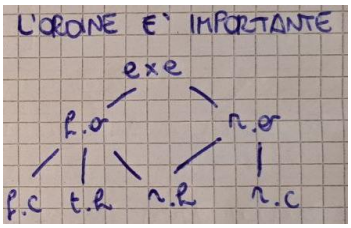
MAKEFILE

- Permette di esprimere dipendenze tra file.

Se f.o dipende da f.c, t.h, r.h
- f.o è detto target.
- f.c, t.h, r.h è una dependency list.

```
exe : f.o r.o
    gcc f.o r.o -o exe
f.o : f.c t.h r.h
    gcc -Wall -pedantic -c f.c
r.o : r.c r.h
    gcc -Wall -pedantic -c r.c
```



Come si esegue?

```
> make                    se il file si chiama Makefile
> make -f nomefile        se il file NON si chiama Makefile
> make -n                 stampa i comandi senza eseguirli
> make radicealbero       esegue solo la radice specificata
```

Variabili

- Stringhe di testo definite una volta e usate in più punti.

```
OBJS = r.o f.o
     ...
exe : $(OBJS)
    gcc $(OBJS) -o exe
```

- Variabili predefinite:

```
$@    nome del target
$^    dependency list
$<    primo nome nella dependency list
```

Phony targets

- Specificare target che non sono file e che hanno come scopo l'esecuzione di una sequenza di azioni.

```
.PHONY : clean test
     ...
clean :
    -rm $(OBJS) $(EXE) *~ core
```

Il '-' davanti a rm serve per far continuare l'esecuzione del make anche in caso di errori nell'esecuzione del comando (e.g. uno dei file specificati non c'è), altrimenti make si blocca al primo comando che ritorna errore.
Oppure si può usare l'opzione '-f' al posto del '-' davanti.

```
        rm -f $(OBJS) $(EXE) *~ core
```

- Posso usarli anche per fare dei test.

```
test : hTest                                esempio slide
    @echo "Eseguo i test.."
    ./hTest 1> output
    diff output output.atteso
    @echo "Test superato!"

test : $(EXE)                               esempio fatto dal prof
    @echo "Eseguo il test.."
    @./$(EXE) metti in maiuscolo
    @echo "Test superato!"
```

'@' serve per non far annunciare ogni comando.