C程序设计

主编 王苗 徐建民

河北大学出版社

C 程序设计

主　编：王　苗　徐建民
副主编：张宇敬　杨学全
丁红伟　赵文胜
齐俊玉

河北大学出版社

责任编辑：韩　勇
封面设计：赵　谦
责任印制：李晓敏

图书在版编目(CIP)数据

C 程序设计/王苗，徐建民主编. 保定：河北大学出版社，2003.7
ISBN 7-81028-944-6

Ⅰ.C… Ⅱ.①王…②徐… Ⅲ.C 语言 程序设计 高等学校—教材 Ⅳ.TP312

中国版本图书馆 CIP 数据核字(2003)第 058854 号

出版者：河北大学出版社(保定市合作路 88 号，邮编 071002)
经　销：全国新华书店
印　刷：河北大学印刷厂
规　格：1/16(787mm×1092mm)
印　张：17.25
字　数：400 千字
版　次：2003 年 7 月第 1 版
印　次：2003 年 7 月第 1 次
书　号：ISBN 7-81028-944-6/TP·53
定　价：24.00 元

前　言

C 程序设计是高等院校计算机及其相关专业重要的必修课之一，是数据结构、操作系统等专业课的先修课，同时程序设计又是计算机专业学生必须掌握的基本技能。基于技能训练的基本思路，在本书的编写过程中主要遵循了以下原则：

(1)知识讲授和技能培养并重的原则：程序设计不仅是一种知识，同时还是一种技能，因此程序设计课程的教学内容在注重知识讲授的同时，还应注重学生技能如编程能力、编程风格、团队协作等方面的培养。

(2)循序渐进的原则：知识的掌握和技能的训练都是一个循序渐进的过程，学生学习程序设计的知识和技能的培养应该分三步走：掌握基本知识，能够读懂实例，模仿编写程序，逐步达到能够独立编写程序的目的。

(3)分解与综合的原则：人们认识复杂事物的一个基本方法是分解，软件开发的过程就遵循了这一原则。对学生来说，一门新课相当于一个需要接受的新事物，采用“分解与综合”的方法可以使他们感觉到学习更容易。因此本教材内容的安排基本遵循“模块大小适中”的原则，即每一章、一节乃至一个知识点都尽可能保持适中，将难点适当分解，便于学生掌握。

(4)规范化的原则：程序设计是软件开发过程的一个基本阶段，软件工程的基本思想之一是规范化文档。一开始就注意培养学生良好的程序设计习惯十分重要。所以本书遵循的第四个原则是每一例题程序都尽可能体现良好的程序设计风格。

(5)建构主义的原则：学生是学习的主体，学生主动参与教学过程对学生的学习十分有益，因此本教材内容的安排适当强调学生的参与。

本教材的内容包括 9 章，分别讲述了基础知识、数据类型及数据的输入输出、运算符及表达式、结构化程序设计及控制语句、函数及变量的存储类别、编译预处理、构造数据类型、指针的综合应用和文件等知识。

编写出一本优秀的教材是一件非常不容易的工作，涉及到诸多的因素。由于作者水平、精力所限，再加上时间紧迫，书中难免有不足甚至错误，恳请读者批评指正。

衷心感谢所有关心本书的师长和朋友。

作　者

2003 年 4 月

内容提要

程序设计是计算机及相关专业的一门重要的必修课，程序设计不仅仅是计算机及相关专业学生必备的一种知识，更是一种必须掌握的基本技能。因此，本书在介绍C语言基本概念和知识的基础上，更强调程序设计技能的训练。

全书共分9章，分别讲述了基础知识、数据类型及数据的输入输出、运算符及表达式、结构化程序设计及控制语句、函数及变量的存储类别、编译预处理、构造数据类型、指针的综合应用和文件的基本知识。

本书文字讲述力求准确、简练，强调知识的层次性；例题习题选用讲究、丰富，强调编程技能的培养；在内容安排上本书遵循了“难点分解”和逐步由简到难的原则，减少学生学习的难度。本书既可作为计算机及相关专业本科学生的教材，也可以作为编程工作者，尤其是C语言初学者的参考书。

目　　录

第 1 章　基础知识

1.1　计算机内的数据表示

随着计算机应用领域的日益广泛，计算机除了用于进行复杂的科学计算，还能进行文字处理、图像识别、声音加工等等。数字、汉字、图像和声音的表象千差万别，但对于计算机而言，它们都被称为数据或信息。

在计算机科学中，数据是指所有能被计算机所识别、存储和处理的符号的总称。

计算机的基本功能就是对数据进行处理。

1.1.1　数制及其转换

1. 进位计数制

进位计数制是常用的计数方法。进位计数制是采用有限个数码来表示数据，数据中各个数字所处的位置决定它的权值，每个数字所表示的数值就等于该数字本身乘以它的位置所代表的权值。

各数位只允许选用有限个数码，每一数位所能表示的最大值等于可选用的最大数码乘以其权值，超过这个值就要向高位进位。允许选用的数码的个数就是计数制的基数。

一般，基数为 r 的 r 进制数：$(N)_r = K_n K_{n-1} \ldots K_1 K_0 . K_{-1} K_{-2} \ldots K_{-m}$ 的数值可表示为：$(N)_r = K_n \times r^n + K_{n-1} \times r^{n-1} + \ldots K_1 \times r^1 + K_0 \times r^0 + K_{-1} \times r^{-1} + K_{-2} \times r^{-2} + \ldots K_{-m} \times r^{-m}$

其中，数位 $K_i(-m \leqslant i \leqslant n)$ 的数值可选择 r 个数码中的任意一个，其权值是 r^i。

(1)十进制计数法。

日常生活中用的最多的是十进制计数法，例如：

$(1286.75)_{10} = 1 \times 10^3 + 2 \times 10^2 + 8 \times 10^1 + 6 \times 10^0 + 7 \times 10^{-1} + 5 \times 10^{-2}$

其各位的权值分别为：10^3、10^2、10^1、10^0、10^{-1}、10^{-2}。

十进制计数法是“逢十进一”的，即基数为 10；十进制的数码为：

0、1、2、3、4、5、6、7、8、9

例如：个位数的权值是 1(10^0)，那么个位数所能表示的最大数值为 $9 \times 10^0 = 9$，超过 9 就要向十位进位；十位数的权值是 10(10^1)，那么十位数所能表示的最大数值为 $9 \times 10^1 = 90$，超过 90 就要向百位进位……

(2)二进制计数法。

在计算机内部，数据是用二进制计数法表示的。计算机内用电子器件的两种不同状

态来表示数字信息，如用高电平来表示 1，用低电平表示 0，因此对于计算机的物理实现而言，用二进制表示数据更方便。

例如，二进制数

$(1001101.101)_2 = 1\times2^6 + 0\times2^5 + 0\times2^4 + 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 + 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3}$

显然，其各位的权值分别为：2^6、2^5、2^4、2^3、2^2、2^1、2^0、2^{-1}、2^{-2}、2^{-3}。

二进制计数法是“逢二进一”的，即基数为 2；二进制的数码为：

0、1

也就是说，二进制数的每一位或者为 0，或者为 1；超过 1 就要向高位进位。

(3)八进制计数法与十六进制计数法。

采用二进制计数法表示数据的一个缺点是所需位数太多，容易出错，因此为了便于阅读和书写，常采用八进制数或十六进制数来代替二进制数。

例如，八进制数

$(621)_8 = 6\times8^2 + 2\times8^1 + 6\times8^0$

其各位的权值分别为：8^2、8^1、8^0。

八进制计数法是“逢八进一”的，即基数为 8，八进制计数法的数码为：

0、1、2、3、4、5、6、7

这八个数码相当于十进制数的 0 到 7，接下来下一位数字就向高位进位，即$(10)_8$，相当于十进制数 8，那么$(11)_8$ 相当于十进制数 9，……

又如，十六进制数

$(8A1F)_{16} = 8\times16^3 + 10\times16^2 + 1\times16^1 + 15\times16^0$

其各位的权值分别为：16^3、16^2、16^1、16^0。

十六进制计数法是“逢十六进一”的，即基数为 16。十六进制计数法的数码为：

0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F。

十六进制中有 16 个数码，但从 10 开始到 15 就没有阿拉伯数字可用了，因此规定用字母 A(a)、B(b)、C(c)、D(d)、E(e)、F(f)来表示 10、11、12、13、14、15。

2. 数制转换

(1)任意进制转换为十进制。

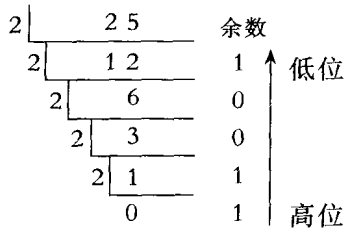
由 r 进制数转换为十进制数可按照如下公式进行多项式展开求和即可：

$$K_nK_{n-1}\cdots K_1K_0.K_{-1}K_{-2}\ldots K_{-m}$$
$$= K_n\times r^n + K_{n-1}\times r^{n-1} + \ldots K_1\times r^1 + K_0\times r^0 + K_{-1}\times r^{-1} + K_{-2}\times r^{-2} + \ldots K_{-m}\times r^{-m}$$

(2)十进制转换为任意进制。

可以采用除基取余法将十进制整数转换为 r 进制整数：将十进制整数除以 r，得到商和余数，余数对应为 r 进制数低位的值；继续让商再除以 r，得到商和余数，……重复此操作，直至商为 0，如此得到的一系列的余数就是所求的 r 进制数的各位数字，先得到的是低位，后得到的是高位。

例如，将$(25)_{10}$转换为二进制整数：



因此，$(25)_{10} = (11001)_2$

可采用乘基取整法将十进制小数转换为 r 进制小数：将十进制小数乘以 r，去掉乘积的整数部分，再将余下的纯小数乘以 r，……重复此操作，直至乘积等于 0 或达到所需的精度为止，如此得到的一系列整数就是 r 进制小数的各位数字，先得到的是高位，后得到的是低位。

例如，将$(0.625)_{10}$转换为二进制小数：

$0.625 \times 2 = 1.25$　　1　高位

$0.25 \times 2 = 0.5$　　0

$0.5 \times 2 = 1$　　1　低位

因此，$(0.625)_{10} = (0.101)_2$

由于整数和小数的转换方法截然不同，将十进制数转换为 r 进制数时，整数部分和小数部分要分开来进行转换。

r 进制小数能精确的转换为十进制小数，但十进制小数往往不能精确的转换为 r 进制小数，如$(0.1)_{10} = (0.0001100110011\cdots\cdots)_2$。

(3)二进制与八进制、十六进制之间的转换。

对于一个二进制数，只要依次(整数部分由低位到高位，小数部分由高位到低位)将其每 3 位或 4 位分成一组(位数不足者，整数部分高位补 0，小数部分低位补 0)，就可以直接转换为八进制数或十六进制数。

例如，二进制数$(1100101111010011.01101)_2$

若按 3 位一组划分，就可以直接写出其对应的八进制数：

001　100　101　111　010　011　.　011　010

1　4　5　7　2　3　.　3　2

即$(1100101111010011.01101)_2 = (145723.32)_8$

若按 4 位一组划分，就可以直接写出其对应的十六进制数：

1100　1011　1101　0011　.　0110　1000

C　B　D　3　.　6　8

即$(1100101111010011.01101)_2 = (CBD3.68)_{16}$

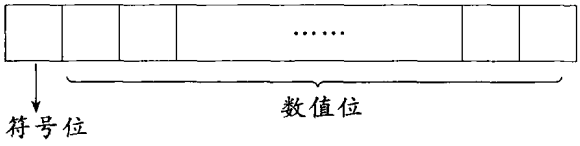
反之，将八进制数中的每一位用 3 位二进制数表示，将十六进制数中的每一位用 4 位二进制数表示，就能转换为对应的二进制数。

1.1.2 原码，反码及补码

计算机中，数据的最小单位是“位”，一般将 8 位二进制位定义为一个字节，计算机中的存储量就是以字节来计算的。

在计算机内，不仅数值是用二进制数表示的，符号也是用二进制数表示的，一般规定：用 0 表示正号“+”，用 1 表示负号“-”；符号位放在数值位之前。一个数连同其符号在机器中的二进制表示形式称为机器数，它所代表的数值称为机器数的真值。

机器数的一般格式为：



为简便起见，下面讨论中用一个字节来表示带符号的数据。

在计算机中，带符号的数的表示方法有三种：原码、反码和补码。

1. 原码

原码表示法是符号位用 0 表示正数，用 1 表示负数，数值位表示数值本身。例如：

$[+27]_{原}=0\ 0\ 0\ 1\ 1\ 0\ 1\ 1$　　$[-27]_{原}=1\ 0\ 0\ 1\ 1\ 0\ 1\ 1$

在原码表示法中，0 的表示方法不惟一：

$[+0]_{原}=0\ 0\ 0\ 0\ 0\ 0\ 0\ 0$　　$[-0]_{原}=1\ 0\ 0\ 0\ 0\ 0\ 0\ 0$。

2. 反码

反码表示法中正数与负数的表示方法不同，正数的反码与原码同形，如：

$[+27]_{反}=0\ 0\ 0\ 1\ 1\ 0\ 1\ 1$

负数的反码为：符号位仍为 1，数值位是对原码取反，如：

$[-27]_{反}=1\ 1\ 1\ 0\ 0\ 1\ 0\ 0$

在反码表示法中，0 的表示也不惟一：

$[+0]_{反}=0\ 0\ 0\ 0\ 0\ 0\ 0\ 0$　　$[-0]_{反}=1\ 1\ 1\ 1\ 1\ 1\ 1\ 1$。

3. 补码

用原码表示数据，简单明了，但用原码进行加减运算却很不方便。

例如，当两个加数的符号相同时，可以使数值位直接相加，结果符号不变。但当两个加数符号不同时，加法运算实际上要转换为减法进行；为了进行减法，应先判断两数的绝对值，让绝对值大的数减去绝对值小的数，运算结果的符号与绝对值大的数的符号相同。计算机实现上述过程是相当繁复的。

因此，在计算机中采用补码表示法来表示数据。采用补码表示数据，能够简化设计与运算，可以将减法运算转化为加法运算。

下面以时钟为例说明补码的原理，假设当前时刻是 3 点，若问两个小时前是几点，那么可以将时针向前拨两小时，即 3 - 2 = 1，得到 1 点；还可以将时针向后拨 10 小时，即 3 + 10 = 13 = 12 + 1，从时钟上看，还是 1 点。从而，3 减去 1，与 3 加上 10 能够达到同样的效果。这是因为时钟的刻度是以 12 为周期的，超过 12 就再从头开始计数。称 2 和 10 是互

补的，即减去 2 就相当于加上 10。这就是减法变加法的原理。

在计算机中，以定长的存储单元来表示数据，因此所能表示数据的范围是有限的，如果超出它的表示范围，高位就会溢出。

正数的补码与原码、反码同形，如：

$[+18]_{原}=[+18]_{反}=[+18]_{补}=00010010$

负数的补码为：符号位为 1，数值位等于原码的各数值位取反，末位再加 1，或者说：

$[x]_{补}=[x]_{反}+1$；

如：

$[-18]_{原}=1\ 0\ 0\ 1\ 0\ 0\ 1\ 0$

$[-18]_{反}=1\ 1\ 1\ 0\ 1\ 1\ 0\ 1$

$[-18]_{补}=1\ 1\ 1\ 0\ 1\ 1\ 1\ 0$

采用补码表示数据，使得正、负的关系转换成了一种纯数值的关系。采用补码形式的数据进行运算时，符号位和数值位一样的参与运算，不必考虑它特别的含义（正、负）。

例如，计算 $-36+58$ 的值。

$[-36]_{补}=1\ 1\ 0\ 1\ 1\ 1\ 0\ 0$　　$[+58]_{补}=0\ 0\ 1\ 1\ 1\ 0\ 1\ 0$

$$\begin{array}{r} 1\ 1\ 0\ 1\ 1\ 1\ 0\ 0 \\ +\ 0\ 0\ 1\ 1\ 1\ 0\ 1\ 0 \\ \hline \text{溢出}\leftarrow\boxed{1}\ 0\ 0\ 0\ 1\ 0\ 1\ 1\ 0 \end{array}$$

显然，$(1\ 1\ 0\ 1\ 1\ 1\ 0\ 0)_2+(0\ 0\ 1\ 1\ 1\ 0\ 1\ 0)_2=(0\ 0\ 0\ 1\ 0\ 1\ 1\ 0)_2=(22)_{10}$，从而验证了 $-36+58=22$。

可见，采用补码表示的数据进行运算的规则是很简单的。而且，在补码表示法中，0 的表示是惟一的，若用一个字节存放数据，规定：

$[0]_{补}=0\ 0\ 0\ 0\ 0\ 0\ 0\ 0$，$[-128]_{补}=1\ 0\ 0\ 0\ 0\ 0\ 0\ 0$。

1.1.3　定点数及浮点数

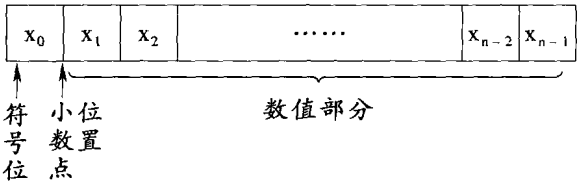
对于一个数据，可能既含有整数部分，又含有小数部分，这就涉及到小数点的表示问题。对小数点的表示方法有两种：定点表示法和浮点表示法。

1. 定点数

若对于所有的数据都限定小数点的位置固定不变，就是定点表示法。

(1)定点小数。

约定小数点的位置在符号位之后，数值部分之前。则定点小数的表示形式如下：



计算机中并不存放小数点，只是隐含约定小数的位置。

假设用一个字节存放数据，则符号位占 1 位，数值位占 7 位，那么：

0 1 0 0 0 1 0 1 表示二进制小数 0.1 0 0 0 1 0 1

1 0 1 1 0 0 0 1 表示二进制小数 -0. 0 1 1 0 0 0 1

若计算机的字长为 n，其中一位是符号位，n－1 位是数值位。那么，

当数值部分的最后一位 x_{n-1} 是 1，其余位是 0 时，数 x 的绝对值最小：

$|x|_{min} = 2^{-(n-1)}$

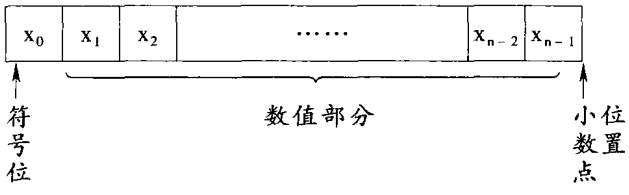
当数值部分全是 1 时，数 x 的绝对值最大：

$|x|_{max} = 1 - 2^{-(n-1)}$。

因此，计算机所能表示的定点小数的绝对值范围是：$2^{-(n-1)} \sim 1 - 2^{-(n-1)}$。

(2)定点整数。

约定小数点的位置在数值部分之后，即数据是纯整数。表示形式为：



字长为 n 的定点整数，考虑其绝对值，最小的数的数值部分全为 0，最大的数的数值部分全为 1，因此其所能表示的数据的绝对值范围是：$0 \sim 2^{n-1} - 1$

在实际处理数据时，一个二进制数 N 可能既有整数部分又有小数部分，那么就需要将其乘上一个倍数以转换成规定的定点整数或定点小数的形式：

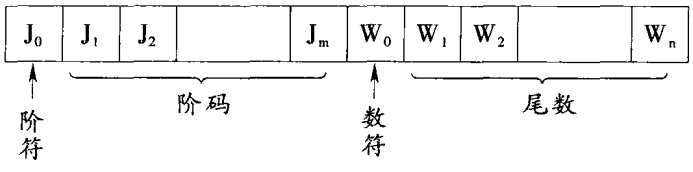
$N = X \times 2^i$

其中 X 为规定的定点整数或定点小数，i 的值一旦选取就固定不变，这是定点数的特点。i 的取值很重要：例如，对于定点小数，i 若选取大了，数据可能溢出；i 若选取小了，可能会有损数据的精度。

2. 浮点数

为了协调数据的表示范围与精度的要求，提出数据的浮点表示方法，即小数点的位置不再固定不变，而是根据需要浮动。

浮点数的格式如下：



浮点数的表示方法类似于数学中的科学计数法，它的真值为：

$$\pm \text{尾数} \times 2^{\pm \text{阶码}}$$

其中尾数的符号由数符表示，阶码的符号由阶符表示，1 表示正，0 表示负。浮点数的符号

由尾数的符号决定，阶码的符号只决定小数点的位置。

浮点数的表示范围主要由阶码决定，有效数字的精度主要由尾数决定。为了充分利用有效数字，将尾数规格化，即限定尾数 W 的绝对值在一定的范围内，一般规定：

$$\frac{1}{2} \leqslant |W| < 1$$

也就是说，尾数部分的最高位是 1。

例如，有一采用浮点表示法的计算机，尾数有 7 位，阶码有 3 位，数符和阶符各一位：

J_0	J_1	J_2	J_3	w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7

那么，它能够表示的数据 X 的绝对值的范围是：

(1)最小值的尾数最小，阶码最大，阶符为负，即

1	1	1	1	0	1	0	0	0	0	0	0

此时，尾数为 2^{-1}，阶码为 $2^3 - 1 = 7$，阶符为负，最小值为 $2^{-1} \times 2^{-7}$。

(2)最大值的尾数最大，阶码最大，阶符为正，即

0	1	1	1	0	1	1	1	1	1	1	1

此时，尾数为 $1 - 2^{-7}$，阶码为 $2^3 - 1 = 7$，阶符为正，最大值为 $(1 - 2^{-7}) \times 2^7$

因此，$2^{-1} \times 2^{-7} \leqslant |X| \leqslant (1 - 2^{-7}) \times 2^7$

那么，尾数有 n 位，阶码有 m 位的浮点数的绝对值的表示范围为：

$2^{-1} \times 2^{-(2^m - 1)} \sim (1 - 2^{-n}) \times 2^{(2^m - 1)}$

1.2 程序与算法

1.2.1 程序及算法的概念

现实生活中，做任何事情都需要经过一定的步骤才能完成。例如，乐队演奏乐曲、教师按教学计划授课、工程师设计施工方案等等都必须按照一定的步骤进行。

为解决一个问题而采取得方法和步骤，称为算法。算法就是被精确定义的一组规则，规定先做什么，再做什么，以及判断某种情况下做哪种操作；或者说算法是步进式的完成所需任务的过程。

为了让计算机来解决问题，除了需要明确解决问题的算法，还必须用特定的计算机指令来控制计算机按照我们的意图有步骤的工作，最终完成特定的任务。

计算机程序是指为让计算机完成特定的任务而设计的指令序列。程序设计是用来沟通算法与计算机的桥梁；程序是编程者写的、计算机能够理解并执行的一些命令的集合，

是解决问题的具体步骤在计算机中的实现。

1.2.2 算法的特点及评价标准

算法反映解决问题的步骤，不同的问题需要用不同的算法来解决，同一问题也可能有不同的解决方法，但是一个算法必须具有以下特性：

(1)有穷性。

一个算法必须总是在执行有限个操作步骤和可以接受的时间内完成其执行过程。也就是说，对于一个算法，要求其在时间和空间上均是有穷的。例如：一个采集气象数据并加以计算进行天气预报的应用程序，如果不能及时得到结果，起不到天气预报的作用，显然就超出了可以接受的时间。

(2)确定性。

算法中的每一步都必须有明确的含义，不允许存在二义性。例如："将成绩优秀的同学名单打印输出"，在这一描述中"成绩优秀"就很不明确，是每门功课均为 95 分以上？还是指总成绩在多少分以上？

(3)有效性。

算法中描述的每一步操作都应该能有效地执行，并最终得到确定的结果。例如：当 Y = 0 时，X/Y 是不能有效执行的。

(4)输入。

一个算法应该有零个或多个输入数据。例如：计算从 1 到 10 的累计和的算法，是无须输入数据。而对任意 10 个整数进行排序的算法，却需要从键盘上输入这 10 个整数。

(5)输出。

一个算法应该有 1 个或多个输出数据。执行算法的目的是为了求解，而"解"就是输出，因此没有输出的算法是毫无意义的。

设计一个好的算法对于正确的、高效的解决问题有着重要的意义。一个好的算法应达到以下目标：

① 正确性：算法应该能够满足问题的要求。

② 可读性：算法不仅仅是让计算机来执行的，更重要的是让人来阅读，可读性好的算法有助于调试程序、发现和修改错误，使得日后对软件功能的扩展、维护易于实现。

③ 健壮性：指算法能够对非法的输入做出合理的处理，而不是产生莫名其妙的结果。

④ 高效率与低存储空间需求：指解决特定问题的算法的执行时间应尽量的短，算法执行过程中需要的存储空间应尽量的小。

1.2.3 算法的表示

算法的表示方法很多，常见的有：自然语言、伪码、传统流程图、N-S 结构图等。

1. 用自然语言表示

自然语言就是人们日常使用的语言，可以是中文、英文等。

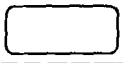
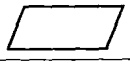
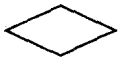
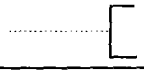
例如，求三个数的最大值的问题，可以描述为：先比较前两个数，找到大的那个数，再让其与第三个数进行比较，找到二者中大的数即为所求。

显然，用自然语言表示的算法通俗易懂，但文字冗长，表达上不易准确，容易出现“歧义性”。所以，一般不用自然语言描述算法。

2. 用传统流程图表示

传统流程图是用规定的一组图形符号、流程线和文字说明来表示各种操作的算法表示方法。传统流程图常用的符号如表 1.1 所示。

表 1.1　传统流程图常用符号

符　号	符号名称	含　　义
	起止框	表示算法的开始和结束
	输入输出框	表示输入输出操作
	处理框	表示对框内的内容进行处理
	判断框	表示对框内的条件进行判断
	流程线	表示流程的方向
	连接点	表示两个具有同一标记的“连结点”应连接成一个点
	注释框	表示对流程图中某些框的操作做必要的补充说明

用传统流程图描述求三个数中最大值的算法如图 1.1 所示。

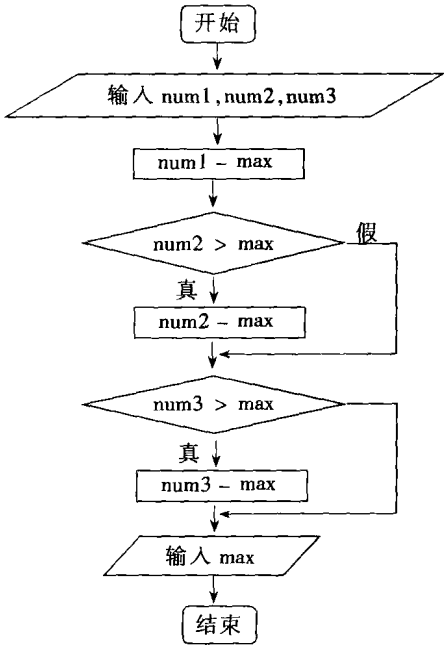


图 1.1　传统流程图

用传统流程图表示算法直观形象，算法的逻辑流程一目了然，便于理解。但占用篇幅较大，画起来比较麻烦，而且由于允许使用流程线，使用者可以随心所欲，使流程可以任意转移，从而造成阅读和修改上的困难。

3. 用 N-S 结构图表示

针对传统流程图存在的问题，美国学者 I. Nassi 和 B. Shneiderman 于 1973 年提出一种新的结构化流程图形式，即简称为 N-S 结构图。Chapin 在 1974 年对其进行了进一步的扩展，因此，N-S 结构图又称为 Chapin 图或盒图。

N-S 结构图的目标是开发一种不破坏结构化基本构成元素的过程设计表示。其主要特点是完全取消了流程线，不允许有随意的控制流，全部算法写在一个矩形框内，该矩形框以三种基

本结构(顺序、选择、循环)描述符号为基础复合而成。

N-S 结构图表示的三种基本结构如下:

(1)顺序结构。

顺序结构是最简单的基本结构。在顺序结构中,要求顺序地执行且必须执行由先后顺序排列的每一个最基本的处理单位。如图 1.2 所示(a)图是用传统流程图表示的顺序结构,(b)图是用 N-S 结构图表示的顺序结构,先执行处理 A,然后再顺序执行处理 B。

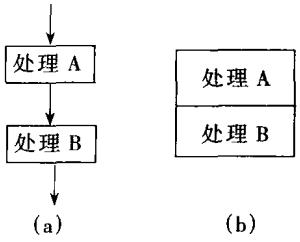


图 1.2 顺序结构

(2)选择结构。

选择结构又称作分支结构。在选择结构中,要根据逻辑条件的成立与否,分别选择执行不同的处理。如图 1.3 所示,当逻辑条件成立是,执行处理 A,否则执行处理 B。

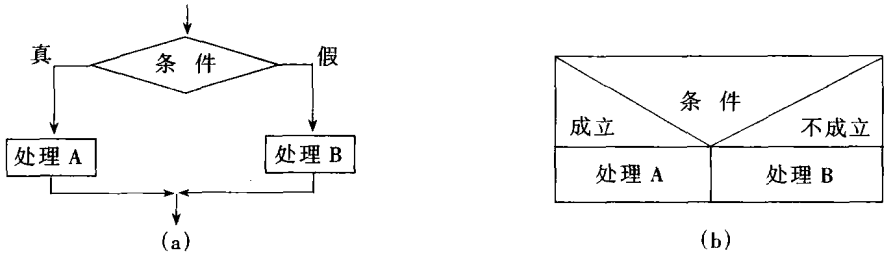


图 1.3 选择结构

(3)循环结构。

循环结构一般分为当型循环和直到型循环。

① 当型循环。

在当型循环结构中,当逻辑条件成立时,就反复执行处理 A(称为循环体),直到逻辑条件不成立时结束。如图 1.4 所示。

图 1.4 当型循环结构

② 直到型循环。

在直到型循环结构中,反复执行处理 A,直到逻辑条件成立时结束(即逻辑条件不成立时继续执行)。如图 1.5 所示。