By: Ehsan Kourkchi (Sep., 5, 2013)

Makefile

A program might consist of a lot of subroutines and functions which are related to each other and are compiled together to form a solid executable package. The functions and sub-routines as the program elements might updated several times. Each function can be stored as a single source file which can be used by multiple programs. Therefore, tracking all the changes and recompiling all functions and libraries in a package can be so complicated and time consuming. If one is going to compile all programs manually, every compilation command must be rewritten several times to compile all routines and stick them together.

In addition, the source codes might be needed to completely recompile on different machines to get the consistent executable file. Different machines can have different architecture and arrangements. Even the name of compilers and the compilation flags might be different. Thus, one need to have a way to keep tracking all the compilation procedure and even program it.

Makefile is a script which contains all the compilations commands. It basically compiles all the source codes in a consistent way to reach the final product. Within this script, one can also takes the advantage of variables to generalize the code. Therefore, any change of the compiler name, compiler flags, directory set-up and adding or removing the files can be handled in a systematic way. Also any missing compilation prerequisite can be easily tracked.

Mamkefile is usually used to compile the source codes and produce the final executable file(s) as the main product. One can simply run makefile by typing "make" in the command line where the file with the name of "`makefile`" exists. If the name of the makfile script is different, it can be run by "`make -f <my_makefile>`". Each makefile have several simple units known as rules.

Most often each rule consist of two lines. The first line is the target file followed by the prerequisite files needed to produce the target file. The second line starts with a tab followed by a compilation command. This command uses the prerequisites files to make the target file. If the prerequisite files do not exist, then there should be other units that are responsible to make them, otherwise the make process is terminated by an error message. See below for the most general form of each makefile unit.

```
Target: prerequisite(s)
     compilation commands
```

The unit which make the final product is the most important unit and its target name is "all". For example:

```
all: file1.o file2.o file3.o
     cc file1.o file2.o file3.o -o final_file.x
```

In this example, file*.o are the object files which are needed to form the `final_file.x` that is executable. This above unit must be followed by the other units to produce the object files. See the following example of the units which create the object files.

```
file1.o: file1.c
     cc -c file1.c -o file1.o
```

By: Ehsan Kourkchi (Sep., 5, 2013)

```
file2.o: file2.c
      cc -c file2.c -o file2.o

file3.o: file3.c
      cc -c file3.c -o file3.o
```

As a practical example, the first ICS212 assignment is used to demonstrate how the makefile works. The source code is stored in a single file which includes all the necessary functions. Therefore to compile it, the makefile can be written as following:

```
all: Assignment_01.c
      cc -o Assignment_01.x Assignment_01.c
```

To make it more complicated, let's assume that each function is stored in an individual file with the same name. So in this case, we have `Assignment_01.c` which only contains the main function. The other files just contain the needed functions which are called by `Assignment_01.c`. So we have the following files:

```
Assignment_01.c
GetTemperature.c
RoundNumber.c
convertFtoC.c
PrintTable.c
```

The simple make file to generate the executable file can be written as following:

```
all: Assignment_01.o GetTemperature.o RoundNumber.o PrintTable.o
      cc Assignment_01.o GetTemperature.o RoundNumber.o PrintTable.o -o
Assignment_01.x

Assignment_01.o:  Assignment_01.c
      cc -c Assignment_01.c

GetTemperature.o:  GetTemperature.c
      cc -c  GetTemperature.c

RoundNumber.o:  RoundNumber.c
      cc -c  RoundNumber.c

PrintTable.o: PrintTable.c
      cc -c PrintTable.c

clear:
      rm -f *.o
```

The first unit (all), compile everything together to form the final executable file, i.e Assignment_01.x. The last unit removes all the created ".o" object files.