CMSC 202
Additional Lecture – Makefiles

Prof. Katherine Gibson

Makefiles

- A makefile is a list of rules that can be called directly from the terminal
  - must be called **`Makefile`** or **`makefile`**
- Rules have four parts
  - **Target** – name of object or executable to create
  - **Dependency List** – what Target depends on
  - **TAB** – used to offset an Action
  - **Action(s)** – list of actions to create the Target

Makefile Rule Example

<u>Target</u>
The file to create. In this case an object file: Inher.o

<u>Dependency List</u>
The files that are required to create the object file. In this case Inher.cpp and Inher.h

```
Inher.o: Inher.cpp Inher.h
	g++ -ansi -Wall -c Inher.cpp
```

<u><TAB></u>
Used to signal what follows as an action *(do not use spaces!)*

<u>Action(s)</u>
What needs to be done to create the target. In this case it is the separate compilation of Inher.cpp

The **`make`** Utility

- Uses a Makefile to automate tasks like the compilation of a program

- Programs are normally multiple files
  - And only a few are changed at a time

- Recompiling everything every time can take a long time, and slows down development
  - Using **`make`** can help with this

Efficiency of **`make`**

- **`make`** only recompiles files that need to be
  - Files that have been updated
  - Files that depend on updated files
- Compares the timestamp of the dependency list items to that of the target
  - If a source is newer than the object file, the object file needs to be recompiled
  - Likewise if an object file is newer than the executable it needs to be re-linked

Example Makefile

```
Project1: Project1.o Inventory.o Cd.o Date.o
   g++ -ansi -Wall  -o proj1 Project1.o Inventory.o Cd.o Date.o

Project1.o: Project1.c Inventory.h
   g++ -ansi -Wall  -c Project1.c

Inventory.o: Inventory.c Inventory.h Cd.h
   g++ -ansi -Wall  -c Inventory.c

Cd.o: Cd.c Cd.h Date.h
   g++ -ansi -Wall -c Cd.c

Date.o: Date.c Date.h
   g++ -ansi -Wall -c Date.c
```

Specifying a Target

- The first target in the file is the “default target”
  - Should be the name of the executable to create
  - **`Project1`** (creates **`proj1`** executable)

- To call a specific rule or create a specific target, use **`make <TARGET>`**

- Omitting the target (typing just “**`make`**”) will create the default target

Dependency Graph

- A file may depend on one or more other files
  - Need to ensure correct compilation order

- Create a dependency graph, with the end goal of a executable named “main”

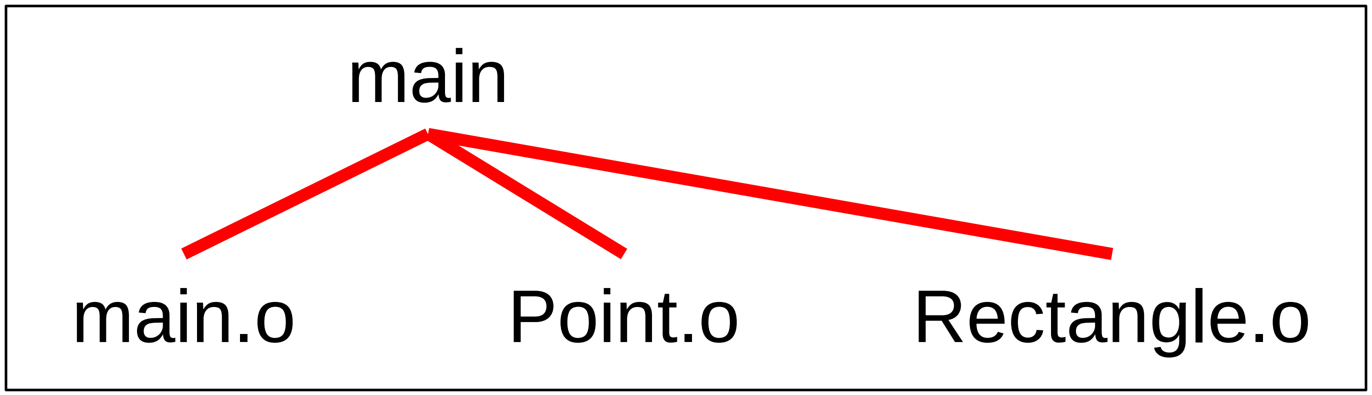
Our files:

```
main.cpp
Point.h        Point.cpp
Rectangle.h    Rectangle.cpp
```

Source: https://www.cs.bu.edu/teaching/cpp/writing-makefiles/

Dependency Graph – Linking

- The “main” executable is generated from 3 object files: **`main.o   Point.o   Rectangle.o`**
    - “main” *depends* on these files
- Files are *linked* together to create “main”

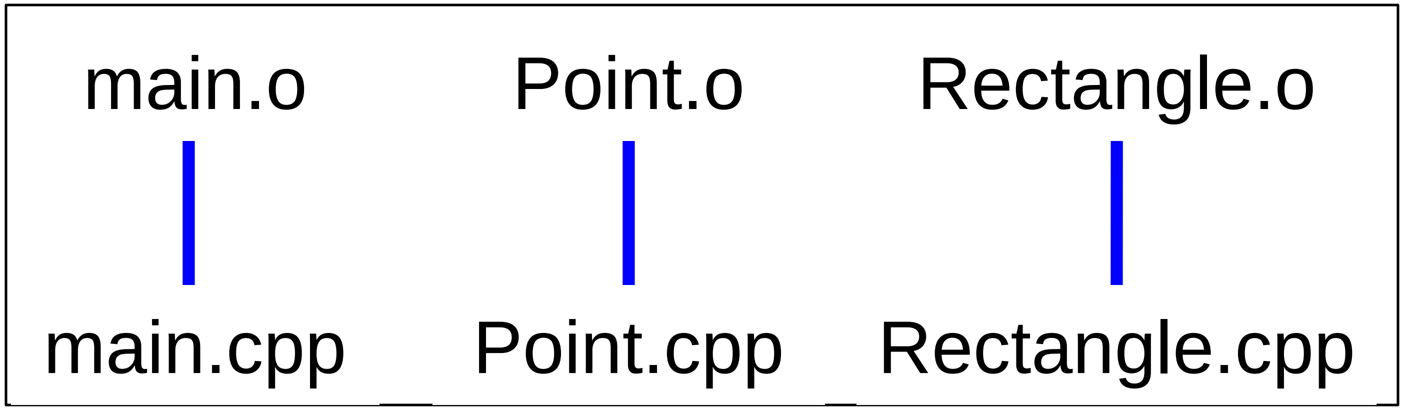


Source: https://www.cs.bu.edu/teaching/cpp/writing-makefiles/

Dependency Graph – Compiling

- Each of the object files *depends* on a corresponding **`.cpp`** file

- Object files are generated by *compiling* the corresponding **`.cpp`** files



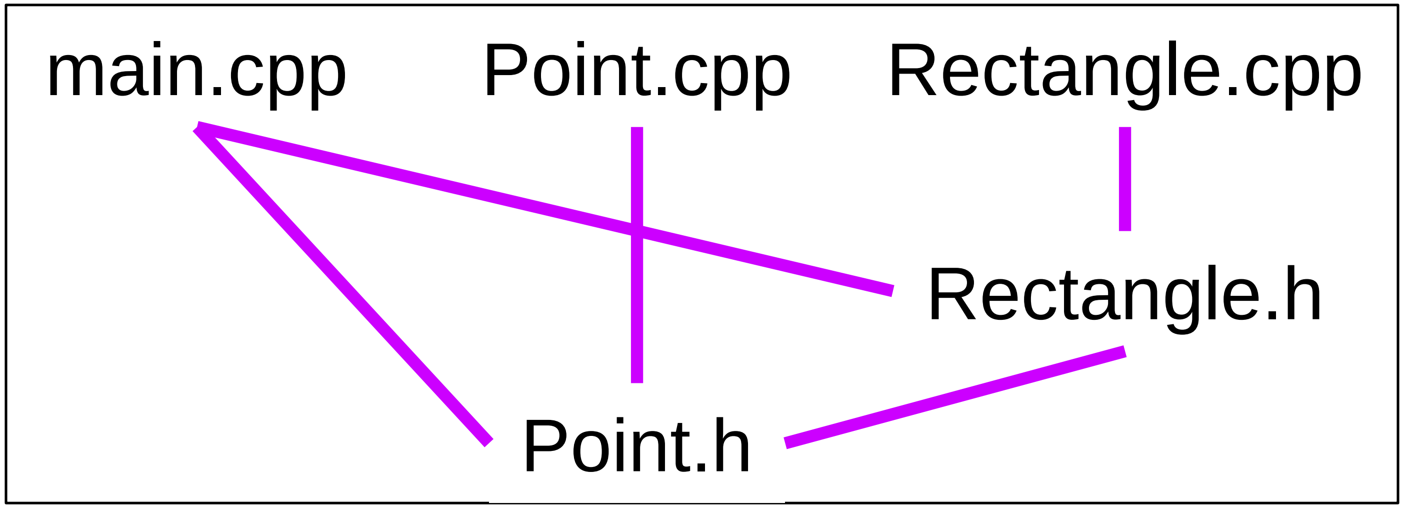
Source: https://www.cs.bu.edu/teaching/cpp/writing-makefiles/

Dependency Graph – Includes

- Many source code files (**`.cpp`** and **`.h`** files) depend on *included* header files

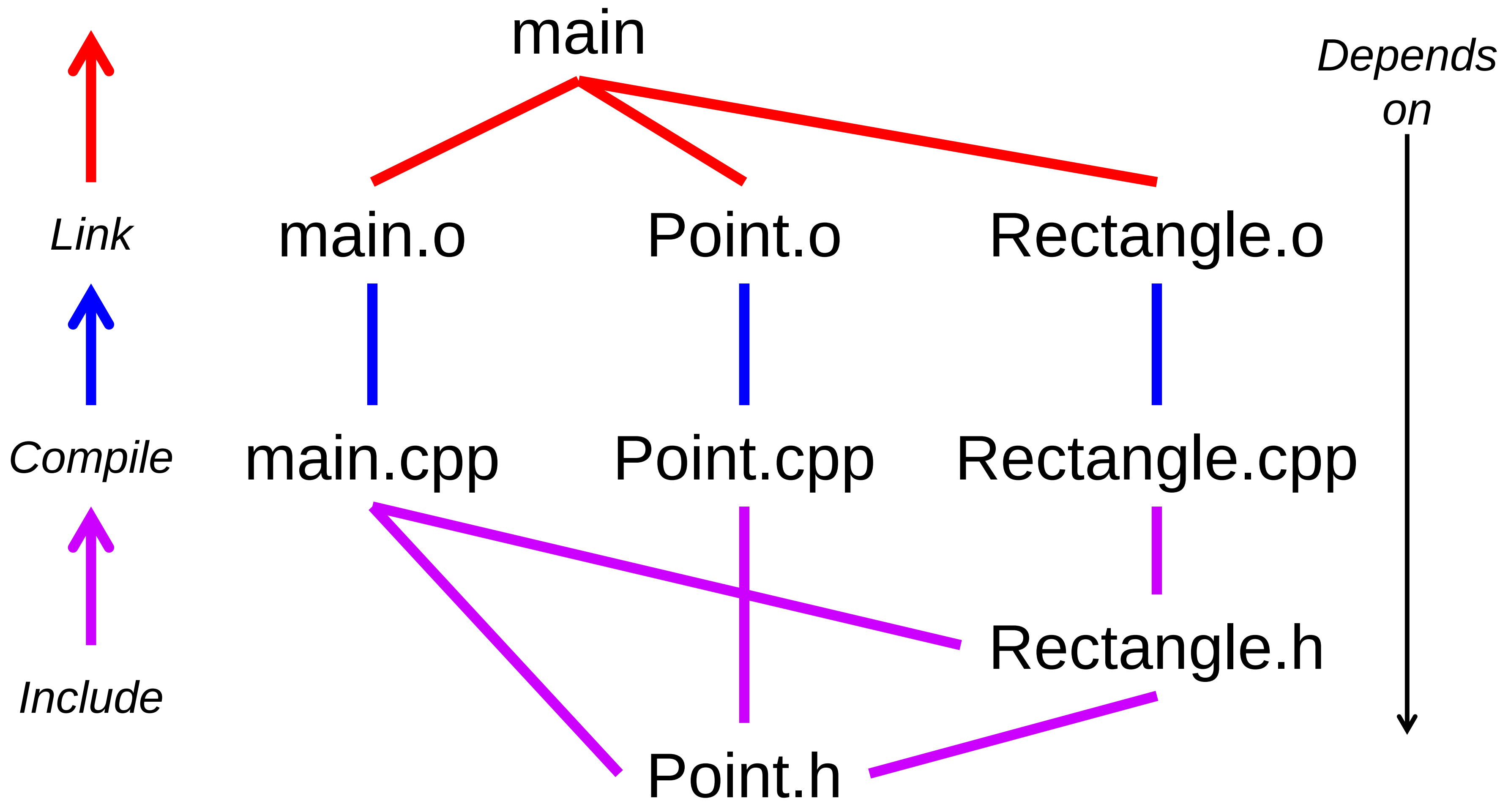
- May also be indirect includes; for example

`Rectangle.cpp` includes **`Point.h`** through **`Rectangle.h`**



Source: https://www.cs.bu.edu/teaching/cpp/writing-makefiles/

Full Dependency Graph



Source: https://www.cs.bu.edu/teaching/cpp/writing-makefiles/

Makefile Macros

Why Even Use Makefiles?

- Compiling, linking, and executing become…
  - Easier
  - Quicker (more efficient)
  - Less prone to human error
- Also allows us to create and run helper rules
  - Clean up unneeded files (like **`hw2.cpp~`**)
- Laziness (but *efficiently* lazy)

Makefile Macros

- Similar to an alias or a **`#define`**
  - Use when you need something over and over

- Syntax to define a macro:

```
PROJ = Proj1
CC   = g++
```

<u>Macro name</u>
Alias for macro

<u>Content</u>
Substituted for macro name in rest of file

Macro Examples

```
DIR1    = /afs/umbc.edu/users/k/k/k38/pub/CMSC341/Proj1/
PROJ    = Proj1
CC      = g++
CCFLAGS = -g -ansi -Wall -I . -I $(DIR1)

OBJECTS = Project1.o Inventory.o Cd.o Date.o
```

Notice that we can use one macro inside another *(declaration order matters)*

Using Macros

- To access a macro, use the following format:

  `$(MACRO_NAME)`

```
Project1: $(OBJECTS)
   $(CC) $(CCFLAGS) -o $(PROJ).c $(OBJECTS)

Project1.o: Project1.c Inventory.h
    $(CC) $(CCFLAGS) -c Project1.c
```

- What do each of these rules actually mean?
  - (In plain English)

Helper Rules

- You can specify targets that do auxiliary tasks and do not actually compile code
  - Remove object and executable files
  - Print source code
  - Submit all code

- Timestamps don’t matter for these tasks
  - Good practice to let the makefile know that
  - These target are called “phony” targets

Phony Targets

- Same syntax, but preceded by a **`.PHONY`** declaration on the previous line

Same as target name

Use a backslash if the command is longer than one line

```
.PHONY: submit
submit:
    scp $(ALL_FILES) \
k38@gl.umbc.edu:cs202proj/proj3/
```

More Helper Rules

- Cleaning utilities

```
clean:
    -rm -f *# *~
cleaner: clean
    -rm -f *.o
cleanest: cleaner
    -rm -f core*; rm -f $(PROJ)
```

Be very, *very*, **very**, **VERY** careful when copying these rules into your Makefile the first time!

The `-f` flag will supress the prompt to confirm deletion – and once it's deleted, it's gone! (For good!)

- Pure laziness

```
make:
    emacs Makefile
```

Full Makefile Example

```
PROJ     = Proj1
CC       = g++
CCFLAGS = -g -ansi -Wall

SOURCES = $(PROJ).c Inventory.h Inventory.c Cd.h Cd.c Date.h Date.c
OBJECTS = $(PROJ).o Inventory.o Cd.o Date.o

$(PROJ): $OBJECTS
   $(CC) $(CCFLAGS) -o $(PROJ) $(OBJECTS)

$(PROJ).o: $(PROJ).c Inventory.h
    $(CC) $(CCFLAGS) -c $(PROJ).c

Inventory.o: Inventory.c Inventory.h Cd.h
    $(CC) $(CCFLAGS) -c Inventory.c

Cd.o: Cd.c Cd.h Date.h
    $(CC) $(CCFLAGS) -c Cd.c

Date.o: Date.c Date.h
    $(CC) $(CCFLAGS) -c Date.c

.PHONY: submit
submit:
   submit cs341 $(PROJ) $(SOURCES) Makefile *.txt

.PHONY: print
Print:
   enscript -G2rE $(SOURCES) Makefile *.txt
```

Target rule
(the first rule
in the file)