# Clase pre-parcial

Arquitectura y Organización del Computador
Primer Cuatrimestre 2025
Link al ejercicio

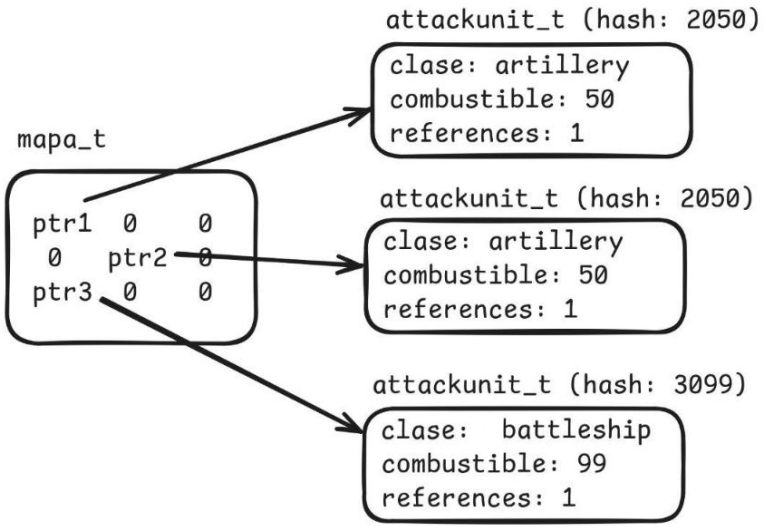
# 1er Recu 2c2024 - “Organized Wars”

En su turno, el jugador podrá colocar en un mapa de juego cuadriculado varias unidades de ataque de distintas clases. Cada clase tiene un valor inicial de combustible cargado, el cual utilizarán en una etapa posterior para realizar acciones como moverse, disparar bombas, etc. Además del combustible precargado, el jugador cuenta con una reserva extra de combustible que puede repartir entre las unidades que desee, potenciando ciertas unidades puntuales.

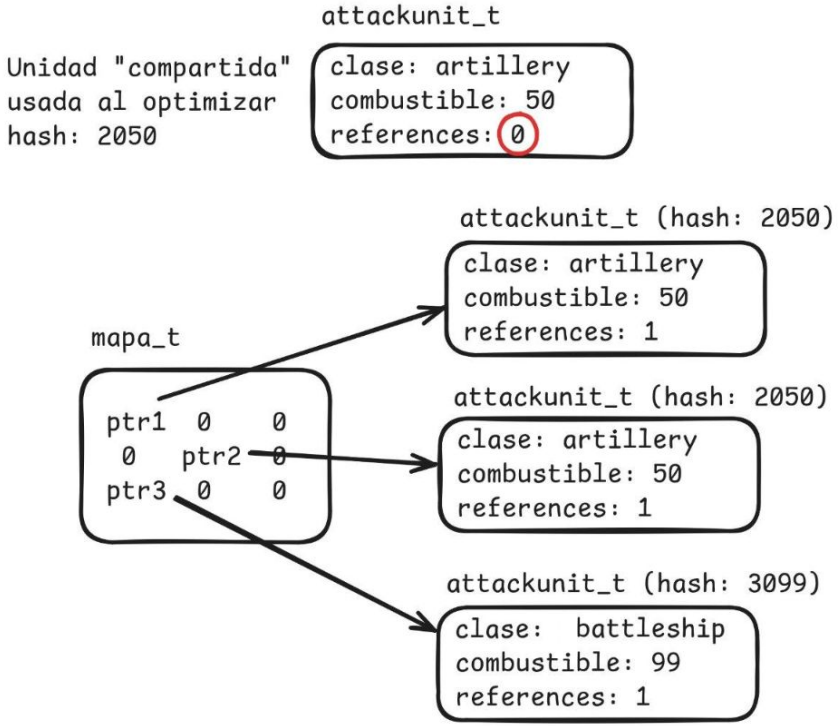
Dado que es común que los jugadores reposicionen y modifiquen los niveles de combustible de las unidades constantemente durante su turno, el sistema de nuestro juego funciona del siguiente modo:

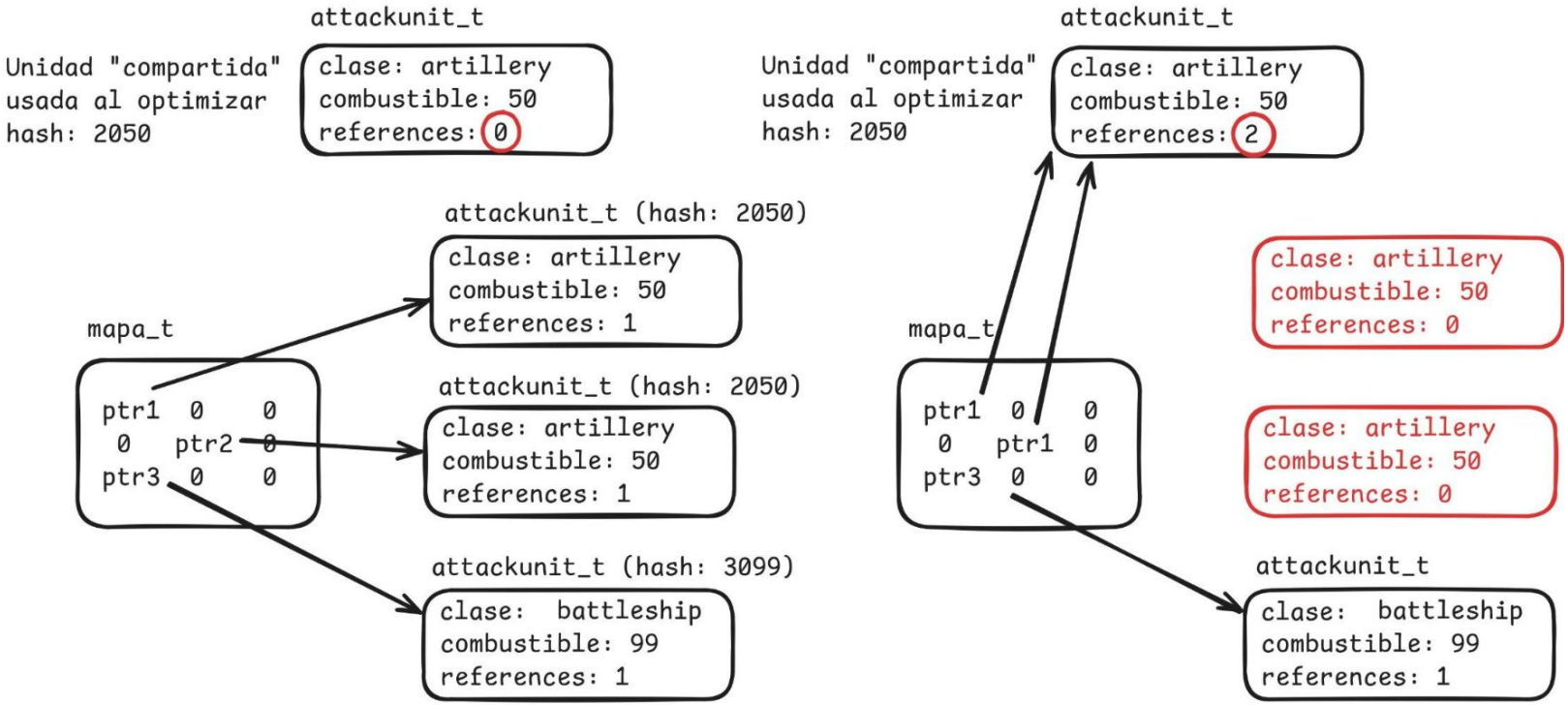
- Durante el transcurso del turno, cada unidad de ataque agregada se instancia independientemente.
- Al momento de finalizar el turno, se revisa que el jugador no haya asignado más combustible extra del que tenía disponible en su reserva. De haber asignado combustible correctamente, se efectiviza el final del turno.
- Una vez finalizado el turno, se corre una optimización que reemplaza todas las instancias independientes de unidades equivalentes por una única instancia "compartida"" (donde dos unidades son equivalentes si el resultado de aplicar una función de hash dada sobre cada una es el mismo).

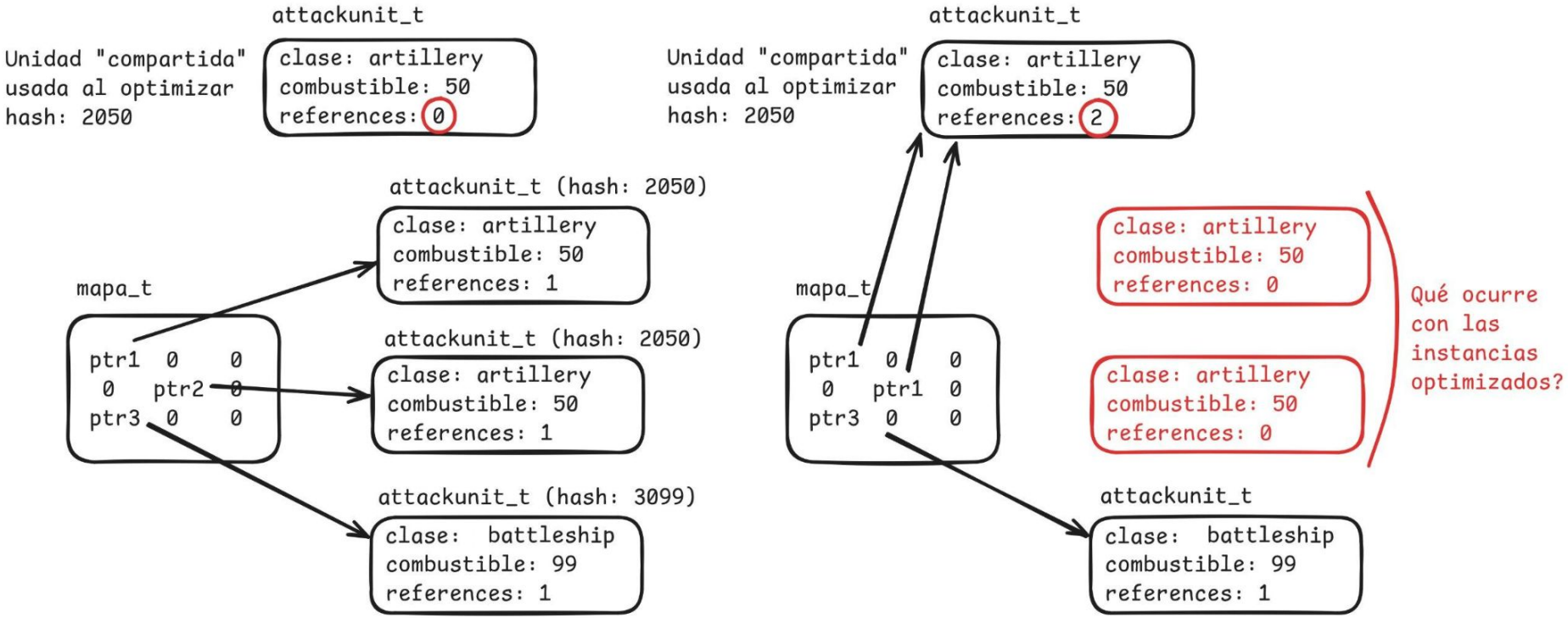
Tenemos un mapa representado con una grilla, donde en cada posición tenemos una unidad de combate. Cada una tiene una **clase** a la que pertenece, una cantidad de **combustible** asignada y la cantidad de posiciones del mapa donde aparece la misma unidad (posiciones del mapa que la **referencian**).



**Optimizar** el tablero consiste en reemplazar todas las instancias cuyos **hashes** coinciden por una única instancia **compartida** equivalente.




attackunit_t
Unidad "compartida"
usada al optimizar
hash: 2050
clase: artillery
combustible: 50
references: 0
attackunit_t (hash: 2050)
clase: artillery
combustible: 50
references: 1
mapa_t
ptr1 0 0
0 ptr2 0
ptr3 0 0
attackunit_t (hash: 2050)
clase: artillery
combustible: 50
references: 1
attackunit_t (hash: 3099)
clase: battleship
combustible: 99
references: 1
attackunit_t
Unidad "compartida"
usada al optimizar
hash: 2050
clase: artillery
combustible: 50
references: 2
clase: artillery
combustible: 50
references: 0
mapa_t
ptr1 0 0
0 ptr1 0
ptr3 0 0
clase: artillery
combustible: 50
references: 0
attackunit_t
clase: battleship
combustible: 99
references: 1


attackunit_t
Unidad "compartida"
usada al optimizar
hash: 2050
clase: artillery
combustible: 50
references: 0
attackunit_t (hash: 2050)
clase: artillery
combustible: 50
references: 1
mapa_t
ptr1 0 0
0 ptr2 0
ptr3 0 0
attackunit_t (hash: 2050)
clase: artillery
combustible: 50
references: 1
attackunit_t (hash: 3099)
clase: battleship
combustible: 99
references: 1
attackunit_t
Unidad "compartida"
usada al optimizar
hash: 2050
clase: artillery
combustible: 50
references: 2
mapa_t
ptr1 0 0
0 ptr1 0
ptr3 0 0
clase: artillery
combustible: 50
references: 0
clase: artillery
combustible: 50
references: 0
Qué ocurre
con las
instancias
optimizados?
attackunit_t
clase: battleship
combustible: 99
references: 1

# Ejercicios

```
void optimizar(mapa_t mapa, personaje_t* compartida, uint32_t *fun_hash)
```

a) Programar en lenguaje assembler la función que **optimiza** todas las unidades del mapa que sean equivalentes a aquella pasada por parámetro. La solución debe hacer uso apropiado de memoria, teniendo en cuenta que las referencias a unidades son guardadas en el mapa.

# Ejercicios

```
void optimizar(mapa_t mapa, personaje_t* compartida, uint32_t *fun_hash)
```

a) Programar en lenguaje assembler la función que **optimiza** todas las unidades del mapa que sean equivalentes a aquella pasada por parámetro. La solución debe hacer uso apropiado de memoria, teniendo en cuenta que las referencias a unidades son guardadas en el mapa.

***Pseudocódigo***:

- Calculamos hash de compartida
- Recorrer la matriz.
  - Si una unidad coincide con el hash parámetro, la reemplazamos.
    - Si tiene más de una referencia podemos solamente eliminar la referencia.
    - Si es la única instancia entonces debemos liberar la memoria también.

# Ejercicios

```
uint32_t contarCombustibleAsignado(mapa_t mapa, uint16_t (*fun_combustible)(char*))
```

b) Programar en lenguaje assembler la función que se utilizará para calcular, antes de finalizar el turno del jugador, la cantidad de combustible **de la reserva** que fue asignado por el jugador. La función pasada por parámetro, toma una clase de unidad y devuelve la cantidad de combustible base que le corresponde.

# Ejercicios

```
uint32_t contarCombustibleAsignado(mapa_t mapa, uint16_t (*fun_combustible)(char*))
```

b) Programar en lenguaje assembler la función que se utilizará para calcular, antes de finalizar el turno del jugador, la cantidad de combustible **de la reserva** que fue asignado por el jugador. La función pasada por parámetro, toma una clase de unidad y devuelve la cantidad de combustible base que le corresponde.

***Pseudocódigo:***

- Recorrer la matriz.
- Acumulamos el combustible agregado a cada unidad.
    - Para ello, debemos calcular la **diferencia** entre la **cantidad de combustible actual** de la unidad y la **cantidad de combustible base** de la misma, que se obtiene con **fun_combustible**.

# Ejercicios

```
void modificarUnidad(mapa_t mapa, uint8_t x, uint8_t y, void *fun_modificar)
```

c) Programar en lenguaje assembler la función que dada una posición en el mapa permita aplicar la función modificadora a la unidad en esa posición **únicamente**. Se debe tener en cuenta el caso en que se quiera modificar una unidad que previamente había sido optimizada, sin hacer uso excesivo o innecesario de recursos del sistema.

# Ejercicios

```
void modificarUnidad(mapa_t mapa, uint8_t x, uint8_t y, void *fun_modificar)
```

c) Programar en lenguaje assembler la función que dada una posición en el mapa permita aplicar la función modificadora a la unidad en esa posición **únicamente**. Se debe tener en cuenta el caso en que se quiera modificar una unidad que previamente había sido optimizada, sin hacer uso excesivo o innecesario de recursos del sistema.

***Pseudocódigo:***

- Obtener la unidad indicada por las coordenadas (x, y).
- Aplicar la función de modificación en caso de ser posible.
  - Si es única, podemos modificarla directamente.
  - Si no lo es, debemos crear la nueva instancia modificada.

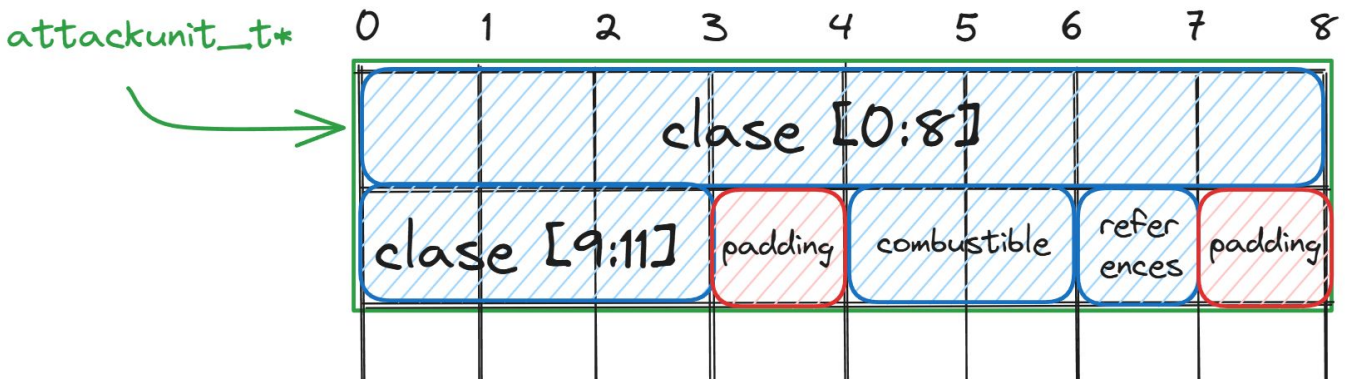
# Implementación en C

¿Cómo se ve una unidad de ataque en memoria?

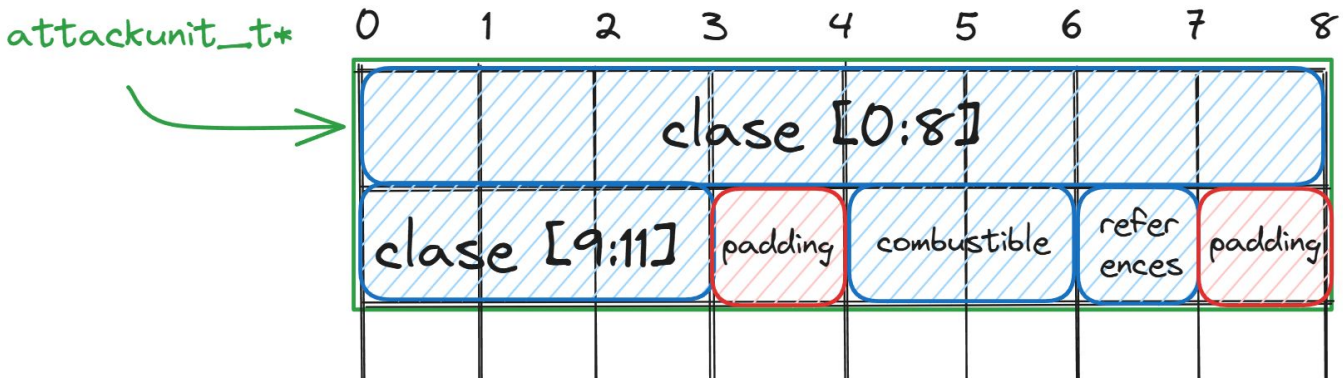
```
5     /**
6      * Una unidad de ataque (tanques, barcos de batalla, etc.) en nuestro videojuego.
7      *
8      * Campos:
9      *    - clase:          El nombre de la clase.
10     *    - combustible:    La cantidad de combustible disponible de la unidad.
11     *    - references:     La cantidad de referencias a la unidad en el mapa.
12     */
13    typedef struct {
14            char clase[11];       //asmdef_offset:ATTACKUNIT_CLASE
15            uint16_t combustible; //asmdef_offset:ATTACKUNIT_COMBUSTIBLE
16            uint8_t references;   //asmdef_offset:ATTACKUNIT_REFERENCES
17    } attackunit_t; //asmdef_size:ATTACKUNIT_SIZE
18
```

# ¿Cómo se ve una unidad de ataque en memoria?

```
5   /**
6    * Una unidad de ataque (tanques, barcos de batalla, etc.) en nuestro videojuego.
7    *
8    * Campos:
9    *   - clase:          El nombre de la clase.
10   *   - combustible:    La cantidad de combustible disponible de la unidad.
11   *   - references:     La cantidad de referencias a la unidad en el mapa.
12   */
13  typedef struct {
14          char clase[11];       //asmdef_offset:ATTACKUNIT_CLASE
15          uint16_t combustible; //asmdef_offset:ATTACKUNIT_COMBUSTIBLE
16          uint8_t references;   //asmdef_offset:ATTACKUNIT_REFERENCES
17  } attackunit_t; //asmdef_size:ATTACKUNIT_SIZE
18
```



# ¿Cómo se ve una unidad de ataque en memoria?




```
37   ;########### ESTOS SON LOS OFFSETS Y TAMAÑO DE LOS STRUCTS
38   ; Completar las definiciones (serán revisadas por ABI enforcer):
39   ATTACKUNIT_CLASE EQU 0
40   ATTACKUNIT_COMBUSTIBLE EQU 12
41   ATTACKUNIT_REFERENCES EQU 14
42   ATTACKUNIT_SIZE EQU 16
```

```
34  void optimizar(mapa_t mapa, attackunit_t* compartida, uint32_t (*fun_hash)(attackunit_t*)) {
35      uint32_t hash_compartida = fun_hash(compartida);
36      for (uint64_t i = 0; i < 255; i++) {
37          for (uint64_t j = 0; j < 255; j++) {
38              attackunit_t* actual = mapa[i][j];
39              if (actual == NULL || compartida == actual) {
40                  continue;
41              }
42              uint32_t hash_actual = fun_hash(actual);
43              if (hash_actual == hash_compartida) {
44                  compartida->references++;
45                  actual->references--;
46                  mapa[i][j] = compartida;
47              }
48              if (actual->references == 0) {
49                  free(actual);
50              }
51          }
52      }
53  }
```

- Calculamos hash de **compartida**
- Recorremos la matriz
- Comparamos hashes
- Si tiene más de una referencia podemos solamente eliminar la referencia.
- Si es la única instancia entonces debemos liberar la memoria también.

```
58  uint32_t contarCombustibleAsignado(mapa_t mapa, uint16_t (*fun_combustible)(char*)) {
59      uint32_t total_combustible_utilizado = 0;
60      for (uint64_t i = 0; i < 255; i++) {
61          for (uint64_t j = 0; j < 255; j++) {
62              attackunit_t* actual = mapa[i][j];
63              if (actual == NULL) {
64                  continue;
65              }
66              uint32_t combustible_base = (uint32_t) fun_combustible(actual->clase);
67              uint32_t combustible_utilizado = actual->combustible - combustible_base;
68              total_combustible_utilizado += combustible_utilizado;
69          }
70      }
71      return total_combustible_utilizado;
72  }
```

- Recorremos la matriz.
- Calculamos combustible base.
- Acumulamos la diferencia.

```
77  void modificarUnidad(mapa_t mapa, uint8_t x, uint8_t y, void (*fun_modificar)(attackunit_t*)) {
78      attackunit_t* unidad_actual = mapa[x][y];
79      // Reescritura más fácil de traducir a assembler.
80      // Recuerden que en la aritmética de punteros de C, se multiplica todo el resultado
81      // implícatemente por sizeof(attackunit_t*) (8 bytes).
82      //attackunit_t* a_modificar = *((attackunit_t**) mapa + x * 255 + y); // 255 -> COLUMNAS
83      if (unidad_actual == NULL) {
84          return;
85      }
86      if (unidad_actual->references > 1) {
87          attackunit_t* nueva_unidad = malloc(sizeof(attackunit_t));
88          unidad_actual->references--;
89          *nueva_unidad = *unidad_actual;
90          nueva_unidad->references = 1;
91
92          mapa[x][y] = nueva_unidad;
93          // reescritura más fácil de traducir a assembler
94          //*((attackunit_t**) mapa + x * 255 + y) = nueva_unidad; // 255 -> COLUMNAS
95          unidad_actual = nueva_unidad;
96      }
97      fun_modificar(unidad_actual);
98  }
```

- Obtener la unidad en (x, y)
- Si no es única, creamos la nueva instancia
- Aplicamos modificación.

# Implementación en ASM

# Ejercicio a)

```
global optimizar
optimizar:
        ; Te recomendamos llenar una tablita acá con cada
        ; ubicación según la convención de llamada. Prest
        ; valores son de 64 bits y qué valores son de 32
        ;
        ; rdi = mapa_t           mapa
        ; rsi = attackunit_t*    compartida
        ; rdx = uint32_t*        fun_hash(attackunit_t*)
        push rbp
        mov rbp, rsp
        push r12
        push r13
        push r14
        push r15
        push rbx
        sub rsp, 8 ; pila alineada

        mov r12, rdi ; mapa
        mov r13, rsi ; compartida
        mov r14, rdx ; fun_hash
        xor r15, r15 ; iterador
```

```
        ; calculo en hash de la unidad compartida
        mov rdi, r13
        call r14
        mov ebx, eax ; hash compartida

.loop:
        mov rdi, [r12 + 8 * r15] ; unidad actual
        cmp rdi, 0 ; ¿Es un null pointer?
        je .nextIteration

        cmp rdi, r13 ; ¿Es compartida == actual?
        je .nextIteration

        call r14 ; la unidad actual ya está en rdi
        cmp eax, ebx ; ¿Es hash_compartida == hash_actual?
        jne .nextIteration

        ; actualizo los contadores de referencias
        inc BYTE [r13 + ATTACKUNIT_REFERENCES] ; compartida->references++
        mov rdi, [r12 + 8 * r15] ; unidad actual
        dec BYTE [rdi + ATTACKUNIT_REFERENCES] ; actual->references--
        mov [r12 + 8 * r15], r13 ; mapa[i][j] = compartida
```

```
        ; ¿tengo que borrar la unidad que acabo de
        cmp BYTE [rdi + ATTACKUNIT_REFERENCES], 0
        jne .nextIteration
        call free ; la unidad actual ya está en rdi

.nextIteration:
        inc r15
        cmp r15, FILAS * COLUMNAS
        jl .loop

        add rsp, 8
        pop rbx
        pop r15
        pop r14
        pop r13
        pop r12
        pop rbp
        ret
```

# Ejercicio b)

```
global contarCombustibleAsignado
contarCombustibleAsignado:
        ; rdi = mapa_t           mapa
        ; rsi = uint16_t*        fun_combustible(char*)
        push rbp
        mov rbp, rsp
        push r12
        push r13
        push r14
        push r15
        push rbx
        sub rsp, 8 ; pila alineada

        mov r15, rdi ; mapa
        mov r14, rsi ; fun_combustible
        xor r13, r13 ; total_combustible_utilizado
        xor r12, r12 ; iterador
```

```
.loop:
        mov rsi, [r15 + 8 * r12] ; unidad actual
        cmp rsi, 0 ; ¿Es un null pointer?
        je .nextIteration

        movzx ebx, WORD [rsi + ATTACKUNIT_COMBUSTIBLE] ; actual->combustible

        mov rdi, rsi + ATTACKUNIT_CLASE ; el puntero a donde comienza el string actual->clase
        call r14
        movzx eax, ax ; combustible_base

        sub ebx, eax ; combustible_utilizado = actual->combustible - combustible_base
        add r13d, ebx ; total_combustible_utilizado += combustible_utilizado

.nextIteration:
        inc r12
        cmp r12, FILAS * COLUMNAS
        jl .loop

        mov rax, r13

        add rsp, 8
        pop rbx
        pop r15
        pop r14
        pop r13
        pop r12
        pop rbp
        ret
```

# Ejercicio c)

```asm
global modificarUnidad
modificarUnidad:
        ; rdi = mapa_t           mapa
        ; sil  = uint8_t          x
        ; dl  = uint8_t          y
        ; rcx = void*            fun_modificar(attackunit_t*)
        push rbp
        mov rbp, rsp
        push r13
        push r14
        push r15
        sub rsp, 8 ; pila alineada

        ; me muevo a la posición del mapa que quiero modificar
        movzx rsi, sil ; extiendo x a 8 bytes
        movzx rdx, dl ; extiendo y a 8 bytes
```

```asm
; me muevo a la posición del mapa que quiero modificar
movzx rsi, sil ; extiendo x a 8 bytes
movzx rdx, dl ; extiendo y a 8 bytes

; acá puedo usar una multiplicación con signo porque en el fondo
; estoy multiplicando dos uint8_t. Nunca voy a tener overflow.
; La instrucción imul es un poco más cómoda de usar que mul.
; También se podría multiplicar por 256 shifteando a izquierda y restando 1 :)
imul rsi, COLUMNAS ; x * COLUMNAS -> rsi
add rdx, rsi ; y + x * 255 -> rdx
shl rdx, 3 ; multiplico rdx por 8 (2^3) para obtener (y + 255 * x) * 8 -> rdx
add rdi, rdx ; offset en el mapa donde tengo que moidificar

mov r15, rdi ; posición a modificar en el mapa. Esto es efectivamente attackunit_t**
mov r14, rcx ; fun_modificar

mov r13, [r15] ; r13 es la unidad a modificar
cmp r13, 0 ; ¿Es un puntero NULL?
je .end
```

# Ejercicio c)

```
; me muevo a la posición del mapa que quiero modificar
movzx rsi, sil ; extiendo x a 8 bytes
movzx rdx, dl ; extiendo y a 8 bytes

; acá puedo usar una multiplicación con signo porque en el fondo
; estoy multiplicando dos uint8_t. Nunca voy a tener overflow.
; La instrucción imul es un poco más cómoda de usar que mul.
; También se podría multiplicar por 256 shifteando a izquierda y restando 1 :)
imul rsi, COLUMNAS ; x * COLUMNAS -> rsi
add rdx, rsi ; y + x * 255 -> rdx
shl rdx, 3 ; multiplico rdx por 8 (2^3) para obtener (y + 255 * x) * 8 -> rdx
add rdi, rdx ; offset en el mapa donde tengo que moidificar

mov r15, rdi ; posición a modificar en el mapa. Esto es efectivamente attackunit_t**
mov r14, rcx ; fun_modificar

mov r13, [r15] ; r13 es la unidad a modificar
cmp r13, 0 ; ¿Es un puntero NULL?
je .end
```

```
mov r9b, [r13 + ATTACKUNIT_REFERENCES]
cmp r9b, 1
jle .skipCopy
; si la untidad que tengo en el mapa tiene más de una referencia, la tengo que copiar.
; Solo quiero modificar esta posición en el tablero del juego.

; decremento las referencias de esta unidad (esta posición será reemplazada con una copia)
dec BYTE [r13 + ATTACKUNIT_REFERENCES]

; reservo memoria en el heap para la nueva unidad
mov rdi, ATTACKUNIT_SIZE
call malloc ; rax contiene un puntero a la nueva attackunit_t
mov rdi, [r13] ; copio bytes 0 a 8
mov [rax], rdi
mov rdi, [r13 + 8] ; copio bytes 8 a 16
mov [rax + 8], rdi

mov [rax + ATTACKUNIT_REFERENCES], BYTE 1 ; esta unidad se referencia solo en esta posición
mov [r15], rax ; escribo en el mapa el puntero a la nueva unidad
```

# Ejercicio c)

```
mov r9b, [r13 + ATTACKUNIT_REFERENCES]
cmp r9b, 1
jle .skipCopy
; si la untidad que tengo en el mapa tiene más de una referencia, la tengo que copiar.
; Solo quiero modificar esta posición en el tablero del juego.

; decremento las referencias de esta unidad (esta posición será reemplazada con una copia)
dec BYTE [r13 + ATTACKUNIT_REFERENCES]

; reservo memoria en el heap para la nueva unidad
mov rdi, ATTACKUNIT_SIZE
call malloc ; rax contiene un puntero a la nueva attackunit_t
mov rdi, [r13] ; copio bytes 0 a 8
mov [rax], rdi
mov rdi, [r13 + 8] ; copio bytes 8 a 16
mov [rax + 8], rdi

mov [rax + ATTACKUNIT_REFERENCES], BYTE 1 ; esta unidad se referencia solo en esta posición
mov [r15], rax ; escribo en el mapa el puntero a la nueva unidad
```

```
.skipCopy:
        ; modifico la unidad que está en la posición actual del tablero
        mov rdi, [r15]
        call r14

.end:
        add rsp, 8
        pop r15
        pop r14
        pop r13
        pop rbp
        ret
```