高等学校教材

C程序设计

张长海　陈　娟　编著

高等教育出版社

内 容 提 要

本书以国际标准 ISO/IEC 9899:1999和国家标准 GB/T 15272－94定义的 C语言为载体,阐述基本的程序设计方法,并对相关的 C语言成分进行较严格的介绍。用 BNF表示 C语言的语法,引进 PAD图表示程序逻辑。全书共分十四章,主要内容包括：BNF、PAD图、程序设计方法、程序开发和结构化程序设计以及 C语言的各种词法单位、数据类型、语句、函数等。每章都包含大量例题,并附有大量习题,以利于读者提高程序设计能力和学习掌握相关语言概念。

本书最大的特点是以“程序设计”为主线,把重点放在讲述程序设计方法上。摈弃了目前各种程序设计书中流行的以“解释程序设计语言”为主的做法。全书整体结构良好,图文并茂,知识体系新颖完整,概念准确;注重对读者进行程序设计方法及算法的训练,力求体现“结构化程序设计”思想,注重培养和训练读者良好的程序设计风格。

本书可作为高等院校计算机系各专业“高级语言程序设计”、“C语言程序设计”、“程序设计基础”等课程的教材和参考书,也可供其他专业学生以及从事计算机工作的有关人员阅读参考。

策划编辑　倪文慧　责任编辑　武林晓　市场策划　陈　振
封面设计　王凌波　责任印制

出版发行　高等教育出版社　　购书热线　010-64054588
社　　址　北京市西城区德外大街 4号　　免费咨询　800-810-0598
邮政编码　100011　　网　　址　http://www.hep.edu.cn
总　　机　010-58581000　　http://www.hep.com.cn

经　　销　新华书店北京发行所
印　　刷

开　　本　787×1092　1/16　　版　　次　　年　月第 1版
印　　张　28.25　　印　　次　　年　月第　次印刷
字　　数　590 000　　定　　价　30.00元

物料号：15116－00

前　言

本书适用于“高级语言程序设计”或“程序设计基础”课程。该课程是计算机系的专业基础课，在计算机专业教学中占有重要地位。学好该课程既可以为后续课程打下良好的基础，又会对学生一生的程序设计技术、技巧、风格和习惯产生深远影响。

本书重点在于程序设计，而对 C语言本身则采取有所取、有所不取的策略。对于那些常用的语言成分，直接讲述与程序设计方法有关的语言成分，详细准确地介绍；对于那些与程序设计方法联系不太紧要，但是还常用的部分，放在最后简单介绍；而对于那些与讲述程序设计方法关系不太大，也不常用的部分则根本不涉及。

本书力图在深度、广度和知识结构上作出合理的安排。试图在既训练学生的编程能力，又培养学生的抽象思维能力上下功夫；使学生既具有较强的编程能力，又能掌握高级语言 C本身的语法和语义，同时在知识结构、知识面上尽量做到广泛、深入。

本书作者从事计算机教学已经 20余年，讲授过 10多门计算机方面的课程。曾十余次为吉林大学计算机系本科生主讲“高级语言程序设计”课。对 C语言进行了深入研究，仔细研究了国际标准 ISO/IEC 9899:1999和中华人民共和国国家标准 GB/T 15272－94。本书是作者二十余年教学实践的总结。

作为大学本科计算机专业基础课教材，本书具有如下特点：

1. 全书整体结构好，知识体系新颖完整，章节安排合理，并注意由浅入深地介绍程序设计知识。比如有关函数的知识，由浅入深地分三章介绍；有关指针的知识分散到各个章节介绍，免得集中在一章，使学生学起来枯燥乏味，接受困难。

2. 注重对学生进行严格的抽象思维训练。严格按照国际标准 ISO/IEC 9899:1999和国家标准 GB/T 15272－94介绍 C语言，并使用 BNF表示语法，使用自然语言叙述语义。对 C语言语法、语义的描述严格、细致、准确，并且形式化，为后续课程（例如编译原理）打下了良好的基础。

3. 本书最大的特点是以“程序设计”为主线，重点放在讲述程序设计方法上，摈弃了目前各种程序设计书中流行的以“解释程序设计语言”为主的做法；注重对学生进行程序设计方法及算法的训练，力图做到严格的理论与具体方法、算法有机结合。全书配备有大量例题，一方面，在讲解例题时着重于算法的构造，以便训练学生的编程能力；另一方面，概念的介绍都以例题作引导，从具体实例出发，使概念引进得自然且容易理解。本书绝大部分例题不是单纯地为了解释语言概念，而是从构造算法出发，以训练学生的实际编程能力为目标。

另外,本书还配备有大量的习题,以便学生做课后练习和进一步提高。

4. 全书自始至终贯穿结构化程序设计思想,所有例题都具有良好的结构和程序设计风格。目的是给学生一个示范,使学生从开始学习程序设计就养成一个良好的程序设计习惯和风格。

5. 本书图文并茂,引进 PAD图表示程序逻辑。 PAD图的结构比传统的流程图、NS图等都好,同时也比直接用程序表示算法更直观,易于理解。

全书共十四章,大致分为四部分。

第一部分基本知识,包括第一、二、三章。第一章介绍程序设计基本概念、BNF和 PAD图;第二章介绍 C语言基本符号、单词、数据及其类型;第三章介绍常量、变量、表达式、简单程序、赋值和输入/输出。

第二部分程序设计,包括第四、五、八、九章。第四章简单介绍模块化程序设计思想,引进子程序和函数概念;第五章讲述结构化程序设计的顺序、分支、重复三种程序逻辑,并介绍实现这三种程序逻辑的 C语言流程控制语句;第八章进一步介绍函数,讲述参数、作用域以及递归程序设计;第九章介绍程序开发和结构化程序设计,包括结构化程序设计原则、程序风格、自顶向下逐步求精的程序设计技术、程序正确性、可移植性、文档以及穷举法和试探法。

第三部分数据组织,包括第六、七、十、十一、十二章。第六章讲述数组;第七章介绍指针;第十章讲述文件及其操作;第十一章讲述对复杂数据的描述,引进结构体和共用体;第十二章讲述动态数据结构及其在程序设计中的应用。

第四部分,包括第十三、十四章。第十三章进一步介绍函数,讲述函数作参数和函数副作用等较深入的问题;第十四章介绍存储类别、位操作、break和 continue语句、编译预处理等一些 C语言独有的特色。

本书的第七、十一、十二、十四章由陈娟执笔,附录四由崔燕提供素材,其余各章节由张长海执笔。全书由张长海统稿。陈娟分别使用 Turbo C 3.0和 Microsoft Visual C++6.0调试并通过了所有例题程序。

在本书的编写过程中,作者参阅并引用了国内外诸多同行的文章、著作,在此向他们致意,并恕不一一列举、标明;在本书的成书和出版过程中得到高等教育出版社的大力支持和帮助,作者在此表示感谢。

由于作者学术水平有限,书中难免存在错误和不足,敬请各位读者批评指正,作者表示由衷的感谢。

作 者

2004年于长春

目　　录

第八章　再论函数

本章将讲述一些与函数有关的较深入的内容，包括参数、作用域、递归等。最后第十三章还将介绍有关函数的更深入的内容。

8.1 参　　数

8.1.1　C参数传递规则

第四章中已经介绍过，C语言只有值参数一种参数类别。函数调用时把实在参数的值传送到形式参数中。

从一般程序设计语言参数规则来看，值参数意味着：

⑴ 函数调用中，与值参数对应的实在参数是一个表达式（单个变量，常量是表达式的特例），并且要求实在参数表达式与形式参数变量之间赋值兼容。

⑵ 当程序运行时，调用函数，进行参数结合的动作是：

① 首先计算实在参数表达式的值；

② 把实在参数的值按赋值转换规则转换成形式参数的类型；

③ 把转换后的实在参数表达式值送入形式参数变量中。

⑶ 在整个函数活化期间，值参数表示形式参数本身，它是函数内的一个局部变量，已经与实在参数脱离关系，与实在参数无关了。在函数内对形式参数的赋值不影响实在参数。

⑷ 当函数执行结束返回后，实在参数值无任何变化，还是调用函数之前的值。

例 8.1　本例用以说明 C函数内对形式参数的操作不影响实在参数。

```
/*  ROGRAM checkout*/
  #include "stdio.h"                    /* 1*/
  int u, v;                             /* 2*/
  void p (int x, int y)  {              /* 3*/
        y=x+y;                          /* 4*/
        printf("%d%d\n", x, y);         /* 5*/
  }                                     /* 6*/
  void  ain (void)  {                   /* 7*/
      u=3;                              /* 8*/
      v=4;                              /* 9*/
```

```
    p (u, v) ;                          /* 10 */
    printf ("%d%d\n", u, v) ;           /* 11 */
    p (6, u+v) ;                        /* 12 */
    printf ("%d%d%d\n", u, v, u+v) ;    /* 13 */
}                                       /* 14 */
```

该程序有全局变量 u、v,同时函数 p还有形式参数 x、y,分析该程序的执行过程。

⑴ 开始执行第 8、9行的赋值,得:

```
u:
v:    4
```

⑵ 执行第 10行函数调用 p (u, v) 进行参数结合,计算实在参数 u、v值,分别送入形式参数 x、y,得

```
x:    3
y:    4
```

⑶ 进入函数执行第 4行赋值 y=x+y;给 y赋值与主程序中 v没有任何关系,得

```
y:    7
```

⑷ 执行第 5行输出函数 printf ("%d%d\n", x, y) ;打印

```
3    7
```

⑸ 函数执行结束,返回主程序第 10末尾。

⑹ 执行第 11行输出函数 printf ("%d%d\n", u, v) ;这时主程序中的 v值还是调用函数 p之前的值 4,虽然函数内给形式参数 y赋值 7,但是不影响主程序。本语句打印

```
3    4
```

⑺ 执行第 12行函数调用 p (6, u+v) 进行参数结合,计算实在参数 6、u+v值,分别送入值参数 x、y,得

```
x:    6
y:    7
```

⑻ 进入函数执行第 4行赋值 y=x+y;给 y赋值与主程序中 u+v没有任何关系,得

```
y:    13
```

⑼ 执行第 5行输出函数 printf ("%d%d\n", x, y) ;打印

```
6    13
```

⑽ 函数执行结束,返回主程序第 12末尾。

⑾ 执行第 13行输出函数 printf ("%d%d%d\n", u, v, u+v) ;这时主程序中 u+v的值还是用调用函数 p之前的 u、v值 3、4来计算,得 7,虽然函数内给形式参数 y赋值 13,但是不影响主程序中实在参数 u+v,本语句打印

```
3    4    7
```

8.1.2 指针作参数

从一般概念上讲,函数某个“形式参数”是指针类型。对应的函数调用,其相应“实在参数”也是一个指针类型表达式。在调用函数时,把实在参数指针值送入形式参数指针变量中,没有什么新的特殊概念。例如,函数 f有一个 int类型的指针参数,它的声明和调用形式可能是:

```
void f(int* x) {
    …
}
void main(void) {
    int* v;
    …
    … f(v) …
    …
}
```

但是真正应用指针参数,其作用是相当大的。由于在函数内部,指针参数变量可以指向它的调用处(外层程序)的其他变量,它起到了其他程序设计语言中变量参数的作用。

如下例 8.2中程序的功能是对随意输入的两个整数,按由大到小的顺序输出。函数 swap的功能是交换两个整数变量的值。

例 8.2　指针作参数的作用。

```
#include <stdio.h>
void swap(int* xx, int* yy) {                    /* 位置 2*/
    int temp;
    temp=* xx;
    * xx=* yy;
    * yy=temp;                                   /* 位置 3*/
}
void main() {
    int x, y;
    int* px, * py;
    scanf("%d%d", &x, &y);
    px=&x;
    py=&y;                                       /* 位置 1*/
    if(x<y)
        swap(px, py);
    printf("\n%d\t%d\n", x, y);
}
```

当输入

3　6

时,程序运行结果为:

6　3

下面分析该程序的运行过程。

(1) 当程序执行到位置 1时,系统在内存为主函数 main()开辟一段存储空间,并给变量 px、py赋值,使它们分别指向 x、y,如图 8.1所示。

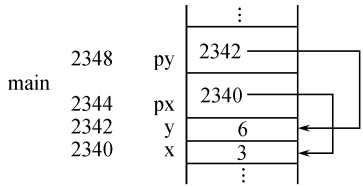

图 8.1　例 8.2中程序执行到位置 1时的内存状态

(2) 程序继续运行,调用函数 swap,系统内存又为函数 swap()分配一段空间。参数结合后,当程序执行到位置 2时,内存状态如图 8.2所示。

(3) 从函数 swap()返回前,当程序执行到位置 3时,内存状态如图 8.3所示。因为指针参数所传递的是地址值,可以通过 xx、yy访问到主程序中的变量 x、y,所以交换了变量 x、y的值。变量 px和 py所指变量依然是 x和 y,但变量 x、y的内容却发生了变化。函数返回后,实现了 x、y值的交换。

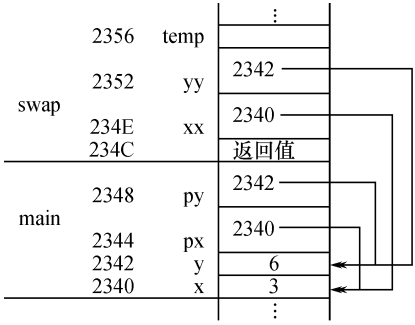

图 8.2　例 8.2中程序执行到位置 2时的内存状态

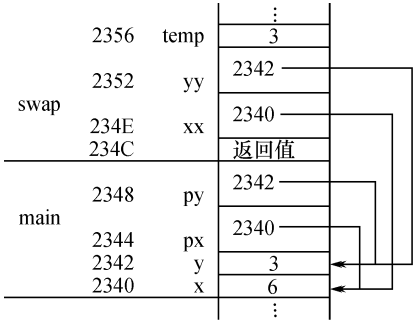

图 8.3　例 8.2中程序执行到位置 3时的内存状态

该例甚至可以不引进指针变量 px、py,直接使用变量 x、y的地址调用函数 swap,主程序如下,函数 swap与前面相同。

```
void main() {
    int x, y;
    scanf("%d%d", &x, &y); /* 位置 1 */
    if(x < y)
        swap(&x, &y);
    printf("\n%d\t%d\n", x, y);
}
```

该程序运行过程如下:

(1) 当程序执行到位置 1时,系统在内存为主函数 main()分配一段空间,如图 8.4所示。

(2) 程序继续运行,调用函数 swap。系统在内存又为函数 swap()分配一段空间,参数结合后,当程序执行到位置 2时,内存状态如图 8.5所示。注意,这时通过形式参数与实在参数的结合,把实在参数表达式“&x”和“&y”的值分别送入形式参数变量 xx、yy中。&x和 &y的值分别是变量 x、y的地址,从而使 xx、yy分别指向主程序中的变量 x、y。

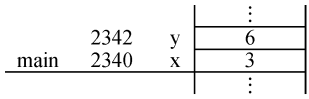

图 8.4　程序执行到位置 1时的内存状态

(3) 从函数 swap()返回前,当程序执行到位置 3时,内存如图 8.6所示。因为指针参数所传递的是地址值,可以通过形式参数 xx、yy访问到主程序中的变量 x、y,所以交换了变量 x、y的值。

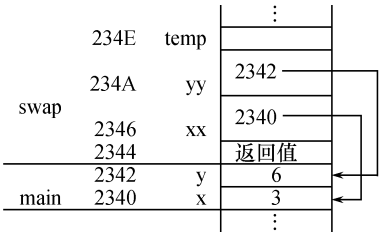

图 8.5　程序执行到位置 2时的内存状态

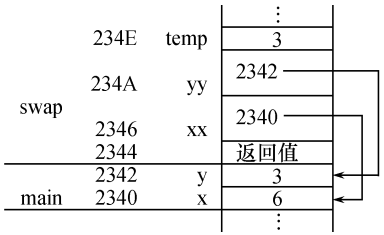

图 8.6　程序执行到位置 3时的内存状态

请读者注意交换 x、y的值是如何实现的,从而体会指针作参数产生的作用。

第四章验证 Pascal定理例题 4.4的程序,传递信息时让人感到十分别扭,函数不能把多个计算结果带回调用处。使用指针参数可以解决这个问题,例题 4.4的程序可以改写。

例 8.3　改写例 4.4,验证 Pascal定理,圆的内接六边形三双对边延线的交点在一条直线上。

```
/* PROGRAM Pascal.theorem*/
#include "math.h"
#include "stdio.h"
#define PI 3.1415927
#define eps 1e-5
float radius;                                   /* 圆的半径 */
float theta1, theta2, theta3, theta4, theta5, theta6;   /* 六个极角的度数 */
float xa, ya, xb, yb, xc, yc, xd, yd, xe, ye, xf, yf;   /* 六个顶点的直角坐标 */
float b1_x, b1_y, b2_x, b2_y, b3_x, b3_y;       /* 三个交点的直角坐标 */
/*主程序之前这段为“函数原型”以及各个函数返回结果所用变量*/
void trans_abcdef();                            /*计算六个顶点直角坐标*/
void coordinate(float, float, float*, float*);  /*计算一个顶点直角坐标*/
void three_inter();                             /*求三个交点*/
void intersection(float, float, float, float, float, float, float
                  , float*, float*);            /*已知四点,求两条直线交点*/
```

```
void equation (float, float, float, float, float, float, float
               , float*, float*, float*, float*);   /*已知四点,求两条直线方程*/
void straightline (float, float, float, float,
               float*, float*);      /*已知两点,求直线方程斜率 (a)和截距 (b)*/
void inter (float, float, float, float,
               float*, float*);      /*已知两个直线方程的斜率和截距,求它们的交点*/
int test (float, float, float, float, float, float);          /* 检验 */
/*主函数*/
void main() {
    /*读入圆形的半径*/
    printf("please input the radius of the circle:");
    scanf("%f",&radius);
    /*读入六个角*/
    printf("please input six angle:");
    scanf("%f%f%f%f%f%f",&theta1,&theta3,&theta3,&theta4,&theta5,&theta6);
    trans_abcdef();                              /*计算六个定点坐标*/
    three_inter();                               /*求三个交点*/
    if(test(b1_x,b1_y,b2_x,b2_y,b3_x,b3_y))      /*验证*/
       printf("ok");
    else {
        printf("There is an error when: radius=%f\n",radius);
        printf("theta1=%f theta2=%f\n",theta1,theta2);
        printf("theta3=%f theta4=%f\n",theta3,theta4);
        printf("theta5=%f theta6=%f\n",theta5,theta6);
    }
}
/*计算六个顶点直角坐标*/
void trans_abcdef() {
    coordinate(radius,theta1,&xa,&ya);
    coordinate(radius,theta2,&xb,&yb);
    coordinate(radius,theta3,&xc,&yc);
    coordinate(radius,theta4,&xd,&yd);
    coordinate(radius,theta5,&xe,&ye);
    coordinate(radius,theta6,&xf,&yf);
}
/*计算一个顶点直角坐标*/
void coordinate(float r, float theta, float* px, float* py) {
    *px=r*cos(PI*theta/180);      /* 先把"角度"转换成"弧度",再转换成直角坐标 */
    *py=r*sin(PI*theta/180);
}
/*求三个交点*/
void three_inter() {
    intersection(xa,ya,xb,yb,xd,yd,xe,ye,&b1_x,&b1_y);
```

```
    intersection (xb, yb, xc, yc, xe, ye, xf, yf, &b2_x, &b2_y) ;
    intersection (xc, yc, xd, yd, xf, yf, xa, ya, &b3_x, &b3_y) ;
}
/*已知四点,求两条直线的交点*/
void intersection (float rx, float ry, float sx, float sy,
                   float tx, float ty, float ux, float uy,
                   float* hx, float* hy) {
    float l1_a, l1_b, l2_a, l2_b;                    /* 两条直线的斜率和截距 */
    equation (rx, ry, sx, sy, tx, ty, ux, uy, &l1_a, &l1_b, &l2_a, &l2_b) ;
    inter (l1_a, l1_b, l2_a, l2_b, hx, hy) ;
}
/*已知四点,求两条直线方程*/
void equation ( bat rx, float ry, float sx, float sy,
                float tx, float ty, float ux, float uy,
                float*ma, float*mb, float* na, float* nb) {
    straightline (rx, ry, sx, sy,ma, mb) ;
    straightline (tx, ty, ux, uy, na, nb) ;
}
/*计算由两点确定直线方程的斜率 (a)和截距 (b)*/
void straightline (float ex, float ey, float fx, float fy, float* a, float* b) {
    *a= (fy-ey) /(fx-ex) ;                           /* 斜率 */
    *b=ey-a* ex;                                     /* 截距 */
}
/*已知两个直线方程的斜率和截距,求它们的交点*/
void inter (float ma, float mb, float na, float nb, float* wx, float* wy) {
    *wx= (nb-mb) /(ma-na) ;
    *wy=ma* wx+mb;
}
/* 检验 */
bool est(float b1_x, float b1_y,
         float b2_x, float b2_y,
         float b3_x, float b3_y)  {
    float pa, pb;                                    /* B1、B2连线方程系数 */
    straightline (b1_x, b1_y, b2_x, b2_y, &pa, &pb) ;
    if (fbs (b3_y - (pa * b3_x + pb) ) <eps)
       return true;
    else
        return false;
}
```

在该程序中,计算一个顶点在直角坐标系中坐标的函数 coordinate用指针参数 px、py代替了原来的全局量传递计算结果坐标。使用形式是:

(1) 在函数声明中,增加形式参数 px、py,并把它声明成指针类型,构成指针参数:

void coordinate（float r, float theta, float* px, float* py）

⑵ 在函数调用中，用变量的指针（地址）作实在参数，对应相应形式参数。对应点 A有：

coordinate（r, theta1, &xa, &ya）；

⑶ 在函数声明中，对形式参数 px、py以间接寻址方式赋值，把值直接送入实在参数指针表示的变量中。对应点 A是 xa、ya。

*px=r*cos(PI*theta/180)；
*py=r*sin(PI*theta/180)；

在这种参数中，注意如下事项：

① 函数调用时实在参数把变量 xa、ya的指针分别送入形式参数 px、py中，因此 px、py分别指向 xa、ya。

② 函数内部，以间接寻址方式赋值，分别通过*px、*py把值送入 px、py所指的变量中，也就是 xa、ya中。

③ 当函数返回后，xa、ya中自然是函数内部给它们赋的值。

在本程序中，求两条直线交点函数 intersection的参数 hx、hy；计算直线方程系数函数 straightline的参数 a、b；求两个直线方程交点函数 inter的参数 wx、wy等都是指针参数。它们都以上述方式传递函数的计算结果。

特别指出函数 three_inter、intersection和 inter之间的参数传递：函数 intersection有指针参数 hx、hy；函数 inter有指针参数 wx、wy；函数 intersection调用 inter时直接使用形式参数 hx、hy对应函数 inter的形式参数 wx、wy，那么 wx、wy指向什么？

```
void hree_inter(void) {
     intersection(…, &b1_x, &b1_y);
    …
}
void ntersection (…, float* hx, float* hy)
     {
      inter(…, hx, hy);
    }
float nter (…, float* wx, float* wy)
     {
    *wx= (nb-mb) /(ma-na);
    *wy=ma*wx+mb;
    }
```

该参数传递过程是：

函数 three_inter中的函数调用 intersection（…, &b1_x, &b1_y）把 b1_x、b1_y的地址送入函数 intersection的形式参数 hx、hy中，形式参数 hx、hy本身是指针类型，它们分别指向 b1_x、b1_y。

函数 intersection中的函数调用 inter(…,hx,hy)直接使用 hx、hy对应函数 inter中的形式参数 wx、wy;wx、wy是指针参数,参数结合时应该得到指针值。实在参数使用函数 intersection的形式参数 hx、hy,它们正好是指针类型,分别保存 b1_x、b1_y的指针值。调用 inter函数后,wx、wy分别指向 b1_x、b1_y,函数 inter中的两个赋值结果分别给 b1_x、b1_y赋值。

请读者认真体会并思考在如下几种情况下,实在参数表达式应该是怎样的形式。

⑴ 单独变量作实在参数,包括:

① 一般非指针类型变量作一般非指针类型形式参数的实在参数;

② 一般非指针类型变量作指针类型形式参数的实在参数;

③ 指针变量作指针类型形式参数的实在参数;

④ 指针变量作一般非指针类型形式参数的实在参数。

⑵ 单独形式参数变量作实在参数,包括:

① 一般非指针类型形式参数变量作一般非指针类型形式参数的实在参数;

② 一般非指针类型形式参数变量作指针类型形式参数的实在参数;

③ 指针类型形式参数变量作一般非指针类型形式参数的实在参数;

④ 指针类型形式参数变量作指针类型形式参数的实在参数。

8.1.3 数组作参数

数组作参数

我们已经知道,数组名实际是一个指针,所以数组名作实在参数传送给形式参数的信息实质上是一个指针值,当然相应形式参数应该是指针类型的。使用数组作函数参数的形式一般是:

⑴ 形式参数用数组声明符说明,如下例子说明形式参数 x是 10个元素的数组。

```
int f(float x[10])
```

⑵ 实在参数用数组名对应形式参数数组,例如,若有声明

```
float a[10];
```

则用如下形式调用函数

```
f(a)
```

函数调用 f(a)把数组 a的首地址送入函数 f的形式参数 x中,在函数执行期间,形式参数 x指向实在参数数组 a,用 a参与进一步运算。也就是说,形式参数和实在参数使用一个数组。

数组与指针有极其密切的关系。数组作参数,传递给形式参数的实际是实在参数数组的首地址,也就是把实在参数数组名的值送入形式参数中。在函数内,使用形式参数数组实际是使用实在参数数组名字开始的那片存储区。事实上,C语言是把数组参数当作指针来