And You Thought There Couldn't be More C++

Outline

- Multi-File Programs
- makefiles

makefiles

- As you get more and more files, compilation at the command line gets more and more tedious
- You can put your compilation commands into a single file named “makefile” and use the Linux utility, “make” to do the compilation
- make program looks at list of requirements in the makefile, checks time stamps, and if something is out of date, re-compiles it
  - That way, only the files that have changed need to be updated

makefiles

- You can use variables in a makefile
  - Common variables:
    - CFLAGS = -g -Wall
    - CC = g++
  - To use the variable, use ${varname}
    - e.g. ${CFLAGS}
- You can also insert comments
  - Comments are preceded by the # symbol

makefiles

- Dependencies
  - Rules in a makefile that specify what needs to be done, in what order
    - [name of rule] : [list of other rules, separated by spaces]
      [list of source files, separated by spaces]
    - [TAB] command to execute in the event the rule is violated
  - Called “dependencies” because one rule can depend on the status of another
  - You *must* use a TAB character, not a sequence of spaces, to ensure that your commands will be interpreted correctly
    - You may have multiple tabbed commands to satisfy a rule

makefile Example

- Let's say we have two files, main.cpp and help.cpp, in our program
- The makefile might look like:

```
main.o: main.cpp

        g++ -c main.cpp

help.o:  help.cpp

        g++ -c help.cpp

main.exe: main.o help.o

        g++ main.o help.o –o main.exe
```

makefiles

- To run a makefile, simply type:
  - make
- make will look for the file called makefile and execute the compilation commands
- If you have several makefiles, you can name them different names (for example MyMakefile) and use the command:
  - make –f MyMakefile
- If you want make to only execute one rule, call that rule:
  - make clean

makefiles – A More Interesting Example

```
# makefile for a frog project

CC=g++
CFLAGS=-g –Wall
RM=rm –f

all: main helloworld

frog.o: frog.h frog.cpp
	${CC} ${CFLAGS} –c frog.cpp
main: main.o frog.o
	${CC} ${CFLAGS} –o main main.o frog.o
helloworld: helloworld.cpp
	${CC} ${CFLAGS} –o helloworld helloworld.cpp
clean:
	${RM} *.o main
```

makefiles

- Not all of your files need to be in the same directory to be compiled by a makefile
- You can use any Linux command inside a makefile as a command (the tabbed parts)
- You can use make with any compiler – that's what the CC and CFLAGS variables were about in the last example
- There are dependency generator programs that will create makefiles for you if your program is very complex

Summary

- Multi-File Programs
- makefiles