统应该如何维护和跟新。

Chapter 8

数据结构

在前面章节中，我们使用变量来存储单个实体，尽管单变量在程序设计语言中被大量使用，但它们不能有效地解决复杂问题。本章将介绍数据结构，它是第 12 章首相数据结构的前奏。

数据结构利用了有关的变量的集合，而这些集合能够单独或作为一个整体被访问。换句话说，一个数据结构代表了有特殊关系的数据的集合。本章将讨论三种数据结构：数组、记录和链表，大多的编程语言都隐式实现了前两种而第三种则通过指针和记录来模拟。

假设 100 个分数，我们需要读入这些书，处理他们并打印。同时还要求将这 100 个分数在处理过程中保留在内存中。可以定义 100 个变量，每个都有不同的名字。

但是定义 100 个不同的变量名带来了其他问题，我们需要 100 个引用来读取它们；需要 100 个引用来处理它们；需要 100 个引用来写它们。

及时是处理这些数目相对小的分数，我们需要的指令数目也是无法接受的。为了处理大量的数据，需要一个数据结构。

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。虽然有些编程语言允许在一个数据中可以有不同类型的元素。我们可以成数组中的元素为第一个元素、第二个元素等，知道最后一个元素。如果将 100 个分数放进数组中，可以指定元素为 csores[1]、scores[2]，等等。索引表示元素在数组中的顺序号，顺序号从数组开始处计数。数组元素通过索引被独立给出了地址，这个数组整体上有个名称：scores，但每个数可以利用他的索引来单独访问。

我们可以使用村换来读写数组中的元素，也可以使用村换来处理元素。现在我们可以不管要处理的元素是 100 个、100 个或 10 000 个，循环使得处理他们变得容易。我们可以使用一个整数变量来控制循环，只要变量的值小于数组找那个元素的个数，那还在循环体里面。

在一个数组中，有两种标识符：数组的名字和各个元素的名字。数组名是整个结构的名称，而元素的名字允许我们查阅这个元素。在上个例子中，数组的名字是 scores，而每个元素的名字是这个名字后面跟索引，如 scores[1]、scores[2] 等。在本章中，我们大部分需要的是元素的名字，但是，有些语言中，也需要使用数组的名字。

到目前为止，我们所讨论的都是一维数组，以内数据仅是在一个方向上线性组成。许多的应用要求数据存储在多维中。常见的例子如表格，就是包括行和

列的数组。

虽然我们能应用一些为数组中每个元素定义的通常操作，但我们还能定义一些把数组作为数据结构的操作，数组作为结构的常用操作有：查找、插入、删除、检索和遍历。

当我们知道元素的值时，经常需要找到元素的序号，这种操作在第 8 章讨论过。我们可以对未排序的数组使用顺序查找，对排序的数组使用折半查找。下面三种操作都要使用查找操作。

通常，计算机语言要求数组的大小在程序被写的时候就被定义，防止在程序的执行过程中被修改。最近，有些语言允许可变长数组。及时语言允许可变长数组，在数组找那个插入一个元素仍需要十分小心。

如果插入操作在数组尾部进行，而且语言允许增加数组的大小，那么可以很容易完成这个操作。

如果插入是在数组的开始或中间，过程就是冗长的和花费时间的。当我们需要在一个有序的数组中插入一个元素时，这就发生了。就像前面描述的，首先查找数组。找到插入的位置后，插入新的元素。例如...

在数组中删除一个元素就像插入操作一样冗长和棘手。

检索操作就是随便地存取一个元素，达到检查或复制元素中的数据的目的。与插入和删除操作不同，当数据结构是数组时，检索是一个容易地操作。实际上，数据是随机存取结构，这意味着数组的每个元素可以随机地被存取，而不需要存取该元素前面的元素或后面的元素。

数组的遍历是指被应用于数组中每个元素上的操作，如读、写、应用数学的运算等。

考虑一下前一节所讨论的操作，就给出了数组应用的提示。如果有一个表，在表创建后有大量的插入和删除操作要进行，这是就不应该使用数组。当删除和插入操作较少，而有大量查找和检索操作时，这时比较适合使用数组。

记录是一组相关元素的集合，它们可能是不同的类型，但整个记录有一个名称。记录中的每个元素称为域。域是具有含义的最小命名数据。它有类型且存在于内存中。它能你被赋值，繁殖也能够被选择和操纵。域不同于变量主要在于它是记录的一部分。

在记录中的元素可以是相同类型或不同类型，但记录中的说有元素必须是关联的。

就像数组一样，在记录中也有两种标识符：记录的名字和记录总各个域的名字。记录的名字是整个结构的名字，而每个域的名字允许我们存取这些域。

我们可以从概念上对数组和记录进行比较。这有助于我们理解什么时候应该使用数组，什么时候应该使用记录。数组定义了元素的集合，而记录定义了元素可以确认的部分。

如果我们需要定义元素的耳机和，同时需要定义元素的属性，那么可以使用记录数组。

数组和记录数组都表示数据项的列表。数组可以被看成是记录数组的一种特例，其中每个元素是自带一个域的记录。

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。另外，一个指针变量标识该别表中的第一个元素。列表的名字就是该指针变量的名字。

链表中的元素习惯上称为节点，链表中的节点是至少包括两个域的记录：一个包含数据，另一个包含链表中下一个节点的地址。

在对链表进行进一步讨论前，我们需要解释一个在图中的几号。使用连线显示两个节点间的连接。显得一段有一箭头，线的另一端是实心圆。箭头表示箭头所指节点的地址副本。实心圆显示了地址的副本存储的地方。

数组和链表都能表示内存中的数据项列表。唯一的区别在于数据项连接在一起的方式。在记录数组中，连接工具是索引。元素 scores[3] 与元素 scores[4] 相连，因为整数 4 是紧跟着整数 3 后面的。在链表中，连接工具是指向下一元素的链。

数组的元素在内存中是一个接一个中间无间隔存储的，即列表是连续的。而链表中的节点在存储中间是有间隔的：节点的链部分包数据项“胶”在一起。换言之，计算机可以选着连续存储它们或把节点分布在整个内存中。这样有一个优点：在立案表彰进行的插入和删除操作更容易些，只需改变指向下一元素地址的指针。但是，这带来了二外的开销：链表的每个节点有一个额外的域，存放内存中下一节点的地址。

就像数组和记录一样，我们需要区分链表名和节点名。一个链表必须要有一个名字。

链表名是头指针名字，该头指针指向表中第一个节点。另一方面，节点在链表中并没有明显的名字，有的只是隐含的名字。节点的名字与指向节点的指针有关。不同的语言在处理节点与指针节点间的关系时是不同的。我们使用 C 语言中使用的约定。

Chapter 9

抽象数据类型

在这一章中，我们讨论抽象数据结构，这是一种比我们在第 11 章所讨论的数据结构处于更高抽象层的数据类型。ADT 使用数据结构来实现。在本章的开始首先对抽象数据类型做一个简短的背景介绍，然后给出定义并提出模型。接着讨论各种不同的抽象数据类型，例如，栈、列队、广义线性表、树、二叉树和图。

使用计算机进行问题求解意味着处理数据。为了处理数据，我们需要定义数据类型和在数据上进行的操作。例如，要求一个列表中的数字之和，我们应该选择数字的类型和定义运算。数据类型的定义和应用于数据的操作定义是抽象数据类型（ADT）背后的一部分概念，隐藏数据上的操作是如何进行的。换言之，ADT 的用户只需要知道对主句类型可用的一组操作，而不需要知道他们是如何应用的。

许多编程语言已经定义了一些简单的抽象数据累心工作位语言的整数部分。例如，C 语言定义了称为整数的简单抽象数据类型。这种类型的抽象数据类型是带有预先定义范围的整数。C 还定义了可以再这种数据类型上应用的几种操作。C 显示地定义了整数上的这些操作和我们期望的结果。写 C 程序来进行两个整数相加，程序员应该知道整数的抽象数据类型和可以应用于该抽象数据类型的操作。

但是，程序员不需知道这些操作是如何实现的。例如，程序员使用表达式 $z \leftarrow x + y$，希望 x 的值被加到 y 的值上。结果被命名为 z。程序员不需要知道加法是如何进行的。在前一章我们学到计算机是如何执行加法运算的：把两个整数以补的格式存储在内存地址中，把它们装进 CPU 的寄存器中，进行二进制相加，把结果回存到另一个内存地址中。但是，程序员不必知道这些。C 语言中的整数就是一个带有预定义操作的简单抽象数据类型。程序员不必关注操作是如何进行的。

几种简单的抽象数据类型已经被实现，在大多数语言中它们对用户是可用的。但是许多有用的复杂抽象数据类型却没有实现。就像我们在这一章中将要看到的，我们需要表抽象数据类型、栈抽象数据类型、列队抽象数据类型等。要提高效率，应该连理这些朝向数据类型，将他们存储在计算机库中，以便使用。例如，表的使用者只需直达该表上有哪些可用的操作，而不需知道这些操作是如何进行的。

因此，对于一个 ADT，用户不用关系任务是如何完成的，而是关系能做什

么。换言之，ADT 包含了一组允许程序使用的操作的定义，而这些操作的实现是隐藏的。这种不需详细地说明实现过程的泛化操作称为抽象。我们抽取了过程的本质，而隐藏了实现的细节。

让我们正式地定义抽象数据类型。抽象数据类型就是与对该数据类型有意义的操作封装在一起的数据类型。然后，用它封装数据和操作并对用户隐藏。

抽象数据类型的模型如图。

计算机语言不提供抽象数据类型包。要使用抽象数据类型，首先要实现它们，把它们存储在库中。本章主要介绍一些常见的抽象数据类型以及应用。但是，我们也为对此有兴趣的读者提供了每个抽象数据类型实现的简单讨论。我们把实现的伪码算法作为挑战练习。

栈是一种限制线性列表，该类列表的添加和删除操作只能在一段实现，称为“栈顶”。如果掺入一系列数据到栈中，然后移走它们，那么数据的顺序将被倒转。数据掺入是的顺序为 5，10，15，20，移走后顺序就变成 20，15，10，5。这种倒转的属性也正是栈被称为后进先出（LIFO）数据结构的原因。

尽管栈有很多操作，但基本操作有 4 种：建栈、入栈、出栈和空。

建栈操作常见一个空栈，可是如下：

```
push(stackName,dataItem)
```

stackName 是要创建栈的名字。这个操作返回一个空栈。

入栈操作在栈顶添加新的元素，格式如下：

```
push(stackName,dataitem)
```

stackName 是栈的名字，dataItem 是要插在栈顶的数据。入栈后，新的元素称为栈顶元素。这个操作返回一个 dataItem 插在顶端的新栈。

出栈操作将栈顶元素移走

```
pop(stackName,dataItem)
```