CMSC 341

Makefile Review

Make Overview

- **make** is a program that automates the compilation of programs whose files are dependent on each other
- A program typically consists of several files, and a programmer usually only works on a few of them at a time
  - Typically vast majority of files remain unchanged and thus do not need to be recompiled

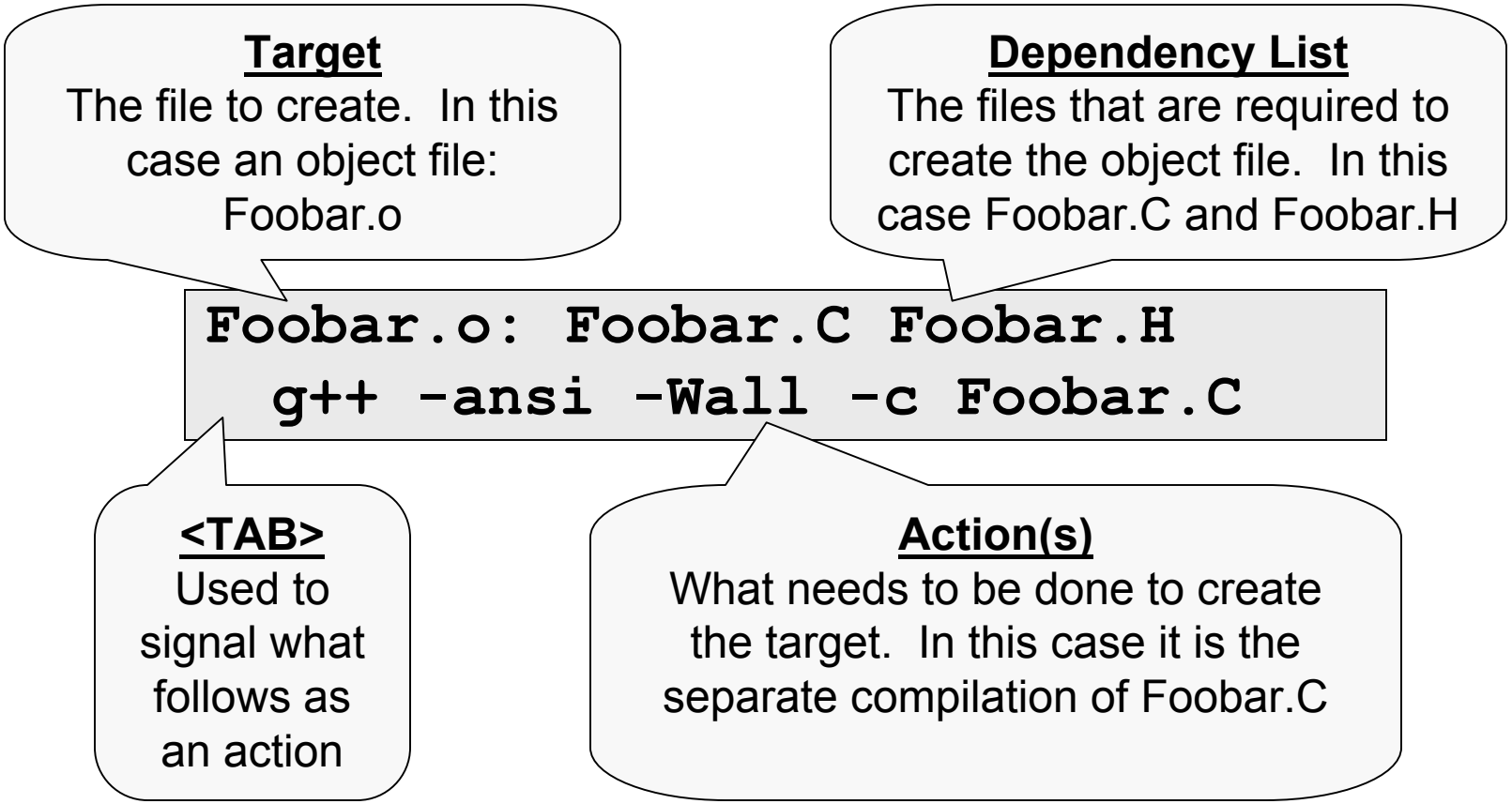
Make Overview

- **make** will only recompile the files that need to be updated (files that depend on modified files), which is much faster than simply recompiling the whole program each time
- **make** looks at the timestamps of source files (*.C, *.H) that are required to generate an object file (*.o)
  - If a source is newer then 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

Makefile Structure

- A makefile is a list of rules of what files are required to generate an object or an executable file
  - File typically called makefile or Makefile
- Each rule consists of 4 parts:
  - Target: the name of the object or executable to create
  - Dependency List: the list of files that the target is dependent upon
  - TAB: used to set off an action
  - Action(s): a list of actions to take in order to create the target (i.e. g++ …)

Makefile Rule



A simple 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
```

Target Specification

- You can automatically create any of the targets by typing **make <TARGET>**
- If you omit a target as a parameter to make, it assumes that you want the first target in the makefile
  - This target is known as the default target and is almost always the name of the executable that you want to create

Dependency Graph

- Discussed in class

Makefile Macros

- Similar to a `#define` or alias – use when you need the same thing over and over
- Syntax to define macro is:
  - MACRO_NAME = content to be substituted for MACRO_NAME
- Benefits: easy to make changes – simply change in 1 place rather than on all targets, etc…

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

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

Makefile Macros

- To access a declared macro simply put the name of the macro inside parenthesis after a dollar sign
- Syntax to recall macro:
  - $(MACRO_NAME)

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

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

Phony Targets

- 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
- These tasks do not require looking at the timestamps on files and thus it is good practice to let make know that there is no “real” target for a rule, and that it should just execute the action list

Phony Targets

- The syntax is just like any other rule except the rule is proceeds a .PHONY declaration
- Syntax is: .PHONY: target

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

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

Advanced Makefile

```
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
```

Templates

- As mentioned in class you should never explicitly compile template classes when using g++
- What implications does this have on makefiles?
  - **Do not** create a target for a template class
    - You will never make a myTemplateClass.o object file, as template code automatically gets compiled when it is seen declared
  - Since object files are not created, they **do not** be listed in the OBJECTS target dependency list
  - **Do** list template class files as dependencies for an target (object) when the files that create that target (object) reference them