

8

指 针

8.1 引言

8.2 指针的概念

8.3 指针变量的定义

8.4 指针变量的引用

8.5 指针变量的运算

8.6 指向数组的指针

8.7 指针作为函数参数

8.8 指向字符串的指针变量

8.9 指向多维数组的指针变量

8.10 指针数组

8.11 多级指针

8.12 命令行参数

8.13 指针函数

8.14 指向void量的指针变量

8.15 指向函数的指针

8.16 动态分配内存空间和动态数组

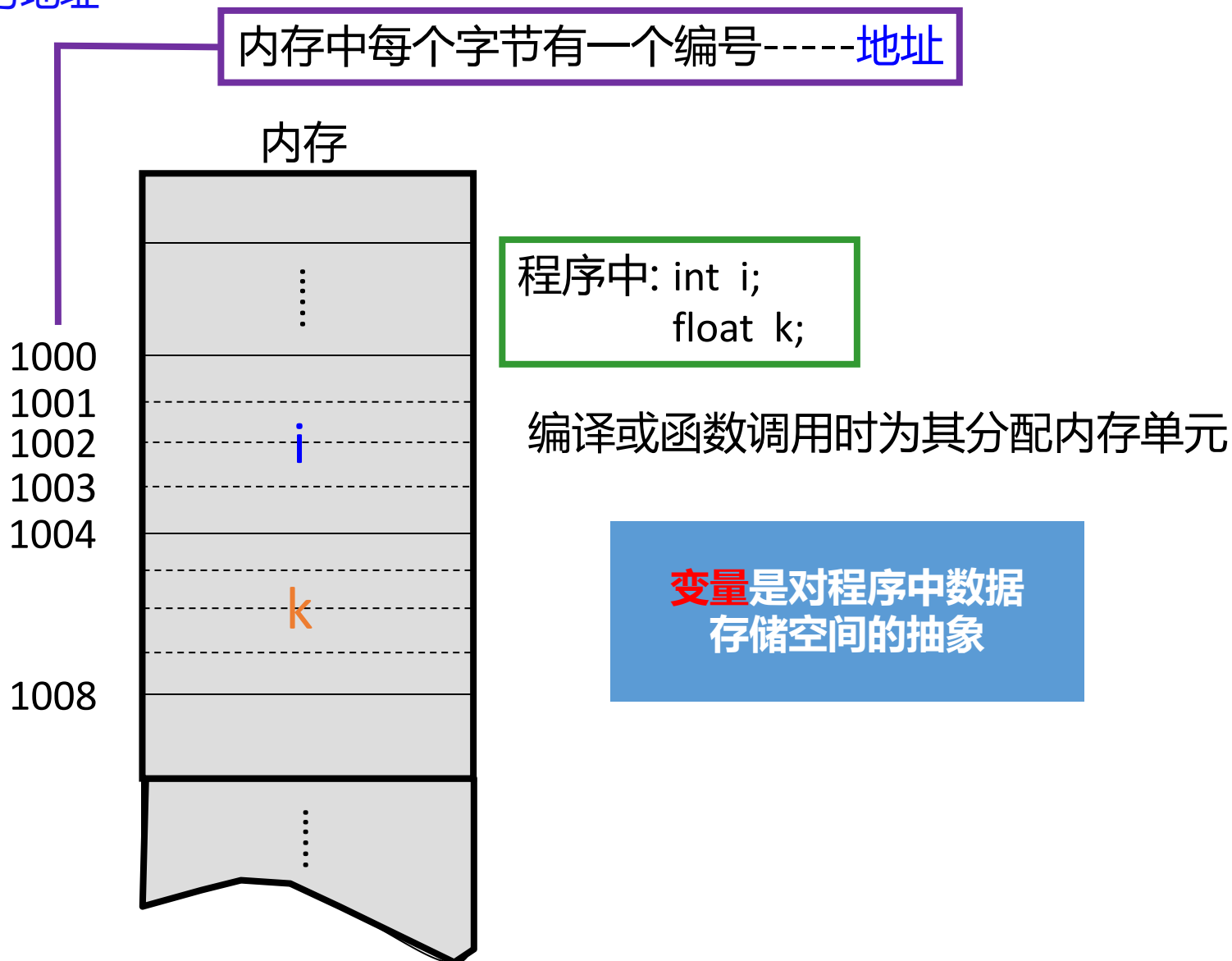
8.1 引言

C程序设计中使用指针可以:

- 使程序简洁、紧凑、高效
- 有效地表示复杂的数据结构
- 动态分配内存
- 得到多于一个的函数返回值

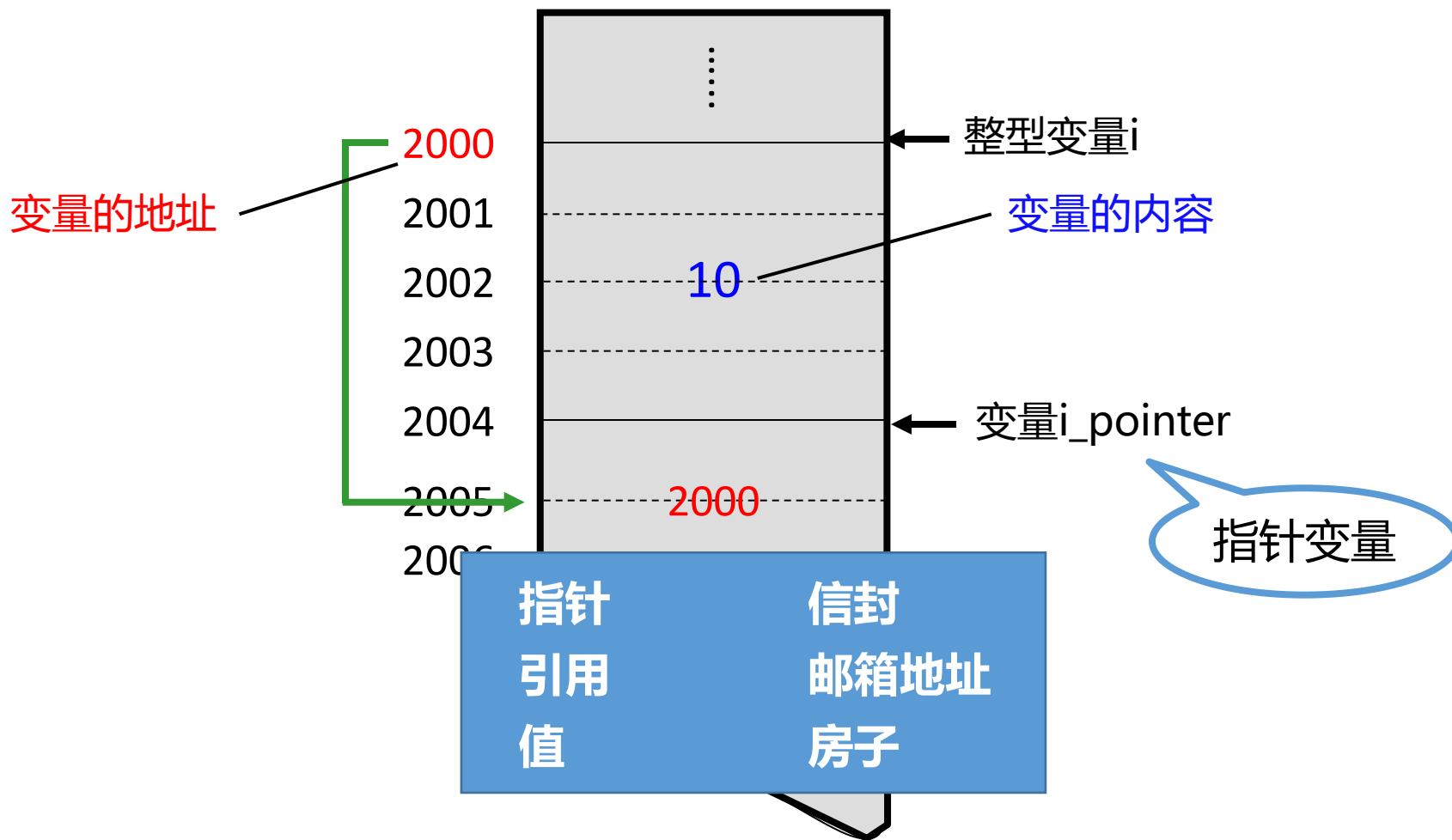
8.2 指针的概念

变量与地址



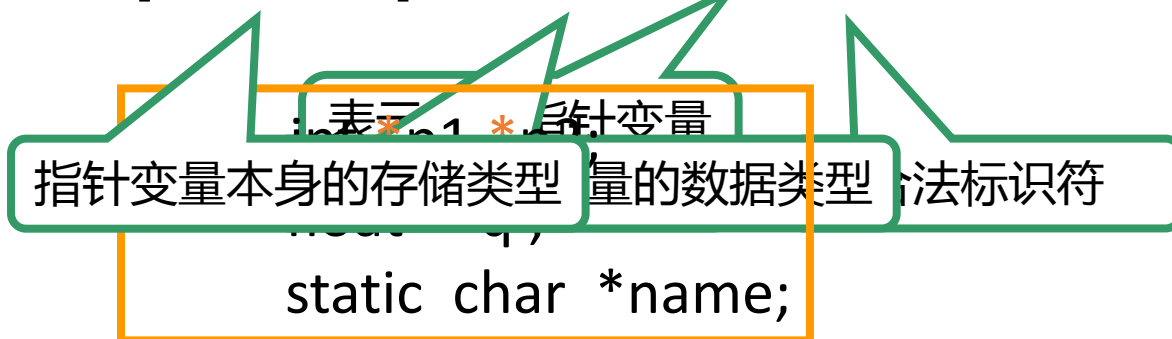
指针与指针变量

- ❖ 指针：一个变量的地址
- ❖ 指针变量：专门存放变量地址的变量叫指针变量



8.3 指针变量的定义

❖ 一般形式：[存储类型] 数据类型 *指针名;



注意：

- 1、`int *p1, *p2;` 与 `int *p1, p2;`
- 2、指针变量名是 `p1`、`p2` ,不是 `*p1`、`*p2`
- 3、指针变量只能指向定义时所规定类型的变量
- 4、指针变量定义后，变量值不确定，应用前必须先赋值

指针变量的初始化

一般形式：[存储类型] 数据类型 *指针名=初始地址值；

```
例  int *p=&i;  
     int i;
```



赋给指针变量，
不是赋给目标变量

```
例  int i;  
     int *p=&i;
```

变量必须已定义过
类型应一致

```
例  int i;  
     int *p=&i;  
     int *q=p;
```

用已初始化指针变量作初值

```
例  main( )  
    {  int i;  
        static int *p=&i;  
        .....  
    }
```



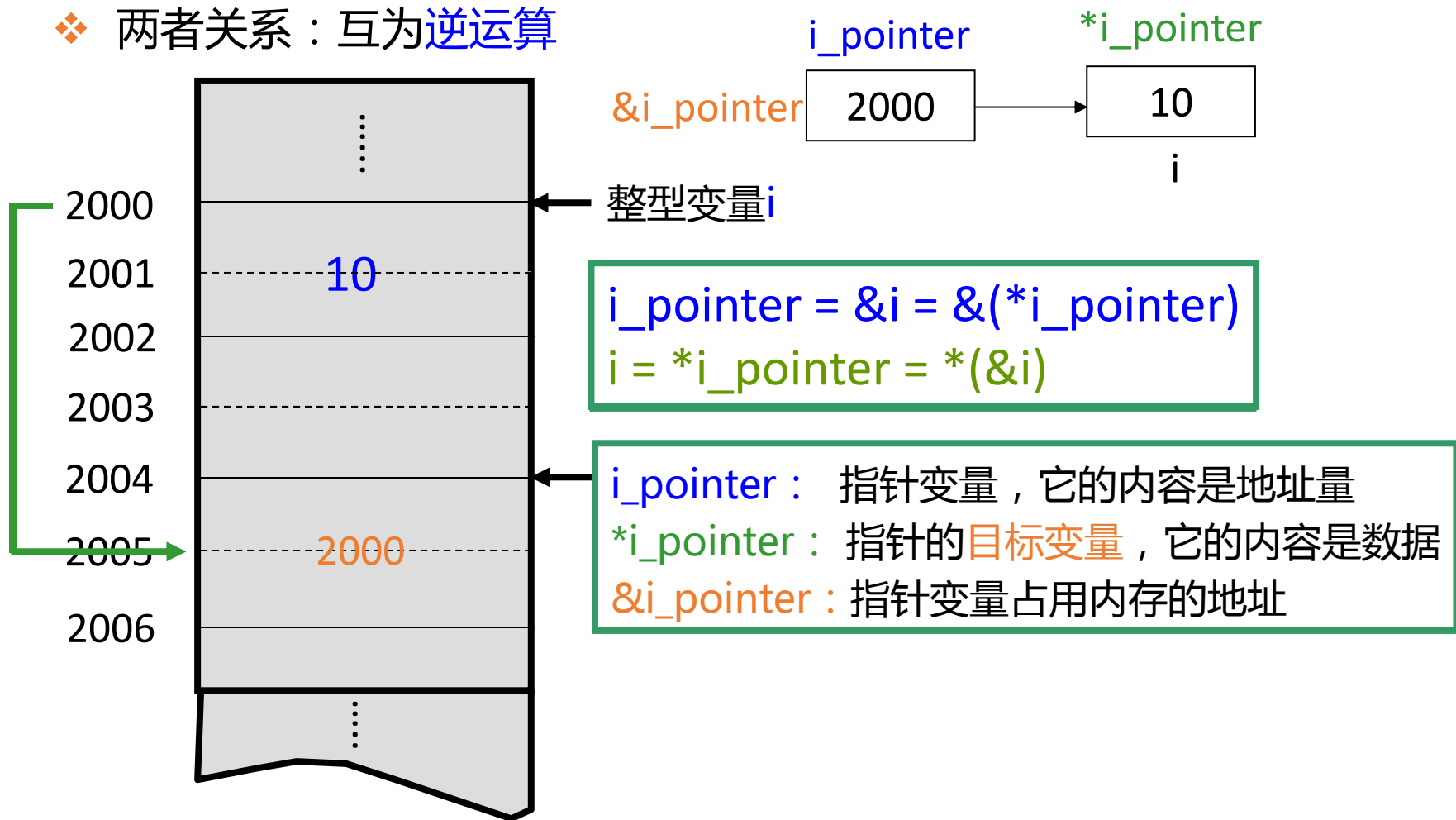
不能用auto变量的地址
去初始化static型指针

8.4 指针变量的引用

含义: 取变量的地址
& 单目运算符
优先级: 2
结合性: 自右向左

❖ 两者关系: 互为逆运算

含义: 取指针所指向的变量
* 单目运算符
优先级: 2
结合性: 自右向左

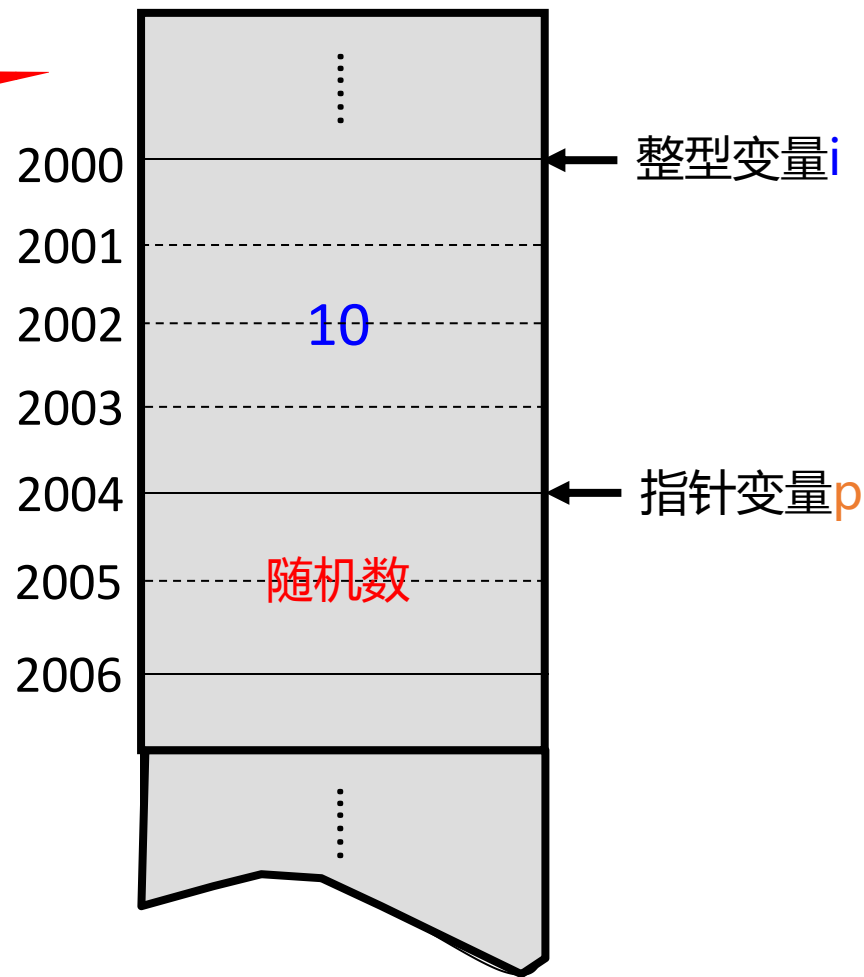


指针变量必须先赋值,再使用

```
例 main( )  
{ int i=10;  
  int *p;  
  *p=i;  
  printf("%d",*p);  
}
```

有危险！

```
例 main( )  
{ int i=10,k;  
  int *p;  
  p=&k;  
  *p=i;  
  printf("%d",*p);  
}
```



8.5 指针变量的运算

❖空指针（零指针）

```
#define NULL 0  
int *p=NULL;
```

表示指针变量的值为空
不指向任何变量或函数

注意：

- 1、“#define NULL 0”已在stdio.h中定义
- 2、除0以外的任何整数都不允许直接赋给指针变量
- 3、定义指针时将其初始化为NULL可以避免指向未知区域
- 4、保证指针未指向有效对象时保持NULL是良好的习惯

❖无效指针（invalid pointer）

- 定义指针变量后未赋值
- 将整型变量转换成指针
- 释放指针所指对象的存储空间
- 指针运算超出范围

【例8.1】分析程序运行结果

```
#include<stdio.h>
main()
{
    int *p, *q, x, y;
    q=&x;    *q=3;
    p=&y;    *p=*q;
    *p=5;
    p=q;    *p=8;
    p=NULL;
    p=100;
    printf("%d\n%d\n%d\n%d\n", *p, *q, x, y);
}
```

例 输入两个数，并使其从大到小输出

```
main()
{ int *p1,*p2,*p,a,b;
  scanf("%d,%d",&a,&b);
  p1=&a; p2=&b;
  if(a<b)
  { p=p1; p1=p2; p2=p;}
  printf("a=%d,b=%d\n",a,b);
  printf("max=%d,min=%d\n",*p1,*p2);
}
```

```
a=5,b=9
max=9,min=5
```

【例8.2】 按正向和反向顺序打印一个字符串

```
#include<stdio.h>
main()
{
    char *ptr1,*ptr2;
    ptr1="happy new year";
    ptr2=ptr1;
    while(*ptr2!='\0')
    .....
        putchar(*ptr2++);
    putchar('\n');
    while(--ptr2>=ptr1)
    .....
        putchar(*ptr2);
    putchar('\n');
}
```

8.6 指向数组的指针

```
例  int array[10];  
    int *p;  
    p=&array[0];    ⇔ p=array;  
或  int *p=&array[0];  
或  int *p=array;
```

数组名是表示数组**首地址**的**地址常量**

指针的运算

❖ 指针变量的赋值运算

- `p=&a;` (将变量a地址 \Rightarrow p)
- `p=array;` (将数组array首地址 \Rightarrow p)
- `p=&array[i];` (将数组元素地址 \Rightarrow p)
- `p1=p2;` (指针变量p2值 \Rightarrow p1)
- 不能把一个整数 \Rightarrow p,也不能把p的值 \Rightarrow 整型变量

❖ 指针的算术运算：

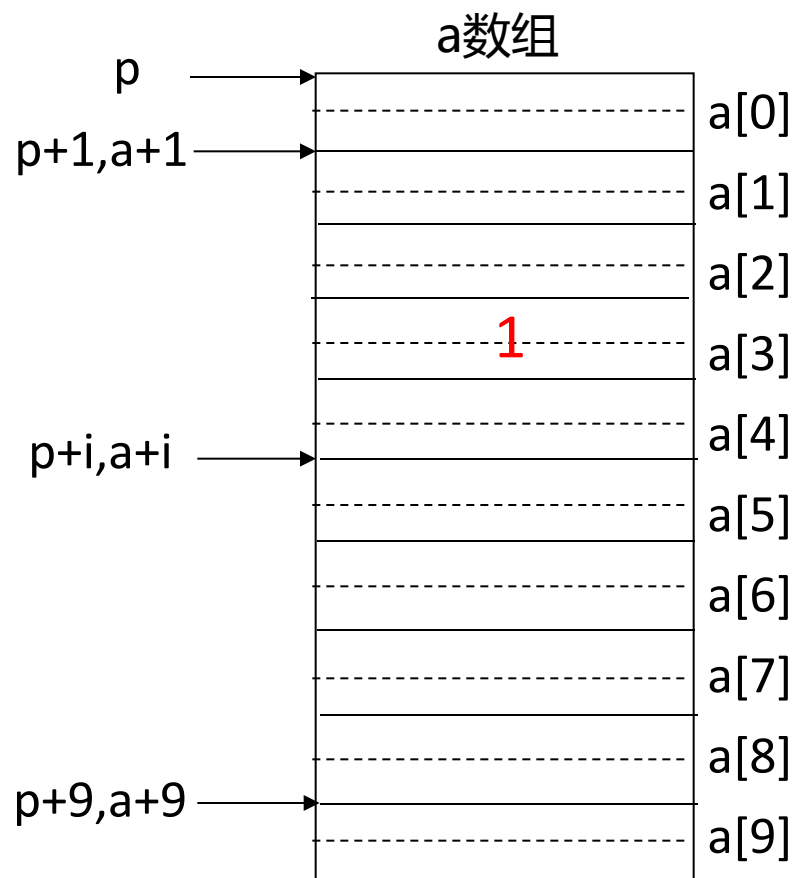
- $p \pm i \Leftrightarrow p \pm i \times d$ (i 为整型数， d 为 p 指向的变量所占字节数)
- $p++$, $p--$, $p+i$, $p-i$, $p+=i$, $p-=i$ 等
- 若 $p1$ 与 $p2$ 指向同一数组， $p1-p2$ =两指针间元素个数 $\Leftrightarrow (p1-p2)/d$
- $p1+p2$ 无意义

例 p 指向float数，则 $p+1 \Leftrightarrow p+1 \times 4$

例 p 指向int型数组，且 $p=\&a[0]$;
则 $p+1$ 指向 $a[1]$

例 `int a[10];`
`int *p=&a[2];`
`p++;`
`*p=1;`

例 `int a[10];`
`int *p1=&a[2];`
`int *p2=&a[5];`
则 $p2-p1=3$;



❖ 指针变量的关系运算

- 若p1和p2指向同一数组，则
 - ◆ $p1 < p2$ 表示p1指的元素在前
 - ◆ $p1 > p2$ 表示p1指的元素在后
 - ◆ $p1 == p2$ 表示p1与p2指向同一元素
- 若p1与p2不指向同一数组，比较无意义
- $p == NULL$ 或 $p != NULL$

数组元素表示方法

[] 变址运算符
 $a[i] \Leftrightarrow *(a+i)$

地址

a	a[0]
a+1	a[1]
a+2	a[2]
	a[3]
	⋮
a+9	a[9]

元素

a[0] *a
a[1] *(a+1)
a[2] *(a+2)

a[9] *(a+9)

地址

p	a[0]
p+1	a[1]
p+2	a[2]
	a[3]
	⋮
p+9	a[9]

元素

*p p[0]
*(p+1) p[1]
*(p+2) p[2]

*(p+9) p[9]

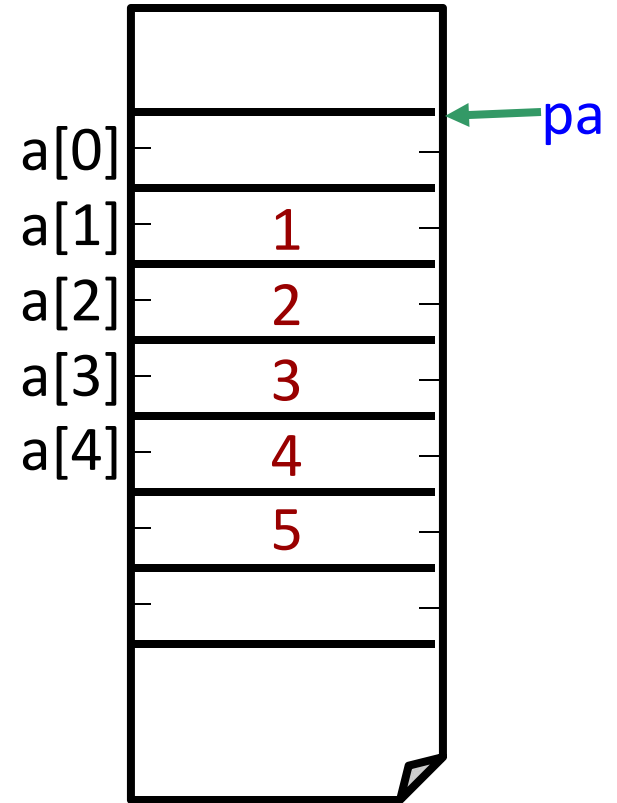
下标法

指针法

$a[i] \Leftrightarrow p[i] \Leftrightarrow *(p+i) \Leftrightarrow *(a+i)$

例 数组元素的引用方法

```
main()
{   int a[5],*pa,i;
    for(i=0;i<5;i++)
        a[i]=i+1;
    pa=a;
    for(i=0;i<5;i++)
        printf("*(pa+%d):%d\n",i,*(pa+i));
    for(i=0;i<5;i++)
        printf("*(a+%d):%d\n",i,*(a+i));
    for(i=0;i<5;i++)
        printf("pa[%d]:%d\n",i,pa[i]);
    for(i=0;i<5;i++)
        printf("a[%d]:%d\n",i,a[i]);
}
```



【例8.4】 编写字符串复制函数函数udf_strcpy()

```
#include<stdio.h>
void udf_strcpy1(char s[],char t[])
{
    int i=0;
    while((s[i]=t[i])!='\0')
        i++;
}
main()
{
    char s[15]="abc",t[]="def";
    udf_strcpy1(s,t);
    puts(s);
}
```

```
void udf_strcpy2(char *s,char *t)
{
    while((*s=*t)!='\0'){
        s++; t++;
    }
}
```

```
void udf_strcpy3(char *s,char *t)
{
    while((*s++=*t++)!='\0');
}
```

```
void udf_strcpy4(char *s,char *t)
{
    while(*s++=*t++);
}
```

【例8.5】 反置数组（递归）

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void invert(char *s,int i,int j);
int udf_strlen(char *s);
main()
{
    char str[80];
    int i=0,j;
    gets(str); puts(str);
    j=udf_strlen(str);
    printf("%d\n",j);
    invert(str,i,j-1);
    puts(str);
}
void invert(char *s,int i,int j)
{
    char t;
    if(i<j)
    {
        t=*(s+i);
        *(s+i)=*(s+j);
        *(s+j)=t;

        invert(s,i+1,j-1);
    }
}
int udf_strlen(char *s)
{
    int i=0;
    while(*s++)i++;
    return(i);
}
```

```
int a[10], *iptr, *q, u, v;  
v=5;  
iptr=&a[v];  
a[4]=40;  
a[5]=50;  
a[6]=60;
```

iptr++

*(iptr++)

(*iptr)++

*iptr++

++iptr

*(++iptr)

*++iptr

例 将数组a中的n个整数按相反顺序存放

```
void inv(int *x, int n)
```

形参用指针变量

```
{ int t,*p,*i,*j,m=(n-1)/2;  
  i=x; j=x+n-1; p=x+m;  
  for(;i<=p;i++,j--)  
  { t=*i; *i=*j; *j=t; }  
}
```

```
main()
```

```
{ int i,a[10]={3,7,0,11,0,6,7,5,4,2};
```

```
  inv(a,10);
```

实参用数组

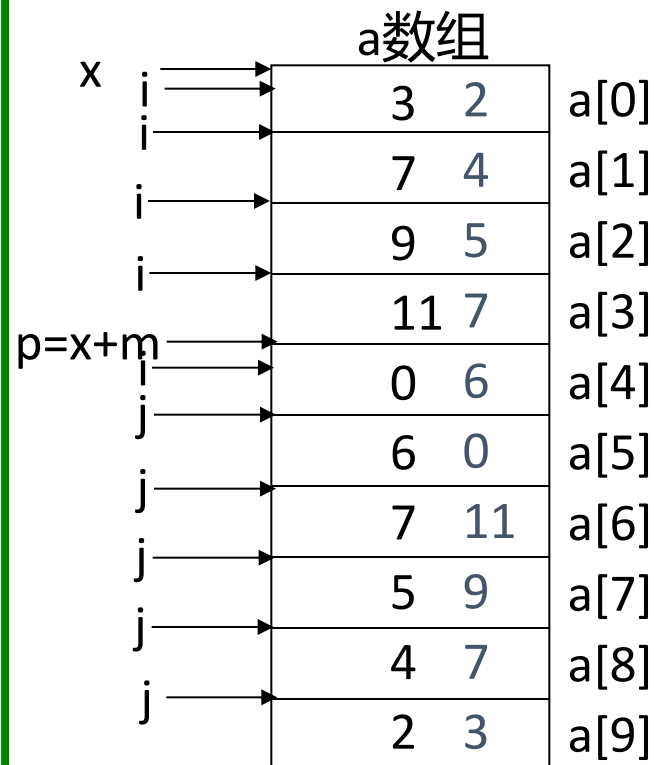
```
  printf("The array has been reverted:\n");
```

```
  for(i=0;i<10;i++)
```

```
    printf("%d,",a[i]);
```

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

```
}
```



例 将数组a中的n个整数按相反顺序存放

```
void inv(int *x, int n)
```

实参与形参均用指针变量

```
{ int t,*i,*j,*p,m=(n-1)/2;  
  i=x; j=x+n-1; p=x+m;  
  for(;i<=p;i++,j--)  
  { t=*i; *i=*j; *j=t; }  
}
```

```
main()
```

```
{ int i,a[10],*p=a;  
  for(i=0;i<10;i++,p++)  
    scanf("%d",p);  
  p=a; inv(p,10);
```

实参与形参均用指针变量

```
  printf("The array has been reverted:\n");  
  for(p=a;p<a+10;p++)  
    printf("%d",*p);  
}_x001A_
```


例 将数组a中的n个整数按相反顺序存放

```
void inv(int x[], int n)
{
    int t,i,j,m=(n-1)/2;
    for(i=0;i<=m;i++)
    {
        j=n-1-i;
        t=x[i]; x[i]=x[j]; x[j]=t;
    }
}

main()
{
    int i,a[10],*p=a;
    for(i=0;i<10;i++,p++)
        scanf("%d",p);
    p=a;    inv(p,10);
    printf("The array has been reverted:\n");
    for(p=a;p<a+10;p++)
        printf("%d ",*p);
}

_x001A_
```

形参用数组

实参用指针变量

❖ 一级指针变量与一维数组的关系

`int *p` 与 `int q[10]`

⑩ 数组名是指针（地址）常量

⑩ `p=q`; `p+i` 是`q[i]`的地址

⑩ 数组元素的表示方法:下标法和指针法,即若`p=q`,则 $p[i] \Leftrightarrow q[i] \Leftrightarrow *(p+i) \Leftrightarrow *(q+i)$

⑩ 形参数组实质上是指针变量,即`int q[]` \Leftrightarrow `int *q`

⑩ 在定义指针变量（不是形参）时,不能把`int *p` 写成`int p[]`;

⑩ 系统只给`p`分配能保存一个指针值的内存区(一般2字节) ; 而给`q`分配`2*10`字节的内存区

例 int a[]={1,2,3,4,5,6,7,8,9,10},*p=a,i;

数组元素地址的正确表示：

(A) &(a+1) (B) a++ (C) &p (~~D~~) &p[i]

数组名是地址常量

p++,p-- (✓)

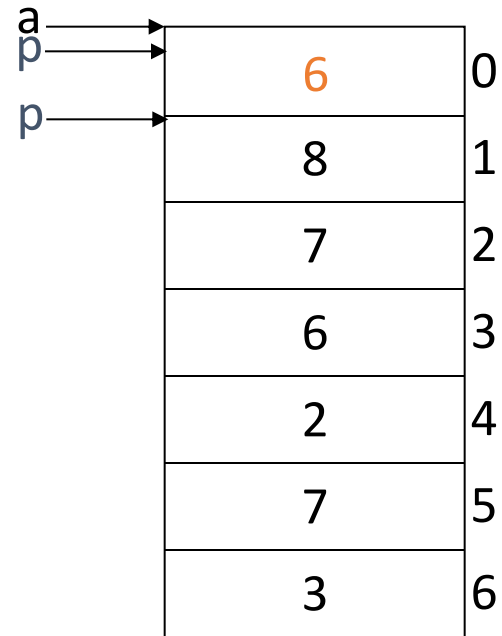
a++,a-- (✗)

a+1, *(a+2) (✓)

例 注意指针变量的运算

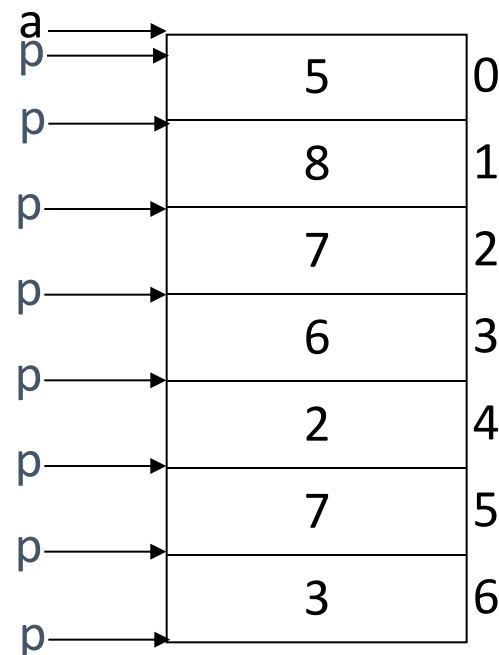
```
void main()
{  int a[]={5,8,7,6,2,7,3};
   int y,*p=&a[1];
   y=(*--p)++;
   printf("%d ",y);
   printf("%d",a[0]);
}
```

输出：5 6



例 注意指针的当前值

```
main()
{  int i,*p,a[7];
   p=a;
   for(i=0;i<7;i++)
       scanf("%d",p++);
   printf("\n");
   p=a;
   for(i=0;i<7;i++,p++)
       printf("%d",*p);
}
```



指针变量可以指到数组后的内存单元

8.7 指针作为函数参数

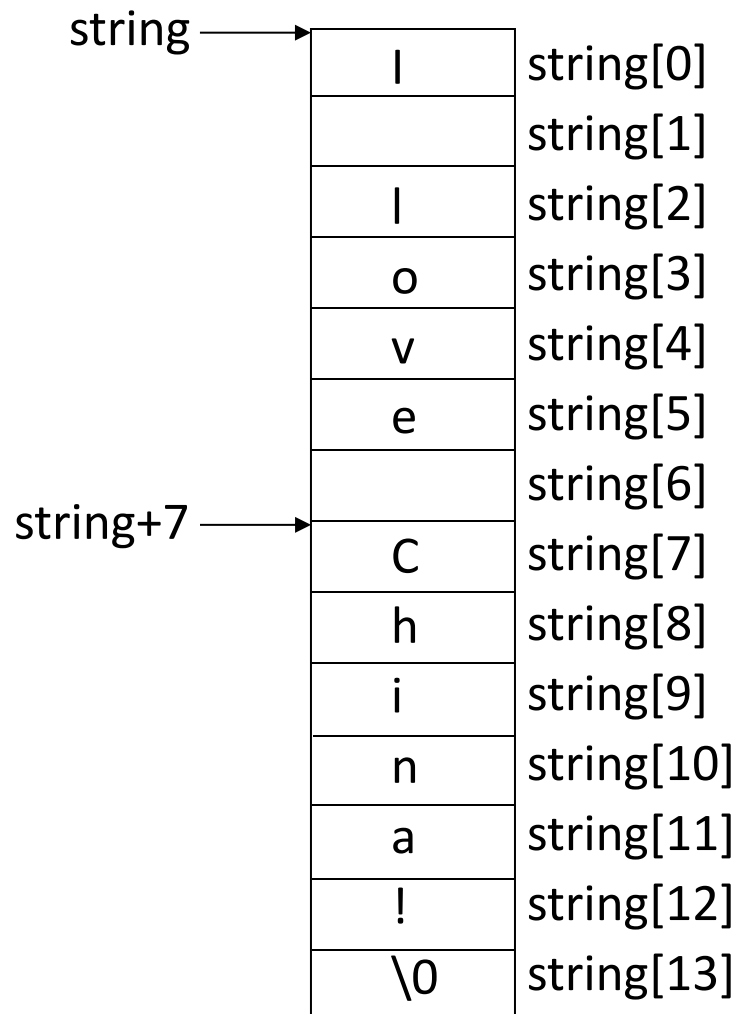
【例8.8】 交换变量函数

```
#include<stdio.h>
void swap3(int *px,int *py)
{
    int *pt;
    pt=px;px=py;py=pt;
    printf("*px=%d,*py=%d\n",*px,*py);
}
main()
{
    int x=1,y=3,*p1=&x,*p2=&y;
    swap3(p1,p2);
    printf("x=%d,y=%d,*p1=%d,*p2=%d\n",x,y,*p1,*p2);
}
```

8.8 指向字符串的指针变量

★ 字符串表示形式：用字符数组实现

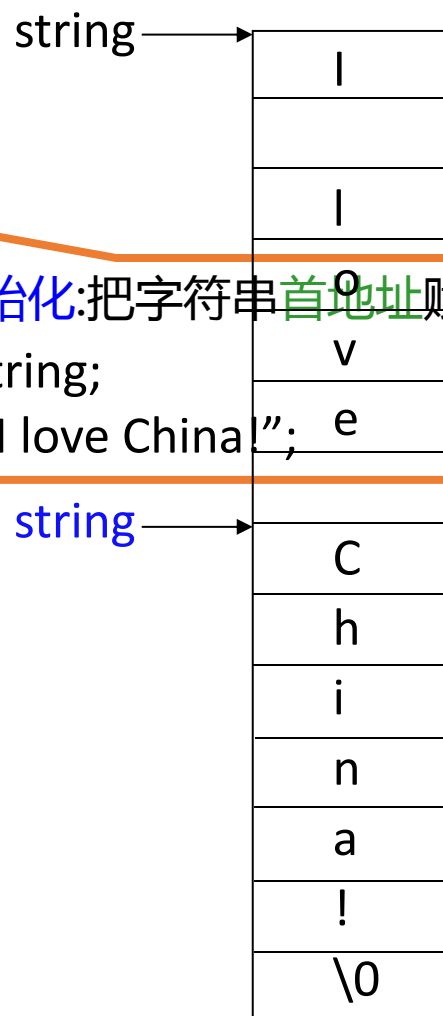
```
main( )  
{ char string[]="I love China!";  
  printf("%s\n",string);  
  printf("%s\n",string+7);  
}
```



★ 字符串表示形式：用**字符指针**实现

```
main( )
{ char *string="I love China!";
  printf("%s\n",string);
  string+=7;
  while(*string)
  {
    putchar(string[0]);
    string++;
  }
}
```

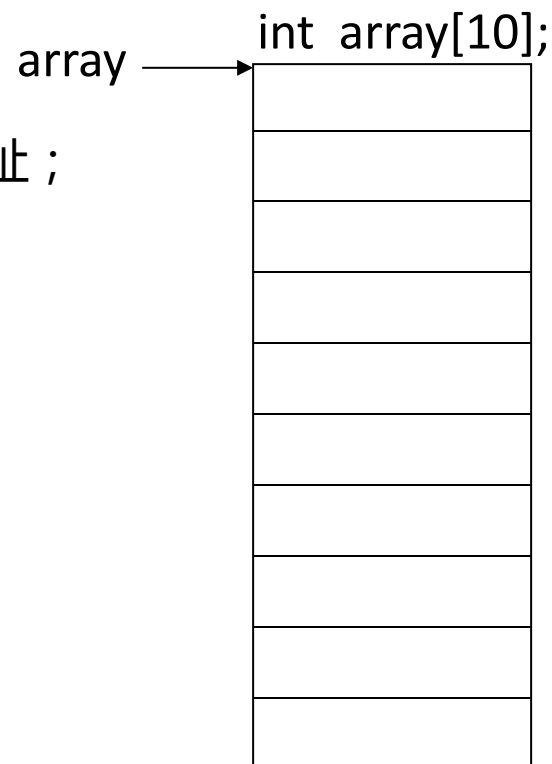
字符指针**初始化**:把字符串**首地址**赋给string
⇔ char *string;
string="I love China!";



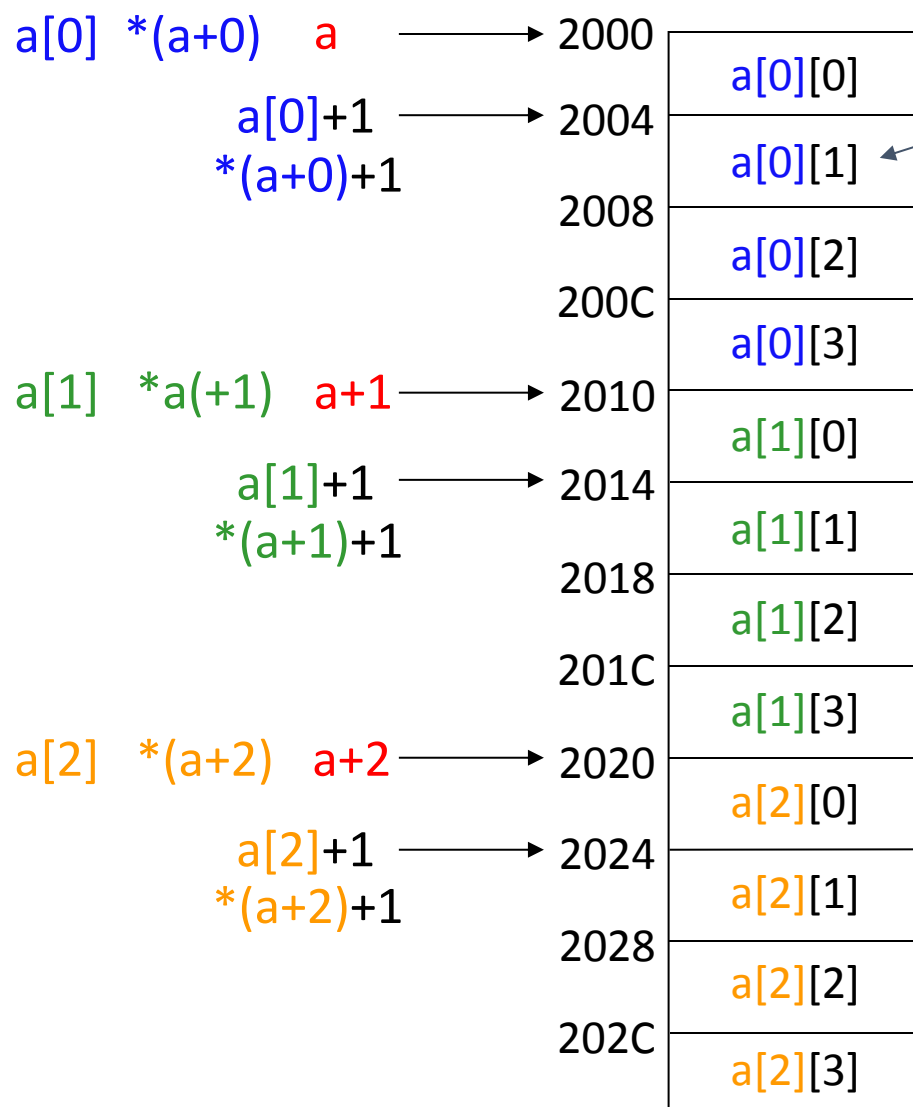
8.9 指向多维数组的指针变量

对于一维数组:

- 1、数组名array表示数组的首地址，即array[0]的地址；
- 2、数组名array是地址常量
- 3、array+i是元素array[i]的地址
- 4、array[i] \Leftrightarrow *(array+i)



二维数组：行指针和列指针 `int a[3][4];`



`*(a[0]+1)`

`*(*(a+0)+1)`

对于二维数组：

(1) `a`、`a+1`、`a+2`是行地址

`a[0]`、`a[1]`、`a[2]`是列地址

(2) `a[0]`、`a[1]`、`a[2]`等同于

`*(a+0)`、`*(a+1)`、`*(a+2)`

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

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]

对二维数组 `int a[3][4]`,有 (从第0行第0列算起)

- ❖ `a`-----二维数组的首地址, 即第0行的首地址
- ❖ `a+i`-----第*i*行的首地址
- ❖ `a[i] ⇔ *(a+i)`-----第*i*行第0列的元素地址
- ❖ `a[i]+j ⇔ *(a+i)+j` -----第*i*行第*j*列的元素地址
- ❖ `*(a[i]+j) ⇔ *(*a+i)+j ⇔ a[i][j]`

`a+i = &a[i] = a[i] = *(a+i) = &a[i][0]`, 地址编号相等, 含义不同

`a+i ⇔ &a[i]`, 表示第*i*行首地址, 指向行

`a[i] ⇔ *(a+i) ⇔ &a[i][0]`, 表示第*i*行第0列元素地址, 指向列

int a[3][4];

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]

地址表示：

- (1) a+1 ← 行地址
- (2) &a[1][0]
- (3) a[1]
- (4) *(a+1)
- } 列地址

地址表示：

- (1) &a[1][2]
- (2) a[1]+2
- (3) *(a+1)+2
- (4) &a[0][0]+1*4+2

二维数组元素表示形式：

- (1) a[1][2]
- (2) *(a[1]+2)
- (3) *(*a+1)+2
- (4) *(&a[0][0]+1*4+2)

❖ 指向二维数组的指针变量

```
main()
{ static int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
  int *p;
  for(p=a[0];p<a[0]+12;p++)
  {
    if((p-a[0])%4==0) printf("\n");
    printf("%4d ",*p);
  }
}
```

p=*a;	✓
p=&a[0][0];	✓
p=*(a+0);	✓
p=a;	✗

p →	int a[3][4];
	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]

指向一维数组的指针变量 (行指针)

❖ 定义形式 数据类型 (*指针名)[一维数组长度];

例 `int (*p)[4];`

❖ 可让p指向二维数组某一行

如 `int a[3][4], (*p)[4];` p的值是一维数组的首地址, p是行指针
`int (*p)[4]`与`int *p[4]`不同

一维数组指针变量长度和
二维数组列数必须相同

`int a[3][4];`

a

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]

p

p[0]+1或 *p+1

*(p+1)或 (*p)[1]

p+1

p[1]+2或 *(p+1)+2

*(p+1)+2

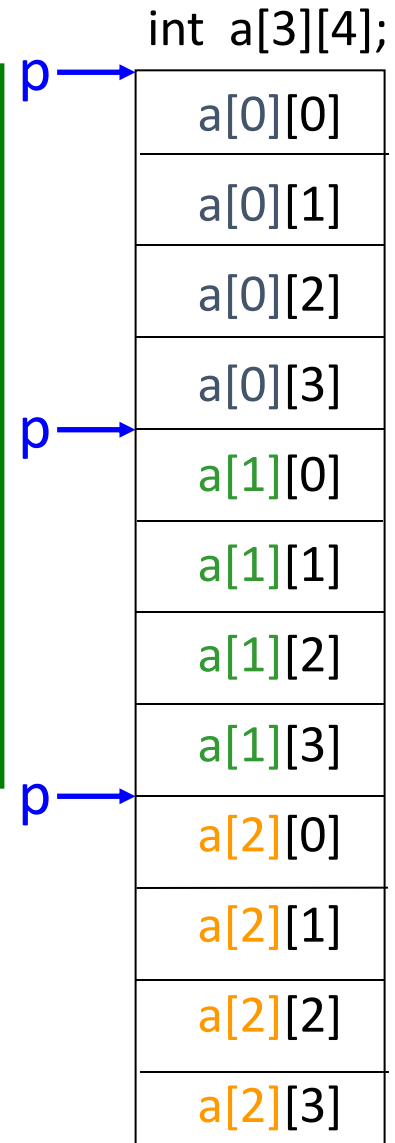
p+2

a+2

例 一维数组指针变量举例

```
main()
{ static int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
  int i,j,(*p)[4];
  for(p=a,i=0;i<3;i++,p++)
    for(j=0;j<4;j++)
      printf("%d ",*(*p+j));  ⇔ p[0][j]
  printf("\n");
}
```

p=a[0];	✗
p=*a;	✗
p=&a[0][0];	✗
p=&a[0];	✓



```

#include<stdio.h>
main()
{ int a[3][4]={{1,2,3,4},{3,4,5,6},{5,6,7,8}};
  int i,j;
  int (*p)[4]=a,*q=a[0];
  for(i=0;i<3;i++)
  { if(i%2==0)
      (*p)[i/2+1]=*q+3;
    else
      p++,++q;
  }
  for(i=0;i<3;i++){
      for(j=0;j<4;j++)
          printf("%4d",a[i][j]);
          putchar('\n');
      }
  printf("%d,%d\n",**p,*q);
}

```

	1	4	3	4
	3	4	7	6
	5	6	7	8

❖ 二维数组的指针做函数参数

- 用指向变量的指针变量
- 用指向一维数组的指针变量
- 用二维数组名

若 `int a[3][4]; int (*p1)[4]=a; int *p2=a[0];`

实参	形参
数组名a	数组名int x[][4]
数组名a	指针变量int (*q)[4]
指针变量p1	数组名int x[][4]
指针变量p1	指针变量int (*q)[4]
指针变量p2	指针变量int *q

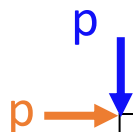
例 3个学生各学4门课，计算总平均分，并输出第n个学生成绩

函数说明

```
main()
{ void average(float *p,int n);
  void search(float (*p)[4],int n);
  float score[3][4]=
  {{65,67,79,60},{80,87,90,81},
   {90,99,100,98}};
  average(*score,12);
  search(score,2);
}
```

列地址

行地址



65	52	79	60
80	87	90	81
90	99	100	98

```
void average(float *p,int n)
{ float *p_end, sum=0,aver;
  p_end=p+n-1;
  for(;p<=p_end;p++)
      sum=sum+(*p);
  aver=sum/n;
  printf("average=%5.2f\n",aver);
}

void search(float (*p)[4], int n)
{ int i;
  printf(" No.%d :\n",n);
  for(i=0;i<4;i++)
      printf("%5.2f ",*(*(p+n)+i));
}
```

float p[][4]

p[n][j]

例 3个学生各学4门课，计算总平均分，并查找一门以上课不及格学生，输出其各门课成绩

```
void search(float (*p)[4], int n)
```

```
{ int i,j,flag;
```

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

```
  { flag=0;
```

```
    for(i=0;i<4;i++)
```

```
      if(*(*p+j)+i)<60) flag=1;
```

```
    if(flag==1)
```

```
    { printf("No.%d is fail, his scores are:\n",j+1);
```

```
      for(i=0;i<4;i++)
```

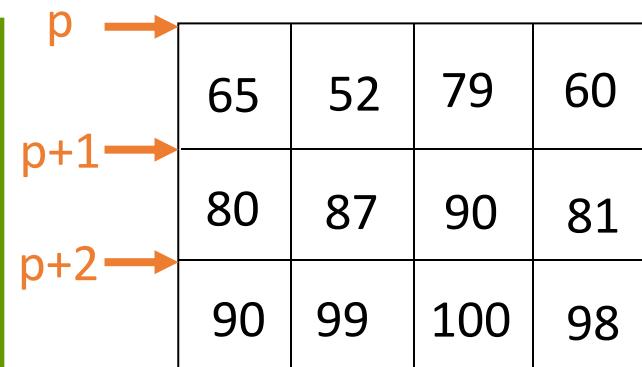
```
        printf("%5.1f ", *(*p+j)+i);
```

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

```
    }
```

```
  }
```

```
}
```



p →	65	52	79	60
p+1 →	80	87	90	81
p+2 →	90	99	100	98

p[j][i]

p[j][i]

```
main()
```

```
{ void search(float (*p)[4], int n);
```

```
  float score[3][4]={ {...}, {...}, {...}};
```

```
  search(score,3);
```

```
}
```

8.10 指针数组

- ❖ 定义：数组中的元素为指针变量
- ❖ 定义形式：`[存储类型] 数据类型 *数组名[数组长度] ;`

例 `static int *p[4];`

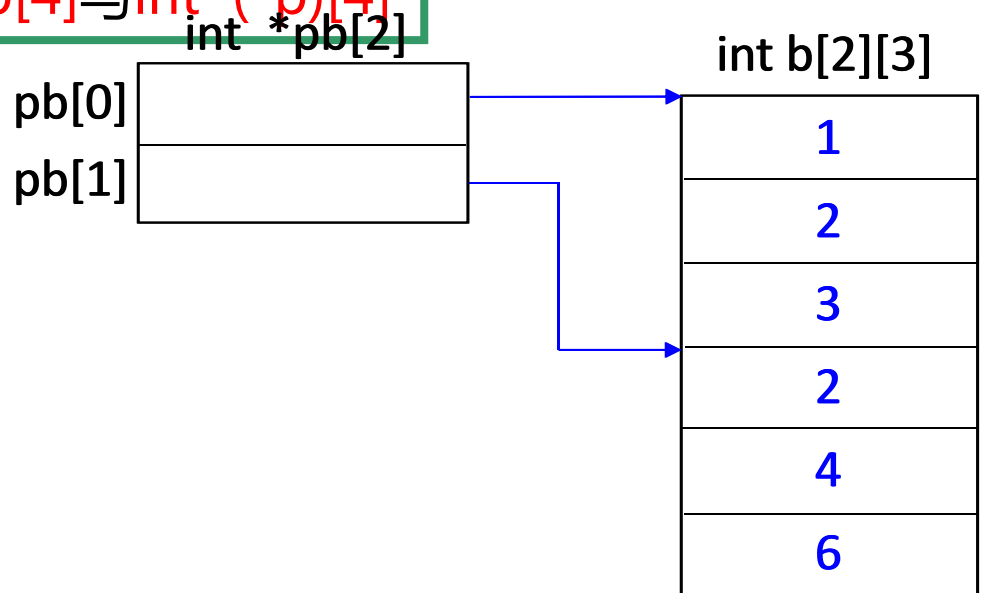
- ❖ 指针变量的赋值与初始化

指针本身的有 指针所指向变量的数据类型

区分 `int *p[4]` 与 `int (*p)[4]`

初始化:

```
main()
{
    int b[2][3];
    int *pb[]={b[0],b[1]};
    .....
}
```



❖ 指针变量的赋值与初始化

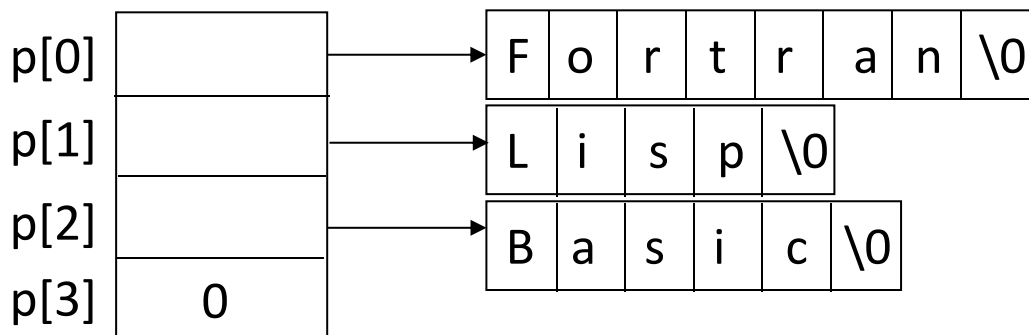
初始化:

```
main()
{ char *p[]={ "Fortran", "Lisp", "Basic", NULL};
  .....
}
```

```
char *p[4];  
p[0]=a; p[1]=b; p[2]=c; p[3]=NULL;  
.....
```

全:

```
main()
char *p[4];
p[0]= "Fortran";
p[1]= "Lisp";
p[2]= "Basic";
p[3]=NULL;
.....
```



❖ 二维字符串数组与指向字符串的指针数组区别：

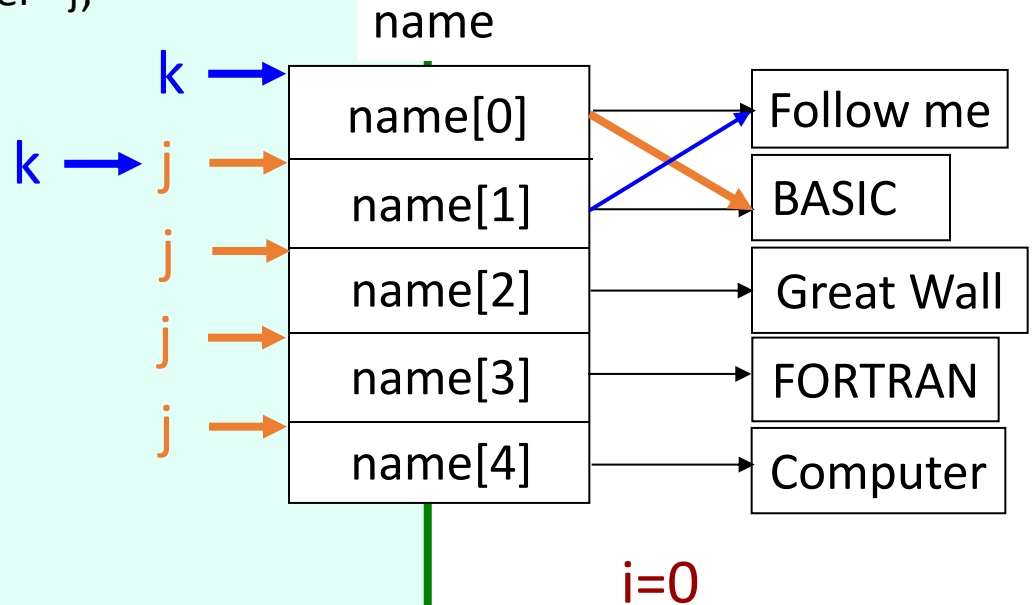
```
char name[5][9]={“gain”,“much”,“stronger”, “point”,“bye”};
```

```
char *name[5]={“gain”,“much”,“stronger”, “point”,“bye”};
```

- 二维数组字符串数组存储空间固定，指向字符串的指针数组相当于每行长度不同的二维数组。
- 指向字符串的指针数组元素的作用相当于二维数组的行名，但指针数组中元素是指针变量，二维数组行名是地址常量

例 对字符串排序 (简单选择排序)

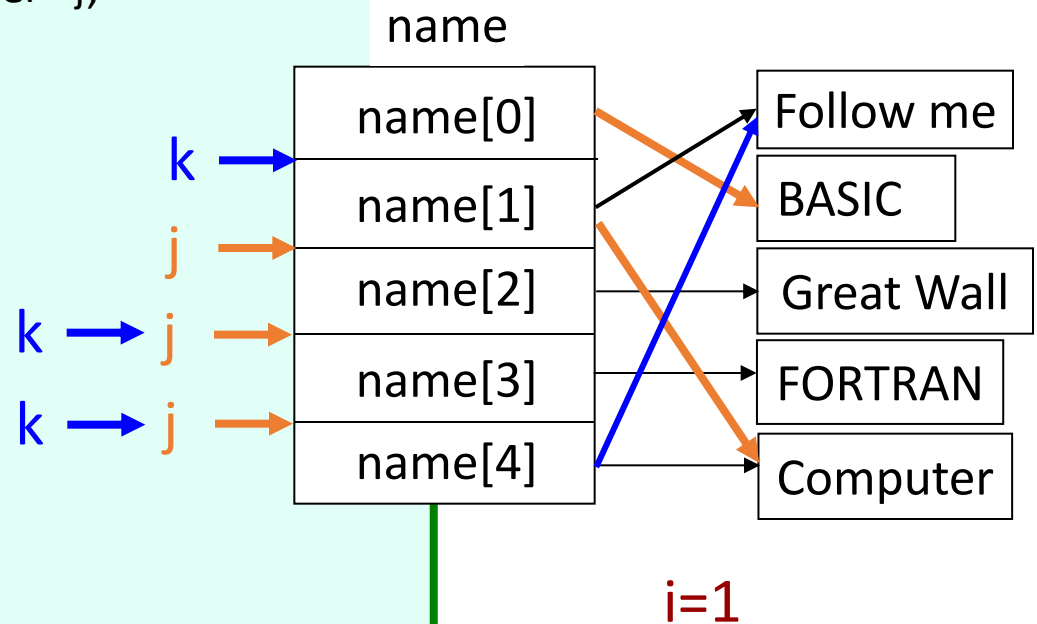
```
main()
{ void sort(char *name[],int n), print(char *name[],int n);
  char *name[]={"Follow me","BASIC",
    "Great Wall","FORTRAN","Computer "};
  int n=5;
  sort(name,n);
  print(name,n);
}
void sort(char *name[],int n)
{ char *temp;
  int i,j,k;
  for(i=0;i<n-1;i++)
  { k=i;
    for(j=i+1;j<n;j++)
      if(strcmp(name[k],name[j])>0) k=j;
    if(k!=i)
      { temp=name[i]; name[i]=name[k]; name[k]=temp;}
  }
}
```



例 对字符串排序 (简单选择排序)

```
main()
{ void sort(char *name[],int n), print(char *name[],int n);
  char *name[]={"Follow me","BASIC",
    "Great Wall","FORTRAN","Computer "};
  int n=5;
  sort(name,n);
  print(name,n);
}

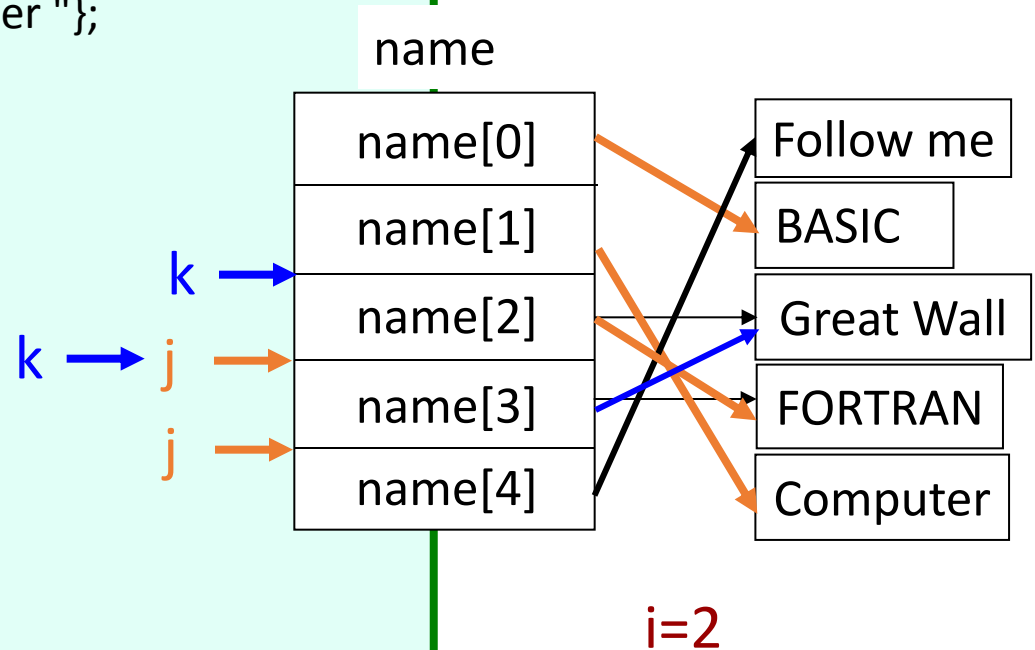
void sort(char *name[],int n)
{ char *temp;
  int i,j,k;
  for(i=0;i<n-1;i++)
  { k=i;
    for(j=i+1;j<n;j++)
      if(strcmp(name[k],name[j])>0) k=j;
    if(k!=i)
      { temp=name[i]; name[i]=name[k]; name[k]=temp;}
  }
}
```



例 对字符串排序 (简单选择排序)

```
main()
{ void sort(char *name[],int n), print(char *name[],int n);
  char *name[]={"Follow me","BASIC",
    "Great Wall","FORTRAN","Computer "};
  int n=5;
  sort(name,n);
  print(name,n);
}

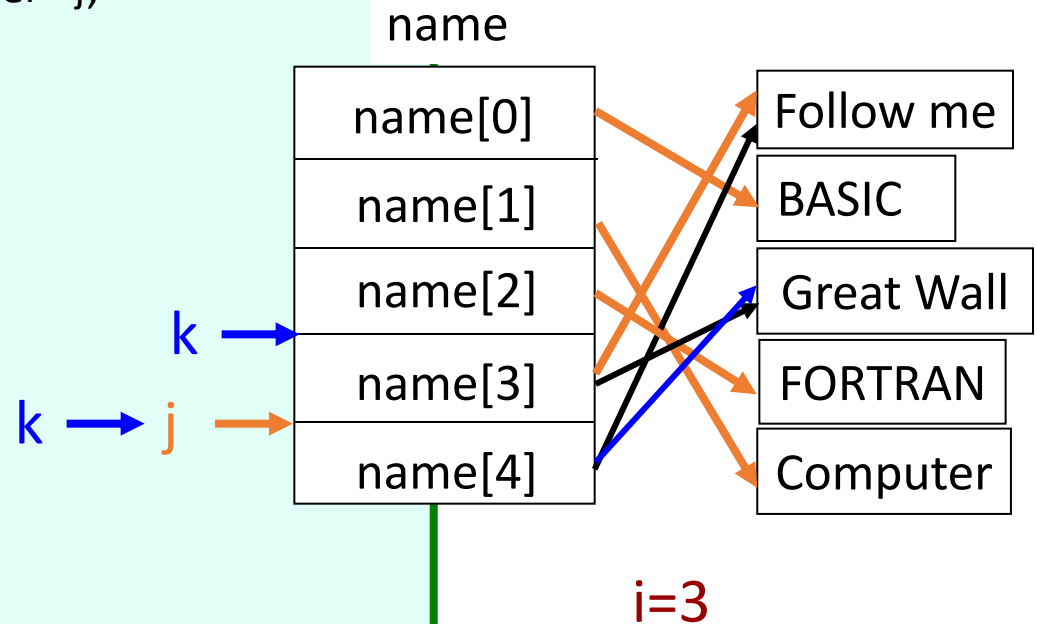
void sort(char *name[],int n)
{ char *temp;
  int i,j,k;
  for(i=0;i<n-1;i++)
  { k=i;
    for(j=i+1;j<n;j++)
      if(strcmp(name[k],name[j])>0) k=j;
    if(k!=i)
      { temp=name[i]; name[i]=name[k]; name[k]=temp;}
  }
}
```



例 对字符串排序（简单选择排序）

```
main()
{ void sort(char *name[],int n), print(char *name[],int n);
  char *name[]={"Follow me","BASIC",
    "Great Wall","FORTRAN","Computer "};
  int n=5;
  sort(name,n);
  print(name,n);
}

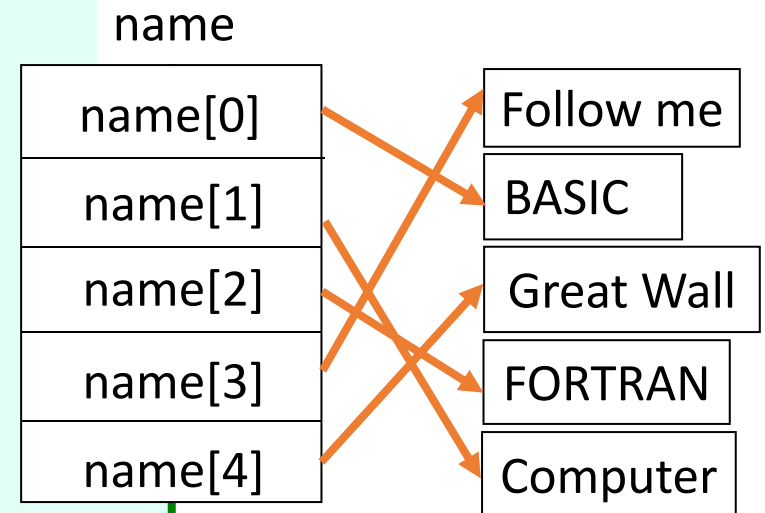
void sort(char *name[],int n)
{ char *temp;
  int i,j,k;
  for(i=0;i<n-1;i++)
  { k=i;
    for(j=i+1;j<n;j++)
      if(strcmp(name[k],name[j])>0) k=j;
    if(k!=i)
      { temp=name[i]; name[i]=name[k]; name[k]=temp;}
  }
}
```



例 对字符串排序（简单选择排序）

```
main()
{ void sort(char *name[],int n), print(char *name[],int n);
  char *name[]={"Follow me","BASIC",
    "Great Wall","FORTRAN","Computer "};
  int n=5;
  sort(name,n);
  print(name,n);
}

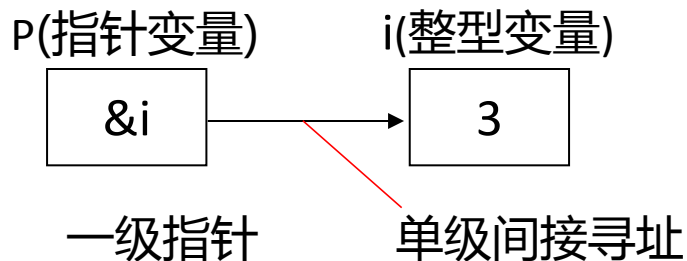
void sort(char *name[],int n)
{ char *temp;
  int i,j,k;
  for(i=0;i<n-1;i++)
  { k=i;
    for(j=i+1;j<n;j++)
      if(strcmp(name[k],name[j])>0) k=j;
    if(k!=i)
      { temp=name[i]; name[i]=name[k]; name[k]=temp;}
  }
}
```



8.11 指向指针的指针

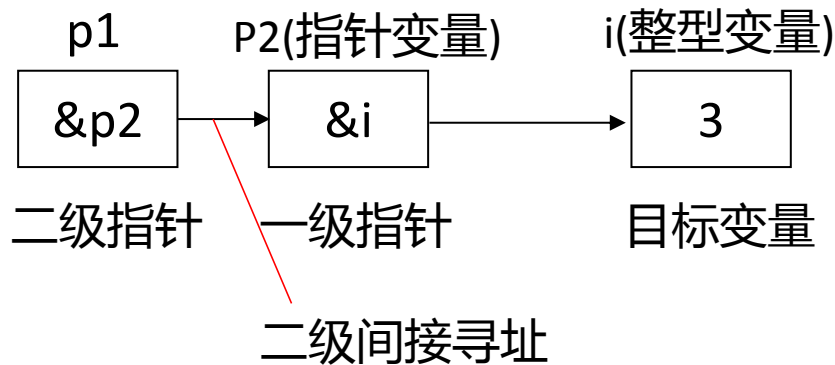
- ❖ 定义: 指向指针的指针
- ❖ 一级指针: 指针变量中存放目标变量的地址

```
例 int *p;  
    int i=3;  
    p=&i;  
    *p=5;
```



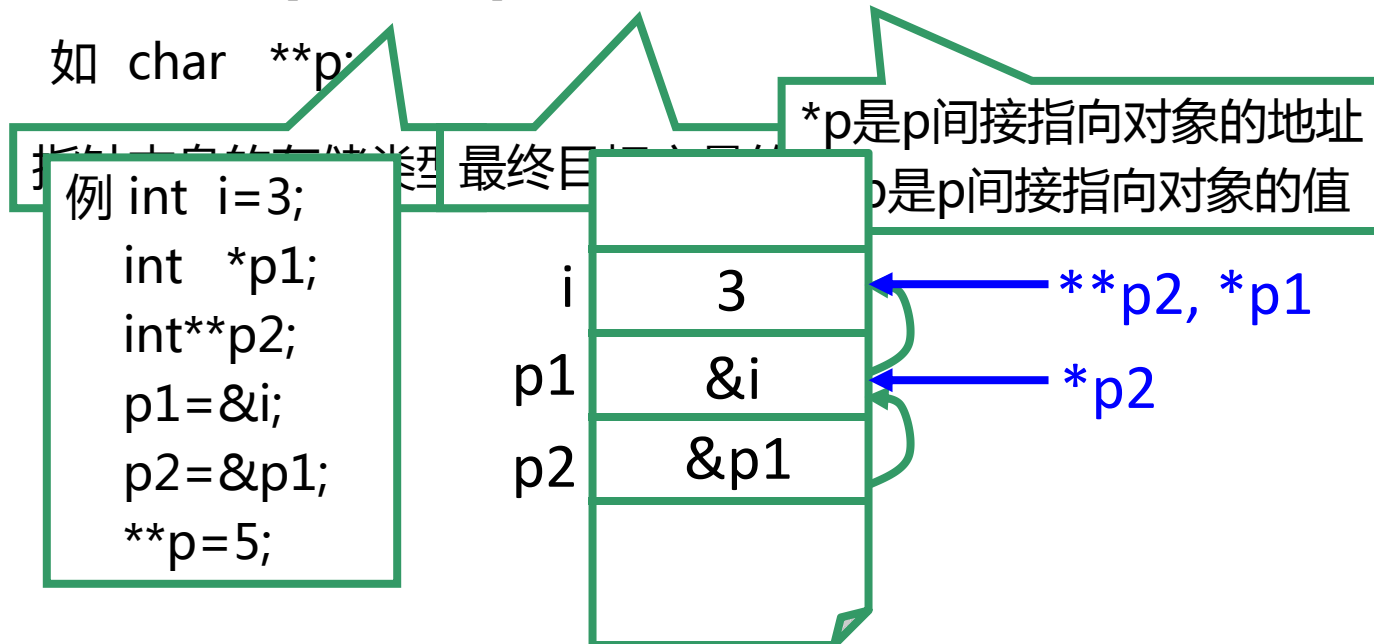
- ❖ 二级指针: 指针变量中存放一级指针变量的地址

```
例 int **p1;  
    int *p2;  
    int i=3;  
    p2=&i;  
    p1=&p2;  
    **p1=5;
```



- 定义形式：[存储类型] 数据类型 **指针

如 char **p;



例 int i, **p;

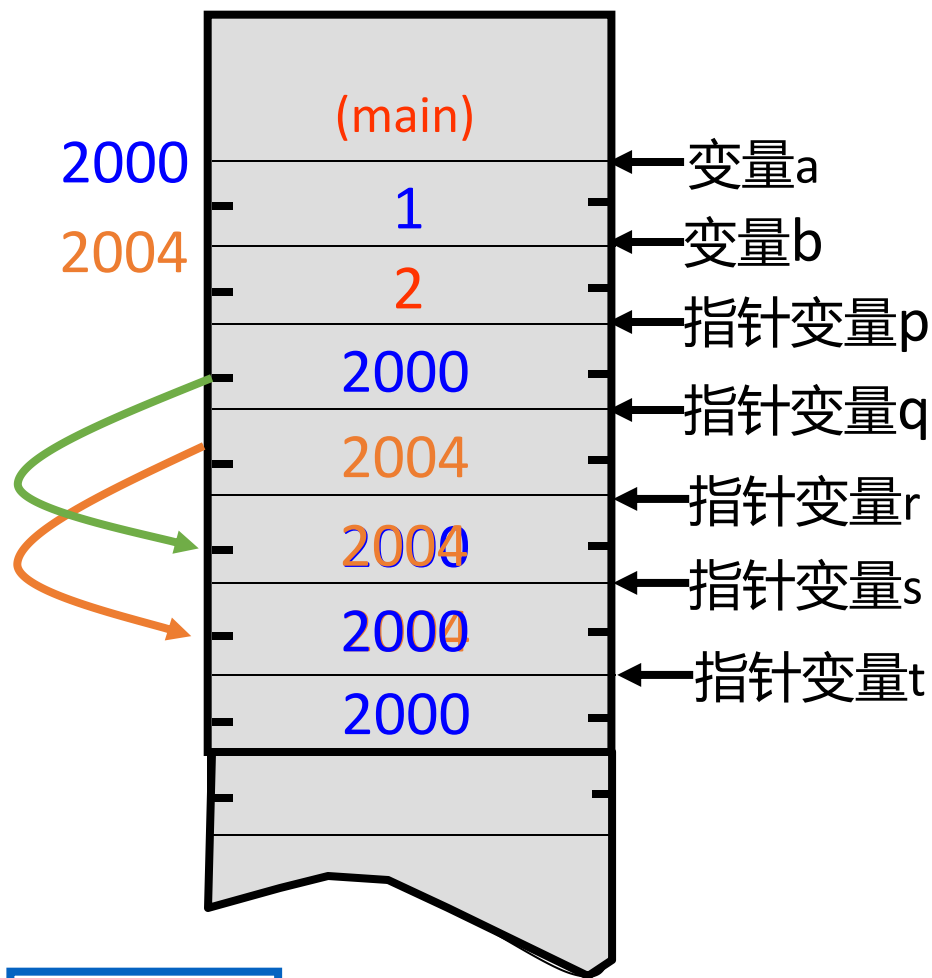
p=&i; ❌ p是二级指针，不能用变量地址为其赋值

❖ 多级指针

例 三级指针 int ***p;
 四级指针 char ****p;

例 一级指针与二级指针

```
#include <stdio.h>
void swap(int *r,int *s)
{ int *t;
  t=r;
  r=s;
  s=t;
}
main()
{ int a=1,b=2,*p,*q;
  p=&a;
  q=&b;
  swap(p,q);
  printf("%d,%d\n",*p,*q);
}
```

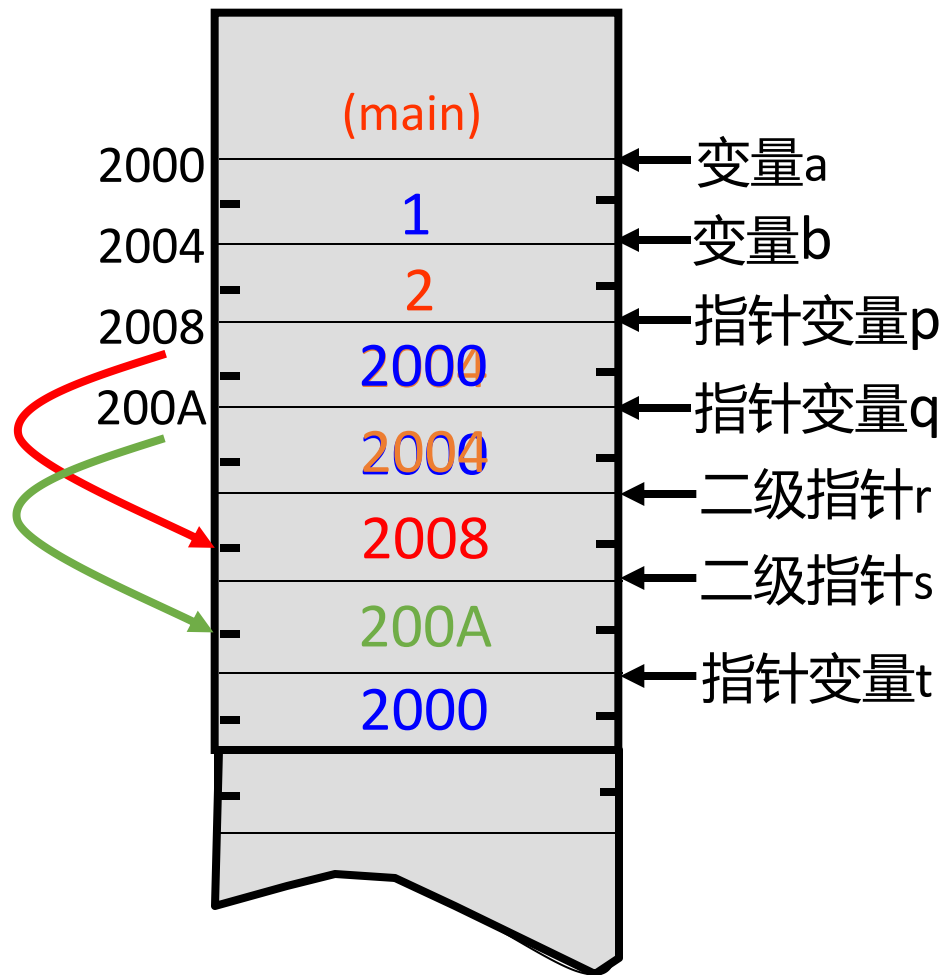


输出: 1,2

例 一级指针与二级指针

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

main()
{ int a=1,b=2,*p,*q;
  p=&a;
  q=&b;
  swap(&p,&q);
  printf("%d,%d\n",*p,*q);
}
```



输出: 2,1

例8.19 用指针数组和二级指针输出成绩

```
#include<stdio.h>
main()
{
    int stu1[]={78,79,73,-1},stu2[]={100,98,-1},stu3[]={88,-1};
    int stu4[]={96,78,33,65,-1},stu5[]={99,88,-1};
    int *grade[]={stu1,stu2,stu3,stu4,stu5};
    int **p=grade,i;
    for (i=1;i<=5;i++){
        printf("student%d grade:",i);
        while(**p>=0){
            printf("%4d",**p);
            (*p)++;
        }
        p++;
        printf("\n");
    }
}
```


8.12 指针数组作main函数的形参

命令行参数

- ❖ 命令行：在操作系统状态下，为执行某个程序而键入的一行字符
- ❖ 命令行一般形式：**命令名** 参数1 参数2.....参数n

```
E:\> copy[.exe] source.c temp.c
```

有3个字符串参数的命令行

- ❖ 带参数的main函数形式：

```
int main(int argc, char *argv[])  
{ .....  
}
```

命令行中参数个数

元素指向命令行参数
中各字符串首地址

注：1、形参名可以任意

2、形参char *argv[]本质上就是一个指向指针数组的**二级指针**，
所以通常也可以写成char **argv

- ❖ 命令行参数传递

系统自动调用
main函数时传递

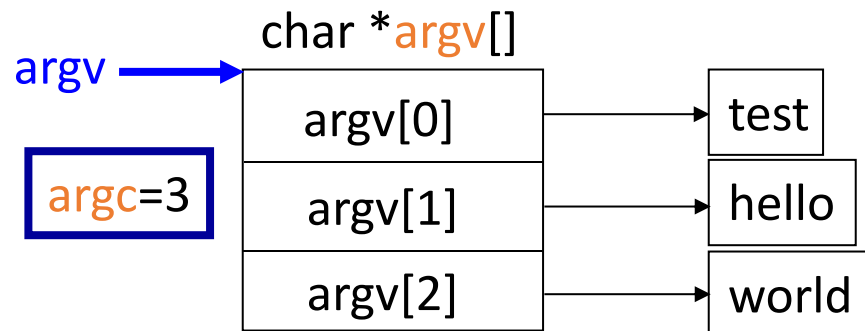
命令行**实参**

main(**形参**)

第一个参数: main所在的
可执行文件名

例 输出命令行参数

```
/*test.c*/  
main(int argc, char *argv[])  
{ while(argc>1)  
  { ++argv;  
    printf("%s\n",*argv);  
    --argc;  
  }  
}
```



1. 编译、链接test.c , 生成可执行文件test.exe
2. 在命令行状态下运行

例8.22 用命令行参数实现两个实数之和

```
#include<stdio.h>
#include<process.h>
#include<math.h>
main(int argc, char *argv[])
{
    double x,y;
    if(argc!=3){
        printf("using:command arg1,arg2<CR>\n");
        exit(1);
    }
    x=atof(argv[1]);
    y=atof(argv[2]);
    printf("sum=%f\n",x+y);
}
```

char **argv

8.13 指针函数

函数定义形式：类型标识符 *函数名(参数表)例

```
int *f(int x, int y)
```

例 用指针函数实现：有若干学生成绩，根据输入的序号输出其全部成绩

```
main()
{ float score[][4]={60,70,80,90},
    {56,89,67,88},{34,78,90,66}};
  float *search(float (*pointer)[4],int n), *p;
  int i,m;
  printf("Enter the number of student:");
  scanf("%d",&m);
  printf("The scores of No.%d are:\n",m);
  p=search(score,m);
  for(i=0;i<4;i++)
    printf("%5.2f\t",*(p+i));
}

float *search(float (*pointer)[4], int n)
{ float *pt;
  pt=*(pointer+n);
  return(pt);
}
```

score数组			
60	70	80	90
56	89	67	88
34	78	90	66

例8.23 用返回值和返回地址两种方法判断两个参数的大小

```
#include<stdio.h>
int larger1(int x,int y);
int *larger2(int *x,int *y);
main()
{
    int a,b,bigger1,*bigger2;
    printf("input 1st integer values:");
    scanf("%d",&a);
    printf("input 2nd integer values:");
    scanf("%d",&b);
    bigger1=larger1(a,b);
    printf("the larger value is %d\n",bigger1);
    bigger2=larger2(&a,&b);
    printf("the larger value is %d\n",*bigger2);
}
int larger1(int x,int y){
    return(x>y?x:y);
}
int *larger2(int *x,int *y){
    return(*x>*y?x:y);
}
```

```
main()
{
    int a=2,b=3;
    int *p;
    p=f3(a,b);
    printf("%d\n",*p);
}

int *f3(int x,int y)
{
    if(x>y)
        return &x;
    else
        return &y;
}
```

注意：不能将形参或局部变量的地址作为函数返回值

8.14 指向void量的指针变量

定义形式：`void *合法标识符`

`void *xp;`

- ❖ 其他类型指针（`char *`、`int *` 等）可以直接赋值给void指针
- ❖ void指针赋值给其他类型指针时，必须进行强制类型转换

```
int vi1,vi2,*ip;
```

```
float vf,*fp;
```

```
void *xp;
```

```
ip=&vi1;
```

```
fp=&vf;
```

```
ip=fp; ❌
```

```
fp=ip; ❌
```

```
xp=fp; ✓
```

```
xp=ip; ✓
```

```
ip=xp; ❌
```

```
ip=(int *)xp; ✓
```

- ❖ 当void型指针指向了具体的对象后，需要用强制类型转换的方式读取对象内容

```
vi2=*(int *)xp;
```

注：

- ❖ void型指针在运算时，`xp+1`表示指针移动一个字节
- ❖ void型指针经常用于编写通用函数

例8.25 用void指针实现 通用数据交换函数

```
#include<stdio.h>
void genswap(void *a,void *b,int size);
main()
{
    int m1=100,m2=200;
    double fx1=123.4,fx2=234.5;
    char str1[20]="Today is well day!";
    char str2[20]="今天是个好天气! ";
    genswap(&m1,&m2,sizeof(int));
    printf("m1=%d,m2=%d\n",m1,m2);
    genswap(&fx1,&fx2,sizeof(double));
    printf("fx1=%f,fx2=%f\n",fx1,fx2);
    genswap(str1,str2,sizeof(str1));
    printf("str1=%s,str2=%s\n",str1,str2);
}
void genswap(void *a,void *b,int size)
{
    char t;
    int i;
    for(i=0;i<size;i++){
        t=*((char *)a+i);
        *((char *)a+i)=*((char *)b+i);
        *((char *)b+i)=t;
    }
}
```

- 1、交换如何实现的？
- 2、为什么要将void指针强制转换成char？

8.15 指向函数的指针

函数首地址：函数在编译时被分配的入口地址,用函数名表示

指向函数的指针变量

❖ 定义形式：**数据类型** (***指针变量名**)();

`int (*p)();`
函数返回值的类型

❖ 函数指针变量赋值：如 `p=max;` `int (*p)()` 与 `int *p()` 不同
()不能省

❖ 函数调用形式：`c=max(a,b);` \Leftrightarrow `c=(*p)(a,b);`
函数指针变量指向的函数必须有函数说明
 \Leftrightarrow `c=p(a,b);`

❖ 对函数指针变量`p±n`, `p++`, `p--`无意义

`strcpy`

指令1

指令2

不同函数

⋮

例 用函数指针变量调用函数，比较两个数大小

```
#include<stdio.h>
main()
{
    int max(int,int),(*p)();
    int a,b,c;
    p=max;
    scanf("%d%d",&a,&b);
    c=(*p)(a,b);
    printf("a=%d,b=%d,max=%d\n",a,b,c);
}
int max(int x,int y){
    int z;
    if(x>y) z=x;
    else    z=y;
    return(z);
}
```

用函数指针变量作函数参数

例 用函数指针变量作参数，求最大值、最小值和两数之和

```
void main()
{ int a,b,max(int,int),
  min(int,int),add(int,int);
  void process(int,int,int (*fun)());
  scanf("%d,%d",&a,&b);
  process(a,b,max);
  process(a,b,min);
  process(a,b,add);
}

void process(int x,int y,int (*fun)())
{ int result;
  result=(*fun)(x,y);
  printf("%d\n",result);
}
```

```
max(int x,int y)
{ printf("max=");
  return(x>y?x:y);
}

min(int x,int y)
{ printf("min=");
  return(x<y?x:y);
}

add(int x,int y)
{ printf("sum=");
  return(x+y);
}
```

8.16 动态分配内存

“动态内存分配”的概念

使用户程序能在运行期间动态的申请和释放内存空间，从而更有效地利用内存并提高程序设计的灵活性。

例如，为了保证程序的通用性，程序运行时需要的最长为10000个的元素的数组保存，但大部分运行的时候只需要30个左右的元素，于是大量分配的存储空间被浪费。

此时，可通过动态内存分配技术，将程序设计成运行时才向计算机申请内存，并在用完时立即释放占用的内存空间（堆和栈的概念）。

动态内存分配函数

以下函数在`malloc.h`或`stdlib.h`中定义（`n,x`为无符号整数，`p`为指针变量）：

❖ `void *malloc(x)`

分配一个长度为`x`字节的连续空间，分配成功返回起始地址指针，分配失败（内存不足）返回NULL

❖ `void *calloc(n,x)`

分配`n`个长度为`x`字节的连续空间（成败结果同上）

❖ `void *realloc(p,x)`

将`p`所指的已分配空间大小调整为`x`个字节

❖ `void free(p)`

将由以上各函数申请的以`p`为首地址的内存空间全部释放

注：

- ❖ 无论何时分配内存，必须要检查是否分配成功，即使申请的空间很小。
- ❖ 申请的内存使用结束后用free（）函数释放，释放后应该把指向这块内存的指针指向NULL，防止程序后面不小心使用了它。
- ❖ 申请内存和释放内存这两个函数是配对函数，如果申请后不释放就是内存泄露。

```
int *pb;  
pb=(int *)malloc(n*sizeof(int));  
if(pb==NULL){  
    puts("memory allocation error.");  
    exit(1);  
}
```

例 8.30 编程实现选择排序，假定事先不知道要排序的元素个数

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
void MakeArray(int v[],int n);
void Sort(int *v,int n);
void PrintArray(int v[],int n);
int main()
{
    int n,*pb;
    printf("input number of element: ");
    scanf("%d",&n);
    pb=(int *)malloc(n*sizeof(int));
    if(pb==NULL){
        printf("error!");
        exit(1);
    }
    MakeArray(pb,n);
    printf("before sort:\n");
    PrintArray(pb,n);
    Sort(pb,n);
    printf("after sort:\n");
    PrintArray(pb,n);
    free(pb);
    return(0);
}
```

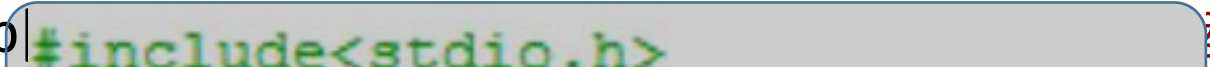
```
void MakeArray(int v[],int n){
    int i;
    srand(time(NULL));
    for(i=0;i<n;i++)
        v[i]=rand()%1000;
}
void PrintArray(int v[],int n){
    int i;
    for(i=0;i<n;i++)
        printf("%5d",v[i]);
    printf("\n");
}
void Sort(int *v,int n){
    int t,int i,j,k;
    for(i=0;i<n-1;i++){
        k=i;
        for(j=i+1;j<n;j++)
            if(v[k]>v[j]) k=j;
        if(k!=i){t=v[i];v[i]=v[k];v[k]=t;}
    }
}
```

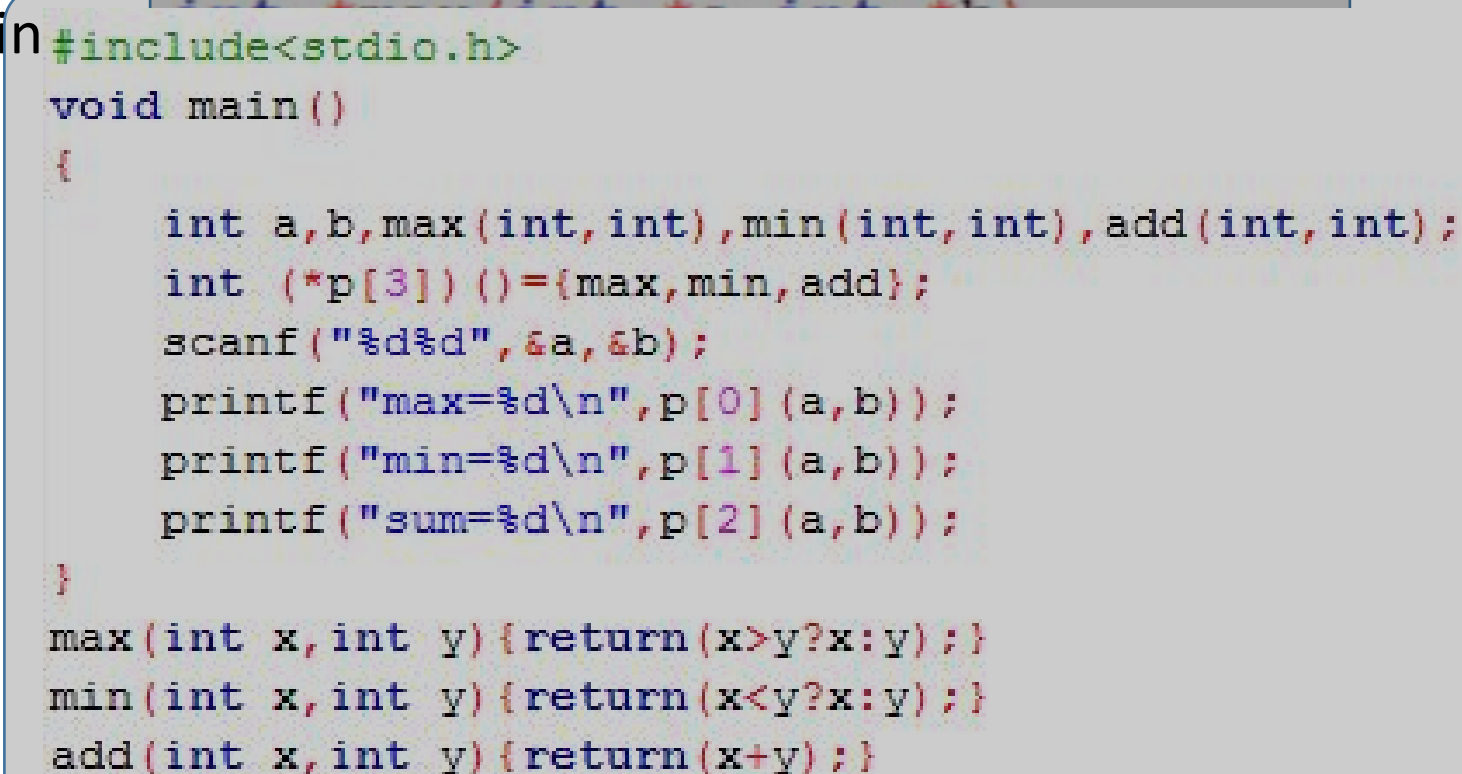
指针的数据类型

定义	含义
<code>int i;</code>	定义整型变量i
<code>int *p;</code>	p为指向整型数据的指针变量
<code>int a[n];</code>	定义含n个整元素的整型数组a
<code>int *p[n];</code>	定义n个指向整型数据的指针变量组成的指针数组p
<code>int (*p)[n];</code>	定义指向含n个元素的一维整型数组的指针变量p
<code>int f();</code>	定义返回整型数的函数f
<code>int *p();</code>	定义返回值为指针的函数p，返回的指针指向一个整型数据
<code>int (*p)();</code>	定义指向函数的指针变量p，p指向的函数返回整型数
<code>int **p;</code>	定义二级指针变量p，它指向一个指向整型数据的指针变量

下列定义的含义

● `int *(*p)();`  指向函数的指针，函数返回int 型指针

● `int (*p)`  型变量

● 

```
#include<stdio.h>

void main()
{
    int a,b,max(int,int),min(int,int),add(int,int);
    int (*p[3])()={max,min,add};
    scanf("%d%d",&a,&b);
    printf("max=%d\n",p[0](a,b));
    printf("min=%d\n",p[1](a,b));
    printf("sum=%d\n",p[2](a,b));
}

max(int x,int y){return(x>y?x:y);}
min(int x,int y){return(x<y?x:y);}
add(int x,int y){return(x+y);}
```