应用软件开发技术

史济民　　主编

電子工業出版社

应用软件开发技术

史济民　主编

电子工业出版社

1993

(京)新登字 055 号

内 容 提 要

本书约 35 万字，分上下两篇，上篇为软件开发技术，下篇为软件开发环境。全书共包括结构化程序设计、工程化的软件开发技术、软件测试与质量保证、用户界面、操作系统及其界面、软件工程环境六章，系统地阐述了软件开发的基础知识，对广大计算机用户及软件开发人员来说，是一份有较大实用价值的参考资料。

本书内容系统性强，讲解清晰易懂，亦可作为大专院校计算机专业的教科书或参考书。

应用软件开发技术

史济民主编

责任编辑　高平

*

电子工业出版社出版(北京市万寿路)

电子工业出版社发行　各地新华书店经销

农科院印刷厂印刷

*

开本：787×1092 毫米 1/16　印张：14

1993 年 9 月第 1 版　1993 年 9 月第 1 次印刷

印数：1—6000 册　.定价：9.8 元

书号：ISBN 7－5053－2249－4/TP・609

前　言

在非计算机专业大学生和在职干部中，懂得一、二门计算机语言的人现在越来越多了。其中有些人编写过一些应用程序，但是总感到知识不足，希望进一步学习提高。可惜的是，各种高级语言的入门性程序设计教材虽然随处可见，但以应用软件开发为主题，包容有关的软件基础知识，适于非计算机专业的读者用作程序设计后续课的教材或自学读物，却不易找到。

近二年来，编者在华东化工学院和上海工程技术大学（纺织学院）的支持下，在非计算机专业学生中连续四次开设了《应用软件开发技术》课，其中大学本科生和硕士研究生各有两次。本书就是在为该课自编讲义的基础上修改而成的。学生普遍反映这一教材理论联系实际，可读性强，使他们在短短一学期内学到了许多围绕软件开发的实用知识。他们称赞这是一本令人“受益匪浅”的提高软件开发能力的入门书。

本书由“软件开发技术”与“软件开发环境”上下两篇组成，上篇含“结构化程序设计”、“工程化的软件开发技术”、“软件测试与质量保证”等3章，下篇含“用户界面”、“操作系统及其界面”和“软件工程环境”等3章。从表面上看，本书与前几年出版的《软件技术基础》或《程序设计技术》一类教材相似，都包含软件工程、算法设计、数据结构、操作系统等多个知识单元。但后一类教材大都是多中心的，一个知识单元构成一个中心，而本书只有一个中心——应用软件开发。全书两篇6章，分别从技术和环境两个侧面，紧紧围绕“软件开发”这一主题展开。犹如“众星拱月”，每一颗星的存在都为了突出月亮，对月亮起到烘托和陪衬的作用。由于各章内容都按照上述主题精心选择，主次分明，结构紧凑，全书仅有30余万字。在高校可用作一学期课程的教材，仅用36（课堂）学时左右就可讲完。

本书内容符合于教委高等学校工科计算机基础课程教学指导委员为《软件开发技术》课制定的基本要求（征求意见稿），可作为高校《程序设计》的直接后续课，供非计算机专业大学生或研究生用作教材，也可供从事计算机应用仅学过一门语言的在职干部自学，或作培训班教材使用。

本书由史济民主编，李昌武参加编写第四章和第一章第1—4节，其余章节由史济民编写。张兆奎、乔沛荣、姚品新等老师热情支持编者在上述两校试开这一新课。徐安东老师阅读了部分书稿，并提出了有益的意见。编者还要感谢国家教委工科计算机基础课程教学指导委员会，这一新课试开不久，课委会就决定将《软件开发技术》列为工科非计算机专业5门计算机基础课之一，使我们受到很大鼓舞。主任委员会胡正家教授、副主任委员谭浩强教授和其他许多委员，对这门课的内容发表了指导性的意见。谭浩强教授还在百忙中审阅了全书，提出许多宝贵的意见。值此机会，编者向他们，以及一切在本书编写过程中给予帮助的同志表示衷心的感谢！

这是一本新型的、用于软件开发入门的教材。在结构安排、内容取舍上都颇费斟酌。我们的目的，是用一门课、一本书，用较短的时间，先把初学者带进门。入门之后，深造就不难了。限于水平，加上是新课，肯定有不完善甚至错误的地方，诚恳希望读者不吝指正。

编者

1992年6月于上海华东化工学院

关于本书内容的说明

本书是以软件工程为中心，包含多种有关的软件基础知识，围绕“应用软件开发”这一主题编写的一本新型教材，由“软件开发技术”和“软件开发环境”上、下两篇组成。

上篇以传统的瀑布开发模型为主线，用三章讨论了软件开发过程中“分析—设计—编码—测试”等各个阶段的活动。第一章从“结构化程序设计”开始，先讲读者在小程序设计中已经学过的详细设计与编码，强调“清晰重于效率”、“设计先于编码”等原则和“逐步细化”的设计方法。然后按照 Wirth 的“程序＝算法＋数据结构”的公式，顺理成章地介绍了算法与数据结构的基础知识，包括算法的复杂性、算法设计的基本方法，数据结构的类型和作用，以及表、树等常用数据结构的基本操作与应用等。D. Gries 说过，“只有学会有效地编写小程序的人，才能学习有效地编写大程序。”学好第一章既可以巩固初级程序设计课中学过的知识，提高掌握小程序设计能力，同时也为读者学习后续的、开发大程序的方法和技术打下基础。

第二章承上启下，首先引入软件生存期的概念。然后从软件开发模型讲到系统开发方法，着重介绍了在软件工程中发展比较成熟、至今仍流行最广的结构化分析与设计技术。为了帮助读者理解这些方法所蕴含的原则，本章追踪讨论了一个具体的实例—教材购销系统。通过对这一软件系统的分析和设计，向读者展示了软件开发的实际步骤，避免了软件工程课本中常见的那种“原则讲得多、应用无实例”，令初学者感到抽象不易理解的弊病。

第三章主要讨论软件测试技术。重点放在测试用例的设计方法，纠错的策略与技术，以及对多模块软件进行层次测试的内容与步骤。与前章相似，本章也充分运用“结合实例讲解”，先后使用了 10 余个有一定代表性的例子。章末用一个整节简介了软件开发全过程的质量保证活动，着重讨论了“测试”与“评审”、“确认”与“验证”之间相辅相成的关系，使读者对软件的质量保证有一完整的概念。

软件开发离不开环境。具有友好的用户界面、由集成工具和强大的系统软件所支持的软件开发环境，不仅能明显地提高软件的生产率，而且正在潜移默化地改变着软件的生产方式。作为入门书，本书拿出大约 2/5 的篇幅介绍与环境有关的知识，以期引起读者对改善环境的充分重视。

对环境知识的介绍从下篇第四章开始。该章重点叙述了被称为“三大友好技术”的窗口、菜单和联机帮助等“用户界面”技术和一些简易的实现方法。章末以 Windows3.0 为例，介绍了一个受到用户欢迎的图形用户界面的实例。

如果说用户界面处于环境的顶层，则操作系统构成了一切环境的基础。它支持软件开发工具，支持其它系统软件，也直接支持用户和用户程序。第五章正是从这个角度，向读者介绍了这一最重要的系统软件。众所周知，操作系统具有资源管理和支持用户二大职能。为了突出后一个职能，本章用近二万字介绍了操作系统的用户界面，包括键盘命令与系统功能调用，并简介了一些流行的 DOS 界面工具，如 PCTools 与 Windows 3.0 等。对于资源管理，本书采取了结合实际操作系统阐明重要概念的方法，既讲清了概念，又介绍了 PC-DOS、CC－DOS、UNIX 等常用操作系统。

第六章也是本书的最后一章，综述了开发环境的组成与类型。通过对环境的抽象，引入了“理想环境模型”的概念，从软件开发者的角度说明了开发模型、开发方法和开发工具三者之间的关系。章末还对应用生成器与第四代语言作了简单介绍。

此为试读，需要完整PDF请访问：www.ertongbook.com

由以上的简介可知，本书涉及软件工程、操作系统、算法设计、数据结构等多个知识单元，向读者提供了开发应用软件所需要的许多软件基础知识。但是，无论自学或课堂讲解，请读者牢记学习本书的目的，紧紧扣住“软件开发”这一个主题。正如本书的结构安排所显示，虽然各章有自己的主题，但在全书的总主题下已经融合为一个整体，不再是一个个独立的知识模块。

本书可用作高校非计算机专业《软件技术基础》课的教材，总学时 50－60，其中讲课 36－40 学时，实践 14－20 学时。讲课时间的分配，上篇一至三章各占 8－10 学时，下篇三章可用 8－10 学时介绍其主要内容，其中四、六两章各用 2 学时，第五章用 4－6 学时。上、下篇也可以穿插讲解，例如采用 1－4－2－5－3－6 章或 1－4－2－3－5－6 章的讲课顺序。实践环节包括上机（含一、三、四、五章）、习题课与课程设计，最好安排一个数百至数千行的程序，让学生从头至尾作一次软件开发练习。如果只讲上篇，下篇改成自学或课外讲座，也可用于 40 学时的课程。

要再次重申，小程序设计是大程序设计的基础。编写小程序的经验丰富，愈容易接受软件工程的思想与方法。但是，如果缺乏软件工程和算法、数据结构及操作系统的有关知识，即使语言学得再多，也决不会成为一名优秀的程序员。

目　录

上　篇　软件开发技术

下 篇 软件开发环境

第一章　结构化程序设计

结构化程序设计是荷兰学者 Dijkstra 教授所创始，通过程序的结构化，来提高程序清晰度和可靠性的程序设计方法。它最初用于指导程序的编码和详细设计。软件工程兴起后，软件的开发一般要经过分析、设计、编码、测试等阶段。结构化的思想，也从编码阶段扩展到整个软件开发的全过程，成为软件工程的重要指导原则之一。

在美国计算机协会“计算机科学课程委员会”制订的＜'77 教程＞中，十分重视结构化程序设计的教学。它强调指出，结构化的思想应该渗透到一切软件课程之中。在国内，80 年代中期以后出版的程序设计教材大都加进了有关结构化程序设计的内容。本章的目的，在于通过介绍结构化的思想与方法，使学过初级程序设计课程的读者温故知新，巩固提高，起到承上启下的作用，为学习后续各章，尤其是软件的分析与设计奠定基础。原来已经熟悉本章内容的读者，可省去不学。

1.1　程序设计风格的演变

1.1.1　关于 GOTO 语句的争论

1968 年 11 月，美国＜ACM 通讯＞发表了荷兰学者 E. W. Dijkstra 教授写给编辑部的一封信，标题是

＜GOTO 语句是有害的＞[1]

在这封信中，Dijkstra 阐明了 GOTO 语句对程序清晰性的危害，然后说，他经过对大量程序的观察，发现一个程序包含的 GOTO 语句越多，其质量往往越差，因此，他认为应该在一切高级语言中取消 GOTO 语句。这封信发表后，在计算机学术界激起了强烈的反响，导致了一场长达数年的争论。

争论的焦点，是 GOTO 语句应不应该废除？持赞成意见的一方认为，GOTO 语句对程序清晰性有很大的破坏作用。凡是使用 GOTO 较多的程序，其控制流时而 GOTO 向前，时而 GOTO 向后，常常使程序变得很难理解，从而增加查错和维护的困难，降低程序的可靠性。他们把这类难懂的程序称为“耗子窝(rat's nest)”程序或“细面条(bowl of spaghetti，简称 BS，直译为一碗通心面)”式程序，用以比喻程序中的语句纵横交错，理不清头绪的混乱状况，有人还发表文章[2]，论述不带 GOTO 语句进行程序设计的好处，被人们称为“无 GOTO 的程序设计(GOTOless programming)”。

但是，以＜计算机程序设计技术＞系列丛书的作者 D. E. Knuth 为代表的另一方则认为，GOTO 语句虽然存在破坏清晰性的缺点，却不应该完全禁止。其理由是：GOTO 语句概念简明，使用方便，在某些情况下，保留 GOTO 语句反能使写出的程序更加简洁。因此，他们认为完

此为试读，需要完整PDF请访问：www.ertongbook.com

全取消 GOTO 语句是“因噎废食”，至少也是不明智的。

1974 年，D. E. Knuth 发表了题为＜带有 GOTO 语句的结构化程序设计＞[3]的文章，指出，多用 GOTO 语句确实有害，但完全废除也不可取，主张限制使用。文章的题目就清楚地表明，作者赞成由 Dijkstra 倡导的以提高程序清晰性为目标的结构化程序设计，但结构化程序仍可以使用 GOTO 语句，不必对 GOTO 采取“一概排斥”的态度。这些意见终于为大家所接受，便使这场持续多年的争论平息了下来。

围绕 GOTO 语句的这场争论吸引了众多的知名学者参加，有其深刻的原因。随着计算机应用在 60 年代的发展，软件的数量与规模均不断增长，软件的成本急剧上升，可靠性反而下降。许多有识之士因此发出了“软件危机”的警告。1968 年，在北大西洋公约组织(NATO)的一次以讨论软件危机为主的会议上，首先提出了“软件工程”的概念。正当大家为摆脱“软件危机”广泛进行探索的时候，Dijkstra 的信件提出了一个对程序清晰性及可靠性有重要关系的问题，理所当然地会引起广泛的注意。

1.1.2 从“效率第一”到“清晰第一”

这场争论的意义，远远超出了怎样对待 GOTO 语句本身。它促使许多人改变了单纯强调程序效率的观点，认识到在软件规模日益增长的新形势下，把程序编写得清晰易读所具有的重要意义。换言之，这场争论推动了程序设计风格从“效率第一”到“清晰第一”的重要转变。

在 Kernighan 与 Plauger 合著的＜程序设计风格要素＞[4]一书中，清楚地阐明了程序效率与清晰性的关系。在读到提高程序的执行速度时，该书建议的指导原则是：

“先求清楚后求快(Makeit clear before you make it faster)”

“保持程序简单以求快(Keep it simpie to make it faster)”。这说明，讲清晰并非不要效率，而是在清晰的前提下求取效率。如果二者发生矛盾，则该书建议的处理原则是：

“(把程序)写清楚，不要为“效率”牺牲清晰(Write clearly——don′t sacrifie clarity for“efficiency.”)”。

把以上几条综合起来，就形成著名的“清晰第一，效率第二”的原则。

需要指出，就程序的总体而言，清晰总是居于“第一位”的。但对于组成多模块程序的每一个模块来说，应该允许有少量的例外。具体地说，对构成多模块程序的绝大多数模块仍应坚持“清晰第一”，以确保整个程序的“可读性”，而对于其中(1)有严格实时要求的个别模块或(2)调用特别频繁，其执行时间在程序的总执行时间中占颇大比例的少数模块，允许以清晰为代价来换取效率，以确保程序满足总体的性能。例如，在用高级语言为大多数模块编码的同时对极少数有特殊实时要求的模块用汇编语言编码，就是一种常见的作法。

1.2 结构化程序设计

在提出废除 GOTO 语句以后不久，Dijkstra 又接着提出程序要实现结构化的主张，并把这一类程序设计称为结构化程序设计(structured programming)[5]。清晰第一，就是结构化程序设计所要求的程序设计风格。为便于说明，请读者先看一个简单的例子。

1.2.1 引例

〔例 1—1〕图 1.1 是用 FORTRAN 语言编码的一个辅程序。除去结尾的 return 语句外，在 10 条语句中共用了 5 条 GOTO 语句，其中 3 条为向前 GOTO，2 条为向后 GOTO。这一程序的可读性差，不易马上看清其用途，其实这一程序的功能并不复杂。图 1.2 是它的流程图，由图可知，如果 M 为 1，则 N 必然为 1，控制流将沿两个选择结构的左边分支执行，然后执行 SUBB，若 M 非 1，则先执行上一个选择结构的右边分支，然后根据 N 是否为 1（由调用它的程序决定）分别执行下一个选择结构的左或右分支，最后仍执行 SUBB。

```
      if (M.EQ.1) goto 100
      call SUBA
 30   if (N.EQ.1) goto 200
      J=J+1
 60   call SUBB
      goto 300
100   N=1
      goto 30
200   k=k+1
      goto 60
300   return
```

图 1.1　一个 FORTRAN 辅程序（编码之一）

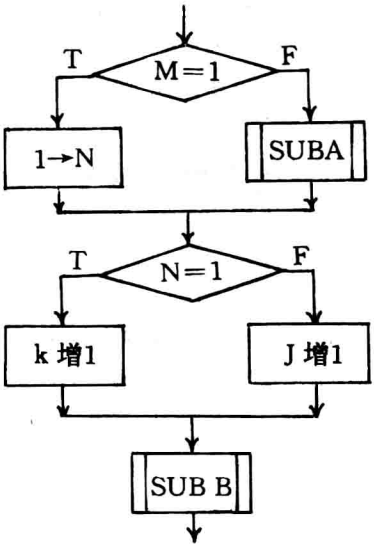


图 1.2　图 1.1 辅程序的流程图

稍加分析，便可把上述编码改进为图 1.3 所示的样子（编码之二）。该编码保留了 3 条向前 GOTO 语句，去掉了 2 条向后 GOTO 语句，程序可靠性也较编码之一有所提高。第三种编码示于图 1.4 中，它在编码中使用了在 FORTRAN77 以后的版本才提供的 if—then—else 语句。与前两种编码相比较，最后一种程序的结构最清楚，可读性也有显著提高。〔例 1—1 完〕

```
      if (M.EQ.1) goto 100
      call SUBA
      if (N.EQ.1) goto 200
      J=J+1
      goto 300
100   N=1
200   k=k+1
300   call SUBB
      return
```

图 1.3　编码之二

```
if (M.EQ.1) then
   N=1
 else
   call SUBA
 endif
if (N.EQ.1) then
   k=k+1
 else
   J=J+1
 endif
call SUBB
return
```

图 1.4　编码之三

这个简单的例子告诉我们，要想提高程序的清晰性，首先需要在程序的结构上下功夫。本

例在编码之三中用 if—then—else 语句来实现选择，两个选择控制结构之间的界限分明，整个辅程序与其流程图一一对应，阅读时一目了然。而前两种编码(之一与之二)均用逻辑 IF 语句实现选择，且插进了较多的 GOTO 语句，以致两个选择控制结构相互交叉，边界不易分清。通常把象编码三这样的程序称为结构化程序(structured program)，而把前两种缺乏清晰控制结构的程序称为非结构化程序(non—structured program)。

为了实现清晰第一的程序设计风格，结构化程序设计要求遵循以下的原则：

1. 保持控制流的直线性；

2. 保持控制流的局部性；

3. 坚持源程序的文档化。

随着软件工程方法的推广，人们越来越重视软件产品要对用户友好。于是对程序设计风格又增加一条新的指导原则，即

4. 满足运行工程学的要求。

从下一小节起，将对这 4 条指导原则逐一进行介绍。

1.2.2 控制流的直线性(linearity of control flow)

1. 程序的基本控制结构

早在 1946 年，Bohm 与 Jacopini 就在一篇文章中证明过[6]，任何程序段均可用顺序、选择和 do-while 型循环等 3 种控制结构或它们的组合来构成。一般称它们为基本的控制结构，其流程图如图 1.5 所示。

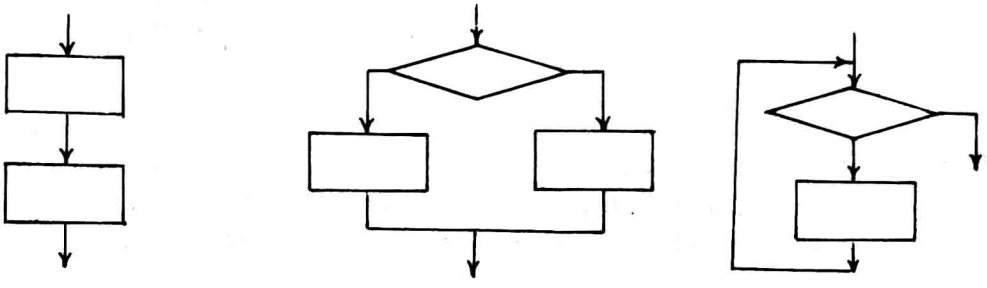


图 1.5 三种基本控制结构的流程图

表 1.1 列出了 PASCAL. True BASIC 与 C 三种常用语言所提供的控制结构及其实现语句。由表可见，这些语言提供的控制结构都多于 3 种基本控制结构。例如在选择结构中，除单边和双边选择外，还可提供 case、select、switch 等语句，用于直接实现多边选择。循环结构不但有 do-while 型，也有 do-until 型和 for 型。Dijkstra 曾建议只使用 3 种基本的控制结构来构成程序，以强调控制结构的标准化。现在把这种程序设计称为经典型(classical)结构化程序设计。而允许使用 3 种基本结构以外的控制结构的程序设计，则称为扩展型(extended)结构化程序设计。

2. 单入口/单出口的原则

其实，一个程序的结构是否清晰，不在于它总共用了多少种控制结构，而在于能否使整个

程序的控制流保持直线性。为此，H. D. Mills 主张每个控制结构只允许有一个入口和一个出口，这就是著名的单入口/单出口原则。如果组成一个程序(段)的所有控制结构都作到了这一点，则无论这个程序(段)有多么长，整个程序的控制流仍将保持直线性，使之清晰可读。Dijkstra 曾把这种程序比喻为妇女颈上戴的项链，因为项炼上的每个环节均与其相邻环节单线相连，能做到长而不乱。

在实际编程中，常常会遇到具有多入口或多出口的控制结构。这时，应采取措施将它们改造为单入口/单出口。下面请看几个例子。

〔例 1—2〕对具有多入口的循环结构的处理。

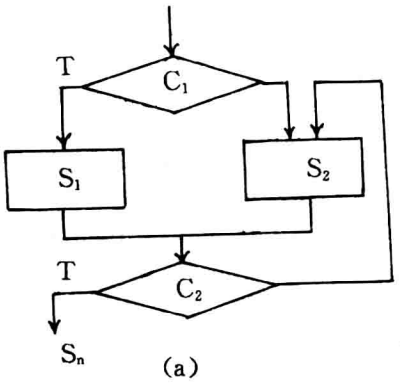
表 1.1　几种常用语言提供的控制结构及其实现语句

	PASCAL	True BASLC	C

选择结构	if＜条件＞then S
	if＜条件＞then S_1
	else S_2
	case＜表达式＞of
	表达式值 1 S_1
	…
	表达式值 i Si
	…
	end
		if＜条件＞then S
		if＜条件＞then S_1
		else S_2
		select case＜表达式＞
		case 表达式值 1 S_1
		…
		case 表达式值 i S_i
		…
		endselect
			if＜条件＞ S
			if＜条件＞S_1
			else S_2
			switch＜表达式＞
			{
			case 表达式值 1 S_1 break
			…
			case 表达式值 i S_i break
			…
			}

循环结构	while＜条件＞do s repeat S* until＜条件＞ for＜循环变量＞ =M to N do S	do[while/until＜条件＞] S* loop[while/until〈条件〉] for〈循环变量〉 =M to N step I S next〈循环变量〉	while〈条件＞S do S* while〈条件〉 for(〈表达式〉,〈表达式〉,〈表达式〉) S*

注：表中的 S* 表示允许有多条语句



(a)

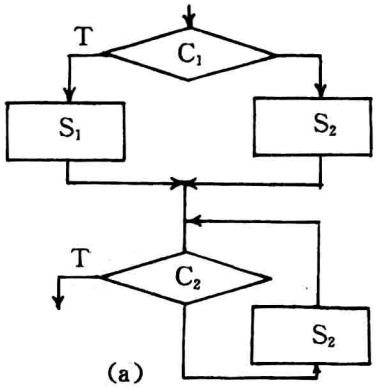
```
    if C1 then goto 30
10  S2
20  if C2 then goto 40
    goto 10
30  S1
    goto 20
40  Sn
```

(b)

图 1.6　多入口循环结构一例

图 1.6 显示了一个多入口循环结构的流程图(a)及其相应的程序(b)。图(a)中的 S_2 与 C_2 构成了一个循环，其循环体 S_2 有两个入口，分别来自 C_1 与 C_2。S_n。表示后续语句。

将 S_2 重复画一次，把与 C_1 相关的选择结构同与 C_2 相关的循环结构分开，就能把循环结构的双入口改造为单入口，如图 1.7 所示。有人把这种方法称为“重复环节法”[7]。改造成的程序，其可读性也大有提高。



if C_1 then S_1 else S_2

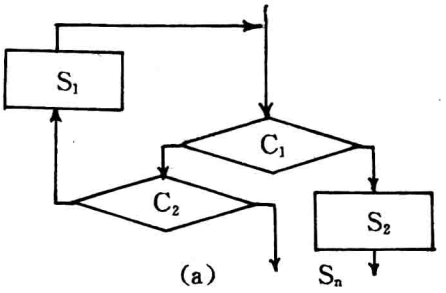
while (not C_2) do S_2

(b)

图 1.7　用重复环节法改造后的图 1.6 的程序

〔例 1—3〕对具有多出口循环结构的处理。

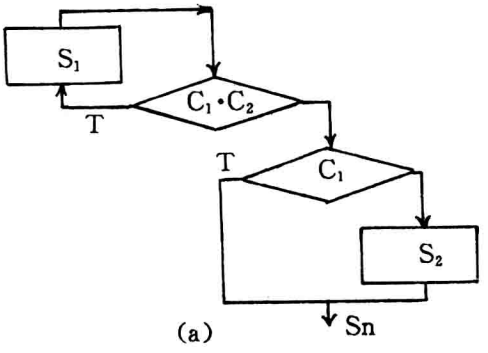
图 1.8 是带有双出口的循环法结构的一个例子。只有当 C_1 与 C_2 同时保持为“真”时，循环才能继续下去。若 C_1 非“真”，则执行 S_2 后退出，若 C_2 非“真”便直接 退出，从而构成了两个循环出口。




```
10    if c̄1 then goto 20
      if c̄2 then goto 30
      S1
      goto 10
20    S2
30    Sn        (b)
```

图 1.8 多出口循环结构一例

图 1.9 是改造成单出口循环后的程序，其可读性较原程序也有明显改进。由于它主要靠对程序逻辑的分析获得，所以可称为“逻辑分析法”[7]。



while(C_1 and C_2)do S_1

if (not C_1)then S_2

S_n

(b)

图 1.9　用逻辑分析法改造后的图 1.8 的程序

〔例 1—4〕对提高退出循环的一种处理方法。

还有一类常见的循环结构在循环体中包含有条件语句，一旦条件得到满足，便提前退出循环。于是在正常的循环出口之外，又添加了一个非正常的提前退出出口。图 1.10(a)便是这类程序的一个例子。

```
while C1 do
  begin
    Sa*
    if  C2  then goto 10
    Sb*
  end
10  Sn
```

(a)

```
EXIT:=false
while C1 and (not EXIT) do
  begin
    Sa
    if C2 then EXIT:=true;
    Sb*
  end
Sn
```

(b)

图 1.10　用标志变量法处理因提前循环引起的多出口

使用“标志变量法”可以把这类循环改造为单出口结构。其方法是，先设置一个标志变量(flag 本例中其变量名为 EXIT)，并赋值为“假”。一旦在某次循环中 C_2 得到满足，便将 EXIT 改“假”成“真”，使程序在下一轮循环中因“notEXIT”变“假”而中止循环。

顺便提一句，在〔例 1—2〕与〔例 1—3〕中，改造前后的程序都是等价的。而使用标志变量法改造得到的程序，与原程序不一定完全等价 。例如，图 1.10(a)的程序在 C_2 得到满足时将马上跳出循环，而图 1.10(b)中的程序在 C_2 满足后，还要等执行 S_b^* 后才退出循环。注意，在这里 * 表示重复，S^* 表示可能有多条语句。

有些高级语言还为提前退出循环提供了专用语句，如 C 语言中的 BREAK 语句，True BASIC 语言中 EXITDO、EXIT FOR 语句等。请看下面的例子。

〔例 1—5〕使用专用语句退出循环。

某程序的功能是：从健盘输入一个数，累加到总和中。当总和超过 1000 或输入为 0 时停止输入，并打印出总和。用 True BASIC 编写此程序，可能如图 1.11 所示。

```
let SUM=0
do
  input X
  if X=0 then exit do
  let SUM=SUM+X
  print SUM
loop until SUM>1000
```

图 1.11　TrueBASIC 程序中 exit do 语句的用法举例

3. 避免模糊/费解的结构

在使用单入口/单出口控制的同时，还应避免使用模糊或费解的结构。以下试举二例，以窥一斑。

〔例 1—6〕导致二义性的 then—if 结构。

在嵌套的选择结构中，then—if 结构常常会导致二义性。以图 1.12(a)中的程序为例，作者的原意是让 else 语句与第一条 if 语句配套。但编译程序处理时，一般均把 else 语句与离它最

此为试读，需要完整PDF请访问：www.ertongbook.com