6.4 数组形参

●函数与一维数组

1. 一维数组元素作函数的实参

由于数组元素与相同类型的简单变量地位完全一样，因此，数组元素作函数参数也和简单变量一样，是值的单向传递。

例: 一维数组a，存放6个数据,程序把相邻两个数交换,并依次输出。

在本例中，一次只传递了数组的两个元素。数组元素a[i]和a[i+1]作为函数的参数时，a[i], a[i+1]是实参，而swap函数中的参数x,y是形参。实参向形参的传递只是简单的"值"传递，即a[i]和x, a[i+1]和y它们是两个不同的变量，a[i]在将"值"传递给x后，两个变量之间再无其它关系。

程序如下:

```
#include <stdio.h>
void swap(int x,int y)
{    int  t;
     t=x;x=y;y=t;
}
void main( )
{    int a[6]={1,0,3,2,5,4},i;
     for(i=0;i<6;i+=2)
         swap(a[i],a[i+1]);
     for(i=0;i<6;i++)
           printf("%d  ",a[i]);

}
```

改为传地址后, 数组a?

2. 数组名作实、形参

◆在一维数组中，用"数组名［下标］"表示的数组元素相当于一个普通变量；而不带下标的数组名代表一批变量，也可以把它看成一个特殊的变量，因为它存放该数组的首地址（即数组第一个元素的地址）。

◆在函数中，直接用数组名作参数时，则传送的是地址值，一般情况下，为了能使函数处理不同的数组，函数应设置数组形参，即把实参数组的首地址传递给形参数组，而不是将全部数组元素都复制到函数中去。地址传递后，实参、形参数组共享相同的内存单元，也就是说形参数组和实参数组其实就是同一个数组。

例如，下面定义的函数sum()用于求n个数之和，这个函数正确地设置有两个形参，一个形参是数组，用于对应实在数组；另一个形参是整型的，用于指定求和数组的元素个数。

```
int sum(int a[], int n)
{  int i, s;
   for(s = i = 0; i < n; i++)
      s += a[i];
   return s;
}
```

利用以上定义的函数sum()，如果有以下变量定义：

```
int  x[] = {1, 2, 3, 4, 5};
int  i, j;
```

则语句

```
i = sum(x, 5);
j = sum(&x[2], 3);
printf("i = %d\nj = %d\n", i, j);
```

将输出

```
i = 15
j = 12
```

将数组x的地址(&x[0])传送给数组形参a,

将数组x中的x[2]的地址(&x[2])传送给形参a

对于数组类型的形参来说，函数被调用时，与它对应的实在数组由多少个元素是不确定的，可能会对应一个大数组，也可能会对应一个小数组，甚至会对应数组中的某一段。

可在数组形参说明中，形参数组不必指定数组元素的个数。

为了正确指明某次函数调用实际参与计算的元素个数，应另引入一个整型形参来指定，就如函数sum()那样。int sum(int a[], int n)其中的n指明了数组a的大小,即a的大小是通过n获到， a也可称为可调数组。

因传递给数组形参的实参是数组段的开始地址，函数内对数组形参的访问就是对实参所指数组的访问。函数也可以改变实参所指数组元素的值。例如，以下initArry()函数的定义：

```
void initArray(int x[], int n, int val)
{ int  i;
  for(i = 0; i < n; i++)x[i] = val;
}
```

函数initArray()是给数组元素赋指定值的。如有数组定义

```
int a[10], b[100];
```

语句

```
initArray(a, 10, 1); //为数组a的所有元素赋值1
initArray(b, 50, 2);//为数组b的前50个元素赋值2
initArray(&b[50], 50, 4);//为数组b的后50个元素赋值4
```

数组形参也可以是多维数组。当数组形参是多维时，除数组形参的第一维大小不必指定外，其他维的大小必须明确指定。例如，下面的函数sumAToB()，用于将一个n×10的二维数组各行的10个元素之和存于另一个数组中。

```
void  sumAToB(int a[][10], int b[], int n)
{ int  i, j;
  for(i = 0; i < n; i++)
    for(b[i] = 0, j = 0; j < 10; j++)
        b[i] += a[i][j];
}
```

在函数sumAToB()的定义中，对形参a的说明写成int a[][]是错误的。因二维数组的元素只是一行行存放，并不自动说明数组的列数（即每行元素个数）。如果在数组形参中不说明它的列数，就无法确定数组元素a[i][j]的实际地址。

因函数的数组形参对应的实参可以是数组某元素的地址，即数组某元素的指针，所以数组形参也是一种指针形参，只是它要求对应的实参是数组某元素的指针，而不是一般变量的指针。所以任何数组形参说明：

类型　形参名[]

都可改写成：　类型　*形参名

例如，前面的函数sum()的定义可改写成如下形式：

```
int sum(int *a, int n)
{ int  i,s;
  for(s=i=0; i<n;i++ )
    s += a[i];
  return s;
}
```

函数的形参也是函数的一种局部变量，指针形参就是函数的指针变量，函数sum()的定义又可改写成如下形式：

```
int sum(int *a, int n)
{ int s = 0;
  for(; n--; )
    s += *a++;
  return s;
}
```

【例6.8】求数组中最大元素值的函数。

```
int maxInArray(int a[], int n)
{
    int i, m;
    for(m = 0, i = 1; i < n ; i++)
      if (a[m] < a[i]) m = i;
    return a[m];
}
```

```
void main()
{
    int a[6]={47,63,23,59,131,79};
    printf("%d\n", maxInArray(a,6));
}
```

递归算法:

```
int maxInArray(int *a,int n)
{
    int  t;
    if(n==1)return *a;
    t=maxInArray (a+1,n-1);
    return *a>t?*a:t;
}
```

```
int maxInArray(int *a,int n)
{
    int  t;
    if(n==1)return a[0];
    t=maxInArray (a+1,n-1);
    return a[0]>t? a[0] :t;
}
```

在许多场合，函数是对表进行操作。为使函数的操作对象具有一般性，可为这样的函数设置表的首元素指针和元素个数等形参，而函数体用指向表元素的指针对表作处理。例如，函数sum()也可改写成以下形式。

```
int sum(int *a, int n)
{ int *ap, s;
   for(s = 0, ap = a; ap < a+n; ap++)
     s += *ap;
   return s;
 }
```

从以上的例子看出：

函数为了处理顺序存储的线性表，通常至少需要两个形参，一个是线性表首元素的指针，另一个是线性表的元素个数。

如果用于字符串处理，表示数组元素个数的形参就可省去。这是因为：

字符串中含有字符串结束符的特殊性，可以利用字符串结束符判断是否继续执行.

例如：

将一个字符串从一个函数传递给另一个函数，可以使用指向字符串的指针变量作为参数。

【例6.9】字符串拷贝函数 strcpy()

```
void strcpy (char *to, char * from)
 {  while (*to++ = *from++);}
void main()
 {  char a[ ]="Fudan University", b[100];
     strcpy (b, a);
     printf ("string a=%s\nstring b=%s\n", a,b);
}
```

注意：

◆C编译系统对形参数组大小不作检查，因此形参数组可以不指定大小，在数组名后跟一对空的方括号即可，而其大小由相应的实参数组决定。

◆在调用时将实参数组的首地址传到形参数组名，也就是说，形参数组并不在内存中重新申请数组的空间，而是和实参数组共享存储单元。

◆实参数组和形参数组类型应保持一致。如在上例中，实参数组a和形参数组to的元素类型都为char类型。

【例6.10】两字符串比较函数 strcmp()

该函数的功能是比较两字符串的大小：strcmp()有两个形参s和t；分别为两个要比较字符串的首字符指针。如s所指的字符串小于t所指的字符串，函数返回值小于0；如s所指字符串大于t所指字符串，函数返回值大于0；如果两个字符串相同，则函数返回0。

函数以s和t为两字符串的顺序考察工作指针，用循环比较s和t所指的两字符，直至两个字符不相等结束比较循环。循环过程中要判字符串是否结束，如果字符串已结束，则两字符串相同，函数返回0；否则，两字符指针分别加1，准备比较下一对字符。当两字符不相等结束循环时，函数可直接以两字符的差返回。

```
int strcmp(char *s,char *t)
//return <0,if s<t;0,if s==t;>0,if s>t
{
     while (*s == *t){//对应字符相等循环
        if (*s == '\0') return 0;
            s++; t++;
     }
     return *s - *t; /* 返回比较结果 */
}
```

函数与一维字符数组综合示例

编写字符串整理的函数：void squeeze(char *s1,char *s2),该函数将s1字符串中所在字符s2也出现的字符去掉。如有：char t[]="AABCXABBCZABC";则调用函数squeeze(t, "AC");后，代码printf("%s\n",t);将输出BXBZB,现用指针编写.

```
void squeeze(char *s1,char *s2)
{
   char *p,*k;
   for (;*s2;s2++)
     for(p=s1;*p;)
       if(*p==*s2)//若相等，把相同的字符移掉
         for(k=p;*k;k++)*k=*(k+1);
       else   p++;//不同时，后移一位
}
```

```
void main()
{
 char t1[]="AABCXAABCCZABC";
 char  t2[]="AC";
 squeeze(t1,t2);
 printf("str1=%s\n",t1);
}
```

也可以将其改为递归函数

```
void squeeze(char *s1,char *s2)
{
  char *p,*k;
  if(*s2)
  {
      for(p=s1;*p;)
          if(*p==*s2)
              for(k=p;*k;k++)*k=*(k+1);
          else p++;
      s2++;
      squeeze(s1,s2);
   }
}
```

函数与一维数组小结：

本节在分析函数调用过程的参数传递问题时，特别强调要搞清楚实参向形参传递的是"值"还是"地址"？严格说来，"地址"也是值（地址值），但它们有一个显著的区别：

① "值传递"在实参将"值"传递给形参后，对形参的修改不会影响到对应的实参。这可以理解为实参和形参各自占用不同的存储空间，实参在将"值"传递给形参后，二者就脱离关系了；

② "地址传递"在实参将"地址"值传递给形参后，形参就和实参共享同一地址单元，而不另外分配存储空间。这可以理解为形参名和实参名只是同一存储单元的两个不同引用名而已，因而对形参的修改就相当于是对实参的修改。

函数与二维数组

●二维数组元素：与相同类型的简单变量地位完全一样。因此，数组元素作函数参数也和简单变量一样，也是值的单向传递。但要注意：二维数组元素在引用时其下标个数为2。

●二维数组名：在二维数组中，没有下标的数组名也可以看成一个特殊变量，存放该二维数组第一个元素的地址。在函数中，若直接用二维数组名作参数，则传送的也是地址值，即将实参数组的首地址传递给形参数组，结果是：形参数组和实参数组共享相同的内存单元。

回顾：

前面已经介绍，一维和二维数组在内存中都是按行连续存放的。所以，当用一个指针变量指向一维或二维数组的首元素后，利用该指针变量就可以访问数组中的任意一个元素。

例如一维数组：先定义

int a[4]={80,81,82,83};

int *p=a ,i;或 int *p=&a[0];

后，可通过指针引用各个数组元素，如图6.12所示。

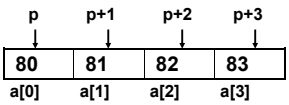


图6. 12　利用指向一维数组首元素的指针访问各元素

依次输出各元素的语句可写为：

```
for( i=0; i<4; i++ )
    printf ("%5d",*(p+i));// printf ("%5d",*p++));
```

又如二维数组：先定义
int a[2][4]={{80,81,82,83},{84,85,86,87}};
int *p=&a[0][0],i,j;// p=a[0];
后，可通过指针引用各个数组元素，如图6.13所示。

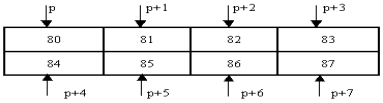


图6.13 利用指向二维数组首元素的指针访问各元素

依次输出各元素的语句可写为：

```
for( i=0; i<8; i++ )
   printf ("%5d",*(p+i));
     //printf ("%5d",*p++);
```

```
for( i=0; i<2; i++ )
   for(j=0;j<4;j++)
        printf ("%5d",*(p+i*4+j));
```

从以上例子可以看出，对于指向数组元素的指针，每次指针加1只是指针后移一个元素。那么，我们能否定义一种指向数组的指针，每次指针变量增减1个单位，指针变量就会向前或向后移一整行呢？回答是肯定的。

6.5指向二维数组一整行的指针

前面介绍过，在二维数组中，每一行都可以看成是一个一维数组，因此我们可以定义一个指向一维数组的指针（指向的数组长度等于二维数组列数），然后通过初始化或赋值使它指向二维数组的首行，则我们就可以通过这个指针访问二维数组中任何一行，继而能访问数组中的任意一个元素。

●指向二维数组一整行的指针的定义与引用

使指向一维数组的指针变量指向二维数组首行的方法有如下两种：

(1) 初始化：

类型说明符 （*指针变量名）[长度]=&首行一维数组名；

或：类型说明符 （*指针变量名）[长度]=二维数组名；

(2) 赋值：

指向一维数组的指针变量=&首行一维数组名；

或：指向一维数组的指针变量=二维数组名；

表示定义一个指向一维数组的指针变量，所指向的数组的元素类型由“类型说明符”说明，数组的元素个数由“一维数组长度”说明。初值通常是一维数组的首地址，形式为"&一维数组名"。

●用指向一维数组的指针指向二维数组的首行

例如：定义

```
short a[2][4]={1,2,3,4,5,6,7,8};
short (*p)[4]=&a[0],i,j;
```

或int (*p)[4]=a;后，内存状态如下图6.14所示。

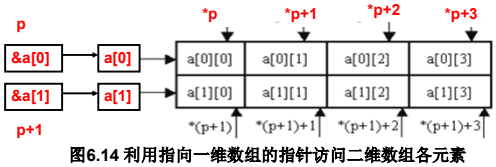


图6.14 利用指向一维数组的指针访问二维数组各元素

依次输出各元素的语句可写为：

```
for(i=0;i<2;i++)
{  for(j=0;j<4;j++)
       printf("%4d",*(*(p+i)+j));//p[i][j];
   printf("\n");
}
```

结合图6.14，程序中的p,&a[0],a[0], &a[0][0]它们是否是同一地址？
答案是肯定的。为什么呢？我们可以这样来理解：

① a是二维数组名，代表数组首地址；
② a[0]是一维数组名，代表第一行的首地址；
③ &a[0][0]是取数组首元素的地址；
④ &a[0]是取数组首行的地址，而首行的地址就是首元素的地址；
⑤ &a与p是针对整个数组a而言的：&a是取整个数组a的首地址。p的值是&a，也表示整个数组a的首地址，从另一个方面来看，p是指向数组的指针，p自然是存放数组的首地址。
⑥ p是指向一维数组a[0]（二维数组第一行）的，自然存放a[0]首地址；
而数组首地址、数组首行的地址、数组首元素的地址是相同的，因而这五个地址是相同的。如果结合图6.14，则更容易验证该结论：&a[0][0]是取数组首元素的地址，&a[0]是取数组首行的地址，&a[0][0]和&a[0]是相同的。从图中可以看出，p的值是&a[0], a[0]的值是&a[0][0]，所以p、a[0]的值与前者也是相同的。

请见下面例子：

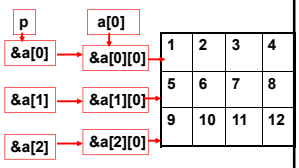
```
#include <stdio.h>
void main(){  int a[3][4] = {{1,  2,  3,  4},
        {5,  6,  7,  8}, {9, 10, 11, 12}};
 int i, (*p)[4]=&a[0];//a
 printf("p=%x\n",p);
 printf("a=%x\n",a);
 printf("&a=%x\n",&a);
 printf("&a[0]=%x\n",&a[0]);
 printf("a[0]=%x\n",a[0]);
 printf("&a[0][0]=%x\n",&a[0][0]);
 for(i=0;i<12;i++)
 { if(i%4==0)printf("\n");
       printf("%4d",*(*p+i));
  } printf("\n");
}
```

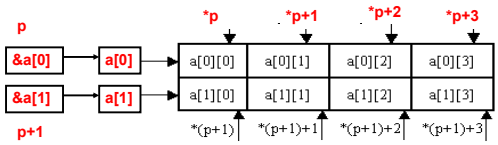
运行结果:

```
p=12ff50
a=12ff50
&a=12ff50
&a[0]=12ff50
a[0]=12ff50
&a[0][0]=12ff50
```

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

```
for(i=0;i<3;i++)
 {  printf("\n");
        for(j=0;j<4;j++)
      printf("%4d",p[i][j]);
 }
// printf("%4d", *(p[i]+j));
//printf("%4d",*(*(p+i)+j));
```





从上图可以看出，指向一维数组的指针p是一个二级指针，它指向二维数组的第一行，由于指针p是用于指向整个一维数组（即第一行）的，所以p+1则会跳过这个一维数组而指向下一个一维数组（即第二行）。同理，若有第三行、第四行，只要把指针名分别变为p+2、p+3即可，依次类推。

由于一维数组名可看作数组的第一个元素(下标为0)的地址，那么二维数组名a可以看作a的首元素一维数组a[0]的地址，即表示二维数组0行的首地址。

a+i可以看作数组a的元素一维数组a[i]的地址。

即:

a[0]是一维数组a[0]的首元素a[0][0]的地址;

a[1]是一维数组a[1]首元素a[1][0]的地址。

一般地，a[i]是一维数组a[i]首元素a[i][0]的地址。

a+i与a[i]的意义不同，a+i表示整个一维数组a[i]的开始地址，a[i]表示一维数组a[i]首元素a[i][0]的地址。

因此a[i]可写成*(a+i)，所以a+i 与*(a+i)也有不同意义。

a[i]或*(a+i)表示二维数组a的元素a[i][0]的地址，即&a[i][0]。

根据地址运算规则，a[i]+j即代表数组a的元素a[i][j]的地址，即&a[i][j]。

因a[i]与 *(a+i)等价，所以*(a+i)+j也与&a[i][j]等价。

归纳： a[i][j]也有以下三种等价表示形式：*(a[i]+j)、*(*(a+i)+j)、(*(a+i))[j]

同样： a[0][0]有以下两种等价表示形式：

*a[0] 因为 a[0] 表示 &a[0][0];

**a 因为 a 表示 &a[0]

表示形式	含　义
a	数组首地址，即&a[0]
a+i	第i行首地址，即&a[i]
a[0]、*(a+0)、*a	第0行第0列元素地址
a[i]+j、*(a+i)+j、&a[i][j]	第i行第j列元素地址
(a[i]+j)、(*(a+i)+j)、a[i][j]	第i行第j列元素的值

结论:

当指向一维数组的指针p指向二维数组a首行后，数组元素a[i][j]可表示为：

*(p[i]+j)　*(*(p+i)+j)　p[i][j]

数组元素a[i][j]的地址可以表示为：

p[i]+j　*(p+i)+j　&p[i][j]

在以上定义中， short (*p)[4]圆括号是必需的。

例如，代码， short *q[4];

定义一个指针数组q，数组q有四个元素，每个元素是一个指向整型变量的指针。

在以上定义中， short (*p)[4]圆括号是必需的。

例如，代码， short *q[4];

定义一个指针数组q，数组q有四个元素，每个元素是一个指向整型变量的指针。

【例6.11】说明指向数组元素的指针和指向数组的指针的区别的示意程序。

```
#include <stdio.h>
int main()
{ int a[3][4] = { { 1,  3,  5,  7},
  { 9, 11, 13, 15},{17, 19, 21, 23}};
  int i, *ip, (*p)[4];
  p = a+1;   ip = p[0];/* ip = &a[1][0] */
  for(i = 1; i <= 4; ip += 2,i++)
       printf("%d\t", *ip);
  printf("\n");  p = a; /* p = &a[0] */
  for (i = 0; i < 2; p++, i++)
      printf("%d\t", *(*(p+i)+1));
  printf("\n");
  return 0;
}
```

1	3	5	7
9	11	13	15
17	19	21	23

程序运行后，将输出

```
9       13      17      21
3       19
```

在程序中，开始时p指向二维数组a的第2行，p[0]或者*p是a[1][0]的地址，ip指向 a[1][0]。在第一个循环中，每次循环后修改ip，使ip增加2。在第二个循环中，每次对p的修改，使p指向二维数组的下一行，而*(*(p+i)+1)引用后i行的第2列元素。其中

((p+i)+1)

也可写成

*(p[i]+1)。

6.6指针数组

当一个数组的元素均为指针类型时，该数组称为指针数组。指针数组的定义、赋初值、数组元素的引用与赋值等操作和一般数组的处理方法基本相同。只是需要注意，指针数组的数组元素是指针类型的，对其元素所赋的值必须是地址值。

定义形式：类型说明符 *数组名[长度]＝｛初值｝；

功能是定义一个指针数组：数组的每个元素都是一个指针，指针类型由“类型说明符”指定。元素个数由“长度”指定，还可以在定义的同时给指针数组元素赋初值。

例子：int *p[10];

定义：指针数组p有10个元素，每个元素都是一个可　指向整型数据的指针变量。

说明：和一般的数组定义一样，数组名p也可作为　p[0]的地址。

注意：在"*"与数组名之外不能加上圆括号，否则变成指向数组的指针变量。

如：　int (*q)[10];是定义指向由10个整型元素组成的数组的指针。

引入指针数组的主要目的是便于统一管理同类的指针。

例如，利用指针数组能实现对一组独立的变量以数组的形式对它们作统一处理。如果有以下定义：

```
int a=10, b=20, c=30,k;
int *p[3] = {&a, &b, &c};
下面的循环语句能顺序访问独立的变量a、b、c
for(k = 0; k < 3; k++)
    printf("%d\t", *p[k])
/* 其中*p[k]可写成**(p+k) */
```

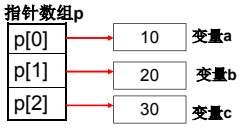


图6.15　指针数组元素指向普通int型变量

指针数组的引用

指针数组元素的引用方法和普通数组元素的引用方法完全相同，可以利用它来引用所指向的普通变量或数组元素，可以对其赋值，也可以参加运算。

(1)引用指针数组元素的格式：

指针数组名[下标]

(2) 引用指针数组元素所指向的普通变量或数组元素的格式：

*指针数组名[下标]

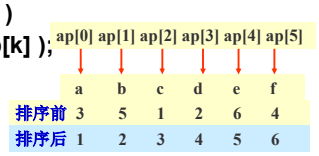
(3)对指针数组元素进行赋值的格式：

指针数组名[下标]＝地址表达式;

【例6.12】 排序时交换变量值

```
#include  <stdio.h>
#define  N sizeof ap/sizeof ap[0]
  /* 整个数组占用的字节数÷一个元素占用的字节数得到该数组有多少个元素 */
int a, b, c, d, e, f;
int main()
{ int *ap[ ] = { &a, &b, &c, &d, &e, &f };
  int k, j, t;
  printf ( "Enter a, b, c, d, e, f.\n" );
  for ( k = 0; k < N; k++ )
     scanf( "%d",ap[k] );//也可写为 *(ap+k)
```

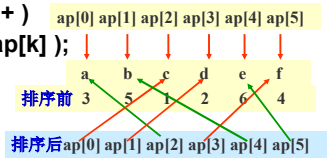
```
for ( k = 1; k < N; k++ )
   for ( j = 0; j < N-k; j++ )
      if ( *ap[j] > *ap[j+1] )
        { t = *ap[j];     /* 交换变量的值 */
          *ap[j] = *ap[j+1];
          *ap[j+1] = t;
        }
for ( k = 0; k < N; k++ )
   printf ( "%d\t", *ap[k] );
printf("\n");
return 0;
}
```



【例6.13】排序时不交换变量的值，而是交换它们的指针

```
#include  <stdio.h>
#define  N sizeof ap/sizeof ap[0]
int a, b, c, d, e, f;
int main()
{  int *ap[ ] = { &a, &b, &c, &d, &e, &f };
   int k, j, *t;
   printf ( "Enter a, b, c, d, e, f.\n");
   for ( k = 0; k < N; k++ )
      scanf ( "%d", ap[k] );
```

```
for ( k = 1; k < N; k++ )
   for ( j = 0; j < N-k; j++ )
      if ( *ap[j] > *ap[j+1] )
        {  t = ap[j];      /* 交换变量的指针 */
           ap[j] = ap[j+1];
           ap[j+1] = t;
        }
for ( k = 0; k < N; k++ )
   printf ( "%d\t", *ap[k] );
printf( "\n" );
return 0;
}
```



当指针数组的元素分别指向两维数组各行首元素时，也可用指针数组引用两维数组的元素。以下代码说明指针数组引用两维数组元素的方法。设有以下代码：

```
int a[10][20], i;
int *b[10];
for(i = 0; i < 10; i++) /* b[i]指向数组元素a[i][0] */
    b[i] = &a[i][0];
```

则表达式a[i][j]与表达式b[i][j]引用同一个元素。从指针数组来看，因为b[i]指向元素a[i][0]，*(b[i]+j)或b[i][j]就引用元素a[i][j]。

当指针数组的元素指向不同的一维持数组的元素时，也可通过指针数组，把它们当作两维数组来使用。

当指针数组的元素指向不同的一维数组的元素时，也可通过指针数组，把它们当作两维数组那样来引用。

如以下代码所示：

```
char w0[ ] = "Sunday",
     w1[ ] = "Monday",
     w2[ ] = "Tuesday",
     w3[ ] = "Wednesday",
     w4[ ] = "Thursday",
     w5[ ] = "Friday",
     w6[ ] = "Saturday";
char *wName[] = {w0,w1, w2, w3, w4, w5, w6};
```

则语句 for(i = 0; i <= 6; i++)
printf("%s\n", wName[i]);
输出星期英文名称。wName[2][4]引用字符w2[4]，其值为’d’。

P145

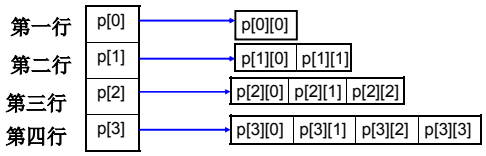
【例6. 14】采用指针数组编程输出二项式系数三角形. 程序把一维数组分割成不等长的段，从指针数组方向来看，把它当作两维数组来处理。

要求程序产生如下形式的二项式的系数三角形：

```
                1                    第0 行
              1   1                  第1 行
            1   2   1                第2 行
          1   3   3   1              第3 行
        1   4   6   4   1            第4 行
      1   5  10  10   5   1          第5 行
    1   6  15  20  15   6   1        第6行
  1   7  21  35  35  21   7   1      第7 行
```

题解分析：

1）使用指针数组，产生元素递增的行需要产生 8 行数据(第0行到第7 行)，第i 行有i+1 个数据。因此可采用指针数组(定义为*p[8])，共8 行，每行含有i+1 个元素。例如:



2）元素中数据的产生法则

在第 0 行，p[0][0] = 1。

在其他第 i 行(i=1 到7)，第0 个和第i-1 个值为1，从j=1 到i，第j 个数据等于第i-1 行的第j-1个数据和第j 个数据的和。即有

p[i][0] = 1; p[i][i] = 1;

p[i][j] = p[i-1][j-1] + p[i-1][j]

例如，在第 2 行和第3 行中，用上述公式，可以得到:

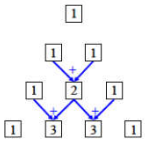
p[2][0] = p[2][2]= 1，

p[2][1] = p[1][0] + p[1][1] = 2

p[3][0] = p[3][3]=1，

p[3][1] = p[2][0] + p[2][1] = 3

p[3][2] = p[2][1] + p[2][2] = 3



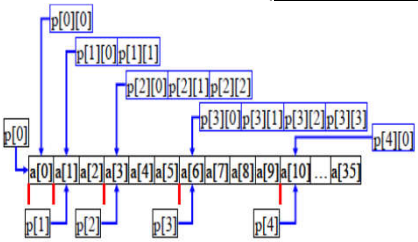
考虑 8 行数据，共需元素个数 8*(8+1)/2＝36。因而定义数组p[36]。

更一般地，如果需要N 行，则共需元素个数=N*(N+1)/2。

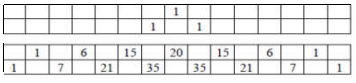
以下依此类推

```
#include <stdio.h>
#define  N  8
int a[N*(N+1)/2], i, j, *p[N];
void main()
{   for ( p[0] = a, i = 1; i < N; i++ )
        p[i] = p[i-1] + i;
```

i	p[i] = p[i-1] + i	p[i]长度
0	p[0] = a = &a[0]	1-0=1
1	p[1] = p[0] + 1 = &a[1]	3-1=2
2	p[2] = p[1] + 2 = &a[3]	6-3=3
3	p[3] = p[2] + 3 = &a[6]	10-6=4
	.....	.....
7	p[7] = p[6] + 7 = &a[28]	36-28=8




```
    for ( i = 0; i < N; i++ )
    {   p[i][0] = p[i][i] = 1;
        for(j = 1; j < i; j++) p[i][j] = p[i-1][j-1] + p[i-1][j];
    }
    for(i = 0; i < N; i++)
    {   printf ( "%*c", 20-2*i, ' ' );
        for (j = 0; j <= i; j++ ) printf ( "%4d", p[i][j] );
        printf ( "\n" );
    }
    printf ( "\n" );
}
```



考察输出的形式，每个数据最多为 2 位数，如果在前后各留一个空格，共4 个字符的位置。也就是说，将以%4d 输出。而在相邻的奇数行和偶数行有2 格的交叠，则第0 行的p[0][0]=1，与第7行的p[7][0]=1，相距7*2=14 格。则在第i 行的左边应该留出14-i*2 格，即则每行开始时，先输出 14-i*2 个空格(多留几格为好)。

6.7 多级指针

- 若某指针变量中存放的不是(普通)变量的地址，而是另一个指针变量的地址，则该指针变量便被称为指向指针变量的指针变量，可以简称为"指向指针的指针"。
- 在有些C语言教材中，"指向指针的指针"常被称为"二级指针"，相应的，指向(普通)变量的指针则被称为"一级指针"。

二级指针变量的定义（及赋初值）格式为：

类型说明符 **指针变量名[=初值];

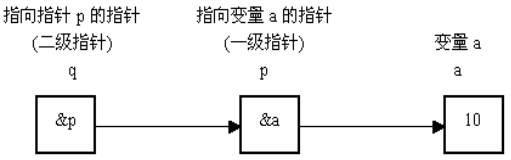
定义二级指针变量时，应注意：

(1) 二级指针变量定义时前面必须有两个"*"号，指向的变量的数据类型由"类型说明符"确定。

(2) 在定义的同时可以赋初值。初值必须是一级指针变量的地址，通常形式为"&一级指针名"。

例如： short a, *p=&a, **q=&p;

定义了一个整型变量a，一个一级指针p和一个二级指针q，通过赋初值使p指向了a，q指向了p。可通过下面的图来描述这种关系



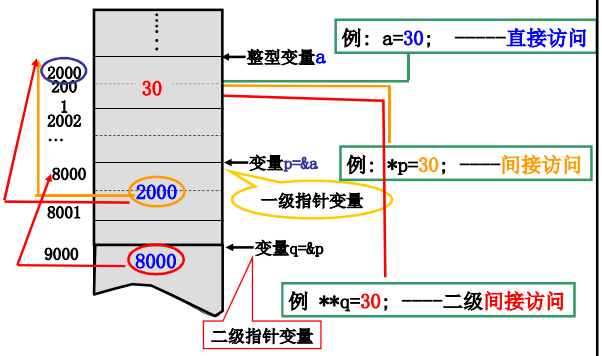
由图可知：变量、一级指针与二级指针之间存在以下关系：

(1) *二级指针变量：代表所指向的一级指针变量。如：*q就代表p。

(2) **二级指针变量：代表它所指一级指针变量所指向的变量。如：**q代表a。

(3) *一级指针变量：代表它所指向的变量。如：*p代表a。

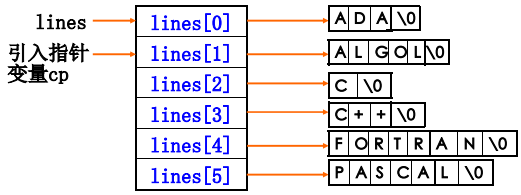
图示指向指针的指针



多级指针往往与指针数组有密切的关系。

例如：char *lines[ ] = { "ADA", "ALGOL", "C", "C++", "FORTRAN", "PASCAL" };

则：lines指针数组的每个元素分别指向每个字符串常量的首字符地址。数组名lines是首元素lines[0]的指针，lines+k是元素lines[k]的指针。



则： cp = &lines[k]。

因此：cp 就是指向字符指针的指针变量。

定义方法： char **cp;

说明：cp前面有两个*号。由于*自右向左结合，首先是 *cp 表示 cp 是指针变量，再有 **cp 表示 cp 是指向一个字符指针变量。

例如：cp = &lines[1];

则： *cp 表示引用 lines[1] (lines[1]也是一个指针)，指向字符串"ALGOL"的首字符。

**cp表示引用lines[1][0]，其值是字符' A'

顺序输出指针数组lines各元素所指字符串：

```
for ( cp = lines; cp < lines+6; cp++ )
        printf ( "%s\n", *cp );
```

采用"%c"格式，逐一输出指针数组lines各元素所指的字符串：

```
int  i, j;
for ( i = 0; i < 6; i++ )
{
    for ( j = 0; lines[i][j] != '\0'; j++ )
          printf ( "%c", lines[i][j] );
    printf ("\n");
}
```

设有数组a[]和指针数组pt[]有以下代码的关系：

```
int a[ ] = { 2, 4, 6, 8, 10};
int *pt[ ]={&a[3],&a[2],&a[4],&a[0], &a[1]};
int **p;
```

利用指针数组pt[]和指针的指针p，遍历数组a[]：

```
for( p = pt; p < pt + 5; p++ )
   printf ( "%d\t", **p );
```

指向指针数组元素的指针即为指针的指针，如以上程序中的指针变量p。*p能引用p所指的数组元素，**p能引用p所指数组元素所指的变量。程序中用**p访问数组a[]的元素。

还可以有更多级指针

例　三级指针　int　***p;

　　四级指针　char　****p;

更多级指针的定义与引用跟二级指针相似，由于在实际应用中很少使用，这里不作介绍。有兴趣可以自己编写相应程序进行验证。

6.8 函数指针

通过前面的学习，我们已经知道，C语言中的指针，既可以指向（普通）变量（如整型、字符型、实型等），也可以指向数组元素或数组。下面我们还要介绍一种指向函数的指针变量——函数指针,该指针实际上只不过是专门用于指向"函数"的.

可变函数调用——函数指针的由来

考虑这样一种情况，假定已经设计了几个函数，但是每次可能需要根据实际情况来调用其中的一个。或者说，需要调用的函数是可变的。可以用以下描述来表达：

if（a）
调用函数 s1（）
else
调用函数 s2（）

希望有一个函数指针 fp，可将以上表达用下面的形式实现：

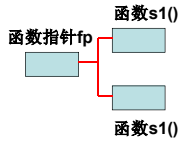
定义函数 s1（）和s2（）和函数指针fp

if（a）fp指向s1（）

else　fp指向s2（）

用调用函数：fp（）

分别调用s1（）和 s2（）



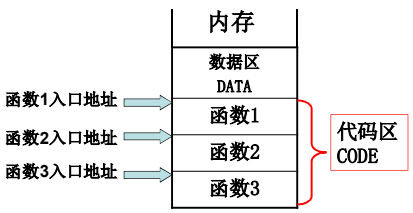
指针与函数的结合使编程更为灵活。

- 指针可以用作函数的参数(传递地址值)，
- 函数也可以返回一个指针类型的值，
- 甚至可以定义指向函数的指针。

●函数指针的作用

在程序编译时，C语言约定，编译系统会给函数分配一段内存单元，这段内存的起始地址就是函数的入口地址（函数中第一条指令的地址）。如果一个指针变量的值等于函数的入口地址，则该指针变量称为指向函数的指针变量，简称为函数指针。

因此，如果设定函数指针在不同的情况下指向不同的函数，就可以根据函数指针指向函数的入口地址来调用函数，以实现根据实际情况来调用不同的函数。



函数指针的含义和功能

1）可以用函数指针定义一个指针变量（函数指针变量），也可以定义一个指针数组（函数指针数组）。
2）函数指针可以出现在形参和实参中。
3）函数指针的指向可以在一个主调函数中确定，然后作为实参传递给被调函数，使得被调函数根据实参和形参的对应关系来调用不同的函数。
4）函数指针的指向可以在一个被调函数中确定，并且通过返回值传递给主调函数。

和其它任何变量一样，在使用指向函数的指针变量之前需要先定义该指针变量，指向函数的指针变量（即函数指针）的一般定义形式为：

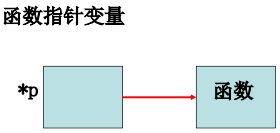
函数指针的定义

函数返回值类型 (*指针变量名)(形参类型); 或者

函数返回值类型 (* 指针数组名[常量表达式])(形参类型表);

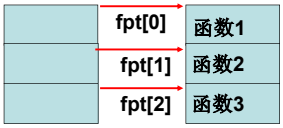
可以看出，除函数名用（*指针变量名）代替外，函数指针的定义形式与函数的原型相同，在函数指针定义中加入形参类型是现代程序设计风格。
如：int (*p) (int, int);形参类型一般不要省略。
若没有参数,请不要忘了括号.
如： int (*p) ( );

例如： int (*p)(int , char *);
表示 p 是一个指向函数的指针变量，所指向函数的返回值类型是int 型的，并有两个形参，分别为int 和char * 类型。



又如： char *(*fpt[3])(void);
表示 fpt 是一个能够指向函数的指针数组，所指向函数的返回值类型是char *型的，函数没有形参，函数指针数组元素的值分别为3 个函数的地址。

函数指针数组fpt[ ]

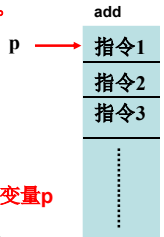


用函数指针变量调用函数

在定义指向函数的指针变量时，指针变量到底指向哪个函数并没有指明，它可以根据所赋的地址不同而指向不同的函数。一旦定义了指针变量指向某个具体的函数之后，就可以通过该指针变量来调用此函数。

一般函数调用形式：函数名(实参表)
用函数指针变量调用形式：
(*函数指针变量名) (实参表)

利用函数指针调用函数，求a和b的和。




```
#include <stdio.h>
int add(int x, int y)/*定义函数add*/
{   return x+y;  }

void main( )
{
   int (*p)(int, int);//定义一个指向函数的指针变量p
   int a,b,sum;
   p = add;      /*使函数指针p指向函数add */
   scanf("%d,%d", &a, &b);
   sum = (*p)(a,b);/*通过函数指针调用函数add，相当
                于sum=add(a,b)*/
   printf("a=%d,b=%d,a+b=%d\n",a,b,sum);
}
```

说明：

(1) 语句p=add，把函数add的入口地址赋给函数指针p，则p指向函数add，因此，*p就代表函数add。

(2) 在给函数指针变量赋值时，只需给出函数名而不必给出参数，如: p=add;
因为将函数入口地址赋给p时，并不牵涉到实参与形参的结合问题，所以不能写成
p=add(a,b);

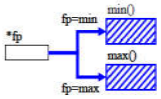
(3) 函数的调用可以通过函数名调用，也可以通过函数指针调用，如例中的
语句： sum=(*p)(a,b);
相当于调用语句： sum=add(a,b);

(4) (*p)( )表示一个指向函数的指针变量，它可以先后指向不同的函数。

(5) 指向函数的指针变量p，只能指向函数的入口地址，而不可能指向函数中某一条指令处的地址，因此象p++、p--、p+n等运算是无意义的。

(6) 注意int (*p)( )和int *p( )的区别：在定义int (*p)( )中，p先与*结合，p是指针变量，然后与( )结合，表示此指针变量指向函数。如果定义为 int *p( )，由于( )比*优先级别要高，则p先与( )结合，p是函数，然后与*结合，表示函数的返回值是一个指针值。(后面介绍)

【例6.15】使用函数指针变量调用函数的示意程序。




```
#include <stdio.h>
int main()
{  int (*fp)(int, int), x, y, z;
   int min(int, int), max(int, int);
   printf("Enter x, y: ");scanf("%d%d",&x,&y);
   fp = min;   /* 让fp指向函数min() */
   z = (*fp)(x, y);  /* 调用fp所指函数 */
   printf("MIN(%d,%d) = %d\n", x, y, z);
   fp = max;/*现在更改fp，使它指向函数max() */
   z = (*fp)(x, y); /* 调用fp所指函数 */
   printf("MAX(%d,%d) = %d\n", x, y, z);
   return 0;
}
```

```
int min(int a, int b)
{
     return a < b ? a : b;
 }

int max(int a, int b)
 {
      return a > b ? a : b;
 }
```

函数指针作函数的形参

前面已经介绍，定义指向函数的指针变量时，并没有指明它到底指向哪个函数，而是在随后的程序运行过程中，根据所赋的地址不同而指向不同的函数。基于这一特点，我们可以把函数指针用作某一函数的形式参数，从而使该函数能根据实际情况调用不同的函数，大大增强该函数的功能。

将函数名或指针作为实参传递给被调用的函数，要求形参是一个函数指针形参。

用这种方式来使用函数指针的基本程序结构为：

●事先定义若干个函数

在主调函数中：

●定义函数指针

●决定函数指针指向某个函数

●将函数指针作为实参，调用被调函数

在被调函数中：

●根据虚实结合，获得函数指针值

●调用函数指针指向的函数

例综合练习： 模拟计算器中的加减乘除运算。

事先定义若干个函数

```
#include <stdio.h>
int add(int x,int y) /*定义实现加法的函数add*/
{   return x+y; }

int sub(int x,int y) /*定义实现减法的函数sub*/
{  return x-y; }

int mul(int x,int y) /*定义实现乘法的函数mul*/
{    return x*y; }

int div(int x,int y) /*定义实现整除的函数div*/
{   return x/y; }
```

```
int compute(int x,int y, int (*p)(int ,int))
{     /*函数指针p作为compute函数的参数*/
      int n;
      n=(*p)(x,y);/*通过函数指针变量p调用函数*/
       return n;
}
void main( )
{     int a, b,result;
      char op;
      printf("请连续输入操作数a，运算符op和操作数b：\n");
      scanf("%d%c%d",&a,&op,&b);
      switch (op)
      {        case '+': result=compute(a, b, add);break;
               case '-': result=compute(a, b, sub);break;
               case '*': result=compute(a, b, mul);break;
               case '/': result=compute(a, b, div);break;
      }
      printf("计算的结果为：%d%c%d=%d\n",a,op,b,result);
}
```

在上面的主程序中，switch语句根据运算符的不同决定将某函数的入口地址（add或sub或mul或div）传递给compute函数中的形式参数（函数指针p），从而使compute函数能够进行加、减、乘、除等多种运算。

例如在执行函数调用语句：compute(a,b,add);时，实参a的值传递给形参x，实参b的值传递给形参y，实参add（add函数的入口地址）传递给形参p（函数指针），然后在函数体中执行(*p)(x, y)就相当于执行add (x , y)。

函数指针变量的用途之一是把指针作为参数传递到其他函数。这是c语言应用的一个比较深入的部分。这里只作简单的介绍。

【例6.16】对给定的实数表，求它的最大值、最小值和平均值。

程序有三个函数max()、min()和ave()，另设一个函数afun()。主函数调用函数afun()，并提供数组名、数组元素个数和求值函数指针作为实参。由函数afun() 根据主函数提供的函数指针实参来调用相对应的函数。

```
#include  <stdio.h>
#define  N sizeof a / sizeof a[0]
double max(double a[ ], int n)
{ int i;  double rmax;
  for (rmax = a[0], i = 1; i < n; i++)
      if (rmax < a[i]) rmax = a[i];
  return rmax;
}
double min(double a[ ], int n)
{ int i;  double rmin;
  for (rmin = a[0], i = 1; i < n; i++)
     if (rmin > a[i]) rmin = a[i];
  return rmin;
}
```

```
double ave (double a[ ], int n)
{ int i;  double rave;
  for (rave = 0.0, i = 0; i < n; i++)
      rave += a[i];
  return rave/n;
}
double afun (double a[ ], int n,
             double (*f)(double *, int))
{
   return (*f)(a, n);
}
```

```
int main()
{ double a[] =
    {1.0, 2.0, 3.0, 4.0, 5.0,
     6.0, 7.0, 8.0, 9.0};
  printf("\n结果是:\n  ");
  printf("最大值 = %f", afun(a, N, max));
  printf("   最小值 = %f", afun(a, N, min));
  printf("   平均值 = %f\n", afun(a, N, ave));
  return 0;
}
```

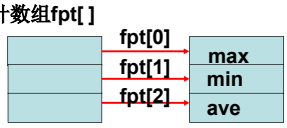
函数指针数组

利用函数指针能存储函数名的性质，可将若干函数指针存于一数组中，调用时可选择不同的函数。

例如：double(*fpt[ ])(double *,int)={max,min, ave};

说明：定义了函数指针数组fpt[]，并对每个数组元素初始化。如：将函数max()、min()和ave()的函数指针填写在函数指针数组fpt[]中，使数组fpt[]成为函数的入口表。

函数指针数组fpt[ ]



【例6.17】这里的程序是上例的改写。程序的主函数从fpt[]中取出函数指针，并以此函数指针调用它所指向的函数。实现上述程序同样功能的主函数可修改成如下形式，而其余函数均与上例程序中的相同。

```
#include  <stdio.h>
#define  N   sizeof a / sizeof a[0]
double max(double *, int), min(double *, int),
ave(double *, int);
int main()
{  double a[] ={1.0, 2.0, 3.0, 4.0, 5.0,
                6.0, 7.0, 8.0, 9.0};
   double  (*fpt[])(double *, int) = { max, min , ave};
/* 定义函数指针数组 */
```

```
char *title[] = {"最大值", "最小值", "平均值"};
char *menuName[] ={"求最大值","求最小值","求平均值",""};
int ans, k;
  while (1) {   printf("请选择以下菜单命令。\n");
    for(k = 0; menuName[k][0] != '\0' ; k++)
        printf("\t%d:%s\n", k+1, menuName[k]);
    printf("\t其它选择结束程序运行。\n");
    scanf("%d", &ans);
    if (ans < 1 || ans > k)  break;
    printf("\n结果是：\t%s = %f\n",
           title[ans-1], (*fpt[ans-1])(a, N));
  }
  return 0;
}
```

6.9 返回指针值的函数

一个函数可以返回一个int型、float型、char型的数据，也可以返回一个指针类型的数据（即地址）。

返回变量指针的函数的定义格式如下：

类型说明符 *函数名（[形参表]）

例如：

int *max(int x, int y)

说明：max是函数名，其返回值为指针类型，且是指向int型数据的指针，x和y是两个整型形参。

【例6.18】编制在给定的字符串中找特定字符的第一次出现。如果找到，返回找到的字符的指针；反之，如果没有找到，则返回NULL值。

设函数为searchCh()，函数应设两个形参，指向字符串首字符的指针和待寻找的字符。查找过程是一个循环，函数从首字符开始，在当前字符还不是字符串结束符，并且当前字符不是要查找字符情况，继续考察下一个字符。待查找循环结束，如果当前字符不是字符串结束符，则找到，返回当前字符指针；否则，就是没有找到，函数返回NULL。

```
char *searchCh(char *s, char c)
{
    while (*s && *s != c)s++;
    return *s ? s : NULL;
}
```

+返回函数指针的函数

返回函数指针的函数的定义或说明的一般形式为：

类型说明符 (*函数名(形参表))(形参类型表);

首先，函数名标识符与后面的圆括号结合，说明该标识符是函数名，圆括号中列出的是这个函数的形参说明。接着与前面的星号结合，函数返回指针。再与最后的圆括号结合，说明返回的是函数指针，圆括号中的内容是指针所指函数的形参说明。最后，最前面的是指针所指函数的返回值的类型。下面的程序例子说明返回函数指针函数的用法。

其中代码

double (*menu(char **))(double*, int)

首先是menu(char **)，标识符menu同圆括号结合，说明menu是函数名，有一个字符指针数组形参。随后是(*menu(char **))，函数同*结合，说明函数返回指针。接着是(*menu(char **))(double*, int)，与最后面的圆括号结合，表示返回的是指向函数的指针，所指向的函数有double*类型和int类型两个形参。最后是

double (*menu(char **))(double*,int)，

说明指向的函数的返回值类型是double类型。

函数menu()是一个菜单函数，它接受用户选择，返回相应处理函数的指针。主函数调用menu()函数，并利用函数menu()返回的函数指针调用相应的处理函数。

【例6.19】函数返回函数指针的示意程序

```
#include <stdio.h>
#define N sizeof a / sizeof a[0]
double max(double *, int), min(double *, int), ave(double *, int);
//其中函数max()、min()和ave()与前面的例子相同。
double (*fpt[])(double *, int) = { max, min , ave, NULL}; /* 函数指针数组 */
char *title[] = {"最大值", "最小值", "平均值"};
char *menuName[] ={"求最大值","求最小值","求平均值",""};
```

```
double a[] ={1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0};
double (*menu(char **titptr))( double *, int)
/* 函数返回函数的指针 */
{ int ans, k;
  printf("请选择以下菜单命令。\n");
  for(k = 0; menuName[k][0] != '\0'; k++)
     printf("\t%d : %s\n", k+1, menuName[k]);
  printf("\t其它选择结束程序运行。\n");
  scanf("%d", &ans);
  if (ans < 1 || ans > 3) return NULL;
  *titptr = title[ans-1];//函数带回功能注释字符串
  return fpt[ans-1];  // 返回函数指针
}
```

```
int main ()
{   double (*fp)( double *, int);
    char *titstr;
    while (1)
    {
        if ((fp = menu(&titstr)) == NULL)
            break;
        printf("\n结果是：%s = %f\n",
            titstr, (*fp)(a, N));
    }
  return 0;
}
```

本章重点概念的复习

- 1．指针和指针变量

指针是一种数据类型，指针的值一般指某个变量的地址。从某种意义上说，指针就是地址。

而指针变量是指用来存放指针（即其它变量的地址）的变量。

- 2．&和*运算符

“&”为取地址运算符。如&x表示变量x的地址。

”*”为取内容运算符。如*x表示指针变量x所指向的内容。

● 3．指向一维数组元素或一维数组的指针、指针数组

当用一个指针p指向一维数组的某个元素后，就可以通过指针变量名(*p)来访问该数组元素，还可以通过移动指针来访问该一维数组中的其它元素。指针的移动是以数组元素为单位的。它属于“一级指针”，其定义形如：int a[5],*p=&a[2];

而指向一维数组的指针是指向一个由m个元素组成的一维数组。指针的移动不以数组元素为单位，而是以一维数组为单位。这种指针一般初始化指向二维数组(n*m)的某一行。它属于“二级指针”，其定义形如：int a[3][4];

int (*p)[4]=&a[0]；或int (*p)[4]=a;

指针数组跟前两种不同，它是一种“数组”，只是数组元素是“指针”而已。 其定义形如：

int a,b,c，*p[3]={&a,&b,&c};

注意定义形式上的区别，*p[3]只比(*p)[3]少了个括号，但含义却完全不同。

● 4．返回指针值的函数（指针函数）和函数指针

返回指针值的函数

函数也可以返回指针值,可以是某变量的指针,或是某函数的指针。其定义形如：

int *max(int x, int y)；

函数指针实际上是一个“指针”，只不过是该指针是专门用于指向“函数”的，其定义形如：

int (*p) (int, int)；

在定义时，指针变量p到底指向哪个函数并没有指明，它可以根据所赋的地址不同而指向不同的函数。

有关指针的小结

定义	含　义
int i;	定义整型变量i
int *p	p为指向整型数据的指针变量
int a[n];	定义整型数组a，它有n个元素
int *p[n];	定义指针数组p，它由n个指向整型数据的指针元素组成
int (*p)[n];	p为指向含n个元素的一维数组的指针变量
int f();	f为带回整型函数值的函数
int *p();	p为带回一个指针的函数，该指针指向整型数据
int (*p)();	p为指向函数的指针，该函数返回一个整型值
int **p;	P是一个指针变量，它指向一个指向整型数据的指针变量

其中:n为常量或符号常量

观察下面的输出结果,体会指针的用法

```
#include <stdio.h>
void main()
{
   char **appt;
   char *a[]={"ABCD","UVW","XYZ"};
   appt=a;
   printf("%d\n",*appt);
   printf("%c\n",**appt);
   printf("%d\n",*(appt+1));
   printf("%c\n",*(*(appt+2)+2));
}
```