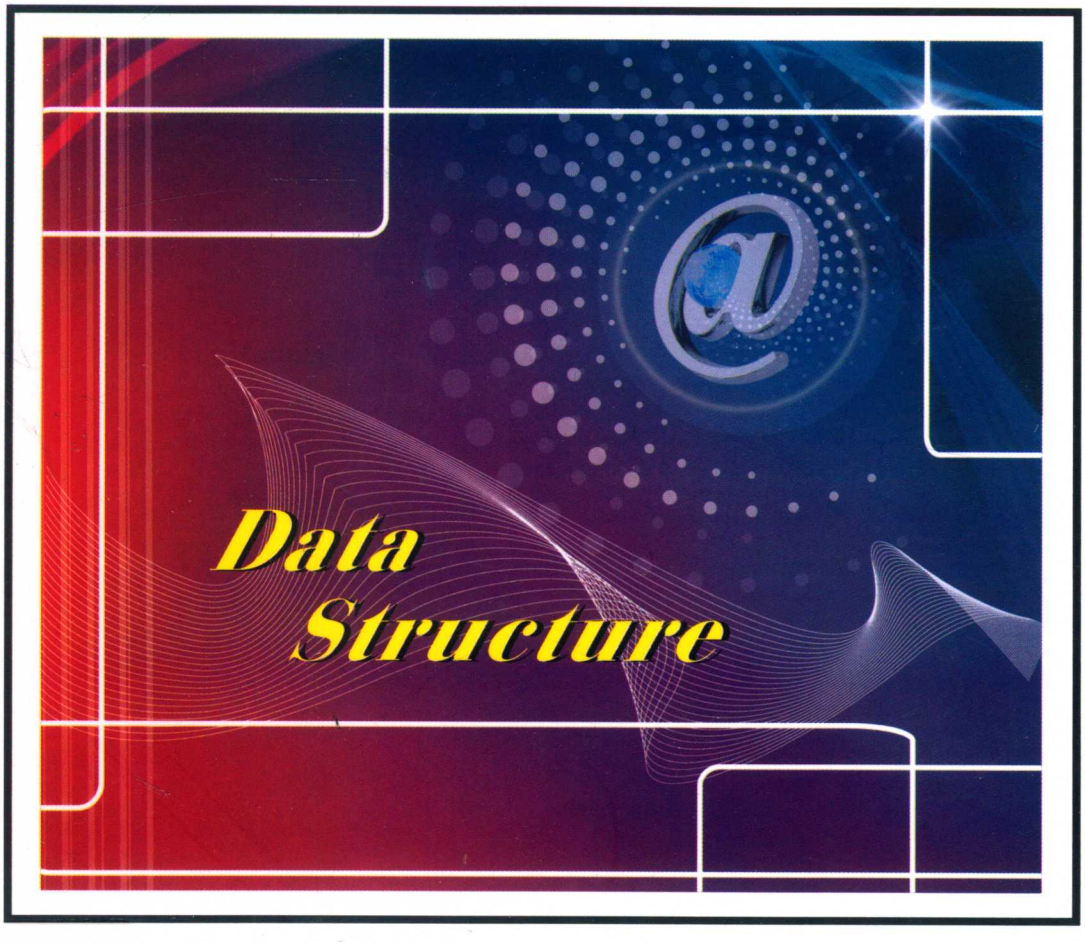
高等学校计算机类"十二五"规划教材

省级精品课程配套教材

数据结构与算法设计

主　编　张小艳　李占利
副主编　齐爱玲　李红卫



西安电子科技大学出版社
http://www.xduph.com

高等学校计算机类“十二五”规划教材

省级精品课程配套教材

数据结构与算法设计

主　编　张小艳　李占利

副主编　齐爱玲　李红卫

西安电子科技大学出版社

内 容 简 介

本书重点介绍了计算机学科中常用的数据结构(包括线性表、栈、队列、串、数组、树、图)的基本概念、逻辑结构、存储结构和在不同存储结构上操作的实现，还介绍了许多经典的查找与排序算法的各种实现过程，并进行了综合分析比较。本书采用C语言描述算法。

本书是在作者多年讲授数据结构课程的实践基础上编写完成的。概念叙述简洁，语言精练，深入浅出，实用性强，同时尽量避免抽象理论的阐述，通过实例分析使读者理解抽象概念。

与本书配套出版的《〈数据结构与算法设计〉实践与题解》，既便于教师教学，也便于学生自学。

本书是省级精品课程配套教材，可作为计算机类专业及信息类相关专业的教材，也可供计算机工程与应用领域技术人员参考。

本书配套课件获教育部多媒体课件比赛三等奖。

图书在版编目(CIP)数据

数据结构与算法设计/张小艳，李占利主编.
—西安：西安电子科技大学出版社，2015.6
高等学校计算机类“十二五”规划教材
ISBN 978-7-5606-3722-8

Ⅰ. ① 数… Ⅱ. ① 张… ② 李… Ⅲ. ① 数据结构—高等学校—教材
② 电子计算机—算法设计—高等学校—教材 Ⅳ. ① TP311.12 ② TP301.6

中国版本图书馆 CIP 数据核字(2015)第 123222 号

策　　划　李惠萍
责任编辑　孟秋黎　李惠萍
出版发行　西安电子科技大学出版社（西安市太白南路 2 号）
电　　话　(029)88242885　88201467　邮　　编　710071
网　　址　www.xduph.com　电子邮箱　xdupfxb001@163.com
经　　销　新华书店
印刷单位　陕西天意印务有限责任公司
版　　次　2015 年 6 月第 1 版　2015 年 6 月第 1 次印刷
开　　本　787 毫米×1092 毫米　1/16　印张 18
字　　数　426 千字
印　　数　1～3000 册
定　　价　32.00 元
ISBN 978 – 7 – 5606 – 3722 – 8 / TP
XDUP 4014001-1

前　言

“数据结构”是计算机学科的一门核心课程，也是其他理工科专业必修或选修的课程。其内容不仅是一般程序设计(特别是非数值型程序设计)的基础，而且是设计和实现编译程序、操作系统、数据库系统及其他系统程序的重要基础。本书就是为“数据结构”课程编写的教材，其内容的选取符合教学大纲的要求。

本书包括四部分内容：数据结构的基本概念(第一章)、基本的数据结构(第二章至第七章)、两种基本技术(第八章和第九章)、经典算法介绍(第十章)。

本书共十章。第一章概括性地介绍了“数据结构”的研究对象，简要介绍了数据、数据元素和数据类型等基本概念，然后对本书中描述算法的方式及算法的度量作了详细的说明。第二章至第七章分别介绍了线性表、栈、队列、串、数组、树、图等数据结构的基本概念、存储结构和不同存储结构上的典型操作的实现；对每种结构从数据元素之间固有的关系出发，通过具体实例给出清晰而恰当的描述，并在讨论基本运算的基础上给出一些应用案例，通过实例分析使学生理解抽象概念。第八、九章讨论了查找及排序的各种实现方法，着重从时间和空间上进行定性或定量的分析与比较。第十章介绍了四种经典的算法的设计思路及实现技巧。本书形式上仍然以传统数据结构的主要内容为主线，但在内容上对传统的数据结构课程进行了更新与扩充。

由于目前C语言已广泛使用，而且数据结构的算法本身又是底层的基本算法，所以，本书采用类C语言作为算法描述语言，将重点放到了问题解决过程的思路和方法上，力求清晰地描述各种基本算法的设计策略。这样可将读者的注意力集中在算法的理解上。本书中所给出的算法需按C语言规范修改之后才可上机运行。学习本书需要有C语言程序设计基础，若具有离散数学和概率论的知识，则对其中某些内容更易理解。

本书的每章后面配备了适量习题供读者进行练习。在习题的选择上，作者

充分考虑了习题的广泛性和典型性。同时与本书配套出版了《〈数据结构与算法设计〉实践与题解》，既方便教师教学参考，也方便学生学习和复习。

本书讲授学时可为48～70。在学时少的情况下，讲授教师可根据学生情况酌情删去某些内容，本书在目录中给出参考性的意见(建议删去带 ** 的章节)。

本书第一、三、六、七章内容由张小艳编写，第九、十章内容由李占利编写，第二、四章内容由齐爱玲编写，第八章内容由李红卫编写，第五章内容由王伯槐编写。全书由张小艳统稿、修改、定稿，李占利审稿。

限于水平有限，书稿虽几经修改仍难免存在不足之处，恳请广大读者批评指正，欢迎交流。

编　者

2015年4月

目　录

第一章 绪 论

如今，计算机已经深入到人类社会的各个领域，其主要应用就是对数据进行加工处理。要对数据进行加工处理，首先就要对数据进行组织。因此，在计算机中如何有效地组织数据和处理数据就是计算机科学的基本研究内容，也是后续课程的基础。

教学目标：本章作为全书的导引，概括性地介绍了“数据结构”的研究对象，进而介绍数据结构中常用的基本概念、术语、算法描述及分析的方法。通过这一章的学习，使读者全面了解数据结构的定义、研究内容以及这门课程的知识体系，从而为后面章节的学习打下基础。

教学提示：数据结构的定义以及算法的时间复杂度的分析是本章的重点。

1.1 数据结构的起源

1946年第一台计算机问世，主要用于军事和科学研究方面的科学计算，人们把计算机理解为数值计算的工具，使用计算机的目的主要是处理数值计算问题。所以用计算机解决一个具体问题时，首先要从具体问题中抽象出一个适当的数学模型，然后设计一个求解此数学模型的算法(algorithm)，最后编出程序并进行测试、调整，直至得到最终解答。

随着计算机科学与技术的不断发展，计算机的应用领域已不再局限于科学计算，而更多地应用于控制、管理等非数值处理领域。与此相应，计算机处理的数据也由纯粹的数值发展到字符、表格、图形、图像、声音等具有一定结构的数据，处理的数据量也越来越大，这就给程序设计带来一个问题：应如何组织待处理的数据以及数据之间的关系(结构)。这类问题涉及的数据结构较为复杂，数据元素之间的相互关系一般无法用数学方程式加以描述。因此，解决这类问题的关键不再是数学分析和计算方法，而是要设计出合适的数据结构。

“数据结构”作为一门独立的课程，最早是美国的一些大学开设的。1968年，美国唐·欧·克努特教授开创了数据结构的最初体系，他所著的《计算机程序设计技巧》第一卷《基本算法》是第一本较系统地阐述数据的逻辑结构和存储结构及其操作的著作。从20世纪60年代末到70年代初，出现了大型程序，软件也相对独立，结构程序设计成为程序设计方法学的主要内容，于是人们也越来越重视数据结构，认为程序设计的实质是对确定的问题选择一种好的结构，加上设计一种好的算法。

“数据结构”在计算机科学中是一门综合性的专业基础课，是介于数学、计算机硬件和计算机软件三者之间的一门核心课程，其内容不仅是一般程序设计(特别是非数值性程序设计)的基础，而且是设计和实现编译程序、操作系统、数据库系统及其他系统程序的重要基础。

1.2 什么是数据结构

什么是数据结构？首先我们来看看什么是数据。

数据是描述客观事物的符号，是信息的载体，它是能够被计算机识别、存储和加工处理的对象。

数据是人们利用文字符号、数字符号以及其他规定的符号对现实世界的事物及其活动所做的描述。数据不仅仅包括整型、实型等数值类型，还可以是文字、表格、图像、声音等非数值类型。比如，我们现在离不开的“百度”，其中有网页、音乐、图片、视频等分类，音乐就是音频数据，图片是图像数据，网页则包括数值、文字、图片等多种数据。

总之，我们这里所说的数据，其实就是具备以下两个条件的符号：

① 可以输入到计算机；

② 能被计算机程序处理。

数据的基本单位称为数据元素，换言之，数据元素是组成数据的、有一定意义的基本单位。

通常，一个数据元素是由用来描述一个特定事物的名称、数量、特征、性质的一组相关信息组成的，在计算机中通常把数据元素作为一个整体进行考虑和处理。多数情况下，一个数据元素可由若干个数据项组成，有时也把数据项称为数据元素的域、字段、关键字。

例如，一所学校对学生的信息管理如图 1.1 所示，其中管理对象就是学生数据。所以，其数据元素就是一个学生的信息；可以用学号、姓名、性别、专业等来表征一个学生，每一个学生的所有信息形成了一个数据元素，通常可以称为一个学生记录。学生的学号、姓名、性别、专业等就是其数据项(或称字段)。

学号	姓名	性别	专　业	年级	……
130102	彭凤姣	女	信息与计算科学	13 级	……
130103	李　丽	女	软件工程	13 级	……
130104	张汉涛	男	信息与计算科学	13 级	……
130105	何颖文	女	计算机应用	13 级	……
130106	高　媛	女	计算机应用	13 级	……
…	…	…	…	…	…

图 1.1　学生信息表

通常，在解决实际应用问题时是把每个学生记录当做一个基本单位进行访问和处理的。

数据项是数据不可分割的最小单位，但是在讨论问题时，数据元素才是数据结构中建立数学模型的着眼点。

我们把具有相同性质的数据元素的集合称为数据对象，它是数据的一个子集。例如，

(1) 字母字符数据对象的集合 C = {'A', 'B', …, 'Z'}，它是字符数据的一个子集；

(2) 偶数数据对象的集合 N = {0, ±2, ±4, …}是整数数据的子集；

(3) 图 1.1 学生表中的学生信息是学生数据的子集。

在具体问题中，数据元素都具有相同的性质(元素值不一定相等)，且属于同一数据对象(数据元素类)。数据元素是数据对象的一个实例。

有了以上概念作为铺垫，下面我们来看什么是数据结构。

在任何问题中，数据元素之间都不会是孤立的，在它们之间都存在着这样或那样的关系，也就是数据的组织形式。

结构，简单理解就是关系，比如分子结构，就是指组成分子的原子之间的排列方式。在现实世界中，数据元素之间不是独立的，而是存在特定关系的，我们将这种关系称为结构。

数据结构是指互相之间存在着一种或多种关系的数据元素的集合。计算机处理的数据是具有一定的组织方式的。比如，可以是表结构，如图 1.1 的学生信息，学生记录之间的关系就是一个紧跟一个的关系。

讨论数据结构的目的是为了在计算机中实现其所需的各种操作。数据结构的操作与其具体问题要求有关。操作的种类和数目不同，即使逻辑结构相同，数据结构的用途也会大不相同(最为典型的例子是第三章所讨论的栈和队列)。定义在数据结构上的操作的种类没有限制，可以根据具体需要而定义。基本的操作主要有以下几种：

(1) 插入：在数据结构中的指定位置上增添新的数据元素。

(2) 删除：删去数据结构中某个指定的数据元素。

(3) 修改：改变数据结构中某个元素的值，在概念上等价于删除和插入操作的组合。

(4) 查找：在数据结构中寻找满足某个特定要求的数据元素的位置或值。

(5) 排序：重新安排数据元素之间的逻辑顺序关系，使数据元素按值由小到大或由大到小的次序排列。

根据插入、删除、修改、查找、排序等操作的特性，所有的操作可以分为两大类：一类是加工型操作，其操作改变了结构的值；另一类是引用型操作，其操作不改变结构的值。

综上所述，数据结构包含三部分：数据、数据之间的关系及在数据集合上的一组操作。也就是说，数据结构是一门研究非数值计算的程序设计问题中的操作对象、对象之间的关系以及在此之上的一系列操作的学科。

1.3 逻辑结构与物理结构

1.3.1 逻辑结构

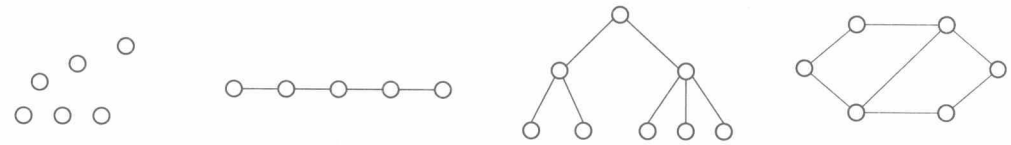
数据的逻辑结构是指数据元素之间的逻辑关系。根据数据元素间关系的不同特性，通常有下列四类基本结构：

(1) 集合结构：在集合结构中，数据元素间的关系是“属于同一个集合”。集合是元素关系极为松散的一种结构，如图 1.2(a)所示。

(2) 线性结构：该结构的数据元素之间存在着一对一的关系，如图 1.2(b)所示。

(3) 树形结构：该结构的数据元素之间存在着一对多的关系。树形结构也称作层次结构，如图 1.2(c)所示。

(4) 图形结构：该结构的数据元素之间存在着多对多的关系。图形结构也称作网状结构，如图 1.2(d)所示。



(a) 集合结构　(b) 线性结构　(c) 树形结构　(d) 图形结构

图 1.2　四类基本结构的示意图

由于集合结构是数据元素之间关系极为松散的一种结构，因此也可用其他结构来表示它。

逻辑结构有两个要素：数据元素集合、关系的集合。在形式上，逻辑结构通常可以采用一个二元组来表示：

Data_Structure=(D，R)

其中，D 是数据元素的有限集，R 是 D 上关系的有限集。

1.3.2　物理结构

物理结构是指数据的逻辑结构在计算机中的存储形式，是逻辑结构在计算机中的实现，包括数据元素的存储及元素之间关系的组织。

数据是数据元素的集合，因此物理结构就是指如何把数据元素存储到计算机的存储器中。存储器主要是针对内存而言的，磁盘、光盘、U 盘等外部存储器的数据组织通常用文件结构来描述。

数据的存储结构要能正确反映数据元素之间的逻辑关系，这是关键。如何存储数据元素之间的关系，是实现物理结构的重点和难点。数据元素的存储结构形式有两种：顺序存储结构和链式存储结构。

1. 顺序存储结构

顺序存储结构是把数据元素存放在地址连续的存储单元中，其数据元素之间的逻辑关系和物理位置一致。

例如，有数据序列(a_1, a_2, a_3, a_4, …, a_n)在存储器中存储如图 1.3 所示。这种数据结构很简单，就像同学们排队进教室占座位，大家按顺序排好，每个人一个座位，不允许插队。顺序存储结构是一种最基本的存储表示方法，通常借助于程序设计语言中的数组来实现。

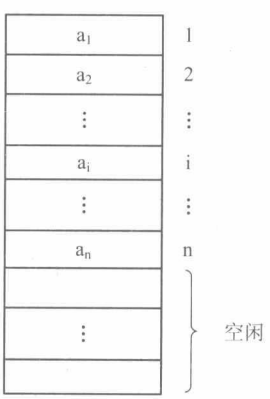


图 1.3　顺序存储结构示意图

在我们学 C 语言时，其中数组就是这样的顺序存储结构。例如，

int a[10];

计算机在内存找一片空地，按照一个整型数据所占位置的大小乘以 10 开辟一段连续的空间，a[0] 放在第一个位置，a[1] 放在第二个位置，以此类推。

2. 链式存储结构

链式存储结构是把数据元素存放在任意的存储单元中，这组存储单元可以连续也可以不连续。其数据元素之间的物理位置不能反映其逻辑关系，因此需要给每个数据元素附设指针字段来存放下一个数据元素所在的位置，也就是说用指针来反映数据元素之间的逻辑关系。设一个指针指向第一个数据元素，通常称为头指针，可以通过头指针找到所有的数据元素的位置。链式存储结构通常借助于程序设计语言中的指针类型来实现。

例如，百家姓的部分姓氏表(zhao, qian, sun, li, zhou, wu, zheng, wang)，是一个线性结构，用链式存储结构存储，如图 1.4 所示。

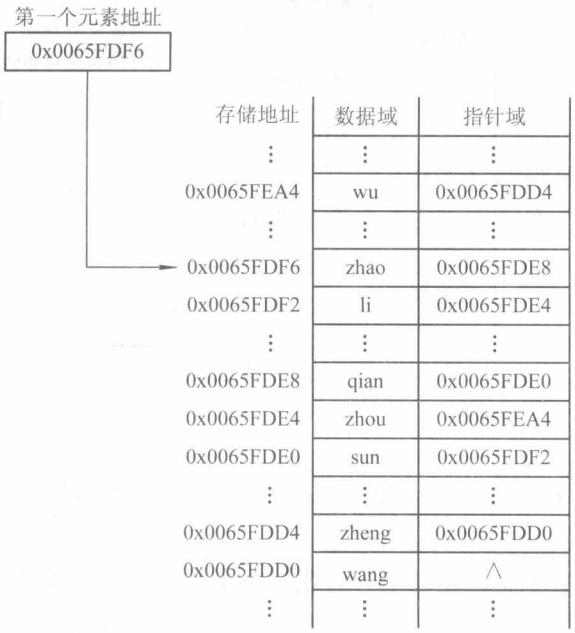


图 1.4 链式存储结构示意图

在这种存储结构下，如何找到表中的任意元素呢？每个数据元素的存储地址放在其直接前驱结点的指针域中，只要知道第一个数据元素的存储地址，就可以“顺藤摸瓜”找到其后继元素。

逻辑结构是面向问题的，而物理结构是面向计算机的，其基本目标就是将数据及其关系存储到计算机的内存中。

1.4 抽象数据类型

1.4.1 数据类型

数据类型是指一个值的集合和定义在这个值集上的一组操作的总称。

数据类型是和数据结构密切相关的一个概念。在用高级语言编写的程序中，每个变量、常量或表达式都有一个它所属的确定的数据类型。数据类型显式地或隐含地规定了在程序

执行期间变量或表达式所有可能的取值范围，以及在这些值上允许进行的操作。

在计算机中，内存是有限的。要充分利用内存，就需要依据需求为应用分配合适大小的空间。因此，对数据进行分类，分出多种数据类型，把数据分成所需内存大小不同的类型，编程时需要用大数据的时候才需要申请大内存。也就是说，数据类型决定了数据占内存的字节数、数据的取值范围、可进行的操作。

在 C 语言中的数据类型如图 1.5 所示。

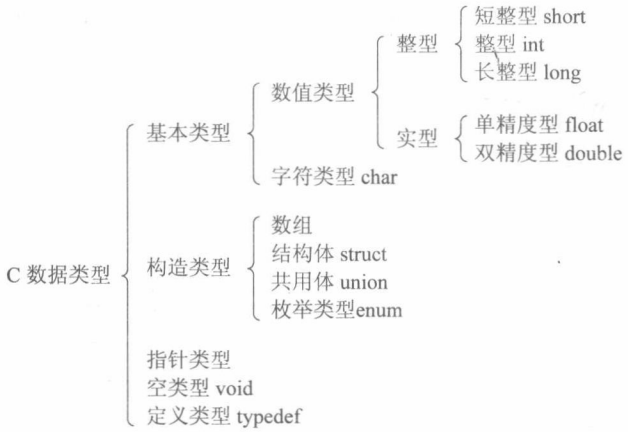


图 1.5　C 语言中的数据类型

例如，在 C 语言中变量声明：

```
int a, b;
```

这里就规定了变量 a、b 在内存中所占的字节数、取值范围(即存放的空间大小)以及施加于 a、b 上的运算(即 int 类型所允许的运算)。

对于高级语言编程者来说，在使用“整型”类型时，既不需要了解在计算机内部是如何表示的，也不需要知道其操作如何实现。如 a + b，设计者仅仅关注其“数学上求和”的抽象特征。因此，我们可以将数据类型进一步抽象，即抽象数据类型。

1.4.2　抽象数据类型

抽象数据类型(Abstract Data Type，ADT)是指一个数学模型以及定义在此数学模型上的一组操作。抽象数据类型需要通过固有数据类型(高级编程语言中已实现的数据类型)来实现。对一个抽象数据类型进行定义时，必须给出它的名字及各操作的名称，即函数名，并且规定这些函数的参数性质。

抽象数据类型的定义仅取决于它的一组逻辑特性，而与其在计算机内部如何表示和实现无关，即不论其内部结构如何变化，只要它的数学特性不变，都不影响其外部的使用。

简单来说，抽象数据类型是将“数据”、“结构”连同对其的“处理操作”封装在一起而形成的复合体。抽象数据类型实际上就是对数据结构的逻辑定义。

例如，将与有序表有关的数据和处理操作封装成一个 ADT，包含数据元素及其关系，操作有初始建表、插入、删除、查找，其描述如下：

```
ADT   OrdList     // OrdList 为抽象数据类型的名字
```

```
{
    数据对象：D = {a_i | a_i∈ElemSet, i=1, 2, …, n, n≥0}    // ElemSet 为数据元素集合
    数据关系：R = {<a_{i-1}, a_i> | a_{i-1}, a_i∈D, i=2, …, n}
    基本操作：
    InitList(&L)            //构造一个空的有序表 L
    ListLength(L)           //输出 L 中数据元素个数
    LocateElem(L, e)        //在表 L 中查找与给定值 e 相等的元素
    ListInsert(&L, i, e)    //在表 L 中第 i 个位置之前插入新的数据元素 e
    ListDelete(*L, i, *e)   //删除表 L 的第 i 个数据元素
}ADT  OrdList
```

1.4.3 抽象数据类型的实现方法

抽象数据类型的实现是要编写相应的程序，实现 ADT 的所有功能。具体来说，就是确定数据结构的存储，使其能够准确地体现 ADT 中定义的数据对象及其关系，并在此基础上实现 ADT 中的所有操作，也就是编写函数实现。

实现 ADT 的方法有三种：封装法、分散法、半封装法。

封装法：将 ADT 中定义的数据及其操作封装成一个整体，比如 C++ 中的类，一个 ADT 对应一个类。这个方法与 ADT 的定义较为一致，数据和处理函数归属明确，符合“面向对象的程序设计方法”的要求。

分散法：将数据和处理数据的函数各自分开。这种方法实现 ADT 可能将属于 ADT 和不属于 ADT 的数据和函数混杂在一起，使得无法从程序的物理结构(即代码的物理次序)上区分哪些数据和函数属于哪个 ADT。例如，我们实现栈的抽象数据类型时，可以用一个数组 elem[ ]存储栈中的元素，再用一个整型变量 top 表示栈顶位置，其操作用一个一个的函数实现。

```
datatype  elem[MAXSIZE];
int  top;
```

半封装法：将 ADT 中的数据和为处理数据需要而定义的相关变量封装在一起形成一个结构，有关处理函数定义在结构之外。这种方法仅做到了对数据存储结构的封装，其特点介于封装法和分散法之间。例如，实现栈的抽象数据类型时，我们把存储栈中元素的数组 elem[ ] 和栈顶位置变量 top 封装在一起，其操作函数实现如下：

```
#define MAXSIZE  <最大元素数>
typedef  struct
{   datatype  elem[MAXSIZE];
    int  top;
} SeqStack
```

本书重点介绍基本数据结构的特点、存储方式和操作实现，没有通过 ADT 描述数据结构，所以没有采用封装法。采用的是半封装法，对数据存储结构进行封装，处理数据的函数分开描述，分开定义，这样能够简化程序结构，减少篇幅，突出算法的核心步骤，

便与阅读和学习。读者可以在理解本书中的各种数据结构的基础上，自行采用封装技术实现之。

1.5 算　　法

用 C 语言编写程序计算 1 + 2 + 3 + 4 + … + 100，大家都会写出这样一段代码：

```
main( )
{    int i, sum=0, n=100;
     for(i=1; i<n; i++) sum=sum+i;
     printf("%d", sum);
}
```

这是最简单的一个计算机程序，它就是一种算法。这个算法效率怎样呢？

实际上对于这个问题，据说高斯在上小学时就给出了一个高效算法：

$$\begin{aligned} \text{sum} &= 1 + 2 + 3 + \cdots + 100 \\ \text{sum} &= 100 + 99 + 98 + \cdots + 1 \\ 2*\text{sum} &= \underbrace{101 + 101 + 101 + \cdots + 101}_{\text{共 } 100 \text{ 个}} \end{aligned}$$

所以 sum=5050。

用程序来实现就很简单了，效率很高。程序如下：

```
main( )
{    int i, sum=0, n=100;
     sum=(1+100)*100/2;
     printf("%d", sum);
}
```

从这个例子可以看出，对于同一个问题可以用不同的方法来解决，我们自然会选择一种效率比较高的方法来解决问题。

1.5.1 算法的基本概念

算法是对特定问题求解步骤的一种描述，是指令的有限序列。其中每一条指令表示一个或多个操作。

在这里我们提到了“指令”，指令是能够被人或计算机等计算设备执行的。它可以是计算机指令，也可以是我们平时的语言文字。

一个算法应该具有下列五个特性：

(1) 有穷性。一个算法必须在有穷步之后结束，即必须在有限时间内完成。

(2) 确定性。算法的每一步必须有确切的定义，无二义性，并且，在任何情况下，算法只有唯一的一条执行路径，即对相同的输入只能得出相同的输出。

(3) 可行性。算法中的每一步都可以通过已经实现的基本运算的有限次执行得以实现。

(4) 输入。一个算法具有零个或多个输入，这些输入取自特定的数据对象。

(5) 输出。一个算法具有一个或多个输出，这些输出同输入之间存在某种特定的关系。

算法的含义与程序十分相似，但又有区别。一个程序不一定满足有穷性。例如操作系统，只要整个系统不遭破坏，它将永远不会停止，即使没有作业需要处理，它仍处于动态等待中，因此操作系统不是一个算法。另一方面，程序中的指令必须是机器可执行的，而算法中的指令则无此限制。算法代表了对问题的求解过程，而程序则是算法在计算机上的特定的实现。一个算法若用程序设计语言来描述，则它就是一个程序。

算法与数据结构是相辅相成的。解决某一特定类型问题的算法可以选定不同的数据结构，而且选择恰当与否直接影响算法的效率；反之，一种数据结构的优劣由各种算法的执行效果来体现。

要设计一个好的算法通常要考虑以下要求：

(1) 正确。算法的执行结果应当满足预先规定的功能和性能要求。算法“正确”的含义大体上可分为四个层次：

一是所设计的程序没有语法错误；

二是所设计的程序对于几组输入数据能够得出满足要求的结果；

三是所设计的程序对于精心选择的典型、苛刻而且带有刁难性的几组输入数据能够得到满足要求的结果；

四是所设计的程序对于一切合法的输入数据都能产生满足要求的结果。

对于这四层含义，其中达到第四层含义下的正确是极其困难的。一般情况下，以达到第三层含义的正确性作为衡量一个程序是否正确的标准。例如，要求 n 个数的最大值问题，算法如下：

```
max:=0;
for(i=1; i<=n; i++)
{    scanf("%f", &x);
     if(x>max)   max=x;
}
```

此算法无语法错误，请考虑，如果输入的数全为负数时，会产生什么结果？其正确性达到了第几层。

(2) 可读。算法首先应该便于人们理解和相互交流，其次才是机器可执行。所以一个算法应当思路清晰，层次分明，简单明了，易读易懂。

(3) 健壮。作为一个好的算法，当输入不合法数据时，应能适当地做出正确反应或进行相应的处理，而不至于产生一些莫名其妙的输出结果。

(4) 高效率低存储。算法效率通常指算法的执行时间。对于同一个问题如果有多个算法可以解决，执行时间短的算法其效率高。所谓存储量的要求，是指算法在执行过程中所需要的最大存储空间。这两者都与问题规模有关。

1.5.2 算法的性能评价

在计算机程序设计中，算法分析是十分重要的。通常对于一个实际问题的解决，可以