

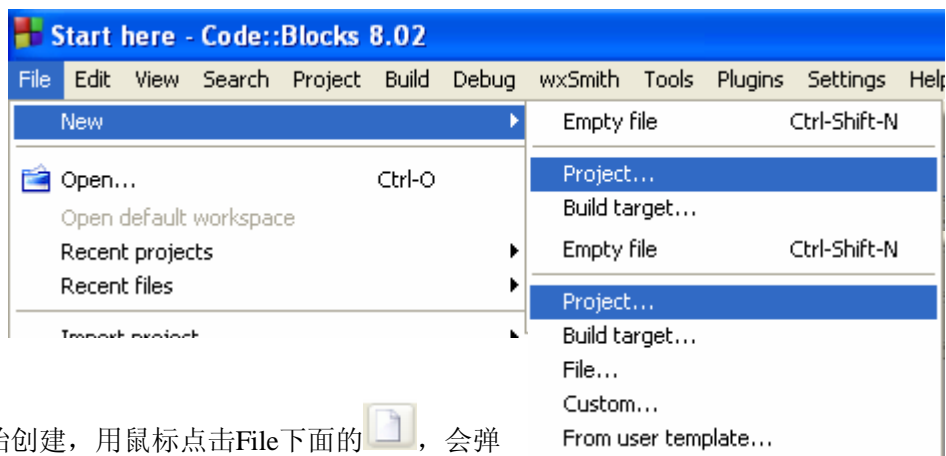
### 3. 编写程序



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。



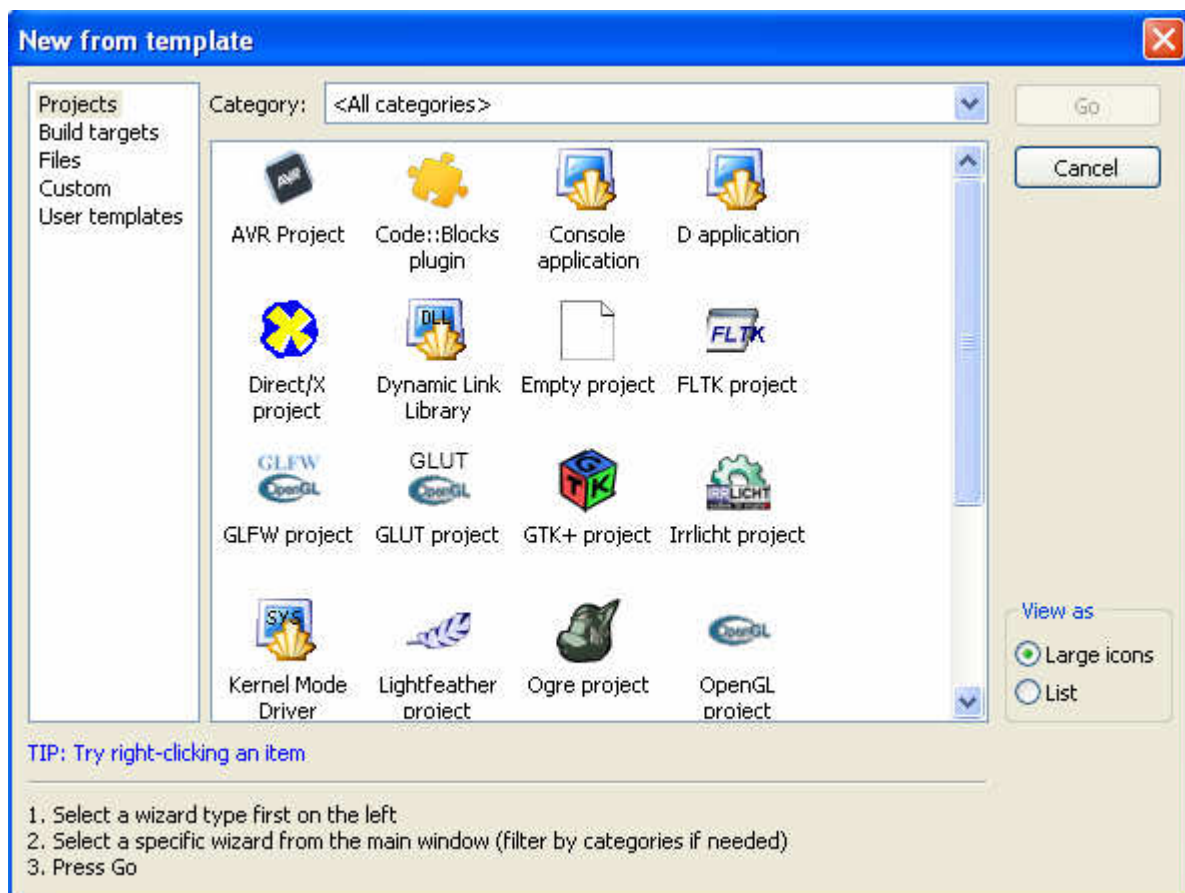
也可以从图标按钮开始创建，用鼠标点击File下面的，会弹出一个对话框，从弹出的对话框中选择Project...按钮，如右图：

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。



[Create a new project](#)

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。

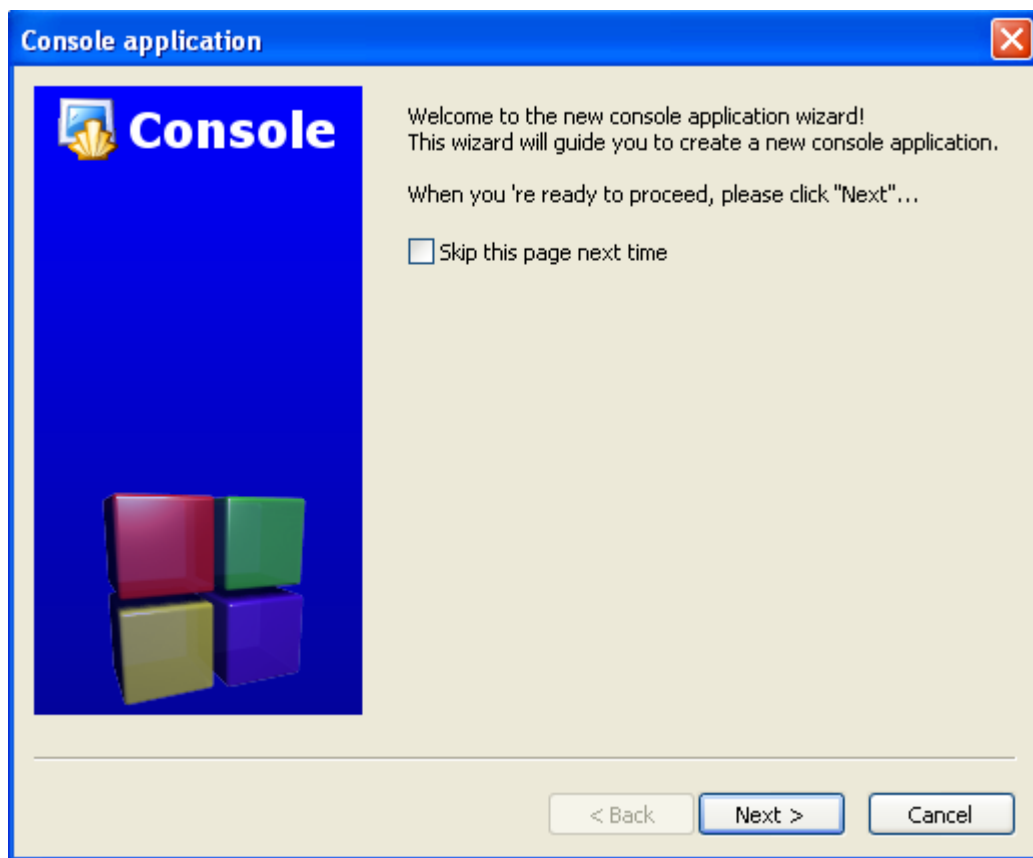


这个窗口中含有很多带有标签的图标，代表不同种类的工程。我们最常用的可能是Console application，用来编写控制台应用程序。其它的是一些更高级的应用。

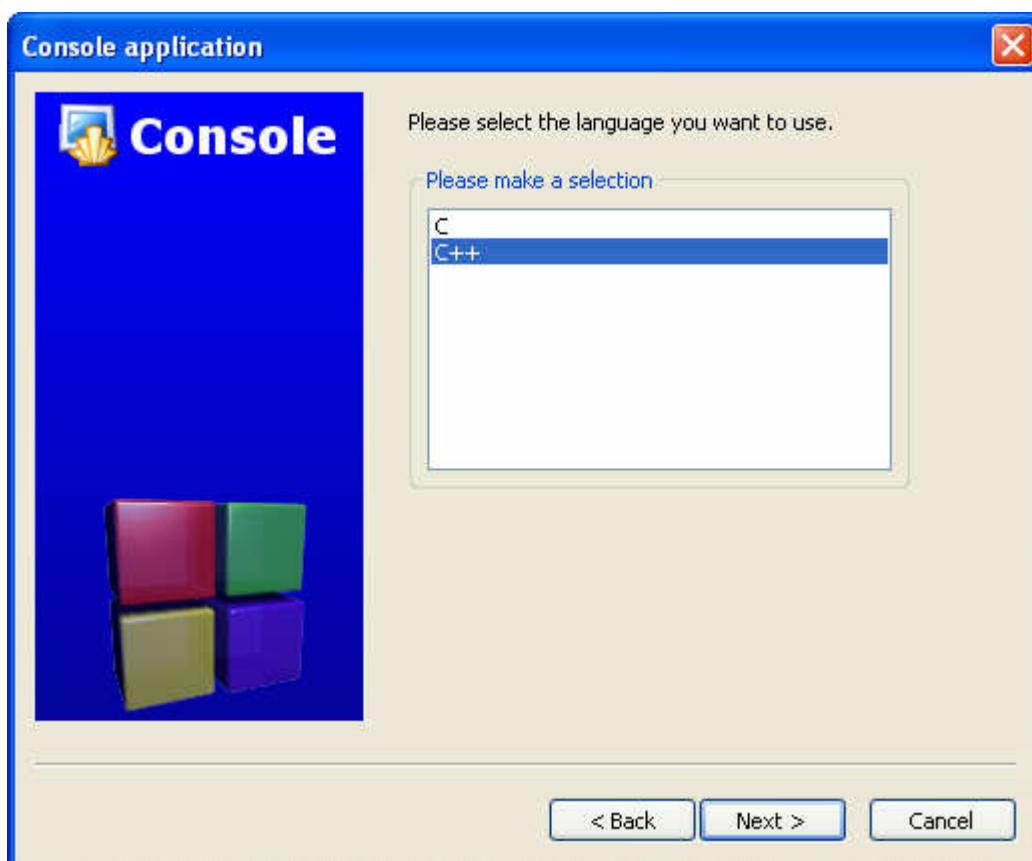
用鼠标选中带有控制台应用(Console application)标签的图标，见右图。



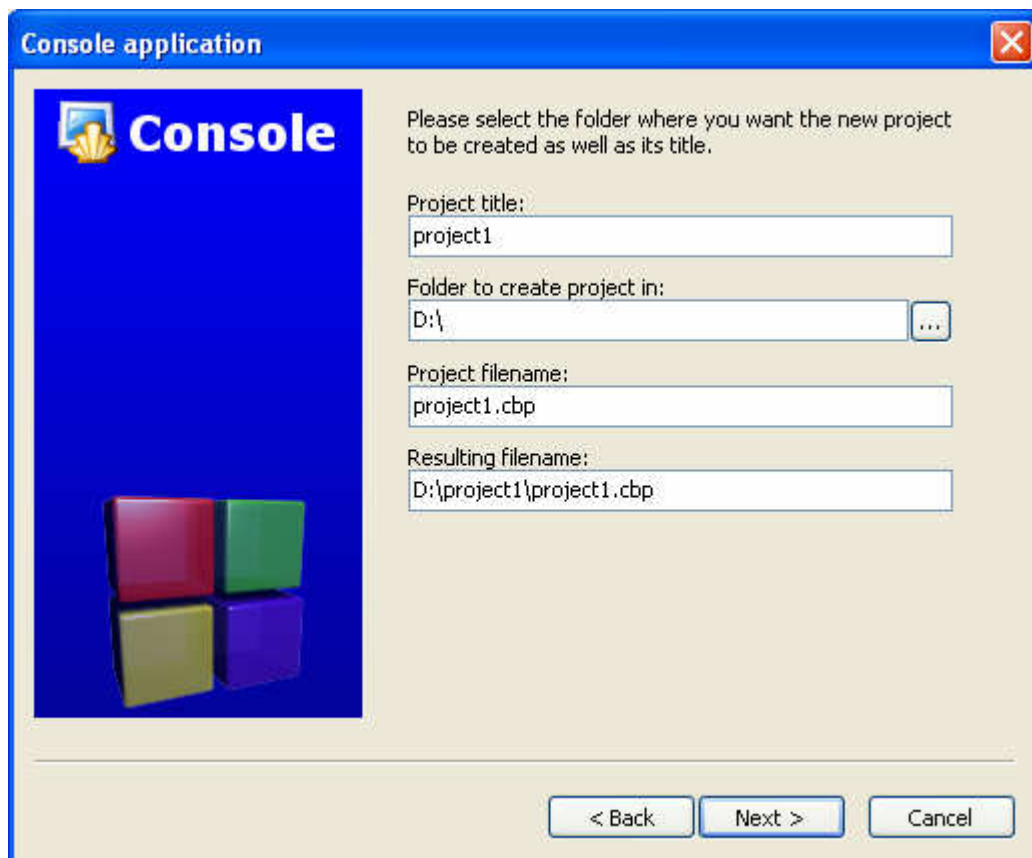
再选择右侧的Go按钮，这样会弹出一个新的对话框，见下图。



再选择Next按钮进入下一步，弹出一个对话框，见下图。

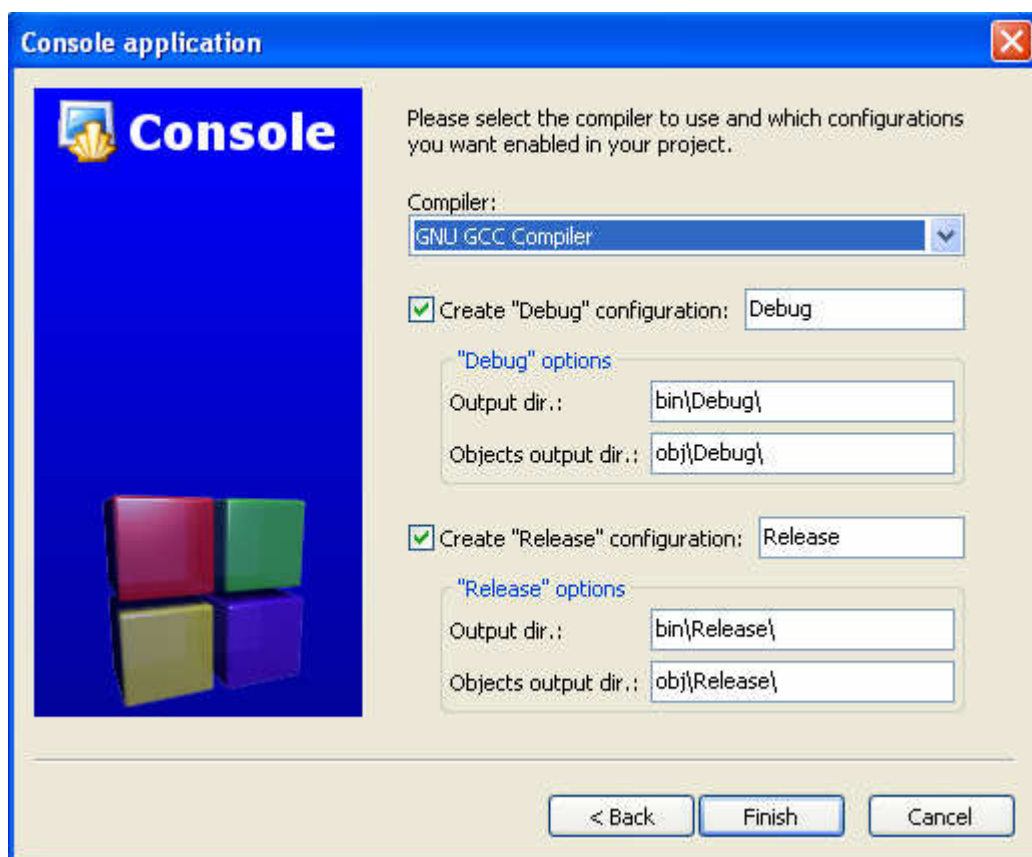


在弹出的对话框中有C和C++两个选项，选择C++表示编写C++控制台应用程序，选择C表示编写C控制台应用程序。这里以编写C++程序为例，因此选择C++，见上图。接下来选择下方的Next按钮进入下一步，又弹出一个对话框，见下图。





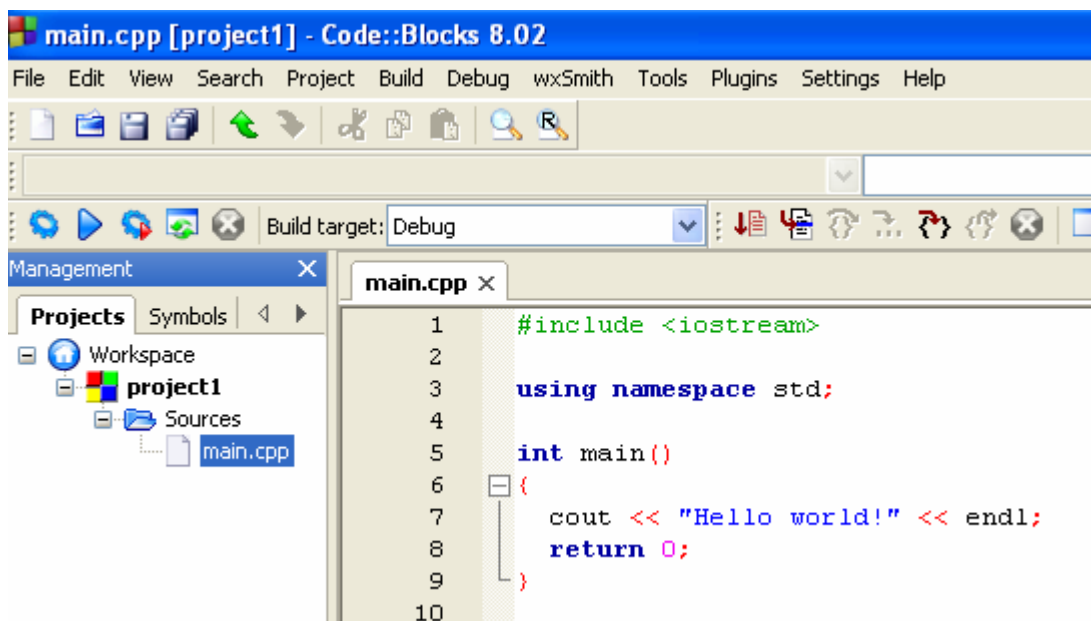
The screenshot shows the 'Console application' dialog box with the title bar. On the left is a blue panel with a 'Console' icon and four colored squares (red, green, yellow, purple). The main area has a light beige background. The text 'Please select the folder where you want the new project to be created as well as its title.' is at the top. Below it are four input fields: 'Project title:' with 'project1', 'Folder to create project in:' with 'D:\', 'Project filename:' with 'project1.cbp', and 'Resulting filename:' with 'D:\project1\project1.cbp'. At the bottom are three buttons: '< Back', 'Next >', and 'Cancel'.

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。



The screenshot shows the 'Console application' dialog box with the title bar. On the left is a blue panel with a 'Console' icon and four colored squares (red, green, yellow, purple). The main area has a light beige background. The text 'Please select the compiler to use and which configurations you want enabled in your project.' is at the top. Below it are several settings: 'Compiler:' with a dropdown menu showing 'GNU GCC Compiler', a checked checkbox 'Create "Debug" configuration:' with a 'Debug' text box, a section titled '"Debug" options' with 'Output dir.:' set to 'bin\Debug\' and 'Objects output dir.:' set to 'obj\Debug\' (both in text boxes), another checked checkbox 'Create "Release" configuration:' with a 'Release' text box, and a section titled '"Release" options' with 'Output dir.:' set to 'bin\Release\' and 'Objects output dir.:' set to 'obj\Release\' (both in text boxes). At the bottom are three buttons: '< Back', 'Finish', and 'Cancel'.

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。

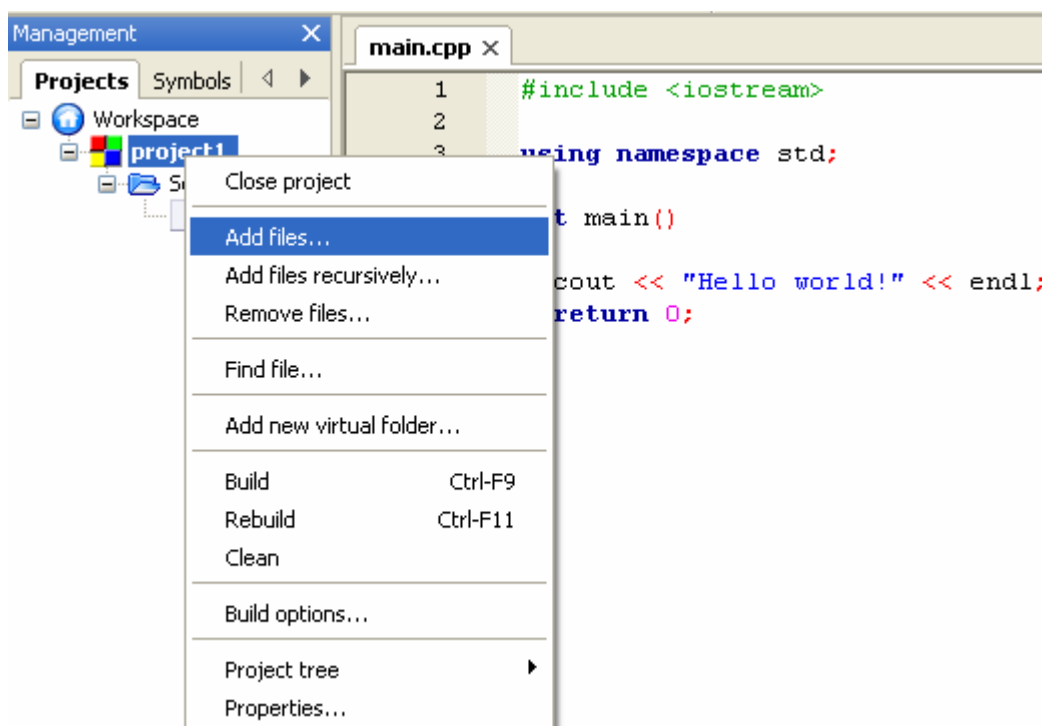


### 3.2 添加和删除文件

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。

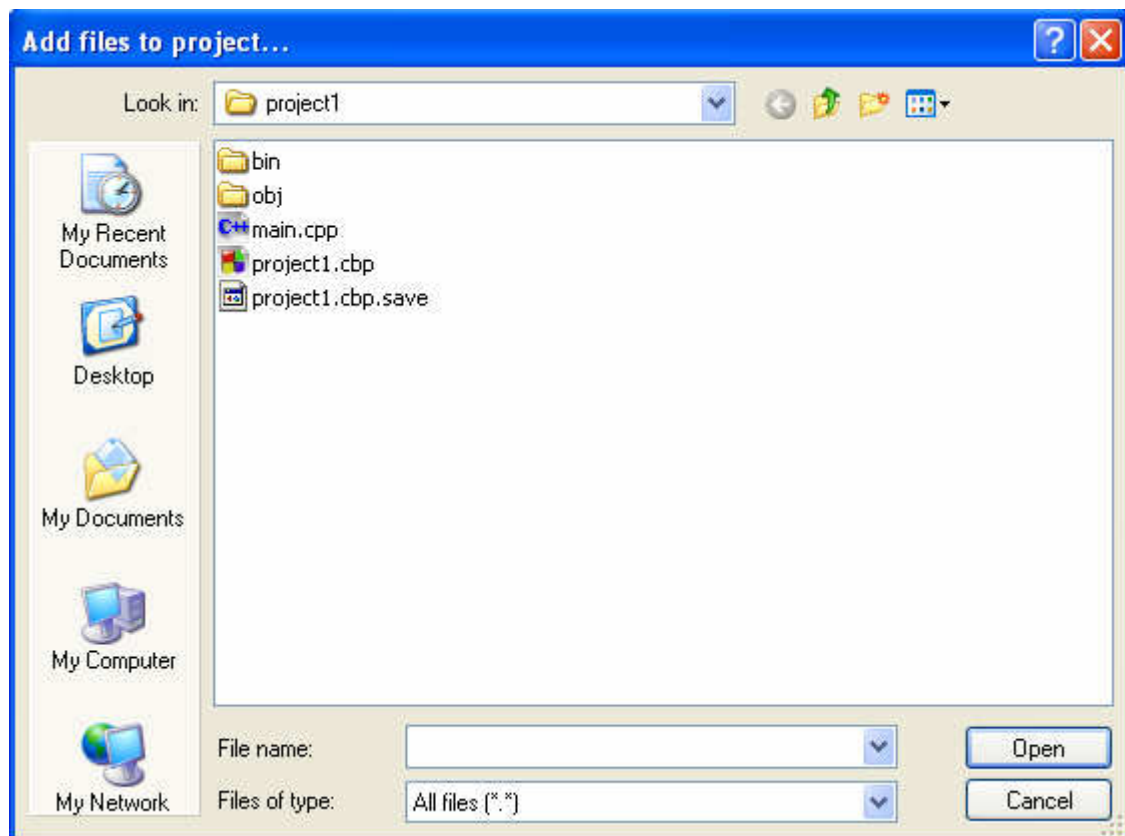
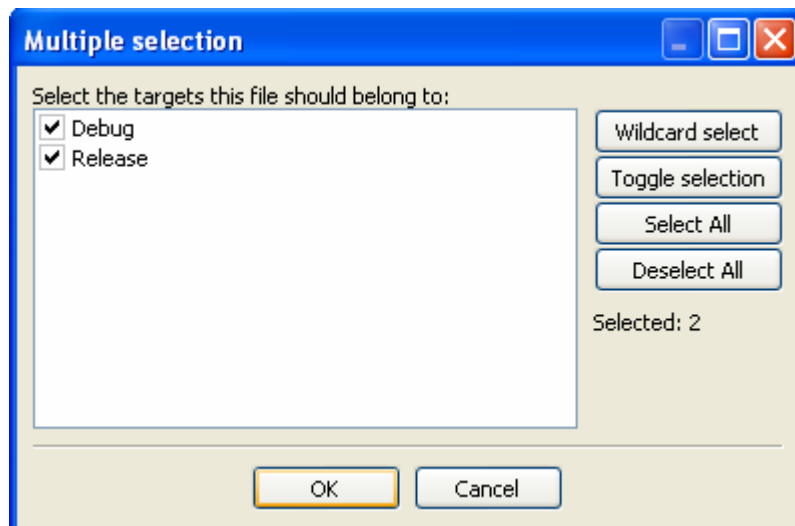
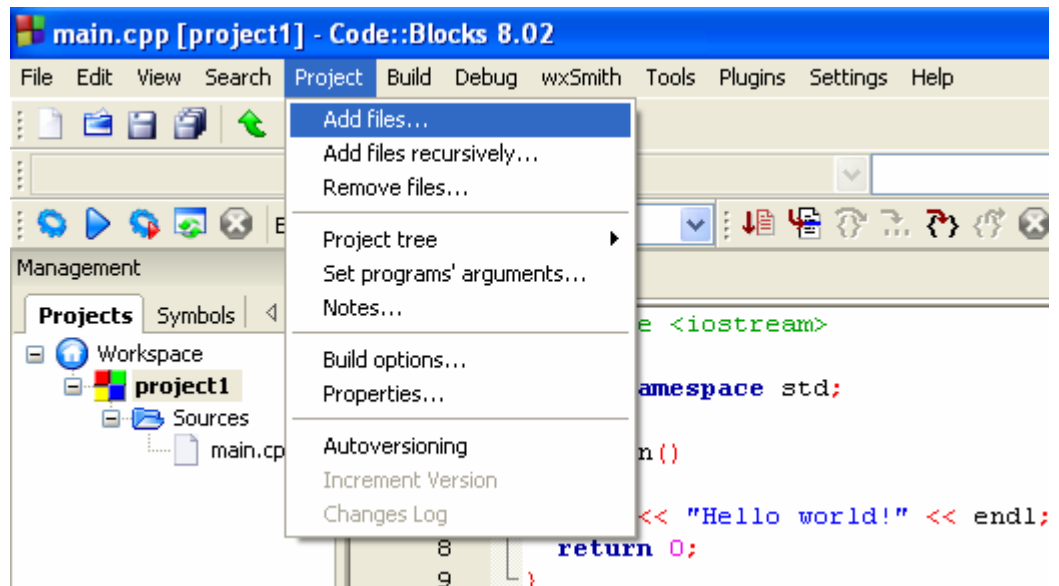
#### 3.2.1 添加文件

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。

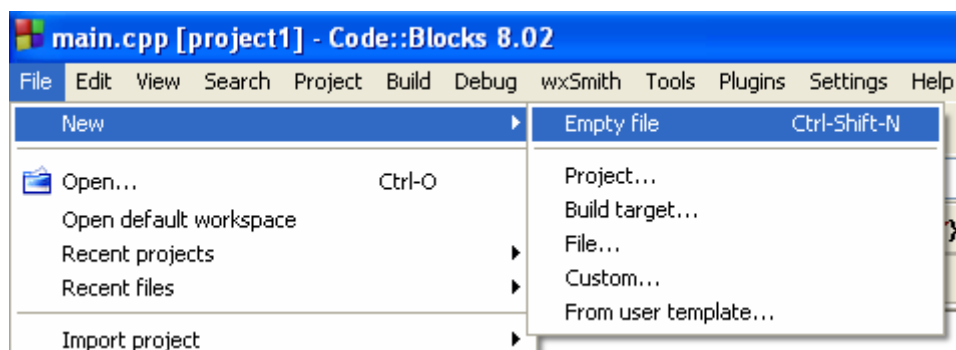
用鼠标点击上面介绍的菜单中Add files...按钮，会弹出一个对话框，见下图。




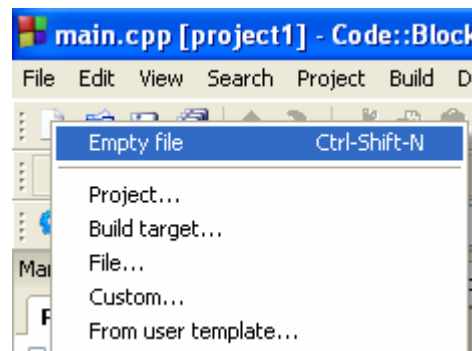
选择要添加的文件，然后选择右下角的Open按钮，见上图。

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。

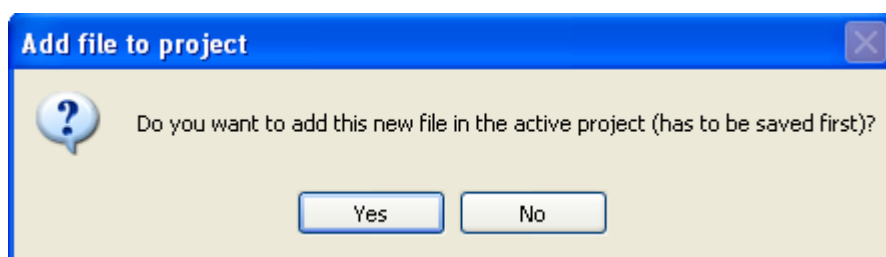
如果此时存储设备上没有我们需要的文件，则需要自己创建一个。创建新文件的方法很多，可以选择File菜单，从下拉菜单中中选择New，通过向右的导向箭头打开New的子菜单，再从这个下拉菜单中选择按钮Empty file Ctrl-Shift-N，见右图。



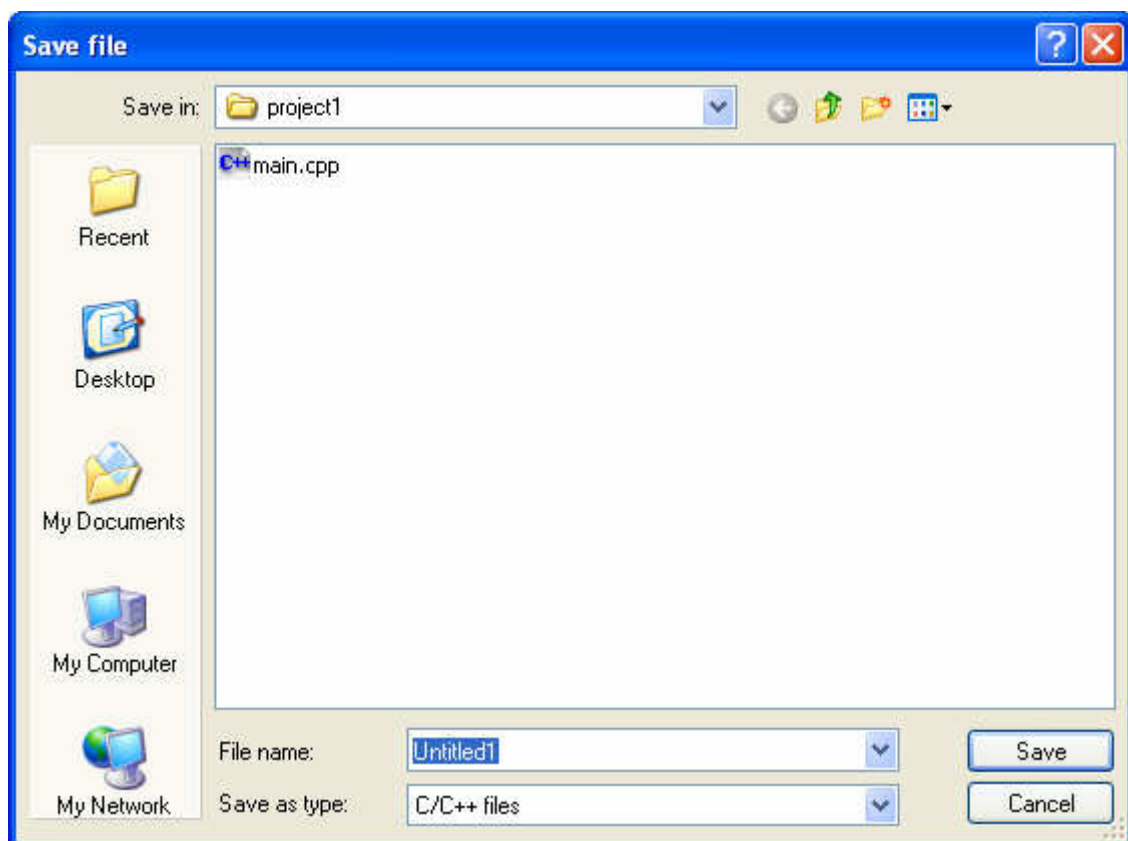
也可以选择这个图标，用鼠标点击一下，也会弹出一个菜单，从弹出的菜单中选择Empty file Ctrl-Shift-N，见右图。



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。



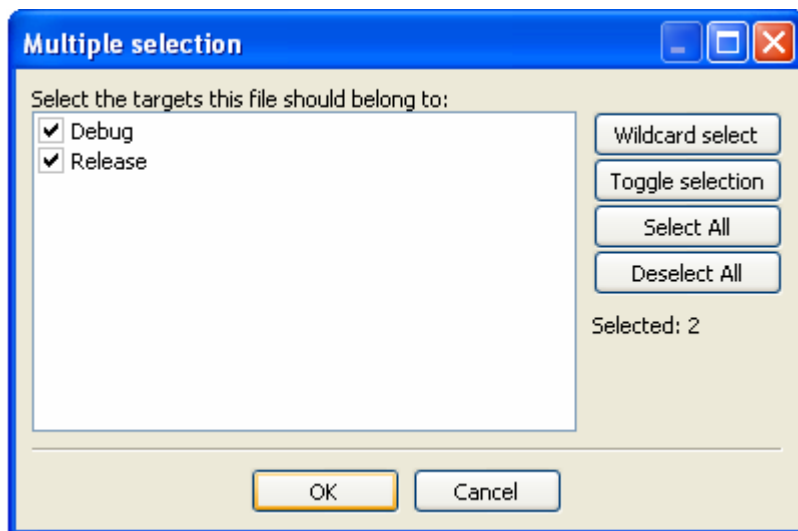
如果您选择Yes，则该文件会被添加到工程中，选择No，则此文件不会被添加到工程。如果您选择了Yes，则会弹出一个新的对话框，让您给新建的这个文件命名，见右图。



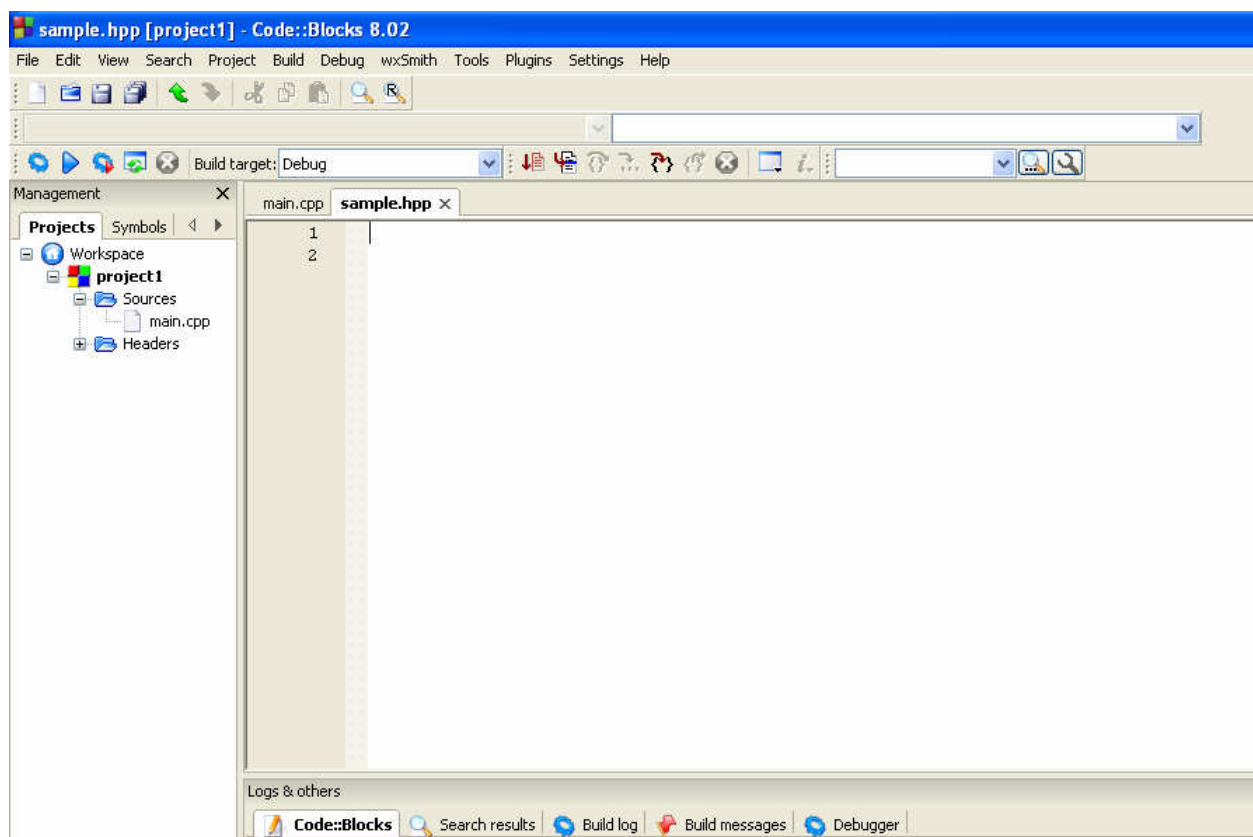


我们假设把这个文件取名为sample.hpp，把它保存到新建的project1文件夹下面。

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。

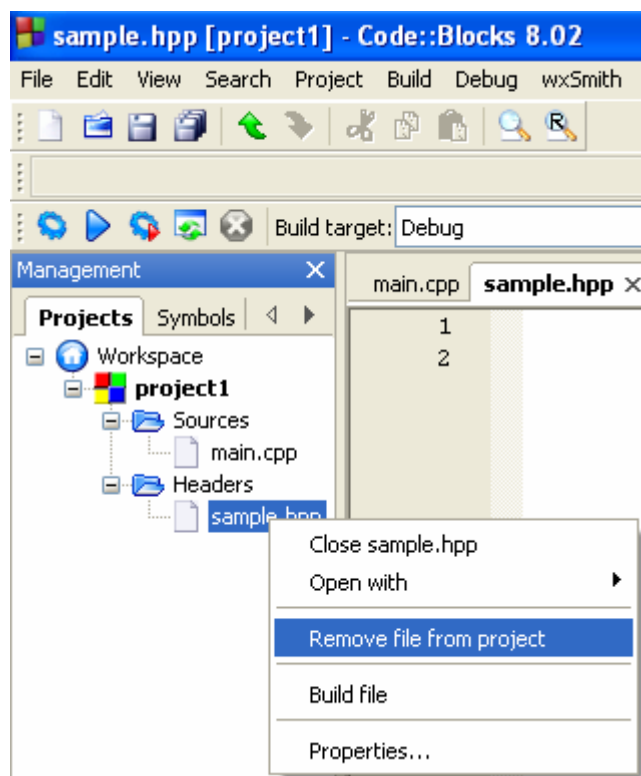


一个目标文件是编译后的文件，可以为debug或者release，debug版本的目标文件允许您使用调试器对该文件进行测试。一般而言，debug版本的目标文件通常较大，因为它包含了一些用于测试的额外信息，release版本的目标文件一般较小，因为它不包含调试信息。

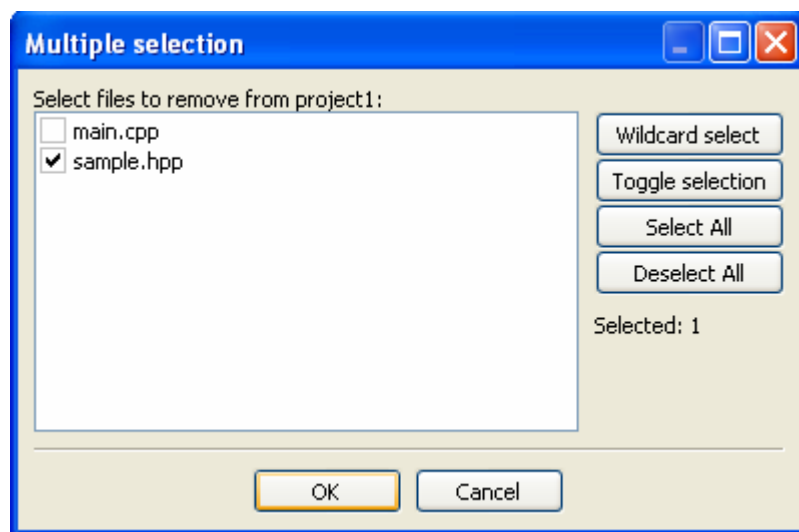
### 3.2.2 删除文件

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。

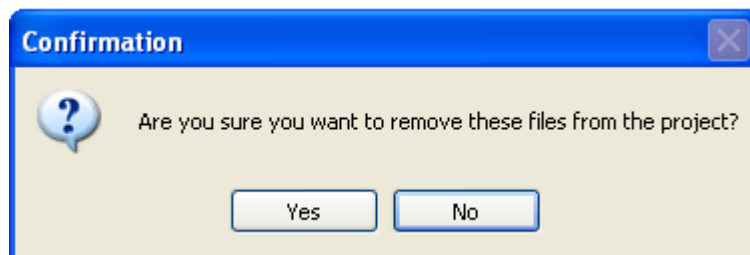
删除这个文件方法很多，这里简要说明两种，一种方法是用鼠标点击图标  Headers 上的 ，这样Headers就会展开，选择sample.hpp，按下鼠标右键弹出的菜单中选择按钮Remove file from project，则sample.hpp就不在隶属于project1了，见下图。



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。

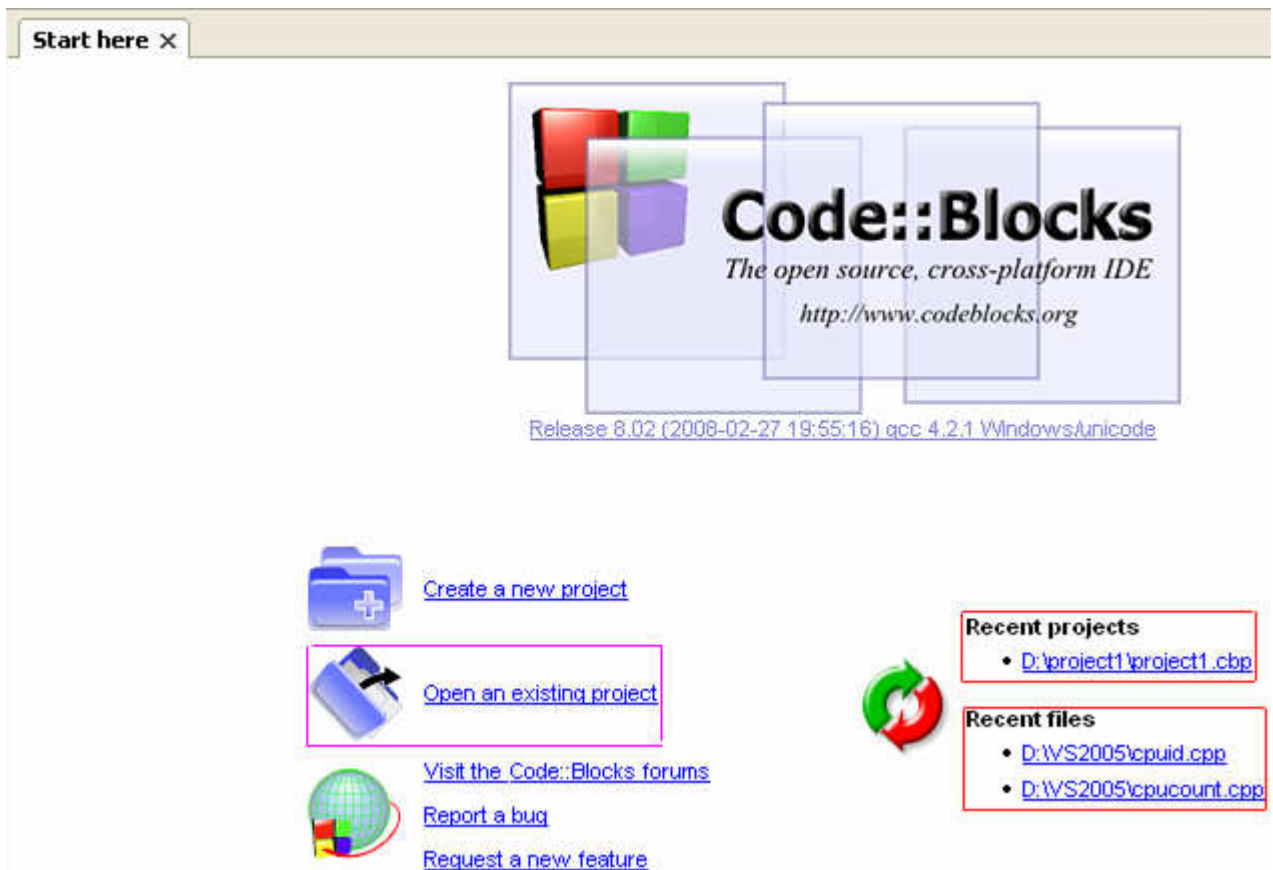


### 3.3 编辑文件

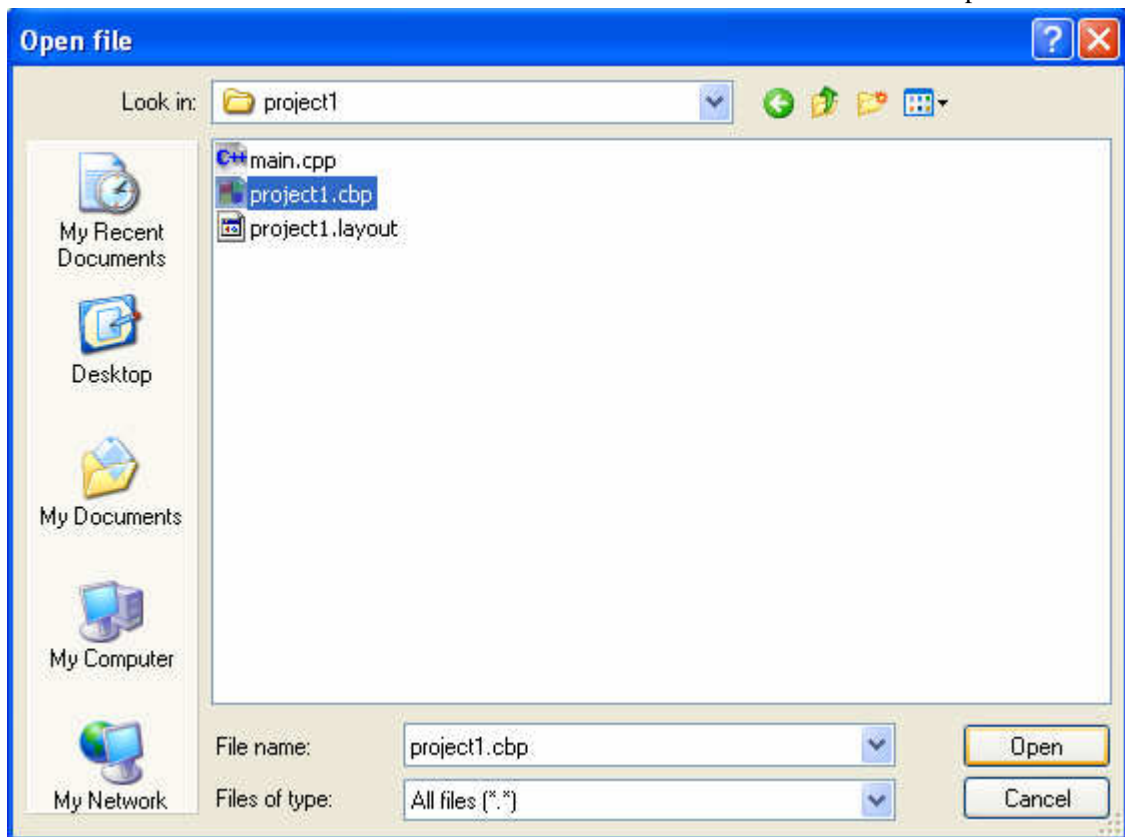
编辑已经存在的文件前需要首先打开这个文件。打开文件的方法很多，由于Code::Blocks会默认记住打开的文件路径，进入Code::Blocks主界面后可以看到最近用Code::Blocks打开过的工程以及最近打开过的文件。


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。

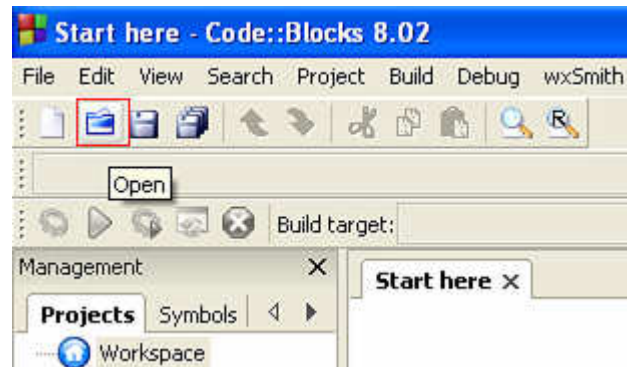




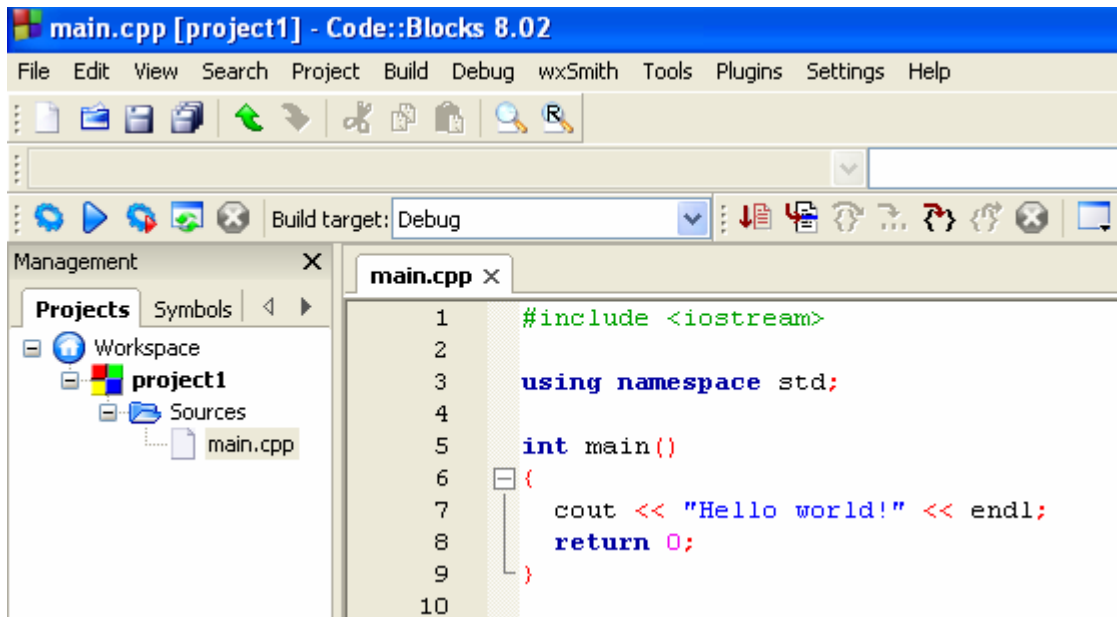
也可以选择带有标签Open an existing project的图标，见右图中用粉红色方框框起来的部分，用鼠标点击它则会弹出一个对话框见右下图，让您选择要打开的工程，选中对应文件名，点击Open按钮。




还可以从主界面Edit主菜单下面的图标，见右图。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。

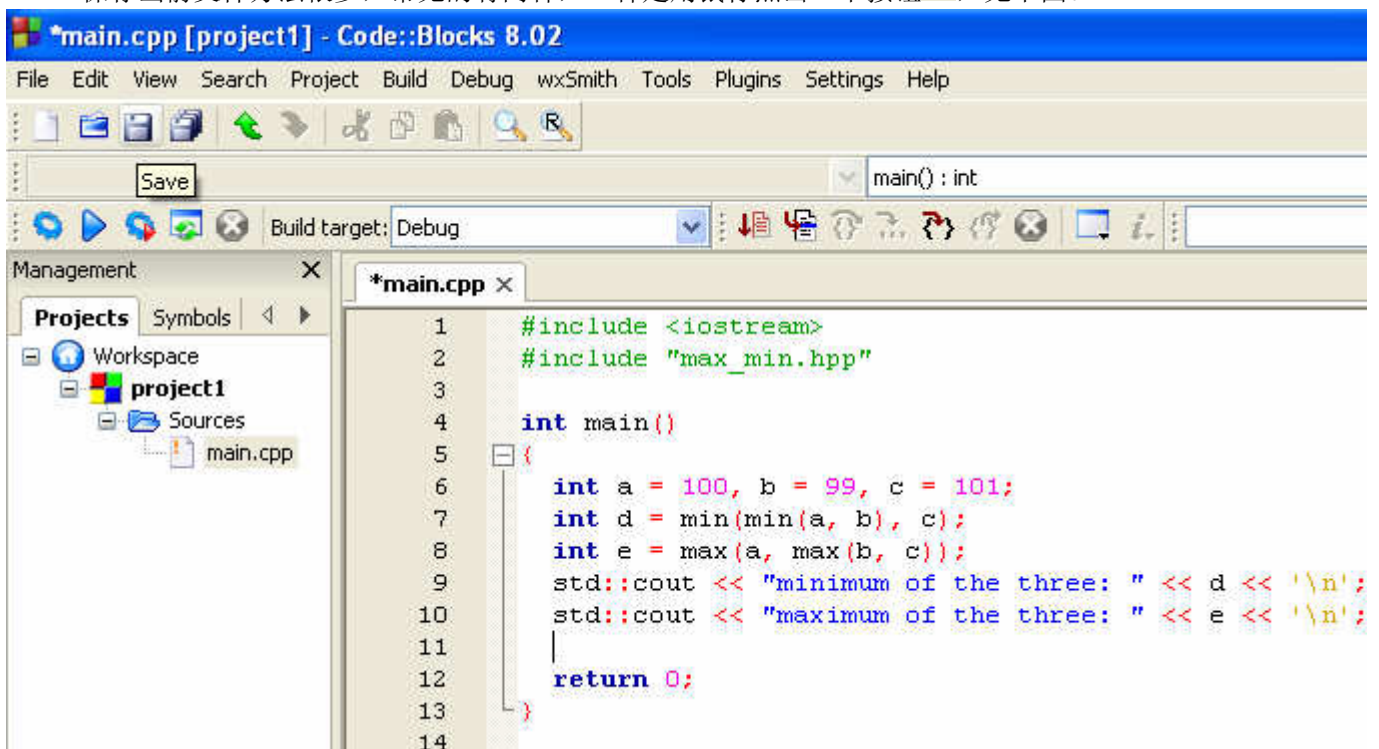



打开一个工程后再打开工程中的某个文件，则显示该文件的源代码，见下图。



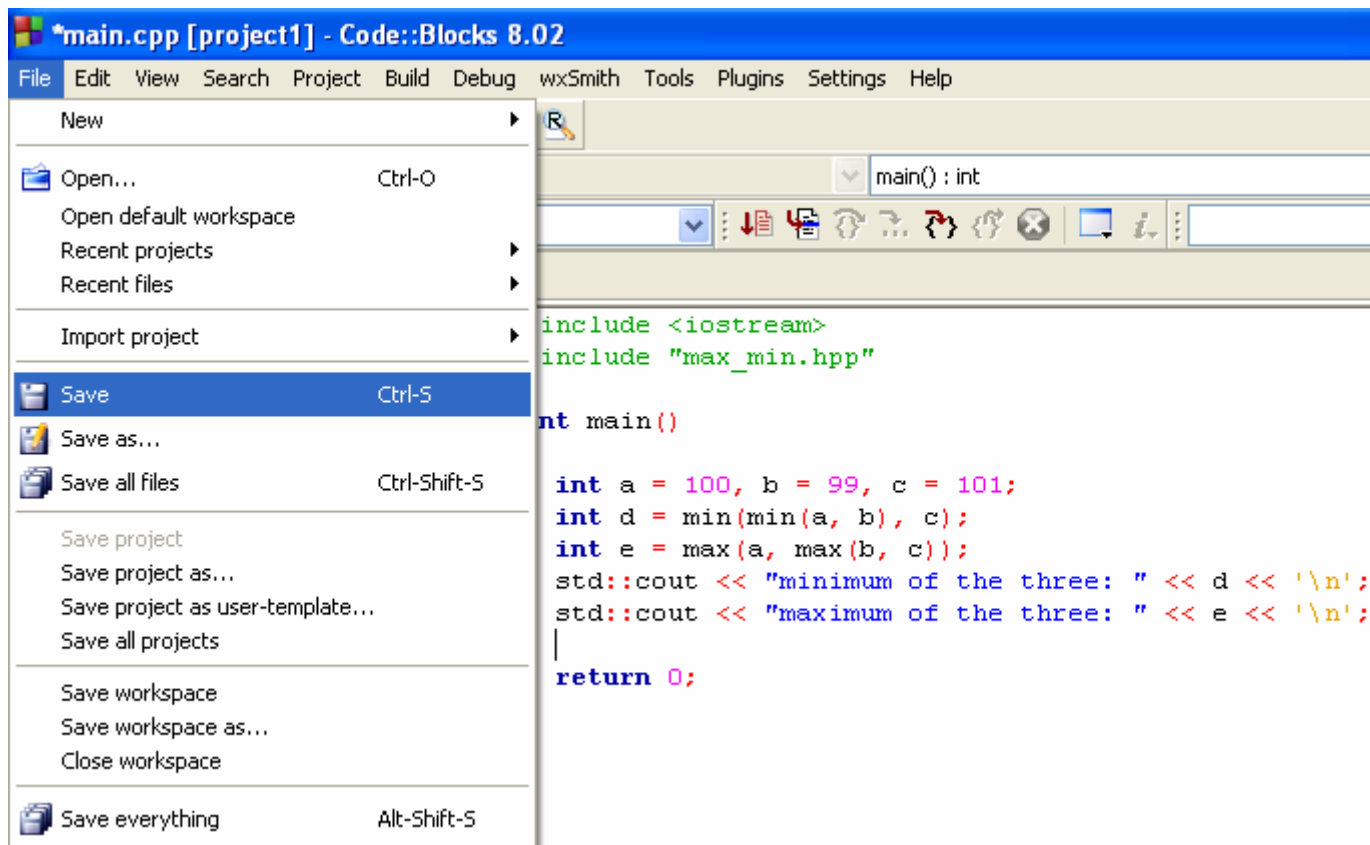
假设我们想修改此文件的源代码，让它求出三个整数中的最小值，则可以这样做。首先，我们修改main.cpp的源代码，修改完毕后，我们保存当前文件。






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。

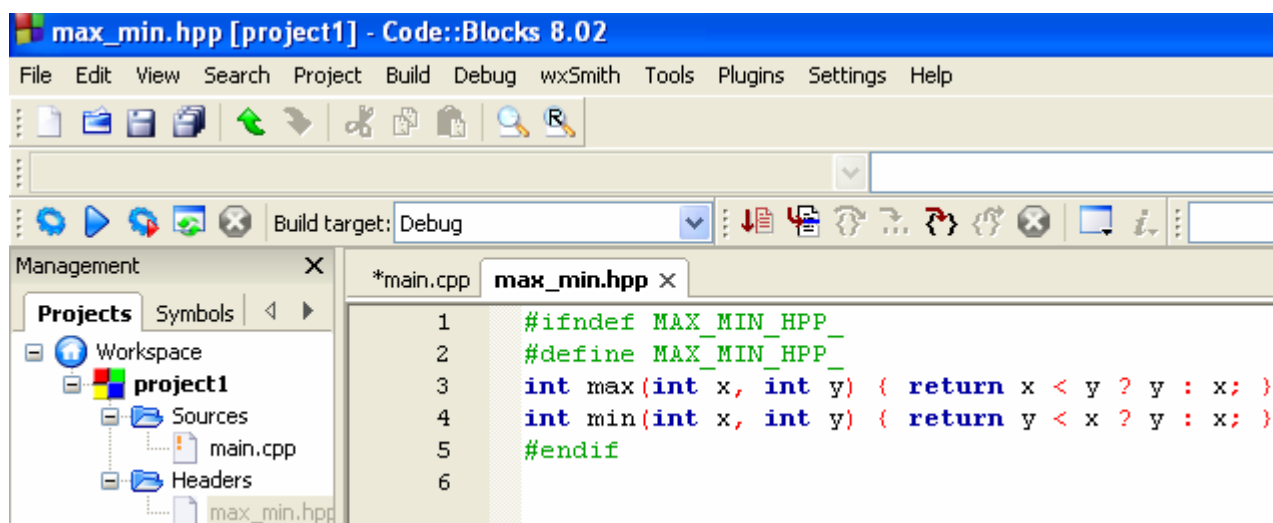


另外一种方法是在主菜单File的下拉菜单中选择按钮  Save

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。

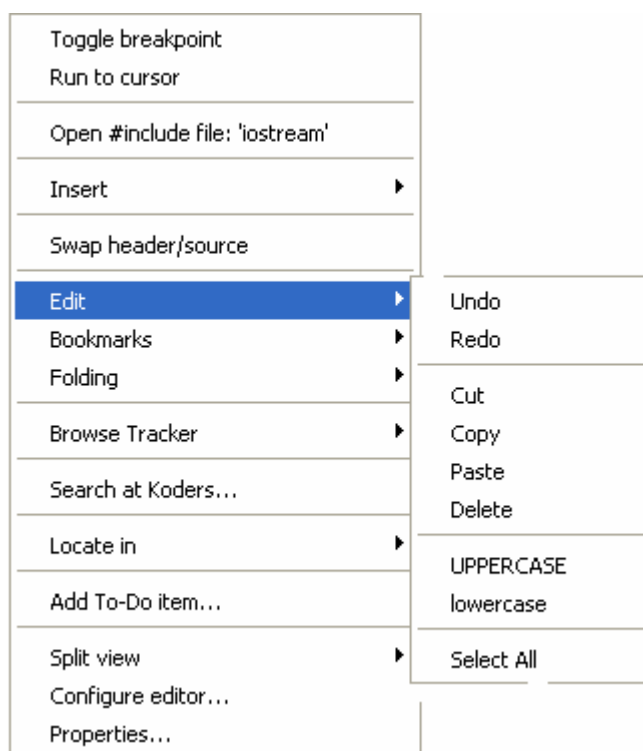
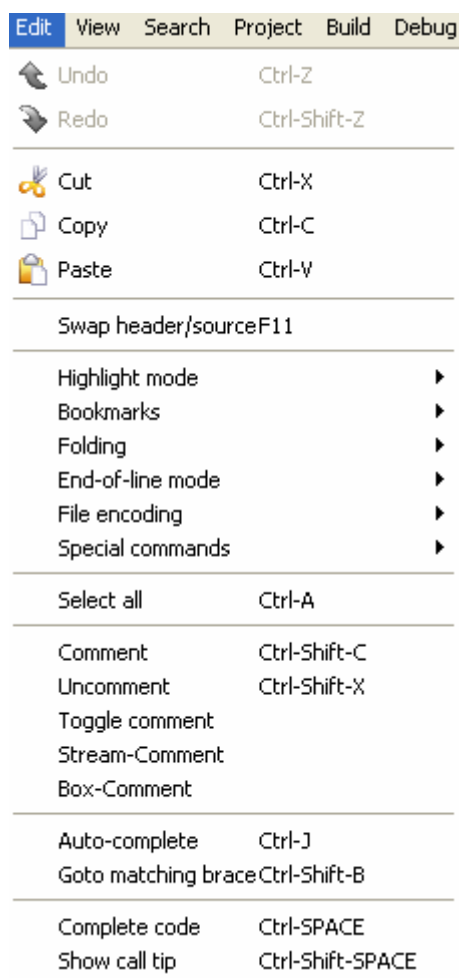
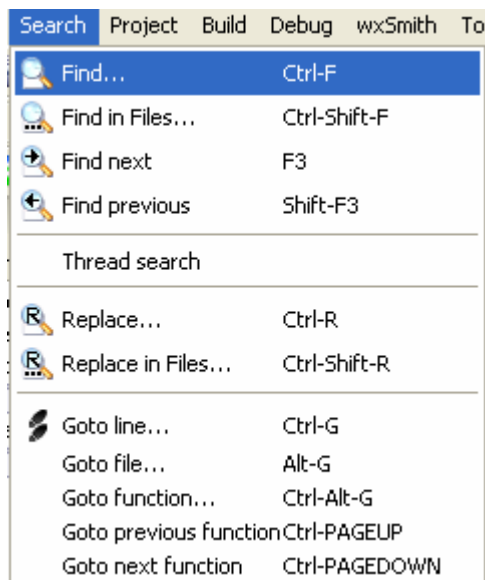


注意到，编辑后的main函数源代码中添加了一行#include“max\_min.hpp”，max\_min.hpp是一个头文件，该头文件包含了两个函数，一个是max，一个是min，分别用来求出两个整数的较大和较小者。现在我们需要建立这个头文件，并把它添加到project1中。添加文件到工程的方法前面已经有所论述，在此不再重复，假设我们已经编辑完毕max\_min.hpp头文件，并添加到project1中，如上图。

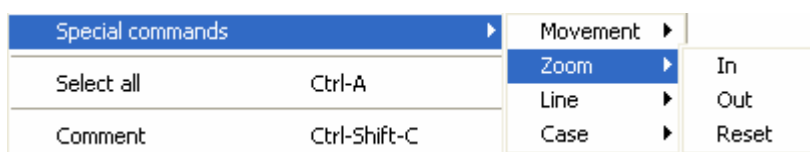
如果编写的源代码较多，有时候可能需要查找和替换某个字符串。Search主菜单的下拉菜单中有几个按钮用来对文件内容进行操作，可以查找和替换等，见右上图。



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。



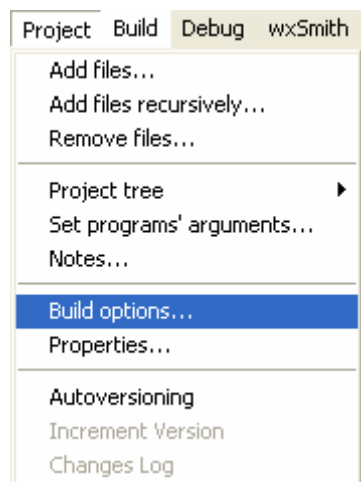
在这些编辑选项中有一项Special commands其中的Zoom(缩放)有时很有用。几个人围在一台电脑屏幕前研究程序的时候，由于字体太小看不清不必重新设置字体大小，可以用子菜单下In和Out按钮方便的缩放。当然了，最简洁的方法是一手按住Ctrl键，一手转动鼠标滚轮(当您使用有滚轮的鼠标时)对代码进行缩放，最后再找到子菜单下的Reset按钮恢复原貌就可以了。



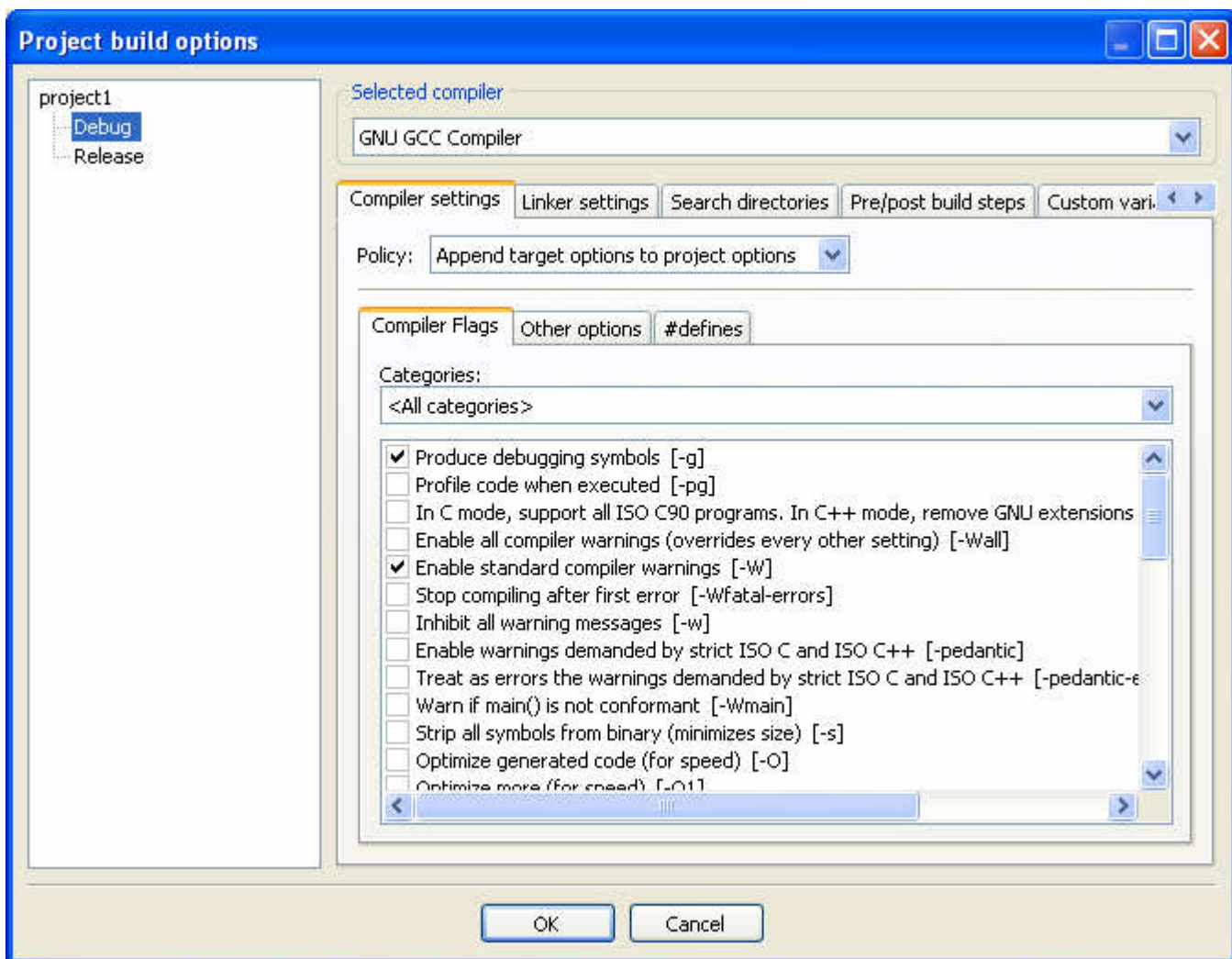
有时候可能需要把一个文件内的部分代码拖动到别的位置，您可以使用Edit菜单下的按钮Cut(剪切)和Paste(粘贴)完成所需，更简洁的办法就是选中后一手按住Alt键，一手用鼠标左键拖动这些代码到目的位置就可以了。

### 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，见右下图。

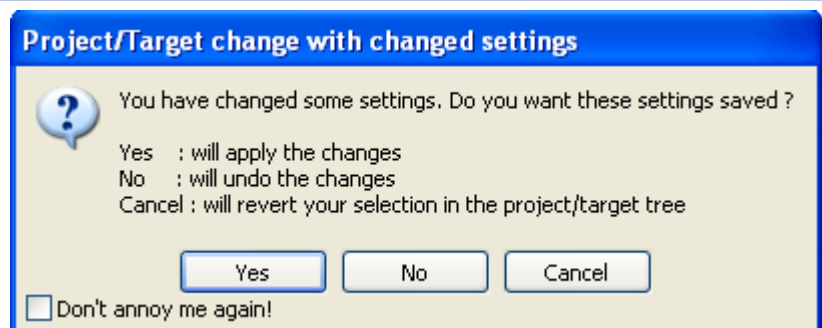
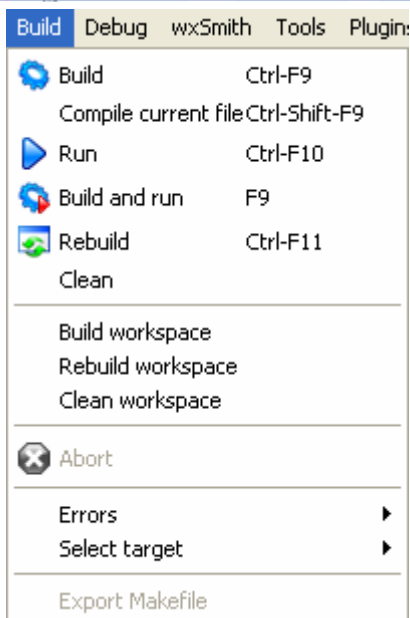
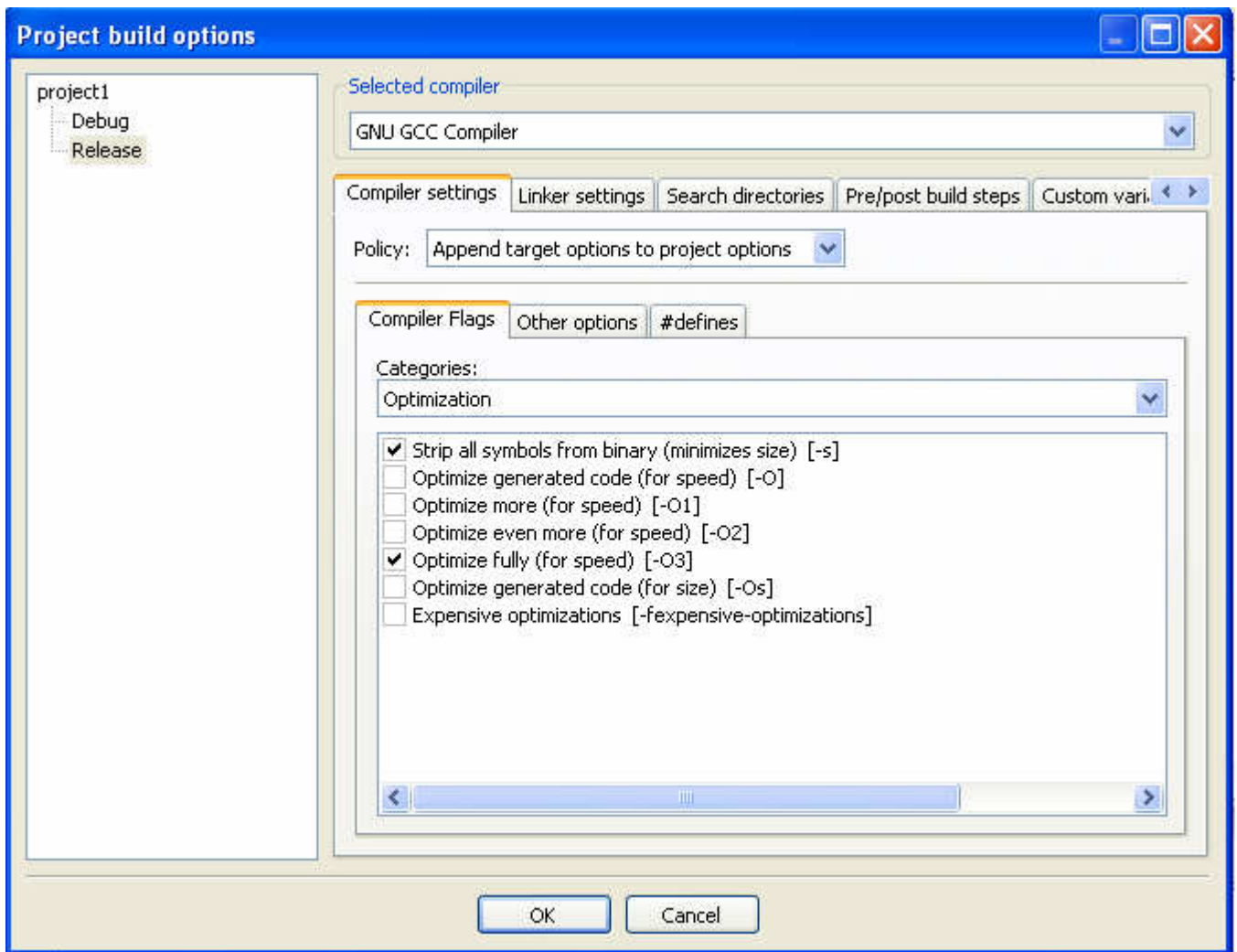


用鼠标点击Build options...按钮会弹出一个关于project1的对话框，有两个类别，debug和release，首先配置debug选项，一般而言，只关注Compiler Flags菜单下的两项，Producing debugging symbols [-g]和Enable standard compiler warnings [-W]。前者表示产生调试信息，后者意味着给出标准的编译警告信息，分别打勾，见下图。

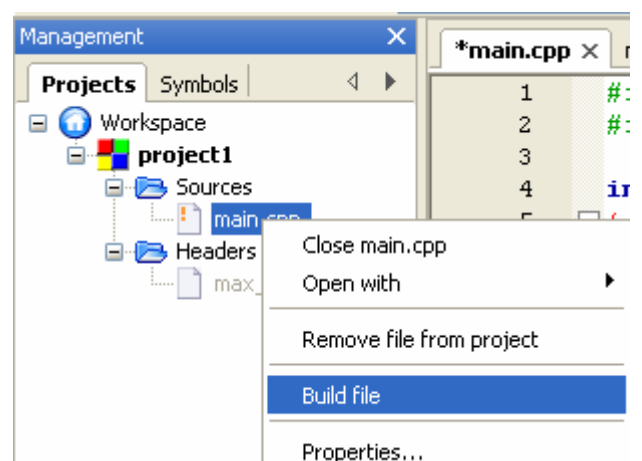





然后配置Release，选择子菜单Compiler Flags下的Optimization选项，对于普通的应用，选择两项足矣，分别是Strip all symbols from binary (minimizes size) [-s]和Optimize fully (for speed) [-O3]，这两项前面打勾，见下图。






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。



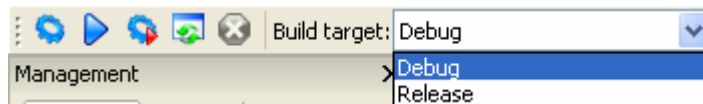
两个菜单中按钮图标相同的，功能也相同。表示编译当前工程，表示运行编译成功的文件，表示编译并运行。如果编译当前文件，可以使用Build下拉菜单中的按钮Compile current fileCtrl-Shift-F9，也可以展开左侧的工程目录树，移动鼠标到需要单独编译的文件上面，利用鼠标右键弹出菜单中选择Build file按钮，见右图。


如果一次编译不成功，修改后重新编译可以使用按钮

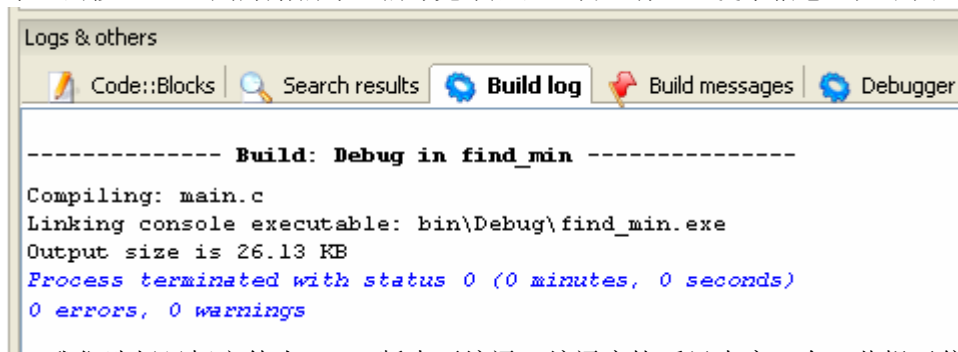


Rebuild，它代表重新编译，如果想清楚上次编译的目标文件，可以使用Build的下拉菜单中的Clean按钮。

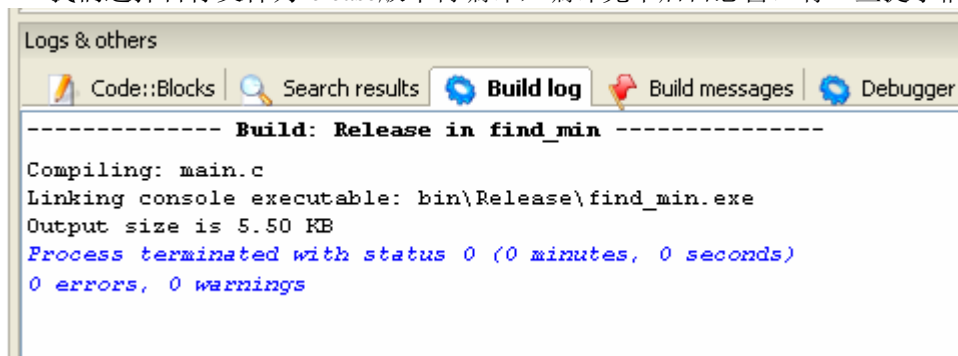
根据客观需要，我们可以把目标文件编译成debug或者release版本，用鼠标从快捷菜单上的Build target小窗口进行选择即可，见下图。




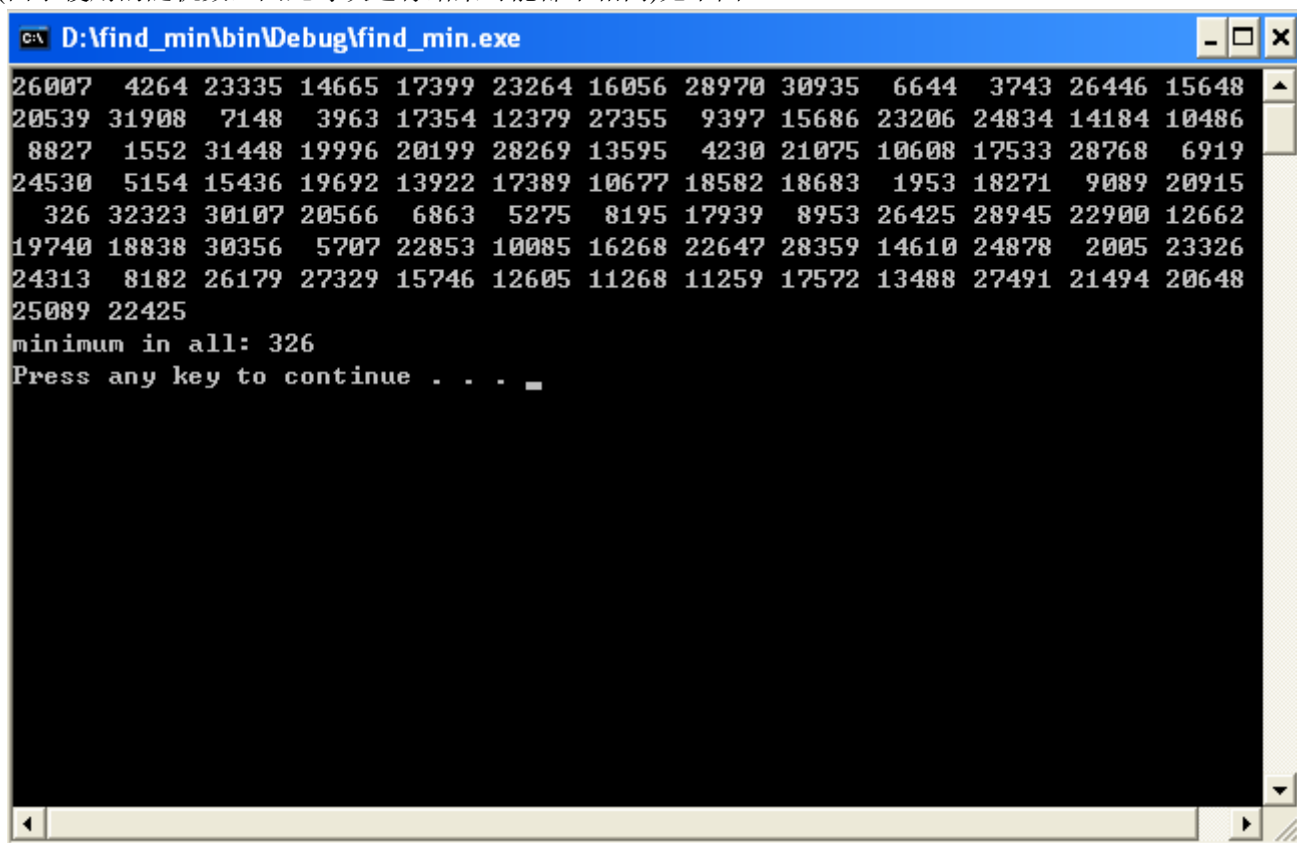
作为例子，我们用C语言编写一个求一个随机整数集中最小值的程序。首先建立工程find\_min，编写程序(程序源代码见本章的下一小节3.5.1)，然后编译。首先选择编译目标文件为debug版本，用鼠标点击快捷菜单上的按钮，则开始编译，编译完毕后日志窗口有一些提示信息，见下图。



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。



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。

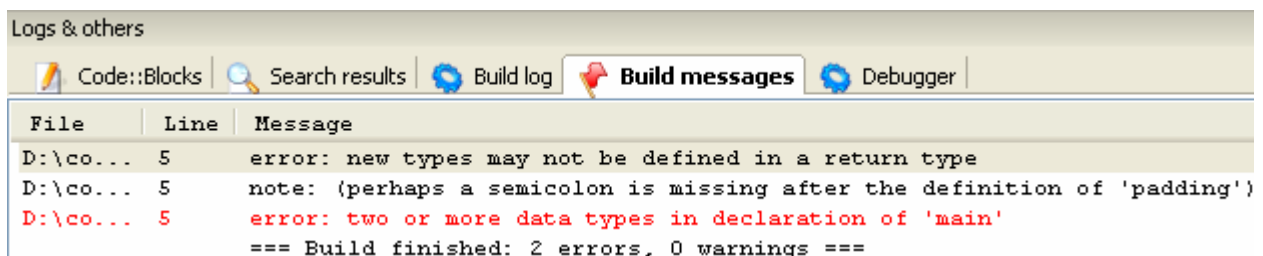
一次编译成功当然最好，但是很多时候不能一次编译成功，这时需要根据给出的出错信息修改源程序，然后重现编译，可能需要反复这个过程，直到编译成功。编译成功说明没有语法错误，但未必没有逻辑错误，程序中的逻辑错误需要您自己检查，当然了，您可以使用调试工具帮助检查逻辑错误。

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.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
```

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

编译提示信息见如下贴图。



上面的提示大致意思是：

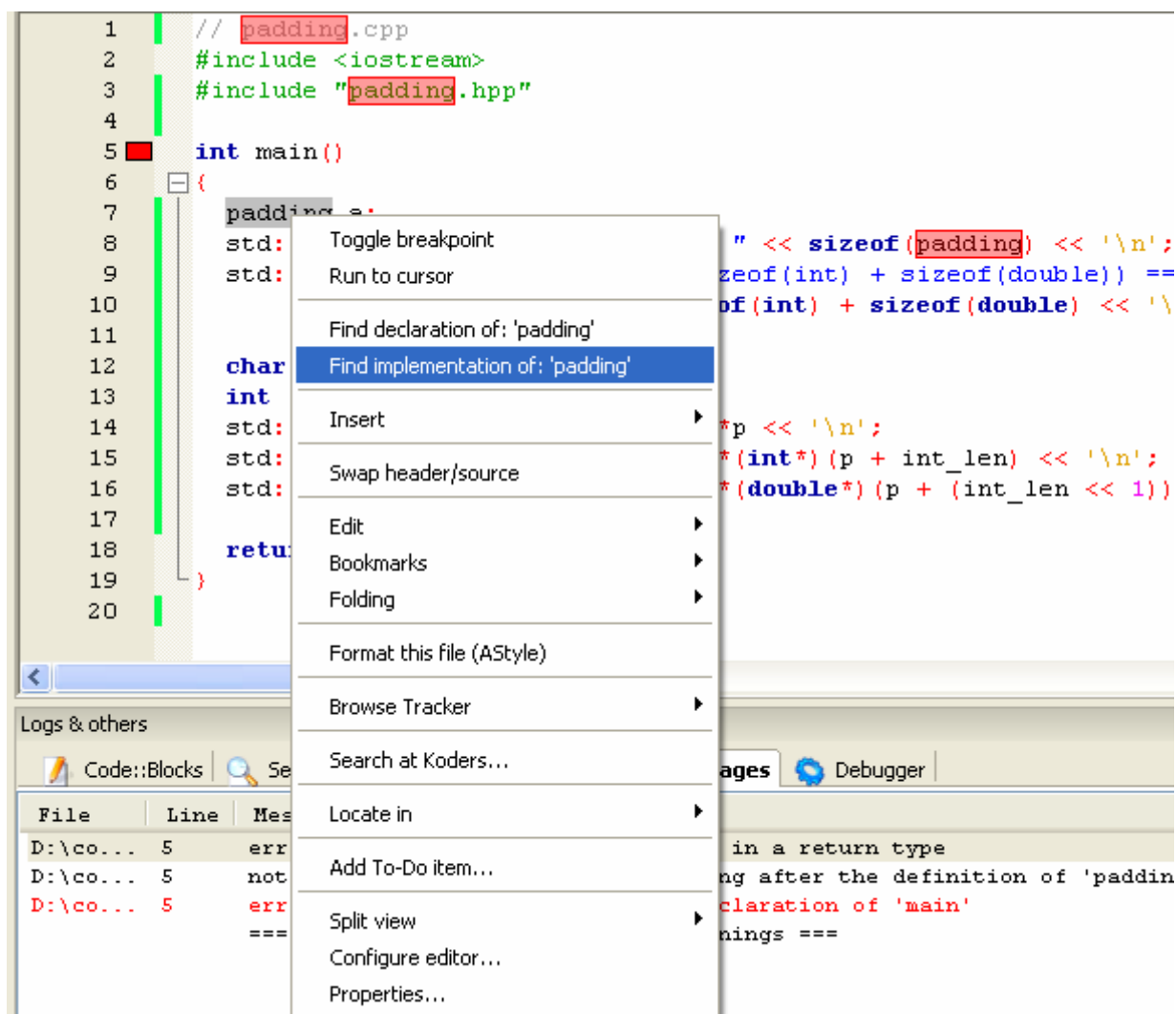
- “出错：在返回类型后新类型必 定义”
- “注意：可能padding定义后少了个分号”
- “出错：main中声明了两个或者更多个类型”

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，

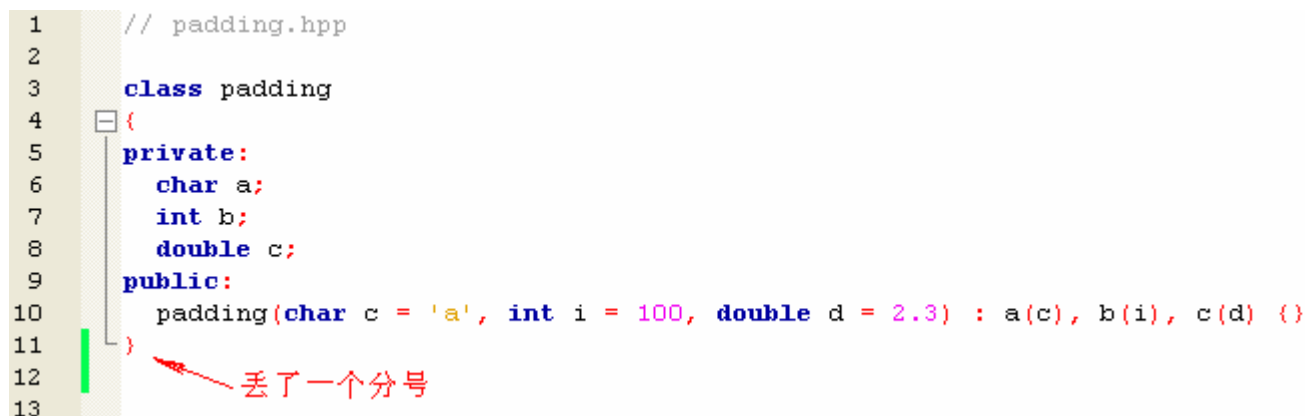
这些错误信息中很大一部分可能没有任何帮助价值，我们需要找出对我们有意义的错误信息。有时我们发现，修改完一个错误重新编译后原来给出的很多错误信息突然变得少多了。



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。

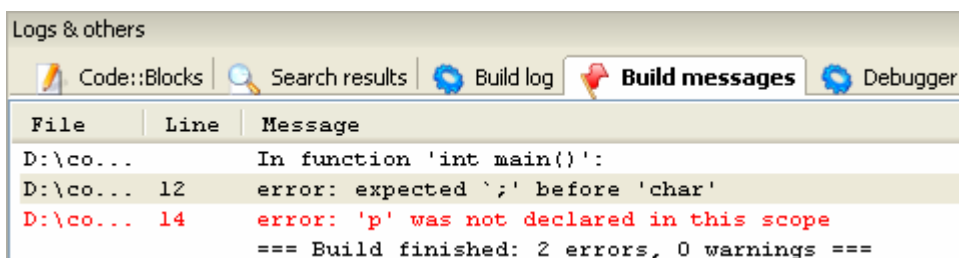
现在开始修改错误。我们首先去找padding的定义，看看是否忘记了一个分号。查找padding的声明或者实现很简单，选中padding，则padding变成了灰色，所有和padding相同的字符窜都变成了红色(注：笔者使用了比较新的Code::Blocks版本，如果您用的版本比较旧，则可能没有这样的功能，颜色可以设置，不一定是红色)，按下鼠标右键在弹出的快捷菜单中选择按钮Find implementation of: 'padding'，见下图。



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

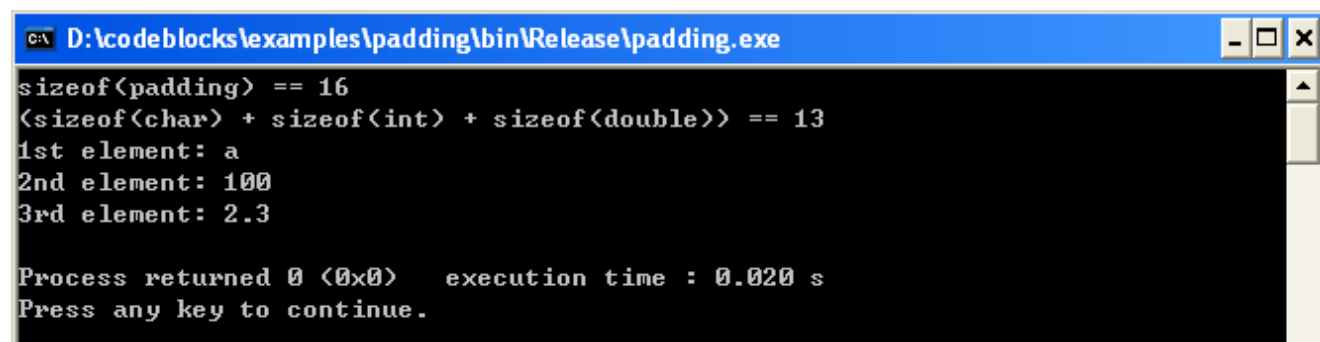
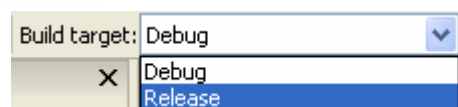
```

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。



**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。

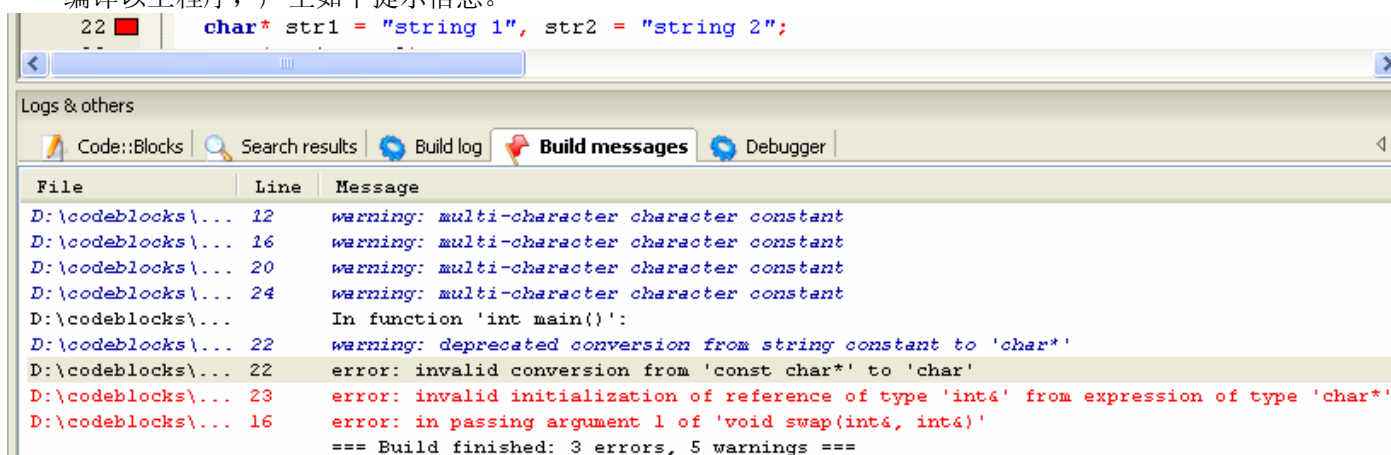
我们接下来再看一个例子。以下程序用来交换两个不同的值, 由于我们并不知道要交换的值的类型, 所以就使用模板, 参数类型用指针, 当交换的两个值是整数的时候进行特化, 参数类型用引用。为了知道调用了那个swap函数进行交换, 在swap函数体里面我们附加输出一点信息。

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 // main.cpp
2 #include <iostream>
3 using namespace std;
4 #include "swap.hpp"
5
6 int main()
7 {
8     int a, b = 1;
9     swap(a, b);
10    cout << a << ' ' << b << '\n\n';
11
12    double c = 3.3, d;
13    swap(c, d);
14    cout << c << ' ' << d << '\n\n';
15
16    char e = 'e', f = 'f';
17    swap(&e, &f);
18    cout << e << ' ' << f << '\n\n';
19
20    char* str1 = "string 1", str2 = "string 2";
21    swap(str1, str2);
22    cout << str1 << ' ' << str2 << '\n\n';
23
24    return 0;
25 }
26
```

```
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
```

编译以上程序，产生如下提示信息。



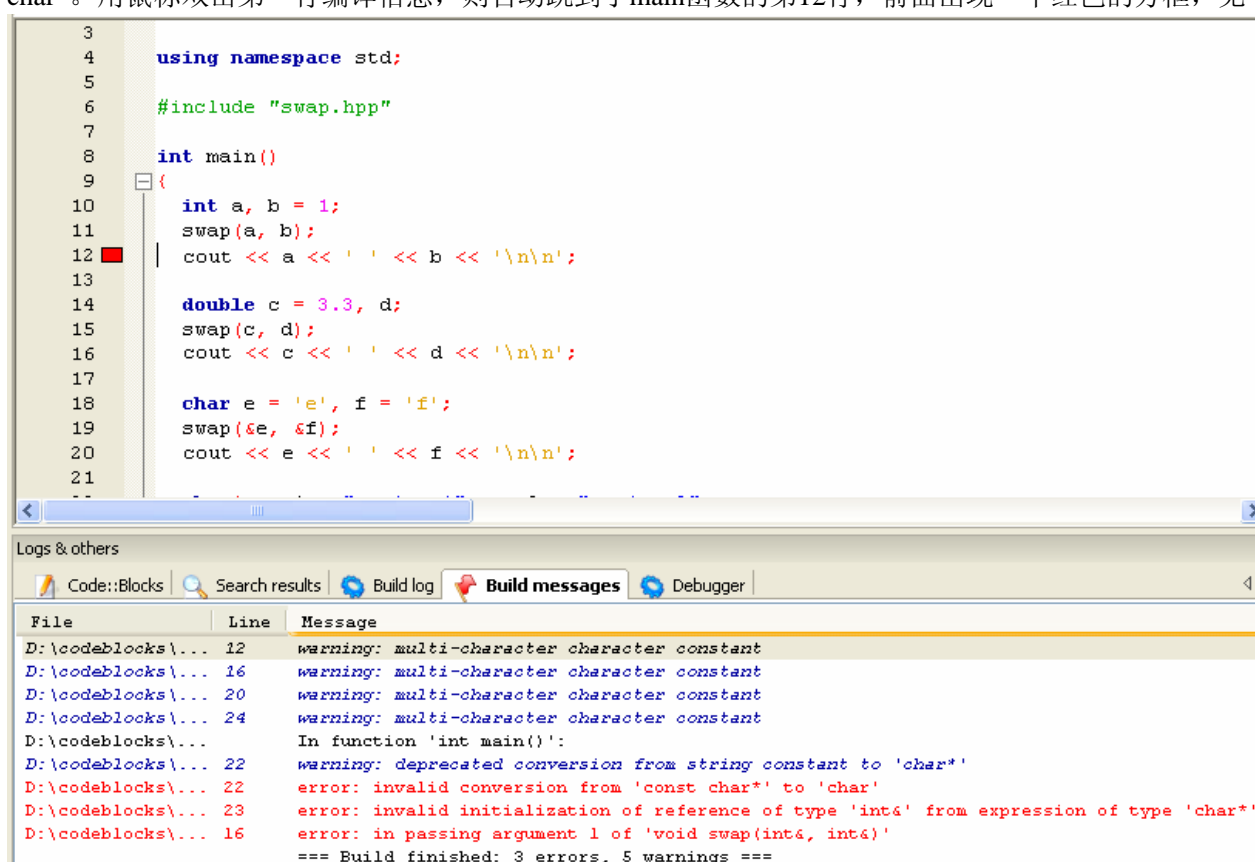
```
22 char* str1 = "string 1", str2 = "string 2";

Logs & others
Code::Blocks Search results Build log Build messages Debugger


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。

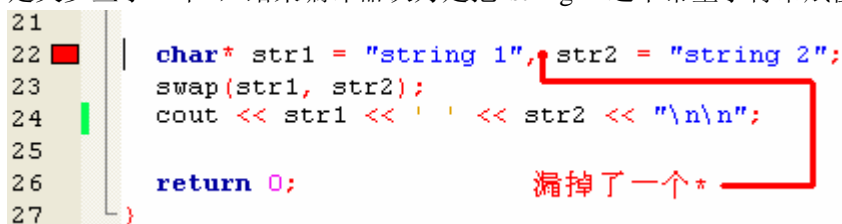
首先我们看警告信息，警告说第12, 16, 20, 24行多个字符常量，第22行，不建议把字符串常量转换成char\*。用鼠标双击第一行编译信息，则自动跳到了main函数的第12行，前面出现一个红色的方框，见下图。



```
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
22 }
```

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串定义少些了一个\*，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。

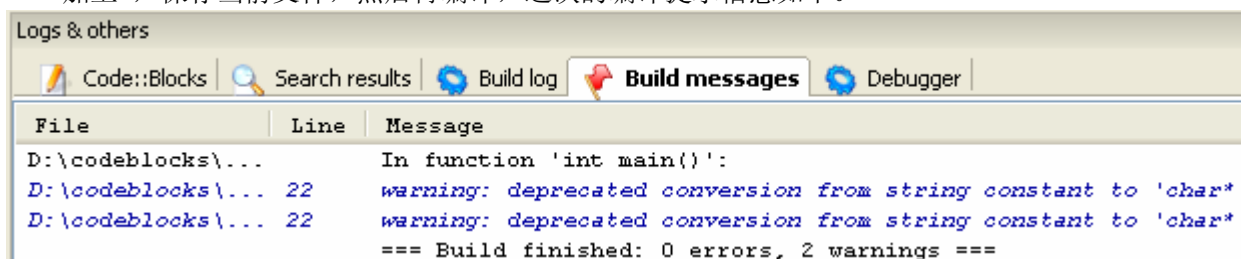


```
21
22 char* str1 = "string 1", str2 = "string 2";
23 swap(str1, str2);
24 cout << str1 << ' ' << str2 << "\n\n";
25
26 return 0;
27 }
```

漏掉了一个\*

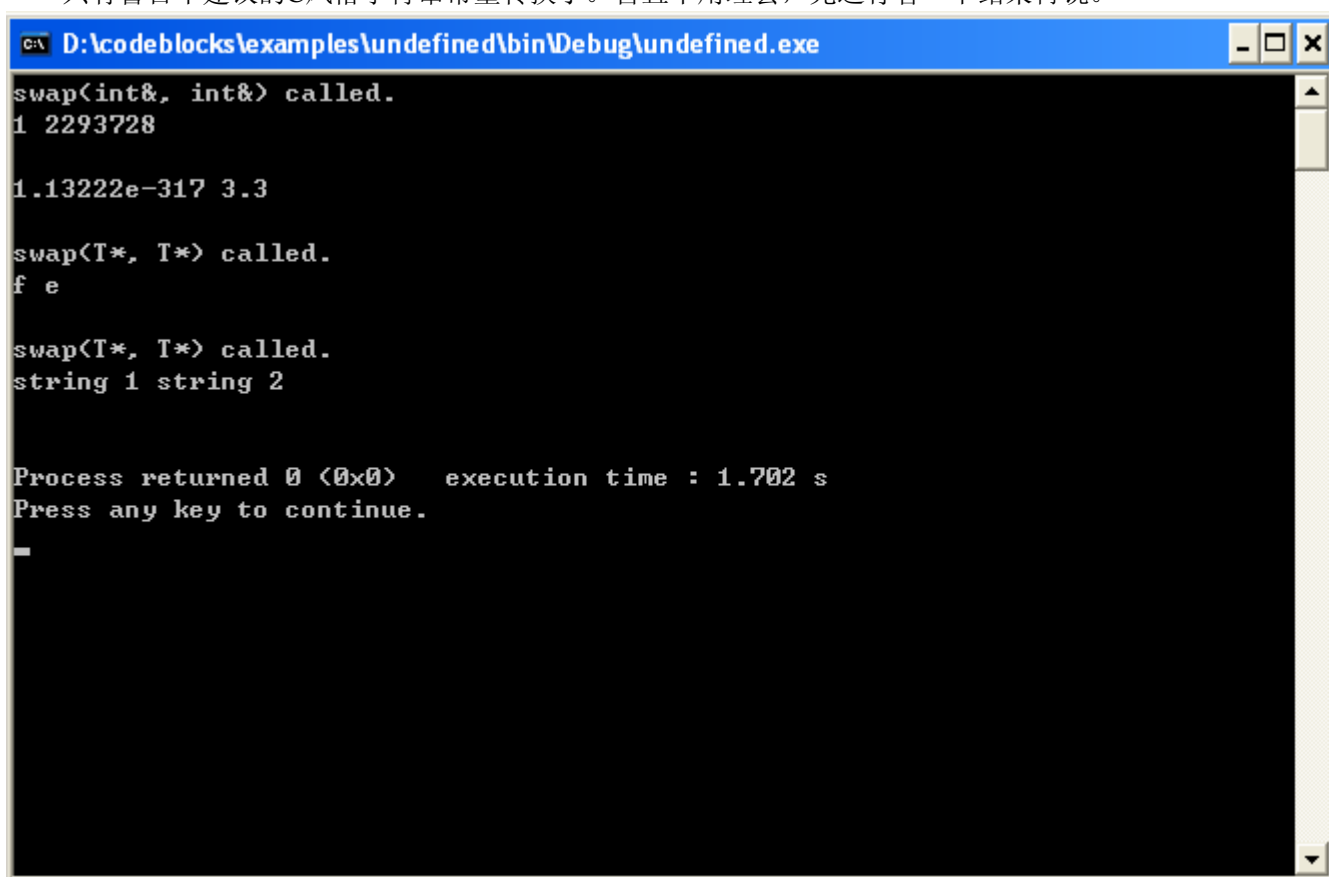


加上\*, 保存当前文件, 然后再编译, 这次的编译提示信息如下。



File	Line	Message
D:\codeblocks\...		In function 'int main()':
D:\codeblocks\... 22	22	warning: deprecated conversion from string constant to 'char*'
D:\codeblocks\... 22	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

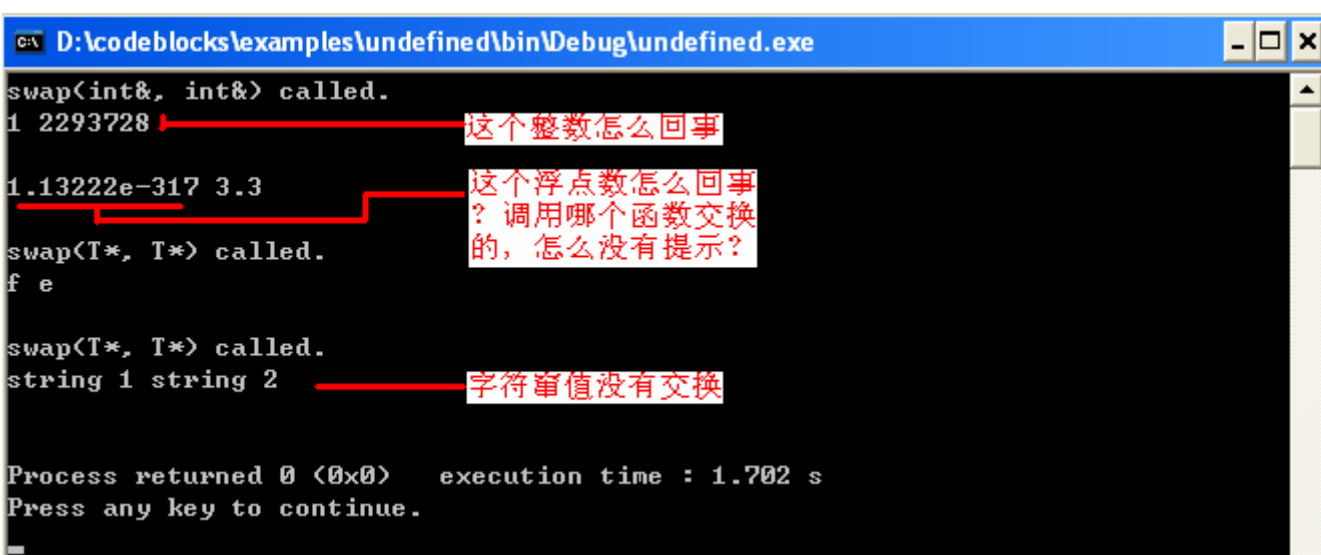
1.13222e-317 3.3

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。



```
C:\ D:\codeblocks\examples\undefined\bin\Debug\undefined.exe
swap(int&, int&) called.
1 2293728 这个整数怎么回事
1.13222e-317 3.3 这个浮点数怎么回事
swap(T*, T*) called. ？调用哪个函数交换的，怎么没有提示？
f e
swap(T*, T*) called.
string 1 string 2 字符串值没有交换

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

```

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
22     char* str1 = "string 1", * str2 = "string 2";
23     swap(str1, str2);
24     cout << str1 << ' ' << str2 << "\n\n";

```

局部变量a没有赋值

局部变量d没有赋值

调用哪个swap?

为什么不能交换两个字符串的值

先改正容易修改的错误，给a和d赋初值，假设a = 0, d = 0.1，保存当前文件。然后看看第15行，到底调用了哪个swap，因为我们的swap.hpp没有跟这个调用相匹配的函数，去能编译不出现语法错误，实在蹊跷，那我们就看看到底调用了哪个swap。选中swap(显示灰色)，按下鼠标右键，在弹出的快捷菜单中选择Find implementation of: 'swap'，见下图。

```

8  int main()
9  {
10     int a = 0, b = 1;
11     swap(a, b);
12     cout << a << ' ' << b << "\n\n";
13
14     double c = 3.3, d = 0.1;
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
22     char* str1 = "string 1", * str2 = "string 2";
23     swap(str1, str2);
24     cout << str1 << ' ' << str2 << "\n\n";

```

Toggle breakpoint

Run to cursor

Find declaration of: 'swap'

Find implementation of: 'swap'

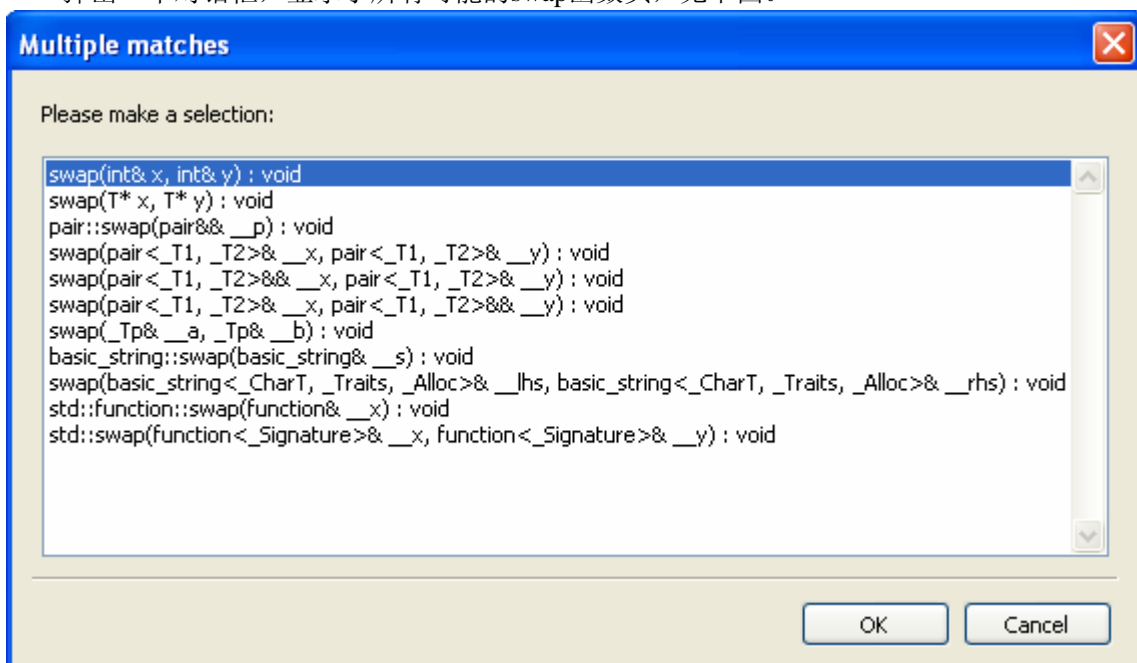
Switch to

Insert

Swap header/source

Edit

弹出一个对话框，显示了所有可能的swap函数头，见下图。



居然有这么多的swap，前两个是我们自己定义的，后面的是系统自带的。不要忘记前面的using namespace std;这会污染整个std命名空间，这样使用是OOP的大忌！注释掉using namespace std;，加一行using std::cout;然后重新编译，看看结果如何。

```
1 // main.cpp
2 #include <iostream>
3
4 //using namespace std;
5 using std::cout;
6 #include "swap.hpp"
7
8 int main()
9 {
10     int a = 0, b = 1;
11     swap(a, b);
12     cout << a << ' ' << b << "\n\n";
13
14     double c = 3.3, d = 0.1;
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";

```

Logs & others

Code::Blocks Search results Build log Build messages Debugger Thread search

File	Line	Message
D:\codeblocks\...		In function 'int main()':
D:\codeblocks\...	15	error: invalid initialization of reference of type 'int&' from
D:\codeblocks\...	16	error: in passing argument 1 of 'void swap(int&, int&)'
D:\codeblocks\...	22	warning: deprecated conversion from string constant to 'char*'
D:\codeblocks\...	22	warning: deprecated conversion from string constant to 'char*'
=== Build finished: 2 errors, 2 warnings ===		

最可能的匹配是void swap(int&, int&)，但是参数类型却不合适，因此编译器报错。知道原因就好，改正方法非常简单，改成swap(&c, &d);就可以了。

接下来，我们看看为何两个字符串的值不能交换吧。从输出信息可知，调用void swap(T\*, T\*);但是这个函数实现中有个判断是否相等的比较if(\*x != \*y)，而此刻的\*x和\*y都是字符's'(两个字符串的首个字符相等)，自然后面的三个语句根本就不执行，不要忘记判断字符串是否相等的C语言库函数是strcmp，而绝非直接判断是否相等。那怎么修改呢？最简单的办法是用C++的string处理这个问题，把字符串当成对象来处理。main函数前加一行#include <string>，然后用std::string定义字符串，即std::string str1("string 1"), str2("string2");，交换两个字符串最好的办法是用std::string的成员函数swap，这样不需要复制字符串内容，仅仅交换指针就可以了，也就是str1.swap(str2);。讲过上述这些修改后，保存，重新编译并运行得到的结果如下图。

```
C:\ D:\codeblocks\examples\undefined\bin\Debug\undefined.exe
swap<int&, int&> called.
1 0

swap<T*, T*> called.
0.1 3.3

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

string 2 string 1

Process returned 0 (0x0)   execution time : 2.163 s
Press any key to continue.

```