高等学校计算机基础教育规划教材

程序设计基础
——从问题到程序

胡　明　王红梅　编著

清华大学出版社

* **把握初学者的知识基础。**

本书将程序设计所需的计算机软硬件基本知识融合进来，解决了初学者的知识衔接问题。

* **遵循初学者的认知规律。**

提炼程序设计语言的基本内容，科学安排知识单元之间的拓扑关系，对于有一定难度的主题采用增量式递进，先讲授基本内容，再讲授高级内容。

* **站在内存的角度理解程序。**

强调内存对于程序设计的重要性，通过图示阐述内存在程序执行过程中的动态变化过程，降低理解程序的抽象程度。

* **强调程序设计的一般过程。**

以程序设计过程为主线，采用“问题　想法　算法　程序”的问题求解过程，正确处理算法和语言的关系，培养学生的计算思维能力，提高学生程序设计语言的应用能力。

* **程序实例体现学以致用。**

程序设计实例（包括例题）不是单纯地为了解释语言概念，而是以任务驱动的方式，带领学生分析问题、构造算法、应用程序设计语言解决实际问题，在潜移默化中学会程序设计，提高用计算机解决实际问题的能力。

ISBN 978-7-302-23915-4
9 787302 239154 >
定价：29.50元

高等学校计算机基础教育规划教材

程序设计基础
——从问题到程序

胡　明　王红梅　编著

清华大学出版社
北京

内容简介

本书以 C/C++ 程序设计语言为工具，以程序设计过程为主线，通过“问题→想法→算法→程序”的问题求解过程，提高学生的程序设计能力和计算思维能力。本书绝大部分程序设计实例（包括例题）不是单纯地为了解释语言概念，而是以任务驱动的方式，带领学生分析问题、构造算法、应用程序设计语言解决实际问题，使学生在潜移默化中学会程序设计。

本书适用于程序设计的初学者，主要面向没有任何编程知识和编程经历的读者，遵循认知规律，科学安排知识单元之间的拓扑关系，概念清晰，实例丰富，深入浅出，是程序设计初学者的理想教材。

图书在版编目（CIP）数据

程序设计基础——从问题到程序/胡明，王红梅编著. --北京：清华大学出版社，2011.1
（高等学校计算机基础教育规划教材）
ISBN 978-7-302-23915-4

Ⅰ. ①程… Ⅱ. ①胡… ②王… Ⅲ. ①C 语言－程序设计－高等学校－教材 Ⅳ. ①TP312

中国版本图书馆 CIP 数据核字(2010)第 188770 号

责任编辑：袁勤勇
责任校对：白　蕾
责任印制：李红英

出版发行：清华大学出版社　　**地　　址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　　编**：100084
社　总　机：010-62770175　　**邮　购**：010-62786544
投稿与读者服务：010-62795954，jsjjc@tup.tsinghua.edu.cn
质　量　反　馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：北京四季青印刷厂
装 订 者：三河市兴旺装订有限公司
经　　销：全国新华书店
开　　本：185×260　　**印　张**：21.25　　**字　数**：498 千字
版　　次：2011 年 1 月第 1 版　　**印　次**：2011 年 1 月第 1 次印刷
印　　数：1～3000
定　　价：29.50 元

产品编号：037728-01

《高等学校计算机基础教育规划教材》

编 委 会

前　言

随着信息时代的到来，计算机技术的普及与应用已成为提高综合国力的需要，每个受过高等教育的人应该成为提高综合国力的主力军。对于非计算机专业的学生，仅仅能够进行文字编辑、网页制作等技能性操作是远远不够的，还应该具有程序设计的初步能力，因为通过编写程序，可以进一步了解计算机的工作原理，掌握用计算机解决实际问题的思想和方法，才能更好地理解和应用计算机；对于计算机专业及相关专业的大学生，程序设计能力是应具备的基本能力，是衡量其是否合格的基本标准，程序设计训练还可以提高学生分析问题、解决问题的能力，以及计算思维（逻辑思维、抽象思维）的能力。因此，学习程序设计非常重要。对于程序设计的初学者，用一种具体的程序设计语言作为工具来学习程序设计是比较通用的方法。那么，众多的程序设计语言从哪里开始学习？如何学习程序设计？

实际上，所有程序设计语言的最终目的都是一样的，就是控制计算机按照人们的意愿去工作。共同的目的使各种各样的程序设计语言具有共同的基本内容，无论哪一种程序设计语言，都是以数据的表示和组织、数据处理、程序的流程控制、数据传递为基本内容，只是不同的程序设计语言采用不同的方法实现上述基本内容，表现为不同的程序设计语言具有不同的表述格式（即语法规则）。

程序设计的过程就是利用计算机求解问题的过程，这个过程最终需要借助程序设计语言来表示解决方案，因此，学习程序设计语言的最终目的是能够表示问题的解决方案。但是，掌握了程序设计语言的语法规则并不意味着能够编写程序解决实际问题，因为计算机不能分析问题并产生问题的解决方案，程序设计者必须分析问题，确定问题的解决方案，然后再用程序设计语言表示这个解决方案。因此，学习程序设计的重点和核心不是程序设计语言本身，而是掌握程序设计的基本思想、基本方法和一般过程，只要理解了程序设计的基本思想和一般过程，掌握了程序设计语言的基本规则，就能够触类旁通、举一反三，同时为学习新语言奠定知识基础。

本书以 C/C++程序设计语言为工具，以程序设计过程为主线，通过问题的求解过程提高学生的程序设计能力和计算思维能力。

本书的读者定位

本书的读者定位是程序设计的初学者，主要面向没有任何编程知识和编程经历的读者，包括：

- 计算机专业的学生；

- 非计算机专业的学生；
- 希望更好地理解和应用计算机的非计算机行业的读者。

本书的教材设计

① 把握知识基础。程序设计需要具备一定的计算机软硬件基本知识，本书将这些基本知识（二进制、内存、整数的补码表示、实数的浮点表示、编译的基本过程、算法及其描述方法等）融合起来，解决知识的衔接问题。

② 提炼程序设计语言的基本内容。本书的重点在于讲授程序设计的基本思想和一般过程，对于 C/C++语言采取有所取、有所不取的策略，主要讲授数据的表示和组织、数据处理、程序的流程控制、数据传递（函数）等基本内容，不深究 C/C++语言的语法细节和语言特性，学生在学会了编写程序后，这些细枝末节的问题完全可以自己解决。

③ 遵循认知规律。C/C++语言的语法细节很多，语法细节之间的联系很多，这种结构不良性导致学习不能完全按照线性的顺序。本书强调循序渐进，科学安排知识单元之间的拓扑关系，如图 1 所示。对于有一定难度的主题（函数、指针、文件）采用增量式的方式，先讲授基本内容，再讲授高级内容。

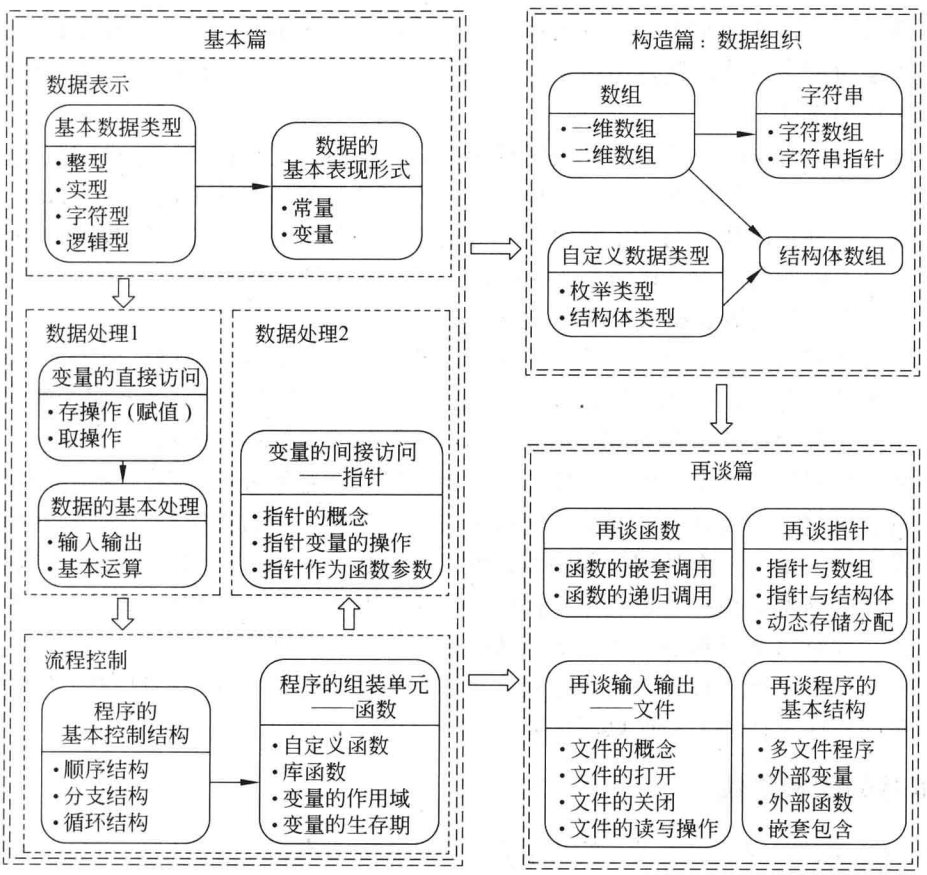


图 1　知识单元的拓扑结构

④ 强调程序设计的一般过程。本书以程序设计过程为主线，采用“问题→想法→算法→程序”的问题求解过程，正确处理算法和语言的关系，通过“问题→想法→算法”的过程（即算法设计）提高学生分析问题、解决问题的能力，达到培养学生计算思维的教学目标，通过“算法→程序”的过程（即编写程序）提高学生程序设计语言的应用能力。

⑤ 站在计算机内存的角度理解程序。强调内存对于程序设计的重要性，通过图示阐述内存（变量）在程序执行过程中的动态变化过程，降低学习程序的抽象程度，使学生不仅知其然，还能知其所以然，然后才能使其然，从而很好地驾驭程序设计语言解决实际问题。

⑥ 程序设计实例体现学以致用。本书绝大部分程序设计实例（包括例题）不是单纯地为了解释语言概念，而是以任务驱动的方式，带领学生分析问题、构造算法、应用程序设计语言解决实际问题，在潜移默化中学会程序设计，提高学生用计算机解决实际问题的能力。

⑦ 注重程序风格，扎实编程基本功。本书强调程序的规范性和可读性，如程序中适当加空格和空行，关键语句后要有注释，变量名、函数名等标识符的规范命名，将逻辑上相对独立的功能组织为函数，而不是将所有代码都写在 main 函数中，使学生在开始写程序时就养成良好的编程习惯。

本书的教学资源

本书有配套的教学网站：http://jsj.ccut.edu.cn/cxsj，网站上有教学大纲、教学日历、电子课件、电子教案、书中所有的程序代码，教学资源将实时补充和更新。

需要强调的是，熟记程序设计语言的语法规则并不是掌握程序设计的捷径，只有通过大量的练习才能掌握程序设计的基本思想和一般过程，才能真正做到学以致用，学生只有亲自动手编写大量代码之后，才能获得真知灼见。本书所有代码均在 Visual C++ 6.0 编程环境下调试通过，2007 级肖丰奇同学和 2008 级胡洁珺同学参与了本书代码的调试工作，参加本书编写的还有王涛、党源源、谷钰、逄焕利、刘钢、陈志雨、姚庆安、孙旸等老师。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，衷心希望能够得到同行专家和读者的批评和指正。作者的电子邮箱是：

huming@mail.ccut.edu.cn
wanghm@mail.ccut.edu.cn

作　者

2010 年 10 月

目　录

第1章

绪　论

程序设计是计算机专业重要的核心课程，程序设计能力是每一个计算机专业的学生应具备的基本能力，因此，学习程序设计非常重要。同时，程序设计又是一门综合学科，程序设计能力依赖于对具体的程序设计语言和开发环境的掌握程度，依赖于数学、统计学、数据结构、编译原理、操作系统、算法设计与分析等知识的积累，依赖于逻辑思维和抽象思维等计算思维方式的运用，因此，如何学习程序设计是一个难题。对于程序设计的初学者，用一种具体的程序设计语言作为工具来学习程序设计是比较通用的方法。那么，众多的程序设计语言从哪里开始学习？如何学习程序设计？

实际上，所有程序设计语言的最终目的都是一样的，就是控制计算机按照人们的意愿去工作。共同的目的使各种各样的程序设计语言有了共同的知识基础，无论选择哪种程序设计语言，无论编写什么程序，都会运用这些知识基础。只要理解了程序设计的基本概念，掌握了程序设计语言的基本规则，就能够触类旁通、举一反三。本书选择 C/C++ 程序设计语言作为工具来讲授程序设计的基本概念和一般方法，由于篇幅所限，本书仅讨论结构化程序设计。事实上，如果不涉及面向对象的部分，C 语言和 C++语言 90%以上的语法是相同的。本书提到的 C/C++语言特性，以目前流行的 32 位计算机及其操作系统为准。

需要强调的是，对于程序设计的初学者，学习程序设计的目的不仅仅是学习某种具体的程序设计语言，也不仅仅是掌握程序设计语言的语法规则，更重要的是学习程序设计的基本概念和一般方法，并能够应用程序设计语言解决实际问题。

1.1　问题求解与程序设计

计算机科学把问题作为研究对象，研究如何用计算机来解决人类所面临的各种问题。只有最终在计算机上能够运行良好的程序才能为人们解决特定的实际问题，因此，程序设计的过程就是利用计算机求解问题的过程。

1.1.1 程序、程序设计与程序设计语言

计算机是一个大容量、可以高速运转，但是没有思维的机器，计算机看起来聪明是因为它能精确、快速地执行算术运算和逻辑运算。一条鸿沟把需要解决问题的人和没有思维的计算机分离开来，如何跨越这条鸿沟呢？程序是跨越这条鸿沟的桥梁（如图 1.1 所示），用计算机解决某一个特定的问题，必须事先编写程序，告诉计算机需要做哪些事，按什么步骤去做，并提供所要处理的原始数据。

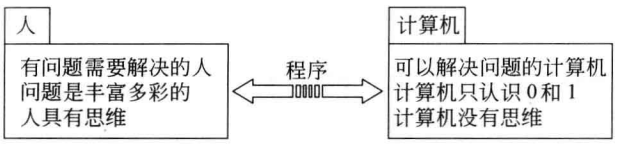


图 1.1　人和计算机通过程序进行沟通

程序是能够实现特定功能的指令序列的集合，这些指令序列描述了计算机求解某一问题的工作步骤。《维基百科全书》对程序设计给出了比较全面的定义：“**程序设计**是给出解决特定问题的程序的过程，是软件构造活动中的重要组成部分。程序设计往往以某种程序设计语言为工具，给出这种语言下的程序。程序设计过程应当包括分析、设计、编码、测试、排错等不同阶段。专业的程序设计人员称为**程序员**。”

从程序设计的定义可以看出，程序设计不仅包括编写程序，而是从要解决的问题开始，进行问题分析，设计好解决方案后，才能够编写出正确的程序，最后还要在计算机上运行程序才能获得问题的解。因此，程序设计的过程就是利用计算机求解问题的过程，程序设计的最终目的就是用程序来控制计算机为人们求解实际问题。

程序设计是一个过程，最终需要借助程序设计语言来表示解决方案，只有在计算机上能够运行良好的程序才能为人们解决特定的实际问题。**程序设计语言**是为了方便描述计算过程而人为设计的符号语言，是人与计算机之间进行信息交流的语言工具。

1.1.2 程序设计的一般过程

用计算机求解任何问题都离不开程序，但是计算机不能分析问题并产生问题的解决方案。人必须分析这些问题，确定问题的解决方案，采用计算机能够理解的指令描述这个问题的求解步骤（即编写程序），然后让计算机执行程序最终获得问题的解。一般来说，对不同求解方法的抽象描述产生了相应的不同算法，而不同的算法将设计出相应的不同程序，因此，程序的获得是由想法到算法，再由算法到程序的过程。程序设计的一般过程如图 1.2 所示。

由问题到想法需要分析问题，抽象出具体的数据模型，形成问题求解的基本思路。对于待求解的问题，首先搞清楚求解的目标是什么，给出了哪些已知信息、显式条件或隐含条件，应该用什么形式的数据表达计算结果。如果没有全面、准确和认真地分析问

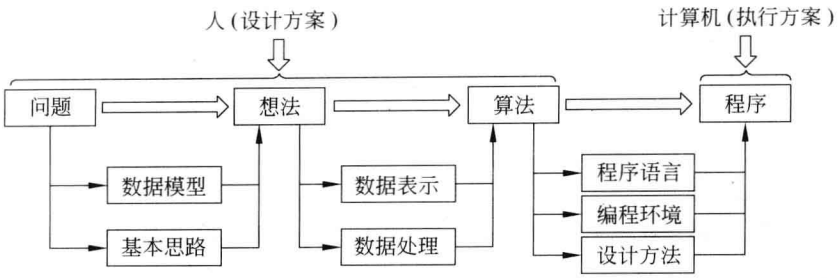


图 1.2　程序设计的一般过程

题，就急急忙忙地编写程序，结果往往是事倍功半，造成不必要的反复，甚至给程序留下严重隐患。

算法用来描述问题的解决方案，是具体的、机械的操作步骤。利用计算机解决问题的最重要一步是将人的想法描述成算法，即设计算法，也就是从计算机的角度设想计算机是如何一步一步完成这个任务的。有些问题很简单，很容易就可以得到问题的解决方案，如果问题比较复杂，就需要更多的思考才能得到问题的解决方案。由想法到算法需要完成数据表示和数据处理，即描述问题的数据模型（待处理的数据以及数据之间的关系，将数据从机外表示转换为机内表示），描述问题求解的基本思路（具体的操作步骤，将问题的解决方案形成算法）。算法是程序设计的基础和精髓，对于计算机专业的学生，学会读懂算法、设计算法，应该是一项最基本的要求，而发明（发现）算法则是计算机学者的最高境界。

由算法到程序需要将算法的操作步骤转换为某种程序设计语言对应的语句，转换所依据的规则就是某种程序设计语言的语法，换言之，就是用某种程序设计语言描述要处理的数据以及数据处理的过程。程序是告诉计算机“做什么”以及“如何做”的指令集合，即把处理问题的步骤以计算机可以识别和执行的语句表示出来。

下面以著名的哥尼斯堡七桥问题（以下简称七桥问题）为例，说明程序设计的一般过程。

【问题】　17 世纪的东普鲁士有一座哥尼斯堡城（现在叫加里宁格勒，在波罗的海南岸），城中有一座岛，普雷格尔河的两条支流环绕其旁，并将整个城市分成北区、东区、南区和岛区 4 个区域，全城共有 7 座桥将 4 个城区连接起来，如图 1.3 所示。于是，产生了一个有趣的问题：一个人是否能在一次步行中经过全部的 7 座桥后回到起点，且每座桥只经过一次。

【想法】　将城区抽象为结点，用 A、B、C、D 表示 4 个城区，将桥抽象为边，用 7 条边表示七座桥，则将七桥问题抽象为一个图模型，如图 1.4 所示。从而将七桥问题抽象为一个数学问题：求经过图中每条边一次且仅一次的回路，后来人们称之为欧拉回路。欧拉回路的判定规则是：

（1）如果通奇数桥的城区多于两个，则不存在欧拉回路；

（2）如果只有两个城区通奇数桥，则可以从这两个城区之一出发找到欧拉回路；

（3）如果没有一个城区通奇数桥，则无论从哪里出发都能找到欧拉回路。

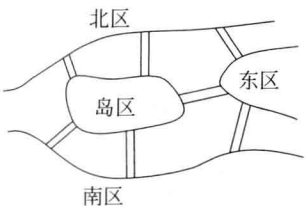


图 1.3　七桥问题示意图

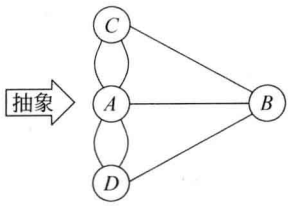


图 1.4　七桥问题的模型

由上述判定规则得到求解七桥问题的基本思路：依次计算图中与每个结点相关联的边的个数（称为结点的度），根据度为奇数的结点个数判定是否存在欧拉回路。

【算法】　求解欧拉回路的算法请参见 8.3.2 节。

【程序】　求解欧拉回路的程序请参见 8.3.2 节。

1.2　算法及其描述方法

程序设计的关键是解决问题的方法和步骤，因此程序设计的核心是算法。算法是问题的解决方案，这个解决方案本身并不是问题的答案，而是能获得答案的指令序列。不言而喻，由于实际问题千奇百怪，问题求解的方法千变万化，所以，算法的设计过程是一个灵活的充满智慧的过程，是一个非常有创造性和值得付出的过程。

1.2.1　算法及其特性

通俗地讲，算法是解决问题的方法，现实生活中关于算法的实例不胜枚举，如菜谱、安装转椅的操作指南等。严格地说，**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。此外，算法还必须满足下列 5 个重要特性，如图 1.5 所示。

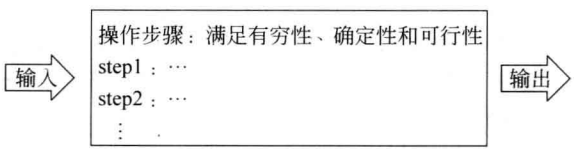


图 1.5　算法及其特性

① 输入：一个算法有零个或多个输入（即算法可以没有输入），这些输入通常取自于某个特定的对象集合。

② 输出：一个算法有一个或多个输出（即算法必须要有输出），通常输出与输入之间有着某种特定的关系。

③ 有穷性：一个算法必须总是在执行有穷步之后结束（对任何合法的输入），且每一步都在有穷时间内完成。

④ 确定性：算法中的每一条指令必须有确切的含义，不存在二义性。并且，在任何条件下，对于相同的输入只能得到相同的输出。

⑤ 可行性：算法描述的操作可以通过已经实现的基本操作执行有限次来实现。

1.2.2 算法的描述方法

算法设计者在构思和设计了一个算法之后，必须清楚准确地将所设计的求解步骤记录下来，即**描述算法**。常用的描述算法的方法有自然语言、程序流程图和伪代码等。下面以欧几里得算法为例介绍描述算法的方法。

【问题】 求两个自然数的最大公约数。

【想法】 设两个自然数是 m 和 n 并满足 $m \geqslant n$，欧几里得算法的基本思想是将 m 和 n 辗转相除直到余数为 0。例如，$m = 35$，$n = 25$，m 除以 n 的余数用 r 表示，计算过程如下：

被除数 m	除数 n	余数 r
35	25	10
25	10	5
10	5	0

当余数 r 为 0 时，除数 n 就是 m 和 n 的最大公约数。

1. 自然语言

用自然语言描述算法，其优点是容易理解，缺点是容易出现二义性，并且算法通常都很冗长。欧几里得算法用自然语言描述如下：

步骤 1：将 m 除以 n 得到余数 r。

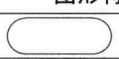
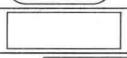
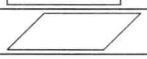
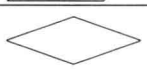
步骤 2：若 r 等于 0，则 n 为最大公约数，算法结束；否则执行步骤 3。

步骤 3：将 n 的值放在 m 中，将 r 的值放在 n 中，重新执行步骤 1。

2. 程序流程图

程序流程图又称程序框图，是一种传统的描述算法的方法，其主要优点是对控制流程的描述很直观，便于初学者掌握。表 1.1 给出了程序流程图的基本符号。

表 1.1 程序流程图的基本符号

图形符号	名　称	含　义
	起止框	开始或结束
	处理框	处理或运算
	输入/输出框	输入/输出
	判断框	根据给定的条件是否满足决定执行两条路径中的某一条路径
	控制流	执行的路径，箭头代表方向

欧几里得算法用程序流程图描述，如图 1.6 所示。

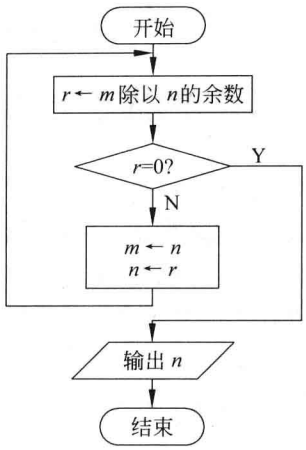


图 1.6　程序流程图描述欧几里得算法

3. 伪代码

伪代码是介于自然语言和程序设计语言之间的描述方法，它保留了程序设计语言严谨的结构、语句的形式和控制成分，处理和条件允许使用自然语言来表达，至于算法中自然语言的成分有多少，取决于算法的抽象级别。由于伪代码书写方便、格式紧凑、容易理解和修改，因此被称为“算法语言”。本书的伪代码采用 C/C++语言的基本语法（也称类 C/C++语言）。欧几里得算法用伪代码描述如下：

伪代码
```
step1: r ← m % n;
step2: 当 r≠0 时，重复执行下述操作：
    step2.1: m ← n;
    step2.2: n ← r;
    step2.3: r ← m % n;
step3: 输出 n;
```

例 1.1　用伪代码描述求解下列问题的算法：

（1）两个瓶子 A 和 B 分别盛放酱油和醋，要求将 A 瓶和 B 瓶的液体互换，即 A 瓶盛放醋，B 瓶盛放酱油。

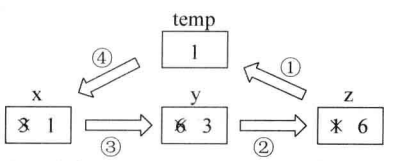
（2）将三个数由小到大排序。

（3）在一个含有 n 个元素的集合中查找最大值元素。

解：（1）取一个空瓶 C，用来暂存某种液体，算法如下：

伪代码
```
step1: 将 A 瓶的酱油倒入 C 瓶，即 C ← A;
step2: 将 B 瓶的醋倒入 A 瓶，即 A ← B;
step3: 将 C 瓶暂存的酱油倒入 B 瓶，即 B ← C;
```

（2）设三个数分别是 x、y 和 z，先将 x 和 y 进行比较，如果 $x>y$，则将 x 和 y 交换，将两个变量进行交换可以采用问题（1）的算法；再将 z 和 y、x 进行比较，有以下三种情况：



注：数字上面打“×”，表示被取代。

图 1.7　$z<x$ 时执行的操作

① 如果 $z\geqslant y$，此时变量 x、y 和 z 即为从小到大排列；

② 如果 $x\leqslant z<y$，则将 y 和 z 交换；

③ 如果 $z<x$，则从小到大依次为 z、x、y，可以将 z 暂存到一个临时变量 temp 中，将 y 的值传给 z，将 x 的值传给 y，将 temp 中暂存的值传给 x，如图 1.7

所示。

以上三种情况可以先判断情况③，如果不满足，则 z 一定大于等于 x，再判断情况②，如果不满足，则一定是情况①，不用执行任何操作，算法如下：

伪代码

```
step1: 如果 x>y,则将 x 和 y 交换;
step2: 如果 z<x,则 temp←z;z ← y;y ← x;x ← temp;
       否则，如果 z<y，则将 y 和 z 交换;
step3: 依次输出 x, y, z;
```

（3）设最大值为 max，可以假定第 1 个元素为最大值元素，依次将第 2，3，…，n 个元素与 max 比较，max 中保存的始终是每次比较后的最大值元素，算法如下：

伪代码

```
step1: max ← 第 1 个元素;
step2: 初始化被比较元素的序号 i ← 2;
step3: 当 i 小于等于 n 时重复执行下述操作:
       step3.1: 如果第 i 个元素大于 max,则 max ← 第 i 个元素;
       step3.2: i ← i + 1;
step4: 输出 max;
```

1.3 程序设计语言

计算机不能理解和执行人类的自然语言，人要和计算机进行交流就必须使用计算机能够识别的语言，因此，需要一种能够准确表达问题的求解步骤，同时还能够被计算机接受的表达方法，这就是程序设计语言。

1.3.1 程序设计语言的发展

程序设计语言的发展是一个不断演化的过程，其根本的推动力是对抽象机制的更高要求，以及对程序设计思想的更好支持。换言之，就是把机器能够理解的语言提升到能够很好地模仿人类思考问题的形式，能够很好地表示人类的思维。

1. 第一代程序设计语言（First Generation Language，1GL）——机器语言

在程序设计语言发展史上首先出现的是机器语言，世界上第一台可执行程序计算机诞生后便产生了机器语言。机器语言是内置在计算机电路中的指令，由 0 和 1 组成，如图 1.8 所示。每种计算机都规定了由 0 和 1 组成的若干条指令。例如，某种计算机规定用 00000100 表示加法指令，遇到这样的二进制位串就执行一次加法操作。这种计算机能直接识别和执行的二进制指令称为**机器指令**。计算机能够执行的全部指令集合构成**计算机指令系统**。不同型号的计算机有不同的指令系统，从而形成了不同型号计算机的特点

和相互间的差别。

机器语言是面向机器的语言，用机器语言编写程序相当烦琐，程序生产率很低，质量难以保证，并且程序不能通用。

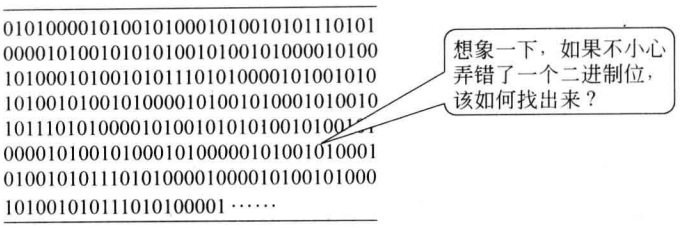


图 1.8　机器语言程序示例

2. 第二代程序设计语言（Second Generation Language，2GL）——汇编语言

用机器语言进行程序设计不仅枯燥费时，而且容易出错。20 世纪 50 年代初出现了汇编语言，它使用助记符表示每条机器指令，例如 MOV 表示传送数据，ADD 表示加，SHL 表示将数据左移，还可以使用十进制数或十六进制数，如图 1.9 所示。显然，汇编语言程序与硬件密切相关，因此程序也不能通用。

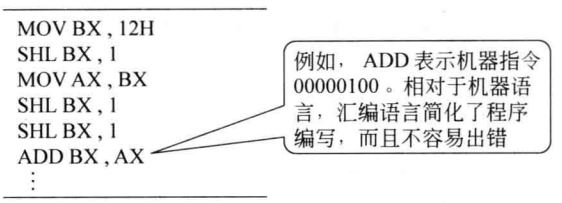


图 1.9　汇编语言程序示例

由于程序最终在计算机上执行时采用的都是机器指令，因此，需要用一种翻译程序把汇编程序翻译成等价的机器指令，如图 1.10 所示。

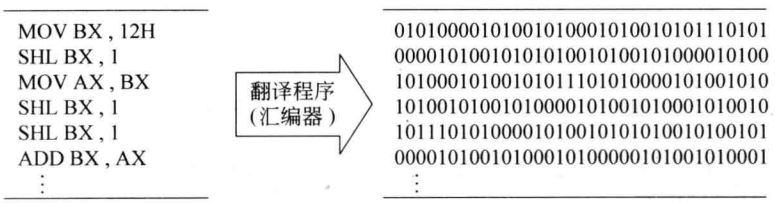


图 1.10　翻译程序将汇编程序转换成对应的机器指令

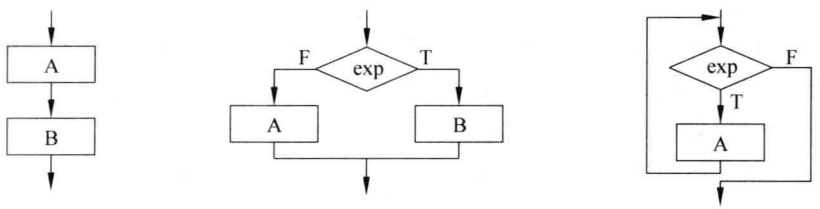
3. 第三代程序设计语言（Third Generation Language，3GL）——高级语言

高级语言的指令形式类似于自然语言和数学语言，不仅容易学习，方便编程，也提高了程序的可读性。每种高级语言都有相应的翻译程序，把高级语言程序翻译成等价的机器指令。高级语言程序不依赖于计算机硬件，通常又称为面向过程（即描述相应的计

算过程）的语言。

20 世纪 50 年代中期出现了第一个高级程序设计语言（简称**高级语言**，相应地，机器语言和汇编语言称为**低级语言**，低级意味着程序员要从机器的层面上考虑问题）——FORTRAN 语言，后来又相继出现了 COBOL、ALGOL、BASIC 等高级语言。目前，高级语言已形成一个庞大的家族，包括结构化程序设计语言、面向对象程序设计语言、可视化程序设计语言、网络程序设计语言等。

1968 年，荷兰计算机科学家迪杰斯特拉（Edsgar W.Dijkstra）发表了论文《GOTO 语句的害处》，指出调试和修改程序的难度与程序中包含 GOTO 语句的数量成正比。从此，各种结构化程序设计理念逐渐确立起来。如果一个程序的模块仅通过顺序、选择和循环这三种基本控制结构进行连接，如图 1.11 所示，并且每个模块只有一个入口和一个出口，则称这个程序是结构化的。Pascal 语言和 MODULA-1 语言都是采用结构化程序设计规则制定的，BASIC 语言也被升级为具有结构化的版本，此外，还出现了灵活且功能强大的 C 语言。



(a) 顺序结构　　(b) 选择结构　　(c) 循环结构

图 1.11　三种基本的控制结构

面向对象的程序设计最早是在 20 世纪 70 年代提出的，其出发点和基本原则是尽可能地模拟现实世界中人类的思维进程，使程序设计的方法和过程尽可能地接近人类解决现实问题的方法和过程。随着面向对象程序设计方法和工具的成熟，从 20 世纪 90 年代开始，面向对象程序设计逐渐成为最流行的程序设计技术，Java、C++、C#等都是面向对象程序设计语言。

可视化程序设计是在面向对象程序设计的基础上发展起来的，可视化程序设计语言把图形用户界面设计的复杂性封装起来，编程人员只需用系统提供的工具在屏幕上画出各种图形对象，并设置这些图形对象的属性，系统就会自动产生界面代码，从而大大提高程序设计的效率。Visual Basic、Visual C++等都是可视化程序设计语言。

随着计算机网络的发展，程序设计语言又呈现出网络化的发展趋势。网络程序设计是在网络环境下进行程序设计，包括服务器端程序设计和客户端程序设计，常用的服务器端程序设计语言有 JSP、ASP 和 PHP 等，常用的客户端程序设计语言有 JavaScript 和 VBScript 等。

4. 第四代程序设计语言（Forth Generation Language，4GL）——非过程式语言

20 世纪 70 年代末到 80 年代初，随着数据库技术和微型计算机的发展，出现了面向

问题的非过程式程序设计语言。与前三代程序设计语言相比，4GL 上升到一个更高的抽象层次，利用 4GL 开发软件只需要考虑“做什么”而不必考虑“如何做”，不涉及太多的算法细节，从而大大提高软件生产率。许多 4GL 为了提高对问题的表达能力和语言的效率，引入了过程化的语言成分。迄今为止，使用最广泛的 4GL 是数据库查询语言，许多数据库语言如 Oracle、Sybase、Informix 等都包含有 4GL 成分。

5. 第五代程序设计语言（Fifth Generation Language，5GL）——知识型语言

由于第三代程序设计语言的发展一直受到冯·诺依曼概念的制约，存在许多局限性。进入 20 世纪 80 年代后，摆脱冯·诺依曼概念的束缚已成为众多计算机语言学家为之奋斗的目标。5GL 力求摆脱传统语言那种状态转换语义的模式，以适应现代计算机系统知识化、智能化的发展趋势。目前，5GL 主要应用在人工智能研究上，典型代表是 LISP 语言和 PROLOG 语言。PROLOG 语言属于逻辑型语言，以形式逻辑和谓词演算为基础，LISP 语言属于函数型语言，以 λ 演算为基础。

目前，4GL 和 5GL 的发展都不是很成熟，在效率、应用等方面都存在诸多问题，常用的程序设计语言仍然是 3GL。由于高级语言程序需要转换为机器指令才能执行，因此，高级语言程序对软硬件资源的消耗就更多，运行效率也较低。由于汇编语言和机器语言可以利用计算机的所有硬件特性并直接控制硬件，同时，汇编语言和机器语言的运行效率较高，因此，在实时控制、实时检测等领域的许多应用程序仍然使用汇编语言和机器语言来编写。

1.3.2 程序设计语言的排名

作为计算机世界中的关键技术，程序设计语言一直经历着改进和变化。目前，国内外使用的程序设计语言不下几百种，最常用的也有几十种，每种程序设计语言都有其各自的特点以及各自的应用领域。

在程序设计语言流行度的评估方面，TIOBE 一直是最为权威的机构之一。TIOBE 每个月发布前 100 位程序设计语言的份额，并进行跨年度同期比较。排行榜的数据取样来源于互联网上富有经验的程序员、商业应用、著名搜索引擎的关键字排名、Alexa（www.alexa.com，专门发布网站世界排名的网站）上的排名等。表 1.2 给出了 2010 年 8 月排名前 20 位的程序设计语言。

表 1.2　2010 年 8 月排名前 20 位的程序设计语言

2010 年 8 月排名	2009 年 8 月排名	排名变化	编程语言	2010 年 7 月流行度	自 2009 年 7 月变化值
1	1	=	Java	17.994%	–1.53%
2	2	=	C	17.866%	+0.65%
3	3	=	C++	9.658%	–0.84%

续表

2010 年 8 月排名	2009 年 8 月排名	排名变化	编程语言	2010 年 7 月流行度	自 2009 年 7 月变化值
4	4	=	PHP	9.180%	–0.21%
5	5	=	Visual Basic	5.413%	–3.07%
6	7	↑	C#	4.986%	+0.54%
7	6	↓	Python	4.223%	–0.27%
8	8	=	Perl	3.427%	–0.60%
9	19	↑↑↑↑↑↑↑↑↑↑	Objective-C	3.150%	+2.54%
10	11	↑	Delphi	2.428%	+0.09%
11	9	↓↓	JavaScript	2.401%	–0.41%
12	10	↓↓	Ruby	1.979%	–0.51%
13	12	↓	PL/SQL	0.757%	–0.23%
14	13	↓	SAS	0.715%	–0.10%
15	20	↑↑↑↑↑	MATLAB	0.627%	+0.07%
16	18	↑↑	Lisp/Scheme/Clojure	0.626%	0.00%
17	16	↓	Pascal	0.622%	–0.05%
18	15	↓↓↓	ABAP	0.616%	–0.12%
19	14	↓↓↓↓↓	RPG(OS/400)	0.606%	–0.15%
20	-	↑↑↑↑↑↑↑↑↑↑	Go	0.603%	0.00%

这么多程序设计语言应该选用哪一种来解决问题呢？还是从程序设计的本质来回答。哪一种程序设计语言能够较好地解决待求解的问题，能够较好地控制计算机的运行来解决这个问题，就可以选用哪一种程序设计语言。换言之，在选用程序设计语言的时候，不要刻意去追求最先进的技术和技巧，要以能够解决问题为标准。

1.4 程序的基本构成

程序设计语言是为了方便描述计算过程而人为设计的符号语言，设计程序设计语言的根本目标在于使人类能够以熟悉的方式编写程序，因此，程序设计语言与自然语言之间有很多相似之处。自然语言的一篇文章由段落、句子、单词和字母组成，类似地，程序设计语言的一个程序由模块、语句、单词和基本字符组成。例如，C/C++程序由一个或多个函数组成，函数由若干条语句构成，语句由单词构成，单词由基本符号构成。C/C++程序的基本构成如图 1.12 所示。

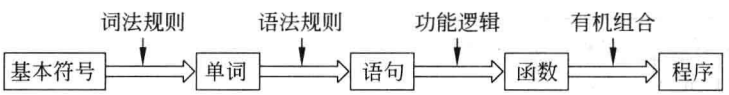


图 1.12 C/C++程序的基本构成

同自然语言一样，程序设计语言也是由语法和语义两方面定义的。其中，语法包括词法规则和语法规则，词法规则规定了如何从语言的基本符号构成词法单位(也称单词)，语法规则规定了如何由单词构成语法单位（例如表达式、语句等），这些规则是判断一个字符串是否构成一个形式上正确的程序的依据；语义规则规定了各语法单位的具体含义，程序设计语言的语义具有上下文无关性，程序文本所表示的语义是单一的、确定的。从某种角度来说，学习程序设计语言主要就是学习这些规则。

1.4.1 基本字符集

在程序设计语言中，字符是最基本的组成元素，将一些特定的字符按照一定的规则进行排列就组成了程序，这些特定的字符构成了程序设计语言的**基本字符集**。除基本字符集中的字符外，其他任何符号不允许出现在用这种程序设计语言编写的程序中。

每种程序设计语言都定义了自己的基本字符集，不同语言的基本字符集相近，一般都是计算机系统字符集（如 ASCII 码）的子集。C/C++语言的基本字符集包括：

① 英文字母，包括 26 个大写英文字母 A～Z 和 26 个小写英文字母 a～z；

② 数字，包括 0～9 共 10 个数字；

③ 空白符，包括空格符、回车符、制表符；

④ 特殊字符，包括 29 个特殊字符，如表 1.3 所示。

表 1.3 C/C++语言基本字符集中的特殊字符

+	-	*	/	^	=	(	)	{	}
[	]	<	>	;	:	|	\	!	#
%	&	'	"	,	.	?	~	_	

1.4.2 词法单位

程序设计语言的词法单位也称为**单词**，是由基本字符集中的字符根据词法规则组合而成的。程序设计语言中基本的单词有关键字、标识符、运算符、分隔符 4 种。

1. 关键字

关键字（也称保留字）是程序设计语言预先声明的单词，关键字的拼写是固定的，具有特殊的含义和作用。换言之，关键字已被程序设计语言本身使用，不能再做其他用途了。C/C++语言的常用关键字如表 1.4 所示。

表 1.4 C/C++语言的常用关键字

break	case	char	const	continue	default	do	double	else
enum	extern	float	for	if	int	long	return	short
signed	static	struct	switch	typedef	union	unsigned	void	while

2. 标识符

标识符是编程人员声明的单词，用来表示各种程序对象（如变量、类型、函数、文件等）的名字。不同的程序设计语言对于标识符的构成遵循不同的规则，C/C++语言中标识符的构成规则如下：

① 以字母（大写或小写）或下划线_开始；

② 可以由字母（大写或小写）、下划线_或数字（0～9）组成；

③ 大写字母和小写字母代表不同的标识符。

例如，以下都是非法的C/C++语言标识符：6num（以数字开始）、ok?（含有特殊字符?）、int（与关键字同名）。

使用标识符时必须注意：

① 是否区分大小写。有些语言（例如C/C++语言）区分大小写，即大写字母和小写字母代表不同的标识符，则student、Student、STUDENT代表不同的标识符；有些语言（例如Visual Basic语言）不区分大小写，则student、Student、STUDENT代表同一个标识符。

② 见名知义。标识符虽然可以由编程人员随意命名，但标识符是用于标识某个程序对象的字符序列，因此，命名应尽量体现相应的含义，以便于阅读和理解。例如，在读程序时，看到一个名为maxNum的变量就可以猜到这个变量大概是表示一个最大的数，而同样的含义用名为a的变量来表示，就很难理解这个变量的含义。

3. 运算符

运算符是程序设计语言预先规定的操作符，用于实现特定的算术运算和逻辑运算。C/C++语言中基本的算术运算符有+、–、*、/和%，分别表示加、减、乘、除和求余运算，基本的逻辑运算符有&&、||和!，分别表示与、或和非运算。

4. 分隔符

分隔符用于分隔单词或程序正文，一般不表示任何实际的操作，仅用于构造程序。空格是程序设计语言最常用的分隔符，C/C++语言规定在任何标识符、关键字和字面常量组成的两个相邻标识符之间至少有一个空格。例如，如果在变量定义int length中关键字int和标识符length之间没有空格，则intlength会被识别为一个单词。

需要强调的是，由多个字符组成的单词中间不允许夹有任何其他符号（包括空格），否则可能会改变程序的含义并且引发错误。

1.4.3 语法单位

程序设计语言的语法单位是由单词根据语法规则构成的，最常见的语法单位是表达式和语句。

1. 表达式

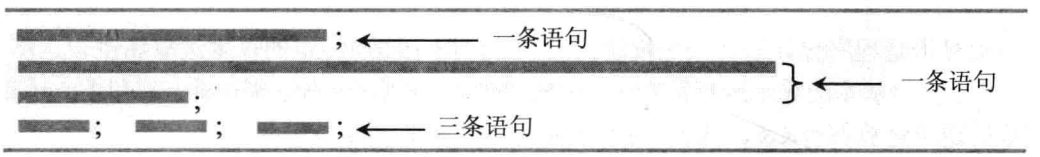
表达式由运算符、运算对象（也称操作数）和圆括号组成，能够对数据进行各种运算处理。表达式的值可以参与其他运算，即可以用作其他运算符的运算对象，这就形成了更复杂的表达式。例如，算术表达式(2 + 3) * 8–5/2。

在不同的程序设计语言中，运算符的种类、数量、表示符号和求值方向一般有所不同。C/C++语言提供了 50 多个运算符，正是丰富的运算符和表达式使 C/C++语言的功能十分完善，这也是 C/C++语言的主要特点之一。本书第 4 章介绍 C/C++语言中基本的算术表达式和逻辑表达式。

2. 语句

程序用于控制计算机进行数据处理以实现问题求解，对数据进行处理是由一个个动作组成的。**语句**是描述动作的基本单位，用来向计算机系统发出操作指令，程序的功能就是通过执行一系列语句来实现的。一个好的程序设计语言提供的语句集合应该能够方便地描述各种动作，一般包括赋值语句（给变量赋值是最基本的动作）、控制语句（控制语句的执行顺序，例如分支语句、循环语句）等，本书第 3 章介绍赋值语句，第 5 章介绍控制语句。

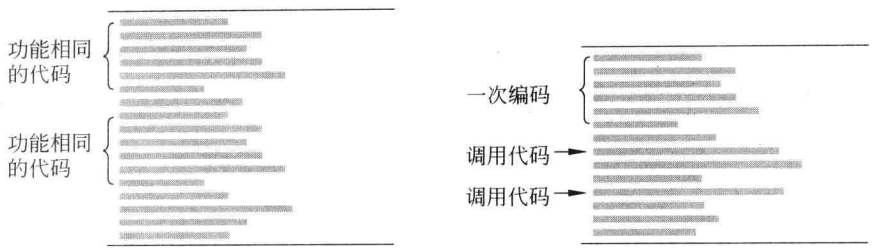
类似于自然语言中的一句话，语句与行的长短无关。有些语句比较长可以写在两行或更多行，但仍是一条语句；如果相邻语句都比较短，也可以写在同一行。因此，需要采取某种办法标识语句的结束位置，例如 C/C++语言规定每条语句都以分号结尾。



3. 模块

为了使程序的逻辑清晰，通常将一个复杂的问题分解为多个子问题，每个子问题完成一项基本功能，求解子问题的语句序列构成**模块**。模块是能够完成某种功能并可重复执行的一段程序，这段程序只需编码一次，然后在需要执行这种功能时调用这段代码，如图 1.13 所示。对于不同的程序设计语言，模块的实现机制不同，BASIC 和 FORTRAN 语言用**子程序**实现模块，Pascal 语言用**过程**实现模块，C 语言用**函数**实现模块，Java 和 C++语言用**类**实现模块。

模块机制将“做什么”和“怎么做”分离开来，不仅缩小了代码长度，而且使得程序的设计、调试和维护更加容易。在软件的发展史上，引进模块的概念是一个重大成就，对程序设计技术的发展起到了重大影响，它是模块化、分块编译、逐步求精等技术的基础。随着程序设计技术的发展，模块已不仅仅是为了简化代码，而是被提升到程序设计抽象的高度，是为了程序概念的抽象和程序的可读性。



(a) 分解前　　(b) 分解后

图 1.13　模块化思想示意图

1.4.4　程序

一个程序可以是非常简单的，也可以是特别复杂的，这取决于程序所要实现的功能和具体的程序设计语言。任何一种程序设计语言对于程序的构成都有具体的规定，程序必须严格按照该语言规定的语法和表达方式编写。

例 1.2　计算 $x+y$ 的值。程序如下：

```
1    /*   add.cpp  */                          ←——注释
2    #include <stdio.h>                        ←——文件包含预处理指令
3      ↓————— 返回类型,由操作系统检测
4    int main ( )                              ←——主函数,程序的开始执行处
5    {
6      int x, y, z;                            ←——库函数,位于 stdio.h 文件中
7      printf("请输入两个整数: ");
8      scanf("%d%d", &x, &y);                  ←——输入数据
9      z = x + y;                              ←——处理数据
10     printf ("这两个整数的和是%d", z);       ←——输出结果
11     return 0;        ←——— 将状态码 0 返回操作系统,表明程序正常结束
12   }
```

注：行号不属于 C/C++程序，是本书为了叙述方便附加的。注意在录入程序时不要录入行号。

C/C++程序具有如下特点：

① 注释。注释不是程序的可执行语句，在程序中的作用是对程序进行注解和说明。C/C++语言提供了两种注释方法：

- 行注释：用//作为行注释的开始，即从符号//开始至本行的结尾均为注释内容。
- 块注释：用/*和*/括起来，即位于符号/*和*/之间的字符均为注释内容。

② 预处理指令。C/C++程序是由函数组成的，C/C++语言的函数分为两大类：一类是编程人员编写的函数，称为**自定义函数**；另一类是 C/C++语言编译器提供的函数，称为**库函数**，如函数 scanf 和 printf 均为库函数。在使用库函数之前，必须使用预处理指令 #include 将该函数所在的库文件包含进来。

③ main 函数。一个 C/C++程序不论包含多少个函数，都有且只能有一个 main 函数，称为**主函数**。程序的可执行代码都在 main 函数中，不论 main 函数在整个程序中的位置如何，C/C++程序的执行总是从 main 函数开始，具体地说，是从{开始到}结束。

④ main 函数的典型构成。程序的主要任务是对数据进行处理，数据处理一定会涉及数据的来源问题，针对输入的数据执行数据处理过程，并输出处理结果。因此，main 函数的典型构成是输入数据、处理数据、输出结果，如图 1.14 所示。

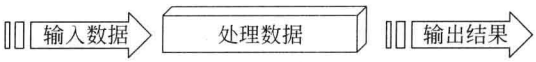


图 1.14　main 函数的典型构成

⑤ main 函数的执行。main 函数由操作系统调用，在 main 函数执行结束时通过 return 语句返回一个状态码，操作系统在程序终止时可以检测到这个状态码。状态码 0 表示程序正常终止。

良好的编程习惯 1.1

将 main 函数的返回值类型声明为 int 型，并在程序终止时返回一个状态码。如果程序正常终止，则 main 函数应该返回 0，为了说明程序异常终止，main 函数应该返回非 0 值。因为操作系统可能会检测程序的终止状态，运行程序的人也可能会检测状态码。如果程序终止时没有返回一个状态码，某些编译器会给出一个警告（warning）。

1.5　程序的上机过程

C/C++程序的上机过程通常包括程序编辑、程序编译、程序连接、运行调试等几个步骤，如图 1.15 所示。

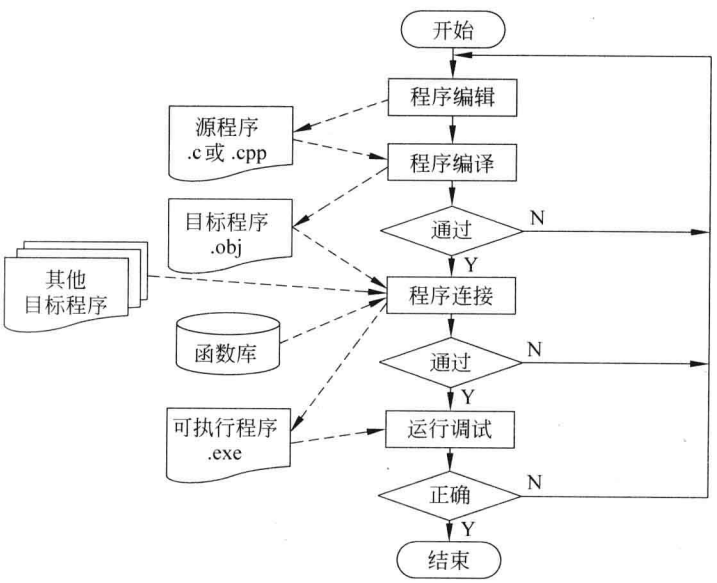


图 1.15　C/C++程序的上机过程

程序编辑。在程序编辑器中录入程序代码，并以文本文件的形式保存。在 VC++编

程环境下编辑的成品是后缀名为.cpp 的源程序文件，在 Turbo C 编程环境下编辑的成品是后缀名为.c 的源程序文件。

程序编译。利用编译器将编辑好的程序翻译成二进制代码，生成目标程序。C/C++程序编译的成品是后缀名为.obj 的目标文件。

程序连接。编译后产生的目标文件还不能直接运行，连接是把目标文件和其他目标文件（如果有的话）、系统提供的库函数以及操作系统提供的资源，连接到一个可执行文件中。C/C++程序连接的成品是后缀名为.exe 的可执行文件。

运行调试。生成可执行文件后就可以运行了，调试的目的是发现并排除程序中的语义错误和逻辑错误。如果程序达到预期目标，则结束程序设计；否则，要进一步检查、修改源程序，重复上述过程。

1.5.1 编程环境

广义上，程序设计的环境包括所有与程序设计相关的硬件环境和软件环境；狭义上，程序设计的环境是指利用程序设计语言进行程序开发的编程环境，这里只讨论狭义上的编程环境。有些语言需要指定的编程环境（例如 Visual Basic 语言），有些语言只需要一个文本编辑器（例如 JavaScript 语言），有些语言有很多可供选择的编程环境（例如 C/C++语言）。C/C++语言常用的编程环境有 Turbo C、Microsoft C、Microsoft Visual C++、Borland C++、Dev C++等。这些版本大都遵循 ANSI C 标准，只是在某些细节或库函数等方面有些差异。

目前的编程环境大都是交互式集成开发环境（Integrated Development Environment，IDE），包括程序编辑、程序编译、运行调试等功能。此外，还包括许多编程的实用程序。熟练使用编程工具和环境，也是提高编程效率的因素之一，初学者应该尽快熟悉编程环境。Microsoft Visual C++ 6.0（以下简称 VC++）集成开发环境如图 1.16 所示。

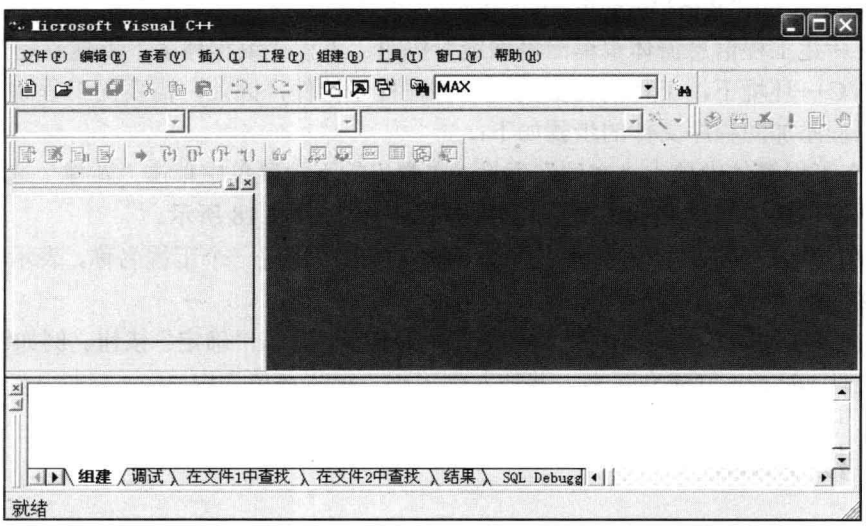


图 1.16 VC++集成开发环境

VC++使用工程来管理程序文件。建立一个工程的步骤如下：

① 在图 1.16 所示编程环境下单击“文件”菜单，在弹出的下拉菜单中单击“新建”选项，在弹出的对话框中选中 Win32 Console Application（控制台应用程序），如图 1.17 所示。

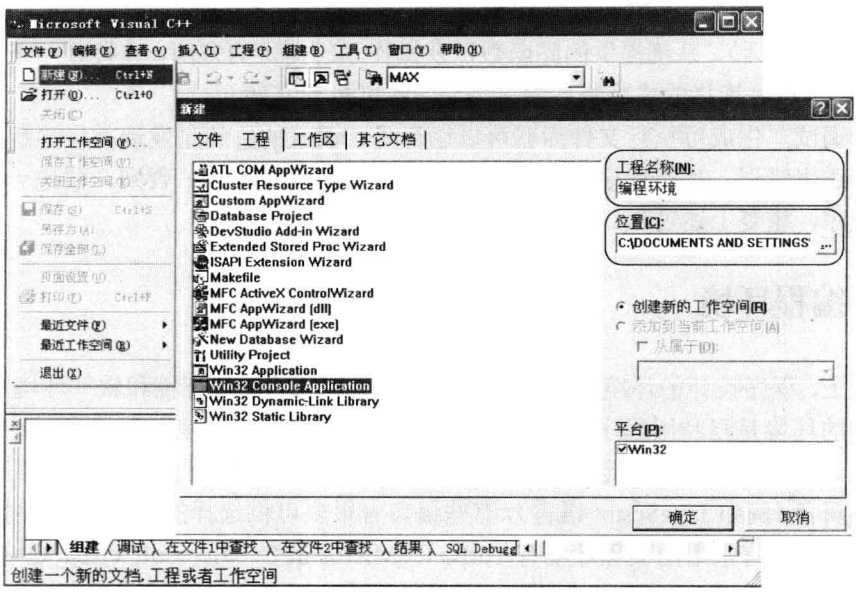


图 1.17　VC++环境下建立工程

② 在“位置”文本框创建一个文件夹，在“工程名称”文本框键入一个工程名称，单击“确定”按钮，如图 1.17 所示。例如，在“位置”文本框中选择“D:program”，在“工程名称”文本框中键入“编程环境”，其含义是在 D 盘 program 文件夹下创建一个工程“编程环境”。

③ 在出现的“Win32 Application Step 1”窗体中选择“一个空工程”，单击“完成”按钮，在新建工程信息窗体中单击“确定”按钮，工程“编程环境”就建立起来了。

在 VC++环境下，工程的主要功能是管理程序文件，因此，需要向工程添加程序文件。向工程添加源程序文件的步骤如下：

① 在工程窗体中单击“文件”菜单，在弹出的下拉菜单中单击“新建”选项，在弹出的对话框中选中 C++ Source File（C++源程序），如图 1.18 所示。

② 在对话框的右侧“添加到工程”下拉列表框中出现一个工程名称，表示将要建立的程序文件加入到该工程中，如图 1.18 所示。

③ 在“文件名”文本框中键入新建文件的名字，单击“确定”按钮。例如键入 add，则程序文件 add.cpp 就建立起来，并加入到工程“编程环境”中。

1.5.2　程序编辑

所有的程序代码都是编程人员在计算机前通过键盘录入的，录入程序以及修改程序

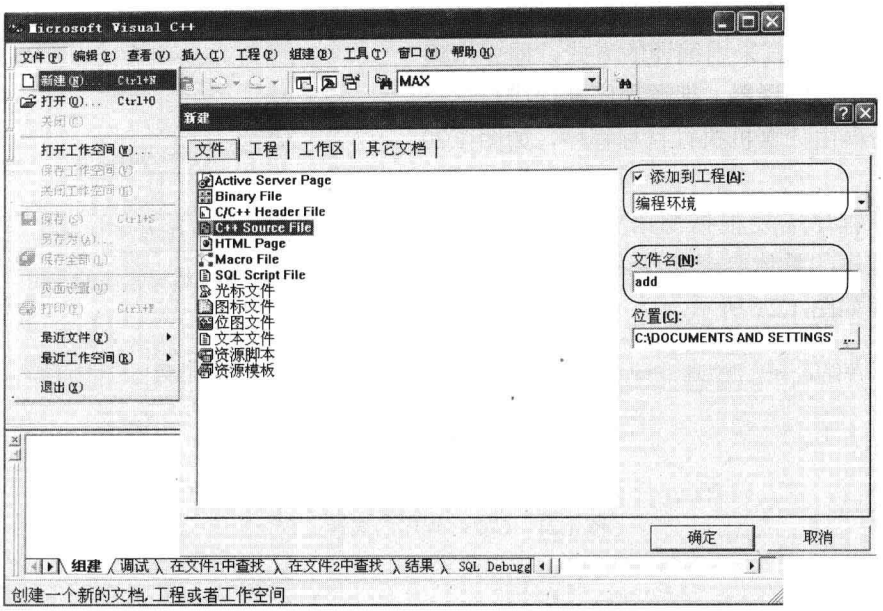


图 1.18　向工程添加程序文件

的过程称为**程序编辑**。一般的编程环境都提供编辑程序的窗口，有些编程环境用不同的颜色标注程序中不同的符号来帮助编程人员减少错误，提供帮助功能来减少编程人员需要记忆和键入的符号，提供自动缩进功能来维护程序的良好格式。VC++编程环境下程序的编辑窗口如图 1.19 所示。

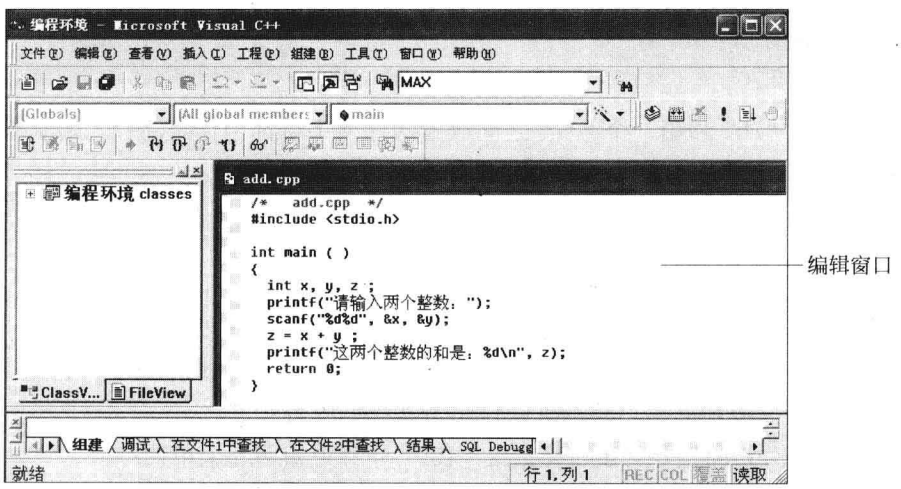


图 1.19　VC++环境下的程序编辑窗口

1.5.3　程序编译

利用高级语言编写的程序不能直接在计算机上执行，必须将高级语言程序（称为源

程序）转换为在逻辑上等价的机器指令（称为目标程序），实现这种转换的程序称为**翻译程序**。C/C++语言采用编译的方式，即先由**编译程序**（也称**编译器**）把源程序翻译成目标程序，然后再由计算机执行目标程序，如图 1.20 所示。

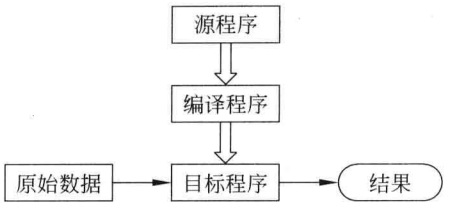


图 1.20　编译程序的作用

编译程序需要根据源语言的具体特点和对目标程序的具体要求来设计，C/C++程序的编译过程如图 1.21 所示。

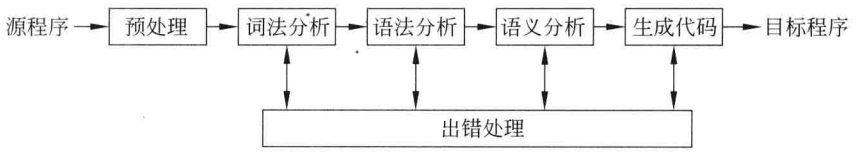


图 1.21　C/C++程序的编译过程

1. 预处理

所谓**预处理**是指在编译之前对源程序进行某些预处理工作，如滤掉注释、文件包含、宏替换等，然后再对预处理之后的源程序进行编译。从功能上讲，预处理扩展了程序设计的环境，简化了程序的开发过程，提高了程序的可读性和可移植性。

需要强调的是，注释不能嵌套，否则预处理阶段滤掉注释后会产生语法错误。例如，对于如下语句序列：

```
printf("请输入两个整数：");                /*在屏幕上显示"请输入两个整数："
scanf("%d%d", &x, &y);                    /*输入数据*/并存储到变量 x 和 y 中*/
```

预处理将第一个“/*”和第一个“*/”之间放置的任何字符都当作是注释，则滤掉注释的结果是：

```
printf("请输入两个整数："); 并存储到变量 x 和 y 中*/
```

2. 词法分析

词法分析的任务是对源程序进行扫描和分解，按照词法规则识别出一个个单词，如关键字、标识符、运算符等，并将单词转化为某种机内表示。

例如，词法分析将语句“double area = 10;”分解为如下 5 个单词：

double　area　=　10　;

①　②　③　④　⑤

其中，单词①是关键字，单词②是标识符，单词③是运算符，单词④是常量，单词⑤是分隔符。

如果发现词法错误，则指出错误位置，给出错误信息。为此，词法分析还需要标记源程序的行号，以便可以将错误信息和行号联系到一起。

良好的编程习惯 1.2

词法分析是用空格等分隔符来识别和分解单词的，因此，在源程序中加入适当的空格不仅是良好行文格式的需要，也便于编译程序进行词法分析。例如，在运算符的两边都加上一个空格，在圆括号和其他分隔符的两边都加上一个空格。

3. 语法分析

语法分析是编译程序的核心部分，它的任务是对词法分析得到的单词序列按照语法规则分析出一个个语法单位，如表达式、语句等。如果发现语法错误，则指出错误位置，给出错误信息。

例如，语法分析将语句“double area = 10 ;”表示成如图 1.22 所示的语法树，并得出分析结果：一个语法上正确的变量初始化语句。

4. 语义分析

语义分析的任务是检查程序中语义的正确性，以保证单词或语法单位能有意义地结合在一起，并为代码生成收集类型信息。语义分析的一个重要部分是类型检查，即对每个运算符的运算对象，检查它们的类型是否合法。

例如，对于语句“double area = 10 ;”，语义分析审查运算符“=”两边的运算对象，发现 area 是实型变量，而 10 是整型常量，则在语法分析得到的语法树上增加一个将整型常量转换成实型常量的语义处理结点 inttoreal，得到如图 1.23 所示的语法树。

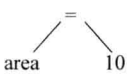


图 1.22　语法树

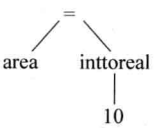


图 1.23　增加语义处理的语法树

5. 生成代码

生成代码的任务是将形式上正确的源程序转换为特定机器的目标程序。显然，高级语言和计算机的多样性为目标代码生成的理论研究和实现技术带来很大的复杂性。

VC++编程环境提供了编译按钮实现程序编译功能，如图 1.24 所示。需要强调的是，在初学程序设计时就应该培养自己的查错能力，养成编译之前人工检查的习惯，不要过分依赖编译器，这一点在编写大型程序时尤为重要。

1.5.4　程序连接

编译后产生的目标文件还不能直接运行，连接是把目标文件和其他目标文件（如果有的话）、系统提供的库函数以及操作系统提供的资源，连接到一个可执行文件中，如图 1.25 所示。

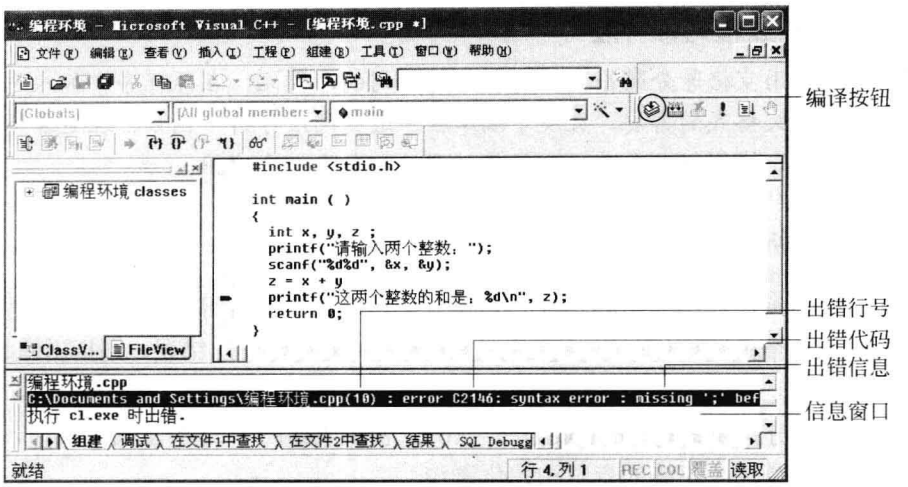


图 1.24　程序编译按钮及错误窗口

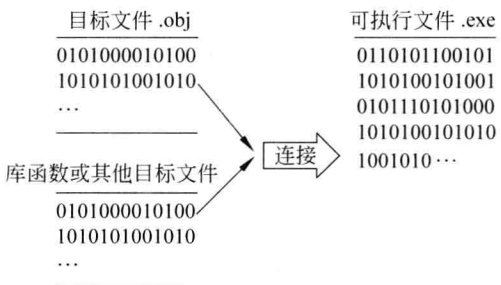


图 1.25　程序连接示意图

VC++编程环境提供了连接按钮实现程序连接功能，如图 1.26 所示。

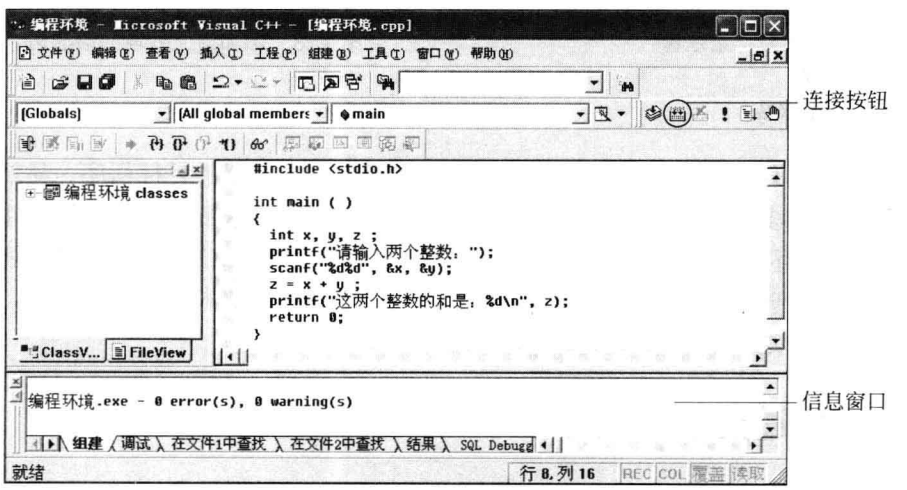


图 1.26　程序连接按钮及信息窗口

1.5.5 运行调试

程序通过编译和连接后，只能说明程序没有语法错误，但不能保证程序没有逻辑错误。如果程序存在逻辑错误，则程序的运行结果可能与期望的结果不同。查找程序中的逻辑错误需要对程序进行调试。**调试**是在程序中查找错误并修改错误的过程，调试一般需要运行程序，在运行程序的过程中给出一些测试数据，通过观察程序的阶段性结果（例如变量的变化情况）来找出错误的位置，判断出错的原因。

调试是一个需要耐心和经验的过程，也是程序设计最基本的技能。调试最困难的工作是找出错误发生的位置。一般的编程环境都会提供一些程序调试的工具，帮助编程人员找到程序的错误点。调试最主要的方法是设置断点并观察变量，具体方法如下：

① 设置断点：将光标移到程序的某条语句上，单击按钮，在该语句上设置断点标记，如图 1.27 所示。单击“组建”菜单，在弹出的下拉菜单中单击“开始调试”子菜单中的 GO 选项，则程序运行到断点处会暂停下来。

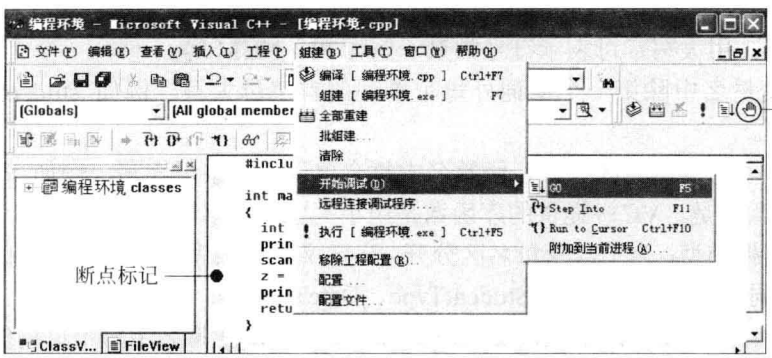


图 1.27　设置断点并调试程序

② 观察变量：当程序运行到断点的位置停下来后，就可以在调试窗口观察各个变量的值，判断此时变量的值是否正确，如图 1.28 所示。如果不正确，则说明在断点之前肯定存在错误，这样就可以把出错的范围集中在断点之前的程序上。

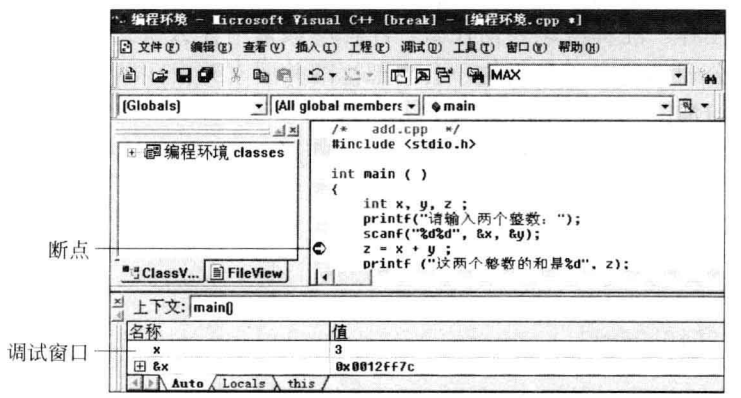


图 1.28　断点及调试窗口

1.6 程 序 风 格

随着经济的全球化，程序设计逐渐成为一种全球行为，为了使编写的应用程序在多个国家都可以使用和交流，制定一些程序设计的标准势在必行。实际上，已经出现了很多程序设计方面的国际化标准，从技术到代码书写，面面俱到。程序风格的主要作用就是使程序代码容易阅读、理解。清晰易懂的代码是衡量程序质量的一个重要标准，程序设计风格也是程序员必备的修养。良好的程序风格来源于长期的编程实践，最好的程序风格就是遵循一些行业标准。

1.6.1 标识符的命名规则

规范的标识符命名可以使程序更加统一，并且易于阅读和修改。本书对标识符的命名采用如下规则：

① 符号常量：用大写字母表示符号常量，例如 PI、NUM 等。

② 变量：变量名中除第一个单词外每个单词的首字母大写，例如 studentName、studentAddress 等。

③ 函数：为了与变量名区分开，函数名中每个单词的首字母大写，例如 PrintTri、Max 等。需要注意的是，VC++提供的库函数是用小写字母命名的。

④ 自定义数据类型：为了与变量名区分开，自定义数据类型名中每个单词的首字母大写，并且以单词 Type 结尾，例如 StudentType、DateType 等。

⑤ 全局变量：为了便于区别全局变量和局部变量，全局变量名中每个单词的首字母大写，例如 StudentNum、SecretNum 等。

1.6.2 注释

注释是程序员为了增加程序的可读性和易懂性而人为增加的说明性信息。究竟应该在什么地方加注释，注释应该包括什么，注释应该采用什么格式，业内并没有一个统一的标准，很多公司都建立了自己的一套内部标准并要求开发团队遵守和使用。下面是一些良好的注释习惯：

① 在程序前说明程序名、作者、功能、编写日期以及其他内容。例如：

```
/**********************************
* 程序名：Example1.cpp             *
* 作　者：wanghm                   *
* 功　能：求最大公约数             *
* 日　期：2010-02-11               *
**********************************/
```

② 在函数前或函数开始的地方说明函数名、作者、功能、形参的作用、编写日期以

及其他内容。例如：

```
/**********************************
* 函数名：CommonFactor              *
* 作　者：wanghm                    *
* 功　能：求两个整数的最大公约数    *
* 形　参：两个自然数 m 和 n         *
* 日　期：2010-02-11                *
**********************************/
```

③ 对程序段、语句、常量或变量定义等加注释，以说明程序段的功能、语句的作用、常量或变量的含义等。例如：

```
int length, width;                    // length 和 width 分别表示长方形的长和宽
sum = 0;                              //初始化累加器
```

当修改有注释的程序时，若修改了程序内容，则相应的注释也必须进行修改。错误的注释往往比没有注释更糟糕。

良好的编程习惯 1.3

因为注释不是真正的可执行部分，所以很多程序员往往不愿意写，但注释对程序的理解和维护非常重要。要成为一个优秀的程序员一定要养成给程序加注释的习惯，特别是在当今软件项目开发团队的合作开发过程中更是如此。

1.6.3 缩进

按照程序的嵌套层次使程序呈现阶梯形的缩进格式，即逻辑上属于同一个层次的互相对齐，逻辑上属于内部层次的推到下一个对齐的位置。例如：

```
int CommonFactor(int m, int n)
{
    int r = m % n;
    while (r != 0)
    {
        m = n;
        n = r;
        r = m % n;
    }
    return n;
}
```

1.6.4 行文格式

程序的行文格式主要通过合理利用空格和空行，使得程序层次分明、段落清晰、意

义明确。良好的行文格式能够增加程序的清晰性，也有助于发现程序中的错误。

大多数情况下，程序中单词之间空格的数量没有严格要求，如果两个单词合并后（即没有空格）不产生其他单词，则单词之间不要求一定有空格。但是，添加必要的空格可以使程序更便于阅读和理解。例如，通常每个运算符的两边都加上一个空格，有些程序员还习惯在圆括号和其他分隔符的两边都加上一个空格。观察下面两条语句，显然第二条语句更便于阅读：

```
volume=length*width*height;
volume = length * width * height;
```

空行可以在视觉上把程序划分成逻辑单元，使读程序的人更容易辨别程序的结构。例如，在主函数 main 之前加一个空行，花括号单独占一行，变量定义单独占一行。就像没有章节的书一样，没有空行的程序很难阅读。观察下面两个程序，显然第二个程序更便于阅读：

```
include <stdio.h>
int main( ){int x=3, y=5; x=x+y; printf("x=%d", x); return 0;}
```

```
include <stdio.h>
                                        //空行
int main( )
{
    int x = 3, y = 5;
    x = x + y;
    printf("x = %d", x);
    return 0;
}
```

良好的编程习惯 1.4

花括号的风格——K&R 风格，是 C 语言的创始人 Kernighan 和 Ritchie 合著的 *The C Programming Language* 一书中使用的风格，即左花括号出现在行的末尾以保持程序的紧凑，右花括号单独占一行，例如：

```
int main( ) {
  ⋮
}
```

花括号的风格——Allman 风格，因 Eric Allman 和其他 UNIX 工作者使用而得名，这种风格为了易于检查匹配，每一个花括号都单独放在一行上，例如：

```
int main( )
{
  ⋮
}
```

习 题 1

一、选择题

1. 在 C/C++程序中，main 函数的位置（　　）。
   A. 必须在最开始　　B. 必须在预处理指令的后面
   C. 可以任意　　D. 必须在最后
2. 对于 C/C++程序，下列说法中正确的是（　　）。
   A. 不区分大小写字母　　B. 一行只能写一条语句
   C. 一条语句可分成几行书写　　D. 每行必须有行号
3. C 程序文件名的后缀为（　　），C++程序文件名的后缀为（　　）。
   A. .c　　B. .cpp　　C. .obj　　D. .exe
4. C/C++程序经过编译后生成目标文件，其文件名的后缀为（　　）。
   A. .c　　B. .obj　　C. .exe　　D. .cpp
5. C/C++程序经过连接后生成可执行文件，其文件名的后缀为（　　）。
   A. .c　　B. .obj　　C. .exe　　D. .cpp
6. 编译程序的主要工作是（　　）。
   A. 检查程序的语法错误　　B. 检查程序的逻辑错误
   C. 检查程序的完整性　　D. 生成目标文件
7. 计算机硬件能唯一识别的语言是（　　）。
   A. 机器语言　　B. 低级语言
   C. 汇编语言　　D. 翻译程序
8. 下列说法正确的是（　　）。
   A. 在 C/C++源程序中，每条语句以逗号结束
   B. 在 C/C++源程序中，每行只能写一条语句
   C. 无论注释内容是什么，在对程序进行编译时都被忽略
   D. 写注释时，“/”和“*”之间可以有空格

二、简答题

1. 什么是程序？什么是程序设计？什么是程序设计语言？
2. 程序设计的过程就是用计算机求解问题的过程，请谈谈你的理解。
3. 什么是机器语言、汇编语言、高级语言？它们各有什么特点？
4. 请列举 10 种常用的程序设计语言。
5. 什么是计算机系统字符集？什么是程序设计语言字符集？二者之间是什么关系？
6. 下列标识符中，哪些是不合法的标识符？出错原因是什么？

A，P_o，P－o，from，to，_123，_abc，temp，int，6day, program, Do

7. 主函数 main 在 C/C++程序中的作用是什么？

8. 用伪代码描述下列问题的算法：

（1）求三个数中的最小值。

（2）在一个含有 n 个元素的集合中查找最小值元素和次小值元素。

（3）判定一个年份是否是闰年。闰年满足的条件是：① 能被 4 整除，但不能被 100 整除的年份；② 能被 400 整除的年份。

（4）求 1＋2＋3＋ … ＋100。

（5）判定一个整数 n 能否同时被 3 和 5 整除。

9. 请重新编辑如下程序，使程序具有良好的程序风格。

```
#include <stdio.h>
int main( )
{int flag=0; int i, x;
printf("请输入 10 个整数，可以是正数也可以是负数：");
for(i=1;i<=10;i++){scanf("%d", &x); if (x<0) {flag=1; break;}}
if (flag==1) printf("输入的整数中有负数\n"); else printf("输入的整数中没有负
数\n");
return 0;}
```

三、程序设计题

1. 计算 $x+y+z$ 的值。

2. 在显示器上输出以下信息：

```
*************************
  Welcome to Changchun
*************************
```

3. 上机调试下面两个程序，观察结果有什么不同。

```
/*    program1.cpp    */
#include <stdio.h>
int main( )
{
  printf("Welcome to China! Welcome to Changchun!\n");
  return 0;
}
/*    program2.cpp    */
#include <stdio.h>
int main( )
{
  printf("Welcome to China!\nWelcome to Changchun!\n");
  return 0;
}
```

第2章

数据的存储和组织

程序的执行过程实际上是对数据进行处理的过程，因此，程序设计的首要问题是数据的表示，即如何将待处理的数据存储在计算机的存储器中。计算机内部一切数据均用0和1的二进制编码来表示，为了能够有效地进行数据处理，就需要了解计算机内部的二进制数字世界，了解各种类型的数据是如何存储在计算机中的。

2.1 数据的存储介质——存储器

计算机使用电子器件的不同状态来表示信息，电信号一般只有两种状态，如高电平和低电平、通路和断路，因此，计算机内部是一个二进制数字世界。

2.1.1 二进制

1. 进位计数制

按进位（当某一位的值达到某个固定量时，就要向高位产生进位）的原则进行计数的方法称为**进位计数制**，简称**进制**。在日常生活中，人们使用最多的是十进制。此外，也使用许多非十进制的计数方法，例如，时间采用的是六十进制，即60秒为1分钟，60分钟为1小时；月份采用的是十二进制，即1年有12个月，等等。不同的进制以基数来区分，若以 r 代表基数，则

$r=10$ 为十进制，可使用0, 1, 2, …, 9共10个数码；

$r=2$ 为二进制，可使用0, 1共2个数码；

$r=8$ 为八进制，可使用0, 1, 2, …, 7共8个数码；

$r=16$ 为十六进制，可使用0, 1, 2, …, 9, A, B, C, D, E, F共16个数码。

一般情况下，r 进制数通常写作$(a_n\cdots a_1a_0.a_{-1}\cdots a_{-m})_r$，其中 $a_i\in\{0, 1, \cdots, r-1\}$（$-m\leqslant i\leqslant n$）。例如，二进制数1101写作$(1101)_2$，十进制数689.12写作$(689.12)_{10}$。

在进位计数制中，处于不同位置的数码代表不同的值。例如，在十进制中，数码8在个位表示8，在十位表示80，在百位表示800，而在小数点后1位表示0.8，所以，每个位置都对应一个权值，称为**位权值**。对于 r 进制数$(a_n\ldots a_1a_0.a_{-1}\ldots a_{-m})_r$，小数点前面的

位权值依次为 $r^0, r^1, \ldots, r^n$，小数点后面的位权值依次为 $r^{-1}, r^{-2}, \cdots, r^{-m}$，每个位置的数码所表示的值等于该数码乘以该位置的位权值。

进位计数制在执行算术运算时，遵守“逢 r 进 1，借 1 当 r”的规则。如十进制的运算规则为“逢 10 进 1，借 1 当 10”，二进制的运算规则为“逢 2 进 1，借 1 当 2”。二进制的算术运算规则非常简单，如表 2.1 所示。

表 2.1　二进制的算术运算规则

加	减	乘	除
0+0=0	0−0=0	0×0=0	0÷0（没有意义）
0+1=1	0−1=1（向高位借 1）	0×1=0	0÷1=0
1+0=1	1−0=1	1×0=0	1÷0（没有意义）
1+1=0（向高位进 1）	1−1=0	1×1=1	1÷1=1

2. 二进制数与十进制数之间的转换

由于人们习惯使用十进制，而计算机内部使用二进制，所以，计算机系统需要进行十进制数和二进制数之间的转换。

（1）二进制数转换为十进制数

将二进制数转换为十进制数只需将二进制数按位权值展开然后求和，所得结果即为对应的十进制数。

例 2.1　将二进制数 1101.11 转换为十进制数。

解：$1101.11 = 1\times2^3+1\times2^2+0\times2^1+1\times2^0+1\times2^{-1}+1\times2^{-2} = 13.75$

则：$(1101.11)_2 = (13.75)_{10}$

（2）十进制数转换为二进制数

将十进制数转换为二进制数需要将十进制数分解为整数部分和小数部分，分别进行转换，然后相加得到转换的最终结果。

将十进制整数转换为二进制整数的规则是：**除基取余，逆序排列**，即将十进制整数逐次除以二进制的基数 2，直到商为 0，然后将得到的余数逆序排列，先得到的余数为低位，后得到的余数为高位。

例 2.2　将十进制整数 46 转换为二进制整数。

解：

除数	商	余数	
46	23	0	↑
23	11	1	逆序排列
11	5	1	
5	2	1	
2	1	0	
1	0	1	

则：$(46)_{10} = (101110)_2$

将十进制小数转换为二进制小数的规则是：**乘基取整，正序排列**，即将十进制小数

逐次乘以二进制的基数 2，直到积的小数部分为 0，然后将得到的整数正序排列，先得到的整数为高位，后得到的整数为低位。

例 2.3 将十进制小数 0.375 转换为二进制小数。

解：

乘数	积	整数	
0.375	0.75	0	正序排列 ↓
0.75	1.5	1	
0.5	1.0	1	

则：$(0.375)_{10}=(0.011)_2$

2.1.2 存储器

存储器分为两种：内存储器（也称为内存）和外存储器（也称为外存或辅存），我们通常所说的存储器指的是内存储器，用于保存正在使用和经常使用的程序和数据。计算机的工作是在程序控制下进行的，其过程是连续不断地从内存中取出指令予以分析和执行，从内存中读取数据进行处理再存储到内存中，如图 2.1 所示。由于数据的处理过程是在内存中进行的，所以，学习程序设计的一个关键是要深刻理解内存。

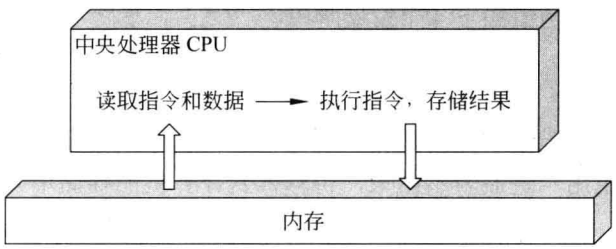


图 2.1 计算机的工作过程

内存的最小存储单位是**位**（bit），每个位可以存储 1 位二进制数。**存储单元**是可管理的最小存储单位，典型的存储单元是一个**字节**（Byte），每个字节可以存储 8 位二进制数。内存由单独的、可编址的存储单元组成，每个存储单元的编号称为**地址**。地址一般从 0 开始连续编号，如图 2.2 所示。

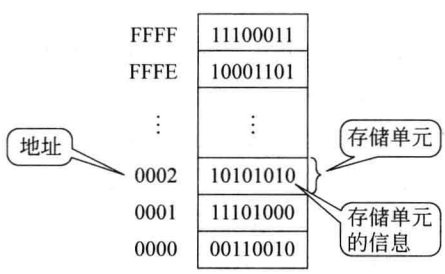


图 2.2 存储器示意图

如果要访问存储器中某个存储单元的信息，就必须知道这个单元的地址，然后按地址存入或取出信息。为了正确访问信息，必须指明一个数据占多少连续的存储单元，例如，在图 2.3 所示的存储器中，假设存储单元中存储的数据是整数，箭头指向某个存储单元，即存储该整数的存储单元的起始地址，如果从箭头开始处读 2 个存储单元，其数据值为 10248，如果读 4 个存储单元，其数据值为 534 536。

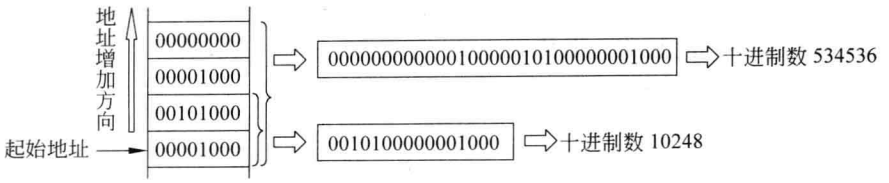


图 2.3　访问数据需要指明所占存储单元数

内存储器有两种：随机存储器 RAM 和只读存储器 ROM，我们通常所说的内存指的是 RAM。RAM 芯片包含可以存储指令和数据的电路，存储在 RAM 里的数据是芯片上流过电路的电流。这意味着只要不断电，任意时刻存储单元的内容都不会是空的，一定是 0 和 1 的编码。

2.2　数据的组织

在计算机内部，任何数据都是以二进制形式存储的，那么，计算机是如何存储数据的？如果程序需要处理大量的数据，计算机是如何组织这些数据的？在程序设计的发展过程中，数据的组织经历了如下 4 个发展阶段。

1. 二进制形式

在程序设计的初期，没有任何数据类型的概念，程序中的整数、实数、字符等数据都需要编程人员人工用二进制表示出来，再输入到计算机的存储单元中，程序对数据的处理是基于内存的，即直接操作内存。这个阶段的编程人员必须熟悉计算机硬件的物理概念和指令系统，这样，不但给编程人员带来了极大的工作量，影响了数据的可读性，而且给编程人员编写复杂程序带来了很大的困难。

2. 基本数据类型

数据通常以某种特定形式（如整数、实数、字符、逻辑等）存在，不同形式的数据其处理规则不同，例如，数值计算需要处理整数和实数，整数和实数都可以参与算术运算；人的姓名需要用字符串来表示，字符串可以拼接，还可以比较大小；有些问题的回答只有“是”或“否”两种逻辑结果，逻辑数据可以参与逻辑运算，等等。为此，程序设计语言引入了数据类型的概念。

FORTRAN、ALGOL 等高级语言的出现开启了数据抽象的篇章。高级程序设计语言根据数据对象的不同使用规则，在二进制形式存储的基础上引入了整型、实型、字符型、逻辑型等基本数据类型，使得编程人员对数据的处理可以不必基于内存，从而回避了“存储器”的概念而将精力集中于所要求解的问题。

3. 构造数据类型和自定义数据类型

基本数据类型的出现使程序设计向前迈了一大步，但是仍然无法满足复杂程序的需

要。在复杂的程序中，编程人员经常要用到一些基本数据类型的组合，于是出现了更加抽象的构造数据类型，例如数组、结构体等构造数据类型。由于用户的实际需要随着程序的不同而不同，构造数据类型虽然可以进一步简化程序设计，但是不能灵活变化，仍然不能有效地解决问题，于是，出现了自定义数据类型。

构造数据类型和自定义数据类型是在基本数据类型的基础上对数据进一步抽象，是已有数据类型的组合，编程人员可以自行组织数据而不局限于基本数据类型，为编程人员编写复杂的程序提供了有效的手段。

4. 抽象数据类型

构造数据类型和自定义数据类型只是将数据组织起来，并没有定义其上的操作。抽象数据类型是对数据类型的进一步抽象，可以将抽象数据类型理解为“数据＋操作”，即把一组数据对象及其上的操作封装成一个整体。例如，整数的数学概念和施加到整数的运算构成一个抽象数据类型，C/C++语言中的基本数据类型 int 是对这个抽象数据类型的物理实现。各种程序设计语言都具有整数类型，尽管它们在不同处理器上实现的方法不同，但由于其抽象数据类型相同，在用户看来都是相同的。抽象数据类型实现了封装和信息隐藏，为编程人员构建复杂的程序提供了极大的方便。

数据组织的发展历程反映了数据的抽象过程，是数据组织的发展趋势，也是程序设计发展的趋势之一。程序设计语言将提供粒度越来越大的数据抽象，来构建越来越复杂的程序，同时使得程序设计越来越容易，如图 2.4 所示。本章介绍基本数据类型，构造数据类型和自定义数据类型将在第 8 章、第 9 章和第 10 章介绍，有关抽象数据类型的知识将在“面向对象程序设计”、“数据结构”等课程中学到。

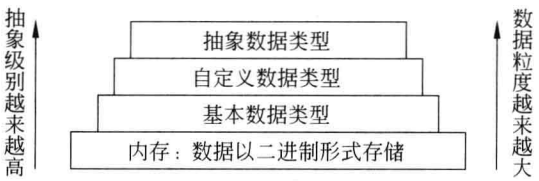


图 2.4　数据组织的发展历程——数据抽象的过程

2.3　基本数据类型

数据类型是数据的一种属性，包含三层含义：第一，分配一定的存储空间，不同类型的数据具有不同的存储格式；第二，确定一个值的集合，不同类型的数据具有不同的取值范围；第三，确定一个运算集合，不同类型的数据可以进行的运算集合不同。

基本数据类型的存储表示和计算机底层有很大关系，一般由计算机硬件直接提供，对应的操作也由硬件直接实现，因而在不同的机器中其定义和实现可能不同。

2.3.1 整型

在C/C++语言中，整型数据的基本类型说明符是int（即integer的简写）。整型数据所占字节数主要取决于机器字长和编译环境，例如，在Borland C和Turbe C中int型占2个字节，在VC++中int型占4个字节。C/C++语言提供了sizeof运算符获得某种类型的数据占用的字节数，例如，sizeof(int)的值即为int型数据占用的字节数。

1. 存储格式

在微型计算机中，整型数据的存储格式一般采用补码表示。**补码表示法**的编码规则是：采用1位二进制数表示整数的符号（称为符号位），正数的补码其符号位为0，其余各位与数的绝对值相同，负数的补码其符号位为 1，其余各位是数的绝对值取反然后在最末位加1。

例2.4 $X=+1000101$，则$[X]_{补}=\mathbf{0}\ 1000101$

$X=-1000101$，则$[X]_{补}=\mathbf{1}\ 0111010+1=\mathbf{1}\ 0111011$

$[X]_{补}=\mathbf{0}\ 1000101$，则$X=+1000101$

$[X]_{补}=\mathbf{1}\ 0111011$，则$X=\mathbf{1}\ 0111011-1=\mathbf{1}\ 0111010=-1000101$

假设int型数据占2个字节，则int型数据的存储格式如图2.5所示。注意：高字节存放在高地址的内存单元中，低字节存放在低地址的内存单元中。

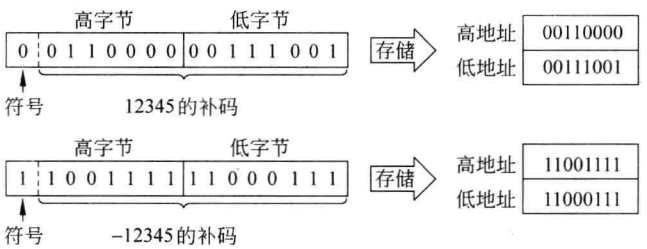


图2.5 int型数据的存储格式

2. 取值范围

抽象地讲，整型数据的值域是全体整数，但计算机是有限的，计算机内存和其他硬件设备只能存储和操作一定量的数据，因此，实际的值域只能是整数的一个子集，每种计算机能表示的整数都在某一范围之内。

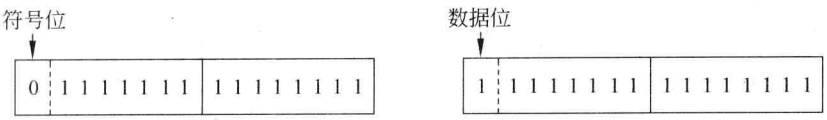
4位二进制能够表示的整数范围如表2.2所示，其取值范围是$-2^3\sim(2^3-1)$。依此类推，16位二进制能够表示的整数范围是$-2^{15}\sim(2^{15}-1)$，32位二进制能够表示的整数范围是$-2^{31}\sim(2^{31}-1)$。

表2.2 4位二进制表示的整数范围

位串	0111	0110	0101	0100	0011	0010	0001	0000	1111	1110	1101	1100	1011	1010	1001	1000
数值	7	6	5	4	3	2	1	0	−1	−2	−3	−4	−5	−6	−7	−8

注：二进制位串是对应数值的补码表示，左边第一位是符号位。

C/C++语言根据取值范围的不同，将整型数据进一步分为：基本整型（int）、短整型（修饰符 short）和长整型（修饰符 long）。在某些情况下，要处理的整数全是非负整数，此时再保留符号位就没有意义了，可以把最高位也作为数值处理，这样的数称为**无符号数**（修饰符 unsigned），对以上三种整型数据都可以加上 unsigned 表示无符号数。显然，无符号整型数据能够表示的整数范围比有符号整型数据能够表示的整数范围扩大了一倍，如图 2.6 所示。



(a) 有符号最大正整数 32767　　(b) 无符号最大正整数 65535

图 2.6　2 个字节的有符号最大正整数和无符号最大正整数

标准 C 没有具体规定各种整型数据在内存中所占的二进制位数，只要求 long 型数据的长度不短于 int 型，short 型数据的长度不长于 int 型，因而在各种机型和编程环境下可能有所不同。表 2.3 给出了 VC++定义的 6 种整数类型。

表 2.3　VC++环境下的整数类型

类　型　名	类型说明符	二进制位数	取 值 范 围
基本整型	int	32	-2^{31}～（$2^{31}-1$）
短整型	short int	16	-2^{15}～（$2^{15}-1$）
长整型	long int	32	-2^{31}～（$2^{31}-1$）
无符号整型	unsigned int	32	0～（$2^{32}-1$）
无符号短整型	unsigned short int	16	0～（$2^{16}-1$）
无符号长整型	unsigned long int	32	0～（$2^{32}-1$）

注：① 除非用 unsigned 说明，否则所有整型数据都是有符号数；

② 标准 C 没有要求类型说明符的顺序，因此 unsigned short int 和 short unsigned int 是相同的；

③ 标准 C 允许省略说明符 int 来缩写除基本整型之外的整数类型，例如 short int 可以缩写为 short，unsigned short int 可以缩写为 unsigned short；

④ C99 中增加了 long long int 类型以表示更大的整数。

3. 运算集合

在 C/C++语言中，整型数据可以进行基本的算术运算（加、减、乘、除、求余等）和逻辑运算（关系运算以及与、或、非等逻辑运算）。需要强调的是，由于整型数据能够表示的整数是有限的，如果运算的结果超过了系统能够表示的整数范围，则运算会得到错误的结果，这种错误现象称为**溢出**。

例 2.5　假设整数用 1 个字节（8 位二进制）表示，计算 68＋61 的值。

解：68＝＋1000100　　$[68]_{补}$＝**0** 1000100

61＝＋0111101　　$[61]_{补}$＝**0** 0111101

$$\begin{array}{rll} & \mathbf{0}\ 1000100 & [68]_{补} \\ + & \mathbf{0}\ 0111101 & [61]_{补} \\ \hline & \mathbf{1}\ 0000001 & [-127]_{补} \end{array}$$

由于 68+61=129 超出了 8 位二进制能够表示的整数范围，得到了错误的结果，因此，对于整型数据进行算术运算要注意溢出问题。

良好的编程习惯 2.1

无符号整数主要用于系统编程和低级的、与机器相关的应用中。因此，一般情况下，都将整数定义为有符号整数，通常是 int 型。

2.3.2 实型

实型也称为浮点型，用来表示实数，因此，实型数据也称为实数或浮点数。实型数据在计算机中一般采用浮点形式存储，C/C++语言提供了两种浮点数格式：单精度（类型说明符是 float）和双精度（类型说明符是 double），这两种类型能够表示数据的精度和范围有所不同。

1. 存储格式

在计算机中通常采用**浮点表示法**表示实数。一个实数 X 的浮点形式表示为：

$$X=M\times r^{E} \tag{2.1}$$

其中，r 表示基数，由于计算机采用二进制，因此，基数为 2；E 为 r 的指数，称为**阶码**，其值确定了实数 X 中小数点的位置；M 为实数 X 的小数，称为**尾数**，其位数反映了实数 X 的精度（即有效数字）。由于尾数中的小数点可以随 E 值的变化而左右浮动，所以，这种表示法称为浮点表示法。大多数微型计算机都把尾数规定为纯小数（绝对值小于 1 的小数），并且采用补码形式来表示阶码和尾数，如图 2.7 所示。

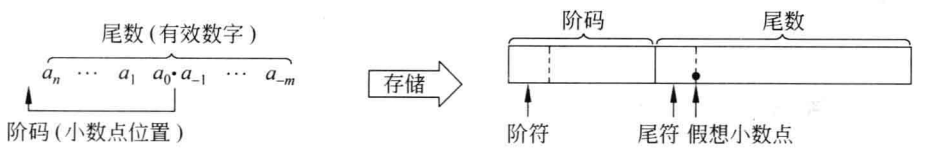


图 2.7 浮点表示法的一般格式

例 2.6 假设某 C/C++编译系统的 float 型数据占 32 位二进制，其中用 24 位表示尾数，用 8 位表示阶码，写出 68.625 在内存中的存放形式。

解：$(68.625)_{10}=(1000100.101)_2=(0.1000100101\times 2^{111})_2$，则 68.625 在内存中的存放形式如图 2.8 所示。注意，高字节存放在高地址的内存单元中，低字节存放在低地址的内存单元中。

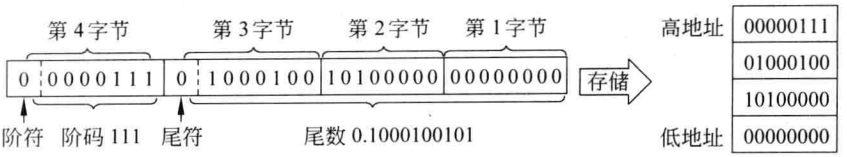


图 2.8　68.625 在内存中的存放形式

2. 取值范围

一般情况下，float 型数据在内存中占 4 个字节，double 型数据在内存中占 8 个字节，但是用多少位来表示阶码（即指数部分），用多少位来表示尾数（即小数部分），标准 C 没有具体规定，由具体的编译系统来决定。显然，尾数所占的位数越多，能表示的数据精度就越高，阶码所占的位数越多，能表示的数据范围就越大。

根据 IEEE 标准，float 型数据占 32 位，其中尾数占 24 位，阶码占 8 位，而 double 型数据占 64 位，其中尾数占 52 位，阶码占 12 位。大多数 C/C++编译系统都遵循 IEEE 规范，表 2.4 给出了 VC++定义的实数类型及其取值范围。

表 2.4　VC++环境下的实数类型

类型名	类型说明符	二进制位数	取值范围（阶码）	精度（尾数）
单精度浮点型	float	32	10^{-38}～10^{38}	7~8 位十进制有效数字
双精度浮点型	double	64	10^{-308}～10^{308}	15~16 位十进制有效数字

注：C99 中增加了 long double 类型以表示更大的实数。

3. 运算集合

在 C/C++语言中，实型数据可以进行基本的算术运算（加、减、乘、除等），同样需要注意溢出问题，要保证运算结果在系统能够表示的实数范围。由于实型数据的精度是有限的，所以一般应避免将两个实型数据比较大小，例如，对于 float 型实数 12345.6789 和 12345.6782，由于 float 型数据只能存储 8 位有效数字，第 9 位数字是系统加上的随机数，所以，这两个实数的比较结果不能确定。实型数据一般不能进行逻辑运算，对实数进行逻辑运算没有实际意义。

2.3.3　字符型

计算机除了能够进行数值计算，还具备对非数值信息的处理能力。字符是最基本的非数值数据，因而在程序设计语言的基本数据类型中大都提供了字符型。

1. 存储格式

字符型数据在计算机中被编码成二进制位串进行存储，具体的编码规则取决于系统采用的字符集。微机上常用的字符集是标准 ASCII 码（American Standard Code for Information Interchange，美国信息交换标准代码）。它由 7 位二进制数表示一个字符，总

共可以表示 128 个字符。附录 A 给出了标准 ASCII 码，每个字符都有一个由二进制位串决定的编码值，例如，字符 a 对应的二进制位串是 1100001，其编码值为 97。

在 C/C++语言中，字符型的类型说明符是 char。一般情况下，char 型数据在内存中需要一个字节（即 8 位二进制）存储对应的 ASCII 码，例如，字符 a 在内存中的存放形式如图 2.9 所示。

图 2.9　字符 a 在内存中的存放形式

2. 取值范围

由于字符型数据没有符号，因此，在计算机底层字符型的二进制位串与无符号整数的存储格式相同，字符型数据的取值范围是 ASCII 码值为 0～255 对应的字符。

3. 运算集合

不同的程序设计语言为字符型数据提供的运算集合不同。C/C++语言中的字符型数据可以进行算术运算，例如，'b'－'a'的值为这两个字符在 ASCII 码中的距离，即 ASCII 码值之差。但是，对两个字符型数据执行加、乘和除等算术运算没有实际意义。

2.3.4　逻辑型

计算机除了能够进行算术运算，还能够进行逻辑判断，具备分辨各种情况的信息处理能力，因此，很多程序设计语言的基本数据类型都包括逻辑型。

1. 存储格式

逻辑型也称为布尔型，不同的程序设计语言对逻辑型数据的表示不尽相同，同一程序设计语言的不同编译器对逻辑型数据的表示也不尽相同，甚至同一编译器的不同版本对逻辑型数据的表示也有所区别。例如，Pascal 语言中的逻辑型说明符是 boolean，并且以逻辑值 TRUE 表示真，以逻辑值 FALSE 表示假；C/C++语言中没有逻辑型，以 1（或非 0）表示真，以 0 表示假；Visual C++在 5.0 版本之前没有逻辑型，在 5.0 版本以后将逻辑型定义为一种基本数据类型，其类型说明符为 bool（来源于逻辑代数的奠基人乔治·布尔 George Boole），在内存中通常需要一个字节的存储空间。

2. 取值范围

原则上，逻辑型数据只有两个可能的取值：真和假，但在不同的程序设计语言中逻辑型数据的取值有所不同。例如，Pascal 语言以逻辑值 TRUE 表示真，以逻辑值 FALSE 表示假；C 语言以非 0 表示真，以 0 表示假。

3. 运算集合

原则上，逻辑型数据只能进行逻辑运算，但不同的程序设计语言为逻辑型数据提供的运算集合不同。例如，Pascal 语言中的逻辑型数据只能进行逻辑运算；C/C++语言中的

逻辑型数据可以作为数值型数据进行算术运算。

习 题 2

一、选择题

1. 在计算机内一切信息的存取、传输和处理都是以（　　）形式进行的。

A. ASCII 码　　B. 二进制　　C. 十进制　　D. 十六进制

2. 十进制数 35 转换成二进制数是（　　）。

A. 100011　　B. 0100011　　C. 100110　　D. 100101

3. 用 8 位二进制表示有符号整数，可表示的最大整数是（　　）。

A. 127　　B. 128　　C. 256　　D. 255

4. –23 的 8 位二进制补码是（　　）。

A. 00010111　　B. 11101001　　C. 11101000　　D. 10010111

5. 计算机工作时，内存储器用来存储（　　）。

A. 程序和指令　　B. 数据和信号

C. 程序和数据　　D. ASCII 码和数据

6. 在内存中，存储单元是（　　）。

A. 最小存储单位　　B. 可管理的最小存储单位

C. 以字节为单位　　D. 存储程序

7. 下面（　　）不是 C/C++语言的基本数据类型。

A. unsigned int　　B. double　　C. char　　D. string

8. 在 C/C++语言中，字符型数据进行（　　）运算没有实际意义。

A. +　　B. -　　C. >　　D. <

二、简答题

1. 程序设计语言中为什么要引入数据类型？

2. 基本数据类型是在什么基础上定义的？标准 C 有哪些基本数据类型？

3. 假设 int 型数据占 2 个字节，写出 123 和–123 在内存中的存储格式。

4. 假设 int 型数据占 2 个字节，计算 30000 + 30000 的值。

5. 将十进制小数 0.1 转换为二进制小数（假设精度为小数点后 10 位二进制数）。

6. 假设某 C/C++编译系统的 float 型数据占 32 位二进制，其中用 24 位表示尾数，用 8 位表示阶码，写出十进制数 4.6 在内存中的存放形式。

7. 写出字符串 computer 在内存中的存放形式。

8. 在 C/C++语言中，如何表示逻辑值“真”和“假”？如何判断一个运算量的“真”和“假”？

三、程序设计题

1. 设长方形的长为 x，宽为 y，计算长方形的周长和面积。

2. 已知某位学生的数学、物理和英语成绩分别是 89.5 分、73 分和 72.5 分，求该学生三门课程的总分和平均分。

3. 对于给定字符，请输出该字符的前驱字符和后继字符。

第3章

数据的基本表现形式

任何程序都是对数据进行操作的指令集合，程序的执行过程实际上是对数据进行处理的过程，因此，在程序设计中，数据的存储和管理占有非常重要的地位。在程序设计语言中，数据的基本表现形式有两种：常量和变量，常量用来表示在程序运行过程中不能改变的数据，变量用来表示在程序运行过程中可以改变的数据。为了便于数据的存储和管理，常量和变量都有各自的数据类型。

【任务 3.1】计算圆的面积

【问题】 给定圆的半径，求圆的面积。

【想法】 设圆的半径为 radius，圆的面积为 area，则计算圆的面积公式如下：

$$area = \pi \times radius^2 \tag{3.1}$$

【算法】 设 radius 表示圆的半径，area 表示圆的面积，算法如下：

伪代码

```
step1: 给定半径值 radius;
step2: 根据式(3.1)计算圆的面积 area;
step3: 输出 area;
```

算法需要处理 π（假设 π 等于 3.14）和 2（平方）等在运行过程中不变的数据（称为常量），以及 radius 和 area 等在运行过程中变化的数据（称为变量），程序设计语言如何表示常量和变量呢？

3.1 常　　量

所谓**常量**，就是在程序的运行过程中其值不能被改变的量，即不接受程序修改的固定值，例如程序中的具体数字、字符等。程序设计语言一般提供两种类型的常量：字面常量和符号常量。

3.1.1 字面常量

字面常量是指常量本身的字面意义就是它所代表的常量值。字面常量可以直接书写在程序中，并且字面常量的数据类型由其表示方法决定。例如，任务 3.1 中的 2（平方）是数据类型为整型的常量，令 π 等于 3.14，则 3.14 是数据类型为实型的常量。C/C++语言的字面常量及其数据类型如图 3.1 所示。

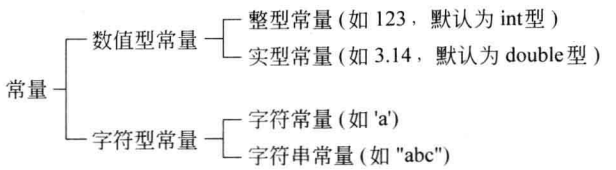


图 3.1　C/C++语言的常量及其数据类型

1. 整型常量

整型常量一般用来表示整数。C/C++语言允许以十进制（不能以数字 0 开头）、八进制（以数字 0 开头）和十六进制（以 0x 或 0X 开头）表示一个整型常量。编写程序时通常采用十进制、八进制和十六进制更适合编写某些低级程序。表 3.1 是整型常量示例。

表 3.1　整型常量示例

示例 / 进制	合法的整型常量表示	不合法的整型常量表示
十进制	123（十进制整数 123）	0123（不能以数字 0 开头）
	+123（十进制整数 123）	–123,456（不能含有逗号）
	–123（十进制整数–123）	12A（不能含有非十进制数码）
八进制	0123（相当于十进制整数 83）	123（无前导 0）
	+0123（相当于十进制整数 83）	O123（前导不能是字母 O 或 o）
	–0123（相当于十进制整数–83）	087（不能含有非八进制数码）
十六进制	0x123（相当于十进制整数 291）	5A（无前导 0x 或 0X）
	+0x9f（相当于十进制整数 159）	OX12A（前导不能是字母 O 或 o）
	–0X1AF0（相当于十进制整数–6896）	0x9g7（不能含有非十六进制数码）

需要说明的是，任何一个整数都可以用这三种形式表示，十进制、八进制和十六进制只是整数的三种不同的表现形式，在计算机内部都转换为相应的二进制数存储。例如，整数 10 的十进制、八进制和十六进制表示分别为 10、012 和 0xa，在内存中对应同一个二进制数，如图 3.2 所示。

2. 实型常量

实型常量一般只采用十进制，其表示形式有小数形式和指数形式。

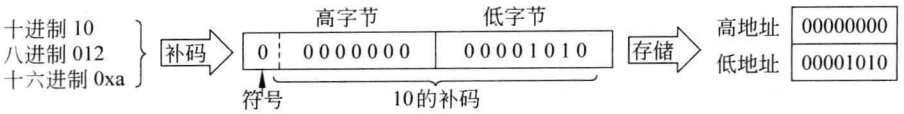


图 3.2　不同进制的表现形式对应同一个二进制存储

① 小数形式：包含小数点，并且小数点两边必须有数字。

② 指数形式：包含字母 E（或 e），并且 E 之前必须有数字，E 之后必须是整数。

表 3.2 是实型常量示例。

表 3.2　实型常量示例

形式 \ 示例	合法的实型常量表示	不合法的实型常量表示
小数形式	12.3，12.0，0.12	.3(小数点前没有数字)
	–12.0，–0.12	3. (小数点后没有数字)
指数形式	1E2，12.3E5，12.3E–5	1.2E1.5(E 之后是小数)
	–12.3E5，–12.3E–5	E5(E 之前没有数字)

3. 字符常量

字符常量通常指的是单个字符，在 C/C++语言中用单引号将单个字符括起来，其中单引号是分隔符，例如'a'、'A'、'1'、'#'等都是字符常量。有些特殊字符（例如换行符等不可见字符）无法采用这种方式表示，为了表示字符集中的每一个字符，C/C++语言提供了转义字符。**转义字符**以反斜线\开头，其含义是将反斜线后面的字符转换成另外的含义，例如'\n'表示换行。常用的转义字符表示如表 3.3 所示。

表 3.3　常用的转义字符表示

转义字符	含　义	转义字符	含　义
\a	响铃（BEL）	\\	反斜线
\b	退格	\?	问号
\f	换页	\'	单引号
\n	换行	\"	双引号
\r	回车	\0	空字符（NULL）
\t	水平跳格（tab）	\ddd	1～3 位八进制数所代表的字符
\v	垂直跳格	\xhh	1～2 位十六进制数所代表的字符

注：（1）反斜线后面的字母（即转义字符）只能是小写字母；

（2）垂直跳格\v 和换页\f 对屏幕没有任何影响，但会影响打印机的操作；

（3）反斜线后面的八进制数（或十六进制数）表示对应的 ASCII 码所表示的字符，例如\376 代表图形字符■。

4. 字符串常量

字符串常量通常指的是字符序列，在 C/C++语言中用双引号将字符序列括起来，其

中双引号是分隔符，例如"This is a string"、"a"、"%d"等都是字符串常量。不同的程序设计语言存储字符串的方式有所不同，C/C++语言对字符串的存储方式为：串中的每个字符（转义字符被看成是一个字符）以 ASCII 码值的二进制形式连续存储，并在最后一个字符的后面存放一个终结符'\0'（ASCII 码值为 0）。例如字符串"This is a string"在内存中的存放形式如图 3.3 所示。注意在写字符串时不能加终结符'\0'，'\0'是系统自动加上的。

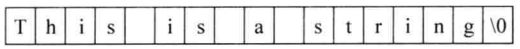


图 3.3　字符串"This is a string"在内存中的存放形式

3.1.2　符号常量

符号常量是用标识符来代表常量，实际上就是为字面常量起个名字。C/C++语言可以用预处理指令#define 定义符号常量，也可以用 const 语句定义符号常量。

1. 用#define 定义符号常量

【语法】　用#define 定义符号常量的一般形式如下：

```
#define 符号常量  常量值          ←——  行尾没有分号
```

其中，#define 是预处理指令，因此，行尾不能有分号；符号常量是一个标识符；常量值可以是一个字面常量，也可以是一个表达式。

【语义】　将符号常量的值定义为常量值，在对源程序进行预处理时，将每一个符号常量用其表示的常量值进行替换。

例如，如下预处理指令定义了符号常量 PI 的值：

```
#define  PI  3.14
```

程序中的语句：

```
area = PI * radius * radius;
```

预处理后的结果为：

```
area = 3.14 * radius * radius;
```

2. 用 const 定义符号常量

【语法】　用 const 定义符号常量的一般形式如下：

```
const 类型名 符号常量 = 常量值;      ←——  以分号结尾
```

其中，类型名是任意合法的数据类型，包括基本数据类型和自定义数据类型；符号常量是一个标识符；常量值可以是一个字面常量，也可以是一个表达式，但是其值的数据类

型必须与类型名兼容；const 是一条语句，因此要以分号结尾。

【语义】 定义一个符号常量并指定常量值。

例如，如下语句定义了符号常量 PI 的值：

```
const double PI = 3.14;
```

程序中的语句：

```
area = PI * radius * radius;
```

在执行该语句时，符号常量 PI 用 3.14 代替。

良好的编程习惯 3.1

为了与变量名、函数名等标识符区分开，符号常量通常只用大写字母，这虽然不是 C/C++语言本身的要求，但是大多数 C/C++程序员都遵循这个规范，而且这个规范已经被沿用几十年了。

使用符号常量有如下好处：

① 程序的可读性好。符号常量的名字通常要尽可能地表达它所代表的含义，以提高程序的可读性。

② 程序的可修改性好。如果程序中需要对多次使用的符号常量的值进行修改，只需对定义符号常量中的常量值进行修改，即只修改一次。例如，如果程序需要提高 π 的精度，则只需修改符号常量的定义语句：

```
const  double  PI = 3.1415926;
```

③ 避免误操作。符号常量的值在其作用域内不能再被赋予其他值，这样可以避免程序中的误操作。例如，如果程序中已经对 PI 进行了符号常量定义：

```
const  double  PI = 3.14;
```

则对于如下赋值语句：

```
PI = 3.1415926;
```

编译器将给出语法错误提示。

3.2 变　　量

所谓**变量**，就是在程序的运行过程中其值可以被改变的量。变量是程序设计的一个最基本的概念，大多数程序在执行时都需要从外界接收一些输入，在得到结果之前往往需要执行一系列的计算。因此，在程序的执行过程中需要用存储单元来存储这些数据，这类存储单元就是变量。

3.2.1 变量的概念

变量用一个标识符来表示，称为**变量名**。每个变量都属于某个确定的数据类型，在内存中占据一定的存储单元，在该存储单元中存放变量的值，因此，变量具有如下属性：

① 地址：变量所在存储单元的编号。不同类型的变量占据不同数量的存储单元，存放变量的第一个存储单元的地址（即变量的起始地址）就是该变量的地址。

② 变量名：变量所在存储单元起始地址的助记符。变量名本身不占用存储空间，可以通过变量名直接操作这个变量。

③ 变量值：存储在相应存储单元中的数据，即该变量所具有的值。可以通过变量名存取该存储单元中的数据。

④ 类型：变量所属的数据类型。类型决定了变量的存储方式（即该变量占据存储单元的字节数和存储格式），以及允许对变量采取的操作。

编译器在对源程序进行编译时，会给每个变量按照其所属的数据类型分配一块特定大小的存储单元，并将变量名与这个存储单元的起始地址**绑定**在一起。例如，设有 int 型变量 weight 分配在内存 F000 开始的存储单元中，变量 weight 的值为 100，变量的属性如图 3.4 所示。

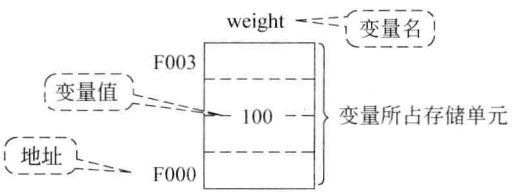


图 3.4　变量的属性

在程序中对变量进行存取操作，实际上是通过变量名找到相应的存储地址，将数据存入该存储单元，或从该存储单元中读取数据。高级语言把计算机硬件中的存储单元抽象为变量，使得编程人员对数据的处理可以不必基于内存，从而回避存储器的概念而将精力集中于所要求解的问题。

3.2.2 变量的定义和初始化

程序中需要哪些变量，变量应该采用什么数据类型，是由具体问题的需要决定的。变量的属性由变量定义规定，即在变量定义中引进变量并规定该变量的属性。

【语法】　变量定义的一般形式如下：

类型说明符　变量名列表；　　←——　以分号结尾

其中，类型说明符必须是有效的数据类型，包括基本数据类型和自定义数据类型；变量名列表是一个变量名或由逗号分隔的多个变量名；最后用分号表示结束变量定义。

【语义】　将变量名列表的各个变量定义为类型说明符的类型，编译器为各变量分配

相应的存储单元。

良好的编程习惯 3.2

很多程序员在为变量命名时只用小写字母，而且为了命名更清晰，还会在中间插入下划线，例如 student_name、student_address。有些程序员避免使用下划线，把变量名中除第一个单词外的每个单词的首字母大写，例如 studentName、studentAddress。本书采用第二种方法。

变量定义后编译器会给该变量分配一块存储空间，但是从程序开始执行到给变量赋值之前，该变量是没有确定值的，这时称该变量为“**值无定义的**”，严格来说，该变量的值是一个随机数。下面是几个变量定义的例子，其存储示意图如图 3.5 所示。

```
int radius, area;              //定义 radius 和 area 为整型变量,只能存储整数
float length;                  //定义 length 为单精度实型变量,可以存储小数
double width;                  //定义 width 为双精度实型变量,可以存储小数
char ch;                       //定义 ch 为字符型变量,可以存储字符
```

图 3.5　值无定义的变量

变量的初始化是指在定义变量的同时为其赋初值，使该变量成为“**值有定义的**”。

【语法】　变量初始化的一般形式如下：

类型说明符　变量名 = 值;

其中，类型说明符是有效的数据类型；变量名必须是一个，如果为多个变量进行初始化，则用逗号分隔；最后用分号表示结束变量的初始化。

【语义】　将变量定义为类型说明符表示的类型，并给该变量赋初值。

下面是变量初始化的例子，变量初始化的结果如图 3.6 所示。

```
int radius = 10, area;              //定义变量 radius 和 area,并为 radius 赋初值
float length = 3.5, width = 2.5;    //定义变量 length 和 width,并赋初值
```

图 3.6　变量的初始化

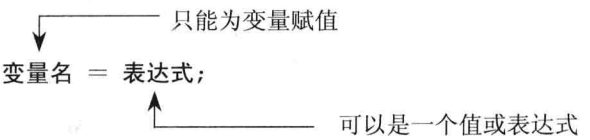
良好的编程习惯 3.3

在定义一个变量时就给变量赋初值，使变量处于“值有定义的”状态。如果定义一个变量时没有为其赋初值，然后就直接引用这个变量是很危险的，因为此时变量的值可能是一个随机值，从而引发程序的逻辑错误，有时会得到莫名其妙的结果。

3.2.3 变量的赋值

从内存的角度看，程序运行的过程是通过变量定义向内存申请存储空间，并通过赋值的方式不断修改这些存储单元的内容，最终得到问题的解。因此，为变量赋值是程序中最基本的操作，甚至可以说程序是由嵌入到不同控制结构中的赋值语句组成的。

【语法】 变量赋值的一般形式如下：



其中，=是赋值运算符。

【语义】 计算表达式的值，然后将这个值（即表达式的运算结果）存储到变量中。

所谓变量就是一段内存单元，给变量赋值就是向这段内存单元写入数据，可以将赋值运算符=理解为写入←，右侧是要写入的数据，左侧是接收数据的变量。例如，length = 3.5 的含义是 length ← 3.5，即将 3.5 写入（存储）到变量 length 中。下面是变量赋值的例子，其存储示意图如图 3.7 所示。

```
float length, width, area;      //定义 length, width,area 为 float 型变量
length = 3.5;                   //赋值语句,将 3.5 存储到变量 length 中
width = 2.5;                    //赋值语句,将 2.5 存储到变量 width 中
area = length * width;  //赋值语句,将 length * width 的结果存储到变量 area 中
```

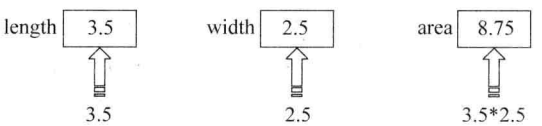


图 3.7 变量的赋值操作

一个变量被赋值后便成为“值有定义的”，对“值有定义的”变量可以读出该变量的值，并且读取操作执行后该变量的值保持不变，相当于从中复制一份出来。例如，在图 3.7 中为变量 area 赋值时首先读取变量 length 和 width 的值，然后执行表达式的计算，最后将表达式的值赋给变量 area，而变量 length 和 width 的值保持不变。

对“值有定义的”变量可以重新给它赋予新值，变量一旦被赋予新值便失去了它原来的值，相当于用新值覆盖了旧值。例如，执行变量初始化语句：

```
int radius = 10;                //定义 radius 为 int 型变量，同时赋初值为 10
```

再执行赋值语句：

```
radius = radius + 20;
```

则变量 radius 的值被赋予新值，如图 3.8 所示。这是因为变量占用的存储单元只能存放一个值，向该存储单元存入一个新值，其旧值必然被覆盖掉。

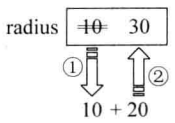


图 3.8　变量的重新赋值

良好的编程习惯 3.4

C/C++语言提供了复合赋值运算符来简写赋值语句，将算术运算符与赋值运算符组合在一起就构成了复合赋值运算符，具体有+=、-=、*=、/=和%=等。例如，可以将赋值语句“radius = radius + 20;”简写为“radius += 20;”。但为了提高程序的清晰性，很多程序员不习惯使用复合赋值运算符。

C/C++语言还提供了自增（++）和自减（--）运算符实现变量的加 1 和减 1 操作，++操作使变量的值增 1，--操作使变量的值减 1。

【语法】 变量自增（自减）操作的一般形式如下：

前置运算：运算符放在变量之前	或	后置运算：运算符放在变量之后
++(--)变量		变量++(--)

【语义】 前置运算：先使变量的值增（减）1，再以变化后的值参与其他运算；后置运算：先以变量的值参与其他运算，再使变量的值增（减）1。

从本质上讲，自增（自减）运算是变量加 1（减 1）运算的简写，例如，语句：

```
i++;
```

和

```
++i;
```

等价于

```
i = i + 1;
```

语句：

```
b = ++a;
```

等价于

```
a = a + 1;
b = a;
```

语句：

```
b = a++;
```

等价于

```
b = a;
a = a + 1;
```

给变量赋值需要注意如下两个问题：

① 取值范围。由于变量所占的存储单元数是有限的，因此变量能够表示的数据都有一定的取值范围，对变量进行赋值要保证变量值在这个范围内。例如：

```
short int radius = 60000;
```

由于在 VC++中，short int 型变量占 2 个字节，其取值范围是-32 768～32 767，因此，60000 超出了 short int 型变量的取值范围从而产生溢出。又如：

```
float length = 1.0e100;
```

由于在 VC++中，float 型变量占 4 个字节，其取值范围是 10^{-38}～10^{38}，因此，1.0e100 超出了 float 型变量的取值范围从而产生溢出。再如

```
float length = 123.456789;
```

由于 float 型变量最多只能精确表示 8 位有效数字，因此，只能精确存储前 8 个数字即 123.45678，最后一位数字是一个随机值。

良好的编程习惯 3.5

有些编译器对于 float 型数据的有效数字达不到 8 位，建议程序中对于浮点型数据采用 double 型存储。

② 数据类型。当赋值运算符“=”两侧的数据类型不一致时，系统将进行自动类型转换，一般是将右侧表达式值的数据类型转换成左侧变量的数据类型，数值数据的具体转换情况如下：

- 将实型数据赋给整型变量时，将舍弃该实型数据的小数部分。例如：

  ```
  int radius = 6.85;
  ```

  则变量 radius 的值为 6。
- 将整型数据赋给实型变量时，数值不变，但将该整型数据以浮点形式存储到变量中。例如：

  ```
  float length = 123;
  ```

  则变量 length 的值为 123.00000。

3.2.4 强制类型定义

在使用变量之前需要对其进行定义，某些语言的定义是显式的，例如 C/C++语言规定变量在使用前必须进行定义；某些语言提供隐式或默认的定义，例如 FORTRAN 语言将变量 INDEX 默认为整型变量。

大多数程序设计语言要求对程序中的所有变量必须“先定义，后使用”，即在变量定义中引进变量并规定该变量的类型等属性，称为变量的**强制类型定义**。强制类型定义便于编译器的工作，使编译器为变量分配相应的存储单元以及将变量名与其存储单元的起始地址进行绑定。具体体现在：

① 便于编译器进行词法检查，使程序中的变量不发生拼写错误。例如，如果程序中

定义了变量 studentNum：

```
int studentNum;
```

而在程序的执行部分错写作 studantNum，例如：

```
studantNum = 3;
```

则编译器在对程序进行编译时会检查出变量 studantNum 没有定义过，这样可以避免程序中出现单词的拼写错误。

② 便于编译器分配相应的存储单元，将变量按照某种规定的格式进行存储。例如，如果程序中定义了整型变量 length 和 width：

```
int length, width;
```

则在 VC++中，编译器在对程序进行编译时，会为变量 length 和 width 各分配 4 个字节的存储空间，并按整数方式（例如补码形式）存储数据。

③ 便于编译器检查运算是否合法，使表达式的运算不发生类型错误。例如，如果程序中定义了实型变量 a 和 b：

```
double a, b;
```

而在程序的执行部分有求余运算：

```
a = a % b;
```

则编译器在对程序进行编译时会发现该运算对象的类型不合法（%运算要求运算对象为整型），给出相应的错误信息。

3.3 解决任务 3.1 的程序

```
1   /*   duty3-1.cpp   */
2   #include <stdio.h>                  //使用库函数 printf
3   #define PI 3.14;                    //定义符号常量 PI
4                                       //空行,以下为主函数
5   int main( )
6   {
7     int radius;                       //定义 radius 为整型变量,存储半径
8     double area;                      //定义 area 为实型变量,存储圆的面积
9     radius = 10;                      //给变量 radius 赋值
10    area = PI * radius *radius;       //给变量 area 赋值
11    printf("radius = %d\n", radius);
                                        //输出整型数据,其中%d 为整型格式符
12    printf("area = %6.2f\n", area);
                                        //输出实型数据的宽度为 6 位,其中小数占 2 位
13    return 0;                         //将 0 返回操作系统,表明程序正常结束
14  }
```

运行结果如下：

```
radius = 10
area = 314.00
```

3.4 程序设计实例

3.4.1 实例 1——华氏温度转换为摄氏温度

【问题】 将给定的华氏温度转换为对应的摄氏温度。

【想法】 设 C 表示摄氏温度，F 表示华氏温度，华氏温度转换为摄氏温度的计算公式如下：

$$C = 5 \times (F - 32) / 9 \tag{3.2}$$

【算法】 设变量 temC 表示摄氏温度，变量 temF 表示华氏温度，算法如下：

伪代码

```
step1: 给定华氏温度 temF;
step2: 根据式(3.2)计算摄氏温度 temC;
step3: 输出 temC;
```

【程序】

```
1   /*   温度转换.cpp   */
2   #include <stdio.h>                    //使用库函数 printf
3                                         //空行,以下为主函数
4   int main( )
5   {
6     double temC, temF;
                          //double 型变量 temC 和 temF 存储摄氏温度和华氏温度
7     temF = 100;                         //为变量 temF 赋值
8     temC = 5 * (temF - 32)/9;           //将表达式的运算结果赋给变量 temC
9     printf("华氏温度%6.2f 对应的摄氏温度是%6.2f\n", temF, temC);
                                          //输出结果
10    return 0;                           //将 0 返回操作系统,表明程序正常结束
11  }
```

运行结果如下：

```
华氏温度 100.00 对应的摄氏温度是 37.78
```

3.4.2 实例 2——计算本息和

【问题】 假设人民币储蓄年利率为 2.5%，利息所得税为 20%，现存入 1000 元人民币，求一年后本息和为多少元？

【想法】 设 r 表示年利率，x 表示本金，y 表示利息，则利息的计算公式如下：

$$y = x \times r \tag{3.3}$$

【算法】 设变量 interest 表示利息，变量 foundMoney 表示本金，sumMoney 表示本息和，rate 表示年利率，算法如下：

伪代码

```
step1: 给定本金值 foundMoney;
step2: 根据式(3.3)计算利息 interest;
step3: sumMoney = foundMoney + interest * 0.8;
step4: 输出 sumMoney;
```

【程序】 由于利率是一个相对固定的常量，为了提高程序的可读性以及方便修改利率，将利率 RATE 定义为符号常量。程序如下：

```
1   /*    本息和.cpp    */
2   #include <stdio.h>                    //使用库函数 printf
3   const double RATE = 0.025;            //将利率定义为符号常量 RATE
4
5   int main( )
6   {
7     double interest, foundMoney, sumMoney;      //定义变量
8     foundMoney = 1000;          //变量 foundMoney 表示本金，赋值为 1000
9     interest = foundMoney * RATE;      //计算利息赋给变量 interest
10    sumMoney = foundMoney + interest * 0.8;     //计算本金和利息的和
11    printf("存入人民币%8.2f 元，", foundMoney);    //输出结果
12    printf("一年后本息和为%8.2f 元\n", sumMoney); //输出结果
13    return 0;                     //将 0 返回操作系统，表明程序正常结束
14  }
```

运行结果如下：

```
存入人民币 1000.00 元,一年后本息和为 1020.00 元
```

习 题 3

一、选择题

1. 下面 4 个选项中，均是合法整型常量的选项是（　　）。

   A. 160　–0xffff　011　　B. –0xcdf　01a　0xe

   C. –01　986,012　0668　　D. –0x48a　2e5　0x

2. 下面 4 个选项中，均是不合法转义字符的选项是（　　）。

   A. '\"'　'\\'　'\xf'　　B. '\1011'　'\'　'\ab'

   C. '\011'　'\f'　'\}'　　D. '\abc'　'\101'　'xlf'

3. 下面 4 个选项中，均是合法 char 型常量的选项是（　　）。

A. 'a'　'!'　'this'　　B. '"　'\"　'\ab'

C. '1'　'a'　'*'　　D. '\78'　'\76'　'\72'

4. 若有变量定义语句：char ch = '\72'；则变量 ch（　　）。

A. 包含 1 个字符　　B. 包含 2 个字符

C. 包含 3 个字符　　D. 变量定义不合法

5. 假设变量已正确定义并初始化，下面合法的赋值语句是（　　）。

A. a := b + 1;　　B. a = b = c + 2;　　C. 18.5 = a + b;　　D. a + 1 = c;

6. 假设变量已正确定义并初始化，下面合法的赋值语句是（　　）。

A. a == 1;　　B. i++;　　C. a + i = 5;　　D. 5 - i = a;

7. 下面正确的常量定义是（　　）。

A. #define base = 2.13　　B. #define base 1/3

C. #define int integer　　D. #define count 999;

8. 下面正确的变量定义是（　　）。

A. int a; b; c;　　B. double x1, x2;

C. char 'A', 'B';　　D. int a, double x;

二、简答题

1. 字符常量和字符串常量有什么区别？

2. 定义符号常量有什么好处？什么情况下应该将字面常量定义为符号常量？

3. C/C++语言为什么规定变量要“先定义，后使用”？变量定义的含义是什么？

4. C/C++语言为什么有转义字符的概念？请举例说明。

三、程序设计题

1. 已知底面半径和高，求圆柱体的体积。

2. 设 $P_1 = (x_1, y_1)$和 $P_2 = (x_2, y_2)$分别是二维空间的两个点，求 P_1 和 P_2 之间距离的平方。

3. 将给定的角度值转换为对应的弧度值。

4. 不用中间变量，交换 A 和 B 两个变量的值。

5. 在古希腊时代，宙斯叫来一个铁匠铸造一个环绕地球的铁环，要求铁环的直径正好与地球的直径相匹配。但可怜的铁匠出了点差错，他造出的铁环比原定的周长长出 1 米。宙斯让铁环环绕着地球，让它和地球的一个点相接触，如图 3.9 所示。现在的问题是：铁环在地球另一端的狭缝最大宽度是多少？

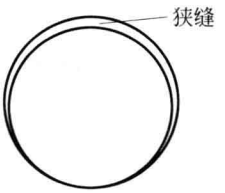


图 3.9　地球与铁环

第4章

数据的基本处理

从功能上讲，程序是解决某个问题的一组指令的集合。因此，程序应包括对数据的描述（即求解问题涉及的数据）和对数据处理的描述（即求解问题的具体步骤），数据处理的基本任务是对数据进行加工处理，也就是进行一系列的运算，计算机可实施的基本运算是算术运算和逻辑运算。数据处理一定会涉及数据的来源问题，程序应该能够对各种不同的数据进行处理，并按照用户要求以适当的格式输出处理结果，因此，程序设计语言应该提供灵活的输入输出操作来解决这一问题。

4.1 输 入 输 出

【任务 4.1】计算圆的面积（改进版）

【问题】 从键盘上输入圆的半径，求圆的面积。在任务 3.1 中，程序只能对给定的半径计算圆的面积，现要求程序能够根据键盘的输入值计算圆的面积。

【想法】 在程序 duty3-1 中，如果改变圆的半径必须修改程序。如果能在程序的运行过程中由用户指定半径值，就可以动态计算圆的面积。

【算法】 设变量 radius 和 area 分别表示圆的半径和面积，定义符号常量 PI 的值为 3.14，算法如下：

伪代码

```
step1: radius 接收从键盘输入的半径值;
step2: area = PI * radius * radius;
step3: 输出 area;
```

算法需要接收来自键盘的输入数据并存储到相应变量中，程序设计语言如何接收用户在程序的运行过程中提供的输入数据呢？

4.1.1 输入输出的概念

用计算机解决问题时，通常需要用户提供必要的数据信息，要求程序能够对不同的

数据进行处理，程序处理的结果需要以用户可以理解的形式输出，因此，数据的输入输出是程序的重要组成部分。所谓输入输出是以计算机主机为主体而言的，从计算机向外部输出设备（如显示器、打印机、磁盘等）输出数据称为**输出**，从外部输入设备（如键盘、鼠标、扫描仪、磁盘等）向计算机输入数据称为**输入**，如图 4.1 所示。

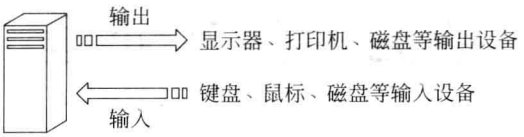


图 4.1　输入输出的概念

可以将输入输出分为两大类：一类是以终端为对象，即从终端（例如键盘和鼠标）输入数据，将运行结果输出到终端（例如显示器和打印机）上，标准的输入终端是键盘，标准的输出终端是显示器；一类是以除终端以外的外部介质为对象，在程序运行时，常常需要将一些数据（运行的最终结果或中间数据）输出到磁盘上保存起来，以后需要时再从磁盘中将数据输入到计算机中，因此，最常见的外部介质是磁盘。

4.1.2　格式化输入输出函数

C/C++语言没有提供输入输出语句，C/C++程序的输入输出操作是由系统提供的输入输出库函数实现的。由于输入输出牵涉到具体的计算机设备，把输入输出操作作为函数来处理，就可以使 C/C++语言本身的规模减小，编译程序简单，提高程序的可移植性。本节简要介绍格式化输入输出函数 scanf 和 printf，详细的标准输入输出函数将在 6.2.2 节中讨论。使用库函数 scanf 和 printf 需要包含头文件 stdio.h（standard input and output 的缩写）。

1. 格式化输出函数 printf

函数 printf 的一般调用格式为：

```
printf("格式控制", 输出列表);
```

其中，格式控制包含两类字符，一类是由%开头的格式控制符，用于将输出列表上的输出项进行格式转换；另一类是普通字符（除格式控制符外均是普通字符），直接输出。输出列表是要输出的数据，这些数据可以是常量、变量或表达式，且与格式控制符一一对应。

例如，将变量 ch 的值以字符形式输出，将变量 radius 的值以整数形式输出，将变量 area 的值以小数形式输出，并且该值的宽度为 6 位，其中小数占 2 位，格式化输出函数调用如下：

```
printf("ch = %c, radius = %d, area = %6.2f\n", ch, radius, area);
```

其中，ch =为普通字符，%c 表示将要输出一个字符型数据，“, radius =”为普通字符，%d 表示将要输出一个整型数据，“, area =”为普通字符，%f 表示将要输出一个实型数据，“%6.2f”表示该实型数据在屏幕上所占的宽度为 6 位，其中小数部分占 2 位，\n 为普通字符，其输出结果是将光标移到下一行的起始位置。假设变量 ch 的值为'a'，变量 radius 的值为 10，变量 area 的值为 314.5，则运行结果如下：

```
ch = a, radius = 10, area = 314.50
```

2. 格式化输入函数 scanf

函数 scanf 的一般调用格式为：

scanf（**"格式控制"，地址列表**）

其中，格式控制是由%开头的格式控制符；地址列表由逗号分隔的若干个变量地址组成，每个变量地址是在变量名的前面加上取地址运算符&，地址列表中的变量与格式控制符一一对应。

例如，假设 x 为 int 型变量，y 为 double 型变量，ch 为 char 型变量，从键盘上输入一个整数存储到变量 x 中，输入一个小数存储到变量 y 中，输入一个字符存储到变量 ch 中，格式化输入函数调用如下：

```
scanf("%d %lf %c", &x, &y, &ch);
```

其中，%d 表示将要输入一个整数，%lf 表示将要输入一个 double 型小数，%c 表示将要输入一个字符。假设从键盘上输入（□表示空格，< Enter >表示回车）：

```
25□8.5□a<Enter>
```

则变量 x 的值为 25，变量 y 的值为 8.5，变量 ch 的值为'a'。

例 4.1 从键盘上输入一个小写英文字母，将其转换为对应的大写英文字母。

解：由于在 ASCII 表中，小写英文字母的编码与对应大写英文字母的编码相差 32。例如，小写英文字母 a 的 ASCII 码为 97，大写英文字母 A 的 ASCII 码为 65。因此，可以直接将小写英文字母的编码减去 32 得到对应大写英文字母的编码，其转换关系如图 4.2 所示。

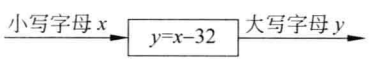


图 4.2 小写字母和大写字母之间的映射关系

【算法】 设变量 littleChar 存放小写英文字母，变量 upperChar 存放转换得到的大写英文字母，算法如下：

伪代码

```
step1: littleChar 接收从键盘输入的一个小写字母;
step2: upperChar = littleChar - 32;
step3: 输出 upperChar;
```

【程序】 程序如下：

```
/*    example4-1.cpp   */
#include <stdio.h>          //使用库函数 printf 和 scanf
                            //空行,以下是主函数
int main( )
{
  char littleChar, upperChar;
                            //定义字符型变量 littleChar 和 upperChar
  printf("请输入一个小写字母: ");                    //输出提示信息
  scanf("%c", &littleChar);
                            //注意变量 littleChar 前有取地址运算符"&"
  upperChar = littleChar - 32;
                            //将表达式的运算结果赋给变量 upperChar
  printf("小写字母%c 对应的大写字母为%c\n",littleChar,upperChar);
  return 0;                 //将 0 返回操作系统,表明程序正常结束
}
```

运行结果如下（下划线为用户输入）：

```
请输入一个小写字母: x
小写字母 x 对应的大写字母为: X
```

4.1.3 解决任务 4.1 的程序

```
/*   duty4-1.cpp   */
#include <stdio.h>                  //使用库函数 printf 和 scanf
const double PI = 3.14;             //定义符号常量 PI,注意末尾有分号
                                    //空行,以下是主函数
int main( )
{
  int radius;                       //定义 radius 为整型变量,存储半径
  double area;                      //定义 area 为实型变量,存储圆的面积
  printf("请输入圆的半径: ");          //输出提示信息
  scanf("%d", &radius);             //输入一个整数存放到变量 radius 中
  area = PI * radius *radius;       //将表达式的计算结果赋给变量 area
  printf("radius = %d\n", radius);
                                    //输出变量 radius 的值,'\n'表示换行
  printf("area = %6.2f\n", area);
                                    //输出数据的宽度为 6 位,其中小数占 2 位
  return 0;                         //将 0 返回操作系统,表明程序正常结束
}
```

运行结果如下（下划线为用户输入）：

```
请输入圆的半径: 10
radius = 10
area = 314.00
```

4.2 数据的基本运算

【任务 4.2】疯狂赛车

【问题】 4 个赛车手进行比赛，下面是 4 个评论员的评论：

A：2 号赛车一定能赢

B：4 号赛车一定能赢

C：3 号赛车一定不能赢

D：B 的评论是错误的

已知只有 1 位评论员是正确的，请问 2 号赛车是否能赢？

【想法】 设赢得胜利的赛车是 thisCar，将 4 位评论员的评论分别描述为逻辑表达式，统计 thisCar 为'2'时逻辑表达式成立的个数，若仅有一个表达式成立，则 2 号赛车能够赢得胜利，否则 2 号赛车不能赢得胜利。

【算法】 设变量 thisCar 表示赢得胜利的赛车，算法如下：

伪代码

```
step1: thisCar = '2';
step2: count = 逻辑表达式成立的个数;
step3: 如果 count 等于 1,则输出"2 号赛车能够赢得胜利!";
       否则输出"2 号赛车不能赢得胜利!";
```

为了解决现实世界的各种复杂问题，不但需要使用常量和变量来保存数据，还需要对这些数据进行各种运算处理，表达式是进行运算处理的基本构件，例如解决任务 4.2 的算法需要将各个评论描述为相应的逻辑表达式。事实上，问题求解的关键之一是将问题描述用程序设计语言表示成计算机能够实现的表达式。

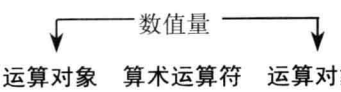
在不同的程序设计语言中，运算符的种类、数量、表示符号和求值方向一般有所不同。C/C++语言提供了 50 多个运算符，正是丰富的运算符和表达式使 C/C++语言的功能十分完善，这也是 C/C++语言的主要特点之一。本节介绍 C/C++语言中基本的算术运算和逻辑运算。

4.2.1 算术运算

所谓**算术运算**就是数值运算，由算术运算符、运算对象和圆括号组成的式子称为**算术表达式**。算术表达式的运算对象和运算结果均为数值型数据，一般可以是整型和实型。按照日常习惯，人们在进行算术运算时往往不区分运算对象是整型还是实型，为了照顾这种习惯，大多数语言允许在整型和实型之间进行混合算术运算。

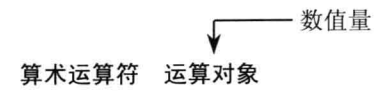
【语法】 算术表达式的一般形式如下：

二目运算：



或

一目运算：



其中，二目运算（也称双目运算）是指该运算符需要两个运算对象，一目运算（也称单目运算）是指该运算符需要一个运算对象；运算对象通常是数值量，可以是常量、变量，还可以是另一个表达式。C/C++语言的算术运算符如表 4.1 所示。

【语义】 执行算术运算符规定的算术运算。

表 4.1 C/C++语言的算术运算符

运算符	运算规则	运算类别	运算对象	运算结果	举 例
+	取正	一目	整型或实型	整型或实型	+5, +12.3
-	取负				-5, -12.3
+	加法	二目	整型或实型	若两个运算对象都是整型，则为整型，否则为实型	3 + 5, 3.5 + 8, x + 3
-	减法				3 - 5, 3.5 - 8, x - 3
*	乘法				3 * 8, 3.5 * 8, x * 8
/	除法				3 / 8, 3.5 / 8, x / 8
%	求余		整型	整型	3 % 8, 16 % 8, x % 8

例 4.2 写出下列数学表达式对应的算术表达式：

（1）$2x + 5y$

（2）$x^2 + 2x + 1$

（3）$s(s-a)(s-b)(s-c)$

解：对应的算术表达式如下：

（1）2 * x + 5 * y

（2）x * x + 2 * x + 1

（3）s * (s-a) * (s-b) * (s-c)

在 C/C++语言中进行算术运算需要注意以下问题：

① 两个整数相除的结果为整数，舍去小数部分，例如，8/3 的结果为 2。但是，如果有一个运算对象为负值，则舍去的方向不确定。例如，对于-8/3，有的系统给出的结果为-2，有的系统给出的结果为-3。大多数系统采取“向零靠拢”的方法，即取整后的结果向零靠拢，则-8 / 3 的结果为-2。

② 求余运算的运算对象均为整数，不能用于实数，例如，8%3 = 2。但是，如果运算对象为负值，则求余运算的结果不确定。例如，对于-8%3，有的系统给出的结果为-2，有的系统给出的结果为 1。

③ 实数的运算误差。尽量不要将一个很大的数和一个很小的数直接相加或相减，有些编译器可能会“丢失”这个很小的数；尽量不要将实数进行比较大小，实数的比较结果不可靠。

4.2.2 逻辑运算

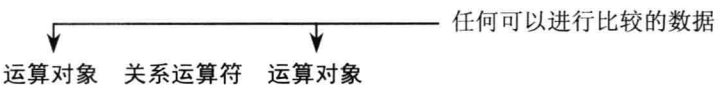
逻辑思维最基本的功能就是能够分辨各种情况，这些各种可能的情况称为逻辑条件。通常采用关系运算和逻辑运算来表达逻辑条件，关系运算可以看作是逻辑运算的简单形式，较复杂的逻辑条件就需要采用逻辑运算来表示。

1. 关系运算

所谓**关系运算**就是比较运算，也就是将两个值进行比较，比较的结果取决于编译系统采用的字符集，常用的字符集是 ASCII 码。例如，对于 a > 5，如果 a = 8 则满足 a > 5（即 a > 5 成立），如果 a = 3 则不满足 a > 5（即 a > 5 不成立）。

由关系运算符、运算对象和圆括号组成的式子称为**关系表达式**。关系表达式的运算对象是任何可以进行比较的数据，在 C/C++语言中，可以是整型、实型或字符型；关系表达式的运算结果是逻辑值，即只有逻辑真（True）和逻辑假（False）两个值，C/C++语言规定 0 表示逻辑假，非 0 表示逻辑真。

【语法】 关系表达式的一般形式如下：



其中，运算对象可以是常量、变量，还可以是另一个合法的表达式；C/C++语言的关系运算符如表 4.2 所示。

【语义】 执行关系运算符定义的比较运算，若满足该关系表达式，则结果为逻辑真，否则结果为逻辑假。

表 4.2 C/C++语言的关系运算符

运算符	运算规则	运算类型	运算对象	运算结果	举 例
<	小于	二目	整型、实型或字符型	满足则为真(1) 不满足则为假(0)	5 < 3, x < y
<=	小于等于				3<=3, x<=2
>	大于				5>3, x>y
>=	大于等于				3>=3, 2>=x
==	等于				3 == 3, x == 2
!=	不等于				3 != 3, x != 2

例 4.3 将任务 4.2 的各个评论描述为关系表达式，并计算当 2 号赛车能赢时，各关系表达式的计算结果。

解：设变量 thisCar 为字符型，则各个评论对应的关系表达式如下：

A：thisCar == '2'

B：thisCar == '4'

C：thisCar != '3'

D：thisCar != '4'

当变量 thisCar 的值为'2'时，则各个关系表达式的计算结果如下：

A：thisCar == '2'，即'2' == '2'，其结果为 1，即逻辑真；

B：thisCar == '4'，即'2' == '4'，其结果为 0，即逻辑假；

C：thisCar != '3'，即'2' != '3'，其结果为 1，即逻辑真；

D：thisCar != '4'，即'2' != '2'，其结果为 0，即逻辑假；

2. 逻辑运算

由逻辑运算符、运算对象和圆括号组成的式子称为**逻辑表达式**。有些语言规定参与逻辑运算的运算对象必须是逻辑型，C/C++语言规定参与逻辑运算的运算对象可以是整型、实型和字符型；逻辑表达式的运算结果是逻辑值，即只有逻辑真和逻辑假两个值。

【语法】 逻辑表达式的一般形式如下：

二目运算：

运算对象　逻辑运算符　运算对象

或

一目运算：

逻辑运算符　运算对象

其中，运算对象可以是常量、变量，还可以是另一个合法的表达式；C/C++语言的逻辑运算符如表 4.3 所示。

【语义】 执行逻辑运算符规定的逻辑运算。

表 4.3　C/C++语言的逻辑运算符

运算符	运算规则	运算类型	运算对象	运算结果	举　例
&&	逻辑与	二目	整型、实型或字符型	满足则为真(1) 不满足则为假(0)	5 && 2, x && y
||	逻辑或				5 || 2, x || y
!	逻辑非	一目			!2, !x

C/C++语言将所有非 0 值作为逻辑真，将 0 值作为逻辑假，逻辑运算规则如下：

① 如果运算对象 1 和运算对象 2 都是非 0 值，则“运算对象 1 && 运算对象 2”的运算结果为逻辑真；

② 如果运算对象 1 和运算对象 2 至少有一个是非 0 值，则“运算对象 1 || 运算对象 2”的运算结果为逻辑真；

③ 如果运算对象是 0 值，则“!运算对象”的运算结果为逻辑真；

④ 所有其他情况，运算结果均为逻辑假。

例 4.4　写出下列逻辑条件对应的逻辑表达式：

（1）小于 100 的偶数。

（2）x 大于 3 并且小于 8。

（3）闰年的判定条件是：① 能被 4 整除，但不能被 100 整除的年份；② 能被 400 整除的年份。

解：对应的逻辑表达式如下：

（1）设变量 x 表示一个整数，则逻辑表达式为：(x < 100) && (x % 2 == 0)。

（2）对应的数学表达式为 3 < x < 8，但在程序设计语言中，连续的关系运算一般需要用逻辑运算来表示，因此，逻辑表达式为：(3 < x) && (x < 8)。

（3）设变量 year 表示年份，则条件①可以表示为(year % 4 == 0 && year %100 != 0)，条件②可以表示为(year % 400 == 0)，年份 year 只要满足条件①和②之一即是闰年，因此，闰年的判定条件是：(year % 4 == 0 && year %100 != 0) || (year % 400 == 0)。

4.2.3 运算符的优先级和结合性

每种运算符都具有确定的优先级和结合性。**运算符的优先级**是指在同一表达式中具有不同的运算符时，表达式的求值顺序，优先级高的先于优先级低的进行运算；**运算符的结合性**是指在运算符的优先级相同时，表达式的求值方向，即运算对象按什么顺序进行运算，通常有两种求值方向：左结合和右结合，左结合是将运算对象用左侧的运算符进行运算，右结合是将运算对象用右侧的运算符进行运算。

表达式的求值过程依据运算符的运算规则以及规定的优先级和结合性来进行。在 C/C++语言中，算术运算符（+、–、*、/、%）、关系运算符（>、>=、<、<=、==、!=）和逻辑运算符（&&、||、!）的优先级和结合性如表 4.4 所示。

表 4.4　运算符的优先级和结合性

运　算　符	运 算 规 则	结　合　性	优　先　级
!	逻辑非	左结合	高 ↑ 低
+、–	取正、取负	右结合	
*、/、%	乘、除、取余	左结合	
+、–	加、减		
<、<=、>、>=	小于、小于等于、大于、大于等于		
==、!=	等于、不等于		
&&、||	逻辑与、逻辑或		

例 4.5　假设 x =10，y = 20，计算 38 + 5 > x + y && 38 % 5 >= y 的值。

解：表达式的求值过程如图 4.3 所示，具体过程如下：

（1）计算 38 + 5 得到结果为 43；

（2）计算 x + y 得到结果为 30；

（3）计算 43（38 + 5 的结果）> 30（x + y 的结果）得到结果为逻辑值 1；

（4）计算 38 % 5 得到结果为 3；

（5）计算 3（38 % 5 的结果）>= y 得到结果为逻辑值 0；

（6）计算 1（43 > 30 的结果）&& 0（3 >= y 的结果）得到逻辑值 0。

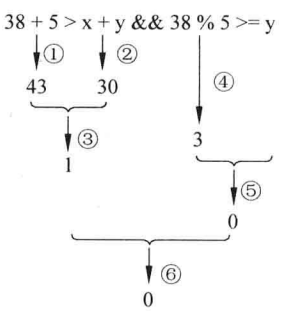


图 4.3　表达式的求值过程

良好的编程习惯 4.1

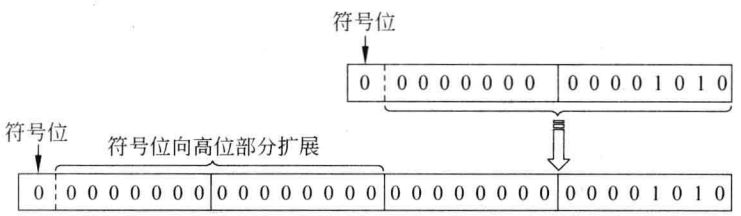
由于表达式、运算符以及运算符优先级的复杂性，就是熟练的程序员也难免出错，程序中应该避免使用晦涩并且容易出错的表达式。可以在容易出错或不确定的地方按照自己的本意给表达式适当加括号，也可以将复杂的表达式拆分为多个子表达式，然后分别计算子表达式的值。

4.2.4 运算对象的类型转换

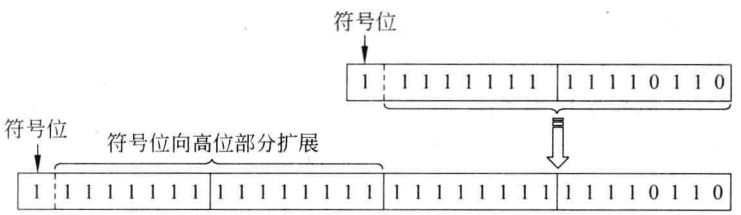
计算机硬件在进行表达式计算时，通常要求操作数占用相同的二进制位数，并且要求存储方式也相同。例如，计算机硬件可以直接将两个 16 位整数相加，但是不能直接将 16 位整数和 32 位整数相加，也不能直接将 32 位整数和 32 位浮点数相加。因此，在表达式中，如果一个二目运算符两侧的运算对象的数据类型不同，则应该按照“先转换，后运算”的原则，首先将运算对象转换成同一数据类型，然后基于同一数据类型进行运算。数据类型转换有两种方式：自动转换和强制转换。

1. 自动转换

自动类型转换由编译程序完成而无需编程人员的干预，因此也称为**隐式类型转换**，一般的转换原则是将短长度的数据类型（占用较少的二进制位）转换为长长度的数据类型（占用较多的二进制位），这称为**类型提升**。例如，16 位 short int 型数据转换为 32 位 int 型数据的过程示例如图 4.4 所示。



(a) short int型的 10转换为 int型



(b) short int型的 −10转换为 int型

图 4.4　short int 型转换为 int 型的过程示例

需要强调的是，类型提升通常是一次完成的，例如，int 型数据可以直接转换为 double 型，char 型数据可以直接转换为 int 型也可以直接转换为 double 型。类型提升后只是数

据的存储方式发生了变化，数据值并不发生变化，因此，这种转换是安全的。

例 4.6 假设有如下变量定义，计算 ch % a + (x + y) / a 的值。

```
int   a = 10;
char  ch = 'a';
float  x = 2.5;
double  y = 123.45;
```

解：运算对象的类型转换及表达式的求值过程如图 4.5 所示，计算过程如下：

① 计算 ch % a：将变量 ch 由 char 型转换为 int 型再进行计算，即 97 % 10 得到结果为 7。

② 计算(x+y)：将变量 x 由 float 型转换为 double 型再进行计算，即 2.5 + 123.45 得到结果为 125.95。

③ 计算 125.95 / a：将变量 a 由 int 型转换为 double 型再进行计算，即 125.95 / 10.0 得到结果为 12.595。

④ 计算 7+12.595：将 7 由 int 型转换为 double 型再进行计算，即 7.0 + 12.595 得到结果为 19.595。

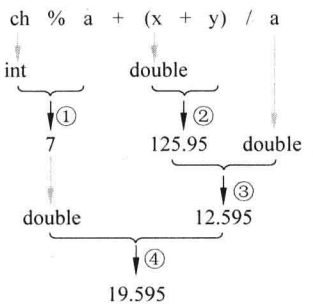


图 4.5 运算对象的类型转换及表达式的求值过程

将长长度的数据类型转换为短长度的数据类型，一般直接截取长长度数据类型的低位部分，因此，有可能使数据值发生变化，有些编译器会发出警告。例如，将 32 位 int 型数据转换为 16 位 short int 型数据的过程示例如图 4.6 所示。

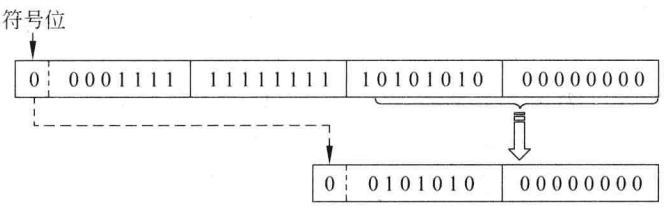
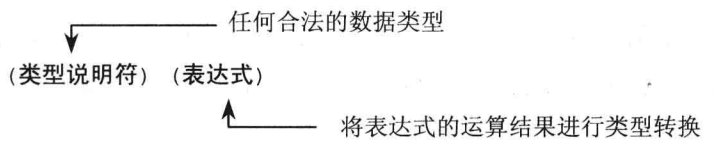


图 4.6 int 型的 0xfffaa00 转换为 short int 型后的值是 0x2a00

2. 强制转换

自动类型转换的规则同样也适用于强制类型转换，通常强制类型转换是编程人员有意识地改变某个数据的类型，因此也称为**显式类型转换**。

【语法】 强制类型转换的一般形式如下：



其中，“( )”是强制类型转换符；类型说明符和表达式都必须加括号，如果表达式是变量可以不加括号。

【语义】 将表达式的值转换为类型说明符表示的类型。

例 4.7 两个小数进行求和运算，将运算结果存放在一个整型变量中。

解：设变量 x 和 y 分别存储两个小数，变量 a 存储得到的整数值，如果对不同的运算对象进行强制类型转换，其结果有可能不同，程序如下：

```
/*   example4-7.cpp   */
#include <stdio.h>                    //使用库函数 printf
                                      //空行,以下是主函数
int main( )
{
  float x = 3.6, y = 3.8;             //定义并初始化 float 型变量 x 和 y
  int a;                              //定义 int 型变量 a
  a = x + y;                          //系统进行自动类型转换
  printf("a = %d\n", a);              //输出变量 a 的值
  a = (int)(x + y);                   //将加法运算的结果进行强制类型转换
  printf("a = %d\n", a);              //输出变量 a 的值
  a = (int)x + y;                     //将 x 进行强制类型转换再执行加法运算
  printf("a = %d\n", a);              //输出变量 a 的值
  return 0;                           //将 0 返回操作系统,表明程序正常结束
}
```

运行结果如下：

```
a = 7
a = 7
a = 6
```

如果把一个长长度类型的数据转换为一个短长度类型的数据，通常会有潜在的损失，编译器会给出一个警告。而强制类型转换告诉编译器："不必进行类型检查，我知道并且不关心这个潜在的损失，我知道转换带来的后果。"编译器则会关闭这个警告，也就是说，在程序中引入了一个漏洞，并阻止编译器报告这个漏洞产生的问题。更糟糕的是，编译器会认同这个漏洞，而不执行任何检查。一旦进行了强制类型转换，程序员必须自己面对各种问题。

良好的编程习惯 4.2

由于强制类型转换会在程序中引入漏洞，一般情况下，尽可能避免使用强制类型转换，它只是用于解决非常问题的特殊手段。

需要强调的是，无论是自动转换还是强制转换，都只是为了执行本次运算而对数据的类型进行临时性的转换，是将某个数据的值转换成一个中间量，不改变原数据的类型。例如，在表达式 a = (int) x + y 中，只是为了执行本次运算将变量 x 的值转换成一个 int 型的临时量，变量 x 的数据类型仍然是 float 型。

4.2.5 解决任务 4.2 的程序

```
/*    duty4-2.cpp    */
#include <stdio.h>                          //使用库函数 printf
                                            //空行,以下是主函数
int main( )
{
  char thisCar = '2';                       //thisCar 表示赛车编号,为字符型
  int count;          //定义 int 型变量 count 表示逻辑表达式成立的个数
  count = (thisCar == '2') + (thisCar =='4') + (thisCar != '3')
  +(thisCar != '4');
  if (count == 1)                           //逻辑表达式只有一个成立
     printf("2 号赛车能够赢得胜利! \n");     //输出结果,然后换行
  else
    printf("2 号赛车不能赢得胜利! \n");      //输出结果,然后换行
  return 0;                                 //将 0 返回操作系统,表明程序正常结束
}
```

运行结果如下：

```
2 号赛车不能赢得胜利!
```

4.3 程序设计实例

4.3.1 实例 1——华氏温度转换为摄氏温度（改进版）

【问题】 从键盘上输入华氏温度，将其转换为对应的摄氏温度。在 3.4.1 节中实现了将给定的华氏温度转换为摄氏温度，现要求程序能够根据键盘的输入值进行转换。

【想法】 在程序的运行过程中由用户输入华氏温度，就可以进行相应的转换。

【算法】 设变量 temC 表示摄氏温度，变量 temF 表示华氏温度，算法如下：

伪代码

```
step1: temF 接收从键盘输入的华氏温度值;
step2: temC = 5 * (temF - 32) / 9;
step3: 输出 temC;
```

【程序】

```
/*   温度转换（改进版）.cpp   */
#include <stdio.h>                          //使用库函数 printf 和 scanf
                                            //空行,以下是主函数
int main( )
{
  double temC, temF;                        //定义 double 型变量 temC 和 temF
```

```
7       printf("请输入华氏温度: "); //输出提示信息
8       scanf("%lf", &temF);        //从键盘输入 double 型实数赋给变量 tempF
9       temC = 5 * (temF - 32)/ 9;     //将表达式的计算结果赋给变量 temC
10      printf("华氏温度%6.2f 对应的摄氏温度是%6.2f",temF, temC);
11      return 0;                   //将 0 返回操作系统,表明程序正常结束
12  }
```

运行结果如下（下划线为用户输入）：

```
请输入华氏温度: 100
华氏温度 100.00 对应的摄氏温度是 37.78
```

4.3.2 实例 2——通用产品代码 UPC

【问题】 通用产品代码（Universal Product Code，UPC）是指商品上条形码下方的数字，每个 UPC 表示为一个 12 位的数，超市可以通过扫描商品上的 UPC 条形码获得该商品的生产商以及价格等信息。UPC 中最后一位是校验位，用来校验条码扫描的结果是否正确。假设计算校验位的方法是：

（1）将第 1 位、第 3 位、第 5 位、第 7 位、第 9 位和第 11 位的数字相加；

（2）将第 2 位、第 4 位、第 6 位、第 8 位和第 10 位的数字相加；

（3）将奇数位数字相加的结果乘以 3 再减去偶数位数字相加的结果；

（4）将减法的结果除以 10 再取余数。

从键盘输入一个 11 位的数字，用上述方法计算这 11 位数字的校验位，形成 12 位的 UPC 并输出。

【想法】 由于一个 11 位的数字超出 long int 型的表示范围，所以，不能用一个整型变量直接存储这个 UPC 码。可以用 11 个变量（upc1~upc11）来接收每一位数字，然后对这 11 个变量进行相应的算术运算，将运算结果存储到变量 upc12 中，最后依次输出变量 upc1~upc12 的值。

【算法】 设变量 upc1~upc12 表示 UPC 的每一位数字，sum1 和 sum2 分别对应奇数位数字之和以及偶数位数字之和，算法如下：

伪代码

```
step1: 输入一个 11 位数,将每一位数字存储到变量 upc1~upc11 中;
step2: sum1 = upc1 + upc3 + upc5 + upc7 + upc9 + upc11;
step3: sum2 = upc2 + upc4 + upc6 + upc8 + upc10;
step4: upc12 = (sum1 * 3 - sum2) % 10;
step5: 依次输出 upc1~upc12;
```

【程序】 程序如下：

```
1   /*   UPC.cpp   */
2   #include <stdio.h>                    //使用库函数 printf 和 scanf
3                                         //空行,以下是主函数
```

```
int main( )
{
  int upc1, upc2, upc3, upc4, upc5, upc6, upc7, upc8, upc9, upc10,
  upc11, upc12;
  int sum1, sum2;
  printf("请依次输入 UPC 码的前 6 位: ");          //输出提示信息
  scanf("%d %d %d %d %d %d", &upc1, &upc2, &upc3, &upc4, &upc5,
  &upc6);
  printf("请依次输入 UPC 码的后 5 位: ");
                                        //输出提示信息
  scanf("%d %d %d %d %d", &upc7, &upc8, &upc9, &upc10, &upc11);
  sum1 = upc1 + upc3 + upc5 + upc7 + upc9 + upc11;
                                        //奇数位数字相加赋给 sum1
  sum2 = upc2 + upc4 + upc6 + upc8 + upc10;
                                        //偶数位数字相加赋给 sum2
  upc12 = (sum1 * 3 - sum2) % 10;           //计算 upc12
  printf("UPC 码是%d%d%d%d%d%d%d%d%d%d%d%d\n",upc1,upc2,upc3,upc4,
  upc5,upc6,upc7,upc8,upc9,upc10,upc11);

  printf("校验码是%d\n", upc12);
  return 0;                             //将 0 返回操作系统,表明程序正常结束
}
```

运行结果如下（下划线为用户输入）：

```
请依次输入 UPC 码的前 6 位：6 8 0 5 0 9
请依次输入 UPC 码的后 5 位：0 0 1 3 6
UPC 码是 68050900136
校验码是 4
```

习 题 4

一、选择题

1. 对于变量定义 int a = 7; float x = 2.5, y = 4.7;，表达式 x + a % 3 + (int)(x + y) % 2 / 4 的值是（　　）。

A. 2.5　　B. 2.75　　C. 3.5　　D. 0

2. 设变量 a 是整型，f 是单精度型，d 是双精度型，则表达式 a * f + (d – a)的运算结果的数据类型为（　　）。

A. int　　B. float　　C. double　　D. 不确定

3. 假设变量 x 和 y 为 double 型，且 x 已赋值为 2，则表达式 y = x + 3 / 2 的值是(　　)。

A. 3.5　　B. 3　　C. 2.0　　D. 3.0

4. 下列运算符中优先级最高的是（　　）。

A. <　　B. +　　C. &&　　D. !=

5. 判断 char 型变量 ch 是否为小写字母的正确表达式是（　　）。

A. 'a' <= ch <= 'z'　　B. (ch >= a) && (ch <= z)

C. ('a' >= ch) || ('z' <= ch)　　D. (ch >= 'a') && (ch <= 'z')

6. 语句 printf("I\bandy\\\bou\n");的输出结果是（　　）。

A. I\bandy\\\bou\n　　B. I\bandy\\\bou

C. Iandy\ou　　D. andyou

7. 若有变量定义 int a; float b; 则以下输入语句正确的是（　　）。

A. scanf("%f %f", &a, &b);　　B. scanf("%lf %lf", &a, &b);

C. scanf("%d %f", &a, &b);　　D. scanf("%d %6.2f",&a, &b);

8. 执行下面程序段，给变量 x 和 y 赋值时，不能作为数据分隔符的是（　　）。

```
int x, y;
scanf("%d %d", &x, &y);
```

A. 空格　　B. Tab 键　　C. 回车　　D. 逗号

9. 执行下面程序段，假设用户输入为 1□□2□3（□表示空格），则变量 ch1，ch2 和 ch3 的值为（　　）。

```
char ch1, ch2, ch3;
scanf("%c%c%c", &ch1, &ch2, &ch3);
```

A. '1', '2', '3'　　B. '1','□', '2'　　C. '1','□', '□'　　D. '1','□', '3'

二、程序设计题

1. 输入一个三位数，将其加密后输出。加密方法是对该数的每一位数字，将其加 6 后除以 10 取余数，作为该位上的新数字，再交换个位数字和百位数字。

2. 某市出租车 3 公里的起价为 8 元，3 公里以外按 1.5 元/公里计费。要求：输入行车里程数，输出应付车费。

3. 在平面上，穿过两个点(x_1, y_1)和(x_2, y_2)的直线是由下面的方程定义的：

$$ax + by = c \quad (\text{其中，} a = y_2 - y_1, b = x_1 - x_2, c = x_1y_2 - y_1x_2)$$

这样一条直线把平面分成两个半平面：其中一个半平面中的点都满足 $ax + by > c$，另一个半平面中的点都满足 $ax + by < c$。对于任意两个点 P_1 和 P_2，判断 P_1 和 P_2 是否位于这条直线的同一边。

4. 两个城市 *A* 和 *B* 相距 400 公里，两列火车同时相向开出，一列火车以每小时 40 公里的速度从城市 *A* 驶向城市 *B*，另一列火车以每小时 60 公里的速度从城市 *B* 驶向城市 *A*。同时，有一只忙碌的小蜜蜂在第一列火车的前面开始了它有趣的旅行，它以每小时 75 公里的速度向着第二列火车飞行，遇到第二列火车就立即逆转方向飞，以后它遇到火车就这样逆转方向飞。小蜜蜂一直这样飞行直到两列火车相遇，请问小蜜蜂一共飞行了多长的距离？

第5章

程序的基本控制结构

从行文的角度来说，程序是顺序书写的，但程序的执行却不一定是顺序的。在程序的执行过程中，应该能够根据运行时刻的状态有选择地执行某些程序段或反复执行某些程序段。已经证明，任何程序都可以由顺序、选择和循环这三种基本控制结构组成，因此，灵活掌握这三种基本控制结构是编写程序的重要基础。

5.1 顺 序 结 构

【任务 5.1】整数的逆值

【问题】 给定一个两位数，将这个两位数的个位和十位颠倒得到其逆值（也称逆向值），例如 85 的逆值是 58。

【想法】 设这个两位数的个位数字是 x，十位数字为 y，则该数的逆值为 x×10 + y。首先需要分离出这个两位数的个位和十位，分离的方法是整除和求余。例如，通过对 85 除 10 取整数部分得到十位数字 8，通过对 85 除 10 取余数部分得到个位数字 5。

【算法】 设变量 num 表示一个两位数，变量 x 表示整数 num 的个位数字，变量 y 表示整数 num 的十位数字，变量 numDevo 表示整数 num 的逆值，算法如下：

伪代码

```
step1: x = num % 10;
step2: y = num / 10;
step3: numDevo = x * 10 + y;
step4: 输出 numDevo;
```

程序设计中经常遇到这种情况：对于算法的某个部分，若执行，则顺序执行每一条指令，若不执行，则一条指令也不执行，这部分在逻辑上是一个整体。大多数语言提供了顺序结构来处理这种情况。

5.1.1 复合语句实现顺序结构

所谓**顺序结构**是由若干条语句组成的语句块，从执行过程上看，按照位置的先后顺序依次执行每一条语句。

在 C/C++语言中，顺序结构由复合语句实现，**复合语句**由一对花括号括起来的若干条语句组成。从逻辑上讲，复合语句是一个整体，可以将它看成是一条语句，可以放在能够使用语句的任何地方。

【语法】 复合语句的一般形式如下：

```
{                          ← 可以是任何语句
  语句 1
  语句 2
    ⋮
  语句 n
}                          ← 没有分号
```

其中，语句是满足 C/C++语言语法规定的任何语句。注意右花括号后面没有分号。

【语义】 依次执行花括号中的每条语句，其执行流程如图 5.1 所示。

例 5.1 交换两个变量的值。

解：交换两个变量的值可以借助一个中间变量，假设两个变量为 x 和 y，中间变量为 temp，交换过程如图 5.2 所示。

图 5.1 复合语句的执行流程

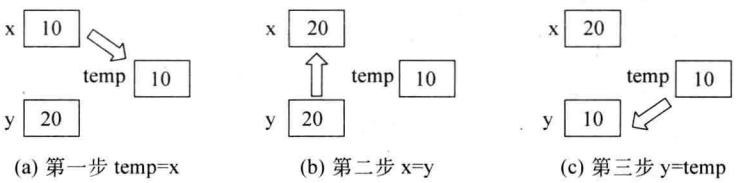


图 5.2 交换两个变量值的过程

【程序】 首先用变量 x 和 y 接收从键盘输入的两个整数，然后用复合语句实现交换变量 x 和 y 的操作。为了比较交换前后的结果，在执行交换之前和之后分别显示变量 x 和 y 的值。程序如下：

```
1   /*   example5-1.cpp   */
2   #include <stdio.h>                  //使用库函数 printf 和 scanf
3                                       //空行,下面是主函数
4   int main( )
5   {
6     int x, y, temp;                   //temp 为暂存变量
7     printf("请输入两个整数: ");        //输出提示信息
8     scanf("%d%d", &x, &y);            //从键盘接收两个整数
9     printf("这两个整数是: %5d%5d\n", x, y);   //x 和 y 的输出宽度均占 5 位
```

```
    {                                          //复合语句实现交换变量 x 和 y 的值
      temp = x;
      x = y;
      y = temp;
    }
    printf("交换后的值为：%5d%5d\n", x, y);  //交换后 x 和 y 的值发生改变
    return 0;                                  //将 0 返回操作系统,表明程序正常结束
}
```

运行结果如下（下划线为用户输入）：

```
请输入两个整数：10  20
这两个整数是：    10   20
交换后的值为：    20   10
```

在程序 example5-1.cpp 中，主函数中的所有语句均是顺序执行的，复合语句的花括号也可以省略。

5.1.2 解决任务 5.1 的程序

```
/*    duty5-1.cpp   */
#include <stdio.h>                    //使用库函数 printf 和 scanf
                                      //空行，下面是主函数
int main( )
{
  int num, x, y, numDevo;
  printf("请输入一个两位数：");       //输出提示信息
  scanf("%d", &num);                  //从键盘接收一个整数
  x = num % 10;                       //x 存储 num 的个位数字
  y = num / 10;                       //y 存储 num 的十位数字
  numDevo = x * 10 + y;               //交换个位数字和十位数字得到 numDevo
  printf("%2d 的逆值为：%2d\n", num, numDevo);
  return 0;                           //将 0 返回操作系统,表明程序正常结束
}
```

运行结果如下（下划线为用户输入）：

```
请输入一个两位数：35
35 的逆值为：53
```

5.2 选 择 结 构

【任务 5.2】水仙花数

【问题】 水仙花数是指一个三位数，其各位数字的立方和等于该数本身。例如，153 $= 1^3 + 5^3 + 3^3$，则 153 是一个水仙花数。要求从键盘上输入一个三位数，判断该数是否是

水仙花数。

【想法】 首先将这个三位数的个位、十位和百位数字分离出来，然后判断各位数字的立方和是否等于该数。分离出各位数字的方法是整除和求余。例如，通过对 153 除 10 取余数部分得到个位数字：153 % 10 = 3；通过对 153 除 10 取整数部分得到高两位：153 / 10 = 15；通过对 15 除 10 取余数部分得到十位数字：15 % 10 = 5；通过对 15 除 10 取整数部分得到百位数字：15 / 10 = 1。

【算法】 设变量 x 存储一个三位数，变量 x1、x2 和 x3 分别存储 x 的个位、十位和百位数字，算法如下：

伪代码

```
step1: x1 = x % 10;  y = x / 10;
step2: x2 = y % 10;
step3: x3 = y / 10;
step4: 如果 x1 * x1 * x1 + x2 * x2 * x2 + x3 * x3 * x3 等于 x,则该数是水
       仙花数;否则该数不是水仙花数;
```

程序是为了解决某个实际问题而设计的，而问题的处理往往包括多种情况，不同的情况需要进行不同的处理，几乎所有的程序设计语言都提供了选择结构来完成这种选择性处理。选择结构包括单分支、双分支和多分支等结构，单分支和双分支的选择结构一般由逻辑值控制，根据给定的条件成立与否决定是否执行某个程序段；多分支的选择结构一般由算术值控制，根据表达式的值选择执行某个程序段。

5.2.1 逻辑值控制的选择结构

1. 单分支的选择结构

单分支选择结构的基本特点是：程序的流程由一个分支构成，且只有满足给定的条件时才执行这个分支。单分支的选择结构一般由 if 语句实现。

【语法】 if 语句的一般形式如下：

```
        ┌──── 判断条件
if (表达式) ←──── 必须有括号
    语句
```

其中，表达式一般是逻辑表达式，注意括号不能省略；语句是任何满足 C/C++语言语法规则的语句。

【语义】 计算表达式的值；当表达式的值为真时执行语句；否则顺序执行 if 语句的下一条语句。if 语句的执行流程如图 5.3 所示。

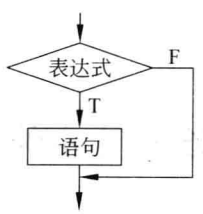


图 5.3 if 语句的执行流程

例 5.2 求两个整数中的较大值。

解：设整数 x 和 y 的较大值为 max，可以先假定 x 较大，再判断 max 是否小于 y，如果 max 小于 y，则较大值为 y。

【程序】 首先用变量 x 和 y 接收从键盘输入的两个整数，然后用 if 语句实现比较。程序如下：

```
/*   example5-2.cpp   */
#include <stdio.h>                          //使用库函数 printf 和 scanf
                                            //空行,下面是主函数
int main( )
{
  int x, y, max;
  printf("请输入两个整数: ");               //输出提示信息
  scanf("%d%d", &x, &y);                    //从键盘接收两个整数
  max = x;                                  //假设 x 的值较大
  if (max < y)                              //如果 max(即 x)小于 y
     max = y;                               //则较大值是 y
  printf("%d 和%d 的较大值是: %d\n", x, y, max);
  return 0;                                 //将 0 返回操作系统,表明程序正常结束
}
```

运行结果如下（下划线为用户输入）：

```
请输入两个整数：15  25
15 和 25 的较大值是：25
```

2. 双分支的选择结构

双分支选择结构的基本特点是：程序的流程由两个分支构成，在程序的一次执行过程中，根据给定的条件是否成立决定执行哪一个分支。双分支选择结构一般由 if-else 语句实现。

【语法】 if-else 语句的一般形式如下：

```
          ←——————  判断条件
if (表达式) ←——————  必须有括号
    语句 1  ←——————  表达式成立执行
else
    语句 2  ←——————  表达式不成立执行
```

其中，表达式一般是逻辑表达式，注意括号不能省略；语句 1 和语句 2 是任何满足 C/C++ 语言语法规则的语句。

【语义】 计算表达式的值；当表达式的值为真时执行语句 1，否则执行语句 2。if-else 语句的执行流程如图 5.4 所示。

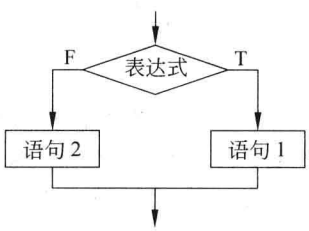


图 5.4 if-else 语句的执行流程

例 5.3 求两个整数中的较大值。

解：设整数 x 和 y 的较大值为 max，可以直接将 x 和 y 进行比较，如果 x 大于等于 y，则较大值为 x，否则，较大值为 y。

【程序】 首先用变量 x 和 y 接收从键盘输入的两个整数，然后用 if-else 语句实现比较。程序如下：

```
/*   example5-3.cpp   */
#include <stdio.h>                    //使用库函数 printf 和 scanf
                                      //空行,下面是主函数
int main( )
{
  int x, y, max;
  printf("请输入两个整数: ");          //输出提示信息
  scanf("%d%d", &x, &y);              //从键盘接收两个整数
  if (x >= y)                         //如果 x 大于或等于 y
    max = x;                          //则较大值是 x
  else
    max = y;                          //否则较大值是 y
  printf("%d 和%d 的较大值是: %d\n", x, y, max);
  return 0;                           //将 0 返回操作系统，表明程序正常结束
}
```

运行结果与例 5.2 相同。

3. 分支结构的嵌套

从语法上看，分支结构中的语句可以是任何语句，包括复合语句、分支语句、循环语句以及其他各类语句。如果分支结构中的语句又是一个分支语句，则构成分支结构的嵌套。如果嵌套在 if 语句中的语句是 if-else 语句，或嵌套在 if-else 语句中的语句是 if 语句，则会出现多个 if 和多个 else 重叠并且个数不等的情况，这时要注意 if 和 else 的配对问题。例如，假设 x = 20，y = 10，z = 15，在如下语句中，else 与哪个 if 配对（即 else 属于哪个 if 语句）？

```
if (x > y) if (y > z) x = 0; else x = 1;
```

可以有两种解释，第一种解释：

```
if (x > y)
    if (y > z) x = 0;
    else x = 1;
```

第二种解释：

```
if (x > y)
{
    if (y > z) x = 0;
}
else
     x = 1;
```

显然，两种解释的执行结果是不同的，第一种解释 else 与第 2 个 if 配对，执行结果 x 的值为 1，第二种解释 else 与第 1 个 if 配对，执行结果 x 的值仍为 20，这就产生了二义性。为了解决这个二义性问题，大多数语言都规定：else 与其前面最近的尚未配对的 if 相配对，如图 5.5 所示。

例 5.4 编写程序实现数学中的符号函数：

$$\text{sign}(x)=\begin{cases}1, & x>0\\ 0, & x=0\\ -1, & x<0\end{cases}$$

解：如果 x 大于 0，则 sign(x)的值为 1；否则 x 有可能小于或等于 0，进一步判断，如果 x 小于 0，则 sign(x)的值为–1；否则 x 一定等于 0，则 sign(x)的值为 0。

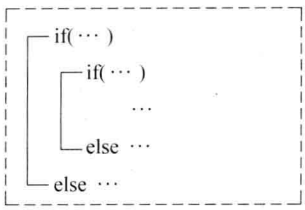


图 5.5 else 与 if 的配对原则

【程序】 设变量 sign 表示 sign(x)的值，主函数首先用变量 x 接收从键盘输入的整数，然后用嵌套的 if-else 语句实现比较。程序如下：

```
/*  example5-4.cpp  */
#include <stdio.h>                          //使用库函数 printf 和 scanf
                                            //空行,下面是主函数
int main( )
{
  int x, sign;                              //sign 存储数学函数 sign(x)的函数值
  printf("请输入 x 的值: ");                  //输出提示信息
  scanf("%d", &x);                          //从键盘接收一个整数
  if (x > 0)
    sign = 1;                               //数学函数 sign(x)的函数值为 1
  else if (x < 0)
        sign = -1;                          //数学函数 sign(x)的函数值为-1
      else                                  //注意嵌套 if-else 语句的缩进格式
        sign = 0;                           //数学函数 sign(x)的函数值为 0
  printf("x 的值为: %d,符号函数的值为: %d\n", x, sign);
  return 0;                                 //将 0 返回操作系统,表明程序正常结束
}
```

运行结果如下（下划线为用户输入）：

```
请输入 x 的值: -6
x 的值为: -6，符号函数的值为: -1
```

例 5.5 将三个整数由小到大输出。

解：设三个整数为 x、y 和 z，算法请参见 1.2.2 节。

【程序】 首先用变量 x、y 和 z 接收从键盘上输入的三个整数，然后用 if 语句比较 x 和 y 的值，再用嵌套的 if-else 语句将 z 和 x、y 进行比较，程序如下：

```
/*  example5-5.cpp  */
#include <stdio.h>                          //使用库函数 printf 和 scanf
                                            //空行,下面是主函数
int main( )
{
  int x, y, z, temp;
  printf("请输入三个整数: ");                 //输出提示信息
  scanf("%d%d%d", &x, &y, &z);              //从键盘接收三个整数
  if (x > y)                                //先判断 x 和 y
  {                                         //以下为复合语句
```

```
11          temp = x; x = y; y = temp;
                                    //使得 x 保存较小的整数,y 保存较大的整数
12      }
13      if (z < x)                  //再将 z 与 x 和 y 中的较小者比较,如果 z 最小
14      {                           //从小到大依次是 z、x、y,注意变量赋值顺序
15        temp = z; z = y; y = x; x = temp;
16      }
17      else if (z < y)             //否则 z 不是最小,进一步判断:如果 z 比 y 小
18      {                           //从小到大依次是 x、z、y,注意变量赋值顺序
19         temp = y; y = z; z = temp;
20      }
21      printf("这三个整数由小到大依次是:%3d %3d %3d\n", x, y, z);
22      return 0;                   //将 0 返回操作系统,表明程序正常结束
23  }
```

运行结果如下（下划线为用户输入）：

```
请输入三个整数:1 6 3
这三个整数由小到大依次是:  1  3  6
```

5.2.2 算术值控制的选择结构

多分支选择结构的基本特点是：程序的流程由多个分支构成，在程序的一次执行过程中，根据不同情况执行不同的程序段。在 C/C++语言中，多分支的选择结构由 switch 语句实现，switch 语句也称为开关语句或选择语句。

【语法】 switch 语句的一般形式如下：

```
                                    算术表达式
switch (表达式)  <-------------     必须有括号
{                                   常量值,相当于语句标号
    case 常量表达式 1: 语句 1  [break;]
    case 常量表达式 2: 语句 2  [break;]
       ⋮
    case 常量表达式 n: 语句 n  [break;]
     [default: 语句 n+1  break;]
}  <-------------    没有分号
```

其中，表达式一般是算术表达式，其运算结果通常为整型或字符型数据；常量表达式中不能包含变量，每个常量表达式的值必须互不相同（即对于同一个常量值，只能执行同一段程序）；每个 case 后面的语句可以有多条并且可以不用花括号括起来；方括号表示该项是可选的。

【语义】 计算表达式的值；将表达式的值逐个与 case 后面的常量表达式的值进行比较，当表达式的值与某一个常量表达式的值相等时，顺序执行该 case 后面的语句（case 子句只起到语句标号的作用）；如果表达式的值与所有常量表达式的值都不相等时，则执行 default 后面的语句；当执行 break 语句时，跳出 switch 语句，顺序执行 switch 语句的

下一条语句。switch 语句的执行流程如图 5.6 所示。

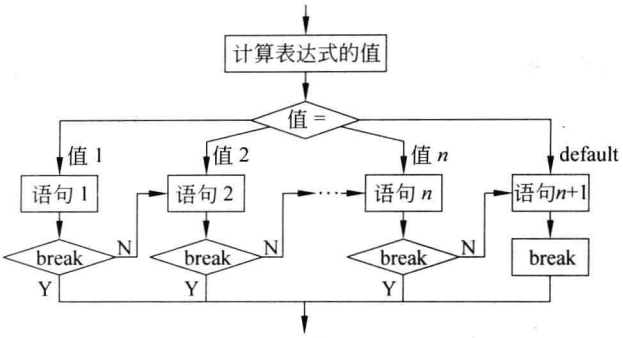


图 5.6　switch 语句的执行流程

例 5.6　将百分制成绩转换为对应的等级制成绩，其中 90～100 分的成绩等级为 A，80～89 分的成绩等级为 B，70～79 分的成绩等级为 C，60～69 分的成绩等级为 D，0～59 分的成绩等级为 E。

解：设百分制成绩为 score，则 score 是[0，100]区间的一个实数，switch 语句中表达式的运算结果通常为整型，因此，需要将 score 转换为整数。

【程序】　首先用变量 score 接收从键盘上输入的百分制成绩，然后用 switch 语句实现转换。注意十位数字决定转换的等级，所以将 score 转换为整数再整除 10。程序如下：

```
/*    example5-6.cpp   */
#include <stdio.h>                  //使用库函数 printf 和 scanf
                                    //空行,下面是主函数
int main( )
{
  double score;                     //百分制成绩可以是实数,设 score 为实型
  int temp;                         //temp 为暂存变量
  char grade;                       //等级制成绩用字母表示,设 grade 为字符型
  printf("请输入一个百分制成绩：");      //输出提示信息
  scanf("%lf", &score); //从键盘接收一个 double 型数据要用格式符%lf
  temp = (int)score / 10;           //执行整除前进行强制类型转换
  switch (temp)                     // switch 语句表达式的运算结果通常为整型
  {
    case 10:                        //case 10 和 case 9 共用一个程序段
    case 9:  grade = 'A'; break;
    case 8:  grade = 'B'; break;
    case 7:  grade = 'C'; break;
    case 6:  grade = 'D'; break;
    default:  grade = 'E'; break;
  }
  printf("百分制成绩%4.1f 对应的等级制成绩为：%c\n", score, grade);
  return 0;                         //将 0 返回操作系统,表明程序正常结束
}
```

运行结果如下（下划线为用户输入）：

```
请输入一个百分制成绩：87.5
百分制成绩 87.5 对应的等级制成绩为：B
```

良好的编程习惯 5.1

由于 switch 语句可能会很长，如果 case 子句较短，则程序较容易阅读。因此，一般将 case 子句写在一行，即将关键字 case、子句的语句和 break 语句放在一行。

5.2.3 解决任务 5.2 的程序

首先用变量 x 接收从键盘输入的三位整数，分离出个位、十位和百位数字，再用 if 语句实现判断，程序如下：

```
1   /*    duty5-2.cpp   */
2   #include <stdio.h>                    //使用库函数 printf 和 scanf
3                                         //空行,下面是主函数
4   int main( )
5   {
6     int x, x1, x2, x3, y;
7     printf("请输入一个三位整数：");     //输出提示信息
8     scanf("%d", &x);                    //从键盘接收一个整数
9     x1 = x % 10;                        //x1 保存 x 的个位数字
10    y = x / 10;                         //y 暂存 x 的高两位数
11    x2 = y % 10;                        //x2 保存 x 的十位数字
12    x3 = y / 10;                        //x3 保存 x 的百位数字
13    if (x1 * x1 * x1 + x2 * x2 * x2 + x3 * x3 * x3 == x)
14      printf("%d 是水仙花数\n", x);
15    else
16      printf("%d 不是水仙花数\n", x);
17    return 0;                           //将 0 返回操作系统,表明程序正常结束
18  }
```

运行结果如下（下划线为用户输入）：

```
请输入一个三位整数：153
153 是水仙花数
```

5.3 循 环 结 构

【任务 5.3】鸡兔同笼问题

【问题】 鸡有 2 只脚，兔子有 4 只脚，假设笼子里共有 M 只头 N 只脚，问鸡和兔子各有多少只？

【想法】 设鸡有 x 只，兔子有 y 只，则有如下方程组成立：

$$\begin{cases} x+y=M \\ 2x+4y=N \end{cases}$$

【算法】 设变量 chicken 表示鸡的个数，rabbit 表示兔子的个数，算法如下：

伪代码

```
step1: chicken 从 0~M 重复执行下述操作:
    step1.1: rabbit = M - chicken;
    step1.2: 如果(2 * chicken + 4 * rabbit 等于 N),则跳出循环;
    step1.3: chicken++;
step2: 如果是提前跳出循环,则输出 chicken 和 rabbit 的值;
       否则输出"无解";
```

在实际问题的求解过程中，往往需要重复执行某个操作，几乎所有的程序设计语言都提供循环结构来描述具有规律性的重复处理。循环结构用有限的语句描述多次重复执行的操作，不仅可以大大缩短程序的长度，而且发挥了计算机擅长重复运算的特点，因此，循环结构是程序设计中使用最频繁的控制结构。

循环结构的基本特点是：当给定的条件成立时，重复执行某程序段，给定的条件称为**循环条件**，重复执行的程序段称为**循环体**。通常情况下，循环结构利用某个变量来控制循环条件，通过改变这个变量的值最终结束循环，这个变量称为**循环变量**。

5.3.1 当型循环结构

当型循环结构的基本特点是：循环体有可能一次也不执行。在 C/C++语言中，当型循环结构由 while 语句实现，因此也称 while 循环。

【语法】 while 语句的一般形式如下：

```
                循环条件
while （表达式）  必须有括号
  语句          循环体
```

其中，表达式一般是逻辑表达式，用来表示循环条件，注意括号不能省略；语句是重复执行的循环体，可以是任何满足 C/C++语言语法规则的语句。

【语义】 计算表达式的值；当表达式的值为真时执行语句（即循环体）；重新计算表达式的值，以决定是否再次执行循环体；当表达式的值为假时结束循环，顺序执行 while 语句的下一条语句。while 语句的执行流程如图 5.7 所示。

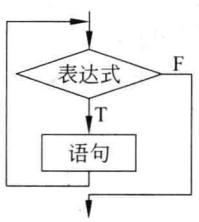


图 5.7 while 语句的执行流程

例 5.7 计算 n!。

解：由数学知识可知，n!＝n×（n–1）×…×2×1，设变量 result 保存计算结果，首先将变量 result 初始化为 1，然后将 result 乘以 1 再存入变量 result、将 result 乘以 2 再存入变量 result……，直至将 result 乘以 n 再存入变量 result，则 result 即为 n!。

【算法】 设变量 result 保存阶乘结果，设变量 i 表示乘数。算法如下：

伪代码

```
step1: 初始化阶乘结果 result = 1;
step2: 循环变量 i 从 1~n，重复执行下述操作:
    step2.1: result = result * i;
    step2.2: i++;
step3: 输出 result;
```

【程序】 注意到 n!值以指数级的速度增长，则变量 result 很快就超出 int 型数据的表示范围，因此，将变量 result 设为 long int 型。程序如下：

```
1   /*  example5-7.cpp  */
2   #include <stdio.h>                    //使用库函数 printf 和 scanf
3                                         //空行,下面是主函数
4   int main( )
5   {
6     int n, i = 1;                       //i 为循环变量并初始化为 1
7     long int result = 1;
                                  // result 为阶乘结果,n!可能会超出 int 的表示范围
8     printf("请输入一个整数: ");          //输出提示信息
9     scanf("%d", &n);                    //从键盘接收一个整数
10    while (i <= n)                      //当 i 小于等于 n 时执行循环
11    {
12      result = result * i;
13      i++;                              //循环变量增 1
14    }
15    printf("%d 的阶乘为: %ld\n", n, result);
                                          //%ld 输出 long int 型整数
16    return 0;                           //将 0 返回操作系统,表明程序正常结束
17  }
```

运行结果如下（下划线为用户输入）：

```
请输入一个整数: 10
10 的阶乘为: 3628800
```

如果循环次数由循环体的执行情况确定，而且循环体有可能一次也不执行，则一般选用当型循环。

例 5.8 欧几里得算法：辗转相除法求两个自然数 m 和 n 的最大公约数。

解：算法请参见 1.2.2 节。

【程序】 首先用变量 m 和 n 接收从键盘输入的两个整数，由于算法要求 m 大于等于 n，因此，如果 m 小于 n，则将 m 和 n 的值进行交换，然后用 while 循环实现辗转相除。程序如下：

```
1   /*   example5-8.cpp   */
2   #include <stdio.h>                    //使用库函数 printf 和 scanf
```

```
3                                                    //空行,下面是主函数
4       int main( )
5       {
6         int m, n, r, temp;
7         printf("请输入两个正整数，以空格分隔：");          //输出提示信息
8         scanf("%d %d", &m, &n);                    //从键盘接收两个整数
9         if (m < n)                                 //如果 m 小于 n,则将 m 和 n 进行交换
10        {
11          temp = m; m = n; n = temp;
12        }
13        r = m % n;
14        while (r != 0)                             //当余数 r 不等于 0 时执行循环
15        {
16          m = n;
17          n = r;
18          r = m % n;
19        }
20        printf ("这两个整数的最大公约数是：%d\n", n);
21        return 0;                                  //将 0 返回操作系统,表明程序正常结束
22      }
```

运行结果如下（下划线为用户输入）：

```
请输入两个正整数，以空格分隔：25  35
这两个整数的最大公约数是：5
```

良好的编程习惯 5.2

在循环语句中，循环条件也可以是具有整型值的算术表达式，但这是一种不好的程序风格。良好的程序风格是用条件表达式或逻辑表达式来表示循环条件，例如，while (r) 应写为 while (r != 0)。

5.3.2 直到型循环结构

直到型循环结构的基本特点是：循环体至少执行一次。在 C/C++语言中，直到型循环结构由 do-while 语句实现，因此也称 do-while 循环。

【语法】 do-while 语句的一般形式如下：

```
do
    语句          ←————    循环体
while (表达式);   ←——      以分号结尾
          ↑————————        循环条件
```

其中，表达式一般是逻辑表达式，表示循环条件，注意括号不能省略；语句是重复执行的循环体，可以是任何满足 C/C++语言语法规则的语句。

【语义】 执行语句（即循环体）；计算表达式的值，当表达式的值为真时再次执行语句，当表达式的值为假时结束循环，顺序执行 do-while 语句的下一条语句。do-while 语句的执行流程如图 5.8 所示。

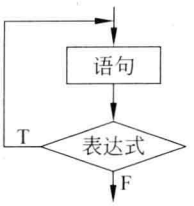


图 5.8 do-while 语句的执行流程

例 5.9 计算 n!。

解：用 do-while 循环改写程序 example5-7.cpp，程序如下：

```
/*  example5-9.cpp  */
#include <stdio.h>                    //使用库函数 printf 和 scanf
                                      //空行,下面是主函数
int main( )
{
  int n, i = 1;                       //i 为循环变量并初始化为 1
  long int result = 1;
                                      // result 为阶乘结果,n!可能会超出 int 的表示范围
  printf("请输入一个整数: ");          //输出提示信息
  scanf("%d", &n);                    //从键盘接收一个整数
  do                                  //循环变量 i 从 1 开始,则至少执行一次循环体
  {
    result = result * i;
    i++;                              //循环变量增 1
  } while (i <= n);
  printf("%d 的阶乘为: %ld\n", n, result);
  return 0;                           //将 0 返回操作系统,表明程序正常结束
}
```

运行结果与例 5.7 相同。

如果循环次数由循环体的执行情况确定，而且循环体至少执行一次，则可以选用直到型循环。

例 5.10 计算整数中所含数字的位数。

解：将输入的整数反复除以 10，直到商为 0，则执行除法的次数就是该整数所含数字的位数。例如，285 / 10 = 28，28 / 10 = 2，2 / 10 = 0，共执行 3 次除法，则 285 包含 3 位数字。

【算法】 设变量 x 表示整数，digits 存储整数 x 的位数，算法如下：

伪代码

```
step1: 初始化位数 digits = 0;
step2: 重复执行下述操作,直到 x 等于 0:
    step2.1: digits++;
    step2.2: x = x / 10;
step3: 输出 digits;
```

【程序】 由于整数至少含有一位数字（包括整数 0），显然选用直到型循环更合适。程序如下：

```
/*    example5-10.cpp    */
#include <stdio.h>                          //使用库函数 printf 和 scanf
                                            //空行,下面是主函数
int main( )
{
  int x, digits = 0;                        //digits 存储整数 x 的位数
  printf("请输入一个整数: ");                //输出提示信息
  scanf("%d", &x);                          //从键盘接收一个整数
  do
  {
    digits++;                               //位数增加 1 位
    x = x / 10;                             //将 x 缩小 10 倍
  } while (x != 0);                         //当 x 不等于 0 时执行循环
  printf("有%d 位数字\n", digits);
  return 0;                                 //将 0 返回操作系统,表明程序正常结束
}
```

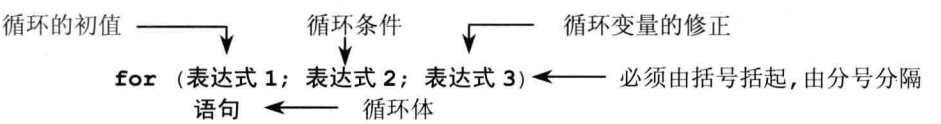
运行结果如下（下划线为用户输入）：

```
请输入一个整数：123
有 3 位数字
```

5.3.3 计数型循环结构

如果程序中的某些程序段需要重复执行确定的次数，可以采用计数型循环结构。在C/C++语言中，计数型循环结构由 for 语句实现，因此也称 for 循环。

【语法】 for 语句的一般形式如下：



其中，表达式 1 一般是赋值表达式，用于设置循环开始时某些变量的初值，如果为多个变量赋初值，则赋值表达式以逗号分隔（称为逗号表达式）；表达式 2 一般是逻辑表达式，表示循环条件；表达式 3 一般是赋值表达式，用于改变循环变量，以保证最终能够结束循环；语句是重复执行的循环体，可以是任何满足 C/C++语言语法规则的语句。

【语义】 计算表达式 1 的值；判断表达式 2 是否成立，如果成立则执行语句（即循环体），如果不成立则退出循环；计算表达式 3 的值；判断表达式 2 是否成立，以决定是否再次执行循环体。for 语句的执行流程如图 5.9 所示。

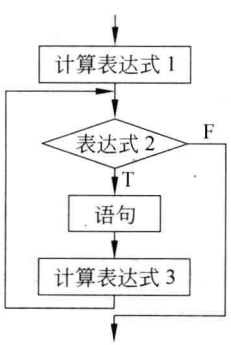


图 5.9 for 语句的执行流程

例 5.11 计算 n!。

解：用 for 循环改写程序 example5-7.cpp，程序如下：

```
/*  example5-11.cpp  */
#include <stdio.h>                          //使用库函数 printf 和 scanf
                                            //空行,下面是主函数
int main()
{
  int n, i;                                 //i 为循环变量
  long int result;  // result 为阶乘结果,n!可能会超出 int 的表示范围
  printf("请输入一个整数: ");                //输出提示信息
  scanf("%d", &n);                          //从键盘接收一个整数
  for (result = 1, i = 1; i <= n; i++)      //表达式 1 为逗号表达式
    result = result * i;
  printf("%d 的阶乘为: %ld\n", n, result);
  return 0;                                 //将 0 返回操作系统,表明程序正常结束
}
```

运行结果与例 5.7 相同。

如果在执行循环体之前能够确定循环次数，或者能够确定循环变量的初值、终值和循环步长，一般选用计数型循环。

例 5.12 计算 2 + 4 + 6 + … + 100 的值。

解：设变量 sum 作为累加器，则循环变量 i 的初值为 2、终值为 100、步长为 2，重复执行加法操作，因此采用 for 循环。程序如下：

```
/*  example5-12.cpp  */
#include <stdio.h>                          //使用库函数 printf 和 scanf
                                            //空行,下面是主函数
int main( )
{
  int sum = 0, i;                           //sum 存储累加和,i 是循环变量
  for (i = 2; i <= 100; i = i + 2)  //注意循环变量 i 的改变
  {
    sum = sum + i;
  }
  printf("100 之内所有偶数的和为: %d\n", sum);
  return 0;                                 //将 0 返回操作系统,表明程序正常结束
}
```

运行结果如下：

```
100 之内所有偶数的和为: 2550
```

在 for 循环中，表达式 1、表达式 2 和表达式 3 均可以省略，但是分号不能省略。例如，如下均为合法的 for 循环：

```
i = 0;
for( ; i < n; i++)              //省略表达式 1,可以在 for 循环外面为循环变量赋初值
{
```

```
    ⋮
}

for(i = 0;  ; i++)                    //省略表达式 2,可以在循环体内测试循环条件
{
   if (i >= n) break;
}

for(i = 0; i < n;)                    //省略表达式 3,可以在循环体内改变循环变量
{
   i++;
}
```

5.3.4 循环结构的嵌套

从语法上看，循环结构中的语句可以是任何语句，包括复合语句、分支语句、循环语句以及其他各类语句。如果循环结构中的语句又是一个循环语句，则构成循环结构的嵌套。在 C/C++语言中，while 循环、do-while 循环和 for 循环可以相互嵌套，但是嵌套的循环语句不能有交叉，如图 5.10 所示。

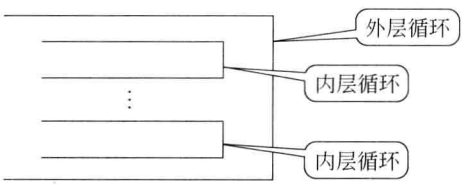


图 5.10　合法的嵌套循环形式

例 5.13　打印九九乘法表，即打印如下形式的乘法表：

1×1 =1

1×2 =2　2×2 =4

⋮

1×9 =9　2×9 =18　3×9 =27　4×9 =36　5×9 =45　6×9 =54　7×9 =63　8×9 =72　9×9 =81

解：九九乘法表需要一行一行地打印，共打印 9 行，打印第 i 行时，需要打印 i 列。程序需要两层嵌套的循环，外层循环变量为 i，其变化范围为[1, 9]，内层循环变量为 j，其变化范围为[1, i]，程序如下：

```
1    /*  example5-13.cpp   */
2    #include <stdio.h>                    //使用库函数 printf 和 scanf
3                                          //空行,下面是主函数
4    int main( )
5    {
6      int i, j;                        //i 为外层循环变量,j 为内层循环变量
7      for (i = 1; i <= 9; i++)          //打印第 i 行
8      {
```

```
        for (j = 1; j <= i; j++)          //打印第 j 列
          printf("%d×%d = %-2d  ", j, i, i * j);
                                          //列号在前,乘法结果左对齐
        printf("\n");                     //第 i 行打印完毕,换行
      }
      return 0;                           //将 0 返回操作系统,表明程序正常结束
    }
```

5.3.5 解决任务 5.3 的程序

```
/*   duty5-3.cpp   */
#include <stdio.h>                        //使用库函数 printf 和 scanf
                                          //空行,下面是主函数
int main( )
{
  int M, N;                               //M 存储头的个数,N 存储脚的个数
  int chicken, rabbit;
                    // chicken 存储鸡的个数,rabbit 存储兔子的个数
  printf("请输入头的个数和脚的个数: ");      //输出提示信息
  scanf("%d%d", &M, &N);                  //从键盘接收两个整数
  for (chicken = 0; chicken <= M; chicken++)
                                          //chicken 为循环变量
  {
    rabbit = M - chicken;                 //满足方程 chicken + rabbit = M
    if (2 * chicken + 4 * rabbit == N) break;
                                          //方程组已解,跳出循环
  }
  if (chicken <= M)                       //如果是则提前跳出循环
    printf("鸡有%d 只, 兔子有%d 只\n", chicken, rabbit);
  else
    printf("输入数据不合理, 无解\n");
  return 0;                               //将 0 返回操作系统,表明程序正常结束
}
```

运行结果如下（下划线为用户输入）：

```
请输入头的个数和脚的个数: 4  12
鸡有 2 只, 兔子有 2 只
```

5.4 其他控制语句

【任务 5.4】素数判定

【问题】 对给定的整数 x，判定是否是素数。

【想法】 素数除了 1 和其自身外没有其他因子，可以将整数 x 除以 2 ~ x–1，如果

能整除，则 x 不是素数。

【算法】 设变量 x 存储需要判定的整数，算法如下：

伪代码

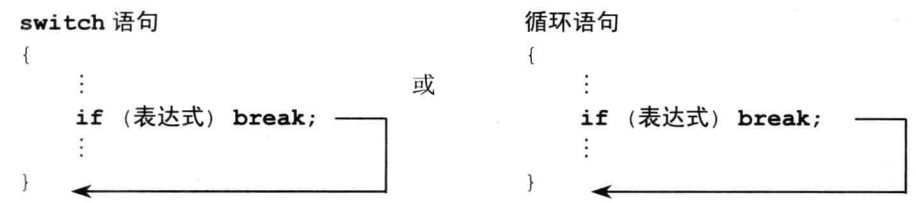
```
step1: 循环变量 i 从 2~x-1,循环执行下述操作:
    step1.1: 如果 x % i 等于 0,则说明 x 不是素数,跳出循环;
    step1.2: i++;
step2: 如果提前跳出循环,则 x 不是素数;否则 x 是素数;
```

在循环过程中，有时需要提前跳出循环，或者满足一定的条件时不执行循环体中剩下的语句而重新开始新一轮的循环，C/C++语言提供了 break 语句和 continue 语句实现循环的中途退出。

5.4.1 break 语句

break 语句主要用在 switch 语句和循环语句中，其作用是跳出 switch 语句和循环语句，即结束 switch 语句和循环语句。

【语法】 break 语句一般与 if 语句配合使用，其一般形式如下：



其中，循环语句可以是任意一种循环结构（当型循环、直到型循环或计数型循环）。

【语义】 如果表达式成立，则跳出 switch 语句（或循环语句），执行 switch 语句（或循环语句）的下一条语句。

例 5.14 从键盘上输入 10 个整数，判断是否含有负数。

解：依次读取从键盘上输入的整数并进行判断，当输入负整数时，没有必要继续读入数据，则执行 break 语句跳出循环。程序如下：

```
1   /*   example5-14.cpp    */
2   #include <stdio.h>                  //使用库函数 printf 和 scanf
3                                       //空行，下面是主函数
4   int main( )
5   {
6     int flag = 0;  //flag 为是否有负数的标志，0 表示没有负数，1 表示有负数
7     int i, x;                         //i 为循环变量，x 暂存从键盘输入的整数
8     printf("请输入 10 个整数，可以是正数也可以是负数：");
9     for (i = 1; i <= 10; i++)         //已知循环次数，用计数型循环
10    {
11      scanf("%d", &x);                //从键盘接收一个整数
12      if (x < 0) {                    //接收到负数
13        flag = 1; break;              //没有必要再接收其他数据，跳出循环
```

```
14          }
15        }
16        if (flag == 1)
17          printf("输入的整数中有负数\n");
18        else
19          printf("输入的整数中没有负数\n");
20        return 0;                          //将 0 返回操作系统，表明程序正常结束
21      }
```

5.4.2 continue 语句

continue 语句主要用在循环语句中，其作用是跳过循环语句中尚未执行的部分，重新开始新一轮循环。需要强调的是，continue 语句并没有结束循环语句，只是结束本次循环的执行。

【语法】 continue 语句一般与 if 语句配合使用，其一般形式如下：

```
循环语句
{
    ⋮
    if (表达式) continue;
    ⋮
}
```

其中，循环语句可以是任意一种循环结构（当型循环、直到型循环或计数型循环）。

【语义】 如果表达式成立，则结束本次循环，重新开始新一轮的循环。

例 5.15 从键盘上输入 10 个整数，输出其中的负数并统计负数出现的次数。

解：依次读取从键盘上输入的整数并进行判断，如果该数为正数，则执行 continue 语句，程序回到 for 语句执行 i++，开始下一轮循环，否则输出该数并累计次数。设变量 count 表示负数出现的次数，程序如下：

```
1     /*   example5-15.cpp    */
2     #include <stdio.h>                   //使用库函数 printf 和 scanf
3                                          //空行,下面是主函数
4     int main( )
5     {
6       int count = 0;                     //count 为负数出现的次数
7       int i, x;                          //i 为循环变量,x 暂存从键盘输入的整数
8       printf("请输入 10 个整数,可以是正数也可以是负数: ");
9       for (i = 1; i <= 10; i++)          //已知循环次数，用计数型循环
10      {
11        scanf("%d", &x);                 //从键盘接收一个整数
12        if (x >= 0)                      //如果接收到的是正数
13          continue;                      //跳过 for 循环余下语句,执行 i++
14        else
15        {
16          printf("%5d", x);
```

```
            count++;                          //计数器加 1
          }
        }
        printf("\n 负数出现%d 次\n", count);
        return 0;                             //将 0 返回操作系统,表明程序正常结束
    }
```

5.4.3 解决任务 5.4 的程序

```
    /*   duty5-4.cpp   */
    #include <stdio.h>                        //使用库函数 printf 和 scanf
                                              //空行,下面是主函数
    int main( )
    {
      int x, i;                               //i 为循环变量
      printf("请输入一个整数: ");              //输出提示信息
      scanf("%d", &x);                        //从键盘接收一个整数
      for (i = 2; i < x; i++)                 //从 2 开始试除,一直试到 x - 1
      {
        if (x % i == 0)                       //能够整除,不是素数,跳出循环
          break;
      }
      if (i < x)                              //如果是则提前跳出循环
        printf("%d 不是素数\n", x);
      else
        printf("%d 是素数\n", x);
      return 0;                               //将 0 返回操作系统,表明程序正常结束
    }
```

运行结果如下（下划线为用户输入）：

```
请输入一个整数：137
137 是素数
```

5.5 程序设计实例

5.5.1 实例 1——百元买百鸡问题

【问题】 已知公鸡 5 元一只，母鸡 3 元一只，小鸡 1 元三只，花 100 元钱买 100 只鸡，问公鸡、母鸡、小鸡各多少只？

【想法】 设公鸡、母鸡和小鸡的个数为 x、y、z，则有如下方程组成立：

$$\begin{cases} x+y+z=100 \\ 5\times x+3\times y+z/3=100 \end{cases} \quad \text{且} \quad \begin{cases} 0\leqslant x\leqslant 20 \\ 0\leqslant y\leqslant 33 \\ 0\leqslant z\leqslant 100 \end{cases}$$

注意，方程组可能有多个解，需要输出所有满足条件的解。

【算法】 设变量 x 表示公鸡的个数，y 表示母鸡的个数，z 表示小鸡的个数，count 表示解的个数，算法如下：

伪代码

```
step1: 初始化解的个数 count = 0;
step2: 循环变量 x 从 0~20 循环执行下述操作:
    step2.1: 循环变量 y 从 0~33 循环执行下述操作:
        step2.1.1: z = 100 - x - y;
        step2.1.2: 如果 5*x + 3*y + z/3 等于 100,则 count++;输出 x、y 和
                   z 的值;
        step2.1.3: y++;
    step2.2: x++;
step3: 如果 count 等于 0,则输出无解信息;
```

【程序】 注意到小鸡 1 元三只，则小鸡的个数应该是 3 的倍数，因此，在判断总价是否满足方程时要先判断 z 是否是 3 的倍数。程序如下：

```
1   /*    百元百鸡.cpp   */
2   #include <stdio.h>                    //使用库函数 printf 和 scanf
3                                         //空行,下面是主函数
4   int main( )
5   {
6     int x, y, z;                  //x、y 和 z 分别表示公鸡、母鸡和小鸡的个数
7     int count = 0;                      //解的个数初始化为 0
8     for (x = 0; x <= 20; x++)           //公鸡个数 x 的范围是 0 到 20
9     {
10      for (y = 0; y <= 33; y++)         //母鸡个数 y 的范围是 0 到 33
11      {
12        z = 100 - x - y;                //满足方程 x + y + z = 100
13        if ((z % 3 == 0) && (5 * x + 3 * y + z/3 == 100))
                                          //满足总价是 100 元
14        {
15          count++;                      //解的个数加 1
16          printf("公鸡有%2d 只,母鸡有%2d 只,小鸡有%2d 只\n", x, y, z);
17        }
18      }
19    }
20    if (count == 0)
21      printf("输入数据不合理，问题无解\n");
22    return 0;                           //将 0 返回操作系统,表明程序正常结束
23  }
```

运行结果如下：

```
公鸡有 0 只,母鸡有 25 只,小鸡有 75 只
公鸡有 4 只,母鸡有 18 只,小鸡有 78 只
公鸡有 8 只,母鸡有 11 只,小鸡有 81 只
公鸡有 12 只,母鸡有 4 只,小鸡有 84 只
```

5.5.2 实例 2——歌德巴赫猜想

【问题】 歌德巴赫猜想：任意大于 2 的偶数可以分解为两个素数之和。歌德巴赫猜想是世界著名的数学难题，至今未能在理论上得到证明，请验证歌德巴赫猜想。

【想法】 设偶数为 n，将 n 分解为 n1 和 n2，使得 n = n1 + n2，并且 n1 和 n2 为素数。可以将 n1 从 2 开始，令 n2 = n – n1，如果 n1 和 n2 均为素数，则得到结果；否则将 n1 增 1 再试，显然 n1 的最大值为 n/2。

【算法】 设变量 n 表示输入的偶数，将 n 分解为 n1 和 n2，算法如下：

伪代码

```
step1: n1 从 2~n/2 循环执行下述操作:
    step1.1: n2 = n - n1;
    step1.2: 如果 n1 不是素数,则 n1++; 转 step1.1 试下一组数;
    step1.3: 如果 n2 不是素数,则 n1++; 转 step1.1 试下一组数;
    step1.4: n1 和 n2 均为素数,则转 step2 输出结果;
step2: 输出 n1 和 n2;
```

【程序】 首先用变量 n 接收从键盘输入的一个偶数，然后 n1 从 2 开始，n2 从 n – 2 开始，逐个试探 n1 和 n2 是否为素数，判断素数的算法请参见任务 5.4。程序如下：

```
/*   歌德巴赫猜想.cpp   */
#include <stdio.h>                    //使用库函数 printf 和 scanf
                                      //空行,下面是主函数
int main( )
{
  int n, n1, n2;
  int i;                              //i 为循环变量
  printf("请输入一个偶数: ");          //输出提示信息
  scanf("%d", &n);                    //从键盘接收一个整数
  for (n1 = 2; n1 <= n/2; n1++) //n1 最小是 2,最大是 n/2
  {
    n2 = n - n1;                      //满足 n1 + n2 = n
    for (i = 2; i < n1; i++)          //判断 n1 是否为素数
    {
      if (n1 % i == 0)                //n1 不是素数,跳出循环
        break;
    }
    if (i < n1)              //如果 n1 不是素数,则不用判断 n2 是否为素数
       continue;                      //跳过外层 for 循环余下语句,执行 n1++
    for (i = 2; i < n2; i++)          //进一步判断 n2 是否为素数
    {
      if (n2 % i == 0)                //n2 不是素数,跳出循环
        break;
    }
    if (i >= n2)             //如果 n2 是素数,则完成分解,跳出外层 for 循环
      break;
```

```
    }
    printf("%d 可分解为%d+%d\n", n, n1, n2);
    return 0;                          //将 0 返回操作系统,表明程序正常结束
}
```

运行结果如下（下划线为用户输入）：

```
请输入一个偶数：44
44 可分解为 3+41
```

习 题 5

一、选择题

1. if-else 语句嵌套使用时，else 与（　　）相配对。

   A. 缩排位置相同的 if　　B. 其上最近的 if

   C. 其下最近的 if　　D. 其上最近的未配对的 if

2. 以下错误的 if 语句是（　　）。

   A. if (x > y) z = x;　　B. if (x == y) z = 0;

   C. if (x != y) z = x else z = y;　　D. if (x < y) {x++; y--;}

3. 以下程序的输出结果是（　　）。

```
void main( )
{
  int a = 20, b = 30, c = 40;
  if (a > b) a = b;
  b = c; c = a;
  printf("a = %d, b = %d, c = %d"\n, a, b, c);
}
```

   A. a = 20, b = 30, c = 20　　B. a = 20, b = 40, c = 20

   C. a = 30, b = 40, c = 20　　D. a = 30, b = 40, c = 30

4. 对于如下程序，正确的判断是（　　）。

```
void  main( )
{
  int a, b;
  scanf("%d, %d", &a, &b);
  if (a > b) a = b; b =a;
  else a++; b++;
  printf("%d, %d", a, b);
}
```

   A. 有语法错误不能通过编译　　B. 若输入 4, 5 则输出 5, 6

   C. 若输入 5, 4 则输出 4, 5　　D. 若输入 5, 4 则输出 5, 5

5. 以下程序的输出结果是（　　）。

```
void main( )
{
  int x = 1, a = 0, b = 0;
  switch(x)
  {
    case 0: b++;
    case 1: a++;
    case 2: a++;b++;
  }
  printf("a = %d, b = %d\n", a, b);
}
```

A. a = 2, b = 1　　B. a = 1, b = 1　　C. a = 1, b = 0　　D. a = 2, b = 2

6. 以下循环语句的执行次数是（　　）。

```
for (int i = 2; i != 0; i--) printf("%d", i);
```

A. 无限次　　B. 0 次　　C. 1 次　　D. 2 次

7. 语句 while (!E)中的表达式!E 等价于（　　）。

A. E == 0　　B. E != 1　　C. E != 0　　D. E == 1

8. 可将 for (表达式 1;　;表达式 3)理解为（　　）。

A. for (表达式 1; 0 ;表达式 3)　　B. for (表达式 1; 1 ;表达式 3)

C. for (表达式 1;表达式 1;表达式 3)　　D. for (表达式 1;表达式 3;表达式 3)

9. 以下程序段（　　）。

```
x = -1;
do{
  x = x * x;
} while (!x);
```

A. 是死循环　　B. 有语法错误

C. 循环体执行 1 次　　D. 循环体执行 2 次

10. 以下程序段中，while 循环的循环次数是（　　）。

```
int i = 0;
while (i < 10)
{
  if (i < 5) continue;
  if (i == 5) break;
  i++;
}
```

A. 1　　B. 10

C. 死循环　　D. 有语法错误

二、简答题

1. 在 switch 语句中，case 子句可以是条件表达式吗？为什么？

2. break 语句和 continue 语句的作用是什么？二者有什么区别？

3. C/C++语言提供了三种循环语句：while 循环、do-while 循环和 for 循环，请说明这三种循环语句各适用于什么情况。

三、程序设计题

1. 已知火车的出发时间和到达时间，计算并输出旅途时间。假设用两个 4 位整数 time1 和 time2 分别表示火车的出发时间和到达时间，其中前两位表示小时，后两位表示分钟。

2. 某高速公路每公里的收费标准如表 5.1 所示，请计算实际的收费额。

表 5.1　某高速公路每公里的收费标准

小汽车（car）	0.50 元/公里
卡车（track）	1.00 元/公里
大客车（bus）	1.50 元/公里

3. 计算数列 $a_k = 1/k(k+1)$的前 n 项和。

4. 判断给定的自然数是否为降序数。所谓降序数是指对于 $n = d_1\ d_2\ d_3 \cdots d_k$，满足 $d_i \geqslant d_{i+1}$（$1 \leqslant i \leqslant k-1$）。

5. 输出华氏—摄氏温度转换表，华氏温度的转换范围是[low, high]，转换增量为 step。

6. 有一个皮球从高为 H 米的位置自由落下，触地后反弹到原高度的一半，再落下，再反弹，如此反复。皮球在第 n 次触地时，在空中经过的总路程是多少米？第 n 次反弹的高度是多少米？

7. 输入两个正整数 m 和 n，输出 m 和 n 之间的所有素数。

8. 我国古代数学家祖冲之采用正多边形逼近的割圆术将圆周率 π 精确到小数点后 8 位。请用祖冲之的方法求 π 的近似值，要求精确到小数点后 16 位。

9. 甲、乙、丙三人同时放鞭炮，甲每隔 A 秒放一个，乙每隔 B 秒放一个，丙每隔 C 秒放一个，他们各自放 D 个。对任意给定的 A、B、C 和 D，求能听到多少声鞭炮响。

第6章

程序的组装单元——函数

C/C++程序提倡把一个大问题划分为一个个小问题，对每个小问题编制一个函数，因此，C/C++程序一般是由大量的小函数而不是少量的大函数构成的，即所谓的“小函数构成大程序”。这样的好处是让各部分相互独立且任务单一，这些独立的小函数可作为一种构件，用来构成规模较大的程序。

6.1 用户定义的函数——自定义函数

【任务 6.1】欧几里得算法（函数版）

【问题】 求任意两个自然数的最大公约数，要求用函数实现。

【想法】 将最大公约数的功能提取出来编制一个函数，在主函数中调用该函数实现求两个自然数的最大公约数。

【算法】 设函数 CommonFactor 实现求两个自然数的最大公约数，其算法描述如下：

输入：两个自然数 m 和 n

功能：求两个自然数的最大公约数

输出：m 和 n 的最大公约数

伪代码

```
step1: r = m % n;
step2: 当 r≠0 时重复执行下述操作：
    step2.1: m = n;
    step2.2: n = r;
    step2.3: r = m % n;
step3: 输出 n;
```

从构成的角度来看，函数是由一些语句组成的小程序，用来描述逻辑上相对独立的功能。从使用的角度来看，可以把函数看成一个“黑盒子”，只要将输入数据送进去就能得到需要的结果，而函数内部究竟是如何构成的，外部程序（即调用者）并不需要知道，外部程序只需要知道函数接口，即函数名、给函数输入什么以及函数将输出什么，如

图 6.1 所示。

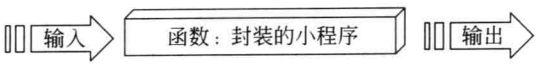
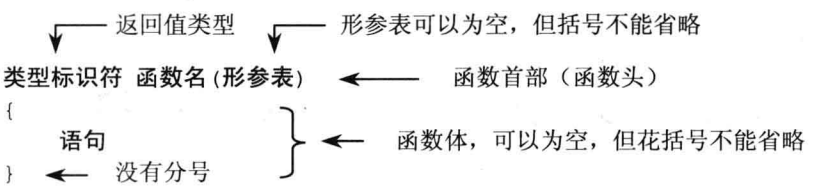


图 6.1　函数的使用示意图

程序设计语言中函数的概念来源于数学中的函数。在数学中可定义函数，例如函数 $f(x)=x^2+2x+1$，其中 x 是自变量，函数 $f(x)$的值取决于 x 的值是多少，例如当 $x=2$ 时表示为 $f(2)$，其值为 $2^2+2\times2+1=9$。在 C/C++语言中，使用函数（即调用函数）必须先定义该函数——类似于数学中的函数定义，然后再调用该函数——类似于数学中计算函数的某个特定值。

6.1.1　函数定义

【语法】　函数定义的一般形式如下：



其中，类型标识符、函数名和形参表构成了函数首部，也称为函数头。类型标识符规定了函数的返回值类型（默认是 int 型），函数名为一个标识符，形参表（即形式参数表）是由逗号分隔的变量声明，表示函数的输入；花括号括起来的部分称为函数体，是组成函数的语句序列，描述函数的具体行为。

【语义】　定义函数。为了节省存储空间，编译程序在定义函数时通常不为函数分配存储空间，在函数调用时才分配存储空间。

良好的编程习惯 6.1

为了与变量名区分开，有些程序员习惯将函数名中每个单词的首字母大写，例如 PrintTri、Max 等，而且函数名要见名知义，能够体现函数要完成的具体功能。

在定义函数时，由于函数尚未执行，其结果（即返回值）是未知的，但能够确定结果的数据类型，因此，在定义函数时要指明函数的返回值类型。从函数是否有返回值的角度，可以将函数分为有返回值函数和无返回值函数。

1. 有返回值函数

有返回值函数在函数执行结束后向调用者返回一个执行结果，称为**函数的返回值**。有返回值函数在函数定义时必须说明返回值的类型，在函数体中由 return 语句给出具体的返回值。

【语法】 有返回值 return 语句的一般形式如下：

```
return (表达式);
```

其中，表达式的结果类型应与函数的返回值类型一致，或能够通过自动类型转换将表达式的运算结果转换为返回值类型；表达式的括号可以省略。

【语义】 计算表达式的值，结束函数的执行并将表达式的值返回给调用者。

例 6.1 定义函数计算 $f(x)=x^2+2x+1$。

解：计算 $f(x)$需要输入 x，函数执行后要返回 $f(x)$的计算结果。函数定义如下：

```
int Fun(int x)                          //函数定义,x 为形参
{
  int y;                                //变量 y 存储计算结果
  y = x * x + 2 * x + 1;
  return y;                             //结束函数 Fun 的执行,并将 y 的值返回
}
```

2. 无返回值函数

无返回值函数只完成某种特定的处理，函数执行后无须向调用者返回计算结果。无返回值函数在函数定义时必须将返回值的类型说明为 void(即空类型)，函数体中的 return 语句只结束函数的执行。

【语法】 无返回值 return 语句的一般形式如下：

```
return;
```

【语义】 结束函数的执行。如果函数体中没有 return 语句，则执行到函数体结束。

例 6.2 定义函数打印九九乘法表。

解：打印九九乘法表不需要输入数据，也没有具体的计算结果，因此，形参和返回值类型均为空类型。函数定义如下：

```
void Table99( )
{
  int i, j;                             // i 为外层循环变量,j 为内层循环变量
  for (i = 1; i <= 9; i++)              //打印第 i 行
  {
    for (j = 1; j <= i; j++)            //打印第 j 列
      printf("%d×%d = %-2d", j, i, i * j);      //列号在前,乘法结果左对齐
    printf("\n");                       //第 i 行打印完毕,换行
  }
  return;                               //结束函数 Table99 的执行
}
```

良好的编程习惯 6.2

无论函数是否有返回值，都为函数定义指定一个明确的返回值类型，并且函数体中一定要出现对应的 return 语句。如果形参为空，有些程序员习惯将形参指定为 void，以

明确该函数为无参函数，如 void Table99(void)。

函数体可以是空的，即函数体中可以没有任何语句，称为**空函数**。例如：

```
void Table99( )
{

}
```

使用空函数仅仅是为了“占位”，调用者可以按正常方式对其进行调用，但不执行任何操作。

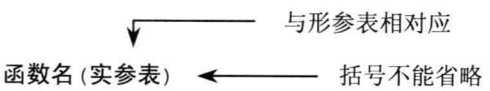
良好的编程习惯 6.3

空函数对于较大程序的编写、调试和扩充很有用。对于较大程序不能写出所有函数再进行调试，对于那些还没有具体内容的函数可以只写出空函数，等到需要扩充函数功能时或相应函数调试完后再补上具体内容。

6.1.2 函数调用

函数定义只是按照程序逻辑将相关语句组织在一起，要想执行这些语句，就需要执行函数，即**函数调用**。函数在没有被调用之前是静止的，此时的形参只是一个符号，其作用相当于占位符，标志着在形参出现的位置应该有一个什么类型的数据。**形参本质上是变量**，其初始值在函数调用时才提供，函数在被调用时把实参赋给形参，然后执行函数，函数执行结束后，释放形参的存储空间。

【语法】 函数调用的一般形式如下：



其中，实参（实际参数）的个数、顺序和类型与该函数定义中对应形参的个数、顺序和类型相匹配。注意，即使参数为空，也不能省略括号。

【语义】 函数调用的一般过程如图 6.2 所示，具体过程如下：

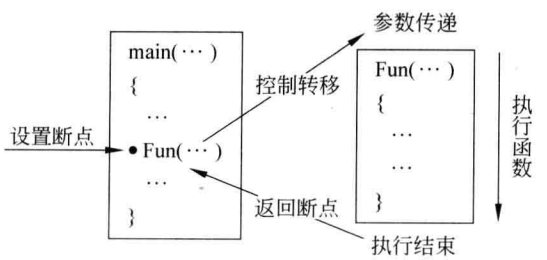


图 6.2 函数调用的过程

① 设置断点，控制转移：在函数调用处设置断点，将控制转移给被调用函数。

② 参数传递，执行函数：将实参的值传递给对应的形参，然后执行这个函数。

③ 执行结束，返回断点：函数执行结束，返回到函数调用处。

从函数是否有参数的角度，可以将函数分为有参函数和无参函数。对于有参函数，实参可以是常量、变量，也可以是表达式，甚至可以是函数，但无论是何种数据，在进行函数调用时，实参必须是“值有定义的”，而且实参的值与形参是“赋值兼容的”。在调用函数时，实参与形参结合的具体过程是：

① 计算实参表达式的值；

② 将实参的值按赋值转换规则转换成对应形参的数据类型；

③ 为形参分配存储空间；

④ 将类型转换后的实参的值传递给对应的形参变量，然后执行函数。

例 6.3 设计程序计算 $f(x)=x^2+2x+1$。

解：首先定义函数 Fun(x)，然后在主函数中调用函数 Fun(x)，程序的执行过程如图 6.3 所示，程序如下：

```
1   /*   example6-3.cpp   */
2   #include <stdio.h>                    //使用库函数 printf 和 scanf
3   int Fun(int x)                        //函数定义,x 为形参
4   {
5     int y;
6     y = x * x + 2 * x + 1;
7     return y;                           //结束函数 Fun 并将 y 的值返回给调用者
8   }
9   int main( )
10  {
11    int a = 10, b;
12    b = Fun(5);                         //函数调用,以常量值 5 作为实参
13    printf("常量可以作为实参,函数的返回值为: %d\n", b);
14    b = Fun(a);                         //函数调用,以变量 a 作为实参
15    printf("变量可以作为实参,函数的返回值为: %d\n", b);
16    b = Fun(a + 8.5);                   //函数调用,以表达式 a + 8.5 作为实参
17    printf("表达式可以作为实参,函数的返回值为: %d\n", b);
18    return 0;                           //将 0 返回操作系统,表明程序正常结束
19  }
```

运行结果如下：

```
常量可以作为实参,函数的返回值为: 36
变量可以作为实参,函数的返回值为: 121
表达式可以作为实参,函数的返回值为: 361
```

良好的编程习惯 6.4

现在来解释主函数 main 中的 return 语句。由于主函数 main 由操作系统调用， main 函数执行结束后要返回到操作系统，则 return 语句结束函数 main 的执行并将返回值传递给操作系统。

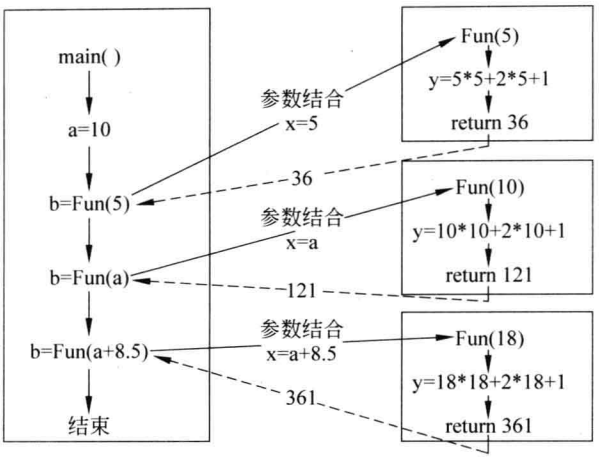


图 6.3　程序的执行过程（ ——► 函数调用 - - - -► 函数返回）

6.1.3　函数声明

C/C++语言规定：函数必须先定义后调用，因此，程序中要确保函数调用之前已经完成函数定义，否则编译器没有关于函数的任何信息，不能正确地完成翻译工作。

为了避免函数在定义前调用，一种方法是安排程序，使得每个函数定义都在此函数调用前进行。更好的解决办法是：在调用前声明这个函数的原型，即向编译程序声明将要调用此函数，并将函数原型（包括函数的返回值类型、函数名和形参表等）通知编译程序，以保证程序在编译时能判断对该函数的调用是否正确。事实上，C/C++程序的典型结构是：

```
第 1 部分：预处理指令
          符号常量定义
          全局变量定义
          函数声明

第 2 部分：主函数

第 3 部分：函数定义
```

【语法】　函数声明的一般形式如下：

```
类型标识符 函数名(形参表)；  ◄—— 以分号结束
```

函数声明类似于函数定义中的函数首部，不同之处是在其结尾处有分号。

【语义】　对函数原型进行声明，说明函数的参数和返回值类型等情况。

例 6.4　设计程序计算 $f(x)=x^2+2x+1$，采用函数声明方式。

解：在主函数之前声明函数 Fun，具体程序如下：

```
1   /*    example6-4.cpp   */
2   #include <stdio.h>                    //使用库函数 printf 和 scanf
```

```
3    int Fun(int x);                          //函数声明,注意以分号结尾
4                                             //空行,分隔程序的第 1 部分和第 2 部分
5    int main( )
6    {
7      int a = 10, b;
8      b = Fun(a);                            //函数调用,变量 a 为实参
9      printf ("f(10)的值为: %d\n", b);
10     return 0;                              //将 0 返回操作系统,表明程序正常结束
11   }
12                                            //空行,分隔程序的第 2 部分和第 3 部分
13   int Fun(int x)                           //函数定义,x 为形参
14   {
15     int y;
16     y = x * x + 2 * x + 1;
17     return y;                              //结束函数 Fun 并将 y 的值返回给调用者
18   }
```

良好的编程习惯 6.5

尽管编译程序不关心函数原型中形参的名字，在函数声明时，最好不要忽略形参的名字，因为形参的名字可以直观地显示每个形参的目的，并且提醒编程人员在函数调用时如何安排实参的顺序。一个良好的编程习惯是将函数定义中的函数首部复制到函数声明部分，然后在行尾加上分号。

6.1.4 解决任务 6.1 的程序

```
1    /*   duty6-1.cpp   */
2    #include <stdio.h>                       //使用库函数 printf 和 scanf
3    int CommonFactor(int m, int n);          //函数声明，m 和 n 为形参
4                                             //空行,以下是主函数
5    int main( )
6    {
7      int x, y, factor;      //factor 接收调用函数 CommonFactor 的返回值
8      printf("请输入两个整数: ");             //输出提示信息
9      scanf("%d%d",&x, &y);                  //从键盘接收两个整数
10     factor = CommonFactor(x, y);           //函数调用,x 和 y 为实参
11     printf("%d 和%d 的最大公约数是: %d\n", x, y, factor);
12     return 0;              //将 0 返回操作系统,表明程序正常结束
13   }
14                                            //空行,以下是其他函数定义
15   int CommonFactor(int m, int n)           //函数定义,返回值类型为 int 型
16   {
17     int r, temp;
18     if (m < n)                             //如果 m 小于 n,则交换 m 和 n 的值
19     {
20       temp = m; m = n; n = temp;
21     }
```

```
    r = m % n;
    while (r != 0)                    //当余数 r 不等于 0 时执行循环
    {
      m = n;
      n = r;
      r = m % n;
    }
    return n;
}
```

运行结果如下（下划线为用户输入）：

```
请输入两个整数：25  35
25 和 35 的最大公约数是：5
```

6.2 系统定义的函数——库函数

【任务 6.2】素数判定（函数版）

【问题】 对给定的整数 x，判定 x 是否是素数。

【想法】 任务 5.4 中是将整数 x 除以 2 ~ x–1，如果能整除，则 x 不是素数。进一步地，可以将试除范围缩小到 $2 \sim \lfloor \sqrt{x} \rfloor$（向下取整），如果能整除，则 x 不是素数。

【算法】 设函数 Prime 判断给定的整数是否为素数，其算法描述如下：

输入：整数 x

功能：素数判定

输出：如果 x 是素数，则返回 1，否则返回 0

伪代码

```
step1: 循环变量 i 为 2~√x ,循环执行下述操作:
    step1.1: 如果 x % i == 0,则说明 x 不是素数,跳出循环;
    step1.2: i++;
step2: 如果提前跳出循环,则 x 不是素数,返回 0;否则 x 是素数,返回 1;
```

C/C++语言的编译程序提供了大量库函数用于实现常见的基本功能，如库函数 sqrt(x) 用于计算 $\sqrt{x}$。在进行程序设计时不要闭门造车，如果熟悉库函数的功能和调用形式，可以省去很多不必要的工作。在使用库函数时，用户无须进行函数定义，也不必在程序中进行函数声明，只需在程序中包含该函数所在的头文件，即可在程序中直接调用相关函数。

6.2.1 头文件与文件包含

在 C/C++语言的编译系统中，库函数一般按功能（例如数学函数、字符串处理函数

等）组织在相应的头文件（.h）中，不同的 C/C++语言编译系统提供的库函数的数量、名字和功能不完全相同，使用时请查阅帮助。VC++提供的头文件在文件夹“C:\Program Files\Microsoft Visual Studio\VC98\Include”中，如图 6.4 所示。

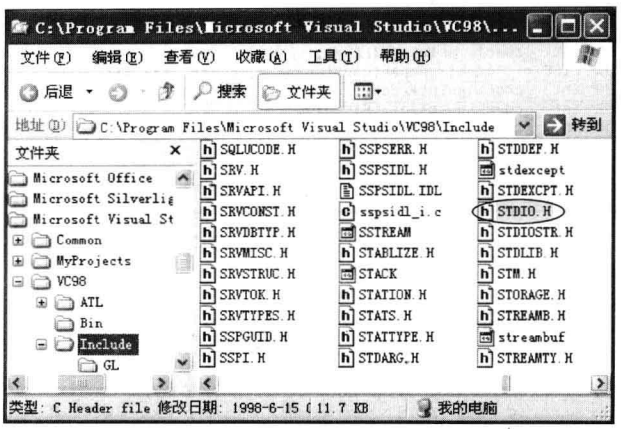


图 6.4　VC++环境下的头文件

在使用库函数时，需要在程序中包含该库函数所在的头文件。C/C++语言提供了#include 文件包含预处理指令，将一个头文件包含到源程序文件中。

【语法】　#include 指令的一般形式如下：

#include <**文件名**> ←—— 没有分号　　或　**#include "文件名"** ←—— 没有分号

其中，文件名可以带路径。如果使用尖括号，则到系统指定的包含目录查找被包含文件。如果使用双引号，则首先在系统当前目录下查找被包含文件，没找到再到系统指定的包含目录去查找。一般使用尖括号包含系统定义的头文件，使用双引号包含用户自定义的头文件或源程序文件。

【语义】　在预处理阶段，将该文件的全部内容复制到源程序中#include 指令的位置，形成新的源程序，如图 6.5 所示。

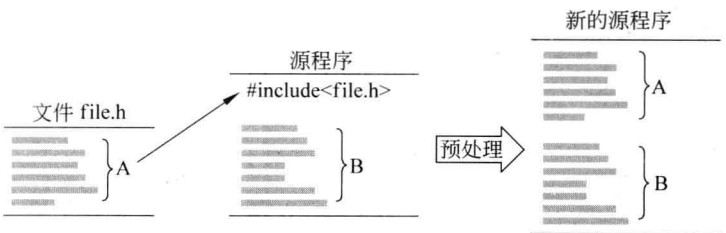


图 6.5　文件包含的预处理过程

头文件一般包含符号常量定义、类型定义以及函数原型等。图 6.6 所示为头文件 stdio.h 的部分内容，可以看到该头文件中包含函数 printf 的原型，用#include 指令包含到源程序中，就相当于进行了函数声明，这就是为什么在使用库函数时要包含相应的头

文件。

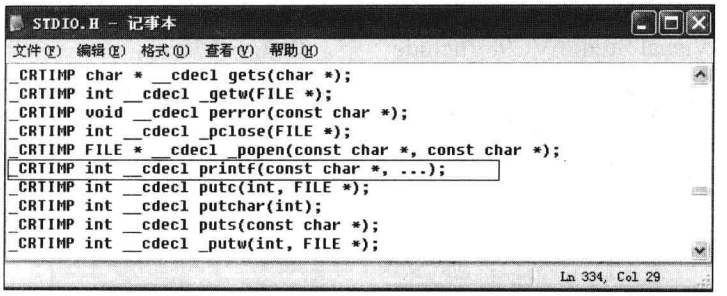


图 6.6　头文件 stdio.h 的内容

6.2.2　标准输入输出函数

标准输入输出函数主要包括字符数据的输入输出函数 getchar 和 putchar、格式化输入输出函数 scanf 和 printf，以及字符串数据的输入输出函数 gets 和 puts。本节介绍字符数据的输入输出函数和格式化输入输出函数，字符串数据的输入输出函数将在第 9 章介绍。

1. 字符数据的输入输出函数

（1）字符数据的输出函数 putchar

【函数原型】　putchar 函数的原型如下：

```
int putchar(表达式)
```

其中，表达式的计算结果是 int 型或 char 型。

【语义】　将表达式的值转换成字符类型并输出到终端（标准输出终端是显示器）。putchar 函数不仅可以输出可显示的字符，还可以输出转义字符。

（2）字符数据的输入函数 getchar

【函数原型】　getchar 函数的原型如下：

```
int getchar( )
```

没有形参

【功能】　接收从终端（标准输入终端是键盘）输入的一个字符，读入的字符一般要保存到一个字符型变量中。

例 6.5　在键盘上输入一行文字，统计这行文字的字符个数并将这行文字原样输出。

解：逐个读入每一个字符，累加字符个数并输出该字符。

【算法】　设变量 ch 接收从键盘上输入的字符，变量 count 累加字符个数，算法如下：

伪代码

```
step1: 初始化累加器 count = 0;
step2: ch = 读入第一个字符;
step3: 当 ch 不是回车符时,循环执行下述操作:
    step3.1: count++;
    step3.2: 显示字符 ch;
    step3.3: ch = 读入下一个字符;
step4: 输出 count;
```

【程序】 getchar 和 putchar 只用于输入输出字符数据，注意没有格式控制。程序如下：

```
1   /*   example6-5.cpp   */
2   #include <stdio.h>                  //使用库函数 printf、getchar 和 putchar
3                                       //空行,以下是主函数
4   int main( )
5   {
6     char ch;
7     int count = 0;                    //count 存储字符个数,初始化为 0
8     printf("请输入一行文字: ");       //输出提示信息
9     ch = getchar( );                  //从键盘接收一个字符,注意括号不能省略
10    while (ch != '\n')                //当输入的字符不是回车符时执行循环
11    {
12      count++;
13      putchar(ch);                    //输出字符 ch
14      ch = getchar( );                //从键盘接收一个字符
15    }
16    putchar('\n');                    //输出换行符,光标停到下一行的起始位置
17    printf("这行文字有%d 个字符\n", count);
18    return 0;                         //将 0 返回操作系统,表明程序正常结束
19  }
```

运行结果如下（下划线为用户输入）：

```
请输入一行文字：我喜欢编程
我喜欢编程
这行文字有 10 个字符
```

2. 格式化输入输出函数

（1）格式化输出函数 printf

【函数原型】 printf 函数的原型如下：

```
int printf("格式控制", 输出列表)
```

其中，格式控制包含两类字符，一类是由%开头的格式控制符，用于将输出列表上的输出项进行格式转换，表 6.1 给出了常用的格式控制符；另一类是普通字符（除格式控制符外均是普通字符），直接输出。输出列表是由逗号分隔的若干输出项，每个输出项可以是变量或表达式。

【语义】 按照格式控制的要求，在终端上输出各个输出项的值。

表 6.1　printf 函数的格式控制符

输出数据	格式控制符	输 出 格 式	举　例
整数	%d	以十进制形式输出一个整数（正数不输出符号）	int a = 5;　printf("%d", a) ;
	%md		int a = 5;　printf("%3d", a) ;
	%–md		int a = 5;　printf("%–3d", a) ;
	%ld	以十进制形式输出一个长整数	long a = 5;　printf("%ld", a) ;
	%o（字母 o）	以八进制形式输出一个整数	int a = 5;　printf("%o", a) ;
	%x	以十六进制形式输出一个整数	int a = 5;　printf("%x", a) ;
无符号整数	%u	以十进制形式输出一个无符号整数	unsigned a = 5;　printf("%u", a) ;
	%mu		unsigned a = 5;　printf("%3u", a) ;
	%–mu		unsigned a = 5;　printf("%–3u", a) ;
实数	%f	以小数形式输出一个实数	double x = 3.5;　printf("%f", x) ;
	%m.nf		double x = 3.5;　printf("%5.2f", x) ;
	%–m.nf		double x = 3.5;　printf("%–5.2f", x) ;
	%e	以指数形式输出一个实数	double x = 3.5;　printf("%e", x) ;
字符	%c	输出字符	char ch = 'a';　printf("%c", ch) ;
	%mc		char ch = 'a';　printf("%3c", ch) ;
	%–mc		char ch = 'a';　printf("%–3c", ch) ;
字符串	%s	输出字符串	char *str = "abc";　printf("%s", str) ;
	%ms		char *str = "abc";　printf("%5s", str) ;
	%–ms		char *str ="abc";　printf("%–5s", str) ;

说明：（1）负号“–”表示该输出项以左对齐方式输出，缺省则表示以右对齐方式输出；（2）m 表示字段宽度，即该输出项所占字符的个数；（3）n 表示小数部分所占字符的个数；（4）如果输出项的实际位数小于 m，则用空格补足，如果大于 m，则按实际的位数输出。

格式控制中的格式控制符用于将输出列表中的输出项进行格式转换，而且格式说明符应该和输出列表中的输出项在数量和类型上一一对应。换言之，应该根据输出列表的输出需要来安排相应的格式控制符。例如：

```
int a = 5, b = 10;
double x = 3.5, y = 2.5;
printf("a = %d, b = %d, x = %5.2f, y = %5.2f\n", a, b, x, y);
```

（2）格式化输入函数 scanf

【函数原型】　scanf 函数的原型如下：

```
int scanf("格式控制", 地址列表)
```

变量的地址列表

其中，格式控制是由格式控制符组成的一个常量字符串，表 6.2 给出了常用的格式控制

符；地址列表由逗号分隔的若干个变量地址组成，每个变量地址是在变量名的前面加上取地址运算符“&”。

【功能】 按照格式控制的要求，将从终端输入的数据赋给地址列表中的各个变量。

表 6.2 scanf 函数的格式控制符

格式控制符	对输入的要求	举　例
%d	输入一个十进制整数	int a; scanf("%d", &a);
%x	输入一个十六进制整数	int a; scanf("%x", &a);
%o（字母 o）	输入一个八进制整数	int a; scanf("%o", &a);
%u	输入一个十进制无符号整数	unsigned int a; scanf("%u", &a);
%c	输入一个字符	char ch; scanf("%c", &ch);
%s	输入一个字符串	char str[80]; scanf("%s", str);
%f	以小数形式输入一个单精度实数	float x; scanf("%f", &x);
%e	以指数形式输入一个单精度实数	float x; scanf("%e", &x);
%lf	以小数形式输入一个双精度实数	double x; scanf("%lf", &x);
%le	以指数形式输入一个双精度实数	double x; scanf("%le", &x);

格式控制符必须与地址列表上的变量类型相匹配，换言之，应该根据输入列表上相应变量的需要来安排相应的格式控制符。例如：

```
int a, b;
char c;
float d;
scanf("%d %d %c %f", &a, &b, &c, &d);
```

如果两个格式控制符之间没有分隔符，则在输入时需要用空格、回车或 Tab 键作为两个输入数据之间的间隔。例如，对于格式输入语句：

```
scanf("%d%d%d", &a, &b, &c);
```

则下面的输入方式（□表示空格，<Enter>表示回车）都是正确的：

方式一：1□2□3 <Enter>

方式二：1 < Enter>

2 < Enter >

3 < Enter >

方式三：1（按 Tab 键）2（按 Tab 键）3 < Enter >

在格式控制中出现的除格式控制符之外的其他字符（称为普通字符）不能被输出，则在输入时需要在对应位置输入这些字符。例如，对于格式输入语句：

```
scanf("a = %d, b = %d, c = %d", &a, &b, &c);
```

则数据的输入形式为：a = 1, b = 2, c = 3 < Enter >。

良好的编程习惯 6.6

为了减少不必要的输入，在 scanf 函数的格式控制中尽量不要出现普通字符，尤其不能将提示信息加入格式控制中，一般在 scanf 函数之前用 printf 函数输出提示信息。

6.2.3 数学函数

在 VC++编程环境下，使用数学库函数需要在源程序中包含头文件 math.h，具体请参见附录 C。

例 6.6 已知三角形的三条边长分别为 a、b、c，求这个三角形的面积。

解：设三角形的三条边长分别为 a、b、c，求三角形面积的公式为：

$$M=\sqrt{s(s-a)(s-b)(s-c)} \quad \text{其中} \quad s=\frac{a+b+c}{2} \tag{6.1}$$

【算法】 设函数 TriAngle 实现求三角形的面积，其算法描述如下：

输入：三角形的边长 a、b 和 c

功能：求三角形的面积

输出：以 a、b 和 c 为边长的三角形的面积

伪代码

```
step1: 如果 a、b 和 c 不能构成一个三角形,则返回 0; 算法结束;
step2: s = (a + b + c)/2;
step3: s = s * (s - a) * (s - b) * (s - c);
step4: area = sqrt(s);
step5: 返回 area;
```

【程序】 首先用变量 x、y 和 z 接收从键盘输入的三个实数，然后调用函数 TriAngle 计算三角形面积，求平方根可以用库函数 sqrt，程序如下：

```
1   /*   example6-6.cpp   */
2   #include <stdio.h>                          //使用库函数 printf 和 scanf
3   #include <math.h>                           //使用库函数 sqrt
4   double TriAngle(double a, double b, double c);      //函数声明
5                                               //空行,以下是主函数
6   int main( )
7   {
8     double x, y, z, area;
9     printf("请输入三角形三条边的边长: "); //输出提示信息
10    scanf("%lf %lf %lf", &x, &y, &z);
                                                //从键盘接收三个 double 型实数
11    area = TriAngle(x, y, z);
                                    //函数调用,x、y、z 为实参,area 接收返回值
12    if (area == 0)
13      printf("输入的数据不能构成三角形\n");
14    else
```

```
    printf("三角形的面积为: %6.2f\n", area);
  return 0;                    //将 0 返回操作系统,表明程序正常结束
}
                               //空行,以下是其他函数定义
double TriAngle(double a, double b, double c)
{
  double s, area;              //s 暂存计算的中间结果,area 为三角形的面积
  if ((a + b <= c) || (a + c <= b) || (b + c <= a))
    return 0;                  //返回 0,其含义是三条边不能构成一个三角形
  s = (a + b + c)/2;
  s = s * (s - a) * (s - b) * (s - c);
  area = sqrt(s);              //调用库函数,参数和返回值均为 double 型
  return area;                 //结束函数 TriAngle,并将 area 返回给调用者
}
```

运行结果 1（下划线为用户输入）：

```
请输入三角形三条边的边长：2 3 4
三角形的面积为:  2.90
```

运行结果 2（下划线为用户输入）：

```
请输入三角形三条边的边长：2 3 6
输入的数据不能构成三角形
```

6.2.4 随机函数

产生随机数的方法通常采用线性同余法，即随机数序列 $a_0, a_1, \cdots, a_n$ 满足：

$$\begin{cases} a_0 = \text{seed} \\ a_n = (ba_{n-1} + c) \bmod m \quad n = 1,2,\cdots \end{cases} \tag{6.2}$$

其中，$b \geqslant 0$，$c \geqslant 0$，$m > 0$，seed $\leqslant m$。seed 称为**随机种子**，当 b、c 和 m 的值确定后，给定一个随机种子，由式（6.2）产生的随机数序列也就确定了。换言之，如果随机种子相同，则会产生相同的随机数序列。所以，严格地说，随机数只是一定程度上的随机，随机数实际上应该称为**伪随机数**。

C/C++语言提供了随机数生成函数 rand，该函数返回 0～RAND_MAX 的一个随机整数，RAND_MAX 是在头文件 stdlib.h 中定义的符号常量，其值为 32767。为了产生不同的随机数，可以调用库函数 srand 初始化随机种子，例如可以调用库函数 time 得到当前的系统时间，将当前系统时间作为随机种子。库函数 rand 和 srand 都在头文件 stdlib.h 中，取系统时间函数 time 在头文件 time.h 中。具体的函数原型如下：

① int rand( )：产生并返回一个随机整数。

② void srand(unsigned int seed)：提供用于 rand( )函数的随机种子值。

③ unsigned int time(NULL)：获得当前系统时间，其中 NULL 是在头文件 stdio.h 中定义的符号常量，其 ASCII 码值为 0。

以当前系统时间作为随机种子的函数调用如下：

```
srand(time(NULL));
```

例 6.7 产生 5 个 0～99 之间的随机数。

解：首先调用函数 srand(time(NULL))初始化随机种子，然后调用随机函数 rand 产生随机数。由于该随机数是在 0～RAND_MAX 的一个整数，则 rand( )%100 即可将其映射到 0～99 之间。程序如下：

```
/*   example6-7.cpp   */
#include <stdio.h>          //使用库函数printf、scanf和符号常量NULL
#include <stdlib.h>         //使用库函数 rand 和 srand
#include <time.h>           //使用库函数 time
                            //空行，以下是主函数
int main( )
{
  int i, x;                 //i 为循环变量
  srand(time(NULL));        //初始化随机种子为当前系统时间
  for (i = 1; i <= 5; i++)  //已知循环次数，用计数型循环
  {
    x = rand( ) % 100;   //产生[0,99]之间的随机整数，注意不能省略括号
    printf("第%2d 个随机数是%2d\n", i, x);
  }
  return 0;                 //将 0 返回操作系统，表明程序正常结束
}
```

运行结果如下：

```
第 1 个随机数是 8
第 2 个随机数是 37
第 3 个随机数是 28
第 4 个随机数是 98
第 5 个随机数是 34
```

6.2.5 解决任务 6.2 的程序

```
/*   duty6-2.cpp   */
#include <stdio.h>          //使用库函数 printf 和 scanf
#include <math.h>           //使用库函数 sqrt
int Prime(int x);           //函数声明
                            //空行，以下是主函数
int main( )
{
  int a;
  printf("请输入一个整数：");  //输出提示信息
  scanf("%d", &a);          //从键盘接收一个整数
  if (Prime(a))             //调用函数 Prime，a 为实参，返回值为逻辑值
    printf("%d 是素数\n", a);
  else
    printf("%d 不是素数\n", a);
  return 0;                 //将 0 返回操作系统，表明程序正常结束
}
                            //空行，以下是其他函数定义
```

```
18    int Prime(int x)                          //函数定义,x 为形参,返回值为 int 型
19    {
20      int i, n;                               //i 为循环变量
21      n = sqrt(x);          //为提高程序效率,在循环体外面调用一次 sqrt 函数
22      for (i = 2; i <= n; i++)                //从 2 开始试除,一直试到 sqrt(x)
23      {
24        if (x % i == 0)                       //能够整除,则 x 不是素数
25          return 0;                           //结束函数 Prime,并返回 0 给调用者
26      }
27      return 1;                               //结束函数 Prime,并返回 1 给调用者
28    }
```

运行结果 1（下划线为用户输入）：

```
请输入一个整数：37
37 是素数
```

运行结果 2（下划线为用户输入）：

```
请输入一个整数：91
91 不是素数
```

6.3 变量的作用域

程序中需要定义一些变量，有些变量可以在整个程序中引用，有些变量只能在某个局部范围（例如函数内部）内引用，变量可以引用的范围称为**变量的作用域**。变量的作用域取决于该变量的定义位置，一般可以将变量的作用域分为局部变量和全局变量两种。变量的作用域是一个静态概念，是从程序的行文角度描述变量的存在性。

【任务 6.3】鸡兔同笼问题（全局变量版）

【问题】 鸡有 2 只脚，兔子有 4 只脚。假设笼子里共有 M 只头 N 只脚，问鸡和兔子各有多少只？要求用函数实现。

【想法】 同任务 5.3。可以将求解方程组的功能独立为函数。

【算法】 设函数 CRP 实现方程组的求解，其算法描述如下：

输入：头的个数 M，脚的个数 N

功能：求解鸡兔同笼问题

输出：鸡的个数 chicken，兔子的个数 rabbit

伪代码

```
step1: chicken 从 0~M 重复执行下述操作:
    step1.1: rabbit = M - chicken;
    step1.2: 如果(2 * chicken + 4 * rabbit 等于 N),则跳出循环;
    step1.3: chicken++;
step2: 如果是提前跳出循环,则输出 chicken 和 rabbit 的值;
       否则输出"无解";
```

如果函数只有一个计算结果，则可以在函数体中使用 return 语句将函数的计算结果传递给调用者。如果函数有两个以上的计算结果，如何将函数的多个计算结果传递给调

用者呢？全局变量可以保存函数的多个计算结果，从而避免在函数间传递数据。

6.3.1 局部变量

局部变量是在函数内部、复合语句或语句块的内部定义的变量，其作用域仅限于函数、复合语句或语句块，离开作用域后再引用局部变量是非法的。

1. 函数级局部变量

在函数首部定义的形参、函数体定义的变量都属于函数级局部变量。例如，在如下程序中，形参 x 和变量 y 都属于函数级局部变量，其作用域仅限于 Fun 函数；变量 a 和 b 也是局部变量，其作用域仅限于 main 函数。因此，在 main 函数中引用变量 x 和 y 是非法的，在函数 Fun 中引用变量 a 和 b 也是非法的。

```
#include <stdio.h>
int Fun(int x);
int main( )
{
  int a = 10, b;
  b = Fun(a);
  printf("%d", x);  //编译错误，变量 x 只能在函数 Fun 中引用
  return 0;
}
int Fun(int x)
{
  int y;
  y = x * x + 2 * x + 1;
  printf("%d", a);  //编译错误，变量 a 只能在函数 main 中引用
  return y;
}
```

局部变量 a 和 b 的作用域

局部变量 x 和 y 的作用域

由于函数级局部变量的作用域仅限于各自的函数，因此，允许在不同的函数中定义同名变量。它们在内存中占据不同的内存单元，属于不同的作用域，因此互不干扰。例如，如下程序在 main 函数和 Fun 函数都定义了局部变量 x 和 y，但是它们代表不同的变量，如图 6.7 所示。

```
#include <stdio.h>
int Fun(int x);
int main( )
{
  int x = 10, y;
  y = Fun(x);
  printf ("%d", y);
  return 0;
}
int Fun(int x)
{
```

同名变量，占用不同的内存单元，属于不同的作用域，互不干扰

```
    int y;
    y = x * x + 2 * x + 1;
    return y;
}
```

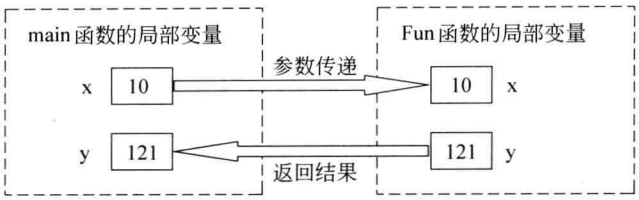


图 6.7　不同函数中的同名局部变量互不干扰

2. 复合语句级局部变量

在复合语句中定义的变量属于复合语句级局部变量，其作用域仅限于花括号括起的复合语句。例如，在如下程序中，变量 temp 属于复合语句级局部变量，因此，在复合语句外引用变量 temp 是非法的。

```
#include <stdio.h>
int main( )
{
  int a = 5, b = 10;
  {
    int temp;
    temp = a; a = b; b = temp;
  }
  printf("%d", temp);
  return 0;
}
```

局部变量 temp 的作用域

//编译错误,对变量 temp 的访问超出其作用域

3. 语句块级局部变量

在语句块中定义的变量属于语句块级局部变量，常见的语句块是循环语句、switch 语句等。例如，在如下程序中，循环变量 i 的作用域仅限于 for 语句。

```
#include <stdio.h>
int main( )
{
  int n = 5, fac = 1;
  for (int i = 1; i <= n; i++)
  {
    fac = fac * i;
  }
  printf("%d 的阶乘等于%d\n", n, fac);
  return 0;
}
```

局部变量 i 的作用域

局部变量 n 和 fac 的作用域

良好的编程习惯 6.7

为了保证程序的清晰性，尽量不要在函数、复合语句或语句块的中间定义局部变量，局部变量一般定义在函数、复合语句或语句块的开始处。

6.3.2 全局变量

全局变量（也称全程变量）是定义在所有函数（包括 main 函数）之外的变量，其作用域从变量定义开始到程序结束。全局变量不属于哪一个函数，可以被作用域内的所有函数引用。例如，在如下程序中，全局变量 Sum 的作用域为整个源程序，主函数 main 和函数 Accumulate 均可以访问全局变量 Sum，每次调用函数 Accumulate 均是对全局变量 Sum 进行累加操作。

```
#include <stdio.h>
int Sum = 0;
void Accumulate( );
int main( )
{
  Accumulate( );
  printf("调用 1 次累加和: %d\n", Sum);     //Sum 的值为 10
  Accumulate( );
  printf("调用 2 次累加和: %d\n", Sum);     //Sum 的值为 20
  return 0;
}
void Accumulate( )
{
  for (int i = 1; i <= 10; i++)
    Sum++;
}
```

全局变量 Sum 的作用域

全局变量可以加强函数之间的数据联系，但由于这些函数依赖于全局变量，使得函数的独立性降低。从模块化程序设计的观点来看这是不利的，因此应尽量避免使用全局变量。

良好的编程习惯 6.8

全局变量可以定义在程序的开始，也可以定义在两个函数的中间，还可以定义在程序的结尾。为了保证程序的清晰性，全局变量一般定义在程序的最前面，即第一个函数之前。为了便于区别全局变量和局部变量，一般将全局变量的首字母大写，例如 Sum。

6.3.3 解决任务 6.3 的程序

首先接收从键盘输入的两个整数，然后调用函数 CRP 求解鸡兔同笼问题满足的方程组。由于将变量 Chicken 和 Rabbit 定义为全局变量，因此，函数 CRP 只有两个参数 M

和 N，以值传递方式实现算法的输入。程序如下：

```
/*   鸡兔同笼（全局变量版）.cpp   */
#include <stdio.h>                          //使用库函数 printf 和 scanf
int Chicken = 0, Rabbit = 0;                //全局变量定义并初始化
void CRP(int M, int N);                     //函数声明
                                            //空行,以下是主函数
int main( )
{
  int M, N;
  printf("请输入头的个数和脚的个数: "); //输出提示信息
  scanf("%d%d", &M, &N);                    //从键盘接收两个整数
  CRP(M, N);              //调用函数 CRP,没有返回值,结果存储在全局变量中
  if (Chicken !=0||Rabbit !=0)  //Chicken 和 Rabbit 至少有一个不为 0
    printf("鸡有%d 只，兔子有%d 只\n", Chicken, Rabbit);
  else
    printf("输入数据不合理,无解\n");
  return 0;                             //将 0 返回操作系统,表明程序正常结束
}
                                            //空行,以下是其他函数定义
void CRP(int M, int N)                      //函数定义,M 和 N 为形参
{
  for (Chicken = 0; Chicken <= M; Chicken++)
                                            //Chicken 为循环变量
  {
    Rabbit = M - Chicken;                   //满足方程 Rabbit + Chicken = M
    if (2 * Chicken + 4 * Rabbit == N) break;
                                            //方程组已解,跳出循环
  }
  if (Chicken > M)                          //循环执行完也没找到方程组的解
  {
    Chicken = 0; Rabbit = 0;
  }
  return;                                   //结束函数 CRP
}
```

运行结果同任务 5.3。

6.4 变量的生存期

程序中的变量都占有一定的内存单元，当变量占有内存时就可以引用这个变量，但并不是所有的变量在程序运行的整个期间都占有内存。为了节省内存空间，在程序的运行过程中，只有在必要时才为变量分配内存空间，即生成这个变量；当变量没有必要存在时，系统将释放该变量占有的内存空间，即撤销这个变量，变量被撤销后就不能再引用这个变量。变量从生成到撤销的这段时间称为**变量的生存期**。变量的生存期是一个动态

概念，是从程序的运行角度描述变量的存在性。

C/C++程序在运行时将具有生存期的变量保存在数据区中，数据区一般分为静态存储区和动态存储区两种，全局变量和静态变量存储在静态存储区中，局部变量和自动变量存储在动态存储区中，如图 6.8 所示。

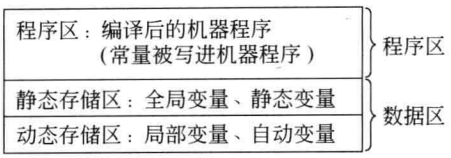


图 6.8　C/C++程序运行时占用内存情况

【任务 6.4】字数统计（静态变量版）

【问题】　从键盘上输入若干行文字，统计出现的字符总数。

【想法】　设变量 sum 累计字符数，主函数 main 调用函数 Count 统计每一行文字的字符数，则每次执行 Count 函数都应该将字符个数累加到同一个变量 sum 中。统计一行文字后询问是否继续输入，如果继续输入，则再次调用函数 Count，否则输出字符数并结束程序。

【算法】　设函数 Count 统计每一行文字的字符数，其算法描述如下：

输入：无

功能：统计一行文字的字符个数

输出：该行字符个数 sum

伪代码

```
step1: ch = 读入一个字符;
step2: 当 ch 不是回车时,循环执行下述操作:
    step2.1: sum++;
    step2.2: ch = 读入下一个字符;
step3: 返回 sum;
```

全局变量和静态变量一经生成就始终占有内存空间，直到程序执行完毕才释放这段内存空间，因此，可以用来保存需要全程操作的数据。

6.4.1　自动变量

自动的含义是在生成变量时系统自动为变量分配内存空间，在撤销变量时系统自动收回变量占用的内存空间，此类变量称为**自动变量**。局部变量都属于自动变量，例如函数级局部变量，在函数调用时生成局部变量，分配相应的内存空间，在函数调用结束后，撤销局部变量并释放其内存空间。由于每调用一次函数都为局部变量重新分配内存空间，因此，如果在程序的运行过程中两次调用同一个函数，则分配给该函数中局部变量的内存地址可能不同。

自动变量用关键字 auto 修饰，auto 缺省时则隐含该变量为自动变量。例如，如下程序中，局部变量 a、b、x、y 都属于自动变量，但通常省略修饰符 auto。

```
#include <stdio.h>
```

```
int Fun(auto int x);                    //形参 x 是自动变量,通常省略修饰符 auto
int main( )
{
  auto int a = 10, b;                   //a 和 b 是自动变量,通常省略修饰符 auto
  b = Fun(a);
  printf("f(10) = %d", b);
  return 0;
}
int Fun(auto int x)                     //形参 x 是自动变量,通常省略修饰符 auto
{
  auto int y;                           //y 是自动变量,通常省略修饰符 auto
  y = x * x + 2 * x + 1;
  return y;
}
```

6.4.2 静态变量

如果希望在函数调用结束后仍保留局部变量的值，即不释放该变量占用的存储单元，则必须将该变量定义为**静态变量**。静态变量一经分配内存空间，在程序的运行过程中就始终占有该内存空间，但静态变量的作用域不变，因此，仍然只能在其作用域范围内引用。静态变量的初始化只在第一次调用函数时执行一次，以后执行函数时不再进行初始化操作。

静态变量用关键字 static 修饰。例如，在如下程序中，函数 Accumulate 的局部变量 sum 被定义为静态变量，则在函数 Accumulate 调用结束后，不释放该变量占用的内存空间。再次调用函数 Accumulate 时，不再为变量 sum 分配内存空间，变量 sum 中保存的是上次调用函数 Accumulate 的计算结果，可以在其值的基础上进行计算。

```
#include <stdio.h>
int Accumulate( );
int main( )
{
  int num;
  num = Accumulate( );
  printf("调用 1 次累加和: %d\n", num);          //num 的值为 10
  num = Accumulate( );
  printf("调用 2 次累加和: %d\n", num);          //num 的值为 20
  printf("%d", sum);                  //编译错误,对变量 sum 的引用超出其作用域
  return 0;
}
int Accumulate( )
{
  static int sum = 0;                 //初始化操作只执行一次
  for (int i = 1; i <= 10; i++)
    sum++;
  return sum;
}
```

静态变量 sum 的作用域

6.4.3 解决任务 6.4 的程序

从键盘上输入数据时，不是键入一个字符就送入键盘缓冲区，而是按 Enter（回车）键后才送入键盘缓冲区。为了保证正确接收从键盘输入的字符，C/C++语言提供了清除键盘缓冲区函数 fflush(stdin)，其函数原型在头文件 stdio.h 中。在函数 Count 开始读入字符之前，要调用函数 fflush(stdin)清除键盘缓冲区。多行文字要累加到同一个变量中，因此，函数 Count 中将变量 sum 设为静态变量。程序如下：

```
/*    duty6-4.cpp   */
#include <stdio.h>          //使用库函数 printf、scanf、getchar 和 fflush
int Count( );               //函数声明
                            //空行,以下是主函数
int main( )
{
  int sum;        //与函数 Count 中的静态变量 sum 同名,但是两个不同的变量
  char ch;
  do
  {
    printf("请输入一行文字: ");     //输出提示信息
    sum = Count( );                  //累加一行文字，sum 接收累加结果
    printf("继续吗? ");
    scanf("%c", &ch);                //从键盘接收一个字符
  } while (ch == 'y' || ch == 'Y'); //当 ch 的值是'y'或'Y'时执行循环
  printf("字符数: %d\n", sum);     //输出结果
  return 0;                        //将 0 返回操作系统,表明程序正常结束
}
                                   //空行,以下是其他函数定义
int Count( )                       //函数定义,无参函数,返回值为 int 型
{
  static int sum = 0;              //静态变量,初始化一次并始终占有内存
  char ch;
  fflush(stdin);                   //清除键盘缓冲区
  ch = getchar( );                 //读入一个字符
  while (ch != '\n')               //不是回车,累加字数并继续读入
  {
    sum++;
    ch = getchar( );
  }
  return sum;                  //结束函数 Count,并将 sum 返回给调用者
}
```

运行结果如下（下划线为用户输入）：

```
请输入一行文字: I am a chinese,
继续吗? y
请输入一行文字: I love China!
继续吗? n
字符数: 28
```

6.5 程序设计实例

6.5.1 实例 1——三角函数表

【问题】 打印三角函数表，要求以角度为单位输出对应的 sin 值和 cos 值。

【想法】 设定起始角度、终止角度和计算步长，例如从 10°到 90°、每隔 2°输出对应的 sin 值和 cos 值。可以调用库函数求 sin 值和 cos 值，但是库函数 sin(x)和 cos(x)要求参数 x 是弧度而不是角度，需要进行角度到弧度的转换，转换公式如下：

$$\text{弧度} = \text{角度} \times \pi / 180 \tag{6.3}$$

【算法】 设函数 TriTable 完成打印三角函数表，其算法描述如下：

输入：起始角度 startDegree，终止角度 endDegree 和计算步长 step

功能：打印三角函数表

输出：无

伪代码

```
step1: 打印表头 x、sin(x)和 cos(x);
step2: 循环变量 i 从 startDegree 到 endDegree,重复执行下述操作:
    step2.1: 根据式(6.3)将角度 i 转换为弧度;
    step2.2: 调用库函数 sin 和 cos 计算三角函数值;
    step2.3: 打印计算结果;
    step2.4: i = i + step;
```

【程序】 能够确定循环变量 i 的初值、终值和循环步长，可以用 for 循环实现算法。程序如下：

```
1   /*   三角函数表.cpp   */
2   #include <stdio.h>                  //使用库函数 printf 和 scanf
3   #include <math.h>                   //使用库函数 sin 和 cos
4   #define PI 3.14                     //定义符号常量
5   void TriTable(int startDegree, int endDegree, int step);
                                        //函数声明
6                                       //空行,以下是主函数
7   int main( )
8   {
9     int startDegree, endDegree, step;
10    printf("请输入起始角度、终止角度和步长: ");     //输出提示信息
11    scanf("%d %d %d", &startDegree, &endDegree, &step);
                                        //从键盘接收三个整数
12    TriTable(startDegree, endDegree, step);
13    return 0;                         //将 0 返回操作系统,表明程序正常结束
14  }
15                                      //空行,以下是其他函数定义
```

```
16    void TriTable(int startDegree, int endDegree, int step)
                                              //函数定义
17    {
18      double sinD, cosD;
19      printf("x\tsin(x)\tcos(x)\n");     //打印表头,'\t'为跳格
20      printf("================\n");      //打印分隔线
21      for (int i = startDegree; i <= endDegree; i = i + step)
22      {
23        sinD = sin(i * PI / 180);    //调用库函数求 sin 值,注意参数是弧度
24        cosD = cos(i * PI / 180);    //调用库函数求 cos 值,注意参数是弧度
25        printf("%d\t%5.2f\t%5.2f \n", i, sinD, cosD);
                                      //'\t'为跳格,对齐打印
26      }
27      printf("================\n"); //打印表格底部分隔线
28      return;                       //结束函数 TriTable
29    }
```

运行结果如下（下划线为用户输入）：

```
请输入起始角度、终止角度和步长：10  90  20
x      sin(x)  cos(x)
=================
10     0.17    0.98
30     0.50    0.87
50     0.77    0.64
70     0.94    0.34
90     1.00    0.00
=================
```

6.5.2 实例 2——猜数游戏

【问题】 首先由计算机产生一个随机数，并给出这个随机数所在的区间，然后由游戏者来猜测这个数。如果游戏者给出的数比这个随机数大，则显示“大了，请重新猜!”；如果游戏者给出的数比这个随机数小，则显示“小了，请重新猜!”；如果游戏者给出的数正好等于这个随机数，则显示“恭喜，猜对了!”，并给出猜测的次数；如果游戏者猜测次数超过 8 次，则显示“超过次数，游戏结束!”。

【想法】 设变量 secret 表示计算机给出的随机数，变量 guess 表示游戏者给出的数，则 guess 和 secret 进行比较，有以下三种情况：

① guess 大于 secret，则显示“大了，请重新猜!”，游戏者重新给出一个数；

② guess 小于 secret，则显示“小了，请重新猜!”，游戏者重新给出一个数；

③ guess 等于 secret，则显示“恭喜，猜对了!”。

重复上述过程，直到游戏者猜中或超过规定次数。

【算法】 设函数 Guess 实现猜数游戏，其算法描述如下：

输入：随机数 secret

功能：执行游戏过程

输出：无

伪代码

```
step1: 初始化猜测次数 count = 0;
step2: 重复执行下述操作,直到 count 达到 8 次:
    step2.1: 游戏者输入一个数 guess;
    step2.2: count++;
    step2.3: 如果 guess 等于 secret,则跳出循环;
    step2.4: 如果 guess 大于 secret,则显示"大了,请重新猜!";
             否则,显示"小了,请重新猜!";
step3: 如果 count 达到 8 次,则显示"超过次数,游戏结束!";
       否则,显示"恭喜,猜对了!";并显示计数器 count 的值;
```

【程序】 首先由计算机产生一个随机数 secret，然后调用函数 Guess 实现猜数游戏，在每次猜数之前，都给出区间提示。程序如下：

```
1   /*   猜数游戏.cpp    */
2   #include <stdio.h>              //使用库函数printf、scanf和符号常量NULL
3   #include <stdlib.h>             //使用库函数 srand 和 rand
4   #include <time.h>               //使用库函数 time
5   void Guess(int secret);         //函数声明
6                                   //空行,以下是主函数
7   int main( )
8   {
9     int secret = 0;
10    srand(time(NULL));            //用当前系统时间初始化随机种子
11    secret = 1 + rand( ) % 100;   //产生一个 1～100 之间的随机数
12    Guess(secret);                //调用 Guess 函数开始游戏
13    return 0;                     //将 0 返回操作系统,表明程序正常结束
14  }
15                                  //空行,以下是其他函数定义
16  void Guess(int secret)
17  {
18    int guess, count = 0;         //count 为计数器,累计猜测次数
19    int low = 1, high = 100;      //初始化猜数区间[low, high]
20    do
21    {
22      printf("请输入一个%d ~ %d之间的整数: ", low, high);
                                    //提示猜数区间
23      scanf("%d", &guess);            //从键盘接收一个整数
24      count++;                    //猜测次数加 1
25      if (guess == secret)            //猜中则强制跳出循环
26        break;
27      if (guess > secret)
28      {
29        high = guess - 1;         //调整猜数区间
30        printf("大了,请重新猜! \n");
```

```
        }
        else
        {
          low = guess + 1;              //调整猜数区间
          printf("小了,请重新猜! \n");
        }
      } while (count < 8);              //当 count 小于 8 时执行循环
      if (count >= 8)                   //不是提前跳出循环
        printf("超过次数,游戏结束! \n");
      else
        printf("恭喜,猜对了!共猜测%d 次! \n", count);
    }
```

运行结果如下（下划线为用户输入）：

```
请输入一个 1 ~ 100 之间的整数：5
小了,请重新猜!
请输入一个 6 ~ 100 之间的整数：50
大了,请重新猜!
请输入一个 6 ~ 49 之间的整数：30
小了,请重新猜!
请输入一个 31 ~ 49 之间的整数：40
大了,请重新猜!
请输入一个 31 ~ 39 之间的整数：35
小了,请重新猜!
请输入一个 36 ~ 39 之间的整数：37
恭喜,猜对了! 共猜测 6 次!
```

习 题 6

一、选择题

1. 下列说法中正确的是（　　）。

   A. 实参必须是常量`　　B. 形参可以是常量、变量或表达式

   C. 实参可以是任何类型　　D. 实参与其对应形参的数据类型一致

2. 在 C/C++语言中，函数的隐含存储类别是（　　）。

   A. auto　　B. static　　C. extern　　D. 无储存类别

3. 在函数中未指定存储类别的局部变量，其隐含的存储类别为（　　）。

   A. auto　　B. static　　C. extern　　D. register

4. C/C++语言规定，函数返回值的类型是由（　　）。

   A. return 语句中的表达式类型所决定

   B. 调用该函数时的主调函数类型所决定

   C. 调用该函数时系统临时决定

   D. 在定义该函数时所指定的函数类型所决定

5. 下列说法不正确的是（　　）。

A. 在不同的函数中可以使用相同名字的变量

B. 函数中的形参是局部变量

C. 在一个函数内定义的变量只在本函数范围内有效

D. 在一个函数内的复合语句中定义的变量在本函数范围内有效

6. 以下程序的输出结果是（　　）。

```
#include <stdio.h>
void Fun(int x, int y, int z)
{
  z = x + y;
}
int main( )
{
  int c;
  Fun(2, 3, c);
  printf("%d\n", c);
  return 0;
}
```

A. 0　　B. 5　　C. 6　　D. 无法确定

二、简答题

1. C/C++语言提倡小函数构成大程序，如何理解函数的构件作用？

2. C/C++语言为什么提供函数声明机制？

3. 在调用函数时，为什么要进行参数传递？参数传递的具体过程是什么？

4. 头文件的作用是什么？如何知道要调用的库函数在哪个头文件中？

5. 简述随机数的产生原理。

三、程序设计题（要求用函数实现）

1. 输入一个日期，输出这天是该年的第几天。

2. 判断给定的自然数是否为降序数。所谓降序数是指对于 $n = d_1\ d_2\ d_3 \cdots d_k$，满足 $d_i \geqslant d_{i+1}$（$1 \leqslant i \leqslant k-1$）。

3. 从键盘输入 10 个整数，求这 10 个整数中的最大值。

4. 输出 1 和 100 之间的所有素数。

5. 输出华氏—摄氏温度转换表，华氏温度的转换范围是[low, high]，转换增量为 step。

6. 齿轮啮合问题。设有三个齿轮相互衔接，求当三个齿轮中的某两对齿相互衔接后到下一次这两对齿再次相互衔接，每个齿轮最少各转多少圈？

第7章

变量的间接访问——指针

指针是程序设计语言的一个重要概念，指针使得程序在运行时能够获得变量地址，并通过这个地址访问相应的内存单元。并不是每种程序设计语言都提供了指针类型，例如 Java 语言就不允许编程人员通过指针访问内存，C/C++语言提供了指针因而具有很大的灵活性。

7.1 指　　针

【任务 7.1】获取密电码

【问题】 某谍报组织有一个重要信息——密电码存放在银行的一个保险箱 A 中，由于不知道保险箱 A 的编号，所以不能直接获得密电码，但知道保险箱 A 的编号存在于另一个保险箱 B 中，如何通过保险箱 B 获得密电码？

【想法】 设保险箱 A 和 B 分别用变量 key 和 p 表示，保险箱 A 的编号相当于变量 key 的地址，可以通过变量 key 直接获得该内存单元的内容即密电码。现要求不能通过变量 key 直接对该变量所在存储单元进行访问，但变量 key 的存储地址存放在变量 p 中，则可以通过变量 p 取出变量 key 的存储地址，从而实现对变量 key 的间接访问，如图 7.1 所示。

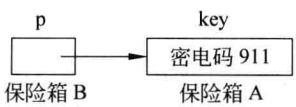


图 7.1　变量 p 中存放变量 key 的存储地址

【算法】 设变量 key 和 p 分别表示保险箱 A 和 B，算法如下：

伪代码

```
step1: 将密电码存放在变量 key 中;
step2: 将变量 key 的存储地址存放在变量 p 中;
step3: 通过变量 p 获得变量 key 的值;
```

实现上述算法需要解决如下两个问题：

① 变量 p 用来保存变量 key 的存储地址，那么，如何在程序运行时获得变量 key 的存储地址？

② 如何通过变量 p 实现对变量 key 的间接访问？

7.1.1 指针的概念

为了正确理解指针，首先必须在机器层面理解变量在内存中的存储方式和访问方式，深刻理解变量地址的含义。

在程序中定义了一个变量，编译程序会根据该变量的数据类型在内存中分配相应的存储单元，该存储单元的起始地址就是这个变量的地址，变量的存储地址由变量名表示，称为变量的**左值**。该存储单元存储的数据就是这个变量的值，变量的值也由变量名表示，称为变量的**右值**。理解起来，变量在赋值运算符“=”的左侧体现其左值特征——变量的地址，在赋值运算符“=”的右侧体现其右值特征——变量的值。可以通过变量名对变量所占存储单元进行访问（即存取操作），这种访问方式称为**变量的直接访问**。例如，如下语句实现对变量 num 的直接访问：

```
int num = 100;          //对变量 num 进行存操作,num 体现左值特征——变量的地址
int sum;
sum = num;              //对变量 num 进行取操作,num 体现右值特征——变量的值
```

编译器为变量 num 分配相应存储单元，假设变量 num 被分配在 B000 开始的 4 个字节（设 int 型数据占 4 个字节）的内存单元中，则变量 num 的地址就是 B000，变量初始化语句“int num =100;”将值 100 通过变量名 num 存入起始地址为 B000 的存储单元，如图 7.2 所示。

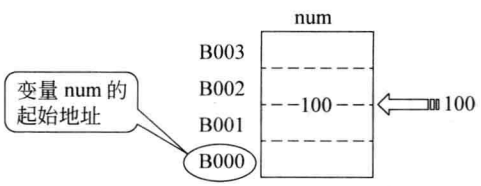


图 7.2　对变量 num 的直接访问——存操作

赋值语句“sum = num;”通过变量名 num 从起始地址为 B000 的存储单元中将变量 num 的值取出，然后存入变量 sum 中，如图 7.3 所示。

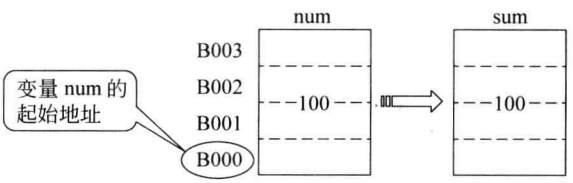


图 7.3　对变量 num 的直接访问——取操作

假设另外有一个变量 p 保存了变量 num 的存储地址，则相当于有一个指针指向了变量 num 的内存单元，如图 7.4 所示。保存地址的变量称为**指针变量**，在不致混淆的情况下，通常将指针变量简称为**指针**。指针 p 保存了变量 num 的存储地址，称指针 p 指向了

变量 num。在程序设计时通常不关心变量的具体地址，因此，指针 p 指向变量 num 通常描述为如图 7.5 所示的形式。

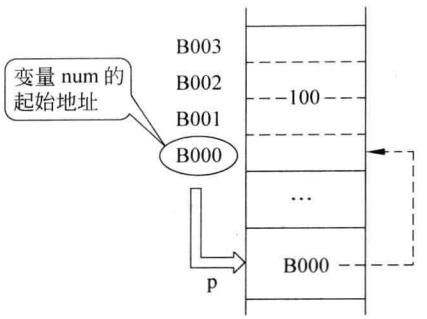


图 7.4　变量 p 存储变量 num 的起始地址

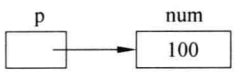


图 7.5　指针 p 指向变量 num

当指针 p 指向了变量 num 时，可以通过指针 p 访问变量 num，C/C++语言提供了间接引用运算符“*”实现对指针所指变量进行间接访问，则可以用*p 实现对指针 p 所指变量进行间接访问，如图 7.6 所示。这种通过指针对变量所在存储单元进行访问的方式称为**变量的间接访问**。

(a) 存操作　　(b) 取操作

图 7.6　通过指针 p 对变量 num 的间接访问

7.1.2　指针变量的定义和初始化

从本质上讲，指针变量的使用方法与普通变量相同，也需要先定义后使用，但是需要注意指针变量在数据类型和值上的特殊性：指针变量的数据类型是指针指向变量的数据类型，而不是指针变量本身的数据类型；指针变量的值是某个变量在内存中的起始地址，因此，**指针就是地址**。

1. 指针变量的定义

【语法】　定义指针变量的一般形式如下：

```
        ┌──── 指针定义符
        ↓
基类型 *指针变量名;
```

其中，基类型是该指针变量指向变量的数据类型，可以是任意合法的数据类型；“*”称为指针定义符，用来说明指针变量以区别于普通变量；指针变量名是合法的标识符。

【语义】 定义一个指向基类型的指针变量。

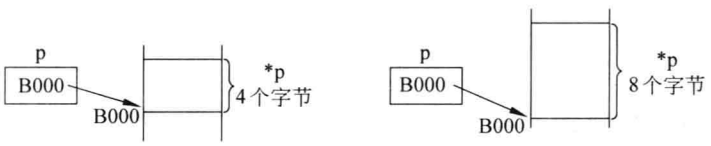
如下语句定义了 int 型变量 num，指向 int 型变量的指针变量 p，以及指向 double 型变量的指针变量 q。注意，指针变量是 p 和 q，而不是*p 和*q，指针定义符“*”用来标识变量 p 和 q 是指针变量。

```
int num, *p;
double *q;
```

良好的编程习惯 7.1

不要将指针变量命名为 x、y、z 或 a、b、c 等容易混淆类型的变量名，为指针变量命名时，常在变量名前加字母 p 或 ptr（pointer），以表示该变量是指针变量。

在定义指针变量时必须指定该指针指向变量的数据类型，这主要是为了便于编译器管理指针变量。由于不同类型的数据在内存中所占的存储单元数不同，指针只是指向了某个存储单元的起始地址，因此，基类型确定了存取数据时应该对多大的存储单元进行操作。例如，如图 7.7 所示，如果指针 p 指向 int 型变量，假设 int 型数据占 4 个字节，则*p 操作的是从地址 B000 开始 4 个字节的数据；如果指针 p 指向 double 型变量，假设 double 型数据占 8 个字节，则*p 操作的是从地址 B000 开始 8 个字节的数据。



(a) 指针 p指向 int型变量　　(b) 指针 p指向 double型变量

图 7.7　指针与所指变量类型的绑定

需要强调的是，指针变量存放的是该指针指向变量的存储地址，内存地址通常是一个无符号整数，因此，所有指针变量都占有相同大小的存储空间，具体占有的存储单元数与计算机系统和编译器有关。

2. 指针变量的初始化

在定义指针变量时，如果没有给指针变量赋初值，则指针变量是“值无定义的”，指针变量的值是一个随机数，可能指向内存中任何位置，这种指针称为**野指针**。野指针在程序中是很危险的，可能会引发系统崩溃。

定义指针变量后，必须将该指针和一个特定的内存地址进行关联，然后才可以使用指针。也就是说，指针变量要处于“值有定义的”状态才可以使用。在定义指针变量的同时赋初值称为指针变量的初始化。

【语法】 初始化指针变量的一般形式如下：

基类型 *指针变量名 = 内存地址;

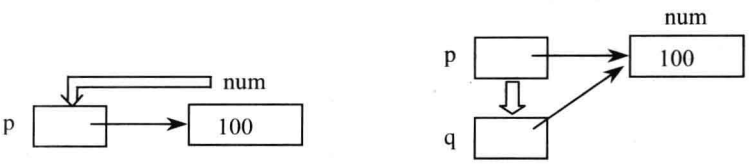
其中，内存地址通常是用取地址运算符“&”获得变量的存储地址，或是另一个已经定

义的指针。

【语义】 定义一个指向基类型的指针变量，并将内存地址存储在指针变量中，其结果是指针变量指向了该内存地址。

例如，如下是合法的指针变量初始化操作，其存储示意图如图 7.8 所示。

```
int num = 100;
int *p = &num;          //&num 为变量 num 在内存中的起始地址
int *q = p;             //p 为指向变量 num 的指针
```



(a) int *p = #　　(b) int *q = p;

图 7.8　指针变量初始化操作示意图

有时，在定义指针变量时无法与某个具体的变量进行关联，可以在定义指针后将其初始化为 NULL（ASCII 码为 0），NULL 称为**空指针**。值为 NULL 的指针不指向任何内存单元，可以避免无法预料的错误发生。例如：

```
int *p = NULL;
```

良好的编程习惯 7.2

任何时刻都不能让指针变量处于“值无定义的”状态，在指针没有指向某个有效对象时，将其定义为空指针 NULL 是一种良好的编程习惯。在 VC++中，符号常量 NULL 定义在 stdio.h 中。

7.1.3　指针变量的赋值

指针变量可以在程序中赋值，即将一个内存地址存入指针变量，其结果是将指针指向该内存地址。

【语法】 有两种常用的赋值方法：

（1）将某个变量的地址赋给指针变量，一般形式如下：

```
指针变量 = 变量地址;
```

其中，变量地址通常是用取地址运算符“&”获得变量的存储地址，其结果是指针变量指向该变量。

（2）将一个指针赋给另一个指针，一般形式如下：

```
指针变量 1 = 指针变量 2;
```

其中，指针变量 2 和指针变量 1 的基类型相同，且指针变量 2 必须是“值有定义的”，其结果是指针变量 1 指向指针变量 2 所指的内存单元。

【语义】 将某个内存地址存入指针变量。

例如，如下是合法的指针变量赋值操作：

```
int num = 100;
int *p = NULL, *q = NULL;   //指针 p 和 q 均初始化为空
p = &num;                   //指针 p 指向变量 num
q = p;                      //指针 q 指向指针 p 所指的内存单元
```

指针变量的相互赋值需要注意赋值的相容性，只有基类型相同的指针变量才可以相互赋值。例如，对于如下指针变量赋值操作，某些编译器会给出警告：

```
int num = 100, *p = NULL;   //p 为指向 int 型变量的指针变量
double *q = NULL;           //q 为指向 double 型变量的指针变量
p = &num;                   //指针 p 指向变量 num
q = p;                      //指针变量 p 和 q 的基类型不同,某些编译器会给出警告
```

通常不允许将一个整数赋给指针变量，由于指针的操作需要编程人员保证其安全性，编译器通常只会给出一个警告。例如：

```
int *p = NULL;              //p 为指向 int 型数据的指针变量
p = 100;                    //编译器只会给出警告
```

其结果是指针 p 指向地址为 100 的内存单元，这是内存的低端，通常用来存储系统资源，对该内存单元的操作可能会引起系统错误，严重的会产生死机等现象。

7.1.4 指针所指变量的间接访问

定义一个指针变量并与某个内存地址关联后，就可以使用间接引用运算符“*”对指针所指变量进行间接访问。

【语法】 间接引用运算符“*”的一般形式如下：

```
*指针变量
↑
间接引用运算符
```

其中，“*”称为间接引用运算符，用来访问该指针变量指向的存储单元；指针变量必须是“值有定义的”，即该指针必须指向某个存储单元。

【语义】 访问指针变量所指存储单元。

实质上，指针变量就是通过间接引用运算符“*”来实现对变量的间接访问。例如，有如下变量定义：

```
int num, sum;
int *p = &num;              //指针变量 p 初始化为指向变量 num
```

则下面两条赋值语句是等价的，其操作示意图如图 7.6(a)所示。

```
num = 100;          //对变量 num 的直接访问——存操作
*p = 100;           //为指针 p 所指变量赋值,对变量 num 的间接访问——存操作
```

下面两条赋值语句也是等价的，其操作示意图如图 7.6(b)所示。

```
sum = num;          //对变量 num 的直接访问——取操作
sum = *p;           //将指针 p 所指变量的值取出,对变量 num 的间接访问——取操作
```

7.1.5 解决任务 7.1 的程序

```
1    /*   duty7-1.cpp   */
2    #include <stdio.h>                    //使用库函数 printf
3                                          //空行,以下是主函数
4    int main( )
5    {
6      int key = 911, *p = NULL; //p 是指向 int 型数据的指针，已初始化为空
7      p = &key;           //将变量 key 的地址存入指针 p，使指针 p 指向变量 key
8      printf("密电码所在保险箱的编号是：%X\n", p);
                                           //以十六进制输出指针 p 的值
9      printf("密电码是：%d\n", *p);       //输出指针 p 所指变量的值
10     return 0;                           //将 0 返回操作系统,表明程序正常结束
11   }
```

运行结果如下：

```
密电码所在保险箱的编号是：12FF7C
密电码是：911
```

7.2 指针作为函数的参数

【任务 7.2】鸡兔同笼问题（函数版）

【问题】 鸡有 2 只脚，兔子有 4 只脚，假设笼子里共有 M 只头 N 只脚，问鸡和兔子各有多少只？要求用函数实现。

【想法】 设鸡有 x 只，兔子有 y 只，则有如下方程组成立：

$$\begin{cases} x + y = M \\ 2x + 4y = N \end{cases}$$

【算法】 设函数 CR 实现求解方程组，算法描述如下：

输入：头的个数 M，脚的个数 N

功能：求解鸡兔同笼问题

输出：鸡的个数 x，兔子的个数 y

```
step1: x从0到M循环执行下述操作:
    step1.1: y = M - x;
    step1.2: 如果(2 * x + 4 * y等于N),则跳出循环;否则x++;
step2: 如果提前跳出循环,则输出x和y的值;
       否则令x = 0, y = 0,输出x和y的值;
```

如果函数只有一个输出结果，则可以在函数体中使用 return 语句将函数的计算结果传递（返回）给调用者。如果函数有两个以上的输出结果，而 return 语句只能返回一个计算结果，如何将函数的多个计算结果返回给调用者呢？

在 C/C++程序中，函数间的调用是以传递参数的方式进行函数内部与外部环境（即调用者）之间的数据交互。从本质上讲，C/C++语言只提供了一种参数传递方式：值传递方式，即函数调用时将实参的值传递给形参。当实参的值是指针（或地址）时，称为指针传递方式，也称为地址传递方式。采用指针传递方式可以将函数的多个计算结果返回给调用者。

7.2.1 值传递方式——函数的输入

值传递方式在函数定义时，将形参定义为普通类型（即非指针类型），实参可以是类型与形参相容的常量、变量或表达式；在函数调用时，系统为形参分配存储空间，然后将实参的值传递到形参中；在调用结束后，系统自动释放形参的存储空间。值传递方式的特点是：被调用函数的执行不会影响函数的实参，即在被调用函数中不能对函数的实参进行修改，因此，通常以值传递方式实现函数的输入。

例 7.1 求两个自然数 *m* 和 *n* 的最小公倍数。要求用函数实现。

解：可以利用两个自然数的最大公约数来求这两个数的最小公倍数。例如，35 和 25 的最大公约数为 5，则 35 和 25 的最小公倍数等于 35 × 25 ÷ 5 = 175。

【算法】 设函数 CommonMultiple 实现求最小公倍数，其算法描述如下：

输入：两个自然数 *m* 和 *n*

功能：求两个自然数的最小公倍数

输出：*m* 和 *n* 的最小公倍数

```
step1: mTemp = m; nTemp = n;          //暂存变量m和n的值
step2: r = m % n;
step3: 执行下述操作,直到r等于0:
    step3.1: m = n;
    step3.2: n = r;
    step3.3: r = m % n;
step4: 返回mTemp * nTemp/n;           //用最初的m和n计算最小公倍数
```

【程序】 函数 CommonMultiple 的参数以值传递方式接收两个自然数，函数的计算

结果为 m 和 n 的最小公倍数，在函数体中用 return 语句返回。程序如下：

```
/*   example7-1.cpp   */
#include <stdio.h>                          //使用库函数 printf 和 scanf
int CommonMultiple(int m, int n);           //函数声明
                                            //空行,以下是主函数
int main( )
{
  int m, n, multiple;
  printf("请输入两个自然数：");
  scanf("%d%d", &m, &n);
  multiple = CommonMultiple(m, n);          //函数调用,m 和 n 是实参
  printf("%d 和%d 的最小公倍数是%d\n", m, n, multiple);
  return 0;                                 //将 0 返回操作系统,表明程序正常结束
}
                                            //空行,以下是其他函数定义
int CommonMultiple(int m, int n)            //函数定义，m 和 n 是形参
{
  int mTemp = m, nTemp = n;                 // mTemp 和 nTemp 暂存 m 和 n 的值
  int r = m % n;
  while (r != 0)
  {
    m = n;
    n = r;
    r = m % n;
  }
  return (mTemp * nTemp / n);               //结束函数,返回表达式的运算结果
}
```

运行结果如下（下划线为用户输入）：

```
请输入两个自然数：35  25
35 和 25 的最小公倍数是 175
```

7.2.2 指针传递方式——函数的输出

指针传递方式在函数定义时，将形参声明为指针类型，实参可以是基类型与形参相容的指针或变量地址；在函数调用时，系统为形参分配存储空间，并将实参的值（即地址）传递到形参中；在调用结束后，系统自动释放形参的存储空间。指针传递方式的特点是：在被调用函数中可以对实参地址所对应的存储单元进行访问，即可以读取或修改该内存单元的值。因此，可以通过指针传递方式实现函数的输出，即将被调用函数修改的值传递（返回）给调用者。

例 7.2 求两个自然数 *m* 和 *n* 的最大公约数和最小公倍数。要求用函数实现。

【算法】 设函数 CommonFactorMultiple 实现求两个自然数的最大公约数和最小公倍数，其算法描述如下：

输入：两个自然数 *m* 和 *n*

功能：求两个自然数的最大公约数和最小公倍数

输出：*m* 和 *n* 的最大公约数和最小公倍数

伪代码

```
step1: mTemp = m; nTemp = n;                    //暂存变量 m 和 n 的值
step2: r = m % n;
step3: 执行下述操作,直到 r 等于 0:
    step3.1: m = n;
    step3.2: n = r;
    step3.3: r = m % n;
step4: 输出最大公约数 n; 输出最小公倍数 mTemp * nTemp / n;
```

【程序】 由于算法需要返回两个计算结果，考虑用指针传递方式返回计算结果。函数 CommonFactorMultiple 有 4 个参数，其中参数 m 和 n 以值传递方式实现算法的输入，参数 p 和 q 以指针传递方式返回两个计算结果，参数传递过程如图 7.9 所示，程序如下：

```
1   /*   example7-2.cpp   */
2   #include <stdio.h>                    //使用库函数 printf 和 scanf
3   void CommonFactorMultiple(int m, int n, int *p, int *q);
                                          //函数声明
4                                         //空行,以下是主函数
5   int main( )
6   {
7     int m, n, factor, multiple;
8     printf("请输入两个自然数: ");
9     scanf("%d%d", &m, &n);
10    CommonFactorMultiple (m, n, &factor, &multiple);
                                          //函数调用,传递 4 个参数
11    printf("%d 和%d 的最大公约数是%d\n", m, n, factor);
12    printf("%d 和%d 的最小公倍数是%d\n", m, n, multiple);
13    return 0;                           //将 0 返回操作系统,表明程序正常结束
14  }
15                                        //空行,以下是其他函数定义
16  void CommonFactorMultiple (int m, int n, int *p, int *q)
17  {           //m、n、p、q 是形参,其中 m 和 n 是值传递,p 和 q 是指针传递
18    int mTemp = m, nTemp = n;           // mTemp 和 nTemp 暂存 m 和 n 的值
19    int r = m % n;
20    while (r != 0)
21    {
22      m = n;
23      n = r;
24      r = m % n;
25    }
26    *p = n;                             //将 n 保存到 p 所指变量
27    *q = mTemp * nTemp / n;             //将表达式的运算结果保存到 q 所指变量
28  }
```

运行结果如下（下划线为用户输入）：

```
请输入两个自然数：35  25
35 和 25 的最大公约数是 5
35 和 25 的最小公倍数是 175
```

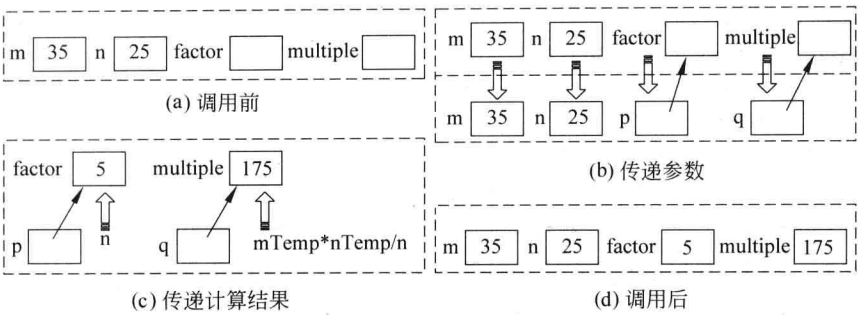


图 7.9　指针传递方式返回计算结果

7.2.3　指针传递方式——函数的输入输出

由于指针传递方式可以在被调用函数中读取或修改实参所指内存单元的值，因此，可以通过指针传递方式实现被调用函数与调用者之间的双向数据传递，即以指针传递方式接收算法的输入，并将计算结果传递给调用者。

例 7.3　交换两个变量的值。要求用函数实现。

解：函数 Swap1 采用值传递方式交换两个变量 x 和 y 的值，被调用函数完成了交换功能，但不能把交换结果传递给调用者，参数传递过程如图 7.10 所示；函数 Swap2 采用指针传递方式交换两个变量 x 和 y 的值，被调用函数实质上是在实参指针所指内存单元进行交换操作，因此能够把交换结果传递给调用者，参数传递过程如图 7.11 所示。程序如下：

```
1   /*  example7-3.cpp  */
2   #include <stdio.h>                  //使用库函数 printf 和 scanf
3   void Swap1(int x, int y);           //函数声明,形参 x 和 y 是值传递方式
4   void Swap2(int *p, int *q);         //函数声明,形参 p 和 q 是指针传递方式
5                                       //空行,以下是主函数
6   int main( )
7   {
8     int a = 5, b = 10;
9     Swap1(a, b);        //函数调用,将实参 a 和 b 的值分别传递给对应形参
10    printf("值传递方式的执行结果：a = %d, b = %d\n", a, b);
11    Swap2(&a, &b);      //函数调用，将实参 a 和 b 的地址分别传递给对应形参
12    printf("指针传递方式的执行结果：a = %d,b = %d\n", a, b);
13    return 0;                         //将 0 返回操作系统,表明程序正常结束
14  }
15                                      //空行,以下是其他函数定义
16  void Swap1(int x, int y)            //函数定义,x 和 y 是形参
17  {
```

```
    int temp;
    temp = x; x = y; y = temp;          //交换 x 和 y 的值
}
void Swap2(int *p, int *q)              //函数定义,p 和 q 是形参
{
    int temp;
    temp = *p; *p = *q; *q = temp;      //交换 p 所指变量和 q 所指变量的值
}
```

运行结果如下：

```
值传递方式的执行结果：a = 5, b = 10
指针传递方式的执行结果：a = 10, b = 5
```

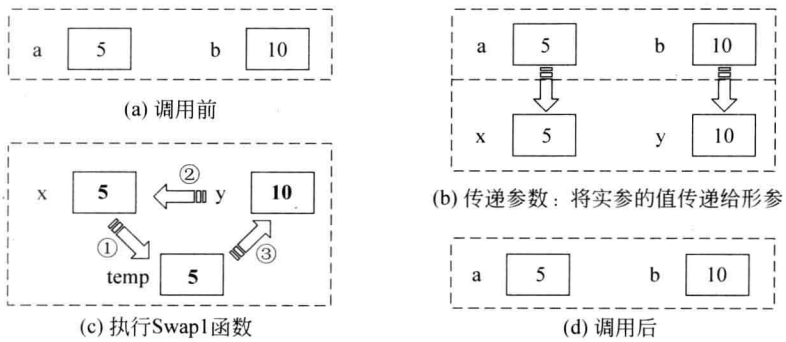


图 7.10　值传递方式示意图

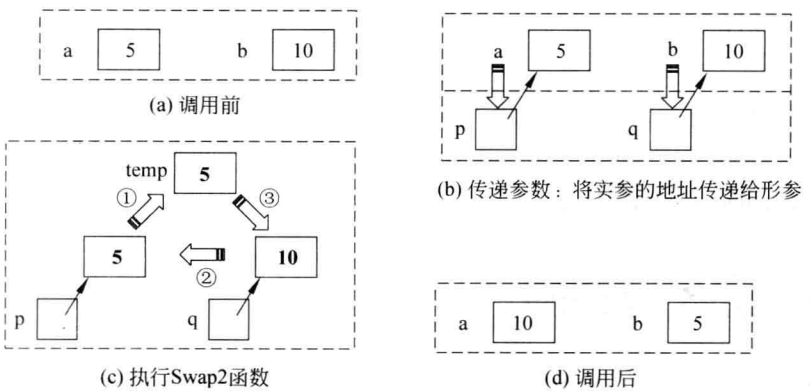


图 7.11　地址传递方式示意图

例 7.4　将三个整数由小到大输出。要求用函数实现。

解：设函数 TriSort 完成将三个整数由小到大排序，则函数 TriSort 需要接收三个整数，调用结束后需要将排序后的三个整数再传递给调用者，因此采用指针传递方式接收算法的输入并将排序结果返回。程序如下：

```
/*  example7-4.cpp   */
#include <stdio.h>              //使用库函数 printf 和 scanf
void TriSort(int *p, int *q, int *r);
                                //函数声明,形参 p、q、r 均是指针传递方式
```

```
                                        //空行,以下是主函数
int main( )
{
  int x, y, z;
  printf("请输入三个整数: ");
  scanf("%d%d%d", &x, &y, &z);
  TriSort(&x, &y, &z);          //函数调用,实参是变量 x、y 和 z 的地址
  printf("这三个整数由小到大依次是: %d, %d, %d", x, y, z);
  return 0;                     //将 0 返回操作系统,表明程序正常结束
}
                                        //空行,以下是其他函数定义
void TriSort (int *p, int *q, int *r)       //函数定义
{
  int temp;
  if (*p > *q)
  {
    temp = *p; *p = *q; *q = temp;          //交换*p 和*q
  }
  if (*r < *p)                              //即*r < *p < *q
  {
    temp = *r; *r = *q; *q = *p; *p = temp;
  }
  else if (*r < *q)                         //即*p < *r < *q
  {
    temp = *q; *q = *r; *r = temp;
  }
  return;
}
```

运行结果如下（下划线为用户输入）：

```
请输入三个整数: 2 6 3
这三个整数由小到大依次是: 2 3 6
```

7.2.4 解决任务 7.2 的程序

函数 CR 有 4 个参数，其中参数 M 和 N 以值传递方式接收算法的输入，参数 p 和 q 以指针传递方式返回两个计算结果，程序如下：

```
/*   鸡兔同笼（函数版）.cpp   */
#include <stdio.h>                  //使用库函数 printf 和 scanf
void CR(int M, int N, int *p, int *q);          //函数声明
                                    //空行,以下是主函数
int main( )
{
  int M, N;
  int chicken, rabbit;
  printf("请输入笼子里动物头的个数和脚的个数: ");
```

```
    scanf("%d%d", &M, &N);
    CR(M, N, &chicken, &rabbit);
                              //函数调用,前两个参数传值,后两个参数传地址
    if (chicken != 0 || rabbit != 0)
      printf("鸡有%d只，兔子有%d只\n", chicken, rabbit);
    else
      printf("输入数据矛盾，无解.\n");
    return 0;                 //将0返回操作系统,表明程序正常结束
}
                              //空行,以下是其他函数定义
void CR(int M, int N, int *p, int *q)
{
  int x, y;
  for (x = 0; x <= M; x++)                    //x为循环变量
  {
    y = M - x;
    if (2 * x + 4 * y == N) break;            //方程组已解,跳出循环
  }
  if (x <= M)
  {
    *p = x; *q = y;           //将x存放到p所指变量,y存放到q所指变量
  }
  else
  {
    *p = 0; *q = 0;
  }
  return;
}
```

运行结果如下（下划线为用户输入）：

```
请输入笼子里动物头的个数和脚的个数：7  20
鸡有4只,兔子有3只
```

7.3 程序设计实例

7.3.1 实例1——歌德巴赫猜想（函数版）

【问题】 歌德巴赫猜想：任意大于2的偶数可以分解为两个素数之和。请验证歌德巴赫猜想，要求用函数实现。

【想法】 在第5章实例2中用主函数验证了歌德巴赫猜想，这里采用相同的解法。

【算法】 设函数Gguess完成将一个偶数分解为两个素数之和，其算法描述如下：

输入：一个偶数n

功能：将偶数分解为两个素数之和

输出：两个素数x和y

伪代码

```
step1: 循环变量 x 从 2~n/2，重复执行下述操作：
    step1.1: y = n - x;
    step1.2: 如果 x 不是素数,则 x++; 转 step1 试探下一组数;
    step1.3: 如果 y 不是素数,则 x++; 转 step1 试探下一组数;
    step1.4: x 和 y 均为素数,则转 step2 输出结果;
step2: 输出 x 和 y;
```

【程序】 函数 Gguess 有 3 个参数，其中参数 n 以值传递方式接收函数的输入，参数 p 和 q 以指针传递方式返回两个计算结果，程序如下：

```
1   /*   歌德巴赫猜想(函数版).cpp   */
2   #include <stdio.h>              //使用库函数 printf 和 scanf
3   #include <math.h>               //使用库函数 sqrt
4   void Gguess(int n, int *p, int *q);
                                    //函数声明,p 和 q 所指变量存储结果
5                                   //空行,以下是主函数
6   int main( )
7   {
8     int n, n1, n2;
9     printf("请输入一个偶数: ");
10    scanf("%d", &n);
11    Gguess(n, &n1, &n2);          //函数调用,n、&n1 和&n2 为实参
12    printf("%d 可分解为%d+%d\n", n, n1, n2);        //输出结果
13    return 0;                     //将 0 返回操作系统,表明程序正常结束
14  }
15                                  //空行,以下是其他函数定义
16  void Gguess(int n, int *p, int *q)
                                    //函数定义,形参 n 传值,形参 p 和 q 传指针
17  {
18    int x, y, i;
19    for (x = 2; x <= n/2; x++)    //从 2 开始试探,即将 n 分解为 2 和 n-2
20    {
21      y = n - x;
22      for (i = 2; i <= sqrt(x); i++)  //判断 x 是否为素数
23      {
24        if (x % i == 0)           //能整除,则 x 不是素数
25          break;
26      }
27      if (i <= sqrt(x))           //如果 x 不是素数,则不用判断 y 是否为素数
28        continue;
29      for (i = 2; i <= sqrt(y); i++)      //判断 y 是否为素数
30      {
31        if (y % i == 0)           //能整除,则 y 不是素数
32          break;
33      }
34      if (i > sqrt(y))            //如果 y 是素数,则跳出外层循环
35      break;
36    }
```

```
    *p = x; *q = y;                          //返回计算结果
}
```

运行结果如下（下划线为用户输入）：

```
请输入一个偶数：44
44 可分解为 3+41
```

7.3.2 实例 2——求一元二次方程的根

【问题】 求一元二次方程 $ax^2+bx+c=0$ 的根。要求用函数实现。

【想法】 一元二次方程 $ax^2+bx+c=0$ 的求根公式如下：

$$x=\frac{-b\pm\sqrt{b^2-4ac}}{2a} \tag{7.1}$$

【算法】 设函数 Equation 实现方程求解，算法描述如下：

输入：一元二次方程的系数 a、b 和 c

功能：求一元二次方程的根

输出：一元二次方程的根 x1 和 x2

伪代码

```
step1: 计算 delta = b * b - 4 * a * c;
step2: 如果 delta 等于 0,则方程有两个相等的根,x1 = x2 = -b / (2 * a);
       否则，方程有两个不相等的根,x1 = (-b + sqrt(delta)) / (2 * a);
                                  x2 = (-b - sqrt(delta)) / (2 * a);
step3: 返回 x1 和 x2;
```

【程序】 主函数用变量 a、b 和 c 接收从键盘输入的三个系数，如果满足 a≠0 且 $b^2-4ac\geqslant 0$，则调用函数 Equation 求解方程。函数 Equation 有 5 个参数，其中参数 a、b 和 c 以值传递方式接收函数的输入，参数 p 和 q 以指针传递方式返回函数的两个计算结果。程序如下：

```
/*   一元二次方程.cpp   */
#include <stdio.h>                   //使用库函数 printf 和 scanf
#include <math.h>                    //使用库函数 sqrt
void Equation(double a, double b, double c, double *p, double *q);
                                     //空行,以下是主函数
int main( )
{
  double a, b, c, delta, root1, root2;
                                     //a、b、c 为系数,root1、root2 是根
  do
  {
    printf("请输入一元二次方程的系数：");
    scanf("%lf%lf%lf", &a, &b, &c);
```

```
    delta = b * b - 4 * a * c;
  } while ((a == 0) || (delta < 0));
                                  //保证 a 不等于 0 且 delta 大于等于 0
  Equation(a, b, c, &root1, &root2);          //函数调用
  printf("方程的根为: %6.2f\t%6.2f \n", root1, root2);
  return 0;                       //将 0 返回操作系统,表明程序正常结束
}
                                  //空行,以下是其他函数定义
void Equation(double a, double b, double c, double *p, double *q)
{                          //形参 a、b 和 c 接收输入,形参 p 和 q 输出方程的两个根
  double x1, x2;
  double delta = b * b - 4 * a * c;
  if (delta == 0)                 //delta 等于 0,则方程有两个相等的实根
  {
    x1 = -b / (2 * a);
    x2 = x1;
  }
  else                            //delta 大于 0,则方程有两个不相等的实根
  {
    x1 = (-b + sqrt(delta)) / (2 * a);
    x2 = (-b - sqrt(delta)) / (2 * a);
  }
  *p = x1; *q = x2;               //保存计算结果
}
```

运行结果如下（下划线为用户输入）：

```
请输入一元二次方程的系数: 1 2 1
方程的根为:  -1.00  -1.00
```

习 题 7

一、选择题

1. 若有变量定义 int a, b, *p = &b; 能够正确从键盘读入 2 个整数并分别赋给变量 a 和 b 的语句是（　　）。

A. scanf("%d %d", &a, &p);　　B. scanf("%d %d", &a, p);

C. scanf("%d %d", a, p);　　D. scanf("%d %d", a, *p);

2. 若有变量定义 int a = 512, *p = &a; 则*p 的值为（　　）。

A. 无确定值　　B. 0　　C. 变量 a 的地址　　D. 512

3. 下面（　　）能够实现交换指针 p 和 q 所指内存单元的值。

A. temp = *p ; *p = *q ; *q = temp;　　B. temp = p ; p = q ; q = temp;

C. temp = p ; *p = *q ; q = temp;　　D. temp = &p ; *p = *q ; q = *temp;

4. 两个基类型相同的指针变量之间不能进行（　　）运算。

A. <　　B. >　　C. +　　D. -

5. 若有变量定义 int a = 5, *p = &a, *q = &a; 则下面不能正确执行的赋值语句是（　　）。

A. a = p - q;　　B. p = a;

C. p = q;　　D. a = (*p) * (*q);

6. 若有变量定义 int x, y = 5, *p = &x; 则能完成 x = y 赋值功能的语句是（　　）。

A. x = *p;　　B. *p = y;　　C. x = &y;　　D. *p = &y;

7. 若有变量定义 int m = 5, n, *p; 则以下正确的程序段是（　　）。

A. p = &n; scanf("%d", &p);　　B. p = &n; scanf("%d", *p);

C. scanf("%d", &n); *p = n;　　D. p = &n; *p = m;

二、程序设计题

1. 从键盘输入 10 个整数，求这 10 个整数的最大值和序号。

2. 把 1、2、3、4、5、6、7、8、9 组合成三个三位数，要求每个数字仅用一次，并且每个三位数均是完全平方数。

3. 有人买了一筐鸡蛋，只记得数量肯定不只 100 个，还记得当时两个两个地数余 1 个，三个三个地数余 2 个，五个五个地数余 4 个，七个七个地数正好数完，求筐里至少有多少个鸡蛋？

4. 歌德巴赫猜想：任意一个奇数（大于 1）都可以分解为三个素数之和。随机产生 10 个大于 1 的奇数进行验证，并给出每个奇数的分解结果。

5. 任意给定两个正整数 a 和 n，计算 $a + aa + aaa + \cdots + aa\cdots a$（$n$ 个 a）的和。

第8章

批量同类型数据的组织——数组

实际问题需要处理的数据常常具有这样的特点：数据量很大，并且数据具有相同的数据类型。大多数程序设计语言提供了数组来组织这种具有相同数据类型的批量数据。**数组**是具有相同数据类型的数据集合，其中每个数据称为**数组元素**，数组元素的整体有一个共同的名称，称为**数组名**，数组元素在数组中的序号称为**下标**，数组名和下标可以唯一标识某个数组元素。

数组是程序设计语言中常见的构造数据类型，按照其维度分为一维数组、二维数组和多维数组，常用的是一维数组和二维数组。

8.1 一维数组

【任务 8.1】舞林大会

【问题】 某学校组织现代舞比赛，聘请了 N 名评委为参赛选手打分，评分原则是去掉一个最高分和一个最低分，取剩下评分的平均值作为该选手的最后得分。评委给出的评分范围为[0, 10]，并且可以是小数。假设评委的评分由键盘输入，要求输出参赛选手的最终得分。

【想法】 首先输入并保存 N 名评委的评分，然后累加总分 sum、找出最高分 max 和最低分 min， 则参赛选手的最终得分为(sum – max – min)/(N–2)。累加和即是将所有评分加到一起；找出最高分的方法可以用“打擂台”来形容，先假定第一个评分为最高分，再依次将余下的每个评分与 max 进行比较，每次比较都将较大者存储在 max 中；找出最低分采用类似的方法。实际上，可以在一遍循环中同时完成累加总分、找出最高分和最低分的操作。

【算法】 设函数 Average 实现求参赛选手的最终得分，其算法描述如下：

输入：N 个实数

功能：计算参赛选手的最终得分

输出：除最高分和最低分外，剩下评分的平均值

伪代码

```
step1: 初始化 sum = 第 1 个评分;max = 第 1 个评分;min = 第 1 个评分;
step2: 循环变量 i 为 2～N,重复执行下述操作:
    step2.1: sum = sum + 第 i 个评委的评分;
    step2.2: 如果第 i 个评分大于 max,则 max = 第 i 个评分;
    step2.3: 如果第 i 个评分小于 min,则 min = 第 i 个评分;
    step2.4: i++;
step3: 返回(sum - max - min)/(N - 2);
```

算法需要存储 N 个评分，这涉及如下两个问题：

（1）如何定义变量？假设用 N 个简单变量存储 N 个评分，如何定义这么多简单变量？如何为这么多变量命名？而且这么多的变量定义会使程序很冗长。

（2）如何处理变量？如果用 N 个简单变量存储 N 个评分，则这些变量在内存中占用各自的存储单元，不能体现变量之间的关联性，而且分散存储的变量难以实现对这些变量的连续访问。

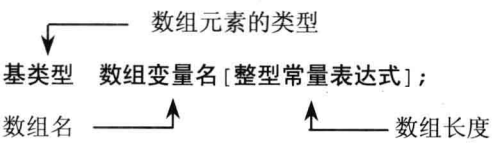
8.1.1 一维数组的定义和初始化

一维数组仅使用一个下标即可唯一标识数组元素。本质上，一维数组就是数据集合的线性排列，因此，一维数组也称为**向量**。

1. 一维数组的定义

同简单变量一样，一维数组变量也要先定义后使用。

【语法】 定义一维数组的一般形式如下：



其中，基类型是任意合法的数据类型，表示数组元素的数据类型；数组变量名是一个标识符，表示数组在内存中的起始地址；方括号是数组标志；整型常量表达式的运算结果表示数组元素的个数，也称为数组长度。

【语义】 定义一维数组变量，编译器为其分配一段连续的存储空间。由于数组元素具有相同的数据类型，因此，每个数组元素占有相同大小的存储单元，如图 8.1 所示。

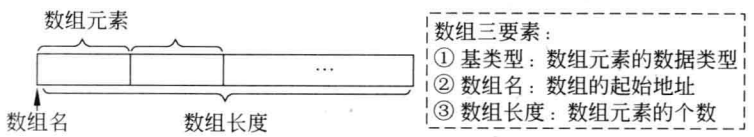


图 8.1 一维数组的存储示意图

以下都是合法的一维数组定义：

```
int a[10];          //定义数组 a,数组元素的类型为整型,共有 10 个数组元素
char ch[10];        //定义数组 ch,数组元素的类型为字符型,共有 10 个数组元素
double b[5];        //定义数组 b,数组元素的类型为双精度型,共有 5 个数组元素
```

C/C++语言不允许对数组长度进行动态定义，因此，数组长度必须是一个固定的值。例如，如下一维数组定义是非法的：

```
int n = 10;
int a[n];                          //数组长度不能是变量
```

数组长度可以是符号常量，例如，如下是常用的一维数组定义方式：

```
const int N = 10;                  //或 #define N 10
int a[N];                          //N 是符号常量,被预处理为 int a[10]
```

2. 一维数组元素的引用

数组是将固定数目的数组元素组织成一个序列，每个数组元素在数组中都有一个序号（称为下标）。引用数组元素使用数组名和该元素在数组中的下标。

【语法】 引用一维数组元素的一般形式如下：

数组变量名[**整型常量表达式**]

数组下标

其中，整型常量表达式的运算结果作为数组下标，为了便于编译程序工作，C/C++语言规定数组下标从 0 开始，因此数组下标的取值范围是[0, 数组长度−1]。

【语义】 引用该下标对应的数组元素。

需要强调的是，数组变量定义和数组元素引用中常量表达式的含义不同。在定义数组时，常量表达式表示数组长度，即数组元素个数，是一个常量，其值不能改变；在引用数组元素时，常量表达式表示元素的序号，即数组元素的下标，是一个变量，可以在[0, 数组长度−1]的范围内变化。

定义数组以后，数组中的每个元素其实就相当于一个变量，同简单变量一样，也具有变量名、变量值、变量类型和变量地址等变量的基本属性。例如，int a[10]定义了一个 int 型数组，数组元素 a[i]相当于一个 int 型简单变量，其地址是&a[i]，其元素值是 a[i]，如图 8.2 所示。

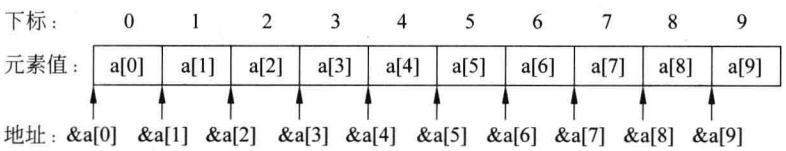


图 8.2 数组元素具有变量的基本属性

3. 一维数组的初始化

定义一维数组变量后，编译程序会给该变量分配一块连续的存储空间，但是从程序

开始执行到给数组赋值之前，该数组是没有确定值的，即数组变量为**“值无定义的”**。可以在定义数组变量时为数组元素赋初值，使数组变量成为“值有定义的”。在定义一维数组的同时为数组元素赋初值称为一维数组的初始化。

【语法】 初始化一维数组的一般形式如下：

```
基类型 数组变量名[整型常量表达式] = {初值表};
                    ↑
                    └──── 缺省则为初值个数
```

其中，初值表是由逗号分隔的数据值；整型常量表达式表示数组的长度，如果缺省，则将初值表中的初值个数作为数组长度。

【语义】 将初值表中各数据值顺序赋给数组中的相应元素。如果提供的初值个数小于数组长度，则未指定值的数组元素被赋值为 0；如果提供的初值个数多于数组长度，其结果取决于编译器，有些编译器会忽略多余的初值，有些编译器会给出错误信息。

在定义一维数组时，可以给全部数组元素初始化，例如：

```
int a[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```

也可以部分初始化，即给前面部分的数组元素赋初值，例如：

```
int a[10] = {0, 1, 2, 3, 4};
```

相当于

```
int a[10] = {0, 1, 2, 3, 4, 0, 0, 0, 0, 0};
```

也可以省略数组长度，例如：

```
int a[ ] = {0, 1, 2, 3, 4};
```

相当于

```
int a[5] = {0, 1, 2, 3, 4};                    //将初值个数作为数组长度
```

良好的编程习惯 8.1

在定义数组时，为了提高程序的可读性，建议无论是否给数组的全部元素赋初值，都不要省略数组长度。但是，如果给数组的全部元素赋初值，而且初值个数较多，则可以省略数组长度，从而避免人工计算的烦琐和误差。

8.1.2 一维数组的操作

1. 输入输出操作

在 C/C++语言中，数组变量不能作为 scanf 函数和 printf 函数的实参，换言之，不能整体读入一个数组，也不能整体输出一个数组。可以使用循环语句将一批数据读入数组，

在循环体中读取键盘的输入并送到指定数组元素中，例如：

```
int a[10],i;
for (i = 0; i < 10; i++)
  scanf("%d", &a[i]);                    //元素a[i]相当于int型简单变量,其地址为&a[i]
```

也可以使用循环语句输出数组中的全部元素，例如：

```
int i, a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
for (i = 0; i < 10; i++)
  printf("%d", a[i]);
```

2. 赋值操作

在 C/C++语言中，可以在数组初始化时对数组进行整体赋值，除此之外没有提供数组整体赋值的语句，因此，如下语句是错误的：

```
int a[5] = {1, 2, 3, 4, 5}, b[5];
b = a;                                   //不能对数组进行整体赋值
```

可以使用赋值语句给数组元素赋值，例如：

```
int a[80];
a[0] = 1; a[10] = 18; a[18] = 20;
```

如果数组元素的值具有某种规律，可以采用循环语句，在循环体中使用赋值语句进行赋值，例如：

```
int a[10], i;
for (i = 0; i < 10; i++)
  a[i] = 2 * i;                          //数组元素为偶数
```

需要强调的是，VC++编译器没有提供数组下标越界检查，但是在运行时会出现下标越界错误。因此，为数组元素赋值需要注意数组下标不能越界，例如：

```
int a[10], i;
for (i = 0; i <= 10; i++)
  a[i] = 2 * i;              //当 i 等于 10 时数组下标越界,但 VC++编译器没有错误提示
```

3. 其他操作

C/C++语言没有定义施加于一维数组上的运算，对一维数组的操作是通过对其数组元素的操作实现的。本质上，数组元素是一个简单变量，因此，数组元素的使用方法与同类型简单变量的使用方法相同。例如，如下操作都是合法的：

```
int a[10];                               //定义 int 型数组 a
a[0] = 1; a[1] = 2;
a[2] = a[0] + a[1] * 5;                  //取数组元素值并执行算术运算
```

例 8.1 在一维数组 r[n]中查找最大值元素。

解：设变量 max 存储一维数组 r[n]的最大值元素，首先假定 r[0]为最大值，然后从 r[1]起依次将每一个数组元素与 max 进行比较，每次比较都将较大者存储在 max 中。程序如下：

```
/*  example8-1.cpp  */
#include <stdio.h>                    //使用库函数 printf 和 scanf
#define N 5
                                      //空行,以下是主函数
int main( )
{
  int a[N], i, max;                   //max 存储最大值
  printf("请输入%d 个整数: ", N);
  for (i = 0; i < N; i++)
  {
    scanf("%d", &a[i]);
  }
  max = a[0];                         //假定 a[0]为最大值
  for (i = 1; i < N; i++)
  {
    if (max < a[i])
      max = a[i];
  }
  printf("最大值为: %d\n", max);
  return 0;                           //将 0 返回操作系统,表明程序正常结束
}
```

运行结果如下（下划线为用户输入）：

```
请输入 5 个整数: 1  2  3  4  5
最大值为: 5
```

8.1.3 一维数组作为函数的参数

一维数组作为函数的参数是比较典型的地址传递方式。在函数定义时，将形参声明为一维数组，无须指定数组长度，即数组名后只跟一个空的方括号，形参实质上是一个指针变量；在函数调用时，将数组名作为实参，且实参数组与形参数组的基类型一致，参数传递的过程是将实参数组的首地址传递给形参，实际上传递的是整个数组；在调用结束时，系统自动释放形参（实质上是指针变量）的存储空间。

一维数组作为函数的参数，实参数组和形参数组的长度可以一致也可以不一致，编译器对形参数组长度不进行检查，形参数组长度由函数调用时的实参数组决定。为了方便函数对数组元素进行处理，一般另设一个参数传递数组长度。

例 8.2 在一维数组 r[n]中查找最大值元素。要求用函数实现。

解：求解思想与例 8.1 相同。设函数 Max 实现求数组 r[n]的最大值元素，具体说明

如下：

- 函数声明：int Max(int r[ ], int n) 相当于 int Max(int *r, int n)
- 函数调用：Max (a, 5)
- 参数结合的过程相当于：int *r = a, int n = 5，参数传递的过程如图 8.3 所示。

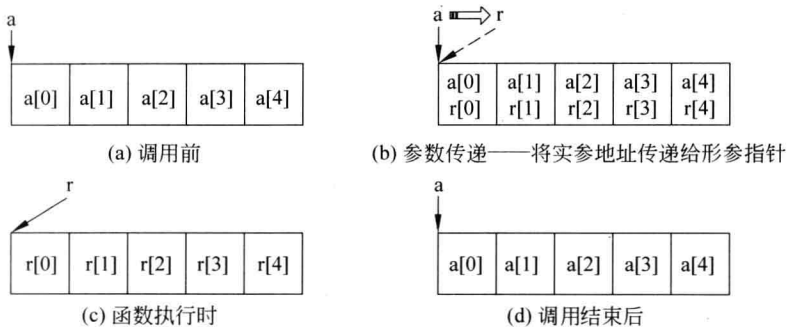


图 8.3　一维数组作为函数的参数

【程序】　主函数用数组 a[N]接收从键盘输入的 N 个整数，然后调用函数 Max 求数组 a[N]的最大值，程序如下：

```
/*  example8-2.cpp   */
#include <stdio.h>                  //使用库函数 printf 和 scanf
#define N 5                         //定义符号常量
int Max(int r[ ], int n);           //函数声明
                                    //空行,以下是主函数
int main( )
{
  int i, a[N];
  printf("请输入%d 个整数: ", N);
  for (i = 0; i < N; i++)
  {
    scanf("%d", &a[i]);             //读入第 i 个数组元素
  }
  printf("最大值为: %d\n", Max(a, N));  //输出调用函数 Max 的返回结果
  return 0;                         //将 0 返回操作系统,表明程序正常结束
}
                                    //空行,以下是其他函数定义
int Max(int r[ ], int n)            //函数定义,一维数组作为形参
{
  int i, max = r[0];                //max 存储最大值并假定 r[0]最大
  for (i = 1; i < n; i++)           //从 r[1]到 r[n -1]依次将 r[i]与 max 比较
  {
    if (max < r[i])
      max = r[i];                   //max 始终保存当前最大值
  }
  return max;                   //结束函数 Max,并将变量 max 的值返回到调用处
}
```

一维数组作为参数的特点是：形参数组和实参数组占用同一段内存单元，因此，形参数组元素的值发生改变相当于对应实参数组元素的值发生改变。

例 8.3 在一维数组 r[n]中查找最大值和次最大值元素。

解：设数组 r[n]至少有 2 个元素，变量 max1 和 max2 分别存储最大值和次最大值，首先将数组前两个元素中的较大者作为当前的最大值，较小者作为当前的次最大值，然后从第 3 个元素起依次取数组的每一个元素 r[i]与 max1 和 max2 进行比较，有以下三种情况：

（1）如果 r[i]≥max1，则当前的最大值为 r[i]，次最大值为 max1；

（2）如果 max1＞r[i]＞max2，则当前的最大值仍为 max1，次最大值为 r[i]；

（3）如果 r[i]≤max2，则当前的最大值仍为 max1，次最大值仍为 max2。

【算法】 设函数 FirstSecondMax 实现求一维数组 r[n]中的最大值和次最大值，其算法描述如下：

输入：一维数组 r[n]

功能：在一维数组中查找最大值和次最大值元素

输出：最大值 max1 和次最大值 max2

伪代码

```
step1: 如果 r[0] > r[1],则 max1 = r[0]; max2 = r[1];
       否则 max1 = r[1]; max2 = r[0];
step2: 下标 i 从 2~n-1 重复执行下述操作:
    step2.1: 如果 r[i] >= max1,则 max2 = max1; max1 = r[i];
             否则，如果 r[i] > max2,则 max2 = r[i];
    step2.2: i++;
step3: 返回 max1 和 max2;
```

【程序】 函数 FirstSecondMax 有 4 个参数，其中参数 r 和 n 是典型的一维数组作为函数的参数，用来接收算法的输入，参数 p 和 q 以指针传递方式返回函数的两个计算结果。程序如下：

```
1   /*  example8-3.cpp  */
2   #include <stdio.h>                  //使用库函数 printf 和 scanf
3   #define N 10;                       //定义符号常量
4   void FirstSecondMax(int r[ ], int n, int *p, int *q); //函数声明
5                                       //空行,以下是主函数
6   int main( )
7   {
8     int i, a[N], firstMax, secondMax;
9     printf("请输入%d 个整数: ", N);
10    for (i = 0; i < N; i++)
11    {
12      scanf("%d", &a[i]);             //接收键盘输入的整数并存入元素 a[i]
13    }
14    FirstSecondMax(a, N, &firstMax, &secondMax);        //函数调用
15    printf("最大值是: %d, 次最大值是: %d\n", firstMax, secondMax);
16    return 0;                         //将 0 返回操作系统,表明程序正常结束
```

```
}
                                        //空行,以下是其他函数定义
void FirstSecondMax(int r[ ], int n, int *p, int *q)  //函数定义
{
  int max1, max2, i;     //max1 保存当前最大值，max2 保存当前次最大值
  if (r[0] > r[1])
  {
    max1 = r[0]; max2 = r[1];          //最大值是 r[0],次最大值是 r[1]
  }
  else
  {
    max1 = r[1]; max2 = r[0];          //最大值是 r[1],次最大值是 r[0]
  }
  for (i = 2; i < n; i++)
                      //依次将 r[2]~r[n - 1]与 max1 和 max2 进行比较
  {
    if (r[i] >= max1)                  //r[i]为当前最大值
    {
      max2 = max1; max1 = r[i];
    }
    else if (r[i] > max2)              //r[i]为当前次最大值
          max2 = r[i];
  }
  *p = max1; *q = max2;                //保存运算结果
}
```

运行结果如下（下划线为用户输入）：

```
请输入 10 个整数：1 2 3 4 5 5 4 3 2 1
最大值是：5,次最大值是：5
```

8.1.4 解决任务 8.1 的程序

首先用数组 a[N]保存 N 个评委给出的评分，然后调用函数 Average 计算参赛选手的最终得分，程序如下：

```
/*    duty8-1.cpp   */
#include <stdio.h>                  //使用库函数 printf 和 scanf
#define N 5;                                    //定义符号常量 N
double Average(double r[ ], int n);             //函数声明
                                                //空行,以下是主函数
int main( )
{
  double a[N];                                  //评分可以是小数
  for (int i = 0; i < N; i++)
  {
    printf("请第%d 个评委输入评分：", i + 1);   //数组下标从 0 开始
    scanf("%lf", &a[i]);            //接收 double 型实数要用格式符"%lf"
```

```
    }
    printf("参赛选手的最终得分是：%4.1f\n", Average(a, N)); //函数调用
    return 0;                          //将 0 返回操作系统,表明程序正常结束
}
                                       //空行,以下是其他函数定义
double Average(double r[ ], int n)     //函数定义,一维数组作为参数
{
    double max = r[0], min = r[0], sum = r[0];    //sum 累加总分
    for (int i = 1; i < n; i++)
    {
        sum = sum + r[i];              //累加总分
        if (max < r[i])
            max = r[i];                //保存当前最高分
        if (min > r[i])
            min = r[i];                //保存当前最低分
    }
    return (sum - max - min)/(n - 2);  //结束函数,返回结果
}
```

运行结果如下（下划线为用户输入）：

```
请第 1 个评委输入评分：8
请第 2 个评委输入评分：9
请第 3 个评委输入评分：8.5
请第 4 个评委输入评分：9.2
请第 5 个评委输入评分：9
参赛选手的最终得分是：8.8
```

8.2 二维数组

【任务 8.2】幻方问题

【问题】 幻方又称魔方阵、幻方阵，游戏规则是在一个 $n \times n$ 的矩阵中填入 1 到 n^2 的数字，使得每一行、每一列、每条对角线的累加和都相等。如图 8.4 所示是一个 3 阶幻方，每一行、每一列、每条对角线的累加和都等于 15。

6	1	8
7	5	3
2	9	4

图 8.4 3 阶幻方示例

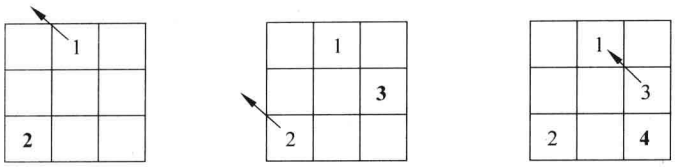
【想法】 下面是一种“左上斜行法”的填数方法，该方法适用于任意奇数阶幻方，具体填数过程如下：

① 由 1 开始填数，将 1 放在第 0 行的中间位置；

② 将幻方想象成上下、左右相接，每次往左上角走一步，会有下列情况：

- 左上角超出上边界，则在最下边相对应的位置填入下一个数，如图 8.5(a)所示；
- 左上角超出左边界，则在最右边相对应的位置填入下一个数，如图 8.5(b)所示；
- 按上述方法找到的位置已填数，则在原位置的同一列下一行填入下一个数，如

图 8.5(c)所示。



(a) 左上角超出上边界　(b) 左上角超出左边界　(c) 左上角已填数

图 8.5 “左上斜行法”的填数过程

【算法】 设函数 MagicSquare 实现“左上斜行法”构造幻方，其算法描述如下：

输入：幻方的阶数 n

功能：填写奇数阶幻方

输出：无

伪代码

```
step1: 初始化填数的位置 i = 0, j = n/2;              //即第 0 行的中间位置
step2: 在位置(i, j)填入 1;
step3: 数字 k 从 2 ~ n*n 重复执行下述操作:
    step3.1: 从位置(i, j)往左上角走一步到位置(i - 1, j - 1);
    step3.2: i = i - 1; 如果 i 超出上边界,则 i = n - 1;
    step3.3: j = j - 1; 如果 j 超出左边界,则 j = n - 1;
    step3.4: 如果位置(i, j)已经填数,则在原位置的同一列下一行填入 k;
             否则,在位置(i, j)填入 k;
    step3.5: k++;
```

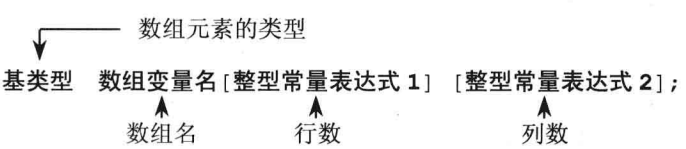
显然，可以用一个二维数组表示幻方。**二维数组**需要两个下标即可唯一标识数组元素，主要用于表示二维表和矩阵。

8.2.1 二维数组的定义和初始化

1. 二维数组的定义

同一维数组变量一样，二维数组变量也要先定义后使用。

【语法】 定义二维数组的一般形式如下：



其中，基类型是任意合法的数据类型，表示数组元素的数据类型；数组变量名是一个标识符，表示数组在内存中的起始地址；整型常量表达式 1 的运算结果表示二维数组的行数；整型常量表达式 2 的运算结果表示二维数组的列数。注意，整型常量表达式 1 和整型常量表达式 2 分别写在各自的方括号内。

【语义】 定义一个二维数组变量，编译器为其分配一段连续的存储空间。

以下都是合法的二维数组定义：

```
int a[10][5];              //定义二维数组 a,元素类型为整型,共有 10×5 个元素
char ch[10][5];            //定义二维数组 ch,元素类型为字符型,共有 10×5 个元素
double b[5][10];           //定义二维数组 b,元素类型为双精度型,共有 5×10 个元素
```

2. 二维数组元素的引用

引用二维数组元素使用数组名、该元素在数组中的行下标和列下标。

【语法】 引用二维数组元素的一般形式如下：

数组变量名[整型常量表达式 1] [整型常量表达式 2]

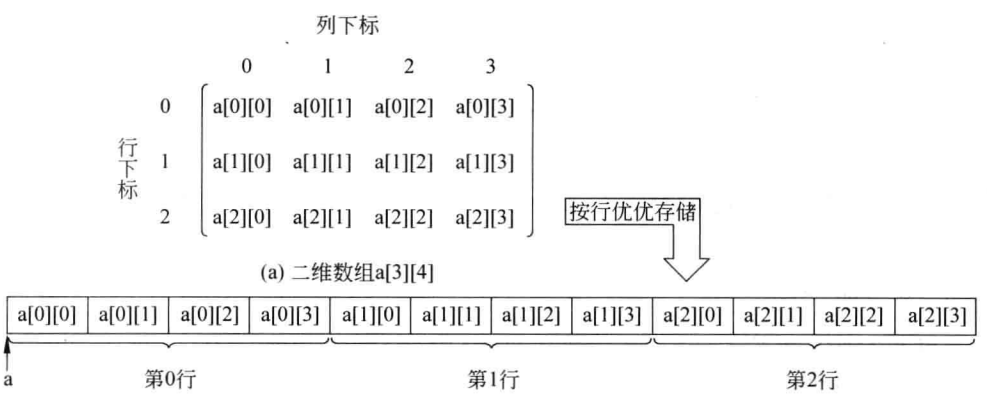
行下标　　列下标

其中，整型常量表达式 1 的运算结果作为数组的行下标，其取值范围是[0, 行数–1]；整型常量表达式 2 的运算结果作为数组元素的列下标，其取值范围是[0, 列数–1]。

【语义】 引用行下标和列下标对应的二维数组元素。

在 C/C++语言中，二维数组按行优先形式存储，即先存储第 0 行，再存储第 1 行，依此类推，其中每一行的元素再按列顺序存储。例如，如下二维数组定义后，其存储示意图如图 8.6 所示。

```
int a[3][4];               //定义二维数组 a,元素类型为整型,共有 3×4 个数组元素
```



(a) 二维数组a[3][4]

(b) 二维数组a[3][4]在内存中的存储方式

图 8.6　二维数组及其存储方式

3. 二维数组的初始化

定义二维数组变量后，从程序开始执行到给数组赋值之前，数组变量为**“值无定义的”**。可以在定义数组变量时为数组元素赋初值，使数组变量成为“值有定义的”。在定义二维数组的同时为数组元素赋初值称为二维数组的初始化。

【语法】 初始化二维数组的一般形式如下：

基类型 数组变量名[整型常量表达式 1] [整型常量表达式 2] = {初值表};

所有数据值写在一对花括号内

或

基类型 数组变量名[整型常量表达式 1] [整型常量表达式 2] = {{初值表}, ……,{初值表}};

每一行的数据值写在一对花括号内

其中，初值表是由逗号分隔的数据值；整型常量表达式 1 表示二维数组的行数；整型常量表达式 2 表示二维数组的列数。

【语义】 将初值表中各数据值依次赋给数组中的相应元素。如果提供的初值个数小于数组长度，则未指定值的数组元素被赋值为 0；如果提供的初值个数多于数组长度，其结果取决于编译器，有些编译器会忽略多余的初值，有些编译器会给出错误信息。

初始化二维数组时可以将所有数据值写在一对花括号内，也可以将每一行的数据值写在一对花括号内。例如，如下二维数组的初始化语句：

```
int a[3][4] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
```

也可以写作：

```
int a[3][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};
```

写成如下形式会增加程序的清晰性：

```
int a[3][4] = {{1, 2, 3, 4},                    //为第 0 行元素赋值
               {5, 6, 7, 8},                    //为第 1 行元素赋值
               {9, 10, 11, 12}};                //为第 2 行元素赋值
```

8.2.2 二维数组的操作

1. 输入输出操作

二维数组变量不能作为 scanf 函数和 printf 函数的实参，换言之，二维数组不能实现整体读入和整体输出。可以使用循环语句将一批数据读入数组，在循环体中读取键盘的输入并送到指定数组元素中，例如：

```
int a[5][10], i, j;
for (i = 0; i < 5; i++)
  for (j = 0; j < 10; j++)
    scanf("%d", &a[i][j]);     //元素 a[i][j]相当于简单变量,其地址为&a[i][j]
```

也可以使用循环语句输出数组中的全部元素，例如：

```
int a[5][10], i, j;
for (i = 0; i < 5; i++)
```

```
for (j = 0; j < 10; j++)
  printf("%d", a[i][j]);
```

2. 赋值操作

在 C/C++语言中，可以在数组初始化时对数组进行整体赋值，除此之外没有提供数组整体赋值的语句。与一维数组的赋值操作相同，二维数组也可以使用赋值语句给数组元素赋值，例如：

```
int a[5][80];
a[0][4] = 12; a[1][12] = 10; a[2][20] = 12;
```

如果数组元素的值具有某种规律，可以采用两层嵌套的循环语句，在循环体中使用赋值语句进行赋值，需要注意数组下标不能越界。例如：

```
int a[5][10], i, j;
for (i = 0; i < 5; i++)                    //处理第 i 行元素
  for (j = 0; j < 10; j++)                 //处理第 j 列元素
    a[i][j] = i + j;                       //为第 i 行第 j 列元素赋值
```

3. 其他操作

C/C++语言没有定义施加于二维数组上的运算，对二维数组的操作是通过对其数组元素的操作实现的。二维数组元素的使用方法与同类型简单变量的使用方法相同。例如，下列操作都是合法的：

```
int a[3][4];                               //定义二维数组 a,共有 3×4 个数组元素
a[0][0] = 1; a[0][1] = 2;
a[0][2] = a[0][0] + a[0][1] / 10;          //取数组元素值并执行算术运算
```

例 8.4 求二维数组 r[m][n]的最大值元素。

解：设变量 max 存储二维数组 r[m][n]的最大值，先假定 a[0][0]为最大值，然后逐行依次将数组的每一个元素与 max 比较，每次比较都将较大者存放在变量 max 中。程序如下：

```
1   /*  example8-4.cpp  */
2   #include <stdio.h>                     //使用库函数 printf 和 scanf
3                                          //空行,以下是主函数
4   int main( )
5   {
6     int a[10][10], max, i, j, m, n;      //定义二维数组最多 10 行 10 列
7     printf("请输入二维数组的行数和列数: ");
8     scanf("%d%d", &m, &n);               //指定二维数组的行数和列数
9     printf("请输入%d 个整数: ", m * n);
10    for (i = 0; i < m; i++)              //处理二维数组的第 i 行
11      for (j = 0; j < n; j++)            //处理二维数组的第 j 列
12        scanf("%d", &a[i][j]);           //读入二维数组的每一个数组元素
13    max = a[0][0];                       //假定 a[0][0]为最大值
```

```
    for (i = 0; i < m; i++)    //依次将数组的每一个元素与 max 进行比较
      for (j = 0; j < n; j++)
        if (max < a[i][j])
          max = a[i][j];        //max 保存当前最大值
    printf("最大值是：%d\n", max);
    return 0;                   //将 0 返回操作系统,表明程序正常结束
}
```

运行结果如下（下划线为用户输入）：

```
请输入二维数组的行数和列数：2 3
请输入 6 个整数：11 12 32 44 51 67
最大值是：67
```

8.2.3 二维数组作为函数的参数

二维数组作为函数的参数属于指针传递方式。在函数定义时，将形参声明为二维数组；在函数调用时，将二维数组名作为实参，且实参数组与形参数组的基类型一致，参数传递的过程是将实参数组的首地址、行数和列数传给形参；在调用结束后，系统自动释放形参的存储空间。

在参数传递时，由于从实参传递过来的是数组的首地址，在内存中按行优先存放，如果在形参中不说明列数，则编译器无法确定该数组的行数和列数。因此，形参数组可以指定行数和列数，也可以省略行数，但必须指明列数，而且必须为常量表达式。为了方便函数对数组元素进行处理，一般另设参数表示形参数组的行数和列数。

例 8.5 求二维数组 r[m][n]的最大值元素。要求用函数实现。

解：求解思想与例 8.4 相同。设函数 Max 实现求二维数组 r[m][n]的最大值，具体说明如下：

- 函数声明：int Max(int r[10][10], int m, int n)或 int Max(int r[ ][10], int m, int n)，注意不能是 int Max(int r[m][n])，因为“[ ]”内必须为常量表达式。
- 函数调用：Max (a, 3, 4)。
- 参数结合的过程相当于：int *r = a, int m = 3, int n = 4。

【程序】 主函数用数组 a[10][10]接收从键盘输入的 m*n 个整数，然后调用函数 Max 求数组 a[m][n]的最大值，程序如下：

```
/*  example8-5.cpp  */
#include <stdio.h>                  //使用库函数 printf 和 scanf
int Max(int r[10][10], int m, int n);          //函数声明
                                    //空行,以下是主函数
int main( )
{
  int a[10][10], i, j , m, n;       //定义二维数组最多 10 行 10 列
  printf("请输入二维数组的行数和列数：");
  scanf("%d%d", &m, &n);            //指定二维数组的行数和列数
  printf("请输入%d 个整数：", m * n);
```

```
    for (i = 0; i < m; i++)
      for (j = 0; j < n; j++)
        scanf("%d", &a[i][j]);
    printf("最大值是: %d\n", Max(a, m, n));
                                        //输出调用函数 Max 的结果
    return 0;                           //将 0 返回操作系统,表明程序正常结束
}
                                        //空行,以下是其他函数定义
int Max(int r[10][10], int m, int n)    //函数定义,形参是二维数组
{
  int i, j, max = r[0][0];
  for (i = 0; i < m; i++)   //依次将数组的每一个元素与 max 进行比较
    for (j = 0; j < n; j++) //m 和 n 为实参数组的行数和列数
      if (max < r[i][j])
        max = r[i][j];
  return max;
}
```

运行结果如下（下划线为用户输入）：

```
请输入二维数组的行数和列数：2 3
请输入 6 个整数：11 12 32 44 51 67
最大值是：67
```

8.2.4 解决任务 8.2 的程序

```
/*   幻方.cpp   */
#include <stdio.h>                  //使用库函数 printf 和 scanf
void MagicSquare(int r[100][100], int n);       //函数声明
                                    //空行,以下是主函数
int main( )
{
  int a[100][100], n, i, j;         //定义二维数组最多 100 行 100 列
  printf("请输入一个 100 以内的奇数: ");          //确定幻方的阶数
  scanf("%d", &n);
  for (i = 0; i < n; i++)           //初始化二维数组 a[n][n]
    for (j = 0; j < n; j++)
      a[i][j] = 0;
  MagicSquare(a, n);                //调用函数实现具体的填数过程
  for (i = 0; i < n; i++)           //输出 n 阶幻方
  {
    for (j = 0; j < n; j++)
      printf("%d\t", a[i][j]);
    printf("\n");                   //输出一行,将光标移到下一行
  }
  return 0;                         //将 0 返回操作系统,表明程序正常结束
}
                                    //空行,以下是其他函数定义
```

```
23    void MagicSquare(int r[100][100], int n)
                                        //函数定义,二维数组作为形参
24    {
25      int i = 0, j = n/2;           //i 和 j 表示二维数组的行列下标
26      int iTemp, jTemp;             // iTemp 和 jTemp 将暂存 i 和 j 的值
27      r[i][j] = 1;                  //将 1 填入第 0 行中间位置
28      for (int k = 2; k <= n*n; k++)
                                      //k 表示即将填的数,将 2~n * n 填入数组
29      {
30        iTemp = i; jTemp = j;       //暂存 i 和 j 的值
31        i = (i - 1 + n) % n;        //即 i = i - 1; if (i < 0) i = n - 1;
32        j = (j - 1 + n) % n;        //即 j = j - 1; if (j < 0) j = n - 1;
33        if (r[i][j] > 0)            //第 i 行第 j 列已经填数
34        {
35          i = (iTemp + 1) % n;      //即 i = iTemp + 1; if (i == n) i = 0;
36          j = jTemp;                //原位置的下一行同一列
37        }
38        r[i][j] = k;          //在 r[i][j]处填入 k
39      }
40      return;
41    }
```

运行结果如下（下划线为用户输入）：

```
请输入一个 100 以内的奇数：3
6       1       8
7       5       3
2       9       4
```

8.3 程序设计实例

8.3.1 实例 1——对角线元素之和

【问题】 求 $n \times n$ 矩阵中两条对角线元素之和。

【想法】 用二维数组 $a[n][n]$存储 $n \times n$ 矩阵，分别求两条对角线元素之和。观察两条对角线元素下标的特点，如图 8.7 所示，主对角线元素行下标 i 与列下标 j 相等，副对角线元素行下标 i 与列下标 j 满足 $i+j=n-1$。

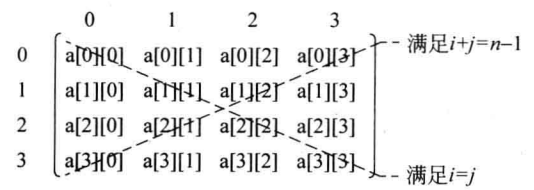


图 8.7 二维数组 a[4][4]的两条对角线行列下标满足的条件

【算法】 设函数 Sum 实现求矩阵中两条对角线元素之和，其算法描述如下：

输入：二维数组 a[n][n]

功能：求矩阵中两条对角线元素之和

输出：两条对角线元素之和 sum

伪代码

```
step1: 初始化累加器 sum = 0;
step2: 循环变量 i 从 0 到 n-1,重复执行下述操作:
    step2.1: sum = sum + a[i][i];
    step2.2: i++;
step3: 循环变量 i 从 0 到 n-1,重复执行下述操作:
    step3.1: sum = sum + a[i][n-1-i];
    step3.2: i++;
step4: 返回 sum;
```

【程序】 主函数用二维数组 a 接收从键盘输入的 n * n 个整数，再调用函数 Sum 求二维数组 a 的两条对角线元素之和。程序如下：

```
/*  对角线元素之和.cpp  */
#include <stdio.h>                          //使用库函数 printf 和 scanf
int Sum(int r[10][10], int n);              //函数声明
                                            //空行,以下是主函数
int main( )
{
  int a[10][10], i, j , n, sum; //定义二维数组最多 10 行 10 列
  printf("请输入矩阵的阶数: ");              //确定实际的阶数
  scanf("%d", &n);
  printf("请输入%d 个整数: ", n * n);
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      scanf("%d", &a[i][j]);                //依次输入每一个元素
  sum = Sum(a, n);                          //函数调用,变量 sum 接收函数的返回值
  printf("该矩阵两条对角线元素之和为%d\n", sum);
  return 0;                                 //将 0 返回操作系统,表明程序正常结束
}
                                            //空行,以下是其他函数定义
int Sum(int r[10][10], int n)               //函数定义,二维数组作为形参
{
  int i, sum = 0;
  for (i = 0; i < n; i++)                   //n 为实参数组的阶数
    sum = sum + r[i][i];                    //累加主对角线元素
  for (i = 0; i < n; i++)
    sum = sum + r[i][n-1-i];                //累加副对角线元素
  return sum;                               //结束函数,并将 sum 返回到调用处
}
```

运行结果如下（下划线为用户输入）：

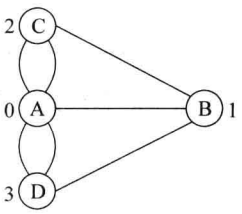
```
请输入矩阵的阶数: 3
```

```
请输入 9 个整数：1 2 3 4 5 6 7 8 9
该矩阵两条对角线元素之和为 30
```

8.3.2 实例 2——哥尼斯堡七桥问题

【问题】 参见 1.1.2 节。

【想法】 参见 1.1.2 节。进一步地，将结点 A、B、C、D 编号为 0、1、2、3，用二维数组 mat[4][4]表示七桥问题的图模型，如果结点 i（0≤i≤3）和 j（0≤i≤3）之间有 k 条边，则元素 mat[i][j]的值为 k，如图 8.8 所示。求解七桥问题的关键是求与每个结点相关联的边数，即是在二维数组 mat[4][4]中求每一行元素之和。



(a) 七桥问题的图模型

	0(A)	1(B)	2(C)	3(D)
0(A)	0	1	2	2
1(B)	1	0	1	1
2(C)	2	1	0	0
3(D)	2	1	0	0

(b) 对应的二维数组mat[4][4]

图 8.8 用二维数组表示七桥问题的图模型

【算法】 设函数 EulerCircuit 求解七桥问题，算法描述如下：

输入：二维数组 mat[4][4]

功能：计算七桥问题中通奇数桥的结点个数

输出：通奇数桥的结点个数 count

伪代码

```
step1: count 初始化为 0;
step2: 下标 i 从 0~n - 1 重复执行下述操作:
    step2.1: 计算第 i 行元素之和 degree;
    step2.2: 如果 degree 为奇数,则 count++;
step3: 返回 count;
```

【程序】 主函数首先初始化二维数组 mat[4][4]，即建立七桥问题对应的图模型，然后调用函数 EulerCircuit 计算图模型中通奇数桥的结点个数，再根据欧拉规则判定是否存在欧拉回路。程序如下：

```
1   /*   欧拉回路.cpp   */
2   #include <stdio.h>                          //使用库函数 printf 和 scanf
3   int EulerCircuit(int mat[10][10], int n);       //函数声明
4                                               //空行,以下是主函数
5   int main( )
6   {
7     int mat[10][10] = {{0,1,2,2},{1,0,1,1},{2,1,0,0},
      {2,1,0,0}};
8     int num;
```

```
    num = EulerCircuit(mat, 4);          //调用函数得到通奇数桥的结点个数
    if (num > 2)                         //有多于两个地方通奇数桥
      printf("有%d个地方通奇数桥，不存在欧拉回路\n ", num);
    else if (num == 2 || num == 0)
                                         //两个地方通奇数桥或没有一个地方通奇数桥
          printf("存在欧拉回路\n");
    return 0;                            //将0返回操作系统,表明程序正常结束
}
                                         //空行,以下是其他函数定义
int EulerCircuit(int mat[10][10], int n)
                                         //函数定义,二维数组作为形参
{
    int i, j, count = 0, degree;         //count累计通奇数桥的结点个数
    for (i = 0; i < n; i++)              //依次累加每一行的元素
    {
      degree = 0;                        // degree存储通过结点i的桥数,初始化为0
      for (j = 0; j < n; j++)            //依次处理每一列的元素
      {
        degree = degree + mat[i][j];     //将通过结点i的桥数求和
      }
      if (degree % 2 != 0)               //桥数为奇数
        count++;
    }
    return count;                        //结束函数,并将count返回到调用处
}
```

运行结果如下：

```
有4个地方通奇数桥,不存在欧拉回路
```

习 题 8

一、选择题

1. 在C/C++语言中引用数组元素时，其数组下标允许是（　　）。

   A. 整型常量　　B. 整型表达式

   C. 整型常量或整型表达式　　D. 任何类型的表达式

2. 若二维数组a有m列，则在a[i][j]前的元素个数为（　　）。

   A. j * m + i　　B. i * m + j　　C. i * m + j – 1　　D. j * m + i–1

3. 若有变量定义int a[ ][3] = {1, 2, 3, 4, 5, 6, 7}; 则数组a第一维的大小是（　　）。

   A. 2　　B. 3　　C. 4　　D. 无确定值

4. 以下不正确的变量定义是（　　）。

   A. double x[5] = {2.0, 4.0, 6.0, 8.0, 10.0};

   B. int y[5] = {0, 1, 3, 5, 7, 9};

C. char c1[ ] = {'1', '2', '3', '4', '5'};

D. char c2[ ] = {'\x10', '\xa', '\8'};

5. 以下不能对二维数组 a 进行正确初始化的语句是（　　）。

A. int a[2][3] = {0};　　B. int a[ ][3] = {{1, 2},{0}};

C. int a[2][3] = {{1, 2},{3, 4},{5, 6}};　　D. int a[ ][3] = {1, 2, 3, 4, 5, 6};

6. 若使用一维数组名作函数实参，则以下正确的说法是（　　）。

A. 必须在主调函数中说明此数组的大小

B. 实参数组类型与形参数组类型可以不匹配

C. 在被调用函数中，不需要考虑形参数组的大小

D. 实参数组名与形参数组名必须一致

7. 若用数组名作为函数的实参，传递给形参的是（　　）。

A. 数组的首地址　　B. 数组第一个元素的值

C. 数组中全部元素的值　　D. 数组元素的个数

8. 若有以下数组定义和函数调用语句，则函数 Fun 的形参定义为（　　）。

```
int a[3][4] = {1};
Fun(a);
```

A. Fun(int b[ ][6])　　B. Fun(int b[3][ ])

C. Fun(int b[ ][4])　　D. Fun(int b[2][5])

二、程序设计题

1. 删除一维数组中值相同的多余元素，即值相同的元素只保留一个。

2. 将一维整型数组 A 调整为左右两部分，左边所有元素均为奇数，右边所有元素均为偶数。

3. 在一维数组中删除所有值为 x 的元素。

4. 大整数加法。为了实现大整数的加法，可以把一个大整数 A 存储在数组中。例如：假设大整数最多 20 位，则可以将大整数 A = 123456789 存储在数组 data[20]中，编写程序求两个大整数的和。

5. 若在矩阵 $\boldsymbol{A}$ 中存在一个元素 $a_{i,j}$（$0 \leqslant i \leqslant n-1$，$0 \leqslant j \leqslant m-1$），该元素是第 i 行元素中的最大值且又是第 j 列元素中的最小值，则称此元素为该矩阵的一个鞍点。假设以二维数组存储矩阵 $\boldsymbol{A}$，求该矩阵中的所有鞍点。

6. 求矩阵 $\boldsymbol{A}_{mn}$ 的转置矩阵 $\boldsymbol{B}_{nm}$，其中 $B_{ij} = A_{ji}$。

7. 求矩阵 $\boldsymbol{A}_{mp}$ 和矩阵 $\boldsymbol{B}_{pn}$ 的乘积 C_{mn}。

第9章

字符数据的组织——字符串

字符串简称串，是由零个或多个字符组成的有限序列。字符串是重要的非数值处理对象，在事务处理程序中，顾客的姓名、货物的产地等，一般是作为字符串进行处理的。在文字编辑、符号处理等许多领域，字符串也得到了广泛应用，因而在程序设计语言中大都有字符串变量的概念，而且提供了库函数实现基本的串操作。

【任务 9.1】恺撒加密

【问题】 朱迪斯·恺撒在其政府的秘密通信中使用恺撒密码进行信息加密，恺撒加密因而得其名。恺撒加密的基本思想是将待加密信息（称为明文）中每个字母在字母表中向后移动常量 *key*，得到加密信息（称为密文）。例如，假设字母表为英文字母表，*key* 等于 3，则对于明文 *computer systems* 将加密为 *frpsxwhu vbvwhpv*。

【想法】 扫描明文字符串，依次将明文中的每一个字母进行替换。例如，如果 *key* 等于 3，则将 *a* 替换为 *d*，将 *b* 替换为 *e*，依此类推。如果到字母表尾部则绕回到开头，因此，将 *x* 替换为 *a*，将 *y* 替换为 *b*，将 *z* 替换为 *c*。以小写字母为例，对待加密字母 ch 的加密过程如下：

① 求待加密字母 ch 在字母表中的位移量：ch – 'a'；

② 再向后移动 key 个位移量：ch – 'a' + key；

③ 如果超过字母表的尾部，则绕回到开头：(ch – 'a' + key)%26；

④ 求结果位移量对应的小写字母：'a' + (ch – 'a' + key)%26。

【算法】 设函数 Encrypt 实现恺撒加密，其算法描述如下：

输入：明文 str1，密钥 key

功能：恺撒加密

输出：密文 str2

伪代码

```
step1: 对明文中的每一个字符 ch,执行下述操作:
    step1.1: 如果 ch 是大写字母,则 ch = 'A' + (ch - 'A' + key) % 26;
    step1.2: 如果 ch 是小写字母,则 ch = 'a' + (ch - 'a' + key) % 26;
step2: 输出密文;
```

作为非数值处理对象，字符串的表示和处理与整型、实型等数值数据有很大不同。本章介绍C/C++语言中字符串变量的表示和处理。

9.1 字符串变量的定义和初始化

C/C++语言提供了字符数组和字符串指针两种方式来表示字符串。

9.1.1 字符数组

字符串在内存中就是一个字符数组，因此，字符串可以用字符数组变量来表示。

1. 字符数组变量的定义

【语法】 定义字符数组变量的一般形式如下：

```
char 串变量名[整型常量表达式];
```

串名(即数组名)　　串长 + 1

其中，串变量名为字符串变量的标识符，也就是字符数组名；整型常量表达式表示字符的个数，为了表示字符串的长度，字符串的尾部存储一个特殊字符'\0'（ASCII 码为 0）作为串的终结符，因此整型常量表达式的值为串长+1。

【语义】 定义一个字符数组变量，系统为其分配一段连续的存储空间，串变量名为首地址，整型常量表达式的值为这段存储空间的长度。

例如，下面是合法的字符数组变量定义，其存储示意图如图 9.1 所示。

```
char str[6];                    //定义字符数组 str,能表示含有 5 个字符的字符串
```

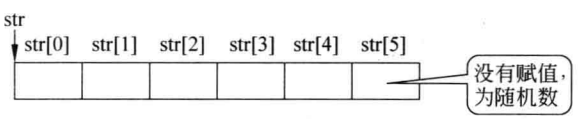


图 9.1　字符数组变量的定义

2. 字符数组变量的初始化

定义字符数组变量后，编译程序会给该变量分配一段连续的存储空间，但是从程序开始执行到给变量赋值之前，该变量是“值无定义的”，如图 9.1 所示。字符数组变量的初始化是指在定义字符数组变量的同时为其赋初值。

【语法】 初始化字符数组变量的一般形式如下：

```
char 串变量名[整型常量表达式] = 字符串常量;
```

或

```
char 串变量名[整型常量表达式] = 字符序列;
```

其中，字符串常量是由双引号括起的字符串，字符串常量中无须出现终结符'\0'，编译器会为自动字符数组加上终结符；字符序列是由花括号括起的、由逗号分隔的字符序列，字符序列中需要指定终结符'\0'；整型常量表达式表示字符数组的长度，如果缺省，则将字符串常量的长度+1 或字符序列的个数作为字符数组的长度。

【语义】 为字符数组变量初始化。

可以将字符数组初始化为一个字符串常量，例如，下面是合法的字符数组变量初始化，其存储示意图如图 9.2 所示。

```
char str[6] = "China";                    //定义字符数组 str,其初值为"China\0"
```

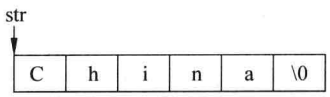


图 9.2 字符数组变量的初始化

如果字符数组的长度多于字符串常量的长度，则将剩余单元初始化为'\0'。例如，下面是合法的字符数组初始化语句，其存储示意图如图 9.3 所示。

```
char str[10] = "China";                   //定义字符数组 str,其初值为"China\0"
```

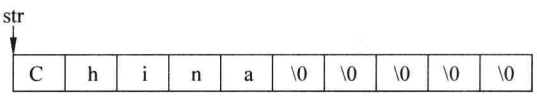


图 9.3 字符数组变量的初始化

可以将字符数组初始化为一个字符序列，如下初始化语句的字符序列中指定了终结符'\0'，则将字符数组初始化为一个字符串，其存储示意图如图 9.2 所示。

```
char str[ ] = {'C', 'h', 'i', 'n', 'a', '\0'}; //字符数组 str 初始化为"China\0"
```

如下初始化语句的字符序列中没有指定终结符'\0'，则字符数组的初始化结果仅仅是一个字符数组而不是字符串，其存储示意图如图 9.4 所示。

```
char str[ ] = {'C', 'h', 'i', 'n', 'a'};      //str 是字符数组而不是字符串
```

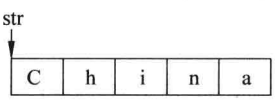


图 9.4 字符数组变量的初始化

9.1.2 字符串指针

字符串在内存中通常是连续存储的，指向字符串的指针存储字符串在内存中的起始地址（即第一个字符的存储地址），简称字符串指针。

1. 字符串指针变量的定义

【语法】 定义字符串指针的一般形式如下：

```
char *字符串指针变量名;
     ↑
     指针定义符
```

其中，“*”是指针定义符；字符串指针变量名是一个合法的标识符。

【语义】 定义字符串指针变量。

例如，如下语句定义了指向某个字符串的指针变量 str，但指针 str 尚未指向某个具体的字符串。

```
char *str;                    //定义字符串指针变量 str,该指针是悬空的
```

2. 字符串指针变量的初始化

定义一个字符串指针变量后，系统为该指针变量分配存储空间，但没有将该指针与某个有效对象相关联，因此，该指针是悬空的。在定义字符串指针变量的同时将其指向某个具体的字符串，称为字符串指针变量的初始化。

【语法】 初始化字符串指针变量的一般形式如下：

```
char *字符串指针变量名 = 字符串常量;
     ↑
     指针定义符
```

【语义】 定义字符串指针变量，并将其指向字符串常量。

如下是字符串指针变量的初始化语句，其存储示意图如图 9.5 所示。

```
char *str= "China";      //定义字符串指针变量 str,指向字符串"China\0"
```

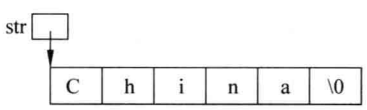


图 9.5 字符串指针变量的初始化

9.2 字符串的操作

9.2.1 输入输出操作

C/C++语言提供了字符串输入输出函数：gets 函数和 puts 函数，这两个函数包含在头文件 stdio.h 中，能够实现字符串整体的输入输出。C/C++语言提供的标准输入输出函数 scanf 函数和 printf 函数也能够实现字符串的输入输出操作。

1. 字符串输入输出函数

（1）gets 函数

【函数原型】 gets 函数的原型如下：

```
char *gets(char *str)
```

其中，str 可以是字符数组变量，也可以是指向某个确定存储单元的字符串指针。

【功能】 将键盘的输入以字符串的形式存储在 str 所指内存单元中，直至遇到回车换行符'\n'，并将'\n'转换为字符串终结符'\0'。

gets 函数是以回车换行符作为键盘输入字符串的结束标志，因此，gets 函数能接收包含空格符的字符串。例如：

```
char str[80];                    //定义字符数组 str
gets(str);
```

假设从键盘上输入（□表示空格，< Enter >表示回车）：

I□□love□□China< Enter >

则字符数组变量 str 中的字符串将是"I□□love□□China\0"。

良好的编程习惯 9.1

在实际使用时，由于无法限制用户从键盘上键入字符串的长度，因此，用于接收字符串的字符数组的长度应该足够长，以便能够存储从键盘输入的字符串以及终结符，否则 gets 函数将把超过字符数组长度之外的字符顺序存储在字符数组后面的存储单元中，从而可能覆盖其他内存单元，造成程序错误。

定义字符串指针后，如果没有与某个内存地址相关联，则指针的值是不确定的，即不能明确指针具体指向的内存单元，无法确定输入字符串的存储位置，因此，如下语句是错误的：

```
char *str;                       //定义字符串指针变量 str
gets(str);                       //指针 str 的值不确定,不能正确实现读操作
```

良好的编程习惯 9.2

为了避免引用未赋值的字符串指针，建议在定义字符串指针时将其初始化为空，例如： char *str = NULL。

如果将字符串指针初始化为指向某个字符串常量，由于常量不能被修改，无法将读入的字符串覆盖字符串常量，因此，如下语句是错误的：

```
char *str = "China!";            //定义并初始化字符串指针变量 str
gets(str);                       //相当于修改字符串常量,不能正确实现读操作
```

字符串指针必须指向某个确定的存储单元，才能调用 gets 函数实现读入字符串操作，例如，如下语句能够正确读入字符串，其操作示意图如图 9.6 所示。

```
char *str, ch[80];                    //定义字符串指针变量 str 和字符数组 ch
str = ch;                             //将指针 str 指向字符数组 ch
gets(str);                            //将读入的字符串存入 str 所指内存单元
```

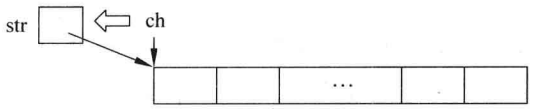


图 9.6　将字符串指针 str 与内存地址 ch 相关联

（2）puts 函数

【函数原型】　puts 函数的原型如下：

```
int puts(char *str)
```

其中，str 可以是字符数组，可以是已经与某个内存地址相关联的字符串指针，也可以是字符串常量。

【功能】　将字符串输出到标准终端上，并将字符串的终结符'\0'转换为换行符'\n'。

如下字符串输出语句都是正确的：

```
char ch[ ] = "I  love  China !";              //定义并初始化字符数组 ch
char *str = "I  love  China !";               //定义并初始化字符串指针变量 str
puts(ch);
puts(str);
puts("I  love  China !");
```

2. 标准输入输出函数

（1）scanf 函数

【函数原型】　scanf 函数的原型如下：

```
int scanf("%s", char *str)
```

其中，"%s"是格式控制符，表示输入一个字符串；str 可以是字符数组，也可以是指向某个确定存储单元的字符串指针。

【功能】　忽略前导空格，将键盘输入的字符串保存到 str 所指内存单元中，直到遇到空格或回车换行符，并自动在字符串后面加上终结符'\0'。

由于 scanf 函数以空格或回车换行符作为键盘输入字符串的结束标志，因此，不能接收包含空格符的字符串。例如，如下语句：

```
char str[80];                 //定义字符数组 str
scanf("%s", str);             //注意不要写成&str,因为 str 是地址
```

当从键盘上输入（□表示空格，<Enter>表示回车）：

```
I□ □love□ □China<Enter>
```

则变量 str 中的字符串将是"I"。如果要接收输入的全部内容，应该用三个变量来接收空格分隔的字符串，语句如下：

```
char str1[10], str2[10], str3[10];          //定义字符数组 str1, str2, str3
scanf("%s %s %s ", str1, str2, str3);
```

则数组 str1 中的字符串将是"I"，数组 str2 中的字符串将是"love"，数组 str3 中的字符串将是"China"。所以，scanf 函数可以连续输入多个字符串。

与 gets 函数类似，字符串指针必须与某个内存地址相关联，才能调用 scanf 函数实现读入字符串操作，因此，如下语句是错误的：

```
char *str1, *str2 = "China";    //定义字符串指针变量 str1 和 str2
scanf("%s", str1);       //str1 尚未与某个内存地址相关联,不能正确实现读操作
scanf("%s", str2);       //str2 指向字符串常量,不能正确实现读操作
```

字符串指针必须指向某个确定的存储单元，才能调用 scanf 函数实现读入字符串操作，例如，如下语句序列能够正确读入字符串：

```
char *str, ch[80];       //定义字符串指针变量 str 和字符数组 ch
str = ch;                //将指针 str 指向字符数组 ch
scanf("%s", str);        //将读入的字符串存入 str 所指内存单元
```

（2）printf 函数

【函数原型】 printf 函数的原型如下：

```
int printf("%s", char *str)
```

其中，"%s"是格式控制符，表示输出一个字符串；str 可以是字符数组，也可以是已经与某个内存地址相关联的字符串指针，还可以是字符串常量。

【功能】 将字符串输出到标准终端上。

如下字符串输出语句都是正确的：

```
char ch[ ] = "I  love  China !";    //定义并初始化字符数组 ch
char *str = "I  love  China !";     //定义并初始化字符串指针变量 str
printf("%s\n", ch);                 //输出字符串 ch 后不能自动换行,需要'\n'
printf("%s\n", str);                //输出字符串 str 后不能自动换行,需要'\n'
printf("I  love  China !\n");       //输出字符串常量后不能自动换行,需要'\n'
```

例 9.1 显示问候语。要求从键盘上输入一个姓名 XXX，然后显示“Hello, XXX!”。

解：首先定义一个字符数组 name，由于一个汉字占 2 个字节，假设姓名最多为 4 个汉字，还要存储终结符'\0'，因此 name 数组的长度至少为 9。然后接收从键盘输入的字符串，再输出该字符串。程序如下：

```
1   /*   example9-1.cpp   */
2   #include <stdio.h>          //使用库函数 printf 和 scanf
3                               //空行,以下是主函数
```

```
4    int main( )
5    {
6      char name[10];          //定义字符数组 name
7      printf("请输入姓名: "); //输出提示信息
8      scanf("%s", name);      //接收从键盘输入的字符串,自动附加终结符'\0'
9      printf("Hello, %s!\n", name);      //输出结果
10     return 0;               //将 0 返回操作系统,表明程序正常结束
11   }
```

运行结果如下（下划线为用户输入）：

```
请输入姓名: Amy
Hello, Amy!
```

9.2.2 赋值操作

字符数组变量通常不能进行整体赋值。字符数组变量名是字符数组的起始地址，是一个地址常量，不能给地址常量赋值。因此，如下语句是错误的：

```
char ch[10];
ch = "China";                          //ch 是地址常量,常量不能被赋值
```

字符数组的赋值需要使用循环语句将字符逐个进行赋值。由于普通数组的元素个数一般是确定的，可以用元素个数来控制循环，而字符数组并没有显式存储字符个数，只是规定在终结符'\0'之前的字符均是字符串的有效字符，因此，一般通过终结符来控制循环。换言之，一般通过检测字符是否为'\0'来判断字符串是否结束。例如，如下语句实现字符数组的复制：

```
char ch1[20], ch2[ ] = "I  love  China !";
int i;
for (i = 0; ch2[i] != '\0'; i++)
  ch1[i] = ch2[i];                     //逐个将数组 ch2 的字符赋给 ch1
ch1[i] = '\0';                         //为字符数组 ch1 存入终结符
```

字符串指针变量是字符串的起始地址，是一个指针变量，因而可以赋值。例如，如下语句实现字符串的复制：

```
char *str1, *str2 = "I  love  China !";
str1 = str2;                           //str1 是字符串指针变量,变量可以被赋值
```

由于字符串的整体赋值是比较常用的操作，C/C++语言提供了 strcpy、strncpy、memcpy 等库函数实现字符串的整体赋值。下面以 strcpy 为例进行介绍。

【函数原型】 strcpy 函数的原型如下：

```
char *strcpy(char *strDestination, char *strSource)
```

其中，strDestination 可以是字符数组，也可以是指向某个确定存储单元的字符串指针；

strSource 可以是字符数组，也可以是已经被赋值的字符串指针，还可以是字符串常量。

【功能】 将字符串 strSource（源串）复制到字符串 strDestination（目的串）中。

【返回值】 如果复制成功，则返回指向字符串 strDestination 的指针。

如下语句都能够正确实现字符串的复制操作：

```
char ch1[20], ch2[ ] = "I  love  China !";
char *str = "I  love  China !";
strcpy(ch1, ch2);
strcpy(ch1, str);
strcpy(ch1, "I  love  China !");
```

9.2.3 字符串的比较

在计算机中，每个字符都对应唯一的数值表示——称为字符编码，字符间的大小关系就定义为对应字符编码之间的大小关系。字符编码有很多种，微型计算机常用 ASCII 码。例如，字符'a'和'b'的 ASCII 码分别为 97 和 98，则'a' < 'b'。

字符串的比较是通过组成串的字符之间的比较来进行的，其比较规则是：将两个字符串逐个字符比较，直至遇到不同的字符或'\0'为止；如果全部字符都相同，则两个字符串相等；如果出现不同的字符，则以第一个不同字符的比较结果作为两个字符串的比较结果。例如，"ab" < "ac"，"ba" < "bbcc"，"bacc" < "bc"。

两个字符串的比较不能使用“>”、“==”或“<”等关系运算符，C/C++语言提供了 strcmp、stricmp、strncmp、strnicmp 等库函数实现两个字符串之间的比较。下面以 strcmp 为例进行介绍。

【函数原型】 strcmp 函数的原型如下：

```
int strcmp(char *string1, char *string2)
```

其中，string1 和 string2 可以是字符数组和字符串指针，也可以是字符串常量。

【功能】 将字符串 string1 与字符串 string2 进行比较。

【返回值】 若 string1 大于 string2，则返回值大于零；若 string1 小于 string2，则返回值小于零；若 string1 等于 string2，则返回值等于零。

如下语句都能够正确实现字符串的比较操作：

```
char ch1[20] = "I  love  you !", ch2[ ] = "I  love  China !";
char *str1 = "I  love  you !", *str2 = "I  love  China !";
strcmp(ch1, ch2);                                    //返回 1
strcmp(str1, ch1);                                   //返回 0
strcmp("I  love  China !", "I  love  you !");        //返回-1
strcmp(str2, ch1);                                   //返回-1
strcmp(str1, ch2);                                   //返回 1
strcmp("I  love  you !", ch1);                       //返回 0
```

9.2.4 常用字符串库函数

使用字符串库函数需要包含头文件 string.h。字符串库函数没有提供字符数组的越界检查，所以，要保证各字符串已分配足够的存储空间，否则可能引起不可预知的错误。常用的字符串库函数请参见附录 C。

例 9.2 从键盘输入一串字符，再逆序输出。

解：定义字符数组接收从键盘输入的字符串，调用库函数 strlen 求该字符串的长度，再用 for 循环逆序输出，程序如下：

```
1   /*    example9-2.cpp   */
2   #include <stdio.h>                  //使用库函数 printf 和 gets
3   #include <string.h>                 //使用库函数 strlen
4                                       //空行,以下是主函数
5   int main( )
6   {
7     char str[100];                    //字符数组存储从键盘输入的字符串
8     int n = 0;                        //变量 n 保存字符串的长度
9     printf("请输入一个字符串：");      //输出提示信息
10    gets(str);                        //接收包括空格在内的字符串
11    n = strlen(str);                  //求字符串 str 的长度
12    printf("该字符串的逆序是：");
13    for (int i = n - 1; i >= 0; i--)        //逆序输出每一个字符
14      printf("%c", str[i]);
15    printf("\n");                     //输出换行符
16    return 0;                         //将 0 返回操作系统,表明程序正常结束
17  }
```

运行结果如下（下划线为用户输入）：

```
请输入一个字符串：I love China!
该字符串的逆序是：!anihC evol I
```

9.3 解决任务 9.1 的程序

```
1   /*    duty9-1.cpp   */
2   #include <stdio.h>                  //使用库函数 printf、scanf 和 gets
3   void Encrypt(char str[ ], int key);             //函数声明
4                                       //空行,以下是主函数
5   int main( )
6   {
7     char str[100];                    //定义字符数组 str
8     int k;
9     printf("请输入一个字符串：");      //输出提示信息
10    gets("%s", str);                  //接收从键盘输入的字符串,包括空格
```

```
    printf("请输入密钥: "); //输出提示信息
    scanf("%d", &k);
    printf("明文是: %s, ", str);
    Encrypt(str, k);        //函数调用,实参 str 是字符数组 str 的首地址
    printf("密文是: %s\n", str);              //输出结果
    return 0;               //将 0 返回操作系统,表明程序正常结束
}
                            //空行,以下是其他函数定义
void Encrypt(char str[ ], int key)
                            //函数定义,str[ ]无须指明长度
{
  for (int i = 0; str[i] != '\0'; i++)
  {
    if (str[i] >= 'A' && str[i] <= 'Z')     //str[i]为大写字母
      str[i] = 'A' + (str[i] - 'A' + key) % 26;
    if (str[i] >= 'a' && str[i] <= 'z')     //str[i]为小写字母
      str[i] = 'a' + (str[i] - 'a' + key) % 26;
  }
  return;
}
```

运行结果如下（下划线为用户输入）：

```
请输入一个字符串：abcxyz
请输入密钥：3
明文是：abcxyz，密文是：defabc
```

9.4 程序设计实例

9.4.1 实例 1——字数统计

【问题】 Microsoft Word 字处理软件的字数统计功能可以对一篇文档统计字符数(计空格）和字符数（不计空格），如图 9.7 所示。请模拟该功能，对一个字符串进行字符统计，分别统计包括空格和不包括空格的字符数（假设字符串少于 80 个字符）。

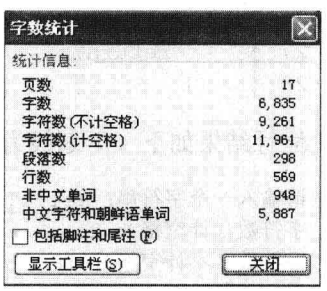


图 9.7 Word 软件的字数统计功能

【想法】 用字符数组实现，包括空格的字符数即是字符串的长度，不包括空格的字符数即是在计算字符串长度时不统计空格。

【算法】 设函数 CharCount 实现字数统计功能，其算法描述如下：

输入：字符串 str

功能：字符统计

输出：包括空格的字符数 count1，不包括空格的字符数 count2

伪代码

```
step1: 对字符串中的每一个字符 ch,执行下述操作:
    step1.1: count1++;
    step1.2: 如果 ch 不是空格,则 count2++;
step2: 输出 count1 和 count2;
```

【程序】 由于函数 Count 得到两个计算结果 count1 和 count2，考虑用指针作为函数的参数传递计算结果。程序如下：

```
1   /*  字数统计.cpp  */
2   #include <stdio.h>              //使用库函数 printf 和 gets
3   void CharCount(char str[ ], int *p, int *q);    //函数声明
4                                   //空行,以下是主函数
5   int main( )
6   {
7     char ch[80];        //定义尽可能长的字符数组以接收从键盘输入的字符串
8     int countSum = 0, countNoSpace = 0;
9     printf("请输入一个字符串：");                    //输出提示信息
10    gets(ch);           //注意不能用 scanf 函数,因为要接收空格
11    CharCount(ch, &countSum, &countNoSpace);       //函数调用
12    printf("字符数（计空格）：%d\n", countSum);
13    printf("字符数（不计空格）：%d\n", countNoSpace);
14    return 0;                     //将 0 返回操作系统,表明程序正常结束
15  }
16                                  //空行,以下是其他函数定义
17  void CharCount(char str[ ], int *p, int *q)
18  {
19    int count1 = 0, count2 = 0;
20    for (int i = 0; str[i] != '\0'; i++)
21    {
22      count1++;
23      if (str[i] != '\40')    //空格的 ASCII 码为 32,相当于八进制数 40
24        count2++;
25    }
26    *p = count1; *q = count2; //保存运算结果
27    return;                   //结束函数的执行
28  }
```

运行结果如下（下划线为用户输入）：

```
请输入一个字符串：Hello! I am a student.
字符数（计空格）：21
字符数（不计空格）：18
```

9.4.2 实例 2——字符串匹配

【问题】 给定一个主串 s 和一个模式（也称为子串）t，在主串 s 中寻找模式 t 的过程称为字符串匹配，也称为模式匹配。如果匹配成功，则返回模式 t 中第一个字符在主串 s 中的序号。例如，模式"am"在主串"I am a student."中的序号是 3。

【想法】 从主串 s 的第一个字符开始和模式 t 的第一个字符进行比较，用变量 start 记录主串中开始比较的位置，如果对应字符相等，则继续比较后续字符；如果不相等，则主串 s 回溯到 start 的下一个位置，模式 t 回溯到第一个字符开始新一趟比较。重复上述过程，若模式 t 中的字符全部比较完毕，则匹配成功，返回 start；否则匹配失败，返回 0。字符串匹配的过程如图 9.8 所示。

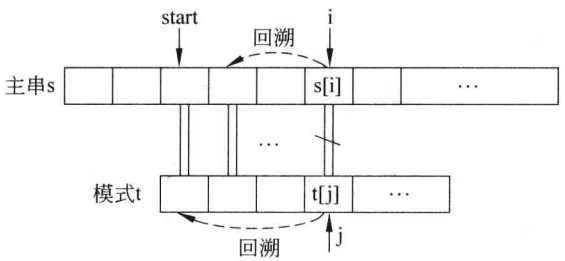


图 9.8 字符串匹配的过程

【算法】 设函数 Cmp 实现字符串匹配，其算法描述如下：

输入：主串 s，模式 t

功能：字符串匹配

输出：模式 t 在主串 s 中的序号

伪代码

```
step1: 初始化主串 s 的起始下标 i = 0;初始化模式 t 的起始下标 j = 0;
step2: 初始化比较的起始位置 start = 0;
step3: 重复下述操作,直到 s 或 t 的所有字符比较完毕:
    step3.1: 如果 s[i]等于 t[j],则继续比较主串 s 和模式 t 的下一对字符;
    step3.2: 将 start 后移一位,将 i 和 j 回溯,准备下一趟比较;
step4: 如果 t 中所有字符都比较完毕,则返回起始位置 start 对应的序号;
       否则,匹配失败,返回 0;
```

【程序】 主函数接收从键盘输入的主串和模式，然后调用函数 Cmp 实现字符串匹配，程序如下：

```
1   /*   字符串匹配.cpp   */
2   #include <stdio.h>                    //使用库函数 printf 和 gets
3   int Cmp(char s[ ], char t[ ]);        //函数声明
4                                         //空行,以下是主函数
5   int main( )
6   {
```

```
    char s[100], t[20];
    int index;
    printf("请输入主串: ");          //输出提示信息
    gets(s);                         //需要接收字符串中的空格
    printf("请输入模式: ");          //输出提示信息
    gets(t);
    index = Cmp(s,t);          //调用函数,返回 t 在 s 中的序号
    if (index == 0)
      printf("匹配不成功! \n");
    else
      printf("匹配成功! %s 在%s 中的序号是: %d\n", t, s, index);
    return 0;                        //将 0 返回操作系统,表明程序正常结束
}
                                     //空行,以下是其他函数定义
int Cmp(char s[ ], char t[ ])   //函数定义,形参为字符数组
{
  int i = 0, j = 0, start = 0;
                                     //start 记录每趟比较的起始位置
  while(s[i] != '\0' && t[j] != '\0')        //当串 s 和串 t 均未结束
  {
    if (s[i] == t[j])                //对应字符相等
    {
      i++; j++;                      //准备比较下一对字符
    }
    else                             //一趟匹配失败,不再比较余下字符
    {
      start++;                       //起始位置加 1
      i = start; j =0;               //i 和 j 分别回溯
    }
  }
  if (t[j] == '\0')
    return (start + 1);              //匹配成功,返回本趟起始位置对应的序号
  else
    return 0;                        //返回匹配失败标志
}
```

运行结果如下（下划线为用户输入）：

```
请输入主串：I am a student.
请输入模式：am
匹配成功! am 在 I am a student.中的序号是: 3
```

习 题 9

一、选择题

1. 以下不能正确进行字符串初始化的语句是（　　）。

A. char str[5] = "good!";　　B. char str[ ] = "good!";

C. char str[8] = "good!";　　D. char str[5] ={'g', 'o', 'o', 'd'};

2. 若有如下变量定义，则正确的叙述为（　　）。

```
char x[ ] = "abcdefg";
char y[ ] = {'a', 'b', 'c', 'd', 'e', 'f', 'g'};
```

A. 数组 x 和数组 y 的值相同　　B. 数组 x 和数组 y 的长度相同

C. 数组 x 的长度大于数组 y 的长度　　D. 数组 x 的长度小于数组 y 的长度

3. 以下程序的输出结果是（　　）。

```
void main( )
{
  char arr[2][4];
  strcpy(arr[0], "you");  strcpy(arr[1], "me");
  arr[0][3] = '&';
  printf("%s\n", arr);
}
```

A. you&me　　B. you　　C. me　　D. arr

4. 假设有如下变量定义 char str1[8], str2[8] = "good"; 则不能实现将字符数组 str2 赋值给 str1 的语句是（　　）。

A. str1 = str2;　　B. strcpy(str1, str2);

C. strncpy(str1, str2, 6);　　D. memcpy(str1, str2, 5);

5. 下列（　　）能够判断两个字符串 str1 和 str2 是否相等。

A. if (str1 == str2)　　B. if (str1 = str2)

C. if (strcmp(str1, str2))　　D. if (strcmp(str1, str2) == 0)

6. 以下程序的输出结果是（　　）。

```
#include <stdio.h>
int main( )
{
  char ch[7] = "89ab12";
  for(int i = 0, s = 0; ch[i] >= '0' && ch[i] <= '9'; i++)
    s = 10 * s + ch[i] - '0';
  printf("%d\n", s);
  return 0;
}
```

A. 89ab12　　B. 8912　　C. 89　　D. 144

7. 以下正确的语句是（　　）。

A. char s[5] = "abcdef";　　B. char *s; gets(s);

C. char s[5]; scanf("%s", &s);　　D. char *s; s = "abcdef";

二、程序设计题

1. 统计字符串中单词的个数，各单词之间用空格分隔，空格数可以是多个。

2. 从键盘输入两个字符串，将其首尾相接后输出。要求不调用库函数 strcat。

3. 在字符串 s 的第 i 个位置插入字符串 t。

4. 用指针作为参数，实现交换两个字符串变量的值。

5. 打字程序。在屏幕上输出一行英文字符串（带空格），然后提示用户原样输入这行字符串，并给出用户输入的正确率。

第10章

自定义数据类型

在实际问题中，某些数据之间是有联系的，基本数据类型一般只能表示单一的数据。为了能够描述更复杂的数据以及数据之间的联系，大多数程序设计语言都允许编程人员根据实际问题自定义数据类型。不同的程序设计语言所允许定义和使用的自定义数据类型有所不同，例如，C/C++语言提供了结构体类型，而 FORTRAN 语言就没有提供这种数据类型。本章介绍 C/C++语言中的枚举类型、结构体类型等常用自定义数据类型。

10.1 可枚举数据的组织——枚举类型

【任务 10.1】荷兰国旗问题

【问题】 要求重新排列一个由 Red、White 和 Blue（这是荷兰国旗的颜色）构成的数组，使得所有的 Red 都排在最前面，White 排在其次，Blue 排在最后。

【想法】 设数组 a[n] 存储 Red、White 和 Blue 三种元素，设置三个参数 i、j、k，其中 i 之前的元素（不包括 a[i]）全部为红色；k 之后的元素（不包括 a[k]）全部为蓝色；j 表示当前元素。则 i 初始化为 0，k 初始化为 n–1，j 初始化为 0。j 从前向后扫描，在扫描过程中根据 a[j]的颜色，将其交换到序列的前面或后面，当 j=k 时，算法结束。荷兰国旗问题的求解思想如图 10.1 所示。

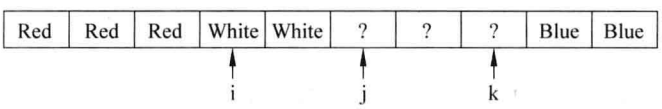


图 10.1 荷兰国旗的求解思想

注意到，当 j 扫描到 Red 时，将 a[i]和 a[j]交换，只有当前面出现连续个 Red 时，交换到位置 j 的元素是 Red，否则交换到位置 j 的元素一定是 White，因此交换后 j 应该加 1；当 j 扫描到 Blue 时，将 a[k]和 a[j]交换，Red、White 和 Blue 均有可能交换到位置 j，则 a[j]需要再次判断，因此交换后不能改变 j。

【算法】 设函数 Sort 实现荷兰国旗问题，其算法描述如下：

输入：数组 a[n]，有 Red、White 和 Blue 三种元素

功能：荷兰国旗

输出：有序数组 a[n]

伪代码

```
step1: 初始化 i = 0; k = n - 1; j = 0;
step2: 当 j <= k 时，依次考查元素 a[j]:
    step2.1 如果 a[j]是 Red，则交换 a[i]和 a[j]; i++; j++;
            否则，如果 a[j]是 Blue，则交换 a[k]和 a[j]; k--;
                  否则 a[j]是 White，则 j++;
```

在设计程序时，有时会遇到某些变量只能在一个有限范围内取值的情况。例如，一周有七天，一年有 12 个月，一副扑克牌的花色有黑桃、红桃、方块和梅花等。所谓枚举，就是把各种可能的取值一一列举出来。如果一个变量只有几种可能的取值，可以将其定义为枚举类型。枚举类型将枚举变量的所有可能取值列举出来，一方面增加了程序的可读性，另一方面限定了变量的取值范围。

10.1.1 枚举类型的定义

【语法】 定义枚举类型的一般形式如下：

自定义类型名 ——— 本质上是符号常量表

enum 枚举类型名 {枚举元素表}; ← 以分号结尾

其中，enum 是关键字；枚举类型名是用户定义的类型标识符，enum 和枚举类型名联合构成枚举类型的名称；枚举元素表由逗号分隔的多个枚举元素组成，每个枚举元素可以看作是用户定义的整型符号常量，其默认值按照定义的顺序依次为 0、1、2、……，如果在定义时指定某个枚举元素的值（称为枚举值），则其后面的枚举元素值依次加 1。

【语义】 定义枚举类型，该类型的取值集合是枚举元素表。

例 10.1 定义枚举类型表示一周的七天。

解：设一周从星期日开始，枚举类型定义如下：

```
enum WeekType {Sun, Mon, Tue, Wed, Thu, Fri, Sat};
```

则 Sun、Mon、Tue、Wed、Thu、Fri、Sat 的枚举值依次为 0、1、2、3、4、5、6。

设一周从星期一开始，枚举类型定义如下：

```
enum WeekType {Mon = 1, Tue, Wed, Thu, Fri, Sat, Sun};
```

则 Mon、Tue、Wed、Thu、Fri、Sat、Sun 的枚举值依次为 1、2、3、4、5、6、7。

10.1.2 枚举变量的定义与初始化

1. 枚举变量的定义

与基本数据类型相同，枚举类型只是一个模板，定义枚举类型并不进行内存分配，在定义枚举变量时才进行内存分配。

【语法】 定义枚举变量的一般形式如下：

```
enum 枚举类型名 变量名列表;
```

其中，enum 枚举类型名是已经定义的枚举类型，注意 enum 不能省略。也可以在定义枚举类型的同时定义枚举变量，其一般形式如下：

```
enum 枚举类型名
{
  枚举元素表
} 变量名列表;
```

其中，变量名列表由逗号分隔的变量名组成。

【语义】 定义枚举变量，并为枚举变量分配存储空间。

例 10.2 假定一周从星期一开始，定义枚举类型表示一周的七天，定义变量 today、nextday 是该枚举类型。

解：以下两种定义方式都是合法的：

```
enum WeekType {Mon = 1, Tue, Wed, Thu, Fri, Sat, Sun};    //定义枚举类型
enum WeekType today, nextday;                              //定义枚举变量
```

```
enum WeekType
{
  Mon = 1, Tue, Wed, Thu, Fri, Sat, Sun
} today, nextday;                        //定义枚举类型同时定义枚举变量
```

2. 枚举变量的初始化

定义枚举变量后，从程序开始执行到为枚举变量赋值之前，该枚举变量是“值无定义的”。可以在定义枚举变量时为枚举变量赋初值，使枚举变量成为“值有定义的”。在定义枚举变量的同时为枚举变量赋初值称为枚举变量的初始化。

【语法】 初始化枚举变量有两种形式，对应定义枚举变量的两种形式。

① 先定义枚举类型，再定义枚举变量并初始化，其一般形式如下：

```
enum 枚举类型名 变量名 = 初值;
```

其中，enum 枚举类型名是已经定义的枚举类型。

② 定义枚举类型的同时，定义枚举变量并初始化，其一般形式如下：

```
enum 枚举类型名
{
  枚举元素表
} 变量名 = 初值;
```

其中，初值为枚举元素而不是枚举值。如果为多个枚举变量初始化，则用逗号分隔。

【语义】 定义并初始化枚举变量。

以下两种形式都是合法的枚举变量初始化：

```
enum WeekType {Mon = 1, Tue, Wed, Thu, Fri, Sat, Sun};     //定义枚举类型
enum WeekType today = Mon;                                  //定义枚举变量并初始化
```

```
enum WeekType
{
  Mon = 1, Tue, Wed, Thu, Fri, Sat, Sun
} today = Mon;                          //定义枚举类型同时定义枚举变量并初始化
```

10.1.3 枚举变量的操作

C/C++语言规定：在为枚举变量赋值时，只能将枚举元素赋给枚举变量，本质上枚举元素是 int 型符号常量，因此，该变量实际得到的是该枚举元素的枚举值。例如，对于如下赋值语句，变量 today 的值为 1：

```
today = Mon;
```

不能直接把枚举值赋给枚举变量，但是，可以将枚举值进行强制类型转换再赋给枚举变量，例如，如下两条赋值语句的结果是相同的：

```
today = Mon;
today = (enum WeekType)1;
```

本质上，枚举类型是缩小范围的 int 型，因此，枚举变量可以执行 int 型数据允许的操作，例如＋、－、*、/等算术运算，以及以值传递方式作为函数的形参。但是，枚举变量不能直接输出枚举元素，一般使用 switch 语句输出枚举值对应的枚举元素。

例 10.3 根据输入的数字在屏幕上输出相应的星期名和下一天的星期名。

解：定义表示一周星期名的枚举类型，然后接收从键盘输入的数字，强制类型转换后赋给枚举变量 today，再输出对应的星期名。下一天的枚举值为 Mon + (today – Mon + 1) % 7，强制类型转换后再赋给枚举变量 nextday。程序如下：

```
1  /*    example10-3.cpp    */
2  #include <stdio.h>                          //使用库函数 printf 和 scanf
3  enum WeekType {Mon=1, Tue, Wed, Thu, Fri, Sat, Sun};//定义枚举类型
4  void PrintDay(enum WeekType day);   //函数声明
5                                      //空行，以下是主函数
6  int main( )
7  {
```

```
    enum WeekType today, nextday;        //定义枚举变量 today 和 nextday
    int index;
    printf("请输入今天是星期几,输入对应数字:");  //输出提示信息
    scanf("%d", &index);
    today = (enum WeekType)index;        //将整数 index 强制转换为枚举元素
    printf("Today is ");
    PrintDay(today);                     //函数调用
    nextday = (enum WeekType)(Mon + (today - Mon + 1)%7);
    printf("Nextday is ");
    PrintDay(nextday);
    return 0;                            //将 0 返回操作系统,表明程序正常结束
}
                                         //空行，以下是其他函数定义
void PrintDay(enum WeekType day)         //函数定义，输出枚举元素
{
  switch (day)                           //根据枚举值输出对应的星期名
  {
    case Mon: printf("Monday"); break;
    case Tue: printf("Tuesday"); break;
    case Wed: printf("Wednesday"); break;
    case Thu: printf("Thursday"); break;
    case Fri: printf("Friday"); break;
    case Sat: printf("Saturday"); break;
    case Sun: printf("Sunday"); break;
    default: break;
  }
  printf("\n");                          //输出换行符
  return;                                //结束函数 PrintDay 的执行
}
```

运行结果如下（下划线为用户输入）：

```
请输入今天是星期几,输入对应数字: 1
Today is Monday
Nextday is Tuesday
```

10.1.4 解决任务 10.1 的程序

在荷兰国旗问题中，数组 a[N]只有三种元素，因此，定义枚举类型 enum Color，枚举元素表为{Red, White, Blue}。主函数首先调用函数 CreatIn 随机生成数组 a[N]，方法是产生[0，2]之间的随机数，强制类型转换后赋给相应的数组元素。然后调用函数 Sort 将数组 a[N]排序，最后调用函数 PrintOut 输出荷兰国旗，即排序后的数组 a[N]。为了将排序前后的数组进行对比，可以在调用函数 Sort 之前输出数组 a[N]。程序如下：

```
/*   duty10-1.cpp   */
#include <stdio.h>                   //使用库函数 printf 和符号常量 NULL
#include <stdlib.h>                  //使用库函数 srand 和 rand
```

```
#include <time.h>                                   //使用库函数 time
#define N 10                                        //定义符号常量 N
enum Color {Red, White, Blue};                      //定义枚举类型
void CreatIn(enum Color a[ ], int n);               //函数声明，随机生成数组
void Sort(enum Color a[ ], int n);                  //函数声明，排序
void PrintOut(enum Color a[ ], int n);              //函数声明，输出荷兰国旗
                                                    //空行，以下是主函数
int main( )
{
  enum Color a[N];                                  //定义枚举类型数组 a[N]
  CreatIn(a, N);                                  //函数调用，随机生成 N 个元素
  printf("初始序列为: ");
  PrintOut(a, N);                                   //函数调用，输出初始序列
  Sort(a, N);                                       //函数调用，将数组 a 排序
  printf("荷兰国旗是: ");
  PrintOut(a, N);                                 //函数调用，输出排序后的序列
  return 0;                                     //将 0 返回操作系统，表明程序正常结束
}
                                                    //空行，以下是其他函数定义
void CreatIn(enum Color a[ ], int n)                //函数定义，形参是一维数组
{
  int temp;
  srand(time(NULL));                              //初始化随机种子为当前系统时间
  for (int i = 0; i < n; i++)
  {
    temp = rand( ) % 3;                             //产生一个 0～2 随机数
    a[i] = (enum Color)temp;                        //强制类型转换为枚举元素
  }
  return;                                           //结束函数 CreatIn 的执行
}
void Sort(enum Color a[ ], int n)                   //函数定义，形参是一维数组
{
  int i = 0, k = n - 1, j = 0;                      //下标 i、j、k 初始化
  enum Color temp;
  while (j < k)
  {
    switch (a[j])                                   //考查当前元素
    {
      case Red:  temp = a[i]; a[i++] = a[j]; a[j++] = temp; break;
      case Blue:  temp = a[j]; a[j] = a[k]; a[k--] = temp; break;
      case White: j++; break;                       //处理下一个元素
    }
  }
  return;                                           //结束函数 Sort 的执行
}
void PrintOut(enum Color a[ ], int n)               //函数定义，形参是一维数组
{
  for (int i = 0; i < n; i++)
  {
```

```
53          switch (a[i])                          //输出数组元素对应的枚举元素
54          {
55            case Red:  printf("Red "); break;
56            case White: printf("White "); break;
57            case Blue:  printf("Blue "); break;
58          }
59        }
60        printf("\n");                              //输出换行符
61        return;                                    //结束函数 PrintOut 的执行
62    }
```

运行结果如下：

```
初始序列为：Red White Blue Red White White Blue Red Blue White
荷兰国旗是：Red Red Red White White White White Blue Blue Blue
```

10.2 不同类型数据的组织——结构体类型

【任务 10.2】统计入学成绩

【问题】 某重点大学的博士入学考试科目为外语和两门专业课，对于每个考生，输入各科考试成绩，并计算总分。

【想法】 输入一个考生的各项信息，再计算总分。

【算法】 设变量 sum 存储考生的总分，算法如下：

伪代码

```
step1: 输入一个考生的各项信息;
step2: sum = 外语成绩 + 专业课 1 成绩 +专业课 2 成绩;
step3: 输出 sum;
```

在实际应用中，一个数据往往由多个分量组成，每个分量描述了数据的某个方面，在概念上构成一个整体。例如，某大学博士入学考试成绩由考号、姓名、外语成绩和两门专业课成绩等分量组成，所有分量组合起来形成了一条完整的学生成绩信息，表达了一个整体的含义。可以用普通变量分别存储数据的各个分量信息，如图 10.2 所示。但是这些变量在内存中占用各自的存储单元，分散存储的变量没有体现概念上的整体（这些变量都属于同一个考生）。

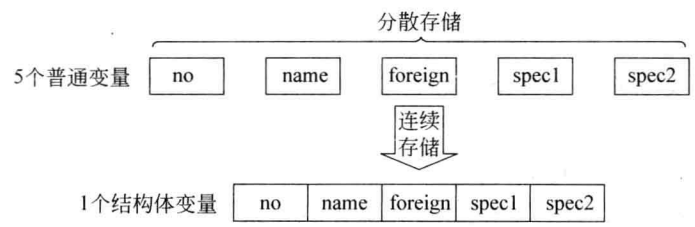
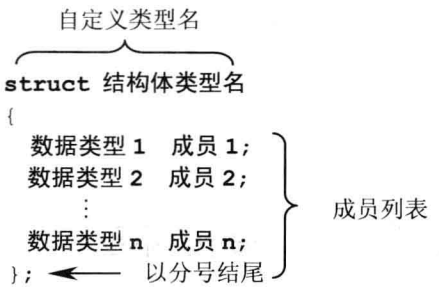


图 10.2 简单变量与结构体变量之间的关系

大多数程序设计语言提供了结构体类型来描述这类数据，习惯上，将分量称为成员、数据项或域。例如，可以用一个结构体变量 student 存储博士入学考试的成绩信息，结构体变量 student 的所有成员在内存中连续存储，如图 10.2 所示。

10.2.1 结构体类型的定义

【语法】 定义结构体类型的一般形式如下：



其中，struct 是关键字，结构体类型名是用户定义的类型标识符，struct 和结构体类型名联合构成结构体类型的名称；成员的数据类型、数量和顺序不限，且成员的数据类型可以是任意合法的数据类型；结构体类型定义必须以分号结尾。

【语义】 定义含有 n 个成员的结构体类型。注意编译器不为类型分配存储空间。

良好的编程习惯 10.1

为了与变量名区分开，本书在为自定义数据类型命名时将每个单词的首字母大写，并且以单词 Type 结尾，例如 StudentType、DateType。

例 10.4 定义结构体类型 DateType 表示日期的年、月、日等信息。

解：结构体类型 DateType 包括 3 个成员 year、month 和 day，定义如下：

```
struct DateType
{
  int year;
  int month;
  int day;
};
```

在定义结构体类型时，成员可以是任意合法的数据类型，如果成员的数据类型是已经定义的结构体类型，则构成结构体类型的嵌套定义。

例 10.5 定义结构体类型 StudentType 表示学生的基本信息，包括姓名、学号、出生日期等信息。

解：在结构体类型 StudentType 中，出生日期包括年、月、日等信息，其数据类型是结构体类型，构成了结构体类型的嵌套定义，具体类型定义如下：

```
struct DateType
{
  int year, month, day;          //与变量定义类似，相同类型的成员可以写在一行
};
struct StudentType
{
  char name[10];
  char no[10];
  struct DateType birthday;      //struct DateType 是已经定义的结构体类型
};
```

10.2.2 结构体变量的定义和初始化

结构体类型是用户自定义数据类型，与基本数据类型具有同样的地位和作用，可以出现在基本数据类型允许出现的任何地方，例如可以用来定义变量的数据类型。结构体类型只是一个模板，本身并不占用内存空间，只有定义结构体变量时才为该变量分配存储空间。

1. 结构体变量的定义

【语法】 定义结构体变量的一般形式如下：

```
struct 结构体类型名 变量名列表;
```

其中，struct 结构体类型名是已经定义的结构体类型，注意 struct 不能省略。也可以在定义结构体类型的同时定义结构体变量，一般形式如下：

```
struct 结构体类型名
{
  成员列表
} 变量名列表;
```

【语义】 定义结构体变量，并为结构体变量分配存储空间。

定义结构体变量后，编译程序会给该变量分配一段连续的存储空间，依次存储结构体变量的各个成员，结构体变量所占存储单元数是各个成员所占存储单元数之和。例如，如下都是合法的结构体变量定义，其存储示意图如图 10.3 所示。

```
struct DateType                        //定义结构体类型 struct DateType
{
  int year, month, day;
};
struct DateType birthday;              //定义结构体变量 birthday

struct DateType
{
  int year, month, day;
} birthday;                  //在定义结构体类型的同时定义结构体变量 birthday
```

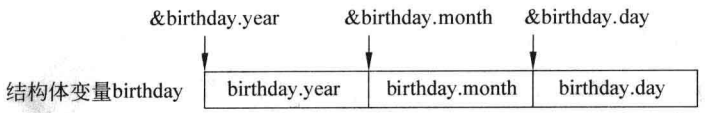


图 10.3　结构体变量 birthday 的存储示意图

2. 结构体变量的引用

C/C++语言不允许对结构体变量进行整体引用，只能引用结构体变量的某个具体成员。

【语法】　引用结构体变量的某个成员，其一般形式如下：

```
结构体变量名.成员名
```

成员运算符

其中，“.”称为成员运算符，且具有左结合性。

【语义】　引用结构体变量的某个成员。

例如，在定义结构体变量 birthday 后，birthday.year、birthday.month 和 birthday.day 分别实现对变量 birthday 各成员的访问。

结构体变量以及各成员都可以用取地址运算符“&”获得具体的存储地址。如图 10.3 所示，&birthday 为结构体变量 birthday 在内存中的起始地址，&birthday.year、&birthday.month 和&birthday.day 分别为每个成员在内存中的起始地址。

3. 结构体变量的初始化

定义结构体变量后，从程序开始执行到为结构体变量赋值之前，该结构体变量是“值无定义的”。可以在定义结构体变量时为结构体变量的各成员赋初值，使结构体变量成为“值有定义的”。在定义结构体变量的同时为结构体变量的各成员赋初值称为结构体变量的初始化。

【语法】　初始化结构体变量有两种形式，分别对应定义结构体变量的两种形式。

① 先定义结构体类型，再定义结构体变量并赋初值，一般形式如下：

```
struct 结构体类型名 变量名 = {初值1,初值2,…,初值n};
```

初值列表

② 定义结构体类型的同时，定义结构体变量并赋初值，一般形式如下：

```
struct 结构体类型名
{
   成员列表
} 变量名 = {初值1,初值2,…,初值n};
```

初值列表

其中，初值列表中的初值顺序与结构体类型定义中的成员顺序必须一一对应。

【语义】 为结构体变量赋初值。

如下都是合法的结构体变量初始化：

```
struct DateType
{
  int year, month, day;
};
struct DateType birthday = {1968, 3, 26};  //初始化结构体变量 birthday
```

```
struct DateType
{
  int year, month, day;
} birthday = {1968, 3, 26};                //初始化结构体变量 birthday
```

10.2.3 结构体变量的操作

1. 输入输出操作

在 C/C++语言中，结构体变量不能作为 scanf 函数和 printf 函数的实参，换言之，不能整体读入一个结构体变量，也不能整体输出一个结构体变量，只能对结构体变量的各个成员进行输入输出操作。例如，如下语句实现结构体变量的输入和输出操作：

```
struct DateType birthday;
scanf("%d%d%d", &birthday.year, &birthday.month, &birthday.day);
printf("%d-%d-%d", birthday.year, birthday.month, birthday.day);
```

2. 赋值操作

如果在定义一个结构体变量时没有进行初始化，则在程序中不可以对结构体变量进行整体赋值，因此，如下语句是错误的：

```
struct DateType birthday;
birthday = {1968, 3, 26};                  ×
```

但可以在程序中为其各个成员逐一赋值。例如：

```
struct DateType birthday;
birthday.year = 1968;
birthday.month = 3;
birthday.day = 26;
```

如果两个结构体变量的类型相同，则可以在程序中将一个结构体变量的值整体赋给另一个结构体变量。例如：

```
struct DateType birthday1 = {1968, 3, 26}, birthday2;
birthday2 = birthday1;
```

3. 其他操作

C/C++语言没有定义施加于结构体类型上的运算，对结构体变量的操作是通过对其成员的操作实现的。本质上，结构体变量的成员是一个简单变量，因此，其成员的使用方法与同类型简单变量的使用方法相同。例如，如下对结构体变量的操作都是正确的：

```
struct DateType birthday1 = {1968, 3, 26}, birthday2;
int year;
birthday2.year = 1963;
year = birthday1.year - birthday2.year;          //两个结构体变量相差的年数
```

例 10.6 将输入的日期按下列格式之一输出：1. 中文（中国）格式；2. 中文（中国台湾）格式；3. 英语（美国）格式；4. 英语（英国）格式。

解：定义结构体类型 struct DateType 表示日期，输入一个日期的年、月、日信息，显示提示信息选择格式，再根据选择输出相应格式的日期。程序如下：

```
/*  example10-6.cpp  */
#include <stdio.h>                          //使用库函数 printf 和 scanf
struct DateType                             //定义结构体类型
{
  int year;
  int month;
  int day;
};
                                            //空行，以下是主函数
int main( )
{
  struct DateType date;                     //定义结构体变量 date
  int select;
  printf("请分别输入一个日期的年、月、日：");
  scanf("%d%d%d", &date.year, &date.month, &date.day);
  printf("1. 中文（中国）格式     2. 中文（中国台湾）格式\n");
  printf("3. 英语（美国）格式     4. 英语（英国）格式\n");
  printf("5. 其他格式\n 请输入对应数字:\n"); //输出提示信息
  scanf("%d", &select);
  switch (select)                           //根据选择输出不同的日期格式
  {
    case 1: printf("中国格式: %d-%d-%d\n", date.year, date.month,
                   date.day);
            break;
    case 2: printf("中国台湾格式:%d/%d/%d\n", date.year, date.month,
                   date.day);
            break;
    case 3: printf("美国格式:%2d/%2d/%d\n", date.month, date.day,
                   date.year);
```

```
            break;
        case 4: printf("英国格式:%2d/%2d/%d\n", date.day, date.month,
                    date.year);
            break;
        default: printf("其他格式:%d-%d-%d\n", date.year, date.month,
                    date.day);
            break;
    }
    return 0;                                    //将 0 返回操作系统,表明程序正常结束
}
```

运行结果如下（下划线为用户输入）：

```
请分别输入一个日期的年、月、日：2009 3 8
1．中文（中国）格式      2．中文（中国台湾）格式
3．英语（美国）格式      4．英语（英国）格式
5．其他格式
请输入对应数字：3
美国格式：3/ 8/2009
```

10.2.4 解决任务 10.2 的程序

```
/*    duty10-2.cpp   */
#include<stdio.h>                                //使用库函数 printf 和 scanf
struct StudentType                               //定义结构体类型
{
  char no[10];                                   //学号 no 是字符串,最多 9 位
  char name[10];                                 //姓名 name 是字符串,最多 4 个汉字
  double foreign;
  double spec1;
  double spec2;
};
                                                 //空行,以下是主函数
int main( )
{
  struct StudentType stu;                        //定义结构体变量 stu
  double sum;
  printf("请输入考生考号：");                    //输出提示信息
  scanf("%s", stu.no);                           //stu.no 是数组名,不用加&
  printf("请输入考生姓名：");                    //输出提示信息
  scanf("%s", stu.name);                         //stu.name 是数组名,不用加&
  printf("请输入考生外语成绩：");                //输出提示信息
  scanf("%lf", &stu.foreign);
  printf("请输入专业课 1 成绩：");               //输出提示信息
  scanf("%lf", &stu.spec1);
  printf("请输入专业课 2 成绩：");               //输出提示信息
  scanf("%lf", &stu.spec2);
```

```
    sum = stu.foreign + stu.spec1 + stu.spec2;
    printf("%s 的总分是%5.1f\n", stu.name, sum);
    return 0;                                    //将 0 返回操作系统,表明程序正常结束
}
```

运行结果如下（下划线为用户输入）：

```
请输入考生考号：20093301
请输入考生姓名：李哲
请输入考生外语成绩：65
请输入专业课 1 成绩：87
请输入专业课 2 成绩：90
李哲的总分是 242.0
```

10.3 批量不同类型数据的组织——结构体数组

【任务 10.3】统计入学成绩（改进版）

【问题】 某重点大学的博士入学考试科目为外语和两门专业课，对于每个考生，输入各科考试成绩，并计算总分。

【想法】 首先输入并保存全体考生的成绩信息，然后再计算并保存每个考生的总分。设数组 stu[N]存放所有考生的成绩信息，则数组元素的类型为结构体类型。

【算法】 设函数 InputMarks 完成输入并保存全体考生的成绩信息，函数 AddMarks 完成统计并保存每个考生的总分，函数 OutputMarks 输出每个考生的总分。各函数对应的算法较简单，请读者自行完成。

数组元素的类型可以是任意合法的数据类型，如果数组元素的类型是结构体类型，这种数组称为**结构体数组**。在实际应用中，常常用结构体数组来描述批量不同类型的数据集合。例如，某大学博士入学考试的成绩信息由考号、姓名、外语成绩和两门专业课成绩等组成，可以用结构体变量来描述考生的成绩信息，假设共有 500 名考生，则全体考生的成绩信息可以用结构体数组来描述。

从本质上讲，结构体数组相当于一个二维表，表结构对应结构体类型，表中每一行信息对应一个数组元素，表中的行数对应结构体数组的长度，如图 10.4 所示。

结构体类型	考号	姓名	外语	专业课一	专业课二
stud[0]	0001	陆宇	87	67	88
stud[1]	0002	李明	68	85	78
⋮	⋮	⋮	⋮	⋮	⋮
stud[499]					

数组长度

图 10.4 结构体数组与二维表的对应关系

10.3.1 结构体数组的定义和初始化

1. 结构体数组的定义

【语法】 定义结构体数组有两种形式，分别对应定义结构体变量的两种形式。

① 先定义结构体类型，再定义结构体数组，一般形式如下：

```
struct 结构体类型名 数组变量名[数组长度];
```

其中，struct 结构体类型名是已经定义的结构体类型，注意 struct 不能省略。

② 在定义结构体类型的同时定义结构体数组，一般形式如下：

```
struct 结构体类型名
{
  成员列表
} 数组变量名[数组长度];
```

【语义】 定义结构体数组变量，并分配存储空间。

如下语句定义了一个结构体数组，其存储示意图如图 10.5 所示。

```
struct StudentType                                    //定义结构体类型
{
  char no[10];
  char name[10];
  double foreign;
  double spec1;
  double spec2;
};
struct StudentType stu[500];                          //定义结构体数组
```

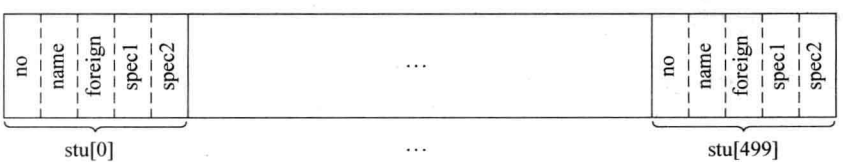


图 10.5 结构体数组的存储示意图

2. 引用结构体数组元素的成员

【语法】 引用结构体数组元素的成员，其一般形式如下：

```
结构体数组名[下标].成员名
  数组                  ↑—— 成员运算符
数组元素即结构体
结构体的某个成员
```

其中，“.”为成员运算符。

【语义】 引用结构体数组元素的某个具体成员。

例如，stu[i].name、stu[i].no、stu[i].foreign 分别表示引用结构体数组元素 stu[i]的 name 成员、no 成员和 foreign 成员。

3. 结构体数组的初始化

与普通数组一样，结构体数组可以在定义时进行初始化。

【语法】 初始化结构体数组有两种形式，分别对应定义结构体变量的两种形式。

① 先定义结构体类型，再定义结构体数组并初始化，一般形式如下：

```
struct 结构体类型名 数组变量名[数组长度] = {{初值表}, ……, {初值表}};
```

其中，struct 结构体类型名是已经定义的结构体类型，注意 struct 不能省略。

② 在定义结构体类型时定义结构体数组并初始化，一般形式如下：

```
struct 结构体类型名
{
  成员列表
} 数组变量名[数组长度] = {{初值表}, ……, {初值表}};
```

其中，初值表是由逗号分隔的数据值集合，每个初值表对应一个数组元素，其数据类型为结构体类型。

【语义】 初始化结构体数组。

以下两种形式都是合法的结构体数组初始化语句：

```
struct StudentType                                          //定义结构体类型
{
  char no[10];
  char name[10];
  double foreign;
  double spec1;
  double spec2;
};
struct StudentType stu[500] ={{0001, 陆  宇, 87, 67, 88},
                              {0002, 李  明, 68, 85, 78},
                              {0003, 汤晓影, 52, 65, 73}};  //以分号结尾
```

```
struct StudentType
{
  char no[10];
  char name[10];
  double foreign;
  double spec1;
  double spec2;
} stu[500] ={{0001, 陆  宇, 87, 67, 88},          //定义结构体数组并初始化
             {0002, 李  明, 68, 85, 78},
             {0003, 汤晓影, 52, 65, 73}};         //注意以分号结尾
```

10.3.2 解决任务 10.3 的程序

```
/*   duty10-3.cpp   */
#include <stdio.h>                          //使用库函数 printf 和 scanf
#define N 2;                                //定义符号常量 N
struct StudentType                          //定义结构体类型
{
  char no[10];                              //学号 no 是字符串,最多 9 位
  char name[10];                            //姓名 name 是字符串,最多 4 个汉字
  double foreign;
  double spec1;
  double spec2;
  double total;                             //total 保存总成绩
};
void InputMarks(struct StudentType student[ ], int n); //录入成绩
void AddMarks(struct StudentType student[ ], int n);   //统计总分
void OutputMarks(struct StudentType student[ ], int n);//输出成绩
                                            //空行,以下是主函数
int main( )
{
  struct StudentType stu[N];                //定义结构体数组
  InputMarks(stu, N);                       //函数调用,实参 stu 为数组首地址
  AddMarks(stu, N);                         //函数调用,实参 stu 为数组首地址
  printf("各个考生的总成绩为: \n");
  OutputMarks(stu, N);                      //函数调用,实参 stu 为数组首地址
  return 0;                                 //将 0 返回操作系统,表明程序正常结束
}
                                            //空行,以下是其他函数定义
void InputMarks(struct StudentType student[ ], int n)
{                                           //函数定义,形参是一维数组
  for (int i = 0; i < n; i++)
  {
    printf("请输入第%d 个考生考号:", i+1);          //输出提示信息
    scanf("%s", student[i].no);      //student[i].no 是数组名,不用加&
    printf("请输入第%d 个考生姓名:", i+1);          //输出提示信息
    scanf("%s", student[i].name); //student[i].name 是数组名,不用加&
    printf("请输入第%d 个考生外语成绩:", i+1);      //输出提示信息
    scanf("%lf", &student[i].foreign);
    printf("请输入第%d 个考生专业课 1 成绩:", i+1);  //输出提示信息
    scanf("%lf", &student[i].spec1);
    printf("请输入第%d 个考生专业课 2 成绩:", i+1);  //输出提示信息
    scanf("%lf", &student[i].spec2);
  }
  return;                                   //结束函数 InputMarks 的执行
}
void AddMarks(struct StudentType student[ ], int n)
```

```
{                                                    //函数定义,形参是一维数组
  for (int i = 0; i < n; i++)                        //依次计算每个考生的总分
  {
    student[i].total=student[i].foreign+student[i].spec1+student[i].spec2;
  }
  return;                                            //结束函数 AddMarks 的执行
}
void OutputMarks(struct StudentType student[ ], int n)
{                                                    //函数定义,形参是一维数组
  for (int i = 0; i < n; i++)                        //依次输出每个考生的成绩
  {
    printf("%s 的总分是%5.1f\n", student[i].name, student[i].total);
  }
  return;                                          //结束函数 OutputMarks 的执行
}
```

运行结果如下（下划线为用户输入）：

```
请输入第 1 个考生考号：20093301
请输入第 1 个考生姓名：李哲
请输入第 1 个考生外语成绩：65
请输入第 1 个考生专业课 1 成绩：87
请输入第 1 个考生专业课 2 成绩：90
请输入第 2 个考生考号：20093302
请输入第 2 个考生姓名：王奇
请输入第 2 个考生外语成绩：60
请输入第 2 个考生专业课 1 成绩：82
请输入第 2 个考生专业课 2 成绩：88
李哲的总分是 242.0
王奇的总分是 230.0
```

10.4 为自定义数据类型定义别名

枚举类型、结构体类型都是用户自定义的数据类型，与基本数据类型不同的是，自定义数据类型需要先进行类型定义，而且在变量定义时还需要说明是何种自定义数据类型，例如，枚举类型需要关键字 enum，结构体类型需要关键字 struct。C/C++语言提供了类型定义机制，用存储类型说明符 typedef 为自定义数据类型定义别名，然后自定义数据类型就可以像基本数据类型一样使用。

【语法】 typedef 的一般形式如下：

```
typedef 自定义数据类型 类型名;
```

其中，自定义数据类型是一个已经定义或正在定义的数据类型；类型名是一个合法的标识符。

【语义】 为自定义数据类型定义类型名，相当于为自定义数据类型定义别名。

例如，可以先定义枚举类型 enum Week，再为 enum Week 定义新类型名：

```
enum Week {Mon = 1, Tue, Wed, Thu, Fri, Sat, Sun}; //定义枚举类型
typedef enum Week WeekType;                        //为枚举类型定义类型名
```

也可以在定义枚举类型的同时为其定义新类型名：

```
typedef enum                                       //可以省略标识符 Week
{
  Mon = 1, Tue, Wed, Thu, Fri, Sat, Sun
} WeekType;                                        //定义枚举类型的同时为其定义新类型名
```

然后可以直接用新类型名 WeekType 定义枚举变量：

```
WeekType today;                                    //直接用新类型名定义变量 today
```

例如，可以先定义结构体类型 struct Date，再为 struct Date 定义新类型名：

```
struct Date                                        //定义结构体类型
{
  int year, month, day;
};
typedef struct Date DateType;                      //为结构体类型定义新类型名 DateType
```

也可以在定义结构体类型的同时为其定义新类型名：

```
typedef struct                                     //可以省略标识符 Date
{
  int year, month, day;
} DateType;                                        //定义结构体类型的同时为其定义新类型名
```

然后可以直接用新类型名 DateType 定义结构体变量：

```
DateType birthday;                                 //直接用新类型名定义结构体变量 birthday
```

应用类型定义机制还可以为基本数据类型定义别名，例如，可以将 float 型重新命名为 REAL 型：

```
typedef float REAL;                                //为 float 起别名 REAL
REAL a = 2.5, b = 3.6;                             //相当于 float a = 2.5, b = 3.6;
```

需要强调的是，类型定义机制仅仅是为一个已经存在的数据类型定义别名，所定义的新类型名仍然是一个模板，即在类型定义时并不存在任何实体，没有分配存储空间，只有在变量定义时才为变量进行内存分配。

10.5 程序设计实例

10.5.1 实例 1——最近对问题

【程序】 最近对问题要求在包含 n 个点的集合中找出距离最近的两个点。在空中交通控制问题中，若将飞机作为空间移动的一个点来处理，则具有最大碰撞危险的两架飞机，就是这个空间中最接近的一对点。这类问题是计算几何中研究的基本问题之一。

【想法】 简单起见，只考虑二维的情况，并假设每个点是以标准笛卡儿坐标形式 (x, y) 给出的，两个点 $P_i=(x_i, y_i)$和 $P_j=(x_j, y_j)$之间的距离是标准的欧氏距离：

$$d=\sqrt{(x_i-x_j)^2+(y_i-y_j)^2} \tag{10.1}$$

定义结构体数组 pot[N]存储 N 个点的坐标，分别计算每一对点之间的距离，然后找出距离最小的那一对。为了避免对同一对点重复计算，只考虑 $j<i$ 的那些点对(P_i, P_j)。

【算法】 设函数 MinPot 实现在 N 个点中找出距离最近的点对，其算法描述如下：

输入：N 个点的坐标 pot[N]

功能：最近对问题

输出：最近点对的序号 minI 和 minJ，以及最近距离 minDist

伪代码

```
step1: 初始化最近距离 minDist = MAX;
step2: 初始化最近对点序号 minI = 0; minJ = 0;
step3: 循环变量 i 从 0 ~ n - 1，计算点 i 和其他所有点之间的距离:
    step3.1: 循环变量 j 从 0 ~ i - 1,计算点 i 和点 j 之间的距离:
        step3.1.1: dist = 点 i 和点 j 之间的距离;
        step3.1.2: 如果 dist 小于 minDist,则 minDist = dist; minI = i; minJ =j;
        step3.1.3: j++;
    step3.2: i++;
step4: 输出 minDist, minI 和 minJ;
```

【程序】 标准笛卡儿坐标中的点由横坐标和纵坐标唯一确定，因此，定义枚举类型 PointType，包括横坐标 x 和纵坐标 y 两个成员。主函数首先调用函数 CreatPot 随机产生 N 个点，每个点的横坐标和纵坐标的区间是[0，99]，然后调用函数 MinPot 求最近对。函数 MinPot 有三个返回值，分别是最近距离和两个最近点的序号，用指针传递方式返回两个最近点的序号，最近距离作为函数 MinPot 的返回值。程序如下：

```
1  /*   最近对问题.cpp   */
2  #include <stdio.h>                  //使用库函数 printf 和符号常量 NULL
3  #include <stdlib.h>                 //使用库函数 srand 和 rand
4  #include <time.h>                   //使用库函数 time
5  #include <math.h>                   //使用库函数 sqrt
6  #define N 10                        //定义符号常量 N
```

```
typedef struct                          //定义结构体类型 PointType
{
  int x, y;
} PointType;
void CreatPot(PointType pot[ ], int n); //随机产生 n 个点
double MinPot(PointType pot[ ], int n, int *p, int *q);
                                        //求最近点对
                                        //空行,以下是主函数
int main( )
{
  PointType pot[N];                     //定义结构体数组 pot[N]
  double min;
  int minI, minJ;                       //存放最近点对的下标
  CreatPot(pot, N);                    //函数调用,实参 pot 是数组首地址
  printf("产生的随机点是: \n");
  for (int i = 0; i < N; i++)           //输出随机产生的 N 个点
  {
    printf("%2d (%2d, %2d)\n", i + 1, pot[i].x, pot[i].y);
  }
  min = MinPot(pot, N, &minI, &minJ);  //函数调用,min 接收返回值
  printf("最近点是第%d个点和第%d个点,距离是%5.2f\n", minI, minJ, min);
  return 0;                            //将 0 返回操作系统,表明程序正常结束
}
                                        //空行,以下是其他函数定义
void CreatPot(PointType pot[], int n)   //函数定义,形参是一维数组
{
  srand(time(NULL));                    //初始化随机种子为当前系统时间
  for (int i = 0; i < n; i++)           //产生 n 个点
  {
    pot[i].x = rand( ) % 100;           //随机产生[0, 99]之间的随机数
    pot[i].y = rand( ) % 100;
  }
  return;                               //结束函数 CreatPot 的执行
}
double MinPot(PointType pot[ ], int n, int *p, int *q)
{                      //函数定义,形参是一维数组,形参 p 和 q 是指针传递形式
  int i, j, minI, minJ;
  double dist, minDist = 1000;    //初始化最近距离为一个足够大的数 1000
  for (i = 0; i < n; i++)
     for (j = 0; j < i; j++)      //为避免重复计算,只考虑 j<i 的那些点对
     {                                  //以下 2 行是一条语句
          dist = (pot[i].x - pot[j].x) * (pot[i].x - pot[j].x) +
                 (pot[i].y - pot[j].y) * (pot[i].y - pot[j].y);
          dist = sqrt(dist);            //调用库函数求根号
          if (dist < minDist)
          {
            minDist = dist;             //minDist 保存当前最短距离
            minI = i; minJ = j;         //保存当前最近点对的下标
          }
```

```
55            }
56        *p = minI; *q = minJ;                    //保存最终结果
57        return minDist;        //结束函数的执行,并将最近距离 minDist 返回到调用处
58    }
```

运行结果如下：

```
产生的随机点是:
1 (76, 74)
2 (18, 68)
3 (11, 26)
4 ( 2, 29)
5 (18, 15)
6 (61, 54)
7 (84, 87)
8 (60, 83)
9 (78, 95)
10(83, 32)
最近点是第 3 个点和第 2 个点,距离是 9.49
```

10.5.2 实例 2——手机电话簿

【问题】 模拟手机电话簿管理中的新建和查询功能，假设手机最多可以存储 50 个联系人，每个联系人信息由姓名和电话号码组成。

【想法】 定义结构体类型 TeleType 保存联系人的姓名 name 和电话号码 number，由于最多存储 50 个联系人，需定义一个结构体数组 people[50]。初始时联系人总数为 0，新建联系人即是将联系人的姓名和电话号码等信息追加到数组 people 中，查询电话号码可以在数组 people 中进行顺序查找。

【算法】 设函数 NewBuild 实现新建联系人功能，其算法描述如下：

输入：结构体数组 people，当前联系人总数 Count

功能：新建联系人

输出：无

伪代码

```
step1: 输入姓名和电话号码;
step2: 将姓名和电话号码追加到数组 people 中;
step3: Count++;
```

设函数 Search 实现查询功能，其算法描述如下：

输入：结构体数组 people，当前联系人总数 Count

功能：查询电话号码

输出：如果查找成功，则输出相应的电话号码；否则输出查找失败信息

伪代码

```
step1: 输入要查找的姓名 name;
step2: 下标 i 为 0 ~ Count -1, 在数组 people 中依次查找:
    step2.1: 如果people[i].name等于name,则查找成功,返回people[i].number;
    step2.2: i++
step3: 输出查找失败信息;
```

【程序】 由于函数 NewBuild 和 Search 都要使用当前联系人总数，设变量 Count 为全局变量。程序如下：

```
/*   手机电话簿.cpp   */
#include <stdio.h>                          //使用库函数 printf 和 scanf
#include <string.h>                         //使用库函数 strcmp
int Count = 0;                              //定义全局变量 Count 并初始化为 0
typedef struct                              //定义结构体类型 TeleType
{
  char name[10];                            //保存姓名,最多 4 个汉字
  char number[12];                          //保存电话号码,手机号码为 11 位
} TeleType;
void NewBuild(TeleType people[ ]);          //函数声明,新建联系人
void Search(TeleType people[ ]);            //函数声明,查询电话
                                            //空行，以下是主函数
int main( )
{
  int select;
  TeleType people[50];          //定义结构体数组 people,最多 50 个联系人
  do                                        //重复执行,直到从键盘输入 0
  {
    printf("1. 新建功能  2. 查询功能  0.退出\n");     //输出提示信息
    printf("请选择操作:");
    scanf("%d", &select);                   //输入数字以选择对应功能
    switch (select)
    {
      case 1: if (Count < 49) NewBuild(people); //人数未满可新建联系人
              else printf("人数已满,不能新建联系人!\n");
              break;
      case 2: if (Count > 0) Search(people);   //人数不为零时可以查询
              else printf("未存储联系人信息,无法查询!\n");
              break;
      case 0: printf("退出程序!\n"); break;
      default: printf("输入错误,请重新输入!\n"); break;
    }
  } while(select != 0);
  return 0;                                 //将 0 返回操作系统,表明程序正常结束
}
                                            //空行,以下是其他函数定义
void NewBuild(TeleType people[ ])//函数定义,形参为一维数组
{
  printf("请输入姓名: ");                   //输出提示信息
```

```
    scanf("%s", people[Count].name);    //接收从键盘输入的字符串
    printf("请输入电话号码: ");          //输出提示信息
    scanf("%s", people[Count].number); //接收从键盘输入的字符串
    Count++;                            //联系人总数加 1
    printf("新建成功!\n");
    return;                             //结束函数 NewBuild 的执行
  }
  void Search(TeleType people[ ])       //函数定义,形参为一维数组
  {
    char name[10];
    printf("请输入要查询的姓名: ");      //输出提示信息
    scanf("%s", name);
    for (int i = 0; i < Count; i++)     //依次遍历每个联系人信息
    {
      if (strcmp(name, people[i].name) == 0)  //两个字符串相等,匹配成功
      {
        printf("查询成功! ");
        printf("%s 的联系电话是%s\n", people[i].name, people[i].number);
        break;
      }
    }
    if (i >= Count)
      printf("没有存储%s 的信息,查找失败!\n", name);  //遍历所有元素未找到
    return;                                          //结束函数 Search 的执行
  }
```

运行结果如下（下划线为用户输入）：

```
1. 新建功能   2. 查询功能   0. 退出
请选择操作: 1
请输入姓名: 王亮
请输入电话号码: 8571234
新建成功!
1. 新建功能   2. 查询功能   0. 退出
请选择操作: 2
请输入要查询的姓名: 王亮
查找成功! 王亮的联系电话是 8571234
1. 新建功能   2. 查询功能   0. 退出
请选择操作: 0
退出程序!
```

习 题 10

一、选择题

1. 下列说法正确的是（　　）。

A. 结构体类型一经定义，系统就为其分配内存单元

B．结构体变量所占内存单元数是各成员所占内存单元数之和

C．可以对结构体变量进行赋值和存取等运算

D．定义结构体变量后，只能引用结构体变量的具体成员，不能引用结构体变量

2. 对于枚举类型 enum ColorType {red, green, blue, yellow = 7, black}; 则枚举常量 red 和 black 的值分别是（　　）。

A．1, 5　　B．1, 7　　C．0, 8　　D．1, 8

3. 以下对结构体变量 day 的定义中，正确的是（　　）。

A．
```
struct Date
{
  int x, y;
} day;
```

B．
```
typedef struct
{
  int x, y;
} day;
```

C．
```
struct
{
  int x, y;
} Date;
struct Date day;
```

D．
```
struct Date
{
  int x, y;
};
struct day;
```

4. 对于如下结构体变量定义，能够正确实现输入操作的语句是（　　）。

```
struct StudentType
{
  char name[10];
  double score[3];
}stu;
```

A. scanf("%s", stu.name);　　B. scanf("%lf ", stu.score);

C. scanf("%s", &stu.name);　　D. scanf("%lf ", &stu.score);

5. 结构体变量在程序执行期间（　　）。

A. 所有成员都驻留在内存中　　B. 只有一个成员驻留在内存中

C. 部分成员驻留在内存中　　D. 正在使用的成员驻留在内存中

二、程序设计题

1. 假设婚姻状况有以下 4 种：已婚（married）、离婚（divorced）、孀居（widowed）和单身（single），定义枚举类型描述婚姻状况。

2. 建立一个职工基本情况统计表，基本情况包括工作证编号、姓名、性别、年龄等内容，要求统计：①女职工占总职工数的比例；②所有职工的平均年龄；③30～40、40～50、50～60 这三个年龄段的职工人数。

第11章

再谈函数

设计一个复杂的应用程序时，往往把程序划分成若干个功能较为单一的程序模块，然后分别实现各个模块，再把所有模块按照程序逻辑装配起来，这就是模块化程序设计。在 C/C++语言中，函数是实现模块化的基本手段，是程序的组成单位，函数通过互相调用体现其程序逻辑，完成数据处理。

11.1 函数的嵌套调用

【任务 11.1】字符串的循环左移

【问题】 将一个字符串向左循环移动 i 个位置，例如，将“abcdefg”向左循环移动 3 个位置为“defgabc”。这个算法有很多应用，例如，在文本编辑器中移动行的操作，磁盘整理时交换两个不同大小的相邻内存块等。

【想法】 可以通过下述方法将长度为 n 的字符串循环左移 i 位：先将字符串中的前 i 个字符置逆，再将后 $n-i$ 个字符置逆，最后将整个字符串置逆。例如，将字符串“abcdefg”循环左移 3 位，则先将“abc”置逆为“cba”，再将“defg”置逆为“gfed”，最后将“cbagfed”置逆为“defgabc”。

【算法】 设函数 Converse 完成循环左移操作，其算法描述如下：

输入：字符串 str，循环左移位数 i

功能：字符串循环左移

输出：循环左移后的字符串

伪代码

```
step1: n = 字符串 str 的长度;
step2: 将字符串 str 的前 i 个元素逆置;
step3: 将字符串 str 的后 n-i 个元素逆置;
step4: 将字符串 str 的所有元素逆置;
```

将函数 Converse 中字符串置逆操作独立为函数 Reverse，其算法描述如下：

输入：字符串 str，置逆的起始位置 low 和终止位置 high

功能：将字符串置逆

输出：置逆后的字符串

伪代码

```
step1: 计算置逆区间的中间位置 mid = (high - low) / 2;
step2: 循环变量 i 为 0～mid，重复执行下述操作：
    step2.1: 将位置 low + i 的字符与位置 high - i 的字符交换;
    step2.2: i++;
```

函数的嵌套调用就是在函数的执行过程中再嵌套调用其他函数，例如，在 Converse 函数的执行过程中嵌套调用 Reverse 函数。本质上，函数的嵌套调用过程和一般函数的调用过程相同。

11.1.1 函数的嵌套调用

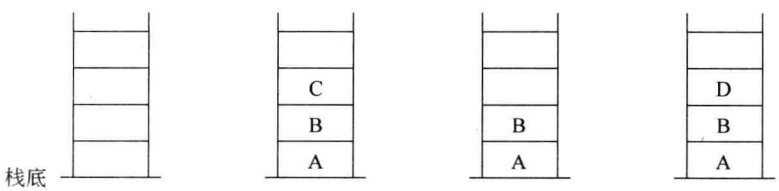
为了简化编译程序的工作，C/C++语言规定：函数定义是相互独立的，不允许在函数内部定义另一个函数，即函数不能嵌套定义，如图 11.1 所示。

(a) 正确——平行定义　　(b) 错误——嵌套定义

图 11.1　C/C++语言对函数定义的规定

虽然函数不能嵌套定义，但函数可以嵌套调用。函数的嵌套调用是在函数的执行过程中再调用其他函数。例如，在函数 A 尚未执行结束时调用函数 B，在函数 B 尚未执行结束时调用函数 C。那么，当函数 C 执行结束时，应该返回到什么位置呢？为保证函数嵌套调用的正确执行，系统自动设立了工作栈保存返回信息。

栈是限定仅在一端进行插入（称为进栈、入栈）和删除（称为出栈）操作的线性序列，允许插入和删除的一端称为栈顶，另一端称为栈底。图 11.2(a)所示为一个空栈，栈内没有数据。图 11.2(b)为 A、B、C 进栈后的状态。由于出栈只能在栈顶进行，则执行删除操作是将栈顶数据 C 出栈，如图 11.2(c)所示。由于入栈只能在栈顶进行，则 D 进栈后成为当前的栈顶数据，如图 11.2(d)所示。



(a) 空栈　(b) A、B、C进栈　(c) C出栈　(d) D进栈

图 11.2　栈的操作示意图

在程序开始执行时，系统自动设立工作栈；在函数调用时，系统首先将当前函数（即

调用函数）的运行环境和返回地址进栈，然后再执行被调用函数，所谓运行环境就是本函数运行时占用的存储空间，包括形参、局部变量等占用的存储空间；在被调用函数执行结束后，首先将工作栈的栈顶数据出栈，恢复调用函数的运行环境和返回地址，使相应的形参和局部变量恢复为调用前的值，然后继续执行调用函数。下面通过一个例子说明函数嵌套调用过程中工作栈的变化过程。

例 11.1 公共子序列。字符串匹配是严格的匹配，即强调模式在主串中的连续性，例如，模式"bc"是主串"abcd"的子串，而"ac"就不是主串"abcd"的子串。但在实际应用中，有时不需要模式的连续性，例如，模式"哈工大"与主串"哈尔滨工业大学"是非连续匹配的，称模式"哈工大"是主串"哈尔滨工业大学"的子序列。要求编写程序，判断给定的模式是否为两个主串的公共子序列。

解：分别判断模式 t 是否是主串 s1 和 s2 的子序列，如果模式 t 是主串 s1 的子序列，同时模式 t 也是主串 s2 的子序列，则模式 t 是主串 s1 和 s2 的公共子序列。

【算法】 设 s1 和 s2 分别表示主串 1 和主串 2，t 表示模式，函数 CommonString 实现判断 t 是否为主串 s1 和 s2 的公共子序列，则函数 CommonString 需要分别判断模式 t 是否是主串 s1 和 s2 的子序列，将判断子序列的功能独立为函数 Cmp。函数 CommonString 的算法较简单，请读者完成，函数 Cmp 的算法描述如下：

输入：主串 strA，模式 strB

功能：判断子序列

输出：如果 strB 是 strA 的子序列，则返回 1，否则返回 0

伪代码

```
step1: 初始化比较的起始位置 i = 0; j = 0;
step2: length1 = 字符串 strA 的长度; length2 = 字符串 strB 的长度;
step3: 当 i < length1 并且 j < length2 时重复执行下述操作:
    step3.1: 如果 strA[i]等于 strB[j], 则 i++; j++;
    step3.2: 否则 i++;
step4: 如果 j 等于 length2, 说明 strB 中字符全部匹配, 返回 1;否则返回 0;
```

【程序】 采用字符数组存储模式、主串 1 和主串 2，程序如下：

```
1   /*   example11-1.cpp   */
2   #include <stdio.h>                          //使用库函数 printf 和 gets
3   #include <string.h>                         //使用库函数 strlen
4   int CommonString(char str1[], char str2[],char str3[]);//函数声明
5   int Cmp(char strA[], char strB[]);          //函数声明
6                                               //空行,以下是主函数
7   int main( )
8   {
9     char s1[80], s2[80], t[10];//s1 存放主串 1, s2 存放主串 2, t 存放模式
10    int flag = 0;        //flag 是匹配标志,0 表示 t 不是 s1 和 s2 的公共子序列
11    printf("请输入主串 1:");                   //输出提示信息
12    gets(s1);                                 //主串 s1 可以包含空格
13    printf("请输入主串 2:");                   //输出提示信息
14   gets(s2);                                  //主串 s2 可以包含空格
```

```
    printf("请输入待匹配字符串:");
    gets(t);                                  //模式 t 也可以包含空格
    flag = CommonString(s1, s2, t);//函数调用,三个实参均是字符数组首地址
    if (flag == 1)
      printf("匹配成功!\"%s\"是公共子串\n", t);  //\"为转义字符,表示"
    else
      printf("匹配不成功!\"%s\"不是公共子串\n", t);
    return 0;                                 //将 0 返回操作系统,表明程序正常结束
}
                                              //空行,以下是其他函数定义
int CommonString(char str1[], char str2[], char str3[])
{       //函数定义,形参数组 str1 和 str2 为主串,形参数组 str3 为模式
  int flag1 = 0, flag2 = 0;                   //flag1 和 flag2 均是匹配标志
  flag1 = Cmp(str1, str3);                 //函数调用,判断 str3 是否是 str1 的子序列
  flag2 = Cmp(str2, str3);                 //函数调用,判断 str3 是否是 str2 的子序列
  if (flag1 == 1 && flag2 == 1)
    return 1;                                 //str3 是 str1 和 str2 的公共子序列
  else
    return 0;                                 //str3 不是 str1 和 str2 的公共子序列
}
int Cmp(char strA[ ], char strB[ ])
{                    //函数定义,形参数组 strA 是主串,形参数组 strB 是模式
  int i = 0, j = 0;                           //初始化主串和模式的下标
  int length1, length2;
  length1 = strlen(strA);                     //求得主串 strA 的长度
  length2 = strlen(strB);                     //求得模式 strB 的长度
  while (i < length1 && j < length2) //主串和模式均未结束
  {
    if (strA[i] == strB[j])                   //对应字符相等
    {
      i++;  j++;                              //准备比较主串和模式的下一对字符
    }
    else
      i++;              //字符 strB[j]尚未匹配,准备和主串的下一个字符比较
  }
  if (j == length2)                           //strB 中所有字符比较完毕
    return 1;                                 //strB 是 strA 的子序列
  else
    return 0;                                 //strB 不是 strA 的子序列
}
```

运行结果如下（下划线为用户输入）：

```
请输入主串 1：哈尔滨工业大学
请输入主串 2：哈尔滨工程大学
请输入待匹配字符串：哈工大
匹配成功！"哈工大"是公共子串
```

程序 example11-1.cpp 的具体执行过程如下，栈中数据的最后一项表示返回地址，为简单起见，返回地址用该语句所在行号表示：

① 主函数用变量 s1、s2 和 t 接收从键盘输入的三个字符串：哈尔滨工业大学、哈尔

滨工程大学、哈工大，然后执行函数调用 CommonString(s1, s2, t)，调用之前将本层函数的运行环境进栈，如图 11.3(a)所示；

② 在函数 CommonString(s1, s2, t)的执行过程中，进行函数调用 Cmp(str1, str3)，调用之前将本层函数的运行环境进栈，如图 11.3(b)所示；

③ 函数 Cmp(str1, str3)执行完毕，将匹配成功信息“1”返回给调用函数，将工作栈的栈顶数据出栈，恢复调用前的运行环境，如图 11.3(c)所示；

④ 继续执行函数 CommonString(s1, s2, t)，再进行函数调用 Cmp(str2, str3)，调用之前将本层函数的运行环境进栈，如图 11.3(d)所示；

⑤ 函数 Cmp(str2, str3)执行完毕，将匹配成功信息“1”返回给调用函数，将工作栈的栈顶数据出栈，恢复调用前的运行环境，如图 11.3(e)所示；

⑥ 继续执行函数 CommonString(s1, s2, t)，将匹配成功信息“1”返回给调用函数，将工作栈的栈顶数据出栈，恢复调用前的运行环境，如图 11.3(f)所示；

⑦ 主函数执行完毕，输出匹配成功信息。

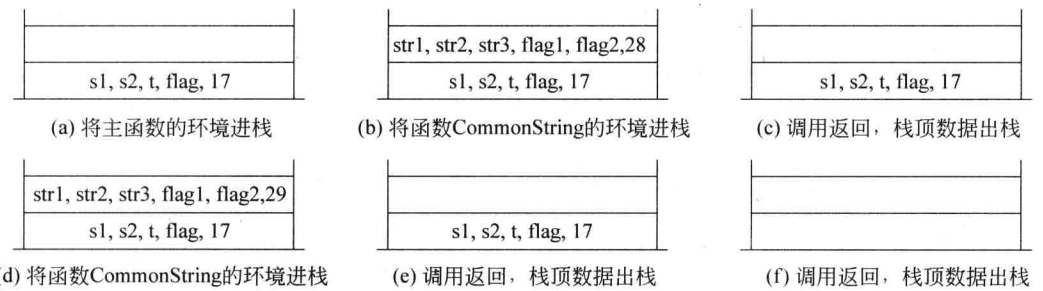


图 11.3　函数嵌套调用过程中工作栈的变化示意图

采用图示方法描述函数调用的运行轨迹，从中可较直观地了解到各调用层次及执行情况，具体方法如下：

① 写出函数当前调用层执行的各条语句，并用有向弧表示语句的执行顺序；

② 对函数调用，从调用语句处画一条实线有向弧指向被调用函数入口，表示调用路线，从被调用函数末尾处画一条虚线有向弧指向调用语句的下面，表示返回路线；

③ 在调用路线上标出本次调用的顺序号，在返回路线上标出本层调用的返回值。

程序 example11-1.cpp 在输入主串 1、主串 2、模式分别为哈尔滨工业大学、哈尔滨工程大学、哈工大的执行过程如图 11.4 所示。

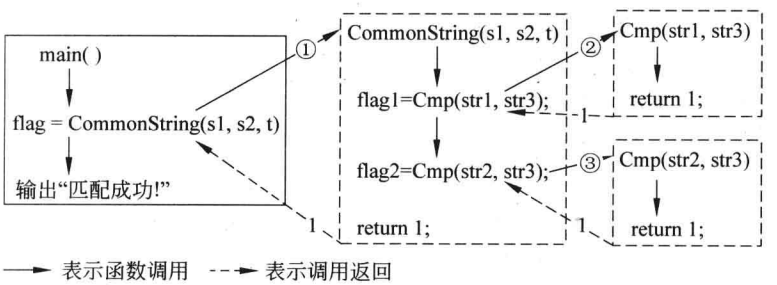


图 11.4　函数嵌套调用的执行过程

C/C++语言对函数嵌套调用的层数未加限制，但嵌套调用的层数过多会影响程序的执行效率。

11.1.2 解决任务 11.1 的程序

```
/*   duty11-1.cpp   */
#include <stdio.h>                          //使用库函数 printf 和 scanf
#include <string.h>                         //使用库函数 strlen
void Reverse(char ch[ ], int low, int high); //函数声明
void Converse(char ch[ ], int i);           //函数声明
                                            //空行，以下是主函数
int main( )
{
  char ch[50];
  int i;
  printf("请输入一个字符串:");              //输出提示信息
  scanf("%s", ch);                          //不接收空格
  printf("请输入循环左移的位数:");          //输出提示信息
  scanf("%d", &i);
  Converse(ch, i);                          //函数调用,将字符串 ch 循环左移 i 位
  printf("循环左移%d 位后的字符串为: %s\n", i, ch);
  return 0;                                 //将 0 返回操作系统,表明程序正常结束
}
                                            //空行,以下是其他函数定义
void Converse(char ch[ ], int i)            //函数定义,形参是一维数组
{
  int n = strlen(ch);                       //求得字符串 ch 的长度
  Reverse(ch, 0, i-1);                      //函数调用,将字符串 ch 从 0~i-1 置逆
  Reverse(ch, i, n-1);                      //函数调用,将字符串 ch 从 i~n-1 置逆
  Reverse(ch, 0, n-1);                      //函数调用,将字符串 ch 从 0~n-1 置逆
  return;                                   //结束函数 Converse 的执行
}
void Reverse(char ch[], int low, int high)
{                                           //函数定义,置逆区间是[low, high]
  char temp;                                //temp 为暂存变量
  for (int i = 0; i <= (high-low)/2; i++)   //置逆区间中点是(high-low)/2
  {                  //以下三条语句实现交换 ch[low + i]和 ch[high - i]
    temp = ch[low + i]; ch[low + i] = ch[high - i]; ch[high - i]= temp;
  }
  return;                                   //结束函数 Reverse 的执行
}
```

运行结果如下（下划线为用户输入）：

```
请输入一个字符串：abcdefg
请输入循环左移的位数：3
循环左移 3 位后的字符串为：defgabc
```

11.2 函数的递归调用

【任务 11.2】Fibonacci 数列

【问题】 Fibonacci（斐波那契）数列是一个经典问题：把一对兔子（雌雄各 1 只）放到围栏中，从第 2 个月以后，每个月这对兔子都会生出一对新兔子，其中雌雄各 1 只，从第 2 个月以后，每对新兔子每个月都会生出一对新兔子，也是雌雄各 1 只，问一年后围栏中有多少对兔子？

【想法】 令 $f(n)$表示第 n 个月围栏中兔子的对数，显然第 1 个月有 1 对，由于每对新兔子在第 2 个月以后才可以生出兔子，因此，第 2 个月仍然有 1 对，第 n 个月时，那些在第 n–1 个月就已经在围栏中的兔子仍然存在，第 n–2 个月就已经在围栏中的每对兔子都会生出一对新兔子，即 $f(n)=f(n-1)+f(n-2)$。因此，斐波那契数列存在如下递推式：

$$f(n)=\begin{cases}1 & n=1\\ 1 & n=2\\ f(n-1)+f(n-2) & n>2\end{cases} \tag{11.1}$$

【算法】 设函数 Fib 求解第 n 个月时围栏中兔子的对数，其算法描述如下：

输入：月数 n

功能：求 Fibonacci 数列

输出：兔子的个数 fib

伪代码

```
step1: 如果 n 等于 1 或 2, 则 fib = 1;
step2: 否则, fib = Fib(n - 1) + Fib(n - 2);
step3: 返回 fib;
```

从静态行文的角度看，在定义一个函数时，若在函数体中出现对函数自身的调用，则称该函数是**递归函数**；从动态执行的角度看，在调用一个函数时，若被调用函数尚未结束，又出现对函数自身的调用，则称该调用是**递归调用**。递归是程序设计的一种重要方法，递归程序通过不断调用自己，将待求解问题转化为解法相同的子问题，最终实现问题求解。

11.2.1 函数的递归调用

1. 递归的定义

递归就是函数直接调用自己或通过一系列调用语句间接调用自己，是一种描述问题

和解决问题的基本方法。递归方法的基本思想是：将一个难以直接解决的原问题分解为两部分，一部分是规模（即输入量）足够小的子问题，可以直接求解；另一部分是一些规模较小的子问题，子问题的解决方法与原问题的解决方法相同。如果子问题的规模仍然不够小，则再将子问题分解为规模更小的子问题，如此分解下去，直到问题规模足够小，可以直接求解为止，再将子问题的解合并为一个更大规模的子问题的解，自底向上逐步求出原问题的解。递归过程不能无限地进行下去，因此，递归有两个基本要素：

① 递归出口：确定递归到何时终止，即递归的结束条件。

② 递归体：确定递归的方式，即原问题是如何分解为子问题的。

2. 递归程序适用的情况

很多问题本身就是以递归形式给出的，所以，可以用递归程序求解。例如，阶乘的递归定义：

$$n!=\begin{cases}1 & n=1 \text{ 或 } n=0\\ n\times(n-1)! & n>1\end{cases} \tag{11.2}$$

有些问题虽然定义本身不具有明显的递归特征，但其求解方法是递归的，如汉诺塔问题就是这类问题的一个典型代表。

3. 递归程序的实例

例 11.2 设计递归程序求 $n!$。

解：可以将求 n!独立为函数 Fac(n)，而 Fac(n)的实现依赖于 Fac(n−1)，显然可以通过递归调用函数 Fac 实现求 n!，程序如下：

```
/*  example11-2.cpp  */
#include <stdio.h>                    //使用库函数 printf 和 scanf
int Fac(int n);                       //函数声明
                                      //空行,以下是主函数
int main( )
{
  int m, n = 4;
  m = Fac(n);                  //函数调用,计算 n 的阶乘并将结果存入变量 m
  printf("%d! = %d\n", n, m);      //输出结果
  return 0;                           //将 0 返回操作系统,表明程序正常结束
}
                        //空行,以下是其他函数定义
int Fac(int n)          //函数定义,形参 n 采用传值方式,返回值类型为 int 型
{
  int fac;
  if (n == 1)           //递归出口,即当 n 等于 1 时结束递归
    fac = 1;
  else
    fac = n * Fac(n-1);    //函数递归调用,返回值与 n 相乘再存入 fac
   return fac;             //结束函数 Fac 的执行并将结果 fac 返回到调用处
}
```

采用图示方法描述函数递归调用的运行轨迹，可以较直观地了解到各调用层次及执行过程。程序 example11-2.cpp 的执行过程如图 11.5 所示。

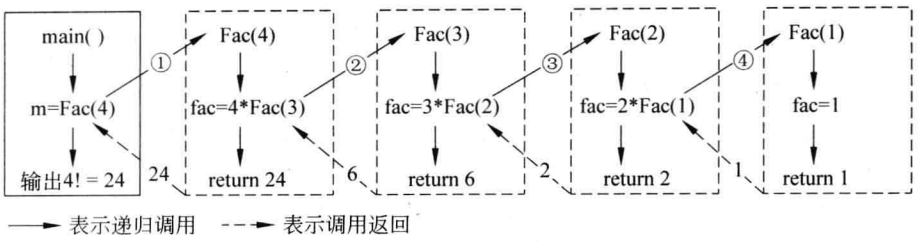


图 11.5　递归函数的执行过程

4. 递归程序的内部执行过程

在计算机内部，一个函数的递归调用类似于多个函数的嵌套调用，只不过调用函数和被调用函数是同一个函数，为便于理解，可以看成是调用同一个函数的复制。递归调用的内部执行过程如下：

① 运行开始时，系统设立工作栈来保存每次调用的运行环境，包括形参、局部变量和返回地址；

② 在递归调用前，将调用函数的形参、局部变量以及调用后的返回地址进栈；

③ 在调用结束后，将栈顶数据出栈，恢复调用前的运行环境，使相应的形参和局部变量恢复为调用前的值，然后从返回地址指定的位置继续执行调用函数。

程序 example11-2.cpp 在递归过程中工作栈的变化情况如图 11.6 所示，变量的值写在括号中，为简单起见，返回地址用该语句所在行号表示。

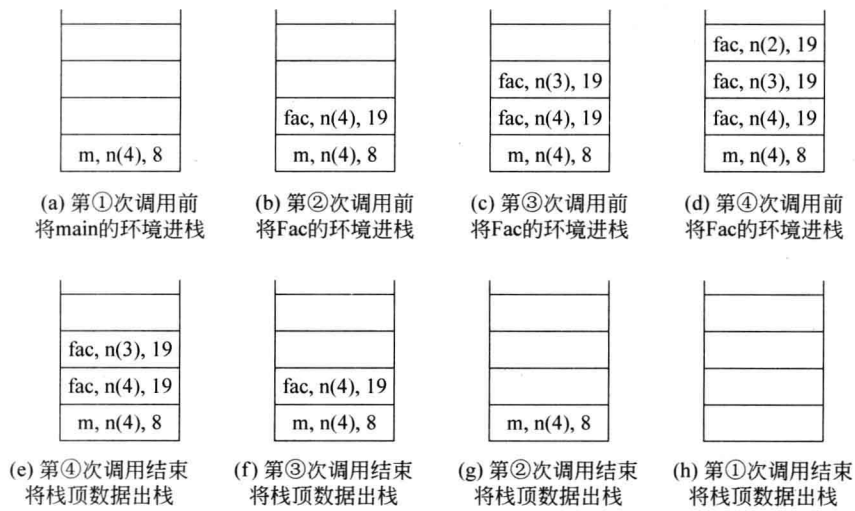


图 11.6　递归过程中工作栈的变化情况

11.2.2 解决任务 11.2 的程序

```
/*   duty11-2.cpp   */
#include <stdio.h>                          //使用库函数 printf 和 scanf
int Fib(int n);                             //函数声明
                                            //空行,以下是主函数
int main( )
{
  int m, n = 5;
  m = Fib(n);              //函数调用,计算 Fibonacci 数列的第 n 项并存入变量 m
  printf("F(%d) = %d\n", n, m);
  return 0;                                 //将 0 返回操作系统,表明程序正常结束
}
                                            //空行,以下是其他函数定义
int Fib(int n)           //函数定义,形参 n 采用传值方式,返回值类型是 int 型
{
  int fib;
  if ((n == 1) || (n == 2)                  //递归出口,当 n 等于 1 或 2 时结束递归
    fib = 1;
  else
    fib = Fib(n - 1) + Fib(n - 2);     //两个函数调用都是递归调用
  return fib;                    //结束函数 Fib 的执行,并将结果 fib 返回到调用处
}
```

当 $n=5$ 时，递归程序的执行过程如图 11.7 所示。

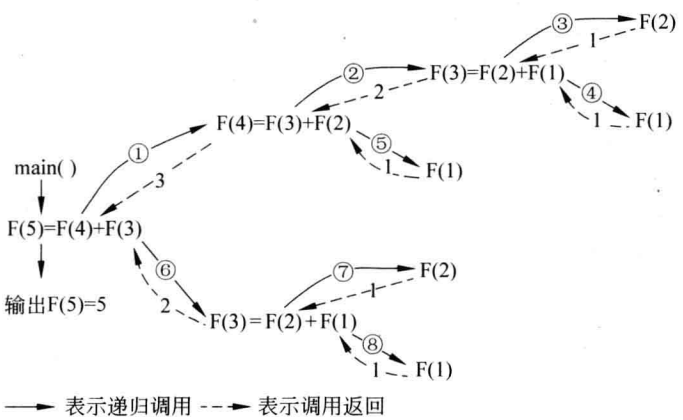


图 11.7　$n=5$ 时斐波那契数的计算过程

11.3 程序设计实例

11.3.1 实例 1——弦截法求方程的根

【问题】 用弦截法求方程 $f(x)=0$ 的根，例如求方程 $x^3-2x^2+x-2=0$ 的根。

【想法】 弦截法的具体方法如下：取两个不同的点 x_1、x_2，使得 $f(x_1)$和 $f(x_2)$的符号相反，则区间(x_1, x_2)内必有一个根，即点$(x_1, f(x_1))$和点$(x_2, f(x_2))$确定的直线交 x 轴于 x，如图 11.8 所示。若 $f(x)$和 $f(x_1)$的符号相同，则根必在区间(x, x_2)内，将 x 作为新的 x_1；否则，根必在区间(x_1, x)内，将 x 作为新的 x_2。重新计算点$(x_1, f(x_1))$和点$(x_2, f(x_2))$确定的直线与 x 轴的交点 x，直到$|f(x)| < \varepsilon$ 为止，ε 为一个很小的数，例如 10^{-6}。

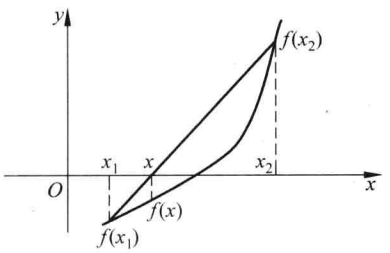


图 11.8 弦截法求解示意图

点$(x_1, f(x_1))$和点$(x_2, f(x_2))$确定的直线与 x 轴的交点 x 可以由下式求出：

$$x=(x_1*f(x_2)-x_2*f(x_1))/(f(x_2)-f(x_1)) \qquad (11.3)$$

【算法】 设函数 Root 实现弦截法，其算法描述如下：

输入：$f(x_1)$和 $f(x_2)$符号相反的两个点 x_1 和 x_2

功能：弦截法求方程的根

输出：方程 $f(x)=0$ 的根 x

伪代码

```
step1: 重复执行下述操作，直到|f(x)| < ε:
    step1.1: 求点(x1, f(x1))和点(x2, f(x2))确定的直线与 x 轴的交点 x;
    step1.2: y = f(x); y1 = f(x1);
    step1.3: 如果 y 和 y1 的符号相同，则 x1 = x; 否则 x2 = x;
step2: 返回 x;
```

【程序】 主函数确定方程根所在的区间[x1, x2]，然后调用函数 Root 用弦截法求方程的根，主函数和 Root 函数中都需要调用函数 F(x)求 f(x)的值。程序如下：

```
1   /*   弦截法.cpp   */
2   #include <stdio.h>                          //使用库函数 printf 和 scanf
3   #include <math.h>                           //使用库函数 fabs
4   #define PRECISION 0.000001                  //符号常量 PRECISION 表示精度
5   double F(double x);                         //函数声明,求 f(x)的值
6   double Root(double x1, double x2);//函数声明,求方程在区间[x1,x2]的根
7                                               //空行,以下是主函数
8   int main( )
9   {
10    double x1, x2, y1, y2, x;
11    do                                        //保证方程在区间[x1, x2]一定有根
12    {
```

```
    printf("请确定方程的根所在区间:");  //输出提示信息
    scanf("%lf %lf", &x1, &x2);    //接收从键盘输入的两个 double 型实数
    y1 = F(x1);                    //函数调用,计算 f(x1)的值
    y2 = F(x2);                    //函数调用,计算 f(x2)的值
  } while (y1 * y2 >= 0);         //当 f(x1)和 f(x2)的符号相同时执行循环
  x = Root(x1, x2);                //函数调用, 求方程的根并存入变量 x
  printf("方程的根为%6.3f\n", x);  //输出结果
  return 0;                        //将 0 返回操作系统,表明程序正常结束
}
                                   //空行, 以下是其他函数定义
double F(double x)       //函数定义,形参 x 采用传值方式,返回 double 型数据
{
  double y;
  y = x * (x * x - 2 * x + 1) - 2;  //减少乘法次数以提高程序效率
  return y;                    //结束函数 F 的执行,并将函数值 y 返回到调用处
}
double Root(double x1, double x2) //函数定义,形参 x1 和 x2 采用传值方式
{
  double x, y, y1;
  do
  {
    x = (x1 * F(x2)- x2 * F(x1))/(F(x2)- F(x1));  //与 x 轴的交点 x
    y = F(x);                      //函数调用,计算 f(x)的值
    y1 = F(x1);                    //函数调用,计算 f(x1)的值
    if (y * y1 > 0)                //f(x)与 f(x1)同号
      x1 = x;                      //求根区间调整为[x, x2]
    else                           //f(x)与 f(x2)同号
     x2 = x;                       //求根区间调整为[x1, x]
  } while (fabs(y) > PRECISION); //当根不满足精度要求时执行循环
  return x;                    //结束函数 Root 的执行,并将根 x 返回到调用处
}
```

运行结果如下（下划线为用户输入）：

```
请确定方程的根所在区间：0  6
方程的根为 2.000
```

11.3.2 实例 2——汉诺塔问题

【问题】 汉诺塔问题来自一个古老的传说：有一座宝塔（塔 A），其上有 64 个金碟，所有碟子按从大到小的次序从塔底堆放至塔顶。紧挨着这座宝塔有另外两个宝塔（塔 B 和塔 C），要求把塔 A 上的碟子移动到塔 C 上去，其间借助于塔 B 的帮助。每次只能移动一个碟子，任何时候都不能把一个碟子放在比它小的碟子上面。

【想法】 当 $n=3$ 时的求解过程如图 11.9 所示，显然，这是一个递归求解的过程。

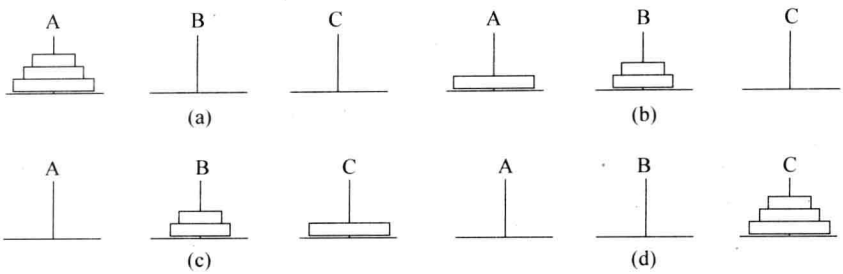


图 11.9　汉诺塔问题求解示意图

【算法】　设函数 Hanio 实现汉诺塔问题，其算法描述如下：

输入：碟子个数 n

功能：移动汉诺塔

输出：无

伪代码

```
step1: 将塔 A 上的 n-1 个碟子借助塔 C 先移到塔 B 上;
step2: 将塔 A 上剩下的一个碟子移到塔 C 上;
step3: 将 n-1 个碟子从塔 B 借助于塔 A 移到塔 C 上;
```

【程序】　主函数接收键盘输入的碟子个数 n，然后调用函数 Hanio 完成具体移动。函数 Move 打印一次移动，程序如下：

```
/*   汉诺塔.cpp   */
#include <stdio.h>                          //使用库函数 printf 和 scanf
void Move(char A, char B);    //函数声明,将塔 A 上的一个碟子移到塔 B 上
void Hanio(int n, char A, char B, char C);    //函数声明,移动 n 个碟子
                                            //空行,以下是主函数
int main( )
{
  int n;
  char A = 'a', B = 'b', C = 'c';//为 A、B、C 赋值,即标识塔 A、B、C
  printf("请输入汉诺塔的阶数: ");     //输出提示信息
  scanf("%d", &n);
  Hanio(n, A, B, C);                        //函数调用,实现 n 阶汉诺塔
  return 0;                                 //将 0 返回操作系统,表明程序正常结束
}
                                            //空行,以下是其他函数定义
void Move(char A, char B)                   //函数定义,形参 A 和 B 采用传值方式
{
  printf("%c-->%c\t", A, B);                //输出移动步骤,变量 A 和 B 写在双引号外
  return;                                   //结束函数 Move 的执行
}
void Hanio(int n, char A, char B, char C)
{          //函数定义，将 n 个碟子由塔 A 借助塔 B 移到塔 C 上,形参采用传值方式
  if (n == 1)                       //递归边界条件,当 n 等于 1 时结束递归
    Move(A, C);                     //函数调用,将塔 A 上的一个碟子移到塔 C 上
  else
```

```
    {
      Hanio(n-1, A, C, B);  //递归调用,将 n-1 个碟子由塔 A 借助塔 C 移到塔 B
      Move(A, C);           //函数调用,将塔 A 上的一个碟子移到塔 C 上
      Hanio(n-1, B, A, C);  //递归调用,将 n-1 个碟子由塔 B 借助塔 A 移到塔 C
    }
    return;                 //结束函数 Hanio 的执行
}
```

运行结果如下（下划线为用户输入）：

```
请输入汉诺塔的阶数：3
a-->c    a-->b    c-->b    c-->a    b-->a    b-->c    a-->c
```

习 题 11

一、选择题

1. C/C++语言规定（　　）。

   A. 函数不能嵌套定义，但可以嵌套调用

   B. 函数不能嵌套定义，也不可以嵌套调用

   C. 函数可以嵌套定义，也可以嵌套调用

   D. 函数可以嵌套定义，但不可以嵌套调用

2. 在函数调用时，系统将当前函数的运行环境进栈，运行环境不包括（　　）。

   A. 形参变量　　B. 局部变量　　C. 全局变量　　D. 返回地址

3. 以下程序运行后的输出结果是（　　）。

```
#include <stdio.h>
int Fun(int x, int y)
{
  return x + y;
}
int main( )
{
  int a = 2, b = 5, c = 8;
  printf("%d\n", Fun(Fun(a + b, c), a - b));
  return 0;
}
```

   A. 编译出错　　B. 9　　C. 12　　D. 21

4. 以下程序运行后的输出结果是（　　）。

```
#include <stdio.h>
#include <string.h>
char str[ ] = "student";
void Fun(int i)
{
```

```
  if (i < strlen(str))
  {
    printf("%c", str[i]);
    Fun(i + 2);
  }
}
int main( )
{
  int i = 0;
  Fun(i);
  return 0;
}
```

A. student　　B. suet　　C. sue　　D. stu

5. 以下程序运行后的输出结果是（　　）。

```
#include <stdio.h>
int Fun(int i)
{
  int sum = 0;
  if (i ==1)
    return 1;
  else
  {
   sum += Fun(i - 1);
   return sum;
  }
}
int main( )
{
  printf("%d\n", Fun(5));
  return 0;
}
```

A. 0　　B. 1　　C. 8　　D. 15

二、程序设计题

1. 求三个数中最大值和最小值之间的差。
2. 计算 1! + 2! + 3! + … + n!的值。
3. 将十六进制整数转换为对应的十进制整数。
4. 编写递归函数，实现将给定的字符串逆序输出。
5. 验证卡布列克运算，该运算是指对任意各位数字不完全相同的四位数 n：

（1）把四位数字从小到大排列，形成由原来四位数字组成的最小的四位数 n_1；

（2）把四位数字从大到小排列，形成由原来四位数字组成的最大的四位数 n_2；

（3）执行 n_2–n_1，得到一个新的四位数 n。

重复上述操作，最后总能得到整数 6174。

6. 编写递归函数，实现在一个有序序列中进行折半查找。折半查找的基本思想是：取有序序列的中间元素作为比较对象，若给定值与中间元素相等，则查找成功；若给定值小于中间元素，则在有序序列的左半区继续查找；若给定值大于中间元素，则在有序序列的右半区继续查找。不断重复上述过程，直到查找成功，或所查找的区域为空，查找失败。

第12章

再谈指针

由于指针可以直接对内存进行操作，所以指针的功能非常强大。正确灵活地使用指针可以有效地表示复杂的数据结构，并可动态分配内存空间，提高程序的运行效率。

12.1 指针与数组

【任务 12.1】判断回文

【问题】 输入一个字符串，判断该字符串是否为回文。回文即首尾对称的字句，例如“abcba”、“abba”均为回文。要求用指针实现。

【想法】 设两个指针变量 p 和 q，其中 p 指向串的首部，q 指向串的尾部，p 从前向后，q 从后向前，依次取出字符进行匹配，直到 p 和 q 相遇，如果匹配不成功，则说明不是回文。

【算法】 设函数 TurnString 实现判断回文，其算法描述如下：

输入：字符串 str

功能：判断回文

输出：如果是回文，则返回 1，否则返回 0

伪代码

```
step1: 指针 p 指向字符串 str 的首部，指针 q 指向字符串 str 的尾部;
step2: 重复执行下述操作，直到 p 和 q 相遇:
    step2.1: 如果 p 所指字符不等于 q 所指字符，则 str 不是回文，返回 0;
    step2.2: p++; q--;
step3: str 是回文，返回 1;
```

由于数组在内存中占用一段连续的存储单元，并且每个数组元素占用的存储单元数相同，因此，对数组元素的访问不仅可以通过下标实现，还可以通过指针实现，而且指针方式的效率更高。

12.1.1 指向一维数组的指针

对一维数组元素的访问可以有以下三种方式。

1. 通过下标访问数组元素

对于一维数组，可以通过下标直接访问数组元素。例如，如下语句通过数组元素的下标实现为每个数组元素赋值：

```
int a[10];
for (int i = 0; i < 10; i++)
  a[i] = i;
```

2. 通过地址访问数组元素

对于一维数组元素 a[i]，“[ ]”实际上是**下标运算符**，即将元素 a[i]的存储地址转换为 a + i，如图 12.1 所示。

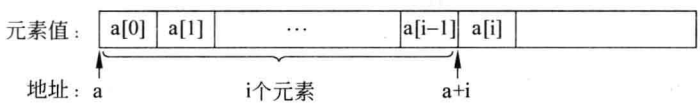


图 12.1　下标运算符的转换示意图

因此，为了提高运行速度，可以直接通过地址访问数组元素。例如，如下语句通过数组元素的存储地址实现为每个数组元素赋值：

```
int a[10];
for (int i = 0; i < 10; i++)
  *(a + i) = i;
```

其中，a + i 表示数组元素 a[i]在内存中的起始地址，*(a + i)用间接引用运算符“*”访问地址为 a + i 的存储单元，相当于 a[i]。

但是，数组名表示数组在内存中的起始地址，是地址常量，在程序中不能改变，操作起来缺少灵活性。

3. 通过指针访问数组元素

从本质上讲，指针就是地址，既然数组名作为数组的起始地址不能改变，考虑定义指针变量指向数组的起始地址，然后通过这个指针对数组进行操作。例如，如下语句定义了指针 p 并将其指向数组 a[10]的起始地址，如图 12.2 所示，则元素 a[i]的地址是 p + i，*(p + i) 访问地址为 p + i 的存储单元，即 a[i]的元素值，可以通过指针 p 实现为每个数组元素赋值。

```
int a[10];
int *p = a;
```

```
for (int i = 0; i < 10; i++)
  *(p + i) = i;
```

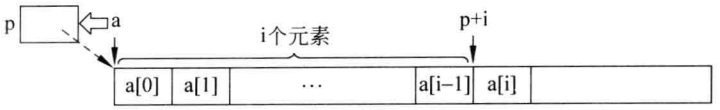


图 12.2　指向一维数组的指针

既然指针作为一个变量可以改变，如下语句同样实现为数组元素赋值，如图 12.3 所示。由于 p++自增运算要比 p + i 算术运算快得多，因此指针方式的执行效率比地址方式的执行效率要高。

```
int a[10];
int *p = a;
for (int i = 0; p < (a + 10); i++, p++)          //第 3 个表达式是逗号表达式
  *p = i;
```

p++运算将指针 p 指向数组的下一个元素。p++不是简单地使指针变量 p 的值增 1，而是编译为 p = p + 1×size，其中 size 表示一个数组元素占用的字节数。假设 int 型数据占用 4 个字节，指针 p 的当前值为 2000，则 p++的值为 2000 + 1×4 = 2004，如图 12.3 所示。

通过指针访问数组元素同样需要注意指针的越界问题，如果指针 p 所指存储单元已经不是数组空间，这时再引用指针 p 可能产生无法预料的结果，如图 12.4 所示。

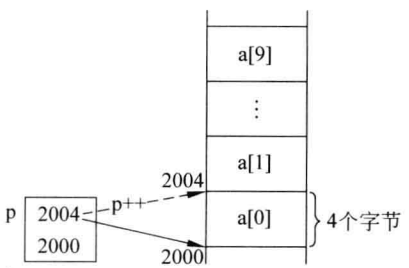


图 12.3　p++操作示意图

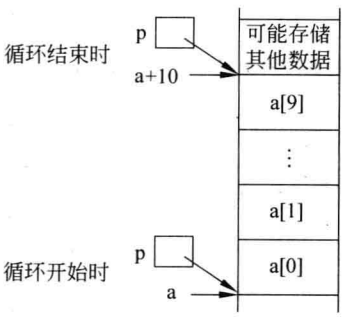


图 12.4　指针越界示意图

例 12.1　在一维数组 r[n]中查找最大值元素。要求用指针实现。

解：求解思想与例 8.1 相同。设指针 p 指向数组 r[n]的起始地址，然后通过指针 p 实现对数组的操作。程序如下：

```
1   /*  example12-1.cpp   */
2   #include <stdio.h>                    //使用库函数 printf 和 scanf
3   #define N 5                           //定义符号常量 N
4                                         //空行，以下是主函数
5   int main( )
6   {
7     int a[N], max;                      //max 存储最大值
8     int *p = a;                         //指针 p 指向数组 a[N]的起始位置
```

```
    printf("请输入%d 个整数: ", N);     //输出提示信息
    for (int i = 0; i < N; i++)       //依次输入每一个数组元素
    {
      scanf("%d", &a[i]);             //从键盘接收一个整数并存入元素 a[i]
    }
    for (max = *p, p = a + 1; p < (a + N); p++) //表达式 1 为逗号表达式
    {
      if (max < *p)                   //将指针 p 所指元素与 max 进行比较
        max = *p;                         //指针 p 所指元素为当前最大值
    }
    printf("最大值为: %d\n", max);          //输出结果
    return 0;                         //将 0 返回操作系统，表明程序正常结束
}
```

12.1.2 指向二维数组的指针

二维数组是一维数组的推广，可以把二维数组的每一行看成是一维数组，其中每个数组元素是一维数组。如图 12.5 所示，数组 a 包含 3 行，即数组 a 有 3 个元素，分别是 a[0]、a[1]和 a[2]，每一行是一维数组。例如，a[0]包含 4 个元素，分别是 a[0][0]、a[0][1]、a[0][2]和 a[0][3]，因此，a[i]实际上是一个行指针，指向二维数组 a 的第 i 行，a[i]相当于一维数组名，是地址常量，本身并不占用实际的存储空间。

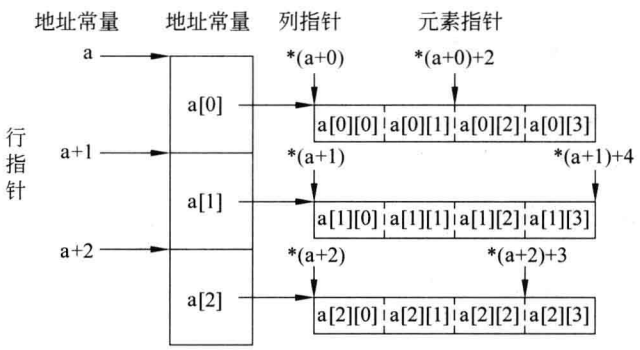


图 12.5 二维数组的行指针和列指针

a 为二维数组 a[3][4]的首地址，也是 a[0]（即第 0 行元素）的首地址，则 a + 1 表示数组 a[1]（即第 1 行元素）的首地址，a + 2 表示数组 a[2]（即第 2 行元素）的首地址。a + i 是行指针，即指向第 i 行元素的首地址，*(a + i)将行指针 a + i 转换为列指针，即指向第 i 行第 0 列元素，即&a[i][0]，则*(a + i) + j 指向第 i 行第 j 列元素，即&a[i][j]。通过指针访问二维数组的不同形式如表 12.1 所示。

二维数组关于地址和指针的概念比较复杂，请仔细对比、深刻理解下面各语句。设指针 p 指向 int 型数据，由于二维数组 a[3][4]的数组名 a 是行指针，表示数组 a[0]的首地址，即 a 的数组元素是一维数组，因此，不能将二维数组名 a 赋给指针 p：

表 12.1　通过指针访问二维数组的不同形式

形　式	性　质	含　义
a	行指针	二维数组名，指向一维数组 a[0]，相当于&a[0][0]
a + i	行指针	指向一维数组 a[i]，相当于&a[i][0]
*a	列指针	即*(a + 0)，指向第 0 行第 0 列，相当于&a[0][0]
*(a + i)	列指针	指向第 i 行第 0 列，相当于&a[i][0]
*(a + i) + j	元素指针	指向第 i 行第 j 列，相当于&a[i][j]

```
int a[3][4];
int *p = a;                          //编译错误，无法实现类型转换
```

但第 0 行（即 a[0]）的数组元素为 int 型，因此，可以将行地址常量 a[0]赋给指针 p：

```
int a[3][4];
int *p = a[0];
```

也可以将元素 a[0][0]的地址赋给指针 p：

```
int a[3][4];
int *p = &a[0][0];
```

由上述二维数组元素存储地址的计算过程可以看出，C/C++语言对二维数组元素的访问可以有以下三种方式。

1. 通过下标访问数组元素

访问二维数组元素可以使用该元素在数组中的行下标和列下标。例如，如下语句通过行下标和列下标实现为二维数组元素赋值：

```
int a[3][4];
for(int i = 0; i < 3; i++)
  for(int j = 0; j < 4; j++)
    a[i][j] = i + j;
```

2. 通过地址访问数组元素

对于二维数组元素 a[i][j]，“[ ]”实际上是**变址运算符**，即将元素 a[i][j]的存储地址转换为 a[i] + j，再转换为*(a + i) + j。因此，为了提高运行速度，可以直接通过地址访问数组元素。例如，如下语句通过数组元素的存储地址实现为二维数组元素赋值：

```
int a[3][4];
for(int i = 0; i < 3; i++)
  for(int j = 0; j < 4; j++)
    *(*(a + i) + j) = i + j;          //*(a + i) + j 为元素 a[i][j]的存储地址
```

3. 通过指针访问数组元素

可以定义指针指向二维数组的起始地址，然后通过这个指针访问数组元素。例如，如下语句通过指针实现为二维数组元素赋值：

```
int a[3][4];
int *p = &a[0][0];
for(int i = 0; i < 3; i++)
  for(int j = 0; j < 4; j++)
    *(p++) = i + j;                                    //相当于*p = i + j; p++;
```

例 12.2 求二维数组 r[m][n]的最大值元素。要求用指针实现。

解：求解思想与例 8.4 相同。设指针 p 指向二维数组 r[m][n]的起始地址，然后通过指针 p 实现对数组的操作。程序如下：

```
1   /*  example12-2.cpp  */
2   #include <stdio.h>                          //使用库函数 printf 和 scanf
3                                               //空行，以下是主函数
4   int main( )
5   {
6     int a[10][10], max, m, n;                 //定义二维数组最多 10 行 10 列
7     int *p = &a[0][0];                        //指针 p 指向元素 a[0][0]
8     printf("请输入二维数组的行数和列数:");    //输出提示信息
9     scanf("%d%d", &m, &n);                    //确定实际的行数 m 和列数 n
10    printf("请输入%d 个整数:", m*n);          //输出提示信息
11    for(int i = 0; i < m; i++)                //依次输入每一个二维数组元素
12      for(int j = 0; j < n; j++)
13          scanf("%d", &a[i][j]);//接收从键盘输入一个整数并存入元素 a[i][j]
14    max = *p;          //指针 p 已指向元素 a[0][0],假定 p 所指元素为最大值元素
15    for (p = &a[0][1]; p < *(a + m - 1) + n; p++)
16    {           //*(a + m - 1) + n - 1 为二维数组 a[m][n]最后一个元素地址
17      if (max < *p)                           //将指针 p 所指元素与 max 进行比较
18        max = *p;                             //max 保存当前最大值
19    }
20    printf("最大值是:%d\n", max);
21    return 0;                          //将 0 返回操作系统,表明程序正常结束
22  }
```

12.1.3 指针数组

数组元素可以是任意合法的数据类型，如果数组元素是指针，则构成了指针数组。

【语法】 定义一维指针数组的一般形式如下：

```
        ←——— 指针定义符
基类型 *数组名[数组长度];
        数组
      数组元素为指针
    指针指向的数据类型
```

可以看出，与定义普通数组不同的是指针定义符“*”，由于指针定义符“*”的优先级低于下标运算符“[]”的优先级，所以，该定义相当于

基类型 *(数组名[数组长度]);

【语义】 定义指针数组，数组元素为指向基类型的指针。

如下语句定义并初始化一维数组 str[3]，元素 str[i]为字符串指针，其存储示意图如图 12.6 所示。

```
char *str[3] = {"Red", "Green", "Blue"};
```

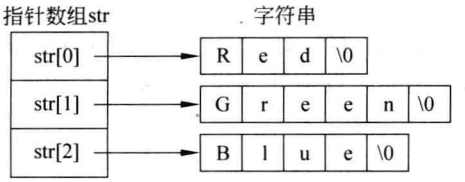


图 12.6 指针数组的存储示意图

例 12.3 求 N 个字符串中值最大的字符串。要求用指针实现。

解： 求解思想与例 8.1 相同，定义指针数组存储 N 个字符串，字符串比较操作调用库函数 strcmp 实现，程序如下：

```
1   /*   example12-3.cpp   */
2   #include <stdio.h>                          //使用库函数 printf 和 scanf
3   #include <string.h>                         //使用库函数 strcmp
4   #define N 5                                 //定义符号常量 N
5                                               //空行,以下是主函数
6   int main( )
7   {
8     char *str[N] = {"Red", "Green", "Blue", "Yellow", "Black"};
9     int index = 0;                            //index 保存最大字符串的下标
10    for (int i = 1; i < N; i++)
11    {
12      if (strcmp(str[i], str[index])>0) //str[i]大于 str[index]
13        index = i;                            //str[i]为当前最大字符串
14    }
15    printf("最大的字符串是%s\n", str[index]);
16    return 0;                                 //将 0 返回操作系统,表明程序正常结束
17  }
```

运行结果如下：

```
最大的字符串是 Yellow
```

12.1.4 解决任务 12.1 的程序

```
/*   duty12-1.cpp   */
#include <stdio.h>                  //使用库函数 printf 和 scanf
#include <string.h>                 //使用库函数 strlen
int TurnString(char str[ ]);        //函数声明,判断字符串 str 是否为回文
                                    //空行,以下是主函数
int main( )
{
  char str[80];
  printf("请输入字符串: ");          //输出提示信息
  scanf("%s", str);
  if (TurnString(str))              //函数调用,将返回值作为逻辑值
    printf("该字符串是回文!\n");
  else
    printf("该字符串不是回文!\n");
  return 0;                         //将 0 返回操作系统,表明程序正常结束
}
                                    //空行,以下是其他函数定义
int TurnString(char str[ ])         //函数定义,一维字符数组作为形参
{
  char *p = str;                    //指针 p 指向字符串的首部
  char *q = str + strlen(str)-1;    //指针 q 指向字符串的尾部
  for (; p < q; p++, q--)           //表达式 1 为空,指针 p 后移,指针 q 前移
  {
    if (*p != *q)                   //对应字符不匹配
      return 0;          //返回 0 表示匹配失败,字符串 str 不是回文
  }
  return 1;              //返回 1 表示全部比较完毕,匹配成功,字符串 str 是回文
}
```

运行结果如下（下划线为用户输入）：

```
请输入字符串：abcba
该字符串是回文!
```

12.2 指针与结构体

【任务 12.2】统计入学成绩（函数版）

【问题】 某重点大学的博士入学考试科目为外语和两门专业课，对于每个考生，输入各科考试成绩，并计算总分。要求用函数实现。

【想法】 输入一个考生的各项信息，再计算总分。

【算法】 设函数 AddMarks 计算一个考生的总分，其算法描述如下：

输入：一个考生的成绩信息
功能：计算总分
输出：该生的总分

伪代码
```
step1: 输入一个考生的各项信息;
step2: sum = 外语成绩 + 专业课 1 成绩 +专业课 2 成绩;
step3: 输出 sum;
```

将结构体变量的值传递给函数，通常采用指针传递方式，即形参是指向结构体类型的指针，实参是结构体变量的地址或指向结构体变量的指针，参数传递是将结构体变量的首地址传递给形参。

12.2.1 指向结构体的指针

指针可以指向任意合法的数据类型，如果指针指向结构体变量，则可以通过指针访问结构体变量的成员。指向结构体变量的指针称为结构体指针。

1. 结构体指针的定义

【语法】 定义结构体指针的一般形式如下：

指针定义符

结构体类型 *指针变量名;
基类型为结构体指针

其中，结构体类型是已经定义或正在定义的结构体类型；“*”是指针定义符，指针变量名是一个合法的标识符。

【语义】 定义指向结构体类型的指针。

定义结构体指针后，需要将该指针与某个结构体变量的地址相关联，例如，如下语句定义并初始化了结构体指针 p，如图 12.7 所示。

```
struct DateType
{
  int year, month, day;
};
struct DateType birthday = {1968, 3, 26};
struct DateType *p = &birthday;   //&birthday 为结构体变量 birthday 的首地址
```

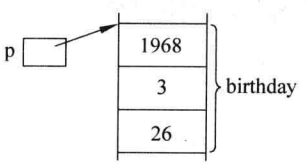


图 12.7 指针 p 指向结构体变量 birthday

2. 通过指针引用结构体成员

【语法】 通过指针引用结构体成员的一般形式如下：

```
(*指针).成员
```

其中，“*指针”一定要用括号括起来，因为成员运算符“.”的优先级高于间接引用运算符“*”的优先级。C/C++语言还提供了运算符“->”（减号后紧跟大于号）引用结构体变量的成员，一般形式如下：

```
指针->成员
```

指针
结构体
结构体的某个成员

【语义】 通过结构体指针引用结构体变量的某个具体成员。

在结构体指针与某个结构体变量的地址相关联后，就可以通过指针引用结构体变量的具体成员。例如，如下语句通过指针 p 输出结构体变量 birthday 的各成员：

```
struct DateType
{
 int year, month, day;
};
struct DateType birthday = {1968, 3, 26};
struct DateType *p = &birthday;
printf("%4d-%2d-%2d\n", p->year, p->month, p->day);
```

结构体类型可以嵌套定义，在引用结构体成员时需要注意使用合适的运算符。例如，如下语句定义了结构体指针 p 指向结构体变量 stu，则 p->name 引用 stu 的成员 name，p->birthday 引用 stu 的成员 birthday，而 p->birthday.day 引用 birthday 的成员 day。

```
struct DateType
{
 int year, month, day;
};
struct StudentType
{
  char name[10];
  struct DateType birthday;
};
struct StudentType stu = {"言言", {1997, 1, 4}};
struct StudentType *p = &stu;
printf("%s 的生日是%d 月%d 日\n", p->name, p->birthday.month, p->birthday.day);
```

12.2.2 结构体指针作为函数参数

将结构体变量传递给函数，可以有以下三种方法：

① 形参是结构体成员，实参是对应结构体成员的值，参数传递是将结构体成员的值传递给形参。

② 形参是结构体变量，实参是结构体变量的值，参数传递是将结构体变量的值传递给形参。

③ 形参是指向结构体类型的指针，实参是结构体变量的地址或指向结构体变量的指针，参数传递是将结构体变量的首地址传递给形参。

前两种方法属于值传递方式，在结构体规模较大时，空间和时间的开销很大，因此一般较少使用。下面介绍第 3 种方法。

例 12.4 将输入的日期按下列格式之一输出：①中文（中国）格式；②中文（中国台湾）格式；③英语（美国）格式；④英语（英国）格式。要求用函数实现。

解：定义结构体类型 struct DateType 表示日期，输入一个日期的年、月、日信息，然后调用函数 FormatDay 输出相应格式的日期。函数 FormatDay 的形参是指向 struct DateType 的指针，实参是 struct DateType 型变量 date 的起始地址。程序如下：

```
/*  example12-4.cpp  */
#include <stdio.h>                          //使用库函数 printf 和 scanf
struct DateType                             //定义结构体类型 struct DateType
{
  int year, month, day;
};
void FormatDay(struct DateType *p);  //函数声明,格式化输出日期
                                            //空行,以下是主函数
int main( )
{
  struct DateType date;                     //定义结构体变量 date
  printf("请输入一个日期,分别对应年、月、日:");  //输出提示信息
  scanf("%d%d%d", &date.year, &date.month, &date.day);
  FormatDay(&date);                 //函数调用,实参是结构体变量 date 的首地址
  return 0;                         //将 0 返回操作系统,表明程序正常结束
}
                                            //空行,以下是其他函数定义
void FormatDay(struct DateType *p)   //函数定义,形参 p 采用传指针方式
{
  int select;
  printf("1. 中文（中国）格式     2. 中文（中国台湾）格式\n");
  printf("3. 英语（美国）格式     4. 英语（英国）格式\n");
  printf("5. 其他格式\n 请输入对应数字: \n");      //输出提示信息
  scanf("%d", &select);
  switch (select)
  {                                 //以下采用结构体指针实现对结构体成员的引用
    case 1:printf("中国格式: %d-%d-%d\n", p->year, p->month, p->day);
```

```
            break;
        case 2:printf("中国台湾格式: %d/%d/%d\n", p->year, p->month, p->day);
            break;
        case 3:printf("美国格式: %2d/%2d/%d\n", p->month, p->day, p->year);
            break;
        case 4:printf("英国格式: %2d/%2d/%d\n", p->day, p->month, p->year);
            break;
        default:printf("其他格式: %d-%d-%d\n", p->year, p->month, p->day);
            break;
    }
    return;                                    //结束函数 FormatDay 的执行
}
```

运行结果与例 10.6 相同。

12.2.3 解决任务 12.2 的程序

```
/*    duty12-2.cpp   */
#include<stdio.h>                              //使用库函数 printf 和 scanf
struct StudentType                             //定义结构体类型
{
  char no[10];                                 //学号 no 是字符串,最多 9 位
  char name[10];                               //姓名 name 是字符串,最多 4 个汉字
  double foreign;
  double spec1;
  double spec2;
};
double AddMarks(struct StudentType *p);  //函数声明,累加某个考生的总分
                                               //空行,以下是主函数
int main( )
{
  struct StudentType stu;          //定义结构体变量 stu 存放考生的成绩信息
  double sum;
  printf("请输入考生考号: ");                  //输出提示信息
  scanf("%s", stu.no);                         //stu.no 是数组名,不用加&
  printf("请输入考生姓名: ");                  //输出提示信息
  scanf("%s", stu.name);                       //stu.name 是数组名,不用加&
  printf("请输入考生外语成绩: ");              //输出提示信息
  scanf("%lf", &stu.foreign);
  printf("请输入专业课 1 成绩: ");             //输出提示信息
  scanf("%lf", &stu.spec1);
  printf("请输入专业课 2 成绩: ");             //输出提示信息
  scanf("%lf", &stu.spec2);
  sum = AddMarks(&stu);          //函数调用,实参是结构体变量 stu 的首地址
  printf("%s 的总分是%5.1f\n", stu.name, sum);
  return 0;                                  //将 0 返回操作系统,表明程序正常结束
}
```

```
31                                                    //空行,以下是其他函数定义
32    double AddMarks(struct StudentType *p)//函数定义,形参 p 采用传指针方式
33    {
34      double sum;
35      sum = p->foreign + p->spec1 + p->spec2;
36      return sum;          //结束函数 AddMarks 的执行,并将总分 sum 返回到调用处
37    }
```

运行结果与任务 10.2 相同。

12.3 动态存储分配

在程序中定义变量后，编译程序就会给这个变量按照数据类型分配相应的内存空间，这种内存分配方式称为静态存储分配。**静态存储分配**是在编译时为变量分配内存空间，并且一经分配就始终占有固定的存储单元，直到该变量退出其作用域。**动态存储分配**是在程序运行期间根据实际需要随时申请内存，并在不需要时释放。

【任务 12.3】进制转换

【问题】 将十进制整数转换为任意 *r* 进制整数。

【想法】 将十进制整数转换为 *r* 进制整数的规则是：**除基取余，逆序排列**，即将十进制整数逐次除以 *r* 进制的基数 *r*，直到商为 0，然后将得到的余数逆序排列，先得到的余数为低位，后得到的余数为高位。

【算法】 设函数 Transform 实现进制转换，其算法描述如下：

输入：十进制整数 A，转换的进制数 *r*

功能：将十进制整数转换为 *r* 进制整数

输出：转换后的 *r* 进制整数

伪代码

```
step1: 重复下述操作直到 A 为 0:
    step1.1: B = A % r;
    step1.2: 保存 B;
    step1.3: A = A / r;
step2: 逆序输出得到的 B 值;
```

如果用数组存储转换后的 *r* 进制整数，由于不能确定转换后 *r* 进制整数的位数，无法确定数组长度，因此，考虑用动态存储分配，在程序的运行过程中，根据 *r* 进制整数的实际位数分配内存空间。

12.3.1 申请和释放存储空间

在C/C++语言中，动态存储分配是通过指针实现的，通过调用malloc、calloc、realloc和free等库函数实现内存的分配和释放。在不同的编程环境中，malloc等函数的原型放在不同的头文件中，使用时注意查阅手册。在VC++中，malloc等函数的原型在头文件malloc.h中。本节主要介绍malloc函数和free函数，calloc函数和realloc函数请读者查阅相关资料。

1. 通用指针

如果在定义指针变量时不能确定该指针将指向何种类型的数据，就需要定义通用指针（可以指向任何类型的指针），使用时再根据需要进行强制类型转换。

【语法】 定义通用指针的一般形式如下：

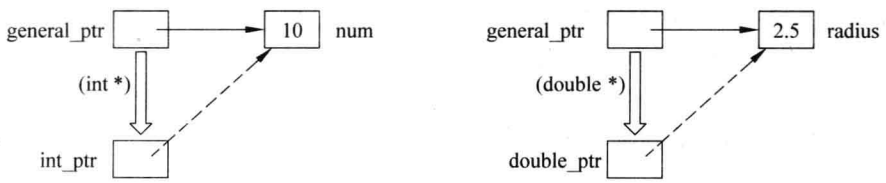
```
        ←——— 指针定义符
void *指针变量名;
通用指针
```

其中，void *表示通用指针，指针变量名是一个合法的标识符。

【语义】 定义通用指针。

如下语句定义了通用指针general_ptr、指向int型数据的指针int_ptr和指向double型数据的指针double_ptr，通用指针general_ptr可以指向任何类型的数据，例如可以指向int型变量num，也可以指向double型变量radius，其操作示意图如图12.8所示。

```
void *general_ptr = NULL;          //定义general_ptr为通用指针并初始化为空
int *int_ptr, num = 10;
double *double_ptr, radius = 2.5;
general_ptr = &num;                 //general_ptr可以指向int型变量
int_ptr = (int *) general_ptr;      //将指针general_ptr进行强制类型转换
general_ptr = &radius;              //general_ptr可以指向double型变量
double_ptr = (double *) general_ptr; //将指针general_ptr进行强制类型转换
```



(a) general_ptr指向int型变量　　(b) general_ptr指向double型变量

图12.8　通用指针操作示意图

2. 申请存储空间

C/C++语言提供了malloc函数实现申请存储空间。

【函数原型】 malloc 函数的原型如下：

```
void *malloc(unsigned int size)
```

通用指针 ← 以字节为单位

其中，void *表示通用指针；size 表示申请分配内存的大小（以字节为单位）。

【功能】 申请分配 size 字节的内存单元，但不清空该内存单元。

【返回值】 如果分配成功，则返回这段内存空间的起始地址，否则返回 NULL。由于 malloc 函数的返回值类型是通用指针，所以，实际使用时需要进行强制类型转换。

一般在调用 malloc 函数后紧跟一条 if 语句判断内存分配是否成功。例如，如下语句申请一个 int 型存储空间，操作示意图如图 12.9 所示。

```
int *p = NULL;
p = (int *)malloc(sizeof(int));      //将指针 p 与申请存储空间的起始地址相关联
if (p == NULL)
   printf("申请空间操作失败");
```

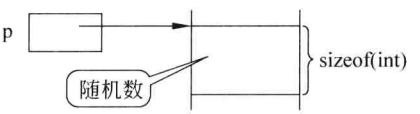


图 12.9 malloc 函数的操作示意图

3. 释放存储空间

在程序的执行过程中，如果不断地动态申请内存空间，使用完却不及时释放，必然会造成可用内存空间减少，导致“内存泄露”。内存是计算机最宝贵的资源，可用内存减少会影响程序的正常运行，因此，动态申请的存储空间要及时释放。C/C++语言提供了 free 函数实现释放存储空间。

【函数原型】 free 函数的原型如下：

```
void free(void *block)
```

通用指针 ← 指针变量

其中，block 是内存空间的某个起始地址，通常为指向该起始地址的指针。

【功能】 释放起始地址是 block 的内存空间。

需要强调的是，free 函数只释放动态申请的存储空间，如果 block 为指针，则该指针仍然存在，即该指针占用的存储单元仍然存在，释放的是该指针指向的内存空间。例如，如下语句执行 free 函数后的存储示意图如图 12.10 所示。

```
int *p = NULL;
p = (int *)malloc(sizeof(int));      //将指针 p 与申请存储空间的起始地址相关联
*p = 10;
free(p);
```

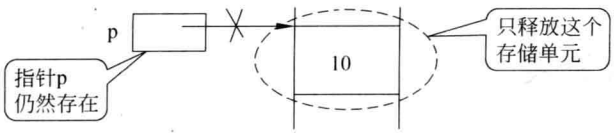


图 12.10　free(p)的操作示意图

例 12.5　求任意 N 个字符串中值最大的字符串。要求使用动态存储分配。

解：由于程序需要在运行过程中接收从键盘输入的字符串，考虑采用动态存储分配，每次读入字符串之前，先申请一块内存单元，然后将读入的字符串存储在这段内存单元中。程序如下：

```
1   /*   example12-5.cpp   */
2   #include <stdio.h>                          //使用库函数 printf 和 gets
3   #include <string.h>                         //使用库函数 strcmp
4   #include <malloc.h>                         //使用库函数 malloc 和 free
5   #define N 5                                 //定义符号常量 N
6                                               //空行,以下是主函数
7   int main( )
8   {
9     char *str[N];
10    int index, i;                             //index 保存最大字符串的下标
11    printf("请输入%d 个字符串,每个字符串以回车结束: \n", N);
12    for (i = 0; i < N; i++)
13    {
14      str[i] = (char *)malloc(80); //申请内存单元,首地址赋给字符串指针 str[i]
15      gets(str[i]);       //接收从键盘输入的字符串并存入 str[i]所指内存单元
16    }
17    index = 0;                                //假定 str[0]为最大字符串
18    for (i = 1; i < N; i++)
19    {
20      if (strcmp(str[i], str[index]) > 0)   //str[i]大于 str[index]
21        index = i;                            //str[i]为当前最大字符串
22    }
23    printf("最大的字符串是%s\n", str[index]);//输出结果
24    for (i = 0; i < N; i++)                   //释放每个字符串占用的存储空间
25      free(str[i]);
26    return 0;                             //将 0 返回操作系统,表明程序正常结束
27  }
```

运行结果如下（下划线为用户输入）：

```
请输入 5 个字符串，每个字符串以回车结束：
Red
Green
Blue
Yellow
Black
最大的字符串是 Yellow
```

12.3.2 指针和链表

数组可以存储批量数据，但数组属于静态存储分配，在定义数组时必须确定数组长度（即元素个数）。有些问题在处理之前无法确定元素个数，定义数组时就需要估算数组长度。如果估算的元素个数过多，则会浪费存储空间；如果估算的元素个数过少，则会不够用。链表可以解决这个问题，链表属于动态存储分配，可以在程序的运行过程中，根据需要申请存储空间。

1. 单链表的存储方式

链表在内存中占用一组任意的存储单元，每个存储单元在存储数据的同时，还必须存储其后继数据（即下一个数据）所在的地址信息，这个地址信息称为**指针**，这两部分组成了数据的存储映象，称为**结点**，如图 12.11 所示。

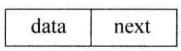


图 12.11　单链表的结点

其中，data 为数据域，用来存放数据值；next 为指针域（亦称链域），用来存放该结点的后继结点的地址。

链表正是通过每个结点的指针域将数据按其逻辑次序链接在一起，由于每个结点只有一个指针域，故也称为**单链表**。通常用结构体类型来描述单链表的结点，其一般形式如下：

```
struct Node
{
 数据域定义 ← 取决于具体问题
 struct Node *next;
}; ← 结尾有分号
```

其中，Node 是正在定义的结构体类型，这是 C/C++语言唯一允许的尚未定义类型就可以使用的情况。

显然，单链表中每个结点的存储地址存放在其前驱结点（即前一个结点）的 next 域中，而第一个结点无前驱，所以设**头指针**指向第一个结点（称为开始结点）；同时，由于最后一个结点无后继结点，故最后一个结点（称为终端结点）的指针域为空，即 NULL（图示中用“∧”表示），也称**尾标志**。为了方便运算，通常在单链表的开始结点之前附设一个类型相同的结点，称为**头结点**。含有 4 个数据 a_1、a_2、a_3 和 a_4 的单链表如图 12.12 所示。

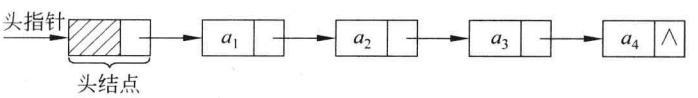


图 12.12　单链表的存储示意图

2. 单链表的查找操作

在单链表中查找第 i 个数据，需要从头指针出发，设置一个工作指针 p，沿着 next 域逐个结点往下搜索。当 p 指向某结点时判断是否为第 i 个结点，若是，则查找成功；否则，将工作指针 p 后移，即将 p 指向原来所指结点的后继结点。对每个结点依次执行上述操作，直到 p 为 NULL 时查找失败，查找过程如图 12.13 所示。

图 12.13　单链表的查找过程示意图

设函数 Search 实现单链表的查找操作，其算法描述如下：

输入：头指针 first，待查找数据的序号 i

功能：在单链表中查找第 i 个数据

输出：如果查找成功，则返回第 i 个结点的数据值；否则返回 0

```
step1: 初始化工作指针 p = first;
step2: 初始化计数器 count = 0;
step3: 当 p 不为空且 count < i 时，重复执行下述操作：
    step3.1: 执行 p = p->next，将工作指针 p 后移指向下一个结点；
    step3.2: count++;
step4: 若 p 为空，则不存在第 i 个数据；否则查找成功，返回结点 p 的数据；
```

需要强调的是，工作指针 p 后移不能写作 p++，而要写作 p = p->next，因为单链表中的结点在内存中不是顺序存储的，则 p++（相当于 p + 1 * sizeof(Node)）后指针 p 不一定指向原结点 p 的后继结点，如图 12.14 所示。

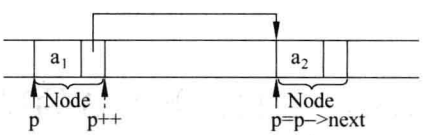


图 12.14　p++和 p = p->next 的操作示意图

3. 单链表的插入操作

假设在单链表中指针 p 所指结点（简称结点 p）的后面插入一个值为 x 的新结点，插入过程如图 12.15 所示，执行的操作步骤如下：

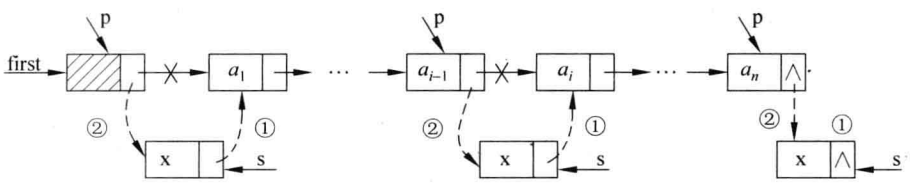


图 12.15　在结点 p 的后面插入结点 s 时指针的变化情况

伪代码

```
step1: s = 申请一个新结点;
step2: s->data = x;
step3: 将结点 s 插入结点 p 的后面;
```

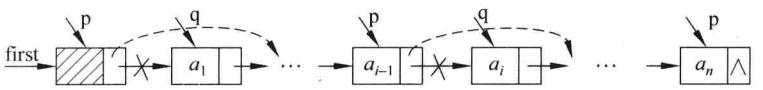
将结点 s 插入结点 p 的后面需要修改两个指针：①将结点 s 的 next 域指向结点 p 的后继结点：s->next = p->next; ②将结点 p 的 next 域指向结点 s：p->next = s。注意修改指针的顺序，如果先将 p->next 指向 s，将无法找到结点 p 的后继结点。由于单链表带头结点，在表头、表中间和表尾插入这三种情况下的操作语句是一致的。

4. 单链表的删除操作

假设在单链表中删除结点 p 的后继结点，删除过程如图 12.16 所示，执行的操作步骤如下：

伪代码

```
step1: 如果结点 p 不存在后继结点,则无法执行删除操作,算法结束;
step2: 否则执行下述操作:
    step2.1: 执行 q = p->next,将指针 q 指向被删结点;
    step2.2: 执行 p->next = q->next,将结点 q 摘链;
    step2.3: 释放结点 q;
```

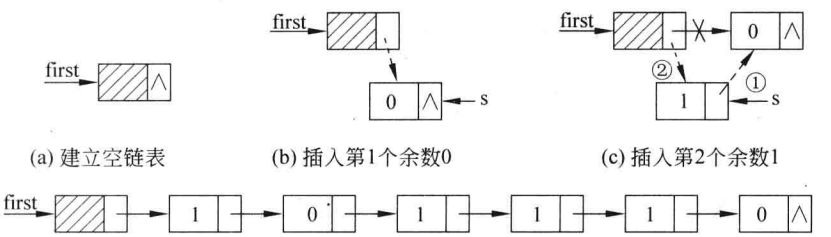


(a) 在表头删除　　(b) 在表中间删除　　(c) 表尾无法删除

图 12.16　删除结点 p 的后继结点时指针的变化情况

12.3.3　解决任务 12.3 的程序

用单链表保存得到的 r 进制整数，首先建立一个空的单链表，然后将得到的余数依次插入单链表中头结点的后面，达到逆序排列的效果。假设将十进制整数 46 转换为二进制整数，转换的结果为 101110，操作示意图如图 12.17 所示。



(a) 建立空链表　　(b) 插入第1个余数0　　(c) 插入第2个余数1

(d) 转换后的结果单链表

①：s->next = first->next; ②：first ->next = s;

图 12.17　单链表保存余数的操作示意图

```
/*   进制转换.cpp   */
#include <stdio.h>                  //使用库函数 printf 和 scanf
#include <malloc.h>                 //使用 malloc 等库函数实现动态存储分配
struct Node                         //定义单链表的结点 Node
{
  int data;
  struct Node *next;
};
struct Node *Transform(int A, int r);
                                    //函数声明,形参 A 和 r 采用传值方式
void PrintLink(struct Node *first); //函数声明,依次输出单链表中各数据
                                    //空行,以下是主函数
int main( )
{
  int A, r;
  struct Node *first;               //定义指向 Node 结点的头指针 first
  printf("请输入一个十进制整数: ");  //输出提示信息
  scanf("%d", &A);
  printf("请输入转换进制的基数: ");  //输出提示信息
  scanf("%d", &r);
  first = Transform(A, r);          //函数调用,返回值赋给头指针 first
  printf("十进制整数%d 转换为%d 进制整数", A, r);
  PrintLink(first);                 //函数调用,实参是头指针
  return 0;                         //将 0 返回操作系统,表明程序正常结束
}
                                    //空行,以下是其他函数定义
struct Node *Transform(int A, int r)
{                                   //函数定义,将十进制整数 A 转换为 r 进制整数
  struct Node *first, *s;
  int B;                            //保存转换后的每一位数字
  first = (struct Node *)malloc(sizeof(struct Node)); //申请头结点
  first->next = NULL;               //first 结点的指针域为空
  while (A != 0)                    //当 A 不等于 0 时执行循环,执行转换
  {
    B = A % r;                      //除基取余,保存余数
    A = A / r;                      //保存整数部分,准备继续转换
    s = (struct Node *)malloc(sizeof(struct Node));
                                    //申请结点, s 指向该结点
    s->data = B;                    //结点 s 的数据域保存转换后得到的一位 r 进制数
    s->next = first->next; //以下两条语句将结点 s 插入到 first 结点的后面
    first->next = s;
  }
  return first; //结束函数 Transform 的执行,并将头指针 first 返回到调用处
}
void PrintLink(struct Node *first)//函数定义,形参 first 采用传指针方式
{                                   //first 为单链表的头指针,且单链表带头结点
  struct Node *p = first->next;//设工作指针 p 并初始化指向第一个数据结点
  if (first->next == NULL)    //如果单链表为空
  {
    printf("0\n");                  //十进制数 A 是 0,转换为 r 进制数一定是 0
```

```
49          return;                          //打印结束,结束函数 PrintLink 的执行
50        }
51        while (p != NULL)                  //当工作指针 p 尚未移出单链表
52        {
53          printf("%d", p->data);           //打印结点 p 的数据域
54          p = p->next;                     //工作指针后移
55        }
56        printf("\n");                      //打印换行符
57        return;                            //打印结束,结束函数 PrintLink 的执行
58    }
```

运行结果如下（下划线为用户输入）：

```
请输入一个十进制整数：46
请输入转换进制的基数：2
十进制整数 46 转换为 2 进制整数 101110
```

12.4 程序设计实例

12.4.1 实例 1——发纸牌

【问题】 假设纸牌的花色有梅花、方块、红桃和黑桃，纸牌的点数有 2、3、4、5、6、7、8、9、10、J、Q、K、A，要求根据用户输入的纸牌张数 *n*，随机发 *n* 张纸牌。

【想法】 为避免重复发牌，设二维数组 sign[4][13]记载是否发过某张牌，其中行下标表示花色，列下标表示点数，数组元素均初始化为 0。设字符串指针数组 card[n]存储随机发的 *n* 张纸牌，例如 card[0] = "梅花 2"。按以下方法依次发每一张牌：首先产生一个 0～3 的随机数 i 表示花色，再产生一个 0～12 的随机数 j 表示点数，如果这张牌尚未发出，则将 sign[i][j]置 1，并将这张牌存储到数组 card[n]中。

【算法】 设函数 SendCards 实现发纸牌，其算法描述如下：

输入：牌数 n

功能：随机发纸牌

输出：存储 n 张牌的数组 card[n]

伪代码

```
step1: 循环变量 k 为 0~n-1,重复执行下述操作:
     step1.1: i = 0~3 的随机数;
     step1.2: j = 0~12 的随机数;
     step1.3: 如果 sign[i][j]等于 1,则结束本次循环,转 step1.1 重新生成第 k 张牌;
              否则,执行下述操作:
          step1.3.1: sign[i][j] = 1;
          step1.3.2: 将第 k 张牌存储到数组 card[k]中;
          step1.3.3: k++;
step2: 返回数组 card[n];
```

【程序】 设字符串指针数组 str1[4]和 str2[13]分别存储一副纸牌的花色和点数，字符串指针数组 card[13]存储随机产生的 n 张纸牌，为避免在函数之间传递大量参数，将数组 str1[4]、str2[13]和 card[13]设为全局变量。程序如下：

```
/*   发纸牌.cpp   */
#include <stdio.h>          //使用库函数 printf、scanf 和符号常量 NULL
#include <stdlib.h>          //使用库函数 srand 和 rand
#include <time.h>            //使用库函数 time
#include <string.h>          //使用库函数 strcpy 和 strcat
#include <malloc.h>          //使用 malloc 等库函数实现动态存储分配
char *str1[4] = {"梅花","黑桃","红桃","方块"};   //定义全局变量存储花色
char *str2[13] = {"2","3","4","5","6","7","8","9","10","J","Q","K","A"};
char *card[13];          //定义全局变量存储随机产生的纸牌,最多发 13 张牌
void SendCards(int n);       //函数声明,随机产生并存储 n 张牌
void Printcards(int n);      //函数声明,输出产生的 n 张牌
                             //空行,以下是主函数
int main( )
{
  int n;
  printf("请输入发牌张数: ");   //输出提示信息
  scanf("%d", &n);
  SendCards(n);         //函数调用,随机产生的 n 张牌存储到指针数组 card 中
  Printcards(n);             //函数调用,输出指针数组 card 中保存的 n 张牌
  return 0;                  //将 0 返回操作系统,表明程序正常结束
}
                             //空行,以下是其他函数定义
void SendCards(int n)     //函数定义,只有一个形参 n,全局变量避免参数传递
{
  int sign[4][13] = {0};     //初始化标志数组，所有牌均未发出
  int k, i, j;
  srand(time(NULL));         //初始化随机种子为当前系统时间
  for (k = 0; k < n; )       //省略表达式 3，发第 k 张牌
  {
    i = rand( ) % 4;         //随机生成花色的编号
    j = rand( ) % 13;        //随机生成点数的编号
    if (sign[i][j] == 1)     //这张牌已发出
      continue;              //跳过循环体余下语句，注意 k 的值不变
    else
    {
      card[k] = (char *)malloc(6);  //存储一张牌需要 6 个字节
      strcpy(card[k], str1[i]); //字符串赋值，相当于 card[k] = str1[i]
      strcat(card[k], str2[j]);//字符串连接,相当于card[k] = card[k] + str2[j]
      sign[i][j] = 1;        //标识这张牌已发出
      k++;                   //准备发下一张牌
    }
```

```
    }
    return;                          //结束函数 SendCards 的执行
}
void Printcards(int n)      //函数定义,只有一个形参 n,全局变量避免参数传递
{
    for (int k = 0; k < n; k++) //依次输出每一张牌
      printf("%-10s", card[k]); //宽度 10 位左对齐输出第 k 张牌
    printf("\n");               //输出换行符
    return;                     //结束函数 Printcards 的执行
}
```

运行结果如下（下划线为用户输入）：

```
请输入发牌张数：4
黑桃 A      梅花 3      方块 6      梅花 J
```

12.4.2 实例 2——约瑟夫环问题

【问题】 约瑟夫环（Josephus）问题是由古罗马的史学家约瑟夫提出的，他参加并记录了公元 66～70 年犹太人反抗罗马的起义。约瑟夫作为一个将军，设法守住了裘达伯特城达 47 天之久。在城市沦陷之后，他和 40 名坚强的将士在附近的一个洞穴中避难。在那里，这些叛乱者坚持“要投降毋宁死”。于是，约瑟夫建议每个人轮流杀死另一个人，而这个顺序是由抽签决定的，约瑟夫有预谋地抓到了最后一签，活了下来并投降了罗马。请设计程序实现约瑟夫环问题。

【想法】 将约瑟夫环问题抽象为如图 12.18 所示数学模型，具体描述为：设 $n(n>0)$ 个人围成一个环，n 个人的编号分别为 1，2，…，n，从第 1 个人开始报数，报到 m 时停止报数，报 m 的人出环，再从他的下一个人起重新报数，报到 m 时停止报数，报 m 的出环，如此下去，直到所有人全部出环为止。当任意给定 n 和 m 后，求 n 个人出环的次序。

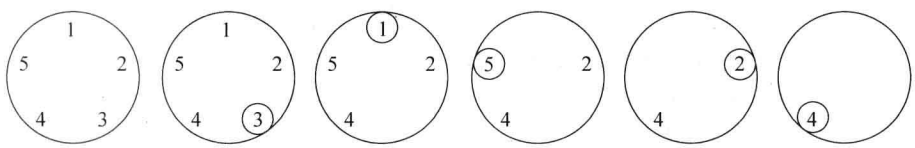


图 12.18 约瑟夫环问题的数学模型（$n=5, m=3$ 时的出圈次序为 3, 1, 5, 2, 4）

【算法】 由于约瑟夫环问题本身具有循环性质，考虑采用循环链表，即终端结点的 next 域指向第 1 个结点，如图 12.19 所示。为了统一对表中任意结点的操作，循环链表不带头结点。

设函数 Creat 构造如图 12.19 所示的约瑟夫环，其算法描述如下：

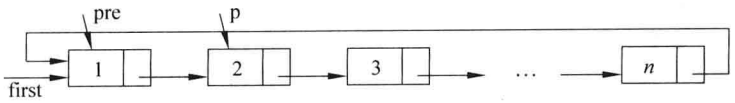


图 12.19　约瑟夫环的初始状态

输入：整数 n

功能：构造约瑟夫环

输出：含有 n 个结点的循环单链表

伪代码

```
step1: 执行下述操作,生成含有一个结点的循环单链表:
    step1.1: 头指针 first = 申请一个新结点;
    step1.2: first->data = 1;
    step1.3: first->next = first;
step2: 循环变量 i 为 2 ~ n,重复执行下述操作:
    step2.1: s = 申请一个新结点;
    step2.2: s->data = i;
    step2.3: 将结点 s 插入终端结点的后面;
    step2.4: i++;
step3: 返回头指针 first;
```

设函数 Joseph 求解约瑟夫环问题，为便于删除操作，设指针 pre 指向结点 p 的前驱结点，初始时计数器 count 从 2 开始计数，如图 12.19 所示。删除结点后 count 从 1 开始计数，具体算法描述如下：

输入：表示约瑟夫环的单链表 first，密码 m

功能：求解约瑟夫环问题

输出：约瑟夫环的出环次序

伪代码

```
step1: 执行下述初始化操作:
    step1.1: pre = first; p = first->next;
    step1.2: count = 2;
step2: 重复下述操作,直到链表中剩下一个结点;
    step2.1: 如果 count 小于 m,则
        step2.1.1: 工作指针 pre 和 p 后移;
        step2.1.2: count++;
    step2.2: 否则,执行下述操作:
        step2.2.1: 输出结点 p 的数据域;
        step2.2.2: 删除结点 p;
        step2.2.3: p 指向 pre 的后继结点;count = 1 重新开始计数;
step3: 链表中只剩下结点 p,输出结点 p 的数据域,删除结点 p;
```

【程序】　在函数 Creat 中，为了将结点 s 插入终端结点的后面，设尾指针 r 指向终端结点。程序如下：

```
/*   约瑟夫环.cpp   */
#include <stdio.h>                    //使用库函数 printf 和 scanf
#include <malloc.h>                   //使用 malloc 等库函数实现动态存储分配
struct Node                           //定义单链表的结点 Node
{
  int data;
  struct Node *next;
};
struct Node *Creat(int n);            //函数声明,构造约瑟夫环
void Joseph(struct Node *first, int m);  //函数声明,打印出环的次序
                                      //空行,以下是主函数
int main( )
{
  int n, m;
  struct Node *head = NULL;           //定义头指针 head 并初始化为空
  printf("请输入约瑟夫环的长度:");  //输出提示信息
  scanf("%d", &n);
  printf("请输入密码: ");            //输出提示信息
  scanf("%d", &m);
  head = Creat(n);                    //函数调用,返回的头指针赋给 head
  Joseph(head, m);                    //函数调用,实参 head 是头指针
  return 0;                           //将 0 返回操作系统,表明程序正常结束
}
                                      //空行,以下是其他函数定义
struct Node *Creat(int n)             //函数定义,返回值是指向 Node 结点的指针
{
  struct Node *first = NULL, *s, *r; //定义指针 first、s 和 r
  int i;
  first = (struct Node *)malloc(sizeof(struct Node));
  first->data = 1;                    //结点 first 的数据域为 1
  first->next = first;    //将结点 first 的指针域指向该结点,构造循环链表
  r = first;          //指针 r 指向单链表的终端结点,目前单链表只有一个结点
  for (i = 2; i <= n; i++)    //依次插入数据域为 2、3、4、…、n 的结点
  {
    s = (struct Node *)malloc(sizeof(Node));//申请结点, s 指向该结点
    s->data = i;                      //结点 s 的数据域为 i
    r->next = s;                      //以下两条语句将结点 s 插入结点 r 的后面
    s->next = first;
    r = s;                            //指针 r 指向当前的终端结点,即新插入的结点
  }
  return first;       //结束函数 Creat 的执行,并将头指针 first 返回到调用处
}
void Joseph(struct Node *first, int m)
{                     //函数定义,形参 first 为循环链表的头指针,形参 m 为密码
  struct Node *pre = first, *p = first->next, *q;  //初始化工作指针
  int count = 2;    //由于 p 指向第 2 个结点,因此,计数器 count 从 2 开始计数
  printf("出环的顺序是:");
  while (p->next != p)                //循环直到循环链表中只剩下一个结点
  {
```

```
50        if (count < m)                    //计数器未累加到密码值
51        {
52          pre = p;                        //将工作指针 pre 后移
53          p = p->next;                    //将工作指针 p 后移
54          count++;                        //计数器加 1
55        }
56        else                              //计数器已经累加到密码值
57        {
58          printf("%-3d", p->data);//宽度 3 位左对齐输出出环的编号
59          q = p;                          //指针 q 暂存即将删除的结点
60          pre->next = p->next;            //将结点 p 摘链
61          p = pre->next;                  //工作指针 p 后移,但 pre 不动
62          free(q);                        //释放指针 q 指向的存储单元
63          count = 1;                      //计数器从 1 开始重新计数
64        }
65      }
66      printf("%-3d\n", p->data); //宽度 3 位左对齐输出剩下一个结点的数据域
67      free(p);                            //释放最后一个结点
68      return;                             //结束函数 Joseph 的执行
69    }
```

运行结果如下（下划线为用户输入）：

```
请输入约瑟夫环的长度：6
请输入密码：2
出环的顺序是：2  4  6  3  1  5
```

习 题 12

一、选择题

1. 若有变量定义 int a[5] = {1, 2, 3, 4, 5}; int *p = a; 则下列（　　）不能实现引用数组第 2 个元素。

A. a[1]　　B. p[1]　　C. *p + 1　　D. *(p + 1)

2. 有一个二维数组 a[3][4]，其第 3 行第 4 列元素的正确表示方法是（　　）。

A. a[3][4]　　B. &a[2][3]　　C. *(a + 2) + 3　　D. *(a[2] + 3)

3. 以下程序求数组中的最大值，划线处的语句是（　　）。

```
#include <stdio.h>
int main( )
{
  int a[5] = {3, 5, 1, 8, 6};
  for(int *p = a, *q = a; p < a + 5; p++)
    if (_____)
      q = p;
  printf("%d\n", *q);
  return 0;
```

```
}
```

A. p > q　　B. *p > *q　　C. a[p] > a[q]　　D. p–a > q–a

4. 以下程序的输出结果是（　　）。

```
#include <stdio.h>
struct HAR
{
  int x, y;
  struct HAR *p;
} h[2];
int main( )
{
  h[0].x = 1; h[0].y = 2; h[0].p = &h[1];
  h[1].x = 3; h[1].y = 4; h[1].p = h;
  printf("%d, %d", (h[0].p)->x, (h[1].p)->y);
  return 0;
}
```

A. 1, 2　　B. 3, 2　　C. 2, 3　　D. 1, 4

5. 若有变量定义 int x[5] = {1, 2, 3}, *p = x; 则能正确表示数组元素地址的是（　　）。

A. x++　　B. &p　　C. &p[2]　　D. p + 3

6. 若有以下函数定义，p 是该函数的形参，要求通过 p 把动态申请的存储单元的地址传回主调函数，则正确的形参定义是（　　）。

```
void Fun(_________)
{
  *p = (double *)malloc(sizeof(double) * 10);
  ⋮
}
```

A. double *p　　B. double p　　C. double **p　　D. double &p

二、程序设计题

1. 解密藏头诗。输入一首藏头诗，将每一句的第一个字提取出来并输出。

2. 中文输入法对于每个发音相同的汉字都有一个列表，可以根据使用的频率调整出现的顺序。例如，设输入法“de”对应五个字“的”、“得”、“地”、“德”、“低”，按顺序设定其初始频率。每次输入一个字，将这个字的频率加 1，再按其频率大小降序输出。

3. 整理姓名表。将输入的若干姓名按字典顺序排序并输出。

4. 使用指针完成两个字符串的比较。

5. 用单链表实现将一个二进制数加 1 的运算。

6. 在某商店的仓库管理系统中，对电视机按其价格从低到高建立一个单链表，链表的每个结点指出同样价格的电视机的台数。现有 *m* 台价格为 *n* 元的电视机入库，请设计程序完成仓库的进货管理。

第13章

再谈输入输出——文件

文件是输入输出的一个重要概念。从操作系统的角度看，每一个与主机相连的外部设备都被看作是一个文件，例如，键盘是标准输入文件，显示器是标准输出文件，磁盘既是输入文件也是输出文件。程序中用到的输入数据既可以通过键盘输入，也可以通过磁盘文件输入，程序的处理结果既可以输出到显示器上，也可以输出到磁盘文件中。磁盘文件一般用来处理输入输出数据量比较大的情况。

【任务 13.1】统计入学成绩（文件版）

【问题】 某重点大学的博士入学考试科目为外语和两门专业课，对于每个考生，输入各科考试成绩，并计算总分。要求用文件实现。

【想法】 对于大量的考生信息，不能每运行一次程序都录入大量数据，已经录入的成绩信息应该用文件保存下来，已经计算的总分也应该保存到文件中。

【算法】 设文件 student.txt 存放学生的成绩信息，函数 WriteToFile 实现录入考生的成绩信息并存入文件 student.txt，其算法描述如下：

输入：无

功能：录入考生的成绩信息

输出：无

伪代码

```
step1：以追加方式打开文件 student.txt;
step2：输入考生的各项信息;
step3：计算该生的总分;
step4：将考生的各项信息以及总分以追加方式存入文件 student.txt;
step5：如果继续录入,转 step2;否则关闭文件 student.txt，算法结束;
```

设函数 ReadFromFile 实现从文件 student.txt 中读出考生的成绩信息并输出，其算法描述如下：

输入：无

功能：输出考生的成绩信息

输出：无

伪代码

```
step1: 以只读方式打开文件 student.txt;
step2: 重复下述操作,直到文件 student.txt 的末尾:
    step2.1: 从文件 student.txt 读出一个考生的成绩信息;
    step2.2: 输出该生的成绩信息;
step3: 关闭文件 student.txt,算法结束;
```

前面各章程序中用到的输入数据都是在程序运行时通过键盘输入，处理结果都是输出到显示器上。程序运行结束后，运行结果也就丢失了，再次运行程序时须再次从键盘输入数据。那么，如何长期保存程序的处理结果？如何避免每次运行程序时都通过键盘输入数据？解决问题的方法就是文件，将数据以文件的形式存储在计算机的外存中，需要时可随时进行读写。VC++编程环境提供了库函数对文件进行操作，函数原型均在头文件“stdio.h”中。

13.1 概　　述

13.1.1 文件的概念

文件是指存储在外部介质（磁盘、磁带等）上的一组相关数据的有序集合，这个数据集合的名字称为**文件名**。文件名用来标识一个文件的属性，其一般结构为：主文件名.扩展名，其命名规则遵循操作系统的规定，扩展名通常表示文件的类型。表 13.1 给出了一些常见的扩展名。

表 13.1　常见的扩展名

扩展名	文 件 类 型	扩展名	文 件 类 型
c 或 cpp	C/C++语言源程序	exe	可执行文件
dat	数据文件	doc	Word 文件
txt	文本文件	ppt	PowerPoint 演示文稿
gif	图像文件	exl	Excel 电子表格文件

文件是操作系统进行数据管理的基本单位。换言之，要读取外部介质中的数据，必须首先按照文件名找到相应的文件，然后从这个文件中将数据读取出来；要将数据存储到外部介质中，必须首先在外部介质上建立一个文件，然后将数据写入这个文件。例如，应用程序的“文件打开”功能实现将文件的内容读入到内存，“文件保存”功能实现将数据从内存写入文件。

13.1.2 文本文件和二进制文件

从用户角度看，文件可分为普通文件和设备文件。普通文件是指保存在磁盘或其他

外部介质上的文件，如目标文件、可执行程序、数据文件等，设备文件是指与主机相连的各种外部设备，如显示器、打印机、键盘等。本章以磁盘文件为例介绍普通文件及其读写操作。

按照数据的组织形式，普通文件可分为文本文件和二进制文件。文本文件即 ASCII 码文件，扩展名是.txt、.c、.cpp、.h、.ini 等的文件大多是文本文件。二进制文件是将内存中的数据镜像（即原样输出）到文件中，扩展名是.exe、.dll、.lib、.dat、.gif、.bmp 等的文件大多是二进制文件。

文本文件中每个字符对应一个字节，用于存放该字符对应的 ASCII 码，例如，如果以文本方式在磁盘文件中存储十进制数 12345，则占 5 个字节，分别是字符'1'、'2'、'3'、'4'和'5'对应的 ASCII 码，如图 13.1(b)所示。因此，文本文件能够以记事本的方式打开。二进制文件按照内存中二进制编码形式存储，例如，如果以二进制形式在磁盘文件中存储十进制数 12345，则占 2 个字节（假设 int 型数据占 2 个字节），如图 13.1(c)所示。

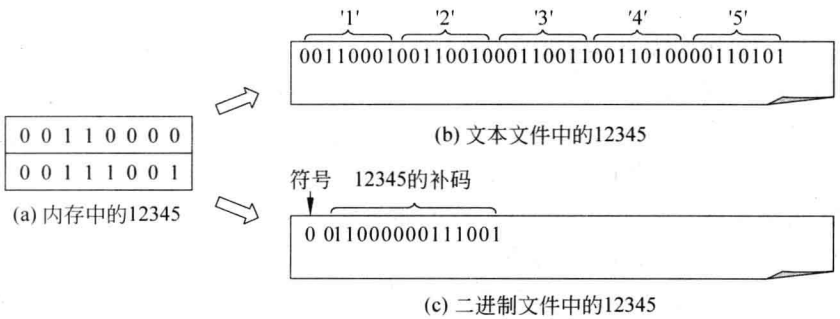


图 13.1 文本文件和二进制文件的存储示意图

文本文件中一个字节代表一个字符，便于对字符逐个进行处理，但一般占用较多存储空间，而且把文本文件读入内存需要将 ASCII 码转换成二进制码，把数据以文本方式写入文件需要将二进制码转换成 ASCII 码。因此，与二进制文件相比，文本文件的读写操作需要花费较多的时间。二进制文件可以节省存储空间和转换时间，但一个字节并不对应一个字符，因此，二进制文件一般不能以记事本的方式打开，即使能够打开，看起来也是一些乱码，如图 13.2 所示。

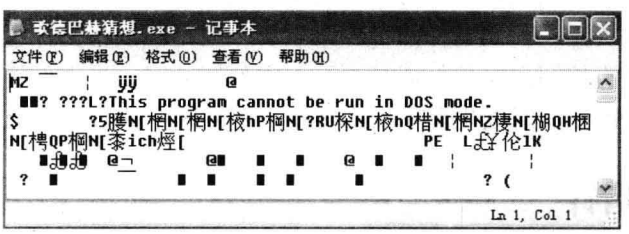


图 13.2 打开的二进制文件是乱码

13.1.3 文件缓冲区

操作系统对磁盘文件的存取速度远远小于对内存的存取速度。为了提高数据的存取效率，应用程序一般通过文件缓冲区对磁盘文件进行读写操作。所谓**文件缓冲区**就是一段连续的内存空间，应用程序与磁盘文件的数据交换通过文件缓冲区来完成，即在应用程序和磁盘文件之间设置一个文件缓冲区，如图 13.3 所示。

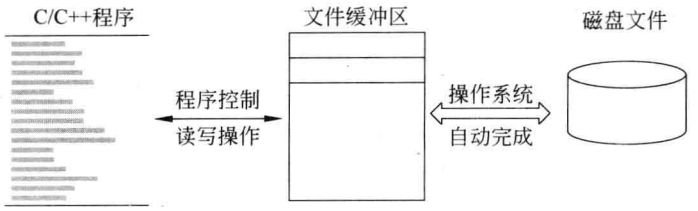


图 13.3 文件缓冲区的工作原理

使用文件一般按照如下操作流程进行：

① 打开文件。在打开一个文件时，系统自动在内存中开辟一个文件缓冲区，缓冲区的大小由具体的语言标准规定。由于磁盘数据的组织方式是按扇区进行的，每个扇区一般为 512B，一般微机中的 C/C++语言编译系统也把缓冲区大小定为 512B，恰好与磁盘的一个扇区大小相同，从而保证磁盘操作的高效性。

② 文件测试。操作系统需要对每一个打开的文件进行管理，如果不同的程序试图打开同一个文件，可能会引起文件读写数据的冲突，操作系统会禁止这种文件访问冲突，因此打开文件后要测试文件是否正确打开。

③ 读写操作。从文件中读数据时，操作系统首先自动把一个扇区的数据导入文件缓冲区中，然后由程序控制读入数据并进行处理，一旦数据读入完毕，系统会自动把下一个扇区的数据导入文件缓冲区中，以便继续读入数据。把数据写入文件时，首先由程序控制把数据写入文件缓冲区，一旦写满文件缓冲区，操作系统会自动把这些数据写入磁盘中的一个扇区，然后把文件缓冲区清空，以便接收新的数据。

④ 关闭文件。每一个打开的文件都会占用一个文件缓冲区，操作系统需要对每一个打开的文件进行管理，因此，可同时打开的文件是有限的，如果不及时关闭文件就会耗尽操作系统的文件资源。关闭文件的另一个重要作用是强制将文件缓冲区的数据写入文件，因为数据不是直接写入文件，而是先写入文件缓冲区，当文件缓冲区满时再写入文件，如果文件缓冲区不满时发生程序异常终止，缓冲区中的数据就会丢失。

13.1.4 文件指针

C/C++语言中的文件操作都是通过调用标准库函数来完成的，而且统一以文件指针的方式实现。在头文件 stdio.h 中定义了一个文件结构类型 FILE，FILE 中包含所有与文件操作相关的信息。在 VC++编程环境中，FILE 文件结构的类型定义如下：

```
struct _iobuf {
    char  *_ptr;                    //文件的当前位置指针
    int   _cnt;
    char  *_base;                   //缓冲区的首地址
    int   _flag;
    int   _file;                    //文件号
    int   _charbuf;
    int   _bufsiz;                  //缓冲区的长度
    char  *_tmpfname;
};
typedef struct _iobuf FILE;     //为结构体类型 struct _iobuf 定义别名 FILE
```

同普通变量相同，文件指针变量也要先定义后使用。

【语法】 定义文件指针变量的一般形式如下：

```
FILE *文件指针变量名;
```

其中，文件指针变量名是一个合法的标识符。

【语义】 定义文件指针变量。

如下语句定义了文件指针变量 fp 并将其初始化为空，以后要将 fp 与某个已打开的文件相关联。

```
FILE *fp = NULL;                        //定义并初始化文件指针变量 fp
```

文件指针是一种特殊的指针，每个打开的文件都有自己的文件指针和文件缓冲区，通过文件指针可以获得该文件的相关信息（例如文件号、文件的位置指针等），这些相关信息在系统打开文件时自动填入和使用，一般的编程人员不必关心 FILE 结构的具体内容。

良好的编程习惯 13.1

为了与非文件变量区分开，有些程序员习惯在有关文件的变量名前加上字母 f(file)，例如，将文件指针命名为 fp。系统提供的有关文件操作的库函数名也都以字母 f 开头，例如 fopen、fclose、fscanf、fprintf 等。

13.1.5 文件的位置指针

C/C++语言的文件是流式文件，即文件由一个个字节组成，对于文本文件，一个字节对应一个字符，对于二进制文件，一个字节对应一个二进制位串，如图 13.4 所示。流式文件将字节以一维方式组织，对文件的读写操作以字节为基本单位，文件中不存在其他数据类型和结构，对文件中数据的解释由程序来完成，这就增加了程序的灵活性。

每个打开的文件都有自己的文件缓冲区，在文件处理过程中，程序需要访问该缓冲区实现数据的读写。文件结构类型 FILE 中的成员_ptr 表示当前的位置指针，指向当前的读写位置，也就是将要操作的字节。一般情况下，在打开一个文件时，_ptr 位于文件首

部，即指向第一个字节，如图 13.4 所示。

图 13.4　流式文件的存储示意图

随着文件读写操作的进行，文件的位置指针_ptr 会自动向后移动。最后一个字节之后是文件结束的位置，因此，文件的位置指针_ptr 的最小值是 0，最大值是文件的长度。为了避免文件已结束时还对文件进行读写操作，C/C++语言提供了符号常量 EOF 用来表示文件结束，其值为−1。如果文件的位置指针_ptr 指向文件的末尾，则返回 EOF，如图 13.5 所示。

图 13.5　_ptr 指向文件的末尾将返回 EOF

如果要强制将文件的位置指针_ptr 指向其他位置，可以调用有关库函数将指针_ptr 移动到指定的位置。C/C++语言提供了 rewind、fseek 等函数移动文件的位置指针，ftell 函数获得_ptr 的当前位置，feof 函数判断_ptr 是否指向文件的末尾，具体请参见附录 C。

13.2　文件的打开与关闭

对一个文件进行读写操作，首先要打开这个文件，操作完毕后要将这个文件关闭。打开文件，实际上是建立文件缓冲区以及文件的各种相关信息，并将文件指针指向该文件，以便执行文件的读写操作。关闭文件，实际上是断开文件指针与文件之间的联系，从而也就禁止了对该文件的读写操作，并将文件缓冲区中的数据写入文件，以避免数据丢失。C/C++语言提供了 fopen 和 fclose 函数用于打开和关闭文件。

13.2.1　文件的打开

【函数原型】　fopen 函数的原型如下：

```
                                要打开的文件名
FILE *fopen(char *filename, char *mode)
文件指针                              打开方式
```

其中，FILE *表示文件指针；filename 表示要打开的文件名；mode 表示打开文件的方式，即打开文件后将要进行哪些操作，具体打开方式如表 13.2 所示。

【功能】　按指定方式打开文件，系统分配相应的文件缓冲区。

【返回值】　如果文件打开成功，则返回指向该文件的指针；否则返回空指针 NULL。

表 13.2　文件的打开方式

文件打开方式		含　义		文件的位置指针
文件类型	t(text)	打开一个文本文件	如缺省，则表示打开文本文件	
	b(binary)	打开一个二进制文件		
操作类型	r(read)	以只读方式打开一个已经存在的文件，不能向文件写数据		文件的开始
	r+	以读写方式打开一个已经存在的文件		
	w(write)	以只写方式创建一个新文件，不能从文件读数据。 如果文件已存在，则覆盖原文件		
	w+	以读写方式创建一个新文件。 如果文件已存在，则覆盖原文件		
	a(append)	以追加方式打开一个已经存在的文件，不能从文件读数据。 如果文件不存在，则创建这个文件		文件的末尾
	a+	以追加方式打开一个已经存在的文件，可以读写文件。 如果文件不存在，则创建这个文件		

注：文件的具体打开方式由操作类型 r、w 或 a，文件类型 t 或 b 这两部分组成，其中操作类型在前，文件类型在后并且可以省略，"+"一般放在文件类型的后面。例如"w+"和"wt+"均表示以读写方式创建一个新的文本文件。

在 fopen 函数中，如果文件名前面不带路径，则默认在当前路径下，即与应用程序所在路径相同；如果文件名前面带路径，则路径中的斜线“\”需要用双斜线“\\”表示，因为 C/C++语言规定斜线“\”是转义符，双斜线“\\”表示实际的“\”。 以下都是合法的打开文件操作：

```
FILE *fp = NULL;
fp = fopen("test.txt", "r");      //以只读方式打开当前文件夹下的 test.txt 文件
```

```
FILE *fp = NULL;
fp = fopen("c:\\aaa\\test.txt", "w+");//以读写方式在指定文件夹下打开一个文件
```

```
FILE *fp = NULL;
char *p = "c:\\aaa\\ test .txt ";
fp = fopen(p, "a+");              //以追加方式在指定文件夹下打开一个文件进行读写
```

如果 fopen 函数返回 NULL，则表明打开文件操作失败，其原因可能是以只读方式打开文件时文件不存在、路径不正确或是文件已被打开。为保证文件操作的可靠性，调用 fopen 函数后最好进行判断，以确认文件正常打开。下面是一种常见的用法：

```
FILE *fp = NULL;
fp = fopen("test.txt", "r");      //以只读方式打开当前文件夹下的 test.txt 文件
if (fp == NULL)
{
  printf("File open error !");
  exit(-1);                       //终止程序的执行并返回状态码-1
}
```

13.2.2 文件的关闭

【函数原型】 fclose 函数的原型如下：

```
                          ←——— 要关闭的文件
int fclose(FILE *filepointer)
```

其中，filepointer 是文件指针。

【功能】 关闭 filepointer 文件。

【返回值】 如果正常关闭，则返回 0；否则返回非 0。

用 fclose 函数关闭的文件一定是已经打开的文件，以下是 fclose 函数常见的用法：

```
FILE *fp = NULL;
fp = fopen(……);                    //以某种方式打开一个文件
⋮                                   //对文件进行读写操作
fclose(fp);                         //关闭 fp 指向的文件
```

13.3 文件的读写操作

磁盘文件的读写操作是针对磁盘而言的，读文件操作（简称读操作）是将数据从磁盘文件中读取出来，写文件操作（简称写操作）是将数据写入到磁盘文件中，如图 13.6 所示。

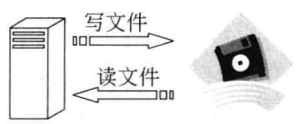


图 13.6 读文件与写文件的概念

C/C++语言提供了字符方式文件读写函数 fgetc 和 fputc、字符串方式文件读写函数 fgets 和 fputs、格式化方式文件读写函数 fscanf 和 fprintf 用于读写文本文件，分别与字符数据输入输出函数 getchar 和 putchar、字符串数据输入输出函数 gets 和 puts、格式化数据输入输出函数 scanf 和 printf 类似，区别在于前者的操作对象是标准输入输出文件（即键盘/显示器），后者的操作对象是磁盘文件。此外，还提供了二进制文件读写函数 fread 和 fwrite 用于读写二进制文件。

13.3.1 字符方式文件读写

字符方式文件读写就是以字符为单位进行文件读写操作，即每次可从文件读出一个字符或向文件写入一个字符。

1. fgetc 函数

【函数原型】 fgetc 函数的原型如下：

其中，filepointer 是文件指针。

【功能】 从 filepointer 文件的当前位置读出一个字符，同时将文件的位置指针_ptr 后移一个字节。读出的字符一般要保存到一个字符型变量中。

【返回值】 如果读取成功，则返回读取的字节值；如果读到文件尾或出错，则返回 EOF。

2. fputc 函数

【函数原型】 fputc 函数的原型如下：

其中，c 是要写入字符的 ASCII 码，可以是字符常量，也可以是字符变量；filepointer 是文件指针。

【功能】 向 filepointer 文件的当前位置写入一个 ASCII 码值为 c 的字符，同时将文件的位置指针指向下一个字节。

【返回值】 如果写入成功，则返回写入的字节值；否则返回 EOF。

例 13.1 将键盘上输入的若干行文字存入文件 test.txt 中。要求用字符方式完成文件读写操作。

解： 以行为单位，接收从键盘上输入的一行文字，该行文字要逐个字符读取并写入文件，输入一行文字的结束标志是回车符。

【算法】 由于要对文件 test.txt 进行写操作，因此，用只写方式打开文件，算法如下：

伪代码

```
step1: 以只写方式打开文件 test.txt;
step2: 执行下述操作,从键盘读入一行文字并写入文件 test.txt:
    step2.1: ch = 从键盘读入一个字符;
    step2.2: 当 ch 不是回车符时,重复执行下述操作:
        step2.2.1: 将 ch 写入文件 test.txt;
        step2.2.2: ch = 从键盘读入下一个字符;
    step2.3: 将'\n' 写入文件 test.txt;
step3: 如果继续读入下一行文字,则转 step2;
       否则, 关闭文件 test.txt,算法结束;
```

【程序】 fputc 函数不仅可以向文件写入可显示的字符，还可以写入转义字符。程序如下：

```
1 |  /*  example13-1.cpp  */
```

```
#include <stdio.h>          //使用printf、fopen等库函数,以及结构体类型FILE
                            //空行,以下是主函数
int main( )
{
  FILE *fp = NULL;          //定义文件指针fp并初始化为空
  char ch, flag = 'y';//flag是继续输入标志,flag等于'y'或'Y'时继续输入
  fp = fopen("test.txt","w");  //以只写方式打开文件test.txt
  if (fp == NULL)           //判断文件是否正确打开
  {
    printf("文件打开失败!\n");
    return -1;              //结束程序的执行,并将状态码-1返回给操作系统
  }
  while((flag == 'y'||flag == 'Y'))//当flag等于'y'或'Y'时执行循环
  {
    printf("请输入一行文字: "); //输出提示信息
    fflush(stdin);          //清空键盘缓冲区,以便接收正确输入
    ch = getchar( );        //变量ch接收从键盘读入的一个字符
    while (ch != '\n')      //当读入的字符不是回车符时,执行内层循环
    {
      fputc(ch, fp);        //将字符ch写入文件fp
      ch = getchar( );      //读入下一个字符
    }
    fputc('\n', fp);     //退出内层循环,结束一行文字,将换行符'\n'写入fp
    printf("继续输入吗? 继续,请输入y或Y: ");
    scanf("%c", &flag);
  }
  fclose(fp);               //操作结束,关闭文件fp
  return 0;                 //将0返回操作系统,表明程序正常结束
}
```

运行结果如下（下划线为用户输入）：

```
请输入一行文字: I love China!
继续输入吗? 继续,请输入y或Y: y
请输入一行文字: I love you!
继续输入吗? 继续,请输入y或Y: n
```

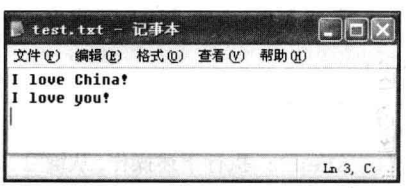


图 13.7　文件 test.txt 的操作结果

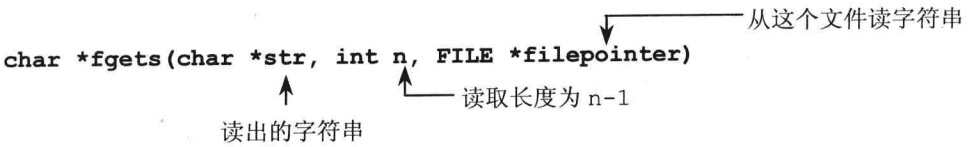
执行程序后，文件 test.txt 的内容如图 13.7 所示。

13.3.2　字符串方式文件读写

字符串方式文件读写就是以字符串为单位进行文件读写操作，即每次可从文件读出一个字符串或向文件写入一个字符串。

1. fgets 函数

【函数原型】　fgets 函数的原型如下：



其中，str 可以是字符数组也可以是字符串指针，用来存储读出的字符串；n 表示字符串的长度，这个长度包括终结符'\0'，因此，读出的字符个数是 n–1；如果在读取过程中遇到换行符'\n'，则将换行符之前的字符串返回，此时，读出的字符个数不一定是 n–1；filepointer 是文件指针。

【功能】 从 filepointer 文件的当前位置读取长度为 n–1 的字符串，在末尾加上字符串终结符'\0'存入 str 所指内存单元中，同时将文件的位置指针后移 n–1 个字节。

【返回值】 如果读取成功，则返回指向字符串的指针；如果读到文件尾或出错，则返回 NULL。

2. fputs 函数

【函数原型】 fputs 函数的原型如下：

其中，str 是要写入的字符串，可以是字符串常量，也可以是字符数组或字符串指针；filepointer 是文件指针变量。

【功能】 向 filepointer 文件的当前位置写入字符串 str，同时将文件的位置指针向后移动字符串长度个字节。

【返回值】 如果写入成功，则返回最后写入字符的字节值；否则返回 EOF。

例 13.2 假设文件 test.txt 的内容如图 13.7 所示，设计程序在显示器上输出文件 test.txt 的内容。要求用字符串方式完成文件读写操作。

解：由于要对文件进行读操作，因此，用只读方式打开文件 test.txt，程序如下：

```
1   /*   example13-2.cpp   */
2   #include <stdio.h>     //使用 printf、fopen 等库函数,以及结构体类型 FILE
3                                          //空行,以下是主函数
4   int main( )
5   {
6     FILE *fp = NULL;                     //定义文件指针 fp 并初始化为空
7     char str[80];
8     fp = fopen("test.txt", "r"); //以只读方式打开当前文件夹的文件 test.txt
9     if (fp == NULL)                      //判断文件是否正确打开
10    {
11        printf("文件打开失败!\n");
12        return -1;              //结束程序的执行,并将状态码-1 返回给操作系统
13    }
14    fgets(str, 15, fp);        //读出 15 个字符,即读出"I love China!\n"
```

```
    printf("%s", str);
    fgets(str, 13, fp);              //读出 13 个字符,即读出"I love you!\n"
    printf("%s", str);
    fclose(fp);                      //关闭文件
    return 0;                        //将 0 返回操作系统,表明程序正常结束
}
```

读取文件 test.txt 的具体过程如图 13.8 所示，运行结果如下：

```
I love China!
I love you!
```

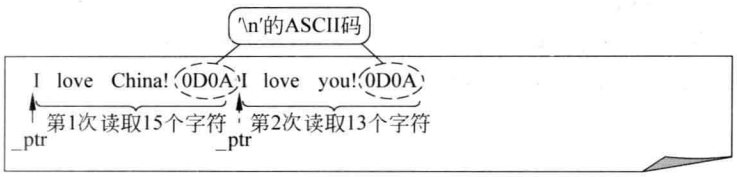


图 13.8　读取文件 test.txt 的具体过程

13.3.3　格式化方式文件读写

格式化方式文件读写就是以某种格式进行文件读写操作，即可以从文件读出或向文件写入某种指定格式的数据。

1. fscanf 函数

【函数原型】　fscanf 函数的原型如下：

```
从这个文件读 ──→                                    ←── 输入列表
int fscanf(FILE *filepointer, char *format[,address,…])
                                  ↑
                                  └── 读取的格式要求
```

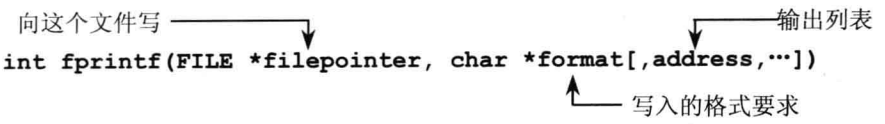
其中，filepointer 是文件指针；format 是格式控制，与 scanf 函数的格式控制相同；address 是输入列表，通常为变量的地址。

【功能】　从 filepointer 文件的当前位置按 format 格式读取数据并存入输入列表的变量中。

【返回值】　如果读取成功，则返回读取的数据个数；如果读到文件尾或出错，则返回 EOF。

2. fprintf 函数

【函数原型】　fprintf 函数的原型如下：



其中，filepointer 是文件指针；format 是格式控制，与 printf 函数的格式控制相同；address 是输出列表，通常是变量或表达式。

【功能】 将输出列表中的数据按照指定格式写入 filepointer 文件的当前位置。

【返回值】 如果写入成功，则返回写入的字节数；否则返回 EOF。

例 13.3 从键盘上输入学生的成绩信息并存入文件 test.txt 中。为简单起见，假设学生的成绩信息只包括姓名和总成绩。要求用格式化方式完成文件读写操作。

解： 由于题目要求对文件进行写操作，考虑到要保存文件 test.txt 中原有的成绩信息，可以用追加方式打开文件。程序如下：

```
/*    example13-3.cpp    */
#include <stdio.h>    //使用 printf、fopen 等库函数,以及结构体类型 FILE
typedef struct        //定义结构体类型 StudentType
{
  char name[10];      //姓名最多 4 个汉字
  double total;
} StudentType;
                      //空行,以下是主函数
int main( )
{
  FILE *fp = NULL;    //定义文件指针 fp 并初始化为空
  char flag = 'y';    //flag 是继续输入标志,flag 等于'y'或'Y'时继续输入
  StudentType stu;    //定义结构体变量 stu
  fp = fopen("test.txt", "a"); //以追加方式打开当前文件夹的文件 test.txt
  if (fp == NULL)     //判断文件是否正确打开
  {
    printf("文件打开失败!\n");
    return -1;        //结束程序的执行,并将状态码-1 返回给操作系统
  }
  while((flag == 'y' || flag == 'Y'))//当 flag 等于'y'或'Y'时执行循环
  {
    printf("请输入学生的姓名: ");    //输出提示信息
    scanf("%s", stu.name);          //stu.name 是字符数组名,不用加&
    printf("请输入学生的总成绩: ");  //输出提示信息
    scanf("%lf", &stu.total);
    fprintf(fp, "%-10s:%6.2f", stu.name, stu.total);//以指定格式写入文件
    fputc('\n', fp);                //将换行符写入文件
    fflush(stdin);                  //清空键盘缓冲区,以接收正确的输入
    printf("继续输入吗? 继续,请输入 y 或 Y:");
    scanf("%c", &flag);
  }
  fclose(fp);                       //关闭文件
  return 0;                         //将 0 返回操作系统,表明程序正常结束
}
```

运行结果如下（下划线为用户输入）：

```
请输入学生的姓名：陆宇
请输入学生的总成绩：237
继续输入吗？继续,请输入 y 或 Y：y
请输入学生的姓名：汤晓影
请输入学生的总成绩：218
继续输入吗？继续,请输入 y 或 Y：n
```

执行程序后，文件 test.txt 的内容如图 13.9 所示。再次执行程序，输入的成绩信息会追加到原有内容之后。

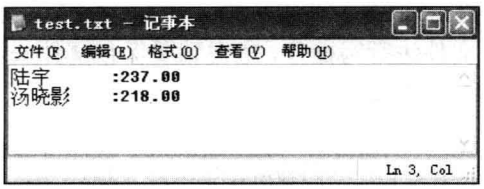


图 13.9　文件 test.txt 的操作结果

13.3.4　二进制方式文件读写

二进制方式文件读写就是对二进制文件进行读写操作，通常以字节为单位。

1. fread 函数

【语法】　fread 函数的原型如下：

其中，ptr 是通用指针，指向内存中的某个起始地址；size 表示数据的大小（以字节为单位）；n 表示读出数据的个数；filepointer 是文件指针。

【功能】　从 filepointer 文件的当前位置读出 n 个数据，每个数据的大小是 size 个字节，并将读出的数据存放在 ptr 所指向的内存单元中，同时，将文件的位置指针向后移动 n*size 个字节。

【返回值】　如果操作成功，则返回读出的数据个数；否则返回 0。

2. fwrite 函数

【函数原型】　fwrite 函数的原型如下：

其中，ptr 是通用指针，指向内存中的某个起始地址；size 表示数据的大小（以字节为单位）；n 表示写入数据的个数；filepointer 是文件指针。

【功能】 将 ptr 所指内存的 n 个大小为 size 个字节的数据写入 filepointer 文件的当前位置，同时，将文件的位置指针向后移动 n*size 个字节。

【返回值】 如果操作成功，则返回写入数据的个数；否则返回 0。

例 13.4 从键盘上输入学生的成绩信息并存入文件 test.dat 中，再输出全部学生的成绩信息。为简单起见，假设学生的成绩信息只包括姓名和总成绩。要求用二进制文件。

解：用追加并读写方式打开二进制文件 test.dat，程序如下：

```
/*   example13-4.cpp   */
#include <stdio.h>      //使用printf、fopen等库函数,以及结构体类型FILE
typedef struct                          //定义结构体类型 StudentType
{
  char name[10];                        //姓名最多为 4 个汉字
  double total;
} StudentType;
                                        //空行,以下是主函数
int main( )
{
  FILE *fp = NULL;                      //定义文件指针 fp 并初始化为空
  char flag = 'y';    //flag 是继续输入标志,flag 等于'y'或'Y'时继续输入
  StudentType stu;                      //定义结构体变量 stu
  fp = fopen("test.dat","ab+"); //以追加方式打开二进制文件 test.dat
  if (fp == NULL)                       //判断文件是否正确打开
  {
    printf("文件打开失败!\n");
    return -1;              //结束程序的执行,并将状态码-1 返回给操作系统
  }
  while ((flag == 'y'||flag == 'Y')) //当 flag 等于'y'或'Y'时执行循环
  {
    printf("请输入学生的姓名：");        //输出提示信息
    scanf("%s", stu.name);              //stu.name 是字符数组名,不用加&
    printf("请输入学生的总成绩：");      //输出提示信息
    scanf("%lf", &stu.total);
    fwrite(&stu, sizeof(StudentType), 1, fp);  //将 stu 写入文件 fp
    fflush(stdin);                  //清空键盘缓冲区,以接收正确的输入
    printf("继续输入吗? 继续,请输入 y 或 Y：");
    scanf("%c", &flag);
  }
  rewind(fp);                           //将文件的当前位置指针置回文件的开头
  printf("全部学生的成绩如下：\n");
  while (!feof(fp))                     //当文件未结束时执行循环
  {
```

```
        fread(&stu,sizeof(StudentType),1,fp);//从文件 fp 读入 StudentType
        printf("%-10s: %6.2f\n", stu.name, stu.total);
    }
    fclose(fp);                          //关闭文件
    return 0;                            //将 0 返回操作系统,表明程序正常结束
}
```

运行结果如下（下划线为用户输入）：

```
请输入学生的姓名：陆宇
请输入学生的总成绩：237
继续输入吗？继续,请输入 y 或 Y：y
请输入学生的姓名：汤晓影
请输入学生的总成绩：218
继续输入吗？继续,请输入 y 或 Y：n
全部学生的成绩如下：
陆宇       : 237.00
汤晓影     : 218.00
```

13.4 解决任务 13.1 的程序

在录入成绩信息时，为保存已经录入的数据，以追加方式打开文件 student.txt；在输出成绩信息时，以只读方式打开文件 student.txt。采用格式化方式对文件进行读写操作，程序如下：

```
/*   duty13-1.cpp   */
#include <stdio.h>      //使用printf、fopen等库函数,以及结构体类型FILE
typedef struct          //定义结构体类型 StudentType
{
  char no[10];          //学号最多 9 位
  char name[10];        //姓名最多为 4 个汉字
  double foreign;
  double spec1;
  double spec2;
  double total;         //存放总分
} StudentType;
void WriteToFile( );    //函数声明,向文件写入数据
void ReadFromFile( );   //函数声明,从文件读出数据
                        //空行,以下是主函数
int main( )
{
  int select;
  do                    //重复执行录入成绩或输出成绩操作
  {
    printf("1. 录入成绩  2. 输出成绩  0. 退出\n");     //输出提示信息
    printf("请输入要执行的操作: ");
    scanf("%d", &select);
```

```
    switch (select)
    {
      case 1: WriteToFile( ); break;//函数调用,录入考生成绩并写入文件
      case 2: ReadFromFile( ); break;//函数调用,从文件读取考生成绩并输出
      default: printf("退出程序! "); break;
    }
  } while ((select == 1||select == 2));  //当用户输入 1 或 2 时执行循环
  return 0;                              //将 0 返回操作系统,表明程序正常结束
}
                                         //空行,以下是其他函数定义
void WriteToFile( )                      //函数定义,没有参数且没有返回值
{
  FILE *fp = NULL;                       //定义文件指针 fp 并初始化为空
  StudentType stu;                       //定义结构体变量 stu
  char flag = 'y';    //flag 是继续输入标志,flag 等于'y'或'Y'时继续输入
  fp = fopen("student.txt", "a"); //以追加方式打开文件 student.txt
  if (fp == NULL)                        //判断文件是否正确打开
  {
    printf("文件打开失败!\n");
    return -1;                 //结束程序的执行,并将状态码-1 返回给操作系统
  }
  while ((flag =='y'||flag =='Y')) //当 flag 等于'y'或'Y'时执行循环
  {
    printf("请输入考生考号:");          //输出提示信息
    scanf("%s", stu.no);                 //stu,no 是字符数组名,不用加&
    printf("请输入考生姓名:");          //输出提示信息
    scanf("%s", stu.name);               //stu,name 是字符数组名,不用加&
    printf("请输入考生外语成绩:");      //输出提示信息
    scanf("%lf", &stu.foreign);
    printf("请输入考生专业课 1 成绩:"); //输出提示信息
    scanf("%lf", &stu.spec1);
    printf("请输入考生专业课 2 成绩:"); //输出提示信息
    scanf("%lf", &stu.spec2);
    stu.total = stu.foreign + stu.spec1 + stu.spec2;
    fprintf(fp, "%10s%10s%8.2f ",stu.no, stu.name, stu.foreign);
    fprintf(fp,"%8.2f%8.2f%8.2f",stu.spec1, stu.spec2, stu.total);
    fputc('\n', fp);                     //将换行符写入文件
    fflush(stdin);                       //清空键盘缓冲区,以接收正确的输入
    printf("继续输入吗? 继续,请输入 y 或 Y:");
    scanf("%c", &flag);
  }
  fclose(fp);                            //关闭文件
  return;                                //结束函数 WriteToFile 的执行
}
void ReadFromFile( )                     //函数定义,没有参数且没有返回值
{
  FILE *fp = NULL;                       //定义文件指针 fp 并初始化为空
```

```
    StudentType stu;                          //定义结构体变量 stu
    fp = fopen("student.txt", "r"); //以只读方式打开文件 student.txt
    if (fp == NULL)                           //判断文件是否正确打开
    {
      printf("文件打开失败!\n");
      return -1;              //结束程序的执行,并将状态码-1 返回给操作系统
    }
    printf("  考生姓名     总分\n");    //打印表头
    while (!feof(fp))                         //当文件未结束时执行循环
    {
      fscanf(fp, "%s%s", stu.no, stu.name);     //以下两条语句为读文件
      fscanf(fp,"%lf%lf%lf%lf",&stu.foreign,&stu.spec1,&stu.spec2,&stu.total);
      printf("%10s%8.2f\n", stu.name, stu.total);  //在显示器上输出
    }
    fclose(fp);                               //关闭文件
    return;                                   //结束函数 ReadFromFile 的执行
}
```

运行结果 1（下划线为用户输入）：

```
1. 录入成绩  2. 输出成绩  0. 退出
请输入要执行的操作: 1
请输入考生考号: 0001
请输入考生姓名: 陆宇
请输入考生外语成绩: 87
请输入考生专业课 1 成绩: 80
请输入考生专业课 2 成绩: 76.5
继续输入吗? 继续,请输入 y 或 Y: y
请输入考生考号: 0002
请输入考生姓名: 汤晓影
请输入考生外语成绩: 75
请输入考生专业课 1 成绩: 82
请输入考生专业课 2 成绩: 78
继续输入吗? 继续,请输入 y 或 Y: n
```

运行结果 2（下划线为用户输入）：

```
1. 录入成绩  2. 输出成绩  0. 退出
请输入要执行的操作: 2
考生姓名     总分
陆  宇     243.50
汤晓影     235.00
```

运行结果 3（下划线为用户输入）：

```
1. 录入成绩  2. 输出成绩  0. 退出
请输入要执行的操作: 0
退出程序!
```

13.5 程序设计实例

13.5.1 实例 1——文件复制

【问题】 模拟实现操作系统的文件复制功能。

【想法】 设将文件 fileS 的内容复制到文件 fileT 中，可以依次读取文件 fileS 的每一个字符，并写入文件 fileT 中，则文件 fileS 应该以只读方式打开，文件 fileT 应该以只写方式打开。

【算法】 设函数 Copy 实现文件复制，其算法描述如下：

输入：源文件 fileS，目标文件 fileT

功能：文件复制

输出：无

伪代码

```
step1: 以只读方式打开文件 fileS;以只写方式打开文件 fileT;
step2: 当 fileS 尚未到文件尾,重复执行下述操作:
    step2.1: ch = 从文件 fileS 读出一个字符;
    step2.2: 将 ch 写入文件 fileT;
step3: 关闭文件 fileS;关闭文件 fileT;
```

【程序】 程序同时打开两个文件，因此，设两个文件指针分别指向两个已打开的文件，源文件和目标文件均可以带路径。程序如下：

```
1   /*   文件复制.cpp   */
2   #include <stdio.h>      //使用printf、fopen等库函数,以及结构体类型 FILE
3   void Copy(char fileS[ ], char fileT[ ]); //函数声明,文件拷贝
4                                            //空行,以下是主函数
5   int main( )
6   {
7     char fileS[30], fileT[30];      //存放源文件名和目标文件名
8     printf("请输入要复制的源文件:");  //输出提示信息
9     scanf("%s", fileS);             //输入源文件名,可以带路径
10    printf("请输入要复制的目标文件:"); //输出提示信息
11    scanf("%s", fileT);             //输入目标文件名,可以带路径
12    Copy(fileS, fileT);             //函数调用,实参是字符数组首地址
13    return 0;                       //将 0 返回操作系统,表明程序正常结束
14  }
15                                    //空行,以下是其他函数定义
16  void Copy(char fileS[ ], char fileT[ ])  //函数定义,字符数组作为形参
17  {
18    char ch;
19    FILE *fpSource, *fpTarget;      //定义文件指针 fpSource 和 fpTarget
20    fpSource = fopen(fileS, "r");   //以只读方式打开文件 fileS
21    if (fpSource == NULL)           //判断文件是否正确打开
22    {
23      printf("文件打开失败!\n");
24      return -1;                  //结束程序的执行,并将状态码-1 返回给操作系统
25    }
26    fpTarget = fopen(fileT, "w");           //以只写方式打开文件 fileT
27    if (fpTarget == NULL)                   //判断文件是否正确打开
28    {
29      printf("文件打开失败!\n");
30      return -1;                //结束程序的执行,并将状态码-1 返回给操作系统
31    }
32    while (!feof(fpSource))//当源文件 fpSource 未结束时执行循环
33    {
34      ch = fgetc(fpSource); //ch 接收从文件 fpSource 中读出的一个字符
```

```
35          fputc(ch, fpTarget);          //将 ch 写入目标文件 fpTarget
36      }
37      fclose(fpSource);                 //关闭源文件
38      fclose(fpTarget);                 //关闭目标文件
39      printf("文件复制成功!\n");        //输出操作成功信息
40      return;                           //结束函数 Copy 的执行
41  }
```

13.5.2 实例 2——注册与登录

【问题】 模拟应用软件的用户注册和登录功能。

【想法】 如果是注册，则将用户输入的用户名和密码存入文件 table.txt；如果是登录，则将用户输入的用户名和密码与文件 table.txt 中的数据进行校验，限制用户输入次数不超过三次。

【算法】 设函数 Login 完成用户注册，算法描述如下：

输入：无

功能：用户注册

输出：无

伪代码

```
step1: 以追加方式打开文件 table.txt;
step2: 请用户输入用户名 name 和密码 password;
step3: 将 name 和 password 写入文件 table.txt;
step4: 关闭文件 table.txt;算法结束;
```

设函数 Rigester 完成用户登录，算法描述如下：

输入：无

功能：用户登录

输出：无

伪代码

```
step1: 以只读方式打开文件 table.txt;
step2: 初始化输入次数 count = 0;
step3: 当 count < 3 时，执行下述操作:
     step3.1: 请用户输入用户名 name 和密码 password;
     step3.2: count++;
     step3.3: 如果 name 和 password 在 table.txt 中，则显示“登录成功!”，跳出循环;
              否则转 step3.1 重新输入;
step4: 如果 count >= 3，则显示“输入超过三次，退出程序!”;
step5: 关闭文件 table.txt; 算法结束;
```

【程序】 程序提示用户选择登录或注册，如果选择登录，则调用函数 Rigester，如果选择注册，则调用函数 Login，其他选择均退出程序。程序如下：

```
/*   注册与登录.cpp   */
#include <stdio.h>    //使用 printf、fopen 等库函数,以及结构体类型 FILE
#include <string.h>   //使用库函数 strcmp
void Rigester( );     //函数声明,登录功能
void Login( );        //函数声明,注册功能
                      //空行,以下是主函数
int main( )
{
  int select;
  printf("1. 登录  2. 注册  0. 退出\n 请选择:");  //输出提示信息
  scanf("%d", &select);
  switch (select)
  {
    case 1: Rigester( ); break;   //函数调用,实现用户登录
    case 2: Login( ); break;      //函数调用,实现用户注册
    default: printf("输入错误,退出程序!\n"); break;
  }
  return 0;                       //将 0 返回操作系统,表明程序正常结束
}
                                  //空行,以下是其他函数定义
void Rigester( )                  //函数定义,没有形参和返回值
{
  FILE *fp = NULL;                //定义文件指针 fp 并初始化为空
  char str1[10], str2[10];
  char name[10], password[10];
  int count = 0, flag = 0;        //flag 是匹配成功的标志,0 表示不成功
  fp = fopen("table.txt", "r");   //以只读方式打开文件 table.txt
  while (flag == 0 && count < 3) //当匹配不成功且录入次数不到 3 次
  {
    printf("请输入用户名: ");      //输出提示信息
    scanf("%s", name);
    printf("请输入密码: ");        //输出提示信息
    scanf("%s", password);
    count++;                      //录入次数加 1
    rewind(fp);                   //文件 fp 的位置指针置回开头
    while (!feof(fp))             //当文件 fp 未结束时执行循环
    {
      fscanf(fp, "%s%s", str1, str2);  //从文件 fp 中读出两个字符串
      if ((strcmp(str1, name) == 0) && (strcmp(str2, password) == 0))
      {
        flag = 1; break;       //置匹配成功标志,退出内层 while 循环
      }
    }
    if (flag == 0)                //退出内层循环后,测试匹配是否成功
      printf("用户名或密码错,请重新输入!\n");
  }
  if (flag == 1)                  //退出外层循环后,测试匹配是否成功
     printf("登录成功! \n");
  else
```

```
        printf("输入超过 3 次,退出程序!\n");
    fclose(fp);                          //关闭文件
    return;                              //结束函数 Rigester 的执行
}
void Login( )                            //函数定义,没有形参和返回值
{
    FILE *fp = NULL;                     //定义文件指针 fp 并初始化为空
    char name[10], password[10];
    fp = fopen("table.txt", "a");        //以追加方式打开文件 table.txt
    printf("请输入用户名: ");             //输出提示信息
    scanf("%s", name);                   //name 为字符数组首地址,不用加&
    printf("请输入密码: ");               //输出提示信息
    scanf("%s", password);               //password 为字符数组首地址,不用加&
    fprintf(fp, "%10s%10s", name, password);//以指定格式写入文件
    printf("注册成功! \n");
    fclose(fp);                                  //关闭文件
    return;                                      //结束函数 Login 的执行
}
```

运行结果 1（下划线为用户输入）：

```
1. 登录  2. 注册   0. 退出
请选择: 2
请输入用户名: wanghm
请输入密码: 123456
注册成功!
```

运行结果 2（下划线为用户输入）：

```
1. 登录  2. 注册   0. 退出
请选择: 1
请输入用户名: wanghm
请输入密码: 123456
登录成功!
```

习 题 13

一、选择题

1. 打开 D 盘上 user 文件夹中的文件 abc.txt 进行读写操作，下面满足此要求的函数调用是（　　）。

A. fopen("D:\user\abc.txt", "r")　　B. fopen("D:\\user\\abc.txt", "r+")

C. fopen("D:\user\abc.txt", "rb")　　D. fopen("D:\\user\\abc.txt", "w+")

2. 若 fp 是指向某文件的指针，且已到文件末尾，则库函数 feof(fp)的返回值是（　　）。

A. EOF　　B. –1　　C. 非 0 值　　D. NULL

3. 以下不能将文件的位置指针重新移到文件开头位置的语句是（　　）。

A. rewind(fp);　　B. fseek(fp, 0, SEEK_SET);

C. fseek(fp, 0, SEEK_END);　　D. fseek(fp, 0, SEEK_CUR);

4. 以下程序执行后，文件 test.txt 的内容是（　　）。

```
#include <stdio.h>
#include <string.h>
void fun(char *fName, char *str)
```

```
{
  FILE *fp;
  fp = fopen(fName, "w");
  for(int i = 0; i < strlen(str); i++)
  fputc(str[i], fp);
  fclose(fp);
}
int main( )
{
  fun("test.txt", "Hello, ");
  fun("test.txt", "World!");
  return 0;
}
```

A. Hello　　B. Hello, World!　　C. World!　　D. Hello,

二、简答题

1. 文件指针的作用是什么？文件位置指针的作用是什么？
2. 在对文件进行操作时，为什么要打开和关闭文件？
3. 文件缓冲区的作用是什么？

三、程序设计题

1. 个人理财。假设用一个文件保存你的个人储蓄情况，储蓄信息包括账号、日期、金额、类型（存款、取款），请设计程序对个人储蓄情况进行管理。

2. 将一个文件中的大写字母全部改写为小写字母，并将改写的部分另存为一个新的文件。

3. 将两个文件 file1 和 file2 连接成一个新文件 newFile。

4. 文件加密。用反转文件的方法对文件进行加密，所谓反转，即是将每个二进制位的 0 反转为 1、1 反转为 0。

5. 文件 testFile.cpp 是不带行号的 C++源程序文件，请为该文件加上行号。

第14章

再谈程序的基本结构

按照 C/C++程序文件（包括头文件和源程序文件）的数量，可以将 C/C++程序分为单文件程序和多文件程序。单文件程序是将所有的程序代码都放到一个源程序文件中，多文件程序通常包含一个或多个用户自定义头文件和一个或多个源程序文件，每个文件称为程序文件模块。严格地讲，结构化程序应该使用多文件结构，尤其对于大型程序。前面各章的程序都属于单文件程序，本章介绍多文件程序。

【任务 14.1】石头、剪子、布游戏

【问题】 石头、剪子、布的游戏规则是：用手表示石头、剪子或布，握紧拳头表示石头，伸出食指和中指表示剪子，伸出五指表示布；石头会硌坏剪子、剪子能剪断布、布能包住石头。设计程序实现游戏者和计算机之间的石头、剪子、布游戏，要求能够重复游戏，直到游戏者选择结束；在此过程中，还能够阅读游戏指南、查看当前战况等。

【想法】 一次游戏的过程是：计算机随机生成石头、剪子或布，游戏者从键盘输入石头、剪子或布，然后进行比较并给出游戏结果。由于需要查看战况，因此，需要保存游戏次数和每次的结果。

【算法】 设枚举类型 HandType {stone = 1, scissor, paper}表示石头、剪子和布，设枚举类型 ResultType{win, lose, tie}表示胜、负和平，设全局数组 gameResult[N]保存游戏结果，设全局变量 count 保存游戏次数，设头文件 game.h 存放枚举类型定义、全局变量定义等。设函数 game 实现一次游戏，函数 game 的算法描述如下：

输入：游戏次数 count，保存游戏结果的数组 gameResult[N]

功能：石头、剪子、布的一次游戏

输出：无

伪代码

```
step1: handComputer = 随机生成石头、剪子或布;
step2: handPlayer = 从键盘输入石头、剪子或布;
step3: 比较 handComputer 和 handPlayer，共有 9 种情况;
step4: gameResult[count] = 游戏结果;
step5: count++;
```

设函数 random 实现随机生成石头、剪子或布，函数 guide 完成游戏指南，函数 check 完成查看游戏战况。函数 random、guide 和 check 的算法较简单，请读者自行完成。

14.1 多文件程序

如果源程序文件的规模较大，应该将源程序文件分解为几个程序文件模块；如果一个大项目需要多人开发，应该将任务分解，每个人编写的程序代码放在自己的程序文件模块中。多文件程序需要解决以下两个关键问题：

① 如何将源程序文件分解为多个程序文件模块？

② 如何把若干个程序文件模块连接成一个完整的可执行文件？

14.1.1 将源程序文件分解为多个程序文件模块

在多文件程序中，头文件（即.h 文件）通常包含某些程序文件模块的共享信息，如符号常量定义、数据类型定义、全局变量定义和函数原型等。例如，某个项目往往使用一些固定的符号常量（如 PI = 3.1415926、E = 2.718 等），或者一些全局变量（如次数、人数等），就可以把这些符号常量和全局变量放到头文件中，然后每个人都可以用#include 预处理指令将这个头文件包含到自己的源程序文件中，这样就不必重复定义这些符号常量和全局变量，从而避免重复工作、减少差错。

在多文件程序中，源程序文件（即.c 或.cpp 文件）通常包含主函数和其他函数定义，相应的函数原型一般放在头文件中。由于整个程序的运行只能从主函数 main 开始，所以，只能有一个源程序文件包含 main 主函数。

例 14.1 图 14.1 所示为边长为 R 的正方形内切直径为 R 的圆，设计多文件程序求阴影部分的面积。

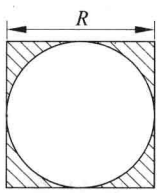


图 14.1 求阴影部分的面积

解：设函数 AreaSquare 求正方形的面积，函数 AreaCircle 求圆的面积，主函数调用函数 AreaSquare 和 AreaCircle 求得正方形的面积 area1 和圆的面积 area2，则 area1 – area2 即为阴影部分的面积。可以将程序分解为 1 个头文件和 3 个源文件，其中头文件 func.h 包含符号常量 PI 和函数声明，程序如下：

```
1   /*   func.h   */
2   #define PI 3.14                          //定义符号常量 PI
3   double AreaSquare(double R);             //函数声明,计算正方形的面积
4   double AreaCircle(double R);             //函数声明,计算圆的面积
```

源文件 func1.cpp 完成 AreaSquare 的函数定义，程序如下：

```
1   /*   func1.cpp   */
2   #include "func.h"                        //包含用户自定义头文件,注意用双引号
```

```
double AreaSquare(double R)          //函数定义,形参 R 为正方形的边长
{
   return R * R;
}
```

源文件 func2.cpp 完成 AreaCircle 的函数定义，程序如下：

```
/*   func2.cpp   */
#include "func.h"                    //包含用户自定义头文件,注意用双引号
double AreaCircle(double r)          //函数定义,形参 r 为圆的半径
{
  return PI * r * r;
}
```

源程序 example14-1.cpp 用来完成主函数，程序如下：

```
/*   example14-1.cpp   */
#include <stdio.h>                   //包含系统头文件,注意用尖括号
#include "func.h"                    //包含用户自定义头文件,注意用双引号
                                     //空行,以下为主函数
int main( )
{
  double R, area1, area2;
  printf("请输入正方形的边长:");  //输出提示信息
  scanf("%lf", &R);
  area1 = AreaSquare(R);             //函数调用,变量 area1 接收函数的返回值
  area2 = AreaCircle(R/2);           //函数调用,变量 area2 接收函数的返回值
  printf("阴影部分的面积是: %5.2f\n", area1 - area2);
  return 0;                          //将 0 返回操作系统,表明程序正常结束
}
```

运行结果如下（下划线为用户输入）：

```
请输入正方形的边长: 4
阴影部分的面积是: 3.44
```

14.1.2 构建多文件程序

VC++使用工程（也称项目）来管理程序文件模块。建立一个工程的步骤如下：

① 进入 VC++编程环境后，单击“文件”菜单，在弹出的下拉菜单中单击“新建”，在弹出的对话框中选中 Win32 Console Application（控制台应用程序），如图 14.2 所示；

② 在“位置”文本框创建一个文件夹，在“工程名称”文本框键入一个工程名称，单击“确定”按钮，如图 14.2 所示；

③ 在出现的“Win32 Application Step 1”窗体中选择“一个空工程”，单击“完成”按钮，在新建工程信息窗体中单击“确定”按钮。

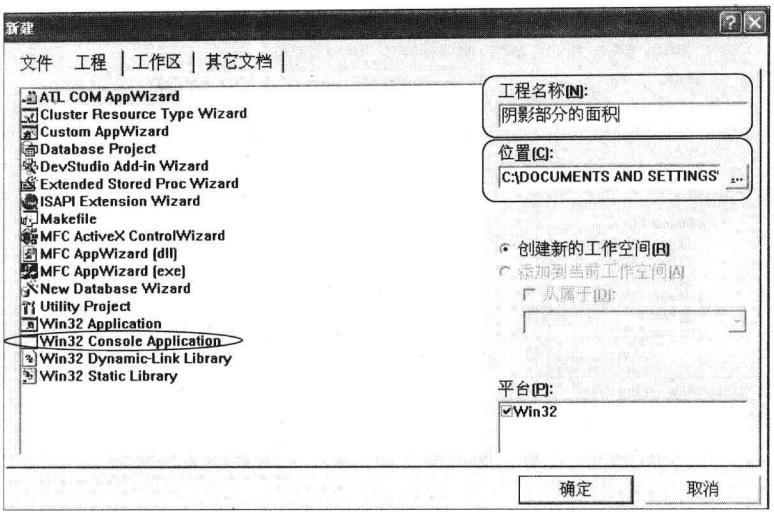


图 14.2 VC++环境下建立工程

工程的主要功能是管理程序文件模块，向工程添加程序文件模块的步骤如下：

① 在工程窗体中单击“文件”菜单，在弹出的下拉菜单中单击“新建”，在弹出的对话框中选中 C++ Source File（源程序文件）或 C/C++ Header File（头文件），如图 14.3 所示；

② 在对话框的右侧“添加到工程”文本框中出现一个工程名称，表示将建立的程序文件模块添加到该工程中，如图 14.3 所示；

③ 在“文件名”文本框中键入新建文件的名字（可以省略扩展名），单击“确定”按钮。

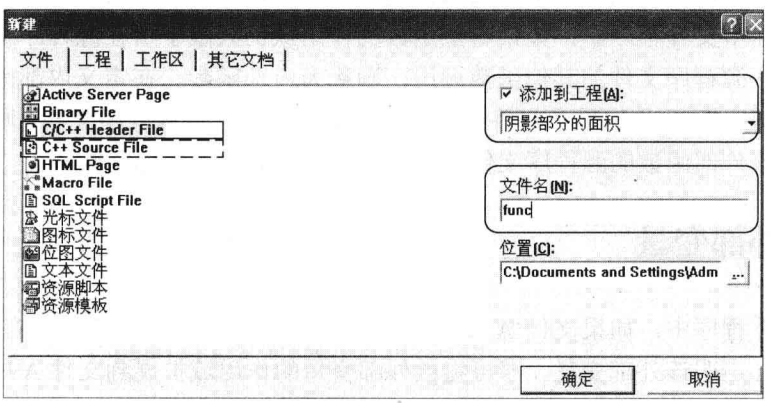


图 14.3 向工程添加头文件

依次向工程“阴影部分的面积”中添加头文件 func.h，源程序文件 func1.cpp、func2.cpp 和 example14-1.cpp，如图 14.4 所示。在左侧窗口“FileView”标签中显示了所有程序文件模块。双击某个文件名，该文件的所有程序代码就出现在右侧代码编辑窗口。

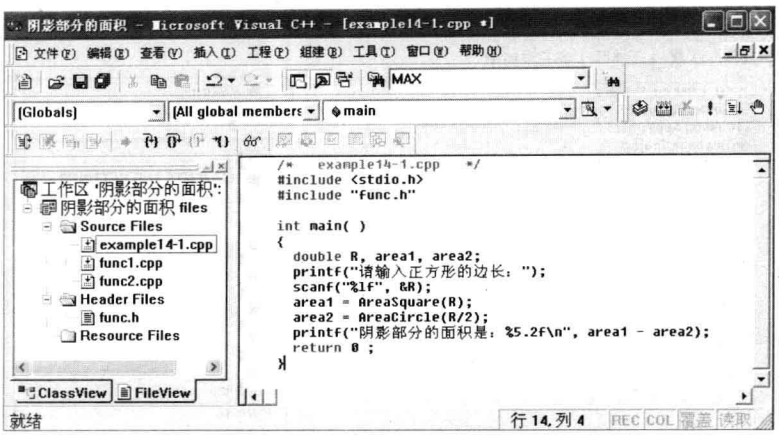


图 14.4　工程“阴影部分的面积”中的程序文件模块

14.2　外部变量和外部函数

外部变量也就是全局变量，是在所有函数（包括主函数 main）之外定义的变量。在单文件程序中，全局变量的作用域是所在源程序文件。在多文件程序中，如果希望全局变量的作用域仍只限于所在源程序文件，而不能被其他源程序文件引用，则称为静态全局变量，在定义静态全局变量时须加上关键字 static；如果希望将全局变量的作用域扩展到其他源程序文件，则称为外部变量，在引用该外部变量的源程序文件中用关键字 extern 声明外部变量。

本质上，函数是全局的，即函数的作用域从函数定义（或函数声明）的位置开始到程序结束。在多文件程序中，如果希望函数的作用域仍只限于所在源程序文件，即该函数只能被所在源程序文件的其他函数调用，则称为内部函数，在定义内部函数时函数首部须加上关键字 static；如果希望将函数的作用域扩展到其他源程序文件，则称为外部函数，在引用该外部函数的源程序文件中用关键字 extern 声明外部函数。

14.2.1　外部变量

在多文件程序中，如果文件 A 要引用文件 B 中定义的外部变量，则在文件 A 中用关键字 extern 声明该外部变量，即可将该外部变量的作用域扩展到文件 A 中。例如，如下均是合法的外部变量声明：

```
extern int count;                    //声明外部变量 count
extern int data[ ];                  //声明外部数组 data,无须指明数组长度
```

关键字 extern 提示编译程序变量 count 是在其他源程序文件中定义的，因此不需要为变量 count 分配存储空间，只是该源程序可以访问变量 count。由于外部变量声明不进行内存分配，因此，声明外部数组无须指明数组长度。

良好的编程习惯 14.1

当同一个变量在不同的源程序文件中进行外部变量声明时，编译程序无法检查所有声明是否与变量定义一致，因此，一个良好的编程习惯是：将共享变量的声明放在头文件中，然后在使用这个变量的源程序文件中用#include 预处理指令包含该头文件。

例 14.2 利用外部变量求两个一维数组元素的累加和。

解：设全局变量 sum 保存数组元素的累加和，则两个数组的元素之和要累加到同一个变量 sum 上，可以将程序分解为 1 个头文件和 2 个源程序文件，头文件 func.h 中存放求和函数 Sum 的函数原型，程序如下：

```
/*   func.h   */
void Sum(int r[ ], int n);
```

源程序文件 sum.cpp 完成函数 Sum 的定义，在源程序文件 sum.cpp 中要引用源程序文件 func.cpp 中定义的全局变量 sum ，因此，需要声明该外部变量，程序如下：

```
/*   sum.cpp   */
extern int sum;              //声明外部变量 sum,该变量在其他源程序文件中定义
void Sum(int r[ ], int n)  //函数定义,累加一维数组的元素值
{
  for (int i = 0; i < n; i++)
      sum = sum + r[i];
  return;                      //结束函数 Sum 的执行
}
```

源程序文件 example14-2.cpp 完成主函数，程序如下：

```
/*   example14-2.cpp   */
#include <stdio.h>          //包含系统头文件,注意用尖括号
#include "func.h"           //包含用户自定义头文件,注意用双引号
int sum = 0;                //定义全局变量 sum 并初始化为 0
                            //空行,以下为主函数
int main( )
{
  int a[5] = {1, 2, 3, 4, 5};//定义一维数组 a[5]并初始化
  int b[10] = {1, 1, 1, 1, 1, 2, 2, 2, 2, 2};
  Sum(a, 5);                //函数调用,累加数组 a[5]所有元素之和
  Sum(b, 10);     //函数调用,在全局变量 sum 上再累加数组 b[10]所有元素之和
  printf("数组元素的累加和是: %d\n", sum);
  return 0;                 //将 0 返回操作系统,表明程序正常结束
}
```

运行结果如下：

```
数组元素的累加和是: 30
```

14.2.2 外部函数

在多文件程序中，如果一个函数可以被其他源程序文件调用，则称为外部函数。在定义外部函数时，在函数首部前面须加上关键字 extern，缺省 extern 则默认为外部函数。在需要调用外部函数的源程序文件中，用关键字 extern 声明该外部函数。

例 14.3 利用外部函数求两个整数中较大值与较小值的差。

解：设函数 Max 求两个整数的较大值，函数 Min 求两个整数的较小值，主函数调用函数 Max 和 Min 求得整数 x 和 y 中的较大值 max 和较小值 min，则 max-min 即为所求。可以将程序分解为 3 个源文件，其中，源文件 func1.cpp 完成求两个整数的较大值，程序如下：

```
/*   func1.cpp   */
extern int Max(int x, int y)      //定义外部函数 Max,extern 可以省略
{
  if (x >= y)
    return x;
  else
    return y;
}
```

源文件 func2.cpp 完成求两个整数的较小值，程序如下：

```
/*   func2.cpp   */
extern int Min(int x, int y)      //定义外部函数 Min,extern 可以省略
{
  if (x <= y)
    return x;
  else
    return y;
}
```

源程序 example14-3.cpp 用来完成主函数，主函数需要调用在源程序文件 func1 中定义的函数 Max 和在源程序文件 func2 中定义的函数 Min，可以将函数 Max 和 Min 的原型放在头文件中，本例使用声明外部函数的方法，程序如下：

```
/*   example14-3.cpp   */
#include <stdio.h>                    //使用库函数 printf 和 scanf
extern int Max(int x, int y);         //外部函数声明,本程序将要调用
extern int Min(int x, int y);         //外部函数声明,本程序将要调用
                                      //空行,以下为主函数
int main( )
{
  int x, y, max, min;
  printf("请输入两个整数: ");          //输出提示信息
  scanf("%d%d", &x, &y);
```

```
11      max = Max(x, y);                    //函数调用,求 x 和 y 的较大值
12      min = Min(x, y);                    //函数调用,求 x 和 y 的较小值
13      printf("最大值与最小值的差是: %d\n", max - min);
14      return 0;                           //将 0 返回操作系统,表明程序正常结束
15  }
```

良好的编程习惯 14.2

声明外部函数可以调用在其他源程序文件中定义的外部函数，但是有些程序员不提倡使用这种方法。试想，如果要在 50 个源程序文件中调用在源程序文件 A 中定义的函数，如果文件 A 中的函数定义发生了改变，如何找到这 50 个外部函数的声明呢？一个良好的编程习惯是：将函数原型放在头文件中，然后在所有调用这个函数的源程序文件中用#include 预处理指令包含该头文件，如果文件 A 中的函数定义发生了改变，则仅须修改相应头文件。

14.3 嵌 套 包 含

头文件自身可以包含#include 指令。如果一个源文件包含同一个头文件两次（称为嵌套包含），可能会产生编译错误。为避免出现嵌套包含现象，可以用条件编译有选择地编译源程序的不同部分。

14.3.1 条件编译

一般情况下，源程序中的所有语句都需要进行编译，但有时希望根据一定的条件去编译源程序中的不同部分，这就是**条件编译**。条件编译常常和宏定义配合使用，使得同一源程序在不同的编译条件下得到不同的目标代码，提高了程序的灵活性。

C/C++语言提供了三种条件编译：#if～#endif、#ifdef～#endif 和#ifndef～#endif。

1. #if～#endif

【语法】 #if～#endif 的一般形式如下：

```
#if  条件 1 ← 常量表达式
     程序段 1
#elif 条件 2
     程序段 2
.....                    } 可以没有
#else
     程序段 n
#endif
```

其中，条件是常量表达式，通常会出现宏名；#elif 指令可以没有，也可以有多个；#else 也可以没有；#endif 表示#if 指令的结束；每条指令单独占一行。

【语义】 如果条件1成立则编译程序段1，否则如果条件2成立则编译程序段2……，如果所有条件都不成立则编译程序段n。

2. #ifdef…#endif

【语法】 #ifdef…#endif的一般形式如下：

```
#ifdef  宏名1
     程序段1
#elif  宏名2
     程序段2
……                    } 可以没有
#else
     程序段n
#endif
```

其中，#elif指令可以没有也可以有多个；#else可以没有；#endif表示#ifdef指令的结束；每条指令单独占一行。

【语义】 如果定义了宏名1则编译程序段1，如果定义了宏名2则编译程序段2……，如果所有宏名都未定义则编译程序段n。

3. #ifndef～#endif

【语法】 #ifndef～#endif的一般形式如下：

```
#ifndef  宏名
     程序段1
#else
     程序段2
#endif
```

其中，#endif表示#ifndef指令的结束；每条指令单独占一行。

【语义】 如果没有定义宏名则编译程序段1，否则编译程序段2。

例14.4 显示不同书籍的相关信息，信息的格式相同。

解：将各种书籍的信息分别用头文件存储，在selectbook.cpp文件中根据符号常量ID的值包含相应的头文件。程序如下：

```
1   /*   officeXP.h     */
2   #define BNAME "Office XP入门版"
3   #define LEVEL "初级"
4   #define PRICE 48
5   #define SERIES "办公自动化"
```

```
1   /*   java2.h    */
2   #define BNAME "Java 2"
3   #define LEVEL "高级"
4   #define PRICE 55
5   #define SERIES "程序设计"
```

```
/*    flash.h     */
#define BNAME "Flash MX 2004"
#define LEVEL "初级"
#define PRICE 28
#define SERIES "多媒体网页设计"
```

```
/*    selectbook.h    */
#if ID == 1
#include "officeXP.h"
#elif ID == 2
#include "java2.h"
#else
#include "flash.h"
#endif
```

```
/*   example14-4.cpp      */
#include <stdio.h>                          //包含系统头文件,注意用尖括号
#define ID 1                                //定义符号常量 ID,ID 也称为宏名
#include "selectbook.cpp"                   //包含用户自定义头文件,注意用双引号
                                            //空行,以下为主函数
int main( )
{
    printf("=====书籍信息=====\n");
    printf("书名: %s\n", BNAME);            //预处理后，符号常量 BNAME、
    printf("等级: %s\n", LEVEL);            // LEVEL、PRICE、SERIES 等
    printf("定价: %d\n", PRICE);            //根据 ID 的值被定义为相应的值
    printf("种类: %s\n", SERIES);
    return 0;                               //将 0 返回操作系统，表明程序正常结束
}
```

运行结果如下：

```
=====书籍信息=====
书名: Office XP 入门版
等级: 初级
定价: 48
种类: 办公自动化
```

14.3.2 保护头文件

当头文件包含其他头文件时，如果出现嵌套包含现象，会导致编译错误。如图 14.5 所示，在编译源程序文件 func.cpp 时，头文件 file1.h 就会被编译两次。可以使用条件编译将头文件的某些内容闭合起来，从而达到保护头文件的目的。

例 14.5 已知底面半径和高，求圆柱体的底面面积和体积。

解：可以将程序分解为 3 个头文件和 3 个源程序文件，头文件 file1.h 包含符号常量 PI 的定义，程序如下：

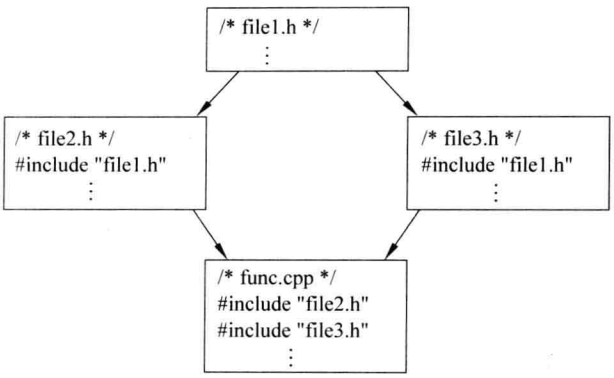


图 14.5　嵌套包含示例

```
/*   file1.h   */
#ifndef PI                          //如果没有定义宏名 PI,即尚未定义该符号常量
#define PI 3.14                     //则定义符号常量 PI
#endif
```

头文件 file2.h 包含头文件 file1.h，以及函数 Area 的原型，程序如下：

```
/*   file2.h   */
#include "file1.h"                  //包含用户自定义头文件
double Area(int r);                 //函数声明,求圆的面积
```

头文件 file3.h 包含头文件 file1.h，以及函数 Volume 的原型，程序如下：

```
/*   file3.h   */
#include "file1.h"                      //包含用户自定义头文件
double Volume(int r, int h);            //函数声明,求圆柱体的体积
```

源程序文件 area.cpp 包含头文件 file2.h，以及函数 Area 的定义，程序如下：

```
/*   area.cpp   */
#include "file2.h" //包含用户自定义头文件,file2.h 中含有函数 Area 的原型
double Area(int r)                                  //函数定义,r 为圆的半径
{
  return PI * r * r;
}
```

源程序文件 volume.cpp 包含头文件 file3.h，以及函数 Volume 的定义，程序如下：

```
/*   volume.cpp   */
#include "file3.h"                  //包含用户自定义头文件,file3.h 中含有
                                    //函数 Volume 的原型
double Volume(int r, int h)         //函数定义,r 为圆的半径,h 为圆柱体的高
{
  return PI * r * r * h;
}
```

源程序文件 func.cpp 包含头文件 file2.h 和 file3.h，以及主函数，程序如下：

```
/*   example14-5.cpp   */
#include <stdio.h>                          //包含系统头文件,注意用尖括号
#include "file2.h"                          //包含用户自定义头文件,注意用双引号
#include "file3.h"                          //包含用户自定义头文件,注意用双引号
                                            //空行,以下为主函数
int main( )
{
  int r, h;
  printf("请输入底面半径和高: ");          //输出提示信息
  scanf("%d%d", &r, &h);
  printf("底面积为: %8.2f\n", Area(r));     //输出调用函数 Area 的结果
  printf("体积为: %8.2f\n", Volume(r, h));//输出调用函数 Volume 的结果
  return 0;                                 //将 0 返回操作系统,表明程序正常结束
}
```

在首次包含头文件 file1.h 时,由于尚未定义符号常量 PI,所以预处理器执行#ifndef…#endif 之间的内容。如果再次包含 file1.h,由于已经定义了符号常量 PI,所以预处理器将忽略#ifndef…#endif 之间的内容,从而避免嵌套包含。

良好的编程习惯 14.3

在多文件程序中,一般将函数定义放在源程序文件中,相应的函数原型放在头文件中,为了使编译程序检查头文件中的函数原型与源程序文件中的函数定义是否一致,在定义函数原型的源程序文件中也包含该头文件。

14.4 解决任务 14.1 的程序

头文件 game.h 包含符号常量定义、数据类型定义、全局变量定义和函数原型,程序如下:

```
/*   game.h   */
#define N 10                    //定义符号常量 N,最多玩 10 次
typedef enum                    //定义枚举类型 HandType 表示石头、剪子、布
{
  stone = 1, scissor, paper
} HandType;
typedef enum                    //定义枚举类型 ResultType 表示胜、负、平
{
  win, lose, tie
} ResultType;
typedef struct                  //定义结构体类型 StateType 表示一次游戏的结果
{
  HandType computer;            //计算机出的手形
  HandType player;              //玩家出的手形
  ResultType result;            //比较结果
} StateType;
```

```
                                    //空行,以下是变量定义
int count = 0;                      //定义全局变量 count 表示游戏次数并初始化为 0
StateType gameResult[N];            //定义全局数组 gameResult 存储全部游戏结果
                                    //空行,以下是函数声明
void Guide( );                      //函数声明,游戏指南
void Check( );                      //函数声明,查看战况
void Game( );                       //函数声明,一次游戏
int Secret(int low, int high);      //函数声明,产生随机数
```

源程序文件 handgame.cpp 完成主函数，为实现重复游戏，将菜单和相应函数调用放在 do-while 循环中，程序如下：

```
/*   handgame.cpp   */
#include <stdio.h>    //使用库函数 printf 和 scanf
#include "game.h"     //使用 guide、game、check 等函数声明
                      //空行，以下为主函数
int main( )
{
  int select;
  do                  //do-while 循环实现重复游戏,以及查看游戏指南或战况
  {
    printf("1.游戏指南 2.开始游戏 3.查看战况 0.退出游戏\n");
    printf("请输入:");
    scanf("%d", &select);
    switch (select)
    {
      case 1: Guide( ); break;     //函数调用
      case 2: Game( ); break;      //函数调用
      case 3: Check( ); break;     //函数调用
      default: printf("游戏结束!\n"); break;
    }
  } while (select == 1 || select == 2 ||select == 3);
  return 0;                        //将 0 返回操作系统，表明程序正常结束
}
```

源程序文件 guide.cpp 包含函数 Guide 的函数定义，该函数用于实现输出游戏指南，程序如下：

```
/*   guide.cpp   */
#include <stdio.h>                 //使用库函数 printf
void Guide( )                      //函数定义,没有参数和返回值
{                                  //以下三条语句均是输出语句
  printf("握紧拳头表示石头,伸出食指和中指表示剪子,伸出五指表示布.\n");
  printf("石头会硌坏剪子、剪子能剪断布、布能包住石头.\n");
  printf("能够重复游戏、阅读游戏指南、查看当前战况等.\n");
  return;                          //结束函数 guide 的执行
}
```

源程序文件 random.cpp 包含函数 Secret 的函数定义，该函数产生 low～high 之间的

随机数，程序如下：

```
/*    random.cpp   */
#include <stdio.h>                          //使用符号常量 NULL
#include <stdlib.h>                         //使用库函数 srand 和 rand
#include <time.h>                           //使用库函数 time
int Secret(int low, int high) //函数定义,生成[low, high]之间的随机数
{
  srand(time(NULL));                        //用当前系统时间初始化随机种子
  return (low + rand()%(high - low + 1)); //返回随机数
}
```

源程序文件 game.cpp 包含函数 Game 的函数定义，该函数实现一次游戏，在比较 handComputer 和 handPlayer 时，先进行判等运算，相当于进行 3 次比较，再用 switch 语句进行其他 6 次比较，程序如下：

```
/*   game.cpp   */
#include <stdio.h>                          //使用库函数 printf 和 scanf
#include "game.h"                           //包含用户自定义头文件

void Game( )
{
  HandType handComputer, handPlayer; //定义枚举变量
  ResultType result;                        //定义枚举变量 result
  int temp, select;
  temp = Secret(1, 3);                      //函数调用,产生[1,3]之间的随机数
  handComputer = (HandType)temp;            //将整数 temp 转换为枚举元素
  printf("请输入您的选择: 1. 石头  2. 剪子  3. 布: ");
  scanf("%d", &select);
  handPlayer = (HandType)select;            //将整数 select 转换为枚举元素
  if (handComputer == handPlayer)           //包含 3 种情况
     result = tie;
  else
  {
    switch (handPlayer)                     //判断其他 6 种情况
    {
        case stone: if (handComputer == scissor) result = win;
                    else result = lose; //handComputer 一定不是 stone
                    break;
        case scissor: if (handComputer == paper) result = win;
                    else result = lose;//handComputer 一定不是 scissor
                    break;
        case paper: if (handComputer == stone) result = win;
                    else result = lose; //handComputer 一定不是 paper
                     break;
        default: result = lose; break; //如果玩家出错手形,则玩家输
    }
  }
  if (result == win)
```

```
        printf("你赢了!\n");
    else if (result == lose)
            printf("你输了!\n");
        else
            printf("平局!\n");
    gameResult[count].computer = handComputer;  //保存游戏结果
    gameResult[count].player = handPlayer;
    gameResult[count].result = result;
    count++;                                    //游戏次数加 1
    return;                                     //结束函数 game 的执行
}
```

源程序文件 check.cpp 包含函数 Check 的函数定义，该函数实现输出当前战况，由于 VC++不能直接输出枚举变量的值，因此，需要调用函数 OutResult1 和 OutResult2 分别输出枚举变量 hand 和 result 的值。程序如下：

```
/*   check.cpp   */
#include <stdio.h>                          //使用库函数 printf
#include "game.h"                           //包含用户自定义头文件
void OutResult1(HandType hand);             //函数声明,只在本程序使用
void OutResult2(ResultType result);         //函数声明,只在本程序使用
                                            //空行,以下为主函数
void Check( )                               //函数定义,没有参数和返回值
{
  printf("计算机\t 游戏者\t 结果\n");        //输出表头
  for (int i = 0; i < count; i++)           //依次输出每一次的游戏结果
  {
   OutResult1(gameResult[i].computer);      //函数调用,输出枚举元素
   OutResult1(gameResult[i].player);        //函数调用,输出枚举元素
   OutResult2(gameResult[i].result);        //函数调用,输出枚举元素
  }
  printf("\n");                             //输出换行符
  return;                                   //结束函数 check 的执行
}
void OutResult1(HandType hand)
{                           //函数定义,输出枚举类型 HandType 的枚举元素
  switch (hand)
  {
    case stone: printf("石头\t"); break;
    case scissor: printf("剪子\t"); break;
    case paper: printf("布\t"); break;
  }
  return;                   //结束函数 OutResult1 的执行
}
void OutResult2(ResultType result)
{                           //函数定义,输出枚举类型 ResultType 的枚举元素
  switch (result)
  {
    case win: printf("胜\n"); break;
```

```
34        case lose: printf("负\n"); break;
35        case tie: printf("平\n"); break;
36      }
37      return;                                    //结束函数 OutResult2 的执行
38  }
```

习 题 14

一、简答题

1. 如何将源程序文件分解为多个程序文件模块？
2. 在 VC++编程环境中，工程的主要功能是什么？
3. 外部变量和全局变量有什么区别？
4. 在一个源程序文件中，如何引用其他源程序文件定义的函数？
5. 条件编译的作用是什么？什么情况下使用条件编译？

二、程序设计题

1. 编制一个简单的四则计算器，要求用多文件实现。

2. 给定一个单词转换对照表 Table，将文本 A 进行转换形成文本 B，转换规则是：把文本 A 中所有在 Table 表中出现的单词进行相应替换。

3. 用扑克摆十二月的游戏。一副扑克牌共有 48 张，其中花色有梅花、方块、红桃和黑桃，点数有 A、2、3、4、5、6、7、8、9、10、J、Q，洗牌后将它们随机摆成 12 摞，每摞 4 张。从第一摞的最下边取出一张牌并翻开，该牌的点数是几就把它放在第几摞的上边，然后从该摞的最下边取出一张牌并翻开，该牌的点数是几就把它放在第几摞的上边，以此类推，直到第一摞的 4 张牌全部翻开为止。若 48 张牌全部翻开并归位，则游戏成功，否则游戏失败。

第15章

基本的算法设计技术

算法是问题的解决方案，这个解决方案本身并不是问题的答案，而是能获得答案的指令序列。不言而喻，由于实际问题千奇百怪，问题求解的方法千变万化，所以，算法的设计过程是一个灵活的充满智慧的过程。对于计算机专业的学生，学会读懂算法、设计算法，应该是一项最基本的要求，而发明算法则是计算机学者的最高境界。

算法设计技术（也称算法设计策略）是设计算法的一般性方法，本章讨论基本的算法设计技术，更深入的内容将在“算法设计与分析”课程中介绍。

15.1 蛮 力 法

15.1.1 设计思想

蛮力法采用一定的策略依次处理待求解问题的所有数据，从而找出问题的解。蛮力法是一种简单直接地解决问题的方法，常常直接基于问题的描述，所以，蛮力法也是最容易应用的方法。

例 15.1 设计程序计算 a^n 的值。要求用蛮力法。

解：由于 $a^n = a * a * \cdots * a$，因此，最直接最简单的想法就是设变量 result 保存计算结果并初始化为 1，然后将 result 和 a 相乘 n 次。程序如下：

```
1   /*   example15-1.cpp   */
2   #include <stdio.h>                    //使用库函数 printf 和 scanf
3                                         //空行,以下是主函数
4   int main( )
5   {
6     int a, n;
7     long int result = 1; //result 存储计算结果,a^n 可能会超出 int 的表示范围
8     printf("请输入 a 的值和 n 的值: ");  //输出提示信息
9     scanf("%d%d", &a, &n);              //从键盘输入两个整数赋给变量 a 和 n
10    for (int i = 1; i <= n; i++)            //把 result 和 a 相乘 n 次
11        result = result * a;
12    printf("%d 的%d 次方等于%d\n", a, n, result);  //输出结果
```

```
    return 0;                                    //将 0 返回操作系统,表明程序正常结束
}
```

运行结果如下（下划线为用户输入）：

```
请输入 a 的值和 n 的值：2 10
2 的 10 次方等于 1024
```

虽然巧妙和高效的算法很少来自于蛮力法，但基于以下原因，蛮力法也是一种重要的算法设计技术：

① 理论上，蛮力法可以解决可计算领域的各种问题。

② 蛮力法可以用来解决非常基本的问题，例如，求一个序列中的最大元素，计算 n 个数的和等。

③ 蛮力法可以用来解决较小规模的问题。如果要解决的问题规模不大，用蛮力法设计的算法其速度是可以接受的，就没有必要花费很大的代价设计更高效的算法。

15.1.2 程序设计实例——简单选择排序

【问题】 将 n 个元素的无序序列调整为有序序列。

【想法】 假设将待排序元素进行升序排列，简单选择排序的基本思想是：将整个序列划分为有序区和无序区，初始时有序区为空，无序区含有所有元素；在无序区中找到值最小的元素，将它与无序区中的第一个元素交换；不断重复上述过程，直到无序区只剩下一个元素。图 15.1 给出了一个简单选择排序的过程示例（方括号内是无序区）。

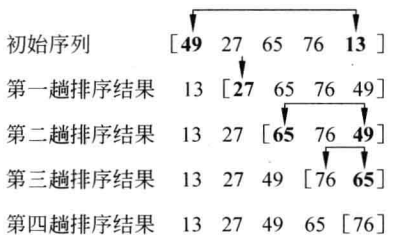


图 15.1 简单选择排序的过程示例

【算法】 设函数 SelectSort 实现对无序序列 r[n]进行简单选择排序，其算法描述如下：

输入：待排序序列 r[n]

功能：简单选择排序

输出：升序序列 r[n]

伪代码

```
step1: 循环变量 i 为 0~n-2,重复执行下述操作:
    step1.1: 在序列 r[i]~r[n-1]中查找最小值元素 r[index];
    step1.2: 如果 r[index]不是 r[i],则将 r[index]与 r[i]交换;
    step1.3: i++;
step2: 输出 r[n];
```

【程序】 由于一维数组作为参数，形参数组和实参数组共用同一段内存单元，对形

参数组的修改即是对实参数组的修改，因此，为了观察排序的结果，在调用函数 SelectSort 之前和之后分别输出数组 r[N]的值。程序如下：

```
/*   简单选择排序.cpp   */
#include <stdio.h>                              //使用库函数 printf
#define N 5                                     //定义符号常量 N
void SelectSort(int r[ ], int n);
                                                //空行,以下是主函数
int main( )
{
  int i, r[N] = {49, 27, 65, 76, 13};     //定义并初始化数组 r[5]
  printf("排序前的序列是: ");
  for (i = 0; i < N; i++)
     printf("%5d", r[i]);                 //输出元素 r[i]的值
  printf("\n");                           //输出换行符
  SelectSort(r, N);                       //函数调用,实参 r 是数组 r[5]的首地址
  printf("排序后的序列是: ");
  for (i = 0; i < N; i++)
      printf("%5d", r[i]);                //输出元素 r[i]的值
  printf("\n");
  return 0;                               //将 0 返回操作系统,表明程序正常结束
}
                                          //空行，以下是其他函数定义
void SelectSort(int r[ ], int n) //函数定义,一维数组作为形参
{
  int i, j, index, temp;
  for (i = 0; i < n - 1; i++)     //对 n 个记录进行 n-1 趟简单选择排序
  {
     index = i;                           //假定 r[index]为最小值元素
     for (j = i + 1; j < n; j++)  //在无序区中查找最小值元素
         if (r[j] < r[index])
             index = j;                   //r[index]为当前最小值元素
     if (index != i)                      //如果最小值元素不是 r[i]
     {                          //以下三条语句实现交换元素 r[i]和 r[index]
       temp = r[i]; r[i] = r[index]; r[index] = temp;
     }                                    //结束 if 语句
  }                                       //结束 for 循环
  return;                                 //结束函数
}
```

运行结果如下：

```
排序前的序列是: 49   27   65   76   13
排序后的序列是: 13   27   49   65   76
```

15.2 穷 举 法

15.2.1 设计思想

穷举法本质上属于蛮力法，是将所有可能的解都列举出来，依次试探这些解是否满足特定的条件或要求。前面的程序设计实例“鸡兔同笼问题”、“百元买百鸡问题”等都属于穷举法。穷举法通常需要解决以下两个关键问题：

① 分析问题的解可能存在的范围，找出穷举范围；

② 分析问题的解所满足的条件，找出约束条件并用逻辑表达式表示。

例 15.2 设计程序求解如图 15.2 所示的数字谜。

解：将 A、B、C 依次试探每一个可能解，A 的穷举范围是 1～9，B 的穷举范围是 2～9（因为乘法结果有进位，则 B 至少是 2），C 的穷举范围是 0～9，约束条件是满足等式关系，程序如下：

```
  ABB
×   B
-----
 ACBC
```

图 15.2 数字谜

```
1   /*   example15-2.cpp   */
2   #include <stdio.h>                          //使用库函数 printf
3                                               //空行，以下是主函数
4   int main( )
5   {
6     int A, B, C, temp1, temp2;
7     for (A = 1; A <= 9; A++)                  //将 A 从 1 到 9 依次试探
8       for (B = 2; B <= 9; B++)                //将 B 从 2 到 9 依次试探
9         for (C = 0; C <= 9; C++)              //将 C 从 0 到 9 依次试探
10        {
11          temp1 = A * 100 + B * 10 + B;       //计算 ABB 的值
12          temp2 = A * 1000 + C * 100 + B * 10 + C;   //计算 ACBC 的值
13          if (temp1 * B == temp2)             //如果 ABB*B 等于 ACBC
14          {
15            printf("A = %d, B = %d, C = %d\n", A, B, C);   //输出结果
16            printf("%d × %d = %d\n", temp1, B, temp2);
17            return 0;                 //结束程序的执行,并将状态码 0 返回操作系统
18          }                           //结束 if 语句
19        }                             //结束内层 for 循环
20    printf("无解\n");
21    return 0;                         //将 0 返回操作系统,表明程序正常结束
22  }
```

运行结果如下：

```
A = 7, B = 9, C =1
799 × 9 = 7191
```

15.2.2 程序设计实例——假币问题

【问题】 在 12 枚外观相同的硬币中，有 1 枚是假币，并且已知假币与真币的重量不同，但不知道假币与真币相比较轻还是较重。可以通过一架天平来任意比较两组硬币，最多称量 3 次就能找出假币，例如，下面是一种称量方案：

```
3
ABCD   EFGH   =
ABCI    EFJK   <
ABIJ    EFGH   =
```

其中，A、B、C、D、E、F、G、H、I、J、K 和 L 分别表示 12 枚硬币，第 1 行表示称量次数，然后是每次的称量结果，称量结果由空格隔开三个字符串，分别表示天平左端放置的硬币、天平右端放置的硬币和天平状态，“>”、“<”和“=”分别表示天平左端高、右端高和平衡，请设计程序找出这枚假币。

【想法】 设 index 表示假币的编号，weight 表示假币与真币重量的比较结果，weight = 0 表示假币比真币轻，weight = 1 表示假币比真币重，则 index 有 n 种猜测，weight 有 2 种猜测，且称量结果应该满足下列条件：

① 如果天平两端没有假币 index，则天平应该是平衡的；

② 如果 weight 等于 0，则称量结果为“>”的右端应该有假币 index，或称量结果为“<”的左端应该有假币 index；

③ 如果 weight 等于 1，则称量结果为“>”的左端应该有假币 index，或称量结果为“<”的右端应该有假币 index。

【算法】 设函数 FakeCoinLight 判断某枚硬币是否是比真币轻的假币，算法描述如下：

输入：称量方案，硬币的编号 index

功能：判断某枚硬币是否是比真币轻的假币

输出：如果硬币 index 为假币，则返回真值，否则返回假值

伪代码

```
step1: 对每一次称量结果，执行下述操作：
    step1.1: 如果天平状态为“=”，则进一步判断，如果天平左端或右端有硬币 index，
             则 index 不是假币，返回假值；否则试探下一次称量结果；
    step1.2: 如果天平状态为“>”，则进一步判断，如果天平右端有硬币 index，则
             index 可能是假币，试探下一次称量结果；否则返回假值；
    step1.3: 如果天平状态为“<”，则进一步判断，如果天平左端有硬币 index，则
             index 可能是假币，试探下一次称量结果；否则返回假值；
step2: 硬币 index 符合每一次称量结果，则 index 是假币，返回真值；
```

设函数 FakeCoinHeavy 判断某枚硬币是否是比真币重的假币，具体判断过程与函数 FakeCoinLight 类似，请读者自行完成。

【程序】 为避免在调用函数时传递大量参数，设全局变量 left[3][5]、right[3][5]和result[3][2]接收从键盘输入的称量结果，主函数首先接收从键盘输入的称量结果，然后调用函数 FakeCoinLight 依次试探每一枚硬币是否为比真币轻的假币，再调用函数 FakeCoinHeavy 依次试探每一枚硬币是否为比真币重的假币。程序如下：

```
/*   假币问题.cpp   */
#include <stdio.h>                          //使用库函数 printf 和 scanf
#include <string.h>     //使用库函数 strchr 判断字符串中是否含有指定字符
char coin[12] = {'A','B','C','D','E','F','G','H','I','J','K','L'};
char left[3][5], right[3][5], result[3][2]; //3 次称量结果,全局变量
int FakeCoinLight(int times, int index); //函数声明
int FakeCoinHeavy(int times, int index); //函数声明
                                            //空行,以下是主函数
int main( )
{
  int times, i, j;                           //变量 times 表示称量次数
  printf("请输入称量的次数: ");
  scanf("%d", &times);
  for (i = 0; i < times; i++)                //依次输入每一次称量的结果
  {
    printf("请输入第%d 次称量结果:", i + 1);
    scanf("%s%s%s", left[i], right[i],result[i]);//left 表示天平左端
  }
  for (j = 0; j < 12; j++)                   //依次枚举 12 枚硬币
  {
    if (FakeCoinLight(times, j)) //函数调用,判断硬币 j 是否是较轻的假币
    {
      printf("假币是%c,且假币比真币轻\n", coin[j]);
      break;
    }
    if (FakeCoinHeavy(times,j))   //函数调用,判断硬币 j 是否是较重的假币
    {
      printf("假币是%c,且假币比真币重\n", coin[j]);
      break;
     }
  }
  if (j >= 12)                          //12 枚硬币均不是假币
     printf("称量结果有问题,无法判断\n");
  return 0;                             //将 0 返回操作系统,表明程序正常结束
}
                                        //空行,以下是其他函数定义
int FakeCoinLight(int times, int index)
{                  //函数定义,判断假币 index 较轻在 times 次称量中是否成立
  for (int i = 0; i < times; i++)                //代入每一次称量结果
  {
    switch (result[i][0])                        //判断天平的状态
    {
```

```
43          case '=':if((strchr(left[i],coin[index])!=NULL)||
                        (strchr(right[i],coin[index])!=NULL))
44                      return 0;//天平左端或右端有硬币 index,则 index 不是假币
45                  else
46                      break;      //天平两端没有硬币 index,则跳出 switch 语
句
47          case '>': if ((strchr(right[i], coin[index]) != NULL))
48                      break; //天平右端有硬币 index,则跳出 switch 语句
49                  else
50                      return 0;//天平右端没有硬币 index,则 index 不是假币
51        case '<': if ((strchr(left[i], coin[index]) != NULL))
52                      break;  //天平左端有硬币 index,则跳出 switch 语句
53                  else
54                      return 0;//天平左端没有硬币 index,则 index 不是假币
55        }
56      }
57      return 1;                   //满足所有称量结果,则 index 是较轻的假币
58    }
59    int FakeCoinHeavy(int times, int index)
60    {                   //函数定义,判断假币 index 较重在 times 次称量中是否成立
61      for (int i = 0; i < times; i++)             //代入每一次称量结果
62      {
63        switch (result[i][0])                     //判断天平的状态
64        {
65          case '=': if ((strchr(left[i],coin[index])!= NULL)||
                        (strchr(right[i], coin[index])!= NULL))
66                      return 0;//天平左端或右端有硬币 index,则 index 不是假币
67                  else
68                      break;  //天平两端没有硬币 index,则跳出 switch 语句
69          case '>': if ((strchr(left[i], coin[index])!= NULL))
70                      break;  //天平左端有硬币 index,则跳出 switch 语句
71                  else
72                      return 0; //天平左端没有硬币 index,则 index 不是假币
73          case '<': if ((strchr(right[i], coin[index])!= NULL))
74                      break;  //天平右端有硬币 index,则跳出 switch 语句
75                  else
76                      return 0;//天平右端没有硬币 index,则 index 不是假币
77        }
78      }
79      return 1;                   //满足所有称量结果,则 index 是较重的假币
80    }
```

运行结果如下（下划线为用户输入）：

```
请输入称量的次数：3
请输入第 1 次称量结果：ABCD  EFGH  =
请输入第 2 次称量结果：ABCI  EFJK  >
请输入第 3 次称量结果：ABIJ  EFGH  =
假币是 K,且假币比真币轻
```

15.3 递 推 法

15.3.1 设计思想

递推法有正推和逆推两种形式，所谓正推就是从前向后递推，已知小规模问题的解递推到大规模问题的解；所谓逆推就是从后向前递推，已知大规模问题的解递推到小规模问题的解。递推法能够将复杂的运算化解为若干重复的运算，充分发挥了计算机擅长重复处理的特点。

例 15.3 用递推法求解 Fibonacci 数列。

解：Fibonacci 数列存在如下递推式：

$$a_n=\begin{cases}1 & n=1\\ 1 & n=2\\ a_{n-1}+a_{n-2} & n>2\end{cases} \tag{15.1}$$

设变量 n3 表示第 n 个月兔子的对数，n1 表示第 $n-1$ 个月兔子的对数，n2 表示第 $n-2$ 个月兔子的对数，采用正推法，程序如下：

```
/*  example15-3.cpp    */
#include <stdio.h>                          //使用库函数 printf 和 scanf
                                            //空行,以下是主函数
int main( )
{
  int n, n1, n2, n3;
  printf("求 Fibonacci 数列,求第几项:"); //输出提示信息
  scanf("%d", &n);
  n1 = 1; n2 = 1;                           //初始化数列前两项
  for (int i = 3; i <= n; i++)              //从第 3 项开始递推
  {
    n3 = n1 + n2;                 //这个月的兔子对数等于前两个月兔子对数之和
    n2 = n1; n1 = n3;
  }
  if (n > 2)
     printf("Fibonacci 数列第%d 项是%d\n", n, n3);
  else
     printf("Fibonacci 数列第%d 项是 1\n", n);
  return 0;                                 //将 0 返回操作系统,表明程序正常结束
}
```

运行结果如下（下划线为用户输入）：

```
求 Fibonacci 数列，求第几项：5
Fibonacci 数列第 5 项是 5
```

例 15.4 用递推法求解猴子吃桃问题：一只猴子摘了很多桃子，每天吃现有桃子的一半多一个，到第 10 天时只有一个桃子，问原有桃子多少个？

解：显然，猴子吃桃问题存在如下递推式：

$$a_n=\begin{cases}1 & n=10\\ (a_{n+1}+1)\times 2 & n=9,8,7,\cdots,1\end{cases} \tag{15.2}$$

由于每天的桃子个数依赖前一天的桃子个数，设变量 peach 表示桃子的个数，采用逆推法，程序如下：

```
/*   example15-4.cpp   */
#include <stdio.h>                          //使用库函数 printf
                                            //空行,以下是主函数
int main( )
{
  int peach = 1;                            //初始化第 10 天桃子的个数
  for (int i = 9; i >= 1; i--)              //向前递推前一天桃子的个数,直到第 1 天
      peach = (peach + 1) * 2;              //递推关系式
  printf("原有%d 个桃子\n", peach);
  return 0;                                 //将 0 返回操作系统,表明程序正常结束
}
```

运行结果如下：

```
原有 1534 个桃子
```

15.3.2 程序设计实例——捕鱼知多少

【问题】 A、B、C、D、E 五个人合伙夜间捕鱼，上岸时都疲惫不堪，各自在湖边的树丛中找地方睡觉了。清晨，A 第一个醒来，将鱼分成 5 份，把多余的一条扔回湖中，拿自己的一份回家了；B 第二个醒来，也将鱼分成 5 份，扔掉多余的一条鱼，拿自己的一份回家了；接着，C、D、E 依次醒来，也都按同样的办法分鱼。问：5 个人至少共捕到多少条鱼？每个人醒来后看到多少条鱼？

【想法】 设 A、B、C、D、E 的编号为 0、1、2、3、4，数组 fish[5]表示 5 个人醒来后看到的鱼数，则 fish[i]（0≤i≤4）满足被 5 整除后余 1。显然，5 个人合伙捕到的鱼数即是 A 醒来后看到的鱼数，第 i（1≤i≤4）个人醒来后看到鱼数 fish[i]满足：

$$\text{fish[i]} = (\text{fish[i-1]}-1) / 5 * 4 \quad (\text{i} = 1, 2, 3, 4) \tag{15.3}$$

【算法】 采用正推法，fish[0]从 1 开始，每次增 5，然后依次考查 fish[n]是否满足被 5 整除后余 1，算法描述如下：

伪代码

```
step1: 初始化 fish[0] = 1;
step2: 重复下述操作，直到 fish[n]均满足被 5 整除后余 1:
    step2.1: fish[0] = fish[0] + 5;
    step2.2: 循环变量 i 从 1 到 4，重复执行下述操作:
        step2.2.1: fish[i] = (fish[i - 1] - 1) / 5 * 4;
        step2.2.2: 如果 fish[i]不满足被 5 整除后余 1，则转 step2.1;
        step2.2.3: i++;
step3: 依次输出数组 fish[n];
```

【程序】 设字符数组 man[5]表示 A、B、C、D、E 五个人，程序如下：

```
/*   捕鱼知多少.cpp   */
#include <stdio.h>                    //使用库函数 printf
                                      //空行,以下是主函数
int main( )
{
  char man[5] = {'A', 'B', 'C', 'D', 'E'};//数组 man[5]表示 5 个人
  int fish[5], i;
  fish[0] = 1;                        //假定 A 看到的鱼数是 1
  do
  {
    fish[0] = fish[0] + 5;            //fish[0]从 1 开始,每次加 5,保证除以 5 余 1
    for (i = 1; i < 5; i++)           //递推其他人看到的鱼数
    {
      fish[i] = (fish[i - 1] - 1)/5 * 4;
      if (fish[i] % 5 != 1)           //如果不满足被 5 整除后余 1,跳出 for 循环
        break;
    }
  } while (i < 5);                    //如果是提前跳出 for 循环,则执行 do-while 循环
  for (i = 0; i < 5; i++)             //输出结果
      printf("%c 看到的鱼数是%d\n", man[i], fish[i]);
  return 0;                           //将 0 返回操作系统,表明程序正常结束
}
```

运行结果如下：

```
A 看到的鱼数是 3121
B 看到的鱼数是 2496
C 看到的鱼数是 1996
D 看到的鱼数是 1596
E 看到的鱼数是 1276
```

15.4 分 治 法

15.4.1 设计思想

分治者，分而治之也。分治法的设计思想是将一个难以直接解决的大问题，划分成一些规模较小的子问题，以便各个击破，分而治之。由分治法产生的子问题往往是原问题的较小模式，反复应用分治手段，可以使子问题与原问题类型一致而其规模却不断缩小，最终使子问题缩小到很容易直接求解，这自然导致递归过程的产生。分治与递归就像一对孪生兄弟，经常同时应用在算法设计之中，并由此产生许多高效的算法。

一般来说，分治法的求解过程由以下三个阶段组成：

① 划分：把规模为 n 的原问题划分为 k 个规模较小的子问题。

② 求解子问题：递归求解各个子问题，有时递归处理也可以用循环来实现。

③ 合并：合并各个子问题的解。

图 15.3 所示是分治法的典型情况。

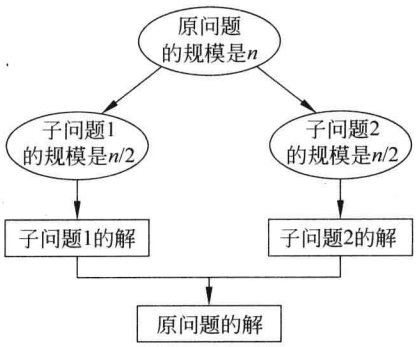


图 15.3　分治法的典型情况

例 15.5　设计程序计算 a^n 的值。要求用分治法。

解：当 $n>1$ 时，可以把该问题分解为两个子问题：计算前 $\lfloor n/2 \rfloor$（向下取整）个 a 的乘积和后 $\lceil n/2 \rceil$（向上取整）个 a 的乘积，再把这两个乘积相乘得到原问题的解。如果 $n=1$，可以简单地返回 a 的值。程序如下：

```
1   /*   example15-5.cpp   */
2   #include <stdio.h>                    //使用库函数 printf 和 scanf
3   long int Muiltple(int a, int n); //函数声明
4                                         //空行,以下是主函数
5   int main( )
6   {
7     int a, n;
8     long int result = 1; //result 存储计算结果,a^n 可能会超出 int 的表示范围
9     printf("请输入 a 的值和 n 的值:");   //输出提示信息
10    scanf("%d%d", &a, &n);
11    result = Muiltple(a, n);       //函数调用,函数的返回值赋给变量 result
12    printf("%d 的%d 次方等于%d\n", a, n, result);
13    return 0;                           //将 0 返回操作系统,表明程序正常结束
14  }
15                                        //空行,以下是其他函数定义
16  long int Muiltple(int a, int n)  //函数定义,返回值的类型是长整型
17  {
18    int mid1, mid2;
19    if (n == 1)                         //递归的边界条件,当 n 等于 1 时
20        return a;                       //结束递归,直接返回 a 的值
21    mid1 = n / 2;                       //分治,计算区间的前一半需要乘多少个 a
```

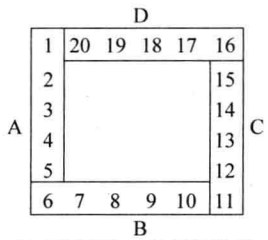
```
    if (n % 2 != 0)                          //计算区间的后一半需要乘多少个 a
        mid2 = mid1 + 1;                     //n 是奇数,则后一半比前一半多一个 a
    else
        mid2 = mid1;                         //n 是偶数,则后一半和前一半 a 的个数相同
    return (Muiltple(a, mid1) * Muiltple(a, mid2));     //递归函数调用
}
```

15.4.2 程序设计实例——数字旋转方阵

【问题】 输出如图 15.4 所示 $N\times N$（$1\leqslant N\leqslant 10$）的数字旋转方阵。

1	20	19	18	17	16
2	21	32	31	30	15
3	22	33	36	29	14
4	23	34	35	28	13
5	24	25	26	27	12
6	7	8	9	10	11

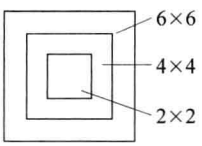
(a) 6×6的旋转方阵



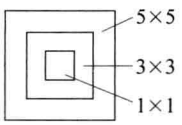
(b) 逐层填数，先填最外层

图 15.4 数字旋转方阵示例

【想法】 用二维数组 data[N][N]表示 N×N 的方阵，观察方阵中数字的规律，可以从外层向里层填数，如图 15.4(b)所示。在填数过程中，每一层的起始位置很重要。设变量 size 表示方阵的大小，则初始时 size = N，填完一层则 size = size – 2；设变量 begin 表示每一层的起始位置，变量 i 和 j 分别表示行号和列号，则每一层初始时 i = begin，j = begin。将每一层的填数过程分为 A、B、C、D 四个区域，每个区域需要填写 size – 1 个数字，且填写区域 A 时列号不变行号加 1，填写区域 B 时行号不变列号加 1，填写区域 C 时列号不变行号减 1，填写区域 D 时行号不变列号减 1。递归的边界条件是 size 等于 0 或 size 等于 1，如图 15.5 所示。



(a) N为偶数，则size等于0
结束递归



(b) N为奇数，则size等于1
结束递归

图 15.5 递归的边界条件

【算法】 设递归函数 Full 实现填数过程，其算法描述如下：

输入：当前层左上角要填的数字 number，左上角的坐标 begin，方阵的阶数 size

功能：填写数字旋转方阵

输出：无

伪代码

```
step1: 如果 size 等于 0，则算法结束;
step2: 如果 size 等于 1，则当前层只有一个位置 data[begin][begin] = number;
step3: 初始化行、列下标 i = begin, j = begin;
step4: 重复下述操作 size - 1 次，填写区域 A:
    step4.1: data[i][j] = number; number++;
    step4.2: 行下标 i++; 列下标不变;
step5: 重复下述操作 size - 1 次，填写区域 B:
    step5.1: data[i][j] = number; number++;
    step5.2: 行下标不变;列下标 j++;
step6: 重复下述操作 size - 1 次，填写区域 C:
    step6.1: data[i][j] = number; number++;
    step6.2: 行下标 i--; 列下标不变;
step7: 重复下述操作 size - 1 次，填写区域 D:
    step7.1: data[i][j] = number; number++;
    step7.2: 行下标不变，列下标 j--;
step8: 递归调用函数 Full 实现在阶数为 size -2 的方阵中左上角 begin + 1 处从数
       字 number 开始填数;
```

【程序】 由于递归函数 Full 在调用过程中需要对同一个数组 data[N][N]填数，为避免传递参数，将数组 data[N][N]设为全局变量，程序如下：

```
1   /*  数字旋转方阵.cpp  */
2   #include <stdio.h>                  //使用库函数 printf 和 scanf
3   #define N 10                        //定义符号常量 N,用于定义数组
4   int data[N][N] = {0};               //定义全局数组 data[N][N]并初始化为 0
5   void Full(int number, int begin, int size); //函数声明,填写旋转方阵
6   void OutPrint(int size);            //函数声明,输出旋转方阵
7                                       //空行,以下是主函数
8   int main( )
9   {
10    int n;
11    printf("请输入方阵的阶数(小于 10):"); //输出提示信息
12    scanf("%d", &n);
13    Full(1, 0, n);    //函数调用,从 1 开始填写 n 阶方阵,左上角的下标为(0, 0)
14    OutPrint(n);                  //函数调用,输出 n 阶方阵
15    return 0;                     //将 0 返回操作系统,表明程序正常结束
16  }
17                                  //空行,以下是其他函数定义
18  void Full(int number, int begin, int size)
19  { //函数定义,从 number 开始填写 size 阶方阵,左上角的下标为(begin, begin)
20    int i, j, k;
21    if (size == 0)              //递归的边界条件,如果 size 等于 0,则无须填写
22        return;
23    if (size == 1)              //递归的边界条件,如果 size 等于 1
24    {
25      data[begin][begin] = number; //则只须填写 number
26      return;
```

```
27      }
28      i = begin; j = begin;                  //初始化左上角下标
29      for (k = 0; k < size - 1; k++) //填写区域A,共填写 size-1 个数
30      {
31        data[i][j] = number; number++;//在当前位置填写 number
32        i++;                                 //行下标加 1
33      }
34      for (k = 0; k < size - 1; k++) //填写区域 B,共填写 size-1 个数
35      {
36        data[i][j] = number; number++; //在当前位置填写 number
37        j++;                                 //列下标加 1
38      }
39      for (k = 0; k < size - 1; k++)   //填写区域 C,共填写 size-1 个数
40      {
41        data[i][j] = number; number++; //在当前位置填写 number
42        i--;                                 //行下标减 1
43      }
44      for (k = 0; k < size - 1; k++)   //填写区域 D,共填写 size-1 个数
45      {
46        data[i][j] = number; number++; //在当前位置填写 number
47        j--;                                 //列下标减 1
48      }
49      Full(number, begin + 1, size - 2);//递归调用,左上角下标为 begin + 1
50      return;                                //结束函数 Full 的执行
51    }
52    void OutPrint(int size)                //函数调用,输出 size 阶方阵
53    {
54      int i, j;
55      for (i = 0; i < size; i++)           //输出第 i 行
56      {
57        for (j = 0; j < size; j++)         //输出第 j 列
58            printf("%4d", data[i][j]);
59        printf("\n");                      //输出一行后输出换行符
60      }
61      return;                              //结束函数 OutPrint 的执行
62    }
```

运行结果如下（下划线为用户输入）：

```
请输入方阵的阶数(小于 10): 6
1  20  19  18  17  16
2  21  32  31  30  15
3  22  33  36  29  14
4  23  34  35  28  13
5  24  25  26  27  12
6   7   8   9  10  11
```

15.5 动态规划法

15.5.1 设计思想

动态规划法也是将待求解问题分解成若干个子问题，但是子问题之间往往不是相互独立的，如果用分治法求解，这些子问题的重叠部分被重复计算多次。一般来说，子问题的重叠关系表现在给定问题满足的递推关系式（称为动态规划函数）中，动态规划法将每个子问题求解一次并将其解保存在一个表格中，当需要再次解此子问题时，只是简单地通过查表获得该子问题的解，从而避免了大量重复计算。动态规划法的一般过程如图 15.6 所示。

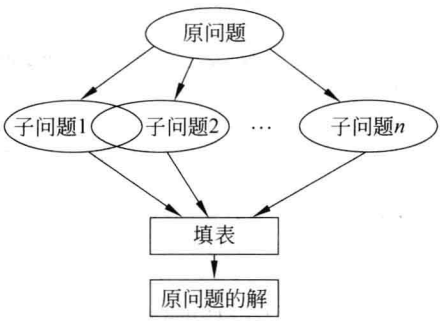


图 15.6 动态规划法的求解过程

例 15.6 求解 Fibonacci 数列。要求用动态规划法。

解：注意计算 $F(n)$是以计算它的两个重叠子问题 $F(n-1)$和 $F(n-2)$的形式来表达的，可以利用一维数组 fib[n+1]填入 $F(n)$的值（补充定义 $F(0)=0$），如图 15.7 所示。

0	1	2	3	4	5	6	7	8	9
0	1	1	2	3	5	8	13	21	34

图 15.7 动态规划法求解斐波那契数列的填表过程

开始时，根据递推式的初始条件可以直接填入 0 和 1，然后根据递推式填写其他元素，显然，数组中最后一项就是 $F(n)$的值。程序如下：

```
/*   example15-6.cpp   */
#include <stdio.h>                          //使用库函数 printf 和 scanf
#define N 50                                //定义符号常量 N,最多求第 50 项
                                            //空行,以下是主函数
int main( )
{
  int fib[N + 1] = {0, 1};                  //定义数组,并初始化前两项
  int n, i;
  printf("请输入要求解的项数: ");           //输出提示信息
  scanf("%d", &n);
  for (i = 2; i <= n; i++)                  //依次求解子问题并填表
     fib[i] = fib[i - 1] + fib[i - 2];
  for (i = 1; i <= n; i++)                  //输出 Fibonacci 数列前 n 项
     printf("%5d", fib[i]);
  printf("\n");                             //输出换行符
  return 0;                                 //将 0 返回操作系统,表明程序正常结束
}
```

运行结果如下（下划线为用户输入）：

```
请输入要求解的项数：10
   1    1    2    3    5    8   13   21   34   55
```

15.5.2 程序设计实例——0/1 背包问题

【问题】 给定 n 个物品和一个背包，物品 i 的重量是 w_i，其价值为 v_i，背包的容量为 C。背包问题是如何选择装入背包的物品，使得装入背包中物品的总价值最大。如果在选择装入背包的物品时，对每个物品只有两种选择：装入背包或不装入背包，即不能将某个物品装入背包多次，也不能只装入一部分，则称为 0/1 背包问题。

【想法】 将 0/1 背包问题的解表示为$(x_1, x_2, \cdots, x_n)$，$x_i = [0, 1]$。假设已确定$(x_1, \cdots, x_{i-1})$，则在决策 x_i 时，问题处于下列两种状态之一：

① 背包容量不足以装入物品 i，则 $x_i = 0$，背包不增加价值；

② 背包容量可以装入物品 i，则 $x_i = 1$，背包的价值增加了 v_i。

令 $V(i, j)$表示把前 $i(1 \leqslant i \leqslant n)$个物品装入容量为$j(1 \leqslant j \leqslant C)$的背包中获得的最大价值，则可以得到如下动态规划函数：

$$V(i,j)=\begin{cases} V(i-1,j) & j < w_i \\ \max\{V(i-1,j), V(i-1,j-w_i)+v_i\} & j > w_i \end{cases} \tag{15.4}$$

式（15.4）的第一个式子表明，如果第 i 个物品的重量大于背包的容量 j，则物品 i 不能装入背包，背包不增加价值。第二个式子表明，如果第 i 个物品的重量小于背包的容量 j，则会有以下两种情况：①如果把第 i 个物品装入背包，则背包中物品的价值等于把前 i-1 个物品装入容量为 j-w_i 的背包获得的价值加上第 i 个物品的价值 v_i；②如果第 i 个物品没有装入背包，则背包中物品的价值等于把前 i-1 个物品装入容量为 j 的背包获得的价值。显然，这两种情况下背包获得价值的较大者是对 x_i 决策后背包获得的最大价值。

考虑边界情况，把 i 个物品装入容量为 0 的背包和把 0 个物品装入容量为 j 的背包，获得的价值均为 0，因此，有如下边界条件成立：

$$V(i,\ 0) = V(0,\ j) = 0 \tag{15.5}$$

例如，有 5 个物品，其重量分别是{2, 2, 6, 5, 4}，价值分别为{6, 3, 5, 4, 6}，背包的容量为 10，求背包获得的最大价值。

根据动态规划函数，设二维数组 $V[n+1][C+1]$存储子问题的解，$V[i][j]$表示把前 i 个物品装入容量为 j 的背包中获得的最大价值，根据式（15.5）把数组的第 0 行和第 0 列初始化为 0，然后一行一行地计算 $V[i][j]$，如图 15.8 所示，背包获得的最大价值是 15。

【算法】 设函数 KnapSack 完成 0/1 背包问题，其算法描述如下：

输入：n 个物品的重量 w[n]和价值 v[n]，背包容量 C

功能：0/1 背包问题

输出：获得的最大价值 maxValue

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	6	6	6	6	6	6	6	6	6
2	0	0	6	6	9	9	9	9	9	9	9
3	0	0	6	6	9	9	9	9	11	11	14
4	0	0	6	6	9	9	9	10	11	13	14
5	0	0	6	6	9	9	12	12	15	15	**15**

图 15.8　0/1 背包求解（填表）过程

伪代码

```
step1: 将数组 V[n + 1][C + 1]的第 0 行初始化为 0;
step2: 将数组 V[n + 1][C + 1]的第 0 列初始化为 0;
step3: 循环变量 i 从 1 到 n 填写第 i 行:
    step3.1: 循环变量 j 从 1 到 C 填写第 j 列:
        step3.1.1:如果 j 小于物品 i 的重量 w[i - 1],则 V[i][j] = V[i - 1][j];
        step3.1.2:否则,V[i][j] = max(V[i-1][j], V[i-1][j- w[i-1]]+v[i-1]);
        step3.1.3: j++;
    step3.2: i++;
```

【程序】　为避免传递参数，将数组 V[N + 1][C + 1]设为全局变量，程序如下：

```
/*    0-1 背包问题.cpp   */
#include <stdio.h>                          //使用库函数 printf 和 scanf
#define N 10                                //定义符号常量 N,最多 10 个物品
#define C 100                               //定义符号常量 C,背包容量最多为 10
int V[N + 1][C + 1];                        //定义全局数组,避免参数传递
int KnapSack(int n, int w[ ], int v[ ], int C);  //函数声明
                                            //空行,以下是主函数
int main( )
{
  int w[N], v[N], C, maxValue;   //w[N]和 v[N]分别存储物品的重量和价值
  int n, i;
  printf("请输入物品的个数: ");          //输出提示信息
  scanf("%d", &n);
  printf("请输入物品的重量: ");          //输出提示信息
  for (i = 0; i < n; i++)
    scanf("%d", &w[i]);
  printf("请输入物品的价值: ");          //输出提示信息
  for (i = 0; i < n; i++)
    scanf("%d", &v[i]);
  printf("请输入背包的容量: ");          //输出提示信息
  scanf("%d", &C);
  maxValue = KnapSack(n, w, v, C); //函数调用,返回值赋给变量 maxValue
  printf("获得最大价值为: %d\n", maxValue);
  return 0;                                 //将 0 返回操作系统,表明程序正常结束
}
                                            //空行,以下是其他函数定义
int KnapSack(int n, int w[ ], int v[ ], int C)
                                      //函数定义,求解 0/1 背包问题
```

```
{
  int i, j;
  for (j = 0; j <= C; j++)          //初始化第 0 行
    V[0][j]=0;
  for (i = 0; i <= n; i++)          //初始化第 0 列
    V[i][0] = 0;
  for (i = 1; i <= n; i++)          //计算第 i 行,进行第 i 次迭代
    for (j = 1; j <= C; j++)        //计算第 j 列
    {
      if (j < w[i - 1])             //注意物品 i 的重量存放在 w[i-1]中
         V[i][j] = V[i - 1][j];
      else if (V[i - 1][j] > V[i - 1][j - w[i - 1]] + v[i - 1])
         V[i][j] = V[i - 1][j];
        else V[i][j] = V[i - 1][j - w[i - 1]] + v[i - 1];
    }
  return V[n][C];                   //返回背包获得的最大价值
}
```

运行结果如下（下划线为用户输入）：

```
请输入物品的个数：5
请输入物品的重量：2 2 6 5 4
请输入物品的价值：6 3 5 4 6
请输入背包的容量：10
获得最大价值为：15
```

15.6 贪 心 法

15.6.1 设计思想

正如其名字一样，贪心法在解决问题的策略上目光短浅，只根据当前已有的信息做出选择，而且一旦做出了选择，不管将来有什么结果，这个选择都不会改变。换言之，贪心法并不是从整体最优考虑，它所做出的选择只是在某种意义上的局部最优。这种局部最优选择不能保证得到整体**最优解**，但通常能得到**近似最优解**。

例 15.7 用贪心法求解付款问题：假设有面值为 5 元、2 元、1 元、5 角、2 角、1 角的货币若干张，需要付款 4 元 6 角现金，如何付款才能使付出货币的数量最少？

解：付款问题的贪心选择策略是，在不超过应付款金额的条件下，选择面值最大的货币，首先选出 1 张面值不超过 4 元 6 角的最大面值的货币，即 2 元，再选出 1 张面值不超过 2 元 6 角的最大面值的货币，即 2 元，再选出 1 张面值不超过 6 角的最大面值的货币，即 5 角，再选出 1 张面值不超过 1 角的最大面值的货币，即 1 角，总共付出 4 张货币。程序如下：

```
/*   付款问题.cpp   */
#include <stdio.h>                  //使用库函数 printf 和 scanf
```

```
#define N 6                                    //定义符号常量N,有N种货币
const int money[N]={50,20,10,5, 2,1};  //为避免实数运算误差,采用整数
                                               //空行,以下是主函数
int main( )
{
  double temp;
  int sum, count = 0;          //sum存储应付款,count存储付出的货币张数
  printf("请输入应付款额: ");
  scanf("%lf", &temp);
  sum = (int)(temp * 10);      //为避免实数运算误差,扩大10倍转换为整数
  do
  {
    for (int i = 0; i < N; i++)   //依次试探,选取不超过sum的最大面值
    {
      if (sum >= money[i])
      {
        count++;
        printf("第%d张的面值为%5.2f\n", count, (double)money[i]/10);
        sum = sum - money[i];       //sum为目前应付款
        break;
      }
    }
  } while (sum > 0);                //如果还有应付款,则继续执行循环
  return 0;                         //将0返回操作系统,表明程序正常结束
}
```

运行结果如下（下划线为用户输入）：

```
请输入应付款额: 4.6
第 1 张的面值为 2.00
第 2 张的面值为 2.00
第 3 张的面值为 0.50
第 4 张的面值为 0.10
```

15.6.2 程序设计实例——埃及分数

【问题】 埃及同中国一样，也是世界文明古国之一。古埃及人只用分子为 1 的分数，在表示一个真分数时，将其分解为若干个埃及分数之和，例如，7/8 表示为 1/2 + 1/3 + 1/24。设计程序把一个真分数表示为最少的埃及分数之和的形式。

【想法】 一个真分数的埃及分数表示不是唯一的，例如，7/8 又可以表示为 1/8 + 1/8 + 1/8 + 1/8 + 1/8 + 1/8 + 1/8。显然，把一个真分数表示为最少的埃及分数之和的形式，其贪心策略是逐步选择真分数包含的最大埃及分数，以 7/8 为例，7/8 > 1/2，则 1/2 是第一次贪心选择的结果；7/8 – 1/2 = 3/8 > 1/3，则 1/3 是第二次贪心选择的结果；7/8 – 1/2 – 1/3 = 1/24，则 1/24 是第三次贪心选择的结果，即 7/8 = 1/2 + 1/3 + 1/24。

接下来的问题是：如何找到真分数包含的最大埃及分数？设真分数为 A/B，B 除以

A 的整数部分为 C，余数为 D，则有下式成立：

$$B = A \times C + D$$

即：

$$B/A = C + D/A < C + 1$$

则：

$$A/B > 1/(C + 1)$$

即 1/(C + 1) 即为真分数 A/B 包含的最大埃及分数。设 E = C + 1，由于

$$A/B - 1/E = ((A \times E) - B)/(B \times E)$$

则真分数减去最大埃及分数后，得到真分数((A×E) – B)/B×E，该真分数可能存在公因子，需要化简，可以将分子和分母同时除以最大公约数。

【算法】 设函数 EgyptFraction 实现埃及分数问题，其算法描述如下：

输入：真分数的分子 A 和分母 B

功能：把一个真分数表示为最少的埃及分数之和

输出：最少的埃及分数之和

伪代码

```
step1: E = B/A + 1;
step2: 输出 1/E;
step3: A = A * E - B; B = B * E;
step4: 求 A 和 B 的最大公约数 R，如果 R 不为 1，则将 A 和 B 同时除以 R;
step5: 如果 A 等于 1，则输出 1/B，算法结束;否则转 step1 重复执行;
```

【程序】 函数 CommonFactor 实现求整数 m 和 n 的最大公约数，程序如下：

```
1   /*  埃及分数.cpp   */
2   #include <stdio.h>                    //使用库函数 printf 和 scanf
3   void EgyptFraction(int A, int B);  //函数声明，表示真分数
4   int CommonFactor(int m, int n);  //函数声明，求最大公约数
5                                         //空行，以下是主函数
6   int main( )
7   {
8     int A, B;
9     printf("请输入真分数的分子:");      //输出提示信息
10    scanf("%d", &A);
11    printf("请输入真分数的分母:");      //输出提示信息
12    scanf("%d", &B);
13    EgyptFraction(A, B);              //函数调用,表示真分数 A/B
14    return 0;                         //将 0 返回操作系统,表明程序正常结束
15  }
16                                        //空行,以下是其他函数定义
17  void EgyptFraction(int A, int B)
18  {                                  //函数定义,把真分数表示为最少的埃及分数之和
19    int E, R;
20    printf("%d/%d = ", A, B);         //输出真分数 A/B
21    do
22    {
```

```
        E = B/A + 1;                          //求真分数 A/B 包含的最大埃及分数
        printf("1/%d", E);                    //输出 1/E
        printf(" + ");
        A = A * E - B;                        //以下两条语句计算 A/B - 1/E
        B = B * E;
        R = CommonFactor(B, A);               //函数调用,求 A 和 B 的最大公约数
        if (R > 1)                            //最大公约数大于 1,即 A/B 可以化简
        {
         A = A/R; B = B/R;                    //将 A/B 化简
        }
    } while (A > 1);                          //当 A/B 不是埃及分数时执行循环
    printf("1/%d\n", B);                      //输出最后一个埃及分数 1/B
    return;                                   //结束函数 EgyptFraction 的执行
}
int CommonFactor(int m, int n)                //函数定义,求 m 和 n 的最大公约数
{
    int r = m % n;
    while (r != 0)                            //当余数不为 0 时执行循环
    {
      m = n;
      n = r;
      r = m % n;
    }
    return n;                                 //返回最大公约数 n
}
```

运行结果如下（下划线为用户输入）：

```
请输入真分数的分子：7
请输入真分数的分母：8
7/8 = 1/2 + 1/3 + 1/24
```

习 题 15

一、简答题

1. 贪心法能保证得到最优解吗？请举例说明。

2. 简述递推法、递归法和动态规划法的设计思想，三者之间有什么联系吗？

3. 动态规划法的设计难点是动态规划函数，动态规划函数描述了子问题的重叠关系，谈谈你对动态规划函数的理解。

二、程序设计题

1. 一辆汽车肇事后逃逸，4 个目击者提供如下线索：甲——牌号三、四位相同；乙——牌号为 31XXXX；丙——牌号五、六位相同；丁——牌号三到六位是一个完全平方数。请求出这辆肇事车的牌号。

2. 设有 n 个顾客同时等待一项服务，顾客 i 需要的服务时间为 t_i（$1 \leqslant i \leqslant n$），应如何安排 n 个顾客的服务次序才能使顾客总的等待时间达到最小？

3. 一个农夫带一只狼、一只山羊和一棵白菜过河，农夫每次只能带一样东西过河，但是任意时刻如果农夫不在场时，狼要吃羊、羊要吃菜，请为农夫设计过河方案。

4. 围绕山顶一圈有 10 个山洞，有一只狐狸和一只兔子在洞中居住，狐狸总想找到兔子并吃掉它，狐狸找兔子的方法是：先在狐狸居住的洞里找；再隔 1 个洞，到第 3 个洞里找；再隔 2 个洞，到第 6 个洞里找；再隔 3 个洞；再隔 4 个洞；依此类推。假定狐狸每天找 10 次，请为兔子选择一个安全的住处。

5. 有 4 个人打算过桥，这个桥每次最多只能有两个人同时通过。他们都在桥的某一端，并且是在晚上，过桥需要一只手电筒，而他们只有一只手电筒，这就意味着两个人过桥后必须有一个人将手电筒带回来。每个人走路的速度是不同的：甲过桥要用 1 分钟，乙过桥要用 2 分钟，丙过桥要用 5 分钟，丁过桥要用 10 分钟，显然，两个人走路的速度等于其中较慢那个人的速度，问题是他们全部过桥最少要用多长时间？

6. 在美国有一个连锁店叫 7-11 店，因为这个商店以前是早晨 7 点开门，晚上 11 点关门。有一天，一个顾客在这个店挑选了四样东西，然后到付款处去交钱。营业员拿起计算器，按了一些键，然后说：“总共是\$7.11。”这个顾客开了个玩笑说：“哦？难道因为你们的店名叫 7-11，所以我就要付\$7.11 吗？”营业员没有听出这是个玩笑，回答说：“当然不是，我已经把这四样东西的价格相乘才得出这个结果的！”顾客一听非常吃惊，“你怎么把它们相乘呢？你应该把它们相加才对！”营业员答道：“噢，对不起，我今天非常头疼，所以把键按错了。”然后，营业员将结果重算了一遍，将这四样东西的价格加在一起，然而，令他俩更为吃惊的是总和也是\$7.11。请找出这四样东西的价格各是多少。

附录 A

标准 ASCII 码

$b_7b_6b_5$ / $b_4b_3b_2b_1$	字符及 ASCII 值		字符及 ASCII 值		字符及 ASCII 值		字符及 ASCII 值		字符及 ASCII 值		字符及 ASCII 值		字符及 ASCII 值		字符及 ASCII 值	
	000	十进制	001	十进制	010	十进制	011	十进制	100	十进制	101	十进制	110	十进制	111	十进制
0000	NUL	0	DLE	16	SP	32	0	48	@	64	P	80	`	96	p	112
0001	SOH	1	DC1	17	!	33	1	49	A	65	Q	81	a	97	q	113
0010	STX	2	DC2	18	"	34	2	50	B	66	R	82	b	98	r	114
0011	ETX	3	DC3	19	#	35	3	51	C	67	S	83	c	99	s	115
0100	EOT	4	DC4	20	$	36	4	52	D	68	T	84	d	100	t	116
0101	ENQ	5	NAK	21	%	37	5	53	E	69	U	85	e	101	u	117
0110	ACK	6	SYN	22	&	38	6	54	F	70	V	86	f	102	v	118
0111	BEL	7	ETB	23	'	39	7	55	G	71	W	87	g	103	w	119
1000	BS	8	CAN	24	(	40	8	56	H	72	X	88	h	104	x	120
1001	HT	9	EM	25	)	41	9	57	I	73	Y	89	i	105	y	121
1010	LF	10	SUB	26	*	42	:	58	J	74	Z	90	j	106	z	122
1011	VT	11	ESC	27	+	43	;	59	K	75	[	91	k	107	{	123
1100	FF	12	FS	28	,	44	<	60	L	76	\	92	l	108	|	124
1101	CR	13	GS	29	-	45	=	61	M	77	]	93	m	109	}	125
1110	SO	14	RS	30	.	46	>	62	N	78	↑	94	n	110	~	126
1111	SI	15	US	31	/	47	?	63	O	79	←	95	o	111	DEL	127

附录 B

运算符的优先级和结合性

运算符	意　义	运算类别	优先级	结合性
( )	圆括号	一目	高	左结合
[ ]	下标运算符	二目		
.	成员运算符，引用结构体成员			
–>	通过指针引用结构体成员			
!	逻辑非	一目		右结合
~	按位取反			
+、–	取正、取负			
++、––	自增、自减			
&	取变量的地址			
*	间接引用运算符			
(type)	强制类型转换			
sizeof	取数据所占的字节数			
*、/、%	乘、除、求余	二目		左结合
+、–	加、减			
<<、>>	向左移位、向右移位			
<、<=、>、>=	小于、小于等于、大于、大于等于			
= =、!=	等于、不等于			
&、^、|	按位与、按位异或、按位或			
&&、||	逻辑与、逻辑或			
? :	条件运算符	三目		右结合
=	赋值运算符	二目		
*=、/=、%= +=、–=	复合赋值运算符：左值与右值完成算术运算后再赋给左值			
|=　^=　&=	复合赋值运算符：左值与右值完成按位逻辑运算后再赋给左值			
>>=　<<=	复合赋值运算符：左值与右值完成移位运算后再赋给左值			
,	逗号运算符，顺序求值	多目	低	左结合

附录 C

常用库函数

不同的 C/C++编译系统所提供的库函数的数目、函数名及函数功能并不完全相同，本附录只列出 VC++ 6.0 提供的常用库函数，具体使用时请读者查阅所用系统的库函数手册。

1. 数学函数

调用数学函数时，要在源文件中包含头文件 math.h，常用数学函数如表 C.1 所示。

表 C.1　常用数学函数

分　类	函数原型	函数功能	返回值	说明
三角函数	double acos(double x)	计算 x 的反余弦	x 的反余弦	x 的取值范围是[−1, 1]
	double asin(double x)	计算 x 的反正弦	x 的反正弦	
	double atan(double x)	计算 x 的反正切	x 的反正切	
	double cos(double x)	计算 x 的余弦	x 的余弦	参数 x 以弧度表示
	double sin(double x)	计算 x 的正弦	x 的正弦	
	double tan(double x)	计算 x 的正切	x 的正切	
指数函数	double exp(double x)	计算 e^x 的值	e^x 的值	
	double pow(double x, double y)	计算 x^y 的值	x^y 的值	
	double sqrt(double x)	计算 x 的平方根	x 的平方根	$x \geqslant 0$
对数函数	double log(double x)	计算 $\log_e x$ 的值，即 lnx	$\log_e x$ 的值	
	double log10(double x)	计算 $\log_{10} x$ 的值	$\log_{10} x$ 的值	
绝对值函数	double fabs(double x)	计算实数 x 的绝对值	x 的绝对值	
	double labs(long int x)	计算长整数 x 的绝对值		
取整函数	double ceil(double x)	计算不小于 x 的最小整数	该整数的双精度形式	向上取整
	double floor(double x)	计算不大于 x 的最大整数		向下取整

2. 字符处理函数

调用字符处理函数时，要在源文件中包含头文件 ctype.h，常用字符处理函数如表 C.2

所示。

表 C.2　常用字符处理函数

分　类	函数原型	函数功能	返回值
字符测试：测试字符的类型	int isalnum(int ch)	测试 ch 是否为字母或数字	是，则返回非 0，否则返回 0
	int isalpha(int ch)	测试 ch 是否为字母	
	int islower(int ch)	测试 ch 是否为小写字母	
	int isupper(int ch)	测试 ch 是否为大写字母	
	int isdigit(int ch)	测试 ch 是否为数字	
	int isxdigit(int ch)	测试 ch 是否为十六进制数码	
	int isascii(int ch)	测试 ch 是否为 ASCII 字符	
	int isspace(int ch)	测试 ch 是否为空格、跳格符等	
	int isgraph(int ch)	测试 ch 是否为可打印字符	
	int iscntrl(int ch)	测试 ch 是否为控制字符	
字符转换	int tolower(int ch)	将 ch 转换为小写字母	该字母对应的 ACSII 码
	int toupper(int ch)	将 ch 转换为大写字母	

3. 字符串处理函数

调用字符串处理函数时，要在源文件中包含头文件 string.h，常用字符串处理函数如表 C.3 所示。

表 C.3　常用字符串处理函数

分　类	函数原型	函数功能	返回值
字符串长度	unsigned int strlen(const char *str)	计算字符串 str 的字符个数	字符个数
字符串复制	char *strcpy(char *str1, const char *str2)	将字符串 str2 复制到字符串 str1 中	字符串 str1 的指针
	char *strncpy(char *str1, const char *str2, unsigned int n)	将字符串 str2 的前 *n* 个字符复制到字符串 str1 中	
字符串比较	int strcmp(const char *str1, const char *str2)	比较字符串 str1 和 str2，区分大小写	str1<str2：返回负数 str1=str2：返回 0 str1>str2：返回正数
	int stricmp(const char *str1, const char *str2)	比较字符串 str1 和 str2，不区分大小写	
	int strncmp(const char *str1, const char *str2, unsigned int n)	比较字符串 str1 和 str2 的前 *n* 个字符，区分大小写	
	int strnicmp(const char *str1, const char *str2, unsigned int n)	比较字符串 str1 和 str2 的前 *n* 个字符，不区分大小写	

续表

分　类	函 数 原 型	函 数 功 能	返 回 值
字符串拼接	char *strcat(char *str1, const char *str2)	将字符串 str2 拼接到字符串 str1 之后	字符串 str1 的指针
	char *strncat(char *str1, const char *str2, unsigned int n)	将字符串 str2 的前 *n* 个字符拼接到字符串 str1 之后	
字符串查找	char *strchr(const char *str, int ch)	在字符串 str 中查找字符 ch 第一次出现的位置	找到，则返回该位置的指针；否则返回 NULL
	char *strrchr(const char *str, int ch)	在字符串 str 中反向查找字符 ch 第一次出现的位置	
大小写转换	char *strlwr(char *str)	将字符串 str 中的字母转换为小写字母	字符串 str 的指针
	char *strupr(char *str)	将字符串 str 中的字母转换为大写字母	

4. 动态内存分配函数

调用动态内存分配函数时，要在源文件中包含头文件 malloc.h，常用动态内存分配函数如表 C.4 所示。

表 C.4　常用动态内存分配函数

分　类	函 数 原 型	函 数 功 能	返 回 值
内存分配	void *malloc(unsigned int size)	分配 size 个字节的内存单元	成功，则返回所分配内存单元的起始地址；否则返回 NULL
	void *calloc(unsigned int n, unsigned int size)	分配 n 个数据的内存单元，每个数据占 size 个字节	
	void *realloc(void *p, unsigned int size)	将 p 所指向的已分配内存单元的大小改为 size 个字节	
内存释放	void free(void *p)	释放 p 所指向的内存单元	无

5. 输入输出函数

调用输入输出函数时，要在源文件中包含头文件 stdio.h，常用输入输出函数如表 C.5 所示。

表 C.5　常用动态内存分配函数

分　类	函 数 原 型	函 数 功 能	返 回 值
标准输入	int getchar( )	从标准输入设备读取一个字符	正确读取，则返回读取的字符；否则返回–1
	char *gets(char *str)	从标准输入设备读取一个字符串放到 str 所指向的内存单元	正确读取，则返回 str 指针；否则返回 NULL

续表

分　类	函数原型	函数功能	返回值
标准输入	int scanf(const char *format, args,…)	从标准输入设备按 format 规定的格式读取数据，依次赋给输入列表 args 所指向的内存单元	正确读取，则返回读入并赋给 args 的数据个数；出错返回 0
标准输出	int putchar(char ch)	将字符 ch 输出到标准输出设备	正确输出，则返回字符 ch；否则返回 EOF
	int puts(const char *str)	将 str 指向的字符串输出到标准输出设备	正确输出，则返回换行符；否则返回 EOF
	int printf(const char *format, args,…)	将输出列表 args 的值按 format 规定的格式输出到标准输出设备	正确输出，则返回输出的字符个数；否则返回负数
文件的打开和关闭	FILE *fopen(const char *filename, const char *mode)	以 mode 方式打开文件 filename	成功，则返回一个文件指针；否则返回 NULL
	int fclose(FILE *fp)	关闭 fp 指向的文件，释放文件缓冲区	成功，则返回 0；否则返回非 0
文件的读取（文件输入）	int fgetc(FILE *fp)	从 fp 指向的文件读取一个字符	正确读取，则返回读取的字符；否则返回 EOF
	char *fgets(char *buf, int n, FILE *fp)	从 fp 指向的文件读取长度为 $n-1$ 的字符串，存入起始地址为 buf 的内存单元	正确读取，则返回地址 buf；否则返回 NULL
	int fscanf(FILE *fp, const char *format, args,…)	从 fp 指向的文件按 format 规定的格式读取数据并依次赋给输入列表 args 所指向的内存单元	正确读取，则返回读取的数据个数；否则返回负数
	int fread(char *buf, unsigned size, unsigned n, FILE *fp)	从 fp 指向的文件读取大小为 size 个字节的 n 个数据，存入 buf 所指向的内存单元	正确读取，则返回读取的数据个数；否则返回 0
文件的写入（文件输出）	int fputc(int ch, FILE *fp)	将字符 ch 输出到 fp 指向的文件	正确写入，则返回该字符；否则返回 EOF
	int fputs(const char *str, FILE *fp)	将 str 指向的字符串输出到 fp 指向的文件	正确写入，则返回 0；否则返回非 0
	int fprintf(FILE *fp , const char *format, args,…)	将输出列表 args 的值按 format 规定的格式输出到 fp 指向的文件	正确输出，则返回输出的字符个数；否则返回负数
	unsigned int fwrite(const char *ptr, unsigned int size, unsigned int n, FILE *fp)	将 ptr 所指向的 n*size 个字节输出到 fp 指向的文件	正确输出，则返回输出的字符个数；否则返回 0
文件的定位	void rewind(FILE *fp)	将 fp 指向文件的当前位置指针置于文件的开头	无
	int fseek(FILE *fp, long int offset, int base)	将 fp 指向文件的当前位置指针移动到以 base 为基准，偏移量为 offset 的位置	成功，则返回文件的当前位置指针，否则返回 −1

续表

分　类	函 数 原 型	函 数 功 能	返 回 值
文件的定位	long int ftell(FILE *fp)	fp 指向文件的当前位置指针（即读写位置）	返回 fp 指向文件的当前位置指针
	int feof(FILE *fp)	测试 fp 指向文件的当前位置指针是否到文件末尾	到文件末尾，则返回非 0；否则返回 0
其他	int ferror(FILE *fp)	测试 fp 指向的文件是否有错误	无错，则返回 0；否则返回非 0
	int fflush(FILE *fp)	如果 fp 指向的文件为写打开，则将文件缓冲区的内容写入文件；如果 fp 指向的文件为读打开，则清除文件缓冲区的内容	成功，则返回 0；否则返回 EOF

6. 实用函数

头文件 stdlib.h 中包含了所有不能归类于某个头文件的实用函数，常用的实用函数如表 C.6 所示。

表 C.6　常用的实用函数

分　类	函 数 原 型	函 数 功 能	返 回 值
字符串转换	double atof(const char *str)	将字符串 str 转换为 double 型	转换后的双精度值
	int atoi(const char *str)	将字符串 str 转换为 int 型	转换后的整型值
	long int atol(const char *str)	将字符串 str 转换为 long int 型	转换后的长整型值
	char *itoa(int value, char *str, int radix)	将整数 value 转换为 radix 进制表示的字符串	指向 str 的指针
	char *ltoa(long int value, char *str, int radix)	将长整数 value 转换为 radix 进制表示的字符串	
	char *ultoa(unsigned long int value, char *str, int radix)	将无符号长整数 value 转换为 radix 进制表示的字符串	
生成伪随机数	int rand(void)	生成 0～RAND_MAX 之间的随机数，RAND_MAX 为 32767	生成的随机数
	void srand(unsigned int seed)	为 rand 函数设置随机种子	无
终止程序	void exit(int code)	终止程序的执行，清空和关闭所有打开的文件，code 表示程序终止时的状态	无
	void abort(void)	终止程序的执行，但不清空、不关闭所有打开的文件	无

附录 D

程序设计实例索引

参 考 文 献

[1] K. N. King 著. 吕秀锋译. C 语言程序设计——现代方法. 北京：人民邮电出版社，2007.

[2] Adrian Kingsley-Hughes, Kathie Kingsley-Hughes 著. 顾晓锋译. 程序设计入门经典. 北京：清华大学出版社，2006.

[3] Brian W Kernighan, Dennis M Rirchie 著. C 程序设计语言（影印版）. 北京：清华大学出版社，1997.

[4] 谭浩强著. C 程序设计（第四版）. 北京：清华大学出版社，2010.

[5] 王红梅，胡明，王涛编著. 数据结构（C++版）. 北京：清华大学出版社，2005.

[6] 王红梅编著. 算法设计与分析. 北京：清华大学出版社，2006.

[7] 何炎祥，石莹，王娜编著. 程序设计基础. 北京：清华大学出版社，2006.

[8] 吴文虎编著. 程序设计基础（第 2 版）. 北京：清华大学出版社，2004.

[9] 张长海，陈娟编著. C 程序设计. 北京：高等教育出版社，2004.

[10] 王敬华，林萍，陈静编著. C 语言程序设计教程. 北京：清华大学出版社，2005.