

大道至简

(标准模板库)精解

The Greatest Truth is The Simplest —

Precision Expounding on C++ STL (Standard Template Library)

闫常友 编著

一杯清茶，一卷书，开卷有益。

—— C++ STL程序员的灯塔

A Cup of Tea, A Volume of Book.
Reading This Book Enriches Our Minds.

——The Beacon for C++ STL Programmer



大道至简

——C++ STL（标准模板库）精解

闫常友 编著



机械工业出版社

众所周知, C++ 是在 C 的基础上发展起来的。它进一步扩充和完善了 C 语言, 是一种面向对象的程序设计语言。经过几十年的发展, C++ 现已支持多种编程规范, 其相应的 C++ 国际标准也在不断更新。C++ STL 现在是在 C++ 标准程序库的一部分, 其作用是对常用数据结构和算法进行封装——标准化组件, 以便让程序员可以直接使用现成的算法和数据结构, 这样既避免了重复开发, 又提高了效率。

本书按照 C++ STL 的内容结构由浅入深地讲解了其应用开发技术。主要内容共计 17 章。第 1 章主要介绍了与 C++ STL 相关的基本概念和基础知识, 并简要介绍了本书后面会用到的一些模板类型。第 2~17 章依次介绍了字符串类模板、容器、算法库、迭代器、数值计算模板、输入/输出类模板、异常处理类模板、通用工具类模板、语言支持类模板、检测类模板、国际化类模板、仿函数、配置器、原子操作类、线性控制类模板以及正则表达式模板等内容。由于 C 标准函数库的相关内容在程序开发中的使用频率较高, 因此本书的附录部分给出了几乎所有传统 C 库函数中的数学计算类函数和数值转换类函数, 以供读者使用。

本书适合已有一定 C/C++ 基础的读者阅读, 也适合有意于深度钻研 C++ STL 技术的朋友们阅读。

图书在版编目 (CIP) 数据

大道至简: C++ STL (标准模板库) 精解/闫常友编著. —北京: 机械工业出版社, 2015. 5

ISBN 978-7-111-51399-5

I. ①大… II. ①闫… III. ①C 语言—程序设计 IV. ①TP312

中国版本图书馆 CIP 数据核字 (2015) 第 206435 号

机械工业出版社 (北京市百万庄大街 22 号 邮政编码 100037)

策划编辑: 申永刚 责任编辑: 申永刚 吴晋瑜

版式设计: 霍永明 责任校对: 张晓蓉 刘怡丹

封面设计: 马精明 责任印制: 乔 宇

保定市中国画美凯印刷有限公司印刷

2015 年 11 月第 1 版第 1 次印刷

184mm × 260mm · 46.75 印张 · 1 插页 · 1158 千字

0001—3000 册

标准书号: ISBN 978-7-111-51399-5

定价: 125.00 元

凡购本书, 如有缺页、倒页、脱页, 由本社发行部调换

电话服务

网络服务

服务咨询热线: 010-88361066

机工官网: www.cmpbook.com

读者购书热线: 010-68326294

机工官博: weibo.com/cmp1952

010-88379203

金书网: www.golden-book.com

编辑热线: 010-88379425

教育服务网: www.cmpedu.com

封面无防伪标均为盗版

前 言

本书详细介绍了C++标准模板库 (C++ Standard Template Library, C++ STL) 的所有内容。本书主要是参照 ISO/IEC 14882 Second edition 的内容编写的, 同时也参考了最新的C++ 11 国际标准 (ISO/IEC 14882: 2011)。

C++ 经历了多年的发展, 其新增特性较多, 而C++ STL 是众多成员辛勤劳动的结晶, 其中大量的可重复代码为广大程序员提供了巨大的方便, 节省了大量时间和人力。C++ STL 的开发者主要是 Alexander Stepanov、David Musser 以及 MengLee 三位大师, 其中 Alexander Stepanov 被誉为 “STL 之父”。1994 年 7 月, 美国国家标准学会 (ANSI) 通过投票决定, 将 STL 纳入C++ 标准, 使之成为C++ 库的重要组成部分。

C++ STL 作为C++ 的一部分, 历经多次修改, 经过多个程序员团队的 “精加工”, 已经不同于最初的版本。STL 为编程人员提供了诸多方便和好处, 以前用传统C++ 编写的复杂代码, 现在通过使用 STL, 仅仅几句话就可以实现。在 STL 中, 模板使用得可谓淋漓尽致。通过使用模板, 用户可获得优质并且高效的代码。STL 的优越性使其迅速流行起来, 并且发展劲头强劲。近些年, 国内的 STL 热也迅速升温, 但相关的资料并不丰富, 较好的资料更是少之又少。对程序员来说, 掌握 STL 编程技术, 并精通 C++ 高级编程技术, 是非常有必要的。

本书以广大程序员的角度, 详细介绍了C++ STL 标准库, 对其中的模板技术更是进行了细致深入的讲解。书中通过大量例题 (均由作者亲自编写或摘自 MSDN) 对各知识点进行实例讲解, 希望读者认真阅读。

鉴于作者水平有限, 书中难免存在不足之处, 敬请广大读者批评指正。

目 录

前言

第1章 预备知识及简介 1

1.1 基本概念	1
1.1.1 何谓“命名空间”	1
1.1.2 头文件	2
1.1.3 面向对象的程序设计	3
1.1.4 C++ 中的声明和定义	6
1.1.5 最简单的C++ 程序	8
1.1.6 指针	10
1.1.7 函数	12
1.1.8 文件	16
1.1.9 编译和链接	19
1.1.10 程序启动和终止	20
1.1.11 异常处理	20
1.1.12 预处理命令	21
1.1.13 宏	29
1.2 类模板定义	31
1.2.1 类模板实例化	32
1.2.2 类模板的成员函数	34
1.2.3 类模板的静态成员	34
1.3 成员模板	36
1.4 友元模板	37
1.5 函数模板	38
1.6 类模板的参数	42
1.7 STL 简介	45
1.7.1 STL 历史	45
1.7.2 STL 组件	46
1.7.3 STL 基本结构	46
1.7.4 STL 编程概述	49
1.8 小结	51

第2章 字符串 52

2.1 字符串类库简述	52
2.2 字符的特点	54
2.3 字符串类模板 (basic_string)	54
2.4 字符串通用操作	55

2.4.1 构造器和析构器	56
2.4.2 大小和容量	58
2.4.3 元素存取 (访问)	60
2.4.4 字符串比较	61
2.4.5 字符串内容的修改和替换	64
2.4.6 字符串联接	71
2.4.7 字符串 I/O 操作	71
2.4.8 字符串查找	72
2.4.9 字符串对迭代器的支持	76
2.4.10 字符串对配置器的支持	78
2.5 小结	79

第3章 容器——对象储存器 80

3.1 容器概念	80
3.2 序列式容器	83
3.2.1 vector (向量) 类模板	83
3.2.2 list (列表) 类模板	103
3.2.3 deque (双端队列) 类模板	124
3.3 关联式容器	133
3.3.1 set/multiset (集合) 类模板	133
3.3.2 map/multimap (图) 类模板	145
3.4 特殊容器用法	163
3.4.1 bitset (位集合) 类模板	163
3.4.2 stack (栈) 类模板	166
3.4.3 queue (队列) 类模板	169
3.4.4 priority_queues (优先队列) 类模板	172
3.5 小结	175

第4章 STL 算法 176

4.1 算法库简介	176
4.2 非修改性算法	177
4.2.1 for_each() 算法	177
4.2.2 元素计数算法	181
4.2.3 最小值和最大值算法	183
4.2.4 搜索算法	184
4.2.5 比较算法	193

4.3 修改性算法	197	6.1.4 复数类运算	251
4.3.1 复制	198	6.1.5 复数的超越函数运算	254
4.3.2 转换	200	6.2 数组(向量)运算	256
4.3.3 互换	204	6.2.1 类 valarray	257
4.3.4 赋值	205	6.2.2 数组子集类——类 slice 和类 模板 slice_array	264
4.3.5 替换	207	6.2.3 类 gslice 和类模板 gslice_ array	266
4.3.6 逆转	208	6.2.4 类 mask_array	270
4.3.7 旋转	210	6.2.5 类 indirect_array	271
4.3.8 排列	211	6.3 通用数值计算	273
4.4 排序及相关操作算法	215	6.3.1 求和算法 accumulate()	273
4.4.1 全部元素排序	215	6.3.2 内积算法 inner_product()	274
4.4.2 局部排序	217	6.3.3 部分和算法 partial_sum()	276
4.4.3 根据某个元素排序	219	6.3.4 序列相邻差算法 adjacent_ difference()	277
4.4.4 堆(Heap)操作算法	221	6.4 全局性数学函数	279
4.4.5 容器合并、交集和差集算法	223	6.5 小结	281
4.4.6 搜索算法	227	第7章 输入/输出类模板	282
4.5 删除算法	229	7.1 IOStream 简介	282
4.6 小结	232	7.1.1 stream 对象	282
第5章 迭代器——访问容器的接口	233	7.1.2 stream 类别	283
5.1 迭代器及其特性	233	7.1.3 stream 操作符	284
5.2 头文件 <iterator>	234	7.1.4 操控器	284
5.3 迭代器类型详述	234	7.2 IOStream 基本类和标准 IOStream 对象	285
5.3.1 输入型迭代器	234	7.2.1 头文件	285
5.3.2 输出型迭代器	235	7.2.2 标准 stream 操作符	285
5.3.3 前向型迭代器	235	7.2.3 stream 状态	289
5.3.4 双向型迭代器	236	7.2.4 标准输入和输出函数	293
5.3.5 随机访问型迭代器	236	7.3 格式化	298
5.3.6 vector 迭代器的递增和递减	237	7.3.1 格式标识	298
5.4 迭代器配接器	237	7.3.2 bool 类型数据的格式控制	300
5.4.1 逆向型迭代器	237	7.3.3 详解“字段宽度、填充字符和 位置调整”	300
5.4.2 插入型迭代器	239	7.3.4 正记号与大写字符	303
5.4.3 流型迭代器	241	7.3.5 数值进制	304
5.5 迭代器辅助函数	244	7.3.6 浮点数输出	305
5.5.1 前进 advance() 函数	244	7.3.7 一般性格式定义	306
5.5.2 距离 distance() 函数	245	7.4 类 streambuf	307
5.5.3 交换两个迭代器所指内容 iter_swap() 函数	246	7.4.1 流缓冲区	307
5.6 小结	248	7.4.2 缓冲区迭代器	309
第6章 数值计算类模板	249	7.4.3 自定义缓冲区	311
6.1 复数运算	249		
6.1.1 一个复数运算例题	249		
6.1.2 复数类成员函数	250		
6.1.3 复数类运算符	251		

7.5 基于字符串的流	318
7.5.1 streambuf 类	318
7.5.2 类模板 basic_istream	319
7.5.3 类模板 basic_ostringstream	320
7.5.4 类模板 basic_stringstream	321
7.6 基于文件的流	321
7.6.1 文件标识及其使用	322
7.6.2 随机访问	332
7.6.3 4 个类模板	336
7.6.4 C 库中的文件存取功能概述	338
7.7 小结	340
第 8 章 异常处理类模板	341
8.1 异常的概念和基本思想	341
8.1.1 异常的概念	341
8.1.2 异常的分类	342
8.1.3 异常的捕捉和处理	344
8.1.4 资源管理	346
8.1.5 异常和效率	348
8.1.6 异常的描述	349
8.1.7 未捕捉的异常	352
8.2 异常类及几个重要问题	353
8.2.1 类 exception	353
8.2.2 调用 abort ()	360
8.2.3 堆栈解退	362
8.2.4 错误代码	362
8.2.5 异常的迷失	363
8.2.6 异常处理的局限性	367
8.3 处理异常详述	369
8.4 异常的特殊处理函数	372
8.5 小结	373
第 9 章 通用工具类模板 (Utility)	374
9.1 通用工具库简介	374
9.1.1 相等比较	374
9.1.2 小于比较	374
9.1.3 复制构造	377
9.1.4 配置器要求	377
9.1.5 运算符	379
9.1.6 对组 (Pairs)	379
9.2 动态内存管理	385
9.2.1 默认配置器	385
9.2.2 raw_storage_iterator	387
9.2.3 临时缓冲区 (Temporary Buffers)	387
9.2.4 特定算法	388
9.2.5 C 函数库中的内存管理函数	389
9.3 堆的内存分配	389
9.3.1 new 和 delete 运算符	389
9.3.2 分配固定维数的数组	390
9.3.3 分配动态内存数组	390
9.3.4 处理堆耗尽	391
9.4 辅助功能	391
9.4.1 数值极限	392
9.4.2 较大/较小值 (最大/最小值) ...	395
9.4.3 两值交换	396
9.4.4 辅助性比较	398
9.4.5 头文件 cstdlib 和 cstdint 简介	399
9.5 日期和时间	400
9.5.1 3 个类型	400
9.5.2 结构体 (tm)	400
9.5.3 相关时间函数	401
9.5.4 时间示例	403
9.6 模板类 auto_ptr	405
9.6.1 auto_ptr 类构造函数	406
9.6.2 类 auto_ptr 的成员及转换	407
9.6.3 使用类 auto_ptr	407
9.7 小结	411
第 10 章 语言支持类模板	412
10.1 类型	412
10.2 执行属性	412
10.2.1 类模板 numeric_limits 及其 成员	412
10.2.2 float_round_style 和 float_ denorm_style	416
10.2.3 数值极限的特殊化	417
10.2.4 C 库函数	418
10.2.5 应用举例	418
10.3 程序的启动和终止	426
10.4 动态内存管理	427
10.4.1 内存的分配和释放	428
10.4.2 内存分配错误	431
10.4.3 应用举例	431
10.5 类型标识符	434
10.5.1 类 type_info	434
10.5.2 类 bad_cast	435
10.5.3 类 bad_typeid	436
10.5.4 操作符 typeid	436

10.5.5 操作符 <code>dynamic_cast</code>	437	12.4.4 模板类 <code>collate</code>	506
10.5.6 应用举例	437	12.4.5 类 <code>time</code>	508
10.6 异常处理	439	12.4.6 模板类 <code>monetary</code>	515
10.6.1 类 <code>exception</code>	439	12.4.7 类 <code>message retrieval</code>	523
10.6.2 特殊异常处理	440	12.4.8 Program-defined facets	526
10.6.3 异常终止	441	12.4.9 C 库 <code>locale</code>	527
10.6.4 未捕获异常 (<code>uncaught_</code> <code>exception</code>)	441	12.5 细述使用刻面	528
10.6.5 应用举例	442	12.5.1 数值的格式化	528
10.7 其他运行支持	444	12.5.2 时间/日期的格式化	530
10.7.1 概述	444	12.5.3 货币符号的格式化	533
10.7.2 应用举例	445	12.5.4 字符的分类和转换	536
10.8 小结	448	12.5.5 字符串校勘	541
第 11 章 检测类模板详解	449	12.5.6 信息国际化	542
11.1 异常类	449	12.6 小结	543
11.1.1 类 <code>logic_error</code>	449	第 13 章 仿函数	544
11.1.2 类 <code>domain_error</code>	450	13.1 仿函数的概念	544
11.1.3 类 <code>invalid_argument</code>	451	13.1.1 仿函数的概念	544
11.1.4 类 <code>length_error</code>	452	13.1.2 仿函数的作用	545
11.1.5 类 <code>out_of_range</code>	454	13.2 预定义仿函数	553
11.1.6 类 <code>runtime_error</code>	455	13.3 辅助用仿函数	554
11.1.7 类 <code>range_error</code>	456	13.3.1 一元组合函数配接器	554
11.1.8 类 <code>overflow_error</code>	457	13.3.2 二元组合函数配接器	557
11.1.9 类 <code>underflow_error</code>	458	13.4 关系仿函数	558
11.2 断言	459	13.4.1 等于 (<code>equal_to</code> <code><type>()</code>)	558
11.3 错误编码	461	13.4.2 不等于 (<code>not_equal_to</code> <code><type>()</code>)	559
11.4 小结	462	13.4.3 小于 (<code>less<type>()</code>)	560
第 12 章 国际化库详解	463	13.4.4 大于 (<code>greater<type>()</code>)	561
12.1 国际化元素	464	13.4.5 大于等于 (<code>greater_equal</code> 和小于等于 (<code>less_equal</code>)	561
12.2 多种字符编码	464	13.5 逻辑仿函数	562
12.2.1 宽字符和多字节文本	464	13.5.1 谓词	562
12.2.2 字符特性	465	13.5.2 逻辑仿函数	563
12.2.3 特殊字符国际化	467	13.6 算术仿函数	567
12.3 类 <code>locale</code>	467	13.6.1 加、减、乘、除仿函数	568
12.3.1 类 <code>locale</code> 概述	467	13.6.2 求余仿函数和求反仿函数	569
12.3.2 类 <code>locale</code> 的 facet	470	13.7 其他类型的仿函数	571
12.3.3 区域表示和混合区域表示	473	13.7.1 证和映射	571
12.3.4 流和区域	477	13.7.2 仿函数 <code>hash</code> 和 <code>subtractive_</code> <code>rng</code>	574
12.3.5 刻面的处理	477	13.8 适配器	574
12.4 标准 <code>locale</code> 的分类	479	13.8.1 成员函数适配器	575
12.4.1 类 <code>ctype</code>	480		
12.4.2 数值类的类 <code>locale</code>	496		
12.4.3 刻面 <code>numeric punctuation</code>	503		

13.8.2 其他适配器	581	16.3.2 lock 模板类	639
13.9 小结	590	16.3.3 call_once	646
第14章 配置器	591	16.4 条件变量	647
14.1 使用配置器	591	16.4.1 类 condition_variable	648
14.2 C++ STL 默认的配置器 (标准配置器)	593	16.4.2 类 condition_variable_any	655
14.3 自定义配置器	594	16.5 模板类 future	659
14.4 配置类的详细讨论	595	16.5.1 模板类 future_error、future_errc 和 future_category 以及共享 状态	659
14.4.1 型别	595	16.5.2 模板类 promise	662
14.4.2 配置类的成员函数	595	16.5.3 模板类 future	666
14.4.3 广义配置器	596	16.5.4 模板类 shared_future	670
14.4.4 动态存储	597	16.5.5 仿函数 async	671
14.4.5 C 风格的分配	597	16.5.6 模板类 packaged_task	673
14.5 未初始化的内存	598	16.6 小结	677
14.6 配置器示例	600	第17章 正则表达式	678
第15章 原子运行库模板	602	17.1 定义及要求	678
15.1 头文件 <atomic> 简介	602	17.2 类模板 basic_regex	679
15.1.1 无锁属性	602	17.2.1 类模板 basic_regex 的声明	679
15.1.2 3 个模板	602	17.2.2 名称空间 std:: regex_constants	685
15.1.3 原子模板的常规操作	603	17.2.3 类 regex_error	687
15.1.4 头文件中的模板函数及算术 运算函数	605	17.2.4 类模板 regex_traits	688
15.1.5 原子类型 atomic_flag	609	17.2.5 类 basic_regex 的使用	692
15.2 顺序及一致性	610	17.3 类模板 sub_match 和 match_results	696
15.3 原子类型	611	17.3.1 类模板 sub_match	696
15.3.1 模板类 atomic	614	17.3.2 类模板 match_results	698
15.3.2 针对整型数据的特殊化模板	616	17.4 正则表达式相关的 3 种算法	704
15.3.3 针对指针的特殊化模板	619	17.4.1 正则匹配算法 regex_match	704
15.4 小结	621	17.4.2 正则搜索算法 regex_search	707
第16章 线程控制类模板	622	17.4.3 正则替换算法 regex_replace	708
16.1 要求和性能	622	17.5 正则表达式的迭代器	711
16.1.1 异常	622	17.5.1 迭代器 regex_iterator	711
16.1.2 本地句柄	622	17.5.2 迭代器 regex_token_iterator	713
16.1.3 时序规定	622	17.6 小结	719
16.1.4 可锁定类型	623	附录 部分 C 函数库详解	720
16.2 线程类	623	附录 A 数学函数	720
16.2.1 线程类成员变量 id	624	A.1 数学函数库中的宏	720
16.2.2 线程类成员函数	624	A.2 浮点计算减法协议开关	721
16.2.3 命名空间 this_thread	625	A.3 数学库中的宏函数	721
16.2.4 线程示例	626	A.4 三角函数和反三角函数	722
16.3 互斥	629	A.5 指数和对数函数	724
16.3.1 mutex 模板类	630	A.6 幂函数和绝对值函数	726

A. 7	误差和 gamma 函数	727	B. 1	字符转整数函数 (atoi () 和 atol ())	731
A. 8	近似取整函数	728	B. 2	字符型转换浮点型函数 (atof () 和 atol ())	732
A. 9	求余函数	729	B. 3	整型数转字符串函数 (itoa (), ltoa () 和 ultoa ())	732
A. 10	操作处理函数	730	B. 4	浮点数转换字符串函数	734
A. 11	最大值、最小值和正差函数	730			
A. 12	浮点乘加函数	730			
A. 13	比较函数 (宏)	731			
附录 B	数据类型转换	731			

第 1 章

预备知识及简介

模板是C++的一个新特性。在现今的C++标准模板库中，几乎所有东西都被设计为模板形式。如果不支持模板，就无法使用标准模板库。模板是实现代码重用的一种工具，即针对一个或多个尚未明确的类型，编写一套函数或类型，以供用户使用。通过使用模板，C++允许推迟对某些类型的选择，直到需要对模板进行处理时才使用。使用模板，还可以使程序员针对不同的相似特性开发出更加大众化的代码。

本章首先介绍了C/C++语言中常用的一些基本概念；其次介绍了类模板定义、成员模板、函数模板、友元模板及类模板的参数等内容；最后介绍了模板库的相关知识。

1.1 基本概念

本节主要介绍C/C++语言中常用的一些基本概念。通过学习本节内容，读者应能正确使用这些概念，以便更加方便、高效地开发代码。

1.1.1 何谓“命名空间”

命名空间（Namespace）是指标识符的可见范围或者有效范围。定义命名空间的目的是为了防止出现名称冲突现象。命名空间是C++中一个较新的特性。为了将多个程序员开发的代码便捷、高效地组合起来，防止出现重复的函数名、类名等，命名空间将不同的代码封装在自己的有效范围内。

命名空间需要使用using来声明，并且每个命名均需要使用using。标准模板库（Standard Template Library, STL）采用命名空间技术解决了大规模软件开发的难题。C++ STL内的所有标识符都被定义为一个名为std的命名空间中，因而可以直接使用指定的标志符号std。例如：

```
std::cout << 2.0 << std::endl;
```

如果在上述语句之前添加以下声明：

```
using std::cout;
```

```
using std::endl;
```

那么上述语句可以写为：

```
cout << 2.0 << endl;
```

如果嫌麻烦，需要逐个声明名称，可以直接声明、使用命名空间std，例如：

```
using namespace std;
```

```
...
```

```
cout << 2.0 << endl;
```

显而易见,使用C++标准库的命名空间 `std` 之后,可以任意使用标准库中的函数和变量。参见例 1-1。

■ 例 1-1

```
#include <iostream>
#include <stdio.h>
using namespace std;
void main()
{
    cout << "欢迎您开始步入C++ 标准模板库!" << endl;           //输出“欢迎您开始步入C++ 标准模板库!”
    getch();                                                         //等待输入任意键,之后程序退出
    return;
}
```

 **提示** 在例 1-1 中,使用了命名空间 `std`,即在 `main()` 函数前添加如下语句:
`using namespace std;`

之后在 `main()` 函数中,可以直接使用 `cout` 函数和 `endl` 函数。语句 `using namespace std` 可以放在源代码的任意位置,不同的位置代表 `std` 不同的有效范围。

 **总结** 本小节主要讲述了命名空间的概念及其最简单的使用方法,并通过例 1-1 使读者对命名空间有一个直接的认识。

1.1.2 头文件

在上一节的例 1-1 中,有 `#include <iostream>` 这样的语句,其中 `iostream` 就是被 `include` 语句包含的头文件。在原来的 C 语言中,头文件都是以 `.h` 的形式存在的;而在 C++ 中,头文件只有名字,没有扩展名,例如 `iostream`。这部分头文件均在原有名字前添加字符 `c` 来加以标志,例如 `cmath`、`cstring` 等。这类头文件的使用方法如下所示:

```
#include <cmath>
#include <cstring>
```

值得注意的是:在书写头文件名时,一定要认真,不要添加其他字符,例如空格。如果添加了其他字符,有些编译程序会提示无法找到该头文件 (Visual C++ 6.0 编译程序可以避免类似的问题)。

头文件的功能主要是将原程序片段收集到一起,形成一个提供给编译程序的文件。一般情况下,头文件中只包含各种声明、常量定义、预编译、注释、类型定义、模板定义等。常规的函数定义、数据定义、导出的模板定义等,不能出现在头文件中。

C++ 标准模板库头文件是标准模板库的外在表现形式。使用标准模板库的唯一途径就是包含相应的头文件。标准模板库头文件是没有后缀的。

 **提示** 请关注头文件 `cmath` 的使用,并注意例 1-2 中的中文注释。

例 1-2

```
#include <iostream>
#include <cmath>           //使用头文件 cmath
#include <stdio.h>
using namespace std;      //使用命名空间 std
void main()
{
    float pi = 3.1415916;
    float radius = 0;
    float area = 0.0;
    float girth = 0.0;
    cout << "请输入圆的半径:" << endl;
    cin >> radius;
    area = pi * pow(radius, 2); //使用幂函数 pow(); 乘方
    girth = 2 * pi * radius;
    cout << "area = " << area << "; girth = " << girth << endl;

    getchar();             //等待程序退出
    return;
}
```



提示 在例 1-2 中, 使用了 C 语言幂函数 `pow (double x, double y)`。这个函数有两个参数 `x` 和 `y`, 函数的功能是求解 `x` 的 `y` 次方。

标准模板库提供了大量的头文件, 使用头文件主要是为了提供类、函数、变量的声明, 以供用户方便地使用这些类、函数、变量等。

1.1.3 面向对象的程序设计

面向对象 (Object Oriented, OO) 是 C/C++ 语言解决问题的一种方法。之前的程序开发是面向过程的。面向过程的程序设计主要考虑解决问题的先后顺序、措施安排等, 具有典型的过程性。而面向对象的程序设计主要是建立在各种对象基础上的软件开发, 每一个具有独立特性和相应功能的个体均可以作为一个对象来加以处理。

人类社会历史充满了“抽象”的概念, 例如天、地、道等。

在软件开发和程序设计领域, 同样存在诸多“抽象”的对象。根据软件开发的需要, 每个对象被赋予不同的属性和方法 (函数)。当该对象被定义时, 用户同样可以方便地使用该对象的属性和方法。大多数程序员都曾学过 C++ 中类的概念。类的对象一般有构造函数、析构函数等。类的对象可以被复制、被传递或被作为其他类的成员, 甚至有些类的成员还可以直接参加数学计算。

面向对象是当前计算机界关心的重点, 它是 20 世纪 90 年代软件开发方法的主流。在对对象有了简单认识之后, 广大程序员开始就面向对象的程序设计进行各种尝试和研究, 历经多年的发展和变革, 才有了今天面向对象软件开发技术的现状。如今, 面向对象的概念和应

用已超越了程序设计和软件开发,并扩展到了各个领域。

C++ 类是一种将抽象类型转换为用户自定义类型的方法。该方法将数据表示和操纵数据的方法组合成一个整齐的包。对于类的实现,一般包括两方面:类的声明和类方法的定义。类的声明是指以数据成员的方式描述数据部分,以成员函数的形式描述公有接口。类方法的定义是指描述如何实现类的成员函数。通俗来说,类的声明提供了类的宏观概貌,而类的定义则提供了细节。下面举一个最简单的例子。



提示 请注意例 1-3 中的中文注释。类 Calculator 作为一个简单的计算器,用于实现一些最简单的数学计算功能。

例 1-3

```
#include <iostream> //包含输入/输出流功能的头文件
#include <cmath> //包含数学函数库的头文件
#include <stdio.h> //包含标准输入/输出的头文件
using namespace std; //使用命名空间 std
class Calculator //声明类 Calculator
{
public: //将下述定义为公共成员
    Calculator(); //构造函数
    ~ Calculator(); //析构函数
    float x; //成员变量 x
    float y; //成员变量 y
    float Adder(float x, float y); //加法运算
    float Substration(float x, float y); //减法运算
    float Multiplication(float x, float y); //乘法运算
    float Division(float x, float y); //除法运算
    float CPow(float x, float y); //乘方运算
    float CSqrt(float x); //开方运算
}; //类声明结束
Calculator::Calculator() //定义构造函数
{
    x=0;y=0;
}
Calculator::~~Calculator() //定义析构函数
{
}
float Calculator::Adder(float x, float y) //定义实现加法运算的成员函数
{
    float He=0;
    He=x+y;
    return He;
}
float Calculator::Substration(float x, float y) //定义实现减法运算的成员函数
{
    float Cha=0;
```



```
    Cha = x - y;
    return Cha;
}

float Calculator::Multiplication(float x, float y)           //定义实现乘法运算的成员函数
{
    float Ji = 0;
    Ji = x * y;
    return Ji;
}

float Calculator::Division(float x, float y)                //定义实现除法运算的成员函数
{
    float Shang = 0;
    if (y == 0.0)
        return -1;
    Shang = x / y;
    return Shang;
}

float Calculator::CPow(float x, float y)                   //定义实现乘方运算的成员函数
{
    float ChengFang = 0;
    ChengFang = pow(x, y);
    return ChengFang;
}

float Calculator::CSqrt(float x)                           //定义实现开平方运算的成员函数
{
    float sqrtC = 0.0;
    sqrtC = sqrt(x);
    return sqrtC;
}

void main()                                                 //程序入口, main() 函数
{
    float x = 0;
    float y = 0;
    cout << "请输入 x:" ;
    cin >> x;                                               //输入数据 x
    cout << endl;
    cout << "请输入 y:" ;
    cin >> y;                                               //输入数据 y
    cout << endl;
    Calculator my;                                           //定义类的对象
    float He = my.Adder(x, y);
    cout << "两数之和:" << He << endl;
    float Cha = my.Substration(x, y);
    cout << "两数之差:" << Cha << endl;
    float Ji = my.Multiplication(x, y);
    cout << "两数之积:" << Ji << endl;
    float Shang = my.Division(x, y);
    cout << "两数之商:" << Shang << endl;
    float Pow = my.CPow(x, y);
    cout << "x 的 y 次方:" << Pow << endl;
```

```
float KaiFang = my.CSqrt(x);  
cout << "x 的平方根:" << KaiFang << endl;  
cout << "任意键程序退出";  
getchar();  
}
```

程序执行后的效果如图 1-1 所示。

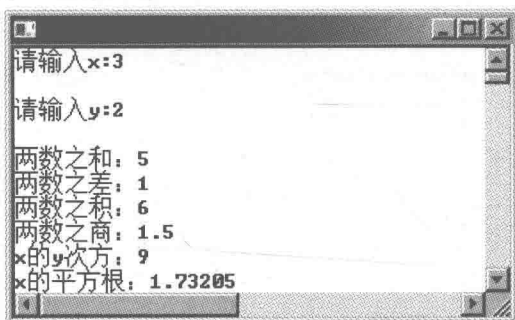


图 1-1 例 1-3 执行效果图



总结 在例 1-3 中，通过声明、定义一个类 Calculator，说明了如何使用类来进行面向对象的编程。通过定义类，形成一个具有一定功能的通用类型或通用模板；使用这个类的对象时，可以调用模板中的所有成员函数。面向对象技术要高于面向过程技术。面向对象的程序开发是给每个对象具备了一定的特殊“能力”，而这些“能力”一般也是通过面向过程的技术实现的。通过本小节的学习，主要理解“面向对象”的概念就可以了。

1.1.4 C++ 中的声明和定义

在程序开发领域，声明是指当一个计算机程序需要调用内存空间时，对内存发出的“占位”指令。定义则是指将声明的变量、函数、类等呈现或描述出来，为之提供一个意义相当的表达，并表明其与众不同之处，最终展现其内涵。

在软件开发过程中，常见的声明和定义语句列举如下（注意中文注释语句）。

例如：

```
int a; //声明一个整型变量 a  
float b; //声明一个浮点型变量 b  
struct S //声明一个结构 S  
{  
    int a;  
    float b;  
};  
extern int x; //声明一个外部变量 x  
void count (bool YN, int& counter); //声明一个函数 count
```

上述几行语句均是典型的声明语句。

```
void count (bool YN, int& counter) //函数 count 的定义
{
    if (YN) //如果 YN 是 true
    {
        counter ++; //计数器加 1
    }
}
```

上述几行代码对函数 count (bool YN, int& counter) 进行定义,用以实现函数的功能。下面通过例 1-4 来验证上述函数的定义。

例 1-4

```
#include <iostream>
using namespace std;
void count (bool YN, int& counter); //函数声明
void count (bool YN, int& counter) //函数定义
{
    if (YN)
    {
        counter ++;
    }
}

void main() //main() 主函数
{
    int counter=0; //计数器
    bool YN=1; //逻辑变量,用于当条件满足时,退出 while 循环
    while (YN)
    {
        count (YN,counter); //计算循环次数
        cout << counter << endl; //输出次数至屏幕
        if (counter > 5) //判断循环次数是否大于 5 次
        {
            YN=0; //如果循环次数超过 5 次,将逻辑变量 YN 置 0
        }
    }
}
```

例 1-4 的执行效果如图 1-2 所示。

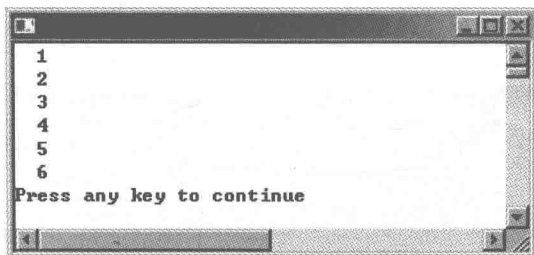


图 1-2 例 1-4 的执行效果