计算机软件
应用系列

C++图形程序设计
——C++接口与图形程序实例

OOP 概念
新特征
内核
疑难解
接口、图形应用
从模仿到创新，迅速掌握 C++

● 严 文 等 编著

● 科 学 出 版 社

计算机软件应用系列

C++图形程序设计

——C++接口与图形程序实例

严 文 等 编著

科学出版社

1995

（京）新登字 092 号

内 容 简 介

本书系统地介绍了C++图形程序设计方法以及图形编程涉及到的各种接口，包括四部分内容。第一部分主要概述各种C++版本的区别和用C++开发程序的流程；第二部分集中讨论C++的核心内容；第三部分阐述C++中不太容易理解的方面，用实例帮助读者澄清C++中的一些概念；第四部分描述了C++在若干方面的重要应用，这部分包含丰富的具有实际功能的应用程序，并运用了大量编程技巧。

本书中的程序实例可在 Borland C++，Turbo C++或 Microsoft C/C++ 7.0（经过少量修改）环境下编译和运行。

本书适用于大、中专院校和理工科院校的广大师生，也可供C++程序员和致力于图形设计的人员参考。

计算机软件应用系列

C++图形程序设计

——C++接口与图形程序实例

严 文 等 编著

责任编辑 刘晓融

科学出版社出版

北京东黄城根北街 16 号

邮政编码：100717

国防科工委印刷厂印刷

新华书店北京发行所发行 各地新华书店经售

*

1995 年 1 月第 一 版 开本：787×1092 1/16

1995 年 1 月第一次印刷 印张：22 1/2

印数：1—4 000 字数：519 000

ISBN 7-03-004254-9/TP・386

定价：29.90 元

前　言

Borland C++是一种具有"Turbo 效率"而且支持 Windows 程序设计的高级编程语言。尽管 Microsoft 公司在 1992 年推出了支持 Windows 的 MS C/C++ 7.0软件包，但 Borland C++使用起来更方便，效率更高，因而赢得了更多用户的喜爱。尤其在 Borland C++ 3.1问世之后，不论是设计 DOS 程序还是设计 Windows 应用程序，在 C++编译器领域，它都具有很强的竞争力。现在，Borland 公司又推出了更高的 C++版本，即 Borland C++ 4.0，这无疑会给它的新老用户提供更方便、更有效的编程环境。

目前，市面上已有许多关于 Borland C++的书籍，例如，要了解 Borland C++软件包的使用方法和语言结构，可以参阅《Borland C++系列教程》(每套四册)；要了解如何用 Borland C++开发 Windows 应用程序，可以参阅《Windows 图文程序设计方法与实例》。这些书籍都由科学出版社出版，它们全面而深入地讨论了 Borland C++在众多方面的应用。只要读者肯花一些时间，精通 Borland C++是不成问题的。但是，随着国内计算机设备的不断丰富，也有相当多的人更愿意坐在机房自己模索计算机语言程序的编制方法，而不是在教室听课或在研究室埋头苦读，这样，提供一本相应的类似培训教程的书籍，以便读者"从模仿到创新"，并快速地掌握 Borland C++程序设计方法和技巧，不仅很有必要，而且相当迫切。

为此，我们编写了本书，并提供了大量由浅入深的程序实例，旨在使读者在最短的时间内通过上机训练成为 Borland C++编程"能手"。本书对读者的唯一要求是具备 C 语言知识。

本书侧重于 Borland C++应用程序设计，全书基本上可以分成四部分：第一部分包括第一章，主要介绍 C，C++，OOP 等相关概念以及 C++的新特征；第二部分包括第二到第四章，主要讨论 C++的核心内容，涉及类、对象、数据封装、操作符重载和继承等；第三部分包括第五章和第六章，讲述虚拟函数、友元函数、this 指针、多态、I/O 等不太容易理解的方面，这一部分的实例可以帮助读者澄清 C++中的一些概念；第四部分包括最后四章，描述 C++在若干方面的重要应用，包括音乐接口、窗口绘制、键盘/鼠标接口、时钟模拟、磁盘文件管理、图形设计及动画处理等，这几章包含了丰富的具有实际功能的应用程序，并运用了大量编程技巧。

本书内容比较全面，程序实例丰富，可供大中专院校师生及 C++（包括 Borland C++和 Turbo C++）程序员参考，也可作为上机培训教材使用。前四章由李梅、朱尽染、张志明、刘秀英执笔，第五章和第六章由李波、何国辉执笔，接下来的四章由朱志勇、李蕾、李宁执笔。魏忠才、程功为本书的编排付出了辛苦的劳动。在此谨表示衷心感谢。

目　　录

第一章　C与C++语言的区别

面向对象的程序设计（OOP）是80年代末、90年代初发展起来的一种程序设计方法和思想，它能有效地控制软件开发中不断增加的复杂性和费用，并通过代码重用达到共享资源、提高软件开发效率的目的，因此，面向对象的程序设计方法已越来越受到软件开发者的关注和重视。Borland C++是十分重要的OOP语言，它是C语言的一个超集，除了对C语言成分作了少量扩充外，还在此基础上增加了面向对象的程序语言机制。

在进行面向对象的程序设计以前，读者首先必须明确，本书采用Borland C++作为面向对象程序设计的工具（书中的程序基本上可在Turbo C++，Borland C++，Turbo C++ for Windows等软件内运行），因此，我们首先讨论C与C++的区别。

1.1　C，C++与OOP

通常，C，Fortran，Pascal等都称为过程语言（procedural language），又称结构化语言（structured language）。用结构化语言程序解决一个简单问题的典型流程是：读取输入、处理数据、输出结果。对于复杂的问题，开发者可能会将其划分成许多功能模块，并在程序中增加功能各异的函数（function）或过程（procedure），以完成各种子功能。这种将问题细分成很多小单元并以“输入—处理—输出”这种单一流程进行处理的方法就是典型结构化语言处理问题的方法。

但是，当问题变得很复杂时，再好的结构化语言也将无能为力。究其原因，主要是在一定的复杂程度范围内，困难程度与复杂程度成正比，但是若超过了一定的复杂程度，困难程度将比复杂程度增加得更快。

仔细分析上述原因可知，关键在于数据。在结构化语言中，数据可分为全局数据（global data）和局部数据（local data）两种，而且不难发现，几乎所有重要数据都是全局数据。假设有一个新程序员受命为某个大型软件计划编写一个小函数，该程序员在设计程序前首先必须了解整个计划所要使用的全局数据，这将花费许多时间。如果在访问全局数据时发生错误，就将影响到原有程序中所有用到此全局数据的函数或过程，这就造成了调试上的困难。

传统结构化语言还有一个缺点，这就是如果修改了某个数据，例如将整型数据改成实型数据，或扩充原来的整型数据为结构类型的数据，尽管只对数据作了较小的改动，但还是会影响整个程序中所有用到此数据的函数。解决这类问题的方法是，限制对数据的访问，也就是将数据“隐藏”起来，只有少量关键函数可以访问它，其它函数则禁止使用，这样一方面可以保护数据，另一方面也可以简化以后的维护和管理工作，而这正是OOP的特点。这种处理方法涉及到OOP的三个核心概念是：封装、继承与多态。

面向对象程序设计的主要原则是，将相关数据和使用该数据的函数封装成一个类（class）。类是一种新的数据类型，类类型的变量称为对象（object），对象中的数据称为

数据成员（data member），对象中处理数据的函数则称为成员函数（member function）。

在对象内，只有属于该对象的函数可以访问对象中的数据，这样就可以防止其它函数修改它的内容，从而达到了保护和隐藏数据（data hiding）的目的，也减少了调试上的困难，并使程序维护起来更加容易。在OOP中，类这种组织方式称为封装（encapsulation）。

在OOP中，类之间还可以传递信息。发送信息的类称为基类（base class），接收信息的类称为派生类(derived class)，基类向派生类传递数据或函数的过程称为继承(inheritance)。继承不仅使得数据共享成为可能，而且减少了编程时间和代码规模。基类、派生类与对象的关系见图1.1。

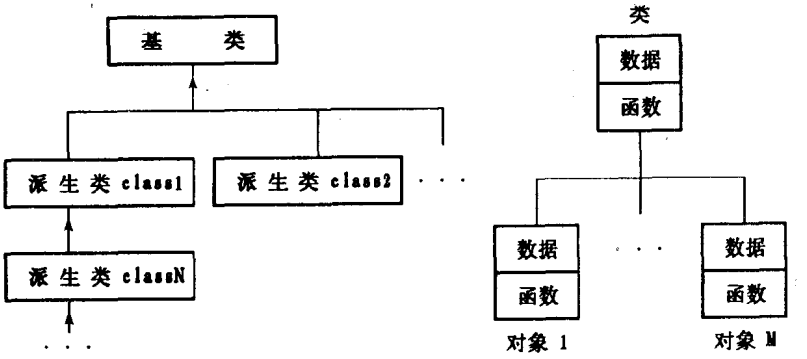

图1.1　基类、派生类与对象的关系

此外，在结构化语言中，功能大致相同而所处理的数据类型或数目不同、或者返回值类型不同的函数不能设计成同一个函数，例如计算整数的平方值和计算实数的n次方的函数只能是名称不同的两个函数，而在OOP或C++中，它们可以是同一个函数，这样可以减少程序员记忆不同函数名称的负担。OOP中的这种机制称为函数重载(function overload)。具体实现时，在编译（compile）或运行（run）程序的阶段根据当前环境确定究竟执行哪个函数的功能，这种机制称为静态联编（static binding，对应编译阶段）或动态联编（dynamic binding，对应运行阶段），这两者形成了OOP的另一特征，即所谓的多态（polymorphism）。

总之，封装、继承与多态是OOP的三大特征（后文将详细讨论），也是从C过渡到C++的桥梁。

C与C++的主要区别在于C++不仅包容了C语言的所有功能（因此可以使用C++语言编写传统的结构化C语言程序），而且注入了OOP概念，形成了以C为基础、以OOP为主流的程序设计风格。

1.2　C++语言成分的新特征

C++语言成分包含与C兼容的部分，同时对以下几个方面作了改进。

1.2.1 C++注释风格

C语言的注释形式如下：

```
/* ... */
```

C语言可接受单行和多行注释，因而以下注释都是正确的：

```
/* C和C++都可接受这类注释，可以是单行注释 */
/* 也可以是多行注释 */
```

在C++中，除了以上方法外，还提供了一种更为方便的形式，即以//开始，止于该行的末尾：

```
// 这是C++注释风格，只适用于单行注释和C++语言程序
```

下面举一个更完整的实例（useless.cpp），以使读者充分了解C++中的注释方法：

```
// useless.cpp: for a C++ programming beginner.
#include <stdio.h> // include standard I/O library
// main () function
/* this main function doesn't return any value,
 * so we declare it to be a VOID-typed function.
 */
void main ()
{
    printf (" New C++ but not C \n");
}
```

程序useless.cpp的输出结果是

```
New C++ but not C
```

由于main ()函数不返回任何值，所以必须声明为void类型。在C中，编译器将未声明返回值类型的函数的返回值视为整型。但在C++中，每个函数都必须有返回值类型声明——无论是否有返回值；若无返回值，则应为void类型。

1.2.2 C++中的简单输入/输出

C++为输入/输出（I/O）提供的库函数中，除了C语言中原有的stdio外，还特别为面向对象的程序设计提供了一组相当方便的I/O库函数及操作符，即流类库iostream，定义在iostream.h中。下面先介绍两个标准输入/输出：

（1）cout（发音see out）：标准输出。

（2）cin（发音see in）：标准输入。

还有两个操作符，即插入操作符（insertion operator）和提取操作符（extract operator），亦称输出操作符和输入操作符：

（1）<<：输出操作符。

（2）>>：输入操作符。

用过C语言的读者可能感到很奇怪，操作符“<<”和“>>”不是右移及左移操作符吗？怎么又是I/O的插入操作符和提取操作符呢？这是因为C++语言允许操作符及函数重载（overload），即相同的操作符可以有不同的功能。这也是C++语言的一大

特点，以后将详细讨论。

使用标准 I/O 操作符的方法很简单，就是将要输出或输入的数据通过输出操作符送给标准输出（cout），或通过输入操作符送给标准输入（cin）。下面先看看输出操作，见程序 cout.cpp。

```
// cout.cpp: an example of how to use iostream library to print data
#include <iostream.h>        // first you have to include
                             // iostream.h instead of stdio.h

void main ()
{
    cout << " New C++ but not C \n";          // print a string
    cout << 12 << " \n";                      // print an integer
    cout << 12.34;                            // print a real number
    cout << " \n";                            // change to a new line
    cout << " Borland C++ is published "      // print mixing data
         << " in " << 1991 << " \n";          // write in many lines
}
```

程序 cout.cpp 的输出结果是

```
New C++ but not C
12
12.34
Borland C++ is published in 1991
```

一般来说，用 cout 输出要比用 printf() 方便得多。由程序 cout.cpp 可知，cout 操作符可以自动识别要输出的数据类型，从而自动调整输出的格式，不必像 printf() 一样每种类型都由用户指定。这样不但方便，而且减少了出错的可能性。连续的输出也可以像 printf() 一样连接起来，例如：

```
cout << " Borland C++ is published in " << 1991 << " \n";
```

同理，可以用输入操作符读取数据，如程序 cin.cpp 所示。

```
// cin.cpp: an example of how to use iostream library to accept data
#include <iostream.h>        // Fisrt, you have to include
                             // iostream.h not stdio.h

void main ()
{
    char name [20];
    int age;
    cout << " Please give your name ( less than 20≠characters ) :";

    cin >> name;
    cout << " Please tell me how old are you ?";
    cin >> age;
    cout << " Your name is " << name << ", ";
    cout << " You are " << age << " years old \n";
}
```

程序 cin.cpp 的输出结果是

```
Please give your name ( less than 20 characters ) : Goodman
Please tell me how old are you ? 28
Your name is Goodman, You are 22 years old
```

比起 C 语言中的 scanf() 函数，程序 cin.cpp 这样的输入格式要方便得多。

1.2.3 定义与声明

1. 数据的定义与声明

现在来看看下面两个例子：

```
int val1;
extern int val2;
```

其中，第一行称为定义，第二行称为声明。所谓定义，是指在引入一个变量名称的同时为该变量分配内存空间，如第一行的“int val1;”就是告诉编译器，要产生一个名为 val1 的整型变量，并且编译器会为其分配两个字节大小的内存空间，这就是所谓的数据定义。第二行的“extern int val2;”则告诉编译器，程序中引用了整型变量 val2，但该变量可能在程序中某个位置已经定义过了，因此在其前面加上关键字 extern，以告诉编译器，该变量为外部变量，这样编译器就不会为该变量分配任何内存，这就是所谓的数据声明。

值得注意的是，在 C++中，程序员可以在程序的任何位置进行变量的声明和定义。这一点与 C 不同，在 C 中，所有变量的声明与定义都必须位于程序或函数的开头。以下是程序 datadef.cpp：

```
// datadef.cpp: an example of how to declare or define data in C++
#include <conio.h>
#include <iostream.h>

void main ()
{
    cout << " Now we are going to print all ASCII code: \n" ;
    cout << " Press any key to continue...\n" << flush;
    getche ();
    // define the variable code without at the begin of
    // the function main ()
    for (int code = 32; code < 256; code++) {
        putchar (code);
        if (code % 80 == 0)     // 80 characters per line
        cout << endl;           // change to a new line
    }
    getche ();
}
```

从程序 datadef.cpp 可知，整型变量 code 的定义不一定要在程序的开头。

```
...
// 整型变量 code 的定义位于程序中
for (int code = 32; code < 256; code++) {
    putchar (code);
```

```
    if (code % 70 == 0)      // 80 characters per line
      cout << endl;          // change to a new line
}
```

在程序 datadef.cpp 中第一次使用了流操纵符 (manipulator) flush，这是有关C++输出方面的应用。利用 flush 流操纵符，可以强迫应用程序将当前输出缓冲区 (buffer) 内的数据送到输出设备 (如屏幕) 上。另外一个常用的流操纵符是 endl，利用此流操纵符可以换行并回到第一列，就像使用 "\n" 一样。例如，下面两条语句是等价的：

```
printf (" Borland Co. C++ 3.x \n");
cout << " Borland Co. C++ 3.x" << endl;
```

2. 函数的声明与定义

就像不可使用一个未经编译器认可的变量一样，也不可使用编译器无法识别的函数。因此，函数声明就是提供函数的原型 (function prototype)，以将程序所需的有关函数的信息通知编译器。C++中函数声明的格式如下：

```
类型声明符 函数名称 (参数表);
```

例如：

```
void noret ( );
int retinteger (int arg);
float retreal (int arg1, float arg2);
```

以上都是合法的函数声明。函数声明的主要作用是告诉编译器，在程序稍后部分将要调用这些已声明的函数。C++中的类型检查比 C 严格，所有函数都必须先声明后使用，这就是在 C++中特别要求有函数原型的原因。

函数定义必须包括两部分：声明符 (declarator) 和函数体 (function body)，例如：

```
int noret ()      // declarator
{                 // start of function body
   ...
}                 // end of function body
```

声明符必须与当初所声明的函数完全符合。所谓完全符合，就是必须有相同的返回值类型、函数名称、参数类型和参数个数。下面这个完整的例子将再次强调这一点。

```
// funcdef.cpp: an example of how to declare and define a function in C++
#include <graphics.h>
#include <conio.h>
#include <iostream.h>
#include <stdlib.h>

enum Size { Small, Big };    // smaller or bigger than 7
enum Result { Lost, Win }; // the result of this compare

// an intresting game of guessing big_or_small compared with 7
void main ()
{
```

```
    // define all data needed
    int money = 1000;          // the money that you have first
    int bet = 0;               // how much you will bet
    int number;                // number of computer given

    // function prototypes
    int getbet (int * mon);                    // get the bet
    Size getplayer ();                         // get the input given by player
    int getcompernumber ();                    // get the number given by computer
    Result compare (Size sob, int num);        // to decide who win
    void showresult (int mon, Result ret);     // to show the result

    // game begin
    Size small_or_big;
    Result win_or_lost;
    // clrscr ();

    int conti = 0;                 // it means doesn't play
    while ( ! conti ) {            // default is playing once
      bet = getbet (&money);       // get bet
      small_or_big = getplayer ();  // get small or big
      number = getcompernumber ();  // get a number
      cout << " \nThe number is " << number << ", ";
      win_or_lost = compare (small_or_big, number);

      if ( win_or_lost == Win )        // win
        money += bet * 2;
      showresult (money, win_or_lost); // show result
      cout << " Do you want to play again ? (Y/N) ";
      char c;
      cin >> c;                        // play again ?
      cout << " \n";
      conti = ( c == 'Y' || c == 'y' ) ? 0 : 1;
    }
}

int getbet (int * money)
{
    int bet;
    cout << " How much do you want to bet ? \n";
    cin >> bet;
    if (bet > * money ) {
      cout << " You don't have so much money, maybe next time \n";
      exit (0);
    }
    * money -= bet;
```

```
    return bet;
}

Size getplayer ()
{
    Size input;
    cout << " (B) igger or (S) maller than 7 ?";
    char c;
    cin >> c;
    cout << " \n";
    input = ( c == 'B' || c == 'b' ) ? Big : Small;
    return input;
}

int getcompernumber ()
{
    int number = rand ();
    number = number % 13;
    return number;
}

Result compare (Size sob, int number)
{
    Result result;
    Size smallerbig = (number > 7) ? Big : Small;
    result = (sob == smallerbig) ? Win : Lost;
    if (number == 7)
      result = Lost;
    return result;
}

void showresult (int money, Result result)
{
    char *wol [] = { " lost.","  win.", NULL };

    cout << " You " << wol [result] << " \n";
    cout << " Current money is:" << money << endl;
}
```

程序 funcdef.cpp 的输出结果是

```
How much do you want to bet ?
1000
(B) igger or (S) maller than 7 ? s

The number is 10, You lost.
Current money is: 900
Do you want to play again ? (Y/N) n
```