# Manteniment de projectes: make (1)

## Programació de Baix Nivell

Sebastià Vila-Marta

Enginyeria de Sistemes TIC
Universitat Politècnica de Catalunya
http://epsem.upc.edu

29 de març de 2020

1 El desenvolupament dels projectes

2 Dependències en un projecte

3 Make

# Construcció de l'executable

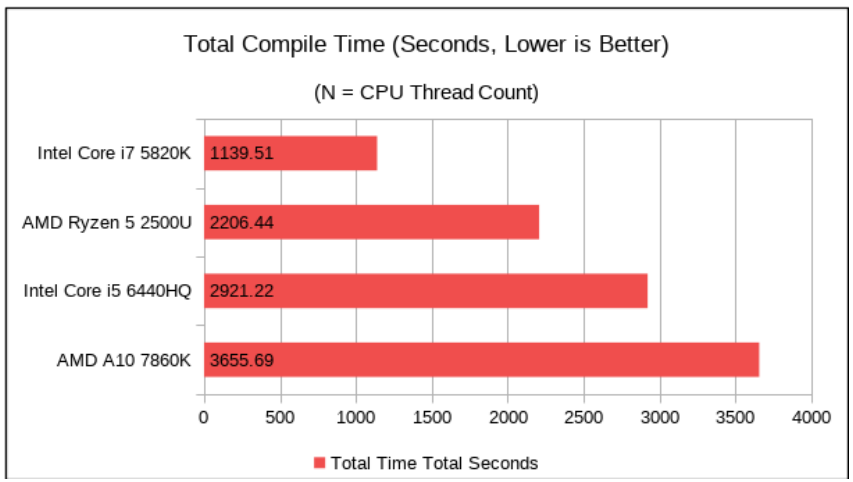
- La fase de desenvolupament d'un projecte de programari comporta un seguit de tasques repetitives que poden arribar a ser complicades quan el projecte esdevé gran.
- Una d'aquestes tasques és la construcció (*build*) de l'executable en els entorns compilats. Essencialment consisteix a:
  - Compilar totes les unitats de compilació.
  - Actualitzar les llibreries de mòduls si n'hi ha.
  - Enllaçar (*link*) els mòduls i llibreries necessaris per crear els executables

# No és tant fàcil...

- Malgrat la construcció sembla fàcil, en realitat no ho és tant i és l'origen de problemes habituals.
- Alguns dels problemes són:
  - No saber què s'ha modificat i què cal recopilar.
  - Per evitar el cas anterior, reconstruir sistemàticament tota l'aplicació amb la pèrdua de temps que comporta.
  - Ordres de compilació/muntatge diferents segons els mòduls, cosa que implica problemes d'execució.
  - Estructures de projecte complexes que requereixen passos de construcció extensos.

# Exemple: Linux kernel 4.17rc6...

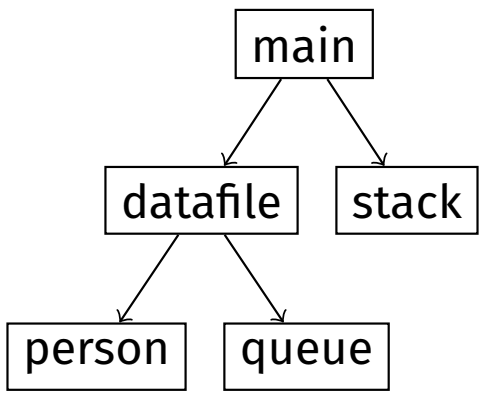
- Compilació del kernel de Linux a màxim rendiment (usant tots els cores disponibles).
- Aquesta versió aprox 20 MLOC
- Els temps a diverses màquines són molt significatius:




- Reconstruir cada vegada que es fa un canvi pot ser molt costós.
- Passegeu-vos per `https://www.openhub.net` per descobrir les mides dels projectes!

# Dependències entre mòduls

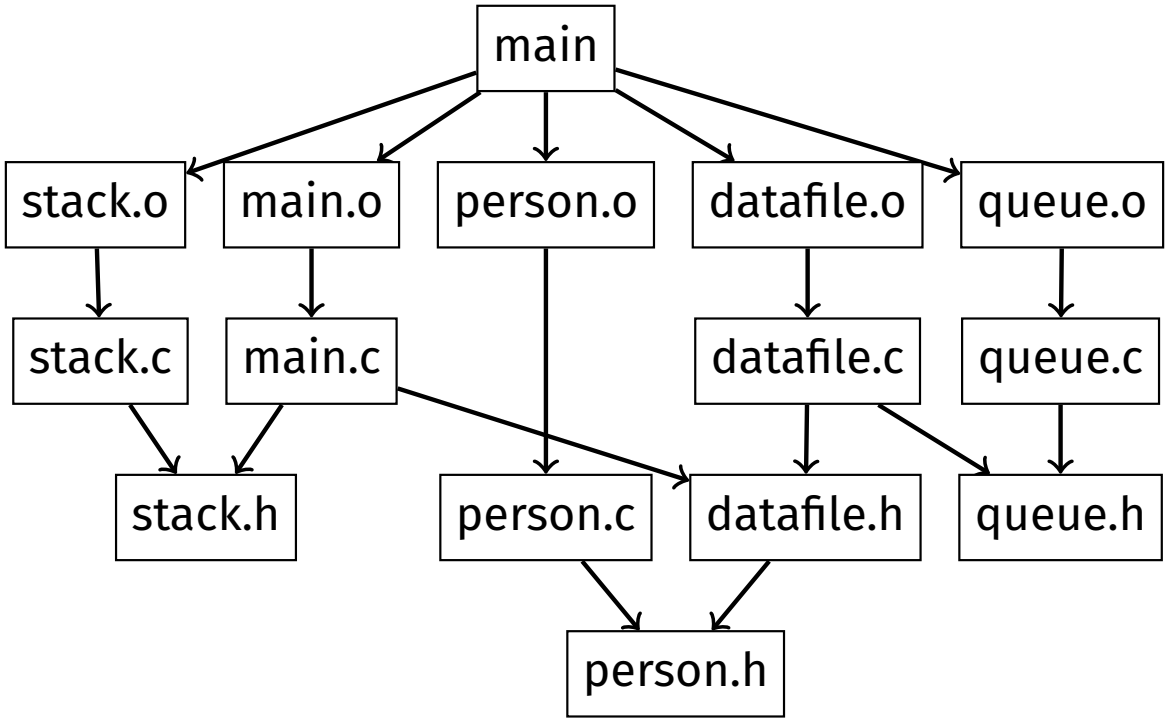
- En el disseny d'un programa, hi ha dependències entre mòduls: un modul *A* usa funcionalitats que ofereix un altre mòdul *B*.
- Típicament representem aquesta estructura en diagrames de dependències entre mòduls.



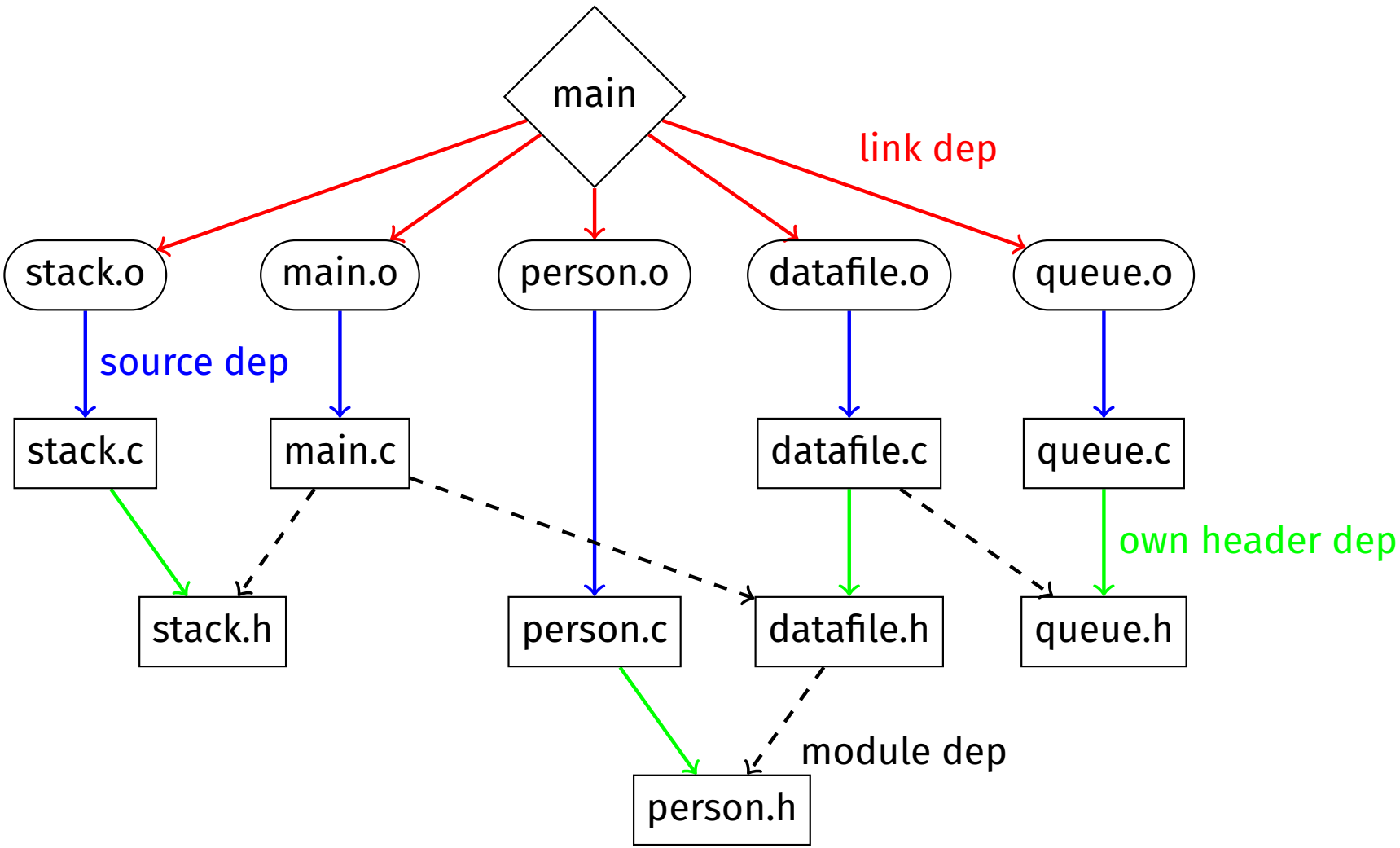
# Dependències entre fitxers

- Quan traslladem una estructura de mòduls a C, cada mòdul `M` s'implementa generalment sobre dos fitxers: `M.h` i `M.c`.
- Les dependències entre mòduls sumades a l'estratègia d'implementació indueixen unes dependències entre fitxers.
- Aquestes dependències venen donades per:
  - La relació d'`include`.
  - La relació entre un objecte i el seu font.
  - La relació entre un executable i els seus objectes.
- En aquest context el fitxer `A` depèn del fitxer `B` si al modificar `B` el fitxer `A` esdevé obsolet.

# Dependències entre fitxers: exemple



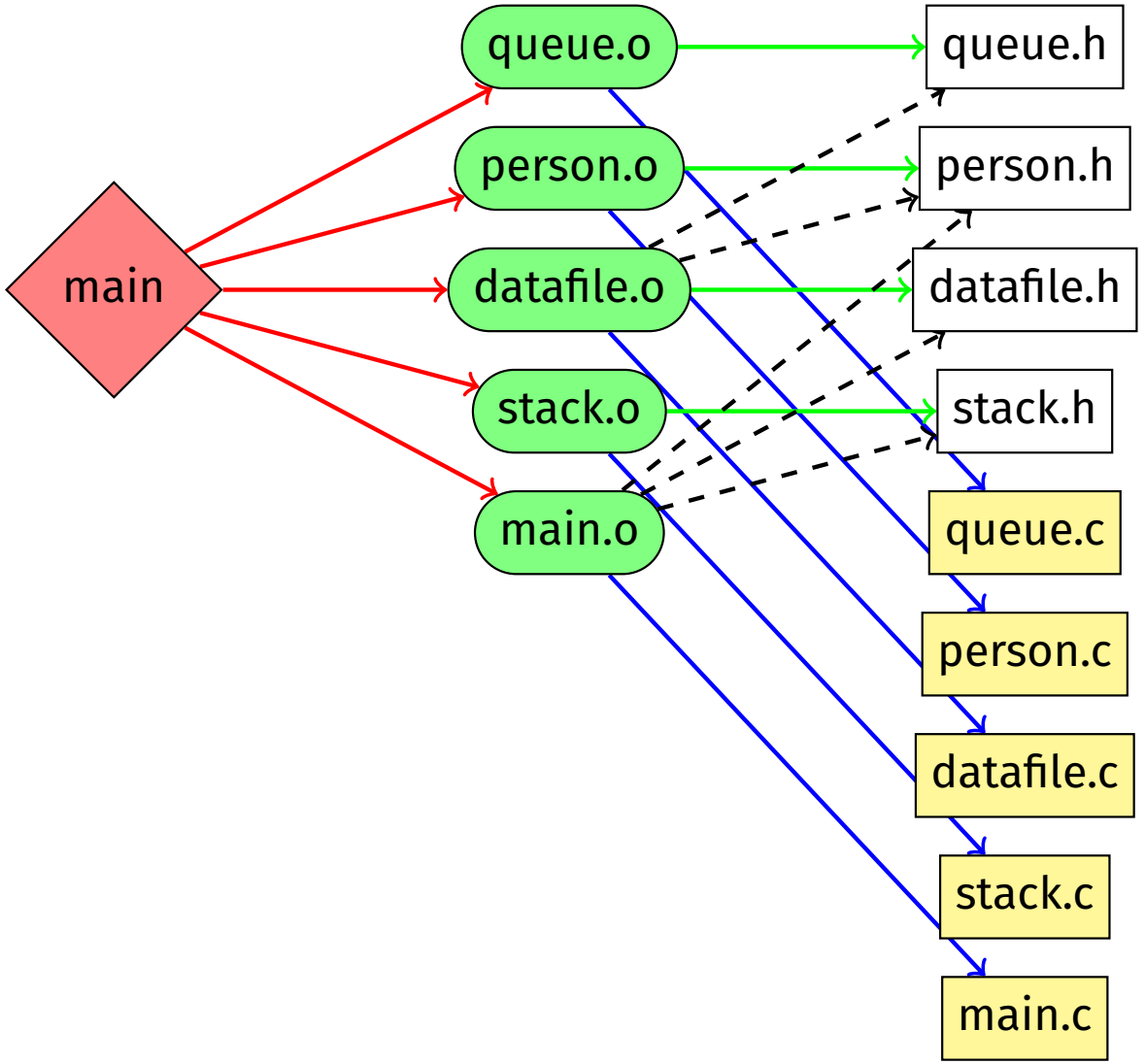
# Dependències entre fitxers: exemple (II)



# Clausura del graf de dependències

- La unitat de compilació a C la formen el fitxer font i tots els include's que s'afegeixen
- Les dues operacions de construcció són:
  - Compilació
  - Muntatge
- Això permet reescriure el graf de dependències entre fitxers fent una clausura del font i tots els *headers* dels que depèn un objecte.
- Aplicar aquesta transformació al graf no fa perdre informació de dependències rellevant per la construcció del projecte.

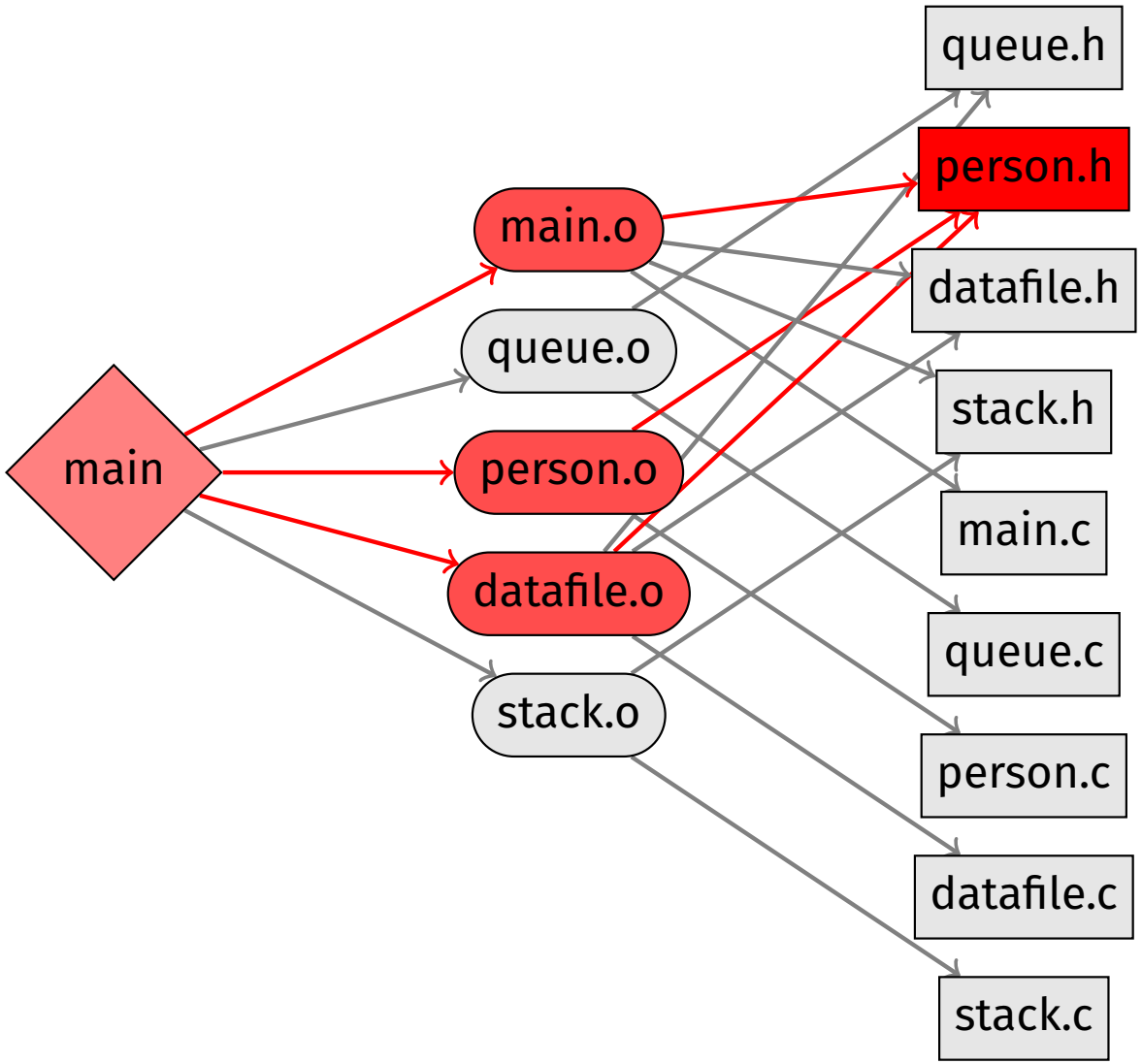
# El graf de dependències reduït



# Que cal reconstruir si...

- Si es modifica el fitxer `person.h`, per exemple, quina és la mínima feina que cal fer per recosntruir el projecte?
- Consultem el graf!

# Que cal reconstruir si...(2)



# Que cal reconstruir si...(3)

- Si es modifica el fitxer `person.h`, per exemple, quina és la mínima feina que cal fer per reconstruir el projecte?
- Consultem el graf!
- Cal procedir en el següent ordre:
  1. Compilar de nou `datafile.c`
  2. Compilar de nou `person.c`
  3. Compilar de nou `main.c`
  4. Muntar de nou l'executable `main`
- Això garanteix que, després de la modificació, el projecte torna a estar correctament actualitzat.

# Make

- `make` és una eina donar suport al manteniment de la construcció d'un projecte en un estat coherent.
- Des del punt de vista de l'usuari `make` és una ordre a la que se li demana, principalment, que actualitzi la construcció d'un projecte fent la mínima feina possible.
- Com a dades principals per fer la feina, `make` necessita el graf de dependències del projecte: l'usa per determinar què cal fer per reconstruir un projecte en el que hi ha hagut modificacions.

  Aquesta informació es desa en un fitxer de text que, tradicionalment, s'anomena `Makefile`.
- Per determinar si un fitxer s'ha modificat, `make` usa el timestamp i no té en compte el contingut del fitxer.

# Makefiles

- En un `Makefile` es descriu, entre altres coses, com és el graf de dependències d'un projecte.
- La sintaxi és basa en regles i cada regla té aquesta forma:

```
<target>: <dependències>
    <accions>
```

- Les accions són ordres de la shell i tant target com dependències són fitxers.
- Essencialment s'ha d'entendre com: si alguna de les dependències és més moderna que el target, aleshores el target és obsolet. Per reconstruir el target cal executar les accions.

# Makefiles (2)

- La regla següent:

```
stack.o: stack.c stack.h
    gcc -c stack.c
```

està dient: si `stack.c` i `stack.h` són més moderns que `stack.o` significa que algú ha modificat `stack.c` o `stack.h` i, per tant, `stack.o` és obsolet i s'ha de reconstruir. Per fer-ho cal executar l'ordre `gcc -c stack.c`

# Makefiles (3)

- Un `Makefile` bàsic corresponent al graf usat com exemple, seria el següent:

```
main: main.o datafile.o person.o stack.o queue.o
    gcc main.o datafile.o person.o stack.o queue.o -o main
stack.o: stack.c stack.h
    gcc -c stack.c
datafile.o: datafile.c datafile.h queue.h person.h
    gcc -c datafile.c
person.o: person.c person.h
    gcc -c person.c
queue.o: queue.c queue.h
    gcc -c queue.c
main.o: main.c
    gcc -c main.c
```

# Ús de make

- L'usuari fa servir l'ordre `make` demanant que actualitzi un fitxer concret.
- La sintaxi és:

  ```
  make <fitxer que cal actualitzar>
  ```

- Així, amb el `Makefile` anterior, si volem actualitzar el fitxer `main` escriuríem l'ordre de shell

  ```
  make main
  ```

- Si volguéssim actualitzar el fitxer `stack.o` escriuríem l'ordre de shell

  ```
  make stack.o
  ```

- Si escrivim l'ordre

  ```
  make
  ```

  sense paràmetres, aleshores s'entén que es vol actualitzar el primer target del `Makefile`. En l'exemple seria el mateix que haver escrit:

  ```
  make main
  ```

# Algoritme de make

- Què fa `make` quan se li demana que actualitzi un fitxer?
- En essència aplica el següent algoritme recursiu:

```
def update_file(f):
    R = get_rule_for(f)
    if R:
        for d in prereq(R):
            update_file(d)
        if any(t_stamp(d) > t_stamp(f) for d in prereq(R)):
            run_actions(R)
```