·北京科海培训中心系列教材

# Turbo C++图形编程技巧

周少柏　查良钿　编

科学技术文献出版社

北京科海培训中心系列教材

# Turbo C++图形编程技巧

周少柏 查良钿 编

科学技术文献出版社

（京）新登字130号

## 内 容 简 介

面向对象的程序设计语言Turbo C++是开发系统软件和实用程序的良好工具。本书通过介绍Borland图形接口，探讨各种实用的高级绘图编程技巧。内容包括三维图形、动画技术、字形构造、图符编辑、上弹窗口、绘图例程和CAD程序等。书中提供经过测试的完整算法源程序，只需配备Turbo C++软件，就可在IBM PC XT，AT，PS/2或其它能显示图形的计算机系统上编译运行。

本书可供从事有关计算机科学研究的工程技术人员、计算机用户参阅。也可作为大专院校有关专业的参考教材。

**Turbo C++图形编程技巧**

周少柏 查良钿 编

科学技术文献出版社出版

（北京复兴路15号 邮政编码100038）

河北省蔚县印刷厂印刷

新华书店北京发行所发行 各地新华书店经售

787×1092毫米 16开本 18印张 273千字

1993年5月第1版 1993年5月第1次印刷

印数：1—5000册

ISBN 7-5023-2010-5/TP·105

定价：16.00元

# 前　言

如果你想采用面向对象的程序设计方法，用Turbo C++编写高级的图形程序，本书可能是理想的阅读和参引材料。这里将详细介绍一个称为Borland图形接口的绘图工具包，它允许你充分利用面向对象的编程技术，大大简化创建二、三维个人计算机图形的工作。

众所周知，C是国际上流行的程序设计语言之一，可适用于多种型号的计算机。而最近开发的Turbo C++，继承了C的全部特性，并增添了不少为提高效率、方便用户而设置的新功能。特别是，它支持“面向对象程序设计”的新潮流，从而迅速成为深受用户欢迎的高级语言。

图形技术是计算机应用的一个重要分支，在国民经济、文化教育、军事等领域都有广阔的发展前途。使用Turbo C++提供的图形库函数，你将不难建立真实世界的各种图形应用程序。

值得一提的是，本书所提供的实例程序，都适用于虚拟图形阵列（VGA）、增强型图形适配器（EGA）、彩色图形适配器（CGA）或Hercules图形适配器。正是这种灵活性，使之非常容易移植而运行在各自的计算机——包括IBM PC及其所有兼容机的系统上。

全书共分十四章。前四章详细介绍Turbo C++中的Borland图形接口；第五章开始讨论如何开发能产生高质量表示图的程序，包括饼图、条形图、楔形图等图形显示；第六、七章涉及处理二维图形对象软件的各种技术，并增添动画效果和使用彩色调色板；第八至十四章建立交互式的图形工具，如图符编辑器、上弹窗口、绘画例程等。采用这些工具，配合安装兼容Microsoft的鼠标器，能构造像计算机辅助设计那样的交互式图形系统。

在编译本书的过程中，得到北京科海培训中心华根娣、夏非彼以及中国科学院计算所王玺玉的热情支持和帮助，借此机会，向他们表示衷心的谢意。

# 目　录

# 第一章 Turbo C++—更好的C

C是美国电话电报公司（AT&T）所属的贝尔实验室（Bell labs）开发的程序设计语言。它以简洁的表达方式、强有力的控制流程、灵活的数据结构以及丰富的操作符而著称，目前已成为国际上流行的编程工具。它的许多优点正被人们逐渐认识，从而应用越来越普遍，受到各方面的关注和重视。

本章概述与C兼容的Turbo C++，它支持面向对象的编程法，提供集成化的程序设计环境，使得应用软件更容易生产，程序更加可靠。人们可以利用预先编制的小型程序（对象软件）来构筑各种大型程序，应用程序已经变成真正的对象——封装式的代码和数据。重复地使用这些程序模块，使得那些过去十分困难和耗时的高级绘图任务，现在变得简单多了。

## 1.1 向Turbo C++过渡

美国Borland International计算机公司研制成功的Turbo C++，是比C更完善的语言版本。如同它的名字所示，它自然是导自于C。其设计思想是从C向上兼容，所有用C语言编写的程序，只需稍加修改，即可用Turbo C++编译程序处理。但采用Turbo C++，将具备更多的优点。例如，对函数变元和变量赋值的强类型检查，可大大减少编程的错误。显然，它更适合于编写大型的程序。

Turbo C++语言相对复杂一些，但却有助于你写雅致的程序。如果你已经熟悉C，最好用Turbo C++取而代之，它将使你成为更好的C程序员。如果你已学过其它算法语言，如FORTRAN、Pascal等，又要想学习别的语言，请选Turbo C++。

随着个人计算机演变得越来越复杂，编程已不是很容易的任务，需要各种软件工具的帮助。Turbo C++支持面向对象的程序设计，提供丰富的库函数，生产效率高，可移植性强，它最终将成为主流语言之一。不少软件厂商已推荐用它来写系统软件和应用程序。

学习Turbo C++并不难，尤其是已有C语言基础的读者。然而，它学习容易精通难。为更快地掌握它，最好的办法是多多读、写有关的程序，并在计算机上反复实践。Turbo C++是针对大型程序的。它使用典型的预处理方式，增加了一些额外开销，编译时间相对长些。但是，这种损失可以从减少错误、方便调试而得到补偿，尤其是在开发大型程序时更能显示出其优越性。

## 1.2 新的语言特点

对C语言进行了一些必要的扩充，如对函数变元和变量赋值的强类型检查，重载函数等等。它们简化了编程、减少错误并使之更适合于大型程序设计。而大程序中有许多严格的接口准则是需要遵守的。下面对一些主要特点分别作介绍。

### 1.2.1 注解

C++中最寻常而又最受欢迎的新特点之一是双斜线（//）注解。这个注解指示符告诉编译程序忽略掉任何跟随的正文，直到该行的末尾为止。仍然支持旧式的C注解，它用/*开始，而以*/结束。//注解的极大优点是，除了比较容易使用之外，还在于它们可以嵌入旧式的注解块中。例如，考虑下面的函数和它旧式的注解：

```
int foo(int a, int b){
   /* here is a comment about the function */
   return a + b;
}
```

如果你想用旧式的注解把整个函数定义注出，那可能做不到，第一个注解行末尾的终止注解记号会提前结束较大的注解信息块。旧式注解是不能嵌套的。过去，为把整个函数或包含注解的大块代码注出，C程序员只好用#ifdef伪指令，如下面的代码所示：

```
#ifdef NEVER
int foo(int a, int b){
   /* here is a comment about the function */
   return a + b;
}
#endif
```

如果用双斜线重写上述的函数，则很容易用旧式注解把整个函数注出，如下所示：

```
/*
int foo(int a, int b){
   // here is a comment about the function
   return a + b;
}
*/
```

### 1.2.2 说明和定义

C++要求所有函数和变量必须在使用前定义。说明是一种陈述，它告诉编译程序有关变量或函数的信息。对于变量，说明指出它的名字和数据类型。对于函数，说明指定函数的名字、它的返回类型以及函数变元（如果有的话）的数据类型。说明不对变量和函数分配任何空间，它只不过是指定变量或函数接口的一种方法。

而在另一方面，定义则为变量和函数明显分配空间。说明描述接口，定义描述如何实现。

变量的说明和定义往往组合在同一个语句中。例如，下面的语句说明和定义变量。请注意，变量可以用定义语句初始化。

```
int   foo = 5;
Rect  theRect;
float f = 3.66;
```

为说明一个变量而不实际定义它，你必须把extern关键字放在变量说明之前，如下面的说明语句所示。extern告诉编译程序该变量在另外的地方定义。

```
extern int foo;
extern Rect theRect;
extern float f;
```

函数的说明和定义则经常是分开的。函数通过列出它的名字、返回类型和变元的类型来说明。如下面的语句所示：

```
int  foo(int a, int b);
void bar(int,short,char *);
char * batz(void);
```

有关上述的函数说明可提出几点：首先，在函数定义中的变元表可以不包含变元的名字，但必须指出每一变元的类型。所以，对C++而言，下面的函数说明全是等价的：

```
int foo(int, int);
int foo(int a, int);
int foo(int, int b);
int foo(int a, int b);
```

对函数说明的另一点提示是返回类型可以为void。void函数不返回任何值。

最后，请注意，变元表也可以为空（void)。它指示该函数不取任何变元，你也能简单地通过使变量表为空白而表明该函数不取任何变元。于是，下面的两个函数说明是等价的：

```
char * batz(void);
char * batz();
```

函数定义提供与函数说明相同的信息，但增加定义该函数实现的代码块，如下面的函数定义所示。当C++遇到函数定义时，它将使用函数块产生实现该函数的真正代码。

```
int foo(int a, int b){
  return a + b;
}
```

那么，什么时候说明和什么时候定义呢？C++程序通常可分解为几个独立的源文件。这些文件被分别编译，然后连接在一起以产生最后的应用程序。把有关的函数和变量放在同一个源文件中，能使你在其它的程序设计方案中重用这个文件。例如，假设你已定义了一组函数，它们用欧姆定律计算电阻、电压和电流的关系，那末可以说明为下面的函数：

```
float GetVolts(float ohms, float current);
float GetCurrent(float volts, float ohms);
float GetOhms(float volts, float current);
```

这些函数的说明放在文件elec.h中。函数的定义则放在文件elec.cp中。实现的文件elec.cp被编译一次以产生目标文件elec.cp o，即包含该函数的实际目标代码的文件。这个文件必须和其它目标文件连接以形成应用程序。

标题文件elec.h包含函数的说明，可以出现在需要调用电流函数的源文件中。elec.h中的说明告诉C++有关函数的充分信息，以便能在调用这些函数时做所需的类型检查。但它不必知道有关这些函数的定义。当它进行这种类型检查时，函数的真正实现是无关重要的。由于单独的函数说明不会产生任何代码，你可以在若干其它文件中包含标题文件，但不必每次重编译那些函数。

当然，如果你包含了带函数说明的标题文件，就必须连接包括这些函数定义（实现）

的目标文件。否则，连接程序将指出它找不到失去的函数。

同样，对于变量，如果需要有一些全局变量包含在若干其它文件中，那么，你可以在标题文件中把它们说明为extern，然后在各自的实现文件中定义它们。用这种办法，它们只在一个文件中定义，但可以从任何包含该说明标题的文件引用它们。

C++对C的一个重大改进是：在C++中，你可以在任何普通语句能出现的地方定义变量。当在函数块中定义局部变量时，这是最有用的。在C中，函数里的所有变量说明必须出现在函数的首部。如果函数很长，这可能会发生问题，因为你要回到头部检查变量的名字。当在很长的函数定义中删去一段代码时，也可能产生问题，因为你也许忘记返回头部消除被删代码所用的那些变量定义。

C++允许正好在使用变量之前定义它们。于是，在很长的函数定义中，你可以把各自的变量定义和使用它们的代码组合在一起。这使代码更易读和可维护。

这种特色的一个普通用法是在for循环的for子句中定义循环计数器变量，如下面的代码段所示：

```
for(int i = 0; i< maxCount; i++){
```

### 1.2.3 类型检查

C++检查所有函数变元的数据类型。这是为了在编译时而不是留待运行时查出更多的错误。C++要求在函数说明中有全部的变元表，其目的在于能在调用函数时检查变元表中的全部变元。

强类型检查意味着在把不兼容的变量传送给函数时，C++将在发出警告或拒绝编译这些函数调用。当所传送的变量类型有可能被转换为函数所期待的类型时，C++将进行某些自动的类型转换。例如，当把浮点数传送给期待整数的函数时，C++将在把它传送给函数之前，自动将浮点数截取为整型的成分。当转换会改变该值时（如浮点数向整数转换），C++将在编译时发出警告。

强类型检查是一个很大的优点，但有时会有些限制。假如想把指针变量传送给期待长整数的函数，你清楚知道指针和长整数有同样的尺寸，但编译程序办不到，它没有标准的方法把指针转换为长整数。在要把指针作为函数变元传送时，你可以显式地把指针强制为长整数类型而解决这个问题，如下面的代码段所示：

```
char * p;          // definition of pointer variable
void foo(long);    // declaration for function with one long arg
foo(p);            // compiler will complain !!!
foo((long)p);      // typecast makes it all OK
```

强制类型是告诉C++你明白你所做的一切的一种方法。实际上，使用C++重载函数的功能，可以定义上述例子所示函数的其它版本，以便能用指针变元或长整数变元调用它，从而消除了使用显式强制类型转换的必要性。有关细节请参见本章“重载函数”一节。

### 1.2.4 参引的变元

当在函数的变元表中说明变元时，可以在变元类型名字后面跟一个&符，以指示该变元被参引而不是赋值传送。这意味着C++将把指向变元的指针而不是把变元的副本传送给

函数。于是，你将能改变函数中变元的值，并使这些改变体现在说明为变元的变量中。

参引传送基本上和Pascal的VAR变元特点相同。参引传送使你能在说明变元时指定变量的名字，而不必指定变量的地址。这也意味着你在函数中不用解参引指针以取得变元的内容。对于结构和其它复杂的数据类型，这是十分有用的。

考虑下面的例子。假定已如下定义一个表示矩形的数据结构：

```
struct Rect {
   short top;
   short left;
   short bottom;
   short right;
};
```

现在，考虑一个计算矩形面积的函数。在C中，你应定义该函数取一指向Rect的指针，如下所示：

```
int Area(Rect * r){
   return ((r->bottom - r->top) * (r->right - r->left));
}
```

当调用该函数时，应提供Rect变量的地址作为变元，如下面的代码段所示：

```
Rect theRect;
int theArea = Area(&theRect);
```

使用C++的按参引调用语法，可如下定义面积函数：

```
int Area(Rect& r){
   return ((r.bottom - r.top) * (r.right - r.left));
}
```

当调用这个函数版本时，可以简单地提供Rect变量的名字作为变元，如下面的代码段所示：

```
Rect theRect;
int theArea = Area(theRect);
```

参引传送语法使你避免了典型用于传送指针变元的解参引和取地址操作。参引传送还迫使调用程序分配被调用函数所用的结构。例如，如果你使用取指针变元的面积函数，就不可避免地要分配一个Rect指针变量，而它实际上并不指向有效的Rect结构，因而错误地把指针传送给Area，如下面的代码所示。

```
Rect *pRect;
int theArea = Area(pRect); // DANGER: using uninitialized Pointer!
```

### 1.2.5 缺省变元值

C++的另一新特色是在函数的说明中把缺省值赋与函数变元的能力。例如，假定你说明一个函数，它以指定的抽样率发音。这个函数取两个变元，一个是指向声频数据的指针，一个是指定抽样率的整数，如下面的说明所示：

```
void PlaySound(char * sound, int sampRate);
```

进一步假定在你的系统中，一般的声音抽样率为22 000Hz。你可以为该抽样率变元

定义一个缺省值，以便该函数的调用程序不必提供抽样率变元（如果该缺省频率是可接受的）。带缺省赋值的说明如下所示：

```
void PlaySound(char * sound, int sampRate = 22000);
```

C++仅当调用程序不提供变元时才替换缺省变元值。变元表中不能多于一个变元有缺省值，但替换是基于变元位置的，所以在变元表中带缺省值的变元不能跟有非缺省的变元。换句语说，缺省变元必须是变元表中的最后一个。于是，如果像下面那样说明PlaySound函数将是不正确的：

```
void PlaySound(int sampRate = 22000, char * sound); // INCORRECT !!
```

### 1.2.6 直接插入的函数

C++允许你定义直接插入函数。直接插入函数不是通过普通的函数调用机构引用的，它涉及堆栈处理。这对那些经常调用的或小型的函数可能是低效的，因为它们的函数调用开销可能比函数实际所做的工作还要多。另一方面，直接插入函数体将直接替换到调用直接插入函数处的代码中。某些变元也作类型检查和替换。替换是在编译时进行的；在运行时没有实际的函数调用，只是执行组成该函数的替换代码。

直接插入函数基本上替代了C程序员通常所用的宏功能，避免了短的、频繁调用例程的函数调用开销。直接插入函数比宏功能优越，它对直接插入函数的变元有强类型检查，并返回类型。例如，在C程序中，你可以定义下面的宏以返回32位长整数的高、低16位字：

```
#define HiWrd(aLong) ((short) (((aLong) >> 16) & 0xFFFF))
#define LoWrd(aLong) ((short) ((aLong) & 0xFFFF))
```

在C++中，你可用直接插入函数代替宏，如下面的直接插入函数定义所示：

```
inline short HiWrd(long aLong)
   {return (short)(((aLong) >> 16) & 0xFFFF);}
inline short LoWrd(long aLong)
   {return (short)((aLong) & 0xFFFF);}
```

### 1.2.7 Const 定义

在传统的C程序中。通常用#define语句定义常数，如下所示：

```
#define  MAXSHORT 32767
#define  MINUSONE -1
```

C++提供一种更好的机构以定义常数，它允许你指定常数的精确数据类型和其值。在C++中，从前的#define语句将用下面的语句代替：

```
const  short MAXSHORT = 32767;
const  long MINUSONE = -1;
```

Const定义主要创建只读变量。Const变量在定义时必须初始化。此后，将不能改变它。Const定义比#define语句优越的是Const变量有定义好的数据类型，并因而在作为函数变元时能进行类型检查。

### 1.2.8 重载函数

C++中的函数重载允许你定义若干种函数版本，每一版本都有不同的变元表。然后，以同一函数名字引用这几种函数定义。编译程序可通过查看变元的类型而区分不同的版本。例如，再看看本章“类型检查”一节的例子。在那里，必须执行显式的强制类型转换，以便把指针变元传送给期待长整数变元的函数，如下所示：

```
char * p;           // definition of pointer variable
void foo(long l);   // declaration for function with one long arg
foo(p);             // compiler will complain !!!
foo((long)p);       // typecast makes it all OK
```

通过重载foo以便它能接受指针变元，你可以避免必要的强制类型转换，如下所示：

```
cnar   p;            // definition of pointer variable
void foo(long l);    // declaration for function with one long arg
void foo(char *`c); // OVERLOADED function declaration
foo(p);              // compiler will NOT complain !!!
```

每一重载函数必须有它自己的定义。函数的每一个重载版本实际上都是独立的函数，所有函数仅共享相同的名字。通过匹配说明变元表提供的变元，C++决定调用命名函数的那一个版本。

### 1.2.9 流

像C一样，C++不包含任何预先定义的输入和输出操作符。所有的I/O都是由库函数处理的。C有一个stdio库，含有质量不高的printf函数。C++支持stdio，但它也定义了一个称为iostream库的新的I/O类型。流提供和stdio库一样的格式化类型和输入输出服务。然而，它还提供更好的类型检查和附加的对用户定义的数据类型的支持。

流是字节的序列。可以从流中抽出数据并通过extraction操作符（≫）把它们送入程序变量中。也可以通过insertion操作符（≪）把数据插入流中。在典型的UNIX环境中，流可以和标准的输入、输出通道相连。例如，标准的输入常常和用户的终端键盘相连，而标准的输出就是终端显示屏。于是，从标准输入流中抽出数据以检索用户在终端键入的字符，把字符插入标准输出流中以发送这些字符到终端显示屏。

iostream库定义了三个流变量——Cin、Cout和Cerr。它们对应于系统的标准输入、标准输出和标准错误输出通道。你可以在程序中使用这三个预定义的变量抽出和插入数据到标准I/O通道。例如，发送字符串到标准输出通道，可以用下面的语句：

```
cout << "This data will go to the output channel \n";
```

更为重要的是，你可以组合插入操作和用insertion操作符的自动格式功能来产生复合输出，如下面的语句所示：

```
int x = 23;
cout << "The value of x is " << x << ".\n";
```

上面的语句将产生如下输出：

```
The value of x is 23.
```