

第8章 指针 (3)



复习回顾

➤ 上次课的内容：

◆ 讲评期中考卷

● 看程序写结果

● 改错题

● 编程题

◆ 指针做函数参数

◆ 指针与数组

◆ 如何看待考试.....

发信人: PzkipfwV (再重申一遍我不是重型坦克~), 信区: CProgramming

标 题: Re: 我终于要下决心开除那个不称职的属下了

发信站: 水木社区 (Sat Sep 4 20:09:42 2010), 站内

作为70后，我来给您说个故事吧~

以前大学的时候，C语言课，班里考试第一名，就是这样的神人一个女生，把C语言当成一门真语言来学——额，我的意思是，你想象一下你怎么学英语的吧.....

就见那女娃的课本上勾满重点，复习的时候如果在教室，就掏出草稿纸来默写语句和函数。当然，有时候她也不在教室的。不要以为她去机房实践了，她是去校园里的小树林，手持课本&课堂笔记大声背诵。除了在机房的课时，她从不自己去学校公共机房。宿舍当然也没有电脑，就这样，期末考了96分，绝杀.....

后记：

此女差点儿拿到保研资格。毕设时原形毕露，在N个哥们帮助下才完成，她付出了请客20多次的代价（此处省略若干字。。。），毕业后去了山东一所二本学校教计算机。

通过指针访问数组元素的方式

➤ 引用一个数组元素，可用下面两种方法：

1. **下标法**，如 $a[i]$ 形式

2. **指针法**，如 $*(a+i)$ 或 $*(p+i)$

其中 a 是数组名， p 是指向数组元素的指针变量，其初值 $p=a$

通过指针访问数组元素实例

➤ 有一个整型数组a，有10个元素，要求用不同方法引用并输出数组中的全部元素。

◆ **解题思路**：引用数组中各元素的值有3种方法

(1) **下标法**

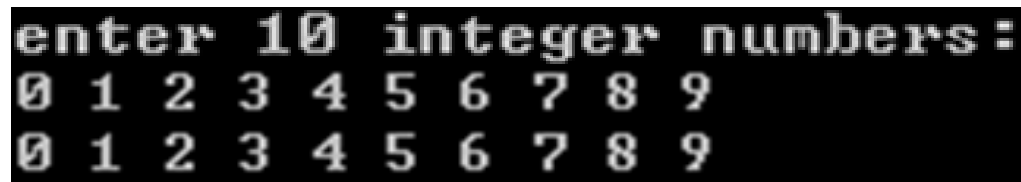
(2) 通过**数组名计算**数组元素地址找出元素的值

(3) 用**指针变量指向**数组元素

◆ 分别写出程序，以资比较分析。

(1) 下标法。

```
1. #include <stdio.h>
2. int main()
3. {
4.     int a[10];  int i;
5.     printf("enter 10 integer numbers:\n");
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. }
```



```
enter 10 integer numbers:
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
```

(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];  int i;
5.     printf("enter 10 integer numbers:\n");
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.}
```

可改为：
`scanf("%d",a+i);`

(3) 用指针变量指向数组元素

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

可改为：

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

若改为以下代码则发生错误

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

三种方法的比较：

① 第(1)和第(2)种方法执行效率相同

◆ 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相同。

② 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($p++$)能提高执行效率

③ 用下标法比较直观，能直接知道是第几个元素。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。

通过指针访问数组元素失败例子

```
#include <stdio.h>
int main()
{
    int *p,i,a[10];
    p = a;
    printf("enter 10 integer numbers:\n");
    for (i=0;i<10;i++)
    {
        scanf("%d",p++);
    }
    for (i=0;i<10;i++,p++)
    {
        printf("%d ",*p);
    }
    printf("\n");
    return 0;
}
```

解决办法：
此处需重新执行
p=a;

**退出循环时p指向a[9]
后面的存储单元**

**因此执行此循环出
问题**

再探数组名作函数参数

- 用数组名作函数参数时，因为实参数组名代表该数组首元素的地址，形参应该是一个指针变量
 - ◆ **表象**：可是为什么函数的形参定义看起来还是一个数组？
 - ◆ **真相**：当函数形参是一个数组名的时候，C编译器都是将形参数组名作为指针变量来处理的

指针作形参，数组名作实参

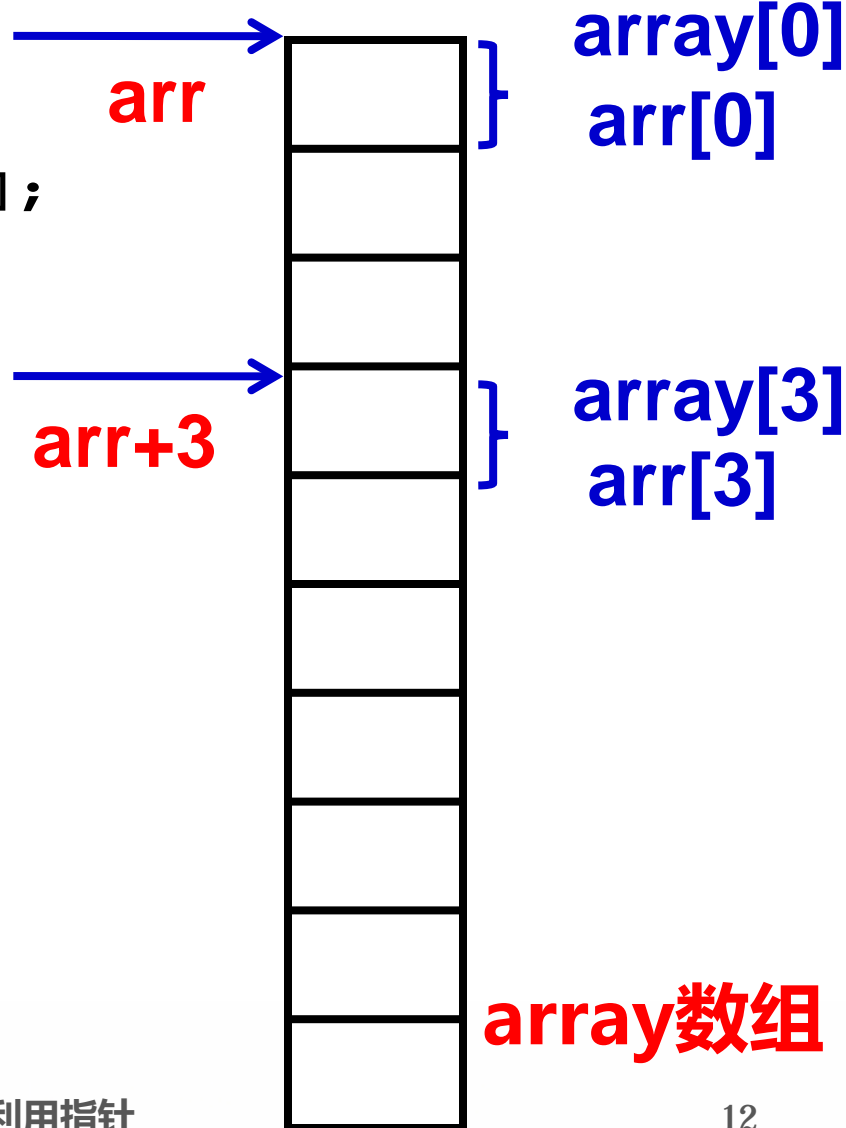
```
int main()  
{  
    void fun(int arr[],int n);  
    int array[10];  
    :  
    fun (array,10);  
    return 0;  
}  
void fun(int arr[],int n)  
{  
    :  
}
```

编译器的理解：

fun(int *arr, int n)

披着数组外皮的指针

```
int main()  
{  
    void fun(int arr[],int n);  
    int array[10];  
    |  
    fun (array,10);  
    return 0;  
}  
  
void fun(int arr[],int n)  
{  
    |  
}
```



形参数组名与实参数组名的不同

- 实参数组名是指针常量，但形参数组名是按指针变量处理
- 在函数调用进行时，形参数组名的值就是实参数组首元素的地址
- 证据：在函数执行期间，形参数组可以再被赋值

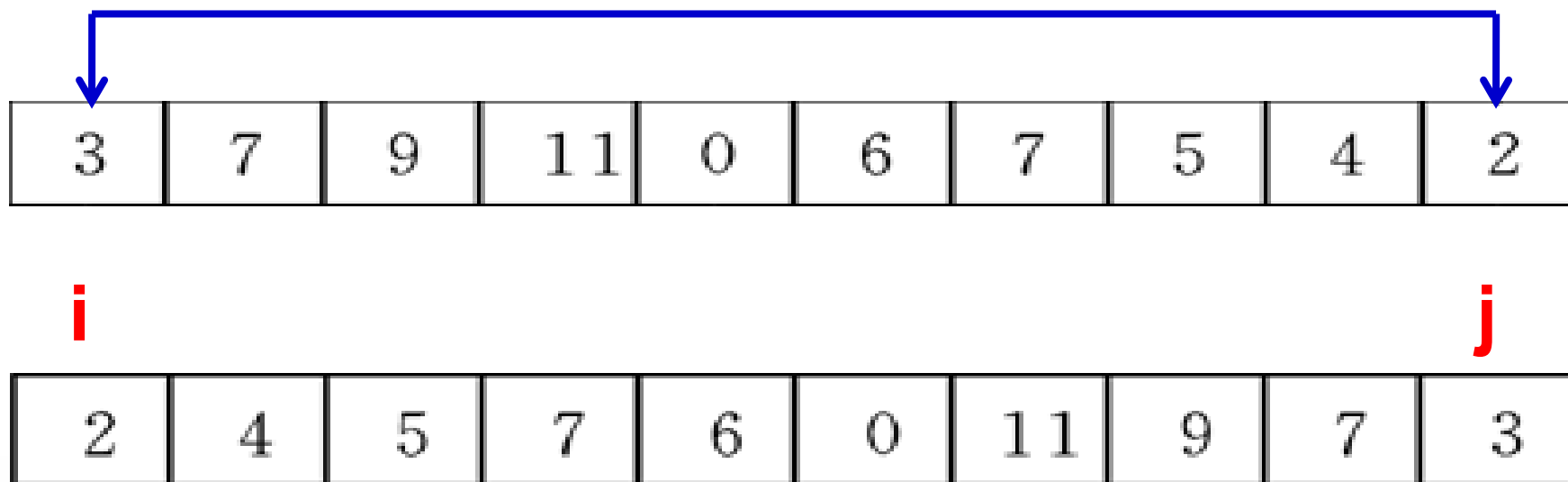
```
void fun (int arr[ ],int n)
{
    printf( "%d\n", *arr);
    arr = arr+3;
    printf( "%d\n", *arr);
}
```



数组名作函数参数应用实例

➤ 将数组a中n个整数按相反顺序存放

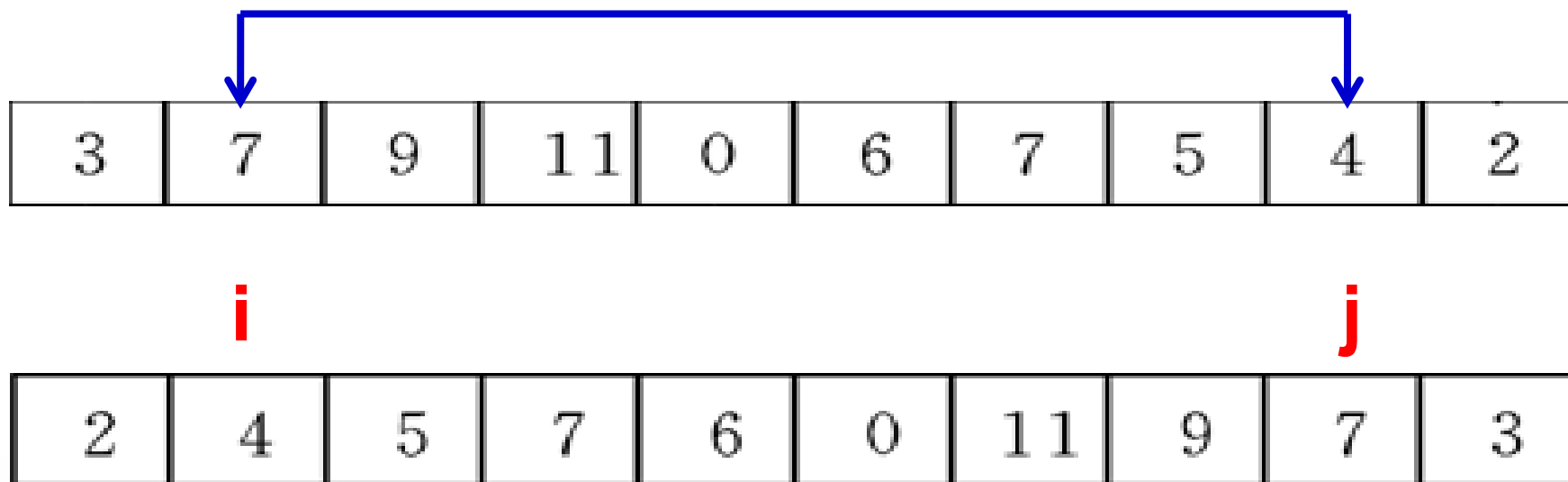
◆ 解题思路：将a[0]与a[n-1]对换，
.....将a[4]与a[5]对换。



数组名作函数参数应用实例

➤ 将数组a中n个整数按相反顺序存放

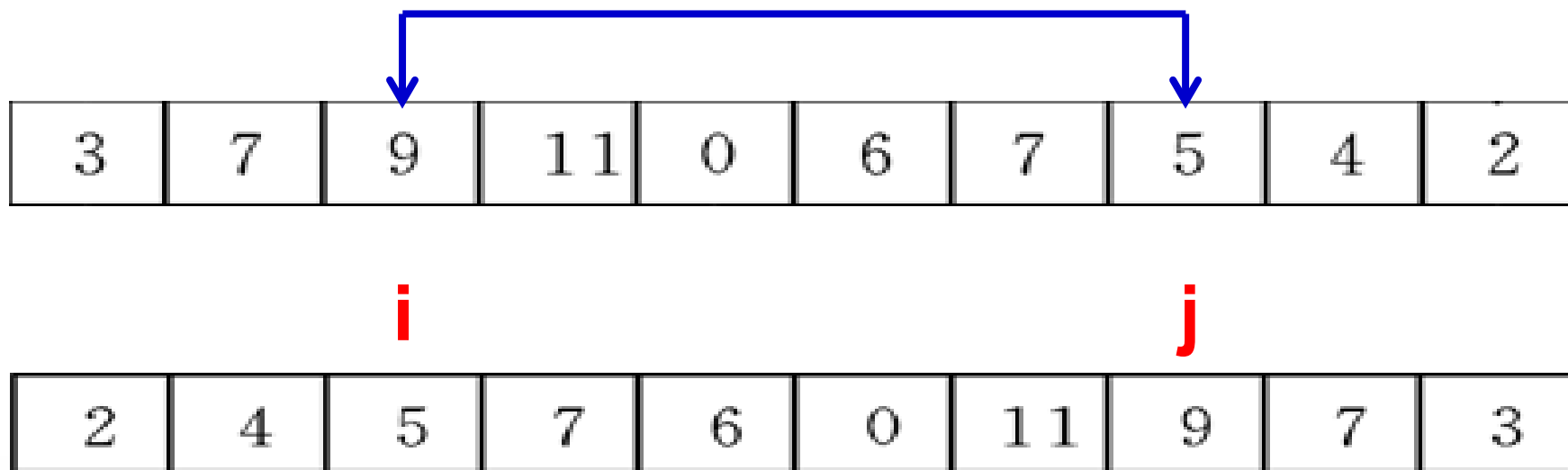
◆ 解题思路：将a[0]与a[n-1]对换，
.....将a[4]与a[5]对换。



数组名作函数参数应用实例

➤ 将数组a中n个整数按相反顺序存放

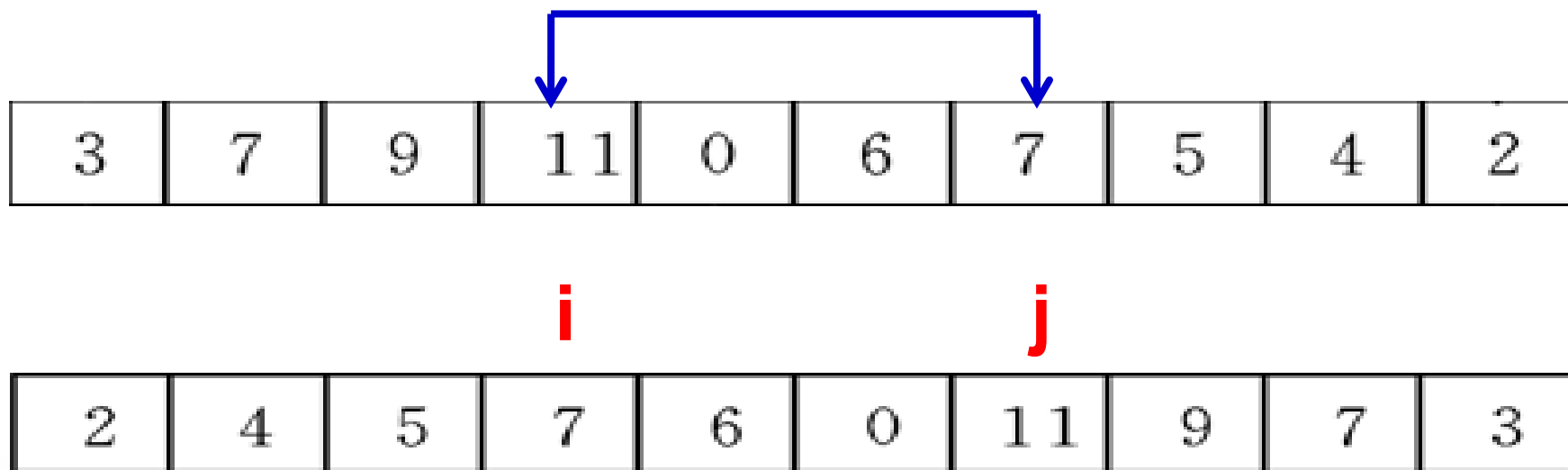
◆ 解题思路：将a[0]与a[n-1]对换，
.....将a[4]与a[5]对换。



数组名作函数参数应用实例

➤ 将数组a中n个整数按相反顺序存放

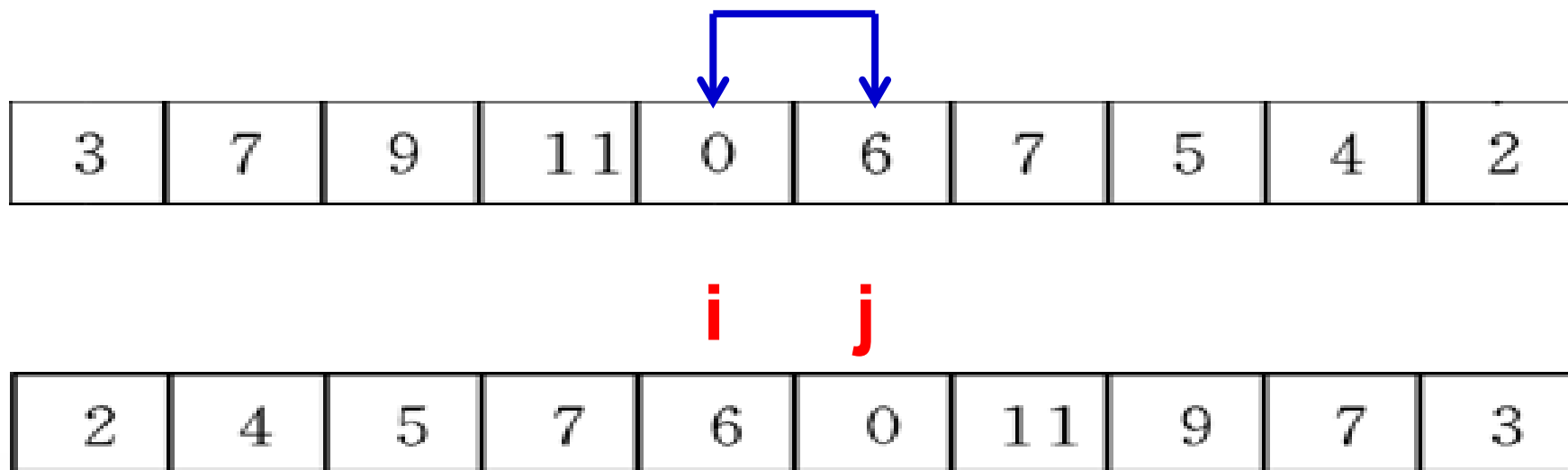
◆ 解题思路：将a[0]与a[n-1]对换，
.....将a[4]与a[5]对换。



数组名作函数参数应用实例

➤ 将数组a中n个整数按相反顺序存放

◆ 解题思路：将a[0]与a[n-1]对换，
.....将a[4]与a[5]对换。



数组名作函数参数应用实例

```
1. #include <stdio.h>
2. int main()
3. {
4.     void inv(int x[], int n);
5.     int i, a[10]={3,7,9,11,0,6,7,5,4,2};
6.     for (i=0;i<10;i++)
7.         printf("%d ",a[i]);
8.     printf("\n");
9.     inv(a,10);
10.    for (i=0;i<10;i++)
11.        printf("%d ",a[i]);
12.    printf("\n");
13.    return 0;
14. }
```

```
void inv(int x[ ],int n)
{
    int temp,i,j,m=(n-1)/2;
    for (i=0;i<=m;i++)
    {
        j = n-1-i;
        temp=x[i]; x[i]=x[j]; x[j]=temp;
    }
}
```

优化

```
void inv(int x[ ],int n)
{
    int temp,*p,*q;
    p=x; q=x+n-1;
    for ( ; p<q; p++,q--)
    {
        temp=*p; *p=*q; *q=temp;
    }
}
```

改写上例，指针变量作实参

```
#include <stdio.h>
int main()
{
    void inv(int *x, int n);
    int i, arr[10], *p = arr;
    for (i=0; i<10; i++, p++)
        scanf("%d", p);
    p = arr;
    inv(p, 10);
    for (p=arr; p<arr+10; p++)
        printf("%d ", *p);
    printf("\n");
    return 0;
}
```

注意指针的初始化
不可少!!!

注意指针做实参时
应该指向哪里

用指针方法进行排序

➤ 用指针方法对10个整数按由大到小顺序排序。

➤ **解题思路：**

- ◆ 在主函数中定义数组a存放10个整数，定义int *型指针变量p指向a[0]
- ◆ 定义函数sort使数组a中的元素按由大到小的顺序排列
- ◆ 在主函数中调用sort函数，用指针p作实参
- ◆ 用选择法进行排序

用指针方法进行排序

```
1. #include <stdio.h>
2. int main()
3. {
4.     void sort(int x[ ],int n);
5.     int i,*p,a[10];
6.     p=a;
7.     for (i=0;i<10;i++)
8.         scanf("%d",p++);
9.     p=a;
10.    sort(p,10);
11.    for (p=a,i=0;i<10;i++)
12.    {
13.        printf("%d ",*p);
14.        p++;
15.    }
16.    printf("\n");
17.    return 0;
18. }
```

注意p的初始化

数组下标方式的实现

```
void sort(int x[],int n)
{
```

```
    int i,j,k,t;
    for (i=0;i<n-1;i++)
    {
```

```
        k=i;
        for (j=i+1;j<n;j++)
            if (x[j]>x[k])
                k=j;
```

```
        if (k!=i)
        {
```

```
            t=x[i];
            x[i]=x[k];
            x[k]=t;
```

```
        }
```

```
    }
```

```
}
```

void sort(int *x,int n)

可改为：

```
if (*(x+j)>*(x+k))
    k=j;
```

可改为：

```
t=*(x+i);
*(x+i)=*(x+k);
*(x+k)=t;
```


指针方式的实现

```
void sort(int *x, int n)
{
    int i,j,k,t;
    for (i = 0; i < n-1; i++)
    {
        k = i;
        for (j = i+1; j < n; j++)
            if (*(x+j) > *(x+k))
                k = j;
        if (k != i)
        {
            t = *(x+i);
            *(x+i) = *(x+k);
            *(x+k) = t;
        }
    }
}
```



12 34 5 689 -43 56 -21 0 24 65
689 65 56 34 24 12 5 0 -21 -43

指针与数组之辨

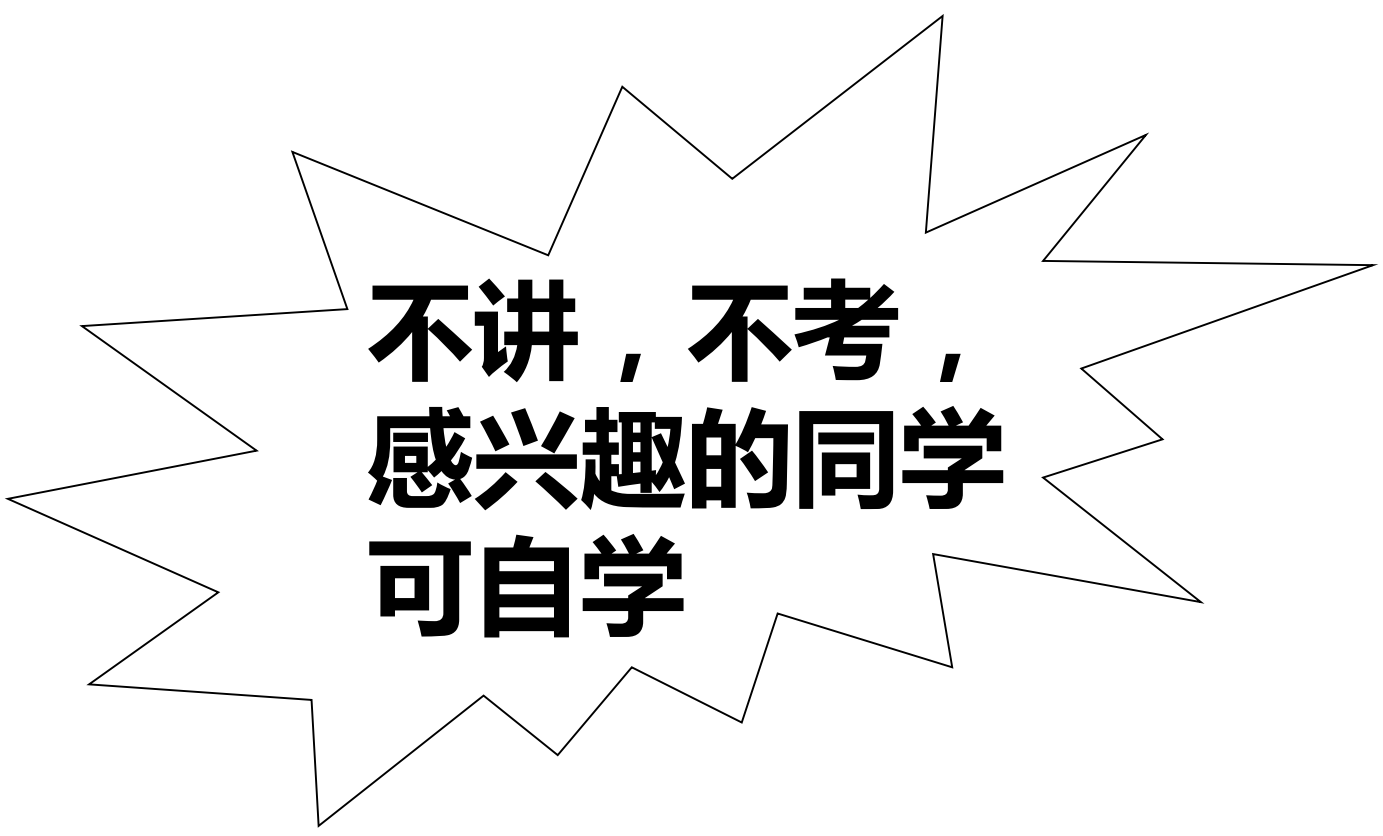
➤ 指针就是地址！

- ◆在32位系统中，任何一种指针变量都占4个字节
- ◆指针变量的值应为某个内存地址
- ◆指针可以指向任何内存单元，但不是任何地方都允许通过指针访问

➤ 数组就是数组，和指针无关！

- ◆数组的大小由定义数组时必须指定的数组元素的类型和个数决定，数组元素可以是任何类型数据。

通过指针引用多维数组



**不讲，不考，
感兴趣的同学
可自学**

指针与字符串

➤ 字符串是存放在字符数组中的。引用一个字符串，可以用以下两种方法。

(1) **用字符数组存放一个字符串**，可以通过数组名和格式声明“%s”输出该字符串，也可以通过数组名和下标引用字符串中一个字符。

(2) **用字符指针变量指向一个字符串常量**，通过字符指针变量引用字符串常量。

如：`char*s="Hello"; printf("%s\n", s);`

不同方式引用字符串元素的例子

➤ 定义一个字符数组，在其中存放字符串 “I love China!”，输出该字符串和第8个字符。

◆ **解题思路**：定义字符数组string，对它初始化，由于在初始化时字符的个数是确定的，因此可不必指定数组的长度。用数组名string和输出格式%s可以输出整个字符串。用数组名和下标可以引用任一数组元素。

不同方式引用字符串元素的例子

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

```
int main()
```

```
{
```

string↓

↓ string+7

```
char string[]="I love China!";
```

```
printf("%s\n",string);
```

```
printf("%c\n",string[7]);
```

```
return 0;
```

```
}
```



```
I love China!  
C
```

不同方式引用字符串元素的例子

➤ 通过字符指针变量输出一个字符串。

◆ **解题思路**：可以不定义字符数组，只定义一个字符指针变量，用它指向字符串常量中的字符。通过字符指针变量输出该字符串。

不同方式引用字符串元素的例子

```
#include <stdio.h>
int main()
{
    string
    char *string="I love China!";
    printf("%s\n", string);
    return 0;
}
```

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

I love China!


指针方式引用字符串元素的例子

```
#include <stdio.h>
int main()
{
    string↓
    char *string = "I love China!";
    printf("%s\n", string);

    string="I am a student.";
    printf("%s\n",string);
    return 0;
}
```

指针方式引用字符串元素的例子

```
#include <stdio.h>
int main()
{
    char *string = "I love China!";
    printf("%s\n", string);
    string↓
    string="I am a student.";
    printf("%s\n",string);
    return 0;
}
```



```
I love China!
I am a student.
```

地址方式处理字符串元素的例子

➤ 将字符串a复制为字符串b，然后输出字符串b。

◆ **解题思路**：定义两个字符数组a和b，用 “I am a student.” 对a数组初始化。将a数组中的字符逐个复制到b数组中。可以用不同的方法引用并输出字符数组元素，今用地址法算出各元素的值。

地址方式处理字符串元素的例子

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

```
int main()
```

```
{
```

```
    char a[ ]="I am a student.", b[20];
```

```
    int i;
```

```
    for (i=0;*(a+i)!='\0';i++)
```

```
        *(b+i)=*(a+i);
```

```
    *(b+i)='\0';
```

可替换为 `printf("string b is:%s\n",b);`

```
    printf("string a is:%s\n",a);
```

```
    printf("string b is:");
```

```
    for (i=0;b[i]!='\0';i++)
```

```
        printf("%c", b[i]);
```

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

```
    return 0;
```

```
}
```

```
string a is:I am a student.
string b is:I am a student.
```

指针方式处理字符串元素的例子

➤ 改用指针变量来处理上一个例子的问题。

◆ **解题思路**：定义两个指针变量p1和p2，分别指向字符数组a和b。改变指针变量p1和p2的值，使它们顺序指向数组中的各元素，进行对应元素的复制。

指针方式处理字符串元素的例子

```
#include <stdio.h>
int main()
{
    char a[]="I am a student.",b[20],*p1,*p2;
    p1=a;  p2=b;
    for (    ; *p1!='\0'; p1++,p2++)
        *p2=*p1;
    *p2='\0';
    printf("string a is:%s\n",a);
    printf("string b is:%s\n",b);
    return 0;
}
```

```
string a is:I am a student.
string b is:I am a student.
```

字符指针作函数参数

- 如果想把一个字符串从一个函数“传递”到另一个函数，可以用**地址传递**的办法，即用**字符数组名作参数**，也可以用**字符指针变量作参数**。
- 在被调用的函数中**可以改变**字符串的内容
- 在主调函数中可以引用改变后的字符串。

字符指针作函数参数的例子

➤ 用函数调用实现字符串的复制。

◆ **解题思路**：定义一个函数

`copy_string`用来实现字符串复制的功能，在主函数中调用此函数，函数的形参和实参可以分别用字符数组名或字符指针变量。分别编程，以供分析比较。

(1) 用**字符数组名**作为函数参数

```
#include <stdio.h>

int main()
{
    void copy_string(char from[], char to[]);
    char a[]="I am a teacher.";
    char b[]="you are a student.";
    printf("a=%s\nb=%s\n",a,b);
    printf("copy string a to string b:\n");
    copy_string(a, b);
    printf("a=%s\nb=%s\n",a,b);
    return 0;
}
```

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

```
a=I am a teacher.
b=You are a student.
copy string a to string b:
a=I am a teacher.
b=I am a teacher.
```

(2)用字符型指针变量作实参

➤ 仅需要修改主函数代码，如下

```
#include <stdio.h>
int main()
{
    void copy_string(char from[], char to[]);
    char a[]="I am a teacher.";
    char b[]="you are a student.";
    char *from = a, *to = b;
    printf("a=%s\nb=%s\n", a, b);
    printf("\ncopy string a to string b:\n");
    copy_string(from, to);
    printf("a=%s\nb=%s\n", a, b);
    return 0;
}
```

(3)用字符指针变量作形参和实参

```
#include <stdio.h>
int main()
{
    void copy_string(char *from, char *to);
    char *a = "I am a teacher.";
    char b[] = "You are a student.";
    char *p = b;
    printf("a=%s\nb=%s\n", a, b);
    printf("\ncopy string a to string b:\n");
    copy_string(a, p);
    printf("a=%s\nb=%s\n", a, b);
    return 0;
}
```

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

函数体有多种简化写法，请见主教材P262

字符数组指针神应用：输出自己

```
#include <stdio.h>
char*f="#include<stdio.h>%cchar*f=%c%s%c;%c"
int main(){printf(f,10,34,f,34,10,10);return 0;}%c";
int main(){printf(f,10,34,f,34,10,10);return 0;}
```

```
#include<stdio.h>
char*f="#include<stdio.h>%cchar*f=%c%s%c;%c"
int main(){printf(f,10,34,f,34,10,10);return 0;}%c";
int main(){printf(f,10,34,f,34,10,10);return 0;}
```

试分析一下这是如何做到的？

字符指针和字符数组的比较

➤ 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有**区别**的，不应混为一谈，主要有以下几点。

(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1个字符的地址），**决不是将字符串放到字符指针变量中。**

字符指针和字符数组的比较

➤ 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有**区别**的，不应混为一谈，主要有以下几点。

(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。

`char *a; a="I love China!";` **对**

`char str[14];str[0]='I';` **对**

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

字符指针和字符数组的比较

- 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有**区别**的，不应混为一谈，主要有以下几点。

(3) 初始化的含义

`char *a="I love China !";` 与

`char *a; a="I love China !";` 等价

`char str[14]="I love China !";` 与

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

字符指针和字符数组的比较

- 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有**区别**的，不应混为一谈，主要有以下几点。

(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，只分配一个指针变量所占的存储单元

字符指针和字符数组的比较

- 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有**区别**的，不应混为一谈，主要有以下几点。

(4) 存储单元的内容

```
char *a;
```

```
scanf("%s",a); 错,因为a还没有分配存储空间
```

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

```
a = str;
```

```
scanf("%s", a); 对
```

字符指针和字符数组的比较

➤ 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有**区别**的，不应混为一谈，主要有以下几点。

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

➤ 例如，可以这样改变指针变量的值。

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

```
int main()
```

```
{
```

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

```
    a = a+7;
```

```
    printf("%s\n",a);
```

```
    return 0;
```

```
}
```

这里不能改为

```
char a[]="I love China!";
```

China!

字符指针和字符数组的比较

➤ 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有**区别的**，不应混为一谈，主要有以下几点。

(6) **字符数组中各元素的值是可以改变的**，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**。

```
char a[]="House", *b="House";
```

```
a[2]='r';      对      b[2]='r';      错
```

字符指针和字符数组的比较

- 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有**区别**的，不应混为一谈，主要有以下几点。

(7) 引用数组元素

对字符数组可以用**下标法**和**地址法**引用数组元素，如`a[5]`或`*(a+5)`。如果字符指针变量`p=a`，则也可以用指针变量带下标的形式和地址法引用，比如`p[5]`或`*(p+5)`。假设`char *a="Xiamen"`；则`a[5]`与`*(a+5)`的值均为第6个字符，即字母'n'

字符指针和字符数组的比较

➤ 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有**区别**的，不应混为一谈，主要有以下几点。

(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。

比如，`char *f="%d\n"; printf(f, 10);`

或者，`char f[]="%d\n"; printf(f, 10);`

均相当于 `printf("%d\n", 10);`

静态数组存储数据的弊端

- 为数组**分配固定大小**的内存称为静态数组
- 无法预先确定要使用多大的数组？那就定义一个足够大的！
 - ◆ 例如一个存储人名的字符数组，能存4个汉字
 - 张飞 **Accept**, 刘备 **Accept**, 赵子龙 **Accept**
 - 乔伊·亚历山大·比基·卡利斯勒·达夫·埃利奥特·福克斯·伊维鲁莫·马尔尼·梅尔斯·帕特森·汤普森·华莱士·普雷斯頓，
Runtime Error!
- 定长很可能意味着（1）浪费空间（2）空间不足

如何实现动态内存管理

- C语言编译系统的库函数提供一系列用于实现动态内存管理的库函数
- ANSI 标准建议的4个相关函数
 - ◆ **malloc** : **m**emory **a**llocation
 - ◆ **calloc** : **c**ontiguous memory **a**llocation
 - ◆ **realloc** : **re**-**a**llocation
 - ◆ **free**

请善于利用缩写记忆函数名

➤ 例如

◆ sqrt: **s**quare **r**oot

◆ fabs : **f**loat **a**bsolute

◆ pow: **p**ower

➤ 但不包括这样的缩写：

◆ ABCDEFG: **A** **B**oy **C**an **D**o **E**verything **F**or
Girl

◆ GFEDCBA: **G**irls **F**orgot **E**verything **D**one &
Catch new **B**oy **A**gain

stdlib.h头文件

- 以上4个函数的声明在stdlib.h头文件中，在用到这些函数时应当用

`"#include <stdlib.h>"`

指令把stdlib.h头文件包含到程序文件中

内存的动态存储区

- 对于程序员，内存可视为三个部分：**堆**、**栈**和**静态区**
 - ◆ **堆（动态存储区，容量很大）**：由malloc系列函数分配的内存，生命周期由free函数决定。在没有free之前一直存在，直到程序结束。
 - ◆ **栈（容量很小）**：保存局部变量。栈上的内容只在函数的范围内存在；当函数运行结束，自动被销毁
 - ◆ **静态区**：保存全局变量和static变量（包括static全局和局部变量）。在整个程序的生命周期内都存在，由编译器在编译的时候分配。

什么是内存的动态分配

- 所谓动态内存分配就是指在程序执行的过程中**根据需要动态地分配或者回收**存储空间的分配内存的方法
- 由于动态分配不需要预先分配存储空间，而且分配的空间还可以根据程序的需要扩大或缩小，因此可以解决静态内存分配所带来的种种弊端

malloc函数

➤ 其函数原型为

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

- ◆ 其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**
- ◆ 函数的值是为所分配区域的第一个字节的地址，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开头位置
- ◆ 例如 `int *p=(int *)malloc(100);` 开辟100字节的临时分配域，函数返回值为其第1个字节的地址

malloc函数

- 注意返回的指针的基类型为**void**，即不指向任何类型的数据，只提供一个地址
- 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)
- 因此在使用指向这块内存的指针时，必须用 `if(NULL != p)` 语句来**验证**内存确实分配成功了

申请0字节内存会发生什么？

- 此时malloc仍然算作成功执行，函数**并不返回空指针（NULL）**
- malloc将**返回一个正常的内存地址，但是无法使用这块大小为0的内存**
- 这点需要尤其小心，因为if(NULL!=p)语句校验将不起作用

malloc函数应用实例

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

int main()
{
    int count, *array;
    array = (int *) malloc(10*sizeof(int));
    if (array == NULL)
    {
        printf("不能成功分配存储空间！");
        return 0;
    }
    for (count=0; count<10; count++)
        array[count] = count;
    for (count=0; count<10; count++)
        printf("%d ", array[count]);
    printf("\n");
    free(array);
    return 0;
}
```

```
0 1 2 3 4 5 6 7 8 9
Press any key to continue
```

作业 2017/12/08

➤ 按下列要求编写程序，提交手写源代码

1. 编写函数void find_two_largest(int *a, int m, int n, int *largest, int *second_largest); a指向长度大于n的数组。函数从数组a[m]到a[n]（不包括a[n]且 $n > m + 2$ ）之间中找出最大和第二大的元素，并把它们分别存到由largest和second_largest指向的变量中。要求使用指针运算而不是取下标来访问数组元素
2. 编写程序读一条消息，然后检查这条消息是否回文（消息中的字母从左往右和从右往左看是一样的）。如He lived as a devil, eh?是回文，Madam, I am Adam.则不是。可忽略所有不是字母的字符，要求使用指针访问数组元素。

思考题 2-1

➤ 下列代码的运行结果是什么？为什么？

```
#include <stdio.h>
int main()
{
    double a=3.0;
    float b=4.3;
    char c='a';
    a++;
    b++;
    c++;

    printf("%lf %lf %c\n", a, b, c);

    return 0;
}
```

思考题 2-2

➤ 下列代码的运行结果是什么？为什么？

```
#include <stdio.h>

int main()
{
    int a=0,b=3,c=5,d;

    d=++a || ++b&& c;

    printf("%d  %d  %d  %d \n",a,b,c,d);

    return 0;
}
```

思考题 2-3

➤ 下列代码的运行结果是什么？为什么？

```
#include <stdio.h>

int main()
{
    int a=0,b=3,c=5,d;

    d=a++ || ++b&&(c=6);

    printf("%d  %d  %d  %d \n",a,b,c,d);

    return 0;
}
```

思考题 2-4

➤ 下列代码的运行结果是什么？输入为PI=3.142✓

```
#include <stdio.h>
int main()
{
    char b, c;
    int a, d;

    scanf("PI=%d%c%c%d", &a, &b, &c, &d);
    printf("a=%d,", a);
    printf("b=%c,", b);
    printf("c=%d,", c);
    printf("d=%d,", d);
    return 0;
}
```