普通高等教育船舶类规划教材

计算机软件基础

陈维钧　阎仁武　编著

上海交通大学出版社

高等学校船舶类专业规划教材

计算机软件基础

陈维钧　阎仁武　编著

上海交通大学出版社

内容提要

本书主要阐述计算机软件的基础知识。全书共分三篇，内容包括：数据结构、DOS 操作系统和关系数据库 FoxPro。本书对上述内容从最基本的概念出发，较系统地介绍了计算机软件方面必要的理论知识和算法设计方法。为方便读者的学习，在各章后均附有习题。

本书供工业自动化专业、自动控制专业的学生使用，也可适用于非计算机专业的本科生及广大从事计算机应用工作的科技人员。

责任编辑　东鲁红
封面设计　刘　纬

计算机软件基础
上海交通大学出版社·出版
（上海市华山路 1954 号　邮政编码：200030）
新华书店上海发行所·发行
上海交通大学印刷厂·印刷
开本：787×1092（毫米）1/16　　印张：15　字数：372000
版次：1996 年 1　第 1 版　　印次：1996 年 2 月　第 1 次
印数：1—4000
ISBN7-313-01650—0/TP·291　定价：12.00 元

出版说明

根据国务院国发(1978)23号文件批转试行的“关于高等学校教材 编审出版若干问题的暂行规定”，中国船舶工业总公司负责全国高等学校船舶 类专业教材编审、出版的组织工作。

为了做好这一工作，中国船舶工业总公司相应地成立了“船舶工程”、“ 船舶动力”两个教材委员会和“船电自动化”、“惯性导航及仪器”、“水声 电子工程”、“液压”、“水中兵器”五个教材小组，聘请了有关院校的教授、专家60余人参加工作。船舶类专业教材委员会(小组)是有关船舶类专业教 材建设的研究、指导、规划和评审方面的专家组织，其任务是做好高等学校船 舶类专业教材的编审工作，为提高教材质量而努力。

在总结前三轮教材编审、出版工作的基础上，根据国家教委对“八·五” 规划教材要“抓好重点教材，全面提高质量，适当发展品种，力争系统配套，完善管理体制，加强组织领导”的要求，船舶总公司于1991年又制定了《1991 ～1995年全国高等学校船舶类专业规划教材选题》。列入规划的选题共107种。

这批教材由各有关院校推荐，同行专家评阅，教材委员会(小组)评议，完稿后又经主审人审阅，教材委员会(小组)复审，然后分别由国防工业出版 社、人民交通出版社以及有关高等学校的出版社出版。

为了不断地提高教材质量，希望使用教材的单位和广大师生提出宝贵意见。

中国船舶工业总公司教材编审室

1992年5月

前　言

在当今的社会上，计算机的使用已经日益普及了。在几年以前，计算机还是作为科研和教学的设备在使用，现在它已普及到生产、管理和社会生活的各个领域，甚至已进入到家庭之中。对于非计算机专业的学生，尤其是与计算机休戚相关的工业自动化专业的学生，决不能只满足于会操作使用计算机，还要求具有相当的计算机软件的基础知识，这是提高计算机使用人员的素质，扩大应用软件的开发队伍的重要的环节。

本书是专为工业自动化专业、自动控制专业的学生编写的教材，内容包括了数据结构、DOS 操作系统和关系数据库 FoxPro。数据结构是计算机软件的一门重要的基础课程，系统软件、应用软件的设计和实现都要用到各种数据结构。而操作系统是计算机系统的核心软件，只有在操作系统的支持下，计算机才能运行其他任何软件。数据库系统作为计算机软件的一个重要分支，在各行各业中已得到了最广泛的使用。作为工业自动化专业、自动控制专业的学生，计算机控制技术已成为主要的控制手段，要设计出高质量的应用软件离不开数据结构的基本概念，用 PC 系列工控机实现实时控制系统离不开 DOS 操作系统所提供的各个层次的功能，FoxPro 关系数据库更是进行生产管理的重要软件。所以本书力图把这三部分的基本概念讲清楚，并试图通过书中的实例、每一章的习题帮助读者系统地掌握计算机软件的有关知识，进一步提高使用计算机的能力和研制应用软件的水平。

在学习本教材之前，学生应学习过《C 语言程序设计》和《微机原理》课程，并具有一定的上机能力，因为本书的内容是与它们紧密结合的。例如，第 I 篇“数据结构”中所有程序的例子都是用 C 语言编写的，如果不掌握 C 语言中函数的递归，不掌握指针的使用，那是无法理解有关的程序算法的。第 II 篇“DOS 操作系统”是用汇编语言作为编程语言来分析 DOS 操作系统所提供的各种功能，而 80X86 系列芯片的指令系统和汇编语言是属于《微机原理》课程中的内容，由于受到本课程学时数的限制，有关这些方面的内容在本书中就不再重复了。本书第 III 篇“ 关系数据库 FoxPro”是学生第一次学习有关数据库的知识，所以它是从数据库的基本概念入手，讲述了 FoxPro 数据库的菜单系统，使学生对信息管理中使用得较多的关系数据库有一个基本的了解。在使用本教材时，对上述的各篇的特点要予以重视，并在教学方法上要各有自己的侧重点。

本书的第 I 篇由华东船舶工程学院阎仁武编写，第 II 篇和第 III 篇由上海交通大学陈维钧编写。哈尔滨工业大学蒋重珣教授审阅了全书并提出了许多宝贵的意见。对此，我们表示衷心的感谢。

本书涉及的内容很广，难免会有缺点和问题，敬请广大读者指正。

编者

1995 年 3 月

目　录

第Ⅰ篇　数据结构

第 1 章　数据结构的基本概念

何谓数据结构？这是一个很难用几句话回答清楚的问题，为了要弄清这个概念，首先要弄清什么叫“数据”，然后回答什么叫数据结构。“数据”是人们比较熟悉的概念，一提到数据，大家自然而然想到生活中打交道的“数”了，诚然这种数是一种数据，但今天在此研究的数据绝非仅限于这种“数”，本书所讨论和研究的是现代计算机所能接收和处理的各种各样的数据，一般称之为“信息”(information)。

大家知道，计算机发展的标志不只是指计算机本身运算速度的提高、体积的缩小和信息存储量的增大，更重要的是指计算机应用范围的不断扩大。在计算机出现的初期，它几乎只用于科学计算方面，也就是说，那时所处理的数据就是人们所熟悉的“数值”，并形成了专门的理论支持——计算方法。但是目前计算机应用范围非常广泛，它在商业管理、工业管理、办公室自动化、生产过程控制、系统模拟、人工智能、图像处理以及社会生活的其他方面已显示出巨大的威力。例如火箭、导弹、人造卫星、航天飞机等的出现都是与计算机的作用分不开的。可以看出，计算机对生产、科学、生活的各个领域的影响越来越大，数据的范畴不再局限于数值，对于非数值问题的求解需建立起理论和方法的支持，就逐渐形成了——数据结构。所以，有人说“计算机的出现和发展正在根本地改变着社会”。计算机所以有这样巨大的作用和威力，就在于它能准确地、高速地接收和处理各种各样更为复杂的数据“对象”——通常称为“信息”或“数据”(data)，像数值、字符、图形、声音、颜色等等。因此可以说：数据是描述客观事物的数值、字符以及所有能输入到计算机中并能被计算机程序处理的符号的集合，它是计算机程序使用加工的原料和输出的结果。

数据的基本单位称作数据元素(data element)，它是数据集合中的个体，在有些情况下，也把数据元素称作结点(node)或记录(record)。数据元素是一个相对的概念，它可以是数字、字符、字符串，也可以是更为复杂的对象；有时，一个数据元素可由若干个数据项 (data item)组成，数据项又称为域(field)或字段。

例如，在学生档案管理系统中的数据元素可能是：

学号	姓名	性别	生日	籍贯	成绩

而该数据元素又分为学号、姓名、性别、生日、籍贯和成绩等 6 个数据项。

又如，图书资料检索系统中的数据元素可能是：

编号	书名	作者	出版日期	内容摘要

而该数据元素又分为编号、书名、作者、出版日期和内容摘要等 5 个数据项。如此等等，数据元素还可能是图表、声音等。

现代计算机系统处理的数据不仅越来越复杂，而且数据量也往往大得惊人，如图书馆检索系统需要把整个图书馆中的几十万甚至几百万本图书的有关信息都保存起来，为了迅速、准确地应答用

户的查询，必须对这些数据进行合理有效的组织。若把这么大量的信息杂乱无章地堆积在一起是不可想象的，因为如果真是这样，要查找一本书的信息需要花费大量的机器时间。另外在一些系统中，数据元素间本来就存在着各种各样的复杂的关系，在保存的信息中应该反映出客观存在的这种关系(*relation*)，又称为结构 (*structure*)。如部门组织体系中，领导与被领导的关系，直接领导和上级领导的关系，兄弟单位关系等。这些客观存在的关系是不能随意改变的，应该把这种关系记录下来。因此，在研究数据时，不仅要研究数据本身的值，而且要研究数据元素之间客观存在的各种不同的结构关系，所以简单地说，数据元素间的结构关系就称为数据结构。从程序设计的观点出发，为了设计出高效率、高可靠性的程序，软件工作者不仅要掌握一般的程序设计技巧，而且还必须仔细研究计算机程序加工的对象——数据，要研究所处理数据的数学特性、数据元素间的结构关系、数据在计算机内的存储方式、对数据进行的运算以及实现这些运算的算法等。

例如，一组学员的信息、一组检测到的数据等，这种在逻辑上组内数据元素间仅有一种前后逻辑次序关系(1 对 1)的一组数据元素，称之为线性表。如在图 1.1(a)所示的一组被检测到的数据上，仅关心采集它们时的先后时刻关系，则这些数据及其先后关系一起构成了一个线性表。又如，一家族的各成员(数据元素)构成的族谱，一单位的各部门(数据元素)间的组织体系等，这种组内各数据元素间在逻辑上的关系，就不止是一种逻辑线性关系，而是可以包含多个逻辑关系的层次结构关系(1 对多)，是一种非线性关系，称之为树结构。如图 1.1(b) 所示的学校组织体系，就是一个树结构。再如，一个地区的各个城镇(数据元素)间公路网等，则是一种比树结构更为复杂的数据结构，数据元素间的关系(多对多)是任意的，即任意两城镇间都可能有公路相通，我们称之为图结构，它也是一种非线性结构。如图 1.1(c)为一公路网的图结构示例。

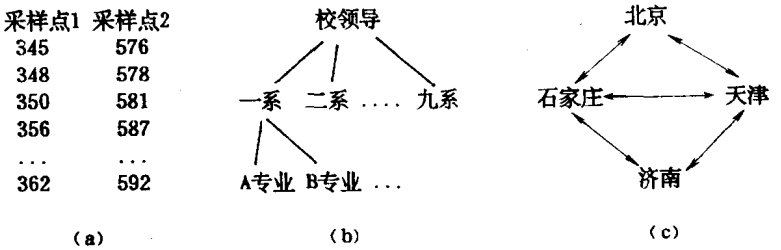

图 1.1　线性表、树结构、图结构示例

通常把数据元素间存在的逻辑结构关系叫做逻辑数据结构，把具有不同逻辑数据结构的数据在计算机内的组织、存储方式叫做存储结构，又称物理数据结构。一种逻辑数据结构可以有不同的存储结构，在给定了一种逻辑数据结构和在其上进行的运算之后，总要选择一种比较有效的存储结构。数据之间的结构关系在计算机中有两种不同的表示方法：顺序映像和非顺序映像。

逻辑数据结构分为两大类：线性结构和非线性结构。线性结构包括字符串、数组、堆栈、队列及其他一般线性表；非线性结构包括树结构和图结构等；这些逻辑结构可以通过顺序映像和非顺序映像得到不同的存储结构。

物理数据结构主要为顺序存储结构和链接存储结构两种。前者是把数据按其数据元素逻辑次序顺序映射到一连续的物理存储区域内，把数据元素间的逻辑结构关系用物理上的位置关系来表征的一种存储结构，即由顺序映象得到；而后者允许把数据元素存储到物理上不连续的各个存储区域中，数据间的逻辑结构关系用附加的链指针字段来表征的一种存储结构，即由非顺序映像得到。

所以，对任一种逻辑数据结构都有两种不同的存储结构，即顺序存储的线性结构、顺序存储的非线性结构、链接存储的线性结构、链接存储的非线性结构。在不引起混淆的情况下，通常把逻辑数据结构简称为数据结构。

第 2 章　线性表的顺序分配及算法

2.1　线性表概述

线性结构是社会生活各个领域中常见的一类数据结构，因此作为信息处理的计算机装置来说，线性表是它处理的最常见、最简单的数据结构。诚如第 1 章中所述，线性表的存储结构有顺序存储结构和链接存储结构之分，分别称之为线性表的顺序分配和链接分配。本篇将研究这种线性表逻辑结构的特性、对它施加的运算、讨论对这种逻辑结构的两种存储结构——顺序分配和链接分配，并给出在不同存储结构下的种种算法，特别是插入和删除算法。本章讨论线性表的一般概念及其顺序分配和算法，第 3 章讨论线性表的链接分配及算法。

2.1.1　什么是线性表

线性表(linear list)的逻辑结构是 n 个结点 (node)，即数据元素 X(1)，X(2)，…，X(n) 的有序集合(n≥0)。它的结构性质仅仅涉及到这些结点的线性(一维)相对位置。如当 n＞0 时，X(1)是第一个结点，或称头结点；X(n)是最后一个结点，或称尾结点；当 1＜k＜n 时，X(k)是第 k 个结点，则称 X(k－1)是相对于 X(k)的前趋结点，X(k＋1) 是相对于 X(k)的后继结点。表内结点的个数 n 称为线性表的长度，当 n＝0 时，称线性表为空表。表的头、尾结点又称表的端点。

一般说来，结点是表征某一数据结构特点及其逻辑联接方式的基本单位，一个线性表的所有结点应该具有相同的性质，即描述同一种类型的事物。结点本身又往往分成若干字段或称域(Field)，每一字段存储与该结点有关的信息，但从整体上看，结点是不可再分的基本单位，称为原子项(Atomic element)。例若干学生的信息构成一个线性表，每个学员信息是一个结点，各结点性质相同，又分为学号、姓名、性别等若干描述学生信息的字段。

2.1.2　对线性表施加的运算

众所周知，在整数集合上可进行加、减运算，在实数集合上可进行加、减、乘、除等运算，相应地，在线性表集合上常施加的运算(或操作)有以下几种：

(1)存取第 k 个结点，以便检验或改变它的某些字段的内容；

(2)在表中给定的位置插入一个新结点，例第 k 个结点前(或后)插入一个新结点；

(3)删除某一结点；

(4)把一个线性表分拆成两个或多个线性表；

(5)复制一个线性表；

(6)把两个或多个线性表合并成一个线性表；

(7)求出线性表中结点的数目，即线性表的求长度；

(8)对线性表中的结点重新排序，使得按某个(些)字段的值而言是递增(或减)的；

(9)在表中查找某个字段具有特定值的结点。

由于运算和线性表的存储结构关系密切，我们应根据对线性表实际施加的操作，既考虑到数据本身的固有特性，又注意其存储结构表示。

2.2 一般线性表的顺序存储结构及其插入删除算法

2.2.1 线性表的顺序存储结构

在计算机内部表示线性表的最简单和最普遍的方法是用一组连续的内存空间来依次存放表中的结点，若每个结点占 C 个机器字，则第 i+1 个结点的存储位置 LOC(X(i+1))与第 i 个结点的存储位置 LOC(X(i))满足下面的关系：

$$LOC(X(i+1))=LOC(X(i))+C$$

一般地，第 i 个结点 X(i)的存储位置与第一个结点 X(1)的位置关系为：

$LOC(X(i))=LOC(X(1))+(i-1)*C$　或写为：$LOC(X(i))=Base+(i-1)*C$

其中，Base 称为基地址，是该线性表第一个结点的起始地址。

线性表的这种机内表示法称作线性表的顺序分配或顺序映像（Sequential mapping）。如图 2.1 所示。

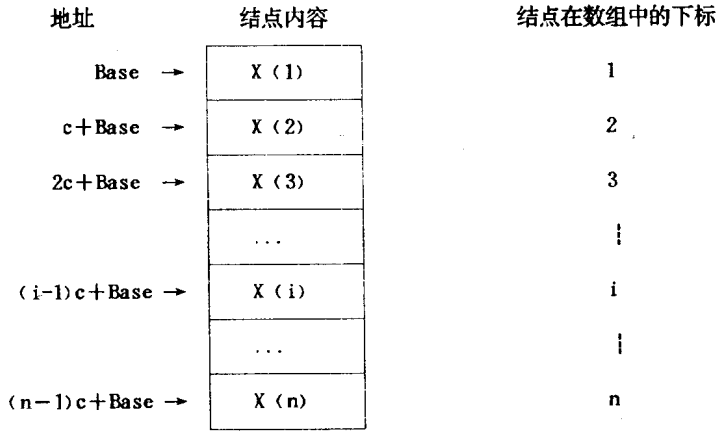

图 2.1　线性表的顺序分配示意图

这种存储方式的特点是线性表中元素的逻辑次序和物理存储次序一致，每个结点的位置与第一个结点的位置相差一个与序号成正比的常数。由于计算机的内存是随机存取装置，所以顺序分配的线性表也是一种随机存取的结构。

可以用一维数组说明和描述这种顺序分配：elementtype linearlist[maxlen]

可见，只要给出任一结点的下标，其地址就可按一个统一的公式计算出来，表中任一结点的存取时间都相同。这种顺序分配方式结构简单，便于随机访问表内的任一结点，用于表示数据项不常变动的线性表是很好的；若表中的数据项变动频繁，即插入和删除操作经常进行，这种结构就显得笨拙了。通过下面的考察能清楚地说明这点。

2.2.2 插入和删除算法

为简单起见，令结点仅有一个整数域，即 int linear[maxlength]。

其中 linear[0]存放当前线性表的长度，而最大长度为 maxlength－1。

(1)插入运算：insertl(i,x) 将整数 X 插入到表中的第 i 个元素之后。

①若 i>linear[0] 或 i<0 则指出位置错误并返回；

若 linear[0]=maxlength−1 则显示满并返回；否则转②；

②将第 linear[0]至第(i+1)位置上的元素依次向后移动一个位置；

③linear[i+1]=x；linear[0]=linear[0]+1； 返回。

(2)删除运算：deletel(i,y) 将第 i 个元素删除且保存于 y 中。

①若 i<1 或 i>linear[0] 则指出位置错误并返回；

若 linear[0]=0 则显示空并返回；否则转(2)；

② y=linear[i]；

③ 将第 i+1 至 linear[0]位置上的元素依次向前移动一个位置；

④ linear[0]=linear[0]−1；返回。

2.2.3 插入算法分析

假设有 n 个结点的线性表 X(1),X(2),…,X(n)都是常数，且按从小到大的次序排列，现如果要在第 i 个结点后插入一个结点 b，为了保持线性表的有序性，应满足：

$X(i)\leqslant b\leqslant X(i+1)$ 即将 b 插入 X(i)之后，则 X(n),X(n−1),…, X(i+1)共 n−i 个结点需依次向后移动一个位置。

设 Pi 为在第 i 个结点之后插入一项的概率，且在每一项之后插入的概率相等，即

$P_i=1/(n+1)$ 则在长度为 n 的线性表中插入一项需移动结点的平均值为：

$$Vinsert=\sum_{i=0}^{n} P_i(n-i)=\frac{n}{2}$$

最坏情况下需移动全部 n 个结点，最好情况需移动 0 个结点。

2.2.4 删除算法分析

相应地，若删除表中的第 i 个结点，为了保持线性表的有序性，则其后的 n−i 个元素 X(i+1),X(i+2),…,X(n)需依次向前移动一个位置。

若设删除第 i 个结点的概率为 P_i，且对每个结点的删除概率相等，即 $P_i=1/n$，则删除一个点所需移动结点的平均值为：

$$Vdelet=\sum_{i=1}^{n} P_i(n-i)=\frac{n-1}{2}$$

最坏情况为移动 n−1 个结点，最好情况为 0。

由此可见，对于顺序分配的线性表进行一次修改(插入或删除)，平均需移动表中一半结点，当表相当大时效率则很低，但对某一类特殊线性表——堆栈和队列来说却是一种十分有效、十分方便的存储结构。因为插入和删除运算都在表的一端进行，除非多表共享时，平均移动次数永远是最好情况，即插入移动次数为 0(在第 n 个结点之后)，删除移动次数也为 0(删除第 n 个结点)。

[例 2.1] 顺序分配的线性表插入和删除的 C 语言程序如下：

```
#define maxlen 5
#include "stdio.h"
int linear[maxlen]; /*说明了一个顺序分配的线性表*/
main()
```

```
{int ai,ax,ay; char ch; ch='3'; linear[0]=0;
while(ch! ='4'){
  printf("\n");
  printf("1. Insert a element\n");        /* 插入一元素 */
  printf("2. Delete a element\n");        /* 删除一元素 */
  printf("3. Display the content\n");     /* 显示线性表的内容 */
  printf("4. Quit\n");                    /* 退出 */
  printf("Enter your choice:\n");
  ch=getche();                            /* 请选择相应的操作(1~4) */
  printf("\n");
  switch(ch){
    case '1':  pri~tf("Enter your order:");scanf("%d",&ai);
               printf("Enter your number:");scanf("%d",&ax);
               insertl(ai,ax);  break; /* 将 ax 插入到第 ai 个元素之后 */
    case '2':  printf("Enter your order:");  scanf("%d",&ai);  ay=-1;
               deletel(ai,ay);  break; /* 将第 ai 个元素删除到 ay 之中 */
    case '3':  display(); break;
    case '4':  exit(0);
         };};};
  int insertl(int i,int x)   /* 将 x 插入到第 i 个元素之后之插入运算 */
  {int j;
  if (linear[0]==i-1) {linear[i]=x; linear[0]++; }
    else{ if (i<=linear[0] && i>=0){
                if (linear[0]<maxlen-1){
                        for (j=linear[0]+1;j>i+1;j=j-1) linear[j]=linear[j-1];
                        linear[i+1]=x;
                        linear[0]++; }
                else printf("The linear list is overflow! \n");}
          else printf("Insert point error! \n"); };
  };
  int deletel(int i,int y)    /* 删除运算 */
  {int j;
    if(i>=1 && i<=linear[0])
    {if(linear[0]>0){ y=linear[i];
          for (j=i+1;j<=linear[0];j++) linear[j-1]=linear[j];
          linear[0]--; return(y); }
      else printf("The linear list is empty!\n"); }
    else printf("Delete position error!\n");
  };
  display()  /* 显示线性表的内容 */
  {int i;  for (i=0;i<=linear[0];i++)
    printf("%s %d %s %d %s","linear[",i,"] is:",linear[i]," ; ");
  };
```

2.2.5 线性表应用举例

[例2.2] 设有 n 个人围成一圈，从第 s 个人开始进行1到 m 报数，数到第 m 个的人出列，然后再从下一个人开始报数，数到第 m 个人再出列，如此反复报数，直到所有的人全部出列为止。现要求按出列次序得到 n 个人员的顺序表。此问题也称为 Josephus 问题。

设想有一组整数1至 n 作为待报数的序列，它们被置于数组 P 中。若第一次是 P(i)出列，则 P

(i+1)至P(n)依次前移一个位置，并将P(i)放入P(n)中，而第二次报数是对剩下的n－1个元素进行，再将报出的数放于P(n－1)中.....如此反复，最后P中保留了报出序列的逆序。将数组P元素按逆序输出，便得到了按次序报数出列人员的顺序表。

明显地，上次报数的出列位置，由于其后的未出列元素均前移一个位置，便成为下次报数的起始位置，设为s1。因为报数是首尾连续进行的，可以用取模的方法由本次的报数起始位置推算得下次的出列位置，若设i为每次报数的人数，则每次出列位置由下式获得，其中％为模除运算符。

$$s1 \leftarrow (s1+m-1) \% i$$

C语言程序如下：

```
#define n 8
main()
{int p[n],i,j,m,s,w;
m=4;s=0; /*p[]为一线性表*/
for (i=0;i<n;i++) p[i]=i+1;
for (i=n;i>1;i--){s=(s+m-1)%i; /*置为环形表*/
                  w=p[s];
                  for (j=s;j<=i-1;j++) p[j]=p[j+1];
                  p[i-1]=w; };
for (i=n-1;i>=0;i--) printf("  %d %s %d",n-i,"--",p[i]);
};
```

运行结构：

1—4　2—8　3—5　4—2　5—1　6—3　7—7　8—6

2.3 堆　栈

2.3.1 堆栈的一般概念

堆栈(Stack)是一种特殊的线性表，对它的插入和删除运算只在表的一端进行，表中允许插入和删除的一端称为栈顶(top)，不允许插入和删除操作的一端称为栈底(bottom)。堆栈是一种动态的数据结构，即对堆栈的插入和删除运算随时都可能发生。假定在某一时刻，栈$S=(a_1 \cdots a_n)$，则a_1是栈底结点，a_n是栈顶结点，表中的数据是以$a_1 \cdots a_n$的次序进栈的，而退栈的结点总是栈顶亦即最后进栈的那个数据。因此堆栈又称为后进先出表(LIFO－Last In First Out)，或称先进后出表(FILO－Frist In Last Out)。

2.3.2 堆栈的顺序分配和运算

堆栈的顺序分配一般是用一片连续的存储空间来存储栈内元素，通常是用一个一维数组来表示栈区，数组下标的下界为栈底位置，上界为栈区的最后一个单元的位置，数组的体积表示栈中元素的最大数目，通常栈底不动，用一个指针top来指示栈顶元素的位置。由于堆栈的大小一旦设定便不变动，所以栈会满，称为上溢，不妨设当top为数组下标上界时表示栈上溢；当top为下界减一时表示栈空(下溢)，则top指出了下次退栈的位置，而下次进栈位置为top＋1所指出的位置。

在C语言中可用struct elementtype s[stacklength]说明之。

图2.2指出了栈中结点和栈顶之间的关系，图中假定栈区的地址为1至m，且每个结点只占一个单元，后面关于栈的插入和删除算法也以这种假设为前提。

通常对堆栈施加的运算有：

(1)入栈或称进栈及压栈——在栈S的顶上加入一个新的结点x，记为push(x,S)；