CPE 212 - Fundamentals of Software Engineering

Makefile Review

Objective:

Overview of the use of makefiles and how they make your life easier

Outline

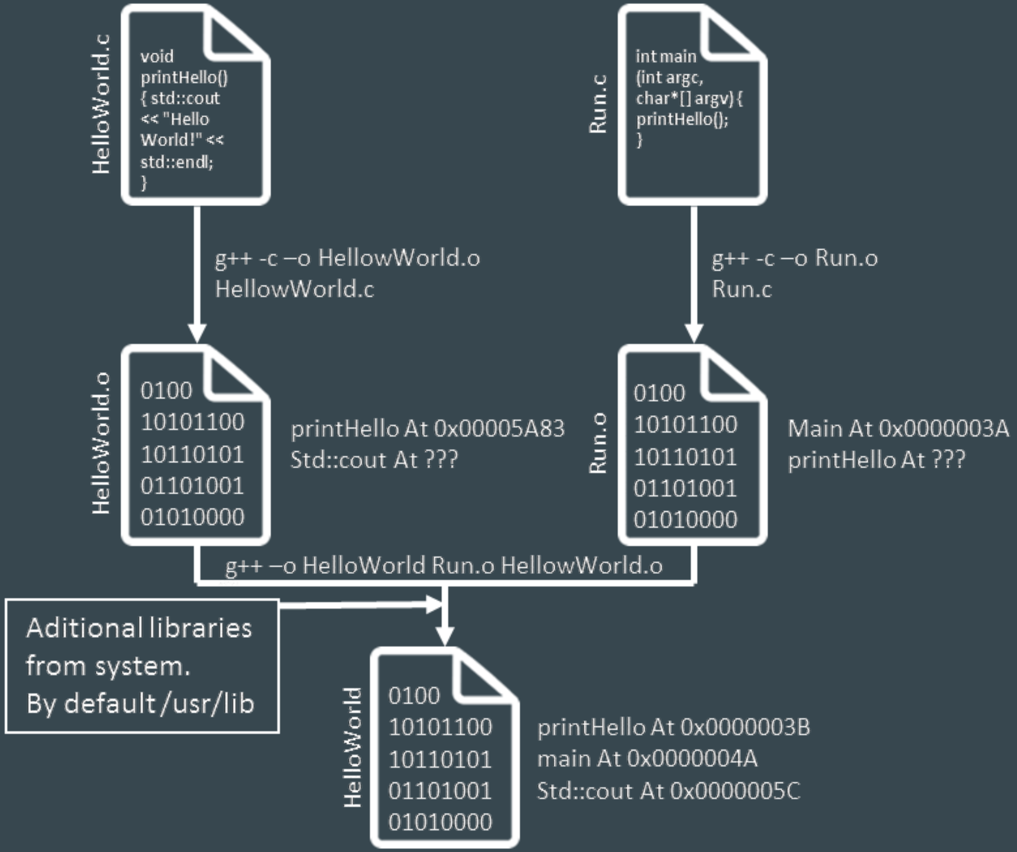
- What is make?
- What is a makefile?
- Compiling Multi-file Programs
- Advantages of Makefiles

make

- make is a utility program that can help you compile, link, and maintain your program in an efficient, repeatable manner
- make utilizes a text file that describes the process of building your program
    - The default name of the text file is makefile or Makefile
    - Other filenames may be used, but you will need to specify its name on the command line after -f

Compiling and Linking

- Suppose that one has a multi-file C++ source program that must be compiled on a Unix or Linux system
  - Run.c – contains the function main() which uses both the name and parents classes
  - HelloWorld.c – contains code for the class name



```
void
printHello()
{ std::cout
<< "Hello
World!" <<
std::endl;
}
```

```
int main
(int argc,
char*[] argv) {
printHello();
}
```

```
g++ -c –o HellowWorld.o
HellowWorld.c
```

```
g++ -c –o Run.o
Run.c
```

```
g++ –o HelloWorld Run.o HellowWorld.o
```

Command Line Compiling

- Where is the problem?

```
bash $  ls
main.cpp        name.cpp     name.h       parents.cpp
parents.h
bash $  g++  -c main.cpp
bash $  g++  -c name.cpp
bash $  g++  -c parents.cpp
bash $  g++  main.o name.o parents.o  -o main
bash $  ls
main           main.cpp        main.o          makefile
name.cpp       name.h          name.o          parents.cpp
parents.h      parents.o
bash $  rm  *.o
bash $  ls
main            main.cpp     makefile     name.cpp     name.h
parents.cpp     parents.h
bash $  vi  parents.cpp
bash $  vi  name.cpp
bash $  g++  -c name.cpp
bash $  g++  main.o name.o parents.o  -o main
bash $
```

Simple Example

- The makefile contains targets, dependencies, and commands listed in the following format

  target: dependencies

  TAB   commands

- TAB is important
- Comments begin with #

```
# Sample makefile utilizing Sun's CC compiler
# Build executable  main
main: main.o  name.o  parents.o
       g++  main.o name.o parents.o  -o main

# Build object file  name.o
name.o: name.h  name.cpp
       g++  -c name.cpp

# Build object file  parents.o
parents.o: name.h  parents.h  parents.cpp
       g++  -c parents.cpp

# Build object file  main.o
main.o: main.cpp  parents.h  name.h
       g++  -c main.cpp

# Clean up by deleting intermediate object files
clean:
       rm  *.o
```

List of dependencies

TAB is followed by UNIX or Linux commands that you would normally input manually in the command line

Remove any file in the current directory (*) with an .o extension

Command Line Compiling

```
bash $  ls
main.cpp     makefile     name.cpp     name.h        parents.cpp
parents.h
bash $  make
g++  -c main.cpp
g++  -c name.cpp
g++  -c parents.cpp
g++  main.o name.o parents.o  -o main
bash $  ls
main           main.cpp       main.o              makefile
name.cpp
name.h       name.o          parents.cpp     parents.h
parents.o
bash $  make  clean
rm  *.o
bash $  ls
main     main.cpp     makefile     name.cpp     name.h
parents.cpp     parents.h
bash $  make
g++  -c main.cpp
g++  -c name.cpp
g++  -c parents.cpp
g++  main.o name.o parents.o  -o main
bash $  vi  name.cpp
bash $ make
g++  -c name.cpp
g++  main.o name.o parents.o  -o main
```

Echo prints each command executed

Updating the name.cpp file

Recompiles only name.o

Simple Real-World Example

- Real world projects consists of dozens or even hundreds of separate source files
  - Recompiling and relinking all files upon each source file modification impractical
- Failure to recompile a modified source file and relink its object file wastes time and money
- Manually tracking which source files have been modified, and thus need to be recompiled, is cumbersome and error prone

```
VERSION?=0.1
CPPFLAGS:= -Wall -Wextra
CPPFLAGS+= -DVERSIONMACRO='"$(VERSION)"'

SOURCES:= $(shell find source/ -name "*.cpp")
SOURCES+= main.cpp

HEADERS:= $(shell find includes/ -name "*.h")

OBJECTS=$(SOURCES:.cpp=.o)

all: $(OBJECTS)
	g++ $(CPPFLAGS) $(OBJECTS) $(HEADERS) -o main_$(VERSION)

.PHONY: all clean deploy distclean allclean

clean:
	-rm $(OBJECTS)

deploy:
	-mkdir deploy
	-cp app_readme.txt deploy
	-cp main deploy
	-tar -zcvf deploy.tar.gz deploy

distclean:
	-rm -r deploy
	-rm deploy.tar.gz

allclean: clean distclean
```

Subtle make Error

- Your makefile appears to be correct but you receive a “missing separator” error when you execute make
- Some editors substitute blank spaces for the TAB character
- Check the editor preferences to turn off this feature