

## 2.1.2 Code::Blocks 集成开发环境的使用与调试方法简介

VS 对于初学者来说属于重量级的 IDE，相对于 VS 来说，Code::Blocks（有时也简写成 CodeBlocks）是一个“轻量级”的开放源码的跨平台 IDE，由纯粹的 C/C++ 语言基于著名的图形界面库 wxWidgets 开发。

Code::Blocks 支持 20 多种主流编译器，本书采用开源的 gcc/g++ 编译器和与之配对的 GDB 调试器。Code::Blocks 还支持插件，使其具有良好的可扩展性。

Code::Blocks 提供了控制台应用等许多工程模板，还支持语法彩色醒目显示、代码自动缩进和补全等功能，帮助用户方便快捷地编辑 C/C++ 源代码。

### 2.1.2.1 Code::Blocks 安装

目前，Code::Blocks 的最新版本是 17.12（即 2017 年 2 月份发布的版本），可从 Code::Blocks 官网 <http://www.codeblocks.org> 下载。该网站提供了 Windows、Linux（多种发行版）、及 Mac OS X 等系统下的安装文件或源文件。本书使用 Windows 版本的 Code::Blocks，可从以下网址下载：  
<https://jaist.dl.sourceforge.net/project/codeblocks/Binaries/17.12/Windows/codeblocks-17.12mingw-setup.exe>。注意，下载时请选择带有 MinGW 的版本，否则还需额外安装 Windows 下的 MinGW G++ 编译器才能使用编译执行功能。前面下载的安装程序已经自带完整的 MinGW 环境，因此无需额外安装 MinGW。

双击下载的文件，就可以开始安装 Code::Blocks 了，主要需要注意以下三点：

（1）选择默认的“Full/完整”安装（如图 2-27 所示），避免安装后的软件中缺少必需的插件

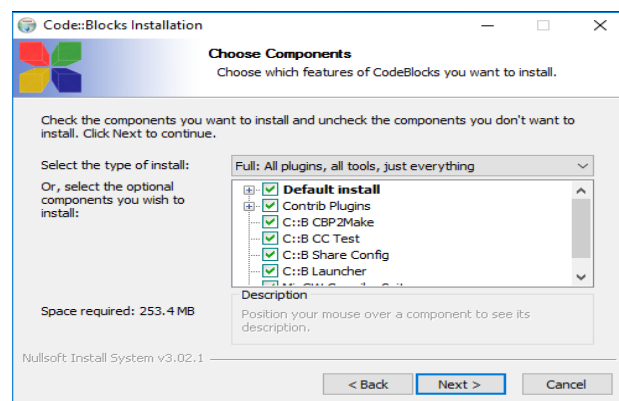


图 2-27 完整安装选项

（2）安装目录最好不要带有空格或汉字。

不要按照默认的带空格的路径 C:\Program Files(x86)CodeBlocks（如图 2-28 所示）安装 Code::Blocks，请点击“Browse.../浏览”选择 C 盘的根目录安装（如图 2-29 所示），当然也可以是其他目录，只要安装目录中没有空格或汉字即可，这是因为 MinGW 里的一些命令行工具，对中文目录或带空格的目录支持有问题。因此，安装在根目录（例如 C:\CodeBlocks）即可。

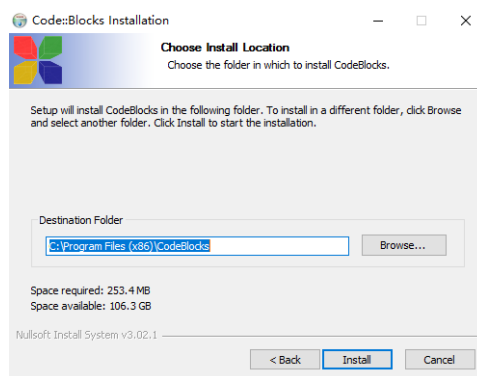


图 2-1 不能选择的安装路径

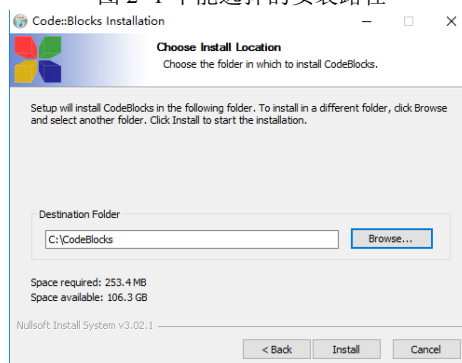


图 2-2 可以选择的安装路径

(3) 迈克菲等某些杀毒软件可能会与本软件发生冲突，因此建议安装之前卸载迈克菲杀毒软件。

安装结束后，双击桌面上的 Code::Blocks 启动图标，或运行在开始菜单里相应的程序启动 Code::Blocks。启动时，能看到如图 2-30 所示的启动界面，就说明安装成功了。



图 2-3 Code::Blocks 启动界面

## 2.1.2.2 Code::Blocks 基本配置

任何一款优秀的集成开发环境，都容许用户根据自己的习惯进行个性化配置。本书只对一些基本的配置加以介绍。

### 1、配置 G++ 编译器及调试器

首先到 X:\CodeBlocks\MinGW\bin 下，检查有没有以下文件：

mingw32-gcc.exe: C 的编译器。

mingw32-g++.exe: C++ 的编译器。

ar.exe: 静态库的连接器。

gdb.exe: 调试器。

windres.exe: Windows 下资源文件编译器。

mingw32-make.exe: 制作程序。

在选择“Full/完整”安装情况下，通常配置不会有问题。若在使用过程中出现无法编译或调试等问题，可能是编译器或调试器配置的路径不正确造成的，此时可以进行重新配置。

首先，点击 Code::Blocks 主菜单 “Settings/设置”，然后选中 “Compiler.../编译器”，在出现的对话框中，选中 “Toolchain executables/工具链执行程序” 标签，然后对照图 2-31，检查包括 MinGW 安装路径在内的配置是否正确，若不正确，则重新配置。其次，点击 Code::Blocks 主菜单 “Settings/设置”，然后选中 “Debugger.../调试器”，在出现的对话框中，选中 “Default/缺省” 标签，然后对照图 2-32，检查包括 gdb.exe 的执行路径在内的配置是否正确，若不正确，则重新配置。

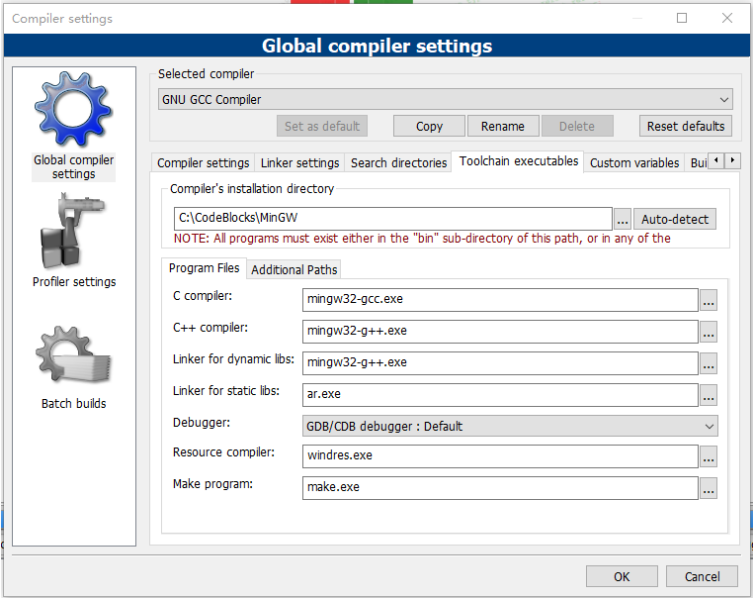


图 2-31 Code::Blocks 编译器配置对话框

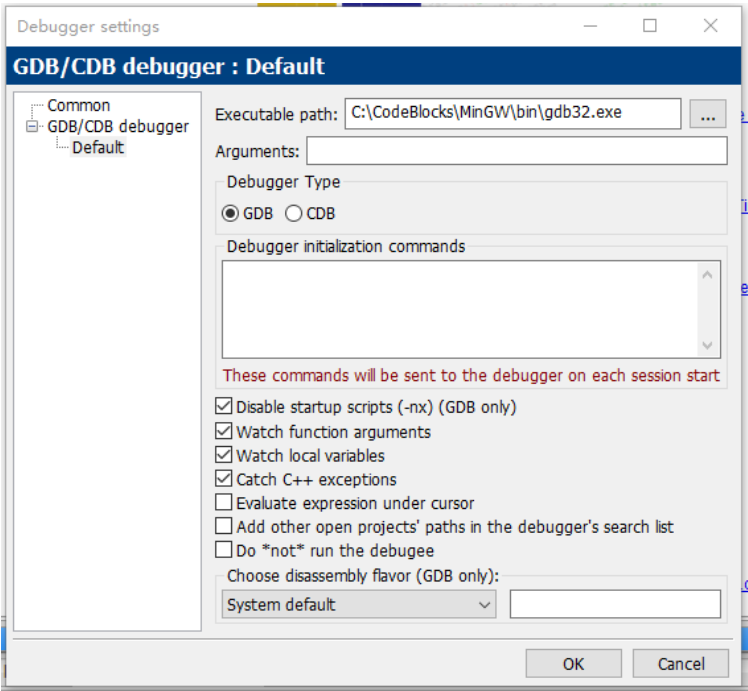


图 2-32 Code::Blocks 调试器配置对话框

## 2、配置编辑选项

若对 Code::Clocks 默认的字体和字号不满意，可以自行修改。具体操作是点击 Code::Blocks 主菜单 “Settings/设置”，然后选中 “Editor/编辑器” 选项，出现如图 2-33 所示的对话框后，点击 “Choose/选择” 选项即可根据用户个人的喜好选择合适的字体、字号等编辑选项。

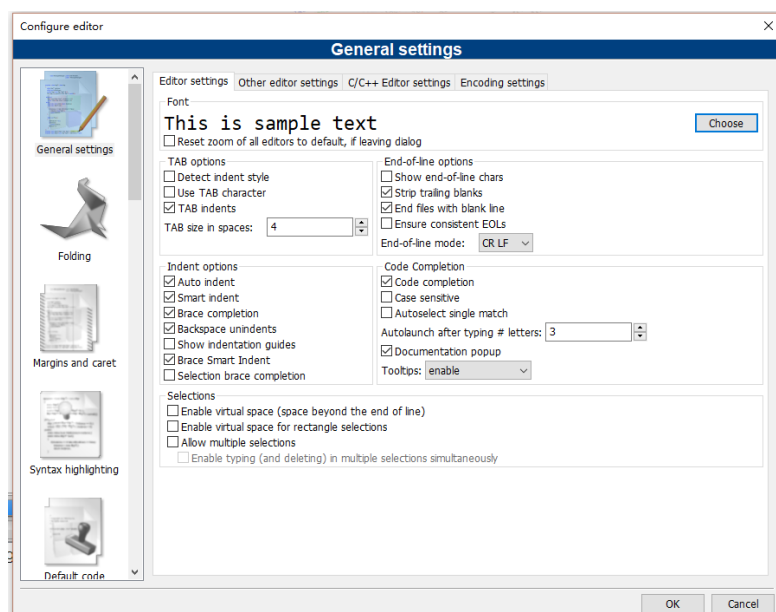


图 2-33 Code::Blocks 编辑配置选项对话框

### 2.1.2.3 创建控制台应用程序

Code::Blocks 支持创建多种类型的程序，本书仅介绍如何创建运行于控制台<sup>1</sup>的程序，即控制台应用程序，这是最基本的应用程序运行模式。

首先，点击主菜单“File/文件->New/新建->Project/项目”（如图 2-34 所示），或者更简单的，在“Start here”页面上，点击链接“Create a new project/创建一个新的项目”（如图 2-35 所示）。

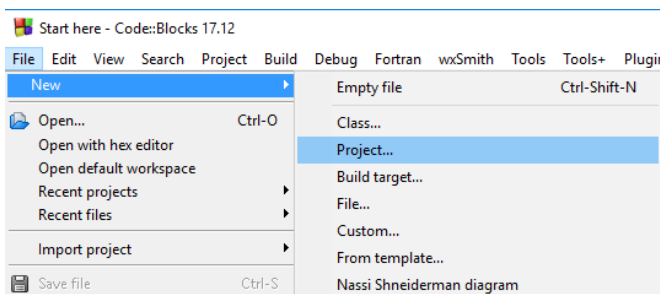


图 2-34 使用菜单功能创建新项目

<sup>1</sup> 控制台（Console），又称字符终端，是类似 DOS 的界面，只能显示字符信息。

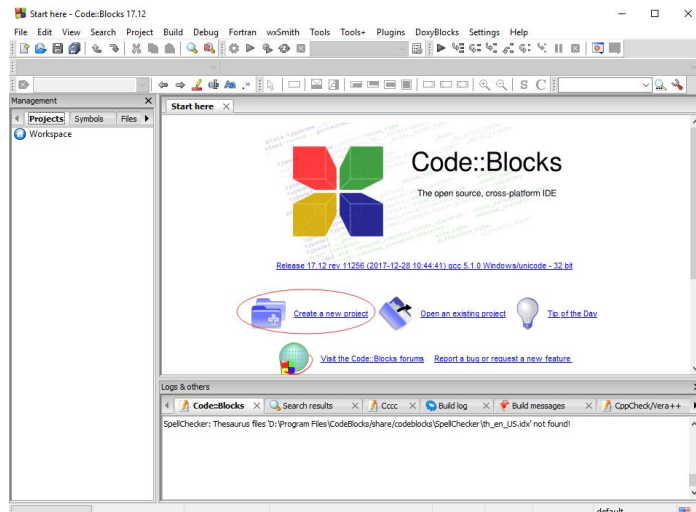


图 2-35 使用 Start here 界面创建新项目

然后出现如图 2-36 所示的新建项目对话框，选中“Console application/控制台应用程序”之后，点击“Go”按钮，就开始创建控制台应用程序的向导了。

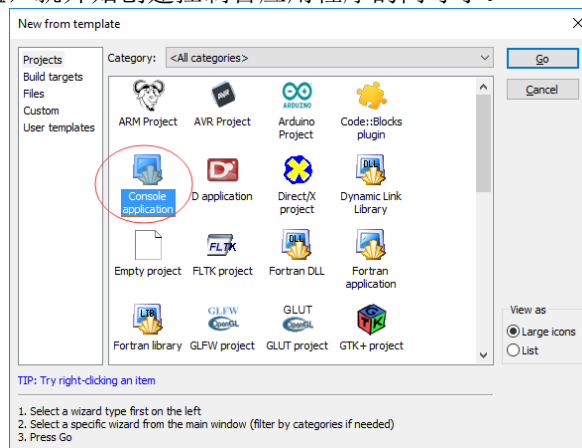


图 2-36 新项目类型选择对话框

向导第一步相当一个欢迎页面，如图 2-37 所示，点击“Next/下一步”按钮；

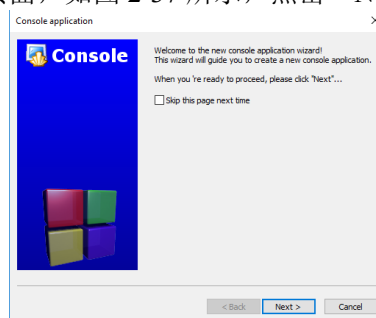


图 2-37 创建控制台应用程序的欢迎界面

第二步，如图 2-38 所示，选择“C”，创建 C 语言程序；

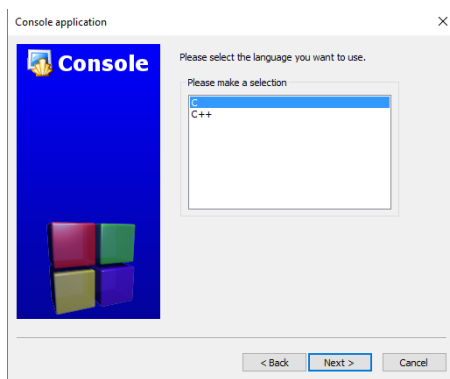


图 2-38 选择编程语言类型

第三步：输入项目名称（例如 **HelloWorld**），本项目将创建在以此命名的文件夹中。如图 2-39 所示，其它选项可以保持默认值，不过最好观察一下它们都在什么位置，特别是第二个选项“Folder to create project in/项目文件夹”指的是项目创建于哪个文件夹下面。这里默认将项目保存在“D:\CodeBlocks”下与项目名称同名的文件夹下。“**.cbp**”是 Code::Blocks 项目文件名的默认后缀。

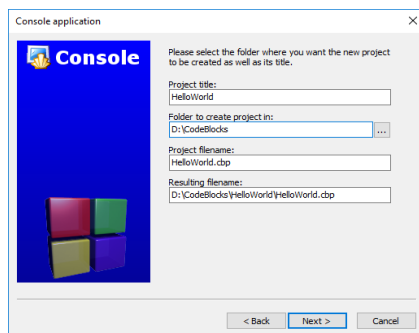


图 2-39 输入项目名称以及创建的位置

第四步：如图 2-40 所示，选择编译器为“GNU GCC Compiler（默认）”，其它也都保持默认值。点击“Finish/完成”按钮，结束向导。

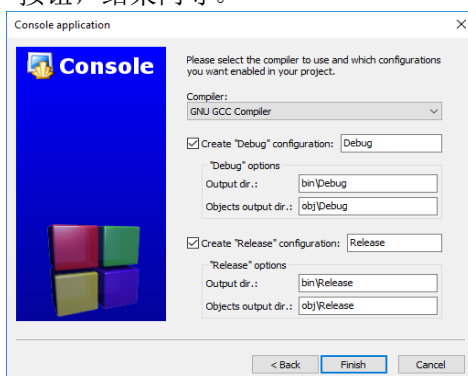


图 2-40 选择编译器类型

此时，在 Code::Blocks 左侧出现项目管理窗口中，在 HelloWorld 项目下的“Sources”中，可以看到在新创建的项目中自动添加了源代码文件 **main.c**，双击该文件开始编辑。可以发现，Code::Blocks 已经默认生成了一个最简单的输出“Hello World”的程序，如图 2-41 所示。

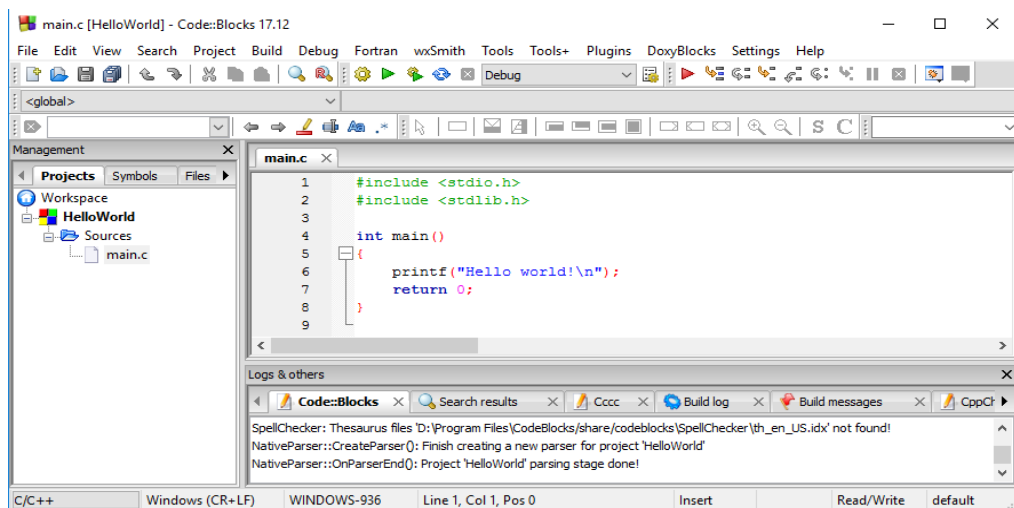





图 2-41 Code::Blocks 代码编辑界面

#### 2.1.2.4 编译和运行控制台应用程序

编译并运行程序的方法有如下几种：

- (1) 单击按钮栏的“编译”按钮  或项目名称的鼠标右键菜单中选择<Build>或<ReBuild>，然后单击“运行”按钮 。
- (2) 直接单击“编译运行”按钮 。
- (3) 在主菜单“Build/构建”中选择“Build and run/构建并运行”选项，如图 2-38 所示。
- (4) 使用快捷键 F9（调试运行）。

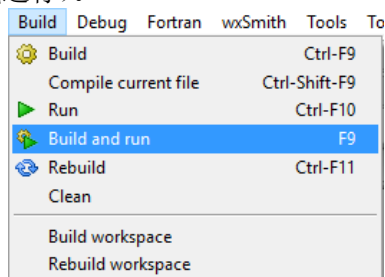


图 2-4 Code::Blocks 的主菜单项<Build>的内容

如果出现如图 2-39 所示运行结果，说明 Code::Blocks 配置正确，这样就可以开始激动人心的编程之旅了。

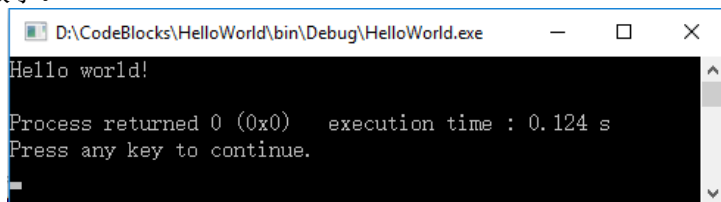


图 2-5 程序运行结果

在程序运行结束后，会将控制台窗口“冻结”，并且还输出了程序的用时，等待按用户按任意键后，窗口才关闭，这一点比 VS 方便。












#### 2.1.2.5 调试程序（Debug）

Code::Blocks 调试工具按钮如图 2-40 所示。



图 2- 6 Code::Blocks 调试工具按钮

从左至右各按钮的功能分别为：

-  (Debug/Continue)：开始和继续调试，即我们之前使用的开始调试功能，另外，若程序中断在某个断点处，点击该按钮后，程序会继续执行，直到遇到下一个断点或程序执行结束。
-  (Run to cursor)：执行程序并且在光标所在行中断，当你不想设置断点，却又想在某处中断时，可以将光标移动到想要中断的那行代码上，然后使用此功能。
-  (Next line)：下一行，执行一行代码，然后在下一行中断，即使本行含有函数调用，也不会进入函数执行，而是直接跳过去，这是最常用的功能。
-  (Step into)：步入，与下一行功能相对，此功能会将控制转入函数执行，如果你对函数里面的程序感兴趣，则可以使用此功能。
-  (Step out)：跳出，当你想跳出正在执行的函数时，可以使用此功能。
-  (Next instruction)：下一条指令，相对于下一行语句而言，其执行单位更小。
-  (Step into instruction)：步入下一条指令，与下一条指令相对，会跳入指令执行。
-  (Break debugger)：暂停调试。
-  (Stop debugger)：中止调试，如果找到错误了，或是不想继续调试了，就可使用此功能。
-  (Debugging windows)：与调试相关的观察窗口，如想查看变量的当前值、CPU 的寄存器状态，以及函数调用栈的调用情况等，可以开启相关的窗口。
-  (Various info)：信息窗口，开启一些比较琐碎的程序执行时的相关信息窗口。

下面，使用如下代码演示程序的基本调试方法。

```
#include <stdio.h>
#include <stdlib.h>
int add(int para1, int para2)
{
    int a, b;
    a = para1;
    b = para2;
    return a + b;
}
int main(void)
{
    int i;
    i = 1;
    i = 10;
    i = add(3, 4);
    printf("i = %d", i);
    return 0;
}
```

## 1、设置断点

断点 (Breakpoint) 设置是调试器的基本功能之一，可以让程序中断在需要的地方，从而方便其分析。如图 2-41 所示，在代码行号的右侧空白处单击鼠标左键，或在鼠标所处境行按 F5 快捷键，出现红色圆点后，即表示在该行成功设置了断点。单击红色圆点后，即可取消断点。



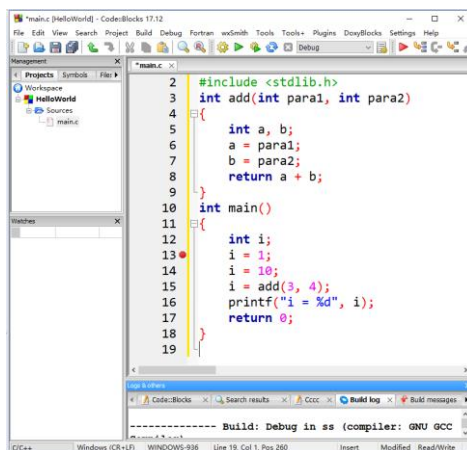


图 2-41 设置断点

## 2、开始调试

如图 2-42 所示，“Build target/构建目标”的选项必须是默认的“Debug/调试”，才能进行调试操作。若为“Release/发布”，则需要改为“Debug”，修改方法见图 2-34。

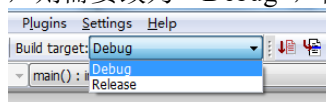


图 2-42 设置编译方式

然后单击“Debug/调试”主菜单下的“Start/Continue/开始/继续”选项（如图 2-43），或使用 F8 快捷键开始调试。

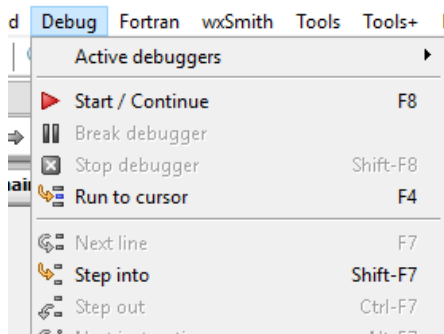


图 2-43 启动调试功能

如图 2-44 所示，此时程序会在遇到的第一个断点处中断，在红色断点圆点内出现一个黄色的小三角，表示它指向的代码行是下一步要执行的语句行。

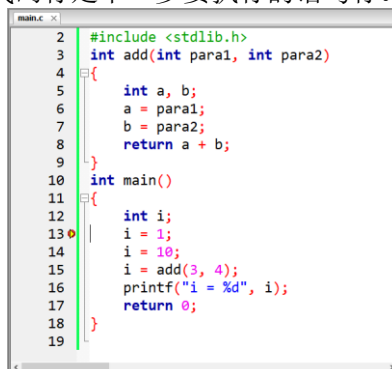


图 2-44 调试程序暂停

## 3、观察变量

程序在断点处中断后，我们更关心此时各个变量的值是否是我们预想的，因此需要观察各个变量的值。点击如图 2-45 中的工具条按钮后，选择“Watches/观察”选项。此时会出现

如图 2-46 所示的变量观察窗口，在其中显示了各个局部变量（这里是变量 i）当前的值。

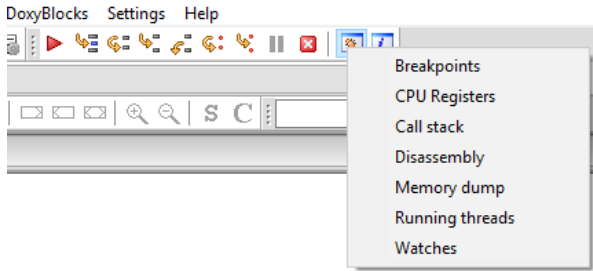


图 2-45 启动变量观察窗口

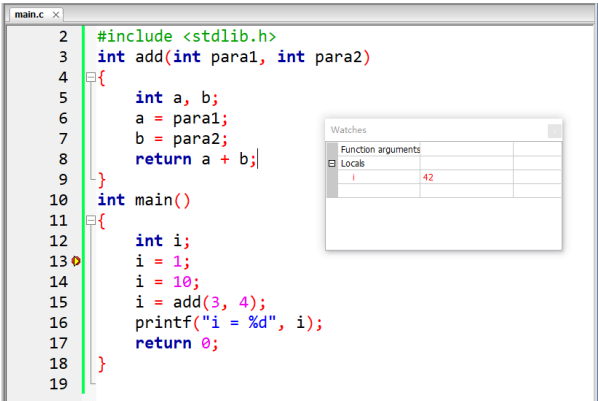



图 2-46 变量观察窗口

我们观察到 i 的当前值为 42，但其实这是一个垃圾数，也称随机数，因为此时第 13 行的赋值操作尚未执行。换个编译器或计算机就有可能得到不同的值。按 F7 键或  (“Next line/下一行”按钮)，开始单步执行。黄色箭头立即指向下一条语句“i=10;”处，而“i=1;”已经执行完毕了，通过监视窗可以看到 i 的值确实变成了 1。继续按 F7 键，黄色箭头随之逐条下移，i 的值也随之而改变。光标移到“i=add(3,4);”这一行。此时再按 F7 键，i 的值变为 7，说明 add()函数返回了 7，再按 F7 键，执行第 16 行的输出语句，向屏幕输出变量 i 的值后，黄色箭头停到“return 0;”处。

注意，按 F7 键是不进入函数内部单步跟踪的。为了对函数内部语句的执行情况进行跟踪，当黄色箭头停在调用它的语句时，改按快捷键 Shift+F7（或“Step into/步入”按钮），黄色箭头暂停在函数 add()内的第一条可执行语句处（第 6 行），如图 2-47 所示：

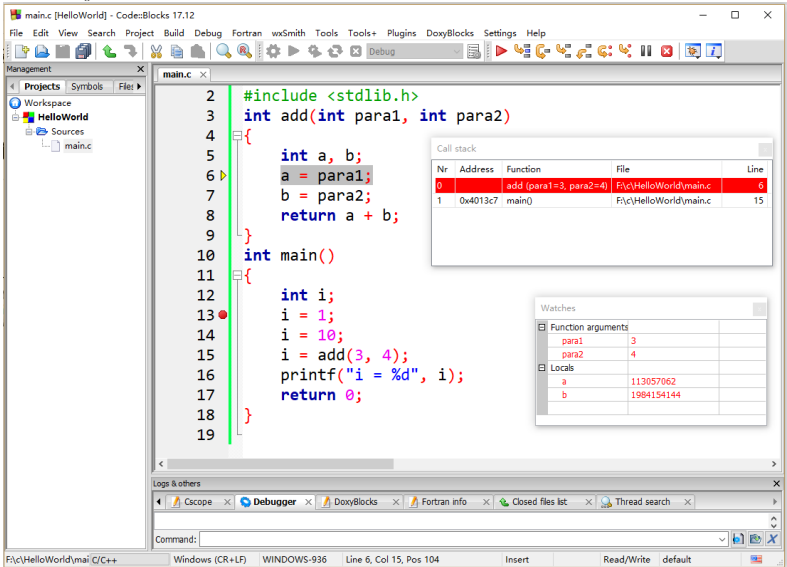




图 2-47 程序进入函数执行

图 2-47 中的“Call stack/函数调用栈”窗口是通过点击工具栏中的 Watches 按钮，并在其下拉菜单中选择“Call stack”选项打开的，从中能看出是 main()调用了 add()，两个参数分别是 3 和 4。在里面的任意一行单击右键，在弹出的菜单中选择“Switch to this frame”，可把运行环境切换到函数的该次调用，进而查看该次调用时各个变量和参数的值。

现在，可以按 F7 键、Shift+F7 等快捷键在函数 add()中慢慢调试了。监视窗中已经显示了 para1 和 para2 两个参数，看看它们的值，直观地体会一下函数参数是如何对应传递的。然后按 F7 键或快捷键 Shift+F7，一步步观察函数里面都做了什么，直到函数返回。

add()函数返回后，停在主函数的“return 0;”处。再向下执行，程序将正常退出。

如果在程序调试过程中，想直接观察变量在内存中的存储方式，可以使用“Memory dump/内存镜像”这个功能。当程序暂停的时候，点击工具栏中的 Watches 按钮，并在其下拉菜单中选择“Memory dump/内存镜像”选项，弹出 Memory 窗口，在 Address 的后面输入“&b”，即变量 b 的地址，此时也可以输入数组名（表示该数组的首地址）。然后按 Enter 键，则变量 b 的内存信息便以字节形式展现出来了，如图 2-48 所示。

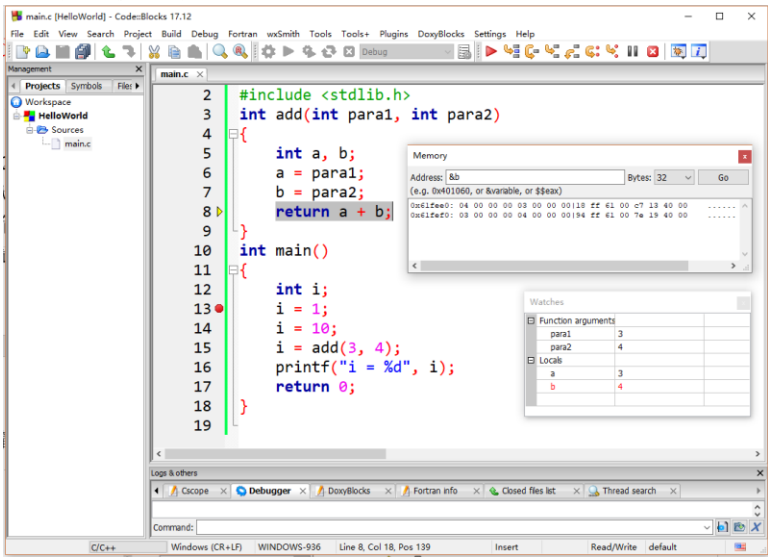


图 2-48 内存镜像窗口

可见，此时变量 b 的值是 4，由于 b 是整数，在内存中占 4 个字节，由低位到高位分别是 04 00 00 00。

## 5、命令行方式运行

当将 main 函数声明为有参数的 main 函数形式（`int main(int argc, char *argv[])`）时，程序在运行时需要通过命令行方式将参数传递给程序，如何在 Code::Blocks 中运行和调试带有命令行参数的程序呢？首先需要点击“Project/项目→Set program’s arguments/设置程序参数”菜单，打开命令行参数设置对话框，在 program arguments 文本框内设置项目运行所需的命令行参数（例如输入一个参数为 test，如图 2-49 所示）。

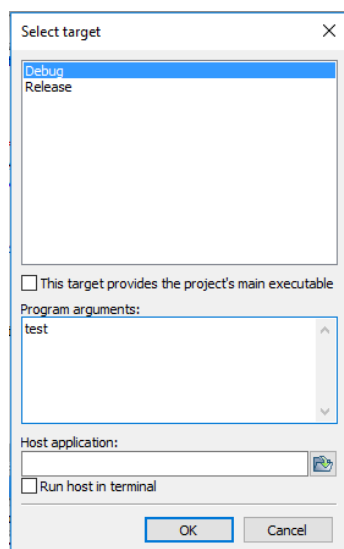


图 2-49 设置命令行参数

为了方便查看参数的值，在第 13 行下断点，并按 F8 开始单步执行程序，程序停留在主函数中的断点处。此时启动变量观察窗口，并添加两个 watch 值 `argv[0]` 和 `argv[1]`，如图 2-50 所示，其中 `argc` 的值为 2，即有两个命令行参数，其中第一个参数 `argv[0]` 为可执行程序的完整目录，而第二个参数 `argv[1]` 即用户输入的 `test` 参数。

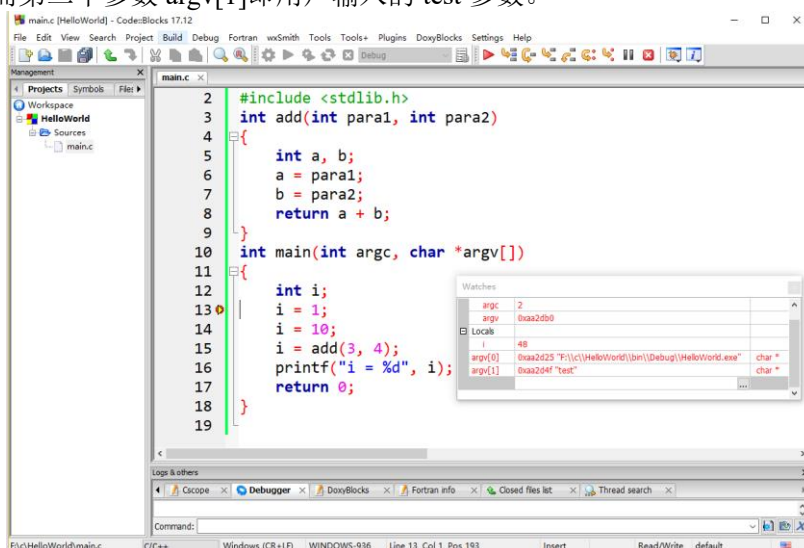


图 2-50 修改 main 函数和下断点，并在变量观察窗口中添加查看参数值

## 2.1.2.6 Code::Blocks 下的多文件项目开发

首先按 2.1.2.3 节的方法，创建控制台项目 Test3，如果不用 Code::Blocks 在新建项目中自动添加的源代码文件 `main.c`，则在 Code::Blocks 的管理器（Management）中，打开项目 Test3 的源文件夹 Sources，在文件名 `main.c` 上单击鼠标右键，选择“Remove file from project/从项目中移除文件”即可手动删除该文件，如图 2-51 所示。也可以在 Code::Blocks 的管理器（中单击文件名 `main.c` 后，直接按“Delete/删除”键删除该文件。这样得到的是一个不包含任何代码文件的空项目，接下来就可以将已有文件添加到项目中，或者为项目添加新文件了。

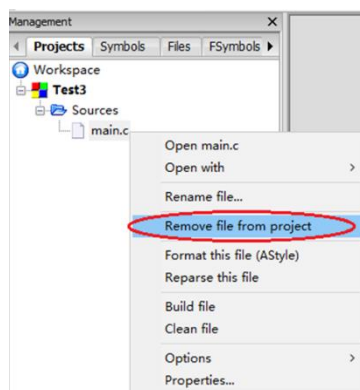


图 2- 51 将文件从项目中删除

将已有文件添加到项目中的方法如下：

第一步：将已有的代码文件拷贝到项目 Test3 所在的文件夹中。

第二步：在 Code::Blocks 中，鼠标右键单击项目 Test3，选择“Add Files.../添加文件”，如图 2-52 所示。

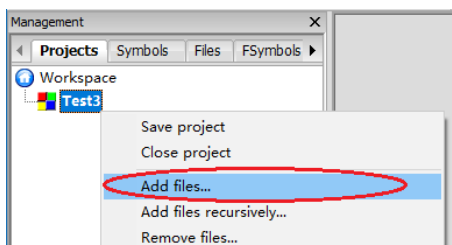


图 2- 52 Code::Blocks 下项目 Test3 单击鼠标右键后的弹出菜单

第三步：选择要添加的代码文件，如图 2-53 所示。

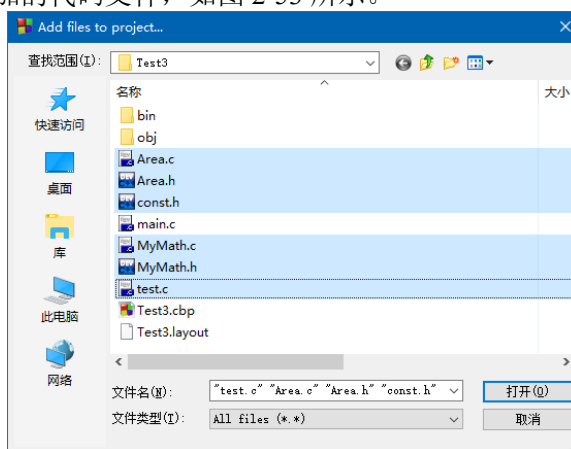


图 2- 53 Code::Blocks 下添加文件界面

第四步：如图 2-54 所示的界面中（选中两个复选框），单击“OK”按钮完成添加。完成文件添加后，在 Code::Blocks 的管理器中，打开项目 Test3 的目录树，能看到刚刚添加的代码文件，如图 2-55 所示。

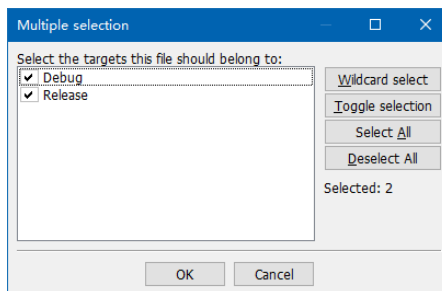


图 2- 54 Code::Blocks 下添加文件后的目标多选窗口

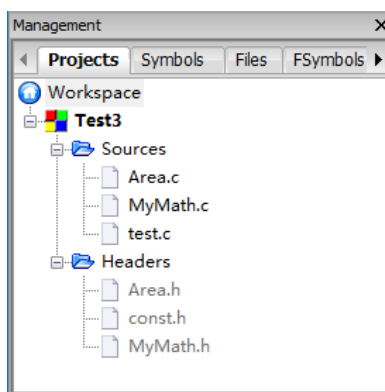



图 2-55 完成文件添加后，Test3 的文件目录树

向项目 Test3 中添加一个全新的代码文件(.h 文件或.c 文件)的方法如下：

第一步：点击 Code::Blocks 按钮栏中的添加新文件按钮，在弹出菜单中选择 “File.../文件”，如图 2-56 所示。

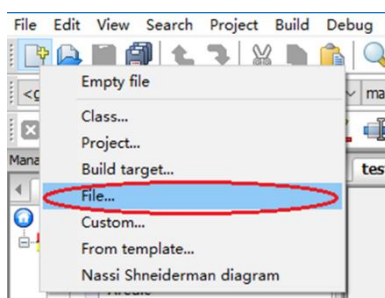


图 2-56 添加文件按钮的鼠标右键菜单

第二步：在弹出的窗体中选择要添加的新文件类型（如“C/C++ header”、“C/C++ source”、“Empty file”等），然后单击按钮 “Go”，如图 2-57 所示。

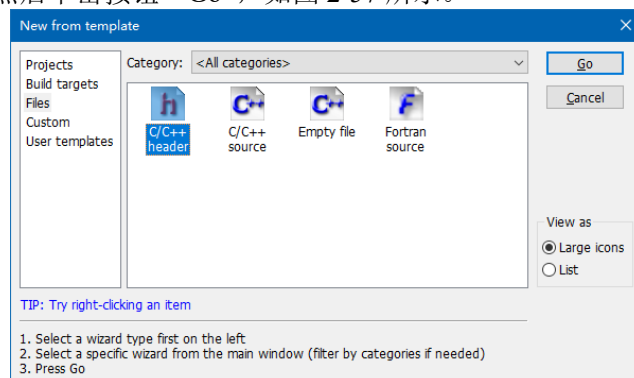


图 2-57 Code::Blocks 中添加新文件时的文件类型选择界面

第三步：在弹出的语言选择窗体中选择 “C”，单击按钮 “Next”，如图 2-58 所示。

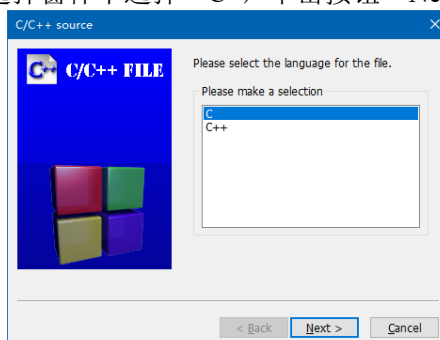


图 2-58 Code::Blocks 中添加新文件时的语言类型选择界面

第四步：在如图 2-59 所示的窗体中的文件路径编辑框输入完整的路径，例如：D:\Programing\_C\Test3\NewFunc.c，并勾选 Add file to active project 选项，单击按钮“Finish”。

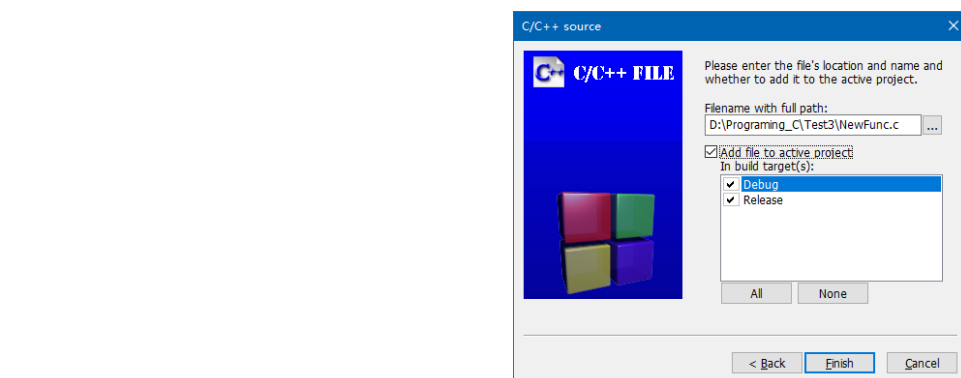


图 2- 59 Code::Blocks 中添加新文件时的语言类型选择界面

第五步：完成文件添加操作后，即可进入文件编辑状态编辑该文件了。

## 2.1.2.7 Code::Blocks 安装和使用中的常见问题

### 1、如果编译报错，怎么办？

如果示例程序不能正常运行，一种可能是你安装了不带编译器和调试器的版本，重新下载带 gcc 编译器和 gdb 调试器的 code::blocks（下载软件名中务必包含 mingw）并安装即可。

另一种可能是编译器配置有问题，如果你曾多次卸载 Code::Blocks 并将其安装到不同的目录下，那么有可能发生配置错误这个问题。此时，可按如下步骤检查编译器设置是否正确。

第一步：打开 Code::Blocks，如图 2-54 所示，点击下拉菜单 Settings，选择第三个菜单项 Compiler。

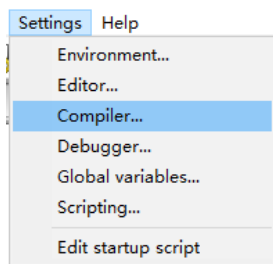


图 2-54 查看编译器设置

第二步：如图 2-55 所示，选择左侧的 Global compiler settings，在右侧的 Selected compiler 中选择 GNU GCC Compiler，并且选择 Toolchain executables 选项，查看编译器的根目录是否是实际安装的根目录。如果不是，则找到 Code::Blocks 安装目录下的自带编译器目录，将找到的编译器根目录复制进去，或者点击其右侧的 [...] 选择编译器安装的目录。

因为前面提到 Code::Blocks 是安装到了 C 盘的根目录，所以编译器的目录应为 C:\CodeBlocks\MinGW。如果读者安装的目录不是 C 盘根目录，那么这里需要做相应的修改，尤其是曾经卸载过一次 Code::Blocks 后更要检查下这里编译器的路径是否修改为了重新安装 Code::Blocks 的路径。

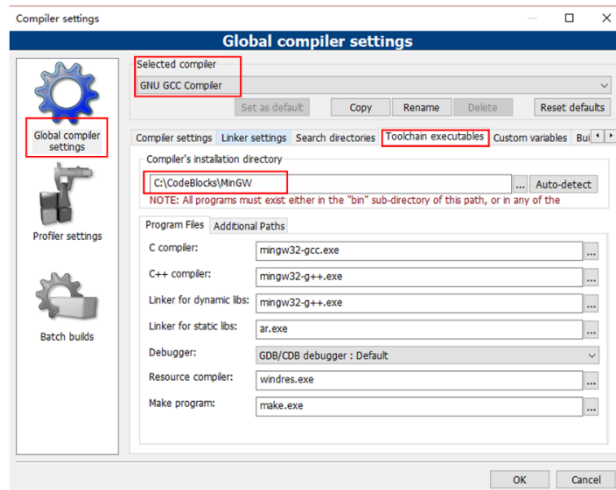


图 2-55 查看编译器的根目录是否正确

第三步：重新打开 Code::Blocks，然后编译，如果编译器没有报错，则说明配置成功了。

## 2、如果不能调试程序，怎么办？

如果程序不能正常调试，一种可能是你编写的程序所保存的目录名中有中文或空格，另一种可能是调试器配置有问题，这种问题常常发生在你多次卸载和安装 Code::Blocks 之后。此时，可按如下步骤检查调试器设置是否正确。

第一步：打开 Code::Blocks，如图 2-56 所示，点击下拉菜单 Settings，选择第四个菜单项 Debugger。

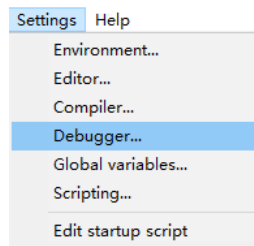


图 2-56 查看调试器设置

第二步：如图 2-57 所示，选择左侧的 Default，在右上方的 Executable path 中查看调试器的根目录是否是实际安装的根目录。如果不是，则找到 Code::Blocks 安装目录下的自带调试器目录，将找到的调试器根目录复制进去，或者点击其右侧的... 选择调试器安装的目录。



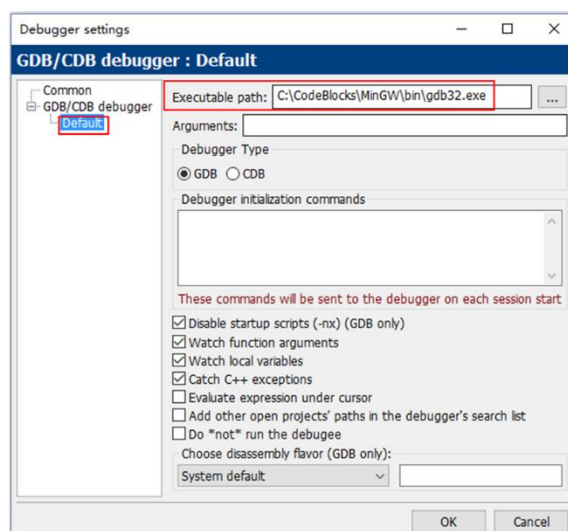


图 2-57 查看调试器的根目录是否正确

### 3、如何设置使用 C99 标准编译？

打开 Code::Blocks，如图 2-58 所示，点击菜单栏 Settings 选项，点击第三个菜单项 Compiler。选择左侧的 Global compiler settings，在右侧的 Compiler Flags 中选择 c99 即可。

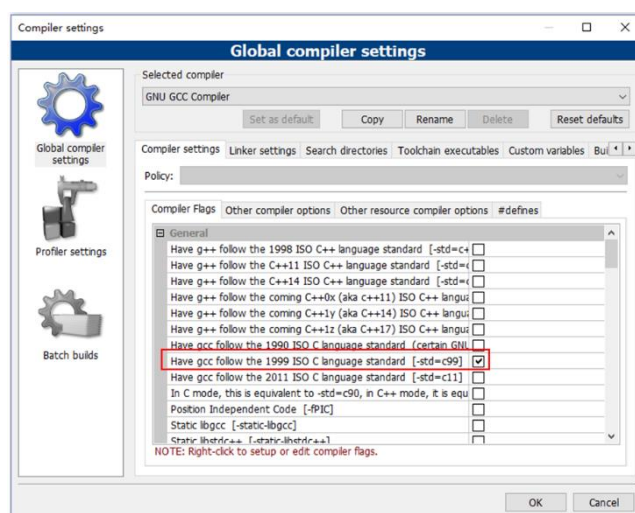


图 2-58 设置使用 C99 标准编译

### 4、在 windows 系统中程序输出中文时出现乱码，怎么办？

在 windows 系统中程序输出中文时，如果出现乱码，则很可能是编码方式不一致导致的，例如发生了 UTF-8 和 GBK 冲突的问题。如果一个文件本来是以 UTF-8 编码方式保存的，但是以 gbk 打开，当然就会出现乱码。有两种解决方法：

第一种解决方法：用 UTF-8 打开文件。UTF-8 是 Linux 系统中常用的中文编码方式，minGW 是 gcc 的编译器，默认是 UTF-8 编码方式，但是打开下拉菜单 Setting，选择第二个菜单项 Editor 后，点击 Encoding Setting 中，如图 2-59 所示，可以看到默认的编码方式是 windows-936（其实就是 GBK）。此时可以把文件打开的编码方式修改为 UTF-8，如图 2-60 所示。修改完设置后必须重新保存文件才有效，这意味着你以后保存的文件都是 UTF-8 编码，因此相比于第一种解决方法，我们更推荐使用第二种解决方法。

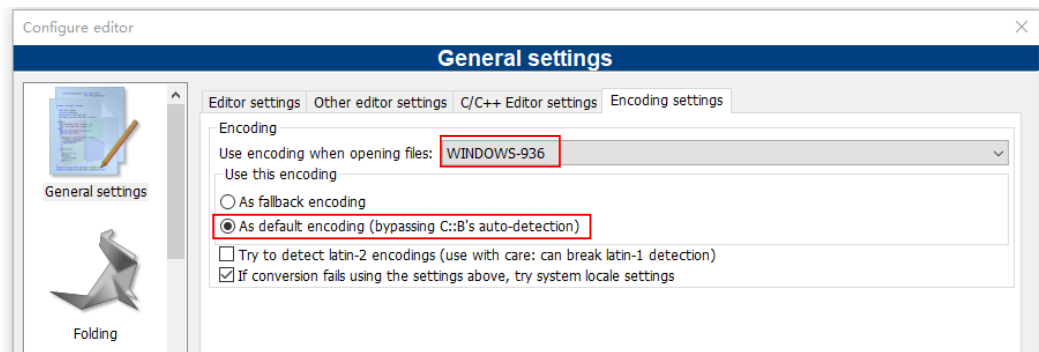


图 2-59 默认的编码方式是 windows-936

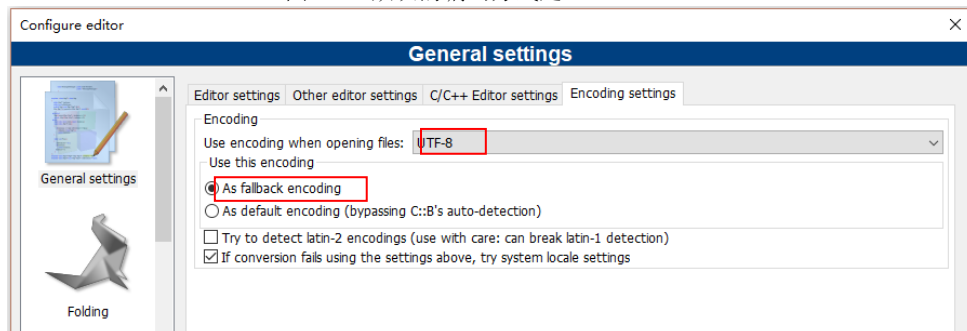


图 2-60 把文件打开的编码方式修改为 utf-8

第二种解决方法：仍使用 windows-936 编码方式打开和保存文件，但是让编译器使用 GBK 编码编译程序，即图 2-59 中的设置保持不变，仍勾选作为默认的编码格式，但是打开下拉菜单 Setting，选择第三个菜单项 Compiler，点击 Other compiler options，在其下面的文本框中键入下面两行内容，然后点击 OK，重新保存文件，就可以让编译器使用 GBK 编码编译程序了。

-finput-charset=GBK

-fexec-charset=GBK

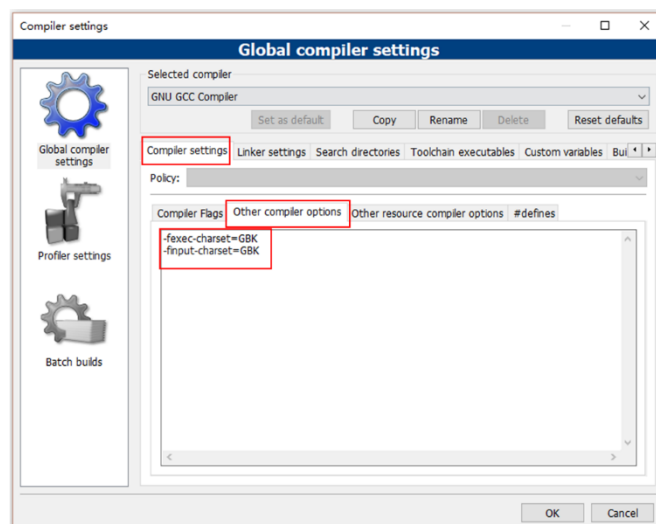


图 2-61 设置让编译器使用 GBK 编码编译程序