Memoria Dinámica

Organización del Computador II

David González Márquez → Florencia Zanollo

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

24-08-2017

Hoy

¿Qué vamos a ver? Y ¿por qué?

- Estructuras Permite combinar datos de diferentes tipos y guardarlos bajo una estructura definida por el usuario.
- Memoria Dinámica Permite solicitar memoria "on-demand" (en tiempo de ejecución).
- Listas Estructura de datos muy utilizada.
- Ejercicios Hay que practicar!

Estructuras

`struct`

Definen un patrón de acceso a un area determinada de memoria

`struct` *nombre_de_la_estructura* {

tipo_1 *nombre_del_campo_1* ;

...

tipo_n *nombre_del_campo_n* ;

}

Estructuras

struct

Definen un patrón de acceso a un area determinada de memoria

```
struct nombre_de_la_estructura {
            tipo_1      nombre_del_campo_1 ;
            ...
            tipo_n      nombre_del_campo_n ;
}
```

Ejemplos:

```
struct p2D {
 int x;
 int y;
};
```

```
struct alumno {
 char* nombre;
 char comision;
 int dni;
};
```

struct

Definen un patrón de acceso a un area determinada de memoria

```
struct nombre_de_la_estructura {
            tipo_1      nombre_del_campo_1 ;
            ...
            tipo_n      nombre_del_campo_n ;
}
```

Ejemplos: → `SIZE`

```
struct p2D {
 int x;          → 4
 int y;          → 4
};
```

```
struct alumno {
 char* nombre;    → 8
 char comision;   → 1
 int dni;         → 4
};
```

struct

Definen un patrón de acceso a un area determinada de memoria

```
struct nombre_de_la_estructura {
        tipo_1      nombre_del_campo_1 ;
        ...
        tipo_n      nombre_del_campo_n ;
}
```

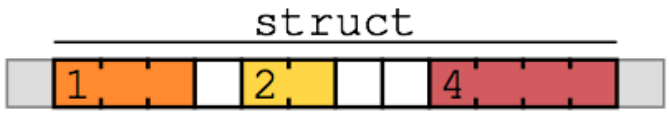
Ejemplos: → SIZE ⇒ OFFSET

```
struct p2D {                    
 int x;          → 4   ⇒ 0
 int y;          → 4   ⇒ 4
};                     ⇒ 8
```

```
struct alumno {                    
 char* nombre;     → 8   ⇒ 0
 char comision;    → 1   ⇒ 8
 int dni;          → 4   ⇒ 12
};                       ⇒ 16
```

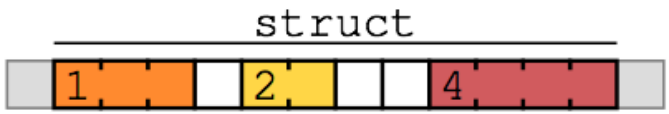
Alineación

- Alineación en los campos del struct:
  Cada campo esta alineado a su tamaño dentro del struct

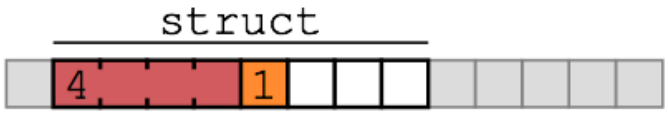


Alineación

- Alineación en los campos del struct:
  Cada campo esta alineado a su tamaño dentro del struct

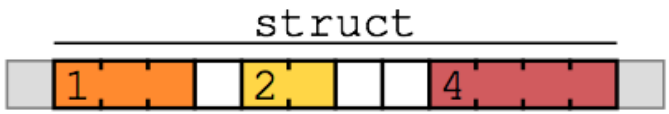


- Alineación del struct:
  Se alinea al tamaño del campo mas grande del struct

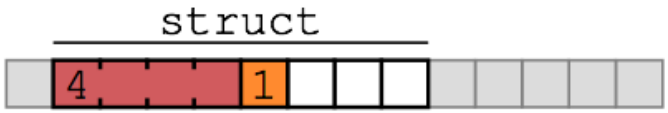


Alineación

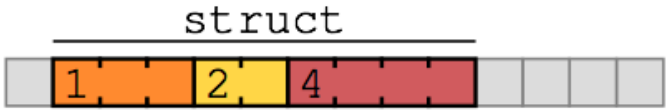
- Alineación en los campos del struct:
  Cada campo esta alineado a su tamaño dentro del struct



- Alineación del struct:
  Se alinea al tamaño del campo mas grande del struct



- `__attribute__((packed))`:
  Indica que el struct no va a ser alinenado



Ejemplos

```
struct alumno {
 char* nombre;
 char comision;
 int dni;
};

struct alumno2 {
 char comision;
 char* nombre;
 int dni;
};
```

Ejemplos: → SIZE

```
struct alumno {
 char* nombre;      → 8
 char comision;     → 1
 int dni;           → 4
};

struct alumno2 {
 char comision;     → 1
 char* nombre;      → 8
 int dni;           → 4
};
```

Ejemplos: $\rightarrow$ SIZE $\Rightarrow$ OFFSET

```
struct alumno {
 char* nombre;      → 8   ⇒ 0
 char comision;     → 1   ⇒ 8
 int dni;           → 4   ⇒ 12
};                        ⇒ 16

struct alumno2 {
 char comision;     → 1   ⇒ 0
 char* nombre;      → 8   ⇒ 8
 int dni;           → 4   ⇒ 16
};                        ⇒ 24
```

Ejemplos: → SIZE ⇒ OFFSET

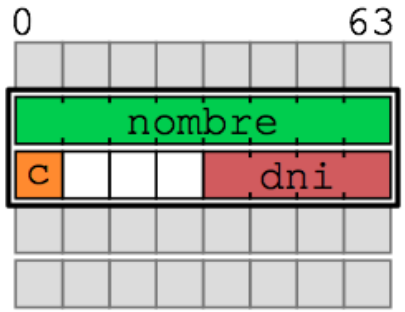
```
struct alumno {
 char* nombre;      → 8      ⇒ 0
 char comision;     → 1      ⇒ 8
 int dni;           → 4      ⇒ 12
};                           ⇒ 16

struct alumno2 {
 char comision;     → 1      ⇒ 0
 char* nombre;      → 8      ⇒ 8
 int dni;           → 4      ⇒ 16
};                           ⇒ 24

struct alumno3 {
 char* nombre;      → 8
 int dni;           → 4
 char comision;     → 1
} __attribute__((packed));
```

Ejemplos: → SIZE ⇒ OFFSET

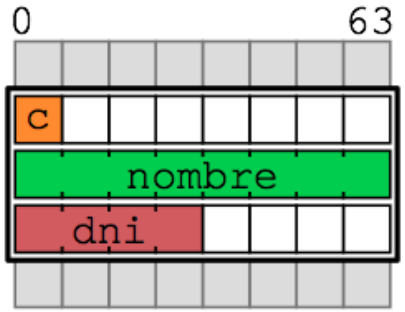
```
struct alumno {
 char* nombre;      → 8      ⇒ 0
 char comision;     → 1      ⇒ 8
 int dni;           → 4      ⇒ 12
};                           ⇒ 16

struct alumno2 {
 char comision;     → 1      ⇒ 0
 char* nombre;      → 8      ⇒ 8
 int dni;           → 4      ⇒ 16
};                           ⇒ 24

struct alumno3 {
 char* nombre;      → 8      ⇒ 0
 int dni;           → 4      ⇒ 8
 char comision;     → 1      ⇒ 12
} __attribute__((packed));   ⇒ ???
```

Ejemplos: → SIZE ⇒ OFFSET

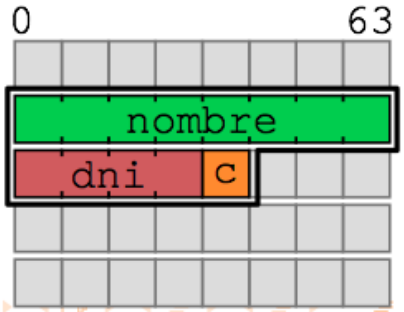
```
struct alumno {
 char* nombre;      → 8      ⇒ 0
 char comision;     → 1      ⇒ 8
 int dni;           → 4      ⇒ 12
};                           ⇒ 16
```




```
struct alumno2 {
 char comision;     → 1      ⇒ 0
 char* nombre;      → 8      ⇒ 8
 int dni;           → 4      ⇒ 16
};                           ⇒ 24
```




```
struct alumno3 {
 char* nombre;      → 8      ⇒ 0
 int dni;           → 4      ⇒ 8
 char comision;     → 1      ⇒ 12
} __attribute__((packed));   ⇒ 13
```



Uso

Definición:

```
struct alumno {
        char* nombre;
        char comision;
        int dni;
};
```

Uso

Definición:

```
struct alumno {
        char* nombre;
        char comision;
        int dni;
};
```

Uso en C:

```
struct alumno alu;
alu.nombre = ‘‘carlos’’;
alu.dni = alu.dni + 10;
alu.comision = ‘a’;
```

Uso en ASM:

```
%define offset_nombre 0
%define offset_comision 8
%define offset_dni 12
mov rsi, ptr_struct
mov rbx, [rsi+offset_nombre]
mov al, [rsi+offset_comision]
mov edx, [rsi+offset_dni]
```

Ejercicio

En el archivo de la clase tienen el ejercicio 1 con el siguiente struct:

```
struct alumno {
 short comision;
 char * nombre;
 int edad;
};
```

Implementen la función mostrar_alumno(struct alumno * un_alumno) que toma el struct alumno e imprime por pantalla sus valores.

Listas

Estructuras:

```
struct lista {
 nodo *primero;
};
```

```
struct nodo {
 int dato;
 nodo *prox;
};
```

Listas

Estructuras: → SIZE ⇒ OFFSET

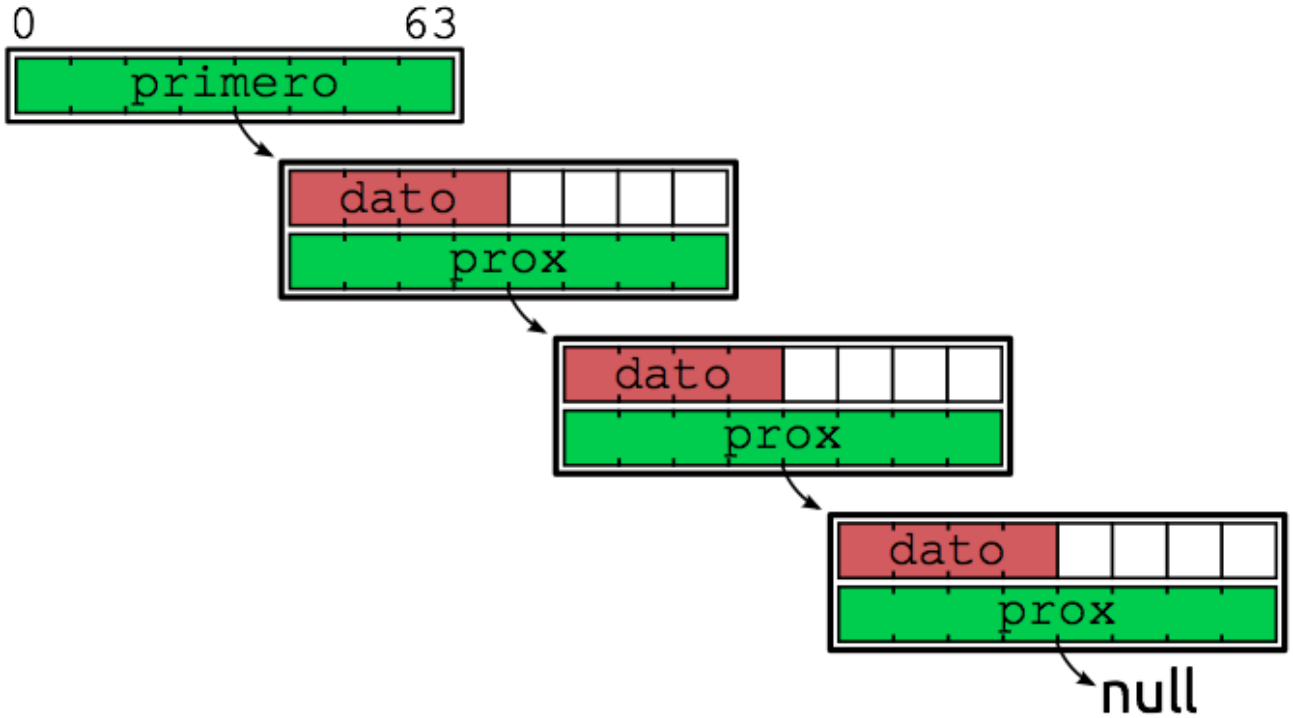
```
struct lista {                  → 8   ⇒ 0
 nodo *primero;                       ⇒ 8
};
```

```
struct nodo {
 int dato;          → 4   ⇒ 0
 nodo *prox;        → 8   ⇒ 8
};                         ⇒ 16
```

Estructuras: → SIZE ⇒ OFFSET

```
struct lista {                    → 8   ⇒ 0
 nodo *primero;                         ⇒ 8
};
```

```
struct nodo {
 int dato;          → 4   ⇒ 0
 nodo *prox;        → 8   ⇒ 8
};                         ⇒ 16
```



Listas - Agregar

¿Cómo agrego elementos a la lista?

Listas - Agregar

¿Cómo agrego elementos a la lista?
Necesito crear un nuevo nodo...

Listas - Agregar

¿Cómo agrego elementos a la lista?
Necesito crear un nuevo nodo...

- ¿Puedo definir el nuevo nodo como variable estática?

Listas - Agregar

¿Cómo agrego elementos a la lista?
Necesito crear un nuevo nodo...

- ¿Puedo definir el nuevo nodo como variable estática?
- Si defino el nuevo nodo como variable local.

Listas - Agregar

¿Cómo agrego elementos a la lista?
Necesito crear un nuevo nodo...

- ¿Puedo definir el nuevo nodo como variable estática?
- Si defino el nuevo nodo como variable local.
  - ¿Dónde está la variable local?

Listas - Agregar

¿Cómo agrego elementos a la lista?
Necesito crear un nuevo nodo...

- ¿Puedo definir el nuevo nodo como variable estática?
- Si defino el nuevo nodo como variable local.
  - ¿Dónde está la variable local?
  - ¿Qué pasa al retornar de la función?

Repaso - Variables estáticas y de pila

Variable estática

Se asigna en un espacio de memoria reservado de manera estática que vive durante toda la ejecución del programa.

Repaso - Variables estáticas y de pila

Variable estática

Se asigna en un espacio de memoria reservado de manera estática que vive durante toda la ejecución del programa.

Ej. ASM:

```
section .data:
numero: dd 10
section .rodata:
literal: db ‘‘Orga 2’’
section .bss
otro_numero: resd 1
```

Ej. C:

```
int numero = 10;
int otro_numero;
char *literal = ‘‘Orga 2’’
int main() {return 0;}
```

Repaso - Variables estáticas y de pila

Variable estática

Se asigna en un espacio de memoria reservado de manera estática que vive durante toda la ejecución del programa.

Variable en la pila

Esta asignada dentro del espacio de pila del programa, puede existir solo en el contexto de ejecución de una función.

Repaso - Variables estáticas y de pila

Variable estática

Se asigna en un espacio de memoria reservado de manera estática que vive durante toda la ejecución del programa.

Variable en la pila

Esta asignada dentro del espacio de pila del programa, puede existir solo en el contexto de ejecución de una función.

Ej. ASM: `sub rsp, 8` (ahora `rsp` apunta a nuestra variable `numero`)

Ej. C: `int* numero;`

Listas - Agregar

¿Cómo agrego elementos a la lista?
Necesito crear un nuevo nodo...

- ¿Puedo definir el nuevo nodo como variable estática?

Listas - Agregar

¿Cómo agrego elementos a la lista?
Necesito crear un nuevo nodo...

- ¿Puedo definir el nuevo nodo como variable estática?
  No, la memoria estática se definió al comienzo del programa y ya no puede agregarse más.

Listas - Agregar

¿Cómo agrego elementos a la lista?
Necesito crear un nuevo nodo...

- ¿Puedo definir el nuevo nodo como variable estática?
  No, la memoria estática se definió al comienzo del programa y ya no puede agregarse más.
- Si defino el nuevo nodo como variable local.
  - ¿Dónde está la variable local?

Listas - Agregar

¿Cómo agrego elementos a la lista?
Necesito crear un nuevo nodo...

- ¿Puedo definir el nuevo nodo como variable estática?
  No, la memoria estática se definió al comienzo del programa y ya no puede agregarse más.
- Si defino el nuevo nodo como variable local.
  - ¿Dónde está la variable local?
    Se encuentra en la pila.
  - ¿Qué pasa al retornar de la función?

Listas - Agregar

¿Cómo agrego elementos a la lista?
Necesito crear un nuevo nodo...

- ¿Puedo definir el nuevo nodo como variable estática?
  No, la memoria estática se definió al comienzo del programa y ya no puede agregarse más.
- Si defino el nuevo nodo como variable local.
  - ¿Dónde está la variable local?
    Se encuentra en la pila.
  - ¿Qué pasa al retornar de la función?
    El nuevo nodo ya no existe y el puntero quedó apuntando a basura.

Listas - Agregar

¿Cómo agrego elementos a la lista?
Necesito crear un nuevo nodo...

- ¿Puedo definir el nuevo nodo como variable estática?
  No, la memoria estática se definió al comienzo del programa y ya no puede agregarse más.
- Si defino el nuevo nodo como variable local.
  - ¿Dónde está la variable local?
    Se encuentra en la pila.
  - ¿Qué pasa al retornar de la función?
    El nuevo nodo ya no existe y el puntero quedó apuntando a basura.

Necesitamos algo más; una forma de pedir memoria que no muera al retornar de la función, es decir, una forma de pedir memoria dinámicamente (mientras el programa está corriendo).

Memoria Dinámica

Variable dinámica

Esta asignada en un espacio de memoria solicitado al sistema operativo mediante una biblioteca de funciones, estas permiten solicitar y liberar memoria. (`malloc`)

Memoria Dinámica

Variable dinámica

Esta asignada en un espacio de memoria solicitado al sistema operativo mediante una biblioteca de funciones, estas permiten solicitar y liberar memoria. (`malloc`)

Solicitar memoria

```
void *malloc(size_t size)
```

Asigna `size` bytes de memoria y nos devuelve su posición.

Memoria Dinámica

Variable dinámica

Esta asignada en un espacio de memoria solicitado al sistema operativo mediante una biblioteca de funciones, estas permiten solicitar y liberar memoria. (`malloc`)

Solicitar memoria

```
void *malloc(size_t size)
```

Asigna `size` bytes de memoria y nos devuelve su posición.

Liberar memoria

```
void free(void *pointer)
```

Libera la memoria en `pointer`, previamente solicitada por `malloc`.

Memoria Dinámica

Variable dinámica

Esta asignada en un espacio de memoria solicitado al sistema operativo mediante una biblioteca de funciones, estas permiten solicitar y liberar memoria. (`malloc`)

Solicitar memoria

```
void *malloc(size_t size)
```

Asigna `size` bytes de memoria y nos devuelve su posición.

Liberar memoria

```
void free(void *pointer)
```

Libera la memoria en `pointer`, previamente solicitada por `malloc`.

"With a great power comes a great responsibility"

Memoria Dinámica

Solicitar memoria desde ASM

```
mov rdi, 24    ; solicitamos 24 Bytes de memoria
call malloc    ; llamamos a malloc que devuelve en rax
               ; el puntero a la memoria solicitada
```

Liberar memoria desde ASM

```
mov rdi, rax   ; rdi contiene el puntero a la memoria
               ; entregado por malloc al solicitar memoria
call free      ; llamamos a free
```

Memoria Dinámica

Solicitar memoria desde ASM

```
mov rdi, 24    ; solicitamos 24 Bytes de memoria
call malloc    ; llamamos a malloc que devuelve en rax
               ; el puntero a la memoria solicitada
```

Liberar memoria desde ASM

```
mov rdi, rax   ; rdi contiene el puntero a la memoria
               ; entregado por malloc al solicitar memoria
call free      ; llamamos a free
```

"With a great power comes a great responsibility"

(Si, también en ASM)

IMPORTANTE

Si se solicita memoria utilizando `malloc` entonces se DEBE liberar utilizando `free`. Toda memoria que se solicite DEBE ser liberada durante la ejecución del programa.

IMPORTANTE

Si se solicita memoria utilizando `malloc` entonces se DEBE liberar utilizando `free`. Toda memoria que se solicite DEBE ser liberada durante la ejecución del programa.

Caso contrario se PIERDE MEMORIA

IMPORTANTE

Si se solicita memoria utilizando `malloc` entonces se DEBE liberar utilizando `free`. Toda memoria que se solicite DEBE ser liberada durante la ejecución del programa.

Caso contrario se PIERDE MEMORIA

Para detectar problemas en el uso de la memoria se puede utilizar:

`Valgrind`

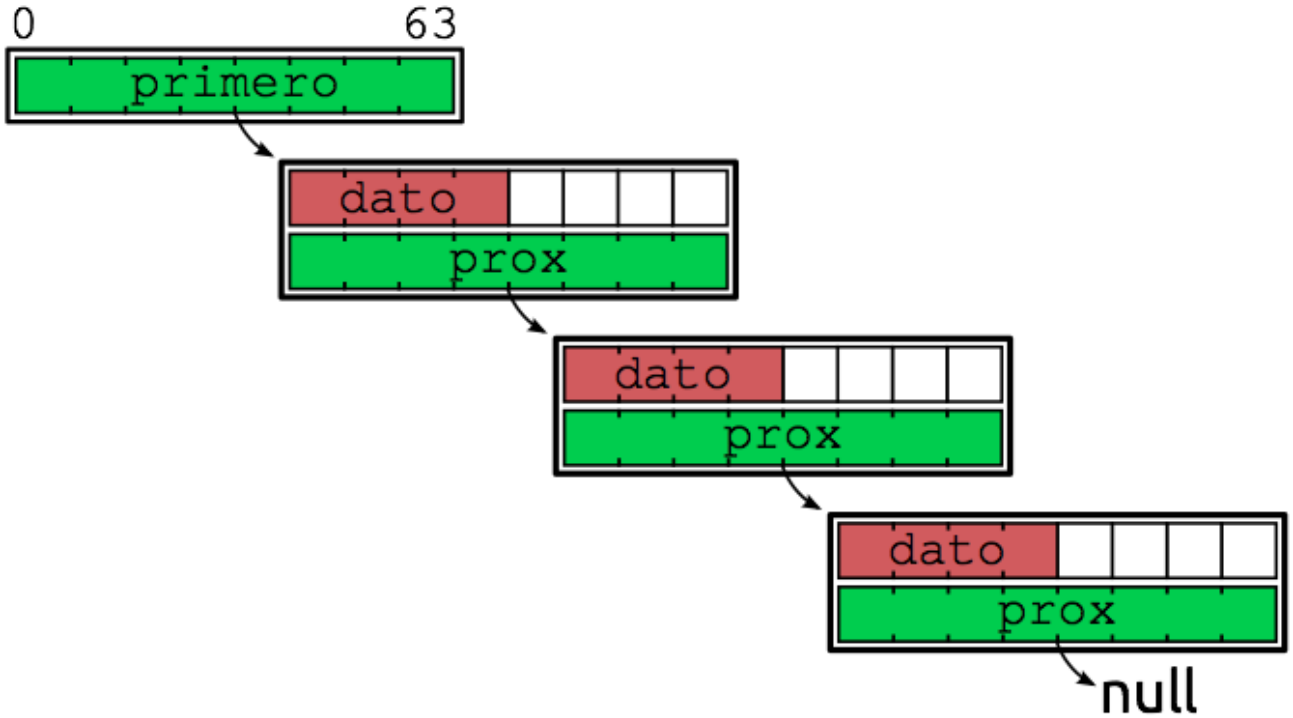
- Ubuntu/Debian: sudo apt-get install valgrind
- Otros Linux/Mac OS: http://valgrind.org/downloads/current.html
- Windows: usen Linux

Volviendo a las Listas

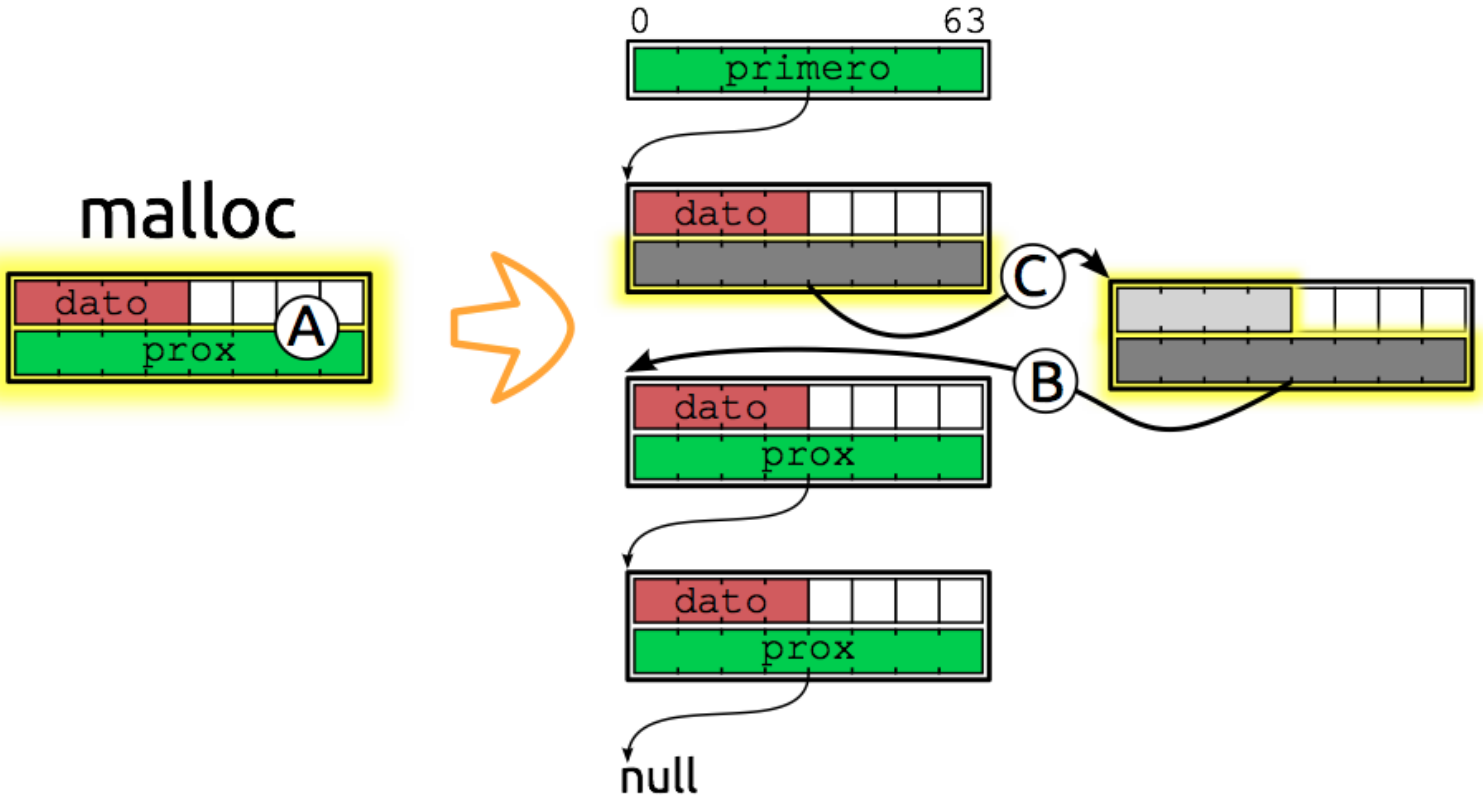
Estructuras: $\rightarrow$ SIZE $\Rightarrow$ OFFSET

```
struct lista {                    → 8   ⇒ 0
 nodo *primero;                         ⇒ 8
};
```

```
struct nodo {
 int dato;          → 4   ⇒ 0
 nodo *prox;        → 8   ⇒ 8
};                         ⇒ 16
```



Listas - Agregar

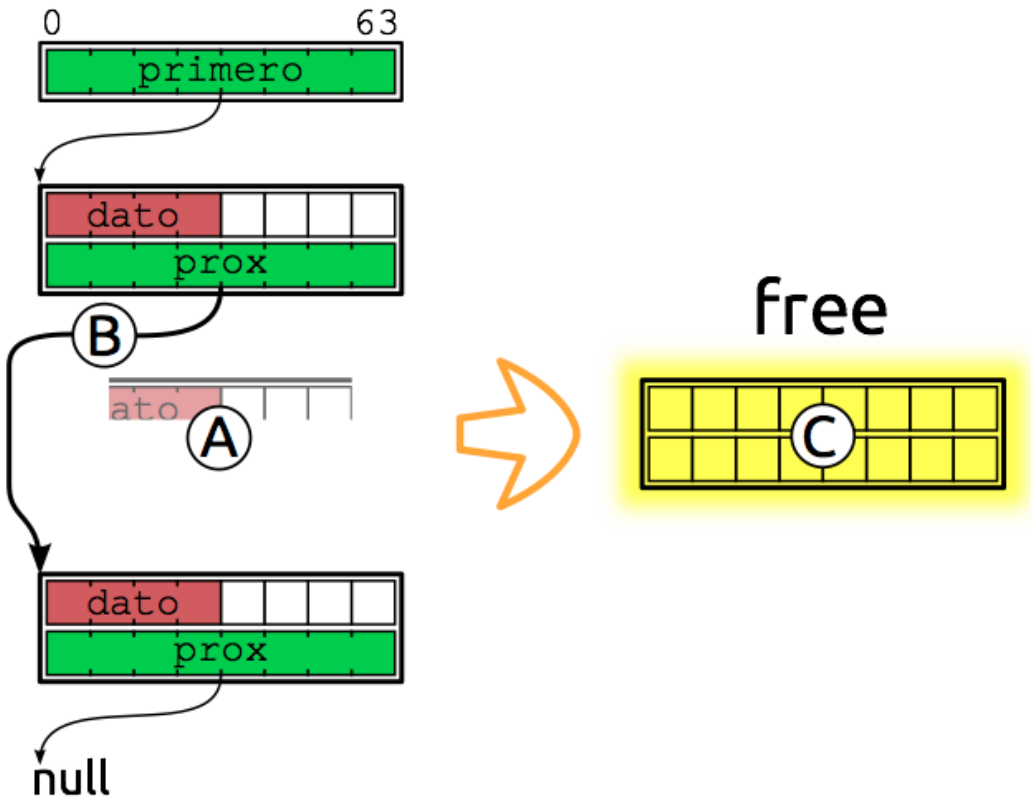


A Crear el nuevo nodo usando `malloc` y asignar su contenido

B Conectar el nuevo nodo a su siguiente en la lista

C Conectar el puntero anterior en la lista al nuevo nodo

Listas - Borrar



A Leer el valor del puntero al siguiente nodo

B Conectar el nodo anterior al siguiente del nodo a borrar

C Borrar el nodo usando `free`

Ejercicios

Estructuras: → SIZE ⇒ OFFSET

```
struct lista {                    → 8   ⇒ 0
 nodo *primero;                         ⇒ 8
};
```

```
struct nodo {
 int dato;          → 4   ⇒ 0
 nodo *prox;        → 8   ⇒ 8
};                         ⇒ 16
```

1. Escribir en ASM las siguientes funciones:

   - `void agregarPrimero(lista* unaLista, int unInt);`
     Toma una lista y agrega un nuevo nodo en la primera posición. Su dato debe ser el valor de `unInt` pasado por parámetro.
   - `void borrarUltimo(lista *unaLista);`
     Toma una lista cualquiera y de existir, borra el ultimo nodo de la lista.

Para tener en cuenta

¿En qué me voy a equivocar?

- Hacer free de la estructura recursiva pero no de los componentes.
- Si se representa la lista sólo como un puntero al primer nodo y se pide borrar el primer elemento ¿Cómo hacemos? Dos opciones:
  - Teniendo una estructura que contenga el comienzo de la lista (como vimos hoy).

```
struct lista {
 nodo *primero;
};
```

  - Con doble puntero (nodo** lista).

¡GRACIAS!