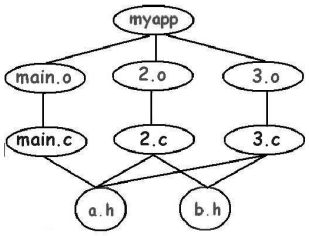
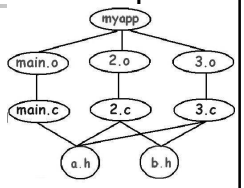
Multiple Targets



- It's useful to make more than a single target file.
  Example: main.c 2.c 3.c
- Collect several groups of commands into a space
  Example: "clean" option … etc
- How to do it? We can extend our "**makefile**" to do this.

Multiple Targets Example

```
all:    myapp
CC = gcc                        # which compiler
INSTDIR = /usr/local/bin        # where to install
INCLUDE = .                     # where are include files kept
CFLAGS = -g  -Wall  -ansi       # Options for development
myapp:  main.o  2.o  3.o
            $(CC)  -o  myapp  main.o  2.o  3.o
main.o:  main.c  a.h
            $(CC)  -I $(INCLUDE)  $(CFLAGS)  -c  main.c
2.o:  2.c  a.h  b.h
            $(CC)  -I$  (INCLUDE)  $(CFLAGS)  -c  2.c
3.o:  3.c  a.h  b.h
            $(CC)  -I$  (INCLUDE)  $(CFLAGS)  -c  3.c
Clean:
            -rm  main.o  2.o  3.o
Install:  myapp
            @if [ -d  $(INSTDIR) ] ; \
                then
                    cp  myapp $(INSTDIR) ; \
                    chmod  a+x  $(INSTDIR) / myapp ; \
                    chmod  og-w  $(INSTDIR) / myapp ; \
                     echo  "Installed  in  $(INSTDIR) "  ; \
                else \
                    echo "Sorry, $(INSTDIR) does not exist" ; \
                fi
```



Multiple Targets Example Explanation

all: myapp

➔ Create myapp executable file

CC = gcc **# which compiler**

➔ make command replaces $(CC) with gcc

INSTDIR = /usr/local/bin **# where to install**

➔ make command replaces $(INSTDIR) with the specific path /usr/local/bin

INCLUDE = . **# where are include files kept**

➔ make command replaces $(INCLUDE) with a dot "."

CFLAGS = -g -Wall -ansi **# Options for development**

➔Option -g : produce debugging information in the operation system's native format
Option –Wall : show all Warnings
Option –ansi : ansii standard

Multiple Targets Example Explanation

➔ make finds the target myapp and sees it depends on the object files main.o 2.o 3.o

myapp: main.o 2.o 3.o
$(CC) -o myapp main.o 2.o 3.o

➔ Now make looks to see if either of these files are listed as targets and since they aren't it executes the commands given in main.o's rule and compiles main.c to get the object file main.o.

main.o: main.c a.h ➔ main.o depends on the files main.c and a.h
$(CC) -I $(INCLUDE) $(CFLAGS) -c main.c

➔gcc –I. -g -Wall -ansi -c main.c
Option -I<dir> : Search in directory <dir> for include files that do not start with an absolute path.
Option –c : Compile or assemble the source files, but do not link.

➔ make looks at the targets 2.o and 3.o and compiles these object files in a similar fashion. make now has all the object files required to make myapp and does so by executing the commands in its rule

2.o: 2.c a.h b.h
$(CC) -I$(INCLUDE) $(CFLAGS) -c 2.c
3.o: 3.c a.h b.h
$(CC) -I$(INCLUDE) $(CFLAGS) -c 3.c

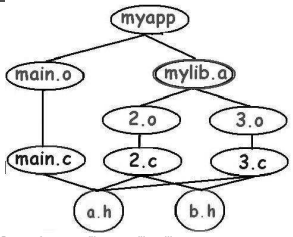
Multiple Targets Example Explanation

Clean:
-rm main.o 2.o 3.o

➔ Clean is a phony target: not the name of a file. It will only have a list of commands. It uses the rm command to remove the objects.
Option "-" ignore the result of the command.

Install: myapp ➔ install depends on myapp
No echo➔ **@if [-d $(INSTDIR)] ; ** ➔ continue ➔ $(INSTDIR) = /usr/local/bin
**then **
**cp myapp $(INSTDIR) ; **
execute permission ➔ **chmod a+x $(INSTDIR) / myapp ; **
No-write by group/others ➔ **chmod og-w $(INSTDIR) / myapp ; **
**echo "Installed in $(INSTDIR) " ; **
**else **
**echo "Sorry, $(INSTDIR) does not exist" ; **
fi

Managing Libraries with make



- syntax
  .c.a: ➔ Get from 2.c file to mylib.a library
  $(CC) -c $(CFLAGS) $< ➔ compile and generate object file
  $(AR) $(ARFLAGS) $@ $*.o ➔ use ar and rv command to revise library

Example: ar rv mylib.a 2.o
$@ = current target = mylib.a
$* = current prerequisite = 2

➔ **mylib.a (2.o)**

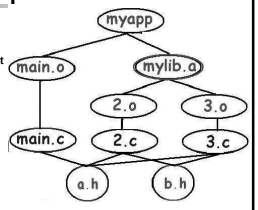
Library Example

```
all: myapp
CC = gcc                        # which compiler
INSTDIR = /usr/local/bin        # where to install
INCLUDE = .                     # where are include files kept
CFLAGS = -g  -Wall  -ansi  # Options for development
MYLIB = mylib.a                 # local libraries

myapp:   main.o   $(MYLIB)
            $(CC)   -o   myapp   main.o   $(MYLIB)

$(MYLIB):   $(MYLIB)   (2.o)   $(MYLIB)   (3.o)

main.o:  main.c   a.h
2.o:  2.c  a.h  b.h
3.o:  3.c  b.h  c.h
clean:
            -rm   main.o   2.o   3.o   $(MYLIB)
install:   myapp
            @if  [ -d  $(INSTDIR)  ] ; \
                then
                    cp   myapp $(INSTDIR) ; \
                    chmod   a+x   $(INSTDIR)  / myapp ; \
                    chmod   og-w   $(INSTDIR) / myapp ; \
                     echo   "Installed  in  $(INSTDIR) "  ; \
                else  \
                    echo "Sorry, $(INSTDIR) does not exist" ; \
                fi
```



Library Example Explanation

```
all: myapp
CC = gcc                        # which compiler
INSTDIR = /usr/local/bin        # where to install
INCLUDE = .                     # where are include files kept
CFLAGS = -g  -Wall  -ansi  # Options for development
MYLIB = mylib.a                 # local libraries

myapp:   main.o   $(MYLIB)
            $(CC)   -o   myapp   main.o   $(MYLIB)

$(MYLIB):   $(MYLIB)(2.o)   $(MYLIB)(3.o)
```

mylib.a(2.o): 2.c a.h b.h
gcc -c -g -Wall -ansi 2.c
ar rv mylib.a 2.o

mylib.a(2.o): 2.c a.h b.h
Gcc -c -g -Wall -ansi 2.c
ar rv mylib.a 2.o

```
main.o:  main.c   a.h
2.o:  2.c  a.h  b.h
3.o:  3.c  b.h  c.h
```

Library Example Explanation

```
Clean:
        -rm   main.o   2.o   3.o   $(MYLIB)
Install:   myapp
        @if  [ -d  $(INSTDIR)  ] ; \
            then
                cp   myapp $(INSTDIR) ; \
                chmod   a+x   $(INSTDIR)  / myapp ; \
                chmod   og-w   $(INSTDIR) / myapp ; \
                 echo   "Installed  in  $(INSTDIR) "  ; \
            else  \
                echo "Sorry, $(INSTDIR) does not exist" ; \
            fi
```

Library Example Explanation

```
$rm –f myapp *.o mylib.a

$make –f Makefile6

gcc  –g   -Wall   –ansi   -c   main.c   –o   main.o

gcc  –g   -Wall   –ansi   -c   2.c   –o   2.o

ar   rv   mylib.a   2.o

ar: creating  mylib.a

gcc –g –Wall –ansi   -c   3.c   -o   3.o

ar   rv   mylib.a   3.o

gcc   –o   myapp   main.o   mylib.a
```

What have you learned?

- Make is a powerful tool for managing large projects and maintaining libraries.
- Make can be used in any programming language; Ant comes out of Make.
- Make saves on compile time, and lets programmers worry about the important stuff.