# 오픈 소스와 MAKEFILE 빌드

컴퓨터공학과 201631001 고성훈

# MAKEFILE 프로젝트 시나리오(CONCEPT)

- 오픈 소스 프로젝트라고 가정함.
- release 빌드와 debug 빌드를 나눠야 한다.
- 외부 오픈 소스 프로젝트를 참조한다.
  - Shared object 단위 컴파일은 제외; 오직 소스 코드 통째로 빌드.
- 단위 테스트를 사용한다.
- `make all` 명령으로 1. 빌드, 2. 단위 테스트, 3. 패키징 작업이 모두 완수되어야 한다.
  - 자동 빌드 시스템에서 돌아갈 수 있도록.
  - 단위 테스트 실패 시 패키징 작업을 수행하지 않는다.
- release 빌드 시에는 1. 빌드 결과물, 2. 소스 코드를 패키징한다.
  - 패키지 파일 이름에는 빌드 시간이 포함되어야 함.

```
kos@DESKTOP-S31FFVV ~/w4/assig
❯ tree --dirsfirst
.
├── inc
│   ├── assert_driver.h
│   ├── boolean.h
│   ├── hello.h
│   ├── my_assert.h
│   └── my_math.h
├── lib
├── src
│   ├── assert_driver.c
│   ├── hello.c
│   ├── main.c
│   ├── my_assert.c
│   └── my_math.c
├── test
│   ├── testlib.c
│   ├── testlib.h
│   └── testmain.c
└── Makefile
```

# 초기 디렉토리 구조

- ‘inc’
  - 프로젝트에서 사용할 사용자 정의 헤더 파일을 보관.
- ‘lib’
  - 외부 라이브러리 소스 코드를 보관.
  - ‘make clone-all’ 명령을 통해 초기화해야 함.
- ‘src’
  - 프로젝트 소스 코드를 보관.
- ‘test’
  - 단위 테스트 헤더 및 소스 코드를 보관.
  - 프로젝트의 일부지만, product 프로그램과는 별개.

# 라이브러리 초기화를 위한 'MAKE CLONE-ALL' 명령 수행

```
 kos@DESKTOP-S31FFVV  ~/w4/assign
❯ make clone-all
Cloning into 'lib/log.c'...
remote: Enumerating objects: 15, done.
remote: Total 15 (delta 0), reused 0 (delta 0), pack-reused 15
Unpacking objects: 100% (15/15), done.
Cloning into 'lib/munit'...
remote: Enumerating objects: 608, done.
remote: Total 608 (delta 0), reused 0 (delta 0), pack-reused 608
Receiving objects: 100% (608/608), 232.77 KiB ¦ 408.00 KiB/s, done.
Resolving deltas: 100% (396/396), done.
```

```
kos@DESKTOP-S31FFVV ~/w4/assign
❯ tree --dirsfirst
.
├── inc
│   ├── assert_driver.h
│   ├── boolean.h
│   ├── hello.h
│   ├── my_assert.h
│   └── my_math.h
├── lib
│   ├── log.c
│   │   ├── src
│   │   │   ├── log.c
│   │   │   └── log.h
│   │   ├── LICENSE
│   │   └── README.md
│   └── munit
│       ├── COPYING
│       ├── Makefile
│       ├── README.md
│       ├── example.c
│       ├── meson.build
│       ├── munit.c
│       └── munit.h
├── src
│   ├── assert_driver.c
│   ├── hello.c
│   ├── main.c
│   ├── my_assert.c
│   └── my_math.c
├── test
```

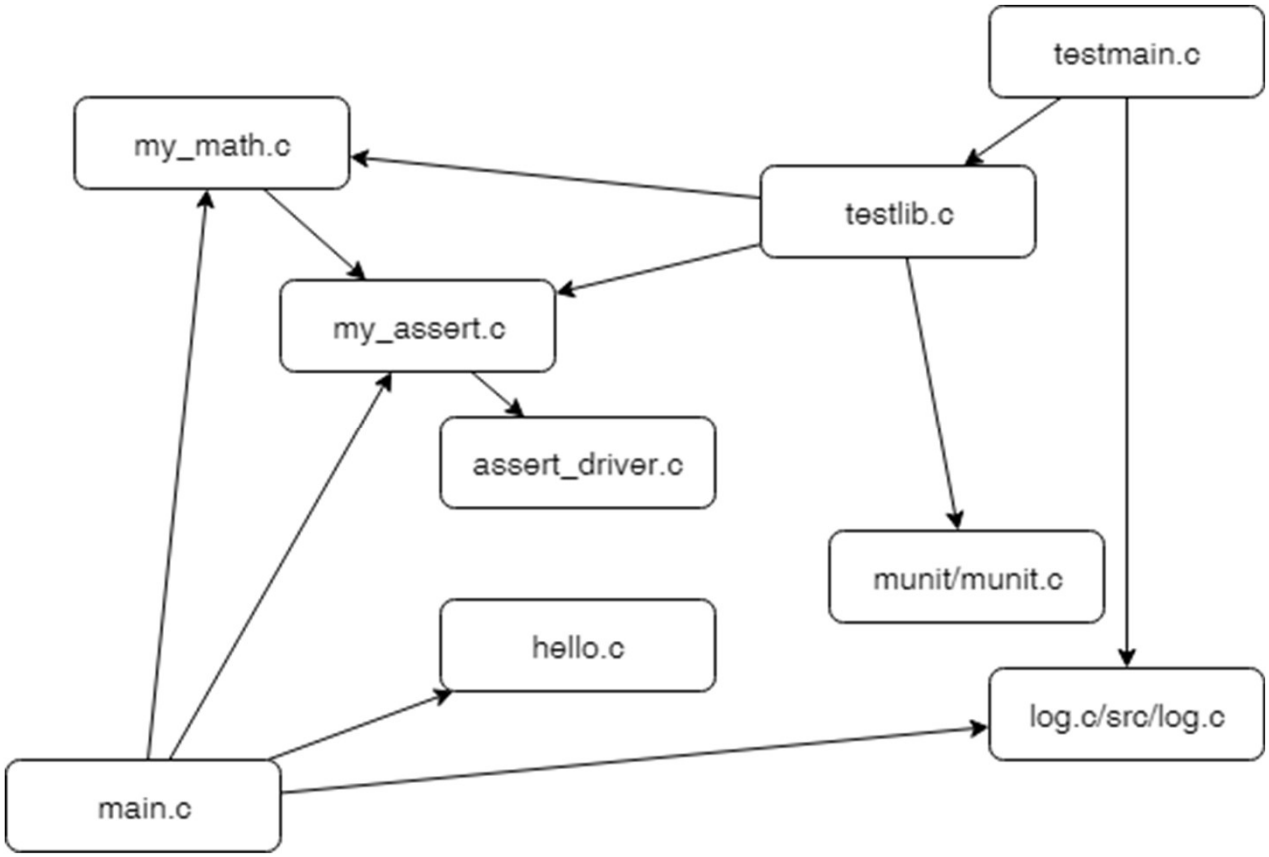
# ‘MAKE CLONE-ALL’ 명령 수행 후

- ‘lib/log.c’
  - 로그 관련 라이브러리.
  - 출처: https://github.com/MewingFlame/log.c.git
- ‘lib/munit’
  - 단위 테스트 수행을 위한 프레임워크 라이브러리.
  - 출처: https://github.com/nemequ/munit.git
  - 단위 테스트(unit test)란?
    - 소스 코드의 단위(unit) 수준의 테스트를 수행함.
    - 단위는 프로그래머 마음대로.
      일반적으로 논리적으로 구분되는 한 코드 덩어리가 단위임.

# ‘MAKE CLONE-ALL’ 없이 ‘MAKE ALL’ 실행하면…

```
 kos@DESKTOP-S31FFVV  ~/w4/assign
❯ make
make: *** No rule to make target 'lib/log.c/src/log.c',
 needed by 'obj/log.c/log.o'.  Stop.
```

# 종속성 다이어그램


testmain.c
my_math.c
testlib.c
my_assert.c
assert_driver.c
munit/munit.c
hello.c
log.c/src/log.c
main.c

```
######################################################################
## Macro definitions.

MAKEFLAGS               += -r
CPPFLAGS                = -I . -I $(LIBDIR)
CFLAGS                  = -Wall -Wextra
LDFLAGS                 = -lm
RM                      = rm -f
MKDIR                   = mkdir -p

# Directories.
OUTDIR                  = .
INCDIR                  = inc
SRCDIR                  = src
TESTDIR                 = test
LIBDIR                  = lib
OBJDIR                  = $(OUTDIR)/obj
DISTDIR                 = $(OUTDIR)/dist

# Targets.
BUILD_TARGET            = $(OUTDIR)/prog.out
TEST_TARGET             = $(OUTDIR)/test.out
SRC_TAR_TARGET          = $(DISTDIR)/src.$(DATETIME).tar
PRODUCT_TAR_TARGET      = $(DISTDIR)/product.$(DATETIME).tar

# Dependencies.
BUILD_DEPENDENCIES      = $(LOGGING_DEPENDENCIES) \
                          $(OBJDIR)/hello.o $(OBJDIR)/my_math.o \
                          $(OBJDIR)/my_assert.o \
                          $(OBJDIR)/assert_driver.o
LOGGING_DEPENDENCIES    = $(OBJDIR)/log.c/log.o
MUNIT_DEPENDENCIES      = $(OBJDIR)/munit/munit.o
TEST_DEPENDENCIES       = $(MUNIT_DEPENDENCIES) \
                          $(BUILD_DEPENDENCIES) \
                          $(OBJDIR)/testlib.o

# Shell commands.
BUILD_OBJ               = $(CC) $(CPPFLAGS) $(CFLAGS) -c -o $@ $<
BUILD                   = $(CC) $(CPPFLAGS) $(CFLAGS) -o $@ $^ $(LDFL
DATETIME                = $(shell date +%FT%Hh%Mm%Ss)
ECHO_TARGET_COMPLETE    = \
        @echo "\e[1;34m-=- target \e[1;32m$@\e[1;34m has completed -=
        echo ""

LOG.C_GIT_URL = "https://github.com/MewingFlame/log.c.git"
MUNIT_GIT_URL = "https://github.com/nemequ/munit.git"
```

# MAKEFILE 1. 매크로 정의

- 사전 정의 매크로 재정의
- 디렉토리 경로
- 타겟
  - BUILD_TARGET
  - TEST_TARGET
  - SRC/PRODUCT_TAR_TARGET
- 종속성
  - BUILD_DEPENDENCIES
  - LOGGING_DEPENDENCIES
  - MUNIT_DEPENDENCIES
  - TEST_DEPENDENCIES
- 쉘 명령
  - BUILD_OBJ
  - BUILD
  - DATETIME
  - ECHO_TARGET_COMPLETE
- `git clone` URL

```
######################################################################
## Build targets.

all: release run-test dist

clone-all: clone-log.c clone-munit

# Release targets.
release: CFLAGS += -O3
release: $(BUILD_TARGET)

# Debug targets.
debug: CPPFLAGS += -DDEBUG
debug: $(BUILD_TARGET) test
	$(ECHO_TARGET_COMPLETE)

# Test targets.
test: CPPFLAGS += -DTEST$(TEST_OPT)
test: clone-munit $(TEST_TARGET)

run-test: test
	$(TEST_TARGET)
	$(ECHO_TARGET_COMPLETE)

# Distribution targets.
dist: package-dist package-src

package-src: $(SRC_TAR_TARGET)

package-dist: $(PRODUCT_TAR_TARGET)

# Clean dummy targets.
clean:
	$(RM) -r $(OBJDIR)
	$(RM) *.out

clean-all: clean
	$(RM) -r dist

clean-lib:
	$(RM) -r lib/*
```

# MAKEFILE 2. 빌드 타겟

- ‘all’
  - 시나리오가 지정한 대로 빌드 과정 수행
- ‘clone-all’
  - 프로젝트 종속성 외부 라이브러리의 다운로드(clone) 명령 수행
- ‘release’
  - 배포 가능한 릴리즈 버전으로 컴파일
  - ‘CFLAGS += -O3‘로 최적화 옵션 추가
- ‘debug’
  - 디버그용으로 컴파일
  - ‘CPPFLAGS += -DDEBUG’로 ‘DEBUG’ 매크로 수동 지정
- ‘test’
  - 단위 테스트 실행기(runner) 컴파일
- ‘dist’
  - 배포를 위한 ‘.tar’ 압축파일 생성 (프로그램, 소스코드 두 개)
- ‘clean’
  - ‘clean’: 빌드 결과물과 중간 결과물만 삭제
  - ‘clean-all’: 배포판 패키징까지 모두 삭제
  - ‘clean-lib’: ‘make clone-all’ 명령을 통해 다운로드한 외부 라이브러리만 삭제

# MAKEFILE 3.
# 라이브러리 타겟

- Log.c 라이브러리
- Munit 라이브러리

```
###############################################################################
## Library targets.

# Logging targets.
clone-log.c:
	@test -e $(LIBDIR)/log.c || git clone $(LOG.C_GIT_URL) $(LIBDIR)/log.c

$(OBJDIR)/log.c/log.o: CPPFLAGS += -DLOG_USE_COLOR
$(OBJDIR)/log.c/log.o: $(LIBDIR)/log.c/src/log.c
	@$(MKDIR) $(OBJDIR) && $(MKDIR) $(OBJDIR)/log.c
	$(BUILD_OBJ)

# Munit targets.
clone-munit:
	@test -e $(LIBDIR)/munit || git clone $(MUNIT_GIT_URL) $(LIBDIR)/munit

$(OBJDIR)/munit/%.o: $(LIBDIR)/munit/%.c
	@$(MKDIR) $(OBJDIR) && $(MKDIR) $(OBJDIR)/munit
	$(BUILD_OBJ)
```

# MAKEFILE 4.
# 파일 기반 타겟

- 소스 코드 패키징 타겟(src.tar)
- 프로그램 패키징 타겟(product.tar)
- 프로그램 빌드 타겟(prog.out)
- 테스트 실행기 빌드 타겟(test.out)
- 테스트 코드 컴파일 타겟
- 프로그램 코드 컴파일 타겟

```
###############################################################################
## File targets.

$(SRC_TAR_TARGET):
	@$(MKDIR) $(DISTDIR)
	tar -cvf $@ Makefile $(INCDIR) $(SRCDIR) $(TESTDIR)
	$(ECHO_TARGET_COMPLETE)

$(PRODUCT_TAR_TARGET): $(BUILD_TARGET)
	@$(MKDIR) $(DISTDIR)
	tar -cvf $@ $(BUILD_TARGET)
	$(ECHO_TARGET_COMPLETE)

$(BUILD_TARGET): $(BUILD_DEPENDENCIES) $(OBJDIR)/main.o
	$(BUILD)
	$(ECHO_TARGET_COMPLETE)

$(TEST_TARGET): $(TEST_DEPENDENCIES) $(OBJDIR)/testmain.o
	$(BUILD)
	$(ECHO_TARGET_COMPLETE)

$(OBJDIR)/test%.o: $(TESTDIR)/test%.c
	@$(MKDIR) $(OBJDIR)
	$(BUILD_OBJ)

$(OBJDIR)/%.o: $(SRCDIR)/%.c
	@$(MKDIR) $(OBJDIR)
	$(BUILD_OBJ)
```

```
 kos@DESKTOP-S31FFVV  ~/w4/assign
❯ make
cc -I . -I lib -DLOG_USE_COLOR -Wall -Wextra -O3 -c -o obj/log.c/log.o lib/log.c/src/log.c
cc -I . -I lib -Wall -Wextra -O3 -c -o obj/hello.o src/hello.c
cc -I . -I lib -Wall -Wextra -O3 -c -o obj/my_math.o src/my_math.c
cc -I . -I lib -Wall -Wextra -O3 -c -o obj/my_assert.o src/my_assert.c
cc -I . -I lib -Wall -Wextra -O3 -c -o obj/assert_driver.o src/assert_driver.c
cc -I . -I lib -Wall -Wextra -O3 -c -o obj/main.o src/main.c
cc -I . -I lib -Wall -Wextra -O3 -o prog.out obj/log.c/log.o obj/hello.o obj/my_math.o obj/my_assert.o obj/assert_driver.o obj/main.o -lm
-=- target prog.out has completed -=-

cc -I . -I lib -DTEST -Wall -Wextra -c -o obj/munit/munit.o lib/munit/munit.c
cc -I . -I lib -DTEST -Wall -Wextra -c -o obj/testlib.o test/testlib.c
cc -I . -I lib -DTEST -Wall -Wextra -c -o obj/testmain.o test/testmain.c
cc -I . -I lib -DTEST -Wall -Wextra -o test.out obj/munit/munit.o obj/log.c/log.o obj/hello.o obj/my_math.o obj/my_assert.o obj/assert_driver.o obj/testlib.o obj/testmain.o -lm
-=- target test.out has completed -=-

./test.out
01:54:13 DEBUG test/testmain.c:9: Running tests...
Running test suite with seed 0xb3968281...
/example/compare                    [ OK    ] [ 0.00001320 / 0.00000000 CPU ]
/example/rand                       [ OK    ] [ 0.00000890 / 0.00000000 CPU ]
/example/parameters
  foo=one, bar=red                  [ OK    ] [ 0.00000730 / 0.00000000 CPU ]
  foo=one, bar=green                [ OK    ] [ 0.00000630 / 0.00000000 CPU ]
  foo=one, bar=blue                 [ OK    ] [ 0.00000630 / 0.00000000 CPU ]
  foo=two, bar=red                  [ OK    ] [ 0.00000940 / 0.00000000 CPU ]
  foo=two, bar=green                [ OK    ] [ 0.00000630 / 0.00000000 CPU ]
  foo=two, bar=blue                 [ OK    ] [ 0.00000590 / 0.00000000 CPU ]
  foo=three, bar=red                [ OK    ] [ 0.00000600 / 0.00000000 CPU ]
  foo=three, bar=green              [ OK    ] [ 0.00000780 / 0.00000000 CPU ]
  foo=three, bar=blue               [ OK    ] [ 0.00000940 / 0.00000000 CPU ]
11 of 11 (100%) tests successful, 0 (0%) test skipped.
-=- target run-test has completed -=-

tar -cvf dist/product.2019-10-08T01h54m12s.tar ./prog.out
./prog.out
-=- target dist/product.2019-10-08T01h54m12s.tar has completed -=-

tar -cvf dist/src.2019-10-08T01h54m12s.tar Makefile inc src test
Makefile
inc/
inc/assert_driver.h
inc/boolean.h
inc/hello.h
inc/my_assert.h
inc/my_math.h
src/
src/assert_driver.c
src/hello.c
src/main.c
src/my_assert.c
src/my_math.c
test/
test/testlib.c
test/testlib.h
test/testmain.c
-=- target dist/src.2019-10-08T01h54m12s.tar has completed -=-

 kos@DESKTOP-S31FFVV  ~/w4/assign
❯
```

# ‘MAKE ALL’ 명령 수행

- release
  - $(BUILD_TARGET)
- run-test
  - test
    - $(TEST_TARGET)
- dist
  - package-src
  - package-dist

# ‘MAKE ALL’ 1. 프로그램 빌드 (RELEASE)

```
 kos@DESKTOP-S31FFVV  ~/w4/assign
❯ make
cc -I . -I lib -DLOG_USE_COLOR -Wall -Wextra -O3 -c -o obj/log.c/log.o lib/log.c
/src/log.c
cc -I . -I lib -Wall -Wextra -O3 -c -o obj/hello.o src/hello.c
cc -I . -I lib -Wall -Wextra -O3 -c -o obj/my_math.o src/my_math.c
cc -I . -I lib -Wall -Wextra -O3 -c -o obj/my_assert.o src/my_assert.c
cc -I . -I lib -Wall -Wextra -O3 -c -o obj/assert_driver.o src/assert_driver.c
cc -I . -I lib -Wall -Wextra -O3 -c -o obj/main.o src/main.c
cc -I . -I lib -Wall -Wextra -O3 -o prog.out obj/log.c/log.o obj/hello.o obj/my_
math.o obj/my_assert.o obj/assert_driver.o obj/main.o -lm
-=- target prog.out has completed -=-

cc -I . -I lib -DTEST -Wall -Wextra -c -o obj/munit/munit.o lib/munit/munit.c
cc -I . -I lib -DTEST -Wall -Wextra -c -o obj/testlib.o test/testlib.c
cc -I . -I lib -DTEST -Wall -Wextra -c -o obj/testmain.o test/testmain.c
cc -I . -I lib -DTEST -Wall -Wextra -o test.out obj/munit/munit.o obj/log.c/log.
o obj/hello.o obj/my_math.o obj/my_assert.o obj/assert_driver.o obj/testlib.o ob
j/testmain.o -lm
-=- target test.out has completed -=-
```

# ‘MAKE ALL’ 2.
# 단위 테스트 실행 (RUN-TEST)

```
./test.out
01:54:13 DEBUG test/testmain.c:9: Running tests...
Running test suite with seed 0xb3968281...
/example/compare                      [ OK    ] [ 0.00001320 / 0.00000000 CPU ]
/example/rand                         [ OK    ] [ 0.00000890 / 0.00000000 CPU ]
/example/parameters
  foo=one, bar=red                    [ OK    ] [ 0.00000730 / 0.00000000 CPU ]
  foo=one, bar=green                  [ OK    ] [ 0.00000630 / 0.00000000 CPU ]
  foo=one, bar=blue                   [ OK    ] [ 0.00000630 / 0.00000000 CPU ]
  foo=two, bar=red                    [ OK    ] [ 0.00000940 / 0.00000000 CPU ]
  foo=two, bar=green                  [ OK    ] [ 0.00000630 / 0.00000000 CPU ]
  foo=two, bar=blue                   [ OK    ] [ 0.00000590 / 0.00000000 CPU ]
  foo=three, bar=red                  [ OK    ] [ 0.00000600 / 0.00000000 CPU ]
  foo=three, bar=green                [ OK    ] [ 0.00000780 / 0.00000000 CPU ]
  foo=three, bar=blue                 [ OK    ] [ 0.00000940 / 0.00000000 CPU ]
11 of 11 (100%) tests successful, 0 (0%) test skipped.
-=- target run-test has completed -=-
```

```
tar -cvf dist/product.2019-10-08T01h54m12s.tar ./prog.out
./prog.out
-=- target dist/product.2019-10-08T01h54m12s.tar has completed -=-

tar -cvf dist/src.2019-10-08T01h54m12s.tar Makefile inc src test
Makefile
inc/
inc/assert_driver.h
inc/boolean.h
inc/hello.h
inc/my_assert.h
inc/my_math.h
src/
src/assert_driver.c
src/hello.c
src/main.c
src/my_assert.c
src/my_math.c
test/
test/testlib.c
test/testlib.h
test/testmain.c
-=- target dist/src.2019-10-08T01h54m12s.tar has completed -=-
```

# ‘MAKE ALL’ 3. TAR 패키징 (DIST)

```
kos@DESKTOP-S31FFVV ~/w4/assign
❯ tree --dirsfirst
.
├── dist
│   ├── product.2019-10-08T01h54m12s.tar
│   └── src.2019-10-08T01h54m12s.tar
├── inc
│   ├── assert_driver.h
│   ├── boolean.h
│   ├── hello.h
│   ├── my_assert.h
│   └── my_math.h
├── lib
│   ├── log.c
│   │   ├── src
│   │   │   ├── log.c
│   │   │   └── log.h
│   │   ├── LICENSE
│   │   └── README.md
│   └── munit
│       ├── COPYING
│       ├── Makefile
│       ├── README.md
│       ├── example.c
│       ├── meson.build
│       ├── munit.c
│       └── munit.h
├── obj
│   ├── log.c
│   │   └── log.o
│   ├── munit
│   │   └── munit.o
│   ├── assert_driver.o
│   ├── hello.o
│   ├── main.o
│   ├── my_assert.o
│   ├── my_math.o
│   ├── testlib.o
│   └── testmain.o
├── src
│   ├── assert_driver.c
│   ├── hello.c
│   ├── main.c
│   ├── my_assert.c
│   └── my_math.c
├── test
│   ├── testlib.c
│   ├── testlib.h
│   └── testmain.c
├── Makefile
├── prog.out
└── test.out
```

# ‘MAKE ALL’ 4. 수행 후 디렉토리 구조

- ‘dist’ 디렉토리
  - 패키징 파일
- ‘obj’ 디렉토리
  - 목적 코드 파일
- ‘prog.out’
  - 실행 가능 프로그램
- ‘test.out’
  - 단위 테스트 실행기

```
 kos@DESKTOP-S31FFVV  ~/w4/assign
❯ ./prog.out
02:17:02 INFO  src/main.c:14: version 1.0
02:17:02 TRACE src/main.c:15: in main():
02:17:02 TRACE src/hello.c:8: in say_hello():
Hello, world!
ipow(2, 2)=4
iabs(-10)=10
02:17:02 TRACE src/my_assert.c:9: in my_assert():
idivide(5, 1)=5
02:17:02 TRACE src/main.c:26: main() has ended.
```

```
 kos@DESKTOP-S31FFVV  ~/w4/assign
❯ ./prog.out 0
02:18:02 INFO  src/main.c:14: version 1.0
02:18:02 TRACE src/main.c:15: in main():
02:18:02 TRACE src/hello.c:8: in say_hello():
Hello, world!
ipow(2, 2)=4
iabs(-10)=10
02:18:02 TRACE src/my_assert.c:9: in my_assert():
02:18:02 TRACE src/my_assert.c:11: Assertion has failed.
Divide by zero.
```

# ‘PROG.OUT’ 실행 결과

```
kos@DESKTOP-S31FFVV ~/w4/assign
make TEST_OPT=FAIL
: -I . -I lib -DLOG_USE_COLOR -Wall -Wextra -O3 -c -o obj/log.c/log.o lib/log.c
src/log.c
: -I . -I lib -Wall -Wextra -O3 -c -o obj/hello.o src/hello.c
: -I . -I lib -Wall -Wextra -O3 -c -o obj/my_math.o src/my_math.c
: -I . -I lib -Wall -Wextra -O3 -c -o obj/my_assert.o src/my_assert.c
: -I . -I lib -Wall -Wextra -O3 -c -o obj/assert_driver.o src/assert_driver.c
: -I . -I lib -Wall -Wextra -O3 -c -o obj/main.o src/main.c
: -I . -I lib -Wall -Wextra -O3 -o prog.out obj/log.c/log.o obj/hello.o obj/my_
ath.o obj/my_assert.o obj/assert_driver.o obj/main.o -lm
=- target prog.out has completed -=-

: -I . -I lib -DTESTFAIL -Wall -Wextra -c -o obj/munit/munit.o lib/munit/munit.

: -I . -I lib -DTESTFAIL -Wall -Wextra -c -o obj/testlib.o test/testlib.c
est/testlib.c: In function ‘test_fail’:
est/testlib.c:22:51: warning: unused parameter ‘params’ [-Wunused-parameter]
static MunitResult test_fail(const MunitParameter params[], void *data) {
                                                  ^~~~~~
est/testlib.c:22:67: warning: unused parameter ‘data’ [-Wunused-parameter]
static MunitResult test_fail(const MunitParameter params[], void *data) {
                                                                  ^~~~
: -I . -I lib -DTESTFAIL -Wall -Wextra -c -o obj/testmain.o test/testmain.c
: -I . -I lib -DTESTFAIL -Wall -Wextra -o test.out obj/munit/munit.o obj/log.c/
g.o obj/hello.o obj/my_math.o obj/my_assert.o obj/assert_driver.o obj/testlib.
obj/testmain.o -lm
=- target test.out has completed -=-

/test.out
2:35:11 WARN  test/testmain.c:7: TESTFAIL has set -- testsuite will always fail
.
unning test suite with seed 0x64a012b0...
xample/compare                    [ OK    ] [ 0.00000610 / 0.00000000 CPU ]
xample/rand                       [ OK    ] [ 0.00000900 / 0.00000000 CPU ]
xample/parameters
 foo=one, bar=red                 [ OK    ] [ 0.00000600 / 0.00000000 CPU ]
 foo=one, bar=green               [ OK    ] [ 0.00000570 / 0.00000000 CPU ]
 foo=one, bar=blue                [ OK    ] [ 0.00000630 / 0.00000000 CPU ]
 foo=two, bar=red                 [ OK    ] [ 0.00000580 / 0.00000000 CPU ]
 foo=two, bar=green               [ OK    ] [ 0.00000600 / 0.00000000 CPU ]
 foo=two, bar=blue                [ OK    ] [ 0.00000580 / 0.00000000 CPU ]
 foo=three, bar=red               [ OK    ] [ 0.00000630 / 0.00000000 CPU ]
 foo=three, bar=green             [ OK    ] [ 0.00000590 / 0.00000000 CPU ]
 foo=three, bar=blue              [ OK    ] [ 0.00000600 / 0.00000000 CPU ]
est/always_fail                   [ FAIL  ]
 of 12 (92%) tests successful, 0 (0%) test skipped.
akefile:69: recipe for target 'run-test' failed
ake: *** [run-test] Error 1
kos@DESKTOP-S31FFVV ~/w4/assign
```

# 빌드 중 단위 테스트가 실패한 경우

- “단위 테스트 실패 시 패키징 작업을 수행하지 않는다.”
- ‘program.out’ 및 ‘test.out’ 까지는 빌드
  - ‘release’ -> ‘run-test’ -> ‘dist’ 순으로 빌드 되므로
- 실패가 하나라도 있을 경우, 이후 타겟은 모두 취소
  - “Makefile:69: recipe for target 'run-test' failed”
  - `int main()` 함수에서 0이 아닌 값을 반환할 경우, make에서 해당 타겟이 실패했다고 판단.
    - 이러한 이유 때문에 ‘void main()’은 사용하면 안 됨

```
 kos@DESKTOP-S31FFVV  ~/w4/assign
❯ make debug
cc -I . -I lib -DDEBUG -DLOG_USE_COLOR -Wall -Wextra -c -o obj/log.c/log.o lib/l
og.c/src/log.c
cc -I . -I lib -DDEBUG -Wall -Wextra -c -o obj/hello.o src/hello.c
cc -I . -I lib -DDEBUG -Wall -Wextra -c -o obj/my_math.o src/my_math.c
cc -I . -I lib -DDEBUG -Wall -Wextra -c -o obj/my_assert.o src/my_assert.c
cc -I . -I lib -DDEBUG -Wall -Wextra -c -o obj/assert_driver.o src/assert_driver
.c
cc -I . -I lib -DDEBUG -Wall -Wextra -c -o obj/main.o src/main.c
cc -I . -I lib -DDEBUG -Wall -Wextra -o prog.out obj/log.c/log.o obj/hello.o obj
/my_math.o obj/my_assert.o obj/assert_driver.o obj/main.o -lm
-=- target prog.out has completed -=-

cc -I . -I lib -DDEBUG -DTEST -Wall -Wextra -c -o obj/munit/munit.o lib/munit/mu
nit.c
cc -I . -I lib -DDEBUG -DTEST -Wall -Wextra -c -o obj/testlib.o test/testlib.c
cc -I . -I lib -DDEBUG -DTEST -Wall -Wextra -c -o obj/testmain.o test/testmain.c

cc -I . -I lib -DDEBUG -DTEST -Wall -Wextra -o test.out obj/munit/munit.o obj/lo
g.c/log.o obj/hello.o obj/my_math.o obj/my_assert.o obj/assert_driver.o obj/test
lib.o obj/testmain.o -lm
-=- target test.out has completed -=-

-=- target debug has completed -=-
```

# ‘MAKE DEBUG’
# : 디버그 빌드 수행

```
kos@DESKTOP-S31FFVV ~/w4/assign
❯ ./prog.out
02:31:52 DEBUG src/main.c:11: Debug build.
02:31:52 INFO  src/main.c:14: version 1.0
02:31:52 TRACE src/main.c:15: in main():
02:31:52 TRACE src/hello.c:8: in say_hello():
Hello, world!
ipow(2, 2)=4
iabs(-10)=10
02:31:52 TRACE src/my_assert.c:9: in my_assert():
idivide(5, 1)=5
02:31:52 TRACE src/main.c:26: main() has ended.
```

# DEBUG 타겟으로 빌드한 'PROG.OUT' 실행

END