第七章 指针(二)

七、指针作为函数参数

八、返回指针变量

九、指向指针的指针

十、指针占用的字节数

十一、指针和数组

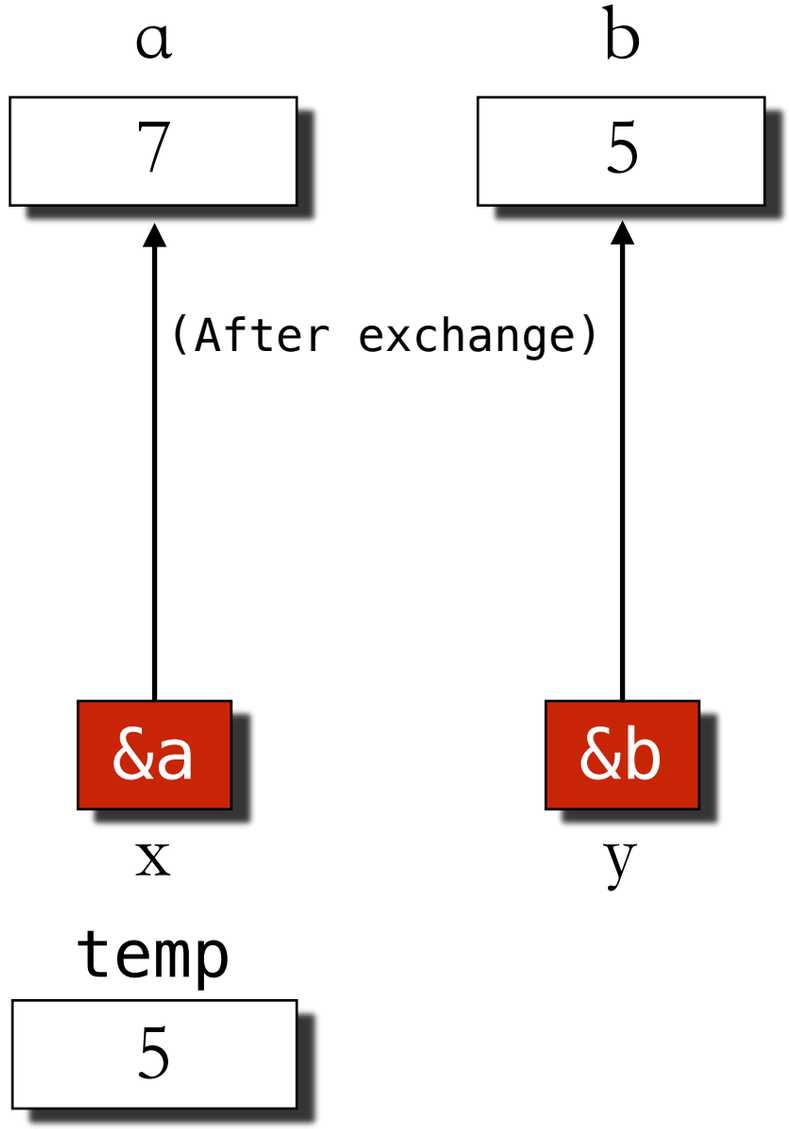
十二、指针作为参数详解

七、指针作为函数参数

```
/* prototype Declarations */
void exchange(int *,int *);

int main(void)
{
     int a = 5;
     int b = 7;
     exchange(&a,&b);
     printf("%d %d\n",a,b);
     return 0;
}/* main */
```

```
void exchange(int *x,int *y)
{
     int temp;
     tmp = *x;
     *x = *;
     *y = temp;
     return;
}/* exchange */
```



代码分析:

- exchange(&a ,&b); 把变量a和变量b的地址，传递给exchange函数。
- void exchange(int * x, int * y) x和y本身是两个整型指针变量。
- 调用该函数后，形参x的内容是变量a的地址，x = &a;
  形参y的内容是变量b的地址，y = &b;
- 如果在函数内修改*x和*y，就相当于修改变量a和变量b。
- temp = *x; 等价于temp = a;//temp存储原来的a的值。
  *x = *y ; 等价于a = b;
  *y = temp; b = 原来的a;

- 函数的作用：交换了*x和*y的值，就相当于交换了x和y两个指针指向的变量a和b的值。

Q：如果exchange的两个参数不是指针，
而是exchange(int x, int y)，那么调用之后，a和b的值是怎样的?

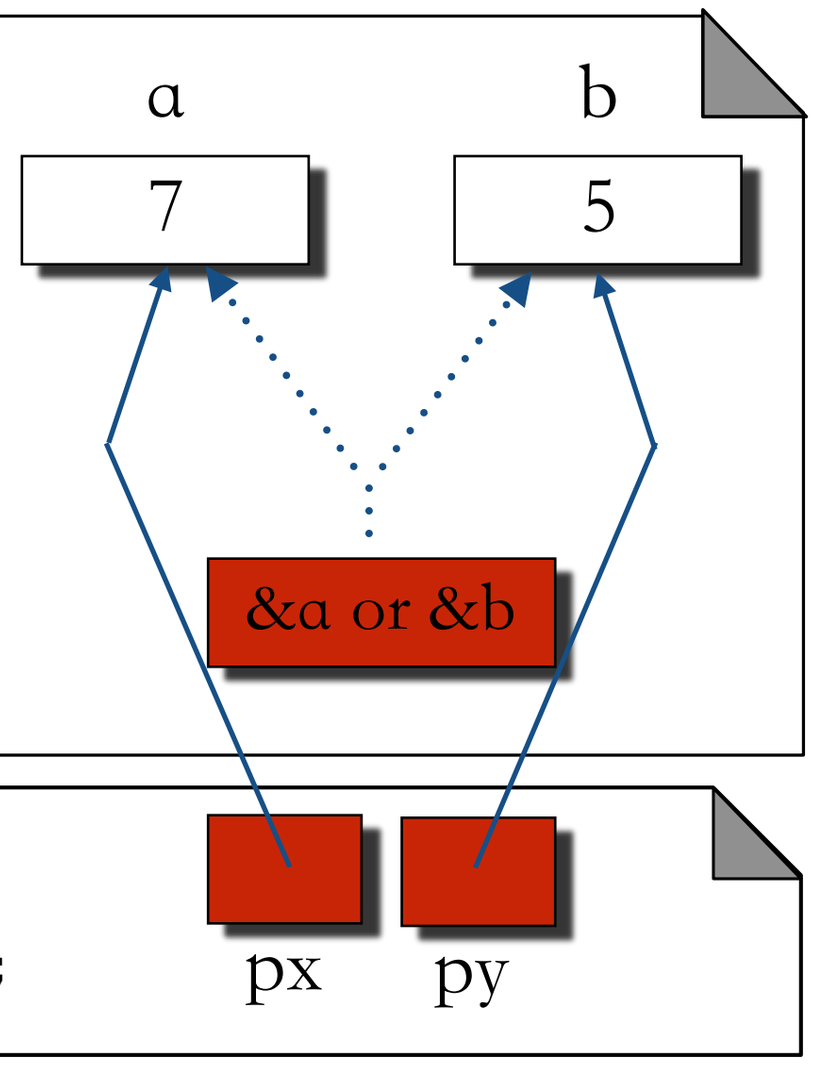
八、函数返回一个指针变量

可以将指针变量作为函数的返回值返回，如：

```
int * smaller (int * p1, int * p2);
```

```
/* prototype Declarations */
int *smaller(int *px,int *py);

int main(void)
{
     int a;
     int b;
     int *p;
     printf(“请输入两个整数:”);
     scanf(“%d%d”,&a,&b);
     p = smaller(&a,&b);
     printf(“较小值为:%d\n”,*p);
}
```

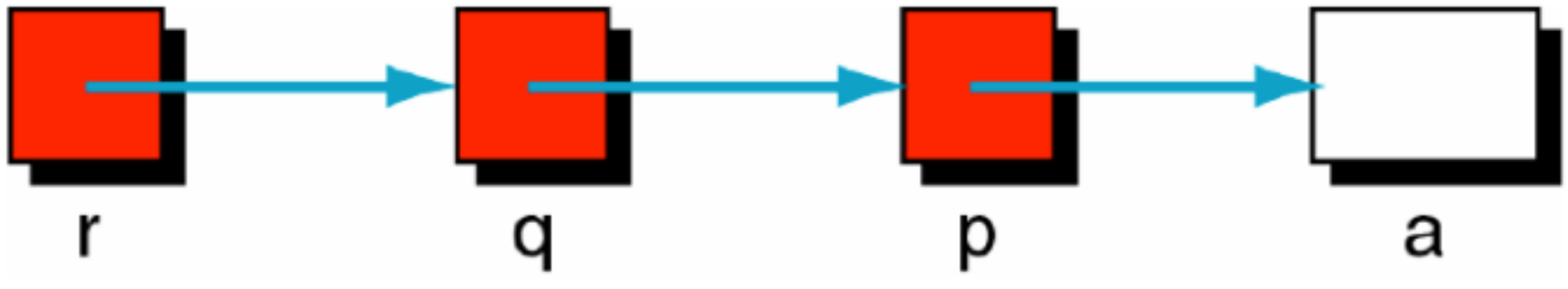



```
int *smaller(int *px,int *py)
{
     return (*px < *py ? px : py);
}
```

九、指向指针的指针 (Pointers to Pointers)

指向指针的指针也可称为二维指针或二级指针，指针本身也是一个变量，所以指针变量本身在内存中也是有空间地址，因此可以声明一个变量来保存这个指针变量的地址，如：

```
int a = 5;
int *p = &a;  //一级指针p指向变量a
int **q = &p; //二级指针q指向变量q
int ***r = &q;//三级指针r指向变量q
```



十、指针变量占用的内存大小

定义一个指针变量，编译器就会给这个指针变量分配一个空间，这个空间存放的是内存的某个地址数据的，即这个指针变量是用来存放内存某地址的，编译器给这个指针变量分配的空间大小是4个字节，32位寻址长度的CPU,指针变量所占的空间大小是4个字节，与所定义指针的类型无关，因为不管定义什么类型的指针变量，它都是用来存地址的。

```
For example:
   char c;          Print:
   char *pc;        sizeof(c)=1 sizeof(pc)=4 sizeof(*pc)=1
   int a;           sizeof(a)=4 sizeof(pa)=4 sizeof(*pa)=4
   int *pa;         sizeof(x)=8 sizeof(px)=4 sizeof(*px)=8
   double x;
   double *px;
```

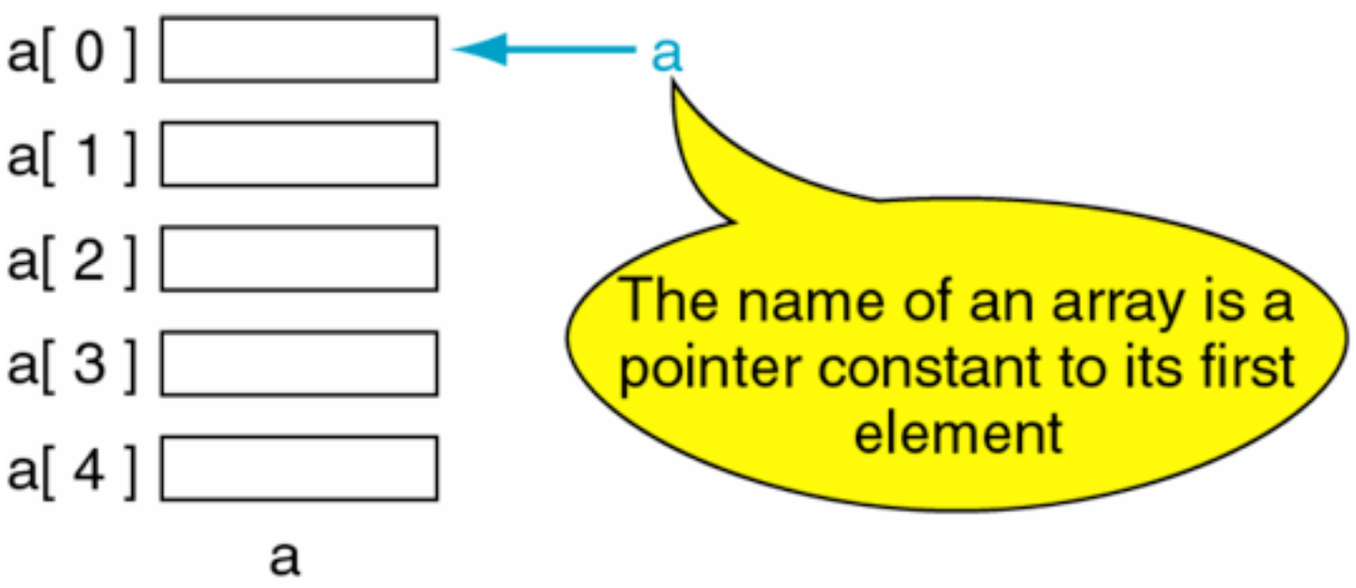
注：所有的指针变量都是占用4个字节（32位），不管是一级指针，二级指针还是多级指针。

十一、指针和数组

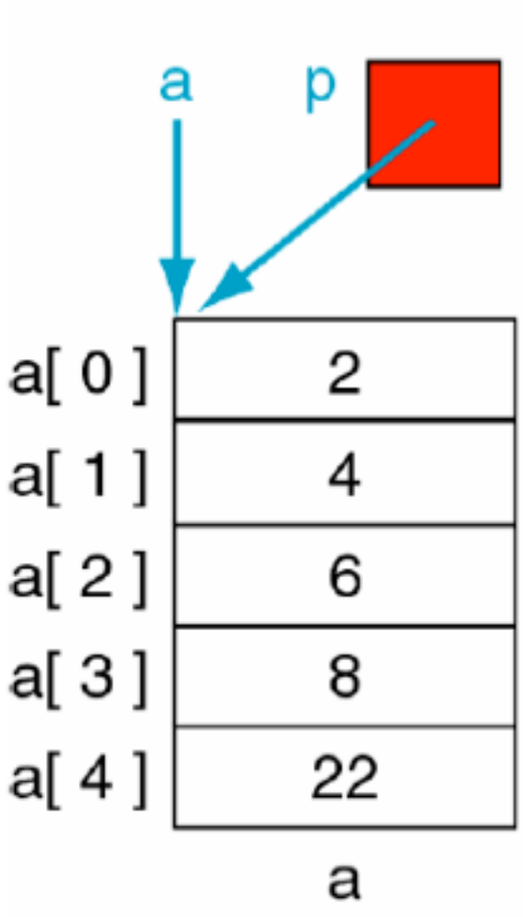
C语言的数组表示一段连续的内存空间，用来存储多个相同类型的值。如:

```
int a[5];
```

- 编译器会在内存中分配5 * sizeof(int) = 20个字节;
- 数组中的5个元素的在内存中是连续分布的;
- 数组名是数组首元素的地址，即数组名是指向数组首元素的“指针”;
- 数组名是指针常量，即数组名的值不能改变，只能指向数组首元素;



示例代码：




```
#include <stdio.h>
int main(void)
{
    int a[5] = {2,4,6,8,22};
    int *p = a;
    int i = 0;
    printf(“%d %d\n”,a[i],*p);
    return 0;
}
```

程序输出结果为：22

将变量名a赋值给指针变量p，a和p都指向a[0]，

*p就是a[0]

使用指针访问数组元素：

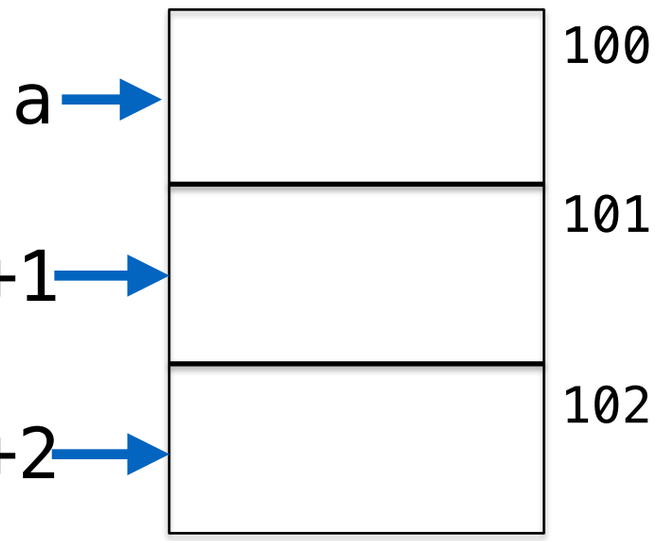
```
#include <stdio.h>
int main (void)
{
    int a[5] = {2, 4, 6, 8, 22};
    int *p;
    p = &(a[1]);
    printf("%d %d\n",a[0],*(p-1));
}
```

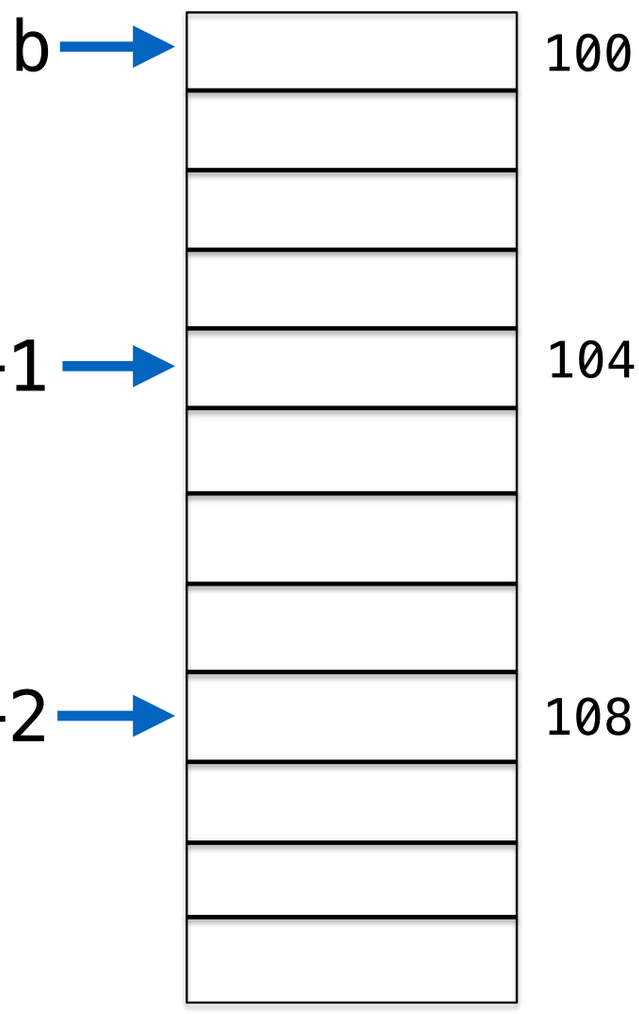
代码分析：

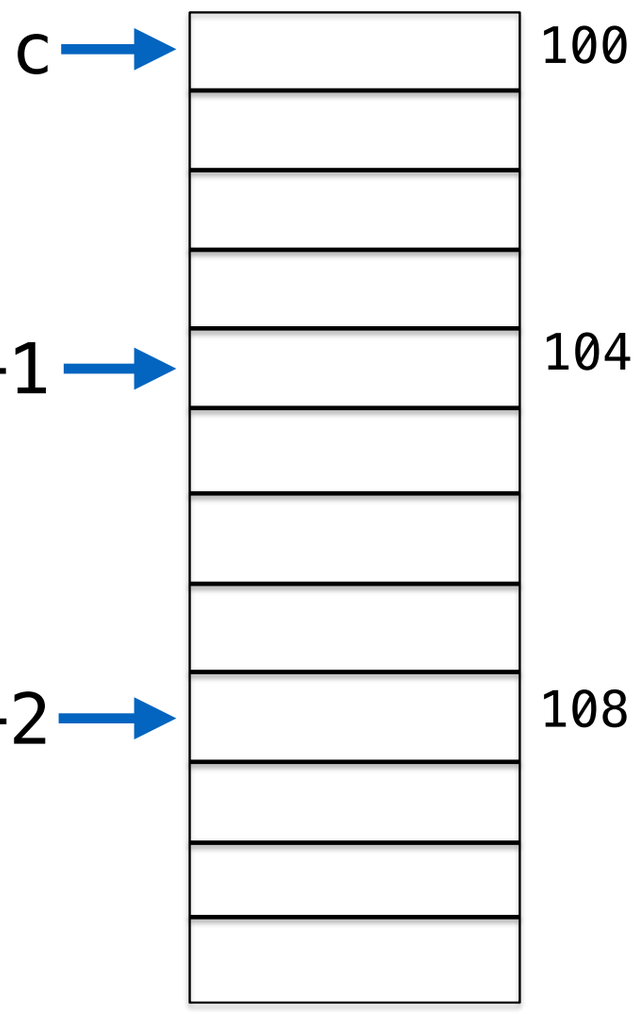
上述代码中指针变量p指向a[1]，p-1是向前移动一个元素占用的字节大小，而不是单纯的数字 -1 ，即指针p - 1是移动一个整型元素的大小4个字节。

Consider the following code:

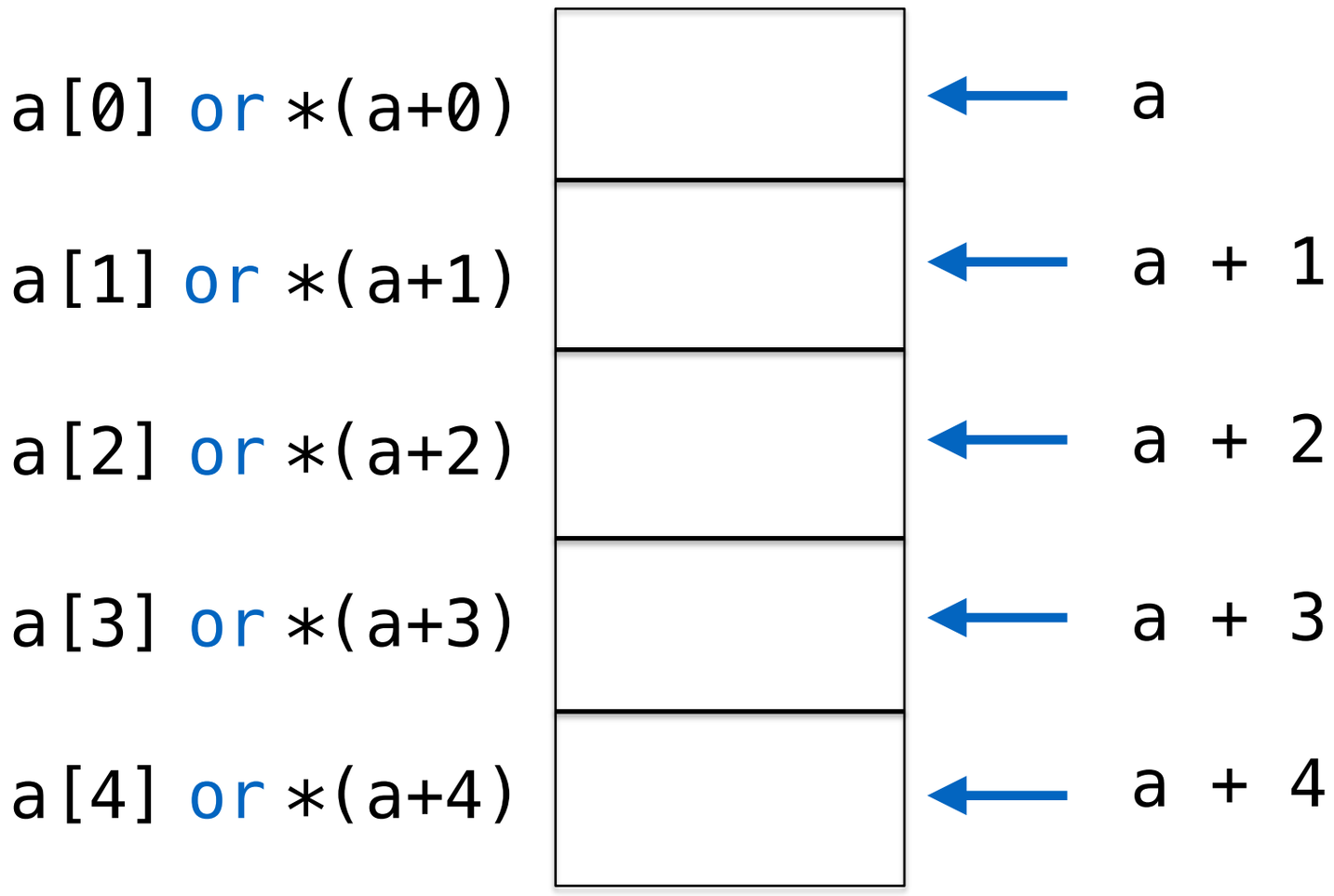
```
char a[3];
int b[3];
float c[3];
```







指针和数组图解：



上述图解可得到：a[i] 等价于 *(a + i) 或 &a[i] 等价于 a + i;

十二、指针作为参数详解

传值调用：

```c
#include <stdio.h>
void f(int x)
{
    x = x+1;
}
int main (void)
{
    int a=3;
    f(a);
    printf("%d",a);  /* prints 3 */
    return 0;
}
```

分析：传值调用是把实参的值的一份拷贝传递给形参，单向传递，形参的改变不会影响实参。

传指针调用：

```c
#include <stdio.h>
void f(int *x)
{
    *x = *x+1;
}
int main (void)
{
    int a=3;
    f(&a);
    printf("%d",a);  /* prints 4 */
    return 0;
}
```

分析：传指针调用是把实参的值（地址）赋值给形参，形参和实参都指针同一个变量，在函数体内通过形参解引用修改变量a的值。

示例代码一：

```c
#include <stdio.h>

void f(int *x)
{
    x = x+1;
}
int main (void)
{
    int a[]={1,2}, *pa=a;
    f(pa);
    printf("%d",*pa);
    return 0;
}
```

分析：示例一将实参pa的值传递给形参x，修改形参x时，pa的值并不受影响，因为x和pa是两个不同的指针变量。

示例代码二：

```c
#include <stdio.h>
void f(int *x)
{
    x += 1;//指针指向下一位
    *x =100; //修改指向内容将2换为100
}
int main (void)
{
    int a[]={1,2}, *pa=a;
    f(pa);
    printf("%d %d\n",*(pa+1), a[1]);
    return 0;
}
```

分析：示例二将实参pa的值传递给形参x，此时pa和x都指向a[0]，x+=1指针指向下一个元素,此时x指向a[1]，故*x =100即a[1] = 100。

The End

渥瑞达·教学部