



C++语言程序设计

第六章 数组 指针与字符串

信息工程学院 王红平

Email: cugwhp@qq.com



本章主要内容

- 数组
- 指针
- 动态存储分配
- 深复制与浅复制
- 字符串



数组的概念

数 数组是具有一定顺序关系的若干相
组 同类型变量的集合体，组成数组的变量
 称为该数组的元素。

数组属于构造类型。



一维数组的声明与引用

数

- 一维数组的声明

类型说明符 数组名[常量表达式];



组

数组名的构成方法与一般变量名相同。

例如: `int a[10];`

表示 `a` 为整型数组, 有10个元素: `a[0]...a[9]`

- 引用

必须先声明, 后使用。

只能逐个引用数组元素, 而不能一次引用整个数组

例如: `a[0]=a[5]+a[7]-a[2*3]`



例6.1 一维数组的声明与引用

数
组

```
#include <iostream>
using namespace std;
void main()
{ int A[10],B[10];
  int i;
  for(i=0;i<10;i++)
  {
    A[i]=i*2-1;
    B[10-i-1]=A[i];
  }
```

```
for(i=0;i<10;i++)
{
  cout<<"A["<<i
    <<"]="<<A[i];
  cout<<" B["<<i
    <<"]="
    <<B[i]<<endl;
}
}
```



一维数组的存储顺序

数

数组元素在内存中顺次存放，它们的地址是连续的。

组

例如：具有10个元素的数组 a，在内存中的存放次序如下：



数组名字是数组首元素的内存地址。

数组名是一个常量，不能被赋值。

一维数组的初始化

数
组

可以在编译阶段使数组得到初值：

- 在声明数组时对数组元素赋以初值。

例如： `int a[10]={0,1,2,3,4,5,6,7,8,9};`

- 可以只给一部分元素赋初值。

例如： `int a[10]={0,1,2,3,4};`


- 在对全部数组元素赋初值时，可以不指定数组长度。

例如： `int a[]={1,2,3,4,5}`



例：用数组来处理求Fibonacci数列问题

```
#include<iostream>
using namespace std;
void main()
{ int i;
  static int f[20]={1,1}; //初始化第0、1个数
  for(i=2;i<20;i++) f[i]=f[i-2]+f[i-1]; //求第2~19个数
  for(i=0;i<20;i++) //输出，每行5个数//
  { if(i%5==0) cout<<endl;
    cout.width(12); //设置输出宽度为12
    cout<<f[i];
  }
}
```



例：用数组来处理求Fibonacci数列问题

运行结果：

1	1	2	3	5
8	13	21	34	55
89	144	233	377	610
987	1597	2584	4181	6765

二维数组的声明及引用

数 数据类型 标识符[常量表达式1][常量表达式2];

例:

组 `int a[5][3];`

表示a为整型二维数组，其中第一维有5个下标（0~4），第二维有3个下标（0~2），数组的元素个数为15，可以用于存放5行3列的整型数据表格。



二维数组的声明及引用

数
组

- 二维数组的声明

类型说明符 数组名[常量表达式][常量表达式]

例如: `float a[3][4];`

可以理解为:

a $\left[\begin{array}{l} a[0] \text{——} a_{00} \ a_{01} \ a_{02} \ a_{03} \\ a[1] \text{——} a_{10} \ a_{11} \ a_{12} \ a_{13} \\ a[2] \text{——} a_{20} \ a_{21} \ a_{22} \ a_{23} \end{array} \right.$

- 存储顺序

按行存放, 上例中数组a的存储顺序为:

$a_{00} \ a_{01} \ a_{02} \ a_{03}$ $a_{10} \ a_{11} \ a_{12} \ a_{13}$ $a_{20} \ a_{21} \ a_{22} \ a_{23}$

- 引用

例如: `b[1][2]=a[2][3]/2`

下标不要越界

二维数组的初始化

数
组

- 将所有数据写在一个{}内，按顺序赋值

例如: `int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};`

- 分行给二维数组赋初值

例如: `int a[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};`

- 可以对部分元素赋初值

例如: `int a[3][4]={{1},{0,6},{0,0,11}};`



数组作为函数参数

数 组

- 数组元素作实参，与单个变量一样。
- 数组名作参数，形、实参数都应是数组名，类型要一样，传送的是数组首地址。对形参数组的改变会直接影响到实参数组。



例6-2 使用数组名作为函数参数

数
组

- 主函数中初始化一个矩阵并将每个元素都输出，然后调用子函数，分别计算每一行的元素之和，将和直接存放在每行的第一个元素中，返回主函数之后输出各行元素的和。



```
#include <iostream>
using namespace std;
void RowSum(int A[][4], int nrow)
{ int sum;
  for (int i = 0; i < nrow; i++)
  {
    sum = 0;
    for(int j = 0; j < 4; j++)
      sum += A[i][j];
    cout << "Sum of row " << i
          << " is " << sum << endl;
    A[i][0]=sum;
  }
}
```

```

void main(void)
{ int Table[3][4] =
  {{1,2,3,4},{2,3,4,5},{3,4,5,6}};

  for (int i = 0; i < 3; i++)
  { for (int j = 0; j < 4; j++)
    cout << Table[i][j] << "  ";
    cout << endl;
  }
  RowSum(Table,3);
  for (int i = 0; i < 3; i++)
    cout << Table[i][0]<<"  ";
}

```


运行结果:

1 2 3 4

2 3 4 5

3 4 5 6

Sum of row 0 is 10

Sum of row 1 is 14

Sum of row 2 is 18

10 14 18

随堂作业

- 1. 以数组作为参数的函数，分别统计一维数组最大值、最小值、平均值。
- 2. 实现上题中，以二维数组作为参数的函数。



对象数组

数

- 声明:

类名 数组名[元素个数];

组

- 访问方法:

通过下标访问

数组名[下标].成员名



对象数组初始化

数 组

- 数组中每一个元素对象被创建时，系统都会调用类构造函数初始化该对象。
- 通过初始化列表赋值。

例：

```
Point A[2]={Point(1,2),Point(3,4)};
```

- 如果没有为数组元素指定显式初始值，数组元素便使用默认值初始化（调用默认构造函数）。



数组元素所属类的构造函数

- 数
组
- 不声明构造函数，则采用默认构造函数。
 - 各元素对象的初值要求为相同的值时，可以声明具有默认形参值的构造函数。
 - 各元素对象的初值要求为不同的值时，需要声明带形参的构造函数。
 - 当数组中每一个对象被删除时，系统都要调用一次析构函数。



例6-3 对象数组应用举例

数

组

```
//Point.h
#if !defined(_POINT_H)
#define _POINT_H
class Point
{ public:
    Point();
    Point(int xx,int yy);
    ~Point();
    void Move(int x,int y);
    int GetX() {return X;}
    int GetY() {return Y;}
private:
    int X,Y;
};
#endif
```



```
//6-2.cpp
#include<iostream>
using namespace std;
#include "Point.h"
Point::Point()
{ X=Y=0;
  cout<<"Default Constructor called."<<endl;
}
Point::Point(int xx,int yy)
{ X=xx;
  Y=yy;
  cout<< "Constructor called."<<endl;
}
Point::~~Point()
{ cout<<"Destructor called."<<endl; }
void Point::Move(int x,int y)
{ X=x; Y=y; }
```

```
#include<iostream>
#include "Point.h"
using namespace std;
int main()
{
    cout<<"Entering main..."<<endl;
    Point A[2];
    for(int i=0;i<2;i++)
        A[i].Move(i+10,i+20);
    cout<<"Exiting main..."<<endl;
    return 0;
}
```


运行结果:

Entering main...

Default Constructor called.

Default Constructor called.

Exiting main...

Destructor called.

Destructor called.

指针

- 内存空间的访问方式

- 通过变量名访问
- 通过地址访问

- 地址运算符：&

例：

```
int var;
```

则&var 表示变量var在内存中的起始地址

指针变量的概念

指

针

概念

指针：内存地址，用于
间接访问内存单元

指针变量：
用于存放地址的变量

声明

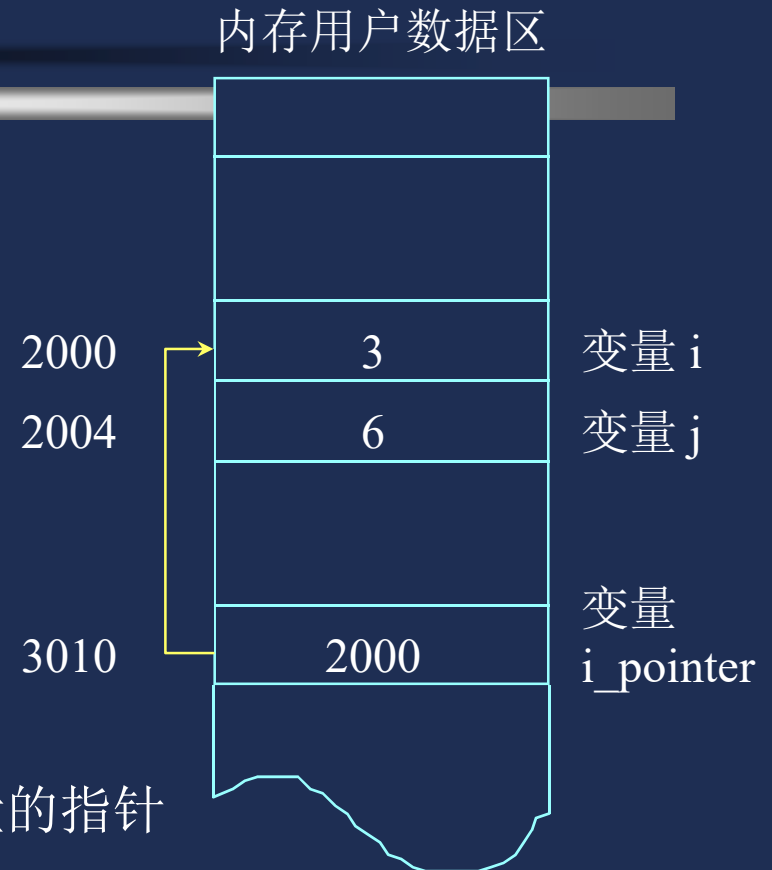
```
例： int i;  
      int *i_pointer=&i;
```

↑
指向整型变量的指针

引用

例1: i=3;

例2: *i_pointer=3;



i_pointer



*i_pointer



i

2000

指针变量的初始化

指
针

- 语法形式

存储类型 数据类型 *指针名 = 初始地址;


例: `int *pa=&a;`



例6-5 指针的声明、赋值与使用

指 `#include<iostream>`
`using namespace std;`
`void main()`

针 `{ int *i_pointer; //声明int型指针i_pointer`
`int i; //声明int型数i`
`i_pointer=&i; //取i的地址赋给i_pointer`
`i=10; //int型数赋初值`
`cout<<"Output int i="<<i<<endl; //输出int型数的值`
`cout<<"Output int pointer i="<<*i_pointer<<endl;`
`//输出int型指针所指地址的内容`
`}`



程序运行的结果是：

Output int i=10

Output int pointer i=10

指向常量的指针

指
针

- 不能通过指针来改变所指对象的值，但指针本身可以改变，可以指向另外的对象。

- 例

```
int a;  
const int *p1=&a;  
int b;  
p1=&b;  
*p1=1;//编译出错
```



指针类型的常量

- 若声明指针常量，则指针本身的值不能被改变。例：

```
int a;
```

```
int* const p2=&a;
```

```
int b;
```

```
p2=&b; //错误，指针常量值不能改变
```



例6-6 void类型指针的使用

指
针

```
void vobject; //错，不能声明void类型的变量
void *pv;     //对，可以声明void类型的指针
int *pint; int i;
void main()   //void类型的函数没有返回值
{
    pv = &i;   //void类型指针指向整型变量
    // void指针赋值给int指针需要类型强制转换：
    pint = (int *)pv;
}
```



指针变量的赋值运算

指针

指针名=地址

- “地址”中存放的数据类型与指针类型必须相符。
- 向指针变量赋的值必须是地址常量或变量，不能是普通整数。但可以赋值为整数0，表示空指针。
- 指针的类型是它所指向变量的类型，而不是指针本身数据值的类型，任何一个指针本身的数据值都是 **unsigned long int** 型。
- 允许声明指向 **void** 类型的指针。该指针可以被赋予任何类型对象的地址。

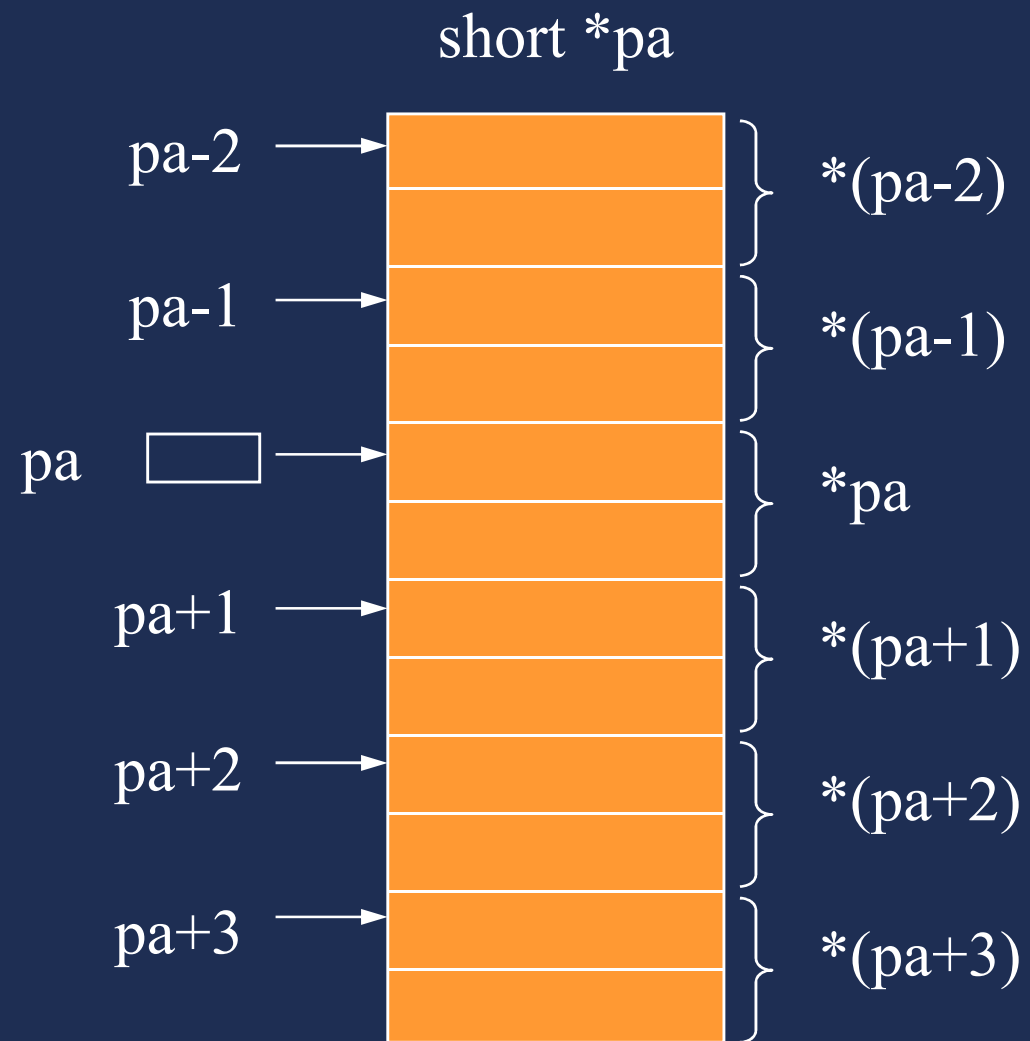
例: **void *general;**

指针变量的算术运算

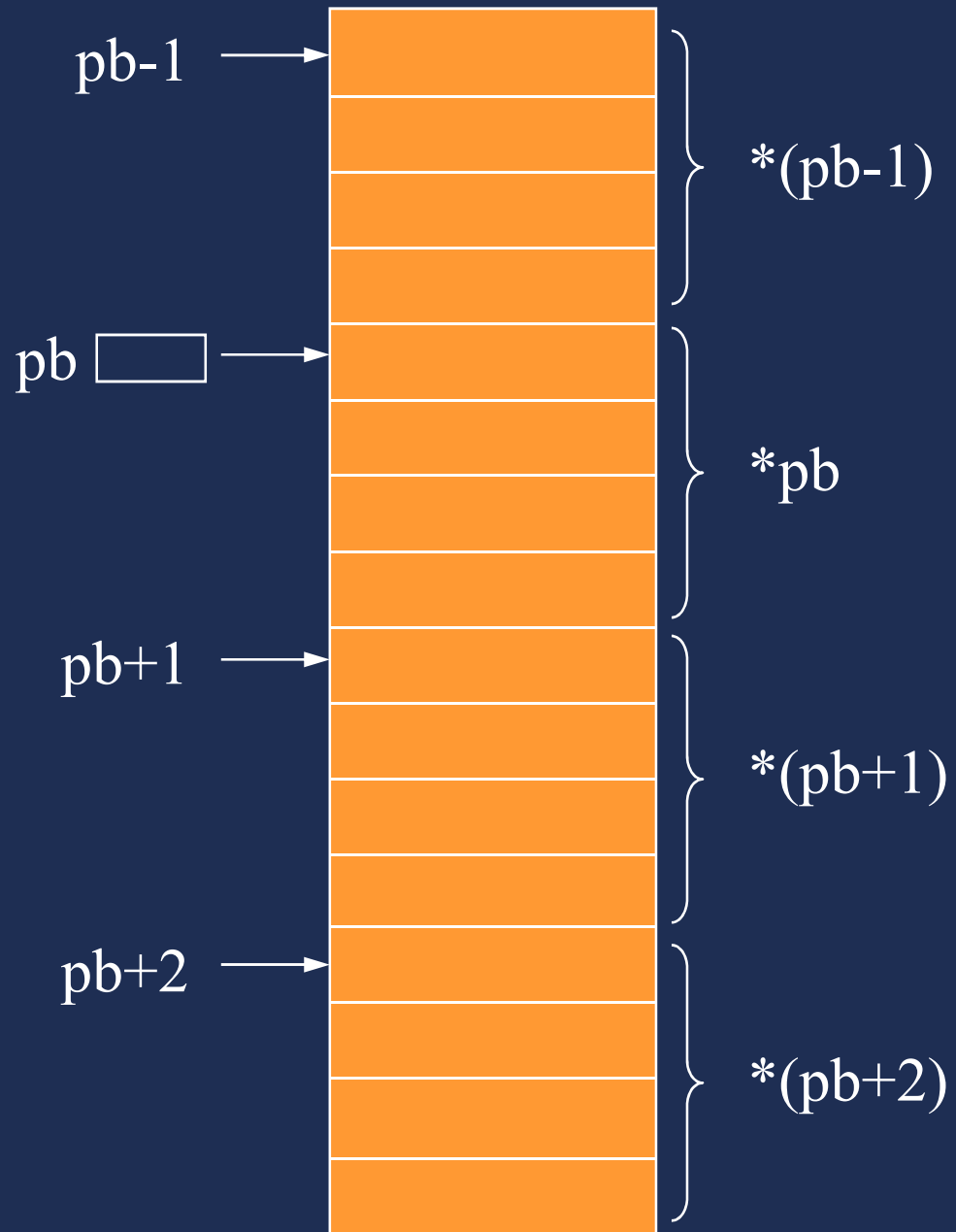
指针

- 指针与整数的加减运算
 - 指针 p 加上或减去 n ，其意义是指针当前指向位置的前方或后方第 n 个数据的地址。
 - 这种运算的结果值取决于指针指向的数据类型。
- 指针加一，减一运算
 - 指向下一个或前一个数据。
 - 例如： $y=*px++$ 相当于 $y=*(px++)$
(*和++优先级相同，自右向左运算)





`long *pb`



指针变量的关系运算

指

针

- 关系运算

- 指向相同类型数据的指针之间可以进行各种关系运算。
- 指向不同数据类型的指针，以及指针与一般整数变量之间的关系运算是无意义的。
- 指针可以和零之间进行等于或不等于的关系运算。例如： $p==0$ 或 $p!=0$

- 赋值运算

- 向指针变量赋的值必须是地址常量或变量，不能是普通整数。但可以赋值为整数0，表示空指针。

指向数组元素的指针

指针 ● 声明与赋值

例: `int a[10], *pa;`
`pa=&a[0];` 或 `pa=a;`

针

● 通过指针引用数组元素

经过上述声明及赋值后:

- `*pa`就是`a[0]`, `*(pa+1)`就是`a[1]`, ... ,
`*(pa+i)`就是`a[i]`.
- `a[i]`, `*(pa+i)`, `*(a+i)`, `pa[i]`都是等效的。
- 不能写 `a++`, 因为`a`是数组首地址是常量。

例6-7

指 设有一个int型数组a，有10个元素。用三种方法输出各元素：

- 针**
- 使用数组名和下标
 - 使用数组名和指针运算
 - 使用指针变量



使用数组名和下标

```
main()
{
    int a[10];
    int i;
    for(i=0; i<10; i++)
        cin>>a[i];
    cout<<endl;
    for(i=0; i<10; i++)
        cout<<a[i];
}
```



使用数组名指针运算

```
main()
{
    int a[10];
    int i;
    for(i=0; i<10; i++)
        cin>>a[i];
    cout<<endl;
    for(i=0; i<10; i++)
        cout<<*(a+i);
}
```



使用指针变量

```
main()
{
    int a[10];
    int *p,i;
    for(i=0; i<10; i++)
        cin>>a[i];
    cout<<endl;
    for(p=a; p<(a+10); p++)
        cout<<*p;
}
```



this指针

- 指** ● 隐含于每一个类的成员函数中的特殊指针。
- 针** ● 明确地指出了成员函数当前所操作的数据所属的对象。
- 当通过一个对象调用成员函数时，系统先将该对象的地址赋给**this**指针，然后调用成员函数，成员函数对对象的数据成员进行操作时，就隐含使用了**this**指针。★ ★

this指针

指 例如: **Point**类的构造函数体中的语句:

`X=xx;`

针 `Y=yy;`

相当于:

`this->X=xx;`

`this->Y=yy;`



指针数组

指
针

- 数组的元素是指针型

- 例: `Point *pa[2];`




由`pa[0]`,`pa[1]`两个指针组成



例6-8 利用指针数组存放单位矩阵

指 `#include <iostream>`
`using namespace std;`
`void main()`
针 `{ int line1[]={1,0,0}; //声明数组, 矩阵的第一行`
`int line2[]={0,1,0}; //声明数组, 矩阵的第二行`
`int line3[]={0,0,1}; //声明数组, 矩阵的第三行`
`int *p_line[3]; //声明整型指针数组`
`p_line[0]=line1; //初始化指针数组元素`
`p_line[1]=line2;`
`p_line[2]=line3;`



//输出单位矩阵

```
cout<<"Matrix test:"<<endl;
for(int i=0;i<3;i++) //对指针数组元素循环
{
    for(int j=0;j<3;j++) //对矩阵每一行循环
    { cout<<p_line[i][j]<<" "; }
    cout<<endl;
}
}
```

输出结果为:

Matrix test:

1, 0, 0

0, 1, 0

0, 0, 1



指向常量的指针做形参

指

```
#include<iostream>
using namespace std;
const int N=6;
```

针

```
void print(const int *p,int n);
void main()
{ int array[N];
  for(int i=0;i<N;i++)
    cin>>array[i];
  print(array,N);
}
```



```
void print(const int *p, int n)
{
    cout<<"{"<<*p;
    for(int i=1;i<n;i++)
        cout<<". "<<*(p+i);
    cout<<"}"<<endl;
}
```



指针型函数

指针与函数

当函数的返回值是地址时，该函数就是指针形函数。

声明形式

存储类型 数据类型 *函数名()



指向函数的指针

指针与函数

- 声明形式

存储类型 数据类型 (*函数指针名)();

- 含义:

- 数据指针指向数据存储区，而函数指针指向的是程序代码存储区。



例6-11 函数指针

指针与函数

```
#include <iostream>
using namespace std;
void print_stuff(float data_to_ignore);
void print_message(float list_this_data);
void print_float(float data_to_print);
void (*function_pointer)(float);
void main()
{
    float pi = (float)3.14159;
    float two_pi = (float)2.0 * pi;
```



```
print_stuff(pi);  
function_pointer = print_stuff;  
function_pointer(pi);  
function_pointer = print_message;  
function_pointer(two_pi);  
function_pointer(13.0);  
function_pointer = print_float;  
function_pointer(pi);  
print_float(pi);  
}
```



```
void print_stuff(float data_to_ignore)
{ cout<<"This is the print stuff function.\n"; }
```

```
void print_message(float list_this_data)
{ cout<<"The data to be listed is "
  <<list_this_data<<endl;
}
```

```
void print_float(float data_to_print)
{ cout<<"The data to be printed is "
  <<data_to_print<<endl;
}
```



运行结果:

This is the print stuff function.

This is the print stuff function.

The data to be listed is 6.283180

The data to be listed is 13.000000

The data to be printed is 3.141590

The data to be printed is 3.141590



对象指针的一般概念

指

针

- 声明形式

类名 *对象指针名;

- 例

Point A(5,10);

Point *ptr;

ptr=&A;

- 通过指针访问对象成员

对象指针名->成员名

ptr->getx() 相当于 (*ptr).getx();



对象指针应用举例

指
针

```
int main()  
{  
    Point A(5,10);  
    Point *ptr;  
    ptr=&A;  
    int x;  
    x=ptr->GetX();  
    cout<<x<<endl;  
    return 0;  
}
```



曾经出现过的错误例子

指 `class Fred;` //前向引用声明

针 `class Barney {`
 `Fred x;` //错误: 类Fred的声明尚不完善
 `};`
 `class Fred {`
 `Barney y;`
 `};`



正确的程序

指 `class Fred; //前向引用声明`
针 `class Barney {`
 `Fred *x;`
 `};`
 `class Fred {`
 `Barney y;`
 `};`



指向类的非静态成员指针

- 指针**
- 通过指向成员的指针只能访问公有成员
 - 声明指向成员的指针
 - 声明指向公有数据成员的指针
类型说明符 类名::*指针名;
 - 声明指向公有函数成员的指针
类型说明符 (类名::*指针名)(参数表);



指向类的非静态成员指针

指针

- 指向数据成员的指针

- 说明指针应该指向哪个成员

指针名=&类名::数据成员名;

- 通过对象名（或对象指针）与成员指针结合来访问数据成员

对象名.* 类成员指针名

或:

对象指针名—>*类成员指针名



指向类的非静态成员的指针

指针 ● 指向函数成员的指针

— 初始化

指针名=类名::函数成员名;

— 通过对象名（或对象指针）与成员指针结合来访问函数成员

(对象名.* 类成员指针名)(参数表)

或:

(对象指针名—>*类成员指针名)(参数表)



指向类的非静态成员的指针

指
针

例6-13 访问对象的公有成员函数的不同方式

```
void main()           //主函数
{ Point A(4,5);       //声明对象A
  Point *p1=&A;        //声明对象指针并初始化
  //声明成员函数指针并初始化
  int (Point::*p_GetX)()=Point::GetX;
  //（1）使用成员函数指针访问成员函数
  cout<<(A.*p_GetX)()<<endl;
  //（2）使用对象指针访问成员函数
  cout<<(p1->GetX)()<<endl;
  //（3）使用对象名访问成员函数
  cout<<A.GetX()<<endl;
}
```



指向类的静态成员的指针

- 指针
- 对类的静态成员的访问不依赖于对象
 - 可以用普通的指针来指向和访问静态成员



例6-14通过指针访问类的静态数据成员

指

针

```
#include <iostream>
using namespace std;
class Point    //Point类声明
{public:      //外部接口
    Point(int xx=0, int yy=0) {X=xx;Y=yy;countP++;} //构造函数
    Point(Point &p);    //拷贝构造函数
    int GetX() {return X;}
    int GetY() {return Y;}
    static int countP;  //静态数据成员引用性说明
private:             //私有数据成员
    int X,Y;
};
Point::Point(Point &p)
{ X=p.X; Y=p.Y; countP++; }

int Point::countP=0;  //静态数据成员定义性说明
```

```
void main() //主函数
{ //声明一个int型指针，指向类的静态成员
  int *count=&Point::countP;
  Point A(4,5); //声明对象A
  cout<<"Point A,"<<A.GetX()<<","<<A.GetY();
  //直接通过指针访问静态数据成员
  cout<<" Object id="<<*count<<endl;
  Point B(A); //声明对象B
  cout<<"Point B,"<<B.GetX()
    <<","<<B.GetY();
  //直接通过指针访问静态数据成员
  cout<<" Object id="<<*count<<endl;
}
```



动态申请内存操作符 new

动态
内存
存储
分配

new 类型名T（初值列表）

功能：在程序执行期间，申请用于存放
T类型对象的内存空间，并依初值列表
赋以初值。

结果值：成功：T类型的指针，指向新
分配的内存。失败：0（NULL）



释放内存操作符delete

动态
存储
分配

delete 指针P

功能：释放指针P所指向的内存。P必须是new操作的返回值。



例6-16 动态创建对象举例

动态
存储
分配

```
#include<iostream>
using namespace std;
class Point
{ public:
    Point()
    { X=Y=0; cout<<"Default Constructor called.\n";}
    Point(int xx,int yy)
    { X=xx; Y=yy; cout<<"Constructor called.\n"; }
    ~Point() { cout<<"Destructor called.\n"; }
    int GetX() {return X;}
    int GetY() {return Y;}
    void Move(int x,int y) { X=x; Y=y; }
private:
    int X,Y;
};
```



```
int main()
{  cout<<"Step One:"<<endl;
   Point *Ptr1=new Point;
   delete Ptr1;
   cout<<"Step Two:"<<endl;
   Ptr1=new Point(1,2);
   delete Ptr1;
   return 0;
}
```

运行结果:

Step One:

Default Constructor called.

Destructor called.

Step Two:

Constructor called.

Destructor called.



动态创建数组操作符 new

动态
存储
分配

new 类型名T[N]

功能：如果内存申请成功，**new**运算返回一个指向新分配内存首地址的指针，是一个T类型的数组。

结果值：成功：T类型的指针，指向新分配的数组。失败：0（NULL）



释放数组操作符delete

动态
存储
分配

`delete []`指针P

功能：释放指针P所指向的数组。P必须是new操作的返回值。



运行结果如下:

Please enter the number of points:2

Default Constructor called.

Default Constructor called.

Deleting...

Destructor called.

Destructor called.



例6-17 动态创建对象数组举例

动
态
存
储
分
配

```
#include<iostream>
using namespace std;
class Point
{ //类的声明同例6-16, 略 };
int main()
{ Point *Ptr=new Point[2]; //创建对象数组
  Ptr[0].Move(5,10); //通过指针访问数组元素的成员
  Ptr[1].Move(15,20); //通过指针访问数组元素的成员
  cout<<"Deleting..."<<endl;
  delete[ ] Ptr; //删除整个对象数组
  return 0;
}
```



运行结果:

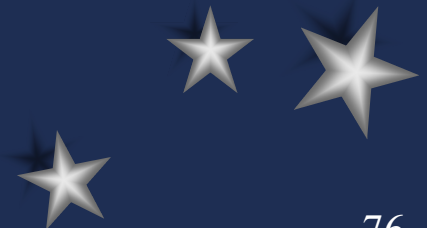
Default Constructor called.

Default Constructor called.

Deleting...

Destructor called.

Destructor called.



例6-18 动态数组类

```
#include<iostream>
using namespace std;
class Point
{ //类的声明同例6-16 ... };
class ArrayOfPoints
{ public:
    ArrayOfPoints(int n)
    { numberOfPoints=n; points=new Point[n]; }
    ~ArrayOfPoints()
    { cout<<"Deleting..."<<endl;
      numberOfPoints=0; delete[] points;
    }
    Point& Element(int n)
    { return points[n]; }
private:
    Point *points;
    int numberOfPoints;
};
```



```
void main()
{
    int number;
    cout<<"Please enter the number of
    points:";
    cin>>number;
    //创建对象数组
    ArrayOfPoints points(number);
    //通过指针访问数组元素的成员
    points.Element(0).Move(5,10);
    //通过指针访问数组元素的成员
    points.Element(1).Move(15,20);
}
```



浅拷贝与深拷贝

浅拷贝与深拷贝

- 浅拷贝
 - 实现对象间数据元素的一一对应复制。
- 深拷贝
 - 当被复制的对象数据成员是指针类型时，不是复制该指针成员本身，而是将指针所指的对象进行复制。



例6-20对象的浅拷贝

浅
拷
贝
与
深
拷
贝

```
#include<iostream>
using namespace std;
class Point
{ //类的声明同例6-16
  //.....
};
class ArrayOfPoints
{
  //类的声明同例6-18
  //.....
};
```




```
void main()
{ int number;
  cin>>number;
  ArrayOfPoints pointsArray1(number);
  pointsArray1.Element(0).Move(5,10);
  pointsArray1.Element(1).Move(15,20);
  ArrayOfPoints pointsArray2(pointsArray1);
  cout<<"Copy of pointsArray1:"<<endl;
  cout<<"Point_0 of array2: "
    <<pointsArray2.Element(0).GetX()
    <<" , "<<pointsArray2.Element(0).GetY()<<endl;
  cout<<"Point_1 of array2: "
    <<pointsArray2.Element(1).GetX()
    <<" , "<<pointsArray2.Element(1).GetY()<<endl;
```

```
pointsArray1.Element(0).Move(25,30);
pointsArray1.Element(1).Move(35,40);
cout<<"After the moving of pointsArray1:"<<endl;
cout<<"Point_0 of array2: "
    <<pointsArray2.Element(0).GetX()
    <<" , "<<pointsArray2.Element(0).GetY()<<endl;
cout<<"Point_1 of array2: "
    <<pointsArray2.Element(1).GetX()
    <<" , "<<pointsArray2.Element(1).GetY()<<endl;
}
```



运行结果如下:

Please enter the number of points:2

Default Constructor called.

Default Constructor called.

Copy of pointsArray1:

Point_0 of array2: 5, 10

Point_1 of array2: 15, 20

After the moving of pointsArray1:

Point_0 of array2: 25, 30

Point_1 of array2: 35, 40

Deleting...

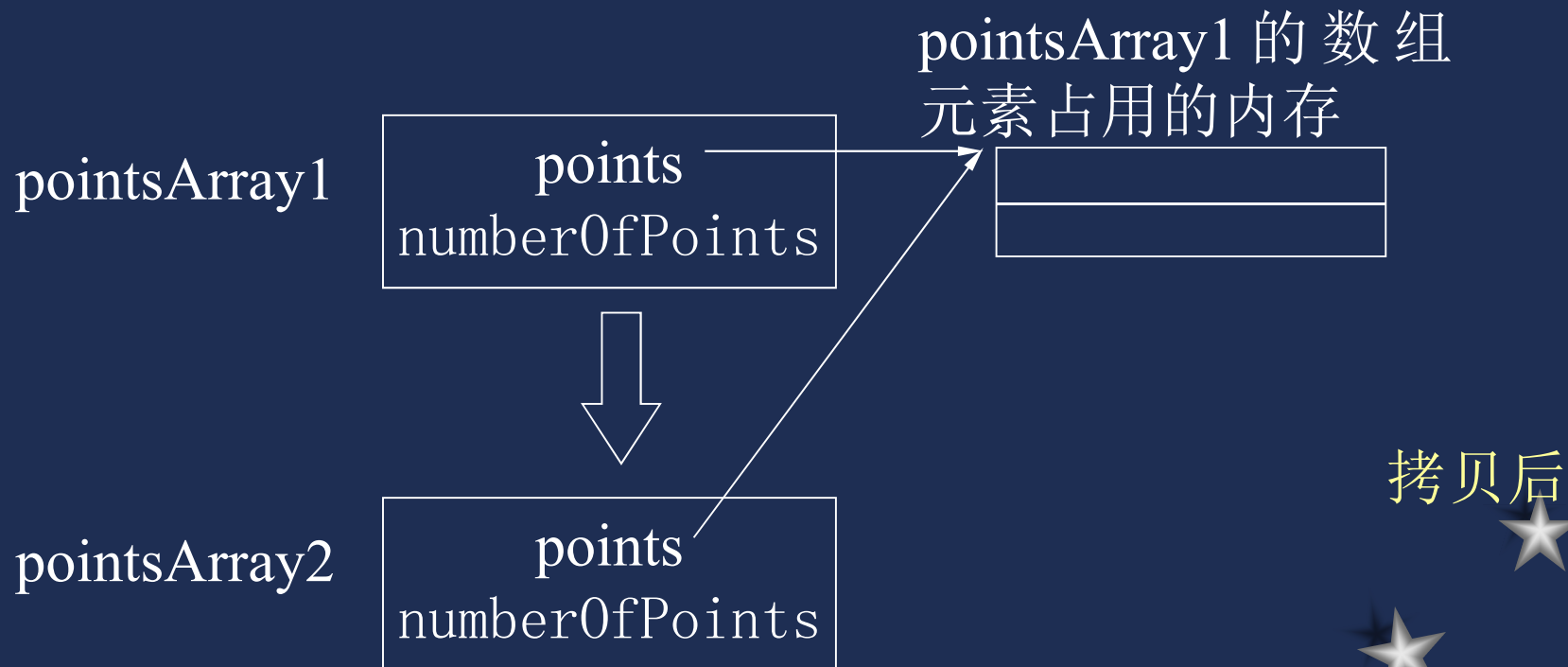
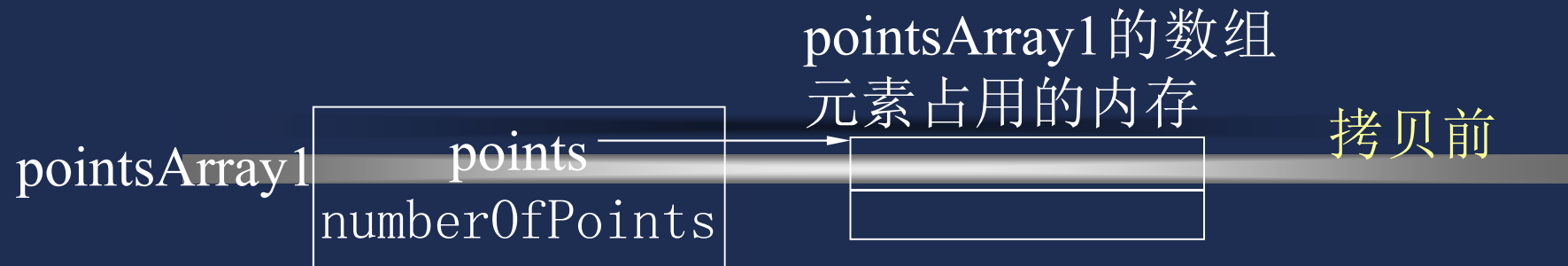
Destructor called.

Destructor called.

Deleting...

接下来程序出现异常，也就是运行错误。





例6-21 对象的深拷贝

浅
拷
贝
与
深
拷
贝

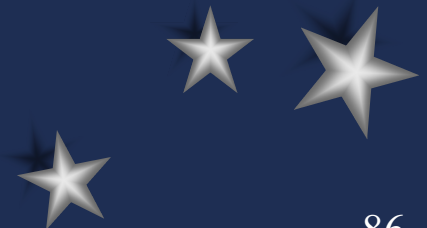
```
#include<iostream>
using namespace std;
class Point
{ //类的声明同例6-16 ..... };
class ArrayOfPoints
{ public:
    ArrayOfPoints(ArrayOfPoints&
pointsArray);
    //其它成员同例6-18
};
```



```
ArrayOfPoints ::ArrayOfPoints  
(ArrayOfPoints& pointsArray)
```

```
{  numberOfPoints  
    =pointsArray.numberOfPoints;  
    points=new Point[numberOfPoints];  
    for (int i=0; i<numberOfPoints; i++)  
        points[i].Move(pointsArray.Element(i).GetX(),  
                        pointsArray.Element(i).GetY());  
}
```

```
void main()  
{  //同例6-20  }
```



程序的运行结果如下:

Please enter the number of points:2

Default Constructor called.

Default Constructor called.

Default Constructor called.

Default Constructor called.

Copy of pointsArray1:

Point_0 of array2: 5, 10

Point_1 of array2: 15, 20

After the moving of pointsArray1:

Point_0 of array2: 5, 10

Point_1 of array2: 15, 20

Deleting...

Destructor called.

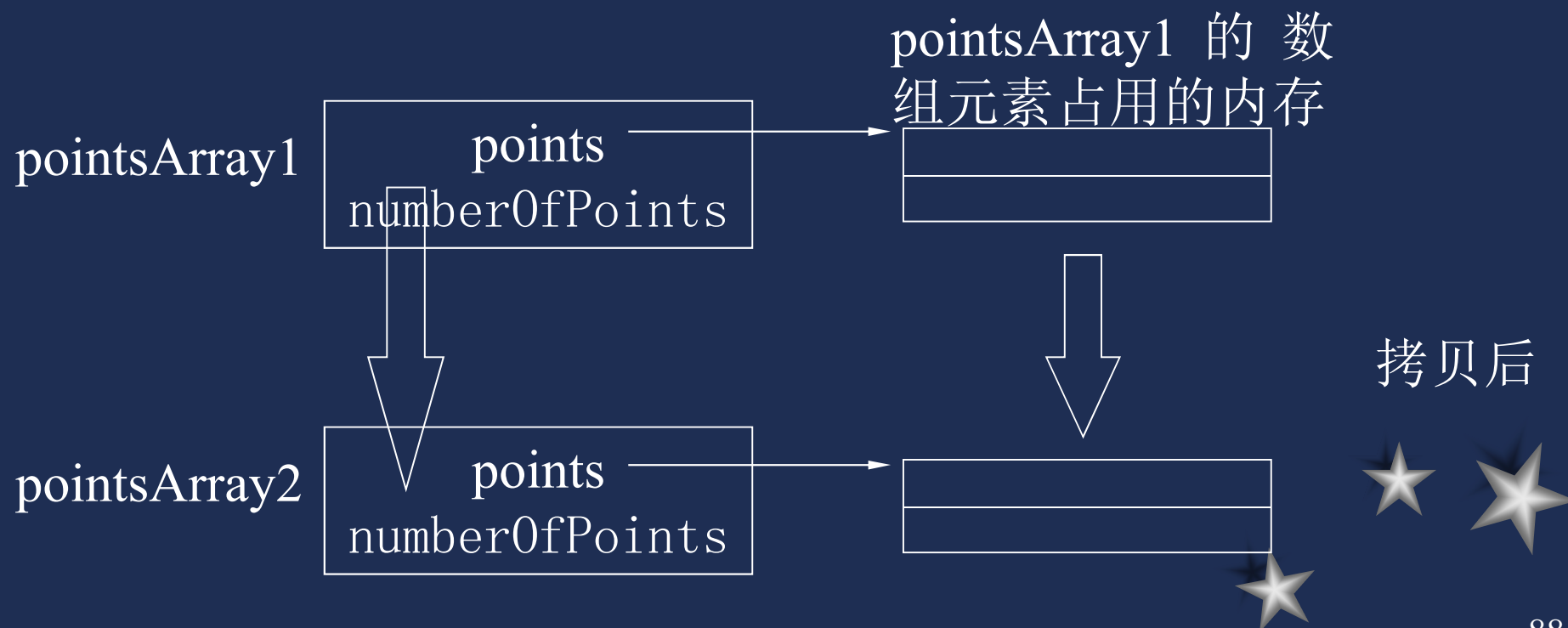
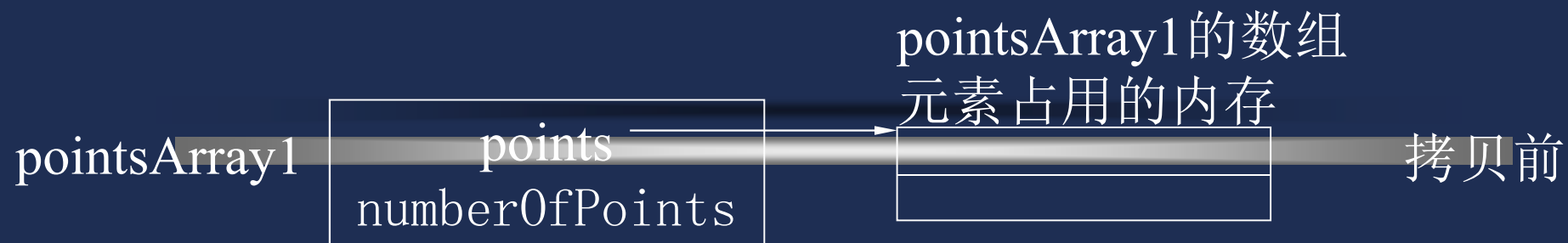
Destructor called.

Deleting...

Destructor called.

Destructor called.





用字符数组存储和处理字符串

字符串

字符数组的声明和引用

字符串

字符串常量，例如："china"

没有字符串变量，用字符数组来存放字符串

字符串以'\0'为结束标志

字符数组的初始化

例：static char

```
str[8]={112,114,111,103,114,97,109,0};
```

```
static char str[8]={'p','r','o','g','r','a','m','\0'};
```

```
static char str[8]="program";
```

```
static char str[]="program";
```



例6-22 输出一个字符串

字
符
串

```
#include<iostream>
using namespace std;
void main()
{
    static char c[10]={'I',' ','a','m',' ','a',' ','b','o','y'};
    int i;
    for(i=0;i<10;i++)
        cout<<c[i];
    cout<<endl;
}
```

运行结果:

I am a boy



例6-23 输出一个钻石图形

字
符
串

```
#include<iostream>
using namespace std;
void main()
{ static char diamond[][5]={{' ',' ','*'},
                              {' ','*',' ','*'}, {'*',' ',' ',' ','*'},
                              {' ','*',' ','*'}, {' ',' ','*'}};

  int i,j;
  for (i=0;i<5;i++)
  { for(j=0;j<5 && diamond[i][j]!=0;j++)
    cout<<diamond[i][j];
    cout<<endl;
  }
}
```

运行结果:

```

      *
     * *
    *   *
   *   *
  *
```



例6.21 string类应用举例

字
符
串

```
#include <string>
#include <iostream>
using namespace std ;
void trueFalse(int x)
{
    cout << (x? "True": "False") << endl;
}
```



```
void main()
{  string S1="DEF", S2="123";
   char CP1[ ]="ABC";
   char CP2[ ]="DEF";
   cout << "S1 is " << S1 << endl;
   cout << "S2 is " << S2 << endl;
   cout<<"length of S2:"<<S2.length()<<endl;
   cout << "CP1 is " << CP1 << endl;
   cout << "CP2 is " << CP2 << endl;
   cout << "S1<=CP1 returned ";
   trueFalse(S1<=CP1);
   cout << "CP2<=S1 returned ";
   trueFalse(CP2<=S1);
   S2+=S1;
   cout<<"S2=S2+S1:"<<S2<<endl;
   cout<<"length of S2:"<<S2.length()<<endl;
}
```

The End!



```
void main()
{
    int number;
    cout<<"Please enter the number of
    points:";
    cin>>number;
    //创建对象数组
    ArrayOfPoints points(number);
    //通过指针访问数组元素的成员
    points.Element(0).Move(5,10);
    //通过指针访问数组元素的成员
    points.Element(1).Move(15,20);
}
```

运行结果如下：

Please enter the number of points:2

Default Constructor called.

Default Constructor called.

Deleting...

Destructor called.

Destructor called.


```
void main() //主函数
```

```
{
```

```
//指向函数的指针，指向类的静态成员函数
```

```
void (*gc)()=Point::GetC;
```

```
Point A(4,5); //声明对象A
```

```
cout<<"Point A,"<<A.GetX()<<","<<A.GetY();
```

```
gc(); //输出对象序号，通过指针访问静态函数成员
```

```
Point B(A); //声明对象B
```

```
cout<<"Point B,"<<B.GetX()<<","<<B.GetY();
```

```
gc(); //输出对象序号，通过指针访问静态函数成员
```

```
}
```



动态创建多维数组

new 类型名T[下标表达式1][下标表达式2]...;

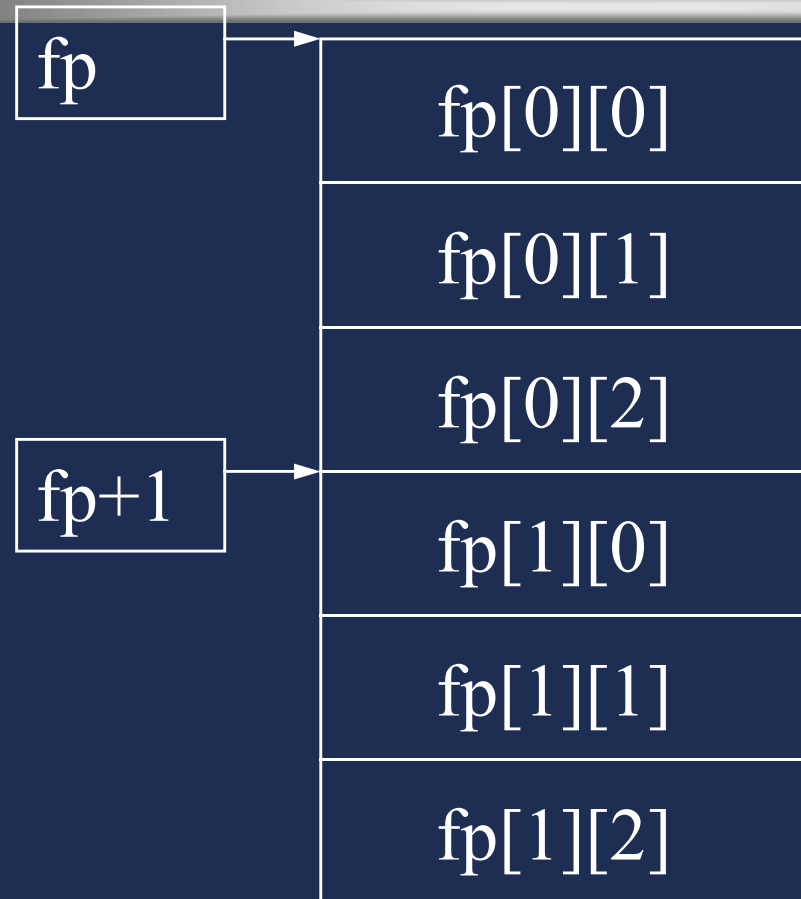
如果内存申请成功，**new**运算返回一个指向新分配内存首地址的指针，是一个T类型的数组，数组元素的个数为除最左边一维外各维下标表达式的乘积。例如：

```
char (*fp)[3];
```

```
fp = new char[2][3];
```



```
char (*fp)[3];
```



例6-18动态创建多维数组

动
态
存
储
分
配

```
#include<iostream>
using namespace std;
void main()
{ float (*cp)[9][8];
  int i,j,k;
  cp = new float[8][9][8];
  for (i=0; i<8; i++)
    for (j=0; j<9; j++)
      for (k=0; k<9; k++)
        *(*(*cp+i)+j)+k)=i*100+j*10+k;
  //通过指针访问数组元素
```

```
for (i=0; i<8; i++)  
{  
    for (j=0; j<9; j++)  
    {  
        for (k=0; k<8; k++)  
            //将指针cp作为数组名使用，  
            //通过数组名和下标访问数组元素  
            cout<<cp[i][j][k]<<" ";  
            cout<<endl;  
        }  
        cout<<endl;  
    }  
}
```



动态存储分配函数

动态存储分配

- `void *malloc(size);`

参数**size**: 欲分配的字节数

返回值: 成功, 则返回**void**型指针。
失败, 则返回空指针。

头文件: `<cstdlib>` 和 `<cmalloc>`



动态内存释放函数

动
态
存
储
分
配

- `void free(void *memblock);`

参数memblock:

指针，指向需释放的 内存。

返回值：无

头文件：<cstdlib> 和 <cmalloc>



浅拷贝与深拷贝

浅拷贝与深拷贝

- 浅拷贝
 - 实现对象间数据元素的一一对应复制。
- 深拷贝
 - 当被复制的对象数据成员是指针类型时，不是复制该指针成员本身，而是将指针所指的对象进行复制。



例6-20对象的浅拷贝

浅
拷
贝
与
深
拷
贝

```
#include<iostream>
using namespace std;
class Point
{ //类的声明同例6-16
  //.....
};
class ArrayOfPoints
{
  //类的声明同例6-18
  //.....
};
```



```
void main()
{ int number;
  cin>>number;
  ArrayOfPoints pointsArray1(number);
  pointsArray1.Element(0).Move(5,10);
  pointsArray1.Element(1).Move(15,20);
  ArrayOfPoints pointsArray2(pointsArray1);
  cout<<"Copy of pointsArray1:"<<endl;
  cout<<"Point_0 of array2: "
    <<pointsArray2.Element(0).GetX()
    <<" , "<<pointsArray2.Element(0).GetY()<<endl;
  cout<<"Point_1 of array2: "
    <<pointsArray2.Element(1).GetX()
    <<" , "<<pointsArray2.Element(1).GetY()<<endl;
```

```
pointsArray1.Element(0).Move(25,30);
pointsArray1.Element(1).Move(35,40);
cout<<"After the moving of pointsArray1:"<<endl;
cout<<"Point_0 of array2: "
    <<pointsArray2.Element(0).GetX()
    <<" , "<<pointsArray2.Element(0).GetY()<<endl;
cout<<"Point_1 of array2: "
    <<pointsArray2.Element(1).GetX()
    <<" , "<<pointsArray2.Element(1).GetY()<<endl;
}
```



运行结果如下:

Please enter the number of points:2

Default Constructor called.

Default Constructor called.

Copy of pointsArray1:

Point_0 of array2: 5, 10

Point_1 of array2: 15, 20

After the moving of pointsArray1:

Point_0 of array2: 25, 30

Point_1 of array2: 35, 40

Deleting...

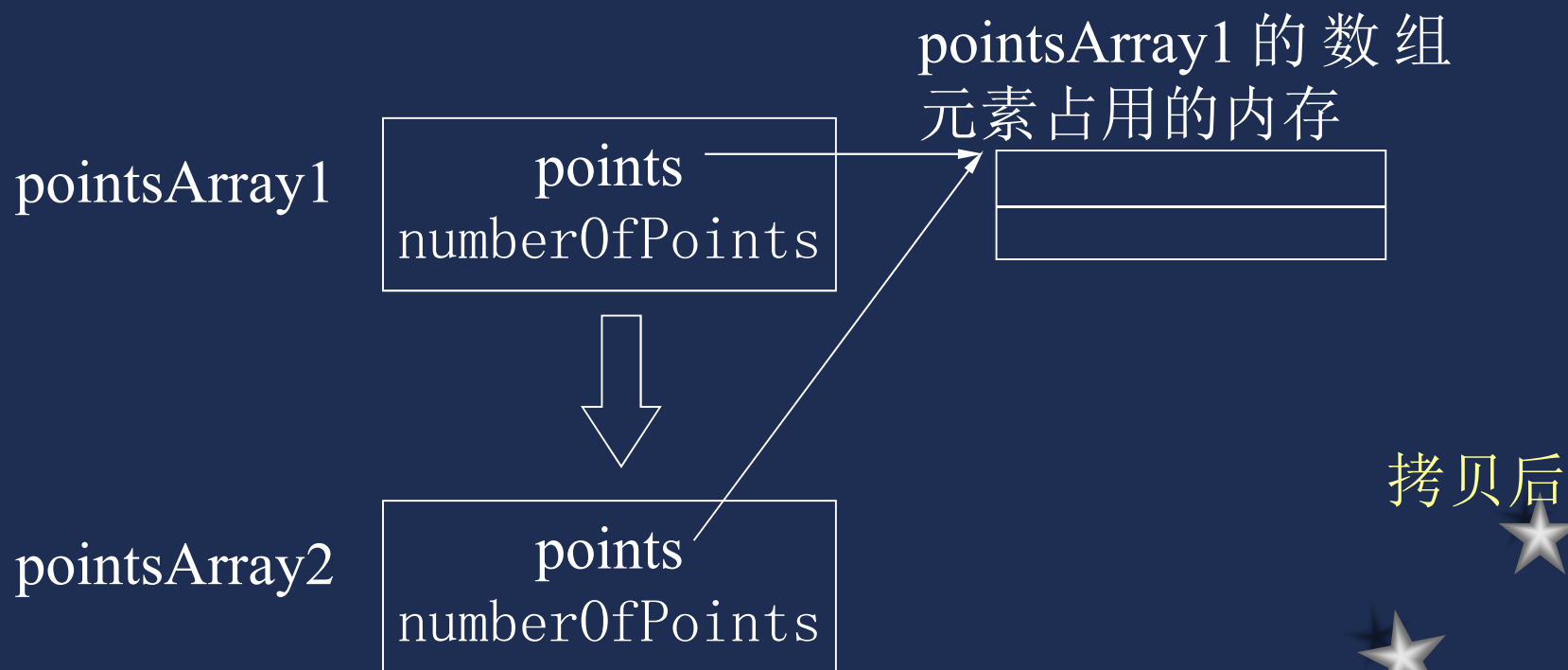
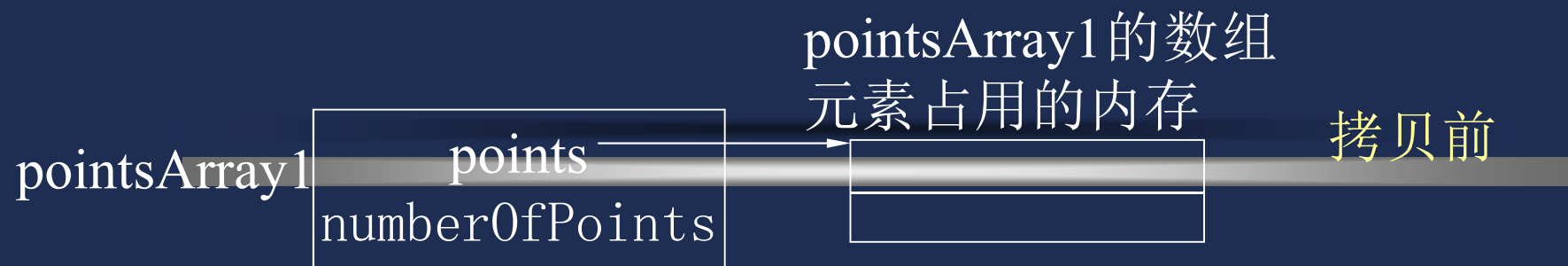
Destructor called.

Destructor called.

Deleting...

接下来程序出现异常，也就是运行错误。





例6-21 对象的深拷贝

浅
拷
贝
与
深
拷
贝

```
#include<iostream>
using namespace std;
class Point
{ //类的声明同例6-16 ..... };
class ArrayOfPoints
{ public:
    ArrayOfPoints(ArrayOfPoints&
pointsArray);
    //其它成员同例6-18
};
```



```
ArrayOfPoints ::ArrayOfPoints  
(ArrayOfPoints& pointsArray)
```

```
{  numberOfPoints  
    =pointsArray.numberOfPoints;  
    points=new Point[numberOfPoints];  
    for (int i=0; i<numberOfPoints; i++)  
        points[i].Move(pointsArray.Element(i).GetX(),  
                        pointsArray.Element(i).GetY());  
}
```

```
void main()  
{  //同例6-20  }
```



程序的运行结果如下:

Please enter the number of points:2

Default Constructor called.

Default Constructor called.

Default Constructor called.

Default Constructor called.

Copy of pointsArray1:

Point_0 of array2: 5, 10

Point_1 of array2: 15, 20

After the moving of pointsArray1:

Point_0 of array2: 5, 10

Point_1 of array2: 15, 20

Deleting...

Destructor called.

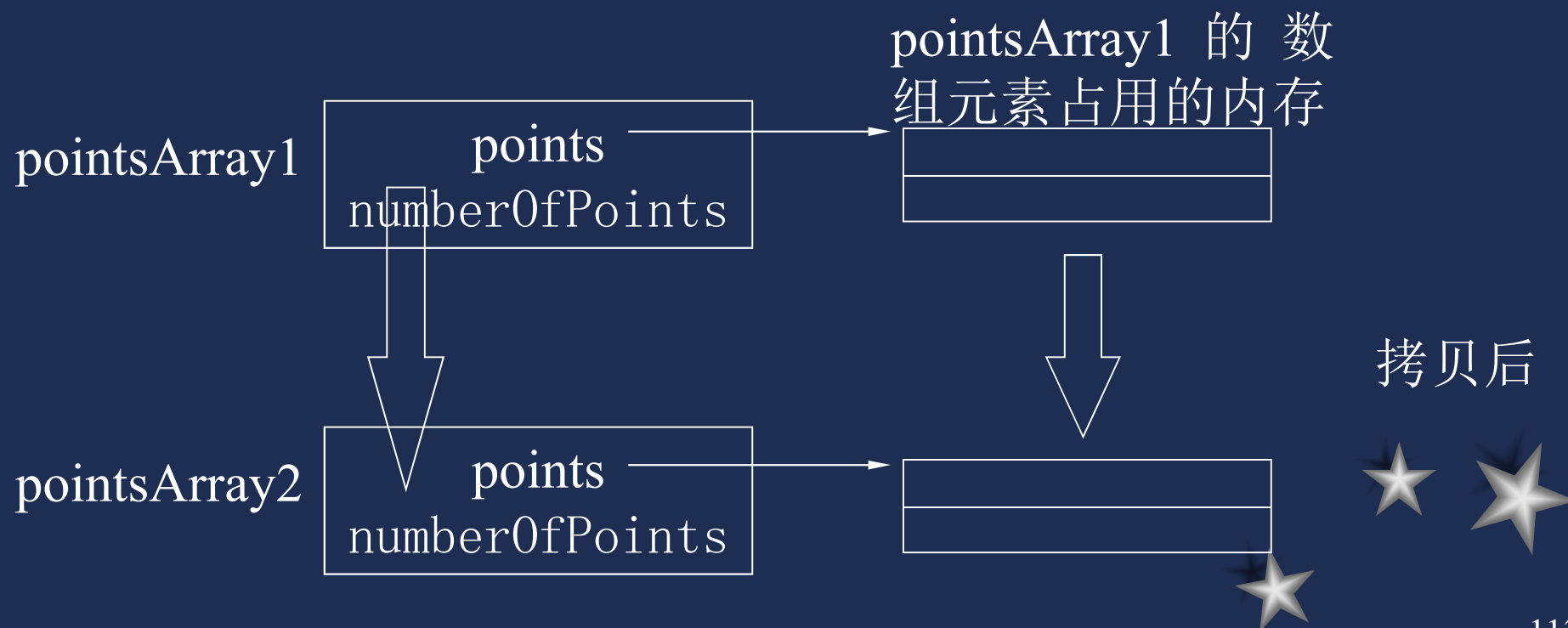
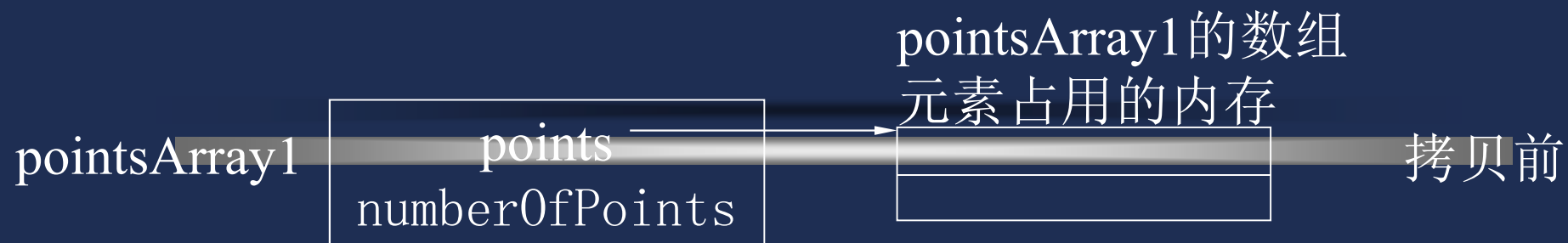
Destructor called.

Deleting...

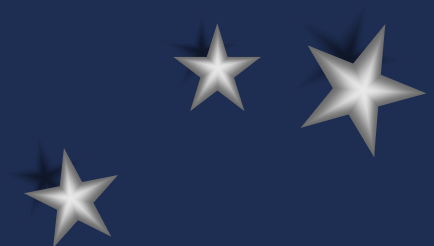
Destructor called.

Destructor called.





The End!



一维数组应用举例

数
组

循环从键盘读入若干组选择题答案，计算并输出每组答案的正确率，直到输入ctrl+z为止。

每组连续输入5个答案，每个答案可以是'a'..'d'。



```
#include <iostream>
using namespace std;
void main(void)
{  char key[ ] = {'a','c','b','a','d'};
   char c;
   int ques = 0, numques = 5, numcorrect = 0;
   cout << "Enter the " << numques << " question tests:" << endl;
   while (cin.get(c))
   {  if (c != '\n')
       if (c == key[ques])
       {  numcorrect++;
          cout << " ";
       }
       else    cout << "*";
       else
       {  cout << " Score " << float(numcorrect)/numques*100 << "%";
          ques = 0;           // reset variables
          numcorrect = 0;
          cout << endl;
          continue;
       }
       ques++;
   }
}
```



运行结果:

acbba

**** Score 60%**

acbad

Score 100%

abbda

*** ** Score 40%**

bdcba

******* Score 0%**



用字符数组存储和处理字符串

字符串

字符数组的声明和引用

字符串

字符串常量，例如："china"

没有字符串变量，用字符数组来存放字符串

字符串以'\0'为结束标志

字符数组的初始化

例：static char

```
str[8]={112,114,111,103,114,97,109,0};
```

```
static char str[8]={'p','r','o','g','r','a','m','\0'};
```

```
static char str[8]="program";
```

```
static char str[]="program";
```



例6-22 输出一个字符串

字
符
串

```
#include<iostream>
using namespace std;
void main()
{
    static char c[10]={'I',' ','a','m',' ','a',' ','b','o','y'};
    int i;
    for(i=0;i<10;i++)
        cout<<c[i];
    cout<<endl;
}
```

运行结果:

I am a boy



例6-23 输出一个钻石图形

字
符
串

```
#include<iostream>
using namespace std;
void main()
{ static char diamond[][5]={{' ',' ','*'},
                              {' ','*',' ','*'}, {'*',' ',' ',' ','*'},
                              {' ','*',' ','*'}, {' ',' ','*'}};

  int i,j;
  for (i=0;i<5;i++)
  { for(j=0;j<5 && diamond[i][j]!=0;j++)
    cout<<diamond[i][j];
    cout<<endl;
  }
}
```

运行结果:

```

      *
     * *
    *   *
   *   *
  *
```



字符串的输入/输出

字符串

- 方法

- 逐个字符输入输出
- 将整个字符串一次输入或输出
例: `char c[]="China";`
`cout<<c;`

- 注意

- 输出字符不包括 '\0'
- 输出字符串时, 输出项是字符数组名, 输出时遇到 '\0' 结束。
- 输入多个字符串时, 以空格分隔; 输入单个字符串时其中 不能有空格。

例如:

程序中有下列语句:

```
static char str1[5], str2[5], str3[5];  
cin>>str1>>str2>>str3;
```

运行时输入数据:

How are you?

内存中变量状态如下:

str1:	H	o	w	\0	
str2:	a	r	e	\0	
str3:	y	o	u	?	\0



若改为:

```
static char str[13];  
cin>>str;
```

运行时输入数据:

How are you?

内存中变量 str 内容如下:

str:

H	o	w	\0									
---	---	---	----	--	--	--	--	--	--	--	--	--



用字符数组存储和处理字符串

字
符
串

注意！若有如下声明：

```
char a[4];
```

— 错误的：

```
a="abc";
```

正确的：

```
strcpy(a,"abc");
```



整行输入字符串

字符串

- **cin.getline(字符数组名St, 字符个数N, 结束符);**
功能：一次连续读入多个字符（可以包括空格），直到读满N个，或遇到指定的结束符（默认为'\n'）。读入的字符串存放于字符数组St中。读取但不存储结束符。
- **cin.get(字符数组名St, 字符个数N, 结束符);**
功能：一次连续读入多个字符（可以包括空格），直到读满N个，或遇到指定的结束符（默认为'\n'）。读入的字符串存放于字符数组St中。
既不读取也不存储结束符。



整行输入字符串举例

字
符
串

```
#include <iostream>
using namespace std;
void main (void)
{ char city[80];
  char state[80];
  int i;
  for (i = 0; i < 2; i++)
  { cin.getline(city,80,',');
    cin.getline(state,80,'\n');
    cout << "City: " << city << " State: "
          << state << endl;
  }
}
```



运行结果

Beijing,China

City: Beijing Country: China

Shanghai,China

City: Shanghai Country: China



字符串处理函数

字
符
串

strcat（连接），**strcpy**（复制），
strcmp（比较），**strlen**（求长度），
strlwr（转换为小写），
strupr（转换为大写）

头文件<**cstring**>



例6.21 string类应用举例

字
符
串

```
#include <string>
#include <iostream>
using namespace std ;
void trueFalse(int x)
{
    cout << (x? "True": "False") << endl;
}
```



例6-6 void类型指针的使用

指
针

```
void vobject; //错，不能声明void类型的变量
void *pv;     //对，可以声明void类型的指针
int *pint; int i;
void main()   //void类型的函数没有返回值
{
    pv = &i;   //void类型指针指向整型变量
    // void指针赋值给int指针需要类型强制转换：
    pint = (int *)pv;
}
```



指针变量的赋值运算

指针

指针名=地址

- “地址”中存放的数据类型与指针类型必须相符。
- 向指针变量赋的值必须是地址常量或变量，不能是普通整数。但可以赋值为整数0，表示空指针。
- 指针的类型是它所指向变量的类型，而不是指针本身数据值的类型，任何一个指针本身的数据值都是 **unsigned long int** 型。
- 允许声明指向 **void** 类型的指针。该指针可以被赋予任何类型对象的地址。

例： **void *general;**

```
void main()
{  string S1="DEF", S2="123";
   char CP1[ ]="ABC";
   char CP2[ ]="DEF";
   cout << "S1 is " << S1 << endl;
   cout << "S2 is " << S2 << endl;
   cout<<"length of S2:"<<S2.length()<<endl;
   cout << "CP1 is " << CP1 << endl;
   cout << "CP2 is " << CP2 << endl;
   cout << "S1<=CP1 returned ";
   trueFalse(S1<=CP1);
   cout << "CP2<=S1 returned ";
   trueFalse(CP2<=S1);
   S2+=S1;
   cout<<"S2=S2+S1:"<<S2<<endl;
   cout<<"length of S2:"<<S2.length()<<endl;
}
```

实习 (5) : 数组与指针

1. 用指针作为函数形参，改写实习 (4) 中，以数组作为函数形参，统计一维数组和二维数组元素的均值。
2. 分别用指针和引用来实现交换两个整数值的函数 swap.
3. 实现一个 PointArray 类，用于点对象数组动态管理，提供接口方位数组元素的内容。（上课的例子）
4. 编码实践 `const int*`, `int const *`, `int * const` 的区别
5. 实现二维数组的动态分配和释放
6. 用函数指针在主函数中调用，求一个一维数组的 `min`, `max`, `sum`, `mean`, `std` 的代码。

