

# 法律声明

---

- 本课件包括：演示文稿，示例，代码，题库，视频和声音等，小象学院拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意，我们将保留一切通过法律手段追究违反者的权利。



关注 小象学院

---

# 第六课 指针与动态内存分配

林沐

# 内容概述

---

## 指针与动态内存分配(上):

1. 指针的引入
2. 指向指针的指针
3. 一维数组与指针
4. 变量是一个元素的数组
5. 字符串与指针
6. 例1-指针的基础用法
7. 指针的类型与移动长度
8. 使用指针强制操作内存
9. 指针变量作为函数参数
10. 指针作为函数的返回值
11. 不可返回函数内部的静态变量地址
12. 例2-内存池的模拟

## 指针与动态内存分配(下):

1. 指向常量的指针与常量指针
2. 多维数组的首地址
3. 多维数组与指针
4. 指针数组
5. 例3-指针数组与字符串
6. 动态内存分配与释放: malloc与free
7. 动态内存分配二维数组
8. 动态内存分配calloc函数
9. 例4-矩阵重塑(LeetCode 566)

# 指针的引入

指针是C语言最为**强大的**功能之一，使用指针可以保存某个变量在**内存中的地址**，并且通过**操作指针**来对该片内存进行灵活的操作，例如**改变原变量的值**或者**构造复杂的数据结构**。指针一般**初始化为NULL(0)**。**&**是取地址运算，**\***是**间接运算符**(也称取消引用运算符dereferencing operator)，通过**\***可以**访问与修改**指针所指的变量值。

**声明指针变量: 变量类型 \* 变量名      获取指针指向的变量中存储的值 \*变量名**

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

```
int main() {
```

```
    int number = 5;
```

```
    int *pointer = NULL;
```

```
    pointer = &number; //将指针pointer指向变量number
```

```
    printf("variable number's address: %p\n", &number);
```

```
    printf("variable number's value: %d\n", number);
```

```
    printf("variable pointer's size: %d\n", sizeof(pointer));
```

```
    printf("variable pointer's address: %p\n", &pointer);
```

```
    printf("variable pointer's value: %p\n", pointer);
```

```
    printf("variable pointer pointing value: %d\n", *pointer);
```

```
    *pointer = 999; //获取指针指向的变量中存储的值
```

```
    printf("variable number's value: %d\n", number);
```

```
    return 0;
```

int \*pointer

0x0028FF44

0x0028FF40

int number

5

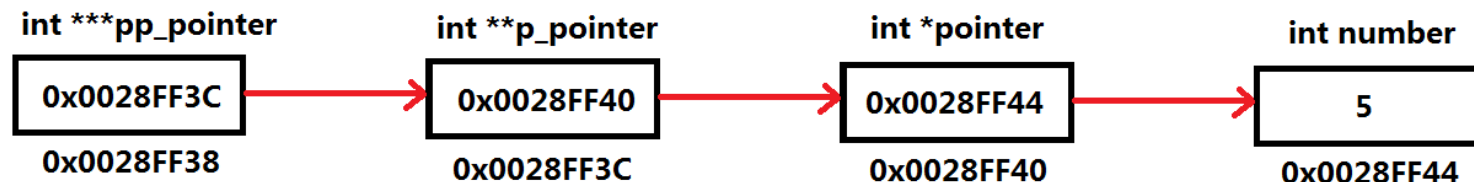
0x0028FF44

```
variable number's address: 0028FF44
variable number's value: 5
variable pointer's size: 4
variable pointer's address: 0028FF40
variable pointer's value: 0028FF44
variable pointer pointing value: 5
variable number's value: 999
请按任意键继续. . .
```



# 指向指针的指针

存储指针的**地址**的指针是**指向指针的指针**，使用\*指针指向普通变量，使用\*\*指针指向\*指针，使用\*\*\*指针指向\*\*指针，...



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

```
int main() {
    int number = 5;
    int *pointer = &number;
    int **p_pointer = &pointer;
    int ***pp_pointer = &p_pointer;

    printf("number's address: %p\n", &number);
    printf("number's value: %d\n", number);
    printf("\n");
    printf("pointer's address: %p\n", &pointer);
    printf("pointer's value: %p\n", pointer);
    printf("pointer pointing value: %d\n", *pointer);
    printf("\n");
    printf("p_pointer's address: %p\n", &p_pointer);
    printf("p_pointer's value: %p\n", p_pointer);
    printf("p_pointer pointing value: %p\n", *p_pointer);
    printf("\n");
    printf("pp_pointer's address: %p\n", &pp_pointer);
    printf("pp_pointer's value: %p\n", pp_pointer);
    printf("pp_pointer pointing value: %p\n", *pp_pointer);
    printf("\n");
    printf("pointer,p_pointer,pp_pointer size = %d, %d, %d\n",
           sizeof(pointer), sizeof(p_pointer), sizeof(pp_pointer));
    return 0;
}
```

```
number's address: 0028FF44
number's value: 5

pointer's address: 0028FF40
pointer's value: 0028FF40
pointer pointing value: 5

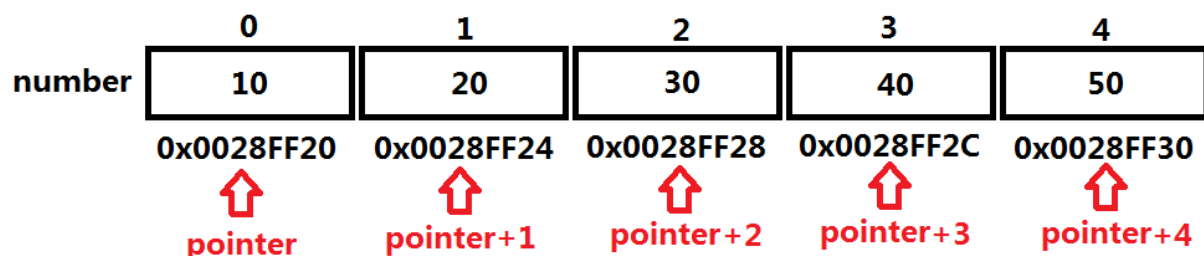
p_pointer's address: 0028FF3C
p_pointer's value: 0028FF40
p_pointer pointing value: 0028FF44

pp_pointer's address: 0028FF38
pp_pointer's value: 0028FF3C
pp_pointer pointing value: 0028FF40

pointer,p_pointer,pp_pointer size = 4, 4, 4
请按任意键继续. . .
```

# 一维数组与指针

数组是**相同类型**的对象集合，通过**数组名称**可以索引数组的各个元素；**数组的名称**实际上是数组的地址，即数组**第一个元素**`a[0]`的地址。同样的，我们可以通过**指针**来遍历、访问、修改数组中的元素值。指针**向前移动**时，移动的单位长度即为该指针大小。



```
#include <stdio.h>
int main(){
    int number[5] = {10, 20, 30, 40, 50};
    int *pointer = number;
    int i;
    printf("number = %p pointer = %p\n", number, pointer);
    printf("sizeof(number) = %d sizeof(pointer) = %d\n",
           sizeof(number), sizeof(pointer));

    printf("\n");
    for (i = 0; i < 5; i++){
        printf("number[%d]'s address = %p\n", i, &number[i]);
        printf("pointer = %p pointer's value = %d\n", pointer, *pointer);
        pointer++;
    }
    printf("\n");
    pointer = number;
    for (i = 0; i < 5; i++){
        printf("number[%d] = %d, %d\n", i, *(pointer + i), pointer[i]);
    }
    return 0;
}
```

```
number = 0028FF20 pointer = 0028FF20
sizeof(number) = 20 sizeof(pointer) = 4

number[0]'s address = 0028FF20
pointer = 0028FF20 pointer's value = 10
number[1]'s address = 0028FF24
pointer = 0028FF24 pointer's value = 20
number[2]'s address = 0028FF28
pointer = 0028FF28 pointer's value = 30
number[3]'s address = 0028FF2C
pointer = 0028FF2C pointer's value = 40
number[4]'s address = 0028FF30
pointer = 0028FF30 pointer's value = 50

number[0] = 10, 10
number[1] = 20, 20
number[2] = 30, 30
number[3] = 40, 40
number[4] = 50, 50
请按任意键继续. . .
```

# 变量是一个元素的数组

C语言中一切内容都与**内存地址**有关，一个元素的变量与数组没有**本质的**区别，都是使用**内存**存储数据，我们可以将**单独的**变量看作是包含**一个元素**的数组。数组下标运算符[]与取消引用运算符\*，本质上实现的是相同的**功能**。

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

```
int main() {  
    int number = 0;  
    int *pointer = &number;  
  
    *pointer = 30;  
    printf("number = %d\n", number);
```

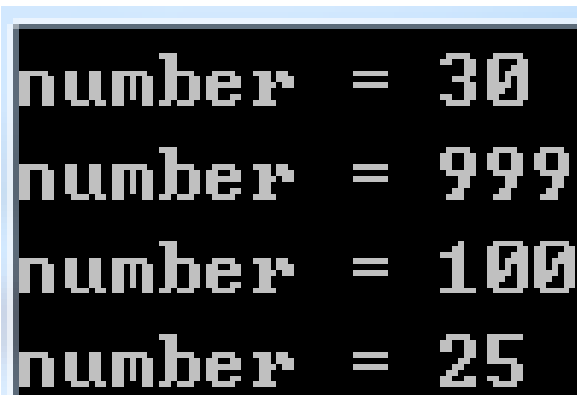
```
    pointer[0] = 999;  
    printf("number = %d\n", number);
```

```
    *&number = 100; //一元运算符从右向左运算，等价于*(&number)  
    printf("number = %d\n", number);
```

```
    (&number)[0] = 25; //数组下标运算优先级更高  
    printf("number = %d\n", number);
```

```
    return 0;
```

```
}
```



```
number = 30  
number = 999  
number = 100  
number = 25
```

# 字符串与指针

利用**指针访问**字符串或修改**字符串**中的数据，就是使用**指针访问**或修改**一维字符数组**中的数据。字符串只是**逻辑上**的概念，要注意的是字符串**以字符串结束符**‘\0’标记结束。通常情况下，操作字符串时使用**指针形式**要比使用**数组形式**更常见，两者**没有本质区别**。

```
#include <stdio.h>

void my_strcpy1(char to[], const char from[]){
    int i = 0;
    while(from[i]){
        to[i] = from[i];
        i++;
    }
    to[i] = '\0';
}

void my_strcpy2(char *to, const char *from){
    while(*from){
        *to = *from;
        to++;
        from++;
    }
    *to = '\0';
}
```

```
int main(){
    char str[10] = "123456789";
    char str2_1[20] = "123456789123456789";
    char str2_2[20] = "123456789123456789";
    my_strcpy1(str2_1, str);
    my_strcpy2(str2_2, str);
    printf("str2_1 = [%s]\n", str2_1);
    printf("str2_2 = [%s]\n", str2_2);
    return 0;
}
```

```
str2_1 = [123456789]
str2_2 = [123456789]
请按任意键继续. . .
```



# 例1-指针的基础用法，课堂练习

定义两个整数，number1与number2，两个指针pointer1与pointer2，使pointer1指向number1，pointer2指向number2；一个指向指针的指针p\_pointer，指向pointer2。实现如下代码：

- 1.通过pointer1与p\_pointer读入数据，存储至number1与number2；
- 2.通过pointer1与p\_pointer将number1与number2修改为999与100。
- 3.通过p\_pointer将pointer2的指向修改为指向number1。

```
#include <stdio.h>
int main() {
    int number1;
    int number2;
    int *pointer1 = &number1;
    int *pointer2 = &number2;
    int **p_pointer = &pointer2;
    printf("Please input two numbers:\n");
    scanf("%d %d", , 
```

```
    printf("number1 = %d, number2 = %d\n", number1, number2);
```

```
    printf("number1 = %d, number2 = %d\n", number1, number2);
    printf("pointer2 point value = %d (number2)\n", *pointer2);
```

```
    printf("pointer2 point value = %d (number1)\n", *pointer2);
    return 0;
}
```

```
Please input two numbers:
15 20
number1 = 15, number2 = 20
number1 = 999, number2 = 100
pointer2 point value = 100 (number2)
pointer2 point value = 999 (number1)
请按任意键继续. . .
```

**3分钟**，填写代码，有问题提出！

# 例1-指针的基础用法，实现

```
#include <stdio.h>

int main() {
    int number1;
    int number2;
    int *pointer1 = &number1;
    int *pointer2 = &number2;
    int **p_pointer = &pointer2;
    printf("Please input two numbers:\n");

    scanf("%d %d", pointer1, *p_pointer);

    printf("number1 = %d, number2 = %d\n", number1, number2);

    *pointer1 = 999;

    **p_pointer = 100;

    printf("number1 = %d, number2 = %d\n", number1, number2);
    printf("pointer2 point value = %d (number2)\n", *pointer2);

    *p_pointer = &number1;

    printf("pointer2 point value = %d (number1)\n", *pointer2);
    return 0;
}
```

```
Please input two numbers:
15 20
number1 = 15, number2 = 20
number1 = 999, number2 = 100
pointer2 point value = 100 (number2)
pointer2 point value = 999 (number1)
请按任意键继续. . .
```

# 指针的类型与移动长度

任意数据类型都有**该类型的指针**所对应，如指向**整型内存**的int\*指针，指向**字符型内存**的char\*指针，指向**双精度内存**的double\*指针，如果只是存储某一地址，**没有特定的类型**，还可以声明为void\*指针，这些指针的**大小都是一样的**。

**指针的类型**最大的作用**操作该类型的内存**并明确指针向前移动的单位(地址)大小，它是该指针**指向类型**的大小。void\*、char\*、int\*、double\*指针**向前移动**的单位长度分别是1字节、1字节、4字节、8字节。

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

```
int main() {
    char array1[10] = {0};
    int array2[10] = {0};
    double array3[10] = {0};
    char *ptr1 = array1;
    int *ptr2 = array2;
    double *ptr3 = array3;
    void *ptr_x = array3;
    int i;
    for (i = 0; i < 10; i++) {
        printf("ptr1=%p ptr2=%p ptr3=%p ptr_x=%p\n",
               ptr1, ptr2, ptr3, ptr_x);

        ptr1++;
        ptr2++;
        ptr3++;
        ptr_x++;
    }
    return 0;
}
```

**//注意各指针向前移动的单位长度不同，  
与该指针的类型直接相关**

```
ptr1=0028FF30 ptr2=0028FF00 ptr3=0028FEB0 ptr_x=0028FEB0
ptr1=0028FF31 ptr2=0028FF04 ptr3=0028FEB8 ptr_x=0028FEB1
ptr1=0028FF32 ptr2=0028FF08 ptr3=0028FEC0 ptr_x=0028FEB2
ptr1=0028FF33 ptr2=0028FF0C ptr3=0028FEC8 ptr_x=0028FEB3
ptr1=0028FF34 ptr2=0028FF10 ptr3=0028FED0 ptr_x=0028FEB4
ptr1=0028FF35 ptr2=0028FF14 ptr3=0028FED8 ptr_x=0028FEB5
ptr1=0028FF36 ptr2=0028FF18 ptr3=0028FEE0 ptr_x=0028FEB6
ptr1=0028FF37 ptr2=0028FF1C ptr3=0028FEE8 ptr_x=0028FEB7
ptr1=0028FF38 ptr2=0028FF20 ptr3=0028FEF0 ptr_x=0028FEB8
ptr1=0028FF39 ptr2=0028FF24 ptr3=0028FEF8 ptr_x=0028FEB9
请按任意键继续. . .
```

# 使用指针强制操作内存

由于存储不同数据类型的**内存与地址**没有本质的区别，我们可以使用指针**强制操作**内存，例如使用一个**double**存储一个长度小于8的**字符串**，需要注意的只是在字符串结尾添加'\0'。

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

```
int main() {
    double temp;
    char *ptr = (char *)(&temp);
    ptr[0] = 'a';
    ptr[1] = 'b';
    ptr[2] = 'c';
    ptr[3] = 'd';
    ptr[4] = 'e';
    ptr[5] = '\\0';
    printf("ptr = [%s]\\n", ptr);
    printf("temp = %lf\\n", temp);
    return 0;
}
```

[illegible]

//使用一个8字节double存储一个长度小于8的字符串"abcde", 注意在字符串末尾添加'\0'

# 指针变量作为函数参数

我们可以将**函数参数**指定为**指针类型**，把**地址**作为相应的变量传递给函数。实际上在一维**数组**或多维数组作为函数的参数时，本质就是将地址作为参数传递给函数，这种传递参数的方式被称为“**按地址传递**”。

按地址传递变量时，函数内部可以**直接访问**到**实际**的变量，即可以在函数体内改变传递进来的**原始变量**的值了。指针变量作为函数参数时，**变量类型 \***与**变量类型 []**的写法是**等价的**。

`#include <stdio.h>` **等价的**

```
void fun(int num[], int n){  
    int i;  
    for (i = 0; i < n; i++){  
        num[i] = 999;  
    }  
}
```

```
void fun2(int *num, int n){  
    int i;  
    for (i = 0; i < n; i++){  
        num[i] = 100;  
    }  
}
```

**//交换x指向的变量与y指向的变量的值**

**//void swap(int x[], int y[])**

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

```
void swap2(int *x, int *y){  
    int temp = x[0];  
    x[0] = y[0];  
    y[0] = temp;  
}
```

```
int main(){  
    int a[10] = {0};  
    int b = 0;  
    int i;
```

```
    fun(&b, 1);  
    fun2(a, 10);  
    printf("a = ");  
    for (i = 0; i < 10; i++){  
        printf("%d ", a[i]);  
    }  
    printf("\n");  
    printf("b = %d\n", b);
```

```
    swap(a + 2, &b);  
    printf("a = ");  
    for (i = 0; i < 10; i++){  
        printf("%d ", a[i]);  
    }  
    printf("\n");  
    printf("b = %d\n", b);
```

```
    swap2(a + 2, &b);  
    printf("a = ");  
    for (i = 0; i < 10; i++){  
        printf("%d ", a[i]);  
    }  
    printf("\n");  
    printf("b = %d\n", b);
```

```
    return 0;
```

```
a = 100 100 100 100 100 100 100 100 100 100  
b = 999  
a = 100 100 999 100 100 100 100 100 100 100  
b = 100  
a = 100 100 100 100 100 100 100 100 100 100  
b = 999  
请按任意键继续. . .
```



# 指针作为函数的返回值

通过指针可以**利用函数的参数**向调用函数(外部)的位置**传递函数计算结果**, 这种方式从**功能上**与函数的返回值**没有本质区别**。函数的返回值也可以**返回指针**(地址), 通过这种方式, 程序可以返回**一个或一组**数据。

```
array[0] = 1, address = 0028FF20
array[1] = 7, address = 0028FF24
array[2] = 2, address = 0028FF28
array[3] = 3, address = 0028FF2C
array[4] = 9, address = 0028FF30
array[5] = 6, address = 0028FF34
array[6] = 5, address = 0028FF38
array[7] = -2, address = 0028FF3C
max_pos1 pointed = 9, max_pos1 = 0028FF30
max_pos2 pointed = 9, max_pos2 = 0028FF30
请按任意键继续. . .
```

```
#include <stdio.h>
int *find_max(int array[], int n){ //找到数组array中最大的元素, 返回该元素的地址
    int *max_pos = array;
    int i;
    for (i = 1; i < n; i++){
        if (*max_pos < array[i]){
            max_pos = array + i;
        }
    }
    return max_pos; //找到数组array中最大的元素, 将该地址存储在max_pos指向的指针中
}

void find_max2(int array[], int n, int **max_pos){
    *max_pos = array;
    int i;
    for (i = 1; i < n; i++){
        if (**max_pos < array[i]){
            *max_pos = array + i;
        }
    }
}

int main(){
    int array[8] = {1, 7, 2, 3, 9, 6, 5, -2};
    int *max_pos1 = 0;
    int *max_pos2 = 0;
    int i;
    for (i = 0; i < 8; i++){
        printf("array[%d] = %d, address = %p\n",
            i, array[i], &array[i]);
    }

    //两个函数功能完全一样, 只是实现不同
    max_pos1 = find_max(array, 8);
    find_max2(array, 8, &max_pos2);

    printf("max_pos1 pointed = %d, max_pos1 = %p\n",
        *max_pos1, max_pos1);

    printf("max_pos2 pointed = %d, max_pos2 = %p\n",
        *max_pos2, max_pos2);

    return 0;
}
```

# 不可返回函数内部的静态变量地址

**//将指针作为函数的返回值，是有极大风险的！**

```
#include <stdio.h> //这么写是绝对错误的，  
int *fun() {  
    int number = 10;  
    return &number;  
}
```

**不可以返回一个静态变量的地址**

```
int *fun2() {  
    int number[10] = {1, 2, 3, 4, 5, 6};  
    return number;  
}
```

**//这么写是绝对错误的，**

**不可以返回一个静态数组的地址**

```
int main() {  
    int *pointer1 = fun();  
    int *pointer2 = fun2();  
    printf("%d %d\n", *pointer1, pointer2[2]);  
    return 0;  
}
```

**，即使程序可能计算正确**

0 3  
请按任意键继续. . .

构建

检查文件依赖性...

正在编译 C:\Users\不可返回函数内部的静态变量地址.c...

[Warning] C:\Users\不可返回函数内部的静态变量地址.c:5: warning: function returns address of local variable

[Warning] C:\Users\不可返回函数内部的静态变量地址.c:10: warning: function returns address of local variable

正在连接...

**WARNING!**

# 例2-内存池的模拟

构建一个10K大小的内存作为**内存池**，后续使用的**变量或数组**，除了循环变量(i, j等)，**均使用该内存池**中的空间。

该程序需要完成如下**功能**：

从键盘读入学生的**名字**(最大长度不超过20个字符，且不包括空格)，该学生的**成绩个数**n，再读入**n个整数**代表这个学生的**n门成绩**，输出该学生的**总成绩**与**平均成绩**。

**程序运行效果：**

```
Please input your name:
LinMu
Please input score number:
8
Please input 8 scores:
98 67 88 70 96 54 83 77

Your name is LinMu.
The scores:
98 67 88 70 96 54 83 77
sum = 633 average = 79.13
```

**思考：**

- 1.内存池即为一片**内存空间**，这片空间如何**定义**？
- 2.内存池需要具备什么样的**功能**？
- 3.如何**使用**内存池？

思考**1分钟**。



# 例2-思考与分析

内存池的空间可能是任意地方申请的数组空间(无论是静态数组或者动态数组), 该数组空间的首地址标记了我们可以使用的内存区域, 该数组的大小即我们可以使用的大小。

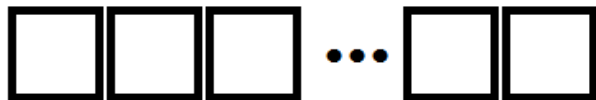
内存池需要实现内存分配的功能, 可以灵活的分配char、int、double类型的变量或者数组。

在使用内存池时, 无论使用哪种类型(char、int、double)的变量或者数组, 均可以用该类型的指针进行操作与使用对应的内存空间。

**//从memory指针指向的空间中, 分配n个(char、int、double)类型的空间, 如果成功分配, 返回分配成功的内存首地址; 否则返回NULL。**

```
char *allocat_char(void *memory, int n) {  
}  
int *allocat_int(void *memory, int n) {  
}  
double *allocat_double(void *memory, int n) {  
}
```

char memory[10240];



**思考:**

- 1.内存池需要用什么样的结构进行存储数据?
- 2.如何记录内存池已使用的内存空间大小?
- 3.思考实现内存池的算法。

**思考1分钟。**

# 例2-调用程序

```
#include <stdio.h>
#define MEMORY_MAX_SIZE 10240

//从memory指针指向的空间中，分配n个(char、
int、double)类型的空间，如果成功分配，返回分配
成功的内存首地址；否则返回NULL。

char *allocat_char(void *memory, int n){
}
int *allocat_int(void *memory, int n){
}
double *allocat_double(void *memory, int n){
}

int main(){
    char memory[MEMORY_MAX_SIZE] = {0};
    run_main_function(memory);
    check_memory(memory);
    return 0;
}
```

```
Please input your name:
LinMu
Please input score number:
8
Please input 8 scores:
98 67 88 70 96 54 83 77

Your name is LinMu.
The scores:
98 67 88 70 96 54 83 77
sum = 633 average = 79.13
```

```
void run_main_function(void *memory){
    int i; //数据输入

    char *name = allocat_char(memory, 20);
    printf("Please input your name:\n");
    scanf("%s", name);
    int *score_num = allocat_int(memory, 1);
    printf("Please input score number:\n");
    scanf("%d", score_num);
    int *score = allocat_int(memory, *score_num);
    printf("Please input %d scores:\n", *score_num);
    for (i = 0; i < *score_num; i++){
        scanf("%d", &score[i]);
    }

    int *sum = allocat_int(memory, 1);
    double *average = allocat_double(memory, 1);
    *sum = 0;
    for (i = 0; i < *score_num; i++){ //计算总成绩
        *sum += score[i]; //与平均成绩
    }
    *average = (double) (*sum) / *score_num;

    printf("\n");
    printf("Your name is %s.\n", name);
    printf("The scores:\n"); //输出
    for (i = 0; i < *score_num; i++){
        printf("%d ", score[i]);
    }
    printf("\n");
    printf("sum = %d average = %.21f\n", *sum, *average);
}
```



# 例2-算法设计

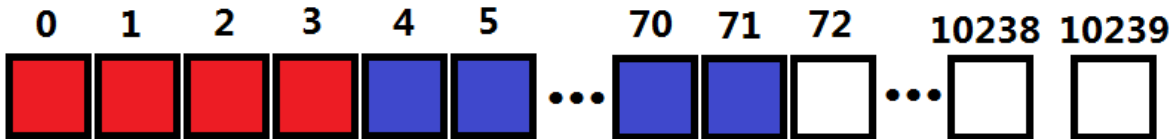
内存池的**存储结构**:

在内存池中, 需要**预留**有一片空间, 记录该内存池**已使用**了多少字节, 该片空间占用**4字节**, 需要存的下一个int型。后续分配**各种类型**(char, int, double)的变量或数组时, 一共使用了多少字节, 该空间的内存中即**记录**多大的整数。

内存池的**分配算法**:

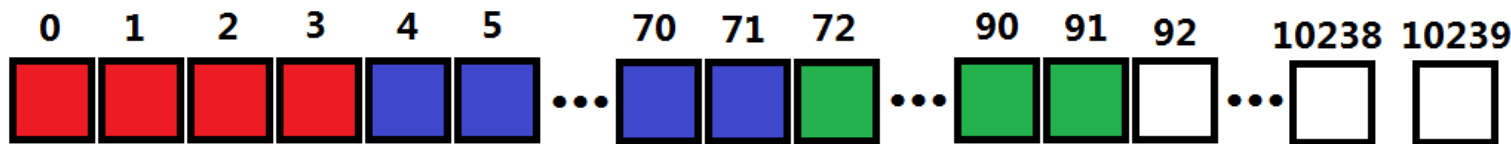
设整型指针\*used\_size指向内存池**最初预留**的4字节空间, 设**新的申请**的空间大小为allocat, 内存池需要从**首地址**向后**偏移sizeof(int) + \*used\_size**位置为新的申请准备内存, 设该地址为new\_memory。如果**没有足够**的内存, 直接返回NULL; 否则, 更新\*used\_size, 并**返回地址**new\_memory。

char memory[10240];



该片空间是4个字节, 存储一个  
int型整数, 表示使用了  
memory数组多少空间, 如68

蓝色代表已经使用的内存  
后续从蓝色箭头指向位置开始使用



↑ 若allocat = 20, 则 若再有申请, 将来在绿色位置开始使用 ↑  
used\_size \*used\_size = 88

# 例2-课堂练习

```
#define MEMORY_MAX_SIZE 10240

void *memory_allocat(void *memory, int allocat){
    int *used_size = 1

    memory = 2

    if ( 3 ){
        return NULL;
    }
    4
    return memory;
}

int get_memory_used(void *memory){
    return *(int *)memory + sizeof(int);
}

char *allocat_char(void *memory, int n){
    return (char *)memory_allocat(memory, 5 * sizeof(char));
}

int *allocat_int(void *memory, int n){
    return (int *)memory_allocat(memory, 5 * sizeof(int));
}

double *allocat_double(void *memory, int n){
    return (double *)memory_allocat(memory, 5 * sizeof(double));
}
```

**3分钟**，填写代码  
，有问题提出！

# 例2-实现

```
#define MEMORY_MAX_SIZE 10240

void *memory_allocat(void *memory, int allocat){
    int *used_size = (int *)memory;
    memory = memory + sizeof(int) + *used_size;
    if (*used_size + allocat > MEMORY_MAX_SIZE){
        return NULL;
    }
    *used_size += allocat;
    return memory;
}

int get_memory_used(void *memory){
    return *(int *)memory + sizeof(int);
}

char *allocat_char(void *memory, int n){
    return (char *)memory_allocat(memory, n * sizeof(char));
}

int *allocat_int(void *memory, int n){
    return (int *)memory_allocat(memory, n * sizeof(int));
}

double *allocat_double(void *memory, int n){
    return (double *)memory_allocat(memory, n * sizeof(double));
}
```

# 例2-测试

```
void check_memory(void *memory){
    printf("\ncheck begin:\n");
    void *check = memory;
    check += sizeof(int);
    printf("name = [%s]\n", (char *)check);
    check += 20 * sizeof(char);
    int *score_num = (int *)check;
    printf("score_num = %d\n", *score_num);
    check += sizeof(int);
    int i;
    for (i = 0; i < *score_num; i++){
        printf("score[%d] = %d\n", i, *(int *)check);
        check += sizeof(int);
    }
    printf("sum = %d\n", *(int *)check);
    check += sizeof(int);
    printf("average = %.2lf\n", *(double *)check);
    check += sizeof(double);
    printf("used_size = %d, %d\n",
        get_memory_used(memory), check - (void *)memory);
}

int main(){
    char memory[MEMORY_MAX_SIZE] = {0};
    run_main_function(memory);
    check_memory(memory);
    return 0;
}
```

```
Please input your name:
LinMu
Please input score number:
8
Please input 8 scores:
98 67 88 70 96 54 83 77

Your name is LinMu.
The scores:
98 67 88 70 96 54 83 77
sum = 633 average = 79.13

check begin:
name = [LinMu]
score_num = 8
score[0] = 98
score[1] = 67
score[2] = 88
score[3] = 70
score[4] = 96
score[5] = 54
score[6] = 83
score[7] = 77
sum = 633
average = 79.13
used_size = 72, 72
请按任意键继续. . .
```

# 课间休息10分钟！

---

## 有问题提出！

# 指向常量的指针与常量指针

在声明指针时，使用**const**关键字指定，该指针**本身存储的值**可以改变，但不能使用该指针修改**指向的内存中存储的值**，这样的指针是**指向常量的指针**(可能原本那个变量不是常量)，这种指针多数用在字符串与大型数据结构的**内存传递**中。

也可以使指针中**存储的地址**不能改变，即**指针的指向**不能改变，我们可以利用该指针修改**指向的内存空间**中存储的数据，这种情况使用const 关键字的方式**略有不同**，该指针称为**常量指针**。在声明指针时我们也可以使指针同时具有上述**两种性质**。

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

```
int main() {
```

```
    int number1 = 10;
```

```
    int number2 = 20;
```

```
    const int *pointer1 = &number1;
```

```
    printf("pointer1 points value: %d\n", *pointer1);
```

```
    pointer1 = &number2; //但我们可以修改pointer1的指向
```

```
    printf("pointer1 points value: %d\n", *pointer1);
```

```
    int *const pointer2 = &number1;
```

```
    *pointer2 = 999;
```

```
    printf("number1 = %d\n", number1); 过它修改number1的值
```

```
    const int *const pointer3 = &number2; //两种性质都具有的指针
```

```
    printf("pointer3 points value: %d\n", *pointer3);
```

```
    return 0;
```

```
}
```

```
pointer1 points value: 10
pointer1 points value: 20
number1 = 999
pointer3 points value: 20
请按任意键继续. . .
```



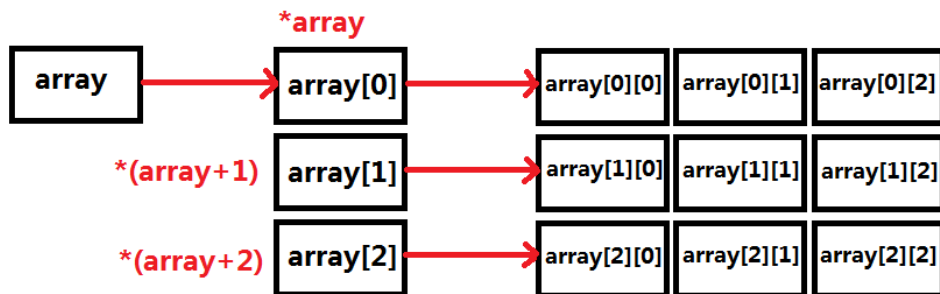
# 多维数组的首地址

实际上**数组名**本身可以看作是一个**常量指针**，该指针的**指向**不可以改变，但我们可以通过该指针**修改**它指向的内存中存储的数据。无论一维数组或多维数组，数组名都是这片内存的**首地址**。

在多维数组中，**前一维度**是后面维度(内存)的首地址，例如，我们将 $n*m$ 的二维数组看作是 $n$ 个长度为 $m$ 的一维数组，有 **$n$ 个指针**指向 $n$ 个一维数组。

```
#include <stdio.h>
int main(){
    int array[3][3] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };
    int i, j;
    printf("array = %p\n", array); //二维数组的首地址
    for (i = 0; i < 3; i++){ //二维数组前一维的地址
        printf("i = %d %p %p %p\n", i, array[i], array + i, *(array + i));
        for (j = 0; j < 3; j++){ //二维数组中的元素
            printf("[%d %d %d] ", array[i][j], *(array[i]+j), (*(array + i)+ j));
        }
        printf("\n");
    }
    return 0;
}
```

```
array = 0028FF10
i = 0 0028FF10 0028FF10 0028FF10
[1 1 1] [2 2 2] [3 3 3]
i = 1 0028FF1C 0028FF1C 0028FF1C
[4 4 4] [5 5 5] [6 6 6]
i = 2 0028FF28 0028FF28 0028FF28
[7 7 7] [8 8 8] [9 9 9]
请按任意键继续. . .
```



# 多维数组与指针

从功能上看，一个多维数组的**首地址**就是一个指针，为了存储该多维数组的首地址，我们不能简单定义**指针**或者**指向指针的指针**指向该首地址，我们需要定义指向n个**单位长度**的指针，这种类型的指针本质上还是一**维指针**。例如，如果要存储一个n\*m的二维数组首地址，我们需要定义指向**m个单位长度**的该二维数组类型的**指针**。

```
#include <stdio.h> //另一种写法就是int ptr[][3]

void fun(int (*ptr)[3]){
    int i, j;
    for (i = 0; i < 3; i++){
        for (j = 0; j < 3; j++){
            ptr[i][j] = 10;
        }
    } //可以使用一维数组(指针)操作二维数组，但前提是得知道
    //二维数组的列，这么写大多数编译器会报错
}

void fun2(int ptr[]){
    int i, j;
    for (i = 0; i < 3; i++){
        for (j = 0; j < 3; j++){
            ptr[i*3+j] = 99;
        }
    }
}

int main(){
    int array[3][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
    int (*ptr)[3] = array; //它很容易和后面讲
    int *ptr2 = array;      解的指针数组所混淆
    printf("array:\n");
    for (i = 0; i < 3; i++){
        for (j = 0; j < 3; j++){
            printf("[%d %d]", ptr[i][j], ptr2[i * 3 + j]);
        }
        printf("\n");
    }
}
```

```
fun(array);
printf("array:\n");
for (i = 0; i < 3; i++){
    for (j = 0; j < 3; j++){
        printf("%d ", ptr[i][j]);
    }
    printf("\n");
}
fun2(array);
printf("array:\n");
for (i = 0; i < 3; i++){
    for (j = 0; j < 3; j++){
        printf("%d ", ptr[i][j]);
    }
    printf("\n");
}
return 0;
```

```
array:
[1 1][2 2][3 3]
[4 4][5 5][6 6]
[7 7][8 8][9 9]
array:
10 10 10
10 10 10
10 10 10
array:
99 99 99
99 99 99
99 99 99
请按任意键继续. . .
```

多维数组与指针.c:24: warning: initialization from incompatible pointer type  
多维数组与指针.c:41: warning: passing arg 1 of 'fun2' from incompatible pointer type

# 指针数组

**指针数组**就是一组某种类型的指针，当处理**字符串数组**时会经常使用到。**指向指针数组的指针**(存储指针数组的首地址)是**指向指针的指针**。

**注意**指针数组的定义方法容易与指向n个单位长度的指针的写法混淆，但他们是**完全不同的**！

```
ptr sizeof = 12 ptrx sizeof = 4
array:
1 2 3
4 5 6
7 8 9
array:
100 100 100
100 100 100
100 100 100
请按任意键继续. . .
```

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

```
int main(){
    int array[3][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9};

    int *ptr[3] = {0}; //注意这两个完全不是一回事！
    int (*ptrx)[3] = NULL; //最直接的就是大小就不一样
```

```
    printf("ptr sizeof = %d ptrx sizeof = %d\n",
           sizeof(ptr), sizeof(ptrx));
```

```
    int i, j;
```

```
    for (i = 0; i < 3; i++){
        ptr[i] = array[i];
    } //将指针数组中的各个指针
    //指向二维数组的各行
```

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

```
    for (i = 0; i < 3; i++){
        for (j = 0; j < 3; j++){
            printf("%d ", ptr[i][j]);
        }
        printf("\n");
    }
```

```
    int **ptr2 = ptr; //使用一个指向指针的指针操作指针数组
```

```
    for (i = 0; i < 3; i++){
        for (j = 0; j < 3; j++){
            ptr2[i][j] = 100;
        }
    }
    printf("array:\n");
    for (i = 0; i < 3; i++){
        for (j = 0; j < 3; j++){
            printf("%d ", array[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

# 例3-指针数组与字符串

**问题描述：**已知正整数n，n的范围是1—100。从键盘读入n个字符串，每个字符串的长度不确定，字符串中只出现小写字母，n个字符串的总长度不超过100000。对这n个字符串进行排序，并打印到屏幕当中。

**任务要求：**

- 1)不可定义如char str[100][100000];这样的二维数组，因为极大的浪费了内存空间。
- 2)排序过程中不要出现字符串的拷贝工作，因为字符串的拷贝有较大的时间开销。

Input Sample:

```
8
zzz
zzzz
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
p
abc
ccckkk
aaa
mmwordilovecoding
```

Output Sample:

```
aaa
abc
ccckkk
mmwordilovecoding
p
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
zzz
zzzz
```

**思考：**

- 1.如何使用一块大的内存空间存储许多字符串，根据题目中的数据范围，该内存定义多大最合适？
  - 2.若要避免排序过程中字符串的拷贝工作，应该利用什么数据类型解决该问题？
- 思考1分钟。

# 例3-算法设计-指针数组排序字符串

对n个字符串进行**排序**，使用一个长度为n的**指针数组**，数组中的每个指针**指向**这n个字符串，排序时**交换**指针数组中**指针的指向**，使得指针数组中的指针的指向是按照字符串**排序后**的顺序排列的，按顺序打印指针数组，即为排序后的字符串。

```
str_ptr[0] -> zzz
str_ptr[1] -> zzzz
str_ptr[2] -> xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
str_ptr[3] -> p
str_ptr[4] -> abc
str_ptr[5] -> ccckkk
str_ptr[6] -> aaa
str_ptr[7] -> mmwordilovecoding
```



# 例3-指针数组排序字符串，课堂练习

```
#include <string.h>

void sort_str(char 1, int str_n){
    int i, j;
    for (i = 0; i < str_n; i++){
        for (j = i + 1; j < str_n; j++){
            if (strcmp(2, 3) > 0){
                char* temp = str_array[i];
                4
                5
            }
        }
    }
}
```

```
str_array:
zzz
zzzz
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
p
abc
ccckkk
aaa
mmwordilovecoding
str_ptr:
aaa
abc
ccckkk
mmwordilovecoding
p
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
zzz
zzzz
```

```
int main(){
    char str_array[8][100] = {
        "zzz",
        "zzzz",
        "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
        "p",
        "abc",
        "ccckkk",
        "aaa",
        "mmwordilovecoding"
    };
    char *str_ptr[8] = {0};
    int i;
    for (i = 0; i < 8; i++){
        str_ptr[i] = str_array[i];
    }
    sort_str(str_ptr, 8);
    printf("str_array:\n");
    for (i = 0; i < 8; i++){
        printf("%s\n", str_array[i]);
    }
    printf("str_ptr:\n");
    for (i = 0; i < 8; i++){
        printf("%s\n", str_ptr[i]);
    }
    return 0;
}
```

**3分钟**，填写代码，  
有问题提出！

# 例3-指针数组排序字符串，实现

```
#include <string.h>

void sort_str(char *str_array[], int str_n){
    int i, j;
    for (i = 0; i < str_n; i++){
        for (j = i + 1; j < str_n; j++){
            if (strcmp(str_array[i], str_array[j]) > 0){
                char* temp = str_array[i];
                str_array[i] = str_array[j];
                str_array[j] = temp;
            }
        }
    }
}
```

```
str_array:
zzz
zzzz
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
p
abc
ccckkk
aaa
mmwordilovecoding
str_ptr:
aaa
abc
ccckkk
mmwordilovecoding
p
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
zzz
zzzz
请按任意键继续. . .
```

```
int main(){
    char str_array[8][100] = {
        "zzz",
        "zzzz",
        "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
        "p",
        "abc",
        "ccckkk",
        "aaa",
        "mmwordilovecoding"
    };
    char *str_ptr[8] = {0};
    int i;
    for (i = 0; i < 8; i++){
        str_ptr[i] = str_array[i];
    }
    sort_str(str_ptr, 8);
    printf("str_array:\n");
    for (i = 0; i < 8; i++){
        printf("%s\n", str_array[i]);
    }
    printf("str_ptr:\n");
    for (i = 0; i < 8; i++){
        printf("%s\n", str_ptr[i]);
    }
    return 0;
}
```

# 例3-算法设计-字符串的存储

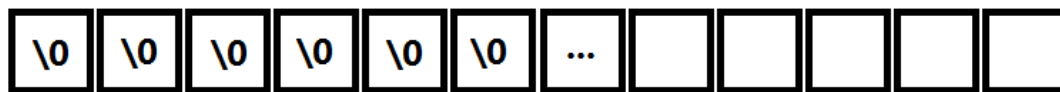
使用**一块连续的空间**存储这些字符串，利用**一个指针**指向这块连续空间中**可以使用**的内存空间，每次利用该指针**读入字符串**后即向后移动该**字符串长度+1**个位置。再利用**字符型指针数组**中的指针指向这些字符串的**首字符**，最后对字符型指针数组**排序**即可。

memory[100100] =

zzz\0zzzz\0xx\0

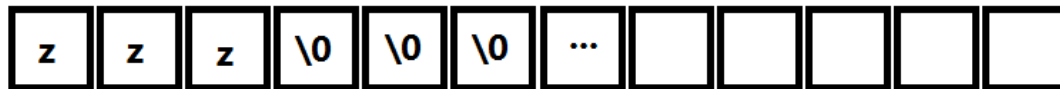
p\0abc\0ccckkk\0aaa\0mmwordilovecoding\0.....

memory:



ptr ↑

memory:



ptr ↑

~~z~~z\0~~z~~zzz\0~~x~~xx\0

~~p~~\0~~a~~abc\0~~c~~ccckkk\0~~a~~aaa\0~~m~~mmwordilovecoding\0.....



# 例3-算法设计-字符串的存储，课堂练习

```
int main() { //一块连续的空间，注意定义的最大长度
    char memory[100100] = {0};
    char *str_array[100] = {0}; //最多100个字符串，
                                //需要用100个字符指针

    char *ptr = 1;

    int n;
    int i;
    scanf("%d", &n);
    for (i = 0; i < n; i++) {
        scanf("%s", 2);
        3
        4
    }
    sort_str(str_array, n);
    for (i = 0; i < n; i++) {
        printf("%s\n", 5);
    }
    return 0;
}
```

```
8
zzz
zzzz
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
p
abc
ccckkk
aaa
mmwordilovecoding
aaa
abc
ccckkk
mmwordilovecoding
p
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
zzz
zzzz
请按任意键继续. . .
```

**3分钟**，填写代码，  
有问题提出！

# 例3-算法设计-字符串的存储，实现

```
int main() {  
    //一块连续的空间，注意定义的最大长度  
    char memory[100100] = {0};  
    char *str_array[100] = {0}; //最多100个字符串，  
                                需要用100个字符指针  
  
    char *ptr = memory;  
  
    int n;  
    int i;  
    scanf("%d", &n);  
    for (i = 0; i < n; i++) {  
        scanf("%s", ptr);  
  
        str_array[i] = ptr;  
  
        ptr = ptr + strlen(ptr) + 1;  
    }  
    sort_str(str_array, n);  
    for (i = 0; i < n; i++) {  
        printf("%s\n", str_array[i]);  
    }  
    return 0;  
}
```

```
8  
zzz  
zzzz  
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  
p  
abc  
ccckkk  
aaa  
mmwordilovecoding  
aaa  
abc  
ccckkk  
mmwordilovecoding  
p  
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  
zzz  
zzzz  
请按任意键继续. . .
```

# 动态内存分配与释放:malloc与free

我们一般使用的内存空间是**静态内存**，这些内存使用的是操作系统的**栈空间**，它们是在程序执行前就预先分配好的。如果我们希望在**程序执行的过程**中分配内存，需要使用操作系统的**堆空间**。**静态的栈空间**中的内存是由操作系统释放的，程序不需要关注这些内存的释放；而使用堆中的**动态空间**时，需要在**程序执行时**进行释放，否则就会造成**内存泄漏**。

包含头文件<stdlib.h>使用以下两个函数：

**void \*malloc(unsigned int num\_bytes);**  
从操作系统的堆空间分配num\_bytes个字节，返回该空间的指针。

**void free(void \*ptr);**  
释放ptr指向的空间内存空间。

```
10
0 100 200 300 400 500 600 700 800 900
请按任意键继续. . .
```

```
#include <stdio.h>
#include <stdlib.h> //要包含这个头文件

int main() {
    int n;
    scanf("%d", &n); //动态分配内存

    int *number = malloc(sizeof(int) * n);
    int i;
    for (i = 0; i < n; i++) {
        number[i] = i * 100;
    }
    for (i = 0; i < n; i++) {
        printf("%d ", number[i]);
    }
    printf("\n");
    free(number); //释放number指向的空间
    return 0;     不写这句话就会造成内存泄漏，
                  这片空间 操作系统没办法再收回了！
}
```

# 动态内存分配二维数组

**//输入二维数组的行n与列m，动态分配一个二维数组:**

```
#include <stdio.h>

int main() {
    int n, m;
    scanf("%d %d", &n, &m); //动态申请长度为n的整型指针数组
    int **number = (int **)malloc(sizeof(int *) * n);
    int i, j;
    for (i = 0; i < n; i++) { //对n个指针，都申请m个空间
        number[i] = (int *)malloc(sizeof(int) * m);
    } //代表n个长度为m的一维数组
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
            number[i][j] = i * m + j;
        }
    }
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
            printf("%d ", number[i][j]);
        }
        printf("\n");
    }

    for (i = 0; i < n; i++) {
        free(number[i]);
    }
    free(number);

    return 0;
}
```

```
3 5
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
请按任意键继续. . .
```

**//注意，先释放行，即每个动态的一维数组的空间，再释放列，指针数组的空间**

# 动态内存分配calloc函数

除了使用malloc函数进行动态内存分配，我们还可以使用**calloc()函数**，同样的使用完这些内存需要**释放空间**。

**void \*calloc(size\_t numElements, size\_t sizeOfElement);**

numElements代表**元素的数目**，sizeOfElement代表**每个元素的大小**，这两个参数的乘积就是要分配的内存空间的大小。

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

```
int main() {
    int n; //与malloc没有本质的区别
    scanf("%d", &n);

    int *number = calloc(n, sizeof(int));
    int i;
    for (i = 0; i < n; i++) {
        number[i] = i * 100;
    }
    for (i = 0; i < n; i++) {
        printf("%d ", number[i]);
    }
    printf("\n");
    free(number); //注意释放动态内存
    return 0;
}
```

```
10
0 100 200 300 400 500 600 700 800 900
请按任意键继续. . .
```

# 例4-矩阵重塑(LeetCode 566)

```
/**
 * Return an array of arrays of size *returnSize.
 * The sizes of the arrays are returned as *columnSizes array.
 * Note: Both returned array and *columnSizes array must be malloced
 * , assume caller calls free().
 */
int** matrixReshape(int** nums, int numsRowSize, int numsColSize,
                    int r, int c, int** columnSizes, int* returnSize) {
```

//它是外界传进来的一个一维指针，  
保存二维数组的各行元素个数

3 \* 5 的二维数组转换为 5 \* 3 的二维数组

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14



0	1	2
3	4	5
6	7	8
9	10	11
12	13	14

设计一个函数matrixReshap，传入一个**二维数组**，\*\*num指针指向该**二维数组**，该二维数组的行为numsRowSize，列为numsColSize，将这个二维数组**转换**为r \* c的二维数组(r是新二维数组的行数，c是新二维数组的列数)并返回**新数组的指针**。如果无法完成这件事，直接将**原数组拷贝**，并返回。

选自 **LeetCode 566**

<https://leetcode.com/problems/reshape-the-matrix/description/>

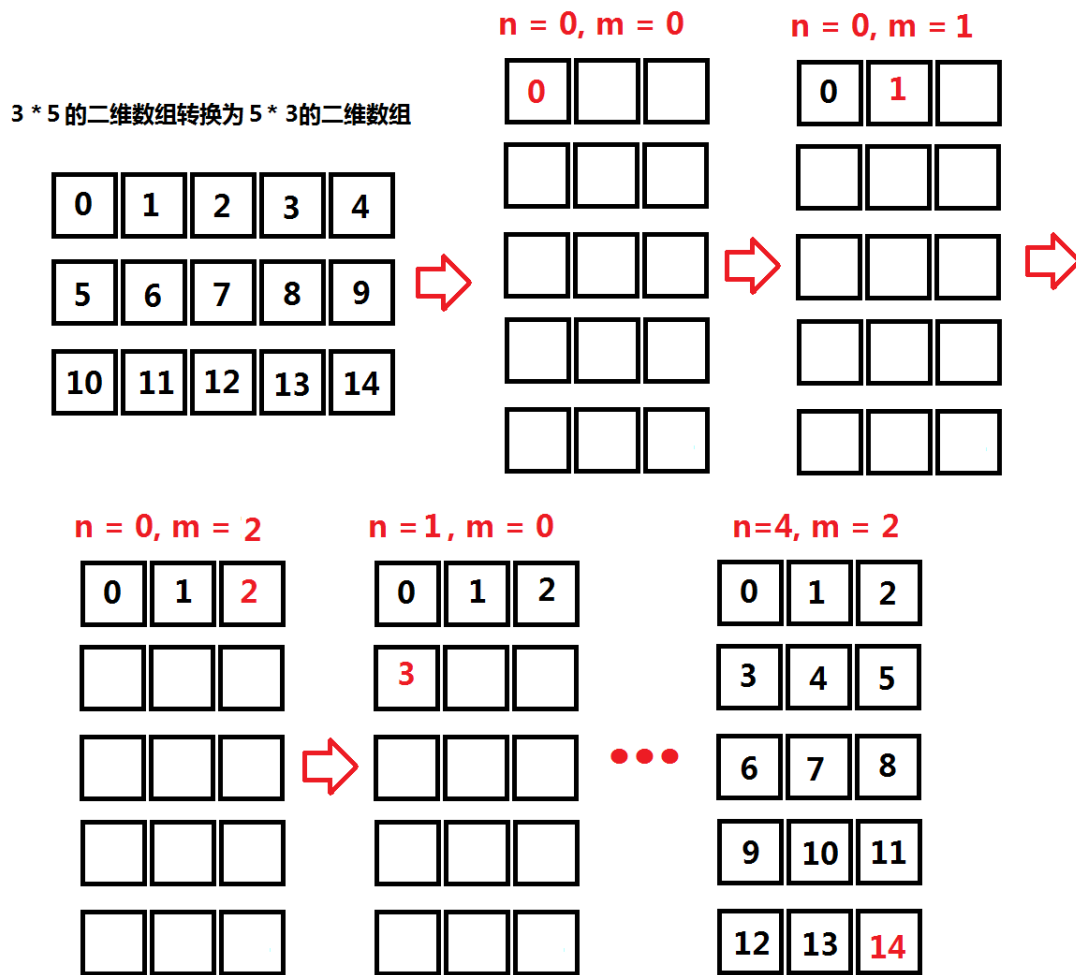
**思考：**

思考该题目的整体算法。

思考**1分钟**。

# 例4-算法设计

解决该问题的算法非常简单，关键是如何**正确的申请**各个动态数组。当完成正确的内存申请后，**遍历原数组**，将原数组向新数组中**填充**，**完成**一行新数组的填充即**换到下一行**的填充，最终将**新的**二维数组填满。



# 例4-课堂练习

```
#include <stdlib.h>
int** matrixReshape(int **nums, int numsRowSize, int numsColSize
, int r, int c, int** columnSizes, int* returnSize) {

    if (numsRowSize * numsColSize != r * c){
        r = numsRowSize;
        c = numsColSize;
    }

    int **new_array = (int **)malloc(1);
    *columnSizes = (int *)malloc(sizeof(int) * r);
    *returnSize = r;
    int i, j;
    for (i = 0; i < r; i++){
        new_array[i] = (int *)malloc(2);
    }

    int m = 0;
    int n = 0;
    for (i = 0; i < numsRowSize; i++){
        for (j = 0; j < numsColSize; j++){
            4 = nums[i][j];
            n++;
            if (5){
                n = 0;
                m++;
            }
        }
    }
    return new_array;
}
```

3 \* 5 的二维数组转换为 5 \* 3 的二维数组

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14



0	1	2
3	4	5
6	7	8
9	10	11
12	13	14

3分钟，填写代码，  
有问题提出！



# 例4-实现

```
#include <stdlib.h>
int** matrixReshape(int **nums, int numRows, int numColSize,
                    , int r, int c, int** columnSizes, int* returnSize) {

    if (numRows * numColSize != r * c) {
        r = numRows;
        c = numColSize;
    }
    int **new_array = (int **)malloc(sizeof(int *) * r);
    *columnSizes = (int *)malloc(sizeof(int) * r);
    *returnSize = r;
    int i, j;
    for (i = 0; i < r; i++) {
        new_array[i] = (int *)malloc(sizeof(int) * c);
        (*columnSizes)[i] = c;
    }
    int m = 0;
    int n = 0;
    for (i = 0; i < numRows; i++) {
        for (j = 0; j < numColSize; j++) {
            new_array[m][n] = nums[i][j];
            n++;
            if (n == c) {
                n = 0;
                m++;
            }
        }
    }
    return new_array;
}
```

3 \* 5 的二维数组转换为 5 \* 3 的二维数组

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14



0	1	2
3	4	5
6	7	8
9	10	11
12	13	14

# 例4-测试与leetcode提交结果

```
int main(){
    int **nums = (int **)malloc(sizeof(int *) * 3);
    int i, j;
    for (i = 0; i < 3; i++){
        nums[i] = (int *)malloc(sizeof(int) * 5);
        for (j = 0; j < 5; j++) {
            nums[i][j] = 5 * i + j;
        }
    }

    int *columnSizes = 0; //注意这里
    int returnSize = 0;

    int **new_array =
        matrixReshape(nums, 3, 5, 5, 3, &columnSizes, &returnSize);

    printf("nums:\n");
    for (i = 0; i < 3; i++){
        for (j = 0; j < 5; j++){
            printf("%d ", nums[i][j]);
        }
        printf("\n");
    }
    printf("new_array:\n");
    for (i = 0; i < returnSize; i++){
        for (j = 0; j < columnSizes[i]; j++){
            printf("%d ", new_array[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

```
nums:
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
new_array:
0 1 2
3 4 5
6 7 8
9 10 11
12 13 14
请按任意键继续. . .
```

Reshape the Matrix

Submission Detail

56 / 56 test cases passed.

Status: **Accepted**

Runtime: 22 ms

Submitted: 0 minutes ago

# 结束

---

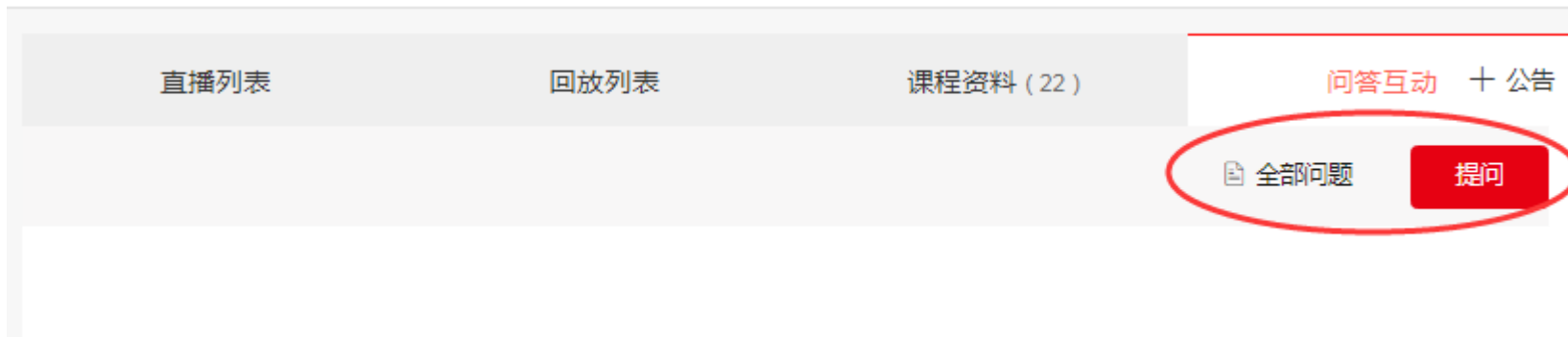
非常感谢大家！

林沐

# 问答互动

在所报课的课程页面，

- 1、点击“全部问题”显示本课程所有学员提问的问题。
- 2、点击“提问”即可向该课程的老师 and 助教提问问题。



# 联系我们

---

## 小象学院：互联网新技术在线教育领航者

— 微信公众号：**小象学院**

