第 6 章 数组、指针与字符串（1）

郑 莉 清华大学

教材：C++语言程序设计（第5版） 郑莉 清华大学出版社

目录

数组的定义与使用

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

数组的定义

类型说明符　数组名[常量表达式] [常量表达式]……；

↑

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]

例如: 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。

数组的使用

- 使用数组元素
  - 数组必须先定义，后使用。
  - 可以逐个引用数组元素。
  - 例如：

    a[0]=a[5]+a[7]-a[2*3]

    b[1][2]=a[2][3]/2

例6-1

```
#include <iostream>
using namespace std;
int main() {
  int a[10], b[10];
  for(int i = 0; i < 10; i++) {
    a[i] = i * 2 - 1;
    b[10 - i - 1] = a[i];
  }
  for (const auto &e:a) //范围for循环，输出a中每个元素
       cout << e << " ";
  cout << endl;
  for (int i = 0; i <10; i++) //下标迭代循环，输出b中每个元素
       cout << b[i] << " ";
  cout << endl;
  return 0;
}
```

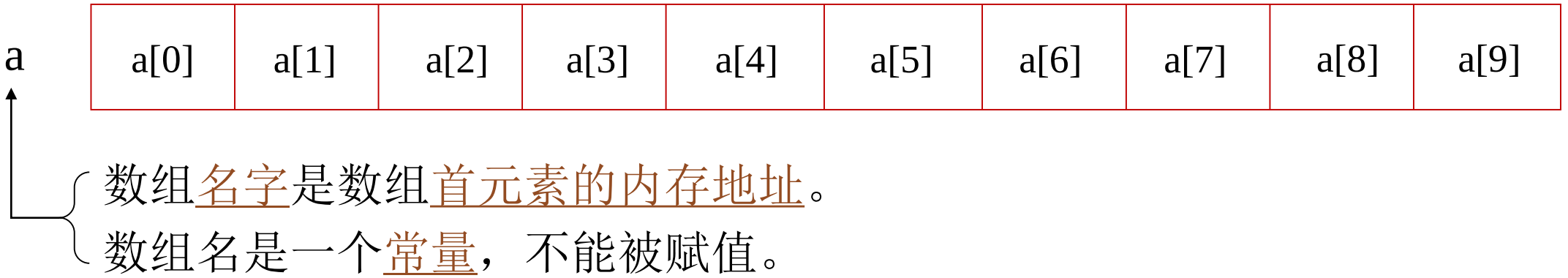
数组的存储与初始化

一维数组的存储

数组元素在内存中顺次存放，它们的地址是连续的。元素间物理地址上的相邻，对应着逻辑次序上的相邻。

例如：

```
int a[10];
```



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

- 列出全部元素的初始值

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

- 可以只给一部分元素指定初值

  例如：int a[10]={0,1,2,3,4};

- 在列出全部数组元素初值时，可以不指定数组长度

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

二维数组的存储

- 按行存放

  例如： float a[3][4];

可以理解为：

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

其中数组a的存储顺序为：

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

二维数组的初始化

- 将所有初值写在一个{}内，按顺序初始化
  - 例如：static int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};
- 分行列出二维数组元素的初值
  - 例如：static int a[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
- 可以只对部分元素初始化
  - 例如：static int a[3][4]={{1},{0,6},{0,0,11}};
- 列出全部初始值时，第1维下标个数可以省略
  - 例如：static int a[][4]={1,2,3,4,5,6,7,8,9,10,11,12};
  - 或：static int a[][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};

- 如果不作任何初始化，局部作用域的非静态数组中会存在垃圾数据，static数组中的数据默认初始化为0
- 如果只对部分元素初始化，剩下的未显式初始化的元素，将自动被初始化为零

补充6_1：求Fibonacci数列的前20项，将结果存放于数组中

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

运行结果：

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

补充6_2: 计算某天是该年的第几天

给出一个包含年月日的日期，输出这个日期在该年中是第几天。要求用一个2*12的数组来存储不同年份的每月的日数，注意输入的判断和闰年的判断。

补充6_2：计算某天是该年的第几天

```
#include <iostream>
using namespace std;
int main() {
        int m[2][12] = {{31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
                        {31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}};
        int year, month, day;
        int run = 0;
        cout << "请输入三个正整数作为年月日：";
        cin >> year >> month >> day;
        if (year <= 0 || month < 1 || month > 12) {
                cout << "输入的日期不存在" << endl;
                return 0;
        }
```

补充6_2：计算某天是该年的第几天

```
if ((year % 100 != 0 && year % 4 == 0) || year % 400 == 0)
        run = 1;
if (day < 1 || day > m[run][month-1]) {
        cout << "输入的日期不存在" << endl;
        return 0;
}
int index = 0;
for (int i = 0; i < month - 1; i ++)
        index += m[run][i];
index += day;
cout << "输入日期为一年中的第" << index << "天" << endl;
return 0;
}
```

清華大學

数组作为函数参数

例6-2 使用数组名作为函数参数

主函数中初始化一个二维数组，表示一个矩阵，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。

例6-2 使用数组名作为函数参数

```
#include <iostream>
using namespace std;
void rowSum(int a[][4], int nRow) {
   for (int i = 0; i < nRow; i++) {
         for(int j = 1; j < 4; j++)
                  a[i][0] += a[i][j];
   }
}
int main() { //主函数
   //定义并初始化数组
  int table[3][4] = {{1, 2, 3, 4}, {2, 3, 4, 5}, {3, 4, 5, 6}};
```

技巧：多维数组通常用多重嵌套循环处理

例6-2 使用数组名作为函数参数

```
//输出数组元素
  for (int i = 0; i < 3; i++)        {
        for (int j = 0; j < 4; j++)
                cout << table[i][j] << "   ";
        cout << endl;
  }
  rowSum(table, 3);        //调用子函数，计算各行和
 //输出计算结果
  for (int i = 0; i < 3; i++)
        cout << "Sum of row " << i << " is " << table[i][0] << endl;
  return 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
```

例6_2修改：使用常数组作为函数参数

修改例6_2，以常数组作为函数的参数，观察编译时的情况

```
#include <iostream>
using namespace std;
void rowSum(const int a[][4], int nRow) {
   for (int i = 0; i < nRow; i++) {
         for(int j = 1; j < 4; j++)
                  a[i][0] += a[i][j];  //编译错误
   }
}
int main() { //主函数
   //函数体同例7_4，略
}
```

- 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（实质上是地址，关于地址详见6.2），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。

对象数组

对象数组的定义与访问

- 定义对象数组

  类名　数组名[元素个数];

- 访问对象数组元素

  通过下标访问

  数组名[下标].成员名

对象数组初始化

- 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

- 元素所属的类不声明构造函数，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

```
//Point.h
#ifndef _POINT_H
#define _POINT_H
class Point {    //类的定义
public:                          //外部接口
   Point();
   Point(int x, int y);
   ~Point();
   void move(int newX,int newY);
   int getX() const { return x; }
   int getY() const { return y; }
   static void showCount();     //静态函数成员
private:                         //私有数据成员
   int x, y;
};
#endif//_POINT_H
```

例6-3 对象数组应用举例

```
//Point.cpp
#include <iostream>
#include "Point.h"
using namespace std;
Point::Point() : x(0), y(0) {
   cout << "Default Constructor called." << endl;
}
Point::Point(int x, int y) : x(x), y(y) {
   cout << "Constructor called." << endl;
}
Point::~Point() {
   cout << "Destructor called." << endl;
}
void Point::move(int newX,int newY) {
   cout << "Moving the point to (" << newX << ", " << newY << ")" << endl;
   x = newX;
   y = newY;
}
```

例6-3 对象数组应用举例

```
//6-3.cpp
#include "Point.h"
#include <iostream>
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.
Moving the point to (10, 20)
Moving the point to (11, 21)
Exiting main...
Destructor called.
Destructor called.
```

指针的概念和定义、与地址相关的运算

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;
  - &var 表示变量 var 在内存中的起始地址

指针的概念

- 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

指针变量

- 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
- 声明和定义

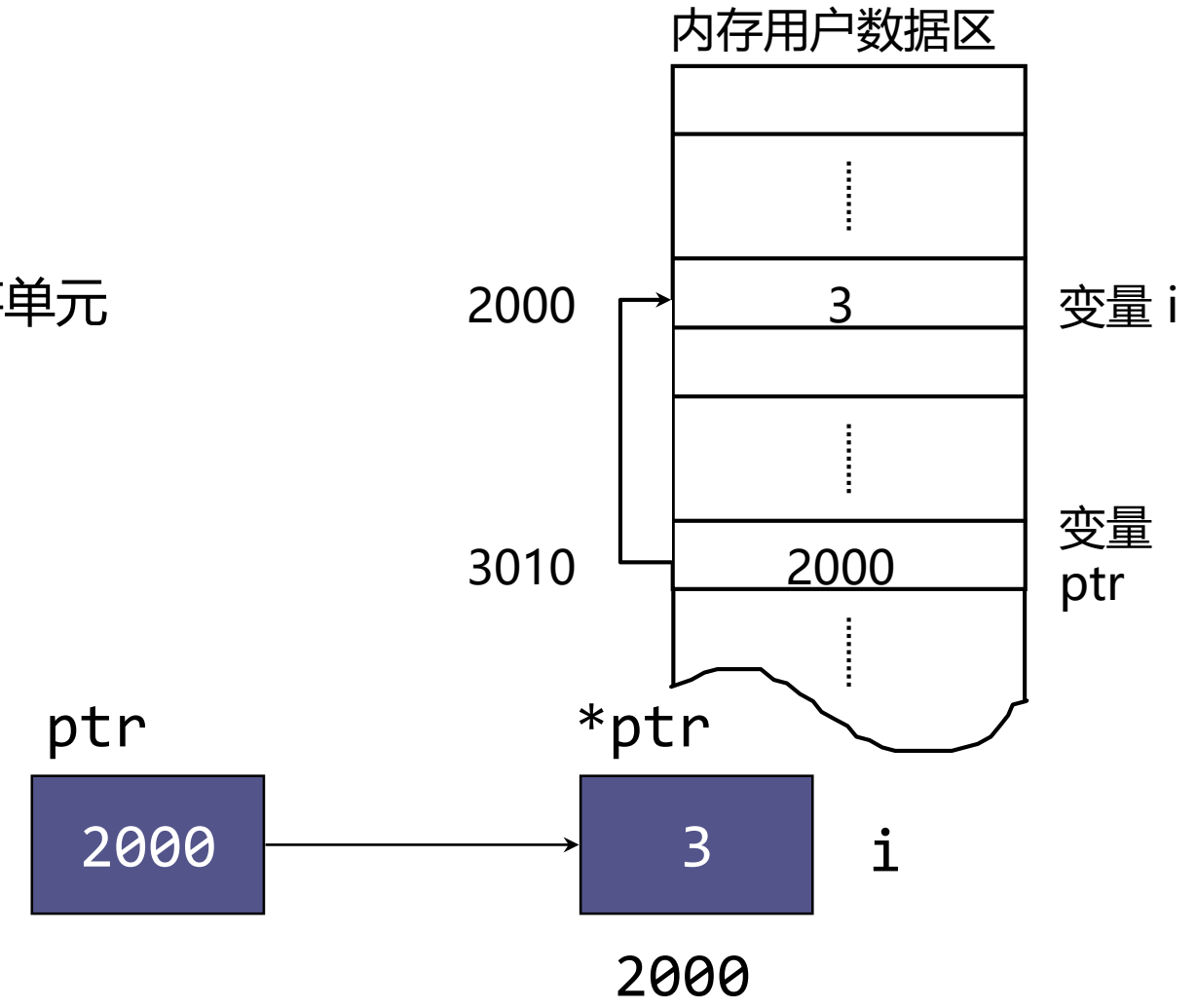
  例：static int i;

  static int* ptr = &i;

  指向int变量的指针
- 引用

  例1：i = 3;

  例2：*ptr = 3;



指针的初始化和赋值

指针变量的初始化

- 语法形式

  存储类型 数据类型 *指针名 = 初始地址;

- 例:

```
int *pa = &a;
```

- 注意事项
  - 用变量地址作为初值时，该变量必须在指针初始化之前已声明过，且变量类型应与指针类型一致。
  - 可以用一个已有合法值的指针去初始化另一个指针变量。
  - 不要用一个内部非静态变量去初始化 static 指针。

指针变量的赋值运算

- 语法形式
  指针名=地址
  注意：“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。
  例如：
  - 通过地址运算“&”求得已定义的变量和对象的起始地址
  - 动态内存分配成功时返回的地址
- 例外：整数0可以赋给指针，表示空指针。
- 允许定义或声明指向 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;`

指针空值nullptr

- 以往用0或者NULL去表达空指针的问题：
  - C/C++的NULL宏是个有很多潜在BUG的宏。因为有的库把其定义成整数0，有的定义成(void*)0。在C的时代还好。但是在C++的时代，这就会引发很多问题。
- C++11使用nullptr关键字，是表达更准确，类型安全的空指针

指向常量的指针

- 不能通过指向常量的指针改变所指对象的值，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;      //p1是指向常量的指针
int b;
p1 = &b;          //正确，p1本身的值可以改变
*p1 = 1;          //编译时出错，不能通过p1改变所指的对象
```

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，p2是指针常量，值不能改变
```

指针的运算

指针的算术运算、关系运算

指针类型的算术运算

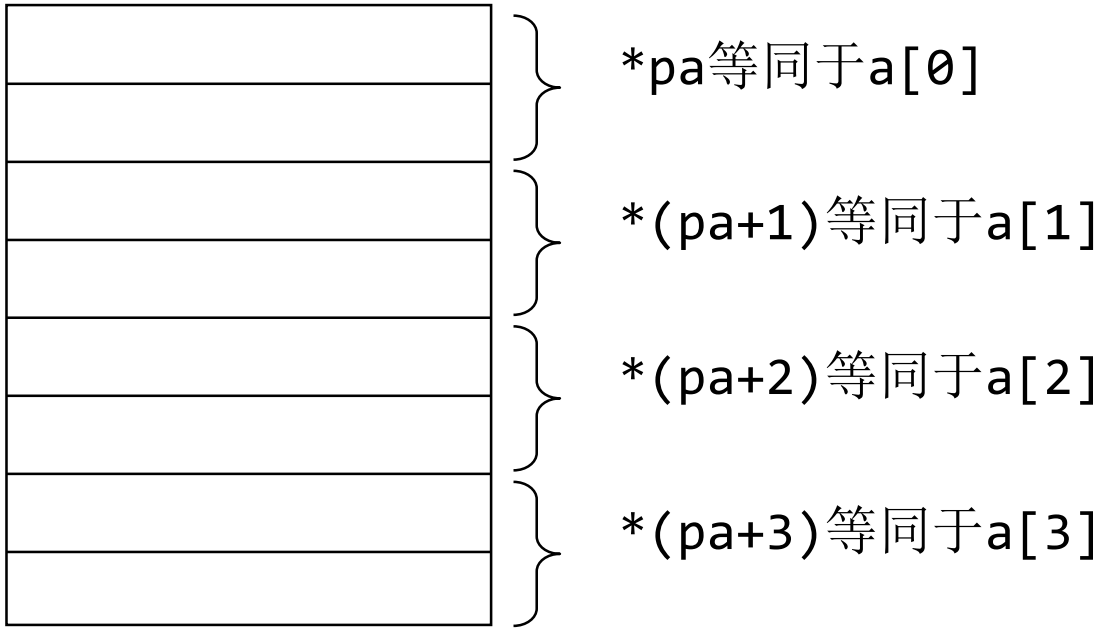
- 指针与整数的加减运算
- 指针++，--运算

指针类型的算术运算

- 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个数据的起始位置。
- 指针的++、--运算
  - 意义是指向下一个或前一个完整数据的起始。
- 运算的结果值取决于指针指向的数据类型，总是指向一个完整数据的起始位置。
- 当指针指向连续存储的同类型数据时，指针与整数的加减运和自增自减算才有意义。

指针与整数相加的含义

```
short a[4];
short *pa=a
```



指针类型的关系运算

- 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

用指针处理数组元素

数组是一组连续存储的同类型数据，可以通过指针的算术运算，使指针依次指向数组的各个元素，进而可以遍历数组。

定义指向数组元素的指针

- 定义与赋值

  例：int a[10], *pa;

  pa=&a[0]; 或 pa=a;

- 经过上述定义及赋值后
  - *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。
- 注意：不能写 a++，因为a是数组首地址、是常量。

例6-7

设有一个int型数组a，有10个元素。用三种方法输出各元素：

- 使用数组名和下标
- 使用数组名和指针运算
- 使用指针变量

例6-7 (1) 使用数组名和下标访问数组元素

```
#include <iostream>
using namespace std;
int main() {
   int a[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 };
   for (int i = 0; i < 10; i++)
         cout << a[i] << "   ";
   cout << endl;
   return 0;
}
```

例6-7 (2) 使用数组名和指针运算访问数组元素

```
#include <iostream>
using namespace std;
int main() {
   int a[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 };
   for (int i = 0; i < 10; i++)
        cout << *(a+i) << "  ";
   cout << endl;
   return 0;
}
```

例6-7 (3) 使用指针变量访问数组元素

```
#include <iostream>
using namespace std;
int main() {
   int a[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 };
   for (int *p = a; p < (a + 10); p++)
         cout << *p << "  ";
   cout << endl;
   return 0;
}
```

字符串

字符串常量

- 例："program"
- 各字符连续、顺序存放，每个字符占一个字节，以‘\0’结尾，相当于一个隐含创建的字符常量数组
- “program”出现在表达式中，表示这一char数组的首地址
- 首地址可以赋给char常量指针：
- const char *STRING1 = "program";

用字符数组存储字符串（C风格字符串）

- 例如

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

p	r	o	g	r	a	m	\0

用字符数组表示字符串的缺点

- 执行连接、拷贝、比较等操作，都需要显式调用库函数，很麻烦
- 当字符串长度很不确定时，需要用new动态创建字符数组，最后要用delete释放，很繁琐
- 字符串实际长度大于为它分配的空间时，会产生数组下标越界的错误

string类

使用字符串类string表示字符串

string实际上是对字符数组操作的封装

string类常用的构造函数

- string();　　//默认构造函数，建立一个长度为0的串

  例:

  string s1;

- string(const char *s);　//用指针s所指向的字符串常量初始化string对象

  例:

  string s2 = “abc”;

- string(const string& rhs);　//复制构造函数

  例:

  string s3 = s2;

string类常用操作

- s + t　　将串s和t连接成一个新串
- s = t　　用t更新s
- s == t　　判断s与t是否相等
- s != t　　判断s与t是否不等
- s < t　　判断s是否小于t（按字典顺序比较）
- s <= t　　判断s是否小于或等于t （按字典顺序比较）
- s > t　　判断s是否大于t （按字典顺序比较）
- s >= t　　判断s是否大于或等于t （按字典顺序比较）
- s[i] 访问串中下标为i的字符
- 例:
  - string s1 = "abc", s2 = "def";
  - string s3 = s1 + s2;　　//结果是"abcdef"
  - bool s4 = (s1 < s2);　　//结果是true
  - char s5 = s2[1];　　//结果是'e'

例6-23 string类应用举例

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

//根据value的值输出true或false
//title为提示文字
inline void test(const char *title, bool value)
{
   cout << title << " returns "
     << (value ? "true" : "false") << endl;
}
```

```
int main() {
   string s1 = "DEF";
   cout << "s1 is " << s1 << endl;
   string s2;
   cout << "Please enter s2: ";
   cin >> s2;
   cout << "length of s2: " << s2.length() << endl;

   //比较运算符的测试
   test("s1 <= \"ABC\"", s1 <= "ABC");
   test("\"DEF\" <= s1", "DEF" <= s1);

  //连接运算符的测试
   s2 += s1;
   cout << "s2 = s2 + s1: " << s2 << endl;
   cout << "length of s2: " << s2.length() << endl;
   return 0;
}
```

思考：如何输入整行字符串？

- 用cin的>>操作符输入字符串，会以空格作为分隔符，空格后的内容会在下一回输入时被读取

输入整行字符串

- getline可以输入整行字符串（要包括string头文件），例如：
  - getline(cin, s2);
- 输入字符串时，可以使用其它分隔符作为字符串结束的标志（例如逗号、分号），将分隔符作为getline的第3个参数即可，例如：
  - getline(cin, s2, ',');

例6-24 用getline输入字符串

```
include <iostream>
#include <string>
using namespace std;
int main() {
  for (int i = 0; i < 2; i++){
        string city, state;
        getline(cin, city, ',');
        getline(cin, state);
        cout << "City:" << city <<   “  State:" << state << endl;
  }
  return 0;
}
```

运行结果：
Beijing,China
City: Beijing State: China
San Francisco,the United States
City: San Francisco State: the United States