UNIVERSITY OF NATURAL SCIENCE

REPORT

LAB02 Compiler

DATA STRUCTURE AND ALGORITHMS

Trương Quốc An – 1753025

18CLC2

Faculty of Information technology (High Quality Program)

PROCESS OF COMPILING A C/C++ PROGRAM

C/C++ is a high-level language programming, so it needs a compiler to convert the code into an executable code so that the program can be run on our computer.

Inside the Compilation process

Compiling a source code file in C/C++ has a four-step process.

1. Pre-processing
2. Compilation
3. Assembly
4. Linking

For example, if we have a C++ source code and run the compile command

```
g++ -Wall -Werr -std=c++11 -o filename filename.cpp
```

The compilation process will look like:

1. The C++ preprocessor copies the contents of the included header files into the source code file, generates macro code, and replaces symbolic constants defined using #define with their values.
2. The expanded source code file produced by the C++ preprocessor is compiled into the assembly language for the platform.
3. The assembler code generated by the compiler is assembled into the object code for the platform.
4. The object code file generated by the assembler is linked together with the object code files for any library functions used to produce an executable file.

➔ We can stop this compilation process in any stage by using these commands:

1. Stop the process after the preprocessor step:

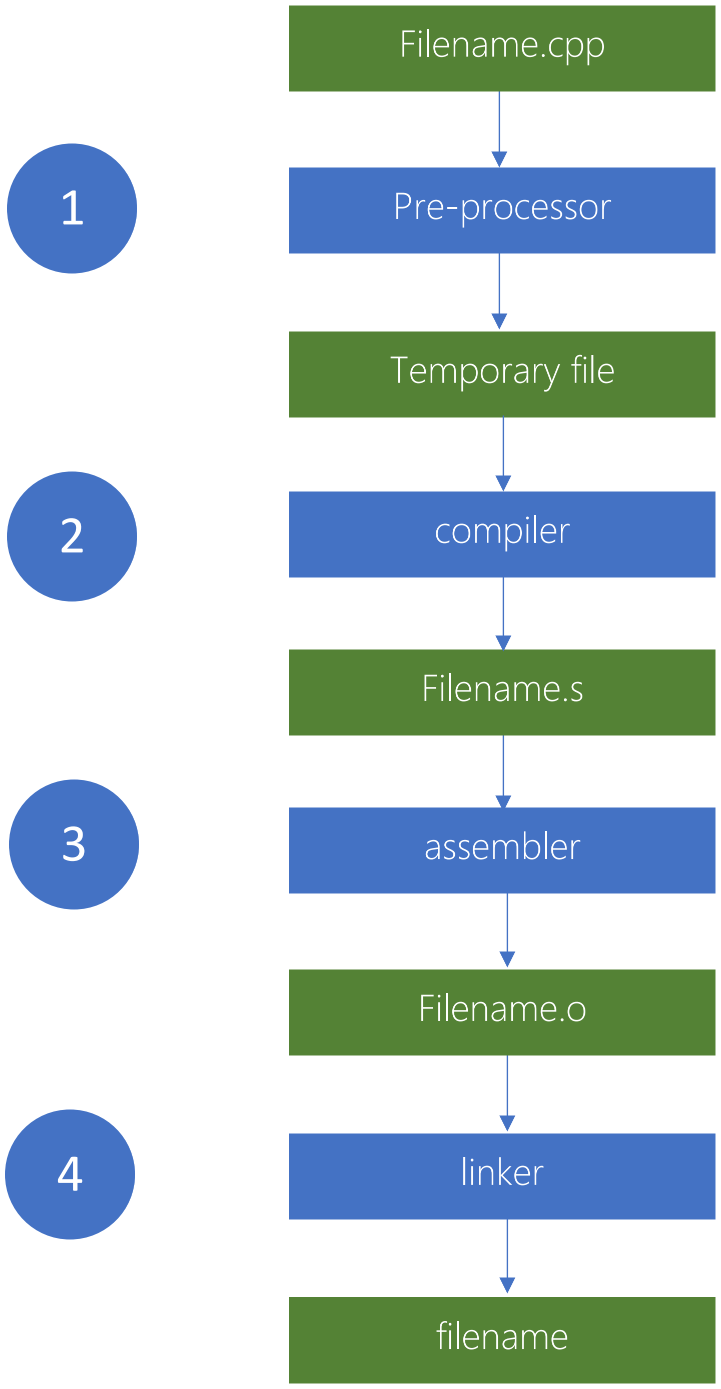
```
g++ -Wall -Werr -std=c++11 -E filename.cpp
```

1. Stop the process after the compile step

```
g++ -Wall -Werr -std=c++11 -S filename.cpp
```

2. Stop the process after the assembly step

```
g++ -Wall -Werr -std=c++11 -C filename.cpp
```



GNU COMPILER COLLECTION (GCC)

GCC is a portable compiler, it runs on most platforms available today and can produre output of many types of processors.

In addition to the processors used in personal computers, it also supports microcontrollers, DSPs, and 64-bit CPUs. GCC is a set of free software release under the GNU General Public License (GPL) and GNU Lesser General Public License (LGPL).

GCC History

The GNU Project - *Richard Stallman* launched the GNU Project in 1984 to build a UNIX-like open source software system. The GNU operating system has not evolved extensively over time. However, it has incubated many excellent and useful open source software tools, such as Make, Sed, Emacs, Glibc, GDB, and GCC as well. These GNU open source software and Linux kernels together constitute the GNU/Linux system. In the beginning, GCC provided stable and reliable compilers, based on the C programming language, for the GNU system. Its full name is GNU C Compiler. Later, more languages (such as Fortran, Obj-C, and Ada) were supported, and the full name of GCC changed to GNU Compiler Collection…

Now, GCC development has entered the "modern chronicle". Facing the competitive pressure of LLVM, the GCC community has actively made many adjustments, such as accelerating compilation and improving the compilation warning information. Over the past 30 years, GCC has evolved from a challenger in the compiler industry to a mainstream compiler for Linux systems, and now faces the challenge of LLVM. Fortunately, the GCC community is making adjustments to accelerate the development of GCC. We can expect that the competition between the two compilation technologies will continue to provide software developers with better compilers.

GCC is not only a native compiler - it can also *cross-compile* any program, producing executable files for a different system from the one used by GCC itself. This allows software to be compiled for embedded systems which are not capable of running a compiler. GCC is written in C with a strong focus on portability, and can compile itself, so it can be adapted to new systems easily.

GCC has multiple language frontends, for parsing different languages. Programs in each language can be compiled, or cross-compiled, for any architecture. For example, an ADA program can be compiled for a microcontroller, or a C program for a supercomputer.

GCC has a modular design, allowing support for new languages and architectures to be added. Adding a new language front-end to GCC enables the use of that language on any architecture, provided that the necessary run-time facilities (such as libraries) are available. Similarly, adding support for a new architecture makes it available to all languages.

Most importantly, GCC is free software, distributed under the GNU General Public License (GNU GPL). This means you have the freedom to use and to modify GCC, as with all GNU software. If you need support for a new type of CPU, a new language, or a new feature you can add it yourself, or hire someone to enhance GCC for you. You can hire someone to fix a bug if it is important for your work.

Furthermore, you have the freedom to share any enhancements you make to GCC. As a result of this freedom you can also make use of enhancements to GCC developed by others. The many features offered by GCC today show how this freedom to cooperate works to benefit you, and everyone else who uses GCC.

Advantages of GCC

- GCC supports more traditional languages than Clang and LLVM, such as Ada, Fortran, and Go.
- GCC supports more less-popular architectures, and supported RISC-V earlier than Clang and LLVM.
- GCC supports more language extensions and more assembly language features than Clang and LLVM. GCC is still the only option for compiling the Linux kernel. Although research on kernel compilation by using Clang and LLVM is also reported in the industry, the kernel cannot be compiled without modifying the source code and compilation parameters.

CLANG COMPILER

The Clang Compiler is an open-source compiler for the C family of programming languages, aiming to be the best in class implementation of these languages. Clang builds on the LLVM optimizer and code generator, allowing it to provide high-quality optimization and code generation support for many targets.

Clang is designed to support the C family of programming languages, which includes C, Objective-C, C++, and Objective-C++ as well as many dialects of those. For language-specific information, please see the corresponding language specific section:

- C Language: K&R C, ANSI C89, ISO C90, ISO C94 (C89+AMD1), ISO C99 (+TC1, TC2, TC3).
- Objective-C Language: ObjC 1, ObjC 2, ObjC 2.1, plus variants depending on base language.
- C++ Language
- Objective C++ Language
- OpenCL C Language: v1.0, v1.1, v1.2, v2.0.

Clang History

Clang is designed to provide a frontend compiler that can replace GCC. Apple Inc. (including NeXT later) has been using GCC as the official compiler. GCC has always performed well as a standard compiler in the open source community. However, Apple Inc. has its own requirements for compilation tools. On the one hand, Apple Inc. added many new features for the Objective-C language (or even, later, the C language). However, GCC developers did not accept these features and assigned low priority to support for these features. Later, they were simply divided into two branches for separate development, and consequently the GCC version released by Apple Inc. is far earlier than the official version. On the other hand, the GCC code is highly coupled and hard to be developed separately. Additionally, in later versions, the code quality continues to decrease. However, many functions required by Apple Inc. (such as improved Integrated Development Environment (IDE) support) must call GCC as a module, but GCC never provides such support. Moreover, the GCC Runtime Library Exemption fundamentally limits the development of LLVM GCC. Also limited by the license, Apple Inc. cannot use LLVM to further improve the code generation quality based on GCC. Therefore, Apple Inc. decided to write the frontend Clang of C, C++, and Objective-C languages from scratch to completely replace GCC.

Clang supports a broad variety of language extensions, which are documented in the corresponding language section. These extensions are provided to be compatible with the GCC, Microsoft, and other popular compilers as well as to improve functionality through Clang-specific features.

The Clang driver and language features are intentionally designed to be as compatible with the GNU GCC compiler as reasonably possible, easing migration from GCC to Clang. In most cases, code "just works". Clang also provides an alternative driver, clang-cl, that is designed to be compatible with the Visual C++ compiler, cl.exe.

In addition to language specific features, Clang has a variety of features that depend on what CPU architecture or operating system is being compiled for. Please see the Target-Specific Features and Limitations section for more details.

Advantages of Clang and LLVM

- Emerging languages are using the LLVM frameworks, such as Swift, Rust, Julia, and Ruby.
- *Clang and LLVM* comply with the C and C ++ standards ***more strictly*** than GCC. GNU Inline and other problems during GCC upgrade do not occur.
- *Clang* also supports some extensions, such as attributes for thread security check.
- *Clang* provides additional useful tools, such as scan-build and clang static analyzer for static analysis, clang-format and clang-tidy for syntax analysis, as well as the editor plug-in Clangd.
- *Clang* provides more accurate and friendly diagnostic information, and highlights error messages, error lines, error line prompts, and repair suggestions. Clang regards the diagnostic information as a feature. The diagnostic information began to be improved only from GCC 5.0 and became mature in GCC 8.

Compilation Performance Comparison

The GCC compilation process is as follows: read the source file, preprocess the source file, convert it into an IR, optimize and generate an assembly file. Then the assembler generates an object file.

Clang and LLVM do not rely on independent compilers but integrate self-implemented compilers at the backend. The process of generating assembly files is omitted in the process of generating object files. The object file is generated directly from the IR.

Besides, compared with the GCC IR, the data structure of LLVM IR is more concise. It occupies less memory during compilation and supports faster traversal. Therefore, Clang and LLVM are advantageous in terms of the compilation time, which is proven by the data obtained from SPEC compilation, as shown in the figure below. Clang reduces the single-thread compilation time by 5% to 10% compared with GCC.

Therefore, Clang offers more advantages for the construction of large projects.

REFERENCE

1. https://medium.com/@alitech_2017/gcc-vs-clang-llvm-an-in-depth-comparison-of-c-c-compilers-899ede2be378
2. http://lampwww.epfl.ch/~fsalvi/docs/gcc/www.network-theory.co.uk/docs/gccintro/gccintro_4.html
3. https://clang.llvm.org/docs/UsersManual.html#introduction
4. https://gcc.gnu.org/onlinedocs/gcc/
5. http://faculty.cs.niu.edu/~mcmahon/CS241/Notes/compile.html