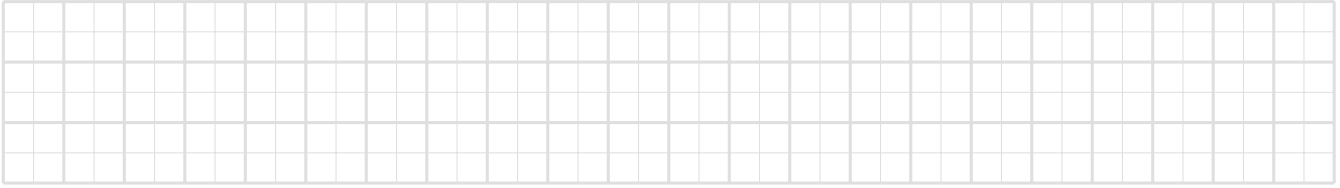




**Exercici 5 [1 punt].** Supposeu una aplicació que usa intensivament el mòdul `timer`. Imagineu que hi ha tants *callbacks* a executar que el temps d'execució supera el temps de `TICK`. Raoneu què succeiria en un cas així.



**Exercici 6 [3 punts].** El modul `timer` del projecte té aspectes millorables. Observeu aquesta situació patològica: (1) Es planifica una acció *A* per a ser executada al cap de 20s i torna l'identificador  $i_A$ ; (2) es cancel·la l'acció d'identificador  $i_A$ ; (3) Es planifica una acció *C* per ser executada al cap de 20s i casualment torna l'identificador  $i_A$ , que estava lliure. Es cancel·la de nou sobre l'identificador  $i_A$ , pensant que cancel·lem *A* però en realitat es cancel·la *C*!! Es un efecte indesitjat conseqüència del reaprofitament dels identificadors de les accions planificades. **CONSELL:** Feu-vos un diagrama de temps si no ho enteneu.

L'objectiu d'aquest exercici és implementar un petit `timer` simplificat però que resolgui el problema anterior. Cal que els identificadors siguin (quasi) únics.

Per simplificar l'exercici, assumiu que:

- Només hi haurà accions planificades simples, i.e.: les que s'executen una sola vegada al cap de  $n$  ticks.
- Hi ha una funció amb signatura **void** `interrupció(void)` que és cridada per la rutina d'interrupció cada tick. A l'exercici no cal, doncs, interactuar amb el maquinari sinó implementar aquesta funció.
- No aturarem mai les interrupcions del timer, encara que no hi hagi accions a fer.

Aquest timer tindrà només dues operacions públiques:

```
unsigned int set_accio(int nticks, callback_t f);  
void cancela(unsigned int id_accio);
```

La seva especificació és similar a la de la pràctica. `set_accio()` activa una acció planificada que al cap de  $nticks$  cridarà el callback `f`. Retorna un identificador `id` de l'acció. `cancela()` cancel·la l'acció que té identificador `id_accio`.

Per tal d'assignar identificadors (quasi) únics seguirem el següent algoritme:

- El primer identificador d'acció serà el 0. Si a l'acció  $A_i$  li hem donat l'identificador  $i$ , a la següent acció que planifiquem  $A_{i+1}$  li donarem l'identificador  $i + 1$ .
- Una vegada arribem a l'identificador `UINT_MAX` haurem esgotat tot el rang dels enters sense signe. Llavors, començarem de nou amb l'identificador 0 i anirem atorgant identificadors seqüencialment però *comprovant que l'identificador atorgat no està en ús*.

**ES DEMANA:**

- Que dissenyeu l'estructura de dades del timer d'acord amb el que s'ha dit abans.
- Que implementeu la funció **void** `interrupció(void)`.
- Que implementeu les dues operacions del mòdul.

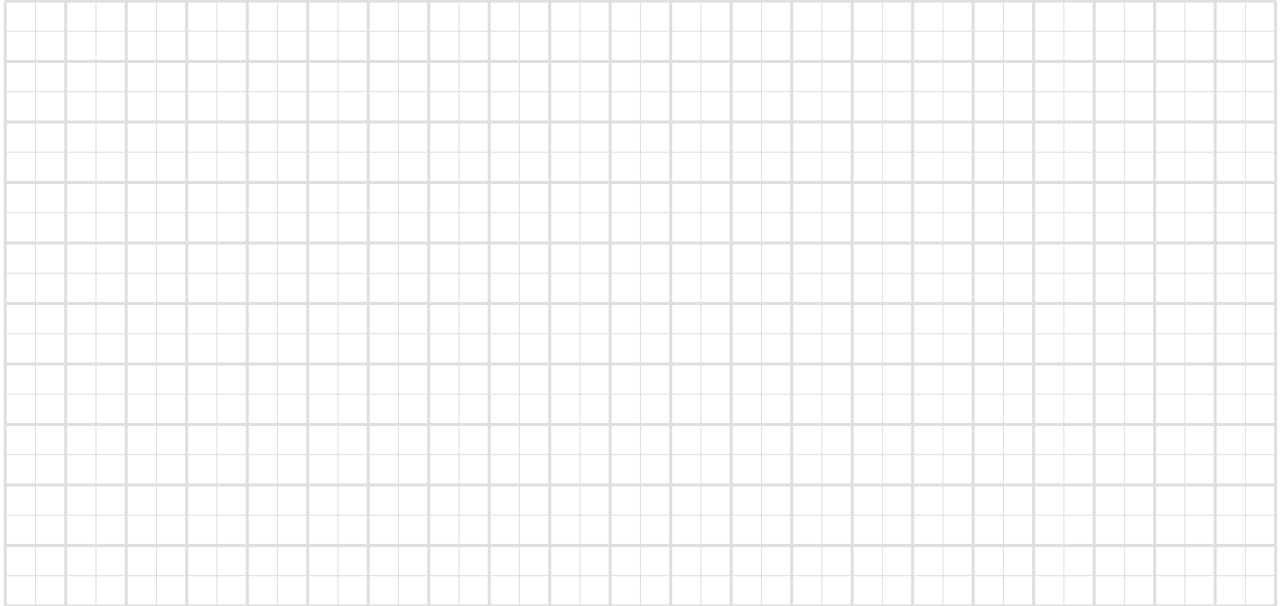
*Només cal que escriviu la part de la implementació del mòdul i no pas el header.*





**Exercici 7 [2 punts].** En el context del `timer` assumiu que l'instant en que es produeix l'interrupció del timer per servir el tick  $i$ , és  $t_i$ . Assumiu també que l'acció temporitzada  $A$  s'ha d'executar en aquest tick. Per la naturalesa del mòdul `timer`, l'instant en que s'executarà l'acció  $A$  no serà  $t_i$  exactament sinó  $t_i + \beta$ , sent  $\beta > 0$  un terme d'error que canvia segons les circumstàncies. L'objectiu d'aquest exercici és raonar sobre  $\beta$ .

- a)  $\beta$  no és un valor constant: generalment canvia per a cada acció i per cada instant de temps. Quins factors influeixen en el valor de  $\beta$ ?



- b) Quines característiques de  $\beta$  fan que el comportament del mòdul `timer` s'acosti a un timer ideal? Com influeixen els factors anteriors en aquestes característiques?

