Makefiles

and

Testing

Open Eclipse

- Open CS 300 example workspace
- Close All Projects
- Open MakeFileTesting

Makefiles

- Script that will build your code!
- Useful if you build your code twice!
- GNU Make

  make -h

https://www.gnu.org/software/make/manual/html_node/Automatic-Variables.html

Makefile Rules

```
What_To_Build: What_Is_Needed_To_Build
    How_To_Build

runMe: runMe.c
    gcc -o runMe -g -Wall runMe.c
```

Makefile

```
target: dependency1 dependency2
    command1
    command2
```

tab!

Given a set of dependencies, make will only run the necessary commands to build the project. Build a **dependency graph**.

If a target is older than any of its dependencies the commands are run to build the target

target and dependencies are files

Command line

zeus$> make tree

- looks for target named *tree* in Makefile and checks to see if it needs to be built

zeus$> make

- looks for the first target in Makefile and checks to see if it needs to be built
  - by convention, this target is named **all:**

Makefile

```
CC=gcc
CFLAGS=-g -Wall

# -g  include debug symbols in the executable so that the code can be
#     run through the debugger effectively
#
# -Wall show all warnings from gcc


.PHONY: clean all


TARGETS=

all:

```

Rational

- Close all projects

- Open Rational

Makefile

- Variables
  - Makefiles can have variables such as CC and CCFLAGS

- Targets
  - By default, Makefile targets are “file targets” used to create other files

- .PHONY
  - Declares targets that do not represent physical files
  - target that is always out-of-date, thus, will always run when asked
  - e.g. make clean

Makefile

```
CC = gcc
CFLAGS = -g -Wall
RATIONAL_OBJECTS = bin/rationalDriver.o bin/rational.o
REDUCE_OBJECTS = bin/reduceRational.o bin/rational.o
ALL_OBJECTS = ${RATIONAL_OBJECTS} ${REDUCE_OBJECTS}

.PHONY: all clean valgrind tarball

all: bin/rationalDriver bin/reduceRational

bin/rationalDriver: ${RATIONAL_OBJECTS}
  ${CC} ${CFLAGS} -o bin/rationalDriver ${RATIONAL_OBJECTS}
```

```
clean:
  rm -f bin/rationalDriver  ${ALL_OBJECTS}

valgrindRational: bin/rationalDriver
  valgrind -v --leak-check=yes bin/rationalDriver

tarball: clean
  tar czf ../puNetIdRational.tar.gz ../Rational

```

Dependency Graph

Testing with Asserts

- rationalDriver.c

  success()

  failure()

  assert()

- You **MUST** reuse these functions to build your test drivers!

```
static void success (char *pszStr)
{
  printf ("SUCCESS: %s\n", pszStr);
}


static void failure (char *pszStr)
{
  printf ("FAILURE: %s\n", pszStr);
}


static void assert (bool bExpression, char *pTrue, char *pFalse)
{
  if (bExpression)
  {
    success (pTrue);
  }
  else
  {
    failure (pFalse);
  }
}
```

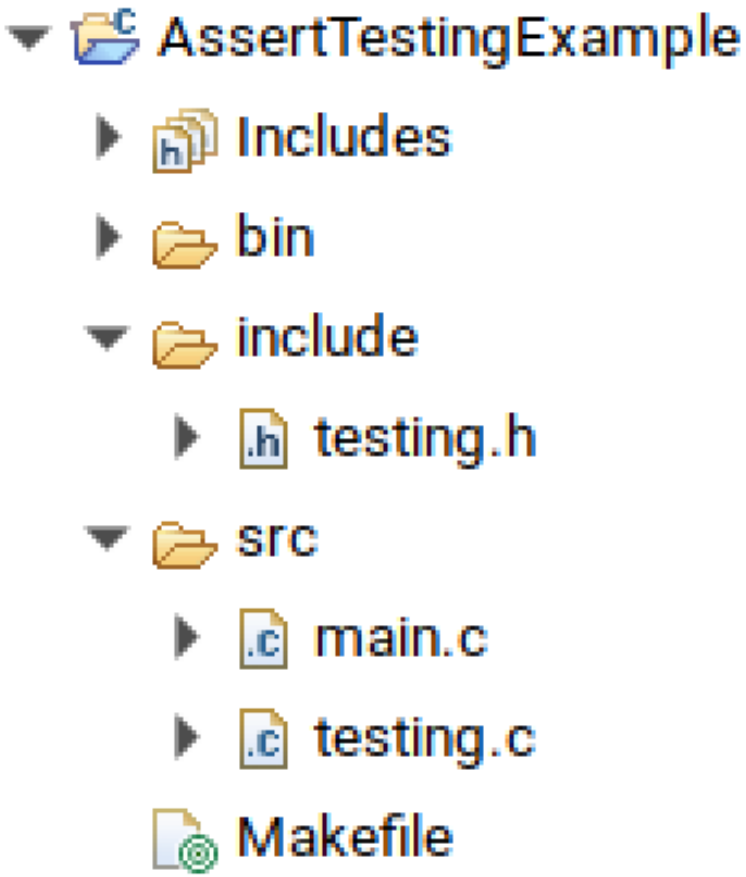
```
typedef struct Rational
{
  int numerator;
  int denominator;
} Rational;

extern void            loadErrorMessages ();
extern void            setRational (Rational *psRational,
                                    int numerator, int denominator);

extern void            getRational (Rational *psRational);

extern void            printRational (const Rational *psRational);

extern bool            isEqualRational (const Rational *psRational1,
                                        const Rational *psRational2);

extern Rational        multiplyRational (const Rational *psRational1,
                                         const Rational *psRational2);

extern Rational        divideRational (const Rational *psRational1,
                                       const Rational *psRational2);

extern Rational        addRational (const Rational *psRational1,
                                    const Rational *psRational2);

extern Rational        reduceRational (const Rational *psRational);
```

Testing!

Practice

- Separate out success(), failure(), and assert() to testing.c/testing.h

Example Project



Makefile

```
all: bin/testingExample

bin/testingExample:
```