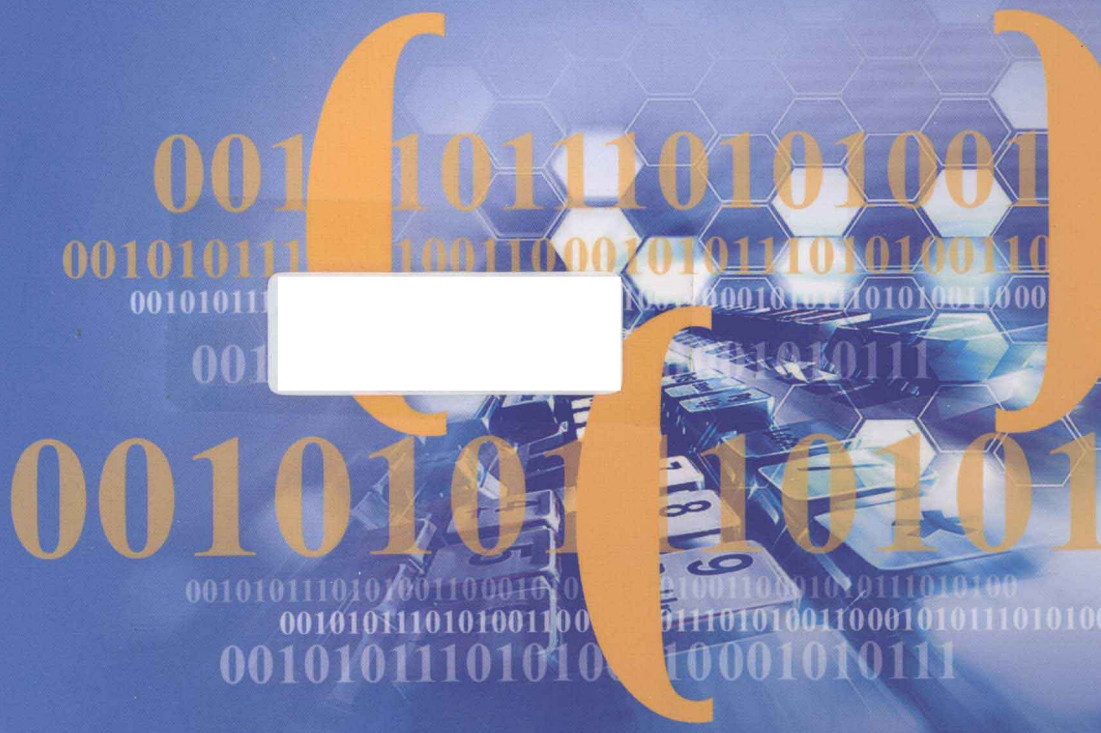
JISUANJI CHENGXU SHEJI XUEXI ZHIDAO YU SHIJIAN

计算机程序设计
学习指导与实践

贾伯琪◎编著



中国科学技术大学出版社

计算机程序设计
学习指导与实践

定价：30.00元

ISBN 978-7-312-03036-9

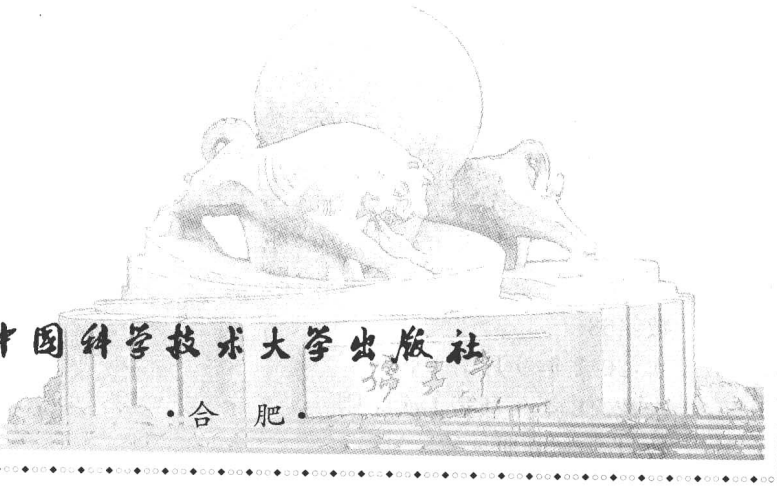
9 787312 030369 >

责任编辑　张善金
封面设计　刘俊霞

计算机程序设计

学习指导与实践

贾伯琪　编著



中国科学技术大学出版社

·合　肥·

内 容 简 介

本书是在原《C程序设计实验指导与练习》一书的基础上编写的，作为与“计算机程序设计”课程相配套的教学用书。主要内容有：一、程序设计预备知识；二、C语言程序的上机过程，介绍在不同的计算机系统上运行C程序的方法；三、知识要点与上机实验，每一实验给出了“学习重点”和“要点综述”，主要为学生自修学习指明重点，建议学习方法，每一实验都有明确的实验目的、要求、范例和实验内容及其常见错误；四、常见错误示例表；五、实验报告示例；六、综合测试题。此外，本书还在附录中给出了“常用库函数”、“ASCII码表”，以供读者在实验过程中使用。

书中的“要点综述”重点明确，内容详尽，提供的各实验均按《计算机程序设计课程教学大纲》内容分类，并按实验教学30～40学时设计，范例和上机习题都是经过精心选择安排的，示范和训练的优化结合，使学习者能够在一定的有限实验时间内达到理想的效果。

本书适合作为高等院校计算机程序设计课程的教辅用书，也可作为自学读者的辅助教材或参考书。

图书在版编目(CIP)数据

计算机程序设计学习指导与实践/贾伯琪编著. —合肥：中国科学技术大学出版社，2012.9
ISBN 978-7-312-03036-9

Ⅰ.计… Ⅱ.贾… Ⅲ.程序设计—高等学校—教学参考资料 Ⅳ.TP311.1

中国版本图书馆CIP数据核字(2012)第124047号

责 任 编 辑：张善金
出 版 发 行：中国科学技术大学出版社
地 址：安徽省合肥市金寨路96号 邮编：230026
网 址：http://press.ustc.edu.cn
电 话：发行部 0551—3606086—8808
印 刷：合肥现代印务有限公司
经 销：全国新华书店
开 本：787 mm×1092 mm 1/16
印 张：18.75
字 数：468千
版 次：2012年9月第1版
印 次：2012年9月第1次印刷
印 数：1—5000册
定 价：30.00元

前　言

“计算机程序设计”是一门十分重要的基础课，也是一门实践性很强的课程，仅仅通过阅读教科书和听课是不可能完全掌握的，学习程序设计的最重要环节就是实践。要想把书本上的知识变为自己所具有的能力，就需要实践—实践—再实践。程序设计的能力是通过动脑思索、动手操作、上机实践获得的。

凡是学习程序设计的人，开始往往都有这样的感觉：看教材或听课时，对老师课堂上讲解的内容和教材上写的内容基本上都能理解，但是当需要自己编写程序时却又无从下手。而每一个讲授程序设计课程的教师都会有这样的经历：有些常见问题，尽管已在课堂上再三强调，反复举例说明，学生还是不能完全理解，上机时更是错误百出。可以说这是学习过程中的必然现象，这些现象需要在上机实践中不断地认识、体会它们，并且有效地消除它们。

教学经验表明，“计算机程序设计”应强调动手实验和上机编程。以任务驱动方式，通过实例，针对问题进行分析、构建数学模型、理出算法、编写程序并上机调试实现。上机实验教学这个环节，要求学生养成良好的使用计算机解决问题的习惯(包括解题思路、算法思想和编程)，培养学生的计算思维能力和动手能力，鼓励学生探索、研究和创新，提高利用计算机这个智能工具来分析问题和解决问题的能力。

本教材是在《C语言程序设计实验指导与练习》实验教材的基础上编写而成的。本书出版前，作为中国科学技术大学“计算机程序设计”课程的配套学生用书，历经多年使用，不断更新、充实和完善，已连续10次修订再版，在教学过程中取得了良好的教学效果，曾获中国科学技术大学优秀讲义一等奖、校级教学成果二等奖。编写本教材的目的在于切实加强实践性教学环节，提高学生编程和上机解题的能力，为后续课程及研究生阶段的课题研究打下坚实的基础。

本书充分考虑了“计算机程序设计”课程的知识点与知识面合理分布，既注重基础，突出重点，又有一定的深度，适应学生的学习需求，有助于学生掌握课程的重点、难点，提高学习效率。

本书是根据《计算机程序设计课程教学大纲》要求编写的。教材对每一章和相应实验给出了“学习重点”和“要点综述”，为学生自修学习指明重点和建议学

习方法。每一实验都给出实验目的、要求、范例和实验内容及其常见错误，使学生在实践中达到对主教材内容的深入理解和熟练掌握，明确上机实验目的，清楚实验内容，以提高课程和实验教学的效果。

此外，本书还包括：算法的概念、算法的基本结构与表示及编程方法；在不同计算机系统上运行C程序的方法简介，例如，在UNIX系统环境下操作及C程序的上机过程，Turbo C++ 3.0的集成开发环境与程序调试，Visual C++ 6.0集成开发环境等；上机实验要求与上机实验报告；C程序编译、连接和运行中的常见错误；综合测试题等。教材中还充实了部分与课程相关的新知识、新内容，方便学生学习与参考，因此它也是实验过程中非常有用的专业性工具书。

教材提供的各实验均按本课程的教学大纲内容分类，按实验教学30～40学时设计，范例和上机习题都是经过精心选择的，既典型又有一般指导意义，既有难度适中的又有难度较大的和富有趣味的，进行针对性的训练和辅导，可有效地帮助学生深化和巩固对计算机语言程序设计基本概念的理解，提高读者阅读、分析程序和程序设计的能力，掌握算法设计和编程方法。示范和练习优化结合，确保在一定的有限实验时间内，达到理想的实验教学要求和课程教学目的。学生可以根据教师的安排和自身情况进行相应实验。

根据国家中长期教育改革和发展规划对本科阶段人才培养的定位，以及科学技术和经济社会发展对人才培养的需要，特别是国家对拔尖人材培养的战略需求，在遵循高等教育规律的前提下，发扬中国科学技术大学的办学传统，瞄准培养具有国际一流水平拔尖人才和科技英才的目标，进一步更新人才培养观念，创新人才培养模式，优化课程体系，改革教学方法和教育评价制度，促进人才培养国际化，探索新时期拔尖创新型人才培养的新途径，构建具有中国科大特色，适应优秀人才成长的培养体系，中国科学技术大学已对本科教育培养方案进行了修订，从2010级新生开始“计算机程序设计”课程将作为第一门计算机课程开设。2011年课程组编写出版了《计算机程序设计(C语言版)》教材，本书的此次修订是根据新的教学大纲要求和新编教材进行的，本书作为辅导教材力求适合于初学者。

本书在编写过程中，作者参阅了大量的有关书籍和资料，并得到了中国科学技术大学教务部门、信息科学技术学院教学委员会的大力支持和关怀；在教材成文的过程中，我的同事(“计算机程序设计”课程组的教师)顾卫兵、苏仕华、帅建梅、何克东、张四海、白雪飞、谭立湘、尹东、刘勇、郑重、吴文涛、秦琳琳、杨坚等起了很大作用，根据教材在教学实践中的使用情况，他们提出了许多很好的建议和宝贵的意见，给予了极大的支持，在此一并表示衷心的感谢。

虽然本书是在教学实践的基础上经过多次修订后编写而成的，但是任何一本好的教材，都需要在教学实践中不断听取教师和学生的反馈意见，不断修改，才能使之日臻完善。由于作者水平有限，书中疏漏之处在所难免，欢迎读者和同行专家多提宝贵意见，以便日后再版时，更好地反映时代特征，更完美地适合教学需要。

编　者

2012年5月

目　录

第一部分 预备知识

计算机的使用改变了人类的生活方式和工作方式，将人类的创造性思维推向一个全新的阶段。思维活动可以利用语言来形式化，而语言层次可以离开意识层次相对独立地活动。计算机语言作为人和机器之间进行意识交流的工具，在各个领域得到了广泛的应用。人通过计算机语言将意识活动交给计算机进行独立地加工，产生进一步的思维活动，所以可以认为计算机是人类思维的工具。计算机思维是一种物化的思维，是人脑思维的进一步延伸。

在计算机语言层次，人与计算机的意识活动的交流是通过程序这个环节来完成的。1976 年，著名的计算机科学家 N. Wirth 出版了一本名为 *Algorithms＋Data＝Programs* 的著作，提出了程序是算法和数据结构相结合的观点，也就是说程序设计主要包括两方面的内容：行为特性的设计和结构特性的设计。行为特性的设计是指完整地描述问题求解的全过程，并精确定义每个解题步骤，这一过程即是算法的设计；而结构特性的设计是指在问题求解的过程中，计算机所处理的数据之间的联系以及这些联系的表示方法。

因此，为了有效地进行程序设计，应当至少具有两个方面的知识：第一，掌握一门高级语言的语法规则；第二，掌握解题的方法和步骤。计算机语言只是一种工具。光学习语言的规则还不够，最重要的是学会针对各种类型的问题，拟订出有效的解题方法和步骤——算法。有了正确的算法，可以利用任何一种计算机高级语言编写程序，使计算机进行工作。

本书的重点不放在介绍 C 语言的具体语法上，而是从算法入手，使读者养成一个习惯：在求解一个问题时，首先应当考虑算法，而不是马上动手写程序。程序处理的对象是数据，数据与数据之间会存在某种形式联系，这就是数据结构。在程序设计时，除了要考虑算法外，还要考虑并选择适当的数据结构。对于不同的数据结构，要用不同的算法去处理。

在这里，我们将重点讨论算法这个组成程序的基石。

一、算法的概念

粗略地说，算法是解决问题的具体步骤。解决任何一个问题都有一定的步骤。例如，你要看电影，就要先买票，然后按时到电影院，进场，找座位，坐下，看电影，退场等。你要考入大学，首先要填报名单，交报名费，拿到准考证，按时参加考试，并获得录取通知书，到指定学校报到注册等。这些都是按一系列的顺序进行的步骤，缺一不可，次序错了也不行。当然它们是一些人们习以为常的原始算法，算法的描述采用了自然语言的方法。

在欧几里得的《几何原本》里，阐述了求两个数的最大公因子的过程，也称欧几里得算法。

欧几里得算法：给定两个正整数 m 和 n，求它们的最大公因子，即同时能够整除 m 和 n 的最大正整数。

Step1.　以 n 除 m，并令 r 为所得余数（显然 $n>r\geqslant 0$）；

Step2. 若 $r=0$，算法结束，n 即为 m 和 n 的最大公因子；

Step3. 置 $m \leftarrow n, n \leftarrow r$，返回 Step1。

在描述欧几里得算法的时候，我们采用了一种比自然语言更抽象一些的方法，或者称半自然语言的方法。同时欧几里得算法的描述与看电影和报考大学的算法相比，具有完全的确定性。在欧几里得算法的描述中引入了一些变量，如 m、n、r，引入了关系运算符 $=$，$\geqslant$，还引入了记号 $\leftarrow$ 表示将该符号右边的变量的值送到左边的变量中去，所以 $\leftarrow$ 也称为赋值号。

在算法的描述乃至程序的设计中，变量是个很重要的概念。所谓变量是指一块在计算机内存中分配的存储空间，或者说，每个变量代表了内存中一个确定的存储区。赋值是指将赋值号右边的变量中存储的值送到左边的变量所代表的存储单元中去。

还需要注意的是，在算法中不仅各步骤间的顺序是重要的，在每步内的动作次序同样也是重要的。例如，在欧几里得算法的 Step3 中，“置 $m \leftarrow n, n \leftarrow r$”绝不能写成“置 $n \leftarrow r, m \leftarrow n$”，因为这样做的结果是在 m、n 和 r 中最后都存储了一个值 r，即都变成了 Step1 除法运算的余数。

由于算法是对实际运算的抽象表述，同时算法描述的往往是比看电影复杂得多的运算，所以一个算法每一步的含义（或者说是作用）往往不是一目了然的。学习一个算法的最好方法莫过于将该算法所允许的实例代入到算法中，按算法的步骤去执行算法，这种由人来执行算法的方法称为“人工模拟”，人工模拟也是开发和检验算法的一种重要步骤，通过人工模拟能了解算法每一步的实际含义。算法中一般都包含了“如果……则……”的步骤，规定对不同的实例以不同的途径处理，所以为了深刻弄懂一个算法，一般还需要设计多个实例来反复体会算法中各种不同处理途径的作用。对欧几里得算法，第一个实例是：

$m=12$ 和 $n=4$，

Step1. $m/n=12/4=3\cdots\cdots0$；

Step2. 由于余数 $r=0$，算法结束，4 即为 12 和 4 的最大公因子。

由于这个实例没有“走”到 Step3，所以由这一个实例不能全面了解欧几里得算法。第二个实例是：

$m=48$ 和 $n=14$，

Step1. $m/n=48/14=3\cdots\cdots6$；

Step2. 余数 r 不为零，进入下一步；

Step3. 置 $m \leftarrow n, n \leftarrow r$，即 $m=14$、$n=6$，返回 Step1。

Step1. $m/n=14/6=2\cdots\cdots2$；

Step2. 余数 r 不为零，进入下一步；

Step3. 置 $m \leftarrow n, n \leftarrow r$，即 $m=6$、$n=2$，返回 Step1。

Step1. $m/n=6/2=3\cdots\cdots0$；

Step2. 由于余数 $r=0$，算法结束，n 的现值 2 为 48 和 14 的最大公因子。

对欧几里得算法的跟踪应当可以告一段落了，但再深入地想一想，欧几里得算法的上述表述是否尚有可改进之处呢？下一个实例是：

$m=14$ 和 $n=48$，

Step1. $m/n=14/48=0\cdots\cdots14$；

Step2. 由于余数 $r=14$，进入 Step3；

Step3. 置 $m \leftarrow n, n \leftarrow r$，即 $m=48$、$n=14$，返回 Step1。

我们可以发现，当 m 小于 n 时，经过 Step1、Step2 和 Step3 后，仅将 m 和 n 互换了一次。所以，可以为欧几里得算法增加算法 Step0：

Step0. 如果 $m<n$，则 $m \longleftrightarrow n$（即交换）。

这样，尽管欧几里得算法由三步变为四步，但对 $m<n$ 的实例，实际减少了算法执行的时间。而按 m 和 n 之间大小关系的出现概率考虑，$m<n$ 及 $m>n$ 出现的概率可能各为一半。

至此可以给算法一个更精确的定义：**一个算法是一个有穷规则的集合，其中的规则规定了解决一个问题的运算序列。**

作为一个算法必须具备以下五个特性：

(1) 有穷性：一个算法必须总是在执行有穷步之后结束，且每一步都可在有穷时间内完成。

欧几里得算法满足这个条件。因为在 Step1 以后，r 的值肯定小于 n，所以如果 r 不等于零，则在经过 Step3 将 r 的值赋给 n 以后，重新进行 Step1 时，n 的值已经减小，所以循环过程中，r 的值是一个递减序列，而正整数的递减序列必然最后要终止。所以对任何给定的 m 和 n 的值，Step1 只会执行有穷次。对足够大的 m 和 n，可能算法执行的次数相当多，但即使达到天文数字，仍然是有穷的。

如果一个计算不具有有穷性，但具有算法其他特性，则可称其为计算方法。例如，对无穷调和级数的和的计算就不是一个算法而是一个计算方法，在确定了项数或确定了对精度的要求以后，才能满足有穷性的要求，也就是说，这时对无穷调和级数的和的计算才可能成为一个算法。

(2) 确定性：算法的每一个步骤都必须有确定的定义。

欧几里得算法的每一个步都是确定的。例如，在 Step1 中，除法的算术运算法则保证了两个正整数相除的步骤，而结果的商和余数都是确定的。

(3) 可行性：一个算法是能行的，即算法中描述的操作都是可以通过已经实现的基本运算执行有限次来实现的。

欧几里得算法所涉及的运算包括整数的表示、整数的除法、整数是否为零的判断及整数的赋值，这些运算都是基本的、能行的。

(4) 输入：一个算法有零个或多个输入，算法的输入是算法执行的初始数据，这些输入取自于特定对象的集合。

欧几里得算法需要两个整数 m 和 n 作为初始数据。

(5) 输出：一个算法有一个或多个输出，作为算法执行的结果。这些输出是同输入有某个特定关系的量。

欧几里得算法的结果是正整数 m 和 n 的最大公约数。

二、算法的基本结构和表示

对算法和程序设计方法的理论研究及程序设计实践指出，算法的基本组成结构只需要有三种，第一种是顺序结构，第二种是选择结构，第三种是循环结构。或者说，任何一个算法，无论其多么简单或多么复杂，都可以由三种结构组合和构造而成。

前面在讨论欧几里得算法时，已经有了变量的概念和赋值运算的概念，并介绍了算法的

一种类似于自然语言的表示方法。算法的作用在于记录及交流人解决问题的思想，同时算法也是作为编制计算机程序的前导步骤。为使算法的描述更简洁和清晰，人们研究和创造了多种表示算法的方法，本书主要介绍算法的伪码表示法、流程图表示法、PAD 图表示法和结构化流程图(N－S 图)。伪码是一种表示算法的代码，伪码采用了类似于程序设计语言的语句表示方法，但伪码不是任何一种程序设计语言，不涉及程序设计的细节。流程图是一种图语言，用流程图可以直观地了解算法的结构，流程图表示中所用的符号已由国家标准局制定了国家标准(见图 1.1)。PAD(Problem Analysis Diagram，意为问题分析图)是软件开发中逐步得到推广使用的一种用来表示算法的图形工具。

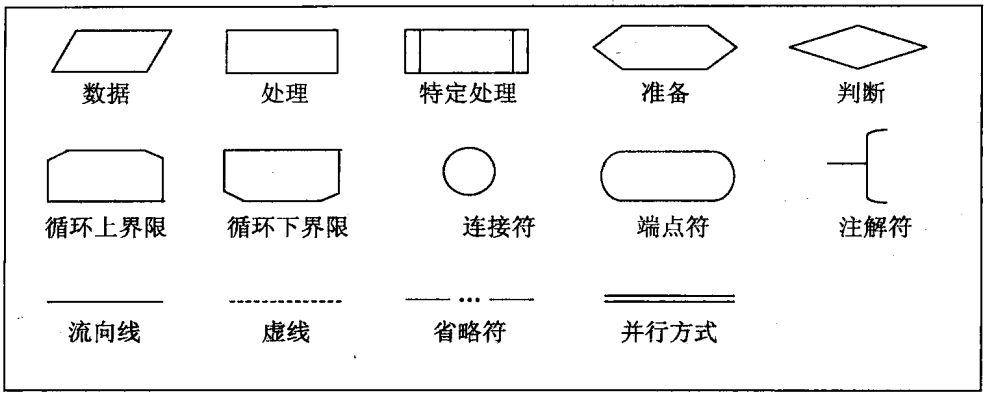


图 1.1　标准流程图的规定符号

下面分别结合算法的举例来讨论顺序结构、选择结构和循环结构的含义，以及伪码表示法、流程图表示法和 PAD 图表示法。

(一) 顺序结构

在伪码表示法中，每一个步骤称为一条语句。根据需要可将多条语句包容在 BEGIN 和 END 之间组成语句块(简称块)，即组织成以下形式：

```
BEGIN
    语句 1
    语句 2
    …
    语句 n
END
```

顺序结构中，各语句或块是按照在算法中排列的先后次序执行的。

【例 1.1】 求两个数的绝对值之和。

算法 1：

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* 输入：任意两个数                                                         *
* 输出：输入的两个数的绝对值之和                                           *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

BEGIN

num1←第一个输入的数

num2←第二个输入的数

num1←num1 的绝对值

num2←num2 的绝对值

sum←num1＋num2

输出 sum 的值

END

图 1.2 所示为算法 1 的流程图表示。

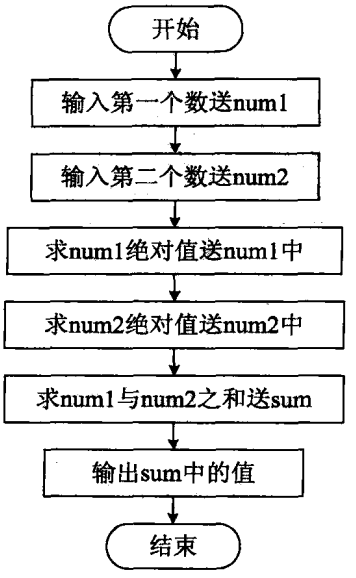


图 1.2 算法 1 的流程图

（二）选择结构

选择结构是根据某种条件选择性地执行算法的某一部分。选择结构最常见的是判断算法中所说明的条件是否成立，如果条件成立，执行某个语句或语句块，否则执行另一个语句或语句块，后一个语句或语句块在某些情况下可以省略。

如图 1.3 所示为选择结构的流程图。可用伪码表示为：

IF〈条件〉

〈语句或语句块 1〉

[ELSE

〈语句或语句块 2〉]

无论是执行了〈语句或语句块 1〉，还是执行了〈语句或语句块 2〉都完成了选择结构的执行。上列表示还说明，如果我们在条件不成立时不要求做任何动作，可以省略 ELSE 的那一部分。

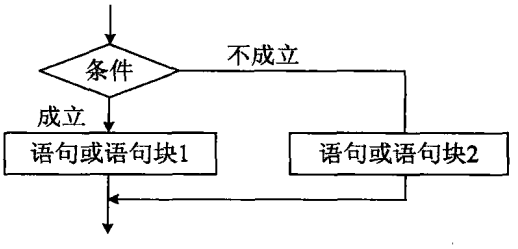


图 1.3 选择结构的流程图

【例 1.2】 输入三个数，求其中最大的数。

算法 2：

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* 输入：任意三个数                                                          *
* 输出：输入的三个数中最大的数                                              *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

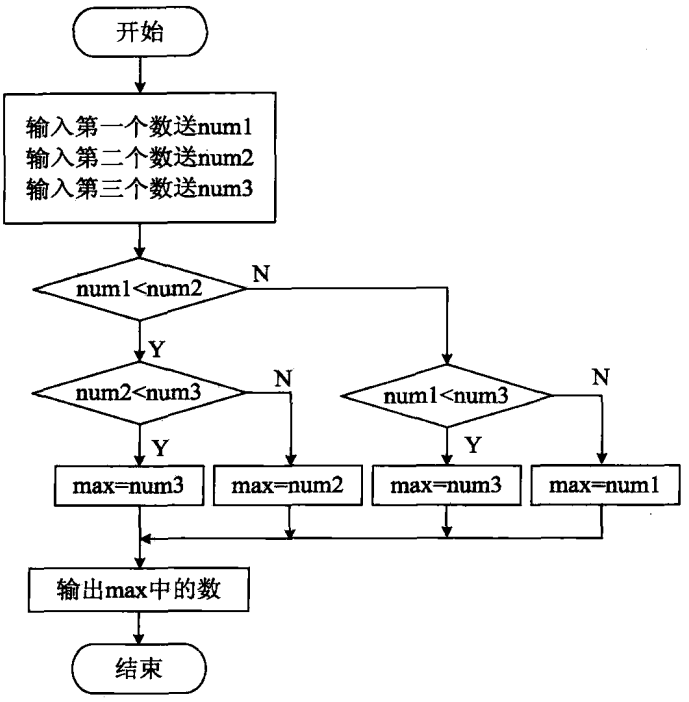


图 1.4　算法流程图

```
BEGIN
    num1←第一个输入的数
    num2←第二个输入的数
    num3←第三个输入的数
    IF(num1<num2)
        IF(num2<num3)
            max ← num3
        ELSE
            max ← num2
    ELSE
        IF(num1<num3)
            max ← num3

        ELSE
            max ← num1
    输出 max 的值
END
```

图 1.4 所示为算法 2 的流程图。

（三）循环结构

循环结构根据某种条件重复性地执行算法所规定的某一部分（称循环体）。循环体执行完成后，再次判断所说明的条件是否成立，如果条件成立再次执行循环体，如此循环重复直至条件不成立为止，从而达到重复性地执行算法所规定的某一部分的目的。

【例 1.3】 设计一个算法，统计输入的一组数据中非负整数的个数。

算法 3：

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* 输入：有限个数的一组数据                                                    *
* 输出：统计这组输入数据中的非负整数的个数                                    *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    BEGIN
    1   number ← 输入数据的个数
    2   count ← 0                    //非负整数的个数，先预置为 0（即累加器清零）
    3   WHILE(number≠0)
            BEGIN
    4           num ← 输入的下一个数
    5           IF(num≥0)
    6               count ← count+1
    7           number ← number−1
```

```
        END
8       输出 count 的值
END
```

图 1.5 所示为算法 3 的流程图。

算法 3 和算法 2 不同，算法 2 中预先确定了输入数据的个数，而算法 3 并没有限定输入数据的个数，也就是说该算法能够灵活地处理不同个数的输入数据。所以定义了一个变量 number，该变量作为算法所要求的第一个输入量，代表算法当前这次所处理的输入数据的个数，从第二个输入量开始才是算法所处理的输入数据。在算法执行中，每处理一个输入数据，将 number 的值减 1，一直到所有的输入数据处理完成为止。

算法 3 用"//"的方法引入了对算法某个步骤的注释。为提高算法或程序的可读性，在其适当的地方加以注释是十分必要的，应有意识地逐步培养这种良好的习惯。

和前面几个算法相比，算法 3 显然要复杂一些。为了学习和验证这个算法，我们可以用人工模拟的方法，将该算法所允许的实例代入到算法中去，按算法的步骤人工执行一次，即跟踪算法的动态执行过程。同时可以对算法作进一步的改进完善。

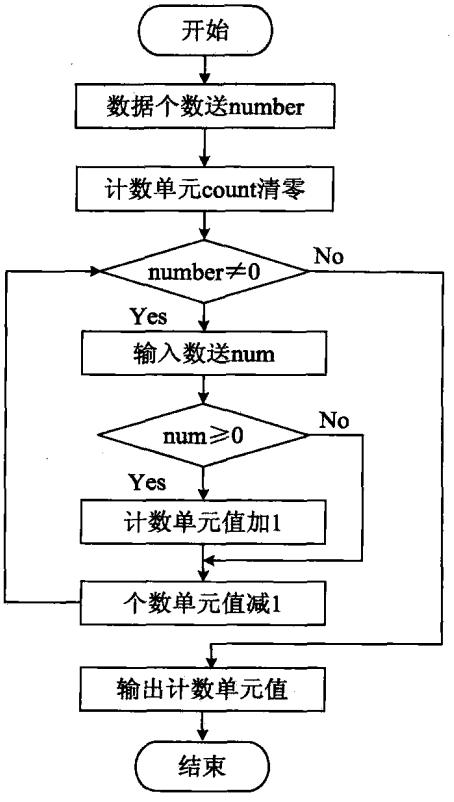


图 1.5 算法 3 的流程图

为了方便跟踪算法的动态执行过程，可将算法中的可执行语句及包括判断的语句按顺序编号。同时为记录算法当前执行的以及下一步将执行的语句，并记录算法中的变量值的变化过程，可以设计一张算法跟踪表。跟踪表的第一列是执行的步数，第二列和第三列分别是这一步所执行和下一步将执行的语句的编号，第四列是当前这一步的注记，以后的各列是算法中所用的各个变量。算法每执行一步，跟踪表的记录就增加一行，直至过程结束。

三、C 语言 PAD 标准图式及借助它编程的方法

(一) PAD(Problem Analysis Diagram)的基本原理与图式

PAD 是在 Warnier 图的基础上产生的一种二维树形结构的软件设计方法。由于它使用 PASCAL 程序的二维展开图，因而也把 PAD 看作是 PASCAL Diagram 的缩写，称之为 PASCAL 展开图。与流程图等原有的程序逻辑图相比，它有四个主要特点：

(1) 清晰(简洁地表现程序逻辑)。

(2) 可视(程序逻辑透明，能与源程序对应)。

(3) 易理解、易学、易读、易记，便于编程，便于测试，便于审查，便于维护。

(4) 对程序设计风格有较强的制约性。

PAD 的核心思想是利用自顶向下设计和反复的、逐步求精的基本概念，把人们思考的解决问题的想法转换成计算机能进行处理的明确而详尽的过程。应用 PAD 将有利于提高程序的设计、检查和维护能力。

PAD 使用六种符号，如图 1.6 所示，它们分别用于描述处理、循环、选择、语句标号、定义以及过程。

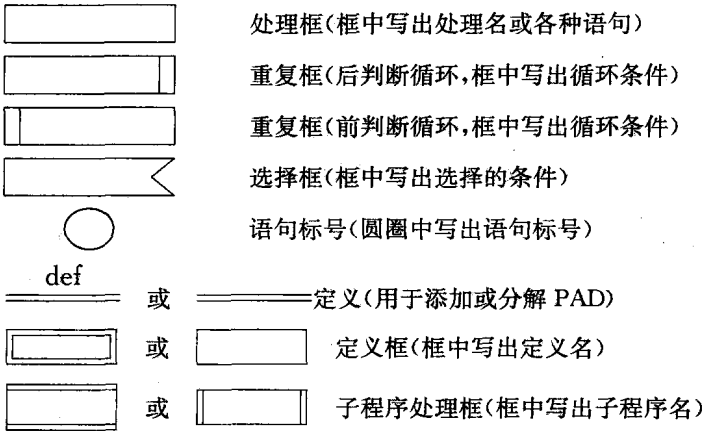


图 1.6 PAD 中所用的符号

与其他规定的软件设计方法类似，PAD 为软件人员提供了设计系统时所应遵循的步骤：

(1) 从设计程序或系统的一种模糊概念开始，标识现有过程的各个部分的顺序，即在时间顺序上模模糊糊部分的前后关系——顺序(连接)的细化。

(2) 标识将要重复的过程部分以及重复的初始和终止条件，即主要循环过程的开始和结束条件等——循环的细化。

(3) 标识将要执行的各个过程部分的条件，即主要选择的条件——选择的细化。

(4) 如果需要平行处理，则标识将要平行执行的那部分——平行性细化。

(5) 重复(1)～(4)步，直到过程完全确定和完全精确为止。

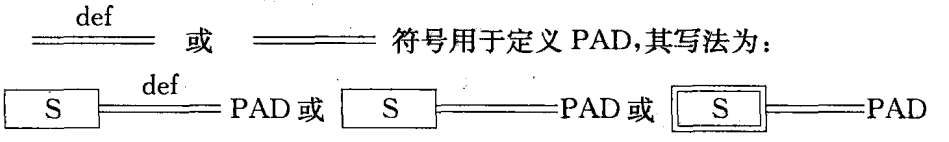
归纳起来，PAD 程序开发原理可用图 1.7 来描述。

(二) PAD 的写法及注意事项

1. PAD 的写法

PAD 是一种二维的树形结构的程序表示法，它表示图形/概念的一种唯一的组合。纵向表示顺序信息，横向描述分支和嵌套层，两者组合即构成表示程序的 PAD 图。

写 PAD 时，用户可以从定义一个程序或程序组的顶层图开始，然后再用 def 符号添加详图。



其中，S 为 PAD 的名字。

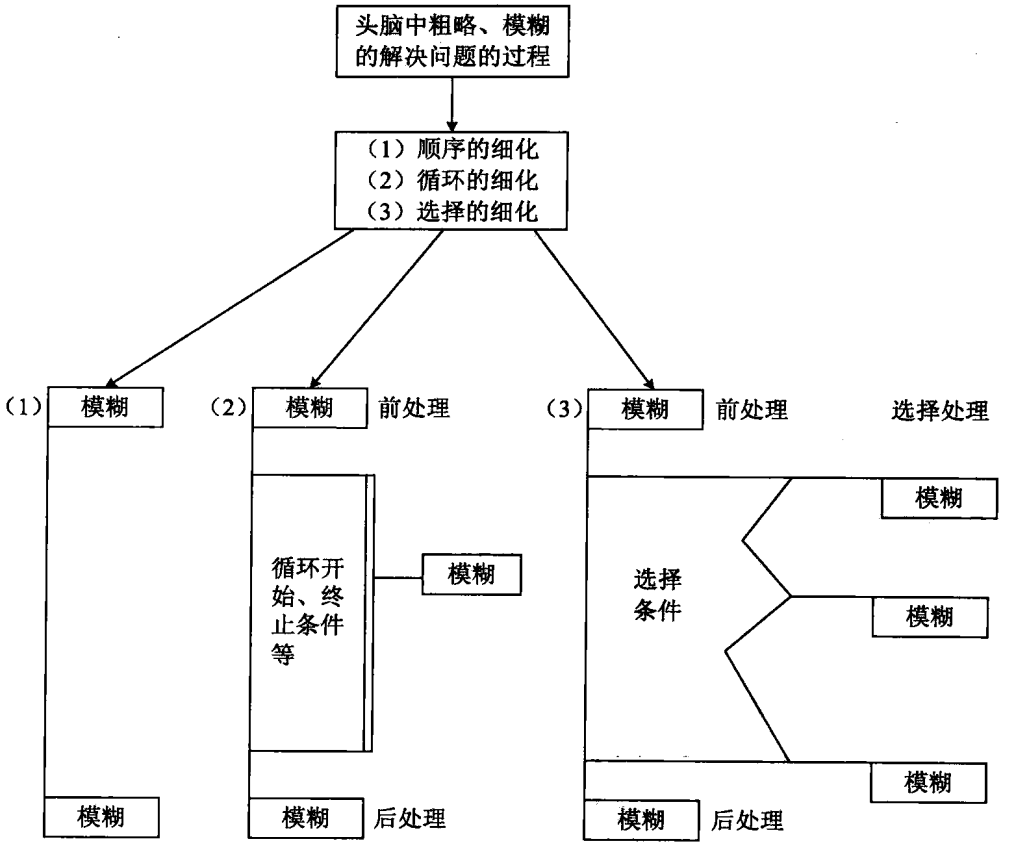
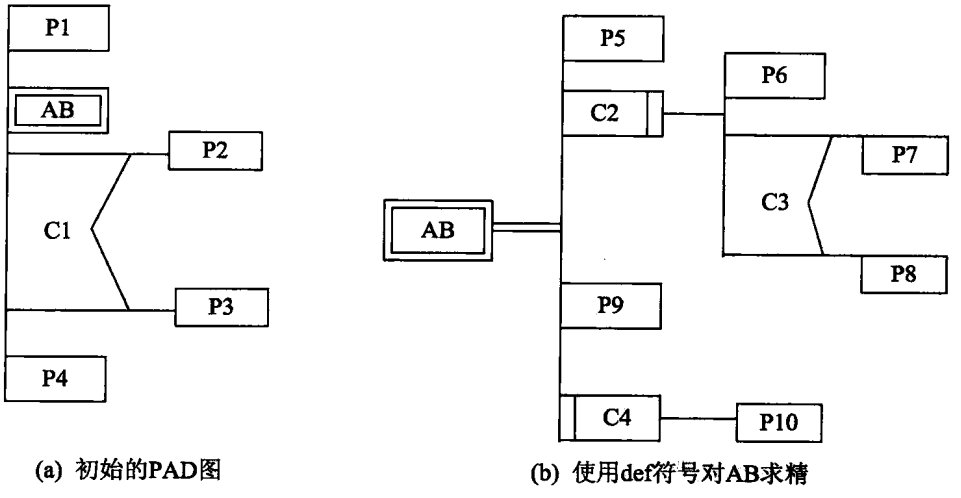


图 1.7 PAD程序开发原理

def 符号用法有三：

（1）用于表示改进阶段的各个详细步骤，即逐步求精，如图 1.8 所示。

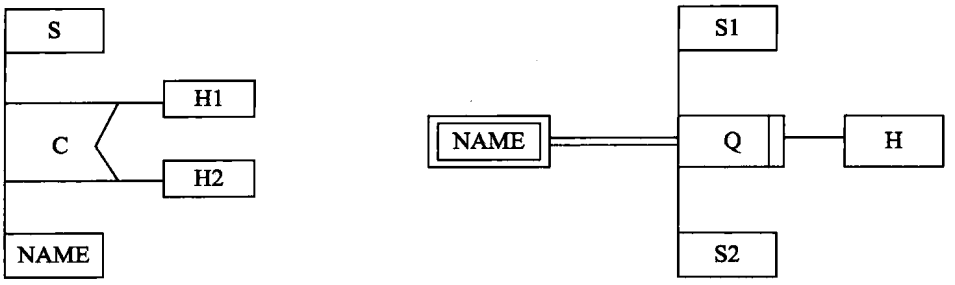


(a) 初始的PAD图　　(b) 使用def符号对AB求精

图 1.8 定义 PAD 细部的用法

(2) 用于把较大的 PAD 分解成较小的 PAD。

(3) 用于对某一处理添加 PAD,或连接上下页,即在本页末尾和下页开始定义之,如图 1.9 所示。



(a) (b)

图 1.9 定义上下页连接的用法

2. 注意事项

(1) 当处理框为空时,可省略该框及其分支,如图 1.10 所示。

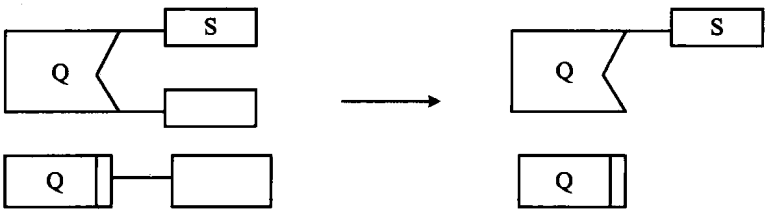


图 1.10 省略处理框及其分支

(2) 在 PAD 图外侧可写上注解,如图 1.11 所示。

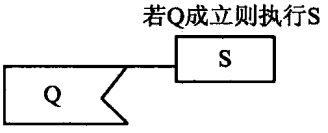


图 1.11 添加注释

(3) 在横向连接 PAD 框时:

①选择框,循环框后可连接任一种 PAD 框(处理框,选择框,循环框,定义框,子程序框)。

②处理框后不能连接任何一种处理框(处理框,选择框,循环框,定义框,子程序框)。

(4) 在纵向写 PAD 时,可以写任何一种处理框(处理框,选择框,循环框,定义框,子程序框)。

(三) C 语言 PAD 标准图式及借助它编程的方法

1. C 语言 PAD 标准图式

PAD 的写法基本上与语言无关,除标准图式外,使用者可以根据需要进行不同的扩充。根据 PAD 的基本原理和 C 语言的特征,可以写出 C 语言的标准图式,如表 1.1 所示。

表 1.1 C语言 PAD标准图式

		PAD	C
顺序		H1; H2;	H1; H2;
选择	if 型	(Q) H1;	if (Q) { H1; }
	if～ else 型	(Q) H1; H2;	if (Q) { H1; } else { H2; }

选择	switch 型	i1 H1;break;
		i2 H2;break;
		i= ⋮
		in Hn;break;
		default; Hd;
			switch (i)
			{
			case i1: H1; break;
			case i2: H1; break;
			……
			case in: H1; break;
			defaule: Hd;
			}

循环	while 型	(Q) H;	while (Q) { H; }
	do～ while 型	(Q) H;	do { H; } while (Q);
	for 型	(e1;e2;e3) H;	for (e1; e2; e3) { H; }

说明： (1) H 为语句序列，若只有一个语句，则不必用花括号括起来。

(2) 凡是语句必须以分号结束。

(3) 凡是条件，必须用圆括号括起来。

PAD 能够很容易地转换为程序。要借助 PAD 编程，首先要写出 PAD 图，然后再根据该图式和上述 C 语言标准图式进行编程。PAD 图编程步骤如下：

(1) 根据算法画出 PAD 图。

(2) 把 PAD 图看作横向生长的树，沿着树迹前进(如图 1.12 中箭头所示的方向)，写出源程序，其原则是：

① 遇到处理框，写出其处理内容。

② 遇到循环框或选择框，则根据表 1.1 所示的 C 语言 PAD 标准图式写出相应的 C 语句。

沿树迹结束程序即告完成。

2. 应用 PAD 编程示例

【例 1.4】 将 2000 年～2050 年是否闰年打印出来。

算法 4:

(1) 根据题目要求画出 PAD 图,如图 1.12 所示。

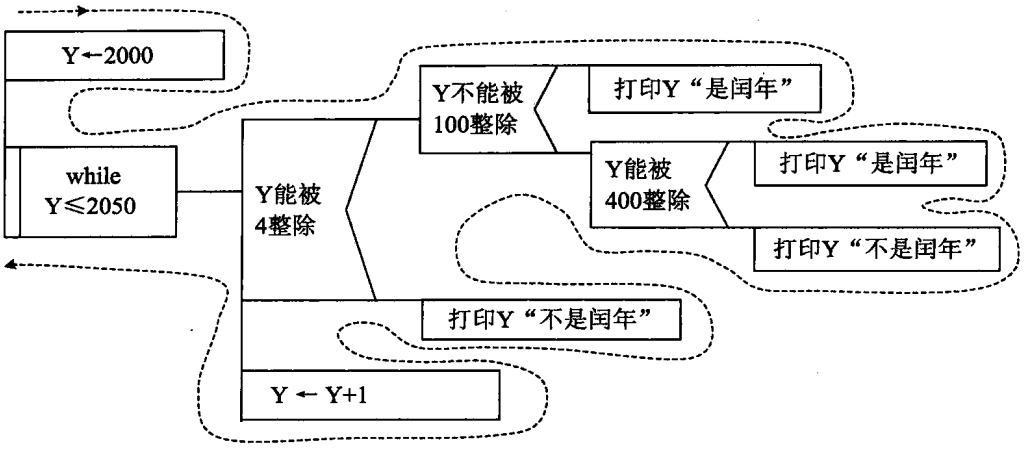


图 1.12 示例 PAD 图

注意: 画 PAD 图时,图中的树迹线可以不画。

(2) 根据 PAD 图写出源程序:

```
#include "stdio.h"
main(){
    int y;
    y=2000;
    while(y<=2050){
        if(y%4==0){
            if(y%100!=0)
                printf("%d is a leap year.\n",y);
            else
                if(y%400==0)
                    printf("%d is a leap year.\n",y);
        }
        else
            printf("%d is not a leap year.\n",y);
        y=y+1;
    }
    return;
}
```

四、结构化流程图(N-S 图)

(一) N-S 图的画法

N-S 图是一种不使用流程线的结构化流程图,1973 年由 I. Nassi 和 B. Shneiderman 提

出，N、S分别是他们二人名字的头一个字母。N－S图不使用流程线，全部算法写在一个矩形框内。它只能由顺序结构、选择结构、循环结构三种基本结构单元组成，这三种基本结构构成的算法整体上是一个大的顺序结构。结构化流程图的画法介绍如下：

(1) 顺序结构的画法如图 1.13 所示。其中 A 框与 B 框分别代表一个基本操作，如加、减、乘、除运算，打印、赋值等。两个或多个矩形组成一个顺序结构。

(2) 选择结构的画法如图 1.14 所示。它表示当条件 P 成立时，执行 A 操作；当条件 P 不成立时，执行 B 操作。

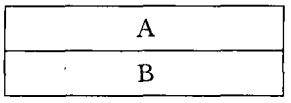


图 1.13 顺序结构

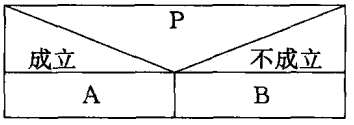


图 1.14 选择结构

(3) 循环结构包括当型循环结构和直到型循环结构。

① 当型循环结构的画法如图 1.15(a)所示。它表示当条件 P1 满足时，执行操作 A，执行完 A 后，再回去判断条件 P1 是否成立，若成立就再执行操作 A，如此反复，直至条件 P1 不满足为止。

② 直到型循环结构的画法如图 1.15(b)所示。它表示先执行操作 A，然后判断条件 P2 是否满足，若不满足，则去执行操作 A，如此继续，直到条件 P2 满足为止。

(a) 当型循环　　(b) 直到型循环

图 1.15 循环结构

比较①和②，当型循环的特点是“先判断后操作”，直到型循环的特点是“先操作后判断”。

(二) 用 N－S 图描述算法应用举例

【例 1.5】 求两整数的最大公约数和最小公倍数。

求最大公约数采用常用的算法是辗转相除法。

求最小公倍数算法为：最小公倍数＝原来的两数乘积/最大公约数。

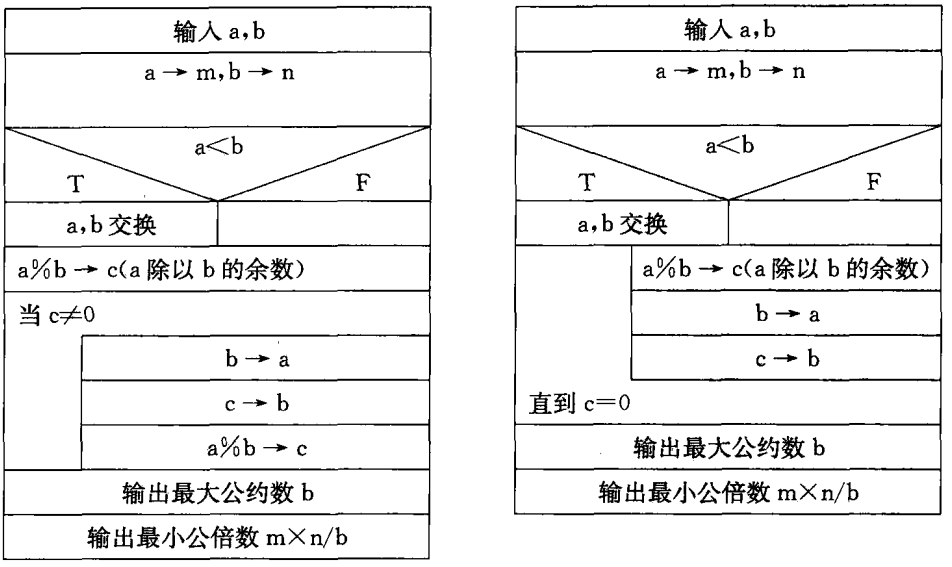
采用当型循环的算法描述如图 1.16(a) 所示，采用直到型循环的算法描述如图 1.16(b) 所示。

五、算法的基本分类

算法是解决问题的方法，不同的领域有各自的算法。如果根据问题的领域来区分，算法可分为数值问题的算法和非数值问题的算法。

数值问题算法是指解决传统数学问题的算法，例如，解方程的算法、解方程组的算法、积分算法和微分算法，等等。随着计算机在数值计算方面的应用和研究的发展，求各种数值问题的近似解的算法也获得迅速的发展和应用。数值问题的算法是数学研究的一个重要方

面，本书将在相应章节给出几个数值问题算法的例子（只限于计算机算法，即计算机能执行的算法）。由于数值运算有现成的模型，可以运用数值分析方法，因此对数值问题的算法研究比较深入，算法比较成熟。各种数值运算都有比较成熟的算法可供选用。常常把这些算法汇编成册，或者写成程序形式存放在磁盘或磁带上供用户调用。例如，有些计算机系统提供“数学程序库”，使用起来十分方便。



(a) 流程图 1　　(b) 流程图 2

图 1.16　N-S 图示例

和数值问题的算法相比，计算机科学往往对非数值问题的算法给予了更多的重视，计算机非数值问题领域同样也是个相当宽广的领域，在各个不同的方向上都有不同的非数值问题的算法，例如，在图论中有涉及图的各种处理的图算法，在规划问题中有各种规划的算法。最常见的是用于事务管理领域，例如，图书检索、人事管理、行车调度管理，等等。非数值运算种类繁多，要求各异，难以规范化，所以算法的研究基本上是依赖于具体研究领域的。因此，一般只对一些典型的非数值运算算法（例如排序算法）作比较深入的研究。其他的非数值运算问题，往往需要使用者参考已有的类似算法重新设计专门算法。

尽管不同的领域有各自的算法，但从算法所采用的方法或思路上看，最基本的大致可以分成以下几种：直接法、枚举法、递推法、递归法、回溯法、数字模拟，等等。下面先介绍直接法、枚举法、递推法、递归法的基本思想，其他一些方法将随着课程学习的深入逐步展开。

1. 直接法

直接法是指根据问题所给的条件，直接通过计算来得到解答。本章的算法 1 是直接算法的一个例子，通过计算直接得到两个数的绝对值之和。

2. 枚举法

枚举法又称穷举法，即通过逐一考察问题的所有可能解，来找出问题的真正解。枚举法应用的条件是问题的可能解必须是有限的，而且这些可能解必须是已知的。

【例 1.6】 给定一个正整数，确定它的整数的立方根是否存在，如果存在则找出这个立方根。

显然，一个正整数的立方根如果存在的话，那么肯定在这个数和零之间，而这之间的正整数的个数是有限的，所以可以设计一个基于枚举法的算法。

算法 5：

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* 输入：输入一个正整数                                                    *
* 输出：这个正整数的立方根（如果这个立方根存在的话）                      *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

    BEGING
        number ← 输入一个正整数
        result ← 0
        WHILE(result≤ number)
            BEGIN
                IF(result × result × result == number)
                    BREAK
                ELSE
                    result ← result+1
            END
        IF(result≤ number)
            result 是 number 的立方根，输出 result 的根
        ELSE
            输出 number 没有立方根的信息
    END
```

3. 递推法

递推法是从已知的初始条件出发，逐次推出中间结果，每一步在理想状态下应逐步接近问题的最后解。下面举一个简单的例子：

【例 1.7】 计算一个正整数的阶乘，即计算 N!。

根据数学知识，N! =1×2×3×…×N，所以可以从 1 开始，得到 1 的阶乘后乘 2 得到 2 的阶乘 2!，再和 3 相乘得 3! ……最后得 N!。

算法 6：

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* 输入：输入一个正整数                                                    *
* 输出：这个正整数的阶乘                                                  *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

    BEGING
        number ← 输入一个正整数
        resuot ← 1
        current ← 1
        WHILE(current ≤ number)
```

```
        BEGIN
            result ← result × current
            current ← current + 1
        END
    输出 result
END
```

阶乘的计算是递推法求解的一个典型的例子，其他的递推求解的例子还有求级数的和，等等。在实际的计算中，往往要注意递推中结果值的增长情况，例如，阶乘的增长就非常之快，10！＝3628800，13！就超过C语言中整数的最大允许范围。

递推法在数值计算中又称迭代法，迭代法经常用于求近似解的问题，根据对前一步结果误差的不同处理方法，迭代法又有逼近迭代和试探迭代等不同的方法。数值计算中要注意解的稳定性问题，即在迭代中每一步的解应越来越接近真正的解，否则迭代就不会成功。

4. 递归法

一个直接或间接调用自身的算法称为递归算法，一个使用函数自身给出定义的函数称为递归函数。在有些问题的算法描述中，递归往往比非递归法直接易懂。

例如，对正整数阶乘的问题，阶乘函数的递归定义为：

$0! = 1$

$N! = (N-1)! \times N$

第一式给出阶乘函数的初始值，初始值是非递归定义的。每个递归函数都需要给出非递归定义的初始值，否则这个函数无法计算出来。第二式用较小自变量的函数值来表达较大自变量的函数值，在定义式两边都引用了阶乘记号，所以是一个递归定义。

递归算法的描述和处理将在本书实验六“函数”部分介绍。

由上面的简单介绍已经可以看出，算法是解决问题的基础，为解决一个问题可能设计出不同的算法。那么什么是一个好的算法呢？衡量算法的质量指标包括算法的正确性、可读性、健壮性以及高的执行效率和低的存储空间要求。正确性是指算法对合法的输入应当能产生所要求的输出；可读性是指算法易于交流和理解；健壮性是指算法应在一定的范围内能对不合法的输入作必要的反应，而不致产生一些莫名其妙的结果；高的执行效率和低的存储空间要求是对算法在时间和空间上的效率要求。

本书不可能罗列所有的算法，只是希望通过一些典型算法的讨论，帮助读者了解如何设计一个算法，推动读者举一反三。希望读者通过这些例子了解怎样提出问题，怎样思考问题，怎样表示一个算法。

六、算法设计习题（用伪码、流程图、PAD图或N－S图表示）

1. 输入10个数，将其中能被3和5整除的数输出。
2. 求方程 $ax^2+bx+c=0$ 的根（分别考虑 b^2-4ac 大于0、小于0、等于0的情况）。
3. 用下面公式求 $\sin x$ 的值（x 的值设为3）：

$$\sin x = \frac{x}{1} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots - \frac{x^{4n-1}}{(4n-1)!} + \frac{x^{4n+1}}{(4n+1)!}$$

直到最后一项的绝对值小于 10^{-7} 时，停止计算。

第二部分 C程序的上机过程与UNIX系统使用

一、一般过程

C语言在不同的操作系统支持下，大致要经历三个阶段。

(1) 选择适当的正文编辑实用程序，将纸上或头脑中形成的程序，通过键盘输入到计算机中，以源程序文本形式存储于外存储器。

(2) 以源程序为输入源，由编译程序将源程序文件编译连接成为可执行文件。如果有错，返回到正文编辑实用程序修改文本，再投入编译。如此反复，直到正确为止。

(3) 将执行程序投入运行，获得计算结果。如果运行不成功(非正常停机，输出结果出错等)，返回编译阶段或编辑正文阶段。

对某些支持背景(例如DOS支持的Turbo C)由编译程序产生的目标文件不能直接执行，需先经过link连接多个目标文件，才能产生可执行文件。所以对C进行编译和运行，要参考支持背景的系统手册或有关使用说明。

C程序的一般上机过程可归纳如图2.1所示。

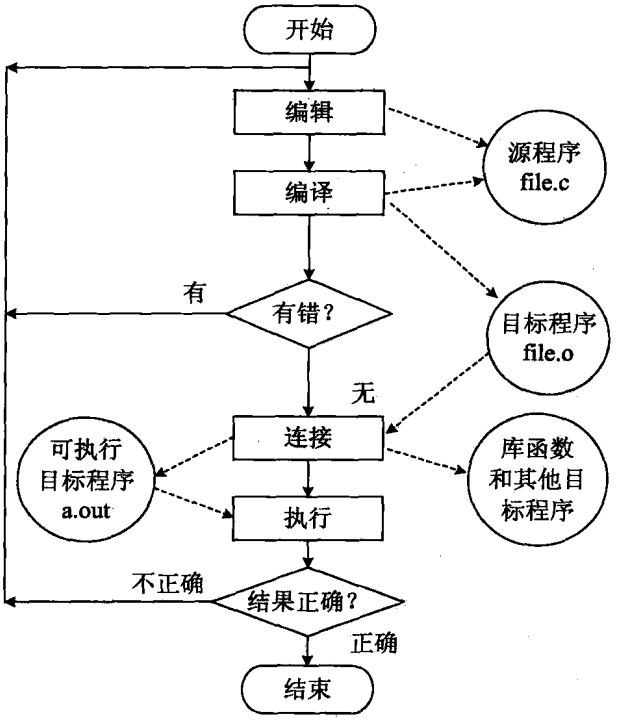


图2.1 C程序的上机过程

二、在 UNIX 系统上 C 语言程序的上机过程

在 UNIX 上，C 语言程序上机过程分为三阶段：编辑—编译—运行。C 程序可以由一个源程序文件也可以由多个源程序文件组成。通常做法为：

(1) 用 vi(或 ed 等)正文编辑实用程序将源程序输入系统，在用户当前目录之下建立一个源程序或若干个源程序文件。

(2) 用 C 语言的编译程序 cc 将源程序编译(可分多步进行)，最后产生可执行目标程序。

(3) 以可执行的目标文件名为命令名，执行该程序并获得预期要求的结果。

编译时或在运行执行程序时产生错误信息，可返回 vi(或 ed)正文编辑阶段或者回到编译程序 cc。

UNIX 系统下，C 语言程序的上机过程可归纳如图 2.2 所示。

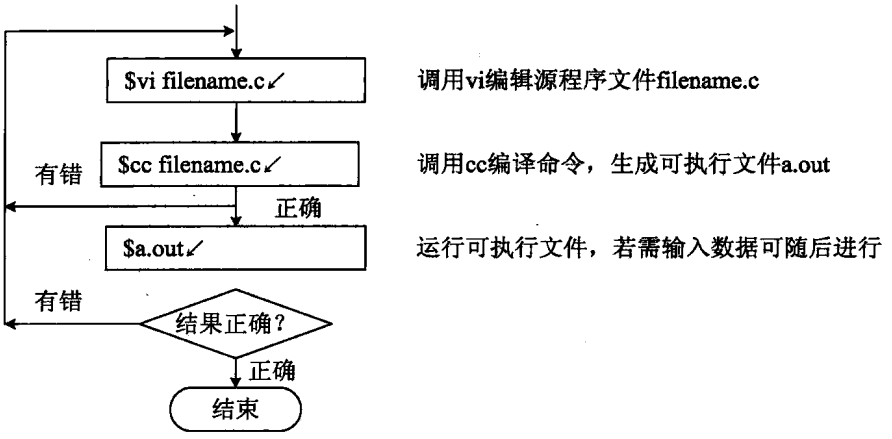


图 2.2　UNIX 系统 C 程序的上机过程

三、C 语言检查程序 lint

cc 编译程序虽然编译效率高，可一次编译成直接执行的目标文件。但 cc 对源程序的语法检查不够。为了解决程序检查问题，UNIX 上开发了称为 lint 的语言检查程序。它可以检查：

(1) 语法上的明显错误，如括号不配对等。

(2) 警告可能包含错误的地方，如数据类型定义与说明不一致等。

(3) 警告指出在此编译环境是正确的，但是如果将程序移植到其他系统则可能有要改动的地方。

lint 使用格式：

$ lint [可选项]　C 源程序文件名↙

lint 不是必须经过的手续，它的速度远比 cc 慢，不一定要用。它的使用场合是：

(1) 程序编译通过了，但运行执行文件时出现难以查找的错误，可利用 lint 提供大量警告性信息输出，协助分析错误出处。

(2) 一个程序要移植到其他系统，查看哪些可能地方要修改。

四、cc命令格式和使用

cc编译程序在UNIX上一次可以建立最终可运行的执行文件，但仍旧包含装配过程。利用可选项的选择，可以决定编译时要不要装配。对于由多个源程序文件组成的程序，分别编译时应选择不要装配的可选项，而只在最后编译时，加上此可选项。

1. 与cc编译有关的文件名格式(带后缀的格式)

文件名.c（带后缀.c)——用C语言编制的源程序

文件名.o（带后缀.o)——cc编译后的目标文件

a.out——编译时，如缺省执行文件指定，则在当前目录上自动产生可直接执行的执行文件a.out。各用户同名，均是a.out，但它的文件系统路径名不同，互相不干扰。

2. cc编译命令格式和功能

格式：

$cc [可选项] 文件名 …

功能：cc命令中的文件名可以是C语言源程序文件(带后缀.c)，也可以是目标文件(带后缀.o)，文件名可以多个。cc产生结果：如有可选项"－o"，产生最终执行文件；如果没有"－o"和"－c"可选项，产生可执行的执行文件并指定为a.out。一个程序(若干个源程序文件)可以进行多步的cc编译，最终形成执行程序。

cc命令可选项有十多种，这里仅指出最常用的两种：

－c——禁止编译时的装配操作，但产生的目标文件和源程序同名，其后缀改为.o，它再作为下次cc编译时的文件。凡带"－c"可选项的编译不是最终的编译。

－o ——紧接－o后的文件名是编译结果指定的执行文件。这时a.out保持不变，如果不指出－o可选项和紧接的文件名，则以a.out为执行文件，原来a.out文件的内容被覆盖。

3. cc编译命令格式示例

设一C程序由四个源程序文件组成：prg1.c，prg2.c，prg3.c，prg4.c 。下面以它为例说明几种编译命令的格式。

(1) 一次编译并以a.out为执行文件：

$cc prg1.c prg2.c prg3.c prg4.c↙

(2) 一次编译并以prg为执行文件：

$cc prg1.c prg2.c prg3.c prg4.c －o prg↙

(3) 多次编译方式之一，以a.out为执行文件：

$cc －c prg1.c↙

$cc －c prg2.c↙

$cc －c prg3.c↙

$cc －c prg4.c↙

$cc prg1.o prg2.o prg3.o prg4.o↙

(4) 多次编译方式之二，以a.out为执行文件：

$cc －c prg1.c prg2.c prg3.c prg4.c↙

$cc prg1.o prg2.o prg3.o prg4.o↙

(5) 多次编译方式之三，仍以a.out为执行文件：

$cc －c prg1.c prg2.c↙

```
$ cc prg1.o prg2.o prg3.c prg4.c↙
```

(6) 多次编译方式之四，指出 prg 为执行文件：

```
$ cc -c prg1.c prg2.c↙
$ cc prg1.o prg2.o prg3.c prg4.c -o prg↙
```

其他使用方式这里不再一一列出。

在不同操作系统或者不同 UNIX 版本(例如它的变种 XENIX)上，cc 编译程序可能在命令写法和形式上有细微差别，使用时应遵循相应的使用格式要求。

特别提示：如源程序文件中包含 #include <math.h>说明，则要求在编译源程序文件时的 cc 编译命令中加-lm 选项。如：

```
$ cc prg1.c prg2.c prg3.c prg4.c -lm↙
```

五、C 语言程序的运行

运行一个 C 语言程序可以分两种情况来说明：

(1) 以 a.out 为执行文件名时，以 a.out 为命令名执行：

```
$ a.out↙
```

考虑到 a.out 是用户目录中唯一的也是最活跃的文件，在多次编译不同程序时，可用 UNIX 的复制命令 cp 将刚生成的 a.out 复制保护起来，例如：

```
$ cp a.out prg↙
```

然后执行：

```
$ prg↙
```

(2) 如已有指定的执行文件名时，直接执行以此文件名为命令名的任务。如：

```
$ prg↙
```

六、编译实例

1. 要求

设计一个最简单的 C 语言程序，它有两个源程序文件组成。每个源程序只含一个函数，它只有 printf 格式输出语句。

2. 程序清单

prg1.c 源程序文件：

```
#include <stdio.h>
main(){
    printf("The UNIX System.\n");
    prt();
}
```

prg2.c 源程序文件：

```
#include <stdio.h>
prt(){
    printf("The C Programming Language.\n");
}
```

3. 上机过程

(1) 利用 UNIX 的 vi 正文编辑程序编辑 prg1.c 和 prg2.c 文件，保存在用户自己的当前

目录中。

(2) 发出编译命令：

$ cc prg1. c prg2. c ↙

(3) 运行 a. out 执行文件：

$ a. out ↙

(4) 结果输出：

The UNIX System.

The C Programming Language.

4. 建议

可有意地在源程序中安排一些错误，了解 cc 编译时的反映；再用 lint 命令检查源程序，了解它的反映，并分析出错误信息。如果有兴趣也可以试用其他编译方式。

七、上机操作示意图

UNIX 系统下 C 语言的上机操作可归纳如图 2.3 所示，具体 UNIX 命令可参阅本书后续内容(详见本部分“八、UNIX 操作系统的基本命令及其使用”)。

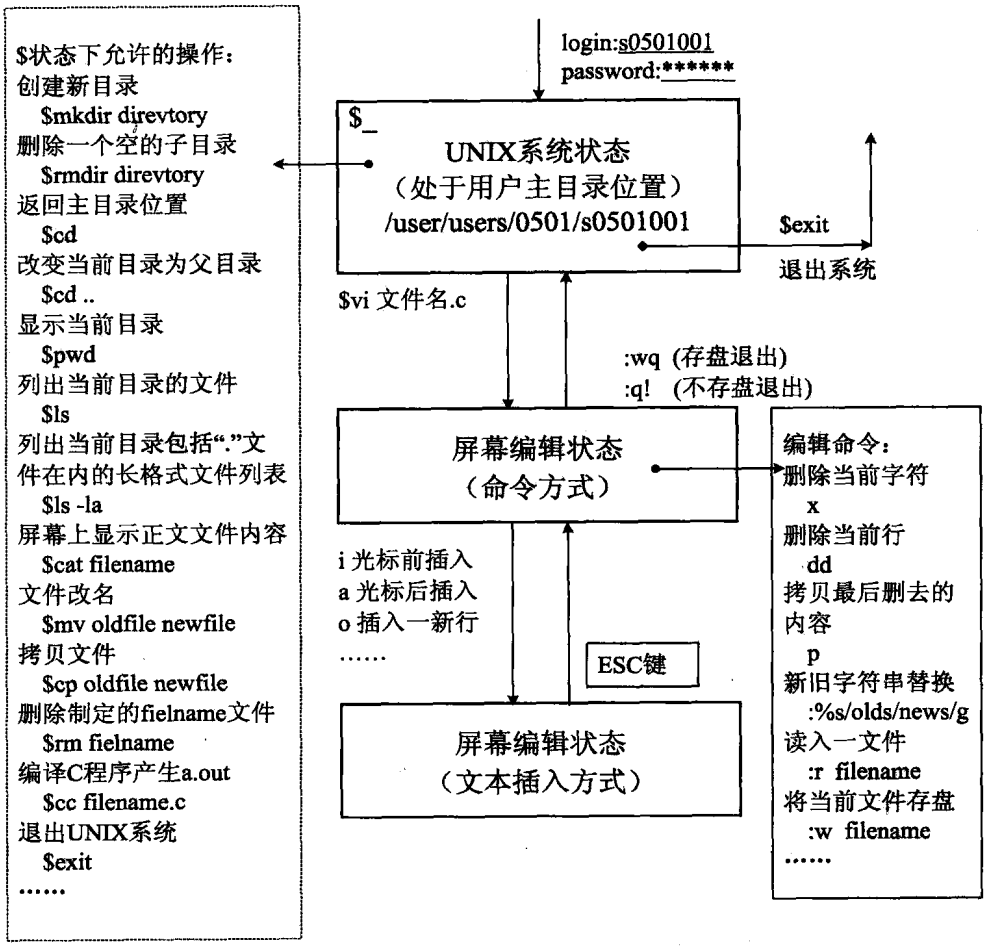


图 2.3　上机操作过程、状态与功能示意图

八、UNIX 操作系统的基本命令及其使用

UNIX 系统是世界上应用最广泛的一种多用户、多任务、交互式的操作系统，它为用户提供了一个简洁、高效、灵活的运行环境。该系统最早由美国 AT&T 公司 Bell 实验室于 1969 年开发。由于 UNIX 操作系统是用 C 语言开发的，可移植性强，促进了它的推广；同时随着计算机技术，特别是硬件的迅速发展，使该系统得到不断的发展和完善。现在不少计算机公司及科研机构都推出自己的 UNIX 操作系统，常见的有：美国加州大学伯克利分校的 UNIX 系统 V，Santa Cruz Operation 公司的 SCO UNIX 操作系统，惠普公司的 HP UNIX 操作系统等。在国外，UNIX 操作系统占据着计算机操作系统的主导地位。

（一）系统登录和退出系统

UNIX 系统是一个多用户系统。所谓多用户，就是允许多个用户同时使用一个计算机系统。每个用户通过唯一的账号名或注册名加以区分，在访问系统时，用户必须给出他的注册名和相应的口令，所以在你注册进入系统之前，必须从负责维护系统的系统管理员那里得到你所使用账号的注册名和注册时需要的口令。

1. 注册进入系统

在使用 UNIX 操作系统以前，必须进行系统登录。注册进入系统有若干步骤，每一步都必须正确完成，否则你就不能进入系统。

当系统启动后，你打开终端，可看到以下提示：

login：

这时输入你的注册名，并按〈Return〉键（以后命令行中以“↙”表示）确认。输入注册名之后，还必须给出正确的口令。口令只有你本人知道。当看到下面的提示时：

Password：

输入你的口令，为了保密起见，输入的口令并不在屏幕上显示。如果你输入的注册名和口令与事先登记在系统中的注册名和口令相吻合，你就完成了登录过程。

例如，某人在系统中有一个账号，其注册名为“s0501001”，则注册过程如下所示：

login：s0501001↙（带下划线的为用户输入部分）

password：* * * * * * *↙

…… 系统提示信息 ……

$

见到 UNIX 系统提示符“$”，表示你已注册成功进入了系统，这时就可以输入 UNIX 操作系统命令了。否则将在屏幕上显示一条错误消息：

Login incorrent

这表示你打错了注册名或口令，此时需要重新输入注册名和口令。

2. 退出系统

注册进入系统，完成工作后，在你离开之前，应当从系统中退出。我们把从系统中退出的过程称为注销。

（1）在多数情况下，要退出系统，可打入：

$ exit↙

或在系统提示符“$”状态下按热键〈Ctrl〉+d。

(2) 如果 exit 不存在,或按〈Ctrl〉+d 不能退出系统,可键入:

$ logout ↙

这时,系统将显示:

login:

这表示你已经从系统中注销了。

(二) 更换口令

口令对用户的账号起着保护作用,它可以防止别人非法侵入你的注册账号,造成泄密或其他损失。最初的口令是由系统管理员在建立账号时为用户设置的。当用户第一次注册进入他自己的账号后,就可以更换这个口令。另外,为了防止别人猜到你所使用的口令,应该定期或不定期地更换口令。

口令由一串大写、小写字母和数字序列组成,而且不宜过短。选择口令时应掌握两个原则:一是自己容易记住;二是别人不容易猜到。一般应尽量避免使用自己的注册名以及自己的名字或生日、住址等作为口令,因为这样的口令比较容易被别人猜出。

使用 passwd 命令更换口令的过程如下:

$ passwd ↙

如果你原来有口令,系统会提示以验证你的身份:

Changing password for s0501001

Old password: * * * * * * * ↙ 输入原有口令

若原有口令输入正确,系统会提示:

New password: * * * * * * * * ↙ 输入新口令

为了避免你在打入新口令时偶然出错,命令将再一次要求给出新口令:

Re-enter your new password: * * * * * * * * ↙ 输入新口令

$

如果两次输入的新口令一致,这个新口令就被系统接受了,下次再注册进入系统时,就要使用这个新口令。

注意: 必须妥善保管好你的口令,不要丢失或忘掉,一旦丢失或忘掉了你的口令,就要请系统管理员帮助了。

(三) 使用 vi 编辑器编写文件

UNIX 操作系统提供的 vi 编辑器是一个功能很强的文件编辑程序,直观方便。

1. 三种操作方式

在 vi 编辑状态下,有以下三种不同的操作方式:

1) 命令方式

当用户调用 vi 编辑程序后,就直接进入了命令方式。在此方式下,用户只能输入 vi 中规定的命令,而不能进行文件内容的输入操作。

2) 输入方式

用户在 vi 命令方式后,执行某些命令,如 i,a 等,就可以进入输入方式。在此方式下,用

户的所有输入都将作为文件的内容存入文件中。

3) 最后行命令方式

用户在 vi 命令方式输入“:”,即可进入最后行方式。此时光标将跳到屏幕上最后一行,显示“:”符,提示用户输入命令。

vi 的三种操作方式,每一种都有其独立的命令集。在不同的方式下,用户输入的字符含义也将随之变化,因此用户应当很清楚当前是在何种方式下操作。

2. 三种操作方式的转换

这三种操作方式之间的相互转换关系如图 2.4 所示。

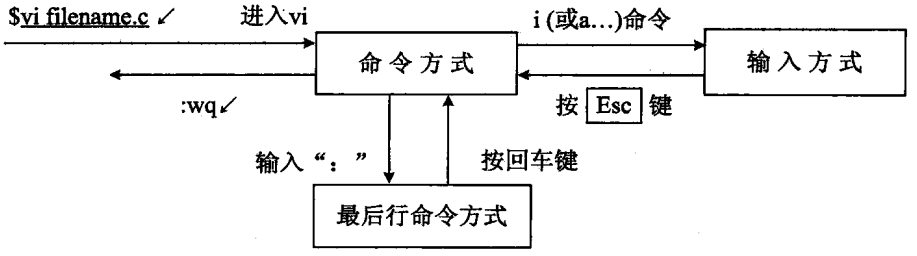


图 2.4　vi 的三种操作方式之间的相互转换

3. vi 的界面

键入命令:

$ vi filename. c ↙

即进入 vi,屏幕显示如图 2.5 所示。

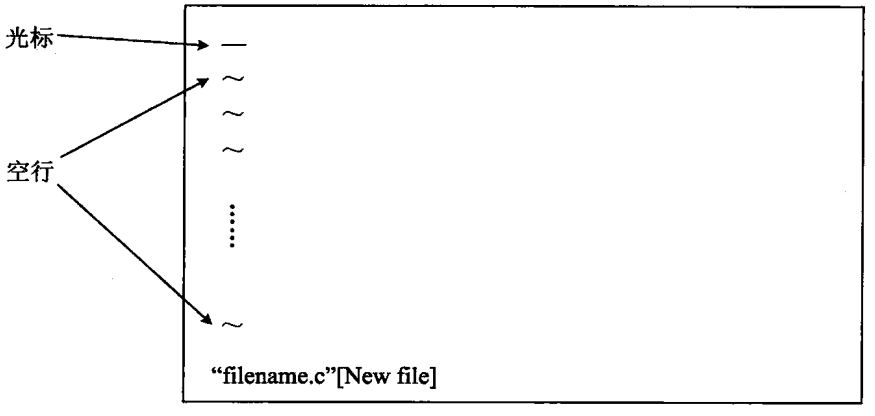


图 2.5　vi 屏幕画面

当[filename. c]是新建立时,在画面的左端将显示“—”,有“~”的行表示空行。在画面的最下面的一行显示文件名及其状态。此时,处于命令状态。按 i 进入输入状态(i 命令表示“在光标之前插入”),就可以输入文本了。当更新指定的[filename. c]时,画面将显示其原有的内容,你可以对其修改更新。

4. vi 中各种方式下的主要命令

命令方式下主要操作命令如表 2.1 所示。

表 2.1 命令方式下主要命令

移动光标	修改操作
l 右移一格 h 左移一格 j 下移一行 k 上移一行 $ 移到行尾 0 移到行首	r 替代当前字符 s 用一个串替代当前字符 cc 替代当前行中的所有字符 cw 用新的正文替代一个词或一个词的剩余字符
屏幕控制	**进入输入方式**
〈Ctrl〉+d 向下滚半屏 〈Ctrl〉+u 向上滚半屏 〈Ctrl〉+f 显示下一屏 〈Ctrl〉+b 显示上一屏 注:〈Ctrl〉+d 操作指同时按下〈Ctrl〉和 d 两个键	i 在光标前插入字符 a 在光标后插入字符 o 在当前行的下面创建新行,并在新行插入字符 I 在行首插入字符 A 在行尾插入字符 O 在当前行的上面创建新行,并在新行插入字符
删除操作与复制	**其他功能操作**
x 删除光标所在位置上的字符 dw 删除词中光标至下一个空格或标点之前的部分 dd 删除光标所在行(即当前行) ndd 删除以当前行为起点的 n 行 p 将缓冲区中内容复制到光标之后 yy 复制当前行 nyy 复制 n 行	u 废除前一次操作 ZZ 存盘,退出 vi 编辑程序 〈Esc〉键 取消命令 : 进入最后行命令方式

输入方式下主要操作命令如下:

〈Backspace〉键　　删除字符,每按一次,即向前删除一个字符

〈Esc〉键　　退出插入方式

最后行方式下主要操作命令如下:

:　　冒号是最后行方式的提示符

:w　　将文件内容存盘

:wq　　将文件内容存盘,之后退出 vi 编辑状态,然后回到 UNIX 系统命令状态

:q　　文件内容不存盘,退出 vi 编辑状态,回到 UNIX 系统提示符状态

:q!　　文件内容不存盘,强制退出 vi 编辑状态,回到 UNIX 系统提示符状态

:! 命令　　执行 shell 命令

:! who　　显示在系统上登录的用户

:%s/旧字符串/新字符串/g　　新旧字符串替换

:r 文件名　　读入一个指定的文件,插入到光标所在位置

:w 文件名　　将当前文件内容写入指定文件

(四) 目录操作

(1) 显示当前目录:

$ pwd ↙

(2) 改变当前目录：

$cd 〈目录名〉↙

(3) 建立目录：

$mkdir 〈目录名〉↙

(4) 删除目录：

$rmdir 〈目录名〉↙

注：如果要删除一个目录，在删除这个目录之前，必须把这个目录中的文件和子目录全部删除，否则不能删除这个目录。

(5) 显示目录文件清单：

$ls [参数] [名字]↙

其中，[名字]可以是文件名或目录名，若是文件名，则显示所指定文件的信息，但不涉及目录的具体内容。若[名字]是目录名，则显示该目录中所有文件信息。若名字部分缺省，则显示当前目录下所有的文件信息。[参数]是对命令功能的限定和扩展，使用不同的参数，文件的清单内容也不同。ls 命令的主要参数如表 2.2 所示。

表 2.2 ls 命令的主要参数

参 数	功 能
-a	显示目录中的所有文件
-l	产生一个长格式文件列表，内容包括文件的存取方式、连接数、属主、组、字节大小和最近一次修改时间

例如，要列出文件名后缀为“.c”的所有 C 程序文件，可执行命令：

$ls *.c↙

要以长格式形式列出当前目录的所有文件信息，可执行命令：

$ls -la↙

(五) 文件操作

(1) 文件拷贝：

$cp 〈源文件名〉〈目的文件名〉↙

功能是将源文件内容复制到目的文件。

(2) 显示文件内容：

$cat 〈文件名〉↙

$more 〈文件名〉↙

以上两个命令都可以显示指定文件内容，区别在于前者不分页显示，后者在显示满一页时就暂停。

(3) 文件移动：

$mv 〈源文件名〉〈目的文件名〉↙

mv 命令用来移动文件，或者进行文件名的变更。这一命令的行为类似于 cp 命令，不同之处在于 mv 删除源文件，而 cp 则保留源文件。

(4) 文件删除：

$ rm〈文件名〉[〈文件名〉……]↙

rm 命令能够一次删除多个文件。必须注意，被删除的文件无法恢复。

(六) 显示系统状态

(1) 显示系统时间：

$ date↙

(2) 显示系统中的用户：

$ who↙

(3) 显示用户进程：

$ ps↙

UNIX 是一个功能强大、性能全面的多用户、多任务操作系统，可以应用从巨型计算机到普通 PC 机等多种不同的平台上，是应用面最广、影响力最大的操作系统。

以上简单介绍了一些常用的 UNIX 命令，更详细的内容请查阅 UNIX 操作系统的相关技术资料。

九、Linux 系统上机过程

Linux 是一种外观和性能与 UNIX 相同的操作系统，是一个类似于 UNIX 的产品。Linux 产品成功地模仿了 UNIX 系统和功能。一般来说，Linux 是一套遵从 POSIX(可移植操作系统环境)规范的一个操作系统，它能够在普通 PC 计算机上实现全部的 UNIX 特性，具有多任务、多用户的能力。Linux 受到广大计算机爱好者喜爱的主要原因之一是它具有 UNIX 的全部功能，任何使用 UNIX 操作系统或想要学习 UNIX 操作系统的人都可以从 Linux 中获益。

(一) Linux 系统上机过程 1

设 Linux 服务器 IP 地址：192.168.11.252，端口：22。

(1) 从开始菜单选择执行客户端软件：Xmanager Enterprise → Xshell，如图 2.6 所示。

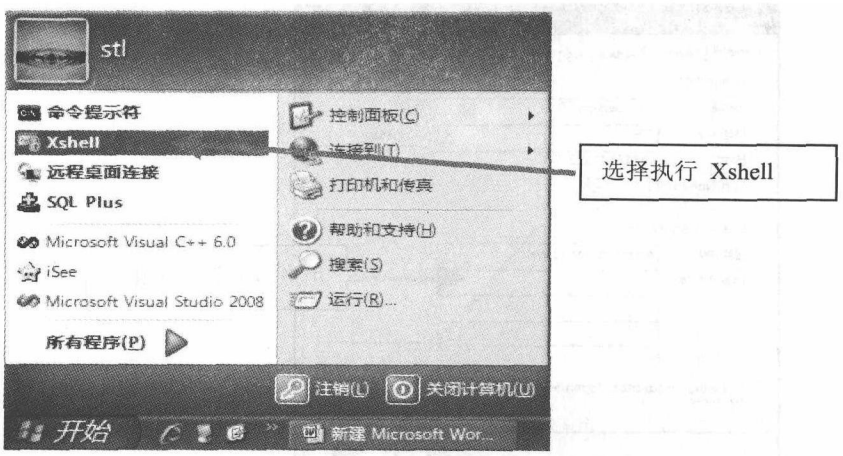


图 2.6　选择执行客户端软件 Xshell

Xmanager 是一个运行于 MS Windows 平台上的高性能的 X Window 服务器。你可以在你的本地 PC 上同时运行 UNIX/Linux 和 Windows 图形应用程序。

Xshell 是一个用于 MS Windows 平台的强大的 SSH，TELNET 和 RLOGIN 终端仿真软件。它使得用户能轻松和安全地从 Windows PC 上访问 UNUX/Linux 主机。

(2) 进入 Xshell 界面，如图 2.7 所示。

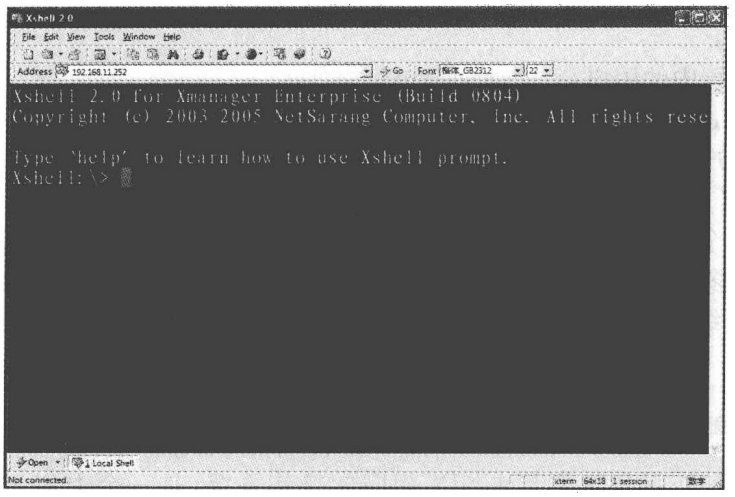


图 2.7　进入 Xshell 界面

(3) 新建连接：在主菜单中选择 File→New，对各属性对话框作相应设置。

New Session 属性对话框中的 Generalye 页面设置：

Methods：SSH

Host(填写服务器 IP 地址)：192.168.11.252

Port Number (端口号)：22

User Authentication 栏中的 Method：Keyboard Interactive

新建连接如图 2.8 所示。

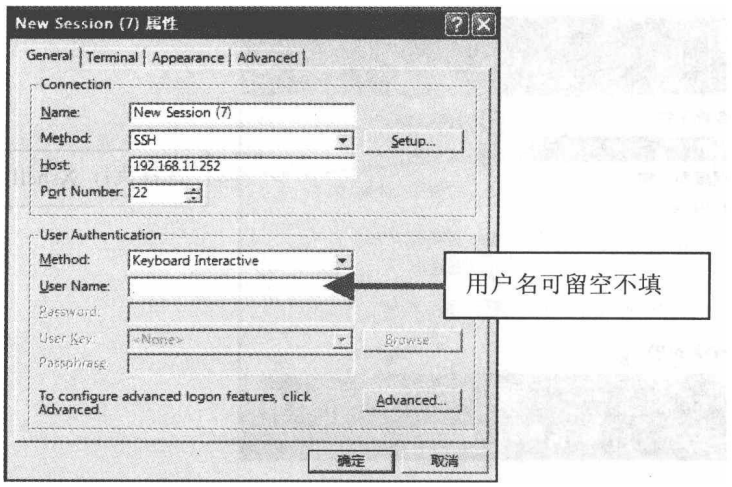


图 2.8　新建连接

New Session 属性对话框中的 Advanced 页面设置：

Enconding(选择语言)：utf8 — Chinese Simplified

Advanced 页面设置如图 2.9 所示。

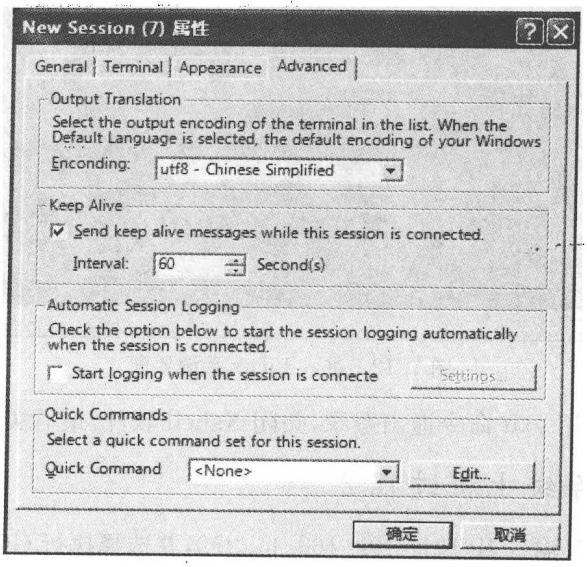


图 2.9　Advanced 页面设置

完成后，双击或选中，点击 Connect 进入登录 Linux 过程。

(4) 出现并在 SSH User Name 窗口输入账号。输入账号操作如图 2.10 所示。

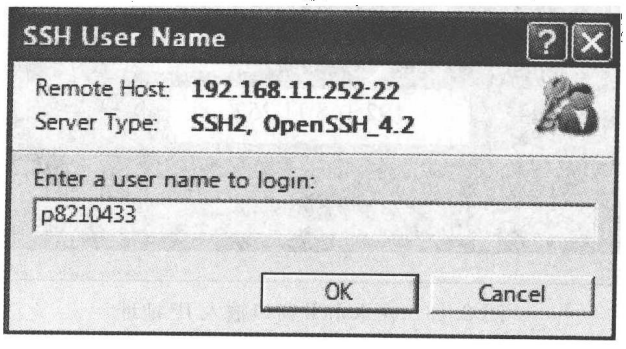


图 2.10　输入账号

(5) 出现并在 SSH User Authentication 窗口输入 Password，如图 2.11 所示。

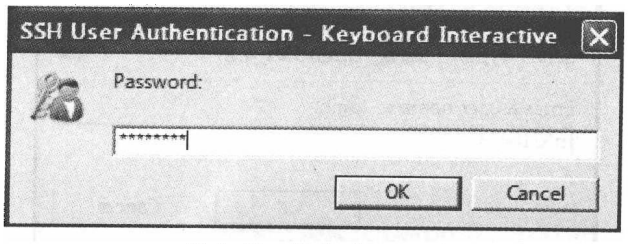


图 2.11　输入 Password

(6) 成功进入 Linux 系统，如图 2.12 所示。

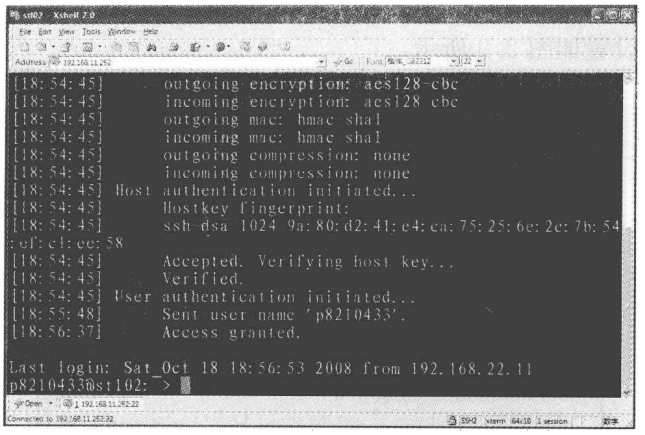


图 2.12　进入 Linux 系统界面

(7) 实验结束时，用 exit 命令退出登录，关闭 Xshell 软件，最后关机。

(二) Linux 系统上机过程 2

(1) 在 Xshell 窗口输入 IP 地址 192.168.11.252，并选择执行 GO，如图 2.13 所示。

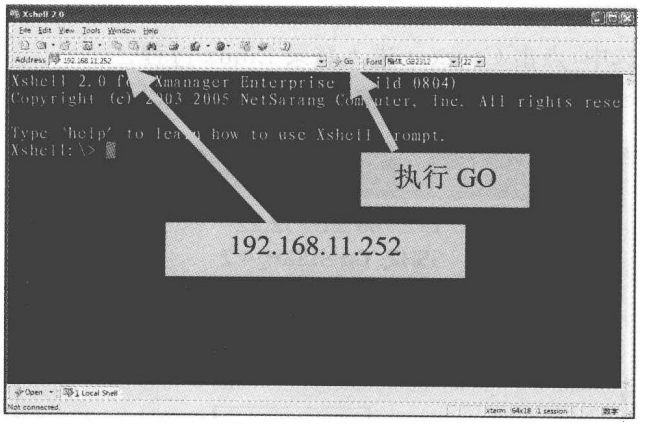


图 2.13　在 Xshell 窗口输入 IP 地址

(2) 在 SSH User Name 窗口输入账号，如图 2.14 所示。

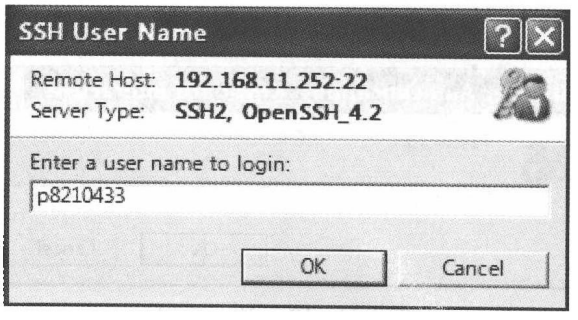


图 2.14　输入账号

(3) 选择键盘交互方式(Keyboard Interactive)选项，如图 2.15 所示。

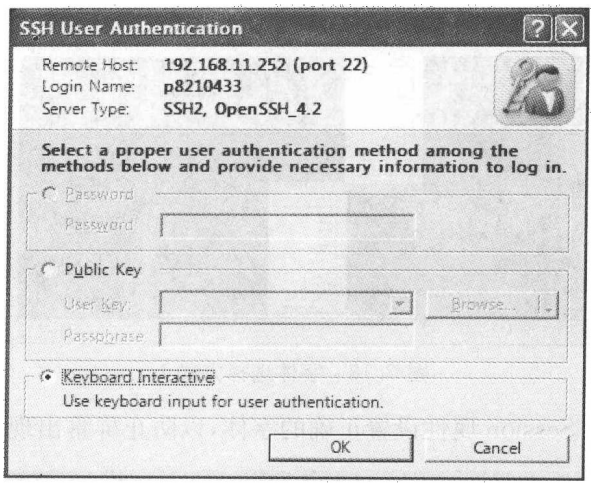


图 2.15　选择键盘交互方式

(4) 输入 Password，如图 2.16 所示。

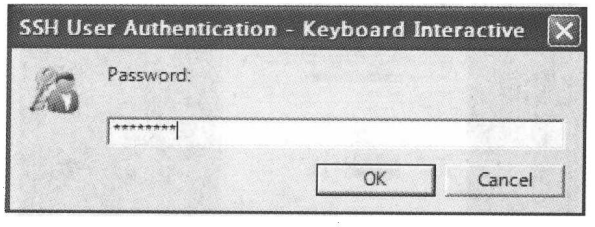


图 2.16　输入 Password

(5) 成功进入 Linux 系统，如图 2.17 所示。

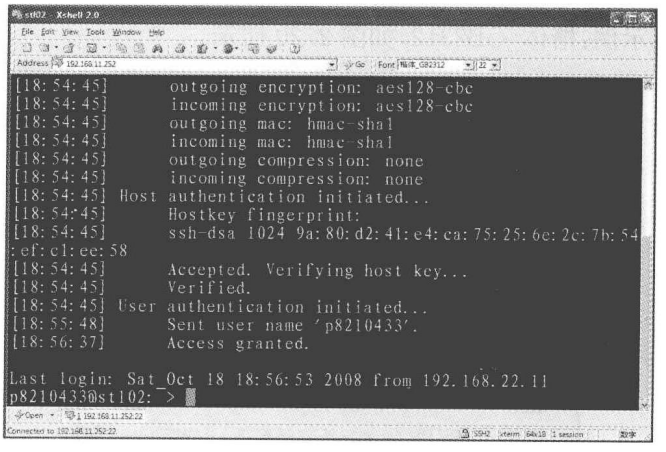


图 2.17　进入 Linux 界面

(6) 在工具栏通过选择输出编码选项(Output Encoding)设置正确的字体，以防止屏幕出现乱码。

正确字体可选 Unicode(NTF8)统一的字符编码标准，采用双字节对字符进行编码→→

(Chinese Traditional)传统的中文方式。如图 2.18 所示。

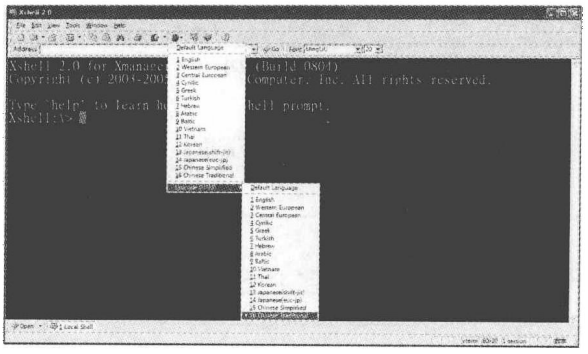


图 2.18　字体选择方式 1

也可通过 Default Session 属性设置正确的字体，以防止屏幕出现乱码，如图 2.19 所示。

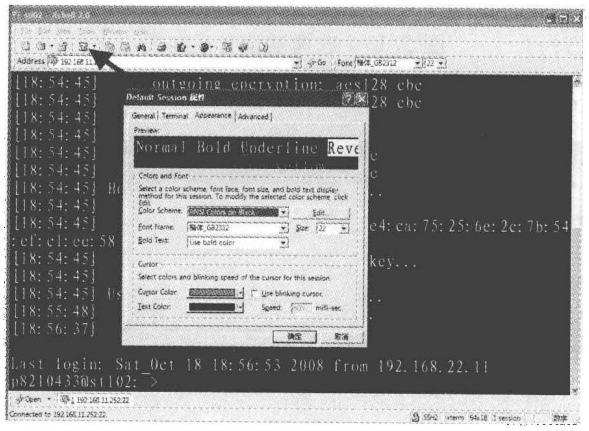


图 2.19　字体选择方式 2

例如，正确字体示例：楷体_GB2312，如图 2.20 所示。

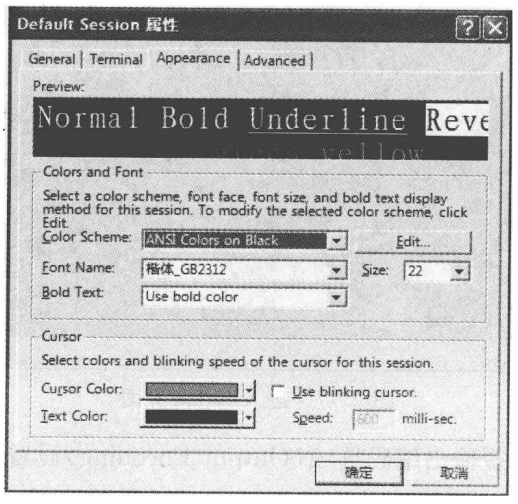


图 2.20　字体选择方式 3

Linux系统中的基本命令及其使用与UNIX系统基本相同。

十、Turbo C++ 3.0的集成开发环境与程序调试

对C和C++程序员而言，Turbo C++ 3.0也是一个功能强大的优化编译器。Turbo C++ 3.0是ANSI C和AT&T C++ 2.1的完整实现，因此C用户也可在Turbo C++ 3.0环境下作程序开发工作。下面对Turbo C++ 3.0的使用方法给予较详细的介绍。

(一) Turbo C++ 3.0的启动、退出和窗口切换

1. 字符命令方式的启动(设Turbo C++ 3.0集成开发程序安装在d:\tc\bin位置)

启动步骤如下：

(1) 依次执行"开始 / 程序 / MS-DOS方式"。

(2) 改变当前目录为d:\tc\bin：

C:\>cd d:\tc\bin ↙

(3) 执行Turbo C++ 3.0集成开发程序：

D:\TC\BIN>tc ↙

(4) 出现Turbo C++ 3.0集成环境窗口(参见图2.21)。

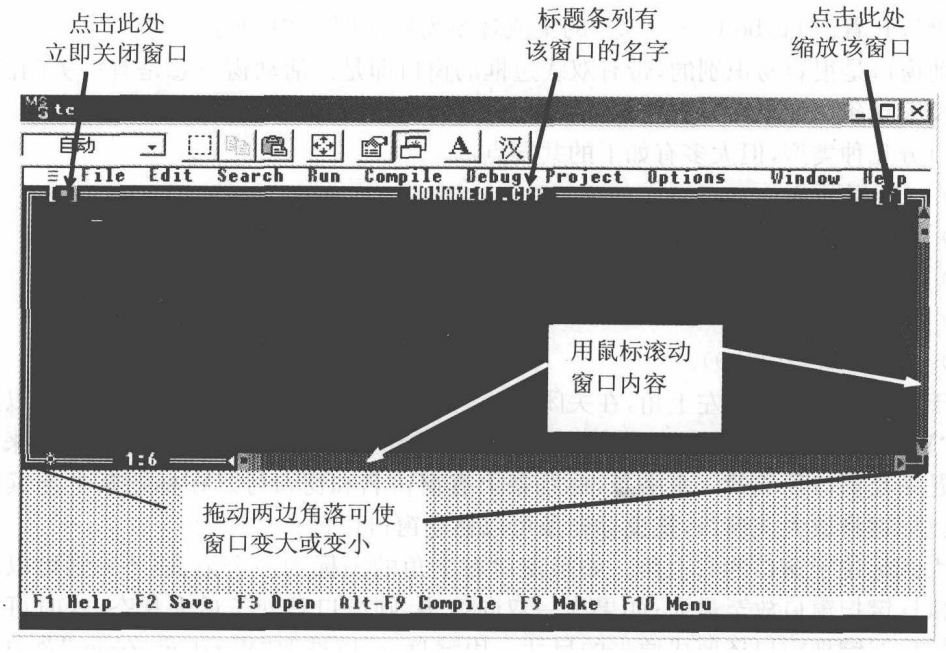


图2.21　一个典型的窗口

2. 图形界面启动

图形界面的启动步骤如下：

(1) 通过"我的电脑"或资源管理器进入D:\TC\BIN目录。

(2) 双击tc.exe程序图标(执行Turbo C++ 3.0集成开发程序)。

(3) 出现Turbo C++ 3.0集成环境窗口(参见图2.21)。

3. Turbo C++ 3.0 集成环境窗口方式和全屏幕方式的切换

使用热键〈Alt〉+〈Enter〉可以实现 Turbo C++ 3.0 集成环境窗口方式和全屏幕方式的切换。

4. 退出 Turbo C,返回 Windows

具体步骤为:

(1) 执行菜单命令"File/Quit",或直接按热键〈Alt〉+X,结束 Turbo C++ 3.0 集成开发程序的运行。

(2) 推出 MS—DOS 方式,返回 Windows:

D:\TC\BIN>exit↙

(二) Turbo C++ 3.0 的集成环境窗口

集成环境窗口是屏幕的一块区域,可以打开、关闭、移动、改变大小、放大、分列、叠盖。集成环境可以打开多个窗口,但在任何时候只能有一个窗口是活动的,活动窗口是当前用户正在上面进行操作的窗口,选取的命令或键入的正文一般只用于活动窗口(如果在几个窗口中打开同一个文件,对该文件的操作将反应在所打开该文件的窗口中)。

如果用户在退出 Turbo C++ 3.0 时,某个窗口中有一个文件没有关闭,那么,当下次用户重新进入 Turbo C++ 3.0 时,用户的工作面将与上次退出时一样,那些原先没有关闭的文件仍打开着。Turbo C++ 3.0 的集成环境窗口如图 2.21 所示。

活动窗口是很容易识别的,带有双线边框的窗口即是。活动窗口总是有一关闭图标、放大图标及滚动条。如果窗口有叠盖,活动窗口总是位于所有窗口的顶端(即最前面)。

窗口分几种类型,但大多有如下的共同点:

(1) 一个标题条。

(2) 一个关闭窗口。

(3) 滚行条。

(4) 一个放大图标。

(5) 一个窗口号(1~9)。

关于图标位于窗口的左上角,在关闭图标上按鼠标键能快速关闭该窗口(也可以选取菜单命令"Window/Close")。检查窗口和求助信息被看作是临时窗口,可以按〈ESC〉来关闭。

标题条是窗口顶端的一水平条,标题条上有窗口名和窗口号。在标题条上连续两次快速按动鼠标键将放大该窗口,拖曳标题条可以移动窗口位置。

放大图标位于窗口的右上角。如果窗口右上角的图标为一个箭头(↑)则可以在该图标上按鼠标键把窗口放至最大;如果为一双向箭头,则窗口已处于最大状态,这时可以在该图标上按鼠标键把窗口还原成原先的尺寸。用键盘,可以选取"Window/Zoom"菜单命令放大窗口。

Turbo C++ 3.0 中先打开的 9 个窗口在其右上角有窗口号,按〈Alt〉+0 可以查已经打开了哪些窗口。按〈Alt〉+窗口号组合键可以激活对应的窗口,例如,如果求助信息窗口的窗口号是 5,但被其他窗口盖住了,按〈Alt〉+5 将把信息窗口调至最前面。

在滚动条上按鼠标键可以使窗口中的内容滚动。在滚动条两头的箭头号上按鼠标键每次滚动一行(按住鼠标键将连续滚行),在滚行条指针两侧的阴影区域上按鼠标键每次滚动

一页，直接将滚动条滑块拖曳至某一位置可以快速使窗口中的文件作相应移动。

拖曳窗口的4个角都可以使窗口变大或变小，通过菜单命令“Window/Size/Move”可重定义窗口大小。

编辑窗口在左下角还显示当前行号、列号，如果对文件作了修改，行号、列号的左边将出现一个星号“*”。

（三）Turbo C++ 3.0的集成环境中C程序的调试过程

Turbo C++ 3.0集成环境中C程序的调试过程可归纳如图2.22所示。

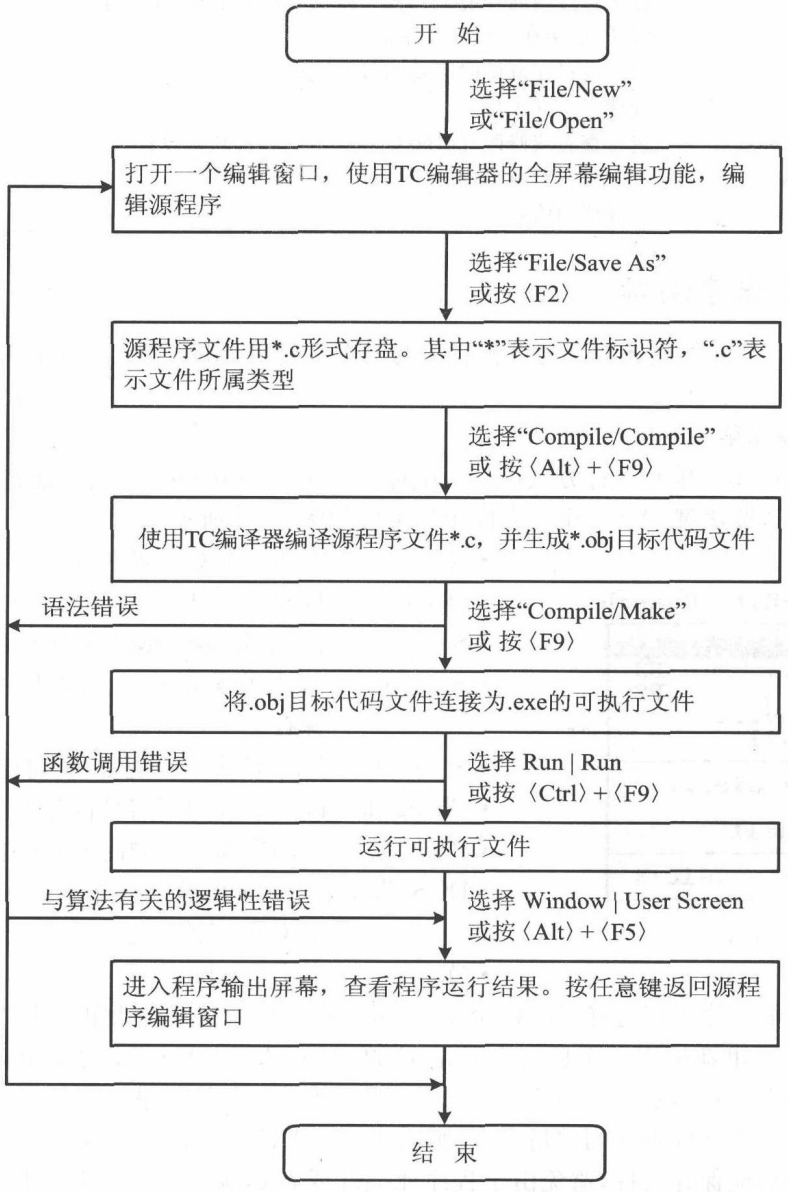


图2.22　Turbo C++ 3.0的集成环境中C程序的调试过程示意图

（四）主菜单中的项目与功能

Turbo C 3.0 集成环境主菜单的项目与功能如表 2.3 所示。

表 2.3　Turbo C 3.0 主菜单中项目及其功能简介

主菜单项目	功　　能
File	包含文件的各项基本操作、目录管理、进入 DOS 和退出 Turbo C
Edit	包含编辑的基本命令：复制、粘贴、剪贴以及撤销和重复操作
Search	包含对文件内容的查找、替换操作
Run	包含运行当前环境下的程序以及开始或结束程序调试的命令
Compile	编译、连接在当前环境下的源程序
Debug	用于设置在调试时需要的各种选择
Project	管理多文件工程
Options	包含查看或修改 Turbo C 环境下的各种默认设置
Window	包含所有对窗口操的作命令
Help	提供帮助信息

（五）源程序编辑

Turbo C 3.0 的集成环境下与编辑有关的操作主要由 File、Edit、Search 菜单提供，下面给予详细介绍。

1. 文件操作菜单 File

打开文件操作菜单有 3 种方法：第一，鼠标单击 File 菜单；第二，按功能键〈F10〉后选择 File 菜单；第三，按热键〈Alt〉＋F。文件操作菜单如图 2.23 所示。

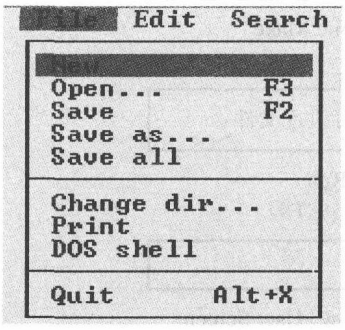


图 2.23　File 子菜单

- New　新建一个 C 的源程序。
- Open　打开一个已经存在的 C 源程序。
- Save　保存当前编辑窗口的源程序。
- Save as　将当前编辑窗口的源程序重新命名并保存。
- Save all　保存 C 集成环境下所有打开的源程序。
- Chang dir　改变系统的当前盘符和目录。
- Print　打印当前窗口的内容或信息。
- DOS Shell　进入 DOS 状态。按 Exit 可以返回 Turbo C 的当前状态。
- Quit　退出 Turbo C。

新建一个源程序可以选择“New”命令，也可以在当前程序处修改再选择“Save as”重新改名存盘。如果重新编辑一个已经存在的 C 源程序，选择“Open”命令打开，弹出“Open a File”对话框，修改默认的“ * . cpp”为“ * . c”，则列出所有当前目录下的扩展名是“. c”的文件。如果要打开的文件不在当前目录下，则需要输入盘符路径和文件名。

编辑完成后应保存文件，避免由于程序死循环或者其他意外造成无法返回而进行存盘操作。如果是修改源程序，只要选择“Save”即可，如果是新程序，需要选择“Save as”命令。

建议用户在保存文件前，最好建立一个自己的文件夹，专门用于存放C程序。

2. 编辑操作菜单 Edit

鼠标单击 Edit 菜单，或按功能键〈F10〉后选择 Edit 菜单，或按热键〈Alt〉＋E 键，均可打开 Edit 菜单的下拉子菜单，如图 2.24 所示。

- Undo　撤销最后一个编辑操作。
- Redo　重复撤销的操作。
- Cut　剪切鼠标或键盘选中的内容。
- Copy　复制鼠标或键盘选中的内容到剪贴板上。
- Paste　粘贴剪切板上内容到光标所在位置。
- Clear　删除鼠标或键盘选中的内容。
- Show clipboard　显示剪贴板上的内容。

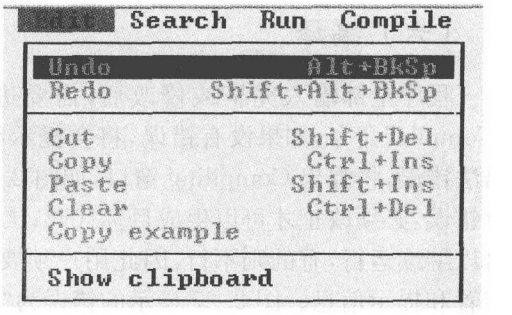


图 2.24　Edit 子菜单

其中选中的内容：指拖动鼠标形成的高亮度区域或按住 Shift 键并同时移动光标形成的高亮度区域。

除了上述通过菜单形式给出的编辑命令外，对集成环境下一些常用的全屏幕编辑操作也应有一定的了解。

1）光标移动

- ←、→、↑、↓　光标左移一个字符、右移一个字符、上移一行、下移一行。
- 〈Home〉、〈End〉　光标移动到行首、移动到行尾。
- 〈Ctrl〉＋←(或〈Ctrl〉＋F)、〈Ctrl〉＋→(或〈Ctrl〉＋B)　光标向左右移动一个词。
- 〈PgUp〉(即 Page Up)、〈PgDn〉(即 Page Down)　光标上移、下移一屏。
- 〈Ctrl〉＋QE、〈Ctrl〉＋QX　光标移动到窗口顶部、底部。
- 〈Ctrl〉＋QR、〈Ctrl〉＋QC　光标移动到文件首、尾。

2）删除和插入

- 〈Aackspace〉　删除光标前一个字符。
- 〈Del〉　删除光标所在字符。
- 〈Ctrl〉＋Y　删除光标所在行。
- 〈Ctrl〉＋T　删除光标右侧整个词。
- 〈Ctrl〉＋QY　删除从光标处到行尾的所有字符。
- 〈Ctrl〉＋N　插入一行。
- 〈Insert〉　编辑中“插入”或“改写”状态的切换(可由光标的变化判断)。

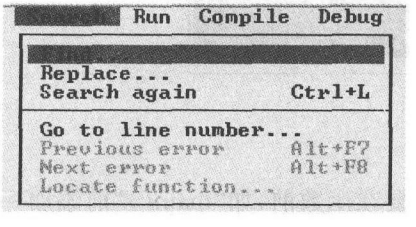


图 2.25　Search 子菜单

3. 查找操作

鼠标单击 Search 菜单，或按功能键〈F10〉后选择 Search 菜单，或按热键〈Alt〉＋S，均可打开 Search 菜单的下拉菜单，如图 2.25 所示。

Search 菜单中常用的菜单命令项有：

- Find　提供 Find Text 对话框，输入要查找的内容并选择查找方式(例如，向前或向后查找、全局查

找或是选定区域查找等)。

• Replace　提供 Replace Text 对话框,输入要查找的内容和要替换的新内容,并选择查找方式。

• Search again　继续查找下一个。

(六)编译

程序全部输入完毕或修改后,需要进行编译。方法是选择菜单项目"Compile"中的"Compile"命令,如果没有错误,将会提示编译成功"Success",生成目标程序。如果程序有语法错误,将弹出 Compiling 窗口,如图 2.26 所示。错误性质有两类,一类是 Error,表示严重错误,必须改正才可以生成目标代码;二是 Warning,表示警告可能的错误,有的不改仍然可以继续运行,有的则不行,因此用户须要注意警告的信息。键入回车键,则屏幕显示出错位置和提示信息。图 2.27 显示的错误为"Statement missing ;"即函数内缺少一个";",观察源程序,可发现在编辑窗口光条显式语句的上一行少了一个";"(注意:这个错误是入门者最常犯的错误,查找错误一般在当前行或上一行找),回车后在编辑窗口进行修改。源程序修改后继续编译,直到成功。有时候一个错误,例如缺少一个分号,会导致一串错误提示信息,所以不必一次改正所有错误再编译。

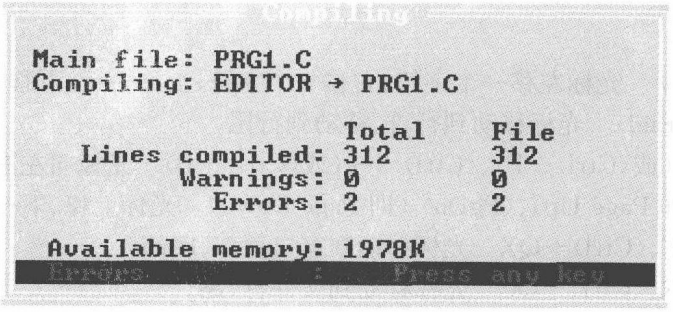


图 2.26　Compiling 对话框

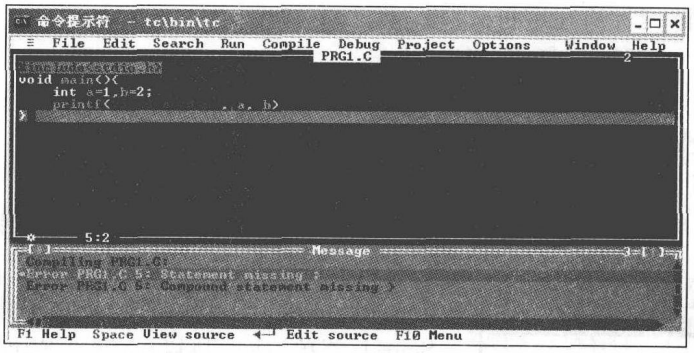


图 2.27　编译错误提示

下面是几个常见的警告和错误:

(1) 警告信息"Function should rethrn a value",只要在函数前,如"main()"前添加一个修饰词"void"即可,例如"void mian()"。

(2) 错误信息“Statement missing ;”，即函数内缺少一个“;”，此时需要在上一行语句末添加一个“;”。

(3) 错误信息“Compound statement missing }”，即函数内缺少一个“}”，此时需要在查找与“{”匹配的“}”，补上“}”。

(七) 连接

程序编译后，需进行连接。方法是选择菜单项目“Compile”中的“Link EXE file”命令，如果没有错误，将会提示“Success”，生成可执行程序。

程序连接还可以采用“Compile”菜单项目中的“Make EXE file”命令或“Build all”命令。前者先判断是否编译过，如果未编译，则先编译后连接。后者无论程序编译与否，都要先编译后连接。如果连接有错误，同样需要修改后重新连接。

常见的连接错误有：

(1) “Undefined symble ‘_Printf’ in module”，即符号_Printf不可识别。这类错误，首先需要查找该符号，如果是变量，则需要判定是否定义或者书写是否有错误，如果是库函数，则要判定书写是否正确(注意大小写不同)或者是否在程序中包含了(即#include)该函数所在的头文件。

(2) “Unable to open file COS. OBJ”，此时用户需要修改Turbo C的环境设置。方法是打开“Options”菜单选择“Directorise”命令，弹出的参数设置对话框，如图2.28所示。其中“Include Directorise”中给出的应该是Turbo C所在文件夹下的INCLUDE目录，“Library Directoris”中给出的应该是Turbo C所在文件夹下的LIB目录，“Output Directory”中给出的是编译、连接生成的目标文件和可执行文件的位置，“Source Directory” 中给出的应该是默认的源程序文件夹(即目录)。如果INCLUDE和LIB目录错误，则会出现上述提示错误信息。修改方法是用鼠标或光标选择要修改的选项，输入正确的目录后回车确认。如果希望保存修改的各项默认值，选择“Options”子菜单的“Save”命令保存修改内容。

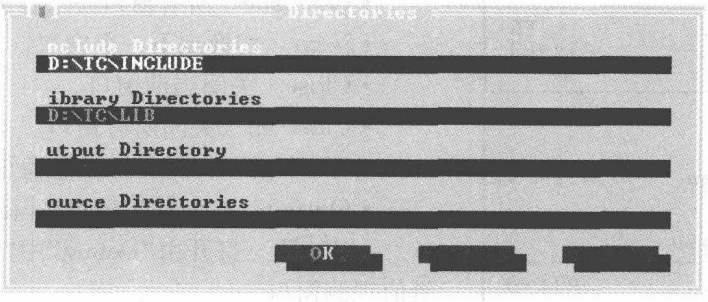


图 2.28　连接错误提示

(八) 运行

生成可执行程序后，就可以执行运行了。运行可执行程序有许多方法：一是在DOS环境下直接键入C程序的可执行文件名，二是在Windows的资源管理器内双击该可执行程序文件名，三是在C的集成环境下运行。这里只介绍第三种方法。

在C的集成环境操作界面下，选择“Run”菜单的“Run”命令，或者直接按〈Ctrl〉+〈F9〉

组合键，即可运行程序。如果源程序没有编译或者连接，将先编译、连接，之后再运行。程序运行结束后，系统又自然回到原状态。

虽然在C的集成环境下，一个未编译的程序选择运行“Run”命令，可以完成编译、连接、运行三个步骤，但是对于初学者，我们还是建议按照上述内容先选择编译，成功后再连接，最后选择运行。这样操作便于理解C程序的开发过程。

运行时，如果结果不符合要求，可以采用单步执行（使用功能键〈F7〉或〈F8〉）的方法观察程序流程是否准确。如果需要，还可以在整个程序执行过程中观察变量值的变化情况，具体方法是执行“Debug/Watches”菜单下的“Add watch”命令，键入需要观察的变量名，则随着程序运行，“Watch”窗口动态显示变量值的变化情况。如果只是需要在程序运行的某个位置观察某个变量的值时，可以选择“Debug”菜单下的“Evaluate/modify”命令，输入变量名（此外，该命令可以使用户能在程序运行中修改变量的值）。以上这些方法，有助于用户了解程序流程和变量的变化，从而找到算法或程序的错误或者漏洞。

（九）观看结果

在Turbo C++ 3.0的集成环境下观看运行结果的方法是：选择“Window”菜单下的“User screen”命令，或者直接按〈Alt〉+〈F5〉组合键。按回车可从用户屏幕返回C的集成运行环境的操作界面。

（十）窗口操作

鼠标单击Window菜单，或按功能键〈F10〉后选择Window菜单，或按热键〈Alt〉+W，均可打开Window菜单的下拉子菜单，如图2.29所示。

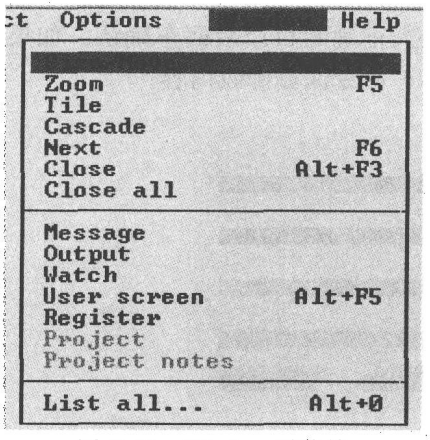


图2.29 Window子菜单

Window菜单中，常用菜单命令项有：

- Zoom 最大化当前窗口。
- Tile 平铺所有窗口。
- Casecade 所有窗口叠层排列。
- Next 切换到下一个窗口。
- Close 关闭当前窗口。
- Close all 关闭所有窗口。
- Message 打开提示信息窗口。
- Output 打开输出窗口（非图形方式）。
- Watch 打开由“Debug”中“Watch”设置的变量值观测窗口。
- User screen 全屏幕显示输出结果。

（十一）退出Turbo C

选择“File”菜单下的“Quit”命令，或者直接按〈Alt〉+X组合键，可退出C集成环境。

十一、VC++ 6.0集成开发环境简介

C++是在C语言基础上的扩充，属于面向对象程序设计的现代大型语言。C程序一般

可以在C++环境下进行调试和运行。但须注意以下几点：

(1) 每一C源程序文件在程序的开头用以下预处理命令包含“stdio.h”头文件：

#include <stdio.h>

(2) 每一C源程序文件名的扩展名必须是“.c”，如file1.c。而不应为“.cpp”。

(3) 每一工作区只可以运行一个C程序，调试运行完成一个程序后，应及时存储保存该程序并关闭当前工作区，然后开始另一C源程序文件的建立等工作。

(一) 输入和编译程序

用户在第一次进行VC++ 6.0集成开发环境之前，最好先在某个逻辑磁盘上建立好一个用户目录，作为进入VC++ 6.0后使用的当前工作目录，用它来保存自己的头文件、程序文件、目录文件、运行文件以及系统自动生成的其他相关文件。

若在你所使用的机器上没有安装VC++ 6.0语言版本的软件，则应设法安装该软件，然后才能使用它。

若在Windows操作系统桌面上有以“Microsoft Visual C++6.0”或“Msdev”为名字的图标，则双击后就启动VC++ 6.0，在屏幕上打开VC++ 6.0集成开发环境操作界面。或者依次单击菜单“开始/程序/Microsoft Visual Studio6.0菜单项打开该菜单。再接着单击或双击Microsoft Visual C++6.0”，也可进入了VC++ 6.0集成开发环境操作界面。

打开的VC++ 6.0操作界面，如图2.30所示，其中最顶行为窗口标题行，将显示当前编辑的程序文件的文件名。第2行为菜单行，其中每一个菜单项都对应一个下拉式菜单，菜单中的每一个菜单项都是一条操作命令，都具有一定的操作功能。第3行为按钮工具行，界面中间左半部为工作区窗口，右半部为程序编辑窗口。整个操作界面的最下部为状态输出窗口。

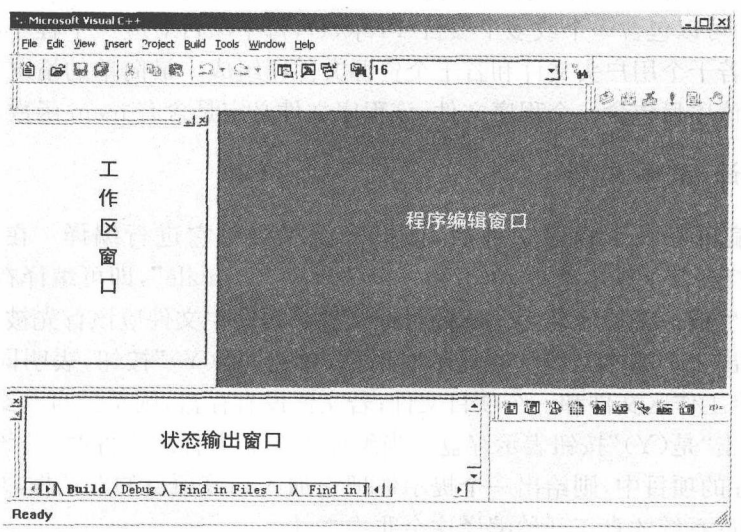


图2.30　VC++ 6.0操作界面窗口

为了新建或打开一个文件，请单击菜单栏中的“File”菜单打开此下拉菜单。在下拉菜单中，选择“New”菜单项则可以新建一个文件，选择“Open”菜单项则可以打开一个已有的文

件。当选择“New”菜单项后将打开 New 对话框，如图 2.31 所示。

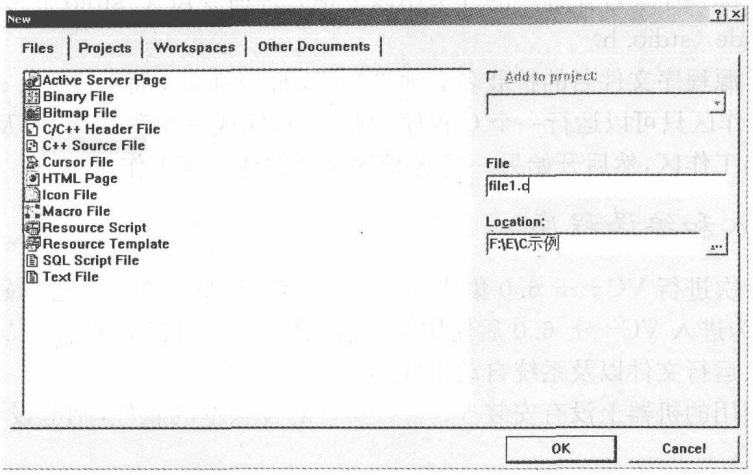


图 2.31　New 对话框

在“New”对话框的“Files”选项卡列表框中，选择“C++ Source File”则可以新建一个 C++或 C 源程序文件(经常被简称为 C++或 C 程序文件)。假定需要建立一个 C 程序文件，则选择“C++ Source File”后，接着在“New”对话框右边的“Location”文本编辑框中输入或通过该框右边按钮选择一个用户目录作为当前工作目录，在其上的“File”文本编辑框中输入一个新的文件名，其扩展名必须显式指定为. c(否则系统给以 C++程序文件的默认扩展名为. cpp)，最后单击右下角的“OK”按钮关闭该对话框，即可回到 VC++集成开发环境界面，接着可以在程序编辑窗口输入和编辑该程序文件的内容。

在 VC++ 6.0 集成开发环境下开发一个软件时，需要建立一个工作区(Workspace)。在一个工作区中可以包含一个或多个项目(Project)，每个项目对应一个程序，每个程序由若干个程序文件、若干个用户头文件和若干个资源文件所组成。最简单的情况是：一个工作区包含一个项目，该项目包含一个程序文件，该程序文件必定是含有 main 函数的主文件。

(二) 编译程序文件

当输入和编辑好一个程序文件后，接下来就需要对它进行编译。在菜单行中选择“Build”菜单项打开其下拉出菜单，单击第一个菜单项“Compile”，即可编译在编辑窗口中打开的程序文件，生成一个扩展名为. obj 的目标文件。程序主文件应该首先被编译，当编译它时，系统首先给出一个建立工作区的提示对话框，单击“是(Y)”按钮，表明同意在当前工作目录下建立一个工作区文件和一个项目文件；若文件没有存盘，则系统自动给出一个提示存盘的对话框，单击“是(Y)”按钮表示存盘。当编译一个非主程序文件时，若该文件没有被加入到主文件所在的项目中，则给出一个提示你把它加入到该项目的对话框，单击“是(Y)”按钮表示加入，然后系统才真正开始编译这个非主文件。

一个程序文件的编译过程是：首先处理所有预处理命令，如对于预处理包含命令，将把它替换为所包含的头文件或程序文件的内容(使用 include 预处理命令也可以包含扩展名为. cpp 或. c 的程序文件)；接着删除掉所有注释内容(实际上是用一个空格取代)；然后按行

从上向下进行语法分析；最后生成相应的目标文件。

若在编译过程中检查出语法错误，则将在状态输出窗口显示产生错误的程序行行号和错误原因，以便用户重新回到编辑窗口修改错误。

编译时检查出的错误包含两类：一类为严重错误(Error)，又称为致命错误，用户必须修改它，否则不能进一步向下处理；另一类为警告错误(Warning)，它不影响进入下一步处理过程，但最好也把它修改掉。

编译完一个程序文件后，将在状态输出窗口显示各类错误的个数，若不出现任何错误则显示出“0 error(s)，0 warning(s)”信息，如图 2.32 所示。

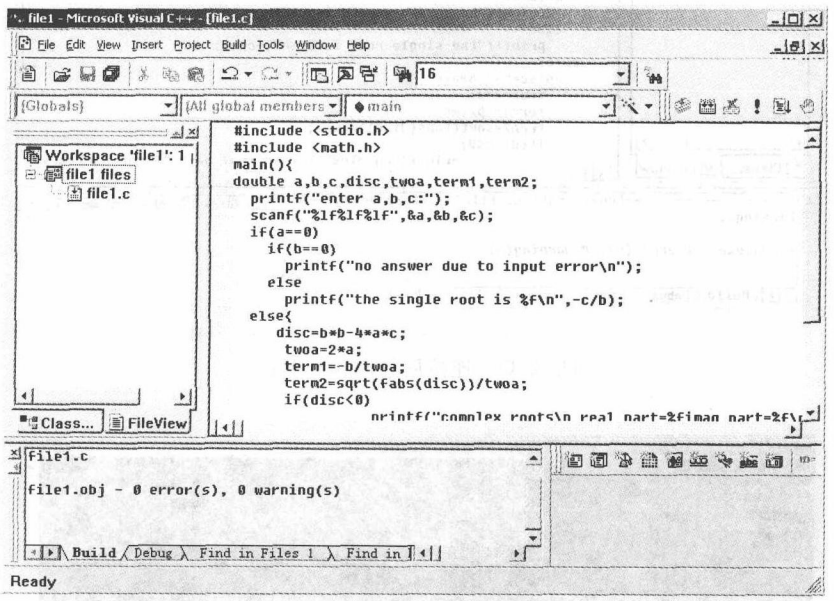


图 2.32　编译后界面状态窗口

注意：带扩展名为.h 的头文件不能被编译。

（三）连接程序文件

连接程序文件就是将一个程序中的主目录文件与其他目录文件和相关的库函数目录文件连接起来，形成一个可执行的文件。具体连接操作是：在菜单行上单击“Build”菜单项，打开其下拉菜单，单击第二个菜单项“Build”即可。若连接过程没有发现任何错误，状态输出窗口将显示“0 error(s)，0 warning(s)”信息，如图 2.33 所示，表示连接成功；若连接过程发现有错误，则将在状态输出窗口显示发生错误的文件、所在的行号和出错原因，用户应根据这些信息修改有关程序文件中的错误，然后再重新进行编译和连接。

（四）运行程序

运行程序就是运行经编译和连接而产生的可执行文件。具体操作是：在菜单栏上单击“Build”菜单项打开其下拉菜单，从中单击“Execute”菜单项即可。程序运行时将自动打开一个输出显示窗口，如图 2.34 所示。

程序运行结束后，输出显示窗口显示“Press any key to continue”提示信息，用户按下任一键后将关闭该窗口，重新回到 VC++ 6.0 操作界面。

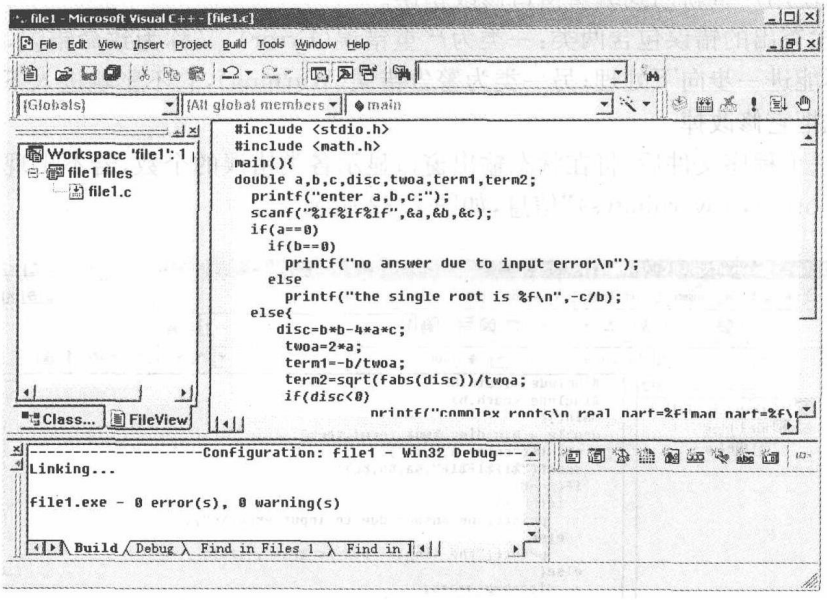


图 2.33　连接后界面状态窗口

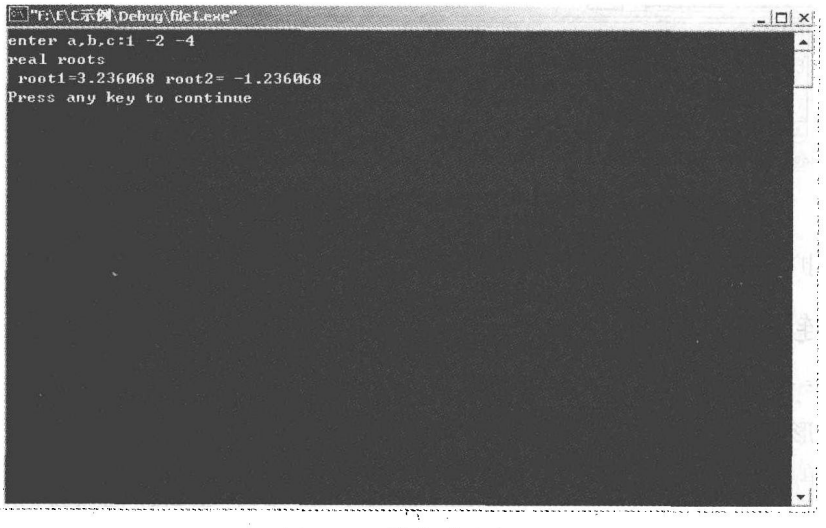


图 2.34　输出显示窗口

十二、VC++ 6.0 调试工具简介

（一）启动 Debug 工具

在“Build”菜单中，有一个“Start Debug”子菜单，如图 2.35 所示，包括以下 4 个菜单项：

• Go　从当前语句开始执行程序直到遇到断点或遇到程序结束。

• Step Into　单步执行程序；遇到函数调用时，进入函数内部并从子函数头开始单步执行。

• Run to Cursor　程序运行到光标所在的行时停止，事实上相当于设置一个断点。

• Attach to Process　调试过程中直接进入到正在运行的进程中。

单击这4个子菜单项中的某一项，将启动调试器，此时“Debug”菜单将代替“Build”菜单出现在菜单条中，窗口下方出现了两个小的窗口，用来显示程序执行中变量的值。如图2.36所示。

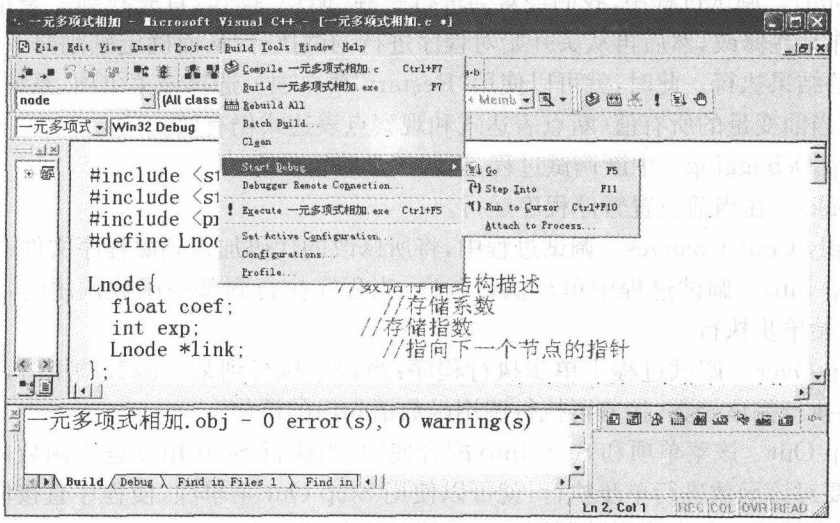


图 2.35　“Start Debug”级联菜单

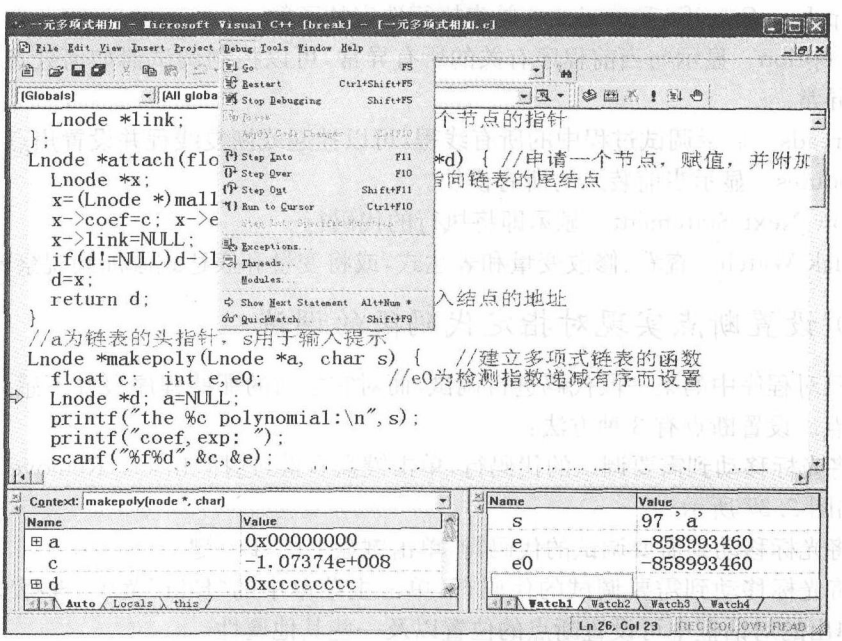


图 2.36　调试器窗口

(二) Debug 菜单

启动调试器后,“Debug”菜单将取代“Build”菜单出现在菜单栏中,“Debug”菜单下各命令的功能分别为:

• Go　用于在调试过程中,从当前语句启动或继续运行程序(等价于“Build MiniBar”工具栏中的“Go”按钮)。执行时,程序会一直正常运行,直到到达断点处停止。这样,在调试过程中,就可以越过某些已知正确或不感兴趣的程序段,从而提高调试速度。

• Restart　调试过程中,我们经常会进行一种“循环”操作:首先找到一条错误的语句,接着对其作某些修改,然后再从头开始对程序进行调试执行,从而确定刚刚修改的语句是否能按预期的结果执行。此时,就可以使用“Restart”菜单项。选择该菜单项,系统重新编译程序,并放弃当前变量的所有值(断点表达式和观察点表达式仍有效)。

• Stop Debugging　中断调试过程,返回正常编辑状态。

• Breaks　在当前位置暂停程序运行。

• Apply Code Changes　调试过程中,将所修改的代码加入到源程序文件中。

• Step Into　调试过程中单步执行程序,当程序执行到某一函数调用时,进入函数内部,从头开始单步执行。

• Step Over　调试过程中单步执行程序:当程序执行到某一函数调用时,不进入函数内部,直接执行完该函数,接着再执行调用函数语句后的语句。

• Step Out　该菜单项和 Step Into 配合使用,当执行 Step Into 进入函数内部时,若发现并不需要对该函数进行单步执行,就可以使用 Step Out 菜单项,使程序直接向下执行,直到从该函数内部返回,在该函数调用语句后面的语句处停止。

• Run to Cursor　程序运行到光标所在的行时停止,事实上相当于设置一个临时断点。

• Step Into Specific Function　单步执行选定的函数。

• Exception　显示与当前程序有关的所有异常,可以控制调试器如何处理系统异常和用户定义异常。

• Threads　显示调试过程中的所有线程,可以挂起或恢复线程并设置焦点。

• Modules　显示当前装入的所有模块。

• Show Next Statement　显示即将执行的代码行。

• Quick Watch　查看、修改变量和表达式,或将变量和表达式添加到观察窗口。

(三) 设置断点实现对指定代码段的调试

如果只对程序中的某一段代码进行调试,而对它之前的那些程序段并不感兴趣,此时可以设置断点。设置断点有 3 种方法:

(1) 将光标移动到需要调试的代码行,单击键盘右键,选择“Insert/Remove Breakpoint”菜单项,如图 2.37 所示。

(2) 将光标移动到需要调试的代码行,单击键盘上的〈F9〉键。

(3) 将光标移动到需要调试的代码行,单击主菜单中的“Edit”菜单,选择“Breakpoint”菜单,在弹出的对话框中可设置断点的位置以及一些其他属性。

设定断点后在需要调试的代码行即出现了一个棕色的小圆,表明此代码行处有一个断

点(参见图 2.37)。

调试程序时,如果有一个断点暂时不需要了,那么可在此断点处单击鼠标右键,在弹出的快捷菜单中选择"Disable Breakpoint"项。此时棕色小圆变成一个空心的圆,表示断点已取消。但并未删除此断点。需要时在断点处单击鼠标右键,选择"Eneble Breakpoint"菜单项,可恢复断点。

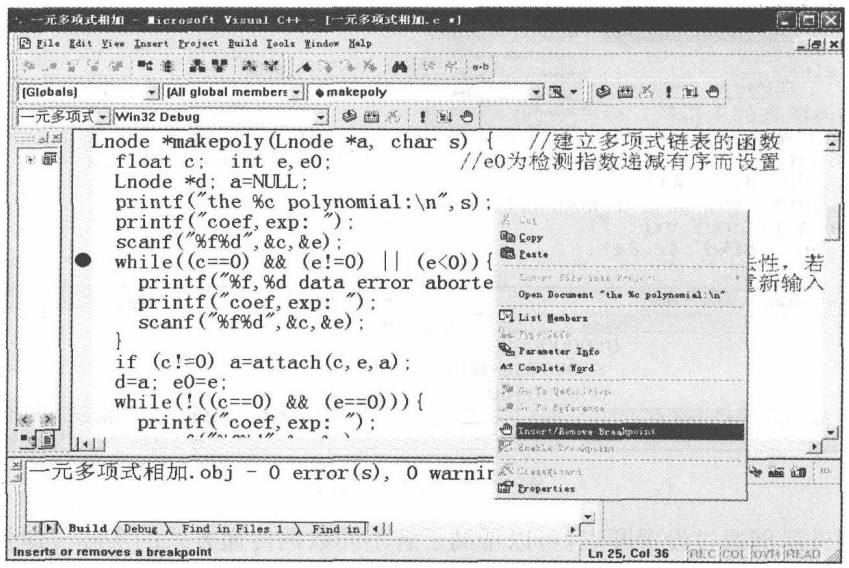


图 2.37　断点的设置与删除

设置断点后,单击"Start Debug"子菜单下的"Go"菜单项,或单击"Build Mini Bar"工具栏中的"Go"按钮,或直接按〈F5〉键,即可运行程序到第一个断点处。重复此操作,可以运行程序到下一个断点。

当调试程序完成或不需要已有的断点时,在断点处单击鼠标右键,在弹出的快捷菜单中选择"Remove Breakpoint"项,删除断点。

(四) C调试窗口

启动了调试器后,调试窗口如图 2.36 所示。单击主菜单"View"下的"Debug Windows"子菜单,如图 2.38 所示,可以打开一些辅助调试的窗口。"Debug Windows"菜单下各命令的功能分别为:

- Watch　显示查看变量窗口。
- Call Stack　显示查看调用堆栈的窗口。
- Memory　显示查看内存窗口。
- Variables　显示查看指定变量窗口。
- Registers　显示查看寄存器窗口。

利用这些窗口可以大大方便调试过程。

(五) 程序调试示例

在程序调试过程中,利用调试器提供的程序调试手段来了解内部数据结构情况,通过分

析、比较与验证等方法，可有效提高学习程序设计语言的效果和编程能力。

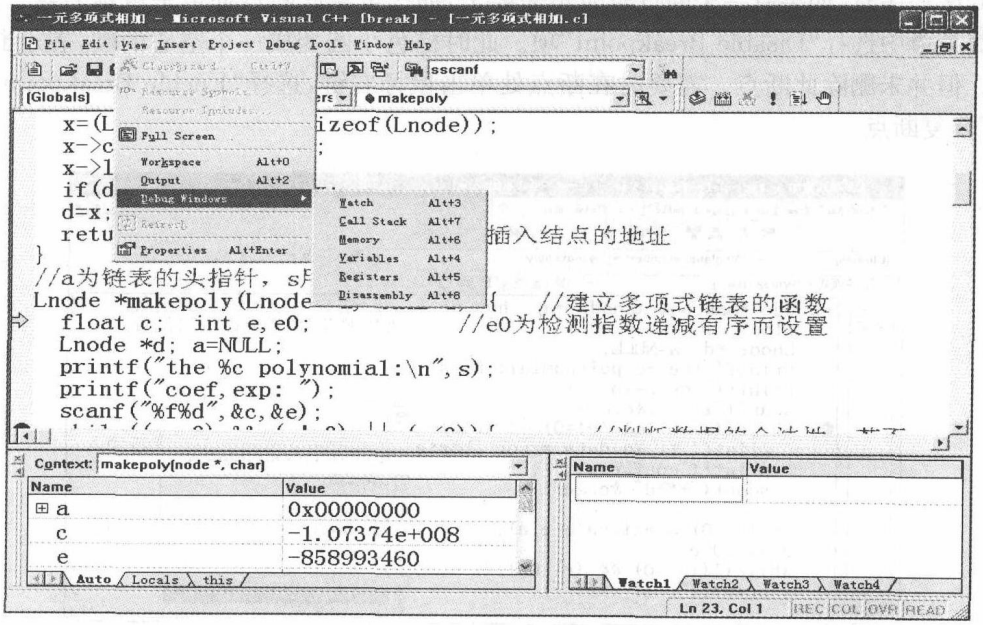


图 2.38　打开辅助窗口

以下调试示例通过设置断点，可以帮助了解不同数据传递方式的实际传递效果。示例源程序如下所示：

```
#include<stdio.h>
void swap1(int a, int b){
    int temp;
    temp=a; a=b; b=temp;
}
void swap2(int *a, int *b){
    int *temp;
    temp=a; a=b; b=temp;
}
void swap3(int *a, int *b){
    int temp;
    temp=*a; *a=*b; *b=temp;
}
void main(){
    int a=1,b=3;
    swap1(a,b); printf("a=%d b=%d\n", a,b);
    swap2(&a,&b); printf("a=%d b=%d\n", a,b);
    swap3(&a,&b); printf("a=%d b=%d\n", a,b);
}
```

(1) 调用 swap1()函数，验证数据传递采用传值方式时的形参状态。如图 2.39、图 2.40 所示。

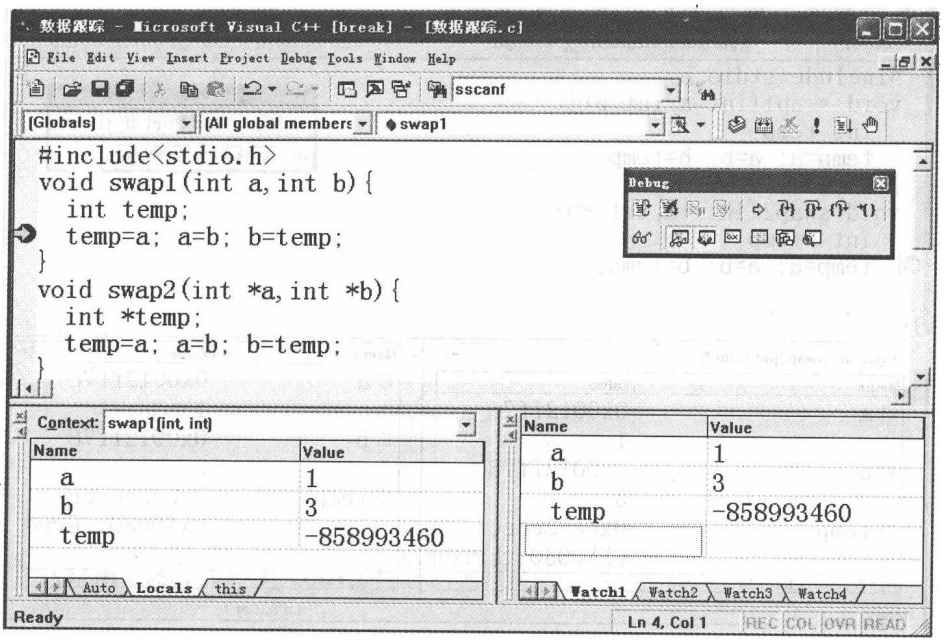


图 2.39　进入 swap1()函数体时数据状态

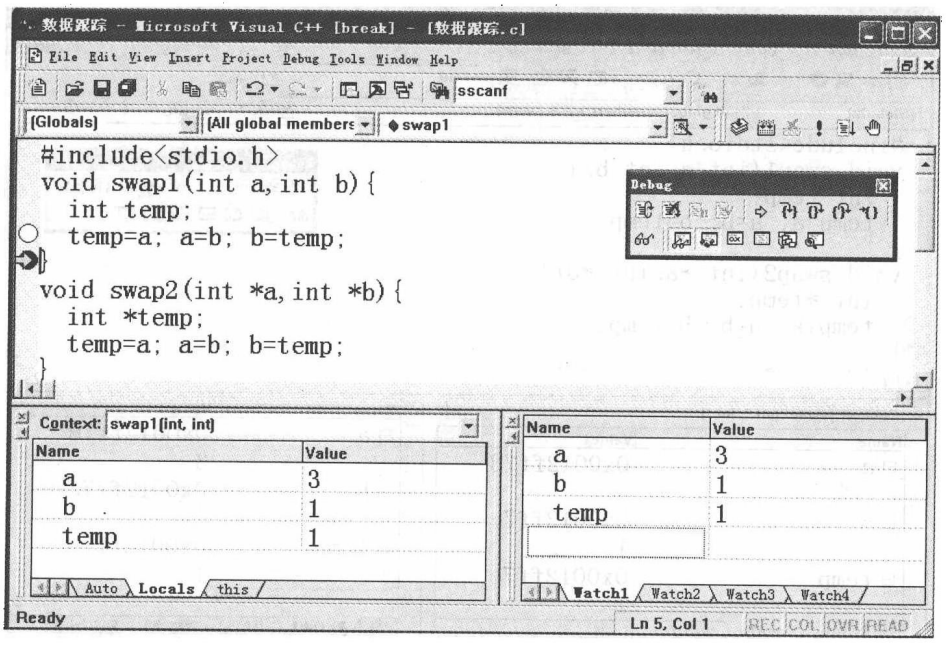


图 2.40　执行 swap1()函数体后，即将返回主调函数时数据状态

(2) 调用 swap2()函数，验证用传地址方式进行数据传递时的形参状态，在函数体中交换指针变量的值，即只改变指针变量的指向。如图 2.41、图 2.42 所示。

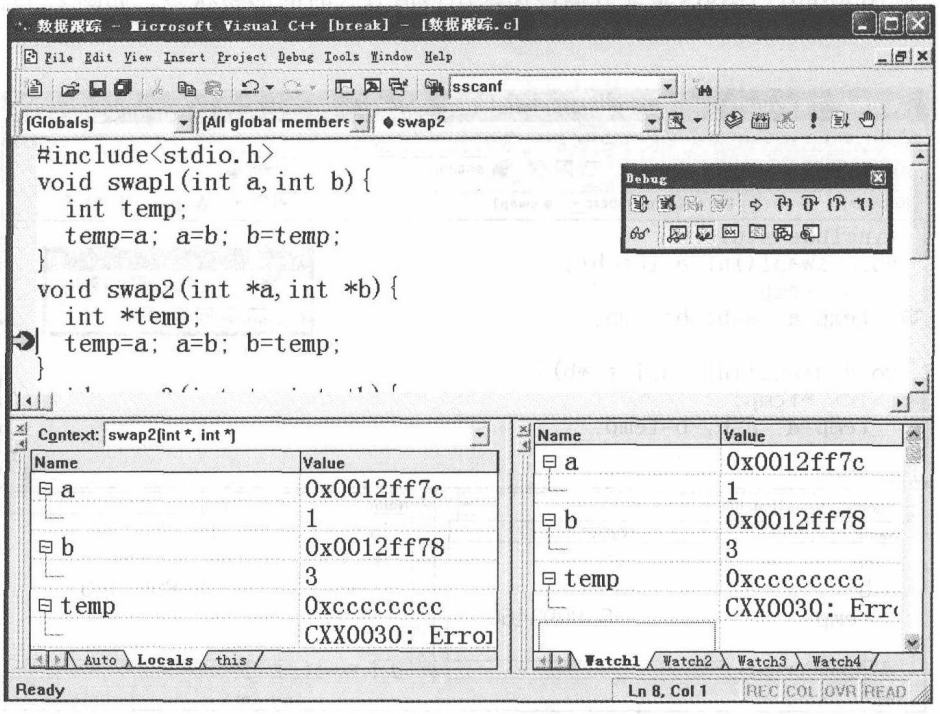


图 2.41　进入 swap2()函数体时数据状态

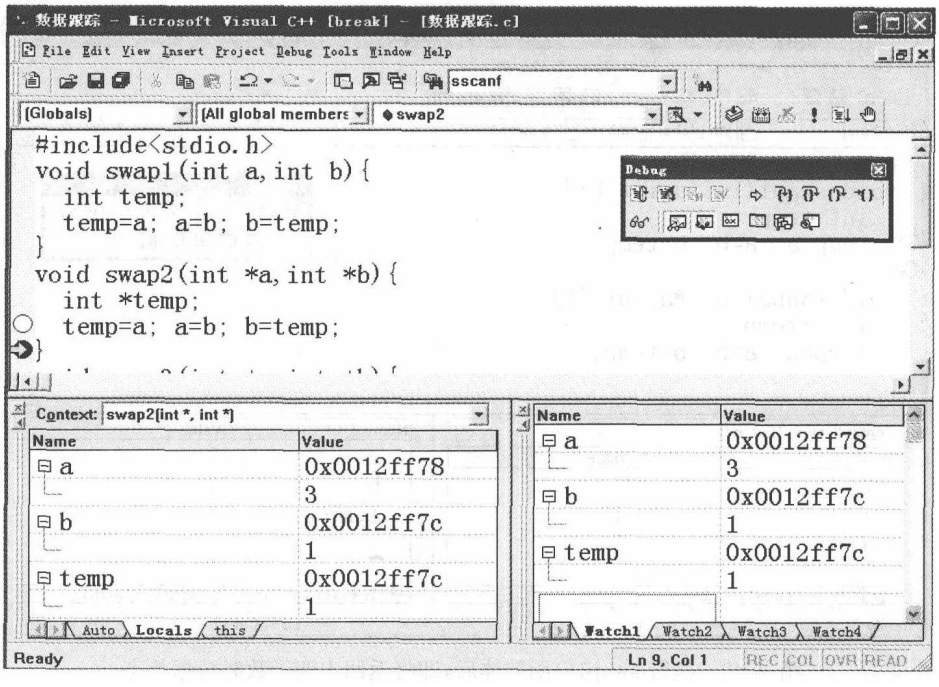


图 2.42　执行 swap2()函数体后，即将返回主调函数时数据状态

(3) 调用 swap3()函数，验证用传值方式进行数据传递时的形参状态。在函数体中交换

指针变量所指存储单元的值，真正实现数据的双向传递。如图 2.43、图 2.44 所示。

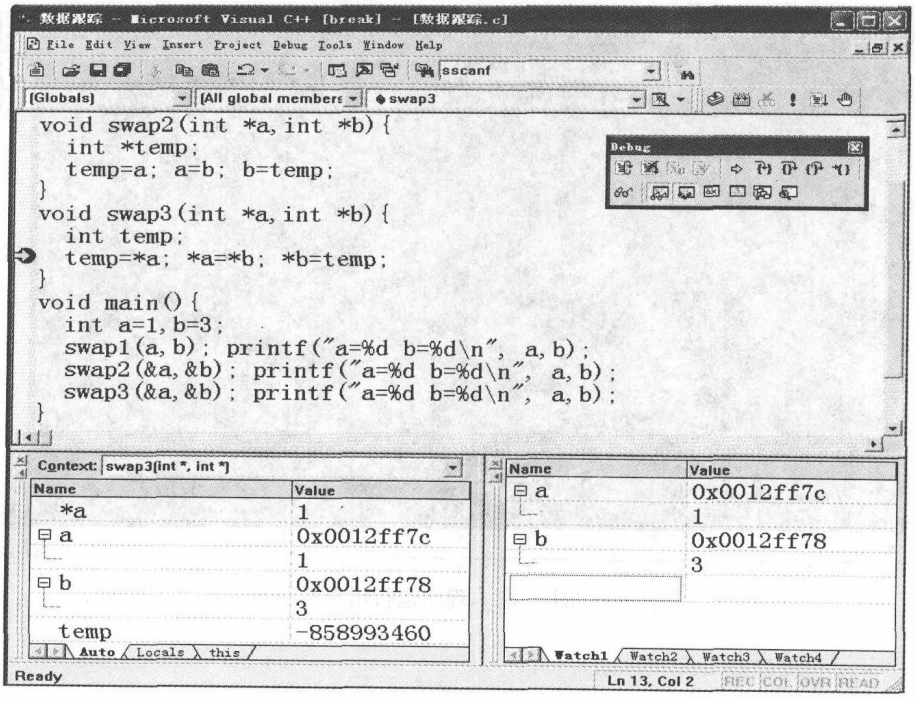


图 2.43　进入 swap3()函数体时数据状态

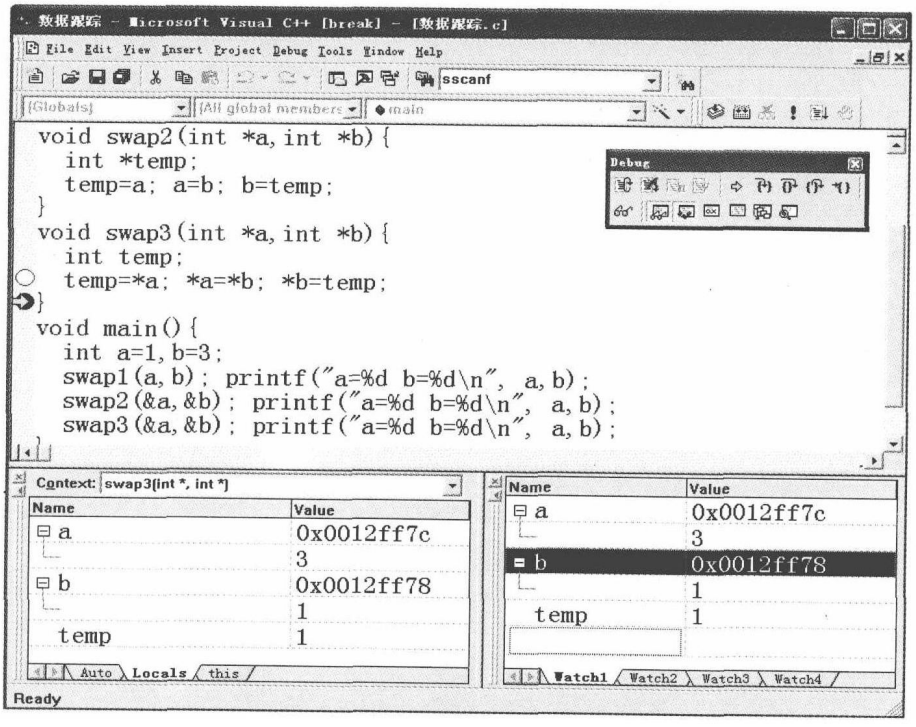


图 2.44　执行 swap3()函数体后，即将返回主调函数时数据状态

(4)运行程序，分别调用 swap1()，swap2()，swap3()后的运行结果如图 2.45 所示。

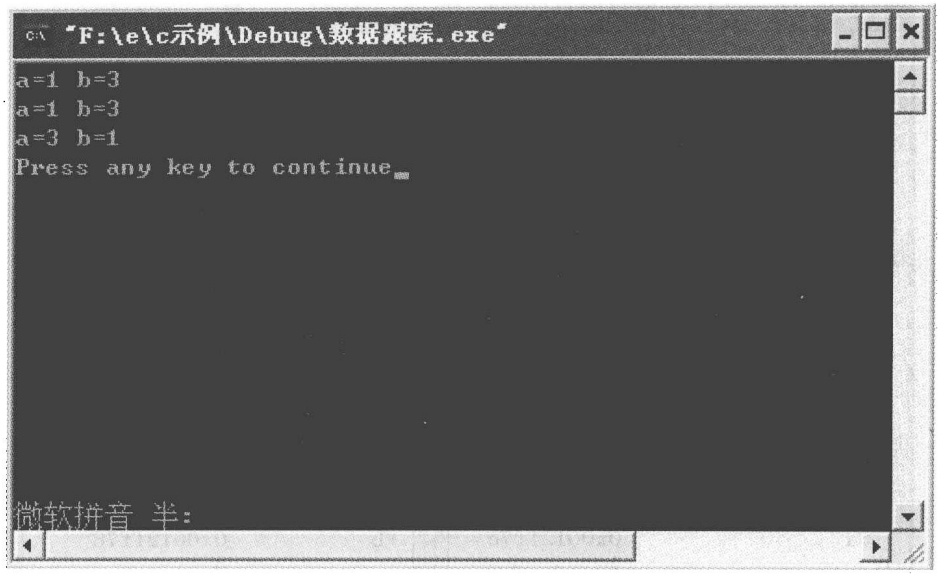


图 2.45　程序运行结果

第三部分 上机实验

实验概说

一、实验目的

作为教学的一个重要环节，实验的目的在于更深入地理解和掌握课程教学中的有关基本概念，应用基本技术解决实际问题，从而进一步提高分析问题和解决问题的能力。C程序设计课程实践性很强，实验环节尤显重要，即要求独立编写程序，学会独立上机调试程序。

学会独立上机调试程序。也就是要善于发现程序中的错误，并且能很快地排除这些错误，使程序能正确运行。计算机技术是实践性很强的技术，要求从事这一领域的人不仅能了解和熟悉有关理论和方法，还要求自己动手实践。对程序设计来说，要求会编写程序并上机调试通过。因此调试程序本身是程序设计课程的一个重要的内容和基本要求，应给予充分的重视。调试程序的经验固然可以借鉴他人的现成经验，但更重要的是通过自己的直接实践来积累，而且有些经验是只能“意会”而难以“言传”。因此，在实验时不但要通过程序完成每一次的实验任务，而且应当在已通过的程序基础上作进一步的修改、提高和完善。甚至“自设障碍”，即把正确的程序改为有错的(如用 scanf 函数为输入变量输入数据时，漏写“&”符号，double 变量使用格式符“%f”；使数组下标出界；使整数溢出，等等)，观察和分析所出现的情况。这样的学习才会有真正的收获。

实验目的可归纳如下：

(1) 验证自己已建立起来的概念或所编写的程序是否正确。

(2) 加深对课堂所学内容的理解和语法规则的记忆。

(3) 理解和掌握运用计算机高级语言进行编程的思想方法。

(4) 掌握常用算法的设计与应用实现。

(5) 熟悉 UNIX 等程序开发环境，掌握 C 程序常用的调试手段。

(6) 学会上机调试程序的方法，不断积累调试经验，提高排错能力。

(7) 使自己具有独立的应用编程和熟练的程序调试能力。

二、实验要求

(1) 做好每次上机前的准备以提高上机效率。

① 预先认真阅读相关实验内容，做到心中有明确的目的和任务，要有备而来。

② 按照实验内容规定的题目，事先在实验预习报告上编写好源程序及运行程序所需的典型数据，并经人工静态检查认为无误；手编程序应书写整齐，应在每个题目之间留出一定的空间，以备记录上机调试情况和运行结果等；对程序中自己有疑问的地方，应做出记号，以便上机时给以注意。

③ 将想要上机验证的问题草拟提纲，制定一个简捷的程序调试计划。

(2) 上机时输入和调试自己所编写的程序。

① 在使用键盘时，应采用正确的指法击键，为今后正确使用键盘打下基础。

② 应该一人一组，尽量独立思考。上机过程中出现的问题，除了是系统的问题外，一般应自己独立处理，尤其对“出错信息”，应善于自己分析判断，并充分利用开发工具提供的错误信息和调试手段进行解决，及时修改与完善源程序，随时记录有价值的内容。

③ 解决问题是学习调试程序的良好机会。初学者切不可不编程序或抄别人的程序去上机，应从一开始就养成严谨的科学作风。

(3) 程序调试通过后，应运行程序并根据事先准备的典型数据测试程序；运行时要注意在输入不同数据时所得到的不同结果(如解一元二次方程上时，不同的 a，b，c 组合应得到实根、复根的不同结果)。应运行几次，分别检查在不同情况下程序是否正确。

(4) 为便于教师检查和以后复习，请不要删除已通过的程序以及有问题待答疑的程序，应保存在自己的工作目录内。源程序名应能反映哪一次实验的哪一个题目，如 exp1_1.c 表示实验一的第 1 题，exp3_5.c 表示实验三的第 5 题，exp5_1_1.c 和 exp5_1_2.c 分别表示实验五中第 1 题的方法 1 和方法 2，等等。若有未完成调试的源程序，可以作存盘保存处理，待下一次实验继续完成。

(5) 上机结束后，应整理出实验报告，实验报告应包括以下内容：

① 实验目的。

② 题目。

③ 程序清单(附算法描述或流程图)。

④ 运行结果(必须是程序清单所对应的输入数据和打印输出的结果)。

⑤ 调试分析和体会。这是实验报告中最重要的一项，也是最容易忽视的一项。实验过程中大量的工作是程序调试，在调试过程中会遇到各种各样的问题，每解决一个问题就能积累一点经验，提高编程的能力。因此，对实验的总结，最重要的是程序调试经验的总结。调试分析也包括对结果的分析。体会主要是指通过本次实验是否达到了实验目的，有哪些基本概念得到了澄清等。

三、实验项目内容及学时分配

我们在第三部分给出了 11 个实验内容作为参考，每一个实验对应教材中相应章节的内容，具体的实验内容分为必做题和选做题两部分，组织上机实验时可以根据个人的能力和条件，先保证完成必做题，然后再做选做题。选做题作为实验内容的一部分，依据完成情况，以一定比例计入上机实验成绩。

建议各实验学时分配如下：

序 号	实验项目内容	学 时
1	C程序的运行环境与运行过程	4
2	数据类型、运算符和表达式	4
3	选择分支结构编程	4
4	循环控制结构编程	4
5	数组	4
6	函数与编译预处理	4
7	指针	4
8	结构体和共用体	4
9,10	位运算与文件	4
11	综合练习	4

实验一　C 程序的运行环境与简单的 C 程序设计

学习重点

- C 程序的基本结构
- 函数形式与 main()函数、库函数与用户自定义函数
- 库函数 printf()的使用格式和输出结果
- 库函数 scanf()的使用格式和输入方法
- 库函数 getchar()和字符的读取
- C 程序的编辑、编译和连接

一、要点综述

1. C 程序基本结构

C 程序由一个主函数 main( )或一个主函数及其他函数(库函数和用户自定义函数)组成,根据需要一般还需外加预处理命令及外部变量定义等。C 程序基本结构如下:

```
预处理命令、外部变量定义等
函数类型   main( ){
    数据说明部分;
    执行语句部分;
}
函数类型   fun_1( 形参说明表列 ){
    数据说明部分;
    执行语句部分;
}
......
函数类型   fun_n( 形参说明表列 ){
    数据说明部分;
    执行语句部分;
}
```

(1) 主函数有且只有一个,可以放在程序的任何位置。但C程序总是从主函数 main( )开始执行,并且在主函数中结束,其他函数只有通过函数调用语句来执行。主函数可以调用任何非主函数;非主函数之间可以相互调用,但是不能调用主函数。

(2) C语言规定函数、变量和符号常数必须先定义后使用,否则必须事先说明。

(3) 分号是语句的结束标志,是语句必要的组成部分。可以一行写几个语句。

(4) C程序的注释信息可以出现在程序的任何地方,有以下两种形式:

① 以"/ * "符号开始,以" * /"符号结束,其中的内容为注释信息。

② 以"//"符号开始注释该行其后的内容为注释信息。

注释信息不会出现在目标程序中。

2. C语言的函数分类

C语言的函数可分为主函数、库函数和用户自定义函数三类。

函数的一般形式为:

```
函数类型  函数名(形参说明表列 ){
    数据说明部分;
    执行语句部分;
}
```

↓ 规定有序

(1) 函数类型用于确定 return 语句返回值的类型。当函数类型缺省时为整型;若函数值为非整型时,则必须使用类型标识符显式指定函数类型;当函数无返回值时,函数类型可用 void 说明。

(2) 函数的定义在程序中是平行的,不允许函数嵌套定义。

3. 库函数 printf( )的使用

格式输出函数 printf()的作用是按规定的格式把输出项输出到显示屏上。

一般形式:

```
printf("输出格式串",输出参数项表列);
```

说明:

(1) 在双引号定界的输出格式串中,可以使用三种不同的字符成分:格式符,转义字符,一般字符。

① 格式符规定了相应输出项的输出形式。常用格式及其意义如下:

%d 按十进制整数形式输出

%x 按十六进制整数形式输出

%f 按浮点数(实数)形式输出

%c 按单一字符形式输出

%s 按字符串形式输出

……

格式符的基本形式:

```
%[<数据输出宽度说明>]<格式字符>
```

即定义输出形式时还可以规定其输出项的宽度。例如%6d，%10.2f等。“%m.nf”表示保留n位小数(四舍五入)并且共占用m列(含小数点)。数据宽度说明中，如果实际数据位小于宽度(绝对值)，则宽度大于零时左补空格，宽度小于零时右补空格。如果实际数据位大于宽度，那么按实际位数输出。若缺省宽度说明，则按实际宽度输出(实数位数按格式符默认位数输出)。

② 转义字符用于输出控制代码和特殊字符，其形式为“\”打头后跟一个字母或数字。如“\n”为回车换行控制代码。又如，想输出字符“%”、“\”、“'”、“"”，应在格式控制部分采用“\%”、“\\”、“\'”、“\"”转义字符形式。

③ 输出格式串中，格式符和转义字符以外的其他字符是一般字符，原样输出，一般用于增加运行结果的可读性。

输出格式串中三种字符配合使用，可以达到不同的显示效果。

(2) 输出参数可以是常量、变量和表达式等，可以有多个输出项，之间用逗号分隔。Printf函数执行时，按照自右向左的顺序，依次计算“输出参数项表列”中各项的值，然后将这些值按照“输出格式串”中规定的格式输出到显示器上。要求输出格式符与输出项的个数必须相同，按各自的先后顺序一一对应，且符合类型匹配的要求。当格式说明的个数小于输出表列的个数时，多余的输出表列的项不输出。Printf函数的返回值是输出数据的个数。

(3) printf函数可以没有输出项，此时利用它可输出提示信息。如：

```
printf("test!");
```

4. 库函数scanf()的使用

Scanf函数功能是接受用户从键盘输入的数据，并按照格式控制符的要求进行类型转换，然后送到地址表列中对应的变量存储单元去。返回值是输入数据的个数。

一般形式：

```
scanf("输入格式串",输入项地址表列);
```

说明：

(1) 输入格式串的组成同printf()，一般只使用格式符。

(2) C语言规定输入项必须由地址构成，项与项之间逗号分隔。

(3) 输入格式串中格式符的个数决定了输入项地址表列中可输入数据的个数，且数据类型必须从左到右一一匹配对应。

(4) “&”为地址运算符或称取地址符。变量的地址表示法是在变量前加“&”符号，如“&a”表示取a变量的地址，为地址表达式，运算结果为a变量在内存中的对应存储单元的地址。

(5) 如果输入格式串中加入了格式符以外的其他字符，则在通过键盘等输入设备输入数据时，这些其他字符也要同样输入；如果输入格式符中无其他字符，则输入时可以用空格、回车或制表符〈Tab〉作为间隔标志；特别注意，使用“%c”时，输入数据不需要间隔标志。

(6) 用户可以指定输入数据的宽度(但不能指定小数点的位数)，系统自动按照此宽度截取读入的数据。

(7) 输入数据遇到空格、回车或制表符〈Tab〉以及其他各种非法输入时，认为该项数据输入结束。

(8) 当遇到格式符“%*”时，表示跳过对应的输入数据项不予读入。

5. C 程序的开发过程

C 程序的开发过程需经历编辑、编译、连接和运行。

(1) 通过编辑生成的文件以文本文件形式存储，称为源程序，扩展名为“.c”。

(2) 编译是将 C 的源程序翻译成计算机可以识别的二进制形式的目标程序，扩展名为“.obj”。

(3) 连接是将目标程序和其他目标程序模块以及系统提供的 C 库函数等进行连接，生成可执行文件，扩展名为“.exe”。

(4) UNIX 系统的默认可执行文件名为 a.out。

C 程序的上机过程如图 3.1 所示。

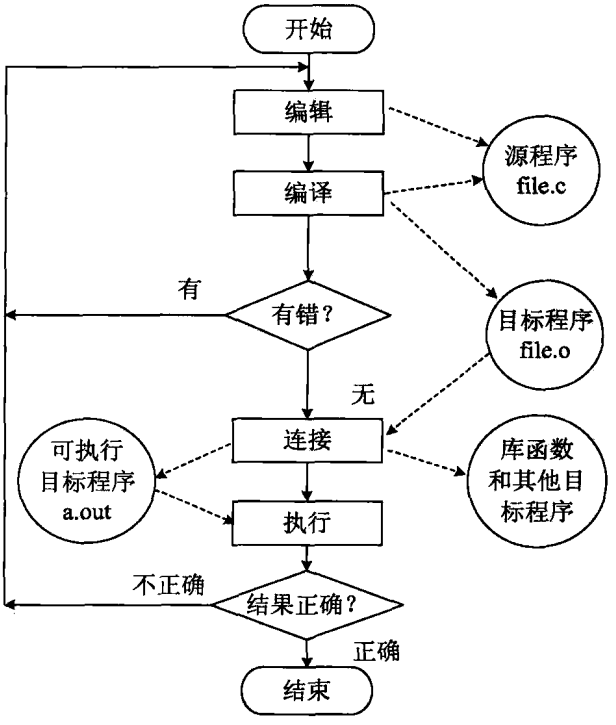


图 3.1 C 程序的上机过程

二、实验目的要求和实验内容

(一) 目的要求

(1) 了解所使用的计算机系统的基本操作方法，学会独立使用该系统。

(2) 掌握在该系统上如何编辑、编译、连接和运行 C 程序。

(3) 通过编写和运行简单的 C 程序，了解程序的基本结构和函数的定义形式，理解系统库函数的说明性预处理文件(stdio.h，math.h 等)的概念，掌握函数的定义和系统库函数的使用。初步了解 C 源程序的特点。

（二）范例

1. 编写一个程序，输入 a，b，c 三个值，输出其中最大者。

源程序文件 prg1_1. c 清单：

```
#include<stdio. h>
int main(){
  int a,b,c,max;
  printf("Enter a,b,c: \n");
  scanf("%d%d%d",&a,&b,&c);
  max=a;
  if(max<b)
    max=b;
  if(max<c)
    max=c;
  printf("max=%d\n",max);
}
```

操作步骤如下：

(1) 利用正文编辑程序将 prg1_1. c 文件建立在用户自己的当前目录上。

(2) 发出编译命令：

```
$cc prg1_1. c
```

(3) 运行 a. out 可执行文件：

```
$a. out ↙
```

(4) 运行结果：

```
Enter a,b,c:
2  5  4 ↙
max =5
```

注意：不同的运行环境和系统存在细微的差异，如 UNIX 系统的 main()函数不允许定义为 void 类型等，上机时需要给以注意和区别。

2. 下例是用于求一元二次方程 $ax^2+bx+c=0$ 的两个实根的一个 C 程序，其中

$$a=1.0,\ b=-(10^{12}+1),\ c=10^{12}$$

输入并运行该程序，检查结果是否正确，分析原因。其中求根公式为：

$$x_1=(-b+\sqrt{b^2-4ac})/(2a)$$

$$x_2=(-b-\sqrt{b^2-4ac})/(2a)$$

源程序文件 prg1_2. c 清单：

```
#include <math. h>
#include <stdio. h>
main(){
```

```
    double a,b,c,p,d,x1,x2;
    a=1.0; b=-(1.0e+12+1.0); c=1.0e+12;
    p=sqrt(b*b-4.0*a*c);
    d=2.0*a;
    x1=(-b+p)/d;
    x2=(-b-p)/d;
    printf("x1=%e\nx2=%e\n",x1,x2);
}
```

操作步骤如下：

(1) 利用正文编辑程序将 prg1_2.c 文件建立在用户自己的当前目录上。

(2) 发出编译命令：

```
$cc prg1_2.c -lm↙
```

注意：在 UNIX 系统开发环境，凡源程序中含 #include <math.h>预处理命令时，其编译命令中必须使用"-lm"命令参数，以便实现正确连接库文件。

(3) 运行 a.out 可执行文件：

```
$a.out↙
```

(4) 运行结果：

```
x1=1.000000e+012
x2=1.000000e+000
```

3. 一个简单的 C 源程序，它由两个源程序文件组成，每个源程序只含一个函数。

源程序文件 prg1_3.c 清单：

```
#include<stdio.h>
main(){
  printf("The UNIX System.\n");
  prt( );
}
```

源程序文件 prg1_4.c 清单：

```
#include<stdio.h>
prt( ){
  printf("The C Programming Language.\n");
}
```

(1) 利用正文编辑程序将 prg1_3.c，prg1_4.c 文件建立在用户自己的当前目录上。

(2) 发出编译命令：

```
$cc prg1_3.c prg1_4.c-o prg34↙
```

(3) 运行 a.out 执行文件：

```
$prg34↙
```

(4) 运行结果：

```
The UNIX System.
```

The C Programming Language.

本题也可以试用其他编译方式。另外，可有意在源程序中安排一点错误，观察 cc 编译时的反映。

（三）实验内容

1. 学习从开机开始的操作，包括按账号登录、列文件目录、显示文件内容、删除文件、复制文件等。

2. 调用文本编辑程序，输入范例程序，并进行编译和运行。应了解所用的系统是用什么命令进行编译和运行的。

3. 了解程序的输入和输出。

4. 编写如下上机习题 1，2，3，4，并上机运行。

（四）上机习题

1. 编写一个 C 程序，输出以下信息：

```
* * * * * * * * * * * * * * * * * * * * * * *
                    Very Good!
* * * * * * * * * * * * * * * * * * * * * * *
```

2. 编写一个计算圆柱体体积的程序。半径 R、高度 H 的数值通过键盘输入。

3. 编程序实现华氏温度到摄氏温度的转换，其转换公式是 $c=\frac{5}{9}\times(f-32)$，式中 f 表示华氏温度，c 表示摄氏温度。

4. 编程序实现功能：从键盘输入三个整数作为边长，判断可否构成三角形？若成立则计算三角形的面积，否则终止计算。

5. 阅读程序分析运行结果，并上机验证。

```
#include <stdio.h>
main(){
  char c1,c2,c3,c4;
  scanf("%c%c",&c1,&c2);
  getchar();
  c3=getchar(); c4=getchar();
  printf("%c %c %c %c\n",c1,c2,c3,c4);
  printf("%c,%d\n",c1,c1);
  printf("%d,%d\n",sizeof(c1),sizeof("1234"));
}
```

若从键盘分行输入以下数据，则输出结果是什么？

12↙

3456↙

提示说明：

(1) 执行 getchar()函数时，首先直接从输入缓冲区中读取字符，直到输入缓冲区为空才

等待从键盘继续输入。一行上输入的所有字符一并送入缓冲区，输入完毕，getchar()函数从缓冲区读取一个字符，只要缓冲区还有剩余的字符，后面的 getchar()函数就会直接从中读取。空格和回车符都属于字符，很有必要充分注意和掌握字符输入的特殊性及其输入方法。

(2) sizeof 为长度运算符，运算对象(即括号中的内容)可以是常量、变量、表达式和数组名等，运算结果为运算对象存储字节长度，对表达式而言是表达式值的字节长度。

(3) 注意思考源程序中第 5 行的功能。

6. 修改范例 prg1_2.c 程序文件，使程序中的变量均为 float 类型，验证结果是否正确，分析原因(提示：从数据精度上找原因)。

(五) 常见错误分析

【错误示例 1.1】 输入三角形三条边的边长(设三条边分别为 a,b,c)，求三角形的面积。

源程序(cw1.c)如下：

```
#include<stdio.h>
main(){
    double a,b,c,s,area;
    scanf("a=%lf,b=%lf,c=%lf",&a,&b,&c);
    s=(a+b+c)/2.0;
    area=sqrt(s*(s-a)*(s-b)*(s-c));
    printf("%6.2f,%6.2f,%6.2f\n",a,b,c);
    printf("s=%-8.2f,area=%-8.2f\n",s,area);
}
```

以上程序在编译时将产生一个错误。因为在程序中要求调用求平方根 sqrt()函数，但没有在程序中用预处理命令“#include”将“maht.h”文件包括到本用户源文件中。违反了函数、变量和符号常数必须先定义后使用，否则应事先说明的规则。

UNIX 系统错误提示为：

```
undifined reference to'sqrt'
```

VC++ 6.0 集成环境中错误提示为：

```
'sqrt' undefined;
```

修改后的程序如下：

```
#include<stdio.h>
#include<math.h>
main(){
    double a,b,c,s,area;
    scanf("a=%lf,b=%lf,c=%lf",&a,&b,&c);
    s=(a+b+c)/2.0;
    area=sqrt(s*(s-a)*(s-b)*(s-c));
    printf("%6.2f,%6.2f,%6.2f\n",a,b,c);
```

```
        printf("s=%-8.2f,area=%-8.2f\n",s,area);
    }
```

在 UNIX 系统下按如下命令行形式编译：

```
$cc cw1.c -lm↙
```

编译成功，生成 a.out 可执行文件。运行 a.out 文件即可得到结果。但此时初学者会遇到一个必须按 scanf()的格式字符串要求输入数据的问题，否则将得到不可预想的结果。提示如下：

```
$a.out↙
输入:3 4 5↙
输出:  0.00,  0.00,  0.00
     S=0.00     ,area=0.00
```

输出结果显然不正确，其原因就是输入数据与 scanf()输入格式串所要求的格式不符。正确的输入应如下所示：

```
$a.out↙
输入:a=3,b=4,c=5↙
输出:  3.00,  4.00,  5.00
     S=6.00     ,area=6.00
```

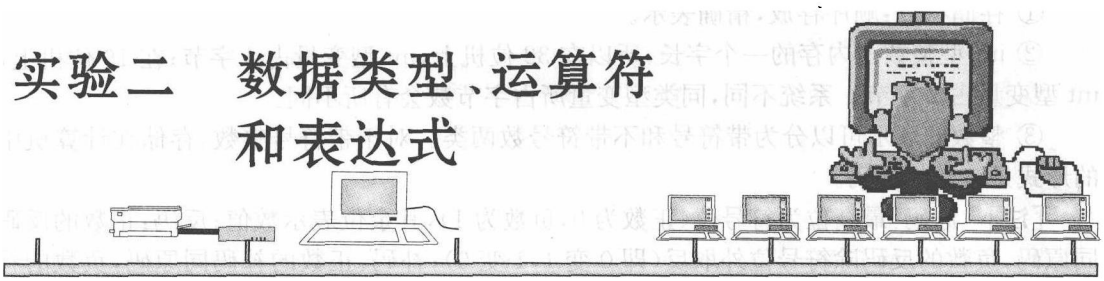
其实输入提示不可能通过 scanf()输入格式串实现，而应通过调用格式输入函数 printf()实现。示例如下：

```
#include<stdio.h>
#include<math.h>
main(){
    double a,b,c,s,area;
    printf("enter a,b,c: ");
    scanf("%lf%lf%lf",&a,&b,&c);
    s=(a+b+c)/2.0;
    area=sqrt(s*(s-a)*(s-b)*(s-c));
    printf("%6.2f,%6.2f,%6.2f\n",a,b,c);
    printf("s=%-8.2f,area=%-8.2f\n",s,area);
}
```

编译源程序，运行 a.out 可执行文件：

```
$cc cw1.c -lm↙
$a.out↙
enter a,b,c: 3 4 5↙                    //在提示后输入 3 项数据并用空格分隔
3.00,  4.00,  5.00
s=6.00     ,area=6.00
```

实验二　数据类型、运算符和表达式



学习重点

- 数据类型和存储形式
- 整数、实数、单个字符与字符串常量的表示形式
- 常量、变量的定义
- 运算符优先级和结合方向
- 表达式类型和求值
- 运算过程与类型转换
- 输入输出

一、要点综述

1. 数据与类型

常数、变量、函数和表达式是程序中的基本操作对象，它们都隐式或显式地与一种数据类型相联系。数据类型是指数据的内在表现形式，数据类型确定数据的存储形式和值域，数据类型也确定相应数据所允许执行的运算等。

2. C 语言的数据类型

C 语言的数据类型有整型、字符型、实型（单精度和双精度）、枚举类型、数组类型、结构体类型、共用体类型、指针类型和空类型。

3. 不同类型的数据在计算机中的存储格式也不同

(1) 整数(int)在 32 位机以 4 字节 32 位存放，其存放形式如图 3.2 所示。

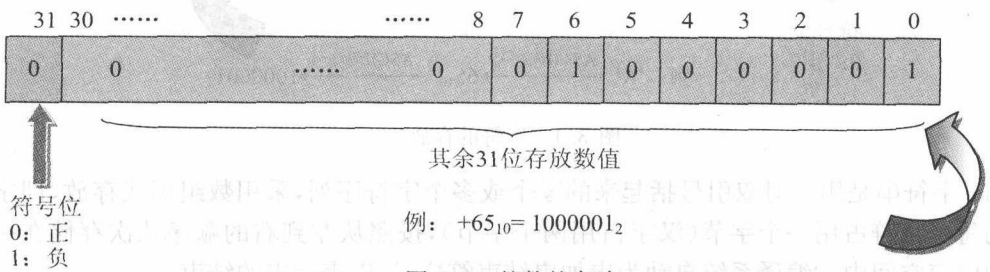


图 3.2　整数的存放

① 存储特点：顺序存放，精确表示。

② int 型整数占内存的一个字长，所以在 32 位机上，int 型变量占 4 字节；在 16 位机上，int 型变量占 2 字节。系统不同，同类型变量所占字节数会有所不同。

③ 整数的表示可以分为带符号和不带符号数两类。对于带符号整数，存储在计算机中的形式是该数的补码。

［注］ 原码：最高位为符号位（正数为 0，负数为 1），其余位表示数值；反码：正数的反码同原码，负数的反码除符号位外取反（即 0 变 1，1 变 0）；补码：正数的补码同原码，负数的补码等于反码加 1（即原码除符号位外取反加 1）。

（2）实数（float）在内存中一律以指数形式存放，占 4 字节 32 位存放，其存放形式分为两个部分：指数部分（阶码）和数值部分（尾数）。这里，以实数 $28.375_{10} = 11100.011_2 = 0.11100011 \times 2^{101}$（标准指数形式）为例，其存放形式如图 3.3 所示。

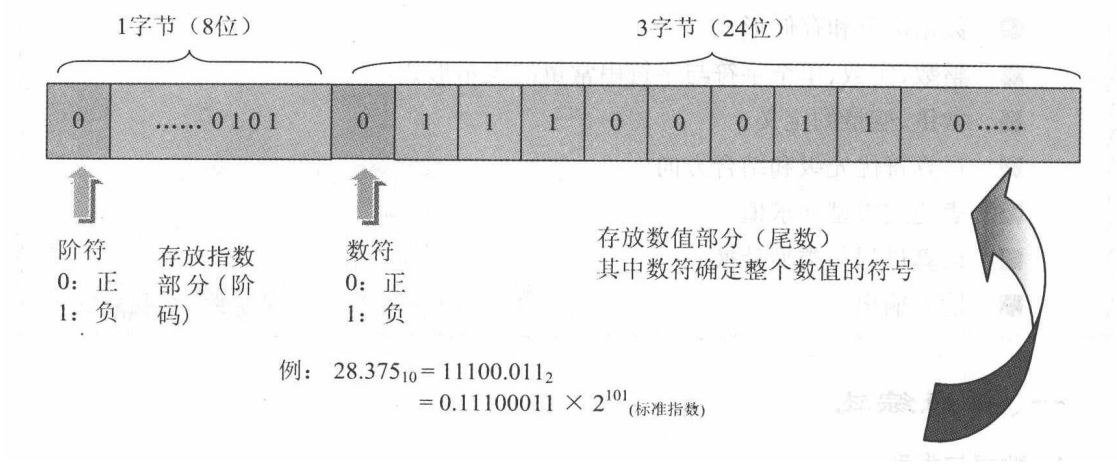


图 3.3 实数的存放

存储特点：指数与小数分别存放，近似表示。

（3）字符型（char）数据以 1 字节 8 位存放。字符“A”为例，其 ASCII 码为 $65_{10} = 1000001_2$，其存放形式如图 3.4 所示。

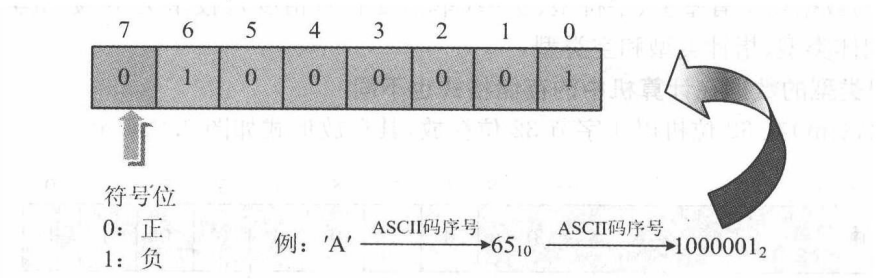


图 3.4 字符的存放

（4）字符串是用一对双引号括起来的零个或多个字符序列，采用数组形式存放，即字符串中的每个字符占用一个字节（汉字占用两个字节），按照从左到右的顺序依次存储在一段连续的内存空间中。编译系统自动为串加串结束符‘\0’，以表示串的结束。

注意：由于字符串末尾都被加上串结束符，所以字符串在内部表示所占的空间要比实际字符多一个字节，如图 3.5 所示为单个字符“A”和字符串“A”的存储对比。

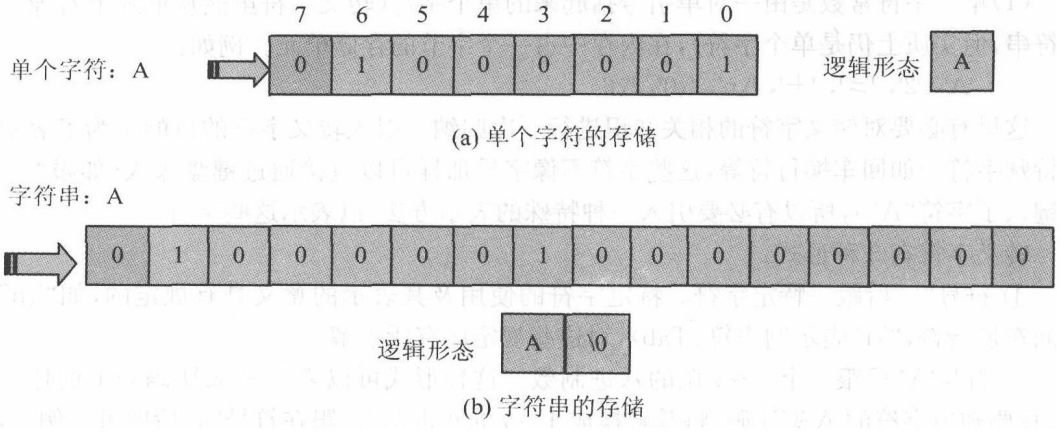


图 3.5 字符与字符串的存储

4. 整常数的表示

(1) 十进制整常数：取值范围 $-2^{n-1}\sim2^{n-1}-1$，n 为机器字位数(字长)；例如：

123，100

(2) 长整数常数：取值范围 $-2^{2n-1}\sim2^{2n-1}-1$，占两个机器字长(这里取存储字长两个机器字，注意有些系统规定不同)；表示形式是在整常数后加字母“l”或“L”。例如：

123l，123L，100l，100L

(3) 八进制整常数：由数字 0 打头，后跟一个八进制数。例如：

0123，0100，05，032727

(4) 十六进制整常数：以“0x”打头，后跟一个十六进制数，例如：

0x10，0x1A，0x1F，0x20

5. 实常数的表示

在 C 语言中单精度和双精度实常数的表示方法相同，编译程序总是把实常数处理成双精度后再进行运算。

实常数由三部分组成：整数部分，小数部分，指数部分。例如：

整数部分	小数部分	指数部分
.−3246.	51789	E+10

实常数的组成规则：

(1) 三个部分可以有一部分或两部分缺省，但整数部分和小数部分不能同时缺省。

(2) 如果一个实常数带小数点，则小数点左右应至少有一边有数字。

(3) 如果实常数带“e”或“E”，则两边都要有至少一位数字。

(4) 指数部分必须是整数。

因此，实常数一般有两种书写形式：小数形式和指数形式。

实数的取值范围：$10^{-38}\sim10^{38}$。当数值大于 10^{38} 时产生上溢，小于 10^{-38} 时产生下溢。一般一个普通实常数在机内表示时具有 6～7 位十进制有效数字，双精度实数具有 16 位十进制有效数字。

6. 字符常数

字符数据包括单个字符(简称字符)和字符串两种。

(1)单个字符常数是由一对单引号括起来的单个字符(转义字符虽然从形式上看像一个字符串,但实质上仍是单个字符),在内存中占一个字节的存储单元。例如:

'A', '2', '#', '+', '\n', '\0', '\t'

这里有必要对转义字符的相关知识进行一次归纳。引入转义字符的目的是为了表示一些特殊字符。如回车换行符等,这些字符不像字母那样可以直接通过键盘输入(如敲"A"键就键入了字符"A"),所以有必要引入一种特殊的表示方法,以表示这些字符。

转义字符有 3 种形式:

① 符号"\"后跟一特定字符。特定字符的使用及其表示的意义是有规定的,如"\n"表示回车换行符,"\t"表示制表符〈Tab〉,对这些规定应有所了解。

② 符号"\"后跟一个 1～3 位的八进制数。这种形式可以表示 ASCII 码表上的任一字符,只要知道字符的 ASCII 码,将其转换成 1～3 的八进制数,跟在符号"\"后即可。例如,如果知道字母"A"的 ASCII 码为 65,转换成 3 位八进制数为 101。则就可以用"\101"来表示字符"A"。

③ 符号"\x"后跟一个 1～2 位的十六进制数。

(2) 字符串常数是用一对双引号括起来的零个或多个字符序列。在存储系统中以数组形式存放,并且编译系统自动为字符串加串结束符"\0",以表示串的结束。

C 语言约定用"\0"作为字符串结束标志,它占内存 1 个字节的空间,但不计入字符串的长度。例如"c_language"共 10 个字符,但在内存中占 11 个字节,最后一个字节存放"\0",该字符串长度为 10,所需的存储空间大小为 11 个字节。

7. 符号常数

为了提高程序的可读性和常数修改的一致性,C 语言允许将程序中多次出现的常数定义为一个标识符,称之为符号常数。符号常数必须遵守先定义后使用的规则。一般在程序的开头定义,以后程序中要用到此常数的地方可用此标识符来代替。

符号常数定义形式为:

```
#define  标识符  常数
```

例如:

```
#define  PI   3.14159
#define  EOF  -1
```

(1) 符号常数习惯上用大写字母表示。

(2) 经过定义,标识符(符号常数)与后面的常数就具有等价的作用,使用符号常数的目的是为了便于大程序的修改和阅读直观。

8. 标识符的作用与命名规则

标识符的作用:标记常数、变量、自定义数据类型、函数及程序的名字。

标识符命名规则:

(1) 以下划线"_"或字母中任一字符打头。

(2) 在第一字符之后,可以是任意的字母、下划线或数字组成的字符序列,这个字符序

列可以是空串。

注意：

① 不要与保留字同名。

② C语言中大小写字母有不同的含义，即区分大小写。“name”和“Name”在C语言中是两个不同的标识符。

③ 习惯上除了符号常数，一般都用小写字母表示。

④ 标识符长度不限，但只识别前8个字符(UNIX系统)。

9. 变量的定义

变量就是在程序运行过程中其值可以改变的量。其实质是内存中的一个存储单元。

(1) 变量必须先定义后使用。

(2) 变量的定义包括变量名、变量的类型、变量的初始值和变量的属性。变量的属性主要说明变量的作用域及存储类别。作用域从字面上理解就是变量的作用范围，此范围内可以对变量进行操作。定义一个变量的同时也就指明了变量的作用域。按作用域来分，变量可以有局部变量和全局变量(外部变量)两大类(进一步的说明见函数一章)。

(3) 变量定义的一般形式：

```
类型标识符 变量标识符表列;
```

(4) 变量定义实际完成以下功能：

① 使一变量属于某一确定的类型，即规定该变量的取值范围及它能执行的运算操作等。

② 编译时通知系统为其分配相应的存储单元，并确定其存储格式，同时根据其定义检查该变量所进行的运算的合法性。

10. 运算符优先级和结合方向

C语言的运算符除具有优先级外，还具有结合方向等特性，规定：

(1) 同一优先级的运算符级别相同，运算次序由结合方向决定。如：

```
3*5/4
```

“*”和“/”优先级相同，结合方向为从左到右，运算次序为先乘后除。又如：

```
a=b=c=0
```

由于赋值运算符“=”的结合方向为从右到左，因此该表达式的运算次序为a=(b=(c=0))。

(2) 不同的运算符要求有不同的运算对象。按运算符所要求的对象个数分类，有：

① 单目运算符，如-a，i++，--i等。

② 双目运算符，如3*5/4。

③ 三目运算符，如x? a:b。

(3)运算符与对象的结合：C编译程序在处理时尽可能多地“自左而右”将若干字符组成运算符(在处理标识符、关键字时也按同一原则处理)。例如：

```
a+++b+++c+++
```

处理为(a++)+(b++)+(c++)。

学习运算符时应特别注意以上运算符的三个特性。

11. 表达式的组成

(1)表达式可由常数、变量、函数调用、运算符及圆括号内的表达式组成。

(2)单个常数、单个变量或单个函数调用都可以看作是一个表达式。

在表达式的学习过程中,需要注意表达式的书写和运算规则。C 语言中任何表达式都有一个确定的值,应关注表达式的值和类型。表达式运算的最后结果作为表达式的值,该值的类型即为表达式的类型。

12. 算术运算符和算术表达式

C 语言提供的算术运算符有:+(加法)、-(减法或负号)、*(乘法)、/(除法)、%(模或取余)。

(1) 算术运算符的优先级:

运算符:负号运算符→乘法、除法、模运算符→加法、减法运算符

优先级:高————————————————→低

(2) 算术运算符的结合性为从左向右。

(3) 模运算符的"%"的两个操作数必须为整型。

(4) 两整型量相除商为整数。

(5) 可以进行整型、单精度和双精度量的混合运算,此时先将整型、单精度量都转换为双精度量,然后进行运算。

(6) 表达式的计算和类型的转换都是逐步进行的。

(7) 计算时应特别注意运算符的三个特性及数据类型,特别是除法运算。

13. 自增运算符"++"和自减运算符"--"

(1) 自增、自减运算符,是两个特殊的单目运算符,其运算对象只能是变量。

(2) 前缀自增、自减运算符优先级是 2,结合方向是右结合。后缀自增、自减运算符的优先级是 1,结合方向是左结合。

(3) 自增、自减运算符的优先级高于其他算术、关系、逻辑、赋值运算符。

(4) 自增、自减运算符操作结果分别使变量增 1 或减 1,但对表达式的运算结果的影响与自增、自减运算符的前置或后置有关。

在自增、自减运算符出现在表达式中的运算符多于一个的情况下:自增、自减运算符放到变量前(前置方式)时,变量先做自增或自减运算,再以变化后的变量值参加表达式中的其他运算;自增、自减运算符放到变量后(后置方式)时,变量先参加表达式中的其他运算,即先将操作数 i 的值作为这个自增、自减表达式的值确定下来,再做自增或自减运算。

若以++i,i++,--i,i--形式分别作为单一表达式语句操作时,前置和后置的运算结果是相同的,即自增、自减运算符仅用于实现变量的增 1、减 1 运算时,前置和后置的结果相同。

(5) 当出现多个"+"或"-"运算符时,C 语言规定自左向右取尽可能多的符号组成运算符。如表达式"a+++b+++c++"相当于"(a++)+(b++)+(c++)"。

(6) 多数系统函数参数的求值顺序是从右向左的。如设 i 原值为 8,则函数调用语句 printf("%d,%d\n",i,++i);将首先计算输出参数表达式++i,结果是输出参数表达式的值为 9,即 i 自增 1 后为 9;然后计算输出参数表达式 i 的值,也为 9,因此该 printf 的输出值为"9,9"。

14. 关系运算符和关系表达式

C 语言提供的关系运算符有:

<(小于),<=(小于等于),>(大于),>=(大于等于),==(等于),!=(不等于)。

(1) 关系运算符的结合性为从左向右。

(2) 综合运算时相关运算符的优先级:

运算符:算术运算符 → <,<=,>,>= → ==,!= → 赋值运算符

优先级:高——————————————→低

(3) 关系成立(为真)时,关系表达式的值为1;关系不成立(为假)时,其值为0。即以1代表"真",以0代表"假",所以C语言的关系表达式的值可以看成是一个整型量,可参与通常的算术运算。

(4) 在C语言中,可以把表达式理解为一个条件,即表达式的值非0时理解为"真",表示条件成立;当表达式的值为0时理解为"假",表示条件不成立。

(5) 表达式的运算是逐步进行的。

(6) 正确区分赋值运算符"="和关系运算符"==".

(7) 不要直接进行实型量相等的比较,因为实型量是近似值。

15. 逻辑运算符和逻辑表达式

C语言提供的逻辑运算符有:!(逻辑非),&&(逻辑与),||(逻辑或)。

(1) 运算符"!"的结合性为从右向左,"&&"和"||"的结合性为从左至右。

(2) 综合运算时相关运算符的优先级:

运算符:! → 算术运算符 → 关系运算符 → && → || → 赋值运算符

优先级:高——————————————→低

(3) C编译系统在给出逻辑运算结果时,以数值1代表"真",以0代表"假";但判断一个量是否为真时,以0代表"假",以非零代表"真",即把一个非零数值作为"真"。

(4) 根据结合性,对运算符"&&"和"||"有以下规定:

① 若"&&"的左操作数为零,则不再对右操作数进行计算,直接得到运算结果为0。

② 若"||"的左操作数为非零,则不再对右操作数进行计算,直接得到运算结果为1。

这是逻辑运算符"&&"和"||"的一个特殊属性。由运算符"&&"或"||"连接的表达式按从左到右的顺序进行求值,即运算符"&&"和"||"保证左边的运算分量优先得到计算。就是说,由"&&"和"||"构成的逻辑表达式严格地从左到右进行计算,并且一旦根据左分量的值就能确定整个逻辑表达式的值时,就不再对右操作数进行求值。因此在逻辑表达式的求解中,并不是所有的逻辑运算符都被执行,只是在必须执行下一个逻辑运算符才能求出表达式的解时,才执行该运算符。

对表达式"a && b && c",只有a为真(非0)时,才需要判别b的值;只有在a和b都为真的情况下才需要判别c的值。只要a为假(0),就不必判别b和c(此时整个表达式已确定为假);如果a为真,b为假,则不判别c。

对表达式"a || b || c",只要a为真(非0),就不必判断b和c。只有a为假,才判别b,a和b都为假才判别c。

设:int a=1,b=2; 则:

① 表达式"(a=0) && (b=1)"操作后结果为:表达式的值为0,且a=0, b=2。

② 表达式"(a=2) || (b=0)"操作后结果为:表达式的值为1,且a=2, b=2。

16. 赋值运算符与赋值表达式

赋值表达式形式：

变量　　=　　表达式

↑　　↑　　↑

左操作数　　赋值运算符　　右操作数

(1) 赋值运算符的结合方向为自右向左。

(2) 左操作数必须是变量。

(3) 赋值表达式的运算结果是将赋值运算符右边的表达式的值赋给左边的变量。

(4) 左操作数(变量)的值和类型，即为表达式的值和类型。

(5) 若赋值运算符两边操作数类型不一致，但都属数值型和字符类型时，则"="右边的表达式值将自动转换成左边变量的类型(即赋值表达式的类型)。

(6) 根据赋值表达式的定义，又可递归得到多重形式(串联)赋值表达式：

$$i=j=k=0 \xleftrightarrow{\text{等价}} i=(j=(k=0))$$

(7) 复合赋值运算符组成形式：

双目操作符=

C语言中提供了5种算术复合赋值运算符：+=，-=，*=，/=，%=。另外还有5种位运算复合赋值运算符：≪=，≫=，&=，^=，|=。

复合赋值表达式形式：

变量　双目运算符=　表达式

复合赋值表达式运算规则：

变量=变量　双目操作符　(表达式)

注意：赋值运算符右边的表达式是作为一个整体参与运算的。转换时注意加括号。

17. 逗号运算符与逗号表达式

逗号表达式一般形式：

表达式1，表达式2，表达式3，……，表达式n

(1) 逗号运算符的结合性是从左向右。

(2) 逗号运算符的优先级在所有运算符中是最低的。

(3) 逗号运算符连接两个或两个以上的表达式，取最后一个表达式(表达式n)的值和类型作为该逗号表达式的值和类型。

(4) 求解过程：先求表达式1，再求表达式2，……依次计算，直至最后一个表达式。

(5) 运算是逐步进行的，需要注意变量值的变化。

18. 条件运算符与条件表达式

条件运算符为"?:"，它是一个三目运算符。

条件表达式的一般表达式：

e1 ? e2 : e3

其中，e1，e2，e3分别代表三个不同的表达式。

(1) 条件表达式的运算结果是，若e1成立，则取e2的值作为整个表达式的值，否则取e3的值作为整个表达式的值。

(2) 条件运算符的优先级仅高于赋值运算符和逗号运算符。

(3) 条件运算符的结合方向为从右向左。

19. 类型转换

(1) 字符、整型、单精度、双精度量混合运算时类型转换的三种基本规则为：

① 属同一类型，但存储单元长度不同之间的必然转换，短字节向长字节转换。

② 整型与浮点数之间的转换，低级向高级转换。

③ 整型量有符号与无符号之间的转换以及与长字节的转换。

混合运算时类型转换如图 3.6 所示。

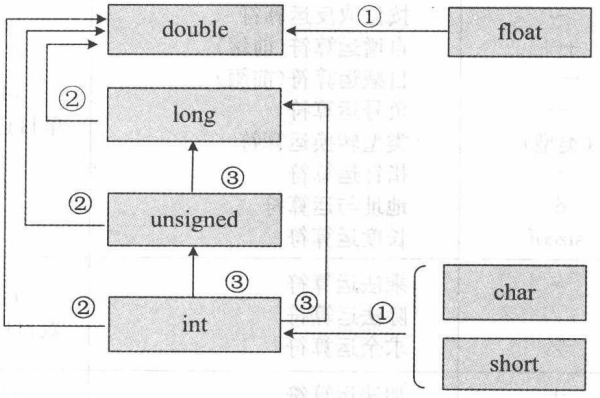


图 3.6 数据类型间的转换

(2) 赋值运算时，如赋值运算符两边数据类型不一致，会产生赋值类型转换，将右边表达式的类型转换为左边变量的类型。

(3) 强制类型转换语法形式为：

```
(类型标识符)表达式
```

其作用是将表达式的值显式地转换成所指定的类型。

20. 长度运算符

语法形式：

```
sizeof(类型标识符)      或      sizeof 表达式
```

其作用是给出运算对象以字节为单位的存储单元的长度。

21. 表达式语句

语法形式：

```
表达式;
```

表达式语句即表达式后加分号构成的语句。其中分号是语句的终结符，是语句的组成部分。

22. C 语言的运算符、优先级、结合性和运算对象

C 语言提供的各种运算符及其优先级、结合性和运算对象汇总列于表 3.1。

表 3.1

优先级	类别	运算符	含义	要求运算对象的个数	结合方向
1	初等运算符	( ) [ ] —>) . ++ ——	圆括号 下标运算符 指向结构体成员运算符 结构体成员运算符 自增运算符(后缀) 自减运算符(后缀)	单目运算符 单目运算符	自左至右
2	单目运算符	! ~ ++ —— — (类型) * & sizeof	逻辑非运算符 按位取反运算符 自增运算符(前缀) 自减运算符(前缀) 负号运算符 类型转换运算符 指针运算符 地址与运算符 长度运算符	(1) 单目运算符	自右至左
3	算术运算符	* / %	乘法运算符 除法运算符 求余运算符	(2) 双目运算符	自左至右
4		+ —	加法运算符 减法运算符	(2) 双目运算符	自左至右
5	位运算	≪ ≫	左移运算符 右移运算符	(2) 双目运算符	自左至右
6	关系运算	< <= > >=	小于 小于等于 大于 大于等于	(2) 双目运算符	自左至右
7		== !=	等于 不等于	(2) 双目运算符	自左至右
8	位运算	&	按位与运算符	(2) 双目运算符	自左至右
9		^	按位异或运算符	(2) 双目运算符	自左至右
10		|	按位或运算符	(2) 双目运算符	自左至右
11	逻辑运算	&&	逻辑与运算符	(2) 双目运算符	自左至右
12		‖	逻辑或运算符	(2) 双目运算符	自左至右
13	条件运算	? :	条件运算符	(3) 三目运算符	自右至左

(续)表 3.1

优先级	类别	运算符	含义	要求运算对象的个数	结合方向
14	赋值运算符	= += -= *= /= %= >>= <<= &= ^= |=	赋值运算符	(2) 双目运算符	自右至左
15	逗号	,	逗号运算符		自左至右

23. 输入输出补充

输入输出一般使用 scanf()和 printf()函数,但对字符数据的输入输出,使用 getchar()和 putchar()函数有时更为便利。

例 将输入(stdin)拷贝到输出文件(stdout)的拷贝程序。

算法描述:

```
读入一个字符
while(字符不是文件结束标志){
    输出刚才读入的字符;
    再读入下一个字符。
}
```

完整的程序:

```
#include <stdio.h>
void main(){
    char c;
    c=getchar();
    while( c != EOF ){
        putchar(c);
        c=getchar();
    }
}
```

运行测试结果:

```
abcde12345
abcde12345
^Z                          //UNIX 系统按^D
```

该程序可优化成如下形式:

```
void main(){
    char c;
    while((c=getchar())!=EOF)
```

```
        putchar(c);
    }
```

说明：

(1) 符号常数 EOF 是文件结束标志。在“stdio. h”中有如下定义：

```
#define  EOF  -1
```

(2) 程序运行时，getchar()遇到文件结束标志返回值－1。

UNIX 系统键盘(标准输入文件 stdin)的结束符号为〈CTRL〉＋D。一般微机系统键盘(标准输入文件 stdin)的结束符号为〈CTRL〉＋Z。

(3) 键盘输入的是一个字符流，空格是字符，回车也是字符，除非使用格式符作数值的转换处理。

(4) C 表达式高效灵活，在一个表达式中可以完成多种功能的运算。这种运算是按算符的优先级和结合性逐步完成的。因此要注意表达式的实际运算顺序。

二、实验操作

(一) 目的要求

(1) 掌握 C 语言数据类型，熟悉如何定义一个整型、字符型、实型变量以及对它们赋值的方法。

(2) 掌握数据的输入输出的方法，能正确使用各种格式转换符。

(3) 掌握 C 的各类运算符的使用，特别是自增和自减运算符的使用；掌握各种表达式的求值。

(4) 了解函数参数的求值顺序。UNIX 等系统采用自右至左的顺序求函数参数的值，因此右边参数中的变量值的改变会影响左边参数中相应变量的值和表达式的值。

(5) 进一步熟悉 C 程序的编辑、编译、连接和运行的过程。

(二) 范例

1. 输入并运行以下程序，分析结果。

源程序文件 prg2_1. c 清单：

```
#include <stdio. h>
main(){
  int a,b;
  char c1,c2;
  float d,e;
  double f,g;
  long m,n;
  unsigned int p,q;
  a=61; b=62;
  c1='a'; c2='b';
```

```
    d=3.56; e=-6.78;
    f=3157.890121; g=0.123456789;
    m=50000; n=-60000;
    p=32768; q=4000;
    printf("sizeof:%d, a=%d, b=%d\n",sizeof(int),a,b);
    printf("sizeof:%d, c1=%c, c2=%c\n",sizeof(char),c1,c2);
    printf("sizeof:%d, d=%-6.2f, e=%-6.2f\n",sizeof(float),d,e);
    printf("sizeof:%d, f=%-15.6f, g=%-15.12f\n",sizeof(double),f,g);
    printf("sizeof:%d, m=%ld, n=%ld\n",sizeof(long),m,n);
    printf("sizeof:%d, p=%u, q=%u\n",sizeof(unsigned),p,q);
}
```

操作步骤如下：

(1) 利用正文编辑程序将 prg2_1.c 文件建立在用户自己的当前目录上。

(2) 发出编译命令：

```
$cc prg2_1.c ↙
```

(3) 运行 a.out 执行文件：

```
$a.out ↙
```

(4) 运行结果：

```
sizeof:4, a=61, b=62
sizeof:1, c1=a, c2=b
sizeof:4, d=3.56, e=-6.78
sizeof:8, f=3157.890121, g=0.123456789000
sizeof:4, m=50000, n=-60000
sizeof:4, p=32768, q=4000
```

提示：

(1) 系统不同，各类型数据的存储单元大小有所不同。

(2) "%-6.2f"中的格式符参数'-'规定相应输出数据以输出域为界靠左对齐输出。

(三) 实验内容

1. 修改范例 prg2_1.c，用 scanf()函数输入数据而不用赋值语句，scanf()函数调用语句如下：

```
scanf("%d%d%c%c%f%f",&a,&b,&c1,&c2,&d,&e);
scanf("%lf,%lf,%ld,%ld,%u,%u",&f,&g,&m,&n,&p,&q);
```

输入的数据如下：

```
61 62ab3.56 -6.78 ↙
3157.890121, 0.123456789,50000,-60000,32768,4000 ↙
```

应输出如下结果：

```
sizeof:4, a=61, b=62
sizeof:1, c1=a, c2=b
```

```
sizeof:4, d=3.56, e=-6.78
sizeof:8, f=3157.890121, g=0.123456789000
sizeof:4, m=50000, n=-60000
sizeof:4, p=32768, q=4000
```

要求:运行修改后的程序并分析运行结果;然后将 scanf()函数中的"%lf"和"%ld"分别改为"%f"和"%d",运行程序并分析结果。

提示:lf 和 ld 格式符分别用于输入 double 型和 long 型数据。若输入格式符与相应存储单元规定的格式不符,则将引起存储错误,最终将会导致运算结果的错误。

2. 输入并运行以下程序:

```
#include <stdio.h>
main(){
    int i,j,m,n;
    i=8; j=10;
    m=++i; n=j++;
    printf("%d,%d,%d,%d\n",i,j,m,n);
}
```

然后分别做以下改动并运行:

(1) 将第 4 行改为:

```
m=i++; n=++j;
```

(2) 程序改为:

```
main(){
  int i,j,m,n;
  i=8; j=10;
  printf("%d,%d\n",i++,j++);
}
```

(3) 在(2)的基础上,将 printf 语句改为:

```
printf("%d,%d\n",++i,++j);
```

(4) 再将 printf 语句改为:

```
printf("%d,%d,%d,%d\n",i,j,++i,++j);
```

(5) 程序改为:

```
main(){
  int i,j,m=0,n=0;
  i=8; j=10;
  m+=i++; n-=--j;
  printf("i=%d,j=%d,m=%d,n=%d\n",i,j,m,n);
}
```

要求:上机前先人工分析程序,写出应得结果,与上机结果对照。

提示:注意函数参数的求值顺序,如 UNIX,Turbo C,VC++等都按自右至左的顺序求

函数参数的值，因此右边参数中变量值的改变会影响左边参数中相应变量或表达式的值。C标准没有具体规定函数参数的求值顺序，但每一个C编译程序都有自己的确定顺序，程序设计者应正确了解所使用系统的具体规定，以避免使用会引起二义性的语句。

3. 求下面算术表达式的值。要求在上机实验前做静态计算，并写好验证程序，待上机时验证，对比结果，分析原因。

(1) 3.5+1/2+56%10

(2) (a++*1/3)　　设 a=2

(3) x+a%3*(int)(x+y)%2/4　　设 x=2.5，a=7，y=4.7

(4) (float)(a+b)/2+(int)x%(int)y　　设 a=2，b=3，x=3.5，y=2.5

(5) x=(x=++y,x+5,x/5)　　设 x=3，y=4

(6) (x>=y>=2)? 1:0　　设 x=4，y=3

提示：

(1) 注意常量的隐含类型和相应运算规则。

(2) 表达式的运算是按运算符的运算规则逐步进行的。

4. 写出下面各逻辑表达式的值，设 a=3,b=4,c=5。要求在上机实验前做静态计算，并写好验证程序，上机后验证，对比结果，分析原因。

(1) b>c&&b==c

(2) !(a>b)&&!c||1

(3) !(x=a)&&(y=b)&&0

(4) !(a+b)+c-1&&b+c/2

5. 写出下面表达式运算后 a 的值。设原来 a=12。要求在上机实验前做静态计算，并写好验证程序，上机后验证，对比结果，分析原因。

(1) a+=a　　(2) a*=2+3

(3) a/=a+a　　(4) a+=a-=a*=a

6. 根据下列计算公式，分别编写程序实现各相应功能，并验证结果：

(1) 球的体 $V=\frac{4}{3}\pi r^3$

(2) 并联电阻的电阻值 $r=1/\left(\frac{1}{r1}+\frac{1}{r2}\right)$

(3) $x=(3.31\times10^{18}+2.10\times10^{-7})/(7.16\times10^{5}+2.01\times10^{3})$

(4) $y=\frac{a\times b}{c+\frac{d}{e}}$

7. 编程实现：输入一个小写字母，将其转换为大写字母输出。

8. 编程实现：分别计算出圆锥体的底圆周长、底面积、侧面积和体积，结果取小数点后2位。圆锥体的底半径和高都在程序运行过程中通过键盘输入。

（四）常见错误分析

C语言没有指定同一运算符中多个操作数的计算顺序（“&&”、“||”、“?:”和“,”运算符除外），即不相关子表达式的计算顺序不确定。例如，双目运算符的两个操作数是不相关的子表达式，由于C语言没有规定不相关子表达式的计算顺序，不同的编译器所采用的优化方

法不同，不同环境下的编译器配置可能不同，所产生的编译代码也可能会不同，因此不相关子表达式的计算顺序就可能不同。

对于非赋值运算的不相关子表达式，计算顺序的不同并不会影响整个表达式的计算结果。如果是以赋值运算作为子表达式，计算顺序的不同可能会导致整个计算结果的不同。例如：

```
for(i=0;i<n; )
a[i]=i++;
```

问题是：数组下标 i 是引用旧值还是新值？对这种情况编译器的解释不同，并因此产生不同的结果。最好将该语句改写为以下形式：

```
for(i=0;i<n;i++ )
a[i]=i;
```

类似地，C 语言也没有指定函数各参数的求值顺序。因此，语句

```
printf("%d %d\n",++n,power(2,n));
```

在不同的编译器中可能会产生不同的结果，这取决 n 的自增运算是在 power 调用之前还是之后执行。解决的办法是把该语句改写成下列形式：

```
++n;
printf("%d %d\n",n,power(2,n));
```

函数调用，嵌套赋值语句，自增、自减运算符都有可能产生“副作用”——在对表达式求值的同时，修改了某些变量的值。在有副作用影响的表达式中，其执行结果同表达式中的变量被修改的顺序之间存在着微妙的依赖关系。

在任何一种编程语言中，如果代码的执行结果与求值顺序相关，则都是不好的程序设计风格。我们有必要了解哪些问题需要避免，如果不知道这些问题在各种机器上是如何解决的，就最好不要尝试运用某种特殊的实现方式。

为了保证所需要的计算顺序，在程序设计中应注意以下几点：

(1) 掌握 C 语言对各个运算符的优先级和结合性的规定。特别要注意区分具有同样外形而根据上下文语义不同的运算符，如+、-、*、& 等。

(2) C 语言中的“&&”、“||”、“?:”和“,”运算符保证计算顺序严格从左到右进行。例如，在表达式“++a||++b&&++c”中，按优先级关系，“&&”高于“||”，但仍按从左到右顺序计算——先做++a，仅当执行后 a 的值为 0 时才会执行 || 右边的运算。

条件运算符的结合方向为从右往左。例如：“a>b? a : b>c? b : c”等价于“a>b? a : (b>c? b:c)”，若设 int a=2,b=3,c=4;，则先判断 a>b，因为不成立，再执行判断 b>c，所以结果为 4。

(3) 当一个表达式中出现多个有不同优先级的运算符时，可以通过加圆括号的方式改变运算顺序，例如：

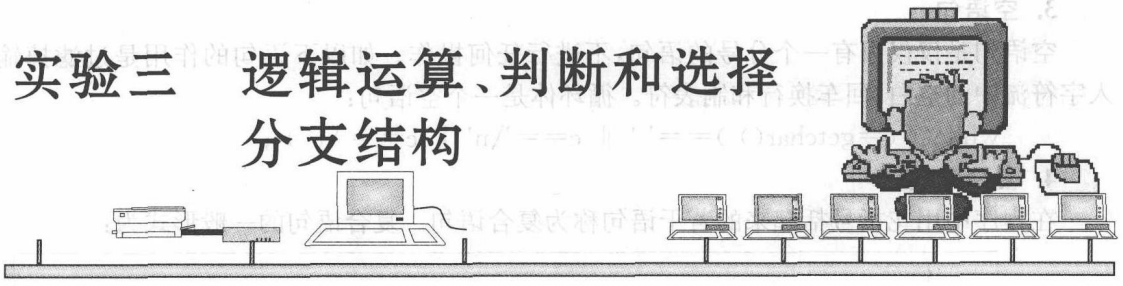
```
(b*b-4*a*c)/(2*a)
```

但对“&&”、“||”、“?:”和“,”运算符并不适用，它们保证计算顺序严格从左到右进行。

(4) 为了强调某部分是作为一个“整体”参与运算，可以引用“中间变量”，把“整体”的值赋给这个变量，然后由这个变量参与有关运算。例如：

```
twoa=2*a;
term1=-b/twoa;
```

实验三　逻辑运算、判断和选择分支结构



学习重点

- C 语句的分类
- 空语句与复合语句
- if 条件分支语句
- if 语句的嵌套
- switch 语句及其使用

一、要点综述

1. 结构化程序设计

任何程序都可以用三种基本结构(顺序结构、选择结构和循环结构)的组合来实现,这三种结构具有一个入口和一个出口的特征。结构化程序设计思想要求限制无条件转移语句的使用。

2. C 语句的分类

C 语句分类如表 3.2 所示。

表 3.2　C 语句的分类

分类		语句	说明
简单的顺序执行语句		函数调用语句	函数调用加分号构成
		表达式语句	表达式加分号构成
		空语句	只有一个分号的语句
		复合语句	花括号括起来的语句
流程控制语句	分支转移	if( )～else ～	条件分支语句
		switch	多分支选择语句
		break,continue,return,goto	无条件转移语句
	循环	while( ) ～	当循环
		do ～ while( )	直到型循环
		for( ) ～	for 循环
标号语句		以上语句添加标号,构成标号语句	

3. 空语句

空语句是仅仅只有一个分号的语句，不进行任何操作。如以下语句的作用是过滤掉输入字符流中的空白、回车换行和制表符。循环体是一个空语句：

```
while(( c=getchar() )==' ' || c=='\n' || c=='\t' ) ;
```

4. 复合语句

在程序中用花括号括起来的若干语句称为复合语句。复合语句的一般形式为：

```
{
    数据说明部分;
    执行语句部分;
}
```

(1) 在复合语句内部定义的变量，其作用域仅限于该复合语句的内部。

(2) 执行语句部分可以使用表 3.2 中给出的任意语句。

(3) 复合语句在语法上等价于一个简单语句，凡是单一语句可以出现的位置都可以使用复合语句。

(4) 复合语句可以嵌套。

5. 顺序结构

顺序结构是由简单的顺序执行语句构成的语句块，块中的语句是按照出现的先后顺序执行。

6. if 语句的基本形式

if 语句基本形式 1：

```
if(表达式)  语句 1  else  语句 2
```

if 语句基本形式 2(基本形式 1 的特例，当语句 2 为空时省略“else　语句 2”)：

```
if(表达式)  语句 1
```

(1) if 语句中的表达式可以是任何表达式，表达式的值为非零，执行语句 1，否则执行语句 2。

(2) 语句 1、语句 2 可以是简单的语句，此时要注意不要遗漏掉语句结束标志“;”。也可以是复合语句，此时要注意加大括号。

(3) 语句 1 为空时不能省略，此时可以用一条空语句表示。

7. if 语句的嵌套

(1) if 语句可以互相嵌套，常用的形式有：

① 在基本形式 1 的 else 后的语句 2 位置层层嵌套基本形式 1 语句。

```
if(表达式 1) 语句 1
else if(表达式 2)语句 2
  ……
else if(表达式 n)语句 n
else 语句 n+1;
```

② 在基本形式 1 中的语句 1 位置层层嵌套基本形式 1 语句。

```
if(表达式 1)
    if(表达式 2)
        if(表达式 2)语句 1
        else 语句 2
    else 语句 3
else 语句 4
```

③ 在基本形式 1 中的语句 1 位置嵌套基本形式 2 语句。

```
if(表达式 1)if(表达式 2)语句 1
            else 语句 2
```

(2) if 语句嵌套使用时,需要特别注意 else 与 if 的配对问题。C 语法规定 else 总是与它前面的同一语法层次中最接近的尚未配对的 if 配对。

(3) 进入 if 语句后,不管其层层嵌套有多复杂,最多只执行其中的一个语句。

8. switch 语句

switch 语句基本形式:

```
switch(表达式){
    case 常量表达式 1:语句 1
    case 常量表达式 2:语句 2
      ……
    case 常量表达式 n:语句 n
    [default :语句 n+1]
}
```

switch 语句的功能是:根据表达式的值来选择执行若干语句或语句序列中的一个。但若要真正起到多分支选择执行功能,须使用 break 语句配合实现。形式如下:

```
switch(表达式){
    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]
}
```

与基本形式相比,这种形式使用更为频繁,也更实用。

(1) break 语句用于终止 switch 语句的执行,跳出 switch 结构。

(2) 表达式和常量表达式要求是整型或字符型,且两者类型相同。

(3) 常量表达式的值必须互不相同,它们只起相当于语句入口标号的作用,没有条件判断分支执行的功能。

(4) 执行过程是:首先计算"表达式",若表达式的值与常量表达式 i 的值相等,就顺序执行该常量表达式后的语句,否则执行 default 后的语句。

二、实验操作

(一) 目的要求

(1) 学会正确使用逻辑运算符和逻辑表达式,了解 C 语言表示和判断逻辑值方法。

(2) 进一步掌握各种表达式的使用。

(3) 熟练掌握 if 语句和 switch 语句。能利用 if 语句实现选择结构,利用 switch 语句实现多分支结构。

(二) 范例

1. 计算某年某月有几天。(注:能被 4 整除但不是 100 整数倍的年是闰年,能被 400 整除的年是闰年。)

源程序文件 prg3_1.c 清单如下:

```
main(){
  int yy,mm,len;
  printf("year,month:");
  scanf("%d%d",&yy,&mm);
  switch(mm){
     case 1:
     case 3:
     case 5:
     case 7:
     case 8:
     case 10:
     case 12:len=31; break;
     case 4:
     case 6:
     case 9:
     case 11:len=30; break;
     case 2:if(yy%4==0 && yy%100!=0 || yy%400==0)
              len=29;
            else
              len=28;
            break;
     default:printf("input error");
            break;
  }
```

```
        printf("The length of %d %d is\n",mm,len);
    }
```

运行结果：

```
year,month:1996 2
The length of 2 29 is
```

2. 编写程序求一元二次方程 $ax^2+bx+c=0$ 的根。求方程解的公式如下：

$$x_{1,2}=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$$

方程的根有以下几种可能：

(1) $a=0$ 且 $b\neq0$ 时，一个实根。

(2) $b^2-4ac=0$，有两个相等的实根。

(3) $b^2-4ac>0$，有两个不相等的实根。

(4) $b^2-4ac<0$，有两个共轭复根。

源程序文件 prg3_2.c 清单如下：

```
#include <math.h>
main(){
double a,b,c,disc,twoa,term1,term2;
  printf("enter a,b,c:");
  scanf("%lf%lf%lf",&a,&b,&c);
  if(a==0)
    if(b==0)
      printf("no answer due to input error\n");
    else
      printf("the single root is %f\n",-c/b);
  else{
    disc=b*b-4*a*c;
      twoa=2*a;
      term1=-b/twoa;
      term2=sqrt(fabs(disc))/twoa;
      if(disc<0)
        printf("complex roots\n real part=%fimag part=%f\n",
                term1,term2);
      else
        printf("real roots\n root1=%f root2= %f\n",
                term1+term2,term1-term2);
  }
}
```

说明：除变量定义语句外，主函数中实际只有三个语句，第一、第二个语句是输入方程的

系数 a,b,c；第三个语句是一个条件语句。最外层 if 后面又是一个条件语句，用来处理单根或输出错误信息；最外层 else 后面是一个复合语句。其中又有一个条件语句，是用来区别处理复根和实根的。

操作步骤如下：

(1) 利用正文编辑程序将 prg3_2.c 文件建立在用户自己的当前目录上。

(2) 发出编译命令：

```
$cc prg3_2.c ↙
```

(3) 运行 a.out 执行文件：

```
$a.out ↙
```

(4) 运行结果：

```
enter a,b,c:1 1 -12 ↙
real roots
root1=3.000000 root2= -4.000000
$a.out ↙
enter a,b,c:1 -2 -4 ↙
real roots
root1=3.236068 root2= -1.236068
$a.out ↙
enter a,b,c:1 -1 -3 ↙
real roots
root1=2.302776 root2= -1.302776
```

(三) 实验内容

1. 修改范例 prg3_2.c 程序，当方程有两个不同的实根时，先计算绝对值大的那个根，然后用公式 x2=c/(a * x1)计算绝对值较小的那个根。

2. 从键盘输入三条边 a,b,c，判断这三条边是否能组成三角形。若能组成三角形，判断是等边三角形、等腰三角形、直角三角形，还是一般三角形，并计算该三角形的面积。

提示：从键盘输入三条边长后，使用 if～else 语句测试所给数据是否满足构成三角形的充要条件。条件成立，进行各项判断和计算该三角形的面积并输出；条件不成立，则输出错误信息“DATA ERROR!”

3. 从键盘输入一个公民的月收入金额，试用下面所给的累进计算公式来计算并输出个人所得税款：3000 元以内不交税；超过 3000 元部分如小于 500 元，税率为 5%；若大于等于 500 元且小于 1500 元，税率为 10%；而大于等于 1500 元且小于 3000 元，税率为 15%；大于等于 3000 元且小于 5000 元，税率为 20%；大于等于 5000 元，税率为 25%。

4. 计算下列分段函数值：

$$f(x)=\begin{cases} x^2+x-6, & x<0 \text{ 且 } x\neq -3 \\ x^2-5x+6, & 0\leqslant x<10 \text{ 且 } x\neq 2 \text{ 及 } x\neq 3 \\ x^2-x-1, & \text{其他} \end{cases}$$

具体要求如下：

(1) 用 if 语句实现分支结构编程。自变量 x 与函数均用双精度类型。

(2) 自变量 x 用 scanf 函数输入，且输入前要有提示。结果的输出采用以下形式：

$$x=具体值, f(x)=具体值$$

(3) 分别输入 $x=-5.0,-3.0,1.0,2.0,2.5,3.0,5.0$，运行该程序。

提示：赋值表达式和由赋值表达式构成的赋值语句的左操作数必须是符合标识符规范的变量。

5. 用 scanf 函数输入一个百分制成绩(整型量)，将其转换成相应的成绩等级(A,B,C,D,E)后输出。规定 90～100 分为 A，80～89 分为 B，70～79 分为 C，60～69 分为 D，60 分以下 E。具体要求如下：

(1) 用 switch 语句实现分支。

(2) 在输入百分制成绩前要有提示。

(3) 在输入百分制成绩后，要判断该成绩的合理性，对于不合理的成绩(即大于 100 分或小于 0 分)应输出出错信息。

(4) 在输出结果中应包括百分制成绩与成绩等级。

(5) 分别输入百分制成绩—90，100，90，85，70，60，45，101，运行该程序。

提示：在 C 语言中，利用整数相除结果为整数的规则，通过一个整数除以 10 的方法，可去掉该整数的个位数，如 88/10 结果为 8，100/10 结果为 10。在程序中可利用这一点，直接用成绩 score/10 作为 switch 的表达式。若 score 定义为实型变量，则需要对表达式的值作强制类型转换，如(int)(score/10)。

6. 编写程序，检查输入的年份是否为闰年。

闰年的条件是：

(1) 能被 4 整除，但不能被 100 整除；

(2) 能被 4 整除，又能被 400 整除。

要求：输入不同的年份给以验证。

7. 下面的程序存在错误？若有错，请改正。

```
#include "stdio.h"
main(){
    float x,y; float z=50.8;
    scanf("%d",x); y=x/z;
    if(x=0); printf("enter error! \n")
    else printf("y=%d\n",y);
}
```

8. 输入一个字母字符，若为小写则将其转换为大写；然后利用 switch 语句判断该字母是否是元音字母并以“Yes”或“No”字样输出判断结果和该字母的大写 ASCII 码和小写 ASCII 码。

（四）常见错误分析

1. else 配对错误

【错误示例 3.1】 如范例 2，写一个程序求一元二次方程 $ax^2+bx+c=0$ 的根。若给出的源程序文件清单如下：

```
#include <math.h>
main(){
  double a,b,c,disc,twoa,term1,term2;
  printf("enter a,b,c:");
  scanf("%lf%lf%lf",&a,&b,&c);
  if(a==0)
    if(b!=0)
      printf("the single root is %f\n",-c/b);
  else{
    disc=b*b-4*a*c;
      twoa=2*a;
      term1=-b/twoa;
        ......
}
```

以上程序显然是错误的，因为 if 语句嵌套使用时，else 总是与它前面的同一语法层次中最接近的那个尚未配对的 if 相结合，故当 a≠0 时，以上程序将不做任何事情。按照 if 与 else 使用原则，该源程序应如以下所示：

```
#include <math.h>
main(){
  double a,b,c,disc,twoa,term1,term2;
  printf("enter a,b,c:");
  scanf("%lf%lf%lf",&a,&b,&c);
  if(a==0){
    if(b!=0)
      printf("the single root is %f\n",-c/b);
  }
  else{
    disc=b*b-4*a*c;
    twoa=2*a;
    term1=-b/twoa;
    ......
}
```

2. switch 没有与 break 语句正确匹配的错误

switch 只有与 break 语句正确配合使用，才真正具有多分支选择语句的功能，否则从某一入口进入后将顺序执行其后的语句。

【错误示例 3.2】 用 scanf 函数输入一个百分制成绩(整型量)，要求输出相应成绩的等级(A,B,C,D,E)后输出。规定 90～100 分为 A，80～89 分为 B，70～79 分为 C，60～69 分为 D，60 分以下 E。

若给出的源程序文件清单如下：

```
main(){
    int score,temp; char grade;
    scanf("%d", &score);
    if(score==100) temp=9;
    else
        temp=(score-score%10)/10;
    switch(temp){
        case 9: grade='A';
        case 8: grade='B';
        case 7: grade='C';
        case 6: grade='D';
        case 5:
        case 4:
        case 3:
        case 2:
        case 1:
        case 0: grade='E';
        default:printf("error\n");
    }
    if(grade>='A'&&grade<='E')
        printf("score=%d,grade=%c\n",score,grade);
}
```

在通过编译后运行该程序时，若输入：

```
95↙
```

则输出结果为：

```
error
score=95,grade=E
```

这显然是错误的，导致错误的原因是源程序中的 switch 语句没有与 break 语句配合使用。正确的源程序应如以下所示：

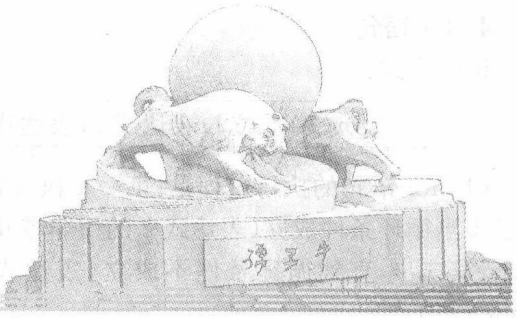
```
main(){
    int score,temp; char grade;
```

```
    scanf("%d",&score);
    if(score==100) temp=9;
    else
        temp=(score-score%10)/10;
    switch(temp){
        case 9: grade='A'; break;
        case 8: grade='B'; break;
        case 7: grade='C'; break;
        case 6: grade='D'; break;
        case 5:
        case 4:
        case 3:
        case 2:
        case 1:
        case 0: grade='E'; break;
        default:printf("error\n");
    }
    if(grade>='A'&&grade<='E')
        printf("score=%d,grade=%c\n",score,grade);
}
```

实验四　循环控制结构编程

学习重点

- while 语句与使用
- do～while 语句与使用
- for 语句与使用
- while，do～while 与 for 的区别
- 循环语句嵌套(多重循环)
- break 和 continue 语句的使用
- break 与 continue 语句的区别



一、要点综述

1. while 语句(当循环)

while 语句一般形式为：

```
while(表达式)　{循环体语句}
```

(1) while 语句的功能是：计算表达式的值，若为零，退出循环语句的执行；若为非零，执行循环体语句(或称循环体)，执行完毕后，再次执行循环语句，计算表达式的值……

(2) 特点是：先判断后执行，循环体有可能一次也不执行。通常用于事先不能确定循环次数的情况。

(3) 循环体可以是一个语句，也可以是多个语句。如果循环体包含一个以上的语句，应该用花括号括起来，以复合语句形式出现；如果不加花括号，则 while 语句的范围只到 while 后面的第一个语句处。

2. 循环控制变量

在表达式中出现的变量可称为循环控制变量。循环控制变量的使用必须注意的几个问题：

(1) 循环控制变量必须有正确的初值。

(2) 在循环体内应有改变其值的语句。

(3) 其值的改变应使得表达式的值最终趋向于零(循环结束条件)，否则将形成死循环。

3. do～while 语句(直到型循环)

语句形式：

```
do{
    循环体语句
}while(表达式) ;
```

(1) 特点是：先执行后判断。执行流程是：执行循环体；若表达式值为非零，则再次执行语句(循环体)，如此循环直至表达式的值为零。由于是先执行后判断，因此 do～while 语句中的循环体至少被执行一次。

(2) 适用于循环体至少执行一次，且总次数不确定的情况。

(3) do～while 语句与 while 语句类似，只不过是将条件表达式从语句的开始移到了尾部。

4. for 语句

语句形式：

```
for(表达式 1;表达式 2;表达式 3){循环体语句}
```

(1) for 语句的执行过程是：第一步执行表达式 1，第二步执行表达式 2(循环条件判断)，第三步当循环条件成立(表达式 2 的值非零)时执行循环体中的语句及表达式 3；然后重复第二步、第三步，直到循环条件不成立(表达式 2 的值为零)结束循环。相当于以下形式的 while 语句：

```
表达式 1;
while(表达式 2){
    循环体语句;
    表达式 3;
}
```

(2) 表达式 1 一般用于给循环变量或循环体内的某些变量赋初值(常采用逗号表达式形式)，如果循环变量或循环体内的某些变量已有确定的初值，则表达式 1 可以省略，但是其后的分号不能省略。

(3) 表达式 2 可以是关系表达式、逻辑表达式、也可以是数值和字符表达式，只要其值为非零，就执行循环体。

(4) 表达式 3 主要用于修改循环控制变量的值，以确保循环在某一时刻可以结束。如果循环体内已经包含控制循环次数的内容，则表达式 3 可以省略。

(5) 表达式 1、表达式 2 和表达式 3 也可以是与循环无关的量，三个表达式甚至都可以省略，(分号不是表达式的一部分，不可省略)。但若省略表达式 2，则循环体中需要有跳出循环的语句。

5. while 语句、do～while 语句和 for 语句的异同

一般情况下三种循环语句可以互相替换，可方便地将某种语句改写为另两种语句。唯一例外的是 do～while 语句，如一开始条件就不成立时，while 语句和 for 语句不执行循环

体，而 do～while 语句至少执行一次循环体，此时改写较为繁杂。三种语句中 for 语句最灵活，不仅循环控制变量的初始化可以放在表达式 1 中，而且循环控制变量的控制甚至整个循环体都可以放在表达式 3 中。

6. 循环嵌套

一个循环体内又包含另一个完整的循环结构，称为循环嵌套。内层循环中还可以继续嵌套循环，这就是多重循环。

(1) 三种循环(while，do～while，for)可以互相多重嵌套，还可以与分支选择语句(if，switch)构成互相多重嵌套。

(2) 循环嵌套时在语法结构上必须注意：

① 内外循环应全包含不交叉。

② 内外循环的控制变量不能同名。

7. return 语句

语句形式：

```
return(表达式);        或        return;
```

return 语句功能：

(1) 将程序控制(执行流程控制)返回到主调函数的调用处。

(2) 在有表达式时，将表达式的值带回到主调函数的调用处；无表达式时，调用处的值是不确定的。

8. break 中断语句

语句形式：

```
break;
```

(1) break 语句功能：

① 强迫终止循环的执行，使提前退出循环。

② 中断 switch 语句的执行。

(2) break 语句不能用于循环语句和 switch 语句之外的任何地方。

9. continue 语句

语句形式：

```
continue;
```

continue 语句功能：结束本次循环，即跳过循环体中 continue 语句下面尚未执行的语句，直接进行下一次是否继续循环的判断。continue 语句只能用于循环体中。

10. continue 语句与 break 语句的区别

(1) break 语句中断包含 break 语句的本层的整个循环语句的执行，即终止本层循环；或中断退出 switch 语句。

(2) continue 只是中断当前循环体的本次执行，而不是终止整个的循环；不能用于 switch 语句。

11. 标号语句与 goto 语句

标号语句的形式：

```
标号:语句
```

(1) C语句前可以加标号构成标号语句，以供转移引用。

(2) 标号是一个标识符，遵循标识符命名规则。

(3) 标号的作用域为标号所在函数，即只允许在定义该标号的函数内语句引用。

goto 语句的形式：

```
goto  语句标号;
```

goto 语句的作用：

(1) 实现无条件转移。执行到 goto 语句后，程序将转移到标号指定的语句继续执行(不能从循环体外转入体内)。需要注意的是：这种用法不符合结构化程序设计的要求，应该尽量避免。

(2) 可用于构成 goto 型循环。

二、实验操作

(一) 目的要求

(1) 掌握用 while 语句，do～while 语句和 for 语句实现循环的方法。

(2) 体会三种循环语句的异同，能在不同情况下正确选用循环语句。

(3) 掌握在程序设计中用循环的方法实现各种常用算法。

(4) 掌握选择结构与循环结构的嵌套。

(二) 范例

1. 编写程序实现加、减、乘、除四则运算。要求程序能读入数据，按算式计算出结果后输出。例如，通过键盘输入：

10.8+0.13*10=↙

程序将计算出结果 109.3。

源程序文件 prg4_1.c 清单：

```
#include <stdio.h>
main(){
  double x,y;
  char operator;
  sign1: printf("arithmetic expression\n");
  scanf("%lf",&x);
  while((operator=getchar())!='=') {
      sign2: scanf("%lf",&y);
      switch(operator) {
      case '+': x+=y; break;
      case '-': x-=y; break;
      case '*': x*=y; break;
      case '/': if(y!=0)
```

```
                    x/=y;
                else{
                    printf("divisor is zero");
                    setbuf(stdin,NULL);              // 清空缓冲区
                    goto sign2;
                }
                break;
        default: printf("illegal operator!");
                setbuf(stdin,NULL);                  // 清空缓冲区
                goto sign1;
        }
    }
    printf("%f",x);
}
```

注：此程序的运行与输入序列相关，当读入赋值运算符“=”时，循环结束。由于简单所以适用性不强，且不能处理运算符的优先级实现先乘除后加减的运算。

操作步骤如下：

(1) 利用正文编辑程序将 prg4_1.c 文件建立在用户自己的当前目录下。

(2) 发出编译命令：

$ cc prg4_1.c ↙

(3) 运行 a.out 执行文件：

$ a.out ↙

(4) 运行结果：

```
arithmetic expression
+10.8+0.13*10=↙
109.300000
```

2. 用牛顿迭代法求 $f(x)=x^3-2x^2+4x+1=0$ 在 x_0 附近的一个实根。

用牛顿迭代法求 $f(x)=0$ 在 x_0 附近的一个实根的方法是：

(1) 选一个接近于 x 的真实根的近似根 x_1。

(2) 通过 x_1 求出 $f(x_1)$。在几何上就是作 $x=x_1$，交 $f(x)$ 于 $f(x_1)$。见图 3.7。

(3) 过 $f(x_1)$ 作 $f(x)$ 的切线，交 x 轴于 x_2。可以用公式求出 x_2。由于

$$f'(x_1)=\frac{f(x_1)}{x_1-x_2}$$

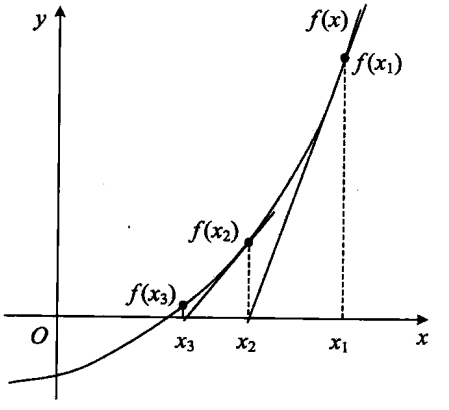


图 3.7 用牛顿迭代法示意图

故

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$

(4) 通过 x_2求出 $f(x_2)$。

(5) 再过 $f(x_2)$作 $f(x)$的切线交 x 轴于 x_3。

(6) 通过 x_3求出 $f(x_3)$，

……

一直求下去，直到接近真正的根，当两次求出的根之差 $|x_{n+1}-x_n|\leqslant\varepsilon$（$\varepsilon$ 一般可取 10^{-6}）就认为 x_{n+1}足够接近于真实根。

牛顿迭代公式是：

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

本例已知： $f(x)=x^3-2x^2+4x+1$

可求得： $f'(x)=3x^2-4x+4$

由此基本思路编程可得源程序文件 prg4_2.c 清单如下：

```
#include <stdio.h>
#include <math.h>
main(){
    double x,x1,f,f1;
    int n=1;
    scanf("%lf",&x);
    do{
        x1=x;
        f=pow(x1,3.0)-2.0*pow(x1,2.0)+4.0*x1+1;
        f1=3.0*pow(x1,2.0)-4.0*x1+4.0;
        x=x1-f/f1;
        printf("n=%4dx1=%-12.7fx=%-12.7f\n",n++,x1,x);
    }while(fabs(x-x1)>=1e-6);
    printf("root=%-12.7f\n",x);
}
```

操作步骤如下：

(1) 利用正文编辑程序将 prg4_2.c 文件建立在用户自己的当前目录下；

(2) 发出编译命令：

```
$cc prg4_2.c -lm↙
```

(3) 运行 a.out 可执行文件：

```
$a.out↙
```

(4) 运行记录：

```
1.0
```

```
n=   1x1=1.0000000     x=-0.3333333
n=   2x1=-0.3333333    x=-0.2287582
n=   3x1=-0.2287582    x=-0.2225152
n=   4x1=-0.2225152    x=-0.2224945
n=   5x1=-0.2224945    x=-0.2224945
root=-0.2224945
```

3. 用二分法求 $f(x)=x^3-6x-1=0$ 在 $x=2$ 附近的一个实根。

二分法的基本思路为：任取两点 x_1 和 x_2，判断 (x_1,x_2) 区间内有无一个实根。如图 3.8 所示，如果 $f(x_1)$、$f(x_2)$ 符号相反，说明 (x_1,x_2) 之间有一实根。取 (x_1,x_2) 的中点 x_0，检查 $f(x_0)$ 与 $f(x_1)$ 是否同符号，如果不同号，说明实根在 (x_0,x_1) 区间，这样就已经将寻根的范围减少了一半。然后用同样的办法再进一步缩小范围。再找 x_1 与 x_0 的中点，并且再舍弃其一半区间。如果 $f(x_0)$ 与 $f(x_1)$ 同号，则说明根在 (x_0,x_2) 区间，再取 x_0 与 x_2 的中点，并舍弃其一半区间。用这种方法不断缩小范围，直到区间相当小为止。

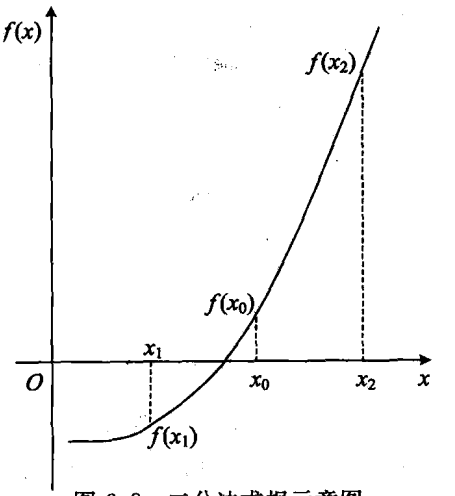


图 3.8 二分法求根示意图

源程序文件 prg4_3.c 清单：

```
#include "math.h"
main(){
  double x0,x1,x2,fx0,fx1,fx2;
  do{
  printf("enter x1,x2: ");
        scanf("%lf,%lf",&x1,&x2);
        fx1=pow(x1,3.0)-6*x1-1;
        fx2=pow(x2,3.0)-6*x2-1;
  }while(fx1*fx2>0);
  do{
      x0=(x1+x2)/2;
       fx0=pow(x0,3.0)-6*x0-1;
      if(fx0*fx1<0) {
          x2=x0;
          fx2=fx0;
      }
      else{
```

```
            x1=x0;
            fx1=fx0;
        }
    }while(fabs(fx0)>=1e-5);
    printf("root=%15.12lf\n",x0);
}
```

操作步骤如下：

(1) 利用正文编辑程序将 prg4_3.c 文件建立在用户自己的当前目录下。

(2) 发出编译命令：

```
$cc prg4_3.c ↙
```

(3) 运行 a.out 执行文件：

```
$a.out ↙
```

(4) 运行结果：

```
enter x1,x2: 1.0,4.0
root= 2.528917789459
```

4. 编一程序，读入形为 xxx…x〈a〉b 的字符串，其中 xxx…x 是 a 进制的数字串，代表一个 a 进制的整数。然后将此整数转换成 b 进制的整数，再输出此整数相应的数字串。a，b 为 2 到 10 之间的数字。如字符串“1765〈8〉2”，表示将 8 进制的数字串 1765 转换成 2 进制的数字串后输出。

根据题意，程序应执行以下工作：

```
{
    读入 a 进制的数字串到数组 digits;
    跳过字符 '<';
    读入数值 a 并转换成相应的整数值;
    跳过字符 '>';
    读入数值 b 并转换成相应的整数值;
    将 digits 中的 a 进制数字串转换成 10 进制整数 c;
    将 c 转换成 b 进制整数并将此整数转换成相应的数字串;
    输出转换后的数字串;
}
```

由上述程序流程可知，为把一个 a 进制的数字串转换 b 进制的数字串，首先将 a 进制数字串转换成十进制整数，然后将十进制整数转换成 b 进制整数，再转换成相应的数串。具体的转换方法如下：

设 a 进制整数为 $d_n d_{n-1} \cdots d_1 d_0$，则此 a 进制整数的十进制整数值为：

$$
\begin{aligned}
c &= d_0 + d_1 a_1 + \cdots d_{n-1} a^{n-1} + d_n a^n \\
&= ((\cdots(d_n a + d_{n-1}) a + \cdots + d_1) a + d_0
\end{aligned}
$$

将一个十进制整数转换成 b 进制整数的方法是除 b 取余法。下面用实例说明此方法的计算过程。设十进制数 c=3891，要转换成七进制整数可以按如下方法进行：

```
7 | 3891
  7 | 555      ……6
    7 | 79     ……2
      7 | 11   ……2
        7 | 1  ……4
            0  ……1
```

的 $3891_{10}=14226_7$。

按前述程序流程编制的源程序文件 prg4_4.c 清单：

```
#include <stdio.h>
#define MAXLEN 16
main(){
    int inform,outform,c,i,next;
    char digits[MAXLEN],a,b,ch;
    next=0;
    while((ch=getchar())<='9'&&ch>='0')
        digits[next++]=ch;          //读入 a 进制的数字串，过滤掉“<”
    a=getchar();
    if(a<='9'&&a>='2')
        inform=a-'0';
    else
        inform=10;
    getchar();          //滤掉“>”处理
    b=getchar();
    outform=(b<='9'&&b>='2')? (b-'0'):10;      //若 b 不在 '2'～'9'，
                                                 则作十进制处理
    for(c=i=0; i<=next-2; i++)
        c=(c+digits[i]-'0') * inform;
    c=c+digits[i]-'0';
    next=0;
    do
        digits[next++]=c%outform+'0';
    while((c/=outform)>0);
    for(i=next-1; i>=0; i--)
        printf("%c",digits[i]);
}
```

操作步骤如下：

(1) 利用正文编辑程序将 prg4_3.c 文件建立在用户自己的当前目录下。

(2) 发出编译命令：

```
$cc prg4_3.c ↙
```

(3) 运行 a.out 执行文件：

```
$a.out ↙
```

(4) 运行结果之一：

```
1765<8>2 ↙
1111110101
```

运行结果之二：

```
3891<a>7 ↙
14226
```

5. 游泳池入水点问题。

如图 3.9 所示，已知某游泳池的长度为 50 米，宽度为 25 米。某人游泳速度为 0.8 米/秒，步行速度为 1.2 米/秒。现要从 A 点运动(步行和游泳)到 C 点，编程求解从 A 点到 B 点间何处下水游到 C 点时间最短，输出该点距 A 点的长度 x 和所求最短时间 y。要求：

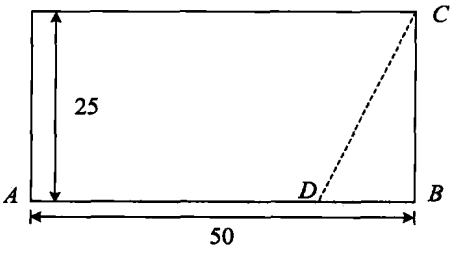


图 3.9 游泳池入水点问题示意图

(1) 给出求解该问题的算法描述。

(2) 根据算法编写完整的求解程序。

说明：距离的长度精度控制在 0.1 米或以下。

解 1 (1) 求解问题的算法描述：本题采用枚举法(亦称穷举法)求解。

分析：

① 假设在 AB 边上行进 x_1 米后的 D_1 点下水，历时 $t_1=x_1/1.2$。

② 从 D 点下水到游到 C 点用时为：

$t_2=$ sqrt(WIDTH * WIDTH+(LENGTH−x1) * (LENGTH−x1))/0.8

式中，WIDTH、LENGTH 分别为游泳池的宽度和长度，sqrt 为求平方根函数。

③ 则总用时(从 A 点走到 D 点下水后游到 C 点)为 $t=t_1+t_2$。

根据以上分析，显然，最短用时走法如存在的话，x_1 肯定在 0 到 50 之间。而根据题目要求，距离精度控制在 0.1 米以下即可，故 x_1 的取值可规定为 500 种，即可取 0，0.1，0.2，…，49.9，50.0。所以可以使用枚举法，一一计算出这 500 种可能走法的用时，取其最小值即为题目所求。

设用 time 存储用时最小值，用 dx 存储 D 点到 A 点的距离，枚举算法可描述如下：

```
预设 x=0.0,time =32768,dx=0
while(x<=50.0){
    temp=x/1.2+sqrt(WIDTH * WIDTH+(LENGTH−x) * (LENGTH−x))/0.8;
    if(若所求时间小于等于前次所求最短时间){
      更新保存最短时间和相应距离两变量的值;
    }
    x=x+0.1;          //步进长精度为 0.1
```

```
}
```

(2) 按前述程序流程编制的源程序文件 prg4_4.c 清单如下：

```
//枚举法求解示例 1,源程序文件 prg4_5.c 清单：
#include<stdio.h>
#include<math.h>
#define LENGTH   50.0
#define WIDTH 25.0
#define V1   0.8
#define V2   1.2
void main(){
    float x=0.0,time,temp,dx=0;
    time =32768;
    while(x<=50.0){
        temp=x/V2+sqrt(WIDTH * WIDTH+(LENGTH-x) * (LENGTH
            -x))/V1;
        if(temp<=time){
            time=temp;
            dx=x;
        }
        x=x+0.1;
    }
    printf("time sec=%f, x=%f\n",time,dx);
    printf("(50+25)/1.2 =%f\n",(LENGTH+WIDTH)/V2);
}
```

运行结果：

```
time min=64.959061, x=27.600069
(50+25)/1.2 =62.500000
```

以上程序使用的是 while 语句，也可使用 for 语句实现，程序如下：

```
//枚举法求解示例 2
#include<stdio.h>
#include<math.h>
double jisuan(double n){
    double t;
    t=n/1.2+sqrt(25 * 25+(50-n) * (50-n))/0.8;
    return(t);
}
void main(){
    double x,tmin,i;
```

```
        for(i=0.02;i<=50.0;i=i+0.01)
            if(jisuan(i)>jisuan(i+0.01)){
                tmin=jisuan(i);
                x=i;
            }
        printf("距 A 点%f, 最短时间%f.\n",x, tmin);
    }
```

运行结果：

距 A 点 27.620000，最短时间 64.959045.

解 2 (1) 算法描述：通过一元二次方程求解。

分析：假设在 D 点下水，AD 间距离为 x 米；则从 A 到 C 的时间函数为：

$$f(x)=\frac{x}{1.2}+\frac{\sqrt{(50-x)^2+25^2}}{0.8}$$

对 $f(x)$ 求导，得导函数：

$$f'(x)=\frac{1}{1.2}+\frac{1}{0.8}\times\frac{1}{2}\times\frac{1}{\sqrt{(50-x)^2+25^2}}\times 2(x-50)$$

要使时间最短，则 $f'(x)=0$，即

$$\frac{1}{1.2}+\frac{x-50}{0.8\sqrt{(50-x)^2+25^2}}=0$$

化简得：

$$x^2-100x+2000=0$$

则此问题化简为求 $x^2-100x+2000=0$ 在 0 到 50 之间的根。

$ax^2+bx+c=0$ 的求根公式为 $x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$，求出 x_1,x_2 后判断是否满足题意。

(2) 根据算法编写完整的求解程序如下：

```
#include<stdio.h>
#include<math.h>
void main(){      //直接求 x*x-100x+2000=0 在(0,50)之间的根解 2
    double a,b,c,disc,x1,x2,t1,t2;
    a=1; b=-100; c=2000;
    disc=b*b-4*a*c;
    x1=(-b+sqrt(disc))/(2*a);
    x2=(-b-sqrt(disc))/(2*a);
    t1=x1/1.2+sqrt((50-x1)*(50-x1)+25*25)/0.8;
    t2=x2/1.2+sqrt((50-x2)*(50-x2)+25*25)/0.8;
    if(x1>=0 && x1<=50)
        printf("x1=%f   t1=%f\n",x1,t1);
    else
        printf("x2=%f   t2=%f\n",x2,t2);
```

```
}
```

运行结果：

```
x2=27.639320   t2=64.959041
```

解 3 (1) 算法描述：通过一元二次方程求解。

分析：同解 2，将问题化简为求 $x^2-100x+2000=0$ 在 0 到 50 之间的根，然后用牛顿迭代法、二分法等方法求可解方程。

(2) 根据算法编写完整的求解程序如下：

用牛顿迭代法求解方程源程序：

```
#include<stdio.h>
#include<math.h>
void main(){
    double Ad,Bd,BC=25.0,AB=50.0,x,x1,t,ft1,ft2;
    scanf("%lf",&x);
    do{
        x1=x;
        ft1=x1*x1-100*x1+2000;
        ft2=2*x-100;
        x=x1-ft1/ft2;
        printf("time min=%f\n",x);
    }while(fabs(x-x1)>=1e-6);
    t=x/1.2+sqrt((50.0-x)*(50.0-x)+25.0*25.0)/0.8;
    printf("!! time min=%f,  d=%f\n",t,x);
    printf("(50+25)/1.2 =%f\n",(50+25)/1.2);
}
```

运行结果：

```
25
time min=27.500000
time min=27.638889
time min=27.639320
time min=27.639320
!! time min=64.959041,  d=27.639320
(50+25)/1.2 =62.500000
```

用二分法求解方程源程序：

```
#include<stdio.h>
#include<math.h>
double fx(double x){
```

```
        double fxy;
        fxy=x*x-100*x+2000;
        return(fxy);
    }
    void main(){
        double x0,x1,x2,fx0,fx1,fx2,t;
        do{
            printf("enter x1,x2: ");
            scanf("%lf%lf",&x1,&x2);
            fx1=fx(x1);
            fx2=fx(x2);
        }while(fx1*fx2>0);
        do{
            x0=(x1+x2)/2;
            fx0=fx(x0);
            if(fx0*fx1<0){
                x2=x0;
                fx2=fx0;
            }
            else{
                x1=x0;
                fx1=fx0;
            }
        }while(fabs(fx0)>=1e-6);
        t=x0/1.2+sqrt((50.0-x0)*(50.0-x0)+25.0*25.0)/0.8;
        printf("距 A 点%f 米处下水,最短时间为%f 秒.\n",x0,t);
    }
```

运行结果：

```
enter x1,x2: 0 50
```

距 *A* 点 27.639320 米处下水，最短时间为 64.959041 秒。

6. 某地刑侦大队对涉及 6 个嫌疑人的一桩疑案进行分析，已知：

(1) A,B 至少有一人作案。

(2) A,E,F 3 人中至少有 2 人参与作案。

(3) A,D 不可能是同案犯。

(4) B,C 或同时作案，或与本案无关。

(5) C,D 中有且仅有一人作案。

(6) 如果 D 没有参与作案，则 E 也不可能参与作案。

根据以上条件，分析哪些人是罪犯。

为解这道题，要结合逻辑运算符与逻辑表达式，将案情的每一条件写成逻辑表达式，从第1条到第6条依次用cc1，…，cc6表示。

(1) A，B至少有一人作案。

可用变量A代表嫌疑人A，变量A=1表示嫌疑人A作案，变量A=0表示嫌疑人A不作案。与此类似，变量B=1表示嫌疑人B作案，变量B=0表示嫌疑人B不作案……如此规定之后，条件(1)可用真值表表示成表3.3。

表3.3 条件(1)真值表

A	B	cc1
0	0	0
1	0	1
0	1	1
1	1	1

显然这是或的关系，因此有

$$cc1=(A \parallel B)$$

(2) A，E，F 3人中至少有2人参与作案。

有3种可能：

第一种，A和E作案，写成“A && E”。

第二种，A和F作案，写成“A && F”。

第三种，E和F作案，写成“E && F”。

这3种可能是或的关系，可用真值表表示成表3.4。

表3.4 条件(2)真值表

A	E	F	cc2
1	1	0	1
1	0	1	1
0	1	1	1
0	0	1	0
0	1	0	0
1	0	0	0
0	0	0	0
1	1	1	1

因此有

$$cc2=(A \&\& E) \parallel (A \&\& F) \parallel (E \&\& F)$$

(3) A，D不可能是同案犯。

即A如果是案犯，则D一定不是案犯；D如果是案犯，则A一定不是案犯。这两者是逻

辑与的关系，可以用真值表表示成表 3.5。

表 3.5 条件(3)真值表

A	D	A && D	cc3
1	0	0	1
0	1	0	1
0	0	0	1
1	1	1	0

因此有

cc3=!(A && D)

(4) B,C 或同时作案，或同时与本案无关。

有两种可能：

第一种情况，同时作案写成“B && C”。

第二种情况，都与本案无关写成“! B && ! C”。

两种可能是或的关系，可用真值表表示成表 3.6。

表 3.6 条件(4)真值表

B	C	! B	! C	B && C	! B && ! C	cc4
1	1	0	0	1	0	1
1	0	0	1	0	0	0
0	1	1	0	0	0	0
0	0	1	1	0	1	1

因此有

cc4=(B && C) || (! B && ! C)

(5) C,D 中有且仅有一人作案。可以用真值表表示成表 3.7。

表 3.7 条件(5)真值表

C	D	! C	! D	C && ! D	D && ! C	cc5
1	0	0	1	1	0	1
0	1	1	0	0	1	1
0	0	1	1	0	0	0
1	1	0	0	0	0	0

因此有

cc5= (C && ! D) || (D && ! C)

(6)如果 D 没有参与作案，则 E 也不可能参与作案。

分析语义为：如果 D 不作案，则 E 也不可能作案；即如果 D 不作案，E 作案是不可能的；若 D 作案，则 E 也可能作案也可能不作案。可以用真值表表示成表 3.8。

表 3.8 条件(6)真值表

D	E	! E	cc6
1	1	0	1
1	0	1	1
0	0	1	1
0	1	0	0

因此有

cc6=(D ‖ !E)。

通过以上的案情分析,已知条件化成了计算机可解的逻辑表达式。

源程序文件 prg4_6 如下:

```
#include<stdio.h>
void judge(int x){
    if(x==0)
        printf("不是罪犯\n");
    else
        printf("是罪犯\n");
}
void main(){
  int cc1,cc2,cc3,cc4,cc5,cc6;      //定义 6 个变量,分别表示 6 个已知条件
  int a,b,c,d,e,f;      //定义 6 个变量,分别表示 6 个人
  for(a=0;a<=1;a++)      //枚举 a 的两种可能,0 不是罪犯,1 是罪犯
    for(b=0;b<=1;b++)      //枚举 b 的两种可能
      for(c=0;c<=1;c++)      //枚举 c 的两种可能
        for(d=0;d<=1;d++)      //枚举 d 的两种可能
          for(e=0;e<=1;e++)      //枚举 e 的两种可能
            for(f=0;f<=1;f++){      //枚举 f 的两种可能
              cc1=a‖b;      //a 和 b 至少有 1 个人作案
              cc2=!(a&&d);      //a 和 d 不可能是同案犯
              cc3=(a&&e)‖(a&&f)‖(e&&f);
                                        //a,e,f 中至少有 2 个人涉嫌作案
              cc4=(b&&c)‖(!b&&!c);
                                        //b 和 c 或同时作案,或都与本案无关
              cc5=(c&&!d)‖(d&&!c);      //c,d 中有且仅有一人作案
              cc6=d‖!e;      //如果 d 没有参与作案,则 e 也不可能参与作案
              if(cc1&&cc2&&cc3&&cc4&&cc5&&cc6){   //输出判断结果
                printf("A:");judge(a);
                printf("B:");judge(b);
```

```
                    printf("C:");judge(c);
                    printf("D:");judge(d);
                    printf("E:");judge(e);
                    printf("F:");judge(f);
                }
            }
    }
```

运行结果：

A:是罪犯
B:是罪犯
C:是罪犯
D:不是罪犯
E:不是罪犯
F:是罪犯

（三）实验内容

1. 输入两个正整数 m 和 n，求其中最大公约数和最小公倍数。

常用算法提示如下：

求最大公约数：

(1) 求 m 除 n 的余数 r。

(2) 若 r 等于 0，则 n 为最大公约数。

(3) 否则令 $m=n, n=r$，回到第一步。

求最小公倍数：最小公倍数 $= m * n /$ 最大公约数。

2. 从键盘输入 50 个学生的学号和 3 门课成绩，计算每个学生的平均成绩并输出其学号和平均成绩，同时找出其中平均成绩最高和最低的学生并输出其学号和平均成绩。

3. 写一个程序，统计输入正文中的英文字母、空格、制表符、换行符和其他字符的个数。

提示：输入正文是指从键盘输入的一个字符流。在 UNIX 系统下，字符流可按〈Ctrl〉+D 结束，微机系统一般可按〈Ctrl〉+Z 结束。建议采用 getchar()函数读入字符，并结合循环控制结构完成指定功能。

4. 编写程序，从键盘输入 x，利用幂级数展开计算 $\sin x$ 的近似值，要求误差小于 10^{-6}。

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots\cdots$$

提示：对于类似的数列求和问题，关键是找出数列中前后两项之间的关系，并用递推关系式表达出来。本题可设累加器变量 sum、中间变量 t、初值均为 x（第一项）。然后利用循环从第二项开始，不断求得某项的值 t 并与 sum 累加，直到某项绝对值小于 10^{-6} 为止。另外，需要注意奇偶项符号的处理。可在程序中加入标准数学库函数调用 $\sin(x)$，将通过自定义函数与库函数调用两种方法的结果进行验证比照。

5. 按下面给出的求 π 近似值的公式，计算并输出 π 的值，设 $n=1000$。

$$\pi = 2 \times \frac{2^2}{1 \times 3} \times \frac{4^2}{3 \times 5} \times \cdots \times \frac{(2 \times n)^2}{(2n-1)(2n+1)}$$

6. 求 $\sum_{n=1}^{20} n!$（即求 $1! + 2! + 3! + 4! + \cdots + 20!$之和）。

提示:类型不同其取值范围也不同,本题应注意变量类型的选择。

7. 用牛顿迭代法求方程 $2x^3 - 4x^2 + 3x - 6 = 0$ 在 1.5 附近的值。

8. 用二分法求方程 $2x^3 - 4x^2 + 3x - 6 = 0$ 在(−10,10)区间内的根。

9. 输出 500 以内最大的 10 个素数并计算它们的和。

(1) 输出要有文字说明,并同时输出各素数。输出形式为:

素数 1+素数 2+…+素数 10=和值

(2) 虽然在 500 以内的素数个数超过 10 个,但是仍要求考虑 500 以内不够 10 个素数的情况。

(3) 由于偶数肯定不是素数,因此在程序中不必对偶数进行判断。

10. 编程打印出以下图案。

```
      *
    * * *
  * * * * *
* * * * * * *
  * * * * *
    * * *
      *
```

提示:此图形规则变化有序,编程前应先分析图形的变化规律,然后根据其规律采用循环语句控制打印输出该图形。

11. (选做题)编程打印如下格式的 9 * 9 乘法表的下三角部分。

	1	2	3	4	5	6	7	8	9
1	1								
2	2	4							
3	3	6	9						
4	4	8	12	16					
5	5	10	15	20	25				
6	6	12	18	24	30	36			
7	7	14	21	28	35	42	49		
8	8	16	24	32	40	48	56	64	
9	9	18	27	36	45	54	63	72	81

12. (选做题)求 $s = 1 - \frac{1}{2} + \frac{1}{3} - \cdots + \frac{1}{99} - \frac{1}{100}$。

13. (选做题)写一个程序,将输入复制到输出,当输入正文中有连续一个以上的空格时,在输出中用一个空格代替。

14. (选做题)编写简单的四则运算程序,输入一个算式,格式为:a op b ,其中 a, b 是参加运算的两个数,op 为运算符 +、−、*、/ 之一。例如输入:12+5,则输出:12+5=17;输

入:15/4,则输出:15/4=3.75。

要求:

(1)对非法的 op,除法运算时 b 为 0 的情况做出出错处理。

(2)除法运算结果精确到小数后两位。

(3)分别用 if～else if 语句和 switch 语句两种方法实现。

15.(选做题)编写程序,验证哥德巴赫猜想。哥德巴赫提出:一个不小于 6 的偶数必定能表示为两个素数之和。例如:6=3+3,8=3+5,10=3+7,……

要求将 6～100 之间的全部偶数表示为两个素数之和。输出形式如下:

6=3 + 3
8=3 + 5
10=3 + 7
10=5 + 5
12=5 + 7
……

16.(选做题)三色球问题。若一个口袋中放有 12 个球,其中有 3 个红的,3 个白的和 6 个黑的,问从中任取 8 个共有多少种颜色的搭配?

(四)常见错误分析

循环结构程序设计常易犯以下错误:

(1) 循环控制中最常出现的问题之一是循环控制变量没有正确地初始化。while 和 do～while 的循环控制变量初始化一般是在循环控制语句之前完成的,而 for 语句的循环控制变量的初始化通常是在表达式 1 中实现的。

(2) for 和 while 循环是先判断循环条件,后执行循环体;而 do～while 循环则是先执行循环体,然后才判断循环条件。因此,do～while 循环不管在什么情况下,都至少要执行一次循环体。

(3) do～while()后面必须有一个“;”,而 while 循环在 while()后面没有分号,初学者往往会忽略这一点。

(4) 在语法形式上,三种循环的循环体是一个语句单位,即一个语句。若循环体多于一个语句时,必须用花括号括起来,以复合语句形式出现。在 do～while()循环中,不管循环体是否为单一语句,习惯上都用花括号把它括起来,并把“while(表达式);”直接写在“}”的后面,以免把“while(表达式);”部分误认为是一个新 while 循环的开始。

【错误示例 4.1】 使用 while 循环求两个正整数的最小公倍数。

源程序中存在在 while()后面加“;”的错误,程序在运行时将进入死循环状态。读者自行分析改正。

```
#include <stdio.h>
main(){
    int m,n,result;
    printf("input two integers:");
    scanf("%d%d",&m,&n);
```

```
    result=m<n? n:m;
    while(!(result%m==0&&result%n==0));
        result++;
    printf("the lease common multiple of %d and %d is %d\n",m,n,result);
}
```

【错误示例 4.2】 使用 do～while 循环求两个正整数的最小公倍数。

源程序中存在相应循环控制变量没有正确的初始化的错误，该程序在运行时在某些情况下得不到正确的结果。请读者自行分析为什么。

```
#include <stdio.h>
main(){
    int m,n,result;
    printf("input two integers:");
    scanf("%d%d",&m,&n);
    result=m<n? n:m;
    do{                                  //存在循环控制变量没有正确的初始化的错误
        result++;
    }while(!(result%m==0&&result%n==0));
    printf("the lease common multiple of %d and %d is %d\n",m,n,result);
}
```

正确的源程序如下：

```
#include <stdio.h>
main(){
    int m,n,result;
    printf("input two integers:");
    scanf("%d%d",&m,&n);
    result=m<n? n:m;
    result--;                            //根据本程序的求解算法，本执行语句是必需的
    do{
        result++;
    }while(!(result%m==0&&result%n==0));
    printf("the lease common multiple of %d and %d is %d\n",m,n,result);
}
```

【错误示例 4.3】 使用 for 循环求两个正整数的最小公倍数。

存在当循环体内不需要具有特定功能的语句，而编写者没有正确使用空语句，即没有在 for()后加“;”；而误将 for 循环的后续语句“printf……”作为循环体。以至不能得出正确的运行结果。

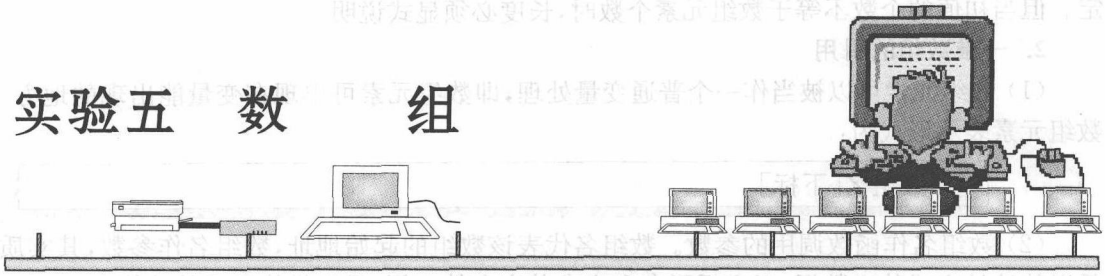
```
#include <stdio.h>
```

```
main(){
  int m,n,result;
  printf("input two integers:");
  scanf("%d%d",&m,&n);
  for(result=m<n? n:m; !(result%m==0&&result%n==0);result++)
  printf("the lease common multiple of %d and %d is %d\n",m,n,result);
}
```

正确的源程序如下所示：

```
#include <stdio.h>
main(){
  int m,n,result;
  printf("input two integers:");
  scanf("%d%d",&m,&n);
  for(result=m<n? n:m; !(result%m ==0&&result%n==0);result++)
    ;                                    //正确使用空语句
  printf("the lease common multiple of %d and %d is %d\n",m,n,result);
}
```

实验五　数　　组



学习重点

- 一维数组的定义、引用、初始化
- 二维数组的定义、引用、初始化
- 数组的存储结构形式
- 数组元素和数组名作函数参数
- 数组名作函数参数时，形实结合的传地址方式
- 以数组为基础的算法应用
- 字符数组应用与编程

一、要点综述

数组是同类型数据元素的有序集合，即数组由基本类型（整形、实型、字符、指针等数据类型）组合而成。这种组合可以是多层的，但最低一级的成员必须是基本类型的元素构成，即数组元素必须是基本类型的量。

1. 一维数组的定义

数组必须先定义后使用。一维数组一般定义形式为：

```
类型表识符　数组名[常量表达式],……;
```

(1) [ ]：为数组下标运算符。在此表示定义一个指定类型和名称的数组，并为其开辟由常量表达式确定大小的连续存储空间。

(2) 数组名：代表该数组存储空间的起始地址，取名规则同变量。

(3) 常量表达式：其值规定所定义数组的长度（大小）。定义数组时必须通过这个常量指定数组的大小，不能包含变量，且必须是正的整型常量表达式。

(4) 数组的下标：从 0 开始，最后一个元素的下标为“数组长度－1”。

(5) 一维数组在两种情况（场合）下，可以省略数组长度的说明，而用[ ]替代。

① 一维数组名作为被调函数的参数，其存储空间和大小由主调函数的实参数组传递确定。

② 对一维数组赋初值意味着对全体数组元素赋初值，此时数组的长度可由初值个数确

定。但当初值的个数不等于数组元素个数时，长度必须显式说明。

2. 一维数组的引用

(1) 数组元素可以被当作一个普通变量处理，即数组元素可出现在变量能出现的地方。数组元素表示形式为：

```
数组名[下标]
```

(2) 数组名作函数调用的参数。数组名代表该数组的起始地址，数组名作参数，其实质是以传地址方式传递数据，以实现形参和实参共享存储空间，具有带值返回的功能。

(3) C 语言不检查数组的边界，当超越数组边界时，系统编译时不会提示错误，但运行时将导致运算结果错误或运行错误，甚至可能导致其他变量或程序被破坏。

3. 一维数组的初始化

一般形式：

```
类型标识符   数组名[常量表达式]={元素值表列}
```

(1) 元素值表列可以是数组所有元素的初值，也可以是前面部分元素的值。对数值型数组，如初始化时未明确设定元素的初值，则其值自动被设置为 0。

(2) 当对全体数组元素赋初值时，元素个数可以省略，但“[ ]”不能省略。此时系统将根据数组初始化时大括号内值的个数，决定该数组的元素的个数。

(3) 数组初始化的赋值方式只能用于数组的定义中，定义之后再赋值就只能一个元素一个元素地赋值。

4. 多维数组

多维数组的定义形式：

```
类型标识符 数组名[常量表达式 1][常量表达式 2][常量表达式 3]……;
```

(1) 多维数组的数组名、每维元素个数的要求同一维数组。

(2) 多维数组中，元素是按行存放的，即按顺序先存放第一行的数组元素，再存放第二行的数组元素，……以此类推。

(3) 由多维数组按行存储方式，可以把二维数组理解为由低一维数组复合而成，即构造而成。如二维数组 a[3][4]数组可看成是由 a[0]、a[1]、a[2]三个元素组成，而每一个元素又是一个一维数组，因此，可以把 a[0]、a[1]、a[2]看成是一维数组名，各代表一维数组的起始地址。以此类推，对多维数组的元素如以三维数组 int a[3][4][5]的元素 a[i][j][k]为例，可按如下方式理解：

层 行 列

a[i][j][k] ⟶ 为元素，代表值

a[i][j] ⟶ 一维数组名，代表一维数组的起始地址

a[i] ⟶ 二维数组名，代表二维数组的起始地址

a ⟶ 三维数组名，代表三维数组的起始地址

5. 多维数组的引用

(1) 多维数组元素同样可被当作一个普通变量处理，可以出现在任何同类型变量能出现的地方。多维数组元素表示形式为：

```
数组名[下标1][下标2][下标3]……;
```

(2) 数组的维数不同,引用数组元素的下标也就不同。

(3) 同一维数组一样,多维数组的下标可以是整型常量、整型变量或者整型表达式。每一维下标都从0开始,如二维数组a[3][4]的元素排列顺序为a[0][0],a[0][1],…,a[2][3]。

6. 多维数组的初始化

(1) 可按行对多维数组初始化。例如:

```
static int a[2][3]={{1,2,3},{4,5,6}};
```

(2) 可按照数组的存储顺序赋初值。例如:

```
static int a[2][3]={1,2,3,4,5,6};
```

(3) 可只对部分元素赋值。对数值型数组,如果进行初始化时未明确设定元素的初值,则数组元素值自动被设置为0。例如:

```
static int a[2][3]={{1,2},{4}};
```

则a[0][2]、a[1][1]、a[1][2]的值为0。

(4) 如果对多维数组全部赋初值,则最左边一维的元素个数可以省略,但其余各维必须明确指定。例如:

```
static int a[ ][3]={{1,2,3},{4,5,6}};
```

(5) 同一维数组一样,多维数组初始化的赋值方式只能用于数组的定义,定义之后再赋值就只能一个元素一个元素地赋值。

7. 数组元素作函数参数

数组元素可以作为函数的实参,此时与同类型的简单变量作为实参一样,属于单向的值传递,数组元素的值传递给形参后,形参的改变不影响作为数组元素的实参。

8. 数组名做函数参数

数组名作为函数参数,此时实参和形参都应是数组名,或者是表示地址的指针变量;数据的传递方式属于传地址方式,传递的是整个数组。

(1) 数组名表示的是数组的起始地址,数组名作函数的实参时,传递的是数组的首地址,其实质是以传地址方式传递数据,实现形参和实参共享存储空间,所以具有带值返回的功能,即形参数组元素的值改变时,实参数组的元素也同时被改变了。

(2) 形参中的数组不但要定义,而且要求与实参数组的类型一致。

(3) 多维数组作为函数参数在参数说明时与一维数组不同,一维数组可不指明长度,而多维数组除最左边一维可以不指定长度外,其他各维均要指明长度。

9. 字符数组

在C语言中,字符串可通过字符数组表示;字符串的处理是基于字符数组的,即具体操作通过字符数组元素的运算实现。字符数组的定义形式为:

```
char 数组名[常量表达式];                    //定义一维字符数组
char 数组名[常量表达式1][常量表达式2]……;    //定义多维字符数组
```

(1) 字符串常量是由双引号括起来的字符序列,在实际存储时系统自动在其尾部添加了一个字符串结束标志"\0",所以采用字符数组存放字符串时,应该包含字符串结束标志"\0"。

(2) 字符数组的元素个数应该大于或等于字符串长度+1,增加的一个字节用于存放字符串结束标志"\0"。

10. 字符数组的初始化和赋值

字符数组的初始化和赋值操作时要特别注意字符串结束标志"\0"的处理。

(1) 采用将字符逐个赋给数组中各个元素的初始化形式时,串尾应包含字符串结束标志"\0"。例如:

static char s[9]={'z','h','a','n','g','s','a','n','\0'}。

(2) 直接用字符串常量给字符数组初始化时,系统自动在串尾加字符串结束标志"\0"。例如:

static char s[9]="zhangsan"。

(3) 在单个字符一个一个地赋值时,串尾应包含字符串结束标志"\0"。例如:

char s[4]; s[0]='O'; s[1]='K'; s[2]='! '; s[3]='\0';

(4) 除初始化形式外,不能采用"字符数组=字符串常量"的赋值形式。

(5) 二维字符数组可以看作是这样一个一维数组,它的每个元素是一个字符串。

(6) 二维字符数组的元素和数组名都可以作为函数参数,并且使用方法与二维数组的使用方法相同。

11. 字符数组的输入输出

设有:

```
char name[20];
```

(1) 可用格式符"%c"输入输出某个元素。例如:

```
scanf("%c", &name[i]);          //其中 i 为数组范围内的合法下标
printf("%c", name[i]);          //其中 i 为数组范围内的合法下标
```

(2) 用字符输入函数 getchar()一个字符一个字符地读入赋给数组元素,最后用赋值语句补上字符串结束标志"\0"。

(3) 用字符输出函数 putchar(),可一个字符一个字符地输出。

(4) 可用"%s"输入输出一个字符串。例如,用"%s"输入字符串:

```
scanf("%s", name);
```

注意:

① 用"%s"输入串时,遇空格结束,因此输入串的中间不能出现空格,否则将作为串输入结束处理。

② 数组名代表该数组存储单元的起始地址,故作为输入参数不用"&"取地址符。

③ 系统自动在串尾加"\0"结束符。

④ 正确区分字符串长度与数组长度这两个不同的概念。例如,用"%s"输出字符串:

```
printf("%s", name);
```

注意:

ⅰ. 与"%s"对应的输出参数要求为字符数组名(地址常量)或字符串的起始地址,系统从第一个字符起依次输出串内容,直至遇"\0"串结束符结束。

ⅱ. 与"%s"对应的输出参数也可以是串中的某个字符的地址,还可以是指向字符串任意位置的指针变量,系统从指定位置起依次输出串内容,直至遇"\0"串结束符结束。

(5) 使用 gets()和 puts()输入输出字符串。例如：

```
gets(name); puts(name);
```

注意:gets()函数遇回车键结束串的输入，并自动在串尾加"\0"结束符，因此输入串中可包含空格和〈Tab〉在内的全部字符，用于字符串的输入比较方便。

(6) 可调用复制字符串函数 strcpy()对字符串整体赋值。例如：

```
char s[9];
strcpy( s,"zhangsan" );
```

12. 常用字符串操作库函数与编程

C 语言提供了很多字符串操作函数，其对应的头文件为"string. h"，用到这些函数时，需要在程序的开头添加预处理命令 #include 〈string. h〉。

常用的字符串操作函数有：

(1) strlen(字符串)：返回字符串的长度。(**注意**:这个长度不包含字符串结束符"\0")。

(2) strcpy(字符数组 1，字符串 2)：将字符串 2(包括字符串结束符"\0")复制到字符数组 1 中。**注意**:字符数组 1 必须足够的大，以能容纳字符串 2 的内容。

(3) strcat(字符数组 1，字符串 2)：将字符串 2 的内容复制连接到字符数组 1 中串的尾部，并返回字符数组 1 的地址。其中字符数组 1 必须足够大，以能容纳字符串 2 的内容。**注意**:连接后字符数组 1 中原串的"\0"被覆盖，连接后的新串串尾保留一个"\0"。

(4) strcmp(字符串 1，字符串 2)：比较字符串 1 和字符串 2 的大小。比较过程为：两个字符串从左至右逐个字符比较(按照字符的 ASCII 码值的大小)，直到字符不同或者遇见"\0"为止。如果全部字符都对应相同且串长相等，则返回值为 0。如果不相同，则返回两个字符串中第一个不相同字符的 ASCII 码值的差值，即字符串 1 大于字符串 2 时函数值为正，否则为负。

以上函数参数中出现的"字符串"可以是字符串常量、存放字符串的字符数组名(要求串尾包含字符串结束符"\0")，或者要处理的字符串的第一个字符的地址(字符型指针)；同样"字符数组"可以是字符数组名或者要处理的字符串的第一个字符的地址(字符型指针)。

13. 以数组为基础的常用算法

以数组为基础可以构建很多实用算法，常用的有：

(1) 冒泡法、选择法和 Sheel(希尔)法等排序算法。

(2) 筛选法求 1 至指定数之间的素数。

(3) 折半查找(又称二分查找)。

(4) 字符串连接、拷贝和比较等字符处理。

(5) 矩阵的相加和相乘等操作。

二、实验操作

(一) 目的要求

(1) 掌握一维数组和二维数组的定义、赋值和输入输出的方法。

(2) 掌握字符数组和字符串函数的使用和编写。

(3) 掌握与数组有关的算法(如查找、排序算法等)。

(4) 练习并掌握数组作为函数参数的用法。

(二) 范例

1. 输入并运行 prg5_1.c,分析运行结果,验证数组的存储结构,利用循环访问数组元素,用格式符"%s"输出字符数组等。

程序清单 prg5_1.c 如下:

```
#include <stdio.h>
main(){
    static int a[6]={10,20,30,40,50},i;
    static char b[6]="USTC";
    printf("\na: ");
    for(i=0; i<6; i++)
        printf("(%x)%-4d ",&a[i],a[i]);
    printf("\nb: ");
    for(i=0; i<6; i++)
        printf("(%x)%-4c ",&b[i],b[i]);
    printf("\nb: ");
    for(i=0; i<6; i++)
        printf("(%x)%-4d ",&b[i],b[i]);
    printf("\nString: %s\n",b);
}
```

运行结果:

```
a: (422a30)10  (422a34)20  (422a38)30  (422a3c)40  (422a40)50  (422a44)0
b: (422a48)U   (422a49)S   (422a4a)T   (422a4b)C   (422a4c)    (422a4d)
b: (422a48)85  (422a49)83  (422a4a)84  (422a4b)67  (422a4c)0   (422a4d)0
String:USTC
```

分析结果:

(1) 运行结果中,括号内是十六进制的地址值,注意:在不同计算机上,地址值可能是不同的。

(2) 观察结果,可看到同一数组各元素的地址是连续的,不同类型数组元素的存储单元的大小是不同的。

(3) 使用"%d"格式输出字符数组各元素的 ASCII 码值,可以看到空字符的 ASCII 码值为 0(指 b 数组最后两个元素)。

(4) 使用"%s"格式可输出存储于字符数组中的一个字符串。

2. 对一个 4 行 4 列的二维数组,完成以下操作。

(1) 求对角线上的元素之和与积。

(2) 求所有靠边的元素之和与积。

(3) 求所有不靠边的元素之和与积。

分析:对于一个行列相同的二维数组,它有两条对角线,一条称为主对角线,另一条称为副对角线。主对角线上的元素下标特点是行、列的值相等;副对角线上的元素下标特点是行、列下标之和等于"行数-1";所有靠边的元素下标特点是行下标等于0或者等于"行数-1",列下标等于0或者等于"列数-1";所有不靠边的元素下标特点是行下标不等于0,也不等于"行数-1";列下标不等于0,也不等于"列数-1"。

程序清单 prg5_2.c 如下:

```
#include <stdio.h>
void main(){
int x[4][4],i,j;
  long djh=0,djj=1,kbh=0,bkbh=0,kbj=1,bkbj=1;
  for(i=0; i<4; i++){
      printf(" 请输入第%d行:",i+1);
      for(j=0; j<4; j++)
        scanf("%d",&x[i][j]);
}
for(i=0; i<4; i++)
  for(j=0; j<4; j++){
      if(i==j || i+j==3) {
        djh+=x[i][j];          //求 djh(对角线元素和)
        djj*=x[i][j];          //求 djj(对角线元素积)
      }
      if(i==0 || i==3 || j==0 || j==3) {
        kbh+=x[i][j];          //求 kbh(靠边元素和)
        kbj*=x[i][j];          //求 kbj(靠边元素积)
      }
      if(i!=0&&i!=3&&j!=0&&j!=3){
        bkbh+=x[i][j];         //bkbh(不靠边元素和)
        bkbj*=x[i][j];         //bkbj(不靠边元素积)
      }
  }
  printf(" 对角线元素和=%ld, 对角线元素积=%ld\n",djh,djj);
  printf(" 靠边元素的和=%ld, 靠边元素的积=%ld\n",kbh,kbj);
  printf(" 不靠边元素的和=%ld, 不靠边元素的积=%ld\n",bkbh,bkbj);
}
```

运行结果:

```
请输入第1行:1 5 7 2
```

请输入第 2 行:7 4 2 1
请输入第 3 行:5 6 8 9
请输入第 4 行:2 4 5 6
对角线元素和=31，对角线元素积=9216
靠边元素的和=54，靠边元素的积=5292000
不靠边元素的和=20，不靠边元素的积=384

3. 编程输出杨辉三角的前十行。

```
1
1   1
1   2   1
1   3   3   1
1   4   6   4   1
1   5  10  10   5   1
1   6  15  20  15   6   1
…   …   …   …   …   …   …   …
```

分析:这些数据的特点是对角线和第一列上的值为 1;其余从第 3 行起,每行自第 2 个数开始的各个数等于上一行同一列上的数与其左侧的数之和。即满足:

a[i][j]=a[i−1][j]+a[i−1][j−1]

程序清单 prg5_3.c 如下:

```
#include <stdio.h>
void main(){
  int a[10][10],i,j;
  for(i=0; i<10; i++){
      a[i][0]=1;
      a[i][i]=1;
  }
  for(i=2; i<10; i++)
    for(j=1; j<i; j++)
        a[i][j]=a[i-1][j]+a[i-1][j-1];
  for(i=0; i<10; i++){
      for(j=0; j<=i; j++)
      printf("%5d",a[i][j]);
      printf("\n");
  }
}
```

运行结果:

```
1
1   1
1   2   1
1   3   3   1
1   4   6   4   1
1   5  10  10   5   1
1   6  15  20  15   6   1
1   7  21  35  35  21   7   1
1   8  28  56  70  56  28   8   1
1   9  36  84 126 126  84  36   9   1
```

4. 设有两个矩阵为：

$$A=\begin{bmatrix}1&1&1\\2&2&2\\3&3&3\end{bmatrix}\qquad B=\begin{bmatrix}4&4\\5&5\\6&6\end{bmatrix}$$

求乘积矩阵 C=AB。要求通过编制矩阵乘法的函数 matproduct(a,b,c,m,n,p)实现。

根据矩阵乘法规则，若 $C=A_{m\times n}\cdot B_{n\times p}$，则计算公式为：

$$C_{ij}=\sum_{k=0}^{n-1}A_{ik}\cdot B_{kj}\qquad i=0,\cdots,m-1;j=0,\cdots,p-1$$

源程序文件 prg5_4.c 清单如下：

```
matproduct(int a[][3],int b[][2],int c[][2],int m,int n,int p){
    int i,j,k,s;
    for(i=0; i<m; i++)
        for(j=0; j<p; j++){
            for(k=s=0; k<n; k++) s+=a[i][k]*b[k][j];
            c[i][j]=s;
        }
}

main(){
    static int a[3][3]={{1,1,1},{2,2,2},{3,3,3}};
    static int b[3][2]={{4,4},{5,5},{6,6}};
    int c[3][2],i,j;
    matproduct(a,b,c,3,3,2);
    for(i=0; i<3; i++)
        for(j=0; j<2;j++)
            printf("%6d%c",c[i][j],((j+1)%2==0)? '\n':' ');
}
```

运行结果：

```
15    15
```

```
30      30
45      45
```

5. 编制程序，读入一串正文行，打印出其最长行的内容及其串长，然后再对最长行采用选择法按由小到大排序输出。

程序的基本算法描述如下：

```
while(还有其他行)
  if(它比以前最长的还要长)
    保存它和它的长度;
打印最长行及串长;
排序最长行并输出;
```

要实现该功能，首先要编写一函数 getline 读一行，要保存长行还必须要有一个函数 copy 复制字符数组。最后的排序处理由 sort 函数完成。

按前述程序流程编制的源程序文件 prg5_5.c 清单如下：

```
#include <stdio.h>
#define MAXLINE 100
void main(){
  int getline(char s[],int lim);
  void copy(char s1[],char s2[]);
  void sort (char s[],int 1);
  int len,max=0; char line[MAXLINE],save[MAXLINE];
  while((len=getline(line,MAXLINE))>0)
      if(len>max) {
      max=len; copy(line,save);
  }
  if(max>0)
      printf("%d   %s\n",max,save);
  sort(save,max);
  printf("%d   %s\n",max,save);
}

getline(char s[],int lim) {
  int c,i;
  for(i=0; i<lim-1&&(c=getchar())!=EOF&&c!='\n'; i++)
      s[i]=c;
  if(c=='\n')
      s[i++]=c;
  s[i]='\0';
  return(i);
```

```
}
void copy(char s1[],char s2[]){
  int i=0;
  while((s2[i]=s1[i])!='\0')
      ++i;
}
void sort(char s[],int l){
  int i,j,k; char t;
  for(i=0; i<l-1; i++){
      k=i;
      for(j=i+1; j<l; j++)
          if(s[j]<s[k]) k=j;
      t=s[k]; s[k]=s[i]; s[i]=t;
  }
}
```

运行结果：

```
test↙
the test↙
test3↙
^D
  9   the test
  9
eehsttt
```

6. 在同一图纸上打印出以下方程的图形：

$$\begin{cases} y_1 = \sin x & \text{用“*”号，} -180^\circ \leqslant x \leqslant 180^\circ \text{ 间隔 } 15^\circ \\ y_2 = x/3 & \text{用“\$”号，} -\pi \leqslant x \leqslant \pi \end{cases}$$

(1) $y_1=\sin x$，用“*”号打印，x 取值为$-180^\circ\sim180^\circ$间所有 15°的整数倍点。

(2) $y_2=x/3$，用“\$”号打印，x 取值为$-\pi\sim\pi$。

(3) 曲线的中心轴线用“.”表示，位置在第 40 列。

设计这个程序需要考虑如下两个问题：

(1) 在一行中有可能要打印三个函数的点：① $y_1=\sin x$；② $y_2=x/3$；③ 中心轴线。我们在这儿采取的是根据函数值来确定打印字符的位置，并由此来置换字符子串的值。我们设字符数组的内容为 80 个空格符，这样不在函数值的位置上的字符为空格，使图形清晰。

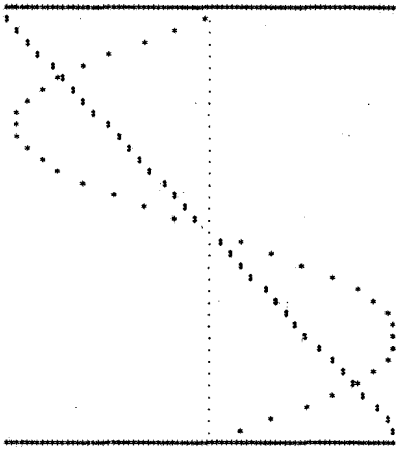
(2)因为 $\sin x$ 的值是在$-1\sim1$ 之间，因此如果不考虑振幅，其函数值密集，使得打印的图形无法显示出正弦曲线，因此程序中应作相应的放大振幅处理。

源程序文件 prg5_5.c 清单：

```
#include <stdio.h>
```

```
#include <math.h>
void main(){
int i,iy1,iy2;
char a[80],b[80];
    double x,y1,y2;
    for(i=0; i<80; i++){
        a[i]=' ',b[i]='*';
    }
    a[79]='\0';b[79]='\0';
    printf("%s\n",b);
    x=-3.1415926;
    while(x<3.1415926){
        y1=sin(x);
        y2=x/3.0;
        iy1=40+38*y1; //本行和下一行的 38 为调整放大振幅的调整系数,可
        iy2=40+38*y2; //适当改变其值,观察图形的变化,以了解其功用。
        a[iy1]='*';
        a[iy2]='$';
        a[40]='.';
        printf("%s\n",a);
        a[iy1]=' ';
        a[iy2]=' ';
        x=x+3.1415926/18;
    }
    printf("%s\n",b);
}
```

运行结果：



7. 若矩阵某位置上的元素在该行上为最小，而在该列上为最大，则称该位置（即行列号）为矩阵的鞍点。编程输出给定矩阵中的所有鞍点。

具体要求：

编制一通用的输出给定矩阵中所有鞍点的函数 andian(int a[],int m,int n)。其中，a 所对应的实参为二维数组，且为 m 行 n 列的矩阵。

分析： 从矩阵的第 0 行开始到最后一行为止，逐行寻找该行中值最小的所有元素，然后对其中的每一个元素所在的列寻找最大值，如果该列上的最大值与该行的最小值相等，则说明该元素为鞍点，将它所在的行号和列号输出。

为了使函数 andian()中的一维数组与矩阵对应，需要进行下标变换。矩阵中第 i 行(i=0,1,…,m−1)第 j 列(j=0,1, …,n−1)的元素在一维数组中的下标为 i * n+j。

函数 andian(a,m,n)PAD 流程图如 3.10 所示。

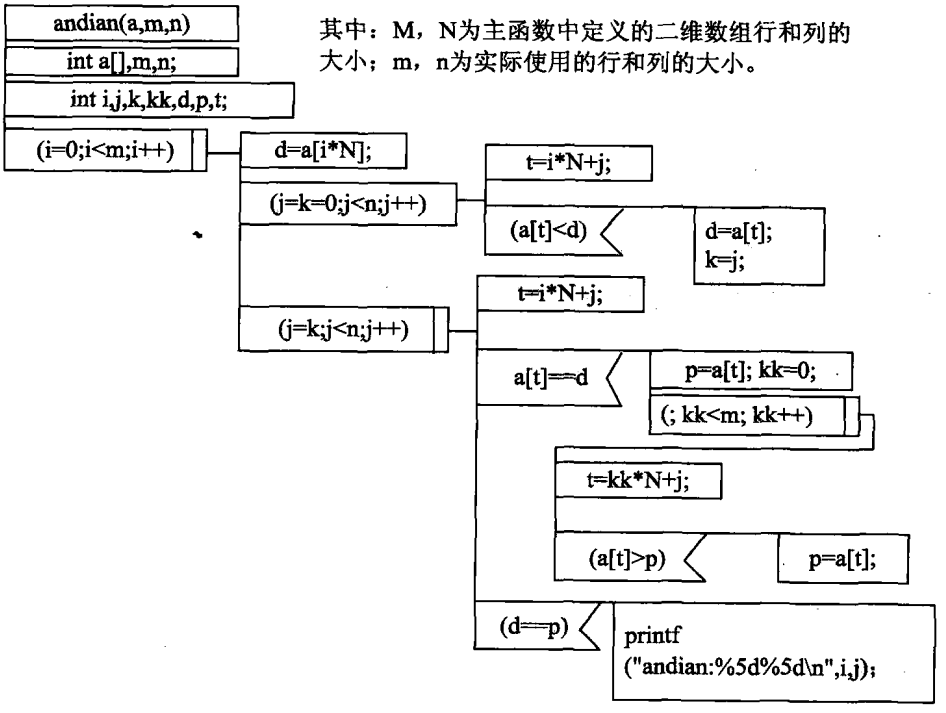


图 3.10 求矩阵鞍点的 PAD 图

源程序文件 prg5_6.c 清单：

```
#include <stdio.h>
#define M 10
#define N 10
main(){
  int i,j,k,m,n,a[M][N];
  printf("enter m,n:");
  scanf("%d%d",&m,&n);
  for(i=0; i<m; i++)
```

```
    for(j=0; j<n; j++)
      scanf("%d",&a[i][j]);
    andian(*a,m,n);
  }
  andian(int a[], int m, int n) {
    int i,j,k,kk,d,p,t;
    for(i=0; i<m; i++){
      d=a[i*N];
      for(j=k=0; j<n; j++){
        t=i*N+j;
        if(a[t]<d) {
          d=a[t];k=j;
        }
    }
      for(j=k; j<n; j++){
        t=i*N+j;
        if(a[t]==d) {
          p=a[t];
          for(kk=0; kk<m; kk++){
            t=kk*N+j;
            if(a[t]>p) p=a[t];
          }
          if(d==p)printf("andian:%5d%5d\n",i,j);
        }
      }
    }
  }
```

运行结果：

```
enter m,n:   3 4
1   2    3   4
9   8   10   8
3   5    6   7
andian:     1     1
andian:     1     3
```

（三）实验内容

1. 上机实现范例第1,2,3题。

2. 输入 n 个整数($n\leqslant 20$),存放在一个一维数组中。然后在第 m 个整数后插入一个输入的整数(后续数组元素都后移一位,称移位腾空),输出最终的数组。

3. 输入 n 个整数($n\leqslant 20$),存放在一个一维数组中。然后输入一个数 m,并将第 m 个整数删除(后续数组元素都得前移一位,称移位补缺),输出最终的数组。

4. 已知 $a[20]$,$b[10]$两个数组,其中,a 为升序数组,编写程序将 b 数组中的各个元素插入 a 数组,并保证 a 仍为升序数组。

5. 从键盘输入 10 个学生的姓名和成绩,分别用冒泡排序法和选择排序法按成绩降序排序并输出(要求姓名和成绩对应关系保持不变)。

6. 有 15 个数存放在一个数组中,输入一个数,要求用折半查找法找出该数是数组中第几个元素的值。如果该数不在数组中,则输出"无此数"。15 个数可采用数组赋初值的方法在程序中给出。要找的数用 scanf()函数输入。

7. 编制用户自定义函数 udf_strcat(s,t),实现两个字符串连接。(注:不要使用系统提供的 strcat()函数实现。)

8. 编制用户自定义函数 udf_strcmp(s,t),实现按字典顺序比较 s,t 两个字符串的大小,s 大于 t 则返回正值,s 等于 t 则返回 0,s 小于 t 则返回负值,正负值的大小正好是 s、t 中第一个所遇到的不相等的两个字符值之差。(注:不要使用系统提供的 strcmp ()函数实现。)

9. 编制用户自定义函数 udf_strcpy(s,t),实现将 t 所代表的字符串拷贝到 s 中,即字符串拷贝函数。(注:不要使用系统提供的 strcpy ()函数实现。)

10. 产生 50 个随机数,用插入排序法按非递减顺序进行排序。

具体要求如下:

(1) 在产生随机数的过程中,每产生一个随机数就插入到前面已有序的数组中。

(2) 输出时要求每行输出五个数据,并上下对齐。

随机数可采用如下函数产生:

```
rand(){
static int seed=0;
  int l=29,c=217,m=1024;
  if(seed==0) {
    printf("Enter rand number seed:(0--32767)");
    scanf("%d",&seed);
  }
  seed=(seed*l+c)-(seed*l+c)/m*m;
  return(seed);
}
```

随机数也可调用库函数 rand()产生,该函数可随机产生 0~32767 内的整数。随机函数需要设置种子,方法是使用下述语句:

```
srand((unsigned int)time(NULL));
```

语句中有时钟 time 参数,因为时间始终都在变,就会使 rand 函数产生的第一个随机函数不会固定不变。

为调用产生随机数的函数,需要使用预处理命令,包含库函数头文件 stdlib. h ;

为调用时钟 time 函数,需要包含库函数头文件 time. h 。具体应用如以下程序段所示:

```
#include<stdlib.h>
#include<time.h>
int main(){
    int i=0,x;
    srand((unsigned int)time(NULL));                    //设置种子
    for(i=0;i<10;i++)
        x=rand();                                       //调用随机数
    ......                                              //完成插入排序等操作
    printf("\nRAND_MAX:%d\n",RAND_MAX);                 //输出最大随机数
}
```

插入排序可参照以下函数：

```
void insertsort(int r[],int i,int x){
    //i 为数组 r 中已有序的元素的个数
    while(i!=0 && r[i-1]<x){
        r[i]=r[i-1];
        i--;
    }
    r[i]=x;
}
```

11. 判别用户输入的字符串是否符合构成标识符的规定。如果符合，则形成一个标识符，并计数；否则，显示出错信息。构成标识符的规定如下：

(1) 以字母或下划线“_”中任一字符打头。

(2) 在第一字符之后，可以是任意的字母、下划线或数字组成的字符序列，这个字符序列可以是空串。

12. 设有两个矩阵为：

$$A=\begin{bmatrix}1&2&3&4&5\\6&7&8&9&10\\11&12&13&14&15\\16&17&18&19&20\\21&22&23&24&25\end{bmatrix}\qquad B=\begin{bmatrix}3&0&16\\17&-6&9\\0&23&-4\\9&7&0\\4&13&11\end{bmatrix}$$

求乘积矩阵 $C=AB$。

具体要求如下：

(1) 矩阵 A 与 B 的元素在程序中直接用数组初始化进行赋值。

(2) 以矩阵形式输出乘积矩阵 C。

13. (选做题)输入一行字符，统计其中单词的个数(单词之间可能有多个空格)。

14. (选做题)输入两个字符串 str1 和 str2，计算 str2 在 str1 中出现的位置(从 0 开始计算)。输出位置结果。例如，输入 str1 为“How are you!”，str2 为“are”，那么位置结果应该是 4。

15. (选做题)写一个程序，将输入复制到输出，对于几个连续的相同的行，只复制一行。

16. (选做题)用实型数组存储多项式，数组的第 i 个元素存储多项式的 i 次幂，如多

项式

$$6.7x^5-10.8x^3+0.49x^2+2.7$$

表示为

0	1	2	3	4	5
2.7	0	0.49	−10.8	0	6.7

分别编写函数，实现上述存储方式中多项式的加法和乘法。

17.（选做题）设多项式

$$P(x)=a_nx^n+a_{n-1}x^{n-1}+\cdots+a_1x+a_0$$

的系数和幂次存于下表中：

幂次	幂次	幂次	幂次	幂次	幂次
系数	系数	系数	系数	系数	系数

例如：

$$P9(x)=3.0x^5+4.2x^3+2.1x^2+7$$

保存成

5	3	2	0
3.0	4.2	2.1	7

设计保存多项式的数据结构并编写一个函数计算多项式的值。

18.（选做题）打印杨辉三角形的前10行。杨辉三角形亦称为Pascal三角形，它描述了二项式系数的规律，是我国南宋时期著名数学家杨辉发现的。杨辉把有关二项式系数的研究记载在其著作《译解九章算法》(1261年)中。法国人Pascal也有类似的结果(1650)年，所以国外书刊中称为Pascal三角形，但比杨辉晚了近400年。

杨辉三角形的形式如图T18所示。此三角形的特点是：

(1)从第0行开始编号，第n行诸数正好是二项式$(a+b)^n$展开各项的系数。

(2)从第2行开始，每行数字除了两端均是1之外，其余数字正好是它上一行中与它左右相邻的两个数之和。

打印杨辉三角形，显然要一行一行地生成打印，得到图T18。

```
        1
      1   1
    1   2   1
  1   3   3   1
1   4   6   4   1
1   5  10  10   5   1
```

图T18 杨辉三角

19.（选做题）编程打印如下函数的图形：

$$\begin{cases} y_1=\sin 2x \\ y_2=2\cos(2x+30^\circ) \end{cases}$$

其中，$x=0^\circ,10^\circ,20^\circ,\cdots,360^\circ$。

20.（选做题）上机实现范例第6题，通过求矩阵鞍点，理解二维数组在内存中的存放形

式，掌握下标变换以实现正确存取数据。

21.（选做题）编制程序，生成并打印 n 阶魔方阵。

魔方阵的元素为 $1\sim n^2$（n 为奇数）之间的自然数，方阵的每一行、每一列及对角线之和都相等，其和为：$n\times(n^2+1)/2$。魔方阵元素的排列有如下规律：

(1) 自然数 1 总是排在方阵第一行当中一列上。

(2) 后续的自然数在当前数的右上方，即行数减 1、列数加 1 的位置。若当前数在第一行但不在最后列，则后续数在最后一行的下一列上；若当前数在最后一列，则后续数在上一行的第一列；若当前数在最后一列，且当前行下标为第 0 行，则后续数放在下一行的当前列上。

(3) 若按照规律(2)得出的位置已被占用，则下一个自然数放在当前数的下一行同列上。

例如，5 阶魔方阵如下所示：

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

22.（选做题）约瑟夫(Josephus)问题。

古代某法官要判决 n 名犯人死刑，它有一条荒唐的逻辑，将犯人首尾相接排成圆圈，然后从第 s 个人开始数起，每数到第 m 个犯人，则拉出来处决；然后再数 m 个犯人，数到第 m 个犯人再处决；……直到剩下的最后一个犯人可以赦免。编写程序，给出处决顺序，并告知哪一个人可以活下来。

23.（选做题）设计并实现一个在单行文本上执行编辑操作的程序。这个编辑器应该能够定位指定的目标子串、删除子串、在指定位置插入子串。编辑器要求原字符串不超过 80 个字符。要编辑的原文本行由主函数输入获得，然后重复输入并处理编辑命令，直到接收到 Q(退出)命令。

文本编辑器程序的结构可参照图 T23 所示。

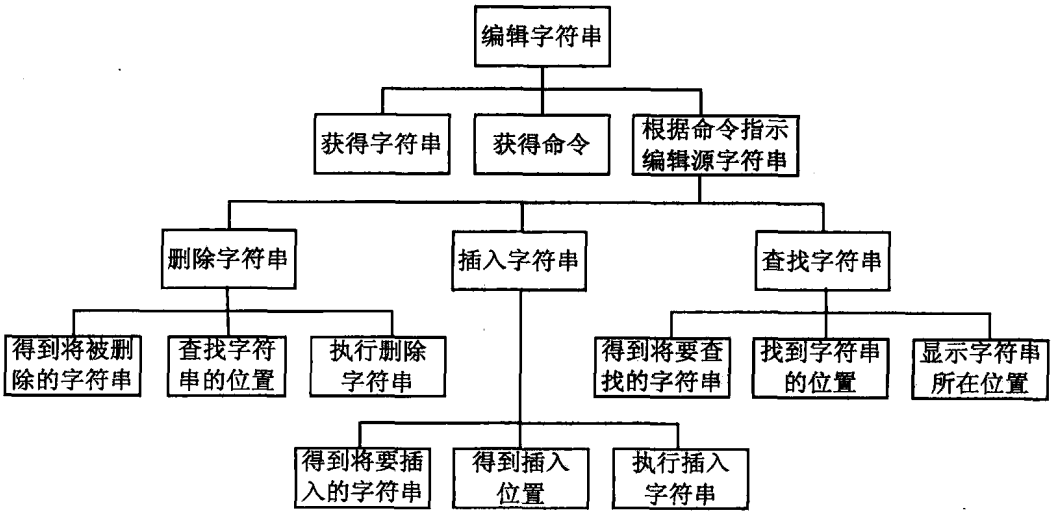


图 T23　文本编辑器程序的结构

（四）常见错误分析

定义数组时，必须使用整型常量表达式指定数组的大小，即定义一维数组时应确定数组中元素的个数，定义二维数组时应确定其行数和每行的列数。

【错误示例 5.1】 以下源程序为生成并打印 n 阶魔方层程序，程序中存在定义数组时没有指定数组大小的错误，请读者自行分析并改正之。

```
#include<stdio.h>
#define MAX 15
main(){
    int i,j,k,n;
    int m[][];          //存在定义数组时没有指定数组大小的错误
    printf("input the number you wanted! \n");
    scanf("%d",&n);
    if((n>0)&&(n%2!=0)){
        printf("n=%d\n",n);
        i=0;
        j=n/2;
        for(k=1;k<=n*n;k++){
            m[i][j]=k;
            if(k%n==0)
                if(i==n-1)
                    i=0;
                else
                    i++;
            else{
                if(i==0)
                    i=n-1;
                else
                    i--;
                if(j==n-1)
                    j=0;
                else
                    j++;
            }
        }
        for(i=0;i<n;i++){
            for(j=0;j<n;j++)
                printf("%5d",m[i][j]);
            printf("\n");
        }
```

```
        }
        else
            printf("error in input data!");
    }
```

改正后，准确的程序在运行后输入 5 时应得到如下结果：

```
input the number you wanted!
5
n=5
    17  24   1   8  15
    23   5   7  14  16
     4   6  13  20  22
    10  12  19  21   3
    11  18  25   2   9
```

在被调函数中定义作形参的多维数组名时，可以指定每一维的大小；也可以省略其第一维（最左边的一维）的大小说明，但是对其他维的大小说明不能省略。

【错误示例 5.2】 求乘积矩阵 $C=AB$。要求通过编制矩阵乘法的函数 matproduct() 实现。

设主函数 main()定义如下：

```
main(){
  static int a[3][3]={{1,1,1},{2,2,2},{3,3,3}};
  static int b[3][2]={{4,4},{5,5},{6,6}};
  int c[3][2],i,j;
  matproduct(a,b,c,3,3,2);
  ......
}
```

若 matproduct()函数定义如下，则是错误的。

```
matproduct(int a[][],int b[][],int c[][],int m,int n,int p){
    ......          //函数体定义省略说明
}
```

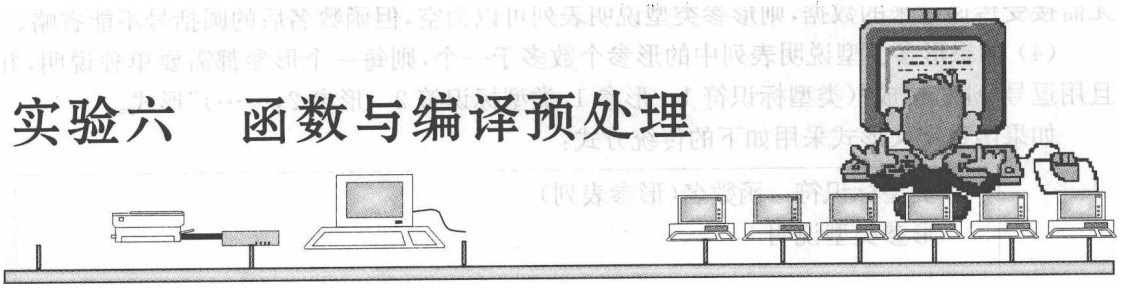
正确的 matproduct()函数应定义如下：

```
matproduct(int a[][3],int b[][2],int c[][2],int m,int n,int p){
    ......          //函数体定义省略说明
}
```

或者

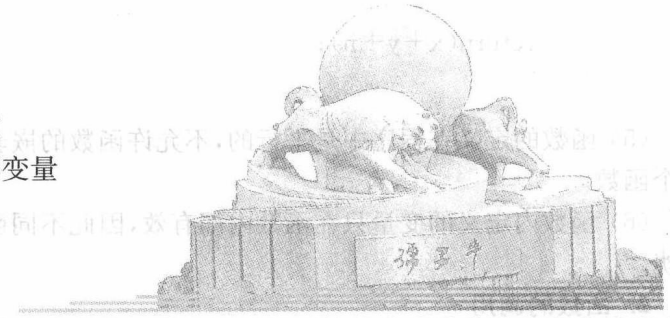
```
matproduct(int a[3][3],int b[3][2],int c[3][2],int m,int n,int p){
    ......          //函数体定义省略说明
}
```

实验六　函数与编译预处理



学习重点

- 函数的定义与调用方法
- 函数形参和实参的数据传递
- 函数的类型与返回值
- 函数的说明和声明
- 函数的递归调用
- 变量的存储类别
- 全局变量和局部变量
- 自动变量和静态局部变量
- 宏定义
- 文件包含



一、要点综述

1. 函数的定义

函数定义的一般形式：

```
类型标识符　函数名(形参类型说明表列){
    数据定义说明部分；
    执行语句部分；
}
```

(1) 类型标识符确定函数返回值的类型，所以应与函数体内的 return 语句返回值的类型相同。它可以是任何一种有效的类型，当函数类型缺省时为整型，当函数无返回值时可以用 void 类型标识符说明。

(2) 函数名必须符合 C 语言规定的标识符命名规则，且函数名必须唯一，不能与函数体内变量或形式参数同名。

(3) 形参类型说明表列中的形参用于接受主调函数传递过来的相应实参的值。若函数

无需接受传递过来的数据，则形参类型说明表列可以为空，但函数名后的圆括号不能省略。

(4) 如果形参类型说明表列中的形参个数多于一个，则每一个形参都需要单独说明，并且用逗号分隔，构成"(类型标识符 1　形参 1，类型标识符 2　形参 2，……)"形式。

如果函数定义形式采用如下的传统方式：

```
类型标识符　函数名(形参表列)
形参类型说明
{
    数据定义说明部分；
    执行语句部分；
}
```

则形式参数无需一个一个地单独说明，即同类型元素可在相应类型标识符后用逗号分隔一起说明。例如：

```
double fun1(x,y,n)
double x,y; int n;
{
    return(x+y+n);
}
```

(5) 函数的定义在程序中是平行的，不允许函数的嵌套定义，即不能在函数内部定义另一个函数。

(6) 函数内定义的变量只在函数内部有效，因此不同函数中变量的名字可以相同，互不干扰。

2. 函数的调用

函数调用时，要求实参与形参在个数、顺序、类型和单位上必须对应一致。

函数调用的一般形式为：

```
函数名(实参表列)；
```

(1) 函数的调用应遵守先定义后使用的调用规则，若函数定义在后，调用在先，则应在主调函数的定义说明部分给出被调函数原型的说明。说明形式如下：

```
类型标识符　函数名(形参类型说明表列)；
```

注意正确区分函数定义和函数说明：

① 定义是对函数功能的确立。

② 说明是对已定义函数的函数名、函数类型，形参的个数、顺序和类型等信息进行申明。

(2) 如果实参表列中含有多个参数，则各参数应用逗号分隔。若实参表列为空，则可以省略，但函数名后的括号不能省略。

(3) 函数调用可以出现在任何允许表达式出现的地方，也可以作为独立的函数调用语句。

(4) 除主函数外，其他函数可以嵌套调用，也可以递归调用。

3. 函数参数的两种传递方式

(1) 传值方式(单向传递)：形参可以是简单变量，实参可以是常量、变量、表达式、数组元素等。特点是：形、实参数一旦传递结合，即数据传递完毕，实参与形参不再有实际联系，也即形参在函数中的值的变化不会改变主调函数中实参的原有值。

(2) 传地址方式(双向传递)：形参可以是数组名和指针变量，实参可以是数组名、指针变量、字符串常量、指定单元的起始地址等。特点是：通过传递数组或某一存储单元的起始地址，使被调函数可利用此地址来访问(存取)相应存储单元的数据，实质上为通过存储单元共享达到数据双向传递的目的。

4. 函数的返回值与函数类型说明

函数的类型应与 return 中的表达式类型一致，函数类型决定返回值的类型。

(1) 若 return 后面括号中的表达式为非整型，则必须在函数名前冠以函数的类型说明。因为函数类型决定返回值的类型。

(2) 若无返回值，可用 void 类型定义函数。

(3) 一个函数可以有多个 return 语句。形式可为：

① return(表达式)；　　//带值返回

② return；　　//不带值返回

(4) 在 UNIX 系统，一个函数若无 return 语句，则遇最外层'}'即返回。

5. 函数间的数据联系

函数间的数据联系可通过以下三个途径：

(1) 通过 return 语句返回一个值(数值或地址)。

(2) 形实参数传递结合(传值方式或传地址方式)。

(3) 通过外部(全局)变量进行数据传递。

6. 函数的递归调用

(1) C 语言的函数除了 main()函数，都可以递归调用，即可以直接或间接地调用自己。

(2) 递归调用必须在满足一定条件时可以结束递归调用，否则无限地递归调用将导致程序无法结束。构成递归的条件：

① 确定的递归结束条件及结束时的值。

② 能用递归表达式形式表示，并且递归向结束条件发展。

7. 变量的存储类别

C 程序中，每一个变量和函数都有数据类型和存储类别两个属性：数据类型确定数据的存放形式、值域及允许的运算等，存储类别确定数据的存储区域等。

完整的变量定义形式：

```
存储类别　数据类型　变量名表列；
```

完整的函数定义形式：

```
存储类别　数据类型　函数名(形参表){……}
```

(1) C 语言用来说明存储类别的关键字有：auto(自动)、register(寄存器)、static(静态)、

extern(外部)。具体说明如图 3.11 所示。

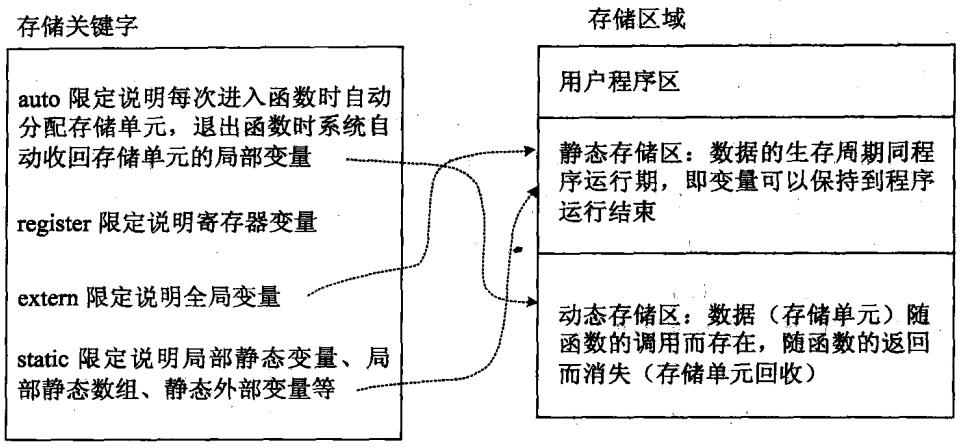


图 3.11　存储类别关键字

(2) 函数内定义的变量,若不指定存储类别,则作 auto 变量处理。函数外定义的变量为全局变量。

(3) 在定义和说明变量时,应按照变量的作用范围以及它们在存储单元中保持值的时间长短,对它们的存储类别进行说明限定。

(4) 函数的存储类别只能使用 extern, static 这两个限定词,以限制函数的调用范围。省略存储类别隐指外部函数,可供其他文件中的函数调用;静态函数只能被本程序文件的函数调用。

8. 局部变量及其作用域

函数内部定义的变量统称为局部变量,包括形参变量,由于存放在动态区,故也称为动态变量或自动变量。局部变量只有在所属函数被调用时系统才为其分配存储单元,当退出作用域后其值不作保留。

(1) 局部变量的作用域:从定义的位置起到函数体(或复合语句)结束止,即函数内定义的变量只限本函数内有效,复合语句内定义的变量只限本复合语句内有效,其他函数(或复合语句)不能使用它。main()函数中定义的变量也只有主函数内有效。

(2) 不同函数中定义的变量可以同名,代表不同的对象,互不干扰。

(3) 形参也是局部变量。

(4) 没有指定类别作 auto 处理。

(5) 内层定义的局部变量优先。

(6) 局部动态变量的初值不定。

9. 寄存器变量及其作用域

为了提高执行效率,允许将局部变量的值放在 CPU 的寄存器中,需要时直接从寄存器取出参加运算。只有局部自动变量和函数的形参可以作为寄存器变量,并且类型只限于 int、char 和指针类型。

10. 外部变量及其作用域

在函数外定义的变量称为外部变量,即全局变量,可以为本文件中其他函数所共用。也

可以使用 static 和 extern 两种关键字进行说明。

(1) 外部变量的有效作用域:从定义变量的位置开始到本源文件结束。即全局变量可以被在其定义位置之后的函数所共享。因此,外部变量为不同函数间进行数据通信(传递)提供了另一途径。

(2) 外部(extern)说明:一个 C 程序可以由几个不同的源程序文件组成,如果组成这一个程序的几个文件需要用到同一个全局变量,或者在定义点之前的函数中想引用某外部变量,则应该在其他引用该全局变量的源程序文件中或相关函数中用关键字 extern 作"外部变量"说明。形式如下:

```
extern  类型标识符  变量表列;
```

(3) 静态(static)说明:如果希望一个源程序文件中的全局变量仅限于该文件使用,则只要在该全局变量定义时的类型说明前加一个 static 即可。

```
static  类型标识符  变量表列;
```

(4) 使用全局变量会使函数的通用性及程序的清晰度降低,要限制使用。

11. 静态变量及其作用域

(1) 函数体内用 static 存储类别定义的变量为静态局部变量。静态局部变量的生存期是整个程序的运行期,即在程序的整个执行过程中始终存在,但是它在作用域之外不能使用。

(2) 注意动态局部变量与静态局部变量的区别。

动态局部变量:

① 其作用域在空间上是函数或复合语句的内部,在时间意义上是函数的一次调用周期或复合语句的执行周期;即进入函数时系统自动为其分配存储单元,函数调用结束时系统自动收回该存储单元。

② 其初值不定。

静态局部变量:

① 其值不随函数调用或复合语句的执行结束而消失,也就是当再次进入函数或复合语句时,这些局部变量仍能保持上次函数调用时的值;也可以说静态局部变量在编译时只分配一次存储单元,并一直保持到程序结束。

② 有确定的初值,数值为 0,字符为"\0"。

③ 静态局部变量的作用域不变。仍为本函数内有效。

12. 宏定义

(1) 宏定义分为不带参数的宏定义和带参数的宏定义。

① 不带参数的宏定义格式:

```
#define  宏名  字符串
```

它的作用是在编译预处理时,将源程序中所有宏名替换成字符串。

② 带参数的宏定义格式:

```
#define  宏名(参数表)  字符串
```

它的作用是在编译预处理时，将源程序中所有宏名替换成字符串，并且将字符串中的参数用实际使用的参数替换。注意：宏名和参数表之间不能有空格，否则空格后面的所有字符序列都作为替换的字符串。

(2) 宏名习惯用大写字母，以与变量区别。

(3) 宏定义就是用一个宏名代替一个字符串，以简化编程。

(4) 对宏定义的预处理就是做逆向的置换。

(5) 可以层层置换，先定义的可被后者引用。

(6) 带参数的宏定义在预处理时，也只作字符和参数的替换，不进行任何计算操作。只有在编译后运行时，才按照替换后的表达式并根据表达式的运算规则进行运算。所以，一般在定义宏时，字符串中的形式参数外面应加一个小括号。

13. 文件包含

文件包含预处理命令的一般形式：

```
#include <filename>      或 #include "filename"
```

(1) 前一种文件名用尖括号括起来的形式，系统在处理时将到包含 C 库函数的头文件所在的目录(通常是 include 子目录)中寻找文件。后一种文件名用双引号括起来的形式，系统在处理时先在当前目录下寻找，若找不到，则再到操作系统的 path 命令设置的自动搜索路径中查找，最后才到 C 语言头文件所在的目录中寻找文件。

(2) 一个 #include 命令只能指定一个被包含文件。

(3) 若文件 1 包含文件 2，文件 2 又要用到文件 3 的内容，则可在文件 1 中用两个 include 命令分别包含文件 2 和文件 3，次序如下所示：

```
#include "文件 3"
#include "文件 2"
```

以上情况也可以分别包含，即一个被包含文件又可以分别包含另一个被包含文件，即包含嵌套。

14. 几种常见条件编译

条件编译命令允许对程序中的内容作选择性地编译，即可以根据一定的条件选择是否编译。

(1) 条件编译形式 1：

```
#ifdef 标识符
    程序段 1
#else
    程序段 2
#endif
```

条件编译形式 1 的作用是：当“标识符”已经有 #define 定义过了，则编译“程序段 1”，否则编译“程序段 2”。其中“#else 程序段 2”可以缺省。

(2) 条件编译形式 2：

```
#ifndef 标识符
    程序段 1
#else
    程序段 2
#endif
```

条件编译形式 2 的作用是：当“标识符”没有由 #define 定义过，则编译“程序段 1”，否则编译“程序段 2”。其中“#else 程序段 2”可以缺省。

(3) 条件编译形式 3：

```
#if 表达式
    程序段 1
#else
    程序段 2
#endif
```

条件编译形式 3 的作用是：当“表达式”值为真时，则编译“程序段 1”，否则编译“程序段 2”。其中“#else 程序段 2”可以缺省。

二、实验目的要求和实验内容

(一) 目的要求

1. 掌握函数的定义及调用函数的方法。
2. 理解模块间的数据联系，掌握函数实参与形参的对应关系及数据传递方式。
3. 理解全局变量和局部变量、静态变量、寄存器变量的概念并掌握使用方法。
4. 了解函数的嵌套调用和递归调用的方法。
5. 掌握宏定义的方法。
6. 掌握文件包含处理方法。

(二) 范例

1. 输入并运行下面程序，分析运行结果。

源程序 prg6_1.c 清单：

```
#include <stdio.h>
main(){
  int fun1(int x[],char y[]);
  static int a[6]={10,20,30,40,50},i;
  static char b[6]="USTC";
  fun1( a, b );
  printf("\na: ");
```

```
    for(i=0; i<6; i++)printf("(%x)%-4d ",&a[i],a[i]);
    printf("\nb: ");
    for(i=0; i<6; i++) printf("(%x)%-4c ",&b[i],b[i]);
    printf("\nb: ");
    for(i=0; i<6; i++) printf("(%x)%-4d ",&b[i],b[i]);
    printf("\nString: %s\n",b);
  }
  int fun1(int x[],char y[]){
    int i;
    printf("\nx: ");
    for(i=0; i<6; i++) printf("(%x)%-4d ",&x[i],x[i]);
    printf("\ny: ");
    for(i=0; i<6; i++) printf("(%x)%-4c ",&y[i],y[i]);
    printf("\ny: ");
    for(i=0; i<6; i++) printf("(%x)%-4d ",&y[i],y[i]);
    printf("\nString: %s\n",y);
    x[5]=x[0]+x[1]+x[2]+x[3]+x[4];
    return 0;
  }
```

运行结果：

```
x: (422a30)10  (422a34)20  (422a38)30  (422a3c)40  (422a40)50  (422a44)0
y: (422a48)U   (422a49)S   (422a4a)T   (422a4b)C   (422a4c)    (422a4d)
y: (422a48)85  (422a49)83  (422a4a)84  (422a4b)67  (422a4c)0   (422a4d)0
String: USTC
a: (422a30)10  (422a34)20  (422a38)30  (422a3c)40  (422a40)50  (422a44)150
b: (422a48)U   (422a49)S   (422a4a)T   (422a4b)C   (422a4c)    (422a4d)
b: (422a48)85  (422a49)83  (422a4a)84  (422a4b)67  (422a4c)0   (422a4d)0
String: USTC
```

分析运行结果：

(1) 前 4 行是被调用 fun1 函数的 x 和 y 形参数组的各元素的值和相应地址。

(2) 后 4 行是主调函数 main 中的 a 和 b 实参数组的各元素的值和相应地址。

(3) 分析结果可以确认形参数组和对应实参数组共用存储单元。

(4) 被调用 fun1 函数中的语句“x[5]=x[0]+x[1]+x[2]+x[3]+x[4];”修改了形参数组元素 x[5]的值，由运行结果验证可知形参数组 x 的改变影响到了实数组 a。

2. 输入年、月、日，输出该日期是该年的第几天。

源程序 prg6_2.c 清单：

```
#include<stdio.h>
int is_leap_year(int year){
```

```
int leap;
  if(year%4==0&&year%100!=0 || year%400==0)
    leap=1;
  else
    leap=0;
  return(leap);
}
int len_of_month(int year,int month){
  int month_days;
  if(month==2)
    if(is_leap_year(year))
      month_days=29;
    else
      month_days=28;
  else if(month==4 || month==6 || month==9 || month==11)
        month_days=30;
      else
        month_days=31;
  return(month_days);
}
int len_of_days(int year,int month,int date){
  int total_days,n;
  for(n=1,total_days=0;n<month;n++)
    total_days+=len_of_month(year,n);
  total_days+=date;
  return(total_days);
}
main(){
  int year,month,date,days;
  printf("pleaseinput year,month,date:");
  scanf("%d%d%d",&year,&month,&date);
  days=len_of_days(year,month,date);
  printf("%d. %d. %d is the %d in %d. \n",year,month,date,days,year);
}
```

运行结果：

```
pleaseinput year,month,date:2000 3 1
2000. 3. 1 is the 61 in 2000
```

3. 编制一个程序，把输入中包含指定字或字段，或者说指定“字符串”的那些行打印

出来。

假定指定的字符串为“the”，输入的以下 4 行：

Now is the time
for all good
men to come to the aid
of their party

通过程序的查找，将产生下面的输入行：

Now is the time
men to come to the aid
of their party

程序的设计思想如下：

```
while(是否还有输入行)
  if(该行中有指定字符串)
    打印该行
```

其中，“是否还有输入行”的判断功能可通过 prg6_3.c 中的 getline 函数完成，所以这里主要考虑编制“该行中是否有指定字符串”的函数(求子串函数)。假定被寻找指定字符串的行在形参字符数组 s[]中；指定字符串在字符数组 t[]中。该函数的功能是判断在字符数组 s 中，是否有指定的字符串 t，若有则返回指定字符串 t 在 s 字符串中的开始位置；若 s 中不包含 t 时，返回值－1。

源程序 prg6_3.c 清单：

```
#define MAXLINE 1000
#include <stdio.h>
void main(){              //find all lines matching a pattern
  int getline(char s[], int lim );
  int index(char s[],char t[]);
  char line[MAXLINE];
  while(getline(line,MAXLINE)>0)
    if(index(line,"the")>=0)
      printf("%s\n",line);
}
int getline(char s[], int lim ){
  int i,c;
  i=0;
  while(--lim>0&&(c=getchar())!=EOF&&c!='\n')
    s[i++]=c;
  if(c=='\n')
    s[i++]=c;
  s[i]='\0';
  return(i);
```

```
}
index(char s[],char t[]){
  int i,j,k;
  for(i=0; s[i]!='\0'; i++){
      for(j=i,k=0; t[k]!='\0'&&s[j]==t[k]; j++,k++)
        ;
      if(t[k]=='\0')
        return(i);
    }
  return(-1);
}
```

输入4行文本：

Now is the time↙
for all good↙
men to come to the aid↙
of their party↙

输出为：

Now is the time
men to come to the aid
of their party

4. 计算 $y=a_0+a_1x+a_2x^2+\cdots+a_nx^n=a_0+x(a_1+x(a_2+\cdots+x(a_{n-1}+x*a_n)\cdots)$。

源程序文件 prg6_4.c 清单：

```
#include "d:\test\prgtest.c"
void main(){
  float y,x;
  float f(float x );
  input_a();
  printf("Enter x:");
  scanf("%f",&x);
  y=f(x);
  printf("y=%g\n",y);
}
float f(float x ){
  int i; float y;
  y=a[N]*x;
  for(i=N-1; i>0; i--)
    y=x*(a[i]+y);
  y=a[0]+y;
```

```
    return(y);
}
```

源程序文件 prgtest. c 清单：

```
/* prgtest. c */
#include <stdio. h>
#define N 20
static float a[N+1];
float f(float x );
void input_a(){
  int i;
  printf("Please Enter a0——an:");
  for(i=0; i<N+1; i++)
    scanf("%f",&a[i]);
}
```

运行结果：

```
Please Enter a0——an:1 2 3 4 5 6 7 8 9 10↙
11 12 13 14 15 16 17 18 19 20 21↙
Enter x:2. 0↙
y=4. 1943e+07
```

源程序文件 prg6_4. c 中的文件包含命令“#include "d:\test\prgtest. c"”指定了 prg6_4. c 包含 prgtest. c 文件，位于 d:\test 目录下，prgtest. c 文件中的文件包含命令“#include <stdio. h>”指定包含标准输入输出头文件，还有宏定义、数组说明、函数说明、函数定义等。在编制由多个源程序文件组成的较大程序时，各个源程序文件共同使用的函数说明和符号常量定义等可以写成独立的包含文件。在需要包含文件内容的源程序中写入包含它的预处理语句，从而可避免重复性劳动，提高工作效力。

（三）实验内容

1. 写一个判断素数的函数，如果参数为素数，则返回 1，否则返回 0。在主函数输入一个整数，输出是否为素数的判断结果。

2. 求两个整数的最大公约数和最小公倍数。要求编写两个函数，一个函数求最大公约数，另一个函数根据求出的最大公约数求最小公倍数。建议使用以下两种数据传递方法分别编程实现：

(1) 不用全局变量，两个整数在主函数中输入，并以参数形式传送给函数 1，求出的最大公约数返回主函数，然后再与两个整数一起作为实参传递给函数 2，以求出最小公倍数，返回到主函数输出最大公约数和最小公倍数。

(2) 用全局变量的方法。将两个整数的最大公约数、最小公倍数设为全局变量。

3. 求组合数和排列数。计算公式如下：

组合数：
$$C_m^n = \frac{m!}{n!(m-n)!}$$

排列数：
$$P_m^n = \frac{m!}{(m-n)!}$$

要求该程序包含两个自定义函数用于完成相应功能：

(1) 函数 c_p_mn 既可求 m 个元素中 n 个数的组合数，又可求 m 个元素中 n 个元素的排列数。要求：m、n 的输入及输出在主函数中实现。

(2) 函数 factorial 用于计算阶乘。

4. 编写函数，其功能是在 float 类型的一维数组中查找最大值、最小值并将它们返回到调用程序。

5. 写一个函数，使输入的一个字符串按反序存放，在主函数中输入和输出字符串。

6. 编写一个函数，由实参传来一个字符串，统计此字符串中字母、数字、空格和其他字符的个数，在主函数中输入字符串并输出统计后的结果。

7. 输入 10 个学生 5 门课的成绩，分别用函数实现下列功能：

(1) 计算每个学生的平均分。

(2) 计算每门课程的平均分。

(3) 找出所有 50 个分数中最高分所对应的学生和课程。

(4) 计算平均分方差：

$$\sigma = \frac{1}{n}\sum x_i^2 - \left(\frac{\sum x_i}{n}\right)^2$$

其中，x_i 为某一学生的平均分。

8. 写一个函数验证哥德巴赫猜想，哥德巴赫猜想是说：一个不小于 6 的偶数可以表示为两个素数之和。例如：6=3+3，8=3+5，10=3+7，……在主函数中输入一个不小于 6 的偶数 n，函数中输出类似 10=3+7 的结果。

9. 用递归法将一个整数 n 转换成字符串。例如，输入 483，应输出字符组成的序列“483”。n 的位数不确定，可以是系统允许的任意整数。

10. 写一个函数，使给定的一个 $n*n$ 的二维整数型数组原地转置，即行列互换。

11. 定义一个带参数的宏，功能是两个参数的值互换。在主程序中输入两个数作为使用宏的实参，输出已交换后的两个值。

说明：规定宏只可以带两个实参，例如 swap(a,b)，不能用三个实参。

12. 阅读以下程序并给出结果。然后上机运行该程序验证阅读结果，如与运行结果不符，请分析原因。

```
#include <stdio.h>
#define PRODUCT(a,b) a*b
void fun(int i){
    static int x=1;
    x += PRODUCT(x+i, x-i);
    printf("x=%d\n",x);
}
int main(){
    int i, x=1;
```

```
    for (i=1; i<=3; i++,x++)
        fun(x+i);
}
```

13. (选做题)求方程 $ax^2+bx+c=0$ 的根,用 3 个函数分别求当 b^2-4ac 大于 0、等于 0 和小于 0 时的根并输出结果。从主函数中输入 a、b、c 的值。

14. (选做题)编写一个函数 atof(s),其功能是将数字字符串 s 转换成与其等价的双精度浮点数,并能处理以科学表示法表示的数据,如 123.45e−5 的形式(注:其中指数部分可限定为 4 位,E 或 e 字符占 1 位、±号占 1 位、后 2 位用以表示整数形式的指数)。

15. (选做题)写一个函数,将一个字符串中元音字母复制到另一个字符串,然后输出。

16. (先做题)写一个将十进制数转换指定进制数(二进制、八进制或十六进制)的函数,在主函数中输入一个十进制数,输出相应的指定进制数。

17. (选做题)编写一个函数,将一行字符串中最长的单词输出。此行字符串由主函数以参数形式传递给该函数。

18. (选做题)用条件编译方法实现以下功能:

输入一行电报文字,任选两种方式之一输出:一为原文输出;一为将字母变成其下一字母(如"a"变成"b",…,"z"变成"a")。其他字符不变。

19. (选做题)编写程序,找出 100～1000 之间的所有可逆素数。

要求:判断素数、颠倒顺序的功能用两个自定义函数实现。

提示:可逆素数是指一个素数的各位数值顺序颠倒后得到的数仍为素数,如 113,311 等。

20. (选做题)已知函数 poly 是用递归方法计算 n 阶勒让德多项式的值,数学函数如下:

$$\begin{cases} 1 & n=0 \\ \mathrm{Poly}_x(x)=x & n=1 \\ ((2n-1)*x-\mathrm{poly}_{n-1}(x)-(n-1)*\mathrm{poly}_{n-2}(x)/n & n>1 \end{cases}$$

(四) 常见错误分析

从用户使用的角度看,函数可以分成标准(库)函数和用户自定义函数两大类;从函数定义形式的角度看,函数又可以分成有参函数和无参函数两大类。**函数必须先定义后使用,否则应事先说明。**与此相应的常见错误如以下所示:

【错误示例 6.1】 以下源程序求 30°的正弦值。

```
#include <stdio.h>
main(){
  double x;
  x=sin(30.0);
  printf("sin(30.0)=%f\n",x);
}
```

程序中存在两个错误:

(1) 引用库函数 sin()时的首要条件是使用相应系统头文件。因为系统为所有库函数都

提供了相应头文件，在头文件中，含有对每一库函数的函数原型的具体说明，如函数名、函数的返回值的类型、形参表中形参的类型、个数、顺序等。为了保证能够得到正确的结果，就必须使用预处理命令“#include <math.h>”(库函数 sin()所在头文件)。

(2) sin()、cos()、tan()等库函数参数的单位应为弧度，而程序中没有进行转换。

正确的源程序应为如下所示：

```
#include <stdio.h>
#include <math.h>        //包含“math.h”头文件，以实现对 sin()函数原型的说明
main(){
  double x;
  x=sin(30.0/180.0*3.14159);          //将 30°转换为单位弧度
  printf("sin(30.0)=%f\n",x);
}
```

【错误示例 6.2】 以下源程序连续求 x(初值为 2)的平方，直至 $x<256.0$。

```
#include <stdio.h>
main(){
  double x=2.0;
  while(x<256.0){
    x=square(x);
    printf("%f\n",x);
  }
}
double square(double y){
    return(y*y);
}
```

程序中，函数 square()定义在后而使用在前，又没有在调用之前进行函数说明，故将出错。

此类错误改正的常用方法有二：

(1) 在引用 square()函数之前给以说明。

```
#include <stdio.h>
main(){
  double x=2.0;
  double square(double y);          //square()函数原型说明
  while(x<256.0){
    x=square(x);
    printf("%f\n",x);
  }
}
```

```
double square(double y){
    return(y * y);
}
```

(2) 先定义 square()函数,后引用它。

```
#include <stdio.h>
double square(double y){
  return(y * y);
}
main(){
  double x=2.0;
  while(x<256.0){
    x=square(x);
    printf("%f\n",x);
  }
}
```

另一类常见函数使用错误是试图通过将实参变量 x 的一个副本传递到被调函数,并且期望在调用被调函数之后,实参变量能表现新的值(已被改变的相应形参的值)。但形参 y 是局部的 auto 型变量,不管就其作用域或还是从所属函数调用结束后 y 就随即消失来看,参数传递采用的都是传值方式,即实参与形参只是在函数调用时作简单的复制传递,一旦完成这种传递后,实参与形参就不再有任何联系分析,不存在返回值影响实参的可能。

【错误示例 6.3】 以下源程序连续求 x 的平方值,直至 $x<256.0$。

```
#include <stdio.h>
main(){
  double x=2.0;
  double square(double y);
  while(x<256.0){
    square(x);
    printf("%f\n",x);
  }
}
double square(double y){
  return(y * y);
}
```

程序中,函数 square 的参数传递为传值方式,故调用 square 函数对主函数中的实参 x 不产生影响,不能得到预期的运行结果,同时还导致程序陷入死循环。存在调用 ()后引用返回值错误问题。

若要使实参变量 x 的值对应形参 y 在被调函数中修改以后的值,可以使用的方法较多,

这里介绍其中两种常用的正确方法：

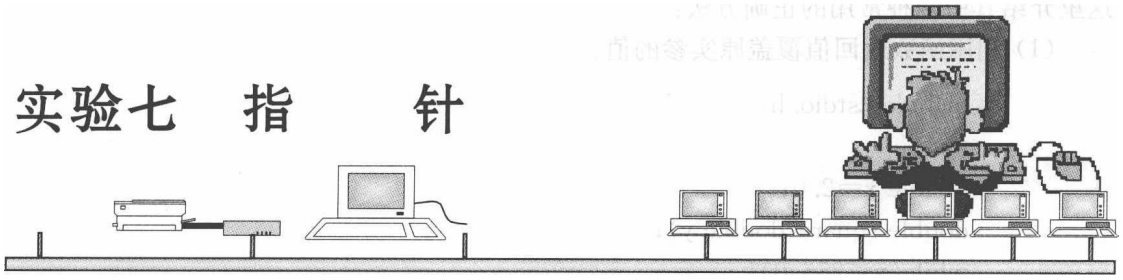
(1) 利用函数返回值覆盖原实参的值。

```
#include <stdio.h>
main(){
  double x=2.0;
  double square(double y);
  while(x<256.0){
    x=square(x);          //利用调用 square(x) 函数的返回值覆盖原实参的值
    printf("%f\n",x);
  }
}
double square(double y){
  return(y*y);
}
```

(2) 可以通过把实参变量的地址传递给一个函数的方法来改变实参变量原来的值，即采用传地址方式，实现数据在函数间的双向传递。具体将在“实验七：指针”中介绍。

```
#include <stdio.h>
main(){
  double x=2.0;
  void square(double *y);
  while(x<256.0){
square(&x);              //采用传地址方式，实现数据在函数间的双向传递
printf("%f\n",x);
  }
}
void square(double *y){
  *y=*y * *y;
}
```

实验七　指　　针



学习重点

- 指针变量的定义、赋值和运算
- 指针变量与一维数组
- 指针变量与字符串
- 指针变量做函数参数
- 返回值为指针型函数的定义
- 指针数组定义与含义
- 指向二维数组中的元素以及指向二维数组中的一维数组的指针变量的定义和引用
- 指向指针的指针的定义和引用
- 命令行参数的定义和含义
- 指向函数的指针变量的定义、赋值与引用

一、要点综述

1. 地址、指针和数据访问方式

在计算机中，内存是一个具有连续编码的空间，每个存储单元（微型机中通常指一个字节）都有一个唯一的、固定的编号，这个编号称为地址。不同的数据类型占据不同字节的存储空间，而每一个字节都有一个地址，一般把每个数据的首字节的地址称为该数据的地址。

（1）凡存放在内存中的程序和数据都有一个确定的地址。一个变量的地址称为该变量的指针。用来存放一个变量地址的变量称为指针变量。简单地说，指针就是地址，指针变量是用来存放地址的变量。

（2）通过变量名来直接引用（存取）变量的方式称为“直接引用”方式。将变量的地址存放在一个指针变量中，然后通过存放变量地址的指针变量来引用（存取）变量的方式称为“间接引用”方式。

2. 指针运算符“&”和“*”

(1) 取地址符“&”：其操作数可以是各种类型的简单变量、数组元素、结构成员，不能用于表达式、常量和寄存器变量，是描述变量地址的工具。属单目运算符。

(2) 指针运算符(或称“间接访问”运算符)“*”：属单目运算符。

其作用如下：

① 定义说明指针变量，如“inti, j, k, *i_pointer;”当指针运算符出现在定义语句中时，说明其后的标识符为存放地址的变量，系统为其分配相应的存储单元。

② 通过其操作数(指针)实现“间接引用(存取)”存储单元的数据，即将其操作数作为最终目标的地址来处理，存取的对象是该地址相应存储单元里的内容。

③ 注意：“*”要求其操作数具有指针意义的值。如“k=*i_pointer;”。

3. 指针变量的定义

指针变量必须先定义后引用，在定义中必须指出它所指向对象的数据类型。一旦说明后，就不能随意指向别的不同类型的变量。即将一个变量的地址赋给指针变量时，该变量的类型必须与指针变量的指向类型一致。

指针变量定义的一般方式：

```
类型标识符 *指针变量名,……;
```

定义指针变量时可以初始化。例如：

```
int a, *p=&a;
```

4. 指针变量的赋值

(1) 指针变量赋值常用方法：

① 使用取地址运算符“&”获取某同类型变量或数组元素的地址，以“指针变量名=&变量名”形式赋值。例如：

```
int a,*p; p=&a;
```

② 利用同类型指针变量赋值。例如：

```
int a,*p,*q; p=&a; q=p;
```

③ 利用特殊的地址符号常量 NULL(等于 0)给地址变量赋空值。例如：

```
int a,*p; p=NULL;
```

(2) 取地址运算符“&”的运算对象必须是一个变量。

(3) 指针变量是用来存放地址的，不要给指针变量赋常数值。

(4) 在指针变量没有指向确定的存储单元前，不要对它所指的对象赋值。例如：“int p; scanf("%d",p);”和“int p; *p=100;”都是错误的。

5. 指针与数组

指针与数组两者关系密切，通过指针可方便地存取数组元素。

(1) 数组的指针表示：

① 数组名：代表该数组存储单元的起始地址，可称为数组的指针，是地址常量。

② 数组元素：相当于变量，代表值，有固定的存储单元及相应地址，可通过“&”取地址符得到其地址。对元素的访问类似于变量，同样也可通过指向数组元素的指针找到所需的元素。

(2) 数组元素的几种常用引用方法。设有：

int a[10]，*pa；pa=a；

且 i 为 a 数组有效下标范围内某一确定序号(有效下标)，则可有：

① 通过数组名所代表的地址引用数组元素。例如：*(a+i)，等价于 a[i]。

② 通过指针引用数组元素。例如：*pa 等价于 a[0]，*(pa+i) 等价于 a[i]。

③ 通过带下标的指针引用数组元素。例如：pa[i]，等价于 a[i]。

(3)若 pa 是指向 a 数组中的某一确定元素时，则按定义有：

① pa+1 指向数组的下一个元素(直接后继)。

② pa−i 指向当前所指元素之前的第 i 个元素(前趋元素)。

③ pa+i 指向当前所指元素之后的第 i 个元素(后继元素)。

系统的内部实现：

pa+i 所指向的地址=pa + i*(数组元素所占字节数)

(4) 注意数组名和指针之间的区别。设有：

int a[10]，*pa；pa=a；

① 指针是变量，pa=a；pa++都是有意义的操作。

② 数组名代表数组存储单元的起始地址，是地址常量，因此，a=pa；a++都是非法操作。

(5) 正确区分赋值概念。设有：

int a[10]，*pa；

对以下两种赋值的意义要非常清楚：

```
pa=&a[0];          //赋地址，pa 指向 a 数组的第 0 号元素
*pa=123;           //赋数值，pa 所指的存储单元的值被更新为 123
```

(6)“++”和“−−”：

“++”和“−−”运算符用于指针变量十分有效，可以使指针变量直接向前或向后移动，指向下一个或上一个数组元素。例如 *pa++，关键在于弄清楚是先取得 pa 所指内存单元的值(*pa)，还是先 pa 指针自增 1(pa++)？注意这里的“++”是后置方式，所以 *pa++ 的操作结果是先取到 pa 所指内存单元的值，然后 pa 指针自增 1，即 pa 指向下一个元素。

6. 指针变量作函数参数

(1) 当一个有确定指向的指针变量或取一个变量的地址作为函数参数时，其作用是将一个变量的地址传送到另一个函数中。此时形参从实参获得相应变量的地址，即形参和实参指向同一个变量，也即形参和实参共享存储单元，因此当形参指向的变量发生变化时，实参指向的变量也随之变化。利用此特性可实现函数间数据传递(双向传递)。

(2) 必须注意：如果改变的不是指针形参指向变量的值，而只是改变指针形参的值，则效果是不同的。

7. 数组名，指针和函数参数

采用数组名和指针作为函数的参数，可通过传地址方式，实现形参和实参共享存储单元，达到数据双向传递的目的。当一个指针变量指向一维数组的第一个元素或者等于一维数组名时，数组名和指针变量的含义相同，都表示数组的首地址。所以在函数中地址的传递

也可以使用指向数组首地址的指针变量，即实参和形参使用数组名时可以使用指针变量替换。这样实参和形参可以有四种对应组合形式：

① 形参和实参都用数组名。

② 实参和形参都用指针变量。

③ 实参用数组名，形参用指针变量。

④ 实参为指针变量，形参为数组名。

(1) 一个指针变量可以指向数组中的任何元素，此时指向该数组元素的指针变量同样可以作为函数参数，传递的是指向的数组元素的地址。

(2) 实参指针变量必须有确定的地址，即必须事先指向一个可使用的存储空间，否则，可能由于局部变量的初值的不确定性，导致用户程序对重要数据区或系统文件的读写，造成严重的不可预料的后果。

8. 多维数组与指针

(1) 地址的计算与表示：

由于C语言按行优先规则存储数组元素，因此可以把二维数组看作是一维数组的集合，即二维数组是这样的一个特殊的一维数组：它的元素是一个一维数组。例如：二维数组 a[3][4]可以看成是3行4列，即它由3个一维数组 a[0]、a[1]、a[2]组成，所以二维数组只带一个下标时，表示的是构成二维数组的一行的一维数组名。以三维数组的元素为例，若设：

```
int b[3][4][5];
```

则其元素 b[i][j][k]各部分可分解表示为：

b[i][j][k] ……	元素(值)	
b[i][j] ……	一维数组名(列地址)	地址的相应单位为列
b[i] ……	二维数组名(行地址)	地址的相应单位为行
b ……	三维数组名(层或页地址)	地址的相应单位为层或页

数组名代表的维数不同，计算的地址单位也就不同，要以层、行和列的观点理解地址表示方式。

(2) 以二维数组为例说明几个注意要点。设有：

```
int a[3][4], *p; p=*a;
```

① 指针赋值时赋值运算符的两边的地址单位应一致。如 p=*a；这里 *a 的含义是将行地址转换为列地址。

② a+1 相当于以一维数组为存储单位(行)+1，表示下一行地址。

③ a[0]一维数组名单位列，a[0]+1 相当于以一个元素为存储单位+1，表示下一列地址。

④ p 是指向整型存储单元的指针，相当于以元素为单位，故 p+1 就是下移一个元素位置。若要通过 p 访问 a[i][j]则应表示为以下形式：*(p+i*4+j)。

⑤ 不同维数的数组名所代表的地址的单位不同，其存取元素的方式也不同。若以二维数组名对其下属元素进行存取，需二次间址访问；例如：*(*(a+2)+1)，存取的元素为 a[2][1]。

⑥ 若以一维数组名对其下属进行存取，一次间址访问即可：例如：*(a[0]+2)，存取的元素为 a[0][2]。

以行和列的观点解释多维数组的概念，当数组为三维时，可在行和列的基础上增加层或页的概念。其他可以此类推。

(3) 指针变量作形参以接受通过实参数组名传递来的地址时，可以采用以下常见形式：

① 用指向元素的指针变量(单位列)。

② 用指向一维数组的指针变量(单位行)。

需要注意的是形参与实参结合时对应的地址计算单位应匹配一致。

(4) 注意区分指向元素的指针与指向一维数组的指针。

① 指向元素的指针变量，即当指针变量的地址单位等于只带一维下标的二维数组时，它的定义、赋值、引用与指向一维数组元素的指针变量形式相同。例如：

```
int a[3][4],p; p=a[2];
```

此时，p 指向一维数组 a[2]的起始地址，即 p、a[2]、&a[2][0]相同。对其进行加法操作时 p+1等同于 a[2]+1，都指向数组元素 a[2][1]。所以 *(p+1)等于元素 a[2][1]的值。

② 指向一维数组的指针变量，其定义形式如下：

类型标识符 (*指针变量名)[指向的一维数组元素的个数];

例如：已知二维数组 int a[3][4]，则定义一个指向二维数组 a 中一维数组的指针变量 p 的形式是：

```
int(*p)[4];
```

9. 字符数组和指针

从只要知道字符串的起始地址，就可以对该串进行操作这一原理可知，字符指针可以代替数组名的作用，因此在 C 语言中可用两种方法表示或实现字符串的操作。

(1) 字符常数在内存中以字符数组的形式存储。

(2) 在字符数组初始化时系统首先为字符数组分配存储空间，然后采用逐个字符复制的形式给以初始化。例如：

```
static char s1[16]="Hello!";
```

(3) 给字符指针初始化是将字符串常数的起始地址赋给字符指针变量，使得字符指针指向该字符串。例如：

```
char *ps="Hello!";
```

(4) 不能通过赋值运算将一个字符串赋给字符数组名，但可以赋给一个字符型指针变量，此时该指针变量指向的是字符串的第一个字符。

(5) 指向字符串的指针变量的使用规则同指向一维数组的指针变量。

(6) 需要将字符串传递给被调函数时，可以使用字符数组名(或数组元素地址)作参数，也可以使用指向字符数组的指针变量作参数，此时属于传地址方式，形参的改变会影响实参。必要时还可以将字符串常量作为实参的参数，也属于传地址方式，此时传递的是字符串常量第一个字符的地址，但应注意不要修改形参的值，确切地说不要试图修改对应实参(字符串常量的值)。

10. 指针运算

指针是指向某类型数据存储单元的地址，需注意以下问题：

(1) 变量、数组的存储单元的分配是由编译程序决定的，存储单元的地址是确定的，因

此不能将任意的整数向指针变量赋值。

(2) 指针的值是具体类型的特定变量地址所允许的整数，因此不能像整型变量那样对指针进行所有的算术、逻辑和关系运算。

(3) 指针允许实施的运算：

① 算术运算：

设 p,q 为指针，且指向同类型数据集合，n 为整数，以下为合法且意义为：

p+n：p 指向同类型数据集合的当前位置后的第 n 个成员。

编译程序实施的内部运算：p+n*(类型所占字节的长度)。

p-n：p 指向同类型数据集合的当前位置前的第 n 个成员。

编译程序实施的内部运算：p-n*(类型所占字节的长度)。

p++：当前指针位置后移一个成员，先以 p 指针作即时操作数，然后自增。

++p：当前指针位置后移一个成员，先自增然后以自增后的 p 作为即时操作数。

p--：当前指针位置前移一个成员，先以 p 指针作即时操作数，然后自减。

--p：当前指针位置前移一个成员，先自减然后以自减后的 p 作为即时操作数。

p-q：其值表示 p,q 之间的成员个数，其值为整数，而不是指针。

② 关系运算(前提是：指针必须是指向同一类型的数据)：

若<,<=,==,!=,>=,>等运算符两边的操作数指向同一数据集合(如数组、结构等)，就成为有意义的比较，反之无意义。

p<q：表示 p,q 为指向同一数据集合中的两个不同的元素，它表明如果 p 所指向的元素在 q 所指元素之前时表达式值为“真”；否则为“假”。

其他指针关系运算意义与此类似，不再详述。

③ 赋值运算(常用于给指针赋初值)：

若设 p,q 为指针，且指向同类型数据集合，n 为整数，则 p=q, p=q+n, p=q-n, p+=n, p-=n 均有效。

(4) 指针的特殊值(NULL)：

NULL 其实就是整数 0，但习惯上通过 define 定义为 NULL。

C 语言允许指针为零值，即 NULL 值，它作为指针的异常事件的标志。例如，动态申请内存空间失败，则返回值为 NULL。C 语言保证有效指针不会是 NULL 值。

11. 指针数组

指针数组是以数组形式构造起来的互相有一定联系的同类型指针变量的有序集合。数组中的每个元素的值是一个指针，这些指针所指向的对象类型相同。

指针数组定义形式：

```
类型标识符    *数组名[数组长度说明],……;
```

(1) 要注意指针数组及指针数组元素与指向一维数组的指针变量的区别。例如：

```
int (*p)[4], *q[4];
```

其中，p 是指向一个包含 4 个整型元素的一维数组的指针变量，q 是一个由 q[0]、q[1]、q[2]、q[3]共 4 个指向整型数据的指针组成的一维指针数组。

(2) 指针数组适合数据排序处理，尤其是字符串的排序，处理过程中不必改动需交换的

字符串的位置，只需交换指向字符串的指针即可。

(3) 字符指针数组初始化，实际上是将每个字符串的首地址传送给相应的字符指针数组元素。例如：

```
static char *name[5]={"zhao","qian","sun","li","wang"};
```

12. 多级指针(指向指针的指针)和命令行参数

(1) 不失一般性，以二级指针为例，多级指针定义形式如下：

```
类型标识符  **p;
```

相对于格式符"%s"，二级指针还需间接访问一次，而一级指针直接访问即可。例如：

```
static char *name[4]={"wang","zhang","li","chen"};
char **p; int i;
p=name;
for(i=0; i<4; i++) printf("%s\n", *p++);
```

(2) 命令行参数(指针数组作 main 函数的形参)：

在系统提示符后键入的使计算机操作系统完成一定任务的输入行称为命令行。一般形式(以 UNIX 系统为例)如下：

```
$参数0(命令名)  参数1  参数2  ……  参数n
```

其中，命令名作为命令行的元素参数 0。

C 程序 main()函数的两种形式：

① 不带参数的主函数，编译后生成一个不带参数的 shell 命令，其一般形式为：

```
main( ){……}
```

② 带参数的主函数，编译后生成一个带参数的 shell 命令，其一般形式为：

```
main( int argc, char *argv[]){……}
```

其中：

ⅰ. argc：表示命令行参数的个数(包括运行文件名，即命令名)。

ⅱ. argv：是指向命令参数的指针数组，其大小由 argc 确定。它的元素依次指向命令行中的各个字符串，包括命令名(即 argv[0]或 *argv 指向命令名)。

argc 和 argv 是习惯上使用的 main 函数的形参名，实际上，这两个参数可由用户自己命名，但它们的类型是固定的。

参数是命令功能的扩充，通过命令行参数的使用，使得可以通过为命令名指定参数的形式来提高执行程序的方便性和灵活性。

13. 返回值为指针型数据的函数

一个函数可以带回一个整数值、实数值、字符值等，也可以带回指针型的数据，即地址，其概念与以前类似，只是带回的值是指针类型而已。

返回值为指针型数据的函数(可简称为指针函数)定义形式为：

```
类型标识符  *函数名(形参数据说明表列 ){……}
```

例如：

```
int  *funl( int x, int y ) { …… }
```

在返回值为指针型数据的函数中，可通过 return 语句返回一个数据的存储单元地址或一个数据集合的存储单元的起始地址，以此达到返回一个数据或一个数据集合的目的。

14. 函数指针(指向函数的指针)

函数的指针就是指向函数的指针变量。

函数指针必须先说明后使用，其定义形式如下：

```
数据类型标识符  (*指针变量名)( );
```

例如：

```
int (*p)();
```

表示 p 是指向一个函数的指针，此函数的返回值为整型。

(1) 函数指针赋值形式：

```
指向函数的指针变量名=函数名;
```

要注意，函数指针的类型应与所指函数类型匹配一致。

(2) 函数名代表该函数的入口地址，将函数的入口地址赋给指向函数的指针变量后，就可以通过指向函数的指针变量调用函数。调用形式如下：

```
(*指针变量名)(实参表);
```

以上形式等价于“函数名(实参表)”的调用形式，只是前一个是间接调用函数方式，而后一个是直接调用方式。

(3) 指向函数的指针变量可以作为函数参数在函数间传递，此时实参可以是函数名(或指向函数的指针变量)，形参应是指向函数的指针变量。

(4) 注意指针函数与函数指针定义上的区别：

```
int (*p)();          //定义一个指向函数的指针变量 p
int *p();)           //定义一个返回值为指针型数据的函数
```

二、实验操作

(一) 目的要求

1. 掌握指针概念和指针变量的定义、赋值和运算。
2. 掌握指针在数组、字符串和函数方面的应用。
3. 掌握指针在程序设计中的作用，学会使用指针编程。
4. 理解指向指针的指针和命令行参数的概念及其使用方法。

(二) 范例

1. 设计一个函数，对 5 个字符串按由小到大排序，对字符串赋初值和输出都在 main 函

数中进行。

原程序 prg7_1.c 清单：

```
#include <stdio.h>
#include <string.h>
void main(){
  void sort(char *s[],int n);
  static char *str[5]={"basic","fortran","cobol","pascal","c"};
  int i;
  sort(str, 5);
  printf("new sequence of string is: \n");
  for(i=0; i<5; i++)
    printf("%s\n",str[i]);
}
void sort(char *s[],int n){
  char *temp; int i,j,k;
  for(i=0; i<n-1; i++){
      k=i;
      for(j=i+1; j<n; j++)
          if(strcmp(s[k],s[j])>0) k=j;
      if(k!=i){
          temp=s[i];
          s[i]=s[k];
          s[k]=temp;
      }
  }
}
```

运行结果：

```
new sequence of string is:
basic
c
cobol
fortran
pascal
```

2. 写一程序，给出一个一维数组的元素值，先后 4 次调用相关函数，分别求：

(1) 数组的各元素值之和。

(2) 下标为奇数的元素之和。

(3) 各元素的平均值。

(4) 最大元素值。

要求：

(1) 编写函数 arr_add、odd_add、arr_ave、arr_max 分别用来求数组的各元素值之和、下标为奇数的元素之和和、各元素的平均值、最大元素值。

(2) 编写函数 process 的作用是调用一个函数并输出此函数的返回值。这个函数的地址由实参传给 prodess 函数的形参 fun，fun 是指向函数的指针变量。在 main 函数中调用 process 函数时，实参有 3 个：

①数组名 a(即数组的首地址)。

②元素个数 n。

③ 在 process 函数中需要调用的函数。

源程序 prg7_2.c 清单：

```
#define  N 12
void main(){
  static float a[]={1.5,3.8,5.6,7.8,91.6,1.61,13.3,15.0,
                    17.5,19.9,21.7,23.0};
  float arr_add(float arr[], int n);              //函数原型说明
  float odd_add(float *p,int n);                  //函数原型说明
  float arr_ave(float *p,int n);                  //函数原型说明
  float arr_max(float a[], int n);                //函数原型说明
  void process(float *p,float (*fun)(),int n);//函数原型说明
  int n=N;

  printf("the sum of %d elements is:",n);
  process(a, arr_add, n);
  printf("the sum of odd elements is:");
  process(a, odd_add, n);
  printf("the average  of %d elements is:",n);
  process(a, arr_ave, n);
  printf("the maxinum of %d elements is:",n);
  process(a, arr_max, n);
  return;
}

float arr_add(float arr[], int n){
  int i;
  float sum=0;
  for(i=0; i<n; i++)
    sum=sum+arr[i];
  return(sum);
}

float odd_add(float *p,int n){
```

```
    int i; float sum=0;
    for(i=0; i<n; i=i+2,p=p+2)
      sum=sum+ *p;
    return(sum);
  }

  float arr_ave(float *p, int n){
    int i; float sum=0, ave;
    for(i=0; i<n; i++)
      sum=sum+p[i];
    ave=sum/n;
    return(ave);
  }

  float arr_max(float a[], int n){
    int i=0; float temp=a[i++];
    for( ; i<n; i++)
      if(temp<a[i])
        temp=a[i];
    return(temp);
  }

  void process(float *p,float (*fun)(),int n){
    float result;
    result=(*fun)(p,n);
    printf("%8.2f\n",result);
  }
```

运行结果：

```
the sum of 12 elements is:   222.31
the sum of odd elements is:   151.20
the average   of 12 elements is:   18.53
the maxinum of 12 elements is:   91.60
```

（三）实验内容

以下实验内容的功能要求以函数形式和用指针处理实现。

1. 上机实现范例 1 和范例 2。
2. 采用冒泡法实现对 n 个整数按递减次序的排序。
3. 采用选择法实现对 n 个整数按递减次序的排序。
4. 将数组 a 中 n 个整数按相反顺序存放，并输出对换后的结果。
5. 编一函数实现一个 $n\times n$ 的矩阵转置。在程序的主函数中用 scanf 函数输入以下矩阵元素；将数组名作为函数实参；函数调用后在主函数中输出已转置的矩阵。

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

6. 写一函数 udf_strlen()，求一个字符串的长度。在 main 函数中输入字符串，并输出其长度。

7. 有一字符串，包含 n 个字符。写一函数 udf_strcpy()，将此字符串中第 m 个字符开始的全部字符复制成为另外一个字符串。

8. 编一函数 udf_strcat()，实现两个字符串的连接(不要用系统提供的 strcat()函数)。

9. 编一函数 udf_strcmp()，实现两个字符串的比较，函数调用形式为：

strcmp(s1,s2)

如果 s1=s2，则函数返回值为 0；如果 s1≠s2，返回它们两者第一个不同字符的 ASCII 码差值(如"BOY"，与"BAD"，第二个字母不同，"O"与"A"之差为 79－65＝14)。

要求两个字符串 s1，s2 由 main 函数输入，strcmp 函数的返回值也在 main 函数输出。

10. 编制函数，其功能是在 float 类型一维数组中查找最大值、最小值并将它们返回到调用程序并输出。要求：采用形实参数结合传递的方法实现。

11. 输入 5 个字符串，用指向指针的指针方法按由小到大的顺序排序并输出。

12. 利用指针的动态申请空间方式编写一个程序，求两个向量的乘积。

13. (选做题)输入一行文字，找出其中大写字母、小写字母、空格、数字以及其他字符各有多少？

14. (选做题)编制程序功能：使用一维数组的指针访问方式实现杨辉三角形的输出，并且要求将杨辉三角形按照等腰三角形的形式输出在屏幕中央。

15. (选做题)有 n 个人围成一圈，顺序排号。从第 1 个人开始报数(从 1 到 3 报数)，凡报到 3 的人退出圈子，问最后留下的是原来第几号？

16. (选做题)用指向指针的方法对 n 个整数排序并输出。要求将排序单独写成一个函数。n 和各整数在主函数中输入。排序后的结果在主函数中输出。

17. (选做题)将一个 5×5 的矩阵中最大的元素放在中心，4 个最小的元素(顺序为从左到右，从上到下顺序依次从小到大存放)，写一函数实现之。用 main 函数调用。

18. (选做题)写一个用矩形法求定积分的通用函数，分别求：

$$\int_0^1 \sin x\mathrm{d}x,\ \int_{-1}^1 \cos x\mathrm{d}x,\ \int_0^2 \mathrm{e}^x\mathrm{d}x$$

说明：$\sin x$、$\cos x$、e^x 函数在系统的数学函数库中已有定义，头文件为"math. h"。

(四) 常见错误分析

指针是一种非常有用的工具，许多程序设计中都要用到它。但是错误地使用指针时可能会造成严重的后果或破坏。如在读操作中，错误地使用指针只是得到一些无用的存储单元内容，而在写操作中，错误地使用指针很可能改写程序代码或重要的数据，甚至改写系统文件，其后果不堪设想；若指针指向的存储区域不可访问，则通过该指针操作时将会发生运行错误。因此在使用指针时，一定要谨慎，避免出错。以下是一些常见的指针使用错误：

【错误示例 7.1】 本例的源程序将值 100 赋给一个未知其地址的存储单元，即不确定的存储单元。

```
#include <stdio.h>
main(){
  int x, *p;
  x=100;
  *p=x;            //local variable 'p' used without having been initialized
  printf("%d\n", *p);
}
```

本例程序中的指针变量 p 因为没有初始化，所以 p 就很可能指向一个“致命”区域，造成严重后果。

避免此类错误的方法是在使用指针前，先让它指向一个有效的存储空间，如以下所示：

```
#include <stdio.h>
main(){
  int x,y, *p=&y;
  x=100;
  *p=x;
  printf("%d\n", *p);
}
```

【错误示例 7.2】 本例的源程序错误地将字符串“pointer variable”复制到未知的存储单元，这是一种严重的、致命的错误，使用指针前，必须使指针 p 指向一个有效的存储空间。

```
#include <stdio.h>
#include <string.h>
main(){
  char *p;        //local variable 'p' used without having been initialized
  strcpy( p, "pointer variable");
  printf("%s\n",p);
}
```

本例所指的有效的存储空间可用两种方法获得：

(1) 通过数组定义获得。

```
#include <stdio.h>
#include <string.h>
main(){
  char s[40], *p=s;
  strcpy(p,"pointer variable");
  printf("%s\n",p);
```

```
}
```

(2) 通过动态申请存储空间获得。

```
#include <stdio.h>
#include <string.h>
#include <process.h>
#include <malloc.h>
main(){
  char *p;
  p=(char *)malloc(40*sizeof(char));
  if(p==NULL){
    printf("out of mempry\n");
    exit(1);
  }
  strcpy(p,"pointer variable");
  printf("%s\n",p);
}
```

【错误示例 7.3】 本例源程序所表现出来的问题是错误地使用未初始化的指针数组，及其编程者对指针数组使用的误解。指针数组的元素是存放存储单元地址的指针变量，不能直接存储数值。

```
#include <stdio.h>
main(){
int i;
char *str[5];
for(i=0; i<5; i++)
  scanf("%s",str[i]);        //字符指针数组不能用于直接存放字符串
for(i=0; i<5; i++)
  printf("%s\n",str[i]);
}
```

常用修改方案有二：

(1) 定义一个二维的字符数组，然后把每行的起始地址赋给指针数组的相应元素。

```
#include <stdio.h>
main(){
  int i;
  char s[5][80], *str[5];      //定义二维的字符数组 s[5][80]
  for(i=0; i<5; i++)
    str[i]=s[i];       //将 s 数组的每行起始地址赋给 str 指针数组的相应元素
```

```
  for(i=0; i<5; i++)
  scanf("%s",str[i]);
for(i=0; i<5; i++)
  printf("%s\n",str[i]);
}
```

(2) 先调用动态申请存储空间函数 malloc()为指针数组的各元素分配存储空间，然后再使用。

```
#include <stdio.h>
#include <process.h>
#include <malloc.h>
main(){
  int i;
  char *str[5];
  for(i=0;i<5;i++)            //为各元素动态申请存储空间
    if((str[i]=(char *)malloc(80*sizeof(char)))==NULL){
      printf("out of mempry\n");
      exit(1);
    }
  for(i=0; i<5; i++)
    scanf("%s",str[i]);
  for(i=0; i<5; i++)
     printf("%s\n",str[i]);
}
```

实验八　结构体和共用体

学习重点

- 结构体类型的定义
- 结构体变量的定义、存储特点与引用
- 结构体数组与应用
- 指向结构体变量或结构体数组的指针变量的定义和引用
- 结构体与函数(数据传递)
- 动态内存分配函数 malloc、calloc、free 的使用
- 单向链表的概念和结点的定义
- 链表应用(单向链表的建立、插入、删除与输出)
- 共用体和枚举类型的定义
- 共用体和枚举类型变量的定义和赋值
- 类型定义

一、要点综述

1. 结构体类型

结构体(结构变量)是不同类型的数据元素的有序集合。相当于其他高级语言的记录型,用以组织和处理复杂的数据(客观世界的实体)。

结构体类型定义的一般形式:

```
struct 结构体名{
    类型标识符 1      成员名 1;
    类型标识符 2      成员名 2;
    ......
    类型标识符 n      成员名 n;
};
```

其中,struct 是关键字,是结构体类型的标志;结构体名是由用户定义的标识符。struct 和结

构体名两者构成结构体特定类型标识符。结构体类型的组成成分称为成员，成员说明表列中的成员名应符合用户标识符的命名规则，即与变量相同。

(1) 类型定义只是说明了一个客观实体相应的属性描述：包括属性个数、所属类型、存储顺序、所占存储空间的大小等。

(2) 结构成员可以是任何基本数据类型的变量，如 int，char，float 和 double 类型等，这些成员的类型可以相同，但往往是不同的。结构成员也可以是数组、指针类型的变量。

(3) 由定义可知，其相应的结构变量的存储单元的大小为各成员所需容量的总和。

(4) 结构体类型可以嵌套定义，即允许一个结构中的一个或多个成员是其他结构类型的变量。嵌套的子结构类型及其变量的定义可以放在外层结构的成员表中，也可以采用把各个子结构类型单独定义，而在结构成员表中仅出现相应结构类型的变量说明。

(5) 结构体类型不允许递归定义，即：一个结构类型的成员中不能含有类型为本结构的变量。

(6) 在同一结构内各成员的名称不能相同。但不同结构中的成员名可以相同，并且结构的成员名可以与程序中的变量名相同。编译系统能根据上下文将它们区分开，知道它们表示不同的对象。

2. 结构体变量的定义及其初始化

结构体类型变量必须先定义后使用。

(1) 结构体类型变量定义的三种常用形式：

① 先定义结构体类型再定义相应变量。

② 在定义结构体类型的同时定义变量。

③ 利用无名结构类型定义变量，即直接定义(省略结构体名)结构体类型变量。

(2) 结构变量的初始化同数组初始化，传统的 C 规定，只有结构变量为全局变量或静态的局部结构变量才能初始化，不能对动态局部结构变量进行初始化。新的 ANSI C 和标准 C 的版本中取消了上述限制，允许对自动结构变量进行初始化，但初始化的时间是在相应函数被调用执行时进行。

结构变量的初始化就是在定义结构体变量时，将结构体变量各成员的初值顺序地组成初值表，并用逗号分隔放在一对大括号中。

对结构变量的初始化的一般形式：

```
结构类型　结构变量名={初值表};
```

3. 结构成员的引用

对结构成员的引用可以分为对结构体变量各成员的引用和对整个结构体变量的引用。

(1) 表达式中只能引用结构变量成员，不能整体引用结构变量。

对结构变量中的成员引用的一般形式：

```
结构体变量名.成员名
```

① “.”为成员运算符，“结构体变量名.成员名”相当于一个变量。

② 如果结构体变量为嵌套结构，就应使用成员运算符层层结合，一直找到最低一级的成员，即只能对最低一级的成员进行赋值或存取运算。

③ 结构体成员都属于某种数据类型，可以像同类变量一样进行各种操作。

④ 可以引用成员的地址，也可以引用结构变量的地址。

(2) 直接使用结构变量名，是对整个结构体变量的引用。相同类型的结构体变量之间可以进行整体赋值，此时允许直接使用结构变量名。其他操作如输入、输出等必须通过引用结构体变量的成员进行相应的操作。

4. 结构体数组

结构数组是一个数组，其元素的数据类型为结构类型。

(1) 定义结构体数组的方法和定义其他类型数组的方法类似，只是数据类型不同而已。

(2) 对结构体数组的初始化时，可以根据结构体数组及其结构体数组元素的成员存储有序的规律，将数组每个元素的初值(元素是一个结构变量，其初值的写法参见结构变量的初始化)都放在一对大括号中。如果给出了全部元素的初值，则数组的长度可以不指定，由系统根据初值的数目来确定数组的长度。

(3) 对结构体数组的引用一般是对数组元素成员的引用。引用只要遵循对数组元素的引用规则和对结构体变量成员的引用规则即可，一般形式如下：

```
结构体数组名[下标].成员名
```

5. 指向结构体变量的指针变量

指向结构体变量的指针变量的定义和赋值与一般指针变量的定义和赋值相同，只是其所指存储单元的数据类型为结构体类型，它只能赋予定义时规定的所指类型的结构变量或结构体数组元素的地址以及同类型的指针变量值。

若一个结构体指针变量指向一 3 个结构体变量，则对该结构体变量成员的引用方式有如下三种形式：

① 根据结构体变量名使用成员运算符引用成员。形式如：

```
结构体变量名.成员名
```

② 利用指向该结构体的指针变量，使用成员运算符引用成员。形式如：

```
(*结构体指针变量名).成员名
```

其中，“(*结构体指针变量名)”的圆括号是必需的。

③ 利用指向该结构体的指针变量，使用指向成员运算符引用成员。形式如：

```
结构体指针变量名->成员名
```

其中，“->”为指向成员运算符，它由符号“-”和“>”构成。指向成员运算符形式直观，使用方便。

在 C 语言中，结构的成员运算符“.”和指向成员运算符“->”，以及表示函数的运算符“()”、表示数组下标的运算符“[]”，都具有最高的优先级。当->和++(或--)同时出现在一个表达式中时，应注意++(或--)所作用的对象是谁。

设有：

```
struct date{ int  year; int month; int day }d[5], *pd; pd=d;
```

则有：

• pd->year 表示 pd 所指结构数组元素成员 year 的值。

• ++ pd->year 等价于++(pd->year),是把 pd 所指结构数组元素成员 year 的值加 1,并不是 pd 加 1。

• pd->year++等价于(pd->year)++,是把 pd 所指结构数组元素成员 year 的值加 1,属于后缀表达式。

• (++ pd)->year 是把 pd 的值先加 1,使它指向下一个(后继)结构数组元素,然后取得 pd 所新指向结构数组元素成员 year 的值。

• pd++->year 是先取出 pd 当前所指向结构数组元素成员 year 的值,然后 pd 加 1,指向下一个(后继)结构数组元素。

如果 p 是指向结构变量的指针,xp 是该结构中的一个指针变量,则有:

• *p->xp 等价于 *(p->xp),表示间接访问 p 所指结构变量中 xp 所指的对象。

• *p->xp++等价于 *((p->xp)++),表示先间接访问 p 所指结构变量中 xp 所指对象的内容,然后该成员 xp 的值加 1。

• *p++->xp 表示先间接访问 p 所指结构变量中 xp 所指对象的内容,然后 p 加 1,指向下一个(后继)结构变量。

实际使用中类似的表达式还很多,关键是注意掌握运算符的优先级、结合性以及++(或--)前缀和后缀形式的功能差别。

6. 指向结构体数组的指针变量

指针变量也可以指向结构体类型的数组,此时它所指的类型应与结构体数组的类型一致,可以将数组名或其数组元素地址赋给该指针,这样通过该指针就可以引用所指结构体数组的元素,其中利用指向成员运算符的引用形式较常用。

注意:若指针 p 是指向结构的指针,则对 p 所施加的运算均是按结构体的实际长度计算的,因此 p++总是指向结构数组的下一个元素,而不是指向某一元素中的某一成员。

7. 结构体数据作函数参数

(1) 早期的 C 语言版本中不允许用结构变量作为函数的参数,解决结构变量传递的办法是:

① 用结构变量的成员作参数,属"传值"方式,具体应用规则与同类型变量相同。

② 利用取地址运算符"&"取结构变量的地址,以指向结构体变量(或结构体数组)的指针作实参,将结构体变量(或结构体数组)的地址传给形参,属"传地址"方式。

③ 函数的返回值也可以是一个指向结构的指针。

根据 ANSI C 的新标准规定,不仅允许使用指向结构体变量的指针作为函数的参数,还允许直接使用结构体变量作为函数的参数,但此时为"传值"方式,系统将实参结构变量相应的内存单元内容全部顺序传递(复制)给形参,即传递的是结构体变量全部成员的值。形参也必须是同类型的结构变量。

(2) 指向结构体的指针作函数参数时,函数的实参和形参是指向同一种结构体类型的指针变量(或同一种结构体类型的地址),地址传递的规则不变,即形参与实参共享存储空间,当形参指向的结构体变量的值发生变化后,对应的实参指向的结构体变量的值也同样发生变化。

8. 动态存储分配

在C语言中，数组和结构的大小都必须是固定的，不支持如动态数组和动态结构之类的类型。但在实际编程中，往往会遇到这种情况：所需内存空间的大小无法预先确定下来，要根据实际得到数据的多少来决定。解决这类问题的办法是：动态分配存储空间。动态分配存储空间必须通过指针，并利用C编译程序提供的内存动态分配函数实现。

(1) 常用到的内存管理函数有：malloc()，calloc()和free()。

① malloc()函数原型如下：

```
void  *malloc( unsigned size );
```

功能：分配size字节的连续存储区，返回值为所分配内存区的起始地址，如不成功，则返回0。

例如：在内存中需要开辟一个存放4个double型数据的32个字节空间。并让定义为double *pd;的指针pd指向它，则可以采用如下形式语句实现：

```
pd=(double *)malloc(4*sizeof(double));
```

② calloc()函数原型如下：

```
void *calloc ( unsigned n, unsigned size );
```

功能：分配n个数据项的内存连续空间，每个数据项的大小为size。返回值为所分配内存单元的起始地址，并将所分配的各个单元全部清零。如不成功，则返回0。例如，在内存中需要开辟一个存放4个double型数据的空间。并让定义为double *pd;的指针pd指向它，则可以采用如下形式语句实现：

```
pd=(double *)calloc(4,sizeof(double));
```

③ free()函数原型如下：

```
void free( void *p );
```

功能：释放指针变量p所指的由动态函数分配的存储空间，返回值无。

(2) 说明：ANSI标准要求动态分配系统返回void指针。void指针具有一般性，它可以指向任何类型的数据，但目前大多数C编译所提供的这类函数返回的是char指针。无论上述情况中的哪一种，都需要用强制类型转换的方法把它们转换为所需要的类型。

9. 链表

(1) 链表的基本特征：

① 有一个“头指针”变量(一般取名为head)，用来指向链表的第一个元素。

② 链表中的每一个元素称为“结点”。结点包含两部分内容：一是实际的数据信息；二是指向下一个结点的指针。

③ 链表的最后一个元素称为“表尾结点”，它不指向其他结点，因此将它的指针部分置为“NULL”，标志链表的结束。

④ 链表中，一个结点的后续结点是由该结点包含的指针指出的，与后续结点在内存中的位置无关，也就是链表的各个结点在内存中可以不连续存放。但如果要查找该链表中某个结点，则必须由该链表头指针所指的第一个结点开始，顺序查找。

⑤ 单向链表中的结点可以用一个结构体类型来描述。在结构体类型定义中不能包含自身，即不允许递归定义。因为如果一个结构体类型定义中包含有本类型的结构变量，那么就无法确定该结构体类型的大小。但是在一个结构体定义中可以包含指向该结构体类型对

象的指针变量。由于存放指针变量的存储单元的字节数是可以确定的，所以这种类型的结构体变量所占用的字节数也是可以确定的。这种结构一般称为引用自身的结构。例如：

```
struct LNode{
    long        sno;
    float       score;
    struct LNode *next;
};
```

(2) 链表与结构体数组的异同：

链表与结构体数组的相似之处是：

① 都由若干相同类型的结构体变量组成。

② 结构体变量间有一定顺序关系。

链表与结构体数组的区别：

① 结构体数组中各元素是连续存放的，而链表中的结点可以不连续存放。

② 结构体数组元素可通过下标运算或相应指针变量的“移动”进行顺序或随机访问；而链表中结点不便于随机访问，只能从链头开始一个结点一个结点地顺序访问。

③ 结构体数组在定义时就确定其元素的个数，不能动态增长；而链表的长度往往是不确定的，根据问题求解过程中的实际需要，动态地创建结点并为其分配存储空间。

(3)单向链表的基本操作主要有：链表的建立、链表的输出、链表的插入、链表的删除。

(4)链表的建立：

在建立链表之前必须定义该链表要使用的结构体类型，并要定义一个头指针变量 head。由于链表最初是空的，所以头指针变量 head 应初始化为 NULL。为了在链表上添加新的结点，还需要定义一个附加指针变量(设为 p1)，它所指向对象也是结点的类型。

设结点的类型如下：

```
#define STUDENT struct student
                                //定义 STUDENT 常量标识符作为结构体类型名
#define LEN sizeof(STUDENT)
                                //定义 LEN 符号常数作为结点所需空间的大小
STUDENT{                        //定义结构体(结点)的类型
    long num;                   //信息成员 1——学号
    float score;                //信息成员 2——成绩
    STUDENT *next;              //指向下一个结点的指针成员
};
```

一个新结点要加入到该链表中，可根据头指针变量 head 是否为 NULL 分为两种状态：若 head 为 NULL，则表示该链表是空的，此时加入到该链表中的新结点成为链表中唯一的成员；若 head 不是 NULL，那么该链表就已经包含一个或多个成员。无论是哪一种情况，将一个新的成员添加到该链表的开头所要完成的动作都是相同的。

① 调用 malloc()函数，申请一个结点的内存空间，创建一个新的结点实体(结构体变量)。

② 将新结点的 next 指针变量置为 head 指针变量的当前值。如果是新建的链表，即链表为空，那么 head 的当前值为 NULL；否则，head 的当前值是目前链表中第一个成员的地址。

③ 使 head 指向新成员。

在 head 已初始化为 NULL 的前提下，完成这一任务可由以下代码实现：

```
p1=(STUDENT *)malloc(LEN);      //动态申请存储空间
p1->next=head;                  //新结点的 next 置为 head 的当前值
head=p1;                        //使 head 指向新成员
```

以上介绍的属于表头插入算法，链表的建立还可以通过表尾插入算法和在链表中查找定位插入算法等，详见实验范例 2。

(5) 从链表中删除结点：

从一个链表中删除结点并不是真正从内存中把它删除，而是把它从链表中分离出去，即只需要改变链表中结点的关系。利用指针实现从链表中删除结点是很简单的事情，具体的过程取决于被删除结点在链表中处于什么位置，可以归纳为以下三种情况：

① 删除链表的第一个结点，此时只要把 head 指向链表的第二个结点。示例代码如下：

```
p1=head;                        // p1 指向链表的第一个结点
head=p1->next;                  // head 指向链表的第二个结点
free(p1);                       // 释放 p1 所指结点的内存空间
```

② 删除链表的尾结点，此时需要通过查找定位链表尾结点，并把链表尾结点的前驱结点中的 next 置为 NULL。该操作需要两个辅助指针变量。示例代码如下：

```
STUDENT *p1,*p2;                //定义两个辅助指针变量
p1=head;                        // p1 指向链表的第一个结点
while(p1->next!=NULL){          //查找定位被删尾结点
    p2=p1;p1=p1->next;          //指针一前一后地下移
}
p2->next=p1->next;              //把表尾结点的前驱结点的 next 置为 NULL
free(p1);                       //释放 p1 所指结点的内存空间
```

③ 删除链表其他位置的任一结点，此时需要通过查找定位被删结点，并使该结点的前驱结点中的 next 指向被删结点之后的结点(称后继结点)。该操作类似删除链表的尾结点也需要两个辅助指针变量，只是查找定位条件不同。示例代码如下：

```
STUDENT *p1,*p2;                //定义两个辅助指针变量
p1=head;                        // p1 指向链表的第一个结点
while(num!=p1->num){            //查找定位被删尾结点，num 为被删结点学号
    p2=p1;p1=p1->next;          //指针一前一后地下移
}
p2->next=p1->next;              //被删结点的前驱成员 next 指向被删结点的后继结点
free(p1);                       //释放 p1 所指结点的内存空间
```

为了给程序腾出更多的内存空间，应该利用 free()函数释放被删结点所占用的内存空间。

10. 共用体(又称联合)类型

共用体类型定义的一般形式：

```
union 共用体名{
    类型名 1    成员名 1;
    类型名 2    成员名 2;
    ……
    类型名 n    成员名 n;
};
```

其中，union 是关键字，是共用体类型的标志。共用体名是由用户定义的标识符。union 与共用体名两者构成共用体类型标识符，它规定了所定义的共用体类型。共用体类型也由若干成员组成。

(1) 共用体类型变量的定义也可由三种方法：

① 先定义共用体类型再定义变量。

② 在定义共用体类型的同时定义变量。

③ 直接定义共用体类型变量(省略共用体名)。

(2) 共用体变量的所有成员共同占用一段内存区域，所以共用体变量所占字节数是其成员中占内存空间最大的成员的字节数。由于共用体类型采用的是覆盖技术，因此共用体变量中起作用的总是最后一次存放的成员变量的值。

(3) 共用体变量也必须先定义后使用。不能直接引用共用体变量，只能引用共用体变量的成员。引用的一般形式：

```
共用体变量名.成员名
```

共用体变量的每个成员也可以像同类型变量一样进行其类型允许的各种操作。

(4) 可以把共用体变量的地址赋给同类型的指针变量。因此也可以利用指向共用体变量的指针变量引用其成员：

```
指针变量名->成员名      或 (*指针变量).成员名
```

(5) 对于共用体变量也可以初始化，但只是对成员表中列举的第一个成员进行初始化，不能对其所有列出的成员都赋初值。

11. 枚举类型

枚举类型定义的一般形式为：

```
enum   枚举名{枚举元素 1,枚举元素 2,……,枚举元素};
```

其中，enum 是关键字，是枚举类型的标志。枚举名是由用户定义的标识符。enum 和枚举名两者构成特定的枚举类型。花括号中各个标识符是构成该类型的各个成分，即枚举元素。

(1) 枚举类型中的枚举元素是用户定义的标识符，作为常量处理，称为枚举常量。因此不能对枚举常量赋值。

(2) 枚举元素被处理成一个整型常量，它的值取决于定义时各枚举元素排列的先后顺序，称为序号值。第一个枚举元素的值为0，第二个为1，依次顺序加1。也可以在定义时改变枚举元素的值，没有指定值的枚举元素，其值为前一元素值加1。

(3) 枚举类型变量的定义有两种方法：一是先定义枚举类型，再定义变量；二是直接定义枚举变量。

(4) 枚举元素可以做比较运算：比较的依据是各枚举元素的序号值。

(5) 只能给枚举变量赋枚举常量。不能直接给枚举变量赋整型值，但是可以将整型值通过强制类型转换成枚举类型再赋值。枚举变量只能通过赋值语句得到值，并且通过输出语句输出时得到的是枚举序号值。

12. 类型定义

类型定义是C语言中用来对已经存在的数据类型定义一个新的名字。定义新类型名的一般形式为：

```
typedef    类型名    标识符;
```

其中，类型名是已经定义了的数据类型名，标识符是用户定义的新的类型名(称用户自定义类型)。

(1) 基本数据类型、数组类型、结构体类型、指针类型定义成用户自定义类型。

① 基本数据类型的自定义：

```
typedef  基本类型名  用户自定义类型名;
```

② 数组类型的自定义：

```
typedef  类型说明符  用户类型名[数组长度];
```

③ 结构体类型的自定义：

```
typedef struct{
    数据类型1  成员1;
    数据类型1  成员1;
    ......
    数据类型1  成员1;
}用户自定义类型名;
```

④ 指针类型定义的自定义：

```
typedef  类型说明符  *用户自定义类型名;
```

(2) 使用用户自定义类型来定义变量、数组、指针。

① 定义基本数据类型的变量：

```
用户自定义类型名    变量名表列;
```

② 定义数组：

```
用户自定义类型名    数组名;
```

注意:数组名后不需要跟数组下标运算符和指定数组的大小。

③ 定义指针变量:

用户自定义类型名　　指针变量名表列;

注意:指针变量名前不需要加“*”。

(3) 使用 typedef 并没有创造新的类型,而只是为已经存在的类型多取一个名字而已。

二、实验操作

(一) 目的要求

1. 掌握结构体类型、结构体变量的定义和使用。
2. 掌握结构体数组的定义、使用及其应用编程。
3. 学会使用结构体指针变量的定义和使用。
4. 掌握链表的概念,学会对链表进行操作。
5. 了解共用体的概念与使用。

(二) 范例

1. 打印 1994 年出生的学生人数,并统计全班男女生各多少。

程序 prg8_1.c 清单:

```
#include "stdio.h"
#define N 5
struct student{
    char sex,name[16];
    int birthyear;
};

void main(){
  struct student class[N];
  void input(struct student *p, int n);      //函数原型说明
  void count(struct student s[], int n,int *c1994,int *cm,int *cf);
  int i,count1994=0,countm=0,countf=0;
  input(class,N);
  count(class, N, &count1994, &countm, &countf);
  printf("male: %d\n",countm);
  printf("female: %d\n",countf);
  printf("birthyear=1994: %d\n",count1994);
}

void input(struct student *p, int n){
  struct student *pend;
```

```
    pend=p+n-1;
    printf("NAME,SEX[m/f],BIRTHYEAR\N\n");
    for( ;p<=pend; p++){
        scanf("%s %c%d",p->name,&p->sex,&p->birthyear);
        getchar();
    }
}

void count(struct student s[], int n,int *c1994,int *cm,int *cf){
    int i;
    for(i=0 ;i<n; i++){
        if(s[i].birthyear==1994)
          ++*c1994;
        if(s[i].sex=='m')
          ++*cm;
        else
          ++*cf;
    }
}
```

运行结果：

```
NAME,SEX[m/f],BIRTHYEARN
liuhao f1994↙
liu m1994↙
zhang f1993↙
wang f1995↙
zhao m1994↙

male: 2
female: 3
birthyear=1994: 3
```

2. 链表的建立、链表的输出、链表的插入、链表的删除操作示例。

程序 prg8_2.c 清单：

```
#include <stdio.h>
#include <malloc.h>
#define STUDENT struct student
#define LEN sizeof(STUDENT)
STUDENT{                          //定义结构体(结点)的类型
    long num;
    float score;
```

```
    STUDENT *next;
};
STUDENT *create( ){                    //建立链表函数,表头插入算法
  //p1 用于申请存储空间,为后续操作作准备
  STUDENT *head, *p1;
  p1=(STUDENT *)malloc(LEN);           //动态申请存储空间
  scanf("%ld%f",&p1->num,&p1->score);
  head=NULL;                           //初始化链表为空
  while(p1->num!=0){                   //为链表添加新成员
    p1->next=head;                     //新结点的 next 置为 head 的当前值
    head=p1;                           //使 head 指向新成员
    p1=(STUDENT *)malloc(LEN);
    scanf("%ld%f",&p1->num,&p1->score);
  }
  free(p1);
  return(head);                        //返回链表头指针
}
void print(STUDENT *head){             //遍历打印链表
  STUDENT *p;
  p=head;
  if(head != NULL)
  do{
    printf("%ld %f\n",p->num,p->score);
    p=p->next;
  }while(p != NULL);
}
STUDENT *del(STUDENT *head,long num){          //删除指定节点
  STUDENT *p1, *p2;
  if(head == NULL){                    //链表判空处理
    printf("\nlist null! \n"); return(head);
  }
  p1=head;
  while(num!=p1->num && p1->next!=NULL){   //查找定位被删结点
    p2=p1;p1=p1->next;
  }
  if(num == p1->num){
    if(p1 == head)
    head=p1->next;                     //删除表头元素
  else
```

```
      p2->next=p1->next;                    //删除中间或尾元素
    printf("delete:%ld\n",num);
    free(p1);                               //释放 p1 所指空间
    }
    else
      printf("%ld not been found\n",num);
    return(head);
  }
  STUDENT *insert(STUDENT *head,STUDENT *stud){
    //按顺序插入元素算法,也可用于建立有序链表。
    STUDENT *p0,*p1,*p2;
    p1=head;  p0=stud;
    if(head==NULL){
      head=p0;                              //如果链表为空,则建立首结点
      p0->next=NULL;
    }
    else{
      while((p0->num>p1->num) && (p1->next!=NULL)){
                                                                    //查找定位
        p2=p1;
        p1=p1->next;
      }
      if(p0->num<=p1->num){
        if(head==p1){                       //表头插入
          head=p0; p0->next=p1;
        }
        else{                               //中间插入
          p2->next=p0; p0->next=p1;
        }
      }
      else{                                 //表尾插入
        p1->next=p0;p0->next=NULL;
      }
    }
    return(head);
  }
  void main(){                              //主函数起主控的作用
    STUDENT *head,*s;  long del_num;
    head=create();  print(head);            //建立链表并遍历打印
```

```
    printf("\ninput the deleted number:");
    scanf("%ld",&del_num);
    while(del_num!=0){                          //可按学号连续删除,学号为0结束
      head=del(head,del_num);                   //删除指定结点
      print(head);                              //遍历打印检测运行状态
      printf("\ninput the deleted number:");
      scanf("%ld",&del_num);
    }
    printf("\ninput the insert record:");
    s=(STUDENT *)malloc(LEN);                   //申请插入结点的存储空间
    scanf("%ld%f",&s->num,&s->score);
    while(s->num!=0){                           //可连续插入节点,学号为0结束
      head=insert(head,s);
      print(head);
      printf("\ninput the insert record:");
      s=(STUDENT *)malloc(LEN);
      scanf("%ld%f",&s->num,&s->score);
    }
  }
```

运行结果：

```
98101 98
98103 87
98105 77
0 0
98105 77.000000
98103 87.000000
98101 98.000000

input the deleted number:98103
delete:98103
98105 77.000000
98101 98.000000

input the deleted number:0

input the insert record:98104 88
98104 88.000000
98105 77.000000
98101 98.000000

input the insert record:98106 95
```

```
98104 88.000000
98105 77.000000
98101 98.000000
98106 95.000000

input the insert record:0 0
Press any key to continue
```

3. 产生一个具有 n 个整数值(从 1 到 n)的链表,遍历整个链表并删除能被 3 整除的结点,最后再显示剩下的结点值。

源程序 prg8_3.c 清单:

```
#include <stdio.h>
#define NEW(PP)   (PP=(PITEM)malloc(sizeof(ITEM)))
#define TEST(PP) if(NEW(PP)==NULL){printf("memory error\n");return;}
typedef struct item {
    int val;
    struct item *next;
}ITEM, *PITEM;
main(){
  int i,n;
  PITEM current,first,prev;
  printf("enter n: "); scanf("%d",&n); printf("\n");
  if(n<0){
      printf("\nenter error! \n");
      return;
  }
  TEST(first)                          //用定义的宏分配内存
  current=first;                       //建立第一个结点指针
  first->val=1;                        //建立第一个结点里的值域
  for(i=2; i<=n; i++){
      TEST(current->next);
      current=current->next;           //指向下一个结点
      current->val=i;                  //给结点赋值
  }
  current->next=NULL;                  //表尾处理
  prev=current=first;                  //从表头开始查找删除结点
  for(prev; current!=NULL; prev=current,current=current->next)
    //删除能被 3 整除的的数
    if(current->val%3==0){
        prev->next=current->next;      //删除结点
```

```
            free(current);                    //释放删除结点的内存
            current=prev;
        }
    for(current=first; current!=NULL; current=current->next)
        printf("%4d",current->val);           //打印剩余结点
    printf("\n");
}
```

运行结果：

```
enter n: 8
1  2  4  5  7  8
```

4.（此题供参考）联合体的应用举例：设某单位由 N 名职工参加计算机办公应用职业技能考评。设每个人的数据包括准考证号、姓名、年龄和成绩。单位规定参加考评的年龄为 25 岁以下的职工应进行笔试，笔试成绩为百分制，60 分以上及格。而 25 岁以上的职工进行操作考核，考核成绩为 A、B、C、D 四个等级，C 级以上为及格。请编程统计考核及格的人数，并输出每位考生的准考证号、姓名、年龄和成绩。

分析：可利用在结构体类型中定义联合体类型的数据定义方法来进行考生的处理。首先定义一个结构体数组 person[N]，它包括的成员项有准考证号 num，姓名 name，年龄 age 和成绩 score，而应将成绩项 core 定义为联合体类型，当年龄小于 25 岁的考生，其考试成绩为整型成员项 exam，当为 25 岁以上的考生，其考核成绩为字符型成员项 grade。

源程序 8_3.c 清单：

```
#include <stdio.h>
#define N 5
struct{
    long num;
    char name[16];
    int age;
    union{
        int exam;
        char grade;
    }score;
}person[N];
void main(){
  int count=0,i;
  for(i=0; i<N; i++){
     scanf("%ld%s%d",&person[i].num,
       person[i].name,&person[i].age);
       printf("%s %d\n",person[i].name,person[i].age);
       if(person[i].age<25)
```

```
            scanf("%d",&person[i].score.exam);
        else
            if(person[i].age>=25)
                scanf("%c",&person[i].score.grade);
            else
                printf("input error!");
        if((person[i].score.exam>=60) || (person[i].score.grade!='D'))
            count+=1;
    }
    printf("\nNo   Name      Age    Exam/Grade\n");
    for(i=0; i<N; i++){
        if(person[i].age<25)
            printf("%ld %s %d %d\n",person[i].num,person[i].name,
                person[i].age,person[i].score.exam);
        else
            printf("%ld %s %d %c\n",person[i].num,person[i].name,
                person[i].age,person[i].score.grade);
    }
    printf("\ncount=%d\n",count);
}
```

运行结果：

```
001zhao 24 ↙ 85
002qian 25D↙
003sun 30b↙
004li 23 90↙
005wang 28D↙

No   Name      Age    Exam/Grade
1 zhao 24 85
2 qian 25 D
3 sun 30 b
4 li 23 90
5 wang 28 D

count=3
```

（三）实验内容

1. 定义一个图书相关信息的结构体类型和结构变量，其中包括成员书号、书名、作者、

出版社和价格；从键盘输入 10 本图书信息，计算并输出这 10 本图书的平均价格。

2. 在第 1 题定义的结构体类型中增加一个成员出版日期，该日期是一个嵌套的结构类型变量，其中包括年、月、日；设计一个输入输出图书信息的函数 read 和 print；并写主函数定义一个 10 个元素的结构数组，分别调用输入输出函数输入和输出图书信息。

3. 在第 2 题的基础上，曾加一个按书号递增排序的排序函数 sort，在主函数中调用排序函数后再输出图书信息。

4. 根据以下学生情况表(见表 3.9)，编制一个 C 程序，分别应用选择法和冒泡法对该学生情况表按成绩(grade)从低到高进行排序处理并输出。

表 3.9 学生情况表

学号(num)	姓名(name)	性别(sex)	年龄(age)	成绩(grade)
101	zhang	M	19	95.5
102	wang	F	18	92.0
103	zhao	M	19	85.5
104	li	M	20	96.0
105	gao	F	17	91.0

具体要求如下：

(1) 程序中，输入、输出和两种排序功能模块分别用自定义函数形式实现。

(2) 在主函数中定义一个结构体数组和结构体指针数组，其中结构体数组用以存放以上学生情况表的信息，而结构体指针数组中的每一个指针元素分别指向结构体数组的各元素，利用指针数组实现排序，可避免结构体数组元素间的交换移动，节省排序时间。

(3) 在主函数中通过调用输入、输出和两种排序功能函数，实现首先输入数据，然后输出排序前的学生情况，再输出排序后的学生情况结果，其格式如表 3.9 所示(不要求输出表格线)。

5. 链表基本操作。

已知每个学生情况的结点结构类型为：

```
struct stu{
    int num;
    char name[16];
    char sex;
    int age
    double grade;
    struct stu *next;
};
```

要求：

(1) 初始时链表为空，即链表的头指针为空。

(2) 根据上题给出的学生信息表，依次将每个学生的情况作为一个结点插入到单链表

的链头(即当前插入的结点将成为第一个结点)。

(3) 所有学生情况都插入到链表中后,从链头开始,依次输出链表中的各结点值(即每个学生的情况),格式如表所示(不要求输出表格线)。

(4) 输入一个年龄,如果链表中所包含的年龄等于此年龄,则将此结点删去;再从链头开始,依次输出链表中的各结点值(即剩余结点的每个学生的情况),格式要求同上。

其中要求(2)、(3)和(4)分别用自定义函数形式实现。

6. 13个人围成一圈,从第1个人开始顺序报号1,2,3。凡报到"3"者退出圈子。编程找到最后留在圈子中的人原来的序号。

7.(选做题)建立一个链表,每个节点包括:书号、书名、作者和出版社,并编写按书号查询和删除节点的函数。

8.(选做题) 已知工资单结构定义:

```
struct salsry{
  char name[8];          // 姓名
  double salary;         // 工资
}pay[10]={……};           // 初始化,编程时自己填入
```

要求编写程序,实现按姓名或工资进行检索,检索函数功能要求如下:

namesearch() :按姓名检索,找到则打印姓名和工资,找不到打印"无此人"。

salarysearch():按工资检索,找到则打印姓名和工资,找不到打印"无此工资"。

并要求定义如下结构:

```
struct{
  char c;                // 对话时输入字符
  int ( * func)();       // 函数指针,即检索函数
}call[]={{'n',namesearch},{'s',salarysearch}};
```

程序开始时要进行对话,询问按姓名还是按工资检索,若按姓名检索,打入"n",若按工资检索打入"s"。试编程实现。

9.(选做题)用单链表结构实现直接选择排序,结点结构同第5题。

(四) 常见错误分析

结构变量所需的存储空间是其各成员所需空间的总和。程序中一般通过采用sizeof运算符计算获得特定结构体所需空间的大小。为加强对此概念的理解,给出以下源程序予以说明。

【错误示例8.1】 定义一个存储关于学生信息的结构类型,并求其相应结构变量所需存储空间的大小。

```
#include <stdio.h>
#define STUDENT struct student
STUDENT{
  char   num[11];
  char   name[16];
```

```
    int    age;
    char   sex;
    float grade;
};
main(){
    printf("struct student 结构体变量所需空间的大小为:%-d\n",
            sizeof(STUDENT));
}
```

运行结果为(32 位机):

struct student 结构体变量所需空间的大小为:40

结构体数组和结构体指针数组两者容易混淆,一种常见的错误是在实际应用过程中,没有正确理解两者之间的主要功用和区别。当程序中同时定义了结构体数组和结构体指针数组时,常发生两者之间的误用,且并不易察觉。

【错误示例 8.2】 以下源程序采用结构体数组表示给定的学生信息表,并要求采用选择法对该学生信息表按成绩(grade)从高到低进行排序。

```
#include <stdio.h>
#define STUDENT struct student
STUDENT{
    char  num[12];
    char  name[16];
    char  sex;
    int   age;
    float grade;
};
void sort(STUDENT *p[],STUDENT s[],int n){
    int i,j,k; STUDENT *t;
    for(i=0;i<n-1;i++){
        k=i;
        for(j=i+1;j<n;j++)
         if(s[k].grade<s[j].grade) k=j;          //误用比较对象
        if(k!=i){
            t=p[i];p[i]=p[k];p[k]=t;
        }
    }
}
void print(STUDENT *p[],int n){
    int i;
    for(i=0;i<n;i++)
```

```
        printf("%10s %16s %2c %4d %6.2f\n",
            p[i]->num,p[i]->name,p[i]->sex,p[i]->age,p[i]->grade);
    }
    main(){
      STUDENT s[5]={{"pb05111001","zhang','M',19,95.0},
                    {"pb05111002","wang","F',18,92.0},
                    {"pb05111003","zhao','M',19,85.5},
                    {"pb05111004","li","M',20,96.0,},
                    {"pb05111005","chen","F',17,91.0,}};
      STUDENT *ps[5]; int i;
      for(i=0;i<5;i++)ps[i]=&s[i];
      sort(ps,s,5);
      print(ps,5);
    }
```

以上程序中，注释处存在结构体数组和结构体指针数组误用现象，具体请读者自行分析。

正确排序函数应该如下所示：

```
void sort(STUDENT *p[],STUDENT s[],int n){
  int i,j,k; STUDENT *t;
  for(i=0;i<n-1;i++){
    k=i;
    for(j=i+1;j<n;j++)
      if(p[k]->grade<p[j]->grade)k=j;
    if(k!=i){
      t=p[i];p[i]=p[k];p[k]=t;
    }
  }
}
```

采用此排序函数，可得到正确的运行结果：

```
pb05111004          li  M  20  96.00
pb05111001       zhang  M  19  95.00
pb05111002        wang  F  18  92.00
pb05111005        chen  F  17  91.00
pb05111003        zhao  M  19  85.50
```

若 main 函数中的调用语句为：

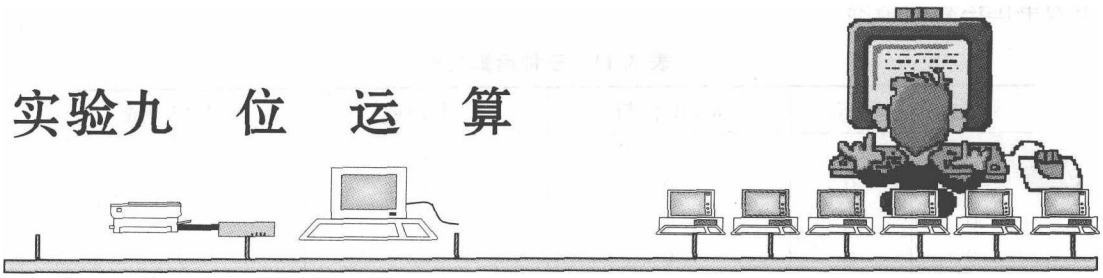
```
sort(ps,5);
```

则正确的相应被调排序函数 sort 可如下所示：

```
void sort(STUDENT *p[],int n){
```

```
    int i,j,k; STUDENT *t;
    for(i=0;i<n-1;i++){
      k=i;
      for(j=i+1;j<n;j++)
        if(p[k]->grade<p[j]->grade)k=j;
      if(k!=i){
        t=p[i];p[i]=p[k];p[k]=t;
      }
    }
  }
```

实验九　位　运　算



学习重点

- 位运算
- 位移操作

一、要点综述

位运算是对操作数以二进制位(bit)为单位进行数据加工。C 语言具有的位运算功能有:位运算(四种);位移操作(两种)两大类。注意:参加位操作的操作数必须是整型常量或变量。

1. 位逻辑运算

C 语言提供了 4 种位逻辑运算功能,如表 3.10 所示。

表 3.10　位逻辑运算种类

运算符	名　称	例　子	功　能
~	位反	~ b	求 b 的位反
&	位与	b & c	求 b 和 c 的位与
|	位或	b | c	求 b 和 c 的位或
^	位异或	b ^ c	求 b 和 c 的位异或

2. 位反运算

位反运算是单目运算,它对一个操作数按位求反,即操作数的各个二进制位为 1 时变换为 0,为 0 时变换成 1。注意:对一个操作数的位求反运算并不改变操作数本身的数值。

3. 位与、位或和位异或运算

位与、位或和位异或是对两个操作数的对应二进制位分别进行相应运算。如表 3.11 所

示为单位运算的结果。

表 3.11 三种运算关系

a位	b位	a &b位与	a|b位或	a ^ b位异或
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0

4. 移位操作

移位操作是对操作数以二进制位(bit)为单位进行左移或右移。移位操作包括两种运算:左移(≪)和右移(≫),它们是双目运算符。

移位操作表达式形式:

〈操作数〉 移位运算符 〈移动的位数〉

5. 左移位操作

(1) 左移位操作中,操作数移位后,右端出现的空格补 0,而移至左端之外的位舍去,不管其是否为符号位。

(2) 每向左移一位,相当于该数乘 2。

(3) 左移位操作不改变操作数的值。例如,对变量 a 进行左移位操作"a≪1"后,a 的值并不发生改变。

6. 右移位操作

(1) 右移位操作与操作数的数据类型是否带有符号有关:

① 不带符号的操作数右移位时,左端出现的空位补 0。

② 带符号的操作数右移位时,左端出现的空位按原最左端的位复制。

无论什么操作数,移出右端的位被舍弃。

(2) 每右移一位相当于操作数除 2。

(3) 右移位操作不改变操作数的值。

7. 位自反赋值运算(复合位运算)

与算数运算符类似,位运算符可和赋值运算符一起构成位自反赋值运算符,如表 3.12 所示。

表 3.12 位自反赋值运算符

运算符	名 称	例 子	等价于
&=	位与赋值	a&=b	a=a&b
|=	位或赋值	a|=b	a=a|b
^=	位异或赋值	a^=b	a=a^b
≫=	右移赋值	a≫=b	a=a≫b
≪=	左移赋值	a≪=b	a=a≪b

位自反赋值运算的过程是,两个操作数进行位操作,然后把结果赋予第一个操作数。为

此第一个操作数必须是变量，且其值将被改变。

8. 位段

位段就是以位(bit)为单位定义长度的结构体类型成员。

位段结构类型一般定义形式为：

```
struct 结构名{
    类型标识符 1     位段名 1:位数 1;
    类型标识符 2     位段名 2:位数 2;
    ......
    类型标识符 n     位段名 n:位数 n;
};
```

(1) 具体使用中应注意成员所占的位数，即长度，以及其由长度限定的存取值域。

(2) 一个位段必须分配在同一个存储单元之中，不能跨两个单元，可以用“unsigned:0 ”实现从下一个存储单元开始存放。

(3) 位段可在数值表达式中引用，并由系统自动转换为整型数，可用整型格式输出。

二、实验操作

(一) 目的要求

1. 掌握按位运算的概念和方法，学会使用位运算符。
2. 学会通过位运算实现对某些位的操作。

(二) 范例

1. 编一个程序，打印出小于某整数的所有素数。

找出小于某整数的所有素数的典型算法是筛选法。例如，如果我们要找出小于 20 的所有素数，那么可以将 2 到 20 之间的所有整数作为筛子中的内容列出来：

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

去掉其中最小的数，它肯定是素数，并去掉此素数的所有倍数。经过这样的处理，可得到素数 2，而筛子中只留下奇数：

3 5 7 9 11 13 15 17 19

重复上述步骤，即最小的数 3 是素数，并去掉它的所有倍数。此时筛子中只留下：

5 7 11 13 17 19

这样的筛洗过程一直重复到筛子中没有一个数为止。

为了最有效地利用存储空间，我们把筛子定义成一个无符号整数类型的数组，此时每个数组元素的 16 位二进制位可以存放 16 个整数的标志，若标志为 1，表示对应此位的整数还存在于此筛子中，若标志为 0，则表示对应此位的整数已被筛选掉。

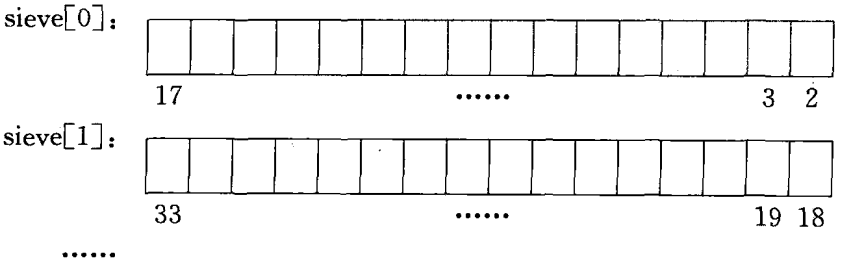
若我们预定义符号常数：

```
#define MAXIMUN 10000
```

然后在程序中说明：

```
unsigned sieve[MAXIMUN];
```

则数组代表筛子，它的形式是：



这个由 10000 个元素组成的数组。最多可存放 160000 个整数(2 到 160001)的标志。程序一开始应将元素中每位都置 1，表示所有的整数都在筛子中。然后每找到一个素数就把该素数和它的所有倍数所对应的位置 0；如此循环，直到筛子为空，即所有位为 0 为止。

源程序 prg9_1.c 清单：

```
#include <stdio.h>          //适合 16 位机
#define FIRSTPRIME 2
#define MAXIMUN 100
void main(){
  int i,leftin,range,rangemod16,factor,multiple,temp,k=1;
  unsigned sieve[MAXIMUN];
  int thisbit(int n);
  printf("range=");
  scanf("%d",&range);
  leftin=range-FIRSTPRIME+1;
  rangemod16=leftin/16+1;
  for(i=0; i<rangemod16; i++){
      sieve[i]=0177777;
  }
  factor=FIRSTPRIME-1;
  do{
      factor++;
      i=(factor-FIRSTPRIME)/16;
      if(sieve[i]&thisbit(factor)){              // factor is prime
        printf("%5d%c",factor,(k++%10)? ' ':'\n');
        multiple=1;
        while(factor * multiple<=range){
            temp=factor * multiple;
            i=(temp-FIRSTPRIME)/16;
            if(sieve[i]&thisbit(temp)){        // remove multiple
            sieve[i]=sieve[i]^thisbit(temp);
```

```
                leftin--;
                }
                multiple+=1;
            }
        }
    }while(leftin);
}
int thisbit(int n){        // set 1 in proper bit of a word
    int k,w;
    k=(n-FIRSTPRIME)%16;
    w=1;
    w=w<<k;
    return(w);
}
```

运行结果：

```
range=100
 2    3    5    7   11   13   17   19   23   29
31   37   41   43   47   53   59   61   67   71
73   79   83   89   97
```

程序中 do～while 语句前的一些语句是做筛洗前的初始化工作：读入希望列出的所有素数的最大值送到变量 range 中；计算筛洗前筛子中所有整数的个数；将 sieve 中的各二进制位置 1，表示 2 到 range 间的整数都存在于筛子中。接着开始的筛洗工作是由一个 do～while 循环语句完成的。在循环体中，将变量 factor 从 2 开始每次增 1，然后找到此整数的标志位所在的数组元素和该元素中的对应位，根据此标志位的值判别此整数是否是素数。若是素数，则输出此素数并通过一个 while 循环语句将此素数及它所有的倍数从筛子中去掉。每去掉一个整数，在 sieve 的相应位上置 0(通过按位加运算)，并将留在筛子中的总数 leftin 减 1。如此反复循环，直到 leftin 为 0。程序中调用函数 thisbit，它的功能是找到某个整数在筛子 sieve 的相应元素中的二进制位置，然后返回一个在此二进位上为 1，其他位为 0 的整数，供调用程序做按位与(&)及按位加(^)运算。

源程序 prg9_1.c 适用于 16 位机，对于在 32 位机上使用，要做相应修改，如源程序 prg9_2.c 所示。

源程序 prg9_2.清单：

```
#include <stdio.h>
#define FIRSTPRIME 2
void main(){
    int i,leftin,range,rangemod32,factor,multiple,temp,k=1;
    unsigned sieve[MAXIMUN];
    int thisbit(int n);
```

```
    printf("range="); scanf("%d",&range);
    leftin=range-FIRSTPRIME+1;
    rangemod32=leftin/32+1;
    for(i=0; i<rangemod32; i++){
        sieve[i]=0xffffffff;
    }
    factor=FIRSTPRIME-1;
    do{
        factor++;
        i=(factor-FIRSTPRIME)/32;
            if(sieve[i]&thisbit(factor)){          // factor is prime
             printf("%5d%c",factor,(k++%10)? ' ':'\n');
             multiple=1;
              while(factor * multiple<=range){
                  temp=factor * multiple;
                  i=(temp-FIRSTPRIME)/32;
                  if(sieve[i]&thisbit(temp)){      // remove multiple
                  sieve[i]=sieve[i]^thisbit(temp);
                  leftin--;
              }
              multiple+=1;
            }
        }
    }while(leftin);
}

int thisbit(int n){          // set 1 in proper bit of a word
    int k,w;
    k=(n-FIRSTPRIME)%32;
    w=1;
    w=w<<k;
    return(w);
}
```

（三）实验内容

1. 编写一个函数 getbis，从一个 16 位的单元中取出某几位（即这几位保留原值，其余位为 0）。函数调用形式为 getbits(value,n1,n2)，其中 value 为该 16 位（两个字节）中 的数据值，n1 为欲取出的起始位，n2 为欲取出的结束位，如“getbits(0101675,5,8)”表示对八进制 101675，取出它的从左面起第五位到第八位。

2.（选做题）将十六进制数转换为二进制数。例如，输入：

f1e2

输出：

1111000111100010

3.（选做题）编写程序，运用位运算来判断整数的奇偶性。

4.（选做题）编写一个函数，将一个整数中的某几个连续位翻转，即这些位原来是1则置为0，原来是0则置为1。

5.（选做题）编写一个函数，该函数能将给定的整数循环右移n位。

6.（选做题）编写程序，运用位运算来实现两个整型变量值的交换。

（四）常见错误分析

参加位运算的操作数必须是整型常量或变量。这一点一定要切记！

【错误示例9.1】

```
#include <stdio.h>
void main(){
    float a,b;          //位操作对象类型定义错误
    a=0x1b; b=a<<2;
    printf("0x%x left_shift 2 bit:0x%x\n",a,b);
}
```

编译时将给出以下错误信息：

```
error C2296: '<<' : illegal, left operand has type 'float'
```

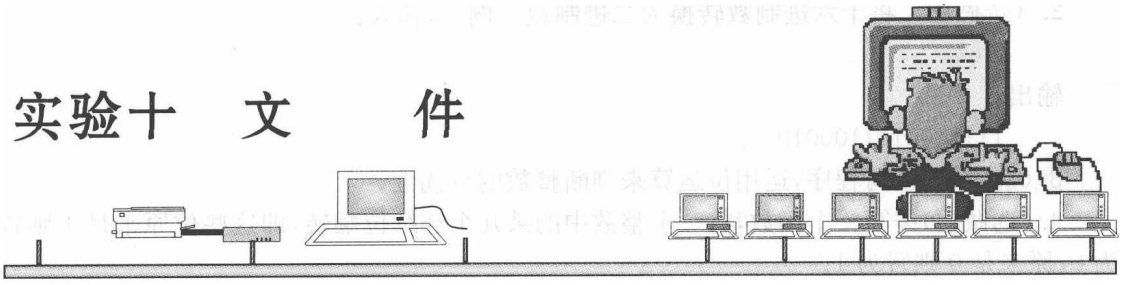
正确的程序如下所示：

```
#include <stdio.h>
void main(){
    unsigned char a, b;
    a=0x1b; b=a<<2;
    printf("0x%x left_shift 2 bit:0x%x\n",a,b);
}
```

运行结果：

```
0x1b left_shift 2 bit:0x6c
```

实验十　文　　件



学习重点

- 文件的概念
- 文件类型指针
- 文件的打开和关闭(fopen()函数和 fclose()函数)
- 格式化的读写(fscanf()函数与 fprintf()函数)
- 数据块的读写(fread()函数与 fwrite()函数)
- 字符的读写(fgetc()函数与 fputc()函数)
- 字符串的读写(fgets()函数与 fputs()函数)
- feof、rewind、fseek 函数的基本使用方法

一、要点综述

1. 文件的概念

(1) 一个 C 文件是一个字节流或二进制流,它把数据看作是一连串的字符序列(字节),而不考虑记录的界限,即 C 文件并不是由记录组成的。

(2) C 语言中的文件,按照数据存放的形式可分为两类:文本文件(ASCII 代码文件或字符代码文件)和二进制文件(非文本文件)。

① 文本文件由一个一个字符组成,每一个字符用一个 ASCII 代码表示,其特点是可以通过显示器或打印机显示,字符处理和阅读比较方便,但占用存储空间较大。

② 二进制文件是指以数据在内存中的存储形式原样输出存储到磁盘上的文件,它按照数据值的二进制代码形式存放。如 123.45 在内存中以浮点数形式存储,占 4 个字节,若以二进制形式存储到磁盘,则仍是 4 个字节按原来在内存中的存储形式一样。其特点是节省存储空间,输入输出无需转换,速度快,但不能在显示器或打印机直接显示输出,所以一般用于程序与程序之间或者程序与设备之间数据的传递。

(3) 对文件的操作要与各种外部设备发生联系,因此对文件的输入输出(读写)过程是通过操作系统来实现的。对文件的读写都是通过标准的 I/O 函数进行的。

(4) 根据计算机系统是否可以为文件自动设置缓冲区,可以分缓冲文件系统(标准 I/O)

和非缓冲文件系统。

① 缓冲文件系统的特点：对程序中的每一个文件都在内存中开辟一个“缓冲区”。从磁盘文件读入的数据先送到“输入缓冲区”中，然后再从缓冲区依次将数据送给接收变量。在向磁盘文件输出数据时，先将程序数据区中变量或表达式的值送到“输出文件缓冲区”中，待装满后才一起输出给磁盘文件。目的减少对磁盘的读写次数，即一次可以读入一批数据或输出一批数据。

② 非缓冲文件系统的特点：不由系统自动设置所需缓冲区，而由用户自己根据需要设置。

原 UNIX 系统用缓冲文件系统处理文本文件，用非缓冲文件系统处理二进制文件。新的 ANSI 标准只建议使用缓冲系统，并对缓冲文件系统的功能进行了扩充，使之既能用于处理字符代码文件，也能处理二进制文件。

2. 文件类型的指针

在 C 程序中对文件的操作都是通过文件指针变量实现的。缓冲文件系统中关键的概念是“文件指针”。

(1) 要调用一个文件至少需要有以下的信息：

① 文件当前读写的位置。

② 与该文件对应的内存缓冲区的地址。

③ 缓冲区未被处理的字符数。

④ 文件的操作方式等。

为此，C 语言在 stdio. h 中定义了一个 FILE 文件结构体类型。例如，某个 UNIX 系统的文件结构体类型定义如下：

```
typedef struct{
    int _fd;            //文件号
    int _cleft;         //缓冲区中剩余的字符
    int_mode;           //文件操作模式
    char * _nextc;      //下一个字符位置(指针)
    char * _buff;       //文件缓冲区位置(指针)
}FILE;
```

说明：不同的系统文件结构体类型 FILE 将有所不同。

(2) 文件结构体类型的指针变量定义形式：

```
FILE  * 指针变量名;
```

(3) 文件指针变量的赋值操作是通过打开文件函数 fopen 实现的。

3. 文件的打开

文件必须先打开后使用，并及时关闭。

打开文件过程实际上就是用户程序和操作系统之间建立起联系的过程，程序要把操作的文件的一些信息通知操作系统，如文件名、文件操作方式(读/写)。

文件的打开方式一般为：

```
FILE *fp;
if((fp=fopen("文件名", "使用文件方式"))==NULL){
    printf("cannot open this file.\n");
    exit(0);
}
```

(1) 文件名可以是包含文件标识符(盘符、路径、文件名和扩展名)的字符串常量、字符数组名或指向字符串的字符指针。

(2) 使用文件方式如表 3.13 所示。

表 3.13 文件打开方式

文件使用方式		含义
r	(只读)	为输入打开一个字符文件
w	(只写)	为输出打开一个字符文件
a	(追加)	向字符文件尾部增补数据
rb	(只读)	为输入打开一个二进制文件
wb	(只写)	为输出打开一个二进制文件
ab	(追加)	向二进制文件尾部增补数
r+	(读写)	为读/写打开一个字符文件
w+	(读写)	为读/写建立一个新的字符文件
a+	(读写)	为读/写打开一个字符文件
rb+	(读写)	为读/写打开一个二进制文件
wb+	(读写)	为读/写建立一个新的二进制文件
ab+	(读写)	为读/写打开一个二进制文件

(3) 如果按指定使用文件方式失败,则将返回一个空指针 NULL。此时一般应结束程序的运行并返回系统,查找原因进行出错处理。

(4) 读文件需要先确认此文件是否已存在,并将读写当前位置设定为文件的开头。

(5) 写文件需要检查原来是否有同名文件,如有则将该文件原有内容删除,如无同名文件就建立一个新文件,然后将写当前位置设定于文件的开头,以便从文件开头写入数据。

4. 文件的关闭

C 语言文件使用完后,应该及时关闭。以防止由于误操作等原因破坏已打开的文件。

文件的关闭通过库函数 fclose 进行:

```
fclose(fp);
```

功能:将文件指针 fp 所指文件关闭。

关闭文件意味着:

① 释放文件有关信息区。

② 将输出文件缓冲区的内容(无论缓冲区是否为满)都输出写入文件,然后关闭文件,这样可以防止丢失本来应写入文件的数据。

5. 常用读写文件的函数

(1) 格式化的读(输入)写(输出)文件函数要求文件的输入格式与输出格式对应统一。

① 格式化的读文件函数一般调用形式：

```
fscanf(fp, 格式字符串,输入项表列)
```

功能：从 fp 所指文件按格式串读入数据。函数执行成功返回读得的数据项个数；若在读第一项前已到达文件结束处，则返回 EOF。

② 格式化的写文件函数一般调用形式：

```
fprintf(fp,格式字符串,输出项表列)
```

功能：将输出项表列按格式串规定输出到 fp 所指文件。如果写入成功，返回写入文件的输出项表列的数目。

(2) 数据块的读文件函数。

① 数据块的读文件函数一般调用形式：

```
fread(buffer,size,count,fp)
```

功能：从 fp 指定文件读入 count 个大小为 size 的字节数据块，存入 buffer 所指存储单元。调用成功返回所读的数据项个数，若遇文件结束或出错返回 0。buffer 是数据存储的起始地址，可以是数组或指向确定的可用存储空间的指针。

② 数据块的写文件函数一般调用形式：

```
fwrite(buffer,size,count,fp)
```

功能：从 buffer 所指存储单元的数据取 size * count 个字节送 fp 所指文件。返回写到 fp 所指文件中的数据项个数。

数据块的读写文件函数每次都需要指定读写的长度，通常用于二进制文件的读写。例如当文件以二进制形式“rb” 打开时，fread()函数就可以读取各种类型的数据信息。

(3) 读写字符文件函数。

① 读字符函数 fgetc()一般调用形式：

```
ch=fgetc(文件型指针变量)
```

其中，ch 为字符变量，用以接受 fgetc()带回的一个字符。

功能：从指定的文件读入一个字符，该文件必须是以读或读写方式打开的。如果在执行 fgetc()函数读字符时遇到文件结束符，函数返回一个文件结束标志 EOF(−1)。

② 写字符函数 fputc()一般调用形式：

```
fputc(ch,文件型指针变量)
```

其中，ch 是要输出的字符，它可以是一个字符常量，也可以是一个字符变量。

功能：把一个字符写到指定文件中去。函数也带回一个值：如果输出成功，则返回值就是输出的字符；如果输出失败，则返回一个 EOF(−1)。

③ 读字符串函数 fgets()一般调用形式：

```
fgets(字符数组,字符数 n,文件型指针变量)
```

功能：从 fp 指定的文件中读取一个串，若满足以下条件读取结束：已读取 n−1 个字符；

当前读取到的字符是回车符；已读取到文件结束标志 EOF。读取成功后函数返回值为字符数组的首地址。

注意：读取的字符个数至多 n－1 个，余下的一个字节空间作为加“\0”串结束符。读取回车符时，回车符也作为一个字符送入由 str 所指的内存缓冲区。然后再加一个字符串结束符“\0”。

④ 写字符串函数 fputs()一般调用形式：

```
fputs(字符串,文件型指针变量)
```

功能：将字符串（可以是字符串常量、存放字符串的数组名或指向字符串的指针变量）写到文件型指针变量指向的文件中。

注意：与 fgets()函数在输入字符串末尾自动加“\0”特性对应，fputs()函数在将 C 字符串写入文件时，其末尾的“\0”字符自动舍去。

6. 文件的其他常用函数

文件常用函数还有：feof，rewind，fseek，ftell 等。

(1) feof 函数调用形式：

```
feof(文件型指针变量)
```

功能：判断文件型指针变量所指向的文件是否到达尾端。如果文件型指针变量所指向的文件的当前位置为文件最后的结束标志 EOF，则函数返回一个非零值，否则返回 0 值。

(2) rewind 函数调用形式：

```
rewind(文件型指针变量)
```

功能：是将指向文件的读写指针重新设置为该文件的开头。

每个打开的文件都有一个文件位置指针，它指示对应文件进行读写操作的具体位置，所以也称作文件读写指针。读写位置总是表示为距文件开头的字节数。

(3) fseek 函数调用形式：

```
fseek(文件型指针变量,偏移量,起始位置)
```

功能：以文件的起始位置为基准，根据偏移量往前或往后移动读写指针。其中偏移量是一个长整型数，表示从起始位置移动的字节数，正数表示指针往后移、负数表示指针往前移。起始位置用数字 0，1，3 或者符号常量 SEEK_SET，SEEK_CUR，SEEK_END 代表文件开始、文件当前位置和文件结束位置。若文件指针设置成功，返回值为 0，否则为非 0 值。重置文件指针时，可以超出文件尾部，但是不能指向文件开头之前。

(4) ftell 函数调用形式：

```
ftell(文件型指针变量)
```

功能：给出文件中读写指针的当前值。返回值是一个长整型，如果测试成功，则返回读写指针距离文件开头的字节数，否则返回－1L。

7. 文件的顺序存取和随机存取

文件按照读写方式，可分为顺序文件和随机文件。顺序文件的读写只能从头开始按顺

序读取，随机文件的读写过程可以随机进行。C 语言生成的文件既可以顺序存取也可以随机存取，并且以上介绍的文件的读写函数皆适用于这两种读写，fseek 函数可以用来实现文件的随机存取。

二、实验目的要求和实验内容

（一）目的要求

1. 掌握文件与文件指针的概念。
2. 学会使用文件打开、关闭、读写文件等基本的文件操作函数。

（二）范例

1. 设在文件 book. dat 中存放了一个书店的存书清单，每一种书包括以下信息：书名（20 个字符）、编号（4 位数字）、库存量（整型）、价格（实型）。编写一程序，计算该书店总存书量和库存书的总价。

源程序 prg10_1. c 清单：

```
#include "stdio. h"
#include "stdlib. h"
void main(){
  FILE *fp;
  int total_book=0, book;
  float total_cost=0. 0, cost;
  if((fp=fopen("book. dat","r"))==NULL){
      printf("book file can't open!");
      exit(0);
  }
  while(feof(fp)==0){
      fseek(fp,24L,1);
      fscanf(fp,"%d%f\n",&book,&cost);
      total_book+=book;
      total_cost+=cost*book;
  }
  printf("total books=%d\n",total_book);
  printf("total cost=%f\n",total_cost);
  fclose(fp);
}
```

运行结果：

```
total books=670
total cost=8820. 000000
```

存书文件 book. dat 内含信息：

How to us Computer	0001	100	5.00
The C Language	0002	50	10.00
Turbo C 2.0 User	0003	200	4.50
Microsoft C6.0User	0004	120	16.00
Windows for c++	0005	200	25.00

2. 编程实现分页打印 ASCII 文件，并加上行号。要求用命令行方式。

源程序 prg9_2.c 清单：

```
#include "stdio.h"
#define TLINE 10
#define LINEP 2
void main(int argc, char *argv[]){
  int flag, page, line, i;
  char buf[100];
  FILE *fp;
  if(argc < 2) {
     printf("\nNo file name\n"); exit(0);
  }
  if((fp=fopen(argv[1],"r"))==0) {
       printf("\nCan't open file \n"); exit(0);
  }
  flag=page=line=1;
  while(flag){
       for(i=0; i<LINEP; i++) printf("\n");
       printf("%s\n",argv[1]);
       for(i=0; i<TLINE; i++){
            if((fgets(buf,80,fp))==0)
            flag=0;
            if(flag) {
                 printf("%6d %s",line++,buf);
                 if(buf[78]!=0) printf("\n");
            }
            else
              for(i+=2; i<TLINE; i++) printf("\n");
       }
       printf("\n%65s%d.\n",". ",page++);
       for(i=0; i<LINEP; i++) printf("\n");
  }
  fclose(fp);
}
```

运行结果：

```
$a.out prg19.c↙
prg21.c
    1   #include "stdio.h"
    2   #define TLINE 10
    3   #define LINEP 2
    4   void main(int argc, char *argv[]){
    5   int flag, page, line, i;
    6   char buf[100];
    7   FILE *fp;
    8   if(argc < 2) {
    9   printf("\nNo file name\n"); exit(0);
   10 }
                                                    .1.
prg21.c
   11 flag=page=line=1;
   12 while(flag){
      ......
                                                    .2.
      ......
      ......
   35     fclose(fp);
   36 }                                             .4.
```

在此程序中：

(1) 首先检查命令行中有无参数(分页打印文件名)，若无则结束程序的执行，并显示信息：

N0 file name

若有则打开这个文件；若文件不存在，则终止程序，并显示信息：

Can't open file

(2) while 语句用来输出文件的内容，并加上行号和页号。

(3) 语句：

```
if(buf[78]!=0)
    printf("\n");
```

用来判别是否超过 78 个字符的行，若超过，则在输出该行的前 79 个字符后插入一个新行符。

(三) 实验内容

1. 有 5 个学生，每个学生有 3 门课程的成绩，从键盘输入学生数据(包括学号、姓名，3 门课程成绩)，并计算出每人的平均成绩。将原有数据和计算出的平均成绩存放在磁盘文件

“stud_dat”中。

2. 将上述第1题生成的“stud_dat”文件中的学生数据，按平均成绩进行排序处理，将已排序的学生数据存入一个新文件“stu_sort”中。

3. 读取“stu_sort”文件内容，并打印输出(输出到屏幕)。

4. 改写“实验八中实验内容第5题”的程序，使之能够把所建链表中的数据写入到一个文件“stud_link”中，并且能够从文件“stud_link”中读取数据重建链表。

5. 编写程序统计某给定ASCII文件中各字母的出现频率。

6. 编写程序，分别求出给定整数文件中等于大于某给定整数值的元素个数。

7. (选做题)编写程序，把给定整数文件中所有大于等于某值的数复制到另一个给定文件中去。

8. (选做题)编写程序，判定给定的任意两个ASCII文件是否相等。

9. (选做题)将上述第2题已排序的学生成绩文件进行插入处理。插入一个学生的3门课成绩，程序先计算新插入学生的平均成绩，然后将它按成绩高低顺序插入，插入后建立一个新文件。

10. (选做题)将上述第9题插入后建立一个新文件内容存入原有的“stud_sort2”文件，相当于完成文件的复制拷贝操作。

11. (选做题)编制一个程序，统计在指定文件包含的字符个数、字的个数、行数并输出。

12. (选做题) 有一磁盘文件，存放有某单位职工的有关信息：姓名(10个字符)、工作证号(4位数字)、性别(1个字符)、工资(实型数)，编译程序从文件中读取数据后，计算该单位N名职工的工资总数，并打印输出该工资表。

13. (选做题) 有一整数文件intfile，现要求将其中的偶数加倍，奇数加1，生成一个新的偶数文件evenfile。

14. (选做题)从键盘输入的一个字符串，将其中的小写字母全部转换为大写字母(其余字符原样不变)，然后输出到文件“ABC. TXT”中保存。

15. (选做题) 编一程序实现对文件ABC. TXT进行加密，采用加密的简单方法是当为“a”～“z”的小写字母时，用后一个字母代替前一个字母(若“z”则用“a”代替)，为其他字符时不变。例如：原文件为：

This is a secret code!

加密后的文件为：

Tijt jt b tfdsfu dpef!

提示：用fgetc()读文本文件时，NUIX系统文件在读取EOF前将会返回一个ASCII码为10(即Line feed换行)的字符，所以可采用while(c!=10) 结束循环；VC++ 6.0不存在这样的问题(情况)，故系统不同处理应有所不同。

16. (选做题)编程实现将某一源程序每行加上行号，并删除其中所有注释后生成一个打印文件(注意C语言程序注释有“//注释内容”和“/*注释内容*/”两种形式)。

(四) 常见错误分析

【错误示例10.1】 计算指定文件的大小。

```
#include <stdio.h>
```

```
#include <process.h>
void main(){
    FILE *fp; long num=0L; char ch;
    if((fp=fopen("command_line.exe","rb"))==NULL){
        printf("Open error! \n");
        exit(0);
    }
    while((ch=fgetc(fp))!=EOF)
        num++;
    printf("num=%d\n",num);
    fclose(fp);
}
```

以上示例中存在两处隐患：

(1) 判断文件型指针变量 fp 是否到文件尾的表达式使用不妥。表达式(ch=fgetc(fp))!=EOF 只能用于文本文件，因为 EOF 的 ASCII 码值为-1，而二进制文件可能包含数值为-1 的数据，这样当读入数据-1 时会被处理为文件结束。ANSI C 提供的 feof 函数可以判断文件是否真的结束，而且既可以用于文本文件，又可以用于二进制文件。

(2) fgetc 函数既可以用于读文本文件，又可以用于读二进制文件。只是当用于二进制文件时，读取的一个字节可能是数据的一部分。

运行验证的错误结果：

```
num=12          //command_line.exe 文件大小为 155690 字节，结果显然错误
```

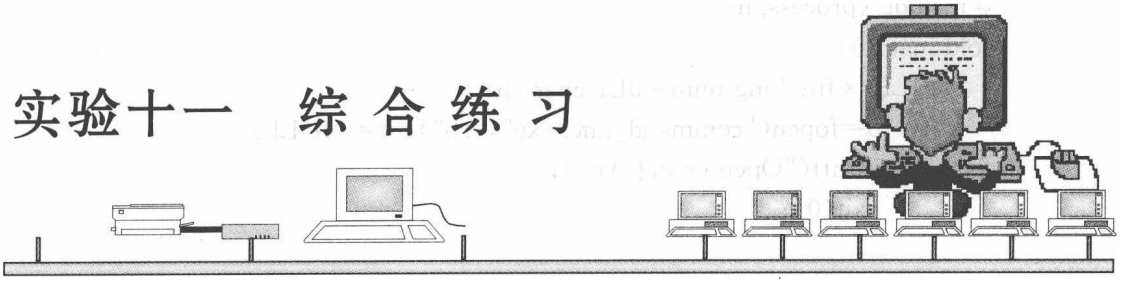
正确的程序如下所示：

```
#include <stdio.h>
#include <process.h>
void main(){
    FILE *fp; long num=0L; char ch;
    if((fp=fopen("command_line.exe","rb"))==NULL){
        printf("Open error! \n");
        exit(0);
    }
    while(! feof(fp)){
        ch=fgetc(fp);
        num++;
    }
    printf("num=%d\n",num);
    fclose(fp);
}
```

运行结果：

```
num=155691
```

实验十一　综合练习



学习重点

■　C语言综合运用

一、要点综述

在程序设计学习过程中，程序阅读也是一种学习、理解和获取经验的有效方法之一。通过阅读可以掌握控制结构的灵活运用、了解所介绍问题的算法、掌握某种处理技巧、体会到一种程序设计风格、积累更多的编程经验。完成本实验，最好的方法是先阅读并理解题目要求，自己酝酿一种求解算法或设计一个程序框架；之后再去参照给出的运行情况信息阅读源程序清单，不理解时再看相关注释、再辅以运行程序，以便更进一步加深理解。

二、实验操作

（一）目的要求

1. 综合应用本课程所学的知识。
2. 掌握应用程序的设计方法。
3. 了解C程序设计在计算方法中的应用。

（二）范例

1. 本程序为识别逆波兰算式的台式计算器程序，可实现四则运算（+，-，*，/），并能打印结果（当遇到“=”时）。如输入以下所示的“2+3*4=”的逆波兰算式：

```
2 3 4 * + =
```

则应得到结果：14.000000。

请分析程序，并回答如下问题：

(1) 程序中每个函数的功能是什么？

(2) 哪些变量是外部变量？哪些变量是局部变量？

(3) 上机实现并实际运算。

整个程序分布在四个源程序文本 cnpt. c，pushpop. c，atofgtp. c 和 chgetun. c 中。

(1) cnpt. c 源程序文本(文件之一)：

```
/* cnpt. c */
#include <stdio. h>
#define MAXVAL 100              /* maximun depth of val stack */
#define MAXOP 20                /* max size operator */
#define NUMBER '0'              /* signal that number found */
#define TOOBIG '9'              /* signal that string is too big */
int sp=0;                       /* stack pointer */
double val[MAXVAL];             /* value atsck */
void main(){
  int type;
  char s[MAXOP];
  double op2,atof(),pop(),push();
  while((type=getop(s,MAXOP))!=EOF)
    switch(type){
        case NUMBER : push(atof(s));
                      break;
        case '+'    : push(pop()+pop());
                      break;
        case '*'    : push(pop()*pop());
                      break;
        case '-'    : op2=pop(); push(pop()-op2);
                      break;
        case '/'    : op2=pop();
                      if(op2!=0. 0)
                        push(pop()/op2);
                      else
                        printf("zero divisor popped\n");
                      break;
        case '='    : printf("\t%f\n",push(pop()));
                      break;
        case 'c'    : clear();
                      break;
        case TOOBIG : printf("%. 20s. . . is too long\n",s);
```

```
                    break;
        default     : printf("unknown command%c\n",type);
                    break;
      }
}
```

(2) pushpop.c 源程序文本(文件之二):

```
/* pushpop.c */
#include <stdio.h>
#define MAXVAL 100
extern int sp;
extern double val[];
double push(double f){              /* push f onto value stack */
  if(sp<MAXVAL)
    return(val[sp++]=f);
  else{
      printf("error:stack full\n");
      clear();
      return(0);
  }
}
double pop(){                       /* pop top value from stack */
if(sp>0)
    return(val[--sp]);
  else{
      printf("error:stack empty\n");
      clear();
  }
}
clear( ){
  sp=0;
}
```

(3) atofgtp.c 源程序文本(文件之三):

```
/* atofgtp.c */
#include <stdio.h>
#define NUMBER '0'
#define TOOBIG '9'
double atof(char s[]){              /* convert string to double */
  double val,power;
```

```
    int i,sign;
    for(i=0; s[i]==' ' || s[i]=='\n' || s[i]=='\t'; i++)
      ;
    sign=1;
    if(s[i]=='+' || s[i]=='-')
      sign=(s[i++]='+')? 1 : -1;
    for(val=0; s[i]>='0'&&s[i]<='9'; i++)
      val=10 * val+s[i]-'0';
    if(s[i]=='.')
      i++;
    for(power=1; s[i]>='0'&&s[i]<='9'; i++){
        val=10 * val+s[i]-'0';
        power * =10;
    }
    return(sign * val/power);
}
getop(char s[],int lim){            /* get next operator or operand */
    int i,c;
    while((c=getch())==' ' || c=='\t' || c=='\n')
      ;
    if(c!='.'&&(c<'0' || c>'9'))
      return(c);
    s[0]=c;
    for(i=1; (c=getch())>='0'&&c<='9'; i++)
      if(i<lim)
        s[i]=c;
    if(c=='.'){
      if(i<lim)
        s[i]=c;
    for(i++; (c=getch())>='0'&&c<='9'; i++)
      if(i<lim)
      s[i]=c;
    }
    if(i<lim){
        ungetch(c);
        s[i]='\0';
        return(NUMBER);
    }
    else{                           /* it's too big, skip rest of line */
```

```
        while(c!='\n'&&c!=EOF)
            c=getch();
        s[lim-1]='\0';
        return(TOOBIG);
    }
}
```

(4) chgetun. c 源程序文本(文件之四):

```
/* chgetun. c */
#include <stdio. h>
#define BUFSIZE 100
char buf[BUFSIZE];                /* buffer for ungetchar */
int bufp=0;                       /* next free position in buf */
getch(){                          /* get a(possibly pushed back) character */
  return((bufp>0)? buf[--bufp] : getchar());
}
ungetch(int c){
  if(bufp>BUFSIZE)
    printf("ungetch:too many characters\n");
  else
    buf[bufp++]=c;
}
```

运行验证结果:

```
input:
2 3 4 * + =
<ctrl>+d
output:
    14. 000000
```

2. 用变步长梯形积分求解函数的定积分。

在实际工程应用中,许多定积分的求值都是通过计算机数值计算的方法近似得到的,本例介绍用最基本的变步长梯形积分求解函数的定积分的方法。

算法基本原理:

这里只考虑最简单的情况。设被积函数是一个一元函数,定积分表达式为:

$$I=\int_a^b f(x)\mathrm{d}x \tag{3.1}$$

积分表示的意义是一元函数 $f(x)$ 在区间 a 到 b 之间与 x 轴所夹的面积,如图 3.12 所示。

如果上述积分可积,对于任意整数 n,取 $h=(b-a)/n$,记

$$x_k = a + k * h$$

$$I_k = \int_{x_{k-1}}^{x_k} f(x)\mathrm{d}x$$

则有：

$$I = \sum_{k=1}^{n} I_k$$

在小区间$[x_{k-1}, x_k]$上，取 I_x的某种近似，就可以求得 I 的近似值。

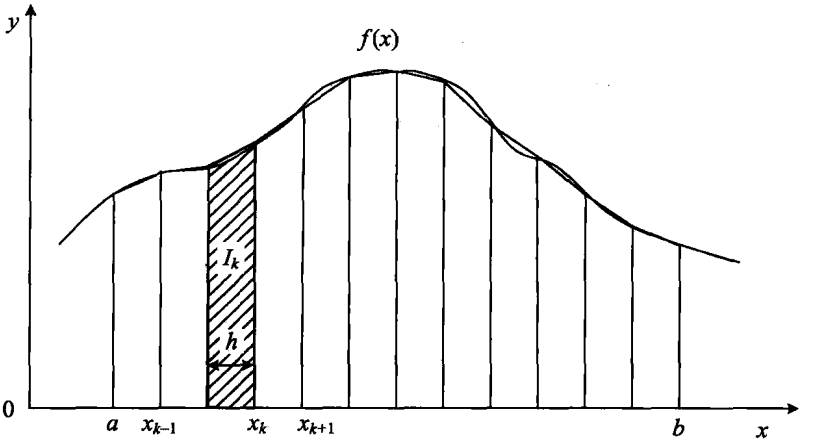


图 3.12 梯形积分原理示意图

在区间$[x_{k-1}, x_k]$上取 $I_k = [f(x_{k-1}) + f(x_k)] * h/2$，就是梯形积分。梯形积分法的一个简单理解就是，把原积分区间划分为一系列小区间，在每个小区间上都用小的梯形面积来近似原函数的积分，当小区间足够小时，我们就可以得到原来积分的近似值。这时，整个积分结果可以近似表示为：

$$T_n = \sum_{k=0}^{n-1} \frac{h}{2}[f(x_k) + f(x_{k+1})] \tag{3.2}$$

在使用这个积分公式之前必须给出合适的步长 h，步长太大精度难以保证，而步长太小会导致计算量的增加。因此，实际计算往往采用变步长的方法，即在步长逐次减半（步长二分）的过程中，反复利用上述求积公式计算，直到所求得的积分结果满足所要求的精度为止。

把积分区间$[a,b]$分成 n 等分，就得到 $n+1$ 个分点，按照公式(3.2)计算积分值 T_n，需要计算 $n+1$ 次函数值。如果积分区间再二分一次，则分点增加到 $2n+1$ 个，我们把二分前后的积分值联系起来考察，分析它们的递推关系。

在二分之后，每一个积分子区间$[x_k, x_{k+1}]$经过二分后只增加了一个分点 $x_{k+1/2} = (x_k + x_{k+1})/2$，二分之后这个区间的积分值为：

$$\frac{h}{4}[f(x_k) + 2f(x_{k+\frac{1}{2}}) + f(x_{k+1})] \tag{3.3}$$

注意：这里的 $h = (b-a)/n$，还是二分之前的步长。把每个区间的积分值相加，就得到二分之后的积分结果：

$$T_{2n} = \frac{h}{4}\sum_{k=0}^{n-1}[f(x_k) + f(x_{k+1})] + \frac{h}{2}\sum_{k=0}^{n-1} f(x_{k+\frac{1}{2}}) \tag{3.4}$$

利用二分前的积分结果(3.2)式，就可以得到递推公式：

$$T_{2n}=\frac{1}{2}T_n+\frac{h}{2}\sum_{k=0}^{n-1}f(x_{k+\frac{1}{2}}) \tag{3.5}$$

对于实际问题，可以采取的一般步骤为：

(1) 取 $n=1$，利用(3.2)式计算积分值。

(2) 进行二分，利用递推公式(3.5)式计算新的积分值。

(3) 进行判断，如果两次计算积分值的差在给定的误差范围之内，二分后的积分值就是所需的结果，计算结束。如果两次计算积分结果之差在误差范围之外，返回第(2)步，继续运行。

值得注意的一点是，这种积分方法对被积函数有一定的要求。设想如果一个函数刚好在积分边界 a,b 和中心点 $(a+b)/2$ 处，取值为 0，而在其他地方的某一段取值不为 0，则第(1)步计算结果为 0，第(2)步计算结果仍然是 0，于是就得出积分结果为 0 的结论，这显然是错误的。

根据以上的算法分析，相应的源程序文件下：

```
#include <stdio.h>
#include <math.h>
#define EPS 1.0E-7

double fun(double x){                          //定义被积函数
    return(log(1.0+x)/(1.0+x*x));
}

double trapezia(double a,double b){
    int n,k,done=0;
    double x,h,Tn,T2n,temp;
    n=1;
    h=b-a;
    Tn=h*(fun(a)+fun(b))/2.0;                  //计算 n=1 时的积分值
    while(!done){
        temp=0;
        for(k=0;k<n;k++){
            x=a+(k+0.5)*h;
            temp+=fun(x);
        }
        T2n=(Tn+h*temp)/2.0;                   //变步长梯形法计算
        if(fabs(T2n-Tn)<EPS)                   //判断积分误差
            done=1;
        else{
            Tn=T2n;                            //进行下一步计算准备
            n*=2;
            h/=2;
```

```
            }
            }
            return(T2n);
    }

    void main(){
        double y;
        y=trapezia( 1, 2 );
        printf("trapezia(1,2):%f\n",y);
        y=trapezia( 0, 2 );
        printf("trapezia(0,2):%f\n",y);
    }
```

运行验证结果：

```
trapezia(1,2):0.282697
trapezia(0,2):0.554895
```

3. 列主元高斯消元法求解方程组。

例如，有下面联立方程组：

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 = b_3 \\ a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 = b_4 \end{cases}$$

试求出 x_1, x_2, x_3, x_4。

在计算方法中解联立方程有很多速度较快的方法，在本例题中所采用的解法是一种最为简单的方法，读者只要具备中学的解联立方程的代数知识，结合例题的讲解就能看懂程序。本例主要帮助读者了解如何根据分析的数学问题来编写程序。

为了解联立方程组，可把方程组的系数放在 A 矩阵中，把方程组等号右边的常数放在 B 矩阵中，解联立方程组主要有两个步骤：第一步是消元，第二步是回代。

第一步，消元。

这一步的目的是为了达到如下变换：

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \begin{pmatrix} b \\ b_2 \\ b_3 \\ b_4 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & a'_{12} & a'_{13} & a'_{14} \\ 0 & 1 & a'_{23} & a'_{24} \\ 0 & 0 & 1 & a'_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \end{pmatrix}$$

现以下面的具体方程作为范例来解释消元的过程。

$$\begin{cases} 2x_1 + 5x_2 + 2x_3 + 3x_4 = 4 \\ 2x_1 + 3x_2 + 3x_3 + 4x_4 = 0 \\ 3x_1 + 6x_2 + 3x_3 + 2x_4 = 4 \\ 4x_1 + 12x_2 \qquad + 8x_4 = 0 \end{cases}$$

(1) 对 A 矩阵的第一列进行消元。

① 找最佳主元素。主元素是指进行消元列上的对角线元素，每次进行消元前先要使主元素的绝对值为该列上元素绝对值的最大值，这样可以使所得结果误差较小。对示例方程组，有：

$$\begin{pmatrix} 2 & 5 & 2 & 3 \\ 2 & 3 & 3 & 4 \\ 3 & 6 & 3 & 2 \\ \boxed{4} & 12 & 0 & 8 \end{pmatrix} \begin{pmatrix} 4 \\ 0 \\ 4 \\ 0 \end{pmatrix}$$

由于第 1 列中第 4 行上的元素值最大，因此需要把第 4 行和第 1 行对调，同时，B 数组中第 4 行和第 1 行中的元素也作相应的对调，然后进行消元，即消去消元列上主元素以外的元素。

② 使每行的元素都除以本行中第 1 列(进行消元的列)元素的值，使第 1 列上的元素值都为 1，过程操作示意如下：

$$\begin{pmatrix} \frac{4}{4} & \frac{12}{4} & \frac{0}{4} & \frac{8}{4} \\ \frac{2}{2} & \frac{3}{2} & \frac{3}{2} & \frac{4}{2} \\ \frac{3}{3} & \frac{6}{3} & \frac{3}{3} & \frac{2}{3} \\ \frac{2}{2} & \frac{5}{2} & \frac{2}{2} & \frac{3}{2} \end{pmatrix} \begin{pmatrix} \frac{0}{4} \\ \frac{0}{4} \\ \frac{4}{3} \\ \frac{4}{2} \end{pmatrix}$$

③ 消去第一列上的除主元素以外的其他元素。具体做法是第 2 行到第 4 行的元素分别减去第 1 行中各元素的值，过程操作示意如下：

$$\begin{pmatrix} 1 & 3 & 0 & 2 \\ 0 & \frac{3}{2}-3 & \frac{3}{2}-0 & 2-2 \\ 0 & 2-3 & 1-0 & \frac{2}{3}-2 \\ 0 & \frac{5}{2}-3 & 1-0 & \frac{3}{2}-2 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ \frac{4}{3}-0 \\ 2-0 \end{pmatrix}$$

至此，第 1 列主对角线上的元素值为 1，其余元素已消去(为 0)。

(2) 对第 2 列元素进行消元。

设 I 代表行号，J 代表列号，K 代表消元的列号。在这一步中 $K=2$。

① 找第 2 列上的最佳元素。判主对角线元素 $A(2,2)$ 的绝对值是否为第 2 列上对角线以下所有元素中的最大者，若不是，则把最大的 $A(I,2)$ 所在行换到第 2 行。

$$\begin{pmatrix} 1 & 3 & 0 & 2 \\ 0 & -\frac{3}{2} & \frac{3}{2} & 0 \\ 0 & -1 & 1 & -\frac{4}{3} \\ 0 & -\frac{1}{2} & 1 & -\frac{1}{2} \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ \frac{4}{3} \\ 2 \end{pmatrix}$$

② 这一步的功能与(1)中的②类似，只是对虚线范围内的元素进行操作。此时 K 的值为 2，即使虚线范围内每行的元素都除以本行中第 2 列(进行消元的列)元素的值，使第 2 列上的元素值都为 1，过程操作示意如下：

$$\left(\begin{array}{c|ccc} 1 & 3 & 0 & 2 \\ \hline 0 & -\frac{3}{2} & \frac{3}{2} & \\ & \frac{-\frac{3}{2}}{-\frac{3}{2}} & \frac{\frac{3}{2}}{-\frac{3}{2}} & \frac{0}{-\frac{3}{2}} \\ 0 & \frac{-1}{-1} & \frac{1}{-1} & \frac{-\frac{4}{3}}{-1} \\ 0 & \frac{-\frac{1}{2}}{-\frac{1}{2}} & \frac{1}{-\frac{1}{2}} & \frac{-\frac{1}{2}}{-\frac{1}{2}} \end{array}\right)\left(\begin{array}{c} 0 \\ \hline 0 \\ \frac{0}{-\frac{3}{2}} \\ \frac{\frac{4}{3}}{-1} \\ \frac{2}{-\frac{1}{2}} \end{array}\right)$$

③ 这一步与(1)中的③类似，只要把 $K=1$ 改成 $K=2$，即消去虚线范围内第 2 列上的除主元素以外的其他元素，做法是第 3 行到第 4 行的元素分别减去第 2 行中各元素的值，过程操作示意如下：

$$\left(\begin{array}{c|ccc} 1 & 3 & 0 & 2 \\ \hline 0 & 1 & -1 & \\ 0 & 1-1 & -1-(-1) & \frac{4}{3}-0 \\ 0 & 1-1 & -2-(-1) & 1-0 \end{array}\right)\left(\begin{array}{c} 0 \\ \hline 0 \\ -\frac{4}{3}-0 \\ -4-0 \end{array}\right)$$

通过这一步消元，第 2 列中主对角线以下所有元素已消去，对角线元素为 1。

(3) 对第 3 列元素消元。

这时，$K=3$，其他步骤都和(1)、(2)所述相同。A、B 矩阵中，只有虚线范围内的元素按上两步的规律变化，其他元素保持不变。

在这一步中由于 -1 的绝对值大于 0，因此，A、B 矩阵中的第 3 行和第 4 行元素对调，然后再消元。在步骤②中由于 $A(4,3)$ 的值为 0，因此不再进行步骤③，直接跳出 I 循环，完成第 3 列的消元。

$$\underset{①}{\left(\begin{array}{c|ccc} 1 & 3 & 0 & 2 \\ 0 & 1 & -1 & 0 \\ \hline 0 & 0 & 0 & \frac{4}{3} \\ 0 & 0 & -1 & 1 \end{array}\right)\left(\begin{array}{c} 0 \\ 0 \\ \hline -\frac{4}{3} \\ -4 \end{array}\right)} \Rightarrow \underset{②}{\left(\begin{array}{c|ccc} 1 & 3 & 0 & 2 \\ 0 & 1 & -1 & 0 \\ \hline 0 & 0 & \frac{-1}{-1} & \frac{1}{-1} \\ 0 & 0 & 0 & \frac{4}{3} \end{array}\right)\left(\begin{array}{c} 0 \\ 0 \\ \hline \frac{-4}{-1} \\ -\frac{4}{3} \end{array}\right)} \Rightarrow$$

$$\underset{③}{\left(\begin{array}{c|ccc} 1 & 3 & 0 & 2 \\ 0 & 1 & -1 & 0 \\ \hline 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & \frac{4}{3} \end{array}\right)\left(\begin{array}{c} 0 \\ 0 \\ \hline 4 \\ -\frac{4}{3} \end{array}\right)}$$

(4) 最后只剩下 $A(4,4)$ 元素进行处理了，在这一步中 $K=4$，矩阵变化情况如下所示：

$$\begin{bmatrix} 1 & 3 & 0 & 2 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & \frac{4}{3} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 4 \\ -\frac{4}{3} \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 3 & 0 & 2 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & \frac{4}{3} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 4 \\ -\frac{4}{3} \end{bmatrix} \Rightarrow$$

①　　②

$$\begin{bmatrix} 1 & 3 & 0 & 2 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 4 \\ -1 \end{bmatrix}$$

③

在每次消元过程中，如果进行消元的列上从主对角线上的元素起以下所有元素的值都为 0，则此方程有无穷多个解或无解，这里我们认为这两种情况是不可解的，因此程序停止运行(实际上有无穷多解时，可以找出解空间的基，由于这已超出本课程的范围，因此这里我们就不再讨论了)。

第二步，回代，求出各 x 的值。

经过上面消元后，联立方程组已变换成如下形式：

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 = b_1 \\ x_2 + a_{23}x_3 + a_{24}x_4 = b_2 \\ x_3 + a_{34}x_4 = b_3 \\ x_4 = b_4 \end{cases}$$

因此各 x_i 的值可表示如下：

$$\begin{cases} x_4 = b_4 \\ x_3 = b_3 - a_{34}x_4 \\ x_2 = b_2 - a_{23}x_3 - a_{24}x_4 \\ x_1 = b_1 - a_{12}x_2 - a_{13}x_3 - a_{14}x_4 \end{cases}$$

以上对列主元高斯消去法解联立方程组的步骤作了详细说明，以下给出相应的算法描述：

输入：方程组阶数，矩阵 a 及向量 b。

列主元高斯(Gauss)消去法求解过程：

```
for(i=1;i<n;i++){
    for(k=i;k<=n;k++)
        if(本列上从主对角线起的元素值是都为 0)
            打印无解消息并返回
    for(j=i+1,min=i,max=fabs(aij);j<=n;j++)
        找列主元素
    if(找到列主元素)
```

```
        则交换相应行
    for(j=i+1;j<=n;j++){
        //高斯消元
        temp=-a_ji/a_ii;
        b_j+=b_i * temp;
        for(k=i;k<n;k++)
            a_jk+=a_ik * temp;
    }
}
//回代过程,求解方程
X_n=b_n/a_nn;
for(i=n-1;i>0;i--){
    x_i=b_i;
    for(j=i+1;j<=n;j++)
        x_i-=a_ij * x_j;
    x_i/=a_ii;
}
```

将上述算法描述编写成用一个函数 gauss,供主函数调用求解联立方程组,具体实现的源程序如下:

```
//源程序名"解线性方程组.c"
//列主元高斯消去法求解 Ax=b。
#include <stdio.h>
#include <math.h>
#define MAX_N 20                    //矩阵的最大维数
int gauss(double a[][MAX_N],double b[MAX_N],double x[MAX_N],int n);
int main(){
    int n,i,j,solut=0;
    double a[MAX_N][MAX_N],b[MAX_N],x[MAX_N];
    printf("\ninput n value(dim of Ax=b): ");
    scanf("%d",&n);
    if(n>MAX_N){
        printf("The input n is large than MAX_N ,please redefine the MAX_N.\n");
        return(1);
    }
    if(n<=0){
        printf("Please input a number between 1 and %d.\n",MAX_N);
        return(1);
    }
```

```
    //输入 Ax=b 的 A 矩阵
    printf("Please input the matrix a(i,j),i,j=0,...,%d: \n",n-1);
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            scanf("%lf",&a[i][j]);
    //输入 b 向量
    printf("Please input the matrix b(i),i=0,...,%d: \n",n-1);
    for(i=0;i<n;i++)
        scanf("%lf",&b[i]);
    solut=gauss(a,b,x,n);            //调用列主元高斯消去法函数
    //输出方程解
    if(solut){
        printf("Solve... x_i=\n");
        for(i=0;i<n;i++)
            printf("%f\n",x[i]);
    }
    else
        printf("no unique solution or no solution.");
    return(0);
}
int gauss(double a[][MAX_N],double b[MAX_N],double x[MAX_N],int n){
    int i,j,k,min,solut=0;
    double max,temp;
    //列主元高斯消去法求解过程
    for(i=0;i<n-1;i++){
        //判断是否有解
        solut=0;
        for(k=i;k<n;k++)
            if(a[k][k]!=0)
                solut=1;
            if(solut==0){
                return(solut);
        }
        //找列主元素
        for(j=i+1,min=i,max=fabs(a[i][j]);j<n;j++)
            if(fabs(a[i][j])>max){
                min=j;
                max=a[j][j];
```

```
            }
        //如果找到列主元素,则交换两行
            if(i<min){
                temp=b[i];b[i]=b[min];b[min]=temp;
                for(j=i;j<n;j++){
                    temp=a[i][j];
                    a[i][j]=a[min][j];
                    a[min][j]=temp;
                }
            }
            //高斯消元
            for(j=i+1;j<n;j++){
                temp=-a[j][i]/a[i][i];
                b[j]+=b[i]*temp;
                for(k=i;k<n;k++)
                    a[j][k]+=a[i][k]*temp;
            }
        }
        //回代过程,求解方程
        x[n-1]=b[n-1]/a[n-1][n-1];
        for(i=n-2;i>=0;i--){
            x[i]=b[i];
            for(j=i+1;j<n;j++)
                x[i]-=a[i][j]*x[j];
            x[i]/=a[i][i];
        }
    }
```

运行结果：

```
input n value(dim of Ax=b): 4
Please input the matrix a(i,j),i,j=0,…,3:
2 5 2 3
2 3 3 4
3 6 3 2
4 12 0 8
Please input the matrix b(i),i=0,…,3:
4 0 4 0
Solve … x_i=
-7.000000
```

```
3.000000
3.000000
-1.000000
............
input n value(dim of Ax=b): 3
Please input the matrix a(i,j),i,j=0,…,2:
2 1 1 1 3 2 1 2 2
Please input the matrix b(i),i=0,…,2:
4 6 5
Solve... x_i=
1.000000
1.000000
1.000000
................
input n value(dim of Ax=b): 3
Please input the matrix a(i,j),i,j=0,...,2:
1 21 -2 -1 -5 0 -1 6
Please input the matrix b(i),i=0,...,2:
24 -63 50
Solve... x_i=
7.000000
4.000000
9.000000
```

4. 有一球以 10m/s 的速度从水平线以 45°向斜上方抛去，该球着地后仍保持原来方向向前弹跳，但速度是抛出时的 90%，以后每着地一次，速度都比上一次减小 10%。求此球在第三次着地前的轨迹(如图 3.13 所示)。

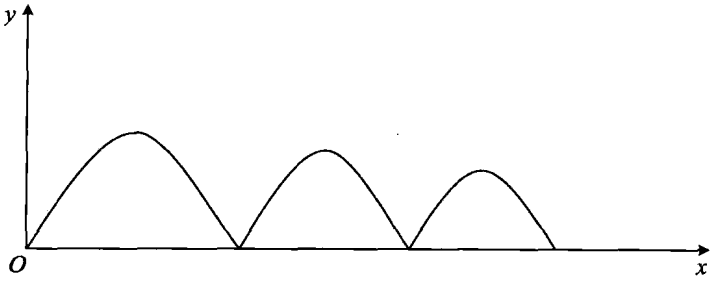


图 3.13 球的运动轨迹

球从抛出到第一次落地前，其轨迹可以用下列公式求出(设不计空气阻力)：

$$x = (v_0 \cos 45°) * t$$

$$y = (v_0 \sin 45°) * t - \frac{1}{2} g t^2$$

其中，$g=9.8\ m/s^2$。

令每 0.1s 采样一次，测 x，y 的值，即取 $\Delta t=0.1s$。在 $y>0$ 期间，表示球未着地，每隔 Δt 打印一次 x、y 的值。当 $y=0$ 期间，表示球着地，第一周期结束。球又向斜上方 45°弹跳，速度减小 10%，开始第二周期，方法同上……

由于 $\Delta t=0.1$，球有可能在两个 Δt 之间接触地面，例如，$t=1.4s$，$y=0.295495m$，而 $t=1.5s$ 时，y 值已变负。对这种情况，将 t 回退到前一次的值（$t=1.4$），将时间间隔减为 Δt 的 1/10，即 $\Delta t=0.01$，再计算每一次 Δt 的 x 和 y 的值。$t=1.44$ 时 y 为正，$t=1.45$ 时 y 为负。这时再将 Δt 乘以 0.1，即 $\Delta t=0.001$，并倒退回上一个 t（$t=1.44s$），再按新的 Δt（0.001）计算 x，y 的值。假设 $|y|\leqslant 0.01m$，就近似地认为着地（如果设 $|y|\leqslant 0.0001$ 或 $|y|\leqslant 0.00001$ 当然更好，更正确，但计算时间要长一些）。当 $|y|\leqslant 0.01$ 时，就认为本次弹跳结束，开始下一次弹跳，速度减小 10%，重新置 $t=0$，将此时 x 的值作为新的原点计算总的 x 坐标。

源程序清单：

```
#include <stdio.h>
#include <math.h>
#define TRUE  1
#define FALSE 0
#define G   9.8
void main(){
  int i,j,flag;
  double t,v,v0,x,x0,y,d,dt;
  t=0.0;
  x0=0.0;
  scanf("%lf%lf",&v0,&d);
  v=v0*cos(3.1415926/4.0);
  for(i=1; i<=3; i++){
      dt=d;
      flag=TRUE;
      x=v*t+x0;
      y=v*t-0.5*G*t*t;
      while(flag){
        if(y>=0.0){
          printf("t=%10.6f   x=%10.6f   y=%10.6f\n",t,x,y);
          t=t+dt;
          x=v*t+x0;
          y=v*t-0.5*G*t*t;
        }
        else if(fabs(y)>0.001){
              t=t-dt;
              dt=0.1*dt;
```

```
            t=t+dt;
            x=v*t+x0;
            y=v*t-0.5*G*t*t;
        }
        else{
            printf("t=%10.6f   x=%10.6f   y=%10.6f\n",t,x,y);
            flag=FALSE;
        }
    }
    v=0.9*v;
    t=0.0;
    x0=x;
    printf("\n");
  }
}
```

运行情况如下：

输入：10　0.1↙

输出：

```
t=  0.000000   x=   0.000000   y=   0.000000        (第一周期)
t=  0.100000   x=   0.707107   y=   0.658107
t=  0.200000   x=   1.414214   y=   1.218214
    ......
t=  1.442000   x=  10.196480   y=   0.007596
t=  1.443000   x=  10.203551   y=   0.000531
t=  1.443100   x=  10.204258   y=  -0.000176

t=  0.000000   x=  10.204258   y=   0.000000        (第二周期)
t=  0.100000   x=  10.840654   y=   0.587396
t=  0.200000   x=  11.477050   y=   1.076792
t=  0.300000   x=  12.113446   y=   1.468188
    ......
t=  1.298600   x=  18.468498   y=   0.001066
t=  1.298700   x=  18.469134   y=   0.000430
t=  1.298800   x=  18.469771   y=  -0.000206

t=  0.000000   x=  18.469771   y=   0.000000        (第三周期)
t=  0.100000   x=  19.042527   y=   0.523757
t=  0.200000   x=  19.615284   y=   0.949513
    ......
```

```
t=  1.167000   x= 25.153839   y=  0.010812
t=  1.168000   x= 25.159567   y=  0.005098
t=  1.169000   x= 25.165294   y= -0.000625
```

程序说明：输入 v_0 和规定的 Δt，用变量 d 代表 Δt，由于上抛 45°，cos45°=sin45°，因此用 $v=v_0\cos45°=v_0\sin45°$，先将 d 的值赋给 dt，以后 dt 的值可以变化，而 d 保持不变。设一个变量 flag 代表逻辑变量，用来表示球着地了没有，没着地时，flag=TRUE，球着地就使 flag=FALSE；当每过 0.1s，就打印一次 x，y 值，使 t 增值 dt，再计算 x，y 值；如果计算出 $y<0$，且 $|y|<0.001$，认为 $y\approx0$，球着地，Flag=FALSE，本循环结束，开始下一周期。若 $|y|>0.001$，则使 t 倒退一个 dt，并使 dt 乘 0.1，再计算。

（三）实验内容

1. 设计一个交互的文本行编辑程序。输入一正文行，要求该文本行编辑程序具有 5 种编辑功能：

插入(I)： 插入一字符串到文本行的指定位置

删除(D： 从文本行中移去一个指定字符。

寻找(F)： 确定一个字符串在正文行的位置。

替换(R)： 用一个字符串替换另一个字符串的每个出现。

停止(S)： 停止编辑，并保存现行文本行。

有 5 种编辑功能的选择建议用菜单形式实现。菜单的基本结构如下：

```
      ......
  do
    {
      显示菜单项，如：
      I  Iinsertroutine
      D  deleteroutine
      F  findretine
      R  replaceroutine
      S  stoproutine
      Key=getchar();
      Switch(key)
          { case 'I' : insertroutine();
                       break;
            case 'D': deleteroutine();
                       break;
            case 'F': findretine();
                       break;
            case 'R': replaceroutine();
                       break;
            case 'S' : stoproutine();
```

```
                    break;
            default : errorhandler();
        }
    }while(key !='S');
    ......
```

2. 设计一个通讯录管理程序。用一个单向链表来管理朋友的情况，每个朋友的情况包括一个姓名和一个电话号码。要求程序可完成的操作有(以菜单形式实现选择)：数据录入添加、显示、修改、查找、保存以及恢复，并以菜单形式实现。

3. 设计一个选修课程管理程序，用一个单向链表来管理学生选修课程的情况，每个学生选修课程情况包含的信息有学号、姓名、选修课程号、课程名和成绩。要求程序可完成的操作有：数据录入添加、显示、修改、查找(能分别按学号、姓名、选修课程号、课程名和成绩查找)、保存以及恢复，并以菜单形式实现。

4. 设计一个小型图书管理程序。要求具有以下基本功能：图书卡录入、借书证录入、借书登记和还书处理。

提示：首先需正确定义图书卡类型、借书证类型及相应的图书卡、借书证的存储单元(数组或链表)。然后根据相应的数据结构编程实现各项基本功能。

5. 结合本专业应用或日常事务，自选课题，编写一有意义的应用程序。如求曲线轨迹等。

第四部分 常 见 错 误

表 4.1 常 见 错 误

序号	常见错误	错误示例	更正示例

1	未定义类型	void main(){
		x=3; y=6;
		printf("%d\n",x+y);
		}
			void main(){
			int x,y;
			x=3; y=6;
			printf("%d\n",x+y);}

2	输出数据类型与格式说明不符	int a; float b;
		a=3; b=12.3;
		printf("%f %d\n",a,b);
			int a; float b;
			I=3; b=12.3;
			printf("%d %f\n",a,b);

3	输入数据类型与格式说明不正确	double x,y; scanf("%f%f",&x,&y); ……	double x,y; scanf("%lf%lf",&x,&y); ……

4-1	int 型数据超出范围	//设机器字长 16 位
		int num=89101;
		printf("%d",num);
			int(字长 16 位)型数据最大范围为:
			−32768～+32767
			即 -2^{15}～$(2^{15}-1)$

4-2	int 型数据超出范围	//设机器字长 32 位
		int num=2247483647;
		printf("%d",num);
			int(字长 32 位)型数据最大范围为:
			−2147483648～+2147483647
			即 -2^{31}～$(2^{31}-1)$

5	输入的变量未使用地址符	scanf("%d%d",a,b);	scanf("%d%d",&a,&b);
6	输入数据与控制格式符不符	scanf("%d,%d",&a,&b); 2 3↙	scanf("%d,%d",&a,&b); 2,3↙

7	库函数与头文件引用不匹配	#include <stdio.h>
		void main(){
		int x,y,z;
		scanf("%d%d",&x,&y);
		z=fabs(x−y);
		printf("%d\n:",z);
		}
			#include <math.h>
			void main(){
			int x,y,z;
			scanf("%d%d",&x,&y);
			z=fabs(x−y);
			printf("%d\n:",z);
			}

8	运算符用错	if(a=b)～else～;	if(a==b)～else～;
9	语句后标点用错或忘写	int a,b, float x,y	int a,b; float x,y;
10	错加分号	if(a>b);	if(a>b)

(续)表 4.1

序号	常见错误	错误示例	更正示例

11	复合语句未加{}括号	while(i<=100)
		sum+=i;
		i++;
			while(i<=100){
			sum+=i;
			i++;
			}

12	括号不配对	while((c=getchar()!='q'){
		……
		}
			while((c=getchar())!='q') {
			……
			}

13	标识符的大小写字母不统一	int a,b,c; C=A+B;	int a,b,c; c=a+b;

14	初始化时数组越界错误	void main(){
		char a[5]="goodbye";
		printf("%s\n",a);
		}
			void main(){
			char a[8]="goodbye";
			printf("%s\n",a);
			}

15	定义和引用数组元素的括号不对	int i, a(10);
		…
		scanf("%d",&a(i));
			int i, a[10];
			…
			scanf("%d",&a[i]));

16	将数组元素个数误认为是下标最大值	int a[10];
		…
		for(i=1; i<=10; i++)
		printf("%d\n",a[i]);
		…
			int a[10];
			…
			for(i=1; i<=9; i++)
			printf("%d\n",a[i]);
			…

17	二维数组的引用与定义错误	int a[ ][ ]; … printf("%d",a[2,3]);	int a[3][4]; … printf("%d",a[2][3]);

18	将数组名当作数组的全部元素	void main(){
		static int a[3]={1,3,5};
		printf("%d%d%d\n",a);
		}
			void main(){
			static int a[3]={1,3,5}; int i;
			for(i=0; i<=2; i++)
			printf("%d",a[i]);
			}

19	字符数组与字符指针相混	void main(){
		char str[14];
		str="computer and c";
		printf("%s\n",str); }
			void main(){
			char *str;
			str="computer and c";
			printf("%s\n",str);
			}

20	引用指针变量前未赋值	void main(){
		char *p;
		scanf("%s",p);
		……
		}
			void main(){
			char *p,c[80];
			p=c;
			scanf("%s",p);
			……
			}

(续)表 4.1

序号	常见错误	错误示例	更正示例

21	在 switch 语句中漏写 break 语句	switch(score) {
		case 5:printf("优秀");
		case 4:printf("良好");
		case 3:printf("中等");
		case 2:printf("及格");
		default:printf("不及格");
		}
			switch(score) {
			case 5:printf("优秀"); break;
			case 4:printf("良好"); break;
			case 3:printf("中等"); break;
			case 2:printf("及格"); break;
			default:printf("不及格");
			}

22	字符与字符串相混	char c; c="A";	char c; c='A';

23	使用++,--中所产生的错误	void main(){
		int i, * p,a[5]={1,3,5,7,9};
		p=a;
		for(i=0;i<5;i++)
		printf("%d", * (++p));
		}
			void main(){
			int ,i, * p,a[5]={1,3,5,7,9};
			p=a;
			for(i=0;i<5;i++)
			printf("%d", * p++);
			}

24	将局部变量与形参混在一起,且形、实参数不匹配对应	void main(){
		float x,y,z;
		x=3.5; y=-7.6;
		z=max(x,y); printf("%f\n",z);
		}
		max(int x,int y,int z) {
		z=x>y? x:y;
		return(z);
		}
			void main(){
			float x,y,z;
			x=3.5; y=-7.6;
			z=max(x,y); printf("%f\n",z);
			}
			max(int x,int y){
			int z;
			z=x>y? x:y;
			return(z);
			}

25	所用函数后定义先使用	void main(){
		float x,y,z;
		x=3.5; y=-7.6; z=max(x,y);
		printf("%f\n",z);
		}
		float max(float x,float y) {
		return(x>y? x:y);
		}
			void main(){
			float max(float x,float y);
			float x,y,z;
			x=3.5; y=-7.6; z=max(x,y);
			printf("%f\n",z);
			}
			float max(float x,float y){
			return(x>y? x:y);
			}

26	实参与形参类型不匹配、被调函数返回数据类型不一致等	void main(){
		float a=3, b=4, c;
		c=fun(a,b);
		…
		}
		fun(int x,int y) {
		……
		}
			void main(){
			float fun(float x,float y);
			float a=3, b=4, c;
			c=fun(a,b);
			…
			}
			float fun(float x,float y){
			……
			}

(续)表 4.1

序号	常见错误	错误示例	更正示例

27	不同类型的指针混用	void main(){
		int i=3, *p1;
		float a=1.5, *p2;
		p1=&i; p2=&a;
		p2=p1;
		printf("%d,%d\n", *p1, *p2);
		}
			void main(){
			float i=3, *p1;
			float a=1.5, *p2;
			p1=&i; p2=&a;
			p2=p1;
			printf("%f,%f\n", *p1, *p2);
			}

28	混用数组名与指针变量名,没有注意数组名与指针变量名的区别	void main(){
		int i, a[5];
		for(i=0; i<5; i++)
		scanf("%d",a++)
		…
		}
			void main(){
			int *p, i, a[5];
			p=a;
			for(i=0; i<5; i++)
			scanf("%d",p++);
			… }

29	使用无效的指针,malloc()所返回的地址并未赋给指针 p,且返回的地址有可能空值(NULL)	#include <stdio.h>
		void main(){
		char *p;
		*p=(char *)malloc(100);
		gets(p);
		printf("%s\n",p);
		}
			#include <stdio.h>
			void main(){
			char *p;
			p=(char *)malloc(100);
			if(p==NULL)
			{ printf("out of memoty\n");
			exit(1); }
			gets(p);
			printf("%s\n",p);
			}

第五部分 实验报告

做完实验后要写实验报告，这是整个实验过程的一个重要环节，也是培养科学作风的重要途径。实际上，实验报告是对整个实验的总结和提高，是从感性到理性的升华。因此，每做完一次实验，都要认真书写实验报告，千万不要把它看成是一种不必要的负担，更不能敷衍了事。通过写实验报告，还可以训练和提高自己的写作能力。

一、如何写实验报告

针对"计算机程序设计"课程实验的特点，建议在书写实验报告时应包括如下内容：

1. 实验目的

实验作为教学的一个重要环节，其目的在于更深入地理解和掌握课程教学中的有关基本概念，应用基本技术解决实际问题，从而进一步提高分析问题和解决问题的能力，我们着手做一个实验的时候，必须明确实验的目的，以保证达到课程所指定的基本要求。在写实验报告时，要进一步确认是否达到了预期的目的。

2. 实验内容

在本书中，每一个实验都安排了多个实验题目，根据教学安排、进度、实验条件、可提供的机时、学生的基础等因素，可以选择其中的几个或全部。因此，在实验报告中，实验内容是指本次实验中实际完成的内容。在每一个实验题目中，一般都提出了一些具体要求，其中有些具体要求是为了达到实验目的而提出的。因此，在实验内容中，不仅要写清楚具体的实验题目，还要包括具体要求。

3. 算法与流程图

算法设计是程序设计过程中的一个重要步骤。在本书中，对于某些实验题目给出了方法说明，有的还提供了流程图。如果在做实验的过程中，使用的算法或流程图和书中的不一样，或者书中没有给出算法和流程图，则在实验报告中应给出较详细的算法说明与流程图，并对主要符号与变量作相应的说明。

4. 程序清单

程序设计的产品是程序，它应与算法或流程图相一致。程序要具有易读性，符合结构化原则。

5. 运行结果

程序运行结果一般是输出语句所输出的结果。对于不同的输入，其输出的结果是不同的。因此，在输出结果之前还应注明输入的数据，以便对输出结果进行分析和比较。

6. 调试分析和体会

这是实验报告中最重要的一项，也是最容易忽视的一项。

实验过程中大量的工作是程序调试，在调试过程中会遇到各种各样的问题，每解决一个

问题就能积累一点经验，提高编程的能力。因此，对实验的总结，最主要的是程序调试经验的总结。调试分析也包括对结果的分析、尚存在的问题和拟解决的方法等。

体会主要是指通过本次实验是否达到了实验目的，有哪些基本概念得到了澄清等。

二、实验报告示例

1. 实验目的

(1) 学习算法设计与描述。

(2) 熟悉用 while，do～while 和 for 语句的使用。

(3) 掌握程序设计中循环结构的应用。

2. 题目

(1) 伟大的革命导师马克思曾经解答了下面的题目：(《马克思数学手稿》)有 30 个人，其中有男人，女人和小孩，在一家小饭馆里共花了 50 先令，每个男人花 3 先令，每个女人花 2 先令，每个小孩花 1 先令，问男人、女人和小孩各有多少？

(2) ……(作为示例其余题目略)

3. 分析

设用 x，y，z 分别代表男人、女人和小孩的人数，根据题目叙述，可得到下面的方程组：

$$\begin{cases} x+y+z=30 & (5.1) \\ 3x+2y+z=50 & (5.2) \end{cases}$$

(5.2)式减去(5.1)式就得到

$$2x+y=20 \tag{5.3}$$

要解决这个问题，就是要求出(5.3)式的非负整数解。

由二元一次不定方程的定理，可设二元一次不定方程

$$ax+by=c \quad (a,\ b,\ c\ \text{为正整数}) \tag{5.4}$$

可设其有一组整数解 $x=x_0$，$y=y_0$，则方程的一切整数解可以表示成

$$\begin{cases} x=x_0-bt \\ y=y_0+at \end{cases}$$

其中，$t=0$，±1，±2，±3，……

在本题中，很容易看出一组整数解是：$x=20$，$y=-20$，那么，一切整数解可以表示为 $x=20-t$，$y=-20+2t$，又因为 x 及 y 代表男人、女人的人数，所以必须使得 $x\geqslant0$，$y\geqslant0$。由 $x=20-t\geqslant0$，而得 $t\leqslant20$，由 $y=-20+2t\geqslant0$，而得 $t\geqslant10$，又有

$$10\leqslant t\leqslant 20 \tag{5.5}$$

又因为小孩的个数由(5.1)式决定，故有 $z=30-x-y=30-t$，由 $x=20-t$，$y=2t-20$，$z=30-t$ 和(5.5)式就可以得到一组解。

4. 设计思路(算法描述)

算法 1：根据上面的分析可得如下伪码描述的算法：

```
BEGIN
    T=10
    WHILE(t<=20)
        BEGIN
```

```
            X=20-t
            Y=2*t-20
            Z=30-t
            T=t+1
            输出 x, y, z
        END
    END
```

算法 2:该问题类似百鸡问题,故可采用枚举法求解。

直接利用二重循环,令外循环变量为 x(整型变量),从 0 逐次加 1 循环到 17($17\times3\geqslant50$);内循环变量为 y(整型变量),从 0 逐次加 1 循环到 25($25\times2\geqslant50$)。在内层循环体中,令 $z=30-(x+y)$,判断 $2x+2y+z=50$ 是否成立,如果满足就打印 x,y,z,如果不满足继续循环。

用 PAD 图描述得算法如图 5.1 所示。

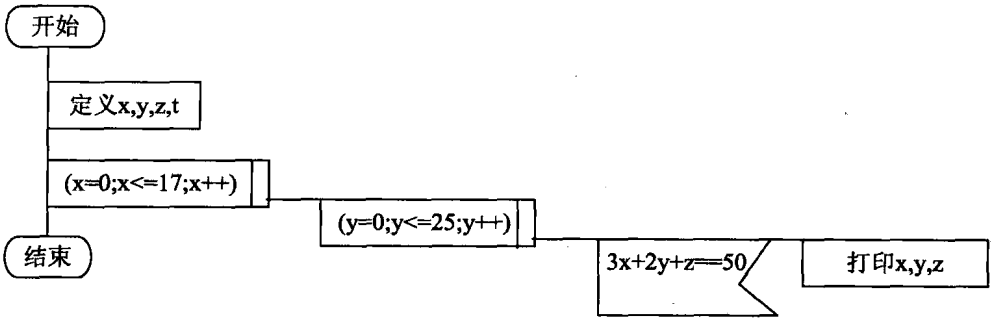


图 5.1 算法 2 的 PAD 图

5. 程序清单

源程序文件 1:

```
/*ex41.c*/
#include <stdio.h>
main()
{ int x=0,y=0,z=0,t;
  t=1;
   while(t<=20)
    { x=20-t;
      y=2*t-20;
      z=30-t;
      t++;
      printf("x=%2d  y=%2d  z=%2d\n",x,y,z);
    }
}
```

源程序文件 2：

```
/*ex42.c*/
#include <stdio.h>
main()
{ int x,y,z;
  for(x=0;x<=17;x++)
    for(y=0;y<=25;y++)
      { z=30-x-y;
        if(3*x+2*y+z==50)
            printf("x=%2d  y=%2d  %2d\n",x,y,z);
      }
}
```

6. 运行结果

运行结果如下：

```
x= 0  y=20  z=10
x= 1  y=18  z=11
x= 2  y=16  z=12
x= 3  y=14  z=13
x= 4  y=12  z=14
x= 5  y=10  z=15
x= 6  y= 8  z=16
x= 7  y= 6  z=17
x= 8  y= 4  z=18
x= 9  y= 2  z=19
x=10  y= 0  z=20
```

7. 调试分析与体会

(1) 循环体内的语句多于一个时必须使用复合语句。开始 while 循环体没有采用复合语句形式，初次运行就进入死循环且无输出，中断程序执行退出后分析：循环体应采用复合语句形式，否则循环变量不能增值，y 与 z 变量也不能正确变化。修改后，运行正确。

(2) 本题采用了两种方法，分别根据数学定理分析求解和根据问题的特性求解，两种方法运行结果一致。

(3) 本题没有考虑三种人中某种人为零的情况，若要考虑此类情况，应在循环体内作选择判断的处理，以控制输出结果。

(4)通过本次实验，基本掌握了循环控制结构的知识及其应用编程。

三、关于实验预习报告(作业)

实验预习报告(作业)是上机实验前的一项必要的准备工作，是理解本次实验所体现概念的自我测试，也是学习程序设计中采用的一种人工静态训练方法。认真做好实验预习报

告有助于明确每项实验的学习目的和要求；有助于准确地安排和充分地利用上机实验时间提高实验效率；有助于逐步提高编程技术的熟练程度。

在本书的实验部分，有些实验作业要求写出习题的答案，另一些则要求通过C编译器来完成。要求在实验预习报告中：

(1) 对于要求写出答案的作业，请参照实验手册中相关部分，把题目和答案写在实验预习报告相应位置上，并做好上机验证方案。

(2) 对于编程作业，事先编写好源程序，并经人工静态检查认为无误；同时准备好运行程序所需的典型数据，以备上机实验时验证使用。

(3) 在每个题目之间留出一定的空间，以便纪录上机调试情况。如程序运行过程中的程序提示、显式在屏幕上的输入数据和输出数据，调试过程中的错误信息和问题、原因分析、解决方法，经验总结和验证结论等。

(4) 及时修正实验预习报告中出现的错误，以备每次实验结束后参照预习报告总结给出本次实验的实验报告。

实验预习以实验二为例，给出部分的示范说明，供参考。

【作业首页】

姓名_____ 日期_____

实验任务：_____

实验内容：

1. 修改范例 prg2_1.c，用 scanf()函数输入数据而不用赋值语句，scanf()函数调用语句如下：

```
scanf("%d%d%c%c%f%f",&a,&b,&c1,&c2,&d,&e);
scanf("%lf,%lf,%ld,%ld,%u,%u",&f,&g,&m,&n,&p,&q);
```

输入的数据如下：

```
61 62ab3.56 -6.78↙
3157.890121, 0.123456789,50000,-60000,32768,4000↙
```

应输出如下结果：

```
sizeof:4, a=61, b=62
sizeof:1, c1=a, c2=b
sizeof:4, d=3.56, e=-6.78
sizeof:8, f=3157.890121, g=0.123456789000
sizeof:4, m=50000, n=-60000
sizeof:4, p=32768, q=4000
```

要求：运行修改后的程序并分析运行结果。然后将 scanf()函数中的%lf 和%ld 改为%f 和%d，运行程序并观察分析结果。

按题意有解：

```
main(){
  int a,b;
  char c1,c2;
```

```
    float d,e;
    double f,g;
    long m,n;
    unsigned int p,q;
    a=61; b=62;
    scanf("%d%d%c%c%f%f",&a,&b,&c1,&c2,&d,&e);
    scanf("%lf,%lf,%ld,%ld,%u,%u",&f,&g,&m,&n,&p,&q);
    printf("sizeof:%d, a=%d, b=%d\n",sizeof(int),a,b);
    printf("sizeof:%d, c1=%c, c2=%c\n",sizeof(char),c1,c2);
    printf("sizeof:%d, d=%-6.2f, e=%-6.2f\n",sizeof(float),d,e);
    printf("sizeof:%d, f=%-15.6f, g=%-15.12f\n",sizeof(double),f,g);
    printf("sizeof:%d, m=%ld, n=%ld\n",sizeof(long),m,n);
    printf("sizeof:%d, p=%u, q=%u\n",sizeof(unsigned),p,q);
}
```

输入：

```
61 62ab3.56 -6.78
3157.890121,0.123456789,50000,-60000,32768,4000
```

应输出：

```
sizeof:4, a=61, b=62
sizeof:1, c1=a, c2=b
sizeof:4, d=3.56, e=-6.78
sizeof:8, f=3157.890121, g=0.123456789000
sizeof:4, m=50000, n=-60000
sizeof:4, p=32768, q=4000
```

留出足够的空间，用以记录上机调试情况。如程序运行过程中的程序提示，显示在屏幕上的输入数据和输出数据，调试过程中的错误信息和问题，原因分析，解决方法，经验总结和验证结论等。

2. 输入并运行以下程序：

```
#include <stdio.h>
main(){
    int i,j,m,n;
    i=8; j=10;
    m=++i; n=j++;
    printf("%d,%d,%d,%d\n",i,j,m,n);
}
```

应得结果：9,11,9,10

然后分别作以下改动并运行：

(1) 将第 4 行改为：

```
m=i++; n=++j;
```

应得结果:9,11,8,10

(2) 程序改为:

```
main(){
    int i,j,m,n;
    i=8; j=10;
    printf("%d,%d\n",i++,j++);
}
```

应得结果:8,10

……(留待读者自己完成)

要求:在上机前先用人工分析程序写出应得结果,上机后将两者对照。

留出足够的空间,用以记录上机调试情况。如程序运行过程中的程序提示,显示在屏幕上的输入数据和输出数据,调试过程中的错误信息和问题,原因分析,解决方法,经验总结和验证结论等。

3. 求下列算术表达式的值。要求在上机实验前作静态计算,并写好验证程序,上机后验证,对比结果,分析原因。

(1) 3.5+1/2+56%10

解: printf("%f\n",3.5+1/2+56%10);

表达式的值应为:9.5

(2) (a++ * 1/3)　　设 a=2

解: int a=2;

```
printf("%d\n",(a++ * 1/3));
```

表达式的值应为:0

(3) x+a%3 * (int)(x+y)%2/4　　设 x=2.5, a=7, y=4.7

解: int a=7; float x=2.5,y=4.7;

```
printf("%f\n",x+a%3 * (int)(x+y)%2/4);
```

表达式的值应为:2.5

……(留待读者自己完成)

4. 写出下列各逻辑表达式的值。设 a=3,b=4,c=5。要求在上机实验前做静态计算,并写好验证程序,上机后验证,对比结果,分析原因。

……(留待读者自己完成)

5. 写出下列表达式运算后 a 的值。设原来 a=12。要求在上机实验前作静态计算,并写好验证程序,待上机后验证,对比结果,分析原因。

……(留待读者自己完成)

6. 根据下列计算公式,分别编写程序实现各相应功能,并验证结果:

(1) 球的体积 $v=\frac{4}{3}\pi r^3$

解:
```
#include <stdio.h>
#include <math.h>
#define PI  3.1415926
main(){
```

```
    double r,v=0;
    scanf("%lf",&r);
    v=4.0/3.0*PI*pow(r,3.0);
    printf("%lf\n",v);
}
```

验证数据：

输入:1　　应输出:4.188790

输入:1.5　　应输出:14.137167

……

留出足够的空间，用以记录上机调试情况。如程序运行过程中的程序提示，显示在屏幕上的输入数据和输出数据，调试过程中的错误信息和问题，原因分析，解决方法，经验总结和验证结论等。

……(其余各题留给读者自己完成)

第六部分　综合测试题

综合测试题一

【说明】 试卷中的"□"表示空一格，答题必要时也用"□"表示一个空格。

一、填空题(表达式求值)(本大题共 10 小题，每小题 1 分，共 10 分)　设各语句的初始化相同：int x=3,y=2,z=1;

1. x=y==z;　　x=(　　)
2. x=!(z>y)&&!x||1;　　x=(　　)
3. x=(y++*1/3);　　x=(　　)
4. x=((x>y>z)? 1:0);　　x=(　　)
5. x*=2+3;　　x=(　　)
6. x=(++z*y,y++,z%y);　　x=(　　)
7. y=--z&&--x;　　x=(　　)
8. x=5||x!=y;　　x=(　　)
9. x=1.5+7%3*(int)(2.5+4.7)%2/4;　　x=(　　)
10. x='a'-'A'　　x=(　　)

二、阅读下列程序段，指出错误并改正(本大题共 5 题小题，每小题 4 分，共 20 分)

说明：在错误处加下划线，然后将改正内容写在行末。

1.
```
#include <stdio.h>
#define MAX(a,b)   (a>=b? a:b)
void main( ){
    int i;
    char a[6]={'a','1','b','2','c','3'};
    char b[6]="234def";
    for(i=0; a[i]!='\0'; i++)
        printf("i=%d,max=%c\n",i, MAX(a[i],b[i]));
}
```

2.
```
#include <stdio.h>
void main( ){
    int a=9; double m;
    m=f1(3.14159, a);
    printf("%f\n",m);
```

```
}
double f1(int i, double j){
    return(i*i+j*j);
}
```

3.
```
#include <stdio.h>
void main( ){
    int i; char *str[5];
    for(i=0; i<5; i++) scanf("%s",str[i]);
    for(i=0; i<5; i++) printf("%s",str[i]);
}
```

4.
```
#include <stdio.h>
void main( ){
    int i, r[10],sum;
    for(i=0; i<=10; i++){
        r[i]=i+1;
        sum+=r[i];
    }
    printf("sum=%d",sum);
}
```

5.
```
#include <stdio.h>
void main( ){
    int i;
    struct student{
    int num=0;
    float score1=0, score2=0, aver=0;
    } stu[3]={{200401,92,83},{200402,85,70},{200403,65,89}};
    for(i=0; i<3;i++){
        stu[i].aver=(stu[i].score1+stu[i].score2)/2;
        printf("%d,%f\n",stu[i].num,stu[i].aver);
    }
}
```

三、阅读分析下列程序，写出运行结果(本大题共 7 小题，每小题 4 分，共 28 分)

1.
```
#include <stdio.h>
void main( ){
    int i,f1=1,f2=1;
    for(i=1;i<=4;i++){
        printf("%6d%6d",f1,f2);
        if(i%2==0)
            printf("\n");
```

1. 运行结果：

```
        f1=f1+f2;
        f2=f2+f1;
    }
}
```

2.
```
#include "stdio.h"
void main( ){
int i,x;
for(i=1, x=1; i<7; i++){
    if(x>=7) break;
    if(x%2==1){
        x+=5;continue;
    }
    x-=3;
    printf("i=%d,x=%d\n",i,x);
    }
    printf("i=%d,x=%d\n",i,x);
}
```

2. 运行结果：

3.
```
#include <stdio.h>
fun(int a){
    int b=1;
    static  int c=1;
    a++; b++; c++;
    return(a+b+c);
}

void main( ){
    int i=1;
    for(  ; i<=3; i++)
    printf("%6d",fun(i));
}
```

3. 运行结果：

4.
```
#include <stdio.h>
void main( ){
    int i,j;
    char c, a[ ]="computer", *p=a;
    printf("%s",a);
    for(i=0,j=7; i<j; i++,j--){
        c=p[i];
        *(p+i)=*(p+j);
        p[j]=c;
    }
```

4. 运行结果：

```
    printf("->%s",a);
}
```

5.
```
#include <stdio.h>
void swap2(int *pa, int *pb){
    int *t;
    t=pa; pa=pb; pb=t;
}
void swap3(int *pa, int *pb){
    int t;
    t=*pa; *pa=*pb; *pb=t;
}
void main( ){
    int x=1,y=3, *p1=&x, *p2=&y;
    swap2(p1,p2); printf("%d,%d\n",x,y);
    swap3(p1,p2); printf("%d,%d\n",x,y);
}
```

5. 运行结果：

6.
```
#include "stdio.h"
void main( ){
    static struct st1{
        char c[4], *s;
    }s1={"abc","def"};
    static struct st2{
        char *cp;
        struct st1 ss1;
    }s2={"ghi",{"jkl","mno"}};
    printf("%c\t%c\n",s1.c[0], *s1.s);
    ++s2.cp;
    ++s2.ss1.s;
    printf("%s\t%s\n",s2.cp, s2.ss1.s);
}
```

6. 运行结果：

7. 有以下程序，编译后生成可执行程序文件display。写出执行“display how are you!”命令行后的结果。

```
#include <stdio.h>
main(int argc, char *argv[ ]){
    int i;
    for(i=1; i<argc; i++)
        printf("%s%c", *++argv,(i<argc-1? ' ':'\n'));
}
```

7. 运行结果：

四、程序填空题(本大题共 2 小题,每空 1 分,共 15 分)

1. 以下程序功能为建立链表及统计链表中所有节点的数据域 n 之和。

```
#include  <stdio.h>
#include  <malloc.h>
struct node{
    int n;
    struct node *next;
};
struct node *creat( ){                                  //建立链表
    struct node *head, *p1, *p2;
    head=____________________________________;
    p1=p2=___________________________________;
    scanf("%d", &p1->n);
    while( p1->n!=0 ){
        if( head==NULL )
          head=__________________________________;
        else
            p2->next=____________________________;
        p2=p1;
        p1=______________________________________;
        scanf("%d",&p1->n);
    }
    p2->next=________________________________;
    return(_______________ );
}
add(_____________________ ){                    //统计链表中所有节点的数据域 n 之和
    struct node *p; int n=0;
    p=head;
    while(p!=NULL){
        n=_______________________________________;
        p=_______________________________________;
    }
    return(n);
}                                               //主函数 main()见后第 7 页
main( ){
    struct node *head;  int m;
    head=creat( );
    m=add( head );
    printf("%d\n", m);
}
```

2. 已知正文有文件“abc. txt”，本题统计该文件中词的个数并求出最长词的长度。正文文件中，词与词之间由空格符，制表符，换行符分隔。程序中，word 存放词的个数，maxLen 存放最长词的长度，length 存放当前词的长度，flag 为状态标记（flag=1 表示当前字符在单词中，flag=0 表示当前字符不在某词中）。

```
#include <stdio. h>
#include <process. h>
void main( ){
    FILE *fp; char ch;
    int word=0,maxLen=0, length=0, flag=0;
    if((fp=fopen( ________________________ ))==NULL){
        printf("cannot open the file! \n");
        exit(0);
    }
    ch=fgetc(fp);
    while(ch!=EOF){
        if(ch=='□' || ch=='\t' || ch=='\n'){          //'□'表示空格
            if(flag==1){
                flag=0;
                if(length>maxLen) ________________________;
            }
        }
        else if(flag==0){
            flag=_____________________;
            length=1;
            word _____________________;
        }
        else
            length++;
        ch=_____________________;
    }
    fclose(fp);
    printf("word=%d,maxLen=%d\n",word,maxLen);
}
```

五、程序设计题（本大题共 2 小题，第 1 小题 12 分，第 2 小题 15 分，共 27 分）

1. 根据主函数的定义，编写以下两个函数：

(1) 求数组 a[4][4]主对角线上元素的和；(6 分)

(2) 实现将数组 a[4][4]的矩阵转置处理。(6 分)

```
void main(){
```

```
    static int a[4][4]={1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4};
    int *p, i, sum;
    sum=total( a, 4 );              //调用 total 函数求主对角线上元素的和
    printf("%d\n",sum);
    p=*a;
    convert( p );                   //调用 convert 实现矩阵转置
    for(i=0; i<4; i++)
        printf("%d %d %d %d\n",a[i][0],a[i][1],a[i][2],a[i][3]);
}
```

2. 编一个程序，用结构体数组描述 60 位学生的信息，包括：学号、姓名、三门课的成绩和总成绩。并以函数形式实现以下功能：(15 分)

(1) 输入 60 位学生的前五项信息，并计算每个学生的总成绩。

(2) 按总成绩递减顺序对成绩表排序。

(3) 输出排序后的成绩表。

综合测试题二

一、选择题(本大题共 10 小题，每小题 1 分，共 10 分)

1. C 语言中用于结构化程序设计的三种基本结构是_____。

A. 顺序结构、选择结构、循环结构

B. if、switch、break

C. for、while、do-while

D. if、for、continue

2. 下列关于 C 语言用户标识符的叙述中正确的是_____。

A. 用户标识符中可以出现下划线和中划线

B. 用户标识符中不可以出现中划线，但可以出现下划线

C. 用户标识符中可以出现下划线，但不可以放在用户标识符的开头

D. 用户标识符中可以出现下划线和数字，他们都可放在用户标识符的开头

3. 有以下程序：

```
main(){
    char a,b,c,*p;
    a='\'; b='\xbc'; c='\0xab'; p="\0127";
    printf("%c %c %c %c\n",a,b,c,*p);
}
```

编译时出现错误，以下叙述中正确的是_____。

A. 程序中只有 a='\';语句不正确

B. b='\xbc';语句不正确

C. c='\0xab';语句不正确

D. a='\';和 c='\0xab';语句都不正确

4. 有以下程序段：

```
int m=0,n=0; char c='a';
scanf("%d%c%d",&m,&c,&n);
printf("%d,%c,%d\n",m,c,n);
```

若从键盘上输入：12A10〈回车〉，输出结果是_____。

A. 12,A,10　　B. 12,a,10　　C. 12,a,0　　D. 12,A,0

5. 设有定义：double x;，则以下正确的输入语句是_____。

A. scanf("%f",x);　　B. scanf("%f",&x,);

C. scanf("%lf",&x);　　D. scanf("%lf",x);

6. 有以下程序：

```
void main(){
    char *s[]={"one","two","three"},*p;
        p=s[1];
        printf("%c,%s\n",*(p+1),s[0]);
}
```

执行后的结果是_____。

A. n,two　　B. w,one　　C. t,one　　D. o,two

7. 设已有定义：char a[10]，*p=a;下面的赋值语句中，正确的是_____。

A. a[10]="Turbo C";　　B. a="Turbo C";

C. p="Turbo C";　　D. *p="Turbo C";

8. 设有已定义：int n=100，*p=&n，*q=p;则以下不正确的赋值语句是_____。

A. p=q;　　B. *p=*q;　　C. p=n;　　D. n=*q;

9. 主调函数中已有定义：int a[3][4],n; 当使用函数调用语句：fun(a,n);时，被调函数fun()的形参说明表列正确的为_____。

A. fun(int a[3][],int n){…}　　B. fun(int a[][4],int n){…}

C. fun(int a[][],int n){…}　　D. fun(int *p[4],int n){…}

10. C程序由函数组成，关于C语言函数的说法错误的是_____。

A. 除整型函数外，其他类型函数必须在定义时给予类型说明

B. 函数原型说明语句必须给出函数类型的说明

C. 函数类型决定返回值的类型

D. 函数返回值的类型就是return语句中的表达式值的类型

二、填空题(本大题共8小题共10个填空，每空1分，共10分)

1. 试将数学公式$\left|\cos\frac{\pi}{2}\right|$写成C表达式为：

_____

2. 设x,n已定义为double类型变量，试将数学公式$\frac{3x^n}{2x-1}$写成C表达式为：

_____

3. 若有定义:int a=7; float x=2.5,y=4.7;
   执行 printf("%f\n", x+a%3*(int)(x+y)%2/4); 的结果为:_____
4. 若有定义:int a=3,b=4;
   执行 printf("%d\n",(a=++b,a+5,a/5));的结果为:_____
5. 若有定义:int a=3,b=4,c=5;
   执行 printf("%d\n",a*=b+c);的结果为:_____
6. 若有定义:int p, a=5;
   执行　if(p=a!=0)printf("%d\n",p);
   　　else printf("%d\n",p+2);
   的结果为:_____
7. 若有定义:int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12},(*p)[4]=a;
   执行 printf("%d\n",*(*(p+1)+3)); 的结果为:_____
8. 某地刑侦大队对涉及 6 个嫌疑人的一桩疑案进行分析:
   ① A,B 至少有一人作案;
   ② A,E,F 3 人中至少有 2 人参与作案;
   ③ A,D 不可能是同案犯;
   ④ B,C 或同时作案,或与本案无关;
   ⑤ C,D 中有且仅有一人作案;
   ⑥ 如果 D 没有参与作案,则 E 也不可能参与作案。

为解这道题,需要将案情的每一条写成逻辑表达式,如变量 A 表示作案,!A 表示不作案,从第 1 条到第 6 条依次用 cc1,……,cc6 表示,请参照以下已经给出的第一个赋值语句形式,请填写余下的 5 个赋值语句中空缺的逻辑表达式:

```
cc1=(A || B);                          // A,B 至少有一人作案
cc2 =____________________________;     // A,E,F3 人中至少有 2 人参与作案
cc3 =____________________________;     // A,D 不可能是同案犯
cc4 =____________________________;     // B,C 或同时作案,或与本案无关
cc5 =____________________________;     // C,D 中有且仅有一人作案
cc6 =____________________________;     //如果 D 没有参与作案,则 E 也不可能参与
                                         作案
```

三、阅读并分析下列程序,写出运行结果(本大题共 5 小题,每题 6 分,共 30 分)

1.
```
#include <stdio.h>
void main(){
    int a[4][4]={{1,2,-3,-4},{0,-12,-
                 13,14},{-21,23,0,-24},
                 {-31,32,-33,0}};
    int i,j,s=0;
    for(i=0;i<4;i++){
        for(j=0;j<4;j++){
```

1. 运行结果:

```
            if(a[i][j]<0) continue;
            if(a[i][j]==0) break;
            s += a[i][j];
        }
        printf("%d\n",s);
    }
}
```

2.
```
#include <stdio.h>
#define UDF_SWAP(a,b) t=a;a=b;b=t;
void udf_swap(int a,int b){
    int t;t=a;a=b;b=t;
}
void main(){
    int x=2,y=3,t=0;
    UDF_SWAP(x,y);
    printf("%d  %d\n",x,y);
    x=2,y=3;
    udf_swap(x,y);
    printf("%d  %d\n",x,y);
}
```

2. 运行结果：

3.
```
#include <stdio.h>
void mul(void){
    int i=1; static int a=1;
    while(i<=a){
        printf("%5d",a*i); i++;
    }
    a++; printf("\n");
}
void main(){
    int a,b; for(a=1;a<4;a++)mul();
}
```

3. 运行结果：

4.
```
#include <stdio.h>
void main(){
    int i=0,CountX=0,CountY=0;
    char str[80]="The □Programing □Language";
    while(str[i]!='\0'){
        if(str[i]>='a'&&str[i]<='z')
            str[i]-=32;
        switch(str[i]){
```

4. 运行结果：

```
            case 'A':
            case 'E':
            case 'I':
            case 'O':
            case 'U': CountX++;break;
            default : CountY++;break;
        }
        i++;
    }
    printf("CountX=%d\n",CountX);
    printf("CountY=%d\n",CountY);
}
```

5.
```
#include <stdio.h>
#include <malloc.h>
char *fun1(char *a,int start,int len){
    char *ans; int i;
    ans=(char *)malloc(strlen(a)+1);
    for(i=start;i<start+len;i++)
        ans[i-start]=a[i];
    ans[len]='\0';
    return(ans);
}
void fun2(char *s,char *t){
    while(*s!='\0')s++;
    while((*s++=*t++)!='\0');
}
void main(){
    char a[80]="ABCDEFGHIJ", b[80]="12345678910", *p;
    int i,j;
    for(i=0,j=1;i<2;i++){
        p=fun1(a,i,++j);
        printf("%s\n",p);
        fun2(p,fun1(b,i,j));
        printf("%s\n",p);
    }
}
```

5. 运行结果：

四、程序填空题（本大题 2 题共 5 个填空，每空 2 分，共 10 分）

1. 下面的程序把从终端读入的文本复制到一个名为 file1. dat 的新文件中。请在划线

处填空。

```
#include <stdio.h>
#include <process.h>
void main(){
    FILE *fp; char ch;
    if((fp=fopen(_____________________________))==NULL)
        exit(0);
    while((ch=getchar())!=EOF)
        ____________________________________;
    fclose(fp);
}
```

2. 以下程序，数组 a 中存放一个递增数列。输入一个整数 x 并将它插入到数组 a 中，使之仍为一个递增数列。请在划线处填空。

```
#include <stdio.h>
void main(){
    int a[ _________ ]={1,10,20,30,40,50,60,70,80,90};
    intx,i,p;
    scanf("%d",&x);
    for(i=0,p=10;i<10;i++)
        if(x<a[i]){
            p=i;
            ____________________________________;
        }
    for(i=9;i>=p;i--)
        a[i+1]=a[i];
    ____________________________________;
    for(i=0;i<=10;i++)
        printf("%5d",a[i]);
    printf("\n");
}
```

五、程序设计题(本题共 2 小题，第 1 题 25 分，第二题 15 分，共 40 分)

1. 编程实现以链表形式处理学生选修课程信息。每一位学生选修课程的记录信息包括：学号 sno、课程号 cno，成绩 score；相应数据项如："PB07210001"，"C0020101"，95.0。请按要求编写各指定功能函数。相应预处理命令、数据类型、主函数框架等已给出，请按要求编写各指定功能函数。

```
#include <stdio.h>
#include <malloc.h>
#include <process.h>
```

```
struct student{                          //结构体类型定义
    char sno[16], cno[12];
    float score;
    struct student *next;
};
main(){                                  //主函数供编程参考
    struct student *head=NULL, *s;
    char search_sno[16];
    head=create();                       //调用 create()建立链表
    printf("输入查询学号:\n");
    scanf("%s",search_sno);
    serch_print(head,search_sno);        //调用 serch_print()检索选修信息
}
```

(1) 编写一个计算字符串长度的 str_len( )函数,要求为:函数的返回值为字符串长度。(4 分)

(2) 编写一个字符串比较 str_cmp( )函数,设有字符串 s 与 t,若 s 大于 t 则返回正值,s 等于 t 则返回 0,s 小于 t 则返回负值。(5 分)

(3) 编写一个建立链表的 create( )函数,要求为:建立结点个数不限,当输入记录的学号字符串长度小于等于 1 时结束,如输入:0　0　0 时结束;函数的返回值为指向链表 head 指针。(10 分)

(4) 编写一个按学号检索学生选修课程信息的 serch_print( )函数,要求:能按指定学号检索该学生所有的选修课程信息,即打印出所有与指定学号相关结点的成员信息。(6 分)

2. 如图 6.1 所示,已知某游泳池的长度为 50 米,宽度为 25 米。某人游泳速度为 0.8 米/秒,步行速度为 1.2 米/秒。编程求解从 A 点到 B 点间何处下水游到 C 点时间最短,输出该点距 A 点的长度 x 和所求最短时间 y。要求:

(1) 给出求解该问题的算法描述。(7 分)

(2) 根据算法编写完整的求解程序。(8 分)

说明:距离精度控制在 0.1 米或以下。

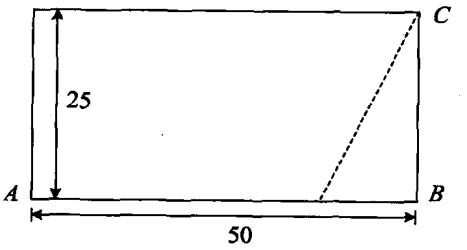


图 6.1　游泳池入水点问题示意图

综合测试题三

一、选择题(本大题共 10 小题,每小题 1 分,共 10 分)

1. 与数学公式 $\frac{3x^n}{2x-1}$ 对应的 C 语言表达式是_____。

A. 3 * x^n/(2 * x−1)　　B. 3 * x * * n/(2 * x−1)

C. 3 * pow(x,n)/(2 * x−1)　　D. 3 * pow(n,x) * (2 * x−1)

2. 设 a 为整型变量,不能正确表达数学关系 10<a<15 的 C 语言表达式为_____。

A. 10<a<15　　B. a==11 || a==12 || a==13 || a==14

C. a>10&&a<15　　D. !(a<=10)&&!(a>=15)

3. 下列关于 if 语句结构说法错误的是_____。

if(表达式)语句 1; else 语句 2;

A. "表达式"可以是任何表达式

B. 当语句 2 为空时,"else 语句 2;"可以省略

C. 语句 1 或语句 2 可以是复合语句,不可以是空语句

D. else 总是与同一语法层次中离它最近的尚未配对的 if 配对

4. 若已定义 x 和 y 为 double 型,x=1,则表达式 y=x+3/2 的值是_____。

A. 1　　B. 2.0　　C. 2　　D. 2.5

5. 若 int a[]={1,2,3,4,5,6,7,8,9,10}, *p=a; 则值为 3 的表达式是_____。

A. p+=2 , *p++　　B. p+=2 , *++p

C. p+=3 , *p++　　D. p+=2 ,++*p

6. 设 int (*ptr)[10]; 其中的 ptr 是_____。

A. 10 个指向整型变量的指针

B. 指向 10 个整型变量的函数指针

C. 一个指向具有 10 个元素的一维数组的指针

D. 具有 10 个指针元素的整型数组

7. 设有定义:double x;,则以下正确的输入语句是_____。

A. scanf("%f",x);　　B. scanf("%f",&x,);

C. scanf("%lf",x);　　D. scanf("%lf",&x) ;

8. 字符串常数"xiaoli"在内存占用的字节数是_____。

A. 6　　B. 7　　C. 8　　D. 32

9. C 程序由函数组成,关于 C 语言函数的说法错误的是_____。

A. 除整型函数外,其他类型函数必须在定义时给以类型说明

B. 函数原型说明语句必须给出函数类型的说明

C. 函数不仅可以递归调用,也可以递归定义

D. 函数类型决定返回值的类型

10. 以下程序中调用 scanf 函数给变量 a 输入数值的方法是错误的,之所以是错误的,其原因是_____。

```
main(){
    int a, *p, *q, b; p=&a;
    printf("input a:");
    scanf("%d", *p);
    ......
}
```

A. *p 表示的是指针变量 p 的地址

B. *p 表示的是变量 a 的值,而不是变量 a 的地址

C. *p 表示的是指针变量 p 的值

D. *p 只能用来说明 p 是一个指针变量

二、填空题(本大题共 8 小题共 10 个填空,每空 1 分,共 10 分)

1. 变量存储类别,从作用域角度分有_____ 变量和_____变量。

2. 设整型变量 x, y, z 均为 5:执行 x - = y-z 后,x = _____ 。

3. 设 m, n, a, b, c, d 均为 0, 执行 (m=a==b) ‖ (n=c==d) 后,m, n 的值分别为:_____ ,_____。

4. 若有定义:int a=1, b=2;

执行 printf("%d\n", (a=++b,a+5,a/5));的结果为_____ 。

5. 设 ch 是 char 型变量,其值为'A',则执行语句

ch=(ch>='A'&& ch<='Z')? (ch+32):ch 后,ch 的值为_____ 。

6. 若有定义:int x=3,y=5;

执行 printf("%d\n",(x>=y>=2)? 1:0);的结果为_____。

7. 若定义 i , j 为 int 型,则以下程序段中内循环体的执行次数为_____ 。

```
for (i=5; i ; i-- )
    for (j=0; j<4 ; j++ )
     { … }
```

8. 设有定义语句

```
struct{
    int x;
    inty;
}d[2] = {{ 1, 3 },{ 2, 7}};
```

则 printf("%d\n",d[0]. y/d[0]. x * d[1]. x) 的输出是_____。

三、阅读程序题:请仔细阅读、分析下列程序,按程序实际执行情况,写出运行结果(本大题共 7 小题,每题 5 分,共 35 分)

1.
```
#include <stdio. h>
void main(){
    int a,b;
    for(a=b=1;a<=100;a=a+1){
        if(b>=10){
          printf("%d\n",b); break;
        }
        if(b%3==1){
            printf("%d\n",b=b+3);
            continue;
        }
    }
    printf("%d\n",a);
```

1. 运行结果:

```
}
```

2.
```
#include <stdio.h>
void main(){
    int i=0,j,n=7788,base=8,num[20];
    do{ i++;
        num[i]=n%base;
        n=n/base;
    }while(n! =0);
    for(j=i;j>=1;j--) printf("%d",num[j]);
    printf("\n");
}
```

2. 运行结果：

3.
```
#include<stdio.h>
int n=2;
int fun(int n){
    int a=1;static int m=5;
    m = n + m + a;
    printf("%4d%4d\n",n,a);
    return(m);
}
void main(){
    int a=1;
    printf("%4d\n",fun( n+fun(a)));
}
```

3. 运行结果：

4.
```
#include <stdio.h>
void main(){
    int k=1; char c='A';
    do{
        switch(++c){
            case 'A': k++;printf("%4d",k); break;
            case 'B': k--;printf("%4d",k);
            case 'C': k+=2; printf("%4d",k); break;
            case 'D': k%2;printf("%4d",k); break;
            case 'E': k=k*2; printf("%4d",k); break;
            default : k=k/3; printf("%4d",k);
        }
    }while(c<'E');
    printf("\n");
}
```

4. 运行结果：

5. #include<stdio.h>

```
void write_it( char *s ){
    printf("%s\n",s);
    if( *s! ='a')
        write_it( s+1 );
    putchar( *s );
}
void main(){
    char str[40]="image";
    write_it( str );
    printf("\n");
}
```

5. 运行结果：

6. #include <stdio.h>

```
#define SWAP(x,y) temp=x; x=y; y=temp
void swap(int *p, int *q){
    int temp;
    temp= *p; *p= *q; *q=temp;
}
void main(){
    int a=1,b=3, *p=&a, *q=&b, *temp;
    printf("%d,%d\n", *p, *q);
    SWAP(p,q); printf("%d,%d\n", *p, *q);
    swap(p,q); printf("%d,%d\n", *p, *q);
}
```

6. 运行结果：

7. #include <stdio.h>

```
void main(){
    int a[2][3]={{1,2,3},{4,5,6}};
    int b[3][2]={{3,2},{2,0},{1,4}};
    int i,j,k,sum=0,c[2][3];
    for(i=0;i<2;i++)
        for(j=0;j<2;j++)
            for(c[i][j]=k=0;k<3;k++)
                c[i][j]+=a[i][k]*b[k][j];
    for(i=0;i<2;i++){
        for(j=0;j<2;j++){
            printf("%d\t",c[i][j]);
            if(i==j || i+j==1)sum+=c[i][j];
        }
        printf("\n");
```

7. 运行结果：

```
    }
    printf("%d\n",sum);
}
```

四、程序填空题（本大题含 3 小题共 15 分，每空 1 分）

1. 下面程序的功能是通过调用函数 f 计算代数多项式当 a=1.7 时的值。代数多项式为：1.1+2.2*a+3.3*a*a+4.4*a*a*a+5.5*a*a*a*a，根据程序功能填空。

```
#include <stdio.h>
    double f(double x, ____________________________ ){
    double y=a[0],t=1;
    int i;
    for(i=0;i<5;i++){
        t= ____________________________ ;
        y=y+a[i]*t;
    }
    return(____________________________ );
}
void main(){
    double b[5]={ 1.1, 2.2, 3.3, 4.4, 5.5 };
    printf("%lf\n",f( 1.7, b, 5 ));
}
```

1				
1	1			
1	2	1		
1	3	3	1	
1	4	6	4	1
…	…	…	…	…

图 6.2　杨辉三角形

2. 函数 YangHui 的功能是把杨辉三角形的数据赋给二维数组的下半三角，形式如图 6.2 框内所示，其构成规律是：

(1) 第 0 列元素和主对角线元素均为 1。

(2) 其余元素为其左上方和正上方元素之和。

(3) 数据的个数每行递增 1。

请将程序补充完整。

```
#define N 6
void YangHui(int x[N][N]){
    int i,j;
    x[0][0]=1;
    for(i=0;i<N;i++){
        x[i][0]=_________________ = 1;
        for(j=1;j<i;j++)
        x[i][j]=____________________________ ;
        }
}
```

3. 本程序首先定义三个函数：

(1) string_to_list 函数，根据一个字符串创建一个链表，每一结点存储一个字符。

(2) print_list 函数，用于输出链表。

(3) concatenate_list 函数，功能为将两个链表连接在一起。

在 main 主函数中通过调用以上函数，建立两个存储“zhang”的链表 a 和存储“san”的链表 b；然后完成将 b 链表连接到 a 链表之后，最后打印输出连接之后的 a 链表，输出结果应为“zhangsan”。请根据程序功能填空。

```
#include<stdio.h>
#include<malloc.h>

struct list{
    char  data;
    structlist *next;
};
struct list *string_to_list( char s[] );
struct list * concatenate_list( struct list *a, struct list *b );
void print_list( struct list *a );
void main( ){
    struct list *head1, *head2;
    char s[]="zhang",t[]="san";
    head1 = string_to_list( s );
    head2 = string_to_list( t );
    concatenate_list( head1, head2 );
    print_list( head1 );
}

struct list *string_to_list( char s[] ){
    struct list *head=NULL, *p;  int i;
    if( s[0] ! = '\0' ){
        head=____________________________________________;
        head->data=s[0];
        p=head;
        for( i=1; s[i]! ='\0'; ++i ){
            p->next=____________________________________________;
            p=____________________________________;
            p->data= s[i] ;
        }
        p->next=____________________________________;
    }
    return(____________________________);
```

```
}

struct list * concatenate_list( struct list *a, struct list *b ){
    struct list *p=a;
    if( p == NULL )
        ____________________________________ ;
    else{
        while( p->next ! = NULL )
            ____________________________________ ;
        p->next=____________________________________;
    }
    return(a);
}
void print_list( struct list *head ){
    struct list *p = head;
    if( p == NULL )
        printf( "NULL! \n" );
    else
        do{
            printf( "%c", p->data );
            ____________________________________________;
        }while( ____________________________________ );
    printf("\n");
}
```

五、程序设计题（本程序设计题，共 30 分）

建立一个 100 人的人口普查信息表，其中包括姓名、年龄、性别、职业及地址，相应结构体类型定义如下所示：

```
struct person{
    char name[30];
    int age;
    char sex, job[30], addr[40];
};
```

要求在主函数中定义一个结构体数组，并采用形实参数结合的数据传递方式定义以下指定功能的用户自定义函数：

(1) 定义 read 函数，读入普查人员的五项数据。

(2) 定义 udf_cmpstr 字符串比较函数（注：功能必须由自己具体定义实现，不要直接或间接通过调用库函数实现）。

(3) 定义 sort 函数，按姓名以递升顺序排序。

(4) 定义 print 函数，将排序后的普查信息表输出到 person_list 文件长期保存。

(5) 定义 binarysearch 函数，以二分查找(亦称折半查找)方式实现按姓名查找该人员在人口普查信息表中的位置，若没有找到则返回－1，表示查找失败。

编写 main 函数，依次调用以上各功能函数，完成具体应用编程。其中调用 binarysearch 函数时，假设查找对象的姓名为“xiaoli”，返回主函数后即打印输出该人员在信息表中的位置，若没有找到则提示“notfound!”信息。

提示：按指定要求编程，结构体数组不要采用全局的结构体数组形式。

```
#include<stdio.h>
#include<process.h>
#define N 100
#define PERSON struct person
struct person{
    char name[30];
    int age;
    char sex, job[30], addr[40];
};
```

综合测试题四

一、选择题(本大题共 10 小题，每小题 1 分，共 10 分)

1. 计算机可以进行自动处理的基础是_____。

   A. 存储程序　　B. 快速运算　　C. 计算精度高　　D. 能进行逻辑判断

2. 计算机的通用性使其可以求解不同的算术和逻辑问题，这主要取决于计算机的_____。

   A. 高速运算　　B. 指令系统　　C. 可编程性　　D. 存储功能

3. 在计算机内部用于表示数据和指令的是_____编码。

   A. 十进制码　　B. 二进制码　　C. ASCII 码　　D. 汉字编码

4. 计算机进行数值计算时的高精度主要取决于_____。

   A. 计算速度　　B. 内存容量　　C. 外存容量　　D. 基本字长

5. 已知字母 A 的 ASCII 码为十进制数 65，且 c2 为字符型，则执行以下语句 c2='A'+'6'-'3'；后，c2 中的值为_____。

   A. D　　B. 68　　C. 不确定　　D. 'C'

6. 设已有定义：char a[5]，*p=a；则下面的选项中正确的赋值语句是_____。

   A. p="abcd";　　B. a="abcd";　　C. *p="abcd";　　D. *a="abcd";

7. 以下错误的描述是_____。

   A. break 语句不能用于循环语句和 switch 语句外的任何其他语句

   B. 在 switch 语句中使用 break 语句或 continue 语句的作用相同

   C. 在循环语句中使用 continue 语句是为了结束本次循环

   D. 在循环语句中使用 break 语句是为了使流程跳出循环体，提前结束循环

8. 设有定义:double x;,则以下正确的输入语句是_____。

A. scanf("%f",x);　　B. scanf("%f",&x,);

C. scanf("%lf",&x);　　D. scanf("%lf",x);

9. 以下对于运算符优先顺序的描述中正确的是_____。

A. 关系运算符<算术运算符<赋值运算符<逻辑与运算符

B. 逻辑与运算符<关系运算符<算术运算符<赋值运算符

C. 赋值运算符<逻辑与运算符<关系运算符<算术运算符

D. 算术运算符<关系运算符<赋值运算符<逻辑与运算符

10. C程序由函数组成,关于C语言函数的说法错误的是_____。

A. 除整型函数外,其他类型函数必须在定义时给以类型说明。

B. 函数原型说明语句必须给出函数类型的说明。

C. 函数类型决定返回值的类型。

D. 函数不仅可以递归定义,而且除main函数外都可以递归调用。

二、填空题(本大题共8小题12个填空,每空1分,共12分)

1. C的数据类型使编译器能够确定在内存中如何存储一个特定的值,以及对该数值可以_____。

2. 变量名必须以字母或是下划线开始,并由字母、数字和下划线组成,用户标识符不能与_____同名。

3. 设已有 double T,t;说明,试将等式 $T=\frac{4t^2}{t+2}-20$ 写成一条C语句。

_____

4. 表达式　1 && 30 % 10 >= 0 && 30 % 10 <= 3　的值为:_____。

5. 设已有 int x=3,y=4; 说明,表达式(x = ++y, x+5, x/5)的值为:_____。

6. 设已有 int j=5, k=2, n, m, p;说明,执行以下三条语句后n,m和p的值是什么?

n = j - ++k; m = j-- + k--;　p = k + j;

解:n的值是_____,m的值是_____,p的值是_____。

7. 设已有 int x=3, y=5, z=2;说明,执行以下三条语句后x,y和z的值是什么?

x *= y + z;　y /= 2 * z + 1;　z += x;

解:x的值是_____,y的值是_____,z的值是_____。

8. 如果n的值是4,并且m的值是5,表达式++(n * m)　的值是21吗?

解释你的答案:_____

三、阅读程序题:请仔细阅读、分析下列程序,按程序实际执行情况,写出运行结果(本大题有6小题共33分,第1、2、3题各5分,第4、5、6题各6分)

1.
```
#include "stdio.h"
int main(){
  int i=1, x=1;
  while(++i <= 20){
```

```
        if(x > 6) break;
        else if(x % 2 == 1){
          x += 5;
          printf("i=%d,x=%d\n", i, x);
          continue;
    }
    x -= i;
          printf("i=%d,x=%d\n", i, x);
      }
  }
```

1. 运行结果：

2.
```
#include <stdio.h>
#define PRODUCT(a, b) a*b
void fun(int n){
    static int x = 1;
    printf("x=%d\n", x + n);
    x += PRODUCT(x + n, x - n);
}
int main(){
    int i, x = 1;
    for (i = 1; i <= 3; i++, x++)
        fun(x + i);
}
```

2. 运行结果：

3.
```
#include <stdio.h>
void udf_sort( int  array[], int n ) {
    int  i, j, k, temp;
    for(i = 0; i < n-1; i++ ){
        k = i;
        for(j = i+1; j < n; j++ )
            if(array[j] < array[k])
                k = j;
        if(k! =i){
            temp = array[k];
            array[k] = array[i];
            array[i] = temp;
        }
        for(j = 0; j < n; j++)
            printf("%d%c", array[j], (j+1)%5 ? '□' : '\n');
    }
}
```

3. 运行结果：

```
int main(){
    int i;
    inta[5] = {2, 5, 3, 1, 4};
    udf_sort(a, 5);
    for(i = 0; i < 5; i++)
        printf("%d%c", a[i], (i+1)%5 ? '□' : '\n');
}
```

4. #include <stdio.h>

```
int multiply(int m, int n){
    int ans;
    if(n == 1)
        ans = m;
    else
        ans = m + multiply(m, n-1);
    printf("%d\n", ans);
    return(ans);
}

main(){
    printf("%d\n", multiply(6, 3));
}
```

4. 运行结果：

5. #include<stdio.h>

```
void change(int *s[],int n){
    int i, j, t;
    for(i = 0; i < 4; i++)
        for(j = i; j < 4; j++){
            t = *(s[j] + i);
            *(s[j] + i) =. *(s[i] + j);
            *(s[i] + j) = t;
        }
}

main(){
    int a[4][4] = { {1,2,3,4},{12,13,14,5},
                    {11,16,15,6}, {10,9,8,7} };
    int i, j, *p[4];
    for(i = 0; i < 4; i++)
        p[i] = a[i];
    change(p, 4);
```

5. 运行结果：

```
        for(i = 0; i < 4; i++)
            for(j = 0; j < 4; j++)
            printf("%2d%c", *(p[i] + j), j<3 ? '□' : '\n');
    }
6. #include<stdio.h>
   void silly1(int x){
        int y;
        y = x + 2;
        x = x * 2;
   }
   void silly2(int *x){
        int y;
        y = *x + 2;
        *x = 2 * *x;
   }
   main(){
        int x=10,y=11;
        silly1(x);  printf("x=%d\n", x);
        silly1(y);  printf("y=%d\n", y);
        silly2(&x); printf("x=%d\n", x);
        silly2(&y); printf("y=%d\n", y);
   }
```

6. **运行结果：**

四、程序设计题（共三个程序设计题，第 1 题 12 分，第 2 题 15 分，第 3 题 18 分，共 45 分）

1. 已知一组实验数据：3.62，2.93，3.16，3.73，2.86，3.40，2.86，3.07，3.29，3.24，编一程序分别求出它们的平均值、方差和均方差。

要求：

(1) 定义一个求解平均值、方差和均方差的函数 udf_vft()，与主调函数间的数据联系要求采用参数传递方式或 return 语句实现，不要采用全局变量传递方式。

(2) 一组实验数据可由数组方式在主函数中构成；在主函数中通过调用 udf_vft()函数得到所需要的平均值、方差和均方差并打印输出，每一结果只保留两位小数。

提示：设平均值、方差和均方差分别用变量 v，f 和 t 表示，由数学知识可知，相应的计算公式为：

$$v = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad f = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - v^2 \qquad t = \sqrt{f}$$

其中，n 表示数据个数，x_i表示第 i 个数据。

2. 编写一个程序，实现对一段文本中特定的某个单词进行统计的功能。

提示：通常文字处理软件（例如 MS Office Word 或者 WPS 等）中都提供了字符串查找匹配功能。一个程序要达到这样的功能首先需要具备有关字符串操作的一些函数来辅助实

现这个功能。本题主函数编写已经完成，源程序代码清单如下，请编写与斜体字表示的函数调用对应的有关字符串操作函数，以实现这个功能。具体要求参照相关答题处。

设主函数位于其他函数之后且内容如下：

```
#include <stdio.h>
#include <string.h>
int main(){
    int i=0,sum=0,length;
    char word[20];
    char text[]="to modernize the country's industry,agriculture,national \
        defence and science and technology step by step to turn China into \
        a strong and prosperous socialist country with a high level of \
        culture and democracy.";
    char temp[20]={0};
    printf("please input the word which you want to count.\n");
    udf_gets(word);                    //udf_gets(word)读字符串函数
    length=udf_strlen(word);           // strlen(word)求串长函数
    while(length!=0 && text[i]!='\0'){
        strncpy(temp,text+i,length);   //逐一取定长的子串
        if(udf_strcmp(word,temp)==0){  //udf_strcmp(word,temp)字符串比较函数
            sum++;                     //匹配成功统计数加1
            i+=length;                 //下标i后移，为取下一个子串作准备
        }
        i++;
    }
    printf("The word \"%s\" appears %d times in the text.\n",word,sum);
}
```

编程要求(1)：编写读字符串*udf_gets( )*函数。

*udf_gets( )*功能为：从标准输入文件 stdin 读入一串字符送到参数所指向的字符串存储空间中去，直到读到换行符 '\n' 结束，此换行符不作为字符串的内容存入串中。若无字符可读(即空串)则返回 NULL，否则返回存放字符串的起始地址。

提示：读入字符功能可调用 getchar()函数实现。本题不允许通过调用系统提供的函数 gets()实现。

函数定义：

编程要求(2)：编写*strlen( )*函数。

*udf_strlen( )*功能为：统计参数所指字符串中字符的个数。

函数定义：

编程要求(3)：编写*udf_strcmp( )*函数。

*udf_strcmp( )*功能为：对字符串 s1 和 s2 进行比较。如果 s1<s2 时，则返回负数；如果 s1=s2 时，则返回值等于 0；如果 s1>s2 时，则返回正数。

函数定义：

3. 有职工信息列表和已完成的 main()函数如下所示：

```
no      salary
1003    3864.78
1002    3726.46
......  ......
1004    2965.32
1001    2567.89
0       0
```

```
#include<stdio.h>
#include<stdlib.h>
struct person {
    int     no;                          //职工编号
    float   salary;                      //职工工资
    struct person *next;                 //指向自身的结构指针
};

main(){
    struct person *head;                 //定义头指针
    struct person * CreateListR(void);   // CreateListR 函数原型说明
    void writListR(struct person *head); // writListR 函数原型说明

    head= CreateListR();       //调用尾部插入建链表函数构建链表
    writListR(head);           //将链表的职工信息写入 person_list.txt 文件
}
```

请编写 CreateListR()和 writListR(head)函数，以完善该程序的功能实现。具体要求参照相关答题处的编程要求(1)和编程要求(2)。

编程要求(1)：编写采用表尾插法算法建立职工信息链表的函数 CreateList()。本函数编写请按算法描述和函数定义过程两步完成：

算法描述：

函数定义：

编程要求(2)：编写将链表的职工信息写入 person_list. txt 文件的函数。
函数定义：

综合测试题一参考答案

一、填空题(表达式求值)设各语句的初始化相同：int x=3,y=2,z=1；

1. 0　2. 1　3. 0　4. 0　5. 15
6. 2　7. 3　8. 1　9. 1　10. 32

二、读下列程序段，指出错误并改正(改错题)

说明：指出错误的表示形式为错误处加下划线：如int a[n][n]；
改正形式为在错误语句之后加修改内容：如 int a[4][4]；

1. ① a 数组无 '\0'，改为：char a[7]= "a1b2c3"
或：char a[7]={'a','1','b','2','c','3','\0'}
② 数组定义过小，不能存放串结束符'\0'至少应改为 a[7],b[7]。
2. ① 函数应先定义后使用，否则事先声明，因此须加函数原型说明：
double f1(double i,int j)；
② 形实参数的类型不匹配，因此须互换处理：
m=f1(a,3.14)；或 double f1(double i,int j)；
3. 字符指针数组不能直接存放字符串修改方案有 2：
① 定义 char s[5][80]；然后每行的起始地址赋给指针数组的相应元素：
for(i=0; i<5; i++)str[i]=s[i]；
for(i=0; i<5; i++) scanf("%s",str[i])；
② 或直接定义二维数组 char str[5][80]；然后：
for(i=0; i<5; i++) printf("%s\n",str[i])；
4. ① 累加器应清零：sum=0；
② 数组越界使用应改为：
for(i=0; i<10; i++)sum += r[i]；
5. 结构体类型定义时不能赋初值，应改为：

```
struct student{
    int num;
    float score1,score2,aver;
} stu[3]={{200401,92,83},{200402,85,70},{200403,65,89}};
```

三、阅读分析下列程序，写出运行结果(阅读分析题)

1. 运行结果：

```
1  1  2   3
   5  8  13
```

2. 运行结果：

```
i=2,x=3
i=4,x=8
```

3. 运行结果：

```
6     8     10
```

4. 运行结果：

```
computer->retupmoc
```

5. 运行结果：

```
1,3
3,1
```

6. 运行结果：

```
a       d
hi      no
```

7. 运行结果：

```
how are you!
```

四、程序填空题

1. 本程序功能为建立链表及统计链表中所有节点的数据域 n 之和。

(1) head= __NULL__ ;

(2) p1=p2=(struct node *)malloc(sizeof(struct node));

(3) head= __p1__ ;

(4) p2->next= __p1__ ;

(5) p1=(struct node *)malloc(sizeof(struct node));

(6) p2->next=NULL;

(7) return(__head__);

(8) add(struct node *head){

(9) sum=sum+p->n;

(10) p=p->next;

2. 已知正文有文件 abc. txt，本题统计该文件中词的个数并求出最长词的长度。一个词与另一个词之间由空格符，制表符，换行符分隔。其中，word 存放词的个数，maxLen 存放最长词的长度，length 存放当前词的长度，flag 为状态标记(flag=1 表示当前字符在单词中，flag=0 表示当前字符不在某词中)。

(1) if((fp=fopen("abc. txt", "r"))==NULL){

(2) if(length>maxLen) maxLen=length;

(3) flag= __1__ ; length=1;

(4) word __++__ ;

(5) ch=fgetc(fp) ;

五、程序设计题

1. 根据主函数的定义，编写以下两个函数：

① 求数组 a[4][4]主对角线上元素的和；

② 实现将数组 a[4][4]的矩阵转置处理。

```
void main(){
    static int a[4][4]={1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4};
    int *p, i, sum;
    sum=total(a,4); printf("%d\n",sum); //调用 total 函数求主对角线上元素的和。
    p=*a; convert(p);                   //调用 convert 实现矩阵转置
    for(i=0; i<4; i++)
        printf("%d %d %d %d\n",a[i][0],a[i][1],a[i][2],a[i][3]);
}
//参考程序清单如下：
#include <stdio.h>
void convert(int *p){
    int i, j, t;
    for(i=0; i<4; i++)
        for(j=i; j<4; j++){
            t=*(p+4*i+j);
            *(p+4*i+j)=*(p+4*j+i);
            *(p+4*j+i)=t;
        }
}
int total(int a[][4], int n){
    int i,sum=0;
    for(i=0; i<n; i++)
        if(a[i][i]! =0)
            sum+=a[i][i];
    return(sum);
}
void main(){
    static int a[4][4]={1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4};
    int *p, i, sum;
    p=*a;
    sum=total(a,4); printf("%d\n",sum);
    convert(p);
```

```
    for(i=0; i<4; i++)
        printf("%d %d %d %d\n",a[i][0],a[i][1],a[i][2],a[i][3]);
}
```

(2) 编一个程序,用结构体数组描述60位学生的信息,包括:学号、姓名、三门课的成绩和总成绩。并以函数形式实现以下功能:

① 输入60位学生的前五项信息,并计算每个学生的总成绩。

② 按总成绩递减顺序对成绩表排序。

③ 输出排序后的成绩表。

注:类型定义和主函数。

```
//参考程序清单如下:
#define N60
#define STUDENT struct student
STUDENT{
    int num;
    char name[16];
    int score[4];
};
void main(){
    STUDENT stu[N];
    void read(STUDENT *p, int n);
    void sort(STUDENT s[], int n);
    void print(STUDENT *p, int n);
    read(stu,N); sort(stu,N); print(stu,N);
}
void read(STUDENT *p, int n){
    int i,j,s;
    for(i=0; i<n; i++,p++){
        scanf("%d%s",&p->num,p->name);
        for(s=j=0; j<3; j++){
            scanf("%d",&p->score[j]);
            s=s+p->score[j];
        }
        p->score[3]=s;
    }
}
void sort(STUDENT s[], int n){
    int i,j,k; STUDENT temp;
    for(i=0; i<n-1; i++){
        k=i;
```

```
            for(j=i+1; j<n; j++){
                if(s[k].score[3]<s[j].score[3]) k=j;
        }
        if(k! =i){
            temp=s[i]; s[i]=s[k]; s[k]=temp;
        }
    }
}
void print(STUDENT *p, int n){
    int i,j;
    for(i=0; i<n; i++,p++){
        printf("%6d%16s",p->num,p->name);
        for(j=0; j<4; j++)
            printf("%6d",p->score[j]);
        printf("\n");
    }
}
```

综合测试题二参考答案

一、选择题

1. A　2. B　3. D　4. A　5. C
6. B　7. C　8. C　9. B　10. D

二、填空题

1. fabs(cos(3.14159/2))　或　a=3.14159, fabs(cos(a/2))
2. 3 * pow(x,n)/(2 * x−1)
3. 2.5
4. 1
5. 27
6. 1
7. 8
8. cc2 = (A&&E) || (A&&F) || (E&&F) ;//A,E,F3 人中至少有 2 人参与作案;
   cc3 = (A&&D) ;// A,D 不可能是同案犯;
   cc4 = (B&&C) || (! B&&! C) ;// B,C 或同时作案,或与本案无关;
   cc5 = C&&D ;// C,D 中有且仅有一人作案;
   cc6 = D || ! E ;//如果 D 没有参与作案,则 E 也不可能参与作案。

三、阅读程序题：请仔细阅读分析下列程序，写出运行结果

1. 运行结果：

```
3
3
26
58
```

2. 运行结果：

```
3  2
2  3
```

3. 运行结果：

```
1
2     4
3     6     9
```

4. 运行结果：

```
CountX=8
CountY=15
```

5. 运行结果：

```
AB
AB12
BCD
BCD234
```

四、程序填空题

1. 下面的程序把从终端读入的文本复制到一个名为 file1.dat 的新文件中。请在划线处填空。

(1) ("file1.dat","w"或"w+"　)

(2) fputc(ch,fp) 或 fprintf(fp,"%c",ch);

2. 以下程序，数组 a 中存放一个递增数列。输入一个整数 x 并将它插入到数组 a 中，使之仍为一个递增数列。请在划线处填空。

(1) 11 或比 11 大的数

(2) break

(3) a[i+1]=x 或　a[p]=x

五、程序设计题

编程实现以链表形式处理学生选修信息。每一学生选修课程的记录信息包括：学号 sno、课程号 cno，成绩 score，相应数据数据项如："PB07210001","C0020101",95.0。请按要求编写各指定功能函数。(说明：本题已给出包含必要的头文件的预处理命令、结构体类型定义和放在程序末尾的主函数，供编程参照使用。)

(1) 编写一个计算字符串长度的 str_len( )函数，要求为：函数的返回值为字符串长度。

```
//参考程序清单如下：
int str_len(char *p){   //求字符串长度
    int i=0;
    while(*p++! ='\0')i++;
    return(i);
}
```

或数组形式编写。

(2) 编写一个字符串比较 str_cmp( )函数，可采用数组或指针方法编写；设有字符串 s 与 t，若 s 大于 t 则返回正值，s 等于 t 则返回 0，s 小于 t 则返回负值。

```
//参考程序清单如下：
intstr_cmp(char s[],char t[]){//数组形式
    int i=0;
    while(s[i]==t[i])
        if(s[i++]=='\0') return(0);
    return(s[i]-t[i]);
}
```

或

```
int str_cmp(char *s,char *t){//指针形式
    while(*s==*t&&*s! ='\0'){s++;t++;}
    return(*s-*t);
}
```

(3) 编写一个建立链表的 create( )函数，要求为：建立结点个数不限，当输入记录的学号字符串长度小于等于 1 时结束，如输入：0　0　0 时结束；函数的返回值为指向链表 head 指针。

```
//参考程序清单如下：
struct student *create( ){          //建立链表函数，表尾插入算法
    struct student *head,*p1,*p2;
    head=NULL;
    p1=p2=(struct student *)malloc(sizeof(struct student));
    scanf("%s%s%f",&p1->sno,&p1->cno,&p1->score);
    while(str_len(p1->sno)>1){
        if(head==NULL)head=p1;
        else p2->next=p1;
        p2=p1;
        p1=(struct student *)malloc(sizeoF(STRUCT STUDENT));
        scanf("%s%s%f",&p1->sno,&p1->cno,&p1->score);
    }
    p2->next=NULL; free(p1);
    return(head);
}
```

或采用表头插入方式、有序插入方式等编写。

(4) 编写一个按学号检索学生选修课程信息的 serch_print( )函数，要求为：能按指定学号检索该学生所有的选修课程信息，即打印出所有与指定学号相关的结点的成员信息。

```
//参考程序清单如下：
void serch_print(struct student *head,char *search_sno){//查找打印信息
    STRUCT STUDENT *p; p=head;
    if(head ! = NULL)
        do{ if(str_cmp(p->sno,search_sno)==0)
                printf("%16s %12s %f\n",p->sno,p->cno,p->score);
            p=p->next;
        }while(p ! = NULL);
}
```

2. 如图 6.3 所示，已知某游泳池的长度为 50 米，宽度为 25 米。某人游泳速度为 0.8 米/秒，步行速度为 1.2 米/秒。编程求解从 A 点到 B 点间何处下水游到 C 点时间最短，输出该点距 A 点的长度 x 和所求最短时间 y。要求：

(1) 给出求解该问题的算法描述。

(2) 根据算法编写完整的求解程序。

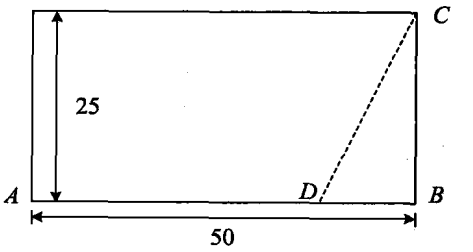


图 6.3　游泳池入水点问题示意图

说明：距离的长度精度控制在 0.1 米或以下。

(1) 求解问题的算法描述：

//参考程序清单如下：

本题采用枚举法(亦称穷举法)求解。

分析：

① 假设在 AB 边上行进 x1 后的 D1 点下水，历时 t1=x1/1.2；

② 计算从 D1 点到 C 点的距离和 D1 点到 C 点历时

t2= sqrt(WIDTH * WIDTH+(LENGTH−x1) * (LENGTH−x1))/0.8

③ 则从 A 点到 B 点间的 D 点下水游到 C 点时间的为 $t=t1+t2$，显然，从 A 点到 B 点间的 $D1$ 点下水游到 C 点最短计算如存在的话，肯定在 0 到 50 之间，而这之间 $D1$ 点在一定精度的步长内个数是有限的，所以可以设计一个基于枚举法的算法，设用 time 存储其最小值，dx 存储距 A 点的长度，其最短时间算法描述如下：

```
预设 x=0.0,time =32768,dx=0
while(x<=50.0){
    temp=x/1.2
        +sqrt(WIDTH * WIDTH+(LENGTH−x) * (LENGTH−x))/0.8;
    if(若所求时间小于等于前次所求最短时间){
        更新保存最短时间和相应距离两变量的值;
    }
    x=x+0.1;//距离增步进长 0.1,这里步进长精度为 0.1
```

```
}
```

(2) 根据算法编写完整的求解程序如下：

```
//参考程序清单如下：
#include<stdio.h>
#include<math.h>
#define LENGTH   50.0
#define WIDTH 25.0
#define V1   0.8
#define V2   1.2

void main(){                          //枚举法求解 1
    float x=0.0,time,temp,ax=0;
    time =32768;
    while(x<=50.0){
        temp=x/V2+sqrt(WIDTH*WIDTH+(LENGTH-x)*(LENGTH-x))/V1;
        if(temp<=time){
            time=temp;
            ax=x;
        }
        x=x+0.1;
    }
    printf("time min=%f, x=%f\n",time,ax);
    printf("(50+25)/1.2 =%f\n",(LENGTH+WIDTH)/V2);
}
```

注：此题解法甚多。

综合测试题三参考答案

一、选择题

1. C　2. A　3. C　4. B　5. A
6. C　7. D　8. B　9. C　10. B

二、填空题

1. 全局 、局部
2. 5
3. 1,0
4. 0
5. 'a' 或 97
6. 0

7. 20
8. 6

三、阅读程序题(请仔细阅读分析下列程序,按程序实际执行情况写出运行结果)

1. 运行结果:

```
4
7
10
4
```

2. 运行结果:

```
17154
```

3. 运行结果:

```
 1   1
 9   1
 17
```

4. 运行结果:

```
0  2  4  4  8
```

5. 运行结果:

```
image
mage
age
ami
```

6. 运行结果:

```
1, 3
3, 1
1, 3
```

7. 运行结果:

```
10      14
28      32
84
```

四、程序填空题

填空第 1 题参考答案:

(1) double a[5], int n
　　double a[5], int n
　　double *a, int n

(2) t * x

(3) y

填空第 2 题参考答案:

(4) x[i][i]

(5) x[i-1][j-1]+x[i-1][j]

填空第3题参考答案：

(6) (struct list *)malloc(sizeof(struct list))

(7) (struct list *)malloc(sizeof(struct list))

(8) p->next

(9) NULL

(10) head

(11) a=b

(12) p=p->next

(13) b

(14) p=p->next

(15) p != NULL

五、程序设计题

1. 建立一个100人的人口普查信息表，其中包括姓名、年龄、性别、职业、及地址，相应结构体类型定义如下所示：

```
struct person{
    char name[30];
    int age;
    char sex, job[30], addr[40];
};
```

要求在主函数中定义一个结构体数组，并采用形实参数结合的数据传递方式定义以下指定功能的用户自定义函数：

(1) 定义read函数，读入普查人员的五项数据。

(2) 定义udf_cmpstr字符串比较函数(注：功能必须由自己具体定义实现，不要直接或间接通过调用库函数实现)。

(3) 定义sort函数，按姓名以递升顺序排序。

(4) 定义print函数，将排序后的普查信息表输出到person_list文件长期保存。

(5) 定义binarysearch函数，以二分查找(亦称折半查找)方式实现按姓名查找该人员的在人口普查信息表的位置，若没有找到则返回-1，表示查找失败。

编写main函数，依次调用以上各功能函数，完成具体应用编程。其中调用binarysearch函数时，假设查找对象的姓名为"xiaoli"，返回主调函数后即打印输出该人员在信息表中的位置，若没有找到则提示"notfound!"信息。

提示：按指定要求编程，结构体数组不要采用全局的结构体数组形式。

```
#include<stdio.h>
#include<process.h>
#define N 100
#define PERSON struct person
```

```
struct person{
    char name[30];
    int age;
    char sex, job[30], addr[40];
};
```

程序设计参考答案：

(1) 定义 read 函数，读入普查人员的五项数据；

```
void read(PERSON s[], int n){
    int  i;
    for(i=0; i<n; i++)
        scanf("%s %d %c %s
    %s",s[i].name,&s[i].age,&s[i].sex,s[i].job,s[i].addr);
}
```

(2) 定义 udf_cmpstr 字符串比较函数(注：功能必须由自己具体定义实现，不要直接或间接通过调用库函数实现)；

```
int udf_cmpstr(char s[], char t[]){
    int i=0;
    while(s[i]==t[i] && s[i]! ='\0')
        ++i;
    return(s[i]-t[i]);
}
```

(3) 定义 sort 函数，按姓名以递升顺序排序；

```
void sort(PERSON s[], int n){
    int  i, j, k; PERSON  temp;
    for(i=0; i<n-1; i++){
    k=i;
    for(j=i+1; j<n; j++)
        if(udf_cmpstr(s[k].name,s[j].name)>0) k=j;
    if(k! =i){
            temp=s[i]; s[i]=s[k]; s[k]=temp;
    }
    }
}
```

(4) 定义 print 函数,将排序后的普查信息表输出到 person_list 文件长期保存

```
void print(PERSON *p, int n){
    FILE *fp; int i;
    if((fp=fopen( "person_list", "w" ))==NULL){
    printf("cannot open this file.\n"); exit(0);
    }
    for(i=0; i<n; i++,p++){          //也可采用 fwrite 等形式完成
    fprintf(fp,"%s %d %c %s %s\n",
                p->name,p->age,p->sex,p->job,p->addr);
    }
    fclose(fp);
}
```

(5) 定义 binarysearch 函数,以二分查找(亦称折半查找)方式实现按姓名查找该人员在人口普查信息表的位置,若没有找到则返回-1,表示查找失败。

```
int  binarysearch(PERSON s[],int n,char *key){
    int  low,  mid,  high;
    low=0;  high=n-1;
    while(low<=high){
        mid=(low+high)/2;
        if(udf_cmpstr(key,s[mid].name)>0)
            low=mid+1;
        else if(udf_cmpstr(key,s[mid].name)==0)
            return(mid);
        else
            high=mid-1;
    }
    return -1;
}
```

编写 main 函数,依次调用以上各功能函数,完成具体应用编程。其中调用 Binarysearch 函数时,假设查找对象的姓名为"xiaoli",返回主函数后即打印输出该人员在信息表中的位置,若没有找到则提示"not found!"信息。

```
void main(){
    PERSON pe[N];  int i;
    read(pe,N);
```

```
    sort(pe,N);
    print(pe,N);
    i=binarysearch(pe,N,"xiaoli");
    if(i! =-1)
        printf("%d %s\n",i,pe[i].name);
    else
        printf("not found!");
}
```

综合测试题四参考答案

一、选择题(本大题共 10 小题,每小题 1 分,共 10 分)

1. A　2. C　3. B　4. D　5. B
6. A　7. B　8. C　9. C　10. D

二、填空题(本大题共 8 小题 12 个填空,每空 1 分,共 12 分)

1. 执行什么样的操作　或　执行的运算
2. 保留字
3. T = 4 * t * t / (t + 2) - 20　或　T = 4 * pow(t, 2) / (t + 2) - 20
4. 1
5. 1
6. n 的值是　2，　m 的值是　8，　p 的值是　6
7. x 的值是　21，y 的值是　1，　z 的值是　23
8. 如果 n 的值是 4 并且 m 的值是 5,表达式　++(n * m)　的值是 21 吗?
   解释你的答案:不是,表达式非法。递增运算不能用于(n * m)这样的表达式 。

三、阅读程序题:请仔细阅读分析下列程序,按程序实际执行情况,写出运行结果(本大题共 6 小题,第 1 题 5 分,其余各题 6 分,共 33 分)

1. 运行结果:
   i=2,x=6
   i=3,x=3
   i=4,x=8
2. 运行结果:
   x=3
   x=6
   x=14
3. 运行结果:
   1 5 3 2 4
   1 2 3 5 4

1 2 3 5 4
1 2 3 4 5
1 2 3 4 5

4. 运行结果：

6
12
18
18

5. 运行结果：

1 12 11 10
2 13 16 9
3 14 15 8
4 5 6 7

6. 运行结果：

x=10
y=11
x=20
y=22

四、程序设计题(共三个程序设计题，第1题12分，第2题15分，第3题18分，共45分)

1. 已知一组实验数据：3.62，2.93，3.16，3.73，2.86，3.40，2.86，3.07，3.29，3.24，编一程序分别求出它们的平均值、方差和均方差。

要求：

(1) 定义一个求解平均值、方差和均方差的函数 udf_vft()，与主调函数间的数据联系要求采用参数传递方式或 return 语句实现，不要采用全局变量传递方式。

(2) 一组实验数据可由数组方式在主函数中构成，并在主函数中通过调用 udf_vft()函数得到所需要的平均值、方差和均方差并打印输出，每一结果只保留两位小数。

提示：

设平均值、方差和均方差分别用变量 v，f 和 t 表示，由数学知识可知，相应的计算公式为：

$$v=\frac{1}{n}\sum_{i=1}^{n}x_i \qquad f=\frac{1}{n}\sum_{i=1}^{n}x_i^2-v^2 \qquad t=\sqrt{f}$$

其中，n 表示数据个数，x_i表示第 i 个数据。

解：
```
#include<stdio.h>
#include<math.h>
#define N 10                                        // N 等于待处理数据的个数
int main(){
    double a[10]={3.62,2.93,3.16,3.73,2.86,3.40,2.86,3.07,3.29,3.24};
    double v,f,t,s1=0,s2=0;
```

```
    void udf_vft(double a[],int n,double *v,double *f,double *t);
    udf_vft(a, N, &v, &f, &t);
    printf("平均值 = %5.2f\n",v);
    printf("方　差 = %5.2f\n",f);
    printf("均方差 = %5.2f\n",t);
}

void udf_vft(double a[],int n,double *v,double *f,double *t){
    double s1=0,s2=0;
    int i;
    for(i=0;i<N;i++){
        s1 += a[i];
        s2 += a[i]*a[i];
    }
    *v=s1/N;
    *f=s2/N;
    *t=sqrt(*f);
}
```

注:本题还可采用主函数定义数组、动态申请存储空间等方法由参数传递返回平均值、方差和均方差。但要注意本题不能使用局部变量或数组返回其值。

2. 编写一个程序,实现对一段文本中特定的某个单词进行统计的功能。

提示:通常文字处理软件(例如 MS Office Word 或者 WPS 等)中都提供了字符串查找匹配功能。一个程序要达到这样的功能首先需要具备有关字符串操作的一些函数来辅助实现这个功能。本题主函数编写已经完成,源程序代码清单如下,请编写与斜体字表示的函数调用对应的有关字符串操作函数,以实现这个功能。具体要求参照相关答题处。设主函数位于其他函数之后且内容如下:

```
#include <stdio.h>
#include <string.h>
int main(){
    int i=0,sum=0,length;
    char word[20];
    char text[]="to modernize the country's industry,agriculture,national \
        defence and science and technology step by step to turn China into \
        a strong and prosperous socialist country with a high level of \
        culture and democracy.";
    char temp[20]={0};
    printf("please input the word which you want to count.\n");
    udf_gets(word,20);//udf_gets(word)读字符串函数
    length=udf_strlen(word);// strlen(word)求串长函数
```

```
    while(length! =0 && text[i]! ='\0'){
        strncpy(temp,text+i,length);      //逐一取定长的子串
        if(udf_strcmp(word,temp)==0){     //udf_strcmp(word,temp)字符串比较函数
            sum++;                        //匹配成功统计数加 1
            i +=length;                   //下标 i 后移,为取下一个子串作准备
        }
        i++;
    }
    printf("The word \"%s\" appears %d times in the text.\n",word,sum);
}
```

编程要求(1):编写*udf_gets( )*函数。

*udf_gets( )*功能为:从标准输入文件 stdin 读入一串字符送到参数所指向的字符串存储空间中去,直到读到换行符'\n'结束,此换行符不作为字符串的内容存入串中。若无字符可读(即空串)则返回 NULL,否则返回存放字符串的起始地址。

提示:读入字符功能可调用 getchar()函数实现。本题不允许通过调用系统提供的函数 gets()实现。

函数定义:

```
char    * udf_gets (char s[], int lim){
  int c, i;
  for(i=0;i<lim-1&&(c=getchar())! =EOF&&c! ='\n';++i)
    s[i]=c;
  s[i]='\0';                    //添加串结束符
  if(i! =0)
    return(s);
  else
    return(NULL);               //返回串的长度 i
}
```

编程要求(2):编写*strlen( )*函数。

*udf_strlen( )*功能为:统计参数所指字符串中字符的个数。

函数定义:

```
int   udf_strlen(char s[]){
  int i=0;
  while(s[i]! ='\0') i++;
  return(i);
}
```

编程要求(3):编写*udf_strcmp( )*函数。

*udf_strcmp( )*功能为:对字符串 s1 和 s2 进行比较。如果 s1<s2 时,则返回负数;如果

s1=s2 时，则返回值等于 0；如果 s1>s2 时，则返回正数。

函数定义：

```
int udf_strcmp( char s[ ], char t[ ] ){
  int i=0;
  while( s[i]==t[i] )
  if( s[i++]=='\0')
    return( 0 );
  return( s[i] - t[i] );
}
```

注：本函数的功能实现方法略有可能不同，要求以功能实现正确、简洁为判分依据。

3. 有职工信息列表和已完成的 main()函数如下所示：

no	salary.
1003	3864.78
1002	3726.46
……	……
1004	2965.32
1001	2567.89
0	0

```
#include<stdio.h>
#include<stdlib.h>
struct person {
    int     no;                          //职工编号
    float   salary;                      //职工工资
    struct person *next;                 //指向自身的结构指针
};

main(){
    struct person *head;                 //定义头指针
    struct person * CreateListR(void);   // CreateListR 函数原型说明
    void writListR(struct person *head); // writListR 函数原型说明
    head= CreateListR();        //调用尾部插入建链表函数构建链表
    writListR(head);            //将链表的职工信息写入 person_list.txt 文件
}
```

请编写 CreateListR()和 writListR(head)函数，以完善该程序的功能实现。具体要求参照相关答题处的编程要求(1)和编程要求(2)。

编程要求(1)：编写采用表尾插法算法建立职工信息链表的函数 CreateList()。本函数编写请按算法描述和函数定义过程两步完成：

算法描述：

```
head = NULL;      //置空链表
```

```
输入一个循环控制数据;
while(数据值不是结束标志) {
        申请新结点,用指针 p 指向该结点;
        给新结点赋予相关数据值;
        if(链表为空) {
            表示新插入的结点是首结点,
            头指针指向首结点 head = p;
        }
        else {
            将新结点插入到尾指针指向的结点之后,rear->next=p;
            使尾指针指向新结点,rear = p;
        }
        输入一个循环控制数据;
}
```

函数定义:

```
struct person * CreateListR(void) {
    struct person *head;          //头指针
    struct person *rear;          //尾指针
    struct person * p ;           //新结点指针
    head=NULL;
    p=(struct person *)malloc(sizeof(struct person));      //申请新结点
    printf("\n 输入一个职工的工号和工资");
    scanf("%d%f",&p->no,&p->salary);
    while(p->no! =0){             //读入职工号不是结束标志(0)时做循环
        if(head==NULL)
            head=p;               //将 p 指向的新结点插入空表
        else
            rear->next=p;         //新结点插入到表尾结点(rear 指向的结点)之后
        rear=p;                   //表尾指针指向新的表尾结点
        p=(struct person *)malloc(sizeof(struct person));      //申请新结点
        printf("\n 输入一个职工的工号和工资");
        scanf("%d%f",&p->no,&p->salary);
    }
    rear->next=NULL;              //终端结点置空
    free(p);
    return head;                  //返回表头指针
}
```

编程要求(2):编写将链表的职工信息写入 person_list.txt 文件的函数。

函数定义：

```
void writListR(struct person *head){
    FILE *fp;
    struct person *p ;
    if((fp=fopen("person_list.txt","w"))==NULL){
        printf("cannot open this file .\n");
        exit(0);
    }
  p=head;
  while(p!=NULL){
    fprintf(fp,"%d %f\n", p->no,p->salary);
    p=p->next;
  }
}
```

运行结果

输入：

```
1003  3864.78
1002  3726.46
1005  3567.89
1004  2965.32
1001  2567.89
0 0
```

person_list.txt 文件内容：

```
1003  3864.78
1002  3726.46
1005  3567.89
1004  2965.32
1001  2567.89
```

附录A 常用库函数

下表分类列出C语言提供的常用库函数及其简要功能。有关这些函数的详细使用和另外的函数等内容，请参阅有关资料。

1. 数学函数

使用数学函数时，应该在源程序中加入命令行：

#include〈math. h〉或#include "math. h"

数学类库函数及其简要功能如表A.1所示。

表A.1 数学类常用库函数

函数名称	函数原型	函数功能
abs	int abs(int x);	求整数x的绝对值
fabs	double fabs(double x);	求浮点数x的绝对值
exp	double exp(double x);	求指数 e^x 的值
log	double log(double x);	求对数 $\log_e x$，即 lnx 的值
log10	double log10(double x);	求对数 $\log_{10} x$ 的值
pow	double pow(double x,double y);	计算方幂 x^y 的值
sqrt	double sqrt(double x);	计算平方根$\sqrt{x}$的值
sin	double sin(double x);	计算x的正弦值
cos	double cos(double x);	计算x的余弦值
tan	couble tan(double x);	计算x的正切值
asin	double asin(double x);	计算x的反正弦值
acos	double acos(double x);	计算x的反余弦值
atan	double atan(double x);	计算x的反正切值
rand	int tand(void); //要求使用#include〈stdlib. h〉	产生－90到32767间的随机整数
sinh	double sinh(double x);	计算x的双曲正弦函数sinh(x)的值
cosh	double cosh(double x);	计算x的双曲余弦函数cosh(x)的值
tanh	double tanh(double x);	计算x的双曲正切函数tanh(x)的值
floor	double floor(double x);	求出不大于x的最大整数，值为双精度浮点数
ceil	double(double x);	求出不小于x的最大整数，值为双精度浮点数

2. 字符函数和字符串函数

ANSI C标准要求在使用字符串函数时要包含头文件“string. h”，在使用字符函数时要包含头文件“ctype. h”。有些版本的C可能不遵循ANSI C标准的规定，而用其他名称的头文件。请使用时查阅有关手册。常用字符和字符串类库函数如表A. 2所示。

表A. 2 常用字符和字符串类库函数

函数名称	函数原型	函数功能	返回值
isalnum	int isalnum(int ch);	检查ch是否是字母(alpha)或数字(number)	是返回1,否则返回0
isalpha	int isalpha(int ch);	检查ch是否是字母	是返回1,不是返回0
iscntrl	int iscntrl(int ch);	检查ch是否是控制字符	是返回1,不是返回0
isdigit	int isdigit(int ch);	检查ch是否是数字字符	是返回1,不是返回0
isgraph	int isgraph(int ch);	检查ch是否是可打印字符(ox21～0x7E之间)	是返回1,不是返回0
islower	int islower(int ch);	检查ch是否是小写字母	是返回1,不是返回0
isupper	int isupper(int ch);	检查ch是否是大写字母	是返回1,不是返回0
isprint	int isprint(int ch);	检查ch是否是可打印字符(ox20～0x7E之间)	是返回1,不是返回0
ispunct	int ispunct(int ch);	检查ch是否是标点字符,不含空格	是返回1,不是返回0
isspace	int isspace(int ch);	检查ch是否是空格、制表符或换行符	是返回1,不是返回0
isxdigit	int isxdigit(int ch);	检查ch是否是一个十六进制数字字符	是返回1,不是返回0
strcat	char * strcat(char * * str1,char * str2);	把字符串str2接到str1后面,str1最后面的'\0'被取消	返回连接后的字符串str1
strchr	char * strchr(char * str,int ch);	找出指向的字符串中第一次出现字符ch的位置	返回指向该位置的指针,如找不到,则返回空指针
strcmp	int strcmp(char * str1,char * str2);	比较两个字符串str1与str2大小	str1<str2,返回负数 str1=str2,返回0 str1>str2,返回正数
strcpy	char * strcpy(char * str1,char * str2);	把str2指向的字符串复制到str1中去	返回str1
strlen	unsigned int strlen(char * str);	统计字符串str中字符的个数(部包括字符串结束符'\0')	返回字符串长度
strstr	char * strstr(char * str1,char * str2);	找出str2字符串在str1字符串中第一次出现的位置	返回该位置的指针,如找不到,则返回空指针
tolow	int tolow(int ch);	将ch字符转换为小写字母	返回与ch相应的小写字母
toupper	int toupper(int ch);	将ch字符转换为小写字母	返回与ch相应的大写字母

3. 输入输出函数

常用输入输出库函数如表 A.3 所示。凡用以下的输入输出函数，都应该使用预处理命令“#include(stdio.h)”把 stdio.h 头文件包含到源程序文件中。

表 A.3 常用输入输出库函数

函数名称	函数原型	函数功能	返回值
fclose	intfclose(FILE * fp);	关闭 fp 所致的文件，释放文件缓冲区	有错返回非零，否则返回 0
feof	int feof(FILE * fp);	检查文件是否结束	遇文件结束符返回非零，否则返回 0
fgetc	int fgetc(FILE * fp);	从 fp 所指定的文件中取得下一个字符	返回所得字符，若读入出错，返回 EOF
fgets	char * fgets(char * buf, int n,FILE * fp);	从 fp 所指定的文件读取一个长度为(n－1)的字符串，存入起始地址为 buf 的空间	返回地址 buf，若遇文件结束符或出错，返回 NULL
fopen	FILE fopen(char * filename, char mode);	以 mode 指定的方式打开名为 filename 的文件	成功，返回一个文件指针；否则返回 0
fprintf	int fprintf(FILE * fp, char * format, args,…);	把 args 的值以 format 指定的格式输出到 fp 所指定的文件中	实际输出的字符数
fputc	int fputc(char ch, FILE * fp);	将字符 ch 输出到 fp 指向的文件中	成功，返回该字符；否则返回非 0
fputs	intfputs(char * str, FILE * fp);	将 str 指向的字符串输出到 fp 指向的文件中	成功返回 0；若出错返回非 0
fread	int fread(char * pt, unsigned size,unsigned n, FILE * fp);	从 fp 所指定的文件读取一个长度为 size 的 n 个数据项，存到 pt 所指向的内存区	返回所读的数据项个数，如遇文件结束或出错返回 0
fscanf	int fscanf(FILE * fp,char format, args,…);	从 fp 所指定的文件中按 format 给定的格式将输入数据送到 args 所指向的内存单元(ags 是指针)	已输入数据个数
fseek	int fseek(FILE * fp,long offset, int base);	将 fp 所指向的文件的位置指针移到以 base 所给出的位置为基准、以 offset 为位移量的位置	返回当前位置；否则返回－1
ftell	long ftell(FILE * fp);	返回 fp 所指向的文件中的读写位置	返回 fp 所指向的文件中的读写位置
fwrite	int fwrire(char * ptr, unsignedsize, unsigned n, FILE * fp);	把 ptr 所指向的 n * size 个字节输出到 fp 所指向的文件中	写到 fp 文件中的数据项的个数
getc	int getc(FILE * fp);	从 fp 所指向的文件中读入一个字符	返回所读的字符，若文件结束或出错，返回－1
getchar	int getchar(void);	从标准输入设备读取下一个字符	所读字符，若文件结束或出错，则返回－1
Printf	int printf(char * format, args, …);	按 format 指向的格式字符串所规定的格式，将输出表列 args 的值输出到标准输出设备	输出字符的个数，若出错，返回负数

(续)表 A.3

函数名称	函数原型	函数功能	返回值
putc	int putc(int ch, FILE * fp);	把一个字符 ch 输出到 fp 所指的文件中	输出的字符 ch,若出错,返回 EOF
putchar	int putchar(char ch);	把字符 ch 输出到标准输出设备	输出的字符 ch,若出错,返回 EOF
puts	int puts(char * str);	把 str 指向的字符串输出到标准输出设备,将"\0"转换位回车换行	返回换行符,若失败,返回 EOF
fetw	int getw(file * fp);	从 fp 所指向的文件读取下一个字(整数)	输入的整数,若文件结束或出错,返回-1
putw	int putw(int w, FILE * fp);	将一个整数 w(即一个字)写到 fp 指向的文件中	返回输出的整数,若出错,返回 EOF
rewind	void rewind(FILE * fp);	将 fp 指示的文件中的位置指针置于文件开头的位置,并清除文件结束标志和错误标志	无
scanf	int scang(char * format, args, …); //要求 args 为指针	从标准输入设备按 fomat 指向的格式字符串所规定的格式,输入数据给 args 所指向的内存单元	读入并赋给 args 的数据个数,遇文件结束返回 EOF,出错返回 0

4. 动态存储分配函数

ANSI 标准建议在"stdlib. h"头文件中包含以下态存储分配函数的信息,但许多 C 编译系统要求用"malloc. h"而不是"stdlib. h"。读者在使用时可向系统员查询有关情况,或查阅有关手册。凡用以下的动态存储分配函数(见表 A. 4),都应该使用预处理命令 #include〈stdlib. h〉或 #include〈malloc. h〉,把适应系统要求的头文件包含到源程序文件中。

ANSI 标准要求动态分配系统返回 void 指针。void 指针具有一般性,它们可以指向任何类型的数据。但目前有的 C 编译所提供的这类函数返回 char 指针。无论是以上两种情况的哪一种,都需要用强制类型转换的方法把 void 或 char 指针转换成所需的类型。

表 A.4 常用动态存储分配函数

函数名称	函数原型	函数功能	返回值
calloc	void * calloc(unsigned n, unsigned size);	分配 n 个数据项的内存连续空间,每个数据项的大小为 size	所分配内存单元的起始地址,如不成功,返回 NULL
malloc	void * malloc(unsigned size);	分配大小为 size 字节的内存区	所分配的内存区起始地址,如内存不够,返回 NULL
realloc	void * realloc(void * p, unsigned size);	将 p 所指向的已分配内存区的大小改为 size,size 可以比原来分配的空间大或小	返回指向该内存区的指针
free	void free(void * p);	释放 p 所指的内存区	无

附录B ASCII码表

计算机中的字符按一定的规则用二进制编码表示，一般用一个字节即8个二进制位进行编码。目前最普遍采用的是由美国国家标准协会(American National Standards Institute, ANSI)所制订的美国标准信息交换码(American Standard Code for Information Interchange, ASCII)来进行编码。该编码规定：8个二进制位的最高位为0，其余7位可进行编码，因此可表示128个字符，其中95个编码对应计算机终端能输入并可显示的95个字符，其余33个编码对应不可显示的控制字符。

ASCII码表(见表B.1)中有33种控制码，十进制码值分别为0～31和127(即NUL～US和DEL)称为非图形字符(又称控制字符)，主要用于打印或显示时的格式控制，对外部设备的操作控制、进行信息分隔、在数据通信时进行传输控制等用途。其中常用的控制字符及其含义见表B.1 ASCII码表。其余95个字符称为图形字符(又称为普通字符)，为可打印字符，包括英文大小写字母共52个，0～9的数字共10个和其他标点符号、运算符等共33个。其中0～9、A～Z、a～z都是顺序排列的，且小写比大写字母码值大32，码值32对应的是表中第一个可显示字符(空格)，数字0的码值为48，大写字母A的码值为65，小写字母a的码值为97。

表B.1 ASCII码表

十进制	十六进制	八进制	控制字符/字符	
0	0	0	NUL	(Null)
1	1	1	SOH	(Start of heading)
2	2	2	STX	(Start of text)
3	3	3	ETX	(End of text)
4	4	4	EOT	(End of transmission)
5	5	5	ENQ	(Enquiry)
6	6	6	ACK	(Acknowlege)
7	7	7	BEL	(Ring bell)
8	8	10	BS	(Backspace 退格)
9	9	11	HT	(Horizontal tab 水平制表)
10	a	12	LF	(Line feed 换行)
11	b	13	VT	(Vertical tab 垂直制表)
12	c	14	FF	(Form feed 换页)

(续)表 B.1

十进制	十六进制	八进制	控制字符/字符	
13	d	15	CR	(Carriage return 回车)
14	e	16	SO	(Shift out)
15	f	17	SI	(Shift in)
16	10	20	DLE	(Data link escape)
17	11	21	DC1	(Device control 1)
18	12	22	DC2	(Device control 2)
19	13	23	DC3	(Device control 3)
20	14	24	DC4	(Device control 4)
21	15	25	NAK	(Negative acknowledge)
22	16	26	SYN	(Synchronous idle)
23	17	27	ETB	(End of transmission block)
24	18	30	CAN	(Cancel 作废)
25	19	31	EM	(End of medium)
26	1a	32	SUB	(Substitute)
27	1b	33	ESC	(Escape 换码)
28	1c	34	FS	(File separator)
29	1d	35	GS	(Group separator)
30	1e	36	RS	(Record separator)
31	1f	37	US	(Unit separator)
32	20	40	SP	(Space 空格)
33	21	41	!	
34	22	42	"	
35	23	43	#	
36	24	44	$	
37	25	45	%	
38	26	46	&	
39	27	47	'	(Single quote)
40	28	50	(	
41	29	51	)	
42	2a	52	*	
43	2b	53	+	
44	2c	54	,	(Comma)

(续)表 B.1

十进制	十六进制	八进制	控制字符/字符
45	2d	55	— (Hyphen)
46	2e	56	.
47	2f	57	/
48	30	60	0
49	31	61	1
50	32	62	2
51	33	63	3
52	34	64	4
53	35	65	5
54	36	66	6
55	37	67	7
56	38	70	8
57	39	71	9
58	3a	72	:
59	3b	73	;
60	3c	74	<
61	3d	75	=
62	3e	76	>
63	3f	77	?
64	40	100	@
65	41	101	A
66	42	102	B
67	43	103	C
68	44	104	D
69	45	105	E
70	46	106	F
71	47	107	G
72	48	110	H
73	49	111	I
74	4a	112	J
75	4b	113	K
76	4c	114	L

(续)表 B.1

十进制	十六进制	八进制	控制字符/字符
77	4d	115	M
78	4e	116	N
79	4f	117	O
80	50	120	P
81	51	121	Q
82	52	122	R
83	53	123	S
84	54	124	T
85	55	125	U
86	56	126	V
87	57	127	W
88	58	130	X
89	59	131	Y
90	5a	132	Z
91	5b	133	[
92	5c	134	\
93	5d	135	]
94	5e	136	^
95	5f	137	_ (Underscore)
96	60	140	` (Grave accent)
97	61	141	a
98	62	142	b
99	63	143	c
100	64	144	d
101	65	145	e
102	66	146	f
103	67	147	g
104	68	150	h
105	69	151	i
106	6a	152	j
107	6b	153	k
108	6c	154	l

(续)表 B.1

十进制	十六进制	八进制	控制字符/字符
109	6d	155	m
110	6e	156	n
111	6f	157	o
112	70	160	p
113	71	161	q
114	72	162	r
115	73	163	s
116	74	164	t
117	75	165	u
118	76	166	v
119	77	167	w
120	78	170	x
121	79	171	y
122	7a	172	z
123	7b	173	{
124	7c	174	|
125	7d	175	}
126	7e	176	~
127	7f	177	DEL (Delete 删除)

参 考 文 献

[1] 贾伯琪，顾为兵，苏仕华，张四海，何克东. 计算机程序设计：C 语言版[M]. 北京：机械工业出版社，2011.

[2] 谭浩强. C 程序设计[M]. 3 版. 北京：清华大学出版社，2005.

[3] Brian W Kernihan, Dennis M Ritchie. The C programming language[M]. 2nd ed. 北京：清华大学出版社，1997.

[4] 谭浩强. C 程序设计题解与上机指导[M]. 北京：清华大学出版社，1992.

[5] 吴文虎. 程序设计基础[M]. 2 版 . 北京：清华大学出版社，2004.

[6] 王金汉，宋秋杰，秦佩玲. C 程序设计语言[M]. 上海：同济大学出版社，1993.

[7] 刘振安，苏仕华. C 程序设计实用教程[M]. 西安：西安电子科技大学出版社，1994.

[8] 徐士良. 计算机常用算法[M]. 北京：清华大学出版社，1989.

[9] 谭浩强，田淑清. FORTRAN77 结构化程序设计[M]. 北京：高等教育出版社，1986.