3.5 — More debugging tactics

BY ALEX ON FEBRUARY 1ST, 2019 | LAST MODIFIED BY ALEX ON JANUARY 22ND, 2020

In the previous lesson (**3.4 -- Basic debugging tactics**), we started exploring how to manually debug problems. In that lesson, we offered some criticisms of using statements to print debug text:

1. Debug statements clutter your code.
2. Debug statements clutter the output of your program.
3. Debug statements require modification of your code to both add and to remove, which can introduce new bugs.
4. Debug statements must be removed after you're done with them, which makes them non-reusable.

We can mitigate some of these issues. In this lesson, we'll explore some basic techniques for doing so.

Conditionalizing your debugging code

Consider the following program that contains some debug statements:

```
#include <iostream>

int getUserInput()
{
std::cerr << "getUserInput() called\n";
    std::cout << "Enter a number: ";
    int x;
    std::cin >> x;
    return x;
}

int main()
{
std::cerr << "main() called\n";
    int x = getUserInput();
    std::cout << "You entered: " << x;

    return 0;
}
```

When you're done with the debugging statement, you'll either need to remove them, or comment them out. Then if you want them again later, you'll have to add them back, or uncomment them.

One way to make it easier to disable and enable debugging throughout your program is to make your debugging statements conditional using preprocessor directives:

```
#include <iostream>

#define ENABLE_DEBUG // comment out to disable debugging

int getUserInput()
{
#ifdef ENABLE_DEBUG
std::cerr << "getUserInput() called\n";
#endif
    std::cout << "Enter a number: ";
    int x;
    std::cin >> x;
    return x;
```

```
}

int main()
{
#ifdef ENABLE_DEBUG
std::cerr << "main() called\n";
#endif
    int x = getUserInput();
    std::cout << "You entered: " << x;

    return 0;
}
```

Now we can enable debugging simply by commenting / uncommenting *#define ENABLE_DEBUG*. This allows us to reuse previously added debug statements and then just disable them when we're done with them, rather than having to actually remove them from the code. If this were a multi-file program, the #define ENABLE_DEBUG would go in a header file that's included into all code files so we can comment / uncomment the #define in a single location and have it propagate to all code files.

This addresses the issue with having to remove debug statements and the risk in doing so, but at the cost of even more code clutter. Another downside of this approach is that if you make a typo (e.g. misspell "DEBUG") or forget to include the header into a code file, some or all of the debugging for that file may not be enabled. So although this is better than the unconditionalized version, there's still room to improve.

Using a logger

An alternative approach to conditionalized debugging via the preprocessor is to send your debugging information to a log file. A **log file** is a file (normally stored on disk) that records events that occur in software. The process of writing information to a log file is called **logging**. Most applications and operating systems write log files that can be used to help diagnose issues that occur.

Log files have a few advantages. Because the information written to a log file is separated from your program's output, you can avoid the clutter caused by mingling your normal output and debug output. Log files can also be easily sent to other people for diagnosis -- so if someone using your software has an issue, you can ask them to send you the log file, and it might help give you a clue where the issue is.

While you can write your own code to create log file and send output to them, you're better off using one of the many existing third-party logging tools available. Which one you use is up to you.

For illustrative purposes, we'll show what outputting to a logger looks like using the **plog** logger. Plog is implemented as a set of header files, so it's easy to include anywhere you need it, and it's lightweight and easy to use.

```
#include <iostream>
#include <plog/Log.h> // Step 1: include the logger header

int getUserInput()
{
    LOGD << "getUserInput() called"; // LOGD is defined by the plog library

    std::cout << "Enter a number: ";
    int x;
    std::cin >> x;
    return x;
}

int main()
{
    plog::init(plog::debug, "Logfile.txt"); // Step 2: initialize the logger
```

```
17
18      LOGD << "main() called"; // Step 3: Output to the log as if you were writing to the consol
19
20      int x = getUserInput();
21      std::cout << "You entered: " << x;
22
23      return 0;
24  }
```

Here's output from the above logger (in the `Logfile.txt` file):

```
2018-12-26 20:03:33.295 DEBUG [4752] [main@14] main() called
2018-12-26 20:03:33.296 DEBUG [4752] [getUserInput@4] getUserInput() called
```

How you include, initialize, and use a logger will vary depending on the specific logger you select.

Note that conditional compilation directives are also not required using this method, as most loggers have a method to reduce/eliminate writing output to the log. This makes the code a lot easier to read, as the conditional compilation lines add a lot of clutter. With plog, logging can be temporarily disabled by changing the init statement to the following:

```
1       plog::init(plog::none , "Logfile.txt"); // plog::none eliminates writing of most messages,
```

We won't use plog in any future lessons, so you don't need to worry about learning it.

As an aside...

If you want to compile the above example yourself, or use plog in your own projects, you can follow these instructions to install it:

First, get the latest plog release:

- Visit the **plog** repo.
- Click the "releases" tab (it's in the same row as "commits", "branches", "packages", etc...)
- Under the release tagged as "latest release" on the left, click the link "Source code (zip)" to download the latest release.

Next, unzip the entire archive to `<anywhere>` on your hard drive.

Finally, for each project, set the `<anywhere>\plog-<version>\include\` directory as an `include directory` inside your IDE. There are instructions on how to do this for Visual Studio here: **A.2 -- Using libraries with Visual Studio** and Code::Blocks here: **A.3 -- Using libraries with Code::Blocks**.

 **3.6 -- Using an integrated debugger: Stepping**

 **Index**

 **3.4 -- Basic debugging tactics**

C++ TUTORIAL | PRINT THIS POST

30 comments to 3.5 — More debugging tactics

Michael
January 29, 2020 at 12:33 am · Reply

I downloaded plog 1.1.4, used Visual Studio 2019 on Windows 10 to try to compile my main file, and got some obscure errors about "this", "__if_exists", and "__if_not_exists" which I traced back to Log.h:

```
#if defined(_MSC_VER) && _MSC_VER >= 1600 && !defined(__INTELLISENSE__) && !defined(__INTEL_
#   define PLOG_GET_THIS()      __if_exists(this) { this } __if_not_exists(this) { 0 }
#else
#   define PLOG_GET_THIS()      0
#endif
```

I was unable to find anything online confirming this, but it seems to me that my compiler doesn't have support for __if_exists. I edited this part of the header file to leave only `#define PLOG_GET_THIS() 0` which fixed the compile issue for me.

BTW Alex you should probably mention which directory the log file shows up in :)

Raton
January 18, 2020 at 5:57 am · Reply

Hello. I tried to use the Plog logger, but I ran into a problem. Here's what I did:

1) Download the .zip file
2) Copy the "plog" file from that .zip file to my project file (the one with my code files)
3) #include "plog/Log.h" in one of my code files

But then the compiler tells me that it can't find "plog.Logger.h". I figured out that it was actually because of the use of angled brackets instead of quotes in Log.h, so I did the change myself. But then the compiler tells me that it can't find "plog/Appenders/IAppender.h", which is included by Logger.h. Again, it's because of the angled brackets in Logger.h. And the same type of errors kept happening. But does it mean I have to replace the angled brackets with quotes in every Plog header? And why did this problem occur in the first place?

Alex
January 22, 2020 at 9:22 pm · Reply

I updated the instructions in the article to show how to install plog. Please give them a whirl and let me know if this resolves your concern. You shouldn't need to modify anything in the plog library itself.

Raton
January 24, 2020 at 9:16 am · Reply

Hello, thanks for your help! It works now :)

the swag man
January 16, 2020 at 4:17 am · Reply

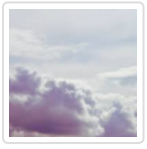
when I try to compile

```
1  int getUserInput()
2  {
3      LOGD << "getUserInput() called"; // LOGD is defined by the plog library
4
5      std::cout << "Enter a number: ";
6      int x;
7      std::cin >> x;
8      return x;
9  }
10
11 int main()
12 {
13     plog::init(plog::debug, "Logfile.txt"); // Step 2: initialize the logger
14
15     LOGD << "main() called"; // Step 3: Output to the log as if you were writing to the con
16
17     int x = getUserInput();
18     std::cout << "You entered: " << x;
19
20     return 0;
21 }
```

llvm and gcc throw

```
1  error: expected expression
```

at line 8 and 20... help me please

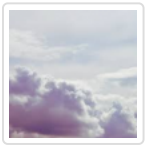


Chayim
December 25, 2019 at 11:03 pm · Reply

In subject:
"Conditionalizing your debugging code"
In the header example "ENABLE-DEBUG" how is it able to use it as "ifdef" because "ENABLE-DEBUG" has no definition?

nascardriver
December 27, 2019 at 7:40 am · Reply

`#ifdef` checks if a macro is defined, not if it has a value.



Chayim
December 25, 2019 at 3:21 am · Reply

In the header example, how is it able to use it as "ifdef" because "ENABLE-DEBUG" has no definition?

Anderson
November 28, 2019 at 10:44 am · Reply

Hey there, quick question.
In the code snippet below, for example:

```
1 #include <iostream>
2 #include "plog/Log.h" // Step 1: include the logger header
```

you can clearly notice the difference between line 1 and 2. I would like to know why the iostream library (or file or ...) is surrounded by ankle brackets while the plog/Log.h header file is surrounded by double quotes in this case since it isn't really user defined.

Is it because the *iostream* library is in the c++ standard library while *plog/Log.h* is a third party library ?

Also, if it isn't too much to ask, does anyone know where I can see the official standard c++ libraries ?

nascardriver
November 29, 2019 at 2:31 am · Reply

"" is used for files in your project director, <> for system files or files that you added to the include path (Both "" and <> are implementation-defined, but that's what they usually do). "plog/Log.h" is in the project directory, so "" is used.

> does anyone know where I can see the official standard c++ libraries ?
https://en.cppreference.com/w/cpp/header

Anderson
November 29, 2019 at 3:02 am · Reply

Thanks, I get it now.

sito
November 5, 2019 at 12:51 pm · Reply

hello! so I'm still learning c++ and I'm using an API which I can't get to work so I need a log file. When I try to include the log.h file from plog I get this error: error C1083: Cannot open include file: 'plog/Logger.h': No such file or directory.
I'm using visual studio 2019 and I've tried going to properties, c++, general and adding the directory to aditional include directorys. I then did the same but for vc++ directorys so that if the header file is not found in the project directory it would go on and search in the directory where plog is located. Despite me doing this and using #include "Log.h" the error won't go away. Any suggestions on what i can do to fix this?

Kyrollos
October 15, 2019 at 10:58 am · Reply

I got an error in "Util.h" when I tried to compile my project:
plog\Util.h|164|error: '_vscwprintf' was not declared in this scope|

Austin G
August 30, 2019 at 7:09 pm · Reply

Output from the logger mentions "main@22" and "getUserInput@10" while those code lines were originally line 18 and line 6 respectively in the code file. Is this anything relevant or just a peculiarity of this particular logger?

Alex
September 1, 2019 at 2:33 pm · Reply

Just a error on my part. I removed an extraneous function from the example which shifted the line numbers, but I forget to update the logger output. Thanks for pointing this out!

Darshan K
August 30, 2019 at 6:23 am · Reply

Getting an error before compiling in Visual Studio 2019 (I wrote my own program quite similar to this) saying: "namespace "plog" has no member "init"". The program runs properly however.

My code:

```
#include <iostream>
#include "plog/Log.h"
int getUserInput()
{
    LOGD << "getUserInput() called";
    std::cout << "Enter an integer: ";
    int x{};
    std::cin >> x;
    return x;
}
int main()
{
    plog::init(plog::debug, "Logfile.txt");
    LOGD << "main() called";
    int x = getUserInput();
    std::cout << "You entered: " << x;
    return 0;
}
```

Another issue is that there is no newline between lines in the log file.

Code does not compile when I copy-paste the program given in the tutorial.

How to fix these issues?

Darshan K
August 31, 2019 at 9:07 am · Reply

For some reason it works now while it didn't work then.

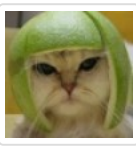
But still, every line gets into the same line in the log file.

nascardriver
August 31, 2019 at 11:28 pm · Reply

You're not printing a line feed in line 14.

Manash Das
August 1, 2019 at 3:58 am · Reply

Hi Alex,
As you have mentioned, in a multi-file program #define ENABLE_DEBUG can go into a header file. Can you please write code for that header file?



Alex
August 3, 2019 at 9:35 pm · Reply

someheader.h:

```
1 #ifndef SOMEHEADER_H
2 #define SOMEHEADER_H
3
4 #define ENABLE_DEBUG
5
6 #endif
```

Manash Das
August 3, 2019 at 9:51 pm · Reply

Thank you very much for help

Yoel
April 13, 2019 at 10:48 pm · Reply

In visual studio 2019 you need to include the "plog" directory by Project-> Properties -> C\C++ -> General -> Additional Include directories -> <path to "plog" not including itself>.

This is because the code references the "plog" directory

```
1 #include <plog/Log.h> // Directory structure must be maintained
```

Locke
December 11, 2019 at 12:58 am · Reply

Thanks Yoel! This explanation finally helped me include plog!

Red Lightning
April 11, 2019 at 8:02 am · Reply

"Log files can also easily sent to other people for diagnosis"
Syntax error: missing "be".

jaffaman
March 29, 2019 at 4:49 pm · Reply

Got this working in Atom no worries.

I am running MinGW to compile.

Steps:
1. Download zip from https://github.com/SergiusTheBest/plog
2. Extract files and copy plog folder (~\plog-master\include) to folder (~MinGW\include).
3. Follow instructions in readme from github.

Ram Rasia
March 8, 2019 at 4:31 pm · Reply

plog library didn't work in MacOSx.

In file included from CPlusPlusTutorial.cpp:3:
./plog/Log.h:7:10: fatal error: 'plog/Logger.h' file not found
#include <plog/Logger.h>
^~~~~~~~~~~~~~~
1 error generated.

Angle brackets cause a problem. I think so it should be double quotes.

Alex
March 8, 2019 at 4:52 pm · Reply

I agree. You should report this to the library maintainer. :)

Dong
March 5, 2019 at 1:04 am · Reply

In the "Plog strategy" example, I type this code and run it in Visual Studio 2013 and DevC++. Both of them makes error about this code.
- In Dev C++, [Error] plog/Log.h: No such file or directory.
- In Visual Studio, first error is the same. The second error is: name followed by '::' must be a class or namespace name. It the same error in Dev C++. I donot know where "plog/Log.h". Thanks for your help.

Alex
March 8, 2019 at 12:45 pm · Reply

You need to get the Log.h file and place it appropriately on your system. I added some instruction to the lesson about how to do this.

WSLaFleur
February 19, 2019 at 3:56 am · Reply

Since there are no comments on this lesson, I just wanted to take a second to say thanks for making these. This is a decent example, since you haven't really touched on utilizing third-party resources much yet.