CPP/C++ Compiler Flags and Options

Table of Contents

1. Compiler flags and invocation
   - 1.1. GCC, GCC/Mingw and Clang++
     - 1.1.1. General Compiler and Linker Flags
     - 1.1.2. Peformance Optimization Flags
     - 1.1.3. Useful compiler switches for reducing binary size
     - 1.1.4. Undefined Behavior Sanitizers UBSAN
     - 1.1.5. Compiler invocation examples
     - 1.1.6. Example - Build executable with unified compilation
     - 1.1.7. Compile source with static Linking
     - 1.1.8. Release / Debug building
   - 1.2. MSVC (VC++ or Visual C++) Compiler Options

1 Compiler flags and invocation

1.1 GCC, GCC/Mingw and Clang++

1.1.1 General Compiler and Linker Flags

Search Path and Library Linking Flags

- -l[linalg]
  - => Links to shared library or shared object - Specifically, it links to linalg.dll on Windows, liblinalg.so (on Unix-like oses like Linux, BSD, AIX, …) or linalg.dylib on MacOSX.
- -L[/path/to/shared-libraries]
  - => Add search path to shared libraries, directory containing *.so, *.dll or *.dlyb files such as libLinearAlgebra.so depending on the current operating system.
- -I[/path/to/header-files]
  - Add search path to header files (.h) or (.hpp).
- -D[FLAG] or -D[FLAG]=VALUE
  - Pass preprocessor flag #if FLAG …

GCC and Clang Most common compiler flags:

- std - Specify the C++ version or ISO standard version.
  - -std=c++11 (ISO C++11)
  - -std=c++14 (ISO C++14)
  - -std=c++1z (ISO C++17)
  - -std=c++20 (C++20 experimental)
  - -std=gnu++ (ISO C++ with GNU extensions)
- Verbosity - [W stands for warning]
  - **-Wall**
    - Turns on lots of compiler warning flags, specifically (-Waddress, -Wcomment, -Wformat, -Wbool-compare, -Wuninitialized, -Wunknown-pragmas, -Wunused-value, -Wunused-value …)

    - **-Werror**
        - Turn any warning into a **compilation error**.
    - -Wextra or just -W (**see more**)
        - Enables extra flags not enabled by -Wall, such as -Wsign-compare (C only), -Wtype-limits, -Wuninitialized …
    - -pendantic or -Wpendantic
        - Issue all warning required by ISO C and ISO C++ standard, it issues warning whenever there are compiler extensions non compliant to ISO C or C++ standard.
    - -Wconversion
    - -Wcast-align
    - -Wunnused
    - -Wshadow
    - -Wold-style-cast
    - -Wpointer-arith -Wcast-qual -Wmissing-prototypes -Wno-missing-braces
- Output file: -o <outputfile>
    - g++ file.cpp -o file.bin
- Common library flags
    - -lm - Compiles against the shared library libm (basic math library, mostly C only)
    - -lpthread - Compile against Posix threads shared library
- Include Path - Directories containing headers files.
    - -I/path/to/include1 -I/path/to/include2 …
- Compilation flags **-D<flag name>**
    - -DCOMPILE_VAR -> Enable flag COMPILE_VAR - It is equivalent to add to the code (#define COMPILE_VAR)
    - -DDO_SOMETHING=1 - Equivalent to add to the code **#define DO_SOMETHING = 1**
    - -DDISABLE_DEPRECATED_FUNCTIONS=0
- **Optmization** - **docs**
    - -O0
        - No optmization, faster compilation time, better for debugging builds.
    - -O2
    - -O3
        - Higher level of optmization. Slower compile-time, better for production builds.
    - -OFast
        - Enables higher level of optmization than (-O3). It enables lots of flags as can be seen **src** (-ffloat-store, -ffsast-math, -ffinite-math-only, -O3 …)
    - -finline-functions
    - -m64
    - -funroll-loops
    - -fvectorize
    - -fprofile-generate
- Misc
    - -fexceptions -fstack-protector-strong –param=ssp-buffer-size=4
- **Special Options**
    - **-g**
        - => Builds executable with debugging symbols for GDB GNU Debugger or LLDB Clang/LLVM Debugger. It should only be used during development for debugging builds.
    - -c
        - => Compiler source(s) to object-code (input to linker). This option is better for incremental compilation when using multiple files.
    - **-pie**
        - => Builds a dynamically linked position independent executable.
    - **-static-pie**
        - => Builds a staticlaly linked position independent executable.
    - **-shared**

        - => Build a shared library (.so or .dylib on U*nix-like Oses) or .dll on MS-Windows.
    - **-fno-exceptions**
        - => Disable C++ exceptions (it may be better for embedded systems or anything where exceptiions may not be acceptable).
    - **-fno-rtti**
        - => Disable RTTI (Runtime Type Information) - There many texts around where game and embedded systems developers report that they disable RTTI due to performance concerns.
    - **-fvisibility=hidden**
        - => Make library symbols hidden by default, in a similar way to what happens in Windows DLLs where exported symbols must have the prefix __declspec(dllexport) or __declspec(dllimport). When all symbols are exported by default, it may increase the likelyhood of undefined behavior if there a multiple definitions of same symbol during linking. See more at:
        - **Simple C++ Symbol Visibility Demo | LabJack**
        -
    - **https://wiki.debian.org/Hardening**
        - => Lots of compiler flags for hardening security.

Linker Flags

- Verbose
    - -Wl,–verbose
    - -Wl,–print-memory-usage
- Directory which linker will search for libraries (*.so files on Unix; *.dylib on MacOSX and *.dll on Windows)
    - -L/path/to-directory
- Strip Debug Information
    - -Wl,-s
- Build Shared Library
    - -shared
- Set Unix Shared Library SONAME **(See)**
    - –Wl,soname,<NAME>
    - –Wl,soname,libraryname.so.1
- General Format of Linker Options
    - -Wl,–<OPTION>=<VALUE>
- Windows-only (MINGW)
    - -Wl,–subsystem,console => Build for console subsystem.
    - -Wl,–subsystem,windows => Build for winodw subsystem.
    - -ld3d9 => Link against DirectX (d3d9.dll)
- Windows-only DLL (Dynamic Linked Libraries)
    - -shared
    - -Wl,–export-all-symbols
    - -Wl,–enable-auto-import

- Static link against LibGCC runtime library:
    - -static-libgcc
- Static link against libstdC++ runtime library:
    - -static-libstdc++
- Static link against Gfortran (GNU fortran) runtime library.
    - -static-libgfortran
- Set Unix RPATH (Note Windows does not have rpath)
    - –Wl,rpath=/path/to/directory1;/path/to/directory2
- Set Unix RPATH to executable current directory.
    - –Wl,rpath=$ORIGIN
- Change the default dynamic linker (UNIX and Linux)
    - -Wl,-dynamic-linker,/path/to/linker/ld-linux.so.2.1

- Common Unix Dependencies:
  - -lpthread => Link against POSIX threads library.
  - -ldl => Link against libdl library for dlopen(), dlclose(), APIs.
- Exclude Runtime Libraries **(gcc docs)**
  - -nostartfiles => "Do not use the standard system startup files when linking. The standard system libraries are used normally, unless -nostdlib, -nolibc, or -nodefaultlibs is used."
  - -nodefaultlibs => Do not use the standard system libraries when linking.
  - -nolibc => Do not use the C library or system libraries tightly coupled with it when linking.
  - -nostdlib => Do not use the standard system startup files or libraries when linking.
- Set heap size
  - -Wl,–heap,201561
- Stack reserve size
  - -Wl,–stack,419525
- Generate Linker Map (mostly used for embedded systems)
  - -Wl,-Map=linker-map.map
- Use a custom linker script (embedded systems)
  - -Wl,T/path/to/linker-script.ld
- Linker version script
  - -Wl,–version-script,criptfile
- Eliminate dead-code or unused code for decreasing the program or firmware size (embedded systesm) - **(Elinux)**
  - -ffunction-sections -fdata-sections -Wl,–gc-sections
- Miscellaneous
  - -Wl,–allow-multiple-definition
  - -fno-keep-inline-dllexport
  - -Wl,–lager-address-aware
  - -Wl,–image-base,358612

Files Generated by the Compiler

- Object Files
  - *.o -> Generated on *NIX - Linux, MacOSX … by GCC or Clang
  - *.obj -> Windows
- Binary Native Executable - Object Code
  - *NIX: Linux, MacOSX, FreeBSD -> Without extension.
  - Windows: *.exe
  - *.hex -> Extension of many compiled firmwares generated by embedded systems compilers such as proprietary compilers for Microcontrollers.
- Shared Objects - Shared Libraries
  - *.dll -> Called dynamic linked libraries on Windows -> libplot.dll
  - *.so -> Called shared Object on Linux -> libplot.so
  - *.dylib -> Extension used on MacOSX.
- Static Library
  - *.a - extension

Review See:

- **Things to remember when compiling and linking C/C++ programs · GitHub**
- **Linker Options**
- **GNU Linker Command Language** => GNU Linker Script Command Language, widely used for embedded systems.
- **Man7.org - Shared Libraries**
- **https://gcc.gnu.org/onlinedocs/gcc/Link-Options.html**
- **https://interrupt.memfault.com/blog/get-the-most-out-of-the-linker-map-file**
- **https://tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html**
- **Link time dead code and data elimination using GNU toolchain**

- https://www.avrfreaks.net/forum/keep-and-wl-gc-sections
- Removing Unused Functions and Dead Code (Washigton University)
- The why and how of RPATHs - Flameeyes
- Creating relocatable Linux executables by setting RPATH with origin
- Compiling NTL for Android
- Building and Using DLLs Prev Chapter 4. Programming with Cygwin (MINGW - Windows)

1.1.2 Peformance Optimization Flags

Levels of Optimization

- (-O0) - Optimization Level 0 (No optimization, default)
  - Better for debugging builds during development, since it provides faster compile-time.
- (-O1) - Optimization Level 1
- (-O2) - Optimization Level 2
  - Enables (-O1); inline small functions;
- (-O3) - Optimization Level 3 (Most aggressive optimization, Highest level of optmization and speed)
  - Problem: Slower compile-time and large binary size.
  - More function inlining; loop vectorization and SIMD instructions.
- (-OS) - Code Size Optimization
  - Enable (-O2), but disable some optimizations flags in order to reduce object-code size.
- (-Oz) [CLANG-ONLY] - Optimizes for size even further than (-OS)
- (-Ofast) - Activate (-O3) optimization disregarding strict standard compliance.
- (-Og) - Optmizing for debugging.
  - Enables all optimization that does not conflicts with debugging. It can be used with the (-g) flag for enabling debugging symbols.

Other Optimization Flags

Linking:

- Link Time Optimization
  - (-flto)

Loops and Vectorization:

- Unroll loops
  - -funroll-loop
- Auto Vectorization
  - (-O3) or (-ftree-vectorize)
- Verbose Auto Vectorization
  - (-ftree-vectorizer-verbose)

Misc:

- Profiling Information for further processing and measuring performance with **gprof** program:
  - (-pg)
- Tune to Pentium 4 everything about the produced code.
  - -mcpu=pentium4
- Generate instructions for Pentium 4
  - -march=pentium4
- Attempt to Inline All Functions, even if they are not annotated with inlining. Note: inlining trades speed for increasing of code size.
  - -finline-functions
- Devirtualize - attempt to convert vritual function calls to direct calls. Enabled by: (-O2), (-O3) and (-Os).

- -fdevirtualize

- Hide Global Sysmbols by default in shared libraries
  - Note: it equires the exposed symbols to be annotated with GGC extension `__attribute__(...)`. Benefits: decreases startup time; quickier function calls; smaller memory footprints.
  - (-fvisibility=hidden)
  - -Bsymbolic

Further Reading

- **Optimizing Floating Point Calculations II** - Michael A. Saum.
  - http://www.math.utk.edu/~msaum/papers/FPOPT2.pdf
- **Faster C++** - Cambridge University
  - http://www-h.eng.cam.ac.uk/help/tpl/languages/C++/fasterC++.html
- **Using the GNU Compiler Collection (GCC): Optimize Options**
  - https://doc.ecoscentric.com/gnutools/doc/gcc/Optimize-Options.html
  - Shows many GCC optimization options.
- **A story about –fast-math -fbroken-math**
  - https://web.archive.org/web/20140328101226/http://www.pointclouds.org/news/2011/08/29/ffast-math/
  - Reports bugs that the optimization –fast-math may cause.
- **Redis Benchmarks with Optimizations**
  - https://matt.sh/redis-benchmark-compilers
  - Explains the optimization flag (-flto)
- **Cache-Friendly Profile Guided Optimization** - Baptiste Wicht
  - https://pdfs.semanticscholar.org/58df/0757996f7d592d28657c7599379dfa89095b.pdf
- **Performance Optimization Getting your programs to run faster CS 691.**
  - https://slideplayer.com/slide/7987999/
- **Pragmatic Optimization in Modern Programming - Mastering Compiler Optimizations** - Marina Kolpakova
  - https://www.slideshare.net/MarinaKolpakova/pragmatic-optimization-in-modern-programming-mastering-compiler-optimizations

1.1.3 Useful compiler switches for reducing binary size

Note: Those switches/flags are also useful for embedded systems.

Compiler switch	Description
-flto	Link time optimization
-ffunction-sections	Eliminate unnused code
-fdata-sections	Eliminate unnused ELF symbols

1.1.4 Undefined Behavior Sanitizers UBSAN

- The Undefined Behavior Sanitizer - UBSAN — The Linux Kernel documentation
- UndefinedBehaviorSanitizer — Clang 9 documentation
- GCC Undefined Behavior Sanitizer - ubsan - RHD Blog
- AddressSanitizer Being Ported To GCC Trunk - Phoronix
- Using AddressSanitizer & ThreadSanitizer In GCC 4.8 - Phoronix
- Marek Polacek - RFC Implement Undefined Behavior Sanitizer

1.1.5 Compiler invocation examples

1.1.6 Example - Build executable with unified compilation

Compile file1.cpp, file.cpp, file2.cpp into the executable app.bin

- Option 1: Compile and link once in a single command. The disadvantage of this way is the slower compile time rather than separate compilation and linking.

```
# CC=gcc
CC=clang++

$ clang++ file1.cpp file2.cpp file3.cpp \
  -std=c++14  -o app.bin -O3 -g \
  -Wall -Wextra -pendantic \
  -lpthread -lblas -lboost_system -lboost_filesystem \
  -I./include/path1/with/headers1 -I./include2 -L./path/lib1 -L./pathLib2
```

Explanation:

- -std=c++14 -> Set the C++ version. This flag can be C++11, C++14, C++17, C++20 …
- -o app.bin -> Set the output native executable file ot app.bin
- -I./include/path1/with/headers
  - Directory with header files
- -g Produce executable with debug symbols
- -Wall -Wextra -Wshadow
  - Warning flags - enable more verbosity which helps to catch bugs earlier.
- -O3 - Use optmization of level 3 - the disadvantage of using optmization is the slower compile time. So this flag should only be enabled on production builds.
- -lpthread -lblas -lboost_system -lboost_filesystem
  - Link against shared libraries (extensions: *.so - Unix, *.dylib or *.dll on Windows) pthread, blas, boost_system …

1.1.7 Compile source with static Linking

```
$ gcc -static example.o -lgsl -lgslcblas -lm
```

1.1.8 Release / Debug building

Compiler:

- GCC and CLANG
  - Debug: No optmized, but faster building time.
    - -O0 -g
  - Release:
    - -O3 -s -DNDEBUG [-march=native] [-mtune=native]
    - -O2 -s -DNDEBGU
- MSVC:
  - Debug:
    - /MDd /Zi /Ob0 /Od /RTC1
  - Release:
    - /MD /O1 /Ob1 /DNDEBUG

Note:

- For GCC and Clang
  - -O0 means no optmization
  - -g - adds debugging symbols to executable.
  - -DNDEBUG - disable assertions

References:

- **c++ - How to build in release mode with optimizations in GCC? - Stack Overflow**
-

1.2 MSVC (VC++ or Visual C++) Compiler Options

MSVC Native tools:

- CC = cl.exe
  - C and C++ Compiler - Can compile both C and C++ code depending on the flag. By default it compiles C++ code.
- rc.exe => Resource Compiler.
- LD = link.exe
  - C++ Linker.
- AS = ml
  - Assembler
- AR = lib
  - Archiver

Compiler: cl.exe

- /nologo - Suppress microsoft's logo
- /out:<file.exe> - Set output file name.
- /EHsc
- /Zi - Add debugging symbols to the executable
- /c - Doesn't link generating *.exe or *.dll, it creates only intermedidate object code for further separate linking. It is useful for compiling large code bases where each compilation unit can be compiled separately.
- /W4 - Set the level of warning to the highest.
- /entry:<entrypoint> - Set the C/C++ runtime, it can be:
  - mainCRTStartup => calls main(), the entrypoint for console mode apps
  - wmainCRTStartup => calls wmain(), as above but the Unicode version
  - WinMainCRTStartup => calls WinMain(), the entrypoint for native Windows apps
  - wWinMainCRTStartup => calls wWinMain(), as above but the Unicode version
  - _DllMainCRTStartup => Calls DLLMain()
- /subsystem:<type> - Set the subsystem - default Console, it can be:
  - **/subsystem:console** - For applications that necessarly runs in the console (aka terminal emulator)
  - **/subsystem:windows** - Doesn't display the cmd.exe terminal when ones click at the application executable.
- /TC -> Specify that file name is C source code
- /TP -> Specify that file name is a C++ source code (default)
- /MD, /MT, /LD => Run-time library - Indcates that object-code (compiled program) is a DLL.
- /GF -> (Eliminate Duplicate Strings) - Enables the compiler to create a single copy of identical strings in the program image and in memory during execution. This is an optimization called string pooling that can create smaller programs.

CRT C Runtime Options

File:	Linking type	Build Type	Compiler Flag
libcmt.lib	Static	Debug	/MT
libcmtd.lib	Static	Release	/MTd
msvcrt.lib	Dynamic (DLL)	Release	/MD
msvcrtd.lib	Dynamic (DLL)	Debug	/MDd

- Static Linking: The library is appended to the executable, only a single file, the executable needs to be deployed.
- Dynamic Linking: The library is DLL shared library and not appended to the executable. The executable needs to be deployed with the library.

See:

- https://github.com/MicrosoftDocs/cpp-docs/blob/master/docs/build/building-on-the-command-line.md

Useful preprocessor:

- Set subsystem

```
#pragma comment(linker, "/SUBSYSTEM:WINDOWS")
#pragma comment(linker, "/SUBSYSTEM:CONSOLE")
```

- Set linker library to be linked. This pragma is particularly useful in graphical applications.

```
#pragma comment(lib, "user32.lib")
```

Examples:

- Compile multiple files generating an executable named out.exe. Note the default subsystems is the console (**/subsystem:console**) and the default entry point is (mainCRTStartup).
  - /Isrc/includes => Directories containing header files
  - /ld or /linker => Linker flags
  - && out.exe => If the compilation is successful runs the generated executable out.exe

```
$ cl.exe source1.cpp source2.cpp /Fe:out.exe /Isrc/includes /ld gdi.lib user32.lib && out.exe
```

- Compile multiple files for windows subsystem (GUI app) and with wmainCRtstartup.

```
$ cl.exe source1.cpp source2.cpp /Fe:out.exe /Isrc/includes /entry:wmainCRtstartup /subsystem:windows /ld user32.lib && out.exe
```

References:

- Command Line Compilation
- Makefiles and Visual Studio | Cognitive Waves
- C++ Windows Makefile
- MS C/C++: The Command-Line Tools

See also:

- How To Use the C Run-Time
-
-

Created: 2021-06-04 Fri 15:07

Validate