

计算机程序设计基础(1)

--- C语言程序设计(10A)

孙甲松

sunjiasong@tsinghua.edu.cn

电子工程系 信息认知与智能系统研究所

罗姆楼6-104

电话: 13901216180/62796193

2021.11.

第10章 指针

10.1 指针变量

10.1.1 指针的基本概念

10.1.2 指针变量的定义与引用

10.1.3 指针变量作为函数参数

10.1.4 指向指针的指针

10.2 指针数组

10.3 数组与指针

10.3.1 一维数组与指针

10.3.2 二维数组与指针

10.3.3 数组指针作为函数参数

10.4 动态内存的申请与释放

10.4.1 malloc()函数

10.4.2 calloc()函数

10.4.3 realloc()函数

10.4.4 free()函数

10.5 字符串与指针

10.5.1 字符串指针

10.5.2 字符串指针作为函数参数

10.5.3 strstr()函数

10.6 函数与指针

10.6.1 用函数指针变量调用函数

10.6.2 函数指针数组

10.6.3 函数指针变量作为函数参数

10.6.4 返回指针值的函数

10.7 main函数的形参

10.8 变步长梯形求积法

10.1 指针变量

10.1.1 指针的基本概念

● 指针（**Pointer**），在C语言中,可以定义一种称为指针类型的变量，这种变量是专门用以存放其他变量所占存储空间的首地址。在程序执行过程中，当要存取一个变量值时，可以首先从存放变量地址的指针变量中取得该变量的存储地址，然后再从该地址中存取该变量值。

● 在C语言中对内存数据（如变量、数组元素等）的存取有两种方法：

① 直接存取

指在程序执行过程中需要存取变量值时，直接用变量名存取变量所占内存单元中的内容。

例如,下列两个语句所构成的程序段中：

```
int x=3, y;
```

```
y=2*x+3;
```

② 间接存取

指为了要存取一个变量值，首先从存放变量地址的指针变量中取得该变量的存储地址，然后再从该地址中存取该变量值。简单地说，在这种存取方式中，存取变量的值是通过存取指针变量所“指向”的内存单元中的内容。在此，所谓“指向”，是通过地址来体现的。

- 一个变量的地址称为该变量的“指针”。如 $\&x$ 值称为变量 x 的指针。由于一个变量要占多个字节的内存单元，因此，所谓变量的地址一般均指首地址（即变量所占内存单元中第一个字节的地址）。

- 专门用于存放其他变量地址的变量称为指针变量。例如，`int *s, x, y;` 其中 s 为指针变量，它既可以存放整型变量 x 的地址，也可以存放整型变量 y 的地址。总之， s 可以存放任意整型变量的地址。

$s = \&x;$ 或 $s = \&y;$ 但不能写为： $*s = \&x;$ 或 $*s = \&y;$;

由上可知，指针变量的值为地址，普通变量的值为数值。

* 为指针运算符

10.1.2 指针变量的定义与引用

- 变量的指针就是变量的地址，即指针变量用于存放变量的地址（即指向变量）。定义指针变量的一般形式为：

类型标识符 *指针变量名；

- 在程序中定义了指针变量后，就可以用取地址运算符“&”将同类型变量的地址赋给它，然后就可以间接存取该同类型变量的值。

在定义指针变量时要注意以下几点：

1) 指针变量名前的“*”只表示该变量为指针变量，以便区别于普通变量的定义，而指针变量名不包含该“*”。

例如，在说明语句 `int x,*s;`中说明了s是一个指针变量，但不能说*s是指针变量。s的类型是`int *`，而x的类型是`int`。

在表达式中,变量前加“*”表示间接存取

赋值语句 `*s=3;`表示将整数3赋给由指针变量s所指的变量的内存中

● 各种指针的定义

例如: `char *p;`
`int *a[10];`
`float **r;`
`short (*s)[100];`
`long *f();`
`void (*g)();`
`double (*h[10])();`

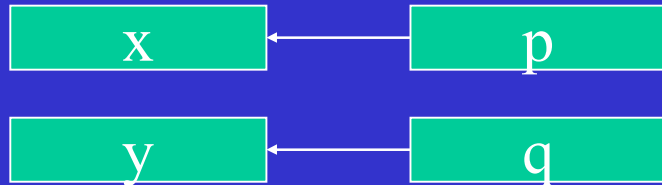
在32位编译器上:

- `sizeof(p) = 4`
- `sizeof(r) = 4`
- `sizeof(s) = 4`
- `sizeof(g) = 4`
- `sizeof(a) = 40`
- `sizeof(h) = 40`

● 一般来说, 定义变量时, 标识符前加 ‘*’, 即为指针的定义, 标识符后再紧跟[]为指针数组, 标识符后再紧跟()为指针函数或函数指针。

● 任何类型的指针大小都一样, 32位编译器为4字节, 64位编译器为8字节, 都是用来存放计算机内存地址。

【例10-1】 设有下列C程序：



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

```
void main( )
```

```
{ double x=0.11,y=0.1;
```

```
    double *p,*q;    /*定义双精度实型指针变量p与q*/
```

```
    p= &x; q= &y;    /*p指向x, q指向y*/
```

```
    printf("&x=%u,&y=%u\n", &x, &y); /*输出变量x与y的地址*/
```

```
    printf("p=%u,q=%u\n", p, q);/*输出指针变量p与q中存放的地址*/
```

```
    printf("x=%f,y=%f\n", x, y);    /*输出变量x与y的值*/
```

```
    printf("*p=%f,*q=%f\n", *p, *q); /*输出指针变量p与q所指向的变量值*/
```

```
}
```

&x=1012727032,&y=1012727024

p=1012727032,q=1012727024

x=0.110000,y=0.100000

***p=0.110000,*q=0.100000**

请按任意键继续...

说明：此结果中指针值在不同计算机或不同时刻运行结果可能都会不一样

2) 一个指针变量只能指向与之同类型的变量。
不同类型的变量所占的字节数是不同的。

【例10-2】 设有下列C程序：

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

```
void main( )
```

```
{ double x=0.1;      /*定义双精度实型变量x,并赋初值为0.1*/
```

```
  int *p;            /*定义整型指针变量p*/
```

```
  p=&x;              /*整型变量指针p指向双精度实型变量x*/
```

```
  printf("x=%f\n",x);    /*输出双精度实型变量x的值*/
```

```
  printf("*p=%f\n",*p);
```

```
                      /*按实型格式输出整型指针变量p所指向的变量值*/
```

```
  printf("*p=%d\n",*p);
```

```
                      /*按整型格式输出整型指针变量p所指向的变量值*/
```

```
}
```


在编译上述程序时,会显示如下警告信息:

10-02.c(5) : warning C4133: “=”: 从“double *”到“int *”的类型不兼容
运行后输出结果为:

x=0.100000 /*双精度实型变量x的值*/

*p=0.000000 /*按实型格式输出的整型指针变量p所指向的变量值*/

*p=-1717986918 /*按整型格式输出的整型指针变量p所指向的变量值*/

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值后才能被引用

5) 与一般的变量一样,也可以对指针变量进行初始化

例如, `int x,*p=&x;` 等价于 `int x,*p;` 等价于 `int x=5,*p=&x;`
`*p=5;` `p=&x; *p=5;`

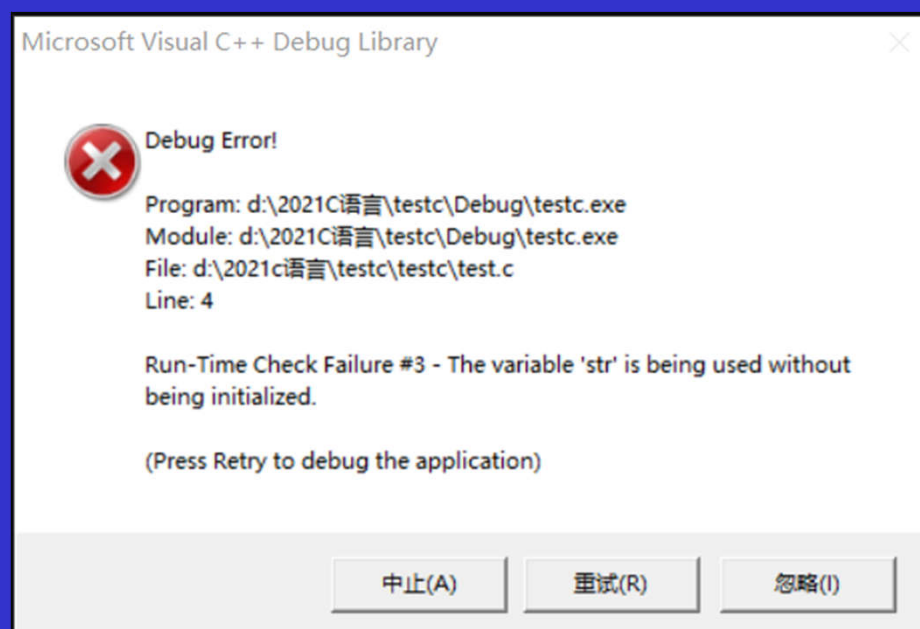
有程序如下:

```
#include <stdio.h>
void main()
{ char *str;
  scanf("%s", str);
  printf("str=%s\n", str);
}
```

编译时出现警告错误:

test.c(4) : warning C4700: 使用了未初始化的局部变量 “str”

运行结果:



注意: 只有当指针变量中具有确定地址值后才能被引用, 否则会产生致命错误!

【例10-3】从键盘输入两个整数赋给变量a与b，不改变a与b的值，要求按先小后大的顺序输出。其C程序如下：

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

```
void main( )
```

```
{ int a,b,*p,*p1=&a,*p2=&b;
```

```
scanf("%d%d", &a, &b);
```

```
if (a>b) { p=p1; p1=p2; p2=p; }
```

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

```
printf("min=%d,max=%d\n",*p1,*p2);
```

```
}
```



scanf("%d%d", p1, p2);

if (*p1 > *p2)
{ p=p1; p1=p2; p2=p; }

将p1和p2指针的值交换， a和b的值保持不变。

上述程序的运行结果为（有下划线的为键盘输入）

45 23

a=45,b=23

min=23,max=45

【例10-3】 C程序若改为:

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

```
void main( )
```

```
{ int a,b, p,*p1=&a,*p2=&b;
```

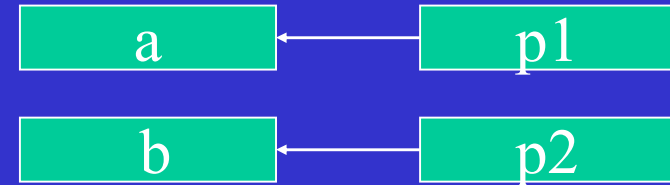
```
scanf("%d%d", p1, p2);
```

```
if (a>b) { p=*p1; *p1= *p2; *p2=p; }
```

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

```
printf("min=%d,max=%d\n",*p1,*p2);
```

```
}
```



将p1和p2指针所指单元的内容交换，因此a和b的值被交换了。

上述程序的运行结果为（有下划线的为键盘输入）

45 23

a=23,b=45

min=23,max=45

注意: *r++ 和 (*r)++ 的区别

例如: `int a[5]={1,2,3,4,5}, *r=a, n, m;`

`n = *r++; m = (*r)++;`

● `n = *r++` 是n先取r所指单元的内容, 然后指针r加1 (指向下一个单元)。执行结果: `n=1`, 指针r指向`a[1]`, a数组元素的值保持不变。 `*(r++)` 与 `*r++` 效果相同。

● `m = (*r)++` 是m先取r所指单元的内容, 然后将r指针所指单元的内容加1。执行结果: `m=1`, 指针r仍指向`a[0]`, 但数组元素`a[0]`的值变为2。

如果写为: `n = *++r; m = *(++r);`

● `n = *++r` 是指针r先加1, n取r所指单元`a[1]`的内容。执行结果: `n=2`, 指针r指向`a[1]`, a数组元素的值保持不变。 `*(++r)` 与 `*++r` 效果相同。

如果写为: `n = ++*r; m = ++(*r);`

● `n = ++*r` 先将r指针所指单元的内容先加1, 然后n取r所指单元的内容。执行结果: `n=2`, 指针r仍指向`a[0]`, 但`a[0]`值变为2。 `++(*r)` 与 `++*r` 效果相同。

10.1.3 指针变量作为函数参数

指针变量也可以作为函数参数。利用指针变量作为函数的形参，可以使函数通过指针变量返回指针变量所指向的变量值，从而实现调用函数与被调用函数之间数据的双向传递。

【例10-4】 编写一个函数，计算

$$S(x) = x - \frac{1}{3}x^3 + \frac{1}{5}x^5 - \dots + \frac{(-1)^n}{2n+1}x^{2n+1}, \quad -\infty < x < \infty$$

直到最后一项的绝对值 $\left| \frac{(-1)^n}{2n+1}x^{2n+1} \right| < 0.000001$ 为止，并返回此时的n值。其中x在主函数中从键盘输入。
再次强调，不要用：

$$\text{pow}(-1, n) * \text{pow}(x, 2 * n + 1)$$

来表示： $(-1)^n x^{2n+1}$

T_n 递推式为：

其C程序如下：

$$\begin{aligned} P_0 &= x \\ P_1 &= -x^3 = -P_0 * x^2 \\ P_2 &= x^5 = -P_1 * x^2 \\ P_3 &= -x^7 = -P_2 * x^2 \\ &\dots\dots\dots \\ P_n &= -P_{n-1} * x * x \\ T_n &= P_n / (2n-1) \end{aligned}$$

```
#include <stdio.h>
#include <math.h>
/* 指针pn指向存放多项式项数的地址 */
double arctan(double x, double eps, int *pn)
{
    int n=0;
    double p,t,s;
    p=x; s=x;
    do
    {
        n=n+1; /*项数计数*/
        p= -p*x*x;
        t=p/(2*n+1);
        s=s+t; /*逐项累加多项式中的各项*/
    } while (fabs(t)>=eps); /*不满足精度要求时继续循环计算*/
    *pn=n; /*将满足精度要求时的项数存放到多项式项数的地址中 */
    return s; /*返回满足精度要求的多项式值*/
}
```

```
void main( )  
{ int n;  
  double x, s;  
  printf("input x:");  
  scanf("%lf", &x);  
  s = arctan(x, 0.0001, &n);  
  printf("n=%d\ns=%f\n", n, s);  
}
```

如果调用函数时提高精度到:
`s = arctan(x, 0.000001, &n);`

运行输出结果为:

input x:1.0
n=500000
s=0.785399

运行后输出结果为:

input x: 1.0
n=5000
s=0.785448

提示: 请把教程233页第
2行0.000001改为0.0001

可以看到:

- 通过函数返回值返回了多项式的计算结果给调用它的函数。
- 而同时通过传递n的地址作为参数给被调用函数, 通过指针操作把迭代次数也传递回了调用它的函数。

如果想调用函数交换两个变量值，

swap函数写为：

```
void swap(int p1, int p2)
{ int t;
  t=p1; p1=p2; p2=t;
  return;
}
```

程序的运行结果为（带有下划线的为键盘输入）：

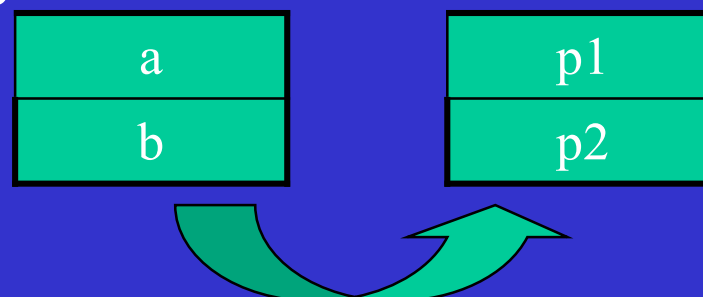
12 34

a=12,b=34

a=12,b=34

主函数写为：

```
#include <stdio.h>
void main( )
{ int a, b;
  scanf("%d%d",&a,&b);
  printf("a=%d,b=%d\n",a, b);
  swap(a, b);
  printf("a=%d,b=%d\n",a, b);
}
```



C语言参数传递是传值，变量的值

函数中将形参p1和p2的值交换，与主函数中a和b的值无关。

【例10-5】利用指针变量实现两个变量值的互换。

其C程序如下：

```
#include <stdio.h>
void swap(int *p1, int *p2)
{ int t;
  t=*p1; *p1=*p2; *p2=t;
  return;
}
```



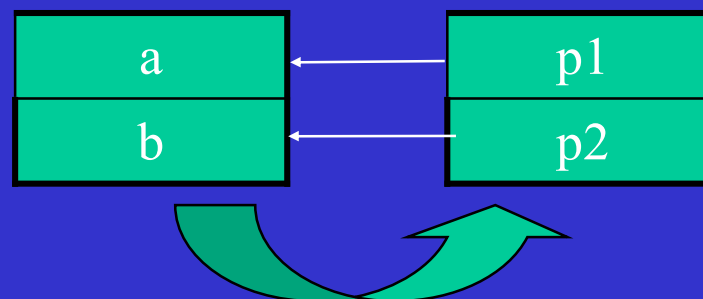
```
void main( )
{ int a, b;
  scanf("%d%d",&a,&b);
  printf("a=%d,b=%d\n",a, b);
  swap(&a, &b);
  printf("a=%d,b=%d\n",a, b);
}
```

程序的运行结果为（带有下划线的为键盘输入）：

12 34

a=12,b=34

a=34,b=12



参数传递是传值，变量的地址值

将p1和p2所指单元的内容交换，因此a和b的值被交换了。

提示：要想让函数修改变量的值，就必须传递变量的地址！

● 在用指针变量作为函数参数时，是通过改变形参指针所指的单元中的值，来改变实参指针所指的单元中的值，因为它们所指的地址是相同的。但如果在被调用函数中只改变了形参指针值(即地址)，则也不会改变实参指针的值(即地址)，即形参指针值的改变是不能改变实参指针值的。

例如,如果将例10-5中的程序改为:

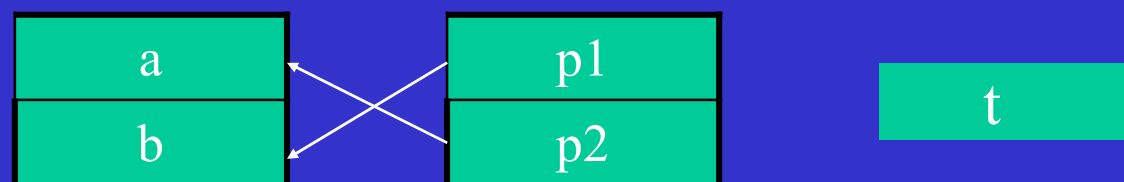
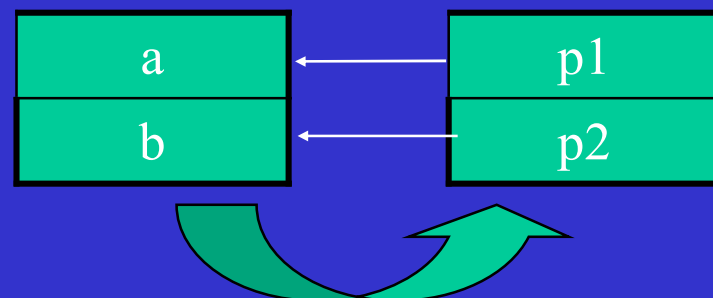
```
#include <stdio.h>
void swap(int *p1, int *p2)
{ int *t;
  t=p1; p1=p2; p2=t;
  return;
}
```



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

运行结果是:

```
12 34
a=12,b=34
a=12,b=34
```



swap函数中的p1和p2指针借助t进行了交换，
结果p1指向了b，而p2指向了a，但a和b的值
并没有被改变！

又例如有程序:

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

```
void f(int *p)
```

```
{ *p += 3; printf("%d\n", *p);
```

```
} /*p指针所指单元的内容加3 */
```

```
void main( )
```

```
{ int a[]={10,20,30,40,50,60}, *q=a;
```

```
printf("%d\n", *q);
```

```
f(q);
```

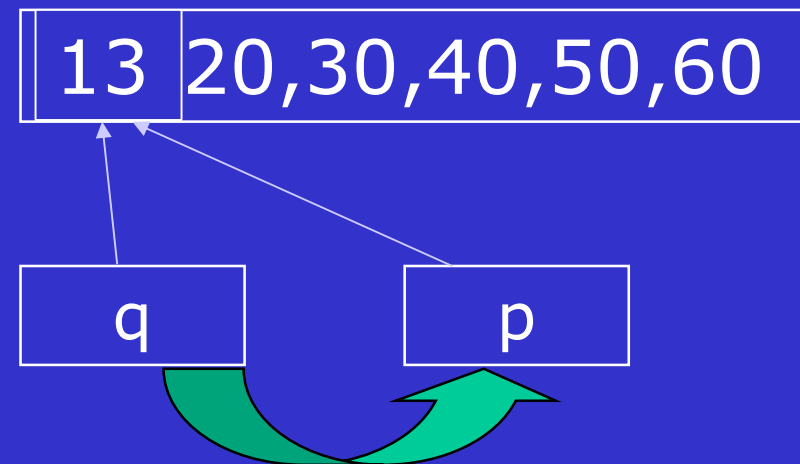
```
printf("%d\n", *q);
```

```
} 运行结果:
```

10

13

13



参数传递是传值，指针值，p和q
指针指向的是同一个内存单元

如果改为:

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

```
void f(int *p)
```

```
{  p += 3; printf("%d\n", *p);
```

```
} /* p指针加3 */
```

```
void main( )
```

```
{  int a[]={10,20,30,40,50,60}, *q=a;
```

```
    printf("%d\n", *q);
```

```
    f(q);
```

```
    printf("%d\n", *q);
```

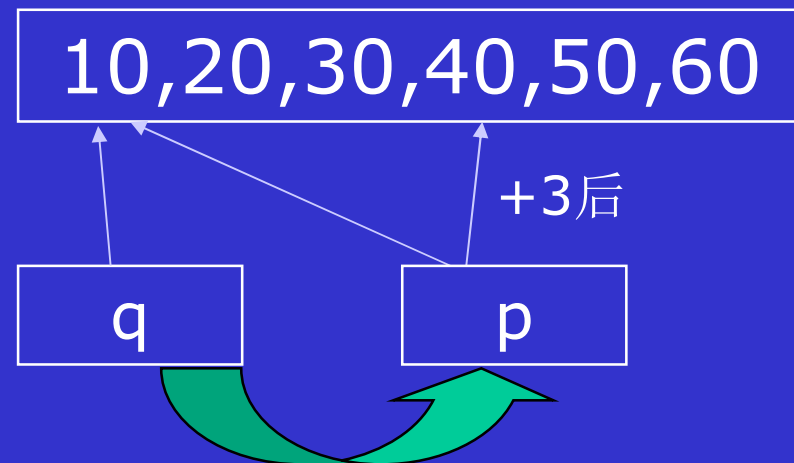
```
}
```

运行结果:

10

40

10



参数传递是传值,指针值
因此p的改变与q无关

10.1.4 指向指针的指针

- 指向指针的指针就是指向指针型变量的指针

例如,有下列程序段:

```
int x,*q,**p;
```

```
q=&x;
```

```
p=&q;
```

```
**p=3;
```

指针变量

指向指针的
指针变量

```
int x,*q;
```

```
q=&x;
```

```
*q=3;
```

```
int x;
```

```
x=3;
```



- 在C语言中,通过指针可以实现间接访问,称为一级间接访问,通过指向指针的指针可以实现二级间接访问。依此类推,C语言允许多级间接访问。

- 但由于间接访问的级数越多,对程序的理解就越困难,出错的机会也会越多,因此,在C程序中很少使用超过二级的间接访问。

【例10-6】

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

```
void swap(int *a, int *b)
```

```
{ int *t;
```

```
  t = a; a = b; b = t;
```

```
}
```

```
void main( )
```

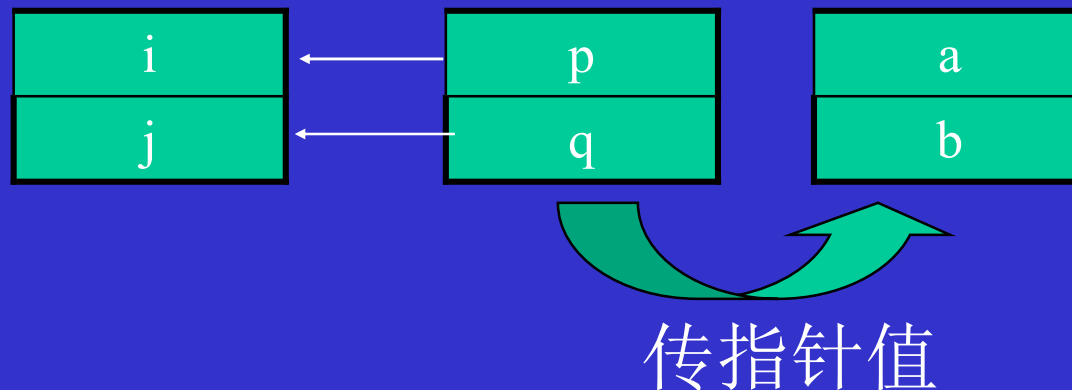
```
{ int i=3, j=5, *p=&i, *q=&j;
```

```
  printf("%d,%d,%d,%d\n", *p,*q, i, j);
```

```
  swap(p, q);
```

```
  printf("%d,%d,%d,%d\n", *p,*q, i, j);
```

```
}
```



运行结果:

3,5,3,5

3,5,3,5

形参a和b指针的交换,与实参p和q指针无关! p仍指向i, q仍指向j


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

```
void swap(int **a, int **b)
```

```
{ int *t;
```

```
    t = *a; *a = *b; *b = t;
```

```
} /* *a等价于p, *b等价于q */
```

```
void main( )
```

```
{ int i=3, j=5; int *p=&i, *q=&j;
```

```
    printf("%d,%d,%d,%d\n", *p, *q, i, j);
```

```
    swap(&p, &q);
```

```
    printf("%d,%d,%d,%d\n", *p, *q, i, j);
```

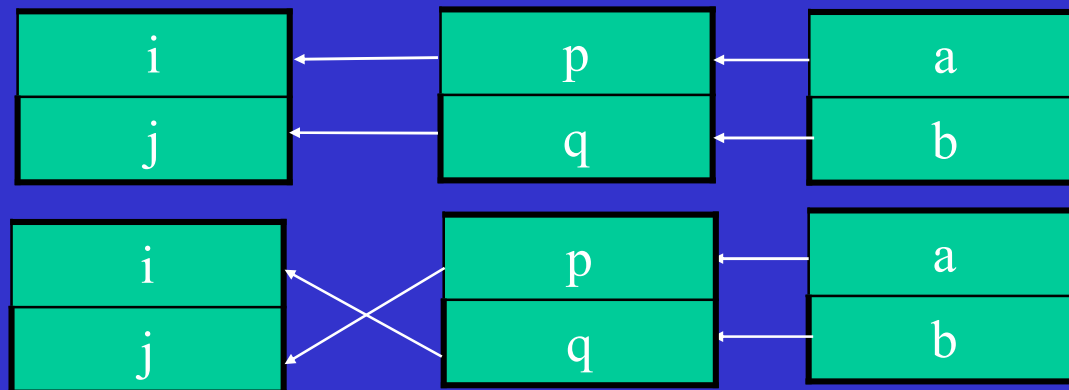
```
}
```

运行结果:

3,5,3,5

5,3,3,5

将a和b所指单元的内容交换，
因此p和q的指针值被交换了。
p指向了j，q指向了i，但i和j
变量的值没有改变。



提示: 要想让函数改变指针的值，就必须传递指针的地址！

同理，改写上一小节最后一个例子程序为：

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

```
void f(int **p)
```

```
{ *p += 3; /* 把p指针所指单元的内容加3 */  
}
```

```
void main( )
```

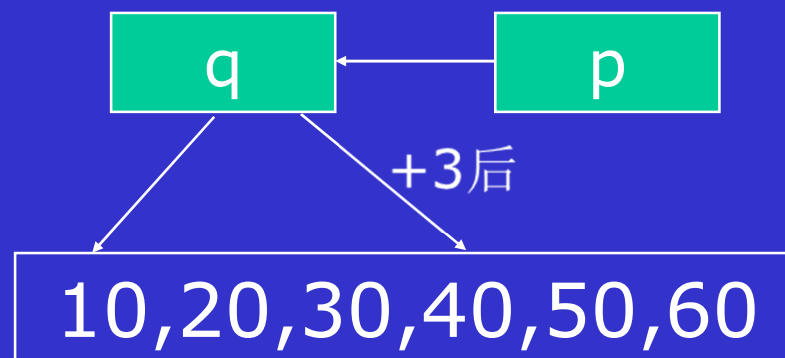
```
{ int a[]={10,20,30,40,50,60}, *q=a;  
  f(&q);  
  printf("%d\n", *q);  
}
```

输出结果：

40

调用f后，q的值改变了！

提示：要想让函数改变指针的值，就必须传递指针的地址！



10.2 指针数组

- 每个数组元素均为指针类型的数组称为指针数组。

类型标识 *数组名[数组长度说明];

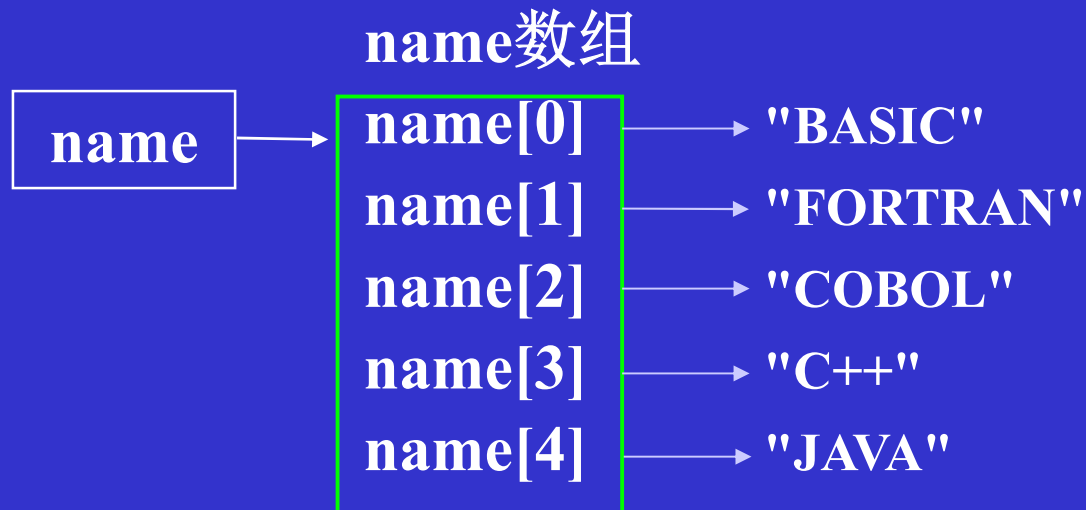
例如, `int *p[4];`

定义了长度为4的一维整型指针数组p, 其中每一个数组元素p[0],p[1],p[2],p[3]为整型指针, 用来存放整型变量的首地址。

又例如,

`char *name[]={"BASIC", "FORTRAN", "COBOL", "C++", "JAVA"};`

定义并初始化了字符型指针数组name: 其中指针数组name长度为5, 每一个指针元素都指向一个字符串常量。



【例10-7】 下列C程序的功能是，首先利用指针数组指向数组中的各元素，然后利用指向指针的指针输出数组中的各元素。

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

```
void main( )
```

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

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

```
  int **p, k;
```

```
  p=num;
```

```
  for (k=0; k<5; k=k+1)
```

```
  { printf("%5d",**p);
```

```
    p=p+1;
```

```
  }
```

```
  printf("\n");
```

```
}
```

printf("%5d",*num[k]);

printf("%5d",a[k]);

p

num数组

num[0]
num[1]
num[2]
num[3]
num[4]

a数组

a[0]=1
a[1]=2
a[2]=3
a[3]=4
a[4]=5

运行结果: 1 2 3 4 5

又例如:

```
int *a[10], b, c, d[2];
```

其中a是指针数组,有10个指针单元,每一个指针单元都可以存放一个整型变量地址。

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

```
a[1] = &c;
```

```
a[2] = d;
```

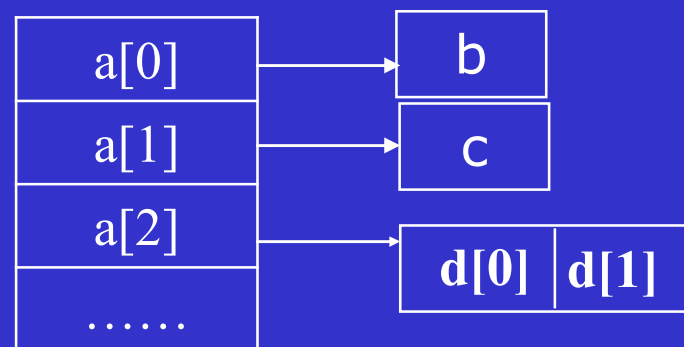
这里 *a[0]等价于b,

*a[1]等价于c, *a[2]等价于d[0]

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

```
或 a[0][0]=1; a[1][0]=2; a[2][0]=3; a[2][1]=4;
```

使得: **b=1, c=2, d[0]=3, d[1]=4**



10.3 数组与指针

10.3.1 一维数组与指针

- 数组的指针是指数组的首地址。
 - 数组元素的指针是指数组元素的地址。因此，同样可以用指针变量来指向数组或数组元素。
 - 在C语言中，由于数组名代表数组的首地址，因此，数组名实际上也是指针。
-

例如，以下四个说明语句是等价的：

```
int a[10], *p=a;  
int a[10], *p=&a[0];  
int a[10], *p; p=a;  
int a[10], *p; p=&a[0];
```

其中前两行是在说明语句中直接为指针变量p赋地址初值（即数组a的首地址），后两行是在说明语句中只定义了一个数组a与指针变量p，然后通过赋值语句为指针变量p赋a数组的地址值（即数组a的首地址）。

- C语言规定，当指针变量p指向数组的某一元素时，p+1将指向下一个元素，即p+1也表示地址。

- 但要注意的是，虽然指针变量p中存放的是地址，但p+1并不表示该地址加1个字节而是加1个该数据类型单元的字节数，指向下一个元素。

例如，如果指针变量p为int型（即指向整型变量或整型数组或整型数组元素），则p+1表示该地址的值加4（假设一个整型数据占4个字节）。

如果指针变量p为double双精度型（即指向双精度型变量或双精度型数组或双精度型数组元素），则p+1表示该地址的值加8（假设一个双精度型数据占8个字节）。

- 当一个指针变量p指向数组a的首地址时，既可以用数组名的下标法和指针法表示数组元素（如a[i]和*(a+i)），也可以用指针变量名的下标法和指针法表示数组元素（如p[i]和*(p+i)），且它们都是等价的。

- 但当一个指针变量p指向数组a的某一个元素时，虽然可以用数组名的下标法和指针法表示数组元素（如a[i]和*(a+i)），也可以用指针变量名的下标法和指针法表示数组元素（如p[i]和*(p+i)），但前两者和后两者是不等价的。

- 当p指向数组的首地址a时，p+i指向元素a[i]。

例如，如果有下列说明语句：

```
int a[10], *p=&a[0];
```

或

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

则下列赋值语句是等价的：

```
a[5]=10;
```

```
*(a+5)=10;
```

```
*(p+5)=10;
```

```
p[5]=10;
```

即a[i]，*(a+i)，*(p+i)，p[i]四种方式等价。

注意：a[i] 等价于 *(a+i) 等价于 *(i+a) 等价于 i[a]

● 但要注意，当p不指向a数组的第一个元素a[0]时，只有前两个等价，与后两个不等价。

例如，如果将上面的说明语句改成：

```
int  a[10], *p=&a[3];
```

则下列赋值语句：

```
*(p+5)=10;
```

```
p[5]=10;
```

等价于

```
a[8]=10;
```

```
*(a+8)=10;
```

【例10-8】 通过键盘为数组元素输入数据:

```
#include <stdio.h>
main() /*用数组名的下标法*/
{ int a[10], i;
  for (i=0; i<10; i++)
    scanf("%d",&a[i]);
  printf("\n");
  for (i=0; i<10; i++)
    printf("%5d\n",a[i]);
}
```

a[i] 等价于 *(a+i)

&a[i] 等价于 &*(a+i) 等价于 a+i

直接使用数组名:

```
#include <stdio.h>
main() /*用数组名的指针法*/
{ int a[10], i;
  for (i=0; i<10; i++)
    scanf("%d", a+i);
  printf("\n");
  for (i=0; i<10; i++)
    printf("%5d\n",*(a+i));
}
```

使用指针变量，将上述程序改为：

```
#include <stdio.h>
main( ) /*用指针变量名的指针法*/
{ int  a[10],*p=a, i;
  for (i=0;i<10; i++)
    scanf("%d", p+i);
  printf("\n");
  for (i=0;i<10; i++)
    printf("%5d\n",*(p+i));
}
```

使用指针变量后，指针变量所指向的数组元素也可以用下标的形式，又可以将上述程序改为：

```
#include <stdio.h>
main( ) /*用指针变量名的下标法*/
{ int  a[10],*p=a, i;
  for (i=0;i<10; i++)
    scanf("%d", &p[i]);
  printf("\n");
  for (i=0;i<10; i++)
    printf("%5d\n",p[i]);
}
```

几点说明:

(1) 指针变量可以指向数组中的任何一个元素。

例如, `int x[20], *q=&x[10];`

指针变量q指向数组x中下标为10的元素, 即在指针变量q中存放的是数组元素x[10]的地址。

(2) 用于指向数组或数组元素的指针变量类型必须与数组的数据类型相同。

例如, 下列说明是不合法的:

```
int x[20]; double *q=x;
```

数组x是整型的, 而指针变量q是双精度实型的。双精度实型指针变量不能指向非双精度实型的数组或数组元素; 同样, 非双精度实型的指针变量也不能指向双精度实型的数组或数组元素。

(3) 数组名代表数组的首地址，它实际上就是指针，数组名是不能被改变的常量指针。

实际上，在定义C语言数组的时候，系统就已经为之分配了存储空间，它的首地址是固定不变的，即不能对数组名再进行赋值（即赋予新的地址值）。例如，下列程序段是错误的：

```
int  x[10]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10};  
int  y[10];  
  
y=x;
```

在这个程序段中，定义了一个数组x，并赋了初值，又定义了一个数组y，赋值语句 y=x；企图将数组x中的数据复制到数组y中，但这是错误的，因为一旦定义了数组y后，系统就为之分配了存储空间，即数组名y已经有了一个固定的首地址，不能再将数组x的首地址赋给它。

又例如:

文件a.c:

```
#include <stdio.h>
void main( )
{ int a[10]={1,2,3,4,5,6,7,8,9,10};
  void f(int *a);
  f(a);
  printf("%d\n", a[3]);
}
```

文件b.c:

```
void f(int *a)
{ a++;
  a[2] += 2;
}
```

把a.c和b.c插入到同一个项目中, 编译链接后, 执行结果是: 6

a.c:

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

```
void main( )
```

```
{int a[10]={1,2,3,4,5,6,7,8,9,10};
```

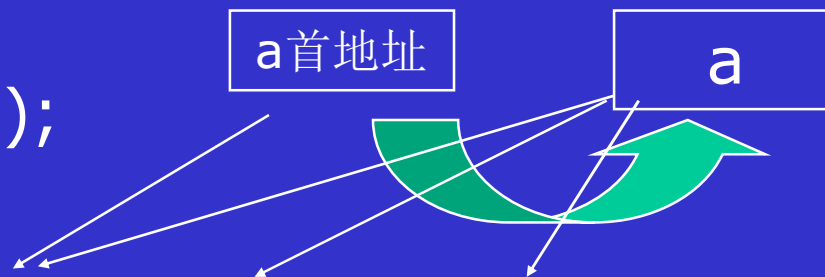
```
void f(int *a);
```

```
f(a);
```

```
printf("%d\n", a[3]);
```

```
}
```

int a[10]; void f(int *a)



b.c:

a[0],a[1],a[2],a[3].....a[9]

```
void f(int *a) /* 这里形参写成: int a[] 也可以 */
```

```
{ a++;
```

```
  a[2] += 2;
```

```
}
```

这里的f(a);函数调用，是把数组a的首地址值传递给了函数f的形参a，形参a是一个指针，可以随便修改。

10.3.2 二维数组与指针

1. 对二维数组的理解

假设定义并初始化了下列二维数组：

```
int a[3][4]={ {1,2,3,4},{5,6,7,8},{9,10,11,12} };
```

(1) 首先,可以将定义的二维数组a看成是定义了包含三个元素的一维指针数组: $a[0], a[1], a[2]$ 。 即

$$a = \begin{bmatrix} a[0] \\ a[1] \\ a[2] \end{bmatrix}$$

因此,数组名a指向一维指针数组中元素a[0]的首地址。即:

$$a = \&a[0]$$

$$a[0] = *a$$

$$a+1 = \&a[1]$$

$$a[1] = *(a+1)$$

$$a+2 = \&a[2]$$

$$a[2] = *(a+2)$$

一般来说有

$$a+i=\&a[i]$$

$$a[i] = *(a+i)$$

(2) 其次，一维指针数组中的每一个元素 $a[i]$ ($i=0,1,2$) (其元素值为地址，即指针) 又分别指向一个一维数组，其中每个一维数组中都包含有4个元素。因此， $a[i]$ ($i=0,1,2$) 可以看成是一维普通数组名，即：

$a[0][] = \{1,2,3,4\}$

$a[1][] = \{5,6,7,8\}$

$a[2][] = \{9,10,11,12\}$

因此， $a[i]$ 指向 $a[i][0]$ ($i=0,1,2$) 的首地址。

$a[i] = \&a[i][0]$

$a[i][0] = *a[i]$

$a[i]+1 = \&a[i][1]$

$a[i][1] = *(a[i]+1)$

$a[i]+2 = \&a[i][2]$

$a[i][2] = *(a[i]+2)$

$a[i]+3 = \&a[i][3]$

$a[i][3] = *(a[i]+3)$

一般来说有：

$a[i]+j = \&a[i][j]$

$a[i][j] = *(a[i]+j)$

由上所述，对于一般的二维数组 a ，有以下关系：

$*(a+i)+j = \&a[i][j]$

$a[i][j] = (*(a+i)+j)$

$a+i$ 表示的是二维数组中第 i 行(即下标为 i 的行)中第一个元素的首地址(即 $\&a[i][0]$)；而 $a[0]+i$ 表示的是二维数组中第 i 个元素(以行为主排列)的首地址

● 特别需要指出的是，对于二维数组a来说，虽然a与a[0]都表示数组的首地址，但它们是有区别的。

因为a+i表示的是二维数组中第i行（即下标为i的行）中第一个元素的首地址（即&a[i][0]）；而a[0]+i表示的是二维数组中第i个元素（以行为主排列）的首地址。例如，对于下列定义的二维数组：

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

虽然a与a[0]都表示元素a[0][0]的首地址（即&a[0][0]），且a+2与a[2]都表示元素a[2][0]的首地址（即&a[2][0]）

但a+5表示的地址（即&a[5][0]）不在该数组空间中

而a[0]+5却表示元素a[1][1]的首地址（即&a[1][1]）

由此可以看出，对于二维数组a来说，虽然a与a[0]都表示数组的首地址，它们都是指针，但数组名a是指向数组行的指针；a[0]是指向数组元素的指针，a与a[0]的指针类型不一样。

2. 二维数组的指针

与二维数组对应的指针有两种：

- ① 指向数组元素的指针；
- ② 指向数组行的指针。

(1) 指向数组元素的指针变量

【例10-9】 下列C程序是将一个二维数组中的元素按矩阵方式输出。

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

```
void main()
```

```
{ int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};
```

```
    int *p;
```

```
    for (p=a[0]; p<a[0]+12; p=p+1)
```

```
    { printf("%5d", *p);
```

```
        if ((p-a[0])%4==0)
```

```
            printf("\n");
```

```
    }
```

```
}
```

运行结果：

1 2 3 4

5 6 7 8

9 10 11 12

请按任意键继续...

(2) 指向数组行的指针变量

又称为行指针，所谓指向数组行的指针变量p，是指当p指向数组的某一行时，p+1将指向数组的下一行。

即：如果p=&a[i]时，则p+1=&a[i+1]。显然，在这种情况下，就不能用指向普通变量的指针作为指向数组行的指针。

定义指向数组行的指针变量的一般形式如下：

类型标识符 (*指针变量名)[数组行元素个数];

例如，

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

表示p是一个行指针变量，用以指向每行有4个元素的二维数组。

特别要注意，不要将

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

与

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

混淆，因为后者是具有4个指针元素的指针数组。

可以将【例10-9】的C程序改成：

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

```
void main()
```

```
{ int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};
```

```
    int *q,(*p)[4];
```

```
    for (p=a; p<a+3; p=p+1) /* 注意：这里的p加1是加了1行 */
```

```
    { for (q=p; q<p+1; q=q+1)
```

```
        printf("%5d",*q);
```

```
        printf("\n");
```

```
    }
```

```
}
```

编译结果：

10-9.c(6) : warning C4047: “=”: “int *” 与 “int (*)[4]” 的间接级别不同

因为q与p不是同一种类型的指针，应该： q=(int *)p

其中p是 int (*)[4] 类型， q是 int * 类型

运行结果：

1 2 3 4

5 6 7 8

9 10 11 12

请按任意键继续...

上面的程序也可以改写为:

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

```
void main()
```

```
{ int a[3][4]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}, i, j;
```

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

```
  p = a;
```

```
  for (i=0; i<3; i++)
```

```
  { for (j=0; j<4; j++)
```

```
      printf("%5d", p[i][j]);
```

```
      printf("\n");
```

```
  }
```

```
}
```

运行结果:

1 2 3 4

5 6 7 8

9 10 11 12

请按任意键继续...

可以看出, 行指针p的使用p[i][j]与数组a的元素a[i][j]是完全一一对应的。因此数组int a[3][4]中的a可以看成是:

int (*a)[4], 也就是说, 二维数组名实际上是一个行指针, 只不过此行指针是不可修改的常量指针。

10.3.3 数组指针作为函数参数

● 一维数组指针作为函数参数

一般来说,在一维数组指针作函数参数时,有以下四种情况:

(1) 实参与形参都用数组名。

(2) 实参用数组名,形参用指针变量。

(3) 实参与形参都用指针变量。

(4) 实参用指针变量,形参用数组名。

`int *x, int n`

`int x[], int n`

| | |
|--|---|
| <pre>void main() { int a[10]; ... f(a,10); ... }</pre> | <pre>void f(int x[], int n) { }</pre> |
|--|---|

| | |
|---|---|
| <pre>void main() { int a[10],*p=a; ... f(p,10); ... }</pre> | <pre>void f(int *x, int n) { }</pre> |
|---|---|

有一种现象要说明一下，就是数组名在形参中退化为指针：

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

```
void f(int x[ ], int n);
```

```
void main( )
```

```
{ int a[10];
```

```
  f(a,10);
```

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

```
}
```

```
void f(int x[ ], int n)
```

```
{ printf("sizeof(x) =%d\n", sizeof(x));
```

```
}
```

运行结果：

sizeof(x) =4

sizeof(a)=40

请按任意键继续...

因为调用时，只是把a数组的首地址值传递给了x。

如果不传递n，函数f中将无法获知数组x的大小。

【例10-10】 下面的C程序是利用数组元素的指针来实现对数组中指定区间内的元素进行选择法排序。

```
void select(int *b, int n)
/*或 void select(int b[ ], int n) */
{ int i, j, k, d;
  for (i=0; i<=n-2; i++)
  { k=i;
    for (j=i+1; j<=n-1; j++)
      if (b[j]<b[k]) k=j;
    if (k!=i)
    { d=b[i];
      b[i]=b[k];
      b[k]=d;
    }
  }
}
```

```
#include <stdio.h>
void main( )
{ int k,*p;
  int s[10]={3,5,4,1,9,6,10,56,34,12};
  for (k=0; k<10; k++)
    printf("%4d",s[k]); /*输出原序列*/
  printf("\n");
  p=s+2;
  /*将数组元素a[2]的地址赋给指针变量p*/
  select(p, 6); /*对数组s中的第3到第8
    个（即s[2]~s[7]）元素进行排序*/
  for (k=0; k<10; k++)
    printf("%4d",s[k]);
    /*输出排序后的结果*/
  printf("\n");
}
```

运行结果为:

3 5 4 1 9 6 10 56 34 12 /* 原序列 */

3 5 1 4 6 9 10 56 34 12 /* 排序后的结果 */

只对第3~8个元素进行了排序。

● 二维数组指针作为函数参数

【例10-11】 在下面的C程序中,主函数中定义了一个 5×4 的矩阵,然后调用函数asd()对该矩阵赋值,最后在主函数中按矩阵形式输出。

```
#include <stdio.h>
void main( )
{ int i, j, a[5][4];
  void asd(int *, int, int);
  asd( (int *)a, 5, 4);
  for (i=0; i<5; i++)
  { for (j=0; j<4; j++)
    { printf("%5d", a[i][j]);
      printf("\n");
    }
  }
}
```

```
void asd(int *b, int m, int n)
{ int k=1, i, j;
  for (i=0; i<m; i=i+1)
    for (j=0; j<n; j=j+1)
    { b[i*n+j]=k;
      k=k+1;
    }
}
```

(int *)a 强行把二维数组指针转化为一维进行参数传递, 以避免出现编译错误信息。

- 还可以利用指针数组来实现二维数组的传递(静态二维数组)

```
#include <stdio.h>
void main( )
{ int i, j, a[5][4], *b[5];
  void asd(int *[ ], int, int);
  for (i=0; i<5; i++)
    b[i] = &a[i][0];
  /* 指针数组指向二维数组每一行的
     第一个元素 */
  asd( b, 5, 4);
  for (i=0; i<5; i++)
  { for (j=0; j<4; j++)
    printf("%5d",a[i][j]);
    printf("\n");
  }
}
```

```
void asd(int *b[ ], int m, int n)
{ int k=1, i, j;
  for (i=0; i<m; i++)
    for (j=0; j<n; j++)
      { b[i][j]=k;
        k++;
      }
}
```




利用指针数组传递二维数组

- 利用指针数组来实现二维数组的传递还可以这样来实现：

在主函数中定义一维指针数组，然后利用`malloc()`函数让指针数组中的每个元素指向一块4个整型元素的存储空间，存放整型二维数组中一行的4个元素。

```
#include <stdio.h>
#include <stdlib.h>
void main( )
{ int i, j, *b[5];
  void asd(int *b[ ], int m, int n);
  for (i=0; i<5; i++)
    b[i] = (int *)malloc(sizeof(int)*4);
  asd(b, 5, 4);
  for (i=0; i<5; i++)
  { for (j=0; j<4; j++)
    printf("%5d",b[i][j]);
    printf("\n");
  }
  for (i=4; i>=0; i--)
    free(b[i]);
}
```

```
void asd(int *b[ ], int m,
        int n)
{ int k=1, i, j;
  for (i=0; i<m; i++)
    for (j=0; j<n; j++)
    { b[i][j]=k;
      k++;
    }
}
```



利用指针构建动态数组，
通常可以开比静态数组更
大的数组。

【例10-12】 编写一个函数，利用指针数组求给定 $n \times n$ 矩阵A的转置矩阵A',并计算对角线元素之和。

```
double trv(int n, double *b[ ])
```

```
{  int k, j;
```

```
    double s, d;
```

```
    s=0.0;
```

```
    for (k=0;k<n;k++)
```

```
    {  s=s+b[k][k];
```

```
        for (j=k+1;j<n;j++)
```

```
        {  d=b[k][j];
```

```
            b[k][j]=b[j][k];
```

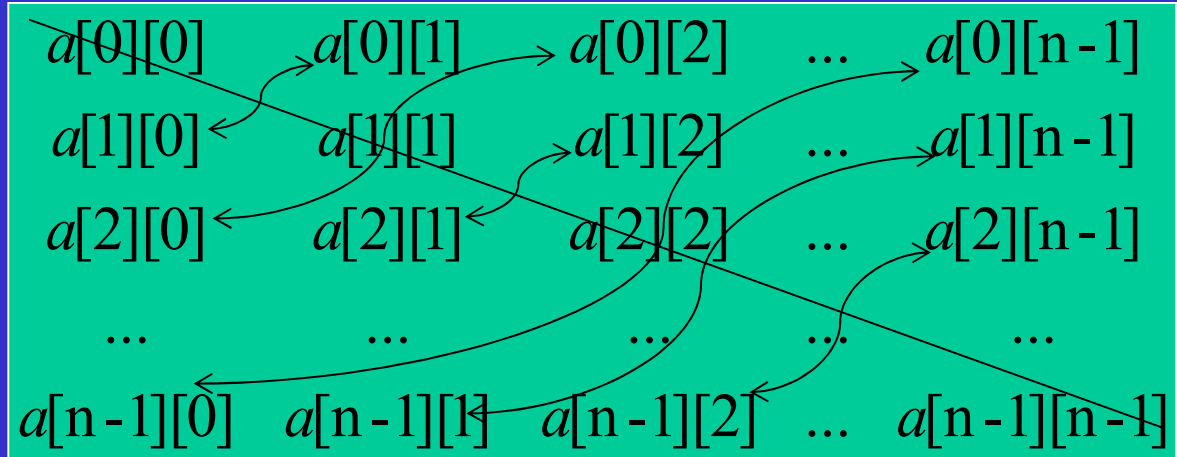
```
            b[j][k]=d;
```

```
        }
```

```
    }
```

```
    return s; /* 返回对角线元素之和*/
```

```
}
```




```

#include <stdio.h>
void main()
{ int k, j;
  double *p[4], a[4][4] =
      {{1.0,2.0,3.0,4.0},
       {5.0,6.0,7.0,8.0},
       {9.0,10.,11.,12.},
       {13.,14.,15.,16.}};
  for (k=0; k<4; k++) /*输出原矩阵*/
  { for(j=0; j<4; j++)
      printf("%7.1f", a[k][j]);
    printf("\n");
  }
  for (k=0; k<4; k++)
      p[k] = &a[k][0];
  /*指针数组指向每一行的第一个元素*/
  /* 也可以写为:  p[k] = a[k]; */

  printf("d=%7.1f\n", trv(4, p));
  /*输出对角线元素之和*/

  /*输出转置后的矩阵*/
  for (k=0; k<4; k++)
  { for (j=0; j<4; j++)
      printf("%7.1f", a[k][j]);
    printf("\n");
  }
}

```



运行结果:

1.0 2.0 3.0 4.0

5.0 6.0 7.0 8.0

9.0 10.0 11.0 12.0

13.0 14.0 15.0 16.0

d= 34.0

1.0 5.0 9.0 13.0

2.0 6.0 10.0 14.0

3.0 7.0 11.0 15.0

4.0 8.0 12.0 16.0

请按任意键继续...

第9次作业（第10章）

p.272-274 习题 1, 2, 4, 11, 12, 13

说明：这是本章作业，可以等下周本章全部讲完再做，或者先做已经讲过的部分作业题目。