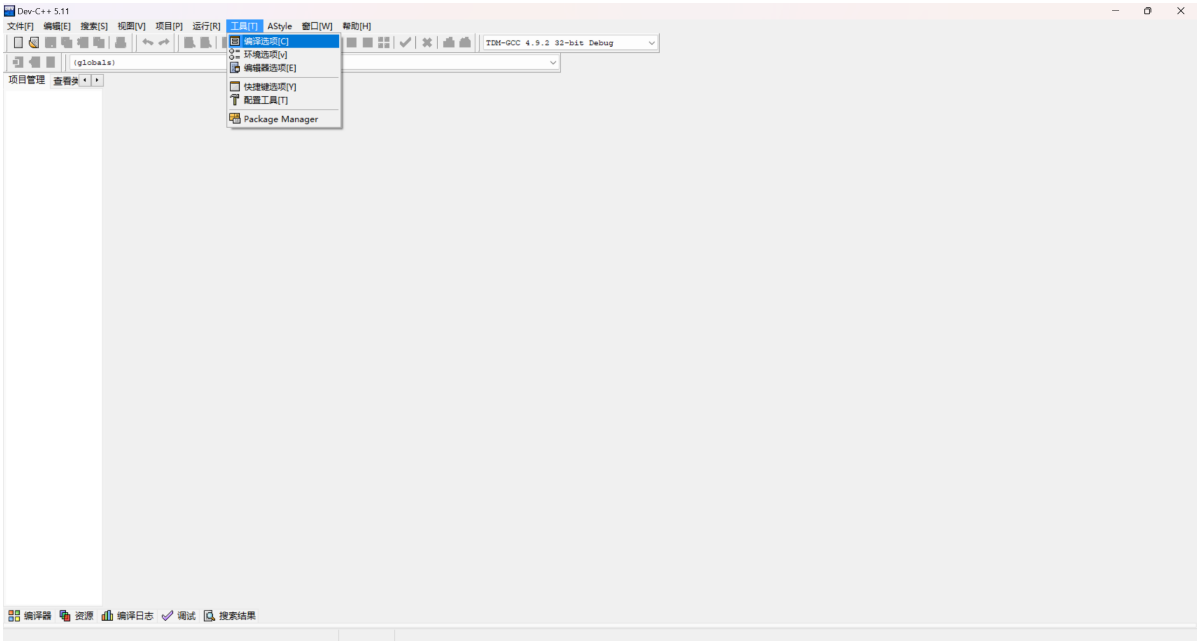
0. 编译警告

我们编程中经常遇到，`scanf("%d", &x);` 时没有写 `&`，`if(a == b)` 中少写了一个 `=` 等之类的问题，那么，我们可以开启 `-Wall` 来显示所有警告，具体操作如下：

Dev-C++：

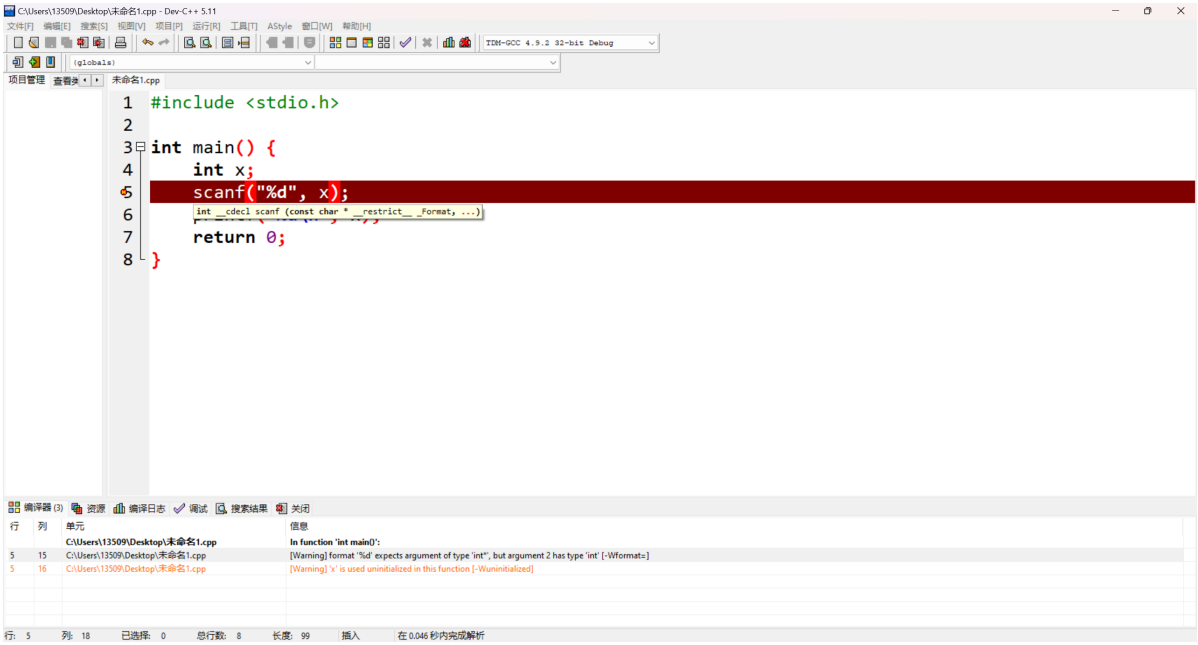
点击 工具、编译选项



点击 代码生成/优化、代码警告，把显示最多警告信息设置为 Yes

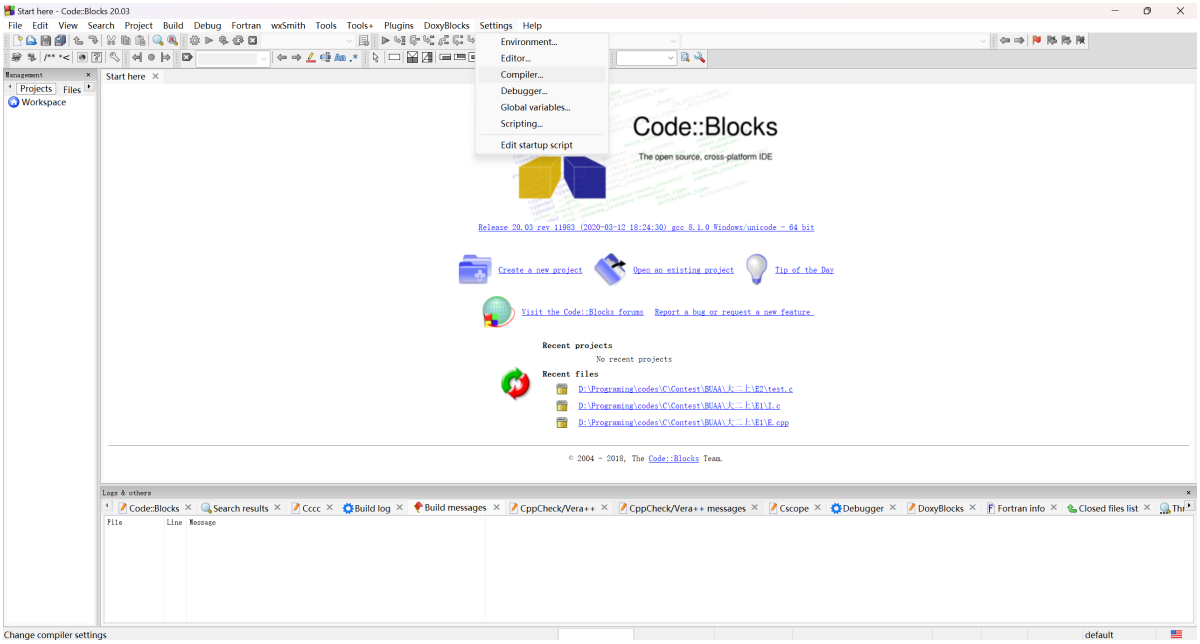
然后就会有类似如下的效果了：

双击这个橙色的警告，会给你指出哪里可能存在问题

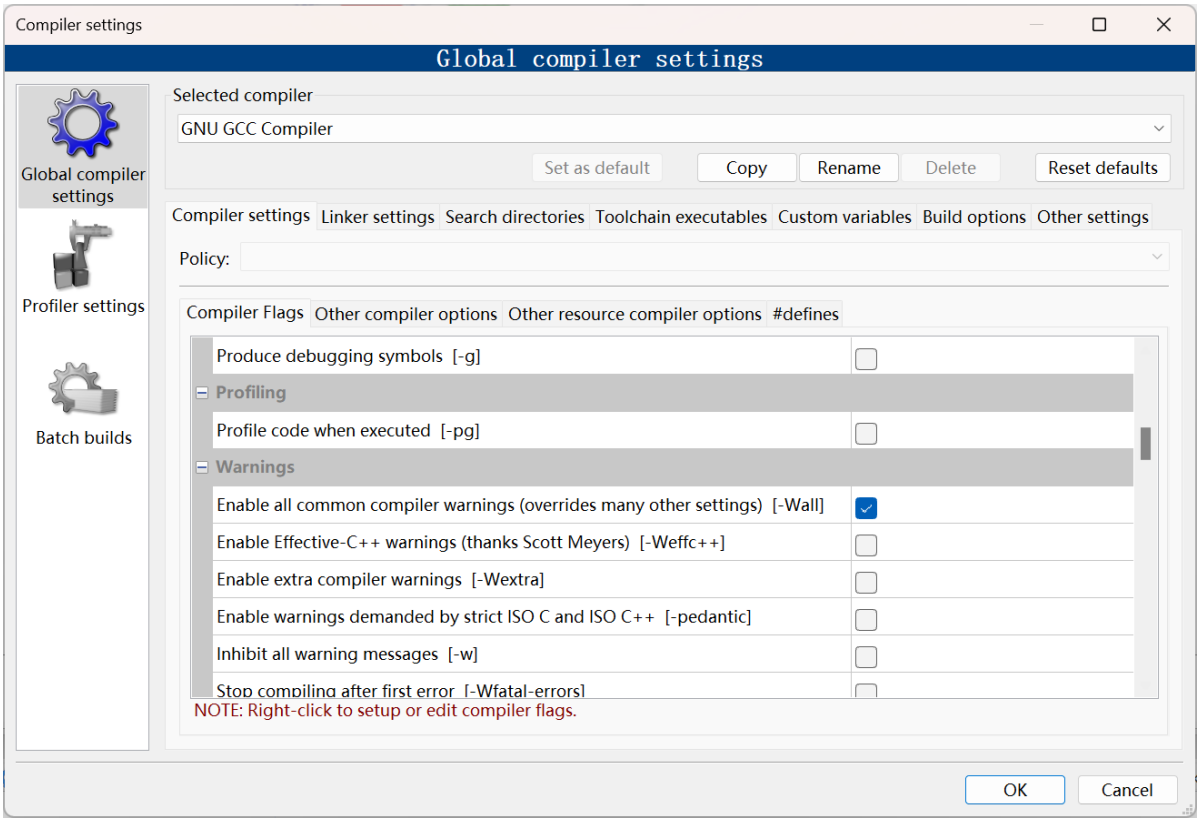


CodeBlocks：

点击 Settings、Compiler...

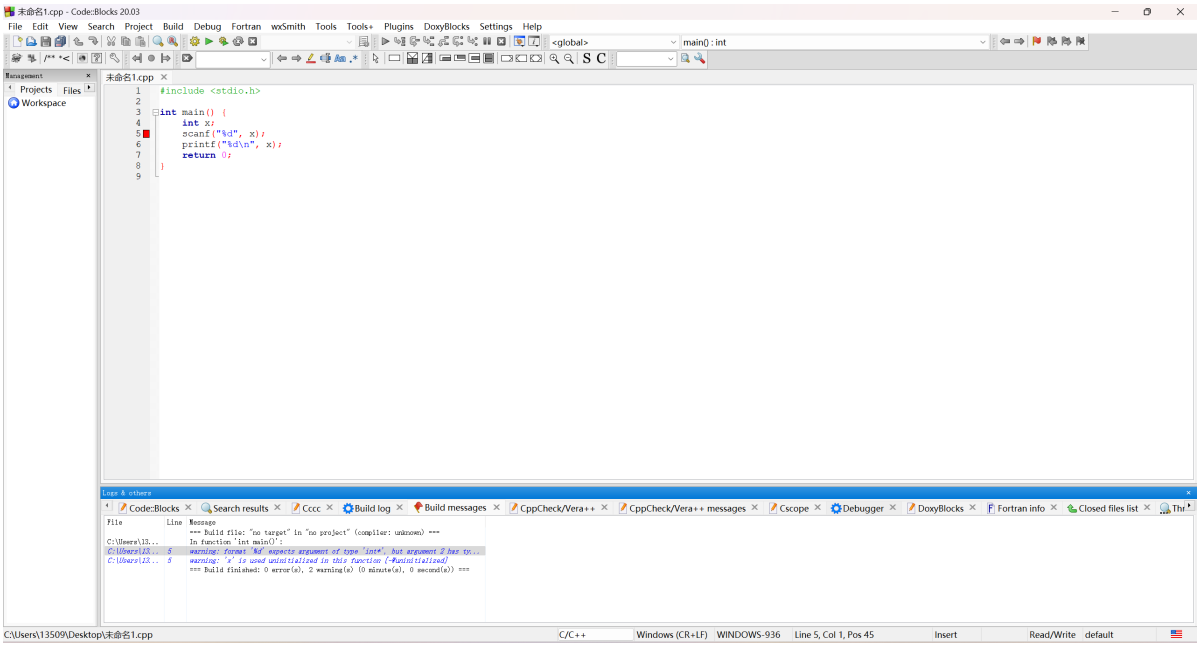


弹出的这个窗口往下滑，Warnings 里把第一个 `Enable all common compiler warnings(overrides many other settings)[-Wall]` 勾选上



然后就会有如下的效果了：

在下面的 `Logs & others` 里选择 `Build messages`，双击这个蓝色的警告，会给你指出哪里可能存在问题



1. `scanf` 输入没写 `&`

同时输入的数据也应和前面的参数匹配。

如果 `long long a;`，但是用 `scanf("%d", &a);` 输入，可能你本地跑着没有问题，但是提交上去就会直接 WA。

2. 输出语句错误

比如题目要求输出 `YE5`，而你写成了 `YES`。不要照着题目的输出文本写，请复制粘贴题目的输出文本

3. 数组初始化

`int a[100] = {-1};` 并非是将数组中所有的都赋值成 `-1`，而是按照 `{}` 中的数字依次赋值给 `a[0], a[1], a[2].....`，其余没被赋值的会被赋值为 `0`。比如说 `int a[10] = {1, 3, 5, 7};`，那么 `a[0] = 1; a[1] = 3; a[2] = 5; a[3] = 7;`，其余的 `a[4]` 到 `a[9]` 都是 `0`。因此实际上我们想要初始化为 `0`，可以直接写成 `int a[10] = {};`。

4. RE

通常是 `/` 或 `%` 运算中，除数为 `0` 导致的。

或者说数组越界了，比如说 `int a[10];` 的有效范围是 `a[0]` 到 `a[9]`，可能你的程序在某些地方访问了 `a[10]` 就会 RE（有时候，这种小范围的越界不一定会报 RE 的错误，也可能是 WA）。

5. 数据范围

请估算数据的大小，使用正确的数据类型。

数据类型	min	max
int	-2,147,483,648	2,147,483,647
unsigned int	0	4,294,967,295
long long	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
unsigned long long	0	18,446,744,073,709,551,615

`double` 的精度为 16~17 位有效数字。

赋值语句是先计算 `=` 右边的值，再赋值给左边的变量，同时他的计算只会根据右边计算式的最高级数据类型进行计算，与左边的变量无关。比如说 `1<<63` 就是错误的，因为我们直接写出来的数字都是 `int` 类型的，如果想要写出 `long long` 范围的值，需在数字后面加 `ll`（两个小写 `L`），例如，`1ll` 就是 `long long` 类型的 `1`。其余的一些：`unsigned` 类型，数字后加 `u`；`unsigned long long` 类型，数字后加 `ull`。

下面是浮点数计算时同学们可能犯的错误，请结合上述材料自行理解：

```c
int a = 5, b = 3;
double c;
c = a / b;
printf("%.5f\n", c);

c = (double)(a / b);
printf("%.5f\n", c);

c = (double) a / b;
printf("%.5f\n", c);

c = 1.0 * a / b;
printf("%.5f\n", c);
```