UCLA CS35L

Week 4

Monday

Reminders

- Assignment 3 homework is due this Friday (4/24)
- Assignment 4 homework is due next Friday (5/1)

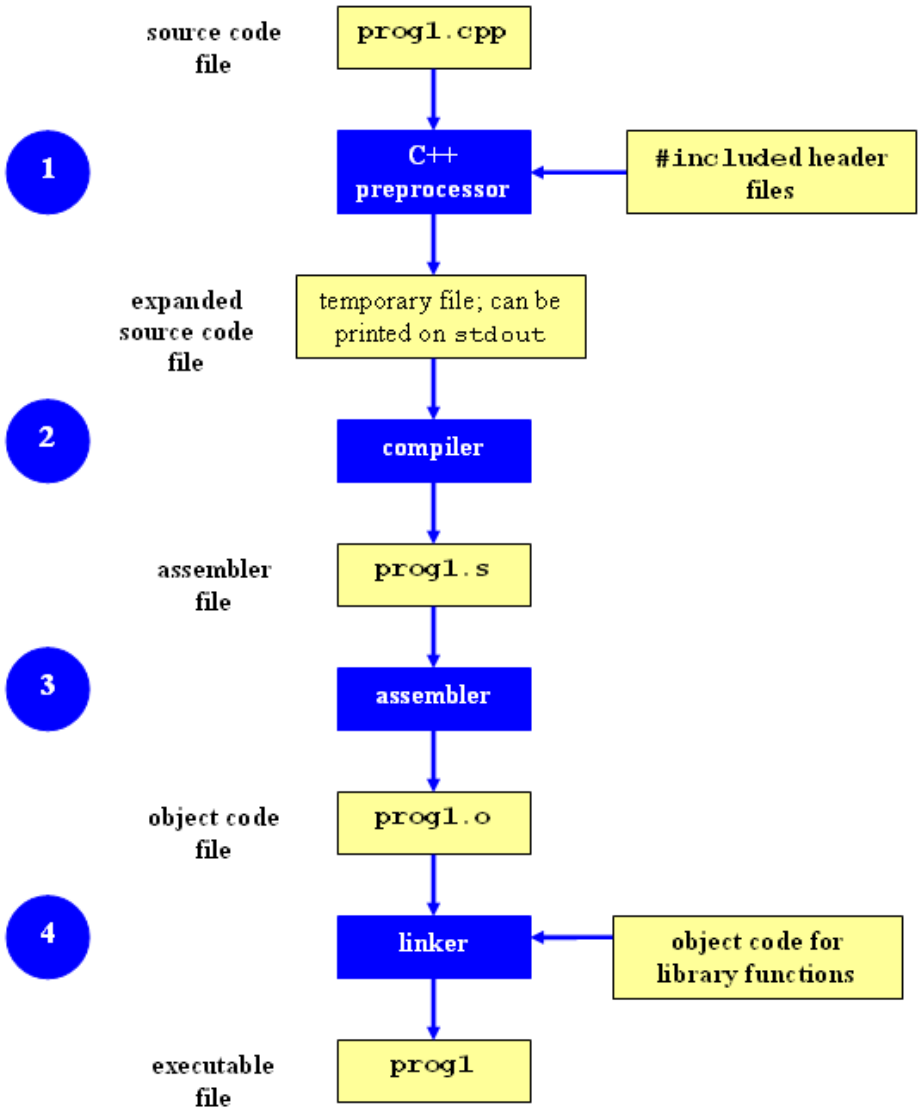
- Anonymous feedback for Daniel - https://forms.gle/tZwuMbALe825DBVn8

Compilation Tools and Make

Compilation Process

g++ prog1.cpp -o prog1

1. **Preprocessor** copies headers, expands macros, and replaces #define constants
2. Source code **compiled** to assembly
3. Assembler code is **assembled** into object code
4. Object code is **linked** with other object code files for library functions, producing the final executable



Compiling with Multiple Files

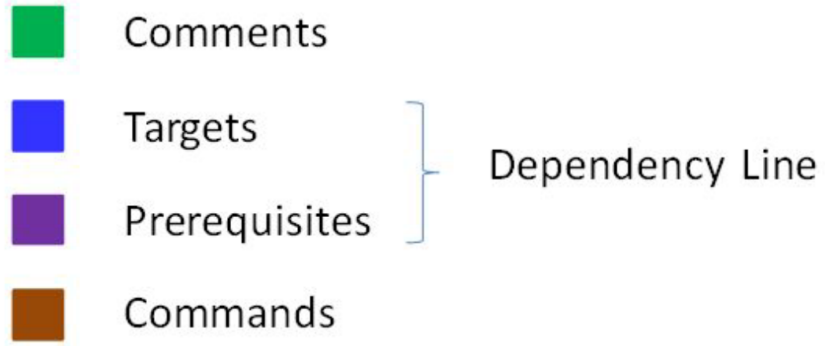
- More complex applications require multiple files to be compiled
- Example: “Car” Program
  - Car.cpp (`#include car.h, engine.h, sensor.h, navigation.h`)
  - Navigation.cpp (`#include navigation.h, sensor.h`)
  - Engine.cpp (`#include engine.h`)
  - Sensor.cpp (`#include sensor.h`)

- How can we compile them together?
  - `g++ car.cpp navigation.cpp engine.cpp sensor.cpp -o car`
  - What happens if we update a single file’s source code?

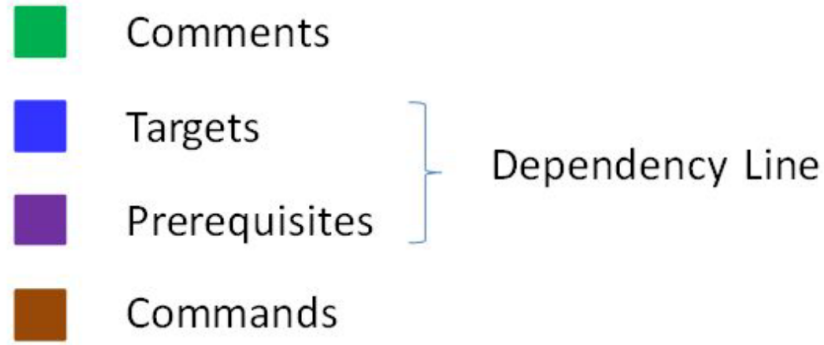
Quick Intro to Makefile

- A makefile is a build tool for C/C++
- Allows you to specify compiler targets/commands and dependencies. Will then build objects as necessary and compiles only what is needed based on which files have been updated.

```
# Makefile - Example 1

all: car.cpp engine.cpp engine.h navigation.cpp navigation.h sensor.cpp sensor.h
     g++ -g -Wall -o car car.cpp engine.cpp navigation.cpp sensor.cpp

clean:
     rm -f car
```



```
# Makefile - Example 2

all: car

car: car.cpp engine.o sensor.o navigation.o
     g++ -g -Wall -o car car.cpp engine.o sensor.o navigation.o

engine.o: engine.cpp engine.h
     g++ -g -Wall -c engine.cpp

sensor.o: sensor.cpp sensor.h
     g++ -g -Wall -c sensor.cpp

navigation.o: navigation.cpp navigation.h sensor.h
     g++ -g -Wall -c navigation.cpp

clean:
     rm -f engine.o sensor.o navigation.o car
```



```
CC=g++
CFLAGS=-Wall -std=c++17 -O3

all: car

car: car.cpp engine.o sensor.o navigation.o
     $(CC) $(CFLAGS) -o car car.cpp engine.o sensor.o navigation.o

engine.o: engine.cpp engine.h
     $(CC) $(CFLAGS) -c engine.cpp

sensor.o: sensor.cpp sensor.h
     $(CC) $(CFLAGS) -c sensor.cpp

navigation.o: navigation.cpp navigation.h sensor.h
     $(CC) $(CFLAGS) -c navigation.cpp

clean:
     rm -f engine.o sensor.o navigation.o car
```

Standard Make commands

- When you have a file named just “makefile” then you can access it via Shell commands
  - Make – compile the default target
  - Make all – should compile everything
  - Make install – should install things in the right place
  - Make clean – should clean things up
  - Make [target] – call commands for just that group

Typical Make Process

- ./configure
- make
- make install

./configure

- A good read https://thoughtbot.com/blog/the-magic-behind-configure-make-make-install
- The **configure** script checks for dependencies for building the program, auto-generated from **configure.ac** and creates the **Makefile**.
- It also takes in options such as **--prefix=/some/path** which specifies that the final executable should be installed into /some/path instead of the default path.
- For example, without --prefix, the default path might be **/usr/local/** which one may not have permission to write to. Instead we can say **./configure --prefix=my_directory** which will install the executable into **my_directory** when you run **make install**

make

- After running **./configure** a **Makefile** will be generated (Note that one can write Makefile by hand as well).
- **make** will build the executables as specified by the **Makefile**.

Make install

- **make install** merely copies the built programs generated by **make** to a target install location.
- If **--prefix=SOME_PATH** was provided in the **./configure** step, the built programs will be copied to **SOME_PATH**

So then for lab…

```
./configure --prefix=someSubDir/youChose
make
make install
```

Patching

- What is patching?
- There is also a Patch command. It applies a **diff** file that includes changes made to a file
- Check man patch for usage options
  - `patch –p[num] < patchFile`
  - NOTE – you need to specify p[num] which defaults to p1 if omitted

-p[num] example

- Diff files contain a path to the directory, for example:
  - `diff --git a/NEWS b/NEWS`
- You may need to strip off parts of the directory that don’t match your system.
  - I don’t need the a/
- The num in –p[num] option, indicates how many leading directories to strip off
  - -p1 will strip off a/ and b/

diff Unified Format

- Good overview - https://en.wikipedia.org/wiki/Diff#Unified_format

- File characters
  - @@ -l,s +l,s @@
    - @@ for beginning and end of a hunk
    - –l: beginning line number
    - –s: number of lines the change hunk applies to for each file
  - Other line starting characters:
    - - : line was deleted from the original
    - + : line was added in the new file
    - ‘ ‘ stayed the same

Patch and diff Combined

- Either generate or receive a diff file
- Use patch command with –p[num] to apply the diff changes to your local copy

Start of Lab Walkthrough

- Use wget to download relevant files
  - wget ftp://ftp.gnu.org/gnu/coreutils/coreutils-8.29.tar.xz
  - wget ftp://ftp.gnu.org/gnu/coreutils/coreutils-8.29.tar.xz.sig
  - wget https://ftp.gnu.org/gnu/gnu-keyring.gpg

tar file

- A **tar** file is an archive, also called a tarball
- Common options:
  -x stands for extraction, i.e. extracting from an archive.
  -z filter through gzip, a compression format
  -J filter the archive through xz
  -f means the following argument is a filename
  -v verbose, i.e. print out more info

e.g.

- `tar -xJvf coreutils-8.29.tar.xz`

Compile CoreUtils

- Make a directory “lab4install” in your home directory
- Go to the coreutils-8.29 you just unzipped
- Review INSTALL file
  - How do we specify where to install to?
- Use ./configure, make, and make install
  - Note the prefix specification for ./configure
- Then to use the custom `ls`, you must specify its path otherwise you will end up using the system default `ls`
  - i.e. `~/lab4install/bin/ls someDir`