应用软件开发

实用程序汇集

汪合生　孙刚春　张　政　编

成都科技大学计算中心
成都科技大学计算机应用教研室
成都市电子学会计算机应用专业委员会

应用软件开发与实用程序汇集

编　　辑：汪合生　孙刚春　张　政
主　　审：刘竞成
责任编辑：李宏信　张　政

成都科技大学计算中心
成都科技大学计算机应用教研室
成都市电子学会计算机应用专业委员会

内 容 简 介

本书以实际系统为例，着重介绍管理软件的开发和数据处理技术。全书共六章。第一章介绍应用软件开发与设计基础。第二章介绍实用中文工资管理系统。第三章介绍实用中文通用报表管理系统。第四章介绍中文科技情报检索系统。第五章介绍高考统分系统。第六章介绍中文人事档案管理系统的设计思想。

本书详细阐述了以上系统的设计思想，给出了详细的程序框图和部分模块程序清单。

本书可以作为大专院校计算机专业师生教学、毕业设计参考，对于从事计算机应用软件开发的广大计算机工作者和科技人员具有较大的参考价值。尤其是对于即将从事软件开发与研制的广大科技人员是一本很好的参考书籍。

应用软件开发与实用程序汇集

（内部印）

前　言

近几年来，电子计算机（主要是微型计算机）在我国已开始普遍推广应用。其表现是电子计算机已经从以大型科学计算为主要目的的技术象牙塔中走了出来，广泛地应用于以数据处理为主体的国民经济各部门之中。而数据处理是当今世界计算机应用最活跃、最庞大的领域，它包括非数值计算方面对任何型式数据（信息）的所有计算、管理和加工处理。例如企业管理、库房管理、报表统计、帐目计算、信息情报检索等。计算机的发展历史表明：它是在科学计算领域中诞生，却是在数据处理领域开拓中找到了真正的立足点。数据处理威力无比，它的成果、进步，不断改变世界面貌，导致了信息化时代的到来。

由于数据处理领域在整个计算机应用领域中占有如此重要的地位，因此，及时地使计算机数据处理方面的成果得到推广应用，是一项刻不容缓的工作。我们编辑印行此书的目的正在于此。鉴于国内目前在数据处理方面的成果交流上还处于十分不畅的局面，我们希望此书同时能对改变此种状况起到一些推动作用，使计算机的普及应用工作在我国稳步扎实地进行，对整个国民经济的发展起到应有的作用。

本书内容是根据成都科学技术大学计算机及自动控制系计算机应用教研室，计算中心部分教师和科技人员近两年来在数据处理、管理软件开发中的一些实际尝试，整理汇集而成。书中所有的这些系统已投入实用和试用，即将作为软件产品推出。

成都科学技术大学计算机与自动控制系主任刘竞成付教授担任全书主审，对本书的编著提出了许多指导性的宝贵意见。

成都科学技术大学计算机与自动控制系付系主任李宏信，计算机应用教研室主任龚荣武、付主任郑桂林分别担任部分章节的主审，对本书的编著提出了许多宝贵意见。

由于我们编著经验不足，应用软件开发和数据处理技术的水平有限，因此书中错误和缺陷一定不少，恳请国内计算机界的同行和读者及时给予批评指正，如蒙赐教，十分欣喜。

编　者

一九八四年九月于成都科技大学

目　录

第一章　应用软件开发基础

汪合生　　孙刚春　　张　政

本章主要介绍程序设计的一般步骤和方法，应用系统的研制和开发，应用软件开发中的技术文件编制。所有这些介绍都从应用的角度来讨论问题，使具有一般计算机软件基础，或者即将从事应用软件开发的读者不感到费解，有所助益。

§1. 程序设计的步骤和方法

今天，程序设计不仅是一种技术，而且必须用科学的原理和方法设计程序，因此它有别于传统的设计方法。尤其是在1968年由NATO（北大西洋公约组织）主办的会议上提出了“软件工程”的概念，用工程方法研究和开发软件后，在程序设计方法、软件管理、开发支撑系统等方面，都提出了许多新的理论、思想和方法。可以预料，在八十年代末，由于结构程序设计、说明、验证、语言设计等领域中的进展将会使程序设计变成为一门真正的工程学科。本节主要介绍如何从问题的定义开始，采用结构式程序设计、模块化程序设计、自顶向下设计等先进的软件设计技术把一个任务变为程序模块，以及如何把这些程序模块组装起来，形成一个良好的工作系统，即讨论对任务(或问题)进行程序设计的步骤、方法和技术。

对问题（或任务）进行程序设计，一般可分为以下五大步骤。

§1.1 课题分析

系统的目标是系统开发的依据和归宿。而系统目标的确定往往依赖于实际课题（或任务)的需要，以及可能提供的各种资源条件，如技术力量、财力、支撑环境等等。因此，对一个任务进行程序设计时，首先要对解决的问题进行透彻的分析，以求对问题有正确的理解，明确最终开发出来的软件产品“做什么”的问题。课题的分析包括解决问题的目的是什么，最终要达到的功能、要求、技术指标，现有的各种条件，已知的数据，速度和精度的要求等等。

弄清问题、确定系统目标是整个软件开发的基点，是解决科研、生产、计划事务管理等实际问题时所不可缺少的一步。

弄清问题、确定系统目标往往不是轻而易举的。其原因主要有三：一是用户很可能对计算机能干什么缺乏明确的概念，没有定量的标准，他们不能把问题的要求或业务处理变成计算机上实现的流程。例如，在开发一个企事业单位的管理信息系统时，一般是先由企、事业单位的主管领导人及业务管理人员提出任务要求，而他们提出的要求往往是不精确的、含混的用户说明，这只能作为形成系统目标的素材。二是软件研制人员对实际问题或某个企、事业单位的管理业务并不熟悉，容易“自以为是”地对用户的任务要求进行补充、修正，或者规定具体的数量界限，结果造成与实际情况有很大的距离。

三是确定的系统目标是否可行，在很大程度上取决于计算机系统的功能。而目前不少单位的做法是先购置（或给定）某个计算机系统，再进行应用系统的分析和设计，这就使得计算机系统与该单位中所需要的信息系统的适应性存在着突出的矛盾。

如果忽视了上述这些因素，轻率的确定目标就进入软件的实际设计，常常造成返工，浪费大量的人力、物力，我们应当引以为戒。

§1.2 确定算法

解决一个问题，常常有几种可供选择的方法。从数学的角度来描述，可能有几种不同的算法。在编制程序之前，先要对不同的算法进行分析、比较，找出最适宜的算法。例如，开发情报检索系统，由于信息量大、实时性高，对文献数据库和各种倒排文件的建立、所采用的查找算法要在时间和空间上作出决策，选择较好的时、空指标 O(t,M)。又如，用迭代法解微分方程，要考虑收敛的快慢，在一定的时间内，能否达到所要求的精度，因此，速度问题是重点应考虑的指标。而有的问题可能要求在给定的内存容量里解决，而对时间的要求并不苛刻，在这种情况下，速度较慢而节省内存的算法可能被优先录取。

简单的算法可以根据有关的数学理论直觉地设计出来；复杂的算法往往可由简单的算法按照各种模式搭配而成。然而，哪种算法最佳，并不是绝对的，它要根据多方面的因素作出决策，因此没有一种机械的方法来生成各种算法。

根据问题的要求，实现的环境，找出适当的算法，再用程序设计语言把它书写出来，这是程序设计的最有特色的部分。因此，确定算法或者说发现算法，是一项创造性工作。

§1.3 程序结构的设计

程序结构的设计是把研究的课题或任务转化为程序的准备阶段。如果程序较小且简单，此阶段可能仅仅是绘制一张流程图。但要设计大型程序，大型程序设计中存在的问题就很多了。其一是复杂程度高，其主要表现为程序规模大，接口信息多，动态性强和并发行强。其二是软件生成周期长。其三是程序的正确性难保证。由于大型程序的复杂程度高、生成周期长，程序中所包含的模块较多，参与设计的人员也多，从而导致其正确性较难保证。正如 E.W.Dijkstra 所说："迄今为止，所有的程序调试设施只能发现程序中的错误，并不能表明程序没有错误"。事实上，不少大型程序在使用了几年以后，自然不断发现其潜在的错误。

为了解决上述问题，程序设计者就要考虑和采取新型的程序设计方法和工具。下面的讨论中，我们将扼要介绍画流程图、模块化程序设计、结构式程序设计和自顶向下设计等方法，并对这些方法作出比较，指出它们的优缺点。也就是说，我们并不向你推荐任何特定的方法，因为目前还没有充分的证据说明某一种方法对于任何问题都比其它方法优越。因此，读者应当知道，目标是要得出一个良好的工作系统，而不是学究式的来对待这些方法。

但是，所有的程序设计方法都有着若干共同的原则，其中有些原则对于各种方法都是适用的。这些原则是：

1. ——小步一小步地进行，不要企图一次做得太多。

2. 把大的作业分割成小的、逻辑上独立的若干任务。尽可能使各个小的任务相互无关，以便能够分别进行测试，并且在改变一个任务时不致于影响其它任务。

3. 控制流程要尽可能简单，以便容易查找错误。

4. 尽可能地用图表或者图形描述，因为它们比文字说明要直观得多。

5. 要把清晰性、简单性放在首位，这样便于系统的阅读、修改和维护。

6. 要以严谨而系统的方式进行设计，防止可能出现混乱的状况。

7. 必须对系统进行查错、测试、维护。

8. 程序设计中，重复不是缺点，复杂化也不是优点，因而应使用简单而一致的术语和方法。

9. 在程序结构设计阶段结束后再着手编制程序。在此之前，不要急于动手编写指令或语句，否则就如同在未确定线路之前就去列元器件清单或设计印刷电路板一样毫无意义。

10. 要特别注意可能变化的各种因素。一旦有变化，力求实现起来尽可能简单。

§1.3.1 画流程图

画流程图是所有程序结构设计方法中最为人们所熟悉的一种方法。

一个程序按功能可分为若干部分，通过流程图把具有一定功能的各个部分有机地联系起来。从而，可以使人们迅速地抓住程序的基本线索，对全局有个完整的了解。这样，容易发现设计思想上的错误和矛盾，也便于找出解决问题的途径。因此，画流程图是程序结构设计时的一种重要手段。有了流程图，对于较大型的程序，便于分成若干个模块，分别进行设计，最后合在一起联调。较大型的程序要有总的流程图，这种流程图可以画得粗一些，侧重于反映各模块之间的相互联系；也要有局部的流程图，用以反映各个功能模块的具体实现方案。既使对于简单的问题可能无需把流程图画在纸上，但是在程序设计者的头脑里，程序的线条，结构、联系必须是清楚的。

流程图中值得注意的是分支结构和循环结构。分支结构反映了不同情况下的不同处理方式，一般是以输入的原始数据或者运算中的结果作为判断的根据。循环结构可以看作是分支结构的一种特殊情况，它是要返回去重复执行已经执行过的程序段，因为程序中常常需要对一个局部过程重复执行。可见，用循环结构可以实现的流程，用分支结构同样可以实现，反之则不然。

流程图法的优点是：

1. 它是一种图解表示方法。画成图的形式比较直观和形象，比用文字和数学表达式来描述程序的基本思路要直观得多。程序设计者可以借以直接了解整个系统及各部分之间的相互关系，逻辑错误和矛盾往往一目了然。

2. 有现成的标准符号，因而流程图形式得到广泛承认。

3. 流程图能被很多没有程序设计基础的人理解。

4. 利用流程图可把整个设计分成若干子任务，通过检查流程图就能了解总的进展情况。

5. 流程图反映出了操作顺序，因而有助于分析导致错误的原因，确定错误源的位置。

6. 流程图被广泛用于程序设计以外的其它领域。

7. 有许多现成的工具有助于编制流程图，其中包括程序员的标准模板和自动绘图程序包。

由于有上述优点，流程图法无疑还将继续被广泛使用。但是，流程图法作为一种程序结构的设计方法，也有如下一些缺点：

1. 除了比较简单的情况以外，当程序比较复杂时，流程图不容易设计、修改。因此，对于许多程序员来说，画流程图与其说是一种程序结构设计方法，倒不如说是累赘。

2. 没有查错或测试流程图的简易方法。

3. 往往要在如下两方面进行折衷：一方面为使流程图有用而需要详细，另一方面又要使流程图不致变得过于繁琐。

4. 流程图只表示出程序的组织结构，表示不出数据的组织结构或输入、输出模块的结构。当然，有时可另外画出数据流程图，用以表示程序如何处理某个特定类型的数据，以及某个特定类型的数据如何通过系统，从程序的一端通向另一端。这类数据流程图在查错和测试时是很有用的，因为对特殊类型数据的处理往往容易发生错误。

5. 流程图对硬件或定时问题没有帮助，也不能就这方面的问题可能发生在什么地方提供线索。

6. 流程图允许高度的非结构式设计，这样在流程图中可能出现许多环路和返回箭头，而遍及流程图上的返回箭头和环路是同先进的结构式设计相对立的。

综上所述，画流程图是一种很有用的程序结构设计方法。由于流程图具有标准形式，并易于非程序设计人员所理解，所以它作为程序资料是很有用的。但是，随着计算机及其应用的不断发展，程序日趋庞大和复杂，流程图也就越来越大，因而流程图作为一种工具，也就有其一定的局限性。比如，程序员很难对一个大而详细的流程图进行查错，而且流程图往往比程序本身更难于设计。

§1.3.2 模块化程序设计

显然，如何把一个大型程序分成若干子任务或模块，流程图会在一定程度上给予启发。然而人们终于认识到，对于大型而复杂的程序，流程图法可能不再是一种令人满意的设计方法了，必须把程序按照一些精心设计和研究过的模式进行搭配和组装。把整个程序分成若干功能独立的子任务或模块，称为模块化程序设计。采用模块化设计中所面临的两个关键问题是：如何把程序分解成模块，以及如何把各个模块装配在一起，以形成一个良好的工作系统。

模块化程序设计的优点是：

1. 单个模块要比一个完整程序易于编写、查错和测试。

2. 一个模块可能应用在很多地方或其它程序中，特别是对于比较通用的模块，可以建立一个标准模块库。

3. 模块化设计便于若干程序员分担任务，并可利用以前写好的程序。

4. 当某个局部需要改变时，可以只修改一个模块，而不会影响到其它模块。

5. 往往可以把错误找出来，然后查明它所在的那一个模块，以便修改。

6. 模块化程序设计能给人以这样的概念：已经取得了多大进展，还留有多少工作要做。

可见，模块化程序设计有很多突出的优点，以至人们往往忽略了它的缺点。这些缺是：

1. 把各个模块装配在一起可能十分困难，特别是当这些模块由不同程序员编写时更是如此。

2. 对模块进行文件编制时必须十分仔细，因为它们可能影响到程序的其它部分。比如，影响用于所有模块的数据结构。

3. 单独地检查和测试模块很困难。因为被调试模块中用到的数据可能由别的模块产生而且还有些模块可能要使用这个模块所得出的结果。因此，可能需要编写一些专用程序，以便为被测试的模块提供数据。显然，这些专用程序需要另行设计，而这种额外的工作并不是系统本身所需要的，也不会为系统增加任何功能。

4. 有些程序很难模块化。如果模块的划分这一步做得不好，装配起来就会很困难，因为一个模块的错误和更改往往会涉及到好几个模块。

5. 模块结构的程序往往需要额外的执行时间和存贮空间，因为几个模块中有些功能是重复的。

因此，虽然模块化程序设计为我们提供了一种有用的程序设计方法，但它也确实存在若干缺点。

采用模块化方法时，应考虑的主要原则是限制各个模块共享的信息量，把易于发生变化的设计判决限制到一个模块中，以尽量减少模块相互之间的访问。

虽然还没有任何经过理论和实践证明的、系统化的程序模块化方法，但以下一些原则无疑将对我们有一定的指导作用：

1. 与公用数据有关的模块应属于同一个总模块。

2. 如果两个模块中，第一个模块使用第二个模块或取决于第二个模块，但反过来不成立，则应把它们分开。

3. 如果一个模块被一个以上的其它模块所利用，则该模块应是与那些模块不同的另一个总模块的一部分。

4. 如果两个模块中，其中第一个模块用于其它很多模块，而第二个模块仅用于少数几个别的模块，这两个模块就应该是分开的。

5. 如果两个模块的利用率差别很大，则它们应分属于不同模块。

6. 在结构或编排上有联系的数据应包含在一个模块之中。

总之，如果你遵守如下一些规则，模块化程序设计将会很有用处：

1. 一个模块通常以包含 20 到 50 个语句为宜。因为过短的模块往往造成时间上的浪费，而过长的模块则缺乏通用性，并且可能难以装配。

2. 力求使模块具有一定的通用性。这样就可使一个模块在改变某些参数的情况下，应用在许多不同的场合。当然，对于重大的改变，则应由单独的模块来处理。

3. 要对延时程序、显示处理程序、键盘处理程序、其它 I/O程序等模块进行精心

设计。因为这些程序模块在其它设计项目或现有程序的很多不同地方都是有用的。

4. 力求使各模块尽可能地截然分开，且逻辑上互相独立。应限制两个模块之间的信息流，并要在一个单一模块中实现各设计判决。

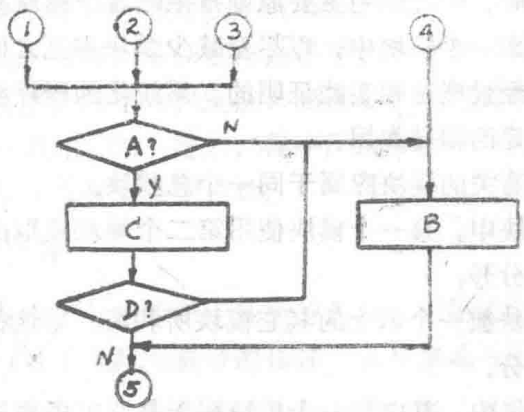
5. 对于比较简单的任务，不要照搬模块化的方法。

随着计算机应用的不断扩大，软件的规模也日趋庞大和复杂，研制大型软件的工作量是相当大的，有的需要数百以至数千人年才能完成。例如，著名的反洲际导弹系统SAFEGUARD中包含各种指令共2145000条。软件人员的平均生产率按每人每年418条计算，则该系统中软件的工作量超过五千人年。因此，由于编写大型、复杂软件的需要，人们越来越认识到程序设计实行模块化是一种必然的趋势。

§1.3.3 结构式程序设计

如何使各个模块截然分开并防止它们的相互影响呢？如何编写一个操作顺序清楚的程序使你便于检查和纠正错误呢？最好的方法是使用所谓“结构式程序设计”的方法。所谓结构式程序设计，是使设计出的程序的每一部分由若干环节组成，其中每个环节包含一个有限结构集，而每种结构只有一个入口和一个出口。

如下图一所示，显然是一个非结构式程序的流程图。如果在模块B中发生某个错误，该错误有5个可能的错误源。这就使我们不但必须检查每一个操作顺序，而且还必须保证，纠正该错误所做的任何修改不得影响其它任一操作顺序的正确性。其结果常常是使调试变得很不确定，往往顾此失彼。



图一 非结构式程序流程图

解决上述问题的办法是建立操作顺序清楚的程序。操作顺序清楚的程序应该是单入口、单出口模块。允许的基本结构是：

一、顺序结构

这是一种线性结构，其中的语句或结构均被顺序执行。比如在如下序列

S1

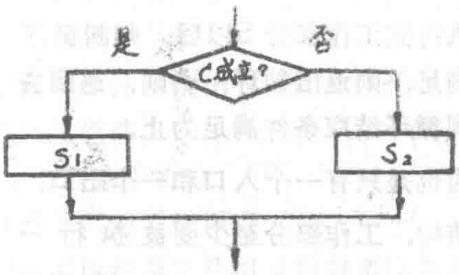
S2

S3

中，计算机先执行 S1，然后执行 S2，再执行 S3。其中 S1、S2、S3 可以是一条指令，也可以是一个程序段或模块。

二、分支结构

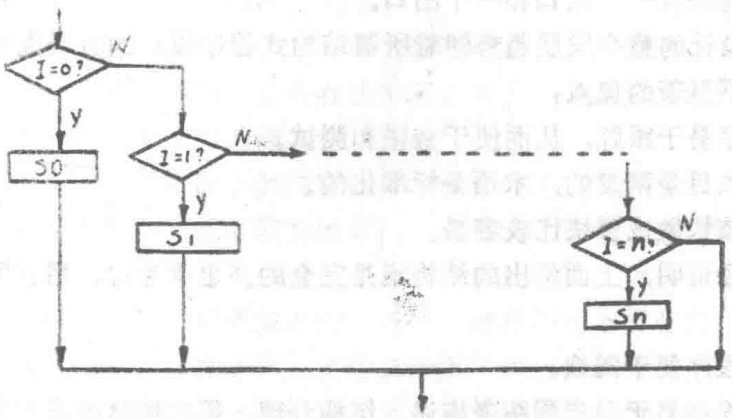
如下图二所示，是一种分支结构的流程图。其中 C 是条件，S1、S2 是语句（指令）或语句序列。



图二　分支结构流程图

计算机执行这种结构时，以给定的条件 C 为依据，若条件满足，则执行 S1，若条件不满足，则执行 S2。图二表示出了这种结构的逻辑。从图可见，在这种结构中，只有一个入口和一个出口；除了通过结构本身以外，再没有进入或离开分支部分的其它通道。

值得一提的是分支结构可以嵌套，即一个分支结构又由分支结构组成，形成多级分支结构。这种嵌套的分支结构称为选择结构，如下图三所示。



图三　选择结构流程图

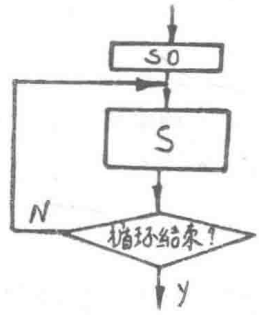
这种结构虽然不象顺序结构、简单分支结构和循环结构那样是基本结构，但也用得相当普遍。这种选择结构是以下标变量工作为选择某个语句或语句序列的依据。如果 I= 0，则执行 S0；如果 I= 1，则执行 S1，等等。其中 S0，S1，……，Sn 为语句或语句序列，共有 n+1 个。对于这种结构，只执行 n+1 个语句或语句序列中的一个。

执行之后，控制便转到选择结构后面的下一个语句或语句序列。如果 I>n，则不执行选择结构中的任何语句，控制直接转到选择结构之后的下一个语句或语句序列。它仍然满足一个入口和一个出口的原则，除了通过选择结构本身以外，再没有进入或离开选择结构的其它通道。

三、循环结构

循环结构如下图四所示。其中 SO 为循环的预置语句；S 为循环的工作部分，可以是语句或语句序列。

执行这种结构时，执行完工作部分 S以后，判断循环结束条件是否满足。若满足，则退出循环；否则，返回去再执行工作部分 S，直到循环结束条件满足为止。



图四　循环结构流程图

显然，这种循环结构也是只有一个入口和一个出口。

对于上述这种循环结构，工作部分至少要被执行一次。因为在这种结构中，是先执行工作部分，再判断循环结束条件。还有一种循环结构是先判断循环结束条件。再执行工作部分。对于这种循环结构，如果一开始就满足循环结束条件，则工作部分将一次都不执行。

可见，结构式程序设计具有如下三个主要特点：

1．只允许三种基本结构。有时，为了方便也可以有少量辅助结构，而辅助结构无非是基本结构的扩展和变换。比如，可以把选择结构作为一种辅助结构，但它不过是典型分支结构的扩展而已。

2．结构可以嵌套达到任意复杂程度，以至任何程序又可包含任何结构。

3．每个结构只有一个入口和一个出口。

当前，程序设计的整个发展趋势朝着所谓结构式程序设计的方向发展，是因为结构式程序设计有以下显著的优点：

1．操作顺序易于跟踪，从而便于查错和测试。

2．结构的数目是限定的，术语是标准化的。

3．把各种结构做成模块比较容易。

4.理论上已经证明，上面给出的结构集是完全的。也就是说，用三种基本结构能写出所有程序。

5．结构化程序便于阅读。

6．结构化程序易于用流程图来描述。结构合理、层次清楚的流程图，不但能减少编码错误，而且还为以后的维护工作带来很大的方便。

7．结构化程序设计可提高程序员的工作效率。

结构式程序设计比起模块化程序设计来说，程序中必须遵守更多的规则。从而将产生更为系统、更有条理的程序。

结构式程序设计虽然是一种较为先进的程序设计方法，但它也有下列缺点：

1．目前只有为数不多的几种高级语言（如 PASCAL、PL/M等）能直接使用这

种结构。而在汇编语言中使用这种结构尚需做一定变换。

2. 结构化程序比起非结构化程序执行速度慢，而且要用较多的存贮单元。

3. 把结构限度定为三种基本形式，就使某些任务很难执行。结构的完备性只意味着可用它们编制所有程序，并不意味着用它们能很方便地编制出高质量的程序。

4. 标准结构往往很混乱，例如，嵌套的分支结构和循环结构可能很难阅读。

5. 结构化程序只考虑结构操作的顺序而不考虑数据流，因而不适于处理数据。

6. 不少程序员认为标准结构不灵活，受约束太多，因而不习惯于结构式程序设计。

总之，结构式程序设计是程序设计系统化的一种较好的方法。在下列几种场合下使用高级结构程序设计语言更显其优越性：

1. 超过1000条以上指令的较大型程序。

2. 对存贮容量的限制并不苛刻的一些场合。

3. 软件开发费用，特别是测试和查错费用成为重要因素的那些小批量应用。

4. 涉及字符串处理，过程控制或其它算法的一些场合，而不是单纯的位处理的场合。

可以预料，存贮器的价格会继续下降，而计算机程序的平均容量和软件开发费用将会不断增加。因而，高级结构程序设计语言中的一些方法将变得更加有用，因为用这些方法虽然要占用较多的存贮空间，但却降低了大型程序的软件开发费用。

应该指出，并不因为结构式程序设计的概念通常是用高级语言来表达的，就意味着结构式程序设计不适用于汇编语言程序设计。恰恰相反，正是由于程序员在用汇编语言进行程序设计时有极大的灵活性，因此更需要结构式程序设计所提供的结构化概念，建立具有单入口和单出口的模块，使用简单的控制结构，并尽量减小每个模块的复杂程度。所有这些必将产生更为有效的汇编语言程序。

综上所述，结构式程序设计方法给程序设计带来了一定的约束。它限定了结构的类型，规定了单入口和单出口结构，这将有助于使本来可能是混乱的过程更有条理。结构化的程序还有助于查错、测试和编制文件。

结构式程序设计要求程序员不但要对任务作出恰当的定义和分析，而且要仔细地做出逻辑安排，这对程序员来说是繁琐而困难的，但有助于设计出结构清楚的工作程序。此外，对于结构化程序，有时很难分辨一个结构在何处结束，而另一个结构又从何处开始，特别是对于嵌套的结构，则更是如此。因此，每种结构都要有自己的结束符。

在你使用结构式程序设计方法时，要注意以下一些规则：

1. 首先编写出一个基本程序流程图，来帮助你确定程序的逻辑。

2. 以顺序结构、分支结构、循环结构为基础。这三种基本结构组成一个完备的结构集，即任何程序都能利用这三种结构来编写。

3. 每一层相对前一层要缩进几格，使语句的层次更加清楚。

4. 要突出简单性和可阅读性。使用有意义的名字对程序加以注释。要知道，简单性和可阅读性是软件设计的准则之一。

5. 要检查逻辑关系，对特殊条件和一些典型情况进行测试。要记住，在某一层次

发生的任何逻辑错误都将会在以后给你带来困难。

§1.3.4 自顶向下设计

如前所述，我们希望把一个大任务划分成若干子任务或模块。但如何分别地检查这些子任务，并把它们组合在一起形成一个系统呢？通常采用“自底向上设计”的方法，对各个子任务分别进行检查，并把它们组合在一起。然而，这种方法不但要求额外的测试和查错工作，而且整个组装任务必须留到最后进行。我们需要的是在程序运行的实际环境下进行查错和测试，以及使系统的组装也使用模块化的方法。这种方法称为“自顶向下设计”。所谓自顶向下设计，是从总体要求出发，采取逐步分解，逐步求精的办法，直至整个系统都得到实际的程序为止。

自顶向下设计的过程是：

1. 首先编写总管理程序。在编写总管理程序时，用程序“根”来代替尚未确定的子程序（子任务）。这些程序根是暂时性的程序，它可以记录输入，为选定的测试问题提供答案，等等。然后测试这个总管理程序，以便检查它的逻辑是否正确。

2. 扩展程序根。每个程序根常常又包含若干子任务，这时再用程序根表示这些子任务，以便扩展、查错和测试这一层的程序根，得出实际的程序。

重复第2步的扩展、查错和测试过程直到所有程序根都被实际程序取代为止。要注意，必须对每一层进行测试和组装，而不是全部集中在最后进行。

自顶向下设计采用模块化程序设计方法，同时也和结构式程序设计相容。

自顶向下设计的缺点是：

1. 总体设计不一定能同系统硬件很好地结合。
2. 它不一定能充分利用现有软件。
3. 程序根可能难于编写，特别是当它们必须在几个地方都能正确工作时。
4. 自顶向下设计可能得不到通用模块。
5. 顶层出现的错误可能造成严重的后果。相反地，自底向上设计中的错误往往只局限于某一个特定模块。

自顶向下设计方法有较大的通用性。对大型软件的设计，采用自顶向下设计能明显地提高程序员的工作效率。不过，自顶向下设计不一定能编出最有效的程序。此外，在用自顶向下设计方法可能会导至大量额外工作的场合，通常采用一定的自底向上的设计方法。

自顶向下设计为问题的定义或流程图扩展到编写实际程序的过程提供了一种系统化方法。它同结构式程序设计一起，形成了一套完整的设计方法。

如同结构式程序设计一样，自顶向下设计并不简单。要正确地运用自顶向下设计方法，设计者必须仔细地分析任务，进行问题的定义，确定算法，并对每一层系统地进行处理。为帮助读者正确地应用自顶向下设计，这里介绍并向你推荐如下一些基本方法：

1. 从基本流程图开始。
2. 采用结构式程序设计方法中所规定的结构。
3. 尽可能使程序根完整而又独立。
4. 精确地确定每个程序根扩展的所有可能结果。

5. 把每个程序根都扩展一层，一步不要试图做得太多。

6. 每扩展一个程序根后，就进行查错和测试，不要企图一次检查整个一层。

7. 仔细而系统地检查每一层。

8. 仔细注意公共的任务和数据结构。

9. 对硬件能做些什么要心中有数，对于看来需要自底向上设计的地方，就要果断地进行自底向上设计。

从前面的分析可见，软件设计可分为总体设计和详细设计。在总体设计阶段要对程序的结构进行设计。程序结构的设计可用前述的几种方法系统地规定程序的逻辑，构造出系统的模块结构，并将它们编成文件。流程图方法使人们对整个系统有直观而完整的了解。模块化程序设计要求把整个程序划分成小的独立的模块。结构式程序设计提供了规定这些模块逻辑的系统化方法。而自顶向下设计则是组装和测试它们的系统化方法。虽然这些方法都是原则性的，但它们确实提供了一套统一的程序设计方法。特别是自提出软件开发“工程化”，软件产品“商品化”的概念以来，模块化程序设计方法，结构式程序设计法和自顶向下设计法成为了设计软件的一些主要方法。由于上述这几种方法各有所长，又都有其局限性，因此，在实际的程序结构设计中，往往是把几种方法综合起来，互为补充，以求取得最好的设计效果。

§1.4 编码

通过前述的程序结构设计之后，已经解决了“做什么”，“怎么做”的问题，得出了软件设计的各种文件，以此指导以后各阶段的工作。这时思路已经清楚，下面的任务就是采用某种编码的途径及策略，根据具体的实现环境，诸如具体的机器、所采用的程序设计语言、环境设施等，将设计转换成可运行的程序，并产生一系列相应的文件。

实现编码的途径主要有两条（1）分步实现方式。设计者根据设计文件先对单个模块进行编码，然后把这些单个模块组成完整的系统。（2）“多稿”方式。设计者根据设计文件一次写出整个系统的源程序，然后再进行一系列的“校订”工作。实现的总体策略也主要有两种：自顶向下和自底向上实现。

无论采用哪种途径和策略，编写程序力求简单明了，层次清楚，程序运行时间短，占用空间少。如果解决一个问题的两个程序，一个运算快，省内存，一个速度慢、费内存，准优准劣，不言而喻。但是在程序设计中，速度和容量常常是一对矛盾。这时，就要全面权衡利弊，根据任务的要求和实现的可能决定取舍，或者进行折衷。

§1.5 查错和测试

即使用了诸如模块化程序设计、结构式程序设计、自顶向下设计等方法，编出的程序难免会有这样或那样的问题和错误，必须要对件软进行查错和测试，即软件的确认。而软件的确认是软件研制过程中最花时间的阶段，据统计，整个软件开发周期中45%以上的人力用于软件确认阶段。

对于软件的查错、测试有三种方法：人工分析、静态分析和动态测试。

人工分析即所谓“桌前检查”，它主要采取结构预走、审查及复查等方式来分析、检查程序。因此，人工分析完全由人来检查代码，这是一种传统的分析程序的方法。

静态分析是使用计算机对源代码进行分析，检查其错误或异常情况，它并不是程序