Makefiles

Data and File Structures Laboratory

http://www.isical.ac.in/~dfslab/2017/index.html

Compiling a program that uses your heap code

- Heap functions declared in `heap.h`; defined in `heap.c`
- Problem solution written in `program1.c`
  (contains `#include "heap.h"`)
- Option 1: no library created

```
$ gcc -o program1 program1.c heap.c
```

  OR

```
$ gcc -o program1 program1.c heap.o
```

- Option 2: static library created

```
$ gcc -c heap.c
$ ar ruvs libheap.a heap.o
$ gcc -o program1 program1.c -lheap
```

- Option 3: Makefiles

Compiling: things to remember

From scratch:

- Compilation commands, flags
- Dependencies: which files need to be recompiled in what sequence after modifications to a particular file (or files)

Using Makefiles:

- Specify a set of *targets*: things for `make` to do (e.g., executables to build)
- Dependencies and compilation instructions defined for each target in Makefile
- Command to run after *any* modification: `make` or `make <target>`
- `make` recompiles necessary object files / binaries

Makefile: example 1

Variable / macro definitions

Target

Dependencies

Compilation commands

```
CC=gcc

hello: main.o factorial.o hello.o
	$(CC) main.o factorial.o hello.o -o hello

main.o: main.c functions.h
	$(CC) -c main.c

factorial.o: factorial.c functions.h
	$(CC) -c factorial.c

hello.o: hello.c functions.h
	$(CC) -c hello.c
```

Command lines have to start with a hard TAB!

How make works

- Finds rule for specified `target` (or first target).
- If any of the dependencies are newer than the target, compilation commands are executed to rebuild the target.
  - If any dependencies have to be rebuilt, they are first rebuilt recursively.

Inbuilt rules

- `.o` files depend on corresponding `.c` file
- How to build `.o` file from `.c` file

Makefile: example 2

```
CC=gcc
OBJECTS = main.o hello.o factorial.o

# You can write comments to explain things to yourself
hello: $(OBJECTS)
	$(CC) $(OBJECTS) -o hello
main.o: functions.h
hello.o: functions.h
factorial.o: functions.h
```

PHONY (dummy) targets

- Useful to “just” run commands
- Not a file, ***BUT***
- `make` runs commands to “build” target

Example:

```
.PHONY: clean
clean:
        $(RM) *.o *~
```

Some common variables

`AR`	Archive-maintaining program (default: `ar`)
`CC`	Program for compiling C programs (default: `cc`)
`CXX`	Program for compiling C++ programs (default: `g++`)
`RM`	Command to remove a file (default: `rm -f`)

Some common flags

`ARFLAGS`	Flags to give the `ar` program (default: `rv`)
`CFLAGS`	Extra flags to give to the C compiler
`CXXFLAGS`	Extra flags to give to the C++ compiler
`LDFLAGS`	Extra flags to give to compilers when they are supposed to invoke the linker

Special symbols

`$@`	name of the target (usually, file to be compiled, LHS of a rule)
`$?`	names of the changed dependencies (all dependencies newer than target)
`$^`	names of *all* dependencies
`$<`	name of first dependency

More examples

Some annotated makefiles

- `www.sis.pitt.edu/mbsclass/tutorial/advanced/makefile/mainmake.htm`
- `www.sis.pitt.edu/mbsclass/tutorial/advanced/makefile/libmake.htm`
- `www.sis.pitt.edu/mbsclass/tutorial/advanced/makefile/cliemake.htm`