

CSEN 383 – Assignment 1

Winter 2024

Chaitra Boggaram (W1651213)

Working on MacOS

1. Be able to have reliable Linux environment

Running on an M2 MacBook

```
ipadhu@Padhus-MBP Downloads % sw_vers
ProductName:          macOS
ProductVersion:      14.2
BuildVersion:        23C64
ipadhu@Padhus-MBP Downloads %
```

2. Compile and run the C program (`forktest.c`) on a Linux system and to provide a screenshot of the compilation and execution.

```
gcc forktest.c -o forktest
```

```
forktest
```

```
ipadhu@Padhus-MBP Downloads % gcc forktest.c -o forktest
ipadhu@Padhus-MBP Downloads % ./forktest
Parent: Process started
Parent: Forking a child.
Parent: Wait for child to complete.
Child: Process started.
Child: Start 10 second idle: 10 9 8 7 6 5 4 3 2 1 0 done!
Child: Terminating.
Parent: Terminating.
ipadhu@Padhus-MBP Downloads %
```

3. Read the gcc man page

I am running `man clang` because of the below explanations and `gcc -v` shows the clang version as provided in the below image.

```
gcc -v
```

```
ipadhu@Padhus-MBP Downloads % gcc -v
Apple clang version 15.0.0 (clang-1500.0.40.1)
Target: arm64-apple-darwin23.2.0
Thread model: posix
InstalledDir: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin
ipadhu@Padhus-MBP Downloads %
```

Apple uses Clang instead of GCC for a few reasons:

Speed and memory: Clang is faster and uses less memory than GCC.

Diagnostics: Clang provides clearer and more concise error and warning messages than GCC.

Language compliance: Clang has better language compliance than GCC, which improves compatibility with newer standards and languages.

Integration: GCC's large code doesn't integrate well with Apple's IDE.

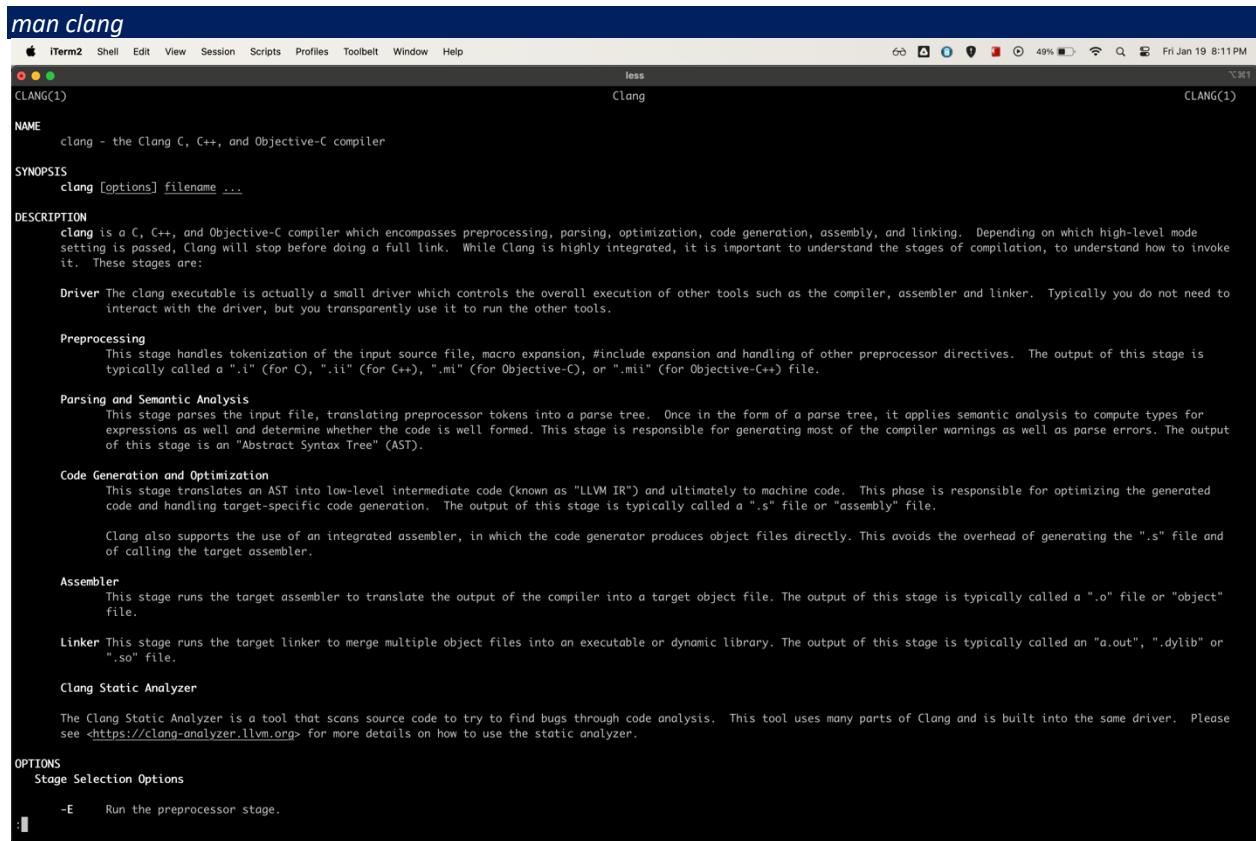
Optimization and reliability: Clang is better than GCC for optimization and reliability because of LLVM.

Clang can replace GCC in many cases without affecting the toolchain. It supports most of the commonly used GCC options. However, for some languages, such as Ada, LLVM is still dependent on GCC or another compiler front end.

Apple doesn't ship a compiler with macOS or iOS, and doesn't want any user-compiled code to run. However, Xcode command-line developer tools include Clang.

Reference:

[https://www.incredibuild.com/integrations/clang#:~:text=The%20original%20contributor%20of%20Clang,IDE%20\(Integrated%20development%20environment\)](https://www.incredibuild.com/integrations/clang#:~:text=The%20original%20contributor%20of%20Clang,IDE%20(Integrated%20development%20environment))



A screenshot of an iTerm2 window displaying the man page for the Clang compiler. The title bar shows 'man clang'. The window contains the following text:

```
NAME      clang - the Clang C, C++, and Objective-C compiler
SYNOPSIS  clang [options] filename ...
DESCRIPTION
  clang is a C, C++, and Objective-C compiler which encompasses preprocessing, parsing, optimization, code generation, assembly, and linking. Depending on which high-level mode setting is passed, Clang will stop before doing a full link. While Clang is highly integrated, it is important to understand the stages of compilation, to understand how to invoke it. These stages are:
  Driver  The clang executable is actually a small driver which controls the overall execution of other tools such as the compiler, assembler and linker. Typically you do not need to interact with the driver, but you transparently use it to run the other tools.
  Preprocessing
    This stage handles tokenization of the input source file, macro expansion, #include expansion and handling of other preprocessor directives. The output of this stage is typically called a ".i" (for C), ".ii" (for C++), ".mi" (for Objective-C), or ".mii" (for Objective-C++) file.
  Parsing and Semantic Analysis
    This stage parses the input file, translating preprocessor tokens into a parse tree. Once in the form of a parse tree, it applies semantic analysis to compute types for expressions as well and determine whether the code is well formed. This stage is responsible for generating most of the compiler warnings as well as parse errors. The output of this stage is an "Abstract Syntax Tree" (AST).
  Code Generation and Optimization
    This stage translates an AST into low-level intermediate code (known as "LLVM IR") and ultimately to machine code. This phase is responsible for optimizing the generated code and handling target-specific code generation. The output of this stage is typically called a ".s" file or "assembly" file.
    Clang also supports the use of an integrated assembler, in which the code generator produces object files directly. This avoids the overhead of generating the ".s" file and of calling the target assembler.
  Assembler
    This stage runs the target assembler to translate the output of the compiler into a target object file. The output of this stage is typically called a ".o" file or "object" file.
  Linker
    This stage runs the target linker to merge multiple object files into an executable or dynamic library. The output of this stage is typically called an "a.out", ".dylib" or ".so" file.
  Clang Static Analyzer
    The Clang Static Analyzer is a tool that scans source code to try to find bugs through code analysis. This tool uses many parts of Clang and is built into the same driver. Please see <https://clang-analyzer.llvm.org> for more details on how to use the static analyzer.
OPTIONS
  Stage Selection Options
  -E      Run the preprocessor stage.
```

Working on NoMachine (Linux machine from SCU)

1. Be able to have reliable Linux environment

Running on an Linux machine

```
[cboggaram@linux10605 Downloads]$ lsb_release -a
LSB Version: :core-4.1-ia32:core-4.1-noarch
Distributor ID: CentOS
Description:   CentOS Linux release 7.9.2009 (Core)
Release:       7.9.2009
Codename:     Core
[cboggaram@linux10605 Downloads]$
```

2. Compile and run the C program (forktest.c) on a Linux system and to provide a screenshot of the compilation and execution.

```
gcc forktest.c -o forktest
forktest
```

```
[cboggaram@linux10605 Downloads]$ gcc forktest.c -o forktest
[cboggaram@linux10605 Downloads]$ ./forktest
Parent: Process started
Parent: Forking a child.
Child: Process started.
Parent: Wait for child to complete.
Child: Start 10 second idle: 10 9 8 7 6 5 4 3 2 1 0 done!
Child: Terminating.
Parent: Terminating.
[cboggaram@linux10605 Downloads]$ man gcc
[cboggaram@linux10605 Downloads]$
```

3. Read the gcc man page

man gcc

```
NAME
gcc - GNU project C and C++ compiler

SYNOPSIS
gcc [-c | -S | -E] [ -std=standard]
      [-g] [-pg] [-Olevel]
      [-Wwarn...] [-Wpedantic]
      [-fdir...] [-fdit...]
      [-Dmacro[=defn]...] [-Umacro]
      [-foption...] [-Wmachine-option...]
      [-o outfile] [ofile] infile...

Only the most useful options are listed here; see below for the remainder. g++ accepts mostly the same options as gcc.

DESCRIPTION
When you invoke GCC, it normally does preprocessing, compilation, assembly and linking. The "overall options" allow you to stop this process at an intermediate stage. For example, the -c option says not to run the linker. Then the output consists of object files output by the assembler.

Other options are passed on to one stage of processing. Some options control the preprocessor and others the compiler itself. Yet other options control the assembler and linker; most of these are not documented here, since you rarely need to use any of them.

Most of the command-line options that you can use with GCC are useful for C programs; when an option is only useful with another language (usually C++), the explanation says so explicitly. If the description for a particular option does not mention a source language, you can use that option with all supported languages.

The gcc program accepts options and file names as operands. Many options have multi-letter names; therefore multiple single-letter options may not be grouped: -dv is very different from -d -v.

You can mix options and other arguments. For the most part, the order you use doesn't matter. Order does matter when you use several options of the same kind; for example, if you specify -L more than once, the directories are searched in the order specified. Also, the placement of the -l option is significant.

Many options have long names starting with -f or with -W-- for example, -fmove-loop-invariants, -Wformat and so on. Most of these have both positive and negative forms; the negative form of -ffoo is -fno-foo. This manual documents only one of these two forms, whichever one is not the default.

OPTIONS
Option Summary
Here is a summary of all the options, grouped by type. Explanations are in the following sections.

Overall Options
-c -S -E -o file -no-canonical-prefixes -pipe -pass-exit-codes -x language -v -## --help[=class[,...]] --target-help --version -wrapper @file -fplugin=file -fplugin-arg-name=arg
-fdump-ada-spec[-slim] -fada-spec-parent=unit -fdump-go-spec=file

C Language Options
-ansi -std=standard -fgnu89-inline -aux-info filename -fallow-parameterless-variadic-functions -fno-asm -fno-builtin -fno-builtin-function -fhosted -ffreestanding -fopenmp -fms-extensions
-fplain-extensions -trigraphs -traditional -traditional-cpp -fallow-single-precision -fcond-mismatch -fmax-vector-conversions -fsigned-bitfields -fsigned-char -funsigned-bitfields
-funsigned-char

C++ Language Options
-fabi-version=0 -fno-access-control -fcheck-new -fconstexpr-depth=n -ffriend-injection -fno-elide-constructors -fno-enforce-eh-specs -ffor-scope -fno-for-scope -fno-gnu-keywords
-fno-implicit-templates -fno-implicit-inline-templates -fno-implement-inlines -fms-extensions -fno-nanansi-builtins -fnthrow-opt -fno-operator-names -fno-optimal-diags -fpermissive
-fno-prety-templates -frepo -fno-rtti -fstats -ftemplate-backtrace-limit=n -ftemplate-depth=n -fno-threadsafe-statics -fuse-cxa-atexit -fno-weak -nostdincc++ -fno-default-inline
-fvisibility-inlines-hidden -fvisibility-ms-compat -ftext-numeric-literals -Wabi -Wconversion-null -Wctor-dtor-privacy -Wdelete-non-virtual-dtor -Wliteral-suffix -Wnarrowing -Wnoexcept
-Wnon-virtual-dtor -Wreorder -Wrestrict -Wstrict-null-sentinel -Wno-non-template-friend -Wold-style-cast -Woverloaded-virtual -Wno-pmf-conversions -Wsign-promo

Manual page gcc(1) line 1 (press h for help or q to quit)
```