SPLITTING THE CODE

- Sometimes we like to split our program into multiple files (or modules).
- Advantages: encapsulation, reusability, size.
  - We can also reduce compilation time.
- A module consists in two parts:
  - the header file, where all the declarations available outside the module go (e.g., `list.h`)
  - the C/C++ code which implements the things declared in the header (e.g., `list.cc`)
- Another module (say `main.cc`) that wants to use `list.cc` will do

```
#include "list.h"
```

  - Then `list.cc` and `main.cc` will be compiled and linked together.
    - We use for this purpose a makefile.

MAKEFILES

- A makefile contains macrodefinitions, e.g.,

```
# this is a comment
CXX = g++
CXXFLAGS = -g -Wall -Werror -ansi -pedantic
OBJ = main.o list.o
```

- Then we have rules of the form:

```
target : [source1 ] [source2 ] [source3 ]
        command1
        command2
        command3
        ...
```

Exactly one TAB on each line here!

  - a target is the name of the file to be produced
    - it is produced by executing the corresponding commands
  - the sources are the files needed to produce the target (if any)

MAKEFILES (CONT'D)

- Example of rules:

```
all: test_list

list.o: list.h list.cc
        $(CXX) $(CXXFLAGS) -c -o list.o list.cc

main.o: list.h main.cc
        $(CXX) $(CXXFLAGS) -c -o main.o main.cc

test_list: $(OBJ)
        $(CXX) $(CXXFLAGS) -o test_list $(OBJ)

clean:
        rm -f test_list *~ *.o *.bak core \#*
```

- You type make *target* in some directory *d*.
  - `make` without arguments produces the first target in the (default) makefile.
- The command looks for a file called `Makefile` in *d* and produces the file *target*.
- All the targets needed by *target* are also made, unless they are up to date.

PUTTING THE FILES TOGETHER