第 6 章　指针

教学内容

- 指针变量的定义和使用
- 指针的算术运算和关系运算
- 指向一维数组元素和多维数组的指针及其应用
- 指向指针的指针及其应用
- 动态内存分配与释放
- C 语言图形处理

教学目标

通过本章的学习，初学者应该通过学习和编写适量的实例程序，掌握指针的概念和运算规则，学会用指针访问变量、一维数组和二维数组，以及用指针处理字符串，理解指针作为函数参数的原理。掌握利用指针传递数据参数和函数参数的方法；了解指向函数的指针和指向指针的指针的概念，动态空间的申请和释放的方法。

教师要求

要帮助 C 语言的初学者掌握变量的地址和指针的关系，指针的使用技巧。准确区分指针和指针指向的变量之间的关系。帮助 C 语言的初学者理解用指针访问变量、数组、函数等的方法。

学生要求

通过阅读理解本章例题和练习，掌握程序设计中指针的使用技巧，深入指针与数组的关系的理解，逐步提高自己的程序设计水平和灵活掌握 C 语言指针在程序设计中的应用。

6.1　指针的概念

6.1.1　什么是指针

我们可以把计算机的内部存储器比作教学大楼的一排教室，每一个教室都可以容纳学生，并通过地址的编号来标识。

在程序运行过程中，所有的程序和数据都是以二进制方式存放在存储器中的。通常 8 个二进制位称一个字节，我们一般把存储器中的一个字节称为一个内存单元。变量是程序中数据所占若干个存储单元的符号表示。程序在编译时，系统会根据变量的数据类型为其分配相应的若干个内存单元，用于存放变量的具体数据。就像每一个教室都可以容纳学生一样，不同类型的变量所分配的内存字节数是不一样的。如整型量占 2 个单元，字符量占 1 个单元等，在第 2 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。对于一个内存单元来说，单元的地址即为指针。通常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，也可称为某内存单元的指针。

在 C 语言中，设有字符变量 ch，其内容为“A”（ASCII 码为十进制数 65），假设字符变量 ch 在存储器中的存储单元是 10010001（地址用二进数表示）。有指针变量 pointer，内容为 10010001，就是字符变量 ch 的存储器地址， 这种情况我们称为 pointer **指向**变量 ch，或说 pointer 是**指向**变量 ch 的指针。

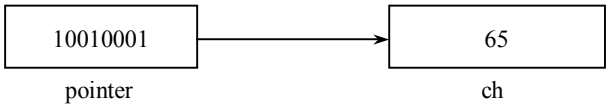

图 6.1　指针 pointer 及指向 ch

为了表示指针变量和它所指向的变量之间的关系，在程序中用“*”符号表示“指向”，*pointer 就是表示指针指向的存储单元，因此，我们可以用指针变量 pointer 和字符变量 ch 分别访问这个存储单元。

6.1.2　为什么要引入指针概念

通过变量地址可以找到该变量所占的存储单元，因此可以说变量地址“指向”变量的存储单元。直接按照变量名称访问变量的方式称为“直接访问”。把变量地址存放在一个特殊变量中，然后通过访问特殊变量的值（地址），再由此地址找到最终要访问的变量的方法，称为“间接访问”。如图 6.1 所示的那样，通过名称 ch 存取内存中的数据，是直接访问，用指针 pointer 来存取内存中的数据，是间接访问。

C 语言的特色之一，也是其精华所在就是指针。使用指针可以实现以下用其他方法不易实现的操作：如对内存中的数据进行处理，而不管这些数据的类型；在函数之间快速地传递数据；直接处理内存地址。这种方法增强了 C 语言的处理能力、提高了程序的执行效率，尤其适合系统软件的开发，也极大地增强了程序的灵活性。另外一方面，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，它是可以“指向”一个数据结构的，因而概念更为清楚，表示更为明确。但是指针不仅不易理解，而且使用不当会带来严重的错误，所以学好指针，关键是理解其概念与操作原理。

6.1.3　指针变量的定义

定义指针变量的一般形式为：

［存储类型］　类型标识符　*变量名;

其中对指针变量的类型说明包括三个内容：

（1）“*”后面的变量为一个指针变量；

（2）变量名即为定义的指针变量名；

（3）类型标识符表示该指针变量所指向的变量的数据类型。

例如：

```
int   *p1;            /*p1 是指向整型变量的指针变量*/
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*/
```

下面对指针变量作几点说明：

（1）指针变量名前的“*”表示该变量为指针变量，而指针变量名不包含该“*”。

（2）一个指针变量只能指向同一类型的变量。

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。不能时而指向一个整型变量，时而又指向一个字符变量。至于 p1 究竟指向哪一个整型变量，应由向 p1 赋值的地址来决定。

（3）指针变量中只能存放地址，而不能将数值型数据赋给指针变量。

（4）只有当指针变量中具有确定地址后才能被引用。

（5）与一般的变量一样，也可以对指针变量进行初始化。后面进一步说明。

指针与指针变量关系：

指针：一个变量的地址，是常量。

指针变量：专门存放变量地址的变量。

存储器的地址是常量，“指针变量”是指取值为地址的变量。定义指针变量的目的是为了通过指针（地址）去访问内存单元。

任何一个指针变量都是用于存放它所指向变量的地址，只要能存放地址就可以了，为何还要区别不同的基类型呢？

其原理是：不同的数据类型变量，C 语言系统为它们开辟的存储空间的字节数是不同的，int 类型的数据存储空间是 2 个字节，float 类型的数据存储空间是 4 个字节等。系统表示每一个存储空间的地址时，是取该存储空间的第 1 个字节的地址作为该变量存储空间的地址。那么当一个基类型为 int 类型的指针变量 p 指向了一个 int 类型的变量 a 时，是将该变量 a 所占的 2 个字节的存储空间中的第 1 个字节存储空间的“地址”存入指针变量 p 中，如图 6.2 所示。

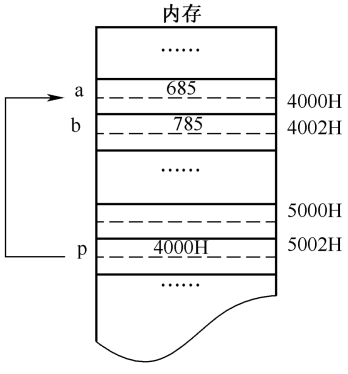

图 6.2　变量和指针内存结构

所以根据指针变量 p 中存放的“地址”，只能寻找到变量 a 第 1 个字节的存储空间，如果只提取变量 a 所占存储空间第 1 个字节的数据，显而易见不是 int 类型变量 a 的原值，因为变量 a 的原值是通过 2 个字节来存储的数据。此时我们可以通过指针变量 p 的类型解决问题，知道了变量 a 的第 1 个字节的地址，再根据指针变量 p 的类型为 int 类型，系统就将从变量 a 的第 1 个字节所在的地址开始，连续提取 2 个字节中的数据，此时的数据就是 int 类型变量 a 的原值。

同理，基类型为 float 类型的指针变量，根据指针变量中存放 float 类型变量的地址值，可以寻找到所需存储空间中的第 1 个字节所在位置，然后再根据类型为 float 类型，连续地提取 4 个字节中的数据，作为被访问的数据，这才是 float 类型变量中存放的真实数据。因此，定义什么样类型指针变量，该指针变量只能存放什么样类型变量的地址，两者必须一致，否则就可能出现错误。

6.1.4 指针变量的使用

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， 决不能赋予任何其他数据，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，用户可以不知道变量的具体地址。也不需要考虑变量的具体地址值是什么。

两个与指针有关的运算符：

（1）&：取地址运算符。

（2）*：指针运算符（或称“间接访问”运算符）。

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 表示变量 a 的地址，&b 表示变量 b 的地址。变量本身必须预先定义。

【例 6-1】 输出变量的值和它的地址值。

```
#include "stdio.h"
main()
{
  int i=125;
  printf("%d\n", i);
  printf("%ld\n", &i);
}
```

程序的输出结果如图 6.3 所示。

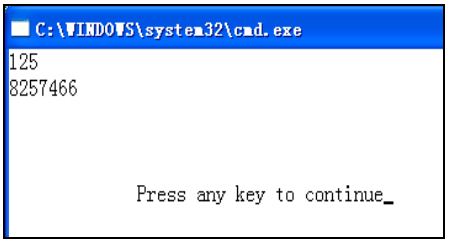

图 6.3 例 6-1 程序运行结果

我们看到变量 i 的地址值是一个长整数。

【例 6-2】 分别用整型变量和指针变量输出变量的值。

```
#include "stdio.h"
main()
{
    int a=10,b=20,*pa,*pb;
    pa=&a;
    pb=&b;
    printf("%d,%d\n",a,b);
    printf("%d,%d",*pa,*pb);
}
```

程序的输出结果如下：

```
10，20
10，20
```

说明：pa，pb 为整型指针变量，在定义时并未指向任何整型变量，然后给指针变量 pa 赋以变量 a 的地址值，pa 指向变量 a，给指针变量 pb 赋以变量 b 的地址值，pb 指向变量 b。分别通过变量和指向变量的指针输出变量的值。

在 printf("%d,%d",*pa,*pb)；中我们使用了*pa，*pb，那么*pa 和*pb 是什么意思呢？*pa 表示指针变量 pa 所指的地址的内容，即变量 a 的内容。*pb 表示指针变量 pb 所指的地址的内容，即变量 b 的内容。

指针变量的初始化方法：

（1）在定义的同时进行初始化。例如：

```
int a=10;
int *p=&a;
```

（2）赋值语句进行初始化。例如：

```
int a=10;
int *p;
p=&a;
```

（3）可以用初始化了的指针变量给另一个指针变量做初始化值。例如：

```
int x=5;
int *p=&x;      /*将变量 x 的地址赋给指针变量 p*/
int *q=p;       /*将 p 的值赋给 q，使 p、q 指向同一变量 x*/
```

（4）以将一个指针变量初始化为一个空指针。例如：

```
int *p=NULL;
```

空指针：

空指针是一个特殊的指针，它的值是 0，C 语言中用符号常量 NULL（在 stdio.h 中定义）表示这个空值，并保证这个值不会是任何变量的地址。空指针对任何指针类型赋值都是合法的。一个指针变量具有空指针值表示当前它没有指向任何有意义的东西。

注意：

（1）不允许把一个数赋予指针变量，故下面的赋值是错误的：

```
int *pa;
pa=5000;
```

（2）被赋值的指针变量前不能再加“*”说明符，如写为下面也是错误的

```
int a=10;
int *pa;
*pa=&a;   （错误）
```

pa 是指针变量，可以存储地址值。而*pa 表示指针变量所指向的存储单元。

【例 6-3】 通过指针变量进行变量赋值。

```
#include "stdio.h"
main()
{
    int i=10,x;
    int *p;
    p=&i;
    x=*p;
    printf("i=%d,x=%d\n",i,x);
}
```

程序的输出结果如下：

```
i=10,x=10
```

说明：我们定义了两个整型变量 i，x，还定义了一个指向整型变量的指针变量 p。i，x 中可存放整数，而 p 中只能存放整型变量的地址。我们可以把 i 的地址赋给 p：

```
p=&I;
```

此时指针变量 p 指向整型变量 i，以后我们便可以通过指针变量 p 间接访问变量 i，例如：

```
x=*p;
```

运算符*访问以 p 为地址的存储区域，而 p 中存放的是变量 i 的地址，因此，*p 访问的是地址为变量 i 的存储区域，所以上面的赋值表达式等价于

```
x=i;
```

【例 6-4】 指针变量之间互相赋值

```
#include "stdio.h"
main()
{
    int a=10,*p1,*p2;
    p1=&a;
    p2=p1;
    a++;
    printf("%d, %d, %d\n",a,*p1,*p2);
}
```

程序的输出结果如下：

```
11,11,11
```

说明：指针变量和一般变量一样，存放在它们之中的值也是可以改变的，也就是说可以改变它们的指向，指针变量 p2 开始没有指向任何变量，这时赋值表达式：

```
p2=p1
```

就使 p2 与 p1 指向同一对象 a，此时*p2 就等价于 a。

如果以上表达式改成：

```
*p2=*p1;
```

则表示把 p1 指向的存储空间内容赋给 p2 所指的存储空间。

【例 6-5】 通过指针变量改变整型变量的值。

```
#include "stdio.h"
main()
{
    int a, b, c, *p1, *p2;
    a=2; b=4; c=6;
    p1=&a; p2=&b;
    printf("a=%d,b=%d,c=%d,*p1=%d,*p2=%d \n",a,b,c,*p1,*p2);
    (*p1)++; (*p2)++;
    printf("a=%d,b=%d,c=%d,*p1=%d,*p2=%d \n",a,b,c,*p1,*p2);
    p2=p1; p1=&c;
    printf("a=%d,b=%d,c=%d,*p1=%d,*p2=%d \n",a,b,c,*p1,*p2);
    (*p1)++; (*p2)++;
    printf("a=%d,b=%d,c=%d,*p1=%d,*p2=%d \n",a,b,c,*p1,*p2);
}
```

程序的输出结果如下：

```
a=2, b=4, c=6, *p1=2, *p2=4
a=3, b=5, c=6, *p1=3, *p2=5
a=3, b=5, c=6, *p1=6, *p2=3
a=4, b=5, c=7, *p1=7, *p2=4
```

说明：程序执行 p1=&a;p2=&b;语句后，指针变量 p1，p2 分别指向变量 a，b。执行语句：

```
(*p1)++; (*p2)++;
```

相当于 a++，b++。然后又执行：

```
p2=p1; p1=&c
```

使 p2 指向变量 a，p1 指向变量 c。

执行语句：

```
(*p1)++; (*p2)++;
```

实际上是 c++，a++。

6.2 指针的运算

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。

我们把指针变量赋值运算作一下总结，一些相关内容在后面详细论述。

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相同数据类型的指针变量。例如：

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

数组名表示数组的首地址，故可赋予相同数据类型的指针变量 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];
int *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";
```

这里应说明的是并非把整个字符串装入指针变量，而是把内存中存放该字符串的首地址赋给指针变量。

（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。

6.2.1 指针的算术运算

对于指向数组元素的指针变量，可以加上或减去一个整数 n。设 pa 是指向数组 a 的首元素的指针变量，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，指向数组元素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数是不同的。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&a[2]
```

如果 p 指向数组中的某个元素，则

p++：指针变量 p 增 1，使 p 指向下一个元素。

p+k：该地址是指向 p 后面的第 k 个元素的指针。

p-k：该地址是指向 p 前面的第 k 个元素的指针。

指针变量的加减运算只能对数组元素指针变量进行，对指向其他类型变量的指针变量作加减运算是毫无意义的。

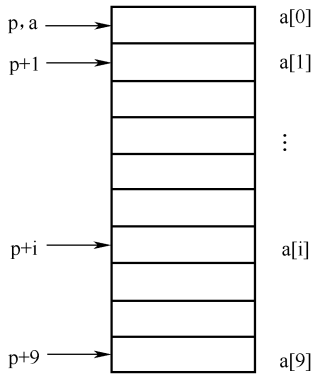

图 6.4　指针和数组关系

6.2.2　指针的关系运算

指向同一数组元素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：

p1==p2：表示 p1 和 p2 指向同一数组元素。

p1>p2：表示 p1 指针所指向的元素在 p2 指针所指向的元素之后。

p1<p2：表示 p1 指针所指向的元素在 p2 指针所指向的元素之前。

6.3　指针与数组之间的关系

6.3.1　指向一维数组元素的指针及其应用程序范例

数组是一组相同类型数据的集合，数组中各个元素在内存占据连续的存储单元，每个内存单元都有相应的地址。数组所占内存单元的首地址称为数组的指针，数组元素所占内存单元的地址称为数组元素的指针。因此，可以用指针变量来指向数组或数组元素。用于存放数组的指针或某一数组元素指针的指针变量称为指向数组的指针变量。C 语言规定，数组名代表数组的首地址，因此，数组名实际上也是指针，但它的值是一个固定不变的指针常量。

指向数组的指针变量称为数组指针变量。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 号元素可以把第 i 元素的首地址赋予它或把数组名加 i 赋给它。

数组元素指针变量定义的一般形式为：

类型标识符　*指针变量名

其中类型标识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定义是相同的。例如：

```
int  a[10],*p;
p=a;
```

定义了整型数组 a，又定义了指针变量 p，通过赋值语句 p=a 将数组 a 的首地址赋给指针变量 p，p 指向了 a 数组。指针变量 p 就称为指向数组 a 的指针变量。也可以将数组中下标为 0 的数组元素的地址赋给指针变量。

```
p=&a[0];
```

引入指针变量后，可以用以下两种方法来访问数组元素：其一为下标法，即用 a[i]形式访问数组元素，这是较为普通的方法；其二为指针法，即采用*(p+i)形式，用间接访问的方法来访问数组元素。

【例 6-6】 用指针访问数组元素。

```
#include "stdio.h"
main(){
    int i,a[10],*p;
    p=a;
    for(i=0;i<10;i++){
       *p=i;     /*利用循环将变量 i 的值赋给由指针 p 指向的 a[]的数组单元*/
       p++;      /*同时指针 p 指向 a[]的下一个单元*/
    }
    p=a;
    for(i=0;i<10;i++){
       printf("a[%d]=%d\n",i,*p);        /*用指针输出数组 a 中的所有元素*/
       p++;                              /*同时指针 p 指向 a[]的下一个单元*/
    }
}
```

程序的输出结果如下：

```
a[0]=0
a[1]=1
a[2]=2
a[3]=3
a[4]=4
a[5]=5
a[6]=6
a[7]=7
a[8]=8
a[9]=9
```

【例 6-7】 用不同的方法输出数组中的元素。

```
#include "stdio.h"
main()
{
    int a[5],i,*p=a;         /*定义数组 a，循环变量 i，指向数组的指针变量 p*/
    for(i=0;i<5;i++)
        a[i]=i+1;
    for(i=0;i<5;i++)
        printf("a[%d]=%d  ",i,a[i]);           /*通过下标方式输出数组元素*/
    printf("\n");
    for(i=0;i<5;i++)
        printf("*(a +%d)=%d  ",i,*(a+i));      /*通过数组名 a 输出数组元素*/
    printf("\n");
    for(i=0;i<5;i++)
        printf("*(p+%d)=%d  ",i,*(p+i));       /*通过指针变量 p 输出数组元素*/
```

```
    printf("\n");
    for(i=0;i<5;i++)
          printf("p[%d]=%d  ",i,p[i]);        /*利用下标法输出数组元素*/
    printf("\n");
}
```

程序的输出结果如下：

```
a[0]=1  a[1]=2  a[2]=3  a[3]=4  a[4]=5
*(a +0)=1  *(a +1)=2  *(a +2)=3  *(a +3)=4  *(a +4)=5
*(p+0)=1  *(p+1)=2  *(p+2)=3  *(p+3)=4  *(p+4)=5
p[0]=1  p[1]=2  p[2]=3  p[3]=4  p[4]=5
```

说明：引入指针变量后，就可以用两种方法来访问数组元素了。

如果 p 的初值为&a[0]，则：

（1）p+i 和 a+i 就是 a[i]的地址，或者说它们指向 a 数组的第 i 个元素。

（2）*(p+i)或*(a+i)就是 p+i 或 a+i 所指向的数组元素，即 a[i]。

例如，*(p+5)或*(a+5)就是 a[5]。

（3）指向数组首地址的指针变量也可以带下标，如 p[i]与*(p+i)等价。

（4）指针变量可以实现本身的值的改变。如 p++是合法的；而 a++是错误的。因为 a 是数组名，它是数组的首地址，是常量。

（5）虽然定义数组时指定它包含 5 个元素，但指针变量可以指到数组以后的内存单元，系统并不认为非法。所以，读者在编写程序时，千万当心下标越界。

注意：

（1）*p++，由于++和*同优先级，结合方向自右而左，等价于*(p++)。

（2）*(p++)与*(++p)作用不同。若 p 的初值为 a，则*(p++)等价 a[0]，p 指向 a[1]，而*(++p)等价 a[1]。

（3）(*p)++表示 p 所指向的元素值加 1。

（4）如果 p 当前指向 a 数组中的第 i 个元素，则：

*(p--)相当于 a[i--]，即：引用 a[i]后，p 指向 a[i-1]。

*(++p)相当于 a[++i]，即：引用 a[i+1]后，p 指向 a[i+1]。

*(--p)相当于 a[--i]，即：引用 a[i-1]后，p 指向 a[i-1]。

【例 6-8】 将数组中的 10 个数逆序存放。

```
#include "stdio.h"
main()
{
   int a[10],*p,*head,*end,temp;
   printf("please input 10 numbers:\n");
   for(p=a;p<=a+9;p++)
      scanf("%d",p);
   printf("The original array:\n");
   for(p=a;p<=a+9;p++)
      printf("%d  ",*p);
   printf("\n");
   head=a; end=a+9;          /*头指针指向 a，尾指针指向 a+9 位置*/
   while(head<end)
    {
      temp=*head;            /*头尾指针所指向的元素的值互换*/
      *head=*end;
      *end=temp;
      head++;                /*头指针前进*/
```

```
        end--;                  /*尾指针后退*/
      }
      printf("The array has been inverted:\n");
      for(p=a;p<=a+9;p++)
           printf("%d  ",*p);
}
```

程序的输出结果如下：

```
The original array:
11  22  33  44  55  66  77  88  99  100
The array has been inverted:
100  99  88  77  66  55  44  33  22  11
```

6.3.2 指向二维数组的指针及其应用程序范例

1. 二维数组的存储结构与地址

设有一个 3 行 4 列的二维数组 a 定义如下：

```
int a[3][4]={ { 1, 3, 5, 7},
              {2, 4, 6, 8},
              {10,11,12,13}};
```

二维数组在内存中是按行顺序存储的，即第一维的下标后变化，第二维的下标先变化。本例中 a 数组在计算机中的存储顺序如下：

```
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]
```

由第 5 章的学习我们知道，C 语言允许把一个二维数组分解为多个一维数组来处理，因此二维数组 a 可分解为 3 个一维数组，即 a[0]、a[1]、a[2]，每一个一维数组又含有 4 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，如图 6.5 所示。

在二维数组中，数组名 a 是指向元素 a[0]的首地址。a+1 是指向 a[1]的地址，即第一行的地址。同理 a+2 是指向 a[2]的地址，即第二行的地址。一般说来，a+i 就是指向第 i 行的地址，即 a+i 值等于&a[i]。因此 a+i，a[i]，*(a+i)，&a[i][0]都是等同的。此外，&a[i]和 a[i] 也是等同的。在二维数组中不能把&a[i]理解为元素 a[i]的地址，因为二维数组中不存在元素 a[i]。

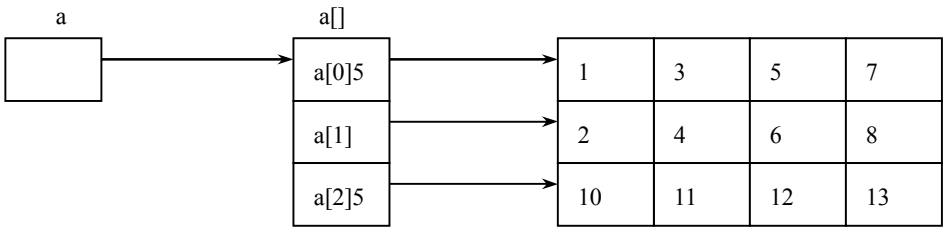

图 6.5　二维数组指针结构图

2. 二维数组指针变量的定义

二维数组指针变量定义的一般形式如下：

类型标识符　(*指针变量名)[长度]

其中“类型标识符”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。例如：

```
int a[3][4];
```

```
int  (*p)[4];
p=a;
```

int (*p)[4]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。

【例 6-9】 用指向二维数组的指针变量访问数组。

```
#include "stdio.h"
main()
{
   int a[3][4]={1,3,5,7,9,11,2,4,6,8,10,12};
   int (*p)[4];                          /* 定义一个指向二维数组的指针变量 */
   int i,j;
   p=a;
   for(i=0;i<3;i++)
   {
     for(j=0;j<4;j++)
        printf("%2d ",*(*(p+i)+j));     /* 用指针访问二维数组的数据元素*/
     printf("\n");
   }
}
```

程序的输出结果如下：

```
1  3   5   7
9  11  2   4
6  8  10  12
```

6.3.3 字符指针及其应用程序范例

在 C 语言中，表示一个字符串有以下两种形式：

（1）用字符数组存放一个字符串。

【例 6-10】 定义一个字符数组，对其进行初始化并输出该字符串。

```
#include "stdio.h"
main()
{
   char  str[]="hello world ";
   printf("%s\n",str);
}
```

（2）用字符指针指向一个字符串。

不论字符串常量或是字符串变量的值，都是一组字符加上串终止符'\0'，在内存中连续存储的，因此可以使用指向字符的指针变量操作字符串。

字符指针的定义形式如下：

char　*指针变量名;

例如：

```
char *str;
```

定义一个字符指针 str。

用一个字符指针变量实现例 6-10。

```
#include<stdio.h>
main()
{
char *str="This is a C program ";
   printf("%s\n",str);
}
```

这个例子程序非常简单，首先定义 str 是一个字符指针变量，然后把字符串的首地址赋值给

它，并输出该字符串的内容。

字符指针变量在使用之前和其他变量一样也需要初始化，字符指针的初始化方法如下：

（1）定义时初始化方式使字符指针指向字符串。例如：

```
char *str="Programming";
```

（2）用赋值运算使指针指向一个字符串。例如：

```
char *str;
str="Programming";  （与第一种初始化方法完全等价）
```

【例 6-11】 编写程序，统计一个字符串中某子串出现的次数。

```
#include "stdio.h"
#include "string.h"
main()
{
   char str1[50],str2[50],*p1,*p2;
   int num=0;
   printf("please input two strings\n");
   gets(str1);
   gets(str2);
   p1=str1;p2=str2;                    /*p1,p2 分别指向二字符串的首地址*/
   while(*p1!='\0')                    /*p1 字符串是否结束*/
   {
     if(*p1= =*p2)                     /*p1,p2 分别指向的位置字符是否相同*/
     {  while(*p1= =*p2&&*p2!='\0')    /*如果相同继续比较*/
        { p1++;
          p2++;}
     }
     else
       p1++;
     if(*p2= ='\0')                    /*p2 处于结束位置*/
       num++;
     p2=str2;
   }
   printf("%d",num);
}
```

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开始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（串）指针方式。

```
char *ps="C Language";
```

可以写为：

```
char *ps;
ps="C Language";
```

而对数组方式：

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

不能写为：

```
char st[20];
st={"C Language"}; /*错！因为 st 是一个地址常量！*/
```

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，以下

语句：

```
char *ps="C Langage";
```

或

```
char *ps;
ps="C Language";
```

都是合法的。

【例 6-12】 已知字符串 str，从中截取一子串。要求该子串是从 str 的第 m 个字符开始，由 n 个字符组成。

```
#include "stdio.h"
#include "string.h"
main( )
{
   char c[80], *p, *str="This is a string.";
   int  i, m, n;
   printf("m,n=");
   scanf("%d,%d",&m,&n);
   if (m>strlen(str) || n<=0 || m<=0)
     printf("Err\n");
   else
    {
     for (p=str+m-1,i=0; i<n; i++)      /*从首地址+m-1 处开始*/
         if(*p)
            c[i]=*p++;                  /*赋值到字符数组 c 中*/
         else
            break;                      /*如读取到 '\0' 则停止循环 */
     c[i]='\0';                         /*在 c 数组中加上子串结束标志 */
     printf("%s\n",c);
    }
}
```

程序设计的思路是：定义字符数组 c 存放子串，字符指针变量 p 用于复制子串，利用循环语句从字符串 str 截取 n 个字符。考虑到以下几种特殊情况：

（1）m 位置后的字符数有可能不足 n 个，所以在循环读取字符时，若读到'\0'停止截取，利用 break 语句跳出循环。

（2）输入的截取位置 m 大于字符串的长度，则子串为空。

（3）要求输入的截取位置和字符个数均大于 0，否则子串为空。

6.3.4 指针数组及其应用程序范例

我们定义一个数组，其数据元素均为指针类型数据，则称为指针数组。指针数组中的每一个元素都相当于一个指针变量。

一维指针数组定义的一般形式如下：

类型标识符 *数组名[数组长度];

其中类型标识符为指针值所指向的变量的类型。

例如：

```
int  *p[4];
```

表示 p 是一个指针数组，它有 4 个数组元素，每个元素值都是一个指针，指向整型变量。

【例 6-13】 指针数组的使用。

```
#include "stdio.h"
main( )
{
    int i=1, j=2, k=3, m=4, n, t;
```

```
    int *pa[4], *pt;                          /*定义一个指针数组 pa 和指针变量 pt*/
    pa[0]=&i; pa[1]=&j;                       /*每一个数据元素指向不同的变量*/
    pa[2]=&k; pa[3]=&m;
    pt=pa[0]; pa[0]=pa[3]; pa[3]=pt;          /*pa[0]和 pa[3]指针值交换*/
    for(n=0; n<4; n++)
       printf("%d ", *pa[n]);
    printf("\n");
    pa[0]=&i; pa[1]=&j;
    pa[2]=&k; pa[3]=&m;
    t=*pa[0]; *pa[0]=*pa[3]; *pa[3]=t; /* pa[0]和 pa[3]指针指向的变量值交换*/
    for(n=0; n<4; n++)
       printf("%d ", *pa[n]);
    printf("\n");
}
```

程序输出结果如下：

```
4 2 3 1
4 2 3 1
```

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。

【例 6-14】 指针数组的使用。

```
#include "stdio.h"
main()
{
    int a[3][3]={1,2,3,4,5,6,7,8,9};
    int *pa[3],*p=a[0],i  ;
    pa[0]=a[0] ;
    pa[1]=a[1];
    pa[2]=a[2];
    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));
}
```

程序输出结果如下：

```
3,1,1
5,4,5
7,7,9
1,1,1
4,2,2
7,3,3
```

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。长度 3 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。而

```
int  *p[3];
```

表示一个指针数组，共有 3 个指针变量。

指针数组也常用来表示一组字符串，一个字符串本身就是一个一维字符数组，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 6-15】 给出 5 个字符串，并按字母顺序排列后输出。

```
#include "stdio.h"
#include "string.h"
main()
{
    char *name[]={"BASIC","DBASE","CLanguage","FORTRAN","PASCAL"};
    int n=5;
    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;
        }
    }
    for(i=0; i<5; i++)
        printf("%s\n", name[i]);
}
```

在程序中定义字符数组并初始化：

```
char *name[]={"BASIC","DBASE","CLanguage","FORTRAN","PASCAL"};
```

完成这个初始化赋值之后，name[0]即指向字符串"BASIC"，name[1] 指向字符串"DBASE"，依此类推。

在一般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。

6.4 指针作为函数的参数及其应用程序范例

函数的参数不仅可以是整型、实型、字符型等数据，还可以是指针类型。用指针作为函数参数，实参与形参之间传递的是地址值，可以实现“双向”传递。

当指针变量作为函数参数时，要求实参与形参是相同类型的指针。

【例 6-16】 使用指针作函数的参数，把结果传递回来。

```
#include "stdio.h"
void add(int *p,int n);
main()
{
    int a=45;
    add(&a,10);
    printf("value is %d\n",a);
```

```
}
void add(int *p,int n)
{
    *p+=n;
}
```

说明：本例中主函数中定义变量 a，调用函数 add 时把变量 a 的地址传递给形参指针变量 p，在函数内部，指针 p 指向的单元的值发生了变化，当程序从函数返回时，变量 a 的值也发生变化。也就是说，把结果传递回来。

【例 6-17】 使用函数求出数组中的最大值，指针作函数的参数。

```
#include "stdio.h"
int max(int *p,int n);
main()
{
    int a[8]={23,4,6,12,33,55,2,45};
    printf("max is %d\n",max(a,8));
}
int max(int *p,int n)
{
   int i,j,mx;
   mx=*p;
   for (i=1;i<n;i++)
      if (*(p+i)>mx) mx=*(p+i);
   return mx;
}
```

在本例中，主函数定义整型数组 a，max 函数中的形参之一 p 是整型指针，当调用 max 函数时，数组 a 的首地址传递给指针变量 p，在函数 max 内部，使用指针 p 访问数组元素。

函数也可这样定义：

```
int max(int  a[],int n)
```

数组名作函数的参数，实现的结果完全一样。

【例 6-18】 删除输入字符串中的字符 k，要求用函数实现，字符串指针作函数的参数。

```
#include "stdio.h"
#include "string.h"
void delchar(char *str,char c);
main()
{
  char str[80];
  gets(str);
  delchar(str,'k');
  puts(str);
}
void delchar(char *str,char c)
{
  char *p;
  for (p=str;*p!='\0';p++)
     if (*p!=c)
        *str++=*p;
  *str='\0';
}
```

说明：指向字符串的字符指针或字符数组名作函数参数，将字符串的首地址从一个函数传递到另外一个函数，在被调函数中改变字符串的内容，在主调函数中可以得到改变后的字符串。

在这个例子中，主函数中定义字符数组 str，输入的字符串存入到字符数组 str 中，调用函数 delchar 时传递字符数组首地址给函数的形参：字符指针，在函数体内部通过指针访问字符数组。当从函数返回时，函数 delchar 中对字符数组的操作结果完全保留下来，也就是说，字符 k 被删

除了。

【例 6-19】 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。

```
#include "stdio.h"
#include "string.h"
void cpystr(char *pss,char *pds);
main()
{
   char *pa="C Language",b[20],*pb;
   pb=b;
   cpystr(pa,pb);       /*把字符串指针作为函数参数的使用*/
   printf("string a=%s\nstring b=%s\n",pa,pb);
}
void cpystr(char *pss,char *pds)
{
   while((*pds=*pss)!='\0')
   {
      pds++;
      pss++;
   }
}
```

说明：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。

程序完成了两项工作：一是把 pss 指向的源字符串复制到 pds 所指向的目标字符串中，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 cpystr 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 cpystr 函数中均可使用这些字符串。也可以把 cpystr 函数简化为以下形式：

```
cpystr(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。由于'\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。

6.5 指针作为函数的返回值及其应用程序范例

函数返回值的类型决定了函数的类型。在前面的学习中，给出的带返回值的函数都是返回一定数值类型的数据，如整型值、字符型值、实型值等，这类的函数称为数值型函数。除此之外，函数返回值还可以是存储某种类型数据的内存地址，即指针。如果一个函数的返回值是指针，则称此函数为指针型函数。

指针型函数定义的一般形式如下：

```
数据类型标识符  *函数名(形式参数表)
{
    函数体
}
```

其中函数名前的“*”表示函数的返回值是一个指针类型，“数据类型标识符”是指针所指向的目标变量的数据类型。例如：

```
int *fun( int x, int y)
{
    int *p;
     ⋮
    return(p);
 }
```

【例 6-20】 编写合并两个字符串的函数，返回合并后的字符串的首地址。

```
#include "stdio.h"
char *strcat2(char *str1, char *str2);
main()
{   char s1[80]="we are ",s2[]="good friends.",*p;
    puts(s1);
    puts(s2);
    p=strcat2(s1,s2);
    puts(p);
}
char *strcat2(char *str1, char *str2)
{
    char *p=str1;
    while(*p!='\0')
        p++;                /*指针 p 指向 str1 的尾部*/
    while(*str2!='\0')
        *p++=*str2++;       /*str2 接到 str1 的尾部*/
     *p='\0';
     return(str1);
}
```

程序运行的结果如下：

```
we are
good friends.
we are good friends.
```

strcat2 是个字符指针的函数，实现 str2 指向的字符串合并到 str1 指向的字符串的后面，其中语句：

```
p++=*str2++;
```

功能：将 str2 指向的字符赋给 p 指向的单元（字符串 str1 中的指定位置），然后 p 和 str 均指向下一个字符。该语句等价于：

```
*p=*str2;  p++;  str2++;
```

【例 6-21】 输入一个整数，输出与其对应的英文星期名称。

```
#include "stdio.h"
char *weekName(int);
main()
{
   int n;
   printf("Enter 1 integer:");
   scanf("%d",&n);
   printf("\n%d week: %s\n",n,weekName(n));
}
char *weekName(int n)
{
   char *name[]={"Error","Monday","Tuesday","Wednesday",
                        "Thursday","Friday","Saturday","Sunday"};
   return((n<1||n>7)?name[0]:name[n]);
}
```

说明：本例中定义了一个指针型函数 weekName，它的返回值是指向一个字符串的指针。该

函数中定义了一个指针数组 name。name 数组初始化赋值为 8 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 n 作为实参，在 printf 语句中调用 weekName 函数并把实参 n 值传送给形参 n。weekName 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“Error”。否则返回主函数输出对应的星期名。

【例 6-22】 有 4 名学生，每个学生考四门课程，要求输入学生序号后能输出该学生的全部成绩，要求用指针型函数实现。

```
#include "stdio.h"
float *search(int (*p)[4],int);
main()
{
    float  score[][4]={{60,70,80,90},{50,78,67,93},{86,88,47,69},{80,90,100,76}};
    int i,n;
    float *p;
    printf("Enter the number of student:");
    scanf("%d",&n);
    printf("The score of No. %d are:\n",n);
    p=search(score,n);
    if (p!=NULL)
       for(i=0;i<4;i++)
         printf("%5.2f ",*(p+i));
}
float *search(int (*p)[4],int n)
{
   if (n<0||n>=4)
     return NULL;
   else
    return *(p+n);   /*返回二维数组其中一行的首列地址*/
}
```

说明：

（1）指针型函数中 return 语句的返回值必须与函数的数据类型相一致的指针。

（2）不能把指针型函数内部定义的局部变量的地址作为返回值。这是因为局部变量在函数运行结束后被释放，相应的地址也要让出来，可能存放其他数据。

6.6 指向函数的指针及其应用程序范例

在 C 语言中规定，一个函数总是占用一段连续的存储区，而函数名就是该函数所占存储区的首地址，所以函数名就是函数的指针。当调用该函数的时候，系统会从这个首地址开始执行该函数。

存放函数首地址（函数指针）的变量，称为指向函数的指针变量，简称函数的指针变量。这样，函数可以通过函数名调用，也可以通过函数指针调用。

指向函数的指针变量的一般定义形式为：

类型标识符 (*指针变量名)([参数类型]);

其中“类型标识符”表示函数的返回值的类型。“(*指针变量名)”表示“*”后面的变量是指针变量。最后的括号表示指针变量所指的是一个函数。例如：

```
int  (*pf)();
```

定义 pf 是一个指针变量，它指向一个返回 int 型值的函数。

定义时，它不固定指向哪一个函数，只是定义了这样一个类型的变量，专门用来存放函数的

首地址，程序中可以先后指向不同的函数。

和变量的指针一样，函数的指针也必须赋值后才能指向具体的函数。由于函数名代表了该函数的首地址，因此，通常直接用函数名为函数指针赋值，即：

函数指针名 = 函数名;

通过函数指针实现函数调用的步骤如下：

（1）指向函数的指针变量的定义：

类型(* 函数指针变量名)();

（2）指向函数的指针变量的赋值，指向某个函数：

函数指针变量名=函数名;

（3）利用指向函数的指针变量调用函数：

(* 函数指针变量名)(实参表)

例如：

函数定义：

```
double  fun();
```

指向函数指针定义：

```
double  (*f)();
```

f 指向 fun 函数

```
f = fun;      只需给出函数名，不必给出参数
```

用指针变量调用函数：

```
(*f)();
```

说明：

（1）定义函数指针变量时，两组括号都不能少。如果少了前面的一组括号，则

返回值类型 * 函数名();

就变成返回值为地址值（指针）的函数。

（2）函数指针变量的类型是被指向的函数的类型，即函数返回值类型。

（3）函数指针的赋值，只要给出函数名，不必给出参数（不要给出实参或形参）。

（4）用指针变量调用函数时，“(*函数指针)”代替函数名。参数表与使用函数名调用函数一样。

（5）可以看出，定义的函数指针变量可以用于指向返回值类型相同的同类函数。

函数的指针也可以作为函数参数，在函数调用时可以将某个函数的首地址传递给被调用的函数，使这个被传递的函数在被调用的函数中调用（看上去好像是将函数传递给另一个函数）。函数指针提供了用指针调用函数的机制（间接调用）。函数指针的使用可以增加函数的通用性。

下面通过例子来说明用指针形式实现对函数调用的方法。

【例 6-23】 指向函数的指针使用示例。采用梯形法求解 $\int_1^5 x\sin(x)$，$\int_0^2 \frac{2x}{3e^x}$。

在第 4 章的例 4-27 我们曾经利用梯形法求函数 f(x)在[a,b]的积分，当时，我们编写的求积分函数 Integrate(double A,double B,int N)和 f(double x)是针对具体的被积函数 x^3+2x+1 的。所以，现在要求其他被积函数的定积分，比如 xsin(x)的积分，必须再编写新的积分函数或对 Integrate(double A,double B,int N)和 f(double x)进行改写。有没有办法编写一个采用梯形法求解定积分的通用函数呢？答案是肯定的，即可以编写出求解任意函数的积分程序。方法就是利用函数的指针作为通用求定积分函数的形参。

我们可以把例 4-27 的积分函数 Integrate(double A,double B,int N)中增加一个形参 double (*f)()，函数 Integrate 改写成：double Integrate(double (*f)( ),double A,double B,int N)。

为了实现本例的要求，把被积函数 $f(x)=x^3+3x+1$、$g(x)=x\sin x$ 和 $h(x)=2x/(3e^x)$的定义，测试它们的 main 函数和改写过的 Integrate 函数，列出如下：

```
#include "stdio.h"
#include "conio.h"
#include "math.h"
double f(double),g(double),h(double);/*被积函数声明*/
/*下面定义通用积分函数*/
double Integrate(double (*f)( ),double A,double B,int N)
 {double h=(B-A)/N,Area=(f(A)+f(B))*h/2;
 int k;
 for(k=1;k<N;k++)
   Area+=h*f(A+k*h);
   return Area;
}

double f(double x) /*定义被积函数 f(x)*/
{
 return x*x*x+3*x+1; /*返回被积函数 f 在 x 点的函数值*/
}

double g(double x) /*定义被积函数 g(x)*/
{
 return x*sin(x); /*返回被积函数 g 在 x 点的函数值*/
}

double h(double x) /*定义被积函数 h(x)*/
{
 return 2*x/3/exp(x); /*返回被积函数 h 在 x 点的函数值*/
}

void main()
{double A=2,B=6,S;
/*调用积分函数求函数 f(x)在[2,5]的积分，其中把该区间 100 等分*/
 S=Integrate(f, A,B,100);
 printf("f(x)=%lf\n",S);
/*调用积分函数求函数 f(x)在[2,5]的积分，其中把该区间 200 等分*/
 S=Integrate(g,1,5,200);
 printf("f(x)=%lf\n",S);
/*调用积分函数求函数 f(x)在[2,5]的积分，其中把该区间 1000 等分*/
S=Integrate(h,0,2,1000);
 printf("f(x)=%lf\n",S);
 getch( );
}
```

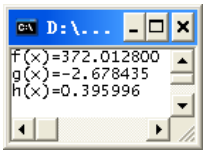

图 6.6　例 6-23 的运行结果

使用函数指针变量还应注意以下两点：

（1）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。

（2）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。

函数指针和返回指针的函数不同：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。

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()是两个完全不同的概念。int(*p)()是一个变量定义，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()则不是变量定义而是函数定义，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。作为函数定义，在括号内最好写入形式参数，如 int *p(void)，这样便于与变量定义区别。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。

6.7 带参数的 main 函数及其应用程序范例

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：

```
void main (int argc, char *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。

由于 main 函数不能被其他函数调用，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。

什么是命令行？C 语言将命令行看作由空格分隔的若干字符串，每个字符串看作是一个命令行参数。由第一个字符串（命令本身）开始从 0 编号。程序执行时，每个参数被处理作字符串，在程序中按照规定方式使用它们。通过主函数的参数获取命令行参数。

DOS 提示符下命令行的一般形式为：

可执行文件名 参数 参数……

例如：

```
rename  a.txt  b.txt
```

作用是把文件 a.txt 改名为 b.txt。其中 a.txt 和 b.txt 是命令 rename 的命令行参数。

设有程序 program1；启动程序的命令行是：

```
program1  there are five arguments
```

这时 program1 是编号为 0 的命令行参数，there 是编号为 1 的命令行参数，依此类推，共 5 个命令行参数。

C 语言程序可通过 main 的参数获取命令行参数。main(void)表示不处理命令行参数。

main 开始执行时，argc 是命令行参数个数；argv 指向命令行中各字符串（参数均按字符串处理）的首地址，前 argc 个指针指向各命令行参数串，最后有一个空指针。数组元素初值由系统自动赋予。

argv[0]指向被运行程序的完整路径及程序文件名。

argv[1]指向命令行程序名后的第一个字符串。

argv[2]指向命令行程序名后的第二个字符串。

……

argv[argc-1]指向命令行程序名后的最后一个字符串。

argv[argc]空指针（NULL）。

【例 6-24】 输出命令行参数示例。

```
#include "stdio.h"
void main(int argc,char *argv[])
{
    while(argc-->0)
    {
    printf("%s\n",*++argv);
    }
}
```

TC 2.0 中，在 Options|Arguments 子菜单项中输入程序运行所需要的参数，如“abc 100 200”（注：不包括“”，用空格分隔不同的字符串），然后按 Ctrl+F9 运行程序，程序就把输入的字符串作命令行参数处理，输出三个字符串。

【例 6-25】

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

说明：在 Options 菜单下 Arguments 子菜单项中输入的是“20”，作为字符串对待，那么命令行中有两个参数，第二个参数 20（即 argv[1]）赋给 n。在程序中*++argv 的值为字符串“20”，然后用函数 atoi 把它换为整型值赋给 n，作为 while 语句中的循环控制变量，输出 20 个偶数。

注意：命令行参数所传送的数据全部都是字符串。即便传送的是数值，也是按字符串方式传送给主函数。程序中使用这些参数时，还需要将数字字符串转换成数值型数据。C 语言标准库函数提供了一些相关的数据类型转换函数。

【例 6-26】 用一个程序实现字符串的加密和解密。

```
#include "stdio.h"
#include "stdlib.h"
void main(int argc,char *argv[])
{
    char ch,*str;
    if (argc!=3)  printf("Arguments Amount Error!\n");
    else
    {
   ch=*argv[1];
   str=argv[2];
/* printf("%s,%s\n",str,argv[2]);   */
   switch(ch)
   {
      case '+':
        while(*str!='\0')
```

```
                {  *str=*str+17; str++;  }
                break;
            case '-':
               while(*str!='\0')
                {  *str=*str-17;  str++;   }
                break;
            default: printf("The second Argument Error!\n");
          }
          printf("%s\n ",argv[2]);
          }
}
```

说明：程序在命令行中输入“+字符串”，表示对字符串加密输出，输入“-字符串”，表示对字符串解密，加密解密的过程就是取出字符串中的每个字符然后加减 17 得到。读者可以根据这个方法设计更加复杂的加密解密程序。

6.8 指向指针的指针及其应用程序范例

指针变量本身也是变量，它在存储器中也有地址。如果指针变量存放的又是另一个指针变量的地址, 则称这个指针变量就是指向指针的指针变量，即指向指针的指针，也叫二级指针。指向指针的指针中存放的是指针变量地址。

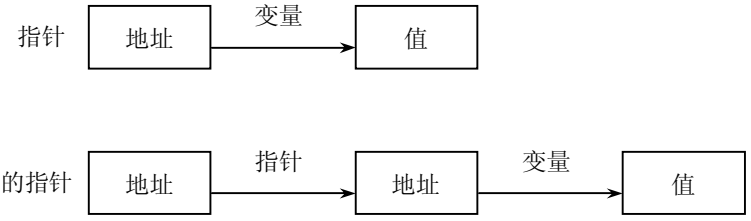

图 6.7 指针和指针的指针示意图

指向指针的指针变量定义的一般形式为：

类型标识符 ** 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。

指针的指针使用：

```
int i=2;        /*定义整型变量*/
int *p1,**p2;  /*定义 p1 为整型指针， p2 为整型指针的指针*/
p1=&i;          /*i 的地址=>p1，即 p1 指向变量 i  */
p2=&p1;         /*指针 p1 的地址=>p2，即指针 p2 指向指针变量 p1*/
```

对变量 i 的访问可以是 i，*p1，又因为*p2=p1，即**p2=*p1，所以对变量 i 的访问可以是 i，*p1，**p2。

下面举一个例子来说明这种关系。

【例 6-27】 指针的指针使用举例。

```
#include "stdio.h"
void main()
{
   int x,*p,**pp;
   x=100;
```

```
    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，读者可以学会指向指针的指针变量的定义和使用方法。

【例 6-28】 指针的指针访问指针数组示例。

```
#include "stdio.h"
main()
{
   char *ps[]={ "BASIC","DBASE","CLanguage","FORTRAN","PASCAL"};
   char **pps;
   int i;
   for(i=0;i<5;i++)
   {
      pps=ps+i;
      printf("%s\n",*pps);
   }
}
```

程序中首先定义了指针数组 ps 并作了初始化赋值，又定义了 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。再通过这些地址即可找到该字符串。

【例 6-29】 指针的指针访问指针数组（用指针的指针来指向数组中的各元素并输出）。

```
#include "stdio.h"
main()
{
    int k, **p,a[5]={1,2,3,4,5};
    int  *num[5];
for (k=0; k<5; k++)
    {num[k]=&a[k];}
    p=num;
    for (k=0; k<5; k++)
    {
       printf("%5d",**p);
       p++;
    }
    printf("\n");
}
```

注：在定义指针数组时直接初始化，如：

```
int  *num[5] ={&a[0], &a[1],&a[2],&a[3],&a[4]};
```

在 TC 2.0 中不允许，但在后续版本中可以。

*6.9 动态数组的实现

6.9.1 C 程序的内存映像

动态存储分配，在任何一个变量使用前，都必须完成关于存储方面的有关安排：存放位置、占据多少存储单元。这个工作叫存储分配。

在前面的程序设计中存在一个问题，就是在编写程序时，如果无法确定要处理数据的规模，如字符串的长度、数值数据的个数等，为保证在数据规模足够大时也能进行处理，在定义变量时

就为它说明一个特别大的空间，而在实际运行时，可能只需要这空间的很小一部分，也有可能给出的空间还不够大。这对内存资源利用和程序设计都是不利的。我们希望计算机能按需分配内存，在运行时需要多大的空间就分配多大的空间，这样就能完全解决上面提到的问题。

上面提出的问题，实际上是计算机内存的动态管理问题。C 语言的动态内存管理函数，实现了动态内存的分配与释放。根据运行中的需要分配存储，取得存储块使用，称为动态存储分配。在运行中根据需要动态进行。

程序里怎样使用分配的存储块？程序是通过名字来使用变量的。动态分配的存储块没有名字，需要其他访问途径，也就是借助于指针。用指针指向存储块，间接使用指定的存储。访问动态分配存储是指针的最重要用途。与此对应：动态释放，不用的动态存储块应及时交还给系统。

动态分配/释放由动态存储管理系统完成，这是程序运行系统的子系统，管理着称为堆（heap）的存储区。大部分常规语言都有这种机制。

6.9.2 动态内存分配与释放函数

void 指针概念：(void *)类型的指针叫通用指针，可以指向任何的变量，C 语言允许直接把任何变量的地址作为指针赋给通用指针。

当需要使用通用指针所指的数据参加运算时，需要写出类型强制转换。如通用指针 vp 所指空间的数据是整型数据，p 是整型指针，用下式转换：

```
p=(int *)vp;
```

标准动态存储管理函数原型在标准头文件<stdlib.h>中描述。

1. *存储分配函数* malloc()

```
void *malloc(unsigned  size);
```

size 是足够大的整数。

返回值类型（void *）：通用指针。

功能：分配一块长度为 size 字节的连续空间，并将该空间的首地址作为函数的返回值。如果函数没有成功执行，返回值为空指针（NULL 或 0）。由于返回的指针的类型为 void，应该通过显式类型转换后才能存入其他基类型的指针变量中，否则会有警告提示。例如：

申请一个动态的整型存储单元：

```
int *pi;
pi=(int *)malloc(sizeof(int));
```

申请 10 个动态的整型存储单元：

```
int *pj;
    pj=(int *)malloc(sizeof(int)*10);
```

由于 malloc 返回的是 void 类型的指针，所以需要进行强制转换，其目的是防止存储器校准过程中可能出现的不安全性。

注意：

（1）函数 malloc 的原型为 void *malloc();必须在程序头部加入#include "alloc.h"。

（2）指针必须指向已经分配好了的空间才有意义。

```
void main ()
{       char *p;
        *p='a';      /*错误 ！因为 p 没有被赋予有效的指针*/
...
}
```

正确使用如下所示：

```
char *p,c;
p=&c;
```

或

```
char *p;
p=(char*)malloc(1);   /*申请一个字符空间*/
```

（3）为了使指针使用安全无错，在申请空间时，应该检测 malloc 函数是否成功分配。方法是判断该函数的返回值。常用的方法是：

```
if  ((pc=(int*)malloc(10))= =NULL)
{…/* 对分配不成功的处理 */
}
```

使用动态存储分配函数应该注意以下几点：

（1）空间大小计算要使用 sizeof 函数进行计算；

（2）调用 malloc 函数后，一定要检查返回值；

（3）结果强制转换后才能赋值使用；

（4）得到的空间使用时不允许越界；

（5）分配成功后关于存储块的管理，系统完全不进行检查；

（6）动态存储块的存在期，在其分配成功时开始，只有在用 free 语句释放才能导致其存储期的结束。

2. *带初始化的存储分配函数* calloc()

```
void calloc(unsigned n, unsigned  size);
```

n 是指数据项数；size 是指每项数据的字节数。

返回值类型为（void *）：通用指针，需要通过类型强制转化成特定的指针类型。

功能：分配一块能够放下大小为 n*size 的存储块，全部内容清 0，返回指向这个块的指针，如果存储申请不能满足，返回空指针。

其函数原型为：

```
void *calloc(unsigned  n, unsigned  size);
```

功能：以 size 为单位大小共分配 n*size 个字节的连续空间，并将该空间的首地址作为函数的返回值。如果函数没有成功执行，返回值为空指针（NULL 或 0）。该函数比 malloc 函数方便之处在于，当动态分配数组空间时，malloc 函数必须手工计算出数组的总字节数，而 calloc 函数不用计算。例如：

```
int *p;
p=(int *)calloc(10,sizeof(int));
```

3. *存储块态释放函数* free()

```
void free(void *p);
```

功能：释放指针 p 所指的存储块。如果 p 的值为空，什么也不做。

调用 free(q)后，p 变量的指向没有改变。但不能再使用它的值。除非重新指向新的数据单元。程序中，应养成对不再使用的存储块立刻释放的习惯。避免造成内存泄露。

free()函数的调用格式：

free(指针变量名)

free()函数的功能是释放以指针变量名所指的位置开始的存储块，以分配时的存储块为基准。

注意：

（1）free(p)不改变 p 的内容。但释放以后就不能用任何的指针再访问这个存储区。

例如，p 和 q 指向同一地址，free(p)以后，q 所指的空间也将不能再使用。因为这部分空间已经被系统回收了。

（2）为了节省空间，如果想为一个指针第二次申请空间，且第一次申请的空间已经无用，

应先将第一次申请的空间释放。

```
pc=malloc(9);
free(pc);
pc=malloc(20);
```

（3）free 与 malloc 必须配对使用，使用 malloc 申请的空间必须用 free 释放。

4. 重新分配存储块函数 realloc()

```
void *realloc(void *p, unsigned n);
```

功能：更改前面做过的存储分配，指针 p 指向一个过去分配的存储块，n 表示现在希望的存储块大小，如果新的要求不能满足，返回 NULL，p 仍指向原来的位置。新的分配要求达到满足，返回新存储块的指针，如果新的存储块大于以前的字节数，原来块中的内容不变。其余部分不进行初始化。不能再通过 p 指针使用原来的存储块。

【例 6-30】 动态内存分配与释放函数使用示例。

```
#include <stdio.h>
#include <string.h>
#include <alloc.h>
void main(void)
{
   char *str=NULL;
   str = (char *)malloc(sizeof(char)*10);     /* 动态分配内存空间*/
   if (str == NULL)
   {
      printf("Not enough memory to allocate buffer\n");
      exit(0);
   }
    strcpy(str, "Hello!");                    /* 字符串复制*/
    printf("String is %s\n", str);            /* 字符串输出*/
    free(str);                                /* 内存空间释放*/
}
```

说明：程序中使用语句 str = (char *)malloc(sizeof(char)*10); 动态分配长度为 sizeof(char)*10 的内存空间。用字符指针指向分配的内存空间。使用语句：free(str); 释放分配的内存空间。

动态存储分配特别需要注意的问题：

（1）无用单元。例如：

```
char *p= (char *)malloc(1000);
char *q= (char *)malloc(1000);
p=q;
```

无用单元是程序中的一部分内存，如第一次动态分配的内存，当 p=q 后，使得原来 p 指向的 1000 个字节存储块没有被使用，也没有被释放给系统，造成白白的浪费，虽然不会对程序造成伤害，如果程序中不予重视，继续发生类似情况，会逐渐吞噬内存，最终导致程序没有足够的内存，C 中不提供自动无用单元收集机制，它把无用单元收集的责任留给程序员。

（2）悬挂引用。例如：

```
char *p; char *q;
p= (char *)malloc(1000);
q=p;
free(p);
*q='A';
```

当指针所指向的内存单元被删除，但程序员认为那个地址仍然是合法的，如 q，就产生了悬挂引用，这是多个指针保存相同地址的结果，悬挂引用是致命的错误。

6.9.3 一维动态数组的实现

在程序运行过程中，数组的大小是保持不变的。这种数组称为静态数组。静态数组的缺点是：由于事先无法准确估计数据量的情况，无法做到既满足程序处理需要，又不浪费存储空间。若分析正确，定义适当大小的数组，一般都能处理。若定义尽可能大的数组以满足任何需要，则浪费大量存储资源。如有多个这种数组就更难办。系统可能无法容纳几个大数组，但实际上它们并不同时需要很大空间。

所谓动态数组是指在程序运行过程中根据实际需要指定大小的数组。

在 C 语言中，可利用动态内存的分配和释放函数，利用指向数组的指针变量作为数组名使用的特点来实现动态数组。动态数组的本质是指向数组的指针变量。

【例 6-31】 一维动态数组的实现示例。

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
    int i,n;
    int *p=NULL;
    printf("Enter Integer Number n:");
    scanf("%d",&n);
    p=(int *)malloc(n*sizeof(int)); /* 分配动态数组使用的内存空间*/
    if (p==NULL)
    {
       printf("Not enough memory to allocate buffer\n");
       exit(0);
    }
    for (i=0;i<n;i++)
      *(p+i)=i;
    for (i=0;i<n;i++)
          printf("%d, ",*(p+i));
    printf("\n");
    free(p);              /*释放分配的内存空间*/
 }
```

6.9.4 二维动态数组的实现

如果我们在解决实际问题时，希望二维数组的行和列的长度都是可以变化的，就需要建立一个动态的二维数组。

首先定义一个指针的指针：

```
int   **pa;
```

现在，我们适当的为 pa 赋值，可以动态申请存储空间，生成一个行数和列数都可变的二维数组。

```
#define ROW   3;
#define COL    4;
```

第一步：为二级指针申请一个指针数组：

```
pa=(int **)malloc(ROW*sizeof(int));
```

pa 指向动态申请的一组指针，指针的个数就是数组的行数。

第二步：为指针数组的每个指针元素申请存储空间。

```
for (i=0;i<n;i++)
   pa[i]=(int *)malloc(COL*sizeof(int));
```

经过以上步骤，可以建立行数和列数都可变的二维数组。

(pa+i)是动态二维数组的第 i 个元素的内容，指向第 i 行第 0 列元素。(pa+i)+j 就指向动态二维数组的第 i 行第 j 列的元素，使用*(*(pa+i)+j)这和静态数组访问方式是完全一样的。

【例 6-32】 用动态数组和指针方式输入一个数组，并输出。

```
#include <alloc.h>
#include <stdio.h>
#include <stdlib.h>
#define  ROW  2
#define  COL  3
void main()
{
    int i,j;
    int **pa=NULL;
    pa=(int **) malloc(ROW*sizeof(int *));  /*申请一个指针数组*/
    if (pa==NULL)
    {
       printf("Not enough memory to allocate buffer\n");
       exit(0);
    }
    for (i=0;i<ROW;i++)
       pa[i]=(int *)malloc(COL*sizeof(int)); /*每个指针申请存储空间*/
    for (i=0;i<ROW;i++)
      for (j=0;j<COL;j++)
            scanf("%d",*(pa+i)+j);
    for (i=0;i<ROW;i++)
       for (j=0;j<COL;j++)
            printf("%d, ",*(*(pa+i)+j));
    for (i=0;i<ROW;i++)
       free(pa[i]);          /*释放每个申请的存储空间*/
    free(pa);
 }
```

*6.10 图形处理函数的简单应用

Turbo C 提供了非常丰富的图形函数，所有图形函数的原型均在 graphics. h 中，本节主要介绍图形模式的初始化、独立图形程序的建立、基本图形功能、图形窗口以及图形模式下的文本输出等函数。另外，使用图形函数时要确保有显示器图形驱动程序*.BGI 文件，同时将集成开发环境 Options/Linker 中的 Graphics lib 选为 on，只有这样才能保证正确使用图形函数。

1. 图形模式的初始化

不同的显示器适配器有不同的图形分辨率。即是同一显示器适配器, 在不同模式下也有不同分辨率。因此, 在屏幕作图之前, 必须根据显示器适配器种类将显示器设置成为某种图形模式, 在未设置图形模式之前, 微机系统默认屏幕为文本模式(80 列, 25 行字符模式), 此时所有图形函数均不能工作。设置屏幕为图形模式, 可用下列图形初始化函数:

```
void far initgraph(int far *gdriver, int far *gmode, char *path);
```

其中 gdriver 和 gmode 分别表示图形驱动器和模式, path 是指图形驱动程序所在的目录路径。注意：是文件扩展名为.BGI 文件所在目录。

图形驱动程序由 Turbo C 发行商提供, 文件扩展名为.BGI。根据不同的图形适配器有不同的图形驱动程序。例如对于 EGA、 VGA 图形适配器就调用驱动程序为 EGAVGA.BGI。一般在 TC 目录中 BGI 子目录下。

【例 6-33】 使用图形初始化函数设置 VGA 高分辨率图形模式。

```
#include "stdio.h"
```

```
#include "conio.h"
#include "graphics.h"
void main()
{
    int gdriver, gmode;
    gdriver=VGA;
    gmode=IBM8514HI;
    initgraph(&gdriver, &gmode, "c:\\tc");
    bar3d(100, 100, 300, 300, 50, 1);
    getch();
    closegraph();
}
```

程序把 gdriver 图形驱动器设置成 VGA 显示模式，把 gmode 显示器模式设置为 IBM8514HI 高分辨率方式。用 initgraph 函数将屏幕设置为图形模式。语句：

```
Bar3d(100, 100, 300, 300, 50, 1);
```

在屏幕上显示一个三维的立方体，语句：

```
closegraph();
```

退出图形状态而进入文本方式（Turbo C 默认方式）。

有时编程者并不知道所用的图形显示器适配器种类，或者需要将编写的程序用于不同图形驱动器, Turbo C 提供了一个自动检测显示器硬件的函数，其调用格式为：

```
void far detectgraph(int *gdriver, *gmode);
```

其中 gdriver 和 gmode 的意义与上面相同。

下面是自动进行硬件测试后进行图形初始化的例子。

【例 6-34】 使用图形初始化函数设置 VGA 高分辨率图形模式。

```
#include "stdio.h"
#include "conio.h"
#include "graphics.h"
void main()
{
    int gdriver, gmode;
    detectgraph(&gdriver, &gmode);     /*自动测试硬件*/
    initgraph(&gdriver, &gmode, "c:\\tc");  /* 根据测试结果初始化图形*/
    bar3d(10, 10, 150, 250,50,1);
    getch();
    closegraph();
}
```

上例程序中先对图形显示器自动检测，然后再用图形初始化函数进行初始化设置，但 Turbo C 提供了一种更简单的方法，即用 gdriver=DETECT 语句后再跟 initgraph()函数就行了。采用这种方法后，例子中 gdriver 的定义可改为：

```
int gdriver=DETECT;
```

退出图形状态的函数 closegraph()，其函数原型为：

```
void far closegraph(void);
```

调用该函数后可退出图形状态而进入文本方式（Turbo C 默认方式），并释放用于保存图形驱动程序和字体的系统内存。

2. 屏幕颜色的设置和清屏函数

对于图形模式的屏幕颜色设置，同样分为背景色的设置和前景色的设置。在 Turbo C 中分别用下面两个函数。

设置背景色：　　void far setbkcolor(int color);

设置作图色（前景色，即画笔颜色）：　　void far setcolor(int color);

其中 color 为图形方式下颜色的规定数值，对 EGA，VGA 显示器适配器，有关颜色的符号

常数及数值见表 6-1 所示。

表 6-1　颜色符合常数及数值表

符号常数	数值	含义	符号常数	数值	含义
BLACK	0	黑色	DARKGRAY	8	深灰
BLUE	1	兰色	LIGHTBLUE	9	深兰
GREEN	2	绿色	LIGHTGREEN	10	淡绿
CYAN	3	青色	LIGHTCYAN	11	淡青
RED	4	红色	LIGHTRED	12	淡红
MAGENTA	5	洋红	LIGHTMAGENTA	13	淡洋红
BROWN	6	棕色	YELLOW	14	黄色
LIGHTGRAY	7	淡灰	WHITE	15	白色

3. 基本图形函数

基本图形函数包括画点、线以及其他一些基本图形的函数。这些函数图形函数的原型在头文件 Graphics.h 中说明。下面对部分函数作简单介绍。

（1）画点函数。

void far putpixel(int x, int y, int color);

该函数表示在指定的坐标处，按 color 给定的颜色画一个点。对于颜色 color 的值可从表 6-1 中获得，x, y 是指图形像素的坐标。

在图形模式下，是按像素来定义坐标的。对 VGA 适配器, 它的最高分辨率为 640×480，其中 640 为整个屏幕从左到右所有像素的个数, 480 为整个屏幕从上到下所有像素的个数。屏幕的左上角坐标为(0, 0)，右下角坐标为(639, 479)，水平方向从左到右为 x 轴正向，垂直方向从上到下为 y 轴正向。Turbo C 的图形函数都是相对于图形屏幕坐标，即像素来说的。

关于点的另外一个函数是：

int far getpixel(int x, int y);获取当前坐标点(x, y)的颜色值。

（2）有关坐标位置的函数。

int far getmaxx(void);返回 x 轴的最大值。

int far getmaxy(void);返回 y 轴的最大值。

int far getx(void);返回光标在 x 轴的位置。

void far moveto(int x, int y);光标从当前坐标点移到(x, y)点。

void far moverel(int dx, int dy);光标从现行位置(x, y)移到(x+dx, y+dy)的位置。

（3）画线函数。

Turbo C 提供了一系列画线函数，下面分别叙述：

void far line(int x0, int y0, int x1, int y1); 从点(x0, y0)到(x1, y1) 画一条直线。

void far lineto(int x, int y); 从当前位置到点(x, y) 画一条直线。

void far linerel(int dx, int dy);从当前位置(x, y)到点(x+dx, y+dy) 画一条直线。

void far circle(int x, int y, int radius);以(x, y)为圆心，radius 为半径，画一个圆。

void far arc(int x, int y, int stangle, int endangle, int radius);以(x, y)为圆心，radius 为半径，从 stangle 开始到 endangle 结束（用度表示）画一段圆弧线。在 Turbo C 中规定 x 轴正向为 0 度，逆

时针方向旋转，依次为 90、180、270 和 360 度（其他有关函数也按此规定，不再重述）。

void ellipse(int x, int y, int stangle, int endangle, int xradius,int yradius);以(x, y)为中心, xradius, yradius 为 x 轴和 y 轴半径，从角 stangle 开始到 endangle 结束画一段椭圆线，当 stangle=0，endangle=360 时，画出一个完整的椭圆。

void far rectangle(int x1, int y1, int x2, inty2);以(x1, y1)为左上角，(x2, y2)为右下角画一个矩形框。

void far drawpoly(int numpoints, int far *polypoints);画一个顶点数为 numpoints 的多边形，各顶点坐标由 polypoints 给出。polypoints 整型数组必须至少有 2 倍顶点数个元素。每一个顶点的坐标都定义为 x、y，并且 x 在前。值得注意的是当画一个封闭的多边形时，numpoints 的值取实际多边形的顶点数加一，并且数组 polypoints 中第一个和最后一个点的坐标相同。

【例 6-35】 用 drawpoly()函数画红色箭头的例子。

```
#include "stdio.h"
#include "conio.h"
#include "graphics.h"
void main()
{
    int gdriver, gmode, i;
    int arw[16]={200, 102, 300, 102, 300, 107, 330,
              100, 300, 93, 300, 98, 200, 98, 200, 102};
    gdriver=DETECT;
    detectgraph(&gdriver, &gmode);
    initgraph(&gdriver, &gmode, "f:\\tc");
    setbkcolor(BLUE);
    cleardevice();
    setcolor(12);       /*设置作图颜色*/
    drawpoly(8, arw);   /*画一箭头*/
    getch();
    closegraph();
}
```

（4）设定线型函数。

在没有对线的特性进行设定时，Turbo C 的默认值是：一个像素点为直径的实线，但 Turbo C 也提供了可以改变线型的函数。线型包括：直径和形状。其中直径只有两种选择：一个像素点和三像素点。而线的形状则有 5 种。下面介绍有关线型的设置函数。

```
void far setlinestyle(int linestyle, unsigned upattern,int thickness);
```

该函数用来设置线的有关信息，其中 linestyle 是线形状的规定，如表 6-2 所示。

表 6-2　线的形状（linestyle）表

含义	符号常数	数值
实线	SOLID_LINE	0
点线	DOTTED_LINE	1
中心线	CENTER_LINE	2
点画线	DASHED_LINE	3
用户定义线	USERBIT_LINE	4

（5）图形模式下的文本输出。

在图形模式下，只能用标准输出函数，如 printf(), puts(), putchar()函数输出文本到屏幕。除此

之外，其他输出函数（如窗口输出函数）不能使用，即使可以输出的标准函数，也只以前景色为白色，按 80 列，25 行的文本方式输出。

Turbo C 2.0 也提供了一些专门用于在图形显示模式下的文本输出函数。

文本输出函数：

void far outtext(char far *textstring);

该函数在当前行输出字符串指针 textstring 所指的文本。

void far outtextxy(int x, int y, char far *textstring);

该函数从坐标(x, y) 位置输出字符串指针 textstring 所指的文本。其中 x 和 y 为像素坐标。

这两个函数都是输出字符串，但经常会遇到输出数值或其他类型的数据，此时就必须使用格式化输出函数 sprintf()。

sprintf()函数的原型为：

```
int sprintf(char *str, char *format, variable-list);
```

它与 printf()函数不同之处是按 format 给定的格式，将 variable-list 的内容写入 str 指向的字符串中，返回值等于写入的字符个数。例如：

```
sprintf(s, "your name is %d", name);
```

这里 s 应是字符串指针或数组，name 为整型变量。

Turbo C 中还有很多的图形函数，在这里不作介绍，读者可参考其他 C 语言图形设计的书籍。

6.11 本章小结

1. 指针是 C 语言中一个重要的组成部分，使用指针编程有以下优点:

（1）提高程序的编译效率和执行速度。

（2）通过指针可使用主调函数和被调函数之间共享变量或数据结构，便于实现双向数据通信。

（3）可以实现动态的存储分配。

便于表示各种数据结构，编写高质量的程序。

2. 指针的运算

（1）取地址运算符&：求变量的地址。

（2）取内容运算符*：表示指针所指向的变量。

（3）赋值运算。

- 把变量地址赋予指针变量
- 同类型指针变量相互赋值
- 把数组，字符串的首地址赋予指针变量
- 把函数入口地址赋予指针变量

（4）加减运算。

对指向数组，字符串的指针变量可以进行加减运算，如 p+n，p-n，p++，p--等。对指向同一数组的两个指针变量可以相减，对指向其他类型的指针变量作加减运算是无意义的。

（5）关系运算。

指向同一数组的两个指针变量之间可以进行大于、小于、等于比较运算。指针可与 0 比较，p==0 表示 p 为空指针，但最好用 p==NULL 来比较。

3. 与指针有关的各种定义和意义

int *p;　　p 为指向整型量的指针变量。

int *p[n];　　p 为指针数组，由 n 个指向整型量的指针元素组成。

int (*p)[n];　　p 为指向整型二维数组的指针变量，二维数组的列数为 n。

int *p()　　p 为返回指针值的函数，该指针指向整型量。

int (*p)()　　p 为指向函数的指针，该函数返回整型量。

int **p　　p 为一个指向另一指针的指针变量，该指针指向一个整型量。

4. 有关指针的定义

有关指针的定义很多是由指针，数组，函数定义组合而成的。但并不是可以任意组合，例如数组不能由函数组成，即数组元素不能是一个函数；函数也不能返回一个数组或返回另一个函数。例如:

```
int a[5]();
```

就是错误的。

5. 关于括号

在解释组合定义符时，标识符右边的方括号和圆括号优先于标识符左边的“*”号，而方括号和圆括号以相同的优先级从左到右结合，但可以用圆括号改变约定的结合顺序。

6. 阅读组合定义符的规则是“从里向外”

从标识符开始，先看它右边有无方括号或园括号，如有则先作出解释，再看左边有无*号。如果在任何时候遇到了闭括号，则在继续之前必须用相同的规则处理括号内的内容。例如:

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

按照由内向外的阅读顺序，下面来分析它:

（1）*a 标识符 a 被说明为一个指针变量;

（2）(*a)()指向一个函数;

（3）*(*a)()返回一个指针;

（4）*(*a)()[10]该指针指向一个有 10 个元素的数组;

（5）*(*(*a)())[10]其类型为指针型;

（6）int *(*(*a)())[10]指向 int 型数据。

因此 a 是一个函数指针变量，该函数返回的一个指针值又指向一个指针数组，该指针数组的元素指向整型量。

习题六

一、选择题

【习题 6-1】变量的指针，其含义是指该变量的（　　）。

A．值　　B．地址　　C．名　　D．一个标志

【习题 6-2】若有语句 int *point,a;，则 point=&a;中运算符&的含义是（　　）。

A．位与运算　　B．逻辑与运算　　C．取指针内容　　D．取地址

【习题 6-3】若 x 是整型变量，pb 是整型的指针变量，则正确的赋值表达式是（　　）。

A．pb=&x　　B．pb=x;　　C．*pb=&x;　　D．*pb=*x

【习题 6-4】下面程序段的运行结果是（　　）。

```
char *s="abcde";
s+=2;
printf("%d",s);
```

A．cde　　B．字符'c'

C．字符'c'的地址　　D．无确定的输出结果

【习题 6-5】设 p1 和 p2 是指向同一个字符串的指针变量，c 为字符变量，则以下不能正确执行的赋值语句是（　　）。

A．c=*p1+*p2;　　B．p2=c　　C．p1=p2　　D．c=*p1*(*p2);

【习题 6-6】若有定义语句：

```
char a[]="It is mine";
char *p="It is mine";
```

则以下不正确的叙述是（　　）。

A．a+1 表示的是字符 t 的地址

B．p 指向另外的字符串时，字符串的长度不受限制

C．p 变量中存放的地址值可以改变

D．a 中只能存放 10 个字符

【习题 6-7】若有定义：int a[2][3]，则对 a 数组的第 i 行 j 列元素地址的正确引用为（　　）。

A．*(a[i]+j)　　B．(a+i)　　C．*(a+j)　　D．a[i]+j

【习题 6-8】 设有如下定义：

```
int  (*ptr)();
```

则以下叙述中正确的是（　　）。

A．ptr 是指向一维组数的指针变量

B．ptr 是指向 int 型数据的指针变量

C．ptr 是指向函数的指针，该函数返回一个 int 型数据

D．ptr 是一个函数名，该函数的返回值是指向 int 型数据的指针

【习题 6-9】设有定义

```
int  (*ptr)[m];
```

其中的标识符 ptr 是（　　）。

A．m 个指向整型变量的指针

B．指向 m 个整型变量的函数指针

C．一个指向具有 m 个整型元素的一维数组的指针

D．具有 m 个指针元素的一维指针数组，每个元素都只能指向整型量

【习题 6-10】若要用下面的程序片段使指针变量 p 指向一个存储整型变量的动态存储单元：

```
int *p;
p=__________ malloc( sizeof(int));
```

则应填入（　　）。

A．int　　B．int *　　C．(*int)　　D．(int *)

二、阅读下列程序，填空或给出程序运行结果

【习题 6-11】以下程序的功能是：通过指针操作，找出三个整数中的最小值并输出。

```
#include "stdio.h"
main()
{
   int *a,*b,*c,num,x,y,z;
```

```
   a=&x; b=&y; c=&z;
   printf("输入 3 个整数: ");
   scanf("%d%d%d",a,b,c);
   printf("%d,%d,%d\n",*a,*b,*c);
   num=*a;
   if  (*a>*b)   _______;
   if  (num>*c)  _______;
   printf("输出最小整数:%d\n",num);
}
```

【习题 6-12】下面程序的功能是将两个字符串 s1 和 s2 连接起来，将 s2 连接到 s1 后面。请填空。

```
#include "stdio.h"
#include "string.h"
 main()
 {  char s1[80],s2[80];
    gets(s1); gets(s2);
    conn(s1,s2);
    puts(s1);
 }
 conn(char *p1,char *p2)
   {  while(*p1)   ______;
      while(*p2)
         {  *p1=______;
p1++;   p2++; }
      *p1='\0';
}
```

【习题 6-13】以下程序将数组 a 中的数据按逆序存放，请填空。

```
 #define M  8
 #include "stdio.h"
  main()
  { int a[M],i,j,t;
    for(i=0;i<M;i++)
scanf("%d",a+i);
   i=0;j=M-1;
   while(i<j)
    {
       t=*(a+i);
*(a+i)= _______;
*(______)=t;
       i++;  j--;
     }
    for(i=0;i<M;i++
printf("%3d",*(a+i));
   }
```

【习题 6-14】给出程序运行结果。

```
#include "stdio.h"
void f(int *x,int *y);
main()
{
int a[8]={1,2,3,4,5,6,7,8},i,*p,*q;
p=a;q=&a[7];
while(p<q)
     {   f(p,q);
     p++;q--;}
for(i=0;i<8;i++)
printf("%d,",a[i]);
}
void f(int *x,int *y)
{  int t;
t=*x;*x=*y;*y=t;
```

```
}
```

【习题 6-15】给出以下程序的输出结果。

```
#include "stdio.h"
void  prt(int *m,int n);
main()
{
int a[]={1,2,3,4,5},i;
prt(a,5);
for(i=0;i<5;i++)
 printf("%d,",a[i]);
void  prt(int *m,int n)
{    int i;
for(i=0;i<n;i++)
 m[i]++;
}
```

【习题 6-16】给出下面程序的运行结果。

```
#include "stdio.h"
#include "string.h"
void  fun(char *w,int n);
main()
{  char *p;
   p="1234567";
   fun(p,strlen(p));
   puts(p);
 }
 void  fun(char *w,int n)
 { char t,*s1,*s2;
    s1=w;s2=w+n-1;
while(s1<s2)
{t=*s1++;*s1=*s2--;*s2=t;}
  }
```

【习题 6-17】给出下面程序的运行结果。

```
#include "stdio.h"
main()
{
char a[]="programming",b[]="language";
char *p1,*p2;
int i;
p1=a;p2=b;
for(i=0;i<7;i++)
       if(*(p1+i)==*(p2+i))
printf("%c",*(p1+i));
}
```

三、程序设计题（要求使用指针方法处理）

【习题 6-18】有 n 个整数，设计程序将前面各个数按顺序向后移动 k 个位置，将最后 k 个数按顺序移动到最前面。

【习题 6-19】编写程序，求出矩阵中每个行的累加和，要求用动态数组存储行的累加和。

【习题 6-20】找出一个二维数组的鞍点，即该位置上的元素在该行中最大，在列上最小，也可能没有鞍点。

【习题 6-21】编写程序判断是否是回文字符串。回文是一种“从前向后读”和“从后向前读”都相同

的字符串。如："rotor"。

【习题 6-22】编制一个字符替换函数，实现已知字符串 str 中，所有属于 ch 中的字符都用 ch2 中对应字符代替。函数原型：

```
void  replace(char *str,  char ch,  char ch2)
```

【习题 6-23】编写一个函数，删除一个字符串的指定字符，函数原型：

```
int  delStr(char *str, char ch)
```

先判断字符是否出现在字符串中，如果未出现，则返回 0，如果字符出现一次或多次，则返回字符的个数。

【习题 6-24】编写函数把参数字符串中的字符反序排列, 函数原型：

```
void  reversestr(char * str)
```

使用指针，不要使用数组下标，也不要声明局部数组来临时存储。