第 1 部分　Visual Studio 6.0 开发环境介绍

本书以 Visual C++ 6.0 作为 C 源程序的实践开发环境，本章将首先介绍 Visual C++ 6.0 环境的基本操作，包括 Visual C++ 6.0 的安装和启动，C 源程序的编辑、运行与调试。

1.1　安装与启动

如果计算机上未安装 Visual C++ 6.0，则按照安装向导直接安装，具体步骤此处不再详述。安装过程中建议同时安装 MSDN，以便日后自学。

成功安装 Visual C++ 6.0 后，可以在桌面上看到如图 1.1 所示的图标或通过“开始”→“程序”→“Microsoft Visual Studio”找到“Visual C++ 6.0”启动程序。

双击 Visual C++ 6.0 图标或者单击“开始”→“程序”→“Microsoft Visual Studio”→“Visual C++ 6.0”即可启动 Visual C++ 6.0 集成开发环境，正常启动开发环境后，可见如图 1.2 所示的主窗口。

图 1.1　Visual C++ 6.0 图标

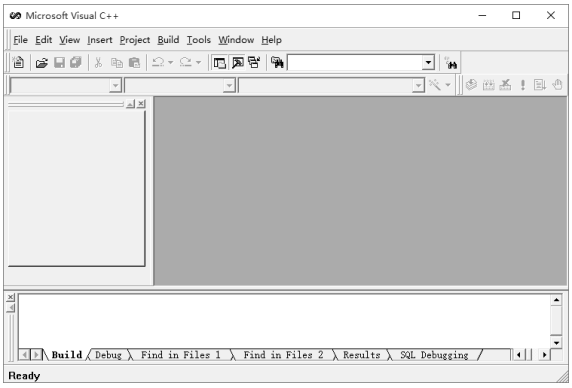

图 1.2　Visual C++ 6.0 主窗口

1.2　环境介绍

编译一个 C 源程序的前提是新建一个程序或者打开现有程序。本节首先介绍如何新建一个程序，如何打开现有程序，并在此基础上对文件进行编译。

1．新建一个工程

在 Visual C++ 6.0 主窗口中的主菜单栏中单击“File（文件）”菜单项，然后单击“New（新建）”菜单项，也可按 Ctrl+N 组合键进入“NEW（新建）”对话框，如图 1.3 所示。

出现如图 1.4 所示的一个“New（新建）”对话框，单击此对话框上方的“Projects（工程）”标签，在其下方菜单栏选择“Win32 Console Application”项，输入工程名，设定存储路径（Location）然后单击“OK”按钮。

单击“OK”按钮后，会弹出如图 1.5 所示的对话框，选中“An empty project”单选按钮，单击“Finish”按钮。

之后弹出如图 1.6 所示的对话框。注意：该对话框上显示了工程文件的。

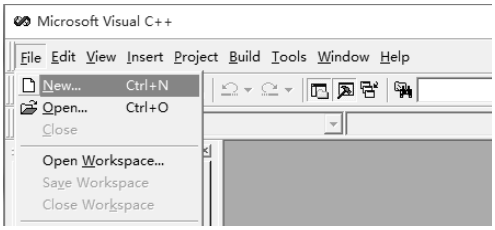

图 1.3　新建工程的操作过程

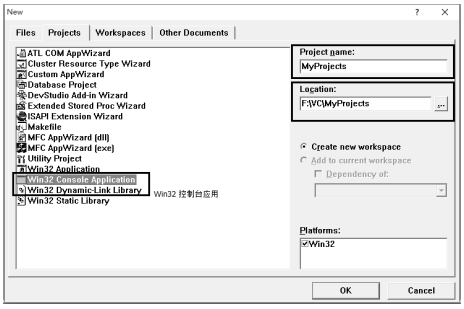

图 1.4　“New”对话框

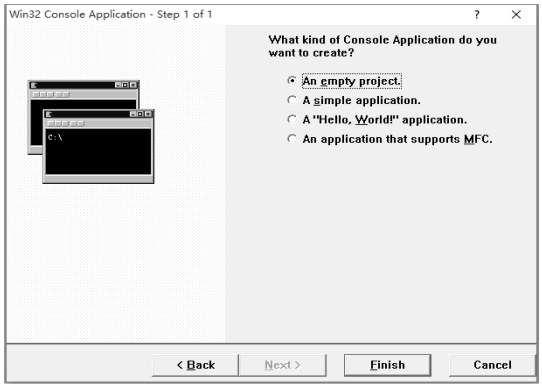

图 1.5　工程类型选择

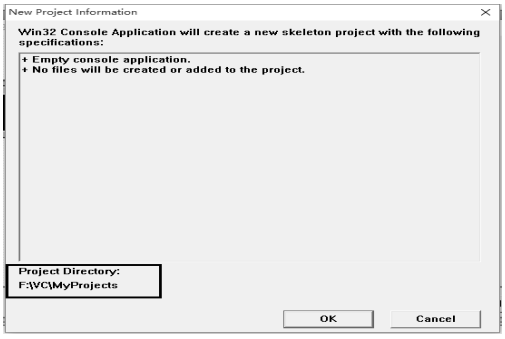

图 1.6　新工程信息

单击“OK”按钮，则进入工程编辑窗口，如图 1.7 所示。在工程编辑窗口可以看到 MyProjects 工程下只有三个空文件夹。

此后，就可以在当前工程下创建一个 C 源程序。

2．新建一个 C 源程序

单击“File”→“New”，在打开如图 1.8 所示的新建对话框中，类似新建工程操作，在“Files”标签选择“C++ Source File”项；然后选中“Add to Project”复选框，并选择添加到新建的工程“MyProjects”；在“File”输入框中，输入后缀名是.c 的文件名，单击“OK”按钮。

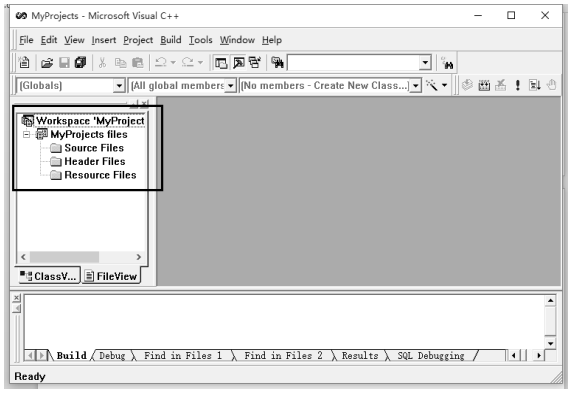

图 1.7　MyProjects 工程编辑窗口

注意，如果在文件名中不显示扩展名.c，则 VC++将为文件附上默认扩展名.cpp，并按照 C++语言的语法进行检查。由于 C++的语法检查要比 C 语言的语法更为严格，因此，建议还是输入文件扩展名.c。

单击“OK”按钮后，可以进入 MyFile.c 的编辑窗口，如图 1.9 所示在窗口的标题栏和工程窗口中都显示出了当前要编辑的文件名字“MyFile.c”。

此时，程序编辑窗口被激活，可以输入和编辑源程序了。这里编写了一个简单的 C 程序，输出“Hello World！”，如图 1.10 所示。

3．打开一个现存的程序

与新建的操作类似，可以通过“File”→“Open Workspace”项进入工程窗口，如图 1.11 所示。双击工程文件（扩展名为.dsw），即可打开已有的工程。此外，也可以通过双击工程文件直接打开现有程序。

进入编辑状态后，如果对程序进行了修改且未保存则在标题栏中文件名字后面会出现“*”提示，选择“File”→“Save”、单击 按钮或者使用 Ctrl+S 组合键都可以保存文件。保存之后，标

题栏的“*”消失。

如果不想将源程序保存到指定的文件中，可以选择“File”→“Save As”项，重新指定文件保存的位置以及文件名字。

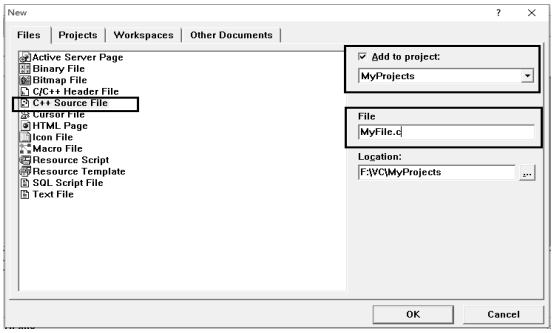

图 1.8　新建 C 源程序

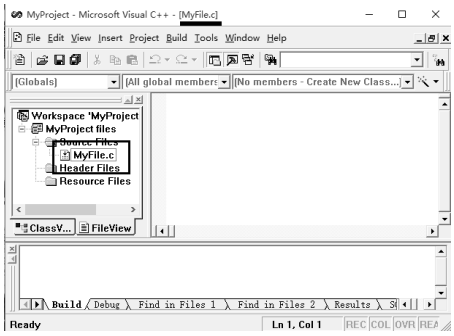

图 1.9　MyFile.c 编辑窗口

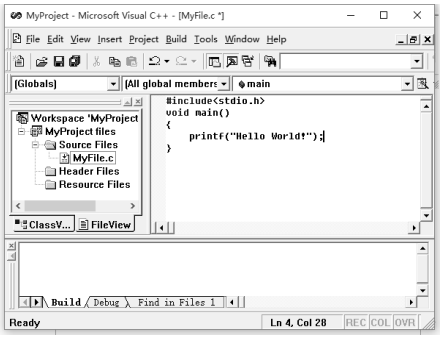

图 1.10　MyFile.c 的编辑

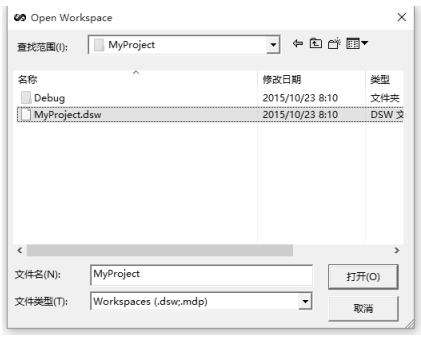

图 1.11　打开工程窗口

1.3　编译、链接和运行程序

1．程序编译

编译可以检查程序中是否存在语法错误并生成目标文件（.obj），编译结果显示在输出窗口中，如图 1.12 所示。

选择“Build”→“Compile MyFile.c”？即可对 MyFile.c 进行编译。此外，也可以通过单击 按钮或者直接按 Ctrl+F7 组合键进行编译。

如果程序中存在语法错误，则可以通过双击错误提示在程序文件中定位到错误所在的代码行。例如，把 printf 语句的分号去掉，则编译出现如图 1.13 所示的结果。根据提示框中的提示信息“syntax error; missing ';' before '}'”以及编辑区中的蓝色提示箭头则可以直接定位到错误处进行修改。

值得注意的是，语法错误分为 error 和 warning 两类。error 是一类致命错误，程序中如果有此类错误则无法生成目标程序，更不能执行。warning 则是相对轻微的一类错误，不会影响目标文件及可执行文件的生成，但是有可能影响程序的运行结果。因此，建议最好把所有错误（无论是 error 还是 warning）一一修正。

2．程序链接

链接将生成可执行文件（.exe）。选择“Build”→“Build”选项可以完成程序的链接。此外，也可以通过单击 按钮或通过快捷键 F7 进行链接。类似的，链接后的结果也会出现在输出窗口中，如图 1.14 所示。如果链接失败，则同样会显示失败的具体原因。

3．程序运行

选择“Build”→“Execute MyFile.exe”选项可以启动程序的执行。此外，也可以通过单击 按

钮或通过 Ctrl+F5 组合键启动程序的运行。开始运行后，将弹出一个 DOS 窗口，显示程序的运行结果，如图 1.15 所示。

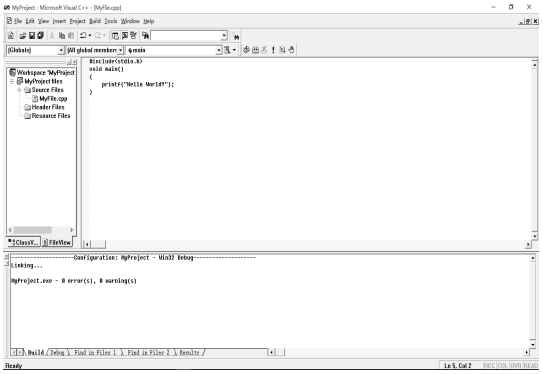

图 1.12　程序编译结果

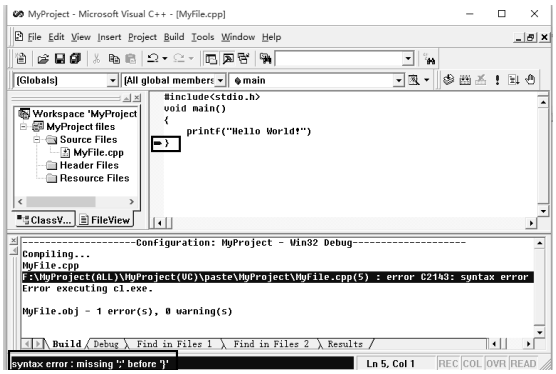

图 1.13　语法错误

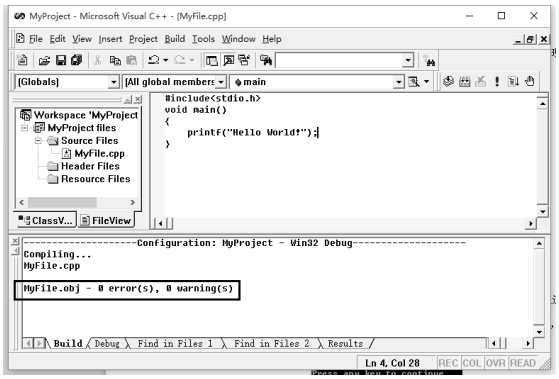

图 1.14　程序链接结果

图 1.15　程序运行结果

1.4　调试程序

调试是为了发现程序中的错误，包括语法错误和逻辑错误。其中，语法错误能够在编译的过程中发现并修改；而逻辑错误往往无法直观地被发现，即程序通常能够被成功地编译和链接甚至执行，然而其执行结果却与预期结果不一致。逻辑错误的调试是比较困难的，因此，一般的程序开发环境都会提供完整的程序调试工具。本节将主要介绍如何使用 Visual C++ 6.0 中的调试工具发现程序中的逻辑错误。

1．设置断点

当需要调试程序时，可以首先大致判断出程序中可能从哪条语句开始出现问题，并将光标移动到该语句行，单击按钮，则在语句行左侧出现一个红点，称为断点，如图 1.16 所示。此后，程序调试运行过程中遇到断点时，将自动暂停执行，进入调试状态。

2．调试界面

设置好断点之后，单击按钮，以调试的方式启动程序的运行。当运行到断点所在的语句时，将直接进入调试状态，如图 1.17 所示。代码行左侧的黄色小箭头表示了程序的当前执行位置。调试工具条中的按钮是常用的调试工具按钮，表 1.1 中列举了常用按钮的功能。屏幕下方左右两个窗口则分别是自动变量框和观察框，可以看到变量的当前值，作为判断程序是否出错的参考。其中，观察框支持对各种变量值的观察，在“Name”列输入变量的名字，则在“Value”列会显示出该变量的当前值。

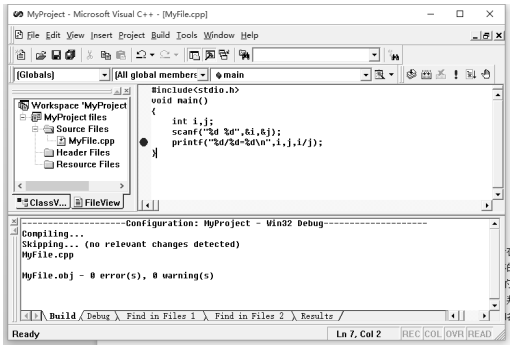

图 1.16　设置断点

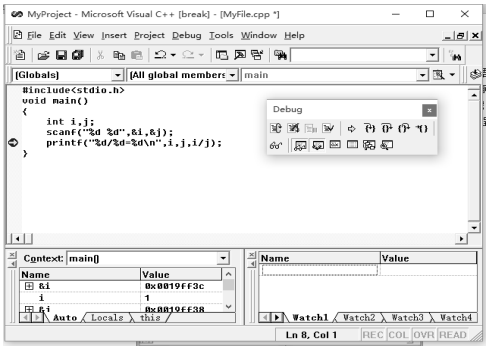

图 1.17　调试界面

表 1.1　调试工具常用按钮说明

按　钮	功　能	按　钮	功　能
	重新开始调试		停止/退出调试
	跟踪到函数中		逐个函数调试
	从跟踪的函数跳出		运行到光标处

3．常见错误

C 语言的最大特点是：功能强、使用方便灵活。C 编译的程序对语法检查并不像其他高级语言那么严格，这就给编程人员留下“灵活的余地”，但还是由于这个灵活给程序的调试带来了许多不便，尤其对初学 C 语言的人来说，经常会出一些连自己都不知道错在哪里的错误。在本节编者整理了部分常见错误，及它们的解决办法。

（1）书写标识符时，忽略了大小写字母的区别。

```
main()
{  int a=5;
   printf("%d",A);
}
```

提示：编译程序把 a 和 A 认为是两个不同的变量名，而显示出错信息。C 语言认为大写字母和小写字母是两个不同的字符。习惯上，符号常量名用大写表示，变量名用小写表示，以增加可读性。

（2）忽略了变量的类型，进行了不合法的运算。

```
main()
{  float a,b;
   printf("%d",a%b);
}
```

提示：%是求余运算，得到 a/b 的整余数。整型变量 a 和 b 可以进行求余运算，而实型变量则不允许进行“求余”运算。

（3）将字符常量与字符串常量混淆。

```
char c;
c="a";
```

提示：在这里就混淆了字符常量与字符串常量，字符常量是由一对单引号括起来的单个字符，字符串常量是一对双引号括起来的字符序列。C 语言规定以“\”作字符串结束标志，它是由系统自动加上的，所以字符串"a"实际上包含'a'和'\'两个字符，而把它赋给一个字符变量是不行的。

（4）忽略了“=”与“==”的区别。

C 语言中，“=”是赋值运算符，“==”是关系运算符。例如：

```
if (a==3)a=b;
```

提示：前者是进行比较，a 是否和 3 相等，后者表示如果 a 和 3 相等，把 b 值赋给 a。由于习惯问题，初学者往往会犯这样的错误。

（5）忘记加分号。

分号是 C 语句中不可缺少的一部分，语句末尾必须有分号。

```
a=1
b=2
```

提示：编译时，编译程序在“a=1”后面没发现分号，就把下一行“b=2”也作为上一行语句的一部分，这就会出现语法错误。改错时，有时在被指出有错的一行中未发现错误，就需要看一下上一行是否漏掉了分号。

（6）多加分号。

对于一个复合语句，如：

```
{ z=x+y;
  t=z/100;
printf("%f",t);
};
```

提示：复合语句的花括号后不应再加分号，否则将会画蛇添足。

（7）输入变量时忘记加地址运算符“&”。

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

提示：这是不合法的。scanf 函数的作用是：按照 a、b 在内存的地址将 a、b 的值存进去。“&a”指 a 在内存中的地址。

（8）输入数据的方式与要求不符。例如：

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

输入时，不能用逗号作两个数据间的分隔符，如下面输入不合法：

```
3，4
```

输入数据时，在两个数据之间以一个或多个空格间隔，也可用 Enter 键、Tab 键。

又如：

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

提示：C 规定，如果在“格式控制”字符串中除了格式说明以外还有其他字符，则在输入数据时应输入与这些字符相同的字符。

（9）输入字符的格式与要求不一致。

在用“%c”格式输入字符时，“空格字符”和“转义字符”都作为有效字符输入。例如：

```
scanf("%c%c%c",&c1,&c2,&c3);
```

如输入“a　b　c”。

提示：字符“a”送给 c1，字符“ ”送给 c2，字符“b”送给 c3，因为%c 只要求读入一个字符，后面不需要用空格作为两个字符的间隔。

（10）输入输出的数据类型与所用格式说明符不一致。

例如，a 已定义为整型，b 定义为实型：

```
a=3;b=4.5;
printf("%f%d\n",a,b);
```

提示：编译时不给出出错信息，但运行结果将与原意不符。这种错误尤其需要注意。

（11）输入数据时，企图规定精度。例如：

```
scanf("%7.2f",&a);
```

提示：输入数据时不能规定精度。

（12）switch 语句中漏写 break 语句。

例如：根据考试成绩的等级打印出百分制数段。

```
switch(grade)
{  case 'A':printf("85~100\n");
   case 'B':printf("70~84\n");
   case 'C':printf("60~69\n");
   case 'D':printf("<60\n");
```

```
    default:printf("error\n");
```

提示：由于漏写了 break 语句，case 只起标号的作用，而不起判断作用。因此，当 grade 值为 A 时，printf 函数在执行完第一个语句后接着执行第二、三、四、五个 printf 函数语句。正确写法应在每个分支后再加上“break;”。

（13）忽视了 while 和 do-while 语句在细节上的区别。

例如：

```
main()
{  int a=0,I;
  scanf("%d",&I);
  while(I<=10)
  {  a=a+I;
    I++;
  }
  printf("%d",a);
}
```

又如：

```
main()
{  int a=0,I;
   scanf("%d",&I);
   do
 {  a=a+I;
    I++;
 }  while(I<=10);
    printf("%d",a);
}
```

提示：可以看到，当输入 I 的值小于或等于 10 时，两者得到的结果相同。而当 I>10 时，两者结果就不同了。因为 while 循环是先判断后执行，而 do-while 循环是先执行后判断。对于大于 10 的数，while 循环一次也不执行循环体，而 do-while 语句则要执行一次循环体。

（14）定义数组时误用变量。例如：

```
int n;
scanf("%d",&n);
int a[n];
```

提示：数组名后用方括号括起来的是常量表达式，可以包括常量和符号常量。即 C 不允许对数组的大小作动态定义。

（15）在定义数组时，将定义的“元素个数”误认为是可使的最大下标值。例如：

```
main()
{  static int a[10]={1,2,3,4,5,6,7,8,9,10};
   printf("%d",a[10]);
}
```

提示：C 语言规定，定义时用 a[10]，表示 a 数组有 10 个元素。其下标值由 0 开始，所以数组元素 a[10]是不存在的。

（16）初始化数组时，未使用静态存储。例如：

```
int a[3]={0,1,2};
```

提示：C 语言规定只有静态存储（static）数组和外部存储（extern）数组才能初始化。应改为：

```
static int a[3]={0,1,2};
```

（17）在不应加地址运算符&的位置加了地址运算符。例如：

```
scanf("%s",&str);
```

提示：C 语言编译系统对数组名的处理是数组名代表该数组的起始地址，scanf 函数中输入项是字符数组名，不必再加地址符“&”。改为：

```
scanf("%s",str);
```