程序设计语言与方法(C语言)

第九章 指针

问题

- 编写一个函数swap，将两个整型变量的值互换
  - int x, int y
  - x <—> y

问题

- 设计并实现一个可以交换两个整型变量的值的函数
  - 数据
    - 输入：两个整型数据x和y
    - 输出：交换值后的两个数据x和y
      - 输出的是两个变量，如何解决?
  - 过程
    - 交换x和y的值

程序

```
#include <stdio.h>

void swap(int x, int y);

int main() {
    int a, b;
    printf("please input:”);
    scanf("%d%d", &a, &b);
    swap(a, b);
    printf("a = %d; b = %d\r\n", a, b);
    return 0;
}

void swap(int x, int y) {
    int temp = x;
    x = y;
    y = temp;
}
```

```
#include <stdio.h>

void swap(int *x, int *y);

int main() {
    int a, b;
    printf("please input:”);
    scanf("%d%d", &a, &b);
    swap(&a, &b);
    printf("a = %d; b = %d\r\n", a, b);
    return 0;
}

void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
}
```

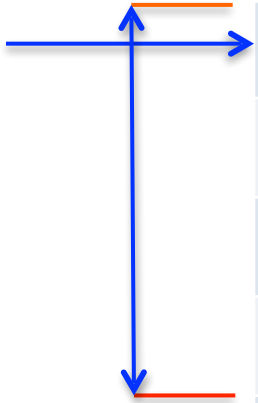
数据与数据的存储位置

- 数据是用二进制表示的数
  - 计算机如何知道数据放在什么地方
    - 存储器
    - 存储单元（字节）
    - 存储单元按序顺序排列
    - 存储单元在排列中的序号（存储单元地址）
- 数据的大小
    - 按字节计算，仅需知道每个数据所在存储区块的首地址
    - 如何知道一个变量或某个数据类型表示数据时的字节数
      - sizeof(类型名称 或 变量名称)

数据存储示意

存储区块的首地址
（&x）

存储容量
（sizeof(x)）



地址	数据	对应的变量名
0028ff10	04	int x = 4;
	00	
	00	
	00	
0028ff14	05	int y = 5;
	00	
	00	
	00	

存储单元地址的表示

- 指针（一个内存单元的地址）
- 通过指针获取数据
  - 指针的值
    - 解决了数据存储位置的问题
  - 指针的类型
    - 解决了如何从一个地址开始的几个字节的二进制数，构成一个合适数据的问题
  - 指针在使用前必须初始化
  - 一个确定的地址

指针访问

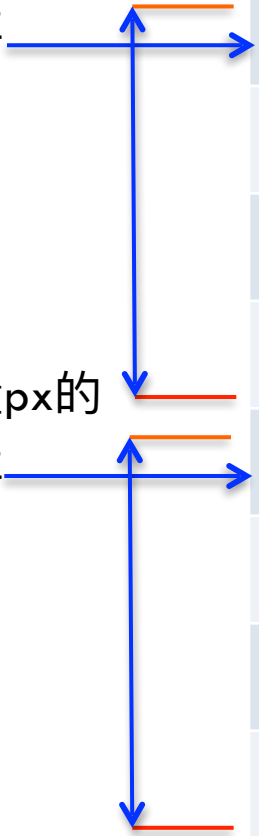
- 对指针本身的访问（取值和赋值）
  - int x = 5; /* 定义一个值为5的整形变量*/
  - int *pa = &x;
  - int *pa;
  - pa = &x;
  - /*定义一个整型指针变量，并将整型变量x的存储位置赋给它*/
- 对指针指向位置的访问（间接访问）
  - int y;
  - /* 获取指针变量px指向存储位置中的数据 */
  - y = *px; /* 间接访问：取值 */
  - /* 向指针指向位置赋值，如将20存储到px指向的存储位置中 */
  - *pa = 20; /* 间接访问：赋值 */

指针示意

x 的值 VS x 的存储位置
px的值 VS px的存储位置
px的值 VS px指向的值

存储整型变量x的
内存块首地址
int x = 5;

存储指针变量px的
内存块首地址
int *px = &x;



地址	数据	对应的变量名
0028ff14	05	int x = 5;
	00	
	00	
	00	
0028ff10	14	Int *px = &x;
	ff	
	28	
	00	

一些说明

1. 指针变量和其他数据类型的变量一样，在内存中也有自己的数据存储空间
2. 其他数据类型变量所占用的内存区块中，存储的数值表示的是一个整数；指针变量所占用的内存区块中，存储的数值表示的是一个内存单元的地址
3. 内存单元从0开始顺序编址，故指针变量的值本质上也是一个无符号整数
4. 其他数据类型占用存储空间（字节数）通常可能不一致（与其所需要存储的数据相关），但存储指针变量所需占用的存储空间与指针指向数据的类型无关，通常都是相同的（只与系统内存的大小相关）

```
      sizeof(int)      = 4,        sizeof(int *)      = 4
sizeof(double)      = 8,        sizeof(double *)   = 4
sizeof(char)        = 1,        sizeof(char *)     = 4
```

由于编译器和机器的不同，得到的数值可能与书本的有点点差异！

向函数传递变量的地址(指针)

- 传的是指针变量
- 实参与形参
  - 赋的是指针的值（变量的地址），而非指针指向的变量的值
  - 函数返回后，变量的值发生变化（不同于传值）

程序

```
#include <stdio.h>

void swap(int x, int y);

int main() {
    int a, b;
    printf("please input:”);
    scanf("%d%d", &a, &b);
    swap(a, b);
    printf("a = %d; b = %d\r\n", a, b);
    return 0;
}

void swap(int x, int y) {
    int temp = x;
    x = y;
    y = temp;
}
```

```
#include <stdio.h>

void swap(int *x, int *y);

int main() {
    int a, b;
    printf("please input:”);
    scanf("%d%d", &a, &b);
    swap(&a, &b);
    printf("a = %d; b = %d\r\n", a, b);
    return 0;
}

void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
}
```

指针变量的存储空间

- sizeof(int*)
- sizeof(double*)
- sizeof(char*)
- ……

注意事项

- 指针是什么?
- 指针变量是什么？指针变量的值是什么?
- 指针变量的类型是依其指向数据的数据类型来确定的。
- 指针变量的直接访问和间接访问的区别在哪里?
- 永远不要使用未初始化的指针；
- “用”时必须明确知道指针指向了哪里，以及其指向的数据的数据类型；
- 向函数传递数值和指针的区别是什么?

变量的值（数据） VS 变量的存储位置（在哪里）
指针变量的值（地址） VS 指针变量的存储位置（在哪里）
指针变量的值（地址） VS 指针指向的东西（数据）
指针变量的类型 VS 指针指向的数据的类型

向函数传递变量的地址(指针)

- 声明的是指针变量，传递的是地址（地址）
- 实参与形参
  - 形参是指针变量，实参是变量的地址，即给指针变量赋值
  - 函数返回后，给定地址的变量的值可能发生变化

向函数传递变量（传值） VS 向函数传递地址（传指针/地址）

程序分析

```
void Swap(int x, int y)
{
  int temp;
  temp = x;
  x = y;
  y = temp;
}

……
int x, y;
……
Swap(x, y);
……
```

```
void Swap(int * x, int * y)
{
  int *pTemp;
  pTemp = x;
  x = y;
  y = pTemp;
}

……
int x, y;
……
Swap(&x, &y);
……
```

```
void Swap(int * x, int * y)
{
  int pTemp;
  pTemp = *x;
  *x = *y;
  *y = pTemp;
}

……
int x, y;
……
Swap(&x, &y);
……
```

调用函数Swap后，实参的值会变化么？为什么？

让函数返回一个指针

- 将两数中的较大者修改为其值的一半
  - 求两数中的较大数?
  - 求两数中较大数的存储位置?

函数是可以返回指针的!

```
#include <stdio.h>
double* Max(double * x, double * y);
int main(void)
{
    double a, b;
    double *p;
    printf("please input:");
    scanf("%lf%lf", &a, &b);
    p = Max(&a, &b);
    *p = *p / 2;
    printf("a=%lf, b=%lf\n", a, b);
    return 0;
}
double * Max(double * x, double * y)
{
    return (*x > *y) ? x : y;
}
```

空指针和无类型的指针

- 空指针
  - 空：指向为空，啥都没有
  - NULL（\0）
  - “值”为NULL（空）指针变量
  - int * p = NULL;
- 无类型的指针变量
  - 指针指向的数据的数据类型未知
  - void * p; /*思考数据类型的含义*/
  - 使用时需要明确告知p的类型（强制类型转换）
    - *((int *)P); int *p1 = (int *)p;

函数分析

```
void Swap(int * x, int * y)
{
  int * pTemp;
  *pTemp = *x;
  *x = *y;
  *y = *pTemp;
}
```

哪有问题?

如何解决?

向系统（机器）申请一个数据存储位置

- 申请和释放存储空间
  - malloc calloc realloc 申请存储空间
    - 按给定的字节数向系统申请存储空间
    - 返回无类型的指针 void *
- 如何获知存储空间的申请是否成功
  - malloc, calloc, realloc函数若不能从系统中获取给定大小的存储空间时，返回一个空值NULL（\0）。
  - 这些函数并不处理存储空间中存储什么类型的数据，因此，其返回值的类型是void*
- free 释放申请的存储空间

程序分析

```
#include <stdlib.h>
void Swap(int * x, int * y)
{
    int * pTemp = NULL;
    pTemp = (int *)malloc(sizeof(int));        申请sizeof(int)个字节的存储空间
    *pTemp = *x;
    *x = *y;
    *y = *pTemp;
    free(pTemp);                               释放先前申请的存储空间
}
```

简写定义指针变量的方法

- typedef int * pint;
- typedef unsigned int uint;
- typedef unsigned char BYTE;
- typedef short int BOOL;

指针运算

- 指针变量的可以运算?
- 指针变量的值是什么?
- 指针变量可以进行哪些运算?
  - +，-， ++， --
- 如何运算?
  - 参与运算的一个操作数是指针变量，另一个参数是（数值 *sizeof(指针的类型)）

按序访问存储块

- 求键盘输入的多个非0正整数的最大数
  - 正整数的数量未知!
  - 如何定义变量?
  - 如何确定用户输入结束?

指向指针变量的指针变量

- 一个指针变量指向的是另一个指针变量
- 指针变量值所标记的存储位置中还是一个指针

```
int x;
int * p;
int ** pp;
p = &x;
pp = &p;
*p 是什么?
*pp 是什么?
*(*pp) 是什么?
```

```
#include <stdio.h>
int main(void)
{
    double x = 23.44;
    double *p;
    double ** pp;
    p = &x;
    pp = &p;
    printf("pp=%p, *pp=%p, *(*pp)=%lf, x=%lf\n",
              pp,       *pp,       *(*pp),       x);
    return 0;
}
```

指向函数的指针

- 函数指针
  - 函数指针是指向函数的指针变量
  - 函数指针的声明（定义）方法
    - 返回值类型 (* 指针变量名)([形参列表]);
  - 函数指针的赋值
    - int func(int x); /*函数声明*/
    - int (*pfun)(int x); /*函数指针声明（定义）*/
      - typedef int (*PFUN)(int x); /*定义了一个函数指针的数据类型*/
      - PFUN ppfun;
    - pfun = func; /*函数指针赋值*/
      - ppfun = func;
  - 函数指针的使用
    - (*ppfun)(23); /*通过函数指针调用函数*/
      - ppfun(23);

函数指针

- 算术运算
  - 将不同的函数赋给指向函数的指针（变量）
  - 参数序列一致，返回值一致，函数名称不能相同

程序回顾

```
#include <stdio.h>
double* Max(double * x, double * y);
int main(void)
{
    double a, b;
    double *p;
    printf("please input:");
    scanf("%lf%lf", &a, &b);
    p = Max(&a, &b);
    *p = *p / 2;
    printf("a=%lf, b=%lf\n", a, b);
    return 0;
}
```

```
double * Max(double * x, double * y)
{
    int t= *x;
    *x = *y;
    *y = t;
    return (*x > *y) ? x : y;
}
```

会出什么问题?

如何解决?

CONST类型限定符

- int * const p;
  - /*指针变量p 是一个常量*/
- const int * p;
  - /*指针变量p指向的数据是一个常量*/
- const int * const p;
  - /*指针变量p是一个常量，其指向的数据也是一个常量*/

不允许修改实参的数值

```
#include <stdio.h>
double* Max(const double *x, const double *y);
int main(void)
{
    double a, b;
    double *p;
    printf("please input:");
    scanf("%lf%lf", &a, &b);
    p = Max(&a, &b);
    *p = *p / 2;
    printf("a=%lf, b=%lf\n", a, b);
    return 0;
}
double* Max(const double *x, const double *y)
{
    return (*x > *y) ? x : y;
}
```

会有一个警告，提示改变了指针的类型！！

但不影响程序结果。

指针与一维数组

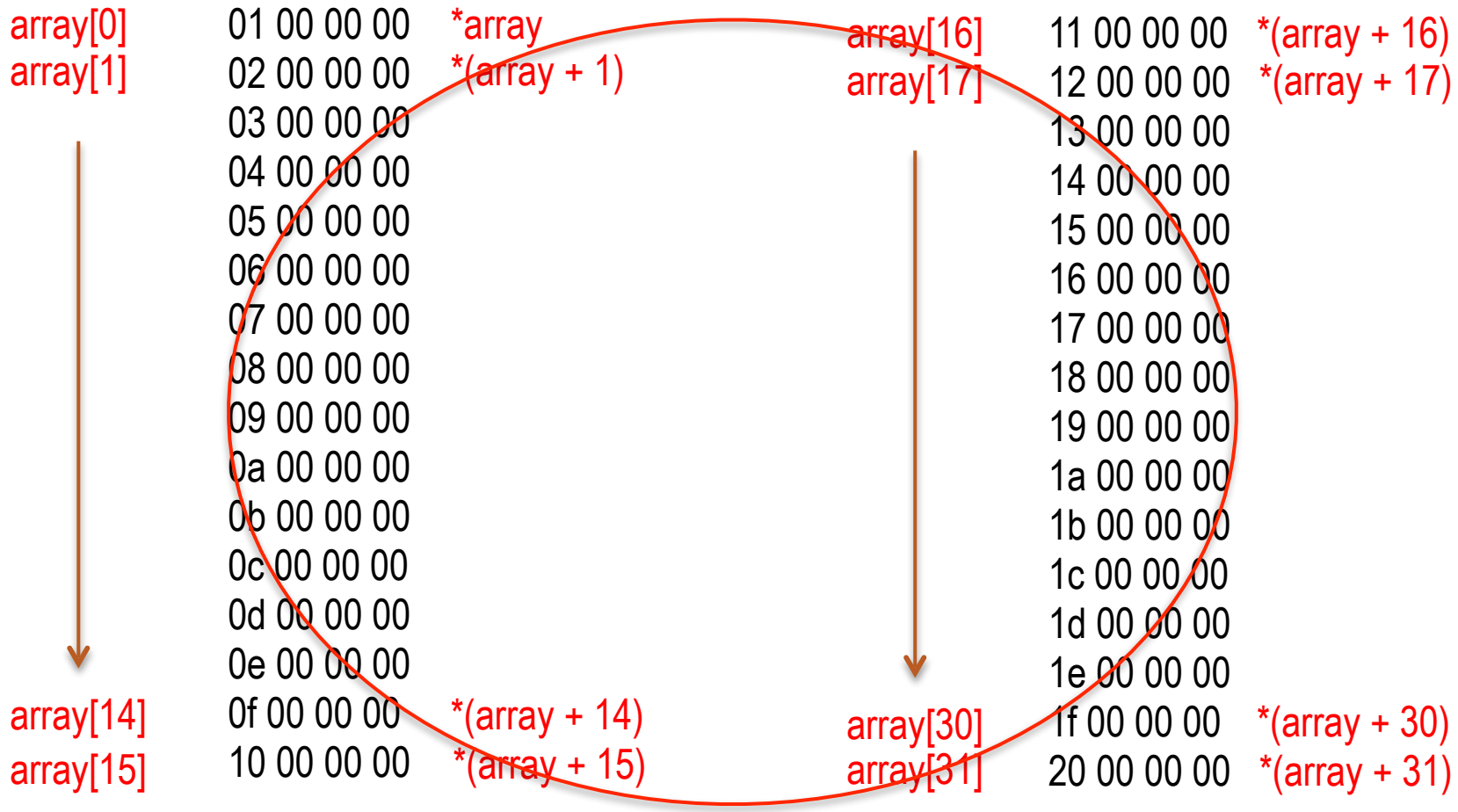
指针

- 存储的是变量的地址
- 指针的类型
- 使用间接访问来获取其指向的变量
- 改变指针的值可以访问其它数据

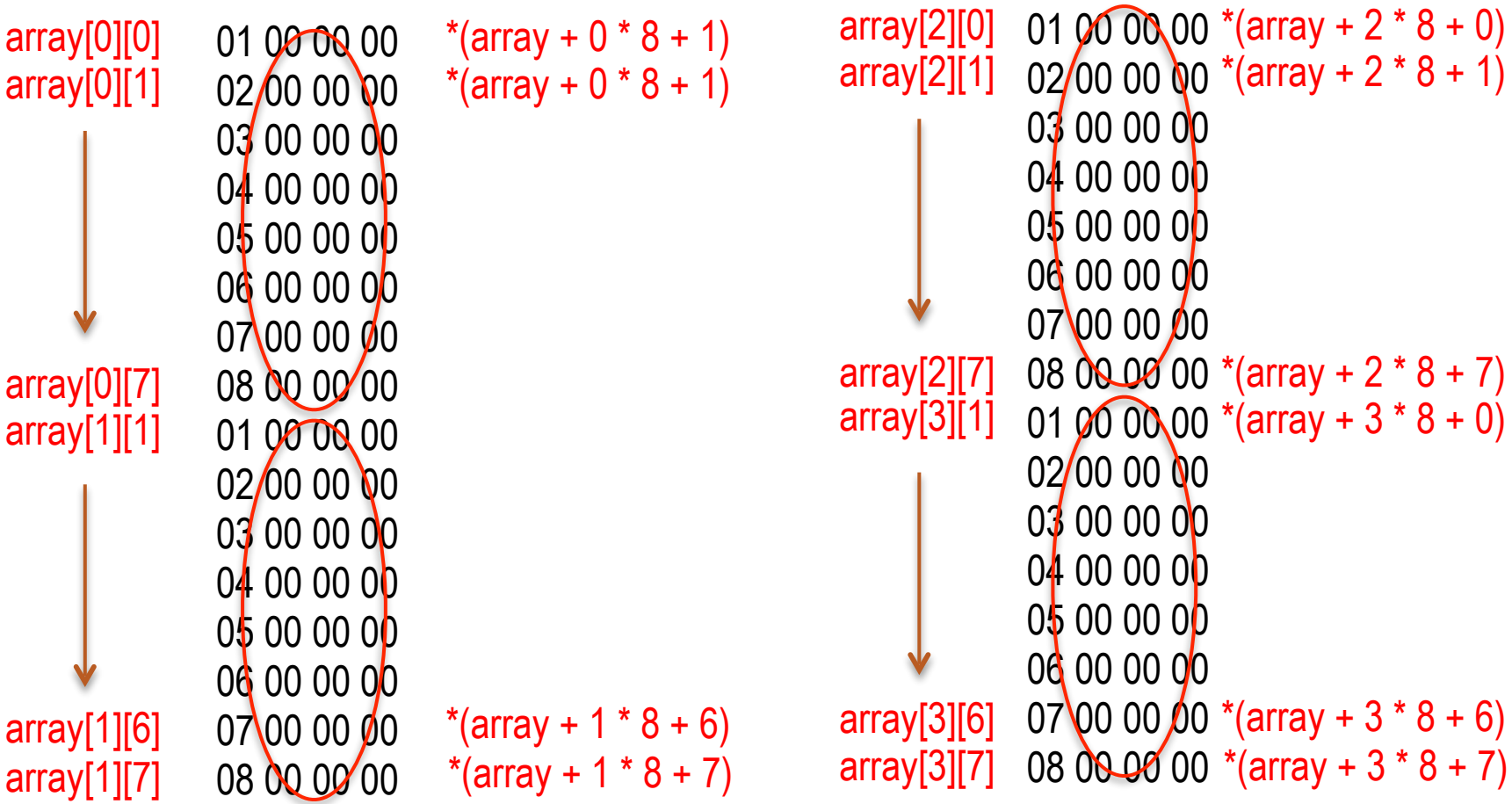
数组

- 存储多个类型相同的变量
- 数组的基类型
- 使用下标来访问数组中的某个变量
- 改变下标的值可以访问不同的变量

一维数组的存储结构


array[0]
array[1]
01 00 00 00 *array
02 00 00 00 *(array + 1)
03 00 00 00
04 00 00 00
05 00 00 00
06 00 00 00
07 00 00 00
08 00 00 00
09 00 00 00
0a 00 00 00
0b 00 00 00
0c 00 00 00
0d 00 00 00
0e 00 00 00
array[14]
array[15]
0f 00 00 00 *(array + 14)
10 00 00 00 *(array + 15)
array[16]
array[17]
11 00 00 00 *(array + 16)
12 00 00 00 *(array + 17)
13 00 00 00
14 00 00 00
15 00 00 00
16 00 00 00
17 00 00 00
18 00 00 00
19 00 00 00
1a 00 00 00
1b 00 00 00
1c 00 00 00
1d 00 00 00
1e 00 00 00
array[30]
array[31]
1f 00 00 00 *(array + 30)
20 00 00 00 *(array + 31)

二维数组的存储结构


array[0][0]
array[0][1]
array[0][7]
array[1][1]
array[1][6]
array[1][7]
01 00 00 00
02 00 00 00
03 00 00 00
04 00 00 00
05 00 00 00
06 00 00 00
07 00 00 00
08 00 00 00
01 00 00 00
02 00 00 00
03 00 00 00
04 00 00 00
05 00 00 00
06 00 00 00
07 00 00 00
08 00 00 00
*(array + 0 * 8 + 1)
*(array + 0 * 8 + 1)
*(array + 1 * 8 + 6)
*(array + 1 * 8 + 7)
array[2][0]
array[2][1]
array[2][7]
array[3][1]
array[3][6]
array[3][7]
01 00 00 00
02 00 00 00
03 00 00 00
04 00 00 00
05 00 00 00
06 00 00 00
07 00 00 00
08 00 00 00
01 00 00 00
02 00 00 00
03 00 00 00
04 00 00 00
05 00 00 00
06 00 00 00
07 00 00 00
08 00 00 00
*(array + 2 * 8 + 0)
*(array + 2 * 8 + 1)
*(array + 2 * 8 + 7)
*(array + 3 * 8 + 0)
*(array + 3 * 8 + 6)
*(array + 3 * 8 + 7)

向函数传递批量数据（数组 VS 数据块）

- 数组传递
  - 数组方式
  - 指针方式
- 数据块传递
  - 传递的是块首指针，块大小（数据个数）
  - 也可以看成是一维数组
- 多维数组的传递问题

指针数组

- 多个指针变量
- 存储指针变量的数组
- 示例
  - 动态二维数组（长度可变）