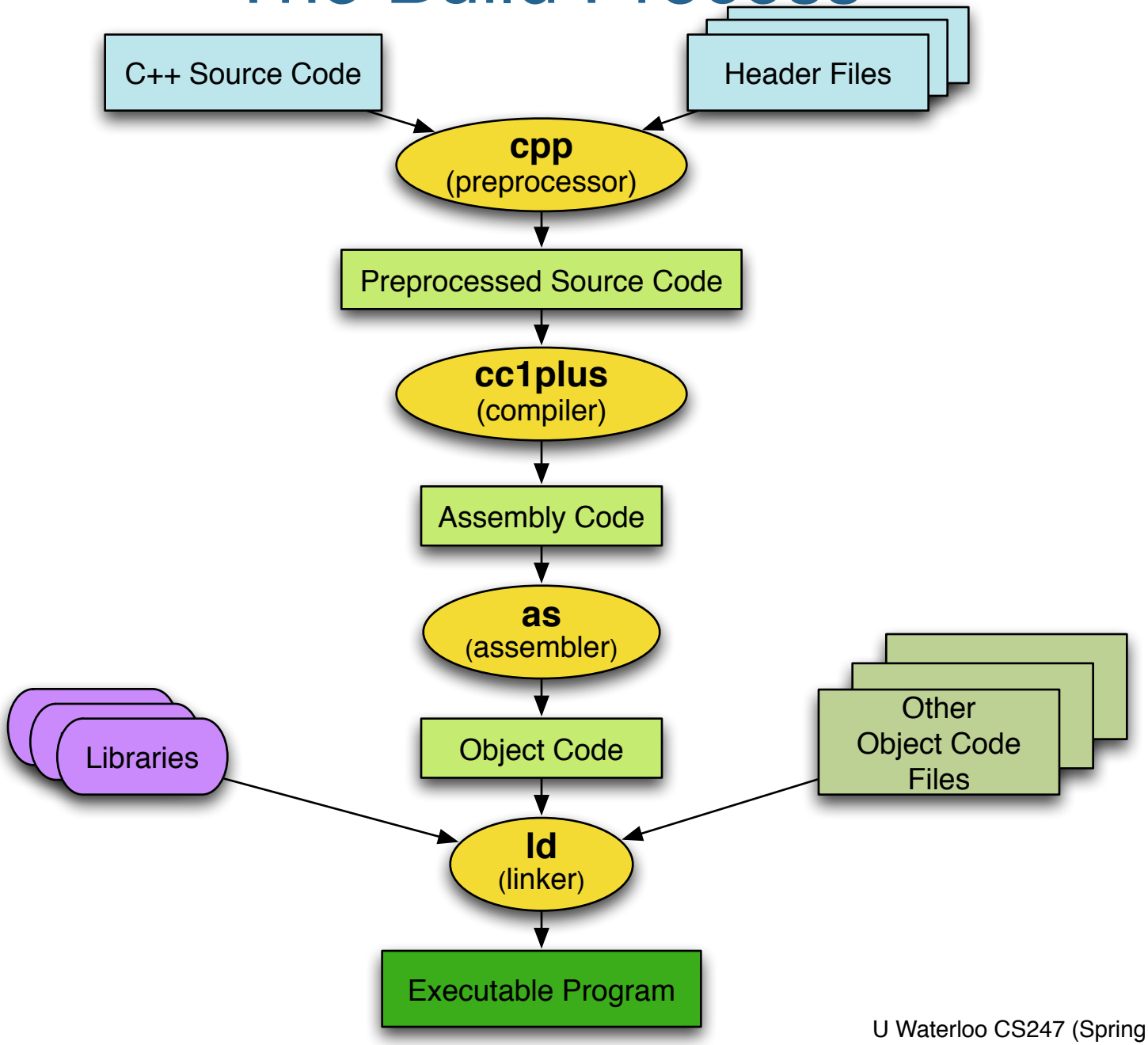
CS 247: Software Engineering Principles

Makefiles

Readings: Eckel, Vol. 1
Ch. 3 The C in C++: Make: Managing separate compilation

The Build Process



Automated Builds

Goal: Want a fully automated build system that

- is sure to incorporate all updated source files into executable
- is incremental and rebuilds only what has changed
- automatically derives the dependency relationships among files

Compilation Dependencies

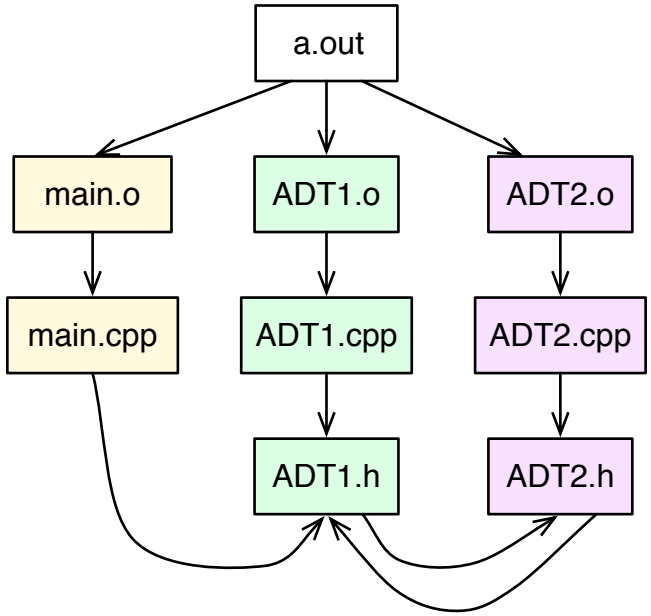
If a file F changes, need to recompile F *and all files that depend on F*

Example:

main.cpp: **#include** "ADT1.h"

ADT1.h: **#include** "ADT2.h"
ADT1.cpp: **#include** "ADT1.h"

ADT2.h: **class** ADT1
ADT2.cpp: **#include** "ADT2.h"



- the executable depends on `.o` files
- the `.o` files depend on `.cpp` files
- the `.cpp` files depend on `.h` files

Make

make is a UNIX command that incrementally builds products

(1) using instructions and file dependencies provided in a Makefile

(2) using file timestamps to decide which files to recompile/rebuild

Example Makefile:

```
# A comment
program.exe: main.o ADT1.o ADT2.o        # dependency graph
    g++ main.o ADT1.o ADT2.o -o program.exe  # build rule
```

tab

Makefile with Macros

Makefile more reusable if rules can be defined in terms of macros that are set at the top of the file.

```
CXX = g++                       # = specifies a new macro
CXXFLAGS = -g -Wall
OBJECTS = main.o ADT1.o ADT2.o
EXEC = program.exe

${EXEC} : ${OBJECTS}            # ${ } expand the macro
    ${CXX} ${CXXFLAGS} ${OBJECTS} -o ${EXEC}

main.o : main.cpp stack.h
    ${CXX} ${CXXFLAGS} -c main.cpp
```

Implicit Rules

gmake has implicit rules for processing files with specific suffixes and when special macros are used.

```
CXX = g++                       # variables and initialization
CXXFLAGS = -g -Wall
OBJECTS = main.o stack.o node.o
EXEC = program

${EXEC} : ${OBJECTS}            # default target
    ${CXX} ${CXXFLAGS} ${OBJECTS} -o ${EXEC}

# gmake knows how to build .o files given CXX and CXXFLAGS
# just need to list dependencies
main.o : main.cpp stack.h
stack.o : stack.cpp stack.h node.h
node.o : node.cpp node.h stack.h
```

Automatically Derive Dependencies

- g++ flag -MMD generates dependencies and outputs them in .d files

```
CXX = g++                       # variables and initialization
CXXFLAGS = -g -Wall -MMD # builds dependency lists in .d files
OBJECTS = main.o stack.o node.o
DEPENDS = ${OBJECTS:.o=.d} # substitute ".o" with ".d"
EXEC = program

${EXEC} : ${OBJECTS}
    ${CXX} ${CXXFLAGS} ${OBJECTS} -o ${EXEC}

clean :                         # separate target; cleans directory
    rm -rf ${DEPENDS} ${OBJECTS} ${EXEC}

-include ${DEPENDS}             # reads the .d files and reruns
                                # dependencies
```