计算机程序设计基础（1）
11 指针（上）

清华大学电子工程系

杨昉

E-mail: fangyang@tsinghua.edu.cn

上节要点

- 数组作为函数参数
  - 形参数组与实参数组采用地址结合，实现**双向传递**
  - 调用时只将实参数组的**首地址**传给形参数组
  - 向前引用说明**不能省略表示数组的[ ]**
  - **二维数组**作为形参时，可以转化为**一维数组**来处理
- 程序举例
  - **二分查找**
  - 简单排序算法：**冒泡排序**、**插入排序**、**选择排序**

课程回顾: 数组2

类型	定义	说明

字符数组输入输出	scanf("%c%c", &a[1], &a[2]);
	printf("%c\n", a[2]);
	scanf("%s%s", b, c);
	printf("%s\n", b);
		%c输入/输出时，为**数组元素地址/元素**;
		%s输入输出时，输入输出均为**数组名**;
		输入自动加结束符'\0'；输出遇'\0'则结束

字符串处理函数	strcat(s1,s2);字符串2**连接**到1的后面; strcpy(s1,s2); 将字符串2**拷贝**到1中; strncpy(s1,s2,n);将字符串2前n个字符**复制**到字符数组1中; strcmp(字符串1,字符串2);按照**字典序**比较两个字符串大小;	strlwr(字符串)将字符串中大写字母转成小写; strupr(字符串)将字符串中小写字母转成大写; sprintf(字符数组名, "输出格式", 变量列表); 将结果**输出到字符数组**中;sscanf(字符数组名, "输入格式", 变量列表);从字符数组中读入数据;
形参与实参数组	函数类型 函数名(数组类型 形参数组名); void exchange(int a[],int b[],int n)	形参与实参数组采用**地址结合**，实现**双向传递**;一维形参数组一般**不指定大小**
二维数组作函数参数	正确: int a[2][4]; int a[ ][4]; 错误: int a[2][ ]; int a[ ][ ]; matmul((int*)a,(int*)b,2,4,3);	可省略**第一维大小**,**不可省略第二维**;实参是二维数组,形参是一维数组,结合还是**地址结合**，但需**强制类型转换**

本节目录

11.1、指针变量

- 指针变量的基本概念
- 指针变量的定义与引用
- 指针变量作为函数参数
- 指向指针的指针

11.2、数组与指针

- 指针数组
- 一维数组与指针
- 二维数组与指针
- 数组指针作为函数参数

11.1、指针变量

- 指针变量的基本概念
- 指针变量的定义与引用
- 指针变量作为函数参数
- 指向指针的指针

指针变量的基本概念

- 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**
  - 普通变量的值是**数据**，指针变量的值是**地址**
- 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
- 一个变量若占多个字节的内存单元
  - 变量的地址一般均指**首地址**，即所占内存单元中第一个字节的地址

指针变量的基本概念

- 对内存数据(如变量、数组元素等)的存取有两种方法:
  - **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
  - **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**

直接存取与间接存取

- 例11-1：间接存取与直接存取示例

□直接存取：直接通过**变量名**存取

```
1  #include <stdio.h>
2  void main(){
3     int x,y;
4     x = 3;
5     y = 4;
6  }
```

□间接存取：通过**指针**来访问数据存储地址

```
1  #include <stdio.h>
2  void main(){
3     int x,y,*s;//定义整型变量x,y
4     //和存储整型变量地址的指针变量s
5     s = &x;
6     //将x的首地址赋给s
7     *s = 3;
8     //将s所指向地址中的内容改为3
9     s = &y;
10    //将y的首地址赋给s
11    *s = 4;
12    //将s所指向地址中的内容改为4
13 }
```

两个代码的最终效果是**等价的**

指针变量的定义与引用

- 定义指针变量的一般形式为

类型标识符　*指针变量名;

  - 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
  - 使用时，在指针变量名前加“*”表示**间接存取**

- 变量的指针就是**变量的地址**

  - 指针变量用于存放变量的地址（即指向变量）

注意问题

- 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 *s;中说明了s是一个指针变量，但**不能说*s是指针变量**
  - 使用时，在指针变量名前加“*”表示对该指针指向的地址中存有的内容进行操作（**间接存取**）
  - 指针不论类型，**在32位编译器下**，**都是4个字节**

注意问题

●例11-2：输出不同类型的指针的大小

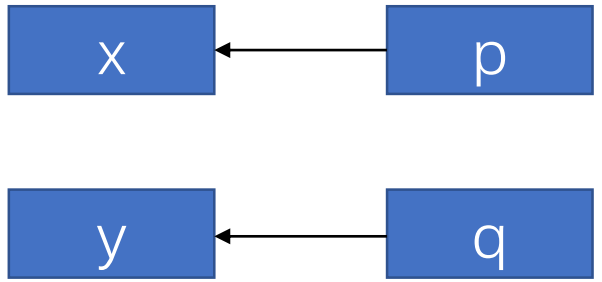
```
1  #include <stdio.h>
2  void main(){
3    short *a;
4    int *b;
5    float *c;
6    char *d;
7    printf("%d %d %d %d\n",sizeof(a),sizeof(b),sizeof(c),sizeof(d));
8  }
```

在32位编译器下，首地址均采用32位比特表示，即4个字节

```
4 4 4 4
请按任意键继续……
```

注意问题

●例11-3：打印指针值




```
1  #include <stdio.h>
2  void main(){
3    double x=0.11,y=0.1;
4    double *p,*q;                          //定义双精度实型指针变量p与q
5    p= &x; q= &y;                          //p指向x, q指向y
6    printf("&x=%u,&y=%u\n",&x,&y);//输出变量x与y的地址
7    printf("p=%u,q=%u\n", p, q);   //输出指针变量p与q中存放的地址
8    printf("x=%f,y=%f\n", x, y);   //输出变量x与y的值
9    printf("*p=%f,*q=%f\n",*p,*q);//输出指针变量p与q所指向的变量值
10 }
```

```
&x=10484640,&y=10484624
p=10484640,q=10484624
x=0.110000,y=0.100000
*p=0.110000,*q=0.100000
请按任意键继续……
```

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

注意问题

- 一个指针变量只能指向与之**同类型的变量**
  - 因为不同类型的变量所占的字节数是不同的
- 例11-4：阅读以下C程序

```
1  #include <stdio.h>
2  void main(){
3    double x=0.1; //定义双精度实型变量x,并赋初值为0.1
4    int *p;       //定义整型指针变量p
5    p=&x;         //整型变量指针p指向双精度实型变量x
6    printf("x=%f\n", x); //输出双精度实型变量x的值
7    printf("*p=%f\n", *p);
8    //按实型格式输出整型指针变量p所指向的变量值
9    printf("*p=%d\n", *p);
10   //按整型格式输出整型指针变量p所指向的变量值
11 }
```

warning C4133: “=”：从“double *”到“int *”的类型不兼容

```
x=0.100000
*p=0.000000
*p=-1717986918
请按任意键继续……
```

注意问题

- 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
- 只有当指针变量中具有**确定地址值**后才能被引用
  - 像下面这样的写法会导致出现**致命错误**（**悬浮指针**）

```
int *p;
*p = 2;
```

- 与一般的变量一样，可以对指针变量进行**初始化**
  - 如以下三段代码是等效的

```
int x, *p = &x;
*p = 5;
```

```
int x, *p;
p = &x;
*p = 5;
```

```
int x = 5, *p = &x;
```

对指针的形象化理解

- 将整个内存理解为一个宾馆
  - 宾馆的一间房间代表一个变量的存储地址
  - 房间号对应变量名称
  - 住进对应房间的客人代表变量的值
  - 客人的房卡对应指针
- 可以用以下案例理解直接访问和间接访问

老王

老王住在宾馆的302房

你在前台得知老王住在302，于是开通了打开302的房卡，用房卡打开了门

间接访问

宾馆内寻找老王的你

直接访问

你在302门口喊道：老王！我要进来啦！老王听到后打开了302的门

对指针的形象化理解

- 指针只能指向同类型的变量：
  - A的房卡用来刷B房间可能存在不兼容
- 指针变量中只能存放地址：
  - 房卡只能对应打开哪间客房，而不能直接对应客人
- 悬浮指针问题：
  - 房卡只有开通了房间权限才能交由客人入住

指针变量的定义与使用

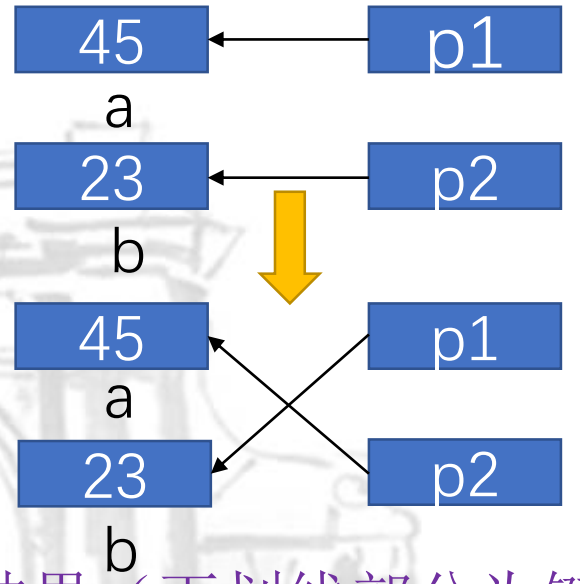
●例11-5：从键盘输入两个整数赋给变量a与b，不改变a与b的值，要求按先小后大的顺序输出

方便对比测试，输入固定为45 23

等价于scanf("%d%d", p1, p2);

```
1  #include <stdio.h>
2  void main(){
3     int a,b,*p,*p1=&a,*p2=&b;
4     scanf("%d%d", &a, &b);
5     if (a>b) { p=p1; p1=p2; p2=p; }
6     printf("a=%d,b=%d\n",a,b);
7     printf("min=%d,max=%d\n",*p1,*p2);
8  }
```

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



输出结果（下划线部分为键盘输入）

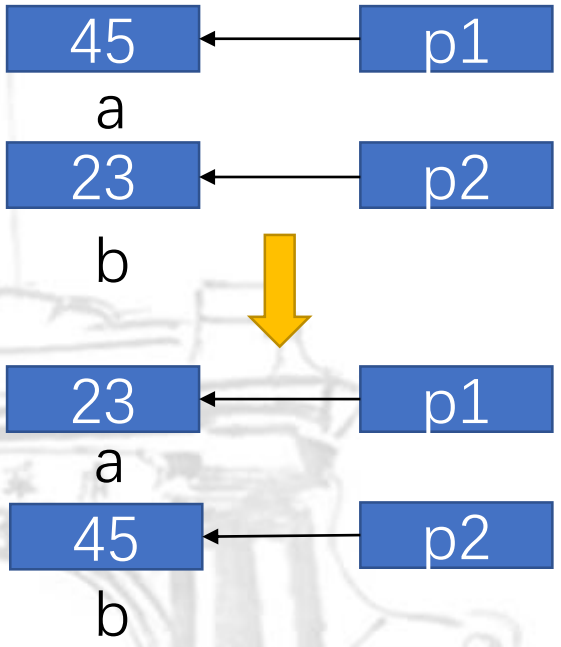
```
45 23
a=45,b=23
min=23,max=45
请按任意键继续……
```

这里将指针p1,p2进行了交换

指针变量的定义与使用

- 修改例11-5代码

```
1  #include <stdio.h>
2  void main(){
3     int a,b,p, *p1=&a,*p2=&b;
4     scanf("%d%d", p1, p2);
5     if (a>b) { p=*p1; *p1= *p2; *p2=p; }
6     printf("a=%d,b=%d\n", a, b);
7     printf("min=%d,max=%d\n",*p1,*p2);
8  }
```



输出结果（下划线部分为键盘输入）

```
45 23
a=23,b=45
min=23,max=45
请按任意键继续……
```

这里将指针p1,p2所指向的单元中的内容进行了交换，因此实际上a,b的值交换了，不符合题干要求!

指针变量作为函数参数

- 指针变量作为**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
  - 实现调用函数与被调用函数之间数据的**双向传递**
- 例11-6：计算$S(x)=x-\frac{1}{3}x^3+\frac{1}{5}x^5-\cdots\frac{(-1)^n}{2n+1}x^{2n+1}$，$-\infty<x<\infty$，直到最后一项$\left|\frac{(-1)^n}{2n+1}x^{2n+1}\right|<0.0001$为止
  - 注意：此处$(-1)^n x^{2n+1}$的计算不要直接计算，在前面一项的计算结果基础上**递推计算**，令$\mathrm{p_n}=(-1)^n x^{2n+1}$，有$\mathrm{p}_n=-\mathrm{p}_{n-1}x^2$

指针变量作为函数参数

□计算所用函数：

```
#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 m=0;
  double p,t,s;
  p=x; s=x;
  do{
    m=m+1;                    //项数计数
    p= -p*x*x;
    t=p/(2*m+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=m;                      //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
}
```

指针变量作为函数参数

- C程序编写主函数：

```
1  void main(){
2    int n;
3    double x, s;
4    printf("input x:");
5    scanf("%lf", &x);
6    s=arctan(x, 0.0001, &n);
7    printf("n=%d\ns=%f\n", n, s);
8  }
```

```
input x:1.0
n=5000
s=0.785448
请按任意键继续……
```

- 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
- 同时传递n的地址作为参数给被调用函数，通过**指针操作把多项式项数**也传递回了调用它的函数

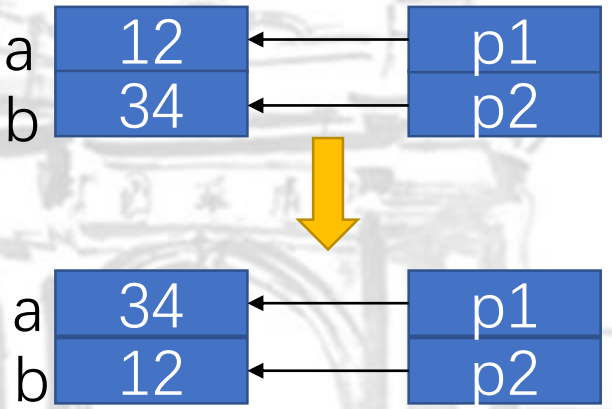
指针变量作为函数参数


●例11-7：利用指针变量实现两个变量值的互换

```
1  #include <stdio.h>
2  void swap(int *p1, int *p2){
3    int t;
4    t=*p1; *p1=*p2; *p2=t;
5    return;
6  }
7  void main(){
8    int a, b;
9    scanf("%d,%d",&a,&b);
10   printf("a=%d,b=%d\n",a, b);
11   swap(&a, &b);
12   printf("a=%d,b=%d\n",a, b);
13 }
```

参数传递是**传值**,变量的地址值

方便对比测试，输入固定为12,34




```
12,34
a=12,b=34
a=34,b=12
请按任意键继续……
```

将p1和p2**所指单元的内容交换**，因此a和b的值被交换了

指针变量作为函数参数

- 在用指针变量作为函数参数时
  - 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**
  - 如果在被调用函数中只改变了**形参指针的值**(即地址),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
- 若改变例11-7中的交换函数
  - 将交换形参指针所指单元的内容改为**交换形参指针指向的单元**
  - 这是不能发挥作用的，因为函数形参值的改变**不会传递给实参**

指针变量作为函数参数

●例11-8：将例11-7的swap()函数修改如下

方便对比测试，输入固定为12, 34

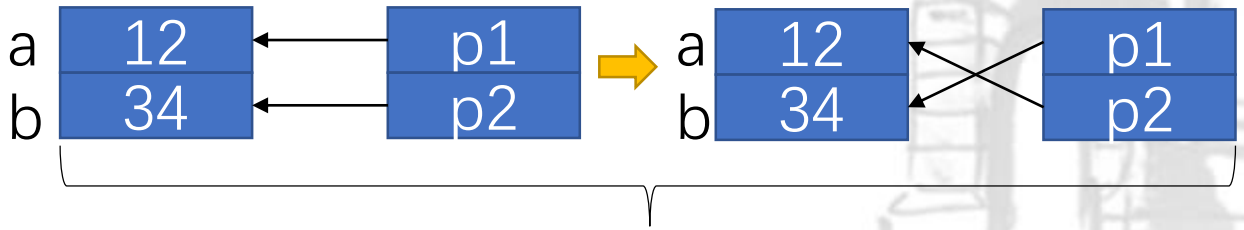
```
1  #include <stdio.h>
2  void swap(int *p1, int *p2){
3     int *t;
4     t=p1; p1=p2; p2=t;
5     return;
6  }
```

可见swap()函数内指针值的改变并没有传回主函数内

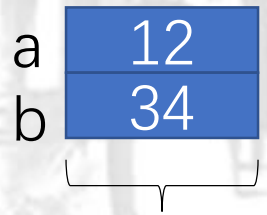
```
12,34
a=12,b=34
a=12,b=34
请按任意键继续……
```

更多案例，参见课本P234下方例题

形参值p1, p2并不传递回主函数



swap()函数内部指针值进行交换



返回主函数后各变量的情况

指向指针的指针


- 指向指针的指针就是**指向指针型数据的指针**
  - 例如下列程序段


```
int x, *q, **p;
q = &x;
p = &q;
**p = 3;
```

定义时“*”表示指针变量

定义时“**”表示指向指针的指针变量

使用时“**”表示间接存取该变量指向的指针所指向的内存地址中的内容

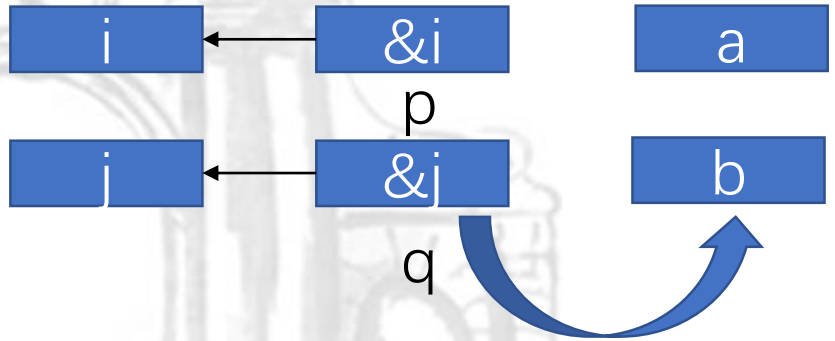
  - 在C语言中,通过指针可以实现间接访问,称为**一级间接访问**
  - 通过指向指针的指针可以实现**二级间接访问**，依此类推,C语言允许**多级间接访问**
  - 但由于间接访问的级数越多,对程序的理解就越困难,出错的机会也会越多,因此,在程序中**很少使用超过二级的间接访问**

指向指针的指针

●例11-9：指向指针的指针示例

```
1  #include <stdio.h>
2  void swap(int *a, int *b){
3    int *t;
4    t = a; a = b; b = t;
5  }
6  void main(){
7    int i=3,j=5, *p=&i, *q=&j;
8    printf("%d,%d,%d,%d\n", *p,*q, i, j);
9    swap(p, q);
10   printf("%d,%d,%d,%d\n", *p,*q, i, j);
11 }
```

```
3,5,3,5
3,5,3,5
请按任意键继续……
```



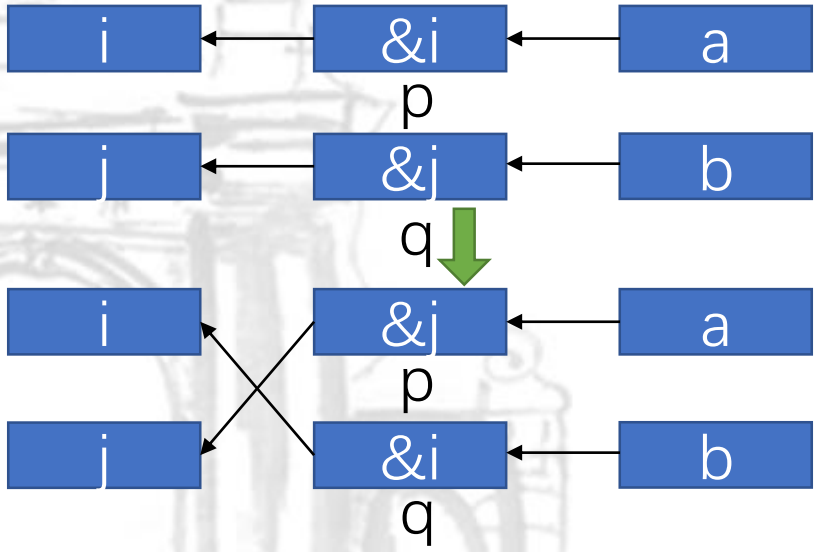
采用**传值方式**，函数体中的**a,b数值的变化和主函数中的p,q无关**

指向指针的指针


- 如果将代码修改如下：

```
1  #include <stdio.h>
2  void swap(int **a, int **b){
3    int *t;
4    t = *a; *a = *b; *b = t;
5  } // *a等价于p, *b等价于q
6  void main(){
7    int i=3, j=5; int *p=&i, *q=&j;
8    printf("%d,%d,%d,%d\n", *p, *q, i, j);
9    swap(&p, &q);
10   printf("%d,%d,%d,%d\n", *p, *q, i, j);
11 }
```

```
3,5,3,5
5,3,3,5
请按任意键继续……
```



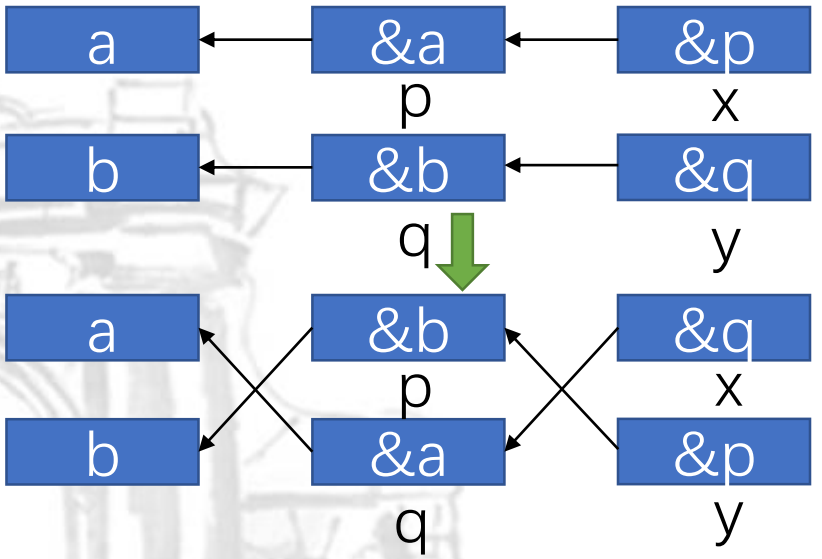
将a和b**所指单元的内容**交换,因此p和q的指针值被交换了。p指向了j，q指向了i，但i和j变量的值没有改变

课堂练习

●练习11-1：阅读以下程序，写出答案

```
1  #include <stdio.h>
2  void main(){
3     int a=12,b=34,*p,*q,**x,**y;
4     int *m,**n;
5     p = &a,  x = &p;
6     q = &b,  y = &q;
7     n = y,  y = x,  x = n;
8     m = p,  p = q,  q = m;
9     printf("%d %d %d %d %d %d\n",**x,**y,*p,*q,a,b);
10 }
```

```
12 34 34 12 12 34
请按任意键继续……
```



面对这种问题，可以像右边一样通过框图的方式来理解，方块旁标注变量名称，方块内部标注变量数值，箭头表示指针变量指向的变量

11.2、数组与指针

- 指针数组
- 一维数组与指针
- 二维数组与指针
- 数组指针作为函数参数

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

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

- 定义并初始化了**字符型指针数组**，数组长度为5，每一个元素都指向了一个字符串常量

指针数组

●例11-10：下面的程序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

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

```
 1   2   3   4   5
请按任意键继续……
```

等价于printf("%5d",*num[k]);

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
- 由于**数组名代表数组的首地址**，**数组名**实际上也是**指针**
  - 以下四个语句是等价的：

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

  - 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）
  - 后两行定义了数组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）

一维数组与指针

- 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
  - p+1表示加1个该数据类型单元的字节数，指向下一个元素
  - 即如果p为int型指针，则p+1是地址p加上4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)

一维数组与指针

- 例11-11：辨析*r++与(*r)++的区别
  - 假设有一个整型数组a[5]，一个指向整型数的指针r存有数组的首地址a，定义一个整型变量n
  - 如果有语句n=*r++;则该语句等价于n=*r; r = r + 1
    - 指针加一的含义是指向当前地址的下一个单元，在这里即指向数组的下一个元素，执行的结果为a[0]的值被赋给变量n，指针r指向元素a[1]
  - 如果有语句n=(*r)++;则该语句等价于n=a[0]++
    - 指针变量本身不进行任何操作，最终的执行结果是n=a[0];a[0]=a[0]+1;指针依旧指向元素a[0]

一维数组与指针

- 当p指向数组的首地址a时，p+i指向元素a[i]
- 当p指向数组元素**a[j]**时，p+i指向元素**a[i+j]**
  - 因此当p指向的不是数组的首地址时，前述的数组名和指针变量名表示法**不能简单等价**
  - 有数组a[10]和指针p如下： int a[10], *p=&a[3];

    则下列四个赋值语句**相互等价**

    *(p+5)=10; ⟺ a[8]=10;

    p[5]=10; ⟺ *(a+8)=10;

一维数组与指针

●例11-12：通过键盘为数组元素输入数据

□使用数组名

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

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

一维数组与指针

□使用指针变量名

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

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

上面四段程序是相互等价的！

几点说明

- 指针变量可以指向数组中的**任何一个元素**
- 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**
- 数组名代表数组的首地址,它实际上就是指针,不能被改变的**常量指针**

几点说明

- 例11-13：不能对数组名**再进行赋值**（赋予新的地址值）
  - 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
  - 如

```
int x[10]={0,1,2,3,4,5,6,7,8,9};
int y[10];
y=x;
```

这种写法是错误的！

几点说明

- 但是右边的程序是正确的
- 原因在于：这里是在函数内部修改了形参a的数值，而形参除了和实参**数值相同**之外**互不影响**
- 相当于**新建了一个指针**指向数组的首地址，再对这个新建的指针进行操作

```
1  #include<stdio.h>
2  void f(int *a){
3    //这里形参写成 int a[] 也可以
4    a++;
5    a[2] += 2;
6  }
7  void main(){
8    int a[10]={1,2,3,4,5,6,7,8,9,10};
9    void f(int *a);
10   f(a);
11   printf("%d\n", a[3]);
12 }
```

```
6
请按任意键继续……
```

二维数组与指针

- 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} };
```

- 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**(实际并不存在)：a[0],a[1],a[2]
  - 数组名a指向该指针数组的首地址，即 $a=\begin{pmatrix} a[0] \\ a[1] \\ a[2] \end{pmatrix}$
  - 则 a=&a[0]　　a+1=&a[1]　　a+2=&a[2]

    a[0]=*a　　a[1]=*(a+1)　　a[2]=*(a+2)

二维数组与指针

- 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**
  - 每个一维数组中都包含有4个元素
  - a[i] (i=0,1,2)可以看成是一维普通数组名

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

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

    a[2][ ]={9,10,11,12}
  - a[i]指向a[i][ ](i=0,1,2)的**首地址**

二维数组与指针

- 由上所述，有以下结论

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)

- 更为一般的，有

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

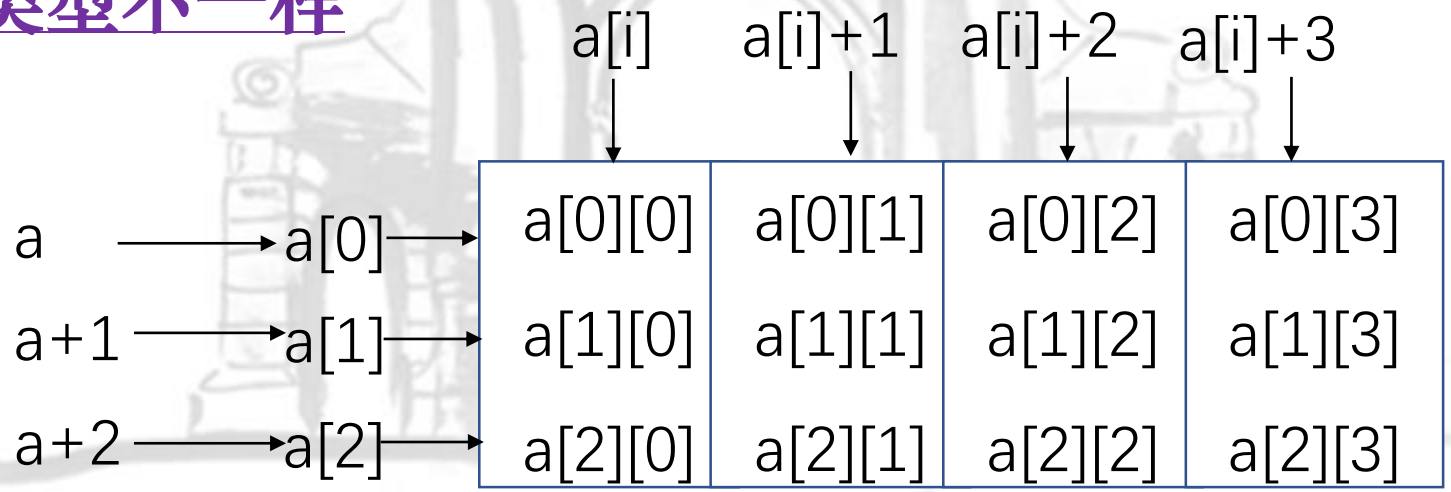
***(a+i)+j =&a[i][j]**　　**a[i][j] = *(*(a+i)+j)**

二维数组与指针


- 对二维数组a，a与a[0]都表示数组的首地址，但是**有区别**
  - 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]）
  - 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
  - 对于如下定义的二维数组： 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]**）
    - ✓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]**）

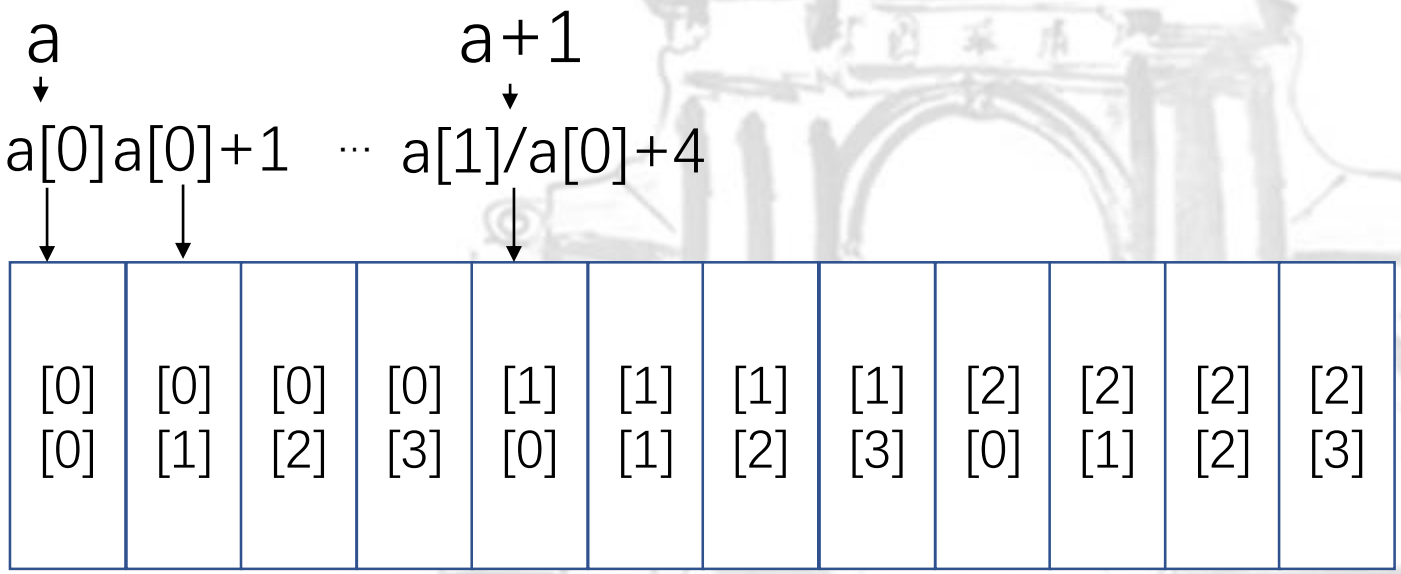
二维数组的指针

- 由上面的分析可以看出，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**
  - 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**



二维数组的指针

- 不难想到，二维数组在内存中事实上是按照以行为主的方式**一维顺序排列存储**的



二维数组的指针

- 二维数组作为函数参数时，在实参中同样将数组名作为参数进行传递，但在形参中使用的是一维数组
  - 使用时首先使用**强制类型转换**(int*)将二维数组名转换为一维数组名
  - 根据二维数组元素以行为主存储的原则，原先二维数组中第i行第j列元素对应到一维数组中元素下标为i*n+j
- 前面的分析中也提到，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

二维数组的指针

- **指向数组元素**的指针变量：与一维数组相同，指向二维数组元素的指针与一般的指向普通变量的指针变量相同
- 例11-14：将一个二维数组中的元素按矩阵方式输出

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

```
   1   2   3   4
   5   6   7   8
   9  10  11  12
请按任意键继续……
```

二维数组的指针：行指针

- **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];

  - 如：**int (*p)[4];** 表示p是一个行指针变量，指向每行有4个元素的数组；注意不要与**int *p[4];**混淆，这是包含四个元素的指针数组

二维数组的指针

●例11-15：将例11-14的代码改用行指针进行编写

```
1  #include <stdio.h>
2  void main( ){
3    int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};
4    int *q,(*p)[4];
5    for (p=a; p<a+3; p=p+1){
6    //注意：这里的p加1是加了1行
7      for (q=p; q<p+1; q=q+1)
8        printf("%5d",*q);
9      printf("\n");
10   }
11 }
```

编译结果：warning C4047: “<”：“int *”与“int (*)[4]”的间接级别不同

```
   1   2   3   4
   5   6   7   8
   9  10  11  12
请按任意键继续……
```

因为q与p不是同一种类型的指针，规范写法应该：q=(int *)p

二维数组的指针

- 指针类型**强制转换**
  - 从前面的例子可以看出，通过强制类型转换，可以**直接**将二维数组的行指针转换为指向数组元素的指针
  - 二者所存储的**地址值是相同**的，只是**指针数据类型不同**导致后续操作中会有不同
  - 例如可以将二维数组名a**直接转换**为指向数组元素的指针q，同时按照**一维数组**的形式输出，结果依然是正确的
  - 可见强制转换**并不会改变指针变量的值**，二维数组在内存中也确实是按照**一维方式以行为主顺序存储**的

二维数组的指针

- 例11-16：将例11-14改用指向数组元素的指针编写
  - 将二维数组名a直接转换为指向数组元素的指针q

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

```
   1   2   3   4
   5   6   7   8
   9  10  11  12
请按任意键继续……
```

二维数组的指针

- 例11-17：行指针与数组元素一一对应
  - 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]

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

```
    1    2    3    4
    5    6    7    8
    9   10   11   12
请按任意键继续……
```

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

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:
  - **实参与形参都用数组名**

```
void main(){
  int a[10];
  ...
  f(a,...);
  ...
}
```

```
void f(int x[],...){
  ...
  ...
  ...
  ...
}
```

  - **实参用数组名,形参用指针变量**

```
void main(){
  int a[10];
  ...
  f(a,...);
  ...
}
```

```
void f(int *x,...){
  ...
  ...
  ...
  ...
}
```

一维数组指针作为函数参数

□形参和实参都用指针变量

```
void main(){
  int a[10],*p=a;
  ...
  f(p,...);
  ...
}
```

```
void f(int *x,...){
  ...
  ...
  ...
  ...
}
```

□实参用指针变量,形参用数组名

```
void main(){
  int a[10],*p=a;
  ...
  f(p,...);
  ...
}
```

```
void f(int x[],...){
  ...
  ...
  ...
  ...
}
```

这四种写法都是正确的，且**指针变量指向的是数组首地址**时，是**完全等价**的

若指针变量指向的**不是数组首地址**，则不能够简单的等价，需要一定的转换

一维数组指针作为函数参数


- 有一种现象需要特别注意：在形参中，**数组名退化为指针**
  - 调用时，只是把a数组的**首地址值传递给了x**
  - 在函数中看来，这就是一个**普通的指针变量**
- 练习11-2：阅读以下程序

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

```
sizeof(x) =4
sizeof(a)=40
请按任意键继续……
```

一维数组指针作为函数参数

- 例11-18：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**
  - 选择排序的基本思想和代码编写在上一章中已经有所阐述
  - 这里只是将形参中的数组改成了指针的形式

```
1  void select(int *b, int n){
2  //或 void select(int b[], int n)
3    int i,j,k,d;
4    for (i=0; i<=n-2; i++){
5      k=i;
6      for (j=i+1; j<=n-1; j++)
7        if (b[j]<b[k]) k=j;
8      if (k!=i){
9        d=b[i];
10       b[i]=b[k];
11       b[k]=d;
12     }
13   }
14 }
```

一维数组指针作为函数参数

```
15 #include <stdio.h>
16 void main(){
17   int k,*p;
18   int s[10]={3,5,4,1,9,6,10,56,34,12};
19   for (k=0; k<10; k++)
20     printf("%4d",s[k]); //输出原序列
21   printf("\n");
22   p=s+2;
23   //将数组元素a[2]的地址赋给指针变量p
24   select(p, 6);
25   //对数组a中的第3到第8个
26   //（即a[2]～a[7]）元素进行排序
27   for (k=0; k<10; k++)
28     printf("%4d",s[k]);
29   //输出排序后的结果
30   printf("\n");
31 }
```

- 传入的实参p指向的是a[2]
- 在select()函数看来，形参是以&a[2]为首地址的一个数组

可见数组只对第3到第8位进行了排序

```
  3   5   4   1   9   6  10  56  34  12
  3   5   1   4   6   9  10  56  34  12
请按任意键继续……
```

二维数组指针作为函数参数

●二维数组名作为实参

- □二维数组在内存中以行为主的方式一维顺序排列存储
- □可以将形参设置为一维数组的方式，同时对下标做相应转换

●例11-19：主函数定义了一个5×4的矩阵，asd()函数对其赋值

```
1  #include <stdio.h>
2  void asd(int *b, int m, int n){
3    int k=1, i, j;
4    for (i=0; i<m; i=i+1)
5      for (j=0; j<n; j=j+1){
6        b[i*n+j]=k;
7        k=k+1;
8      }
9  }
```

```
10 void main(){//行指针强制类型转换
11   int i, j, a[5][4];
12   asd( (int *)a, 5, 4);
13   for (i=0; i<5; i++){
14     for (j=0; j<4; j++)
15       printf("%5d", a[i][j]);
16     printf("\n");
17   }
18 }
```

```
    1    2    3    4
    5    6    7    8
    9   10   11   12
   13   14   15   16
   17   18   19   20
请按任意键继续……
```

二维数组指针作为函数参数

●例11-20：二维数组行指针对矩阵赋值

- □同样的，也可以将形参设置成**二维数组行指针**的形式
- □这样的缺点是需要知道**矩阵的列数**，缺乏通用性

```
1  #include <stdio.h>
2  void asd(int (*b)[4], int m, int n){
3    int k=1, i, j;
4    for (i=0; i<m; i=i+1)
5      for (j=0; j<n; j=j+1){
6        b[i][j]=k;
7        k=k+1;
8      }
9  }
```

```
10 void main(){
11   int i, j, a[5][4];
12   asd( a, 5, 4);
13   for (i=0; i<5; i++){
14     for (j=0; j<4; j++)
15       printf("%5d", a[i][j]);
16   printf("\n");
17   }
18 }
```

```
    1    2    3    4
    5    6    7    8
    9   10   11   12
   13   14   15   16
   17   18   19   20
请按任意键继续. . .
```

二维数组指针作为函数参数

●例11-21：指针数组对矩阵赋值

□指针数组作为实参，每个元素指向二维数组对应行的首地址

```
1  #include <stdio.h>
2  void asd(int **b, int m, int n){
3  //或void asd(int *b[], int m, int n)
4    int k=1, i, j;
5    for (i=0; i<m; i++)
6      for (j=0; j<n; j++){
7        b[i][j]=k;
8        k++;
9      }
10 }
```

```
11 void main(){
12   int i, j, a[5][4], *b[5];
13   for (i=0; i<5; i++)
14     b[i] = &a[i][0];
15 /* 指针数组指向二维数组
16       每一行的第一个元素 */
17   asd( b, 5, 4);
18   for (i=0; i<5; i++){
19     for (j=0; j<4; j++)
20       printf("%5d",a[i][j]);
21     printf("\n");
22   }
23 }
```

```
    1    2    3    4
    5    6    7    8
    9   10   11   12
   13   14   15   16
   17   18   19   20
请按任意键继续……
```

二维数组指针作为函数参数


●例11-22：实现矩阵转置并计算矩阵对角线之和

```
1  #include <stdio.h>
2  double trv(int n, double *b[]){//形参为指针数组
3    int k, j;
4    double s, d;
5    s=0.0;
6    for (k=0;k<n;k++){
7      s=s+b[k][k];          // 累加对角线元素
8      for (j=k+1;j<n;j++){// 交换对称位置元素
9        d=b[k][j];
10       b[k][j]=b[j][k];
11       b[j][k]=d;
12     }
13   }
14   return s; // 返回对角线元素之和
15 }
```

二维数组指针作为函数参数

```
16  void main(){
17    int k, j;
18    double *p[4],a[4][4] ={{1.0,2.0,3.0,4.0},{5.0,6.0,7.0,8.0},
19                           {9.0,10.0,11.0,12.0},{13.0,14.0,15.0,16.0}};
20    for (k=0; k<4; k++){    //输出原矩阵
21      for(j=0; j<4; j++)
22        printf("%7.1f",a[k][j]);
23      printf("\n");
24    }
25    for (k=0; k<4; k++)
26      p[k] = &a[k][0];      //指针数组指向每一行的第一个元素
27    printf("d=%7.1f\n",trv(4, p));  //输出对角线元素之和
28    for (k=0;k<4;k++){                //输出转置后的矩阵
29      for (j=0;j<4;j++)
30        printf("%7.1f",a[k][j]);
31      printf("\n");
32    }
33  }
```

```
   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. . .
```

二维数组指针作为函数参数

- 内存自动分配与手动分配
  - 之前的例子中，都是首先定义好变量，再将变量的地址赋给指针，占据的内存空间都是在定义变量时**自动分配**的
  - 可以为指针**手动分配**内存空间，再给内存赋值(动态数组，下节课介绍)
  - 二维数组可以通过手动分配空间的方式创建
    - 首先定义一个长度为m的指针数组
    - 再为指针数组中每一个指针手动分配n个对应数据类型的内存空间
    - 这样就创建了一个m×n的二维数组，通过这种方式创建的称为**动态二维数组**
    - 由于其内存空间是手动分配的，并不像静态二维数组顺序存储，因此将其转换为一维数组并利用行指针等方法不再适用，只能用**指针数组**进行传递

本节重点

●指针变量

□指针变量的基本概念

- 直接存取与间接存取

□指针变量的定义与使用

- 需要注意的问题：悬浮指针

□指针作为函数参数

- 通过指针，实现形参与实参数值的双向传递

□指向指针的指针

- 实现多级间接访问

本节重点

●指针与数组

□指针数组

□一维数组与指针

- 数组名与首地址、指针变量+1

□二维数组与指针

- 对二维数组的理解
- 静态二维数组在内存中的存储方式
- 数组指针作为函数参数
- 数组名和指针变量作为形参、实参

本节作业

- 第十一次作业：
- P272 习题1
  - 写出解答过程，**电子版 or 纸质版完成后拍照**上传到到网络学堂
- P274 习题12
  - 将**源代码和运行结果截图粘贴到word文档**中提交到网络学堂
  - 其中用到的计算指数、对数、三角函数的函数包含在头文件”math.h”中，见附录B

附加作业

- 指针与常量
  - 请讨论char * const p, char const * p, 和const char *p的区别
- 指针辨析
  - 请打印右边代码的结果，观察其是否相同，并分析原因

```
1  char str1[] = "abc";
2  char str2[] = "abc";
3  const char str3[] = "abc";
4  const char str4[] = "abc";
5  const char *str5 = "abc";
6  const char *str6 = "abc";
7  char *str7 = "abc";
8  char *str8 = "abc";
9
10 printf("%d\n", str1 == str2);
11 printf("%d\n", str3 == str4);
12 printf("%d\n", str5 == str6);
13 printf("%d\n", str7 == str8);
```


THANKS