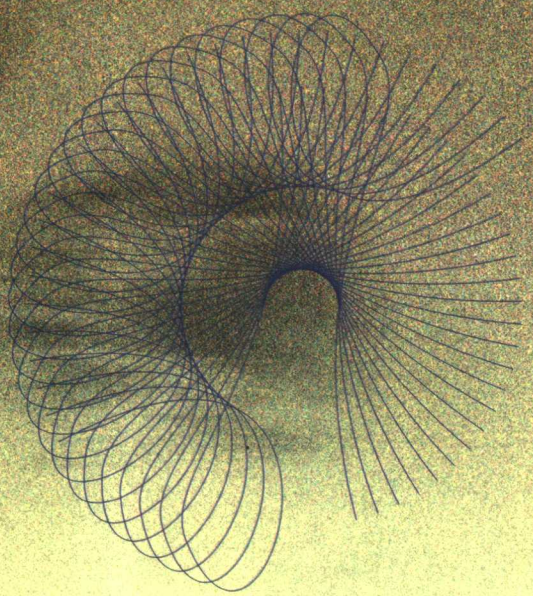
新世纪高等院校精品教材



新编程序设计方法学

陈海波　王申康　编著

浙江大學出版社

新编程序设计方法学

陈海波　王申康　编著

浙江大學出版社

内容简介

本书系统介绍了程序设计方法学的基本理论，结构化程序设计的原理、方法与实例，模块化程序设计的原理、方法和实例，面向对象程序设计方法的原理、方法和实例。

本书适用于计算机本科高年级学生、研究生作为教材使用，也可供计算机软件研究和开发的科研工作者参考。

图书在版编目(CIP)数据

新编程序设计方法学 / 陈海波，王申康编著. —杭州：浙江大学出版社，2004.5
新世纪高等院校精品教材
ISBN 7-308-03654-5

Ⅰ.新… Ⅱ.①陈… ②王… Ⅲ.程序设计－方法－高等学校－教材 Ⅳ.TP311.11

中国版本图书馆 CIP 数据核字(2004)第 028229 号

出版发行 浙江大学出版社
(杭州浙大路 38 号 邮政编码 310027)
(E-mail:zupress@mail.hz.zj.cn)
(网址:http://www.zjupress.com)
责任编辑 杜希武 董雯兰
排　　版 浙江大学出版社电脑排版中心
印　　刷 德清第二印刷厂
开　　本 787mm×1092mm 1/16
印　　张 9.25
字　　数 230 千字
版 印 次 2004 年 5 月第 1 版 2004 年 5 月第 1 次印刷
印　　数 0001－3000
书　　号 ISBN 7-308-03654-5/TP·256
定　　价 18.00 元

前言

本书是在近几年教学过程中不断总结和精炼的结果。主要讨论了程序设计方法学的理论、技术和方法，其中有些是程序设计方法学传统的理论和方法，有些是近几年程序设计方法学的新进展和新成果，同时出于教学考虑，书中尽量避免使用过多的形式化方法。可以说，本书是一本关于程序设计方法学的入门书籍。

本书由4章组成。第一章为程序设计方法学的基本介绍，说明了程序设计方法学的基本概念、产生与发展过程，以及程序设计方法学所关注的基本内容；第二章介绍了程序设计方法学的基本理论，这是程序设计方法学领域的科研工作者近几十年来的研究成果的汇总；第三章介绍结构化程序设计方法学；第四章介绍面向对象的程序设计方法学。

本书设计讲授学时为40学时，讲授对象主要是计算机专业本科高年级学生，也可以供硕士研究生和科研工作者参考。

由于时间仓促，编者水平有限，不妥之处诚恳希望同行与广大读者提出宝贵意见。

目录

第一章　程序设计方法学简介 …… (1)

1.1　程序设计方法学的产生 …… (1)
1.2　程序设计方法学的基本内容 …… (4)

第二章　程序设计方法学的基本理论 …… (8)

2.1　结构化定理 …… (8)
2.1.1　结构化程序 …… (8)
2.1.2　结构化定理 …… (12)
2.2　程序正确性证明 …… (16)
2.2.1　基本概念 …… (16)
2.2.2　部分正确性证明方法 …… (18)
2.2.3　终止性证明方法 …… (23)
2.3　抽象数据类型 …… (25)
2.3.1　基本概念 …… (25)
2.3.2　抽象数据类型的代数规范 …… (32)

第三章　结构化程序设计方法学 …… (40)

3.1　结构化程序设计概述 …… (40)
3.2　结构化程序设计 …… (43)
3.2.1　逐步求精 …… (43)
3.2.2　不变式程序设计 …… (47)

第四章　面向对象程序设计方法学 …… (50)

4.1　面向对象程序概述 …… (50)
4.1.1　面向对象技术的产生与发展 …… (50)
4.1.2　面向对象抽象的原理 …… (53)
4.1.3　面向对象计算的模型 …… (53)
4.2　可复用面向对象程序设计的基本原则 …… (55)
4.2.1　开闭原则(OCP) …… (55)

4.2.2 里氏代换原则(LSP) …… (56)
4.2.3 依赖倒转原则(DIP) …… (57)
4.2.4 迪米特法则(LOD) …… (57)
4.2.5 单责任原则(SRP) …… (58)
4.3 范型程序设计 …… (58)
4.3.1 基本概念 …… (58)
4.3.2 迭代子、函数对象和容器 …… (61)
4.3.3 实例 …… (63)
4.4 设计模式 …… (73)
4.4.1 基本概念 …… (73)
4.4.2 创建型模式 …… (79)
4.4.3 结构型模式 …… (101)
4.4.4 行为型模式 …… (117)
4.5 面向对象程序设计实例 …… (132)

参考文献 …… (140)

第一章

程序设计方法学简介

1.1 程序设计方法学的产生

程序设计方法学是讨论程序的性质以及程序设计的理论和方法的学科。它起源于20世纪70年代初期软件危机的出现，在Dijkstra提出结构化程序设计的思想和概念后得到了快速的发展。各种程序设计方法应运而生，有关程序性质的理论研究成果也不断涌现。除结构化程序设计、结构化定理以外，逐步求精法、功能抽象方法、模块化程序设计方法、递归程序设计方法、面向对象程序设计方法、组件式程序设计方法、范型程序设计方法，以及程序的正确性证明技术、形式推导技术、程序变换技术、抽象数据类型的研究、程序复杂性分析技术，各种计算模型(自动机、图灵、有限框图)等都成为当前程序设计的主要指导方法和理论。应该说，正是程序设计方法学的研究，不断推动着人们程序设计水平、规模和复杂性的提高；反过来，程序设计的提高也推动着程序设计方法学这门学科的不断发展。

程序设计方法学的发展过程与软件的发展过程以及语言的发展过程都存在密切的关系。

首先，程序设计方法学的产生和发展是与软件的发展历程密不可分的。一般认为，软件的发展分为三个阶段。

第一阶段，开创阶段。1955—1965年，计算机软件刚刚起步，高级语言的作用还没有被接受，机器语言和汇编语言仍是程序员的主流编程语言。这个阶段，正如Dijkstra所言："decade of hardware"，程序设计完全是一些特殊的天才才能胜任的工作，这些天才们拥有不为常人所知的技巧，能够使得庞大的机器按照预想执行，而程序设计本身就像蒸汽机起步时代的机械设计一样，被看作是一门艺术，还没有上升到方法学的高度。

第二阶段，稳定阶段。1965—1985年，像科学计算语言FORTRAN、商务计算语言COBOL这样的高级语言开始被人们所接受，软件也不再是随硬件附带的赠品，逐渐成为一个独立的商品，具有其自身的价值。这个时期软件设计方法的主流是结构化分析和设计，以结构化分析、结构化评审、结构化设计以及结构化测试为特征。随着软件应用价值的挖掘，其规模和复杂性不断增加，软件编制的工作量加大，常常需要几百到几千人年。按照原有的手工式方法研制软件周期长、可靠性差、维护困难，软件项目失败的案例屡见不鲜，这个现象就是人们通常所说的"软件危机"。软件危机的出现引起人们对程序设计方法学的重视。1968年，E. W. Dijkstra首先提出了"GOTO语句是有害的"，引起了人们对程序设计方法的大讨论，从此软件的开发走向了方法学的道路。1969年，Dijkstra完成了他的著名文章Notes on Structured Pro-

gramming,围绕着结构化程序设计的理论和方法成为这个时期软件开发的主流。同年,IFIP成立了第一个程序设计方法学工作小组(Working Group 2.3),专门从事这方面的研究和推广工作,程序设计方法学作为一门学科也在这段时间诞生。在这个时期里,关于结构化程序设计的理论占有十分重要的地位,甚至可以说,此时的程序设计方法学就是指结构化程序理论、模块化与逐步求精,以及有关结构化程序正确性证明技术、形式推导技术和变换技术。

【相关材料】 *美国IBM公司在1963—1966年开发的IBM360机的操作系统。这一项目花了5000人一年的工作量,最多时有1000人投入开发工作,写出了近100万行源程序。据统计,这个操作系统每次发行的新版本都是从前一版本中找出1000个程序错误而修正的结果。*

这个项目的负责人F. D. Brooks事后总结了他在组织开发过程中的沉痛教训:"……正像一只逃亡的野兽落到泥潭中做垂死的挣扎,越是挣扎,陷得越深,最后无法逃脱灭顶的灾难。……程序设计工作正像这样一个泥潭……一批批程序员被迫在泥潭中拼命挣扎,……谁也没有料到问题竟会陷入这样的困境……"IBM360操作系统的历史教训成为软件开发项目的典型事例为人们所记取。类似的项目失败有很多,很多人甚至认为人类虽然发明了计算机和软件,但是还缺乏编写大型软件的技术能力,软件会因为对软件开发理论和方法学的缺乏而灭亡。

第三阶段,发展阶段。1985年至今。软件作用和价值牢固地树立起来了。人工智能、知识工程、专家系统以及神经网络领域的研究得以发展与深化。软件市场在世界范围内比较快地速度增长,在美国犹他州已出现以软件为主的第二高技术产业区。目前软件的发展速度已超过硬件产业,占信息产业的主导地位。各种软件理论、技术、方法层出不穷,如面向对象的程序设计方法、类型系统理论、软件复用、设计模式理论、统一建模理论等。可以说在20世纪90年代以后面向对象程序设计方法学成为程序设计的主流。然而,人们对面向对象本质的认识和理解还没有透彻,围绕着面向对象理论产生出各种不同的程序设计方法、概念、理论和技术。这些优秀的理论、技术、方法都从某一个侧面阐述了程序设计的原理和本质,却仍没有能够达到在程序设计方法学的高度将它们统一起来的地步,这也是目前这门学科的发展现状。因此,在第三阶段的后期,人们只是关注程序设计的具体方法和原则,程序设计方法学逐渐成为一个比较笼统和模糊的概念而被淡化。从这个意义上讲,目前的程序设计方法学的内容包括了程序设计的理论和具有一般性的指导方法,是一个广义的概念。

其次,程序设计方法学的产生与发展也是与程序设计语言的发展历程密不可分的。每当程序设计方法的研究有新的突破时,总伴随着新的程序设计语言的产生,新的语言的应用和推广又反过来带动程序设计方法学的发展。

图1.1显示了高级程序设计语言的发展历程。

从程序语言的发展历程来看,每一种新的程序设计方法学诞生,总伴随着标志该方法特色的里程碑式语言的产生。

1. FORTRAN语言的产生使得人们从机器语言和汇编语言走向高级语言,开始了结构化程序设计方法的历程,而PASCAL语言和C语言的成熟标志着结构化程序设计的成熟。

2. LISP语言的产生标志人们开始函数式程序设计方法的历程。

LISP是最早和最重要的符号处理编程语言之一,它于1958年由美国的J. McCarthy提出,并于1960年发表了他的第一篇关于LISP的论文。之后,LISP很快受到人工智能工作者的欢迎,获得广泛应用。目前,除LISP以外,Standard ML,Scheme,Haskell和Miranda都是比较流行的函数式程序设计语言。

对于纯函数式程序设计,它来自于下面简单的基本原理:"一个表达式的值仅仅依赖于它

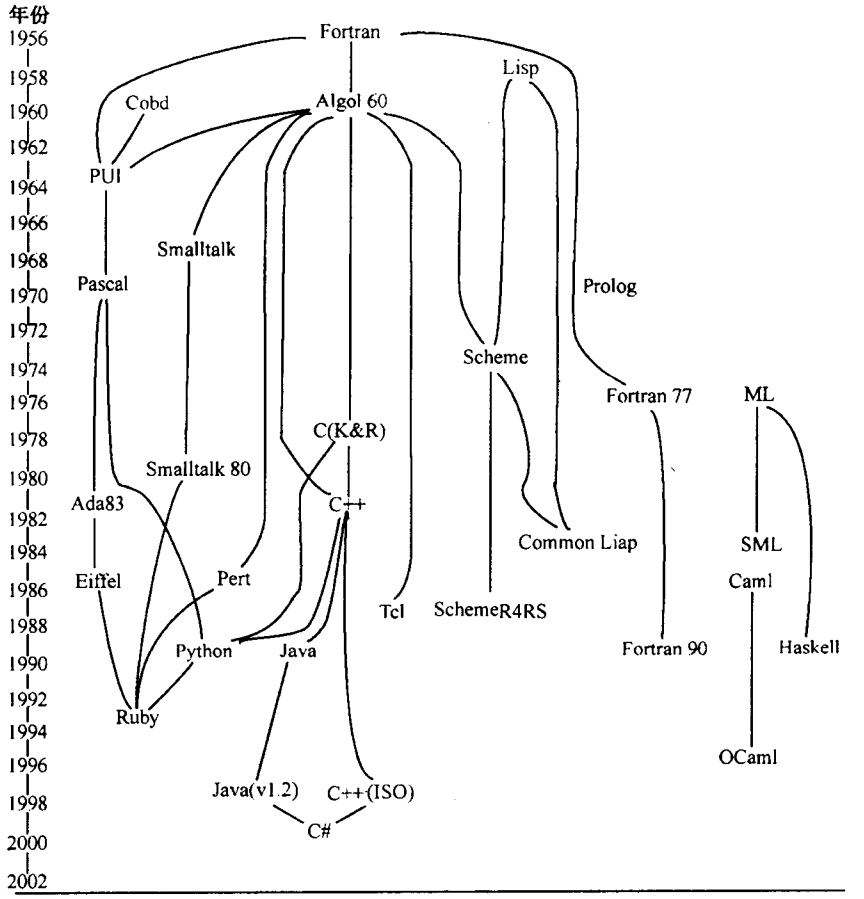


图 1.1

的子表达式的值。”一个像 a + b 这样的表达式的值就是 a 的值和 b 的值的和，这个原理排除了赋值，因为赋值将改变变量的值。因此，纯函数式程序设计方法有时又称为没有赋值的程序设计方法。函数式程序设计语言是一种奇妙而独特的语言，其奇妙之处在于可以独立于任何带有赋值和存储分配的基础机器；其独特之处在于它不需要内存的分配和释放，系统内部会在需要时自动进行存储分配，在不需要时自动释放，并且它将函数作为第一级的值，即函数与过程式语言中的值具有同等的地位，一个函数可以是一个表达式的值，可以作为参数传递，可以被放进数据结构中。理论上，函数式程序设计语言来源于著名的 Lambda 演算。

3. PROLOG 语言的产生标志着逻辑式程序设计方法的产生。

PROLOG 出现在 1972 年，起初是作为一种自然语言理解工具开发的，目前逻辑式程序设计语言更多的还是应用在模式识别、自然处理领域。逻辑式程序设计语言基于逻辑式程序设计方法，直接对应逻辑演算。因此逻辑式程序是由规则和事实组成。Kowaski 用下面的方式描述逻辑式程序设计的工作划分

Algorithm = logic + control

其中逻辑指的是描述算法做什么的规则与事实，而控制则指该算法如何通过某种特定的顺序应用规则。或者说程序设计者提供逻辑，而逻辑型语言提供控制。

4. Simula 的产生开始了面向对象程序设计方法学的研究，而 C + + ，JAVA 的应用标志着面向对象程序设计方法的成熟。

面向对象程序设计方法的概念可以追溯到 Simula,这个语言的设计是为了做模拟的描述和程序设计。Simula 早期工作的一个实例是"机场离港系统"。程序为了能够描述一个繁忙的机场,使用了场景的概念描述:机场中的飞机、乘客、服务员等元素,这些元素后来被分成主动组件和被动组件,进而,对象作为一个统一概念被引进,取代了主动组件和被动组件。Simula 的这些方法被不断的扩充,形成了现代的面向对象程序设计方法的思想。

综上所述,程序设计方法学的主体应用大致经历了:无(手工作坊式的程序设计)→结构化程序设计方法→模块化程序设计方法→面向对象程序设计方法,此外,还有逻辑型程序设计方法、函数型程序设计方法、并行程序设计方法等。伴随着这些方法,产生了相关的理论如下:

结构化程序设计方法——结构化定理、正确性证明、形式推导、程序变换

模块化程序设计方法——抽象数据类型

面向对象程序设计方法——类型系统理论(代数数据类型和类型演算模型)

逻辑型程序设计方法——谓词逻辑理论

函数型程序设计方法——Lamda 演算等形式语义模型

1.2 程序设计方法学的基本内容

我们说程序设计方法学的概念有狭义和广义之分。狭义的程序设计方法学是指传统的有关结构化程序设计的理论、方法和技术;广义的程序设计方法学概念包括了程序设计语言和程序设计的所有理论和方法。正如上节所说,程序设计方法学的概念起源于人们认识到软件危机的产生以及结构化程序设计的研究。在结构化程序设计的研究慢慢衰退以后,程序设计方法学成为一个笼统的概念。那么程序设计方法学作为一门学科,所研究的基本内容是什么呢?要想弄清楚这个问题,首先要清楚程序设计方法学最基本的研究目标。

我们在程序设计中最为关心的是程序的效率和程序的正确性。程序的效率和正确性的保证是由程序的结构和算法决定的,当然,也与程序的易读性、可维护性有密切的关系。简而言之,**程序设计方法学的基本研究目标是通过对程序本质属性的研究来提高程序的效率,保证程序的正确性**,更通俗地说,程序设计方法学的最基本目标是通过对程序本质属性的研究说明什么是"优秀"的程序,怎样才能设计出"优秀"的程序。

一个优秀的程序应该包含这些要素:

1. 结构化

目前面向对象的技术应用越来越广泛,甚至很多人认为结构化的程序设计已经不再需要。事实上,直到目前为止,我们仍然不能脱离结构化程序设计方法的指导。如图 1.2 所示:

```
A
PROC:
switch(state){
Case STATE_ZOOMIN:
                …;goto ERROR;
Case STATE_DRAWRECT:
                …;goto ERROR;
…}
ERROR:
                …;goto PROC;
```

```
B
switch(state){
Case STATE_ZOOMIN:
                …;goto
NOT_SENDING;
Case STATE_DRAWRECT:
                …;goto}
SENDING:
                …;return;
NOT_SENDING:
```

图 1.2

在现代的程序设计中，类似程序 A 这样的非结构化代码仍然存在，对程序 B 是否合理的讨论也仍然存在。即使对于面向对象语言，很多代码部分仍然是过程式的，因此结构化仍然是评价一个程序是否“优秀”的基础。

2. 模块化

模块化程序设计的基本方法(将实现保护起来，让用户通过接口使用功能)，目前也是面向对象的基本方法之一。因此，模块化也是衡量程序优秀的重要指标。如图 1.3 所示。

```
final class Singleton {
    private static Singleton s = new Singleton(47);
     private int i; private Singleton(int x) { i=x; }
     public static Singleton getHandle() { return s; }
     public int getValue() { return i; }
     public void setValue(int x) { i=x; }
```

图 1.3

这类 Singleton 提供 getHandle()接口给用户使用，在保护内部数据实例的同时，实现了对象惟一性的目标。对任何程序而言，低耦合和高内聚始终是设计的目标。

3. 正确性

正确性是一个程序是否优秀的先决条件。在程序设计方法学中，可以通过数学工具在一定程度上证明程序的正确性，甚至通过数学工具自动推导程序。

4. 可重用

可重用是一个程序设计是否优秀的关键性指标。关于可重用这个指标将在第四章详细介绍。

5. 可维护

如图 1.4 代码所示。

```
void MyFunc(COPR_DATA InputRect,int crntQtr,EMP_DATA
EmpRec,float EstimRevenue,float YTDRevenue,int ScreenX,int
ScreenY){
■int i;
■for(i=1;i<100;i++)InputRec.revenue[i]=0;
■EstimRevenue=YTDRevenue*4/crntQtr;
■if(ExpenseType==1)for(i=1;i<12;i++)Profit[i]=Revenue[i]-Expense.Typ
el[i];
```

图 1.4

这段代码存在以下问题：

(1) MyFunc 的命名并没有说明该函数是做什么的；

(2) for(i=1;i<100;i++)以及 for(i=1;i<12;i++)没有指明 100 和 20 的意义；

(3) int ScreenX,int ScreenY 定义了，却没有使用。

以上问题不影响程序运行的效率，甚至也不影响程序的现实正确性，但却是不可维护的，从而影响程序今后的正确性。因此，可维护也是衡量程序是否优秀的指标之一。Dijkstra 建议

语句的设计应使人们通过读文本过程就可以理解计算过程。

6. 性能均衡

性能均衡实际上是指系统效率与实际运行环境限制之间的均衡。理论上一个程序运行效率越高越好,然而往往系统效率的保证需要系统环境方面的代价。例如,当一个设计得高效的程序,需要消耗实际运行环境大得多的物理存储资源,则它不能称为是一个优秀的程序。

程序设计方法学不同于软件开发方法,两者是有联系又有区别的。两者都关注如何提高程序的效率和正确性。不过前者单纯从程序本身的固有特性角度来达到目标,而后者研究程序设计方法的手段更加广泛,它包括了从系统工程、社会学、组织行为学、统计学等方面进行研究。

程序设计方法学也不同于程序设计技巧学,两者的区别实际上是方法学和技术学之间的区别。作为方法学而言,以程序设计中的指导性理论以及基本原则作为研究重点,具有普遍性,而程序设计技巧则属于只能在某些特定环境下使用的特殊性方法。

图 1.5 给出了程序设计方法学的基本内容。

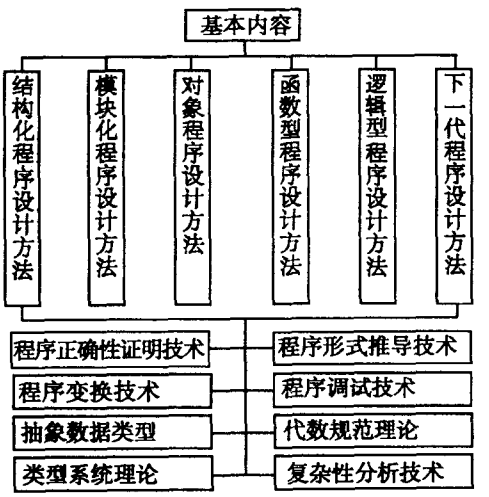


图 1.5

可以看出,程序设计方法学的内容可以分成方法和理论两个方面:前者倾向于对程序设计中一般性问题的指导原则;后者倾向于揭示程序设计的本质,以期实现人们对程序设计的最理想境界——程序设计的自动化和智能化。

鉴于程序设计方法学这门学科的特点,本书采用理论和实际并重方式。它包含了以下基本内容:

1. 程序设计方法学的基本理论

本书中的程序设计方法学基本理论主要是指结构化程序设计方法时期所产生的主流方法学理论,包括了结构化程序以及结构化定理;程序的一般正确性证明理论;程序的形式推导理论;计算复杂度分析技术以及抽象数据类型的理论。面向对象方法学中的一些理论由于缺乏统一性没有被吸纳到本书中。

2. 结构化程序设计方法的主要技术

介绍结构化程序设计方法中的主要技术概念(包括逐步求精、自顶向下、不变式设计)和实例。

3. 面向对象程序设计方法的相关技术

本书不是一本有关面向对象的基本概念和理论的介绍书籍，这些方面的知识读者可以阅读参考文献。作为面向对象程序设计方法，主要讲述面向对象的计算模型以及对软件复用具有指导意义的范型程序设计和设计模式。

第二章

程序设计方法学的基本理论

2.1 结构化定理

2.1.1 结构化程序

一、基本概念

结构化程序设计方法的研究包括了四个紧密相关的课题。

课题 1:程序设计方法。指在用计算机程序解决问题过程中比较有效的一般性的方法,如抽象的层次观念,逐步求精,自顶向下等。

课题 2:程序标记法。任何一个语言都是使用符号标记的,它的意思及其阐述形式对理解和维护程序都是十分重要的。一个凌乱的程序设计语言显然有碍设计思路的清晰。结构化程序设计方法学帮助我们简化程序设计过程,确保适合的控制结构和清晰的数据结构。

课题 3:程序正确性。程序的正确性是指一个程序和它预期的功能之间是否一致,结构化程序设计方法试图通过证明的方法来达到这个目的,因此数学归纳法、形式逻辑、公理化证明等数学工具都能够应用到结构化程序的证明当中。

课题 4:程序验证。程序验证是将程序正确性证明的理论应用到实际环境中去的过程。一个程序的正确不只是语法正确,同时也要语义正确,还与特定的运行环境相关。即通过程序正确性证明理论的应用,确定当输入之前满足某些前提的程序,在执行之后是否能达到我们预计的结果。

课题 1、2 偏重于程序设计方法的应用;课题 3、4 偏重于程序设计方法的理论。本章中将重点介绍结构化程序设计的理论知识,结构化程序设计方法的应用知识将在第三章介绍。

高级语言与汇编语言、机器语言相比给人们进行程序设计带来了极大的方便。在高级语言的发展过程中,人们渐渐习惯先根据需要解决的问题画出流程图,再围绕流程图进行编码。在此过程人们发现,虽然流程图直观,简单而且有吸引力,但也有很多不足之处:首先是费时、费力;其次是缺乏一个好坏的度量标准;接着是流程图开始于一点,然后分成两支,向上和向下,从一页到另外一页,有时再返回,周而复始,难以控制;再次是流程图强调程序流程的路径而不是程序的逻辑;最后流程图违背了对功能的依赖性,破坏了局部修正的能力。正因为如

此，人们希望将流程图进行简化，形成一种更简单的结构，再在这种简单的流程图结构上进行程序设计，这样才可以大大提高程序设计的效率，提高程序的正确率。由此，引出了以下基本概念：流程图程序、正规程序、基本程序和结构化程序。

二、流程图程序

理论上任何一个程序都可以用流程图描述，换句话说，我们可以同一种图形语言来写程序，这样的程序称为流程图程序。它是一种程序比较直观的表现形式。

一个流程图程序由以下图形符号组成。

1. 处理单元(Process Box)

该单元又称为函数结点。该结点只有一个入口线和一个出口线，如图 2.1(a)所示。其中 f 是函数结点的名称。函数结点一般和赋值语句对应。

2. 判断单元 (Decision Box)

该单元又称为谓词结点。该结点有一个入口线和两个出口线，而且谓词结点不改变变量的值。如图 2.1(b)所示，其中 p 是一个谓词，取值真或者假。谓词结点一般对应条件语句。

3. 连接单元(Junction Box)

该单元有两个入口线和一个出口线(又称为汇点)或者有一个入口和多个出口，中间用一个圆表示，控制路径在此会聚或者在此发散，如图 2.1(c)所示。汇点不执行任何运算，只是一个简单的结点。

4. 流线(Flow Lines)

流线又称为连接线，它表示为带箭头的方向，从一个处理单元到另外一个处理单元，如图 2.1(d)表示。它表示程序的运行方向。

5. 开始单元(Start Box)

如图 2.1(e)所示，表示为长圆形的符号及其引出线。它指示出一个算法流程图的逻辑起始点。

6. 结束单元(Finish Box)

如图 2.1(f)所示，它表示算法的逻辑终止。

7. 输入、输出单元(Input/Output Box)

如图 2.1(g)所示，它表示对一个或者一批数据的 IO 操作。

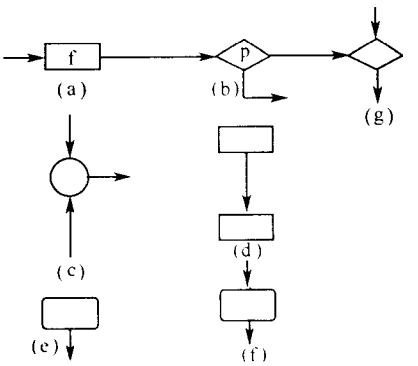


图 2.1

三、正规程序

一个流程图程序如果满足以下两个条件，称为正规程序：

(1)只有一个入口线和一个出口线；

(2)对每一个结点，都有一条从入口线到出口线的通路。

如图 2.2 所示的程序是一个正规程序，而图 2.3 所示的程序不是正规程序。由于正规程序只有一个入口线和一个出口线，因此正规程序可以概括为一个函数结点。如图 2.4 所示的程序可以抽象为函数结点 k。

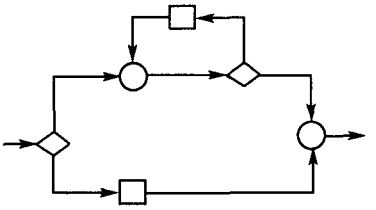


图 2.2

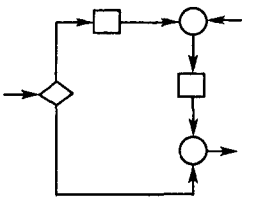


图 2.3

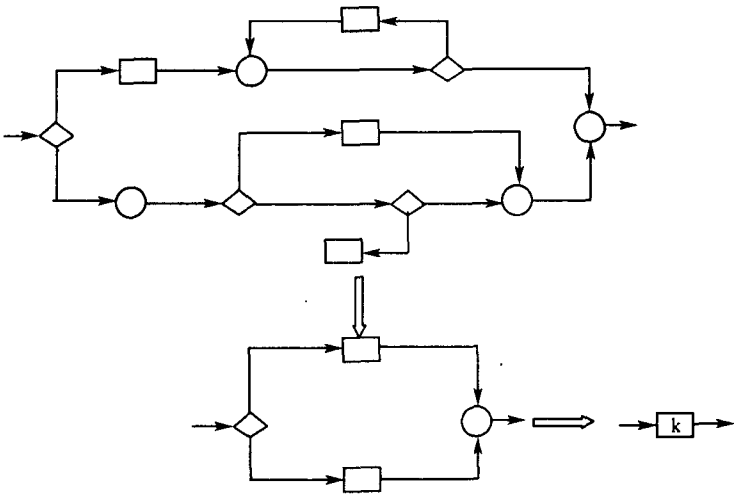


图 2.4

从图 2.4 中可以看出，一个正规程序的某些部分仍然是正规程序，这些部分称为正规子程序。

四、基本程序

一个正规程序，如果不包含多于一个结点的正规子程序，称为基本程序。即基本程序是正规程序的一种，且其正规子程序不多于一个结点。如图 2.5 所示，a 为基本程序，而 b 是正规程序。

上述定义说明基本程序是程序结构的最小单元，不能再进行分解。因此，我们可以将一个流程图程序分解为其最小单元——基本程序的集合。通过对所有的程序控制结构的分析，可以找到 7 种形式的基本程序，如图 2.6 所示。

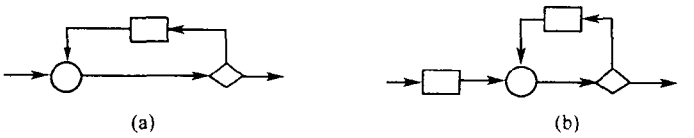


图 2.5

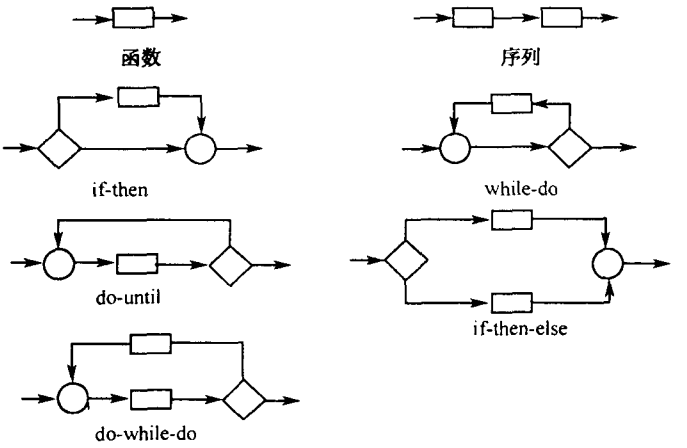


图 2.6

以上 7 种基本程序不一定在所有程序结构中都用到。例如，while-do 程序可以用 do-until 程序替换。因此，对于任何一个流程图程序，我们只要从 7 种程序结构中挑出一部分就可以构成了。对于不同的程序，可以在基本程序中挑出不同的部分组合（顺序，if-then，while-do 和顺序，if-then-else，do-while-do 就是两种不同的组合）。

我们把从 7 种基本程序挑出的这部分叫做基本程序的基集合。反复应用基集合构造出来的程序称为基集合的复合程序。

五、结构化程序

早在 1966 年 Bohm 等人就证明了任何一个程序只要具备三种基本程序：顺序、if-then-else 和 while-do 就足以表示出所有形式的程序。

如果一个基本程序的函数结点用另外一个基本程序替换，则产生一个新的正规程序，称为复合程序。

由基本程序的定义，我们可以引出结构化程序的定义：由基本程序的一个固定的基集合构造出来的复合程序称为结构化程序。

M. H. Williams 把非结构归纳为五种基本的成分，即：

(1)异常的选择通路；

(2)循环多出口点；

(3)循环多入口点；

(4)重叠的循环；

(5)并行的循环。

这五种非结构是不能分解到基本程序的。可以证明，如果一个流程图是非结构化的，则它至少包含这五种非结构成分中的一种。这是因为包含这五种结构的任何一种流程图，都不能