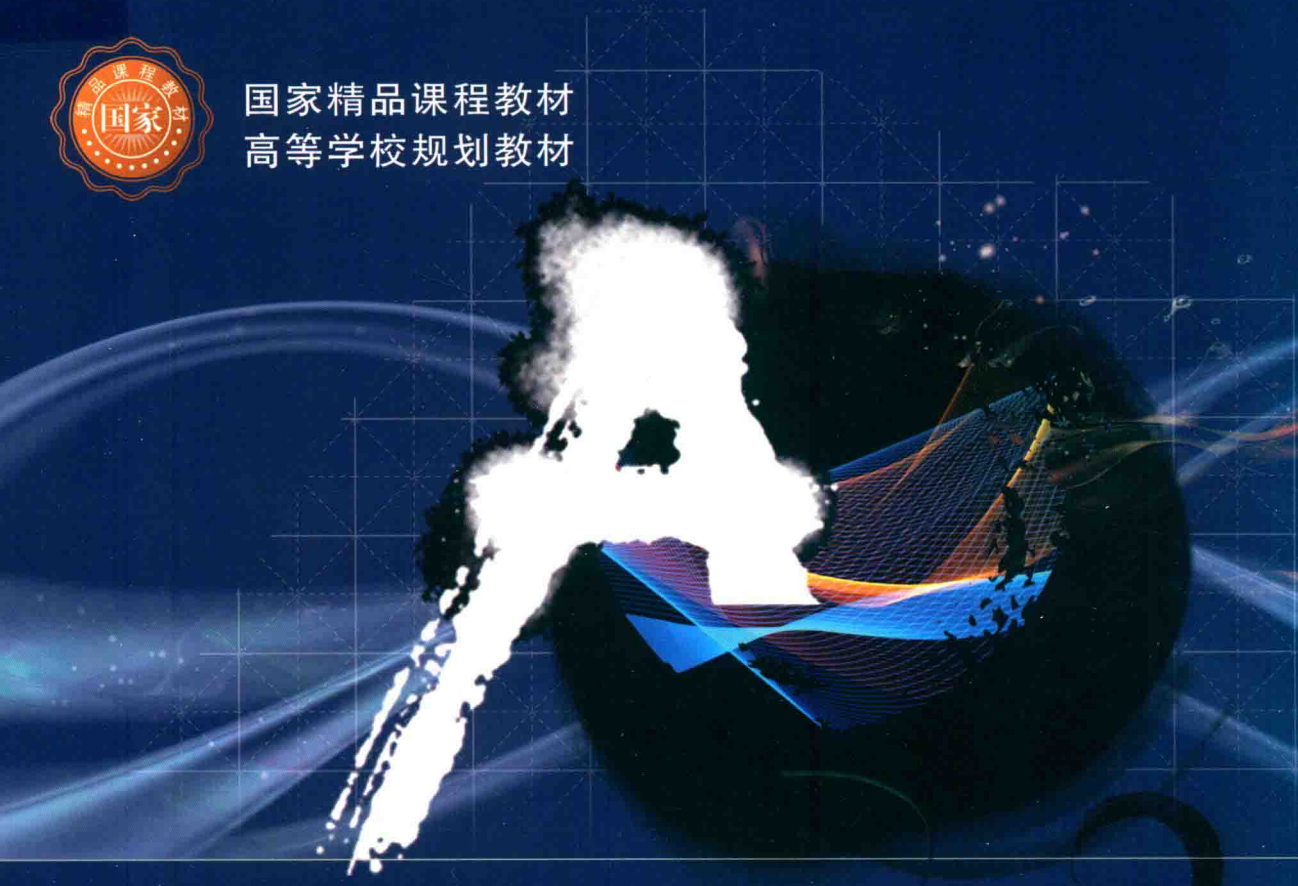
国家精品课程教材
高等学校规划教材



数据结构

（C语言描述）（第3版）

◎ 王晓东　编著

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

高等学校规划教材

数 据 结 构

（C 语言描述）

（第3版）

王晓东　编著

電子工業出版社
Publishing House of Electronics Industry
北京·BEIJING

内 容 简 介

本书是国家精品课程教材，以 ACM 和 IEEE/CS Computing Curricula 课程体系及教育部计算机科学与技术教学指导委员会发布的“高等学校计算机科学与技术本科专业规范”为依据，以基本数据结构为知识单元编写而成。全书分为 10 章，内容包括引论、表、栈、队列、排序与选择算法、树、散列表、优先队列、并查集、图等。全书采用 C 语言作为描述语言，内容丰富，叙述简明，理论与实践并重。每章设有应用举例和算法实验题，并为任课教师免费提供电子课件和课程实验用数据。

本书可作为高等学校计算机、电子信息、信息与计算科学、信息管理与信息系统等专业的数据结构课程教材，也适合工程技术人员和自学者学习参考。

图书在版编目（CIP）数据

数据结构：C 语言描述 / 王晓东编著. —3 版. —北京：电子工业出版社，2019.8

ISBN 978-7-121-34442-8

Ⅰ. ①数… Ⅱ. ①王… Ⅲ. ①数据结构－高等学校－教材②C 语言－程序设计－高等学校－教材

Ⅳ. ①TP311.12②TP312.8

中国版本图书馆 CIP 数据核字（2018）第 119824 号

责任编辑：章海涛
文字编辑：张 鑫
印 刷：三河市鑫金马印装有限公司
装 订：三河市鑫金马印装有限公司
出版发行：电子工业出版社
北京市海淀区万寿路 173 信箱 邮编：100036
开 本：787×1092 1/16 印张：16.25 字数：427 千字
版 次：2007 年 7 月第 1 版
2019 年 8 月第 3 版
印 次：2019 年 8 月第 1 次印刷
定 价：49.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：192910558（QQ 群）。

前　言

要想以最低的成本、最快的速度、最好的质量开发出适合各种应用需求的软件，必须遵循软件工程的原则，设计出高效的程序。一个高效的程序不仅需要编程技巧，而且需要合理的数据组织和清晰高效的算法。这正是计算机科学领域里数据结构与算法设计所研究的主要内容。

计算机科学是一种创造性思维活动，其教育必须面向设计。数据结构正是一门面向设计，且处于计算机学科核心地位的教育课程。通过对数据结构知识的系统学习与研究，理解和掌握数据结构与算法设计的主要方法，为独立完成软件设计和分析奠定坚实的理论基础，对从事计算机系统结构、系统软件和应用软件研究与开发的科技工作者来说是必不可少的。为了适应 21 世纪我国培养各类计算机人才的需要，本课程结合我国高等学校教育工作的现状，追踪国际计算机科学技术的发展水平，更新了教学内容和教学方法，以基本数据结构为知识单元，系统介绍数据结构知识与应用，以期为计算机相关专业的学生提供一个扎实的数据结构设计的知识基础。本课程的教学改革实践取得了丰硕的成果，课程已**被评为国家精品课程**。

本书以 ACM 和 IEEE/CS Computing Curricula 课程体系及教育部计算机科学与技术教学指导委员会发布的**“高等学校计算机科学与技术本科专业规范”**中关于算法与数据结构的知识结构和体系为依据编写，全书分为 10 章。

第 1 章为**引论**，介绍数据结构、抽象数据类型和算法等基本概念，并简要阐述了算法的计算复杂性和对算法的描述。

第 2～4 章依次介绍基于序列的抽象数据类型**表、栈和队列**。

第 5 章介绍在实际应用中常用的**排序与选择**算法。

第 6 章讨论反映层次关系的抽象数据类型**树**。

第 7 章讨论**散列表**等实践中常用的实现集合和符号表的方法。

第 8 章讨论以有序集为基础的抽象数据类型**优先队列**及其实现方法。

第 9 章讨论以不相交集合为基础的抽象数据类型**并查集**及其实现方法。

第 10 章介绍非线性结构**图**及其算法。

考虑到学生的知识基础和课程体系的需要，本书用 **C 语言作为描述语言**，并尽量使数据结构和算法的描述简明、清晰。**参考学时数为 54～68**。

数据结构是一门理论性强、实践难度较大的专业基础课程。为了使学生在深刻理解课程内容的基础上，灵活运用所学的知识解决实际问题，我们在章首增加了**学习要点提示**，章末有**本章小结**和难易适当的**习题**，并特别设计了**算法实验题**，以强化实践环节，要求学生课后通过上机实验来完成。作者的教学实践表明，这类算法实验题对学生掌握课堂教学内容有很大帮助，效果非常好。

国家精品课程资源共享课地址：http://www.icourses.cn/sCourse/course_2535.html。欢迎广大读者访问教学网站，并提出宝贵意见，作者 E-mail：wangxd@fzu.edu.cn。

在本书编写过程中，得到了全国高等学校计算机专业教学指导委员会的关心和支持。福州大学“211 工程”计算机与信息工程重点学科实验室和福建工程学院为本书编写提供了优良的设备和工作环境。傅清祥教授、吴英杰教授、傅仰耿博士和朱达欣教授参加了本书有关章节的讨论，对本书第 3 版的内容及各章节的编排提出了许多建设性意见。田俊教授认真审阅了全书。在此，谨向每一位曾经关心和支持本书编写工作的人士表示衷心的感谢！

由于作者的知识和写作水平有限，书稿虽几经修改，仍难免有缺点和错误。热忱欢迎同行专家和读者批评指正，以使本书不断改进，日臻完善。

作者

目　录

第1章 引 论

学习要点

- 理解算法的概念
- 理解什么是程序，程序与算法的区别和内在联系
- 能够列举求解问题的基本步骤
- 掌握算法在最坏情况、最好情况和平均情况下的时间复杂性
- 掌握算法复杂性的渐近性态的数学表述
- 了解表达算法的抽象机制
- 熟悉数据类型和数据结构的概念
- 熟悉抽象数据类型的基本概念
- 理解数据结构、数据类型和抽象数据类型三者的区别和联系
- 掌握用 C 语言描述数据结构与算法的方法
- 理解递归的概念

1.1 算法及其复杂性的概念

1.1.1 算法与程序

对于计算机科学来说，算法（Algorithm）的概念是至关重要的。例如，在一个大型软件系统的开发中，设计出有效的算法起决定性作用。通俗地讲，算法是指解决问题的一种方法或一个过程。严格地讲，算法是由若干条指令组成的有穷序列，且具有下述 4 条性质。

① 输入：有零个或多个由外部提供的量作为算法的输入。

② 输出：算法产生至少一个量作为输出。

③ 确定性：组成算法的每条指令是清晰的，无歧义的。

④ 有限性：算法中每条指令的执行次数是有限的，执行每条指令的时间也是有限的。

程序（Program）与算法不同。程序是算法用某种程序设计语言的具体实现。程序可以不满足算法的性质④。例如，操作系统是一个在无限循环中执行的程序，因而不是一个算法。然而可把操作系统的各种任务看成一些单独的问题，每一个问题由操作系统中的一个子程序通过特定的算法来实现，该子程序得到输出结果后便终止。

1.1.2 算法复杂性的概念

一个算法的复杂性的高低体现在运行该算法所需要的计算机资源的多少上，所需要的资源越多，该算法的复杂性越高；反之，所需要的资源越少，该算法的复杂性越低。计算机资源中最重要的是时间和空间（即存储器）资源。因此，算法的复杂性有时间复杂性和空间复

杂性之分。

不言而喻，对于任意给定的问题，设计出复杂性尽可能低的算法是设计算法时追求的一个重要目标。另一方面，当给定的问题已有多种算法时，选择其中复杂性最低者，是选用算法应遵循的一个重要准则。因此，算法复杂性分析对算法的设计或选用有重要的指导意义和实用价值。

确切地说，算法的复杂性是指运行算法所需要的计算机资源的量。需要的时间资源的量称为时间复杂性，需要的空间资源的量称为空间复杂性。这个量应该集中反映算法的效率，而从运行该算法的实际计算机中抽象出来。换句话说，这个量应该是只依赖于算法要解的问题的规模和算法的输入的函数。

如果分别用 n 和 I 表示算法要解的问题的规模和算法的输入，用 C 表示复杂性，那么算法复杂性可表示为 $C(n,I)$。如果把时间复杂性和空间复杂性分开，并分别用 T 和 S 来表示，那么应该有 $T=T(n,I)$ 和 $S=S(n,I)$。由于时间复杂性与空间复杂性概念类同，计量方法相似，且空间复杂性分析相对简单些，所以本书将主要讨论时间复杂性。现在的问题是如何将复杂性函数具体化，即对于给定的 n 和 I，如何导出 $T(n,I)$ 和 $S(n,I)$ 的数学表达式，给出计算 $T(n,I)$ 和 $S(n,I)$的法则。下面以 $T(n,I)$ 为例，将复杂性函数具体化。

根据 $T(n,I)$ 的概念，它应该是算法在一台抽象的计算机上运行所需要的时间。设此抽象的计算机所提供的元运算有 k 种，分别记为 $O_1,O_2,\cdots,O_k$。又设每执行一次这些元运算所需要的时间分别为 $t_1,t_2,\cdots,t_k$。对于给定的算法 A，设经统计，用到元运算 O_i 的次数为 e_i，$i=1,2,\cdots,k$。显然，对于每一个 i，$1\leqslant i\leqslant k$，e_i 是 n 和 I 的函数，即 $e_i=e_i(n,I)$。那么有

$$T(n,I)=\sum_{i=1}^{k}t_ie_i(n,I)$$

式中，t_i ($i=1,2,\cdots,k$)是与 n 和 I 无关的常数。

显然，不可能对规模为 n 的每一种合法的输入 I 都统计 $e_i(n,I)$，$i=1,2,\cdots,k$。因此 $T(n,I)$ 的表达式还要进一步简化。或者说，只能在规模为 n 的某些或某类有代表性的合法输入中统计相应的 e_i，$i=1,2,\cdots,k$，评价时间复杂性。

通常考虑最坏、最好和平均三种情况下的时间复杂性，并分别记为 $T_{\max}(n)$，$T_{\min}(n)$ 和 $T_{\text{avg}}(n)$。在数学上有

$$T_{\max}(n)=\max_{I\in D_n}T(n,I)=\max_{I\in D_n}\sum_{i=1}^{k}t_ie_i(n,I)=\sum_{i=1}^{k}t_ie_i(n,I^*)=T(n,I^*)$$

$$T_{\min}(n)=\min_{I\in D_n}T(n,I)=\min_{I\in D_n}\sum_{i=1}^{k}t_ie_i(n,I)=\sum_{i=1}^{k}t_ie_i(n,\tilde{I})=T(n,\tilde{I})$$

$$T_{\text{avg}}(n)=\sum_{I\in D_n}P(I)T(n,I)=\sum_{I\in D_n}P(I)\sum_{i=1}^{k}t_ie_i(n,I)$$

式中，D_n 是规模为 n 的合法输入的集合；I^* 是 D_n 中一个使 $T(n,I^*)$ 达到 $T_{\max}(n)$ 的合法输入；$\tilde{I}$ 是 D_n 中一个使 $T(n,\tilde{I})$ 达到 $T_{\min}(n)$ 的合法输入；而 $P(I)$ 是在算法的应用中出现输入 I 的概率。

以上三种情况下的时间复杂性从不同角度反映算法的效率，各有各的局限性，也各有各的用处。实践表明，可操作性最好且最有实际价值的是最坏情况下的时间复杂性。本书对算法时

间复杂性分析的重点将放在这种情况下。

1.1.3 算法复杂性的渐近性态

随着经济的发展、社会的进步和科学研究的深入，要求用计算机解决的问题越来越复杂，规模越来越大。对求解这类问题的算法进行复杂性分析具有特别重要的意义，因而要特别关注。在此引入复杂性渐近性态的概念。

设$T(n)$是前面所定义的关于算法A的复杂性函数。一般来说，当n单调增加且趋于∞时，$T(n)$也将单调增加趋于∞。对于$T(n)$，若存在$\tilde{T}(n)$，使得当$n\to\infty$时，有$\dfrac{T(n)-\tilde{T}(n)}{T(n)}\to 0$，则$\tilde{T}(n)$是$T(n)$当$n\to\infty$时的渐近性态，或称$\tilde{T}(n)$为算法$A$当$n\to\infty$时的渐近复杂性，以示与$T(n)$区别。因为在数学上，$\tilde{T}(n)$是$T(n)$当$n\to\infty$时的渐近表达式；直观上，$\tilde{T}(n)$是$T(n)$中略去低阶项所留下的主项，所以它比$T(n)$更简单。

例如，当$T(n)=3n^2+4n\log n$[注]$+7$时，$\tilde{T}(n)$的一个答案是$3n^2$，因为这时有

$$\lim_{n\to\infty}\frac{T(n)-\tilde{T}(n)}{T(n)}=\lim_{n\to\infty}\frac{4n\log n+7}{3n^2+4n\log n+7}=0$$

显然，$3n^2$比$3n^2+4n\log n+7$简单得多。

由于当$n\to\infty$时$T(n)$渐近于$\tilde{T}(n)$，所以可以用$\tilde{T}(n)$来替代$T(n)$，作为算法A在$n\to\infty$时的复杂性的度量。而且$\tilde{T}(n)$明显地比$T(n)$简单，这种替代是对复杂性分析的一种简化。还要进一步考虑分析算法的复杂性的目的在于比较求解同一问题的两个不同算法的效率。而当要比较的两个算法的渐近复杂性的阶不相同时，只要能确定出各自的阶，就可以判定哪一个算法的效率高。换句话说，这时的渐近复杂性分析只要关心$\tilde{T}(n)$的阶就够了，不必关心包含在$\tilde{T}(n)$中的常数因子。因此又可对$\tilde{T}(n)$的分析进一步简化，即假设算法中用到的所有不同的元运算各执行一次所需要的时间都是一个单位时间。

上面已经给出了简化算法复杂性分析的方法，即只需要考查当问题的规模充分大时，算法复杂性在渐近意义下的阶。本书的算法分析都将这么进行。为此引入渐近意义下的记号O，Ω，θ和o。

以下设$f(n)$和$g(n)$是定义在正数集上的正函数。

若存在正的常数C和自然数n_0，使得当$n\geqslant n_0$时有$f(n)\leqslant Cg(n)$，则称函数$f(n)$当n充分大时上有界，且$g(n)$是它的一个上界，记为$f(n)=O(g(n))$。这时还称$f(n)$的阶不高于$g(n)$的阶。

举几个例子如下。

（1）因为对所有的$n\geqslant 1$有$3n\leqslant 4n$，所以$3n=O(n)$。

（2）因为当$n\geqslant 1$时有$n+1024\leqslant 1025n$，所以$n+1024=O(n)$。

（3）因为当$n\geqslant 10$时有$2n^2+11n-10\leqslant 3n^2$，所以$2n^2+11n-10=O(n^2)$。

注：没有标注底的 log 可默认为以 2 为底。在算法分析中，可不关心对数的底，这是因为不同的底只相差一个常数，对算法复杂性没有任何影响。

（4）因为对所有 $n \geqslant 1$ 有 $n^2 \leqslant n^3$，所以 $n^2 = O(n^3)$。

（5）一个反例 $n^3 \neq O(n^2)$。因为若不然，则存在正的常数 C 和自然数 n_0，使得当 $n \geqslant n_0$ 有 $n^3 \leqslant Cn^2$，即 $n \leqslant C$。显然，当取 $n = \max\{n_0, \lfloor C \rfloor + 1\}$ 时这个不等式不成立，所以 $n^3 \neq O(n^2)$。

按照符号 O 的定义，容易证明它有如下运算规则：

（1）$O(f) + O(g) = O(\max(f, g))$；

（2）$O(f) + O(g) = O(f + g)$；

（3）$O(f)O(g) = O(fg)$；

（4）若 $g(n) = O(f(n))$，则 $O(f) + O(g) = O(f)$；

（5）$O(Cf(n)) = O(f(n))$，其中 C 是一个正的常数；

（6）$f = O(f)$。

规则（1）的**证明**：设 $F(n) = O(f)$。根据符号 O 的定义，存在正常数 C_1 和自然数 n_1，使得对所有的 $n \geqslant n_1$，有 $F(n) \leqslant C_1 f(n)$。类似地，设 $G(n) = O(g)$，则存在正常数 C_2 和自然数 n_2，使得对所有的 $n \geqslant n_2$ 有 $G(n) \leqslant C_2 g(n)$。

令 $C_3 = \max\{C_1, C_2\}$，$n_3 = \max\{n_1, n_2\}$，$h(n) = \max\{f, g\}$，则对所有的 $n \geqslant n_3$，有

$$F(n) \leqslant C_1 f(n) \leqslant C_1 h(n) \leqslant C_3 h(n)$$

类似地有

$$G(n) \leqslant C_2 f(n) \leqslant C_2 h(n) \leqslant C_3 h(n)$$

因而

$$\begin{aligned} O(f) + O(g) &= F(n) + G(n) \\ &\leqslant C_3 h(n) + C_3 h(n) \\ &= 2C_3 h(n) \\ &= O(h) \\ &= O(\max(f, g)) \end{aligned}$$

其余规则的证明类似，留给读者作为练习。

根据符号 O 的定义，用它评估算法的复杂性，得到的只是当规模充分大时的一个上界。这个上界的阶越低，评估就越精确，结果就越有价值。

与渐近复杂性有关的另一记号是 Ω，其定义如下：若存在正常数 C 和自然数 n_0，使得当 $n \geqslant n_0$ 时有 $f(n) \geqslant Cg(n)$，则称函数 $f(n)$ 当 n 充分大时下有界，且 $g(n)$ 是它的一个下界，记为 $f(n) = \Omega(g(n))$。这时我们还说 $f(n)$ 的阶不低于 $g(n)$ 的阶。

用 Ω 评估算法的复杂性，得到的只是该复杂性的一个下界。这个下界的阶越高，评估就越精确，结果就越有价值。这里的 Ω 只是对问题的一个算法而言的。如果它是对一个问题的所有算法或某类算法而言的，即对于一个问题和任意给定的充分大的规模 n，下界在该问题的所有算法或某类算法的复杂性中取，那么它将更有意义。这时得到的相应下界，称为问题的下界或某类算法的下界。它常常与符号 O 配合，以证明某问题的一个特定算法是该问题的最优算法，或该问题的某算法类中的最优算法。

现在来看符号 θ 的定义。定义 $f(n) = \theta(g(n))$ 当且仅当 $f(n) = O(g(n))$ 且 $f(n) = \Omega(g(n))$。这时认为 $f(n)$ 与 $g(n)$ 同阶。

最后，若对于任意给定的$\varepsilon>0$，都存在正整数n_0，使得当$n\geqslant n_0$时有$f(n)/g(n)<\varepsilon$，则称函数$f(n)$当n充分大时的阶比$g(n)$的低，记为$f(n)=O(g(n))$。

例如，$4n\log n+7=O(3n^2+4n\log n+7)$。

1.2 算法的表达与数据表示

1.2.1 问题求解

用计算机解决一个稍复杂的实际问题，一般都要进行如下步骤。

（1）将实际问题数学化，即把实际问题抽象为一个带有一般性的数学问题。这一步要引入一些数学概念，精确地阐述数学问题，弄清问题的已知条件和所要求的结果，以及在已知条件和所要求的结果之间存在着的隐式或显式的联系。

（2）对于确定的数学问题，设计求解的方法，即算法设计。这一步要建立问题的求解模型，即确定问题的数据模型并在此模型上定义一组运算，然后借助于对这组运算的执行和控制，从已知数据出发导出所要求的结果，形成算法并用自然语言来表述。这种语言不是程序设计语言，不能被计算机接受。

（3）用计算机上的一种程序设计语言来表达已设计好的算法。即将非形式化的自然语言表达的算法转变为用一种程序设计语言表达的算法。这一步称为程序设计或程序编制。

（4）在计算机上编辑、调试和测试编制好的程序，直到输出所要求的结果。

在上述问题求解的过程中，求解问题的算法及其实现是核心内容。本章着重考虑第（2）步，而且把注意力集中在算法表达的抽象机制上，目的是引入抽象数据类型的重要概念，同时为大型程序设计提供一种自顶向下逐步求精的模块化方法，即运用抽象数据类型来描述程序的方法。

1.2.2 表达算法的抽象机制

算法是一个运算序列。它的所有运算定义在一类特定的数据模型上，并以解决一类特定问题为目标。算法的程序表达归根结底是算法要素的程序表达，因为一旦算法的每一项要素都已经用程序清楚地表达，整个算法的程序表达也就不成问题了。

算法实现有如下三要素：

（1）作为运算序列中各种运算的对象和结果的数据；

（2）运算序列中的各种运算；

（3）运算序列中的控制转移。

这三要素依序分别简称为数据、运算和控制。

由于算法层出不穷，千变万化，其运算的对象数据和得到的结果数据名目繁多。最简单最基本的有布尔值数据、字符数据、整数和实数数据等；稍复杂的有向量、矩阵、记录等数据；更复杂的有集合、树和图，还有声音、图形、图像等数据。

同样，运算种类也五花八门。最基本最初等的有赋值运算、算术运算、逻辑运算和关系运算等；稍复杂的有算术表达式、逻辑表达式等；更复杂的有函数值计算、向量运算、矩阵运算、集合运算，以及表、栈、队列、树和图上的运算等；此外，还有以上列举的运算的复合和嵌套。

控制转移相对单纯。在串行计算中，它只有顺序、分支、循环、递归和无条件转移等几种。

最早的程序设计语言是机器语言，即具体的计算机上的一个指令集。当时，在计算机上运行的所有算法都必须直接用机器语言来表达，计算机才能接受。算法的运算序列（包括运算对象和运算结果）都必须转换为指令序列，其中的每一条指令都以编码（指令码和地址码）的形式出现。这与用高级程序设计语言表达的算法相差甚远。对于没受过程序设计专门训练的人来说，程序可读性极差。

用机器语言表达算法的数据、运算和控制十分繁杂，因为机器语言所提供的指令太初等、太原始。机器语言只能表达算术运算、按位逻辑运算和数的大小比较运算等。稍复杂的运算都必须分解为最初等的运算，才能用相应的指令替代它。机器语言能直接表达的数据只有最原始的位、字节和字三种。算法中即使是最简单的数据，如布尔值、字符、整数、实数，也必须映射到位、字节和字中，还要给它们分配存储单元。对于算法中有结构的数据的表达，则要麻烦得多。机器语言所提供的控制转移指令也只有无条件转移、条件转移、进入子程序和从子程序返回等最基本的几种。用它们构造循环、形成分支、调用函数都要事先做许多准备，还需要许多经验和技巧。

直接用机器语言表达算法有许多缺点，如下所述。

（1）大量繁杂的细节牵制着程序员，使他们不可能有更多的时间和精力去从事创造性的劳动，执行对他们来说更为重要的任务，如确保程序的正确性、高效性。

（2）程序员既要把握程序设计的全局又要深入每一个局部直到实现的细节，即使智力超群的程序员也常常会顾此失彼，屡出差错，因而所编制的程序可靠性差，且开发周期长。

（3）由于用机器语言进行程序设计的思维和表达方式与人们的习惯大相径庭，只有经过较长时间职业训练的程序员才能胜任，使得程序设计曲高和寡。

（4）机器语言的书面形式全是“密码”，可读性差，不便于交流与合作。

（5）机器语言高度依赖于具体的计算机，可移植性和可重用性差。

克服上述缺点的办法是对程序设计语言进行抽象，让它尽可能接近算法语言。为此，人们首先注意到的是可读性和可移植性，因为它们较容易通过抽象得到改善。

汇编语言实现了对机器语言的抽象，它将机器语言的每一条指令符号化：以记忆符号替代指令码，以符号地址替代地址码，使含义显现在符号上而不再隐藏在编码中。另一方面，汇编语言摆脱了具体计算机的限制，可在具有不同指令集的计算机上运行，只要该计算机配备了汇编语言的一个汇编程序。这无疑是机器语言朝算法语言靠拢迈出的重要一步。但它离算法语言还太远，程序员还不能从分解算法的数据、运算和控制，直至细化到汇编可直接表达的指令等繁杂的事务中解脱出来。

高级程序设计语言的出现使算法的程序表达产生了一次飞跃。算法最终要表达为具体计算机上的机器语言才能在该计算机上运行，得到所需要的结果。但汇编语言的实践启发人们，表达成机器语言不必一步到位，可以分两步走，即先表达成一种中间语言，然后转换成机器语言。汇编语言作为一种中间语言，并没有获得很大成功，原因是它离算法语言还太远。这便促使人们去设计一种尽量接受算法语言的规范语言，即高级程序设计语言，让程序员可以方便地表达算法，然后借助于规范的高级语言到规范的机器语言的“翻译”，最终将算法表达为机器语言。而且，由于高级语言和机器语言都具有规范性，这里的“翻译”完全可以机械化地由计算

机来完成，就像汇编语言翻译成机器语言一样，只要计算机配备一个编译程序即可。上述两步，前一步由程序员去完成，后一步由编译程序去完成。在规定好它们各自该做什么之后，这两步是完全独立的。前一步要做的只是用高级语言正确地表达给定的算法，产生一个高级语言程序；后一步要做的只是将第一步得到的高级语言程序翻译成机器语言程序。至于程序员如何用高级语言表达算法，编译程序如何将高级语言表达的算法翻译成机器语言表达的算法，二者毫不相干。

处理从算法语言最终表达成机器语言这一复杂过程的上述思想方法就是一种抽象。汇编语言和高级语言的出现都是这种抽象的范例。与汇编语言相比，高级语言的巨大成功在于它在数据、运算和控制三方面的表达中引入了许多使之十分接近算法语言的概念和工具，大大提高了抽象表达算法的能力。

在运算方面，高级语言除允许原封不动地运用算法语言的算术运算、逻辑运算、关系运算、算术表达式和逻辑表达式外，还引入了强有力的函数等工具，并让用户自定义。这一工具的重要性不仅在于它精简了重复的程序文本段，还在于它反映出程序的二级抽象。在函数调用级，人们只关心它能做什么，不必关心它如何做。只是到定义函数时，人们才给出如何做的细节。用过高级语言的读者都知道，一旦函数的名称、参数和功能被规定清楚，在程序中调用它们便与在程序的头部说明它们完全分开。可以修改一个函数甚至更换函数体而不影响调用该函数。如果把函数名看成运算名，把参数看成运算的对象或运算的结果，那么，函数调用和初等运算的引用就完全一样。利用函数及函数的复合或嵌套可以很自然地表达算法语言中任何复杂的运算。

在数据表示方面，高级语言引入了数据类型的概念，即把所有的数据加以分类。每一个数据（包括表达式）或每一个数据变量都属于其中确定的一类。这一类数据称为一个数据类型。数据类型是数据或数据变量类属的说明，它指示该数据或数据变量可能取的值的全体。对于无结构的数据，高级语言除提供标准的基本数据类型外，还提供用户可自定义的枚举类型、子界类型和指针类型等。这些类型的使用方式都符合人们在算法语言中的使用习惯。对于有结构的数据，高级语言提供了数组、记录、集合和文件等标准的结构数据类型。其中，数组是科学计算中的向量、矩阵的抽象；记录是商业和管理中的记录的抽象；集合是数学中小集合的势集的抽象；文件是外存储数据（如磁盘中的数据等）的抽象。人们可以利用所提供的基本数据类型，按数组、记录、集合和文件的构造规则构造有结构的数据。此外，还允许用户利用标准的结构数据类型，通过复合或嵌套构造更复杂、更高层的结构数据。这使得高级语言中的数据类型呈明显的分层。高级语言中数据类型的分层是没有穷尽的，因而用它们可以表达算法语言中任何复杂层次的数据。

在控制方面，高级语言通常提供表达算法控制转移的如下方式：

（1）默认的顺序控制；

（2）条件（分支）控制；

（3）选择（情况）控制；

（4）循环控制；

（5）函数调用，包括递归函数调用；

（6）无条件转移。

以上算法控制转移表达方式不仅满足了算法语言中所有控制表达的要求，而且不再像机器