第 2 章　C/C++编程环境与调试

【问题原由】不论学习何种语言，不仅要掌握其语法规则并能进行算法程序设计，而且还要掌握该语言程序在不同开发平台上进行编辑、编译、运行与调试程序的方法。C/C++是如何实现这些过程的呢？这就是本章所要讨论的问题。

【知识要点】在 Turbo C++ 3.0 与 Visual C++ 6.0 环境下进行 C/C++编程、编译、运行与调试等。

【能力要求】熟悉 Turbo C++ 3.0 与 Visual C++ 6.0 工作环境；了解 C/C++程序设计常见的错误类型；掌握查找程序错误和调试程序的基本方法。

§2.1　Turbo C++ 3.0 编程环境

美国 Borland 公司于 1989 年推出了 Turbo C 2.0，并在继承和发展 Turbo C 2.0 集成开发环境的基础上，于 1991 年推出了 Turbo C++ 3.0。Turbo C++ 3.0 实际上是 Turbo C 的一个超集，不仅包含了 Turbo C 2.0 的全部功能，而且包含了面向对象的基本思想和设计方法。

Turbo C++ 3.0 是集程序的编辑、编译、连接、调试为一体的 C/C++程序开发工具软件，它拥有丰富的库函数和良好的用户界面，支持 Windows 操作系统，因而深受 C/C++编程者的青睐。具体说，Turbo C++ 3.0 开发环境具有以下特点。

（1）支持鼠标操作：就像在 Windows 环境下一样，方便地用鼠标进行单击、拖放等操作。

（2）支持多窗口操作：允许同时打开多个窗口，并且能够方便地进行操作切换。

（3）编辑方便：可以用鼠标选择一行或多行程序代码，然后通过菜单进行剪切、复制、粘贴等操作。

（4）适应性好：Turbo C++ 3.0 是基于 DOS 平台的 C/C++编译系统，只要求很小的系统资源，在几乎所有的微机上（包括 286、386、486、586 等）都可以使用。

（5）系统功能强：Turbo C++ 3.0 提供了比较丰富的功能函数，如字符串处理函数、文件操作函数、BIOS 中断函数、DOS 中断函数、图形函数等。

正是由于 Turbo C++ 3.0 具有速度快、效率高、功能强、适应性好等优点，很受用户欢迎，特别适合 C/C++语言的教学和初学者学习。

2.1.1　Turbo C++ 3.0 的安装

Turbo C++ 3.0 系统是以压缩文件的形式存放在磁盘或光盘上的，用户在使用 Turbo C++ 3.0 之前，必须将厂商提供的 Turbo C++ 3.0 系统按照要求合理地安排在用户的磁盘上，以建立一个 Turbo C++ 3.0 的使用环境，这项工作称为 Turbo C++ 3.0 的安装。

1. *启动安装程序*

Turbo C++ 3.0 系统提供了一个安装程序 install.exe，运行 install.exe 程序即可开始安装。在安装过程中，用户可以根据系统显示在屏幕上的提示进行操作，可以指定存放系统文件的目

录和存储模式，但一般都不必自己指定，而采用系统提供的默认方案。如果采用系统提供的默认方案，则在安装完成后，用户的磁盘（一般为 C 盘）上将会增加以下子目录和文件。

（1）C:\TC\BIN：其中包括 tc.exe、tlink.exe、make.exe 等执行文件。

（2）C:\TC\INCLUDE：其中包括各种库函数的头文件。

（3）C:\TC\LIB：其中包括各种库函数编译库文件。

（4）C:\TC\BGI：其中包括图形操作库文件。

（5）C:\TC\DOC：其中包括 TC 的技术文档。

（6）C:\TC\CLASSLIB：其中包括支持 C++的各种文件。

（7）C:\TC\EXAMPLE：其中包括各种例程。

读者可以在 C:\TC\找到一个文件为 Filelist.doc，其中列出了所有的安装文件。

2. 进入 Turbo C++ 3.0 工作窗口

在完成上述安装后，在 TC\BIN 主目录下已经存放了 tc.exe、tcc.exe 两个执行文件。其中 tc.exe 是将编辑、编译、连接、调试和运行集成为一体的基本模块；tcc.exe 则提供了某些补充功能，例如可以在程序中嵌入汇编代码等。一般情况下只需用到 tc.exe，操作方法如下。

（1）通过“资源管理器”找到文件夹 TC\BIN 中的 tc.exe 文件：双击该文件名，即可进入 Turbo C++ 3.0 环境。

（2）通过“资源管理器”找到文件夹 TC\BIN：将其中的 tc.exe 文件创建为“快捷方式”，然后把它拖到桌面上，在桌面上将出现一个“快捷方式到 tc.exe”的图标。以后每次想进入 Turbo C++ 3.0 环境时，只需双击该图标即可。

Turbo C++ 3.0 的使用有两种形式，一种是窗口式，另外一种是全屏式，在 Windows NT/2000/XP 下，通过按 Alt+Enter 键可以在这两种模式之间切换，推荐使用全屏的模式，这样速度会快一些，并且比较兼容。由于 Windows NT/2000/XP 是 32 位的系统，而 Turbo C++ 3.0 的 DOS 平台是 16 位的系统，因此 Windows NT/2000/XP 下的 DOS 环境是模拟的，比较占用 CPU，一般不要开两个 Turbo C++ 3.0 窗口，这样会使系统变得较慢。

3. Turbo C++ 3.0 工作窗口布局

启动 Turbo C++ 3.0 后，显示集成环境屏幕主界面，如图 2-1 所示，它由三部分组成。

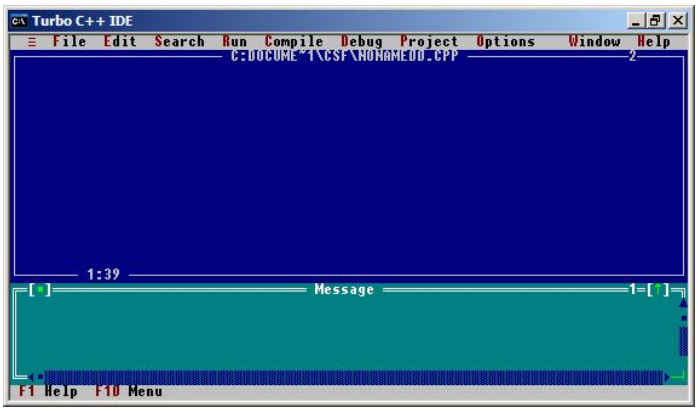

图 2-1 Turbo C++ 3.0 初始屏幕界面

（1）主菜单窗口：屏幕顶部包括 10 个主菜单：File、Edit、Search、Run、Compile、Debug、Project、Options、Window、Help。主菜单各项的主要功能如表 2-1 所示。

表 2-1　Turbo C++ 3.0 主菜单功能表

项目	功能
File（文件）	调入和存储文件、管理目录、调用 DOS 和退出 TC 环境
Edit（编辑）	进入编辑状态，用户可以编辑和修改当前编辑窗口中的源程序
Search（查找）	查找源程序文件
Run（运行）	控制程序的运行方式，编译、连接和运行当前程序
Compile（编译）	编译当前环境内的程序，生成目标和可执行程序
Debug（调试）	调试程序，显示变量的值，查找函数，查看调用栈的状态
Project（项目）	处理由多个源程序文件组成的工程文件
Options（选项）	设置有关编译和连接的选项
Window（窗口）	窗口切换
Help（帮助）	为用户进行上述操作提供在线帮助

（2）编辑主窗口：在主菜单窗口下面，编辑主窗口可以显示各种子窗口，包括源程序代码编辑窗口、编译信息窗口、调试信息窗口和输出窗口等。

（3）提示信息行：位于屏幕最下方，显示一些功能键的作用和菜单的使用提示。例如，F1 Help（帮助）：任何时候按 F1 键都会显示帮助信息；F2 Save（保存）：快速保存源文件；F3 Open（打开）：快速打开源文件；Alt+F9 Compile（编译）：快速编译；F9 Make（生成）：直接编译和连接生成可执行文件；F10 Menu（菜单）：回到主菜单，激活菜单≡（系统）。

2.1.2　输入和编辑源程序

1. 编辑一个新文件

如果要输入和编辑一个新的程序，用鼠标选择主菜单中的 File 菜单的子菜单项 New，在编辑主窗口就出现了一个新的编辑子窗口，光标定位在该子窗口的左上角，如图 2-2 所示。

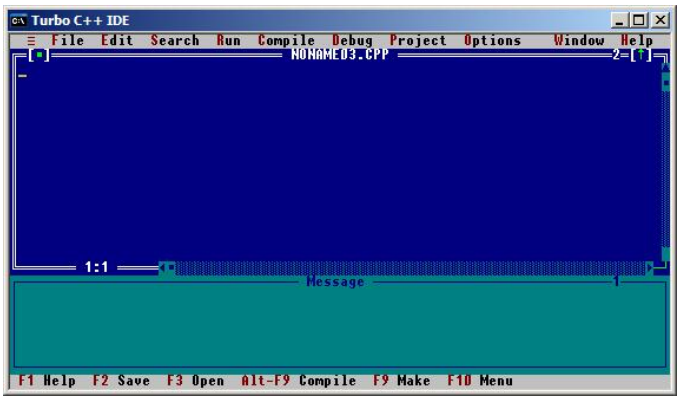

图 2-2　编辑子窗口示意图

该子窗口的左上方有一个绿色方形标志[■]，用鼠标单击该标志即可关闭该窗口，同时也关闭了正在编辑的源程序；该子窗口的右上方有一个绿色箭头[↑]，用鼠标单击该标志可以使该子窗口最大化，最大化的子窗口占据了全部的编辑主窗口的空间；在该子窗口的左下部显示了当前光标的位置1:1，即光标所在位置是第 1 行、第 1 列；这个子窗口右边框和下边框各有

一个滚动条，用鼠标可以拖动滚动条，窗口中的编辑内容也随之上下和左右滚动。

用户在新建的编辑子窗口中可以开始输入和编辑源程序。Turbo C++ 3.0 提供了一个全屏幕编辑环境。用户将已编好的源程序逐行输入，如发现错误可随时修改。在编辑过程中除用到各字符键外还可以用到 Insert 和 Delete 键。Insert 键是一个切换键，用来控制工作状态是否为“插入状态”。按下 Insert 键后，可以看到光标变成一个闪烁的横杠，这时从键盘输入的字符（包括控制字符，如“回车”）会插入到屏幕当前光标处，光标后的字符会自动顺序后移；如果再按一下 Insert 键，则取消插入状态，光标变成一个闪烁的实心方块，此后输入的字符将覆盖（而不是插入）光标处的字符。Turbo C++ 3.0 设置的初始状态是“插入状态”。第一次按 Insert 键改成“覆盖状态”，再按 Insert 键则又改为“插入状态”，如此反复切换。Delete 键是删除光标所在的字符。

输入程序后应对程序作认真检查，并改正已发现的错误。这时应及时将源程序保存起来，按 F2 键或者用鼠标选择 File→Save 菜单，Turbo C++ 3.0 就会弹出一个对话框，要求用户指定文件名，默认的文件名为 NONAME00.CPP，意为“未命名”。一般说，不建议以 NONAME00.CPP 作为自己的文件名，它不仅不便于辨别，而且每次都用 NONAME00.CPP 作文件名，第一次保存的文件的内容就会被第二次保存的文件的内容取代。因此，在对话框中选择合适的路径并给出恰当的文件名进行保存。其操作界面如图 2-3 所示。

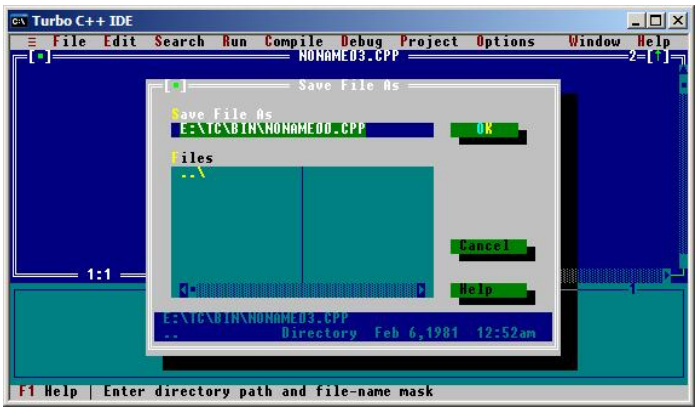

图 2-3　文件保存示意图

2. 编辑一个已存在的文件

假如上次编辑的源文件需要进行编辑，就需要把它从磁盘中调出来。按 F3 键或者用鼠标选择 File→Open 菜单，这时，屏幕上会出现一个“打开文件”对话框，要求用户输入准备调入的文件路径和文件名。输入路径和文件名即可打开源文件。如果记不清所要装入的源文件名，想看一下当前目录中有哪些源文件，则可以在对话框的 Name 输入栏中输入*.CPP，然后直接按回车键，此时 Turbo C++ 3.0 就会显示当前目录下所有后缀为 CPP 的文件名。选择需要装入的文件名，按回车键或单击 Open 按钮，该文件的内容即显示在屏幕上，供用户编辑、修改，如图 2-4 所示。

另外，在“打开文件”对话框的 Files 列表框中，用鼠标或光标可以打开当前目录下的子目录或进入上一级目录。

这时用户可以对该程序进行修改，然后按 F2 键存盘。如果想以另一个文件名存盘，可以单击 File 菜单中的 Save As，按回车键后弹出一个“另存为”对话框，在对话框的 Files 列表

框中选择合适的路径，在 Name 文本框中输入文件名进行保存。

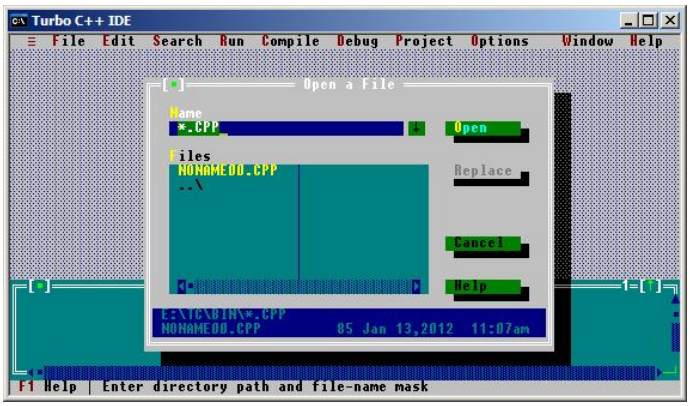

图 2-4　“打开文件”对话框示意图

2.1.3　编译、连接和运行

编辑好源程序并存盘后，应当对源程序进行编译、连接和运行。在 Turbo C++ 3.0 集成环境中，进行编译、连接和运行是十分方便的，既可以将编译、连接和运行分三个步骤分别进行；也可以将编译和连接合起来作为一步进行，然后再运行；还可以将编译、连接和运行三者合在一起一次完成。

在 Turbo C++ 3.0 中既可以对单个文件模块的程序进行编译、连接和运行，也可以一次对多个文件模块的程序进行编译、连接和运行。

1. 对单文件程序进行编译和连接

首先是对单文件程序进行编译，选择 Compile→Compile，或者按 Alt+F9 键，即可对源文件进行编译，如果当前被编译的源程序文件名为 test.cpp，系统就自动将目标文件名定为 test.obj。编译后将生成的目标文件 test.obj 保存在当前工作目录或配置文件 TCCONFIG.TC 所指定的输出目录中，如图 2-5 所示。

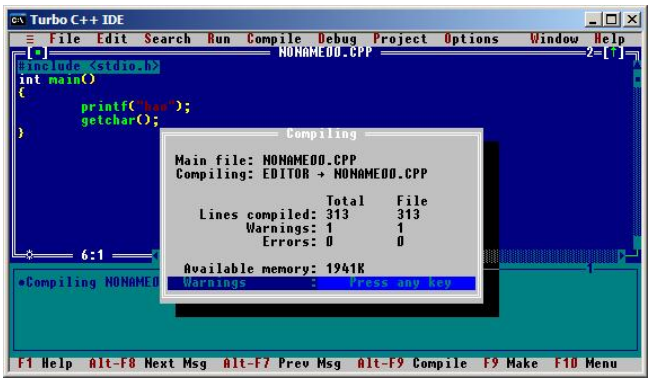

图 2-5　对源文件进行编译

然后对单文件程序进行连接。有了目标文件后，还不能直接运行，还要将目标文件与系统提供的库函数和包含文件等连接成一个可执行文件（后缀为 exe），才能运行这个 exe 文件。选择 Compile→Link，（或按 Alt+C 键后再按 L 键），就可以进行连接，在当前工作目录或配置

文件 TCCONFIG.TC 所指定的输出目录中生成一个可执行文件 test.exe，如图 2-6 所示。

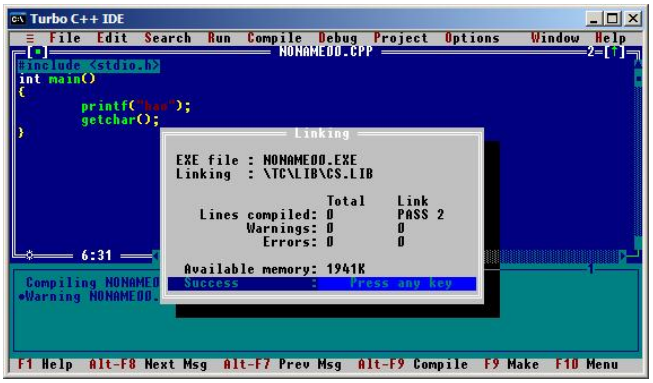

图 2-6　对目标文件进行连接

【注意】必须先进行编译，得到 obj 文件后才能进行连接，否则会出错。

2. 一次完成编译和连接

（1）对单个文件程序进行编译和连接：选择 Compile→Make，就可一次完成编译和连接，在当前工作目录或配置文件 TCCONFIG.TC 所指定的输出目录中生成一个目标文件和一个可执行文件。在 Turbo C++ 3.0 的信息窗口下面有一个“提示信息行”，列出了键盘上各功能键的作用。其中的 F9 Make 表示：按 F9 键相当于选择了 Compile 菜单中的 Make 子菜单，即一次完成编译和连接。用户既可以按以上介绍的通过选择菜单进行编译和连接，也可以直接按 F9 键进行编译和连接。

（2）对多文件程序进行编译和连接：如果一个源程序包含多个文件模块，则应当对各文件分别进行编译，每个源文件编译后生成一个 obj 文件，得到多个 obj 文件，然后将这些目标文件以及库函数连接成一个可执行文件。Turbo C++ 3.0 提供了对多文件程序进行编译和连接的简便方法。要将这些文件组成一个项目（Project），则要建立一个“项目文件”，在该文件中包含各文件的名字，然后将该项目文件编译和连接，得到可执行 exe 文件。具体步骤如下：

① 从\TC\EXAMPLES 目录中找到一个 PRJ 文件，如 EX9.PRJ，将这个文件复制到自己的工作目录下，并重命名为 TEST.PRJ。

② 进入 Turbo C++ 3.0 环境，选择 Project 项目，然后选择 Open project 子菜单，按回车键弹出“打开项目”对话框，如图 2-7 所示。

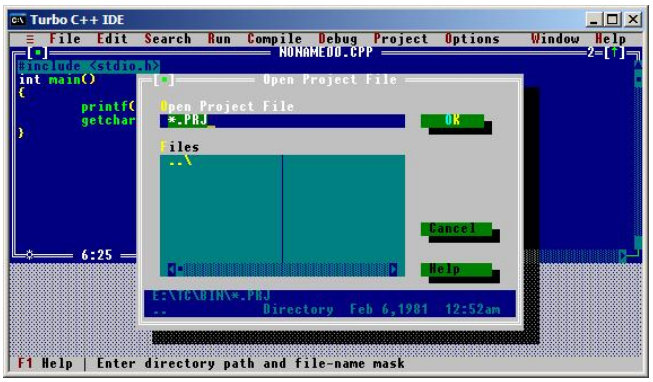

图 2-7　“打开项目”对话框

在 Open Project File 输入框中输入 TEST.PRJ 按回车键，或者直接用鼠标双击 Files 列表中出现的 TEST.PRJ 文件名，此时 TEST.PRJ 文件中的内容被 Turbo C++ 3.0 读入，如图 2-8 所示。

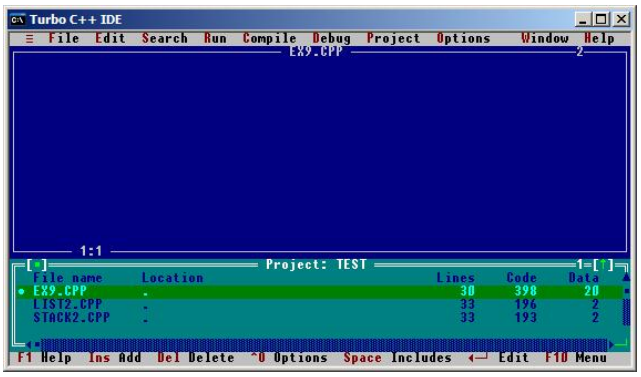

图 2-8　打开的项目列表

在编辑主窗口下方出现一个项目列表窗口 Project:TEST，列表中有三个源文件，这是原来 EX9.PRJ 中所保存的文件，现在需要删除这些文件项，然后添加自己的文件项。

③ 选择列表窗口 Project:TEST 中的要删除的源文件，再选择 Project→Delete item 菜单，则删除选中的文件项 STACK2.CPP，如图 2-9 所示。

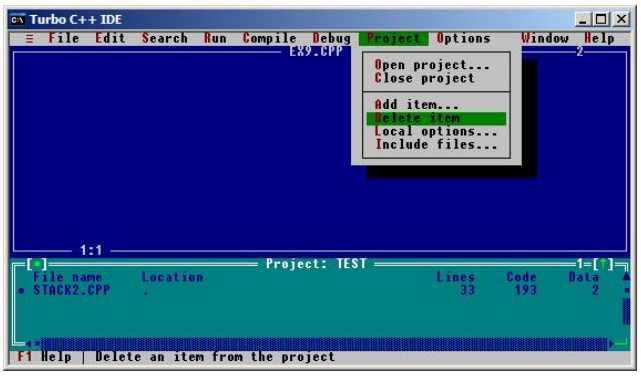

图 2-9　删除一个文件项

接着把另外两个文件项删除，然后选择 Project→Add item 菜单，弹出“添加文件项目”对话框，如图 2-10 所示。

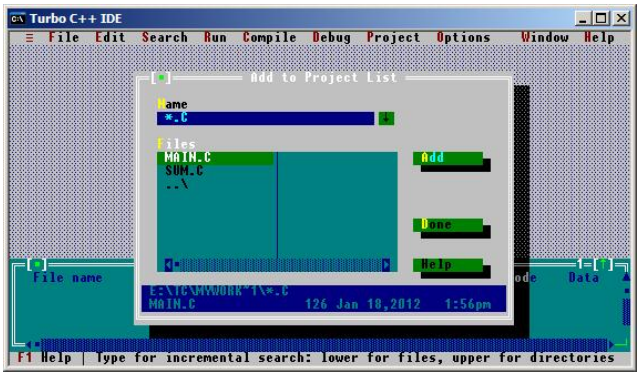

图 2-10　“添加文件项目”对话框

在这个对话框中，用鼠标选择路径，找到需要添加的文件项目，然后选中并单击 Add 按钮，则所选文件就被加入到这个项目中了，添加完若干个文件后，单击 Done 按钮关闭这个对话框。这里选择添加了两个文件 MAIN.C、SUM.C。

④ 重新在 Options→Directories 中设置好工作环境路径。

⑤ 然后选择 Compile→Make 或按 F9 键进行编译连接，生成可执行文件 TEST.EXE。

【注意】在处理完一个多文件程序的编译和连接后，应及时将 Project 项清空（选择 Project →Close project 菜单），否则就会在编译连接时仍然把项目文件 PRJ 当做编译的对象，而不是编译编辑窗口中的源文件。

2.1.4 建立文件工作区域

1. 改变用户工作目录

工作目录指用户编辑的源文件所在的目录。为了管理上的方便和安全一般不应该将不同的人、不同性质和用途的程序混放在一起，而应分别建立子目录。例如几个学生先后共用一台微机，应该为每一学生设一个专用的子目录。如果同一个学生既学习 C 语言，又学习 QBASIC 语言，则应该分别设立两个子目录。这样不同的学生可以在不同的子目录下进行文件的编辑工作，编译生成的目标文件也存放在此子目录中。具体方法如下：选择 File 菜单下的 Change Dir，就会出现一个“改变目录”对话框，提示用户输入所选择的工作目录名，如图 2-11 所示。

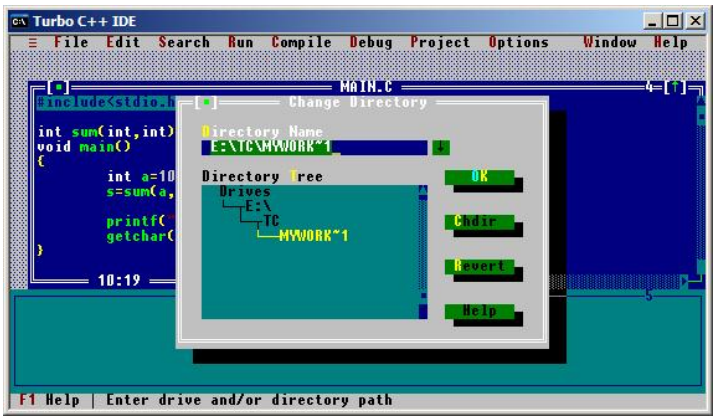

图 2-11 改变工作目录示意图

在图 2-11 中可以看到，系统显示出的目录是 E:\TC\MYWORK~1，这是未修改前的用户当前工作目录。若想把它改为 E:\TC\MINE，此时只需在“改变目录”对话框的 Directory Name 输入框中把目录改为 E:\TC\MINE，或者用鼠标在 Directory Tree 中选择 MINE 然后双击，也可以用鼠标在 Directory Tree 中选择 MINE 后单击 Chdir 按钮，这样一来，在保存源文件和输出文件（obj 文件和 exe 文件）时，如不另外指定，将自动保存在该子目录中。

【注意】在“改变目录”对话框中输入的子目录名必须是已存在的目录，如果不存在此目录，则系统会显示出错信息，用户可再次输入有效的目录名。

2. 确立 Turbo C++ 3.0 工作环境

用上面方法指定的工作目录，可以用来保存源文件和输出文件，但是有时人们进一步希望把源文件和输出文件分别保存在两个子目录中，例如想把编译后的目标文件（后缀为 obj 的文件）和连接后产生的可执行文件（后缀为 exe 的文件）保存在 E:\TC\MYEXE 子目录中。

这就要通知 Turbo C++ 3.0 把这些输出文件存于何处。

此外，还要告诉 Turbo C++ 3.0 包含文件和库函数在哪个子目录中。在安装 Turbo C++ 3.0 时，如果用户不作另外的指定，系统会按照默认的方案建立一个\TC\BIN 目录用来存放 Turbo C++ 3.0 的系统文件，同时在 TC 主目录下建立一个 INCLUDE 子目录用来存放包含文件，另外建立一个 LIB 子目录用来存放库函数。如果在安装 Turbo C++ 3.0 时，用户不采用系统提供的子目录名和安排，而自己另外指定了子目录名，则应通知 Turbo C++ 3.0 系统，以免找不到安装后的执行文件。

3．Turbo C++ 3.0 工作环境设置

以上这些工作属于确立 Turbo C++ 3.0 系统的工作环境。为了确立 Turbo C++ 3.0 系统的工作环境，需要利用主菜单上的 Options 菜单。首先在主菜单窗口上用左右键将亮块移到 Options 处，按回车键显示出下拉菜单（也可用 Alt+O 键），选择 Directories 子菜单，如图 2-12 所示。

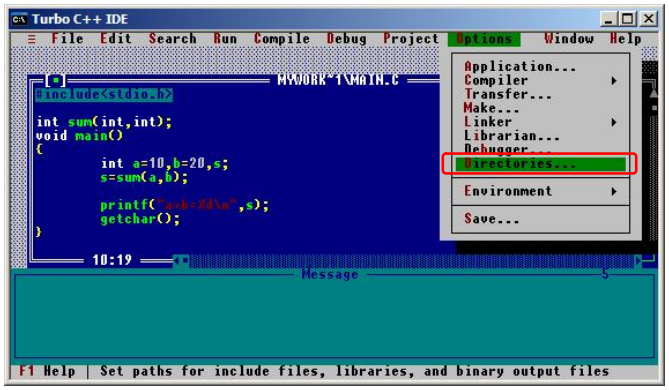

图 2-12　选择 Directories 子菜单

按回车键，弹出一个如图 2-13 所示的窗口。

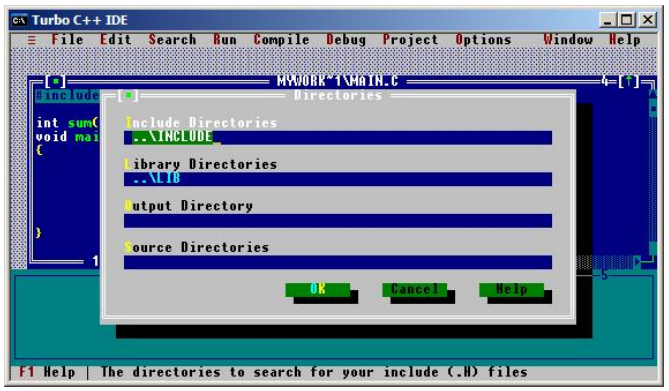

图 2-13　设置 Turbo C++ 3.0 工作环境对话框

窗口中各项的含义如下：

- Include Directories：包含文件所在的目录。
- Library Directories：库函数文件所在的目录。
- Output Directory：存放*.exe 和*.obj 等输出文件的目录。
- Source Directories：源代码文件所在的目录。

第 1 个输入框 Include Directories，是指包含文件所在的目录，默认情况下是系统默认的子目录 E:\TC\INCLUDE。

第 2 个输入框 Library Directories 指库文件所在的目录，图 2-13 中显示了系统默认的子目录 E:\TC\LIB。

第 3 个输入框 Output Directory 指定了输出文件（obj 文件和 exe 文件）存放的目录，图 2-13 显示的是 E:\TC\MYWORK~1，如果用户不指定这个目录，系统就会将它保存在当前工作目录中。现在想修改输出文件目录，在此框内输入 E:\TC\MYEXE，表示要将 obj 文件和 exe 文件保存在 E:\TC\MYEXE 目录中。

第 4 个输入框 Source Directories 指源代码文件所在的目录，若不指定，则默认为当前工作目录。

在进行以上设置后，还应当把这些信息保存起来。保存的办法是把环境信息记录在一个特定的文件中，这个文件称为配置（config）文件。Turbo C++ 3.0 给此文件指定一个默认名 TCCONFIG.TC。在每次启动 Turbo C++ 3.0 时，系统会从当前工作目录和 Turbo C++ 3.0 系统文件所在的目录中寻找 TCCONFIG.TC 文件，如果找到了就把它调入内存。

建立配置文件的方法是在上述设置完毕后单击 OK 按钮退出菜单状态并返回编辑窗口，然后用鼠标选择 Options 菜单的 Save 子菜单，并按回车键，此时会弹出一个“配置文件保存”对话框，如图 2-14 所示。

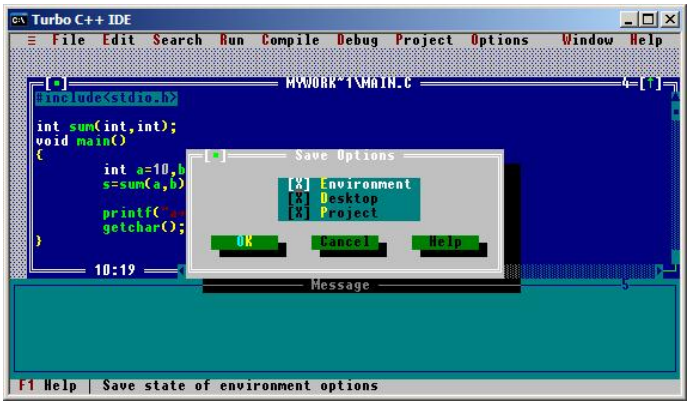

图 2-14　保存 Turbo C++ 3.0 工作环境

在对话框中有三个选项 Environment、Desktop 和 Project，必须保证 Environment 选项被选中，单击 OK 按钮后即可将配置信息保存在 BIN 目录下的配置文件 TCCONFIG.TC 中。如果 Desktop 选项被选中，则有关编辑窗口和当前打开的源文件的信息也都会被记录到配置文件 TCCONFIG.TC 中，下次打开 Turbo C++ 3.0 时会自动调入这些源文件并打开相应的编辑窗口。如果 Project 选项被选中，则有关工作项目的信息也会被保存起来。

对大多数学生来说，不必每次上机前都重新设置和建立配置文件。实际上，机房人员已经根据需要安装和设置好了。在学生上机练习时，先进入自己建立的子目录，然后在这个子目录中用 DOS 命令 E:\TC\BIN\TC.exe（E:\TC\BIN\是可执行文件 TC.exe 所在的目录）调入 Turbo C++ 3.0，这时系统就会自动将源程序和 obj 文件以及 exe 文件都存放在此工作目录中，学生不必作任何设置。如果是从 Windows 平台通过双击 Turbo C++ 3.0 图标（快捷方式），或从 TC 文件夹中执行 tc.exe 而进入 Turbo C++ 3.0 环境，则应通过 File→Change Dir 选项改变工作目录。

§2.2　Visual C++ 6.0 编程环境

C/C++语言程序可以在 Turbo C++ 3.0 中编辑、调试和编译，也可以在 Visual C++中编辑、调试和编译。Visual C++是微软公司针对 C++语言推出的编译系统，并已成为系列产品。

Visual C++ 6.0 是基于 Windows 平台的可视化集成开发环境，也是目前使用极为广泛的可视化软件开发工具。由于它占有很大的市场，因而已有公司推出 Visual C++ 6.0 汉化版，但并不是真正的中文版 Visual C++，只是把菜单汉化了，而且汉化的不准确。本书仍以 Visual C++ 6.0 英文版为背景来介绍 Visual C++的编辑、调试和编译。事实上，Visual C++的不同版本的上机操作方法是大同小异的，只要掌握了其中的一种，就能顺利地使用其他版本。

2.2.1　Visual C++ 6.0 的安装

如果计算机中未安装 Visual C++ 6.0，则应先安装 Visual C++ 6.0。Visual C++是 Visual Studio 的一部分，因此需要找到 Visual Studio 的光盘，执行其中的 setup.exe，并按屏幕上的提示进行安装即可。安装结束后，在 Windows 的“开始”菜单的“程序”子菜单中就会出现 Microsoft Visual Studio 子菜单。

在需要使用 Visual C++时，只需从桌面上顺序选择“开始”→“程序”→Microsoft Visual Studio→Visual C++ 6.0 即可，此时屏幕上在短暂显示 Visual C++ 6.0 的版权页后，出现 Visual C++ 6.0 的主窗口，如图 2-15 所示。

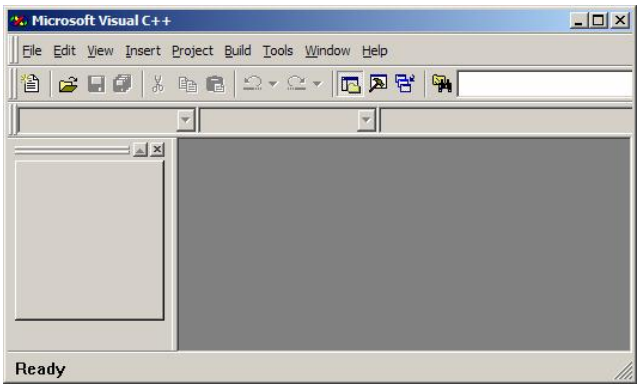

图 2-15　Visual C++ 6.0 的主窗口

此外，也可以先在桌面上建立 Visual C++ 6.0 的快捷方式的图标，这样在需要使用 Visual C++时只须双击桌面上的该图标，即可弹出 Visual C++主窗口。

主窗口的左侧是项目工作区窗口，右侧是程序编辑窗口。工作区窗口用来显示所设定的工作区的信息，程序编辑窗口用来输入和编辑源程序。

在 Visual C++主窗口的顶部是 Visual C++的主菜单栏，它包含 9 个菜单项：File（文件）、Edit（编辑）、View（查看）、Insert（插入）、Project（项目）、Build（构建）、Tools（工具）、Window（窗口）、Help（帮助）。

每次完成了对程序的操作后，必须安全地保存好已经建立的应用程序与数据，并用关闭工作区命令来终止工程。

2.2.2 输入和编辑源程序

先介绍最简单的情况，即程序只由一个源程序文件组成，即单文件程序（有关对多文件程序的操作在本章的稍后介绍）。

1. *新建一个 C 源程序的方法*

在 Visual C++ 6.0 主窗口的主菜单栏中选择 File（文件），然后在其下拉菜单中选择 New（新建），如图 2-16 所示。

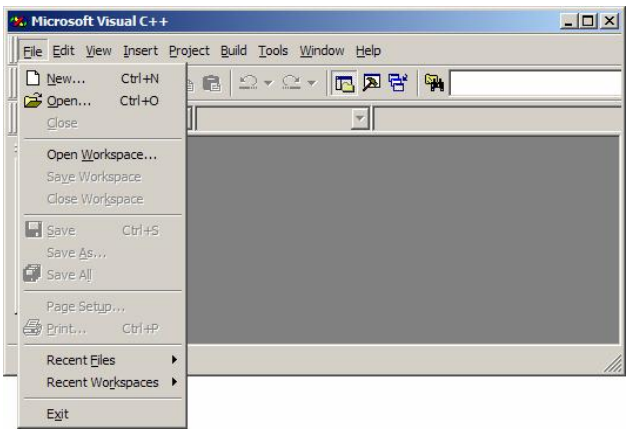

图 2-16 Visual C++ 6.0 主窗口下拉菜单

单击 New 按钮，则屏幕上出现一个 New（新建）对话框，如图 2-17 所示。

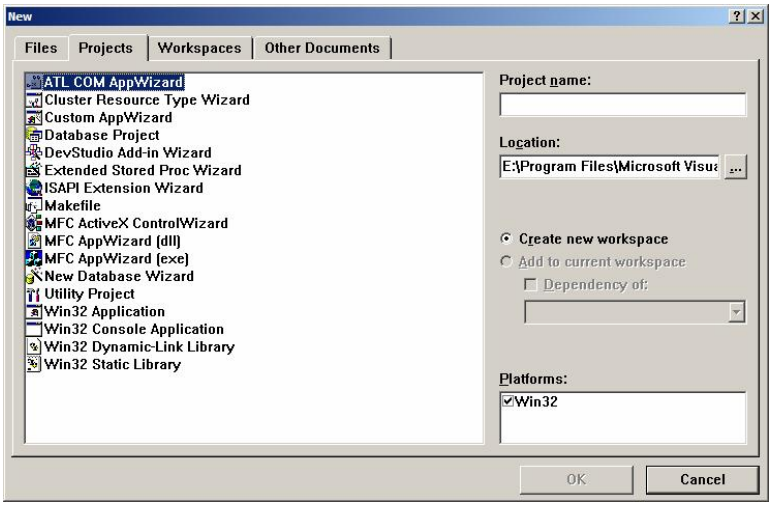

图 2-17 New 对话框

选择此对话框的左上角的 Files（文件）选项卡，其中有 C++ Source File 选项，该项功能是建立新的 C++源程序文件。然后在对话框右半部分的 Location（位置）文本框中输入准备编辑的源程序文件的存储路径（假设为 D:\CC），表示准备编辑的源程序文件将存放在 D:\CC 子目录下。在右上方的 File（文件）文本框中输入准备编辑的源程序文件的名字（输入 cl_1.c）。表示要建立的是 C 源程序，这样，即将进行输入和编辑的源程序就以 cl_1.c 为文件名存放在 D

盘的 CC 目录下。当然，读者完全可以指定其他路径名和文件名。

【注意】我们指定的文件名后缀为.c，如果输入的文件名为 cl_1.cpp，则表示要建立的是 C++源程序。如果不写后缀，系统会默认指定为 C++源程序文件，自动加上后缀.cpp。

在单击 OK 按钮后，回到 Visual C++主窗口，由于在前面已指定了路径（D:\CC）和文件名（c1_1.c），因此在窗口的标题栏中显示出 D:\CC\c1_1.c。可以看到光标在程序编辑窗口闪烁，表示程序编辑窗口已激活，可以输入和编辑源程序了，如图 2-18 所示。

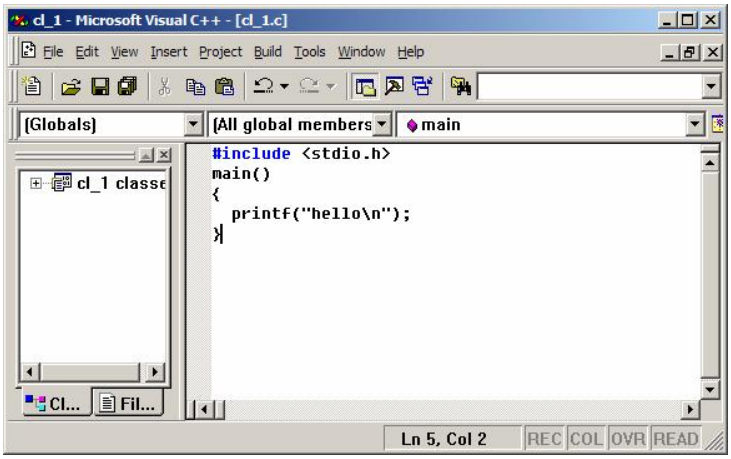

图 2-18　在主窗口输入源程序

在输入过程中难免不出现输入错误。如果用户能及时发现错误，可以利用全屏幕编辑方法进行修改编辑。在图 2-18 的最下部的中间，显示了 Ln 5，Col 2，表示光标当前的位置在第 5 行第 2 列，当光标位置改变时，显示的数字也随之改变。如果经检查无误，则将源程序保存在前面指定的文件中。此时，在主菜单栏中选择 File（文件），并在其下拉菜单中选择 Save（保存）项，如图 2-19 所示。

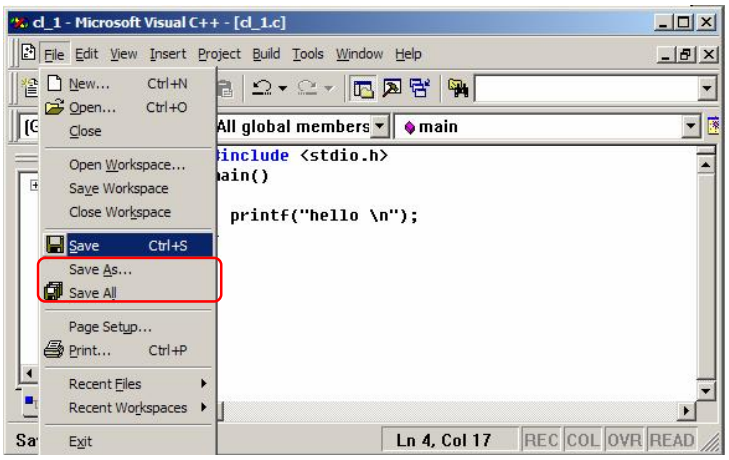

图 2-19　在下拉菜单中选择 Save 项

除此之外，也可以用快捷键 Ctrl+S 来保存文件。

如果不想将源程序存放到原先指定的文件中，可以不选择 Save 项，而选择 Save As（另存为）项，并在弹出的 Save As（另存为）对话框中指定文件路径和文件名。

2. 打开一个已有的程序

如果已经编辑并保存过 C 源程序，而希望打开所需要的源程序文件，并对它进行修改。具体操作步骤如下：

（1）在“Windows 资源管理器”或“我的电脑”中按路径找到已有的 C 程序文件名（如 cl_1.c）。

（2）双击此文件名，则自动进入了 Visual C++集成环境，并打开了该文件，程序显示在编辑窗口中。也可以选择 File→Open 菜单或按快捷键 Ctrl+O，或单击工具栏中的 Open 小图标来打开 Open 对话框，从中选择所需的文件。

（3）如果在修改后，仍保存在原来的文件中，可以选择 File（文件）→Save（保存），或用快捷键 Ctrl+S 或单击工具栏中的小图标来保存文件。

3. 通过已有的程序建立一个新程序的方法

如果已经编辑并保存过 C 源程序（而不是第一次在该计算机上使用 Visual C++），则可以通过一个已有的程序来建立一个新程序，这样做比重新输入一个新文件省事，因为可以利用原有程序中的部分内容。具体操作步骤如下：

（1）打开任何一个已有的源文件（例如 cl_1.c）。

（2）利用该文件修改成新的文件，然后通过 File（文件）→Save As（另存为）将它以另一文件名另存（如以 c1_2.c 名字另存），这样就生成了一个新文件 c1_2.c。

用这种方法很方便，但应注意在保存新文件时，不要错用 File→Save（保存）操作，否则原有文件（cl_1.c）的内容就被修改。

2.2.3 编译、连接和运行

1. 程序的编译

在编辑和保存了源文件（如 cl_1.c）以后，若需要对该源文件进行编译，选择主菜单栏中的 Build（编译），在其下拉菜单中选择 Compile cl_1.c（编译 cl_1.c）项，如图 2-20 所示。

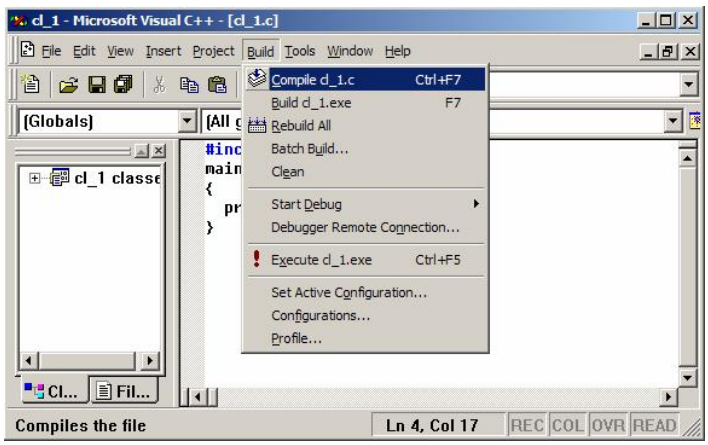

图 2-20　在下拉菜单中选择 Compile

由于建立（或保存）文件时已指定了源文件的名字 cl_1.c，因此在 Build 菜单的 Compile 项中就自动显示了当前要编译的源文件名 cl_1.c。

在选择 Compile cl_1.c 命令后，屏幕上出现一个对话框，提示此编译命令要求一个有效的

项目工作区，并询问是否同意建立一个默认的项目工作区，如图 2-21 所示。

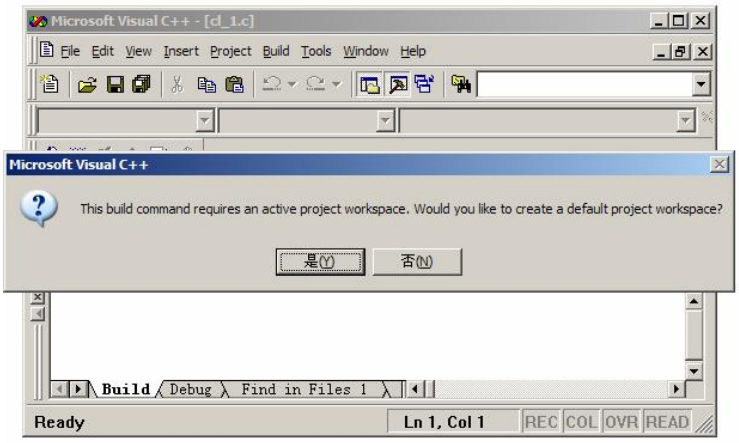

图 2-21　是否同意建立一个默认的项目工作区

单击“是（Y）”按钮，表示同意由系统建立默认的项目工作区，然后开始编译。也可以不用选择菜单的方法，而用快捷键 Ctrl+F7 来完成编译。

在进行编译时，编译系统检查源程序中有无语法错误，然后在主窗口下部的调试信息窗口输出编译的信息，如果有错，就会指出错误的位置和性质，如图 2-22 所示。

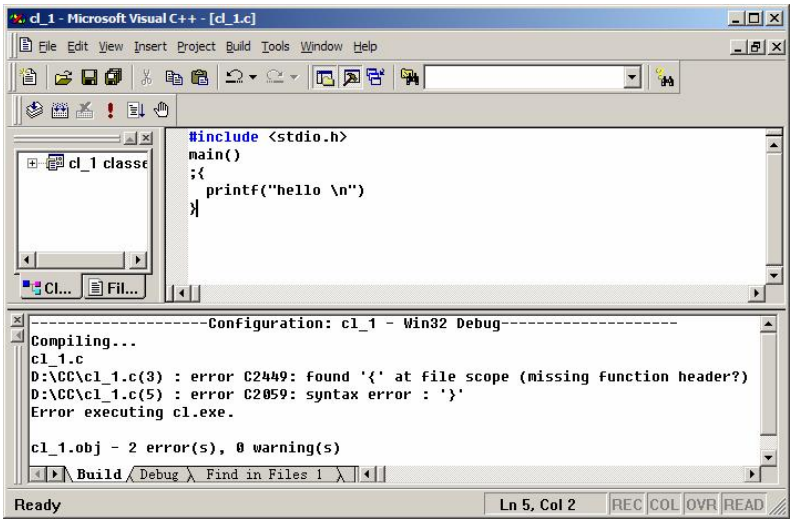

图 2-22　语法检查出错信息显示

2. 程序的调试

程序调试的任务是发现和改正程序中的错误，使程序能正常运行。编译系统能检查出程序中的语法错误。语法错误分为两类：一类是致命错误，以 error 表示，如果程序中有这类错误，就通不过编译，无法形成目标程序，更谈不上运行了；另一类是轻微错误，以 warning（警告）表示，这类错误不影响生成目标程序和可执行程序，但有可能影响运行的结果，因此也应当改正，使程序既无 error，又无 warning。

在图 2-22 所示的调试信息窗口中可以看到编译的信息，指出源程序有 2 个 error 和 0 个

warning。单击调试信息窗口中右侧向上箭头，可以看到出错的位置和性质，如图 2-23 所示。

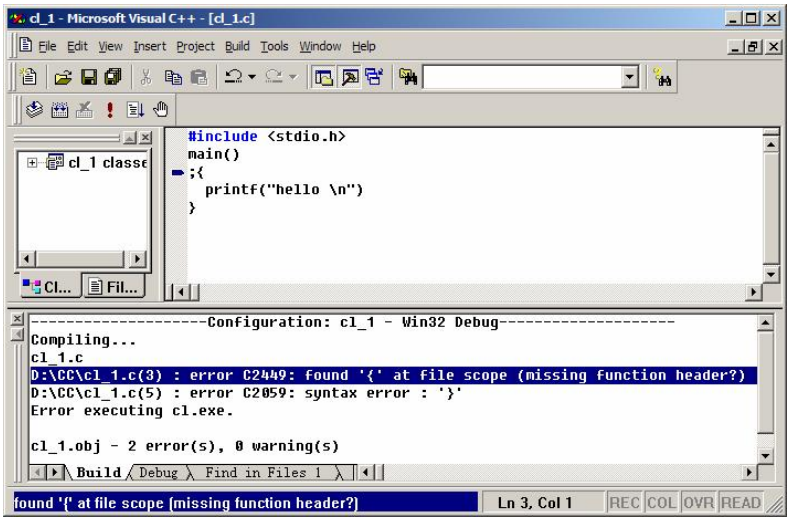

图 2-23　出错的位置和性质

从图 2-23 下部调试信息窗口所示的信息中可以看到：第 3 行有致命错误，错误的性质是：found '{' at file scope (missing function header?)，即在文件作用域发现了“{”，但没有函数首部。检查图 2-23 中的程序，发现第 3 行行首多加了一个分号，因此，编译系统认为它不是函数首部，“{”不属于 main 函数，所以出错。第 5 行错误的性质是：syntax error: '}'，即在此处出现语法错误。经检查程序，发现第 4 行末尾漏写了分号。也许有读者问：第 4 行有错，怎么报错是第 5 行呢？这是因为 C 允许将一个语句分写成几行，所以检查完第 4 行末尾无分号时还不能判定该语句有错，必须再检查下一行，直到发现第 5 行的“}”前没有分号（;），才判定出错。因此在第 5 行报错。所以在解析编译报错信息时，应检查出错点的上下行。

现在进行改错，双击调试信息窗口中的第 1 个报错行，这时在程序窗口中出现一个粗箭头指向被报错的程序行（第 3 行），提示改错位置，如图 2-24 所示。

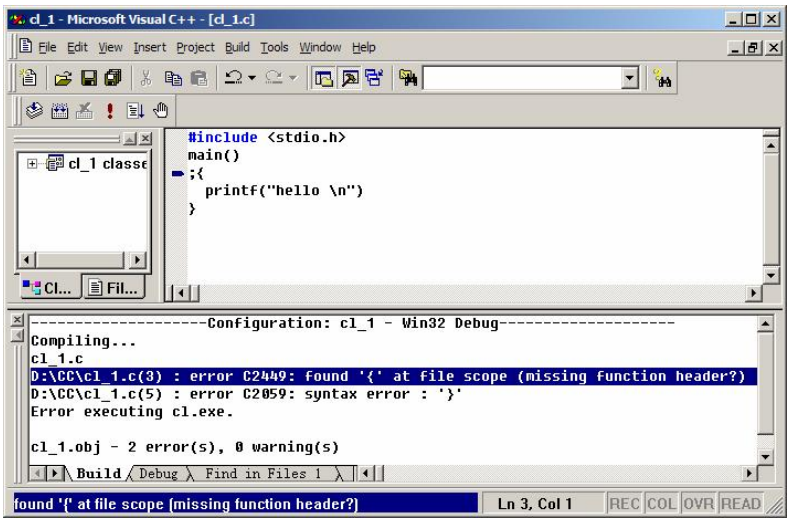

图 2-24　提示改错的位置

将第 3 行行首的分号删去。再用同样的方法找到第 2 个出错位置，在第 4 行末尾加上分号。再仔细阅读程序，认为应该没有问题了。

再选择 Compile cl_1.c 项重新编译，此时编译信息告诉我们：0 error（s），0 warning（s），既没有致命错误（error），也没有警告性错误（warning），编译成功，这时产生一个 c1_1.obj 文件，如图 2-25 所示。

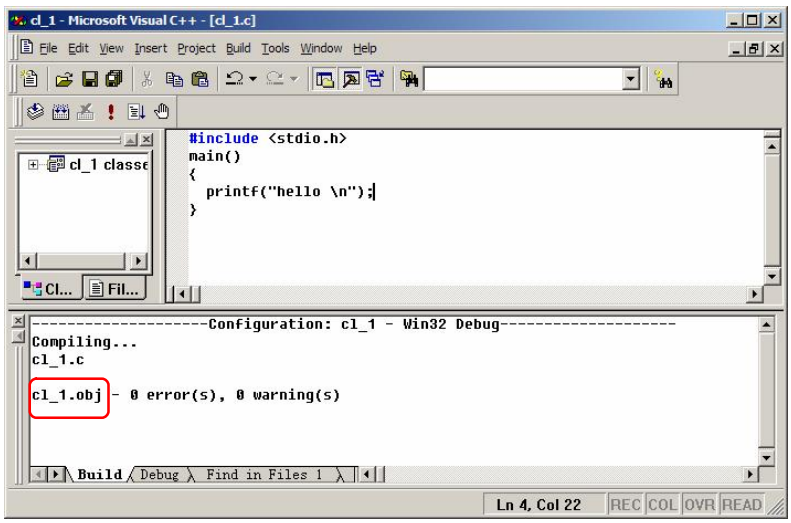

图 2-25　提示已形成产生.obj 文件

3．程序的连接

在得到目标程序后，就可以对程序进行连接了。由于刚才已生成了目标程序 cl_1.obj，编译系统据此确定在连接后应生成一个名为 cl_1.exe 的可执行文件，在菜单中显示了此文件名。此时应选择 Build（构建）→Build cl_1.exe（构建 cl_1.exe），如图 2-26 所示。

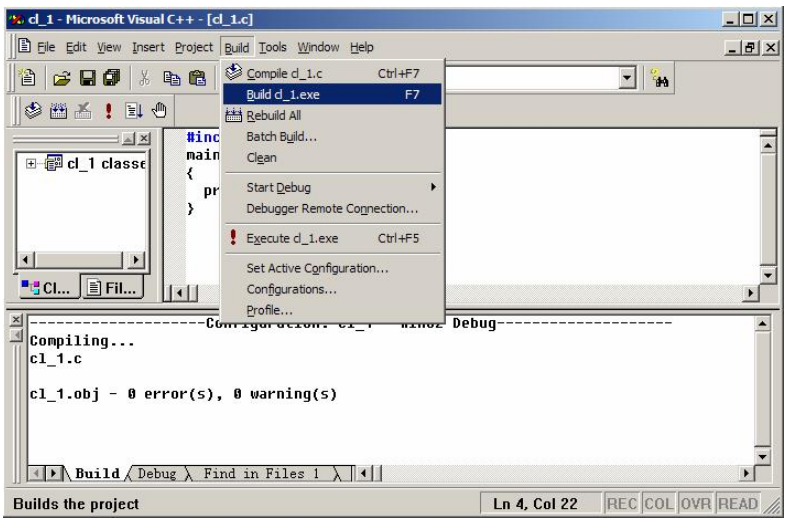

图 2-26　对目标文件进行连接

在完成连接后，在调试信息窗口中显示连接时的信息，说明没有发现错误，生成了一个可执行文件 cl_1.exe，如图 2-27 所示。

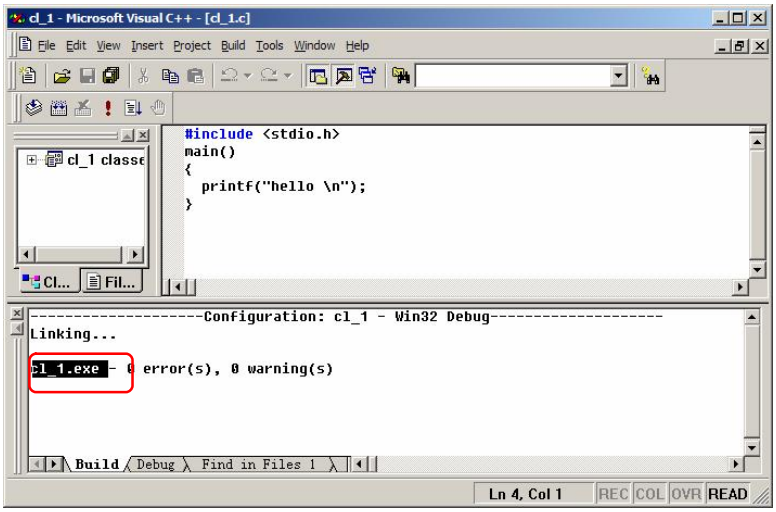

图 2-27　连接生成.exe 文件

以上介绍的是分别进行程序的编译与连接，也可以选择菜单 Build→Build（或按 F7 键）一次完成编译与连接。对于初学者来说，还是提倡分步进行程序的编译与连接，因为程序出错的机会较多，最好等到上一步完全正确后才进行下一步。对于有经验的程序员来说，在对程序比较有把握时，可以一步完成编译与连接。

4. 程序的执行

在得到可执行文件 cl_1.exe 后，就可以直接执行 cl_1.exe 了。选择 Build→Execute cl_1.exe（执行 cl_1.exe），如图 2-28 所示。

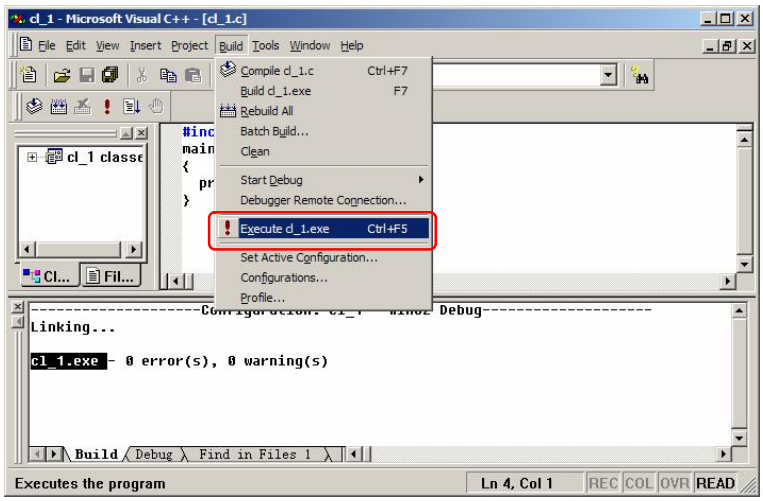

图 2-28　运行.exe 可执行文件

在单击“Execute cl_1.exe”项后，即开始执行 cl_1.exe。也可以不通过选择菜单，而用快捷键 Ctrl+F5 来实现程序的执行。程序执行后，屏幕切换到输出结果的窗口，显示出运行结果，如图 2-29 所示。

可以看到，在输出结果窗口中的第 1 行是程序的输出：

hello

然后换行。第 2 行 Press any key to continue 并非程序所指定的输出，而是 Visual C++在输出完运行结果后由 Visual C++ 6.0 系统自动加上的一行信息，通知用户：“按任何一键以便继续”。当按下任何一键后，输出窗口消失，回到 Visual C++的主窗口，此时可以继续对源程序进行修改补充或进行其他的工作。

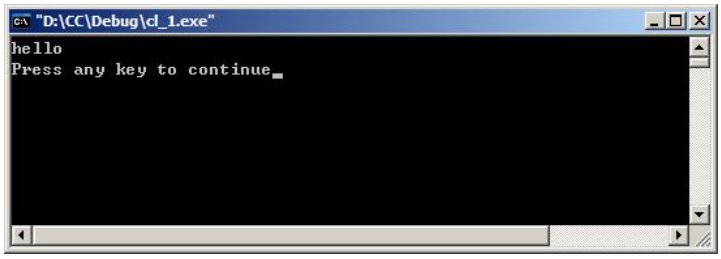

图 2-29　显示运行结果

如果已完成对一个程序的操作，不再对它进行其他处理，应当选择 File（文件）→Close Workspace（关闭工作区），以结束对该程序的操作。

2.2.4　建立文件工作区域

上面介绍的是最简单的情况，一个程序只包含一个源程序文件。如果一个程序包含多个源程序文件，则需要建立一个项目文件（project file），在这个项目文件中包含多个文件（包括源文件和头文件）。项目文件是放在项目工作区中的，因此还要建立项目工作区。在编译时，系统会分别对项目文件中的每个文件进行编译，然后将所得到的目标文件连接成为一个整体，再与系统有关资源连接，生成一个可执行文件，最后执行这个文件。

在实际操作时有两种方法：一种是由用户建立项目工作区和项目文件；另一种是用户只建立项目文件而不建立项目工作区，由系统自动建立项目工作区。

1. 由用户建立项目工作区和项目文件

（1）编辑源程序文件：先用前面介绍过的方法分别编辑好同一程序中的各个源程序文件，并存放在自己指定的目录下。例如，某一个程序包含 file1.c、file2.c 和 file3.c 共 3 个源文件，并已把它们保存在 D:\CC 子目录下。

```
/* file1.c 代码*/
int Add(int a,int b)
{ return a+b; }
int Subtraction(int a,int b)
{ return a-b; }
/* file2.c 代码 */
int Multiplication(int a,int b)
{ return a*b; }
double Division(int a,int b)
{ return (double) a/b; }
/* file3.c 代码 */
#include <stdio.h>
int Add(int,int );
int Subtraction(int,int );
int Multiplication(int,int );
```

```
int main()
{   int a=3,b=5;
    int add,sub,multi;
    add=Add(a,b);
    sub=Subtraction(a,b);
    multi=Multiplication(a,b);
    printf("add=%d\n",add);
    printf("sub=%d\n",sub);
    printf("multi=%d\n",multi);
    return 0;
}
```

（2）建立一个项目工作区：在如图 2-25 所示的 Visual C++主窗口中选择 File（文件）→New（新建），在弹出的 New（新建）对话框中选择上部的 Workspaces（工作区）选项卡，表示要建立一个新的项目工作区。在对话框右部的 Workspace name（工作区名字）文本框中输入指定的工作区的名字（如 ws1）。在 Location（位置）文本框中输入指定的文件目录（如 D:\CC，也可以指定为其他目录），如图 2-30 所示。

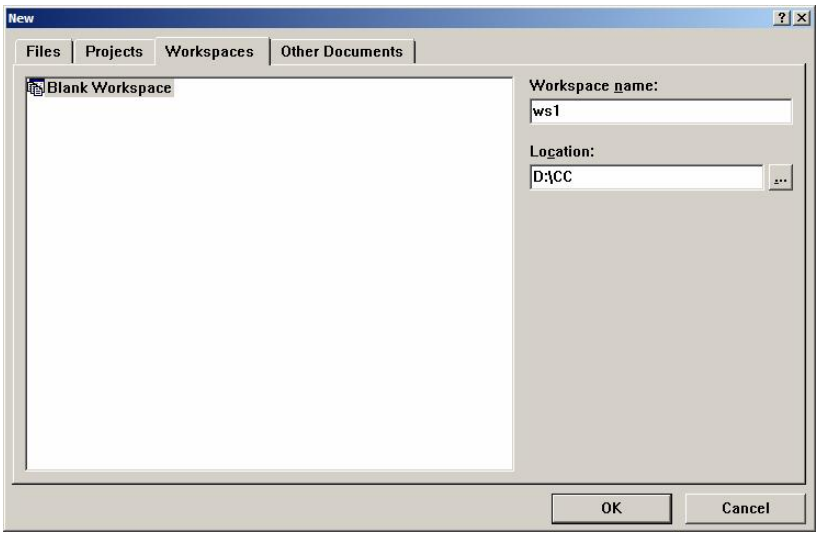

图 2-30　建立工作区

然后单击右下部的 OK 按钮。此时返回 Visual C++主窗口。

（3）建立项目文件：选择 File（文件）→New（新建），在弹出的 New（新建）对话框中选择上部的 Projects（项目，中文 Visual C++把它译为“工程”）选项卡，表示要建立一个项目文件，如图 2-31 所示。

在对话框中左部的列表中选择 Win 32 Console Application 项，并在右部的 Location（位置）文本框中输入项目文件的位置（即文件路径，现在输入 D:\CC），在 Project name（中文界面中显示为“工程”）文本框中输入指定的项目文件名，现在输入 project1。选中窗口右部的 Add to current workspace（添加至现有工作区）单选钮，表示新建的项目文件是放到刚才建立的当前工作区（ws1）中的。此时，Location 栏中内容自动变为 D:\CC\wsl\project1，表示已确认项目文件 project1 存放在工作区 ws1 中，然后单击 OK（确定）按钮，此时弹出一个如图 2-32 所示的对话框。

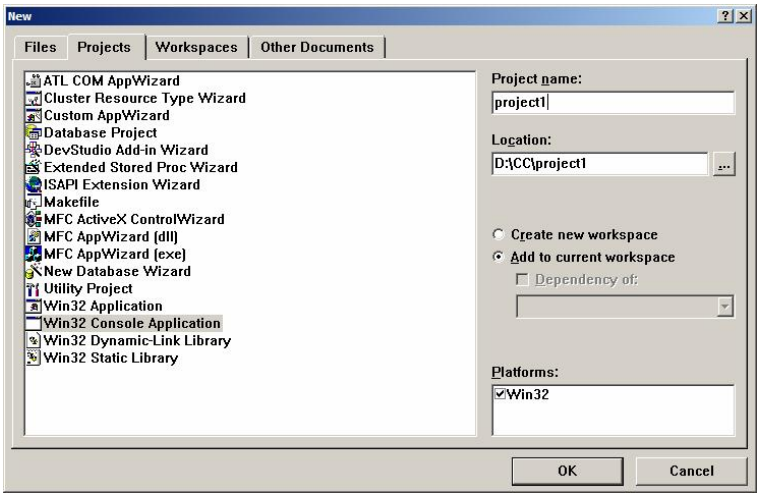

图 2-31　建立项目文件

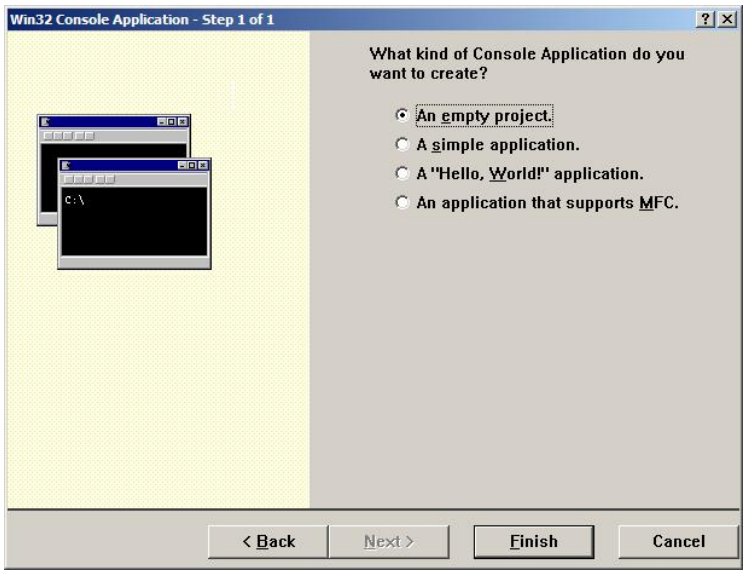

图 2-32　确认项目文件 projectl 存放在工作区

选中 An empty project 单选钮，表示新建立的是一个空的项目，单击 Finish 按钮，系统弹出一个 New Project Information 对话框，显示刚才建立的项目的有关信息，如图 2-33 所示。

在其下方可以看到项目文件的位置（文件路径为 D:\CC\ws1\project1），确认后单击 OK 按钮。此时又回到 Visual C++主窗口，左部窗口中有一个 Workspace 窗口，单击 FileView 选项卡，窗口内显示：Workspace 'ws1': 1 project(s)，表示工作区 ws1 中有一个项目文件，其下一行为 project1 files，表示项目文件 project1 中的文件，现在为空，如图 2-34 所示。

（4）将源程序文件放到项目文件中：在 Visual C++主窗口菜单栏中选择 Project→Add To Project（添加到项目中，在中文界面上显示为“添加工程”）→Files，如图 2-35 所示。

在选择 Files 命令后，屏幕上出现 Insert Files into Project 对话框。在列表框中按路径找到源文件 file1.c、file2.c 和 file3.c 所在的子目录，并选中 file1.c、file2.c 和 file3.c，如图 2-36 所示。

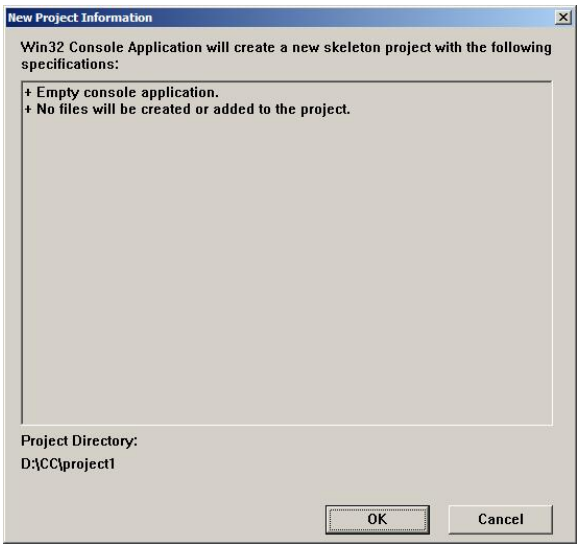

图 2-33　显示已建项目的信息

图 2-34　显示当前项目文件状态

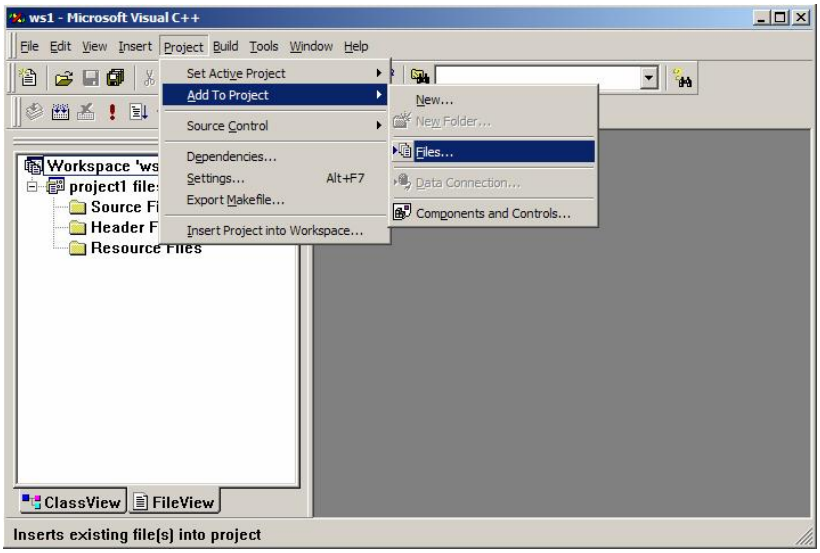

图 2-35　添加项目工程

单击 OK（确定）按钮，就把这 3 个文件添加到项目文件 project1 中了。此时，回到 Visual C++主窗口，再观察 Workspace 窗口，选择其下部的 FileView 选项卡，窗口内显示了项目文件 project1 中包含文件的情况，如图 2-37 所示。

可以看到：project1 中包含了源程序 file1.c、file2.c 和 file3.c。

（5）编译和连接项目文件：由于已经把 file1.c、file2.c 和 file3.c 添加到项目文件 project1 中，因此只须对项目文件 project1 进行统一的编译和连接，方法是：在 Visual C++主窗口菜单中选择 Build（编译）→Build project1.exe（构建 project1.exe），如图 2-38 所示。

在选择 Build project1.exe 后，系统对整个项目文件进行编译和连接，在窗口的下部会显示编译和连接的信息。如果程序有错，会显示出错信息；如果无错，会生成可执行文件 project1.exe。

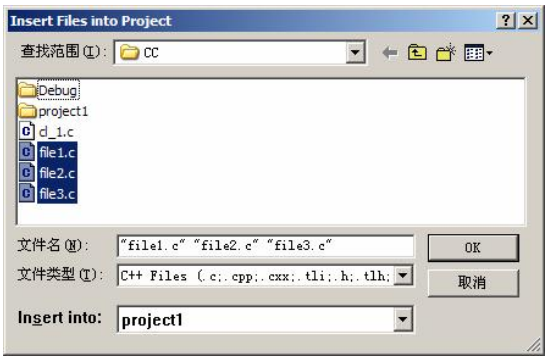

图 2-36　显示已建的源文件

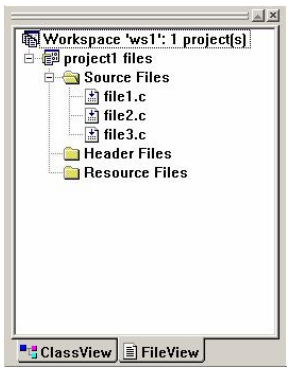

图 2-37　显示项目中包含的源文件

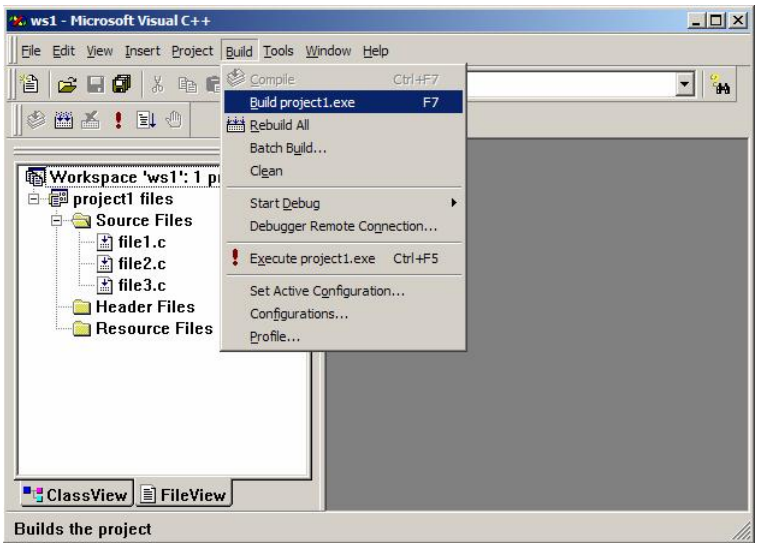

图 2-38　对项目中的源文件进行编译

（6）执行可执行文件：选择 Build（编译）→Execute project1.exe（执行 project1.exe），就执行 project1.exe，如图 2-39 所示。

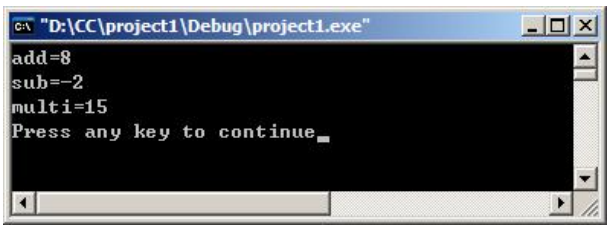

图 2-39　运行结果

2. 用户只建立项目文件

上面介绍的方法是先建立项目工作区，再建立项目文件，步骤比较多。可以采取简化的方法，即用户只建立项目文件，而不建立项目工作区；由系统自动建立项目工作区。在本方法中，保留“由用户建立项目工作区和项目文件”介绍的第（1）、（4）、（5）、（6）步，取消第（2）步，修改第（3）步。具体步骤如下：

（1）分别编辑好同一程序中的各个源程序文件。同前面的第（1）步。

（2）建立一个项目文件（不必先建立项目工作区）。

在 Visual C++主窗口中选择 File（文件）→New（新建），在弹出的 New（新建）对话框中选择上部的 Projects（工程）选项卡，表示要建立一个项目文件，如图 2-40 所示。

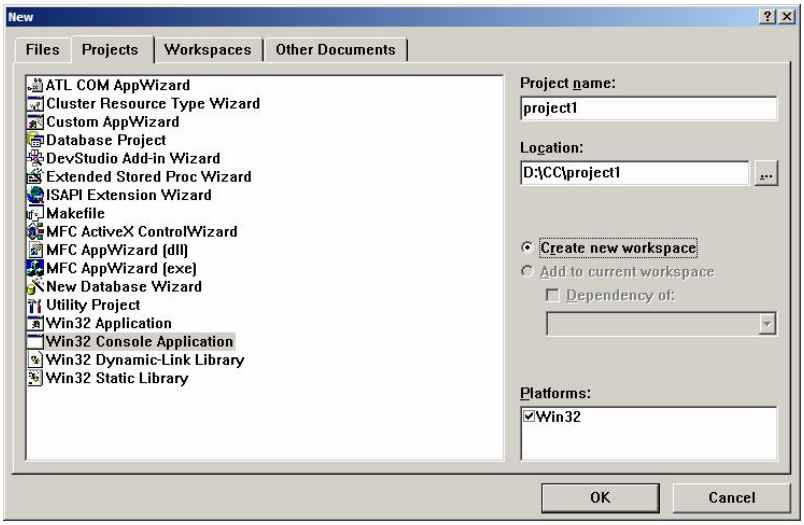

图 2-40　新建一个项目文件

在对话框中左部列表中选择 Win32 Console Application 项，在 Project name（工程）文本框中输入指定的项目文件名（project1）。可以看到在右部的中间处默认选定了 Create new workspace（创建新工作区）单选钮，因为用户未指定工作区，系统自动开辟新工作区。

单击 OK 按钮，出现如图 2-41 所示的 Win32 Console Application-Step 1 of 1 对话框。

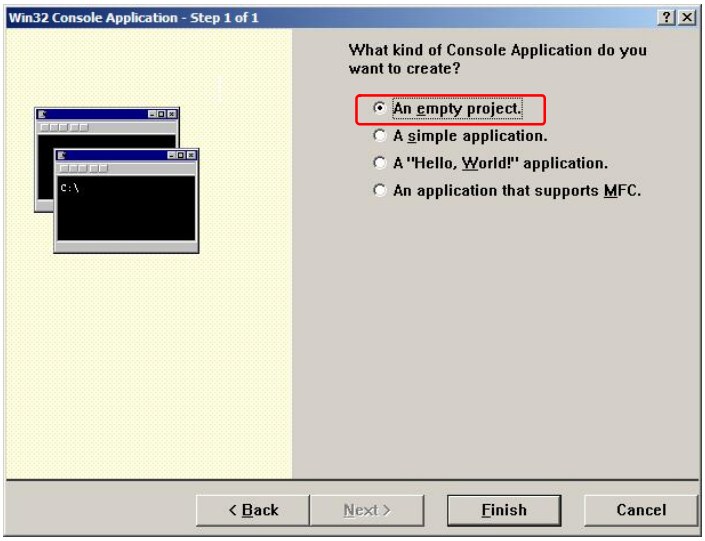

图 2-41　Win32 Console Application-Step 1 of 1 对话框

选择右部的 An empty project 单选钮，单击 Finish（完成）按钮后出现 New Project Information（新建工程信息）对话框，如图 2-42 所示。

从它的下部可以看到项目文件的路径（中文 Visual C++中显示为“工程目录”）为D:\CC\project1。单击 OK（确定）按钮，在 Visual C++主窗口中的 Workspace 窗口的下方单击 FileView 选项卡，窗口中显示 Workspace 'projectl': 1 project(s)，如图 2-43 所示。

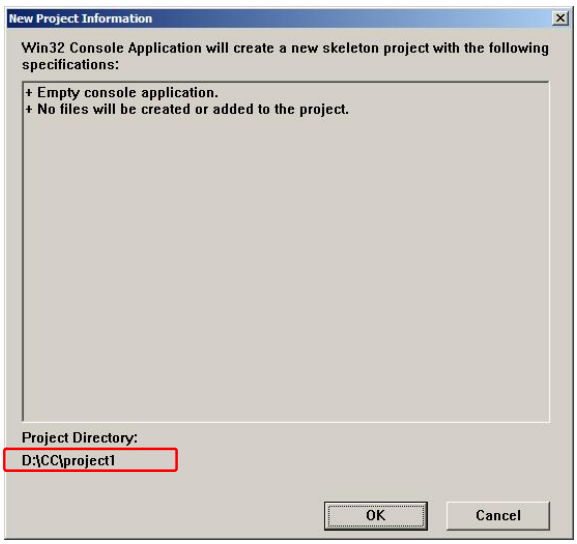

图 2-42　显示新建工程信息

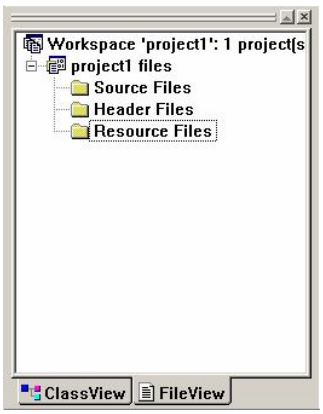

图 2-43　显示已建立的工作区

说明系统已自动建立了一个工作区，由于用户未指定工作区名，系统就将项目文件名 project1 同时作为工作区名。

（3）向此项目文件添加内容。步骤与“由用户建立项目工作区和项目文件”方法中的第（4）步相同。

（4）编译和连接项目文件。步骤与“由用户建立项目工作区和项目文件”方法中的第（5）步相同。

（5）执行可执行文件。步骤与“由用户建立项目工作区和项目文件”方法中的第（6）步相同。

显然，这种方法比前面的方法简单一些。

在介绍单文件程序时，为了尽量简化，没有建立工作区，也没有建立项目文件，而是直接建立源文件。实际上，在编译每一个程序时都需要一个工作区，如果用户未指定，系统会自动建立工作区，并赋予它一个默认名（此时以文件名作为工作区名）。

§2.3　程序错误类型与查找方法

用户在编程过程中，任何编程者都不可能完全避免出现错误。一个简单的 C 语言程序，在编辑、编译、连接、运行、调试等过程中出现的错误，主要是语法错误和工作环境参数设置错误，在编译时编译系统会自动报错，并会给出相关错误信息，用户根据错误信息的提示，可以很容易地排查并纠正错误。但对一个较为复杂的程序，如果把语法错误和工作环境参数设置错误都已排查完了，还是得不到正确的结果，这说明程序中还存在其他错误，使得结果和所预期的不一致，这种错误排查相对困难，本节就常见错误类型和查找错误方法进行讨论。

2.3.1 程序错误类型

由于 C/C++编译程序的自由度较大，对语法的检查不像其他高级语言那么“苛刻”，往往编译通过了，但程序运行结果却不是正确的。Turbo C++ 3.0 和 Visual C++ 6.0 编译程序将查出的源程序错误分为：严重错误（fatat error）、一般错误（error）和警告（warning）（参见附录 A）。如果从编程角度讲，我们可把常见程序错误分为 3 类：语法错误、逻辑错误、运行错误。

1. 常见语法错误

语法错误是指不符合 C/C++语法规则，因而在程序编译时出现“出错信息”。导致编译出现“出错信息”的原因主要有以下几个方面。

（1）大小写错误：C 语言严格区分大小写字母，因此关键字和库函数名都使用小写字母，符号常量、类型定义等一般使用大写字母，变量名和自定义函数名一般建议使用小写字母。编写 C 程序一定要建立起大小写字母表示不同标识符的概念。

（2）关键字或库函数名拼写错误：例如，函数名 printf 写成 print。

（3）变量定义与引用错误：初学者因不习惯定义变量，常常写到哪用到哪。在程序中使用没定义的变量时，编译中会出现“error C2065：'y'：undeclared identifier”之类的错误信息，表示变量 y 未定义。通常，除了直接常量以外，程序中出现的任何标识符（变量、符号常量、类型名、函数名、宏等）都应该先定义后使用，不能对变量重复定义或在可执行语句后定义变量。

（4）括号（花括号、小括号）不配对的错误：花括号常用于构造函数体或复合语句，当要将多个语句构造成一个复合语句一起进行处理时，如果忘记了使用花括号，则只有第一个语句能按原意执行。复合语句常见于选择、循环结构的语句中，遇到类似的结构，一定要仔细检查是否遗漏了花括号。花括号不配对可能牵连出很多复杂错误，通常会与后续的语句相关联。

小括号常用于在构造表达式时来改变运算的优先级，如果左、右括号不配对，不仅会造成计算结果不正确，而且还可能牵连出若干错误。

（5）使用错误的数组下标值：数组下标的有效范围是从 0 到数组元素个数减 1 为止。在使用数组时，应注意正确使用数组的上下限下标值。下标越界时编译程序并不报错，但在运行中会引起意外错误。另外，下标表达式的值应该是有序值（整型、字符型、逻辑值、枚举值），如果是浮点值则会自动截取整数部分当做下标值。

（6）遗漏分号、引号等：分号是语句的一个部分，它的出现标志语句的结束。由于 C 语言的书写格式非常灵活，允许在一行中书写多个语句，也可以将一个语句写在多个行中。若语句末尾遗漏了分号，C 编译器可能产生错误的提示信息，因为编译进行扫描时，必须进行语法检查，确定语句是否使用的是结构语句，编译器可能将下一行看成是该行的一部分，根据这两行的内容就产生了出错提示信息。

【注意】如果编译时指出某一行错误但并查不出错误，则应该检查它的前一行，看是否遗漏了分号。复合语句中的最后一个语句仍然需要分号，不能省略，而“}”之后不必使用分号。在结构和共用体的定义的花括号之后必须加上分号。

遗漏分号可能引起许多出错信息，一旦改正了这一处错误，其他牵连错误就立即消失。因此，当编译出现多个错误提示时，应优先处理前面的错误。因为后面的错误很可能是由于前面的错误引起的，修改前面的一个错误经常可以消除后面的大量错误。

2. 常见逻辑错误

所谓逻辑错误，就是程序并无违背语法规则，也能正常运行，但程序执行的结果与原意不符。逻辑错误通常是由于程序设计人员设计算法或编写的程序有错。导致出现逻辑错误的原因主要有以下几个方面。

（1）遗漏取地址运算符：C/C++没有输入输出语句，而是通过函数来实现的，这与其他语言有很大区别，因此使得有编程经验和没编程经验的人都很容易出现编程错误。例如：

```
#include<stdio.h>
int main()
{   int a=3,b=4,aver;
    scanf("%d%d",a,b);
    aver=(a+b)/2.0;
    printf("%d\n",aver);
}
```

编写者的原意是先对 a 和 b 赋初值 3 和 4，然后通过 scanf 函数向 a 和 b 输入新的值。有经验的人一眼就会看出 scanf 函数写法不对，漏了地址符&，应该是：

```
scanf("%d%d",&a,&b);
```

其含义是把用户从键盘输入的一个整数送到变量 a 的地址所指向的内存单元。如果变量 a 的地址是 1020，则把用户从键盘输入的一个整数送到地址为 1020 的内存单元中，也就是把输入的数赋给了变量 a。但写成：scanf("%d%d",a,b);后虽然存在错误，但这个错误在程序编译时是检查不出来的，也不会显示“出错信息”。程序不仅能通过编译，而且也能运行。

这时，编译系统把用户从键盘输入的一个整数送到变量 a 的值所指向的内存单元。如果 a 的值为 3，则把用户从键盘输入的数送到地址为 3 的内存单元中。显然，这不是变量 a 所在的单元，而是一个不可预料的单元。这样就改变了该单元的内容，有可能造成严重的后果，是很危险的。这种错误比语法错误更难检查，要求程序员有较丰富的经验。

（2）运算符错误：这种错误通常出现在 if、while、do…while 等控制语句中，如在需要做“等于”测试时，错把等号“==”写成了赋值号“=”，由于编译器无法检查出这种错误，程序运行时就会得到错误的结果，从而产生逻辑错误。例如，某程序中有以下语句：

```
while(stop=10)
{   printf("stop the program");
    …
}
```

由于条件表达式是一个赋值语句，表达式的结果永远是 10（非 0，为逻辑真），循环不能自动结束。

（3）运算优先级问题：在编写程序时往往很容易疏忽运算符的运算优先级以及各种运算符的结合性。例如 while(c=getchar()!='*')……，本来编程的目的是接收键盘输入的一个字符赋给 c，再判断该字符是否是“*”键，而上述条件表达式相当于 c=(getchar()!='*')，c 得到的不是一个字符，而是逻辑表达式的值 0 或 1。因此，在难以肯定运算的先后次序时，用加上小括号来确定运算次序是很安全的做法，如上述条件表达式的正确写法为((c=getchar())!=('*')。

初学者在编程时往往是因为对运算符的运算次序或者说对此类表达式的理解不清楚而造成编程错误，特别是自增运算符“++”和自减运算符“--”，需要通过上机测试相应的表达式的值来加深理解。

（4）遗漏了复合语句符号：编程时因为疏忽遗漏了构成复合语句的符号，从而导致了程

序错误。例如：

```
sum=0;
i=1;
while(i<=100)
sum=sum+i;                          //少了构成复合语句的花括号"{"
i++;                                //少了构成复合语句的花括号"}"
```

语法并无错误。但由于缺少花括号，while 语句的范围只包括“sum=sum+i”，而不包括“i++;”。通知给系统的信息是当 i≤100 时，执行“sum=sum+i;”，而 i 的值始终不变，形成一个永不终止的“死循环”。C 系统无法辨别程序中这个语句是否符合程序编写者的原意，而只能忠实地执行这一指令。又例如求：

s=1+2+3+…+100

如果写出以下语句：

```
for(s=0,i=1;i<100;i++)
s=s+i;
```

虽然语法没有错，但按表达式“i＜100”求出的结果却是 1+2+3+…+99 之和，而不是 1+2+3+…+100 之和，原因是少执行了一次循环。这种错误在程序编译时是无法检查出来的，因为语法是正确的，计算机无法知道程序编制者是想累加 100 个数，还是想累加 99 个数，只能按语句表达式执行。

（5）使用分号位置：对于选择和循环等结构语句，常见的错误是错放了分号位置，使得选择语句或循环体语句为空语句。例如：

```
while(表达式){循环体};
if(表达式){语句};
```

当在(表达式)后加上分号时，使其等价于：

```
while(表达式);
{循环体};
if(表达式);
{语句};
```

将一个完整的结构拆分成了两条语句，使循环体变成一个空语句，因为编译程序检查不出这类符合语法规则的错误，造成了更难检查出来的逻辑错误或运行错误。

（6）书写结构语句时的错误：对于 if 语句、switch 语句、for 语句、while 语句、do...while 语句，要特别注意其语句的整体结构的完整性，要充分理解各结构语句的执行流程，正确书写语句形式，并采用缩进方式书写，便于阅读和检查。结构中的复合语句一定要用花括号括起来。例如下面的复合语句：

```
if(表达式)
 {复合句 1}
else
 {复合句 2}
```

如果不用花括号括起来，则因逻辑错误而出现意想不到的运算结果。

（7）循环或选择语句的条件表达式错误：如果在循环或选择语句的条件表达式中有逻辑错误，无法通过编译、连接来排除，此时只能仔细阅读程序，通过给定条件和边界值进行检测后才能排除。此时，如果出现逻辑错误，需要认真检查程序和分析运行结果。如果是算法有错，则应先修改算法，再改程序。如果算法正确而程序语句不对，则直接修改程序。

3．程序运行错误

程序运行错误是指程序既无语法错误，又无逻辑错误，但程序不能正常运行或结果不对。对初学者来说，最常见的这类错误包括：忘记对计数器、总数等变量进行初始化；用于控制输入循环结束的标记值不是一个合法的输入数据值；一个作为除数的变量的值为 0、数据本身不适合以及数据类型不匹配等。

（1）数据类型不匹配：这是最常见的逻辑错误，例如：

```
#include<stdio.h>
void main()
{   int a, b, c;
    scanf("%d, %d", &a,&b);
    c=a/b;
    printf("%d\n", c);
}
```

运行该程序，当输入的 b 值为非零时无问题。当输入的 b 值为零时，则出现“溢出（overflow）”错误。又如，如果输入数据为：

456.78,34.56↙

则输出结果 c 的值为 2，显然是不对的。其原因是由于输入的数据类型与输入格式符%d 不匹配而引起的。

（2）printf 和 scanf 函数调用问题：主要是格式控制与地址表、格式控制与输出表的对应匹配问题，以及表达式的书写形式是否符合函数的要求。

（3）数据输入错误或输入数据的格式不符合要求：在程序中使用 scanf 之类的函数实现数据输入时，要特别注意数据的输入格式是否与该语句的输入要求相一致，否则，会导致错误的运行结果，甚至终止程序的运行。例如：

```
main()
{   int no,age;
    char name[15],sex;
    printf("\nPlease input no,name,sex,age：\n");
    scanf("%d%s%*c%c%d",&no,name,&sex,&age);      //%*c 用来跳过该格式说明对应的输入数据
    printf("No=%d，Name=%s，Sex=%c，Age=%d\n",no,name,sex,age);
}
```

该程序能通过编译和连接。运行该程序时若按以下格式输入数据：

258 LiuYing F 23↙

则得出以下运行结果：

No=258，Name=LiuYing，Sex=F，Age=23

如果在 F 前面多输入了一个空格，则得到的结果便变成：

No=258，Name=LiuYing，Sex=，Age=1160

采用键盘输入数据时很容易发生这类错误，所以在数据输入量比较大时常采用文件方式存放输入数据。程序通过对文件的读/写来输入/输出数据，不仅便于修改其中的错误，而且在重复计算或调试程序时不用反复输入数据。

（4）数值超出范围或用值为零的变量做除数。在使用整型变量和字符变量时，常常忽略了变量保存值的范围，例如在求阶乘运算的程序中，如果使用 int 整型变量来保存阶乘值就很容易造成数值超出范围（溢出），使得结果错误。例如下面的程序：

```
int fac(int n)
{   int k,s=1;
    for(k=1;k<=n;k++)
    s*=k;
    return s;
}
main()
{   printf("%d",fac(10));
}
```

该程序的功能是利用函数 fac 求 10 的阶乘，程序编译时没有语法错误，但运行该程序时结果却是错误的。因为 10 的阶乘值超出了 int 整型变量 s 所能存储的范围。

又例如下面的程序：

```
main()
{   int k=5,s=0;
    printf("%d",k/s);
}
```

如果从数学角度讲，显而易见运算表达式 k/s=5/0 有问题，但在 C/C++中，却能够通过编译，当然在程序运行时会给出出错提示。

以上介绍了常见编程错误的 3 种类型：程序语法错误、程序逻辑错误、程序运行错误。要使编程或运行程序的错误减少到最低程度，对于初学者来说，特别要注意以下几点。

（1）熟悉 C/C++语法规则：必须严格按照语法规则进行编程，例如变量定义规则、语句表达式规则等。并且要养成良好的编程风格和习惯，以便于查找编程错误。

（2）合理选用数据类型：在程序设计过程中通常注意的只是数据的基本类型定义，却常常没注意到该类型所存储信息值的范围（数据大小），因而造成数据逻辑错误。

（3）按程序要求输入数据：运行程序时一定要注意输入的数据类型与输入格式符相匹配。同时，也要求在设计程序时要考虑程序的容错性和程序的健壮性。

为了帮助读者分析程序和调试程序，以上对常见编程错误进行了简要分类和分析，使读者有个大致了解，然后在后面各章中对“常见编程错误”进行详细介绍。

2.3.2 查找错误的基本方法

上面分析了常见的错误类型，如果源程序中存在语法错误，则不能通过编译，并且系统会给出提示信息。但源程序通过编译、连接生成.exe 可执行文件，并不能说明运行结果正确，甚至程序中存在有严重错误。由于常见语法的错误可由编译系统检查，因此，查找程序错误的重点是逻辑错误和运行错误。下面简要介绍查找程序错误的基本方法。

1. 静态检查

在写好或输入一个源程序后，不要急于进行编译，而应对程序进行静态检查。静态检查就是人工阅读程序，一行一行地读，模拟计算机运行来判断各个语句执行后变量值的改变和程序的流向。这一步是十分重要的，它能发现程序设计人员由于编程疏忽或算法设计欠妥而造成的错误，而这一步骤往往容易被人忽视。如果在写好一个程序后，没有通过初步检查就进行编译，把检查程序、发现错误的工作寄托给编译系统，不仅会耗费大量的调试时间，更为重要的是很多问题是编译系统无法检查到的。而且，作为一个程序人员应当养成严谨的科学作风，每一步都要严格把关，不把问题留给后面的工序。静态检查的重点有两个方面。

（1）检查算法结构：按照算法结构检查程序的整体框架，因为程序是由循环、选择、函数调用等结构语句构造起来的。其次，通过观察程序代码，理解程序的作用和整体结构，以及各个函数的功能、调用形式和返回值，分析重要变量和关键语句在执行过程中的变化情况。

对于逻辑错误，往往需要仔细检查和解析才能发现，通常可按如下方法进行检查。

① 将程序与流程图（或伪代码）仔细对照，只要流程图正确，就很容易发现程序的错误所在。此时，不仅能发现逻辑错误，也能发现语法错误。例如，复合语句忘记写花括号，只要一对照流程图就能很快发现语法错误。

② 如果在程序中没有发现问题，就要检查流程图有无错误，即算法有无问题。因此，在进行程序设计时，对于较为复杂的算法，应详细描述算法的流程图，这样既有利于编写正确的程序代码，也有利于程序检查。

（2）检查结构控制条件和边界值：循环结构和选择结构的程序流程完全由条件表达式的真与假来控制，那么，条件表达式的值便成了程序的转折点。因此，必须从各种角度、各种可能的取值加以判断。边界值和特殊情况的处理主要针对输入数据的合法取值范围的边界值、变量可保存值的边界、数组及字符串的边界等是否作过相应的处理，处理方式是否完善。

在静态检查过程中，对不合理的设计应予以修改，力求做到：

① 应当采用结构化程序方法编程，以增加可读性。

② 对程序中的关键算法和重要变量加以注释，不仅可以帮助理解每段程序的作用，而且对调试程序也能起到很大的帮助作用。

③ 在编写复杂的程序时，不要将全部语句都写在 main 函数中，而要多利用函数，一个函数实现一个单独的功能。各函数之间除用参数传递数据这一渠道以外，数据间尽量少出现耦合关系，这样既易于阅读也便于调试。

2. 动态检查

在静态检查无误后，便可以进行动态检查。源程序中的错误是多种多样的，有时有的错误很隐蔽，在纸面上难以查出，此时可以利用系统的编辑、编译和运行来查找问题的所在。

（1）观察程序的运行情况：通过观察程序的运行过程和得到的结果，可以帮助调试者判断大致的出错方向，用以指导后续的调试工作。

（2）增加输出语句：在程序不同位置插入 printf 语句，判断程序执行到该处时某些关键的变量或表达式的值是否正确。通过这种分段压缩的检查方法，发现错误所在区域。

（3）设置断点：设置断点可以使程序执行到某处后停下来，以便查看变量及表达式的值，连续监视一个变量或表达式可以清楚地知道其变化的规律。

（4）采用“实验数据”：对于比较复杂的算法，难以立即判断结果是否正确，为此可以设置“实验数据”，输入这些数据就可以判断运行正确与否。例如求解方程 $ax^2+bx+c=0$。

输入 a、b、c 的值分别为 1、-2、1 时，方程的根 x 的值是 1。这是容易判断的，若根不等于 1，程序显然有错。

（5）使用注释：调试程序时，可以将相对独立且验证正确的程序段注释起来，逐步缩小错误语句的查找范围，起到化繁为简的作用。当错误排除后，再取消注释。

（6）利用 debug（调试）工具：跟踪流程并给出相应信息，使用更为方便。

在编写程序过程中，出现程序错误是很经常的事。对于有经验的编程者而言，错误的产生大多是一时疏忽，而对于没有经验的编程者而言，除了疏忽因素以外，更多的是对语法的掌握不牢和对系统了解的不够。但不论是何类编程者，都要认真对待程序错误的查找与排除。否

则，即使是功能完善和强大的程序也是没有任何价值的。

查找程序错误是一项细致深入的工作，只有找到程序问题的所在，才能解决存在的问题。通常，人们把查找程序错误看作是程序调试的一部分，故统称为程序调试。从这个角度上讲，查找程序错误是程序调试的第一步，也是非常重要的一步。在此过程中，查找程序错误往往反映出一个人的调试水平、调试经验和科学态度。

§2.4 程序调试方法

上面讨论了查找程序错误的两种方法：静态检查和动态检查。静态检查是由人工对源程序进行逐条检查，发现和修改程序中的错误，而动态检查则是利用编译系统的“动态跟踪”来查找和修改程序中的逻辑错误和运行错误，因为语法错误可以通过编译出错信息来修改。

不同的编译系统，具有不同的调试方法，下面分别介绍在 Turbo C++ 3.0 和 Visual C++ 6.0 环境下的调试方法。

2.4.1 在 Turbo C++ 3.0 中调试程序

程序中的错误需要通过调试来排除，程序调试的任务就是排除程序中的错误，使程序能顺利地运行并得到预期的效果。在程序调试过程中不仅要发现和消除语法上的错误，而且还要发现和消除逻辑错误与运行错误。下面通过一个实例，介绍在 Turbo C++ 3.0 环境下几种排查逻辑错误的方法。

假如编程实现求组合数 $C_m^n = \frac{m!}{n!(m-n)!}$

求这样的组合数可以转化为求 3 次阶乘的运算。其中，求阶乘运算的公式为：

$m! = 1*2*3*\cdots*m$

这样，可以定义两个函数：求阶乘和利用阶乘求组合数。假设某用户编写的程序如下：

```
//求阶乘的源程序 cmn.c
#include<stdio.h>
float fac(int k)
{   int i,f;
    f=l;
    for(i=2;i<k;i++)
    f*=i;
    return (f);
}
float cmn(int m,int n)
{   int res;
    res=(fac(m)/fac(n)*fac(m-n));
    return (res);
}
main()
{   int m,n;
    float t;
    printf("m=");
    scanf("%d",&m);
```

```
    printf("n=");
    scanf("%d",&n);
    t=cmn(m,n);
    printf("c(%d,%d)=%10.2f\n",m,n,t);
}
```

该程序在 Turbo C++ 3.0 环境下编译形成可执行文件，运行可执行文件 cmn，输入：

m=3

n=2

输出的结果如下：

c(3,2)=2.00

在编译程序时并没有报错，但结果显然是错误的。因此，程序中肯定存在逻辑错误。调试的方法有以下几种。

1. 单步执行并观察变量值的变化

如果程序只由一个 main 函数构成，则采用单步执行，即按 F8 键或用 Run 菜单下的 Step over 命令；如果程序由多个函数构成，应采用跟踪执行，即按 F7 键或用 Run 菜单下的 Trace into 命令，F7 键与 F8 键的区别在于 F8 键会一步一步执行 main 函数中的语句，但不会跳转到相应的子函数中去，而采用 F7 键则会跳转到相应的子函数中去。按 F7 键，可以看到在编辑窗口中的源程序的主函数 main 处用高亮度显示，表示准备进入 main 运行，同时可以看到屏幕下部的 Message 窗口变成了 Watch 窗口，它是观察数据用的，如图 2-44 所示。

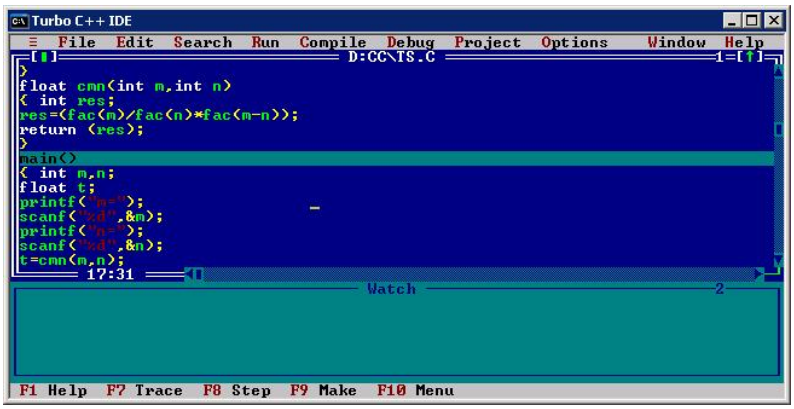

图 2-44　开始跟踪执行时的状态

再按一次 F7 键，亮条移到程序的第一条 printf 语句（因为 int m,n;和 float t;是对变量的定义，不是执行语句，故被跳过），此时 printf 语句并没有执行。再按一次 F7 键，输出 m=，亮条移到 scanf 语句。再按一次 F7 键，执行第一条 scanf 语句，需要输入数据，所以切换到用户屏幕，用户在此输入：

3↙

按 Enter 键后，屏幕显示切换到编辑窗口，亮条移到下一条 printf 语句，再按一次 F7 键，输出 n=，亮条移到 scanf 语句。再按一次 F7 键，执行下一条 scanf 语句，需要输入数据，所以切换到用户屏幕，用户在此输入：

2↙

按 Enter 键后，屏幕显示切换到编辑窗口，亮条移到下一条 t=cmn(m,n);语句。再按一次 F7 键，此时亮条移到 float cmn(int m,int n)。再按一次 F7 键，此时亮条移到赋值语句

res=(fac(m)/fac(n)*fac(m-n))，此时按 Ctrl+F7 组合键（或用 Break/Watch 菜单中的 Add watch 命令），弹出如图 2-45 所示的窗口。

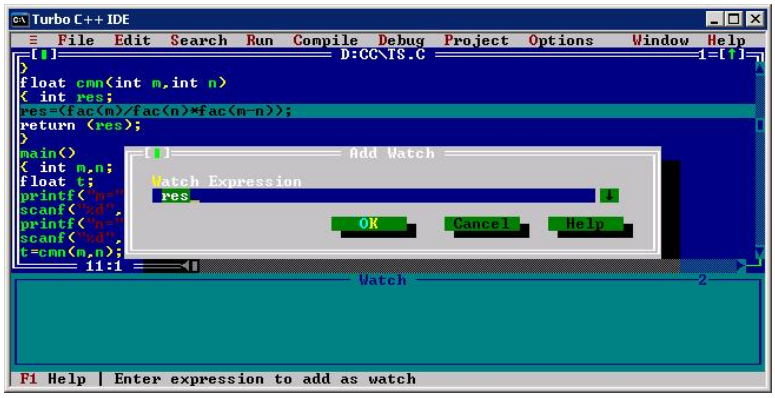

图 2-45　增加观察变量 res

在窗口中输入 res，按 Enter 键，该窗口消失，在 Watch 窗口显示出变量 res 的值，因为该赋值语句没有执行，所以 res 的值是一个随机数，并不是组合数的值。继续按 F7 键，跳转到 fac 函数去执行，使用上述同样的方法在 Watch 窗口中增加 k、f、i 三个变量，继续按 F7 键，直到亮条在 return(f)语句上，此时 Watch 窗口中变量的值如图 2-46 所示。

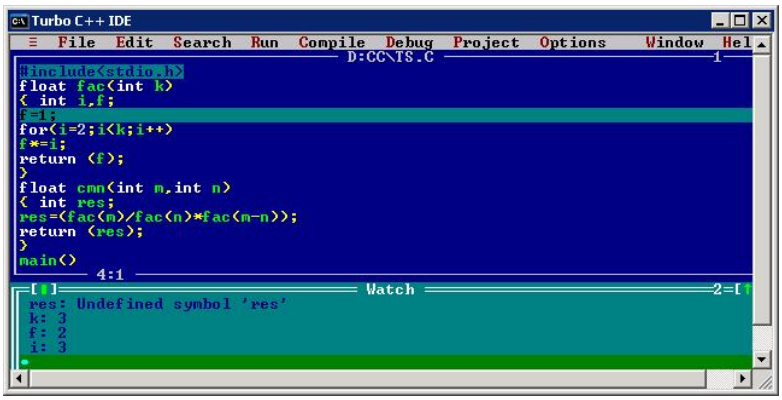

图 2-46　Watch 窗口中变量的值

由此可见，当 i=3 时，退出循环，fac(3)=2，这个值显然与 3!=6 不符。仔细分析，求 3!，只有当 i=4 时才跳出循环，而观察到的结果是当 i=3 时就跳出循环，所以循环中的条件有问题。经分析，i<k 应改为 i<=k，第一个错误被发现；使用同样的方法可以发现另一个错误为赋值语句 res=(fac(m)/fac(n)*fac(m-n)，应改为 res=fac(m)/(fac(n)*fac(m-n))。继续按 F7 键，直到所有的语句执行完毕。修改后重新运行该程序，结果正确。

通过跟踪执行和在 Watch 窗口设置观察变量，可以从变量的值的变化来查找程序中的逻辑错误。

2. 设置程序断点

上面单步跟踪执行能逐行、有效地检查一些关键数据的值，但如果程序很长，是难以逐行进行检查的。对于一个较长的程序，常用的方法是在程序中设若干个断点，程序执行到断点时暂停，检查此时有关变量或表达式的值，如果没有发现错误，就使程序继续执行到下一个断

点。如此一段一段地检查，查找程序中各处的出错点。

设置断点的方法是：将光标移到某一行上，然后按 Ctrl+F8 组合键或用 Break/Watch 菜单下的 Toggle Breakpoint 命令，此时以红色条覆盖，作为断点行。如果想取消断点行，则将光标移到断点行，再按一次 Ctrl+F8 组合键，红色条消失，断点行被取消。程序在运行到断点行时暂停，此时，用户可以用前面介绍过的方法查看有关变量或表达式的值，如果想继续运行，再按一次 Ctrl+F9 组合键即可。把光标移到 return(f)语句上，按 Ctrl+F8 组合键，然后再把光标移到 res=(fac(m)/fac(n)*fac(m-n))语句上，按 Ctrl+F8 组合键，即设置两个断点，如图 2-47 所示。

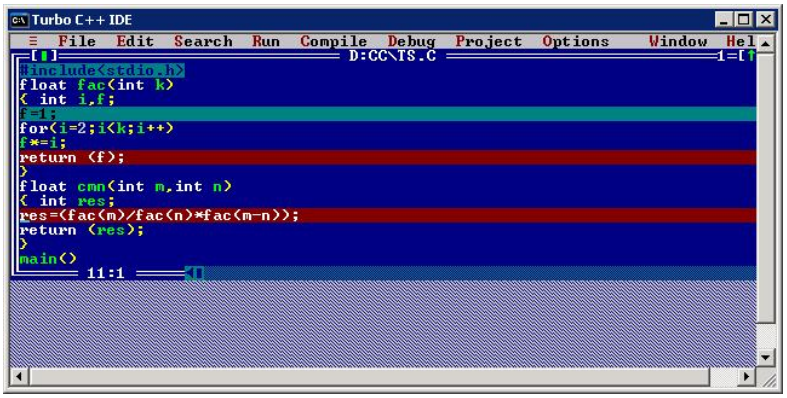

图 2-47　设置程序断点

按 Ctrl+F9 组合键，运行程序，分别输入变量 m、n 的值为 4、3，程序执行到第一个断点处暂停下来，即程序执行到了断点 return(f)处。此时连续按 Ctrl+F7 组合键三次，增加观察变量 i、k、f，在 Watch 窗口中，变量 i、k、f 的值分别为 4、4、6，即 4!=6。继续按 Ctrl+F9 组合键，在 Watch 窗口中，变量 i、k、f 的值分别为 3、3、2，即 3!=2。再一次按 Ctrl+F9 组合键，在 Watch 窗口中，变量 i、k、f 的值分别为 2、l、1，即 1!=1。很显然 3!和 4!结果都不对，进一步分析，求 4!时，i=4 就退出了循环，而求 3!时，i=3 就退出了循环，即 4!的结果应为 6*4=24，3!的结果应为 2*3=6。

因此不难发现，在 for 循环中的条件应改为 i<=k。按 Ctrl+C 组合键终止程序的执行，修改刚才发现的错误，取消在语句 return(f)上的断点，按上面的方法重新运行程序，分别输入变量 m、n 的值为 4、2，程序执行到断点停下来，此时按 Ctrl+F7 组合键，增加观察变量 res，在 Watch 窗口中发现 res 的值为 24，与结果 $c_4^2 = 6$ 不符。

通过检查，赋值语句 res=(fac(m)/fac(n)*fac(m-n))有误，应把该语句改为 res=fac(m)/(fac(n)*fac(m-n))。按 Ctrl+C 组合键终止程序，把错误改过来，重新运行，结果与预期的一致，程序正确，调试完毕。

3. 用 Evaluate 命令来查看变量的值

在前两种方法中，使用 Watch 窗口增加观察变量，通过查看变量的值来调试程序。除此之外，还可以使用 Debug 菜单中的 Evaluate 命令或按 Ctrl+F4 组合键，它不仅可以查看有关变量和表达式的值，还可以修改它们的值，以帮助用户动态调试程序。

把光标移到 return(f)语句上，按 Ctrl+F8 组合键，然后再把光标移到 return(res)语句上，按 Ctrl+F8 组合键，设置两个断点，如图 2-28 所示。按 Ctrl+F9 组合键，运行程序，分别输入变

量 m、n 的值为 4、2，程序执行到第一个断点处暂停下来，即程序执行到了断点 return(0)，此时按 Ctrl+F4 组合键，弹出 Evaluate 窗口，如图 2-48 所示。

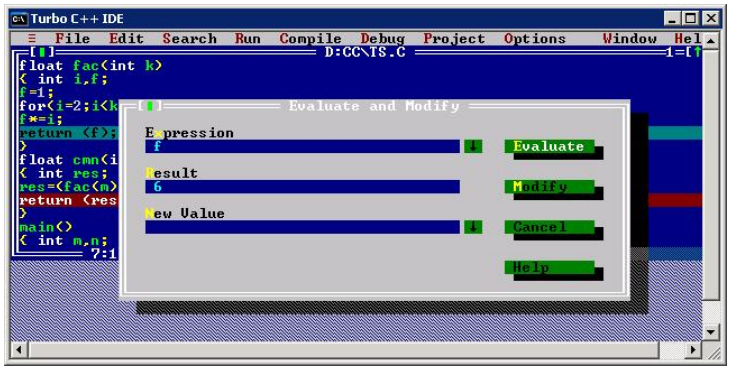

图 2-48　Evaluate 窗口

在 Evaluate 输入框中输入变量 f，按 Enter 键，Result 输入框中则出现 6，即 4!=6，正确的结果应是 4!=24，所以求阶乘的子函数有错误，在 New Value 输入框中输入正确的值 24，继续按 Ctrl+F9 组合键，此时再按 Ctrl+F4 组合键，弹出 Evaluate 窗口，在 Evaluate 输入框中输入变量 f，按 Enter 键，Result 输入框中则出现 1，即 2!=1。正确的结果应是 2!=2，所以求阶乘的子函数有错误，在 New Value 输入框中输入正确的值 2。再一次按 Ctrl+F9 组合键，Result 输入框中则出现 1，即 2!=1。正确的结果应是 2!=2，所以求阶乘的子函数有错误，在 New Value 输入框中输入正确的值 2。很显然 2!和 4!结果都不对，进一步分析，求 4!时，i=4 就退出了循环，而求 2!时，i=2 就退出了循环，即 4!的结果应为 6*4=24，2!的结果应为 2*1=2。因此，不难发现在 for 循环中的条件应改为 i<=k。继续按 Ctrl+F9 组合键，此时再按 Ctrl+F4 组合键，弹出 Evaluate 窗口，在 Evaluate 输入框中输入变量 res，按 Enter 键，Result 输入框中则出现 24，与结果 $c_4^2 = 6$ 不符，通过检查，赋值语句 res=(hc(m)/fae(n)*fac(m-n))有误，该语句应改为：res=fac(m)/(fac(n)*fac(m-n))。按 Ctrl+C 组合键终止程序，把错误改过来，重新运行，结果与预期的一致，程序正确，调试完毕。

上面介绍了三种调试程序的方法，尤其是当程序比较复杂时，这三种方法很有效，但并不是每一个程序都要用这些方法，在学习的过程中要灵活运用。

2.4.2　在 Visual C++ 6.0 中调试程序

在软件开发中，大部分工作往往都体现在程序调试上。程序调试过程一般按照修改语法错误、修改逻辑错误、检查运行结果的步骤进行。在 Visual C++ 6.0 环境下利用调试器实现跟踪调试的方法与 Turbo C++ 3.0 基本相似。调试的实例程序如下：

```
#include <stdio.h>
main()
{   int a,b,c,d;
    scanf("%d%d",&a,&b);
    c=a+b;
    d=a-b;
    printf("%3d%3d\n",c,d);
}
```

1. 程序执行到中途暂停以便观察阶段性结果

方法一：使程序执行到光标所在的那一行暂停。在需暂停的行上单击鼠标，定位光标，然后选择菜单中的 Build→Start Debug→Run to Cursor 命令，或按 Ctrl+F10 组合键，程序执行到光标所在行会暂停。如果把光标移动到后面的某个位置，再按 Ctrl+F10 组合键，程序将从当前的暂停点继续执行到新的光标位置，第二次暂停，如图 2-49 所示。

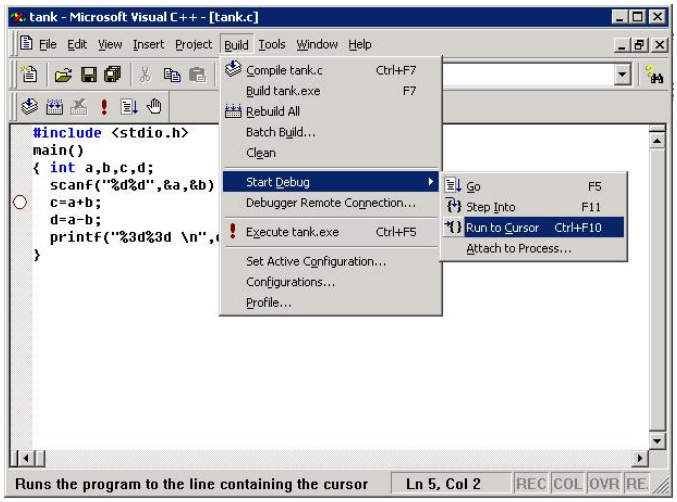

图 2-49　执行到光标所在行暂停

方法二：在需暂停的行上设置断点。在需设置断点的行上单击鼠标，定位光标，然后按“编译”工具条中最右面的按钮，或按 F9 键设置断点。被设置了断点的行前面会有一个红色圆点标志，如图 2-50 所示。

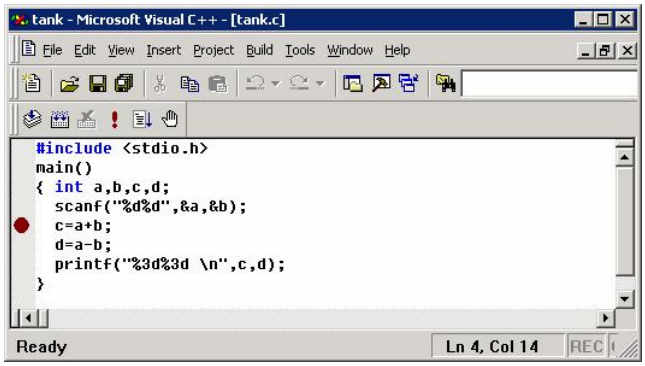

图 2-50　设置断点

2. 设置需观察的结果变量

按照上面的操作，使程序执行到指定位置时暂停，目的是为了查看有关的中间结果。在图 2-51 中，左下角窗口中系统自动显示了有关变量的值，其中 a 和 b 的值分别是 5、6，而变量 c、d 的值是不正确的，因为它们还未被赋值。图中左侧的箭头表示当前程序暂停的位置。如果还想增加观察变量，可在图中右下角的 Name 框中填入相应变量名。

3. 单步执行

当程序执行到某个位置时发现结果已经不正确了，说明在此之前肯定有错误存在。如果

能确定一小段程序可能有错，则先按上面步骤暂停在该小段程序的头一行，再输入若干个查看变量，然后单步执行，即一次执行一行语句，逐行检查下来，观察错误发生在哪一行，如图2-52 所示。

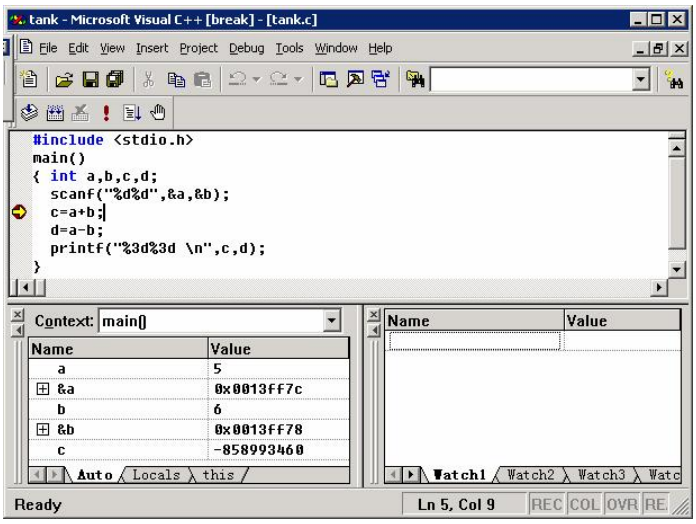

图 2-51　观察结果变量

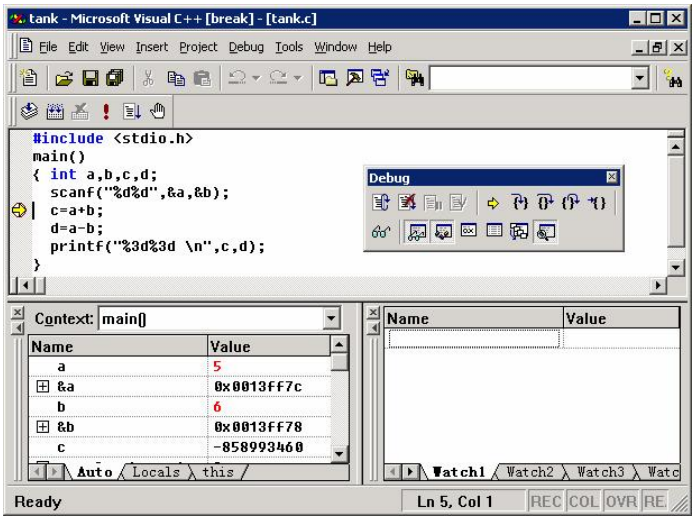

图 2-52　单步执行

运行当前箭头指向的代码即单步执行按 Step Over 按钮或 F10 键；如果当前箭头所指的代码是一个函数的调用，想进入函数进行单步执行，可按 Step Into 按钮或 F11 键；如果当前箭头所指向的代码是在某一函数内，想结束函数的单步执行，使程序运行到函数返回处，可按 Step Out 按钮或 Shift+F11 键。对不是函数调用的语句来说，F11 键与 F10 键作用相同。但一般对系统函数不要使用 F11 键。

4. 取消程序断点

使用断点可以使程序暂停，但每次执行程序都会在断点上暂停。因此调试结束后应取消所定义的断点。方法是：先把光标定位在断点所在行，再按“编译”工具条中最右面的按

钮或 F9 键，该操作是一个开关，按一次是设置，按两次是取消设置。如果有多个断点想全部取消，可执行 Edit 菜单中的 Breakpoints 菜单项，屏幕上会显示 Breakpoints 窗口，如图 2-53 所示。窗口下方列出了所有断点，按 Remove All 按钮，将取消所有断点。

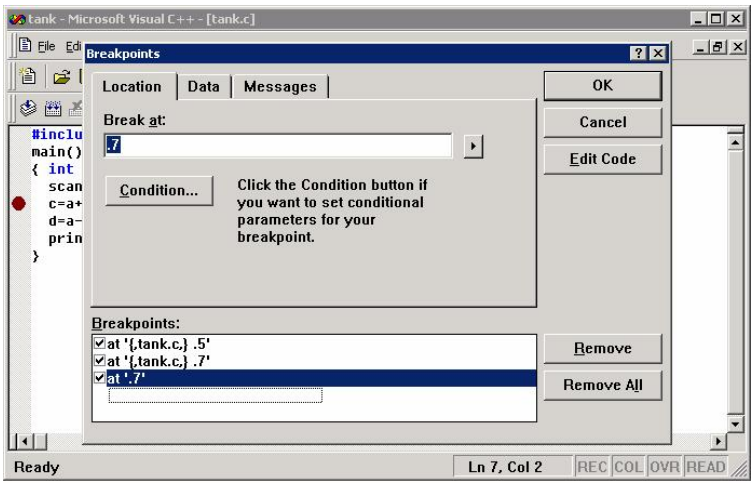

图 2-53　取消所有断点

断点通常用于调试较长的程序，可以避免使用 Run to Cursor（运行程序到光标处暂停）或 Ctrl+F10 功能时，经常要把光标定位到不同的地方。而对于长度为上百行的程序，要寻找某位置并不太方便。如果一个程序设置了多个断点，按一次 Ctrl+F5 组合键会暂停在第一个断点，再按一次 Ctrl+F5 组合键会继续执行到第二个断点暂停，依次执行下去。

5. 使用 Debug 调试

使用 Debug 菜单的 Stop Debugging 菜单项，或 Shift+F5 组合键可以结束调试，从而回到正常的运行状态，如图 2-54 所示。

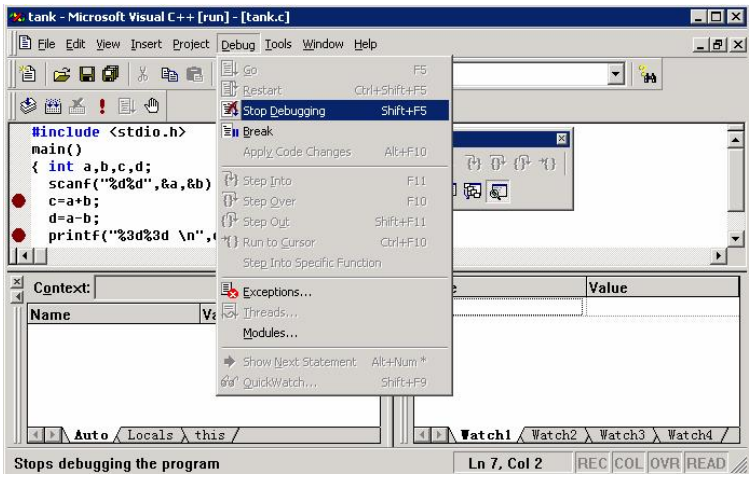

图 2-54　停止调试

以上调试是排除程序错误的基本方法和过程。为了使程序运行结果可靠，最后还应对调试过的程序进行测试，查找程序中存在的错误和缺陷。