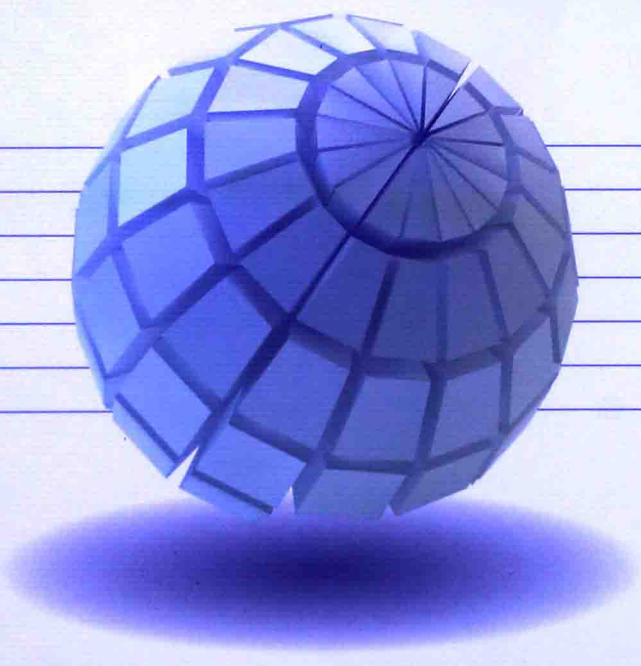
北极光

信盈达技术创新系列图书

嵌入式 C++ 实战教程

深圳信盈达电子有限公司　陈志发　周中孝◎编 著



電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

信盈达技术创新系列图书

嵌入式 C++ 实战教程

深圳信盈达电子有限公司　陈志发　周中孝　编著

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

内 容 简 介

本书是配合C++程序设计的指导教材，独立于任何C++/Visual C++教科书，重点放在C/C++语言的基本语法部分，尤其是类和对象，运算符重载，继承，多态这些内容。本书涵盖了C/C++所有的基础语法知识，并且讲解清晰易懂，内容详尽。从零开始，通过详细的示例，由浅入深、循序渐进地指导初学者掌握C++这门大型编程语言，培养实际分析问题和编程的能力，提高读者和学生的综合素质。

本书适合作为高等院校计算机专业的C++语言程序设计教科书，也可为广大编程人员提供参考。

图书在版编目（CIP）数据

嵌入式C++实战教程 / 陈志发，周中孝编著. —北京：电子工业出版社，2015.1
（信盈达技术创新系列图书）
ISBN 978-7-121-23023-3

Ⅰ. ①嵌… Ⅱ. ①陈… ②周… Ⅲ. ①C语言－程序设计－教材 Ⅳ. ①TP312

中国版本图书馆CIP数据核字（2014）第080409号

策划编辑：李树林
责任编辑：李树林
印　　刷：北京京科印刷有限公司
装　　订：三河市鹏成印业有限公司
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：21.25　字数：544千字
版　　次：2015年1月第1版
印　　次：2015年1月第1次印刷
印　　数：3 000册　　定价：59.80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。

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。

服务热线：（010）88258888。

前言

C++不仅是一种很重要的高级编程语言，而且代表了一种编程思想。它的思想已经被其他编程语言继承并发扬光大。现代每种新语言的诞生，都可以找到 C++的影子。所以说，若精通了 C++，再学习别的语言就很容易了，比如 Java、C#等。

C++在目前使用非常广泛，很多大型的程序都是用 C++写出来的，学好 C++编程语言是很有用的。但是很多初学者学习过 C++语言后认为 C++很难学，学习了好几本书，有不少地方还是不理解。C++难道真的这么难吗？学习 C++真的需要阅读这么多教材和资料吗？

C++其实不难学，那是由于好多书籍、资料的知识点组织结构和讲解方式等不够合理，无意中增加了初学者学习 C++的难度。无论是谁，学习一种新的知识，好的教材是非常重要的。讲解清晰易懂、内容科学合理的教材有助于初学者迅速掌握知识体系和精髓，在学习时间相同的情况下，学习效果会更好。

现在市面上一些 C++书籍不分主次轻重，比如，在初学者根本不知道模板是什么的时候，该书却对 STL 过早地讲解。而一些相对简单的基础概念，却放到后面，进而影响前面其他基础语法知识的学习，这也违背了先易后难的原则。而且表述语言过于专业化，专业术语太多，对于普通知识点的讲解也写得复杂深奥，非常不直观。这样的后果是初学者在按照那本书学习 C++的时候，需要不断前后跳跃式阅读，就像在查字典，不但花费很多时间，而且学习效果也不好，人为增加了学习 C++的难度。对于初学者来说，这样的字典式图书是不合适的，他们需要一本循序渐进、快速、扎实讲解 C++语言的书。

本书是以初学者最易懂的方式来阐述 C++知识的，能让从未学习过编程语言的初学者也能成为高手；同时讲解深刻细致，能让专业 C++程序员阅读本书后仍然有质的飞跃。本书是我在百忙的工作之中抽出大量业余时间完成的，其中的辛苦自然不用多说。但是让我感到安慰的是，本书确实能让学习 C++的初学者少走弯路，并迅速提高。

本书从一个最简单的 C++程序讲起，然后通过这个程序引出一系列相关知识，让初学者循序渐进地学习。同时书中的示例程序都是经过精心设计的。本书特点是实用性强，章节安排合理，清晰易懂，重点突出，深入浅出。相信读者阅读本书后，一定会有很大的提升，能够达到短期内掌握 C++语言的效果。

本书的出版，离不开深圳信盈达电子有限公司所有同事们的支持和帮助，在此向他们表示衷心的感谢。另外，感谢我的父母、亲人和朋友，是他们给予我精神上的支持和鼓励。感谢电子工业出版社，是他们认真专业的审核，让本书由粗糙的初稿变成了精美的图书。

由于时间仓促，编著者水平有限，书中可能有不恰当的地方，希望广大读者批评指正，联系邮箱：niusdw@163.com，欢迎来信交流。

陈志发

2014 年 10 月

目录

第1章 初识C++

1.1 C++简介

C++这个词在中国大陆的程序员圈子中通常被读作“C 加加”，而西方的程序员通常读作“C Plus Plus”，它的前身是 C 语言。C++是在 C 语言的基础上开发的一种集面向对象编程、泛型编程和过程化编程于一体的编程。1980 年，美国贝尔实验室的 Bjarne Stroustrup 博士及其同事在 C 语言的基础上，从 Simula67 中引入面向对象的特征，开发出一种将过程性与对象性相结合的程序设计语言，最初被称为“带类的 C”，1983 年取名为 C++。 此后，C++经过了许多次改进、完善，发展成为现在的 C++。目前的 C++具有两方面的特点：其一，C++是 C 语言的超集，因此它能与 C 语言兼容；其二，C++支持面向对象的程序设计，这使它被称为一种真正意义上的面向对象程序设计语言。 C++支持面向对象的程序设计方法，特别适合中型和大型的软件开发项目。从开发时间、费用，到软件的重用性、可扩充性、可维护性和可靠性等方面，C++均具有很大的优越性。

1.2 C++的发展过程

C++语言发展大概可以分为三个阶段。第一阶段是从 20 世纪 80 年代到 1995 年，这一阶段的 C++语言基本上是传统类型上的面向对象语言，并且凭借着接近 C 语言的效率，在工业界使用的开发语言中占据了相当大份额。第二阶段是从 1995 年到 2000 年，这一阶段由于标准模板库（STL）和 Boost 等程序库的出现，泛型程序设计在 C++中占据了越来越多的比重。当然，同时由于 Java、C#等语言的出现和硬件价格的大规模下降，C++受到了一定的冲击。第三阶段是从 2000 年至今，由于以 Loki、MPL 等程序库为代表的产生式编程和模板元编程的出现，C++出现了发展历史上又一个新的高峰，这些新技术的出现以及和原有技术的融合，使 C++已经成为当今主流程序设计语言中最复杂的一员。

1.3 C++和 C 的区别以及 C++新增特性

1.3.1 C 和 C++的区别

对于初学者来说，仅需要明白：C++是在扩充了 C 面向对象过程功能的基础上，又增加了面向对象的功能。下面列举了二者的一些区别。

从机制上：C 是面向过程的（但 C 也可以编写面向对象的程序）；C++是面向对象的，提供了类。但是用 C++编写面向对象的程序比 C 容易。

从适用的方向：C 适合要求代码体积小、效率高的场合，如嵌入式；C++适合更上层、复杂的场合。Linux 核心大部分是用 C 写的，因为它是系统软件，要求极高的效率。

从名称上也可以看出，C++比 C 多了两个“+”，说明 C++是 C 的超集。那为什么不叫 C+而叫 C++呢？这是因为 C++比 C 扩充的东西太多了，所以就在 C 后面放上两个“+”，于是就成了 C++。

C 语言是结构化编程语言，C++是面向对象的编程语言。

C++侧重于对象而不是过程，侧重类的设计而不是逻辑的设计。

C++是从 C 语言派生的，它与 C 语言是兼容的。

C++在 C 语言基础上新增了一些保留字：class、friend、virtual、inline、private、public、protected、const、this、string；也新增了一些运算符：new、delete、operator::。

1.3.2 C++新增特性

C++新增特性包括：

- 引用；
- const 常量定义；
- 函数的默认参数；
- 内联函数；
- 函数重载；
- 强制类型转换；
- 输入/输出流。

1. 引用

“引用”就是“变量”的别名，对引用的操作与对变量直接操作的作用是完全一样的。比如一个人有个正式名字叫张三，还有一个小名叫小红。那么她和朋友在一起的时候，朋友叫她张三或叫她小红，都是指向同一个人。

引用的声明：数据类型 & 引用名 = 初始值（初始值是变量名）。

引用在定义时必须初始化。注意此处的“&”并不是取地址符，而是“引用说明符”。声明一个引用并不是定义了一个新的变量，只表示给变量取了一个别名，不能再把该引用名作为其他变量名的别名。编译器会给它分配内存空间，因此引用本身占据存储单元，但是引用表现出来给用户看到的不是引用自身的地址，而是目标变量的地址，也就是说对引用取地址就是目标变量的内存地址。

C++的函数允许利用引用进行参数传递，具有高效性和安全性。

引用作为函数参数的特点如下：

（1）在进行实参和形参的结合时，不会为形参分配内存空间，而是将形参作为实参的一个别名。使用引用传递函数的参数，在内存中并没有产生实参的副本，它是直接对实参操作；而使用一般变量传递函数的参数，当发生函数调用时，需要给形参分配存储单元，形参

变量是实参变量的副本。因此，当参数传递的数据较大时，用引用比用一般变量传递参数的效率高，且所占空间小。

（2）用引用能达到用指针传递一样的效果，则函数内对形参的操作相当于直接对实参的操作，即形参的变化会影响实参。引用相对指针的优点如下：利用指针传递时，在被调用函数中同样要给形参分配存储单元，且需要重复使用“*指针变量名”的形式进行运算，这很容易产生错误且程序的阅读性较差。另一方面，在主调函数的调用点处，必须用变量的地址作为实参，则引用更容易使用，更清晰。

示例 1：引用作为形式参数

```
void swap(int & i, int &j)
{
    int tmp = i;
    i = j;
    j = tmp;
}
int main(void)
{
    int a=3,b=4;
    swap(a,b);
}
```

示例 2：函数返回引用类型

```
//返回引用：
int &fn(int &num)
{
    return(num);
}
int main(void)
{
    int n1, n2;
    n1 = fn(n2);
    return 0;
}
```

2．const 常量定义

const 在英文中是固定不变的意思，用 const 修饰的常量不能修改，常用来定义一个符号常量。

在编程中，为了代码更容易维护，通常把一些只读的变量定义为常量形式，这样可以防止程序员在编程过程中由于不小心而导致的对不应该修改的变量进行修改。比如，标准 C 库中的字符串复制函数原型是：

```
Char*strcpy(char* dest, const char *src);
```

细心的读者可能会发现，第 2 个参数是前面增加了 const 关键字修饰。这是因为这个函

数中 dest 是目标指针，存放新的内容；src 是数据源指针，只提供源数据；执行函数并不会修改 src 源指针指向的内存单元内容，所以为提高代码的可读性和安全性，把它定义为常量指针（指向的内存单元是常量，不可修改）。

在形式参数和普通变量上增加 const 关键字修饰，在 C++和 C 中的特性是一样的。但是是在 C++语言中，对 const 的应用进行了扩展。比如，在类成员函数上使用 const 修饰具备不同特殊含义，这在后面的类学习中将会讲述。下面列举常见的 const 修饰用法。

常量：const 类型说明符 变量名

例子：

```
const   int var = 10; //定义一个常量 var，初始化值为 10。
```

定义常量时一定要进行初始化，因为后面不能再对 var 进行修改。

常量引用：const 类型说明符 &引用名

例子：

```
int var;
const int &   ref_var= var;   //定义一个引用，初始化为变量 var，这个也在定义时直接初始化，后面不能对 ref_var 进行任何修改。
```

常量对象：类名 const 对象名

#include <iostream>

#include <string>

using namespace std;

例子：

```
const   string   str_obj="1234567";     //定义一个 string 类常量对象，也是要在定义时初始化，后面不能进行修改。
```

常量成员函数：类名::函数名(函数形参) const　　说明：这个是 C++特有用法

例子：

```
class MyClass
{
public:
    MyClass(){};
    ~MyClass(){};
     int get_x(void) const;   //这个函数只负责返回类 x 的值，并不修改类成员的值，所以加 const
private:
     int x;
};
int MyClass::get_x(void) const
{
    return x;
}
```

上面 class MyClass 的 int get_x(void) const;函数内容只负责返回类成员 x 的值，并没有

对类中的任何成员进行修改，所以在函数后面添加了 const 修饰。这样在函数体的对像类中任何成员的写操作，在编译时都会报错，提高了代码的安全性。

常量数组：类型说明符 const 数组名[大小]

```
int const arr[10]={1,2,3,4,5,6,7,8,9,10};
```

常量指针： const 类型说明符* 指针名

```
int const arr[10]={1,2,3,4,5,6,7,8,9,10};
int const *p_int = arr;   //指向数组 arr
//*p_int = 10;            //错误，常用指针不能修改指向内存地址的值
```

指针常量：类型说明符 * const 指针名

指向常量的指针常量就是一个常量，且它指向的对象也是一个常量。注意，指针常量很容易和常量指针混淆，指针常量意思是指针本身是一个常量，其指向地址不能修改，并不是它指向地址的内存单元是常量，所以指针常量在定义时一定要对它进行初始化。

例子：

```
int   arr[10]={1,2,3,4,5,6,7,8,9,10};
int *const p_int = arr;          //指向数组 arr
*p_int = 10;                     //正确，
p_int = arr;                     //错误，就算是指向原来的址，在编译时也会报错
```

3．函数的默认参数

在 C++中，对 C 语言的函数特征进行了扩展，其中增加的一个新特性就是函数可以有默认值。所谓的默认值，就是在调用时可以不写某些参数的值，编译器会自动把默认值传递给调用语句。默认值可以在声明或定义中设置；也可在声明或定义时都设置；但在都设置时要求默认值是相同的。

对于新的事物，以一个代码为例子来学习会比较快。以下是示例代码：

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

int def_var_1 = 20;
/*
 *    C++函数中可以在定义时设置默认参数，
 *    在调用时传递参数数量可以只传递不同于默认参数的实参。
 */
void default_parameter_func(int num1=def_var_1,int num2 = 3,char ch = '*');


/*
 *    C++函数中可以在定义时设置默认参数，
 *    在调用时传递参数数量可以只传递不同于默认参数的实参。
 */
```

```
void default_parameter_func(int num1/*=def_var_1*/,int num2/* = 3*/,char ch/*= '*'*/ )
{
    cout   <<"default_parameter_func(int num1=def_var_1,int num2 = 3,char ch = '*')" <<   endl;
    cout   <<"当前调用时，参数值分别是"
           <<   "num1="<<num1<<','
           <<"num2="<<num2<<','
           <<"ch="<<ch<<endl;
    cout   << endl;
}


int main(void)
{
    cout   << "/**************默认形式参数测试****************/" <<endl;
    cout   << "执行：default_parameter_func(123,456,'A')" <<endl;
    default_parameter_func(123,456,'A');

    cout   << "执行：default_parameter_func(100)" <<endl;
    default_parameter_func(100);

    cout   << "执行：default_parameter_func(40,9)" <<endl;
    default_parameter_func(40,9);

    cout   << "执行：default_parameter_func()" <<endl;
    default_parameter_func( );

    cout   << "不能执行：default_parameter_func(3,,'L')" <<endl;
    //default_parameter_func((3,,'L'));       //这样写是错误的。

    cout << endl;

    return 0;
}
```

代码中使用的 cout<< 是 C++的输出流运算符号，没有 C++基础的读者可以先不用理解，把它看成相当于标准 C 的 printf 函数一样，能在计算机屏幕上输出内容。代码中定义了一个函数 default_parameter_func，另外声明时给形式参数设置了默认值。“void default_parameter_ func(int num1=def_var_1,int num2 = 3,char ch = '*');”在主程序中调用 default_parameter_func 函数时就可以有多种调用方式。拥有默认参数的，调用时可以不传递，就像上面代码中一样，三个都有默认参数，调用时甚至可以一个参数都不传递，这时函数使用声明中设定的参数。

运行结果如图 1.1 所示。

以下总结一下默认参数的语法与使用：

（1）在函数声明或定义时，直接对参数赋值，这就是默认参数。

（2）在函数调用时，省略部分或全部参数，这时可以用默认参数来代替。

（3）默认参数只可在函数声明中设定一次。只有在没有函数声明时，才可以在函数定义中设定。

（4）如果一个参数设定了默认值，其右边的参数都要有默认值。如：

```
void func(int num1,int num2 = 3,char ch = '*'){};          //正确，按从右到左顺序设定默认值
void func(int num1=2,int num2,char ch='+'){};              //错误，未按照从右到左设定默认值 num1
设定默认值了，而其右边的 num2 没有默认值。
```

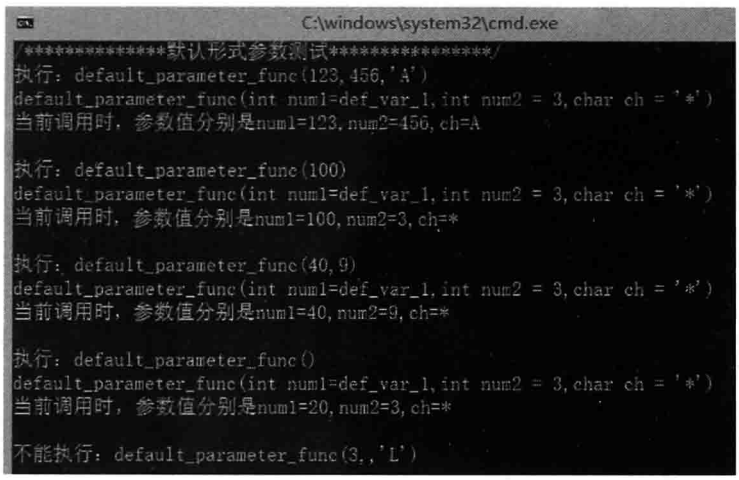


图 1.1　默认参数测试运行结果

（5）默认参数在调用时，则遵循参数调用顺序，自左到右逐个调用。这一点要与第（2）条分清楚，不要混淆。

上述示例代码如下：

```
default_parameter_func(123,456,'A');    //调用时有指定参数，则不使用默认参数
default_parameter_func(40,9);           //调用时只指定 2 个参数，按从左到右顺序调用，相当于
default_parameter_func(40,9,'A');
default_parameter_func(100);            //调用时只指定 1 个参数，按从左到右顺序调用,相当于
default_parameter_func(40,9,'A');
default_parameter_func(3,,9)            //错误，应按从左到右顺序逐个调用
```

（6）默认值可以是全局变量、全局常量，甚至是一个函数，但不可以是局部变量。因为默认参数的调用是在编译时确定的，而局部变量的位置与默认值在编译时是无法确定的。

4. 内联函数

内联函数从代码形式上看，有函数的结构；但是在编译后，却不具备函数的性质。编译时，如同 #define 宏，内联函数通过避免被调用的开销来提高执行效率，一般在代码中用 inline 修饰。

内联函数和宏很类似，但区别在于：宏是由预处理器对其进行替代，而内联函数是通过编译器控制来实现的。而且内联函数是真正的函数，只是在需要用到的时候，内联函数像宏一样展开，所以取消了函数的参数压栈，减少了调用的开销。用户可以像调用函数一样来调用内联函数，而不必担心会产生处理宏所带来的一些负面问题。

内联函数的定义:

```
inline 函数返回值类型    函数名(形式参数列表) { 函数体代码 }
```

当用户定义一个内联函数时，在函数定义前加上 inline 关键字，并且将定义放入头文件就可以了。内联函数示例：

```
inline int max(int a,int b)
{
    return    a > b ? a : b;
}
```

在 C++中，还存在另外一种隐式的内联函数定义规则：在类定义体内部直接实现代码的成员函数都会被自动认为是内联函数。示例如下：

```
class MyClass
{
public:
    MyClass(){};
    ~MyClass(){};

private:
    int i,j;
public:
    int add() { return i+j;};              //自动会被编译成内联函数
    inline int dec() { return i-j;}        //显式声明为内联函数
    int get_num();                         //此处为普通函数，但是可以在类外实现为内联函数
};

//类外实现为内联函数
inline int MyClass::get_num()
{
    return i;
}
```

上面的 add()、dec()、get_num()三个函数都是内联函数。在 C++中，在类的内部定义了函数体的函数，被默认为是内联函数，而不管是否有 inline 关键字。

5. 函数重载

函数重载是函数的一种特殊情况，为方便使用，C++允许在同一范围中声明几个功能类似的同名函数，但是这些同名函数的形式参数（指参数的个数、类型或者顺序）必须不同，也就是用同一个同名函数完成不同的功能。

示例代码如下：

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