前言

首先，在各位对本集的支持，说声谢谢！我通集前后都是用通俗易懂的语言写成，便于阅读。由于本人菜鸟，不乏有不对或者个人偏见之类的地方，望各位能批正！本人定非常感谢！望各位不吝赐教！

本集是由本人一手写下来的，旨在与大家交换意见，交流学习体会。本人是谁？只是个菜鸟。当然，菜鸟不假，但是，如果让你自己给自己定一个位置的话，你会认为你是什么？我想你不会自称“高手”“大虾”吧？记得一位大哥说过，普通人用C 语言在3 年之下，一般来说，还没掌握C 语言；5 年之下，一般来说还没熟悉C 语言；10 年之下，谈不上精通。你一定很惊讶，是的，当我听到时候，也是很惊讶。也许心里莫名有种紧张了，那么接着你一定会问，那我们该怎么学呢？首先我要先问问你，在大学的四年里面，你想学到什么地步？想当年，一个个还没有开始学习C时候的“雄赳赳，气泱泱”的壮志，果然，一个月内，上C语言课的时候都是抬着头，直着眼，有的提前多少时间到教室外等着，是为了抢个好位子；再一个月内，上课时候还是抬着头，直着眼，而这个直眼就不是好的了；再一个月内，上课时候，大都会直眼了，直眼可以理解：一部分是因为听不懂，另一部分是因为走神了，而且，上课都开始提前了，而这个提前不是为了抢个好位子了，而是为了找个靠后的拐角的，因为老师不会提他问，又可以玩自己的宝贝电脑了。听到这，有些人有些体会，而有些人会有些思考，这一类人就是我们通常说的学习认真的人了。而我们知道，大学已不像我们刚过的高中那样的“拼命”的学习方式了，不是说“拼命”收获不到什么，“拼命”确实能学到很多东西。但是我们不赞成这一个学习方式，因为大学是一个综合性的环境，如果你只有学习，你必定落后，正像我现在做的一样，虽然比不上作家，但是，至少我尝试了，我起步了，并不代表我将来不会是个作家 :)。所以，一个好的适合你自己的学习方法学习模式会有一个事半功倍的效果。

我想学习过或接触过编程语言的人会有体会，计算机语言确实是一个浩浩的大海，而这个浩浩只是我们所能接触到的表面，我们还不知道这个大海的深度。而我们一直是个小小的沙粒。而在当代，学习计算机的人已不再话下，学习计算机语言的人，也是浩浩无数。所以我想借此说一下，谦虚永远是我们中华民族的传统美德，小学生在课堂上都在教育这句话，而过了二十年，真正做到这简单的一句话的人又有几个？不论何时，心怀谦虚的心学习才是真正的求知！一个花了半年时间或者更短时间学完C语言基础知识的人，别为了自己的成就冲昏了头脑。其实，你并算不上什么！小心在你这个沙粒洋洋自得之时，早已被别的沙子所卷出的大浪打到沙滩上！如果你是学习计算机专业的或想以后从事编程这一行的，那么我想你更不能少时时虚心、求知的心态。如果你抱着争第一高高在上的态度，我想早晚你会“高”到没有哪家能够容下你。因为你太高了，我只是菜鸟，这里我也没有能力去教育你什么。

好了，废话少说。最后希望各位在C语言的学习中能真正体会到C语言的精髓并找到属于自己的快乐和收获的喜悦。祝各位学业有成，工作顺利！

新际叶魂

QQ：1137142393

參考書目

谭浩强《C语言程序设计》

《C语言教程》

《算法大全》上下两册

《经典C语言程序100例》

《C语言常见问题集》

目錄

第一章 自定义C语言

一、什么是C语言

什么是C语言？我说：就是一句话——C语言就是Computer语言。

我们坐在椅子上面，手指敲打键盘，屏幕出现程序，这一过程就是我们用“数学算法”的语言在和计算机对话。计算机“听到”我们对他说的话（也就是C语言程序），他就按照我们说的去做（编译过程）。而他所做的结果，就用屏幕显示给我们，我们再“检查”他做的怎么样，要是他所做的结果不是我们所期望的，我们就要想想：是不是我对她说的话有问题，要求没说到位？还是他自身的问题（编译器的安装和配置）？（这一过程叫调试过程）。就好像，一位教师布置家庭作业，老师千言万语说作业要求格式，然后，第二天老师检查作业情况，如果一位学生作业情况不佳，老师就会在心里面盘算：是我题目抄错了？要求没说好？还是学生自身的问题？那么，接下来，就是检讨的过程了。只不过，编程时候我们和计算机不是老师和学生的关系，我想用“朋友”的关系更好些吧！如果硬要是老师和学生的关系的话，我想老师也是计算机了吧！:）

前面我说过，C语言是一门“数学算法”的语言。这样称呼并不为过。有的人就问了：你说C语言是一门“数学算法”语言，那么我的一个很简单的输出“hello world!”程序，没有一点算法啊！但是这一个程序也叫C语言程序这一过程也叫C语言编程啊！是的，这一个程序没有一点算法也叫C语言程序这一过程也叫C语言编程，但是，他只是一个程序不是一门语言。而我们在定义C语言时候，我们把所有的printf语句都叫做修饰语，甚至包括数据输出（因为数据输出是一个反馈的过程，只是让人看见他的结果，即使没有数据输出，我们要的结果已经在编译器中，只是没有反馈而已）。如果你能看清这一点，那么你就能理解为什么C语言的精髓和灵魂是——算法。

是语言，就有语法。这话一点没错。如果一门语言没有语法，每个人说得话都不一样别人都听不懂，那我们要这门语言干什么？而语法是以语句为载体的，所以这里不用说语法了，也没法说。我们只能嵌套在后面的每一节的介绍中。

二、大家说C语言

C语言的发展过程

C语言是在70年代初问世的。一九七八年由美国电话电报公司(AT&T)贝尔实验室正式发表了C语言。同时由B.W.Kernighan和D.M.Ritchit合著了著名的“THE C PROGRAMMING LANGUAGE”一书。通常简称为《K&R》，也有人称之为《K&R》标准。但是，在《K&R》中并没有定义一个完整的标准C语言，后来由美国国家标准学会在此基础上制定了一个C 语言标准，于一九八三年发表。通常称之为ANSI C。

当代最优秀的程序设计语言

早期的C语言主要是用于UNIX系统。由于C语言的强大功能和各方面的优点逐渐为人们认识，到了八十年代，C开始进入其它操作系统，并很快在各类大、中、小和微型计算机上得到了广泛的使用。成为当代最优秀的程序设计语言之一。

C语言的特点

C语言是一种结构化语言。它层次清晰，便于按模块化方式组织程序，易于调试和维护。C语言的表现能力和处理能力极强。它不仅具有丰富的运算符和数据类型，便于实现各类复杂的数据结构。它还可以直接访问内存的物理地址，进行位(bit)一级的操作。由于C语言实现了对硬件的编程操作，因此C语言集高级语言和低级语言的功能于一体。既可用于系统软件的开发，也适合于应用软件的开发。此外，C语言还具有效率高，可移植性强等特点。因此广泛地移植到了各类各型计算机上，从而形成了多种版本的C语言。

C语言版本

目前最流行的C语言有以下几种：

- Microsoft C 或称 MS C
- Borland Turbo C 或称 Turbo C
- AT&T C

这些C语言版本不仅实现了ANSI C标准，而且在此基础上各自作了一些扩充，使之更加方便、完美。

面向对象的程序设计语言

在C的基础上，一九八三年又由贝尔实验室的Bjarne Strou-strup推出了C++。 C++进一步扩充和完善了C语言，成为一种面向 对象的程序设计语言。C++目前流行的最新版本是Borland C++4.5,Symantec C++6.1,和Microsoft VisualC++ 2.0。C++提出了一些更为深入的概念，它所支持的这些面向对象的概念容易将问题空间直接地映射到程序空间，为程序员提供了一种与传统结构程序设计不同的思维方式和编程方法。因而也增加了整个语言的复杂性，掌握起来有一定难度。

C和C＋＋

但是，C是C++的基础，C++语言和C语言在很多方面是兼容的。因此，掌握了C语言，再进一步学习C++就能以一种熟悉的语法来学习面向对象的语言，从而达到事半功倍的目的。
C源程序的结构特点

为了说明C语言源程序结构的特点，先看以下几个程序。这几个程 序由简到难，表现了C语言源程序在组成结构上的特点。虽然有关内容还未介绍，但可从这些例子中了解到组成一个C源程序的基本部分和书写格式。

```
main()
{
  printf("c语言世界，您好！\n");
}
```

main是主函数的函数名，表示这是一个主函数。每一个C源程序都必须有，且只能有一个主函数(main函数)。函数调用语句，printf函数的功能是把要输出的内容送到显示器去显示。printf函数是一个由系统定义的标准函数，可在程序中直接调用。

```
#include "stdio.h"
#include "math.h"
main()
{
  double x,s;
  printf("input number:\n");
  scanf("%lf",&x);
  s=sin(x);
  printf("sine of %lf is %lf\n",x,s);
}
```

说明：include称为文件包含命令扩展名为.h的文件也称为头文件或首部文件，定义两个实数变量，以被后面程序使用，显示提示信息，从键盘获得一个实数x，求x的正弦，并把它赋给变量s，显示程序运算结果，main函数结束。

此程序的功能是从键盘输入一个数x，求x的正弦值，然后输出结果。在main()之前的两行称为预处理命令(详见后面)。预处理命令还有其它几种，这里的include 称为文件包含命令，其意义是把尖括号""或引号<>内指定的文件包含到本程序来，成为本程序的一部分。被包含的文件通常是由系统提供的，其扩展名为.h。因此也称为头文件或首部文件。C语言的头文件中包括了各个标准库函数的函数原型。因此，凡是在程序中调用一个库函数时，都必须包含该函数原型所在的头文件。在本例中，使用了三个库函数：输入函数scanf，正弦函数sin,输出函数printf。sin函数是数学函数，其头文件为math.h文件，因此在程序的主函数前用include命令包含了math.h。scanf和printf是标准输入输出函数，其头文件为stdio.h，在主函数前也用include命令包含了stdio.h文件。

需要说明的是，C语言规定对scanf和printf这两个函数可以省去对其头文件的包含命令。所以在本例中也可以删去第二行的包含命令#include。同样，在例1.1中使用了printf函数，也省略了包含命令。

在例题中的主函数体中又分为两部分，一部分为说明部分，另一部分执行部分。说明是指变量的类型说明。例题中未使用任何变量，因此无说明部分。C语言规定，源程序中所有用到的变量都必须先说明，后使用，否则将会出错。这一点是编译型高级程序设计语言的一个特点，与解释型的BASIC语言是不同的。说明部分是C源程序结构中很重要的组成部分。本例中使用了两个变量x，s，用来表示输入的自变量和sin函数值。由于sin函数要求这两个量必须是双精度浮点型，故用类型说明符double来说明这两个变量。说明部分后的四行为执行部分或称为执行语句部分,用以完成程序的功能。执行部分的第一行是输出语句,调用printf函数在显示器上输出提示字符串，请操作人员输入自变量x的值。第二行为输入语句，调用scanf函数，接受键盘上输入的数并存入变量x中。第三行是调用sin函数并把函数值送到变量s中。第四行是用printf 函数输出变量s的值，即x的正弦值。程序结束。

```
printf("input number:\n");
scanf("%lf",'C10F10&x);
s=sin(x);
printf("sine of %lf is %lf\n",'C10F10x,s);
```

运行本程序时，首先在显示器屏幕上给出提示串input number，这是由执行部分的第一行完成的。用户在提示下从键盘上键入某一数，如5，按下回车键，接着在屏幕上给出计算结果。

输入和输出函数

在前两个例子中用到了输入和输出函数scanf和 printf，在第三章中我们要详细介绍。这里我们先简单介绍一下它们的格式，以便下面使用。scanf和 printf这两个函数分别称为格式输入函数和格式输出函数。其意义是按指定的格式输入输出值。因此，这两个函数在括号中的参数表都由以下两部分组成： “格式控制串”，参数表 格式控制串是一个字符串，必须用双引号括起来，它表示了输入输出量的数据类型。各种类型的格式表示法可参阅第三章。在printf函数中还可以在格式控制串内出现非格式控制字符，这时在显示屏幕上将原文照印。参数表中给出了输入或输出的量。当有多个量时，用逗号间隔。例如：

```
printf("sine of %lf is %lf\n",x,s);
```

其中%lf为格式字符，表示按双精度浮点数处理。它在格式串中两次现，对应了x和s两个变量。其余字符为非格式字符则照原样输出在屏幕上

```
int max(int a,int b);
main()
{
  int x,y,z;
  printf("input two numbers:\n");
  scanf("%d%d",&x,&y);
  z=max(x,y);
  printf("maxmum=%d",z);
}
int max(int a,int b)
{
  if(a>b)return a;
   else return b;
}
```

此函数的功能是输入两个整数，输出其中的大数。

```
/*函数说明*/
/*主函数*/
/*变量说明*/
/*输入x,y值*/
/*调用max函数*/
/*输出*/
/*定义max函数*/
/*把结果返回主调函数*/
```

上面例中程序的功能是由用户输入两个整数，程序执行后输出其中较大的数。本程序由两个函数组成，主函数和max 函数。函数之间是并列关系。可从主函数中调用其它函数。max函数的功能是比较两个数,然后把较大的数返回给主函数。max 函数是一个用户自定义函数。因此在主函数中要给出说明(程序第三行)。可见，在程序的说明部分中，不仅可以有变量说明，还可以有函数说明。关于函数的详细内容将在第五章介绍。在程序的每行后用/*和*/括起来的内容为注释部分，程序不执行注释部分。

上例中程序的执行过程是，首先在屏幕上显示提示串，请用户输入两个数，回车后由scanf函数语句接收这两个数送入变量x,y中，然后调用max函数，并把x,y 的值传送给max函数的参数a,b。在max函数中比较a,b的大小，把大者返回给主函数的变量z，最后在屏幕上输出z的值。

C源程序的结构特点

1.一个C语言源程序可以由一个或多个源文件组成。
2.每个源文件可由一个或多个函数组成。
3.一个源程序不论由多少个文件组成，都有一个且只能有一个main函数，即主函数。
4.源程序中可以有预处理命令(include 命令仅为其中的一种),预处理命令通常应放在源文件或源程序的最前面。
5.每一个说明，每一个语句都必须以分号结尾。但预处理命令，函数头和花括号“}”之后不能加分号。
6.标识符，关键字之间必须至少加一个空格以示间隔。若已有明显的间隔符，也可不再加空格来间隔。

书写程序时应遵循的规则

从书写清晰，便于阅读，理解，维护的角度出发，在书写程序时 应遵循以下规则：

1.一个说明或一个语句占一行。
2.用{} 括起来的部分，通常表示了程序的某一层次结构。{}一般与该结构语句的第一个字母对齐，并单独占一行。
3.低一层次的语句或说明可比高一层次的语句或说明缩进若干格后书写。以便看起来更加清晰，增加程序的可读性。在编程时应力求遵循这些规则，以养成良好的编程风格。

C语言的字符集

字符是组成语言的最基本的元素。C语言字符集由字母，数字，空格，标点和特殊字符组成。在字符常量，字符串常量和注释中还可以使用汉字或其它可表示的图形符号。
1.字母　　小写字母a～z共26个，大写字母A～Z共26个

2.数字　　0～9共10个
3.空白符　空格符、制表符、换行符等统称为空白符。空白符只在字符常量和字符串常量中起作用。在其它地方出现时，只起间隔作用，　编译程序对它们忽略。因此在程序中使用空白符与否，对程序的编译不发生影响，但在程序中适当的地方使用空白符将增加程序的清晰性和可读性。
4.标点和特殊字符

C语言词汇

在C语言中使用的词汇分为六类：标识符，关键字，运算符，分隔符，常量，注释符等。
1.标识符

在程序中使用的变量名、函数名、标号等统称为标识符。除库函数的函数名由系统定义外，其余都由用户自定义。C 规定，标识符只能是字母(A～Z，a～z)、数字(0～9)、下划线()组成的字符串，并且其第一个字符必须是字母或下划线。
以下标识符是合法的：
a,x,_3x,BOOK_1,sum5
以下标识符是非法的：
3s 以数字开头
s*T 出现非法字符*
-3x 以减号开头
bowy-1 出现非法字符-(减号)
在使用标识符时还必须注意以下几点：
(1)标准C不限制标识符的长度，但它受各种版本的C 语言编译系统限制，同时也受到具体机器的限制。例如在某版本C 中规定标识符前八位有效，当两个标识符前八位相同时，则被认为是同一个标识符。
(2)在标识符中，大小写是有区别的。例如BOOK和book 是两个不同的标识符。
(3)标识符虽然可由程序员随意定义，但标识符是用于标识某个量的符号。因此，命名应尽量有相应的意义，以便阅读理解，作到“顾名思义”。

2.关键字

关键字是由C语言规定的具有特定意义的字符串，通常也称为保留字。用户定义的标识符不应与关键字相同。C语言的关键字分为以下几类：
(1)类型说明符
用于定义、说明变量、函数或其它数据结构的类型。如前面例题中用到的int,double等
(2)语句定义符
用于表示一个语句的功能。如例1.3中用到的if else就是条件语句的语句定义符。
(3)预处理命令字
用于表示一个预处理命令。如前面各例中用到的include。

3.运算符

C语言中含有相当丰富的运算符。运算符与变量，函数一起组成表达式，表示各种运算功能。运算符由一个或多个字符组成。

4.分隔符

在C语言中采用的分隔符有逗号和空格两种。逗号主要用在类型说明和函数参数表中，分隔各个变量。空格多用于语句各单词之间，作间隔符。在关键字，标识符之间必须要有一个以上的空格符作间隔， 否则将会出现语法错误，例如把int a;写成 inta;C编译器会把inta当成一个标识符处理，其结果必然出错。

5.常量

C 语言中使用的常量可分为数字常量、字符常量、字符串常量、符号常量、转义字符等多种。在第二章中将专门给予介绍。

6.注释符

C 语言的注释符是以“/*”开头并以“*/”结尾的串。在“/*”和“*/”之间的即为注释。程序编译时，不对注释作任何处理。注释可出现在程序中的任何位置。注释用来向用户提示或解释程序的意义。在调试程序中对暂不使用的语句也可用注释符括起来，使翻译跳过不作处理，待调试结束后再去掉注释符。

第二章　C语言家族

第一节　介绍家族成员

一、C语言的数据类型

在第一课中，我们已经看到程序中使用的各种变量都应预先加以说明，即先说明，后使用。对变量的说明可以包括三个方面：

- 数据类型
- 存储类型
- 作用域

在本课中，我们只介绍数据类型说明。其它说明在以后各章中陆续介绍。所谓数据类型是按被说明量的性质，表示形式，占据存储空间的多少，构造特点来划分的。在C语言中，数据类型可分为：基本数据类型，构造数据类型，指针类型，空类型四大类。

1.基本数据类型

基本数据类型最主要的特点是，其值不可以再分解为其它类型。也就是说，基本数据类型是自我说明的。

2.构造数据类型构造数据类型

是根据已定义的一个或多个数据类型用构造的方法来定义的。也就是说，一个构造类型的值可以分解成若干个“成员”或“元素”。每个“成员”都是一个基本数据类型或又是一个构造类型。在C语言中，构造类型有以下几种：

- 数组类型
- 结构类型
- 联合类型

3.指针类型

指针是一种特殊的，同时又是具有重要作用的数据类型。其值用来表示某个量在内存储器中的地址。虽然指针变量的取值类似于整型量，但这是两个类型完全不同的量，因此不能混为一谈。

4.空类型

在调用函数值时，通常应向调用者返回一个函数值。这个返回的函数值是具有一定的数据类型的，应在函数定义及函数说明中给以说明，例如在例题中给出的max函数定义中，函数头为： int max(int a,int b);其中“int ”类型说明符即表示该函数的返回值为整型量。又如在例题中，使用了库函数 sin，由于系统规定其函数返回值为双精度浮点型，因此在赋值

语句s=sin (x);中，s 也必须是双精度浮点型，以便与sin函数的返回值一致。所以在说明部分，把s说明为双精度浮点型。但是，也有一类函数，调用后并不需要向调用者返回函数值， 这种函数可以定义为“空类型”。其类型说明符为void。在第五章函数中还要详细介绍。在本章中，我们先介绍基本数据类型中的整型、浮点型和字符型。其余类型在以后各章中陆续介绍。

对于基本数据类型量，按其取值是否可改变又分为常量和变量两种。在程序执行过程中，其值不发生改变的量称为常量，取值可变的量称为变量。它们可与数据类型结合起来分类。例如，可分为整型常量、整型变量、浮点常量、浮点变量、字符常量、字符变量、枚举常量、枚举变量。在程序中，常量是可以不经说明而直接引用的，而变量则必须先说明后使用。
整型量

整型量包括整型常量、整型变量。整型常量就是整常数。在C语言中，使用的整常数有八进制、十六进制和十进制三种。

整型常量

1.八进制整常数八进制整常数必须以0开头，即以0作为八进制数的前缀。数码取值为0～7。八进制数通常是无符号数。
以下各数是合法的八进制数：
015(十进制为13)　0101(十进制为65)　0177777(十进制为65535)
以下各数不是合法的八进制数：
256(无前缀0)　03A2(包含了非八进制数码)　-0127(出现了负号)

2.十六进制整常数
十六进制整常数的前缀为0X或0x。其数码取值为0~9，A~F或a~f。
以下各数是合法的十六进制整常数：
0X2A(十进制为42)　　0XA0 (十进制为160)　　0XFFFF (十进制为65535)
以下各数不是合法的十六进制整常数：
5A (无前缀0X)　　0X3H (含有非十六进制数码)

3.十进制整常数
十进制整常数没有前缀。其数码为0～9。
以下各数是合法的十进制整常数：
237 -568 65535 1627
以下各数不是合法的十进制整常数：
023 (不能有前导0)　23D (含有非十进制数码)

在程序中是根据前缀来区分各种进制数的。因此在书写常数时不要把前缀弄错造成结果不正确。4.整型常数的后缀在16位字长的机器上，基本整型的长度也为16位，因此表示的数的范围也是有限定的。十进制无符号整常数的范围为0～65535，有符号数为-32768～+32767。八进制无符号数的表示范围为0～0177777。十六进制无符号数的表示范围为0X0～0XFFFF或0x0～0xFFFF。如果使用的数超过了上述范围，就必须用长整型数来表示。长整型数是用后缀“L”或“l”来表示的。例如：

十进制长整常数 158L (十进制为158) 358000L (十进制为-358000)
八进制长整常数 012L (十进制为10) 077L (十进制为63) 0200000L (十进制为65536)
十六进制长整常数 0X15L (十进制为21) 0XA5L (十进制为165) 0X10000L (十进制为65536)

长整数158L和基本整常数158 在数值上并无区别。但对158L，因为是长整型量，C编译系统将为它分配4个字节存储空间。而对158，因为是基本整型，只分配2 个字节的存储空间。因此在运算和输出格式上要予以注意，避免出错。无符号数也可用后缀表示，整型常数的无符号数的后缀为“U”或“u”。例如： 358u,0x38Au,235Lu 均为无符号数。前缀，后缀可同时使用以表示各种类型的数。如0XA5Lu表示十六进制无符号长整数A5，其十进制为165。

整型变量

整型变量可分为以下几类：
1.基本型
类型说明符为int，在内存中占2个字节，其取值为基本整常数。
2.短整量
类型说明符为short int或short'C110F1。所占字节和取值范围均与基本型相同。
3.长整型
类型说明符为long int或long ，在内存中占4个字节，其取值为长整常数。
4.无符号型
类型说明符为unsigned。
无符号型又可与上述三种类型匹配而构成：
(1)无符号基本型 类型说明符为unsigned int或unsigned。
(2)无符号短整型 类型说明符为unsigned short
(3)无符号长整型 类型说明符为unsigned long
各种无符号类型量所占的内存空间字节数与相应的有符号类型量相同。但由于省去了符号位，故不能表示负数。 下表列出了Turbo C中各类整型量所分配的内存字节数及数的表示范围。

类型说明符	数的范围
int	-32768~32767
short int	-32768~32767
signed int	-32768~32767
unsigned int	0~65535
long int	-2147483648~2147483647
unsigned long	0~4294967295

整型变量的说明
变量说明的一般形式为： 类型说明符 变量名标识符，变量名标识符，...; 例如：
int a,b,c; (a,b,c为整型变量)
long x,y; (x,y为长整型变量)
unsigned p,q; (p,q为无符号整型变量)
在书写变量说明时，应注意以下几点：
1.允许在一个类型说明符后，说明多个相同类型的变量。各变量名之间用逗号间隔。类型说明符与变量名之间至少用一个空格间隔。
2.最后一个变量名之后必须以“; ”号结尾。

3.变量说明必须放在变量使用之前。一般放在函数体的开头部分。

```
void main(){
long x,y;
int a,b,c,d;
x=5;
y=6;
a=7;
b=8;
c=x+a;
d=y+b;
printf("c=x+a=%d,d=y+b=%d\n",c,d);
}
```

将main说明为返回void,即不返回任何类型的值

x,y被定义为long型

a,b,c,d被定义为int型

5->x

6->y

7->a

8->b

x+a->c

y+b->d

显示程序运行结果 of long x,y;

```
int a,b,c,d;
c=x+a;
d=y+b;
```

从程序中可以看到：x,y是长整型变量，a,b是基本整型变量。它们之间允许进行运算，运算结果为长整型。但c，d被定义为基本整型，因此最后结果为基本整型。本例说明，不同类型的量可以参与运算并相互赋值。其中的类型转换是由编译系统自动完成的。有关类型转换的规则将在以后介绍。

实型量

实型常量

实型也称为浮点型。实型常量也称为实数或者浮点数。在C语言中，实数只采用十进制。它有二种形式： 十进制数形式指数形式

1.十进制数形式

由数码0~ 9和小数点组成。例如：0.0，.25，5.789，0.13，5.0，300.，-267.8230等均为合法的实数。

2.指数形式

由十进制数，加阶码标志“e”或“E”以及阶码（只能为整数，可以带符号）组成。其一般形式为a E n（a为十进制数，n为十进制整数）其值为 a*10,n 如： 2.1E5 (等于2.1*10,5), 3.7E-2 (等于3.7*10,)-2*) 0.5E7 (等于0.5*10,7), -2.8E-2 (等于-2.8*10,)-2*)以下不是合法的实数 345 (无小数点) E7 (阶码标志E之前无数字) -5 (无阶码标志) 53.-E3

(负号位置不对)　2.7E (无阶码)
标准C允许浮点数使用后缀。后缀为“f”或“F”即表示该数为浮点数。如356f和356.是等价的。例2.2说明了这种情况：

```
void main()
{
 printf("%f\n%f\n",356.,356f);
}
```

void 指明main不返回任何值　利用printf显示结果　结束

实型变量

实型变量分为两类：单精度型和双精度型，
其类型说明符为float 单精度说明符，double 双精度说明符。在Turbo C中单精度型占4个字节（32位）内存空间，其数值范围为3.4E-38～3.4E+38，只能提供七位有效数字。双精度型占8 个字节（64位）内存空间，其数值范围为1.7E-308～1.7E+308，可提供16位有效数字。
实型变量说明的格式和书写规则与整型相同。
例如：　float x,y; (x,y为单精度实型量)
　　double a,b,c; (a,b,c为双精度实型量)
实型常数不分单、双精度，都按双精度double型处理。

```
void main()
{
  float a;
  double b;
  a=33333.33333;
  b=33333.33333333333333;
  printf("%f\n%f\n",a,b);
}
```

此程序说明float、double的不同

```
float a;
double b;
a=33333.33333;
b=33333.33333333333333;
```

从本例可以看出，由于a 是单精度浮点型，有效位数只有七位。而整数已占五位，故小数二位后之后均为无效数字。b 是双精度型，有效位为十六位。但Turbo C 规定小数后最多保留六位，其余部分四舍五入。

字符型量
字符型量包括字符常量和字符变量。

字符常量
字符常量是用单引号括起来的一个字符。例如'a','b','=','+','?'都是合法字符常量。在C语言中，字符常量有以下特点：
1.字符常量只能用单引号括起来，不能用双引号或其它括号。

2.字符常量只能是单个字符，不能是字符串。
3.字符可以是字符集中任意字符。但数字被定义为字符型之后就
不能参与数值运算。如'5'和5 是不同的。'5'是字符常量，不能参与运算。

转义字符
转义字符是一种特殊的字符常量。转义字符以反斜线"\"开头，后跟一个或几个字符。转义字符具有特定的含义，不同于字符原有的意义，故称“转义”字符。例如，在前面各例题printf函数的格式串中用到的“\n”就是一个转义字符，其意义是“回车换行”。转义字符主要用来表示那些用一般字符不便于表示的控制代码。
常用的转义字符及其含义

转义字符	转义字符的意义
\n	回车换行
\t	横向跳到下一制表位置
\v	竖向跳格
\b	退格
\r	回车
\f	走纸换页
\\	反斜线符"\"
\'	单引号符
\a	鸣铃
\ddd	1～3位八进制数所代表的字符
\xhh	1～2位十六进制数所代表的字符

广义地讲，C语言字符集中的任何一个字符均可用转义字符来表示。表2.2中的\ddd和\xhh正是为此而提出的。ddd和hh分别为八进制和十六进制的ASCII代码。如\101表示字“A”，\102表示字母"B"，\134表示反斜线，\XOA表示换行等。转义字符的使用

```
void main()
{
int a,b,c;
a=5; b=6; c=7;
printf("%d\n\t%d %d\n %d %d\t\b%d\n",a,b,c,a,b,c);
}
```

此程序练习转义字符的使用
a、b、c为整数 5->a,6->b,7->c
调用printf显示程序运行结果

```
printf("%d\n\t%d %d\n %d %d\t\b%d\n",a,b,c,a,b,c);
```

程序在第一列输出a值5之后就是“\n”，故回车换行；接着又是“\t”，于是跳到下一制表位置（设制表位置间隔为8），再输出b值6；空二格再输出c 值7后又是"\n"，因此再回车换行；再空二格之后又输出a值5；再空三格又输出b的值6；再次后"\t"跳到下一制表位置（与上一行的6 对齐），但下一转义字符“\b”又使退回一格，故紧挨着6再输出c值7。

字符变量
字符变量的取值是字符常量，即单个字符。字符变量的类型说明符是char。字符变量类型说明的格式和书写规则都与整型变量相同。
例如：

char a,b; 每个字符变量被分配一个字节的内存空间，因此只能存放一个字符。字符值是以ASCII码的形式存放在变量的内存单元之中的。如x的
十进制ASCII码是120，y的十进制ASCII码是121。对字符变量a,b赋予'x'和'y'值：a='x';b='y';实际上是在a,b两个单元内存放120和121的二进制代码： a 0 1 1 1 1 0 0 0
b 0 1 1 1 1 0 0 1
所以也可以把它们看成是整型量。 C语言允许对整型变量赋以字符值，也允许对字符变量赋以整型值。在输出时， 允许把字符变量按整型量输出，也允许把整型量按字符量输出。 整型量为二字节量，字符量为单字节量，当整型量按字符型量处理时， 只有低八位字节参与处理。

```
main()
{
  char a,b;
  a=120;
  b=121;
  printf("%c,%c\n%d,%d\n",a,b,a,b);
}
```

本程序中说明a，b为字符型，但在赋值语句中赋以整型值。从结果看，a，b值的输出形式取决于printf函数格式串中的格式符，当格式符为"c"时，对应输出的变量值为字符，当格式符为"d"时，对应输出的变量值为整数。

```
void main()
{
  char a,b;
  a='x';
  b='y';
  a=a-32;
  b=b-32;
  printf("%c,%c\n%d,%d\n",a,b,a,b);
}
```

a，b被说明为字符变量并赋予字符值
把小写字母换成大写字母
以整型和字符型输出
本例中，a，b被说明为字符变量并赋予字符值，C语言允许字符变量参与数值运算，即用字符的ASCII 码参与运算。由于大小写字母的ASCII 码相差32，因此运算后把小写字母换成大写字母。然后分别以整型和字符型输出。

字符串常量
字符串常量是由一对双引号括起的字符序列。例如： "CHINA" ，"C program:" ， "$12.5" 等都是合法的字符串常量。字符串常量和字符常量是不同的量。它们之间主要有以下区别：
1.字符常量由单引号括起来，字符串常量由双引号括起来。
2.字符常量只能是单个字符，字符串常量则可以含一个或多个字符。
3.可以把一个字符常量赋予一个字符变量，但不能把一个字符串常量赋予一个字符变量。在C语言中没有相应的字符串变量。
这是与BASIC 语言不同的。但是可以用一个字符数组来存放一个字符串常量。在数组一章内

予以介绍。
4.字符常量占一个字节的内存空间。字符串常量占的内存字节数等于字符串中字节数加1。增加的一个字节中存放字符"\0"(ASCII码为0)。这是字符串结束的标志。例如，字符串 "C program"在内存中所占的字节为：C program\0。字符常量'a'和字符串常量"a"虽然都只有一个字符，但在内存中的情况是不同的。
'a'在内存中占一个字节，可表示为：a
"a"在内存中占二个字节，可表示为：a\0符号常量

符号常量
在C语言中，可以用一个标识符来表示一个常量，称之为符号常量。符号常量在使用之前必须先定义，其一般形式为：
#define 标识符 常量
其中#define也是一条预处理命令（预处理命令都?quot;#"开头），称为宏定义命令（在第九章预处理程序中将进一步介绍），其功能是把该标识符定义为其后的常量值。一经定义，以后在程序中所有出现该标识符的地方均代之以该常量值。习惯上符号常量的标识符用大写字母，变量标识符用小写字母，以示区别。

```
#define PI 3.14159
void main()
{
  float s,r;
  r=5;
  s=PI*r*r;
  printf("s=%f\n",s);
}
```

由宏定义命令定义PI 为3.14159 s,r定义为实数 5->r PI*r*r->s
显示程序结果 float s,r; r=5; s=PI*r*r; 本程序在主函数之前由宏定义命令定义PI为3.14159，在程序中即以该值代替PI 。s=PI*r*r等效于s=3.14159*r*r。应该注意的是，符号常量不是变量，它所代表的值在整个作用域内不能再改变。也就是说，在程序中，不能再用赋值语句对它重新赋值。

变量的初值和类型转换

变量赋初值
在程序中常常需要对变量赋初值，以便使用变量。语言程序中可有多种方法，在定义时赋以初值的方法，这种方法称为初始化。在变量说明中赋初值的一般形式为：
类型说明符 变量1= 值1，变量2= 值2，……； 例如：

```
int a=b=c=5;
float x=3.2,y=3f,z=0.75;
char ch1='K',ch2='P';
```

应注意，在说明中不允许连续赋值，如a=b=c=5是不合法的。

```
void main()
{
  int a=3,b,c=5;
  b=a+c;
```

```
  printf("a=%d,b=%d,c=%d\n",a,b,c);
}
a<---3,b<--0,c<---5
b<--a+c
```

显示程序运行结果

变量类型的转换

变量的数据类型是可以转换的。转换的方法有两种， 一种是自动转换，一种是强制转换。

自动转换

自动转换发生在不同数据类型的量混合运算时，由编译系统自动完成。自动转换遵循以下规则：

1.若参与运算量的类型不同，则先转换成同一类型，然后进行运算。

2.转换按数据长度增加的方向进行，以保证精度不降低。如int型和long型运算时，先把int量转成long型后再进行运算。

3.所有的浮点运算都是以双精度进行的，即使仅含float单精度量运算的表达式，也要先转换成double型，再作运算。

4.char型和short型参与运算时，必须先转换成int型。

5.在赋值运算中，赋值号两边量的数据类型不同时， 赋值号右边量的类型将转换为左边量的类型。 如果右边量的数据类型长度左边长时，将丢失一部分数据，这样会降低精度， 丢失的部分按四舍五入向前舍入。图2 1表示了类型自动转换的规则。

```
void main()
{
  float PI=3.14159;
  int s,r=5;
  s=r*r*PI;
  printf("s=%d\n",s);
}
PI<--3.14159
s<--0,r<--5
s<--r*r*PI
```

显示程序运行结果

```
float PI=3.14159;
int s,r=5;
s=r*r*PI;
```

本例程序中，PI为实型；s，r为整型。在执行s=r*r*PI语句时，r和PI都转换成double型计算，结果也为double型。但由于s为整型，故赋值结果仍为整型，舍去了小数部分。

强制类型转换

强制类型转换是通过类型转换运算来实现的。其一般形式为： (类型说明符) (表达式) 其功能是把表达式的运算结果强制转换成类型说明符所表示的类型。例如： (float) a 把a转换为实型(int)(x+y) 把x+y的结果转换为整型在使用强制转换时应注意以下问题：

1.类型说明符和表达式都必须加括号(单个变量可以不加括号)，如把(int)(x+y)写成(int)x+y则成了把x转换成int型之后再与y相加了。

2.无论是强制转换或是自动转换，都只是为了本次运算的需要而对变量的数据长度进行的临

时性转换，而不改变数据说明时对该变量定义的类型。

```
main()
{
  float f=5.75;
  printf("(int)f=%d,f=%f\n",(int)f,f);
}
```

f<--5.75

将float f强制转换成int f float f=5.75;printf("(int)f=%d,f=%f\n",(int)f,f); 本例表明，f虽强制转为int型，但只在运算中起作用， 是临时的，而f本身的类型并不改变。因此，(int)f的值为 5(删去了小数)而f的值仍为5.75。

二、基本运算符和表达式

运算符的种类、优先级和结合性

C语言中运算符和表达式数量之多， 在高级语言中是少见的。正是丰富的运算符和表达式使C语言功能十分完善。 这也是C语言的主要特点之一。

C语言的运算符不仅具有不同的优先级， 而且还有一个特点，就是它的结合性。在表达式中， 各运算量参与运算的先后顺序不仅要遵守运算符优先级别的规定，还要受运算符结合性的制约， 以便确定是自左向右进行运算还是自右向左进行运算。 这种结合性是其它高级语言的运算符所没有的，因此也增加了C语言的复杂性。

运算符的种类C语言的运算符可分为以下几类：

1.算术运算符

用于各类数值运算。包括加(+)、减(-)、乘(*)、除(/)、求余(或称模运算，%)、自增(++)、自减(--)共七种。

2.关系运算符

用于比较运算。包括大于(>)、小于(<)、等于(==)、 大于等于(>=)、小于等于(<=)和不等于(!=)六种。

3.逻辑运算符

用于逻辑运算。包括与(&&)、或(||)、非(!)三种。

4.位操作运算符

参与运算的量，按二进制位进行运算。包括位与(&)、位或(|)、位非(~)、位异或(^)、左移(<<)、右移(>>)六种。

5.赋值运算符

用于赋值运算，分为简单赋值(=)、复合算术赋值(+=,-=,*=,/=,%=)和复合位运算赋值(&=,|=,^=,>>=,<<=)三类共十一种。

6.条件运算符

这是一个三目运算符，用于条件求值(?:)。

7.逗号运算符

用于把若干表达式组合成一个表达式(，)。

8.指针运算符

用于取内容(*)和取地址(&)二种运算。

9.求字节数运算符

用于计算数据类型所占的字节数(sizeof)。
10.特殊运算符
有括号()，下标[]，成员(→，.)等几种。

优先级和结合性
C语言中，运算符的运算优先级共分为15级。1级最高，15级最低。在表达式中，优先级较高的先于优先级较低的进行运算。 而在一个运算量两侧的运算符优先级相同时， 则按运算符的结合性所规定的结合方向处理。 C语言中各运算符的结合性分为两种，即左结合性(自左至右)和右结合性(自右至左)。例如算术运算符的结合性是自左至右，即先左后右。如有表达式x-y+z则y应先与“-”号结合， 执行x-y运算，然后再执行+z的运算。这种自左至右的结合方向就称为“左结合性”。而自右至左的结合方向称为“右结合性”。 最典型的右结合性运算符是赋值运算符。如x=y=z,由于“=”的右结合性，应先执行y=z再执行x=(y=z)运算。 C语言运算符中有不少为右结合性，应注意区别，以避免理解错误。

算术运算符和算术表达式基本的算术运算符
1.加法运算符“+”加法运算符为双目运算符，即应有两个量参与加法运算。如a+b,4+8等。具有右结合性。
2.减法运算符“-”减法运算符为双目运算符。但“-”也可作负值运算符,此时为单目运算,如-x,-5等具有左结合性。
3.乘法运算符“*”双目运算，具有左结合性。
4.除法运算符“/”双目运算具有左结合性。参与运算量均为整型时， 结果也为整型，舍去小数。如果运算量中有一个是实型，则结果为双精度实型。

```
void main(){
printf("\n\n%d,%d\n",20/7,-20/7);
printf("%f,%f\n",20.0/7,-20.0/7);
}
```

双目运算具有左结合性。参与运算量均为整型时， 结果也为整型，舍去小数。如果运算量中有一个是实型，则结果为双精度实型。 printf("\n\n%d,%d\n",20/7,-20/7);
printf("%f,%f\n",20.0/7,-20.0/7);
本例中，20/7，-20/7的结果均为整型，小数全部舍去。而20.0/7和-20.0/7由于有实数参与运算，因此结果也为实型。
5.求余运算符(模运算符)“%”双目运算，具有左结合性。要求参与运算的量均为整型。 求余运算的结果等于两数相除后的余数。

```
void main(){
printf("%d\n",100%3);
}
```

双目运算，具有左结合性。求余运算符% 要求参与运算的量均为整型。本例输出100除以3所得的余数1。

自增 1，自减 1 运算符
自增1运算符记为“++”，其功能是使变量的值自增1。自减1运算符记为“--”，其功能是使变量值自减1。自增1，自减1运算符均为单目运算，都具有右结合性。可有以下几种形式：
++i i自增1后再参与其它运算。--i i自减1后再参与其它运算。
i++ i参与运算后，i的值再自增1。

i--　i参与运算后，i的值再自减1。
在理解和使用上容易出错的是i++和i--。 特别是当它们出在较复杂的表达式或语句中时，常常难于弄清，因此应仔细分析。

```
void main(){
int i=8;
printf("%d\n",++i);
printf("%d\n",--i);
printf("%d\n",i++);
printf("%d\n",i--);
printf("%d\n",-i++);
printf("%d\n",-i--);
} i<--8
i<--i+1
i<--i-1
i<--i+1
i<--i-1
i<--i+1
i<--i-1 int i=8;
printf("%d\n",++i);
printf("%d\n",--i);
printf("%d\n",i++);
printf("%d\n",i--);
printf("%d\n",-i++);
printf("%d\n",-i--);
```

i的初值为8
第2行i加1后输出故为9；
第3行减1后输出故为8；
第4行输出i为8之后再加1(为9)；
第5行输出i为9之后再减1(为8) ；
第6行输出-8之后再加1(为9)；
第7行输出-9之后再减1(为8)

```
void main(){
int i=5,j=5,p,q;
p=(i++)+(i++)+(i++);
q=(++j)+(++j)+(++j);
printf("%d,%d,%d,%d",p,q,i,j);
}
i<--5,j<--5,p<--0,q<--0
i+i+i--->p,i+1-->i,i+1-->i,i+1-->i
j+1->j,j+1->j,j+1->j,j+j+j->q int i=5,j=5,p,q;
p=(i++)+(i++)+(i++);
q=(++j)+(++j)+(++j);
```

这个程序中，对P=(i++)+(i++)+(i++)应理解为三个i相加，故P值为15。然后i再自增1三次相当于加3故i的最后值为8。而对于q 的值则不然，q=(++j)+(++j)+(++j)应理解为q先自增1，

再参与运算，由于q自增1三次后值为8，三个8相加的和为24，j的最后值仍为8。算术表达式表达式是由常量、变量、函数和运算符组合起来的式子。一个表达式有一个值及其类型，它们等于计算表达式所得结果的值和类型。表达式求值按运算符的优先级和结合性规定的顺序进行。 单个的常量、变量、函数可以看作是表达式的特例。

算术表达式
是由算术运算符和括号连接起来的式子， 以下是算术表达式的例子：
a+b (a*2)/c (x+r)*8-(a+b)/7 ++i sin(x)+sin(y) (++i)-(j++)+(k--)

赋值运算符和赋值表达式
简单赋值运算符和表达式，简单赋值运算符记为“=”。由“= ”连接的式子称为赋值表达式。其一般形式为： 变量=表达式 例如：
x=a+b
w=sin(a)+sin(b)
y=i+++--j 赋值表达式的功能是计算表达式的值再赋予左边的变量。赋值运算符具有右结合性。因此
a=b=c=5
可理解为
a=(b=(c=5))
在其它高级语言中，赋值构成了一个语句，称为赋值语句。 而在C中，把“=”定义为运算符，从而组成赋值表达式。 凡是表达式可以出现的地方均可出现赋值表达式。例如，式子x=(a=5)+(b=8)是合法的。它的意义是把5赋予a，8赋予b，再把a,b相加，和赋予x ，故x应等于13。
在C语言中也可以组成赋值语句，按照C语言规定， 任何表达式在其未尾加上分号就构成为语句。因此如x=8;a=b=c=5;都是赋值语句，在前面各例中我们已大量使用过了。
如果赋值运算符两边的数据类型不相同， 系统将自动进行类型转换，即把赋值号右边的类型换成左边的类型。具体规定如下：
1.实型赋予整型，舍去小数部分。前面的例2.9已经说明了这种情况。
2.整型赋予实型，数值不变，但将以浮点形式存放， 即增加小数部分(小数部分的值为0)。
3.字符型赋予整型，由于字符型为一个字节， 而整型为二个字节，故将字符的ASCII码值放到整型量的低八位中，高八位为0。
4.整型赋予字符型，只把低八位赋予字符量。

```
void main(){
int a,b=322;
float x,y=8.88;
char c1='k',c2;
a=y;
x=b;
a=c1;
c2=b;
printf("%d,%f,%d,%c",a,x,a,c2);
}
int a,b=322;
float x,y=8.88;
```

```
char c1='k',c2;
printf("%d,%f,%d,%c",a=y,x=b,a=c1,c2=b);
```

本例表明了上述赋值运算中类型转换的规则。a为整型，赋予实型量y值8　88后只取整数8。x为实型，赋予整型量b值322，　后增加了小数部分。字符型量c1赋予a变为整型，整型量b赋予c2 后取其低八位成为字符型(b的低八位为01000010，即十进制66，按ASCII码对应于字符B)。

复合赋值符及表达式

在赋值符“=”之前加上其它二目运算符可构成复合赋值符。如

+=,-=,*=,／=,%=,<<=,>>=,&=,^=,|=。 构成复合赋值表达式的一般形式为： 变量 双目运算符=表达式 它等效于 变量=变量 运算符 表达式 例如： a+=5 等价于a=a+5　　x*=y+7 等价于x=x*(y+7)　　r%=p 等价于r=r%p

复合赋值符这种写法，对初学者可能不习惯， 但十分有利于编译处理，能提高编译效率并产生质量较高的目标代码。逗号运算符和逗号表达式在

逗号运算符

Ｃ语言中逗号“，”也是一种运算符，称为逗号运算符。 其功能是把两个表达式连接起来组成一个表达式， 称为逗号表达式。

其一般形式为： 表达式1，表达式2 其求值过程是分别求两个表达式的值，并以表达式2的值作为整个逗号表达式的值。

```
void main(){
int a=2,b=4,c=6,x,y;
x=a+b,y=b+c;
printf("y=%d,x=%d",y,x);
}
```

```
a<--2,b<--4,c<--6,x<--0,y<--0
x<--a+b,y<---b+c
```

本例中，y等于整个逗号表达式的值，也就是表达式2的值，x是第一个表达式的值。对于逗号表达式还要说明两点：

1.逗号表达式一般形式中的表达式1和表达式2 也可以又是逗号表达式。例如：表达式1,(表达式2，表达式3) 形成了嵌套情形。因此可以把逗号表达式扩展为以下形式： 表达式1，表达式2，…表达式n 整个逗号表达式的值等于表达式n的值。

2.程序中使用逗号表达式，通常是要分别求逗号表达式内各表达式的值，并不一定要求整个逗号表达式的值。

3.并不是在所有出现逗号的地方都组成逗号表达式，如在变量说明中，函数参数表中逗号只是用作各变量之间的间隔符。

例如　　int a,b,c;

第二节　建立友好关系——设计初步

赋值语句

赋值语句是由赋值表达式再加上分号构成的表达式语句。其一般形式为：变量=表达式；赋值语句的功能和特点都与赋值表达式相同。 它是程序中使用最多的语句之一。 在赋值语句的使用中需要注意以下几点：

1.由于在赋值符“=”右边的表达式也可以又是一个赋值表达式，因此，下述形式 变量=(变量=表达式)；是成立的，从而形成嵌套的情形。其展开之后的一般形式为： 变量=变量=…=表达式;
例如：
a=b=c=d=e=5;按照赋值运算符的右接合性，因此实际上等效于：
e=5;d=e;c=d;b=c;a=b;
2.注意在变量说明中给变量赋初值和赋值语句的区别。给变量赋初值是变量说明的一部分，赋初值后的变量与其后的其它同类变量之间仍必须用逗号间隔，而赋值语句则必须用分号结尾。
3.在变量说明中，不允许连续给多个变量赋初值。 如下述说明是错误的： int a=b=c=5 必须写为 int a=5,b=5,c=5; 而赋值语句允许连续赋值
4.注意赋值表达式和赋值语句的区别。赋值表达式是一种表达式，它可以出现在任何允许表达式出现的地方，而赋值语句则不能。
下述语句是合法的： if((x=y+5)>0) z=x; 语句的功能是，若表达式x=y+5大于0则z=x。下述语句是非法的： if((x=y+5;)>0) z=x; 因为=y+5;是语句，不能出现在表达式中。

数据输出语句

本小节介绍的是向标准输出设备显示器输出数据的语句。在C语言中，所有的数据输入/输出都是由库函数完成的。 因此都是函数语句。本小节先介绍printf函数和putchar函数。printf函数printf函数称为格式输出函数，其关键字最末一个字母f即为“格式”(format)之意。其功能是按用户指定的格式， 把指定的数据显示到显示器屏幕上。在前面的例题中我们已多次使用过这个函数。

一、printf函数调用的一般形式

printf函数是一个标准库函数，它的函数原型在头文件“stdio.h”中。但作为一个特例，不要求在使用 printf 函数之前必须包含stdio.h文件。printf函数调用的一般形式为：printf(“格式控制字符串”，输出表列)其中格式控制字符串用于指定输出格式。 格式控制串可由格式字符串和非格式字符串两种组成。格式字符串是以%开头的字符串，在%后面跟有各种格式字符，以说明输出数据的类型、形式、长度、小数位数等。如“%d”表示按十进制整型输出，“%ld”表示按十进制长整型输出，“%c”表示按字符型输出等。后面将专门给予讨论。

非格式字符串在输出时原样照印,在显示中起提示作用。输出表列中给出了各个输出项，要求格式字符串和各输出项在数量和类型上应该一一对应。

```
void main()
{
int a=88,b=89;
printf("%d %d\n",a,b);
printf("%d,%d\n",a,b);
printf("%c,%c\n",a,b);
printf("a=%d,b=%d",a,b);
}
a<--8,b<--89
printf("%d %d\n",a,b);
printf("%d,%d\n",a,b);
printf("%c,%c\n",a,b);
printf("a=%d,b=%d",a,b);
```

本例中四次输出了a,b的值，但由于格式控制串不同，输出的结果也不相同。第四行的输出语句格式控制串中，两格式串%d 之间加了一个空格(非格式字符)，所以输出的a,b值之间有一个空格。第五行的printf语句格式控制串中加入的是非格式字符逗号， 因此输出的a,b值之间加了一个逗号。第六行的格式串要求按字符型输出 a,b值。第七行中为了提示输出结果又增加了非格式字符串。

二、格式字符串

在Turbo C中格式字符串的一般形式为： [标志][输出最小宽度][.精度][长度]类型 其中方括号[]中的项为可选项。各项的意义介绍如下：

1.类型类型字符用以表示输出数据的类型，其格式符和意义下表所示：

表示输出类型的格式字符	格式字符意义
d	以十进制形式输出带符号整数(正数不输出符号)
o	以八进制形式输出无符号整数(不输出前缀O)
x	以十六进制形式输出无符号整数(不输出前缀OX)
u	以十进制形式输出无符号整数
f	以小数形式输出单、双精度实数
e	以指数形式输出单、双精度实数
g	以%f%e中较短的输出宽度输出单、双精度实数
c	输出单个字符
s	输出字符串

2.标志

标志字符为-、+、#、空格四种，其意义下表所示：

标志格式字符	标 志 意 义
-	结果左对齐，右边填空格
+	输出符号(正号或负号)空格输出值为正时冠以空格,为负时冠以负号
#	对c，s，d，u类无影响；对o类， 在输出时加前
前缀o	对x类，在输出时加前缀0x；对e,g,f 类当结果有小数时才给出小数点

3.输出最小宽度
用十进制整数来表示输出的最少位数。若实际位数多于定义的宽度,则按实际位数输出，若实际位数少于定义的宽度则补以空格或0。
4.精度
精度格式符以“.”开头，后跟十进制整数。本项的意义是：如果输出数字，则表示小数的位数；如果输出的是字符， 则表示输出字符的个数；若实际位数大于所定义的精度数，则截去超过的部分。
5.长度
长度格式符为h,l两种，h表示按短整型量输出，l表示按长整型量输出。

```
void main(){
int a=15;
float b=138.3576278;
double c=35648256.3645687;
char d='p';
printf("a=%d,%5d,%o,%x\n",a,a,a,a);
printf("b=%f,%lf,%5.4lf,%e\n",b,b,b,b);
printf("c=%lf,%f,%8.4lf\n",c,c,c);
printf("d=%c,%8c\n",d,d);
} a<--15
b<--138.3576278
c<--35648256.3645687
d<--'p'

main()
{
int a=29;
float b=1243.2341;
double c=24212345.24232;
char d='h';
printf("a=%d,%5d,%o,%x\n",a,a,a,a);
printf("b=%f,%lf,%5.4lf,%e\n",b,b,b,b);
printf("c=%lf,%f,%8.4lf\n",c,c,c);
printf("d=%c,%8c\n",d,d);
}
```

本例第七行中以四种格式输出整型变量a的值，其中“%5d ”要求输出宽度为5，而a值为15只有两位故补三个空格。 第八行中以四种格式输出实型量b的值。其中“%f”和“%lf ”格式的输出相同，说明“l”符对“f”类型无影响。“%5.4lf”指定输出宽度为5，精度为4，由于实际长度超过5故应该按实际位数输出，小数位数超过4位部分被截去。第九行输出双精度实数，“%8.4lf ”由于指定精度为4位故截去了超过4位的部分。第十行输出字符量d，其中“%8c ”指定输出宽度为8故在输出字符p之前补加7个空格。

使用printf函数时还要注意一个问题， 那就是输出表列中的求值顺序。不同的编译系统不一定相同，可以从左到右， 也可从右到左。Turbo C是按从右到左进行的。如把例2.13改写如下述形式：

```
void main(){
int i=8;
printf("%d\n%d\n%d\n%d\n%d\n%d\n",++i,--i,i--,i++,-i--);
} i<--8
```

这个程序与例2.13相比只是把多个printf语句改一个printf 语句输出。但从结果可以看出是不同的。为什么结果会不同呢?就是因为printf函数对输出表中各量求值的顺序是自右至左进行 的。在式中，先对最后一项“-i--”求值，结果为-8,然后i自减1后为7。 再对“-i++”项求值得-7，然后i自增1后为8。再对“i--”项求值得8，然后i再自减1后为7。再求“i++”项得7，然后I再自增1后为8。 再求“--i”项，i先自减1后输出，输出值为7。 最后才求输出表列中的第一项“++i”，此时i自增1后输出8。但是必须注意， 求值顺序虽是自右至左，但是输出顺序还是从左至右， 因此得到的结果是上述输出结果。

字符输出函数

putchar 函数

putchar 函数是字符输出函数， 其功能是在显示器上输出单个字符。其一般形式为：
putchar(字符变量) 例如：
putchar('A'); 输出大写字母A
putchar(x); 输出字符变量x的值
putchar('\n'); 换行 对控制字符则执行控制功能，不在屏幕上显示。 使用本函数前必须要用文件包含命令：

```
#include<stdio.h>
void main(){
char a='B',b='o',c='k';
putchar(a);putchar(b);putchar(b);putchar(c);putchar('\t');
putchar(a);putchar(b);
putchar('\n');
putchar(b);putchar(c);
}
```

数据输入语句

C语言的数据输入也是由函数语句完成的。 本节介绍从标准输入设备—键盘上输入数据的函数scanf和getchar。 scanf函数 scanf函数称为格式输入函数，即按用户指定的格式从键盘上把数据输入到指定的变量之中。

一、scanf函数的一般形式

scanf函数是一个标准库函数，它的函数原型在头文件“stdio.h”中，与printf函数相同，C语言也允许在使用scanf函数之前不必包含stdio.h文件。scanf函数的一般形式为：scanf(“格式控制字符串”，地址表列)；其中，格式控制字符串的作用与printf函数相同，但不能显示非格式字符串，也就是不能显示提示字符串。地址表列中给出各变量的地址。地

址是由地址运算符“&”后跟变量名组成的。例如，&a,&b分别表示变量a和变量b 的地址。这个地址就是编译系统在内存中给a,b变量分配的地址。在C语言中，使用了地址这个概念，这是与其它语言不同的。 应该把变量的值和变量的地址这两个不同的概念区别开来。变量的地址是C编译系统分配的，用户不必关心具体的地址是多少。 变量的地址和变量值的关系如下： &a--->a567 a为变量名，567是变量的值，&a是变量a的地址。在赋值表达式中给变量赋值，如： a=567 在赋值号左边是变量名，不能写地址，而scanf函数在本质上也是给变量赋值，但要求写变量的地址，如&a。 这两者在形式上是不同的。&是一个取地址运算符，&a是一个表达式，其功能是求变量的地址。

```
void main(){
int a,b,c;
printf("input a,b,c\n");
scanf("%d%d%d",&a,&b,&c);
printf("a=%d,b=%d,c=%d",a,b,c);
}
```

注意&的用法!

在本例中，由于scanf函数本身不能显示提示串，故先用printf语句在屏幕上输出提示，请用户输入a、b、c的值。执行scanf语句，则退出TC屏幕进入用户屏幕等待用户输入。用户输入7、8、9后按下回车键，此时，系统又将返回TC屏幕。在scanf语句的格式串中由于没有非格式字符在“%d%d%d”之间作输入时的间隔， 因此在输入时要用一个以上的空格或回车键作为每两个输入数之间的间隔。

如： 7 8 9

或

7

8

9

格式字符串

格式字符串的一般形式为： %[*][输入数据宽度][长度]类型 其中有方括号[]的项为任选项。各项的意义如下：

1.类型

表示输入数据的类型，其格式符和意义下表所示。

格式	字符意义
d	输入十进制整数
o	输入八进制整数
x	输入十六进制整数
u	输入无符号十进制整数
f或e	输入实型数(用小数形式或指数形式)
c	输入单个字符
s	输入字符串

2.“*”符

用以表示该输入项读入后不赋予相应的变量，即跳过该输入值。 如 scanf("%d %*d %d",&a,&b);当输入为：1 2 3 时，把1赋予a，2被跳过，3赋予b。

3.宽度

用十进制整数指定输入的宽度(即字符数)。例如： scanf("%5d",&a);
输入：
12345678
只把12345赋予变量a，其余部分被截去。又如： scanf("%4d%4d",&a,&b);
输入：
12345678将把1234赋予a，而把5678赋予b。
4.长度
长度格式符为l和h，l表示输入长整型数据(如%ld) 和双精度浮点数(如%lf)。h表示输入短整型数据。
使用scanf函数还必须注意以下几点：
a. scanf函数中没有精度控制，如： scanf("%5.2f",&a); 是非法的。不能企图用此语句输入小数为2位的实数。
b. scanf中要求给出变量地址，如给出变量名则会出错。如 scanf("%d",a);是非法的，应改为scnaf("%d",&a);才是合法的。
c. 在输入多个数值数据时，若格式控制串中没有非格式字符作输入数据之间的间隔则可用空格，TAB或回车作间隔。C编译在碰到空格，TAB，回车或非法数据(如对“%d”输入“12A”时，A即为非法数据)时即认为该数据结束。
d. 在输入字符数据时，若格式控制串中无非格式字符，则认为所有输入的字符均为有效字符。例如：
scanf("%c%c%c",&a,&b,&c);
输入为：
d e f
则把'd'赋予a, 'f'赋予b,'e'赋予c。只有当输入为：
def
时，才能把'd'赋于a,'e'赋予b,'f'赋予c。如果在格式控制中加入空格作为间隔，如 scanf("%c %c %c",&a,&b,&c);则输入时各数据之间可加空格。

```
void main(){
char a,b;
printf("input character a,b\n");
scanf("%c%c",&a,&b);
printf("%c%c\n",a,b);
}
```

scanf("'C14F14%c%c",&a,&b);
printf("%c%c\n",a,b); 由于scanf函数"%c%c"中没有空格，输入M N，结果输出只有M。而输入改为MN时则可输出MN两字符，见下面的输入运行情况： input character a,b
MN
MN

```
void main(){
char a,b;
printf("input character a,b\n");
scanf("%c %c",&a,&b);
printf("\n%c%c\n",a,b);
}
```

scanf("%c %c",&a,&b); 本例表示scanf格式控制串"%c %c"之间有空格时， 输入的数据之

间可以有空格间隔。e. 如果格式控制串中有非格式字符则输入时也要输入该非格式字符。例如：

scanf("%d,%d,%d",&a,&b,&c)；其中用非格式符“ , ”作间隔符，故输入时应为： 5,6,7

又如： scanf("a=%d,b=%d,c=%d",&a,&b,&c);

则输入应为

a=5,b=6,c=7g. 如输入的数据与输出的类型不一致时，虽然编译能够通过，但结果将不正确。

```
void main(){
int a;
printf("input a number\n");
scanf("%d",&a);
printf("%ld",a);
}
```

由于输入数据类型为整型， 而输出语句的格式串中说明为长整型，因此输出结果和输入数据不符。如改动程序如下：

```
void main(){
long a;
printf("input a long integer\n");
scanf("%ld",&a);
printf("%ld",a);
}
```

运行结果为：

```
input a long integer
1234567890
1234567890
```

当输入数据改为长整型后，输入输出数据相等。

键盘输入函数

getchar函数getchar函数的功能是从键盘上输入一个字符。其一般形式为： getchar()；通常把输入的字符赋予一个字符变量，构成赋值语句，如：

```
char c;
c=getchar();
#include<stdio.h>
void main(){
char c;
printf("input a character\n");
c=getchar();
putchar(c);
}
```

使用getchar函数还应注意几个问题：

1.getchar函数只能接受单个字符，输入数字也按字符处理。输入多于一个字符时，只接收第一个字符。

2.使用本函数前必须包含文件“stdio.h”。

3.在TC屏幕下运行含本函数程序时，将退出TC 屏幕进入用户屏幕等待用户输入。输入完毕再返回TC屏幕。

```
void main(){
```

```
char a,b,c;
printf("input character a,b,c\n");
scanf("%c %c %c",&a,&b,&c);
printf("%d,%d,%d\n%c,%c,%c\n",a,b,c,a-32,b-32,c-32);
}
```

输入三个小写字母，输出其ASCII码和对应的大写字母。

```
void main(){
int a;
long b;
float f;
double d;
char c;
printf("%d,%d,%d,%d,%d",sizeof(a),sizeof(b),sizeof(f),sizeof(d),sizeof(c));}
```

输出各种数据类型的字节长度。

第三节　美丽的循环

首先，给大家看一个程序，这个程序是我自己编的，也没有什么名字，既然在这里提到，就叫他“美丽的循环”吧！:)

```
#include<stdio.h>
#include<math.h>
main()
{ int x,m;
    double y;
  for(y=10;y>=-10;y--) {
     m=1.5*sqrt(100-y*y);
  for(x=1;x<30-m;x++)
        printf(" ");
    printf("*");
  for(;x<30+m;x++)
    printf(" ");
    printf("*\n");
       }
       getch(); }
```

运行后你发现这是一个半径为 10 的圆。怎么样？是不是觉得循环很有趣？经过学习这节，你也许会做出更“美丽”的循环。:)

在 C 语言中，我们常用到的循环有 for,while,do-while 几种，下面我们详细一一介绍这几个循环结构。

while语句

while 语句的一般形式为： while(表达式)语句； 其中表达式是循环条件，语句为循环体。while 语句的语义是：计算表达式的值，当值为真(非 0)时， 执行循环体语句。统计从键盘输入一行字符的个数。

```
#include <stdio.h>
void main(){
int n=0;
printf("input a string:\n");
while(getchar()!='\n') n++;
printf("%d",n);
}
```

本例程序中的循环条件为 getchar()!='\n',其意义是， 只要从键盘输入的字符不是回车就继续循环。循环体 n++完成对输入字符个数计数。从而程序实现了对输入一行字符的字符个数计数。

使用 while 语句应注意以下几点：

1.while 语句中的表达式一般是关系表达或逻辑表达式，只要表达式的值为真(非 0)即可继续循环。

```
void main(){
```

```
int a=0,n;
printf("\n input n: ");
scanf("%d",&n);
while (n--)
printf("%d ",a++*2);
}
```

本例程序将执行 n 次循环，每执行一次，n 值减 1。循环体输出表达式 a++*2 的值。该表达式等效于(a*2；a++)

2.循环体如包括有一个以上的语句，则必须用{}括起来， 组成复合语句。

3.应注意循环条件的选择以避免死循环。

```
void main(){
int a,n=0;
while(a=5)
printf("%d ",n++);
}
```

本例中 while 语句的循环条件为赋值表达式 a=5， 因此该表达式的值永远为真，而循环体中又没有其它中止循环的手段， 因此该循环将无休止地进行下去，形成死循环。4.允许 while 语句的循环体又是 while 语句，从而形成双重循环。

d o－w h i l e语句

do-while 语句的一般形式为：

do

语句；

while(表达式)；

其中语句是循环体，表达式是循环条件。

do-while 语句的语义是：

先执行循环体语句一次， 再判别表达式的值，若为真(非 0)则继续循环，否则终止循环。

do-while 语句和 while 语句的区别在于 do-while 是先执行后判断，因此 do-while 至少要执行一次循环体。而 while 是先判断后执行，如果条件不满足，则一次循环体语句也不执行。

while 语句和 do-while 语句一般都可以相互改写。

```
void main(){
int a=0,n;
printf("\n input n: ");
scanf("%d",&n);
do printf("%d ",a++*2);
while (--n);
}
```

在本例中，循环条件改为--n，否则将多执行一次循环。这是由于先执行后判断而造成的。

对于 do-while 语句还应注意以下几点：

1.在 if 语句，while 语句中， 表达式后面都不能加分号， 而在 do-while 语句的表达式后面则必须加分号。

2.do-while 语句也可以组成多重循环，而且也可以和 while 语句相互嵌套。

3.在 do 和 while 之间的循环体由多个语句组成时，也必须用{}括起来组成一个复合语句。
4.do-while 和 while 语句相互替换时，要注意修改循环控制条件。

f o r语句

for 语句是C语言所提供的功能更强，使用更广泛的一种循环语句。其一般形式为：
for(表达式 1；表达式 2；表达 3)
语句；
表达式 1　通常用来给循环变量赋初值，一般是赋值表达式。也允许在 for 语句外给循环变量赋初值，此时可以省略该表达式。
表达式 2　通常是循环条件，一般为关系表达式或逻辑表达式。
表达式 3　通常可用来修改循环变量的值，一般是赋值语句。
这三个表达式都可以是逗号表达式， 即每个表达式都可由多个表达式组成。三个表达式都是任选项，都可以省略。
一般形式中的“语句”即为循环体语句。for 语句的语义是：
1.首先计算表达式 1 的值。
2.再计算表达式 2 的值，若值为真(非 0)则执行循环体一次， 否则跳出循环。
3.然后再计算表达式 3 的值，转回第 2 步重复执行。在整个 for 循环过程中，表达式 1 只计算一次，表达式 2 和表达式，3 则可能计算多次。循环体可能多次执行，也可能一次都不执行。for 语句的执行过程如图所示。

```
void main(){
int n,s=0;
for(n=1;n<=100;n++)
s=s+n;
printf("s=%d\n",s);
}
```

用 for 语句计算 s=1+2+3+...+99+100

```
    int n,s=0;
for(n=1;n<=100;n++)
s=s+n;
printf("s=%d\n",s);
```

本例 for 语句中的表达式 3 为 n++，实际上也是一种赋值语句，相当于 n=n+1，以改变循环变量的值。

```
void main(){
int a=0,n;
printf("\n input n: ");
scanf("%d",&n);
for(;n>0;a++,n--)
printf("%d ",a*2);
}
```

用 for 语句修改例题。从 0 开始，输出 n 个连续的偶数。

```
int a=0,n;
printf("\n input n: ");
scanf("%d",&n);
```

```
for(;n>0;a++,n--)
printf("%d ",a*2);
```

本例的 for 语句中，表达式 1 已省去，循环变量的初值在 for 语句之前由 scanf 语句取得，表达式 3 是一个逗号表达式，由 a++，n-- 两个表达式组成。每循环一次 a 自增 1，n 自减 1。a 的变化使输出的偶数递增，n 的变化控制循次数。

在使用 for 语句中要注意以下几点:

1.for 语句中的各表达式都可省略，但分号间隔符不能少。如：for(；表达式；表达式)省去了表达式 1。for(表达式；；表达式)省去了表达式 2。

for(表达式；表达式；)省去了表达式 3。for(；；)省去了全部表达式。

2.在循环变量已赋初值时，可省去表达式 1，如例 3.27 即属于这种情形。如省去表达式 2 或表达式 3 则将造成无限循环， 这时应在循环体内设法结束循环。例题即属于此情况。

```
void main(){
int a=0,n;
printf("\n input n: ");
scanf("%d",&n);
for(;n>0;)
{ a++;n--;
printf("%d ",a*2);
}
}
```

本例中省略了表达式 1 和表达式 3，由循环体内的 n--语句进行循环变量 n 的递减，以控制循环次数。

```
void main(){
int a=0,n;
printf("\n input n: ");
scanf("%d",&n);
for(;;){
a++;n--;
printf("%d ",a*2);
if(n==0)break;
}
}
```

本例中 for 语句的表达式全部省去。由循环体中的语句实现循环变量的递减和循环条件的判断。当 n 值为 0 时，由 break 语句中止循环，转去执行 for 以后的程序。在此情况下，for 语句已等效于 while(1)语句。如在循环体中没有相应的控制手段，则造成死循环。

3.循环体可以是空语句。

```
#include"stdio.h"
void main(){
int n=0;
printf("input a string:\n");
for(;getchar()!='\n';n++);
printf("%d",n);
}
```

本例中，省去了 for 语句的表达式 1，表达式 3 也不是用来修改循环变量，而是用作输入字

符的计数。这样， 就把本应在循环体中完成的计数放在表达式中完成了。因此循环体是空语句。应注意的是，空语句后的分号不可少，如缺少此分号，则把后面的 printf 语句当成循环体来执行。反过来说，如循环体不为空语句时， 决不能在表达式的括号后加分号， 这样又会认为循环体是空语句而不能反复执行。这些都是编程中常见的错误，要十分注意。

4.for 语句也可与 while,do-while 语句相互嵌套，构成多重循环。以下形成都合法的嵌套。

```
(1)for(){…
    while()
      {…}
    …
        }
(2)do{
      …
    for()
      {…}
    …
    }while();
(3)while(){
           …
           for()
             {…}
           …
         }
(4)for(){
        …
        for(){
        …
          }
        }
void main(){
int i,j,k;
for(i=1;i<=3;i++)
{ for(j=1;j<=3-i+5;j++)
printf(" ");
for(k=1;k<=2*i-1+5;k++)
{
if(k<=5) printf(" ");
else printf("*");
}
printf("\n");
}
}
```

说道循环，我们由不得不提出另外两兄弟：continue 和 break.

b r e a k语句

break 语句只能用在 switch 语句或循环语句中， 其作用是跳出 switch 语句或跳出本层循环，转去执行后面的程序。由于 break 语句的转移方向是明确的，所以不需要语句标号与之配合。break 语句的一般形式为： break; 上面例题中分别在 switch 语句和 for 语句中使用了 break 语句作为跳转。使用 break 语句可以使循环语句有多个出口，在一些场合下使编程更加灵活、方便。

c o n t i n u e语句

continue 语句只能用在循环体中，其一般格式是：

```
continue;
```

其语义是：结束本次循环，即不再执行循环体中 continue 语句之后的语句，转入下一次循环条件的判断与执行。应注意的是， 本语句只结束本层次的循环，并不跳出循环。

```
void main(){
int n;
for(n=7;n<=100;n++)
{
if (n%7!=0)
continue;
printf("%d ",n);
}
}
```

输出 100 以内能被 7 整除的数。

```
int n;
for(n=7;n<=100;n++)
{
if (n%7!=0)
continue;
printf("%d ",n);
}
```

本例中，对 7～100 的每一个数进行测试，如该数不能被 7 整除，即模运算不为 0，则由 continus 语句转去下一次循环。只有模运算为 0 时，才能执行后面的 printf 语句，输出能被 7 整除的数。

```
#include"stdio.h"
void main(){
char a,b;
printf("input a string:\n");
b=getchar();
while((a=getchar())!='\n'){
if(a==b){
printf("same character\n");
break;
}b=a;
}
}
```

检查输入的一行中有无相邻两字符相同。

```
char a,b;
printf("input a string:\n");
b=getchar();
while((a=getchar())!='\n'){
if(a==b){
printf("same character\n");
break;
}b=a;
}
```

本例程序中，把第一个读入的字符送入 b。然后进入循环，把下一字符读入 a，比较 a,b 是否相等，若相等则输出提示串并中止循环，若不相等则把 a 中的字符赋予 b，输入下一次循环。

输出 100 以内的素数。素数是只能被 1 和本身整除的数。可用穷举法来判断一个数是否是素数。

```
void main(){
int n,i;
for(n=2;n<=100;n++){
for(i=2;i<n;i++)
if(n%i==0) break;
if(i>=n) printf("\t%d",n);
}
} int n,i;
for(n=2;n<=100;n++){
for(i=2;i<n;i++)
if(n%i==0) break;
if(i>=n) printf("\t%d",n);
}
```

本例程序中，第一层循环表示对 1～100 这 100 个数逐个判断是否是素数，共循环 100 次，在第二层循环中则对数 n 用 2～n-1 逐个去除，若某次除尽则跳出该层循环，说明不是素数。如果在所有的数都是未除尽的情况下结束循环，则为素数，此时有 i>=n， 故可经此判断后输出素数。然后转入下一次大循环。实际上，2 以上的所有偶数均不是素数，因此可以使循环变量的步长值改为 2,即每次增加 2,此外只需对数 n 用 2～n 去除就可判断该数是否素数。这样将大大减少循环次数，减少程序运行时间。

```
#include"math.h"
void main(){
int n,i,k;
for(n=2;n<=100;n+=2){
k=sqrt(n);
for(i=2;i<k;i++)
if(n%i==0) break;
if(i>=k) printf("\t%2d",n);
}}
```

第四节　远亲近邻函数

看看下面一个程序，你能不用运行就知道他的功能吗？

```
#include<stdio.h>
 int max(int a,int b);
main()
{ int x,y,c;
   printf("input x and y:\n");
   scanf("%d%d",&x,&y);
   c=max(x,y);
    printf("\nthe max is %d",c);
         getch(); }
int max(int a,int b){
    if(a>b) return a;
    else return b;
    }
```

我们在使用函数时候，几乎都是把函数体放在主函数的 main 外面，用主函数去调用它。把函数称作 main 的“远亲近邻”是不是很形象？正如他自己陈述的一样：（运行一下看看）

```
#include<stdio.h>
 void max();
main()
{
 max();
   getch(); }
   void max(){
printf("我是远亲近邻函数");
    }
```

这一节，我们一同研究这个函数他是怎么个“远”还是“亲”、“近”还只是个“邻”。

在第一章中已经介绍过，C 源程序是由函数组成的。 虽然在前面各章的程序中都只有一个主函数 main()， 但实用程序往往由多个函数组成。函数是 C 源程序的基本模块， 通过对函数模块的调用实现特定的功能。C 语言中的函数相当于其它高级语言的子程序。 C 语言不仅提供了极为丰富的库函数(如 Turbo C，MS C 都提供了三百多个库函数)，还允许用户建立自己定义的函数。用户可把自己的算法编成一个个相对独立的函数模块，然后用调用的方法来使用函数。

可以说 C 程序的全部工作都是由各式各样的函数完成的，所以也把 C 语言称为函数式语言。由于采用了函数模块式的结构， C 语言易于实现结构化程序设计。使程序的层次结构清晰，便于程序的编写、阅读、调试。

在 C 语言中可从不同的角度对函数分类。

1. 从函数定义的角度看，函数可分为库函数和用户定义函数两种。

(1)库函数（在本书附录二给出全部 C 语言标准库函数）

由 C 系统提供，用户无须定义， 也不必在程序中作类型说明，只需在程序前包含有该函数原型的头文件即可在程序中直接调用。在前面各章的例题中反复用到 printf 、scanf 、getchar 、putchar、gets、puts、strcat 等函数均属此类。

(2)用户定义函数

由用户按需要写的函数。对于用户自定义函数， 不仅要在程序中定义函数本身， 而且在主调函数模块中还必须对该被调函数进行类型说明，然后才能使用。

2. C 语言的函数兼有其它语言中的函数和过程两种功能，从这个角度看，又可把函数分为有返回值函数和无返回值函数两种。

(1)有返回值函数

此类函数被调用执行完后将向调用者返回一个执行结果， 称为函数返回值。如数学函数即属于此类函数。 由用户定义的这种要返回函数值的函数，必须在函数定义和函数说明中明确返回值的类型。

(2)无返回值函数

此类函数用于完成某项特定的处理任务， 执行完成后不向调用者返回函数值。这类函数类似于其它语言的过程。 由于函数无须返回值，用户在定义此类函数时可指定它的返回为“空类型”， 空类型的说明符为“void”。

3. 从主调函数和被调函数之间数据传送的角度看又可分为无参函数和有参函数两种。

(1)无参函数

函数定义、函数说明及函数调用中均不带参数。 主调函数和被调函数之间不进行参数传送。此类函数通常用来完成一组指定的功能，可以返回或不返回函数值。

(2)有参函数

也称为带参函数。在函数定义及函数说明时都有参数， 称为形式参数(简称为形参)。在函数调用时也必须给出参数， 称为实际参数(简称为实参)。 进行函数调用时，主调函数将把实参的值传送给形参，供被调函数使用。

4. C 语言提供了极为丰富的库函数， 这些库函数又可从功能角度作以下分类。

(1)字符类型分类函数

用于对字符按 ASCII 码分类：字母，数字，控制字符，分隔符，大小写字母等。

(2)转换函数

用于字符或字符串的转换；在字符量和各类数字量 (整型， 实型等)之间进行转换；在大、小写之间进行转换。

(3)目录路径函数

用于文件目录和路径操作。

(4)诊断函数

用于内部错误检测。

(5)图形函数

用于屏幕管理和各种图形功能。

(6)输入输出函数

用于完成输入输出功能。

(7)接口函数

用于与 DOS，BIOS 和硬件的接口。

(8)字符串函数

用于字符串操作和处理。

(9)内存管理函数

用于内存管理。

(10)数学函数

用于数学函数计算。

(11)日期和时间函数

用于日期，时间转换操作。

(12)进程控制函数

用于进程管理和控制。

(13)其它函数

用于其它各种功能。

以上各类函数不仅数量多，而且有的还需要硬件知识才会使用，因此要想全部掌握则需要一个较长的学习过程。 应首先掌握一些最基本、 最常用的函数，再逐步深入。

还应该指出的是，在C语言中，所有的函数定义，包括主函数 main 在内，都是平行的。也就是说，在一个函数的函数体内， 不能再定义另一个函数， 即不能嵌套定义。但是函数之间允许相互调用，也允许嵌套调用。习惯上把调用者称为主调函数。 函数还可以自己调用自己，称为递归调用。main 函数是主函数，它可以调用其它函数，而不允许被其它函数调用。 因此，C程序的执行总是从 main 函数开始， 完成对其它函数的调用后再返回到 main 函数，最后由 main 函数结束整个程序。一个C源程序必须有，也只能有一个主函数 main。

函数定义的一般形式：

1.无参函数的一般形式

```
类型说明符  函数名()
{
类型说明
语句
}
```

其中类型说明符和函数名称为函数头。 类型说明符指明了本函数的类型，函数的类型实际上是函数返回值的类型。 该类型说明符与第二章介绍的各种说明符相同。 函数名是由用户定义的标识符，函数名后有一个空括号，其中无参数，但括号不可少。{} 中的内容称为函数体。在函数体中也有类型说明， 这是对函数体内部所用到的变量的类型说明。在很多情况下都不要求无参函数有返回值， 此时函数类型符可以写为 void。

我们可以改为一个函数定义：

```
void Hello()
{
printf ("Hello,world \n");
}
```

这里，只把 main 改为 Hello 作为函数名，其余不变。Hello 函数是一个无参函数，当被其它函数调用时，输出 Hello world 字符串。

2.有参函数的一般形式

```
类型说明符  函数名(形式参数表)
型式参数类型说明
{
类型说明
```

语句
}

有参函数比无参函数多了两个内容，其一是形式参数表， 其二是形式参数类型说明。在形参表中给出的参数称为形式参数， 它们可以是各种类型的变量， 各参数之间用逗号间隔。在进行函数调用时，主调函数将赋予这些形式参数实际的值。 形参既然是变量，当然必须给以类型说明。例如，定义一个函数， 用于求两个数中的大数，可写为：

```
int max(a,b)
int a,b;
{
if (a>b) return a;
else return b;
}
```

第一行说明 max 函数是一个整型函数，其返回的函数值是一个整数。形参为 a,b。第二行说明 a,b 均为整型量。 a,b 的具体值是由主调函数在调用时传送过来的。在{}中的函数体内， 除形参外没有使用其它变量，因此只有语句而没有变量类型说明。 上边这种定义方法称为“传统格式”。这种格式不易于编译系统检查，从而会引起一些非常细微而且难于跟踪的错误。ANSI C 的新标准中把对形参的类型说明合并到形参表中，称为“现代格式”。

例如 max 函数用现代格式可定义为：

```
int max(int a,int b)
{
if(a>b) return a;
else return b;
}
```

现代格式在函数定义和函数说明(后面将要介绍)时， 给出了形式参数及其类型，在编译时易于对它们进行查错，从而保证了函数说明和定义的一致性。例 1.3 即采用了这种现代格式。在 max 函数体中的 return 语句是把 a(或 b)的值作为函数的值返回给主调函数。有返回值函数中至少应有一个 return 语句。 在 C 程序中，一个函数的定义可以放在任意位置， 既可放在主函数 main 之前，也可放在 main 之后。例如例 1.3 中定义了一个 max 函数，其位置在 main 之后， 也可以把它放在 main 之前。

修改后的程序如下所示。

```
int max(int a,int b)
{
if(a>b)return a;
else return b;
}
void main()
{
int max(int a,int b);
int x,y,z;
printf("input two numbers:\n");
scanf("%d%d",&x,&y);
z=max(x,y);
printf("maxmum=%d",z);
}
```

现在我们可以从函数定义、 函数说明及函数调用的角度来分析整个程序，从中进一步了解函数的各种特点。程序的第 1 行至第 5 行为 max 函数定义。进入主函数后，因为准备调用 max 函数，故先对 max 函数进行说明(程序第 8 行)。函数定义和函数说明并不是一回事，在后面还要专门讨论。 可以看出函数说明与函数定义中的函数头部分相同，但是末尾要加分号。程序第 12 行为调用 max 函数，并把 x,y 中的值传送给 max 的形参 a,b。max 函数执行的
结果 (a 或 b)将返回给变量 z。最后由主函数输出 z 的值。

函数调用的一般形式前面已经说过，在程序中是通过对函数的调用来执行函数体的，其过程与其它语言的子程序调用相似。C 语言中， 函数调用的一般形式为：

函数名(实际参数表) 对无参函数调用时则无实际参数表。 实际参数表中的参数可以是常数，变量或其它构造类型数据及表达式。 各实参之间用逗号分隔。'Next of Page 在 C 语言中，可以用以下几种方式调用函数：

1.函数表达式

函数作表达式中的一项出现在表达式中，以函数返回值参与表达式的运算。这种方式要求函数是有返回值的。例如：z=max(x,y)是一个赋值表达式，把 max 的返回值赋予变量 z。'Next of Page

2.函数语句

函数调用的一般形式加上分号即构成函数语句。例如： printf ("%D",a);scanf ("%d",&b);都是以函数语句的方式调用函数。

3.函数实参

函数作为另一个函数调用的实际参数出现。 这种情况是把该函数的返回值作为实参进行传送，因此要求该函数必须是有返回值的。例如： printf("%d",max(x,y)); 即是把 max 调用的返回值又作为 printf 函数的实参来使用的。在函数调用中还应该注意的一个问题是求值顺序的问题。所谓求值顺序是指对实参表中各量是自左至右使用呢，还是自右至左使用。 对此， 各系统的规定不一定相同。在 3.1.3 节介绍 printf 函数时已提到过，这里从函数调用的角度再强调一下。

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

如按照从右至左的顺序求值。上例的运行结果应为：

8
7
7
8

如对 printf 语句中的++i，--i，i++，i--从左至右求值，结果应为：

9
8
8
9

应特别注意的是，无论是从左至右求值， 还是自右至左求值，其输出顺序都是不变的， 即输出顺序总是和实参表中实参的顺序相同。由于 Turbo C 现定是自右至左求值，所以结果为 8，7，7，8。上述问题如还不理解，上机一试就明白了。函数的参数和函数的值

一、函数的参数

前面已经介绍过，函数的参数分为形参和实参两种。 在本小节中，进一步介绍形参、实参的特点和两者的关系。 形参出现在函数定义中，在整个函数体内都可以使用， 离开该函数则不能使用。实参出现在主调函数中，进入被调函数后，实参变量也不能使用。 形参和实参的功能是作数据传送。发生函数调用时， 主调函数把实参的值传送给被调函数的形参从而实现主调函数向被调函数的数据传送。

函数的形参和实参具有以下特点：

1.形参变量只有在被调用时才分配内存单元，在调用结束时， 即刻释放所分配的内存单元。因此，形参只有在函数内部有效。 函数调用结束返回主调函数后则不能再使用该形参变量。

2.实参可以是常量、变量、表达式、函数等， 无论实参是何种类型的量，在进行函数调用时，它们都必须具有确定的值， 以便把这些值传送给形参。 因此应预先用赋值，输入等办法使实参获得确定值。

3.实参和形参在数量上，类型上，顺序上应严格一致， 否则会发生“类型不匹配”的错误。

4.函数调用中发生的数据传送是单向的。 即只能把实参的值传送给形参，而不能把形参的值反向地传送给实参。 因此在函数调用过程中，形参的值发生改变，而实参中的值不会变化。下例可以说明这个问题。

```
void main()
{
int n;
printf("input number\n");
scanf("%d",&n);
s(n);
printf("n=%d\n",n);
}
int s(int n)
{
int i;
for(i=n-1;i>=1;i--)
n=n+i;
printf("n=%d\n",n);
}
```

本程序中定义了一个函数 s，该函数的功能是求∑ni=1i 的值。在主函数中输入 n 值，并作为实参，在调用时传送给 s 函数的形参量 n(注意，本例的形参变量和实参变量的标识符都为 n， 但这是两个不同的量，各自的作用域不同)。 在主函数中用 printf 语句输出一次 n 值，这个 n 值是实参 n 的值。在函数 s 中也用 printf 语句输出了一次 n 值，这个 n 值是形参最后取得的 n 值 0。从运行情况看，输入 n 值为 100。即实参 n 的值为 100。把此值传给函数 s 时，形参 n 的初值也为 100，在执行函数过程中，形参 n 的值变为 5050。 返回主函数之后，输出实参 n 的值仍为 100。可见实参的值不随形参的变化而变化。

二、函数的值

函数的值是指函数被调用之后， 执行函数体中的程序段所取得的并返回给主调函数的值。如调用正弦函数取得正弦值，调用 max 函数取得的最大数等。对函数的值(或称函数返回值)有以下一些说明：

1. 函数的值只能通过 return 语句返回主调函数。return 语句的一般形式为：
return 表达式;
或者为：
return (表达式);
该语句的功能是计算表达式的值，并返回给主调函数。 在函数中允许有多个 return 语句，但每次调用只能有一个 return 语句被执行， 因此只能返回一个函数值。

2. 函数值的类型和函数定义中函数的类型应保持一致。 如果两者不一致，则以函数类型为准，自动进行类型转换。

3. 如函数值为整型，在函数定义时可以省去类型说明。

4. 不返回函数值的函数，可以明确定义为“空类型”， 类型说明符为“void”。如前例中函数 s 并不向主函数返函数值，因此可定义为：
```
void s(int n)
{ ……
}
```

一旦函数被定义为空类型后， 就不能在主调函数中使用被调函数的函数值了。例如，在定义 s 为空类型后，在主函数中写下述语句 sum=s(n); 就是错误的。为了使程序有良好的可读性并减少出错， 凡不要求返回值的函数都应定义为空类型。函数说明在主调函数中调用某函数之前应对该被调函数进行说明， 这与使用变量之前要先进行变量说明是一样的。 在主调函数中对被调函数作说明的目的是使编译系统知道被调函数返回值的类型， 以便在主调函数中按此种类型对返回值作相应的处理。对被调函数的说明也有两种格式，一种为传统格式，其一般格式为：类型说明符 被调函数名()； 这种格式只给出函数返回值的类型，被调函数名及一个空括号。

这种格式由于在括号中没有任何参数信息， 因此不便于编译系统进行错误检查，易于发生错误。另一种为现代格式，其一般形式为：
类型说明符 被调函数名(类型 形参，类型 形参…);
或为：
类型说明符 被调函数名(类型，类型…);
现代格式的括号内给出了形参的类型和形参名， 或只给出形参类型。这便于编译系统进行检错，以防止可能出现的错误。函数中对 max 函数的说明若用传统格式可写为：
```
int max();
```
用现代格式可写为：
```
int max(int a,int b);
```
或写为:
```
int max(int,int);
```
C 语言中又规定在以下几种情况时可以省去主调函数中对被调函数的函数说明。
1. 如果被调函数的返回值是整型或字符型时， 可以不对被调函数作说明，而直接调用。这时系

统将自动对被调函数返回值按整型处理。主函数中未对函数 s 作说明而直接调用即属此种情形。

2. 当被调函数的函数定义出现在主调函数之前时， 在主调函数中也可以不对被调函数再作说明而直接调用。函数 max 的定义放在 main 函数之前，因此可在 main 函数中省去对 max 函数的函数说明 int max(int a,int b)。

3. 如在所有函数定义之前， 在函数外预先说明了各个函数的类型，则在以后的各主调函数中，可不再对被调函数作说明。例如：

```
char str(int a);
float f(float b);
main()
{
……
}
char str(int a)
{
……
}
float f(float b)
{
……
}
```

其中第一，二行对 str 函数和 f 函数预先作了说明。 因此在以后各函数中无须对 str 和 f 函数再作说明就可直接调用。

4. 对库函数的调用不需要再作说明， 但必须把该函数的头文件用 include 命令包含在源文件前部。数组作为函数参数数组可以作为函数的参数使用，进行数据传送。 数组用作函数参数有两种形式，一种是把数组元素(下标变量)作为实参使用； 另一种是把数组名作为函数的形参和实参使用。一、数组元素作函数实参数组元素就是下标变量，它与普通变量并无区别。 因此它作为函数实参使用与普通变量是完全相同的，在发生函数调用时， 把作为实参的数组元素的值传送给形参，实现单向的值传送。

判别一个整数数组中各元素的值，若大于 0 则输出该值，若小于等于 0 则输出 0 值。编程如下：

```
void nzp(int v)
{
if(v>0)
printf("%d ",v);
else
printf("%d ",0);
}
main()
{
int a[5],i;
printf("input 5 numbers\n");
for(i=0;i<5;i++)
```

```
{
scanf("%d",&a[i]);
nzp(a[i]);
}
}void nzp(int v)
{ ……
}
main()
{
int a[5],i;
printf("input 5 numbers\n");
for(i=0;i<5;i++)
{ scanf("%d",&a[i]);
nzp(a[i]);
}
}
```

本程序中首先定义一个无返回值函数 nzp，并说明其形参 v 为整型变量。在函数体中根据 v 值输出相应的结果。在 main 函数中用一个 for 语句输入数组各元素， 每输入一个就以该元素作实参调用一次 nzp 函数，即把 a[i]的值传送给形参 v，供 nzp 函数使用。

二、数组名作为函数参数

用数组名作函数参数与用数组元素作实参有几点不同：

1. 用数组元素作实参时，只要数组类型和函数的形参变量的类型一致，那么作为下标变量的数组元素的类型也和函数形参变量的类型是一致的。因此，并不要求函数的形参也是下标变量。换句话说，对数组元素的处理是按普通变量对待的。用数组名作函数参数时， 则要求形参和相对应的实参都必须是类型相同的数组，都必须有明确的数组说明。当形参和实参二者不一致时，即会发生错误。

2. 在普通变量或下标变量作函数参数时，形参变量和实参变量是由编译系统分配的两个不同的内存单元。在函数调用时发生的值传送是把实参变量的值赋予形参变量。在用数组名作函数参数时，不是进行值的传送，即不是把实参数组的每一个元素的值都赋予形参数组的各个元素。因为实际上形参数组并不存在，编译系统不为形参数组分配内存。那么，数据的传送是如何实现的呢?在第四章中我们曾介绍过，数组名就是数组的首地址。因此在数组名作函数参数时所进行的传送只是地址的传送， 也就是说把实参数组的首地址赋予形参数组名。形参数组名取得该首地址之后，也就等于有了实在的数组。实际上是形参数组和实参数组为同一数组，共同拥有一段内存空间。图 5.1 说明了这种情形。图中设 a 为实参数组，类型为整型。a 占有以 2000 为首地址的一块内存区。b 为形参数组名。当发生函数调用时，进行地址传送， 把实参数 组 a 的首地址传送给形参数组名 b，于是 b 也取得该地址 2000。 于是 a，b 两数组共同占有以 2000 为首地址的一段连续内存单元。从图中还可以看出 a 和 b 下标相同的元素实际上也占相同的两个内存单元(整型数组每个元素占二字节)。例如 a[0]和 b[0]都占用 2000 和 2001 单元，当然 a[0]等于 b[0]。类推则有 a[i]等于 b[i]。

数组 a 中存放了一个学生 5 门课程的成绩，求平均成绩。

```
float aver(float a[5])
```

```
{
int i;
float av,s=a[0];
for(i=1;i<5;i++)
s=s+a[i];
av=s/5;
return av;
}
void main()
{
float sco[5],av;
int i;
printf("\ninput 5 scores:\n");
for(i=0;i<5;i++)
scanf("%f",&sco[i]);
av=aver(sco);
printf("average score is %5.2f",av);
}
float aver(float a[5])
{ ……
}
void main()
{
……
for(i=0;i<5;i++)
scanf("%f",&sco[i]);
av=aver(sco);
……
}
```

本程序首先定义了一个实型函数 aver，有一个形参为实型数组 a，长度为 5。在函数 aver 中，把各元素值相加求出平均值，返回给主函数。主函数 main 中首先完成数组 sco 的输入，然后以 sco 作为实参调用 aver 函数，函数返回值送 av，最后输出 av 值。 从运行情况可以看出，程序实现了所要求的功能

3. 前面已经讨论过，在变量作函数参数时，所进行的值传送是单向的。即只能从实参传向形参，不能从形参传回实参。形参的初值和实参相同， 而形参的值发生改变后，实参并不变化， 两者的终值是不同的。而当用数组名作函数参数时，情况则不同。 由于实际上形参和实参为同一数组， 因此当形参数组发生变化时，实参数组也随之变化。 当然这种情况不能理解为发生了“双向”的值传递。但从实际情况来看，调用函数之后实参数组的值将由于形参数组值的变化而变化。改用数组名作函数参数。

```
void nzp(int a[5])
{
int i;
printf("\nvalues of array a are:\n");
```

```
for(i=0;i<5;i++)
{
if(a[i]<0) a[i]=0;
printf("%d ",a[i]);
}
}
main()
{
int b[5],i;
printf("\ninput 5 numbers:\n");
for(i=0;i<5;i++)
scanf("%d",&b[i]);
printf("initial values of array b are:\n");
for(i=0;i<5;i++)
printf("%d ",b[i]);
nzp(b);
printf("\nlast values of array b are:\n");
for(i=0;i<5;i++)
printf("%d ",b[i]);
}
void nzp(int a[5])
{ ……
}
main()
{
int b[5],i;
……
nzp(b);
……
}
```

本程序中函数 nzp 的形参为整数组 a，长度为 5。 主函数中实参数组 b 也为整型，长度也为 5。在主函数中首先输入数组 b 的值，然后输出数组 b 的初始值。 然后以数组名 b 为实参调用 nzp 函数。在 nzp 中，按要求把负值单元清 0，并输出形参数组 a 的值。 返回主函数之后，再次输出数组 b 的值。从运行结果可以看出，数组 b 的初值和终值是不同的，数组 b 的终值和数组 a 是相同的。这说明实参形参为同一数组，它们的值同时得以改变。 用数组名作为函数参数时还应注意以下几点：

a. 形参数组和实参数组的类型必须一致，否则将引起错误。

b. 形参数组和实参数组的长度可以不相同，因为在调用时，只传送首地址而不检查形参数组的长度。当形参数组的长度与实参数组不一致时，虽不至于出现语法错误(编译能通过)，但程序执行结果将与实际不符，这是应予以注意的。如修改如下：

```
void nzp(int a[8])
{
int i;
printf("\nvalues of array aare:\n");
```

```
for(i=0;i<8;i++)
{
if(a[i]<0)a[i]=0;
printf("%d",a[i]);
}
}
main()
{
int b[5],i;
printf("\ninput 5 numbers:\n");
for(i=0;i<5;i++)
scanf("%d",&b[i]);
printf("initial values of array b are:\n");
for(i=0;i<5;i++)
printf("%d",b[i]);
nzp(b);
printf("\nlast values of array b are:\n");
for(i=0;i<5;i++)
printf("%d",b[i]);
}
```

nzp 函数的形参数组长度改为 8，函数体中，for 语句的循环条件也改为 i<8。因此，形参数组 a 和实参数组 b 的长度不一致。编译能够通过，但从结果看，数组 a 的元素 a[5]，a[6]，a[7]显然是无意义的。c. 在函数形参表中，允许不给出形参数组的长度，或用一个变量来表示数组元素的个数。

例如：可以写为：

```
void nzp(int a[])
```

或写为

```
void nzp(int a[]，int n)
```

其中形参数组 a 没有给出长度，而由 n 值动态地表示数组的长度。n 的值由主调函数的实参进行传送。

```
void nzp(int a[],int n)
{
int i;
printf("\nvalues of array a are:\n");
for(i=0;i<n;i++)
{
if(a[i]<0) a[i]=0;
printf("%d ",a[i]);
}
}
main()
{
int b[5],i;
printf("\ninput 5 numbers:\n");
```

```
for(i=0;i<5;i++)
scanf("%d",&b[i]);
printf("initial values of array b are:\n");
for(i=0;i<5;i++)
printf("%d ",b[i]);
nzp(b,5);
printf("\nlast values of array b are:\n");
for(i=0;i<5;i++)
printf("%d ",b[i]);
}
void nzp(int a[],int n)
{ ……
}
main()
{
……
nzp(b,5);
……
}
```

本程序 nzp 函数形参数组 a 没有给出长度，由 n 动态确定该长度。在 main 函数中，函数调用语句为 nzp(b，5)，其中实参 5 将赋予形参 n 作为形参数组的长度。

d. 多维数组也可以作为函数的参数。 在函数定义时对形参数组可以指定每一维的长度，也可省去第一维的长度。因此，以下写法都是合法的。

```
int MA(int a[3][10])
```

或

```
int MA(int a[][10])
```

函数的嵌套调用

C 语言中不允许作嵌套的函数定义。因此各函数之间是平行的，不存在上一级函数和下一级函数的问题。 但是 C 语言允许在一个函数的定义中出现对另一个函数的调用。 这样就出现了函数的嵌套调用。即在被调函数中又调用其它函数。 这与其它语言的子程序嵌套的情形是类似的。其执行过程是：执行 main 函数中调用 a 函数的语句时，即转去执行 a 函数，在 a 函数中调用 b 函数时，又转去执行 b 函数，b 函数执行完毕返回 a 函数的断点继续执行，a 函数执行完毕返回 main 函数的断点继续执行。

题：计算 s=2　2!+3　2!

本题可编写两个函数，一个是用来计算平方值的函数 f1， 另一个是用来计算阶乘值的函数 f2。主函数先调 f1 计算出平方值， 再在 f1 中以平方值为实参，调用 f2 计算其阶乘值，然后返回 f1，再返回主函数，在循环程序中计算累加和。

```
long f1(int p)
{
int k;
long r;
long f2(int);
```

```
k=p*p;
r=f2(k);
return r;
}
long f2(int q)
{
long c=1;
int i;
for(i=1;i<=q;i++)
c=c*i;
return c;
}
main()
{
int i;
long s=0;
for (i=2;i<=3;i++)
s=s+f1(i);
printf("\ns=%ld\n",s);
}
long f1(int p)
{
……
long f2(int);
r=f2(k);
……
}
long f2(int q)
{
……
}
main()
{ ……
s=s+f1(i);
……
}
```

在程序中，函数 f1 和 f2 均为长整型，都在主函数之前定义， 故不必再在主函数中对 f1 和 f2 加以说明。在主程序中， 执行循环程序依次把 i 值作为实参调用函数 f1 求 i 2 值。在 f1 中又发生对函数 f2 的调用，这时是把 i 2 的值作为实参去调 f2，在 f2 中完成求 i 2! 的计算。f2 执行完毕把 C 值(即 i 2!)返回给 f1，再由 f1 返回主函数实现累加。至此，由函数的嵌套调用实现了题目的要求。 由于数值很大， 所以函数和一些变量的类型都说明为长整型，否则会造成计算错误。

函数的递归调用

一个函数在它的函数体内调用它自身称为递归调用。 这种函数称为递归函数。C 语言允许函数的递归调用。在递归调用中， 主调函数又是被调函数。执行递归函数将反复调用其自身。 每调用一次就进入新的一层。例如有函数 f 如下：

```
int f (int x)
{
int y;
z=f(y);
return z;
}
```

这个函数是一个递归函数。 但是运行该函数将无休止地调用其自身，这当然是不正确的。为了防止递归调用无终止地进行， 必须在函数内有终止递归调用的手段。常用的办法是加条件判断， 满足某种条件后就不再作递归调用，然后逐层返回。 下面举例说明递归调用的执行过程。

[例 5.9]用递归法计算 n!用递归法计算 n!可用下述公式表示：

n!=1 (n=0,1)

n×(n-1)! (n>1)

按公式可编程如下：

```
long ff(int n)
{
long f;
if(n<0) printf("n<0,input error");
else if(n==0||n==1) f=1;
else f=ff(n-1)*n;
return(f);
}
main()
{
int n;
long y;
printf("\ninput a inteager number:\n");
scanf("%d",&n);
y=ff(n);
printf("%d!=%ld",n,y);
}
long ff(int n)
{ ……
else f=ff(n-1)*n;
……
}
main()
{ ……
y=ff(n);
……
}
```

程序中给出的函数 ff 是一个递归函数。主函数调用 ff 后即进入函数 ff 执行，如果 n<0,n==0 或 n=1 时都将结束函数的执行，否则就递归调用 ff 函数自身。由于每次递归调用的实参为 n-1，即把 n-1 的值赋予形参 n，最后当 n-1 的值为 1 时再作递归调用，形参 n 的值也为 1，将使递归终止。然后可逐层退回。下面我们再举例说明该过程。 设执行本程序时输入为 5， 即求 5!。在主函数中的调用语句即为 y=ff(5)，进入 ff 函数后，由于 n=5,不等于 0 或 1，故应执行 f=ff(n-1)*n,即 f=ff(5-1)*5。该语句对 ff 作递归调用即 ff(4)。 逐次递归展开如图 5.3 所示。进行四次递归调用后，ff 函数形参取得的值变为 1，故不再继续递归调用而开始逐层返回主调函数。ff(1)的函数返回值为 1，ff(2)的返回值为 1*2=2，ff(3)的返回值为 2*3=6，ff(4) 的返
回值为 6*4=24，最后返回值 ff(5)为 24*5=120。

也可以不用递归的方法来完成。如可以用递推法，即从 1 开始乘以 2，再乘以 3…直到 n。递推法比递归法更容易理解和实现。但是有些问题则只能用递归算法才能实现。典型的问题是 Hanoi 塔问题。

Hanoi 塔问题

一块板上有三根针，A，B，C。A 针上套有 64 个大小不等的圆盘， 大的在下，小的在上。如图 5.4 所示。要把这 64 个圆盘从 A 针移动 C 针上，每次只能移动一个圆盘，移动可以借助 B 针进行。但在任何时候，任何针上的圆盘都必须保持大盘在下，小盘在上。求移动的步骤。

本题算法分析如下，设 A 上有 n 个盘子。

如果 n=1，则将圆盘从 A 直接移动到 C。

如果 n=2，则：

1.将 A 上的 n-1(等于 1)个圆盘移到 B 上；

2.再将 A 上的一个圆盘移到 C 上；

3.最后将 B 上的 n-1(等于 1)个圆盘移到 C 上。

如果 n=3，则：

A. 将 A 上的 n-1(等于 2，令其为 n`)个圆盘移到 B(借助于 C)，

步骤如下：

(1)将 A 上的 n`-1(等于 1)个圆盘移到 C 上。

(2)将 A 上的一个圆盘移到 B

(3)将 C 上的 n`-1(等于 1)个圆盘移到 B，

B. 将 A 上的一个圆盘移到 C，

C. 将 B 上的 n-1(等于 2，令其为 n`)个圆盘移到 C(借助 A)，

步骤如下：

(1)将 B 上的 n`-1(等于 1)个圆盘移到 A，

(2)将 B 上的一个盘子移到 C，

(3)将 A 上的 n`-1(等于 1)个圆盘移到 C，

到此，完成了三个圆盘的移动过程。

从上面分析可以看出，当 n 大于等于 2 时， 移动的过程可分解为

三个步骤：

第一步 把 A 上的 n-1 个圆盘移到 B 上；

第二步 把 A 上的一个圆盘移到 C 上；

第三步 把 B 上的 n-1 个圆盘移到 C 上；其中第一步和第三步是类同的。

当 n=3 时，第一步和第三步又分解为类同的三步，即把 n`-1 个圆盘从一个针移到另一个针上，这里的 n`=n-1。 显然这是一个递归过程，据此算法可编程如下：

```
move(int n,int x,int y,int z)
{
if(n==1)
printf("%c-->%c\n",x,z);
else
{
move(n-1,x,z,y);
printf("%c-->%c\n",x,z);
move(n-1,y,x,z);
}
}
main()
{
int h;
printf("\ninput number:\n");
scanf("%d",&h);
printf("the step to moving %2d diskes:\n",h);
move(h,'a','b','c');
}
move(int n,int x,int y,int z)
{
if(n==1)
printf("%-->%c\n",x,z);
else
{
move(n-1,x,z,y);
printf("%c-->%c\n",x,z);
move(n-1,y,x,z);
}
}
main()
{ ……
move(h,'a','b','c');
}
```

从程序中可以看出,move 函数是一个递归函数，它有四个形参 n,x,y,z。n 表示圆盘数，x,y,z 分别表示三根针。move 函数的功能是把 x 上的 n 个圆盘移动到 z 上。当 n==1 时，直接把 x 上的圆盘移至 z 上，输出 x→z。如 n!=1 则分为三步：递归调用 move 函数，把 n-1 个圆盘从 x 移到 y；输出 x→z；递归调用 move 函数，把 n-1 个圆盘从 y 移到 z。在递归调用过程中 n=n-1，故 n 的值逐次递减，最后 n=1 时，终止递归，逐层返回。当 n=4 时程序运行的结果为

```
input number:
4
the step to moving 4 diskes:
a→b
a→c
```

b→c
a→b
c→a
c→b
a→b
a→c
b→c
b→a
c→a
b→c
a→b
a→c
b→c

变量的作用域

在讨论函数的形参变量时曾经提到， 形参变量只在被调用期间才分配内存单元，调用结束立即释放。 这一点表明形参变量只有在函数内才是有效的， 离开该函数就不能再使用了。这种变量有效性的范围称变量的作用域。不仅对于形参变量， C 语言中所有的量都有自己的作用域。变量说明的方式不同，其作用域也不同。 C 语言中的变量，按作用域范围可分为两种， 即局部变量和全局变量。

一、局部变量

局部变量也称为内部变量。局部变量是在函数内作定义说明的。其作用域仅限于函数内， 离开该函数后再使用这种变量是非法的。
例如：

```
int f1(int a) /*函数 f1*/
{
int b,c;
……
}a,b,c 作用域
int f2(int x) /*函数 f2*/
{
int y,z;
}x,y,z 作用域
main()
{
int m,n;
}
```

m,n 作用域 在函数 f1 内定义了三个变量，a 为形参，b,c 为一般变量。在 f1 的范围内 a,b,c 有效，或者说 a,b,c 变量的作用域限于 f1 内。同理，x,y,z 的作用域限于 f2 内。 m,n 的作用域限于 main 函数内。关于局部变量的作用域还要说明以下几点：

1. 主函数中定义的变量也只能在主函数中使用，不能在其它函数中使用。同时，主函数中也不能使用其它函数中定义的变量。因为主函数也是一个函数，它与其它函数是平行关系。这一点是与其它语言不同的，应予以注意。

2. 形参变量是属于被调函数的局部变量，实参变量是属于主调函数的局部变量。

3. 允许在不同的函数中使用相同的变量名，它们代表不同的对象，分配不同的单元，互不干扰，也不会发生混淆。形参和实参的变量名都为 n，是完全允许的。4. 在复合语句中也可定义变量，其作用域只在复合语句范围内。例如：

```
main()
{
int s,a;
……
{
int b;
s=a+b;
……b 作用域
}
……s,a 作用域
}[例 5.11]main()
{
int i=2,j=3,k;
k=i+j;
{
int k=8;
if(i==3) printf("%d\n",k);
}
printf("%d\n%d\n",i,k);
}
main()
{
int i=2,j=3,k;
k=i+j;
{
int k=8;
if(i=3) printf("%d\n",k);
}
printf("%d\n%d\n",i,k);
}
```

本程序在 main 中定义了 i,j,k 三个变量，其中 k 未赋初值。 而在复合语句内又定义了一个变量 k，并赋初值为 8。应该注意这两个 k 不是同一个变量。在复合语句外由 main 定义的 k 起作用，而在复合语句内则由在复合语句内定义的 k 起作用。因此程序第 4 行的 k 为 main 所定义，其值应为 5。第 7 行输出 k 值，该行在复合语句内，由复合语句内定义的 k 起作用，其初值为 8，故输出值为 8，第 9 行输出 i，k 值。i 是在整个程序中有效的，第 7 行对 i 赋值为 3，故以输出

也为 3。而第 9 行已在复合语句之外，输出的 k 应为 main 所定义的 k，此 k 值由第 4 行已获得为 5，故输出也为 5。

二、全局变量

全局变量也称为外部变量，它是在函数外部定义的变量。 它不属于哪一个函数，它属于一个源程序文件。其作用域是整个源程序。在函数中使用全局变量，一般应作全局变量说明。 只有在函数内经过说明的全局变量才能使用。全局变量的说明符为 extern。 但在一个函数之前定义的全局变量，在该函数内使用可不再加以说明。 例如：

```
int a,b; /*外部变量*/
void f1() /*函数 f1*/
{
……
}
float x,y; /*外部变量*/
int fz() /*函数 fz*/
{
……
}
main() /*主函数*/
{
……
}/*全局变量 x,y 作用域 全局变量 a,b 作用域*/
```

从上例可以看出 a、b、x、y 都是在函数外部定义的外部变量，都是全局变量。但 x,y 定义在函数 f1 之后，而在 f1 内又无对 x,y 的说明，所以它们在 f1 内无效。 a,b 定义在源程序最前面，因此在 f1,f2 及 main 内不加说明也可使用。

输入正方体的长宽高 l,w,h。求体积及三个面 x*y,x*z,y*z 的面积。

```
int s1,s2,s3;
int vs( int a,int b,int c)
{
int v;
v=a*b*c;
s1=a*b;
s2=b*c;
s3=a*c;
return v;
}
main()
{
int v,l,w,h;
printf("\ninput length,width and height\n");
scanf("%d%d%d",&l,&w,&h);
v=vs(l,w,h);
```

```
printf("v=%d s1=%d s2=%d s3=%d\n",v,s1,s2,s3);
}
```

本程序中定义了三个外部变量 s1,s2,s3， 用来存放三个面积，其作用域为整个程序。函数 vs 用来求正方体体积和三个面积， 函数的返回值为体积 v。由主函数完成长宽高的输入及结果输出。由于 C 语言规定函数返回值只有一个， 当需要增加函数的返回数据时，用外部变量是一种很好的方式。本例中，如不使用外部变量， 在主函数中就不可能取得 v,s1,s2,s3 四个值。而采用了外部变量， 在函数 vs 中求得的 s1,s2,s3 值在 main 中仍然有效。因此外部变量是实现函数之间数据通讯的有效手段。对于全局变量还有以下几点说明：

1. 对于局部变量的定义和说明，可以不加区分。而对于外部变量则不然，外部变量的定义和外部变量的说明并不是一回事。外部变量定义必须在所有的函数之外，且只能定义一次。其一般形式为： [extern] 类型说明符 变量名，变量名… 其中方括号内的 extern 可以省去不写。

例如： int a,b;

等效于：

extern int a,b;

而外部变量说明出现在要使用该外部变量的各个函数内， 在整个程序内，可能出现多次，外部变量说明的一般形式为： extern 类型说明符 变量名，变量名，…； 外部变量在定义时就已分配了内存单元， 外部变量定义可作初始赋值，外部变量说明不能再赋初始值， 只是表明在函数内要使用某外部变量。

2. 外部变量可加强函数模块之间的数据联系， 但是又使函数要依赖这些变量，因而使得函数的独立性降低。从模块化程序设计的观点来看这是不利的， 因此在不必要时尽量不要使用全局变量。

3. 在同一源文件中，允许全局变量和局部变量同名。在局部变量的作用域内，全局变量不起作用。

```
int vs(int l,int w)
{
extern int h;
int v;
v=l*w*h;
return v;
}
main()
{
extern int w,h;
int l=5;
printf("v=%d",vs(l,w));
}
int l=3,w=4,h=5;
```

本例程序中，外部变量在最后定义， 因此在前面函数中对要用的外部变量必须进行说明。外部变量 l，w 和 vs 函数的形参 l，w 同名。外部变量都作了初始赋值，mian 函数中也对 l 作了初始化赋值。执行程序时，在 printf 语句中调用 vs 函数，实参 l 的值应为 main 中定义的 l 值，等于 5，外部变量 l 在 main 内不起作用；实参 w 的值为外部变量 w 的值为 4，进入 vs 后这两个

值传送给形参 l，wvs 函数中使用的 h 为外部变量，其值为 5，因此 v 的计算结果为 100，返回主函数后输出。变量的存储类型各种变量的作用域不同， 就其本质来说是因变量的存储类型相同。所谓存储类型是指变量占用内存空间的方式， 也称为存储方式。

变量的存储方式可分为“静态存储”和“动态存储”两种。

静态存储变量通常是在变量定义时就分定存储单元并一直保持不变， 直至整个程序结束。前面介绍的全局变量即属于此类存储方式。动态存储变量是在程序执行过程中，使用它时才分配存储单元， 使用完毕立即释放。 典型的例子是函数的形式参数，在函数定义时并不给形参分配存储单元，只是在函数被调用时，才予以分配， 调用函数完毕立即释放。如果一个函数被多次调用，则反复地分配、 释放形参变量的存储单元。从以上分析可知， 静态存储变量是一直存在的， 而动态存储变量则时而存在时而消失。我们又把这种由于变量存储方式不同而产生的特性称变量的生存期。 生存期表示了变量存在的时间。 生存期和作用域是从时间和空间这两个不同的角度来描述变量的特性，这两者既有联系，又有区别。一个变量究竟属于哪一种存储方式， 并不能仅从其作用域来判断，还应有明确的存储类型说明。

在 C 语言中，对变量的存储类型说明有以下四种：

auto 自动变量
register 寄存器变量
extern 外部变量
static 静态变量

自动变量和寄存器变量属于动态存储方式， 外部变量和静态变量属于静态存储方式。在介绍了变量的存储类型之后， 可以知道对一个变量的说明不仅应说明其数据类型，还应说明其存储类型。 因此变量说明的完整形式应为： 存储类型说明符 数据类型说明符 变量名，变量名…；例如：

```
static int a,b;                    说明 a,b 为静态类型变量
auto char c1,c2;                   说明 c1,c2 为自动字符变量
static int a[5]={1,2,3,4,5};       说明 a 为静整型数组
extern int x,y;                    说明 x,y 为外部整型变量
```

下面分别介绍以上四种存储类型：

一、自动变量的类型说明符为 auto。

这种存储类型是 C 语言程序中使用最广泛的一种类型。 C 语言规定， 函数内凡未加存储类型说明的变量均视为自动变量， 也就是说自动变量可省去说明符 auto。 在前面各章的程序中所定义的变量凡未加存储类型说明符的都是自动变量。例如：

```
{ int i,j,k;
char c;
……
}等价于： { auto int i,j,k;
auto char c;
……
}
```

自动变量具有以下特点：

1. 自动变量的作用域仅限于定义该变量的个体内。在函数中定义的自动变量，只在该函数内有

效。在复合语句中定义的自动变量只在该复合语句中有效。 例如：

```
int kv(int a)
{
auto int x,y;
{ auto char c;
} /*c 的作用域*/
……
} /*a,x,y 的作用域*/
```

2. 自动变量属于动态存储方式，只有在使用它，即定义该变量的函数被调用时才给它分配存储单元，开始它的生存期。函数调用结束，释放存储单元，结束生存期。因此函数调用结束之后，自动变量的值不能保留。在复合语句中定义的自动变量，在退出复合语句后也不能再使用，否则将引起错误。例如以下程序：

```
main()
{ auto int a,s,p;
printf("\ninput a number:\n");
scanf("%d",&a);
if(a>0){
s=a+a;
p=a*a;
}
printf("s=%d p=%d\n",s,p);
}
```

s,p 是在复合语句内定义的自动变量，只能在该复合语句内有效。而程序的第 9 行却是退出复合语句之后用 printf 语句输出 s,p 的值，这显然会引起错误。

3. 由于自动变量的作用域和生存期都局限于定义它的个体内(函数或复合语句内)，因此不同的个体中允许使用同名的变量而不会混淆。 即使在函数内定义的自动变量也可与该函数内部的复合语句中定义的自动变量同名。

```
main()
{
auto int a,s=100,p=100;
printf("\ninput a number:\n");
scanf("%d",&a);
if(a>0)
{
auto int s,p;
s=a+a;
p=a*a;
printf("s=%d p=%d\n",s,p);
}
printf("s=%d p=%d\n",s,p);
}
```

本程序在 main 函数中和复合语句内两次定义了变量 s,p 为自动变量。按照 C 语言的规定，在复合语句内，应由复合语句中定义的 s,p 起作用，故 s 的值应为 a+ a，p 的值为 a*a。退出复合语句后的 s,p 应为 main 所定义的 s,p，其值在初始化时给定，均为 100。从输出结果可以分析出两个 s 和两个 p 虽变量名相同， 但却是两个不同的变量。

4. 对构造类型的自动变量如数组等，不可作初始化赋值。

二、外部变量外部变量的类型说明符为 extern。

在前面介绍全局变量时已介绍过外部变量。这里再补充说明外部变量的几个特点：
1. 外部变量和全局变量是对同一类变量的两种不同角度的提法。全局变是是从它的作用域提出的，外部变量从它的存储方式提出的，表示了它的生存期。

2. 当一个源程序由若干个源文件组成时， 在一个源文件中定义的外部变量在其它的源文件中也有效。例如有一个源程序由源文件 F1.C 和 F2.C 组成： F1.C

```
int a,b; /*外部变量定义*/
char c; /*外部变量定义*/
main()
{
……
}
```

F2.C

```
extern int a,b; /*外部变量说明*/
extern char c; /*外部变量说明*/
func (int x,y)
{
……
}
```

在 F1.C 和 F2.C 两个文件中都要使用 a,b,c 三个变量。在 F1.C 文件中把 a,b,c 都定义为外部变量。在 F2.C 文件中用 extern 把三个变量说明为外部变量，表示这些变量已在其它文件中定义，并把这些变量的类型和变量名，编译系统不再为它们分配内存空间。 对构造类型的外部变量， 如数组等可以在说明时作初始化赋值，若不赋初值，则系统自动定义它们的初值为 0。

三、静态变量

静态变量的类型说明符是 static。 静态变量当然是属于静态存储方式，但是属于静态存储方式的量不一定就是静态变量， 例如外部变量虽属于静态存储方式，但不一定是静态变量，必须由 static 加以定义后才能成为静态外部变量，或称静态全局变量。 对于自动变量，前面已经介绍它属于动态存储方式。 但是也可以用 static 定义它为静态自动变量，或称静态局部变量，从而成为静态存储方式。
由此看来， 一个变量可由 static 进行再说明，并改变其原有的存储方式。

1. 静态局部变量

在局部变量的说明前再加上 static 说明符就构成静态局部变量。

例如：

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

静态局部变量属于静态存储方式，它具有以下特点：

(1)静态局部变量在函数内定义，但不象自动变量那样，当调用时就存在，退出函数时就消失。静态局部变量始终存在着，也就是说它的生存期为整个源程序。

(2)静态局部变量的生存期虽然为整个源程序，但是其作用域仍与自动变量相同，即只能在定义该变量的函数内使用该变量。退出该函数后， 尽管该变量还继续存在，但不能使用它。

(3)允许对构造类静态局部量赋初值。在数组一章中，介绍数组初始化时已作过说明。若未赋以初值，则由系统自动赋以 0 值。

(4)对基本类型的静态局部变量若在说明时未赋以初值，则系统自动赋予 0 值。而对自动变量不赋初值，则其值是不定的。 根据静态局部变量的特点， 可以看出它是一种生存期为整个源程序的量。虽然离开定义它的函数后不能使用，但如再次调用定义它的函数时，它又可继续使用， 而且保存了前次被调用后留下的值。 因此，当多次调用一个函数且要求在调用之间保留某些变量的值时，可考虑采用静态局部变量。虽然用全局变量也可以达到上述目的，但全局变量有时会造成意外的副作用，因此仍以采用局部静态变量为宜。

```
main()
{
int i;
void f(); /*函数说明*/
for(i=1;i<=5;i++)
f(); /*函数调用*/
}
void f() /*函数定义*/
{
auto int j=0;
++j;
printf("%d\n",j);
}
```

程序中定义了函数 f，其中的变量 j 说明为自动变量并赋予初始值为 0。当 main 中多次调用 f 时，j 均赋初值为 0，故每次输出值均为 1。现在把 j 改为静态局部变量，程序如下：

```
main()
{
int i;
void f();
for (i=1;i<=5;i++)
f();
}
void f()
{
```

```
static int j=0;
++j;
printf("%d\n",j);
}
void f()
{
static int j=0;
++j;
printf("%d/n",j);
}
```

由于 j 为静态变量，能在每次调用后保留其值并在下一次调用时继续使用，所以输出值成为累加的结果。读者可自行分析其执行过程。

2.静态全局变量

全局变量(外部变量)的说明之前再冠以 static 就构成了静态的全局变量。全局变量本身就是静态存储方式， 静态全局变量当然也是静态存储方式。 这两者在存储方式上并无不同。这两者的区别虽在于非静态全局变量的作用域是整个源程序， 当一个源程序由多个源文件组成时，非静态的全局变量在各个源文件中都是有效的。 而静态全局变量则限制了其作用域， 即只在定义该变量的源文件内有效， 在同一源程序的其它源文件中不能使用它。由于静态全局变量的作用域局限于一个源文件内，只能为该源文件内的函数公用， 因此可以避免在其它源文件中引起错误。从以上分析可以看出， 把局部变量改变为静态变量后是改变了它的存储方式即改变了它的生存期。把全局变量改变为静态变量后是改变了它的作用域， 限制了它
的使用范围。因此 static 这个说明符在不同的地方所起的作用是不同的。应予以注意。

四、寄存器变量

上述各类变量都存放在存储器内， 因此当对一个变量频繁读写时，必须要反复访问内存储器，从而花费大量的存取时间。 为此， C 语言提供了另一种变量，即寄存器变量。这种变量存放在 CPU 的寄存器中，使用时，不需要访问内存，而直接从寄存器中读写， 这样可提高效率。寄存器变量的说明符是 register。 对于循环次数较多的循环控制变量及循环体内反复使用的变量均可定义为寄存器变量。

求∑200i=1imain()

```
{
register i,s=0;
for(i=1;i<=200;i++)
s=s+i;
printf("s=%d\n",s);
}
```

本程序循环 200 次，i 和 s 都将频繁使用，因此可定义为寄存器变量。
对寄存器变量还要说明以下几点：

1. 只有局部自动变量和形式参数才可以定义为寄存器变量。因为寄存器变量属于动态存储方式。凡需要采用静态存储方式的量不能定义为寄存器变量。

2. 在 Turbo C，MS C 等微机上使用的 C 语言中， 实际上是把寄存器变量当成自动变量处理的。因此速度并不能提高。 而在程序中允许使用寄存器变量只是为了与标准 C 保持一致。3. 即使能真正使用寄存器变量的机器，由于 CPU 中寄存器的个数是有限的，因此使用寄存器变量的个数也是有限的。

内部函数和外部函数

函数一旦定义后就可被其它函数调用。 但当一个源程序由多个源文件组成时， 在一个源文件中定义的函数能否被其它源文件中的函数调用呢?为此， C 语言又把函数分为两类：

一、内部函数

如果在一个源文件中定义的函数只能被本文件中的函数调用，而不能被同一源程序其它文件中的函数调用， 这种函数称为内部函
数。定义内部函数的一般形式是： static 类型说明符 函数名(形参表) 例如：
static int f(int a,int b) 内部函数也称为静态函数。但此处静态 static 的含义已不是指存储方式，而是指对函数的调用范围只局限于本文件。 因此在不同的源文件中定义同名的静态函数不会引起混淆。

二、外部函数

外部函数在整个源程序中都有效，其定义的一般形式为： extern 类型说明符 函数名(形参表) 例如：
extern int f(int a,int b)如在函数定义中没有说明 extern 或 static 则隐含为 extern。在一个源文件的函数中调用其它源文件中定义的外部函数时，应 用 extern 说明被调函数为外部函数。例如：

```
F1.C (源文件一)
main()
{
extern int f1(int i); /*外部函数说明，表示 f1 函
数在其它源文件中*/
……
}
F2.C (源文件二)
extern int f1(int i); /*外部函数定义*/
{
……
}
```

第五章　贪心的数组

为何说数组“贪心”？不假！在数组定义的时候，他就尽他的所有想要的抢占好他的内存，它才不管什么其他人呢！然后，他在以后再自己慢慢独享。不过，尽管如此，C 语言家族并没有把这个“贪心的数组”驱逐出去。因为他有能力“贪心”。他也被公认很重要的一种数据类型。

```
#include<stdio.h>
main()
{  int a[3][3],i,j;
for(i=0;i<3;i++)
   for(j=0;j<3;j++)
   scanf("%d",&a[i][j]);
for(i=0;i<3;i++){
   for(j=0;j<3;j++)
   printf("%d ",a[i][j]);
   printf("\n");}
   getch();
   }
```

这是一个最基本的二维数组的输入输出的程序，你会发现，在定义二维数组时候我们直接定义 int a[3][3],系统就直接在内存里给数组 3*3 个 int 型数据类型的内存空间，二维数组的输入输出都要二重循环实现，并且，在输出时候还要换行。接下来，我们具体介绍这个“贪心的数组”。

数组说明的一般形

式为： 类型说明符 数组名 [常量表达式]，……； 其中，类型说明符是任一种基本数据类型或构造数据类型。 数组名是用户定义的数组标识符。 方括号中的常量表达式表示数据元素的个数，也称为数组的长度。

例如：

int a[10]; 说明整型数组 a，有 10 个元素。

float b[10],c[20]; 说明实型数组 b，有 10 个元素，实型数组 c，有 20 个元素。

char ch[20]; 说明字符数组 ch，有 20 个元素。

对于数组类型说明应注意以下几点：

1.数组的类型实际上是指数组元素的取值类型。对于同一个数组，其所有元素的数据类型都是相同的。

2.数组名的书写规则应符合标识符的书写规定。

3.数组名不能与其它变量名相同，例如：

```
void main()
{
int a;
float a[10];
……
```

```
}
```
是错误的。
4.方括号中常量表达式表示数组元素的个数，如 a[5]表示数组 a 有 5 个元素。但是其下标从 0 开始计算。因此 5 个元素分别为 a[0],a[1],a[2],a[3],a[4]。
5.不能在方括号中用变量来表示元素的个数， 但是可以是符号常数或常量表达式。例如：
```
#define FD 5
void main()
{
int a[3+2],b[7+FD];
……
}
```
是合法的。但是下述说明方式是错误的。
```
void main()
{
int n=5;
int a[n];
……
}
```
6.允许在同一个类型说明中，说明多个数组和多个变量。
例如： int a,b,c,d,k1[10],k2[20];

数组元素的表示方法

数组元素是组成数组的基本单元。数组元素也是一种变量， 其标识方法为数组名后跟一个下标。 下标表示了元素在数组中的顺序号。数组元素的一般形式为：数组名[下标] 其中的下标只能为整型常量或整型表达式。如为小数时，C 编译将自动取整。例如，a[5],a[i+j],a[i++]都是合法的数组元素。 数组元素通常也称为下标变量。必须先定义数组， 才能使用下标变量。在C语言中只能逐个地使用下标变量， 而不能一次引用整个数组。例如，输出有 10 个元素的数组必须使用循环语句逐个输出各下标变量：
for(i=0; i<10; i++) printf("%d",a[i]); 而不能用一个语句输出整个数组，下面的写法是错误的： printf("%d",a);
```
void main()
{
int i,a[10];
for(i=0;i<10;i++)
a[i++]=2*i+1;
for(i=9;i>=0;i--)
printf("%d",a[i]);
printf("\n%d %d\n",a[5.2],a[5.8]);
}
```
本例中用一个循环语句给 a 数组各元素送入奇数值,然后用第二个循环语句从大到小输出各个奇数。在第一个 for 语句中，表达式 3 省略了。在下标变量中使用了表达式 i++，用以修改循环变量。当然第二个 for 语句也可以这样作， C语言允许用表达式表示下标。 程序中最后一个 printf 语句输出了两次 a[5]的值， 可以看出当下标不为整数时将自动取整。

数组的赋值给数组赋值的方法除了用赋值语句对数组元素逐个赋值外， 还可采用初始化赋值和动态赋值的方法。数组初始化赋值数组初始化赋值是指在数组说明时给数组元素赋予初值。 数组初始化是在编译阶段进行的。这样将减少运行时间，提高效率。

初始化赋值的一般形式为：static 类型说明符 数组名[常量表达式]={值,值……值}；其中 static 表示是静态存储类型， C 语言规定只有静态存储数组和外部存储数组才可作初始化赋值(有关静态存储，外部存储的概念在第五章中介绍)。在{ }中的各数据值即为各元素的初值， 各值之间用逗号间隔。例如： static int a[10]={ 0,1,2,3,4,5,6,7,8,9 }; 相当于 a[0]=0;a[1]=1...a[9]=9;

C 语言对数组的初始赋值还有以下几点规定：

1.可以只给部分元素赋初值。当{ }中值的个数少于元素个数时，只给前面部分元素赋值。例如： static int a[10]={0,1,2,3,4};表示只给 a[0]～a[4]5 个元素赋值，而后 5 个元素自动赋 0 值。

2.只能给元素逐个赋值，不能给数组整体赋值。 例如给十个元素全部赋 1 值，只能写为：static int a[10]={1,1,1,1,1,1,1,1,1,1};而不能写为： static int a[10]=1;

3.如不给可初始化的数组赋初值，则全部元素均为 0 值。

4.如给全部元素赋值，则在数组说明中， 可以不给出数组元素的个数。例如： static int a[5]={1,2,3,4,5};可写为： static int a[]={1,2,3,4,5};动态赋值可以在程序执行过程中，对数组作动态赋值。 这时可用循环语句配合 scanf 函数逐个对数组元素赋值。

```
void main()
{
int i,max,a[10];
printf("input 10 numbers:\n");
for(i=0;i<10;i++)
scanf("%d",&a[i]);
max=a[0];
for(i=1;i<10;i++)
if(a[i]>max) max=a[i];
printf("maxmum=%d\n",max);
}
```

本例程序中第一个 for 语句逐个输入 10 个数到数组 a 中。 然后把 a[0]送入 max 中。在第二个 for 语句中，从 a[1]到 a[9]逐个与 max 中的内容比较，若比 max 的值大，则把该下标变量送入 max 中，因此 max 总是在已比较过的下标变量中为最大者。比较结束，输出 max 的值。

```
void main()
{
int i,j,p,q,s,a[10];
printf("\n input 10 numbers:\n");
for(i=0;i<10;i++)
scanf("%d",&a[i]);
for(i=0;i<10;i++){
p=i;q=a[i];
for(j=i+1;j<10;j++)
```

```
if(q<a[j]) { p=j;q=a[j]; }
if(i!=p)
{s=a[i];
a[i]=a[p];
a[p]=s; }
printf("%d",a[i]);
}
}
```

本例程序中用了两个并列的 for 循环语句，在第二个 for 语句中又嵌套了一个循环语句。第一个 for 语句用于输入 10 个元素的初值。第二个 for 语句用于排序。本程序的排序采用逐个比较的方法进行。在 i 次循环时，把第一个元素的下标 i 赋于 p，而把该下标变量值 a[i]赋于 q。然后进入小循环，从 a[i+1]起到最后一个元素止逐个与 a[i]作比较，有比 a[i]大者则将其下标送 p，元素值送 q。 一次循环结束后，p 即为最大元素的下标，q 则为该元素值。若此时 i≠p，说明 p,q 值均已不是进入小循环之前所赋之值，则交换 a[i]和 a[p]之值。 此时 a[i]为已排序完毕的元素。输出该值之后转入下一次循环。对 i+1 以后各个元素排序。

二维数组

前面介绍的数组只有一个下标，称为一维数组， 其数组元素也称为单下标变量。在实际问题中有很多量是二维的或多维的， 因此C语言允许构造多维数组。多维数组元素有多个下标， 以标识它在数组中的位置，所以也称为多下标变量。 本小节只介绍二维数组，多维数组可由二维数组类推而得到。二维数组类型说明二维数组类型说明的一般形式是：
类型说明符 数组名[常量表达式 1][常量表达式 2]…;
其中常量表达式 1 表示第一维下标的长度，常量表达式 2 表示第二维下标的长度。例如：
int a[3][4]; 说明了一个三行四列的数组，数组名为 a，其下标变量的类型为整型。该数组的下标变量共有 3×4 个，即： a[0][0],a[0][1],a[0][2],a[0][3]
a[1][0],a[1][1],a[1][2],a[1][3]
a[2][0],a[2][1],a[2][2],a[2][3]

二维数组在概念上是二维的，即是说其下标在两个方向上变化， 下标变量在数组中的位置也处于一个平面之中， 而不是象一维数组只是一个向量。但是，实际的硬件存储器却是连续编址的， 也就是说存储器单元是按一维线性排列的。 如何在一维存储器中存放二维数组，可有两种方式：一种是按行排列， 即放完一行之后顺次放入第二行。另一种是按列排列， 即放完一列之后再顺次放入第二列。在C语言中，二维数组是按行排列的。 在图 4.1 中，按行顺次存放，先存放 a[0]行，再存放 a[1]行，最后存放 a[2]行。每行中有四个元素也是依次存放。由于数组 a 说明为
int 类型，该类型占两个字节的内存空间，所以每个元素均占有两个 字节(图中每一格为一字节)。

二维数组元素的表示方法

二维数组的元素也称为双下标变量，其表示的形式为： 数组名[下标][下标] 其中下标应为整型常量或整型表达式。例如： a[3][4] 表示 a 数组三行四列的元素。下标变量和数组说明在形式中有些相似，但这两者具有完全不同的含义。 数组说明的方括号中给出的是

某一维的长度，即可取下标的最大值； 而数组元素中的下标是该元素在数组中的位置标识。前者只能是常量， 后者可以是常量，变量或表达式。

一个学习小组有 5 个人，每个人有三门课的考试成绩。求全组分科的平均成绩和各科总平均成绩。

课程 成绩姓名	Math	C	DBASE
张	80	75	92
王	61	65	71
李	59	63	70
赵	85	87	90
周	76	77	85

可设一个二维数组 a[5][3]存放五个人三门课的成绩。再设一个一维数组 v[3]存放所求得各分科平均成绩，设变量 l 为全组各科总平均成绩。编程如下：

```
void main()
{
int i,j,s=0,l,v[3],a[5][3];
printf("input score\n");
for(i=0;i<3;i++){
for(j=0;j<5;j++)
{ scanf("%d",&a[j][i]);
s=s+a[j][i];}
v[i]=s/5;
s=0;
}
l=(v[0]+v[1]+v[2])/3;
printf("math:%d\nc languag:%d\ndbase:%d\n",v[0],v[1],v[2]);
printf("total:%d\n",l);
}
```

程序中首先用了一个双重循环。 在内循环中依次读入某一门课程的各个学生的成绩，并把这些成绩累加起来， 退出内循环后再把该累加成绩除以 5 送入 v[i]之中，这就是该门课程的平均成绩。外循环共循环三次，分别求出三门课各自的平均成绩并存放在 v 数组之中。退出外循环之后，把 v[0],v[1],v[2]相加除以 3 即得到各科总平均成绩。最后按题意输出各个成绩。

二维数组的初始化

二维数组初始化也是在类型说明时给各下标变量赋以初值。二维数组可按行分段赋值，也可按行连续赋值。 例如对数组 a[5][3]：

1. 按行分段赋值可写为 `static int a[5][3]={ {80,75,92},{61,65,71},{59,63,70},{85,87,90},{76,77,85} };`

2. 按行连续赋值可写为 `static int a[5][3]={ 80,75,92,61,65,71,59,63,70,85,87,90,76,77,85 };`

这两种赋初值的结果是完全相同的。

```
void main()
{
int i,j,s=0,l,v[3];
```

```
static int a[5][3]={ {80,75,92},{61,65,71},{59,63,70},
{85,87,90},{76,77,85} };
for(i=0;i<3;i++)
{ for(j=0;j<5;j++)
s=s+a[j][i];
v[i]=s/5;
s=0;
}
l=(v[0]+v[1]+v[2])/3;
printf("math:%d\nc languag:%d\ndbase:%d\n",v[0],v[1],v[2]);
printf("total:%d\n",l);
}
```

对于二维数组初始化赋值还有以下说明：

1.可以只对部分元素赋初值，未赋初值的元素自动取 0 值。

例如： static int a[3][3]={{1},{2},{3}}; 是对每一行的第一列元素赋值，未赋值的元素取 0 值。 赋值后各元素的值为： 1 0 02 0 03 0 0

static int a [3][3]={{0,1},{0,0,2},{3}}; 赋值后的元素值为 0 1 00 0 23 0 0

2.如对全部元素赋初值，则第一维的长度可以不给出。

例如： static int a[3][3]={1,2,3,4,5,6,7,8,9}; 可以写为： static int a[][3]={1,2,3,4,5,6,7,8,9};

数组是一种构造类型的数据。 二维数组可以看作是由一维数组的嵌套而构成的。设一维数组的每个元素都又是一个数组， 就组成了二维数组。当然，前提是各元素类型必须相同。根据这样的分析，一个二维数组也可以分解为多个一维数组。 C语言允许这种分解有二维数组 a[3][4]，可分解为三个一维数组，其数组名分别为 a[0],a[1],a[2]。对这三个一维数组不需另作说明即可使用。这三个一维数组都有 4 个元素，例如：一维数组 a[0]的元素为 a[0][0],a[0][1],a[0][2],a[0][3]。必须强调的是，a[0],a[1],a[2]不能当作下标变量使用，它们是数组名，不是一个单纯的下标变量。

字符数组

用来存放字符量的数组称为字符数组。 字符数组类型说明的形式与前面介绍的数值数组相同。例如： char c[10]; 由于字符型和整型通用，也可以定义为 int c[10]但这时每个数组元素占 2 个字节的内存单元。字符数组也可以是二维或多维数组，例如： char c[5][10];即为二维字符数组。 字符数组也允许在类型说明时作初始化赋值。例如：static char c[10]={`c`,` `,`p`,`r`,o`,g`,r`,`a`,`m`};赋值后各元素的值为： 数组 C c[0]c[1]c[2]c[3]c[4]c [5]c[6]c[7]c[8]c[9]其中 c[9]未赋值，由系统自动赋予 0 值。 当对全体元素赋初值时也可以省去长度说明。例如： static char c[]={`c`,` `,`p`,`r`,`o`,`g`,`r`,`a`,`m`};这时 C 数组的长度自动定为 9。

```
main()
{
int i,j;
char a[][5]={{'B','A','S','I','C',},{'d','B','A','S','E'}};
for(i=0;i<=1;i++)
{
```

```
for(j=0;j<=4;j++)
printf("%c",a[i][j]);
printf("\n");
}
}
```

本例的二维字符数组由于在初始化时全部元素都赋以初值， 因此一维下标的长度可以不加以说明。字符串在C语言中没有专门的字符串变量， 通常用一个字符数组来存放一个字符串。在2.1.4节介绍字符串常量时，已说明字符串总是以'\0'作为串的结束符。因此当把一个字符串存入一个数组时， 也把结束符'\0'存入数组，并以此作为该字符串是否结束的标志。 有了'\0'标志后，就不必再用字符数组的长度来判断字符串的长度了。

C语言允许用字符串的方式对数组作初始化赋值。例如：

```
static char c[]={'c', ' ','p','r','o','g','r','a','m'}; 可写为：
static char c[]={"C program"}; 或去掉{}写为：
sratic char c[]="C program";
```

用字符串方式赋值比用字符逐个赋值要多占一个字节，用于存放字符串结束标志'\0'。上面的数组c在内存中的实际存放情况为： C program\0`\0'是由C编译系统自动加上的。由于采用了`\0'标志，所以在用字符串赋初值时一般无须指定数组的长度， 而由系统自行处理。在采用字符串方式后，字符数组的输入输出将变得简单方便。 除了上述用字符串赋初值的办法外，还可用printf函数和scanf函数一次性输出输入一个字符数组中的字符串，而不必使用循环语句逐个地输入输出每个字符。

```
void main()
{
static char c[]="BASIC\ndBASE";
printf("%s\n",c);
}
```

注意在本例的printf函数中，使用的格式字符串为“%s”， 表示输出的是一个字符串。而在输出表列中给出数组名则可。 不能写为： printf("%s",c[]);

```
void main()
{
char st[15];
printf("input string:\n");
scanf("%s",st);
printf("%s\n",st);
}
```

本例中由于定义数组长度为15， 因此输入的字符串长度必须小于15，以留出一个字节用于存放字符串结束标志`\0`。 应该说明的是，对一个字符数组，如果不作初始化赋值，则必须说明数组长度。还应该特别注意的是，当用scanf函数输入字符串时，字符串中不能含有空格，否则将以空格作为串的结束符。例如运行例4.8，当输入的字符串中含有空格时，运行情况为： input string:this is a book this 从输出结果可以看出空格以后的字符都未能输出。 为了避免这种情况， 可多设几个字符数组分段存放含空格的串。程序可改写如下：

```
Lesson
void main()
{
```

```
char st1[6],st2[6],st3[6],st4[6];
printf("input string:\n");
scanf("%s%s%s%s",st1,st2,st3,st4);
printf("%s %s %s %s\n",st1,st2,st3,st4);
}
```

本程序分别设了四个数组， 输入的一行字符的空格分段分别装入四个数组。然后分别输出这四个数组中的字符串。在前面介绍过，scanf 的各输入项必须以地址方式出现，如&a,&b 等。但在例 4.8 中却是以数组名方式出现的，这是为什么呢?这是由于在C语言中规定，数组名就代表了该数组的首地址。 整个数组是以首地址开头的一块连续的内存单元。如有字符数组 char c[10]。设数组 c 的首地址为 2000，也就是说 c[0]单元地址为 2000。则数组名 c 就代表这个首地址。因此在 c 前面不能再加地址运算符&。如写作 scanf("%s",&c);则是错误的。 在执行函数 printf("%s",c) 时，按数组名 c 找到首地址，然后逐个输出数组中各个字符直到遇到字符串终止标志'\0'为止。

字符串常用函数

C语言提供了丰富的字符串处理函数， 大致可分为字符串的输入、输出、合并、修改、比较、转换、复制、搜索几类。 使用这些函数可大大减轻编程的负担。用于输入输出的字符串函数， 在使用前应包含头文件"stdio.h" ； 使用其它字符串函数则应包含头文件"string.h"。 下面介绍几个最常用的字符串函数。

1.字符串输出函数 puts 格式： puts (字符数组名) 功能：把字符数组中的字符串输出到显示器。 即在屏幕上显示该字符串

```
#include"stdio.h"
main()
{
static char c[]="BASIC\ndBASE";
puts(c);
}
```

从程序中可以看出 puts 函数中可以使用转义字符， 因此输出结果成为两行。puts 函数完全可以由 printf 函数取代。 当需要按一定格式输出时，通常使用 printf 函数。

2.字符串输入函数 gets 格式： gets (字符数组名) 功能：从标准输入设备键盘上输入一个字符串。 本函数得到一个函数值，即为该字符数组的首地址。

```
#include"stdio.h"
main()
{
char st[15];
printf("input string:\n");
gets(st);
puts(st);
}
```

可以看出当输入的字符串中含有空格时，输出仍为全部字符串。说明 gets 函数并不以空格作为字符串输入结束的标志， 而只以回车作为输入结束。这是与 scanf 函数不同的。

3.字符串连接函数 strcat 格式： strcat (字符数组名 1，字符数组名 2) 功能：把字符数

组 2 中的字符串连接到字符数组 1 中字符串的后面，并删去字符串 1 后的串标志“\0”。本函数返回值是字符数组 1 的首地址。

```
#include"string.h"
main()
{
static char st1[30]="My name is ";
int st2[10];
printf("input your name:\n");
gets(st2);
strcat(st1,st2);
puts(st1);
}
```

本程序把初始化赋值的字符数组与动态赋值的字符串连接起来。 要注意的是，字符数组 1 应定义足够的长度，否则不能全部装入被连接的字符串

4.字符串拷贝函数 strcpy 格式： strcpy (字符数组名 1，字符数组名 2) 功能：把字符数组 2 中的字符串拷贝到字符数组 1 中。串结束标志“\0”也一同拷贝。字符数名 2， 也可以是一个字符串常量。这时相当于把一个字符串赋予一个字符数组。

```
#include"string.h"
main()
{
static char st1[15],st2[]="C Language";
strcpy(st1,st2);
puts(st1);printf("\n");
}
```

本函数要求字符数组 1 应有足够的长度，否则不能全部装入所拷贝的字符串。

5.字符串比较函数 strcmp 格式： strcmp(字符数组名 1，字符数组名 2) 功能：按照 ASCII 码顺序比较两个数组中的字符串，并由函数返回值返回比较结果。

字符串 1＝字符串 2，返回值＝0；

字符串 2〉字符串 2，返回值〉0；

字符串 1〈字符串 2，返回值〈0。

本函数也可用于比较两个字符串常量，或比较数组和字符串常量。

```
#include"string.h"
main()
{ int k;
static char st1[15],st2[]="C Language";
printf("input a string:\n");
gets(st1);
k=strcmp(st1,st2);
if(k==0) printf("st1=st2\n");
if(k>0) printf("st1>st2\n");
if(k<0) printf("st1<st2\n");
}
```

本程序中把输入的字符串和数组 st2 中的串比较，比较结果返回到 k 中，根据 k 值再输

出结果提示串。当输入为 dbase 时，由 ASCII 码可知“dBASE”大于“C Language”故 k〉0,输出结果“st1>st2”。

6.测字符串长度函数 strlen 格式： strlen(字符数组名) 功能：测字符串的实际长度(不含字符串结束标志‘\0’) 并作为函数返回值。

```
#include"string.h"
main()
{ int k;
static char st[]="C language";
k=strlen(st);
printf("The lenth of the string is %d\n",k);
}
```

程序举例

把一个整数按大小顺序插入已排好序的数组中。 为了把一个数按大小插入已排好序的数组中， 应首先确定排序是从大到小还是从小到大进行的。设排序是从大到小进序的， 则可把欲插入的数与数组中各数逐个比较， 当找到第一个比插入数小的元素 i 时，该元素之前即为插入位置。然后从数组最后一个元素开始到该元素为止，逐个后移一个单元。最后把插入数赋予元素 i 即可。如果被插入数比所有的元素值都小则插入最后位置。

```
main()
{
int i,j,p,q,s,n,a[11]={127,3,6,28,54,68,87,105,162,18};
for(i=0;i<10;i++)
{ p=i;q=a[i];
for(j=i+1;j<10;j++)
if(q<a[j]) {p=j;q=a[j];}
if(p!=i)
{
s=a[i];
a[i]=a[p];
a[p]=s;
}
printf("%d ",a[i]);
}
printf("\ninput number:\n");
scanf("%d",&n);
for(i=0;i<10;i++)
if(n>a[i])
{for(s=9;s>=i;s--) a[s+1]=a[s];
break;}
a[i]=n;
for(i=0;i<=10;i++)
printf("%d ",a[i]);
printf("\n");
```

```
}
```

本程序首先对数组 a 中的 10 个数从大到小排序并输出排序结果。然后输入要插入的整数 n。再用一个 for 语句把 n 和数组元素逐个比较，如果发现有 n>a[i]时，则由一个内循环把 i 以下各元素值顺次后移一个单元。后移应从后向前进行(从 a[9]开始到 a[i]为止)。 后移结束跳出外循环。插入点为 i，把 n 赋予 a[i]即可。 如所有的元素均大于被插入数，则并未进行过后移工作。此时 i=10，结果是把 n 赋于 a[10]。最后一个循环输出插入数后的数组各元素值。程序运行时，输入数 47。从结果中可以看出 47 已插入到 54 和 28 之间。

在二维数组 a 中选出各行最大的元素组成一个一维数组 b。 a=3 16 87 65 4 32 11 108 10 25 12 37b=(87 108 37) 本题的编程思路是，在数组 A 的每一行中寻找最大的元素，找到之后把该值赋予数组 B 相应的元素即可。程序如下：

```
main()
{
static int a[][4]={3,16,87,65,4,32,11,108,10,25,12,27};
int b[3],i,j,l;
for(i=0;i<=2;i++)
{ l=a[i][0];
for(j=1;j<=3;j++)
if(a[i][j]>l) l=a[i][j];
b[i]=l;}
printf("\narray a:\n");
for(i=0;i<=2;i++)
{ for(j=0;j<=3;j++)
printf("%5d",a[i][j]);
printf("\n");}
printf("\narray b:\n");
for(i=0;i<=2;i++)
printf("%5d",b[i]);
printf("\n");
}
```

程序中第一个 for 语句中又嵌套了一个 for 语句组成了双重循环。外循环控制逐行处理，并把每行的第 0 列元素赋予 l。进入内循环后，把 l 与后面各列元素比较，并把比 l 大者赋予 l。内循环结束时 l 即为该行最大的元素，然后把 l 值赋予 b[i]。等外循环全部完成时，数组 b 中已装入了 a 各行中的最大值。后面的两个 for 语句分别输出数组 a 和数组 b。

输入五个国家的名称按字母顺序排列输出。

本题编程思路如下：五个国家名应由一个二维字符数组来处理。然而 C 语言规定可以把一个二维数组当成多个一维数组处理。 因此本题又可以按五个一维数组处理， 而每一个一维数组就是一个国家名字符串。用字符串比较函数比较各一维数组的大小，并排序， 输出结果即可。

编程如下：

```
void main()
{
```

```
char st[20],cs[5][20];
int i,j,p;
printf("input country's name:\n");
for(i=0;i<5;i++)
gets(cs[i]);
printf("\n");
for(i=0;i<5;i++)
{ p=i;strcpy(st,cs[i]);
for(j=i+1;j<5;j++)
if(strcmp(cs[j],st)<0) {p=j;strcpy(st,cs[j]);}
if(p!=i)
{
strcpy(st,cs[i]);
strcpy(cs[i],cs[p]);
strcpy(cs[p],st);
}
puts(cs[i]);}printf("\n");
}
```

本程序的第一个 for 语句中，用 gets 函数输入五个国家名字符串。上面说过C语言允许把一个二维数组按多个一维数组处理， 本程序说明 cs[5][20]为二维字符数组，可分为五个一维数组 cs[0]，cs[1]，cs[2]，cs[3]，cs[4]。因此在 gets 函数中使用 cs[i]是合法的。 在第二个 for 语句中又嵌套了一个 for 语句组成双重循环。 这个双重循环完成按字母顺序排序的工作。在外层循环中把字符数组 cs[i]中的国名字符串拷贝到数组 st 中，并把下标 i 赋予 P。 进入内层循环后，把 st 与 cs[i]以后的各字符串作比较，若有比 st 小者则把该字符串拷贝到 st 中，并把其下标赋予 p。内循环完成后如 p 不等于 i 说明有比 cs[i]更小的字符串出现，因此交换 cs[i]和 st 的内容。 至此已确定了数组 cs 的第 i 号元素的排序值。然后输出该字符串。在外循环全部完成之后即完成全部排序和输出。

第六节　指针

指针简介

指针是C语言中广泛使用的一种数据类型。运用指针编程是C语言最主要的风格之一。利用指针变量可以表示各种数据结构； 能很方便地使用数组和字符串； 并能象汇编语言一样处理内存地址，从而编出精练而高效的程序。指针极大地丰富了C语言的功能。 学习指针是学习C语言中最重要的一环， 能否正确理解和使用指针是我们是否掌握C语言的一个标志。同时， 指针也是C语言中最为困难的一部分，在学习中除了要正确理解基本概念，还必须要多编程，上机调试。只要作到这些，指针也是不难掌握的。

指针的基本概念 在计算机中，所有的数据都是存放在存储器中的。 一般把存储器中的一个字节称为一个内存单元， 不同的数据类型所占用的内存单元数不等，如整型量占 2 个单元，字符量占 1 个单元等， 在第二章中已有详细的介绍。为了正确地访问这些内存单元，必须为每个内存单元编上号。 根据一个内存单元的编号即可准确地找到该内存单元。内存单元的编号也叫做地址。 既然根据内存单元的编号或地址就可以找到所需的内存单元，所以通常也把这个地址称为指针。 内存单元的指针和内存单元的内容是两个不同的概念。 可以用一个通俗的例子来说明它们之间的关系。我们到银行去存取款时， 银行工作人员将根据我们的帐号去找我们的存款单， 找到之后在存单上写入存款、取款的金额。在这里，帐号就是存单的指针， 存款数是存单的内容。对于一个内存单元来说，单元的地址即为指针，其中存放的数据才是该单元的内容。在C语言中， 允许用一个变量来存放指针，这种变量称为指针变量。因此， 一个指针变量的值就是某个内存单元的地址或称为某内存单元的指针。设有字符变量 C，其内容为“K”(ASCII 码为十进制数 75)，C 占用了 011A 号单元(地址用十六进数表示)。设有指针变量 P，内容为 011A， 这种情况我们称为 P 指向变量 C，或说 P 是指向变量 C 的指针。 严格地说，一个指针是一个地址， 是一个常量。而一个指针变量却可以被赋予不同的指针值，是变量。 但在常把指针变量简称为指针。为了避免混淆，我们中约定：“指针”是指地址， 是常量，“指针变量”是指取值为地址的变量。 定义指针的目的是为了通过指针去访问内存单元。

既然指针变量的值是一个地址， 那么这个地址不仅可以是变量的地址， 也可以是其它数据结构的地址。在一个指针变量中存放一个数组或一个函数的首地址有何意义呢？ 因为数组或函数都是连续存放的。通过访问指针变量取得了数组或函数的首地址， 也就找到了该数组或函数。这样一来， 凡是出现数组，函数的地方都可以用一个指针变量来表示， 只要该指针变量中赋予数组或函数的首地址即可。这样做， 将会使程序的概念十分清楚，程序本身也精练，高效。在C语言中， 一种数据类型或数据结构往往都占有一组连续的内存单元。 用“地址”这个概念并不能很好地描述一种数据类型或数据结构， 而“指针”虽然实际上也是一个地址，但它却是一个数据结构的首地址， 它是“指向”一个数据结构的，因而概念更为清楚，表示更为明确。 这也是引入“指针”概念的一个重要原因。

指针变量的类型说明

对指针变量的类型说明包括三个内容：

(1)指针类型说明，即定义变量为一个指针变量；
(2)指针变量名；
(3)变量值(指针)所指向的变量的数据类型。

其一般形式为： 类型说明符 *变量名；

其中，*表示这是一个指针变量，变量名即为定义的指针变量名，类型说明符表示本指针变量所指向的变量的数据类型。

例如： int *p1;表示 p1 是一个指针变量，它的值是某个整型变量的地址。 或者说 p1 指向一个整型变量。至于 p1 究竟指向哪一个整型变量， 应由向 p1 赋予的地址来决定。

再如：staic int *p2; /*p2 是指向静态整型变量的指针变量*/
float *p3; /*p3 是指向浮点变量的指针变量*/
char *p4; /*p4 是指向字符变量的指针变量*/ 应该注意的是，一个指针变量只能指向同类型的变量，如 P3 只能指向浮点变量，不能时而指向一个浮点变量， 时而又指向一个字符变量。

指针变量的赋值

指针变量同普通变量一样，使用之前不仅要定义说明， 而且必须赋予具体的值。未经赋值的指针变量不能使用， 否则将造成系统混乱，甚至死机。指针变量的赋值只能赋予地址， 决不能赋予任何其它数据，否则将引起错误。在C语言中， 变量的地址是由编译系统分配的，对用户完全透明，用户不知道变量的具体地址。 C语言中提供了地址运算符&来表示变量的地址。其一般形式为： & 变量名； 如&a 变示变量 a 的地址，&b 表示变量 b 的地址。 变量本身必须预先说明。设有指向整型变量的指针变量 p，如要把整型变量 a 的地址赋予 p 可以有以下两种方式：
(1)指针变量初始化的方法 int a;
int *p=&a;
(2)赋值语句的方法 int a;
int *p;
p=&a;
不允许把一个数赋予指针变量，故下面的赋值是错误的： int *p;p=1000; 被赋值的指针变量前不能再加“*”说明符，如写为*p=&a 也是错误的

指针变量的运算

指针变量可以进行某些运算，但其运算的种类是有限的。 它只能进行赋值运算和部分算术运算及关系运算。
1.指针运算符

(1)取地址运算符&

取地址运算符&是单目运算符，其结合性为自右至左，其功能是取变量的地址。在 scanf 函数及前面介绍指针变量赋值中，我们已经了解并使用了&运算符。

(2)取内容运算符*

取内容运算符*是单目运算符，其结合性为自右至左，用来表示指针变量所指的变量。在*运算符之后跟的变量必须是指针变量。需要注意的是指针运算符*和指针变量说明中的指针说明符* 不是一回事。在指针变量说明中，“*”是类型说明符，表示其后的变量是指针类型。而表达式中出现的“*”则是一个运算符用以表示指针变量所指的变量。

```
main(){
int a=5,*p=&a;
printf ("%d",*p);
}
.....
```

表示指针变量 p 取得了整型变量 a 的地址。本语句表示输出变量 a 的值。

2.指针变量的运算

(1)赋值运算

指针变量的赋值运算有以下几种形式：

①指针变量初始化赋值，前面已作介绍。

②把一个变量的地址赋予指向相同数据类型的指针变量。例如：

```
int a,*pa;
pa=&a; /*把整型变量 a 的地址赋予整型指针变量 pa*/
```

③把一个指针变量的值赋予指向相同类型变量的另一个指针变量。如：

```
int a,*pa=&a,*pb;
pb=pa; /*把 a 的地址赋予指针变量 pb*/
```

由于 pa,pb 均为指向整型变量的指针变量，因此可以相互赋值。

④把数组的首地址赋予指向数组的指针变量。

例如： int a[5],*pa;

pa=a; (数组名表示数组的首地址，故可赋予指向数组的指针变量 pa)

也可写为：

```
pa=&a[0]; /*数组第一个元素的地址也是整个数组的首地址，
```

也可赋予 pa*/

当然也可采取初始化赋值的方法：

```
int a[5],*pa=a;
```

⑤把字符串的首地址赋予指向字符类型的指针变量。例如： char *pc;pc="c language"; 或用初始化赋值的方法写为： char *pc="C Language"; 这里应说明的是并不是把整个字符串装入指针变量， 而是把存放该字符串的字符数组的首地址装入指针变量。 在后面还将详细介绍。

⑥把函数的入口地址赋予指向函数的指针变量。例如： int (*pf)();pf=f; /*f 为函数名*/

(2)加减算术运算

对于指向数组的指针变量，可以加上或减去一个整数 n。设 pa 是指向数组 a 的指针变量，则 pa+n,pa-n,pa++,++pa,pa--,--pa 运算都是合法的。指针变量加或减一个整数 n 的意义是把指针指向的当前位置(指向某数组元素)向前或向后移动 n 个位置。应该注意，数组指针变量向前或向后移动一个位置和地址加 1 或减 1 在概念上是不同的。因为数组可以有不同的类型， 各种类型的数组元素所占的字节长度是不同的。如指针变量加 1，即向后移动 1 个位置表示指针变量指向下一个数据元素的首地址。而不是在原地址基础上加 1。
例如：

```
int a[5],*pa;
pa=a; /*pa 指向数组 a，也是指向 a[0]*/
pa=pa+2; /*pa 指向 a[2]，即 pa 的值为&pa[2]*/
```

指针变量的加减运算只能对数组指针变量进行， 对指向其它类型变量的指针变量作加减运算是毫无意义的。(3)两个指针变量之间的运算只有指向同一数组的两个指针变量之间才能进行运算， 否则运算毫无意义。

①两指针变量相减
两指针变量相减所得之差是两个指针所指数组元素之间相差的元素个数。实际上是两个指针值(地址) 相减之差再除以该数组元素的长度(字节数)。例如 pf1 和 pf2 是指向同一浮点数组的两个指针变量，设 pf1 的值为 2010H，pf2 的值为 2000H，而浮点数组每个元素占 4 个字节，所以 pf1-pf2 的结果为(2000H-2010H)/4=4，表示 pf1 和 pf2 之间相差 4 个元素。两个指针变量不能进行加法运算。 例如， pf1+pf2 是什么意思呢?毫无实际意义。

②两指针变量进行关系运算
指向同一数组的两指针变量进行关系运算可表示它们所指数组元素之间的关系。例如：
pf1==pf2 表示 pf1 和 pf2 指向同一数组元素
pf1>pf2 表示 pf1 处于高地址位置
pf1<pf2 表示 pf2 处于低地址位置

```
main(){
int a=10,b=20,s,t,*pa,*pb;
pa=&a;
pb=&b;
s=*pa+*pb;
t=*pa**pb;
printf("a=%d\nb=%d\na+b=%d\na*b=%d\n",a,b,a+b,a*b);
printf("s=%d\nt=%d\n",s,t);
}
......
```

说明 pa,pb 为整型指针变量
给指针变量 pa 赋值，pa 指向变量 a。
给指针变量 pb 赋值，pb 指向变量 b。
本行的意义是求 a+b 之和，(*pa 就是 a，*pb 就是 b)。
本行是求 a*b 之积。
输出结果。
输出结果。

```
......
```

指针变量还可以与 0 比较。设 p 为指针变量，则 p==0 表明 p 是空指针，它不指向任何变量；p!=0 表示 p 不是空指针。空指针是由对指针变量赋予 0 值而得到的。例如： #define NULL 0 int *p=NULL; 对指针变量赋 0 值和不赋值是不同的。指针变量未赋值时，可以是任意值，是不能使用的。否则将造成意外错误。而指针变量赋 0 值后，则可以使用，只是它不指向具体的变量而已。

```
main(){
int a,b,c,*pmax,*pmin;
printf("input three numbers:\n");
scanf("%d%d%d",&a,&b,&c);
if(a>b){
pmax=&a;
pmin=&b;}
else{
pmax=&b;
pmin=&a;}
if(c>*pmax) pmax=&c;
if(c<*pmin) pmin=&c;
printf("max=%d\nmin=%d\n",*pmax,*pmin);
}
......
```

pmax,pmin 为整型指针变量。

输入提示。

输入三个数字。

如果第一个数字大于第二个数字. . .

指针变量赋值

指针变量赋值

指针变量赋值

指针变量赋值

判断并赋值

判断并赋值

输出结果

```
......
```

数组指针变量的说明和使用

指向数组的指针变量称为数组指针变量。 在讨论数组指针变量的说明和使用之前，我们先明确几个关系。

一个数组是由连续的一块内存单元组成的。 数组名就是这块连续内存单元的首地址。一个数组也是由各个数组元素(下标变量) 组成的。每个数组元素按其类型不同占有几个连续的内存单元。 一个数组元素的首地址也是指它所占有的几个内存单元的首地址。 一个指针变量既可以指向一个数组，也可以指向一个数组元素， 可把数组名或第一个元素的地址赋予它。如要使指针变量指向第 i 号元素可以把 i 元素的首地址赋予它或把数组名加 i 赋予它。

设有实数组 a，指向 a 的指针变量为 pa，从图 6.3 中我们可以看出有以下关系：
pa,a,&a[0]均指向同一单元，它们是数组 a 的首地址，也是 0 号元素 a[0]的首地址。
pa+1,a+1,&a[1]均指向 1 号元素 a[1]。类推可知 a+i,a+i,&a[i]
指向 i 号元素 a[i]。应该说明的是 pa 是变量，而 a,&a[i]都是常量。在编程时应予以注意。

```
main(){
int a[5],i;
for(i=0;i<5;i++){
a[i]=i;
printf("a[%d]=%d\n",i,a[i]);
}
printf("\n");
}
```

主函数
定义一个整型数组和一个整型变量
循环语句
给数组赋值
打印每一个数组的值
.....
输出换行
.....
数组指针变量说明的一般形式为：
类型说明符 * 指针变量名

其中类型说明符表示所指数组的类型。 从一般形式可以看出指向数组的指针变量和指向普通变量的指针变量的说明是相同的。
引入指针变量后，就可以用两种方法来访问数组元素了。

第一种方法为下标法，即用 a[i]形式访问数组元素。 在第四章中介绍数组时都是采用这种方法。

第二种方法为指针法，即采用*(pa+i)形式，用间接访问的方法来访问数组元素。

```
main(){
int a[5],i,*pa;
pa=a;
for(i=0;i<5;i++){
*pa=i;
pa++;
}
pa=a;
for(i=0;i<5;i++){
printf("a[%d]=%d\n",i,*pa);
pa++;
}
}
```

主函数
定义整型数组和指针

将指针 pa 指向数组 a
循环
将变量 i 的值赋给由指针 pa 指向的 a[]的数组单元
将指针 pa 指向 a[]的下一个单元
.....
指针 pa 重新取得数组 a 的首地址
循环
用数组方式输出数组 a 中的所有元素
将指针 pa 指向 a[]的下一个单元
.....
.....
下面，另举一例，该例与上例本意相同，但是实现方式不同。

```
main(){
int a[5],i,*pa=a;
for(i=0;i<5;){
*pa=i;
printf("a[%d]=%d\n",i++,*pa++);
}
}
```

主函数
定义整型数组和指针，并使指针指向数组 a
循环
将变量 i 的值赋给由指针 pa 指向的 a[]的数组单元
用指针输出数组 a 中的所有元素，同时指针 pa 指向 a[]的下一个单元
.....
.....

数组名和数组指针变量作函数参数

在第五章中曾经介绍过用数组名作函数的实参和形参的问题。在学习指针变量之后就更容易理解这个问题了。 数组名就是数组的首地址，实参向形参传送数组名实际上就是传送数组的地址， 形参得到该地址后也指向同一数组。 这就好象同一件物品有两个彼此不同的名称一样。同样，指针变量的值也是地址， 数组指针变量的值即为数组的首地址，当然也可作为函数的参数使用。

```
float aver(float *pa);
main(){
float sco[5],av,*sp;
int i;
sp=sco;
printf("\ninput 5 scores:\n");
for(i=0;i<5;i++) scanf("%f",&sco[i]);
av=aver(sp);
printf("average score is %5.2f",av);
```

```
}
float aver(float *pa)
{
int i;
float av,s=0;
for(i=0;i<5;i++) s=s+*pa++;
av=s/5;
return av;
}
```

指向多维数组的指针变量

本小节以二维数组为例介绍多维数组的指针变量。

一、多维数组地址的表示方法
设有整型二维数组 a[3][4]如下：
0 1 2 3
4 5 6 7
8 9 10 11

设数组 a 的首地址为 1000，各下标变量的首地址及其值如图所示。在第四章中介绍过，C 语言允许把一个二维数组分解为多个一维数组来处理。因此数组 a 可分解为三个一维数组，即 a[0]，a[1]，a[2]。每一个一维数组又含有四个元素。例如 a[0]数组，含有 a[0][0]，a[0][1]，a[0][2]，a[0][3]四个元素。 数组及数组元素的地址表示如下：a 是二维数组名，也是二维数组 0 行的首地址，等于 1000。a[0]是第一个一维数组的数组名和首地址，因此也为 1000。*(a+0)或*a 是与 a[0]等效的， 它表示一维数组 a[0]0 号元素的首地址。 也为 1000。&a[0][0]是二维数组 a 的 0 行 0 列元素首地址，同样是 1000。因此，a，a[0]，*(a+0)，*a，&a[0][0]是相等的。同理，a+1 是二维数组 1 行的首地址，等于 1008。a[1]是第二个一维数组的数组名和首地址，因此也为 1008。 &a[1][0]是二维数组 a 的 1 行 0 列元素地址，也是 1008。因此 a+1,a[1],*(a+1),&a[1][0]是等同的。 由此可得出：a+i，a[i]，*(a+i)，&a[i][0]是等同的。 此外，&a[i]和 a[i]也是等同的。因为在二维数组中不能把&a[i]理解为元素 a[i]的地址，不存在元素 a[i]。

C 语言规定，它是一种地址计算方法，表示数组 a 第 i 行首地址。由此，我们得出：a[i]，&a[i]，*(a+i)和 a+i 也都是等同的。另外，a[0]也
可以看成是 a[0]+0 是一维数组 a[0]的 0 号元素的首地址，而 a[0]+1 则是 a[0]的 1 号元素首地址，由此可得出 a[i]+j 则是一维数组 a[i]的 j 号元素首地址，它等于&a[i][j]。由 a[i]=*(a+i)得 a[i]+j=*(a+i)+j，由于*(a+i)+j 是二维数组 a 的 i 行 j 列元素的首地址。该元素的值等于*(*(a+i)+j)。

```
[Explain]#define PF "%d,%d,%d,%d,%d,\n"
main(){
static int a[3][4]={0,1,2,3,4,5,6,7,8,9,10,11};
printf(PF,a,*a,a[0],&a[0],&a[0][0]);
printf(PF,a+1,*(a+1),a[1],&a[1],&a[1][0]);
```

```
printf(PF,a+2,*(a+2),a[2],&a[2],&a[2][0]);
printf("%d,%d\n",a[1]+1,*(a+1)+1);
printf("%d,%d\n",*(a[1]+1),*(*(a+1)+1));
}
```

二、多维数组的指针变量

把二维数组 a 分解为一维数组 a[0],a[1],a[2]之后，设 p 为指向二维数组的指针变量。可定义为： int (*p)[4] 它表示 p 是一个指针变量，它指向二维数组 a 或指向第一个一维数组 a[0]，其值等于 a,a[0]，或&a[0][0]等。而 p+i 则指向一维数组 a[i]。从前面的分析可得出*(p+i)+j 是二维数组 i 行 j 列的元素的地址，而*(*(p+i)+j)则是 i 行 j 列元素的值。

二维数组指针变量说明的一般形式为：类型说明符 (*指针变量名)[长度] 其中“类型说明符”为所指数组的数据类型。“*”表示其后的变量是指针类型。“长度”表示二维数组分解为多个一维数组时， 一维数组的长度，也就是二维数组的列数。应注意“(*指针变量名)”两边的括号不可少，如缺少括号则表示是指针数组(本章后面介绍)，意义就完全不同了。

```
[Explain]main(){
static int a[3][4]={0,1,2,3,4,5,6,7,8,9,10,11};
int(*p)[4];
int i,j;
p=a;
for(i=0;i<3;i++)
for(j=0;j<4;j++) printf("%2d ",*(*(p+i)+j));
}
```

'Expain 字符串指针变量的说明和使用字符串指针变量的定义说明与指向字符变量的指针变量说明是相同的。只能按对指针变量的赋值不同来区别。 对指向字符变量的指针变量应赋予该字符变量的地址。如： char c,*p=&c;表示 p 是一个指向字符变量 c 的指针变量。而：char *s="C Language";则表示 s 是一个指向字符串的指针变量。把字符串的首地址赋予 s。请看下面一例。

```
main(){
char *ps;
ps="C Language";
printf("%s",ps);
}
```

运行结果为：

```
C Language
```

上例中，首先定义 ps 是一个字符指针变量， 然后把字符串的首地址赋予 ps(应写出整个字符串，以便编译系统把该串装入连续的一块内存单元)，并把首地址送入 ps。程序中的：char *ps;ps="C Language";等效于： char *ps="C Language";输出字符串中 n 个字符后的所有字符。

```
main(){
char *ps="this is a book";
int n=10;
```

```
ps=ps+n;
printf("%s\n",ps);
}
```

运行结果为：

book 在程序中对 ps 初始化时，即把字符串首地址赋予 ps，当 ps= ps+10 之后，ps 指向字符“b”，因此输出为"book"。

```
main(){
char st[20],*ps;
int i;
printf("input a string:\n");
ps=st;
scanf("%s",ps);
for(i=0;ps[i]!='\0';i++)
if(ps[i]=='k'){
printf("there is a 'k' in the string\n");
break;
}
if(ps[i]=='\0') printf("There is no 'k' in the string\n");
}
```

本例是在输入的字符串中查找有无‘k’字符。下面这个例子是将指针变量指向一个格式字符串，用在 printf 函数中，用于输出二维数组的各种地址表示的值。但在 printf 语句中用指针变量 PF 代替了格式串。 这也是程序中常用的方法。

```
main(){
static int a[3][4]={0,1,2,3,4,5,6,7,8,9,10,11};
char *PF;
PF="%d,%d,%d,%d,%d\n";
printf(PF,a,*a,a[0],&a[0],&a[0][0]);
printf(PF,a+1,*(a+1),a[1],&a[1],&a[1][0]);
printf(PF,a+2,*(a+2),a[2],&a[2],&a[2][0]);
printf("%d,%d\n",a[1]+1,*(a+1)+1);
printf("%d,%d\n",*(a[1]+1),*(*(a+1)+1));
}
```

在下例是讲解，把字符串指针作为函数参数的使用。要求把一个字符串的内容复制到另一个字符串中，并且不能使用 strcpy 函数。函数 cprstr 的形参为两个字符指针变量。pss 指向源字符串，pds 指向目标字符串。表达式：

```
(*pds=*pss)!=`\0'
cpystr(char *pss,char *pds){
while((*pds=*pss)!='\0'){
pds++;
pss++; }
}
main(){
char *pa="CHINA",b[10],*pb;
pb=b;
```

```
cpystr(pa,pb);
printf("string a=%s\nstring b=%s\n",pa,pb);
}
```

在上例中，程序完成了两项工作：一是把 pss 指向的源字符复制到 pds 所指向的目标字符中，二是判断所复制的字符是否为`\0'，若是则表明源字符串结束，不再循环。否则，pds 和 pss 都加 1，指向下一字符。在主函数中，以指针变量 pa,pb 为实参，分别取得确定值后调用 cprstr 函数。由于采用的指针变量 pa 和 pss,pb 和 pds 均指向同一字符串，因此在主函数和 cprstr 函数中均可使用这些字符串。也可以把 cprstr 函数简化为以下形式：

```
cprstr(char *pss,char*pds)
{while ((*pds++=*pss++)!=`\0');}
```

即把指针的移动和赋值合并在一个语句中。 进一步分析还可发现`\0'的 ASCⅡ码为 0，对于 while 语句只看表达式的值为非 0 就循环，为 0 则结束循环，因此也可省去“!=`\0'”这一判断部分，而写为以下形式：

```
cprstr (char *pss,char *pds)
{while (*pdss++=*pss++);}
```

表达式的意义可解释为，源字符向目标字符赋值， 移动指针，若所赋值为非 0 则循环，否则结束循环。这样使程序更加简洁。简化后的程序如下所示。

```
cpystr(char *pss,char *pds){
while(*pds++=*pss++);
}
main(){
char *pa="CHINA",b[10],*pb;
pb=b;
cpystr(pa,pb);
printf("string a=%s\nstring b=%s\n",pa,pb);
}
```

使用字符串指针变量与字符数组的区别

用字符数组和字符指针变量都可实现字符串的存储和运算。 但是两者是有区别的。在使用时应注意以下几个问题：

1．字符串指针变量本身是一个变量，用于存放字符串的首地址。而字符串本身是存放在以该首地址为首的一块连续的内存空间中并以‘\0’作为串的结束。字符数组是由于若干个数组元素组成的，它可用来存放整个字符串。

2．对字符数组作初始化赋值，必须采用外部类型或静态类型，如： static char st[]={“C Language”};而对字符串指针变量则无此限制，如： char *ps="C Language";

3．对字符串指针方式 char *ps="C Language";可以写为： char *ps; ps="C Language";而对数组方式：

```
static char st[]={"C Language"};
```

不能写为：

char st[20];st={"C Language"};
而只能对字符数组的各元素逐个赋值。

从以上几点可以看出字符串指针变量与字符数组在使用时的区别，同时也可看出使用指针变量更加方便。前面说过，当一个指针变量在未取得确定地址前使用是危险的，容易引起错误。但是对指针变量直接赋值是可以的。因为 C 系统对指针变量赋值时要给以确定的地址。因此，
char *ps="C Langage";
或者 char *ps;
ps="C Language";都是合法的。

函数指针变量

在C语言中规定，一个函数总是占用一段连续的内存区， 而函数名就是该函数所占内存区的首地址。 我们可以把函数的这个首地址(或称入口地址)赋予一个指针变量， 使该指针变量指向该函数。然后通过指针变量就可以找到并调用这个函数。 我们把这种指向函数的指针变量称为“函数指针变量”。
函数指针变量定义的一般形式为：
类型说明符 (*指针变量名)();
其中“类型说明符”表示被指函数的返回值的类型。“(* 指针变量名)”表示“*”后面的变量是定义的指针变量。 最后的空括号表示指针变量所指的是一个函数。
例如： int (*pf)();
表示 pf 是一个指向函数入口的指针变量，该函数的返回值(函数值)是整型。
下面通过例子来说明用指针形式实现对函数调用的方法。

```
int max(int a,int b){
if(a>b)return a;
else return b;
}
main(){
int max(int a,int b);
int(*pmax)();
int x,y,z;
pmax=max;
printf("input two numbers:\n");
scanf("%d%d",&x,&y);
z=(*pmax)(x,y);
printf("maxmum=%d",z);
}
```

从上述程序可以看出用，函数指针变量形式调用函数的步骤如下：1．先定义函数指针变量，如后一程序中第 9 行 int (*pmax)();定义 pmax 为函数指针变量。

2．把被调函数的入口地址(函数名)赋予该函数指针变量，如程序中第 11 行 pmax=max;

3．用函数指针变量形式调用函数，如程序第 14 行 z=(*pmax)(x,y)； 调用函数的一般形式为： (*指针变量名) (实参表)使用函数指针变量还应注意以下两点：

a．函数指针变量不能进行算术运算，这是与数组指针变量不同的。数组指针变量加减一个整数可使指针移动指向后面或前面的数组元素，而函数指针的移动是毫无意义的。

b．函数调用中"(*指针变量名)"的两边的括号不可少，其中的*不应该理解为求值运算，在此处它只是一种表示符号。

指针型函数

前面我们介绍过，所谓函数类型是指函数返回值的类型。 在C语言中允许一个函数的返回值是一个指针(即地址)， 这种返回指针值的函数称为指针型函数。
定义指针型函数的一般形式为：

```
类型说明符 *函数名(形参表)
{
…… /*函数体*/
}
```

其中函数名之前加了“*”号表明这是一个指针型函数，即返回值是一个指针。类型说明符表示了返回的指针值所指向的数据类型。
如：

```
int *ap(int x,int y)
{
...... /*函数体*/
}
```

表示 ap 是一个返回指针值的指针型函数， 它返回的指针指向一个整型变量。下例中定义了一个指针型函数 day_name，它的返回值指向一个字符串。该函数中定义了一个静态指针数组 name。name 数组初始化赋值为八个字符串，分别表示各个星期名及出错提示。形参 n 表示与星期名所对应的整数。在主函数中， 把输入的整数 i 作为实参， 在 printf 语句中调用 day_name 函数并把 i 值传送给形参 n。day_name 函数中的 return 语句包含一个条件表达式， n 值若大于 7 或小于 1 则把 name[0] 指针返回主函数输出出错提示字符串“Illegal day”。否则返回主函数输出对应的星期名。主函数中的第 7 行是个条件语句，其语义是，如输入为负数(i<0)则中止程序运行退出程序。exit 是一个库函数，exit(1)表示发生错误后退出程序， exit(0)表示正常退出。

应该特别注意的是函数指针变量和指针型函数这两者在写法和意义上的区别。如 int(*p)()和 int *p()是两个完全不同的量。int(*p)()是一个变量说明，说明 p 是一个指向函数入口的指针变量，该函数的返回值是整型量，(*p)的两边的括号不能少。int *p() 则不是变量说明而是函数说明，说明 p 是一个指针型函数，其返回值是一个指向整型量的指针，*p 两边没有括号。作为函数说明， 在括号内最好写入形式参数，这样便于与变量说明区别。对于指针型函数定义， int *p()只是函数头部分，一般还应该有函数体部分。

```
main(){
int i;
```

```
char *day_name(int n);
printf("input Day No:\n");
scanf("%d",&i);
if(i<0) exit(1);
printf("Day No:%2d-->%s\n",i,day_name(i));
}
char *day_name(int n){
static char *name[]={ "Illegal day",
"Monday",
"Tuesday",
"Wednesday",
"Thursday",
"Friday",
"Saturday",
"Sunday"};
return((n<1||n>7) ? name[0] : name[n]);
}
```

本程序是通过指针函数，输入一个 1～7 之间的整数， 输出对应的星期名。指针数组的说明与使用一个数组的元素值为指针则是指针数组。 指针数组是一组有序的指针的集合。指针数组的所有元素都必须是具有相同存储类型和指向相同数据类型的指针变量。

指针数组说明的一般形式为： 类型说明符*数组名[数组长度]

其中类型说明符为指针值所指向的变量的类型。例如： int *pa[3] 表示 pa 是一个指针数组，它有三个数组元素， 每个元素值都是一个指针，指向整型变量。通常可用一个指针数组来指向一个二维数组。 指针数组中的每个元素被赋予二维数组每一行的首地址， 因此也可理解为指向一个一维数组。图 6—6 表示了这种关系。

```
int a[3][3]={1,2,3,4,5,6,7,8,9};
int *pa[3]={a[0],a[1],a[2]};
int *p=a[0];
main(){
int i;
for(i=0;i<3;i++)
printf("%d,%d,%d\n",a[i][2-i],*a[i],*(*(a+i)+i));
for(i=0;i<3;i++)
printf("%d,%d,%d\n",*pa[i],p[i],*(p+i));
}
```

本例程序中，pa 是一个指针数组，三个元素分别指向二维数组 a 的各行。然后用循环语句输出指定的数组元素。其中*a[i]表示 i 行 0 列元素值；*(*(a+i)+i)表示 i 行 i 列的元素值；*pa[i]表示 i 行 0 列元素值；由于 p 与 a[0]相同，故 p[i]表示 0 行 i 列的值；*(p+i)表示 0 行 i 列的值。读者可仔细领会元素值的各种不同的表示方法。 应该注意指针数组和二维数组指针变量的区别。 这两者虽然都可用来表示二维数组，但是其表示方法和意义是不同的。

二维数组指针变量是单个的变量，其一般形式中"(*指针变量名)"两边的括号不可少。而指针数组类型表示的是多个指针(一组有序指针)在一般形式中"*指针数组名"两边不能

有括号。例如： int (*p)[3];表示一个指向二维数组的指针变量。该二维数组的列数为 3 或分解为一维数组的长度为 3。 int *p[3] 表示 p 是一个指针数组，有三个下标变量 p[0]，p[1]，p[2]均为指针变量。

指针数组也常用来表示一组字符串， 这时指针数组的每个元素被赋予一个字符串的首地址。 指向字符串的指针数组的初始化更为简单。例如在例 6.20 中即采用指针数组来表示一组字符串。 其初始化赋值为：

```
char *name[]={"Illagal day",
"Monday",
"Tuesday",
"Wednesday",
"Thursday",
"Friday",
"Saturday",
"Sunday"};
```

完成这个初始化赋值之后，name[0]即指向字符串"Illegal day"，name[1]指?quot;Monday"......。

指针数组也可以用作函数参数。在本例主函数中，定义了一个指针数组 name，并对 name 作了初始化赋值。其每个元素都指向一个字符串。然后又以 name 作为实参调用指针型函数 day name，在调用时把数组名 name 赋予形参变量 name，输入的整数 i 作为第二个实参赋予形参 n。在 day name 函数中定义了两个指针变量 pp1 和 pp2，pp1 被赋予 name[0]的值(即 *name)，pp2 被赋予 name[n]的值即*(name+ n)。由条件表达式决定返回 pp1 或 pp2 指针给主函数中的指针变量 ps。最后输出 i 和 ps 的值。

指针数组作指针型函数的参数

```
main(){
static char *name[]={ "Illegal day",
"Monday",
"Tuesday",
"Wednesday",
"Thursday",
"Friday",
"Saturday",
"Sunday"};
char *ps;
int i;
char *day_name(char *name[],int n);
printf("input Day No:\n");
scanf("%d",&i);
if(i<0) exit(1);
ps=day_name(name,i);
printf("Day No:%2d-->%s\n",i,ps);
```

```
}
char *day_name(char *name[],int n)
{
char *pp1,*pp2;
pp1=*name;
pp2=*(name+n);
return((n<1||n>7)? pp1:pp2);
}
```

下例要求输入 5 个国名并按字母顺序排列后输出。在以前的例子中采用了普通的排序方法，逐个比较之后交换字符串的位置。交换字符串的物理位置是通过字符串复制函数完成的。反复的交换将使程序执行的速度很慢，同时由于各字符串(国名) 的长度不同，又增加了存储管理的负担。 用指针数组能很好地解决这些问题。把所有的字符串存放在一个数组中， 把这些字符数组的首地址放在一个指针数组中，当需要交换两个字符串时， 只须交换指针数组相应两元素的内容(地址)即可，而不必交换字符串本身。程序中定义了两个函数，一个名为 sort 完成排序， 其形参为指

针数组 name，即为待排序的各字符串数组的指针。形参 n 为字符串的个数。另一个函数名为 print，用于排序后字符串的输出，其形参与 sort 的形参相同。主函数 main 中，定义了指针数组 name 并作了初始化赋值。然后分别调用 sort 函数和 print 函数完成排序和输出。值得说明的是在 sort 函数中，对两个字符串比较，采用了 strcmp 函数，strcmp 函数允许参与比较的串以指针方式出现。name[k]和 name[j]均为指针，因此是合法的。字符串比较后需要交换时， 只交换指针数组元素的值，而不交换具体的字符串， 这样将大大减少时间的开销，提高了运行效率。

现编程如下：

```
#include"string.h"
main(){
void sort(char *name[],int n);
void print(char *name[],int n);
static char *name[]={ "CHINA","AMERICA","AUSTRALIA",
"FRANCE","GERMAN"};
int n=5;
sort(name,n);
print(name,n);
}
void sort(char *name[],int n){
char *pt;
int i,j,k;
for(i=0;i<n-1;i++){
k=i;
for(j=i+1;j<n;j++)
if(strcmp(name[k],name[j])>0) k=j;
if(k!=i){
pt=name[i];
name[i]=name[k];
name[k]=pt;
```

```
}
}
}
void print(char *name[],int n){
int i;
for (i=0;i<n;i++) printf("%s\n",name[i]);
}
```

m a i n函数的参数

前面介绍的main函数都是不带参数的。因此main 后的括号都是空括号。实际上，main函数可以带参数，这个参数可以认为是 main 函数的形式参数。C语言规定 main 函数的参数只能有两个，习惯上这两个参数写为argc和argv。因此,main函数的函数头可写为：main(argc,argv) C语言还规定 argc(第一个形参)必须是整型变量,argv(第二个形参)必须是指向字符串的指针数组。加上形参说明后，main函数的函数头应写为：

```
main (argc,argv)
int argv;
char *argv[];或写成：
main (int argc,char *argv[])
```

由于main函数不能被其它函数调用， 因此不可能在程序内部取得实际值。那么，在何处把实参值赋予 main 函数的形参呢? 实际上,main 函数的参数值是从操作系统命令行上获得的。当我们要运行一个可执行文件时，在DOS提示符下键入文件名，再输入实际参数即可把这些实参传送到main的形参中去。

DOS 提示符下命令行的一般形式为： C:\>可执行文件名 参数 参数……; 但是应该特别注意的是，main 的两个形参和命令行中的参数在
位置上不是一一对应的。因为,main的形参只有二个，而命令行中的参数个数原则上未加限制。argc 参数表示了命令行中参数的个数(注意：文件名本身也算一个参数)，argc 的值是在输入命令行时由系统按实际参数的个数自动赋予的。例如有命令行为： C:\>E6 24 BASIC dbase FORTRAN 由于文件名 E6 24 本身也算一个参数，所以共有 4 个参数，因此 argc 取得的值为 4。argv 参数是字符串指针数组，其各元素值为命令行中各字符串(参数均按字符串处理)的首地址。 指针数组的长度即为参数个数。数组元素初值由系统自动赋予。

```
main(int argc,char *argv){
while(argc-->1)
printf("%s\n",*++argv);
}
```

本例是显示命令行中输入的参数如果上例的可执行文件名为e24.exe，存放在A驱动器的盘内。
因此输入的命令行为： C:\>a:e24 BASIC dBASE FORTRAN
则运行结果为：

```
BASIC
dBASE
FORTRAN
```

该行共有 4 个参数，执行 main 时，argc 的初值即为 4。argv 的 4 个元素分为 4 个字符串的首地址。执行 while 语句，每循环一次 argv 值减 1，当 argv 等于 1 时停止循环，共循环三次， 因此共可输出三个参数。在 printf 函数中，由于打印项*++argv 是先加 1 再打印，故第一次打印的是 argv[1]所指的字符串 BASIC。第二、 三次循环分别打印后二个字符串。而参数 e24 是文件名，不必输出。

下例的命令行中有两个参数，第二个参数 20 即为输入的 n 值。在程序中*++argv 的值为字符串“20”，然后用函数"atoi"把它换为整型作为 while 语句中的循环控制变量，输出 20 个偶数。

```
#include"stdlib.h"
main(int argc,char*argv[]){
int a=0,n;
n=atoi(*++argv);
while(n--) printf("%d ",a++*2);
}
```

本程序是从 0 开始输出 n 个偶数。指向指针的指针变量如果一个指针变量存放的又是另一个指针变量的地址， 则称这个指针变量为指向指针的指针变量。

在前面已经介绍过，通过指针访问变量称为间接访问， 简称间访。由于指针变量直接指向变量，所以称为单级间访。 而如果通过指向指针的指针变量来访问变量则构成了二级或多级间访。在 C 语言程序中，对间访的级数并未明确限制， 但是间访级数太多时不容易理解解，也容易出错，因此，一般很少超过二级间访。 指向指针的指针变量说明的一般形式为：

类型说明符** 指针变量名；

例如： int ** pp; 表示 pp 是一个指针变量，它指向另一个指针变量， 而这个指针变量指向一个整型量。下面举一个例子来说明这种关系。

```
main(){
int x,*p,**pp;
x=10;
p=&x;
pp=&p;
printf("x=%d\n",**pp);
}
```

上例程序中 p 是一个指针变量，指向整型量 x；pp 也是一个指针变量， 它指向指针变量 p。通过 pp 变量访问 x 的写法是**pp。程序最后输出 x 的值为 10。通过上例，读者可以学习指向指针的指针变量的说明和使用方法。

下述程序中首先定义说明了指针数组 ps 并作了初始化赋值。又说明了 pps 是一个指向指针的指针变量。在 5 次循环中， pps 分别取得了 ps[0]，ps[1]，ps[2]，ps[3]，ps[4]的地址值(如图 6.10 所示)。再通过这些地址即可找到该字符串。

```
main(){
static char *ps[]={ "BASIC","DBASE","C","FORTRAN",
"PASCAL"};
char **pps;
```

```
int i;
for(i=0;i<5;i++){
pps=ps+i;
printf("%s\n",*pps);
}
}
```

本程序是用指向指针的指针变量编程，输出多个字符串。

第七节　大头娃娃结构体

还是先给一个简单的例子：

```
#include<stdio.h>
  struct stu{
    char *num;
    char *name;
    int score;
    };
    main()
    {
    struct stu a[3];
    int i;
    for(i=0;i<3;i++){
        printf("input NO. %d student's message:\n",i+1);
        scanf("%s%s%d",a[i].num,a[i].name,&a[i].score);}
    printf("your students:\n");
    printf("num\tname\tscore\n");
     for(i=0;i<3;i++)
      printf("%s\t%s\t%d\n",a[i].num,a[i].name,a[i].score);
      getch();
       }
```

这是一个简单的利用结构体实现学生成绩管理，由程序我们知道，结构体数据类型的算法和使用跟其他数据类型一样，对于初学者来说，第一眼看到结构体数据类型的程序就会潜意识的把眼睛投入到struct **{**};和struct ** a,b;上面，从而会发现，结构体的程序不容易让人理解，这是正常的。

我个人认为，结构体其实与其他数据类型没有什么区别，只是他的“头”大一点而已。为什么这样说呢？我们仔细看一下下面一个程序，在与上面的比较一下。

```
#include<stdio.h>
 main()
    {
    int a[3];
    int i;
    for(i=0;i<3;i++){
        printf("input NO. %d student's message:\n",i+1);
        scanf("%d",&a[i]);}
    printf("your students:\n");
    printf("num\tname\tscore\n");
     for(i=0;i<3;i++)
      printf("%d\n",a[i]);
      getch();
      }
```

现在你有一点理解了吧？是的，结构体就是“头”大一点而已，当然，除了这点，结构体还有其自身的性质和使用方法，那么在这一节我们一同详细研究结构体，一同认识一下这个可爱的“大头娃娃”结构体。

结构类型定义和结构变量说明

在实际问题中，一组数据往往具有不同的数据类型。例如， 在学生登记表中，姓名应为字符型；学号可为整型或字符型； 年龄应为整型；性别应为字符型；成绩可为整型或实型。显然不能用一个数组来存放这一组数据。因为数组中各元素的类型和长度都必须一致，以便于编译系统处理。为了解决这个问题，C 语言中给出了另一种构造数据类型——“结构”。 它相当于其它高级语言中的记录。

“结构”是一种构造类型，它是由若干“成员”组成的。 每一个成员可以是一个基本数据类型或者又是一个构造类型。 结构既是一种“构造”而成的数据类型， 那么在说明和使用之前必须先定义它，也就是构造它。如同在说明和调用函数之前要先定义函数一样。

一、结构的定义

定义一个结构的一般形式为：

```
struct 结构名
{
成员表列
};
```

成员表由若干个成员组成， 每个成员都是该结构的一个组成部分。对每个成员也必须作类型说明，其形式为：

```
类型说明符 成员名;
```

成员名的命名应符合标识符的书写规定。例如：

```
struct stu
{
int num;
char name[20];
char sex;
float score;
};
```

在这个结构定义中，结构名为 stu，该结构由 4 个成员组成。 第一个成员为 num，整型变量；第二个成员为 name，字符数组；第三个成员为 sex，字符变量；第四个成员为 score，实型变量。 应注意在括号后的分号是不可少的。结构定义之后，即可进行变量说明。 凡说明为结构 stu 的变量都由上述 4 个成员组成。由此可见， 结构是一种复杂的数据类型，是数目固定，类型不同的若干有序变量的集合。

二、结构类型变量的说明

说明结构变量有以下三种方法。以上面定义的 stu 为例来加以说明。

1. 先定义结构，再说明结构变量。如：

```
struct stu
{
int num;
char name[20];
char sex;
float score;
};
struct stu boy1,boy2;
```

说明了两个变量 boy1 和 boy2 为 stu 结构类型。也可以用宏定义使一个符号常量来表示一个结构类型，例如：

```
#define STU struct stu
STU
{
int num;
char name[20];
char sex;
float score;
};
STU boy1,boy2;
```

2. 在定义结构类型的同时说明结构变量。例如：

```
struct stu
{
int num;
char name[20];
char sex;
float score;
}boy1,boy2;
```

3. 直接说明结构变量。例如：

```
struct
{
int num;
char name[20];
char sex;
float score;
}boy1,boy2;
```

第三种方法与第二种方法的区别在于第三种方法中省去了结构名，而直接给出结构变量。三种方法中说明的 boy1,boy2 变量都具有相同的结构。说明了 boy1,boy2 变量为 stu 类型后，即可向这两个变量中的各个成员赋值。在上述 stu 结构定义中，所有的成员都是基

本数据类型或数组类型。成员也可以又是一个结构， 即构成了嵌套的结构。

```
struct date{
int month;
int day;
int year;
}
struct{
int num;
char name[20];
char sex;
struct date birthday;
float score;
}boy1,boy2;
```

首先定义一个结构 date，由 month(月)、day(日)、year(年) 三个成员组成。 在定义并说明变量 boy1 和 boy2 时， 其中的成员 birthday 被说明为 data 结构类型。成员名可与程序中其它变量同名，互不干扰。结构变量成员的表示方法在程序中使用结构变量时， 往往不把它作为一个整体来使用。

在 ANSI C 中除了允许具有相同类型的结构变量相互赋值以外，一般对结构变量的使用，包括赋值、输入、输出、 运算等都是通过结构变量的成员来实现的。

表示结构变量成员的一般形式是： 结构变量名.成员名 例如：boy1.num 即第一个人的学号 boy2.sex 即第二个人的性别 如果成员本身又是一个结构则必须逐级找到最低级的成员才能使用。例如：boy1.birthday.month 即第一个人出生的月份成员可以在程序中单独使用，与普通变量完全相同。

结构变量的赋值

前面已经介绍，结构变量的赋值就是给各成员赋值。 可用输入语句或赋值语句来完成。
给结构变量赋值并输出其值。

```
main(){
struct stu
{
int num;
char *name;
char sex;
float score;
} boy1,boy2;
boy1.num=102;
boy1.name="Zhang ping";
printf("input sex and score\n");
scanf("%c %f",&boy1.sex,&boy1.score);
boy2=boy1;
```

```
printf("Number=%d\nName=%s\n",boy2.num,boy2.name);
printf("Sex=%c\nScore=%f\n",boy2.sex,boy2.score);
}
```

本程序中用赋值语句给 num 和 name 两个成员赋值，name 是一个字符串指针变量。用 scanf 函数动态地输入 sex 和 score 成员值，然后把 boy1 的所有成员的值整体赋予 boy2。最后分别输出 boy2 的各个成员值。本例表示了结构变量的赋值、输入和输出的方法。

结构变量的初始化

如果结构变量是全局变量或为静态变量， 则可对它作初始化赋值。对局部或自动结构变量不能作初始化赋值。

外部结构变量初始化。

```
struct stu /*定义结构*/
{
int num;
char *name;
char sex;
float score;
} boy2,boy1={102,"Zhang ping",'M',78.5};
main()
{
boy2=boy1;
printf("Number=%d\nName=%s\n",boy2.num,boy2.name);
printf("Sex=%c\nScore=%f\n",boy2.sex,boy2.score);
}
```

```
struct stu
{
int num;
char *name;
char sex;
float score;
}boy2,boy1={102,"Zhang ping",'M',78.5};
main()
{
boy2=boy1;
……
}
```

本例中，boy2,boy1 均被定义为外部结构变量，并对 boy1 作了初始化赋值。在 main 函数中，把 boy1 的值整体赋予 boy2， 然后用两个 printf 语句输出 boy2 各成员的值。

静态结构变量初始化。

```
main()
{
static struct stu /*定义静态结构变量*/
```

```
{
int num;
char *name;
char sex;
float score;
}boy2,boy1={102,"Zhang ping",'M',78.5};
boy2=boy1;
printf("Number=%d\nName=%s\n",boy2.num,boy2.name);
printf("Sex=%c\nScore=%f\n",boy2.sex,boy2.score);
}
static struct stu
{
int num;
char *name;
char sex;
float score;
}boy2,boy1={102,"Zhang ping",'M',78.5};
```

本例是把 boy1，boy2 都定义为静态局部的结构变量， 同样可以作初始化赋值。

结构数组

数组的元素也可以是结构类型的。 因此可以构成结构型数组。结构数组的每一个元素都是具有相同结构类型的下标结构变量。 在实际应用中，经常用结构数组来表示具有相同数据结构的一个群体。如一个班的学生档案，一个车间职工的工资表等。
结构数组的定义方法和结构变量相似，只需说明它为数组类型即可。例如：

```
struct stu
{
int num;
char *name;
char sex;
float score;
}boy[5];
```

定义了一个结构数组 boy1,共有 5 个元素,boy[0]～boy[4]。每个数组元素都具有 struct stu 的结构形式。 对外部结构数组或静态结构数组可以作初始化赋值，例如：

```
struct stu
{
int num;
char *name;
char sex;
float score;
}boy[5]={
{101,"Li ping","M",45},
```

```
{102,"Zhang ping","M",62.5},
{103,"He fang","F",92.5},
{104,"Cheng ling","F",87},
{105,"Wang ming","M",58};
}
```

当对全部元素作初始化赋值时，也可不给出数组长度。

计算学生的平均成绩和不及格的人数。

```
struct stu
{
int num;
char *name;
char sex;
float score;
}boy[5]={
{101,"Li ping",'M',45},
{102,"Zhang ping",'M',62.5},
{103,"He fang",'F',92.5},
{104,"Cheng ling",'F',87},
{105,"Wang ming",'M',58},
};
main()
{
int i,c=0;
float ave,s=0;
for(i=0;i<5;i++)
{
s+=boy[i].score;
if(boy[i].score<60) c+=1;
}
printf("s=%f\n",s);
ave=s/5;
printf("average=%f\ncount=%d\n",ave,c);
}
```

本例程序中定义了一个外部结构数组 boy，共 5 个元素， 并作了初始化赋值。在 main 函数中用 for 语句逐个累加各元素的 score 成员值存于 s 之中，如 score 的值小于 60(不及格)即计数器 C 加 1， 循环完毕后计算平均成绩，并输出全班总分，平均分及不及格人数。

建立同学通讯录

```
#include"stdio.h"
#define NUM 3
struct mem
{
char name[20];
char phone[10];
```

```
};
main()
{
struct mem man[NUM];
int i;
for(i=0;i<NUM;i++)
{
printf("input name:\n");
gets(man[i].name);
printf("input phone:\n");
gets(man[i].phone);
}
printf("name\t\t\tphone\n\n");
for(i=0;i<NUM;i++)
printf("%s\t\t\t%s\n",man[i].name,man[i].phone);
}
```

本程序中定义了一个结构 mem,它有两个成员 name 和 phone 用来表示姓名和电话号码。在主函数中定义 man 为具有 mem 类型的结构数组。在 for 语句中，用 gets 函数分别输入各个元素中两个成员的值。然后又在 for 语句中用 printf 语句输出各元素中两个成员值。

结构指针变量

结构指针变量的说明和使用一个指针变量当用来指向一个结构变量时， 称之为结构指针变量。
结构指针变量中的值是所指向的结构变量的首地址。 通过结构指针即可访问该结构变量，这与数组指针和函数指针的情况是相同的。结构指针变量说明的一般形式为：
struct 结构名*结构指针变量名
例如，在前面的例 7.1 中定义了 stu 这个结构， 如要说明一个指向 stu 的指针变量 pstu，可写为：
struct stu *pstu;

当然也可在定义 stu 结构时同时说明 pstu。与前面讨论的各类指针变量相同，结构指针变量也必须要先赋值后才能使用。赋值是把结构变量的首地址赋予该指针变量， 不能把结构名赋予该指针变量。如果 boy 是被说明为 stu 类型的结构变量，则： pstu=&boy 是正确的，而： pstu=&stu 是错误的。

结构名和结构变量是两个不同的概念，不能混淆。 结构名只能表示一个结构形式，编译系统并不对它分配内存空间。 只有当某变量被说明为这种类型的结构时，才对该变量分配存储空间。 因此上面&stu 这种写法是错误的，不可能去取一个结构名的首地址。 有了结构指针变量，就能更方便地访问结构变量的各个成员。

其访问的一般形式为： (*结构指针变量).成员名 或为：
结构指针变量->成员名

例如： (*pstu).num 或者： pstu->num
应该注意(*pstu)两侧的括号不可少， 因为成员符“.”的优先级高于“*”。如去掉括号写作*pstu.num 则等效于*(pstu.num)，这样，意义就完全不对了。 下面通过例子来说明结构指针变量的具体说明和使用方法。

```
struct stu
{
int num;
char *name;
char sex;
float score;
} boy1={102,"Zhang ping",'M',78.5},*pstu;
main()
{
pstu=&boy1;
printf("Number=%d\nName=%s\n",boy1.num,boy1.name);
printf("Sex=%c\nScore=%f\n\n",boy1.sex,boy1.score);
printf("Number=%d\nName=%s\n",(*pstu).num,(*pstu).name);
printf("Sex=%c\nScore=%f\n\n",(*pstu).sex,(*pstu).score);
printf("Number=%d\nName=%s\n",pstu->num,pstu->name);
printf("Sex=%c\nScore=%f\n\n",pstu->sex,pstu->score);
}
```

本例程序定义了一个结构 stu，定义了 stu 类型结构变量 boy1 并作了初始化赋值，还定义了一个指向 stu 类型结构的指针变量 pstu。在 main 函数中，pstu 被赋予 boy1 的地址，因此 pstu 指向 boy1 。然后在 printf 语句内用三种形式输出 boy1 的各个成员值。 从运行结果可以看出：
结构变量.成员名
(*结构指针变量).成员名
结构指针变量->成员名

这三种用于表示结构成员的形式是完全等效的。结构数组指针变量结构指针变量可以指向一个结构数组， 这时结构指针变量的值是整个结构数组的首地址。 结构指针变量也可指向结构数组的一个元素，这时结构指针变量的值是该结构数组元素的首地址。设 ps 为指向结构数组的指针变量，则 ps 也指向该结构数组的 0 号元素，ps+1 指向 1 号元素，ps+i 则指向 i 号元素。 这与普通数组的情况是一致的。

用指针变量输出结构数组。

```
struct stu
{
int num;
char *name;
char sex;
float score;
}boy[5]={
{101,"Zhou ping",'M',45},
```

```
{102,"Zhang ping",'M',62.5},
{103,"Liou fang",'F',92.5},
{104,"Cheng ling",'F',87},
{105,"Wang ming",'M',58},
};
main()
{
struct stu *ps;
printf("No\tName\t\t\tSex\tScore\t\n");
for(ps=boy;ps<boy+5;ps++)
printf("%d\t%s\t\t%c\t%f\t\n",ps->num,ps->name,ps->sex,ps->
score);
}
```

在程序中，定义了 stu 结构类型的外部数组 boy 并作了初始化赋值。在 main 函数内定义 ps 为指向 stu 类型的指针。在循环语句 for 的表达式 1 中，ps 被赋予 boy 的首地址，然后循环 5 次，输出 boy 数组中各成员值。 应该注意的是， 一个结构指针变量虽然可以用来访问结构变量或结构数组元素的成员，但是，不能使它指向一个成员。 也就是说不允许取一个成员的地址来赋予它。因此，下面的赋值是错误的。 ps=&boy[1].sex;而只能是：ps=boy;(赋予数组首地址)或者是：ps=&boy[0];(赋予 0 号元素首地址)

结构指针变量作函数参数

在 ANSI C 标准中允许用结构变量作函数参数进行整体传送。 但是这种传送要将全部成员逐个传送， 特别是成员为数组时将会使传送的时间和空间开销很大，严重地降低了程序的效率。 因此最好的办法就是使用指针，即用指针变量作函数参数进行传送。 这时由实参传向形参的只是地址，从而减少了时间和空间的开销。

计算一组学生的平均成绩和不及格人数。用结构指针变量作函数参数编程。

```
struct stu
{
int num;
char *name;
char sex;
float score;}boy[5]={
{101,"Li ping",'M',45},
{102,"Zhang ping",'M',62.5},
{103,"He fang",'F',92.5},
{104,"Cheng ling",'F',87},
{105,"Wang ming",'M',58},
};
main()
{
struct stu *ps;
void ave(struct stu *ps);
```

```
ps=boy;
ave(ps);
}
void ave(struct stu *ps)
{
int c=0,i;
float ave,s=0;
for(i=0;i<5;i++,ps++)
{
s+=ps->score;
if(ps->score<60) c+=1;
}
printf("s=%f\n",s);
ave=s/5;
printf("average=%f\ncount=%d\n",ave,c);
}
```

本程序中定义了函数 ave，其形参为结构指针变量 ps。boy 被定义为外部结构数组，因此在整个源程序中有效。在 main 函数中定义说明了结构指针变量 ps，并把 boy 的首地址赋予它，使 ps 指向 boy 数组。然后以 ps 作实参调用函数 ave。在函数 ave 中完成计算平均成绩和统计不及格人数的工作并输出结果。由于本程序全部采用指针变量作运算和处理，故速度更快，程序效率更高。.

topoic=动态存储分配

在数组一章中，曾介绍过数组的长度是预先定义好的， 在整个程序中固定不变。C 语言中不允许动态数组类型。例如： int n;scanf("%d",&n);int a[n]; 用变量表示长度，想对数组的大小作动态说明， 这是错误的。但是在实际的编程中，往往会发生这种情况， 即所需的内存空间取决于实际输入的数据，而无法预先确定。对于这种问题， 用数组的办法很难解决。为了解决上述问题，C 语言提供了一些内存管理函数，这些内存管理函数可以按需要动态地分配内存空间， 也可把不再使用的空间回收待用，为有效地利用内存资源提供了手段。 常用的内存管理函数有以下三个：

1.分配内存空间函数 malloc
调用形式： (类型说明符*) malloc (size) 功能：在内存的动态存储区中分配一块长度为"size" 字节的连续区域。函数的返回值为该区域的首地址。 “类型说明符”表示把该区域用于何种数据类型。(类型说明符*)表示把返回值强制转换为该类型指针。“size”是一个无符号数。例如： pc=(char *) malloc (100); 表示分配 100 个字节的内存空间，并强制转换为字符数组类型， 函数的返回值为指向该字符数组的指针， 把该指针赋予指针变量 pc。

2.分配内存空间函数 calloc
calloc 也用于分配内存空间。调用形式： (类型说明符*)calloc(n,size) 功能：在内存动态存储区中分配 n 块长度为“size”字节的连续区域。函数的返回值为该区域的首地址。(类

型说明符*)用于强制类型转换。calloc 函数与 malloc 函数的区别仅在于一次可以分配 n 块区域。例如：ps=(struet stu*) calloc(2,sizeof (struct stu))；其中的 sizeof(struct stu)是求 stu 的结构长度。因此该语句的意思是：按 stu 的长度分配 2 块连续区域，强制转换为 stu 类型，并把其首地址赋予指针变量 ps。

3.释放内存空间函数 free

调用形式： free(void*ptr)；功能：释放 ptr 所指向的一块内存空间，ptr 是一个任意类型的指针变量，它指向被释放区域的首地址。被释放区应是由 malloc 或 calloc 函数所分配的区域：分配一块区域，输入一个学生数据。

```
main()
{
struct stu
{
int num;
char *name;
char sex;
float score;
} *ps;
ps=(struct stu*)malloc(sizeof(struct stu));
ps->num=102;
ps->name="Zhang ping";
ps->sex='M';
ps->score=62.5;
printf("Number=%d\nName=%s\n",ps->num,ps->name);
printf("Sex=%c\nScore=%f\n",ps->sex,ps->score);
free(ps);
}
```

本例中，定义了结构 stu，定义了 stu 类型指针变量 ps。 然后分配一块 stu 大内存区，并把首地址赋予 ps，使 ps 指向该区域。再以 ps 为指向结构的指针变量对各成员赋值，并用 printf 输出各成员值。最后用 free 函数释放 ps 指向的内存空间。 整个程序包含了申请内存空间、使用内存空间、释放内存空间三个步骤， 实现存储空间的动态分配。链表的概念在例中采用了动态分配的办法为一个结构分配内存空间。每一次分配一块空间可用来存放一个学生的数据， 我们可称之为一个结点。有多少个学生就应该申请分配多少块内存空间， 也就是说要建立多少个结点。当然用结构数组也可以完成上述工作， 但如果预先不能准确把握学生人数，也就无法确定数组大小。 而且当学生留级、退学之后也不能把该元素占用的空间从数组中释放出来。 用动态存储的方法可以很好地解决这些问题。 有一个学生就分配一个结点，无须预先确定学生的准确人数，某学生退学， 可删去该结点，并释放该结点占用的存储空间。从而节约了宝贵的内存资源。 另一方面，用数组的方法必须占用一块连续的内存区域。 而使用动态分配时，每个结点之间可以是不连续的(结点内是连续的)。结点之间的联系可以用指针实现。 即在结点结构中定义一个成员项用来存放下一结点的首地址，这个用于存放地址的成员，常把它称为指针域。可在第一个结点的指针域内存入第二个结点的首地址， 在第二个结点的指针域内又存放第三个结点的首地址， 如此串连下去直到最后一个结点。最后一个结点因无后续结点连接，其指针域可赋为 0。这样一种连接方式，在数据结构中称为“链表”。

第 0 个结点称为头结点， 它存放有第一个结点的首地址，它没有数据，只是一个指针变量。 以下的每个结点都分为两个域，一个是数据域，存放各种实际的数据，如学号 num，姓名 name，性别 sex 和成绩 score 等。另一个域为指针域， 存放下一结点的首地址。链表中的每一个结点都是同一种结构类型。例如， 一个存放学生学号和成绩的结点应为以下结构：

```
struct stu
{ int num;
int score;
struct stu *next;
}
```

前两个成员项组成数据域，后一个成员项 next 构成指针域， 它是一个指向 stu 类型结构的指针变量。链表的基本操作对链表的主要操作有以下几种：

1.建立链表；
2.结构的查找与输出；
3.插入一个结点；
4.删除一个结点；

下面通过例题来说明这些操作。

建立一个三个结点的链表，存放学生数据。 为简单起见， 我们假定学生数据结构中只有学号和年龄两项。

可编写一个建立链表的函数 creat。程序如下：

```
#define NULL 0
#define TYPE struct stu
#define LEN sizeof (struct stu)
struct stu
{
int num;
int age;
struct stu *next;
};
TYPE *creat(int n)
{
struct stu *head,*pf,*pb;
int i;
for(i=0;i<n;i++)
{
pb=(TYPE*) malloc(LEN);
printf("input Number and Age\n");
scanf("%d%d",&pb->num,&pb->age);
if(i==0)
pf=head=pb;
else pf->next=pb;
pb->next=NULL;
pf=pb;
```

```
}
return(head);
}
```

在函数外首先用宏定义对三个符号常量作了定义。这里用 TYPE 表示 struct stu，用 LEN 表示 sizeof(struct stu)主要的目的是为了在以下程序内减少书写并使阅读更加方便。结构 stu 定义为外部类型，程序中的各个函数均可使用该定义。

creat 函数用于建立一个有 n 个结点的链表，它是一个指针函数，它返回的指针指向 stu 结构。在 creat 函数内定义了三个 stu 结构的指针变量。head 为头指针，pf 为指向两相邻结点的前一结点的指针变量。pb 为后一结点的指针变量。在 for 语句内，用 malloc 函数建立长度与 stu 长度相等的空间作为一结点，首地址赋予 pb。然后输入结点数据。如果当前结点为第一结点(i==0)，则把 pb 值 (该结点指针)赋予 head 和 pf。如非第一结点，则把 pb 值赋予 pf 所指结点的指针域成员 next。而 pb 所指结点为当前的最后结点，其指针域赋 NULL。 再把 pb 值赋予 pf 以作下一次循环准备。

creat 函数的形参 n，表示所建链表的结点数，作为 for 语句的循环次数。

写一个函数，在链表中按学号查找该结点。

```
TYPE * search (TYPE *head,int n)
{
TYPE *p;
int i;
p=head;
while (p->num!=n && p->next!=NULL)
p=p->next; /* 不是要找的结点后移一步*/
if (p->num==n) return (p);
if (p->num!=n&& p->next==NULL)
printf ("Node %d has not been found!\n",n
}
```

本函数中使用的符号常量 TYPE 与例 7.10 的宏定义相同，等于 struct stu。函数有两个形参，head 是指向链表的指针变量，n 为要查找的学号。进入 while 语句，逐个检查结点的 num 成员是否等于 n，如果不等于 n 且指针域不等于 NULL(不是最后结点)则后移一个结点，继续循环。如找到该结点则返回结点指针。 如循环结束仍未找到该结点则输出“未找到”的提示信息。

写一个函数，删除链表中的指定结点。删除一个结点有两种情况：

1. 被删除结点是第一个结点。这种情况只需使 head 指向第二个结点即可。即 head=pb->next。其过程如图 7.5 所示。

2. 被删结点不是第一个结点，这种情况使被删结点的前一结点指向被删结点的后一结点即可。即 pf->next=pb->next。其过程如图 7.6 所示。

函数编程如下：

```
TYPE * delete(TYPE * head,int num)
{
TYPE *pf,*pb;
if(head==NULL) /*如为空表， 输出提示信息*/
```

```
{ printf("\nempty list!\n");
goto end;}
pb=head;
while (pb->num!=num && pb->next!=NULL)
/*当不是要删除的结点，而且也不是最后一个结点时，继续循环*/
{pf=pb;pb=pb->next;}/*pf 指向当前结点，pb 指向下一结点*/
if(pb->num==num)
{if(pb==head) head=pb->next;
/*如找到被删结点，且为第一结点，则使 head 指向第二个结点，
否则使 pf 所指结点的指针指向下一结点*/
else pf->next=pb->next;
free(pb);
printf("The node is deleted\n");}
else
printf("The node not been foud!\n");
end:
return head;
}
```

函数有两个形参，head 为指向链表第一结点的指针变量，num 删结点的学号。 首先判断链表是否为空，为空则不可能有被删结点。若不为空，则使 pb 指针指向链表的第一个结点。进入 while 语句后逐个查找被删结点。找到被删结点之后再看是否为第一结点，若是则使 head 指向第二结点(即把第一结点从链中删去)，否则使被删结点的前一结点(pf 所指)指向被删结点的后一结点(被删结点的指针域所指)。如若循环结束未找到要删的结点， 则输出“未找到”的提示信息。最后返回 head 值。

写一个函数，在链表中指定位置插入一个结点。在一个链表的指定位置插入结点， 要求链表本身必须是已按某种规律排好序的。例如，在学生数据链表中， 要求学号顺序插入一个结点。设被插结点的指针为 pi。 可在三种不同情况下插入。

1．原表是空表，只需使 head 指向被插结点即可。

2．被插结点值最小，应插入第一结点之前。这种情况下使 head 指向被插结点，被插结点的指针域指向原来的第一结点则可。即：pi->next=pb;

head=pi;

3．在其它位置插入，这种情况下，使插入位置的前一结点的指针域指向被插结点，使被插结点的指针域指向插入位置的后一结点。即为：pi->next=pb;pf->next=pi;

4．在表末插入。这种情况下使原表末结点指针域指向被插结点，被插结点指针域置为 NULL。即：

```
pb->next=pi;
pi->next=NULL;  TYPE * insert(TYPE * head,TYPE *pi)
{
TYPE *pf,*pb;
pb=head;
if(head==NULL) /*空表插入*/
(head=pi;
pi->next=NULL;}
```

```
else
{
while((pi->num>pb->num)&&(pb->next!=NULL))
{pf=pb;
pb=pb->next; }/*找插入位置*/
if(pi->num<=pb->num)
{if(head==pb)head=pi;/*在第一结点之前插入*/
else pf->next=pi;/*在其它位置插入*/
pi->next=pb; }
else
{pb->next=pi;
pi->next=NULL;} /*在表末插入*/
}
return head;}
```

本函数有两个形参均为指针变量，head 指向链表，pi 指向被插结点。函数中首先判断链表是否为空，为空则使 head 指向被插结点。表若不空，则用 while 语句循环查找插入位置。找到之后再判断是否在第一结点之前插入，若是则使 head 指向被插结点被插结点指针域指向原第一结点，否则在其它位置插入， 若插入的结点大于表中所有结点，则在表末插入。本函数返回一个指针， 是链表的头指针。 当插入的位置在第一个结点之前时， 插入的新结点成为链表的第一个结点，因此 head 的值也有了改变， 故需要把这个指针返回主调函数。

将以上建立链表，删除结点，插入结点的函数组织在一起，再建一个输出全部结点的函数，然后用 main 函数调用它们。

```
#define NULL 0
#define TYPE struct stu
#define LEN sizeof(struct stu)
struct stu
{
int num;
int age;
struct stu *next;
};
TYPE * creat(int n)
{
struct stu *head,*pf,*pb;
int i;
for(i=0;i<n;i++)
{
pb=(TYPE *)malloc(LEN);
printf("input Number and Age\n");
scanf("%d%d",&pb->num,&pb->age);
if(i==0)
pf=head=pb;
else pf->next=pb;
```

```
pb->next=NULL;
pf=pb;
}
return(head);
}
TYPE * delete(TYPE * head,int num)
{
TYPE *pf,*pb;
if(head==NULL)
{ printf("\nempty list!\n");
goto end;}
pb=head;
while (pb->num!=num && pb->next!=NULL)
{pf=pb;pb=pb->next;}
if(pb->num==num)
{ if(pb==head) head=pb->next;
else pf->next=pb->next;
printf("The node is deleted\n"); }
else
free(pb);
printf("The node not been found!\n");
end:
return head;
}
TYPE * insert(TYPE * head,TYPE * pi)
{
TYPE *pb ,*pf;
pb=head;
if(head==NULL)
{ head=pi;
pi->next=NULL; }
else
{
while((pi->num>pb->num)&&(pb->next!=NULL))
{ pf=pb;
pb=pb->next; }
if(pi->num<=pb->num)
{ if(head==pb) head=pi;
else pf->next=pi;
pi->next=pb; }
else
{ pb->next=pi;
pi->next=NULL; }
}
```

```
return head;
}
void print(TYPE * head)
{
printf("Number\t\tAge\n");
while(head!=NULL)
{
printf("%d\t\t%d\n",head->num,head->age);
head=head->next;
}
}
main()
{
TYPE * head,*pnum;
int n,num;
printf("input number of node: ");
scanf("%d",&n);
head=creat(n);
print(head);
printf("Input the deleted number: ");
scanf("%d",&num);
head=delete(head,num);
print(head);
printf("Input the inserted number and age: ");
pnum=(TYPE *)malloc(LEN);
scanf("%d%d",&pnum->num,&pnum->age);
head=insert(head,pnum);
print(head);
}
```

本例中，print 函数用于输出链表中各个结点数据域值。函数的形参 head 的初值指向链表第一个结点。在 while 语句中，输出结点值后，head 值被改变，指向下一结点。若保留头指针 head， 则应另设一个指针变量，把 head 值赋予它，再用它来替代 head。在 main 函数中，n 为建立结点的数目， num 为待删结点的数据域值；head 为指向链表的头指针，pnum 为指向待插结点的指针。 main 函数中各行的意义是：

第六行输入所建链表的结点数；

第七行调 creat 函数建立链表并把头指针返回给 head；

第八行调 print 函数输出链表；

第十行输入待删结点的学号；

第十一行调 delete 函数删除一个结点；

第十二行调 print 函数输出链表；

第十四行调 malloc 函数分配一个结点的内存空间， 并把其地址赋予 pnum;

第十五行输入待插入结点的数据域值；

第十六行调 insert 函数插入 pnum 所指的结点；

第十七行再次调 print 函数输出链表。

从运行结果看，首先建立起 3 个结点的链表，并输出其值；再删 103 号结点，只剩下 105，108 号结点；又输入 106 号结点数据， 插入后链表中的结点为 105，106，108。联合 “联合”也是一种构造类型的数据结构。在一个“联合”内可以定义多种不同的数据类型，一个被说明为该“联合”类型的变量中，允许装入该“联合”所定义的任何一种数据。 这在前面的各种数据类型中都是办不到的。例如， 定义为整型的变量只能装入整型数据，定义为实型的变量只能赋予实型数据。

联合

在实际问题中有很多这样的例子。 例如在学校的教师和学生中填写以下表格： 姓 名 年 龄 职 业 单位 “职业”一项可分为“教师”和“学生”两类。对“单位”一项学生应填入班级编号，教师应填入某系某教研室。 班级可用整型量表示，教研室只能用字符类型。要求把这两种类型不同的数据都填入“单位”这个变量中， 就必须把“单位”定义为包含整型和字符型数组这两种类型的“联合”。

“联合”与“结构”有一些相似之处。但两者有本质上的不同。在结构中各成员有各自的内存空间， 一个结构变量的总长度是各成员长度之和。而在“联合”中，各成员共享一段内存空间， 一个联合变量的长度等于各成员中最长的长度。应该说明的是， 这里所谓的共享不是指把多个成员同时装入一个联合变量内，而是指该联合变量可被赋予任一成员值，但每次只能赋一种值， 赋入新值则冲去旧值。如前面介绍的“单位”变量， 如定义为一个可装入“班级”或“教研室”的联合后，就允许赋予整型值（班级)或字符串（教研室)。要么赋予整型值，要么赋予字符串，不能把两者同时赋予它。联合类型的定义和联合变量的说明一个联合类型必须经过定义之后， 才能把变量说明为该联合类型。

一、联合的定义

定义一个联合类型的一般形式为：

```
union 联合名
{
成员表
};
```

成员表中含有若干成员，成员的一般形式为： 类型说明符 成员名 成员名的命名应符合标识符的规定。

例如：

```
union perdata
{
int class;
char office[10];
};
```

定义了一个名为 perdata 的联合类型，它含有两个成员，一个为整型，成员名为 class；另一个为字符数组，数组名为 office。联合定义之后，即可进行联合变量说明，被说明为 perdata 类型的变量，可以存放整型量 class 或存放字符数组 office。

二、联合变量的说明

联合变量的说明和结构变量的说明方式相同， 也有三种形式。即先定义，再说明；定义同时说明和直接说明。以 perdata 类型为例，说明如下：

```
union perdata
{
int class;
char officae[10];
};
union perdata a,b; /*说明 a,b 为 perdata 类型*/
```

或者可同时说明为：

```
union perdata
{ int class;
char office[10]; }a,b;或直接说明为： union
{ int class;
char office[10]; }a,b
```

经说明后的 a,b 变量均为 perdata 类型。 a,b 变量的长度应等于 perdata 的成员中最长的长度， 即等于 office 数组的长度，共 10 个字节。a,b 变量如赋予整型值时，只使用了 2 个字节，而赋予字符数组时，可用 10 个字节。

联合变量的赋值和使用

对联合变量的赋值，使用都只能是对变量的成员进行。 联合变量的成员表示为： 联合变量名.成员名 例如，a 被说明为 perdata 类型的变量之后，可使用 a.class a.office 不允许只用联合变量名作赋值或其它操作。 也不允许对联合变量作初始化赋值，赋值只能在程序中进行。还要再强调说明的是，一个联合变量， 每次只能赋予一个成员值。换句话说，一个联合变量的值就是联合变员的某一个成员值。

设有一个教师与学生通用的表格，教师数据有姓名，年龄，职业，教研室四项。学生有姓名，年龄，职业，班级四项。

编程输入人员数据， 再以表格输出。

```
main()
{
struct
{
char name[10];
int age;
char job;
union
{
int class;
char office[10];
} depa;
```

```
}body[2];
int n,i;
for(i=0;i<2;i++)
{
printf("input name,age,job and department\n");
scanf("%s %d %c",body[i].name,&body[i].age,&body[i].job);
if(body[i].job=='s')
scanf("%d",&body[i].depa.class);
else
scanf("%s",body[i].depa.office);
}
printf("name\tage job class/office\n");
for(i=0;i<2;i++)
{
if(body[i].job=='s')
printf("%s\t%3d %3c %d\n",body[i].name,body[i].age
,body[i].job,body[i].depa.class);
else
printf("%s\t%3d %3c %s\n",body[i].name,body[i].age,
body[i].job,body[i].depa.office);
}
}
```

本例程序用一个结构数组 body 来存放人员数据， 该结构共有四个成员。其中成员项 depa 是一个联合类型， 这个联合又由两个成员组成，一个为整型量 class，一个为字符数组 office。在程序的第一个 for 语句中，输入人员的各项数据，先输入结构的前三个成员 name,age 和 job，然后判别 job 成员项，如为"s"则对联合 depa·class 输入(对学生赋班级编号)否则对 depa·office 输入(对教师赋教研组名)。

在用 scanf 语句输入时要注意，凡为数组类型的成员，无论是结构成员还是联合成员，在该项前不能再加"&"运算符。如程序第 18 行中
body[i].name 是一个数组类型，第 22 行中的 body[i].depa.office 也是数组类型，因此在这两项之间不能加"&"运算符。程序中的第二个 for 语句用于输出各成员项的值。

第八节　枚举、位运算

在实际问题中， 有些变量的取值被限定在一个有限的范围内。例如，一个星期内只有七天，一年只有十二个月， 一个班每周有六门课程等等。如果把这些量说明为整型， 字符型或其它类型显然是不妥当的。 为此，C 语言提供了一种称为“枚举”的类型。在“枚举”类型的定义中列举出所有可能的取值， 被说明为该“枚举”类型的变量取值不能超过定义的范围。应该说明的是， 枚举类型是一种基本数据类型，而不是一种构造类型， 因为它不能再分解为任何基本类型。

枚举类型的定义和枚举变量的说明

一、枚举的定义枚举类型定义的一般形式为：

```
enum 枚举名
{ 枚举值表 };
```

在枚举值表中应罗列出所有可用值。这些值也称为枚举元素。

例如： enum weekday

```
{ sun,mou,tue,wed,thu,fri,sat };
```

该枚举名为 weekday，枚举值共有 7 个，即一周中的七天。 凡被说明为 weekday 类型变量的取值只能是七天中的某一天。

二、枚举变量的说明 如同结构和联合一样，枚举变量也可用不同的方式说明， 即先定义后说明，同时定义说明或直接说明。设有变量 a,b,c 被说明为上述的 weekday，可采用下述任一种方式：

```
enum weekday
{
......
};
enum weekday a,b,c;或者为： enum weekday
{
......
}a,b,c;或者为： enum
{
......
}a,b,c;
```

枚举类型变量的赋值和使用

枚举类型在使用中有以下规定：

1．枚举值是常量，不是变量。不能在程序中用赋值语句再对它赋值。例如对枚举 weekday 的元素再作以下赋值： sun=5;mon=2;sun=mon; 都是错误的。

2．枚举元素本身由系统定义了一个表示序号的数值，从 0 开始顺序定义为 0，1，2…。如在 weekday 中，sun 值为 0，mon 值为 1， …,sat 值为 6。

```
main(){
enum weekday
{ sun,mon,tue,wed,thu,fri,sat } a,b,c;
a=sun;
b=mon;
c=tue;
printf("%d,%d,%d",a,b,c);
}
```

3．只能把枚举值赋予枚举变量，不能把元素的数值直接赋予枚举变量。如： a=sum;b=mon; 是正确的。而： a=0;b=1; 是错误的。如一定要把数值赋予枚举变量，则必须用强制类型转换，如： a=(enum weekday)2;其意义是将顺序号为 2 的枚举元素赋予枚举变量 a，相当于：a=tue; 还应该说明的是枚举元素不是字符常量也不是字符串常量， 使用时不要加单、双引号。

```
main(){
enum body
{ a,b,c,d } month[31],j;
int i;
j=a;
for(i=1;i<=30;i++){
month[i]=j;
j++;
if (j>d) j=a;
}
for(i=1;i<=30;i++){
switch(month[i])
{
case a:printf(" %2d %c\t",i,'a'); break;
case b:printf(" %2d %c\t",i,'b'); break;
case c:printf(" %2d %c\t",i,'c'); break;
case d:printf(" %2d %c\t",i,'d'); break;
default:break;
}
}
printf("\n");
}
```

位运算

前面介绍的各种运算都是以字节作为最基本位进行的。 但在很多系统程序中常要求在位(bit)一级进行运算或处理。C 语言提供了位运算的功能， 这使得C 语言也能像汇编语言一样用来编写系统程序。

一、位运算符C 语言提供了六种位运算符：

& 按位与

| 按位或

^ 按位异或

~ 取反

<< 左移

>> 右移

1. 按位与运算 按位与运算符"&"是双目运算符。其功能是参与运算的两数各对应的二进位相与。只有对应的两个二进位均为 1 时，结果位才为 1 ，否则为 0。参与运算的数以补码方式出现。

例如：9&5 可写算式如下： 00001001 (9 的二进制补码)&00000101 (5 的二进制补码) 00000001 (1 的二进制补码)可见 9&5=1。

按位与运算通常用来对某些位清 0 或保留某些位。例如把 a 的高八位清 0 ， 保留低八位， 可作 a&255 运算 (255 的二进制数为 0000000011111111)。

```
main(){
int a=9,b=5,c;
c=a&b;
printf("a=%d\nb=%d\nc=%d\n",a,b,c);
}
```

2. 按位或运算 按位或运算符“|”是双目运算符。其功能是参与运算的两数各对应的二进位相或。只要对应的二个二进位有一个为 1 时，结果位就为 1。参与运算的两个数均以补码出现。

例如：9|5 可写算式如下： 00001001|00000101

00001101 (十进制为 13)可见 9|5=13

```
main(){
int a=9,b=5,c;
c=a|b;
printf("a=%d\nb=%d\nc=%d\n",a,b,c);
}
```

3. 按位异或运算 按位异或运算符“^”是双目运算符。其功能是参与运算的两数各对应的二进位相异或，当两对应的二进位相异时，结果为 1。参与运算数仍以补码出现，例如 9^5 可写成算式如下： 00001001^00000101 00001100 (十进制为 12)

```
main(){
int a=9;
a=a^15;
```

```
printf("a=%d\n",a);
}
```

4．求反运算 求反运算符～为单目运算符，具有右结合性。 其功能是对参与运算的数的各二进位按位求反。例如～9 的运算为： ~(0000000000001001)结果为：1111111111110110

5．左移运算 左移运算符“<<”是双目运算符。其功能把“<< ”左边的运算数的各二进位全部左移若干位，由“<<”右边的数指定移动的位数，
高位丢弃，低位补 0。例如： a<<4 指把 a 的各二进位向左移动 4 位。如 a=00000011(十进制 3)，左移 4 位后为 00110000(十进制 48)。
6．右移运算 右移运算符“>>”是双目运算符。其功能是把“>> ”左边的运算数的各二进位全部右移若干位，“>>”右边的数指定移动的位数。
例如：设 a=15，a>>2 表示把 000001111 右移为 00000011(十进制 3)。 应该说明的是，对于有符号数，在右移时，符号位将随同移动。当为正数时， 最高位补 0，而为负数时，符号位为 1，最高位是补 0 或是补 1 取决于编译系统的规定。Turbo C 和很多系统规定为补 1。

```
main(){
unsigned a,b;
printf("input a number: ");
scanf("%d",&a);
b=a>>5;
b=b&15;
printf("a=%d\tb=%d\n",a,b);
}
```

请再看一例!

```
main(){
char a='a',b='b';
int p,c,d;
p=a;
p=(p<<8)|b;
d=p&0xff;
c=(p&0xff00)>>8;
printf("a=%d\nb=%d\nc=%d\nd=%d\n",a,b,c,d);
}
```

位域

有些信息在存储时，并不需要占用一个完整的字节， 而只需占几个或一个二进制位。例如在存放一个开关量时，只有 0 和 1 两种状态， 用一位二进位即可。为了节省存储空间，并使处理简便，C 语言又提供了一种数据结构，称为“位域”或“位段”。所谓“位域”是把一个字节中的二进位划分为几个不同的区域， 并说明每个区域的位数。每个域有一个域名，允许在程序中按域名进行操作。 这样就可以把几个不同的对象用一个字节的二进制位域来表示。

一、位域的定义和位域变量的说明位域定义与结构定义相仿，其形式为：

```
struct 位域结构名
{ 位域列表 };
```

其中位域列表的形式为： 类型说明符 位域名：位域长度

例如：

```
struct bs
{
int a:8;
int b:2;
int c:6;
};
```

位域变量的说明与结构变量说明的方式相同。 可采用先定义后说明，同时定义说明或者直接说明这三种方式。例如：

```
struct bs
{
int a:8;
int b:2;
int c:6;
}data;
```

说明 data 为 bs 变量，共占两个字节。其中位域 a 占 8 位，位域 b 占 2 位，位域 c 占 6 位。

对于位域的定义尚有以下几点说明：

1. 一个位域必须存储在同一个字节中，不能跨两个字节。如一个字节所剩空间不够存放另一位域时，应从下一单元起存放该位域。也可以有意使某位域从下一单元开始。例如：

```
struct bs
{
unsigned a:4
unsigned :0 /*空域*/
unsigned b:4 /*从下一单元开始存放*/
unsigned c:4
}
```

在这个位域定义中，a 占第一字节的 4 位，后 4 位填 0 表示不使用，b 从第二字节开始，占用 4 位，c 占用 4 位。

2. 由于位域不允许跨两个字节，因此位域的长度不能大于一个字节的长度，也就是说不能超过 8 位二进位。

3. 位域可以无位域名，这时它只用来作填充或调整位置。无名的位域是不能使用的。例如：

```
struct k
{
int a:1
int :2 /*该 2 位不能使用*/
int b:3
int c:2
```

```
};
```
从以上分析可以看出，位域在本质上就是一种结构类型， 不过其成员是按二进位分配的。

二、位域的使用位域的使用和结构成员的使用相同，其一般形式为：位域变量名·位域名 位域允许用各种格式输出。

```
main(){
struct bs
{
unsigned a:1;
unsigned b:3;
unsigned c:4;
} bit,*pbit;
bit.a=1;
bit.b=7;
bit.c=15;
printf("%d,%d,%d\n",bit.a,bit.b,bit.c);
pbit=&bit;
pbit->a=0;
pbit->b&=3;
pbit->c|=1;
printf("%d,%d,%d\n",pbit->a,pbit->b,pbit->c);
}
```
上例程序中定义了位域结构 bs，三个位域为 a,b,c。说明了 bs 类型的变量 bit 和指向 bs 类型的指针变量 pbit。这表示位域也是可以使用指针的。

程序的 9、10、11 三行分别给三个位域赋值。(应注意赋值不能超过该位域的允许范围)程序第 12 行以整型量格式输出三个域的内容。第 13 行把位域变量 bit 的地址送给指针变量 pbit。第 14 行用指针方式给位域 a 重新赋值，赋为 0。第 15 行使用了复合的位运算符"&=",该行相当于： pbit->b=pbit->b&3 位域 b 中原有值为 7，与 3 作按位与运算的结果为 3(111&011=011,十进制值为 3)。同样，程序第 16 行中使用了复合位运算"|=", 相当于：pbit->c=pbit->c|1 其结果为 15。程序第 17 行用指针方式输出了这三个域的值。

类型定义符 t y p e d e f

C语言不仅提供了丰富的数据类型，而且还允许由用户自己定义类型说明符，也就是说允许由用户为数据类型取“别名”。 类型定义符 typedef 即可用来完成此功能。例如，有整型量 a,b,其说明如下： int aa,b; 其中 int 是整型变量的类型说明符。int 的完整写法为 integer，

为了增加程序的可读性，可把整型说明符用 typedef 定义为： typedef int INTEGER 这以后就可用 INTEGER 来代替 int 作整型变量的类型说明了。 例如： INTEGER a,b;它等效于：int a,b; 用 typedef 定义数组、指针、结构等类型将带来很大的方便，不仅使程序书写简单而且使意义更为明确，因而增强了可读性。例如：

typedef char NAME[20]; 表示 NAME 是字符数组类型，数组长度为 20。

然后可用 NAME 说明变量，如： NAME a1,a2,s1,s2; 完全等效于： char

a1[20],a2[20],s1[20],s2[20]
又如：
typedef struct stu{ char name[20];
int age;
char sex;
} STU;
定义 STU 表示 stu 的结构类型，然后可用 STU 来说明结构变量： STU body1,body2;
typedef 定义的一般形式为： typedef 原类型名 新类型名 其中原类型名中含有定义部分，新类型名一般用大写表示， 以
便于区别。在有时也可用宏定义来代替 typedef 的功能，但是宏定义是由预处理完成的，而 typedef 则是在编译时完成的，后者更为灵活方便。

第九节　和“#”谈谈预处理

预处理

在前面各章中，已多次使用过以“#”号开头的预处理命令。如包含命令# include，宏定义命令# define 等。在源程序中这些命令都放在函数之外， 而且一般都放在源文件的前面，它们称为预处理部分。

所谓预处理是指在进行编译的第一遍扫描(词法扫描和语法分析)之前所作的工作。预处理是C语言的一个重要功能， 它由预处理程序负责完成。当对一个源文件进行编译时， 系统将自动引用预处理程序对源程序中的预处理部分作处理， 处理完毕自动进入对源程序的编译。

C语言提供了多种预处理功能，如宏定义、文件包含、 条件编译等。合理地使用预处理功能编写的程序便于阅读、修改、 移植和调试，也有利于模块化程序设计。本节我们就和“#”谈谈预处理。

宏定义

在C语言源程序中允许用一个标识符来表示一个字符串，称为“宏”。被定义为“宏”的标识符称为“宏名”。在编译预处理时，对程序中所有出现的“宏名”，都用宏定义中的字符串去代换， 这称为“宏代换”或“宏展开”。

宏定义是由源程序中的宏定义命令完成的。 宏代换是由预处理程序自动完成的。在C语言中，“宏”分为有参数和无参数两种。 下面分别讨论这两种“宏”的定义和调用。

无参宏定义

无参宏的宏名后不带参数。其定义的一般形式为： #define 标识符 字符串 其中的“#”表示这是一条预处理命令。凡是以“#”开头的均为预处理命令。“define”为宏定义命令。 “标识符”为所定义的宏名。“字符串”可以是常数、表达式、格式串等。在前面介绍过的符号常量的定义就是一种无参宏定义。 此外，常对程序中反复使用的表达式进行宏定义。例如： # define M (y*y+3*y) 定义M表达式(y*y+3*y)。在编写源程序时，所有的(y*y+3*y)都可由 M 代替，而对源程序作编译时，将先由预处理程序进行宏代换，即用(y*y+3*y)表达式去置换所有的宏名M，然后再进行编译。

```
#define M (y*y+3*y)
main(){
int s,y;
printf("input a number: ");
scanf("%d",&y);
s=3*M+4*M+5*M;
printf("s=%d\n",s);
}
```

上例程序中首先进行宏定义，定义M表达式(y*y+3*y),在 s= 3*M+4*M+5* M 中作了宏调

用。在预处理时经宏展开后该语句变为：s=3*(y*y+3*y)+4(y*y+3*y)+5(y*y+3*y);但要注意的是，在宏定义中表达式(y*y+3*y)两边的括号不能少。否则会发生错误。

当作以下定义后： #difine M y*y+3*y 在宏展开时将得到下述语句：s=3*y*y+3*y+4*y*y+3*y+5*y*y+3*y；这相当于； 3y 2+3y+4y 2+3y+5y 2+3y；显然与原题意要求不符。计算结果当然是错误的。 因此在作宏定义时必须十分注意。应保证在宏代换之后不发生错误。对于宏定义还要说明以下几点：

1. 宏定义是用宏名来表示一个字符串，在宏展开时又以该字符串取代宏名，这只是一种简单的代换，字符串中可以含任何字符，可以是常数，也可以是表达式，预处理程序对它不作任何检查。如有错误，只能在编译已被宏展开后的源程序时发现。

2. 宏定义不是说明或语句，在行末不必加分号，如加上分号则连分号也一起置换。

3. 宏定义必须写在函数之外，其作用域为宏定义命令起到源程序结 束。如要终止其作用域可使用# undef 命令，例如： # define PI 3.14159

```
main()
{
……
}
```

undef PIPI 的作用域

```
f1()
```

....表示 PI 只在 main 函数中有效，在 f1 中无效。

4. 宏名在源程序中若用引号括起来，则预处理程序不对其作宏代换。

```
#define OK 100
main()
{
printf("OK");
printf("\n");
}
```

上例中定义宏名 OK 表示 100，但在 printf 语句中 OK 被引号括起来，因此不作宏代换。程序的运行结果为：OK 这表示把“OK”当字符串处理。

5. 宏定义允许嵌套，在宏定义的字符串中可以使用已经定义的宏名。在宏展开时由预处理程序层层代换。例如： #define PI 3.1415926

#define S PI*y*y /* PI 是已定义的宏名*/对语句： printf("%f",s);在宏代换后变为：printf("%f",3.1415926*y*y);

6. 习惯上宏名用大写字母表示，以便于与变量区别。但也允许用小写字母。

7. 可用宏定义表示数据类型，使书写方便。例如： #define STU struct stu 在程序中可用 STU 作变量说明： STU body[5],*p;#define INTEGER int 在程序中即可用 INTEGER 作整型变量说明： INTEGER a,b; 应注意用宏定义表示数据类型和用 typedef 定义数据说明符的区别。宏定义只是简单的字符串代换，是在预处理完成的，而 typedef 是在编译时处理的，它不是作简单的代换， 而是对类型说明符重新命名。被命名的标识符具有类型定义说明的

功能。请看下面的例子： #define PIN1 int* typedef (int*) PIN2;从形式上看这两者相似，但在实际使用中却不相同。下面用 PIN1,PIN2 说明变量时就可以看出它们的区别：PIN1 a,b;在宏代换后变成 int *a,b;表示 a 是指向整型的指针变量，而 b 是整型变量。然而：PIN2 a,b;表示 a,b 都是指向整型的指针变量。因为 PIN2 是一个类型说明符。由这个例子可见，宏定义虽然也可表示数据类型， 但毕竟是作字符
代换。在使用时要分外小心，以避出错。

8. 对“输出格式”作宏定义，可以减少书写麻烦。例 9.3 中就采用了这种方法。

```
#define P printf
#define D "%d\n"
#define F "%f\n"
main(){
int a=5, c=8, e=11;
float b=3.8, d=9.7, f=21.08;
P(D F,a,b);
P(D F,c,d);
P(D F,e,f);
}
```

带参宏定义

C 语言允许宏带有参数。在宏定义中的参数称为形式参数， 在宏调用中的参数称为实际参数。对带参数的宏，在调用中，不仅要宏展开， 而且要用实参去代换形参。

带参宏定义的一般形式为：#define 宏名(形参表) 字符串 在字符串中含有各个形参。带参宏调用的一般形式为： 宏名(实参表);
例如：

```
#define M(y) y*y+3*y /*宏定义*/
:
k=M(5); /*宏调用*/
```

: 在宏调用时，用实参 5 去代替形参 y， 经预处理宏展开后的语句为： k=5*5+3*5

```
#define MAX(a,b) (a>b)?a:b
main(){
int x,y,max;
printf("input two numbers: ");
scanf("%d%d",&x,&y);
max=MAX(x,y);
printf("max=%d\n",max);
}
```

上例程序的第一行进行带参宏定义，用宏名 MAX 表示条件表达式(a>b)?a:b，形参 a,b 均出现在条件表达式中。程序第七行 max=MAX(x,
y)为宏调用，实参 x,y，将代换形参 a,b。宏展开后该语句为： max=(x>y)?x:y;用于计算 x,y 中的大数。对于带参的宏定义有以下问题需要说明：

1．带参宏定义中，宏名和形参表之间不能有空格出现。
例如把： #define MAX(a,b) (a>b)?a:b 写为： #define MAX (a,b) (a>b)?a:b 将被认为是无参宏定义，宏名 MAX 代表字符串 (a,b)(a>b)?a:b。
宏展开时，宏调用语句： max=MAX(x,y);将变为： max=(a,b)(a>b)?a:b(x,y);这显然是错误的。

2．在带参宏定义中，形式参数不分配内存单元，因此不必作类型定义。而宏调用中的实参有具体的值。要用它们去代换形参，因此必须作类型说明。这是与函数中的情况不同的。在函数中，形参和实参是两个不同的量，各有自己的作用域，调用时要把实参值赋予形参，进行“值传递”。而在带参宏中，只是符号代换，不存在值传递的问题。

3．在宏定义中的形参是标识符，而宏调用中的实参可以是表达式。

```
#define SQ(y) (y)*(y)
main(){
int a,sq;
printf("input a number: ");
scanf("%d",&a);
sq=SQ(a+1);
printf("sq=%d\n",sq);
}
```

上例中第一行为宏定义，形参为 y。程序第七行宏调用中实参为 a+1，是一个表达式，在宏展开时，用 a+1 代换 y，再用(y)*(y) 代换 SQ，得到如下语句： sq=(a+1)*(a+1); 这与函数的调用是不同的， 函数调用时要把实参表达式的值求出来再赋予形参。 而宏代换中对实参表达式不作计算直接地照原样代换。

4．在宏定义中，字符串内的形参通常要用括号括起来以避免出错。 在上例中的宏定义中(y)*(y)表达式的 y 都用括号括起来，因此结果是正确的。如果去掉括号，把程序改为以下形式：

```
#define SQ(y) y*y
main(){
int a,sq;
printf("input a number: ");
scanf("%d",&a);
sq=SQ(a+1);
printf("sq=%d\n",sq);
}
```

运行结果为：input a number:3
sq=7 同样输入 3，但结果却是不一样的。问题在哪里呢? 这是由于代换只作符号代换而不作其它处理而造成的。 宏代换后将得到以下语句： sq=a+1*a+1; 由于 a 为 3 故 sq 的值为 7。这显然与题意相违，因此参数两边的括号是不能少的。即使在参数两边加括号还是不够的，请看下面程序：

```
#define SQ(y) (y)*(y)
main(){
```

```
int a,sq;
printf("input a number: ");
scanf("%d",&a);
sq=160/SQ(a+1);
printf("sq=%d\n",sq);
}
```

本程序与前例相比，只把宏调用语句改为： sq=160/SQ(a+1)；运行本程序如输入值仍为 3 时，希望结果为 10。但实际运行的结果如下：input a number:3 sq=160 为什么会得这样的结果呢?分析宏调用语句，在宏代换之后变为： sq=160/(a+1)*(a+1);a 为 3 时，由于“/”和“*”运算符优先级和结合性相同， 则先作 160/(3+1)得 40，再作 40*(3+1)最后得 160。为了得到正确答案应在宏定义中的整个字符串外加括号， 程序修改如下

```
#define SQ(y) ((y)*(y))
main(){
int a,sq;
printf("input a number: ");
scanf("%d",&a);
sq=160/SQ(a+1);
printf("sq=%d\n",sq);
}
```

以上讨论说明，对于宏定义不仅应在参数两侧加括号， 也应在整个字符串外加括号。

5. 带参的宏和带参函数很相似，但有本质上的不同，除上面已谈到的各点外，把同一表达式用函数处理与用宏处理两者的结果有可能是不同的。main(){

```
int i=1;
while(i<=5)
printf("%d\n",SQ(i++));
}
SQ(int y)
{
return((y)*(y));
}#define SQ(y) ((y)*(y))
main(){
int i=1;
while(i<=5)
printf("%d\n",SQ(i++));
}
```

在上例中函数名为 SQ，形参为 Y，函数体表达式为((y)*(y))。在例中宏名为 SQ，形参也为 y,字符串表达式为(y)*(y))。两例是相同的。函数调用为 SQ(i++),宏调用为 SQ(i++)，实参也是相同的。从输出结果来看，却大不相同。分析如下：在例 9.6 中，函数调用是把实参 i 值传给形参 y 后自增 1。 然后输出函数值。因而要循环 5 次。输出 1～5 的平方值。而在宏调用时，只作代换。SQ(i++)被代换为((i++)*(i++))。在第一次循环时，由于 i 等于 1，其计算过程为：表达式中前一个 i 初值为 1，然后 i 自增 1 变为 2，因此表达式中第 2 个 i 初值为 2，两相乘的结果也为 2，然后 i 值再自增 1，得 3。在第二次循环时，i 值已有初值为 3，因此表达式中前一个 i 为 3，后一个 i 为 4， 乘积为 12，然后 i 再自增 1 变为 5。进

入第三次循环，由于 i 值已为 5，所以这将是最后一次循环。计算表达式的值为 5*6 等于 30。i 值再自增 1 变为 6，不再满足循环条件，停止循环。从以上分析可以看出函数调用和宏调用二者在形式上相似， 在本质上是完全不同的。

6．宏定义也可用来定义多个语句，在宏调用时，把这些语句又代换到源程序内。看下面的例子。

```
#define SSSV(s1,s2,s3,v) s1=l*w;s2=l*h;s3=w*h;v=w*l*h;
main(){
int l=3,w=4,h=5,sa,sb,sc,vv;
SSSV(sa,sb,sc,vv);
printf("sa=%d\nsb=%d\nsc=%d\nvv=%d\n",sa,sb,sc,vv);
}
```

程序第一行为宏定义，用宏名 SSSV 表示 4 个赋值语句，4 个形参分别为 4 个赋值符左部的变量。在宏调用时，把 4 个语句展开并用实参代替形参。使计算结果送入实参之中。

文件包含

文件包含是 C 预处理程序的另一个重要功能。文件包含命令行的一般形式为：#include"文件名" 在前面我们已多次用此命令包含过库函数的头文件。例如：

```
#include"stdio.h"
#include"math.h"
```

文件包含命令的功能是把指定的文件插入该命令行位置取代该命令行， 从而把指定的文件和当前的源程序文件连成一个源文件。在程序设计中，文件包含是很有用的。 一个大的程序可以分为多个模块，由多个程序员分别编程。 有些公用的符号常量或宏定义等可单独组成一个文件， 在其它文件的开头用包含命令包含该文件即可使用。这样，可避免在每个文件开头都去书写那些公用量， 从而节省时间，并减少出错。

对文件包含命令还要说明以下几点：

1．包含命令中的文件名可以用双引号括起来，也可以用尖括号括起来。例如以下写法都是允许的： #include"stdio.h" #include<math.h> 但是这两种形式是有区别的：使用尖括号表示在包含文件目录中去查找(包含目录是由用户在设置环境时设置的)， 而不在源文件目录去查找； 使用双引号则表示首先在当前的源文件目录中查找，若未找到才到包含目录中去查找。 用户编程时可根据自己文件所在的目录来选择某一种命令形式。

2．一个 include 命令只能指定一个被包含文件，若有多个文件要包含，则需用多个 include 命令。

3．文件包含允许嵌套，即在一个被包含的文件中又可以包含另一个文件。

条件编译

预处理程序提供了条件编译的功能。 可以按不同的条件去编译不同的程序部分，因而产生不同的目标代码文件。 这对于程序的移植和调试是很有用的。 条件编译有三种形式，下面分别介绍：

1. 第一种形式：

```
#ifdef 标识符
程序段 1
#else
程序段 2
#endif
```

它的功能是，如果标识符已被 #define 命令定义过则对程序段 1 进行编译；否则对程序段 2 进行编译。如果没有程序段 2(它为空)，本格式中的#else 可以没有， 即可以写为：

```
#ifdef 标识符
程序段 #endif
#define NUM ok
main(){
struct stu
{
int num;
char *name;
char sex;
float score;
} *ps;
ps=(struct stu*)malloc(sizeof(struct stu));
ps->num=102;
ps->name="Zhang ping";
ps->sex='M';
ps->score=62.5;
#ifdef NUM
printf("Number=%d\nScore=%f\n",ps->num,ps->score);
#else
printf("Name=%s\nSex=%c\n",ps->name,ps->sex);
#endif
free(ps);
}
```

由于在程序的第 16 行插入了条件编译预处理命令，因此要根据 NUM 是否被定义过来决定编译那一个 printf 语句。而在程序的第一行已对 NUM 作过宏定义，因此应对第一个 printf 语句作编译故运行结果是输出了学号和成绩。在程序的第一行宏定义中，定义 NUM 表示字符串 OK，其实也可以为任何字符串，甚至不给出任何字符串，写为： #define NUM 也具有同样的意义。 只有取消程序的第一行才会去编译第二个 printf 语句。读者可上机试作。

2. 第二种形式：
#ifndef 标识符
程序段 1
#else
程序段 2
#endif
与第一种形式的区别是将“ifdef”改为“ifndef”。它的功能是，如果标识符未被#define 命令定义过则对程序段 1 进行编译， 否则对程序段 2 进行编译。这与第一种形式的功能正相反。

3. 第三种形式：
#if 常量表达式
程序段 1
#else
程序段 2
#endif
它的功能是，如常量表达式的值为真(非 0)，则对程序段 1 进行编译，否则对程序段 2 进行编译。因此可以使程序在不同条件下，完成不同的功能

```
#define R 1
main(){
float c,r,s;
printf ("input a number: ");
scanf("%f",&c);
#if R
r=3.14159*c*c;
printf("area of round is: %f\n",r);
#else
s=c*c;
printf("area of square is: %f\n",s);
#endif
}
```

本例中采用了第三种形式的条件编译。在程序第一行宏定义中，定义 R 为 1，因此在条件编译时，常量表达式的值为真， 故计算并输出圆面积。上面介绍的条件编译当然也可以用条件语句来实现。 但是用条件语句将会对整个源程序进行编译，生成的目标代码程序很长，而采用条件编译，则根据条件只编译其中的程序段 1 或程序段 2， 生成的目标程序较短。如果条件选择的程序段很长， 采用条件编译的方法是十分必要的。

第十节　文件

什么是文件

所谓“文件”是指一组相关数据的有序集合。这个数据集有一个名称,叫做文件名。实际上在前面的各章中我们已经多次使用了文件，例如源程序文件、目标文件、可执行文件、库文件 (头文件)等。文件通常是驻留在外部介质(如磁盘等)上的， 在使用时才调入内存中来。从不同的角度可对文件作不同的分类。从用户的角度看，文件可分为普通文件和设备文件两种。

普通文件是指驻留在磁盘或其它外部介质上的一个有序数据集，可以是源文件、目标文件、可执行程序； 也可以是一组待输入处理的原始数据，或者是一组输出的结果。对于源文件、目标文件、 可执行程序可以称作程序文件，对输入输出数据可称作数据文件。

设备文件是指与主机相联的各种外部设备，如显示器、打印机、键盘等。在操作系统中，把外部设备也看作是一个文件来进行管理,把它们的输入、输出等同于对磁盘文件的读和写。通常把显示器定义为标准输出文件， 一般情况下在屏幕上显示有关信息就是向标准输出文件输出。如前面经常使用的 printf,putchar 函数就是这类输出。键盘通常被指定标准的输入文件， 从键盘上输入就意味着从标准输入文件上输入数据。scanf,getchar 函数就属于这类输入。

从文件编码的方式来看，文件可分为 ASCII 码文件和二进制码文件两种。

ASCII 文件也称为文本文件，这种文件在磁盘中存放时每个字符对应一个字节，用于存放对应的 ASCII 码。例如，数 5678 的存储形式为：

```
ASCII 码：    00110101 00110110 00110111 00111000
               ↓        ↓        ↓        ↓
十进制码：     5        6        7        8
```

共占用 4 个字节。ASCII 码文件可在屏幕上按字符显示， 例如源程序文件就是 ASCII 文件，用 DOS 命令 TYPE 可显示文件的内容。由于是按字符显示，因此能读懂文件内容。

二进制文件是按二进制的编码方式来存放文件的。 例如， 数 5678 的存储形式为：00010110 00101110 只占二个字节。二进制文件虽然也可在屏幕上显示， 但其内容无法读懂。C 系统在处理这些文件时，并不区分类型，都看成是字符流，按字节进行处理。 输入输出字符流的开始和结束只由程序控制而不受物理符号(如回车符)的控制。 因此也把这种文件称作“流式文件”。

本节我们讨论流式文件的打开、关闭、读、写、 定位等各种操作。文件指针在C语言中用一个指针变量指向一个文件， 这个指针称为文件指针。通过文件指针就可对它所指的文件进行各种操作。定义说明文件指针的一般形式为：FILE* 指针变量标识符；其中 FILE 应为大写，它实际上是由系统定义的一个结构， 该结构中含有文件名、文件状态和文件当前位置等信息。 在编写源程序时不必关心 FILE 结构的细节。例如：FILE *fp; 表示 fp 是指向 FILE 结构的指针变量，通过 fp 即可找存放某个文件信息的结构变量，然后按结构变量提供的信息找到该文件， 实施对文件的操作。习惯上也笼统地把 fp 称为指向一个文件的指针。文件的打开与关闭文件在进行读写操作之前要先打开，使用完毕要关闭。 所谓打开文件，实际上是建立文件的各种有关信息， 并使文件指针指向该文件，以便进行其它操作。关闭文件则断开指针与文件之间的联系，也就禁止再对该文件进行操作。

文件打开函数 f o p e n

fopen 函数用来打开一个文件，其调用的一般形式为： 文件指针名=fopen(文件名，使用文件方式) 其中，“文件指针名”必须是被说明为 FILE 类型的指针变量，“文件名”是被打开文件的文件名。 “使用文件方式”是指文件的类型和操作要求。“文件名”是字符串常量或字符串数组。例如：

```
FILE *fp;
fp=("file a","r");
```

其意义是在当前目录下打开文件 file a， 只允许进行“读”操作，并使 fp 指向该文件。
又如：

```
FILE *fphzk
fphzk=("c:\\hzk16',"rb")
```

其意义是打开 C 驱动器磁盘的根目录下的文件 hzk16， 这是一个二进制文件，只允许按二进制方式进行读操作。两个反斜线“\\ ”中的第一个表示转义字符，第二个表示根目录。
使用文件的方式共有 12 种，下面给出了它们的符号和意义。

文件使用方式	意 义
“rt”	只读打开一个文本文件，只允许读数据
“wt”	只写打开或建立一个文本文件，只允许写数据
“at”	追加打开一个文本文件，并在文件末尾写数据
“rb”	只读打开一个二进制文件，只允许读数据
“wb”	只写打开或建立一个二进制文件，只允许写数据
“ab”	追加打开一个二进制文件，并在文件末尾写数据
“rt+”	读写打开一个文本文件，允许读和写
“wt+”	读写打开或建立一个文本文件，允许读写
“at+”	读写打开一个文本文件，允许读，或在文件末追加数 据
“rb+”	读写打开一个二进制文件，允许读和写
“wb+”	读写打开或建立一个二进制文件，允许读和写
“ab+”	读写打开一个二进制文件，允许读，或在文件末追加数据

对于文件使用方式有以下几点说明：
1．文件使用方式由 r,w,a,t,b，+六个字符拼成，各字符的含义是：
r(read)：读
w(write)：写
a(append)：追加
t(text)：文本文件，可省略不写
b(banary)：二进制文件
+：读和写

2．凡用“r”打开一个文件时，该文件必须已经存在， 且只能从该文件读出。

3．用“w”打开的文件只能向该文件写入。 若打开的文件不存在，则以指定的文件名建立该文件，若打开的文件已经存在，则将该文件删去，重建一个新文件。

4．若要向一个已存在的文件追加新的信息，只能用“a ”方式打开文件。但此时该文件必须是存在的，否则将会出错。

5．在打开一个文件时，如果出错，fopen 将返回一个空指针值 NULL。在程序中可以用这一信息来判别是否完成打开文件的工作，并作相应的处理。因此常用以下程序段打开文件：

```
if((fp=fopen("c:\\hzk16","rb")==NULL)
{
printf("\nerror on open c:\\hzk16 file!");
getch();
exit(1);
}
```

这段程序的意义是，如果返回的指针为空，表示不能打开 C 盘根目录下的 hzk16 文件，则给出提示信息“error on open c:\ hzk16file!”，下一行 getch()的功能是从键盘输入一个字符，但不在屏幕上显示。在这里，该行的作用是等待， 只有当用户从键盘敲任一键时，程序才继续执行， 因此用户可利用这个等待时间阅读出错提示。敲键后执行 exit(1)退出程序。

6．把一个文本文件读入内存时，要将 ASCII 码转换成二进制码， 而把文件以文本方式写入磁盘时，也要把二进制码转换成 ASCII 码，因此文本文件的读写要花费较多的转换时间。对二进制文件的读写不存在这种转换。

7．标准输入文件(键盘)，标准输出文件(显示器)，标准出错输出(出错信息)是由系统打开的，可直接使用。文件关闭函数ｆｃｌｏｓｅ文件一旦使用完毕，应用关闭文件函数把文件关闭， 以避免文件的数据丢失等错误。

fclose 函数

调用的一般形式是： fclose(文件指针)； 例如：
fclose(fp)；正常完成关闭文件操作时，fclose 函数返回值为 0。如返回非零值则表示有错误发生。文件的读写对文件的读和写是最常用的文件操作。

在C语言中提供了多种文件读写的函数：
- 字符读写函数 ：fgetc 和 fputc
- 字符串读写函数：fgets 和 fputs
- 数据块读写函数：freed 和 fwrite
- 格式化读写函数：fscanf 和 fprinf

下面分别予以介绍。使用以上函数都要求包含头文件 stdio.h。字符读写函数ｆｇｅｔｃ和ｆｐｕｔｃ字符读写函数是以字符(字节)为单位的读写函数。 每次可从文件读出或向文件写入一个字符。

一、读字符函数 fgetc

fgetc 函数的功能是从指定的文件中读一个字符，函数调用的形式为： 字符变量=fgetc(文件指针)； 例如：ch=fgetc(fp);其意义是从打开的文件 fp 中读取一个字符并送入 ch 中。

对于 fgetc 函数的使用有以下几点说明：

1. 在 fgetc 函数调用中，读取的文件必须是以读或读写方式打开的。

2. 读取字符的结果也可以不向字符变量赋值，例如：fgetc(fp);但是读出的字符不能保存。

3. 在文件内部有一个位置指针。用来指向文件的当前读写字节。在文件打开时，该指针总是指向文件的第一个字节。使用 fgetc 函数后， 该位置指针将向后移动一个字节。 因此可连续多次使用 fgetc 函数，读取多个字符。 应注意文件指针和文件内部的位置指针不是一回事。文件指针是指向整个文件的，须在程序中定义说明，只要不重新赋值，文件指针的值是不变的。文件内部的位置指针用以指示文件内部的当前读写位置，每读写一次，该指针均向后移动，它不需在程序中定义说明，而是由系统自动设置的。

读入文件 e10-1.c，在屏幕上输出。

```
#include<stdio.h>
main()
{
FILE *fp;
char ch;
if((fp=fopen("e10_1.c","rt"))==NULL)
{
printf("Cannot open file strike any key exit!");
getch();
exit(1);
}
ch=fgetc(fp);
while (ch!=EOF)
{
putchar(ch);
ch=fgetc(fp);
}
fclose(fp);
}
```

本例程序的功能是从文件中逐个读取字符，在屏幕上显示。 程序定义了文件指针 fp，以读文本文件方式打开文件“e10_1.c”， 并使 fp 指向该文件。如打开文件出错， 给出提示并退出程序。程序第 12 行先读出一个字符，然后进入循环， 只要读出的字符不是文件结束标志(每个文件末有一结束标志 EOF)就把该字符显示在屏幕上，再读入下一字符。每读一次，文件内部的位置指针向后移动一个字符，文件结束时，该指针指向 EOF。执行本程序将显示整个文件。

二、写字符函数 fputc

fputc 函数的功能是把一个字符写入指定的文件中，函数调用的 形式为： fputc(字符量，文件指针)； 其中，待写入的字符量可以是字符常量或变量，例如：fputc('a',fp); 其意义是把字符 a 写入 fp 所指向的文件中。

对于 fputc 函数的使用也要说明几点：

1. 被写入的文件可以用、写、读写，追加方式打开，用写或读写方式打开一个已存在的文件时将清除原有的文件内容，写入字符从文件首开始。如需保留原有文件内容，希望写入的字符以文件末开始存放，必须以追加方式打开文件。被写入的文件若不存在，则创建该文件。

2. 每写入一个字符，文件内部位置指针向后移动一个字节。

3. fputc 函数有一个返回值，如写入成功则返回写入的字符， 否则返回一个 EOF。可用此来判断写入是否成功。

从键盘输入一行字符，写入一个文件， 再把该文件内容读出显示在屏幕上。

```
#include<stdio.h>
main()
{
FILE *fp;
char ch;
if((fp=fopen("string","wt+"))==NULL)
{
printf("Cannot open file strike any key exit!");
getch();
exit(1);
}
printf("input a string:\n");
ch=getchar();
while (ch!='\n')
{
fputc(ch,fp);
ch=getchar();
}
rewind(fp);
ch=fgetc(fp);
while(ch!=EOF)
{
putchar(ch);
ch=fgetc(fp);
}
printf("\n");
fclose(fp);
```

```
}
```

程序中第 6 行以读写文本文件方式打开文件 string。程序第 13 行从键盘读入一个字符后进入循环，当读入字符不为回车符时， 则把该字符写入文件之中，然后继续从键盘读入下一字符。 每输入一个字符，文件内部位置指针向后移动一个字节。写入完毕， 该指针已指向文件末。如要把文件从头读出，须把指针移向文件头， 程序第 19 行 rewind 函数用于把 fp 所指文件的内部位置指针移到文件头。 第 20 至 25 行用于读出文件中的一行内容。

把命令行参数中的前一个文件名标识的文件， 复制到后一个文件名标识的文件中， 如命令行中只有一个文件名则把该文件写到标准输出文件(显示器)中。

```
#include<stdio.h>
main(int argc,char *argv[])
{
FILE *fp1,*fp2;
char ch;
if(argc==1)
{
printf("have not enter file name strike any key exit");
getch();
exit(0);
}
if((fp1=fopen(argv[1],"rt"))==NULL)
{
printf("Cannot open %s\n",argv[1]);
getch();
exit(1);
}
if(argc==2) fp2=stdout;
else if((fp2=fopen(argv[2],"wt+"))==NULL)
{
printf("Cannot open %s\n",argv[1]);
getch();
exit(1);
}
while((ch=fgetc(fp1))!=EOF)
fputc(ch,fp2);
fclose(fp1);
fclose(fp2);
}
```

本程序为带参的 main 函数。程序中定义了两个文件指针 fp1 和 fp2，分别指向命令行参数中给出的文件。如命令行参数中没有给出文件名，则给出提示信息。程序第 18 行表示如果只给出一个文件名，则使 fp2 指向标准输出文件(即显示器)。程序第 25 行至 28 行用循环语句逐个读出文件 1 中的字符再送到文件 2 中。再次运行时，给出了一个文件名(由例 10.2 所建立的文件)， 故输出给标准输出文件 stdout，即在显示器上显示文件内容。第三次运行，给出了二个文件名，因此把 string 中的内容读出，写入到 OK 之中。可用 DOS 命令 type

显示 OK 的内容：字符串读写函数ｆｇｅｔｓ和ｆｐｕｔｓ

一、读字符串函数 fgets 函数的功能是从指定的文件中读一个字符串到字符数组中，函数调用的形式为： fgets(字符数组名，n，文件指针)； 其中的 n 是一个正整数。表示从文件中读出的字符串不超过 n-1 个字符。在读入的最后一个字符后加上串结束标志'\0'。例如：fgets(str,n,fp);的意义是从 fp 所指的文件中读出 n-1 个字符送入字符数组 str 中。
[例 10.4]从 e10_1.c 文件中读入一个含 10 个字符的字符串。

```
#include<stdio.h>
main()
{
FILE *fp;
char str[11];
if((fp=fopen("e10_1.c","rt"))==NULL)
{
printf("Cannot open file strike any key exit!");
getch();
exit(1);
}
fgets(str,11,fp);
printf("%s",str);
fclose(fp);
}
```

本例定义了一个字符数组 str 共 11 个字节，在以读文本文件方式打开文件 e101.c 后，从中读出 10 个字符送入 str 数组，在数组最后一个单元内将加上'\0'，然后在屏幕上显示输出 str 数组。输出的十个字符正是例 10.1 程序的前十个字符。

对 fgets 函数有两点说明：
1．在读出 n-1 个字符之前，如遇到了换行符或 EOF，则读出结束。
2．fgets 函数也有返回值，其返回值是字符数组的首地址。

二、写字符串函数 fputs

fputs 函数的功能是向指定的文件写入一个字符串，其调用形式为： fputs(字符串，文件指针) 其中字符串可以是字符串常量，也可以是字符数组名， 或指针 变量，例如：
fputs(“abcd“，fp)；
其意义是把字符串“abcd”写入 fp 所指的文件之中。

```
#include<stdio.h>
main()
{
FILE *fp;
char ch,st[20];
if((fp=fopen("string","at+"))==NULL)
{
printf("Cannot open file strike any key exit!");
```

```
getch();
exit(1);
}
printf("input a string:\n");
scanf("%s",st);
fputs(st,fp);
rewind(fp);
ch=fgetc(fp);
while(ch!=EOF)
{
putchar(ch);
ch=fgetc(fp);
}
printf("\n");
fclose(fp);
}
```

本例要求在 string 文件末加写字符串，因此，在程序第 6 行以追加读写文本文件的方式打开文件 string 。 然后输入字符串， 并用 fputs 函数把该串写入文件 string。在程序 15 行用 rewind 函数把文件内部位置指针移到文件首。 再进入循环逐个显示当前文件中的全部内容。

数据块读写函数ｆｒｅａｄ和ｆｗｒｉｔｅ

C 语言还提供了用于整块数据的读写函数。 可用来读写一组数据，如一个数组元素，一个结构变量的值等。读数据块函数调用的一般形式为： fread(buffer,size,count,fp); 写数据块函数调用的一般形式为：fwrite(buffer,size,count,fp); 其中 buffer 是一个指针，在 fread 函数中，它表示存放输入数据的首地址。在 fwrite 函数中，它表示存放输出数据的首地址。 size 表示数据块的字节数。count 表示要读写的数据块块数。fp 表示文件指针。

例如：fread(fa,4,5,fp); 其意义是从 fp 所指的文件中，每次读 4 个字节(一个实数)送入实数组 fa 中，连续读 5 次，即读 5 个实数到 fa 中。

从键盘输入两个学生数据，写入一个文件中， 再读出这两个学生的数据显示在屏幕上。

```
#include<stdio.h>
struct stu
{
char name[10];
int num;
int age;
char addr[15];
}boya[2],boyb[2],*pp,*qq;
main()
{
FILE *fp;
```

```
char ch;
int i;
pp=boya;
qq=boyb;
if((fp=fopen("stu_list","wb+"))==NULL)
{
printf("Cannot open file strike any key exit!");
getch();
exit(1);
}
printf("\ninput data\n");
for(i=0;i<2;i++,pp++)
scanf("%s%d%d%s",pp->name,&pp->num,&pp->age,pp->addr);
pp=boya;
fwrite(pp,sizeof(struct stu),2,fp);
rewind(fp);
fread(qq,sizeof(struct stu),2,fp);
printf("\n\nname\tnumber age addr\n");
for(i=0;i<2;i++,qq++)
printf("%s\t%5d%7d%s\n",qq->name,qq->num,qq->age,qq->addr);
fclose(fp);
}
```

本例程序定义了一个结构 stu,说明了两个结构数组 boya 和 boyb 以及两个结构指针变量 pp 和 qq。pp 指向 boya,qq 指向 boyb。程序第 16 行以读写方式打开二进制文件“stu_list”，输入二个学生数据之后，写入该文件中， 然后把文件内部位置指针移到文件首，读出两块学生数据后，在屏幕上显示。

格式化读写函数 f s c a n f 和 f p r i n t f

fscanf 函数，fprintf 函数与前面使用的 scanf 和 printf 函数的功能相似，都是格式化读写函数。 两者的区别在于 fscanf 函数和 fprintf 函数的读写对象不是键盘和显示器，而是磁盘文件。这两个函数的调用格式为： fscanf(文件指针，格式字符串，输入表列)；fprintf(文件指针，格式字符串，输出表列)； 例如：

```
fscanf(fp,"%d%s",&i,s);
fprintf(fp,"%d%c",j,ch);
```

用 fscanf 和 fprintf 函数也可以完成例 10.6 的问题。修改后的程序如例 10.7 所示。

```
#include<stdio.h>
struct stu
{
char name[10];
int num;
int age;
char addr[15];
```

```
}boya[2],boyb[2],*pp,*qq;
main()
{
FILE *fp;
char ch;
int i;
pp=boya;
qq=boyb;
if((fp=fopen("stu_list","wb+"))==NULL)
{
printf("Cannot open file strike any key exit!");
getch();
exit(1);
}
printf("\ninput data\n");
for(i=0;i<2;i++,pp++)
scanf("%s%d%d%s",pp->name,&pp->num,&pp->age,pp->addr);
pp=boya;
for(i=0;i<2;i++,pp++)
fprintf(fp,"%s %d %d %s\n",pp->name,pp->num,pp->age,pp->
addr);
rewind(fp);
for(i=0;i<2;i++,qq++)
fscanf(fp,"%s %d %d %s\n",qq->name,&qq->num,&qq->age,qq->addr);
printf("\n\nname\tnumber age addr\n");
qq=boyb;
for(i=0;i<2;i++,qq++)
printf("%s\t%5d %7d %s\n",qq->name,qq->num, qq->age,
qq->addr);
fclose(fp);
}
```

本程序中 fscanf 和 fprintf 函数每次只能读写一个结构数组元素，因此采用了循环语句来读写全部数组元素。 还要注意指针变量 pp,qq 由于循环改变了它们的值，因此在程序的 25 和 32 行分别对它们重新赋予了数组的首地址。

文件的随机读写

前面介绍的对文件的读写方式都是顺序读写， 即读写文件只能从头开始，顺序读写各个数据。 但在实际问题中常要求只读写文件中某一指定的部分。 为了解决这个问题可移动文件内部的位置指针到需要读写的位置，再进行读写，这种读写称为随机读写。 实现随机读写的关键是要按要求移动位置指针，这称为文件的定位。文件定位移动文件内部位置指针的函数主要有两个， 即 rewind 函数和 fseek 函数。

rewind 函数前面已多次使用过，其调用形式为： rewind(文件指针)； 它的功能是把文件内部的位置指针移到文件首。 下面主要介绍
fseek 函数。

fseek 函数用来移动文件内部位置指针，其调用形式为： fseek(文件指针，位移量，起始点)； 其中：“文件指针”指向被移动的文件。 “位移量”表示移动的字节数，要求位移量是 long 型数据，以便在文件长度大于 64KB 时不会出错。当用常量表示位移量时，要求加后缀“L”。“起始点”表示从何处开始计算位移量，规定的起始点有三种：文件首，当前位置和文件尾。
其表示方法如表。

起始点	表示符号	数字表示
文件首	SEEK—SET	0
当前位置	SEEK—CUR	1
文件末尾	SEEK—END	2

例如：
fseek(fp,100L,0);其意义是把位置指针移到离文件首 100 个字节处。还要说明的是 fseek 函数一般用于二进制文件。在文本文件中由于要进行转换，故往往计算的位置会出现错误。文件的随机读写在移动位置指针之后， 即可用前面介绍的任一种读写函数进行读写。由于一般是读写一个数据据块，因此常用 fread 和 fwrite 函数。下面用例题来说明文件的随机读写。

在学生文件 stu list 中读出第二个学生的数据。

```
#include<stdio.h>
struct stu
{
char name[10];
int num;
int age;
char addr[15];
}boy,*qq;
main()
{
FILE *fp;
char ch;
int i=1;
qq=&boy;
if((fp=fopen("stu_list","rb"))==NULL)
{
printf("Cannot open file strike any key exit!");
getch();
exit(1);
}
rewind(fp);
```

```
fseek(fp,i*sizeof(struct stu),0);
fread(qq,sizeof(struct stu),1,fp);
printf("\n\nname\tnumber age addr\n");
printf("%s\t%5d %7d %s\n",qq->name,qq->num,qq->age,
qq->addr);
}
```

本程序用随机读出的方法读出第二个学生的数据。程序中定义 boy 为 stu 类型变量，qq 为指向 boy 的指针。以读二进制文件方式打开文件，程序第 22 行移动文件位置指针。其中的 i 值为 1，表示从文件头开始，移动一个 stu 类型的长度， 然后再读出的数据即为第二个学生的数据。

文件检测函数

C 语言中常用的文件检测函数有以下几个。
一、文件结束检测函数 feof 函数调用格式： feof(文件指针)；
功能：判断文件是否处于文件结束位置，如文件结束，则返回值为 1，否则为 0。

二、读写文件出错检测函数 ferror 函数调用格式： ferror(文件指针)；
功能：检查文件在用各种输入输出函数进行读写时是否出错。 如 ferror 返回值为 0 表示未出错，否则表示有错。

三、文件出错标志和文件结束标志置 0 函数 clearerr 函数调用格式：clearerr(文件指针)；
功能：本函数用于清除出错标志和文件结束标志，使它们为 0 值。

C 库文件

C 系统提供了丰富的系统文件，称为库文件，C 的库文件分为两类，一类是扩展名为".h"的文件，称为头文件， 在前面的包含命令中我们已多次使用过。在".h"文件中包含了常量定义、 类型定义、宏定义、函数原型以及各种编译选择设置等信息。另一类是函数库，包括了各种函数的目标代码，供用户在程序中调用。 通常在程序中调用一个库函数时，要在调用之前包含该函数原型所在的".h" 文件。
在附录中给出了全部库函数。

ALLOC.H	说明内存管理函数(分配、释放等)。
ASSERT.H	定义 assert 调试宏。
BIOS.H	说明调用 IBM—PC ROM BIOS 子程序的各个函数。
CONIO.H	说明调用 DOS 控制台 I/O 子程序的各个函数。
CTYPE.H	包含有关字符分类及转换的名类信息(如 isalpha 和 toascii 等)。
DIR.H	包含有关目录和路径的结构、宏定义和函数。
DOS.H	定义和说明 MSDOS 和 8086 调用的一些常量和函数。
ERRON.H	定义错误代码的助记符。
FCNTL.H	定义在与 open 库子程序连接时的符号常量。
FLOAT.H	包含有关浮点运算的一些参数和函数。

GRAPHICS.H　说明有关图形功能的各个函数，图形错误代码的常量定义，正对不同驱动程序的各种颜色值，及函数用到的一些特殊结构。

IO.H　包含低级 I/O 子程序的结构和说明。

LIMIT.H　包含各环境参数、编译时间限制、数的范围等信息。

MATH.H　说明数学运算函数，还定了 HUGE VAL 宏， 说明了 matherr 和 matherr 子程序用到的特殊结构。

MEM.H　说明一些内存操作函数(其中大多数也在 STRING.H 中说明)。

PROCESS.H　说明进程管理的各个函数，spawn…和 EXEC …函数的结构说明。

SETJMP.H　定义 longjmp 和 setjmp 函数用到的 jmp buf 类型， 说明这两个函数。

SHARE.H　定义文件共享函数的参数。

SIGNAL.H　定义SIG[ZZ(Z] [ZZ)]IGN和SIG[ZZ(Z] [ZZ)]DFL常量,说明rajse和signal两个函数。

STDARG.H　定义读函数参数表的宏。(如 vprintf,vscarf 函数)。

STDDEF.H　定义一些公共数据类型和宏。

STDIO.H　定义 Kernighan 和 Ritchie 在 Unix System V 中定义的标准和扩展的类型和宏。还定义标准 I/O 预定义流：stdin,stdout 和 stderr，说明 I/O 流子程序。

STDLIB.H　说明一些常用的子程序：转换子程序、搜索/ 排序子程序等。

STRING.H　说明一些串操作和内存操作函数。

SYS\STAT.H　定义在打开和创建文件时用到的一些符号常量。

SYS\TYPES.H　说明 ftime 函数和 timeb 结构。

SYS\TIME.H　定义时间的类型 time[ZZ(Z] [ZZ)]t。

TIME.H　定义时间转换子程序 asctime、localtime 和 gmtime 的结构，ctime、difftime、 gmtime、 localtime 和 stime 用到的类型，并提供这些函数的原型。

VALUE.H　定义一些重要常量， 包括依赖于机器硬件的和为与 Unix System V 相兼容而说明的一些常量，包括浮点和双精度值的范围。

第三章 C 语言常见问题集

1、64 位机上的 64 位类型是什么样的？

C99 标准定义了 long long 类型，其长度可以保证至少 64 位，这种类型在某些编译器上实现已经颇有时日了。其它的编译器则实现了类似 longlong 的扩展。另一方面，也可以实现 16 位的短整型、32 位的整型和 64 位的长整型，有些编译器正是这样做的。

2、extern 在函数声明中是什么意思？

它可以用作一种格式上的提示表明函数的定义可能在另一个源文件中，但在

```
extern int f();
```

和

```
int f();
```

之间并没有实质的区别。

3、 以下的初始化有什么区别？char a[] = "string literal"; char *p = "string literal"; 当我向 p[i] 赋值的时候，我的程序崩溃了。

字符串常量有两种稍有区别的用法。用作数组初始值(如同在 char a[] 的声明中)，它指明该数组中字符的初始值。其它情况下，它会转化为一个无名的静态字符数组，可能会存储在只读内存中，这就是造成它不一定能被修改。在表达式环境中，数组通常被立即转化为一个指针，因此第二个声明把 p 初始化成指向无名数组的第一个元素。

为了编译旧代码，有的编译器有一个控制字符串是否可写的开关。

4、 声明 struct x1 { . . . }; 和 typedef struct { . . . } x2; 有什么不同？

第一种形式声明了一个“结构标签”；第二种声明了一个“类型定义”。主要的区别是在后文中你需要用“struct x1” 引用第一种，而用“x2” 引用第二种。也就是说，第二种声明更像一种抽象类新—– 用户不必知道它是一个结构，而在声明它的实例时也不需要使用 struct 关键字。

5、 为什么 struct x { . . . }; x thestruct; 不对？

C 不是 C++。结构标签不能自动生成类型。

6、 我遇到这样声明结构的代码：struct name { int namelen; char namestr[1];}; 然后又使用一些内存分配技巧使 namestr 数组用起来好像有多个元素。这样合法和可移植吗？

这种技术十分普遍，尽管 Dennis Ritchie 称之为“和 C 实现的无保证的亲密接触”。官方的解释认定它没有严格遵守 C 标准，尽管它看来在所有的实现中都可以工作。仔细检查数组边界的编译器可能会发出警告。

另一种可能是把变长的元素声明为很大，而不是很小；

```
char namestr[MAXSIZE];
```

MAXSIZE 比任何可能存储的 name 值都大。但是，这种技术似乎也

不完全符合标准的严格解释。这些“亲密” 结构都必须小心使用，因为只有程序员知道它的大小，而编译器却一无所知。C99 引入了“灵活数组域” 概念，允许结构的最后一个域省略数组大小。这为类似问题提供了一个圆满的解决方案。

7、 如何向接受结构参数的函数传入常数值?

传统的 C 没有办法生成匿名结构值；你必须使用临时结构变量或一个小的结构生成函数。

8、 怎样从/向数据文件读/写结构?

用 fwrite() 写一个结构相对简单:

```
fwrite(&somestruct, sizeof somestruct, 1, fp);
```

对应的 fread() 调用可以再把它读回来。但是这样写出的文件却不能移植。同时注意如果结构包含任何指针，则只有指针值会被写入文件，当它们再次读回来的时候，很可能已经失效。最后，为了广泛的移植，你必须用“b” 标志打开文件;移植性更好的方案是写一对函数，用可移植(可能甚至是人可读) 的方式按域读写结构，尽管开始可能工作量稍大。

9、 我的编译器在结构中留下了空洞，这导致空间浪费而且无法与外部数据文件进行”二进制” 读写。能否关掉填充，或者控制结构域的对齐方式?

这些“空洞” 充当了“填充”，为了保持结构中后面的域的对齐，这也许是必须的。为了高效的访问，许多处理器喜欢(或要求) 多字节对象(例如，结构中任何大于 char 的类型) 不能处于随意的内存地址，而必须是 2 或 4 或对象大小的倍数。编译器可能提供一种扩展用于这种控制(可能是#pragma)，但是没有标准的方法。

10、 为什么 sizeof 返回的值大于结构的期望值，是不是尾部有填充?

为了确保分配连续的结构数组时正确对齐，结构可能有这种尾部填充。即使结构不是数组的成员，填充也会保持，以便 sizeof 能够总是返回一致的大小。

11、 如何确定域在结构中的字节偏移?

ANSI C 在<stddef.h> 中定义了 offsetof() 宏，用 offsetof(struct s, f) 可以计算出域 f 在结构 s 中的偏移量。如果出于某种原因，你需要自己实现这个功能，可以使用下边这样的代码:

```
#define offsetof(type, f) ((size_t) \
((char *)&((type *)0)->f - (char *)(type *)0))
```

这种实现不是 100% 的可移植；某些编译器可能会合法地拒绝接受。

12、 程序运行正确，但退出时却“core dump”了，怎么回事？

问题程序:

```
struct list {
char *item;
struct list *next;
}
/* 这里是 main 程序*/
main(argc, argv)
{ ... }
```

缺少的一个分号使 main() 被定义为返回一个结构。由于中间的注释行，这个联系不容易看出来。因为一般上，返回结构的函数在实现时，会加入一个隐含的返回指针，这个产生的 main() 函数代码试图接受三个参数，而实际上只有两个传入(这里，由 C 的启动代码传入)。

13、 枚举和一组预处理的#define 有什么不同？

只有很小的区别。C 标准中允许枚举和其它整形类别自由混用而不会出错。（但是，假如编译器不允许在未经明确类型转换的情况下混用这些类型，则聪明地使用枚举可以捕捉到某些程序错误。）

枚举的一些优点：自动赋值；调试器在检验枚举变量时，可以显示符号值；它们服从数据块作用域规则。（编译器也可以对在枚举变量被任意地和其它类型混用时，产生非重要的警告信息，因为这被认为是坏风格。）一个缺点是程序员不能控制这些对非重要的警告；有些程序员则反感于无法控制枚举变量的大小。

14、为什么这样的代码：a[i] = i++; 不能工作？

子表达式 i++ 有一个副作用——它会改变 i 的值——由于 i 在同一表达式的其它地方被引用，这会导致无定义的结果，无从判断该引用(左边的 a[i] 中)是旧值还是新值。(注意，尽管在 K&R 中建议这类表达式的行为不确定，但 C 标准却强烈声明它是无定义的

15、 使用我的编译器,下面的代码 int i=7; printf("%d\n", i++ *i++); 返回 49？不管按什么顺序计算，难道不该打印出 56 吗？

尽管后缀自加和后缀自减操作符++ 和-- 在输出其旧值之后才会执行运算，但这里的“之后”常常被误解。没有任何保证确保自增或自减会在输出变量原值之后和对表达式的其它部分进行计算之前立即进行。也不能保证变量的更新会在表达式“完成” (按照 ANSI C 的术语，在下一个“序列点” 之前，参见问题 3.7) 之前的某个时刻进行。本例中，编译器选择使用变量的旧值相乘以后再对二者进行自增运算。

包含多个不确定的副作用的代码的行为总是被认为未定义。(简单而言，“多个不确定副作用” 是指在同一个表达式中使用导致同一对象修改两次或修改以后又被引用的自增，自减和赋值操作符的任何组合。这是一个粗略的定义) 甚至都不要试图探究这些东

西在你的编译器中是如何实现的(这与许多 C 教科书上的弱智练习正好相反)；正如 K&R 明智地指出，“如果你不知道它们在不同的机器上如何实现，这样的无知可能恰恰会有助于保护你。”

16、 对于代码 int i = 3; i = i++; 不同编译器给出不同的结果，有的为 3，有的为 4，哪个是正确的？

没有正确答案;这个表达式无定义。同时注意,i++ 和++i 都不同于 i+1。如果你要使 i 自增 1，使用 i=i+1，i+=1，i++ 或++i,而不是任何组合，

17、 这是个巧妙的表达式：a ^= b ^= a ^= b 它不需要临时变量就可以交换 a 和 b 的值。

这不具有可移植性。它试图在序列点之间两次修改变量 a，而这是无定义的。

例如,有人报告如下代码:

```
int a = 123, b = 7654;
a ^= b ^= a ^= b;
```

在 SCO 优化 C 编译器(icc) 下会把 b 置为 123，把 a 置为 0。

18、 我可否用括号来强制执行我所需要的计算顺序？

一般来讲，不行。运算符优先级和括弧只能赋予表达是计算部分的顺序．在如下的代码中 f() + g() * h()尽管我们知道乘法运算在加法之前，但这并不能说明这三个函数哪个会被首先调用。如果你需要确保子表达式的计算顺序，你可能需要使用明确的临时变量和独立的语句。

19、 可是&& 和|| 运算符呢？我看到过类似 while((c = getchar()) != EOF && c != ’\n’) 的代码……

这些运算符在此处有一个特殊的“短路” 例外：如果左边的子表达式决定最终结果(即,真对于|| 和假对于&&) 则右边的子表达式不会计算。因此，从左至右的计算可以确保，对逗号表达式也是如此。而且，所有这些运算符(包括? :) 都会引入一个额外的内部序列点。

20、 我怎样才能理解复杂表达式？“序列点” 是什么？

序列点是一个时间点(在整个表达式全部计算完毕之后或在||、&&、?: 或逗号运算符处，或在函数调用之前)，此刻尘埃落定，所有的副作用都已确保结束。ANSI/ISO C 标准这样描述：在上一个和下一个序列点之间，一个对象所保存的值至多只能被表达式的计算修改一次。而且前一个值只能用于决定将要保存的值。第二句话比较费解。它说在一个表达式中如果某个对象需要写入，则在同一表达式中对该对象的访问应该只局限于直接用于计算将要写入的值。这条规则有效地限制了只有能确保在修改之前才访问变量的表达式为合法。例如 i = i+1 合法，而 a[i] = i++ 则非法。那么，对于 a[i] = i++; 我们不知道 a[] 的哪一个分量会被改写,但 i 的确会增加 1，对吗？不一定！如果一个表达式和程序变得未定义，则它的所有方面都会变成未定义

21、 ++i 和 i++ 有什么区别?

如果你的 C 语言书没有说明它们的区别，那么买一本好的。简单而言：++i 在 i 存储的值上增加一并向使用它的表达式“返回” 新的，增加后的值；而 i++ 对 i 增加一，但返回原来的是未增加的值。

22、 为什么如下的代码 int a = 100, b = 100; long int c = a * b; 不能工作?

根据 C 的内部类型转换规则，乘法是用 int 进行的，而其结果可能在转换为 long 型并赋给左边的 c 之前溢出或被截短。可以使用明确的类型转换，强迫乘法以 long 型进行:long int c = (long int)a * b;注意，(long int)(a * b) 不能达到需要的效果。当两个整数做除法而结果赋与一个浮点变量时，也有可能有同样类型的问题,解决方法也是类似的。

23、 *p++ 自增 p 还是 p 所指向的变量?

后缀++ 和-- 操作符本质上比前缀一目操作的优先级高，因此*p++ 和*(p++) 等价，它自增 p 并返回 p 自增之前所指向的值。要自增 p 指向的值，使用(*p)++，如果副作用的顺序无关紧要也可以使用++*p。

24、 我有一个 char * 型指针正巧指向一些 int 型变量，我想跳过它们。为什么如下的代码((int *)p)++; 不行?

在 C 语言中，类型转换意味着“把这些二进制位看作另一种类型，并作相应的对待”；这是一个转换操作符，根据定义它只能生成一个右值(rvalue)。而右值既不能赋值，也不能用 ++ 自增。(如果编译器支持这样的扩展，那要么是一个错误，要么是有意作出的非标准扩展。) 要达到你的目的可以用:p = (char *)((int *)p + 1);或者,因为 p 是 char * 型，直接用 p += sizeof(int);但是，在可能的情况下，你还是应该首先选择适当的指针类型，而不是一味地试图李代桃僵。

25、 我能否用 void 指针作为参数，使函数按引用接受一般指针?**

不可移植。C 中没有一般的指针的指针类型。void* 可以用作一般指针只是因为当它和其它类型相互赋值的时候，如果需要，它可以自动转换成其它类型；但是，如果试图这样转换所指类型为 void* 之外的类型的 void** 指针时，这个转换不能完成。

26、 我有一个函数 extern int f(int *); 它接受指向 int 型的指针。我怎样用引用方式传入一个常数？下面这样的调用 f(&5); 似乎不行。

在 C99 中，你可以使用“复合常量”:f((int[]){5});在 C99 之前，你不能直接这样做；你必须先定义一个临时变量，然后把它的地址传给函数:int five = 5;f(&five);

27、 我看到了用指针调用函数的不同语法形式。到底怎么回事?

最初，一个函数指针必须用* 操作符(和一对额外的括弧) “转换为” 一个“真正的” 函数才能调用:int r, func(), (*fp)() = func;r = (*fp)();而函数总是通过指针进行调用

的，所有“真正的” 函数名总是隐式的退化为指针(在表达式中，正如在初始化时一样。参见问题 1.14)。这个推论表明无论 fp 是函数名和函数的指针 r = fp();ANSI C 标准实际上接受后边的解释，这意味着* 操作符不再需要，尽管依然允许。

28、 臭名昭著的空指针到底是什么？

语言定义中说明，每一种指针类型都有一个特殊值—— “空指针” —— 它与同类型的其它所有指针值都不相同，它“与任何对象或函数的指针值都不相等”。也就是说，取地址操作符& 永远也不能得到空指针，同样对 malloc() 的成功调用也不会返回空指针，如果失败，malloc() 的确返回空指针，这是空指针的典型用法:表示“未分配” 或者“尚未指向任何地方” 的指针。空指针在概念上不同于未初始化的指针。空指针可以确保不指向任何对象或函数；而未初始化指针则可能指向任何地方。如上文所述，每种指针类型都有一个空指针，而不同类型的空指针的内部表示可能不尽相同。尽管程序员不必知道内部值，但编译器必须时刻明确需要那种空指针，以便在需要的时候加以区分

29、 怎样在程序里获得一个空指针？

根据语言定义，在指针上下文中的常数 0 会在编译时转换为空指针。也就是说，在初始化、赋值或比较的时候，如果一边是指针类型的值或表达式，编译器可以确定另一边的常数 0 为空指针并生成正确的空指针值。因此下边的代码段完全合法：

```
char *p = 0;
if(p != 0)
```

然而，传入函数的参数不一定被当作指针环境，因而编译器可能不能识别未加修饰的 0 “表示” 指针。在函数调用的上下文中生成空指针需要明确的类型转换,强制把 0 看作指针。例如，Unix 系统调用 execl 接受变长的以空指针结束的字符指针参数。它应该如下正确调用：

```
execl("/bin/sh", "sh", "-c", "date", (char *)0);
```

如果省略最后一个参数的(char *) 转换，则编译器无从知道这是一个空指针,从而当作一个 0 传入。。如果范围内有函数原型，则参数传递变为“赋值上下文”，从而可以安全省略多数类型转换，因为原型告知编译器需要指针，使之把未加修饰的 0 正确转换为适当的指针。函数原型不能为变长参数列表中的可变参数提供类型。在函数调用时对所有的空指针进行类型转换可能是预防可变参数和无原型函数出问题的最安全的办法。。

30、 用缩写的指针比较“if(p)” 检查空指针是否可靠？如果空指针的内部表达不是 0 会怎么样？

当 C 在表达式中要求布尔值时，如果表达式等于 0 则认为该值为假，否则为真。换言之，只要写出 if(expr)无论“expr” 是任何表达式，编译器本质上都会把它当 if((expr) != 0) 处理。如果用指针 p 代替“expr” 则 if(p) 等价于 if(p != 0)。而这是一个比较上下文，因此编译器可以看出 0 实际上是一个空指针常数，并使用正确的空指针值。这里没有任何欺骗；编译器就是这样工作的，并为、二者生成完全一样的代码。空指针的内部表达无关紧要。布尔否操作符! 可如下描述:

```
!expr 本质上等价于(expr)？0:1
或等价于((expr) == 0)
```

从而得出结论
if(!p) 等价于 if(p == 0)
类似 if(p) 这样的“缩写”，尽管完全合法，但被一些人认为是不好的风格(另外一些人认为恰恰是好的风格;

31、 NULL 是什么，它是怎么定义的?

作为一种风格，很多人不愿意在程序中到处出现未加修饰的 0。因此定义了预处理宏 NULL (在<stdio.h> 和其它几个头文件中) 为空指针常数，通常是 0 或者((void *)0)。希望区别整数 0 和空指针 0 的人可以在需要空指针的地方使用 NULL。使用 NULL 只是一种风格习惯; 预处理器把所有的 NULL 都还原回 0，而编译还是依照上文的描述处理指针上下文的 0。特别是，在函数调用的参数里，NULL 之前(正如在 0 之前) 的类型转换还是需要。对 0 和 NULL 都有效(带修饰的 NULL 和带修饰的 0 完全等价)。NULL 只能用作指针常数;

32、 在使用非全零作为空指针内部表达的机器上，NULL 是如何定义的?

跟其它机器一样: 定义为 0 (或某种形式的 0;)。当程序员请求一个空指针时，无论写“0”还是“NULL”，都是有编译器来生成适合机器的空指针的二进制表达形式。因此，在空指针的内部表达不为 0 的机器上定义 NULL 为 0 跟在其它机器上一样合法：编译器在指针上下文看到的未加修饰的 0 都会被生成正确的空指针。

33、 如果 NULL 定义成#define NULL ((char *)0) 难道不就可以向函数传入不加转换的 NULL 了吗?

一般情况下，不行。复杂之处在于，有的机器不同类型数据的指针有不同的内部表达。这样的 NULL 定义对于接受字符指针的的函数没有问题，但对于其它类型的指针参数仍然有问题(在缺少原型的情况下)，而合法的构造如
FILE *fp = NULL;
则会失败。
不过，ANSI C 允许 NULL 的可选定义
#define NULL ((void *)0)
除了潜在地帮助错误程序运行(仅限于使用同样类型指针的机器，因此帮助有限) 以外，这样的定义还可以发现错误使用 NULL 的程序(例如，在实际需要使用 ASCII NUL 字符的地方; 参见问题 5.7)。
无论如何，ANSI 函数原型确保大多数(尽管不是全部)指针参数在传入函数时正确转换。因此，这个问题有些多余。

34、 如果 NULL 和 0 作为空指针常数是等价的，那我到底该用哪一个呢?

许多程序员认为在所有的指针上下文中都应该使用 NULL，以表明该值应该被看作指针。另一些人则认为用一个宏来定义 0，只不过把事情搞得更复杂，反而令人困惑。因而倾向于使用未加修饰的 0。没有正确的答案。C 程序员应该明白，在指针

上下文中 NULL 和 0 是完全等价的，而未加修饰的 0 也完全可以接受。任何使用 NULL (跟 0 相对) 的地方都应该看作一种温和的提示，是在使用指针；程序员(和编译器都) 不能依靠它来区别指针 0 和整数 0。
在需要其它类型的 0 的时候，即便它可能工作也不能使用 NULL，因为这样做发出了错误的格式信息。(而且，ANSI 允许把 NULL 定义为((void *)0)，这在非指针的上下文中完全无效。特别是，不能在需要 ASCII 空字符(NUL) 的地方用 NULL。如果有必要，提供你自己的定义
#define NUL '\0'

35、 这有点奇怪。NULL 可以确保是 0，但空(null) 指针却不一定？
随便使用术语“null” 或“NULL” 时，可能意味着以下一种或几种含义:
1. 概念上的空指针,。它使用以下的东西实现的……
2. 空指针的内部(或运行期) 表达形式，这可能并不是全零，而且对不用的指针类型可能不一样。真正的值只有编译器开发者才关心。C 程序的作者永远看不到它们，因为他们使用……
3. 空指针常数，这是一个常整数 0。它通常隐藏在……
4. NULL 宏，它被定义为 0 (参见问题 5.4)。最后转移我们注意力到……
5. ASCII 空字符(NUL)，它的确是全零，但它和空指针除了在名称上以外，没有任何必然关系；而……
6. “空串” (null string)，它是内容为空的字符串("")。在 C 中使用空串这个术语可能令人困惑，因为空串包括空字符('\0'),但不包括空指针，这让我们绕了一个完整的圈子……
本文用短语“空指针” (“null pointer”，小写) 表示第一种含义，标识“0” 或短语“空指针常数” 表示含义 3，用大写 NULL 表示含义 4。

36、 我在一个源文件中定义了 char a[6]，在另一个中声明了 extern char *a 。为什么不行？
你在一个源文件中定义了一个字符串，而在另一个文件中定义了指向字符的指针。extern char * 的申明不能和真正的定义匹配。类型 T 的指针和类型 T 的数组并非同种类型。请使用 extern char a[ ]。

37、 在 C 语言中“指针和数组等价” 到底是什么意思？
在 C 语言中对数组和指针的困惑多数都来自这句话。说数组和指针“等价”不表示它们相同，甚至也不能互换。它的意思是说数组和指针的算法定义可以用指针方便的访问数组或者模拟数组。
特别地，等价的基础来自这个关键定义:
一个 T 的数组类型的左值如果出现在表达式中会蜕变为一个指向数组第一个成员的指针(除了三种例外情况)；结果指针的类型是 T 的指针。
这就是说，一旦数组出现在表达式中，编译器会隐式地生成一个指向数组第一个成员地指针，就像程序员写出了&a[0] 一样。例外的情况是，数组为 sizeof 或&

操作符的操作数，或者为字符数组的字符串初始值。
作为这个这个定义的后果，编译器并那么不严格区分数组下标操作符和指针。在形如 a[i] 的表达式中，根据上边的规则，数组蜕化为指针然后按照指针变量的方式如 p[i] 那样寻址，如问题 6.2 所述，尽管最终的内存访问并不一样。如果你把数组地址赋给指针：

```
p = a;
```

那么 p[3] 和 a[3] 将会访问同样的成员。

38、 我该如何动态分配多维数组？

传统的解决方案是分配一个指针数组，然后把每个指针初始化为动态分配的“列”。以下为一个二维的例子：

```
#include <stdlib.h>
int **array1 = malloc(nrows * sizeof(int *));
for(i = 0; i < nrows; i++)
array1[i] = malloc(ncolumns * sizeof(int));
```

当然，在真实代码中，所有的 malloc 返回值都必须检查。你也可以使用 sizeof(*array1) 和 sizeof(**array1) 代替 sizeof(int *) 和 sizeof(int)。
你可以让数组的内容连续，但在后来重新分配列的时候会比较困难，得使用一点指针算术：

```
int **array2 = malloc(nrows * sizeof(int *));
array2[0] = malloc(nrows * ncolumns * sizeof(int));
for(i = 1; i < nrows; i++)
array2[i] = array2[0] + i * ncolumns;
```

在两种情况下，动态数组的成员都可以用正常的数组下标 arrayx[i][j] 来访问 (for 0 <= i <nrows 和 0 <= j <ncolumns)。
如果上述方案的两次间接因为某种原因不能接受，你还可以同一个单独的动态分配的一维数组来模拟二维数组：

```
int *array3 = malloc(nrows * ncolumns * sizeof(int));
```

但是，你现在必须手工计算下标，用 array3[i * ncolumns + j] 访问第 i，j 个成员。使用宏可以隐藏显示的计算，但是调用它的时候要使用括号和逗号，这看起来不太象多维数组语法，而且宏需要至少访问一维。
另一种选择是使用数组指针：

```
int (*array4)[NCOLUMNS] = malloc(nrows * sizeof(*array4));
```

但是这个语法变得可怕而且运行时最多只能确定一维。
当然，使用这些技术，你都必须记住在不用的时候释放数组(这可能需要多个步骤)。而且你可能不能混用动态数组和传统的静态分配数组。最后，在 C99 中你可以使用变长数组。
所有这些技术都可以延伸到三维或更多的维数。
我怎样编写接受编译时宽度未知的二维数组的函数？
这并非易事。一种办法是传入指向[0][0] 成员的的指针和两个维数，然后“手工” 模拟数组下标。void f2(int *aryp, int nrows, int ncolumns)

```
{ ... array[i][j] is accessed as aryp[i * ncolumns + j] ... }
```

这个函数可以用数组如下调用：

```
f2(&array[0][0], NROWS, NCOLUMNS);
```

但是，必须注明的一点是，用这种方法通过“手工” 方式模拟下标的程序未能严格遵循 ANSI C 标准；根据官方的解释，当 x >= NCOLUMNS 时，访问 &array[0][0][x] 的结果未定义。

C99 允许变长数组，一旦接受 C99 扩展的编译器广泛流传以后，VLA 可能是首选的解决方案。gcc 支持可变数组已经有些时日了。

当你需要使用各种大小的多维数组的函数时，一种解决方案是象动态模拟所有的数组。

39、 我怎样在函数参数传递时混用静态和动态多维数组？

没有完美的方法。假设有如下声明

```
int array[NROWS][NCOLUMNS];
int **array1; /* 不齐的*/
int **array2; /* 连续的*/
int *array3; /* "变平的" */
int (*array4)[NCOLUMNS];
```

指针的初始值如问题 6.13 的程序片段，函数声明如下

```
void f1a(int a[][NCOLUMNS], int nrows, int ncolumns);
void f1b(int (*a)[NCOLUMNS], int nrows, int ncolumns);
void f2(int *aryp, int nrows, int ncolumns);
void f3(int **pp, int nrows, int ncolumns);
```

其中 f1a() 和 f1b() 接受传统的二维数组，f2() 接受“扁平的” 二维数组，f3() 接受指针的指针模拟的数组，下面的调用应该可以如愿运行：

```
f1a(array, NROWS, NCOLUMNS);
f1b(array, NROWS, NCOLUMNS);
f1a(array4, nrows, NCOLUMNS);
f1b(array4, nrows, NCOLUMNS);
f2(&array[0][0], NROWS, NCOLUMNS);
f2(*array, NROWS, NCOLUMNS);
f2(*array2, nrows, ncolumns);
f2(array3, nrows, ncolumns);
f2(*array4, nrows, NCOLUMNS);
f3(array1, nrows, ncolumns);
f3(array2, nrows, ncolumns);
```

下面的调用在大多数系统上可能可行，但是有可疑的类型转换，而且只有动态 ncolumns 和静态 NCOLUMNS 匹配才行：

```
f1a((int (*)[NCOLUMNS])(*array2), nrows, ncolumns);
f1a((int (*)[NCOLUMNS])(*array2), nrows, ncolumns);
f1b((int (*)[NCOLUMNS])array3, nrows, ncolumns);
f1b((int (*)[NCOLUMNS])array3, nrows, ncolumns);
```

同时必须注意向 f2() 传递&array[0][0] (或者等价的*array) 并不完全符合标准;如果你能理解为何上述调用可行且必须这样书写，而未列出的组合不行，那么你对 C 语言中的数组和指针就有了很好的理解了。

为免受这些东西的困惑，一种使用各种大小的多维数组的办法是令它们“全部” 动态分配，如问题 6.13 所述。如果没有静态多维数组—— 如果所有的数组都按 array1 和 array2 分配—— 那么所有的函数都可以写成 f3() 的形式。

40、 在调用 malloc() 的时候，错误“不能把 void * 转换为 int *” 是什么意思？

说明你用的是 C++ 编译器而不是 C 编译器。

41、 我见到了这样的代码 char *p = malloc(strlen(s) + 1); strcpy(p, s); 难道不应该是 malloc((strlen(s) + 1) * sizeof(char))？

永远也不必乘上 sizeof(char)，因为根据定义，sizeof(char) 严格为 1。另一方面，乘上 sizeof(char) 也没有害处，有时候还可以帮忙为表达式引入 size t 类型

42、 动态分配的内存一旦释放之后你就不能再使用，是吧？

是的。有些早期的 malloc() 文档提到释放的内存中的内容会“保留”，但这个欠考虑的保证并不普遍而且也不是 C 标准要求的。几乎没有那个程序员会有意使用释放的内存，但是意外的使用却是常有的事。考虑下面释放单链表的正确代码:

```
struct list *listp, *nextp;
for(listp = base; listp != NULL; listp = nextp) {
nextp = listp->next;
free(listp);
}
```

请注意如果在循环表达式中没有使用临时变量 nextp，而使用 listp = listp->next 会产生什么恶劣后果。

43、 为什么在调用 free() 之后指针没有变空？使用(赋值，比较) 释放之后的指针有多么不安全？

当你调用 free() 的时候，传入指针指向的内存被释放，但调用函数的指针值可能保持不变，因为 C 的按值传参语义意味着被调函数永远不会永久改变参数的值。严格的讲，被释放的指针值是无效的，对它的任何使用，即使没有解参照，也可能带来问题，尽管作为一种实现质量的表现，多数实现都不会对无伤大雅的无效指针使用产生例外。

44、 为什么 strcat(string, ’!’); 不行？

字符和字符串的区别显而易见，而 strcat() 用于连接字符串。

C 中的字符用它们的字符集值对应的小整数表示。字符串用字符数组表示; 通常你操作的是字符数组的第一个字符的指针。二者永远不能混用。要为一个字符串增加!，需要使用

```
strcat(string, "!");
```

45、 我在检查一个字符串是否跟某个值匹配。为什么这样不行？char *string; . . . if(string == "value") { /* string matches ”value” */ . . . }

C 中的字符串用字符的数组表示，而 C 语言从来不会把数组作为一个整体操作(赋值，比较等)。上面代码段中的== 操作符比较的是两个指针—— 指针变量 string 的值和字符串常数“value” 的指针值—— 看它们是否相等，也就是说，看它们是否指向同一个位置。它们可能并不相等，所以比较决不会成功。

要比较两个字符串，一般使用库函数 strcmp():

```
if(strcmp(string, "value") == 0) {
/* string matches "value" */
}
```

46、 如果我可以写 char a[] = "Hello, world!"; **为什么我不能写** char a[14]; a = "Hello, world!";

字符串是数组，而你不能直接用数组赋值。可以使用 strcpy() 代替:

```
strcpy(a, "Hello, world!");
```

47、 C 语言中布尔值的候选类型是什么？为什么它不是一个标准类型？我应该用#define 或 enum **定义** true **和** false **值吗？**

C 语言没有提供标准的布尔类型，部分因为选一个这样的类型涉及最好由程序员决定的空间/时间折衷。(使用 int 可能更快，而使用 char 可能更节省数据空间。然而，如果需要和 int 反复转换，那么小类型也可能生成更大或更慢的代码。)

使用#define 还是枚举常数定义 true/false 可以随便，无关大雅。使用以下任何一种形式

```
#define TRUE 1 #define YES 1
#define FALSE 0 #define NO 0
enum bool {false, true}; enum bool {no, yes};
```

或直接使用 1 和 0 ，只要在同一程序或项目中一致即可。如果你的调试器在查看变量的时候能够显示枚举常量的名字，可能使用枚举更好。

有些人更喜欢这样的定义

```
#define TRUE (1==1)
#define FALSE (!TRUE)
```

或者定义这样的“辅助” 宏

```
#define Istrue(e) ((e) != 0)
```

但这样于事无益，

48、 书写多语句宏的最好方法是什么？

通常的目标是书写一个象包含一个单独的函数调用语句的宏。这意味着“调用者” 需要提供最终的分号，而宏体则不需要。因此宏体不能为简单的括弧包围的

复合语句，因为如果这样，调用的时候就会发生语法错(明显是一个单独语句，但却多了一个额外的分号)，就像在 if/else 语句的 if 分支中多了一个 else 分句一样。所以，传统的结局方案就是这样使用:

```
#define MACRO(arg1, arg2) do { \
/* declarations */ \
stmt1; \
stmt2; \
/* ... */ \
} while(0) /* 没有结尾的; */
```

当调用者加上分号后，宏在任何情况下都会扩展为一个单独的语句。优化的编译器会去掉条件为 0 的“无效” 测试或分支，尽管 lint 可能会警告。

如果宏体内的语句都是简单语句，没有声明或循环，那么还有一种技术，就是写一个单独的，用一个或多个逗号操作符分隔的表达式。例如，问题的第一个 DEBUG() 宏。这种技术还可以“返回” 一个值。

49、 我第一次把一个程序分成多个源文件，我不知道该把什么放到.c 文件，把什么放到.h 文件。(“.h” 到底是什么意思？)

作为一般规则，你应该把这些东西放入头(.h) 文件中:

- 宏定义(预处理#defines)
- 结构、联合和枚举声明
- typedef 声明
- 外部函数声明
- 全局变量声明

当声明或定义需要在多个文件中共享时，尤其需要把它们放入头文件中。特别是，永远不要把外部函数原型放到.c 文件中。另一方面，如果定义或声明为一个.c 文件私有，则最好留在.c 文件中。

50、 一个头文件可以包含另一头文件吗？

这是个风格问题，因此有不少的争论。很多人认为“嵌套包含文件” 应该避免:盛名远播的“印第安山风格指南”(Indian Hill Style Guide,) 对此嗤之以鼻；它让相关定义更难找到；如果一个文件被包含了两次，它会导致重复定义错误；同时他会令 makefile 的人工维护十分困难。另一方面，它使模块化使用头文件成为一种可能(一个头文件可以包含它所需要的一切，而不是让每个源文件都包含需要的头文件)；类似 grep 的工具(或 tags 文件) 使搜索定义十分容易，无论它在哪里；一种流行的头文件定义技巧是:

```
#ifndef HFILENAME_USED
#define HFILENAME_USED
... 头文件内容...
#endif
```

每一个头文件都使用了一个独一无二的宏名。这令头文件可自我识别，以便可以安全的多次包含；而自动 Makefile 维护工具(无论如何，在大型项目中都是必不可少的) 可以很容易的处理嵌套包含文件的依赖问题。

附录一　C 经典程序 100 例

【程序 1】

题目：有 1、2、3、4 个数字，能组成多少个互不相同且无重复数字的三位数？都是多少？

1.程序分析：可填在百位、十位、个位的数字都是 1、2、3、4。组成所有的排列后再去掉不满足条件的排列。

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  int i,j,k;
  printf("\n");
  for(i=1;i<5;i++) /*以下为三重循环*/
    for(j=1;j<5;j++)
      for (k=1;k<5;k++)
      {
        if (i!=k&&i!=j&&j!=k) /*确保 i、j、k 三位互不相同*/
        printf("%d,%d,%d\n",i,j,k);
      }
  getch();
}
```

==========

【程序 2】

题目：企业发放的奖金根据利润提成。利润(I)低于或等于 10 万元时，奖金可提 10%；利润高于 10 万元，低于 20 万元时，低于 10 万元的部分按 10%提成，高于 10 万元的部分，可可提成 7.5%；20 万到 40 万之间时，高于 20 万元的部分，可提成 5%；40 万到 60 万之间时高于 40 万元的部分，可提成 3%；60 万到 100 万之间时，高于 60 万元的部分，可提成 1.5%，高于 100 万元时，超过 100 万元的部分按 1%提成，从键盘输入当月利润 I，求应发放奖金总数？

1.程序分析：请利用数轴来分界，定位。注意定义时需把奖金定义成长整型。

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  long int i;
  int bonus1,bonus2,bonus4,bonus6,bonus10,bonus;
```

```
  scanf("%ld",&i);
  bonus1=100000*0. 1;
  bonus2=bonus1+100000*0.75;
  bonus4=bonus2+200000*0.5;
  bonus6=bonus4+200000*0.3;
  bonus10=bonus6+400000*0.15;
  if(i<=100000)
    bonus=i*0.1;
    else if(i<=200000)
      bonus=bonus1+(i-100000)*0.075;
        else if(i<=400000)
          bonus=bonus2+(i-200000)*0.05;
            else if(i<=600000)
              bonus=bonus4+(i-400000)*0.03;
                else if(i<=1000000)
                  bonus=bonus6+(i-600000)*0.015;
                    else
                      bonus=bonus10+(i-1000000)*0.01;
  printf("bonus=%d",bonus);
  getch();
}
```

==========

【程序 3】

题目：一个整数，它加上 100 后是一个完全平方数，再加上 168 又是一个完全平方数，请问该数是多少?

1.程序分析：在 10 万以内判断，先将该数加上 100 后再开方，再将该数加上 268 后再开方，如果开方后

的结果满足如下条件，即是结果。请看具体分析：

2.程序源代码：

```
#include "math.h"
#include "stdio.h"
#include "conio.h"
main()
{
  long int i,x,y,z;
  for (i=1;i<100000;i++)
  {
    x=sqrt(i+100); /*x 为加上 100 后开方后的结果*/
    y=sqrt(i+268); /*y 为再加上 168 后开方后的结果*/
    if(x*x==i+100&&y*y==i+268) /*如果一个数的平方根的平方等于该数，这说明此数是完全平方数*/
    printf("\n%ld\n",i);
  }
  getch();
```

```
}
```

==========

【程序 4】

题目：输入某年某月某日，判断这一天是这一年的第几天？

1.程序分析：以 3 月 5 日为例，应该先把前两个月的加起来，然后再加上 5 天即本年的第几天，特殊

情况，闰年且输入月份大于 3 时需考虑多加一天。

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  int day,month,year,sum,leap;
  printf("\nplease input year,month,day\n");
  scanf("%d,%d,%d",&year,&month,&day);
  switch(month) /*先计算某月以前月份的总天数*/
  {
    case 1:sum=0;break;
    case 2:sum=31;break;
    case 3:sum=59;break;
    case 4:sum=90;break;
    case 5:sum=120;break;
    case 6:sum=151;break;
    case 7:sum=181;break;
    case 8:sum=212;break;
    case 9:sum=243;break;
    case 10:sum=273;break;
    case 11:sum=304;break;
    case 12:sum=334;break;
    default:printf("data error");break;
  }
  sum=sum+day; /*再加上某天的天数*/
  if(year%400==0||(year%4==0&&year%100!=0)) /*判断是不是闰年*/
    leap=1;
  else
    leap=0;
  if(leap==1&&month>2) /*如果是闰年且月份大于 2,总天数应该加一天*/
    sum++;
  printf("It is the %dth day.",sum);
  getch();
}
```

==========

【程序 5】

题目：输入三个整数 x,y,z，请把这三个数由小到大输出。

1.程序分析：我们想办法把最小的数放到 x 上，先将 x 与 y 进行比较，如果 x>y 则将 x 与 y 的值进行交换，

然后再用 x 与 z 进行比较，如果 x>z 则将 x 与 z 的值进行交换，这样能使 x 最小。

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  int x,y,z,t;
  scanf("%d%d%d",&x,&y,&z);
  if (x>y)
    {t=x;x=y;y=t;} /*交换 x,y 的值*/
  if(x>z)
    {t=z;z=x;x=t;} /*交换 x,z 的值*/
  if(y>z)
    {t=y;y=z;z=t;} /*交换 z,y 的值*/
  printf("small to big: %d %d %d\n",x,y,z);
  getch();
}
```

==========

【程序 6】

题目：用*号输出字母 C 的图案。

1.程序分析：可先用'*'号在纸上写出字母 C，再分行输出。

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  printf("Hello C-world!\n");
  printf(" ****\n");
  printf(" *\n");
  printf(" * \n");
  printf(" ****\n");
  getch();
}
```

==========

【程序 7】

题目：输出特殊图案，请在 c 环境中运行，看一看，Very Beautiful!

1.程序分析：字符共有 256 个。不同字符，图形不一样。

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
```

```
{
  char a=176,b=219;
  printf("%c%c%c%c%c\n",b,a,a,a,b);
  printf("%c%c%c%c%c\n",a,b,a,b,a);
  printf("%c%c%c%c%c\n",a,a,b,a,a);
  printf("%c%c%c%c%c\n",a,b,a,b,a);
  printf("%c%c%c%c%c\n",b,a,a,a,b);
  getch();
}
```

==========

【程序 8】

题目：输出 9*9 口诀。

1.程序分析：分行与列考虑，共 9 行 9 列，i 控制行，j 控制列。

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  int i,j,result;
  printf("\n");
  for (i=1;i<10;i++)
  {
    for(j=1;j<10;j++)
    {
      result=i*j;
      printf("%d*%d=%-3d",i,j,result); /*-3d 表示左对齐，占 3 位*/
    }
    printf("\n"); /*每一行后换行*/
  }
  getch();
}
```

==========

【程序 9】

题目：要求输出国际象棋棋盘。

1.程序分析：用 i 控制行，j 来控制列，根据 i+j 的和的变化来控制输出黑方格，还是白方格。

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  int i,j;
  for(i=0;i<8;i++)
  {
```

```
    for(j=0;j<8;j++)
      if((i+j)%2==0)
        printf("%c%c",219,219);
      else
        printf("  ");
    printf("\n");
  }
  getch();
}
```

==========

【程序 10】

题目：打印楼梯，同时在楼梯上方打印两个笑脸。

1.程序分析：用 i 控制行，j 来控制列，j 根据 i 的变化来控制输出黑方格的个数。

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  int i,j;
  printf("\1\1\n"); /*输出两个笑脸*/
  for(i=1;i<11;i++)
  {
    for(j=1;j<=i;j++)
      printf("%c%c",219,219);
    printf("\n");
  }
  getch();
}
```

==========

【程序 11】

题目：古典问题：有一对兔子，从出生后第 3 个月起每个月都生一对兔子，小兔子长到第三个月

后每个月又生一对兔子，假如兔子都不死，问每个月的兔子总数为多少？

1.程序分析： 兔子的规律为数列 1,1,2,3,5,8,13,21....

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  long f1,f2;
  int i;
  f1=f2=1;
  for(i=1;i<=20;i++)
  {
```

```
    printf("%12ld %12ld",f1,f2);
    if(i%2==0) printf("\n"); /*控制输出，每行四个*/
    f1=f1+f2; /*前两个月加起来赋值给第三个月*/
    f2=f1+f2; /*前两个月加起来赋值给第三个月*/
  }
  getch();
}
```

==========

【程序 12】

题目：判断 101-200 之间有多少个素数，并输出所有素数。

1.程序分析：判断素数的方法：用一个数分别去除 2 到 sqrt(这个数)，如果能被整除，
则表明此数不是素数，反之是素数。

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
#include "math.h"
main()
{
  int m,i,k,h=0,leap=1;
  printf("\n");
  for(m=101;m<=200;m++)
  {
    k=sqrt(m+1);
    for(i=2;i<=k;i++)
    if(m%i==0)
    {
      leap=0;
      break;
    }
    if(leap)
    {
      printf("%-4d",m);
      h++;
      if(h%10==0)
        printf("\n");
    }
    leap=1;
  }
  printf("\nThe total is %d",h);
  getch();
}
```

==========

【程序 13】

题目：打印出所有的“水仙花数”，所谓“水仙花数”是指一个三位数，其各位数字立方和

等于该数

本身。例如：153 是一个“水仙花数”，因为 153=1 的三次方＋5 的三次方＋3 的三次方。

1.程序分析：利用 for 循环控制 100-999 个数，每个数分解出个位，十位，百位。

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  int i,j,k,n;
  printf("'water flower'number is:");
  for(n=100;n<1000;n++)
  {
    i=n/100;/*分解出百位*/
    j=n/10%10;/*分解出十位*/
    k=n%10;/*分解出个位*/
    if(i*100+j*10+k==i*i*i+j*j*j+k*k*k)
      printf("%-5d",n);
  }
  getch();
}
```

==========

【程序 14】

题目：将一个正整数分解质因数。例如：输入 90,打印出 90=2*3*3*5。

程序分析：对 n 进行分解质因数，应先找到一个最小的质数 k，然后按下述步骤完成：

(1)如果这个质数恰等于 n，则说明分解质因数的过程已经结束，打印出即可。

(2)如果 n<>k，但 n 能被 k 整除，则应打印出 k 的值，并用 n 除以 k 的商,作为新的正整数你 n,

重复执行第一步。

(3)如果 n 不能被 k 整除，则用 k+1 作为 k 的值,重复执行第一步。

2.程序源代码：

```
/* zheng int is divided yinshu*/
#include "stdio.h"
#include "conio.h"
main()
{
  int n,i;
  printf("\nplease input a number:\n");
  scanf("%d",&n);
  printf("%d=",n);
  for(i=2;i<=n;i++)
    while(n!=i)
    {
      if(n%i==0)
```

```
      {
        printf("%d*",i);
        n=n/i;
      }
      else
        break;
    }
  printf("%d",n);
  getch();
}
```

==========

【程序 15】

题目：利用条件运算符的嵌套来完成此题：学习成绩>=90 分的同学用 A 表示，60-89 分之间的用 B 表示，

60 分以下的用 C 表示。

1.程序分析：(a>b)?a:b 这是条件运算符的基本例子。

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  int score;
  char grade;
  printf("please input a score\n");
  scanf("%d",&score);
  grade=score>=90?'A':(score>=60?'B':'C');
  printf("%d belongs to %c",score,grade);
  getch();
}
```

==========

【程序 16】

题目：输入两个正整数 m 和 n，求其最大公约数和最小公倍数。

1.程序分析：利用辗除法。

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  int a,b,num1,num2,temp;
  printf("please input two numbers:\n");
  scanf("%d,%d",&num1,&num2);
  if(num1<num2)/*交换两个数，使大数放在 num1 上*/
  {
    temp=num1;
```

```
    num1=num2;
    num2=temp;
  }
  a=num1;b=num2;
  while(b!=0)/*利用辗除法，直到 b 为 0 为止*/
  {
    temp=a%b;
    a=b;
    b=temp;
  }
  printf("gongyueshu:%d\n",a);
  printf("gongbeishu:%d\n",num1*num2/a);
  getch();
}
```

==========

【程序 17】

题目：输入一行字符，分别统计出其中英文字母、空格、数字和其它字符的个数。

1.程序分析：利用 while 语句,条件为输入的字符不为'\n'.

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  char c;
  int letters=0,space=0,digit=0,others=0;
  printf("please input some characters\n");
  while((c=getchar())!='\n')
  {
    if(c>='a'&&c<='z'||c>='A'&&c<='Z')
      letters++;
      else if(c==' ')
        space++;
        else if(c>='0'&&c<='9')
          digit++;
        else
          others++;
  }
  printf("all in all:char=%d space=%d digit=%d others=%d\n",letters,
  space,digit,others);
  getch();
}
```

==========

【程序 18】

题目：求 s=a+aa+aaa+aaaa+aa...a 的值，其中 a 是一个数字。例如 2+22+222+2222+22222(此时
共有 5 个数相加)，几个数相加有键盘控制。

1.程序分析：关键是计算出每一项的值。

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  int a,n,count=1;
  long int sn=0,tn=0;
  printf("please input a and n\n");
  scanf("%d,%d",&a,&n);
  printf("a=%d,n=%d\n",a,n);
  while(count<=n)
  {
    tn=tn+a;
    sn=sn+tn;
    a=a*10;
    ++count;
  }
  printf("a+aa+...=%ld\n",sn);
  getch();
}
```

==========

【程序 19】

题目：一个数如果恰好等于它的因子之和，这个数就称为“完数”。例如 6=1＋2＋3.编程找出 1000 以内的所有完数。

1. 程序分析：请参照程序<--上页程序 14.

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  static int k[10];
  int i,j,n,s;
  for(j=2;j<1000;j++)
  {
    n=-1;
    s=j;
    for(i=1;i<j;i++)
    {
      if((j%i)==0)
      {
```

```
        n++;
        s=s-i;
        k[n]=i;
      }
    }
    if(s==0)
    {
      printf("%d is a wanshu",j);
      for(i=0;i<n;i++)
      printf("%d,",k[i]);
      printf("%d\n",k[n]);
    }
  }
  getch();
}
```

==========

【程序 20】

题目：一球从 100 米高度自由落下，每次落地后反跳回原高度的一半；再落下，求它在第 10 次落地时，共经过多少米？第 10 次反弹多高？

1.程序分析：见下面注释

2.程序源代码：

```
#include "stdio.h"
#include "stdio.h"
main()
{
  float sn=100.0,hn=sn/2;
  int n;
  for(n=2;n<=10;n++)
  {
    sn=sn+2*hn;/*第 n 次落地时共经过的米数*/
    hn=hn/2; /*第 n 次反跳高度*/
  }
  printf("the total of road is %f\n",sn);
  printf("the tenth is %f meter\n",hn);
  getch();
}
```

【程序 21】

题目：猴子吃桃问题：猴子第一天摘下若干个桃子，当即吃了一半，还不瘾，又多吃了一个第二天早上又将剩下的桃子吃掉一半，又多吃了一个。以后每天早上都吃了前一天剩下的一半零一个。到第 10 天早上想再吃时，见只剩下一个桃子了。求第一天共摘了多少。

1.程序分析：采取逆向思维的方法，从后往前推断。

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  int day,x1,x2;
  day=9;
  x2=1;
  while(day>0)
  {
    x1=(x2+1)*2;/*第一天的桃子数是第 2 天桃子数加 1 后的 2 倍*/
    x2=x1;
    day--;
  }
  printf("the total is %d\n",x1);
  getch();
}
```

==========

【程序 22】

题目：两个乒乓球队进行比赛，各出三人。甲队为 a,b,c 三人，乙队为 x,y,z 三人。已抽签决定

比赛名单。有人向队员打听比赛的名单。a 说他不和 x 比，c 说他不和 x,z 比，请编程序找出

三队赛手的名单。

1.程序分析：判断素数的方法：用一个数分别去除 2 到 sqrt(这个数)，如果能被整除，

则表明此数不是素数，反之是素数。

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  char i,j,k;/*i 是 a 的对手，j 是 b 的对手，k 是 c 的对手*/
  for(i='x';i<='z';i++)
    for(j='x';j<='z';j++)
    {
      if(i!=j)
      for(k='x';k<='z';k++)
      {
        if(i!=k&&j!=k)
        {
          if(i!='x'&&k!='x'&&k!='z')
            printf("order is a--%c\tb--%c\tc--%c\n",i,j,k);
         }
      }
    }
```

```
  getch();
}
```

==========

【程序 23】

题目：打印出如下图案（菱形）

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

1.程序分析：先把图形分成两部分来看待，前四行一个规律，后三行一个规律，利用双重 for 循环，第一层控制行，第二层控制列。

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  int i,j,k;
  for(i=0;i<=3;i++)
  {
    for(j=0;j<=2-i;j++)
      printf(" ");
    for(k=0;k<=2*i;k++)
      printf("*");
    printf("\n");
  }
  for(i=0;i<=2;i++)
  {
    for(j=0;j<=i;j++)
      printf(" ");
    for(k=0;k<=4-2*i;k++)
      printf("*");
    printf("\n");
  }
  getch();
}
```

==========

【程序 24】

题目：有一分数序列：2/1，3/2，5/3，8/5，13/8，21/13...求出这个数列的前 20 项之和。

1.程序分析：请抓住分子与分母的变化规律。

2.程序源代码：

```
#include "stdio.h"
```

```
#include "conio.h"
main()
{
  int n,t,number=20;
  float a=2,b=1,s=0;
  for(n=1;n<=number;n++)
  {
    s=s+a/b;
    t=a;a=a+b;b=t;/*这部分是程序的关键，请读者猜猜 t 的作用*/
  }
  printf("sum is %9.6f\n",s);
  getch();
}
```

==========

【程序 25】

题目：求 1+2!+3!+...+20!的和

1.程序分析：此程序只是把累加变成了累乘。

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  float n,s=0,t=1;
  for(n=1;n<=20;n++)
  {
    t*=n;
    s+=t;
  }
  printf("1+2!+3!...+20!=%e\n",s);
  getch();
}
```

==========

【程序 26】

题目：利用递归方法求 5!。

1.程序分析：递归公式：fn=fn_1*4!

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  int i;
  int fact();
  for(i=0;i<5;i++)
  printf("\40:%d!=%d\n",i,fact(i));
```

```
  getch();
}
int fact(j)
int j;
{
  int sum;
  if(j==0)
    sum=1;
  else
    sum=j*fact(j-1);
  return sum;
}
```

==========

【程序 27】

题目：利用递归函数调用方式，将所输入的 5 个字符，以相反顺序打印出来。

1.程序分析：

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  int i=5;
  void palin(int n);
  printf("\40:");
  palin(i);
  printf("\n");
  getch();
}
void palin(n)
int n;
{
  char next;
  if(n<=1)
  {
    next=getchar();
    printf("\n\0:");
    putchar(next);
  }
  else
  {
    next=getchar();
    palin(n-1);
    putchar(next);
  }
```

```
}
```

==========

【程序 28】

题目：有 5 个人坐在一起，问第五个人多少岁？他说比第 4 个人大 2 岁。问第 4 个人岁数，他说比第

3 个人大 2 岁。问第三个人，又说比第 2 人大两岁。问第 2 个人，说比第一个人大两岁。最后

问第一个人，他说是 10 岁。请问第五个人多大？

1.程序分析：利用递归的方法，递归分为回推和递推两个阶段。要想知道第五个人岁数，需知道

第四人的岁数，依次类推，推到第一人（10 岁），再往回推。

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
age(n)
int n;
{
  int c;
  if(n==1) c=10;
  else c=age(n-1)+2;
  return(c);
}
main()
{
  printf("%d",age(5));
  getch();
}
```

==========

【程序 29】

题目：给一个不多于 5 位的正整数，要求：一、求它是几位数，二、逆序打印出各位数字。

1. 程序分析：学会分解出每一位数，如下解释：(这里是一种简单的算法，师专数 002 班赵鑫提供)

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main( )
{
  long a,b,c,d,e,x;
  scanf("%ld",&x);
  a=x/10000;/*分解出万位*/
  b=x%10000/1000;/*分解出千位*/
  c=x%1000/100;/*分解出百位*/
  d=x%100/10;/*分解出十位*/
  e=x%10;/*分解出个位*/
```

```
  if (a!=0) printf("there are 5, %ld %ld %ld %ld %ld\n",e,d,c,b,a);
  else if (b!=0) printf("there are 4, %ld %ld %ld %ld\n",e,d,c,b);
    else if (c!=0) printf(" there are 3,%ld %ld %ld\n",e,d,c);
      else if (d!=0) printf("there are 2, %ld %ld\n",e,d);
        else if (e!=0) printf(" there are 1,%ld\n",e);
  getch();
}
```

==========

【程序 30】

题目：一个 5 位数，判断它是不是回文数。即 12321 是回文数，个位与万位相同，十位与千位相同。

1.程序分析：同 29 例

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main( )
{
  long ge,shi,qian,wan,x;
  scanf("%ld",&x);
  wan=x/10000;
  qian=x%10000/1000;
  shi=x%100/10;
  ge=x%10;
  if(ge==wan&&shi==qian)/*个位等于万位并且十位等于千位*/
    printf("this number is a huiwen\n");
  else
    printf("this number is not a huiwen\n");
  getch();
}
```

==========

【程序 31】

题目：请输入星期几的第一个字母来判断一下是星期几，如果第一个字母一样，则继续判断第二个字母。

1.程序分析：用情况语句比较好，如果第一个字母一样，则判断用情况语句或 if 语句判断第二个字母。

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
void main()
{
  char letter;
  printf("please input the first letter of someday\n");
  while((letter=getch())!='Y')/*当所按字母为 Y 时才结束*/
  {
```

```
    switch (letter)
    {
      case 'S':printf("please input second letter\n");
      if((letter=getch())=='a')
        printf("saturday\n");
        else if ((letter=getch())=='u')
          printf("sunday\n");
          else printf("data error\n");
      break;
      case 'F':printf("friday\n");break;
      case 'M':printf("monday\n");break;
      case 'T':printf("please input second letter\n");
      if((letter=getch())=='u')
        printf("tuesday\n");
        else if ((letter=getch())=='h')
          printf("thursday\n");
        else printf("data error\n");
      break;
      case 'W':printf("wednesday\n");break;
      default: printf("data error\n");
    }
  }
  getch();
}
```

==========

【程序 32】

题目：Press any key to change color, do you want to try it. Please hurry up!

1.程序分析：

2.程序源代码：

```
#include "conio.h"
#include "stdio.h"
void main(void)
{
  int color;
  for (color = 0; color < 8; color++)
  {
    textbackground(color);/*设置文本的背景颜色*/
    cprintf("This is color %d\r\n", color);
    cprintf("Press any key to continue\r\n");
    getch();/*输入字符看不见*/
  }
}
```

==========

【程序 33】

题目：学习 gotoxy()与 clrscr()函数
1.程序分析：
2.程序源代码：

```
#include "conio.h"
#include "stdio.h"
void main(void)
{
  clrscr();/*清屏函数*/
  textbackground(2);
  gotoxy(1, 5);/*定位函数*/
  cprintf("Output at row 5 column 1\n");
  textbackground(3);
  gotoxy(20, 10);
  cprintf("Output at row 10 column 20\n");
  getch();
}
```

==========

【程序 34】
题目：练习函数调用
1. 程序分析：
2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
void hello_world(void)
{
  printf("Hello, world!\n");
}
void three_hellos(void)
{
  int counter;
  for (counter = 1; counter <= 3; counter++)
    hello_world();/*调用此函数*/
}
void main(void)
{
  three_hellos();/*调用此函数*/
  getch();
}
```

==========

【程序 35】
题目：文本颜色设置
1.程序分析：
2.程序源代码：

```
#include "stdio.h"
```

```
#include "conio.h"
void main(void)
{
  int color;
  for (color = 1; color < 16; color++)
  {
    textcolor(color);/*设置文本颜色*/
    cprintf("This is color %d\r\n", color);
  }
  textcolor(128 + 15);
  cprintf("This is blinking\r\n");
  getch();
}
```

==========

【程序 36】

题目：求 100 之内的素数

1.程序分析：

2.程序源代码：

```
#include "stdio.h"
#include "math.h"
#define N 101
main()
{
  int i,j,line,a[N];
  for(i=2;i<N;i++) a[i]=i;
    for(i=2;i<sqrt(N);i++)
      for(j=i+1;j<N;j++)
      {
        if(a[i]!=0&&a[j]!=0)
          if(a[j]%a[i]==0)
            a[j]=0;
      }
  printf("\n");
  for(i=2,line=0;i<N;i++)
  {
    if(a[i]!=0)
    {
      printf("%5d",a[i]);
      line++;
    }
    if(line==10)
    {
      printf("\n");
      line=0;
```

```
    }
  }
  getch();
}
```

==========

【程序 37】

题目：对 10 个数进行排序

1.程序分析：可以利用选择法，即从后 9 个比较过程中，选择一个最小的与第一个元素交换，下次类推，即用第二个元素与后 8 个进行比较，并进行交换。

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
#define N 10
main()
{
  int i,j,min,tem,a[N];
  /*input data*/
  printf("please input ten num:\n");
  for(i=0;i<N;i++)
  {
    printf("a[%d]=",i);
    scanf("%d",&a[i]);
  }
  printf("\n");
  for(i=0;i<N;i++)
    printf("%5d",a[i]);
  printf("\n");
  /*sort ten num*/
  for(i=0;i<N-1;i++)
  {
    min=i;
    for(j=i+1;j<N;j++)
      if(a[min]>a[j])
        min=j;
    tem=a[i];
    a[i]=a[min];
    a[min]=tem;
  }
  /*output data*/
  printf("After sorted \n");
  for(i=0;i<N;i++)
  printf("%5d",a[i]);
  getch();
}
```

==========

【程序 38】

题目：求一个 3*3 矩阵对角线元素之和

1.程序分析：利用双重 for 循环控制输入二维数组，再将 a[i][i]累加后输出。

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"

/* 如果使用的是 TC 系列编译器则可能需要添加下句 */
static void dummyfloat(float *x){ float y; dummyfloat(&y);}

main()
{
  float a[3][3],sum=0;
  int i,j;
  printf("please input rectangle element:\n");
  for(i=0;i<3;i++)
    for(j=0;j<3;j++)
      scanf("%f",&a[i][j]);
  for(i=0;i<3;i++)
    sum=sum+a[i][i];
  printf("duijiaoxian he is %6.2f",sum);
  getch();
}
```

==========

【程序 39】

题目：有一个已经排好序的数组。现输入一个数，要求按原来的规律将它插入数组中。

1. 程序分析：首先判断此数是否大于最后一个数，然后再考虑插入中间的数的情况，插入后

此元素之后的数，依次后移一个位置。

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  int a[11]={1,4,6,9,13,16,19,28,40,100};
  int temp1,temp2,number,end,i,j;
  printf("original array is:\n");
  for(i=0;i<10;i++)
    printf("%5d",a[i]);
  printf("\n");
  printf("insert a new number:");
  scanf("%d",&number);
  end=a[9];
```

```
  if(number>end)
    a[10]=number;
  else
  {
    for(i=0;i<10;i++)
    {
      if(a[i]>number)
      {
        temp1=a[i];
        a[i]=number;
        for(j=i+1;j<11;j++)
        {
          temp2=a[j];
          a[j]=temp1;
          temp1=temp2;
        }
        break;
      }
    }
  }
  for(i=0;i<11;i++)
    printf("%6d",a[i]);
  getch();
}
```

==========

【程序 40】

题目：将一个数组逆序输出。

1.程序分析：用第一个与最后一个交换。

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
#define N 5
main()
{
  int a[N]={9,6,5,4,1},i,temp;
  printf("\n original array:\n");
  for(i=0;i<N;i++)
    printf("%4d",a[i]);
  for(i=0;i<N/2;i++)
  {
    temp=a[i];
    a[i]=a[N-i-1];
    a[N-i-1]=temp;
  }
```

```
  printf("\n sorted array:\n");
  for(i=0;i<N;i++)
    printf("%4d",a[i]);
  getch();
}
```

【程序 41】

题目：学习 static 定义静态变量的用法

1.程序分析：

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
varfunc()
{
  int var=0;
  static int static_var=0;
  printf("\40:var equal %d \n",var);
  printf("\40:static var equal %d \n",static_var);
  printf("\n");
  var++;
  static_var++;
}
void main()
{
  int i;
  for(i=0;i<3;i++)
    varfunc();
  getch();
}
```

==========

【程序 42】

题目：学习使用 auto 定义变量的用法

1.程序分析：

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  int i,num;
  num=2;
  for(i=0;i<3;i++)
  {
    printf("\40: The num equal %d \n",num);
    num++;
    {
```

```
      auto int num=1;
      printf("\40: The internal block num equal %d \n",num);
      num++;
    }
  }
  getch();
}
```

==========

【程序 43】

题目：学习使用 static 的另一用法。

1.程序分析：

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  int i,num;
  num=2;
  for(i=0;i<3;i++)
  {
    printf("\40: The num equal %d \n",num);
    num++;
    {
      static int num=1;
      printf("\40:The internal block num equal %d\n",num);
      num++;
    }
  }
  getch();
}
```

==========

【程序 44】

题目：学习使用 external 的用法。

1.程序分析：

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
int a,b,c;
void add()
{
  int a;
  a=3;
  c=a+b;
}
```

```
void main()
{
  a=b=4;
  add();
  printf("The value of c is equal to %d\n",c);
  getch();
}
```

==========

【程序 45】

题目：学习使用 register 定义变量的方法。

1.程序分析：

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
void main()
{
  register int i;
  int tmp=0;
  for(i=1;i<=100;i++)
  tmp+=i;
  printf("The sum is %d\n",tmp);
  getch();
}
```

==========

【程序 46】

题目：宏#define 命令练习(1)

1.程序分析：

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
#define TRUE 1
#define FALSE 0
#define SQ(x) (x)*(x)
void main()
{
  int num;
  int again=1;
  printf("\40: Program will stop if input value less than 50.\n");
  while(again)
  {
    printf("\40:Please input number==>");
    scanf("%d",&num);
    printf("\40:The square for this number is %d \n",SQ(num));
    if(num>=50)
```

```
      again=TRUE;
    else
      again=FALSE;
  }
  getch();
}
```

==========

【程序 47】

题目：宏#define 命令练习(2)

1.程序分析：

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
/*宏定义中允许包含两道衣裳命令的情形，此时必须在最右边加上"\"*/
#define exchange(a,b) { \
                        int t;\
                        t=a;\
                        a=b;\
                        b=t;\
                      }
void main(void)
{
  int x=10;
  int y=20;
  printf("x=%d; y=%d\n",x,y);
  exchange(x,y);
  printf("x=%d; y=%d\n",x,y);
  getch();
}
```

==========

【程序 48】

题目：宏#define 命令练习(3)

1.程序分析：

2.程序源代码：

```
#define LAG >
#define SMA <
#define EQ ==
#include "stdio.h"
#include "conio.h"
void main()
{
  int i=10;
  int j=20;
  if(i LAG j)
```

```
    printf("\40: %d larger than %d \n",i,j);
    else if(i EQ j)
      printf("\40: %d equal to %d \n",i,j);
      else if(i SMA j)
        printf("\40:%d smaller than %d \n",i,j);
      else
        printf("\40: No such value.\n");
  getch();
}
```

==========

【程序 49】

题目：#if #ifdef 和#ifndef 的综合应用。

1. 程序分析：

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
#define MAX
#define MAXIMUM(x,y) (x>y)?x:y
#define MINIMUM(x,y) (x>y)?y:x
void main()
{
  int a=10,b=20;
#ifdef MAX
  printf("\40: The larger one is %d\n",MAXIMUM(a,b));
#else
  printf("\40: The lower one is %d\n",MINIMUM(a,b));
#endif
#ifndef MIN
  printf("\40: The lower one is %d\n",MINIMUM(a,b));
#else
  printf("\40: The larger one is %d\n",MAXIMUM(a,b));
#endif
#undef MAX
#ifdef MAX
  printf("\40: The larger one is %d\n",MAXIMUM(a,b));
#else
  printf("\40: The lower one is %d\n",MINIMUM(a,b));
#endif
#define MIN
#ifndef MIN
  printf("\40: The lower one is %d\n",MINIMUM(a,b));
#else
  printf("\40: The larger one is %d\n",MAXIMUM(a,b));
#endif
```

```
  getch();
}
```

==========

【程序 50】

题目：#include 的应用练习

1.程序分析：

2.程序源代码：

test.h 文件如下:

```
#define LAG >
#define SMA <
#define EQ ==
```

主文件如下:

```
#include "test.h" /*一个新文件 50.c，包含 test.h*/
#include "stdio.h"
#include "conio.h"
void main()
{
  int i=10;
  int j=20;
  if(i LAG j)
    printf("\40: %d larger than %d \n",i,j);
    else if(i EQ j)
      printf("\40: %d equal to %d \n",i,j);
      else if(i SMA j)
        printf("\40:%d smaller than %d \n",i,j);
      else
        printf("\40: No such value.\n");
  getch();
}
```

==========

【程序 51】

题目：学习使用按位与 & 。

1.程序分析：0&0=0; 0&1=0; 1&0=0; 1&1=1

2.程序源代码：

```
#include "stdio.h"
main()
{
int a,b;
a=077;
b=a&3;
printf("\40: The a & b(decimal) is %d \n",b);
b&=7;
printf("\40: The a & b(decimal) is %d \n",b);
```

```
}
==============================================================
```

【程序 52】

题目：学习使用按位或 | 。

1.程序分析：0|0=0; 0|1=1; 1|0=1; 1|1=1

2.程序源代码：

```
#include "stdio.h"
main()
{
int a,b;
a=077;
b=a|3;
printf("\40: The a & b(decimal) is %d \n",b);
b|=7;
printf("\40: The a & b(decimal) is %d \n",b);
}
==============================================================
```

【程序 53】

题目：学习使用按位异或 ^ 。

1.程序分析：0^0=0; 0^1=1; 1^0=1; 1^1=0

2.程序源代码：

```
#include "stdio.h"
main()
{
int a,b;
a=077;
b=a^3;
printf("\40: The a & b(decimal) is %d \n",b);
b^=7;
printf("\40: The a & b(decimal) is %d \n",b);
}
==============================================================
```

【程序 54】

题目：取一个整数 a 从右端开始的 4～7 位。

程序分析：可以这样考虑：

(1)先使 a 右移 4 位。

(2)设置一个低 4 位全为 1,其余全为 0 的数。可用~(~0<<4)

(3)将上面二者进行&运算。

2.程序源代码：

```
main()
{
unsigned a,b,c,d;
scanf("%o",&a);
b=a>>4;
```

```
c=~(~0<<4);
d=b&c;
printf("%o\n%o\n",a,d);
}
```

==========

【程序 55】

题目：学习使用按位取反~。

1.程序分析：~0=1; ~1=0;

2.程序源代码：

```
#include "stdio.h"
main()
{
int a,b;
a=234;
b=~a;
printf("\40: The a's 1 complement(decimal) is %d \n",b);
a=~a;
printf("\40: The a's 1 complement(hexidecimal) is %x \n",a);
}
```

==========

【程序 56】

题目：画图，学用 circle 画圆形。

1.程序分析：

2.程序源代码：

```
/*circle*/
#include "graphics.h"
main()
{int driver,mode,i;
float j=1,k=1;
driver=VGA;mode=VGAHI;
initgraph(&driver,&mode,"");
setbkcolor(YELLOW);
for(i=0;i<=25;i++)
{
setcolor(8);
circle(310,250,k);
k=k+j;
j=j+0.3;
}
}
```

==========

【程序 57】

题目：画图，学用 line 画直线。

1.程序分析：

2.程序源代码：

```
#include "graphics.h"
main()
{int driver,mode,i;
float x0,y0,y1,x1;
float j=12,k;
driver=VGA;mode=VGAHI;
initgraph(&driver,&mode,"");
setbkcolor(GREEN);
x0=263;y0=263;y1=275;x1=275;
for(i=0;i<=18;i++)
{
setcolor(5);
line(x0,y0,x0,y1);
x0=x0-5;
y0=y0-5;
x1=x1+5;
y1=y1+5;
j=j+10;
}
x0=263;y1=275;y0=263;
for(i=0;i<=20;i++)
{
setcolor(5);
line(x0,y0,x0,y1);
x0=x0+5;
y0=y0+5;
y1=y1-5;
}
}
```

==========

【程序 58】

题目：画图，学用 rectangle 画方形。

1.程序分析：利用 for 循环控制 100-999 个数，每个数分解出个位，十位，百位。

2.程序源代码：

```
#include "graphics.h"
main()
{int x0,y0,y1,x1,driver,mode,i;
driver=VGA;mode=VGAHI;
initgraph(&driver,&mode,"");
setbkcolor(YELLOW);
x0=263;y0=263;y1=275;x1=275;
for(i=0;i<=18;i++)
{
```

```
setcolor(1);
rectangle(x0,y0,x1,y1);
x0=x0-5;
y0=y0-5;
x1=x1+5;
y1=y1+5;
}
settextstyle(DEFAULT_FONT,HORIZ_DIR,2);
outtextxy(150,40,"How beautiful it is!");
line(130,60,480,60);
setcolor(2);
circle(269,269,137);
}
```

==========

【程序 59】

题目：画图，综合例子。

1.程序分析：

2.程序源代码：

```
# define PAI 3.1415926
# define B 0.809
# include "graphics.h"
#include "math.h"
main()
{
int i,j,k,x0,y0,x,y,driver,mode;
float a;
driver=CGA;mode=CGAC0;
initgraph(&driver,&mode,"");
setcolor(3);
setbkcolor(GREEN);
x0=150;y0=100;
circle(x0,y0,10);
circle(x0,y0,20);
circle(x0,y0,50);
for(i=0;i<16;i++)
{
  a=(2*PAI/16)*i;
  x=ceil(x0+48*cos(a));
  y=ceil(y0+48*sin(a)*B);
  setcolor(2); line(x0,y0,x,y);}
setcolor(3);circle(x0,y0,60);
/* Make 0 time normal size letters */
settextstyle(DEFAULT_FONT,HORIZ_DIR,0);
outtextxy(10,170,"press a key");
```

```
getch();
setfillstyle(HATCH_FILL,YELLOW);
floodfill(202,100,WHITE);
getch();
for(k=0;k<=500;k++)
{
  setcolor(3);
  for(i=0;i<=16;i++)
  {
    a=(2*PAI/16)*i+(2*PAI/180)*k;
    x=ceil(x0+48*cos(a));
    y=ceil(y0+48+sin(a)*B);
    setcolor(2); line(x0,y0,x,y);
  }
  for(j=1;j<=50;j++)
  {
    a=(2*PAI/16)*i+(2*PAI/180)*k-1;
    x=ceil(x0+48*cos(a));
    y=ceil(y0+48*sin(a)*B);
    line(x0,y0,x,y);
  }
}
restorecrtmode();
}
```

==========

【程序 60】

题目：画图，综合例子。

1.程序分析：

2.程序源代码：

```
#include "graphics.h"
#define LEFT 0
#define TOP 0
#define RIGHT 639
#define BOTTOM 479
#define LINES 400
#define MAXCOLOR 15
main()
{
int driver,mode,error;
int x1,y1;
int x2,y2;
int dx1,dy1,dx2,dy2,i=1;
int count=0;
int color=0;
```

```
driver=VGA;
mode=VGAHI;
initgraph(&driver,&mode,"");
x1=x2=y1=y2=10;
dx1=dy1=2;
dx2=dy2=3;
while(!kbhit())
{
  line(x1,y1,x2,y2);
  x1+=dx1;y1+=dy1;
  x2+=dx2;y2+dy2;
  if(x1<=LEFT||x1>=RIGHT)
  dx1=-dx1;
  if(y1<=TOP||y1>=BOTTOM)
    dy1=-dy1;
  if(x2<=LEFT||x2>=RIGHT)
    dx2=-dx2;
  if(y2<=TOP||y2>=BOTTOM)
    dy2=-dy2;
  if(++count>LINES)
  {
    setcolor(color);
    color=(color>=MAXCOLOR)?0:++color;
  }
}
closegraph();
}
```

==========

【程序 71】

题目：编写 input()和 output()函数输入，输出 5 个学生的数据记录。

1.程序分析：

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
#define N 5
struct student
{
  char num[6];
  char name[8];
  int score[4];
}stu[N];
input(stu)
struct student stu[];
{
```

```
  int i,j;
  for(i=0;i<N;i++)
  {
    printf("\n please input %d of %d\n",i+1,N);
    printf("num: ");
    scanf("%s",stu[i].num);
    printf("name: ");
    scanf("%s",stu[i].name);
    for(j=0;j<3;j++)
    {
      printf("score %d.",j+1);
      scanf("%d",&stu[i].score[j]);
    }
    printf("\n");
  }
}
print(stu)
struct student stu[];
{
  int i,j;
  printf("\nNo. Name Sco1 Sco2 Sco3\n");
  for(i=0;i<N;i++)
  {
    printf("%-6s%-10s",stu[i].num,stu[i].name);
    for(j=0;j<3;j++)
    printf("%-8d",stu[i].score[j]);
    printf("\n");
  }
}
main()
{
  input();
  print();
  getch();
}
```

==========

【程序 72】

题目：创建一个链表。

1.程序分析：

2.程序源代码：

```
/*creat a list*/
#include "stdlib.h"
#include "stdio.h"
#include "conio.h"
```

```
struct list
{
  int data;
  struct list *next;
};
typedef struct list node;
typedef node *link;
void main()
{
  link ptr,head;
  int num,i;
  ptr=(link)malloc(sizeof(node));
  ptr=head;
  printf("please input 5 numbers==>\n");
  for(i=0;i<=4;i++)
  {
    scanf("%d",&num);
    ptr->data=num;
    ptr->next=(link)malloc(sizeof(node));
    if(i==4) ptr->next=NULL;
    else ptr=ptr->next;
  }
  ptr=head;
  while(ptr!=NULL)
  {
    printf("The value is ==>%d\n",ptr->data);
    ptr=ptr->next;
  }
  getch();
}
```

==========

【程序 73】

题目：反向输出一个链表。

1.程序分析：

2.程序源代码：

```
/*reverse output a list*/
#include "stdlib.h"
#include "stdio.h"
#include "conio.h"
struct list
{
  int data;
  struct list *next;
};
```

```
typedef struct list node;
typedef node *link;
void main()
{
  link ptr,head,tail;
  int num,i;
  tail=(link)malloc(sizeof(node));
  tail->next=NULL;
  ptr=tail;
  printf("\nplease input 5 data==>\n");
  for(i=0;i<=4;i++)
  {
    scanf("%d",&num);
    ptr->data=num;
    head=(link)malloc(sizeof(node));
    head->next=ptr;
    ptr=head;
  }
  ptr=ptr->next;
  while(ptr!=NULL)
  {
    printf("The value is ==>%d\n",ptr->data);
    ptr=ptr->next;
  }
  getch();
}
```

==========

【程序 74】

题目：连接两个链表。

1.程序分析：

2.程序源代码：

```
#include "stdlib.h"
#include "stdio.h"
#include "conio.h"
struct list
{
  int data;
  struct list *next;
};
typedef struct list node;
typedef node *link;
link delete_node(link pointer,link tmp)
{
  if (tmp==NULL) /*delete first node*/
```

```
  return pointer->next;
  else
  {
    if(tmp->next->next==NULL)/*delete last node*/
      tmp->next=NULL;
    else /*delete the other node*/
      tmp->next=tmp->next->next;
    return pointer;
  }
}
void selection_sort(link pointer,int num)
{
  link tmp,btmp;
  int i,min;
  for(i=0;i<num;i++)
  {
    tmp=pointer;
    min=tmp->data;
    btmp=NULL;
    while(tmp->next)
    {
      if(min>tmp->next->data)
      {
        min=tmp->next->data;
        btmp=tmp;
      }
      tmp=tmp->next;
    }
    printf("\40: %d\n",min);
    pointer=delete_node(pointer,btmp);
  }
}
link create_list(int array[],int num)
{
  link tmp1,tmp2,pointer;
  int i;
  pointer=(link)malloc(sizeof(node));
  pointer->data=array[0];
  tmp1=pointer;
  for(i=1;i<num;i++)
  {
    tmp2=(link)malloc(sizeof(node));
    tmp2->next=NULL;
    tmp2->data=array[i];
```

```
    tmp1->next=tmp2;
    tmp1=tmp1->next;
  }
  return pointer;
}
link concatenate(link pointer1,link pointer2)
{
  link tmp;
  tmp=pointer1;
  while(tmp->next)
    tmp=tmp->next;
  tmp->next=pointer2;
  return pointer1;
}
void main(void)
{
  int arr1[]={3,12,8,9,11};
  link ptr;
  ptr=create_list(arr1,5);
  selection_sort(ptr,5);
  getch();
}
```

==========

【程序 75】

题目：放松一下，算一道简单的题目。

1.程序分析：

2.程序源代码：

```
main()
{
  int i,n;
  for(i=1;i<5;i++)
  {
    n=0;
    if(i!=1)
      n=n+1;
    if(i==3)
      n=n+1;
    if(i==4)
      n=n+1;
    if(i!=4)
      n=n+1;
    if(n==3)
      printf("zhu hao shi de shi:%c",64+i);
  }
```

```
  getch();
}
```

==========

【程序 76】

题目：编写一个函数，输入 n 为偶数时，调用函数求 1/2+1/4+...+1/n,当输入 n 为奇数时，调用函数

1/1+1/3+...+1/n(利用指针函数)

1.程序分析：

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  float peven(),podd(),dcall();
  float sum;
  int n;
  while (1)
  {
    scanf("%d",&n);
    if(n>1)
      break;
  }
  if(n%2==0)
  {
    printf("Even=");
    sum=dcall(peven,n);
  }
  else
  {
    printf("Odd=");
    sum=dcall(podd,n);
  }
  printf("%f",sum);
  getch();
}
float peven(int n)
{
  float s;
  int i;
  s=1;
  for(i=2;i<=n;i+=2)
    s+=1/(float)i;
  return(s);
}
```

```
float podd(n)
int n;
{
  float s;
  int i;
  s=0;
  for(i=1;i<=n;i+=2)
    s+=1/(float)i;
  return(s);
}
float dcall(fp,n)
float (*fp)();
int n;
{
  float s;
  s=(*fp)(n);
  return(s);
}
```

==========

【程序 77】

题目：填空练习（指向指针的指针）

1.程序分析：

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  char *s[]={"man","woman","girl","boy","sister"};
  char **q;
  int k;
  for(k=0;k<5;k++)
  {          ;/*这里填写什么语句*/
    printf("%s\n",*q);
  }
  getch();
}
```

==========

【程序 78】

题目：找到年龄最大的人，并输出。请找出程序中有什么问题。

1.程序分析：

2.程序源代码：

```
#define N 4
#include "stdio.h"
#include "conio.h"
```

```
static struct man
{
  char name[20];
  int age;
}person[N]={"li",18,"wang",19,"zhang",20,"sun",22};
main()
{
  struct man *q,*p;
  int i,m=0;
  p=person;
  for (i=0;i<N;i++)
  {
    if(m<p->age)
    q=p++;
    m=q->age;
  }
  printf("%s,%d",(*q).name,(*q).age);
  getch();
}
```

==========

【程序 79】

题目：字符串排序。

1.程序分析：

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  char *str1[20],*str2[20],*str3[20];
  char swap();
  printf("please input three strings\n");
  scanf("%s",str1);
  scanf("%s",str2);
  scanf("%s",str3);
  if(strcmp(str1,str2)>0) swap(str1,str2);
  if(strcmp(str1,str3)>0) swap(str1,str3);
  if(strcmp(str2,str3)>0) swap(str2,str3);
  printf("after being sorted\n");
  printf("%s\n%s\n%s\n",str1,str2,str3);
  getch();
}
char swap(p1,p2)
char *p1,*p2;
{
```

```
  char *p[20];
  strcpy(p,p1);
  strcpy(p1,p2);
  strcpy(p2,p);
}
```

==========

【程序 80】

题目：海滩上有一堆桃子，五只猴子来分。第一只猴子把这堆桃子凭据分为五份，多了一个，这只

猴子把多的一个扔入海中，拿走了一份。第二只猴子把剩下的桃子又平均分成五份，又多了

一个，它同样把多的一个扔入海中，拿走了一份，第三、第四、第五只猴子都是这样做的，

问海滩上原来最少有多少个桃子？

1.程序分析：

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  int i,m,j,k,count;
  for(i=4;i<10000;i+=4)
  {
    count=0;
    m=i;
    for(k=0;k<5;k++)
    {
      j=i/4*5+1;
      i=j;
      if(j%4==0)
        count++;
      else
        break;
    }
    i=m;
    if(count==4)
    {
      printf("%d\n",count);
      break;
    }
  }
  getch();
}
```

【程序 81】

题目：809*??=800*??+9*??+1 其中??代表的两位数,8*??的结果为两位数，9*??的结果为 3 位数。求??代表的两位数，及 809*??后的结果。

1.程序分析：

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
output(long b,long i)
{
  printf("\n%ld/%ld=809*%ld+%ld",b,i,i,b%i);
}
main()
{
  long int a,b,i;
  a=809;
  for(i=10;i<100;i++)
  {
    b=i*a+1;
    if(b>=1000&&b<=10000&&8*i<100&&9*i>=100)
    output(b,i);
  }
  getch();
}
```

==========

【程序 82】

题目：八进制转换为十进制

1.程序分析：

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  char *p,s[6];int n;
  p=s;
  gets(p);
  n=0;
  while(*(p)!='\0')
  {
    n=n*8+*p-'0';
    p++;
  }
  printf("%d",n);
  getch();
}
```

==========

【程序 83】
题目：求 0—7 所能组成的奇数个数。
1.程序分析：
2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  long sum=4,s=4;
  int j;
  for(j=2;j<=8;j++)/*j is place of number*/
  {
    printf("\n%ld",sum);
    if(j<=2)
      s*=7;
    else
      s*=8;
    sum+=s;
  }
  printf("\nsum=%ld",sum);
  getch();
}
```

==========

【程序 84】
题目：一个偶数总能表示为两个素数之和。
1.程序分析：
2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
#include "math.h"
main()
{
  int a,b,c,d;
  scanf("%d",&a);
  for(b=3;b<=a/2;b+=2)
  {
    for(c=2;c<=sqrt(b);c++)
      if(b%c==0) break;
    if(c>sqrt(b))
      d=a-b;
    else
      break;
    for(c=2;c<=sqrt(d);c++)
      if(d%c==0) break;
```

```
    if(c>sqrt(d))
      printf("%d=%d+%d\n",a,b,d);
  }
  getch();
}
```

==========

【程序 85】

题目：判断一个素数能被几个 9 整除

1.程序分析：

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  long int m9=9,sum=9;
  int zi,n1=1,c9=1;
  scanf("%d",&zi);
  while(n1!=0)
  {
    if(!(sum%zi))
      n1=0;
    else
    {
      m9=m9*10;
      sum=sum+m9;
      c9++;
    }
  }
  printf("%ld,can be divided by %d \"9\"",sum,c9);
  getch();
}
```

==========

【程序 86】

题目：两个字符串连接程序

1.程序分析：

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  char a[]="acegikm";
  char b[]="bdfhjlnpq";
  char c[80],*p;
  int i=0,j=0,k=0;
```

```
  while(a[i]!='\0'&&b[j]!='\0')
  {
    if (a[i]<b[j])
    {
      c[k]=a[i];i++;
    }
    else
      c[k]=b[j++];
    k++;
  }
  c[k]='\0';
  if(a[i]=='\0')
    p=b+j;
  else
    p=a+i;
  strcat(c,p);
  puts(c);
  getch();
}
```

==========

【程序 87】

题目：回答结果（结构体变量传递）

1.程序分析：

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
struct student
{
  int x;
  char c;
}a;
main()
{
  a.x=3;
  a.c='a';
  f(a);
  printf("%d,%c",a.x,a.c);
  getch();
}
f(struct student b)
{
  b.x=20;
  b.c='y';
}
```

==========

【程序 88】

题目：读取 7 个数（1—50）的整数值，每读取一个值，程序打印出该值个数的＊。

1.程序分析：

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  int i,a,n=1;
  while(n<=7)
  {
    do
    {
      scanf("%d",&a);
    }while(a<1||a>50);
    for(i=1;i<=a;i++)
      printf("*");
    printf("\n");
    n++;
  }
  getch();
}
```

==========

【程序 89】

题目：某个公司采用公用电话传递数据，数据是四位的整数，在传递过程中是加密的，加密规则如下：

每位数字都加上 5,然后用和除以 10 的余数代替该数字，再将第一位和第四位交换，第二位和第三位交换。

1.程序分析：

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  int a,i,aa[4],t;
  scanf("%d",&a);
  aa[0]=a%10;
  aa[1]=a%100/10;
  aa[2]=a%1000/100;
  aa[3]=a/1000;
  for(i=0;i<=3;i++)
  {
    aa[i]+=5;
```

```
    aa[i]%=10;
  }
  for(i=0;i<=3/2;i++)
  {
    t=aa[i];
    aa[i]=aa[3-i];
    aa[3-i]=t;
  }
  for(i=3;i>=0;i--)
    printf("%d",aa[i]);
  getch();
}
```

==========

【程序 90】

题目：专升本一题，读结果。

1.程序分析：

2.程序源代码：

```
#include "stdio.h"
#define M 5
main()
{
  int a[M]={1,2,3,4,5};
  int i,j,t;
  i=0;j=M-1;
  while(i<j)
  {
    t=*(a+i);
    *(a+i)=*(a+j);
    *(a+j)=t;
    i++;j--;
  }
  for(i=0;i<M;i++)
    printf("%d",*(a+i));
  getch();
}
```

==========

【程序 91】

题目：时间函数举例 1

1.程序分析：

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
#include "time.h"
void main()
```

```
{
  time_t lt; /*define a longint time varible*/
  lt=time(NULL);/*system time and date*/
  printf(ctime(&lt)); /*english format output*/
  printf(asctime(localtime(&lt)));/*tranfer to tm*/
  printf(asctime(gmtime(&lt))); /*tranfer to Greenwich time*/
  getch();
}
```

==========

【程序 92】

题目：时间函数举例 2

1.程序分析：

2.程序源代码：

```
/*calculate time*/
#include "time.h"
#include "stdio.h"
#include "conio.h"
main()
{
  time_t start,end;
  int i;
  start=time(NULL);
  for(i=0;i<30000;i++)
    printf("\1\1\1\1\1\1\1\1\1\1\n");
  end=time(NULL);
  printf("\1: The different is %6.3f\n",difftime(end,start));
  getch();
}
```

==========

【程序 93】

题目：时间函数举例 3

1.程序分析：

2.程序源代码：

```
/*calculate time*/
#include "time.h"
#include "stdio.h"
#include "conio.h"
main()
{
  clock_t start,end;
  int i;
  double var;
  start=clock();
  for(i=0;i<10000;i++)
```

```
    printf("\1\1\1\1\1\1\1\1\1\1\n");
  end=clock();
  printf("\1: The different is %6.3f\n",(double)(end-start));
  getch();
}
```

==========

【程序 94】

题目：时间函数举例 4,一个猜数游戏，判断一个人反应快慢。（版主初学时编的）

1.程序分析：

2.程序源代码：

```
#include "time.h"
#include "stdlib.h"
#include "stdio.h"
#include "conio.h"
main()
{
  char c;
  clock_t start,end;
  time_t a,b;
  double var;
  int i,guess;
  srand(time(NULL));
  printf("do you want to play it.('y' or 'n') \n");
loop:
  while((c=getchar())=='y')
  {
    i=rand()%100;
    printf("\nplease input number you guess:\n");
    start=clock();
    a=time(NULL);
    scanf("%d",&guess);
    while(guess!=i)
    {
      if(guess>i)
      {
        printf("please input a little smaller.\n");
        scanf("%d",&guess);
      }
      else
      {
        printf("please input a little bigger.\n");
        scanf("%d",&guess);
      }
    }
```

```
    end=clock();
    b=time(NULL);
    printf("\1: It took you %6.3f seconds\n",var=(double)(end-start)/18.2);
    printf("\1: it took you %6.3f seconds\n\n",difftime(b,a));
    if(var<15)
      printf("\1\1 You are very clever! \1\1\n\n");
      else if(var<25)
        printf("\1\1 you are normal! \1\1\n\n");
      else
        printf("\1\1 you are stupid! \1\1\n\n");
    printf("\1\1 Congradulations \1\1\n\n");
    printf("The number you guess is %d",i);
  }
  printf("\ndo you want to try it again?(\"yy\".or.\"n\")\n");
  if((c=getch())=='y')
    goto loop;
}
============================================================
```

【程序 95】

题目：家庭财务管理小程序

1.程序分析：

2.程序源代码：

```
/*money management system*/
#include "stdio.h"
#include "dos.h"
#include "conio.h"
main()
{
  FILE *fp;
  struct date d;
  float sum,chm=0.0;
  int len,i,j=0;
  int c;
  char ch[4]="",ch1[16]="",chtime[12]="",chshop[16],chmoney[8];
pp:
  clrscr();
  sum=0.0;
  gotoxy(1,1);printf("|-----------------------------------------------------
------------------|");
  gotoxy(1,2);printf("| money management system(C1.0) 2000.03 |");
  gotoxy(1,3);printf("|-----------------------------------------------------
------------------|");
  gotoxy(1,4);printf("| -- money records -- | -- today cost list -- |");
                gotoxy(1,5);printf("|                ------------------------
```

```
|-------------------------------------|");
  gotoxy(1,6);printf("| date: -------------- | |");
  gotoxy(1,7);printf("| | | | |");
  gotoxy(1,8);printf("| -------------- | |");
  gotoxy(1,9);printf("| thgs: ------------------ | |");
  gotoxy(1,10);printf("| | | | |");
  gotoxy(1,11);printf("| ------------------ | |");
  gotoxy(1,12);printf("| cost: ---------- | |");
  gotoxy(1,13);printf("| | | | |");
  gotoxy(1,14);printf("| ---------- | |");
  gotoxy(1,15);printf("| | |");
  gotoxy(1,16);printf("| | |");
  gotoxy(1,17);printf("| | |");
  gotoxy(1,18);printf("| | |");
  gotoxy(1,19);printf("| | |");
  gotoxy(1,20);printf("| | |");
  gotoxy(1,21);printf("| | |");
  gotoxy(1,22);printf("| | |");
  gotoxy(1,23);printf("|-----------------------------------------------------
--------------------|");
  i=0;
  getdate(&d);
  sprintf(chtime,"%4d.%02d.%02d",d.da_year,d.da_mon,d.da_day);
  for(;;)
  {
    gotoxy(3,24);printf(" Tab __browse cost list Esc __quit");
    gotoxy(13,10);printf(" ");
    gotoxy(13,13);printf(" ");
    gotoxy(13,7);printf("%s",chtime);
    j=18;
    ch[0]=getch();
    if(ch[0]==27)
      break;
    strcpy(chshop,"");
    strcpy(chmoney,"");
    if(ch[0]==9)
    {
mm:
      i=0;
      fp=fopen("home.dat","r+");
      gotoxy(3,24);printf(" ");
      gotoxy(6,4);printf(" list records ");
      gotoxy(1,5);printf("|-------------------------------------|");
      gotoxy(41,4);printf(" ");
```

```
      gotoxy(41,5);printf(" |");
      while(fscanf(fp,"%10s%14s%f\n",chtime,chshop,&chm)!=EOF)
      {
        if(i==36)
        {
          getch();
          i=0;
        }
        if((i%36)<17)
        {
          gotoxy(4,6+i);
          printf(" ");
          gotoxy(4,6+i);
        }
        else
          if((i%36)>16)
          {
            gotoxy(41,4+i-17);
            printf(" ");
            gotoxy(42,4+i-17);
          }
        i++;
        sum=sum+chm;
        printf("%10s %-14s %6.1f\n",chtime,chshop,chm);
      }
      gotoxy(1,23);printf("|-----------------------------------------------
-----------------------|");
      gotoxy(1,24);printf("| |");
      gotoxy(1,25);printf("|-----------------------------------------------
-----------------------|");
      gotoxy(10,24);printf("total is %8.1f$",sum);
      fclose(fp);
      gotoxy(49,24);printf("press any key to.....");getch();goto pp;
    }
    else
    {
      while(ch[0]!='\r')
      {
        if(j<10)
        {
          strncat(chtime,ch,1);
          j++;
        }
        if(ch[0]==8)
```

```
  {
    len=strlen(chtime)-1;
    if(j>15)
    {len=len+1; j=11;}
    strcpy(ch1,"");
    j=j-2;
    strncat(ch1,chtime,len);
    strcpy(chtime,"");
    strncat(chtime,ch1,len-1);
    gotoxy(13,7);printf(" ");
  }
  gotoxy(13,7);printf("%s",chtime);ch[0]=getch();
  if(ch[0]==9)
    goto mm;
  if(ch[0]==27)
    exit(1);
}
gotoxy(3,24);printf(" ");
gotoxy(13,10);
j=0;
ch[0]=getch();
while(ch[0]!='\r')
{
  if (j<14)
  {
    strncat(chshop,ch,1);
    j++;
  }
  if(ch[0]==8)
  {
    len=strlen(chshop)-1;
    strcpy(ch1,"");
    j=j-2;
    strncat(ch1,chshop,len);
    strcpy(chshop,"");
    strncat(chshop,ch1,len-1);
    gotoxy(13,10);printf(" ");
  }
  gotoxy(13,10);printf("%s",chshop);ch[0]=getch();
}
gotoxy(13,13);
j=0;
ch[0]=getch();
while(ch[0]!='\r')
```

```
      {
        if (j<6)
        {
          strncat(chmoney,ch,1);
          j++;
        }
        if(ch[0]==8)
        {
          len=strlen(chmoney)-1;
          strcpy(ch1,"");
          j=j-2;
          strncat(ch1,chmoney,len);
          strcpy(chmoney,"");
          strncat(chmoney,ch1,len-1);
          gotoxy(13,13);printf(" ");
        }
        gotoxy(13,13);printf("%s",chmoney);ch[0]=getch();
      }
      if((strlen(chshop)==0)||(strlen(chmoney)==0))
        continue;
      if((fp=fopen("home.dat","a+"))!=NULL);
        fprintf(fp,"%10s%14s%6s",chtime,chshop,chmoney);
      fputc('\n',fp);
      fclose(fp);
      i++;
      gotoxy(41,5+i);
      printf("%10s %-14s %-6s",chtime,chshop,chmoney);
    }
  }
  getch();
}
```

==========

【程序 96】

题目：计算字符串中子串出现的次数

1.程序分析：

2.程序源代码：

```
#include "string.h"
#include "stdio.h"
#include "conio.h"
main()
{
  char str1[20],str2[20],*p1,*p2;
  int sum=0;
  printf("please input two strings\n");
```

```
  scanf("%s%s",str1,str2);
  p1=str1;p2=str2;
  while(*p1!='\0')
  {
    if(*p1==*p2)
    {
      while(*p1==*p2&&*p2!='\0')
      {
        p1++;
        p2++;
      }
    }
    else
      p1++;
    if(*p2=='\0')
      sum++;
    p2=str2;
  }
  printf("%d",sum);
  getch();
}
```

==========

【程序 97】

题目：从键盘输入一些字符，逐个把它们送到磁盘上去，直到输入一个#为止。

1.程序分析：

2.程序源代码：

```
#include "stdio.h"
main()
{
  FILE *fp;
  char ch,filename[10];
  scanf("%s",filename);
  if((fp=fopen(filename,"w"))==NULL)
  {
    printf("cannot open file\n");
    exit(0);
  }
  ch=getchar();
  ch=getchar();
  while(ch!='#')
  {
    fputc(ch,fp);
    putchar(ch);
    ch=getchar();
```

```
  }
  fclose(fp);
}
```

==========

【程序 98】

题目：从键盘输入一个字符串，将小写字母全部转换成大写字母，然后输出到一个磁盘文件“test”中保存。

输入的字符串以！结束。

1.程序分析：

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  FILE *fp;
  char str[100],filename[10];
  int i=0;
  if((fp=fopen("test","w"))==NULL)
  {
    printf("cannot open the file\n");
    exit(0);
  }
  printf("please input a string:\n");
  gets(str);
  while(str[i]!='!')
  {
    if(str[i]>='a'&&str[i]<='z')
      str[i]=str[i]-32;
    fputc(str[i],fp);
    i++;
  }
  fclose(fp);
  fp=fopen("test","r");
  fgets(str,strlen(str)+1,fp);
  printf("%s\n",str);
  fclose(fp);
  getch();
}
```

==========

【程序 99】

题目：有两个磁盘文件 A 和 B,各存放一行字母，要求把这两个文件中的信息合并（按字母顺序排列），

输出到一个新文件 C 中。

1.程序分析：

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
main()
{
  FILE *fp;
  int i,j,n,ni;
  char c[160],t,ch;
  if((fp=fopen("A","r"))==NULL)
  {
    printf("file A cannot be opened\n");
    exit(0);
  }
  printf("\n A contents are :\n");
  for(i=0;(ch=fgetc(fp))!=EOF;i++)
  {
    c[i]=ch;
    putchar(c[i]);
  }
  fclose(fp);
  ni=i;
  if((fp=fopen("B","r"))==NULL)
  {
    printf("file B cannot be opened\n");
    exit(0);
  }
  printf("\n B contents are :\n");
  for(i=0;(ch=fgetc(fp))!=EOF;i++)
  {
    c[i]=ch;
    putchar(c[i]);
  }
  fclose(fp);
  n=i;
  for(i=0;i<n;i++)
    for(j=i+1;j<n;j++)
      if(c[i]>c[j])
      {t=c[i];c[i]=c[j];c[j]=t;}
  printf("\n C file is:\n");
  fp=fopen("C","w");
  for(i=0;i<n;i++)
  {
    putc(c[i],fp);
    putchar(c[i]);
```

```
  }
  fclose(fp);
  getch();
}
```

==========

【程序 100】

题目：有五个学生，每个学生有 3 门课的成绩，从键盘输入以上数据（包括学生号，姓名，三门课成绩），计算出

平均成绩，况原有的数据和计算出的平均分数存放在磁盘文件"stud"中。

1.程序分析：

2.程序源代码：

```
#include "stdio.h"
#include "conio.h"
struct student
{
  char num[6];
  char name[8];
  int score[3];
  float avr;
}stu[5];
main()
{
  int i,j,sum;
  FILE *fp;
  /*input*/
  for(i=0;i<5;i++)
  {
    printf("\n please input No. %d score:\n",i);
    printf("stuNo:");
    scanf("%s",stu[i].num);
    printf("name:");
    scanf("%s",stu[i].name);
    sum=0;
    for(j=0;j<3;j++)
    {
      printf("score %d.",j+1);
      scanf("%d",&stu[i].score[j]);
      sum+=stu[i].score[j];
    }
    stu[i].avr=sum/3.0;
  }
  fp=fopen("stud","w");
  for(i=0;i<5;i++)
    if(fwrite(&stu[i],sizeof(struct student),1,fp)!=1)
```

```
        printf("file write error\n");
    fclose(fp);
    getch();
}
```

附录二 C语言函数库

这个在标准C库的中的函数原型列表是按照字母顺序的和按照原型功能分组的列表。

说明

注解列内容是非常简短的说明函数的使用。这个列表未完成，无论如何，它提供了C运行时库的主要函数的信息。它将，丝毫表示标准C库可用的函数允许你自己做更多的研究。一些C库函数记载在别处的可能在NiceBASIC是无效的。检则适当的引用文件可以找到更多信息。

注意: 以下的原型非官方的 NiceBASIC 的原型 (看引用文件)，无论如何，它们将你足够的信息完全地使用函数。

在引用文件列内容的文件名是你必段引用的，使用 # 引用 指示在你的程序开始处。如果你不包括适当的引用文件,程序不能编译,或它将编译明显正确的但在运行时给出一个不正确的结果。所有 C 运行时头文件位于 crt 目录下；例如，如果你指示头文件为 math.bi, 引用 "crt/math.bi" 或引用 "crt\math.bi", 我们最好引用 "crt.bi" 包括所有全部。

原型列内容有以下信息：

- 函数名称；
- 圆括号中是函数必须的参数，加上参数的数据类型；
- 函数的返回值数据类型。

例如 atoi(a 为 字符型 指针) 为 整数型 意思是函数 atoi 返回一个值数型为 **整数型** 并且需要一个 **字符型 指针** 作为它的参数。

字母顺序列表

名称	原型 (和参数)	引用文件	注解
abs	abs(n 为 整数型) 为 整数型	stdlib.bi	返回绝对值
acos	acos(a 为 双精度小数型) 为 双精度小数型	math.bi	返回反余弦 (角度为弧度)
asin	asin(a 为 双精度小数型)	math.bi	返回反正弦 (角度为弧度)

	为 双精度小数型		
atan	atan(a 为 双精度小数型) 为 双精度小数型	math.bi	返回反正切 (角度为弧度)
atan2	atan2(y 为 双精度小数型, x 为 双精度小数型) 为 双精度小数型	math.bi	返回反正切 (传递相对的 为 y 和 相邻的 为 x)
atoi	atoi(s 为 字符型 指针) 为 整数型	stdlib.bi	转换数字文本到一个整数型数值。
atof	atof(s 为 字符型 指针) 为 双精度小数型	stdlib.bi	转换数字文本到一个双精度小数型数值。
calloc	calloc(NumElts 为 整数型, EltSiz 为 整数型) 为 任意指针	stdlib.bi	申请内存。返回指针型指向一个缓冲区为一个数组有 NumElts 成员, 每个的尺寸是 EltSiz 字节。
ceil	ceil(d 为 双精度小数型) 为 双精度小数型	math.bi	返回最接近的整数大于传入的值。绝对取整
clearerr	clearerr(s 为 FILE 指针)	stdio.bi	清除错误指示在文件流 (读或写)。
cos	cos(ar 为 双精度小数型) 为 双精度小数型	math.bi	返回一个单位为弧度的角度的余弦值。
cosh	cosh(x 为 双精度小数型) 为 双精度小数型	math.bi	返回一个单位为弧度的角度的双曲余弦值。
div	div(num 为 整数型, denom 为 整数型) 为 div_t	stdlib.bi	返回除法的商和余数为结构类型 div_t.

ecvt	ecvt(x 为 双精度小数型)为 字符型 指针	math.bi	转换数值到字符串。
exit	exit(status 为整数型)	stdlib.bi	退出程序。这将刷新文件缓冲和关闭所有打开文件，并且运行所有函数调用 atexit().
exp	exp(a 为 双精度小数型) 为双精度小数型	math.bi	返回 e 值为底的参数为多少次幂 (反自然对数).
fabs	fabs(d 为 双精度小数型)为 双精度小数型	math.bi	返回绝对值 类型 为双精度小数型.
fclose	fclose(s 为 FILE 指针) 为 FILE 指针	stdio.bi	关闭文件。返回 0 如果成功否则文件尾
feof	feof(s 为 FILE 指针) 为 整数型	stdio.bi	返回文件结束标志。 (0 如果不在文件尾). 标志符将清除自己但不能被重置用 clearerr().
ferror	ferror(s 为 FILE 指针) 为 整数型	stdio.bi	返回流的错误标志 (0 如果无发生错误). 错误标志重置用 clearerr() 或 rewind().
fflush	fflush(s 为 FILE 指针) 为 整数型	stdio.bi	刷新(i.e. 删除) 一个流(使用标准输入刷新流从键盘缓冲区). 返回 0 如果 成功。
fgetc	fgetc(s 为 FILE 指针) 为 整数型	stdio.bi	单个字符输入 (为 ASCII) 从传入的流 (标准输入键盘缓冲区).
fgetpos	fgetpos(s 为 FILE 指针，c 为 fpos_t 指针) 为 整数型	stdio.bi	保存文件指针在流 s 的位置到 c.里。取文件流指针位置
fgets	fgets(b 为 字符型 指针，n 为 整数型，s 为 FILE 指针)为字符型 指针	stdio.bi	从流 s 读入 n-1 字符到缓冲区 b.
floor	floor(d 为 双精度小数型)	math.bi	返回最接近的整数小于传入的值。取整

	为 双精度小数型		
fmod	fmod(x 为 双精度小数型, y 为 双精度小数型) 为 双精度小数型	math.bi	计算 x 除以 y.的余数
fopen	fopen(file 为字符型 指针, mode 为 字符型 指针) 为 FILE 指针	stdio.bi	打开一个文件。传入 DOS 的文件名和一个代码指示读、写或追加. 代码 r 为读, w 为写, + 为读写, a 为追加 和 b 指示二进制模式.
fprintf	fprintf(s 为 FILE 指针, fmt 为 字符型 指针, ...) 为整数型	stdio.bi	输出到流 s 为多个项目为单个 % 标注在 fmt 对应参数列表的内容。
fputc	fputc(c 为 整数型, s 为 FILE 指针) 为整数型	stdio.bi	输出单个字符 c 到流 s.
fputs	fputs(b 为 字符型 指针, s 为 FILE 指针) 为 整数型	stdio.bi	发送字符流在 b 中的到流 s, 返回 0 如果操作失败。.
fread	fread(buf 为任意 指针, b 为 size_t, c 为 size_t, s 为 FILE 指针) 为整数型	stdio.bi	读取 c 个组尺寸数据为 b 字节从文件 s 到缓冲区 buf. 返回实际读取数据组的数目。
free	free(p 为 任意指针)	stdlib.bi	释放分配给指针型 p 的内存，允许这些内存可再使用。
freopen	freopen(file 为字符型 指针, mode 为 字符型 指针, s 为 FILE 指针) 为 FILE 指针	stdio.bi	打开一个文件为重定向一个流，例如 freopen("myfile", "w", stdout) 将重定向标准输出到打开的 "myfile".
frexp	frexp(x 为 双	math.bi	计算一个值 m 因为 x 等于 m 2 次幂. p 是指针型指向 m。求平方根

	精度小数型，p 为 整数型 指针) 为 双精度小数型		
fscanf	fscanf(s 为 FILE 指针, fmt 为 字符型 指针，...) 为整数型	stdio.bi	从 s 流读取为多个组有 % 标识在 fmt 相应列出的指针型
fseek	fseek(s 为 FILE 指针，offset 为 整数型, origin 为整数型) 为 整数型	stdio.bi	定位文件指针。起始点为 0, 1 或 2 开始，偏移字节数到流结束位置。
fsetpos	fsetpos(s 为 FILE 指针，p 为 fpos_t 指针) 为 整数型	stdio.bi	设置流 s 文件指针值为 p.指向的。
ftell	ftell(s 为 FILE 指针) 为 long	stdio.bi	定位 流 s 文件指针的位置.
fwrite	fwrite(buf 为任意 指针，b 为 整数型，c 为 整数型，s 为 FILE 指针) 为 整数型	stdio.bi	写出 c 个组 数据尺寸为 b 字节从缓冲区 buf 到文件 s. 返回实际写入数据组的数目。
getc	getc(s 为 FILE 指针) 为整数型	stdio.bi	单个输入文本 (in ASCII) 从传入的流. (标准输入键盘缓冲区)
getchar	getchar() 为整数型	stdio.bi	单个输入字符从标准输入
gets	gets(b 为 字符型 指针) 为字符型 指针	stdio.bi	读取一个字符流从标准输入直到 遇到 \n 或 文件尾.
hypot	hypot(x 为 双精度小数型，y 为 双精度小数型) 为 双精度小数型	math.bi	计算直角三角形的斜边通过侧边 x 和 y.

isalnum	isalnum(c 为整数型) 为整数型	ctype.bi	返回一个非零值如果字符 c 是字母数字。
isalpha	isalpha(c 为整数型) 为整数型	ctype.bi	返回一个非零值如果字符 c 是字母。
iscntrl	iscntrl(c 为整数型) 为整数型	ctype.bi	返回一个非零值如果字符 c 是一个控制符。
isdigit	isdigit(c 为整数型) 为整数型	ctype.bi	返回一个非零值如果字符 c 是数字。
isgraph	isgraph(c 为整数型) 为整数型	ctype.bi	返回一个非零值如果字符 c 是字母。
islower	islower(c 为整数型) 为整数型	ctype.bi	返回一个非零值如果字符 c 是一个小写字符。
isprint	isprint(c 为整数型) 为整数型	ctype.bi	返回一个非零值如果字符 c 是可以显示。
ispunct	ispunct(c 为整数型) 为整数型	ctype.bi	返回一个非零值如果字符 c 是标点字符。
isspace	isspace(c 为整数型) 为整数型	ctype.bi	返回一个非零值如果字符 c 是一个空格。
isupper	isupper(c 为整数型) 为整数型	ctype.bi	返回一个非零值如果字符 c 是一个大写字符。
isxdigit	isxdigit(c 为整数型) 为整数型	ctype.bi	返回一个非零值如果字符 c 是十六进制文本数字(0 至 F 或 f).
ldexp	ldexp(x 为双精度小数型, n 为整数型) 为双精度小数型	math.bi	返回 x 与 2 的 n 次幂的乘积.
ldiv	ldiv(num 为	stdlib.bi	返回除法的商与余数为一个结构类型 ldiv_t.

	long, denom 为 long) 为 ldiv_t		
log	log(a 为双精度小数型) 为双精度小数型	math.bi	返回参数的自然对数。
log10	log10(a 为双精度小数型) 为双精度小数型	math.bi	返回参数以 10 为底的对数。
malloc	malloc(bytes 为整数型) 为任意指针	stdlib.bi	申请内存。返回指针型指向指定大小的缓冲区。
modf	modf(d 为双精度小数型, p 为双精度小数型指针) 为双精度小数型	math.bi	返回浮点小数 d 的小数部分。 p 指向的整数部分表示为一个小数。
perror	perror(mess 为字符型指针)	stdio.bi	输出流 stderr 的一个信息传递给参数。
pow	pow(x 为双精度小数型, y 为双精度小数型) 为双精度小数型	math.bi	返回 x 的 y.次幂。
pow10	pow10(x 为双精度小数型) 为双精度小数型	math.bi	返回 10 的 x 次幂 (log10()的反函数).
printf	printf(fmt 为字符型指针, ...) 为整数型	stdio.bi	输出基本输出为许多组有单一 % 标识符在 fmt 匹配列表中的参数。
putc	putc(c 为整数型, s 为 FILE 指针) 为整数型	stdio.bi	输出单个字符到流 s.
putchar	putchar(c 为	stdio.bi	输出单个字符 c 到基本输出.

	整数型) 为 整数型		
puts	puts(b 为 字符型 指针) 为整数型	stdio.bi	发送字符流在 b 中的到基本输出, 返回 0 如果操作失败。
rand	rand() 为 整数型	stdlib.bi	返回一个伪随机数。种子是必须的。种子设置用 srand.
realloc	realloc(p 为任意 指针, newsize 为 size_t) 为 任意 指针	stdlib.bi	重新申请内存。返回指向一个缓冲区为改变目标尺寸的 p.
rewind	rewind(s 为 FILE 指针)	stdio.bi	流除一个文件流中指示的错误 (读 或写). 必需在读取一个改进文件之前。
scanf	scanf(fmt 为字符型 指针, ?) 为整数型	stdio.bi	从基本输入读取一些组为有 % 标识在 fmt 相应列出的指针。
sin	sin(ar 为 双精度小数型) 为双精度小数型	math.bi	返回一个以弧度为单位的角度的正弦。
sinh	sinh(x 为 双精度小数型) 为 双精度小数型	math.bi	返回一个以弧度为单位的角度的双曲正弦。
sprintf	sprintf(p 为 字符型 指针, fmt 为 字符型 指针, ...) 为整数型	stdio.bi	输出到字符 p 为多个组为单个 % 标识到 fmt 有匹配到参数列表的。
sqrt	sqrt(a 为 双精度小数型) 为双精度小数型	math.bi	返回传入值的平方根。如果值是负数则范围发生错误
srand	srand(seed 为无符整数型)	stdlib.bi	设置种子给随机数。一个可能的种子是当前时间.
sscanf	sscanf(b 为字符型 指针, fmt 为 字符型指针, ...) 为整	stdio.bi	从缓冲区 b 读取为多个组为有 % 标识到 fmt 相应列出的指针。

	数型		
strcat	strcat(s1 为字符型 指针, s2 为 字符型指针) 为 字符型指针	文本.bi	连接 (追加) 字符串 s2 到 s1.
strchr	strchr(s 为 字符型 指针, c 为 整数型) 为 字符型 指针	文本.bi	返回指针指向 c 在 s 中最先出现的 或 NULL 如果没有发现。
strcmp	strcmp(s1 为字符型 指针, s2 为 字符型指针) 为 整数型	string.bi	比较 字符串 s2 到 s1. 返回 0 或 最先不匹配的字符为 ASCII 值比较结果。
strcpy	strcpy(s1 为字符型 指针, s2 为 字符型指针) 为 字符型指针	string.bi	复制 s2 到 s1.
strcspn	strcspn(s1 为字符型 指针, s2 为 字符型指针) 为 整数型	string.bi	返回 s1 中一定数量的字符在遇到任何一个 s2.字符串之前。
strerror	strerror(n 为整数型) 为 字符型 指针	string.bi	返回一个指针指向系统发生的错误信息对应的传入的错误码。
strlen	strlen(s 为 字符型 指针) 为 整数型	string.bi	返回一个以空中止指向 s 的字符串的字节数(总数非空).
strncat	strncat(s1 为字符型 指针, s2 为 字符型指针, n 为整数型) 为 字符型指针	string.bi	连接(追加) n 个字从 字符串 s2 到 s1.
strncmp	strncmp(s1 为字符型 指针, s2 为 任意 指	string.bi	比较字符串 s2 的 n 字节数到同样的 s1. 返回 0 或 首个不匹配字符的 ASCII 值的负数差别。

	针, n 为 整数型) 为 整数型		
strncpy	strncpy(s1 为字符型 指针, s2 为 字符型指针, n 为整数型) 为 字符型指针	string.bi	复制 n 字节数从 s2 到 s1.
strpbrk	strpbrk(s1 为字符型 指针, s2 为 字符型指针) 为字符型 指针	string.bi	返回一个指针指向在 s1 中首个遇到字符串 s2.
strrchr	strrchr(s 为字符型 指针, c 为 整数型) 为字符型 指针	string.bi	返回指针指向在 s 中最后遇到的 c 或 NULL 如果查找失败。
strspn	strspn(s1 为字符型 指针, s2 为 字符型指针) 为 整数型	string.bi	返回 s1 中一定数量的字符在没遇到 s2.之前。
strstr	strstr(s1 为 字符型 指针, s2 为 字符型 指针) 为 字符型指针	string.bi	寻找字符串 s2 在 s1 中的位置，并且指针指向它的最先字符。
strtod	strtod(s 为 字符型 指针, p 为 字符型 指针) 为 双精度小数型	stdlib.bi	转换 字符串到双精度小数型, 假设字符串是数字形式的。
strtok	strtok(s1 为字符型 指针, s2 为 字符型指针) 为 字符型指针	string.bi	返回指针指向利用字符串 s1 的连续标记. 标记可以看作是分隔符被排列在 s2.
system	system(命令行 为 字符型 指针) 为 整数型	stdlib.bi	执行，从一个程序的内部，命令行地址指向操作系统写入的字符串。

tan	tan(ar 为双精度小数型) 为双精度小数型	math.bi	返回一个以弧度为单位的角度的正切。
tanh	tanh(x 为双精度小数型) 为双精度小数型	math.bi	返回一个以弧度为单位的角度的双曲正切。
tolower	tolower(c 为整数型) 为整数型	ctype.bi	转换字符从大写到小写(为 ASCII 码).
toupper	toupper(c 为整数型) 为整数型	ctype.bi	转换字符从小写到大写(为 ASCII 码).
ungetc	ungetc(c 为整数型, s 为 FILE 指针) 为整数型	stdio.bi	推一个字符 c 返回到流 s, 返回文件尾如果不成功，不推多于一个的字符。

缓冲区处理

#引用 "crt/string.bi"

原型 (和参数)	注解
memchr(s 为任意指针, c 为整数型, n 为 size_t) 为任意指针	在缓冲区中搜索一个字符
memcmp(s1 为任意指针, s2 为任意指针, n 为 size_t) 为整数型	比较两个缓冲区。
memcpy(dest 为任意指针, src 为任意指针, n 为 size_t) 为任意指针	复制一个缓冲区到另一个。
memmove(dest 为任意指针, src 为任意指针, n 为 size_t) 为任意指针	移到一定数量的字节从一个缓冲区到另一个。
memset(s 为任意指针, c 为整数型, n 为 size_t) 为任意指针	设置缓冲区的所有字节为指定的字符。

字符分类和转换

#引用 "crt/ctype.bi"

原型 (和参数)	注解
isalnum(c 为 整数型) 为 整数型	返回真如果 c 是 字母数字.
isalpha(c 为 整数型) 为 整数型	返回真如果 c 是 字母.
isascii(c 为 整数型) 为 整数型	返回真如果 c 是 ASCII .
iscntrl(c 为 整数型) 为 整数型	返回真如果 c 是 控制符.
isdigit(c 为 整数型) 为 整数型	返回真如果 c 是 十进制数.
isgraph(c 为 整数型) 为 整数型	返回真如果 c 是 图表字符.
islower(c 为 整数型) 为 整数型	返回真如果 c 是 小写字母.
isprint(c 为 整数型) 为 整数型	返回真如果 c 是 可输出字符.
ispunct(c 为 整数型) 为 整数型	返回真如果 c 是 标点字符.
isspace(c 为 整数型) 为 整数型	返回真如果 c 是 空格字符.
isupper(c 为 整数型) 为 整数型	返回真如果 c 是 大写字母.
isxdigit(c 为 整数型) 为 整数型	返回真如果 c 是 十六进制文本数字.
toascii(c 为 整数型) 为 整数型	转换 c 到 ASCII .
tolower(c 为 整数型) 为 整数型	转换 c 到小写.
toupper(c 为 整数型) 为 整数型	转换 c 到大小.

数据转换

#引用 "crt/stdlib.bi"

原型 (和参数)	注解
atof(string1 为 字符型 指针) 为 双精度小数型	转换字符串到浮点小数值。
atoi(string1 为 字符型 指针) 为 整数型	转换字符串到一个整数值。
atol(string1 为 字符型 指针) 为 整数型	轻换字符串到一个整数值。
itoa(value 为 整数型, 字符型 为 字符型 指针, radix 为整数型) 为 字符型 指针	转换一个整数值到一个字符串用 radix.指定的进制文本格式。
ltoa(value 为 long, 字符型 为 字符型 指针, radix 为 整数型) 为 字符型 指针	转换一个整数到字符串用给定的进制格式。

strtod(string1 为 字符型 指针, endptr 为 字符型 指针) 为 双精度小数型	转换字符串到一个浮点小数值。
strtol(string1 为 字符型 指针, endptr 为 字符型 指针, radix 为 整数型) 为 long	转换字符串到一个整数使用给定的进制格式。
strtoul(string1 为 字符型 指针, endptr 为 字符型 指针, radix 为 整数型) 为 ulong	转换字符串到无符号整数。

目录操作

```
#引用 "crt/io.bi"
```

原型 (和参数)	注解
_chdir(path 为 字符型 指针) 为 整数型	改变当前目录到给定的路径。
_getcwd(path 为 字符型 指针, numchars 为 整数型) 为字符型 指针	返回当前工作目录名。
_mkdir(path 为 字符型 指针) 为 整数型	创建一个目录用给定的路径名。
_rmdir(path 为 字符型 指针) 为 整数型	删除指定目录。

文件操作

```
#引用 "crt/sys/stat.bi"
#引用 "crt/io.bi"
```

原型 (和参数)	注解
chmod(path 为 字符型 指针, pmode 为 整数型) 为 整数型	改变一个文件的权限设置。
fstat(handle 为 整数型, buffer 为 类型 stat 指针) 为 整数型	取文件的状态信息。
remove(path 为 字符型 指针) 为 整数型	删除一个命名文件。
rename(oldname 为 字符型 指针, newname 为 字符型 指针) 为 整数型	重命名一个文件。
stat(path 为 字符型 指针, buffer 为 类型 stat 指针) 为整数型	取文件状态信息为一个命名文件。
umask(pmode 为 无符整数型) 为 无符整数型	设文件权限标志。

流 I/O

#引用 "crt/stdio.bi"

原型 (和参数)	注解
clearerr(file_pointer 为 FILE 指针)	清除流的错误标记
fclose(file_pointer 为 FILE 指针) 为 整数型	关闭一个文件
feof(file_pointer 为 FILE 指针) 为 整数型	检测文件结束是否发生在一个流。
ferror(file_pointer 为 FILE 指针) 为 整数型	检测在文件 I/0 期间是否有错误发生。
fflush(file_pointer 为 FILE 指针) 为 整数型	写出 (刷新) 缓冲区到文件。
fgetc(file_pointer 为 FILE 指针) 为 整数型	从流取一个字符。
fgetpos(file_pointer 为 FILE 指针, fpos_t current_pos) 为 整数型	取流的当前位置。
fgets(string1 为 字符型 指针, maxchar 为 整数型, file_pointer 为 FILE 指针) 为 字符型 指针	从一个文件中读字符串。
fopen(filename 为 字符型 指针, access_mode 为 字符型指针) 为 FILE 指针	打开一个文件给缓冲 I/0.
fprintf(file_pointer 为 FILE 指针, format_string 为字符型 指针, args) 为 整数型	写格式化文本输出到一个文件
fputc(c 为 整数型, file_pointer 为 FILE 指针) 为 整数型	写一个字符到流。
fputchar(c 为 整数型) 为 整数型	写一个字符到基本输出。
fputs(string1 为 字符型 指针, file_pointer 为 FILE 指针) 为 整数型	写一个字符串到一个流。
fread(buffer 为 字符型 指针, size 为 size_t count 为 size_t, file_pointer 为 FILE 指针) 为 size_t	读取无规则数据从流到一个缓冲区。
freopen(filename 为 字符型 指针, 访问权 为 字符型指针 mode, file_pointer 为 FILE 指针) 为 FILE 指针	重分配一个文件指针到一个不同的文件。
fscanf(file_pointer 为 FILE 指针, 取格式文本 为 字符型 指针字符型, args) 为 整数型	从一个流读取格式化输入。
fseek(file_pointer 为 FILE 指针, offset 为 long, origin 为 整数型) 为 整数型	设置文件中的当前位置到一个新的位置。
fsetpos(file_pointer 为 FILE 指针, current_pos 为	设置文件中的当前位置到一个新的位置。

fpos_t) 为 整数型	
ftell(file_pointer 为 FILE 指针) 为 long	取文件中的当前位置。
fwrite(buffer 为 字符型 指针, size 为 size_t, count 为 size_t file_pointer 为 FILE 指针) 为 size_t	写出不规则数据从一个缓冲区到一个流。
getc(file_pointer 为 FILE 指针) 为 整数型	从流读取一个字符。
getchar() 为 整数型	从基本输入读取一个字符。
gets(buffer 为 字符型 指针) 为 字符型 指针	从基本输入读取一行到一个缓冲区
printf(取格式文本 为 字符型 指针 _string, args) 为 整数型	写出格式化文本输出到基本输出。
putc(c 为 整数型, file_pointer 为 FILE 指针) 为 整数型	写出一个字符到一个流。
putchar(c 为 整数型) 为 整数型	写出一个字符到基本输出。
puts(string1 为 字符型 指针) 为 整数型	写出一个字符串到基本输出。
rewind(file_pointer 为 FILE 指针)	反绕一个文件。
scanf(format_string 为 字符型 指针, args) 为 整数型	读取格式化文本输入从基本输入。
setbuf(file_pointer 为 FILE 指针, buffer 为 字符型 指针)	设置一个新缓冲区给流。
setvbuf(file_pointer 为 FILE 指针, buffer 为 字符型指针, buf_type 为 整数型, buf 为 size_t size) 为 整数型	设置新缓冲区和控制流当中的缓冲等级。
sprintf(string1 为 字符型 指针, format_string 为字符型指针, args) 为 整数型	写出格式化文本输出到一个字符串。
sscanf(buffer 为 字符型 指针, format_string 为 字符型指针, args) 为 整数型	读取格式化输入从一个字符中。
tmpfile() 为 FILE 指针	打开一个临时文件。
tmpnam(file_name 为 字符型 指针) 为 字符型 指针	取临时文件名。
ungetc(c 为 整数型, file_pointer 为 FILE 指针) 为整数型	向回推字符到流缓冲区

底层 I/O

#引用 "crt/io.bi"

到目前为止只在 Win32 下, 链接到 MSVCRT.DLL

原型 (和参数)	注解
_close(handle 为 整数型) 为 整数型	关闭一个文件打开为无缓冲的 I/O.
_creat(filename 为 字符型 指针, pmode 为 整数型) 为整数型	创建一个新文件用指定的权限设置。
_eof(handle 为 整数型) 为 整数型	检测文件结束。
_lseek(handle 为 整数型, offset 为 long, origin 为整数型) 为 long	跳转到文件指定的位置。
_open(filename 为 字符型 指针, oflag 为 整数型, pmode 为 无符整数型) 为 整数型	打开一个文件为底层 I/O.
_read(handle 为 整数型, buffer 为 字符型 指针, length 为 无符整数型) 为 整数型	读取二进制数据到一个缓冲区。
_write(handle 为 整数型, buffer 为 字符型 指针, count 为 无符整数型) 为 整数型	写出二进制数据从一个缓冲区到一个文件。

数学

#引用 "crt/math.bi"

原型 (和参数)	注解
abs(n 为 整数型) 为 整数型	取一个整数的绝对值。
acos(x 为 双精度小数型) 为 双精度小数型	计算 x.的反余弦。
asin(x 为 双精度小数型) 为 双精度小数型	计算 x.的反正弦。
atan(x 为 双精度小数型) 为 双精度小数型	计算 x.的反正切。
atan2(y 为 双精度小数型, x 为 双精度小数型) 为 双精度小数型	计算 y/x.的反正切。
ceil(x 为 双精度小数型) 为 双精度小数型	取最小整数值超过 x.
cos(x 为 双精度小数型) 为 双精度小数型	计算弧度角的余弦。
cosh(x 为 双精度小数型) 为 双精度小数型	计算 x.的双曲线余弦。
div(number 为 整数型, denom 为 整数型) 为 div_t	用一个整数除以另一个。
exp(x 为 双精度小数型) 为 双精度小数型	计算 x.指数。
fabs(x 为 双精度小数型) 为 双精度小数型	计算 x.的绝对值。
floor(x 为 双精度小数型) 为 双精度小数型	取小于 x.的最大整数。

fmod(x 为 双精度小数型, y 为 双精度小数型) 为 双精度小数型	x 除以 y 的整数商并且返回余数。
frexp(x 为 双精度小数型, expptr 为 整数型 指针) 为 双精度小数型	分解 x 对数和无指数。
labs(n 为 long) 为 long	计算整数 n.的绝对值。
ldexp(x 为 双精度小数型, 反对数 为 整数型) 为 双精度小数型	重构 x 来自对数和二指数。
ldiv(number 为 long, denom 为 long) 为 ldiv_t	用整数除以另一个。
log(x 为 双精度小数型) 为 双精度小数型	计算自然对数(x).
log10(x 为 双精度小数型) 为 双精度小数型	计算以 10 为底 x.的对数。
modf(x 为 双精度小数型, intptr 为 双精度小数型 指针) 为 双精度小数型	分解 x 为小数和整数部分。
pow(x 为 双精度小数型, y 为 双精度小数型) 为 双精度小数型	计算 x 的 y.次幂。
rand() 为 整数型	取一个随机整数在 0 和 32.之间。
random(max_num 为 整数型) 为 整数型	取一个随机整数在 0 和 max_num.之前。
randomize()	设置一个随机数种子用于随机数发生器。
sin(x 为 双精度小数型) 为 双精度小数型	计算弧度角的正弦。
sinh(x 为 双精度小数型) 为 双精度小数型	计算 x.的双曲线正弦。
sqrt(x 为 双精度小数型) 为 双精度小数型	计算 x.的平方根。
srand(seed 为 无符整数型)	设置一个新种子给随机数发生器 (rand).
tan(x 为 双精度小数型) 为 双精度小数型	计算弧度角的正切。
tanh(x 为 双精度小数型) 为 双精度小数型	计算 x.的双曲线正切。

内存分配

#引用 "crt/stdlib.bi"

原型 (和参数)	注解
calloc(num 为 size_t elems, elem_size 为 size_t) 为 任意指针	申请一个数组并且初始化为零。
free(mem_address 为 任意 指针)	释放一个内存块。

malloc(num 为 size_t bytes) 为 任意 指针	申请一个内存块。
realloc(mem_address 为 任意 指针, newsize 为 size_t) 为任意 指针	重新申请 (调整尺寸) 一个内存块。

进程控制

#引用 "crt/stdlib.bi"

原型 (和参数)	注解
abort()	异常中止一个进程。
execl(path 为 字符型 指针, arg0 为 字符型 指针, arg1 为字符型 指针,..., NULL) 为 整数型	运行一个子进程 (传递命令行).
execlp(path 为 字符型 指针, arg0 为 字符型 指针, arg1 为 字符型 指针,..., NULL) 为 整数型	运行一个子进程(用 PATH, 传递命令行).
execv(path 为 字符型 指针, argv 为 字符型 指针) 为整数型	运行一个子进程(传递参数变量).
execvp(path 为 字符型 指针, argv 为 字符型 指针) 为整数型	运行一个子进程(用 PATH, 传递参数变量).
exit(status 为 整数型)	停止进程在刷新所有缓冲区之后。
getenv(varname 为 字符型 指针) 为 字符型 指针	取指定的环境变量
perror(string1 为 字符型 指针)	输出错误信息到相应的最后系统错误。
putenv(envstring 为 字符型 指针) 为 整数型	插入一个新的定义到环境变量表。
raise(signum 为 整数型) 为 整数型	产生一个 C 信号(异常).
system(string1 为 字符型 指针) 为 整数型	执行一个内置操作系统命令

搜索和排序

#引用 "crt/stdlib.bi"

注意: 比较 回调函数必需的搜索和排序必须声是为 cdecl. 它必须返回一个值 <0 如果第一个参数将被定位在下一个在排序数组之前， >0 如果第一个参数将被定位在下一个之后，和零如果它们的相对位置则无关紧要(相等值).

原型 (和参数)	注解

bsearch(key 为 任意 指针, base 为 任意 指针, num 为 size_t, 控制台_宽度 为 size_t, compare 为 函数(elem1 为 任意 指针, elem2 为 任意指针) 为 整数型) 为 任意 指针	执行二分检索
qsort(base 为 任意 指针, num 为 size_t, 控制台_宽度 为 size_t, compare 为 函数(elem1 为 任意 指针, elem2 为 任意 指针) 为 整数型)	使用快速排序算法排序一个数组。

字符串处理

#引用 "crt/string.bi"

原型 (和参数)	注解
stpcpy(dest 为 字符型 指针, src 为 字符型 指针) 为 字符型 指针	复制一个字符串到另一个。
strcmp(string1 为 字符型 指针, string2 为 字符型 指针) 为 整数型	比较 string1 和 string2 以字母顺序为规则
strcpy(string1 为 字符型 指针, string2 为 字符型 指针) 为 字符型 指针	复制 string2 到 string1.
strerror(errnum 为 整数型) 为 字符型 指针	取相应的错误信息用指定错误码。
strlen(string1 为 字符型 指针) 为 整数型	确定一个字符串的长度。
strncat(string1 为 字符型 指针, string2 为 字符型 指针, n 为 size_t) 为 字符型 指针	追加 n 个字符从 string2 到 string1.
strncmp(string1 为 字符型 指针, string2 为 字符型 指针, n 为 size_t) 为 整数型	比较两个字符串的开始 n 个字符。
strncpy(string1 为 字符型 指针, string2 为 字符型 指针, n 为 size_t) 为 字符型 指针	复制 string2 开始处 n 个字符到 string1.
strnset(string1 为 字符型 指针, c 为 整数型, size _t n) 为 字符型 指针	设置字符串开始处 n 字符为 c.
strrchr(string1 为 字符型 指针, c 为 整数型) 为字符型指针	在字符串中寻找最后一个出现的 c 字符。

时间

#引用 "crt/time.bi"

原型 (和参数)	注解
asctime(当前时间 为 类型 tm 指针) 为 字符型 指针	转换时间从类型 tm 到字符串
clock() 为 clock_t	取进程经过的时间在定时器里。
ctime(当前时间 为 time_t 指针) 为 字符型 指针	转换二进制时间到字符串。
difftime(time_t time2, time_t time1) 为 双精度小数型	计算两个时间的差值，单位为秒。
gmtime(当前时间 为 time_t 指针) 为 类型 tm 指针	取格林尼治时间 (GMT)在一个 tm 结构里。
localtime(当前时间 为 time_t 指针) 为 类型 tm 指针	取本地时间在一个 tm 结构里。
time(timeptr 为 time_t 指针) 为 time_t	取当前时间从 0 时 GMT 1/1/70 开始经历的秒数。.