

# 学习和使用

# Visual C++

顾铁成 萧柔 等编

徐殿祥 审校

南京电子技术出版社



100020

7/P312

G63

OOP 技术丛书

# 学习和使用 Visual C++

顾铁成 萧 柔 等编著

徐殿祥 审校



南京大学出版社

1995·南京

(苏)新登字 011 号

## 内 容 简 介

Microsoft Visual C++是一种编程工具,用于在集成 Windows 环境中构造和调试基于 Windows 的应用程序和库。Visual C++综合了高级C++应用程序框架类和集成的 Windows 宿主开发工具,因而使复杂的 Windows 应用程序开发任务大为简化。

Visual C++中用到的 Windows 宿主开发工具包括 Visual Workbench、App Studio、AppWizard、ClassWizard,以及其他一些可从 Visual Workbench 中(或由它本身)访问的实用程序。

本书比较详细、系统地介绍 Visual C++程序设计的方法与技术。

本书及其姊妹篇《Visual C++开发工具 Visual Workbench 和 AppStudio 指南》是一套学习和使用 C++和 Visual C++极好的教材和教学参考书,可供从事计算机系统软件及应用软件开发的工程技术人员使用。

35196/13



学习和使用 Visual C++

顾铁成 萧 柔 等编著

徐殿祥 审校

\*

南京大学出版社出版

(南京大学校内 邮政编码 210093)

江苏省新华书店发行 陆军指挥学院印刷厂印刷

\*

开本: 787×1092 1/16 印张: 20.5 字数: 499 千

1995 年 9 月第 1 版 1995 年 9 月第 1 次印刷

印数: 1-5000

ISBN 7-305-02873-8/TP·136

定价: 25.00 元

# 前 言

Microsoft Visual C++ 是目前正在流行的编程工具,用于在 MS Windows 环境中构造和调试基于 Windows 的应用程序和库。Visual C++ 综合了高级 C++ 应用程序框架类和 MS Windows 宿主开发工具,因而使复杂的 Windows 应用程序开发任务大为简化。

Visual C++ 中用到的 Windows 宿主开发工具包括集成开发环境 Microsoft Visual Workbench, App Studio, AppWizard, ClassWizard, 以及其他一些可以从 Visual Workbench 中(或由它本身)访问的实用程序。

在开发过程中,可以利用 AppWizard 来为项目创建 C++ Microsoft Foundation Class Library 源文件。如果要创建和编辑对话框、菜单、工具条等控件资源,可以使用 AppStudio。ClassWizard 则可以用来为资源或 View 和 Document 类增加 C++ 类框架代码和消息图。

Visual C++ 有两种版本: Visual C++ Standard Edition (标准版本)和 Visual C++ Professional Edition (专业版本)。两种版本使用相同的、但略有些区别的集成开发环境。

本书分为上下两篇。上篇包括第一章到第十章,较详细地、系统地介绍 C++ 语言的基础;下篇包括第十一章到第十九章,介绍了 Visual C++ 程序设计的方法与技术。

参加本书编写工作的人员有顾铁成、萧柔、潘金贵、高青、薛仲清等十二位同志。并由潘金贵和顾铁成同志作了修改和统编。徐殿祥博士对全书进行了仔细的审阅和校改,值此表示深深的谢意。

由于水平、时间所限,不妥之处在所难免,欢迎广大读者批评指正。

编著者

1995 年 8 月于南京

# 目 录

## 第一篇 学 习 C++

### 第一部分 C++ 介 绍

<b>第一章 C++初步</b> .....	3
1.1 使用流进行输入和输出 .....	3
1.2 标准的输出流 .....	4
1.3 格式化输出 .....	5
1.4 标准的错误流 .....	5
1.5 标准输入流 .....	5
1.6 C++注释 .....	6
1.7 函数原型 .....	7
<b>第二章 C++对C的增强</b> .....	9
2.1 缺省函数参数 .....	9
2.2 变量声明的出现位置 .....	10
2.3 域分辨操作符 .....	11
2.4 inline 函数 .....	12
2.5 const 修饰词 .....	14
2.6 枚举类型 .....	15
2.7 重载的函数 .....	16
2.8 链接说明 .....	19
<b>第三章 引用</b> .....	21
3.1 作为别名的引用 .....	21
3.2 引用的初始化 .....	22
3.3 引用与指针:相似性和区别 .....	23
3.4 作为函数参数的引用 .....	23
3.5 作为返回值的引用 .....	26
3.6 小结 .....	27

## 第二部分 类

<b>第四章 类简介</b> .....	28
4.1 在C中创建新的数据类型 .....	28
4.2 在C++中创建新的数据类型 .....	30
4.3 对象的创建和消除 .....	37
4.4 访问数据成员 .....	38
4.5 访问函数与公有数据成员 .....	41
4.6 返回引用 .....	42
4.7 const 对象和成员函数 .....	43
4.8 成员对象 .....	44
4.9 使用头和源文件 .....	47
<b>第五章 类与动态内存分配</b> .....	49
5.1 自由存储区 .....	49
5.2 new 操作符 .....	50
5.3 delete 操作符 .....	50
5.4 自由存储区与内部类型 .....	51
5.5 具有指针成员的类型 .....	52
5.6 赋值操作符 .....	55
5.7 this 指针 .....	57
5.8 在返回语句中使用 *this .....	58
5.9 避免 this 指针的不良使用 .....	59
5.10 赋值与初始化 .....	59
5.11 拷贝构造函数 .....	60
5.12 传递和返回对象 .....	61
5.13 传递和返回对对象的引用 .....	63
<b>第六章 类的其他特性</b> .....	65
6.1 静态成员 .....	65
6.2 静态的数据成员 .....	65
6.3 静态成员函数 .....	67
6.4 友元 .....	68
6.5 类对象数组 .....	73
6.6 自由存储区和类数组 .....	74
6.7 高级自由存储技术 .....	77
<b>第七章 继承性与多态性</b> .....	84
7.1 在C中处理相关类型 .....	84
7.2 在C++中处理相关类型 .....	86
7.3 重定义基类的成员 .....	89
7.4 派生类构造函数 .....	91

7.5	基类和派生类之间的转换	92
7.6	使用基类指针的集合	94
7.7	虚函数	95
7.8	多态性	97
7.9	动态联编	98
7.10	虚函数的实现	98
7.11	纯虚函数	100
7.12	基类和派生类中的析构函数	102
7.13	保护成员	102
7.14	公有和私有基类	103
7.15	多重继承性	104
<b>第八章</b>	<b>操作符重载与转换函数</b>	<b>106</b>
8.1	操作符重载	106
8.2	操作符重载的规则	107
8.3	什么时候不应重载操作符	108
8.4	为数值类重载操作符	109
8.5	将操作符定义为友元函数	112
8.6	重载算术操作符的一些诀窍	113
8.7	为数组类重载操作符	114
8.8	类的转换	116
8.9	由构造函数执行的转换	117
8.10	转换操作符	118
8.11	转换和操作符之间的歧义性	120
8.12	转换之间的歧义性	121

### 第三部分 面向对象的设计

<b>第九章</b>	<b>面向对象设计基础</b>	<b>124</b>
9.1	面向对象程序设计的特点	124
9.2	抽象	124
9.3	类	127
9.4	封装性	127
9.5	类层次	130
9.6	继承代码	130
9.7	继承界面	131
9.8	设计一个面向对象的系统	132
9.9	类的确定	132
9.10	赋予属性和行为	133
9.11	找出类之间的关系	134
9.12	将类安排成层次	135

9.13	复合与继承性	136
9.14	根据继承关系设计类	137
9.15	多重继承性	138
<b>第十章</b>	<b>面向对象设计举例</b>	<b>139</b>
10.1	需求分析	139
10.2	类的设计	140
10.3	定义预备的类界面	143
10.4	层次的扩展	151

## 第二篇 程序设计技术

### 第一部分 提高程序的性能

<b>第十一章</b>	<b>预编译头文件的使用</b>	<b>157</b>
11.1	什么时候预编译源代码	158
11.2	生成和使用预编译的头文件	158
11.3	预编译头编译程序选项	158
11.4	在项目中使用预编译的头文件	166
<b>第十二章</b>	<b>为 16 位的 C 程序管理内存</b>	<b>171</b>
12.1	指针的尺寸	171
12.2	选择一种标准的内存模式	173
12.3	混合内存模式	180
12.4	内存模式的定制	185
12.5	基址指针和数据的使用	191
12.6	为函数使用基址寻址	200
12.7	虚内存管理器的使用	201
<b>第十三章</b>	<b>为 16 位 C++ 程序管理内存</b>	<b>207</b>
13.1	内存模式与类	207
13.2	自由存储区	211
13.3	成员函数的基址寻址	215
<b>第十四章</b>	<b>16 位嵌入汇编程序的使用</b>	<b>218</b>
14.1	嵌入汇编的优点	218
14.2	__asm 关键字	218
14.3	在 __asm 分程序中使用汇编语言	219
14.4	在 __asm 分程序中使用 C 或 C++	221
14.5	寄存器的使用和保存	225
14.6	使用浮点指令	226
14.7	跳转至标号	226
14.8	调用 C 函数	228



14.9	调用C++函数 .....	228
14.10	将__asm 分程序定义为C宏 .....	228
14.11	优化 .....	230
<b>第十五章</b>	<b>浮点数学运算的控制</b> .....	<b>231</b>
15.1	浮点类型的声明 .....	231
15.2	long double 类型的运行时刻库支持 .....	233
15.3	数学软件包 .....	233
15.4	选择浮点选项(/FP) .....	234
15.5	有关浮点选项的库考虑 .....	237
15.6	浮点选项之间的兼容性 .....	237
15.7	使用 NO87 环境变量 .....	238
15.8	不兼容的问题 .....	238

## 第二部分 特殊环境

<b>第十六章</b>	<b>Windows 程序设计</b> .....	<b>239</b>
16.1	优化 Windows 程序的保护模式前言和结语代码 .....	239
16.2	指定程序的开始执行点 .....	241
16.3	Windows DLL 初始化代码 .....	242
16.4	Windows 终止例程 .....	242
<b>第十七章</b>	<b>QuickWin 程序</b> .....	<b>244</b>
17.1	QuickWin 图形库的功能 .....	244
17.2	使用 QuickWin 的两种方式 .....	245
17.3	QuickWin 与 Windows 的比较 .....	247
17.4	QuickWin 用户界面 .....	247
17.5	QuickWin 的增强功能综述 .....	251
17.6	构造 QuickWin 程序 .....	253
17.7	运行 QuickWin 程序 .....	253
17.8	编写增强的 QuickWin 程序 .....	254
17.9	MS-DOS 图形与 QuickWin 图形之间的区别 .....	263
<b>第十八章</b>	<b>混合语言程序设计</b> .....	<b>268</b>
18.1	混合语言调用 .....	268
18.2	语言约定要求 .....	269
18.3	编译和链接 .....	273
18.4	对高级语言的C调用 .....	274
18.5	对 Basic 的C调用 .....	275
18.6	对 FORTRAN 的C调用 .....	277
18.7	对 Pascal 的C调用 .....	280
18.8	对汇编语言的C调用 .....	283
18.9	对高级语言的C++调用 .....	288

18.10 混合语言程序设计中的数据处理 ..... 289

**第十九章 编写可移植的 C 程序** ..... 298

19.1 有关硬件的假设 ..... 298

19.2 有关编译程序的假设 ..... 309

19.3 数据文件的可移植性 ..... 314

19.4 Visual C++ 的可移植性考虑 ..... 314

19.5 Visual C++ 字节顺序 ..... 314

**参考文献** ..... 316

第一篇  
学 习 C++



# 第一部分

## C++介绍

### 第一章 C++初步

C++是由C派生而来的。它是C的一个超集(只有几点例外),意即C中有的C++中都有。C++对C中固有的特性作了一些简单的增强,另外还增加了C中所没有的一些新的重要特性。

本章将介绍C和C++约定之间的一些不同之处。首先介绍C++处理输入和输出的新方式,这对设计一个实用的程序来说是必须了解的。接着,本章还将介绍C++的注释和函数原型。

#### 1.1 使用流进行输入和输出

下面是一个非常简单而又典型的C++程序HELLO.CPP:

```
#include <iostream.h>

void main()
{
    cout<<"Hello,World\n";
}
```

这个程序是有名的C程序HELLO.C的C++版本。程序中包含的不是STDIO.H,而是IOSTREAM.H;它没有调用printf,而是采用了一种我们所不熟悉的语法,其中出现了一个未

定义的名为 `cout` 的变量,按位左移操作符(`<<`)和一个串。

## 1.2 标准的输出流

在C++中,执行输入和输出的设施称为“流”(stream)。本书中的示例程序都采用流来读取和显示信息。名 `cout` 代表标准的输出流,它可以用来显示信息:

```
cout<<"Hello,World\n";
```

串“Hello,World\n”被送至标准输出设备,即屏幕上。`<<`操作符称为“插入”(insertion)操作符。它从要发送的内容(即串)指向它要去向的地方(即屏幕)。

假定我们希望打印的是一个整数,而不是串。在C中,可使用一个 `printf` 语句,该语句有一个描述参数的格式串:

```
printf("%d",amount);
```

在C++中,不需要格式串:

```
#include <iostream.h>
```

```
void main()
```

```
{
```

```
    int amount=123;
```

```
    cout<<amount;
```

```
}
```

程序输出 123。

无需使用格式串,就可以将任何基本数据类型的数据发送至输出流中。`cout` 流会识别不同的数据类型,并对它们作出正确的解释。

下面的例子说明如何用一条语句将一个串、一个整数和一个字符常量送至输出流。

```
#include <iostream.h>
```

```
void main()
```

```
{
```

```
    int amount=123;
```

```
    cout<<"the value of amount is"<<amount<<'.';
```

```
}
```

这个程序向 `cout` 中发送三种不同类型的数据:一个串常量,一个整型变量 `amount`,以及一个字符常量 `'.'`,它给句子加上一个(英文)句号。程序打印出下面的信息:

```
The value of amount is 123.
```

注意多个值是如何用一条语句来显示的:`<<`操作符重复作用于每个值。

## 1.3 格式化输出

到目前为止,我们所给出的示例程序都还没有向 `cout` 中发送格式化的输出。假设我们希望以十六进制(而不是十进制)形式来显示一个整数。`printf` 函数可以很好地完成这一任务,`cout` 又是怎么做的呢?

**注意:** 每当想到如何用 C++ 来完成 C 能完成的任务时,记住整个 C 语言都是 C++ 的一部分。如果找不到一种更好的办法,可以回过去用 C。

C++ 将一组“操纵子”(manipulator)与输出流联系起来。它们可改变整型参数的缺省格式。为了改变格式,要将操纵子插入流中。这些操纵子的名称为 `dec`,`oct` 和 `hex`。

下一个程序例子说明了如何以三种不同的形式来显示一个整数值。

```
#include <iostream.h>

main()
{
    int amount=123;
    cout<<dec<<amount<<' '
        <<oct<<amount<<' '
        <<hex<<amount<<';
}
```

程序插入了各操纵子(`dec`,`oct` 和 `hex`),用来将 `amount` 的值转换为不同的表示形式。

这个程序的输出为:

```
123 173 76
```

变量 `amount` 的十进制值为 123,上面的各个值分别是其不同数制下的表现形式。

## 1.4 标准的错误流

如果要将输出送到标准错误设备上,就要用 `cerr`(而不能用 `cout`)了。利用这一技术,可以从标准输出重定向到另一个文件或设备的程序中,将信息送到屏幕上。

## 1.5 标准输入流

除了显示信息外,有时还需要从键盘中读取数据。C++ 标准输入版本的形式为 `cin`。下面的例子说明了如何利用 `cin` 从键盘读取一个整数。

```
#include <iostream.h>

void main()
```

```

{
    int amount;
    cout<<"Enter an amount. . . \n";
    cin>>amount;
    cout<<"The amount you entered was " <<amount;
}

```

这个例子程序提示用户输入一个量值。然后, cin 将输入的值送给变量 amount。下一条语句利用 cout 来显示该量值,从而说明 cin 操作是有效的。

cin 还可以用来读取其他类型的数据。下一个例子说明了如何通过键盘读一个串。

```

#include <iostream.h>

void main()
{
    char name[20];
    cout<<"Enter an amount. . . \n";
    cin>>name;
    cout<<"The name you entered was " <<name;
}

```

这个程序例子所说明的做法有一个严重的缺陷。字符数组只有 20 个字符长。如果键入了过多的字符,栈就会上溢,并出现奇怪的现象。get 函数可解决这个问题。它在有关 iostream 类库参考的内容中解释。至于目前,例子程序假定读者不会键入过多的字符。

**注意:** 回忆一下, printf 和 scanf 并不是 C 语言本身的一部分,而是运行时类库中定义的函数。类似地, cin 和 cout 流也不是 C++ 语言的一部分,而是在流库中定义的,这就是为什么在使用它们时要包含 IOSTREAM.H 的缘故。此外, <<和>> 操作符的含义与它们所处的上下文有关。仅当与 cout 和 cin 一起使用时,它们才能显示或读取数据。

## 1.6 C++ 注释

C++ 支持以 /\* 开始、以 \*/ 结束的 C 注释形式。此外,它还有另外一种注释形式,很多编程人员都比较爱用这一种形式,即 // 序列。每当此序列出现时(除非它是在一个串中), // 后到当前行末间的所有内容都是注释。

下面的程序例子在前一个例子中加了一些注释:

```

// C++ 注释
#include <iostream.h>

void main()
{

```



```

char name[20]; // 声明一个串
cout<<"Enter an amount... \n"; // 请求一个名
cin>>name; // 读取名
cout<<"The name you entered was "<<name
}

```

## 1.7 函数原型

在标准的 C 中,可以在一个函数被定义之前先声明它。函数声明刻划了函数的名,它的返回值,以及它的参数数目及类型。这个特性称为“函数原型”(function prototype),它使得编译程序可以将函数调用与原型比较,并强制做类型检查。

C 并不要求每个函数都有原型。如果没有给出函数原型,最多也就是出现一条警告。但是,C++却要求每个函数都有一个原型。

下面的示例程序利用一个名为 display 的函数来打印“Hello, world”。

```

// 一个没有函数原型的程序
// 注意: 它将通不过编译
#include <iostream.h>

void main()
{
    display("Hello,World\n");
}

void display(char *s)
{
    cout<<s;
}

```

由于 display 函数没有原型,故这个程序通不过C++编译程序的语法检查。

在前一个例子的基础之上,下一个例子加了个函数原型。该程序能通过编译,不出现错误。

```

// 一个带有函数原型的程序
#include <iostream.h>

void display(char *s);

void main()
{
    display("Hello,World");
}

```