## debug工具概述

本文主要介绍以下几个同学们常用的IDE的debug工具使用方式

* Codeblocks 16.01
* Dev-C++ 5.11
* Visual studio 2017

这些IDE中的debug工具大同小异，主要常用的功能有下面这些：

1. Breakpoint/Line breakpoint 在行上设置断点，在此行执行之前暂停
2. Debug/Continue 开始debug，或者从当前停下的地方继续执行，执行到下一个断点处
3. Next line 执行到下一行之前停止
4. Step into 进入本行中的函数体中执行
5. Step out 结束当前函数执行，返回到调用函数的位置
6. Call stack 查看当前函数调用栈
7. Watch variable 查看变量的值
8. Variable breakpoint 为变量设置断点，可以在此变量被读/写时暂停程序执行

除此之外debug工具还有很多其他功能，但不常用，就不赘述了，一般的debug需求上面这些都可以搞定了，有更多需求的请查看IDE或者debug工具(gdb)的文档。

1. Codeblocks 16.01
   1. Debug准备工作

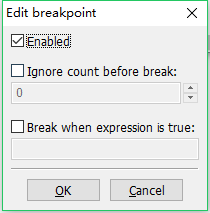
在项目编译之前，将上面的编译模式调为Debug



然后，很! 重! 要! 的! 一! 点! 项目路径不能有中文，这是我卡了2个小时后得出的结论。

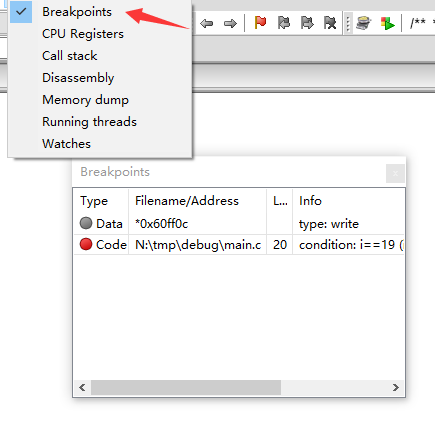
* 1. 设置断点。右键toggle breakpoint或者F5键在当前行设置断点，设置了断点的行前面会有小红点。断点在debug过程中也可以添加和删除。

断点可以设置条件，右键断点的小红点Edit breakpoint 可以编辑触发条件。

Ignore count… 是在触发之前跳过多少次，比如你在循环里有个断点，你想在前100次循环不触发，那在这里打对号填100即可。

Break when… 是满足表达式时触发，比如可以填i==100，跟上面这个效果差不多。你也可以写个复杂的表达式，只要能放在if里面的表达式都可以放在这里。

断点可以在断点窗口中查看/激活/禁用/删除

红色的是已激活的断点，灰色是禁用的断点，禁用的断点不会被触发。

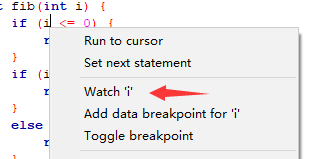
* 1. 开始debug。

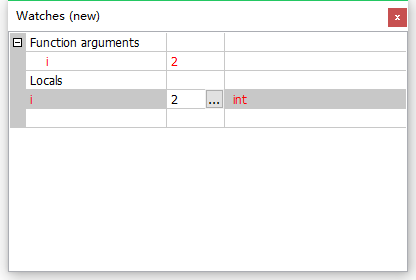
红色三角开始debug，这会让程序执行到第一个断点处暂停。如下



程序暂停后会有黄色三角指示程序的暂停位置

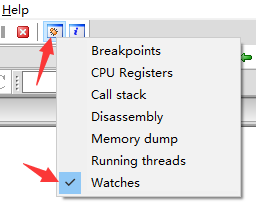
或者使用，可以让程序执行到现在光标所在的位置暂停。

* 1. 添加变量watch。在程序暂停的过程中时，可以右键代码中的变量，选Watch ‘变量名’，就可以把变量添加到watch列表来方便查看变量的值，添加之后会弹出Watches窗口如下



里面是所有已添加watch的变量的名字以及值。

如果不小心关掉了，你可以在这里打开



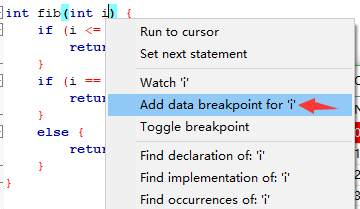
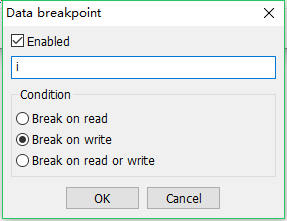
添加到watch之后，就可以方便的看变量的值了。

* 1. Next line, step into, step out。这几个都在Debug按钮的后面，Next line 和 Step into都是执行到下一行，区别在于Step into如果碰到了函数调用，会进入到那个函数的第一行然后暂停，而Next line不会进入函数。在当前行没有函数的情况下，两者行为相同。Step out用于在函数当中，如果你不想看当前函数值行了，点击这个会直接回到调用这个函数的位置。
  2. 红叉按钮可以终止程序执行。继续点击红三角是继续程序执行直到下一个断点。
  3. 设置变量断点。右键变量可以为变量设置断点，点击后弹出右边的窗口，Enabled是是否启用断点，第二行是变量名，应该已经自动填好了，下面的是断点触发条件，

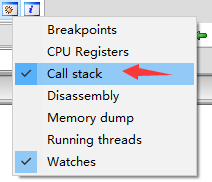
Break on read是在读取此变量的时候即暂停程序，比如为i设置断点，那么在对i进行判断、把i作为参数传递、将i赋给其他变量等操作时都会触发断点。

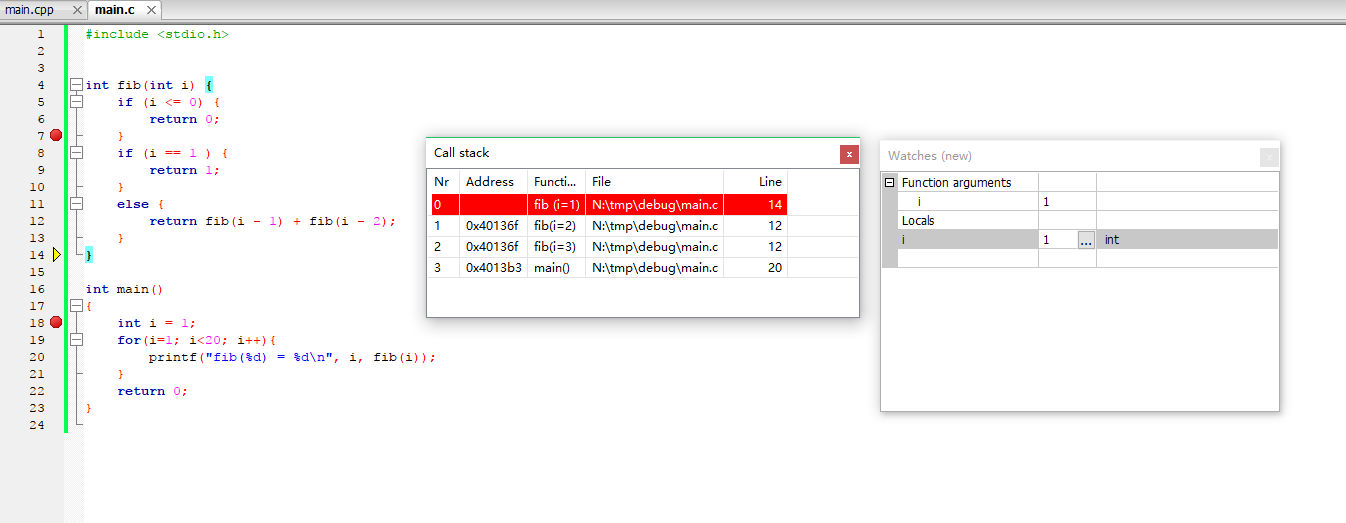
Break on write 是在修改此变量时暂停程序，即对i赋值时。

Break on read or write 顾名思义读和写都会触发断点。

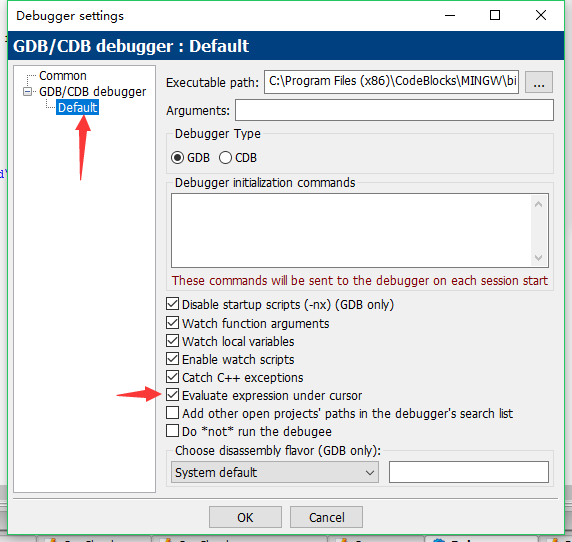
* 1. 查看函数调用栈。



打开call stack 呼出调用栈窗口。这个在你函数嵌套比较复杂或者使用递归的情况下比较有用，可以看到函数调用的层级，比如下面一个递归的斐波那契程序，

这是当外面调用i=3后，某一时刻的调用栈，可以看到在main函数中，当前执行到20行，然后调用了fib(i=3)，在fib(i=3)中调用了fib(i=2)，然后fib(i=2)中又调用了fib(i=1)。你可以通过双击其中的某一个来切换到那个函数的上下文，查看那个函数中的局部变量的值。

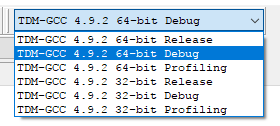
* 1. 设置显示指针下的变量的值。在Settings -> Debugger中进行如下配置，



然后在debug过程中程序暂停后可以鼠标指向某个变量，等待一下就会显示变量的当前值。

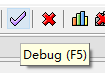
1. Dev-C++
   1. Debug准备工作

在编译之前，将右上的编译模式调为Debug，32bit或64bit都行



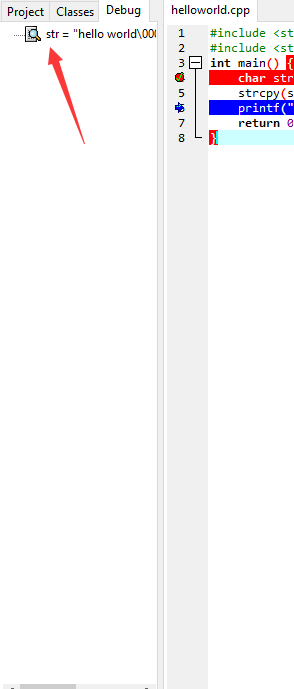
* 1. 设置断点。右键toggle breakpoint或者F4键在当前行设置断点，设置了断点的行前面会有小红点。断点在debug过程中也可以添加和删除。
  2. 开始debug。

对号开始debug，这会让程序执行到第一个断点处暂停。如下



程序暂停后会有蓝色三角和蓝色背景指示程序的暂停位置

* 1. 添加变量watch。在程序暂停的过程中时，可以右键代码中的变量，选Add Watch，就可以把变量添加到watch列表来方便查看变量的值，Watch 窗口默认在代码窗口左边



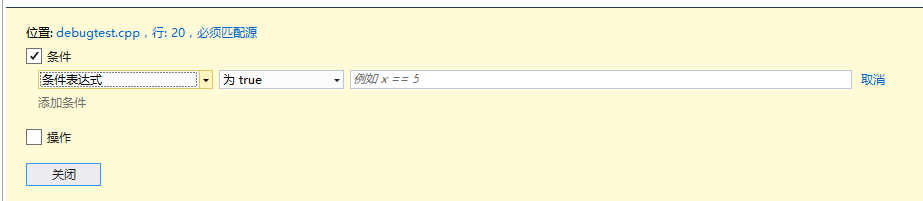
里面是所有已添加watch的变量的名字以及值。

或者在暂停时，鼠标指到变量上等一下，就会显示这个变量的当前值。



* 1. Next line, into function, skip function。这几个都在下方的Debug窗口中，Next line 和 into function都是执行到下一行，区别在于into function如果碰到了函数调用，会进入到那个函数的第一行然后暂停，而Next line不会进入函数。在当前行没有函数的情况下，两者行为相同。skip function用于在函数当中，如果你不想看当前函数中的行了，点击这个会直接回到调用这个函数的位置。
  2. 设置变量断点。貌似不支持这个功能。
  3. 查看函数调用栈。好像也没有。

1. Visual Studio
   1. 编译模式设为Debug ，不然一些变量的值可能会看不到。
   2. 设置断点。右键toggle breakpoint或者F5键在当前行设置断点，设置了断点的行前面会有小红点。断点在debug过程中也可以添加和删除。Vs中断点可以设置条件，在断点红点出悬停点击设置按钮可以设置触发条件，条件表达式只要是有效的c语言表达式即可。



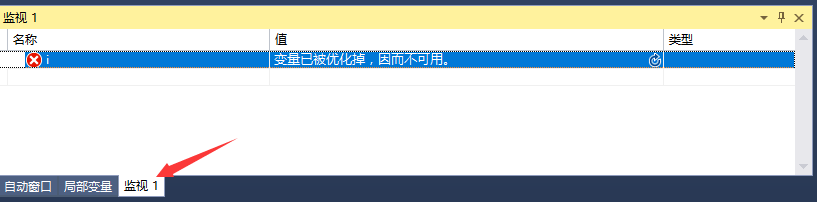
* 1. 开始debug。

调试->开始调试，或F5，或在某一行右键运行到光标处。

程序暂停后会有黄色箭头指示程序的暂停位置

暂停过程中，鼠标指向变量可以显示变量当前值。

* 1. 添加变量watch（监视）。在程序暂停的过程中时，可以右键代码中的变量，选添加监视，就可以把变量添加到监视列表来方便查看变量的值，监视窗口默认在这里（下面就是用release模式编译的效果。。变量被优化掉了没法查看）



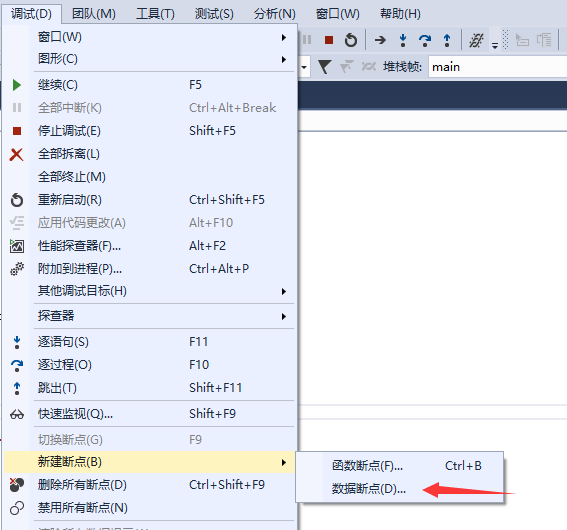
里面是所有已添加watch的变量的名字以及值。

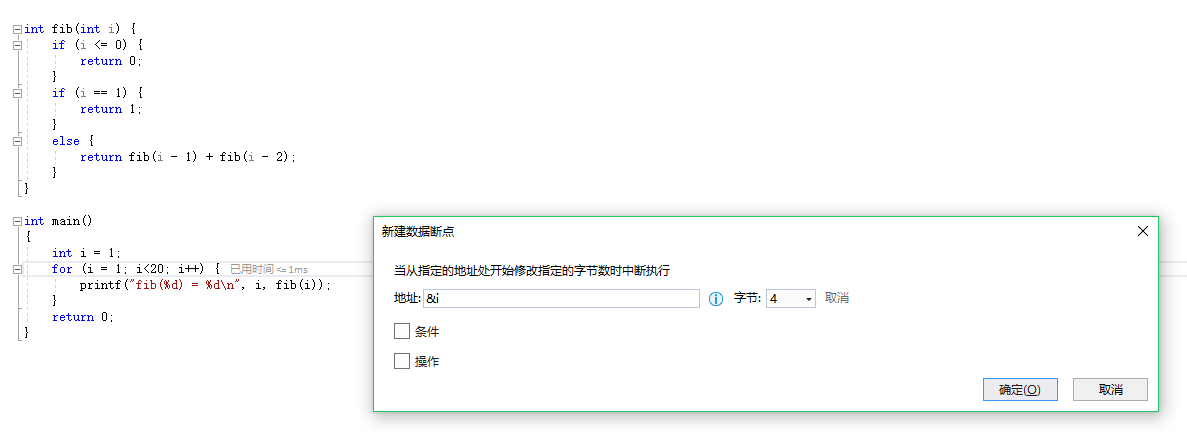
如果找不到，请到 调试 -> 窗口 里面去找。注意在没有运行的情况下是没有这个窗口的。

* 1. Next line（逐过程）, step into（逐语句）, step out（跳出）。

Next line 和 Step into都是执行到下一行，区别在于Step into如果碰到了函数调用，会进入到那个函数的第一行然后暂停，而Next line不会进入函数。在当前行没有函数的情况下，两者行为相同。Step out用于在函数当中，如果你不想看当前函数值行了，点击这个会直接回到调用这个函数的位置。

* 1. 红色方块按钮可以停止程序执行。会继续执行直到遇到下一个断点。
  2. 设置变量断点。调试->新建断点->数据断点。弹出的窗口中填写要监视变量的地址。

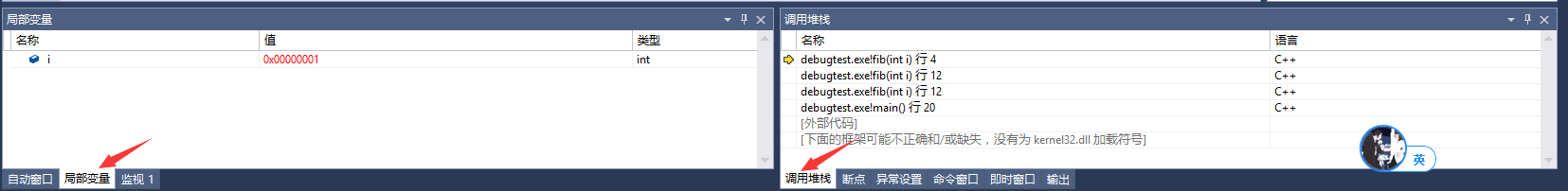




这里我要监视变量i，就要填i的地址，也就是&i。这里需要注意要填对变量的字节数，如果是char就是1字节，short是2字节，int和float是4字节，double是8字节。

这里的变量都是在变量修改时触发的。选中条件可以编辑触发条件。

* 1. 查看函数调用栈。



在“调用堆栈”中可以查看当前函数调用情况，这个在你函数嵌套比较复杂或者使用递归的情况下比较有用，可以看到函数调用的层级，比如上上图中的递归的斐波那契程序，上图显示的是当外面调用i=3后，某一时刻的调用栈，可以看到在main函数中，当前执行到20行，然后调用了fib(i=3)，在fib(i=3)中调用了fib(i=2)，然后fib(i=2)中又调用了fib(i=1)。你可以通过双击其中的某一个来切换到那个函数的上下文，查看那个函数中的局部变量的值。

1. 总结及闲扯

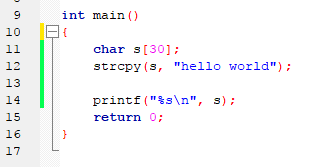
其实debug不一定要用这些debug工具，其实用好printf就完全可以搞定了，我后面写了一些基本的debug思路。但是在某些情况下熟悉debug工具确实能更有效的debug，毕竟不需要改代码，而且交互式的方便使用。

前面两个IDE的调试器都是在gdb上加了个ui界面（gdb是个命令行中使用的调试工具），而且只提供了部分的gdb功能，像devc++中设置条件这些东西我也没找到。

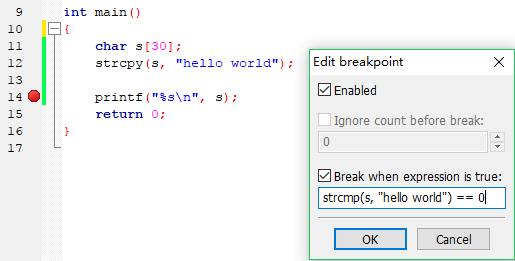
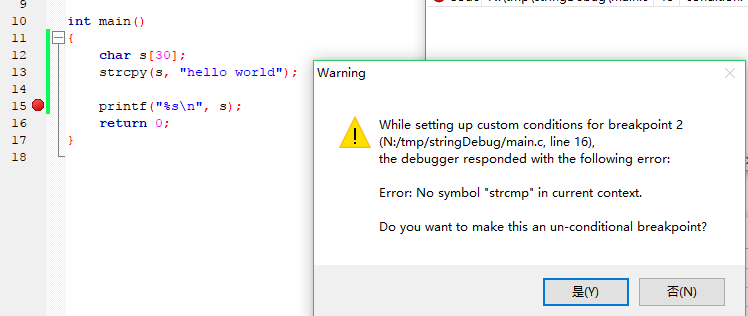
Vs的调试工具是微软自己的，感觉功能会比上面的多一点。

使用的话codeblocks和vs的调试器挺好用的，devc++的就没啥用，不能设断点条件要点continue点断手了。

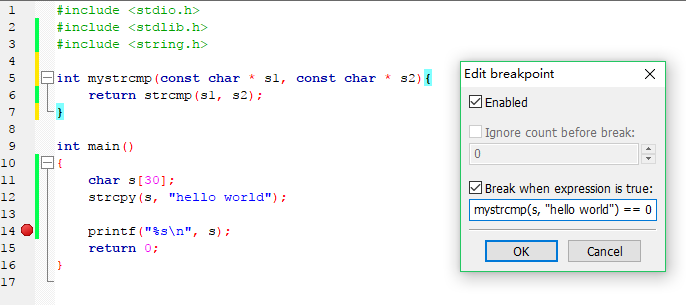
对于数据断点这个东西，其实比较微妙，对于一个普通的变量比如int, float, double什么的是可以的，不过一旦牵涉到指针就不一定有用了。比如我们常见的字符串，字符串的名字相当于是一个指针，你对这个字符串名字加数据断点是没有意义的，他并不会在你预期的情况下停下来。如果你想要在字符串等于某个值的情况下停下来，可以在合适的行上加条件断点。比如下图的程序，我想在字符串变成hello world的时候停下来



在codeblocks中：可以在14行加个断点，设置断点条件，这里面需要进行字符串比较，我试了一下这里面的表达式不会到你include的头文件里去找函数，比如我用strcmp，它就不认识

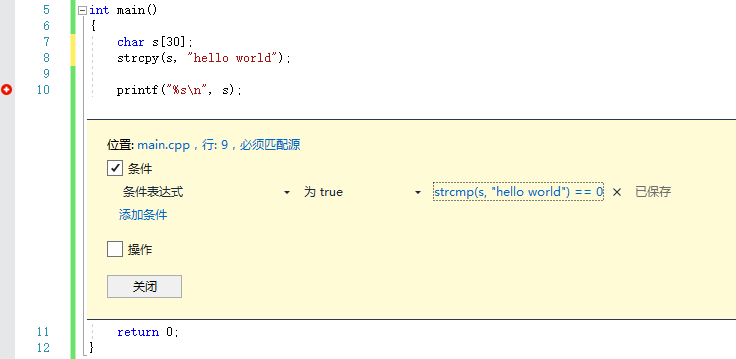
 

但他是能认识你自己写的函数的，所以可以这样搞，给strcmp包一层，如下



你自己写个函数调strcmp，然后在断点条件里写你自己的函数即可。

在vs中：vs就比较方便，你直接调strcmp就好了，别忘了include string.h。



Codeblocks的话因为用的是现成的调试器，所以需要配置调试器的路径，这个安装包里是带着的，一般没问题，不过如果你的提示找不到调试器这种问题的话，还是重装一下，注意重装的时候把配置文件删除，在

C:\Users\{你的用户名}\AppData\Roaming\CodeBlocks\

这个文件夹里，都删掉。

安装包这里下

<http://www.codeblocks.org/downloads/26#windows>

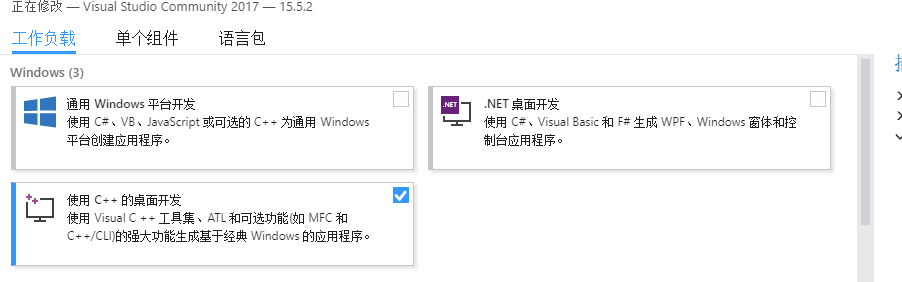
选里面的codeblocks-16.01mingw-setup.exe

其实我比较推荐用vs，功能比较强大，以后工作有可能会用得到，就是比较庞大，不熟悉的话用起来会有点摸不着头脑，不过早点熟悉总是比较好。下载可以在

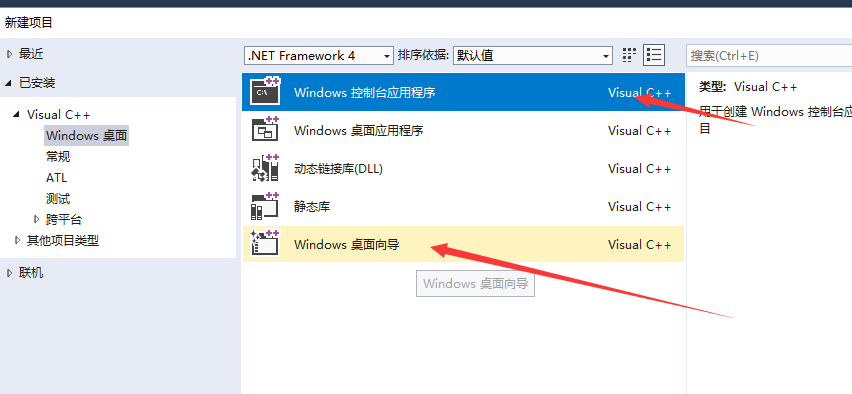
<https://www.visualstudio.com/zh-hans/?rr=https%3A%2F%2Fwww.google.co.jp%2F>

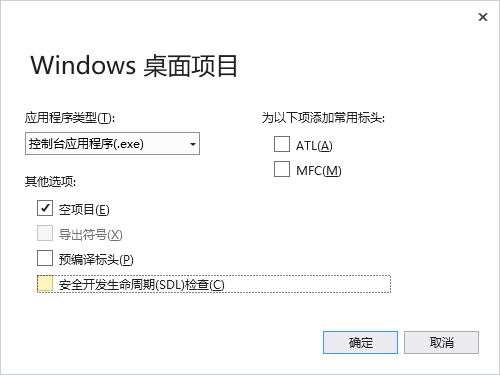
下community 2017版就好，免费的。

安装选一个c++桌面开发就好了

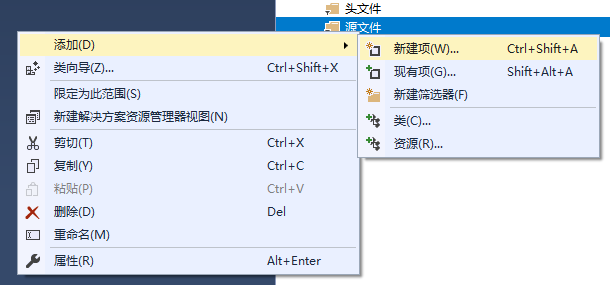


注意在vs使用的时候，最新版建工程有点不一样了，如下图，直接选控制台应用程序的话会直接创建完成并在你的项目里加一堆预编译头什么的，这对我们没什么用。要建一个空项目可以选下面的windows桌面向导。



如图这样选就ok

建完项目如下图添加个cpp就可以写代码了



我这里没有mac电脑，就没法写mac的debug了，不过codeblocks是有mac版的，可以去试一下。或者喜欢研究的可以看看gdb怎么用，这个是所有平台都有的。

## 二、debug思路

这里我把程序错误分类两类，一类是程序崩掉没结果，另一类是程序结果错误。

对于第一类错误，一般都是访问了无效的内存，比如数组访问越界、访问了无效的指针位置、程序爆栈（递归的时候调用层数太多）之类的。这种一般比较好找，用debug工具的话一般会在错误访问的地方停下在，不过不一定停在你写的函数那里，有可能停在你调用的系统函数那里，你可以在调用栈里切换到你的函数位置看是哪里出了问题。或者直接在程序各个地方加几个printf，也可以看出来程序在哪里崩掉的。

第二类错误的话，就是在各个关键位置查看变量的值，是不是对的。可以用debug工具去watch，也可以print出来。其实我一般偏向print，因为跑完我可以查看所有结果，万一有个很大的循环我也不知道哪里出了问题，debug一次次continue也比较累。不过debug的好处是你一次可以查看到所有你想看的东西，根据判断切换要关注变量。print的话有可能你这次打印的不是出问题的那个变量，可能你就得调整print语句再跑来看结果。

这里注意，如果你用print的话不要偷懒，多打几个字把输出的格式调好，这样方便看。比如printf(“i = %d, my\_value = %d\n”, i, my\_value);

Debug语句用完别急着删，可以先注释一下，后面你可能还会用到的。

一般边界条件容易出错，比如循环结束条件什么的，可以给print加个条件，比如if(i > 100){printf (“%d, “, abc);}这样的，或者可以用带条件的断点。

其实不是所有情况下你都能有debug工具可用的，print作为最方便的方法大家还是应该掌握一下的。