Computational Physics

Makefiles

automatic building...

Fernando Barao, Phys Department IST (Lisbon)

C++ Makefiles

- ✔ UNIX/Linux systems have a powerful tool called **make** that allows you to manage compilation and linking of multiple modules into an executable
- ✔ The **make** utility reads a specification file, **makefile**, that describes how the modules of a software system depend on each other
- ✔ The several steps to compile a program including also some other actions like directory cleaning can be grouped in this special file *Makefile* or *makefile*. To run it:

```
$ make <tag>
```

- ✔ **make** options:
  -d display debug information
  -f filename specifies a different name for the specification file
  -h display a brief description of all options
  -n do not run any makefile command, just display them
  -s run in silent mode

C++ Makefiles (cont.)

- The Makefile is organized in blocks with a *target-list*, *dependency-list* and *commands*.
  - Every action is preceded of spaces generated by a **TAB**.
  - A long line can be continued by terminatingit with a backsplash (\) character
  - add comment by using #
  - we can use shell's filename pattern characters (?, *)

```
# block example
target-list: dependency-list
<TAB>   command-list
```

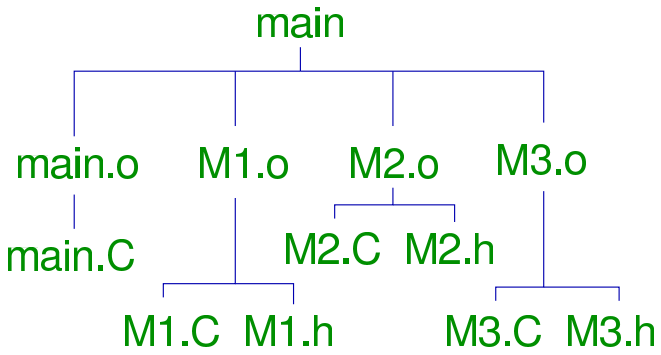

C++ Makefiles (cont.)

- Suppose we have a main program **main.C** and list of C++ modules called **M1.C, M2.C, M3.C**
- The compilation of **main.C** will depend on the compilation of the modules

```
main: main.o M1.o M2.o M3.o
        g++ -o main.exe main.o M1.o M2.o M3.o
```

- ...but we need to produce the object code of the modules and text.C

```
main.o: main.C
        g++ -c main.C
M1.o: M1.C M1.h
        g++ -c M1.C
M2.o: M2.C M2.h
        g++ -c M2.C
M3.o: M3.C M3.h
        g++ -c M3.C
```



C++ Makefiles (cont.)

- To remove all object files of current directory

```
clean:
        rm -f *.o
```

- **Suffix default rules**

  The make rules shown precedingly contain some redundant commands

  The make utility has many predefined make rules, also known as *suffix rules*, that allow the make utility to perform automatic tasks

```
test: test.o M1.o M2.o M3.o
        g++ -o test.exe test.o M1.o M2.o M3.o

M1.o: M1.C M1.h
M2.o: M2.C M2.h
M3.o: M3.C M3.h

.C.o:
        g++ -fPIC -Wall -g -c $<
```


C++ Makefiles (cont.)

- **MACROS**

  The *make* utility supports simple macros that allow text substitution (define before using them)

- **user defined macros**

```
1. macro_name = text
2. define macro_name
   text
   endef
```

- to use the macros: **$ (macro_name)**

- **built-in macros**

  $@ name of current target

  $? list of dependencies that have changed more recently than target

  $< the name of the current dependency that has been modified more recently than current target

  $^ a space-separated list of all dependencies without duplications

C++ Makefiles (cont.)

```
# user macros
define CC
  g++
endef
CFLAGS = -Wall -g -fPIC
LDFLAGS = -lm
OBJS = main.o M1.o M2.o M3.o
SRCS = main.C M1.C M2.C M3.C
HEAS = main.h M1.h M2.h M3.h
# actions
build: test.exe
       @echo building $?
test.exe: $(OBJS)
       $(CC) $(CFLAGS) -o $@ $^
       @echo compiling $?
test.o: $(SRCS) $(HEAS)
.C.o:
        $(CC) $(CFLAGS) -c $<
clean:
        rm -f *.o
```


C++ Makefiles: built-in functions

- The **wildcard** function can be used to get a list of all the C++ source files in a directory, like this:

```
sources := $(wildcard *.C)
```

- We can use the **patsubst** function to change the list of C++ source files into a list of object files by replacing the **'.C'** suffix with **'.o'** in the result

```
objects := $(patsubst %.c,%.o,$(wildcard *.c))
```

- for instance, for generating an executable inside the Makefile from all *objects*

```
Rtest.exe : $(objects)
        cc -o Rtest.exe $(objects)
```

- The **shell** function performs the same function that backquotes (') perform in most shells: it does command expansion

```
contents := $(shell cat Rtest.C)
```

- For searching for a substring we can use the **findstring** function

```
$(findstring str1, str2)
```

C++ Makefiles: built-in functions

- ✔ Extract all but the directory-part of each file name in names

```
$(notdir src/*.C)
```

- ✔ Extract all but the suffix of each file name in names

```
$(basename src/foo.C src-1.0/bar hacks)    => produces the result 'src/foo
```