全国计算机等级考试名师名导

C 语言程序设计应试辅导（二级）

杨　非　等 编著

清 华 大 学 出 版 社
北　京

内 容 简 介

本书是根据教育部考试中心最新的全国计算机等级考试大纲（2004）编写的。本书共分为 11 章。主要内容有：二级公共基础，C 程序设计的基础知识和简单语句，C 程序中的控制结构，指针和函数的基本概念，数组，字符串，函数的进一步讨论，结构体、共用体和用户定义类型，文件，其他，上机考试。除上机考试一章外，书中其他各章都包含相应的知识点、难点、重点解析，典型试题及解析，自测训练题和答案等内容。

本书将考试、复习内容浓缩和融合在一起，知识精练、重点突出、例题丰富；既可以作为等级考试的应试辅导用书，也可以作为大专院校师生的教学参考书。

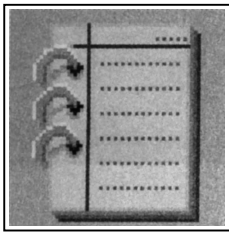
图书在版编目(CIP)数据

C 语言程序设计应试辅导（二级） / 杨非等编著. —北京：清华大学出版社，2005.7
（全国计算机等级考试名师名导 / 谭浩强主编）
ISBN 7-302-11159-6

Ⅰ. C…　Ⅱ. 杨…　Ⅲ. C 语言–程序设计–水平考试–自学参考资料　IV. TP312

中国版本图书馆 CIP 数据核字（2005）第 058942 号

出 版 者：清华大学出版社　　地　　址：北京清华大学学研大厦
　　http://www.tup.com.cn　　邮　　编：100084
　　社 总 机：010-62770175　　客户服务：010-62776969
责任编辑：薛　阳
印 刷 者：
装 订 者：肖　米
发 行 者：新华书店总店北京发行所
开　　本：185×260　印张：27.5　字数：681 千字
版　　次：2005 年 7 月第 1 版　2005 年 7 月第 1 次印刷
书　　号：ISBN 7-302-11159-6 / TP・7375
印　　数：1~5000
定　　价：25.00 元



前　言

1994 年国家教委考试中心推出了面向社会的“全国计算机等级考试”，至今已有 10 多年的历史。由于这一考试从一开始就重视对考生进行计算机相关知识和实践能力的测试，因而得到了社会的广泛认可，10 年来已有上千万人报名参加了这一考试。随着 C 语言的广泛应用，参加 C 语言程序设计考试的人数也逐年增加，2004 年就有 50 万人报名参加该科考试。为了帮助应试者有效地复习“C 语言程序设计”的考试内容，熟悉考试形式，检查自己掌握的程度，我们根据全国计算机等级考试二级 C 语言程序设计的大纲及试题内容，编写了本书。

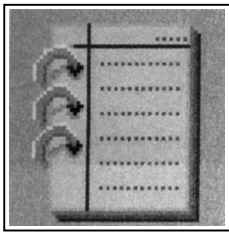
本书的所有例题均取自近年来的试题，按知识点由浅到深进行了归纳，分成各章节，并逐一进行解析，在分析的过程中同时指出应当注意、容易出错的地方。本书的编排既能帮助应试者进行系统的复习，又能帮助应试者了解应知应会的内容。每一章的最后都有大量的自测题，并给出了答案，以便读者检验自己对本部分知识掌握的程度。

2005 年新的二级考试大纲要求考生掌握一些基本的软件基础知识，本书在第 1 章中集中叙述了这一方面的内容。

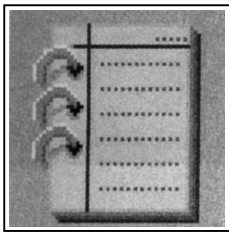
在上机考试时，需要在规定的时间内完成 3 个上机题。按照新的二级考试大纲，把以前第 1 题中的 DOS 操作系统的操作题改为 C 语言程序的填空题，每题有 3 个空。这就要求考生能够读懂程序和所用算法，填入正确内容，以便运行程序后能得出正确结果。第 2 题和第 3 题仍是改错题和编程题。本书的第 11 章对上机题做了分析，给出了几个填空题的样题，以便考生了解和熟悉题型。对于改错题，我们对教育部考试中心编写的上机考试习题集中的改错题进行了分析和归纳，通过例题的形式，对改错题中可能出现的错误进行了分析。对于编程题，我们对该习题集中编程题所涉及的典型算法进行了分析和归纳。在第 11 章中，对有关的编程思路进行了细致的解释，希望有助于读者理解主要算法。上机考试是对考生实际编程能力的一种考核，这种能力不可能用死记硬背的方法去掌握，希望广大读者通过上机练习来培养初步的编程能力。

编者　杨非　周甄　鲍有文

2005 年



目　录



第 1 章

二级公共基础

1.1 本章知识点

二级公共基础的内容主要包括数据结构与算法、程序设计基础、软件工程基础以及数据库设计基础 4 个部分。

1.1.1 基本数据结构与算法

本部分的考试要点。

（1）算法的基本概念，算法复杂度的概念和意义（时间复杂度与空间复杂度）。

（2）数据结构的定义，数据的逻辑结构与存储结构，数据结构的图形表示，线性结构与非线性结构的概念。

（3）线性表的定义，线性表的顺序存储结构及其插入与删除运算。

（4）栈和队列的定义，栈和队列的顺序存储结构及其基本运算。

（5）线性单链表、双向链表与循环链表的结构及其基本运算。

（6）树的基本概念，二叉树的定义及其存储结构，二叉树的前序、中序和后序遍历。

（7）顺序查找与二分查找算法，基本排序算法（交换类排序、选择类排序、插入类排序）。

1．算法的基本概念

（1）算法与数据结构的关系

程序设计主要包括两个方面，一是行为特性的设计，二是结构特性的设计。

行为特性的设计一般是指将解决问题过程中的每一个细节准确地加以定义，并将全部的解题过程用某种工具完整地描述出来。这一过程也称为算法的设计。

结构特性的设计，是指为问题的解决确定合适的数据结构。

数据结构与算法之间有着密切的关系。特别是对于数据处理问题，算法的效率通常与数据结构在计算机中的表示有着直接的关系。

（2）算法的基本特征

所谓算法是指对解题方案准确而完整的描述。

对于一个问题，如果可以通过一个计算机程序，在有限的存储空间内运行有限长的时间而得到正确的结果，则称这个问题是算法可解的。但算法不等于程序，也不等于计算方法。当然，程序也可以作为算法的一种描述，但程序通常还需考虑很多与方法和分析无关的细节问题，这是因为在编写程序时要受到计算机系统运行环境的限制。

算法实际上是一种抽象的解题方法，它具有动态性。作为一个算法，一般应具有以下几个基本特征。

① 能行性（effectiveness）

算法的能行性主要包括两个方面。一是算法中的每一个步骤必须是能实现的，二是算法执行的结果要能达到预期的目的。

② 确定性（definiteness）

算法的确定性是指算法中的每一个步骤都必须是有明确定义的，不允许有模棱两可的解释，也不允许有多义性。这一性质也反映了算法与数学公式的明显差别。

③ 有穷性（finiteness）

算法的有穷性是指算法必须能在有限的时间内做完，即算法必须能在执行有限个步骤之后终止。

算法的有穷性还应包括合理的执行时间的含义。因为，如果一个算法需要执行千万年，也失去了实用价值。

④ 拥有足够的情报

一个算法是否有效，还取决于为算法所提供的情报是否足够。通常，算法中的各种运算总是要施加到各个运算对象上，而这些运算对象又可能具有某种初始状态，这是算法执行的起点或是依据。因此，一个算法执行的结果总是与输入的初始数据有关，不同的输入将会有不同的结果输出。当输入不够或输入错误时，算法本身也就无法执行或执行有错。一般来说，当算法拥有足够的情报时，此算法才是有效的，而当提供的情报不够时，算法并不有效。

综上所述，所谓算法，是一组严谨地定义运算顺序的规则，并且每一个规则都是有效的，且是明确的，此顺序将在有限的次数下终止。

（3）算法的基本要素

一个算法通常由两个基本要素组成：一是对数据对象的运算和操作，二是算法的控制结构。

① 算法中对数据的运算和操作

每个算法实际上是按解题要求从环境能进行的所有操作中选择合适的操作所组成的一组指令序列。因此，计算机算法就是计算机能处理的操作所组成的指令序列。

通常，计算机可以执行的基本操作是以指令的形式描述的。一个计算机系统能执行的所有指令的集合，称为该计算机系统的指令系统。计算机程序就是按解题要求从计算机指令系统中选择合适的指令所组成的指令序列。

② 算法的控制结构

一个算法的功能不仅取决于所选用的操作，而且还与各操作之间的执行顺序有关。算法中各操作之间的执行顺序称为算法的控制结构。

算法的控制结构给出了算法的基本框架，它不仅决定了算法中各操作的执行顺序，而且也直接反映了算法的设计是否符合结构化原则。描述算法的工具通常有传统流程图、N-S 结构化流程图、算法描述语言等。一个算法一般都可以用顺序、选择、循环 3 种基本控制结构组合而成。

（4）算法设计的基本方法

计算机解题的过程实际上是在实施某种算法，这种算法称为计算机算法。

常用的算法设计方法有以下几种。

① 列举法

列举法的基本思想是：根据提出的问题，列举所有可能的情况，并用问题中给定的条件检验哪些是需要的，哪些是不需要的。因此，列举法常用于解决“是否存在”或“有多少种可能”等类型的问题，例如求解不定方程的问题。

列举法的特点是算法比较简单。但当列举的可能情况较多时，执行列举算法的工作量将会很大。通常，在设计列举算法时，要对实际问题进行详细的分析，将与问题有关的知识条理化、完备化、系统化，从中找出规律；或对所有可能的情况进行分类，引出一些有用的信息，就可以大大减少列举量。

列举算法是计算机算法中的一个基础算法。

② 归纳法

归纳法的基本思想是：通过列举少量的特殊情况，经过分析，最后找出一般的关系。显然，归纳法要比列举法更能反映问题的本质，并且可以解决列举量为无限的问题。但是，从一个实际问题中总结归纳出一般的关系，并不是一件容易的事情，尤其是要归纳出一个数学模型更为困难。从本质上讲，归纳就是通过观察一些简单而特殊的情况，最后总结出有用的结论或解决问题的有效途径。

③ 递推

所谓递推，是指从已知的初始条件出发，逐次推出所要求的各中间结果和最后结果。其中初始条件或是问题本身已经给定，或是可以通过对问题的分析与化简而得到确定。递推本质上也属于归纳法，工程上许多递推关系式实际上是通过对实际问题的分析与归纳而得到的，因此，递推关系式往往是归纳的结果。

④ 递归

递归的基本思想是：将一个复杂的问题归结为若干个较简单的问题，然后将每一个较简单的问题再归结为更简单的问题，这个过程可以一直做下去，直到归结为最简单的问题为止。递归是一种很重要的算法设计方法。

递归分为直接递归与间接递归两种。

⑤ 减半递推技术

解决实际问题的复杂程度往往与问题的规模有着密切的关系。因此，利用分治法解决这类实际问题是有效的。所谓分治法，就是对问题分而治之。工程上常用的分治法是减半递推技术。

所谓“减半”，是指将问题的规模减半，而问题的性质不变；所谓“递推”，是指重复“减半”的过程。

⑥ 回溯法

在工程上，有些实际问题很难将其归纳出一组简单的递推公式或直观的求解步骤，并且也不能进行无限的列举。对于这类问题，一种有效的方法是“试”。通过对问题的分析，找出一个解决问题的线索，然后沿着这个线索逐步试探。对于每一步的试探，若成功，就得到问题的解；若失败，就逐步回退，改换别的路线再进行试探。这种方法称为回溯法。回溯法在处理复杂数据结构方面有着广泛的应用。

（5）算法复杂度

算法的复杂度主要包括时间复杂度和空间复杂度。

① 算法的时间复杂度

所谓算法的时间复杂度，是指执行算法所需要的计算工作量。

算法的工作量用算法所执行的基本运算次数来度量，而算法所执行的基本运算次数是问题规模的函数，即

算法的工作量=f(n)

其中 n 是问题的规模。

在同一问题规模下，如果算法执行所需的基本运算次数取决于某一特定输入时，可以用以下两种方法来分析算法的工作量。

- 平均性态（Average Behavior）

所谓平均性态分析，是指用各种特定输入下的基本运算次数的带权平均值来度量算法的工作量。算法的平均性态定义为

$$A(n)=\sum_{x\in D_n} p(x)t(x)$$

其中：x 是所有可能输入中的某个特定输入，p(x)是 x 出现的概率（即输入为 x 的概率），t (x)是算法在输入为 x 时所执行的基本运算次数，D_n 表示当规模为 n 时，算法执行时所有可能输入的集合。

- 最坏情况复杂性（Worst-Case Complexity）

所谓最坏情况分析，是指在规模为 n 时，算法所执行的基本运算的最大次数。它定义为

$$W(n)=\max_{x\in D_n}\{t(x)\}$$

显然，W(n)的计算要比 A(n)的计算方便得多。由于 W(n)实际上是给出了算法工作量的一个上界，因此，它比 A(n)更具有实用价值。

② 算法的空间复杂度

一个算法的空间复杂度，一般是指执行这个算法所需要的内存空间。

一个算法所占用的存储空间包括算法程序所占的空间、输入的初始数据所占的存储空间以及算法执行过程中所需要的额外空间。其中额外空间包括算法程序执行过程中的工作单元以及某种数据结构所需要的附加存储空间。如果额外空间量相对于问题规模来

说是常数，则称该算法是原地(in place)工作的。在许多实际问题中，为了减少算法所占的存储空间，通常采用压缩存储技术，以便尽量减少不必要的额外空间。

2．数据结构的基本概念

数据结构作为计算机的一门学科，主要研究和讨论以下 3 个方面的问题。

- 数据集合中各数据元素之间所固有的逻辑关系，即数据的逻辑结构；
- 在对数据进行处理时，各数据元素在计算机中的存储关系，即数据的存储结构；
- 对各种数据结构进行的运算。

（1）什么是数据结构

简单地说，数据结构是指相互有关联的数据元素的集合。

在数据处理领域中，每一个需要处理的对象都可以抽象成数据元素。数据元素一般简称为元素。

前后件关系是数据元素之间的一个基本关系，但前后件关系所表示的实际意义随具体对象的不同而不同。一般来说，数据元素之间的任何关系都可以用前后件关系来描述。

① 数据的逻辑结构

通俗地说，数据结构是指带有结构的数据元素的集合。在此，所谓结构实际上就是指数据元素之间的前后件关系。

一个数据结构应包含以下两方面的信息：

- 表示数据元素的信息；
- 表示各数据元素之间的前后件关系。

所谓数据的逻辑结构，是指反映数据元素之间逻辑关系的数据结构。

由前面的叙述可以知道，数据的逻辑结构有两个要素：一是数据元素的集合，通常记为 D；二是 D 上的关系，它反映了 D 中各数据元素之间的前后件关系，通常记为 R。即一个数据结构可以表示成

$$B=(D，R)$$

其中 B 表示数据结构。为了反映 D 中各数据元素之间的前后件关系，一般用二元组来表示。例如，假设 a 与 b 是 D 中的两个数据，则二元组（a，b）表示 a 是 b 的前件，b 是 a 的后件。这样，在 D 中的每两个元素之间的关系都可以用这种二元组来表示。

② 数据的存储结构

数据的逻辑结构在计算机存储空间中的存放形式称为数据的存储结构（也称数据的物理结构）。

一个数据结构中的各数据元素在计算机存储空间中的位置关系与逻辑关系有可能是不同的。

由于数据元素在计算机存储空间中的位置关系可能与逻辑关系不同，因此，为了表示存放在计算机存储空间中的各数据元素之间的逻辑关系（即前后件关系），在数据的存储结构中，不仅要存放各数据元素的信息，还需要存放各数据元素之间的前后件关系的信息。

一般来说，一种数据的逻辑结构，根据需要可以表示成多种存储结构，常用的有顺序、链接、索引等存储结构。而采用不同的存储结构，其数据处理的效率是不同的。因

此，在进行数据处理时，选择合适的存储结构是很重要的。

（2）数据结构的图形表示

在数据结构的图形表示中，对于数据集合 D 中的每一个数据元素，用中间标有元素值的方框表示，一般称之为数据结点，并简称为结点；为了进一步表示各数据元素之间的前后件关系，对于关系 R 中的每一个二元组，用一条有向线段从前件结点指向后件结点。

有时在不会引起误会的情况下，在前件结点到后件结点连线上的箭头可以省去。

在数据结构中，没有前件的结点称为根结点；没有后件的结点称为终端结点（也称为叶子结点）。数据结构中除根结点与终端结点外的其他结点一般称为内部结点。

（3）线性结构与非线性结构

根据数据结构中各数据元素之间前后件关系的复杂程度，一般将数据结构分为两大类型：线性结构与非线性结构。

如果一个非空的数据结构同时满足下列两个条件：

（1）有且只有一个根结点；

（2）每一个结点最多有一个前件，也最多有一个后件；

则称该数据结构为线性结构。线性结构又称线性表。

如果一个数据结构不是线性结构，则称之为非线性结构。

线性结构与非线性结构都可以是空的数据结构。一个空的数据结构究竟是属于线性结构还是属于非线性结构，这要根据具体情况来确定。如果对该数据结构的运算是按线性结构的规则来处理的，则属于线性结构；否则属于非线性结构。

3．线性表及其顺序存储结构

（1）线性表的基本概念

线性表是由 $n(n\geq 0)$个数据元素 a_1，a_2，…，a_n 组成的一个有限序列。表中的每一个数据元素，除了第一个外，有且只有一个前件，除了最后一个外，有且只有一个后件。即线性表或是一个空表，或可以表示为

$$(a_1，a_2，\cdots，a_i，\cdots，a_n)$$

其中 $a_i(i=1，2，\cdots，n)$是属于数据对象的元素，通常也称其为线性表中的一个结点。

显然，线性表是一种线性结构。数据元素在线性表中的位置只取决于它们自己的序号，即数据元素之间的相对位置是线性的。

非空线性表有如下一些结构特征：

① 有且只有一个根结点 a_1，它无前件；

② 有且只有一个终端结点 a_n，它无后件；

③ 除根结点与终端结点外，其他所有结点有且只有一个前件，也有且只有一个后件。线性表中结点的个数 n 称为线性表的长度。当 $n=0$ 时，称为空表。

（2）线性表的顺序存储结构

线性表的顺序存储结构具有以下两个基本特点：

① 线性表中所有元素所占的存储空间是连续的；

② 线性表中各数据元素在存储空间中是按逻辑顺序依次存放的。

由此可以看出，在线性表的顺序存储结构中，其前后件两个元素在存储空间中是紧邻的，且前件元素一定存储在后件元素的前面。

（3）顺序表的插入运算

设长度为 n 的线性表为

$$(a_1,a_2,\cdots,a_i,\cdots,a_n)$$

现要在线性表的第 i 个元素 a_i 之前插入一个新元素 b，插入后得到长度为 n+1 的线性表为

$$(a'_1,a'_2,\cdots,a'_j,a'_{j+1},\cdots,a'_n,a'_{n+1})$$

则插入前后的两线性表中的元素满足如下关系：

$$a'_j=\begin{cases}a_j & 1\le j\le i-1\\ b & j=i\\ a_{j-1} & i+1\le j\le n+1\end{cases}$$

在一般情况下，要在第 i(1≤i≤n)个元素之前插入一个新元素，首先要从最后一个(即第 n 个)元素开始，直到第 i 个元素，其间共 n-i+1 个元素依次向后移动一个位置，移动结束后，第 i 个位置就被空出，然后将新元素插入到第 i 项。插入结束后，线性表的长度就增加了 1。

显然，在线性表采用顺序存储结构时，如果插入运算在线性表的末尾进行，即在第 n 个元素之后（可以认为是在第 n+1 个元素之前）插入新元素，则只要在表的末尾增加一个元素即可，不需要移动表中的元素；但如果要在线性表的第 1 个元素之前插入一个新元素，则需要移动表中所有的元素。在一般情况下，如果插入运算在第 i(1≤i≤n)个元素之前进行，则原来第 i 个元素之后（包括第 i 个元素）的所有元素都必须移动。在平均情况下，要在线性表中插入一个新元素，需要移动表中一半的元素。因此，在线性表顺序存储的情况下，要插入一个新元素，其效率是很低的，特别是在线性表比较大的情况下更为突出，由于数据元素的移动而消耗较多的处理时间。

（4）顺序表的删除运算

设长度为 n 的线性表为

$$(a_1,a_2,\cdots,a_i,\cdots,a_n)$$

现要删除第 i 个元素，删除后得到长度为 n-1 的线性表为

$$(a'_1,a'_2,\cdots,a'_j,\cdots,a'_{n-1})$$

则删除前后的两线性表中的元素满足如下关系：

$$a'_j=\begin{cases}a_j & 1\le j\le i-1\\ \\ a_{j+1} & i\le j\le n-1\end{cases}$$

一般情况下，要删除第 i(1≤i≤n)个元素时，则要从第 i+1 个元素开始，直到第 n 个元素，其间共 n-i 个元素依次向前移动一个位置。删除结束后，线性表的长度就减小了 1。

显然，在线性表采用顺序存储结构时，如果删除运算在线性表的末尾进行，即删除第 n 个元素，则不需要移动表中的元素；但如果要删除线性表中的第 1 个元素，则需要移动表中所有的元素。在一般情况下，如果要删除第 i(1≤i≤n)个元素，则原来第 i 个元素之后的所有元素都必须依次往前移动一个位置。在平均情况下，要在线性表中删除一个元素，需要移动表中一半的元素。因此，在线性表顺序存储的情况下，要删除一个元素，其效率也是很低的，特别是在线性表比较大的情况下更为突出，由于数据元素的移动而消耗较多的处理时间。

4．栈和队列

（1）栈及其基本运算

① 栈的基本概念

栈实际上也是线性表，只不过是一种特殊的线性表。在这种特殊的线性表中，其插入与删除运算都只在线性表的一端进行。即在这种线性表的结构中，一端是封闭的，不允许进行插入与删除元素；另一端是开口的，允许插入与删除元素。即栈(stack)是限定在一端进行插入与删除的线性表。

在栈中，允许插入与删除的一端称为栈顶，而不允许插入与删除的另一端称为栈底。栈顶元素总是最后被插入的元素，从而也是最先能被删除的元素；栈底元素总是最先被插入的元素，从而也是最后才能被删除的元素。即栈是按照“先进后出”(First In Last Out，FILO）或“后进先出”(Last In First Out，LIFO）的原则组织数据的，因此，栈也被称为“先进后出”表或“后进先出”表。由此可以看出，栈具有记忆作用。

通常用指针 top 来指示栈顶的位置，用指针 bottom 指向栈底。

往栈中插入一个元素称为入栈运算，从栈中删除一个元素(即删除栈顶元素)称为退栈运算。栈顶指针 top 动态反映了栈中元素的变化情况。

② 栈的顺序存储及其运算

与一般的线性表一样，在程序设计语言中，用一维数组 S(1：m)作为栈的顺序存储空间，其中 m 为栈的最大容量。通常，栈底指针指向栈空间的低地址一端（即数组的起始地址这一端）。

在栈的顺序存储空间 S(1：m)中，S(bottom)通常为栈底元素（在栈非空的情况下），S(top)为栈顶元素。top=0 表示栈空；top=m 表示栈满。

栈的基本运算有 3 种：入栈、退栈与读栈顶元素。

- **入栈运算**

入栈运算是指在栈顶位置插入一个新元素。这个运算有两个基本操作：首先将栈顶指针进一（即 top 加 1），然后将新元素插入到栈顶指针指向的位置。

当栈顶指针已经指向存储空间的最后一个位置时，说明栈空间已满，不可能再进行入栈操作。这种情况称为栈“上溢”错误。

- **退栈运算**

退栈运算是指取出栈顶元素并赋给一个指定的变量。这个运算有两个基本操作：首先将栈顶元素（栈顶指针指向的元素）赋给一个指定的变量，然后将栈顶指针退一（即 top 减 1）。

当栈顶指针为 0 时，说明栈空，不可能进行退栈操作。这种情况称为栈“下溢”错误。

- 读栈顶元素

读栈顶元素是指将栈顶元素赋给一个指定的变量。必须注意，这个运算不删除栈顶元素，只是将它的值赋给一个变量，因此，在这个运算中，栈顶指针不会改变。

当栈顶指针为 0 时，说明栈空，读不到栈顶元素。

（2）队列及其基本运算

① 队列的基本概念

队列（queue）是指允许在一端进行插入、而在另一端进行删除的线性表。允许插入的一端称为队尾，通常用一个称为尾指针（rear）的指针指向队尾元素，即尾指针总是指向最后被插入的元素；允许删除的一端称为排头(也称为队头)，通常也用一个排头指针（front）指向排头元素的前一个位置。显然，在队列这种数据结构中，最先插入的元素将最先能够被删除，反之，最后插入的元素将最后才能被删除。因此，队列又称为“先进先出”（FIFO）或“后进后出”（LILO）的线性表，它体现了“先来先服务”的原则。在队列中，队尾指针 rear 与排头指针 front 共同反映了队列中元素动态变化的情况。

往队列的队尾插入一个元素称为入队运算，从队列的排头删除一个元素称为退队运算。

② 循环队列及其运算

所谓循环队列，就是将队列存储空间的最后一个位置绕到第一个位置，形成逻辑上的环状空间，供队列循环使用。在循环队列结构中，当存储空间的最后一个位置已被使用而再要进行入队运算时，只要存储空间的第一个位置空闲，便可将元素加入到第一个位置，即将存储空间的第一个位置作为队尾。

在循环队列中，用队尾指针 rear 指向队列中的队尾元素，用排头指针 front 指向排头元素的前一个位置，因此，从排头指针 front 指向的后一个位置直到队尾指针 rear 指向的位置之间所有的元素均为队列中的元素。

循环队列的初始状态为空，即 rear=front=m。

循环队列主要有两种基本运算：入队运算与退队运算。

每进行一次入队运算，队尾指针就进一。当队尾指针 rear=m+1 时，则置 rear=1。

每进行一次退队运算，排头指针就进一。当排头指针 front=m+1 时，则置 front=1。

在实际使用循环队列时，为了能区分队列满还是队列空，通常还需增加一个标志 s，s 值的定义如下：

$$s=\begin{cases}0 & \text{表示队列空}\\ 1 & \text{表示队列非空}\end{cases}$$

由此可以得出队列空与队列满的条件如下：

队列空的条件为 s=0；

队列满的条件为（s=1）且（front=rear）。

循环队列入队与退队的算法如下。