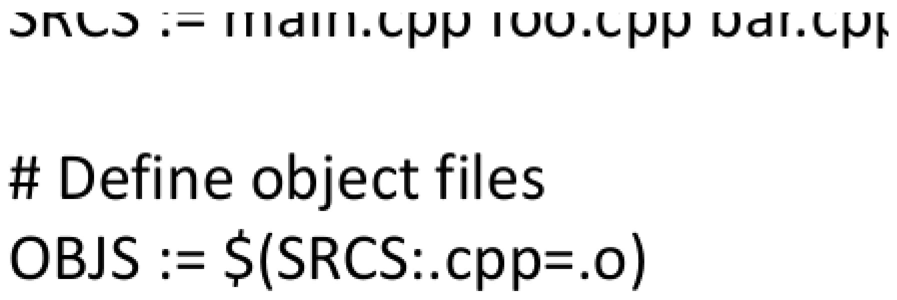
- .PHONY Ensures the target always runs, even if a file with the same name exists
- `.PHONY` is used in Makefiles to **declare phony targets**, which means:
    1. **They do not correspond to actual files** in the directory.
    2. **They always execute** when invoked, regardless of whether a file with the same name exists.
- make -j4 will lead to running 4 jobs in parallel and speed up compilation.
- .SUFFIXES: removes all default suffix based rules.
- Add a “-” before a command to suppress the error associated with that command.
- Add -k when running make to continue running even in the face of errors

<u>SOME GOOD PTS FOR LAB</u>



```
SRCS := main.cpp foo.cpp bar.cp

# Define object files
OBJS := $(SRCS:.cpp=.o)
```

- very useful
  But keep in mind that here SRCS is a variable which stores a list not a list itself, it is crucial to syntax you can’t directly put a list here.

`$(dir names…)`

Extracts the directory-part of each file name in *names*. The directory-part of the file name is everything up through (and including) the last slash in it. If the file name contains no slash, the directory part is the string '`./`'. For example,

```
$(dir src/foo.c hacks)
```

produces the result '`src/ ./`'.

`$(notdir names…)`

Extracts all but the directory-part of each file name in *names*. If the file name contains no slash, it is left unchanged. Otherwise, everything through the last slash is removed from it.

A file name that ends with a slash becomes an empty string. This is unfortunate, because it means that the result does not always have the same number of whitespace-separated file names as the argument had; but we do not see any other valid alternative.

For example,

```
$(notdir src/foo.c hacks)
```

- produces the result '`foo.c hacks`'.
  Here note that names are filenames. Hence if you are thinking of using a list of files here then you have to pass the list itself not the variable holding the list as $(SRCS) unlike what we did in previous point.
- <u>SOME IMPORTANT BASIC POINTS:</u>
  - $ in makefiles is used for Variable substitution(`$(VAR)` or `${VAR}`) or function execution($(function args))(make functions).
  - If you are writing a shell command for a target. Then writing $ will do the same stuff as above again. But we might want to give in $ in the command(say for command substitution), for this we have to write $$, just $ will try to interpret it as a make function or variable name which is wrong but $$ will give it just as a single $ to the shell.
  - Make function shell helps to run a command in shell. src=$(shell find . -type f) will store a list of files(recursive search even of subdirectories etc because of find) in src variable.
  -