第2章

C++的基本语法和使用环境

亲自撰写和执行程序是学好程序语言的不二法门。本章通过两个简单的程序，介绍 C++ 程序的基本结构和开发环境，让初学者能逐渐建立使用 C++ 的信心。

2.1 基本程序开发步骤

因为 CPU 只能执行机器码，因此所有的高级程序语言都需要经过编译才能够执行。C++ 程序的开发包括下列六个步骤:

1. **编辑** (edit)
2. **预处理** (preprocess)
3. **编译** (compile)
4. **连接** (link)
5. **装载** (load)
6. **执行** (execute)

所以一个完整的程序开发环境必须包括**编辑器**(editor)，**预处理器** (preprocessor)，**编译器** (compiler)，**连接器** (linker)和**装载器** (loader) 等软件工具，以及提供含有标准链接库(the Standard C++ Library) 和 MFC (微软基础类，the Microsoft Foundation Classes) 等现成软件组件的链接库。

程序的编辑

开发程序一开始，我们必须通过**编辑器** (editor) 来完成程序原始文本文件的撰写。撰写完成的含有 C++ 文字指令之程序文件称为源代码 (source code)。编辑结果通常存放于硬盘、磁盘、CD-R 或 CD-RW 中，C++ 的源文件通常有 .cpp 或 .C 的扩展名(extension)。在 UNIX 系统下，我们有 emacs 和 vi 两种编辑器可以选用，而在个人计算机 Windows 操作系统下可用的编辑器种类更多。如果我们使用 Microsoft Visual C++ 或 Borland C++ Builder 等集成开发环境，则其中已经提供了专用的编辑器，这类专用编辑器和其它部分有较紧密的衔接关系。

程序的编译

其次，要使用**编译器** (compiler) 将程序转译成机器码，称为**目标码** (object code)。在执行编译前，有一个叫做**预处理器** (preprocessor) 的程序会自动被调用，它依照程序内的**预处理指令** (preprocessor directives) 进行编译前的特定处理，包括将称为**头文件** (header file) 的文本文件放入源代码中，以及在原始程序中取代某些特定的文字等动作。在以下各章中，我

们会零星地介绍常用的预处理指令，较完整的说明则放在第 11 章里面。

编译器 (compiler) 的功能在于将预处理器产生的**中间文件** (又称为**转译单元**，translation unit) 转译成**二进制机器码**，称为**目标文件** (object file)，并同时进行语法检查和程序代码最佳化的操作。为了简化编译器的设计工作，某些系统的编译器所产生的文件是我们能够阅读的汇编代码 (assembly code)，其后再自动调用汇编代码编译器 (assembler) 产生目标程序码。目标文件中各个指令和数据被编译器设定的位置只是相对地址 (relative address)，还不是绝对地址 (absolute address)。

最后的编译阶段称为**连接** (linking)。程序多少都会引用一些不是自己撰写，而且已经编译成机器码的程序片段，例如内建的或自行开发的链接库。**连接器** (linker)的工作是将目标文件与在程序中指定的这些机器码文件连接成完整的**执行文件** (executable file)。此外，连接器还进行了将地址从相对地址转换为绝对地址的工作。

以 UNIX 操作系统为例，要针对一个称为 test.C 的源文件完成上述预处理、编译和连接的整个过程，只要在命令提示行 (command line) 的提示符号 (prompt) 后输入

```
cc test.C
```

再按下回车键 (亦即 Enter 键) 即可。如果没有错误，就可以成功编译，而得到一个称为 a.out 的执行文件。

如果是使用 2.4 节即将介绍的 Borland C++ Compiler，则对于文件 test.cpp 的编译只要在 DOS 的命令提示符后键入

```
bcc32 test
```

再按回车键 (Enter) 即可。成功的编译会在显示器上显示以下的输出信息：

```
Borland C++ 5.5.1 for Win32 Copyright (c) 1993, 2000 Borland
Test.CPP:
Turbo Incremental Link 5.00 Copyright (c) 1997, 2000 Borland
```

并在原文件所在的文件夹内产生 Test.obj，Test.tds，和 Test.exe 三个文件。其中 **Test.exe** 就是我们要的执行文件。

也就是说，不管在 UNIX，DOS 还是 Windows 操作系统中操作，预处理器和连接器的动作都隐藏在单一的编译指令之下。除非因为指令错误或找不到指定的连接文件和链接库，

使用者通常觉察不到这两个程序动作的存在.

程序的装载和执行

在程序执行前，需要**装载器** (loader) 的协助把**可执行码** (executable code，例如上述的 a.out 和 Test.exe) 放置于主内存内。在某些情况下，这个装载的动作包括动态连接文件 (DLL，Dynamically Linked Libraries，具有扩展名.dll) 的载入。

上述的过程可以画成图 2.1.1 的步骤示意图：

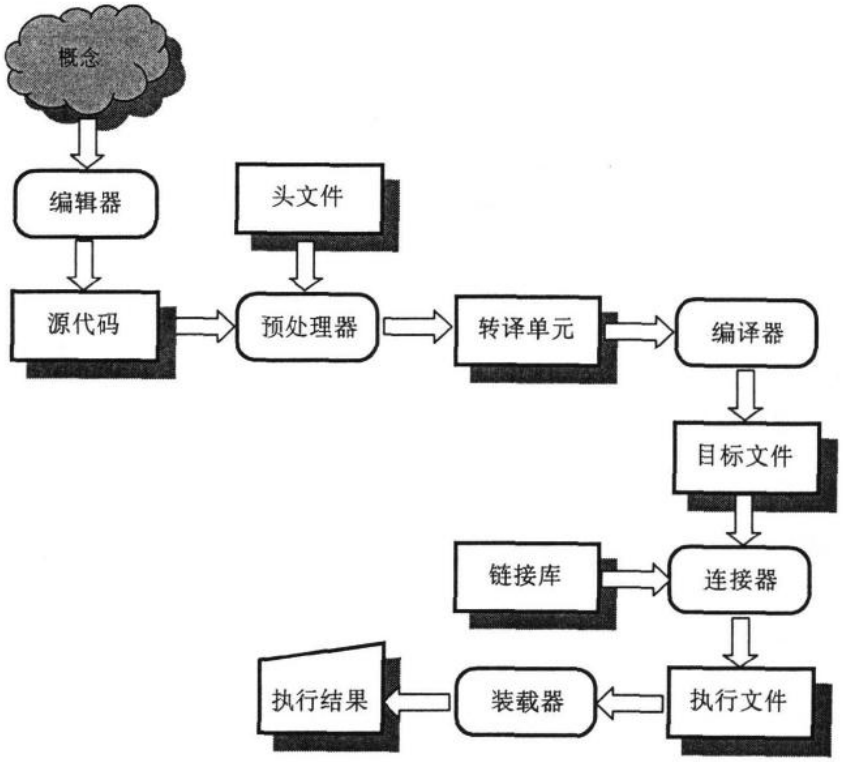

图 2.1.1 C++ 程序开发的步骤

提示

DLL 是一个新的技术，可以把执行文件切割成许多小文件，以便于选择性地在执行时连接特定的执行文件，避免过于庞大的单一执行文件。在有升级或修正需要时，也只要取代部分执行文件就可以。

要进行程序的装载和执行也很简单，沿用上述的例子，在 UNIX 下输入的指令是

```
./a.out
```

在 DOS 下则只要输入

```
test.exe
```

或

```
test
```

再按回车键即可。 如果要把原本会在显示器上显示的结果存到一个称为 Result.txt 的文件内，在 DOS 下也只需要在装载和执行时，同时在执行文件名称后面加上符号“>”，再加上文件名称就可以：

```
test > Result.txt
```

2.2 第一个完整的 C++ 程序

C++ 的语法对于第一次接触这个程序语言的人会显得有些奇特。我们要在本节中介绍一个非常简单，但是可以显示 C++ 语法特性和程序组织的程序。这个程序经过编译后，执行时能在显示器上显示一行文字。

以下是程序的全貌：

```
// Test.cpp
// 程序的目的是在显示器上显示一行文字
#include <iostream>
```

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

// --  程序主体部分 ----------------------
int main()
{
  cout << "Hello, 您好!"  // 主要的语句
       << endl;
  return 0;
}
```

这个程序包括了**注释**，**预处理指令**以及**程序主体**三个部分:

1. **注释** (comments)

在下列程序中的粗体字部分是程序的注释部分：

```
// Test.cpp
// 程序的目的是在显示器上显示一行文字
#include <iostream>
using std::cout;
using std::endl;

// --  程序主体部分 ----------------------
int main()
{
  cout << "Hello, 您好!" // 主要的语句
       << endl;
  return 0;
}
```

程序注释是加于程序源代码内的说明文字，在编译的时候就会被删除，与程序的执行无关。话虽如此，注释仍是程序撰写的时候重要的组成部分，其目的如下：

(1) 在每个文件的开头置入一段文字，以说明其目的、使用方式、以及修改日期等记录，以便于日后追查。
(2) 补充说明变量和函数等自行定义的组件之意义，以及使用上的注意事项。
(3) 插入程序段落的特殊分界符号，使程序更容易阅读。
(4) 在撰写程序时，为了调试上的方便，常可将程序的某些部分暂时转成注释。如此一来，就可以不用真的把该部分程序删除，随时可以恢复，避免无谓的重复输入。

注释有两种做法：

(1) 从双斜线“//”的右边开始，一直到整行结束，都视为注释。
(2) 以“/*”开头，一直到“*/”，中间所包括的文字、数字或符号都视为注释。

因此，上述程序的注释部分也可以写成下列形式：

```
/* Test.cpp
程序的目的是在显示器上显示一行文字 */
#include <iostream>
using std::cout;
using std::endl;

/* --  程序主体部分 ----------------*/
int main()
{
  cout << "Hello, 您好!" // 主要的语句
       << endl;
  return 0;
}
```

除了第 4 种注释的目的 (也就是调试的过程，要将许多行都转为注释) 之外，我们通常使用双斜线的注释法。

2. **预处理指令** (preprocessor directives)

在下列程序中的粗体字部分是对**预处理器** (preprocessor) 下指令的地方，称为**预处理指令**：

```
// Test.cpp
// 程序的目的是在显示器上显示一行文字
#include <iostream>
using std::cout;
using std::endl;
// --  程序主体部分 ----------------------
int main()
{
  cout << "Hello, 您好!" // 主要的语句
       << endl;
  return 0;
}
```

以符号 # 开头的指令行在程序编译前就会由预处理器先行处理，每个预处理指令都必须占据独立的一行指令。本例

```
#include <iostream>
```

的意义是要求将掌管输入/输出数据流的头文件 (header file) <iostream> 包括进程序里面。

指令行

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

称为 **using 声明** (using declaration)，说明“cout”和“endl”是**名称空间** std 中的标准组件，即将在程序中使用。std 是 standard (标准) 的缩写，而名称空间 (namespace) 是 C++ 的一种新的机制，其目的是为了避免分成很多文件的大型程序间发生变量名称重复的问题。我们会在第 11 章介绍预处理指令，随后在第 16 章介绍名称空间。

3. **程序主体**

在下列程序中的粗体字部分是程序的主体部分：

```
// Test.cpp
// 程序的目的是在显示器上显示一行文字
#include <iostream>
using std::cout;
using std::endl;
// --  程序主体部分 ----------------------
int main()
{
  cout << "Hello, 您好!" // 主要的语句
       << endl;
  return 0;
}
```

在 main 之后的小括号 () 表示 main 是一个程序的组成单元，称为**函数** (function)。C++ 程序可能含有一个或一个以上的函数，但只能有一个叫做 main。为了在语句中强调 main 是一个函数，我们在以下的说明中将写成 main()。

不管 main() 是不是程序中第一个出现的函数，程序都是从 main() 开始执行。在 main()左边的**关键词** int 表示 main() 的执行结果将会传回一个整数，int 是 integer (整数) 的缩写。

提示

关键词 (keyword)是 C++ 语法内的保留字，具有特定的意义，不可以更动。

在 main() 后面的大括号 { } 所涵盖的部分是每个函数的本体部分 (function body)。我们将在第 6 章介绍函数的完整语法。

在大括号内的指令

```
cout << "Hello, 您好!" << endl;
```

才是这个 C++ 程序最主要的目的。完整的指令称为**语句** (statement)，每一个语句都是以分号";"作为结尾。在这个输出语句中 "Hello, 您好!" 称为**字符串**，我们使用运算符 "<<" 将这个字符串送到标准输出，也就是显示器上。

"cout"念作 C-Out，代表由头文件<iostream> 定义的一个**输出数据流对象** (output stream object)，显示器是默认的输出单元，也叫标准输出 (我们将在第十一章详细介绍"数据流")。运算符号 "<<" 称为**数据流插入运算符** (stream insertion operator)，它会将其右边的数据送进输出数据流里面，造成在显示器上显示的结果。

输出语句内最后的关键词"endl"是一种**格式操作符** (manipulators)，用来设定输出格式。格式操作符 endl 首先将换行符号 '\n' 加到输出数据流，然后强迫缓冲区内的数据立即输出。当"endl"和"cout"一起使用时，它的效果相当于将存在输出数据流内的结果立即显示出来并换行。

main()前面的"int"这个关键词与程序本体内的最后一个语句

```
return 0;
```

是对应的，表示如果程序执行到最后将传回 0，它的数据形态是 **int**，代表程序正常结束。

在新修定的 ANSI/ISO 语法标准下，上述的"**int**"和"**return 0;**"都可以省略，编译器在编译时会自动加回去。因此，程序 Test.cpp 可以写成下列更简略的形式:

```
#include <iostream>
using namespace std;
main()
{
  cout << "Hello, 您好!" << endl;
}
```

指令行

```
using namespace std;
```

称为 **using 指令** (using directive)。在 using 指令之后，所有名称空间 std 内的标准组件都开放出来，不再受限，所以不需要再使用"using std:: cout;"和"using std:: endl;"两个 using 声明。但也因此有可能发生名称重复的问题。

C++的语句可以置放于程序本体的任何地方，而且空白处的多寡可以任意设定，不会影

响程序的执行，只要程序内的各种名称不致因为紧靠在一起无法辨识即可。因此，下列的写法都可以产生正确的输出结果：

(1)

```
#include <iostream>
using namespace std;
main(){
  cout << "Hello, 您好!" << endl;
}
```

(2)

```
#include <iostream>
using namespace std;
main(){ cout << "Hello, 您好!" << endl;}
```

(3)

```
#include <iostream>
using namespace std;
main()
{
  cout << "Hello, 您好!" << endl;
}
```

不过，为了保持程序结构清楚以利于理解，我们通常将函数的本体部分相对于大括号向右缩排，而采用如 (3) 的编排方式。

延迟 DOS 窗口画面自动关闭的方法

如果是以下列的方式执行 C++ 程序：

1. 从文件管理器 (File Manager) 中找到执行文件，直接在文件名上以鼠标连按两下，或是
2. 在 Microsoft Visual C++ 或 Borland C++ Builder 等窗口开发环境下执行具有 cout 显示指令的程序。

则执行所产生的 DOS 画面会在执行完毕后自动关闭。除非程序内容包括存盘等动作，

否则执行结果将会一闪而逝，不易看到。为了延迟执行所产生的 DOS 画面自动关闭，可以使用下述方式修改程序：

```
// TestW.cpp
// 延迟 DOS 画面自动关闭
#include <iostream>
#include <conio>
using std::cout;
using std::cin;
using std::endl;

// --  程序主体部分 ----------------------
int main()
{
  cout << "Hello, 您好!" << endl;
  getch();
  return 0;
}
```

在上述程序中我们以

```
#include <conio>
```

要求预处理器将掌管键盘和显示器 (con 为控制台 console 的缩写) 输入/输入方式的头文件 **<conio>** 包括到程序里面来，并在程序结尾处加上等待使用者从键盘输入字符的指令“`getch();`”，以保留显示结果。在其它部分执行结束后，将会等待使用者触动键盘才会完全结束程序，关闭 DOS 画面。

图 2.2.1 即是使用上述编译过后的执行文件 TestW.exe，从 Windows 资源管理器直接在文件名 TestW.exe 上以鼠标连按两下的结果。

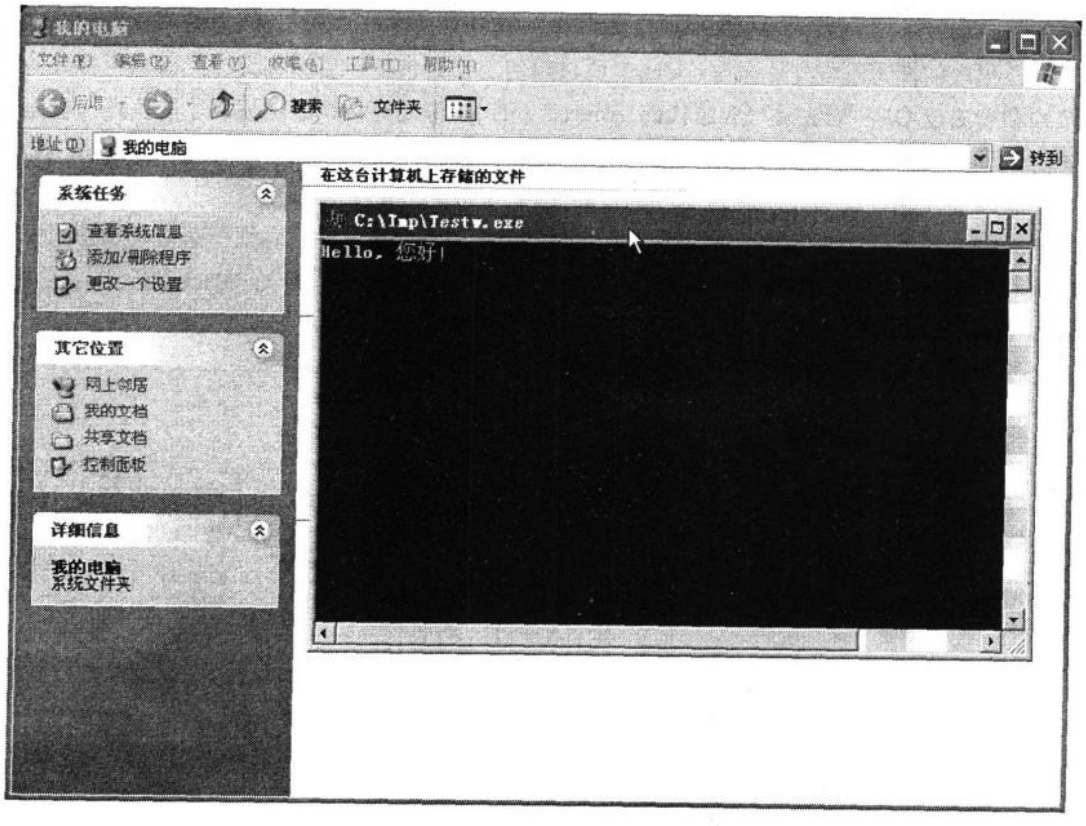

图 2.2.1　从 Windows 资源管理器直接以鼠标执行 TestW.exe

2.3　Borland C++编译器的取得和安装使用

由于 C++的重要性与日俱增，目前可用的编译器 (compiler) 及开发环境种类很多。例如 Microsoft Visual C++，Borland C++ Builder，Watcom C++，Fujitsu C++ compiler，Dev-CH，IBM Visual Age C++，GNU C++ compiler，KAI C++，HP aCH，DJGPP，GNU-Win32，TenDRA 等，只要是符合 ANSI/ISO 标准的 C++编译器都能正确地处理本书所介绍的 C++程序。

不管我们采用哪一种开发环境，都依循

Edit-Compile-Run

也就是“撰写-编译-执行”的循环。

在程序的**撰写**阶段 (又称为 coding) 所进行的工作是依工作目标和设想的概念将拟好的结构和算法以 C++ 语法撰写成**源代码** (source code)。源代码必须经由编译才能执行。在编译阶段难免因为语法或输入过程的疏忽以致无法成功编译，这时就必须回头重新修改程序，再重新编译。编译成功，才能得到**可执行码** (executable code)，可以用来执行。但执行的结果未必符合原先的预期，这时就又必须回到程序的撰写和修改，甚至必须重新修正概念以及由其衍生的程序结构和算法。也就是说，编译器只能找出**语法**的错误 (syntactic error)，无法找出**语义**错误 (semantic error)。

这个过程可以用图 2.3.1 清楚地表达：

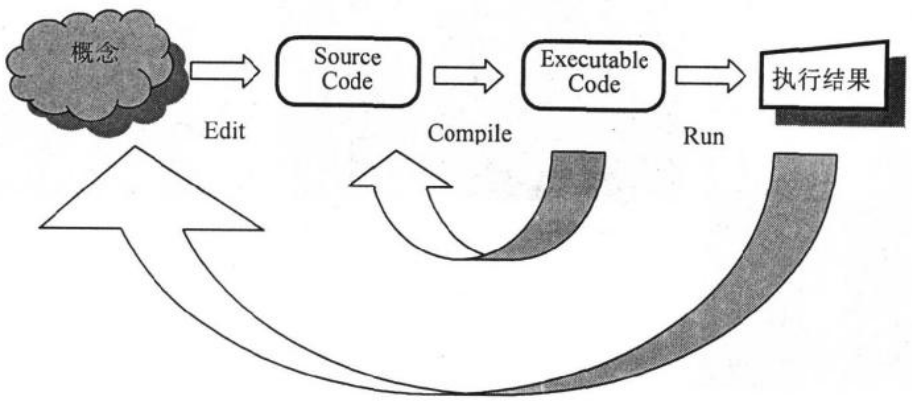

图 2.3.1 C++程序开发流程图

Source Code 的撰写可以使用任何文字编辑软件，如“记事本”、“WordPad”、Text Editor 等，也可使用商用程序开发环境内附的编辑器。使用上述“记事本”或“WordPad”时，要特别注意存盘的动作，避免被自动加上 .txt 的扩展名。

在众多专业编辑软件里，我们比较推荐 IDM Computer Solutions, Inc. 的 **UltraEdit**，不仅在于它可以正确地处理扩展名，可以同时开启多个文件，以供参照、对比，而且在于它可以使用不同的颜色以区分各种变量、数字、关键词、说明文字等内容，增进程序撰写的正确性和效率。我们甚至把 UltraEdit 的编辑画面设定文件 wordfile.txt 放在随书附赠的程序里，

只要将它复制到

```
C:\Program Files\UltraEdit\
```

就可以马上获得经过我们精心调整的编辑画面。它会自动区别扩展名为 .h 和 .cpp 的 C++ 程序文件，并正确显示。 UltraEdit 的试用版可以从下列网站取得:

```
http://www.idmcomp.com
http://www.ultraedit.com
```

Borland C++ Compiler 5.5 的下载、安装及使用

本节所描述的编译器是 Borland 公司所提供的免费下载程序。使用这个编译器的好处除了免费外，最大的好处是 Borland 公司是 C++ 语法标准的主要制定成员之一，并不断在网络上提供最新的修订版本，可担保我们使用最可靠的编辑器。此外，该程序非常精简，只有不到 10MB 的大小，却足供 C++语言的深入探讨和练习，也能开发许多有用的应用程序。

我们已将此编译器放在随书所附光盘内。读者如果要自行下载，首先必须进入 Internet，到下列网址

```
http://www.borland.com/bcppbuider/freecompiler/cppcssstemps.html
```

下载前需先填入简单的个人数据，如姓名、电话、传真号码及 e-mail 地址等，并按下对于安装程序版权的“同意”按钮。文件的名称是 freecommandLinetools.exe，大小为 8.727 MB。

当我们下载完毕后，在 Windows 的“资源管理器”里，直接点选本执行文件，在其上按两下鼠标左键，就可自动解压缩完成初步安装。其预设安装位置为 C:\Borland\Bcc55，安装后的大小为 50.7MB。

要完成所有的安装，还需要进行以下两个步骤:

(1) 将 C:\Borland\Bcc55\Bin 加入系统文件预设路径。亦即修改 C:\下的 autoexec.bat 文件，加入

```
PATH=%PATH%;C:\Borland\Bcc55\Bin;
DOSKEY
```

两行指令。设定了路径后，任何地方皆可存放和执行 C++程序。此外，第二行DOSKEY是为了便于在DOS下使用方向键,以减少重复输入指令的动作。

(2) 制作 bcc32.cfg 和 ilink32.cfg 两个配置文件，并存放于文件夹 C:\Borland\Bcc55\Bin 里。其中 bcc32.cfg 为编译器的编译选项指定文件，内含下列两行指令：

```
-I"C:\\Borland\Bcc55\include"
-L"C:\Borland\Bcc55\lib"
```

而 ilink32.cfg 则为连接器的连结选项指定文件，含下列单行指令：

```
-L"C:\Borland\Bcc55\lib"
```

为了读者方便，我们已将这两个文件准备好，只要将本书所附光盘内的文件直接复制到上述路径即可。如果要自己制作，同样的，要注意“记事本”或“WordPad”编辑的文件通常会再被加上.txt 的扩展名，造成如 bcc32.cfg.txt 的错误，必须删除多余的扩展名才能正确运作。

完成上述两个步骤后，只要重新开机就可以正常使用。详细的安装说明可参照光盘内的“安装程序.txt”，详细的使用说明则可以阅读安装后的 readme.txt 和 bcb5tool.hlp 两个文件。

除了上述的编译器外，我们还可以从 Borland 公司的网址：

http://www.borland.com/bcppbuilder/turbodebugger/turbodebug55steps.html

下载执行文件 TurboDebugger.exe 安装，以配合 Borland C++ Compiler 使用，协助复杂程序的调试工作。本执行文件大小为 590KB。其工作原理为：“可以自由设定暂停位置，并能在暂停或因错误而中止时，进入内存检查各个变量储存的数据，以发现错误发生的位置和原因。”

如果已经安装了 Borland C++ Builder 5.0 之后的版本

如果你已经安装了 Borland C++ Builder 5.0 以上的版本，并在安装后重新开机过，则上述各步骤，包括 autoexec.bat 路径的设定，以及 bcc32.cfg 和 ilink32.cfg 两个文件的设置，都会自动安排妥当，可以立即以上述的方式在 DOS 下操作。顶多只要在 autoexec.bat 文件内加上 DOSKEY 的指令以方便操作即可。

迅速进入 DOS 模式和正确路径的设定方式

虽然本编译器为 32 位的版本，但只能在 DOS 的环境下操作。DOS 最为人诟病之处就在于必须以键盘下指令。我们在 autoexec.bat 内加入的 DOSKEY 设定虽然已经能够大量减少

使用键盘重复输入相同指令的动作，但如果每一次进入 DOS 模式，仍然必须下指令以移到源程序文件所在的路径，将不胜其烦。

为了解决这个问题，我们需要使用“资源管理器”到 C:\WINDOWS\下找到“DOSPRMPT”的图标或到 C:\WINDOWS\PIF\COMMAND 下找到“COMMAND”的图标：

如果是 Windows 2000 或 Windows XP，则是到 C:\WINNT\system32\或 C:\WINDOWS\system32\下找“CMD.exe”

将其复制到我们存放程序的文件夹内。从此，只要在“资源管理器”内，在这个文件夹内的 DOS 图标上连按两下，就可迅速进入 DOS 模式，同时也位于正确的路径上了。我们已将这两个文件准备好，只要找到本书所附 CD 内的文件，直接复制使用即可。

以上述 test.cpp 为例，使用 Borland 编译器在 DOS 下编译和执行，整个操作过程的画面如图 2.3.2 所示：

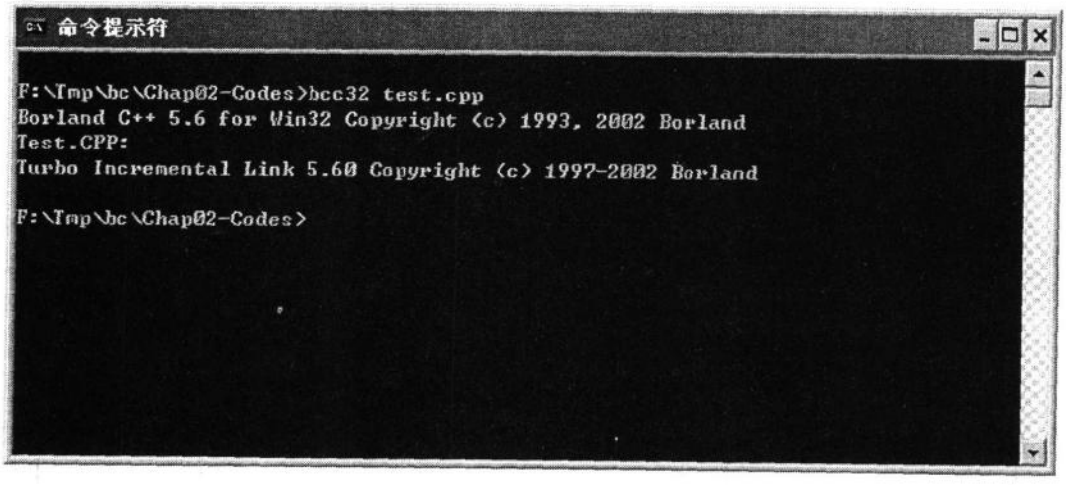

图 2.3.2　使用 Borland 编译器在 DOS 下编译和执行 test.cpp 的画面