Review – Makefiles

Makefiles

- Purpose
    - Compile an executable or executables that consist of multiple source files together
- Method
    - Using UNIX commands, build the executable or even build + install the executable
    - Think of a makefile as any sequence of UNIX commands
    - In our case, sequence of gcc commands
- Structure
    - Collection of rules
    - What is a rule?

      ```
      Target : Dependency1 Dependency2 ... DependencyN
                 Command 1
                 Command 2
      ```

      The target or name of the rule is what we want to produce. Each dependency is a file that goes into the sequence of commands to produce the target. All commands are one tab in from the left.

    - Example:

      Suppose we want to build our file (main.c) into an object file.

      ```
      main.o : main.cc
                 gcc –c main.c
      ```

    - What did the above command do?

      main.o is what we want to produce
      main.c is a dependency, we cannot do the target unless it exists
      gcc –c main.c creates an object output file. The –c option for gcc creates an object file but does not link it.

    - Example 2:

      Tie it together to create a full executable out of the result.

      ```
      main.o : main.cc
                 gcc –c main.c

      Lab1 : main.o
                 gcc –o Lab1 main.o
      ```

    - What did that do?

      *Lab*1 is the executable we were going to create.
      Lab1 required main.o in order to create that target.

gcc –o Lab1 main.o compiles/links main.o and creates an executable named Lab1. The –o flag sets where gcc will output its end result which is typically used in only the last step (creation of the executable).

- What will the different commands do?

  ```
  make main.o
  ```

  This will simply create main.o

  ```
  make Lab1
  ```

  This will create Lab1 and if necessary, also create main.o in order to create Lab1.

- What else could we do?

  ```
  clean:
             rm *.o
  ```

  Clean up all object files and force a rebuild of everything.

  ```
  install: Lab1
            cp Lab1 ~/.
            more README
  ```

  Copy the Lab1 executable to the main source directory and then show the README file using more. Keep in mind that a makefile can execute any command we can execute at the command prompt. In other words, we could compile a Java file, set the time/date, change file permissions, list directories, etc.

- The default that occurs without an argument is the first item in the list, e.g. when you just type `make` without an argument.

More Complicated Makefiles

- Using variables

  ```
  MYVAR = Whatever text I want
  # Comments in the makefile
  ```

- Example

  ```
  # Select gcc or g++ as the compiler
  COMPILER = gcc

  # Options for compiling C files to objects
  #  -Wall pays attention to all warnings and –c insures we create
  an object file
  COMPOPT = -Wall

  # Actual compile line
  COMPILE = $(COMPILER) $(COMPOPT)
  ```

```
# Object files
OBJS = main.o test.o

# Output program
EXE = Lab1

$(EXE) : $(OBJS)
     $(COMPILER) -o $(EXE) $(OBJS)

main.o : main.cc
     $(COMPILE) main.c

clean:
     rm *.o
```

For more info, see the tutorial at:

- http://www.cs.washington.edu/orgs/acm/tutorials/dev-in-unix/makefiles.html