2010

全国硕士研究生入学统一考试

计算机

专业基础综合考试辅导与模拟试题

浙江大学计算机学院 编著

- 数据结构：陈 越 何钦铭 冯 雁
- 计算机组成：楼学庆
- 操作系统：李善平 季江民 钱 徽
- 计算机网络：张泉方 郑扣根 邱劲松 陆魁军 黄正谦

ZHEJIANG UNIVERSITY PRESS
浙江大学出版社

全国硕士研究生入学统一考试

计算机专业基础综合
考试辅导与模拟试题

浙江大学计算机学院　编著

ZHEJIANG UNIVERSITY PRESS
浙江大学出版社

图书在版编目（CIP）数据

全国硕士研究生入学统一考试计算机专业基础综合考试辅导与模拟试题 / 浙江大学计算机学院编著. —杭州：浙江大学出版社，2009.5
（全国硕士研究生入学统一考试辅导系列）
ISBN 978-7-308-06769-0

Ⅰ.全… Ⅱ.浙… Ⅲ.电子计算机－研究生－入学考试－自学参考资料 Ⅳ.TP3

中国版本图书馆 CIP 数据核字（2009）第 073653 号

全国硕士研究生入学统一考试计算机专业基础综合考试辅导与模拟试题

浙江大学计算机学院 编著

丛书策划 樊晓燕
责任编辑 黄娟琴
封面设计 刘依群
出版发行 浙江大学出版社
（杭州天目山路 148 号 邮政编码 310028）
（网址：http://www.zjupress.com）
排 版 杭州中大图文设计有限公司
印 刷 富阳市育才印刷有限公司
开 本 787mm×1092mm 1/16
印 张 22.75
字 数 582 千
版 印 次 2009 年 5 月第 1 版 2009 年 5 月第 1 次印刷
书 号 ISBN 978-7-308-06769-0
定 价 45.00 元

本书编委会

数据结构 陈 越 何钦铭 冯 雁

计算机组成 楼学庆

操作系统 李善平 季江民 钱 徽

计算机网络 张泉方 郑扣根 邱劲松 陆魁军 黄正谦

前　言

计算机学科专业基础综合科目(涵盖“数据结构”、“计算机组成原理”、“操作系统”和“计算机网络”)的全国联考,是2009年教育部对初试科目调整的结果,旨在考查考生是否比较系统地掌握了上述四门专业基础课程的概念、基本原理和方法,并能够运用所学的基本原理和基本方法分析、判断和解决有关理论问题和实际问题。

我们集合了浙江大学计算机学院在这四门课程教学中具有最丰富经验的教师,对教育部编制的考研大纲进行了充分细致的研读,对2009年的第一套真题进行了详细的分析,又参阅了许多优秀教材以及已经出版的各种考研辅导资料,集体编写了这本辅导书,希望对准备考研的莘莘学子有所帮助。

本书由“考试辅导”、“真题解析”、“模拟试题”和“参考答案”四部分组成。

在“考试辅导”部分,我们对四门课的知识点做了一个精简的复习,力求做到紧扣大纲,没有遗漏,也没有延展。我们认识到,复习资料跟普通教材不同,备考者需要在尽可能短的时间里温习已经学过的知识,因此不需要辅导资料给出深入浅出的分析讲解,只需要给出一个明确简洁的知识结构和条理。我们将每门课的复习篇幅控制在50页左右,努力在覆盖大纲要求的全部知识点的基础上,尽可能节省备考者的时间,帮助大家有的放矢地进行复习。

在“真题解析”部分,我们根据2009年真题的官方统一试卷参考答案及评分标准,对试题进行了详细解读,不仅给出了解答,而且分析了题目考查的知识点、要点、难点以及判卷中发现的典型错误等,帮助备考者把握各科目考试的难度,以进行更加有效的复习。

在“模拟试题”和“参考答案”部分,我们提供了10套模拟试卷,并且给出了参考答案及详尽的分析,供备考者在复习的后期做参考和练习。这些题目中有相当一部分是编写者根据首套真题的难度、严格按照统考大纲有针对性地出的新题,也有一部分是参考了已经出版的各种考研辅导资料,其中对国内各大高校往年的专业考试题目多有借鉴,在此特别表示感谢。

本书编写者阵容强大。浙江大学计算机学院的数据结构、计算机组成原理、操作系统和计算机网络均是不同级别的精品课程，参加本书编写的人员均为课程的主讲教师，长期工作在教学一线，具有丰富的教学经验。其中“数据结构”是国家双语示范课程、浙江省精品课程，由陈越、何钦铭、冯雁主笔该课程相关的内容；“计算机组成原理”是教育部－Intel精品课程，由楼学庆主笔该课程相关的内容；“操作系统”是国家精品课程，由李善平、季江民、钱徽主笔该课程相关的内容；“计算机网络”是国家双语示范课程，由张泉方、郑扣根、邱劲松、陆魁军、黄正谦主笔该课程相关的内容。

由于计算机学科专业基础综合科目的联考仅仅展开了一年，我们编写辅导书的参考依据尚不十分充分，编写时间也比较仓促，书中难免有错漏之处，诚恳希望读者和专家批评指正，我们将不胜感激。有关反馈意见，可发送电子邮件到 chenyue@ zju. edu. cn，进行交流。

编　者

2009 年 5 月

目　录

计算机专业综合考试总体分析 …… (1)
计算机专业综合考试考查内容解析 …… (2)
第一部分　数据结构 …… (2)
第二部分　计算机组成原理 …… (50)
第三部分　操作系统 …… (100)
第四部分　计算机网络 …… (143)

2009 年计算机科学与技术学科全国硕士研究生入学统一试卷解析 …… (201)

模拟试题 …… (223)

模拟试题(一) …… (223)
模拟试题(二) …… (230)
模拟试题(三) …… (236)
模拟试题(四) …… (242)
模拟试题(五) …… (247)
模拟试题(六) …… (253)
模拟试题(七) …… (259)
模拟试题(八) …… (265)
模拟试题(九) …… (270)
模拟试题(十) …… (275)

模拟试题参考答案及解析 …… (281)

模拟试题(一)参考答案及解析 …… (281)
模拟试题(二)参考答案及解析 …… (289)
模拟试题(三)参考答案及解析 …… (295)
模拟试题(四)参考答案及解析 …… (301)
模拟试题(五)参考答案及解析 …… (308)
模拟试题(六)参考答案及解析 …… (315)
模拟试题(七)参考答案及解析 …… (322)
模拟试题(八)参考答案及解析 …… (330)
模拟试题(九)参考答案及解析 …… (338)
模拟试题(十)参考答案及解析 …… (346)
参考文献 …… (354)

计算机专业综合考试总体分析

全国硕士研究生入学统一考试是为全国高等院校和科研院所招收硕士研究生而设置的具有选拔性质的考试。计算机学科专业基础综合科目(涵盖数据结构、计算机组成原理、操作系统和计算机网络)的全国联考,是2009年教育部对初试科目调整的结果,旨在考查考生是否比较系统地掌握了上述四门专业基础课程的概念、基本原理和方法,并能够运用所学的基本原理和基本方法分析、判断和解决有关理论问题和实际问题。

考试大纲规定试卷满分为150分,考试时间为180分钟。答题方式为闭卷、笔试。

根据考试大纲的规定,试卷的内容结构为:数据结构45分,占30%;计算机组成原理45分,占30%;操作系统35分,约占23%;计算机网络25分,约占17%。试卷题型结构为:单项选择题80分(40小题,每小题2分),综合应用题70分。

从2009年的真题看,试卷题目分布如下:

(1)数据结构范围内的选择题10题(第1~10题),每题2分,共20分;综合应用题2题(第41、42题),共25分;总计是45分。

(2)计算机组成原理范围内的选择题12题(第11~22题),每题2分,共24分;综合应用题2题(第43、44题),共21分;总计是45分。

(3)操作系统范围内的选择题10题(第23~32题),每题2分,共20分;综合应用题2题(第45、46题),共15分;总计是35分。

(4)计算机网络范围内的选择题8题(第33~40题),每题2分,共16分;综合应用题1题(第47题),共9分;总计是25分。

上面的分析显示,2009年的首次全国硕士研究生入学统一考试计算机学科专业基础综合试卷是严格按照考试大纲规定的内容结构和题型结构出的,试题基本上涵盖了各科目最基础、最重要的知识点。

为了便于读者掌握考试要点,进行实战准备,以下我们将严格按照考试大纲规定的考查范围,分为数据结构、计算机组成原理、操作系统、计算机网络四方面对考试大纲进行解析。

计算机专业综合考试考查内容解析

第一部分　数据结构

【考查目标】

1. 理解数据结构的基本概念；掌握数据的逻辑结构、存储结构及其差异以及各种基本操作的实现。

2. 掌握基本的数据处理原理和方法的基础上，能够对算法进行设计与分析。

3. 能够选择合适的数据结构和方法进行问题求解。

【考查内容】

一、线性表

（一）线性表的定义和基本操作

（二）线性表的实现

1. 顺序存储结构

2. 链式存储结构

3. 线性表的应用

二、栈、队列和数组

（一）栈和队列的基本概念

1. 栈

2. 队列

（二）栈和队列的顺序存储结构

1. 栈的顺序存储结构

2. 队列的顺序存储结构

（三）栈和队列的链式存储结构

1. 栈的链式存储结构

2. 队列的链式存储结构

（四）栈和队列的应用

1. 栈的应用：表达式求值

2. 栈的应用：迷宫问题

（五）特殊矩阵的压缩存储

1. 对称矩阵的压缩存储
2. 对角矩阵的压缩存储

三、树与二叉树

(一)树的概念

(二)二叉树

1. 二叉树的定义及其主要特征
2. 二叉树的顺序存储结构和链式存储结构
3. 二叉树的遍历
4. 线索二叉树的基本概念和构造
5. 二叉排序树
6. 平衡二叉树

(三)树、森林

1. 树的存储结构
2. 森林与二叉树的转换
3. 树和森林的遍历

(四)树的应用

1. 等价类问题
2. 哈夫曼(Huffman)树和哈夫曼编码

四、图

(一)图的概念

(二)图的存储及基本操作

1. 邻接矩阵法
2. 邻接表法

(三)图的遍历

1. 深度优先搜索
2. 广度优先搜索

(四)图的基本应用及其复杂度分析

1. 最小(代价)生成树
2. 最短路径
3. 拓扑排序
4. 关键路径

五、查找

(一)查找的基本概念

(二)顺序查找法

(三)折半查找法

(四)B - 树

1. B - 树的查找
2. B - 树的插入
3. B - 树的删除

(五)散列(Hash)表及其查找

1. 散列函数的设计
2. 冲突解决方法
（六）查找算法的分析及应用
六、内部排序
（一）排序的基本概念
（二）插入排序
1. 直接插入排序
2. 折半插入排序
（三）气泡排序（Bubble Sort）
（四）简单选择排序
（五）希尔排序（Shell Sort）
（六）快速排序
（七）堆排序
（八）二路归并排序（Merge Sort）
（九）基数排序
（十）各种内部排序算法的比较
（十一）内部排序算法的应用

一、线性表

（一）线性表的定义和基本操作

线性表（Linear List）是由若干个元素组成的一个有序序列。设序列中的第 i 个元素为 A_i，$1 \leqslant i \leqslant N$，则线性表可以表示为：

$(A_1, A_2, A_3, \cdots, A_{i-1}, A_i, A_{i+1}, \cdots, A_N)$

称 A_{i-1} 为 A_i 的前驱元素，A_{i+1} 为 A_i 的后继元素，线性表中的元素个数 N 称为线性表的长度；$N=0$ 时称为空表，i 称为元素 A_i 的位序。

线性表的基本操作主要有：

- 初始化 MakeEmpty(L)：构造一个空的线性表 L。
- 随机访问 FindKth(i,L)：根据指定的位序 i，返回相应元素 A_i。
- 定位查找 Find(X,L)：已知 X，返回线性表中与 X 相同的第一个元素的相应位序 i。若不存在返回空。
- 插入 Insert(X,i,L)：在指定位序 i 前，插入一个新元素 X。
- 删除 Delete(i,L)：删除指定位序 i 的元素。
- 求表长 Length(L)：返回线性表 L 的长度 N。
- 倒序 Reverse(L)：将给定的线性表 L 逆转。

（二）线性表的实现

与大多数数据结构一样，线性表也可分别用顺序存储结构和链式存储结构实现。

1. 顺序存储结构

数组是线性表最简单、直观的顺序存储结构实现方法。可用数组的各分量存储元素的内容，数组的下标来表示元素的位置。这样，FindKth 操作的时间代价为常数，但由于数组的连续存放特性使得 Insert 和 Delete 操作要付出较大的代价。例如，删除表的第一个元素或在表的第一个元素前插入新元素，都要顺序挪动表中的其他所有元素，因此这两个操作在最差的情况下复杂度为 $O(N)$。用数组表示线性表的另一个缺陷是数组的长度必须事先确定，即便是动态分配空间也是如此。

2. 链式存储结构

链表是线性表的链式存储结构实现方式。在链表中，每个结点是一个包含数据域和指针域的结构。数据域反映了元素本身的信息，指针域则通过后继或前驱指针反映元素之间的逻辑关系。

链表的主要形式有：单向链表、双向链表、循环链表等。

(1)单向链表

单向链表的每个结点除包含数据域外，只有一个指向后继结点的指针。一般有一个指针 head 指向该链表的第一个元素，同时最后一个结点的指针域为 NULL(或 0)，如图 1-1。

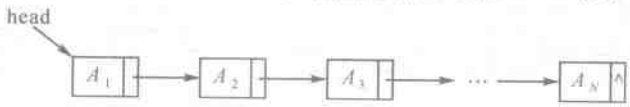


图 1-1　单向链表的表示

实际应用中常用一个称为“头结点”的指针指向链表的第一个单元，并用它表示一个具体的链表，图 1-2 表示为有头结点 header 的链表的图示形式。

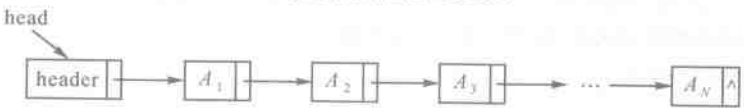


图 1-2　有头结点的单向链表

单向链表中每个结点的结构类型一般为：

```
typedef struct  node {
   ElemType  info;          /* 数据域 */
   struct node *next;         /*指针域 */
}Node;
```

链表表示使得删除线性表中的元素不必再移动数据，例如要删除元素 A_i，只要将这个单元的指针值赋给其前一单元 A_{i-1} 的指针域就行了。图 1-3 表示了删除 A_i 的数据变化情况，A_{i-1} 的原来指针值用虚箭头表示，实的弧线表示删除 A_i 后的指针值。注意：需要释放被删除的结点 A_i 的空间。

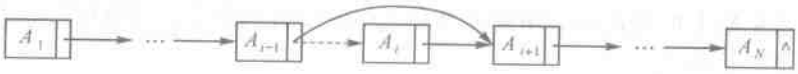


图 1-3　删除数据单元

与删除元素类似，线性表中插入新的元素也不必移动数据。一般在结点 A_i 后插入新结点 X 的步骤为：

- 为一个新结点分配空间；

- 将数据元素 X 存入新结点的数据域；
- 将后继结点 A_{i+1} 的地址（即 A_i 指针域值）赋给新结点的指针域；
- 修改 A_i 指针域的指针值，使其指向新的结点。

例如，图 1-4 表示 A_i 之后插入元素 X 的指针变化情况，首先将 A_i 指针值赋给新数据元素 X 单元的指针域，然后修改 A_i 指针值使其指向 X 单元。

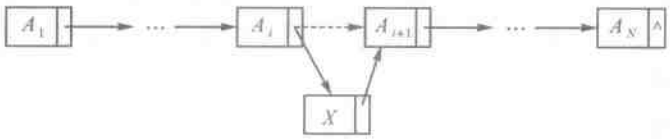


图 1-4　插入数据单元

通过把单向链表的最后一个单元的空指针改为指向第一个单元，就可以把一个单向链表改造成循环链表，如图 1-5(a)所示。图 1-5(b)是一个带有头结点的非空循环链表，图 1-5(c)则是一个带有头结点的空的循环链表。

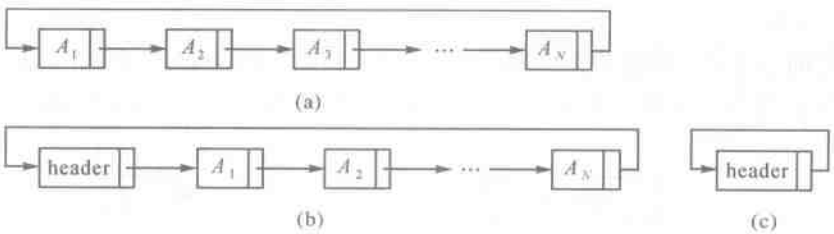


图 1-5　单向循环链表

下面给出将单向链表 h 原地逆转的程序实现：

```
struct node *Reverse(struct node *h)
{ //将不带头结点的单链表就地逆转,返回新链表的首地址
    struct node *p, *q, *t;
    p=h;
    q=NULL;
    while (p!=NULL) {
          t=p->next;
          p->next=q;
          q=p;
          p=t;
    }
    return q;     //返回逆转后的新链表的首地址
}
```

(2)双向链表

在单向链表基础上增加指向前驱单元指针（prior 指针）的链表叫做双向链表。图 1-6 是双向链表的图示表示形式。

图 1-6　双向链表

双向链表中每个结点的结构类型一般为：

```
typedef struct  node {
    ElemType  info;          /* 数据域 */
    struct node *prior;       /*前驱指针 */
    struct node *next;        /*后继指针 */
} Node;
```

如果将双向链表最后一个单元的 next 指针指向链表的第一个单元，而第一个单元的 prior 指针指向链表的最后一个单元，这样构成的链表称为双向循环链表（图 1-7 所示）。

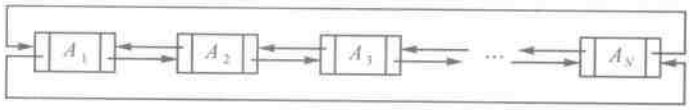


图 1-7　双向循环链表

在双向链表中插入或删除一个结点时，需要注意同时修改该结点的前驱和后继结点的相关指针域值，即：前驱结点的 next 域值和后继结点的 prior 域值。

3. 线性表的应用

用链表表示多项式是线性表应用的一个经典例子。

多项式的一般形式可为：

$$A(x) = a_{m-1}\,x^{e_{m-1}} + \cdots + a_0\,x^{e_0}$$

它由一系列数据项组成，每个数据项包括非零系数 a_i 和非负整型指数 e_i 两个成分，且指数是从大到小排列，即 $e_{m-1} > e_{m-2} > \cdots > e_1 > e_0$。

链表表示多项式要求每个结点要存储多项式中的一项，要包括系数和指数两个数据域以及一个指针域三个数据成员，其结点结构可以表示为：

coef	expon	link

对应的数据结构定义为：

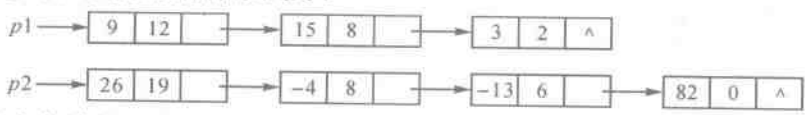
```
typedef struct Poly_node *pPoly;
typedef struct Poly_node {
    int coef;
    int expon;
    pPoly link;
} Poly;
```

例如，有两个多项式 $p1$ 和 $p2$，它们分别为：

$$p1 = 9x^{12} + 15x^8 + 3x^2$$

$$p2 = 26x^{19} - 4x^8 - 13x^6 + 82$$

两个多项式的链表存储形式为：



对链表存放的两个多项式进行加法运算是从 $p1$ 和 $p2$ 所指各自的多项式第一个结点开始比较两个数据项指数的大小，比较结果为以下 3 种情况之一，并做相应的处理。

（1）两数据项指数相等。系数相加，并作为结果多项式对应项的系数，连同这些指数一并存入结果多项式。沿两结点的链域，取两个多项式的下一项，再进行新一轮的比较和处理。

(2)$p1$ 中的数据项指数较大。$p2$ 当前项不变,将 $p1$ 的当前项存入结果多项式,并取 $p1$ 的下一项,再进行新一轮的比较和处理。

(3)$p2$ 中的数据项指数较大。$p1$ 当前项不变,将 $p2$ 的当前项存入结果多项式,并取 $p2$ 的下一项,再进行新一轮的比较和处理。

当某一多项式最后一个结点处理完时,停止上述加合过程,将未处理完的另一个多项式的所有结点依次复制到结果多项式中去。

函数 PolyAdd 实现上述两个多项式加法运算。

```
pPoly  PolyAdd (pPoly p1, pPoly p2)
{
   pPoly front, rear, temp;
   int sum;
   rear = (pPoly) malloc(sizeof(Poly_node));  //申请一个结点作为结果多项式的临时头结点
   if( IS_NULL(rear) )
   {
         fprintf(stderr, "无存储空间\n");
         exit(1);
   }
   front = rear; //front, rear 分别指向结果多项式链表的头和尾
   while( p1 && p2 )
      switch ( COMPARE(p1 -> expon, p2 -> expon) ) {
         case -1:
                  attach(p2 -> coef, p2 -> expon,&rear);
                  p2 = p2 -> link;
                  break;
         case 0: sum = p1 -> coef + p2 -> coef;
                  if( sum ) attach(sum, p1 -> expon, &rear);
                  p1 = p1 -> link;
                  p2 = p2 -> link;
                  break;
         case 1: attach(p1 -> coef, p1 -> expon, &rear);
                  p1 = p1 -> link;
      }
   for( ; p1; p1 = p1 -> link )
      attach(p1 -> coef, p1 -> expon, &rear);
   for( ; p2; p2 = p2 -> link )
      attach(p2 -> coef, p2 -> expon, &rear);
   rear -> link = NULL;
   temp = front;
   front = front -> link;
   free(temp);   //释放临时头结点
   return front;
}
```

attach(coef, expon, &rear)将 coef 和 expon 构成的新的项加入结果多项式的末端;COMPARE(p1 -> expon, p2 -> expon)比较两个指数的大小,根据大、小、等三种情况分别返回1、-1、0。

二、栈、队列和数组

(一)栈和队列的基本概念

1. 栈

栈(Stack)是一种仅允许对表的一端进行插入和删除操作的线性表。一般人们把表的末端称为栈顶,另一端则称为栈底,栈顶的第一个元素被称为栈顶元素。对一个栈最基本的操作包括进栈(Push)和出栈(Pop)。因为栈只能在栈顶进行进栈和出栈的操作,即栈的修改是按照先进后出的原则进行的,所以栈又称为后进先出表(Last In First Out, LIFO)。

不含任何数据元素的栈为空栈,对一个空栈进行出栈操作会产生错误;同样,对一个没有剩余空间的满栈进行进栈操作也会产生错误。

栈的主要操作为:

- 初始化栈 Stack CreatStack(max_stack_size):生成空栈,其最大长度为max_stack_size。
- 栈满判断 Boolean IsFull(stack, max_stack_size):判断栈 stack 是否已满,若 stack 中元素个数等于 max_stack_size 时返回 TRUE;否则返回 FALSE。
- 入栈 void Push(stack, item):若栈已满,输出栈为满信息;否则将数据元素 item 插入到栈 stack 栈顶处。
- 栈空判断 Boolean IsEmpty(stack):判断栈 stack 是否为空,若是返回 TRUE;否则返回 FALSE。
- 出栈 Element Pop(stack):删除并返回栈顶元素,若 IsEmpty(stack)为 TRUE,返回栈为空信息;否则将栈顶数据元素从栈中删除并返回。

2. 队列

队列也是一个有序线性表,但队列的插入和删除操作是分别在线性表的两个不同端点进行的。删除的一端称为队头,插入的一端称为队尾。队列通常又被称为先进先出表(First In First Out, FIFO)。

队列的主要操作为:

- 初始化队列 Queue CreatQueue(max_qsize):生成长度为 max_qsize 的空队列。
- 队列满判断 Boolean IsFullQ(queue, max_qsize):判断队列 queue 是否已满,若是返回 TRUE;否则返回 FALSE 。
- 入队列 Void AddQ(queue, item):若 IsFullQ(queue, max_qsize)为 TRUE,输出队列已满信息;否则将数据元素 item 插入到队列 queue 中去。
- 队列空判断 Boolean IsEmptyQ(queue):判断队列 queue 是否为空,若是返回 TRUE;否则返回 FALSE。
- 出队列 Element DeleteQ(queue):删除并返回队头元素。若 IsEmptyQ(queue)为 TRUE,返回队列为空信息;否则将队头数据元素从队列中删除并返回。

(二)栈和队列的顺序存储结构

1. 栈的顺序存储结构

由于栈是线性表,因而栈的存储结构可采用顺序和链式两种形式。顺序存储的栈称为顺序栈,链式存储的栈称为链栈。

栈的顺序存储结构通常由一个一维数组和一个记录栈顶元素位置的变量组成。假设用一维数组 S[MaxSize](下标 0 ~ MaxSize - 1)表示一个栈,MaxSize 为栈中可存储数据元素的最大个数,即栈的最大长度。习惯上将栈底放在数组下标小的那端,栈顶位置用一个整型变量 top 记录当前栈顶元素的下标值。当 top 指向 -1 时,表示空栈。当 top 指向 MaxSize - 1 时,表示满栈。用 C 语言描述顺序栈类型 Stack 如下:

```
#define MaxSize   <储存数据元素的最大个数>
typedef struct {
    ElemType data[MaxSize];
    int top;
} Stack;
```

在上述栈中,执行 Push 操作时首先判断栈是否满;若不满,则 top 加 1,并将新元素放入 data 数组的 top 分量中。执行 Pop 操作时首先判断栈是否空;若不空,返回 data[top],同时将 top 减 1。

下面举用一个数组实现两个栈的例子。

问题:请用一个数组实现两个栈,要求最大地利用数组空间,使得只要数组有空间,入栈操作就可以成功。写出相应的入栈和出栈操作函数。

可以使这两个栈分别从数组的两头开始向中间生长;当两个栈的栈顶指针相遇时,表示两个栈都满了。

```
#define MaxSize   <储存数据元素的最大个数>
struct DStack {
    ElemType data[MaxSize];
    int top1;
    int top2;
} s;
```

栈初始化时,s.top1 = -1, s.top2 = MaxSize。

```
void Push(struct DStack *sp, ElemType x, int tag)
{   //sp 为栈地址变量,x 为新插入的元素,tag 作为区分两个栈的标志,取值为 1 和 2
    if (sp->top2 - sp->top1 == 1)  return ;   //栈满
    if (tag == 1)      //对第一个栈操作
        sp->data[++top1] = x;
    else               //对第二个栈操作
        sp->data[--top2] = x;
}

ElemType Pop(struct DStack *sp, int tag)
{   //sp 为栈地址变量,tag 作为区分两个栈的标志,取值为 1 和 2
    if (tag == 1) {    //对第一个栈操作
```