0.11 — Configuring your compiler: Warning and error levels

BY ALEX ON SEPTEMBER 19TH, 2018 | LAST MODIFIED BY NASCARDRIVER ON DECEMBER 27TH, 2019

When you write your programs, the compiler will check to ensure you've followed the rules of the C++ language (assuming you've turned off compiler extensions, as per lesson **0.10 -- Configuring your compiler: Compiler extensions**).

If you have done something that definitively violates the rules of the language, during compilation the compiler will emit an **error**, providing both line number containing the error, and some text about what was expected vs what was found. The actual error may be on that line, or on a preceding line. Once you've identified and fixed the erroneous line(s) of code, you can try compiling again.

In other cases, the compiler may find code that seems like it might be in error, but the compiler can't be sure (remember the motto: "trust the programmer"). In such cases, the compiler may opt to issue a **warning**. Warnings do not halt compilation, but are notices to the programmer that something seems amiss.

Best practice

Don't let warnings pile up. Resolve them as you encounter them (as if they were errors).

In most cases, warnings can be resolved either by fixing the error the warning is pointing out, or by rewriting the line of code generating the warning in such a way that the warning is no longer generated.

In rare cases, it may be necessary to explicitly tell the compiler to not generate a particular warning for the line of code in question. C++ does not support an official way to do this, but many individual compilers (including Visual Studio and GCC) offer solutions (via non-portable #pragma directives) to temporarily disable warnings.

By default, most compilers will only generate warnings about the most obvious issues. However, you can request your compiler be more assertive about providing warnings for things it finds strange.

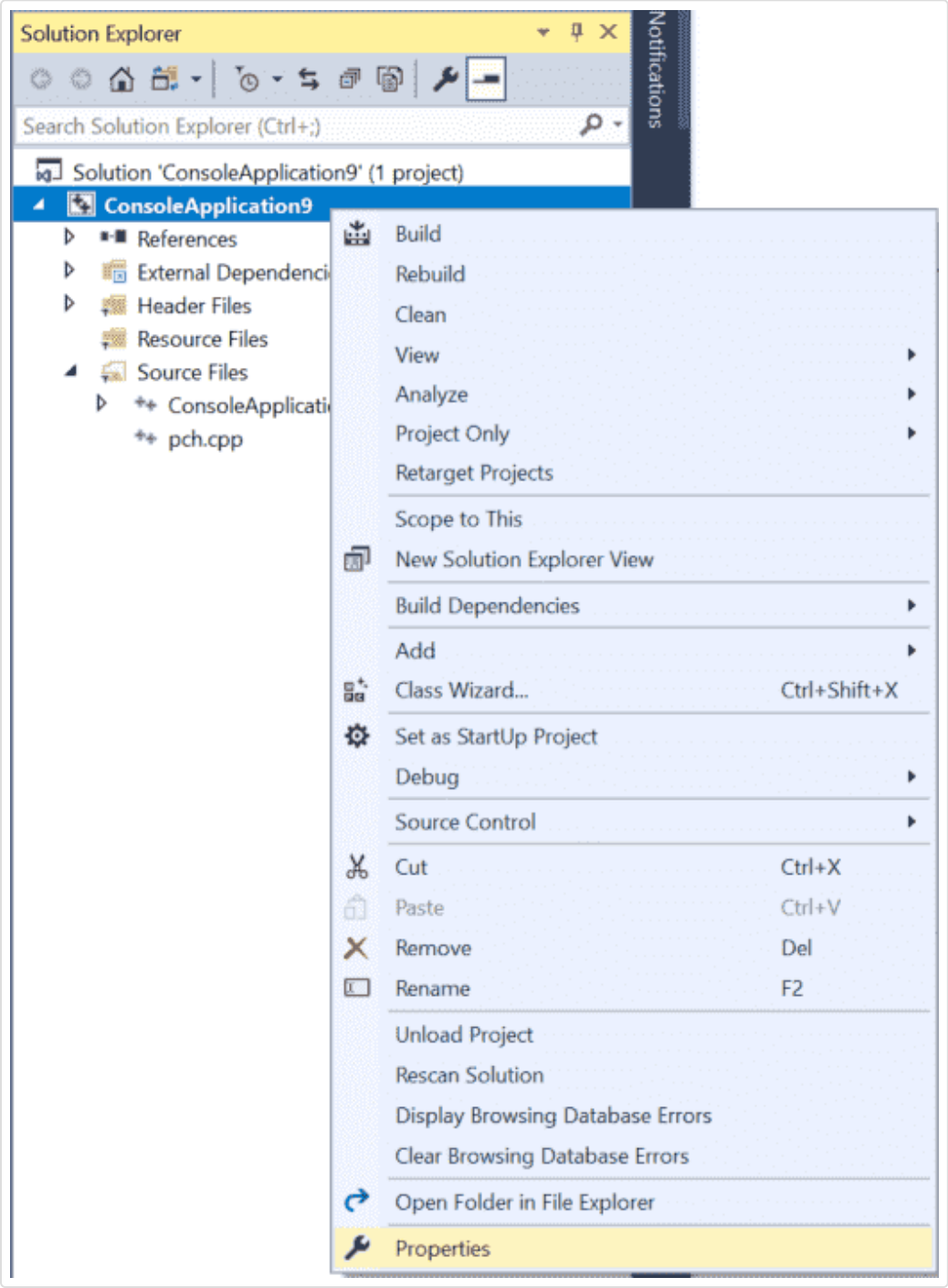
Best practice

Turn your warning levels up to the maximum, especially while you are learning. It will help you identify possible issues.

Increasing your warning levels

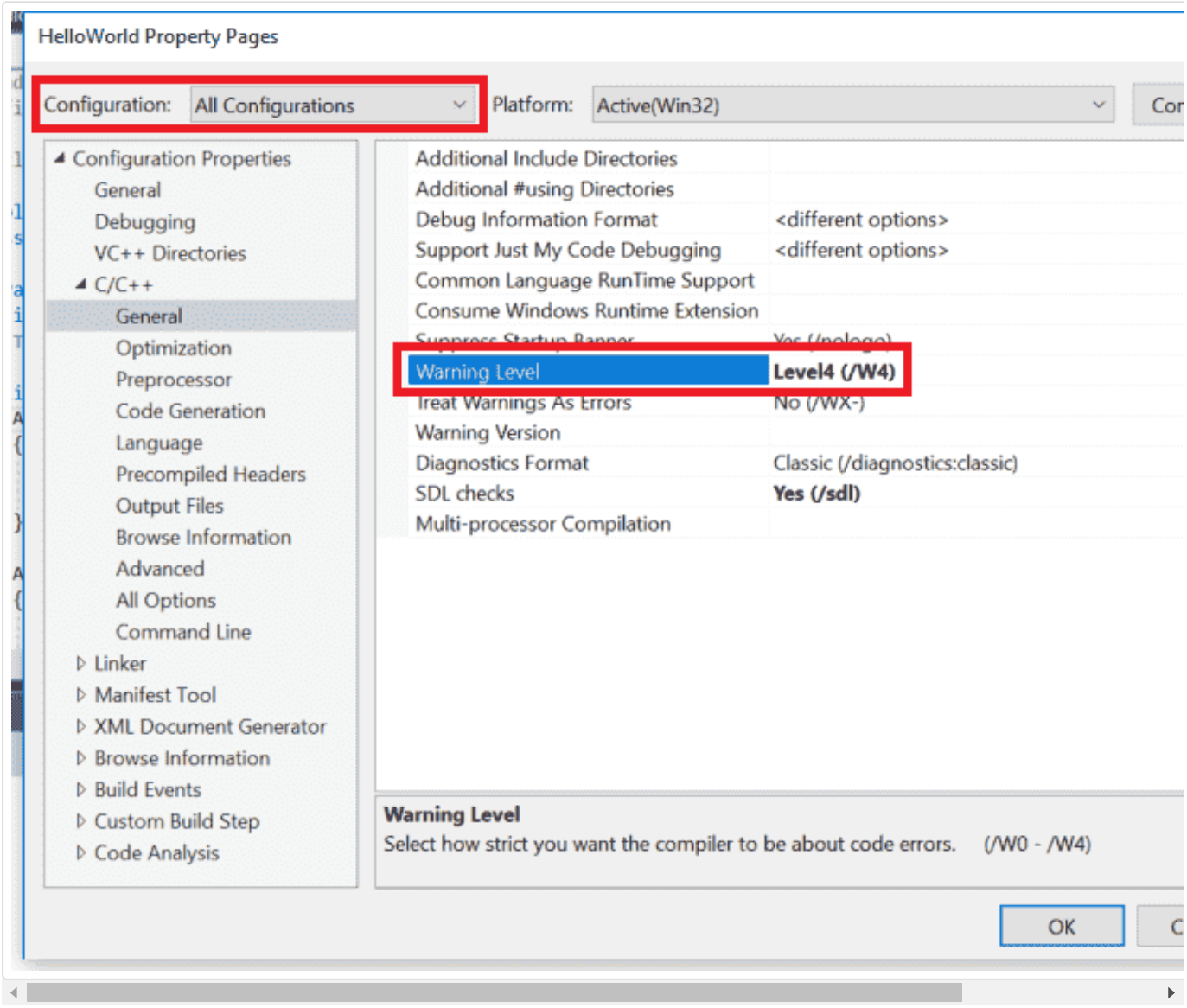
For Visual Studio users

To increase your warning levels, right click on your project name in the *Solution Explorer* window, then choose *Properties*:



From the *Project* dialog, first make sure the *Configuration* field is set to *All Configurations*.

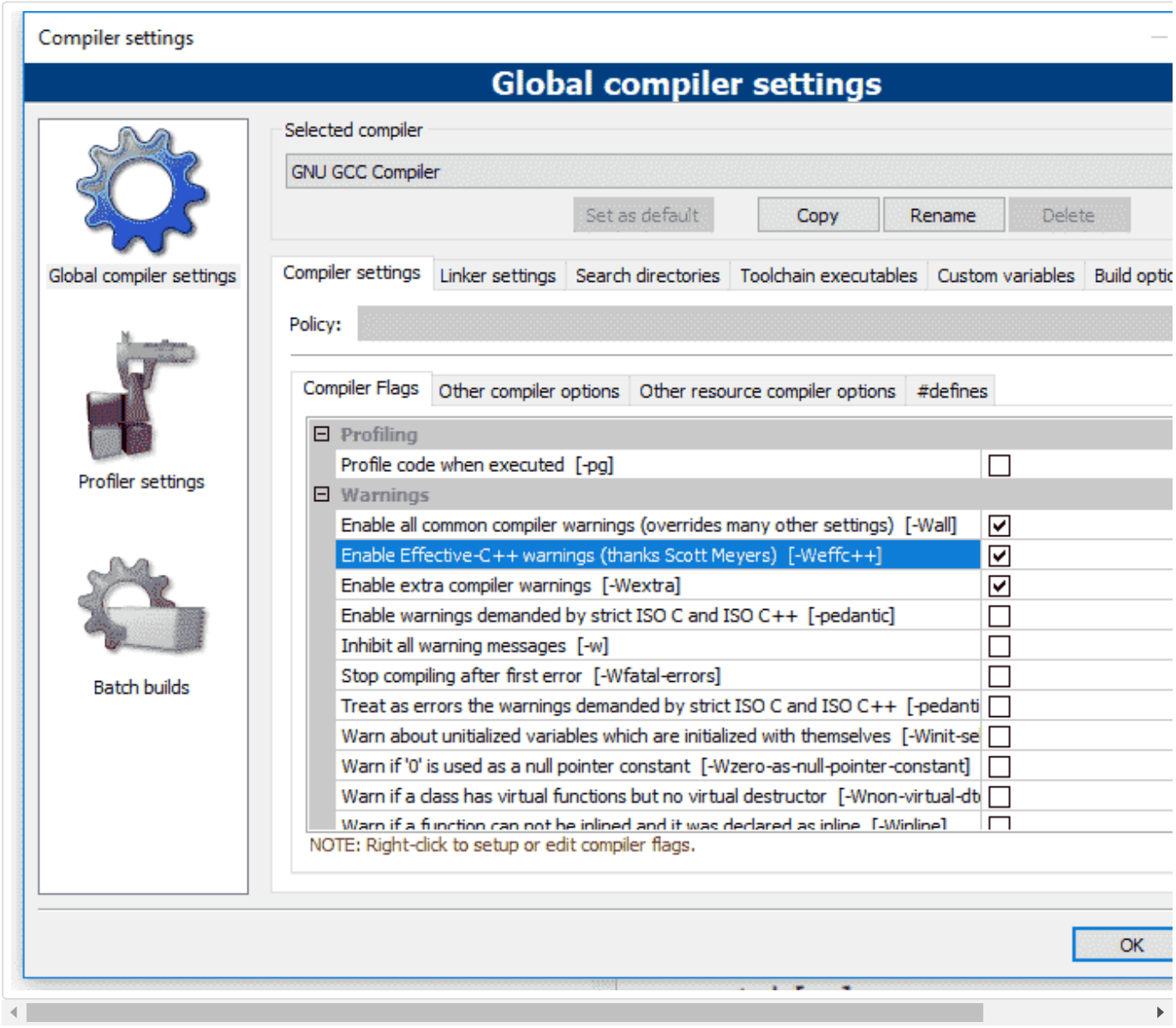
Then select *C/C++* > *General tab* and set *Warning level* to *Level4 (/W4)*:



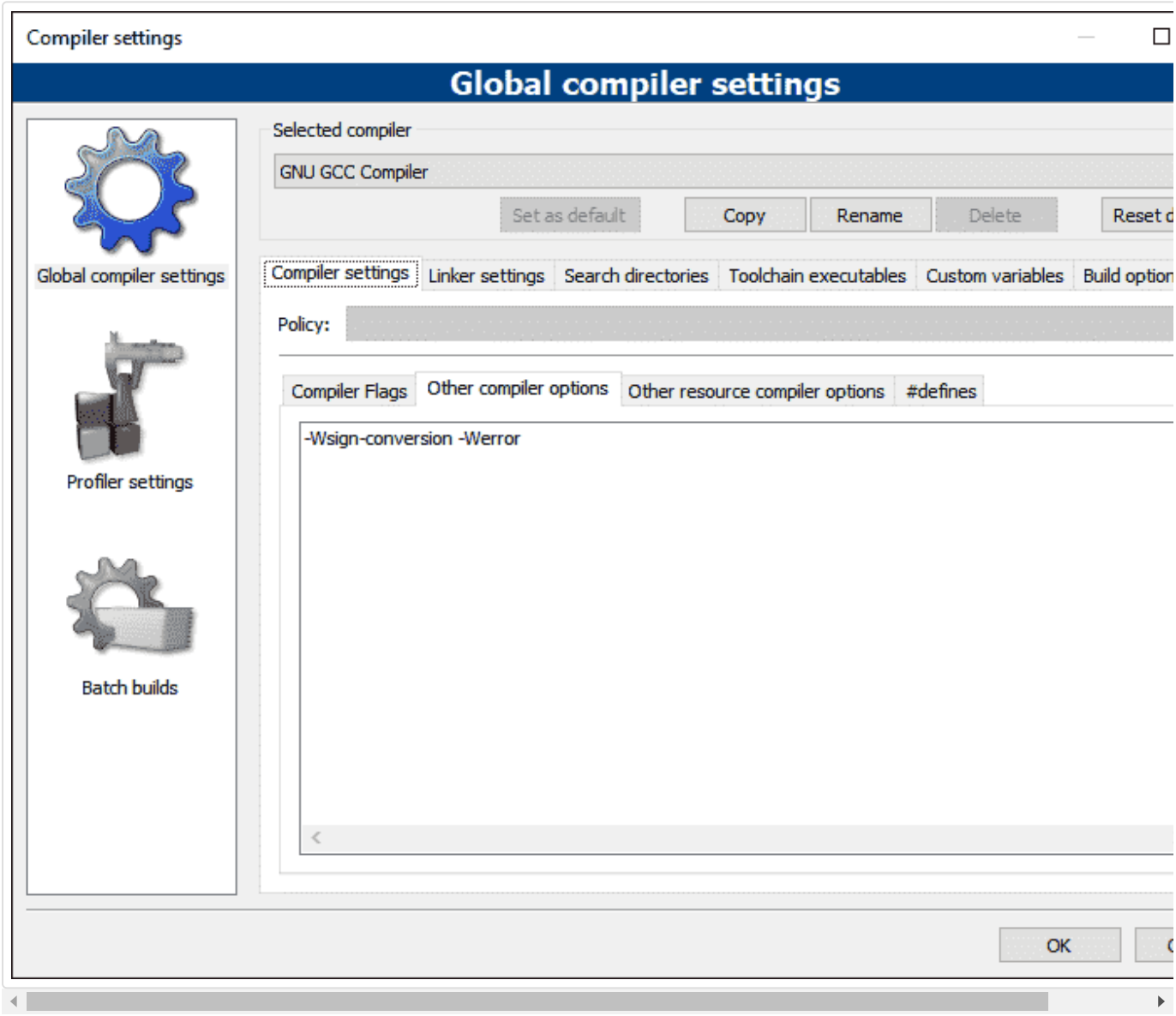
Note: Do not choose *EnableAllWarnings (/Wall)* or you will be buried in warnings generated by the C++ standard library.

For Code::Blocks users

From *Settings menu* > *Compiler* > *Compiler settings tab*, find and check the options that correlate with *-Wall*, *-Weffc++*, and *-Wextra*:



Then go to the *Other compiler options tab*, and add *-Wsign-conversion* to the text edit area:



Note: The *-Werror* parameter is explained below.

For GCC/G++ users

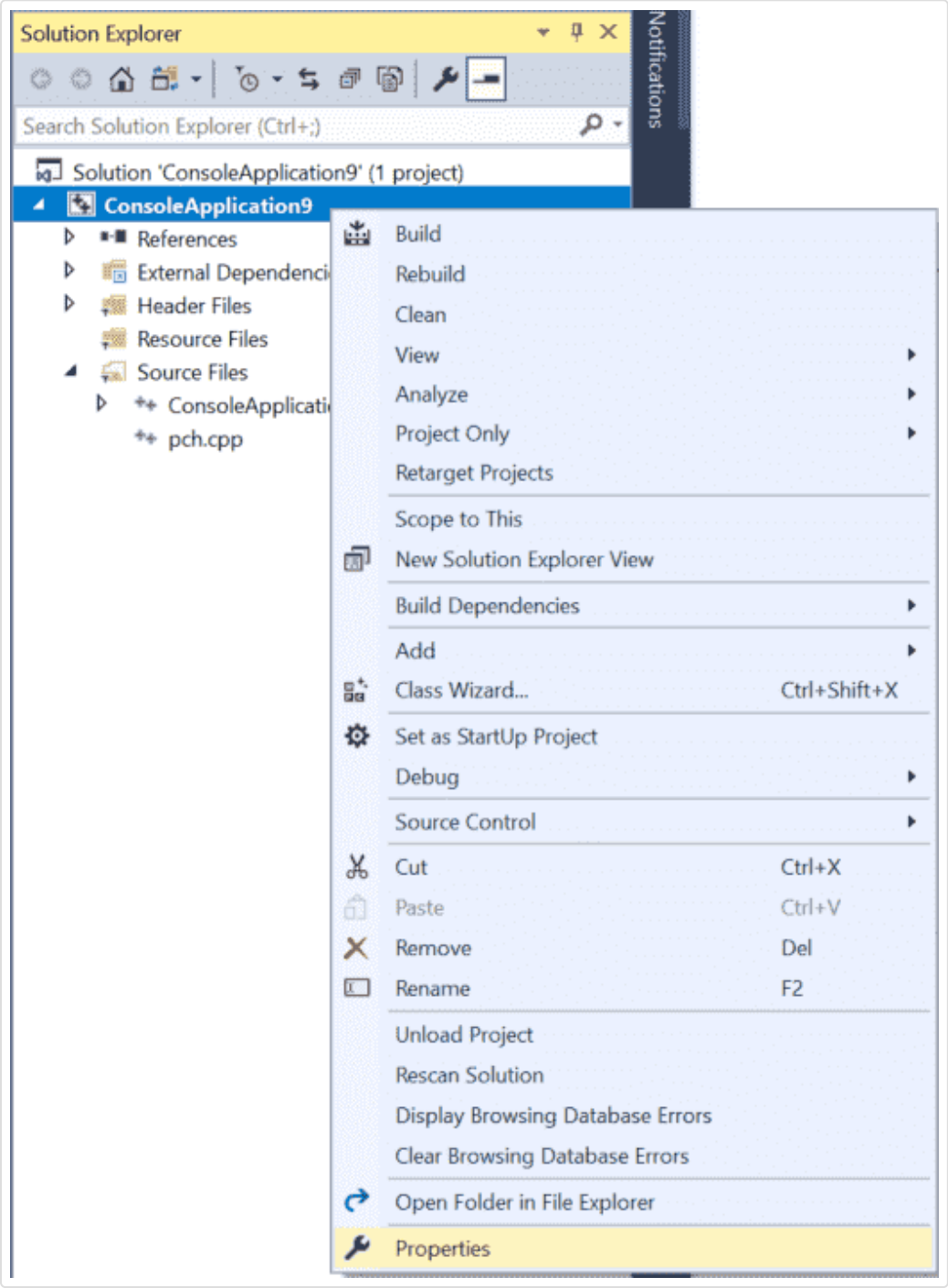
Add the following flags to your command line: *-Wall -Weffc++ -Wextra -Wsign-conversion*

Treat warnings as errors

It is also possible to tell your compiler to treat all warnings as if they were errors (in which case, the compiler will halt compilation if it finds any warnings). This is a good way to enforce the recommendation that you should fix all warnings (if you lack self-discipline, which most of us do).

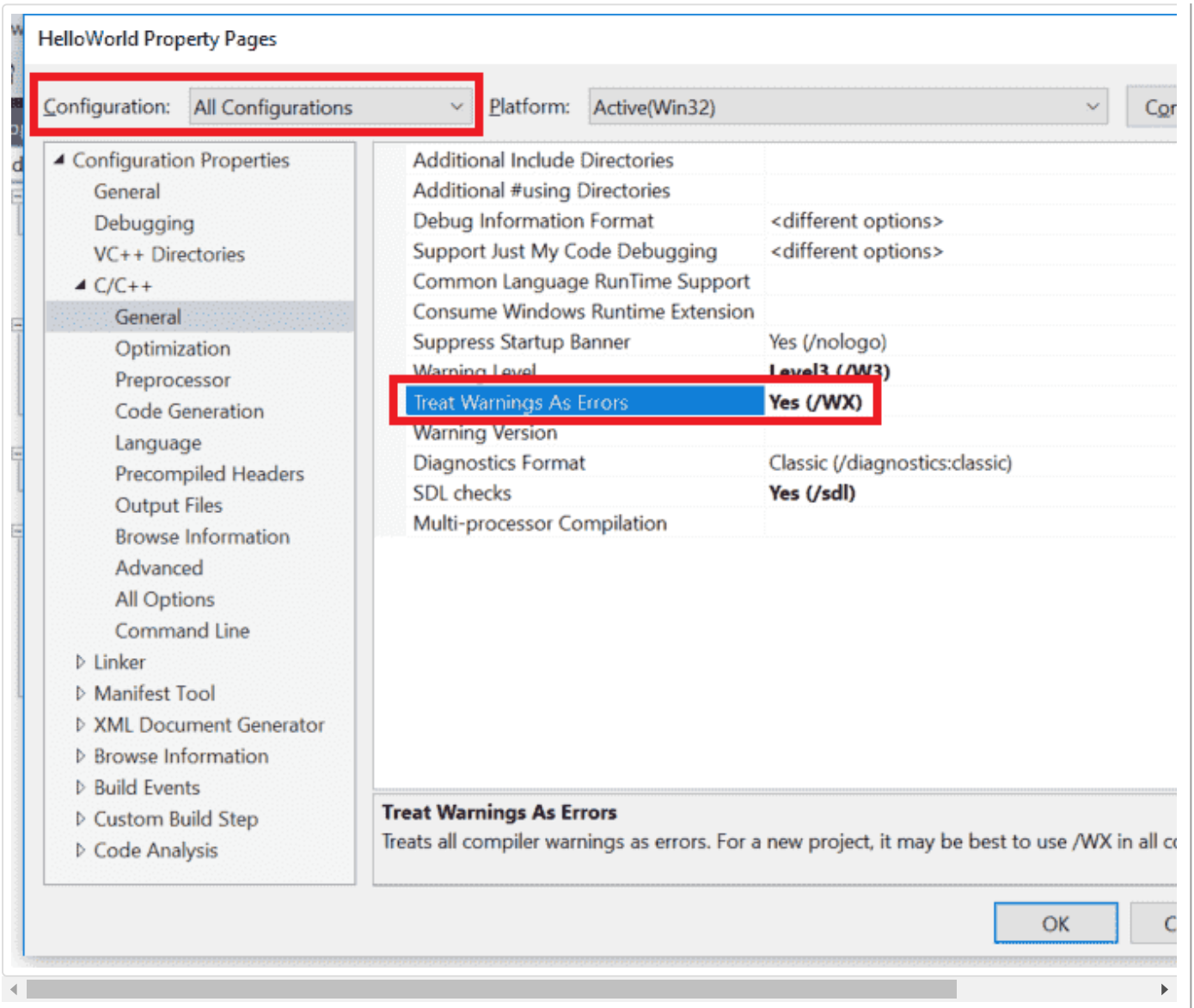
For Visual Studio users

To treat warnings as errors, right click on your project name in the *Solution Explorer* window, then choose *Properties*:



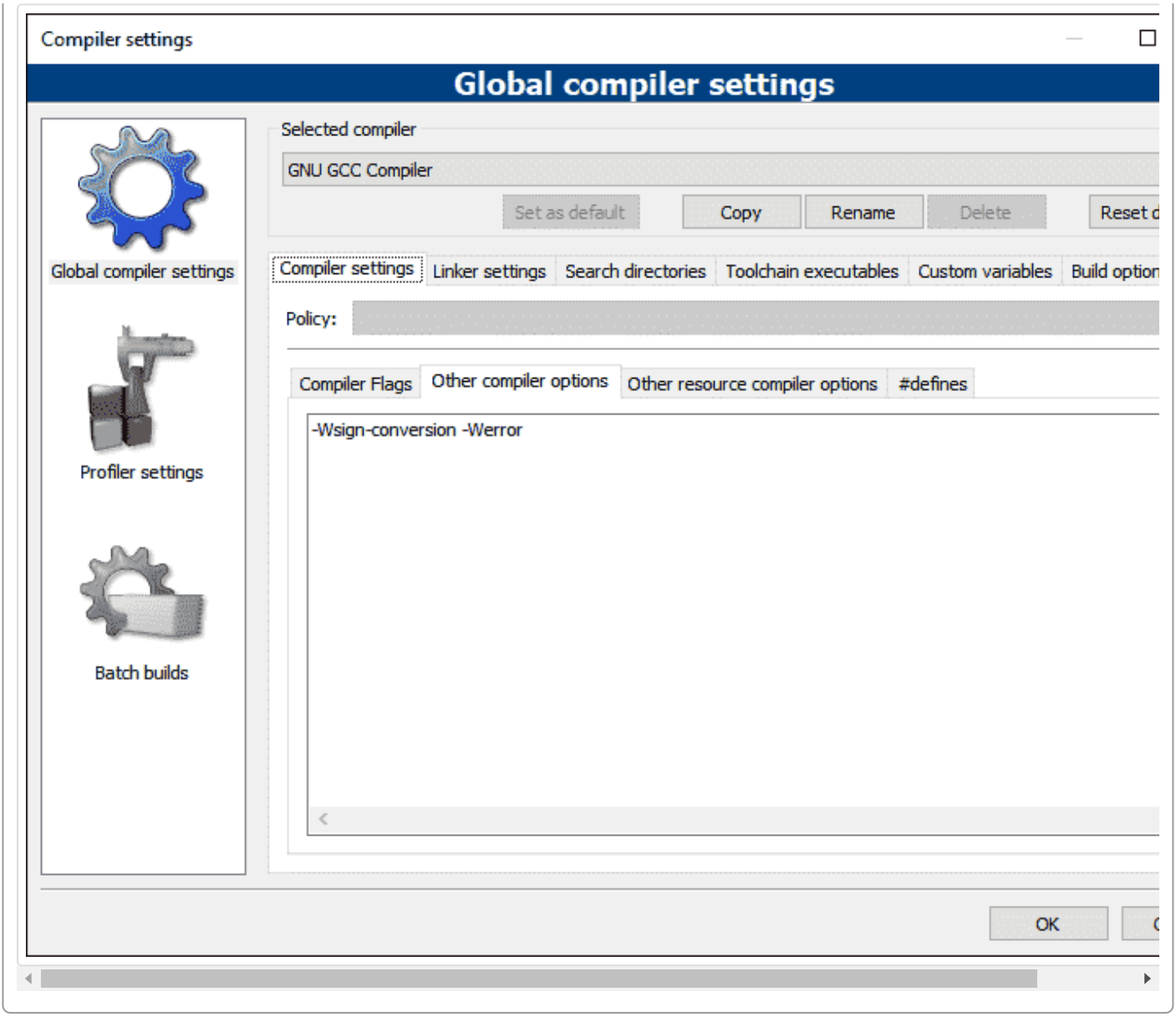
From the *Project* dialog, first make sure the *Configuration* field is set to *All Configurations*.

Then select *C/C++* > *General tab* and set *Treat Warnings As Errors* to *Yes (/WX)*.



For Code::Blocks users

From *Settings menu* > *Compiler* > *Other compiler options tab*, add *-Werror* to the text edit area:



For GCC/G++ users

Add the following flag to your command line: *-Werror*

0.12 -- Configuring your compiler: Choosing a language standard

Index

0.10 -- Configuring your compiler: Compiler extensions

C++ TUTORIAL | PRINT THIS POST

60 comments to 0.11 — Configuring your compiler: Warning and error levels

Rowan
December 30, 2019 at 1:41 am · Reply

Hello,

I'm following these tutorials using Visual Studio 2019. Should I enable the 'SDL checks' option?

The tooltip says it is recommended and that it can improve security. This is also stated on the MS Docs Page for the feature: https://docs.microsoft.com/en-us/cpp/build/reference/sdl-enable-additional-security-checks?view=vs-2019

In your Visual Studio screenshots on this page, I notice you have SDL checks enabled; it is set to 'Yes' (/sdl).

Thanks.

Alex
January 2, 2020 at 2:56 pm · Reply

I believe the latest version of Visual Studio 2019 has /sdl enabled by default, which is why it's enabled in the screenshot.

Personally, I leave it at the default setting. You can always disable it if it gets in the way. Do note that it may mask bugs in your code that could show up if/when you port your code to other compilers, and there may be a slight performance impact from having it enabled.

Rowan
January 3, 2020 at 5:46 am · Reply

Thanks for your response.

Considering that it may mask bugs, and that I would prefer my code to be as bug-free as possible, I will disable the SDL checks option.

Vitaliy Sh.
December 25, 2019 at 5:17 am · Reply

"Note: The -Werror parameter is explained below."

```
1 <em>-Werror</em>
```

?

nascardriver
December 27, 2019 at 7:53 am · Reply

Emphasized, thanks!

Crosstak
November 30, 2019 at 5:58 am · Reply

I can't say just how helpful this whole tutorial has been. Thank you Alex and nascardriver.
I have run into a little problem that has been confusing me. I know this is technically C code but I was just messing around in Ubuntu using the terminal and I made a file named Test.c containing this code:

```
#include <stdio.h>

int main(int argc, char *argv[]){
    printf("Hello World\n");
}
```

I compiled it by entering this line in the terminal: "gcc -o Test Test.c -Wall -Werror". I expected to get an error because of it not having a return value but it compiled fine. I then entered "./Test" then "echo $?" in the terminal and these printed the lines "Hello world" and "0" respectively. Is there a reason why my code is assuming a return value of 0 and not throwing an error?

nascardriver
November 30, 2019 at 6:10 am · Reply

If `main` finishes without having a return-statement, it returns 0 (Also in C++). `main` is the only function that behaves like this. Unless you're testing something or writing an example, you should `return 0;` for consistency with other functions.

Crosstak
November 30, 2019 at 6:43 am · Reply

Thank you very much

Vitaliy Sh.
November 13, 2019 at 11:05 am · Reply

There was a reference to Makefiles in the lesson 0.5, in "For advanced readers".

Can you please, add some reference to writing the shell scripts for a GCC/g++, for the educational and day-by-day purposes?

```
#!/bin/sh

# See the g++ manual page.

echo "g++ started with following arguments:"

declare -a all_flags

i=0

for flag in                 \
    -std=c++17          \
    -g                  \
    -pedantic-errors    \
    -Wall               \
    -Wextra             \
    -Weffc++            \
    -Wsign-conversion   \
    -Werror             \
    ${@}
do
    {
```

```
        [[ {flag} =~ -.* ]]         \
        && echo -e "\t{flag}"
    }   || echo -e "\t\t${flag}"

    all_flags[${i}]={flag}

    i=$(( ${i} + 1 ))
done

echo -e "\nStart Compilation? [0/1]?"

[[ ${x} == 1 ]]     \
&&  g++ ${all_flags[@]}
```

nascardriver
November 14, 2019 at 1:39 am · Reply

This script is compiler-specific and not very human friendly. I don't see the need for it when there are friendlier and compiler-independent alternatives like CMake.

Vitaliy Sh.
November 14, 2019 at 5:46 am · Reply

You are obviously right...
I were confused by:
... Add the following flag to your command line: -Werror ...
I did mistake by separating the "start g++ from command line" and the Makefiles as opposite things.

Vitaliy Sh.
November 13, 2019 at 7:10 am · Reply

... > Compiler >Compiler settings tab, ...

Typo: ">Compiler" - no space.

Jim
November 12, 2019 at 7:20 pm · Reply

Hey Alex,
Have been away and just coming back to C++. I'm using C::B and there is a note in yellow that says: For GCC/G++ users

Add the following flags to your command line: -Wall -Weffc++ -Wextra -Wsign-conversion.

I didn't see it shown on this lesson so should we add "-Wall -Weffc++ Wextra" to -Wsign-conversion that is shown there also?

I'm 81 and not very good at programming but I still enjoy your lessons. Thanks

nascardriver
November 13, 2019 at 3:48 am · Reply

Hey Jim,

yes, add all 4 in code blocks. Also enable -pedantic-errors.

Lee rV
October 26, 2019 at 3:19 pm · Reply

For laughs, I turned on /wall in a simple project that uses <iostream>, <string>, and <ctype.h>.

The compiler came back with over 200 warnings! Combined with "Treat warnings as errors", and I was hilariously out of luck, as I'm way too much of a newbie to start re-writing the standard library to be "correct"!

"Wall", indeed!

nascardriver
October 27, 2019 at 1:23 am · Reply

Your compiler shouldn't warn you about anything in system headers, only about issues in your own code. This sounds like a misconfigured compiler/project.

Anastasia
August 28, 2019 at 6:17 am · Reply

Hi!
Which flags/options it would be recommended to use for clang (command line, no IDE) with as many warnings as possible (not treated as errors though), all compiler extensions turned off, fastest compilation and maybe some additional diagnostic information (if possible)?

nascardriver
August 28, 2019 at 6:52 am · Reply

clang uses the same cli as gcc. If you had those settings for gcc, you can use them for clang as well.

> as many warnings as possible
-Weverything
That's not recommended though, it will flood you with warnings. You can use -Weverything to find the flags that are relevant to you.
I use

```
-Wall
-Wextra
-Weffc++
-Wconversion
-Wmissing-variable-declarations
```

> all compiler extensions turned off

```
-pedantic-errors
```

> fastest compilation
If you mean that you want your program to run fast, -O3 is the highest optimization. If you really mean fast compilation, I don't know, I suppose it always tries to be fast. llvm offers a drop-in replacement for ld, you can use that to link faster. You won't notice a difference with small projects.

> some additional diagnostic information
clang-tidy might be what you're looking for. It works well with CMake (CMake allows you to configure the compiler without knowing which compiler you're using. You don't need to worry about command line options or manually listing your files every time you want to compile).

Anastasia
August 28, 2019 at 7:09 am · Reply

> If you had those settings for gcc, you can use them for clang as well.
Oh, I didn't know that. I browsed quickly through the man pages, but there are only a few flags and options mentioned in comparison to gcc.

> If you mean that you want your program to run fast
No, I meant without options that would negatively impact the compilation speed. Optimization does impact it and makes the code harder to debug (the man says).

> CMake allows you to configure the compiler without knowing which compiler you're using.
I think it is important now when I'm only learning to do everything manually and understand how things work. I wouldn't learn c++ if I was looking for easy ways :D
It's noted though for the future.

> llvm offers a drop-in replacement for ld, you can use that to link faster.
> clang-tidy might be what you're looking for.
I'll look into these for sure, thank you!

edit: @Alex, can you please add to this lesson that clang users should use the same flags as gcc? Thanks!

Vitaliy Sh.
November 14, 2019 at 7:41 am · Reply

Me isn't want to flood, but:
In a "Qt Creator" there is preset:
-Weverything and 12th -Wno-XXXes.

Is that was a wrong choice? The IDE looks fancy.

The Clang's "User Manual" says:

...
Using -Weverything means that updating your compiler is more difficult because you’re exposed to experimental diagnostics which might be of lower quality than the default ones.
...

nascardriver
November 14, 2019 at 7:49 am · Reply

-Weverything is more than you should use. The -Wno* might get rid of the unnecessary warnings. Post you code when you solve quizzes and I'll tell you if you're doing something that a warning told you to do when it's actually not good.

Marie Bethell
August 23, 2019 at 3:16 pm · Reply

Hello!
To save my error and warning settings, I created a user template in Code::Blocks. But I noticed that now in my Code::Blocks folder, instead of each new project having it's own folder, it just dumps all the source files and headers together (no separation between projects). Before each project had it's own folder with all dependent files in it as well. Any idea what's going on here? Thanks!

Rahab
June 30, 2019 at 12:38 pm · Reply

When I choose all configuration I do't find the C/C++ option, therefore I couldn't disable the compiler extensions nor could I do any of the instructions on this lesson. why Can't I find It?

Alex
July 1, 2019 at 12:12 pm · Reply

Did you install the C++ workload when you installed Visual Studio?

Rahab
July 2, 2019 at 4:34 am · Reply

No, I just walked step by step installing the visual studio, what's the C++ workload and how do I install It ?

Alex
July 6, 2019 at 9:57 am · Reply

It's what's described in lesson 0.6 -- Installing an IDE. Are you able to create a C++ project at all?

Rahab
July 7, 2019 at 4:36 am · Reply

yes i created the first project

Alex
July 9, 2019 at 10:24 am · Reply

I'm not sure then. Maybe skip it for now and continue.

Rahab
July 27, 2019 at 6:55 am · Reply

okay thank you

winwin
June 16, 2019 at 9:27 pm · Reply

In the first paragraph:(assuming you've turned off compiler extensions, as per lesson 0.10 -- Configuring your compiler: Compiler extensions).
Is word 'per' mistyped? It should be 'pre lesson'.

nascardriver
June 17, 2019 at 4:32 am · Reply

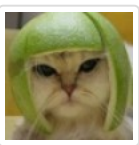
"per" is a word too. It's like "as instructed in" or "via". "pre" doesn't work here.

Charlton
June 13, 2019 at 10:26 am · Reply

If we have -std=c++17 in other compiler options in code::blocks. Should -Wsign-conversion -Werror go on the same line or on a new line.

-std=c++17 -Wsign-conversion -Werror
or
-std=c++17
-Wsign-conversion -Werror

Does it even matter?



Alex
June 18, 2019 at 2:28 pm · Reply

Don't think it matters.

Divyansh
May 30, 2019 at 5:46 am · Reply

Do we have to change these settings (mentioned above) every time we make a new project or these setting will be applicable on all new projects ?

nascardriver
May 30, 2019 at 6:03 am · Reply

These settings are per-project.
You can set them once, save the project as a template, and when you create a new project you create it from that template.

Flynn
May 4, 2019 at 8:08 am · Reply

Hello again, was just wondering how I could go about adding the additional warnings you talked about in the above lesson.
Thanks in advance, Flynn.

JS
April 16, 2019 at 3:43 am · Reply

People who use the command line for compiling can create a shortcut like this. In your .bashrc or .bash_profile (found in the home directory), add following

```
# switch to your home directory
cd
vi .bash_profile
# at the end of the file add following
alias gpp='g++ -pedantic-errors -Wall -Weffc++ -Wextra -Wsign-conversion -Werror'
```

Now you can use

```
gpp hello-world.cpp
```

to compile your program.

Ryan
March 5, 2019 at 1:20 pm · Reply

Hi, when I use

```
g++ -Wall -Weffc++ -Wextra -Wsign-conversion
```

, I get the error:

```
clang: error: no input files
```

. Am I doing this right?

nascardriver
March 6, 2019 at 5:25 am · Reply

You need to tell g++ which files you want to compile

```
g++ -std=c++2a -Wall -Wextra -Weffc++ -Wconversion -pedantic-errors ./main.cpp
//                                                                  ^^^^^^^^^^
```

Zaki
February 25, 2019 at 1:09 am · Reply

Hello sir, I have 2 questions:
1. What is the function of -fexecptions flags when I compile the project and can I omit that ?
2. For debug version of the project, is it necessary to use -g flag?

Thanks for the answers in advance

nascardriver
February 25, 2019 at 5:28 am · Reply

Hi Zaki!

1. It enables exception handling. It's enabled by default for projects written in C++, you can ignore it.
2. If you want to debug your program, yes. -g adds debugging information (eg. function names) to the executable. Without those, it can be hard to figure out what's happening.

Bob
October 26, 2018 at 4:48 pm · Reply

Do we need to worry about this with Xcode? The warnings seem to be already pretty aggressive~

Alex
October 28, 2018 at 4:46 pm · Reply

I don't have any familiarity with Xcode. Perhaps another reader can answer.

Chris
December 30, 2018 at 1:56 pm · Reply

I'm having the same issue, none of the examples they've given in the past 4 sections has applied to Xcode, at least not that I can find.

xing zhang
January 8, 2019 at 2:16 am · Reply

you can set treat warning as error in xcode and the path is build setting--> treat warnning as error(just type in 'treat warning' to filter)

I'm in love with alex's tutor
October 26, 2018 at 3:46 am · Reply

Dear Mr.Alex!
Really Thank you so much, this really helps me a lot to find out minor mistakes of mine . This increases a programmer perfection to make his program free against any bugs.
Bless you forever Sir!

Qluefqen
October 20, 2018 at 4:40 am · Reply

When I try to use this flag:

```
-Wextra-Wsign-conversion
```

I get this error:

```
g++: error: unrecognized command line option ‘-Wextra-Wsign-conversion’; did you mean ‘-Wno-sign-conversion’?
```

So I tried it with

```
-Wno-sign-conversion
```

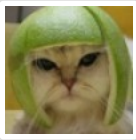
and got no errors or warnings but am not clear on what that flag is. Can you help?

Compiler is

```
g++ (Debian 8.2.0-7) 8.2.0
```

. Running Debian Buster (Testing) KDE and using Kate and Konsole if that makes any diff.

PS: Made a minor edit and screwed up the formatting without changing it. Sorry.



Alex
October 22, 2018 at 1:34 pm · Reply

There should be a space between -Wextra and -Wsign-conversion.

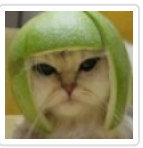
vbot
September 25, 2018 at 6:06 am · Reply

Hi,

enabling all warnings (/Wall) in VS 2017 15.8.5 will throw hundreds of MSVC-warnings that are turned off by default. See:
https://docs.microsoft.com/en-us/cpp/preprocessor/compiler-warnings-that-are-off-by-default?view=vs-2017

here it says: "The C runtime libraries and the C++ standard libraries are intended to emit no warnings only at warning level /W4."

Greets



Alex
September 25, 2018 at 8:55 am · Reply

I appreciate the warning (ha ha). I've updated the article accordingly. Thank you much!

William
September 23, 2018 at 5:26 pm · Reply

Shouldn't the flag for "treat warnings as errors" to be -Werror? The -Wfatal-errors flag means to stop the compilation once an error occur. Ref: https://gcc.gnu.org/onlinedocs/gcc-4.8.0/gcc/Warning-Options.html#Warning-Options

nascardriver
September 24, 2018 at 12:16 am · Reply

Yep, the screenshot of codeblocks is also showing the wrong entry.

Alex
September 24, 2018 at 8:32 am · Reply

Text and picture updated. Thanks for pointing out the error!

Ryan
September 23, 2018 at 12:25 am · Reply

Hello! I may have found a typo, and wanted to point it out in case it was a mistake. When you said "you can request your compiler be more assertive about providing warnings about for things it finds strange", I think you might have accidentally put 'about' a second time.

Alex
September 24, 2018 at 7:53 am · Reply

No may about it, definitely a typo. Again, thanks for pointing all of these out.

Ryan
September 24, 2018 at 12:37 pm · Reply

My pleasure! I'm glad I could help. I worded it in a subjective way to make sure I wasn't possibly just misinterpreting something and falsely pointing it out, especially since I'm not entirely familiar with the field yet. That's what I'm here to change, though!

Ahmed
September 19, 2018 at 3:22 pm · Reply

Any tut for the following:
Add the following flag to your command line: -Wfatal-errors

Alex
September 19, 2018 at 4:24 pm · Reply

No, but basically just do this:

gcc main.cpp -o outputfile -Wfatal-errors

Ahmed
September 19, 2018 at 4:59 pm · Reply

Thank you :)

nascardriver
September 20, 2018 at 12:45 am · Reply

*g++