第3章

银行常用程序设计语言

程序设计语言是用来编写计算机程序的语言。程序设计的好坏不仅影响到程序的使用是否方便，而且涉及程序员编写程序的质量。本章首先介绍结构化程序设计的基本原理，然后对银行常用的几种程序设计语言进行简要的叙述及比较分析。

3.1 结构化程序设计基本原理

结构化程序设计，是著名学者 E. W. 戴文斯特拉和 C. A. R. 霍尔于 20 世纪 60 年代后期提出的一种开创性的程序设计方法，它已成为当今计算机程序设计的重要基石和先进工具之一。结构化程序设计有以下基本特征：

(1) 程序设计＝算法＋数据结构＋程序设计方法学＋计算机语言，其核心和基础是算法；

(2) 从根本上讲，计算机程序是算法的具体表述形式之一，即用计算机语言描述的算法；

(3) 程序编码，即用计算机语言编写的程序，是算法构造(即算法设计)的最终表现形态。

简言之，离开了算法这个计算机程序设计的基础和关键，程序及其编码就成了无木之林、无源之水。

3.1.1 程序设计基本要素

1. 解题的基本模式

人与计算机解题的基本模式如图 3-1 所示，这表明计算机只是部分地取代了人脑解题的功能，因此应用计算机解决各种不同的实际问题，没有人的积极参与是不行的。从计算机科学来讲，真正客观地把“人”与“计算机”有机地统一起来，组成以人为主导、以计算机为主体的，各自发挥特长与效能的强大而完整的现代信息处理高级系统“人-计算机系统”，则是计算机程序设计(computer program design)。

结构化程序设计，总是从问题分析开始的，是借助计算机解决具体问题的基本过程。即根据给定问题的性质和要求，考虑计算机系统的性能和特点，采用计算机科学的方法和技术，实现“以人为主导、以计算机为主体”解决给定问题的辩证统一过程。通常可概略地分为

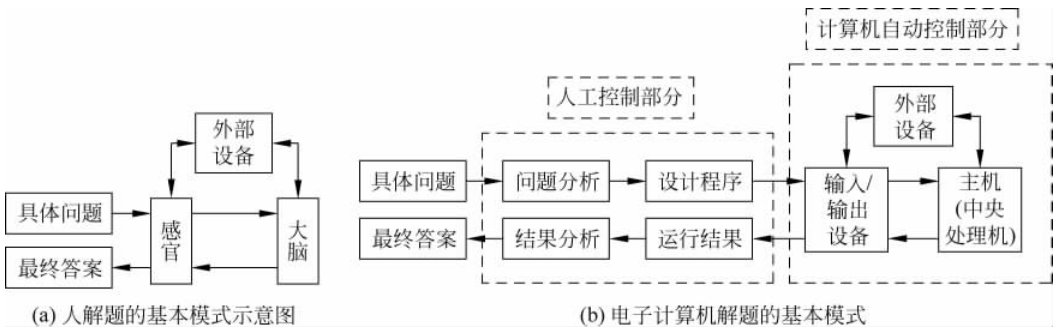

图 3-1　人与计算机解题的基本模式

问题分析、算法设计、程序编码、分析调试、运行维护五个基本阶段。

2. 问题分析

问题分析阶段一般可进一步分解成如下诸环节，即精确描述问题、识别数据输入、判定信息输出、正确设置变量、建立数学模型、决定处理方式、选择计算方法等。其中，建立数学模型与选择计算方法尤其重要。

1）精确描述问题

精确描述问题，是指必须准确无误地认清所给的问题，并精密无差地明确解决它的系统目标。这一环节倘若有疏谬，则将“差之毫厘、失之千里”。显然，在精确描述问题时，应力求考虑周密、分析详尽、描述精确，而避免草率行事、似是而非、描述失当。同时，还应当防止错误化、简单化、复杂化这三种不良倾向。

例 3-1　错误化的问题描述：如把求解一元二次方程错误地当成求解二元一次方程组或一元一次方程来处理，或者把建立职工工资表的问题混同于学生成绩表问题。

例 3-2　简单化的问题描述：如在运动员运动成绩名次处理中，最容易发生的不完备问题描述是忽视“同成绩者必须并列同一名次”的原则，而误将名次排列问题简单地等同于成绩排序问题。

例 3-3　复杂化的问题描述：如求一元二次方程实根问题，误被扩大为还要考虑求其虚根。

一般说来，成功的问题描述通常是给定问题领域的终端用户专家与计算机的应用开发专家互相取长补短、共同切磋、通力合作的产物。因此，双方应针对有关此问题的各种情况进行有效的交流，以便对给定问题尽可能考虑周详、表达准确、描述完整。显而易见，凡从事或参与计算机应用与开发者，应当自觉养成善于学习、勤于总结、擅长积累、长于合作、精于描述的良好风尚。

2）识别数据输入

识别数据输入，是指在解决给定问题所涉及的若干数据及其处理过程中，识别哪些数据是需要从外部提供(即输入)给计算机系统，并决定如何提供。计算机系统处理的数据，按其产生的方式，可分为输入性数据与非输入性数据(由计算机系统本身自动生成)。因此，正确辨别输入性数据与非输入性数据，对提高计算机解题的工作效率与适应能力意义重大。

例 3-4　求任意给定的一元二次方程 $ax^2+bx+c=0$ 的根。需要输入它的系数 a、b、c 的值(因为它们均为输入性数据)，并以采用键盘人工输入的方式为宜；而其根 x_1 和 x_2 的

值就不需要采用输入方式来获取(因为它们都是非输入性数据),可用求根公式和韦达定理由计算机自动计算来求得。

例 3-5 建立职工工资表,应根据每个职工输入其基本数据:工号、姓名、各项应得工资基础数据(如:基本工资、工龄工资、物价补贴、奖金、……)、各项应扣工资基础数据(如:房租费、水电费、互助储金、罚款、……)等输入性数据,且以采用由磁盘文件或数据库文件输入的方式为妥;而非输入性数据如应得工资、应扣工资与实发工资可不用输入方式来获得,而用求和或求差的运算公式由计算机计算生成。

3) 判定信息输出

判定信息输出,是指在解决给定问题过程中,必须使终端用户得到所关心的信息。即把人们真正所需的信息从计算机系统内输出,并决定怎样输出。在这一阶段,应当正确判定为解决给定问题所必需的信息输出及其适度的输出总量;同时还应当防止信息输出量不足与过剩两种不良倾向。若信息输出量不足则不能完整地给出符合要求的正确答案,而信息输出量过剩则会造成计算机资源的严重浪费。

例 3-6 在输出职工个人的工资信息时,若只输出职工实发工资数而没有同时输出其各项应得与应扣工资等基础信息,则会因其信息输出量不足而使人们不知自己为什么会得此实发工资数,从而增加财务人员不必要的解释工作量,造成工资发放工作的被动;同样,若再累赘地给出诸如婚否、年龄、性别、学历之类与工资发放无关的人事管理信息,则毫无意义。

4) 正确设置变量

正确设置变量,是指对所论问题的数据输入、加工处理、信息输出等主要操作中所涉及的数据(包括原始数据、中间结果、最终结果等),必须采用适当的变量形式来描述、处理和存储,如图 3-2 所示。正确、合理地设置所需各变量,是计算机应用技术的基本功之一。

图 3-2 数据变量的设置

例 3-7 在求解一元二次方程 $ax^2+bx+c=0$ 的实根时,通常可选用变量 a、b、c、x_1、x_2,以分别表示所给一元二次方程的二次项系数、一次项系数、常数项、实根 1、实根 2。显然,如果只选用 a、b、c 三个变量,则将不便于求根处理;而若再增加 x、x_3、x_4 等无关的变量,则既无必要又不经济。

5) 建立数学模型

建立数学模型,是指在对给定问题进行了科学的定性分析基础上,进一步抽象出对此问题的定量化数学描述形式。一般说来,通过数学模型来处理并解决给定问题,比直接对原始形态下的非数学模型的朴素形式更为深刻、简明、有效。因此,建立数学模型是问题分析阶段的核心工作,它是借助计算机解决给定问题的基本前提。

通常,有许多问题可以抽象描述为数学解析表达式,例如求解一元二次方程,解线性方程组,投入产出等;但是必须指出,也有相当多的问题未必总能抽象为相应的数学解析表达

式,例如职工工资表处理、银行业务处理、图书资料检索等。因此,数学解析表达式仅仅是数学模型的主要形式之一,而切不可误以为“数学模型就是数学解析表达式”。

应当指出,数值领域问题通常易于建立其数学模型(例如:数值积分、数值微分、线性规划、动态规划、线性方程、存储问题、排队问题等都有较成熟的数学模型);而非数值领域问题(例如:排序、查找、分类、索引等)则常常难于建立其数学模型。因此实际应用中,如果所论问题已有其相应的较成熟的数学模型,则应优先考虑借用或改进现有数学模型;否则,就应当由设计者利用解析法、数值法、直观法、试探法、类推法等方法,自行构造相应的数学模型。

由于描述同一问题的数学模型未必唯一,因而努力挑选出能更恰当地反映和更简捷地解决给定问题的优化数学模型至关重要。

例 3-8 在求解一元二次方程实根时,若已求得一个实根 x_1,则另一实根可用如下两种数学模型之一求得:一是利用求根公式;二是利用韦达定理。显然,第二个数学模型比第一个数学模型更为简易、精确。

6) 决定处理方式

决定处理方式,是指对给定问题究竟是采用人工处理方式,还是采用计算机处理方式,应视所论问题的实际情况而权衡利弊后慎重决定,因为并非总是以计算机处理方式为上策。计算机应用,实际是当今历史条件下一种新的社会经济活动;因而,应当用现代社会经济的一般尺度,即社会效益与经济效益,来衡量和决定是否宜于应用计算机解决给定问题。通常,是否采用计算机处理方式的判定条件如下:

(1) 当人工处理方式所需时间和费用比计算机处理方式代价高时;

(2) 当解决所给定问题的数值计算或数据处理工作量非常大,以至于难于用人工处理方式来进行时;

(3) 当同一性质的给定问题重复出现率高,要求多次作类似处理,而且每次解决有关问题的处理过程基本上大同小异(即只是某些数据有所不同而已);

(4) 当要求解决所给问题的响应速度快、信息可靠性好、数据精度高、自动化程度高时。

显然,若解决的给定问题并不具备上述任何一个条件,则表明该问题的处理不必采取计算机处理方式,而应当考虑采用人工处理方式,并同时终止以下各阶段的计算机处理过程。

例 3-9 按照上述第(1)个条件,对某年某月某职工工资表的一次性特殊处理,不宜采用计算机处理方式,因为计算机仅一次性地处理这一特定工资表的代价高于人工处理同一问题的代价。但是,如果希望每年每月都能尽快建立某单位的当月职工工资表,则采用计算机处理方式,因为它符合上述第(3)、(4)两个条件。

例 3-10 对任给 $n(\leqslant 10)$个实数,要找出其中最大者与最小者,自然用人工处理最为简便;但是对任给 $n(\geqslant 10^{10})$个实数,同样也要找出其中最大者与最小者,那就不是任何一个人毕生所能完成的(因为以每秒处理 1 个数据的速度估算,要处理完 10^{10}个数据要 300 年以上),因此只能采用计算机处理方式。

7) 选择计算方法

选择计算方法,是指必须精心挑选适合所论问题的最优计算方法,因为计算方法得当与否,其效果往往差别很大。一般来说,描述给定问题的数学模型不一定是便于计算机处理的数学表述形式,因而通常还必须精心选择能将指定数学模型转化为适于计算机处理的计算

方法(因为表述同一数学模型的计算方法常常不是唯一的)。必须强调指出,计算方法选择得当与否,不仅直接影响到解决所给问题的效率、精度和费用,而且关系到整个工作的成功与失败。

例 3-11 求一元五次方程 $x^5+7x-6=0$ 的实根(绝对误差精确到 0.000001)的计算方法,可有如下两种迭代法,但其结果却截然不同。

先看计算方法 1:$x_2=\sqrt[5]{6-7\times x_1}$,其迭代计算过程及结果如下:

迭代次数	x_1	x_2	绝对误差$\lvert x_1-x_2\rvert$
1	0.000 000	1.430 970	1.430 970
2	1.430 970	− 1.320 610	2.751 580
……			
10	1.742 900	− 1.440 400	3.183 300
11	− 1.440 400	1.742 900	3.183 300

显然,计算方法 1 无论迭代计算多少次,永远不可能求得合乎要求的原方程的实根。

再看计算方法 2:$x_2=6/(x_1^4+7)$,其迭代计算过程及结果如下:

迭代次数	x_1	x_2	绝对误差$\lvert x_1-x_2\rvert$
1	0.000 000	0.857 143	0.857 143
2	0.857 143	0.795 780	0.061 363
3	0.795 780	0.810 699	0.014 919
……			
9	0.807 955	0.807 957	0.000 002
10	0.807 957	0.807 957	0.000 000

至此已经求得原方程的实根 $x\approx 0.807\,957$(其绝对误差不超过 0.000 001)。

比较计算方法 1 与计算方法 2 及其结果,可得出如下结论,对该数学模型 $x^5+7x-6=0$ 而言,计算方法 1 是发散的、失败的计算方法;而计算方法 2 则是收敛的、成功的计算方法。

由此可见,搞好问题分析诸环节的确事关重大,而那种轻视甚至忽视问题分析,对所论问题一知半解或者不甚了解,便贸然急于仓促编写程序的不良习惯,应当坚决戒除。

3.1.2 算法设计初步

继问题分析阶段之后,就进入应用计算机解决问题的关键性基本阶段—算法设计(algorithm design)。算法是计算机程序设计的核心,是计算机科学中最基本的重要概念之一。正确学习、理解、掌握这一基本概念是非常重要的。

1. 系统与算法

系统一般分为可控系统(controllable system)与非控系统(Uncontrollable system)。显然,与人类关系密切的不是非控系统(例如,彗星运动系统、雷电生成系统、台风形成系统、……),而是可控系统(例如,车队作战系统、电力调度系统、电器自控系统、……)。

可控系统又可分为施控子系统(controlling subsystem)与受控子系统(controlled subsystem)。可控系统的根本特性是通过其施控子系统对受控子系统的调控作用来对整个

系统实行控制。可控系统的行为方式,即系统运行全过程中施控子系统与受控子系统彼此响应、协同工作的行为方式,其根本模式可抽象为系统算法(system algorithm),即可控系统为实现系统目标,要求其操作执行者对操作对象遵照系统所认定的操作方式,遵从系统所设定的控制方式,并一步一步具体施行的有穷操作过程的描述。

因此,如下事实显然成立并且十分重要:

- 系统目标不同,则其算法必不同;
- 系统目标相同,而操作执行者不同,则其算法必不同;
- 系统目标、操作执行者相同,而操作对象不同,则其算法必不同;
- 系统目标、操作执行者、操作对象相同,而操作方式不同,则其算法必不同;
- 系统目标、操作执行者、操作对象、操作方式相同,控制方式不同,则算法必不同。

当且仅当系统目标、操作执行者、操作对象、操作方式、控制方式都相同的算法,才是相同的算法。例如:

(1) 提供淡水的水井、汲取卤水的卤井、抽取石油的油井虽然都是"井",但由于其系统目标各异,故它们的勘测打井算法不同;

(2) 有人驾驶侦察机与无人驾驶侦察机,其系统目标都是"侦察敌方、获取情报",但由于其操作执行者各异,故它们的侦察飞行算法必不同;

(3) 由某人等分一根长为 35cm 的线段,其系统目标、操作执行者、操作对象均无区别,若认定不同的操作方式(仅用圆规和直尺作图寻找线段中点的几何操作,采用对折等长线条决定线段中点的模拟操作;利用量具测算线段中点的测算操作等),则其等分线段算法必不同;

(4) 四川都江堰水利工程系统的系统目标、操作执行者、操作对象、操作方式都不变,但若设定不同的控制方式(如:正常水位时的灌溉水量调度控制方式,警戒水位时抗洪水量调度控制方式),则其河水调度算法必不同。

由此可见,系统算法是一切可控系统固有的重要本质特征与统一的基本行为模式,其系统目标、操作执行者、操作对象、操作方式、控制方式五大基本要素,对任何可控系统都是至关重要的。它们之间互相依存、相互协同、共成算法的天然联系,可概括成如下公式:

系统算法=系统目标+操作执行者+操作对象+操作方式+控制方式

2. 计算机与算法

以人为操作执行主导者、计算机为操作执行主体者的系统算法,称为计算机算法(computer algorithm),简称算法。但是,它绝不是天然自生的,而是由人在问题分析基础上专门为用计算机解决给定问题而设计产生的一类特殊算法。所谓计算机算法,是指:"人-计算机"系统为解决给定问题,需要以"人为主导、计算机为主体",对所论问题的数据,采取所设定的顺序结构、选择结构、循环结构、并行结构、子算法结构(必要时方可辅以捷径结构)及其结构化组合的控制方式,来组织和控制所认定的操作方式,并一步一步具体实施的有穷操作过程的描述。

事实上,人的主导作用主要体现在:首先是算法设计,即如何针对计算机的特点和要求,来科学地设计解决给定问题的算法;其次是程序编码(或称编写程序,简称编程),即如何按照给定计算机语言的要求和特点,正确地把所设计的算法编写(翻译)成该计算机语言下的程序;最后是结果分析,即如何依据客观规律与问题要求,慎重地甄别计算机执行给定程序后所得执行结果的正确性、科学性与合理性。

而计算机的主体作用主要体现在：一旦人设计好解决给定问题的算法，编写好该算法所决定的计算机语言程序，并把该程序交给计算机执行之后，计算机将精确、及时、有效地自动执行该程序(必要时可由人辅以适当的配合操作)，并输出人们所希望的执行结果。显然，借助计算机来解决给定的问题，其热点、重点、难点都是算法设计，而绝不是人们常常误认为的程序编码。因此循着“对→好→巧→妙→绝”的算法设计发展轨迹，努力探索和追求解决同一问题的算法族中佼佼者—最优算法，是算法设计人员应有的宝贵品质和良好习惯。

例 3-12 一个最简单的算法的描述

算法		注释
第 1 步	开始	{算法自此开始}
第 2 步	行输出“您好!”	{在某行原样输出该字符串，且每个汉字占 2 格}
第 3 步	结束	{算法到此结束}

3. 算法的系统目标

算法的系统目标是解决给定问题。对于“人-计算机”系统而言，即在解决给定问题全过程中，算法必须同时为人和计算机两方面都提供及时、可靠、方便、友好的“人-计算机”工作界面，使人和计算机能得以充分发挥各自特长与效能。算法之所以特别重要和有用，是因为算法被执行(即其操作被一步一步具体施行)后所输出的信息，即执行结果，不仅是算法的系统目标的具体实现，也是人们所关心结果的具体兑现。因此，凡能正确实现系统目标的算法，即能向人们提供正确及时、清晰直观、简明易懂、信息完备的执行结果的算法，都是应当提倡的良好算法；反之，则是应该防止的病态算法甚至错误算法。

例 3-13 试设计并比较“求最小 1、2、3 位数的算术平均数”的良态算法和病态算法。

问题分析：显然，最小 1、2、3 位数必为 1、10、100。若设 x 为用以描述所求算术平均数的变量，并考虑到为人和计算机提供良好界面，则良态算法可用自然语言描述如下：

算法		注释(良态算法)
第 1 步	开始	{算法自此开始}
第 2 步	x←(1＋10＋100)/3	{求算术平均值 x 的赋值操作}
第 3 步	行输出“最小 1、2、3 位数的算术平均数为”	{在某行展示最终结果的性质}
第 4 步	行输出 x	{在下一行展示所得算术平均值}
第 5 步	结束	{算法到此结束}

该算法中，赋值操作“x←(1＋10＋100)/3”的算法意义是，把(1＋10＋100)/3 的值存放到变量 x 中。因此，良态算法的执行结果是：最小 1、2、3 位数的算术平均数为␣37.0(其中，“␣”表示一个空格)

但是，如果设计时考虑不周、处理不当，而误失上述第 3 步操作，便只能得到病态算法。因为它只能输出一个孤零零的数“37.0”，而这常会使人(包括算法设计者和算法使用者)迷惑不解——产生诸如“这个‘37.0’到底是何物？来自何方？”之类的疑问。自然，如果连良态算法中的第 3 步也误被丢弃，则所得的算法就是全然无用的“病态算法”了。

4. 算法的操作执行者与操作对象

认清算法的操作执行者及其操作对象的性质、特点与效能，是算法设计顺利进行并取得

成功的基本保证。

1）算法的操作执行者

算法的操作执行者，其主导是人，其主体是计算机。既然其主导者是人而不是计算机，故在算法设计中“以人昏昏，使计算机昭昭”是不行的；既然其主体者是计算机而不是人，故在设计算法中，既要充分发挥人的主导作用，又绝不能“越俎代庖”对计算机取而代之。

2）算法操作对象

既然算法的系统目标是解决给定问题，算法的操作执行者是“人-计算机”，因此算法的操作对象必然是该问题所涉及的数据，在计算机科学中，脱离数据的算法与脱离算法的数据都是同样不可思议的。

计算机的数据，总是属于一定的数据类型，各种类型数据总是以一定的数据形态（即常数、变量、函数、表达式）出现在算法（或程序）中。每一种数据类型，决定本类数据的取值方式与运算方式；每一种数据形态，则具有自己的结构方式与使用方式。属于同一数据类型的不同形态数据（常数、变量、函数、表达式），统称为同型量（或同型数据）。不同计算机语言的数据、数据类型、数据形态各有所不同，但其结构本质与构造实质则都是同构化的。因此，通过对它们共同的典型代表，如表 3-1 所示，与数据形态的学习，必将有利于人们学习并掌握各种计算机语言下的数据类型。为了增强算法（或程序）的可移植性，建议在算法（或程序）设计中尽可能优先使用最基础的数据类型，即整型、实型、字符（串）型、逻辑型、数组型、记录型、文件型、数据库等，特别是前四者所构成的标准（内存数据）类型。

表 3-1　同构化数据类型的简要划分示意

<table>
<tr><td rowspan="15">数据类型</td><td rowspan="10">内存
数据类型</td><td rowspan="6">基本类型</td><td rowspan="4">低级类型</td><td rowspan="3">标准类型</td><td rowspan="2">数值型</td><td>整型</td></tr>
<tr><td>实型</td></tr>
<tr><td colspan="2">字符（串）型</td></tr>
<tr><td colspan="2">逻辑型</td></tr>
<tr><td colspan="3">其他非标准类型</td></tr>
<tr><td rowspan="2">中级类型</td><td colspan="3">数组型</td></tr>
<tr><td colspan="3">其他非数组型</td></tr>
<tr><td rowspan="4">扩展类型</td><td rowspan="3">高级类型</td><td colspan="3">内存记录型</td></tr>
<tr><td colspan="3">指针型</td></tr>
<tr><td colspan="3">其他型</td></tr>
<tr><td colspan="4">超级类型（发展中各内存数据类型）</td></tr>
<tr><td rowspan="5">外存
数据类型</td><td rowspan="2">基本类型</td><td>低级类型</td><td colspan="3">字符、基本项、组合项</td></tr>
<tr><td rowspan="2">中级类型</td><td colspan="3">外存记录</td></tr>
<tr><td rowspan="3">扩展类型</td><td colspan="3">文件</td></tr>
<tr><td rowspan="2">高级类型</td><td colspan="3">数据库</td></tr>
<tr><td colspan="3">其他</td></tr>
<tr><td colspan="4">超级类型（发展中各外存数据类型）</td></tr>
</table>

5. 算法的操作方式与控制方式

算法的操作方式与控制方式，是算法最富有生气、最具有活力、最反映才智的关键基本要素。前者，其形态往往变化不大，并且其构造较简单而具体，有如构成算法的躯体；后者，其形式常常变化多端，而且其构造较复杂而抽象，恰似主宰算法的灵魂。不同计算机语言的操作方式与控制方式各有不同，但其结构本质与构造实质则是同构化的。人们通过对同构

化操作方式与控制方式的学习，将对算法有更深的认识。

1）算法的职能操作与操作形式

数据运算与职能操作，是构成算法所认定的同构化操作方式的基础，是用以解决所论问题操作方式的描述手段。当然，不同计算机语言的数据运算与职能操作各有特色，但是无论何种计算机语言都无一例外地以这样或那样的表现形态，至少设置有如表 3-2 所示同构化的基本数据运算与基本职能操作。它们之间的关系是，前者是后者的素材，后者是前者的应用。实际上，数据及其运算不能独立于职能操作与控制结构之外而存在，它只有在职能操作（例如赋值、输出）与控制结构之中，才能发挥其效能；职能操作也不能独立于控制结构之外而独立存在，它唯有在控制结构之中，才能产生其效力。

表 3-2　算法的同构化数据运算与职能操作的划分示意

<table>
<tr><td rowspan="8">操作方式</td><td rowspan="6">数据运算</td><td rowspan="5">基本数据运算</td><td rowspan="2">数值运算</td><td>整型运算：+、-、*、DIV、MOD（加、减、乘、取商、取余）</td></tr>
<tr><td>实型运算：+、-、*、/、**（加、减、乘、除、乘方）</td></tr>
<tr><td colspan="2">字符串运算：+（连接）</td></tr>
<tr><td colspan="2">逻辑运算：AND（与）、OR（或）、NOT（非）</td></tr>
<tr><td colspan="2">比较运算：>、>=（≥）、<、<=（≤）、=、<>（≠）</td></tr>
<tr><td colspan="3">扩展数据运算（略）</td></tr>
<tr><td rowspan="2">职能操作</td><td rowspan="3">基本职能操作</td><td colspan="2">输入</td></tr>
<tr><td colspan="2">赋值</td></tr>
<tr><td colspan="2">输出</td></tr>
<tr><td colspan="3">扩展职能操作（略）</td></tr>
</table>

所谓算法的职能操作，是指从外部环境向计算机系统提供数据的输入类操作、计算机系统内部加工数据（包括常数、变量、函数、表达式）的处理类操作、计算机系统向外部环境发送信息的输出类操作的总称。所谓算法的基本职能操作，则是指任何计算机语言都必须提供的这三类不同性质职能操作中最简单、最基本、最典型代表——输入操作、赋值操作与输出操作。学习并掌握好这三大基本职能操作，是算法设计的基础条件。

例 3-14　设计一个算法，求：$s=1*2+2*3+3*4+\cdots+9*10$ 的值

问题分析：本题实际上蕴含了两个内容，即：顺次产生自然数 k（$=1,2,\cdots,9$）；同时，依次将所产生出的 $k*(k+1)$ 的积逐个累加。由于判定是否已生成所需各自然数 k 的工作可先可后，故解决此求和问题，可用如下两个算法之一来实现，其程序流程图如图 3-3 所示。

算法	注释（求和算法 1）
第 1 步　开始	{算法自此开始}
第 2 步　s←0；k←1	{给累加变量 s、计数变量 k 赋初值}
第 3 步　当 $k<10$ 时，顺次执行第 4 步；反之（即当 $k\geqslant 10$ 时），则转到第 7 步	{当条件成立时，就反复执行第 4、5 步}
第 4 步　s←$s+k*(k+1)$	{累加求和}
第 5 步　k←$k+1$	{累计产生自然数}
第 6 步　返回第 3 步	{循环执行第 3 步}
第 7 步　行输出“$s=$”，s	{输出所求之和}
第 8 步　结束	{算法到此结束}

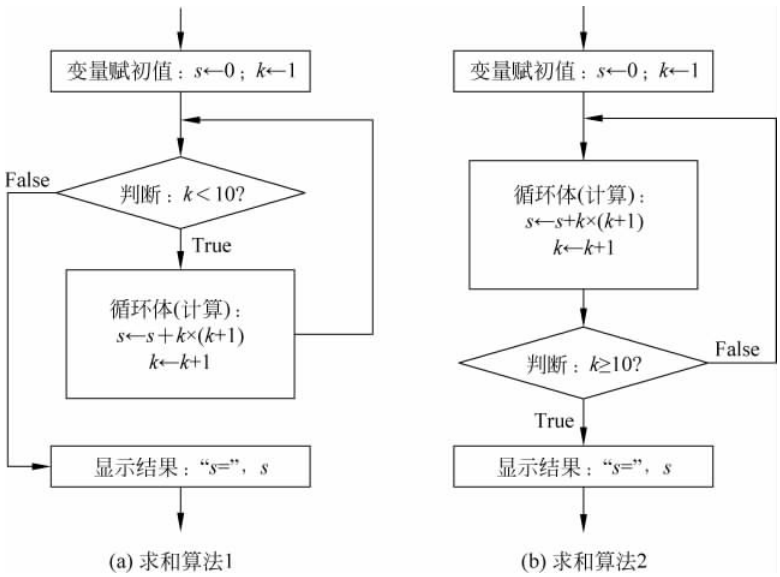

图 3-3　求和算法的程序流程图

算法	注释(求和算法 2)
第 1 步　开始	{算法自此开始}
第 2 步　s←0；k←1	{给累加变量 s、计数变量 k 赋初值}
第 3 步　s←s+k * (k+1)	{累加求和}
第 4 步　k←k+1	{累计产生自然数}
第 5 步　直到 k≥10 时，顺次执行第 6 步；反之(即 k<10 时)，则返回第 3 步	{直到条件成立时，才不反复执行第 3、4 步}
第 6 步　行输出"s ="，s	{输出所求之和}
第 7 步　结束	{算法到此结束}

有关说明：

输入操作是指由人向计算机直接提供所需数据并用相应变量存放它的职能操作。

赋值操作的功能是把"="右端的同种数据类型的值赋给它左端的变量，即在计算机加工、处理数据和信息全过程中，最常见的职能操作是把已获得的某型量(即某种类型的常数、变量、函数、表达式)的值，用一个同型变量存放它——赋值给该变量以备后用。赋值操作是计算机科学中最简单、最重要、最基础的职能操作。

通过赋值操作，人们能更清楚地看到，计算机科学中的变量与数学中的变量的确"既有密切联系，但更有本质区别"。即计算机科学中的变量，通俗地说，具有"新的不来，旧的不去；新的一来，旧的必失"的特性。换言之，一个变量一经获得其值(无论是由输入操作获取，还是由赋值操作获取的)，则可自行保持该值不变(而根本不管它在算法或程序中是否被使用过多次)，直到用新的值来取代它(使该变量另获得其新值)时为止。

输出操作是指，由计算机在打印机上自上而下、从左向右、逐字逐行地，或者在显示屏幕 CRT(或 x-y 平面绘图仪等)上可上可下、能左能右、逐点逐字地直观地发送并展示人所需信息的职能操作。

2）算法的流程控制与控制结构

流程控制与控制结构，是构成算法所设定的同构化控制方法的基础，是用以解决所论问题的同构化控制方式的描述手段。

（1）算法流程控制。

所谓流程，是指算法被执行时其（职能）操作实际执行路径（或流向）。一般说来，一个算法可能的执行路径通常都是非唯一的，而且正因为算法执行路径的多样性，才使算法能对所论问题的各种可能情形，都自有其相应的算法解题功能与之相适应。因而不同情况下的实际执行路径应当不尽相同、互不相混。这种以控制算法各流程来适应并解决所论问题的各种情况的执行路径控制方式，称为算法流程控制。

（2）算法基本结构。

虽然，算法形式千姿百态，算法流程纷繁万状，算法构造千变万化，但是，就其流程控制模式及其组成形式与构造而言，可以归结成顺序控制结构、选择控制结构、循环控制结构、并行控制结构和子算法控制结构五种。鉴于顺序、选择、循环结构是所有控制模式与控制结构的基础和核心，因此这三者统称为算法基本结构。

它们共同的本质属性是：有且仅有一个入口；有且仅有一个出口；无死块（或死程序段），即永远也无法执行到的操作块（或程序段）；无死循环块，即一经开始便永无休止地循环执行下去的操作块。必须强调指出，这四个本质属性缺一不可，否则，将破坏算法的结构化，而使之成为非结构化病态算法。

① 顺序结构——这是一种最简单而基础的算法基本结构，是任何从最简单到最复杂的算法都离不开的基本结构，其特点是：在本结构中，各操作按其出现的先后，依次顺序执行。

② 选择结构——这是一种最常用而重要的算法基本结构，是解决任何一个稍有复杂性问题的算法所必不可少的基本结构。其特点是：在本结构中，根据所给定的选择条件为真（即成立）与否，而从各可能的不同分支中选择执行其某一分支的相应操作，它显然有“多者必择其一，且仅择其一”的特性。

例 3-15 如解决“将任给两实数按从大到小排序”的算法就是含有选择结构（即该算法的第 3 步操作）的最简单算法示例。

问题分析：自然，只需按所给两实数 a 和 b 的大小性质来安排其先后排列顺序即可实现排序。同时，“$a>b$”或者“$a<b$”均可选作两数大小性质的判据。于是，由判据选取的不同而可得如下能解决同一问题的两个不同算法。

算法	注释（算法 1）
第 1 步　开始	{算法自此开始}
第 2 步　输入 a,b	{输入任给两实数}
第 3 步　如果 $a>b$，则行输出 a,b；	{$a>b$ 成立吗？若成立，则先 a 后 b 输出之；
否则，行输出 b,a	否则，先 b 后 a 输出之}
第 4 步　结束	{算法到此结束}

显然，若把上述算法中第 3 步改为如下所示，便可得解决同一问题的另一不同算法。

第 3 步　如果 $a<b$，则行输出 b,a；	{$a<b$ 成立吗？若成立，则先 b 后 a 输出之；
否则，行输出 a,b	否则，先 a 后 b 输出之}

③ 循环结构——这也是一种最常用而重要的算法基本结构，是解决绝大多数实际问题

的算法所必需的基本结构。其特点是：在本结构中，根据所给定的循环条件为真(即成立)与否，来决定如何重复地循环执行同一组操作。例 3-14 是含有循环结构(它的第 3～5 步)的典型算法示例。

6. 算法的基本特征

一个算法，必须具备有穷性、确定性、数据输入、信息输出和可执行性五大主要基本特征。凡不具备这五大基本特征者必非算法。

1) 有穷性

有始有终是算法最基本的特征。有始无终的过程绝不是算法。换言之，一个算法必须在它所涉及的每一种情况下，都能在实际执行有穷步操作之后而告结束。

应当注意，操作步骤有限的静态过程，未必能确保在动态情形下其实际执行次数的有穷性。因此，判别算法的有穷性应以动态算法为主要对象，并且动态算法的实际执行次数还必须不超过人们所能容忍的合理限度。

例 3-16 “并非算法，只是过程”的计数过程。

算法	注释
第 1 步　开始	{算法自此开始}
第 2 步　n←0	{初始化，给变量 n 赋初值 0}
第 3 步　n←n+1；输出 n	{边计数，边输出}
第 4 步　返回第 3 步	{准备继续不断地执行第 3 步}
第 5 步　结束	

乍一看来，这一过程仅有五步，且有“结束”，似为算法。该计数过程一旦开始执行，就永无休止之时。因而在动态执行中它并不具备有穷性，故并非算法。不过，只要对它稍加修改，便可消除其无穷性而使之成为一个算法。

2) 确定性

算法的每一步操作，其顺序和内容都必须唯一确定，而不得有任何歧义性。如果一个过程中的某操作步骤具有“既可这样，又可那样”一类模棱两可(甚至多可)的非确定性，即歧义性，那么它必非算法。务请特别注意：防止歧义性是计算机算法(或程序)设计中必须严格遵循的基本准则之一，即必须时刻注意使任何符号、任何数据、任何操作都具有其唯一确定的精确意义与确切位置。因此，在算法中一定要慎重地考虑各步操作的关系与顺序，并且一般情况下不能轻易调换，对此切不可掉以轻心。

3) 数据输入

一个算法，有零个或多个提供原始数据的输入操作(即输入操作个数≥0)；但一个输入操作，则至少有一个输入项。

4) 信息输出

既然算法是用来解决给定问题的，那么一个算法必须将人们所关心的信息输送出来供人们所用。也就是说，一个算法至少有一个已获得的有效信息输出操作(即输出操作个数≥1)；但一个输出操作，则可以没有输出项(即输出项为空的换行或空行输出操作)。

5) 可执行性

一个算法的任何一步操作都必须是可执行的(或称有效执行)，即必须是可以交给计算机付诸实施并能具体实现的基本操作。换言之，各步操作不仅在理论上而且在实践上，均能

由计算机(或由人模拟计算机而用纸和笔)执行有穷次即可实际完成。

3.1.3 数据结构初步

数据往往类型不同、形态各异、运算互别。但是,无论何种数据在算法设计中,其组织和构造都有其基本方式与共同规律。数据的组织和构造,不是孤立的,而是与算法的系统目标、操作执行者密切相关的;数据的构造和组织不是随意的,而是由算法的操作对象——数据的性质所制约和决定的。例如,流水账目与图书检索属系统目标不同;串行式计算与并行式计算属操作执行者不同;单行道各汽车站与铁路网各火车站的站间关系属数据不同,而它们的数据组织方式和方法也必然有所不同。一般说来,数据的组织和构造方式称为数据结构(Data structure),且可概述为:数据结构=系统目标+操作执行者+数据性质+数据组织。

研究数据的性质、规律及其组织和构造方式、方法的计算机科学分支也称为数据结构。常见的数据结构有数组、记录、文件、栈、队(列)、串、链表、树、图、数据库等。理论和实践都证明:如不了解施加于数据之上的算法,就无法决定如何组织和构造数据。算法的结构和构造,常常在很大程度上依赖于作为其基础的数据的组织与构造方式和方法。

例 3-17 以不同的数据结构,设计"在具有 $n(1\leqslant n\leqslant 10\ 000)$个无同姓名用户的电话簿中查找某人的电话号码"的两种算法。

问题分析:假设电话簿各用户数据(姓名、电话号码)为随机排列方式(例如:按申报安装电话时登记的顺序排列),则只好采用如下顺序查找的算法来实现。但是,假设已将电话簿各用户数据以姓名的拼音字母为序排列,则适宜采用如下二分(或称折半、中点)查找的算法来实现它。

算法	注释(电话用户顺序查找算法)
第 1 步　开始	{算法自此开始}
第 2 步　输入 n,$x\$$	{n 为用户总人数,$x\$$ 为待查者姓名}
第 3 步　$i\leftarrow 0$	{计数初始化}
第 4 步　当 $i<n$ 时,顺次执行第 5 步,反之(当 $i\geqslant n$ 时)则转到第 8 步	
第 5 步　输入 name$\$[i]$,teleph$\$[i]$	{用下标变量存放第 i 个用户的姓名、电话号码}
第 6 步　$i\leftarrow i+1$	{生成下一用户序号}
第 7 步　返回第 4 步	{循环执行第 4～6 步}
第 8 步　$i\leftarrow 1$	{准备查找第 1 位用户}
第 9 步　当 $i\leqslant n$ 时,顺次执行第 10 步,反之(当 $i>n$ 时)则转到第 12 步	{当未找到并且尚有未查对用户时}
第 10 步　如果 name$\$[i]=x\$$	{第 i 个用户正是待查找者吗?}
第 10-1-1 步　则行输出"已找到:",$x\$$,";电话号码:",teleph$\$[i]$	{"已找到"信息输出}
第 10-1-2 步　$i\leftarrow n+2$	{"已找到,强制中断查找"处理}
第 10-2-1 步　否则 $i\leftarrow i+1$	{生成下一用户序号}

```
第 11 步   返回第 9 步
第 12 步   如果 i=n+1                          {已查对完各用户吗?}
第 12-1-1 步   则行输出"查无此人:",x$          {"无此人"信息输出}
第 13 步   结束                                {算法到此结束}
算法                                           注释(电话用户二分查找算法)
第 1~7 步                                      {与上述算法相同,故略}
第 8 步   a←1; b←n                             {设置初始查找范围}
第 9 步   finding←0                            {设置"要查对"标志}
第 10 步   i←Int((a-b)/2)                      {生成当前查找范围中点}
第 11 步   如果 name$[i]=x$                    {第 i 个用户正是待查找者吗?}
第 11-1-1 步   则行输出"已找到:",x$,           {"有此人"信息输出}
               "; 电话号码:",teleph$[i]
第 11-1-2 步   finding←1                       {设置"已找到"标志}
第 11-2 步   否则如果 a>b                      {已查对完各用户吗?}
第 11-2-1 步   则行输出"查无此人:",x$          {"无此人"信息输出}
第 11-2-2 步   finding←2                       {设置"查不到"标志}
第 11-3 步   否则如果 x$>name$[i]              {应在当前中点之右吗?}
第 11-3-1 步   则 a←i+1                        {修改查找范围低端点}
第 11-3-2 步   否则 b←i-1                      {修改查找范围高端点}
第 12 步   直到 finding<>0 时,顺次            {直到无须查找时,停止循环;
           行第 13 步; 反之(finding=0 时)      否则执行第 10~12 步}
           则返回第 10 步
第 13 步   结束                                {算法到此结束}
```

显然,这两种查找算法的速度和效率相距甚远,而其数据构造方式亦明显有别。然而,如果把数据排列有序化的二分查找算法,盲目地移用于数据排列杂乱无章状况下的"二分查找",那就难免"南辕北辙,背道而驰"了。因为杂乱无章、不讲结构的数据并不符合二分查找算法对数据结构的要求。

因此,从这个意义上讲,数据结构必定是算法结构的基础,而算法则必须是数据结构与算法结构的统一。据此,著名计算机科学家、Pascal 语言发明者 N·沃恩教授曾提出"算法+数据结构=程序"的著名公式。这个公式的重要性在于:不能离开数据结构去抽象地分析程序的算法,也不能脱离算法去孤立地研究程序的数据结构,而只能从算法与数据结构的统一上去认识和把握算法。换言之,程序无非是在数据的某些特定的表示方式和结构形式的基础上,对抽象算法的计算机语言的具体表述。

3.1.4 程序设计方法初步

描述算法的工具,可以是自然语言,也可以是非自然语言(例如计算机语言)。鉴于自然语言有时难于做到意义唯一、叙述准确,故为了避免这种含混不清和防止歧义性,人们往往采用意义精确、唯一且已形式化了的计算机语言(或者自然语言与计算机语言兼而用之的准自然语言或准计算机语言)来描述一个算法。当用计算机语言来描述一个算法时,则其表述

形式就是计算机语言程序，简称计算机程序(Computer program)或程序。而设计程序的全过程，则称程序设计(Program design)。当一个算法的描述形式详尽到足以用一种计算机语言来表述时，则“程序”不过是瓜熟蒂落、唾手可得的产品。因而，算法是程序的源泉和基础，算法设计是程序设计的核心和关键。

如前所述，程序是算法与数据结构的统一。然而，“无规矩不成方圆，无方法不成程序”。如果没有一定的程序设计方法，就不能实现算法与数据结构的统一，因为它们二者的这种统一，既不是天生具有的，也不是天然兑现的。

1. 程序设计方法学的产生

20世纪60年代末至20世纪70年代初，是程序设计观念发生重大变革的重要时期。在此之前，人们虽已认识到程序是算法和数据结构的统一，但却把程序设计简单地视为是用有关计算机语言来反映这一全过程的某种艺术。然而，在这一时期，随着大型软件系统(例如，操作系统、数据库、管理信息系统等)的出现，产生了所谓的“软件危机”，即一个大型软件系统的研制，通常要耗用大量的人力、物力、财力和时间，但当时在传统的非结构化程序设计观念(只重程序设计效率、忽视程序清晰、不讲程序结构)的束缚下所研制出来的这类软件产品，却常常潜伏着诸如可靠性差、错误较多、维护和修改相当困难等隐患和弊病，它们随时都可能突然给人们带来意想不到的重大损失甚至灾难。

“软件危机”震撼了计算机界，也冲击了人们对程序设计的传统观念，促使人们认真反省和研究程序设计中一系列根本性的基本问题。例如，程序的基本结构是什么？程序设计应当采用什么方法？是否应当“算法设计先于程序编码(Design before coding)”？是否必须做到程序“清晰第一，效率第二”？程序设计技术与方法如何规范化和工程化？等等。

正是在这种历史背景下，人们争论、探索、研究、实践和总结，终于形成了一整套关于如何正确进行程序设计的理论和方法，并升华成一门带有艺术性特点的新学科，即计算机科学重要分支——程序设计方法学。其要害和核心是已得到证明的结构化原则，即结构化算法的控制结构必须并且只能由顺序结构、选择结构、循环结构这三大基本结构所构成；反之亦然。程序设计方法学主要包括下述重要内容。

(1) 如何设计数据结构的方法；

(2) 算法构造、实施与证明方法；

(3) 程序连接方法；

(4) 程序调试、测试及验证方法；

(5) 程序设计技术与方法(例如：结构化程序设计技术与方法、自顶向下设计技术与方法、逐步求精技术与方法、模块化设计技术与方法等等)。

2. 程序设计表现技术

提高开发软件产品(或信息系统)生产率，必须致力于系统分析、系统设计、程序设计、调试分析、运行维护各阶段的生产率的提高。大力研究和采用构造完善、使用简便的程序设计表现技术，一直是提高程序设计阶段生产率的基本手段与重要内容之一。

程序设计表现技术无非是两大类：以图形为描述基础的图形表现技术和以语言为描述基础的语言表现技术。其中图形表现技术有许多种，例如，Jackson图、SPD图、PAD图等。本书在此主要介绍一种较为常用的程序设计表现技术N-S图(N-S chart)。

1) 程序流程图与 N-S 图

程序流程图(Flow Diagram)是最古老的设计表达工具之一,早在 20 世纪 40 年代,V. Neumann 等人就已指出"编码(Coding)始于画流程图"。随着结构化程序设计方法的普及,程序流程图在描述程序逻辑时的随意性和灵活性,恰恰变成了它的缺点。1973 年,Nassi 和 Shneiderman 发表了题为"结构化程序的流程图技术"的文章,提出用方框图(Block Diagram)来代替传统的程序流程图,根据这两位创始人的名字,许多人把它简称为 N-S 图,又称盒图。

2) 用 N-S 图表示的三种基本控制结构

有关说明:

(1) 顺序结构,如图 3-4(a)所示。其特点是:在本结构中,各操作块(即算法中有限个线性连续操作的集成块,简称块,它所对应的则是程序段。即程序中有限个线性连续指令或语句的集成段)按照各自出现的先后顺序,依次逐块执行。

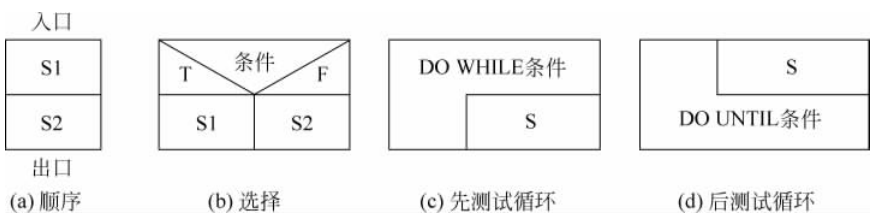

图 3-4 N-S 图的基本控制结构示意

(2) 选择结构,如图 3-4(b)所示。其特点是:在本结构中,根据所给定选择条件为真(即成立)与否而从各可能的不同分支中选择执行其某一分支的相应操作,并显然有"多者必择其一,且仅择其一"的特性。

(3) 循环结构,如图 3-4(c)和(d)所示。

图 3-4(c)是循环结构最常用的第一种典型形态,它的循环体执行方式为:当给定循环条件为真时,循环结构就反复执行其循环体(即 S 块);反之(即当该循环条件为假时),则终止执行其循环体,而去执行该循环结构的后继块。显然,若它的循环条件初始值为假时,则并不执行其循环体,故它的循环体执行次数最少可为零(即并不执行其循环体)。

图 3-4(d)是循环结构最常见的第二种典型形态。它的循环体执行方式为:除首次必然无条件地执行其循环体外,其他情况下均与当型循环结构循环体执行方式相反,即直到所给定循环条件为真时,才终止对其循环体的执行,而去执行该循环结构的后继块。显然,即使它的循环条件初始值已为真,也要执行一次循环体,故它的循环体执行次数至少是 1。

3) N-S 图的主要特点

众所周知,结构化程序设计主张把程序的逻辑结构限制为顺序、选择和循环等几种有限的基本结构,以提高程序的易读性和易维护性。N-S 图的主要特色就是只能描述结构化程序所允许的标准结构,程序的层次结构清晰,有利于养成良好的、规范的程序设计风格。

例 3-18 程序流程图和 N-S 图的实例对比,"在一组数中找出其中的最大数"。如图 3-5 和图 3-6 所示。

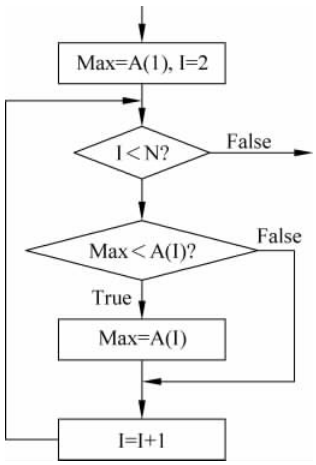

图 3-5　传统程序流程图表述

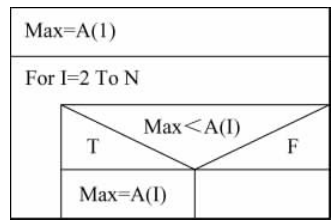

图 3-6　N-S 图表述

3.2　银行常用程序设计语言

按照软件工程的观点，计算机程序设计语言的发展至今已经历了 4 代 3 个阶段，如图 3-7 所示。其中高级语言种类繁多，从软件工程的角度，可以把它们分为基础语言、结构化语言和面向对象语言三大类。

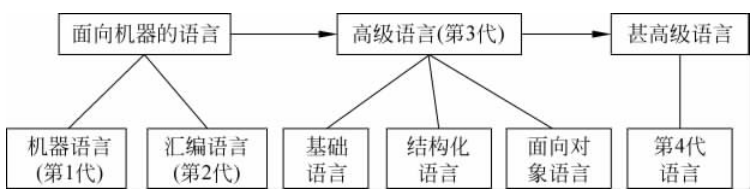

图 3-7　程序设计语言的发展和分类

(1) 基础语言。FORTRAN、COBOL 和 BASIC 是这类语言的代表，之所以称它们为基础语言，是因为它们都有较长的使用历史，有大量已开发的软件，今天仍拥有广大的用户。

(2) 结构化语言。20 世纪 70 年代以来，在结构化程序设计影响下，先后出现了一批常用的结构化语言，Pascal、C 等语言就是其中著名的代表。

(3) 面向对象语言的代表是 C++语言和 Java 语言。C++是从 C 语言进化而来，其程序结构的本质与 C 语言是一致的，因此它既可进行结构化程序设计，也可进行面向对象程序设计。Java 语言是当今流行的新兴网络编程语言，它不仅能够编写小应用程序实现嵌入网页的声音和动画功能，而且能够应用于独立的大中型应用程序，其强大的网络功能能够把整个 Internet 作为一个统一的运行平台。

3.2.1 COBOL 语言概述

1. COBOL 语言的发明者

COBOL 语言是已故美国海军少将,Grace Hopper(格蕾丝·霍波)博士于 1959 年开发成功的。COBOL 语言从诞生至今已经走过了半个多世纪,但它依然被应用于软件开发领域,不愧是高级语言领域的一棵"常青树"。

被誉为"计算机编译软件之母"和"世界上第三位程序员"的葛丽丝是位传奇女性,1946 年,她在发生故障的 Mark Ⅱ 计算机的继电器触点里,找到了一只被夹扁的小飞蛾,正是这只小虫子导致了机器的故障而停机,她顺手将飞蛾夹在工作笔记里,并诙谐地把程序故障称为"Bug"。Bug 的意思是"臭虫",而这一奇怪的称呼,后来演变成计算机行业的专业术语,即故障。虽然现代计算机再也不可能夹扁任何飞蛾,但是,人们还是习惯地把排除程序故障叫作 Debug(除虫)。

20 世纪 60 年代,硬件价格极其昂贵,为了节省宝贵的内存空间,葛丽丝博士苦思冥想采用了 6 位数字来存储日期,即年、月、日各两位。随着 COBOL 语言的影响日愈扩大,这一习惯做法被人们一直沿用下来,直到 2000 年前,这种为了节省存储空间的习惯用法居然导致了殃及全球的、危害巨大的"千年虫"(Y2K),这当然是葛丽丝博士昔日所始料不及的事。

霍波一生没有子女,但她非常热爱孩子。她深知,自己的成功源于刻苦的努力和自小受到的良好教育,所以她特别重视对年轻人的教育。她曾经为青年学生作过近千场演讲,讲述计算机的未来,她将在讲演中获得的纪念品和酬金都无偿捐献给了她热爱的海军。她常常对人说:"与其说我的最大贡献是发展了程序设计技术,不如说我培养了大批程序设计人才。"

为表彰她对计算机发展和美国海军的贡献,美国劳伦斯伯克利国家实验室研发的超级计算机 HOPPER 系统,以她的名字 Grace Hopper 命名,美国海军的一艘驱逐舰也被命名为"格蕾丝号",加利福尼亚海军数据处理中心也改称"霍波服务 Grace 博士"。1971 年,为了纪念现代数字计算机诞生 25 周年,美国计算机学会 ACM 特别设立了"格蕾丝·霍波奖",此奖颁发给当年最优秀的 30 岁以下的青年计算机工作者。因此,"格蕾丝·霍波奖"成了全球电脑界"少年英雄"的标志。

2. COBOL 语言的发展概况

COBOL 是 Common Business Oriented Language 的缩写,即面向事务处理的通用语言,或称通用商业语言。实际上,COBOL 不仅是商业数据处理的理想语言,而且广泛应用于数据管理领域,例如财会工作、统计报表、计划编制、情报检索、人事管理等。因此 COBOL 语言也被称为"用于管理的语言"。

在计算机的应用领域中,数据处理(Data processing)是应用最广泛的一个领域。数据处理的日益广泛应用,要求人们设计出能满足实际数据处理工作中各种要求的一种计算机语言,COBOL 语言就是在这种形势下应运而生的。

1959 年 5 月,美国国防部召开了一个有政府机关、企业、计算机厂家代表参加的会议,各方面都认为有必要设计出一种数据处理专用的计算机语言。会上确定了常设机构,以研究这种语言,这个会议称为 CODASYL(Conference on Data Systems Languages),意为数据

系统语言会议。1959年12月提出了世界上第一个COBOL语言文本，次年4月由美国政府正式发表，因此称COBOL-60。后来进一步扩充和完善，出现了扩展COBOL-61，它们为后来的版本提供了基础。

1965年美国出现了更完善的版本，即COBOL-65，但直到1968年8月才由美国国家标准化协会(American National Standard Institute，ANSI)通过批准了这个语言的标准版本，作为各厂家的依据。这就是ANSI COBOL X3.23-1968。1972年国际标准化组织ISO决定把它作为ISO COBOL-72国际标准COBOL文本，该文本已为许多国家所承认。此后，又分别出现了多个版本，如ANSI COBOL-74、ISO COBOL-78、COBOL-85、COBOL-2002等。

3. COBOL语言的特点

COBOL语言的主要特点有：

1) 最适于数据处理领域

所谓数据处理是指对大量数据的收集、统计、分类和加工。例如企业管理、库存管理、报表统计、账目计算、信息情报检索等方面的应用都属于数据处理。

数据处理的特点是：算术运算量少而逻辑处理多；输入输出量大；数据间存在着一定的逻辑关系(数据项间有清晰的层次关系，例如职工工资包括应发工资、扣除部分、实发工资等几部分，应发工资又包括基本工资、附加工资等)；大量的分类排序(如按年龄大小排名单、按受教育程度分类……等)；对打印报表要求较高、多样化等等。

在企业(如银行、商业、工厂)和其他部门(如领导机关、业务处理部门)的管理工作中，一般并无很复杂的计算公式，不要求太高深的数学基础，但是数据处理的量很大。

COBOL正是针对数据处理要求而设计的。COBOL所处理的问题具有数据繁多而运算简单的特点，COBOL也有加、减、乘、除、乘方等运算以及表达式的概念，但这些不是COBOL的重点。它的主要功能是描述数据结构和分析处理大批量的数据。

COBOL对数据的处理过程与人工处理的过程是相似的，即与人们的思维过程比较接近，因此，一般的管理人员是比较容易理解和掌握COBOL语言的。

2) COBOL比较接近于自然语言(指的是英语)

COBOL程序看起来很像一篇用英语写的文章。例如：

ADD A TO B 表示A + B → B，即A加B，结果放在变量B中

MOVE C TO D 表示将变量C的值传送给变量D中

COBOL大量采用普通英语词汇和句型，学过英语的人看COBOL程序感到通俗易懂。也就是说它的特点是成文自明。

3) 通用性强

由于COBOL语言的标准化程度较高，不同厂家生产的计算机系统所提供的COBOL是COBOL标准的全集或一个子集，一个计算机上的COBOL程序向另一计算机系统上移植，是比较容易实现的。

4) COBOL的结构严谨，层次分明

每个COBOL程序分四大部分(称为部，Division)，每个部下面又分为若干节(Section)，节下面又分为若干段(Paragraph)。每一部分都有固定的样式，这个特点使初学者比较容易通过模仿别人程序中的有关部分，从而较快地写出自己的程序。

5) COBOL 的缺点是比较烦琐

如同中国古代的八股文一样,程序无论大小简繁,一律都要写齐四大部分,对每个部分都要进行必要的定义和说明,因此源程序显得比较冗长。

据国外统计,在大、中型计算机系统上运行 COBOL 程序所占用的计算机时间为全部机时的一半以上,超过了任何一种其他语言,是目前世界上使用得最多的一种计算机语言之一。

4. 最简单的 COBOL 程序介绍

为了使初学者从一开始就了解 COBOL 源程序的格式以及它的组成,建立起一个整体的概念,我们先介绍两个最简单的 COBOL 源程序。

例 3-19 使计算机在指定的外部设备(终端显示器或打印机)上显示(或打印)出字符串"This is a COBOL program",然后停止运行。

```
列:1     6     7     8          12
                     IDEN       TIFICATION          DIVISION.(标识部)
                     PROG       RAM ID. EXAM1.        (程序标识段)
                     ENVI       RONMENT             DIVISION.(环境部)
                     DATA                           DIVISION.(数据部)
                     PROC       EDURE               DIVISION.(过程部)
                     S.         DISPLAY 'This is a COBOL program'.
                                STOP RUN.
```

说明:程序倒数第 2 段的"S."是段名,在本例中过程部只包含一个段,即 S 段;在 S 段中有两个句子,每个句子以句点"."和空格结束。

例 3-20 将 A 和 B 的值相加,其结果放在 B 中。

相关说明:

(1) 这个程序的程序名是"EXAM2"。

(2) 在本例中数据部下面有一个 WORKING-STORAGE SECTION(工作单元节,或称工作存储节),用它来描述程序中用到的中间工作单元。今有两个数据项 A 和 B,用"PICTURE IS 9(3)"来说明(描述)A 和 B 的类型是数值型的,"9"代表数值型,"(3)"代表数据长度为三位,即 A 和 B 的值是三位整数。

(3) 在过程部中,只有一个 S 段,在 S 段中有两个句子,每个句子以句点和空格为结束标志。第一个句子中包含四个语句,每个语句完成一个特定的操作。ACCEPT A 和 ACCEPT B 是从指定的外部设备上先后接收两个数值给 A 和 B(指定的外部设备可以是控制台或终端的键盘)。

```
列:1     6     7     8          12
                     IDEN       TIFICATION          DIVISION.(标识部)
                     PROG       RAM ID. EXAM2.        (程序标识段)
                     ENVI       RONMENT             DIVISION.(环境部)
                     DATA                           DIVISION.(数据部)
                     WORK       ING-STORAGE SECTION.  (工作单元节)
```

续表

列：1	6	7	8	12	
			77 A	PICTURE IS 9(3).	（对变量 A 进行描述）
			77 B	PICTURE IS 9(3).	（对变量 B 进行描述）
			PROC	EDURE	DIVISION.（过程部）
			S.	ACCEPT A	（输入 A 的值）
				ACCEPT B	（输入 B 的值）
				ADD A TO B	（A + B→B）
				DISPLAY A,B.	（显示 A 和 B 的值）
				STOP RUN.	（停止运行）

3.2.2 C 语言概述

1. C 语言的发展简史

C 语言是目前世界上最广泛使用的计算机高级程序设计语言。C 语言既可以编写计算机系统软件，也可以编写各种应用软件，所以在数百种计算机语言中，C 语言仍然是目前最流行、最受欢迎的计算机语言。

最初，C 语言是为编写 UNIX 操作系统而设计的。在 C 语言产生之前，操作系统主要是用汇编语言编写的，但是汇编语言依赖计算机硬件，程序的可读性和移植性都比较差。人们一直在寻找一种很接近计算机硬件的高级语言，C 语言就是在这种情况下应运而生的。

C 语言是在 B 语言的基础上发展起来的，它的根源可以追溯到 ALGOL 60。1960 年出现的 ALGOL 60 是一种面向问题的高级语言，它离硬件比较远，不易用来编写系统程序；1963 年英国剑桥大学在 ALGOL 语言基础上增添了硬件处理能力后，推出了 CPL (Combined Programming Language)语言，CPL 语言在 ALGOL 60 的基础上接近硬件一些，但规模比较大，难以实现。1967 年英国剑桥大学的 Matin Richards 对 CPL 语言作了简化，推出了 BCPL 语言(Basic Combined Programming Language)。1970 年美国贝尔实验室的 Ken Thompson 以 BCPL 语言为基础，作了进一步简化，设计出了很简单的 B 语言(取 BCPL 的第一个字母)，并用 B 语言编写了第一个 UNIX 操作系统，在 PDP-7 上实现。但 B 语言过于简单，功能有限。1972 年至 1973 年间，贝尔实验室的 D. M. Ritchie 在 B 语言的基础上设计出了 C 语言(取 BCPL 的第二个字母)。C 语言既保持了 BCPL 和 B 语言的优点，即精练，接近硬件，又克服了它们的缺点，即过于简单，数据无类型等。

最初的 C 语言是为描述和实现 UNIX 操作系统提供一种工作语言而设计的。后来，C 语言多次作了改进，直到 1975 年 UNIX 第 6 版公布后，C 语言的突出优点才引起人们的普遍注意。1983 年，美国国家标准化协会(ANSI)根据 C 语言问世以来各种版本对 C 的发展和扩充，制定了新的标准，称为 ANSI C。1987 年又公布了新标准 87 ANSI C。目前流行的各种 C 版本都是以这个标准为基础。

现在使用的各种 C 语言编译系统虽然基本部分是相同的，但也有一些不同。在微机上使用的有 Microsoft C，Turbo C，Quick C 等，它们的不同版本又略有差异。因此，在使用一个系统之前，要了解所用的计算机系统配置的 C 编译系统的特点和规定。

2. C 语言的特色

C 语言能得到快速发展,受到用户的广泛欢迎,是因为它独具特色。

1) C 语言与其他语言的比较

作为高级语言的 C 语言常被称为中级计算机语言,这并不意味着 C 语言功能差难以使用,或不像某些高级语言那样完善。C 语言被称为中级计算机语言,只是意味着它把其他高级语言的基本结构与低级语言的实用性结合了起来。

(1) C 语言与汇编语言比较。

C 语言允许对位、字节和地址进行操作(指针),这三者是计算机基本的工作单元,在编制系统程序时要经常用到,所以它适用于写系统程序。由于汇编语言是非结构化语言,含有大量的跳转、子程序调用以及变址,这种结构的缺陷使得汇编语言程序难以读懂,难以维护,也不能移植。而 C 语言的结构化、模块化克服了汇编语言难读难维护的缺点。C 语言又具有汇编语言的功能,目标代码长度也差不多,效率几乎与汇编语言相近,且具有很好的可移植性。

(2) C 语言与其他高级语言比较。

C 语言有丰富的运算符 34 种,其中有很多运算符对应于常用的机器指令,比如"++"等运算符可直接编译成机器代码,使用起来简单精练。

C 语言有多样化的表达式类型,因此可以将 C 语言说成是表达式语言。C 语言中的表达式几乎无处不在,而在其他高级语言中,表达式则要受到很大的限制。

C 语言的数据类型丰富,具有现代语言的各种数据结构。C 语言的数据类型有整型、实型、字符型、数组、指针、结构体、共用体等。它们能用来实现各种复杂的数据结构,如链表、树、队列、栈等。其中指针类型使参数传递简单、迅速,节省内存。

C 语言的输入输出使用的是数据流,而其他高级语言使用的是记录。

C 语言程序生成的机器代码质量高,内存占用少,运行速度快,程序执行效率高。这也是衡量一种语言优劣的重要指标之一。

2) C 语言是结构化语言

C 语言是以函数为模块来编写源程序的,所以 C 程序是模块化的。C 语言具有结构化的控制语句,如 If-Else 语句、While 语句、Do-While 语句、For 语句等,因此是结构化的理想语言,符合现代编程风格的要求。

3. 简单的 C 程序介绍

C 语言是一种结构化的程序语言,通过掌握 C 语言程序设计来理解结构化程序设计的思想。下面介绍两个简单的 C 程序。

例 3-21 屏幕输出一条信息,程序如下:

```
main ()
{
  Printf ( "This is the first program . \n" )
}
```

运行结果:

```
This is the first program .
```

其中 main()被称作“主函数”,在任何一个 C 程序中都必须有一个且只有一个 main()函数。在 main()函数中有很多 C 语句,用一对大括号括起来。在本例的程序中,共有一条 C 语句。这条 C 语句是一个输出函数,作用是向屏幕输出信息。它将双引号之间的内容原样输出到屏幕。“\n”是换行符,它的作用是将光标移到下一行的开始处。

例 3-22 计算两个数的乘积,程序如下:

```
main ()
{
  int x,y,result;                              /* 定义变量 */
  x = 14:y = 12;                               /* 给变量赋值 */
  result = x * y;                              /* 求积 */
  printf ( " \n The result is % d ",result ); /* 输出两个数的乘积 */
}
```

运行结果如下:The result is 28

程序中,/* …… */表示注释,注释部分不参与也不影响程序的运行,这些注释只是用来帮助人们阅读和理解程序的,注释部分可以加在程序的任何部分。第 3 行是变量定义,说明了 3 个整型变量;第 4 行是赋值语句,给变量 x 和 y 赋值;第 5 行将 $x * y$ 的结果送入变量 result 中;第 6 行是将结果输出到屏幕,“%d”表示在这个位置上将用一个整型数值代替,本程序中是用整数 28 来代替%d 这个位置。

3.2.3 面向对象程序语言概述

1. 面向对象程序设计

面向对象的风范是在结构化程序设计概念和数据抽象的基础上建立起来的。基本的变化是面向对象的程序是围绕被操作的数据而设计的,而不是围绕操作本身。计算机执行的工作被称为数据处理。数据和动作在一个基本的水平上连接,每一个都要求另一个有目的。面向对象程序使这个关系更为明显。

面向对象程序设计把数据结构和操作结合起来,而这正是我们如何考虑世界的方法。我们通常把给定的对象类型和一组特定的动作结合起来。例如,我们知道汽车有轮子,可以移动,可以通过方向盘来改变方向。类似地,我们知道树是有本质茎和叶子的植物。因此我们自然认为用汽车做的事不能用树来做,反之亦如此。我们不能操纵树的方向,而给汽车浇水,也不会使汽车长大。面向对象程序设计指定其数据类型的性质和行为,这使我们可以确切地知道各种数据类型的作用及如何使用。

在面向对象程序中可以在既相似又有区别的数据类型之间建立关系。人们自然地将事物分类,我们常常把新概念与已有概念联系,而且可以基于事物之间的关系上进行演绎。通常我们用一个树状结构来概念化世界,后继的细节层建立在前面的一般原则之上。面向对象程序以同样自然的方法工作,使新的数据/操作框架建立在已有的框架基础上,在增加新功能的同时结合基础框架的功能。

面向对象程序设计不是要摒弃结构化程序设计方法,相反,它是在吸收结构化程序设计优点的基础上,引进了一些全新的、强有力的概念,从而开创了程序设计的新天地。面向对象程序设计方法把可重用性视为软件开发的中心问题,通过装配可重复使用的软构件来生

产软件。

2. 面向对象技术的应用和发展

20世纪70年代末,面向对象方法学的一些基本概念已在系统工程领域内萌发出来,如对于系统中的某个模块或构件可表示为问题空间的一个对象或一类对象。到了20世纪80年代,面向对象的程序设计方法得到了很快的发展,并显示出其强大的生命力。因而使得面向对象技术在系统工程、计算机、人工智能等领域得到了广泛的应用。20世纪90年代以来,随着面向对象技术的不断成熟,其应用向更高的层次、更广泛的领域发展。

如同结构化方法是从结构化程序设计语言发展而来,面向对象方法也是由面向对象程序设计语言发展而来。程序设计语言的发展过程是抽象程度不断演变和提高的过程。1967年第1个具有面向对象特征的SIMULA语言问世,标志着面向对象方法的诞生。20世纪80年代初Xerox PARL推出Smalltalk-80,使得面向对象引起了广泛的注意。而由贝尔实验室设计的C++,使得面向对象的编程语言(Object Oriented Programming language,OOPL)走向大众化。然而,在面向对象(Object Oriented,OO)方法从程序设计语言开始,并不意味着它只是一种程序设计方法,甚至是某种程序设计语言,它的意义要深远得多。

在OOPL蓬勃发展的20世纪80年代,人们基于以往已提出的有关信息隐蔽和抽象数据类型等概念、以及由Moudula-2、Ada和Smalltalk等语言所奠定的基础,再加上客观需求的推动,逐步地发展和建立起较为完整的面向对象的软件系统的概念和机制。而将这些基本概念和运行机制应用到其他各个领域,就得到了一系列相应领域的面向对象的技术和应用。

1) 面向对象分析(Object Oriented Analysis,OOA)

系统分析过程是对一个问题空间的研究,OOA模型描述了表示某个特定应用领域中的对象,以及各种各样的结构关系和通信关系。

2) 面向对象设计(Object Oriented Design,OOD)

从OOA到OOD是一个逐渐扩充模型的过程。在分析阶段,把系统分解成实体和关系,而设计则是解决这些实体和关系如何实现。

3) 面向对象语言(Object Oriented Language,OOL)

在这种语言中,可以把数据和处理数据的过程结合为一个对象。对象既可以像数据一样被处理,又可以像过程一样描述处理的流程和细节。

4) 面向对象数据库(Object Oriented Data Base,OODB)

把对象作为存取和检索的单位,把传统数据库的数据定义语言和数据操作语言融为一体,这种概念也是构成语义数据库和知识库的基础。

5) 面向对象的智能程序设计(Object Oriented Intelligent Programming,OOIP)

把知识系统中的智能实体作为对象,一个对象所具有的知识是该对象的静态属性,一个对象所具有的知识处理方法则是该对象的智能行为的体现。

6) 面向对象的体系结构(Object Oriented Architecture,OOA)

能支持对象的描述、存储和处理的计算机体系结构。一些新的计算机体系结构,如某些多处理系统、神经网络计算机及连接机制等,均试图支持面向对象的程序设计和软件系统的概念。

面向对象技术自产生以来已经得到了很大的发展，并且已在计算机科学、信息科学和系统科学中得到了有效的应用，显示出其强大的生命力。面向对象方法学的最基本特点是尽可能地模拟人的思维方式，并且和方法学相适应的面向对象技术也提供了这样的可能性，即可以随着对某个系统的需求逐步具体的过程而逐步地设计和实现这个系统。

未来，面向对象技术将会在更深、更广、更高的方向上取得进展。

(1) 更深的方向。如面向对象技术的理论基础和形式化描述；用面向对象技术的概念设计操作系统。

(2) 更广的方向。如面向对象的知识表示；面向对象的仿真系统；面向对象的多媒体系统；面向对象的虚拟现实系统。

(3) 更高的方向。如从思维科学的高度来丰富面向对象方法学的本质属性，突破现有的面向对象技术的一些局限、研究统一的面向对象的范式等。

3. 面向对象语言的分类

一个被称作是面向对象的语言应具有哪些特点呢？美国布朗大学的 P. Wegner 对面向对象程序设计语言进行了分类分析，认为面向对象语言按对对象、类、继承性的支持程度分为基于对象的语言，基于类的语言和面向对象的语言。

1) 基于对象的语言

一个基于对象的语言，是指它支持对象作为语言特征。对象的概念出自不同的语言有着根本不同的表示。

在基于对象的语言中，对象具有一个操作集合和一个“记忆”操作结果的局部共享状态，它可以作为一个模块来看待。Ada 和 Moudula-2 中都提供了数据抽象和抽象数据类型的机制，它们的模块和程序包满足对象的封装性，但其对象不具有某个类，也无继承的概念，在支持抽象数据类型时，缺乏灵活性和可扩充性。

2) 基于类的语言

在基于类的语言中，类是对具有相同语义特性的一组对象的抽象，它具有创建类实例的功能，为方法(过程或操作)提供存储空间。

在基于类的语言中，类可看成是功能很强的软件部件，类的数据包含在类的私有变量中，只有在类之内才可以访问，从外部是不可访问的。因此，内部数据结构和过程的改变不会影响其他软件模块的实现。

3) 面向对象的语言

面向对象语言中的类除具有基于类语言的语义之外，其独特的语义就是具有继承机制。它使面向对象语言具有类层次结构，并使类层次结构中的下层子类可以继承其上层超类的语义特性。除继承性外，多态性也是面向对象语言的突出优点。

由上面讨论，我们可以看出，基于类的语言是基于对象的语言的真子集，而面向对象的语言又是基于类的语言的真子集。对象用于把操作及其所处理的数据组合在一起，提供面向数据的程序设计原则。类用于管理对象的集合允许把对象作参数传递，赋给变量以及组织成结果。类继承用于组织类的集合，使类的层次关系可以描述应用领域。因此面向对象语言可以在语言中实施对对象与类两者的管理，从而为应用的设计与实现提供统一的机制。

练习题

1. 解题的基本模式有哪两种？其特点是什么？
2. 分析问题有哪些主要的环节？举例说明各自的特点是什么？
3. 举例说明何谓算法？
4. 算法设计中的关键基本要素是什么？
5. 举例说明算法有哪些类型的职能操作？
6. 举例说明何谓算法基本结构？
7. 算法的五大主要基本特征是什么？
8. 何谓数据结构？
9. 根据 N-S 图的要求画出本章例 3-17 的结构框图。
10. 简述 COBOL 语言的主要特点。
11. 简述 C 语言的主要特点。
12. 简述面向对象语言和技术的主要特点。
13. 绘制程序流程图，程序功能是计算 1＋(1＋3)＋(1＋3＋5)＋…＋(1＋3＋5 ＋…＋39)，并将最终结果在屏幕上输出显示。要求结果存放于变量 sum 中。