Part 01. 리눅스 개발 환경

Chapter 02.
GCC 설치 및 컴파일

진행 순서

Chapter 02_01 GCC 개요

GCC
http://gcc.gnu.org/

GCC(the GNU Compiler Collection)
GNU 프로젝트의 오픈 소스 컴파일러 컬렉션, 리눅스 계열 플랫폼의 사실상 표준 컴파일러.

지원언어
C, C++, Objective-C, Fortran, Ada, Go

Chapter 02_02 GCC 설치, dnf(yum)

- yum (Yellow dog Updater, Modified)
  - redhat 계열에서 패키지 관리 프로그램인 RPM 기반의 시스템을 위한 자동 업데이터 겸 패키지 설치/제거 도구
  - 페도라, CentOS 등 많은 RPM 기반 리눅스 배포판에서 사용
  - 우분투 등 데비안 계열의 apt(Advanced Packaging Tool)와 유사

  기본 사용법
  - 패키지 설치 : yum install 패키지명
  - 패키지 삭제 : yum remove 패키지명
  - 패키지 업그레이드 : yum update 패키지명
  - 패키지 조회 : yum search 패키지명
  - 패키지 목록 : yum list 패키지명
  - yum 데이터베이스 동기화 업데이트 : yum update

- dnf: CentOS 8 에서 도입된 command, 기존 yum과 동일

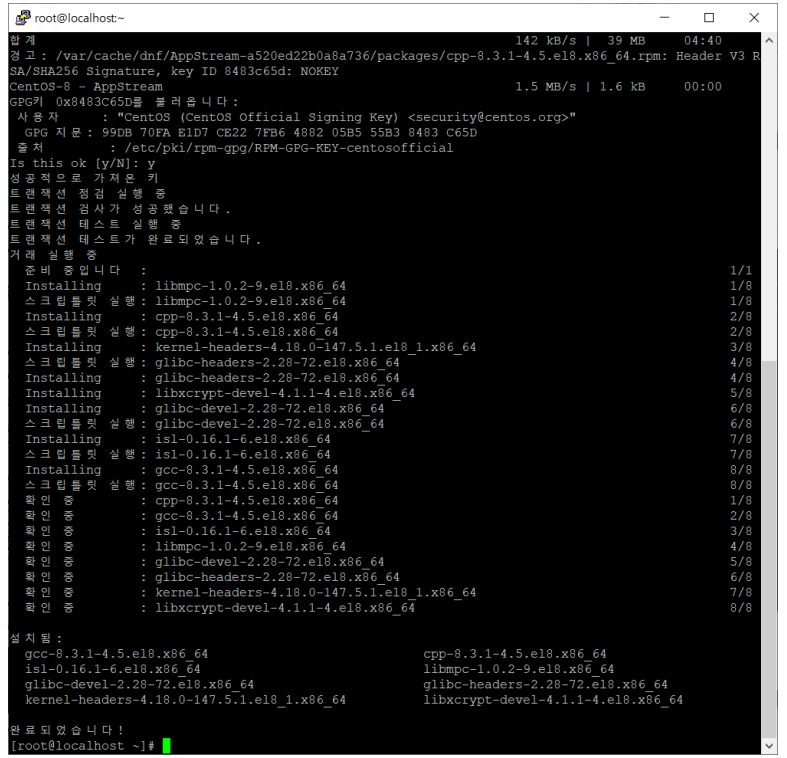
Chapter 02_02 GCC 설치, dnf(yum)

```
root@localhost:~
[root@localhost ~]# yum install gcc
마지막 메타 데이터 만료 확인 : 0:21:16 전에 2020년 03월 24일 (화) 오전 08시 39분 45초.
Dependencies resolved.
================================================================================
 Package                Architecture    Version                      Repository          Size
================================================================================
Installing:
 gcc                    x86_64          8.3.1-4.5.el8                AppStream           23 M
Installing dependencies:
 cpp                    x86_64          8.3.1-4.5.el8                AppStream           10 M
 isl                    x86_64          0.16.1-6.el8                 AppStream          841 k
 libmpc                 x86_64          1.0.2-9.el8                  AppStream           59 k
 glibc-devel            x86_64          2.28-72.el8                  BaseOS             1.0 M
 glibc-headers          x86_64          2.28-72.el8                  BaseOS             469 k
 kernel-headers         x86_64          4.18.0-147.5.1.el8_1         BaseOS             2.7 M
 libxcrypt-devel        x86_64          4.1.1-4.el8                  BaseOS              25 k

Transaction Summary
================================================================================
설치  8 Packages

Total download size: 39 M
Installed size: 98 M
Is this ok [y/N]:
```

참고사항: C++ 컴파일러 설치의 경우
yum install gcc-c++

yum install gcc

- 의존성(dependency)가 걸려 있는 모든 패키지 설치 해줌
- 옵션 y: 진행 중간에 나오는 입력란 자동으로 y(yes) 처리 해줌
  ex) yum –y install gcc



Chapter 02_02 GCC 설치, dnf(yum)

설치 확인

```
root@localhost:~
[root@localhost ~]# yum list gcc
마지막 메타 데이터 만료 확인 : 0:44:44 전에 2020년 03월 24일 (화) 오전 08시 39분 45초.
설치된 패키지
gcc.x86_64                              8.3.1-4.5.el8                              @AppStream
[root@localhost ~]# gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/libexec/gcc/x86_64-redhat-linux/8/lto-wrapper
OFFLOAD_TARGET_NAMES=nvptx-none
OFFLOAD_TARGET_DEFAULT=1
Target: x86_64-redhat-linux
Configured with: ../configure --enable-bootstrap --enable-languages=c,c++,fortran,lto --prefix=/us
r --mandir=/usr/share/man --infodir=/usr/share/info --with-bugurl=http://bugzilla.redhat.com/bugzi
lla --enable-shared --enable-threads=posix --enable-checking=release --enable-multilib --with-syst
em-zlib --enable-__cxa_atexit --disable-libunwind-exceptions --enable-gnu-unique-object --enable-l
inker-build-id --with-gcc-major-version-only --with-linker-hash-style=gnu --enable-plugin --enable
-initfini-array --with-isl --disable-libmpx --enable-offload-targets=nvptx-none --without-cuda-dri
ver --enable-gnu-indirect-function --enable-cet --with-tune=generic --with-arch_32=x86-64 --build=
x86_64-redhat-linux
Thread model: posix
gcc version 8.3.1 20190507 (Red Hat 8.3.1-4) (GCC)
[root@localhost ~]#
```

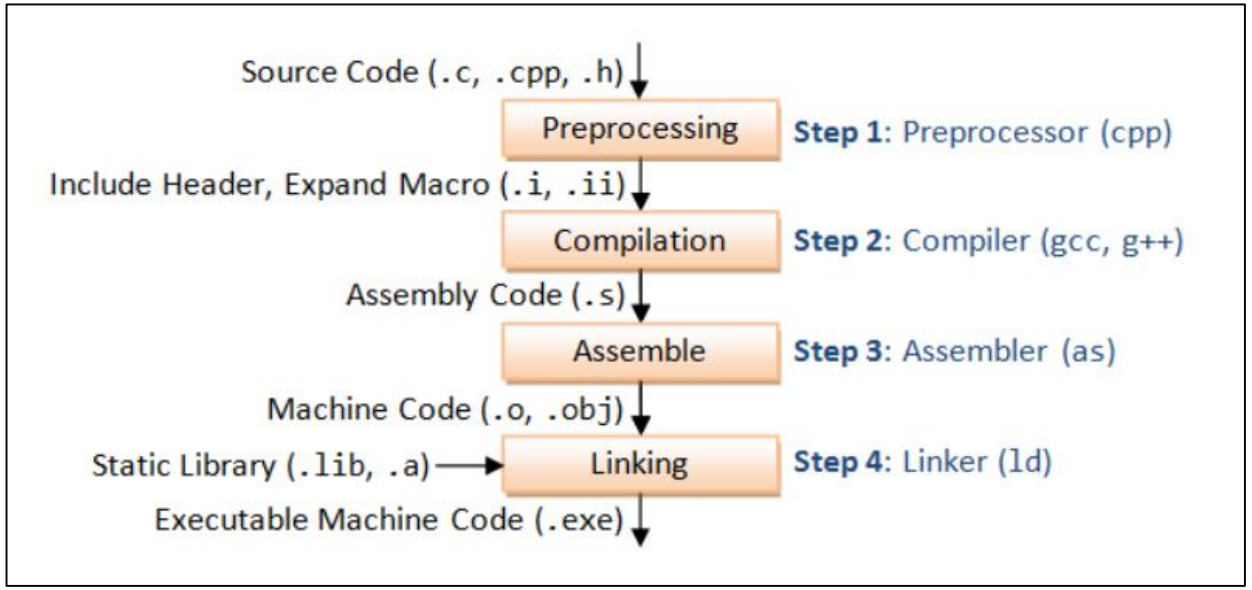
```c
#include <stdio.h>

int main()
{
        printf("Hello World!\n");
        return 0;
}
```

```
[root@localhost ~]# gcc hello.c -o hello
[root@localhost ~]# ls -hl hello
-rwxr-xr-x. 1 root root 11K  3월  24 09:29 hello
[root@localhost ~]# ./hello
Hello World!
[root@localhost ~]#
```

gcc infile [–o outfile]
: -o 없다면 a.out

Chapter 02_03 컴파일러 개요

gcc 컴파일 처리 과정



출처: https://www3.ntu.edu.sg/home/ehchua/programming/cpp/gcc_make.html

- 전처리기(Preprocessor) : 헤더(#include), 매크로(#define) 등과 같은 전처리 지시자 해석
- 컴파일러(Compiler) : 소스코드를 어셈블리어(*.s) 형태로 변환
  (인간이 이해하기 쉬운 언어를 컴퓨터가 이해하기 쉬운 언어로 번역)
- 어셈블러(Assembler) : 어셈블리 코드를 기계어(Machine Code) 오브젝트 파일(*.o)로 변환
- 링커(Linker) : 생성된 목적 파일들을 묶고 라이브러리를 링킹하여 실행파일을 생성

Chapter 02_03 컴파일러 개요

hello.c – 소스코드
| 전처리기: 헤더(#include), 매크로(#define) 처리
hello.i – 확장된 소스 코드를 포함한 중간 파일
| 컴파일러: 시스템 프로세서용 어셈블리 코드로
hello.s – 어셈블리 파일
| 어셈블러: 어셈블리 코드를 기계어로
hello.o – 오브젝트 파일
| 링커: 라이브러리 링킹
hello – 실행파일

1) 전처리 과정

```
# cpp hello.c  >  hello.i
```

결과로 중간 파일 " hello.i " 에는
확장 된 소스 코드가 포함

hello.i

```
# 1 "hello.c"
...
# 1 "/usr/include/stdio.h" 1 3 4
# 27 "/usr/include/stdio.h" 3 4
...
# 1 "/usr/lib/gcc/x86_64-redhat-linux/8/include/stddef.h" 1 3 4
# 216 "/usr/lib/gcc/x86_64-redhat-linux/8/include/stddef.h" 3 4
...
# 1 "/usr/lib/gcc/x86_64-redhat-linux/8/include/stdarg.h" 1 3 4
# 40 "/usr/lib/gcc/x86_64-redhat-linux/8/include/stdarg.h" 3 4
....
extern int printf (const char *__restrict __format, ...);
...
int main()
{
 printf("Hello World!\n");
 return 0;
}
```

Chapter 02_03 컴파일러 개요

hello.c – 소스코드
| 전처리기: 헤더(#include), 매크로(#define) 처리
hello.i – 확장된 소스 코드를 포함한 중간 파일
| 컴파일러: 시스템 프로세서용 어셈블리 코드로
hello.s – 어셈블리 파일
| 어셈블러: 어셈블리 코드를 기계어로
hello.o – 오브젝트 파일
| 링커: 라이브러리 링킹
hello – 실행파일

2) 컴파일 과정

```
# gcc  -S  hello.i
```

-S 옵션은 객체 코드 대신
어셈블리 코드를 생성하도록 지정
결과 어셈블리 파일은 "hello.s"

hello.s

```
 .file   "hello.c"
        .text
        .section        .rodata
.LC0:
        .string "Hello World!"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    $.LC0, %edi
        call    puts
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (GNU) 8.3.1 20190507 (Red Hat 8.3.1-4)"
        .section        .note.GNU-stack,"",@progbits
```

Chapter 02_03 컴파일러 개요

hello.c – 소스코드
| 전처리기: 헤더(#include), 매크로(#define) 처리
hello.i – 확장된 소스 코드를 포함한 중간 파일
| 컴파일러: 시스템 프로세서용 어셈블리 코드로
hello.s – 어셈블리 파일
| 어셈블러: 어셈블리 코드를 기계어로
hello.o – 오브젝트 파일
| 링커: 라이브러리 링킹
hello – 실행파일

3) 어셈블리 과정

```
# as –o hello.o hello.s
```

참고사항:

ELF는 실행 가능한 바이너리 또는 오브젝트 파일 등의 형식을 규정한 것이다.
ELF파일은 ELF헤더가 맨 앞에 위치하고, 프로그램 헤더 테이블과 섹션 헤더 테이블이 그 뒤에 위치한다.

```
# readelf –h hello.o
```

hello.o

```
^?ELF^B^A^A^@^@^@^@^@^@^@^@^@^A^@>^@^A^
@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@<98>^B^@^@^@^@^@^@^@^@^@@^@^@^@^
@^@@^@^M^@^L^@UH<89>å¿^@^@^@^@è^@^@^@
^@¸^@^@^@^@]ÃHello World!^@^@GCC: (GNU) 8.3.1
20190507 (Red Hat 8.3.1-
4)^@^@^T^@^@^@^@^@^@^@^AzR^@^Ax^P^A^[^L^G
^H<90>^A^@^@^\^@^@^@^\^@^@^@^@^@^@^@^U^
@^@^@^@A^N^P<86>^BC^M^FP^L^G^H^@^@^@^@^
@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^A^@^@^@^D^@ñÿ^@^@^@^@^@^@^
@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^C^@^A^@^
@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^C^@^C^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^C^@^D^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^C^@^E^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^
@^@^@^C^@^G^@^@^@^@^@^@^@^@^@^@^@^@^@^
@^@^@^@^@^@^@^@^@^C^@^H^@^@^@^@^@^@^
@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^C^@^F^
@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^R^@^A^@^@^@^@^@^@^@^@^@^U^@^@^
@^@^@^@^@^N^@^@^@^P^@^@^@^@^@^@^@^@^
@^@^@^@^@^@^@^@^@^@^@^@^@^@hello.c^@main^@pu
ts^@^@^@^@^@^@^E^@^@^@^@^@^@^@^@
```

Chapter 02_03 컴파일러 개요

hello.c – 소스코드
| 전처리기: 헤더(#include), 매크로(#define) 처리
hello.i – 확장된 소스 코드를 포함한 중간 파일
| 컴파일러: 시스템 프로세서용 어셈블리 코드로
hello.s – 어셈블리 파일
| 어셈블러: 어셈블리 코드를 기계어로
hello.o – 오브젝트 파일
| 링커: 라이브러리 링킹
hello – 실행파일

4) 링킹 과정

```
# ld  -o  hello  hello.o  ...libraries...
```

참고사항
gcc 전체 컴파일 과정 확인
gcc –v –save-temp –o hello hello.c

gcc –v –save-temp –o hello hello.c

```
...
Target: x86_64-redhat-linux
Configured with: ../configure --enable-bootstrap --enable-languages=c,c++,fortran,lto --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/info --with-bugurl=http://bugzilla.redhat.com/bugzilla --enable-shared --enable-threads=posix --enable-checking=release --enable-multilib --with-system-zlib --enable-__cxa_atexit --disable-libunwind-exceptions --enable-gnu-unique-object --enable-linker-build-id --with-gcc-major-version-only --with-linker-hash-style=gnu --enable-plugin --enable-initfini-array --with-isl --disable-libmpx --enable-offload-targets=nvptx-none --without-cuda-driver --enable-gnu-indirect-function --enable-cet --with-tune=generic --with-arch_32=x86-64 --build=x86_64-redhat-linux
...
COMPILER_PATH=/usr/libexec/gcc/x86_64-redhat-linux/8/:/usr/libexec/gcc/x86_64-redhat-linux/8/:/usr/libexec/gcc/x86_64-redhat-linux/:/usr/lib/gcc/x86_64-redhat-linux/8/:/usr/lib/gcc/x86_64-redhat-linux/
LIBRARY_PATH=/usr/lib/gcc/x86_64-redhat-linux/8/:/usr/lib/gcc/x86_64-redhat-linux/8/../../../../lib64/:/lib/../lib64/:/usr/lib/../lib64/:/usr/lib/gcc/x86_64-redhat-linux/8/../../../:/lib/:/usr/lib/
...
```

Chapter 02_04 GCC 활용

GCC 주요 옵션 정리

-o : output 실행 파일 이름 지정
ex) gcc hello.c –o hello

-Wall : 모든 경고 활성화 (경고 메시지)
ex) gcc –Wall hello.c –o hello

-E : 전처리 과정 결과 생성
ex) gcc –E hello.c > hello.i

-S : 어셈블리 코드 생성
ex) gcc –S hello.c > hello.s

-C : 컴파일 코드 생성(링킹 없음)
ex) gcc –C hello.c

-save-temps : 모든 컴파일 중간파일 생성
ex) # gcc -save-temps hello.c
ls
a.out hello.c hello.i hello.o hello.s

-l : 공유 라이브러리 링크
ex) gcc –Wall hello.c –o hello -lpthread

-fPIC : 위치 독립적인 코드 생성
공유 라이브러리를 작성하는 동안 위치 독립적인 코드가 생성 되어야함
이를 통해 공유 라이브러리가 고정 주소 대신 임의의 주소로 로드 될 수 있습니다. 이를 위해 -fPIC 옵션이 사용
ex) 아래 명령은 소스 파일 Cfile.c 로 부터 libCfile.so 공유 라이브러리를 생성하는 명령
gcc –c -Wall –Werror –fPIC Cfile.c
gcc –shared –o libCfile.so Cfile.o
따라서 공유 라이브러리를 만드는데 –fPIC 옵션이 사용 된 것을 볼 수 있음

-v : 모든 실행 커맨드 출력
ex) gcc –Wall –v hello.c –o hello
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/i686-linux-gnu/4.6/lto-wrapper
Target: i686-linux-gnu
...

-ansi : ISO C89 스타일 지원
ex) ISO C89 스타일의 경우 C++ 주석(//)을 지원하지 않으므로 사용 시 에러

Chapter 02_04 GCC 활용

GCC 주요 옵션 정리

-funsigned-char : char를 unsigned char로 취급
ex) char c = -10;
printf("c is %d\n", c);
# gcc –Wall –funsigned-char main.c –o main
# ./main
c is 246

-fsinged-char : char 변수를 signed로 취급
./main
c is -10

-D[Macro] : 컴파일 시점 사용자 지정 매크로
(define 유사)

```
int main() {
#ifdef MY_MACRO
    printf("Macro Defined\n");
#endif
    printf("Hello World\n");
}
```

gcc –Wall –DMY_MACRO main.c –o main
./main
Macro Defined
Hello World

-Werror : 경고를 에러로 변환
gcc 컴파일 경고가 에러로 표현됨

@file : gcc 옵션은 파일을 통해 제공될 수 도 있습니다.
옵션을 포함하는 파일 이름 앞에 @ 옵션을 사용하여 수행
cat opt_file
-Wall –o main
gcc main.c @opt_file

-I : 전처리 과정에서 헤더 파일을 탐색하는 기본 디렉토리를 추가
gcc –I../include .....

-U[Macro] : -D와 반대로 소스 코드 내에 #undef[Macro] 옵션을 추가 동일

최적화 옵션 : 실행파일의 크기를 줄여 실행 속도를 향상
-O0 : 최적화를 수행하지 안는다.
-O1 : -O0 보다 조금 낫다
-O2 : 가장 많이 사용, 일반 응용 프로그램이나 커널 컴파일할 때 사용
-O3 : 가장 높은 레벨의 최적화, 모든 함수를 인라인 함수와 같이 취급
-O5 : 사이즈 최적화 실행, 임베디드 시스템 등 자원이 협소한 곳에서 사용

디버깅 옵션:
-g : gdb 제공 정보를 바이너리에 삽입