

# Memoria Dinámica

## Organización del Computador II

Damián Furman  
(Diapos robadas a David González Márquez)

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

20-08-2015

- Estructuras
- Memoria Dinámica
- Listas
- Ejercicios

## 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 ;  
}
```

## 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
```

```
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;       $\rightarrow$  8    $\Rightarrow$  0  
    char comision;      $\rightarrow$  1    $\Rightarrow$  8  
    int dni;            $\rightarrow$  4    $\Rightarrow$  12  
};                      $\Rightarrow$  16
```

```
struct alumno2 {  
    char comision;      $\rightarrow$  1    $\Rightarrow$  0  
    char* nombre;      $\rightarrow$  8    $\Rightarrow$  8  
    int dni;            $\rightarrow$  4    $\Rightarrow$  16  
};                      $\Rightarrow$  24
```

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

```
struct alumno {  
    char* nombre;       $\rightarrow$  8       $\Rightarrow$  0  
    char comision;      $\rightarrow$  1       $\Rightarrow$  8  
    int dni;            $\rightarrow$  4       $\Rightarrow$  12  
};                      $\Rightarrow$  16
```

```
struct alumno2 {  
    char comision;      $\rightarrow$  1       $\Rightarrow$  0  
    char* nombre;      $\rightarrow$  8       $\Rightarrow$  8  
    int dni;            $\rightarrow$  4       $\Rightarrow$  16  
};                      $\Rightarrow$  24
```

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

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

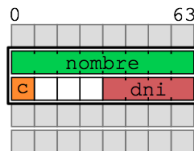
```
struct alumno {  
    char* nombre;       $\rightarrow$  8       $\Rightarrow$  0  
    char comision;      $\rightarrow$  1       $\Rightarrow$  8  
    int dni;            $\rightarrow$  4       $\Rightarrow$  12  
};                      $\Rightarrow$  16
```

```
struct alumno2 {  
    char comision;      $\rightarrow$  1       $\Rightarrow$  0  
    char* nombre;      $\rightarrow$  8       $\Rightarrow$  8  
    int dni;            $\rightarrow$  4       $\Rightarrow$  16  
};                      $\Rightarrow$  24
```

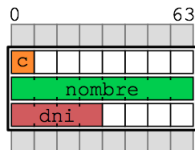
```
struct alumno3 {  
    char* nombre;       $\rightarrow$  8       $\Rightarrow$  0  
    int dni;            $\rightarrow$  4       $\Rightarrow$  8  
    char comision;      $\rightarrow$  1       $\Rightarrow$  12  
} __attribute__((packed));  $\Rightarrow$  ???
```

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

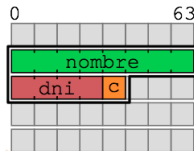
```
struct alumno {  
    char* nombre;     $\rightarrow$  8     $\Rightarrow$  0  
    char comision;    $\rightarrow$  1     $\Rightarrow$  8  
    int dni;          $\rightarrow$  4     $\Rightarrow$  12  
};                   $\Rightarrow$  16
```



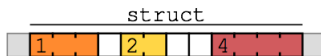
```
struct alumno2 {  
    char comision;     $\rightarrow$  1     $\Rightarrow$  0  
    char* nombre;      $\rightarrow$  8     $\Rightarrow$  8  
    int dni;           $\rightarrow$  4     $\Rightarrow$  16  
};                   $\Rightarrow$  24
```



```
struct alumno3 {  
    char* nombre;     $\rightarrow$  8     $\Rightarrow$  0  
    int dni;          $\rightarrow$  4     $\Rightarrow$  8  
    char comision;    $\rightarrow$  1     $\Rightarrow$  12  
} __attribute__((packed));  $\Rightarrow$  13
```



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

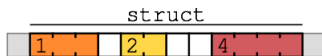




# Alineación

- Alineación en los campos del struct:

Cada campo esta alineado a su tamaño dentro del struct



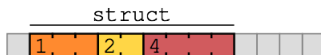
- Alineación del struct:

Se alinea al tamaño del campo mas grande del struct



- `__attribute__((packed))`:

Indica que el struct no va a ser alinenoado



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



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 off_set_nombre 0  
%define off_set_comision 8  
%define off_set_dni 12  
mov rsi, ptr_struct  
mov rbx, [rsi+off_set_nombre]  
mov al, [rsi+off_set_comision]  
mov edx, [rsi+off_set_dni]
```

Definan en un archivo en C un struct de la siguiente manera:

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

Y llamen desde la función de C a una función de Assembler que les imprima el nombre del alumno, la comisión y la edad

Ahora cambien el struct anterior por:

```
struct alumno {  
    char* nombre;  
    char comision;  
    int edad;  
}__attribute__((packed));
```

## Variable estática

Esta asignada en un espacio de memoria reservado que solo será utilizado para almacenar la variable en cuestión.

## Variable estática

Esta asignada en un espacio de memoria reservado que solo será utilizado para almacenar la variable en cuestión.

ej. ASM:    numero: dd 10  
             mensaje: db ‘‘hola pepe’’

ej. C:        const int numero = 10;  
              const char\* mensaje = ‘‘hola pepe’’;

## Variable estática

Esta asignada en un espacio de memoria reservado que solo será utilizado para almacenar la variable en cuestión.

## Variable en la pila

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

## Variable estática

Esta asignada en un espacio de memoria reservado que solo será utilizado para almacenar la variable en cuestión.

## 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;`

## Variable estática

Esta asignada en un espacio de memoria reservado que solo será utilizado para almacenar la variable en cuestión.

## Variable en la pila

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

## 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.

## 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”*

## 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
```

## 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

## Estructuras:

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

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

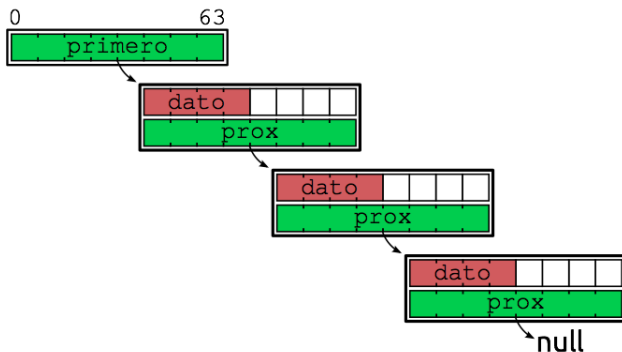


Estructuras:  $\rightarrow$  SIZE  $\Rightarrow$  OFFSET

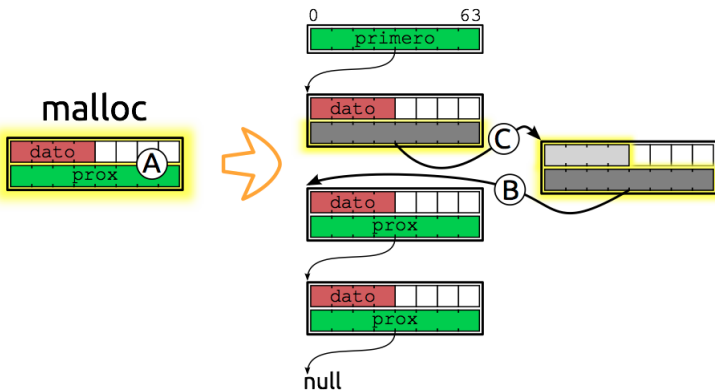
struct lista {			struct nodo {		
nodo *primero;	$\rightarrow 8$	$\Rightarrow 0$	int dato;	$\rightarrow 4$	$\Rightarrow 0$
};		$\Rightarrow 8$	nodo *prox;	$\rightarrow 8$	$\Rightarrow 8$
			};		$\Rightarrow 16$

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

<pre>struct lista {</pre>		<pre>struct nodo {</pre>	
nodo *primero;	$\rightarrow 8 \Rightarrow 0$	int dato;	$\rightarrow 4 \Rightarrow 0$
};	$\Rightarrow 8$	nodo *prox;	$\rightarrow 8 \Rightarrow 8$
		};	$\Rightarrow 16$

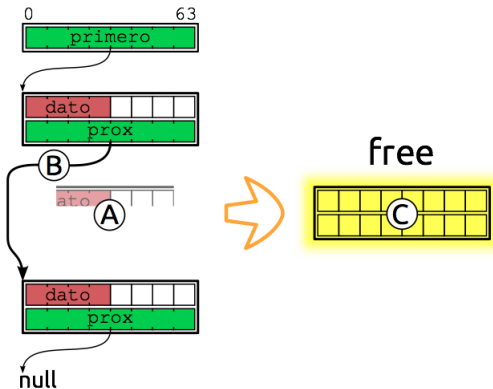


# Listas - Agregar



- 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

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

struct lista {			struct nodo {		
nodo *primero;	$\rightarrow 8$	$\Rightarrow 0$	int dato;	$\rightarrow 4$	$\Rightarrow 0$
};		$\Rightarrow 8$	nodo *prox;	$\rightarrow 8$	$\Rightarrow 8$
			};		$\Rightarrow 16$

❶ 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.

# ¡GRACIAS!

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

<pre>struct lista {</pre>		<pre>struct nodo {</pre>	
nodo *primero;	$\rightarrow 8 \Rightarrow 0$	int dato;	$\rightarrow 4 \Rightarrow 0$
};	$\Rightarrow 8$	nodo *prox;	$\rightarrow 8 \Rightarrow 8$
		};	$\Rightarrow 16$

❶ Escribir en ASM las siguientes funciones:

- void borrarPrimero(lista \*unaLista);  
Toma una lista cualquiera y de existir, borra el primer nodo de la lista.
- void agregarUltimo(lista\* unaLista, int unInt);  
Toma una lista y agrega un nuevo nodo en la ultima posición.  
Su dato debe ser el valor de unInt pasado por parámetro.