

Code::Blocks

for C/C++ Novice

Chipset

Copyright (C) 2008-2010 Chipset

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

目 录

封皮.....	0
目录.....	1
前言.....	2
1. 安装Code::Blocks.....	3
1.1 下载.....	3
1.2 安装.....	3
2. Code::Blocks的编程环境配置.....	7
2.1 环境.....	7
2.2 编辑器.....	12
2.3 编译器和调试器.....	13
3. 编写程序.....	17
3.1 创建一个工程.....	17
3.2 添加和删除文件.....	20
3.3 编辑文件.....	25
3.4 编译程序.....	30
3.5 调试程序.....	43
3.6 阅读别人编写的程序.....	74
4. 附录.....	79
4.1 Linux下安装Code::Blocks.....	79
4.2 Mac OS X下安装Code::Blocks.....	80
4.3 Code::Blocks搭配高版本gcc编译器.....	80
4.4 安装配置boost.....	82
特别说明.....	87

前言

用高级计算机语言，例如C、C++，编写的程序，需要经过编译器编译，才能转化成机器能够执行的二进制代码。然而，把头脑中的思想转变成能够正常工作的计算机程序需要付出一定的努力和时间，因为为了让程序能够达到我们想要的结果，我们往往需要反复修改代码。本书的目的是帮助初学者学习组织程序编码逐步隔离并发现程序中的逻辑错误。通过本书，您可以学会怎么一步步的跟踪代码，找到问题出在什么地方，搞明白为何您的程序不能正常运行，这个过程称谓调试程序。手工跟踪能够有效的帮助初学者找到bug出在什么位置，消除bug，让程序正常运行。自动化的工具同样也能够帮助您跟踪程序，尤其当程序很复杂时效果更加明显，这种工具叫做调试器。调试器能够让运行中的程序根据您的需要暂停，查看程序怎么运作的。有些调试器是以命令行的形式工作的，较新的调试器有些具备好的图形界面，调试器能够方便的帮助您看到您定义的变量状态。基于图形界面的调试器是集成开发环境(IDE，即Integrated Development Environment)的一部分。本书的作用就是帮助您学习使用这种环境以便更好的掌握编程技巧。

一个调试器并不能解决您程序中出现问题，它仅仅是一种帮助您编程的工具。您首先应该运用您手中的纸和笔分析程序，搞清到底怎么回事，一旦确定错误大致出在什么位置，便可以用调试器观察您的程序中特定变量的值。通过观察这些代码，可以了解到您的程序是怎么一步步执行的。

C/C++的IDE非常多，对于学习C/C++语言的朋友而言，用什么IDE可能并不重要，重要的是学习C/C++语言本身，不过，会用一款自己习惯的IDE进行程序的编写和调试确实很方便。

本书主要论述一款开源、免费、跨平台的集成开发环境Code::Blocks的安装、配置、以及程序的调试和编译等。Code::Blocks支持十几种常见的编译器，安装后占用较少的硬盘空间，个性化特性十分丰富，功能十分强大，而且易学易用。我们这里介绍的Code::Blocks集成了C/C++编辑器、编译器、和调试器于一体，使用它可以很方便的编辑、调试和编译C/C++应用程序。Code::Blocks具有很多实用的个性化特性，这里只会简单介绍少数几个常用的特性。

我们希望本书能够帮助您体验编程的乐趣的同时也能帮助您提高调试和编写程序的基本功。

如欲了解更多有关Code::Blocks的信息，请访问Code::Blocks的官方网站<http://www.codeblocks.org>。

1. 安装Code::Blocks

1.1 下载

为了安装Code::Blocks IDE, 首先需要下载它们。如果您使用的是Windows 2000 或 Windows XP 或 Windows Vista操作系统, 从下面地址下载Code::Blocks8.02(目前来说, 8.02是最新的版本)这个IDE:

<http://downloads.sourceforge.net/codeblocks/codeblocks-8.02mingw-setup.exe>

以上地址下载的文件中包含了MinGW它(内嵌了GCC编译器和gdb调试器)

如果您仅仅希望把Code::Blocks当作编辑器使用, 或者打算自己配置编译器和调试器的话, 可以下载不带MinGW的版本, 到下面的地址去下载。

<http://downloads.sourceforge.net/codeblocks/codeblocks-8.02-setup.exe>

本书的作者建议初学C/C++的朋友下载内置MinGW的版本, 这样不致于花费太多时间配置编译器和调试器, 从而把大部分时间用于学习调试和编写程序。待将来您熟悉了Code::Blocks, 再搭配高版本的MinGW或者其它编译器一起使用。

如果您使用Mac OS X 或Linux操作系统, 请参阅附录A中的安装说明。

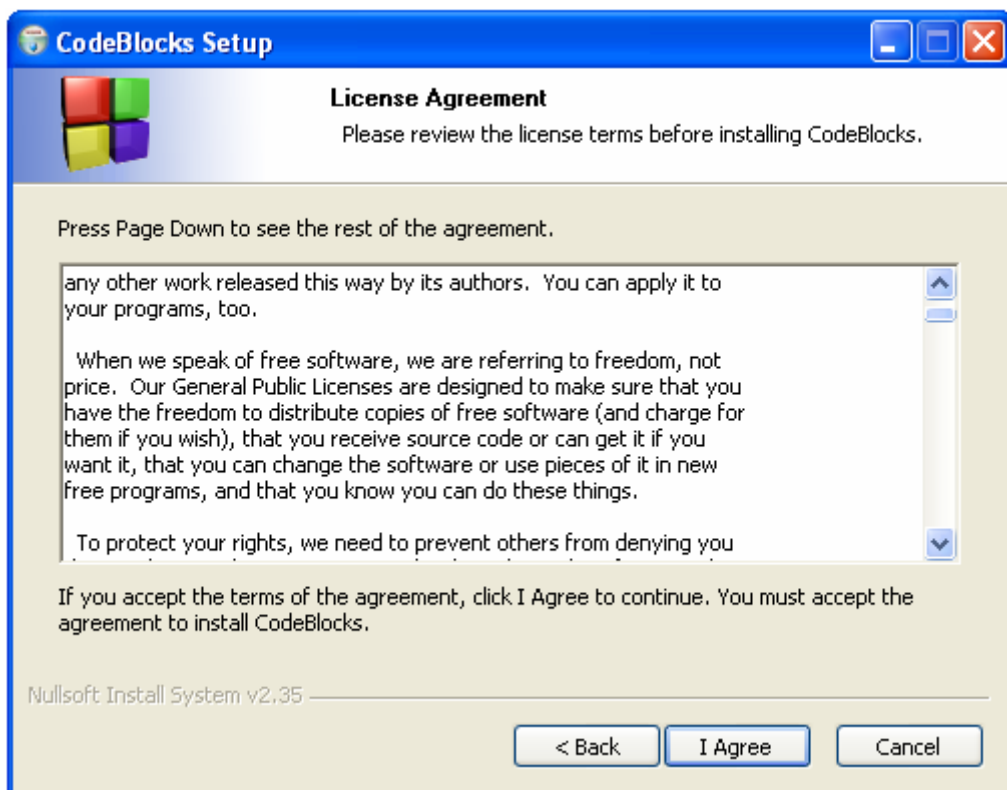
1.2 安装

安装过程可以参照如下步骤进行(以在笔者的英文版Windows XP Professional SP2操作系统上安装Code::Blocks8.02为例)。

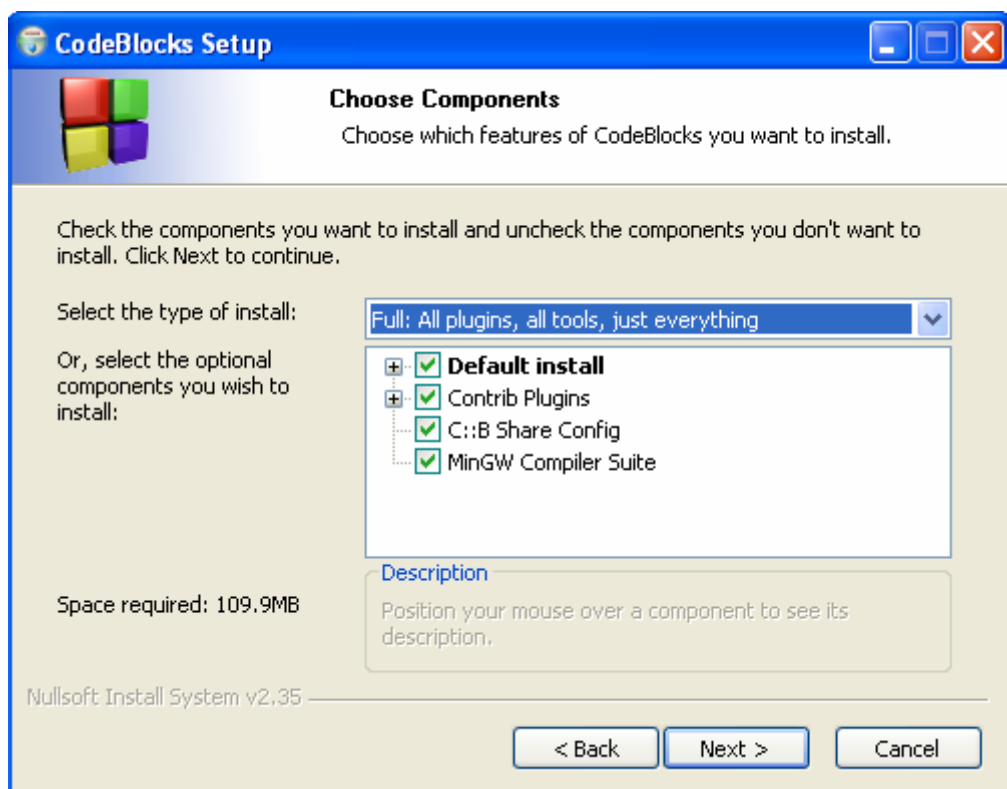
运行下载后的安装文件进入左下图界面。



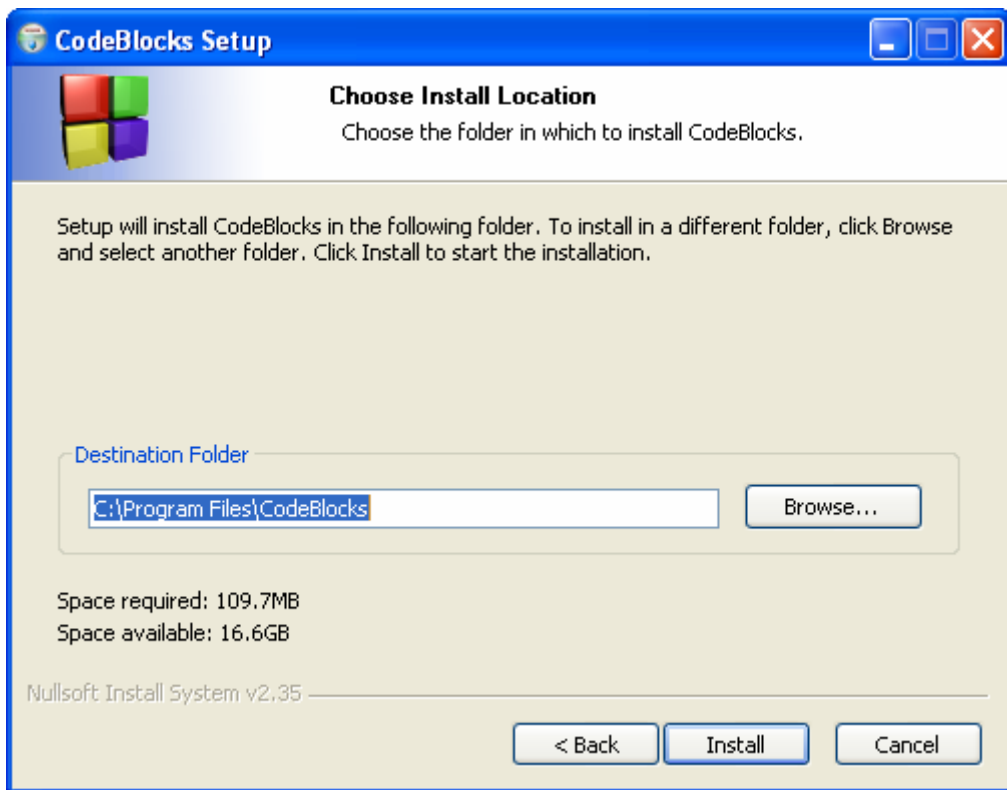
再用鼠标点击Next按钮，可以进入下图界面。



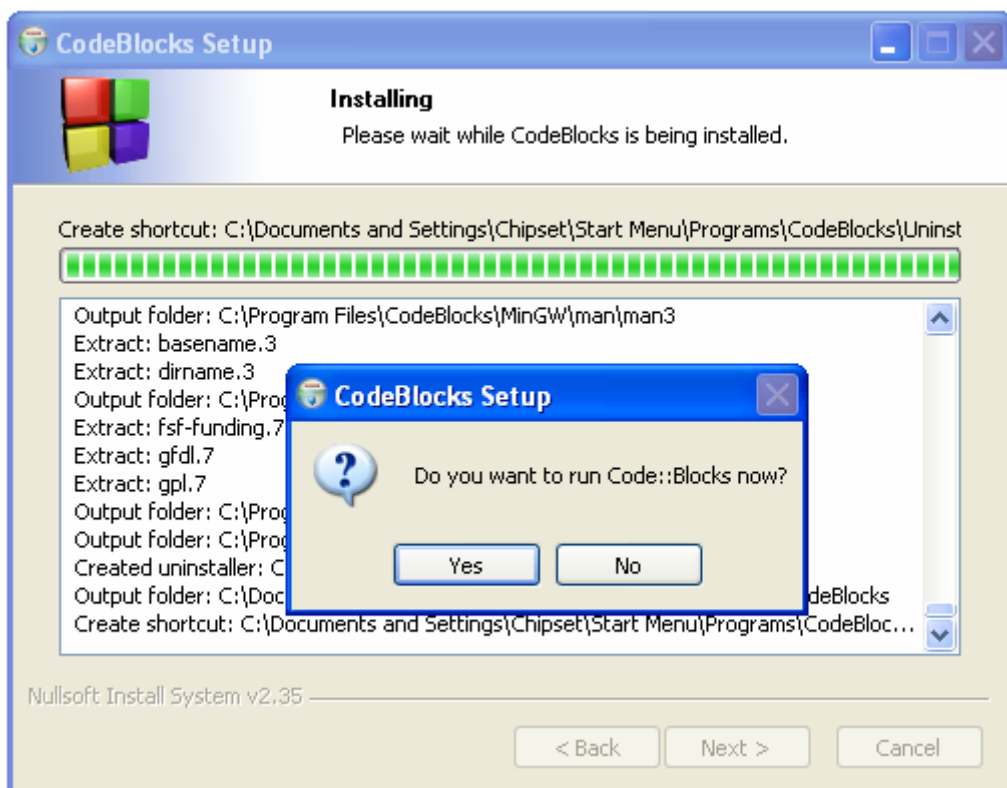
用鼠标选择 I Agree 按钮，进入如下图界面。



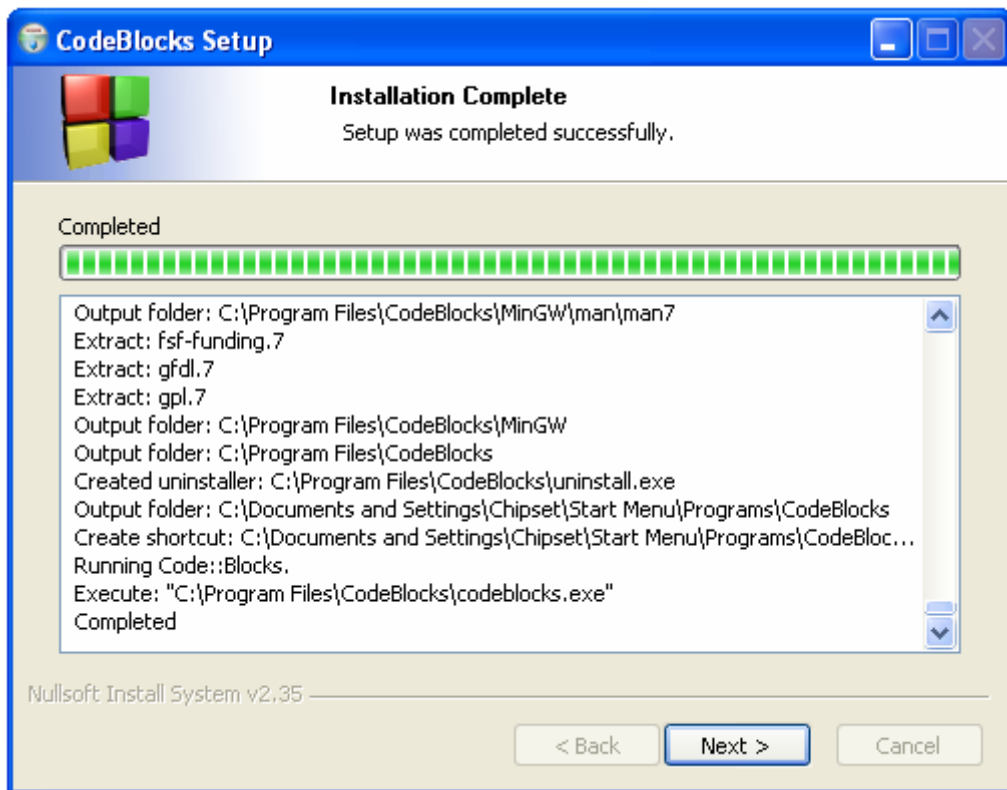
选择全部安装(Full: All plugins, all tools, just everything), 见上图, 再点击 Next 按钮, 进入一个新界面, 如下图。



点击 Browse...按钮选好安装路径(默认安装路径为 C:\Program Files\CodeBlocks), 用鼠标选择 Install 按钮, 可以看到安装过程正在进行, 并弹出一个对话框, 见下图。



用鼠标选择 No 按钮，则对话画框关闭，见下图。



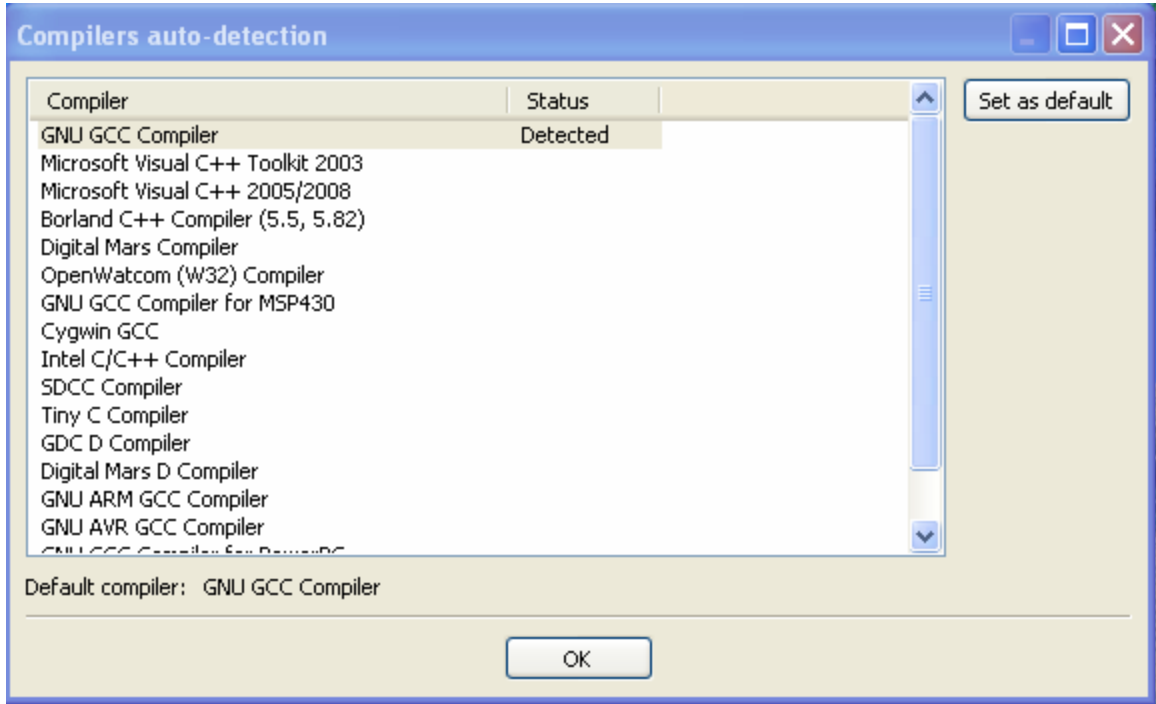
用鼠标选择 Next 按钮，进入界面见下图。



最后用鼠标选择 Finish，则安装过程就完成了。

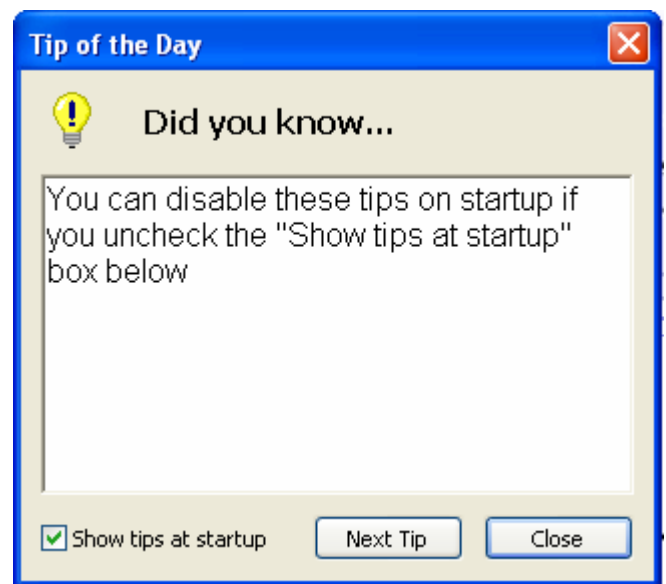
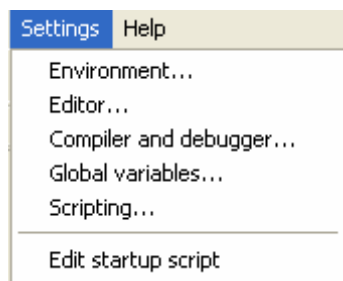
2. Code::Blocks 编程环境配置

第一次启动 Code::Blocks，可能会出现如下对话框，告诉您自动检测到 GNU GCC Compiler 编译器，用鼠标选择对话框右侧的 Set as default 按钮，然后再选择 OK 按钮，见下图。



假如您的 Code::Blocks 安装正确的话，接下来就进入 Code::Blocks 的主界面，但是会弹出一个标签为 Tips of the Day 的小对话框，见右下图。

把 Show tips at startup 前面的勾去掉，然后选择 Close，这样下次启动就不会再出现这个小对话框。



进入 Code::Blocks 主界面，选择主菜单 Settings，弹出一个窗口，见左上图。然后我们就可以分别对环境(Environment...)，编辑器(Editor...)，编译器和调试器(Compiler and debugger...)三个子菜单进行配置了。

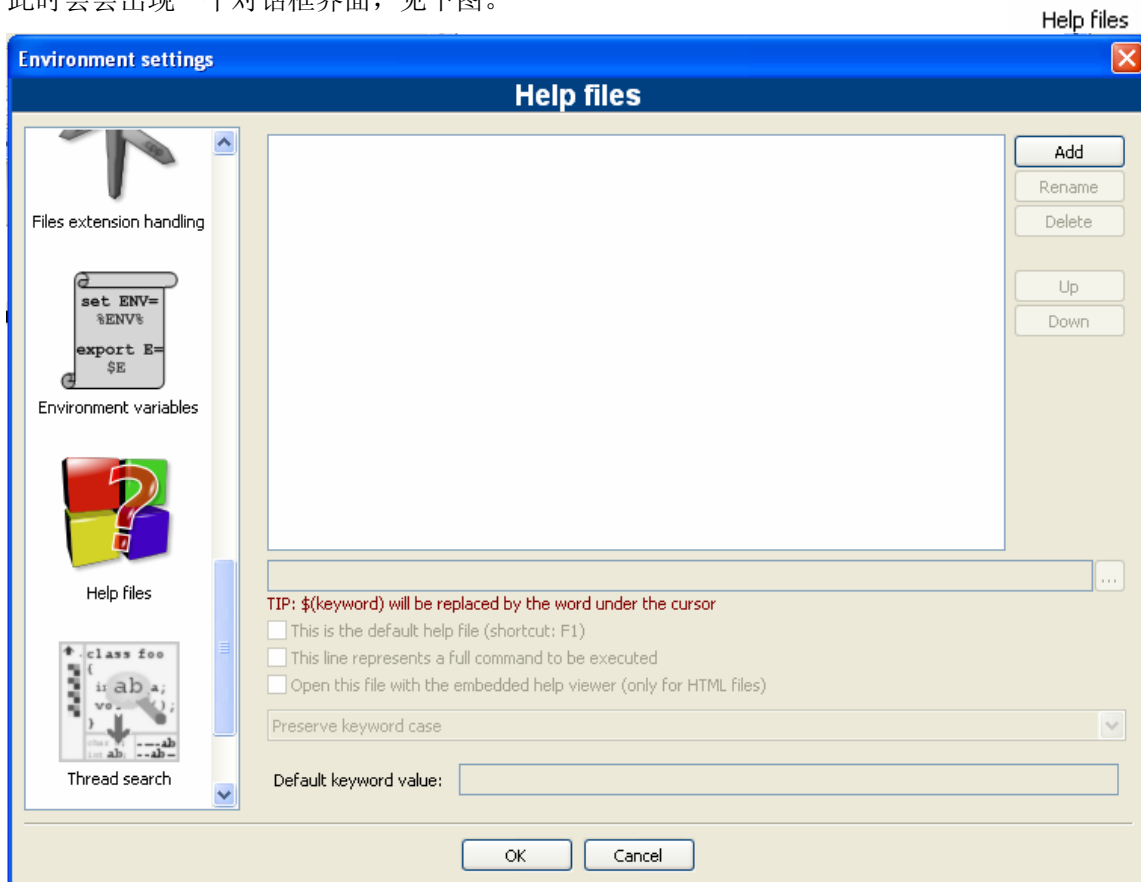
2.1 环境

选择主菜单 Settings 下的第一个子菜单 Environment...,会弹出一个窗口,用鼠标拖动左侧的滚动条,可以见到很多带有文字的图标。这些下面带有文字的图标代表了不同的功能按钮。

2.1.1 配置帮助文件

拖动滚动条,用鼠标选择这个图标,见右图。

此时会出现一个对话框界面,见下图。



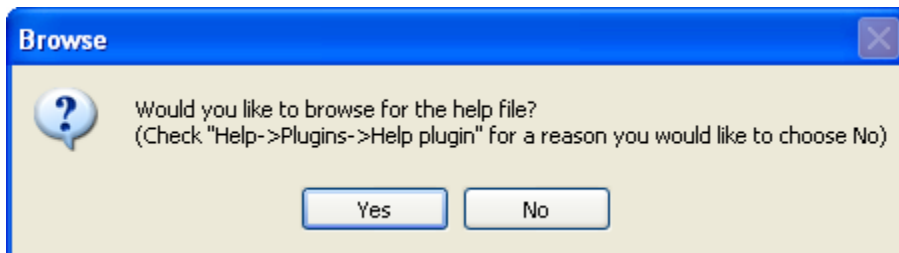
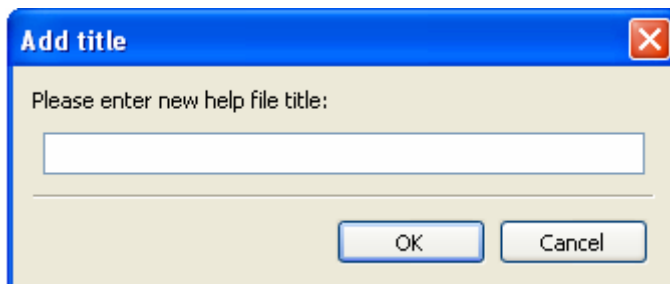
然后可以添加一些我们可能需要的帮助文件。我们编写基本的C/C++应用程序,仅需要知道C/C++的库函数用法就可以了。如果您没有C/C++语言库函数的文档,请到<http://www.cppblog.com/Files/Chipset/cppreference.zip>去下载C++ Reference,解压后放到Code::Blocks目录下(也可以放到别处),以便添加进来编程时方便查阅。可以按照如下步骤进行添加:

(1) 添加文件

用鼠标点击右上侧的Add按钮,得到对话框见右图。

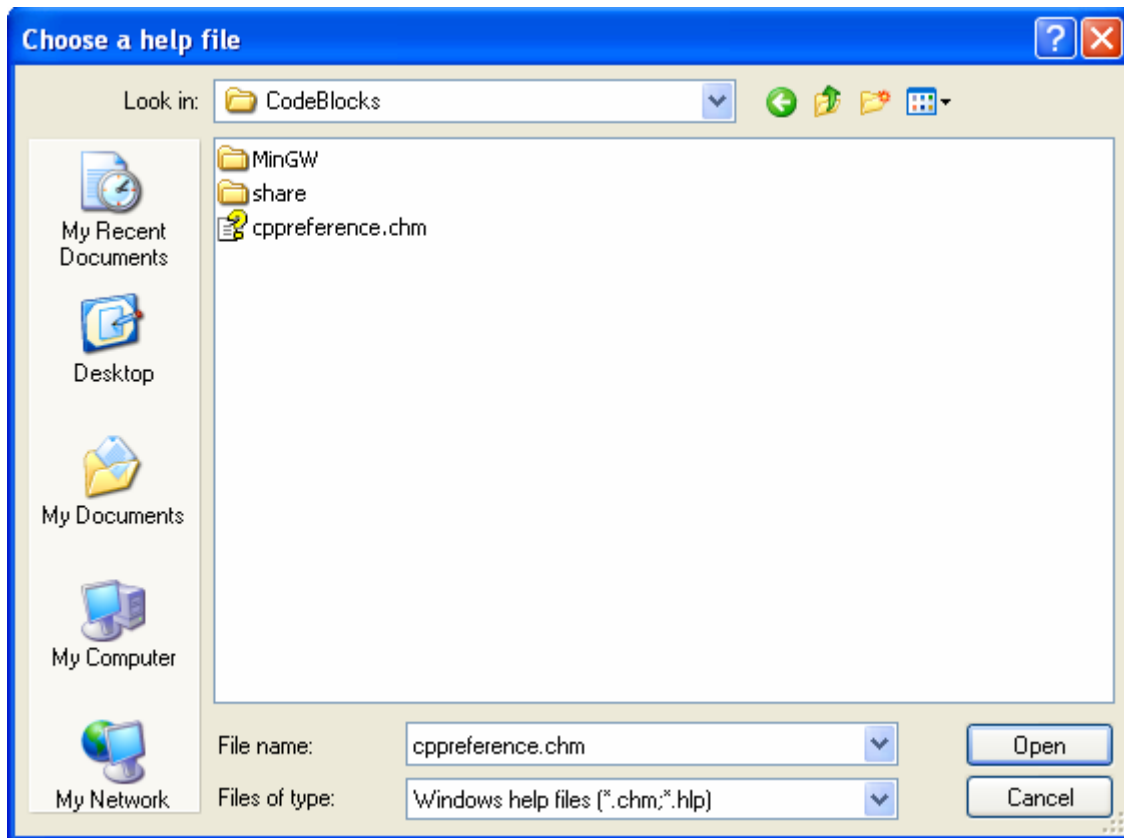
(2) 键入帮助文件题头

给添加的文件取一个题头名,该名字可以跟实际文件名相同,也可以不同,然后选择OK按钮,又弹出一个对话框见



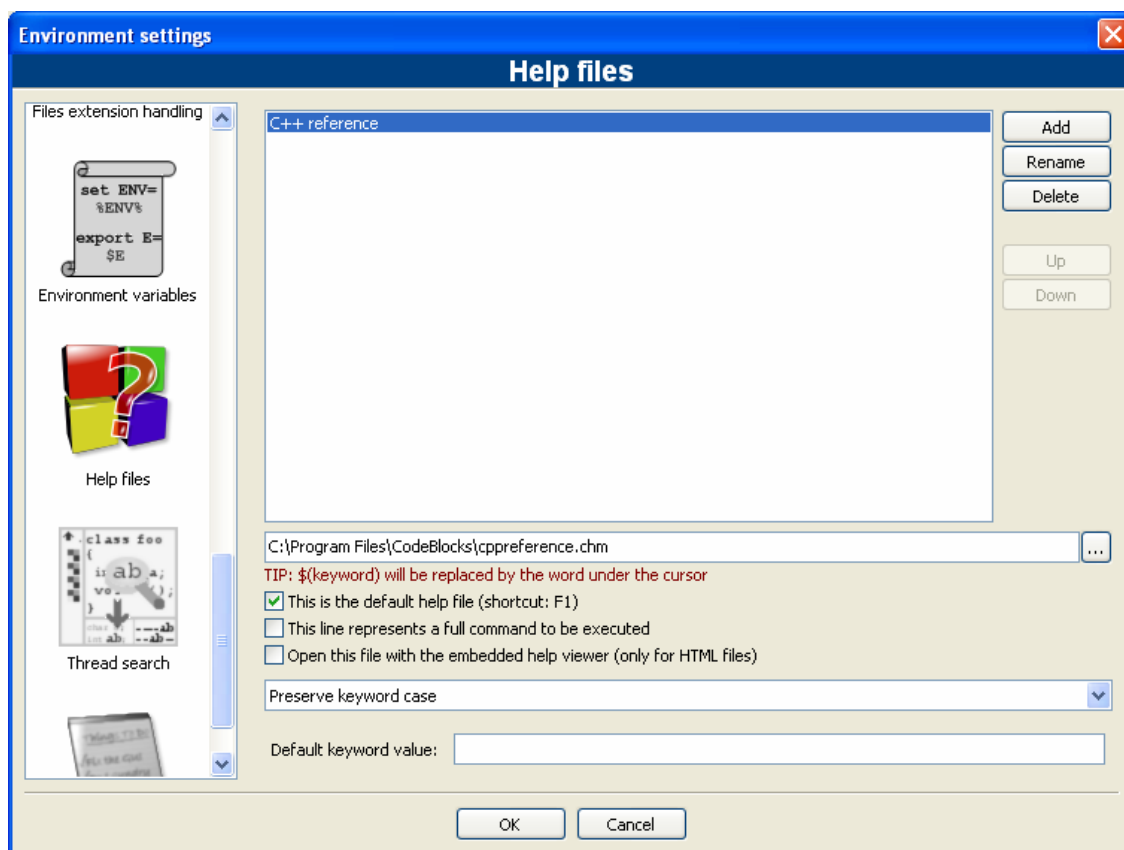
右图。

选择Yes按钮，进入下一步，见下图。



(3) 选择需要打开的文件

找到帮助文件的路径，选中帮助文件cppreference.chm，然后选择Open就又回到了刚进入Help files的对话框，只不过多了一行字C++ Reference，见下图。



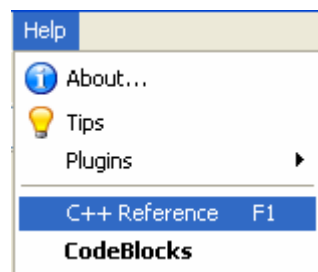
并且有刚加载的文件cppreference.chm的对应路径。可以继续按照上述步骤添加更多帮助文件，也可以用右上侧的按钮Rename对题头C++ Reference进行改名或者用Delete按钮删除此题头。

(4) 使帮助文件可用

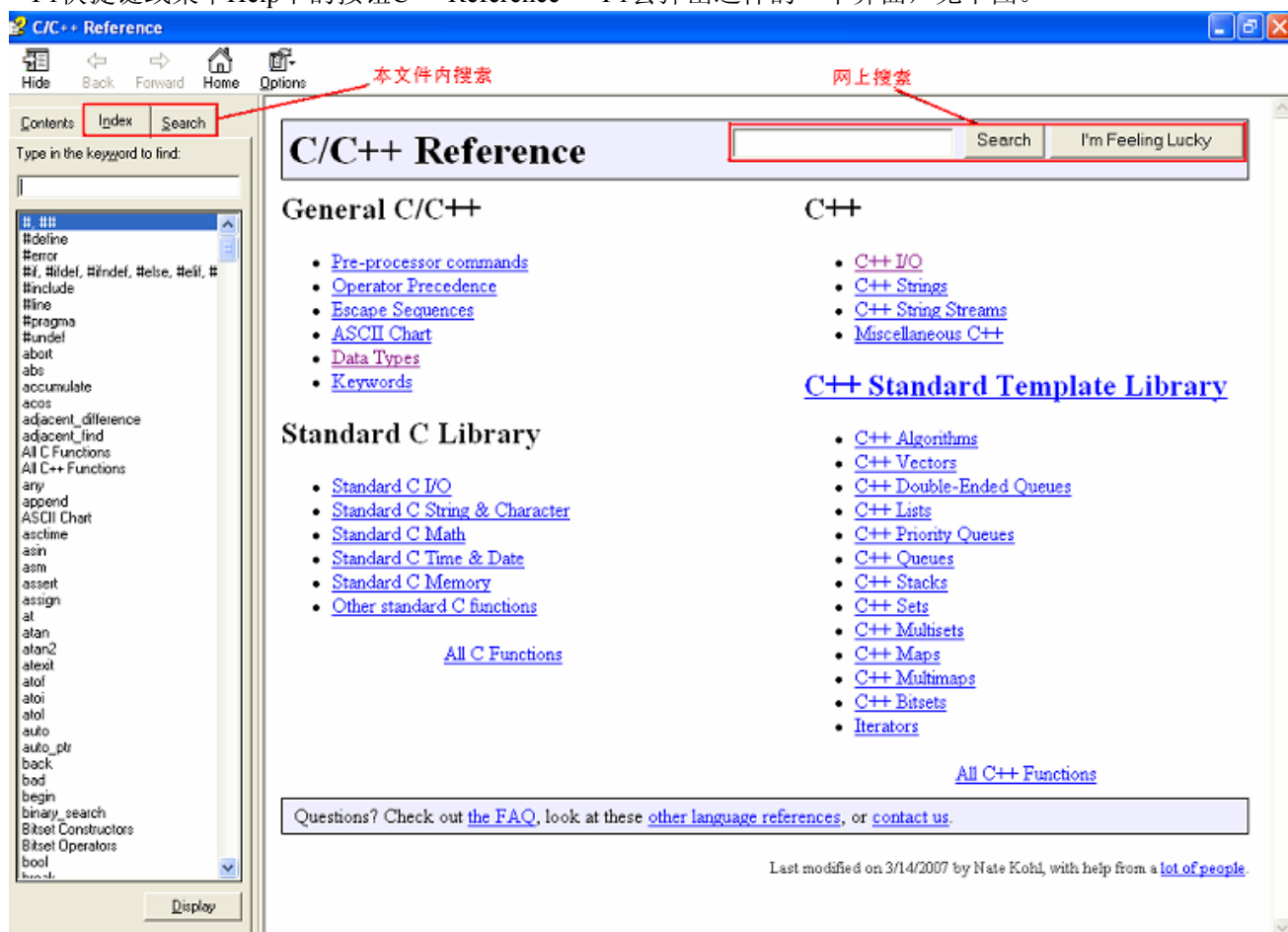
为了方便使用，选中C++ Reference并用鼠标在下面的标签This is the default help file (shortcut: F1)前面的小方框中打勾，见上图，然后再用鼠标点击下面的OK按钮。

(5) 测试帮助文件是否可以成功加载

进入Code::Blocks(如果刚才您未退出Code::Blocks就无需再重新进入)，选择主菜单Help下的C++ Reference F1按钮，如右图。或者按下F1快捷键，就可以成功加载刚才设置需要加载的帮助文件cppreference.chm了。



经过上述这些设置后，Code::Blocks就可以成功加载帮助文件了，按下F1快捷键或菜单Help下的按钮C++ Reference F1会弹出这样的一个界面，见下图。



假如我们需要查阅标准的C++库函数，可以选择左侧的Index或Search按钮键入函数名进行查询。如果您使用的电脑已经联网，在左上部空框内键入要查询的函数名，再用鼠标点击右侧Search按钮就可以进行http://www.cppreference.com网上查询。

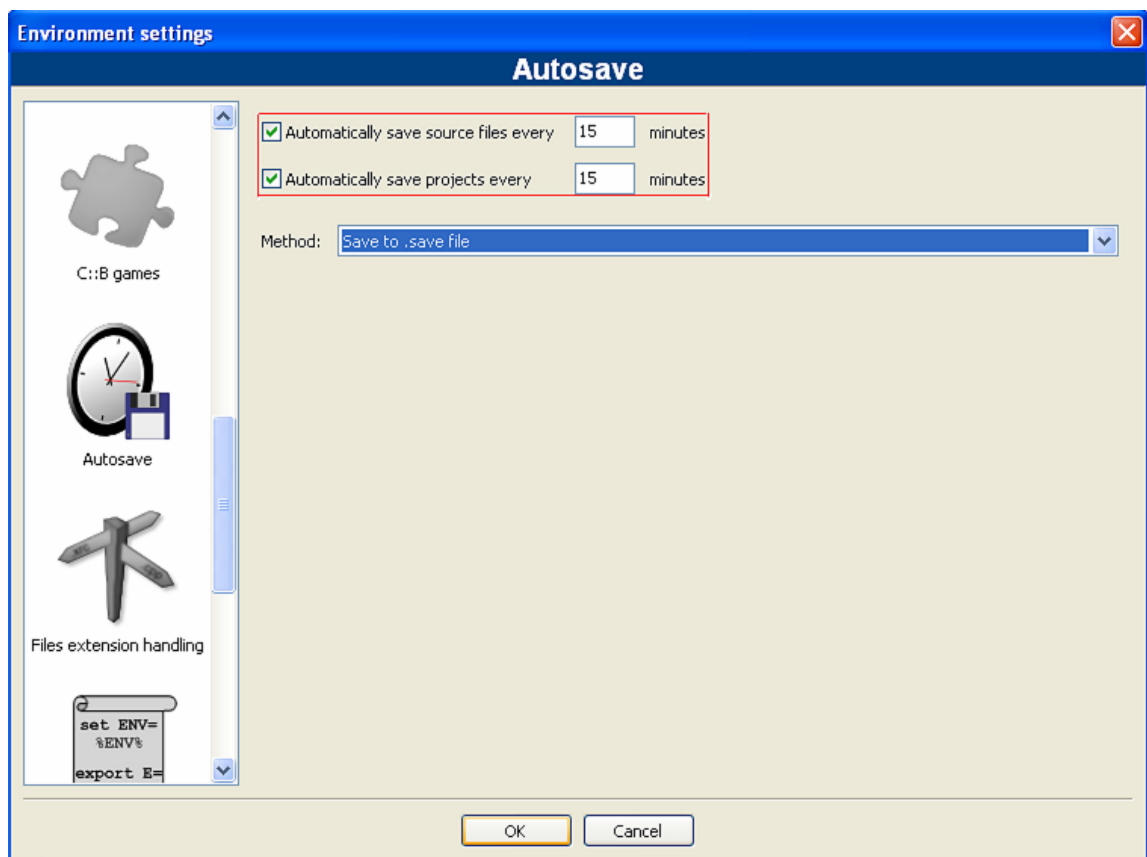
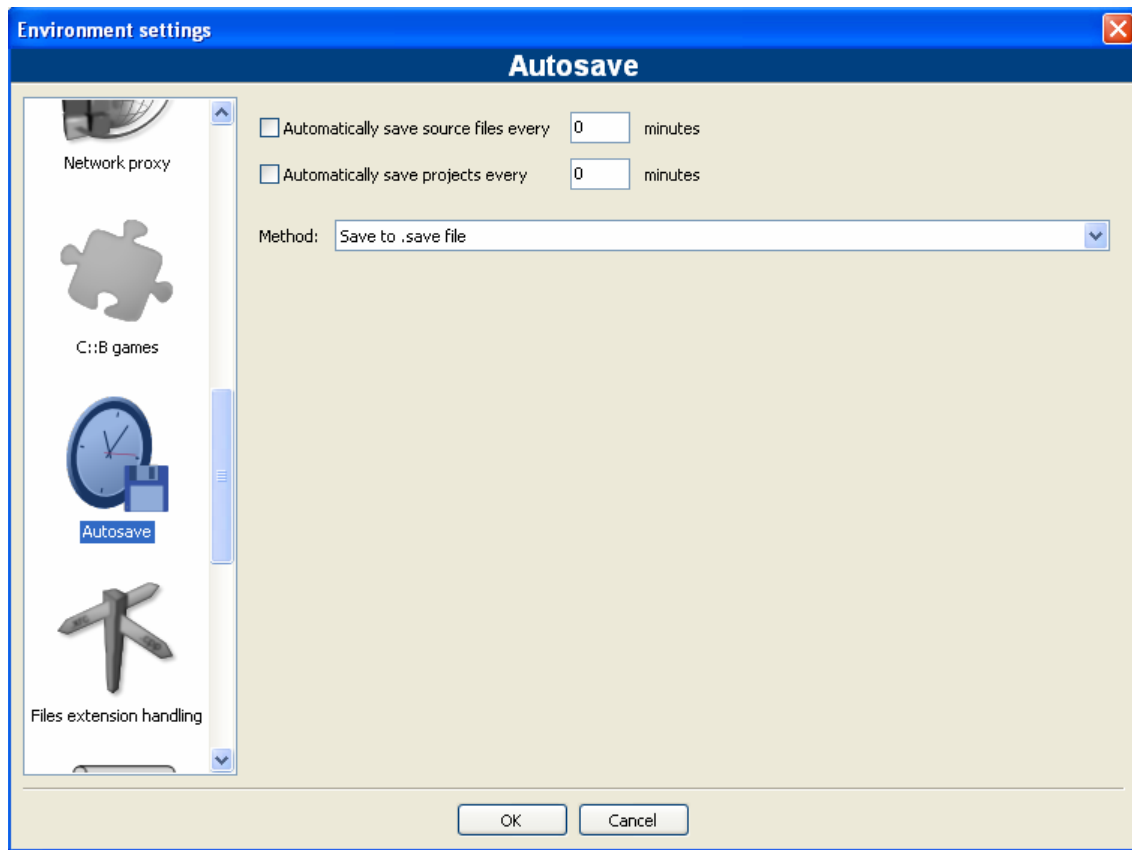
2.1.2 自动保存

编写或者调试程序的过程中偶尔出现断电，如果没有后被电源，此时可能会丢失部分程序内容。为

此，我们需要设置Code::Blocks能自动保存功能所对应的选项。

进入Code::Blocks后，选择主菜单Settings下Environment...子菜单，弹出一个对话框，用鼠标拖动左侧的滚动条，找到如右图标。

选中它，界面如下图。



分别设置自动保存源文件和工程的时间，例如均为15分钟，见上图中红色框中部分。

Method为保存文件的方法，有三种，分别是Create backup and save to original file, Save to original file, 以及Save to .save file。选择最后一个，Save to .save file就可以了，设置完毕后，选择OK按钮。

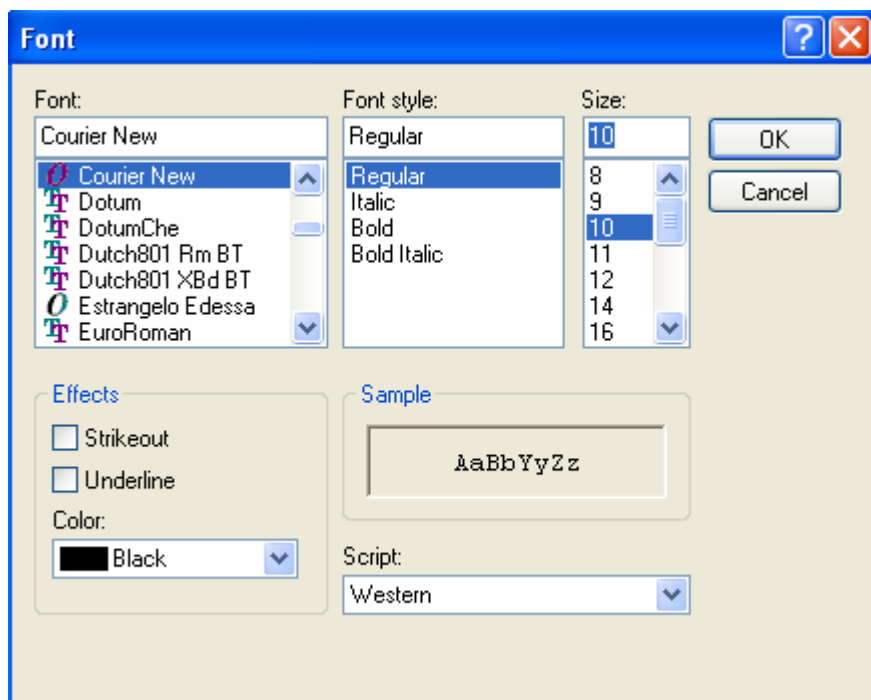
2.2 编辑器

编辑器主要用来编辑程序的源代码，Code::Blocks内嵌的编辑器界面友好，功能比较完备，操作也很简单。

2.2.1 通用设置

启动Code::Blocks，选择主菜单Settings下的子菜单Editor...会弹出一个对话框，默认通用设置General settings栏目，选中一些选项如右下图。

然后设置字体，字体设置首先选择右上角的 Choose 按钮，会弹出一个对话框，对话框主要有三个竖向栏目，最左侧的栏目Font:用来选择字体类型，选择Courier New，中间栏目Font style: 是字体样式，选择Regular，最右边的栏目Size:是文字大小，根据个人习惯和电脑显示器显示面积大小进行选择，一般10~12，其它选项不变。见右图。



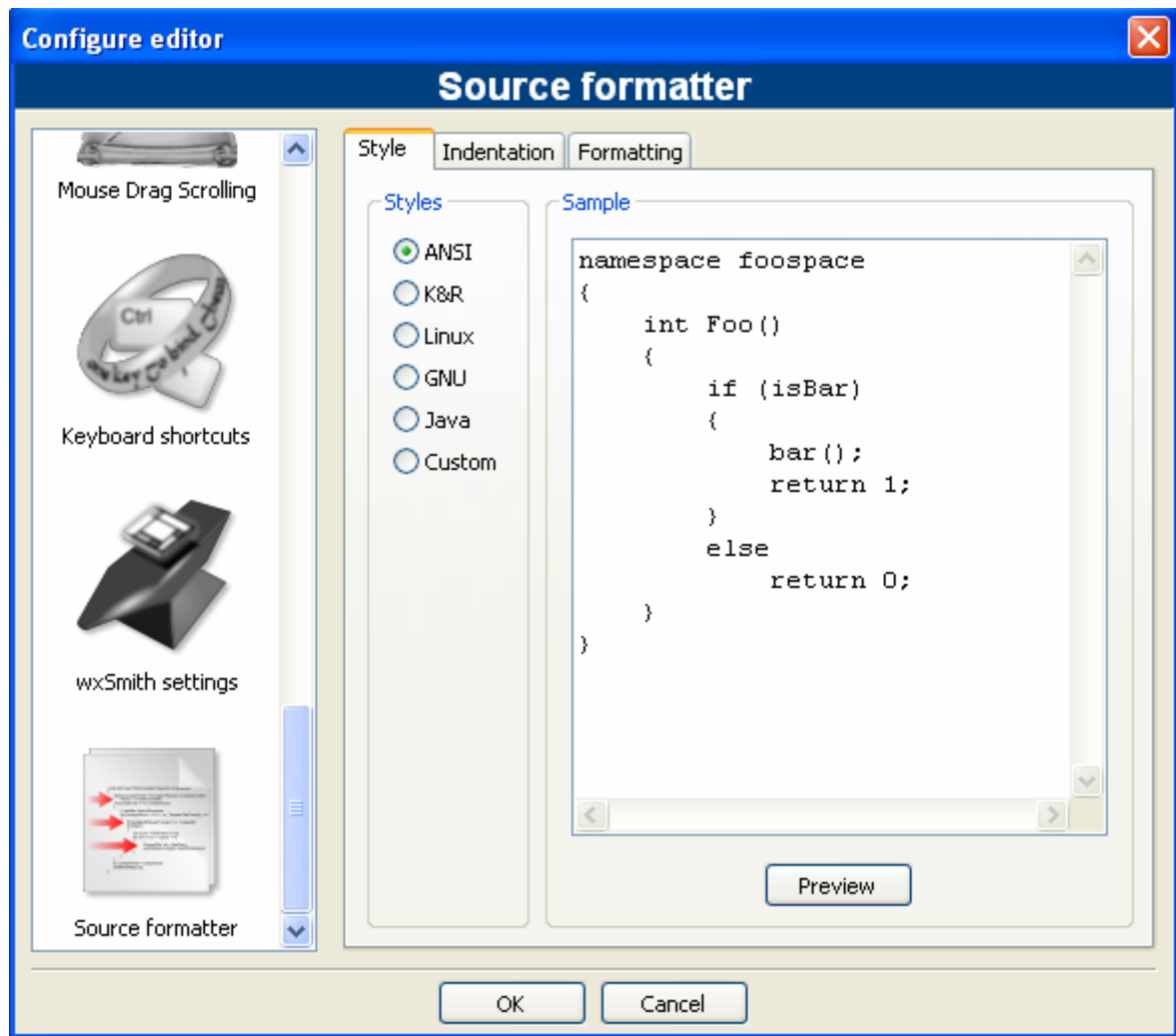
然后用鼠标选择OK，则字体参数设置完毕，进入上一级对话框General settings，再选择OK，则General settings设置完毕，回到Code::Blocks主界面。

2.2.2 源代码格式

不同的人编写代码风格不同，Code::Blocks提供了几种代码的书写格式。首先从Settings主菜单进入子菜单Editor...，然后从弹出的对话框中移动滚动条，找到标签为Source formatter的按钮，见右图。选中它，可以看到右侧Style菜单下有几种风格分别为ANSI, K&R, Linux, GNU, Java, Custom, 最右侧则是这些风格的代码预览Preview。可以根据个人习惯进行选择，如果选择Custom则需要自己设置两个子菜单Indentation和Formatting下的各个选项，选中自己习惯或者喜欢的风格(笔者的习惯是用ANSI)，然后点击OK按钮。



Source formatter



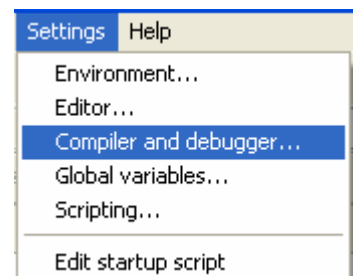
如此以来，编辑器的基本设置就完成了，尽管还有很多其它的选项和参数，但是并不太常用，因此这里就不做详细介绍了。

2.3 编译器和调试器

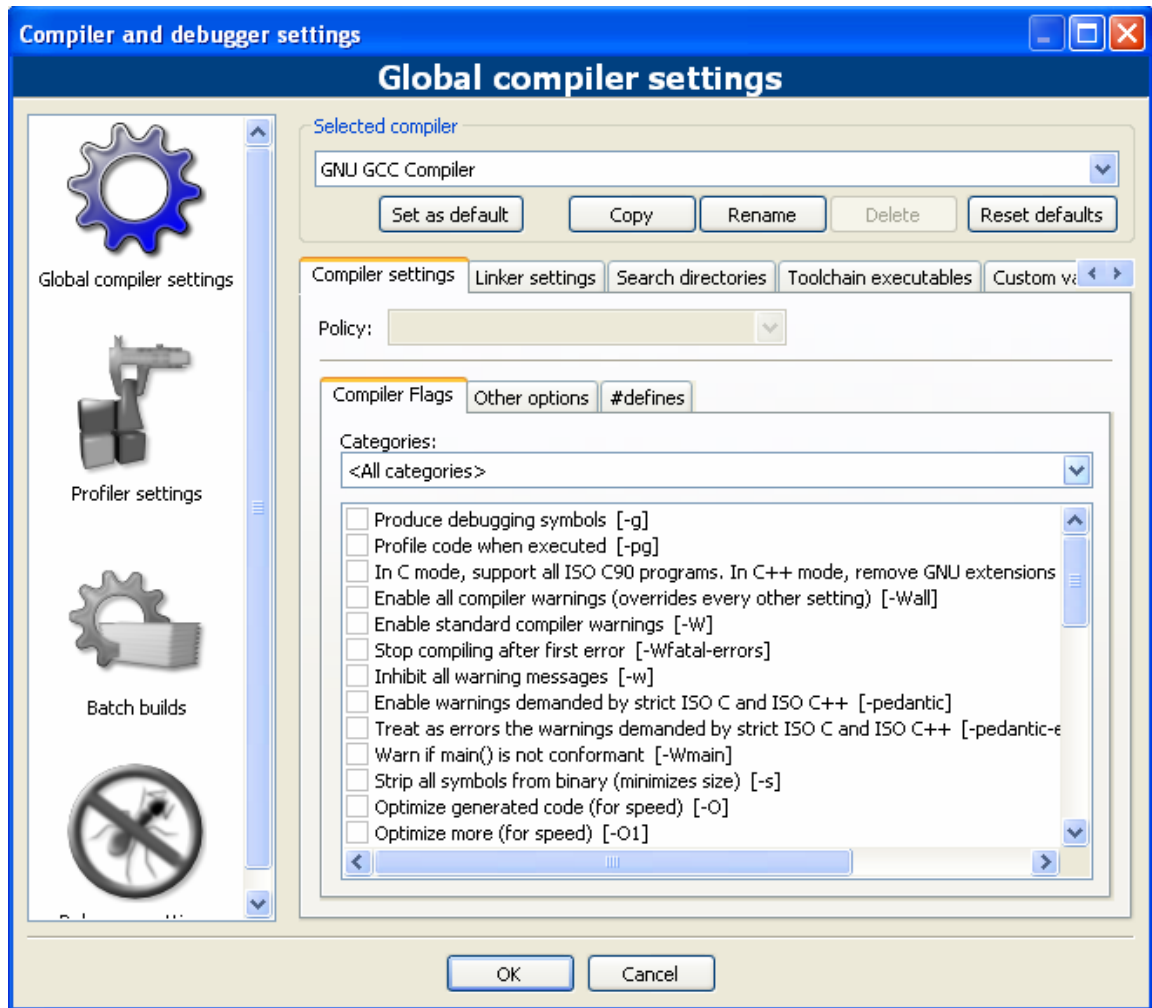
编译器和调试器可能最重要，因为编写的源代码需要转换成机器可以识别的二进制才能执行，而且编写程序时，很难保证一次正确。这里主要讲述一下，编译器和调试器的全局配置，也就是说这里配置的每个选项都会影响到将来建立的每个工程。

2.3.1 进入编译器和调试器配置界面

首先从Code::Blocks界面的主菜单Settings下拉菜单下选择Compiler and debugger...按钮，见右图。

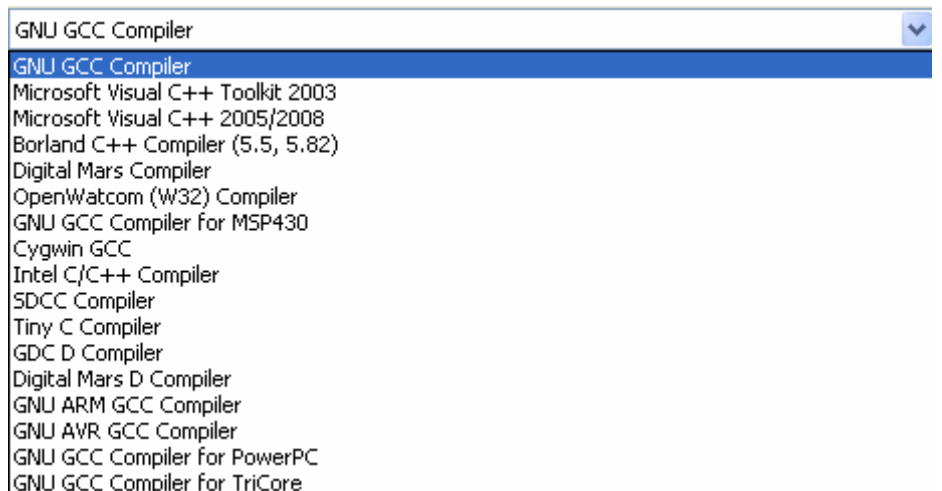


会弹出一个对话框如下，这就是编译器和调试器的配置界面，见下图。



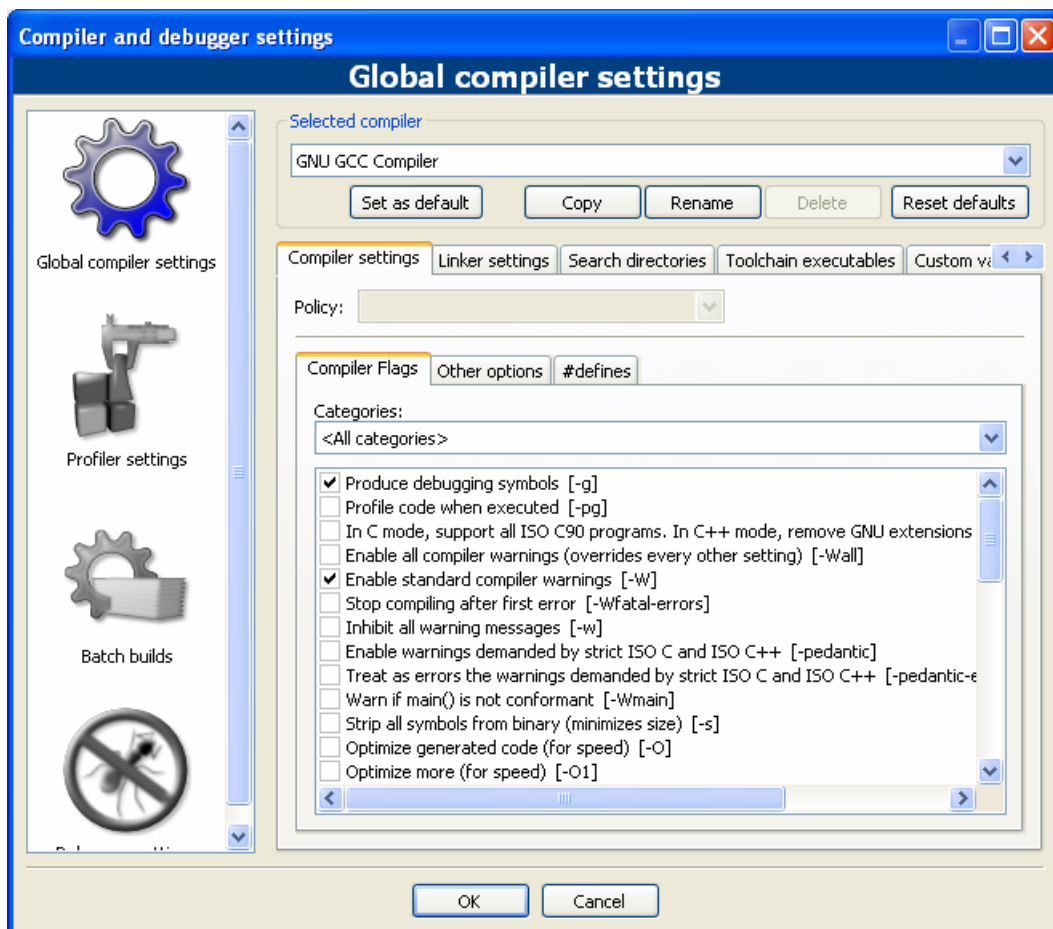
2.3.2 编译器选择

Code::Blocks支持多种编译器，默认编译器GNU GCC Compiler，当然，您也可以选择其它的编译器，只不过需要事先安装好您想用的编译器，见右图。

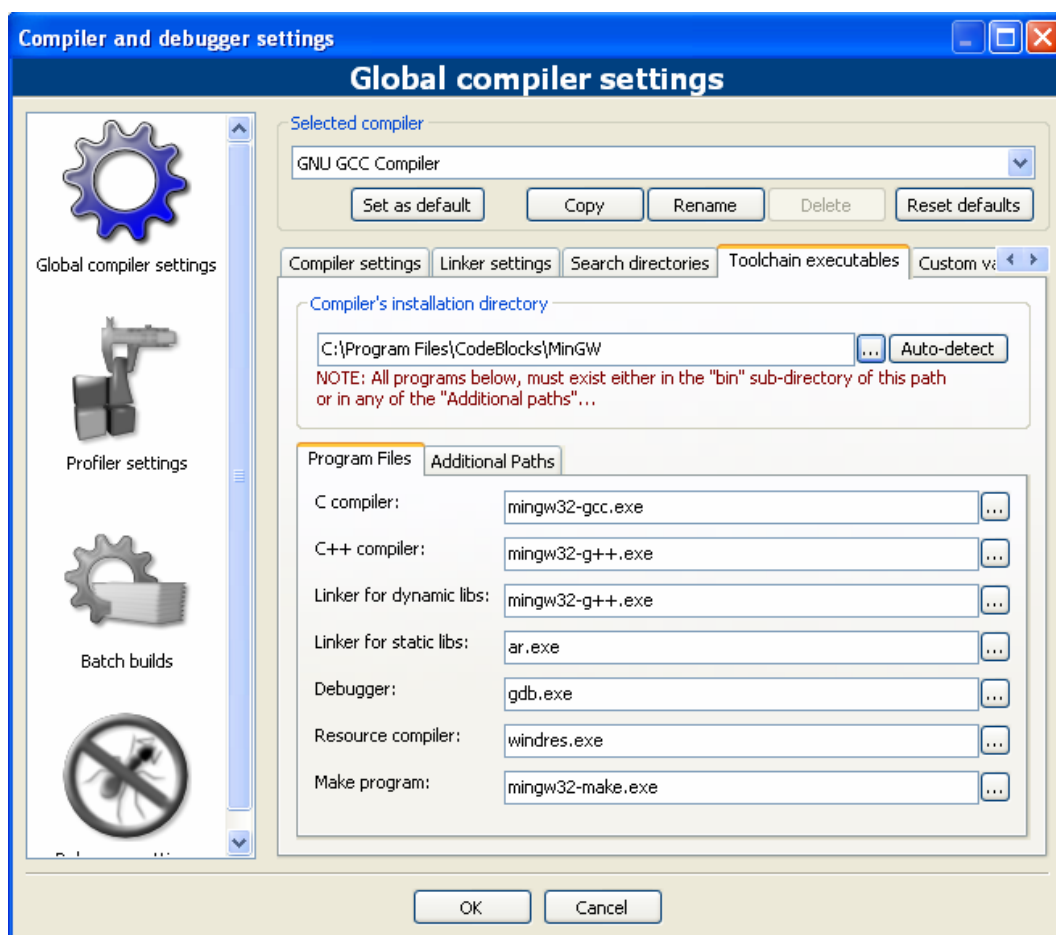


2.3.3 扩展编译选项配置


选择Compiler Settings菜单下的Compiler Flags子菜单，选中其中两个选项，Produce debugging symbols [-g]和Enable standard compiler warnings [-W]，也可以什么都不选，见下图。



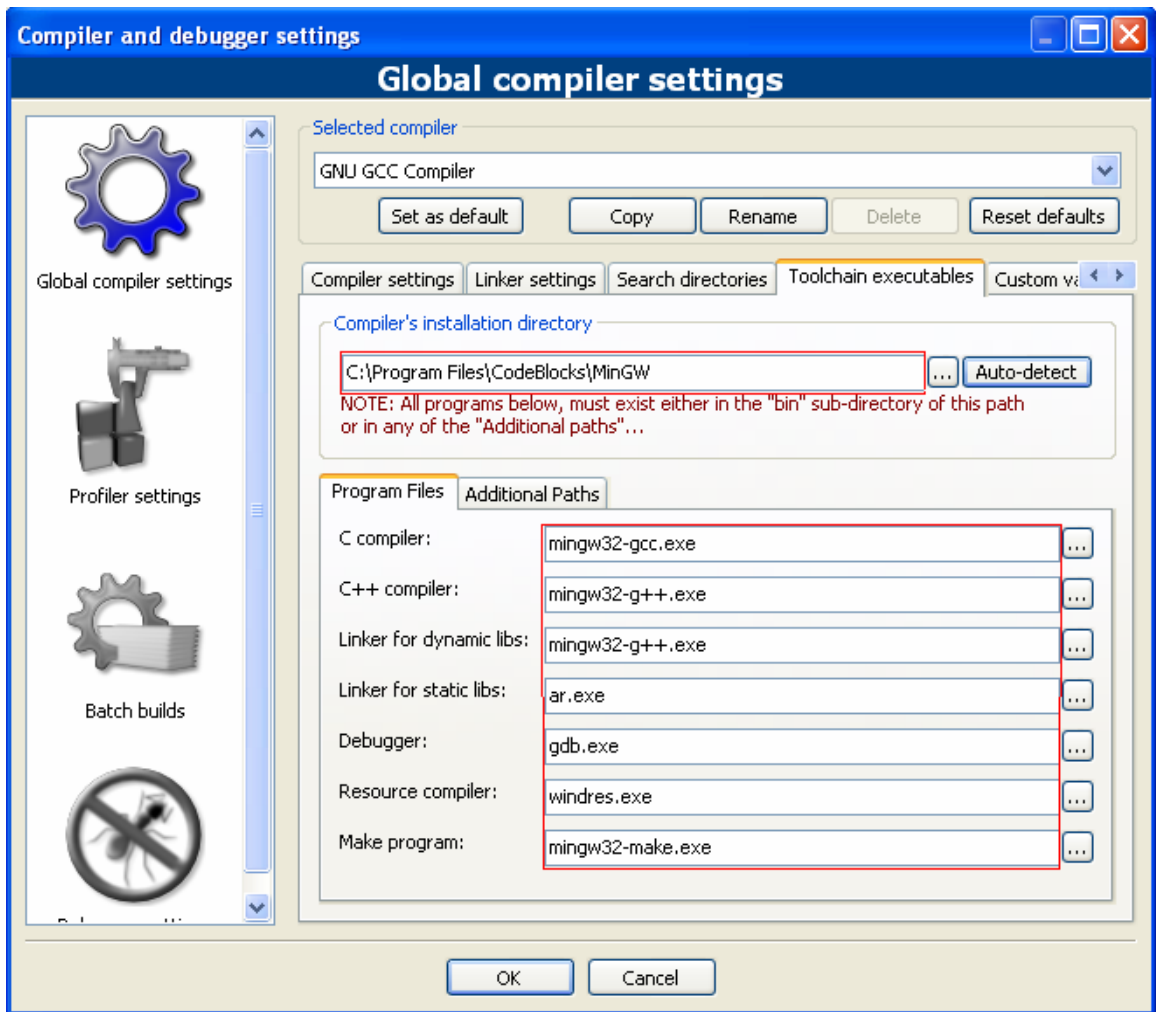
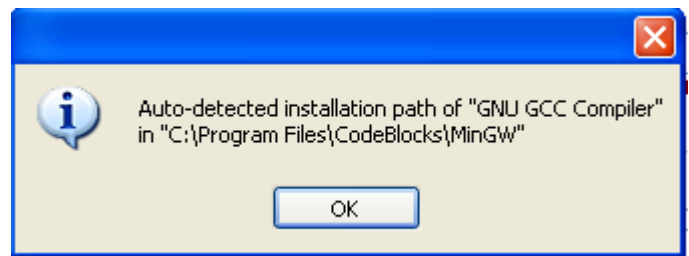
然后选择Toolchain executables子菜单，会出现一个界面，见下图。



点击右侧的Auto-detect按钮，一般而言能自动识别编译器的安装路径，见右图。

如果不能自动识别编译器安装的路径，就需要点击按钮进行手工配置好该路径。并且也要配置好C compiler:, C++ compiler:,

Linker for dynamic libs:, Linker for static libs:, Debugger:, Resource compiler:, Make program:这几个选项的文件名。见见下图中用红色方框框起来的部分。



最后用鼠标点击最下方的OK按钮，则编译器和调试器基本配置完毕。



3. 编写程序

下载安装Code::Blocks以后，您就可以编写代码了。Code::Blocks创建一个工作空间(workspace)跟踪您当前的工程(project)。如果有必要，您还可以在您当前的工作空间创建多个工程。一个工程就是一个或者多个源文件(包括头文件)的集合。源文件(source file) 就是您程序中包含源代码的文件，如果您正在编写C++程序，您就正在编写C++源代码(文件后缀名为.cpp)。您创建库文件(library files, 文件后缀名为.h或.hpp)时，会用到头文件(header file) 。一个库(library)是为了实现特定目标的函数集合，例如数学运算。

创建一个工程可以方便的把相关文件组织在一起。一个工程刚建立时，一般仅仅包含一个源文件。但是，伴随着您编程经验的增长可能会用到更复杂的工程，此时一个工程可能包含很多源文件以及头文件。

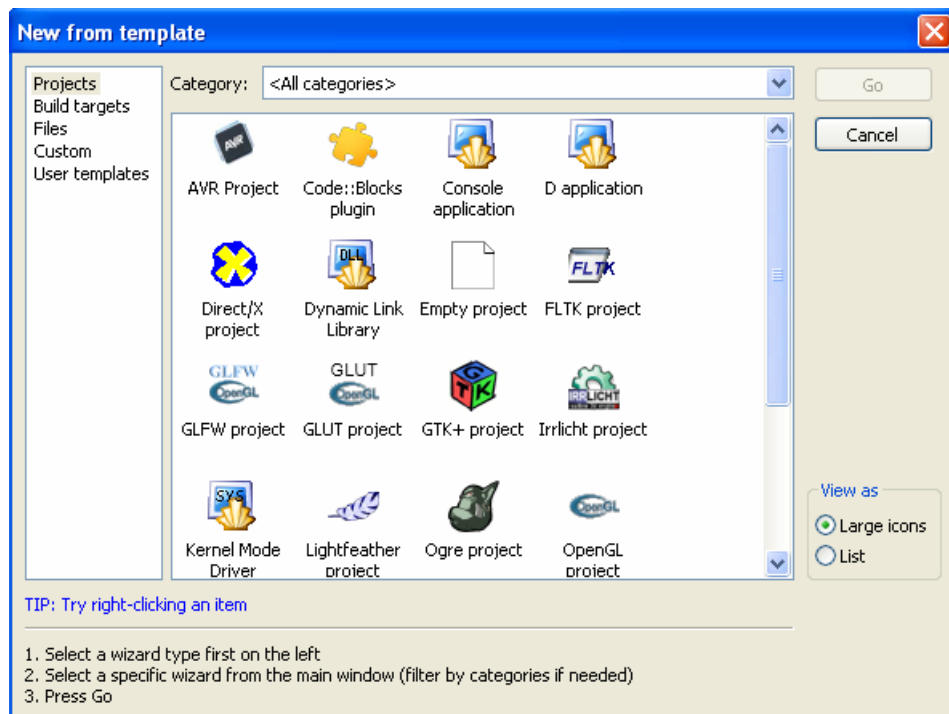
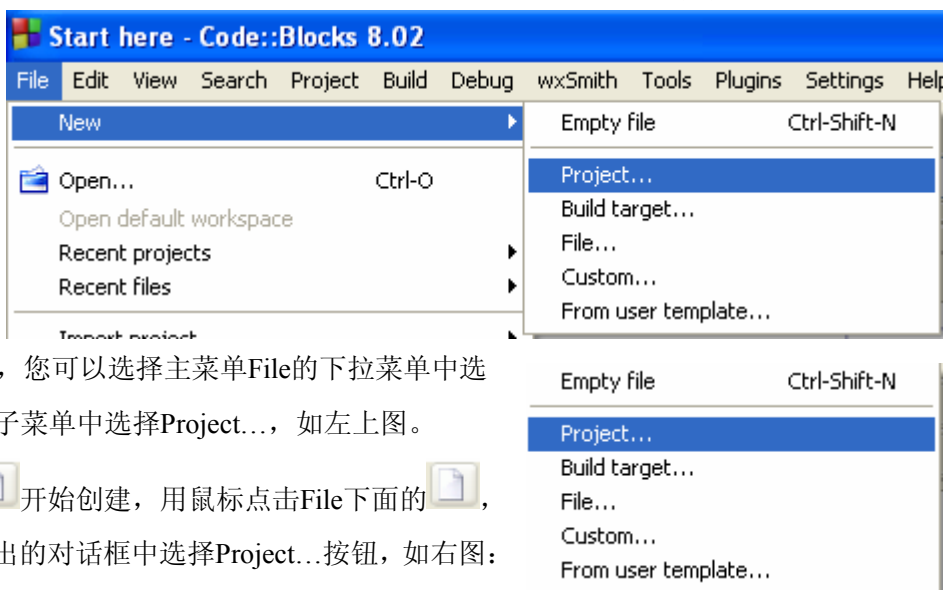
3.1 创建一个工程

创建工程的方法很多，您可以选择主菜单File的下拉菜单中选择二级菜单New，然后从子菜单中选择Project...，如左上图。

也可以从图标按钮开始创建，用鼠标点击File下面的，会弹出一个对话框，从弹出的对话框中选择Project...按钮，如右图：

您还可以从Code::Blocks主界面中选择Create a new project按钮进行创建，见右面Create a new project的图标。

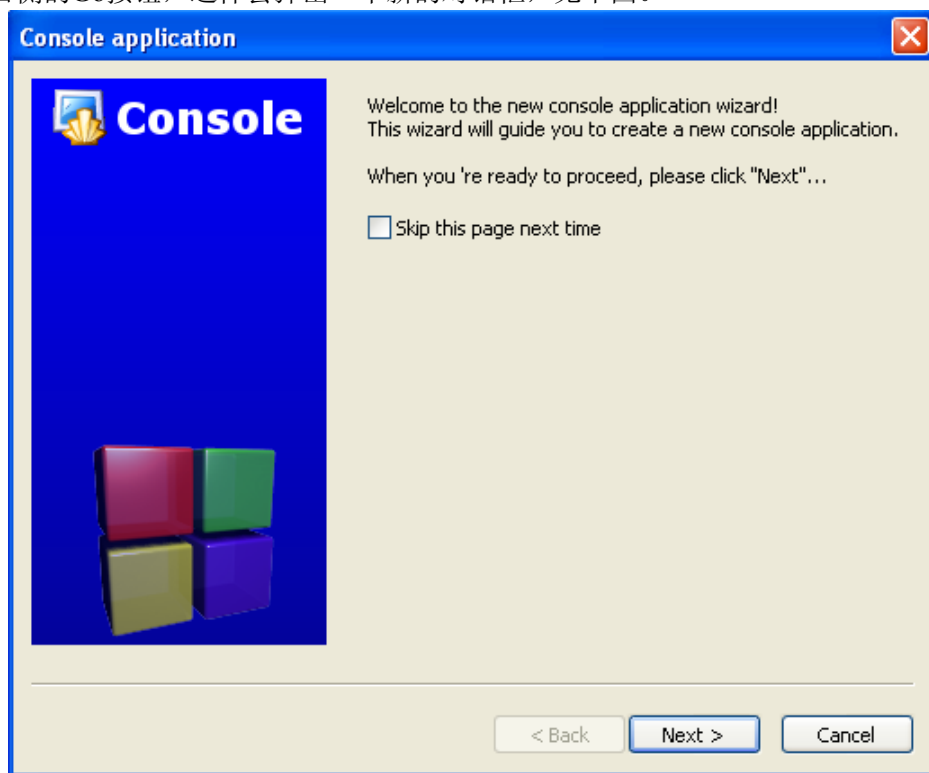
无论使用哪种方式创建一个工程，都会打开一个对话框，见下图。



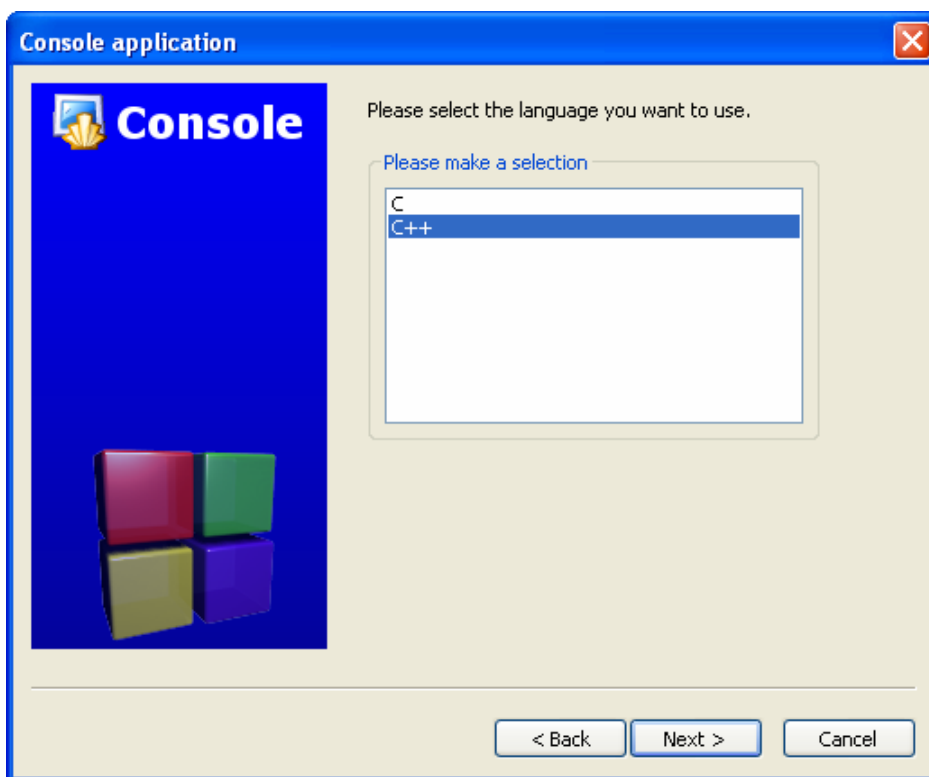
这个窗口中含有很多带有标签的图标，代表不同种类的工程。我们最常用的可能是 Console application，用来编写控制台应用程序。其它的是一些更高级的应用。

用鼠标选中带有控制台应用(Console application)标签的图标，见右图。

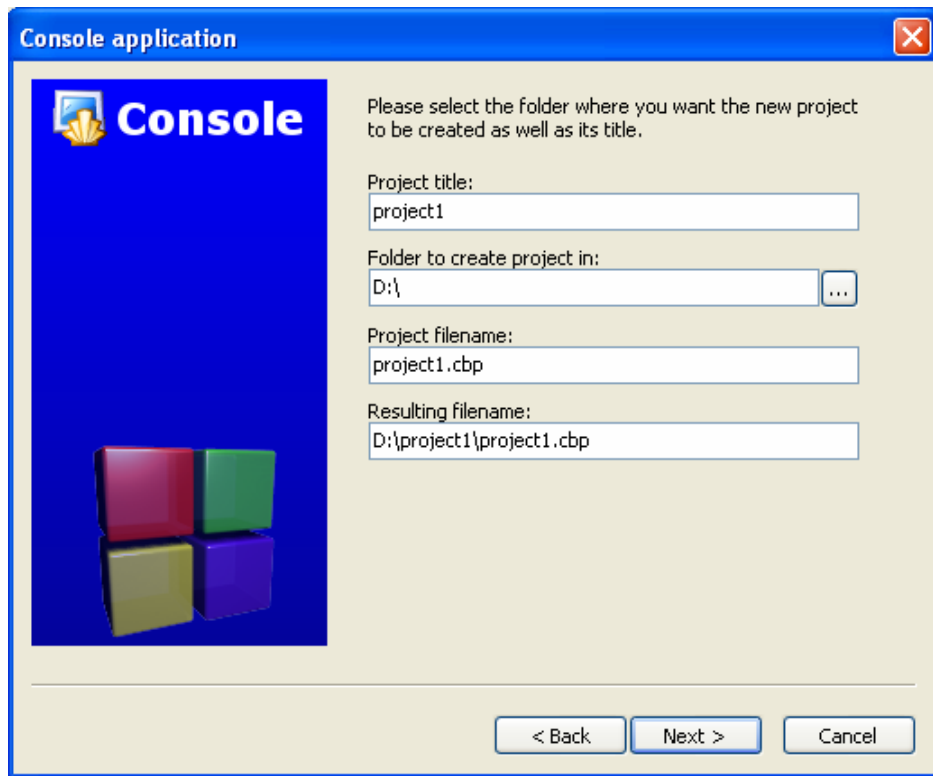
再选择右侧的Go按钮，这样会弹出一个新的对话框，见下图。



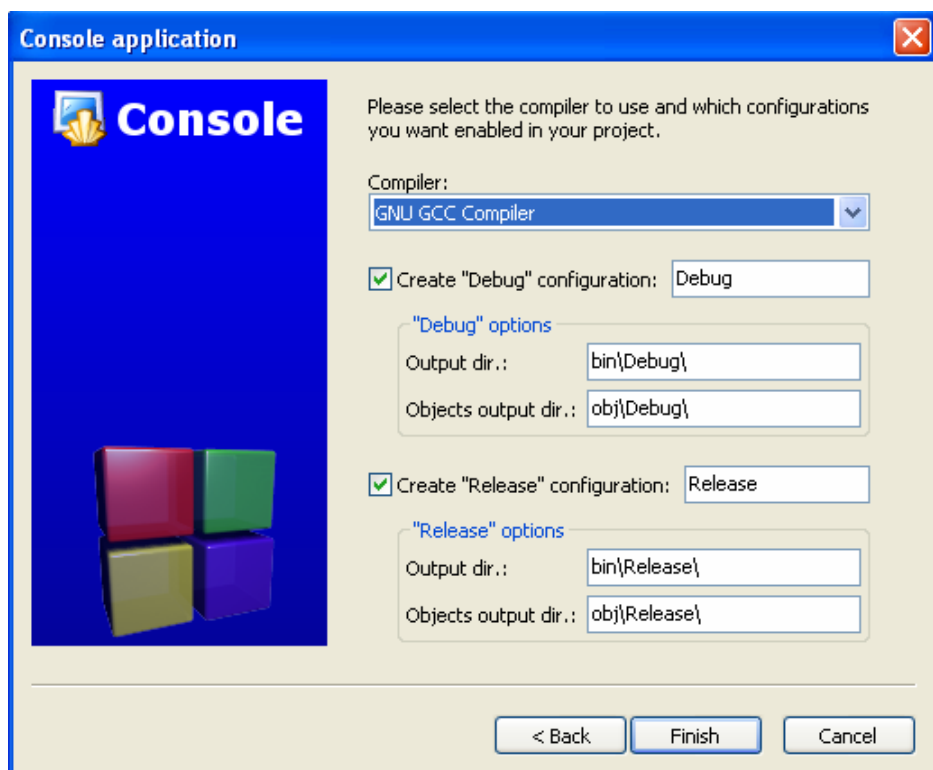
再选择Next按钮进入下一步，弹出一个对话框，见下图。





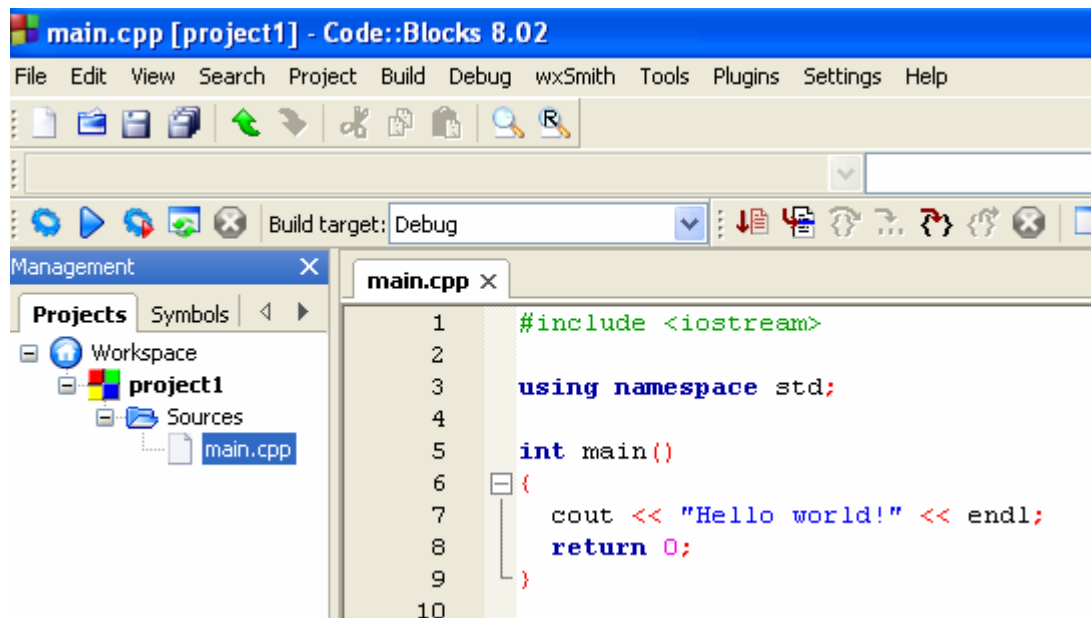
在弹出的对话框中有C和C++两个选项，选择C++表示编写C++控制台应用程序，选择C表示编写C控制台应用程序。这里以编写C++程序为例，因此选择C++，见上图。接下来选择下方的Next按钮进入下一步，又弹出一个对话框，见下图。



弹出的对话框中有4个需要填写文字的地方，填上前两个(工程名和工程文件夹路径)，后两个位置需要填写的内容可以自动生成。见上图。然后选择Next按钮进入下一步，见下图。



编译器选项仍旧选择默认的编译器，剩下的全部打勾，见上图。然后选择Finish按钮，则创建了一个为project1的工程。用鼠标点逐级击使之变成，依次展开左侧的project1，Sources，main.cpp，最后显示文件main.cpp的源代码，见下图。

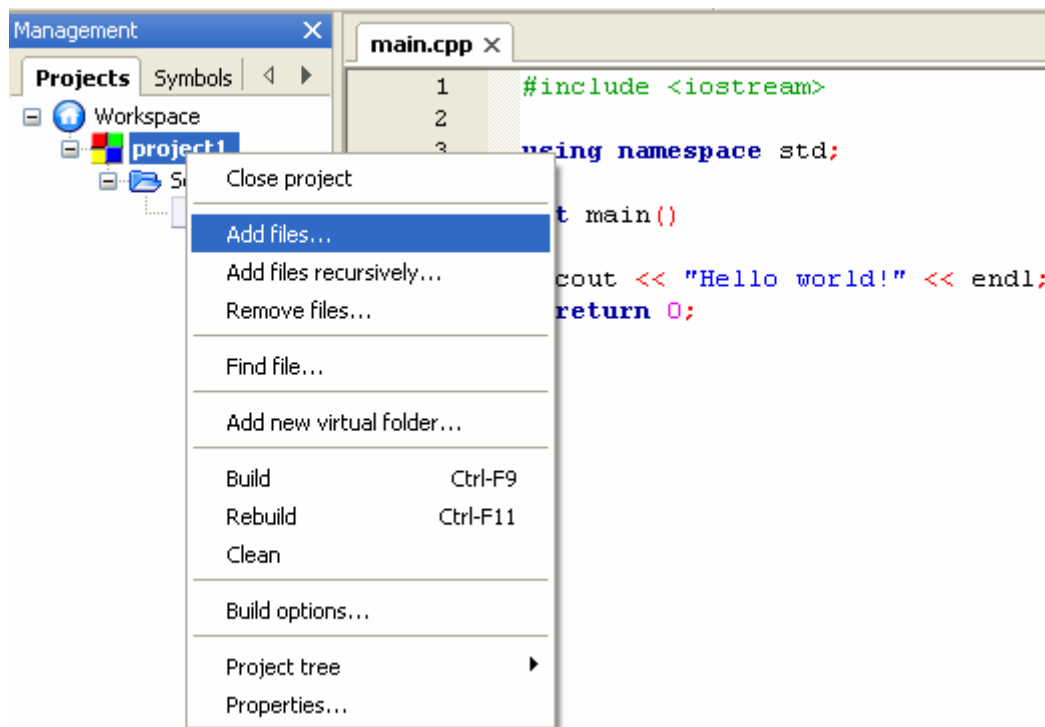


3.2 添加和删除文件

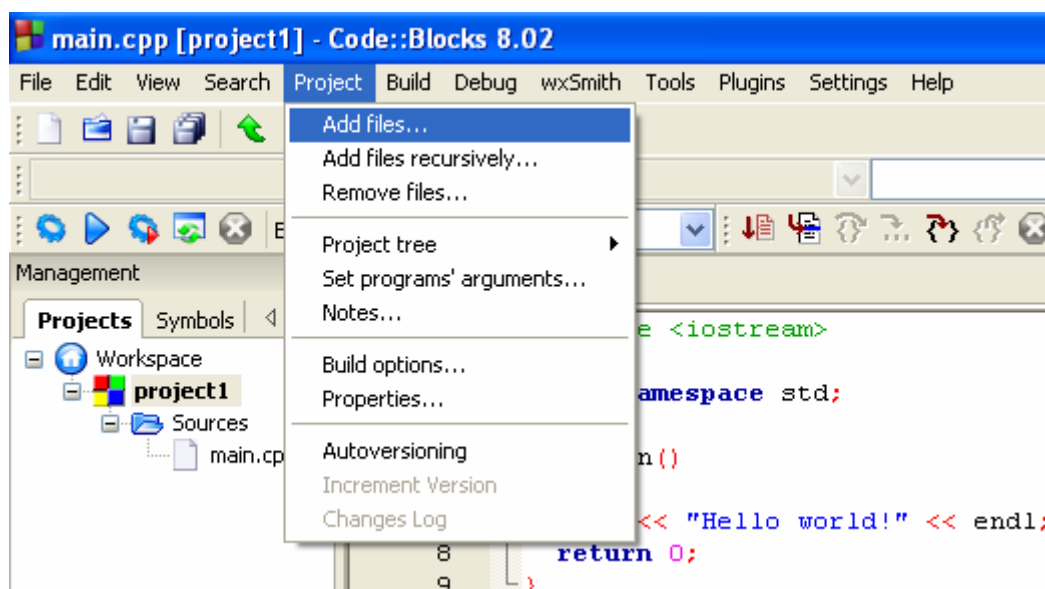
当建立一个工程后，我们往往需要往工程中添加新文件，工程不需要的文件则要从工程中删除之。

3.2.1 添加文件

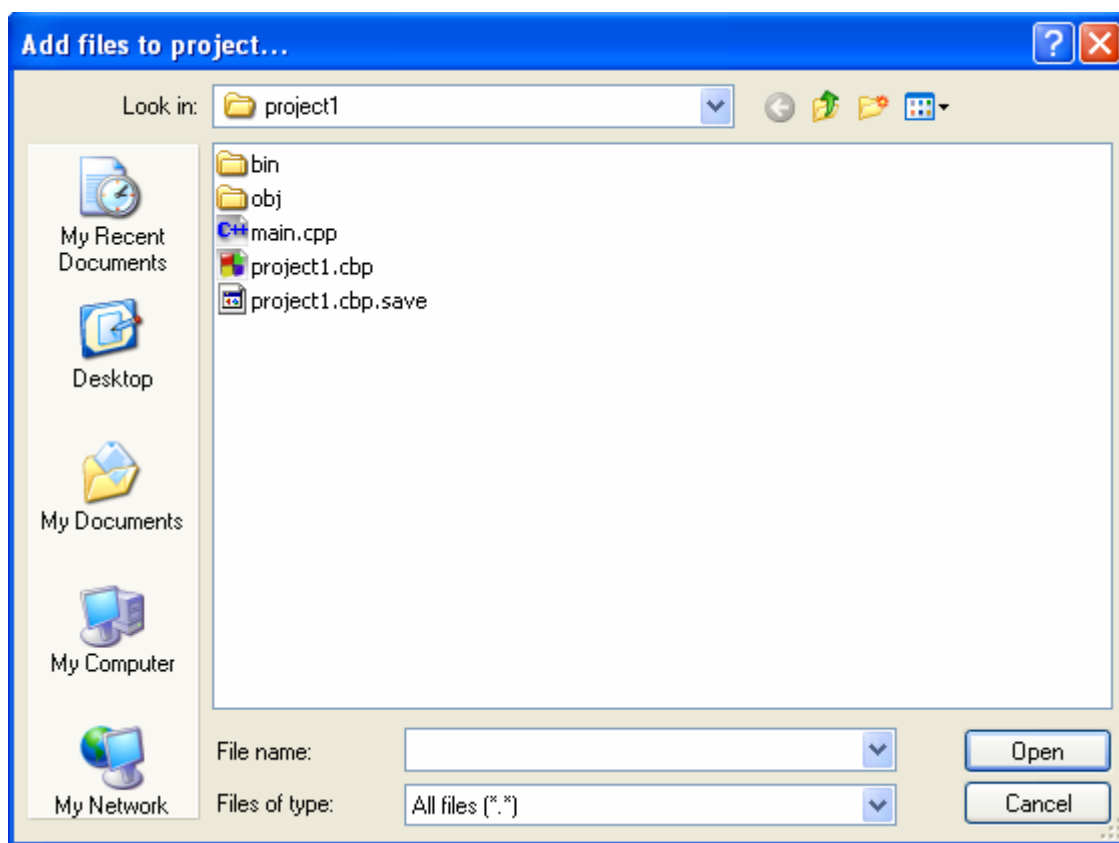
给工程添加文件的方法很多，一种办法是移动鼠标选择刚建立的工程题头上(project1)，按下鼠标右键，弹出一个菜单，菜单上有几个按钮，Close project是关闭当前工程，Add files...是添加文件到工程中，另外还有Remove files...是从当前工程中删除文件。Build用来编译当前工程，Rebuild用来重新编译当前工程，Clean用来清除编译生成的文件，见下图。



另外一种方法是从Project主菜单选择下拉菜单中的按钮。Add files...用来添加新文件到当前工程中，Remove files...是从当前工程中删除文件，见下图。

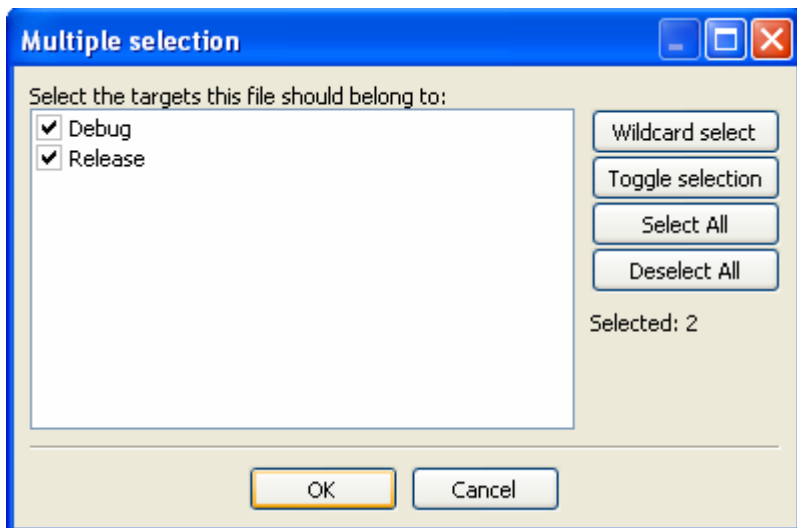


用鼠标点击上面介绍的菜单中Add files...按钮，会弹出一个对话框，见下图。

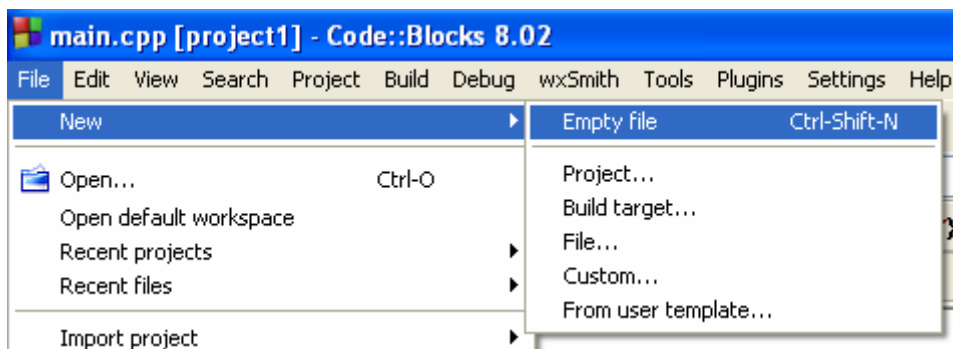


选择要添加的文件，然后选择右下角的Open按钮，见上图。

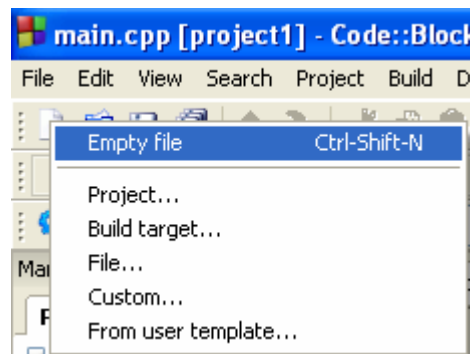
点击Open后，又弹出一个对话框见右图，把Debug和Release全部选中，然后选择OK按钮，则添加到了当前中。



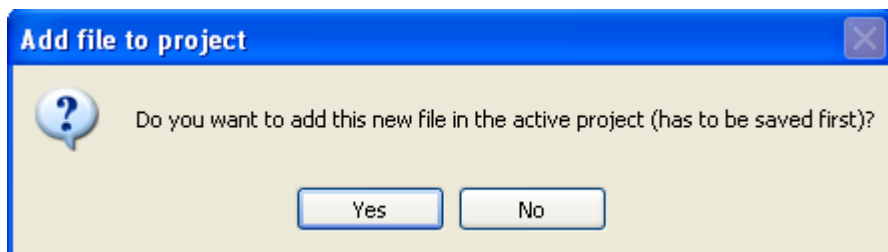
如果此时存储设备上没有我们需要的文件，则需要自己创建一个。创建新文件的方法很多，可以选择File菜单，从下拉菜单中中选择New，通过向右的导向箭头打开New的子菜单，再从这个下拉菜单中选择按钮Empty file，Ctrl-Shift-N，见右图。



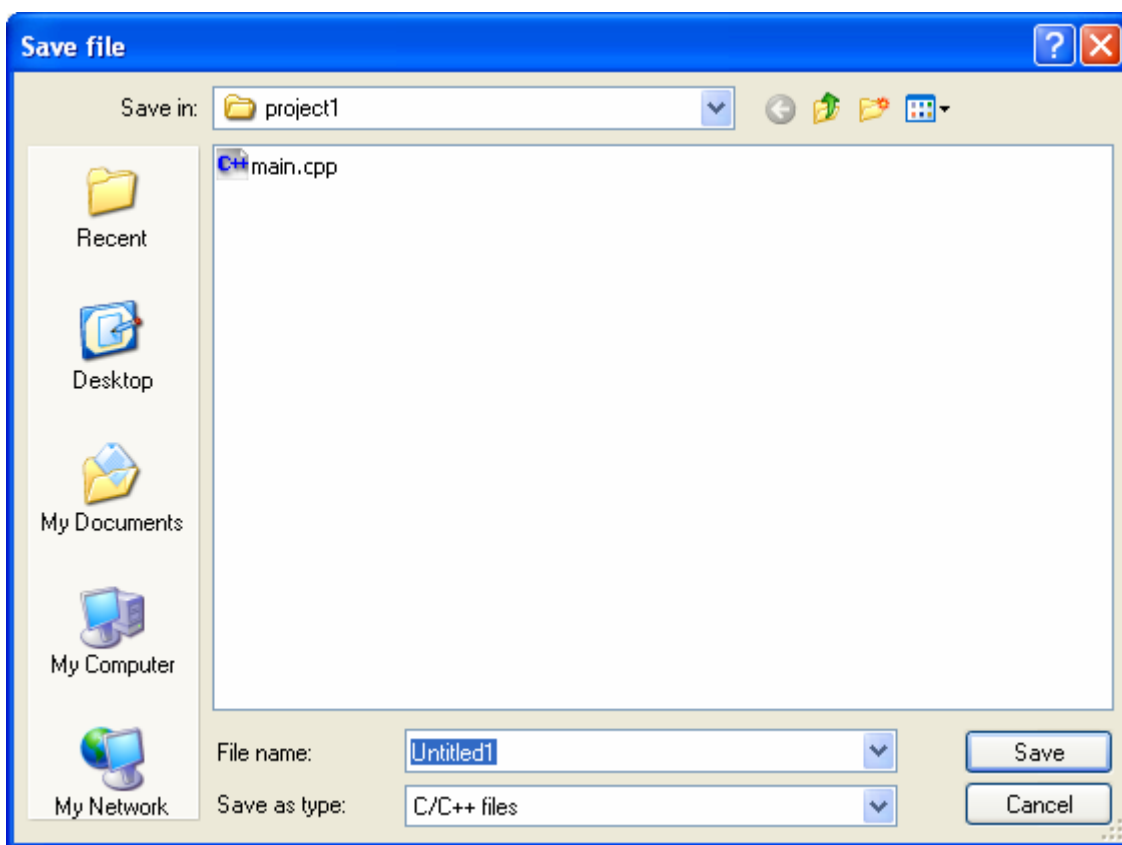
也可以选择这个图标，用鼠标点击一下，也会弹出一个菜单，从弹出的菜单中选择Empty file Ctrl-Shift-N，见右图。



如果您建立了一个工程，然后又想新建一个空文件，则Code::Blocks会问您是否要把这个新文件添加到当前处于激活状态的工程中，见下图。

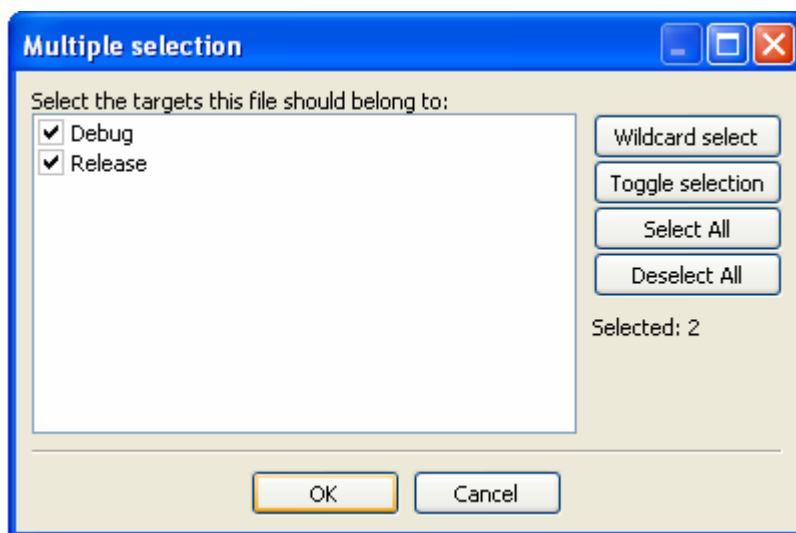


如果您选择Yes，则该文件会被添加到工程中，选择No，则此文件不会被添加到工程。如果您选择了Yes，则会弹出一个新的对话框，让您给新建的这个文件命名，见下图。

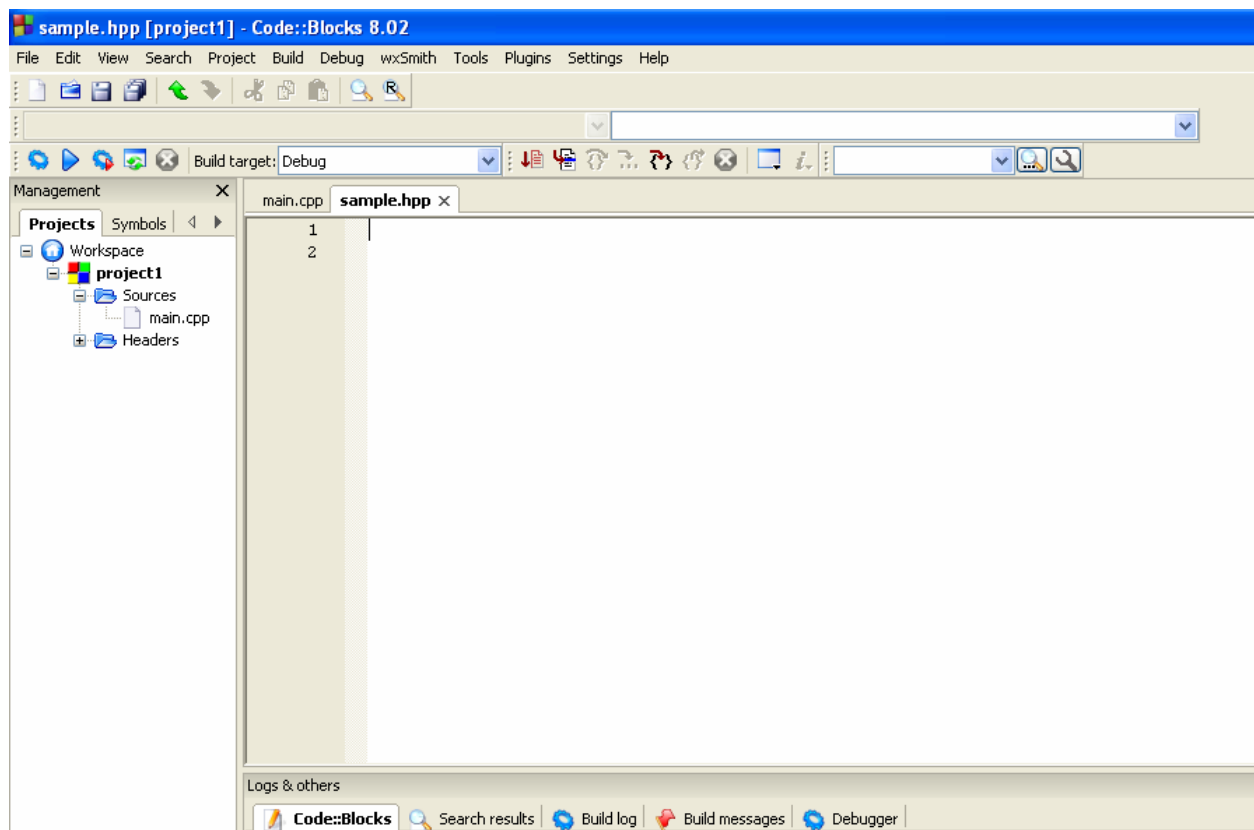


我们假设把这个文件取名为sample.hpp，把它保存到新建的project1文件夹下面。

当用鼠标点击Save后，又弹出一个对话框，问您目标(target)文件属于哪种类型，选中Debug和Release，然后选择OK。见右图。




这样我们就可以编辑sample.hpp了(文件名后缀为.hpp或者.h的是头文件)，系统自动把它归为头文件，见下图。

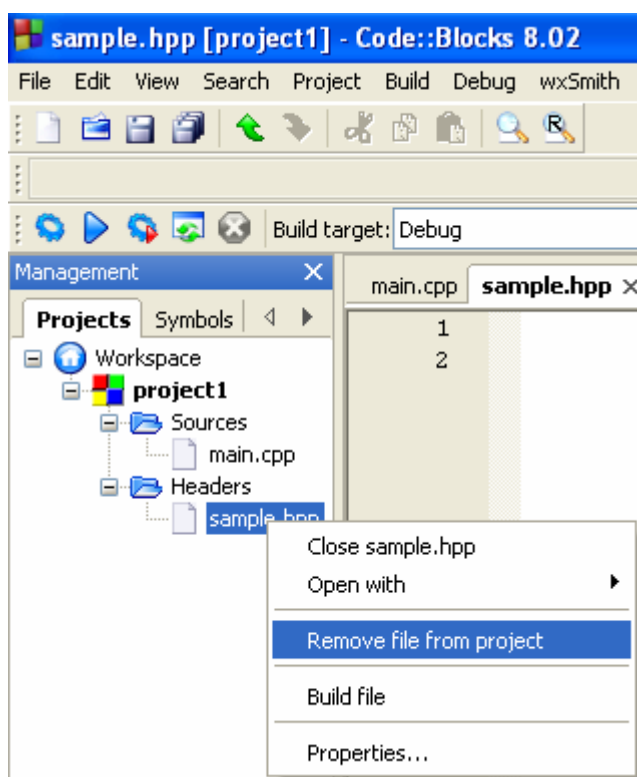


一个目标文件是编译后的文件，可以为debug或者release，debug版本的目标文件允许您使用调试器对该文件进行测试。一般而言，debug版本的目标文件通常较大，因为它包含了一些用于测试的额外信息，release版本的目标文件一般较小，因为它不包含调试信息。当您的程序编译完毕，应该交付release目标文件。

3.2.2 删除文件

我们前面创建的文件sample.hpp仅仅是为了示例说明如何创建文件，因此并不是project1所必须的，需要把它从project1中删除。

删除这个文件方法很多，这里简要说明两种，一种方法是用鼠标点击图标 Headers上的，这样Headers就会展开，选择sample.hpp，按下鼠标右键弹出的菜单中选择按钮Remove file from project，则sample.hpp就不在隶属于project1了，见右图。




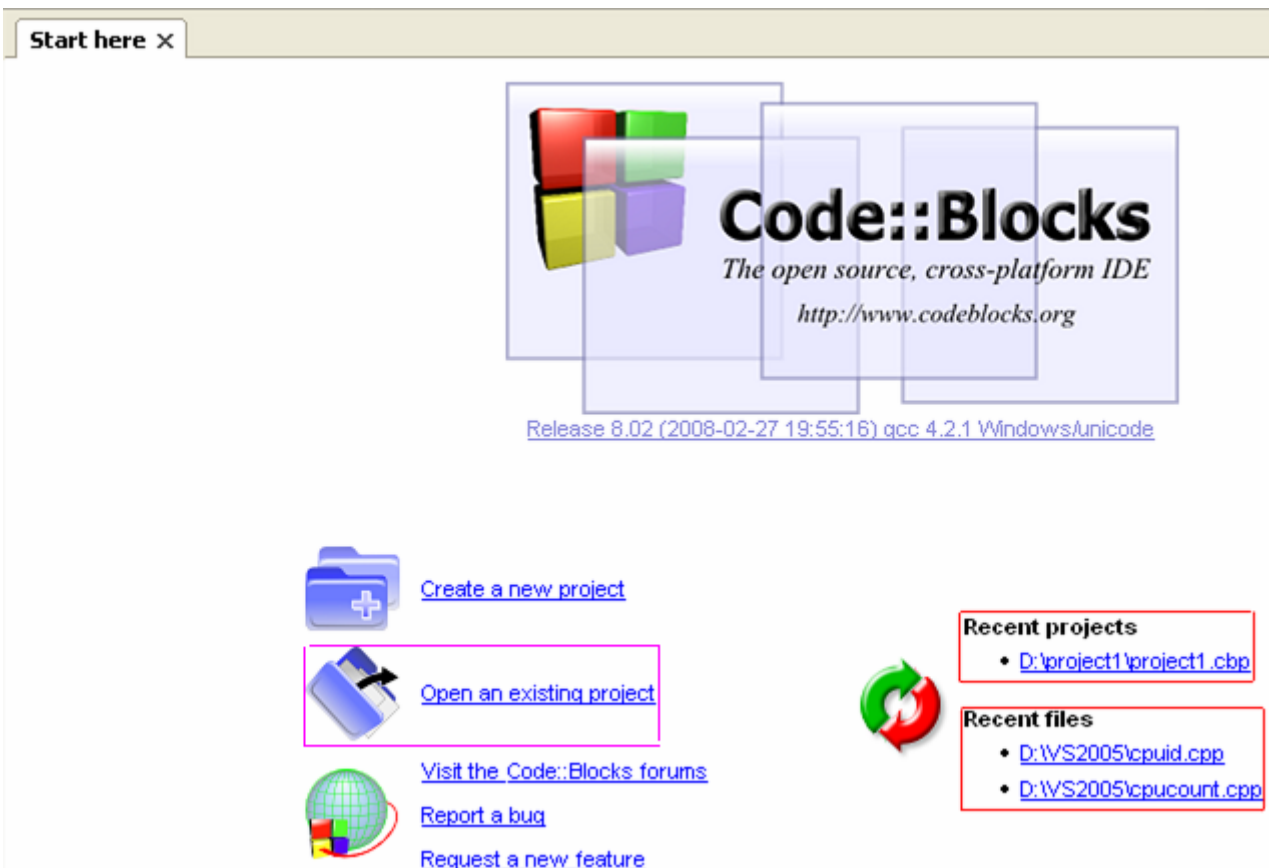
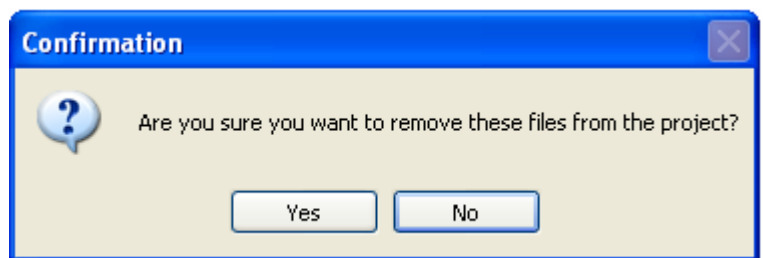
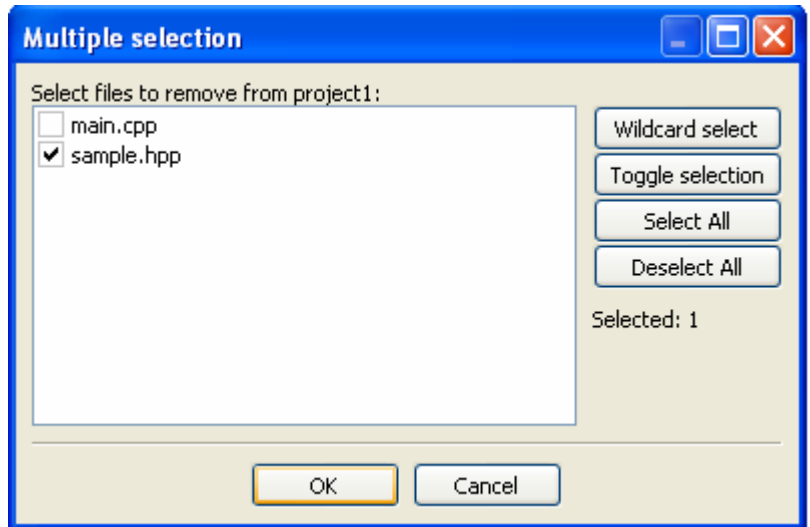
我们还可以从主菜单选择Project的下拉菜单中选择Remove files...按钮，这样会弹出一个对话框，对话框中是当前处于激活状态工程project1的所有文件，需要把哪个文件从该工程中删除，则选中那个文件，然后选择OK。见右图。


当用鼠标点击OK按钮后，还会弹出一个对话框让您确认，选择Yes，则此文件sample.hpp就不再隶属于当前工程project1了，见右下图。

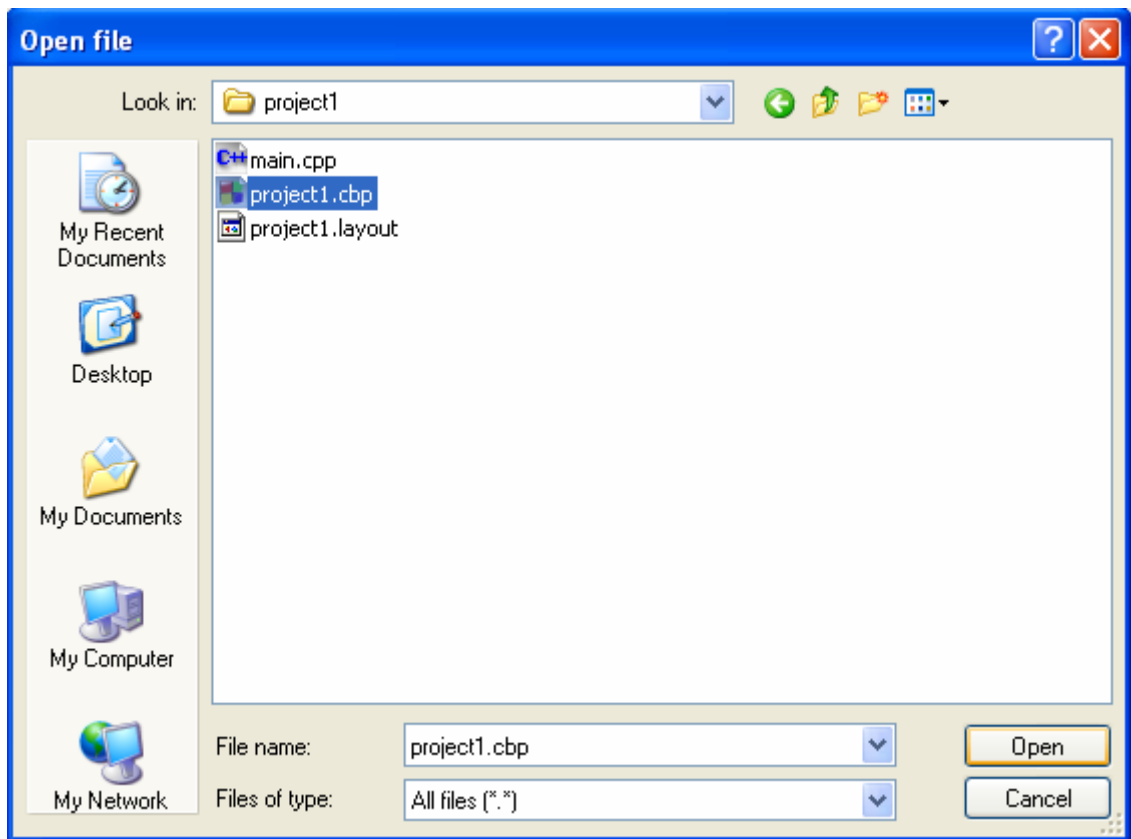
3.3 编辑文件


编辑已经存在的文件前需要首先打开这个文件。打开文件的方法很多，

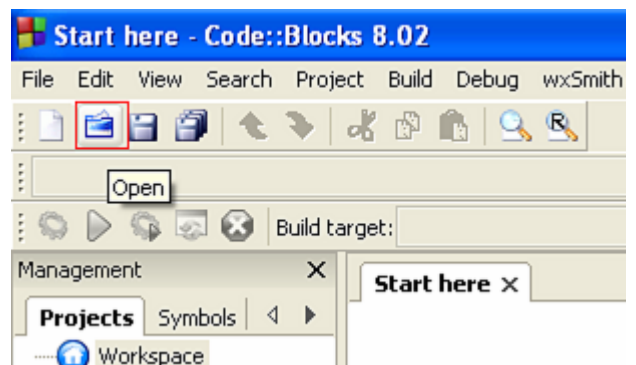
由于Code::Blocks会默认记住打开的文件路径，进入Code::Blocks主界面后可以看到最近用Code::Blocks打开过的工程以及最近打开过的文件。见右下图中标识右侧用红色方框框起来的部分Recent projects以及Recent files。用鼠标选择之，即可打开对应的工程或文件。



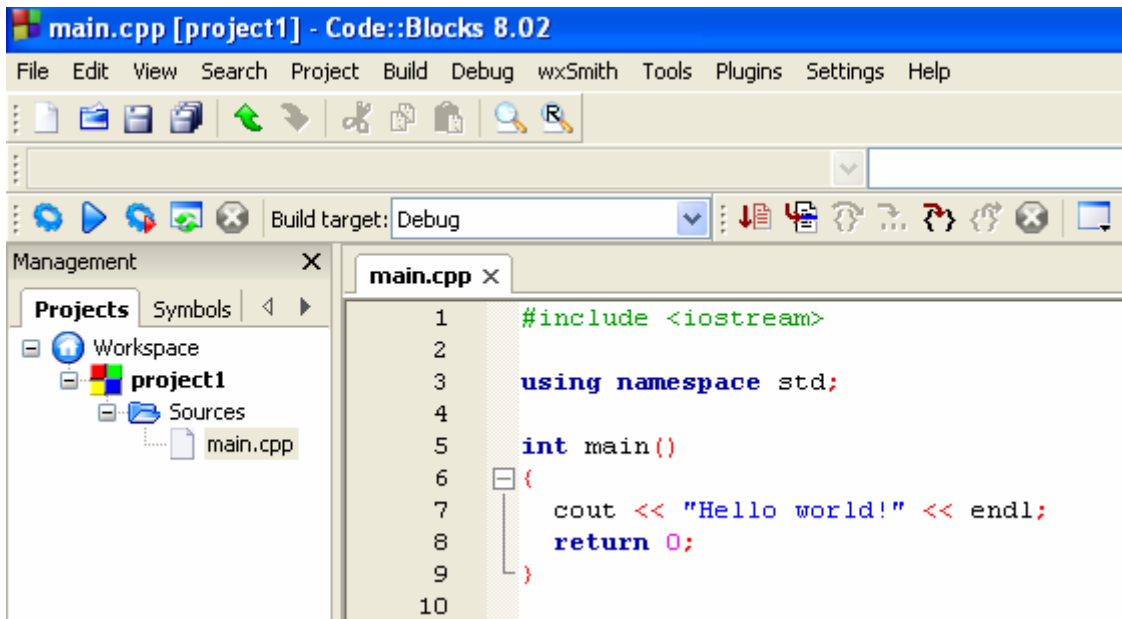
也可以选择带有标签Open an existing project的图标，见右图中用粉红色方框框起来的部分，用鼠标点击它则会弹出一个对话框见右下图，让您选择要打开的工程，选中对应文件名，点击Open按钮。




还可以从主界面Edit主菜单下面的图标，见左下图。点击该图标会弹出一个对话框，对话框见右图，然后找到要打开的文件，选中它，点击Open按钮，也可以打开相应的工程或者文件。

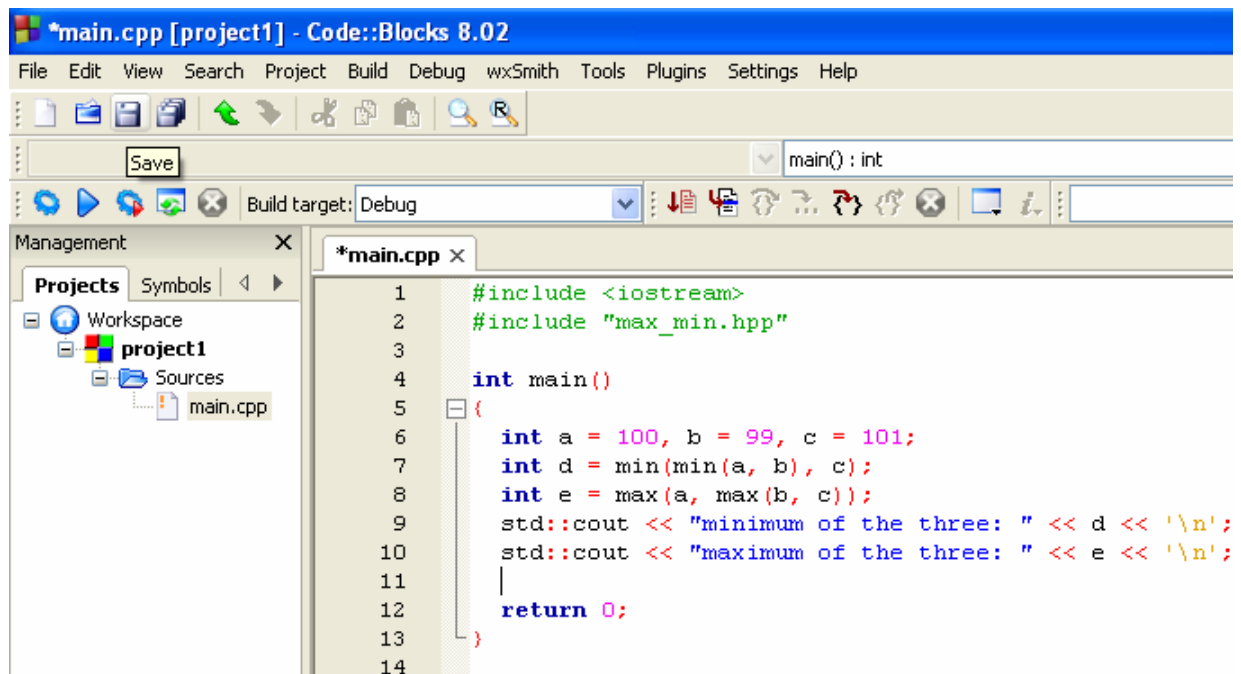


打开一个工程后再打开工程中的某个文件，则显示该文件的源代码，见下图。



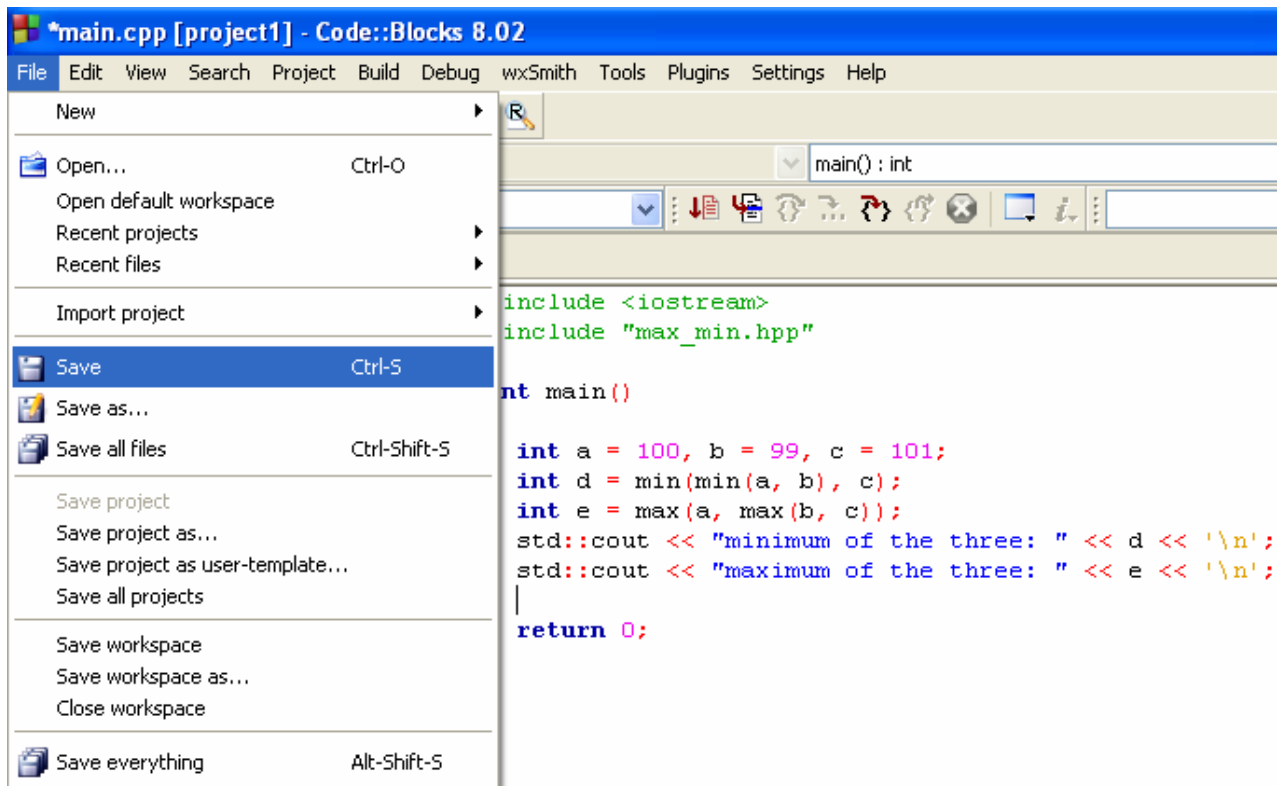
假设我们想修改此文件的源代码，让它求出三个整数中的最小值，则可以这样做。首先，我们修改main.cpp的源代码，修改完毕后，我们保存当前文件。






保存当前文件方法很多，常见的有两种，一种是用鼠标点击一下按钮，见下图。

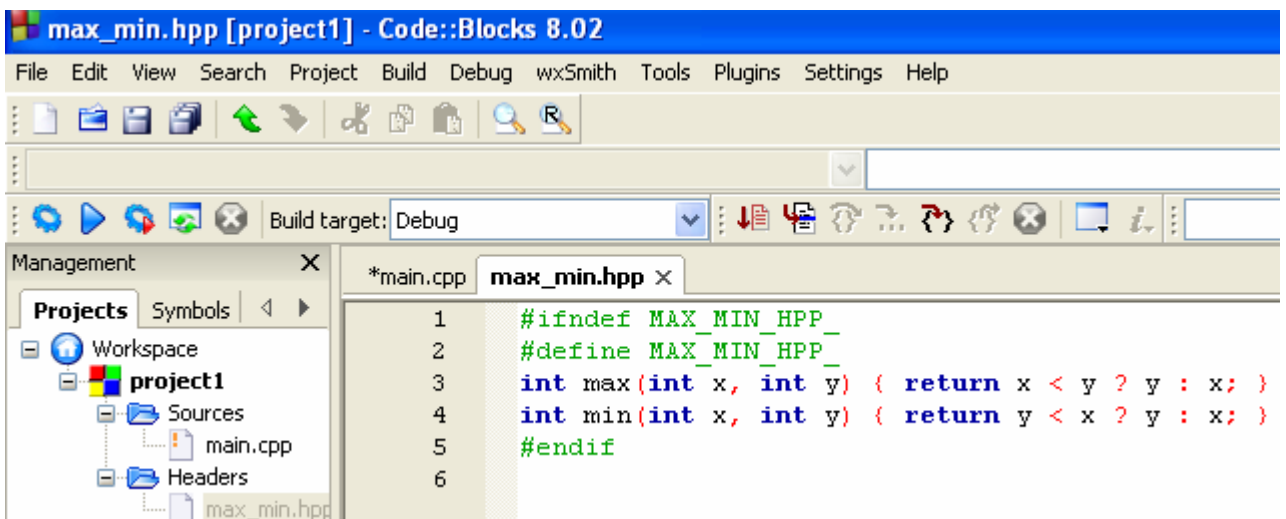


另外一种方法是在主菜单File的下拉菜单中选择按钮 Save

Ctrl-S，见下图。

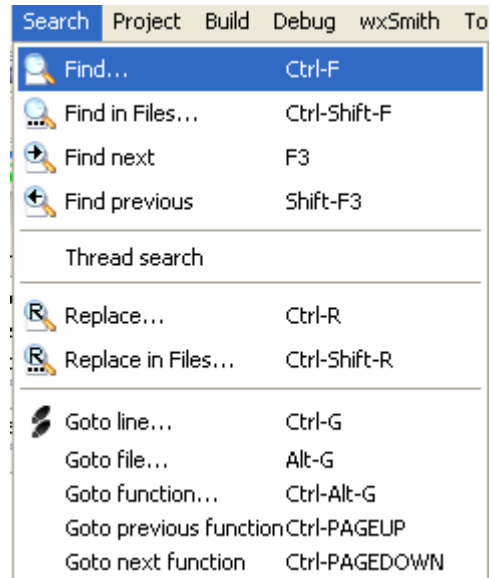


假如我们想保存project1中的所有文件，可以找到图标，用鼠标点击它一下即可，或者从File的下拉菜单中找到按钮 Save all files Ctrl-Shift-S，用鼠标点击一下也可。如果我们想保存整个project1工程，也可以从File的下拉菜单中找到按钮 Save project 点击一下则就保存了project1。此外File的下拉菜单中还有 Save all projects按钮，用来保存当前工作空间中的所有工程， Save workspace按钮用来保存当前工作空间， Save everything Alt-Shift-S按钮用来保存Code::Blocks已经打开的所有内容。如果我们想给当前文件创建一个副本，改成其它名字则在File的下拉菜单中选择 Save as...按钮，取一个新名字保存之，如果我们想给当前工程创建一个副本，改成其它名字则在File的下拉菜单中选择 Save project as...按钮，取一个新名字保存之。此外还有 Save workspace as...等。如果想关闭当前文件选择File下拉菜单中的按钮 Close file Ctrl-W，关闭当前工程选择按钮 Close project。关闭所有文件就选择 Close all files Ctrl-Shift-W按钮，关闭所有工程选择 Close all projects按钮。



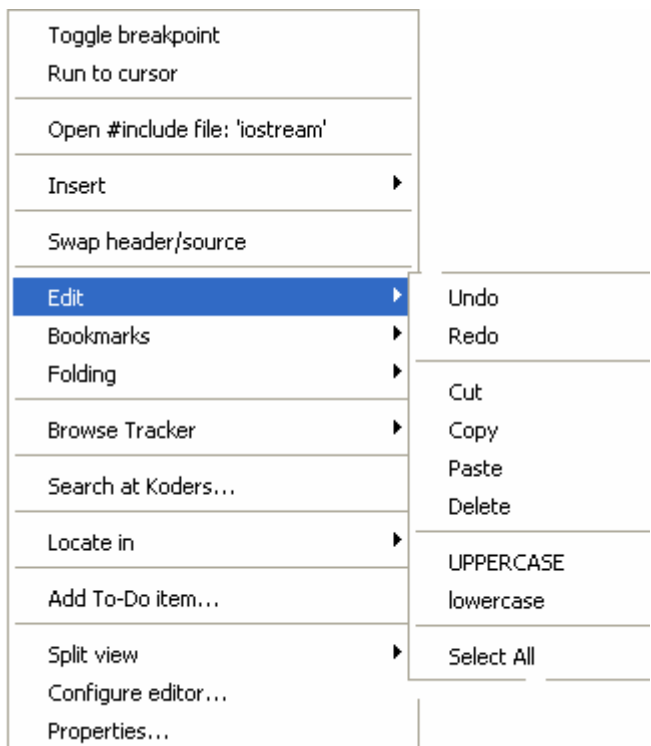
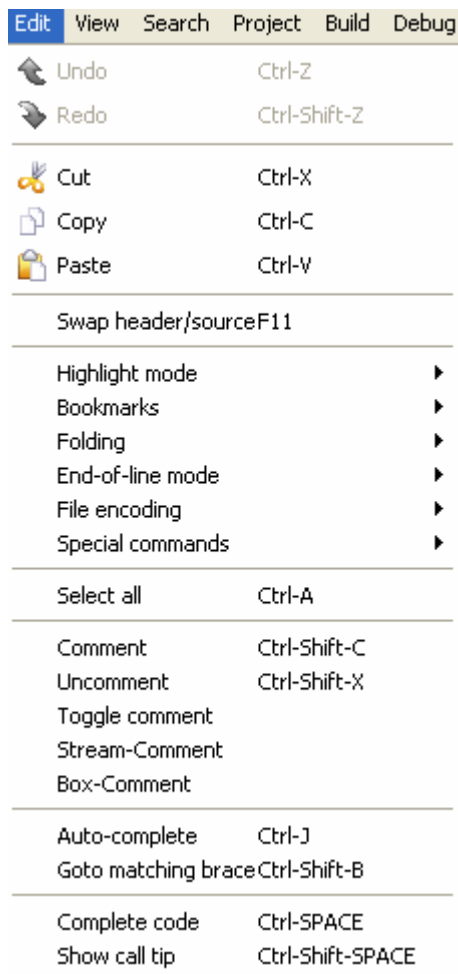
注意到，编辑后的main函数源代码中添加了一行#include “max_min.hpp”，max_min.hpp是一个头文件，该头文件包含了两个函数，一个是max，一个是min，分别用来求出两个整数的较大和较小者。现在我们需要建立这个头文件，并把它添加到project1中。添加文件到工程的方法前面已经有所论述，在此不再重复，假设我们已经编辑完毕max_min.hpp头文件，并添加到project1中，如上图。

如果编写的源代码较多，有时候可能需要查找和替换某个字符串。Search主菜单的下拉菜单中有几个按钮用来对文件内容进行操作，可以查找和替换等，见右上图。



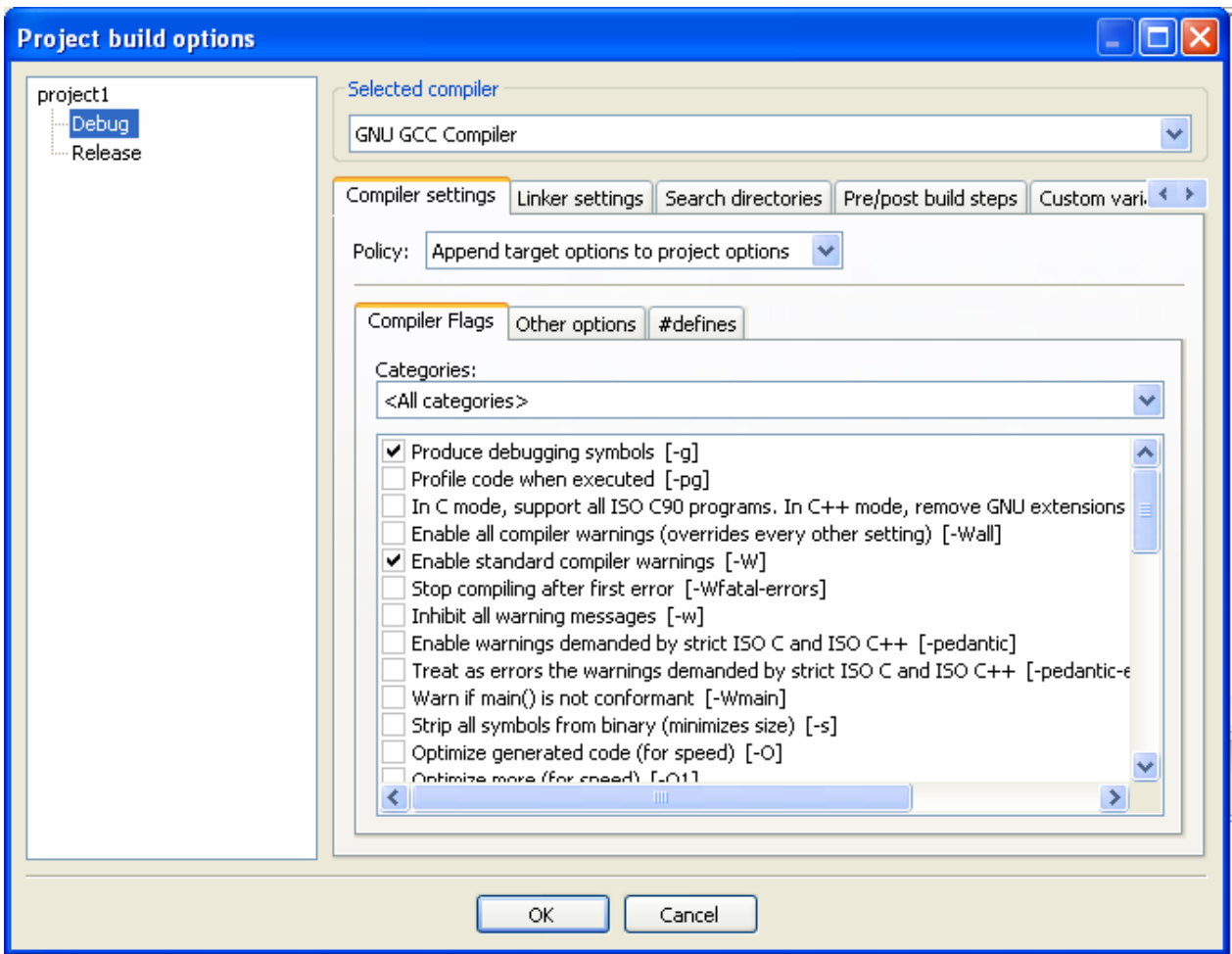
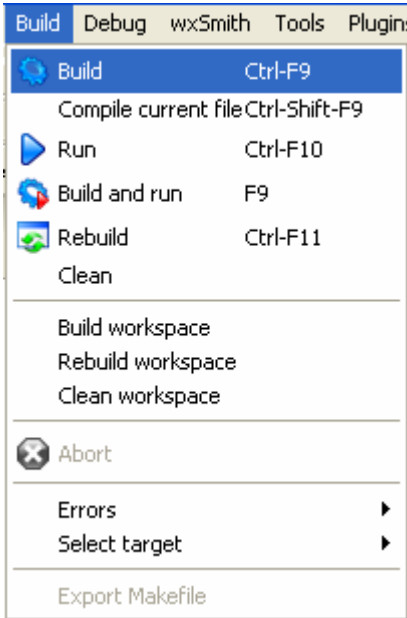
此外，还有一些快捷按钮，可以完成查找(Find)

或替换操作(Replace)，见上图用红色方框框起来的两个按钮。该快捷菜单上还有其它功能按钮，Undo(向前反悔)，Redo(向后反悔)，Cut(剪切)，Copy(复制)，Paste(粘贴)等，操作简单实用。Edit下拉菜单提供了更加丰富的功能按钮，除了上述这些外，还有Comment(注释)，去掉注释(Uncomment)，Folding(折叠)，File encoding(文件编码选项) 等等，见左下图。此外，编辑模式下，按下鼠标右键弹出的快捷菜单中也有很多编辑选项，见右下图。



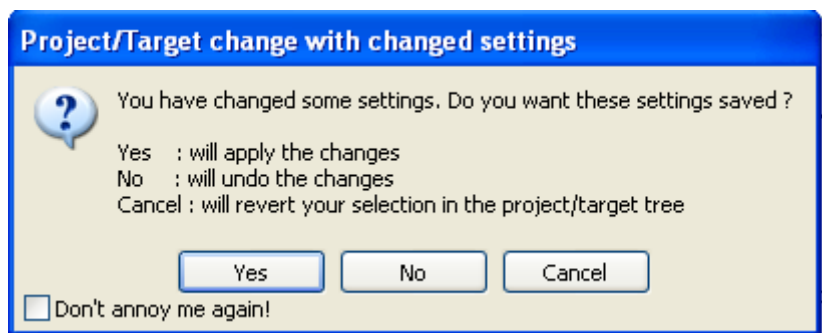
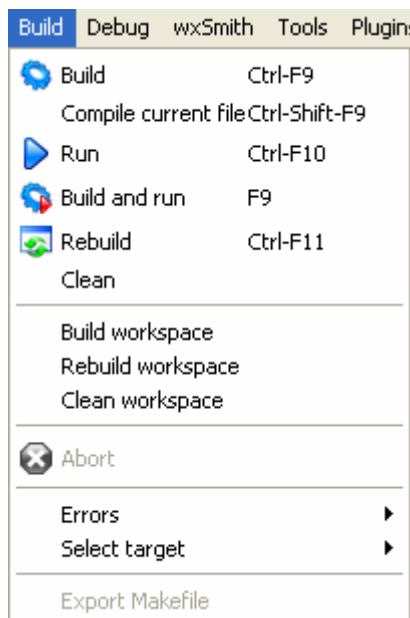
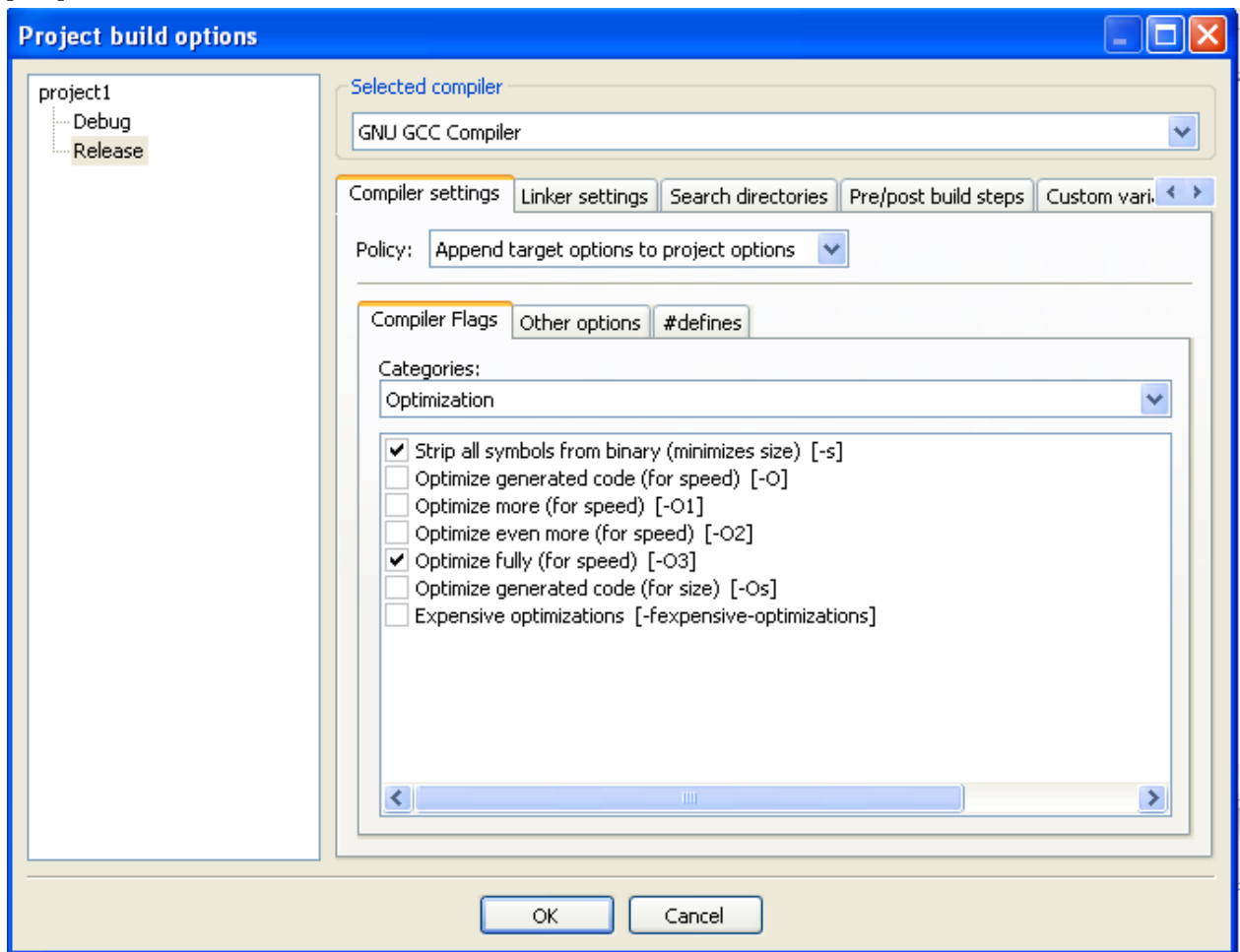
3.4 编译程序

编译一个工程前先从Project的下拉菜单中找到Build options...选项，见右图。

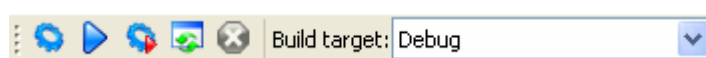


用鼠标点击Build options...按钮会弹出一个关于project1的对话框，有两个类别，debug和release，首先配置debug选项，一般而言，只关注Compiler Flags菜单下的两项，Producing debugging symbols [-g]和Enable standard compiler warnings [-W]。前者表示产生调试信息，后者意味着给出标准的编译警告信息，

分别打勾，见上图。然后配置Release，选择子菜单Compiler Flags下的Optimization选项，对于普通的应用，选择两项足矣，分别是Strip all symbols from binary (minimizes size) [-s]和Optimize fully (for speed) [-O3]，这两项前面打勾，见下图。



从debug切换到Release会弹出一个对话框，选择Yes。见上图。
编译一个工程或者文件的功能按钮都在Build下拉菜单，见左图。
此外，编译程序还有快捷菜单，见下图。

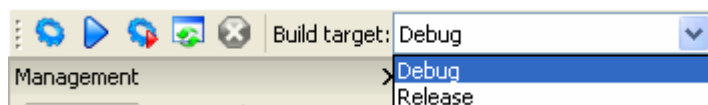


两个菜单中按钮图标相同的，功能也相同。

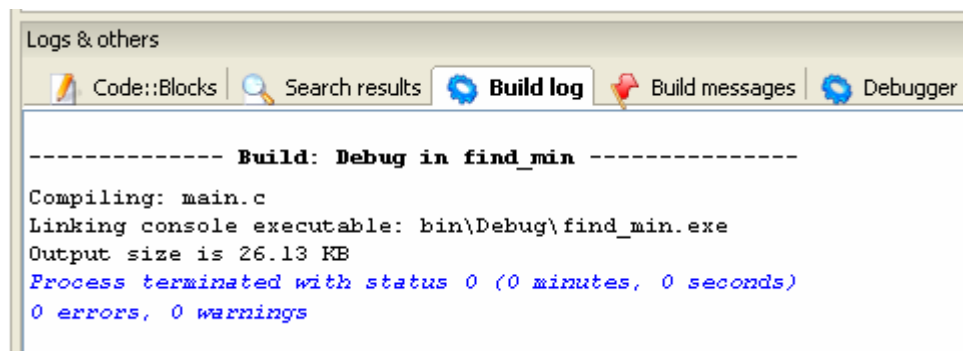
表示编译当前工程，表示运行编译成功的文件，表示编译并运行。如果编译当前文件，可以使用 Build 下拉菜单中的按钮 Compile current file Ctrl-Shift-F9，也可以展开左侧的工程目录树，移动鼠标到需要单独编译的文件上面，利用鼠标右键弹出菜单中选择 Build file 按钮，见右图。

如果一次编译不成功，修改后重新编译可以使用按钮 Rebuild，它代表重新编译，如果想清楚上次编译的目标文件，可以使用 Build 的下拉菜单中的 Clean 按钮。

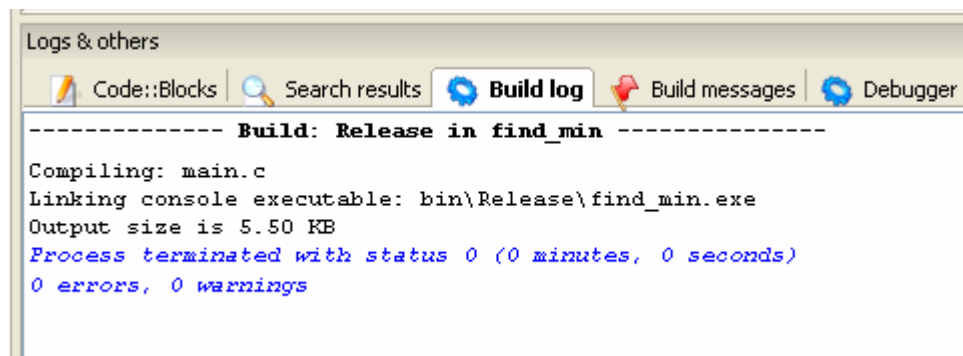
根据客观需要，我们可以把目标文件编译成 debug 或者 release 版本，用鼠标从快捷菜单上的 Build target 小窗口进行选择即可，见下图。




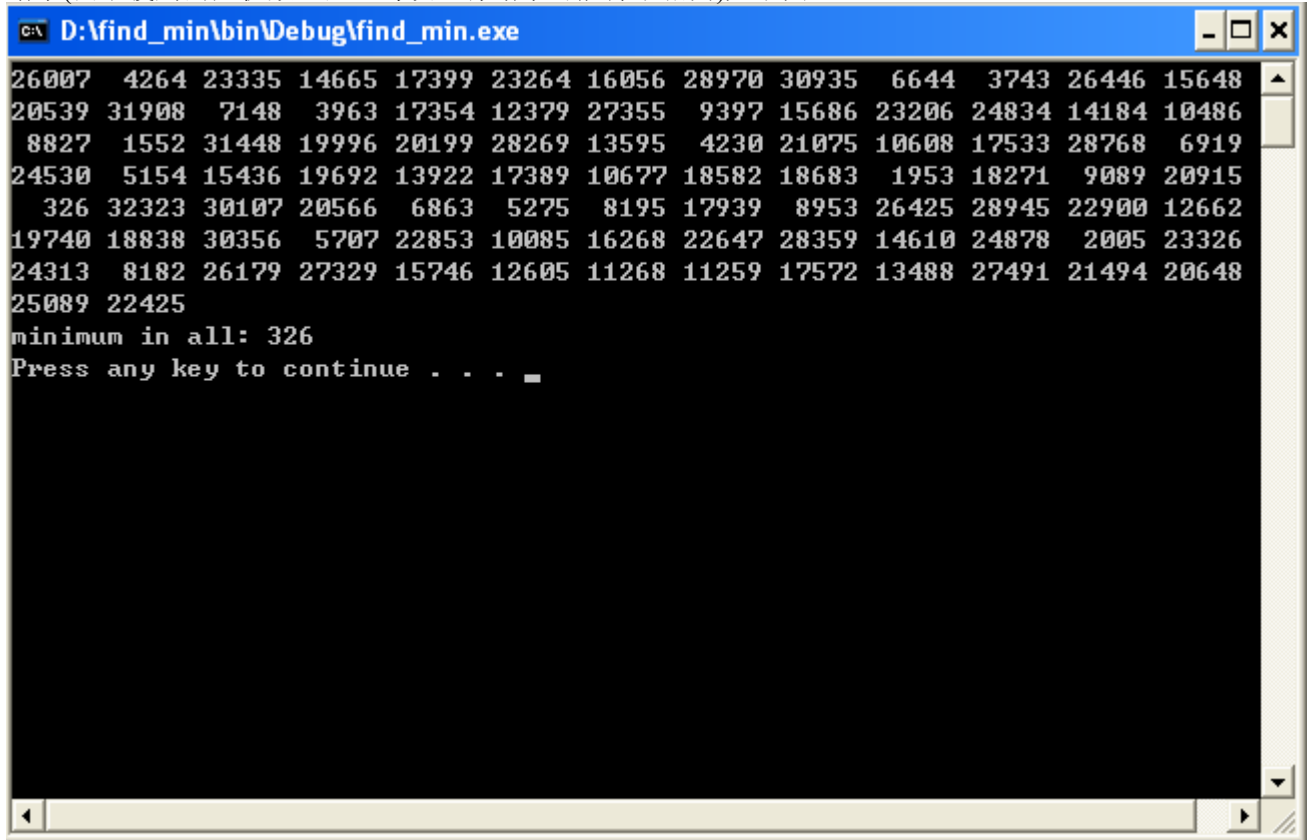
作为例子，我们用 C 语言编写一个求一个随机整数集中最小值的程序。首先建立工程 find_min，编写程序(程序源代码见本章的下一小节 3.5.1)，然后编译。首先选择编译目标文件为 debug 版本，用鼠标点击快捷菜单上的按钮，则开始编译，编译完毕后日志窗口有一些提示信息，见下图。



我们选择目标文件为 release 版本再编译，编译完毕后日志窗口有一些提示信息，见下图。如下图。



运行该目标文件(debug或者release版本), 可以用鼠标点击快捷菜单上的按钮 , 则一种可能的运行结果(由于使用的随机数, 因此每次运行结果可能都不相同)见下图。



```
G:\ D:\find_min\bin\Debug\find_min.exe
26007 4264 23335 14665 17399 23264 16056 28970 30935 6644 3743 26446 15648
20539 31908 7148 3963 17354 12379 27355 9397 15686 23206 24834 14184 10486
8827 1552 31448 19996 20199 28269 13595 4230 21075 10608 17533 28768 6919
24530 5154 15436 19692 13922 17389 10677 18582 18683 1953 18271 9089 20915
326 32323 30107 20566 6863 5275 8195 17939 8953 26425 28945 22900 12662
19740 18838 30356 5707 22853 10085 16268 22647 28359 14610 24878 2005 23326
24313 8182 26179 27329 15746 12605 11268 11259 17572 13488 27491 21494 20648
25089 22425
minimum in all: 326
Press any key to continue . . .
```

生成debug和release版本的二进制文件运行结果相同, 但它们二进制文件大小不同, 此处find_min的debug版本二进制文件大小26.13KB, release版本的二进制文件大小5.5KB。由于前者包含了一些测试信息, 所以它的二进制文件较大。

一次编译成功当然最好, 但是很多时候不能一次编译成功, 这时需要根据给出的出错信息修改源程序, 然后重现编译, 可能需要反复这个过程, 直到编译成功。编译成功说明没有语法错误, 但未必没有逻辑错误, 程序中的逻辑错误需要您自己检查, 当然了, 您可以使用调试工具帮助检查逻辑错误。

再看一个例子, 下面这个例子用来查看当前编译器的补白(padding), 也有很多人称补白为对齐(alignment), 补白是编译器为了提升程序执行速度, 让不同数据类型占用同样的长度一起处理, 从而减少CPU取指令次数, 提升运算速度。下面给出的代码不能一次编译成功, 需要我们找到问题所在, 并改正其中的语法错误。

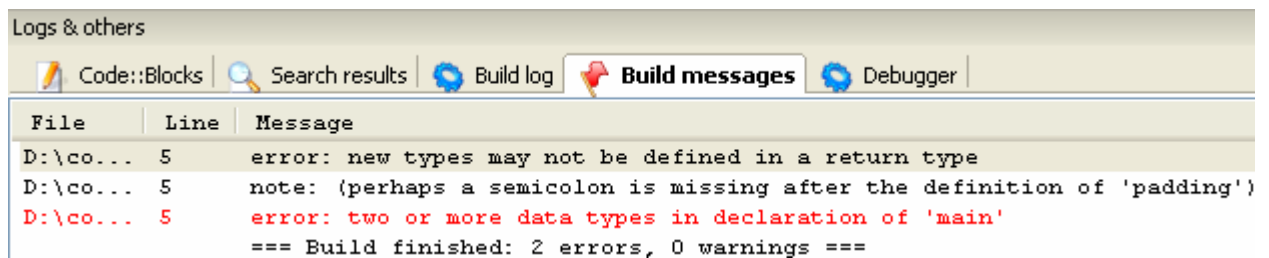
```
1 // padding.hpp
2
3 class padding
4 {
5 private:
6     char a;
7     int b;
8     double c;
9 public:
10     padding(char c = 'a', int i = 100, double d = 2.3) : a(c), b(i), c(d) {}
11 }
12
```

```

1 // padding.cpp
2 #include <iostream>
3 #include "padding.hpp"
4
5 int main()
6 {
7     padding a;
8     std::cout << "sizeof(padding) == " << sizeof(padding) << '\n';
9     std::cout << "(sizeof(char) + sizeof(int) + sizeof(double)) == "
10         << sizeof(char) + sizeof(int) + sizeof(double) << '\n';
11
12     char* p = (char*)(&a);
13     int int_len = sizeof(int);
14     std::cout << "1st element: " << *p << '\n';
15     std::cout << "2nd element: " << *(int*)(p + int_len) << '\n';
16     std::cout << "3rd element: " << *(double*)(p + (int_len << 1)) << '\n';
17
18     return 0;
19 }
20

```

编译提示信息见如下贴图。



上面的提示大致意思是：

“出错：在返回类型后新类型必须定义”

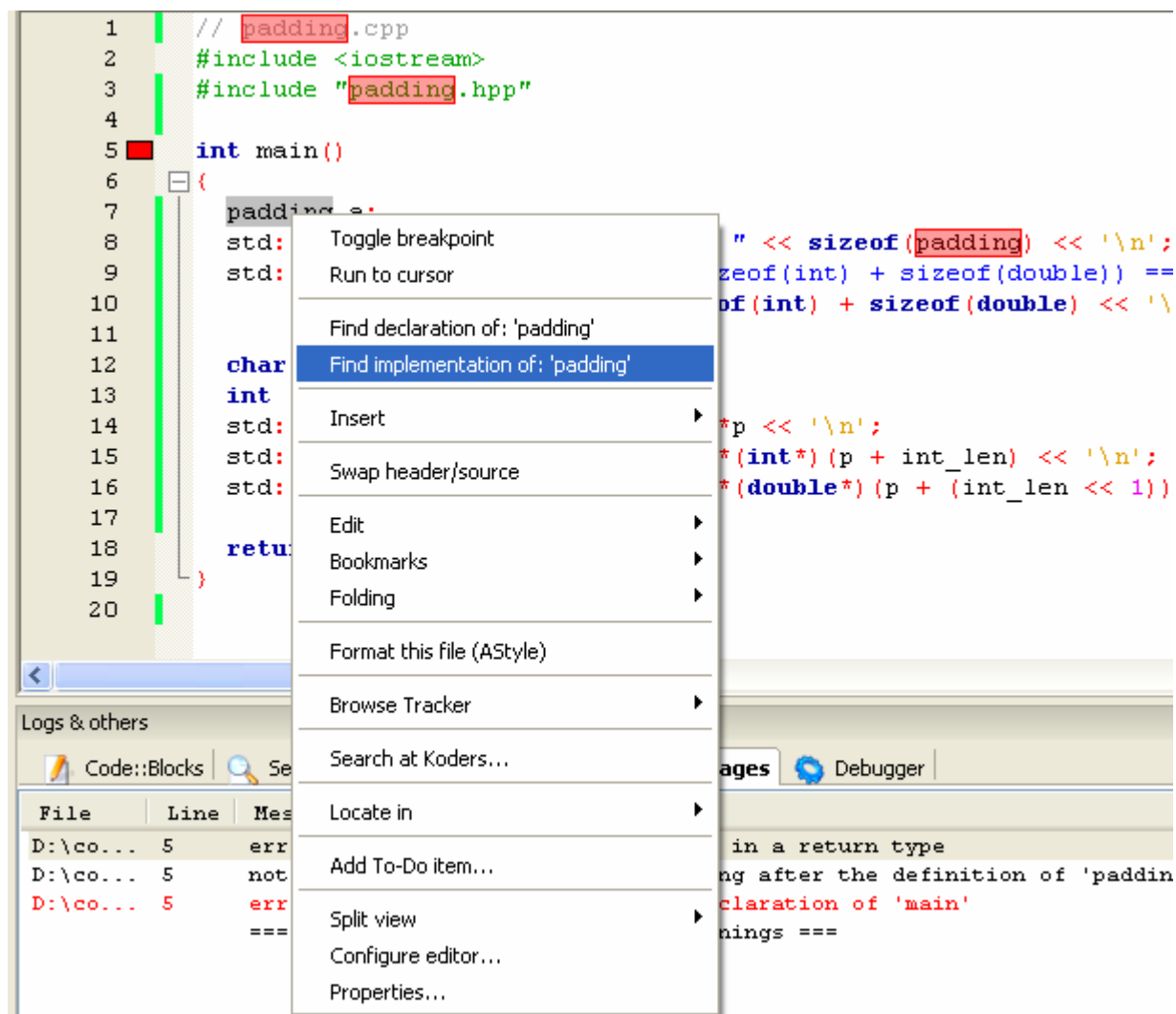
“注意：可能padding定义后少了一个分号”

“出错：main中声明了两个或者更多个类型”

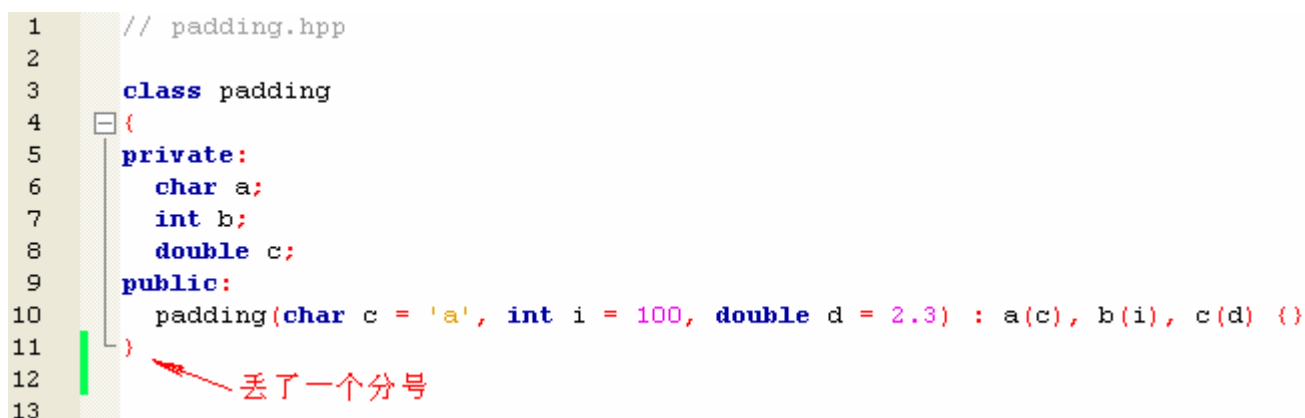
修改源程序中的错误技巧性非常强，如果程序不能通过编译，那么根据出错的提示信息修改源程序时，能看懂哪些错误信息就先修改哪里的错误，有时可能源程序仅一两个错误，但编译器提示错误信息给出很多，这些错误信息中很大一部分可能没有任何帮助价值，我们需要找出对我们有意义的错误信息。有时我们发现，修改完一个错误重新编译后原来给出的很多错误信息突然变得少多了。



此外，如果您经常编程，最好能读懂编译器给出的提示信息。如果您的计算机专业英语非常好，那将对您的编程有很大作用，不仅仅是看编译器给出的提示信息方便了修改程序中的语法错误，而且编程过程中经常需要查阅函数的说明文档，计算机专业英语对于编程有很大帮助。

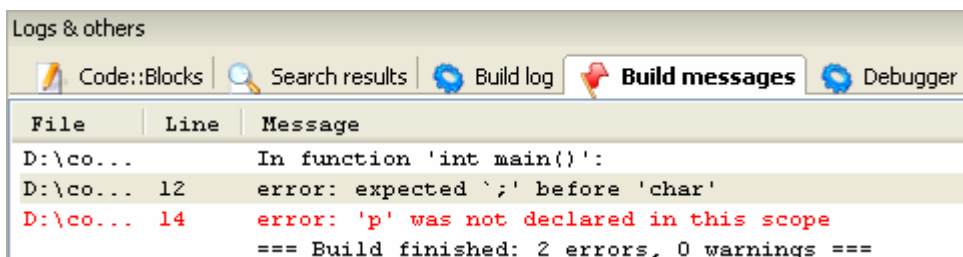
现在开始修改错误。我们首先去找padding的定义，看看是否忘记了一个分号。查找padding的声明或者实现很简单，选中padding，则padding变成了灰色，所有和padding相同的字符窜都变成了红色(注：笔者使用了比较新的Code::Blocks版本，如果您用的版本比较旧，则可能没有这样的功能，颜色可以设置，不一定是红色)，按下用鼠标右键在弹出的快捷菜单中选择按钮“Find implementation of: 'padding'”，见下图。



选中它后屏幕会自动切换到padding的实现，我们发现，padding类最后果然少了一个分号，见下图。



加上这个分号后点击按钮  保存当前文件padding.hpp，然后点击按钮  编译。结果又有出错信息，见下图。



我们想修改哪行出错的代码，就用鼠标在下面的提示信息栏选中它，然后双击鼠标左键，则屏幕自动切换到那行处。

我们先不管第14行，看看第12行，提示说，char前本应该有;，第11行是个空行，不要忘记编译器会忽略掉空行和注释，因此我们查看前一行第10行，发现果然丢了一个分号。

```

1 // padding.cpp
2 #include <iostream>
3 #include "padding.hpp"
4
5 int main()
6 {
7     padding a;
8     std::cout << "sizeof(padding) == " << sizeof(padding) << '\n';
9     std::cout << "(sizeof(char) + sizeof(int) + sizeof(double)) == "
10         << sizeof(char) + sizeof(int) + sizeof(double) << '\n';
11
12     char* p = (char*)(&a);
13     int int_len = sizeof(int);
14     std::cout << "1st element: " << *p << '\n';
15     std::cout << "2nd element: " << *(int*)(p + int_len) << '\n';
16     std::cout << "3rd element: " << *(double*)(p + (int_len << 1)) << '\n';
17
18     return 0;
19 }
20

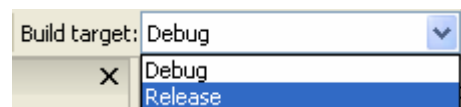
```

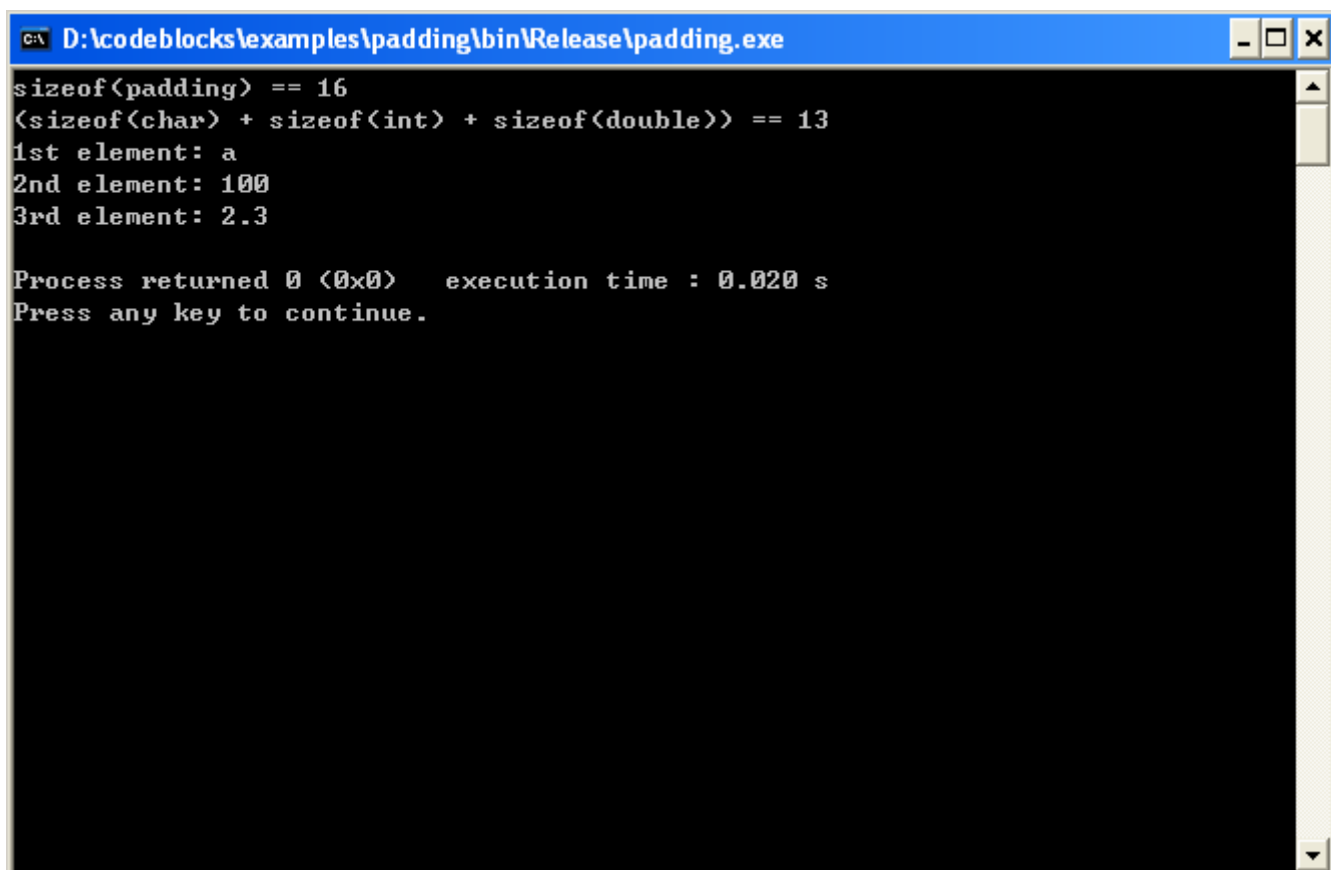
丢了一个分号 ——

补上一个分号，保存，然后继续编译。这下编译成功了，提示信息见下图。



再编译一个release版本(Build target:栏目选择Release然后编译)运行之，结果见下图。





```
C:\> D:\codeblocks\examples\padding\bin\Release\padding.exe
sizeof(padding) == 16
<sizeof(char) + sizeof(int) + sizeof(double)> == 13
1st element: a
2nd element: 100
3rd element: 2.3

Process returned 0 (0x0) execution time : 0.020 s
Press any key to continue.
```

注意：编译以上程序需要事先建立工程padding并设置好相应编译选项(Build options...), 跟前面建立Project1时设置编译选项类似, 为了节约篇幅这里就没有赘述。因为系统默认一些选项, 所以大多数情况下不用理会编译选项的设置也能正常编译程序, 但是编译程序时产生的提示信息未必就是您感兴趣和期望的, 例如, 有人可能希望产生标准的出错和警告信息, 有人则可能希望产生任何可能的出错和警告信息, 个别人则希望什么提示信息都没有从而省得心烦, 再者, 期望产生目标代码的类型可能也不同, 有人可能希望最高级优化, 有人则可能希望专门针对某种CPU优化, 还有人可能认为是否优化无所谓。鉴于此, 最好根据个人喜好和实际情况设置这些编译选项。

我们接下来再看一个例子。以下程序用来交换两个不同的值, 由于我们并不知道要交换的值的类型, 所以就使用模板, 参数类型用指针, 当交换的两个值是整数的时候进行特化, 参数类型用引用。为了知道调用了那个swap函数进行交换, 在swap函数体里面我们附加输出一点信息。

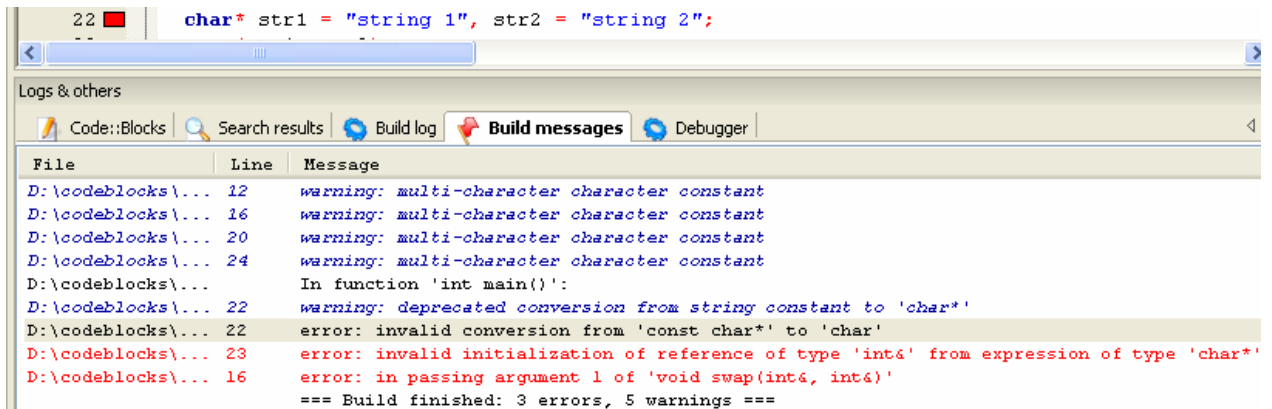
跟往常一样, 我们先建立一个工程, 工程名取undefined, 然后配置该工程的调试和编译参数。配置这些选项: ☒ Enable all compiler warnings (overrides many other settings) [-Wall]以便让编译器给出标准的提示信息, 当生成debug版本的二进制文件时, ☒ Produce debugging symbols [-g], 产生调试信息, 当需要生成release版本的二进制文件时, ☒ Optimize fully (for speed) [-O3], 进行三级优化, 让生成的二进制文件占用空间比较小, ☒ Strip all symbols from binary (minimizes size) [-s]。建议读者自行配置以上几个编译参数选项作为练习。

```

1 // swap.hpp
2 #include <iostream>
3
4 template <class T>
5 void swap(T* x, T* y)
6 {
7     std::cout << "swap(T*, T*) called.\n";
8     if (*x != *y)
9     {
10         T tmp = *x;
11         *x = *y;
12         *y = tmp;
13     }
14 }
15
16 void swap(int& x, int& y)
17 {
18     std::cout << "swap(int&, int&) called.\n";
19     if (x != y)
20     {
21         x ^= y;
22         y ^= x;
23         x ^= y;
24     }
25 }
26
27
28 // main.cpp
29 #include <iostream>
30
31 using namespace std;
32
33 #include "swap.hpp"
34
35 int main()
36 {
37     int a, b = 1;
38     swap(a, b);
39     cout << a << ' ' << b << '\n\n';
40
41     double c = 3.3, d;
42     swap(c, d);
43     cout << c << ' ' << d << '\n\n';
44
45     char e = 'e', f = 'f';
46     swap(&e, &f);
47     cout << e << ' ' << f << '\n\n';
48
49     char* str1 = "string 1", str2 = "string 2";
50     swap(str1, str2);
51     cout << str1 << ' ' << str2 << '\n\n';
52
53     return 0;
54 }

```

编译以上程序，产生如下提示信息。

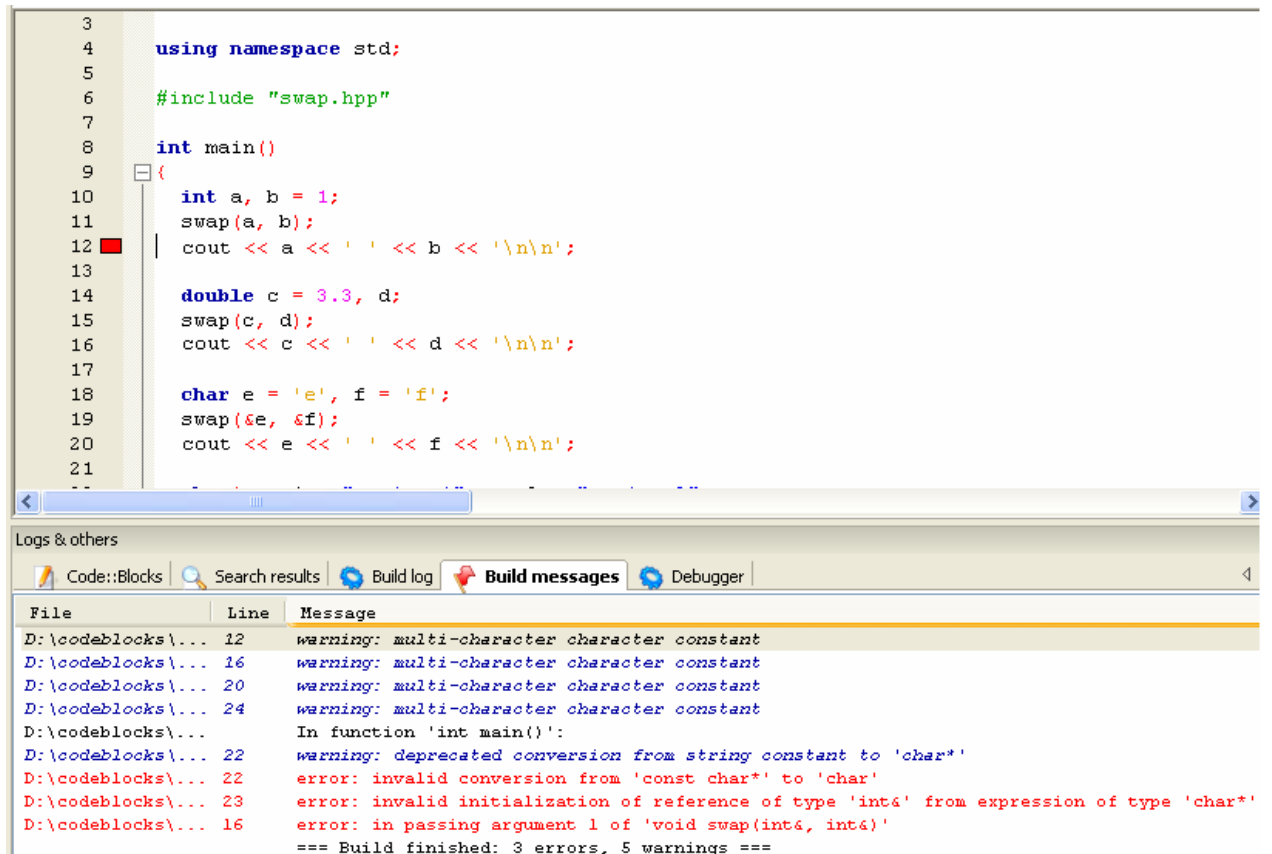


```
22 char* str1 = "string 1", str2 = "string 2";
```

File	Line	Message
D:\codeblocks\...	12	warning: multi-character character constant
D:\codeblocks\...	16	warning: multi-character character constant
D:\codeblocks\...	20	warning: multi-character character constant
D:\codeblocks\...	24	warning: multi-character character constant
D:\codeblocks\...		In function 'int main()':
D:\codeblocks\...	22	warning: deprecated conversion from string constant to 'char*'
D:\codeblocks\...	22	error: invalid conversion from 'const char*' to 'char'
D:\codeblocks\...	23	error: invalid initialization of reference of type 'int&' from expression of type 'char*'
D:\codeblocks\...	16	error: in passing argument 1 of 'void swap(int&, int&)'
=== Build finished: 3 errors, 5 warnings ===		


如此简单的一个小程序，居然产生了5个警告信息，而且编译器还报告3个错误。

首先我们看警告信息，警告说第12, 16, 20, 24行多个字符常量，第22行，不建议把字符串常量转换成char*。用鼠标双击第一行编译信息，则自动跳到了main函数的第12行，前面出现一个红色的方框，见下图。



```
3
4 using namespace std;
5
6 #include "swap.hpp"
7
8 int main()
9 {
10     int a, b = 1;
11     swap(a, b);
12     cout << a << ' ' << b << '\n\n';
13
14     double c = 3.3, d;
15     swap(c, d);
16     cout << c << ' ' << d << '\n\n';
17
18     char e = 'e', f = 'f';
19     swap(&e, &f);
20     cout << e << ' ' << f << '\n\n';
21 }
```

File	Line	Message
D:\codeblocks\...	12	warning: multi-character character constant
D:\codeblocks\...	16	warning: multi-character character constant
D:\codeblocks\...	20	warning: multi-character character constant
D:\codeblocks\...	24	warning: multi-character character constant
D:\codeblocks\...		In function 'int main()':
D:\codeblocks\...	22	warning: deprecated conversion from string constant to 'char*'
D:\codeblocks\...	22	error: invalid conversion from 'const char*' to 'char'
D:\codeblocks\...	23	error: invalid initialization of reference of type 'int&' from expression of type 'char*'
D:\codeblocks\...	16	error: in passing argument 1 of 'void swap(int&, int&)'
=== Build finished: 3 errors, 5 warnings ===		

果然，把两个换行符冠以单引号了，两个字符组成了一个字符串，应该冠以双引号，再看其它行上的几个，出现同样的问题了。用  Replace 一次性全部取代改正过来并保存当前文件。再看第22行，我们使用的是C语言风格定义了字符串，语法上没有问题，暂且不用管它。

接下来，看看编译错误信息。第22行，把const char*类型转换成char类型。仔细看一看发现第二个

字符定义少些了一个*, 结果编译器认为是把”string 2”这个常量字符串赋值给一个字符变量str2了, 见下图。

```
21
22 char* str1 = "string 1", str2 = "string 2";
23 swap(str1, str2);
24 cout << str1 << ' ' << str2 << "\n\n";
25
26 return 0;
27 }
```

漏掉了一个*

加上*, 保存当前文件, 然后再编译, 这次的编译提示信息如下。

Logs & others		
Code::Blocks Search results Build log Build messages Debugger		
File	Line	Message
D:\codeblocks\...		In function 'int main()':
D:\codeblocks\... 22		warning: deprecated conversion from string constant to 'char*'
D:\codeblocks\... 22		warning: deprecated conversion from string constant to 'char*'
=== Build finished: 0 errors, 2 warnings ===		

只有警告不建议的C风格字符串常量转换了。暂且不用理会, 先运行看一下结果再说。

```
C:\ D:\codeblocks\examples\undefined\bin\Debug\undefined.exe
swap<int&, int&> called.
1 2293728

1.13222e-317 3.3

swap<T*, T*> called.
f e

swap<T*, T*> called.
string 1 string 2

Process returned 0 (0x0)   execution time : 1.702 s
Press any key to continue.
-
```

结果并不理想, 疑点很多, 见下图中红色文字标示。