



中國人民大學
RENMIN UNIVERSITY OF CHINA

信息学院
SCHOOL OF INFORMATION

程序设计荣誉课程

12. 语法4——指针

授课教师：游伟 副教授、孙亚辉 副教授

授课时间：周二14:00 – 15:30，周四10:00 – 11:30（教学三楼3304）

上机时间：周二18:00 – 21:00（理工配楼二层机房）

课程主页：<https://www.youwei.site/course/programming>

C语言运算符概览

算术运算符:	+ - * / % ++ --
赋值运算符:	= 及其扩展
求字节数 :	sizeof
强制类型转换:	(类型)
函数调用符运算符:	()
关系运算符:	< <= == > >= !=
逻辑运算符:	! &&
条件运算符:	?:
下标运算符:	[]
分量运算符:	. ->
位运算符 :	<< >> ~ & ^
指针运算符:	* &
逗号运算符:	,

优先级	运算符	名称或含义	使用形式	结合方向
1	[]	数组下标	数组名[常量表达式]	左到右
	()	圆括号	(表达式) 函数名(形参表)	
	.	成员选择 (对象)	对象.成员名	
	->	成员选择 (指针)	对象指针->成员名	
2	-	负号运算符	-表达式	右到左
	(类型)	强制类型转换	(数据类型)表达式	
	++	自增运算符	++变量名 变量名++	
	--	自减运算符	--变量名 变量名--	
	*	取值运算符	*指针变量	
	&	取地址运算符	&变量名	
	!	逻辑非运算符	!表达式	
	~	按位取反运算符	~表达式	
3	sizeof	长度运算符	sizeof(表达式)	左到右
	/	除	表达式 / 表达式	
	*	乘	表达式*表达式	
4	%	余数 (取模)	整型表达式%整型表达式	左到右
	+	加	表达式+表达式	
5	-	减	表达式-表达式	左到右
	<<	左移	变量<<表达式	
6	>>	右移	变量>>表达式	左到右
	>	大于	表达式>表达式	
	>=	大于等于	表达式>=表达式	
	<	小于	表达式<表达式	
	<=	小于等于	表达式<=表达式	

优先级	运算符	名称或含义	使用形式	结合方向
7	==	等于	表达式==表达式	左到右
	!=	不等于	表达式!= 表达式	
8	&	按位与	表达式&表达式	左到右
9	^	按位异或	表达式^表达式	左到右
10		按位或	表达式 表达式	左到右
11	&&	逻辑与	表达式&&表达式	左到右
12		逻辑或	表达式 表达式	左到右
13	?:	条件运算符	表达式1? 表达式2: 表达式3	右到左
14	=	赋值运算符	变量=表达式	右到左
	/=	除后赋值	变量/=表达式	
	=	乘后赋值	变量=表达式	
	%=	取模后赋值	变量%=表达式	
	+=	加后赋值	变量+=表达式	
	-=	减后赋值	变量-=表达式	
	<<=	左移后赋值	变量<<=表达式	
	>>=	右移后赋值	变量>>=表达式	
	&=	按位与后赋值	变量&=表达式	
	^=	按位异或后赋值	变量^=表达式	
	=	按位或后赋值	变量 =表达式	
15	,	逗号运算符	表达式,表达式,...	左到右

运算符优先级和结合性

目录

1. 基本概念
2. 指针与数组
3. 指针与结构体
4. 指针与函数
5. 动态内存分配与释放

12.1 基本概念

- 若在程序中定义了一个变量，在运行时系统就会给这个变量分配内存单元。编译系统根据程序中定义的变量类型，分配一定长度的内存空间。
- 内存的每一个字节有一个编号，即“内存地址”。通过地址能找到所需的变量单元，即地址指向该变量单元，将地址形象化地称为“指针”。
- C语言中的地址包括位置信息(内存编号，或称纯地址)和它所指向的数据的类型信息，或者说它是“带类型的地址”。
- 存储单元的地址和存储单元的内容是两个不同的概念。在程序中一般是通过变量名来引用变量的值。通过取地址操作获得变量的地址。
- 直接按变量名进行的访问，称为“直接访问”方式。还可以采用另一种称为“间接访问”的方式，即将变量的地址存放在另一变量（指针变量）中，然后通过该指针变量来找到对应变量的地址，从而访问变量。

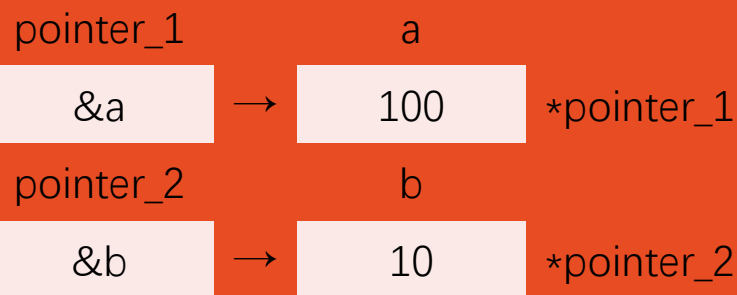
```
short i=1,j=2,k=3;  
short *x = &j;  
printf("%d %d\n", i, *x);
```

变量名	地址	内容
i	2000	1
	2001	
j	2002	2
	2003	
k	2004	3
	2005	

12.1 基本概念

■ 例12-1：通过指针变量访问整型变量

```
1. #include <stdio.h>
2. int main()
3. { int a=100,b=10;
4.   //定义整型变量a,b, 并初始化
5.   int *pointer_1,*pointer_2;
6.   //定义指向整型数据的指针变量pointer_1, pointer_2
7.   pointer_1=&a; //把变量a的地址赋给指针变量pointer_1
8.   pointer_2=&b; //把变量b的地址赋给指针变量pointer_2
9.   printf("a=%d,b=%d\n",a,b); //输出变量a和b的值
10.  printf("*pointer_1=%d,*pointer_2=%d\n",
11.         *pointer_1,*pointer_2); //输出变量a和b的值
12. }
```



注意

- 定义指针变量时，左侧应有类型名，否则就不是定义指针变量。

```
C:\WINDOWS\system32\cmd.exe
a=100,b=10
*pointer_1=100,*pointer_2=10
请按任意键继续. . .
```



*pointer_1; //企图定义pointer_1为指针变量。出错



int *pointer_1; //正确，必须指定指针变量的基类型

定义指针变量

类型名 *指针变量名;

```
int *pointer_1, *pointer_2;
```

左端的int是在定义指针变量时必须指定的“**基类型**”。指针变量的基类型用来指定此指针变量可指向的变量的类型。

有基本数据类型(如int, float等)的变量，就可以有指向这些类型变量的指针，因此，指针变量是基本数据类型派生出来的类型，它不能离开基本类型而独立存在。

在定义指针变量时要注意:

- (1) 指针变量前面的“*”表示该变量为指针型变量。指针变量名则不包含“*”。
- (2) 定义指针变量时必须指定基类型。一个变量的指针的含义包括两个方面，一是以存储单元编号表示的纯地址（如编号为2000的字节），一是它指向的存储单元的数据类型（如int, float等）。
- (3) 如何表示指针类型。指向整型数据的指针类型表示为“int *”，读作“指向int的指针”或简称“int指针”。
- (4) 指针变量中只能存放地址（指针），不要将一个整数赋给一个指针变量。

引用指针变量

- ① 给指针变量赋值。
- ② 引用指针变量指向的变量。
- ③ 引用指针变量的值。

```
int a, *p;  
p=&a;           //把a的地址赋给指针变量p           ①  
printf("%d",*p); //以整数形式输出指针变量p所指向的变量的值，即a的值           ②  
*p=1;           //将整数1赋给p当前所指向的变量，由于p指向变量a，相当于把1赋给a，即a=1           ②  
printf("%p",p);  //以地址形式输出指针变量p的值，由于p指向a，相当于输出a的地址，即&a           ③
```

注意

- 要熟练掌握两个有关的运算符：

- (1) **&** 取地址运算符。&a是变量a的地址。
- (2) ***** 指针运算符（或称“间接访问”、“解引用”运算符），*p代表指针变量p指向的对象。

12.1 基本概念

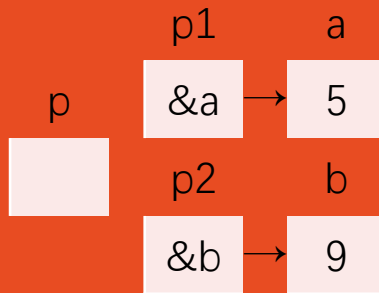
■ 例12-2：输入a和b两个整数，按先大后小的顺序输出a和b

解题思路:不交换整型变量的值，而是交换两个指针变量的值（即a和b的地址）。

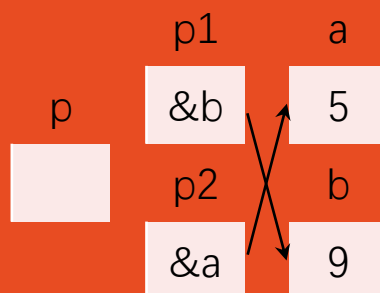
```
1. #include <stdio.h>
2. int main()
3. {
4.     int *p1,*p2,*p,a,b;           //p1,p2的类型是int*类型
5.     printf("please enter two integer numbers:");
6.     scanf("%d,%d",&a,&b);         //输入两个整数
7.     p1=&a;                         //使p1指向变量a
8.     p2=&b;                         //使p2指向变量b
9.     if(a<b) {                     //如果a<b
10.        p=p1;p1=p2;p2=p;          //使p1与p2的值互换
11.    }
12.    printf("a=%d,b=%d\n",a,b);     //输出a,b
13.    printf("max=%d,min=%d\n",*p1,*p2); //输出p1和p2所指向的变量的值
14.    return 0;
15.}
```



交换前



交换后



```
C:\WINDOWS\system32\cmd.exe
please enter two integer numbers:5,9
a=5,b=9
max=9,min=5
请按任意键继续. . .
```

注意

a和b的值并未交换，它们仍保持原值，但p1和p2的值改变了。

12.1 基本概念

■ void

```
void f(void);
```

- void用在函数定义当中可以表示函数没有返回值，或者没有形式参数
- void不能用于定义变量，因为它无法指定要分配的内存空间大小

■ void*

```
void *p = (void *)("abc"); void *str = (void *)(p);
```

- void*表示一个有效的指针，可是数据的类型尚未确定
- void*可以定义变量，占用4个字节（32位平台）或8个字节（64位平台）
- void*类型的指针变量在后续的使用过程一般要强制类型转化

■ NULL

- NULL是强制类型转换为void指针的整数常量0：#define NULL ((void *)0)
- 给指针变量赋值NULL，意味着这个指针不指向任何东西
- 通常用NULL初始化指针变量

```
int *p = NULL;
```

12.2 指针和数组

- 通过指针引用一维数组
- 通过指针引用多维数组
- 通过指针引用字符串
- 指针数组与多重指针

12.2.1 通过指针引用一维数组

■ 数组元素的指针

一个变量有地址，一个数组包含若干元素，每个数组元素都在内存中占用存储单元，它们都有相应的地址。指针变量既然可以指向变量，当然也可以指向数组元素（把某一元素的地址放到一个指针变量中）。所谓数组元素的指针就是数组元素的地址。可以用一个指针变量指向一个数组元素。

```
int a[10]={1,3,5,7,9,11,13,15,17,19}; //定义a为包含10个整型数据的数组
int *p; //定义p为指向整型变量的指针变量
p=&a[0]; //把a[0]元素的地址赋给指针变量p
```



引用数组元素可以用下标法，也可以用指针法，即通过指向数组元素的指针找到所需的元素。

```
p=&a[0]; //p的值是a[0]的地址    ≡    p=a; //p的值是数组a首元素(即a[0])的地址
```

注意

- 程序中的数组名不代表整个数组，只代表数组首元素的地址。

在定义指针变量时可以对它初始化：

```
int *p;
p=&a[0]; //不应写成*p=&a[0];    ≡    int *p=&a[0];    ≡    int *p=a;
```

12.2.1 通过指针引用一维数组

■ 在引用数组元素时指针的运算

在指针已指向一个数组元素时，可以对指针进行以下运算：

- 加一个整数(用+或+=)，如 $p+1$ ，表示指向同一数组中的下一个元素；
- 减一个整数(用-或-=)，如 $p-1$ ，表示指向同一数组中的上一个元素；
- 自加运算，如 $p++$ ， $++p$ ；
- 自减运算，如 $p--$ ， $--p$ 。

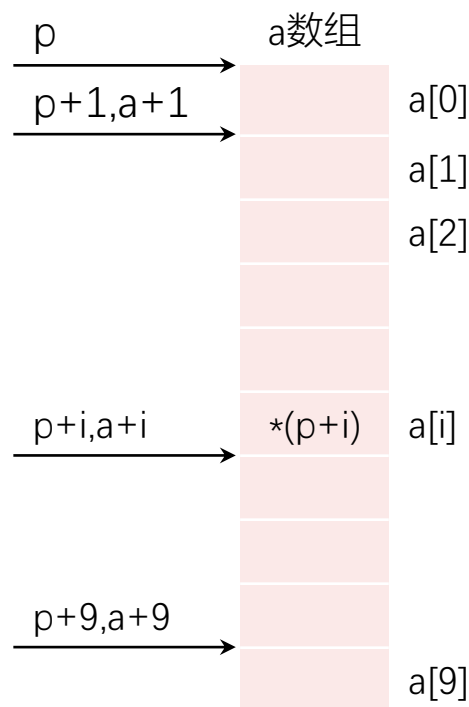
两个指针相减，如 $p1-p2$ (只有 $p1$ 和 $p2$ 都指向同一数组中的元素时才有意义)，结果是两个地址之差除以数组元素的长度。注意：两个地址不能相加，如 $p1+p2$ 是无实际意义的。

如果 p 的初值为 $\&a[0]$ ，则 $p+i$ 和 $a+i$ 就是数组元素 $a[i]$ 的地址，或者说，它们指向 a 数组序号为 i 的元素。

$*(p+i)$ 或 $*(a+i)$ 是 $p+i$ 或 $a+i$ 所指向的数组元素，即 $a[i]$ 。 $[]$ 实际上是变址运算符，即将 $a[i]$ 按 $a+i$ 计算地址，然后找出此地址单元中的值。

注意

- 执行 $p+1$ 时并不是将 p 的值(地址)简单地加1，而是根据定义的基本类型加上一个数组元素所占用的字节数。



12.2.1 通过指针引用一维数组

■ 在引用数组元素时指针的运算

- 两个指针相减和比较通常没有什么用
- 但在两个指针已分别指向同一数组的两个元素时，可用于判断相对顺序

```
1. #include <stdio.h>
2. int main()
3. {
4.     int a[] = {28, 41, 7};
5.     int *p = a;
6.     int *q = a+1;
7.     printf("p-q: %d q-p: %d\n", p-q, q-p);
8. }
```

p-q: -1 q-p: 1

```
1. #include <stdio.h>
2. int main()
3. {
4.     int a[] = {28, 41, 7};
5.     int *p = a;
6.     int *q = a+1;
7.     printf("p>q: %d q>p: %d\n", p>q, q<p);
8. }
```

p>q: 0 q>p: 1

12.2.1 通过指针引用一维数组

■ 例12-3：一个整型数组a有10个元素，要求输出数组中全部元素

①下标法

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

②通过数组名计算数组元素地址，找出元素的值

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

③用指针变量指向数组元素

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

第(1)和第(2)种方法执行效率是相同的。C编译系统是将 $a[i]$ 转换为 $*(a+i)$ 处理的，即先计算元素地址。因此用第(1)和第(2)种方法找数组元素费时较多。

第(3)种方法比第(1)、第(2)种方法快，用指针变量直接指向元素，不必每次都重新计算地址，像 $p++$ 这样的自加操作是比较快的。这种有规律地改变地址值($p++$)能大大提高执行效率。

12.2.1 通过指针引用一维数组

用下标法比较直观，能直接知道是第几个元素。适合初学者使用。

用地址法或指针变量的方法不直观，难以很快地判断出当前处理的是哪一个元素。但用指针变量的方法进行控制，可使程序简洁、高效。

注意

- 在使用指针变量指向数组元素时，有以下几个问题要注意：

(1) 可以通过改变指针变量的值指向不同的元素。

如果不用p变化的方法而用数组名a变化的方法（例如，用a++）行不行呢？

因为数组名a代表数组首元素的地址，它是一个指针型常量，它的值在程序运行期间是固定不变的。既然a是常量，所以a++是无法实现的。

(2) 要注意指针变量的当前值，避免越界读写。

```
for(p=a;a<(p+10);a++)  
    printf("%d",*a);
```

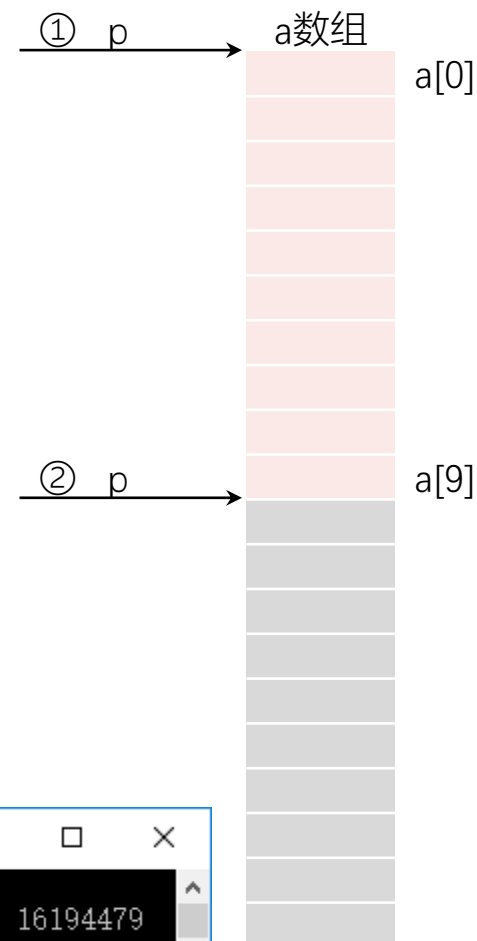


注意：虽然定义数组时指定它包含10个元素，并用指针变量p指向某一数组元素，但实际上p可以指向超过数组界限的存储单元，结果不可预期，应避免出现这样的情况。

12.2.1 通过指针引用一维数组

■ 例12-4：通过指针变量输出整型数组a的10个元素

```
1. #include <stdio.h>
2. int main()
3. {
4.     int *p,i,a[10];
5.     p=a;                                //p指向a[0] ①
6.     for(i=0; i<10; i++)
7.         scanf("%d", p++);              //输入10个整数给a[0]~a[9]
8.     for(i=0; i<10; i++,p++)              //重新使p指向a[0]
9.         printf("%d ", *p);              p = a;
10.    printf("\n");                        //想输出a[0]~a[9] ②
11.    return 0;
12.}
```



```
C:\WINDOWS\system32\cmd.exe
please enter 10 integer numbers:0 1 2 3 4 5 6 7 8 9
-858993460 -858993460 2 -858993460 -858993460 5240960 -858993460 -1395560665 5241052 16194479
请按任意键继续. . .
```

12.2.1 通过指针引用一维数组

- (1) 指向数组元素的指针变量也可以带下标，如 $p[i]$ 。 $p[i]$ 被处理成 $*(p+i)$ ，如果 p 是指向一个整型数组元素 $a[0]$ ，则 $p[i]$ 代表 $a[i]$ 。但是必须弄清楚 p 的当前值是什么？如果当前 p 指向 $a[3]$ ，则 $p[2]$ 并不代表 $a[2]$ ，而是 $a[3+2]$ ，即 $a[5]$ 。
- (2) 利用指针引用数组元素，比较方便灵活，有不少技巧。请分析下面几种情况：

设 p 开始时指向数组 a 的首元素（即 $p=a$ ）：

① $p++;$ //使 p 指向下一元素 $a[1]$
 $*p;$ //得到下一个元素 $a[1]$ 的值

② $*p++;$ /*由于 $++$ 和 $*$ 同优先级，结合方向自右而左，因此它等价于 $*(p++)$ 。先引用 p 的值，实现 $*p$ 的运算，然后再使 p 自增1*/

③ $*(p++);$ //先取 $*p$ 值，然后使 p 加1
 $*(++p);$ //先使 p 加1，再取 $*p$

④ $++(*p);$ /*表示 p 所指向的元素值加1，如果 $p=a$ ，则相当于 $++a[0]$ ，若 $a[0]$ 的值为3，则 $a[0]$ 的值为4。注意：是元素 $a[0]$ 的值加1，而不是指针 p 的值加1*/

⑤ 如果 p 当前指向 a 数组中第 i 个元素 $a[i]$ ，则：

2	++	自增运算符	++变量名 变量名++	右到左
	--	自减运算符	--变量名 变量名--	
	*	取值运算符	*指针变量	
	&	取地址运算符	&变量名	

$*(p--)$ //相当于 $a[i--]$ ，先对 p 进行“ $*$ ”运算，再使 p 自减
 $*(++p)$ //相当于 $a[++i]$ ，先使 p 自加，再进行“ $*$ ”运算
 $*(--p)$ //相当于 $a[--i]$ ，先使 p 自减，再进行“ $*$ ”运算

12.2.2 通过指针引用多维数组

■ 多维数组元素的地址

```
int a[3][4]={{1,3,5,7},{9,11,13,15},{17,19,21,23}};
```

		a[0]	a[0]+1	a[0]+2	a[0]+3
a	a[0]	2000	2004	2008	2012
a+1	a[1]	1	3	5	7
		2016	2020	2024	2028
a+2	a[2]	9	11	13	15
		2032	2036	2040	2044
		17	19	21	23

表示形式	含义	类型	值
a	二维数组名, 指向一维数组a[0], 0行起始地址 (即&a[0])	int ** (严格说是int (*)(4))	地址 2000
a[0], *(a+0), *a	0行0列元素地址 (即&a[0][0])	int *	地址 2000
a+1, &a[1]	指向第1行起始地址	int ** (严格说是int (*)(4))	地址 2016
a[1], *(a+1)	1行0列元素a[1][0]地址	int *	地址 2016
a[1]+2, *(a+1)+2, &a[1][2]	1行2列元素a[1][2]地址	int *	地址 2024
*(a[1]+2), *(*(a+1)+2), a[1][2]	1行2列元素a[1][2]的值	int	元素值 13

C语言的地址信息中既包含位置信息(如内存编号2000), 还包含它所指向的数据的类型信息。

a[0]是一维数组名, 它是一维数组中起始元素的地址; a是二维数组名, 它是二维数组的首行起始地址。

二者的纯地址是相同的, 但它们的基类型不同, 即它们指向的数据的类型不同, 前者是整型数据, 后者是一维数组。

12.2.2 通过指针引用多维数组

■ 例12-5：输出二维数组的有关数据(地址和元素的值)

```
1. #include <stdio.h>
2. int main()
3. {
4.     int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
5.     printf("%p,%x\n",a,*a);           //0行起始地址和0行0列元素地址
6.     printf("%p,%x\n",a[0],*(a+0));    //0行0列元素地址
7.     printf("%p,%x\n",&a[0],&a[0][0]); //0行起始地址和0行0列元素地址
8.     printf("%p,%x\n",a[1],a+1);       //1行0列元素地址和1行起始地址
9.     printf("%p,%x\n",&a[1][0],*(a+1)+0); //1行0列元素地址
10.    printf("%p,%x\n",a[2],*(a+2));     //2行0列元素地址
11.    printf("%p,%x\n",&a[2],a+2);       //2行起始地址
12.    printf("%d,%d\n",a[1][0],*(*(a+1)+0)); //1行0列元素的值
13.    printf("%d,%d\n",*a[2],*(*(a+2)+0)); //2行0列元素的值
14.    return 0;
15.}
```

注：格式控制符“%p”中的p是pointer（指针）的缩写。指针的值是语言实现（编译程序）相关的。但是几乎所有实现中，指针的值都是一个表示地址空间中某个存储器单元的整数。printf函数族中对于%p一般以十六进制整数方式输出指针的值。

```
000000000061FDE0,61fde0
000000000061FDE0,61fde0
000000000061FDE0,61fde0
000000000061FDF0,61fdf0
000000000061FDF0,61fdf0
000000000061FE00,61fe00
000000000061FE00,61fe00
9,9
17,17
```

12.2.2 通过指针引用多维数组

■ 例12-6：有一个3×4的二维数组，要求用指向元素的指针变量输出二维数组各元素的值

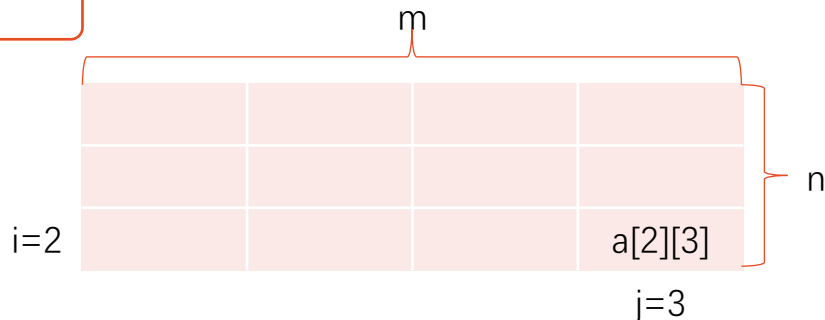
```
1. #include <stdio.h>
2. int main()
3. {
4.     int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
5.     int *p;                                //p是int *型指针变量
6.     for(p=a[0];p<a[0]+12;p++)             //使p依次指向下一个元素
7.     {
8.         if((p-a[0])%4==0) printf("\n");    //p移动4次后换行
9.         printf("%4d",*p);                  //输出p指向的元素的值
10.    }
11.    printf("\n");
12.    return 0;
13.}
```



p是一个int *型(指向整型数据)的指针变量，它可以指向一般的整型变量，也可以指向整型的数组元素。每次使p值加1，使p指向下一元素。

如果要输出某个指定的数值元素（例如a[2][3]），则应事先计算该元素在数组中的相对位置（即相对于数组起始位置的相对位移量）。计算a[i][j]在数组中的相对位置的计算公式为： $i*m + j$ ，其中，m为二维数组的列数（二维数组大小为n×m）。

```
C:\WINDOWS\system32\cmd.exe
1  3  5  7
9 11 13 15
17 19 21 23
请按任意键继续...
```



12.2.2 通过指针引用多维数组

■ 指向由m个元素组成的一维数组的指针变量

`int(*p)[4]`: 定义一个由4个元素组成的一维数组的指针变量`p`。()`优先级高`，说明`p`是一个指针，指向一个整型的一维数组，这个一维数组的元素个数是4。`p`的类型是`int (*)[4]`，基类型是`int[4]`。`p`的步长是4，也就是说执行`p+1`时，`p`要跨过4个整型数据的长度。

`(p+2)+3`: 括号中的加2是以`p`的基类型(一维整型数组)的长度为单位，即`p`每加1，地址就增加16个字节（4个元素，每个元素4个字节）；`(p+2)`之后基类型不变，再加3相当于总共加了5个基类型的长度。综上，`(p+2)+3`的地址比`p`的地址增加了80个字节。

`*(p+2)+3`: 由于经过`*(p+2)`的运算，得到`a[2]`，即`&a[2][0]`，它已转化为指向列元素的指针了，故加3是以元素的长度为单位的，加3就是加 (3×4) 个字节。综上，`*(p+2)+3`的地址比`p`的地址增加了44字节。

虽然`p+2`和`*(p+2)`具有相同的值，但由于它们所指向的对象的长度不同，因此`(p+2)+3`和`*(p+2)+3`的值就不相同了。

12.2.2 通过指针引用多维数组

■ 例12-7：指向由m个元素组成的一维数组的指针变量

```
1. #include <stdio.h>
2. int main()
3. {
4.     int aa[4]={1,3,5,7};           //定义一维数组aa，包含4个元素
5.     int (*p)[4];                   //定义指向包含4个元素的一维数组的指针变量
6.     p=&aa;                          //使p指向一维数组
7.     printf("%d\n", (*p)[3]);        //输出a[3]，输出整数7
8.     printf("%d %d\n", sizeof(p), sizeof(aa), sizeof(&aa[0]));
9.     return 0;
10.}
```

7

8 16 8



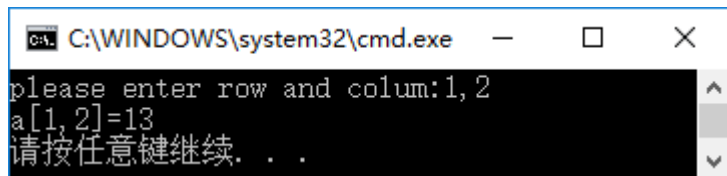
int (*p)[4]; 表示(*p)有4个元素，每个元素为整型。也就是p所指的對象是有4个整型元素的数组，即p是指向一维数组的指针。

应该记住，此时p只能指向一个包含4个元素的一维数组，不能指向一维数组中的某一个元素。p的值是该一维数组的起始地址。虽然这个地址与该一维数组首元素的地址相同，但它们的基类型是不同的。

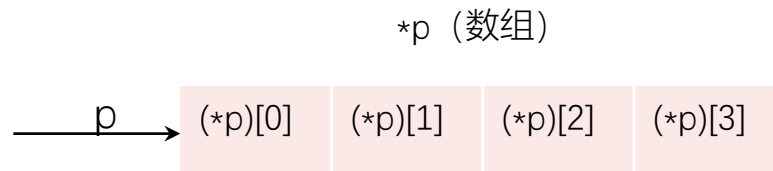
12.2.2 通过指针引用多维数组

■ 例12-8：输出二维数组任一行任一列元素的值

```
1. #include <stdio.h>
2. int main()
3. {
4.     int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23}; //定义二维数组a
5.     int (*p)[4],i,j; //指针变量p指向包含4个整型元素的一维数组
6.     p=a; //p指向二维数组的0行
7.     printf("please enter row and colum:");
8.     scanf("%d,%d",&i,&j); //输入要求输出的元素的行列号
9.     printf("a[%d,%d]=%d\n",i,j,*(*(p+i)+j)); //输出a[i][j]的值
10.    return 0;
11.}
```



```
C:\WINDOWS\system32\cmd.exe
please enter row and colum:1,2
a[1,2]=13
请按任意键继续. . .
```



12.2.3 通过指针引用字符串

■ 字符串的引用方式

在C语言中只有字符变量，没有字符串变量。

```
char *string="I love China!";
```

≡

```
char *string;           //定义一个char *型变量
string="I love China!";
//把字符串第1个元素的地址赋给字符指针变量string
```

注意

- string被定义为一个指针变量，基类型为字符型。它只能指向一个字符类型数据，而不能同时指向多个字符数据，更不是把"I love China!"这些字符存放到string中（指针变量只能存放地址），也不是把字符串赋给*string。只是把"I love China!"的第1个字符的地址赋给指针变量string。

可以对指针变量进行再赋值，如：

```
string="I am a student."; //对指针变量string重新赋值
```

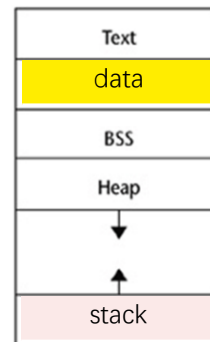
可以通过字符指针变量输出它所指向的字符串，如：

```
printf("%s\n",string); //%s可对字符串进行整体的输入输出
```

%s是输出字符串时所用的格式符，在输出项中给出字符指针变量名string，则系统会输出string所指向的字符串第1个字符，然后自动使string加1，使之指向下一个字符，再输出该字符……如此直到遇到字符串结束标志'\0'为止。注意，在内存中，字符串的最后被自动加了一个'\0'。

12.2.3 通过指针引用字符串

■ 例12-9：字符数组与字符指针变量



```
1. #include <stdio.h>
2. int main()
3. {
4.     char string1[]={'I',' ','l','o','v','e',' ','C','h','i','n','a','!'};    //定义字符数组string
5.     printf("%s\n",string1);          //用%s格式声明输出string，可以输出整个字符串
6.     printf("%c\n",string1[7]);       //用%c格式输出一个字符数组元素
7.     printf("%p %u\n", string1, sizeof(string1));          I love China!
8.     return 0;          C
9. }          0xffa0198f 13
```

```
1. #include <stdio.h>
2. int main()
3. {
4.     char string2[]="I love China!"; //定义字符数组string
5.     printf("%s\n",string2);          //用%s格式声明输出string，可以输出整个字符串
6.     printf("%c\n",string2[7]);       //用%c格式输出一个字符数组元素
7.     printf("%p %u\n", string2, sizeof(string2));          I love China!
8.     return 0;          C
9. }          0xffef363e 14
```

```
1. #include <stdio.h>
2. int main()
3. {
4.     char *string3="I love China!"; //定义字符指针变量string并初始化
5.     printf("%s\n",string3);          //输出字符串
6.     printf("%c\n",string3[7]);       //用%c格式输出一个字符数组元素
7.     printf("%p %p %u\n", &string3, string3, sizeof(string3));
8.     return 0;          I love China!
9. }          C
          0xffcafc8 0x565fb008 4
```

string1 →

I	string[0]
	string[1]
I	string[2]
o	string[3]
v	string[4]
e	string[5]
	string[6]
C	string[7]
h	string[8]
i	string[9]
n	string[10]
a	string[11]
!	string[12]

string2 →

I
I
o
v
e
C
h
i
n
a
!
\0

string(7) →

string3 →

I
I
o
v
e
C
h
i
n
a
!
\0

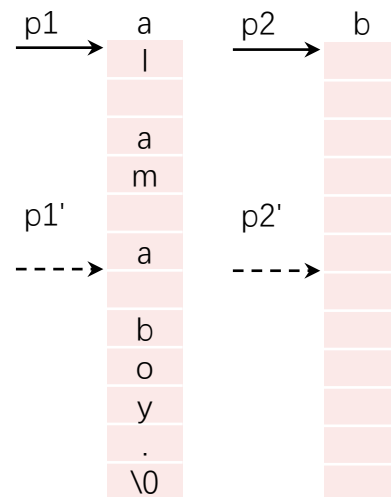
12.2.3 通过指针引用字符串

■ 例12-10：字符数组/字符串的复制

```
1. #include <stdio.h>
2. int main()
3. {
4.     char a[]="I am a boy.", b[20];    //定义字符数组
5.     int i;
6.     for(i=0; a[i]!='\0';i++)
7.         b[i]=a[i];                    //将a[i]的值赋给b[i]
8.     b[i]='\0';                        //在b数组的有效字符之后加'\0'
9.     printf("string a is:%s\n",a);
10.    printf("string b is:");
11.    for(i=0;b[i]!='\0';i++)
12.        printf("%c",b[i]);
13.    printf("\n");
14.    return 0;
15.}
```

```
1. #include <stdio.h>
2. int main() {
3.     char a[]="I am a boy.", b[20];
4.     char *p1,*p2;
5.     p1=a; //p1指向a数组中的第0个元素
6.     p2=b; //p2指向b数组中的第0个元素
7.     for(;*p1!='\0';p1++,p2++)    //p1,p2每次自加1
8.         *p2=*p1; //将p1所指向的元素的值赋给p2所指向的元素
9.     *p2='\0';                    //在复制全部有效字符后加'\0'
10.    printf("string a is:%s\n",a); //输出a数组中的字符
11.    printf("string b is:%s\n",b); //输出b数组中的字符
12.    return 0;
13.}
```

对字符串中字符的存取，可以用下标方法，也可以用指针方法。



12.2.3 通过指针引用字符串

■ 使用字符指针变量和字符数组的比较

(1) 字符数组由若干个元素组成，每个元素中放一个字符，而字符指针变量中存放的是地址(字符串第0个字符的地址)，绝不是将字符串放到字符指针变量中。

(2) 赋值方式。可以对字符指针变量赋值，但不能对数组名赋值。(数组名是常量)

(3) 初始化的含义。

```
char *a="I love China!";
```

≡

```
char *a;
```

```
a="I love China!";
```

```
char str[14]="I love China!";
```

≠

```
char str[14];
```

```
str[]="I love China!";
```



(4) 存储单元的内容。编译时为字符数组分配若干存储单元，以存放各元素的值，而对字符指针变量，只分配一个存储单元。

```
char *a;
```

```
scanf("%s",a);
```



```
char *a,str[10];
```

```
a=str; scanf("%s",a);
```



(5) 指针变量的值是可以改变的，而字符数组名代表一个固定的值(数组首元素的地址)，不能改变。

(6) 字符数组中各元素的值是可以改变的(可以对它们再赋值)，但字符指针变量指向的字符串常量中的内容是不可以被取代的(不能对它们再赋值)。

```
char a[]="House";
```

```
a[2]='r';
```



```
char *b="House";
```

```
b[2]='r';
```



(7) 引用数组元素。对字符数组可以用下标法(用数组名和下标)引用一个数组元素(如a[5])，也可以用地址法(如*(a+5))引用数组元素a[5]。如果定义了字符指针变量p，并使它指向数组a的首元素，则可以用指针变量带下标的形式引用数组元素(如p[5])，同样，可以用地址法(如*(p+5))引用数组元素a[5]。

```
char *format="a=%d,b=%f\n";
```

```
printf(format,a,b);
```

(8) 用指针变量指向一个格式字符串，可以用它代替printf函数中的格式字符串。

12.2.3 通过指针引用字符串

■ 例12-11：改变指针变量的值

```
1. #include <stdio.h>
2. int main()
3. {
4.     char *a="I love China!";
5.     a=a+7;                //改变指针变量的值，即改变指针变量的指向
6.     printf("%s\n",a);    //输出从a指向的字符开始的字符串
7.     return 0;
8. }
```



```
1. #include <stdio.h>
2. int main()
3. {
4.     char str[]={"I love China!";
5.     str=str+7;
6.     printf("%s\n",str);
7.     return 0;
8. }
```



指针变量a的值是可以变化的。printf函数输出字符串时，从指针变量a当时所指向的元素开始，逐个输出各个字符，直到遇'\0'为止。而数组名虽然代表地址，但它是常量，它的值是不能改变的。

指针变量的值是可以改变的，而字符数组名代表一个固定的值(数组首元素的地址)，不能改变。

12.2.4 指针数组与多重指针

定义一维指针数组的一般形式为

类型名 *数组名[数组长度];

int *p[4];

一个数组，若其元素均为指针类型数据，称为**指针数组**，也就是说，指针数组中的每一个元素都存放一个地址，相当于一个指针变量。

int *p[4]: 定义一个拥有4个元素的指针数组。[]优先级高，先与p结合称为一个数组，再由int *说明这是一个整型指针数组。

```
1. int *p[4];  
2. int a[4][4];  
  
3. for (int i = 0; i < 4; i++) p[i] = a[i];  
4. printf("%d\n", sizeof(p));
```

指针数组比较适合用来指向若干个字符串，使字符串处理更加方便灵活。

12.2.4 指针数组与多重指针

■ 例12-12：将若干字符串按字母顺序（由小到大）输出

排序前		指针数组	字符串	
	name[0]	→	Follow me	F o l l o w m e \0
	name[1]	→	BASIC	B A S I C \0
	name[2]	→	Great Wall	G r e a t W a l l \0
	name[3]	→	FORTRAN	F O R T R A N \0
	name[4]	→	Computer design	C o m p u t e r d e s i g n \0

指向互换

排序后		指针数组	字符串	
	name[0]	↗	Follow me	F o l l o w m e \0
	name[1]	↘	BASIC	B A S I C \0
	name[2]	↗	Great Wall	G r e a t W a l l \0
	name[3]	↘	FORTRAN	F O R T R A N \0
	name[4]	↗	Computer design	C o m p u t e r d e s i g n \0

12.2.4 指针数组与多重指针

■ 例12-12：将若干字符串按字母顺序（由小到大）输出

```
1. #include <stdio.h>
2. #include <string.h>

3. void sort(char *name[],int n);      //函数声明
4. void print(char *name[],int n);    //函数声明

5. int main()
6. {
7.     char *name[] = {
8.         "Follow me", "BASIC", "Great Wall",
9.         "FORTRAN","Computer design"
10.    }; //定义指针数组，它的元素分别指向5个字符串
11.    int n=5;
12.    sort(name,n); //调用sort函数，对字符串排序
13.    print(name,n); //调用print函数，输出字符串
14.    return 0;
15.}

16.void print(char *name[],int n) //定义print函数
17.{ int i;
18. for(i=0;i<n;i++) printf("%s\n",name[i]);
19.}
```

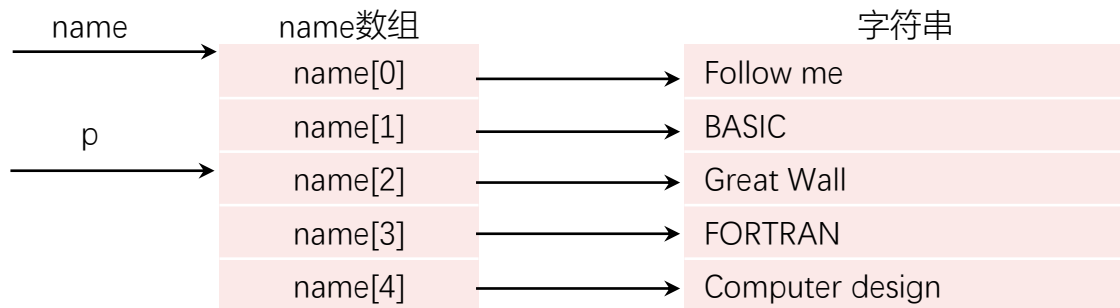
```
20.void sort(char *name[],int n)      //定义sort函数
21.{
22.    char *temp;
23.    int i,j,min;

24.    for(i=0;i<n-1;i++)              //用选择法排序
25.    {
26.        min = i;
27.        for(j=i+1; j<n; j++)
28.            if (strcmp(name[min],name[j])>0) min = j;
29.        if(k!=i)
30.        {
31.            temp=name[i];
32.            name[i]=name[k];
33.            name[k]=temp;}
34.    }
35. }
36.}
```


12.2.4 指针数组与多重指针

■ 指向指针数据的指针变量

在了解了指针数组的基础上，需要了解**指向指针数据的指针变量**，简称为**指向指针的指针**。



`name`是一个指针数组，它的每一个元素是一个指针型的变量，其值为地址。`name`既然是一个数组，它的每一元素都应有相应的地址。数组名`name`代表该指针数组首元素的地址。`name+i`是`name[i]`的地址。`name+i`就是指向指针型数据的指针。还可以设置一个指针变量`p`，它指向指针数组的元素。`p`就是指向指针型数据的指针变量。

定义一个指向指针数据的指针变量：

```
char **p;
```

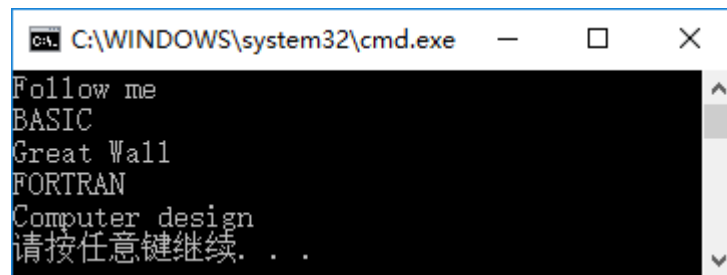
`p`的前面有两个`*`号。`p`指向一个字符指针变量（这个字符指针变量指向一个字符型数据）。如果引用`*p`，就得到`p`所指向的字符指针变量的值。

```
p=name+2;  
printf("%d\n",*p);           //name[2]的值（它是一个地址）  
printf("%s\n",*p);           //以字符串形式(%s)输出字符串"Great Wall"
```

12.2.4 指针数组与多重指针

■ 例12-13：使用指向指针数据的指针变量

```
1. #include <stdio.h>
2. int main()
3. {
4.     char *name[]={"Follow me","BASIC","Great Wall","FORTRAN","Computer design"};
5.     char **p;
6.     int i;
7.     for(i=0;i<5;i++)
8.     {
9.         p=name+i;
10.        printf("%s\n",*p);
11.    }
12.    return 0;
13. }
```



```
C:\WINDOWS\system32\cmd.exe
Follow me
BASIC
Great Wall
FORTRAN
Computer design
请按任意键继续. . .
```



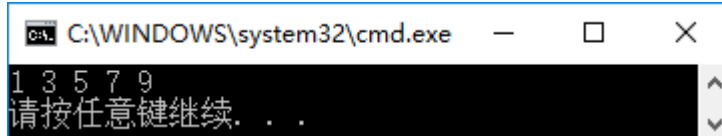
p是指向char*型数据的指针变量，即指向指针的指针。在第1次执行for循环体时，赋值语句“p=name+i;”使p指向name数组的0号元素name [0]，*p是name [0] 的值，即第1个字符串首字符的地址，用printf函数输出第1个字符串（格式符为%s）。执行5次循环体，依次输出5个字符串。

指针数组的元素也可以不指向字符串，而指向整型数据或实型数据等。

12.2.4 指针数组与多重指针

■ 例12-14：有一个指针数组，其元素分别指向一个整型数组的元素，用指向指针数据的指针变量，输出整型数组各元素的值

```
1. #include <stdio.h>
2. int main()
3. {
4.     int a[5]={1,3,5,7,9};
5.     int *num[5]={&a[0],&a[1],&a[2],&a[3],&a[4]};
6.     int **p,i;           //p是指向指针型数据的指针变量
7.     p=num;               //使p指向num[0]
8.     for(i=0;i<5;i++)
9.     {
10.         printf("%d ",**p);
11.         p++;
12.     }
13.     printf("\n");
14.     return 0;
15. }
```



C:\WINDOWS\system32\cmd.exe

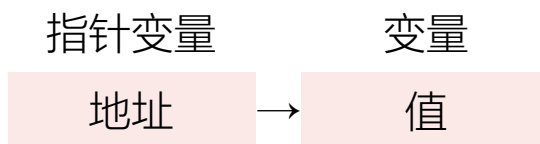
```
1 3 5 7 9
请按任意键继续. . .
```

12.2.4 指针数组与多重指针

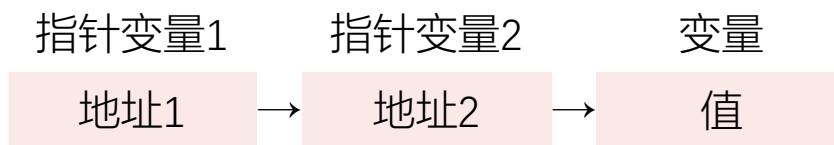
■ 指向指针数据的指针变量

利用指针变量访问另一个变量就是“间接访问”。

如果在一个指针变量中存放一个目标变量的地址，这就是“单级间址”；



指向指针数据的指针用的是“二级间址”方法；



从理论上说，间址方法可以延伸到更多的级，即多重指针。



12.3 指针与结构体

- 结构体指针：指向结构体变量的指针。
- 一个结构体变量的起始地址就是这个结构体变量的指针。
- 如果把一个结构体变量的起始地址存放在一个指针变量中，这个指针变量就指向该结构体变量。

12.3 指针与结构体

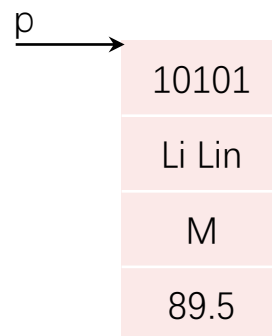
■ 例12-15：通过结构体指针变量输出结构体变量中成员的信息

```
1. #include <stdio.h>
2. #include <string.h>

3. typedef struct {           //声明结构体类型Student
4.     long num;
5.     char name[20];
6.     char sex;
7.     float score;
8. } Student;

9. int main() {
10.     Student stu;           //定义Student类型的变量stu
11.     Student *p;            //定义指向Student 类型数据的指针变量p
12.     p=&stu;                 //p指向stu
13.     stu.num=10101;          //对结构体变量的成员赋值
14.     strcpy(stu.name,"Li Lin"); //用字符串复制函数给stu.name赋值
15.     stu.sex='M';
16.     stu.score=89.5;

17.     printf("No.:%ld\nname:%s\nsex:%c\nscore:%5.1f\n", stu.num, stu.name, stu.sex, stu.score);
18.     printf("\nNo.:%ld\nname:%s\nsex:%c\nscore:%5.1f\n", (*p).num, (*p).name, (*p).sex, (*p).score);
19.     return 0;
20. }
```



如果p指向一个结构体变量stu，以下3种用法等价：

① stu.成员名

stu.num

② (*p).成员名

(*p).num

③ p->成员名

p->num

(*p).num也可表示为p->num

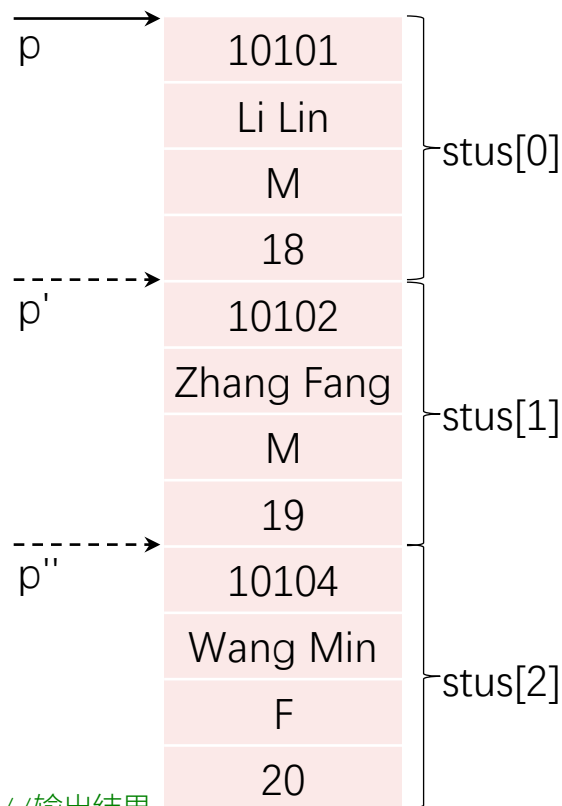
12.3 指针与结构体

■ 例12-16：有3个学生的信息，放在结构体数组中，要求输出全部学生的信息

```
1. #include <stdio.h>
2. typedef struct      //声明结构体类型Student
3. {
4.     int num;
5.     char name[20];
6.     char sex;
7.     int age;
8. } Student;

9. struct Student stus[3]={ //定义结构体数组stus并初始化
10.     {10101,"Li Lin",'M',18},
11.     {10102,"Zhang Fang",'M',19},
12.     {10104,"Wang Min",'F',20}
13. };

14. int main() {
15.     Student *p;           //定义指向Student结构体变量的指针变量
16.     printf(" No. Name      sex age\n");
17.     for (p=stus; p<stus+3; p++)
18.         printf("%5d %-20s %2c %4d\n",p->num, p->name, p->sex, p->age); //输出结果
19.     return 0;
20. }
```



12.4 指针与函数

- 指针变量作为函数参数
- 数组名作函数参数
- 指向数组的指针作函数参数
- 字符指针作为函数参数
- 返回指针的函数
- 指向函数的指针

12.4.1 指针变量作为函数参数

函数的调用可以（而且只可以）得到一个返回值（即函数值），而使用指针变量作参数，可以得到多个变化了的值。如果不用指针变量是难以做到这一点的。要善于利用**指针法**。

如果想通过函数调用得到n个要改变的值，可以这样做：

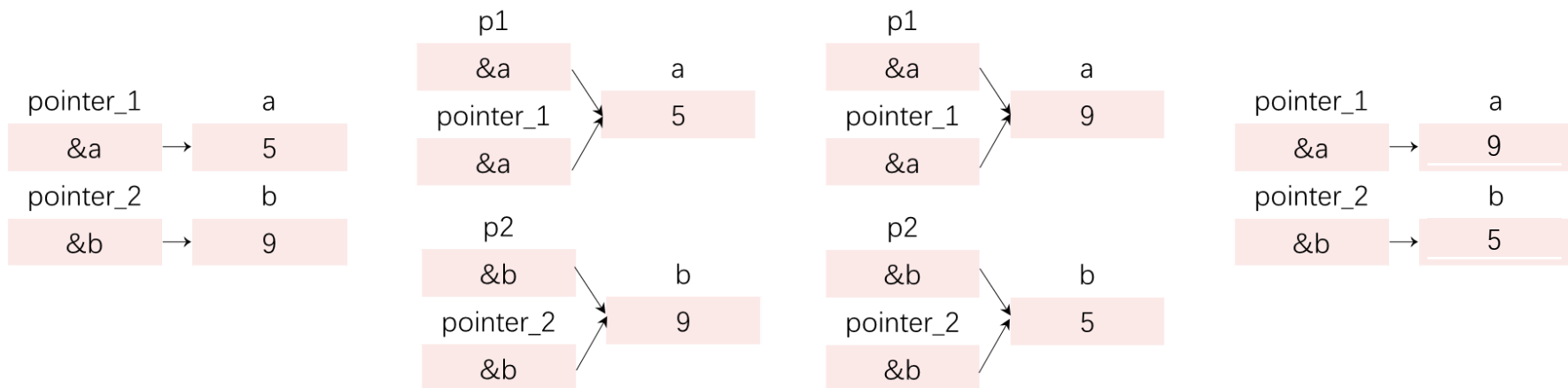
- ① 在主调函数中设n个变量，用n个指针变量指向它们；
 - ② 设计一个函数，有n个指针形参。在这个函数中改变这n个形参的值；
 - ③ 在主调函数中调用这个函数，在调用时将这n个指针变量作实参，将它们的值，也就是相关变量的地址传给该函数的形参；
 - ④ 在执行该函数的过程中，通过形参指针变量，改变它们所指向的n个变量的值；
 - ⑤ 主调函数中就可以使用这些改变了值的变量。
-

12.4.1 指针变量作为函数参数

■ 例12-17：使用函数处理，对输入的两个整数按大小顺序输出

```
int main() {  
    int a,b;  
    int *pointer_1,*pointer_2; //定义两个int *型的指针变量  
    scanf("%d,%d",&a,&b);      //输入两个整数  
    pointer_1=&a;              //使pointer_1指向a  
    pointer_2=&b;              //使pointer_2指向b  
    if (a<b)  
        swap(pointer_1,pointer_2); //如果a<b, 调用swap函数  
    printf("max=%d,min=%d\n",a,b); //输出结果  
    return 0;  
}
```

```
void swap(int *p1,int *p2)  
{  
    int temp;  
    temp=*p1;                //使*p1和*p2互换  
    *p1=*p2;  
    *p2=temp;  
} //本例交换a和b的值，而p1和p2的值不变。
```



12.4.1 指针变量作为函数参数

■ 例12-18：使用函数处理，对输入的两个整数按大小顺序输出

```
1. void swap(int *p1,int *p2)
2. {
3.     int temp;
4.     temp=*p1;
5.     *p1=*p2;
6.     *p2=temp;
7. }
```



```
1. void swap(int *p1,int *p2)
2. {
3.     int *temp;
4.     *temp=*p1;
5.     *p1=*p2;
6.     *p2=*temp;
7. }
```



```
1. void swap(int x,int y)
2. {
3.     int temp;
4.     temp=x;
5.     x=y;
6.     y=temp;
7. }
```



*p1就是a，是整型变量。而*temp是指针变量temp所指向的变量。但由于未给temp赋值，因此temp中并无确定的值(它的值是不可预见的)，所以temp所指向的单元也是不可预见的。

所以，对*temp赋值就是向一个未知的存储单元赋值，而这个未知的存储单元中可能存储着一个有用的数据，这样就有可能破坏系统的正常工作状况。

在函数调用时，a的值传送给x，b的值传送给y。执行完swap函数后，x和y的值是互换了，但并未影响到a和b的值。在函数结束时，变量x和y释放了，main函数中的a和b并未互换。

a	b	a	b
5	9	5	9
↓	↓		
5	9	9	5
x	y	x	y

12.4.2 数组名作函数参数

以变量名和数组名作为函数参数的比较

实参类型	变量名	数组名
要求形参的类型	变量名	数组名或指针变量
传递的信息	变量的值	实参数组首元素的地址
通过函数调用能否改变实参的值	不能改变实参变量的值	能改变实参数组元素的值

C语言调用函数时虚实结合的方法都是采用“**值传递**”方式，当用变量名作为函数参数时传递的是变量的值，当用数组名作为函数参数时，由于数组名代表的是数组首元素地址，因此传递的值是地址，所以要求形参为指针变量。

注意

- 实参数组名代表一个固定的地址，或者说是指针常量，但形参数组名并不是一个固定的地址，而是按指针变量处理。

在函数调用进行虚实结合后，形参的值就是实参数组首元素的地址。

在函数执行期间，它可以再被赋值。

```
void fun (arr[ ],int n)
{
    printf("%d\n", *arr); //输出array[0]的值
    arr=arr+3;           //形参数组名可以被赋值
    printf("%d\n", *arr); //输出array[3]的值
}
```

12.4.2 数组名作函数参数

如果有一个实参数组，要想在函数中改变此数组中的元素的值，实参与形参的对应关系有以下4种情况。

- ① 形参和实参都用数组名
- ② 实参用数组名，形参用指针变量
- ③ 实参形参都用指针变量
- ④ 实参为指针变量，形参为数组名

①

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

int f(int x[], int n)
{
    :
}
```

②

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

int f(int *x, int n)
{
    :
}
```

③

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

int f(int *x, int n)
{
    :
}
```

④

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

int f(int x[], int n)
{
    :
}
```

12.4.2 数组名作函数参数

■ 例12-19：将数组a中n个整数按相反顺序存放

```
1. #include <stdio.h>
2. void inv(int x[], int n);    //inv函数声明
3. int main()
4. {
5.     int i,a[10]={3,7,9,11,0,6,7,5,4,2};
6.     printf("The original array:\n");
7.     for(i=0;i<10;i++)
8.         printf("%d ",a[i]); //输出未交换时数组各元素的值
9.     printf("\n");
10.    inv(a,10);                //调用inv函数, 进行交换
11.    printf("The array has been inverted:\n");
12.    for(i=0;i<10;i++)
13.        printf("%d ",a[i]);  //输出交换后数组各元素的值
14.    printf("\n");
15.    return 0;
16.}

17.void inv(int x[], int n)      //形参x是数组名
18.{
19.    int temp,i,j,m=(n-1)/2;
20.    for(i=0;i<=m;i++)
21.    {
22.        j=n-1-i;
23.        temp=x[i]; x[i]=x[j]; x[j]=temp; //交换x[i]和x[j]
24.    }
25.    return;
26.}
```

① 形参和实参都用数组名

```
1. #include <stdio.h>
2. void inv(int x[], int n);    //inv函数声明
3. int main()
4. {
5.     void inv(int *x,int n);
6.     int i,a[10]={3,7,9,11,0,6,7,5,4,2};
7.     printf("The original array:\n");
8.     for(i=0;i<10;i++)
9.         printf("%d ",a[i]);
10.    printf("\n");
11.    inv(a,10);
12.    printf("The array has been inverted:\n");
13.    for(i=0;i<10;i++)
14.        printf("%d ",a[i]);
15.    printf("\n");
16.    return 0;
17.}

18.void inv(int *x,int n)      //形参x是指针变量
19.{
20.    int *p,temp,*i,*j,m=(n-1)/2;
21.    i=x; j=x+n-1; p=x+m;
22.    for(;i<=p;i++,j--) {
23.        temp=*i; *i=*j; *j=temp;    // *i与*j交换
24.    }
25.    return;
26.}
```

② 实参用数组名, 形参用指针变量

12.4.2 数组名作函数参数

■ 例12-20：用指针方法对10个整数按由大到小顺序排序（选择排序法）

```
1. #include <stdio.h>
2. void sort(int x[],int n); //sort函数声明
3. int main() {
4.     int i,*p,a[10];
5.     p=a;                //指针变量p指向a[0]
6.     printf("please enter 10 integer numbers:");
7.     for(i=0;i<10;i++) scanf("%d",p++);    //输入10个整数
8.     p=a; //指针变量p重新指向a[0]
9.     sort(p,10);         //调用sort函数, 实参用指针变量
10.    for(p=a,i=0;i<10;i++) {
11.        printf("%d ",*p);    //输出排序后的10个数组元素
12.        p++;
13.    }
14.    printf("\n");
15.    return 0;
16.}
```

```
17.void sort(int *x,int n) //定义sort函数, 形参是指针变量
18.{
19.    int i,j,k,t;
20.    for (i=0;i<n-1;i++) {
21.        k=i;
22.        for (j=i+1;j<n;j++)
23.            if (* (x+j) > * (x+k)) k=j;
24.        if (k!=i) {
25.            t=* (x+i); * (x+i)=* (x+k); * (x+k)=t;
26.        }
27.    }
28.}
```

③ 实参形参都用指针变量

```
1. #include <stdio.h>
2. void sort(int x[],int n); //sort函数声明
3. int main() {
4.     int i,*p,a[10];
5.     p=a;                //指针变量p指向a[0]
6.     printf("please enter 10 integer numbers:");
7.     for(i=0;i<10;i++) scanf("%d",p++);    //输入10个整数
8.     p=a; //指针变量p重新指向a[0]
9.     sort(p,10);         //调用sort函数, 实参用指针变量
10.    for(p=a,i=0;i<10;i++) {
11.        printf("%d ",*p);    //输出排序后的10个数组元素
12.        p++;
13.    }
14.    printf("\n");
15.    return 0;
16.}
```

```
17.void sort(int x[],int n) //定义sort函数, 形参是数组名
18.{
19.    int i,j,k,t;
20.    for(i=0;i<n-1;i++) {
21.        k=i;
22.        for(j=i+1;j<n;j++) if(x[j]>x[k]) k=j;
23.        if(k!=i) {    t=x[i]; x[i]=x[k]; x[k]=t;}
24.    }
25.}
```

④ 实参为指针变量，形参为数组名

12.4.3 指向数组的指针作函数参数

- 一维数组名可以作为函数参数，多维数组名也可作函数参数。

用指针变量作形参，以接受实参数组名传递来的地址

- 方法1：用指向变量的指针变量
- 方法2：用指向一维数组的指针变量
- 注意：上述两种方法的指针形参基类型不一样

12.4.3 指向数组的指针作函数参数

■ 例12-21：有一个班，3个学生，各学4门课，计算总平均分数以及第n个学生的成绩

```
1. #include <stdio.h>
2. void average(float *p,int n);
3. void search(float (*p)[4],int n);

4. void average(float *p,int n)    //定义求平均成绩的函数
5. {
6.     float *p_end;
7.     float sum=0,aver;
8.     p_end = p+n-1;

9.     //n的值为12时, p_end的值是p+11, 指向最后一个元素
10.    for(; p<=p_end; p++)
11.        sum=sum+(*p);

12.    aver=sum/n;
13.    printf("average=%5.2f\n",aver);
14.}
```

```
15. void search(float (*p)[4],int n) {
16.    //p是指向具有4个元素的一维数组的指针
17.    int i;
18.    printf("The score of No.%d are:\n",n);
19.    for(i=0;i<4;i++)
20.        printf("%5.2f ",*(*(p+n)+i));
21.    printf("\n");
22.}

23. int main() {
24.    float score[3][4]={
25.        {65,67,70,60},{80,87,90,81},{90,99,100,98}
26.    };
27.    average(*score,12);           //求12个分数的平均分
28.    search(score,2);              //求序号为2的学生的成绩
29.    return 0;
30.}
```

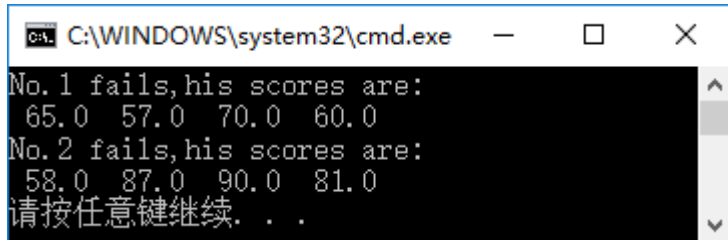
注意

- 实参与形参如果是指针类型，应当注意它们的**基类型必须一致**。不应把int *类型的指针(即数组元素的地址)传给int (*)[4] 型(指向一维数组)的指针变量，反之亦然。

12.4.3 指向数组的指针作函数参数

■ 例12-22：有一个班，3个学生，各学4门课，查找有一门以上课程不及格的学生，输出他们的全部课程的成绩

```
1. #include <stdio.h>
2. void search(float (*p)[4],int n); //函数声明
3. int main()
4. {
5.     float score[3][4]={
6.         {65,57,70,60},{58,87,90,81},{90,99,100,98}
7.     }; //定义二维数组函数score
8.     search(score,3); //调用search函数
9.     return 0;
10. }
```



```
C:\WINDOWS\system32\cmd.exe
No.1 fails,his scores are:
65.0 57.0 70.0 60.0
No.2 fails,his scores are:
58.0 87.0 90.0 81.0
请按任意键继续. . .
```

```
11. void search(float (*p)[4],int n)
12. //形参p是指向包含4个float型元素的一维数组的指针变量
13. {
14.     int i,j,flag;
15.     for(j=0;j<n;j++) {
16.         flag=0;
17.         for(i=0;i<4;i++) /*(* (p+j)+i)就是score[j][i]
18.             if(* (* (p+j)+i)<60) flag=1;
19.         if(flag==1) {
20.             printf("No.%d fails,his scores are:\n",j+1);
21.             for(i=0;i<4;i++) //输出score[j][i]的值
22.                 printf("%5.1f ",* (* (p+j)+i));
23.             printf("\n");
24.         }
25.     }
26. }
```

12.4.4 字符指针作为函数参数

字符指针作为函数参数时，实参与形参的类型有以下几种对应关系：

实参	形参
字符数组名	字符数组名
字符数组名	字符指针变量
字符指针变量	字符指针变量
字符指针变量	字符数组名

12.4.4 字符指针作为函数参数

■ 例12-23：用函数调用实现字符串的复制。(1) 用字符数组名作为形参和实参

```
1. #include <stdio.h>
2. void copy_string(char from[], char to[]);

3. int main()
4. {
5.     char a[] = "I am a teacher.";
6.     char b[] = "You are a student.";
7.     printf("string a=%s\nstring b=%s\n", a, b);
8.     printf("copy string a to string b:\n");
9.     copy_string(a, b);      //用字符数组名作为函数实参
10.    printf("\nstring a=%s\nstring b=%s\n", a, b);
11.    return 0;
12.}

13. void copy_string(char from[], char to[]) //形参为字符数组
14. {
15.     int i=0;
16.     while(from[i]!='\0') { to[i] = from[i]; i++;}
17.     to[i]='\0';
18. }
```

a,p from	a I a m a t e a c h e r . \0	a to Y o u a r e a s t u d e n t . \0	b I a m a t e a c h e r . \0
-------------	---	--	---

12.4.4 字符指针作为函数参数

- 例12-23：用函数调用实现字符串的复制。
- (2) 用数组名作为函数形参，字符型指针变量作为函数实参

```
1. #include <stdio.h>
2. void copy_string(char from[], char to[]);

3. int main()
4. {
5.     char a[] = "I am a teacher.";
6.     char b[] = "You are a student.";
7.     char *from = a, *to = b; //from,to分别指向数组a,b首元素
8.     printf("string a=%s\nstring b=%s\n", a, b);
9.     printf("copy string a to string b:\n");
10.    copy_string(from, to);      //实参为字符指针变量
11.    printf("\nstring a=%s\nstring b=%s\n", a, b);
12.    return 0;
13.}

14. void copy_string(char from[], char to[]) //形参为字符数组
15. {
16.     int i=0;
17.     while(from[i]!='\0') { to[i] = from[i]; i++;}
18.     to[i]='\0';
19. }
```



指针变量from的值是a数组首元素的地址，指针变量to的值是b数组首元素的地址。它们作为实参，把a数组首元素的地址和b数组首元素的地址传递给形参数组名from和to(它们实质上也是指针变量)。其他与(1)相同。

12.4.4 字符指针作为函数参数

■ 例12-23：用函数调用实现字符串的复制。(3) 用字符指针变量作形参和实参

```
1. #include <stdio.h>
2. void copy_string(char from[], char to[]);

3. int main()
4. {
5.     char a[] = "I am a teacher.";
6.     char b[] = "You are a student.";
7.     char *from = a, *to = b; //from,to分别指向数组a,b首元素
8.     printf("string a=%s\nstring b=%s\n", a, b);
9.     printf("copy string a to string b:\n");
10.    copy_string(from, to);    //实参为字符指针变量
11.    printf("\nstring a=%s\nstring b=%s\n", a, b);
12.    return 0;
13.}

14. void copy_string(char *from, char *to) //形参为字符指针变量
15. {
16.     for(; *from != '\0'; from++, to++) *to = *from;
17.     *to = '\0';
18. }
```

变种

```
void copy_string(char *from, char *to)
{
    for(;; (*to++=*from++) != '\0');
    //或for(;; *to++=*from++);
}
```

```
void copy_string(char *from, char *to)
{
    while((*to=*from) != '\0')
        //或while(*to=*from)
        {
            to++; from++;
        }
}
```

```
void copy_string(char *from, char *to)
{
    while(*from != '\0')
        //或while(*from)
        {
            *to++=*from++;
            *to = '\0';
        }
}
```

```
void copy_string(char *from, char *to)
{
    while((*to++=*from++) != '\0');
    //或while(*to++=*from++)
}
```

```
void copy_string(char from[], char to[])
{
    char *p1, *p2;
    p1=from; p2=to;
    while((*p2++=*p1++) != '\0');
}
```

12.4.5 返回指针的函数

类型名 *函数名(参数表列)

一个函数可以返回一个整型值、字符值、实型值等，也可以返回指针型的数据，即地址。其概念与以前类似，只是返回的值的类型是指针类型而已。

```
int *p(int x,int y);
```

p是函数名，调用它以后能得到一个int*型(指向整型数据)的指针，即整型数据的地址。x和y是函数p的形参，为整型。

注意

在“*p”两侧没有括号，在p的两侧分别为*运算符和()运算符。而()优先级高于*，因此p先与()结合，显然这是函数形式。这个函数前面有一个*，表示此函数是指针型函数（函数值是指针）。最前面的int表示返回的指针指向整型变量。返回指针的函数（指针函数）的本质是一个函数，其返回值为指针。

12.4.5 返回指针的函数

■ 例12-24：班上有a个学生，每个学生有b门课程的成绩。要求在输入学生序号后，输出该学生全部成绩。用指针函数来实现。

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

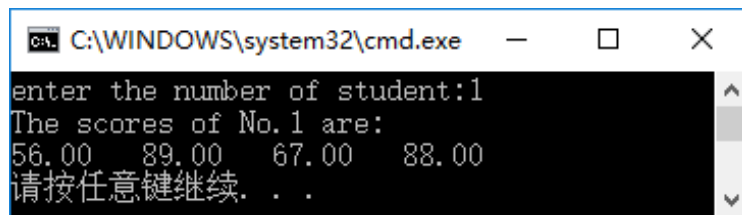
2. float *search(float (*pointer)[4],int n); //函数声明

3. int main()
4. {
5.     float score[][4]={
6.         {60,70,80,90},{56,89,67,88},{34,78,90,66}
7.     }; //定义数组，存放成绩

8.     float *p;
9.     int i,k;

10.    printf("enter the number of student:");
11.    scanf("%d",&k); //输入要找的学生的序号
12.    printf("The scores of No.%d are:\n",k);
13.    //调用search函数，返回score[k][0]的地址
14.    p=search(score,k);
15.    for(i=0;i<4;i++) printf("%5.2f\t",*(p+i)); //输出
16.    printf("\n");
17.    return 0;
18.}
```

```
19.float *search(float (*pointer)[4],int k)
20.//形参pointer是指向一维数组的指针变量
21.{
22.    float *pt;
23.    pt=(pointer+k); //pt的值是&score[k][0]
24.    return(pt);
25.}
```



```
C:\WINDOWS\system32\cmd.exe
enter the number of student:1
The scores of No.1 are:
56.00 89.00 67.00 88.00
请按任意键继续. . .
```

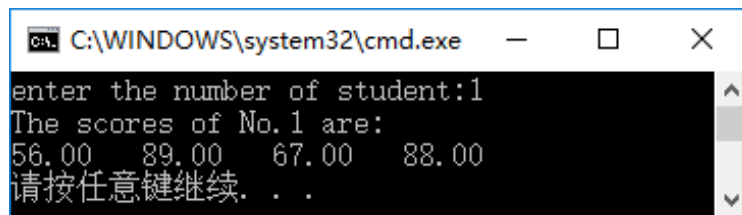
pointer	score数组			
pointer	60	70	80	90
pointer+1	56	89	67	88
	34	78	90	66

12.4.5 返回指针的函数

■ 例12-25：班上有a个学生，每个学生有b门课程的成绩。要求找出其中有不及格的课程的学生及其学生号。用指针函数来实现。

```
1. float *search(float (*pointer)[4]); //函数声明
2. int main() {
3.     float score[][4]={
4.         {60,70,80,90},{56,89,67,88},{34,78,90,66}
5.     }; //定义数组，存放成绩
6.     float *p;
7.     int i,j;
8.     for(i=0;i<3;i++) { //循环3次
9.         //调用search函数,如有不及格返回score[i][0]的地址,否则返回NULL
10.        p=search(score+i);
11.        if(p==*(score+i)) {
12.            //返回的是score[i][0]的地址，表示p的值不是NULL
13.            printf("No.%d score:",i);
14.            for(j=0;j<4;j++) printf("%5.2f  ",*(p+j));
15.            printf("\n");
16.        }
17.    }
18.    return 0;
19.}
```

```
20.float *search(float (*pointer)[4])
21.//定义函数，形参pointer是指向一维数组的指针变量
22.{
23.    int i=0;
24.    float *pt;
25.    pt=NULL; //先使pt的值为NULL
26.    for(;i<4;i++) if(*(pointer+i)<60) pt=pointer;
27.    //如果有不及格课程，使pt指向score[i][0]
28.    return(pt);
29.}
```



```
C:\WINDOWS\system32\cmd.exe
enter the number of student:1
The scores of No.1 are:
56.00  89.00  67.00  88.00
请按任意键继续. . .
```

pointer	score数组			
pointer+1	60	70	80	90
	56	89	67	88
	34	78	90	66

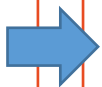
12.4.5 返回指针的函数

■ 例12-26：请指出下面代码的问题

```
1. #include <stdio.h>
2. #define N 100

3. //读进n个整数，并返回指向第k个整数的指针
4. int *f(int n, int k) {
5.     int a[N];
6.     int i;
7.     for (i = 0; i < n; i++) scanf("%d", &a[i]);
8.     return &a[k];
9. }

10. int main()
11. {
12.     int *p = f(10, 5);
13.     printf("%d\n", *p);
14. }
```



```
1. #include <stdio.h>
2. #define N 100

3. int a[N];

4. //读进n个整数，并返回指向第k个整数的指针
5. int *f(int n, int k) {
6.     int i;
7.     for (i = 0; i < n; i++) scanf("%d", &a[i]);
8.     return &a[k];
9. }

10. int main()
11. {
12.     int *p = f(10, 5);
13.     printf("%d\n", *p);
14. }
```

问题：数组a是函数f的局部变量，在f被调用时分配空间，在f返回时回收分配的空间，指向被回收区域的指针是无效指针。访问无效指针将引发不可预期的行为。

12.4.6 指向函数的指针

如果在程序中定义了一个函数，在编译时会把函数的源代码转换为可执行代码并分配一段存储空间。这段内存空间有一个起始地址，也称为函数的入口地址。每次调用函数时都从该地址入口开始执行此段函数代码。

函数名就是函数的指针，它代表函数的起始地址。

可以定义一个指向函数的指针变量，用来存放某一函数的起始地址，这意味着此指针变量指向该函数。例如：

```
int (*p)(int,int);
```

定义p是一个指向函数的指针变量，它可以指向函数类型为整型且有两个整型参数的函数。此时，指针变量p的类型用int (*)(int,int)表示。

注意

在“*p”两侧有括号，表明是一个指针。后面的(int, int)表明这是一个接收两个int型参数的函数，最前面的int表明返回值是int。**指向函数的指针（函数指针）的本质是一个指针，其指向一个函数。**

12.4.6 指向函数的指针

■ 定义和使用指向函数的指针

类型名 (*指针变量名)(函数参数表列)

```
int (*p)(int,int);
```

- (1) 定义指向函数的指针变量，并不意味着这个指针变量可以指向任何函数，它只能指向在定义时指定的类型的函数。【类型兼容】
- (2) 如果要用指针调用函数，必须先使指针变量指向该函数。【使用前要赋值】
- (3) 在给函数指针变量赋值时，只须给出函数名而不必给出参数。
- (4) 用函数指针变量调用函数时，只须将(*p)代替函数名即可（p为指针变量名），在(*p)之后的括号中根据需要写上实参。
- (5) 对指向函数的指针变量不能进行算术运算，如p+n,p++,p--等运算是无意义的。
- (6) 用函数名调用函数，只能调用所指定的一个函数，而通过指针变量调用函数比较灵活，可以根据不同情况先后调用不同的函数。

12.4.6 指向函数的指针

■ 例12-27：输入两个整数，然后让用户选择1或2。选1时调用max函数输出二者中的大数，选2时调用min函数输出二者中的小数。

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

2. int max(int,int);           //函数声明
3. int min(int x,int y);      //函数声明

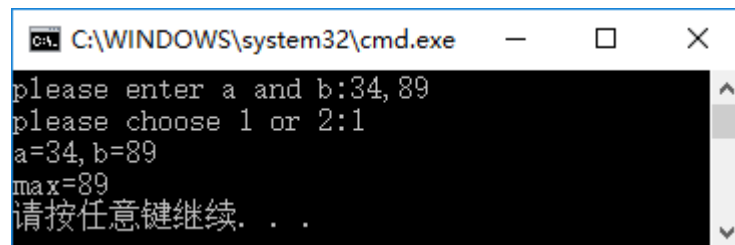
4. int main() {
5.     int (*p)(int,int);      //定义指向函数的指针变量
6.     int a,b,c,n;
7.     printf("please enter a and b:");
8.     scanf("%d,%d",&a,&b);
9.     printf("please choose 1 or 2:");
10.    scanf("%d",&n); //输入1或2

11.    if(n==1) p=max; //如输入1, 使p指向max函数
12.    else if (n==2) p=min; //如输入2, 使p指向min函数

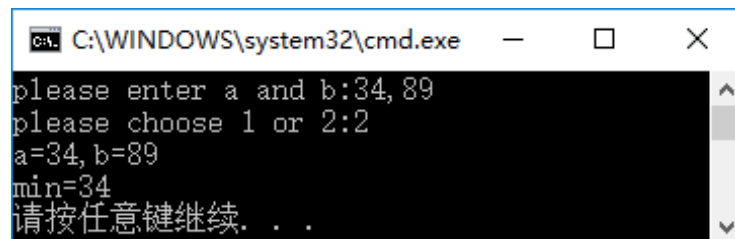
13.    c=(*p)(a,b);           //调用p指向的函数
14.    printf("a=%d,b=%d\n",a,b);
15.    printf("%s=%d\n", (n==1)?"max":"min", c);
16.    return 0;
17.}
```

```
18.int max(int x,int y) {
19.    if(x>y) return x;
20.    else return y;
21.}
```

```
22.int min(int x,int y) {
23.    if(<y) return x;
24.    else return y;
25.}
```



```
C:\WINDOWS\system32\cmd.exe
please enter a and b:34,89
please choose 1 or 2:1
a=34,b=89
max=89
请按任意键继续. . .
```



```
C:\WINDOWS\system32\cmd.exe
please enter a and b:34,89
please choose 1 or 2:2
a=34,b=89
min=34
请按任意键继续. . .
```

12.4.6 指向函数的指针

■ 用指向函数的指针作函数参数

指向函数的指针变量的一个重要用途是把函数的入口地址作为参数传递到其他函数。

指向函数的指针可以作为函数参数，把函数的入口地址传递给形参，这样就能够能够在被调用的函数中使用实参函数。

它的原理可以简述如下：有一个函数（假设函数名为fun），它有两个形参（x1和x2），定义x1和x2为指向函数的指针变量。在调用函数fun时，实参为两个函数名f1和f2，给形参传递的是函数f1和f2的入口地址。这样在函数fun中就可以调用f1和f2函数了。函数指针参数又被称为回调函数（callback function）。

实参函数名 f1

f2

```
void fun(int (*x1)(int), int(*x2) (int,int))
```

```
{    int a,b,i=3,j=5;
```

```
    a=(*x1)(i);
```

```
    b=(*x2)(i,j);
```

```
}
```

//定义fun函数，形参是指向函数的指针变量

//调用f1函数，i是实参

//调用f2函数，i,j是实参

12.4.6 指向函数的指针

■ 例12-28：有两个整数a和b，由用户输入1,2或3。如输入1，程序就给出a和b中的大者；输入2，就给出a和b中的小者；输入3，则求a与b之和。

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

2. //fun函数声明
3. int fun(int x,int y, int (*p)(int,int));
4. int max(int,int);           //max函数声明
5. int min(int,int);           //min函数声明
6. int add(int,int);           //add函数声明

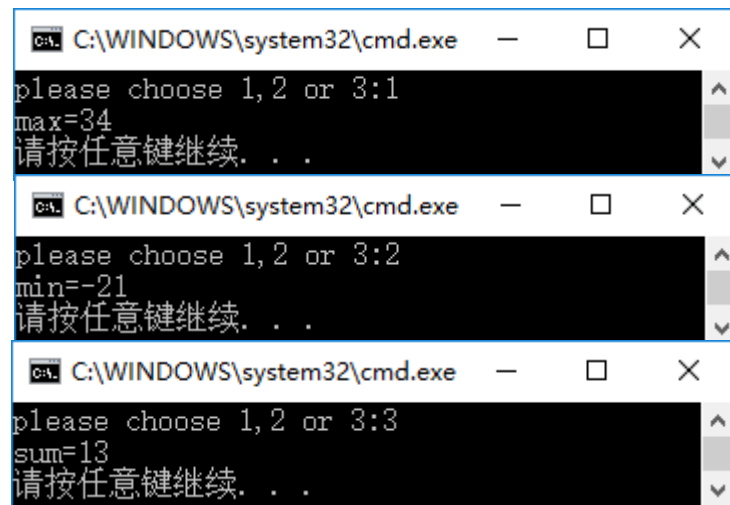
7. int main()
8. {
9.     int a=34,b=-21,n;
10.    printf("please choose 1,2 or 3:");
11.    scanf("%d",&n);           //输入1,2或3之一
12.    if(n==1) fun(a,b,max);     //输入1时调用max函数
13.    else if(n==2) fun(a,b,min); //输入2时调用min函数
14.    else if(n==3) fun(a,b,add); //输入3时调用add函数
15.    return 0;
16.}

17.int fun(int x,int y,int (*p)(int,int))
18.{
19.    int result;
20.    result=(*p)(x,y);
21.    printf("%d\n",result);      //输出结果
22.}
```

```
23.int max(int x,int y) {
24.    if(x>y) return x; else return y;
25.}

26.int min(int x,int y) {
27.    if(<y) return x; else return y;
28.}

29.int add(int x,int y) { return x+y; }
```



The image shows three sequential screenshots of a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". Each screenshot shows the program's output for a specific user input. In the first, input is 1, output is "max=34". In the second, input is 2, output is "min=-21". In the third, input is 3, output is "sum=13". Each output is followed by the prompt "请按任意键继续. . .".

```
C:\WINDOWS\system32\cmd.exe
please choose 1,2 or 3:1
max=34
请按任意键继续. . .

C:\WINDOWS\system32\cmd.exe
please choose 1,2 or 3:2
min=-21
请按任意键继续. . .

C:\WINDOWS\system32\cmd.exe
please choose 1,2 or 3:3
sum=13
请按任意键继续. . .
```

12.4.6 指向函数的指针

■ 巧用库提供的排序函数

C 库函数 - `qsort()`

 C 标准库 - <stdlib.h>

描述

C 库函数 `void qsort(void *base, size_t nitems, size_t size, int (*compar)(const void *, const void*))` 对数组进行排序。

声明

下面是 `qsort()` 函数的声明。

```
void qsort(void *base, size_t nitems, size_t size, int (*compar)(const void *, const void*))
```

参数

- **base** -- 指向要排序的数组的第一个元素的指针。
- **nitems** -- 由 **base** 指向的数组中元素的个数。
- **size** -- 数组中每个元素的大小，以字节为单位。
- **compar** -- 用来比较两个元素的函数。

返回值

该函数不返回任何值。

<https://www.runoob.com/cprogramming/c-function-qsort.html>

12.4.6 指向函数的指针

■ 巧用库提供的排序函数

compar参数：指向一个比较两个元素的函数。比较函数的原型如下：

```
int compar(const void *p1, const void *p2);
```

注意：两个形参必须是`const void *`类型，同时在调用`compar` 函数时，传入的实参也必须转换成`const void *`类型。在`compar`函数的内部会将`const void *`类型强制转换成实际类型。

- 如果`compar`返回值小于0 (< 0)，那么`p1`所指向元素会被排在`p2`所指向元素的前面；
- 如果`compar`返回值等于0 ($= 0$)，那么`p1`所指向元素与`p2`所指向元素的顺序不确定；
- 如果`compar`返回值大于0 (> 0)，那么`p1`所指向元素会被排在`p2`所指向元素的后面

12.4.6 指向函数的指针

■ 例12-29：身份证排序 (YOJ-135)

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>

4. #define N 100000
5. #define LEN 19

6. char ids[N][LEN];

7. int compare(const void *a, const void *b)
8. {
9.     char *astr = (char *)a;
10.    char *bstr = (char *)b;
11.    int cmp_res;

12.    //先比较生日
13.    cmp_res = strncmp(astr + 6, bstr + 6, 8);
14.    //再比较身份证号
15.    if (cmp_res==0) cmp_res = strcmp(astr, bstr);
16.    //要求从大到小排序, 故大的要放左边, 返回负数
17.    return -cmp_res;
18.}
```

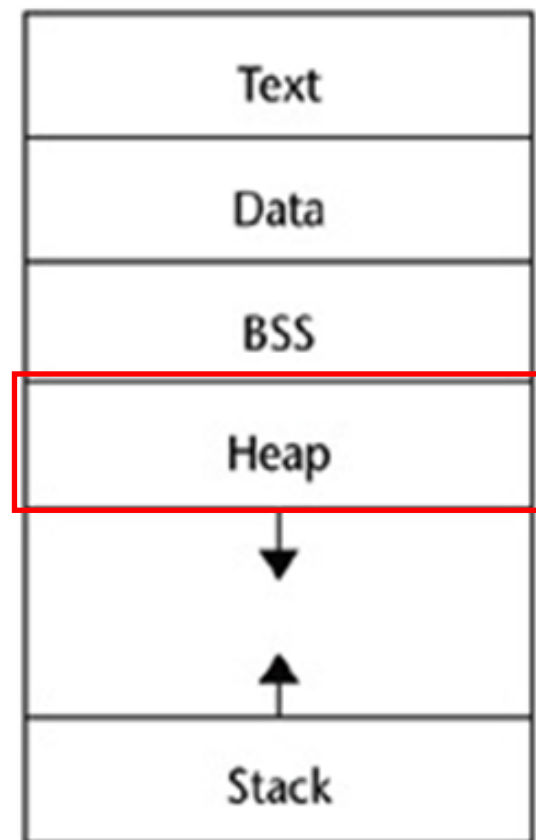
```
19. void main()
20. {
21.     int n, i;
22.
23.     scanf("%d", &n);
24.     for (i = 0; i < n; i++)
25.         scanf("%s", ids[i]);

26.     qsort(ids, n, LEN, compare);

27.     for (i = 0; i < n; i++)
28.         printf("%s\n", ids[i]);
29. }
```

12.5 动态内存分配与释放

全局变量是分配在内存中的静态存储区的，非静态的局部变量(包括形参)是分配在内存中的动态存储区的，这个存储区是一个称为**栈**(stack)的区域。除此以外，C语言还允许建立内存动态分配区域，以存放一些临时用的数据，这些数据不必在程序的声明部分定义，也不必等到函数结束时才释放，而是需要时随时开辟，不需要时随时释放。这些数据是临时存放在一个特别的自由存储区，称为**堆**(heap)区。可以根据需要，向系统申请所需大小的空间。由于未在声明部分定义它们为变量或数组，因此不能通过变量名或数组名去引用这些数据，只能通过指针来引用。



注意：使用到动态内存分配与释放的程序需要引用<stdlib.h>头文件。

12.5.1 动态分配内存

用malloc函数开辟动态存储区

函数原型为

```
void *malloc(unsigned int size);
```

作用是在内存的动态存储区中分配一个长度为size的连续空间。形参size的类型定为无符号整型(不允许为负数)。此函数的值（即“返回值”）是所分配区域的第0个字节的地址，或者说，此函数是一个指针型函数，返回的指针指向该分配域的第0个字节。

```
malloc(100);    //开辟100字节的临时分配域，函数值为其第1个字节的地址
```

指针的基类型为void，即不指向任何类型的数据，只提供一个纯地址。如果此函数未能成功地执行(例如内存空间不足)，则返回空指针(NULL)。

12.5.1 动态分配内存

用calloc函数开辟动态存储区

函数原型为

```
void *calloc(unsigned n, unsigned size);
```

作用是在内存的动态存储区中分配n个长度为size的连续空间，这个空间一般比较大，足以**保存一个数组**。

用calloc函数可以为**一维数组**开辟动态存储空间，n为数组元素个数，每个元素长度为size。这就是动态数组。函数返回指向所分配域的第0个字节的指针；如果分配不成功，返回NULL。

```
p=calloc(50,4);           //开辟50×4个字节的临时分配域，把首地址赋给指针变量p
```

12.5.1 动态分配内存

用realloc函数重新分配动态存储区

函数原型为 `void *realloc(void *p,unsigned int size);`

如果已经通过malloc函数或calloc函数获得了动态空间，想改变其大小，可以用realloc函数重新分配。

用realloc函数将p所指向的动态空间的大小改变为size。p的值不变。如果重分配不成功，返回NULL。

```
realloc(p,50);    //将p所指向的已分配的动态空间改为50字节
```

注意：

- 无论是扩大或缩小，原有内存中的内容将保持不变。对于缩小，则被缩小的那一部分的内容会丢失，其余内容保持不变。
- realloc并不保证调整后的内存空间和原来的内存空间保持同一内存地址。相反，realloc返回的指针很可能指向一个新的地址

12.5.1 动态分配内存

■ malloc v.s. calloc v.s. realloc

- malloc不会初始化所分配的内存空间。由malloc()函数分配的内存空间可能遗留有各种各样的数据。
- calloc会将所分配的内存空间中的每一位都初始化为零。
- realloc可以对给定的指针所指的空間进行扩大或者缩小。realloc函数所分配的空间也是未初始化的

```
1. #include <stdio.h>
2. #include <memory.h>

3. void main() {
4.     int *x = malloc(4 * 3);
5.     int *y = calloc(3, 4);
6.     int i;

7.     printf("x: ");
8.     for (i = 0; i < 3; i++) printf("%d ", *(x + i));
9.     printf("\n");

10.    printf("y: ");
11.    for (i = 0; i < 3; i++) printf("%d ", *(y + i));
12.    printf("\n");
```

```
13.    for (i = 0; i < 3; i++) *(y + i) = i;

14.    y = realloc(y, 4 * 5);
15.    printf("new y: ");
16.    for (i = 0; i < 5; i++) printf("%d ", *(y + i));
17.    printf("\n");
18.}
```

```
x: 13723024 0 13697360
y: 0 0 0
new y: 0 1 2 0 0
```

12.5.1 动态分配内存

用free函数释放动态存储区

函数原型为

```
void free(void *p);
```

作用是释放指针变量p所指向的动态空间，使这部分空间能重新被其他变量使用。

p应是最近一次调用calloc或malloc函数时得到的函数返回值。

```
free(p);    //释放指针变量p所指向的已分配的动态空间
```

free函数无返回值。

12.5.1 动态分配内存

■ 例12-30：建立动态数组，输入5个学生的成绩，另外用一个函数检查其中有无低于60分的，输出不合格的成绩。程序结束前，释放动态分配的数组内存空间

```
1. #include <stdio.h>
2. #include <stdlib.h>                                //程序中用了malloc函数，应包含stdlib.h

3. void check(int *);                                  //函数声明

4. int main()
5. {
6.     int *p,i;                                        //p1是int型指针
7.     p = (int *)malloc(5*sizeof(int));               //开辟动态内存区，将地址转换成int *型，然后放在p1中
8.     for(i=0;i<5;i++) scanf("%d", p+i);             //输入5个学生的成绩
9.     check(p);                                         //调用check函数
10.    free(p);
11.    return 0;
12.}

13.void check(int *p)                                  //定义check函数，形参是int*指针
14.{
15.    int i;
16.    printf("They are fail:");
17.    for(i=0;i<5;i++) if(p[i]<60) printf("%d ",p[i]); //输出不合格的成绩
18.    printf("\n");
19.}
```

优先级	运算符	名称或含义	使用形式	结合方向
1	[]	数组下标	数组名[常量表达式]	左到右
2	&	取地址运算符	&变量名	右到左

12.5.1 动态分配内存

■ 例12-31：动态建立学生结构体数组，输入n个学生的信息，另外用一个函数来检查学生性别，输出男生的信息。程序结束前，释放动态分配的数组内存空间

```

1. #include <stdio.h>
2. #include <stdlib.h>

3. typedef struct
4. {
5.     int sno;           //学号为整型
6.     char name[20];     //姓名为字符串
7.     char sex;          //性别为字符型 (F: 女生, M: 男生)
8.     short age;         //年龄为短整型
9. } Student;

10. void check(Student *stu) {
11.     if (stu->sex == 'M')
12.         printf("sno: %d name: %s sex: %c age: %d\n",
13.             stu->sno, stu->name, stu->sex, stu->age);
14. }

15. int main()
16. {
17.     int i, n;
18.     Student *stu, **p;

19.     scanf("%d\n", &n);
20.     stu = (Student *)calloc(n, sizeof(Student));
21.     p = &stu;

22.     for (i = 0; i < n; i++)
23.         scanf("%d %s %c %d",
24.             &((*p)[i].sno), (*p)[i].name,
25.             &((*p)[i].sex), &((*p)[i].age));

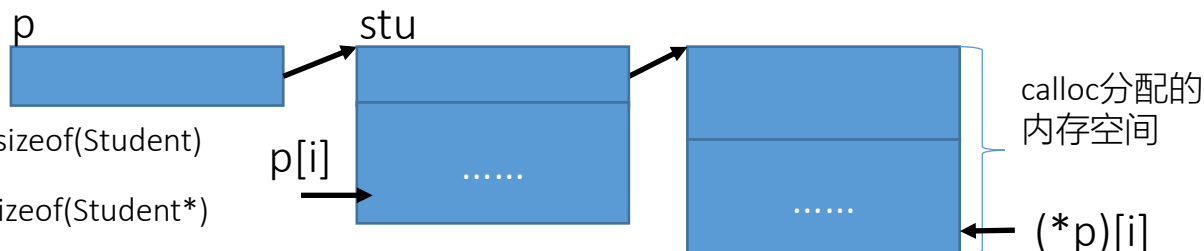
26.     for (i = 0; i < n; i++)
27.         check(&(*p)[i]);
28.
29.     free(stu);
30. }

```

获得表示第i个学生的结构体地址

正确写法: $\&(*p)[i]$ 注: $\text{addr}((\&p)[i]) \equiv \text{stu} + i * \text{sizeof}(\text{Student})$

错误写法: $p[i]$ 注: $\text{addr}(p[i]) \equiv \text{addr}(\text{stu}) + i * \text{sizeof}(\text{Student} *)$



有关指针的小结

(1) 首先要准确理解指针的含义。“指针”是C语言中一个形象化的名词，形象地表示“指向”的关系，其在物理上的实现是通过地址来完成的。

- `&a`是变量`a`的地址，也可称为变量`a`的指针。
- 指针变量是存放地址的变量，也可以说，指针变量是存放指针的变量。
- 指针变量的值是一个地址，也可以说，指针变量的值是一个指针。
- 指针变量也可称为地址变量，它的值是地址。
- `&`是取地址运算符，`&a`是`a`的地址，也可以说，`&`是取指针运算符。`&a`是变量`a`的指针（即指向变量`a`的指针）。
- 数组名是一个地址，是数组首元素的地址，也可以说，数组名是一个指针，是数组首元素的指针。
- 函数名是一个指针(指向函数代码区的首字节)，也可以说函数名是一个地址(函数代码区首字节的地址)。
- 函数的实参如果是数组名，传递给形参的是一个地址，也可以说，传递给形参的是一个指针。

有关指针的小结

(2) 一个地址型的数据实际上包含3个信息：

- ① 表示内存编号的纯地址。
- ② 它本身的类型，即指针类型。
- ③ 以它为标识的存储单元中存放的是什么类型的数据，即基类型。

```
int a;
```

```
/* &a为a的地址，它就包括以上3个信息，它代表的是一个整型数据的地址，int是&a的基类型(即它指向的是int型的存储单元)。&a就是“指向整型数据的指针类型”或“基类型为整型的指针类型”，其类型可以表示为“int*”型。*/
```

有关指针的小结

(3) 要区别指针和指针变量。指针就是地址，而指针变量是用来存放地址的变量。

(4) 什么叫“指向”？地址就意味着指向，因为通过地址能找到具有该地址的对象。对于指针变量来说，把谁的地址存放在指针变量中，就说此指针变量指向谁。

```
int a,*p;    //p是int*型的指针变量，基类型是int型
float b;
p=&a;        //a是int型，合法
p=&b;        //b是float型，类型不匹配
```

void *指针是一种特殊的指针，不指向任何类型的数据。如果需要用此地址指向某类型的数据，应先对地址进行类型转换。

注意

并不是任何类型数据的地址都可以存放在同一个指针变量中的，只有与指针变量的基类型相同的数据的地址才能存放在相应的指针变量中。

有关指针的小结

(5) 要深入掌握在对数组的操作中正确地使用指针，搞清楚指针的指向。

```
int *p, a[10];    //p是指向int型类型的指针变量
p=a;             //p指向a数组的首元素
```

(6) 有关指针变量的归纳比较

变量定义	类型表示	含义
int i;	int	定义整型变量i
int *p;	int *	定义p为指向整型数据的指针变量
int a[5];	int [5]	定义整型数组a，它有5个元素
int *p[4];	int *[4]	定义指针数组p，它由4个指向整型数据的指针元素组成
int (*p)[4];	int (*)[4]	p为指向包含4个元素的一维数组的指针变量
int f();	int ()	f为返回整型函数值的函数
int *p();	int *()	p为返回一个指针的函数，该指针指向整型数据
int (*p)();	int (*)()	p为指向函数的指针，该函数返回一个整型值
int **p;	int **	p是一个指针变量，它指向一个指向整型数据的指针变量
void *p;	void *	p是一个指针变量，基类型为void(空类型)，不指向具体的对象

有关指针的小结

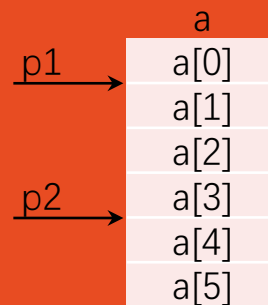
(7) 指针运算

- ① 指针变量加（减）一个整数。

```
p++; //将该指针变量的原值(是一个地址)和它指向的变量所占用的存储单元的字节数相加
```

- ② 指针变量赋值。将一个变量地址赋给一个指针变量。 不应把一个整数赋给指针变量。

```
p=&a; //将变量a的地址赋给p
p=array; //将数组array首元素地址赋给p
p=&array[i]; //将数组array第i个元素的地址赋给p
p=max; //max为已定义的函数，将max的入口地址赋给p
p1=p2; //p1和p2是基类型相同指针变量，将p2的值赋给p1
```



- ③ 两个指针变量可以相减。如果两个指针变量都指向同一个数组中的元素，则两个指针变量值之差是两个指针之间的元素个数。
- ④ 两个指针变量比较。若两个指针指向同一个数组的元素，则可以进行比较。指向前面的元素的指针变量“小于”指向后面元素的指针变量。如果`p1`和`p2`不指向同一数组则比较无意义。

有关指针的小结

(8) 指针变量可以有空值，即该指针变量不指向任何变量。

```
p=NULL;
```

NULL是一个符号常量，代表整数0。在stdio.h头文件中对NULL进行了定义：`#define NULL 0`。它使p指向地址为0的单元。系统保证使该单元不作它用（不存放有效数据）。

注意

p的值为NULL与未对p赋值是两个不同的概念。前者是有值的（值为0），不指向任何变量，后者虽未对p赋值但并不等于p无值，只是它的值是一个无法预料的值，也就是p可能指向一个事先未指定的单元。

任何指针变量或地址都可以与NULL作相等或不相等的比较。

```
if(p==NULL)
```

指针的优点：

- ① 提高程序效率；
- ② 在调用函数时当指针指向的变量的值改变时，这些值能够为主调函数使用，即可以从函数调用得到多个可改变的值；
- ③ 可以实现动态存储分配。

如果使用指针不当，会出现隐蔽的、难以发现和排除的故障。因此，使用指针要十分小心谨慎。