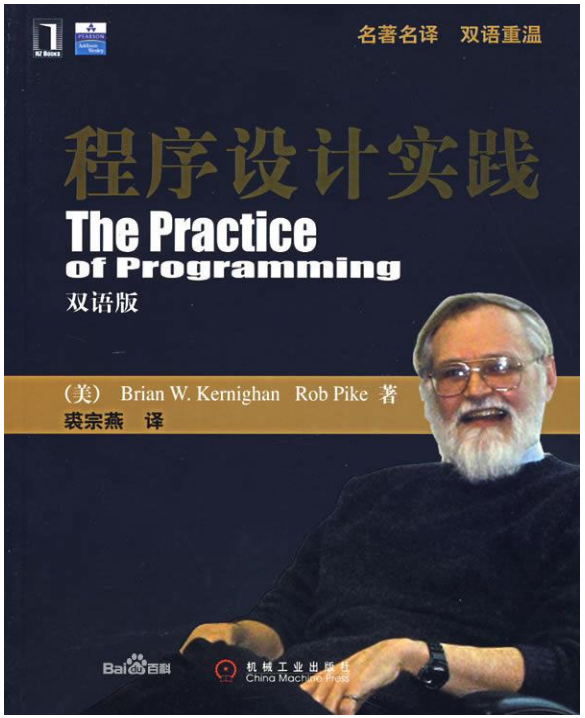
课前阅读

现实就是这样，总是存在许多程序错误，需要通过测试来发现，通过排错去纠正。

排错涉及到一种逆向推理，就像侦破一个杀人谜案。有些不可能的事情发生了，而仅有的信息就是它确实发生了！因此，我们必须从结果出发，逆向思考，去发现原因。



有时你“看到的代码”实际上是你自己的意愿，而不是你实际写出来的东西。离开它一小段时间能够松驰你的误解，帮助代码显出其本来面目。

——— 《程序设计实践》

关于数组、指针、函数的综合讨论

2014.11.05

（III-3200）

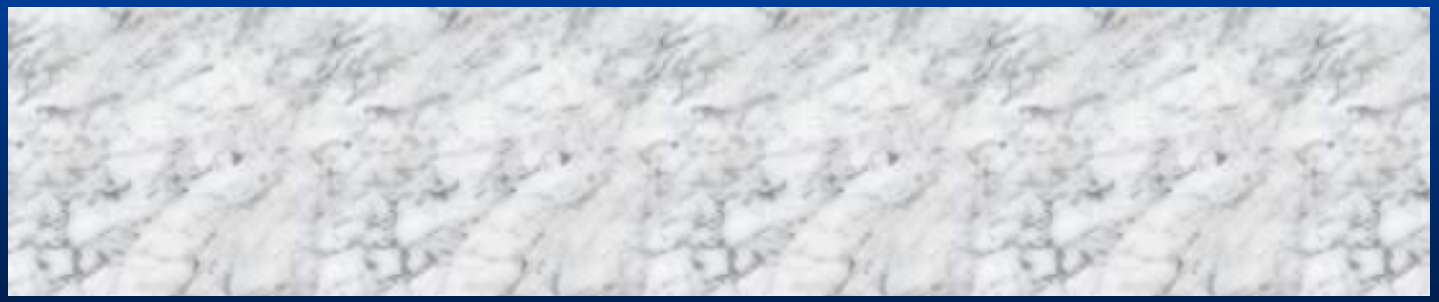
任务3：姓名排序

- 电视歌手大奖赛开赛报名时，由于人数较多，一些参赛信息需要及时录入计算机并用计算机进行管理。
- 其中一个很重要的工作就是：要按选手姓名（汉语拼音）排序后编号，以决定选手比赛的顺序。
- 请你编程实现对姓名拼音串按英文字典顺序排序的程序。
- 为测试程序，假定共有10名选手，选手姓名拼音最长不超过20个英文字符，且中间无空格。

```
#include <iostream>
using namespace std;


int main()  {
```

```
   return 0;
}
```

```
#include <iostream>
using namespace std;

int main()  {
   char namelist[10][20];
   for (int i=0; i<10; i++)  {
       cout << i << " singer name: ";
       cin >> namelist[i];
   }
```



```
   return 0;
}
```

```
#include <iostream>
using namespace std;

int main()  {
   char namelist[10][20];
   for (int i=0; i<10; i++)  {
       cout << i << " singer name: ";
       cin >> namelist[i];
   }

   for (int i=0; i<10; i++)
       cout << i << ' ' << namelist[i] << endl;
   return 0;
}
```

```
#include <iostream>
using namespace std;

int main()  {
   char namelist[10][20]; // 数组下标从0开始!
   for (int i=0; i<10; i++)  {
       cout << i << " singer name: ";
       cin >> namelist[i];
   }
   // 此处代码省略（见后）—用冒泡排序法对姓名进行排序
   for (int i=0; i<10; i++)
       cout << i << ' ' << namelist[i] << endl;
   return 0;
}
```

先看看整数数组的冒泡算法

Page 93

```
for (j=1; j<=5; j++)
{
      for (i=1; i<=6-j; i++)
      {
            if (a[i] < a[i+1])
            {
                  p = a[i];
                  a[i] = a[i+1];
                  a[i+1] = p;
            }
      }
}
```

如何修改成以0为首元素下标?

改成以0为首元素下标

```
for (j=0; j<5; j++)
{
      for (i=0; i<5-j; i++)
          // why not “i<6-j” ?
      {
            if (a[i] < a[i+1])
            {
                  p = a[i];
                  a[i] = a[i+1];
                  a[i+1] = p;
            }
      }
}
```

依样画葫芦

```
for (j=0; j<5; j++)
{
    for (i=0; i<5-j; i++)
        // why not "i<6-j" ?
    {
          if (a[i] < a[i+1])
          {
                  p = a[i];
                  a[i] = a[i+1];
                  a[i+1] = p;
          }
    }
}
```

```
for (j=0; j<9; j++) {
   for (i=0; i<9-j; i++) {
        if (namelist[i] < namelist[i+1]) {
                p = namelist[i];
                namelist[i] = namelist[i+1];
                namelist[i+1] = p;
        }
   }
}
```

```
for (int i=0; i<9; i++)  //轮(遍,趟)数=元素数目-1
{
    for (int j=0; j<9-i; j++)  //比较数=总轮数-当前轮次
    {
        if (                                    )
        {
                                                
                                                
                                                
        }
    }
}
```

```
for (int i=0; i<9; i++)
{
    for (int j=0; j<9-i; j++)
    {     // page 346, “strcmp()”
          if (strcmp(namelist[j], namelist[j+1]) > 0)
          {
                
          }
    }
}
```

字符串元素的特殊交换算法

```
for (int i=0; i<9; i++)
{
    for (int j=0; j<9-i; j++)
    {
        if (strcmp(namelist[j], namelist[j+1]) > 0)
        {     // page 345, "strcpy()"
            char tmp[20];
            strcpy(tmp, namelist[j]);
            strcpy(namelist[j], namelist[j+1]);
            strcpy(namelist[j+1], tmp);
        }
    }
}
```

如何按字典逆序来排序姓名呢?

如何按字典逆序来排序姓名

```
for (int i=0; i<9; i++)
{
    for (int j=0; j<9-i; j++)
    {
        if (strcmp(namelist[j], namelist[j+1]) < 0)
        {
            // 同前省略
        }
    }
}
```

比较关系符改为小于号，即按字典逆序排序姓名。

如何根据用户输入来决定按何种次序排序姓名呢?

排序准则的动态变化

- 可以用一个变量来记录用户的输入，根据输入选择来判定用户是想正逆排序姓名还是想逆序排序姓名。
- 该变量的类型可以是：
  - 布尔型 **bool**
  - 整数类型 **int**
  - 字符类型 **char**
  - 。 。 。

为什么这些类型都可以？

假设使用 **char UserInput;** 来记录用户选择：

U表示升序，D表示降序

排序准则的动态变化

```
if (UserInput == 'D') { // Down, 降序排列
      if (strcmp(namelist[j], namelist[j+1]) < 0)
      {
            // 交换相邻元素，同前省略
      }
 }
if (UserInput == 'U') { // Up, 升序排序
      if (strcmp(namelist[j], namelist[j+1]) > 0)
      {
            // 交换相邻元素，同前省略
      }
}
```

```
char UserInput;
cout << "Please input order
(U = Up, D = Down):";

cin >> UserInput;

// 确保输入合法
assert(UserInput == 'D' ||
       UserInput == 'U');
```

如果输入不合法，则程序会以崩溃的方式退出。

```
for (int i=0; i<9; i++) {
   for (int j=0; j<9-i; j++) {
      if (UserInput == 'D')       {
         if (strcmp(namelist[j], namelist[j+1]) < 0) {
            char tmp[20];
            strcpy(tmp, namelist[j]);
            strcpy(namelist[j], namelist[j+1]);
            strcpy(namelist[j+1], tmp);
         }
      }
      if (UserInput == 'U') {
         if (strcmp(namelist[j], namelist[j+1]) > 0) {
            char tmp[20];
            strcpy(tmp, namelist[j]);
            strcpy(namelist[j], namelist[j+1]);
            strcpy(namelist[j+1], tmp);
         }
      }
   }
}
```

程序代码还能更简捷一些吗?

程序代码的简化

```
if (UserInput == 'D')  {
   if (strcmp(namelist[j], namelist[j+1]) < 0) {
      char tmp[20];
      strcpy(tmp, namelist[j]);
      strcpy(namelist[j], namelist[j+1]);
      strcpy(namelist[j+1], tmp);
   }
}
```

接连两个if判断，可合成一个!

```
if (cond_1) {
   if (cond_2) {
      // ...
   }
}
```

```
if (cond_1 &&
    cond_2) {
      // ...
   }
}
```

```
for (int i=0; i<9; i++) {
   for (int j=0; j<9-i; j++) {
      if ( (UserInput == 'D') &&
           (strcmp(namelist[j], namelist[j+1]) < 0) ) {
         char tmp[20];
         strcpy(tmp, namelist[j]);
         strcpy(namelist[j], namelist[j+1]);
         strcpy(namelist[j+1], tmp);
       }

      if ( (UserInput == 'U') &&
           (strcmp(namelist[j], namelist[j+1]) > 0) ) {
         char tmp[20];
         strcpy(tmp, namelist[j]);
         strcpy(namelist[j], namelist[j+1]);
         strcpy(namelist[j+1], tmp);
      }
   }
}
```

程序代码还能做进一步的简化吗?

```
for (int i=0; i<9; i++) {
    for (int j=0; j<9-i; j++) {
        if ( (UserInput == 'D') &&
             (strcmp(namelist[j], namelist[j+1]) < 0) ) {
            char tmp[20];
            strcpy(tmp, namelist[j]);
            strcpy(namelist[j], namelist[j+1]);
            strcpy(namelist[j+1], tmp);
        }

        if ( (UserInput == 'U') &&
             (strcmp(namelist[j], namelist[j+1]) > 0) ) {
            char tmp[20];
            strcpy(tmp, namelist[j]);
            strcpy(namelist[j], namelist[j+1]);
            strcpy(namelist[j+1], tmp);
        }
    }
}
```

怎么简化?

```
for (int i=0; i<9; i++) {
    for (int j=0; j<9-i; j++) {
        if ( (UserInput == 'D') &&
             (strcmp(namelist[j], namelist[j+1]) < 0) ) {
            Swap(namelist[j], namelist[j+1]);
        }

        if ( (UserInput == 'U') &&
             (strcmp(namelist[j], namelist[j+1]) > 0) ) {
            Swap(namelist[j], namelist[j+1]);
        }
    }
}
```

怎么定义Swap(...)函数呢?

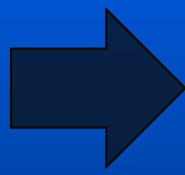
Swap(…)函数的定义

- 注意到：调用该函数时，实参是 namelist[j]和 namelist[j+1]，它们分别对应二维数组变量 namelist 的
  - 第j“行” 和
  - 第j+1“行”
- 二维数组的“行变量”对应的是一个数组：

➔

```
void Swap(char first[ ], char second[ ]);
```

```
if ( (UserInput == 'D') &&
     (strcmp(namelist[j], namelist[j+1]) < 0) ) {
  char tmp[20];
  strcpy(tmp, namelist[j]);
  strcpy(namelist[j], namelist[j+1]);
  strcpy(namelist[j+1], tmp);
}
```



```
void Swap(char first[], char second[])
{
      char tmp[20];
      strcpy(tmp, first);
      strcpy(first, second);
      strcpy(second, tmp);
}
```

```
for (int i=0; i<9; i++)
{
   for (int j=0; j<9-i; j++)
   {
       if ( (UserInput == 'D') &&
            (strcmp(namelist[j], namelist[j+1]) < 0) )
          Swap(namelist[j], namelist[j+1]);

       if ( (UserInput == 'U') &&
            (strcmp(namelist[j], namelist[j+1]) > 0) )
          Swap(namelist[j], namelist[j+1]);
   }
}
```

依样画葫芦

Page 93

```
for (j=0; j<5; j++)
{
        for (i=0; i<5-j; i++)
         {
                if (a[i] < a[i+1])
                 {
                        p = a[i];
                        a[i] = a[i+1];
                        a[i+1] = p;
                }
        }
}
```

```
for (j=0; j<5; j++)
{
        for (i=0; i<5-j; i++)
         {
                if (a[i] < a[i+1])
                {
                        Swap(a[i], a[i+1]);
                }
        }
}
```

Swap(…)函数的定义

依样画葫芦

- 注意到：调用该函数时，程序中的实参是a[i]和 a[i+1]，它们分别对应一维数组变量a的第i和第i+1个元素
- 一维整数数组的元素，是一个整数，其类型是整数类型
- 故有

```
void Swap(int first, int second);
```

依样画葫芦

```
#include <iostream>
using namespace std;

void Swap(int first, int second)
{
   int p;
   p = first;
   first = second;
   second = p;
}

int main()
{
   int i, j, p, a[6];
   memset(a, 0, sizeof(a));
   for (i=0; i<6; i++) {
       cout << "请输入待排序的数a[" << i << "]=";
       cin >> a[i];
   }
```

依样画葫芦

```
    for (j=0; j<5; j++)
    {
        for (i=0; i<5-j; i++)
        {
            if (a[i] < a[i+1])
                Swap(a[i], a[i+1]);
        }
    }
    for (i=0; i<6; i++)
        cout << a[i] << endl;

    return 0;
}
```

可惜啊！程序结果并不正确！

BUG（问题）在哪里?

```
void Swap(int first, int second)
{
        int p;
        p = first;
        first = second;
        second = p;
}
```

变量first和second是函数Swap的形式参数，该函数被调用时，实参的值是被赋值给形参的，即相当于是复制了一份。所以，在函数Swap中交换的是“复制品”之间内容的交换，并不影响实参的内容（值）！

验证一下

```
#include <iostream>
using namespace std;

void Swap(int first, int second)
{
   int p;
   p = first;
   first = second;
   second = p;
}

int main()
{
   int a = 3, b = 4;
   cout << "Before calling Swap(), a = " << a
        << ", b = " << b << endl;
   Swap(a, b);
   cout << "After calling  Swap(), a = " << a
        << ", b = " << b << endl;
   return 0;
}
```

```
d:\20101102>g++ swap_test.cpp

d:\20101102>a
Before calling Swap(), a = 3, b = 4
After calling Swap(), a = 3, b = 4
```

```
void Swap(int first, int second)
{
  cout << "&first = " << &first
       << ", &second = " << &second << endl;
  // 同前省略
}

int main()
{
  int a = 3, b = 4;
  cout << "&a = " << &a
       << ", &b = " << &b << endl;

  // 同前省略
}
```

进一步验证

比较实参与形参的内存单元

```
d:\20101102>g++ swap_test_2.cpp

d:\20101102>a
&a = 0x22ff44, &b = 0x22ff40
Before calling Swap(), a = 3, b = 4
&first = 0x22ff20, &second = 0x22ff24
After calling  Swap(), a = 3, b = 4
```

如何改正这个BUG（问题）

```
void Swap(int* first, int* second)
{
  int p;
  p = *first;
  *first = *second;
  *second = p;
}
```

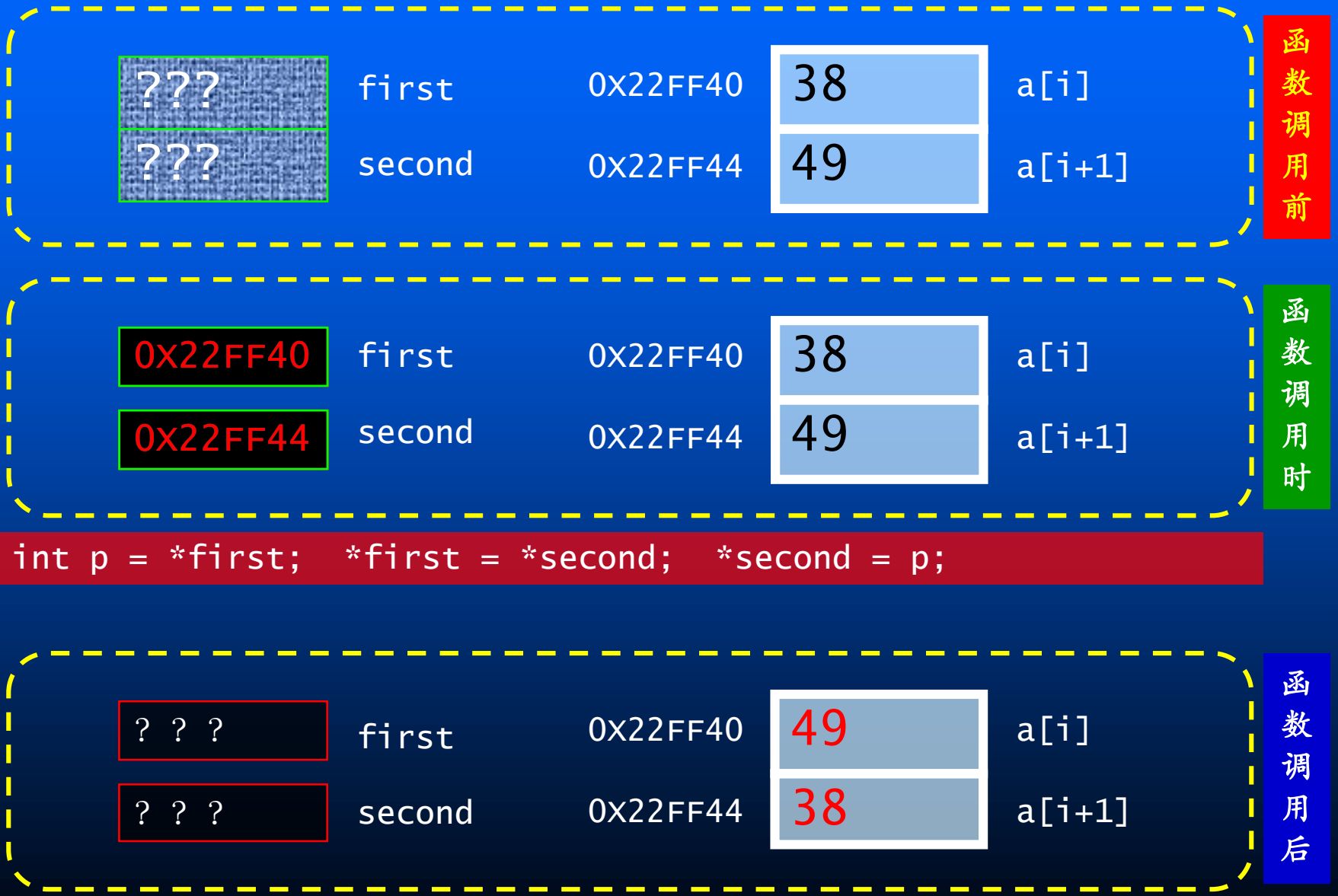
```
// in main()
for (j=0; j<5; j++)
  for (i=0; i<5-j; i++)
     if (a[i] < a[i+1])
          Swap(&(a[i]), &(a[i+1]));
```

为什么这样BUG就被去掉了?

为什么这样BUG就被去掉了?

为什么这样BUG就被去掉了?

为什么这样BUG就被去掉了?



回到 姓名排序（二维数组） 问题上来

```
for (int i=0; i<9; i++)
{
   for (int j=0; j<9-i; j++)
   {
       if ( (UserInput == 'D') &&
            (strcmp(namelist[j], namelist[j+1]) < 0) )
          Swap(namelist[j], namelist[j+1]);

       if ( (UserInput == 'U') &&
            (strcmp(namelist[j], namelist[j+1]) > 0) )
          Swap(namelist[j], namelist[j+1]);
   }
}
```

程序代码还能做进一步的简化吗?

strcmp(…)的返回值是什么？

```
#include <iostream>
using namespace std;

int main()
{
   cout << strcmp("AAA", "BBB") << endl;
   cout << strcmp("DDD", "BBB") << endl;
   return 0;
}
```

程序输出为：

−1

1

```
for (int i=0; i<9; i++)
{
   for (int j=0; j<9-i; j++)
   {
       if ( (UserInput == 'D') &&
            (strcmp(namelist[j], namelist[j+1]) == -1) )
          Swap(namelist[j], namelist[j+1]);

       if ( (UserInput == 'U') &&
            (strcmp(namelist[j], namelist[j+1]) == 1) )
          Swap(namelist[j], namelist[j+1]);
   }
}
```

< 0

> 0

int UserInput ?

抽取两种排序准则的共性

```
// ...
int UserInput;
cin >> UserInput;
assert(UserInput == -1 || UserInput == 1);

for (int i=0; i<9; i++)
{
   for (int j=0; j<9-i; j++)
   {
       if (strcmp(namelist[j], namelist[j+1])
            == UserInput) // 两种排序准则合并在一起处理
           Swap(namelist[j], namelist[j+1]);
   }}
// ...
```

思考题：如果用户仍然希望使用U和D表示升序和降序，程序应如何修改？

下面，任务又改变啦！

领导和观众都强烈要求按歌手类型分开比赛，以保证公平竞争。

即，会有多个不同人数的姓名数组需要排序处理。程序该如何修改呢?

如通俗歌手有20人，美声歌手有15人。

难道是这样么？不过…有点…

```
for (int i=0; i<19; i++) // 通俗歌手排序
  for (int j=0; j<19-i; j++)
     if (strcmp(NL_1[j], NL_1[j+1])
          == UserInput)
        Swap(NL_1[j], NL_1[j+1]);

for (int i=0; i<14; i++) // 美声歌手排序
  for (int j=0; j<14-i; j++)
     if (strcmp(NL_2[j], NL_2[j+1])
          == UserInput)
        Swap(NL_2[j], NL_2[j+1]);
```

冒泡泡的函数上场啦

```
Void bubble(char NL[][20], int num,
                int UserInput)
{
    for (int i=0; i<num-1; i++)
        for (int j=0; j<num-1-i; j++)
            if (strcmp(NL[j], NL[j+1])
                == UserInput)
                Swap(NL[j], NL[j+1]);
}
// ...
```

```
bubble(NL_1, 20, UserInput);
bubble(NL_2, 15, UserINput);
```

依样画葫芦

```
void bubble(int a[], int num)
{
   for (int j=0; j<num-1; j++)
   {
       for (int i=0; i<num-1-j; i++)
       {
             if (a[i] < a[i+1])
                 Swap(&(a[i]), &(a[i+1]));
       }
   }
}
```

另一种常见的定义方式

```
void bubble(int* a, int num)
{
   for (int j=0; j<num-1; j++)
   {
       for (int i=0; i<num-1-j; i++)
       {
             if (a[i] < a[i+1])
                  Swap(&(a[i]), &(a[i+1]));
       }
   }
}
```

编译器认为上述两种写法是相同的

如果排序准则也要求动态设定

依样画葫芦

```
void bubble(int* a, int num, int order)
{
   for (int j=0; j<num-1; j++)
   {
       for (int i=0; i<num-1-j; i++)
       {
           if (a[i] < a[i+1])
               Swap(&(a[i]), &(a[i+1]));
       }
   }
}
```

上述函数应该如何修改其实现呢?

回顾一下姓名排序代码

```
Void bubble(char NL[][20], int num,
                int UserInput)
{
    for (int i=0; i<num-1; i++)
        for (int j=0; j<num-1-i; j++)
        {
            if (strcmp(NL[j], NL[j+1])
                == UserInput)
                Swap(NL[j], NL[j+1]);
        }
}
```

增加一个 int_cmp(…) 函数?

依样画葫芦

```
void bubble(int* a, int num, int order)
{
        for (int j=0; j<num-1; j++)
        {
                for (int i=0; i<num-1-j; i++)
                {
                        if (int_cmp(a[i], a[i+1])
                          == order)
                            Swap(&(a[i]), &(a[i+1]));
                }
        }
}
```

int_cmp(int, int)的实现

```
int int_cmp(int a, int b)
{
    if (a < b)
        return -1;
    if (a > b)
        return 1;
    return 0;
} // 完全参照 strcmp 的功能实现
```

为什么不这样……实现呢?

```
bool int_cmp(int a, int b, int order)
{
    if (a < b)
        return order == -1;
    if (a > b)
        return order == 1;
    return false;
} // v1.0
```

为什么不这样……实现呢?

```
bool int_cmp(int a, int b, int order)
{
    if ( (a < b) && (order == -1) )
       return true;
    if ( (a > b) && (order ==  1) )
       return true;
    return false;
} // v2.0
```

为什么不这样……实现呢?

```
bool int_cmp(int a, int b, int order)
{
    if ( ((a < b)&&(order == -1) ||
         ((a > b)&&(order ==  1)) )
       return true;

    return false;
} // v3.0
```

为什么不这样……实现呢?

```
bool int_cmp(int a, int b, int order)
{
    return ((a < b)&&(order == -1)) ||
           ((a > b)&&(order ==  1)) ;
} // V4.0
```

```
void bubble(int* a, int num, int order)
{
  for (int j=0; j<num-1; j++)
  {
     for (int i=0; i<num-1-j; i++)
     {
        if (int_cmp(a[i], a[i+1], order))
           Swap(&(a[i]), &(a[i+1]));
     }
  }
}
```

你还想到怎么修改上面的代码?

（未完持续……）

```
int main()
{
   int a1[4] = {1,2,3}, a2[5] = {1,2,1,2,9};
   int *p = a1, *q;  // 注意指针q的定义方式

   for (int i=0; i<4; i++) cout << p[i] << '=' << *(p+i) << ' ';
   cout << endl;

   for (p=a2+4; p>=a2; p--) cout << *p << ' '; // 指针用作循环变量
   cout << endl;

   q = a2;
   for (; q<a2+5; q++) cout << *q << ' ';
   cout << endl;

   p = a1; // 数组变量可以直接赋值给指针变量
   cout << "p = " << p << " a1 = " << a1 << endl;

   p = a2;
   cout << "      p = " << p         << ", a2    = " << a2 << endl;
   cout << "    p+1 = " << p + 1     << ", a2+1  = " << a2 + 1 << endl;
   cout << " *(p+2) = " << *(p + 2)  << ", p[2]  = " << p[2] << endl;
   cout << "*(a2+2) = " << *(a2 + 2) << ", a2[2] = " << a2[2] << endl;

   return 0;
}// pointer-array-1.cpp
```

参考输出

```
1=1  2=2  3=3  0=0
9  2  1  2  1
1  2  1  2  9
p  =  0x22ff30  a1  =  0x22ff30
        p  =  0x22ff10,  a2      =  0x22ff10
      p+1  =  0x22ff14,  a2+1    =  0x22ff14
 *(p+2)  =  1,  p[2]    =  1
*(a2+2)  =  1,  a2[2]  =  1
```

请在此页总结要点

```
#include <iostream>
using namespace std;

int main() {
   int a1[4] = {1,2,3}, a2[5] = {1,2,1,2,9};
   int *p = a1, *q;

   // 指针变量与数组变量的“大小”是不同的
   cout << "sizeof(a1) = " << sizeof(a1) << endl;
   cout << "sizeof(a2) = " << sizeof(a2) << endl;
   cout << "sizeof(p)  = " << sizeof(p) << endl;
   cout << "sizeof(q)  = " << sizeof(q) << endl;

#ifdef SHOW_ERROR
   a1 = q; // 数组变量不能接受赋值
   a2 = p;
#endif

   return 0;
} // pointer-array-2.cpp
```

参考输出

```
d:\20101109>g++ pointer-array-2.cpp

d:\20101109>a
sizeof(a1) = 16
sizeof(a2) = 20
sizeof(p)  = 4
sizeof(q)  = 4

d:\20101109>g++ pointer-array-2.cpp -DSHOW_ERROR
pointer-array-2.cpp: In function `int main()':
pointer-array-2.cpp:15: error: incompatible types in assignment of `int*' to `in
t[4]'
pointer-array-2.cpp:16: error: incompatible types in assignment of `int*' to `in
t[5]'
```

请在此页总结要点

```
#include <iostream>
using namespace std;

void test_1(int a[], int len) // 数组变量作函数参数
{
  cout << "test_1():  a = " << a << ", len = "
       << len << endl << "\t\t";
  for (int i=0; i<len; i++)
      cout << a[i] << '=' << *(a+i) << ' ';
  cout << endl;
}

void test_2(int* a, int len) // 指针变量作函数参数
{
  cout << "test_2():  a = " << a << ", len = "
       << len << endl << "\t\t";
  for (int i=0; i<len; i++)
      cout << a[i] << '=' << *(a+i) << ' ';
  cout << endl;
}
```

```
int main()
{
   int a1[4] = {1,2,3};
   int a2[5] = {1,2,1,2,9};

   cout << "main():    a1 = " << a1
        << ", a2 = " << a2 << endl;

   test_1(a1, 4);
   test_1(a2, 5);

   test_2(a1, 4);
   test_2(a2, 5);

   int* p = a2;
   test_1(p, 2);
   test_2(p, 3);

   return 0;
} // pointer-array-3.cpp
```

参考输出

```
d:\20101109>g++ pointer-array-3.cpp

d:\20101109>a
main():     a1 = 0x22ff30, a2 = 0x22ff10
test_1():   a = 0x22ff30, len = 4
                1=1 2=2 3=3 0=0
test_1():   a = 0x22ff10, len = 5
                1=1 2=2 1=1 2=2 9=9
test_2():   a = 0x22ff30, len = 4
                1=1 2=2 3=3 0=0
test_2():   a = 0x22ff10, len = 5
                1=1 2=2 1=1 2=2 9=9
test_1():   a = 0x22ff10, len = 2
                1=1 2=2
test_2():   a = 0x22ff10, len = 3
                1=1 2=2 1=1
```

请在此页总结要点

```
#include <iostream>
using namespace std;

void test_3(char a[], int len)
{
   cout << "test_3():  a = " << a << ", len = "
        << len << endl << "\t\t";

   for (int i=0; i<len; i++)
       cout << a[i] << '=' << *(a+i) << ' ';
   cout << endl;
}

void test_4(char* a, int len)
{
   cout << "test_4():  a = " << a << ", len = "
        << len << endl << "\t\t";

   for (int i=0; i<len; i++)
       cout << a[i] << '=' << *(a+i) << ' ';
   cout << endl;
}
```

```
int main() {
   char* name1       = "XXX"; // 这是初始化，不是赋值!
   char  name2[]     = "tsinghua";
   char  name3[10]   = "computer";
   name1 = "J0X-???"; // ← OK!  改变字符指针变量的值
   // name2 = "YYY"; name3 = "ZZZ"; ← ERROR!
   //  (数组形式的)字符串赋值，必须使用 strcpy 函数!
   test_3(name1, 4);
   test_4(name1, 4);

   test_3(name2, 8);
   test_4(name2, 8);

   test_3(name3, 8);
   test_4(name3, 8);

   test_3(name3, 10);
   test_4(name3, 10);

   test_3(name3, 3);
   test_4(name3, 3);

   return 0;
} // pointer-array-4.cpp
```

注意代码中关于 name1 = “J0X-???”;的说明。“J0X-???”是一个常量字符串，它也对应一个地址值，所以语句 name1 = “J0X-???”;的意思是:将存放“J0X-???”的内存单元首地址赋给变量 name1。因为 name1 是指针类型，不是数组类型，所以这样做是正确的。

而 name2 = “YYY”;语句等号左边是数组类型的变量，等号右边是常量字符串指针（地址值），类型不符合，不能赋值!

参考输出

```
test_3():  a = J0X???, len = 4
                J=J 0=0 X=X ?=?
test_4():  a = J0X???, len = 4
                J=J 0=0 X=X ?=?
test_3():  a = tsinghua, len = 8
                t=t s=s i=i n=n g=g h=h u=u a=a
test_4():  a = tsinghua, len = 8
                t=t s=s i=i n=n g=g h=h u=u a=a
test_3():  a = computer, len = 8
                c=c o=o m=m p=p u=u t=t e=e r=r
test_4():  a = computer, len = 8
                c=c o=o m=m p=p u=u t=t e=e r=r
test_3():  a = computer, len = 10
                c=c o=o m=m p=p u=u t=t e=e r=r  =    =
test_4():  a = computer, len = 10
                c=c o=o m=m p=p u=u t=t e=e r=r  =    =
test_3():  a = computer, len = 3
                c=c o=o m=m
test_4():  a = computer, len = 3
                c=c o=o m=m
```

请在此页总结要点

```
void test_1(int a[], int len)
{
    // 同前省略
}

void test_5(int a[], int len) // 数组参数a中的元素内容被更新了!
{
    cout << "test_5():  a = " << a << ", len = " << len << endl;
    for (int i=0; i<len; i++)
    {
        cout << "\t\ta[" << i << "] = " << a[i] << " ==> ";
        a[i] = a[i] + i * 2;    // 在这里把数组元素更新了
        cout << a[i] << endl;
    }
    cout << endl;
}

void test_6(int a[], int len, int b[])
{
    cout << "test_6():  a = " << a << ", len = " << len
         << " b = " << b << endl << "\t\t";
    for (int i=0; i<len; i++)
        b[i] = a[i] + i * 2;  // 运算结果赋值给了参数数组 b
    cout << endl;
}
```

```
int main()
{
  int a1[4] = {1,2,3};
  int a2[5] = {1,2,1,2,9};
  test_5(a1, 4);
  test_1(a1, 4);

  test_6(a1, 4, a2); // 数组a2的内容发生了改变
  test_1(a2, 4);
  test_1(a2, 5);

  return 0;
} // pointer-array-5.cpp
```

参考输出

```
d:\20101109>g++ pointer-array-5.cpp

d:\20101109>a
test_5():  a = 0x22ff30, len = 4
                a[0] = 1 ==> 1
                a[1] = 2 ==> 4
                a[2] = 3 ==> 7
                a[3] = 0 ==> 6

test_1():  a = 0x22ff30, len = 4
                1=1 4=4 7=7 6=6
test_6():  a = 0x22ff30, len = 4 b = 0x22ff10

test_1():  a = 0x22ff10, len = 4
                1=1 6=6 11=11 12=12
test_1():  a = 0x22ff10, len = 5
                1=1 6=6 11=11 12=12 9=9
```

请在此页总结要点

```
void test_7(int a[3][4])  // 两个维度都是固定的 二维数组
{
   int len1 = 3, len2 = 4;
   for (int i=0; i<len1; i++)
   {
        for (int j=0; j<len2; j++)
        {
               cout << a[i][j] << ' ';
        }
        cout << endl;
   }
}

void test_8(int a[][4], int len)  // 第二维固定的 二维数组
{
   int len1 = len, len2 = 4;
   for (int i=0; i<len1; i++)
   {
        for (int j=0; j<len2; j++)
        {
               cout << a[i][j] << ' ';
        }
        cout << endl;
   }
}
```

参数a实际上是一维的指针数组!

```
void test_9(int* a[4], int len) {
    int len1 = len, len2 = 4;
    for (int i=0; i<len1; i++) {
        for (int j=0; j<len2; j++)
            cout << a[i][j] << ' ';
        cout << endl;
    }
}

#ifdef SHOW_ERROR
void test_10(int a[][], int len1, int len2) {
    for (int i=0; i<len1; i++) {
        for (int j=0; j<len2; j++)
            cout << a[i][j] << ' ';
        cout << endl;
    }
}
#endif

void test_11(int** a, int len1, int len2) {
    for (int i=0; i<len1; i++) {
        for (int j=0; j<len2; j++)
            cout << a[i][j] << ' ';
        cout << endl;
    }
}
```

```
int main() {
    int b[3][4] = {{1,1,1,81}, {2,2,2,72}, {3,3,3,63}};
    int c[4][3] = {{1,2,3}, {4,5,6}, {7,8,9}, {9,8,7}};

    cout << "call test_7() to output matrix b:" << endl;
    test_7(b);      // test_7(int a[3][4])

    cout << "call test_8() to output matrix b:" << endl;
    test_8(b, 3);  // test_8(int a[][4], int len)

#ifdef SHOW_ERROR
    test_7(c);
    test_8(c, 4);

    test_9(b, 3);
    test_10(b, 3, 4);     // test_10(int a[][], int, int)
    test_11(b, 3, 4);     // test_11(int**, int, int)

    test_11((int*)(b), 3, 4);    // 强制转换也不行!
    test_11((int*)(c), 4, 3);
#endif

    return 0;
} // pointer-array-6.cpp
```

参考输出

```
call test_7() to output matrix b:
1 1 1 81
2 2 2 72
3 3 3 63
call test_8() to output matrix b:
1 1 1 81
2 2 2 72
3 3 3 63
```

参考输出

```
d:\20101109>g++ pointer-array-6.cpp -DSHOW_ERROR
pointer-array-6.cpp:45: error: declaration of `a' as multidimensional array must
 have bounds for all dimensions except the first
pointer-array-6.cpp: In function `void test_10(int, int)':
pointer-array-6.cpp:50: error: `a' undeclared (first use this function)
pointer-array-6.cpp:50: error: (Each undeclared identifier is reported only once
 for each function it appears in.)
pointer-array-6.cpp: In function `int main()':
pointer-array-6.cpp:81: error: cannot convert `int (*)[3]' to `int (*)[4]' for a
rgument `1' to `void test_7(int (*)[4])'
pointer-array-6.cpp:82: error: cannot convert `int (*)[3]' to `int (*)[4]' for a
rgument `1' to `void test_8(int (*)[4], int)'
pointer-array-6.cpp:84: error: cannot convert `int (*)[4]' to `int**' for argume
nt `1' to `void test_9(int**, int)'
pointer-array-6.cpp:85: error: invalid conversion from `int (*)[4]' to `int'
pointer-array-6.cpp:45: error: too many arguments to function `void test_10(int,
 int)'
pointer-array-6.cpp:85: error: at this point in file
pointer-array-6.cpp:86: error: cannot convert `int (*)[4]' to `int**' for argume
nt `1' to `void test_11(int**, int, int)'
pointer-array-6.cpp:88: error: cannot convert `int*' to `int**' for argument `1'
 to `void test_11(int**, int, int)'
pointer-array-6.cpp:89: error: cannot convert `int*' to `int**' for argument `1'
 to `void test_11(int**, int, int)'
```

请在此页总结要点

这两个函数到底应该如何被使用呢？

```
void test_9(int* a[4], int len)
{
   int len1 = len, len2 = 4;
   for (int i=0; i<len1; i++)
   {
       for (int j=0; j<len2; j++)
              cout << a[i][j] << ' ';
       cout << endl;
   }
}

void test_11(int** a, int len1, int len2)
{
   for (int i=0; i<len1; i++)
   {
       for (int j=0; j<len2; j++)
              cout << a[i][j] << ' ';
       cout << endl;
   }
}
```

```
void test_9(int* a[4], int len) {
   int len1 = 4, len2 = len;
   for (int i=0; i<len1; i++) {
       for (int j=0; j<len2; j++) cout << a[i][j] << ' ';
       cout << endl;
   }
}

int main() {
   int* d[4];

   for (int i=0; i<4; i++) d[i] = new int[3];

   for (int i=0; i<4; i++)
       for (int j=0; j<3; j++)
           d[i][j] = 10 * i + j;
   test_9(d, 3);

   for (int i=0; i<4; i++) delete[] d[i];

   return 0;
} // pointer-array-7.cpp
```

参考输出

```
d:\20101109>g++ pointer-array-7.cpp

d:\20101109>a
0 1 2
10 11 12
20 21 22
30 31 32
```

一维数组的动态内存分配

- 先来看看数组中元素在内存中的位置

```
int a[30];
```


```
a     == &(a[0])
a+8   == &(a[8])
a+9   == &(a[8]) + 4
a+10 == &(a[9]) + 4 == &(a[9]) + sizeof(int)
...
sizeof(a) == sizeof(int) * 30
&(a[30]) == a + sizeof(int) * 30
```

测试代码

```
#include <iostream>
using namespace std;

int main()
{
   int a[30];
   for (int i=0; i<=30; i++)
       cout << i << ": " << &(a[i])
            << " - " << a+i << endl;

   return 0;
} // pointer-array-9.cpp
```

结果表明：一维数组中的元素是依次存放的；下标相邻的单元，内存地址也相邻；第一个元素地址值最小。

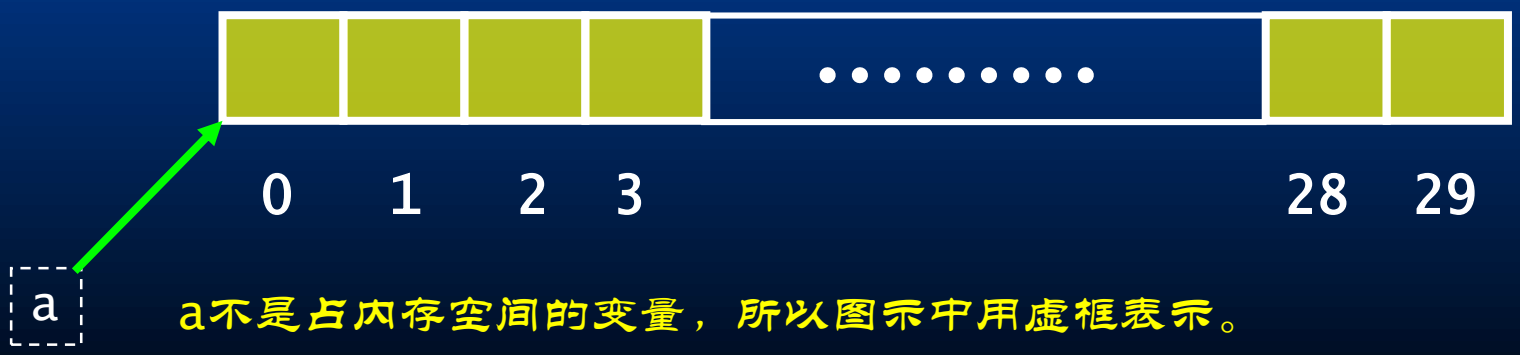
参考输出

```
0: 0x22fec0 - 0x22fec0
1: 0x22fec4 - 0x22fec4
2: 0x22fec8 - 0x22fec8
3: 0x22fecc - 0x22fecc
4: 0x22fed0 - 0x22fed0
5: 0x22fed4 - 0x22fed4
6: 0x22fed8 - 0x22fed8
7: 0x22fedc - 0x22fedc
8: 0x22fee0 - 0x22fee0
9: 0x22fee4 - 0x22fee4
10: 0x22fee8 - 0x22fee8
11: 0x22feec - 0x22feec
......
26: 0x22ff28 - 0x22ff28
27: 0x22ff2c - 0x22ff2c
28: 0x22ff30 - 0x22ff30
29: 0x22ff34 - 0x22ff34
30: 0x22ff38 - 0x22ff38
```

一维数组的动态内存分配

- 注意到，数组a占用的内存空间（即元素数量），是在源程序中直接给定的（如３０），在程序运行期间不能改变：既不能增长、也不能缩减。通常称此类数组为“静态数组”。

```
int a[30]; //  所以，a实际上是一个常量指针，不能给a赋值。
```



一维数组的动态内存分配

- 若想要在程序执行时再确定“数组”的大小，必须使用“动态数组”。这需要通过 new 运算符来实现：

```
int* a = new int[30];
```

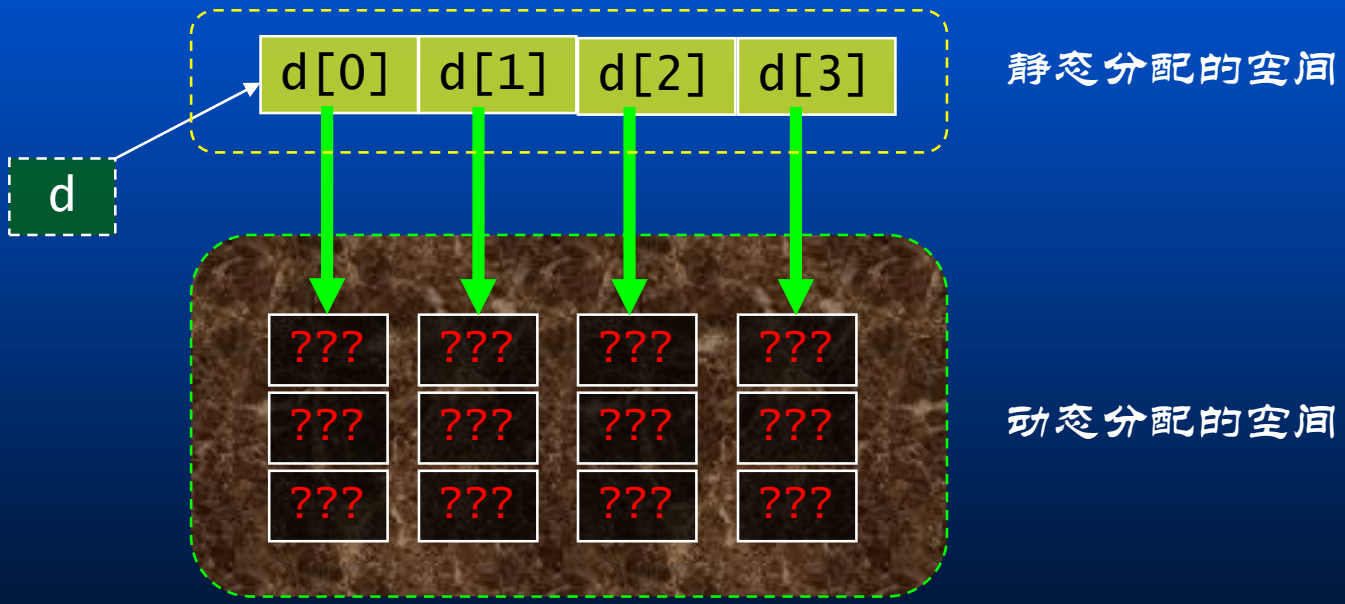
- 使用该运算符的语法规则如下，其中NUM既可以是变量，也可以是常量！

```
TYPE* var = new TYPE[NUM];
```

a是占内存空间的指针变量，所以图示中用实线框表示。

```
int main() {
   int* d[4];

   for (int i=0; i<4; i++) d[i] = new int[3];
```

数组d的元素类型为 int*，它含有4个元素。
也可称为：指针数组——因为各元素为指针类型。



d不是占内存空间的变量，所以图示中用虚框表示。

一维数组的动态内存分配

- 凡是由 new 运算符“生成”的内存空间，在不再需要（使用）它时，应该（或者说，必须）使用 delete 运算符将内存“释放”掉。

- 对于动态生成的一维数组，参照下面的格式（delete[]，这里的[]表示要释放数组空间）释放分配的内存单元：

```
delete[] a;     // 若已有 int* a = new int[30];
```

```
#include <iostream>
using namespace std;

void test_11(int** a, int len1, int len2)
{
  for (int i=0; i<len1; i++)
  {
     for (int j=0; j<len2; j++)
          cout << a[i][j] << ' ';
     cout << endl;
  }
}
```

```
int main() {
  int** e;
  e = new int*[4];
  for (int i=0; i<4; i++) e[i] = new int[3];

  for (int i=0; i<4; i++)
     for (int j=0; j<3; j++)
          e[i][j] = 10 * i + j;
  test_11(e, 4, 3);

  for (int i=0; i<4; i++) delete[] e[i];
  delete[] e;
  return 0;
} // pointer-array-8.cpp
```

参考输出

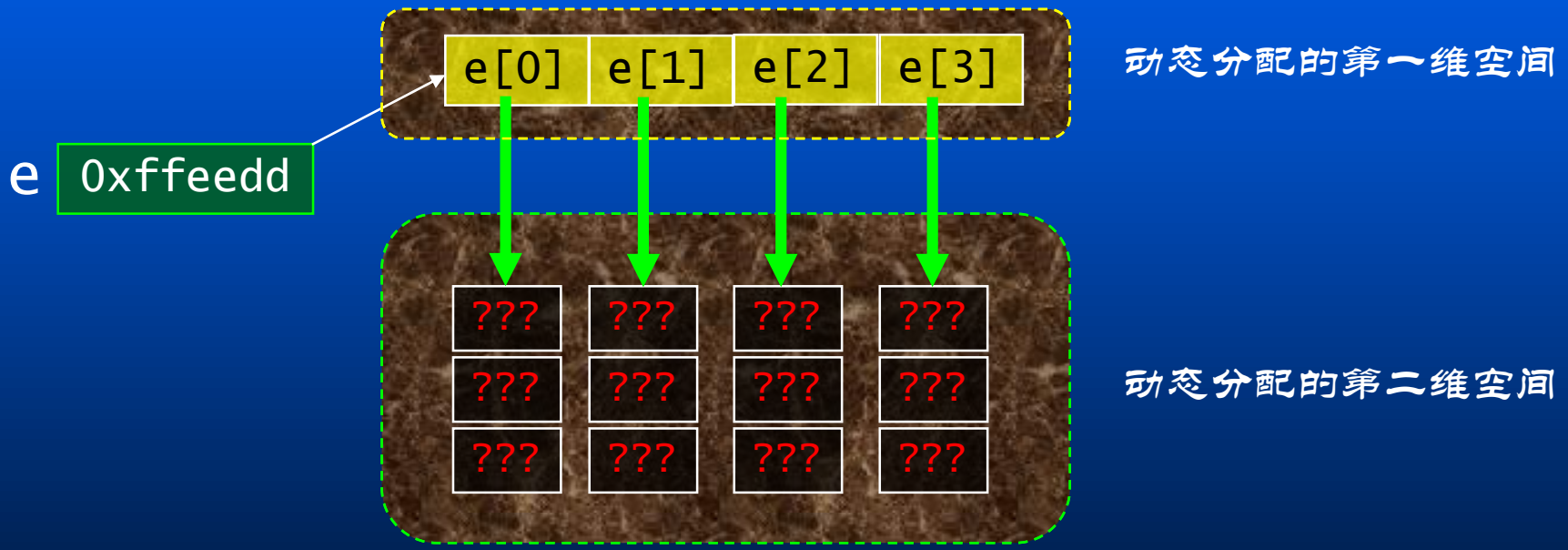
```
d:\20101109>g++ pointer-array-8.cpp

d:\20101109>a
0 1 2
10 11 12
20 21 22
30 31 32
```

二维数组的动态内存分配

- 二维（甚至更多维）数组（设为m*n）的动态分配过程，是分两步来完成的：
  - 根据需要，分配第一维的空间。实际上是m个指针组成的动态数组。
  - 分别为m个指针分配一维动态数组空间，每个都包含n个元素。这一步与前面所讲的“一维数组的动态内存分配”是一样的。
- 示例：

```
int** e;
e = new int*[4];
for (int i=0; i<4; i++) e[i] = new int[3];
```

指针e指向连续4个内存单元的首地址，
这些内存单元中存放都是内存地址，类型为 int*。
所以，也可称e为：指向指针的指针。
通常，可以将之直观理解为特殊的“二维数组”变量。

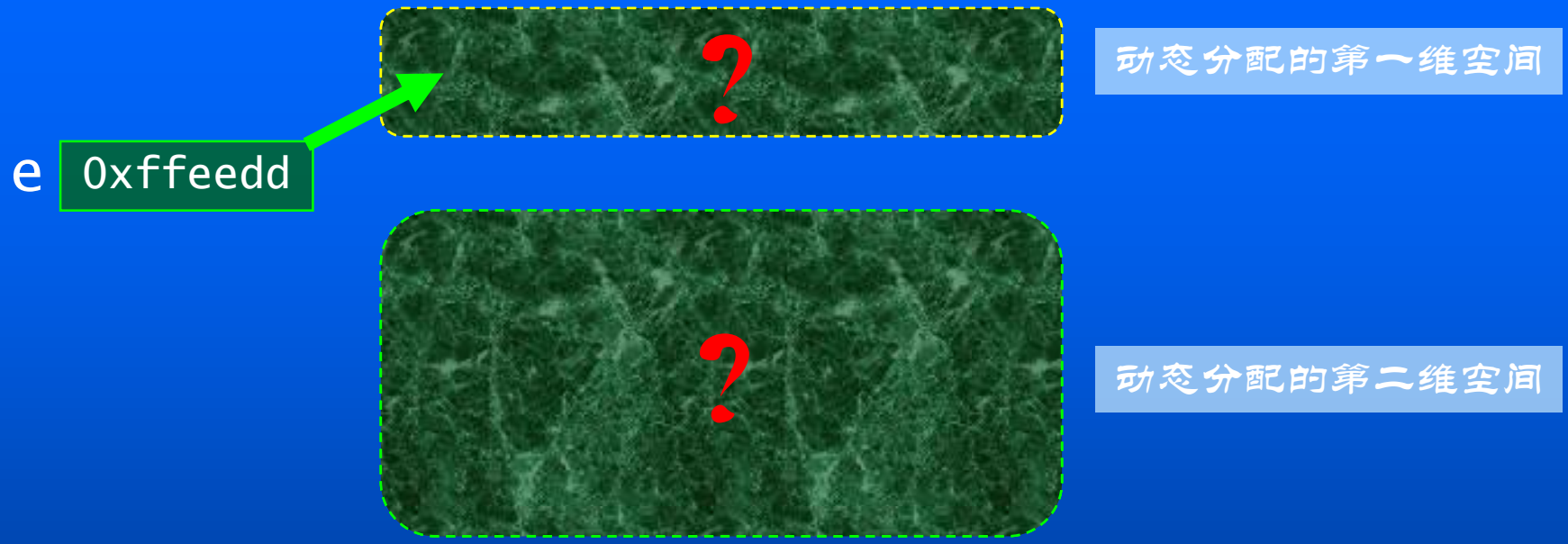


e是占内存空间的指针变量，所以图示中用实线框表示。

二维数组的动态内存释放

- 二维（甚至更多维）数组内存的动态释放过程，实际上是分配过程的严格逆过程：
  - 先通过逐一枚举第一维各元素，将该维中的各元素（即第二维的一维动态数组）指向的内存单元释放掉。
  - 释放第一维占用的内存空间。
- 示例：

```
for (int i=0; i<4; i++) delete[] e[i];
delete[] e;
```



e是占内存空间的指针变量，所以图示中用实线框表示。

上图中有错误吗？

以一维动态数组为例，说明指针被delete前后的变化和注意事项

```
#include <iostream>
using namespace std;

int main()
{
   int *a = new int[4];
   for (int i=0; i<4; i++)
       a[i] = i * 10 + i % 7;

   for (int i=0; i<4; i++)
       cout << a[i] << ' ';
   cout << endl;

   cout << "before delete[], a = " << a << endl;
   delete[] a;  // 分配给a的内存全部释放了
   cout << "after  delete[], a = " << a << endl;

   for (int i=0; i<4; i++)
       cout << a[i] << ' ';
   cout << endl;

   return 0;
} // pointer-array-B.cpp
```

悬挂指针（野指针）问题

```
d:\20101109>g++ pointer-array-B.cpp

d:\20101109>a
0 11 22 33
before delete[], a = 0x3a1728
after  delete[], a = 0x3a1728
3810056 3801284 22 33
```

虽然a指向的空间被释放了，但变量a的内容并没有改变！
若通过变量a再次访问那些内存空间，结果将不可预知。
—— 切记！

解释原因，需用到大三的《操作系统》课程的知识

猜猜下面程序段的输出

PART-1

```
#include <iostream>
using namespace std;

int main()
{
  double array[4][3]; // 4 * 3 matrix

  int num = 1;
  for (int i=0; i<4; i++)     // row
      for (int j=0; j<3; j++) // col
            array[i][j] = num++;

  for (int i=0; i<4; i++)     // row
      for (int j=0; j<3; j++) // col
            cout << &(array[i][j]) << ':'
                 << array[i][j] << endl;
  cout << "================================" << endl;
```

你猜对了吗？

```
0x22fee0:1
0x22fee8:2
0x22fef0:3
0x22fef8:4
0x22ff00:5
0x22ff08:6
0x22ff10:7
0x22ff18:8
0x22ff20:9
0x22ff28:10
0x22ff30:11
0x22ff38:12
================================
```

猜猜下面程序段的输出

PART-2

```
double *ptr = (double*) array;

// 一层循环，将二维数组当作一维动态数组来访问
for (int i=0; i<12; i++)
    cout << ptr+i << ':' << *(ptr+i)
         << " --> "
         << ptr[i] << endl;

cout << "================================" << endl;
```

你猜对了吗？

```
0x22fee0:1 --> 1
0x22fee8:2 --> 2
0x22fef0:3 --> 3
0x22fef8:4 --> 4
0x22ff00:5 --> 5
0x22ff08:6 --> 6
0x22ff10:7 --> 7
0x22ff18:8 --> 8
0x22ff20:9 --> 9
0x22ff28:10 --> 10
0x22ff30:11 --> 11
0x22ff38:12 --> 12
==================================
```

猜猜下面程序段的输出

```
    // 两层循环，将一维动态数组变量当作二维数组来访问
    for (int i=0; i<4; i++)
    {
        for (int j=0; j<3; j++)
            cout << *(ptr + i*3 + j) << ' ';
        cout << endl;
    }

    return 0;
} // pointer-array-C.cpp
```

PART-3

你猜对了吗？

```
1  2  3
4  5  6
7  8  9
10  11  12
```

读完上面的代码，你有什么要总结的吗？☆

请在此页总结要点

结　束