# 回忆一下

```
#include <stdio.h>
void main(void)
{
int i = 0;
printf("%x\n",&i);
}
```

```
a0fa38
请按任意键继续. . .
```

```
int i = 0;
int p = &i;
printf("0x%x\n",&i);
printf("0x%x\n",p);
```

```
0x78fc80
0x78fc80
请按任意键继续. . .
```

```
int i = 0;
int p = &i;
printf("0x%x\n",&i);
printf("%p\n",p);
```

```
0x1bf9c4
001BF9C4
请按任意键继续. . .
```

```
int a[2];
printf("%p\n",&a);
printf("%p\n",a);
printf("%p\n",&a[0]);
printf("%p\n",&a[1]);
```

```
00BDFC94
00BDFC94
00BDFC94
00BDFC98
请按任意键继续. . .
```

```
scanf("%d",&i);
```

将取得的变量的地址（&i)传递给一个函数(scanf())
那么我们能否通过这个地址访问这个变量的数值呢？
能否有一个能保存变量地址的变量呢？

指针

# 第五章

# 指针

# 重难点

1. 理解地址、指针和指针变量的概念
2. 熟练掌握指针变量的定义、初始化、赋值和引用
3. 熟练掌握指针的运算
4. 熟练掌握使用指针操作数组
5. 熟练掌握使用指针进行字符串处理
6. 学会利用指针动态分配内存

重点

1. 理解指针与指针数据类型
2. 学会常见指针运算
3. 理解数组名与指针变量
4. 动态数组的实现
5. 数组指针与指针数组

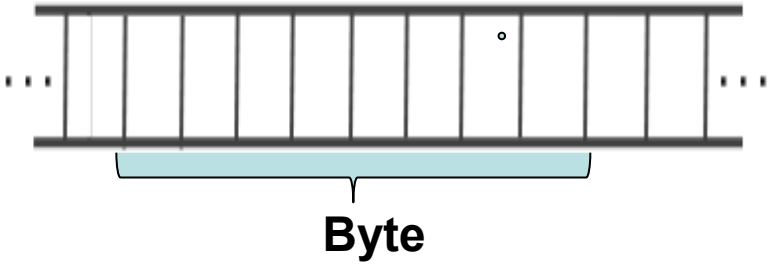
难点

# 5.1 指针和指针变量


## 一. 存储单元：

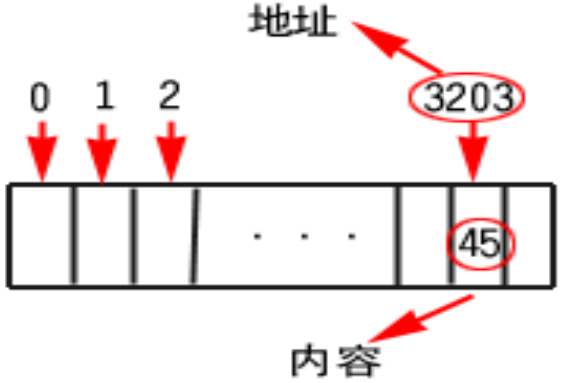
内存：内部存储器是由线性连续的存储单元组成的。

存储单元的最小单位是字节。




## 二. 地址：

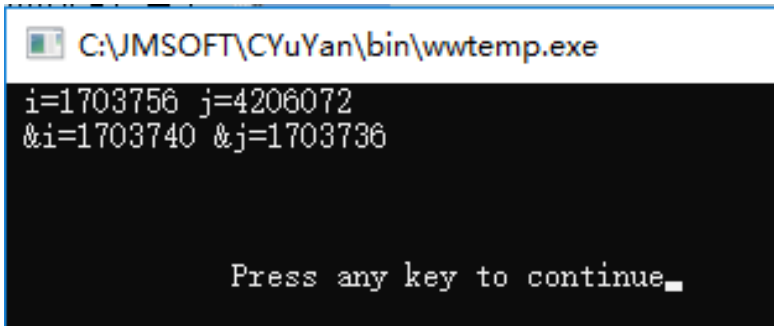
为了访问内存中的某个存储单元，我们要为它编号，这种编号称为地址。通过地址我们就能够访问该地址标识存储单元所存储的数据。



## 三、变量的地址：

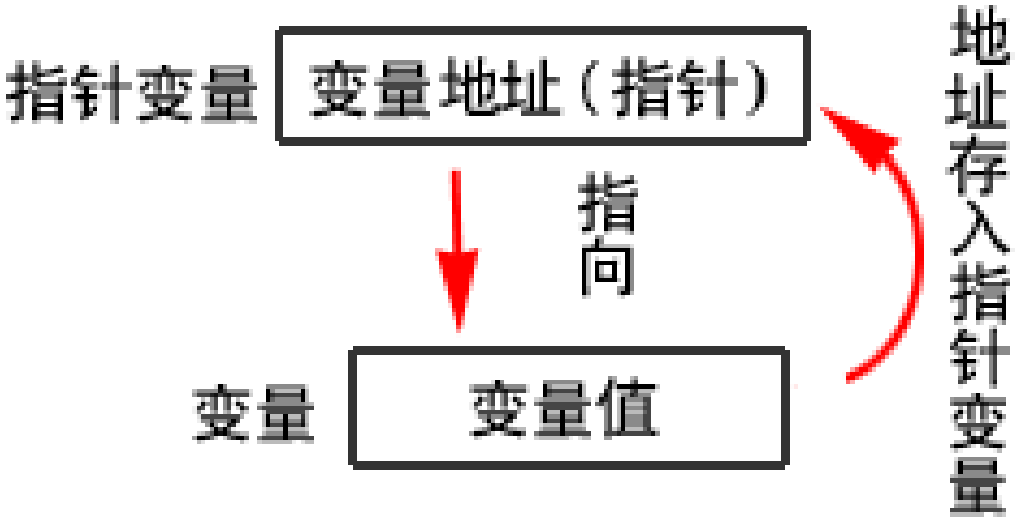
变量在内存中总占用几个连续的字节，开始字节的地址，就是变量的地址。**当没有赋值时，变量的值是不确定的。**

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

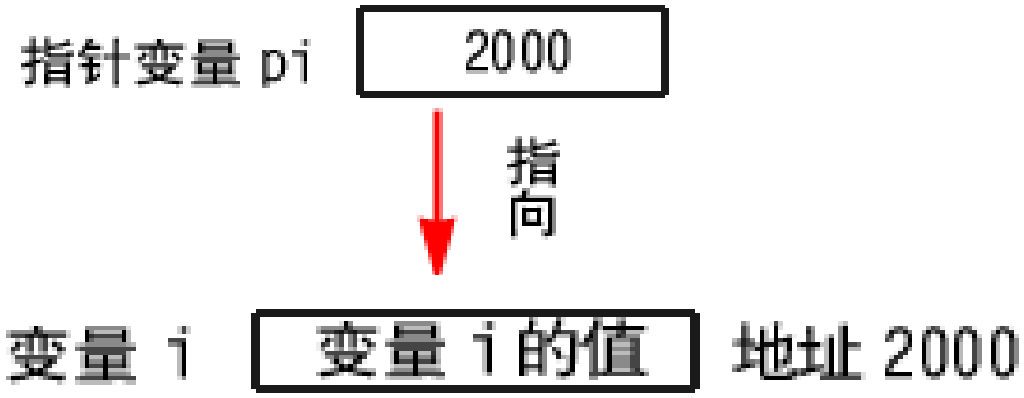


四、指针：一个变量的地址称为该变量的指针。

五、指针变量：若一个变量专用于存放另一个变量的地址（指针），则该变量称为指针变量。



```
int i;  int *pi=&i;
```




当变量i的地址存入指针变量pi后，即称指针pi指向了变量i。

## 六、变量的存取方法：

- **直接存取：** 直接根据变量名存取数据： i=100
- **间接存取：** 通过指针变量存取数据： *pi=100

针对源程序而言

# 5.2 指针变量的定义和使用

## 5.2.1、指针变量的定义：

语法： 类型说明符 *变量名 1,*变量名 2……;

例: int *p1, *p2;

说明:

1. 指针变量是一个存放另一个变量地址的变量，所占内存由编译系统决定。
2. 指针变量的存储类型、作用域、可见性、寿命同普通变量;
3. 指针变量的数据类型指明了该指针指向的内存空间所存储数据的数据类型。
4. 在引用指针变量前必须首先让它指向一个变量。

## 5.2.2、指针变量的赋值：

**语法：类型说明符 *指针名=初始地址值;**

**例：**
```
int  i;
int *p=&i;
```

**说明：**

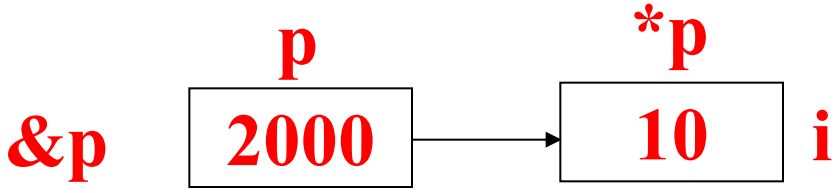
1. 用变量地址初始化指针时，该变量必须预先定义
2. 变量和指针的type应一致
3. 空指针不表示任何指向，而是一种状态标志

```
int *px=0;      char *py=“”,*py=NULL;
```

## 5.2.3、指针的存取运算

“&”（地址运算符）：取变量的存储地址
“*”（引用运算符）：取指针所指向单元的内容

```
int i=10, *p=&i;
```




i ------普通变量，它的内容是10
&i------指针，它的内容是地址：2000 =p
p--------指针变量，它的内容是地址:2000
*p-------指针所指向单元的内容 - - 数据:10 =i
&p - - --指针变量占用内存的地址:二级指针

```
void main(void)
{
  printf("%2d",sizeof(int *));
  printf("%2d",sizeof(char *));
  printf("%2d",sizeof(double *));
  printf("\n");
}
~
```

```
8 8 8
```

**在64位操作系统里，指针占8个字节**

```
void main(void)
{
  int i = 5;
  int *p = &i;
  printf("%3d",*p);
  printf("%3d",*p+1);
  printf("\n");
}
```

```
  5  6
```

***两种含义：**

1. **定义时，表示后面的变量是指针**
2. **使用时，表示取值（取指针指向的内存空间的值）**

习题1.1．定义：float x,*p=&x; 则表达式中错误是
A）*&p B）*&x C）&*p D）&*x

D

1.10．下面程序的输出结果是 。

```
#include <stdio.h>
void main(void)
{
    int n=6, *p=&n;
    *p=8;
    printf("%d\n",n);
}
```

A）8 B）6
C）7 D）不确定

A

1.11．下面程序的输出结果。

```
#include <stdio.h>
void main(void)
{
    int a=3, b=6, c=9;
    int *pa=&a, *pb=&b, *p;
    *(p=&c)=*pa*(*pb);
    printf("%d\n",c);
}
```

A）8 B）18
C）27 D）9

B

# 5.3 指针与数组

```c
#include <stdio.h>

int main()
{
  int a[5]={1,2,3,4,5};
  int i;
  for(i=0;i<5;i++)
  {
    printf("%d ",a[i]);
  }
  printf("\n");
  return 0;
}
```

```c
int main()
{
  int a[5]={1,2,3,4,5};
  int i;
  int *p = a;
  for(i=0;i<5;i++)
  {
    //printf("%d ",a[i]);
    printf("%d ",*(p+i));
  }
  printf("\n");
  return 0;
}
```

```c
printf("%d ",*(a+i));
```

```
[nnbao@enine-ahu-asipp unix_test]$ gcc 1_pointer.c -o 1_pointer
[nnbao@enine-ahu-asipp unix_test]$ ./1_pointer
1 2 3 4 5
```


```
void main(void)
{
    int i;
    char *str = "helloworld";
    printf("%c\n",*str);
    for(i=0;i<10;i++)
    {
        printf("%c  ",*(str+i));
        printf("%c\n",str[i]);
    }
}
```

```
h
h  h
e  e
l  l
l  l
o  o
w  w
o  o
r  r
l  l
d  d
```

## 5.3.1 指针与一维数组：

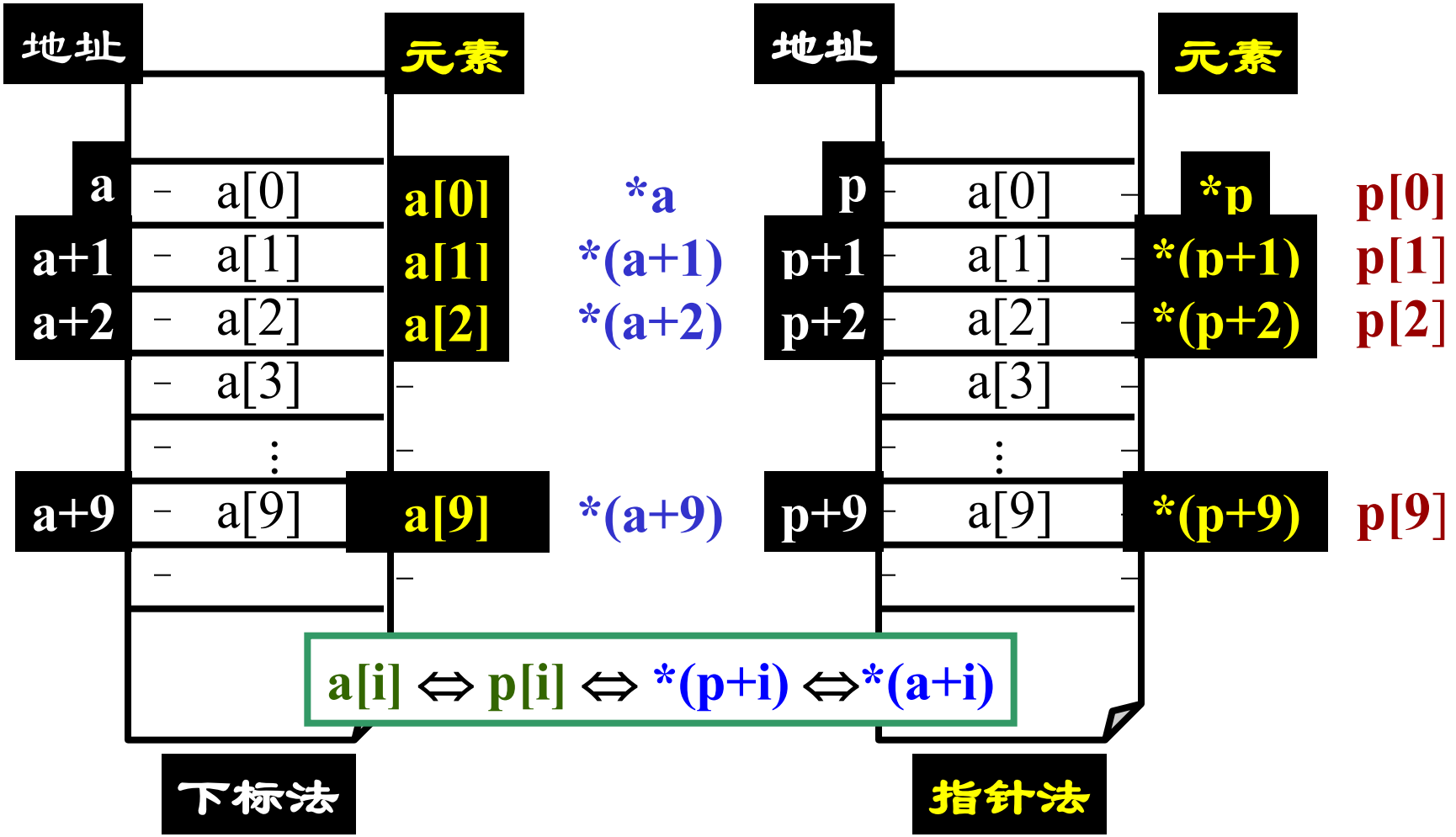
设： int Num[5]={2, 4, 6, 8, 10}, *Ptr=Num;

**下标法**：①地址&Num[i] ②元素Num[i]
**指针法**：①地址Num+I ②元素*(Num+i)

**Ptr+i ⇔ Num+i ⇔ &Num[i] ⇔ &Ptr[i]**

***(Ptr+i) ⇔ *(Num+i) ⇔ Num[i] ⇔ Ptr[i]**

设：int a[10]，*p；p=a；


地址
元素
a
a[0]
a[1]
a[2]
a[3]
a[9]
a+1
a+2
a+9
*a
*(a+1)
*(a+2)
*(a+9)
a[0]
a[1]
a[2]
a[9]
地址
元素
p
p+1
p+2
p+9
*p
*(p+1)
*(p+2)
*(p+9)
p[0]
p[1]
p[2]
p[9]
a[i] ⇔ p[i] ⇔ *(p+i) ⇔*(a+i)
下标法
指针法

## 说明：

(1) 数组名代表数组的首地址（起始地址），也就是第一个元素的地址；**指针变量使用前必须赋值**。

(2) **指针变量是地址变量，数组名是地址常量**：指针变量的值可以改变；而数组名一旦定义就不能变了。

例如： int i, *p, a[6]; 则： p=&i ; √

~~a=&i ;~~ ~~a++;~~ ~~a+=i ;~~

不能给常量赋值

对于一维数组，int a[10],*p=a;则

&a[i]=a+i

a[i]=*(a+i)

相应的 *(p+i)=*(a+i)=a[i]

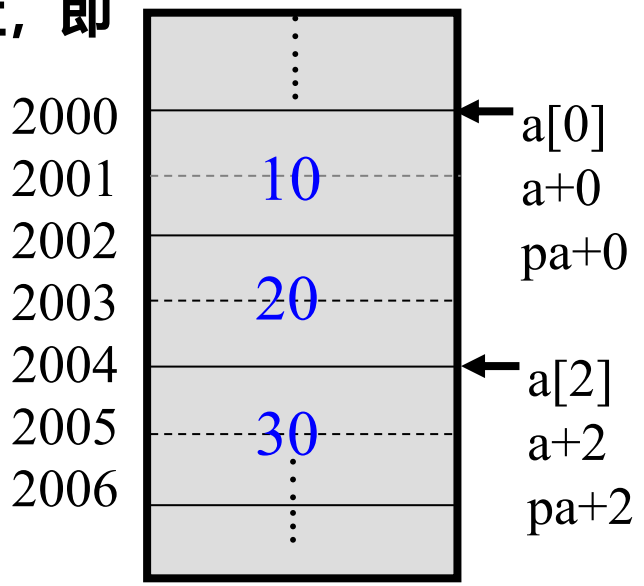
如果题设改为 int a[10],*p=a+1; 上式还成立吗?

# 5.3.1 指针与一维数组：

## 1. pointer±n

意义：指向该指针下移或上移n个数据之后的内存地址，即
(pointer)±n*sizeof(type)

```
short a[]={10,20,30,…};
short *pa=a;
&a[0]⇔a+0 ⇔pa+0
&a[2]⇔a+2 ⇔pa+2
```




**注意：（1）参加pointer ± n运算的指针必须指向在内存中连续存放的数据区域，如数组；**

**（2）pointer ± n指向原数据前后的第n 个数据，但pointer指针本身的指向并未改变；**

**（3）pointer±=n同时改变了指针的指向，表达式和运算结束后的pointer均指向原数据前后的第n 个数据。**

## 习题5.1.5. 以下程序的输出结果是 。 D

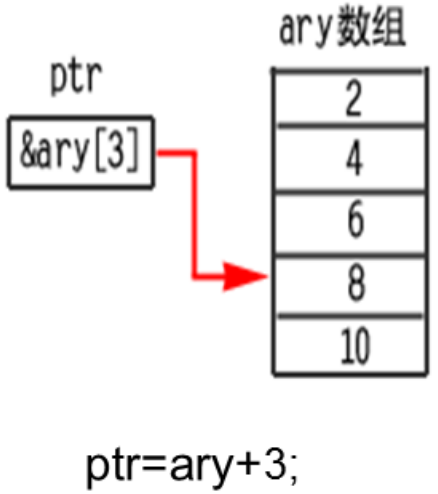
```
#include "stdio.h"
void main( void )
{
    int a[]={1,2,3,4,5,6},*p;
    p=a; *(p+3)+=2;
    printf("%d,%d\n", *p, *(p+3));
}
```

A) 0,5 B) 1,5 C) 0,6 D) 1,6

## 2. pointer++/--

**意义：指向该指针下移或上移1个数据之后的内存地址，即**

(pointer)±sizeof(type)



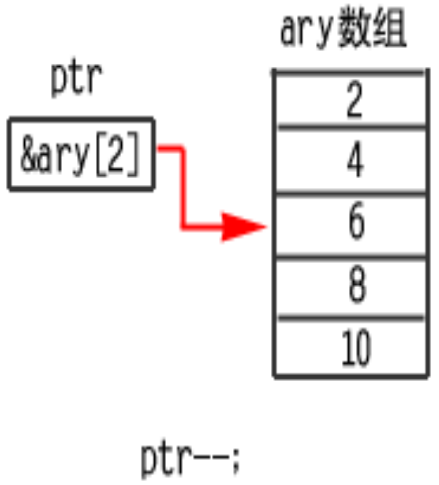

ptr=ary+3;

```
eg:   int *ptr, ary[5]={2,4,6,8,10} ;
      ptr=ary+3;    ptr--;
```

**注意：**

***ptr++ ⇔ *(ptr++) ≠ (*ptr)++**

***++ptr ⇔ *(++ptr) ≠ ++(*ptr)**




ptr--;

**++与*优先级相同，结合性自右向左：**

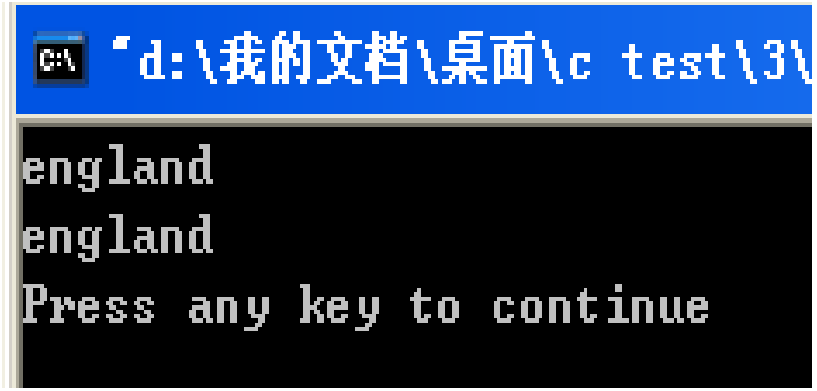
## 【例5-5】 利用指针自增、自减访问数组。

```
#include <stdio.h>
void main( void )
{
   int Num[5], i, *Ptr=Num;
   printf("顺序输入5个整数： ");
   for(i=0; i<5; i++)
      scanf("%d", Ptr++ );
   printf("逆序打印Num： ");
   for(i=0; i<5; i++)
     printf(" %d", *--Ptr);
   printf("\n");
}
```

```
顺序输入5个整数： 1 2 3 4 5
逆序打印Num：  5 4 3 2 1
```

## 指针自增自减功能在字符串中的应用.

```
#include <stdio.h>
void main (void)
{
char a[20]="china";
char b[20]="england";
char *t=a,*s=b;
while((*t++=*s++)!='\0');
puts(a);
puts(b);
}
```




分析：char *t, char *s分别指向两个字符串，将s指向的字符串复制到t指向的字符串中。

3. **指针相减：两个指向同一数组不同元素的指针相减，其结果为两个指针之间的数据的个数：**

ptr1-ptr2=(ptr1的值- ptr2的值) /sizeof(type)

```
/* Description: 指针法求字符串长度.    */
#include <stdio.h>
void main( void )
{
    char str[20], *ptr;
    printf("请输入一字符串:");
    gets(str);
    ptr=str;
    while(*ptr)//循环体为空，
    ptr++;
    printf("字符串长度为%d\n", ptr-str);
}
```

```
请输入一字符串:happy
字符串长度为5
```

**4.指针比较：** 两个指向同一数组不同元素的指针进行比较运算——比较两个指针的位置，结果是**逻辑值**。

| | |
|---|---|
| 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 |

特例：指针变量还可以与空指针 NULL 或 0 比较。
ptr1==0 表明 ptr1 是空指针，它不指向任何变量。
ptr1!=0 表示 ptr1 不是空指针。

下面程序的作用是?

```
#include <stdio.h>
#include <string.h>
void main( void )
{
 char str[ ]="abcd", *s, *t, c;
 printf("原字符串为: %s\n", str);
 s=str; t=s+strlen(s)-1;
 while(s<t)
  c=*s, *s=*t, *t=c, ++s,--t;
 printf("新的字符串为: %s\n", str);
}
```


分析：程序使用指针变量 s、t分别指向首字符、尾字符，所指向字符互换后 s 指向下一个字符，t 指向前一个字符，续继循环，直到 s 指向的字符不再在 t 指向的字符之前。

# 5. 指针赋值运算：

**(1)取地址运算——把变量地址值赋值给指针变量：**

如：int a, *pa ; pa=&a ;

**(2)指针间赋值——把指针变量的值赋给另一个指针变量：**

如：int a,*pa,*pb; pa=&a ; pb=pa;

**(3)数组名赋值（数组名是指针常量）**

如：int a[10],*pa; pa=a;

**(4)算术赋值运算：**

如：int a[10],*pa; pa=a+3; pa+=2;

**可以这样赋值吗？** ⟶

如 int *p;
p=1000;

1.12．下面程序的输出结果是 。

```
#include <stdio.h>
void main( void )
{
int a[ ]={1,3,5,7,9,11,13,15}, *p=a;
    printf("%d, %d\n", *p+3, *(p+3));
}
```

A）4,5　B）6,5

C）4,7　D）6,7

C

1.14．下面程序的输出结果是 D 。

```
#include <stdio.h>
void main( void )
{
    int a[ ]={2,4,6,8,10};
    int *p=&a[4];

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

A）10　B）9

C）7　D）8

## 5.3.3. 指针与二维数组：

定义：short a[3][4];

(1) a是数组名，包含三个元素a[0],a[1],a[2]

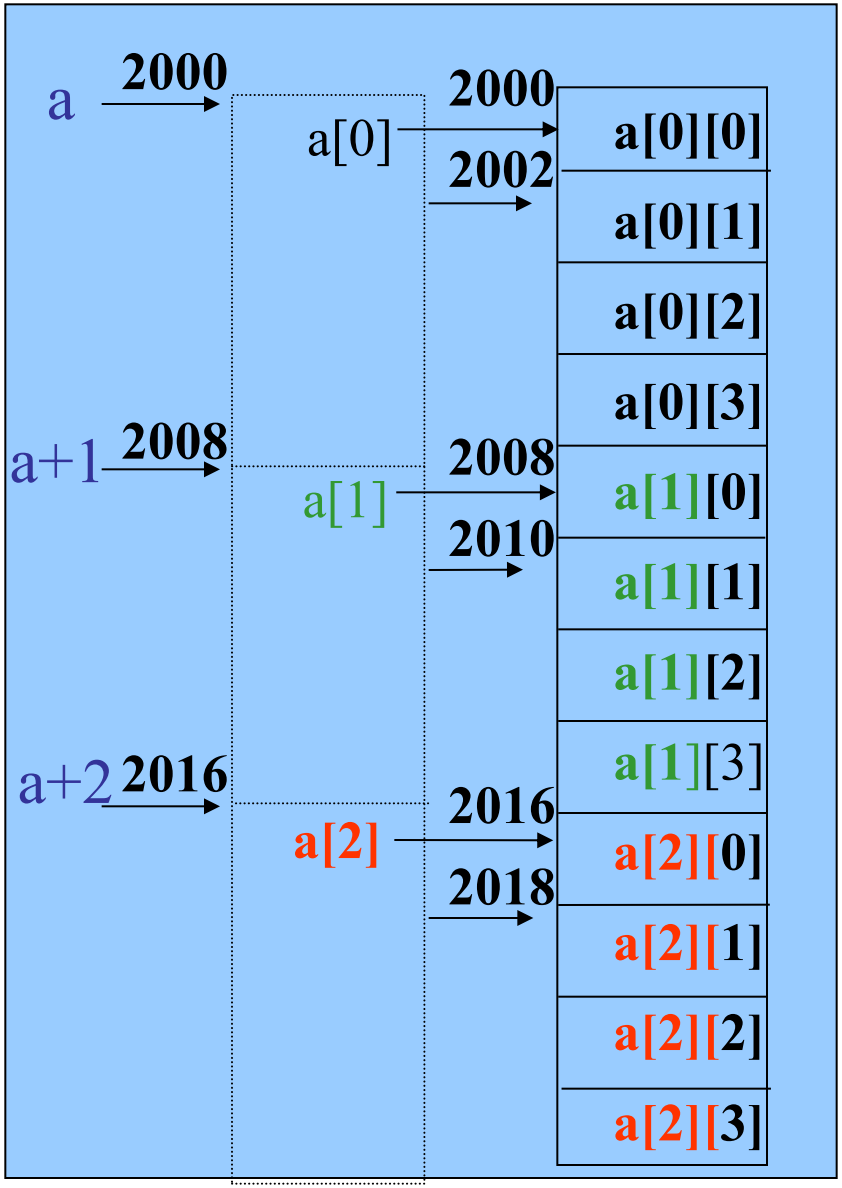
(2) 每个元素a[i]又是一个一维数组，包含4个元素

a+i⇔&a[i]: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

a[i]+j⇔*(a+i)+j ⇔ & a[i][j]

a[i][j] ⇔ *(*(a+i)+j) ⇔ *(a[i]+j) ⇔(*(a+i))[j]



## 5.3.3. 指针与二维数组：

定义：short a[3][4];

(1) a是数组名，包含三个元素a[0],a[1],a[2]

(2) 每个元素a[i]又是一个一维数组，包含4个元素

| | | | |
|---|---|---|---|
| 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] |

a+i⇔&a[i]: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

a[i]+j⇔*(a+i)+j ⇔ & a[i][j]

a[i][j] ⇔ *(*(a+i)+j) ⇔ *(a[i]+j) ⇔(*(a+i))[j]

| 表示形式 | 含义 | 地址 |
| --- | --- | --- |
| a | 二维数组名，指向一维数组a[0]，即0行首地址 | 2000 |
| a[0],<br>*(a+0),<br>*a | 0行0列元素地址 | 2000 |
| a+1，&a[1] | 1行首地址 | 2008 |
| a[1],*(a+1) | 1行0列元素a[1][0]的地址 | 2008 |
| a[1]+2,<br>*(a+1)+2,<br>&a[1][2] | 1行2列元素a[1][2] 的地址 | 2012 |
| *(a[1]+2),<br>*(*(a+1)+2),<br>a[1][2] | 1行2列元素a[1][2]的值 | a[1][2]的值 |

【想一想】

如果两个指针的地址值相等，它们是否就相同或等价?

a=2000　　a[0]=2000　　a=a[0]?

a+2=2016　a[2]=2016　　a+2= a[2]?

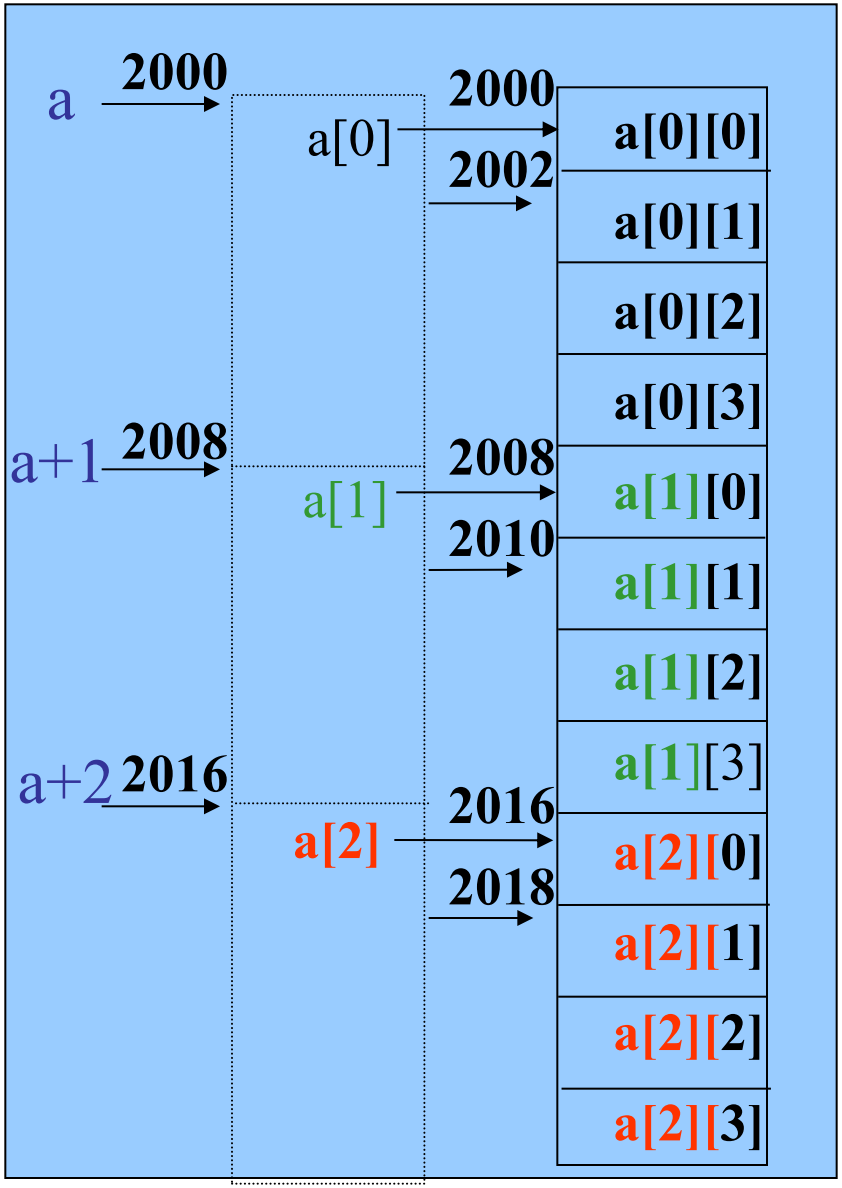
虽然它们的地址值相等，但它们的类型不同，所指向的对象也不同：

a 是一个行指针（数组指针），指向了a[0]，也就是第0行；

a[0]并不是一个实际的元素，它代表第0行的四个元素，本身又是一个指针，指向了a[0][0], a[0]称为列指针!

a=2000　　a+1=2008　　(行指针)

a[0]=2000　　a[0]+1=2002　　(列指针)



1.8．对于数组a[5][5]，*(a+2)+3 表示__B__，*(a[3]+2)表示__C__ 。

A）a[2][3]　B）&a[2][3]

C）a[3][2]　D）&a[3][2]

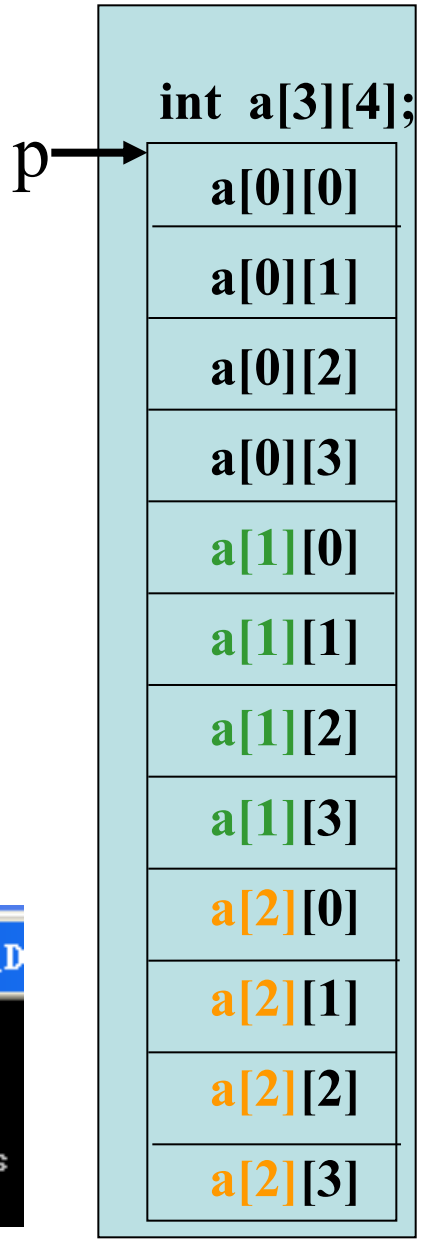
| | | | | |
|---|---|---|---|---|
| a[0][0] | a[0][1] | a[0][2] | a[0][3] | a[0][4] |
| a[1][0] | a[1][1] | a[1][2] | a[1][3] | a[1][4] |
| a[2][0] | a[2][1] | a[2][2] | a[2][3] | a2][4] |
| a[3][0] | a[3][1] | a[3][2] | a[3][3] | a[3][4] |
| a[4][0] | a[4][1] | a[4][2] | a[4][3] | a[4][4] |

# (3) 通过一级指针引用二维数组元素：

```
#include <stdio.h>
void main( void )
{
   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);
   }
}
```

```
"d:\我的文档\桌面\c test\3\D

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



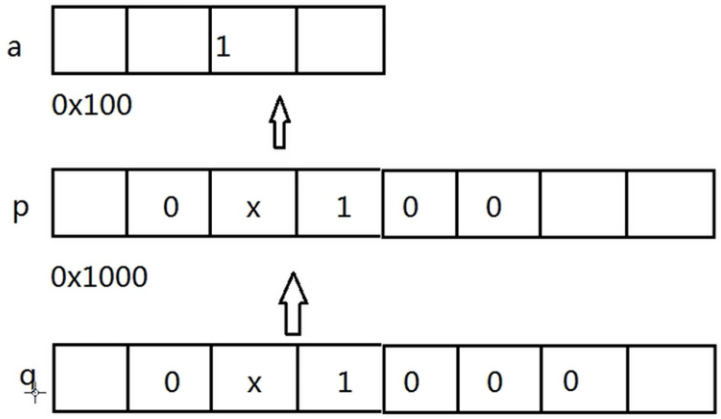
# （4）数组指针

## ①二级指针：

**类型说明符 **二级指针变量名;**

```
int   num=100;
int  *prt1=&num;
int **prt2=&ptr1;
```

```
int a = 1;
int *p = &a;
int **q = &p;
```




*ptr2等于prt1, **ptr2 等于num, 即 100。

二级指针要进行自加、自减等各种指针运算，就必须具备类似行地址、列地址的结构才行。

**注：如果将一个指向基本数据的指针变量地址赋给二级指针变量，这个二级指针变量不能进行各种指针运算，最多能做*、**引用运算。**

1.16 若定义char s[5][10], *p=*s;则使用p 来表示数组元素s[2][3]的表达式是____C____。

A）*(*(p+2)+3)　　B）*(p+2*3+10)

C）*(p+2*10+3)　　D）*(*(p+3)+2)

## ②数组指针：指向一维数组的指针变量，又称行指针

**类型说明符 (*变量名)[一维数组长度];**

如： short (*P)[3];
short A[][3]={{2,4,6},{8,10,12},{14,16,18}};

让 P指向 A[0]的方法如下：
P=A; 或 P=&A[0];

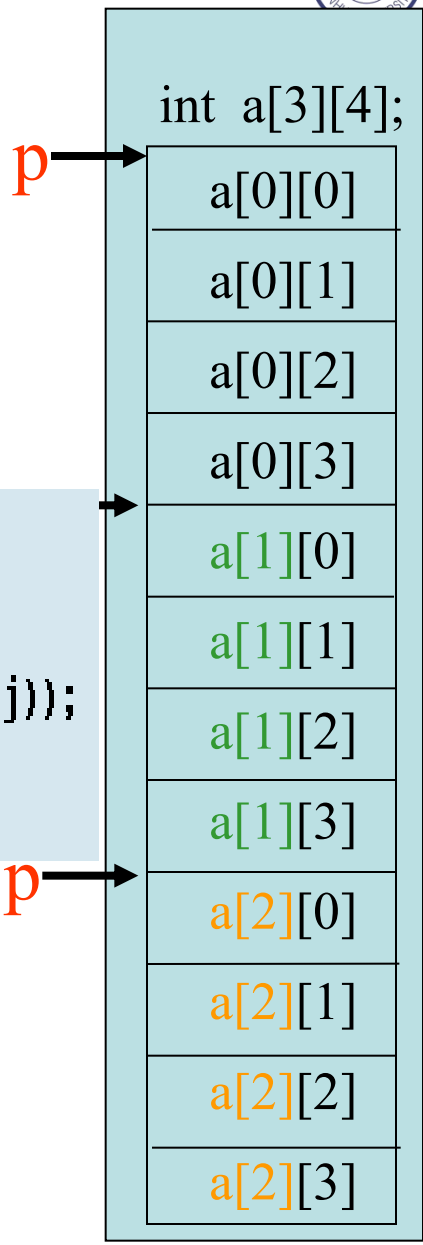
利用 P访问 A[i][j]的方法如下：
P[i][j] ⇔A[i][j]
*(*(P +i)+j) ⇔*(*(A +i)+j)
*(P[i]+j) ⇔*( A[i]+j)
(*(P +i))[j] ⇔ (*(A +i))[j]

```
#include <stdio.h>

void main( void )
{
    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+
        printf("\n");
    }
}
```

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

```
d:\我的文档\桌面\c test\3\
1 3 5 7
9 11 13 15
17 19 21 23
Press any key to continue
```



1.9 若定义int a[][3]={{1,2,3},{4,5,6}}, (*p)[3]=a; 则表达式*(*p+1)的值是__D__。

A）1 B）3 C）4 D）2

10. 定义 int (*p)[3]; 其中 p_____C_____ 。

A）定义不合法
B）是一个指针数组，每个元素是一个指向整型变量的指针
C）是一个数组指针，它指向一个具有三个整数的一维数组
D）是一个指向整型变量的指针

# 5.4 指针和字符串

## 5.4.1 一级字符型指针和单个字符串处理

定义一个一级字符型指针来处理单个字符串。一级字符型指针的定义方法：

char *一级字符指针变量名;

和整型指针变量一样，字符指针变量可以在定义时初始化，也可以通过赋值方式将其指向待处理的字符串，例如：

```
char *S1="C program", *S2;
char string[]="China";
S2=string;
```

设：char *cp; char str[20];

str由若干元素组成，每个元素放一个字符；而cp中存放字符串首地址

char str[20]; str="C program"; (×)

char *cp; cp="C program"; (√)

str是地址常量；cp是地址变量

cp接受键入字符串时,必须先开辟存储空间

例 char str[10];
scanf("%s",str); (✓)

而 char *cp;
scanf("%s", cp); (×)

改为：char *cp,str[10];
cp=str;
scanf("%s",cp); (✓)

# 例 字符串复制：采用数组元素复制实现

```
#include <stdio.h>
void main(void)
{
    char a[]="Today is Monday";
    char b[]="Tomorrow is Tuesday.";
    int i=0;
        printf(" string_a=%s\n string_b=%s\n",a,b);
    while(a[i]!='\0')
     {   b[i]=a[i];
         i++;
     }
     b[i]='\0';   //为什么要补上 ‘\0’?
     printf("\nstring_a=%s\nstring_b=%s\n",a,b);
}
```

```
 string_a=Today is Monday
 string_b=Tomorrow is Tuesday.

string_a=Today is Monday
string_b=Today is Monday
请按任意键继续. . .
```

# 例 字符串复制：采用字符指针实现

```
#include <stdio.h>
void main(void)
{
    char a[]="Today is Monday",*from=a;
    char b[]="Tomorrow is Tuesday.",*to=b;
    int i=0;
          printf(" string_a=%s\n string_b=%s\n",a,b);
          for(;*from!='\0';from++,to++)
        *to=*from;
    *to='\0';  //为什么要补这一句?
     printf("\nstring_a=%s\nstring_b=%s\n",a,b);
}
```

```
 string_a=Today is Monday
 string_b=Tomorrow is Tuesday.

string_a=Today is Monday
string_b=Today is Monday
```

1.8. 以下程序的输出结果是__A__ 。

```
char *s="ABCD", *p;
for( p=s;  *p ;  p++)
    printf("%s\n", p);
```

A）ABCD
BCD
CD
D

B）A
B
C
D

C）D
C
B
A

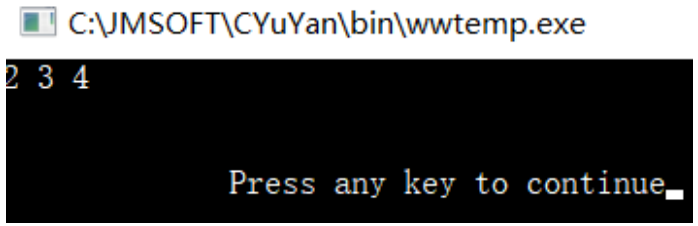
D）ABCD
ABC
AB
A

1.16. 若 char s[10]; s=“abcd”;printf(“%s”,s);程序的输出结果是__D__。

A）abcd B）a C）dcba D）编译错误

## 2.9. 阅读程序，写出程序运行结果。

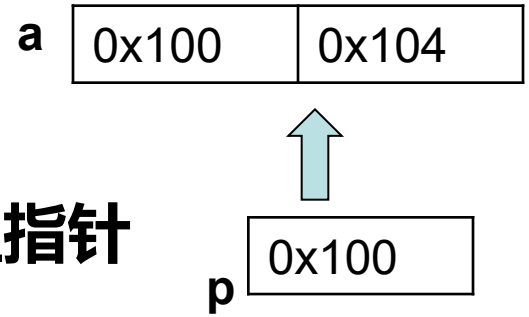
```
#include <stdio.h>
void main( void )
{
char *p="12345";
while (*p!='4')
    printf("%c ", *(p++)+1);
}
```



# 5.4.2 指针数组处理多个字符串

**如果数组的每个元素都是同种类型的指针，该数组便称为指针数组。指针数组中的每一个元素都是指针变量，只能存放地址。**

**定义指针数组：**




**语法：type *pointer[size];**

**功能：开辟一片内存存放size个type类型指针**

**说明：指针数组遵循数组的所有语法约定。**

**注意：区别 int *p[2] 、int (*p)[2] 的含意。**

**指针数组**

**数组指针(行指针)**

当指针数组中的每一个元素都指向一个一维数组时，指针数组就相当于二维数组的多个行地址（数组指针），指针数组名就相当于二维数组名。

**指针数组和二维数组的区别**：
指针数组中的每一个元素所指向的一维数组长度可以不一，一维数组之间也不需要在内存中连续存放，指针数组中的每一个元素均为指针变量，地址可变。

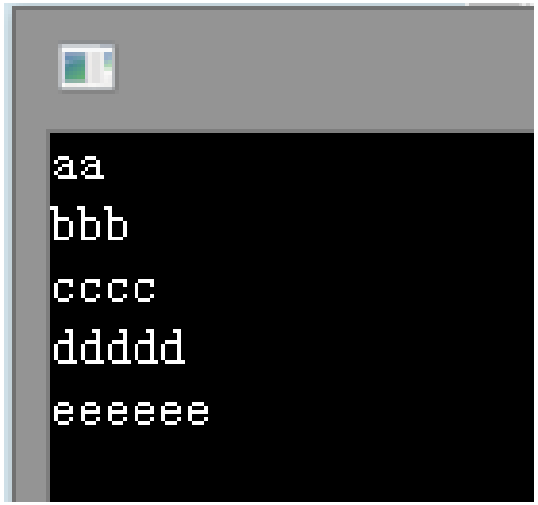
指针数组特别适合于处理若干个长度不等的字符串。指针数组使得字符串处理更加简单方便，不浪费内存空间。

## 指针数组应用：

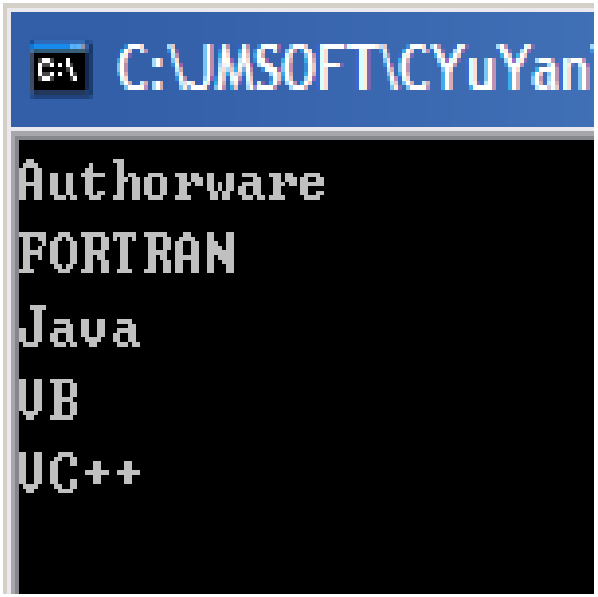
### 1. 多维数组降阶：

### 2.处理多个长度不等的字符串

```
#include <stdio.h>
void  main( void )
{
    char  *p[10]={"aa","bbb","cccc","ddddd","eeeeee"};
    int i ;
    for(i=0;i<5;i++)
      puts(p[i]);
}
```



```
// Program: EG05-11.c
// Description: 利用字符指针数组将五门课程名按字典顺序输出
#include <stdio.h>
#include <string.h>
void main(void)
{
   char *tmp,*str[5]={"VB","FORTRAN","VC++","Authorware","Java"};
   short i, j, k, n=5;
   for(i=0;i<n-1;i++)
   {
      k=i;
      for(j=i+1;j<n;j++)
         if(strcmp(str[k], str[j])>0)
            k=j;
       if(k!=i)
         tmp=str[i], str[i]=str[k], str[k]=tmp;
   }        //选择法排序
   for(i=0;i<n;i++)
        puts(str[i]);
}
```

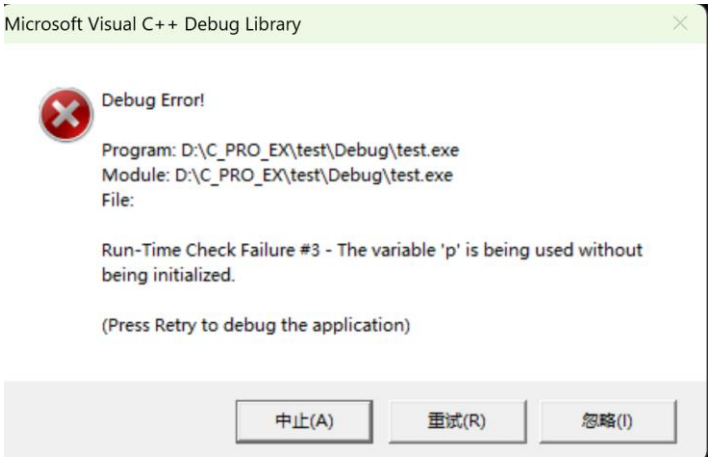


```
/*输入2个字符串，将二者连接后的结果输出(用指针完成)。*/
#include <stdio.h>
#include <string.h>
void main( void )
{
   char str1[20], str2[20], *ptr1, *ptr2;
   printf("请输入2个字符串:");
   gets(str1); gets(str2);
   ptr1=str1;  ptr2=str2;
   while(*ptr1)
       ptr1++;
   while(*ptr1=*ptr2)
       ptr1++, ptr2++;
   printf("合并字符串为%s\n",  str1);
}
```

# 野指针

```
#include <stdio.h>
void main(void)
{
int *p;
*p=100;
printf("%d",*p);
}
```

```
int *p=NULL;//空指针
```




# 如何合法地使用内存?

```
//系统分配的内存
int a;
int *p=&a;
//用户申请（释放）内存
```

## 5.5 指针和动态内存分配

void *malloc(unsigned int size);

功能：在内存的**动态**存储区中**分配**一块长度为“size”字节的连续内存空间。

返回值：分配成功，返回所分配内存空间的起始地址，否则，返回NULL。

实际使用时，需要先用 **sizeof**()求出一个元素所需字节数，再**乘上所需元素个数**，调用函数的地址必须由普通指针 void *强制**转换**为所需**类型**，最后还要**查看是否分配成功**，如果没有分配成功，必须停止后继操作。

void free(void *pointer);

功能：**释放** pointer 所指向的一块内存空间，pointer 是一个任意类型的指针变量，它指向被释放区域的首地址。被释放区应是由 malloc 或 calloc 函数所分配的区域。

# 动态内存分配的简单示例

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

void main(void) {
    int size, i;
    int *array;
    int sum = 0;
    printf("Enter the size of the array: ");
    scanf("%d", &size);
    array = (int *)malloc(size * sizeof(int));    // 动态分配内存
    if (array == NULL) {
        printf("Memory allocation failed.\n");
    }
    printf("Enter %d integers:\n", size);
    for (i = 0; i < size; i++) {
        scanf("%d", &array[i]);
        sum += array[i];
    }
    printf("Sum of the elements: %d\n", sum);
            free(array);    // 释放内存
}
```

```
Enter the size of the array:
Enter 3 integers:
1 2 4
Sum of the elements: 7
```

# 指针小结

| 定　　义 | 含　　义 |
| --- | --- |
| int Num[n]; | 定义有 n 个整型元素的数组 Num，数组名 Num 为一级指针常量 |
| int *Ptr; | Ptr 为一级指针变量，指向整型数据 |
| int **Ptr; | Ptr 是二级指针变量，指向一级整型指针 |
| int *Ptr[n]; | 定义有 n 个一级整型指针变量元素的数组，指针数组名 Ptr 为二级指针常量 |
| int (*Ptr)[n]; | 定义数组指针/行指针变量 Ptr，指向内存中 n 个整型数据元素，Ptr 为二级指针变量 |

使用指针时的常见错误：

（1）试图修改地址常量：如对数组名进行自加、自减、将一个字符串赋值给它等。
（2）使用未初始化的野指针，导致系统混乱。
（3）对指一个基本类型数据的一级指针变量加减。
（4）对两个未指向同一数组元素的指针进行比较、相减
（5）指针访问越界：比如将一个长字符串复制到短字符串所在空间。
（6）对通过一级指针间接指向一个基本类型数据的二级指针变量进行复杂的指针运算。
（7）尽管内存分配失败，仍然继续非法访问内存。
（8）动态申请的内存在程序运行结束后仍不释放造成内存泄漏。
（9）继续使用已释放所申请内存后的“野指针”。