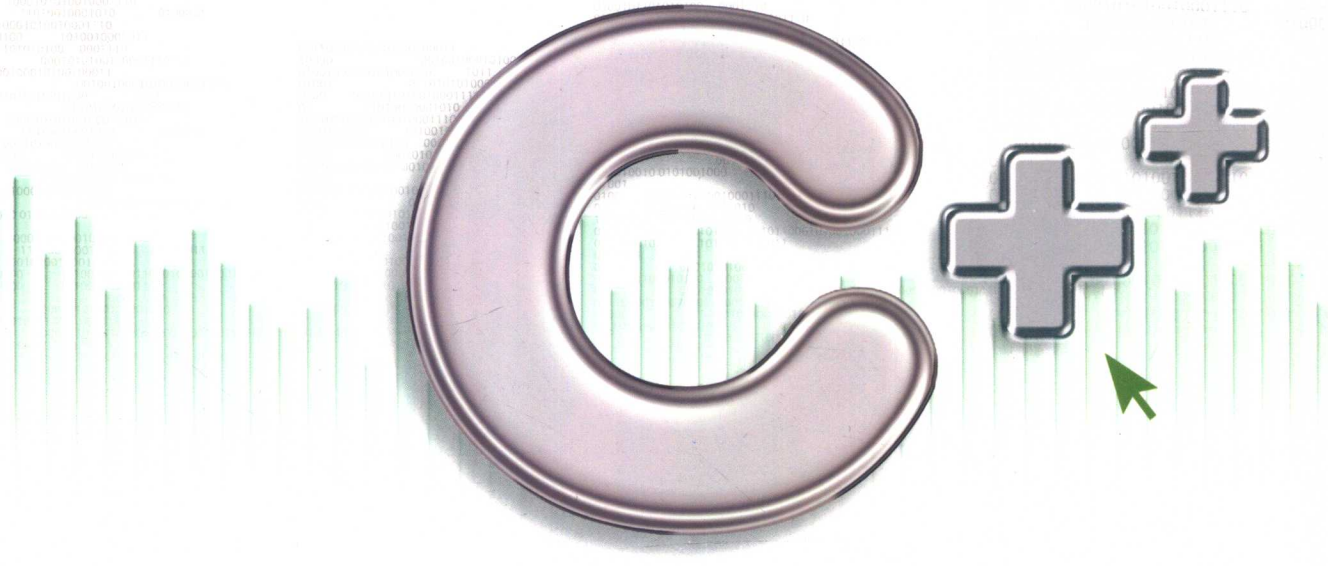
高等教育应用型人才“十三五”规划教材
高等教育应用型人才 计算机类专业 规划教材
本教材学习网站: www.lgsworks.com
江苏高校品牌专业建设工程资助项目



C/C++ 程序设计实用案例教程

◎丁 展 梁颖红 李广水 主编

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

高等教育应用型人才“十三五”规划教材
高等教育应用型人才计算机类专业规划教材
本教材学习网站：www.lgsworks.com
江苏高校品牌专业建设工程资助项目

C/C++程序设计实用案例教程

丁 展 梁颖红 李广水 主 编

電子工業出版社
Publishing House of Electronics Industry
北京 • BEIJING

内 容 简 介

本书主要利用 C/C++语言进行综合课程设计学习。书中案例涵盖了字符串处理、排序与查找、栈、链表与队列、树与图、递归与分治、集合与映射，以及 Win32 GUI 编程基础等知识。在本书的最后给出了四个综合课程设计的开发案例，每个案例都提供了完整的源代码，结合书与示例代码，读者可以很容易理解。

本书可以作为应用型本科院校程序设计课程的教材，也可作为软件开发人员的指导用书。

图书在版编目（CIP）数据

C/C++程序设计实用案例教程 / 丁展，梁颖红，李广水主编. —北京：电子工业出版社，2018.8
ISBN 978-7-121-34603-3

Ⅰ. ①C… Ⅱ. ①丁… ②梁… ③李… Ⅲ. ①C 语言－程序设计－高等学校－教材 Ⅳ. ①TP312.8

中国版本图书馆 CIP 数据核字（2018）第 142582 号

策划编辑：李 静
责任编辑：朱怀永　　文字编辑：李 静
印　　刷：北京七彩京通数码快印有限公司
装　　订：北京七彩京通数码快印有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编 100036
开　　本：787×1092　1/16　印张：14.75　字数：377.6 千字
版　　次：2018 年 8 月第 1 版
印　　次：2018 年 8 月第 1 次印刷
定　　价：42.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：（010）88254604，lijing@phei.com.cn。

前　言

随着信息技术的迅猛发展，程序设计在各个领域都起到举足轻重的作用。目前大部分高校的计算机、软件及相关专业都开设了各种程序设计和数据结构课程。然而掌握一门程序设计语言或者了解数据结构知识并不等同于“会编程”。在实际教学中存在大量的只会考试但不会写代码解决问题的学生，这是令人焦虑的现象。

虽然已出版的关于算法设计的书籍比较多，但大都介绍算法的理论基础。对于实际生活中的各种问题，需要灵活应用算法。本书精选了大量综合编程案例，基本涵盖了当前基础算法领域的热点问题。

本书的特色如下。

一、本书提供大量的综合编程实例，涉及各种基础算法的应用领域。

二、所有的综合编程实例都按照设计思路、相关概念、原理、具体实现进行介绍，学生能够很容易地掌握实例的算法设计思路。

三、本书介绍图形用户界面(GUI)编程基础，将四个综合编程实例的算法进行可视化，从而让读者理解并掌握算法可视化方法，进一步帮助学生理解与调试算法。

要运行本书中的绝大部分实例，只需要安装任何 C/C++开发工具，如 CodeBlocks、Visual Studio 2015 社区版、Dev C++等。如果要运行四个算法的可视化实例，则需要安装 CodeBlocks 或 Visual Studio 2015 社区版。以上工具均可以免费下载。如果计算机运行的还是 VC ++6.0，我们强烈建议卸载。因为它的编译器版本太低，不支持 C/C++新标准，并且对模板的支持不够。

本书共有 8 章，内容如下。

第 1 章字符串。主要介绍字符串复制、连接、与数的转换、查找、删除，以及字典比较等各种字符串操作。其后，给出若干字符串处理常见问题的解决方法。本章最后给出综合编程实例：公民数据模拟。

第 2 章排序和查找。主要介绍桶排序、qsort 排序、STL 中 std::sort 排序、std::greater 用法、二分查找算法、std::find 查找用法，并给出两个综合实例：员工 KPI 排名与 MOOC 期终成绩。

第 3 章栈、链表与队列。主要介绍 STL 中 std::stack、std::list、std::queue 与 std::deque 的用法。对以上数据结构分别给出了若干综合编程实例，如关于栈的火车调度问题、链表的约瑟夫环问题、队列的卡片游戏等。

第 4 章树与图。树的内容主要包括完全二叉树定义、二叉树遍历、手写二叉树的遍历、

二叉树高度计算，以及二叉树删除，其后给出四个关于树的综合编程实例。图内容主要包括图基本操作、图表示方法，最后给出三个关于图的综合编程实例。

第 5 章递归与分治。主要介绍七个典型的递归与分治问题：汉诺塔、子串组合、数组组合、格子排列、八皇后、循环赛日程安排与棋盘覆盖。

第 6 章集合与映射。主要介绍 STL 集合容器 std::set、映射容器 std::map、多键映射容器 std::multimap，以及哈希映射的用法。同时给出集合相似度与哈希冲突解决的综合实例，

第 7 章 Win32 GUI 编程基础。主要介绍使用 Win32 GUI 开发图形界面程序的基础。在实际工程中，算法的过程或者结果通常都需要可视化，一方面可以用来调试程序，另一方面可以用来展示算法效果。因此本章使用 CodeBlocks 或者 Visual Studio 2015 开发图形界面程序。内容涉及 Win32 消息基础和图形设备接口 GDI。同时给出综合编程实例：简单多边形的创建、绘制、平移与旋转。

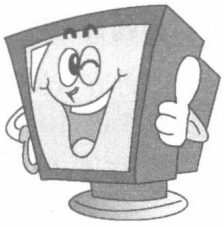
第 8 章介绍四个综合编程课程设计实例。分别是扑克洗牌、二叉树重建可视化、L-System 分形树建模，以及迷宫问题。这些实例要求读者能将算法实现和图形界面编程进行结合，从而把最终的算法和交互过程展现出来。

本书由丁展、梁颖红和李广水主编。在编写过程中，我们力求精益求精，但难免存在一些不足之处，如果读者使用本书时遇到问题，可以发送 E-mail 到 dingzh@jit.edu.cn，我们会及时给您回复。

编　者

2018 年 5 月

目 录



第 1 章　字符串处理

字符串几乎在所有程序设计语言中都占据了重要地位。在 C 语言中，C 语言库函数提供了一系列 API 处理各种字符串问题。而在 C++中，可以使用类 std::string、std::istringstream、std::ostringstream，以及若干 STL 算法解决各种字符串问题。

1.1　字符串基本操作

1.1.1　字符串复制

C 语言中可以使用 strcpy 函数实现字符串复制，strcpy 函数声明如下：

```
char * strcpy ( char * destination, const char * source );
```

strcpy 会将 source 指向的字符串内容连同末尾的 null 字符一起复制到 destination 指向的内存空间。为了避免溢出，destination 指向的字符串空间必须能容纳 source 中所有字符(包括 null 字符)，并且 source 和 destination 在内存上不能存在重叠现象。函数将返回 destination 指针。

例如，代码如下：

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

int main ()
{
  char str1[]="Test string";
  char str2[40];
  char str3[40];
  strcpy (str2,str1);
  strcpy (str3,"copy successful");
  printf ("str1: %s\nstr2: %s\nstr3: %s\n",str1,str2,str3);
  return 0;
}
```

输出：

```
str1: Test string
str2: Test string
str3: copy successful
```

一些公司在面试或笔试时喜欢直接使用 strcpy 的内部实现作为题目，以下是一个较为高效的实现过程，代码如下：

```
char *strcpy(char *dest, const char *src)
{
    char *save = dest;
    while(*dest++ = *src++);
    return save;
}
```

除了 strcpy 外，C 语言库函数中还提供了 strncpy 函数用于复制指定数目的字符到目标区域。strncpy 函数声明如下：

```
char * strncpy ( char * destination, const char * source, size_t num );
```

strncpy 从 source 指向的空间中复制前 num 个字符到 destination 指向的空间中。如果还没有复制完 num 个字符时，source 已经达到了字符串尾部，那么函数将补充复制若干个 null 字符到 destination 指向的空间中，直到复制 num 个字符为止。

如果这是第一次碰到 size_t 类型，请别慌张，size_t 是无符号整型，等价于 unsigned int。后续很多函数都会涉及此类型。

strncpy 例子，代码如下：

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

int main ()
{
  char str1[]= "To be or not to be";
  char str2[40];
  char str3[40];

  /* copy to sized buffer (overflow safe): */
  strncpy ( str2, str1, sizeof(str2) );

  /* partial copy (only 5 chars): */
  strncpy ( str3, str2, 5 );
  str3[5] = '\0';    /* null character manually added */

  puts (str1);
  puts (str2);
  puts (str3);

  return 0;
}
```

输出：

```
To be or not to be
To be or not to be
To be
```

需要指出的是，在使用 strncpy 函数时，通常需要自行添加 null 终结符到 destination 中。

在 C++语言中可以直接使用 std::string 类处理各种字符串问题。有三种方法可以对字符串复制：std::string 的构造函数，操作符重载函数 operator=，以及成员函数 assign。

第一种方法：

```
char str1[]= "To be or not to be";
std::string str2 = str1;
std::string str3;
str3 = str1;
```

第二种方法：

```
char str1[]= "To be or not to be";
std::string str2;
str2.assign(str1, str1+strlen(str1));
```

如果只想复制若干个字符，还可以有以下第三种方法：

```
char str1[]= "To be or not to be";
std::string str2;
str2.assign(str1, str1+2);
```


assign 是 std::string 的类成员函数，三个重载函数声明如下：

```
string& assign (const char* s);
string& assign (const char* s, size_t n);
template <class InputIterator>
string& assign (InputIterator first, InputIterator last);
```

1.1.2 字符串连接

C 语言中可以使用 strcat 函数实现字符串的连接，也就是将一个字符串复制到另一个字符串的尾部，strcat 函数声明如下：

```
char * strcat ( char * destination, const char * source );
```

strcpy 会将 source 指向的字符串内容连同末尾的 null 字符一起复制到 destination 末端的内存空间中。在连接开始时，source 的第一个字符将会覆盖 destination 末端的 null 字符，在连接结束时，source 最后的 null 字符将会复制到 destination 的最后。

destination 指向的字符串空间必须能容纳 source，以及 destination 中所有字符数之和（包括 null 字符），并且 source 和 destination 在内存上不能存在重叠现象。函数将返回 destination 指针。

例如，代码如下：

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

int main ()
{
  char str[80];
  strcpy (str,"these ");
  strcat (str,"strings ");
  strcat (str,"are ");
  strcat (str,"concatenated.");
  puts (str);
  return 0;
}
```

输出：

```
these strings are concatenated.
```

一些公司在面试或笔试时也喜欢直接使用 strcat 的内部实现作为题目，以下是一个较为高效的实现过程，代码如下：

```
char *strcat(char *dest, const char *src)
{
    char *save = dest;
    while (*dest)
      dest++;
    while (*dest++ = *src++);
    return save;
}
```


在 C++中，可以直接使用 std::string 类成员函数 append 或者直接调用操作符重载的“+=”符号函数用于字符串的连接。

例如，使用 append 成员函数，代码如下：

```
#include <iostream>
#include <string>

int main ()
{
  std::string str;
  std::string str2="Writing ";
  std::string str3="print 10 and then 5 more";

  // used in the same order as described above:
  str.append(str2);                       // "Writing "
  str.append(str3,6,3);                   // "10 "
  str.append("dots are cool",5);          // "dots "
  str.append("here: ");                   // "here: "
```

```
    str.append(10u,'.');                    // ".........."
    str.append(str3.begin()+8,str3.end());    // " and then 5 more"
    str.append<int>(5,0x2E);                // "....."

    std::cout << str << '\n';
    return 0;
}
```

输出：

```
Writing 10 dots here: ... and then 5 more...
```

或者使用 operator +=成员函数，代码如下：

```
#include <iostream>
#include <string>

int main ()
{
    std::string name ("John");
    std::string family ("Smith");
    name += " K. ";          // c-string
    name += family;          // string
    name += '\n';            // character

    std::cout << name;
    return 0;
}
```

输出：

```
John K. Smith
```

当然，还可以使用全局的 operator +函数来处理 std::string 的连接，代码如下：

```
#include <iostream>
#include <string>

int main ()
{
    std::string firstlevel ("com");
    std::string secondlevel ("google");
    std::string scheme ("http://");
    std::string hostname;
    std::string url;

    hostname = "www." + secondlevel + '.' + firstlevel;
    url = scheme + hostname;
    std::cout << url << '\n';
```

```
    return 0;
}
```

输出：

```
http://www.google.com
```

1.1.3 反转字符串

反转字符串也是一些程序和算法需要的功能。在 Windows 下使用 C 时，可以使用 strrev 反转字符串，strrev 函数声明如下：

```
char * strrev( char *str );
```

例如，代码如下：

```
#include<stdio.h>
#include<string.h>
int main()
{
    char a[] = "my name";
    strrev(a);
    puts(a);
    return   0;
}
```


输出：

```
eman ym
```

需要说明的是，Linux 系统库中并没有包含这个函数，所以必须自己定义反转字符串函数。strrev 的一个简单的实现过程如下，代码如下：

```
#include <string.h>
char *strrev(char *str)
{
    int i = strlen(str) - 1, j = 0;
    char ch;
    while (i > j)
    {
        ch = str[i];
        str[i] = str[j];
        str[j] = ch;
        i--;
        j++;
    }
    return str;
}
```

利用指针和异或运算，可以高效实现字符串的反转，代码如下：

```
void strrev(char *p)
{
  char *q = p;
  while(q && *q) ++q;
  for(--q; p < q; ++p, --q)
    *p = *p ^ *q,
    *q = *p ^ *q,
    *p = *p ^ *q;
}
```

在C++中，则可以直接使用STL算法库中的std::reverse函数对字符串或者std::string对象进行反转。例如，代码如下：

```
#include <iostream>
#include <string> //for std::string
#include <algorithm> //for std::reverse
#include <cstring> //for strlen
using namespace std;

int main()
{
    std::string s = "Hello world!";
    std::reverse(s.begin(), s.end());
    std::cout<<s<<endl;

    char str[] = "get ready!";
    std::reverse(str, str+strlen(str));
    cout << str << endl;
    return 0;
}
```

输出：

```
!dlrow olleH
!ydaer teg
```

从上例中可以看出，std::reverse不仅可以对std::string对象进行反转，也可以对普通字符串进行反转。std::reverse是STL算法函数之一，所以必须引入头文件<algorithm>。

请大家检查如下字符串反转代码，程序运行时会崩溃，代码如下：

```
#include <iostream>
#include <algorithm> //for std::reverse
#include <cstring> //for strlen
using namespace std;

int main()
```

```
{
    char *str = "get ready!";
    std::reverse(str, str+strlen(str));
    cout << str << endl;
    return 0;
}
```

原因是 str 是一个字符指针，它指向的是一个字符串常量，而常量是无法修改的。

1.1.4 大小写转换

在 C 语言的库函数中，并没有提供对字符串大小写转换的函数。但库函数提供了 toupper 和 tolower 函数用于将单个字符的大小写转换。toupper 将字符转换成大写，tolower 将字符转换成小写。函数声明如下：

```
int toupper ( int c );
int tolower ( int c );
```

如果传入的参数 c 没有对应的大小写字符，那么返回值就是传入的 c 值。另外，这两个库函数声明在<ctype.h>中而不是在<string.h>中。

例如，代码如下：


```
#include <stdio.h>
#include <ctype.h>
int main ()
{
  int i=0;
  char str[]="Test String.\n";
  char c;
  while (str[i])
  {
    c=str[i];
    putchar (tolower(c));
    i++;
  }
  return 0;
}
```

输出：

```
test string.
```

可以利用字符大小写转换的函数，实现字符串的大小写转换，一个基本的小写转换方法如下，代码如下：

```
char* ToLower(char* str)
{
    int i;
```