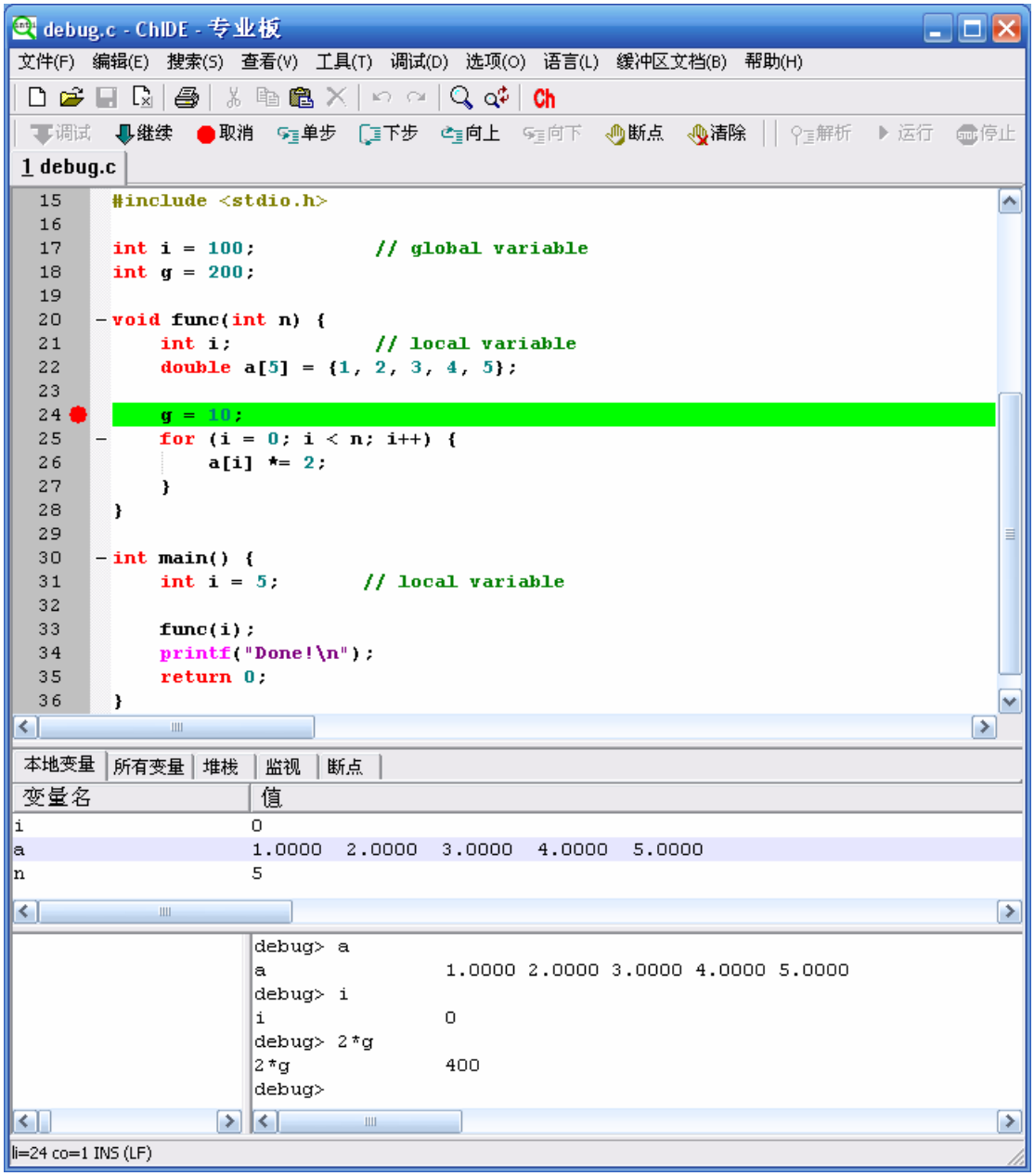
ChIDE集成开发环境和Ch命令窗口使用入门

Ch 版本 6.3



SoftIntegration的联系方式
通讯地址：SoftIntegration,Inc. 216 F Street, #68 Davis, CA 95616，USA
电 话：+1 530 297 7398
传 真：+1 530 297 7398
网 址：http://www.softintegration.com
电子邮箱：info@softintegraion.com

版本6.3.0，2010年11月。

.

目　录

1. 引言

Ch 是一个可嵌入的 C/C++程序解释器，是 C 语言的超集，它支持 C++中的类、C99 标准中的大部分功能及用户友好的高级特性，可应用于跨平台脚本、嵌入式脚本、shell 编程、二维和三维绘图、数值运算及快速仿真等。由于 ch 具有优越的数值计算功能，它可以方便地用于科学和工程领域。除此之外，Ch 在 C/C++程序设计教学方面，尤其是在课堂交互式展示教学及易于学生学习等方面具有独特的优越性。

ChIDE 是一个 C 和 C++程序开发的集成开发环境，可以在其中编写与运行程序，在其中编写程序时，它可以自动用不同颜色突出显示不同的语法结构；ChIDE 是一个跨平台的集成开发环境，它集编辑、调试和运行 C/Ch/C++程序为一体，可以在不编译程序的情况下解释运行程序；在此环境中调试程序时，用户可以设定断点、单步执行程序、随时查看变量的值；ChIDE 用 Embedded Ch 语言开发，它是一种最适合于初学 C/C++语言者使用的集成开发环境；它还能够以用户指定的编译器编译程序，如基于 Window 平台的 Microsoft Visual Studio .NET、基于 Linux 与 Mac OS X x86 的 gcc/g++等。

由于 Ch 是个解释器，C/C++表达式、语句、函数及程序可以不编译而在其中直接运行，所以它是 C/C++语言教学使用最理想的工具。教师在课堂上可以利用笔记本电脑快速地、交互地展示编程的方法，在回答学生问题时更是如此。对于 C/C++语言学习者，可以避开冗长地、乏味地编译、连接、执行、调试程序的过程，很快地学会 C/C++的新特性。为了帮助初学者，在程序出现错误时，Ch 特别加入了许多告警和错误信息，而不是晦涩难懂的、诸如“段错误”、“总线错误”等提示信息或程序直接崩溃。

本文将简要介绍怎样快速利用 Ch 集成开发环境 ChIDE 及 Ch 命令窗口来学习计算机编程和编写 C/Ch/C++程序。

2. 在 ChIDE 中执行 C/Ch/C++程序

2.1. 启动程序

在不同的系统中，ChIDE均可以通过运行**chide**程序来启动。

在Windows操作系统中，可以很方便地通过双击如图 1所示的桌面图标来启动ChIDE。

在Mac OS X x86系统中，ChIDE也可以在相应的应用程序文件夹（Applicatoins folder）或面板(dashboard)中点击如图 1所示的图标来启动。

图 1 Windows、Mac OS X 和 Linux中的ChIDE 图标

在 Linux 系统中，可以点击开始菜单的进入程序工具(Programming Tools)来启动。可以使用命令

```
ch -d
```

在桌面上创建 Ch 图标。当安装的 Ch 带有 ChIDE 时，桌面上也会创建一个 ChIDE 图标。

2.2. 编辑和执行 C/Ch/C++程序

ChIDE文本编辑工作方式类似于 Macintosh 或 Windows 中的记事本等编辑器，但增加了语法高亮显示功能。在ChIDE中，默认设置可以同时打开20个文件，但只有一个文件是可见的，详见3.7节介绍。

例如，在 ChIDE中，新建一个文件，输入如下代码：

```
/* File: hello.c
   Print 'Hello, world' on the screen. */
#include <stdio.h>

int main() {
   printf("Hello, world!\n");
   return 0;
}
```

如图 2所示，在ChIDE的编辑窗格中，对不同的语法结构用不同颜色突出显示。

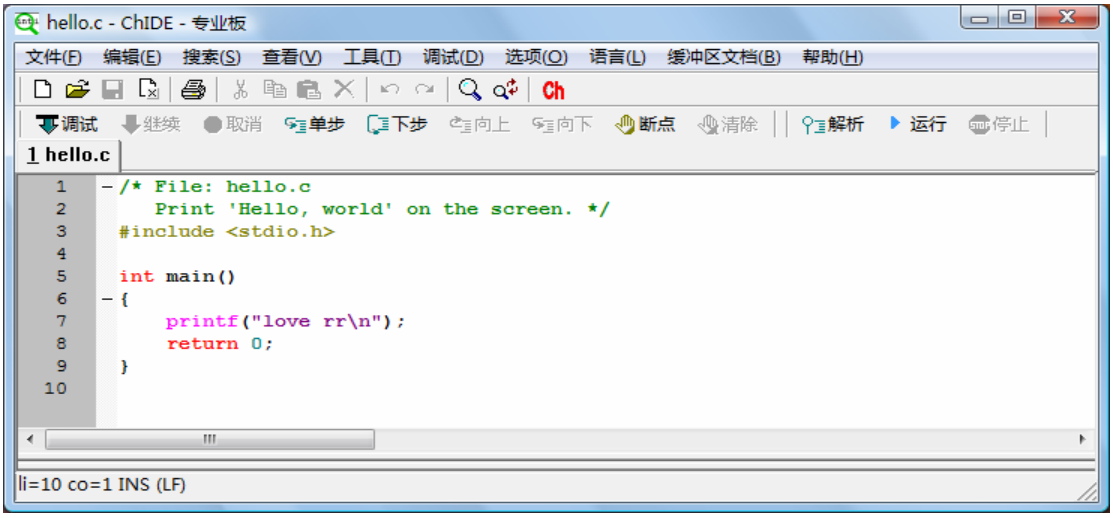


图 2 ChIDE编辑窗格中的程序

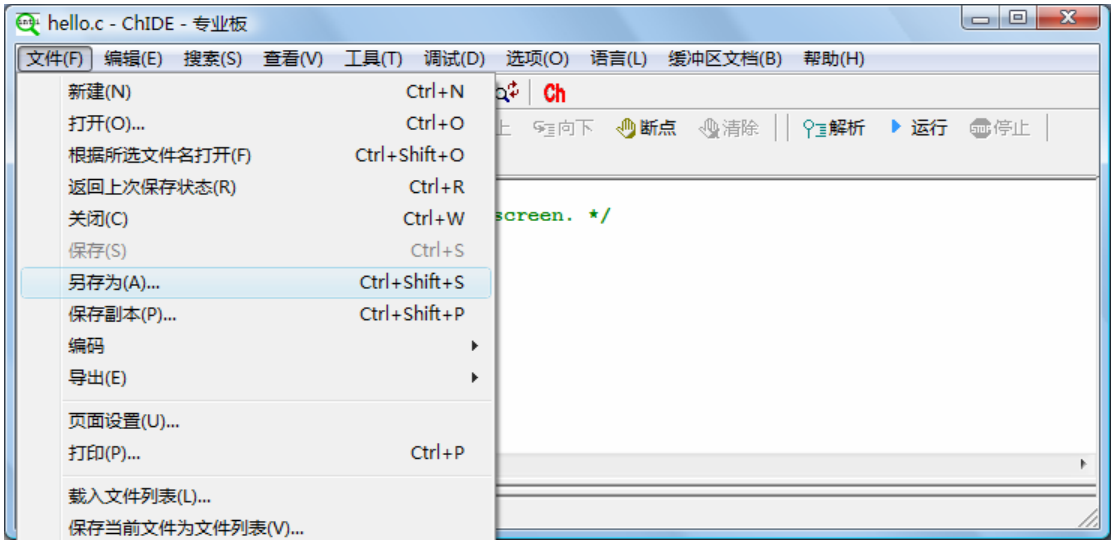


图 3 在 ChIDE 中保存编辑的程序

这段代码与保存在CHHOME/demos/bin/路径下的程序hello.c相同，这里的CHHOME是Ch的主目录，在Windows操作系统中Ch默认的主目录是C:/Ch;在Mac操作系统中Ch默认的主目录是/usr/local/ch。可以通过点击菜单“文件丨打开”来调入此文件。默认情况下，当ChIDE启动时会自动打开此文件。在Windows中，还可以将资源管理器中列出的文件拖放入ChIDE中将其打开。

如图 3所示，通过点击“文件|另存为”将此文件保存为 hello.c。也可如图 4所示，在调试工具栏下方显示文件名的标签上右击，打开右键菜单，找到命令“另存为”来保存该文件。

如图 5所示，编辑窗格左边显示的行号、书签页边列以及折叠状态列可通过点击菜单“查看|行号、查看丨书签页边列、查看丨折叠状态列”来隐藏或显示。可以通过单击折叠状态列中的折叠符号“+”或“-”来展开和收起一个程序块。

如图 6所示，ChIDE有四种窗格：编辑窗格、调试窗格、调试命令窗格和输出窗格。调试窗格位于编辑窗格的下面或右边，它的大小被初始化为零，但可以通过拖动它与编辑窗格之间的分隔条来改变其大小；调试命令窗格位于调试窗格的下面或右边。关于调试窗格与调试命令窗格的详细说明请参见第4部分。输出窗格位于调试窗格的下面或右边、调试命令窗格的左边，它的大小被初始化为零，同样可以通过拖动它和调试窗格之间的分隔条来改变其大小。在默认状态下，程序的运行结果会直接显示在输出窗格中。

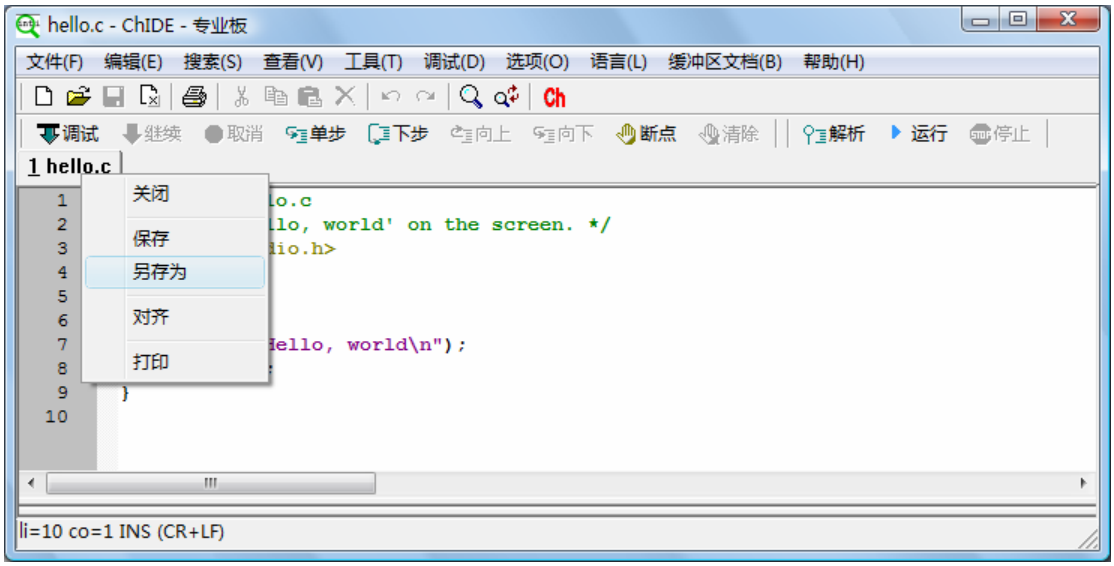


图 4 右键单击文件名所在标签来保存编辑的程序

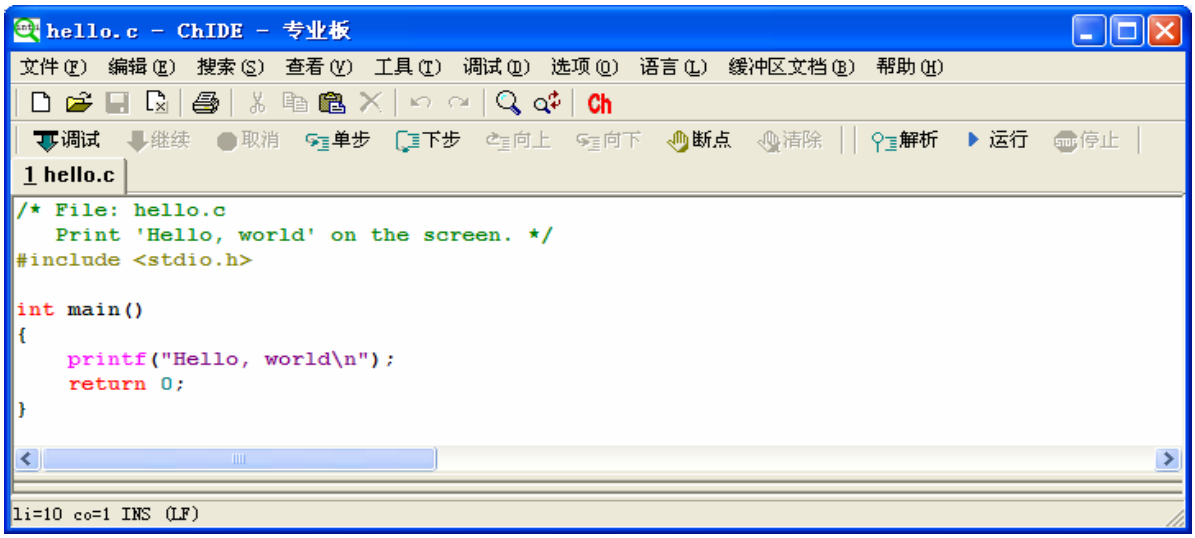


图 5 未显示行号、书签页边列以及折叠状态列的程序

点击菜单“查看|窗格垂直并列”可以将 ChIDE 的各个窗格变为垂直布局。在这种布局下，编辑窗格在最左边，调试窗格在中间，调试命令窗格和输出窗格在右侧。当关闭 ChIDE 时，ChIDE 的当前布局将会被保存。再次打开 ChIDE 的时候，将采用上次保存的布局。可以通过点击菜单“查看|使用默认布局”使 ChIDE 的窗格布局变为默认的布局。

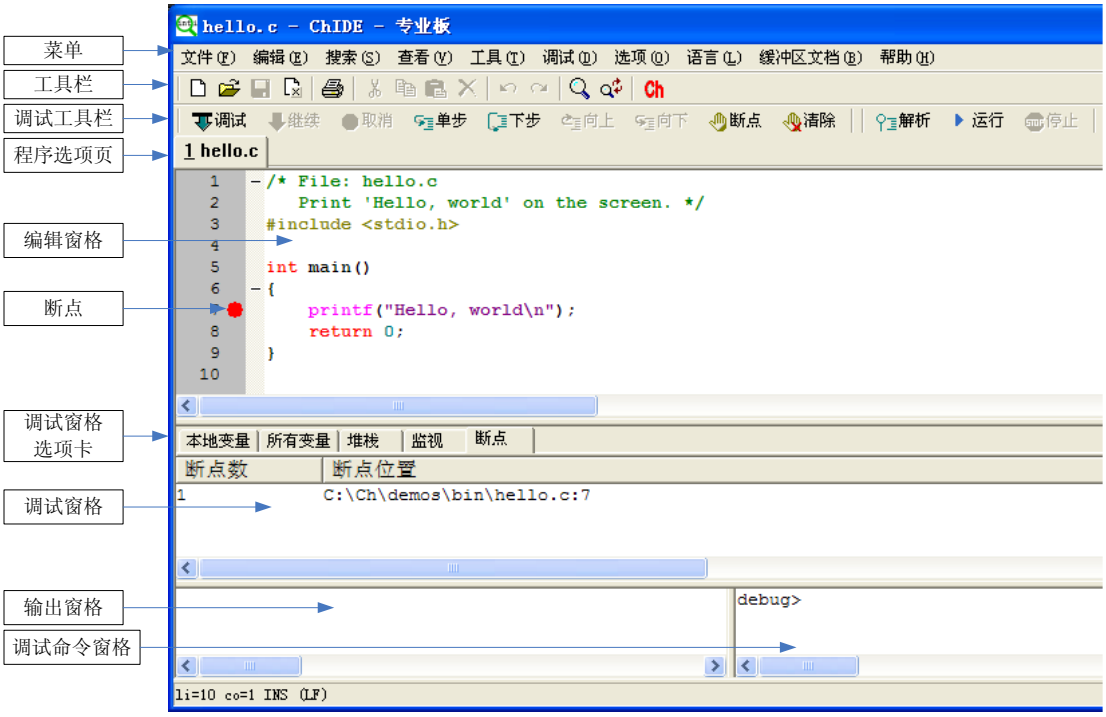


图 6 与 ChIDE 布局相关的术语

具有.c、.ch、.cpp、.cc和.cxx扩展名的C/Ch/C++程序，以及没有文件扩展名的C/Ch/C++程序都可以在ChIDE中运行。如图 7所示，点击工具栏中的“运行”按钮或菜单命令“工具|运行”都可以运行 hello.c程序，还可以用快捷键 F2来直接运行程序。详细的快捷键说明请参见第3.5节。

运行程序 hello.c 时，如果输出窗格原来不可见，运行后它将自动出现，并显示：

```
>ch -u "hello.c"
Hello, world
>Exit code: 0
```

如图 7所示，第一行

```
>ch -u "hello.c"
```

为蓝色字体，表示 ChIDE 调用命令 **ch** 来运行程序。第二行为黑色字体，为运行 Ch 程序 hello.c 输出的结果。最后的蓝色字体行表示程序运行结束并显示它的返回代码。返回代码为 0 表示程序成功运行，相应的语句为：

```
return 0;
```
或
```
exit (0);
```

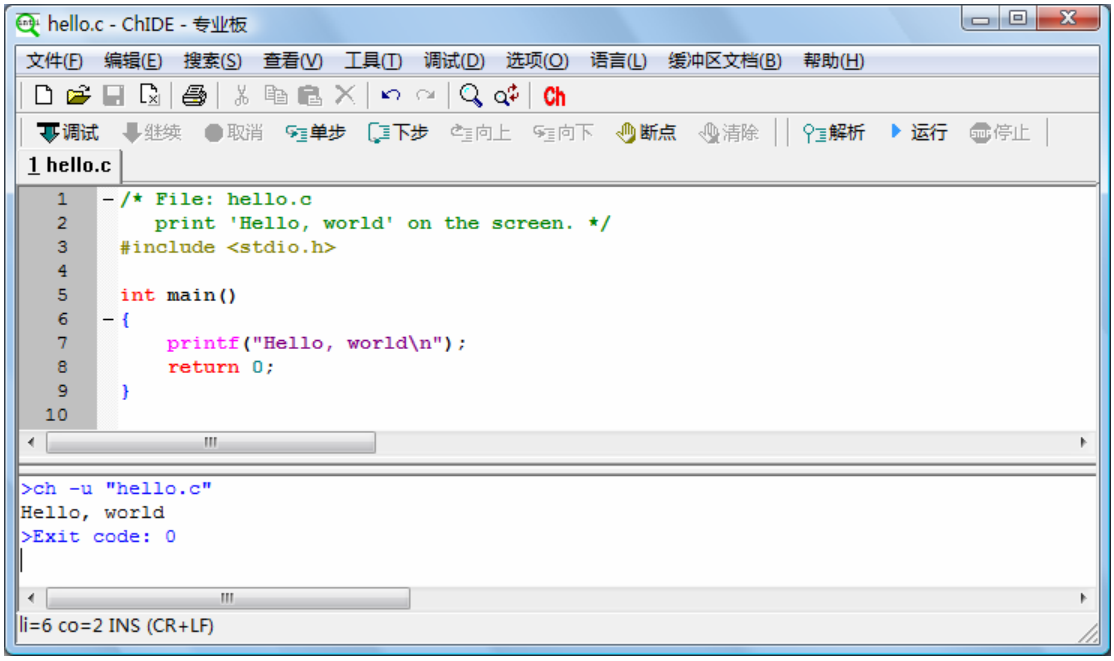


图 7 运行程序以及程序运行结果

返回代码不为 0 表示程序运行失败或被不正常终止，相应语句如下所示：

```
return -1;
```
或
```
exit(-1);
```

程序的返回代码为-1。

ChIDE 能够识别 Ch 产生的错误信息。为了说明这一点，我们可以添加一个错误到上述源程序中。把语句

```
printf("Hello, world\n");
```
变为
```
printf("Hello, world\n";
```

点击菜单命令“工具|运行”运行修改后的程序，输出结果应该如下：

```
ERROR: missing ')' before ';'
ERROR: syntax error before or at line 7 in file 'hello.c'
  ==>:    printf("Hello, world\n";
  BUG:    printf("Hello, world\n";<== ???
ERROR: cannot execute command 'hello.c'
```

如图 8所示，因为程序运行失败，所以输出窗格的最后一行显示返回代码：

```
Exit code: -1
```

如图 8所示，用鼠标左键双击输出窗格中的红色输出行，输出窗格中的错误信息和编辑窗格中

相应的错误行会用黄色背景突出显示，如图 9所示，光标会移到该错误行，如果出现滚动条，则输出窗格会自动滚动到适当位置以显示这一行。ChIDE能够识别出错误存在于哪个文件的第几行，所以ChIDE会自动打开存在错误的文件（例如头文件）。

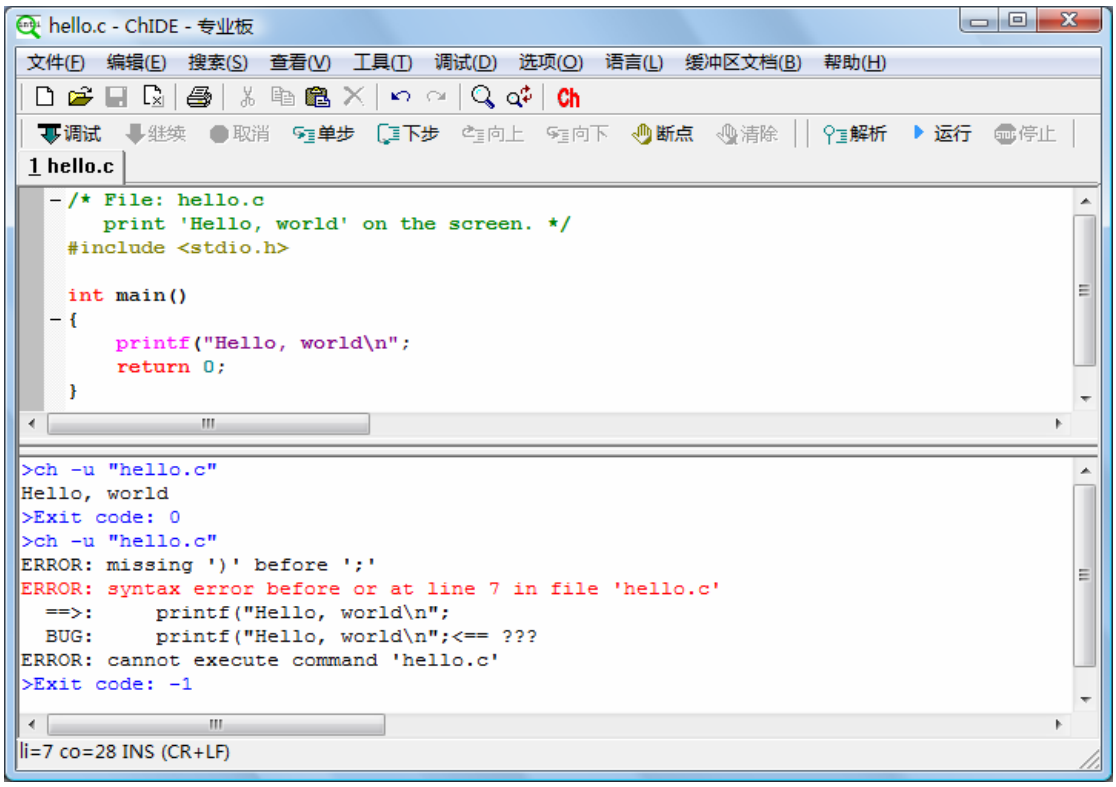


图 8 运行hello.c的错误行输出结果

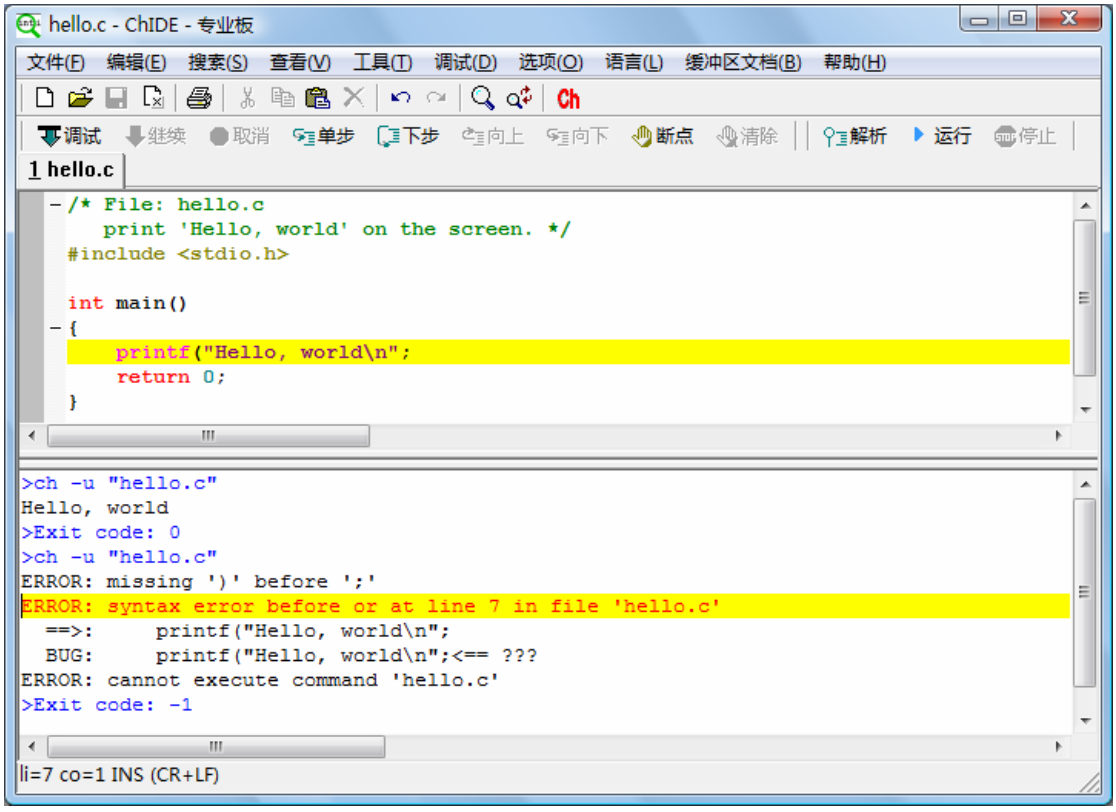


图 9 在执行程序hello.c时找到的错误行

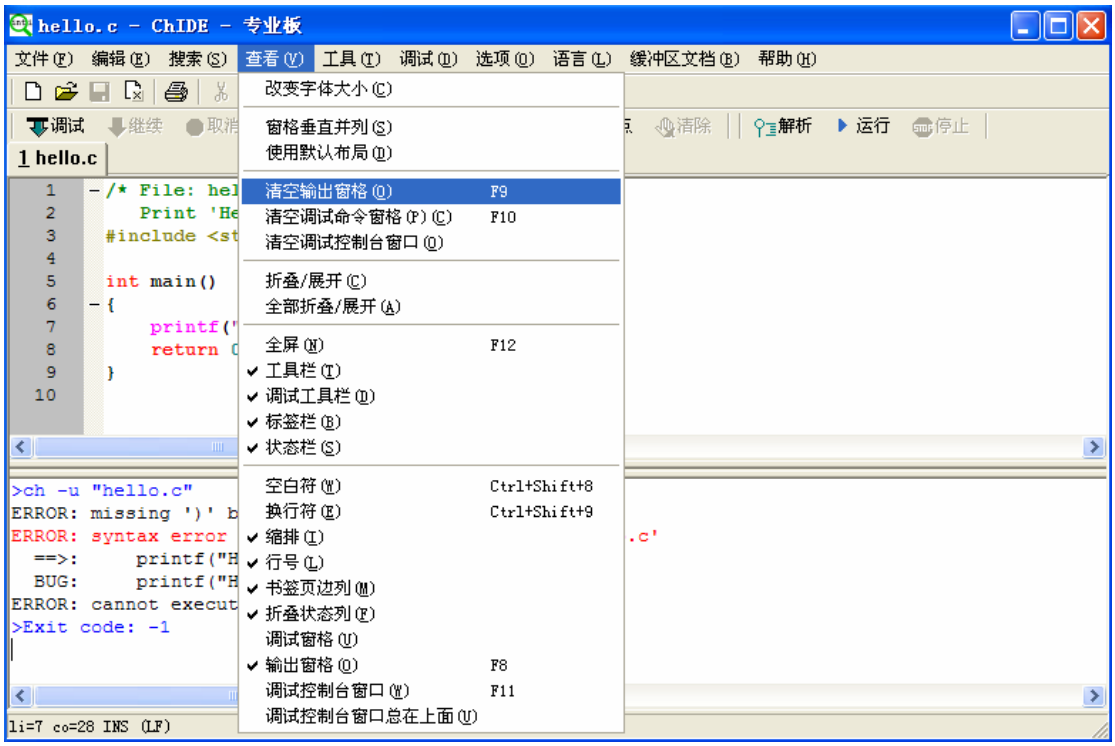


图 10 清空输出窗格内容

在这个简单例子中，我们很容易看出问题出在哪里。当文件很大时，点击菜单“工具|下一个出错信息”，或者用功能键F4可以查看每个错误。在执行“工具|下一个出错信息”命令之前，输出窗格中的第一条错误信息和编辑窗格中相应的错误行会用黄色背景突出显示。

使用菜单命令“工具|前一个出错信息”或者Shift+F4键，可以查看前一个错误信息。

如图 10所示，输出窗格可以通过菜单命令“查看|输出窗格”来打开或关闭。输出窗格中的内容可以通过菜单命令“查看|清空输出窗格”或者F9键来清除。

如果程序运行很长时间没有返回，可通过点击调试工具栏上的“停止”按钮或者点击菜单“工具|中止执行”来停止程序运行。

可以通过点击调试工具栏上的“解析”按钮或点击菜单“工具|解析”来检查程序中的语法错误，但并不运行程序。

2.3. 执行具有输入的 C/Ch/C++程序

在ChIDE中也可以运行需要用户输入数据的程序，如使用C函数**scanf()**得到用户输入的程序。我们来看一个例子，如图 11所示，我们加载程序scanf.c，这个程序在Windows平台下存放在C:/Ch/demos/bin/中，在Linux或Mac OS X下存放在/usr/local/ch/demos/bin/中。

如图 11所示，当执行这个程序时，将提示让用户输入一个数字，此时用户必须在输出窗格中输入一个数字，这时，输入窗格与输出窗格合二为一，用户输入的数字“56”及程序运行的输出结果均将在输出窗格中显示出来。

2.4. 执行具有绘图功能的 C/Ch/C++程序

执行具有绘图功能的C/Ch/C++程序与执行其它程序的方法是一样的，这里我们用一个例子来演示其执行过程。

在ChIDE中输入如图 12所示的程序，或者直接调入程序C:/Ch/demos/bin/fplotxy.cpp，当这

个程序执行时，它将绘制出如图 13所示的图形。绘图函数**fplotxy()**在Ch或SoftIntegration C++绘图库（SIGL）中可以找到其定义。程序使用绘图函数**fplotxy()**绘出数学函数func()在区间［0，360］上的图形，绘图时在区间［0，360］中等间隔取37个采样点。

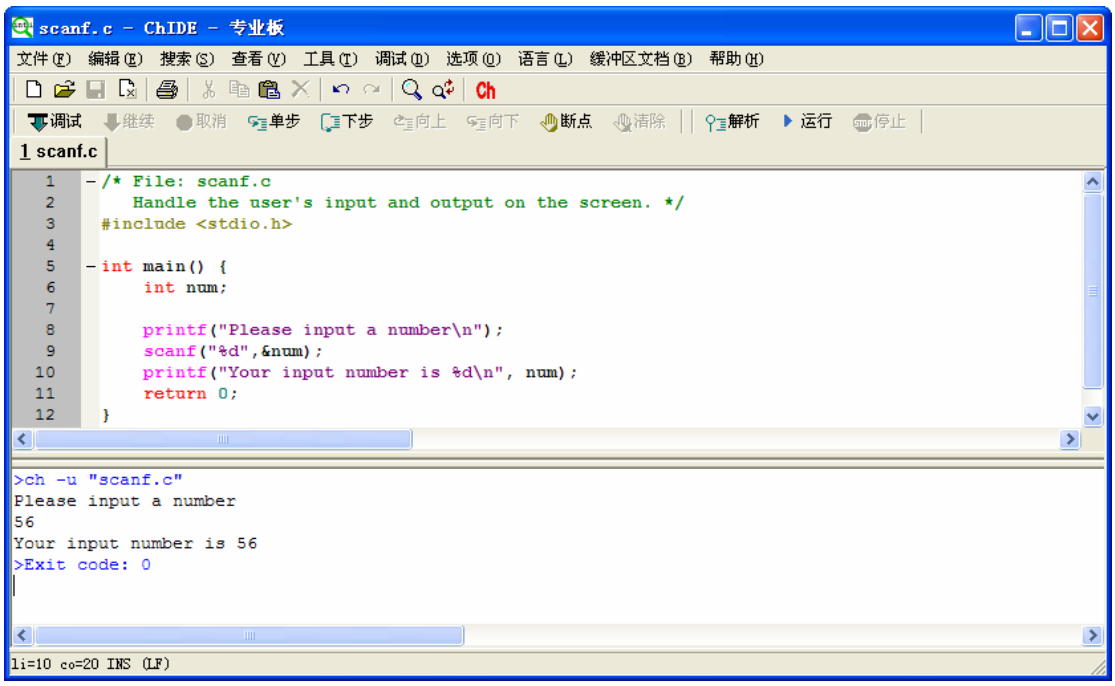


图 11 执行具有输入输出的程序

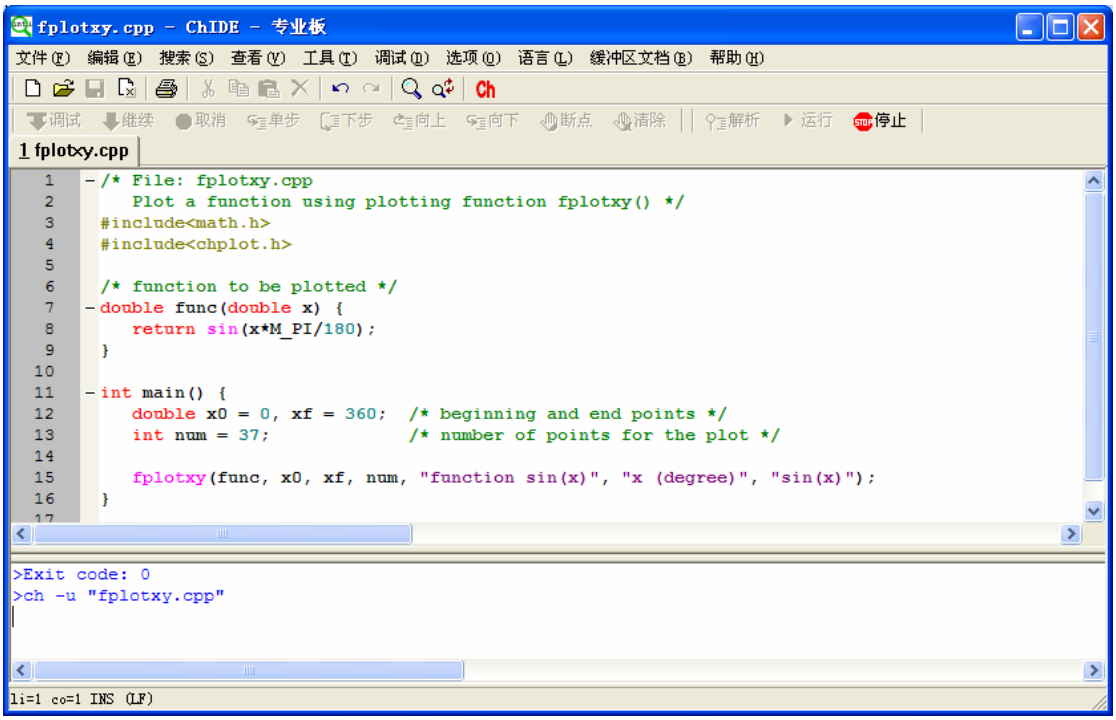


图 12 调用绘图函数**fplotxy()**程序

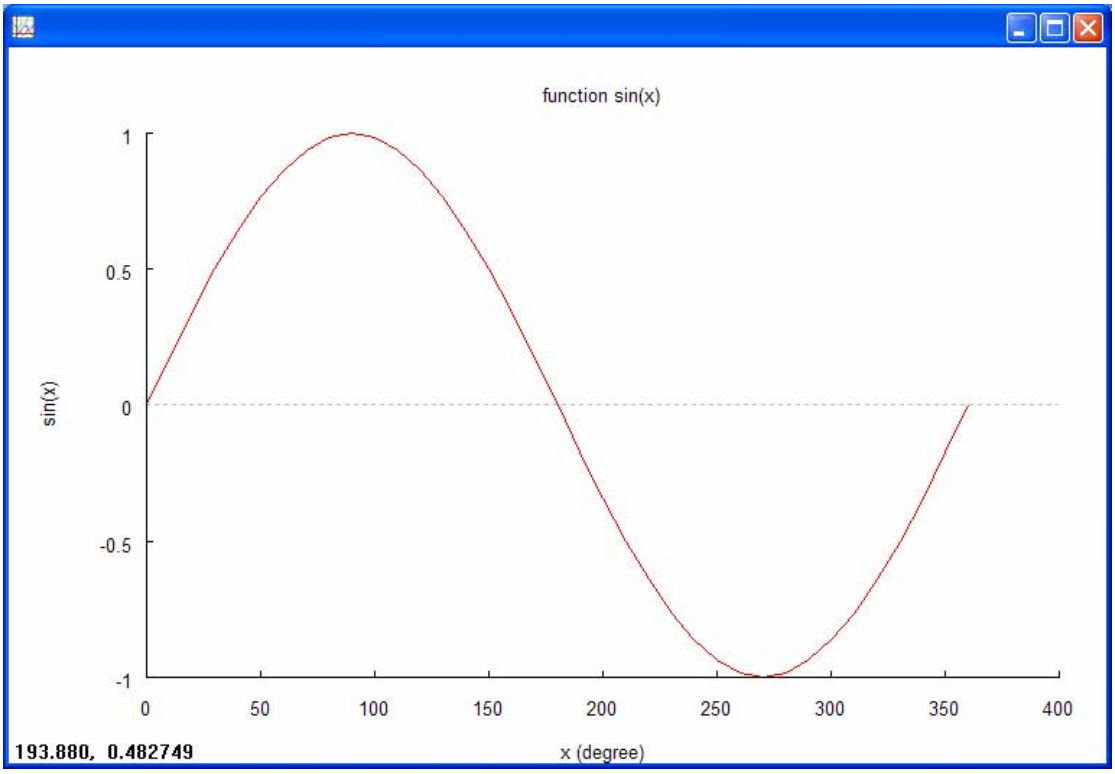


图 13 图 12所示程序的输出图形

具有绘图功能的程序需要包含头文件chplot.h，如果要编译，程序需有扩展名.cpp,编译时会将其按C++程序处理，并且链接SIGL C++绘图库。怎样用C++编译器来编译C++程序将在第7部分详细说明。

在CHHOME/demos/bin和CHHOME/demos/lib/libch/plot目录下存放有很多个展示Ch绘图能力及用法的程序。例如，程序C:/Ch/demos/bin/plotxy.cpp用来展示如何调用绘图函数**plotxy()**绘制存放在数组中的数据的图形，执行这个程序时，它将绘制出如图 13所示的图形。程序C:/Ch/demos/bin/fplotxyz.cpp用来展示如何调用绘图函数**plotxyz()**绘制函数z=cos(x) sin(y)的图形,函数z的定义域为：x∈[-3,3],y∈[-4,4],绘制图形时，在变量x、y的定义域内分别等间隔取了80个采样点。程序C:/Ch/demos/bin/legend.cpp用来展示如何在具有多条曲线的图形中增加图例。

2.5. 执行带有命令行参数的 C/Ch/C++程序

在ChIDE中可以执行带有命令行参数的程序。若要设定命令行参数，可以点击菜单“工具丨命令行参数”打开命令行参数无模式对话框，其中你可以看到已经设定的命令行参数，并且也可以修改这些参数。命令行参数对话框打开时，ChIDE中的快捷键仍然可以使用，所以可以很方便地以不同命令行参数运行同一个程序。另外一种方法也可以达到同样的效果，即在输出窗格中输入要执行的程序并在其前面加上一个“*”号，如下所示，

```
* C:/Ch/demos/bin/commandarg.c
* "C:/Ch/demos/bin/commandarg.c"
```

也同样会弹出命令行参数无模式对话框。若命令行参数无模式对话框已经弹出，则ChIDE会忽略“*”号。在输出窗格输入并执行命令的详细方法将在第6部分详细介绍。

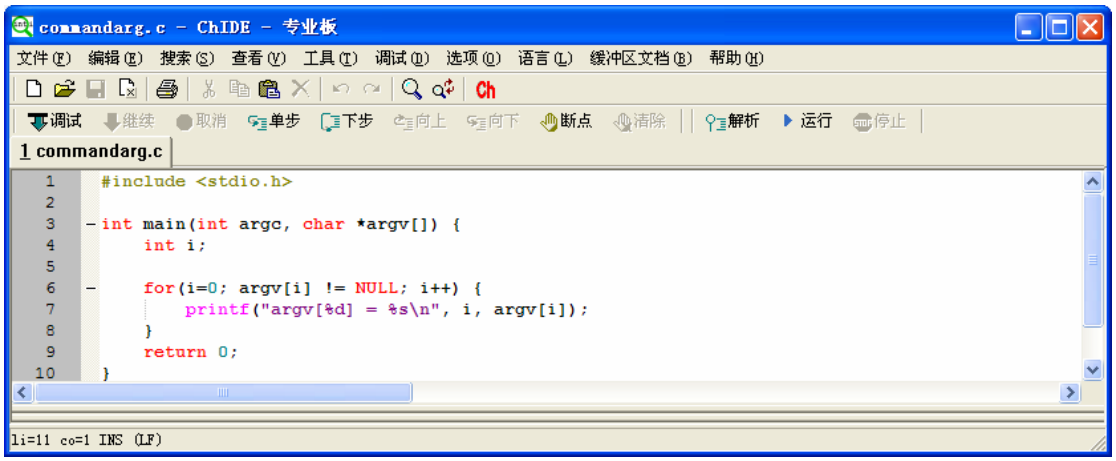


图 14 带有命令行参数的程序

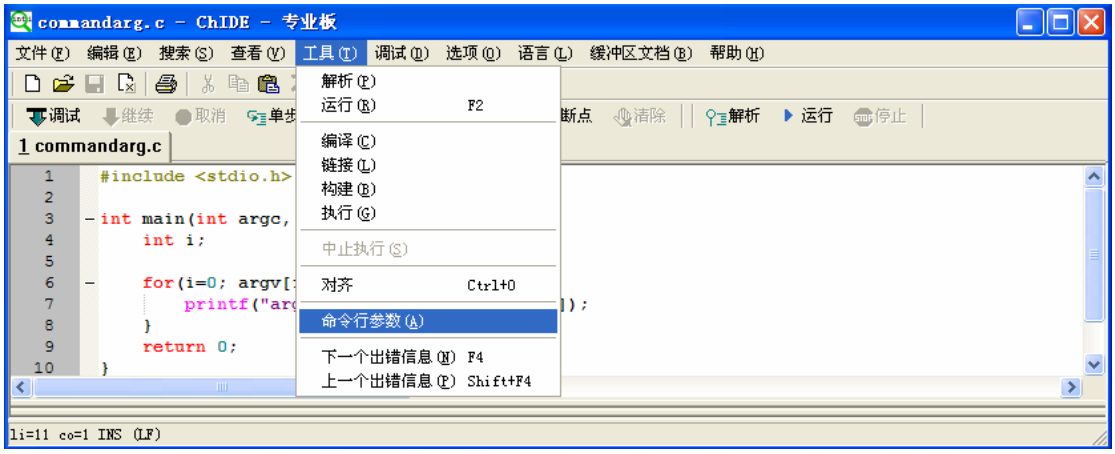


图 15 打开命令行参数无模式对话框

如图 14所示，该程序将接收命令行参数并将其打印出来。如图 15所示，点击菜单“工具 | 命令行参数”，ChIDE弹出命令行参数无模式对话框；图 16演示怎样设置命令行参数；程序执行后的输出见图 17所示。

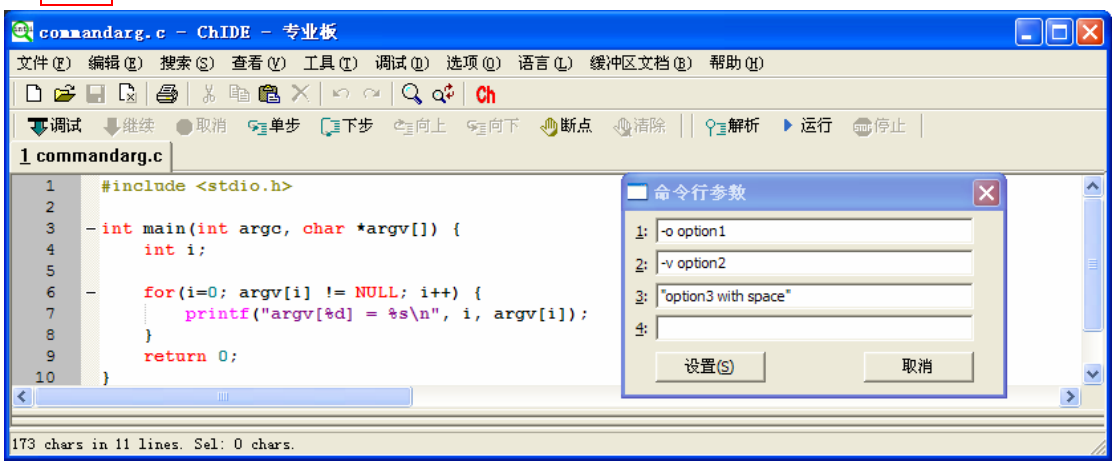


图 16 设置命令行参数

2.6. 对齐 C/Ch/C++程序

考虑到程序的可读性以及软件的可维护性，程序中的每一行都应该采用适当的对齐方式。这对有多重嵌套循环和多个选择结构的程序尤其重要。通过菜单命令“工具|对齐”可以正确调整编辑窗格中程序的对齐方式。也可以右键单击文件名所在标签（位于调试工具栏下面），选择右键菜单

中的“对齐”命令来调整程序的对齐方式。如图 4所示，右键单击文件名所在标签之后可找到“对齐”命令。

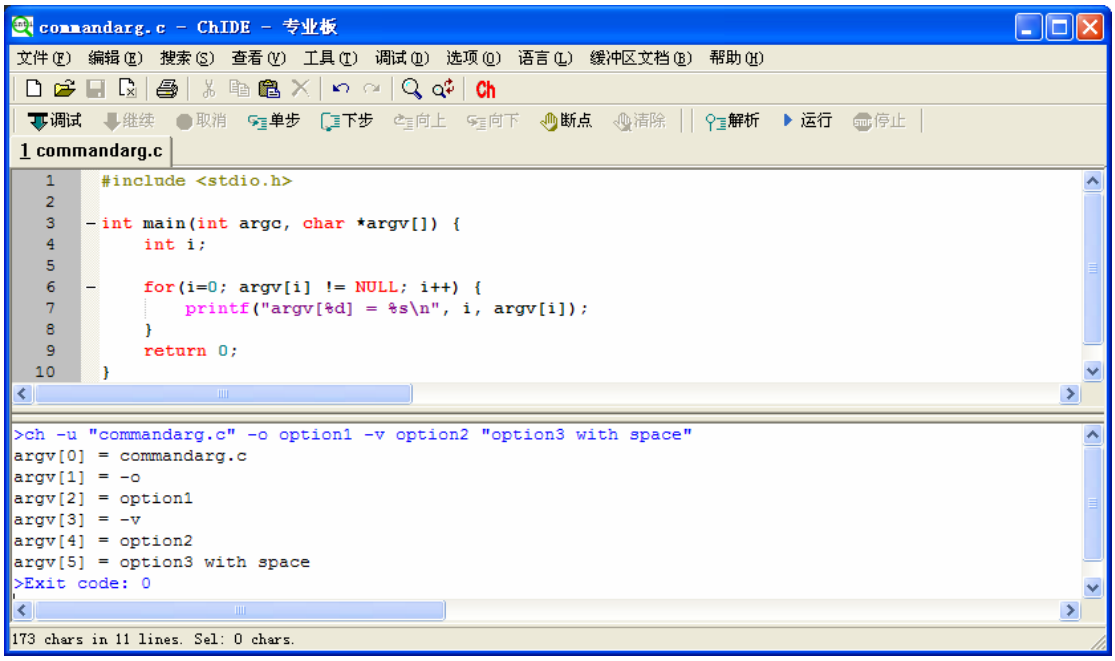


图 17 带有命令行参数的程序的运行结果

3. ChIDE 的编辑功能

ChIDE的文本编辑功能与文字处理器（如Microsoft Word或Notepad）的文本编辑功能相差不多。在编辑窗格中编辑程序时，可以使用工具栏中的编辑类按钮和菜单“编辑”下菜单命令来完成编辑工作。本节介绍ChIDE的编辑功能。

3.1. 编辑功能

在Windows操作系统中，如图 18所示，在编辑窗格中单击鼠标右键，ChIDE会弹出经常使用的编辑命令菜单。

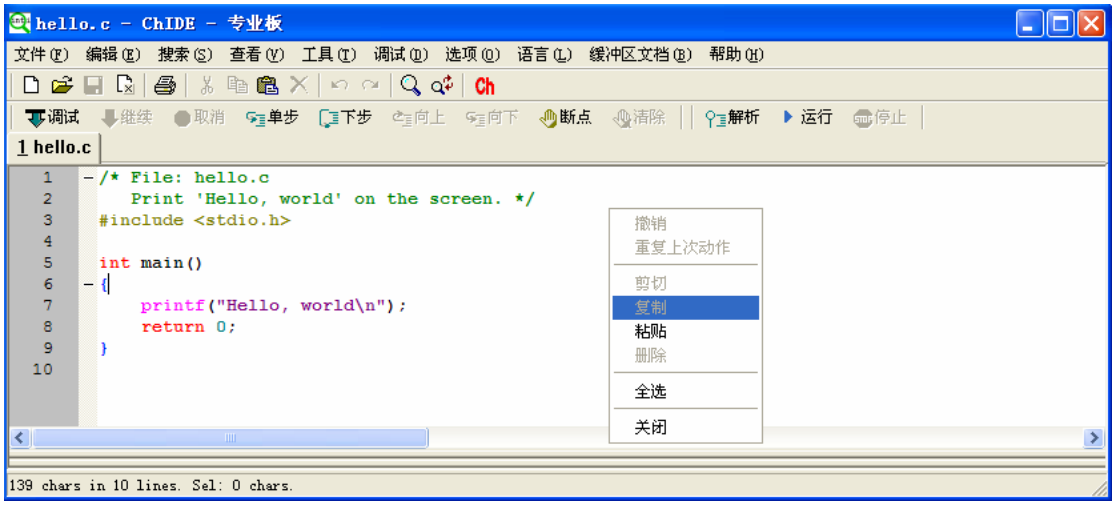


图 18 编辑窗格中的右键菜单

当用户在输入程序时，若正在输入的字符串与前面输入的某个单词匹配时，则匹配的单词会显

示出来，这时用户可以按回车键来直接输入这个匹配的单词。用户在输入时也可以按“Ctrl+Enter”键来调出所有与当前输入相匹配的单词，然后用箭头键来选择需要的单词，再按回车键来直接输入这个匹配的单词。

ChIDE支持矩形区域文字选中，在widnows操作系统下的操作方法为按下Alt键，再按住鼠标左键移动鼠标；在Linux和Mac OS X下，则按下Ctrl键，再按住鼠标左键移动鼠标。

使用快捷键及缩略语功能可以加快编辑速度。第3.5节中的表 2列出了许多用于快速编辑的功能键定义。缩略语的定义将在第3.6节中介绍。

3.2. 查找与替换

ChIDE的查找与替换功能支持全字符匹配、正则表达式（Regular expressions）、区分大小写、向前向后、循环查找等选项；使用“使用反斜线”选项，还可以查找或替换控制字符。替换时可以选择单个替换、在选取文字中替换或全部替换。当使用正则表达式功能进行替换时，在替换字符串中可以使用带有标记符号的子表达式。正则表达式替换功能不会进行跨行匹配。

3.3. 改变字体大小

为便于课堂教学，编辑窗格中文本字体的大小可以通过点击菜单命令“查看|改变字体大小”来改变。在课堂教学过程中，也可以通过如第3.5节 表 2所示的快捷键“Ctrl+Keypad+”、“Ctrl+Keypad-”和“Ctrl+Keypad/”来方便的将字体变大、变小、恢复正常的字体。注意由于在笔记本电脑中没有小键盘，若要想使用这些快捷键，得首先使用“Shift+NumLk”键将笔记本电脑键盘的“锁定数字”功能打开，然后再使用键盘中的相应的数字键，如使用快捷键“Ctrl+Keypad+”时，需要按下“Ctrl”键及右“Shift”左边的“+”键。

3.4. 折叠功能

ChIDE支持对C/Ch/C++及其它几种语言（第8部分中列出）的源程序中的程序块进行折叠。对于C/Ch/C++语言，折叠的位置是基于程序的对齐方式的；对于 其它几种语言，折叠的位置是基于大括号的。如第2.2节中的图 2与图 5所示，可以点击折叠位置上的折叠标记来折叠或展开程序块。在折叠状态列中使用快捷键“Ctrl+Shift+Click”将展开或折叠所有最顶层的折叠程序块；在某一个折叠位置使用快捷键“Ctrl+Click”将展开或折叠这个折叠块中的所有折叠位置；在某一个折叠位置使用快捷键“Shift +Click”将展开这个折叠块中的所有折叠位置；

3.5. 快捷键

ChIDE中的快捷键的定义与通常我们在Windows和GTK+中大部分是相同的。当按下“Shift”键时，同时按下能够移动光标的键（上、下、左、右箭头键、Page Up/Down、Home 、End）将增加或减少选中文字的多少；按下“Alt”和“Shift”键，再按住鼠标左键移动鼠标将选中矩形区域。一些国家使用的键盘可能有些键不存在；在一些系统中，某些键已经被系统占用，如GTK+中的窗口管理器（Window Manager）占用了一些键。每一个快捷键均有对应的菜单命令。

表 1列出了常用的命令及其快捷键。

表 2列出了一些不常用的命令及其快捷键，这些命令没有与之对应的菜单命令。

默认情况下，功能键F9、F10、F11和F12在Mac OS X操作系统下是被禁用的。要想在ChIDE使用这些快捷键，可以使用以下步骤将其解禁：

- 点击左上角的“苹果标志”（Apple symbol）；
- 点击“系统首选项”（System Preferences）；

- 点击“键盘与鼠标”；
- 点击“键盘快捷键”；
- 解除禁止使用功能键F9、F10、F11和F12。

表 1　ChIDE中的常用命令及其快捷键.

命令	快捷键
帮助	F1
运行C/Ch/C++程序	F2
查找下一个	F3
查找上一个	Shift+F3
下一个出错信息	F4
上一个出错信息	Shift+F4
调试（从头开始调试）	F5
单步（进入语句）	F6
单步（越过语句）	F7
打开/关闭输出窗格	F8
清空输出窗格	F9
清空调试命令窗格	F10
打开或关闭调试控制台窗口	F11
全屏	F12

表 2　ChIDE中的不常用命令及其快捷键

命令	快捷键
显示字体变大	Ctrl+Keypad+
显示字体变小	Ctrl+Keypad-
恢复正常字体	Ctrl+Keypad/
切换打开的文件	Ctrl+Tab
缩进	Tab
提升	Shift+Tab
从后往前删除整个单词	Ctrl+BackSpace
从前往后删除整个单词	Ctrl+Delete
从后往前删除到行首	Ctrl+Shift+BackSpace
从前往后删除到行尾	Ctrl+Shift+Delete
光标到程序的开始位置	Ctrl+Home
光标到程序的结束位置	Ctrl+End
从当前位置到程序的开始位置全部选中	Ctrl+Shift+Home
从当前位置到结束位置全部选中	Ctrl+Shift+End
光标移至行首	Alt+Home
光标移至行尾	Alt+End
从当前位置到行首全部选中	Alt+Shift+Home
从当前位置到行尾全部选中	Alt+Shift+End
折叠或展开一个折叠块	Ctrl+Keypad*
创建或删除一个书签	Ctrl+F2

选取下一个书签	Alt+F2
查找选择	Ctrl+F3
向后查找选择	Ctrl+Shift+F3
上滚	Ctrl+Up
下滚	Ctrl+Down
剪切当前行	Ctrl+L
拷贝当前行	Ctrl+Shift+T
删除当前行	Ctrl+Shift+L
当前行与前一行交换	Ctrl+T
在当前位置复制当前行	Ctrl+D
查找匹配预处理条件，跳过嵌套条件	Ctrl+K
选到匹配预处理条件	Ctrl+Shift+K
向后查找匹配预处理条件，跳过嵌套条件	Ctrl+J
向后选到匹配预处理条件前段	Ctrl+Shift+J
光标移至前一段	Ctrl+[
光标移至前一段首并选中当前位置至前一段首的内容	Ctrl+ Shift+ [
光标移至后一段尾	Ctrl+]
光标移至后一段尾并选中当前位置至后一段尾的内容	Ctrl+ Shift+]
光标移至前一单词	Ctrl+Left
光标移至前一单词并选中当前位置至前一单词的内容	Ctrl+ Shift+Left
光标移至后一单词	Ctrl+Right
光标移至后一单词并选中当前位置至后一单词的内容	Ctrl+ Shift+Right
光标移至当前单词前面	Ctrl+/
光标移至当前单词前面并选中当前单词的前半部分	Ctrl+ Shift+/
光标移至当前单词后面	Ctrl+\
光标移至当前单词后面并选中当前单词的后半部分	Ctrl+ Shift+\

3.6. 缩略语功能

为了加快文字编辑与编程，ChIDE中加入缩略语功能，就是系统定义的一些缩写词，让其代表预先定义的一段文本。使用一个缩略语时，先输入它，然后使用菜单命令“编辑丨展开缩略语”或使用快捷Ctrl+B来展开缩略语，这个缩略语就被一个在速写文件中定义的文本所代替。速写文件文件有两个，一个是全局的，一个是用户自定义的。全局速写文件文件可以通过菜单命令“选项丨打开ChIDE全局速写文件”打开，用户可以在其中添加自己定义的缩略语。用户自定义的速写文件可以通过菜单命令“选项丨打开ChIDE用户速写文件”来打开。

速写文件中包含多行如下形式的缩略语定义行：

缩略语名字= 扩展文本

缩略语名字可以由任意字符组成（除了某些控制字符，如回车（CR）和换行（LF）)， 包括重音字符（Accented characters）及多字节字符，如汉字亚洲语系字符。

缩略语名字有如下限制：不能以 # 或空格 或制表键开头，但其内部可有空格；不能包含“=”；不能超过32字符，当然32个字符也足够了。

缩略语的扩展文本可以包含换行符“\n”。在扩展文本中，用字符“|”表示插入扩展文本后的光标位置。若在扩展文本中包含字符“丨”，请用字符“||”来代替。

当使用缩略语扩展时，缩略语名称不必同前面的文本分离，你可以在一个单词内直接扩展它。

如果一个名称是另一个名称的后半部分，那么只有较短的会被扩展。例如定义“ring”和“gathering”，则只有“ring”部分会被扩展。

使用缩略语功能可以加快程序输入与调整对齐方式，表 3列出了预先定义的全局缩略语。

在默认的用户自定义速写文件中包含了一个名字为“hw”的缩略语。如图 19所示，如果你键入缩略语“hw”，再按下快捷键“Ctrl+B”，则会扩展出课后作业的说明部分。你可以使用菜单命令“选项丨打开ChIDE用户速写文件”打开用户自定义速写文件，并且修改缩略语“hw”，加入你自己的名字及与你开发的类、工程等相关信息。默认的用户自定义速写文件中还包括中文缩略语“作业”。

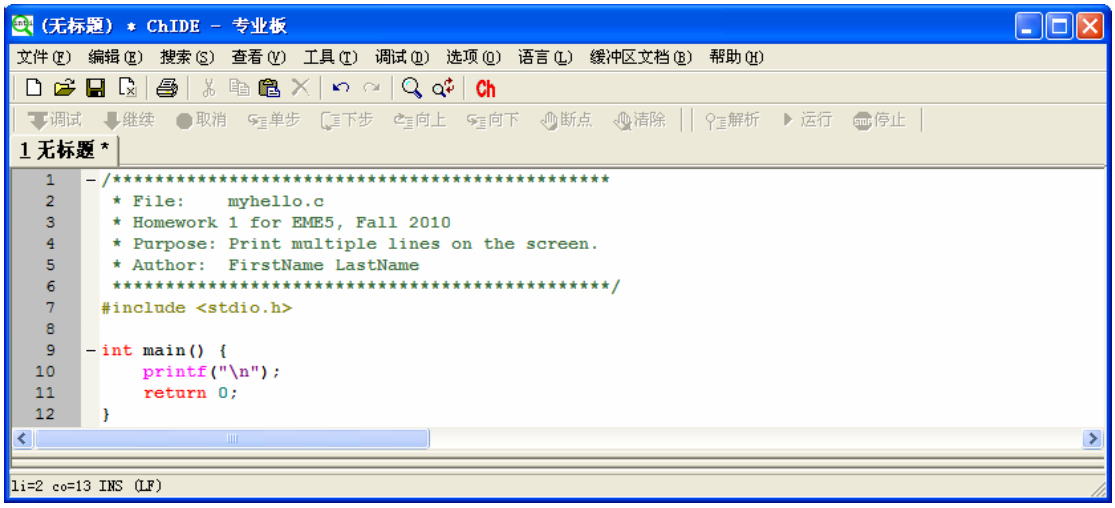


图 19 缩略语“hw”的扩展文本

表 3 默认的全局缩略语定义表

缩略语	扩展文本	缩略语	扩展文本
com	/* | */	fenv.h	include fenv.h
inc	#include < | >	float.h	include float.h
myinc	#include " | "	inttypes.h	include inttypes.h
def	#define |	iso646.h	include iso646.h
main	function main()	limits.h	include limits.h
mainarg	function main() with arguments	locale.h	include locale.h
if	if statement	math.h	include math.h
elseif	else if statement	setjmp.h	include setjmp.h
else	else statement	signal.h	include stdarg.h
for	for loop	stdarg.h	include stdarg.h
while	while loop	stdbool.h	include stdbool.h
do	do-while loop	stddef.h	include stddef.h
switch	switch statement	stdint.h	include stdint.h
foreach	foreach loop	stdio.h	include stdio.h
a	[|] for an array index	stdlib.h	include stdlib.h
c	’ | ’ for a character	string.h	include string.h
s	” | ” for a string	tgmath.h	include tgmath.h
p	(|) for parentheses	time.h	include time.h

pi	M PI|	wchar.h	include wchar.h
epsilon	FLT EPSILON|	wctype.h	include wctype.h
cond	| ? : for conditional operator	chdl.h	include chdl.h
sizeof	sizeof(|)	chplot.h	include chplot.h
struct	struct structure	chshell.h	include chshell.h
union	union structure	numeric.h i	nclude numeric.h
enum	enum structure	func	a function definition
class	class structure	prot	|(); for a function prototype
stdlib.h	include stdlib.h	call	|(); for calling a function
time.h	include time.h	printf	printf("|\n");
assert.h	include assert.h	scanf	scanf("|", &);
complex.h	include complex.h	sin	sin(|)
ctype.h	include ctype.h	a standard C function name	call the standard C function
errno.h	include errno.h		

3.7. 缓冲区功能

在默认情况下，ChIDE有20个缓冲区，每个缓冲区包含一个文件。缓冲区的数量可以在用户选项文件中进行设定。在使用缓冲区文档菜单在缓冲区文档之间切换时，可以直接选择缓冲区文档的名字或使用菜单命令“缓冲区文档丨上一个文档”、“缓冲区文档丨下一个文档”；也可以使用快捷键Ctrl+Tab（如第3.5节表 2所示）在缓冲区文档间循环切换。

当所有缓冲区都包含文件，这时你又要打开一个新文件，则必须保存某一个文件才可重用该文件所用的缓冲区，这种情况下系统会提示你保存该文件。

3.8. 文件列表功能

一个文件列表就是许多个文件的集合。为了以后能快速地加载这些文件，你可以把当前所有缓冲区文件作为一个文件列表来保存。文件列表文件是以“.session”为扩展名无格式文本文件。

可以用菜单命令“文件|载入文件列表”和“文件|保存当前文件为文件列表”来保存或加载文件列表。

当关闭ChIDE时，当前所有缓冲区内的文件会被自动保存到一个文件列表中；当你再次打开ChIDE时，上次保存的文件列表会自动被加载入缓冲区中。

4. 在 ChIDE 中调试 C/Ch/C++程序

ChIDE具有典型的二进制 C 程序调试器的所有功能，其调试命令如图 20所示。调试菜单包括“开始”、“单步”等命令，这些命令在调试工具栏中也有相应的按钮。调试程序时，调试工具栏中可以使用的按钮是可以点击的，对当前调试状态不适应的按钮是被灰化的。

4.1. 在调试模式下运行程序

点击调试工具栏中的按钮“调试”或按功能键“F5”，用户可以在调试模式下运行编辑窗格中的程序，当遇到断点时，程序会停止运行。用户可以通过命令“单步”或“下步”来逐行运行程序。

调试程序时，若执行“单步”命令或按F6键，调试进入函数；若执行命令“下步”或按F7键，调试不进入函数，执行当前函数后停在该函数的下面一行；“继续”命令会使程序连续运行直到遇到断点或程序结束时停止，关于断点的详细说明请参见第4.3节。

如果一个程序运行失败或长时间没有返回，可以点击“取消”按钮终止该程序的运行。

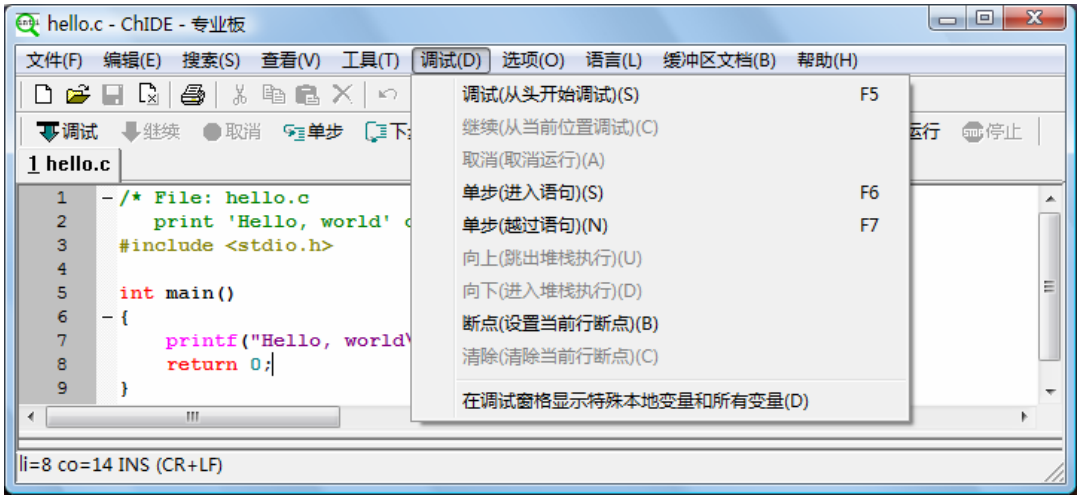


图 20 调试菜单

4.2. 调试模式下的输入输出窗口 --- 调试控制台窗口

在调试状态下运行程序时，标准输入、输出和错误流被重定向到独立的调试控制台窗口，如图21所示。在默认情况下，调试控制台窗口总是位于其它窗口之上。可以通过点击菜单“查看丨调试控制台窗口总在上面”来打开或关闭此设置。可通过点击菜单“查看丨调试控制台窗口”来打开或关闭调试控制台窗口。如图 10所示，调试控制台窗口中的内容可以通过点击菜单“查看丨清空调试控制台窗口”来清空。调试控制台窗口的背景颜色、文本颜色以及窗口大小和字体大小都可以通过右击窗口左上角的ChIDE图标、选取右键菜单中的“属性”项来改变相应的设置。需要特别说明的是，在Windows Vista系统中，用户需要具有管理者权限才能完成上述操作。

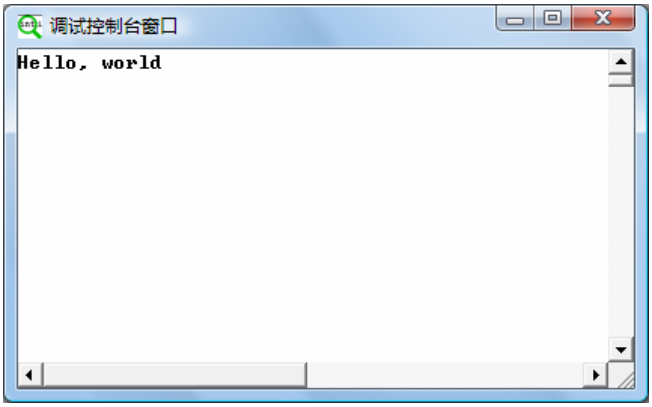


图 21 用于输入/输出的调试控制台窗口

4.3. 设定或清除断点

在调试运行程序前或调试程序时，均可以添加新的断点。如图 6所示，可以通过点击程序中某一行左边的空白处在本行增加一个断点；如果要清除断点，可以点击某一行左边空白处高亮显示的红色标记；可以通过点击调试窗格顶部的“断点”选项卡来查看设定了断点的语句，这时调试窗格中会显示断点的数量及每个断点的位置。可以通过点击调试工具栏上的“断点”按钮在当前光标所

在行设定断点，也可以通过点击调试工具栏上的“清除”按钮清除该断点。若程序中没有设定断点，则调试工具栏上的“清除”将不可用。不能在变量声明语句上设置断点，但可以在如下带有变量初始化的变量声明语句上设置断点：

```
int i = 10;
```

在程序运行或调试时不能编辑程序，否则，会有一条警告语句出现：“调试期间对文件所作改动不能立即生效”。程序运行结束后可对其进行编辑。当删除或添加程序中的代码时，程序中的断点也会自动调整。

程序的断点也可以设置在函数调用语句及控制变量上。关于怎样在调试命令窗格中使用调试命令来调试程序，我们将在第4.6节详细介绍。

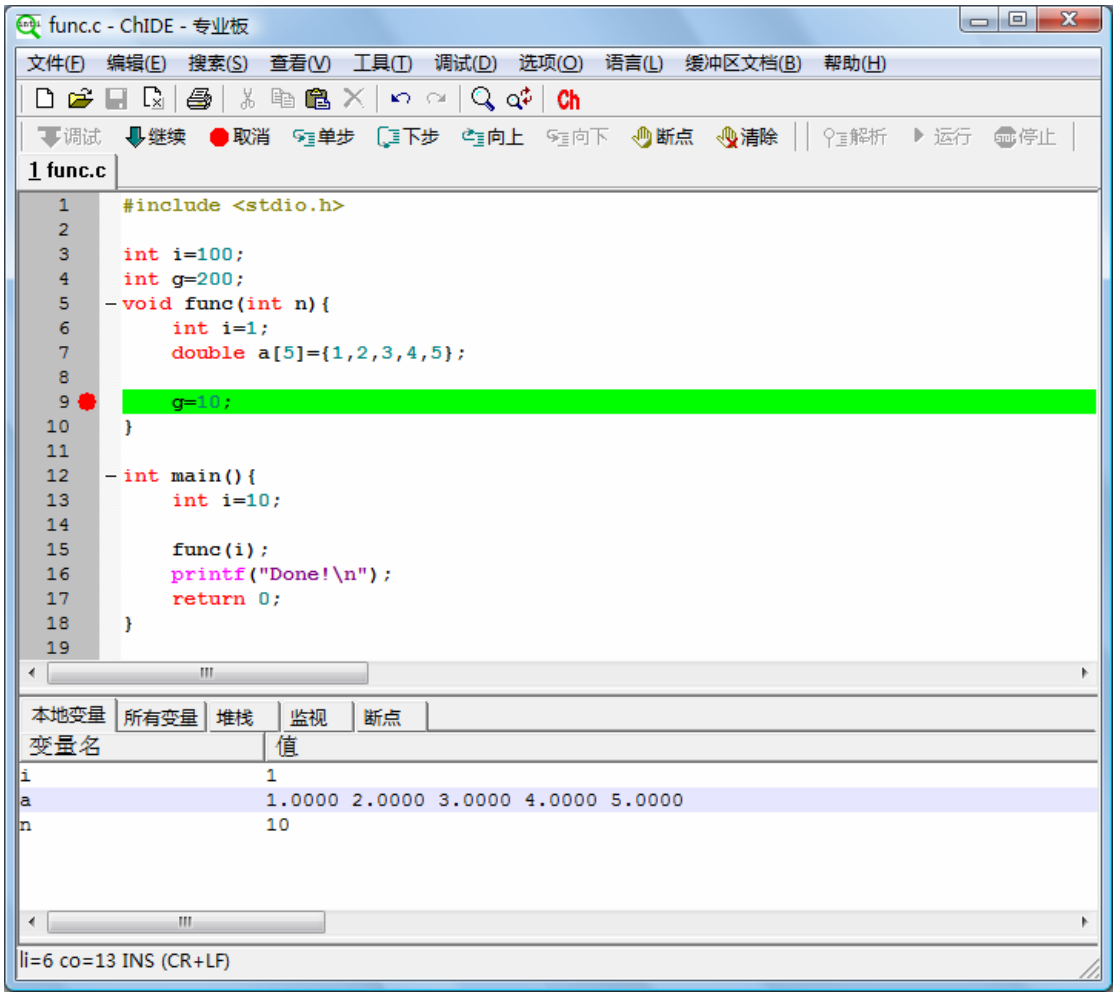


图 22 显示被调用函数中本地变量的名称和值

4.4. 在调试窗格中监视局部变量

在调试程序时，使用“单步”命令可以进入函数内部进行调试。如果通过“单步”调试命令进入的函数所在的文件还不在ChIDE的缓冲区中，那么包含此函数的文件将会被加载到缓冲区中，当这个函数执行结束后，前述被加载到缓冲区中的文件将会被清除，但是，如果在该文件中设置了断点，则它将会保留在缓冲区中，不管在程序执行中还是程序执行已经结束。

在点击调试工具栏上“单步”或“下步”按钮调试运行程序时，可以通过点击调试窗格顶部的“本地变量”选项卡来查看当前堆栈中变量的名称和值。当被调试程序运行到某个函数中时，点击

调试窗格顶部的“本地变量”选项卡可以查看该函数中局部变量和函数的参变量的值；如果脚本程序没有函数**main()**，“本地变量”选项卡命令显示程序中全局变量的值。如图 22所示，当程序func.c（存放在目录CHHOME/demos/bin中）运行到第9行时，本地整数变量i和n分别为1和10，**double** 型数组a各个元素的值分别为1、2、3、4和5。

4.5. 在调试窗格中监视不同堆栈中的变量

在调试中，用户可以在调试窗格中切换显示不同的函数堆栈中变量的值。可以通过点击调试工具栏上的“向上”命令按钮或“向下”命令按钮来在调试窗格中显示调用函数或被调用函数作用范围内的变量，或在调试命令窗格中读取或设定这些变量的值。使用不同颜色来高亮显示当前行和调用函数的调用行。例如：点击图 22所示调试工具栏上的“向上”命令按钮，程序的控制流移动到调用函数**main()**的第15行，如图 23用兰色高亮显示的那一行。在图 22所示窗口中，“向下”命令不能点击；因为我们将当前堆栈向上移动一层，所以在图 23 中，“向下”命令可以点击。此时的调试窗格显示调用函数 **main()**中的变量i及其值。

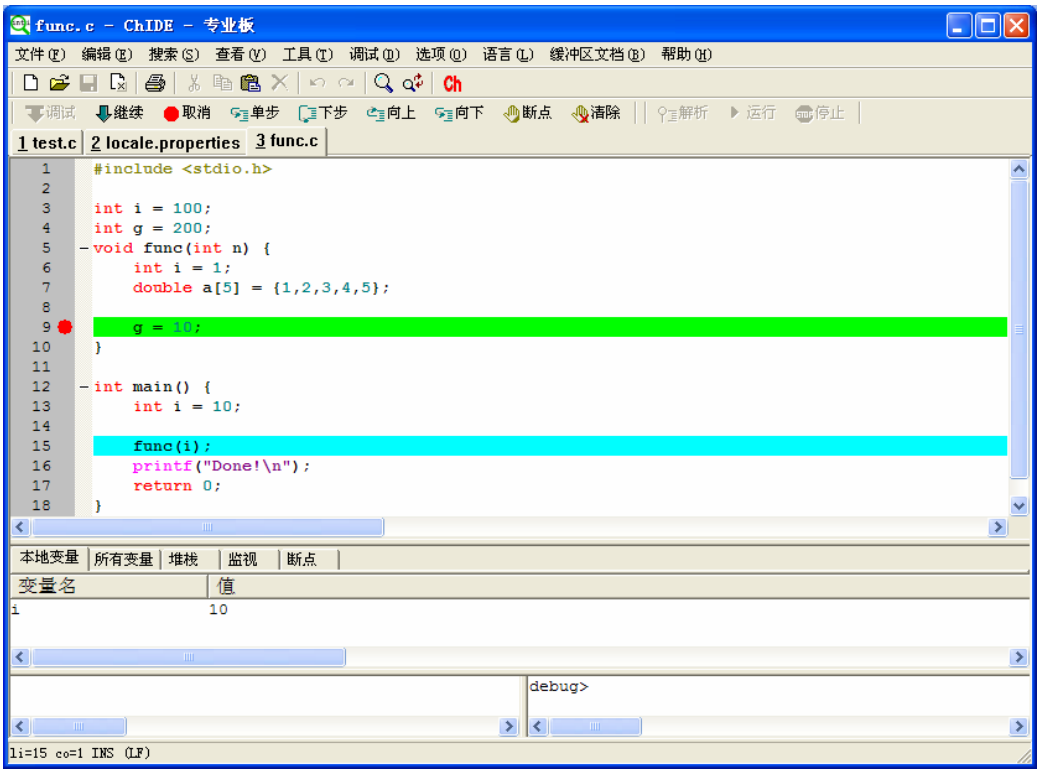


图 23 显示调用函数中的本地变量的名称和值

点击调试窗格顶部的“堆栈”选项卡，在调试窗格中可显示函数、成员函数、程序名称和相应的堆栈级别。当前运行的函数堆栈级别为0，堆栈级别为n+1的函数调用堆栈级别为n的函数。例如，如图 24所示，函数func()被函数**main()**调用，**main()**被程序func.c调用。

如图 25所示，可以使用调试窗格顶部的“所有变量”选项卡命令来显示所有堆栈中的变量名称和相应的值。如图 23所示，在调试窗格中，当前行和调用行用相应的高亮颜色显示，以标明不同的函数堆栈级别。在图 25中，程序停在第9行，调试窗格中显示了函数 func() 和 **main()**中的本地变量及全局变量的名称和值，我们可以看到，在运行第9行语句前，全局变量g的值为200。

执行图 20中所示的菜单命令“调试 | 在调试窗格显示特殊本地变量和所有变量”，调试窗格会在“本地变量”和“所有变量”选项卡中显示特殊变量的名称和值（如__func__）。

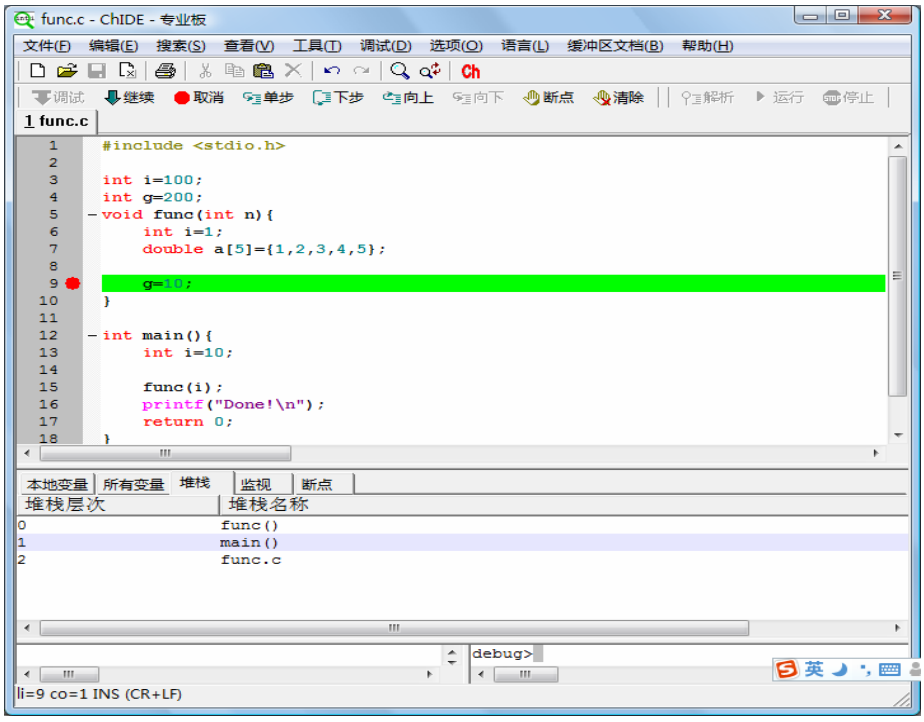


图 24 显示程序执行过程中的函数堆栈级别

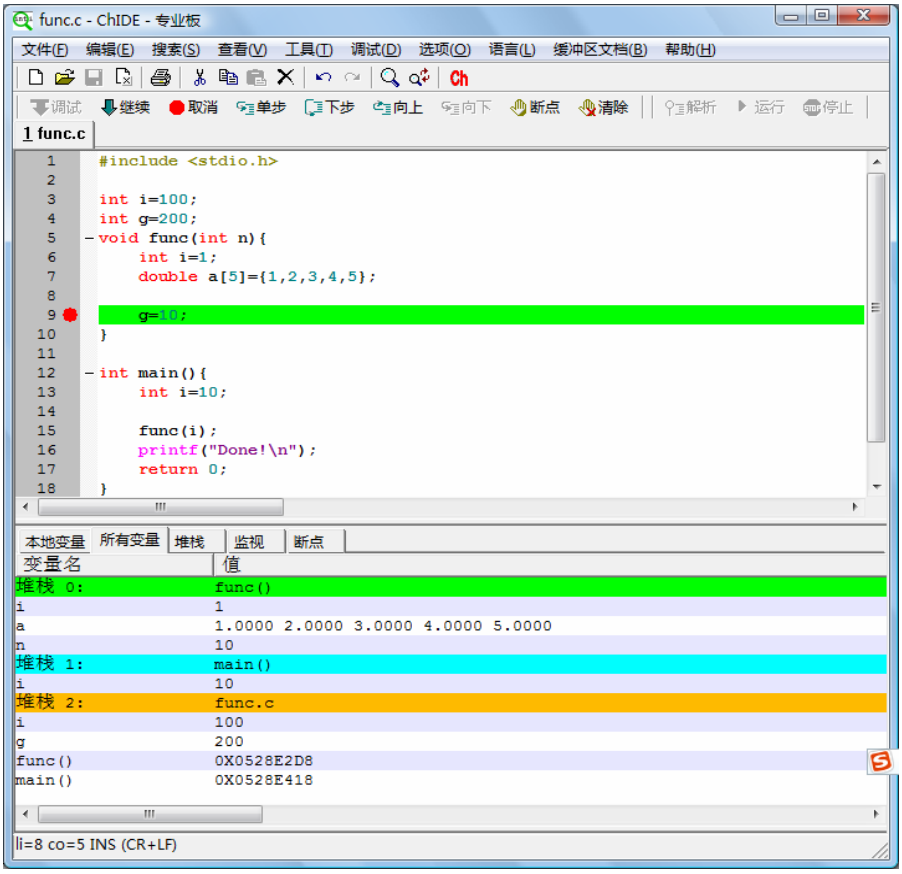


图 25 显示所有堆栈中的所有变量的名称和值

4.6. 在调试命令窗格中使用调试命令

在调试程序时，可以在调试命令窗格中使用许多调试命令对程序进行调试。调试命令窗格中的

如下提示符说明调试器已经准备好接收调试命令。

```
debug>
```

在调试命令窗格中，输入命令“help”，则会显示所有可用的调试命令，如图 26所示。每一行的冒号左边的内容是命令名称，冒号右边解释命令的作用。调试工具栏中的所有命令在调试命令窗格中都有相应的命令，但是，有些调试功能只有通过调试命令窗格才能实现。

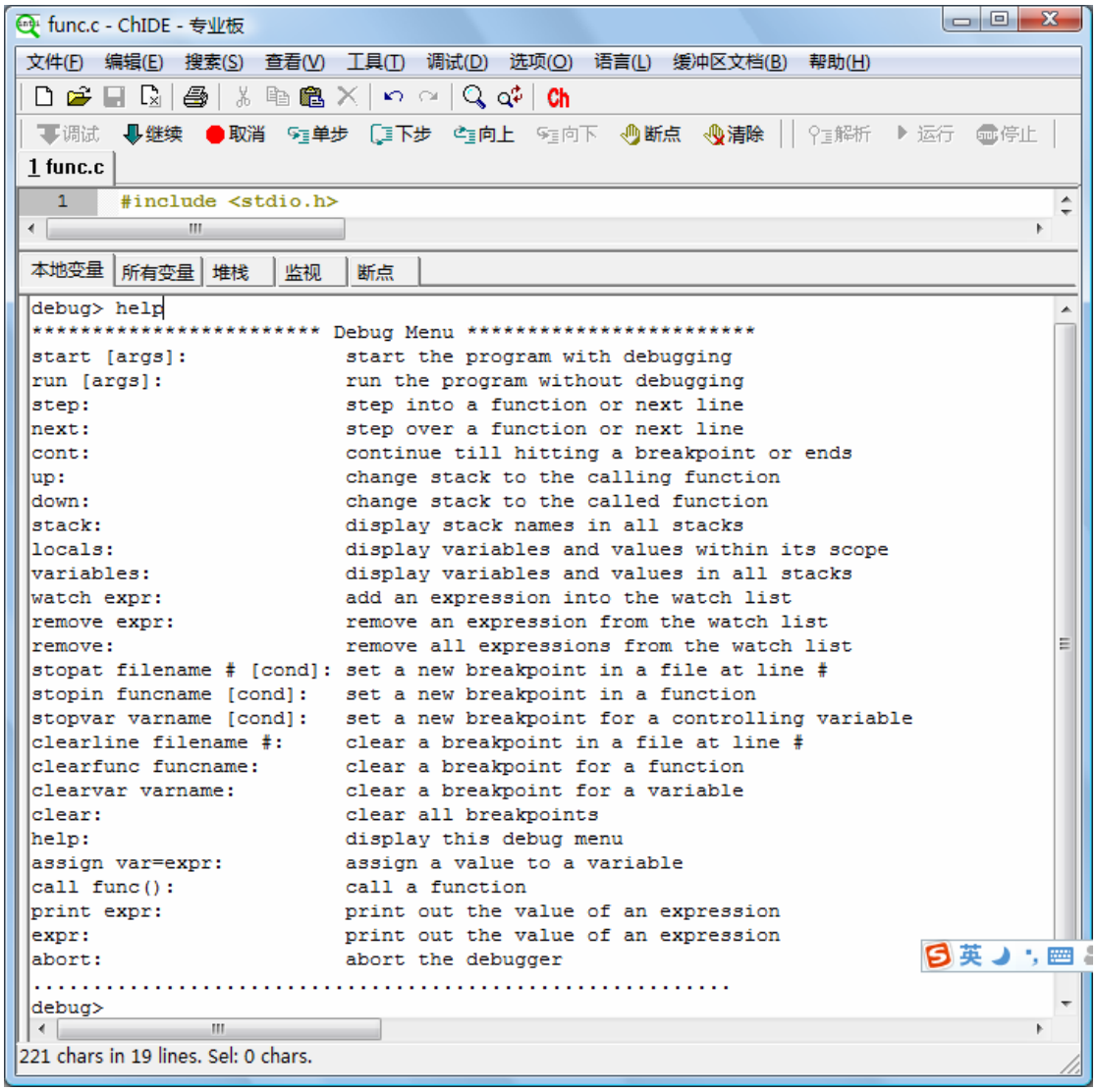


图 26 调试窗格中的调试命令

调试命令“assign”、“call”、“print”可以用来操作变量、表达式、函数，其中命令“assign”给变量赋值；命令“call”调用函数；命令“print”输出函数、变量或表达式的值。无法使用“print”命令输出void 类型的表达式以及返回值为void 类型的函数的值。在调试命令窗格中，输入一个表达式后按下回车键即可显示该表达式的值。如果表达式是一个带有返回值为void类型的函数，则只调用该函数，而不计算表达式的值。例如：命令

```
debug> assign i=2*10
debug> call func()
```

```
debug> print i
20
debug> 2*i
40
debug>
```

将数值2*10赋值给变量i，调用函数func()，在变量i的有效范围内输出表达式2*i的值。另一个例子，如图 27所示，当我们调试运行程序func.c并停在第9行时，在调试命令窗格中输入相应命令可以得到变量i及表达式2*g的值。

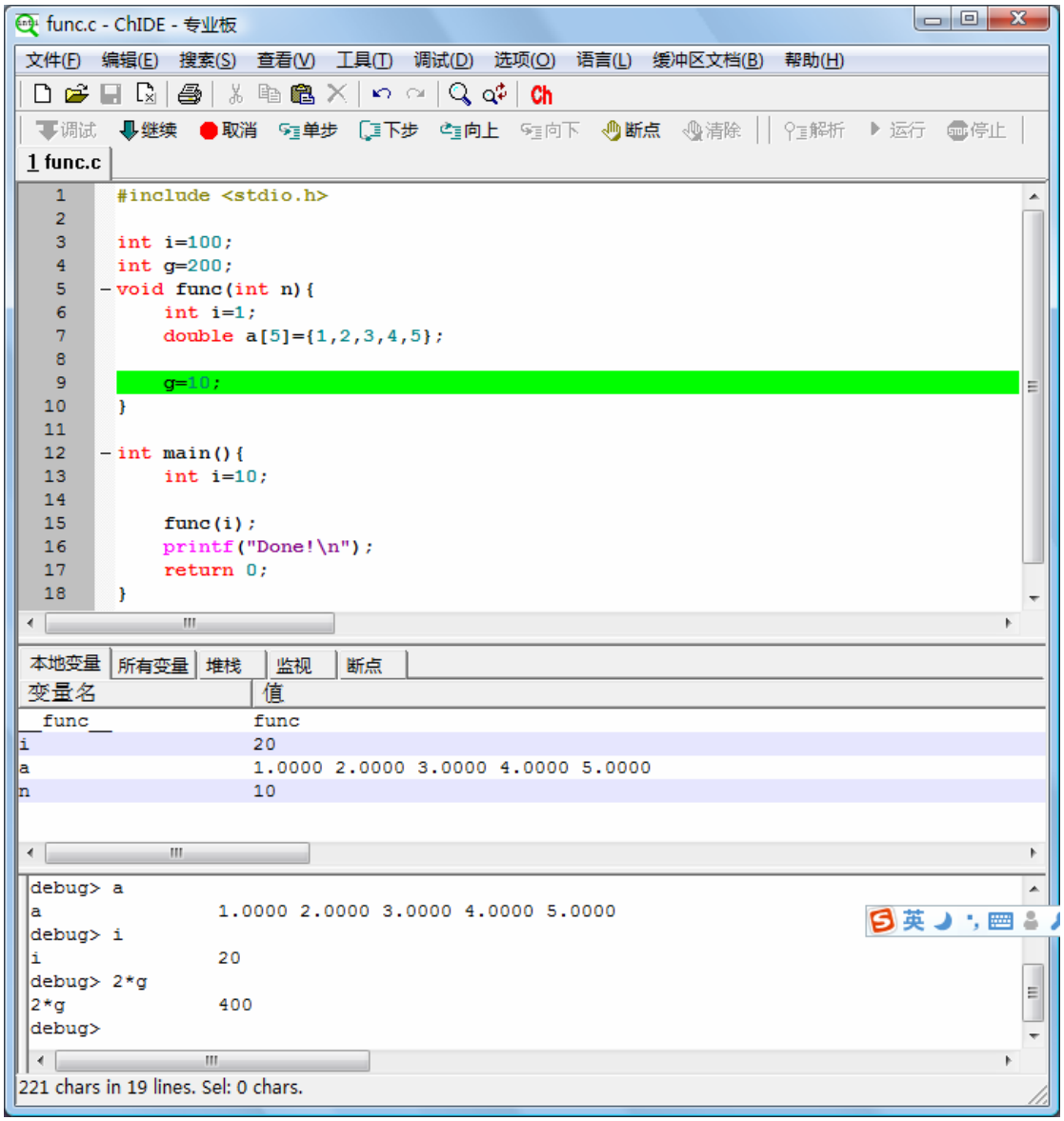


图 27 在调试命令窗格中使用调试命令

在调试命令窗格中使用调试“start”命令可以使一个程序进入调试状态。执行调试命令“**start**”和“**run**”时可加入命令行参数（可选项），这些参数经过处理并传递给主函数**main()**，作为主函数**main()**的实参。若想运行如图 17所示程序（该程序存放在：C:\Ch\demos\bin\commandarg.c），在调试命令窗格中输入调试命令

debug> start -o option1 -v option2 “option3 with space”

则字符串“C:\Ch\demos\bin\commandarg.c”、“-o”、 “option1”、“-v”、“option2”和“option3 with space”将分别赋值给主函数

```
int main(int argc, char *argv[])
```

的 argv 参数：argv[0]、argv[1]、argv[2]、argv[3]、argv[4]和 argv[5]。调试控制台窗口的输出将和图 17中所示的输出窗格中的内容一样。带有空格的命令行参数应该用双引号将其引起来，如前述的字符串“option3 with space”所示。

当遇到有断点时，程序将停止运行。但如果使用“run”命令，则程序将忽视断点直接运行直到结束。与调试工具栏上的命令相似，用户通过使用调试命令“**step**”或“**next**”，可以逐行运行程序。执行调试命令“**step**”时，若遇到函数则进入函数进行调试；执行调试命令“**next**”时，若遇到函数则直接执行该函数后，并停在该函数的下一行。在调试中，调试命令“**cont**”继续运行程序直到遇到断点或者程序结束；用户可以切换显示不同函数的堆栈，即可以通过使用调试命令“**up**”、“**down**”分别进入显示调用函数和被调用函数的堆栈信息；在调试命令窗格中可访问在其作用域内的任何变量；调试命令“**stack**”可以显示所有堆栈中的函数和程序名称。调试命令“**locals**”可以显示当前堆栈中的所有变量名称及其值；调试命令“**variables**”显示所有堆栈中的所有变量及其值。

调试命令“**watch**”后加上一个表达式（包括简单变量），其作用是将该表达式添加到监视表达式列表中。可以在程序运行时或运行前添加监视表达式。可以通过使用调试命令“**remove expr**”将表达式“expr”从监视表达式列表中删除。不带表达式的调试命令“**remove**”将删除监视表达式列表中的所有内容。例如，如图 28所示，在调试命令窗格中输入命令：

```
debug> watch 2*g
debug> watch i
```

则添加表达式2 * g和变量i的到监视表达式列表中。当程序在停在断点上或用单步命令执行到下一个语句时，这些被监视表达式的值可以通过点击调试窗格顶部的“监视”选项卡来查看，如图 28所示。

在程序运行之前或在程序的调试执行过程中，可以添加新的断点来暂停程序的运行。利用调试命令可以用三种方式设置断点：文件名及行号、函数和控制变量。若在函数中设置断点，程序会在进入函数第一个可执行语句处停止运行。若为一个变量设置断点，则当变量的值改变时，程序在该变量处停止运行。设置每个断点时都可以为该断点设置一个可选项，即设定一个条件表达式，当程序执行到达断点处时，若该断点存在条件表达式，则系统会计算该条件表达式的值，只有断点处的条件表达式的值为真或被改变时（到底要在条件表达式的值为真或被改变时触发断点，需要在设定该断点时指定），此断点才会被触发。在默认情况下，只有表达式为真，断点才会被触发。使用调试命令“**stopat**”设置新断点时，其后的参数指明该断点所在的文件名和行号，当程序运行到此位置时就会停止。使用调试命令“**stopin**”能为函数设置新断点，程序会在进入该函数的第一个可执行行处停止运行。使用调试命令“**stopvar**”能为控制变量设置新断点，程序运行时计算该变量的值，当变量的值改变时，程序停止运行。每当执行完这些命令，断点列表会添加一个断点。前述设置断点的调试命令的最后两个可选的参数为：条件表达式和触发模式。例如，在某个程序（注意要写全其访问路径）的某行（下面用#表示该行的行号）设置一个断点的调试命令如下：

```
debug> stopat filename #
debug> stopat filename # condexpr
```

```
debug> stopat filename # condexpr condtrue
```

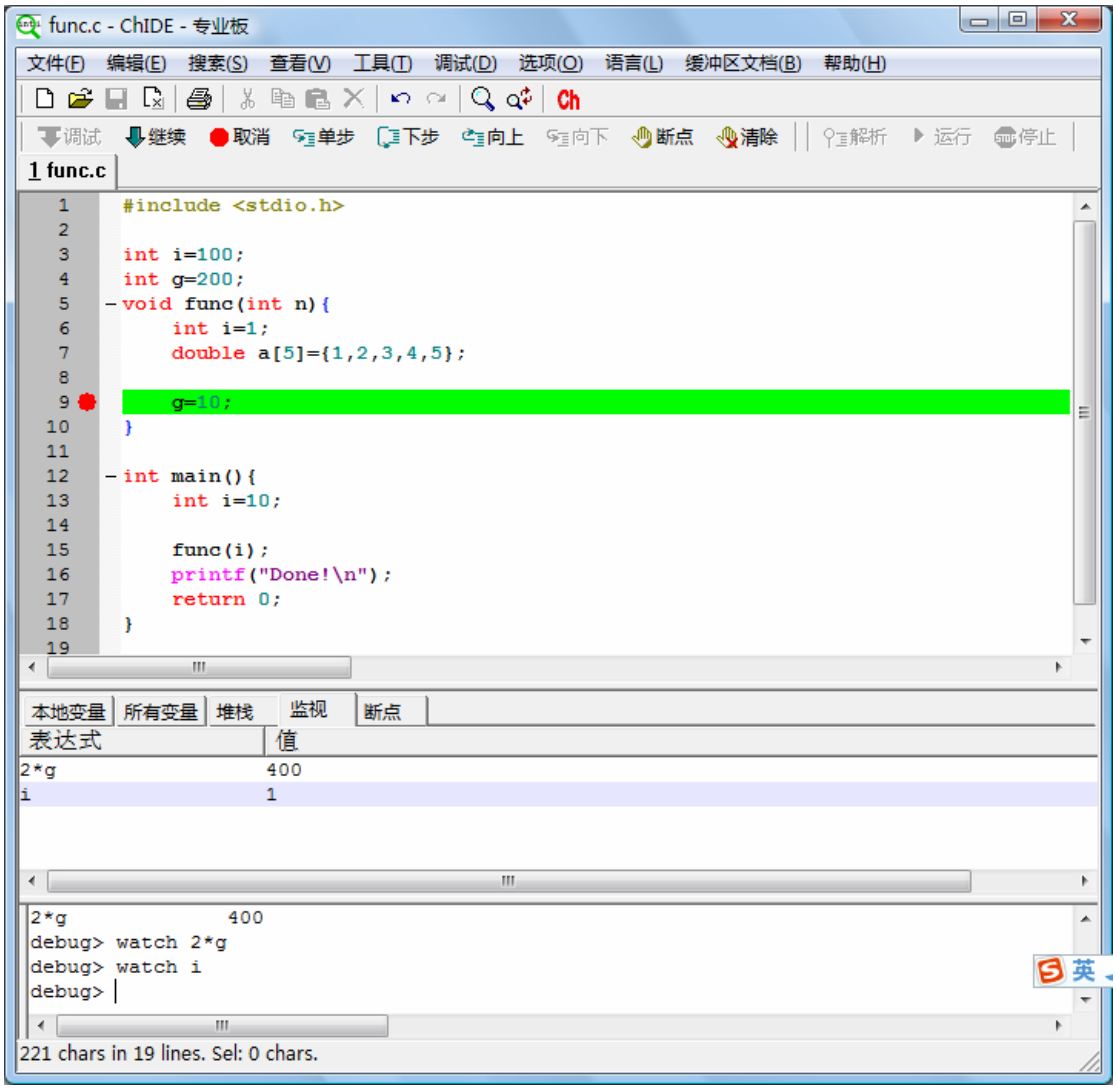


图 28 在调试命令窗格中设置监视表达式，在调试窗格查看表达式的值

当程序运行至该断点处时，系统会计算条件表达式“condexpr”的值，若该表达式的值为“真”时，并且调试命令中设定“condtrue”为“真”或没有设定“condtrue”这个参数的值时，系统才会触发该断点；反之，若用户在设定断点时，设定“condtrue”这个参数的值时为“假”时，则当条件表达式“condexpr”的值改变时，系统才会触发该断点。例如，如下命令在目录 C:/Ch/demos/bin 中的程序 func.c 的第 6 行处设置一个断点：

```
debug> stopat C:/Ch/demos/bin/func.c 6
```

下面的这个调试命令，当程序运行到第6行时，系统会计算表达式 i+j 的值，如果表达式 i+j 的值为真，断点被触发。

```
debug> stopat C:/Ch/demos/bin/func.c 6 i+j 1
```

上面的调试命令和下面的这条调试命令作用一样：

```
debug> stopat C:/Ch/demos/bin/func.c 6 i+j
```

命令：

```
debug> stopat C:/Ch/demos/bin/func.c 6 i+j 0
```

同样在func.c的第6行设置断点，当程序运行到第6行时，系统会计算计算表达式 i+j 的值 ，如果表达式i+j的值改变了，则该断点被触发。

此外，带有相应参数的调试命令“**clearline**”、“**clearfunc**”和“**clearvar**”能分别清除设置在行、函数、变量处的断点。调试命令“**clear**”清除调试器的所有断点。

如果程序运行失败或者太长时间运行，命令“**abort**”能中止程序运行。

如图 10所示，菜单命令“查看|清空调试命令窗格”可以清空调试命令窗格中的内容。

5. Ch Shell 命令入门

像一些传统的shell程序，如MS-DOS shell、Bash-shell和C-shell一样，Ch shell 也可以运行命令。与这些传统shell程序不同的是，C/C++的表达式、语句、函数和程序均可以在Ch Shell中运行。

执行命令Ch可以启动Ch Shell。在Windows、Linux和Mac OS X操作系统中，可以通过点击桌面或ChIDE工具栏上的红色的Ch图标（如图 29所示）来启动Ch Shell。

图 29 Windows，Linux 和 Mac OS X操作系统桌面上的 Ch 图标

假设我们使用管理员帐户，即“Administrator”，则在 Windows 操作系统中运行 Ch shell 后，Ch Shell命令窗口中的屏幕提示符变为：

```
C:/Documents and Settings/Administrator>
```

如图 30所示，其中，C:/Documents and Settings/Administrator> 是桌面上用户的主目录。Ch shell窗口的背景颜色、文本颜色以及窗口大小和字体大小都可以通过右击窗口左上角的Ch图标、选取右键菜单中的“属性”项来改变相应的设置。需要特别说明的是，在Windows Vista系统中，用户需要具有管理者权限才能完成上述操作。目录C:/Documents and Settings/Administrator也被称为当前工作目录。如果用户账户不是管理员，账户名称“Administrator”将会被修改为相应的账户名称。提示符的出现表明Ch Shell系统可以接受用户的输入命令了。默认的提示符“>”可以设定为其它符号。如果用户输入的命令语句正确，则Ch shell将会成功运行用户输入的命令。运行成功后，系统会再次显示“>”。如果在程序或表达式运行过程中发生错误，Ch Shell会输出相应的错误消息帮助用户调试程序。

C语言中所有的语句和表达式都可以在Ch命令窗口中交互式运行。例如，下面的命令：

```
C:/Documents and Settings/Administrator> printf("Hello, world")
Hello, world
```

通过调用函数**printf()**输出字符串“Hello，world”，如图 30所示。

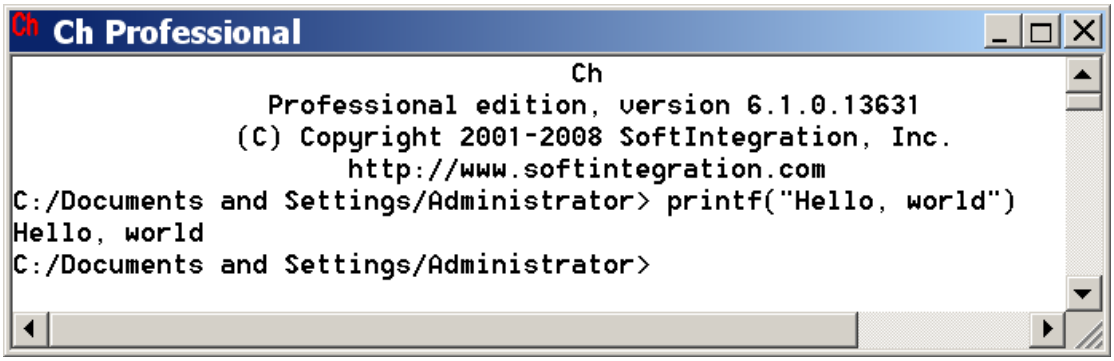


图 30 Ch shell窗口

同图 30相比，为了节省本文的空间，我们省略了最后一行提示“C:/Documents and Settings/Administrator>”。值得注意的是：在命令窗口中输入的C语言语句中，语句最后的分号是可以省略的，如前面所示的调用函数**printf()**的语句结尾就没有分号。

5.1. 跨平台处理文件的命令

在系统提示符“>”下，Ch Shell系统不仅可以运行 C 程序及语句，也可以运行其它命令（例如，用于输出当前工作目录的命令**pwd**）。在这种情况下，Ch shell的用法和 Windows 操作系统下 MS-DOS shell的用法是一样的。

在Ch Shell命令窗口或 Ch程序 中均可以运行命令，每一种计算机操作系统都有上千个系统命令包括相应的在线帮助文档，没有人知道所有这些命令。但每个计算机专家(computer wizard)都有一个较小的经常使用的命令集和工具，并大体知道很多其它命令的用途。在这一部分中，我们会通过例子描述如何使用常用的处理文件命令，如表 4所示。Ch shell中的这些命令是跨平台的，如Windows、Linux或者Mac OS X。使用这些命令，用户可以有效地在这些不同的平台上操作文件和运行C程序。

假定Ch被安装在Windows中默认安装目录 C:/Ch 中，当前工作目录为 C:/Documents and Settings/Administrator，该目录也是用户的主目录。用于文件处理的跨平台命令的使用方法可用如下例子说明：

```
C:/Documents and Settings/Administrator> mkdir c99
C:/Documents and Settings/Administrator> cd c99
C:/Documents and Settings/Administrator/c99> pwd
C:/Documents and Settings/Administrator/c99
C:/Documents and Settings/Administrator/c99> cp
C:/Ch/demos/bin/hello.c hello.c
C:/Documents and Settings/Administrator/c99> ls
hello.c
C:/Documents and Settings/Administrator/c99> chide hello.c
```

如表 4所示的用法一样，命令**mkdir**用于创建新目录，命令**mkdir**有一个参数，即要创建的目录的名称。首先，我们使用如下命令创建一个名叫c99的目录：

```
mkdir c99
```

其次我们使用以下命令改变工作目录到新的目录C:/Documents and Settings/Administrator/c99：

```
cd c99
```

然后，我们使用以下命令显示当前工作目录：

```
pwd
```

为将图 2所示的位于目录 C:/Ch/demos/bin 中的 C 程序 hello.c复制到当前工作目录中，使用命令：

```
cp C:/Ch/demos/bin/hello.c hello.c
```

当前目录中的文件可以使用如下命令显示：

```
ls
```

这时，工作目录C:/Documents and Settings/Administrator/c99中只有一个文件hello.c。建议你将所有你编写的C语文程序都存放在该目录中，这样以后可以很方便地找到你需要的程序。

最后，使用如下用命令启动程序 hello.c：

```
chide hello.c
```

结果如图 2所示。这样就可以在ChIDE中编辑及执行程序了。

表 4 用于处理文件的跨平台命令

命令	用法	描述
cd	cd	改变主目录
	cd dir	改变目录 dir
cp	cp 文件 1 文件 2	复制文件 1 到文件 2
ls	ls	列出工作目录的内容
mkdir	mkdir dir	创建新目录 dir
pwd	pwd	显示工作目录的名称
rm	rm 文件	将文件放到回收站中
chmod	chmod+x file	改变文件模式使之可执行
chide	chide 文件.c	运行 ChIDE 编辑和运行文件.c

为了便于课堂演示，可以用如下命令打开多个源文件：

- `chide file1.c file2.c header.h`

当你要处理的文件的路径带有空格时，则要将路径放置在一对双引号内，如用下面这条命令删除相应路径下的文件hello.c：

```
> rm  "C:/Documents settings/Administrator/c99/hello.c"
```

5.2. Ch 中命令、头文件及函数搜索路径的设置

在Ch shell中输入命令并运行时，Ch Shell会在预设的目录中搜索是否存在该命令。在Ch Shell

中，字符串系统变量**_path**提供命令搜索的路径。在变量**_path**中的各个路径用分号隔开。当启动Ch shell时，系统变量**_path**包含默认的搜索路径。例如，在 Windows 中，默认搜索路径为：

```
C:/Ch/bin;C:/Ch/sbin;C:/Ch/toolkit/bin;C:/Ch/toolkit/sbin;C:/WINDOWS;C:/WINDOWS/SYSTEM32;
```

用户可以使用字符型函数**stradd()**在Ch shell的启动文件中添加新的搜索路径，添加方法我们将在稍后说明。该函数的参数为字符串，其返回值也是字符串。例如，假定目录 C:/Documents and Settings/Administrator/c99并不包含在命令搜索路径中，若当前工作目录为 C:/Documents and Settings/Administrator，这时你试图运行程序 hello.c 时，由于Ch shell 无法找到该程序，所以会给出两条出错信息：

```
C:/Documents and Settings/Administrator> hello.c
ERROR: variable 'hello.c' not defined
ERROR: command 'hello.c' not found
```

启动 Ch 或 运行Ch 程序时，在默认情况下，在Unix、Linux及Mac OS X下，Ch会读取用户主目录下的启动文件.**chrc**；在Windows下会读取用户主目录下的启动文件**_chrc**。在下文中，我们假定Ch在Windows中使用，并且在启动时会读取用户主目录下的启动文件**_chrc**，该启动文件中设置了用于命令、函数、头文件等的搜索路径。在Windows中安装Ch时，系统会在用户主目录中创建一个默认配置的启动文件**_chrc**。然而，在默认情况下，Unix 系统中的用户主目录中没有该启动文件。系统管理员可以在用户主目录中添加这样一个启动文件。如果用户主目录中没有启动文件，用户可以在启动Ch 时加上可选参数 “–d”，如下所示：

```
ch –d
```

从目录CHHOME/config/中将启动样本文件复制到用户目录下。值得注意的是这里的CHHOME不是字符串“CHHOME”，而是Ch主目录。如前所述，默认情况下，Ch主目录在Windows操作系统是C:/Ch，在Unix操作系统是/usr/local/ch。

在 Windows 中，在Ch shell中执行命令

```
C:/Documents and Settings/Administrator>ch –d
```

将会在用户主目录C:/Documents and Settings/Administrator 中创建一个启动文件 **_chrc**。这个文件可以可以通过ChIDE的菜单命令

“选项丨打开Ch本地初始化文件”

打开与编辑，如图 31所示。在Linux系统中，上述命令**ch –d** 还将在桌面上创建Ch图标。如果安装Ch的同时也安装了ChIDE，上述命令也会自动在桌面上创建ChIDE图标。

若想在搜索路径中包含目录 C:/Documents and Settings/Administrator/c99，以下语句：

```
_path=stradd(_path,"C:/Documents and Settings/Administrator/c99;");
```

需要添加到用户主目录中的启动文件**_chrc** 中。这样无论当前工作目录是什么，Ch 都可以执行该目录下的程序hello.c。当 C:/Documents and Settings/Administrator/c99 被添加到搜索路径**_path** 后，你必须重新启动 Ch Shell。这时，你就可以在如下所示的目录下运行程序 hello.c。

```
C:/Documents and Settings/Administrator> hello.c
Hello, world
```

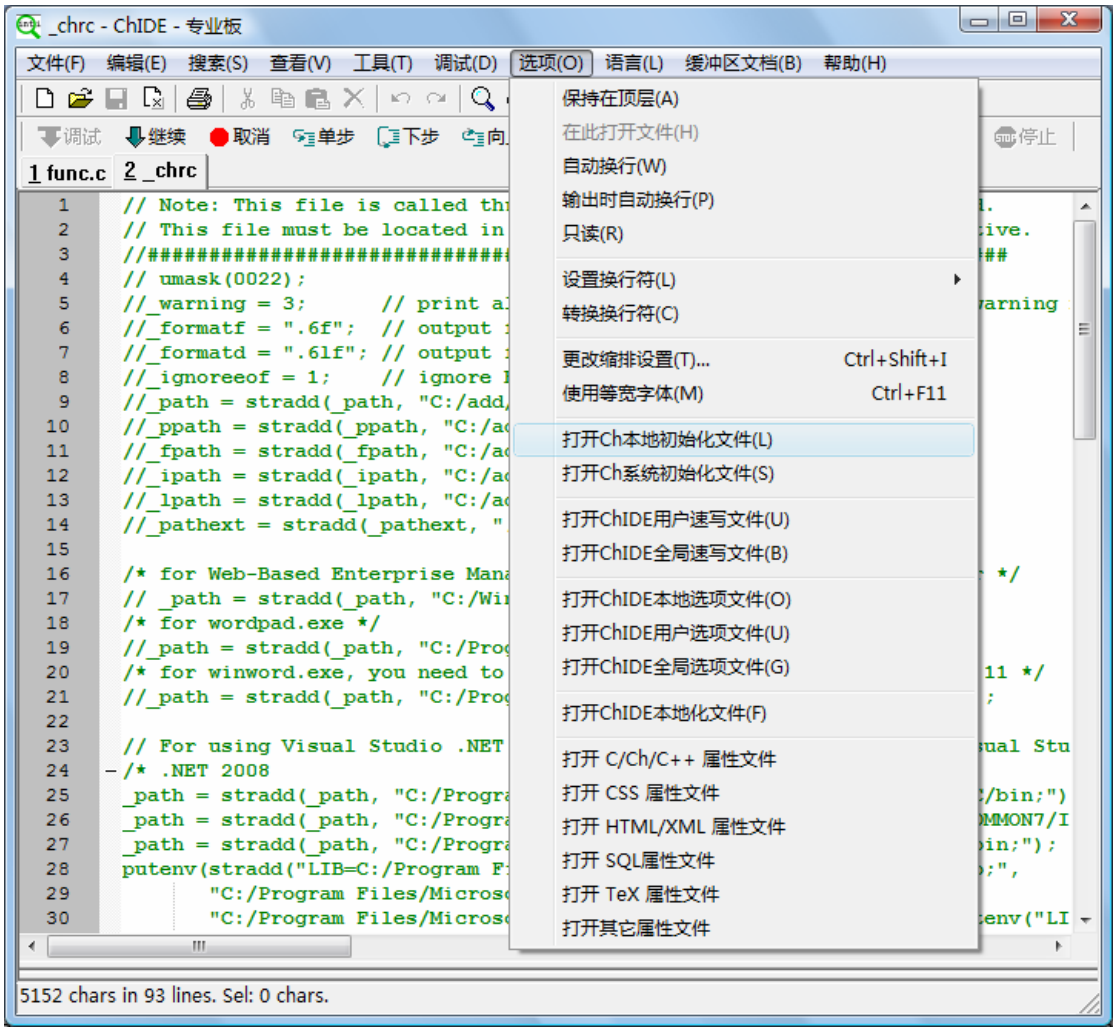


图 31 打开Ch启动文件进行编辑

在Unix或Mac OS X系统中，默认情况下搜索路径不包含当前工作目录，在用户主目录中的启动文件 **.chrc** 中加入以下语句可以将当前工作目录加入到搜索路径中：

```
_path = stradd(_path, ".;");
```

上述语句中的“.”表示当前工作目录。

与**_path**命令类似，Ch 将在在系统变量**_ipath**指定的搜索路径中搜索程序所包含的头文件，**_ipath**中的每个路径以分号隔开。例如，以下语句：

```
_ipath = stradd(_ipath, "C:/Documents and Setting/Administrator/c99;");
```

将目录 C:/Documents and Setting/Administrator/c99 添加到头文件的搜索路径中。头文件是由预处理指令 **include** 声明的文件，例如：

```
#include <headerfile.h>
```

同样地，用户也可以用以下语句将前述目录添加到函数文件的搜索路径**_fpath** 中：

```
_fpath = stradd(_fpath,"C:/Documents and Setting/Administrator/c99;");
```

这里所说的函数文件是指包含函数定义的文件，我们将在第5.5节详细介绍。

5.3. 交互式执行 C/Ch/C++程序

在Ch Shell中无需编译即可简单方便地运行 C 程序。如5.1节所示，我们假设当前工作目录为C:/Documents and Settings/Administrator/c99，在 Ch 中运行该目录中的程序 hello.c 会得到如下所示的输出：

```
C:/Documents and Settings/Administrator/c99> hello.c
Hello, world
C:/Documents and Settings/Administrator/c99> _status
0
```

在Ch Shell系统中，程序运行的返回代码被保存在系统变量**_status** 中。因为程序hello.c 成功运行，所以其返回代码为0，在命令行输入“**_status**”时即可显示刚执行过的程序的返回代码。

在 Unix、Linux及Mac OS X 中，为了让C程序 hello.c 能够以命令方式被执行，则必须将该文件模式改为可执行类型，我们可以利用命令 **chmod** 可以改变文件的模式，使用命令：

```
chmod +x hello.c
```

后，程序 hello.c 就被设定为可执行类型，这时就可以在 Ch 命令窗口中直接运行程序 hello.c。

5.4. C/Ch/C++表达式和语句的交互式执行

在后面章节中为了简化表达，Ch 命令行中只给出提示符“>”。如果在 Ch shell 中输入一个 C 的表达式，那么 Ch 将会计算该表达式的值并将结果显示在屏幕上。例如，输入一个表达式 1+3*2，Ch 将输出结果 7，如下所示：

```
> 1+3*2
7
```

任何一个有效的 C 表达式都可以在 Ch 中得以计算，因此 Ch 也可以方便地作为一个计算器来使用。

另一个例子是在提示符下可以先声明一个变量，然后再进行后续计算，如：

```
> int i
> sizeof(int)
4
> i = 30
```

```
30
> printf("%x", i)
1e
> printf("%b", i)
11110
> i = 0b11110
30
> i = 0x1E
30
> i = -2
-2
> printf("%b", i)
11111111111111111111111111111110
> printf("%32b", 2)
00000000000000000000000000000010
```

在上面的例子中，变量i被定义为 **int** 类型，一般情况下它将占用4字节内存，然后将整数30赋值给变量i，再以十进制、16进制、二进制形式显示变量i的值；接下来，以不同进制下的数字常量赋值给变量i；最后，给出了-2的二进制补码。当然所有的C语言数据类型均可以用这样交互的方式进行展示。

在默认情况下，**float**类型与**double**类型的数值分别以2位和4位小数据位进行显示，如：

```
> float f = 10
> 2*f
20.00
> double d = 10
> d
10.0000
```

指针变量及变量的地址可以用下面的语句进行演示：

```
> int i=10, *p
> &i
1eddf0
> p = &i
1eddf0
> *p
10
> *p = 20
20
> i
20
```

在上述例子中，我们用指针变量p指向变量i的地址。指向指针的指针变量也可以用同样的方法

进行演示。下面的例子展示了数组与指针间的关系：

```
> int a[5] = {10,20,30,40,50}, *p;
> a
1eb438
> &a[0]
1eb438
> a[1]
20
> *(a+1)
20
> p = a+1
1eb43c
> *p
20
> p[0]
20
```

在上例中，a[1]、*(a+1)、*p、p[0]变四种方法得到的是同一数组元素的值。多维数组与指针间的关系也可以用同样的方法进行展示。在Ch中，为了防止潜在的错误发生，系统会检查数组的边界，如：

```
> int a[5] = {10,20,30,40,50}
> a[-1]
WARNING: subscript value -1 less than lower limit 0
10
> a[5]
WARNING: subscript value 5 greater than upper limit 4
50
> char s[5]
> strcpy(s, "abc")
abc
> s
abc
> strcpy(s, "ABCDE")
ERROR: string length s1 is less than s2 in strcpy(s1,s2)
ABCD
> s
ABCD
```

在上述例子中，具有5个数据元素的数据a的合法下标为从0到4，字符串数组s只能存放具有4个字母的字符串，因为每个字符串结尾均需要有一个结束符‘NULL’，即‘\0’。诸如此类的与数据边界有关的错误(bug)，Ch均会发出一个警告信息。

C语言中的结构体与C++中的类的字节对齐方式可以用以下例子来展示：

```
> struct tag {int i; double d;} s
> s.i =20
20
> s
.i = 20
.d = 0.0000
> sizeof(s)
16
```

在该例子中，虽然**int** 类型和 **double** 类型在内存中分配的存贮空间分别为4和8。但包含一个 **int** 类型和一个 **double** 类型成员变量的结构体 s 所占用的内存空间为16，而不是12，这就是为了结构体或类的字节对齐。

5.5. C/Ch/C++函数的交互式执行

一个程序可以由多个文件组成，每个文件包含若干个相关的函数，这些函数之间是可以相互可调用的。C 标准库中的所有函数都可以被用户定义的函数调用，也可以在Ch shell中被交互式运行，例如：

```
> srand(time(NULL))
> rand()
4497
> rand()
11439
> double add(double a, double b) {double c; c=a+b+sin(1.5); return c;}
> double c
> c = add(10.0, 20)
30.9975
```

函数**srand()**以当前系统时间为参数设定产生随机数的随机序列，调用随机函数**rand()**产生随机数。在Ch shell命令提示符下定义了一个名字为add的函数，它调用通用数学函数**sin()**，接下来调用了add函数。

通常我们将包含多个函数定义的文件的扩展名命名为.ch，以此表明它是一个 Ch 程序的一部分。可以在Ch编程环境下创建单函数文件，即它只包含一个函数定义，通常情况下将其扩展名命名为.chf，例如：addition.chf，单函数文件的文件名与其中定义的函数名称必须相同。单函数文件中的函数在Ch中会被当作内置函数处理。

与**_path** 相似，函数的搜索路径被保存在系统变量**_fpath**中，其中的每个路径用分号隔开。在默认情况下，变量**_fpath** 包含Ch主目录路径下的如下路径：lib/libc、lib/libch、lib/libopt和libch/numeric。如果Ch shell中的系统变量_fpath 被交互式修改，那么它仅对当前shell环境下被调用的函数有效。子shell不能使用、继承父shell中设置的函数搜索路径。所以要想直接运行这些单函数文件，需要修改启动文件中的系统变量**_fpath**，注意，在Windows系统中，启动文件为用户主目录下的文件**_chrc**，在Unix系统中，启动文件为用户主目录下的文件.**chrc**。

例如，单函数文件addition.chf的文件所包含的程序如程序 1所示。其中的函数addition()将被当作一个系统内置函数，可以被直接调用，用于计算两个输入参数 a、b 之和。假定 addition.chf 存

放在C:/Documents and Settings/Administrator/c99目录中，目录C:/Documents and Settings/Administrator/c99应该被添加到用户主目录中启动文件中(在Windows系统中，启动文件为用户主目录下的文件**_chrc**，在Unix系统中，启动文件为用户主目录下的文件.**chrc**)，具体语句如下：

```
_fpath=stradd(_fpath, "C:/Documents and Settings/Administrator/c99;");
```

这样，函数 addition()就可以在Ch Shell命令窗口中被交互的调用，如下所示：

```
> int i = 9
> i = addition(3, i)
12
```

```
/* File: addition.chf
A function file with file extension .chf */
int addition(int a, int b) {
    int c;
    c = a + b;
    return c;
}
```

程序 1　单函数文件 addition.chf

```
/* File: program.c
   Program uses function addition() in function file addition.chf */
#include <stdio.h>
/* This function prototype is optional when function addition() in
   file addition.chf is used in Ch */
int addition(int a, int b);
int main() {
   int a = 3, b = 4, sum;
   sum = addition(a, b);
   printf("sum = %d\n ", sum);
   return 0;
}
```

程序 2 使用函数文件 addition.chf 的程序

当然，在程序中也可以调用这个函数addition()，在程序 2中，由于函数addition()的原型已在单函数文件addition.chf 中定义，所以在**main()**函数不必定义函数addition()的原型，而可以直接调用addition()函数。但是如果函数文件的搜索路径不包含文件addition.chf所在的目录，调用函数addition()时，将会出现如下警告消息：

WARNING: function ‘addition()’not defined

在 Ch Shell中，调用一个单函数文件中的函数时，这个单函数文件将被调入内存中。如果你在该单函数文件被调用后修改了此函数文件，接下来在Ch shell中再次调用该函数时，系统仍然调用的

是你修改之前的调入内存的函数。为了能够调用修改后的函数，则可以使用以下两种方法：在命令窗口提示符后输入ch，启动一个新的 Ch shell；使用命令**remvar**去除系统中原先定义的函数，其命令为：

```
> remvar addition
```

将函数 addition()从内存中卸载。命令 **remvar** 也可以用于去除已声明的变量。

5.6. C++功能的交互式执行

Ch 不仅可以交互式运行 C 程序，而且支持 C++的类及其他功能。例如：

```
> int i
> cin >> i
10
> cout << i
10
> class tagc {private: int m_i; public: void set(int); int get(int &);}
> void tagc::set(int i) {m_i = 2*i;}
> int tagc::get(int &i) {i++; return m_i;}
> tagc c
> c.set(20)
> c.get(i)
40
> i
11
> sizeof(tagc)
4
```

在 C++中可以使用 **cin** 和 **cout** 进行输入和输出操作。通过方法 tagc::set() 来设置类tagc的私有成员m_i的值，通过方法 tagc::get()来得到类tagc的私有成员m_i的值。通过引用来传递方法tagc::get()的参数。类 tagc 占用内存空间为4个字节，当然这不包括成员函数所占用的存贮空间。

6. 在输出窗格中交互执行命令

在输出窗格中可以交互式地运行命令和C/C++程序。如图 32所示，首先运行程序hello.c；然后使用命令 **pwd** 显示当前工作目录，使用命令 **ls** 列出当前工作目录中的文件名和目录。带有可选命令参数的命令也可以在输出窗格中使用。例如，使用如下形式的**ls**命令可以将列出的内容进行分类：

```
ls -F
```

当要执行的文件的路径中带有空格时，需要用一对双引号括起来，如下所示：

```
> "C:/Documents and Settings/Administrator/c99/hello.c"
```

执行带有命令行参数的C/Ch/C++程序的方法请参见第2.5节。

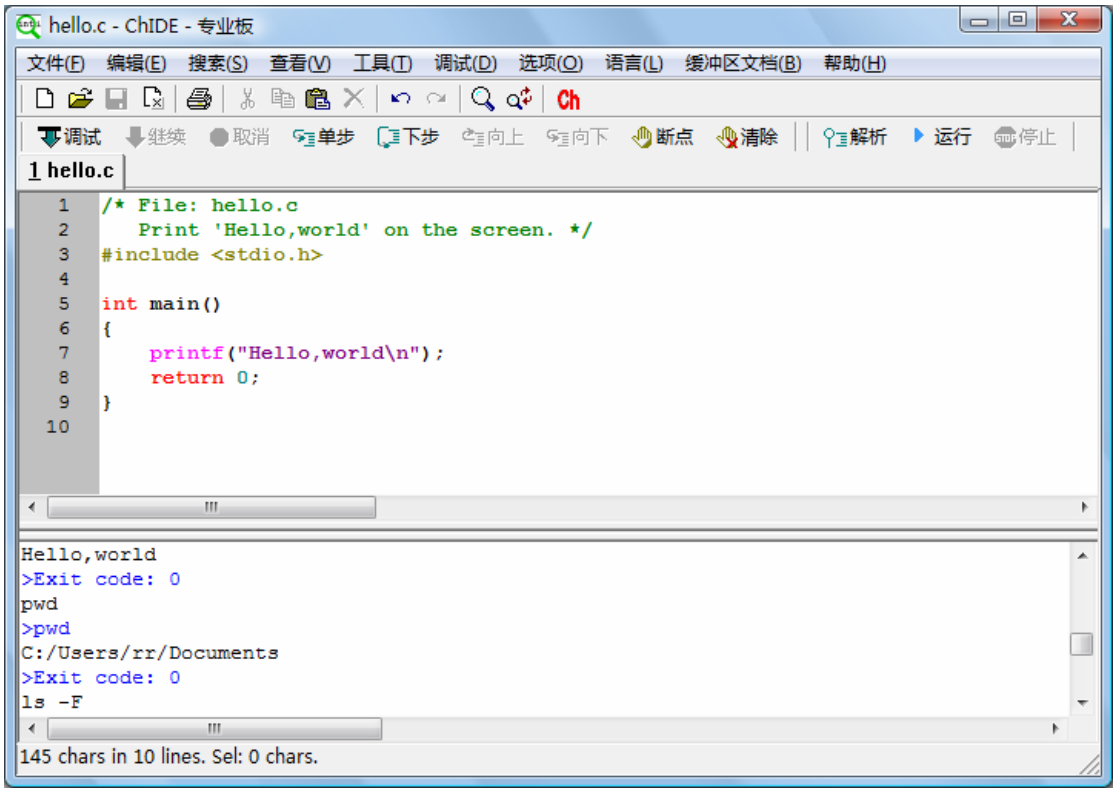


图 32 在输出窗格中执行命令

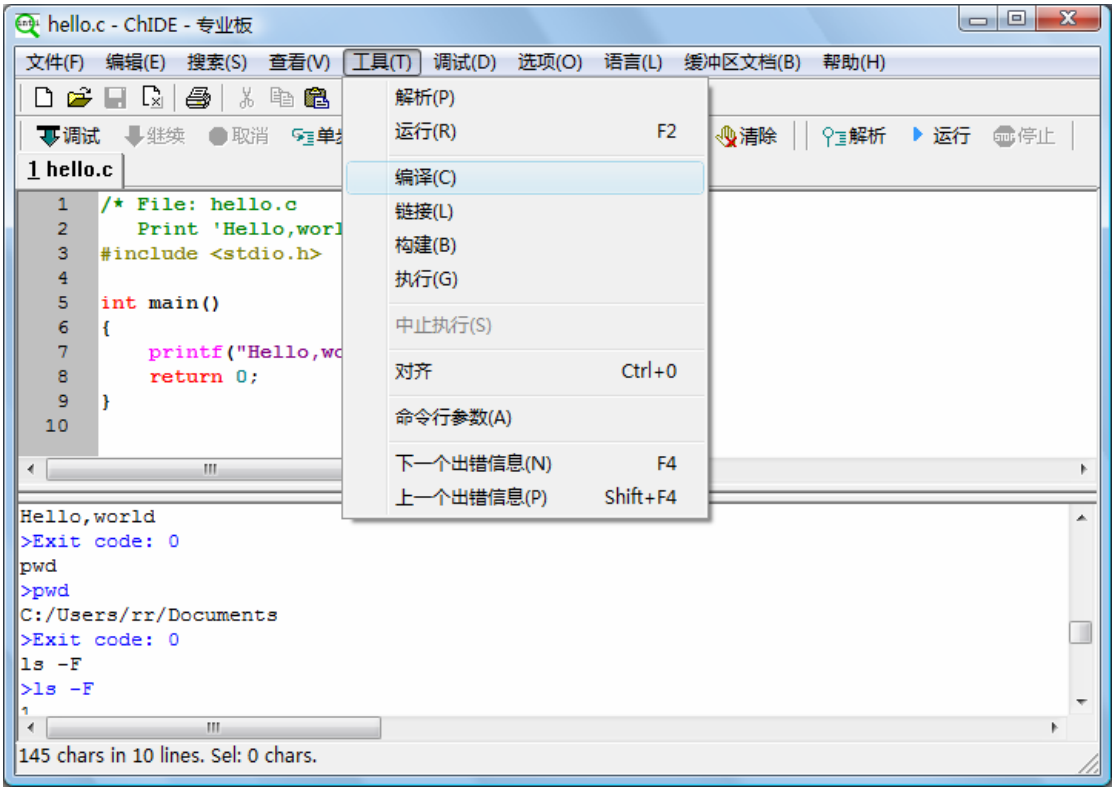


图 33 编译C/C++程序

7. 在 ChIDE 中编译和链接 C/C++程序

ChIDE也可用C/C++编译器来编译、链接C/C++程序，然后执行产生的可执行二进制程序。在Windows中，安装ChIDE时，系统会自动设置使用最新版本的 Microsoft Visual Studio .NET编译器来编译C/C++程序。与Visual Studio 编译器有关的环境变量和命令存放在用户主目录中的启动配置文件**_chrc**中，如图 31所示，该文件可以被打开和编辑。在 Linux 和 Mac OS X x86 中，ChIDE 分别使用编译器 GUN gcc 和 g++来编译 C 和 C++ 程序，可以打开 C/Ch/C++ 属性文件cpp.properties来设置采用哪个编译器。在ChIDE中，可以使用菜单命令“选项丨打开C/Ch/C++ 属性文件”来打开C/Ch/C++ 属性文件 cpp.properties。

如图 33所示，菜单命令“工具|编译”可以用来编译程序。编译C或C++程序时出现的错误信息和输出信息都在ChIDE的输出窗格中显示。在Windows中，编译程序的输出是一个扩展名为.obj的对象文件，这个对象文件可以用菜单命令“工具|链接”来链接、生成一个可执行程序。Windows中可执行文件的扩展名为.exe。

如果当前目录中有make文件makefile或Makefile文件，可使用菜单命令“工具|构建”调用make文件来生成应用程序。如图 34所示，也可以通过右键单击make文件的文件名，在出现的右健菜单中选取相应的make命令（在Linux和Mac系统下为make,在Windows系统下为make或nmake）来生成应用程序。

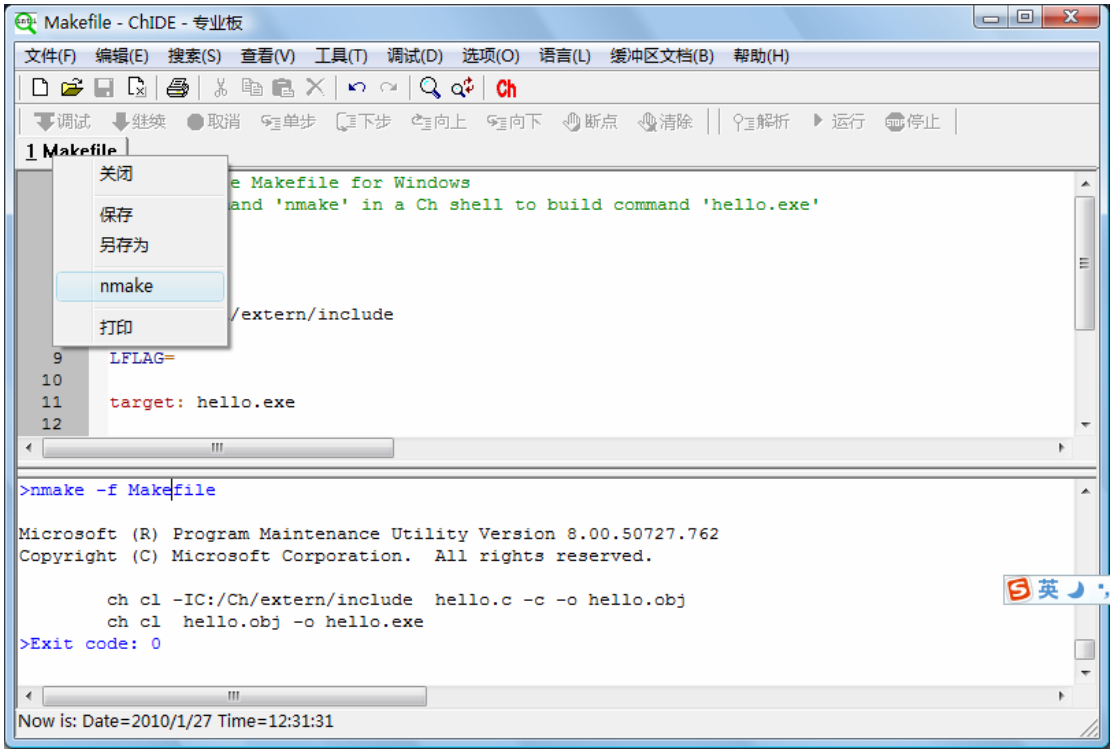


图 34 用makefile来编译C/C++程序

用ChIDE编辑make文件makefile或Makefile文件时，系统会将语句高亮显示。因为在make文件中字符“Tab”被make命令预留作为特殊字符来使用，即它将作为某些编译命令的开始字符，不要将其替换为空格。带有扩展名为.mak的文件或具有以下文件名的文件，会被ChIDE当作make文件：

```
makefile
makefile.win
makefile_win
```

```
makefile.Win
makefile_Win
Makefile
Makefile.win
Makefile_win
Makefile.Win
Makefile_Win
```

菜单命令“工具|执行”可以执行当前开发的程序。

8. ChIDE 可编辑的语言

ChIDE是一个通用文本编辑器，目前在编辑以下语言文件时，能够将其语法结构高亮显示:

- C/Ch/C++*
- CSS*
- HTML*
- Make
- SQL and PLSQL
- TeX and LaTeX
- XML*

其中对于带 * 的语言，ChIDE还支持如3.4节所述的折叠功能。

ChIDE通过文件扩展名来识别语言类别，并调用相应的语言设置来处理这个文件。但你可以通过菜单命令“语言丨某一种具体的语言”来改变当前编辑文件的语言类别。

9. 本地化支持

当在非英文版的操作系统中安装Ch时，ChIDE中的菜单及对话框会以当地语言来显示。目前ChIDE用户界面支持超过30种语言：

南非语、阿拉伯语、巴斯克语、巴西葡萄牙语、保加利亚语、加泰罗尼亚语、简体中文、繁体中文、捷克语、丹麦语、荷兰语、法语、加利西亚语、德语、希腊语、匈亚利语、印度尼西亚语、意大利语、日语、韩语、马来西亚语、挪威语、波兰语、罗马尼亚语、葡萄牙语、俄语、塞尔维亚语、斯洛文尼亚语、西班牙语、西班牙语（墨西哥）、瑞典语、泰国语、土尔其语、乌克兰语、威尔士语。

ChIDE可以很容易地支持一种新的本地化语言。