clang-cl
What it is, how it works, and how to use it

Hans Wennborg
Euro-LLVM 2014
Why give a talk about a compiler driver?

Chapters about drivers = 0
What is the driver good for?

I take the arguments from the user

And pass them to cc1
Do we even need the driver?

Couldn't the user just
Pass it to cc1's stdin directly?
What did the driver ever do for us?

- The driver allows us to build real programs
- It is a great compatibility layer
- Chrome Linux/Mac clang build very similar to gcc
- Now trying to do the same on Windows.
This is cl.exe

D:\>cl a.cc
Microsoft (R) C/C++ Optimizing Compiler Version 18.00.21005.1 for x64
Copyright (C) Microsoft Corporation. All rights reserved.

a.cc
Microsoft (R) Incremental Linker Version 12.00.21005.1
Copyright (C) Microsoft Corporation. All rights reserved.

/out:a.exe
a.obj
D:\>.
This is clang-cl.exe

D:\>clang-cl a.cc
D:\>
This is clang-cl.exe
D:\>clang-cl b.cc
D:\>b.cc(4,21): warning: comparison of address of 'y' equal to a null pointer is always false [-Wautological-pointer-compare]
    return x == 1 && y == 0;
1 warning generated.
D:\>
clang-cl in Visual Studio
clang-cl in Visual Studio
clang-cl in Visual Studio

```cpp
CustomText::CustomText() :
    hwnd_(NULL),
    wszText_(NULL),
    cTextLength_(0),
    pD2DFactory_(NULL),
    prt_(NULL),
    pBlockBrush_(NULL),
    pBitmapBrush_(NULL),
    pDWriteFactory_(NULL),
    pTextFormat_(NULL),
    pTextLayout_(NULL),
    pTextWriter_(NULL),
    pWICFactory_(NULL)
{
}
```

Output:

```
1> clang-cl.exe : warning : argument unused during compilation: '/EHsc'
1> clang-cl.exe : warning : argument unused during compilation: '/GS'
1> clang-cl.exe : warning : argument unused during compilation: '/fp:precise'
1> clang-cl.exe : warning : argument unused during compilation: '/FdDebug_Win32\vc120.pdb'
1> clang-cl.exe : warning : argument unused during compilation: '/Gd'
1>CustomeText.cpp(38,5): warning : field 'cTextLength_' will be initialized after field 'pD2DFactory_' [-wreorder]
1> cTextLength_(0),
1> ^
1> 1 warning generated.
```
clang-cl in Visual Studio
How does clang-cl work?

```
clang-cl.exe == clang.exe --driver-mode=cl
```
How does clang-cl work?
How does clang-cl work?
How does clang-cl work?

def _SLASH_C : CLFlag<"C">, HelpText<"Don't discard comments when preprocessing">,
    Alias<C>;
def _SLASH_c : CLFlag<"c">, HelpText<"Compile only">, Alias<c>;
def _SLASH_D : CLJoinedOrSeparate<"D">, HelpText<"Define macro">,
    MetaVarName<"<macro[=value]>">, Alias<D>;
def _SLASH_E : CLFlag<"E">, HelpText<"Preprocess to stdout">, Alias<E>;
def _SLASH_GR : CLFlag<"GR">, HelpText<"Enable RTTI">, Alias<frtti>;
def _SLASH_GR_ : CLFlag<"GR-">, HelpText<"Disable RTTI">, Alias<fno_rtti>;
def _SLASH_GF_ : CLFlag<"GF-">, HelpText<"Disable string pooling">,
    Alias<fwritable_strings>;
def _SLASH_Gy : CLFlag<"Gy">, HelpText<"Put each function in its own section">,
    Alias<ffunction_sections>;
def _SLASH_Gy_ : CLFlag<"Gy-">, HelpText<"Don't put each function in its own section">,
    Alias<fno_function_sections>;}
How does clang-cl work?
How does clang-cl work?

D:\>clang-cl /c /GR c.cc
error: cannot mangle RTTI descriptors for type 'C' yet
error: cannot mangle the name of type 'C' into RTTI descriptors yet
error: cannot mangle RTTI descriptors for type 'D' yet
error: cannot mangle the name of type 'D' into RTTI descriptors yet
error: cannot mangle RTTI descriptors for type 'C' yet
error: cannot mangle the name of type 'C' into RTTI descriptors yet
6 errors generated.

D:\>
How does clang-cl work?

D:\>clang-cl /c /GR /fallback c.cc
error\clang\: cannot mangle RTTI descriptors for type 'C' yet
error\clang\: cannot mangle the name of type 'C' into RTTI descriptors yet
error\clang\: cannot mangle RTTI descriptors for type 'D' yet
error\clang\: cannot mangle the name of type 'D' into RTTI descriptors yet
error\clang\: cannot mangle RTTI descriptors for type 'C' yet
error\clang\: cannot mangle the name of type 'C' into RTTI descriptors yet
6 errors generated.
clang-cl.exe: warning: falling back to C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\BIN\c1.exe
C.cc

D:\>
Chromium's content_shell built with clang-cl

The LLVM Compiler Infrastructure

Overview

The LLVM Project is a collection of modular and reusable compiler and toolchain technologies. Despite its name, LLVM has little to do with traditional virtual machines, though it does provide helpful libraries that can be used to build them. The name "LLVM" itself is not an acronym; it is the full name of the project.

LLVM began as a research project at the University of Illinois, with the goal of providing a modern, SSA-based compilation strategy capable of supporting both static and dynamic compilation of arbitrary programming languages. Since then, LLVM has grown to be an umbrella project consisting of a number of subprojects, many of which are being used in production by a wide variety of commercial and open source projects as well as being widely used in academic research. Code in the LLVM project is licensed under the "UIUC" BSD-Style license.

The primary sub-projects of LLVM are:

1. The LLVM Core libraries provide a modern source-and target-independent optimizer, along with code generation support for many popular CPUs (as well as some less common ones!). These libraries are built...
The driver provides convenience and compatibility
clang-cl is a cl.exe compatible driver mode for clang
It understands the environment, the flags, and the tools
Integrates with Visual Studio
/fallback allows bring-up of large projects.