Memoria Dinámica

Estructuras, Malloc/Free y Listas

David Alejandro González Márquez

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

Hoy

- Estructuras
- Memoria Dinámica
- Listas
- Ejercicios

Estructuras

- Definen un **patrón de acceso** a memoria.
  - Equivalente a un esténcil para nombrar a un conjunto de bytes.

Estructuras

- Definen un **patrón de acceso** a memoria.
  - Equivalente a un esténcil para nombrar a un conjunto de bytes.
- Se declaran como una **lista de campos** con su nombre y tipo.
  - Desde ASM debemos conocer los tamaños de cada uno,
  - y calcular el `offset` en bytes a cada campo.

Estructuras

- Definen un **patrón de acceso** a memoria.
  - Equivalente a un esténcil para nombrar a un conjunto de bytes.
- Se declaran como una **lista de campos** con su nombre y tipo.
  - Desde ASM debemos conocer los tamaños de cada uno,
  - y calcular el `offset` en bytes a cada campo.
- Los structs pueden ser:
  - packed: No respeta reglas de alineación.
  - unpacked: Respeta reglas de alineación.

Estructuras

`struct`

Definen un patrón de acceso a un área 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 área 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;
};
```

Estructuras

`struct`

Definen un patrón de acceso a un área 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
};
```

Estructuras

`struct`

Definen un patrón de acceso a un área 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
```

Estructuras

struct

Definen un patrón de acceso a un área 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;
 char comision;
 int dni;
};
```

Estructuras

struct

Definen un patrón de acceso a un área 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
 char comision;   → 1
 int dni;         → 4
};
```

Estructuras

struct

Definen un patrón de acceso a un área 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
 int dni;          → 4
};
```

Estructuras

`struct`

Definen un patrón de acceso a un área 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
};
```

Estructuras

struct

Definen un patrón de acceso a un área 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
};
```

Estructuras

`struct`

Definen un patrón de acceso a un área determinada de memoria

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

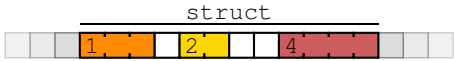
Ejemplos: → `SIZE` ⇒ `OFFSET`

```
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

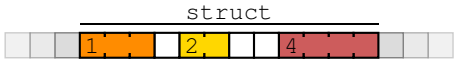
- Alineación en los campos del struct:
  Cada campo esta alineado a su tamaño dentro del struct



Alineación

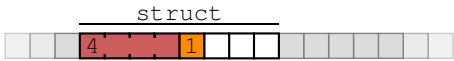
- Alineación en los campos del struct:

  Cada campo esta alineado a su tamaño dentro del struct



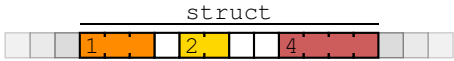
- Tamaño del struct:

  Debe ser múltiplo del campo más grande del struct

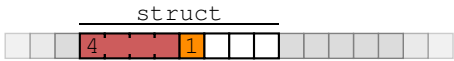


Alineación

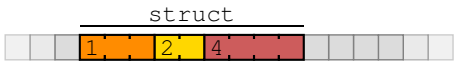
- Alineación en los campos del struct:
  Cada campo esta alineado a su tamaño dentro del struct



- Tamaño del struct:
  Debe ser múltiplo del campo más grande del struct



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



Ejemplos:

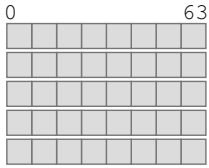
```
struct alumno {
 char* nombre;
 char comision;
 int dni;
};

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

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

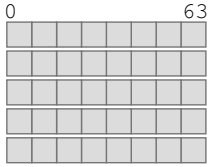
Ejemplos: → SIZE

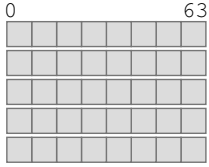
```
struct alumno {
 char* nombre;      → 8
 char comision;     → 1
 int dni;           → 4
};

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

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

Ejemplos: → SIZE

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

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

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

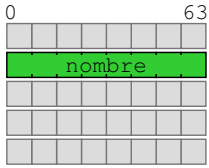




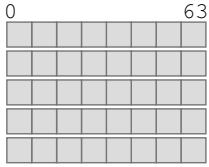


Ejemplos: → SIZE ⇒ OFFSET

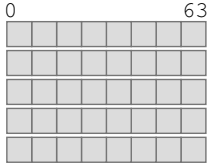
```
struct alumno {
 char* nombre;      → 8      ⇒ 0
 char comision;     → 1
 int dni;           → 4
};
```




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

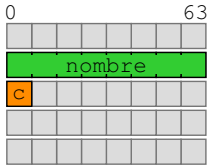



```
struct alumno3 {
 char* nombre;      → 8
 int dni;           → 4
 char comision;     → 1
} __attribute__((packed));
```

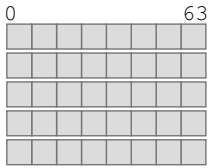


Ejemplos: → SIZE ⇒ OFFSET

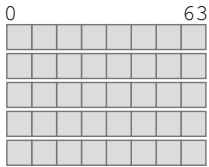
```
struct alumno {
 char* nombre;      → 8      ⇒ 0
 char comision;     → 1      ⇒ 8
 int dni;           → 4
};
```




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

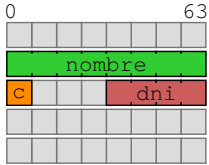



```
struct alumno3 {
 char* nombre;      → 8
 int dni;           → 4
 char comision;     → 1
} __attribute__((packed));
```

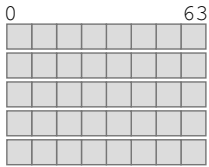


Ejemplos: → SIZE ⇒ OFFSET

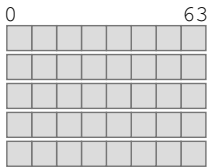
```
struct alumno {
 char* nombre;      → 8     ⇒ 0
 char comision;     → 1     ⇒ 8
 int dni;           → 4     ⇒ 12
};
```




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

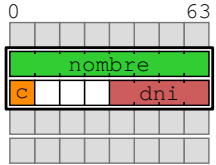



```
struct alumno3 {
 char* nombre;      → 8
 int dni;           → 4
 char comision;     → 1
} __attribute__((packed));
```

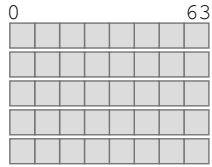


Ejemplos: → SIZE ⇒ OFFSET

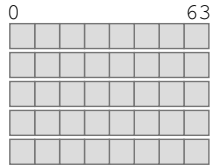
```
struct alumno {
 char* nombre;      → 8     ⇒ 0
 char comision;     → 1     ⇒ 8
 int dni;           → 4     ⇒ 12
};                          ⇒ 16
```




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

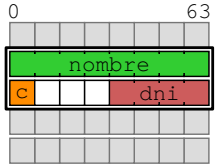



```
struct alumno3 {
 char* nombre;      → 8
 int dni;           → 4
 char comision;     → 1
} __attribute__((packed));
```

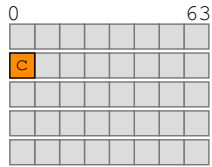


Ejemplos: → SIZE ⇒ OFFSET

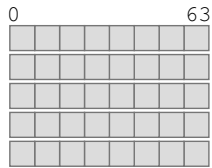
```
struct alumno {
 char* nombre;      → 8     ⇒ 0
 char comision;     → 1     ⇒ 8
 int dni;           → 4     ⇒ 12
};                          ⇒ 16
```




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

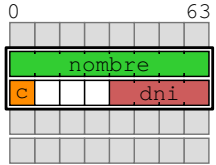



```
struct alumno3 {
 char* nombre;      → 8
 int dni;           → 4
 char comision;     → 1
} __attribute__((packed));
```

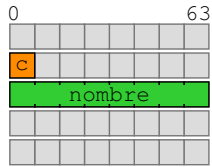


Ejemplos: → SIZE ⇒ OFFSET

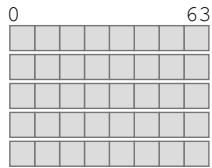
```
struct alumno {
 char* nombre;      → 8     ⇒ 0
 char comision;     → 1     ⇒ 8
 int dni;           → 4     ⇒ 12
};                          ⇒ 16
```




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

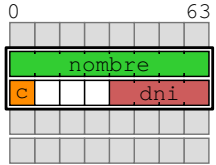



```
struct alumno3 {
 char* nombre;      → 8
 int dni;           → 4
 char comision;     → 1
} __attribute__((packed));
```

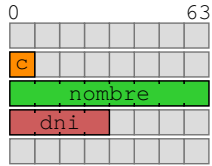


Ejemplos: → SIZE ⇒ OFFSET

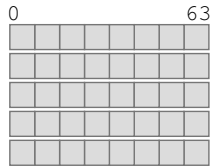
```
struct alumno {
 char* nombre;      → 8     ⇒ 0
 char comision;     → 1     ⇒ 8
 int dni;           → 4     ⇒ 12
};                          ⇒ 16
```




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

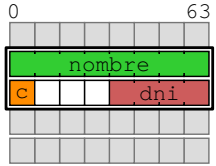



```
struct alumno3 {
 char* nombre;      → 8
 int dni;           → 4
 char comision;     → 1
} __attribute__((packed));
```

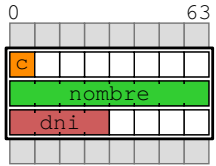


Ejemplos: → SIZE ⇒ OFFSET

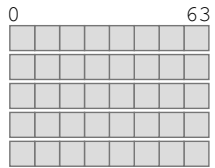
```
struct alumno {
 char* nombre;      → 8     ⇒ 0
 char comision;     → 1     ⇒ 8
 int dni;           → 4     ⇒ 12
};                          ⇒ 16
```




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

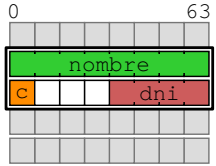



```
struct alumno3 {
 char* nombre;      → 8
 int dni;           → 4
 char comision;     → 1
} __attribute__((packed));
```

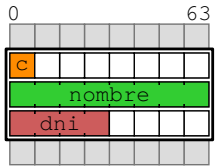


Ejemplos: → SIZE ⇒ OFFSET

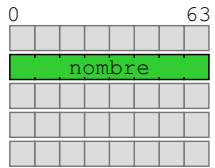
```
struct alumno {
 char* nombre;      → 8     ⇒ 0
 char comision;     → 1     ⇒ 8
 int dni;           → 4     ⇒ 12
};                          ⇒ 16
```




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

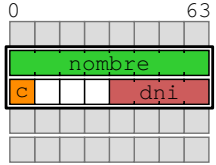



```
struct alumno3 {
 char* nombre;      → 8     ⇒ 0
 int dni;           → 4
 char comision;     → 1
} __attribute__((packed));
```

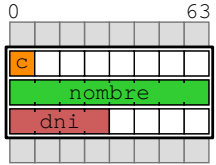


Ejemplos: → SIZE ⇒ OFFSET

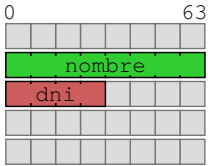
```
struct alumno {
 char* nombre;      → 8     ⇒ 0
 char comision;     → 1     ⇒ 8
 int dni;           → 4     ⇒ 12
};                          ⇒ 16
```




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

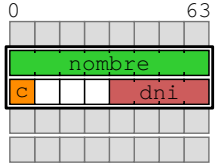



```
struct alumno3 {
 char* nombre;      → 8     ⇒ 0
 int dni;           → 4     ⇒ 8
 char comision;     → 1
} __attribute__((packed));
```

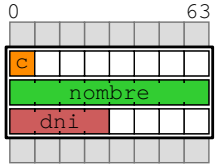


Ejemplos: → SIZE ⇒ OFFSET

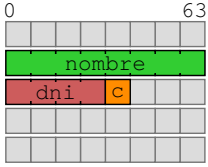
```
struct alumno {
 char* nombre;      → 8     ⇒ 0
 char comision;     → 1     ⇒ 8
 int dni;           → 4     ⇒ 12
};                          ⇒ 16
```




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

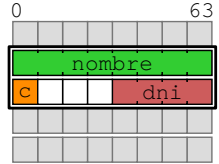



```
struct alumno3 {
 char* nombre;      → 8     ⇒ 0
 int dni;           → 4     ⇒ 8
 char comision;     → 1     ⇒ 12
} __attribute__((packed));
```

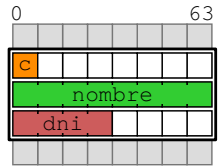


Ejemplos: → SIZE ⇒ OFFSET

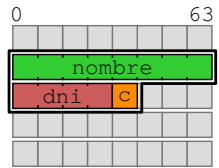
```
struct alumno {
 char* nombre;      → 8     ⇒ 0
 char comision;     → 1     ⇒ 8
 int dni;           → 4     ⇒ 12
};                          ⇒ 16
```




```
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

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 alu1;

alu1.nombre = 'carlos';
alu1.dni = alu.dni + 10;
alu1.comision = 'a';
```

Uso

Definición:

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

Uso en C:

```
struct alumno alu1;

alu1.nombre = 'carlos';
alu1.dni = alu.dni + 10;
alu1.comision = 'a';

struct alumno *alu2;

alu2->nombre = 'carlos';
alu2->dni = alu.dni + 10;
alu2->comision = 'a';
```

Uso

Definición:

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

Uso en C:

```
struct alumno alu1;

alu1.nombre = 'carlos';
alu1.dni = alu.dni + 10;
alu1.comision = 'a';

struct alumno *alu2;

alu2->nombre = 'carlos';
alu2->dni = alu.dni + 10;
alu2->comision = 'a';
```

Uso en ASM:

```
%define off_nombre 0
%define off_comision 8
%define off_dni 12
```

Uso

Definición:

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

Uso en C:

```
struct alumno alu1;

alu1.nombre = 'carlos';
alu1.dni = alu.dni + 10;
alu1.comision = 'a';

struct alumno *alu2;

alu2->nombre = 'carlos';
alu2->dni = alu.dni + 10;
alu2->comision = 'a';
```

Uso en ASM:

```
%define off_nombre 0
%define off_comision 8
%define off_dni 12

mov rsi, ptr_struct
mov rbx, [rsi+off_nombre]
mov al, [rsi+off_comision]
mov edx, [rsi+off_dni]
```

Memoria

Variable estática

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

Memoria

Variable estática

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

Ejemplo ASM:

```
section .data:
    numero: dd 10

section .rodata:
   mensaje: db 'hola pepe'

section .bss
   otro_numero: resd 1
```

Ejemplo C:

```
const int numero = 10;

const char* mensaje = 'hola pepe';

int otro_numero;
```

Memoria

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.

Memoria

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: `add rbp, 8` (Suponer `rbp` como la base del *stack frame*)

ej. C: `int* numero;`

Memoria

Variable estática

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

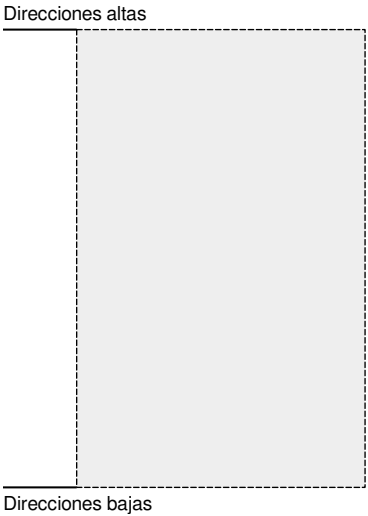
Variable en la pila

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

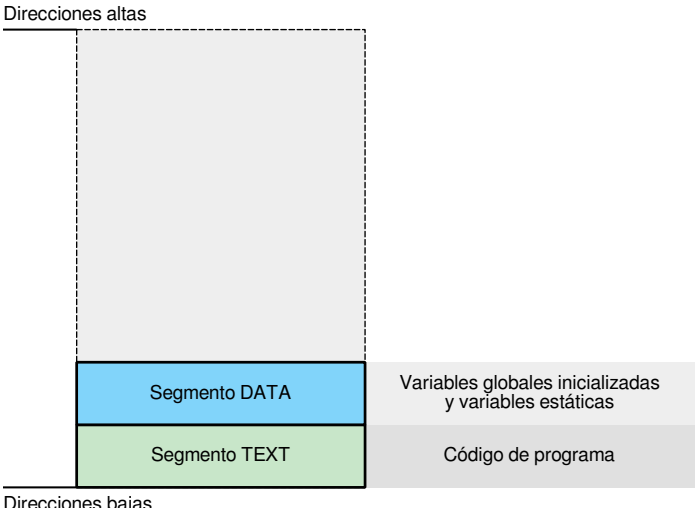
Variable dinámica

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

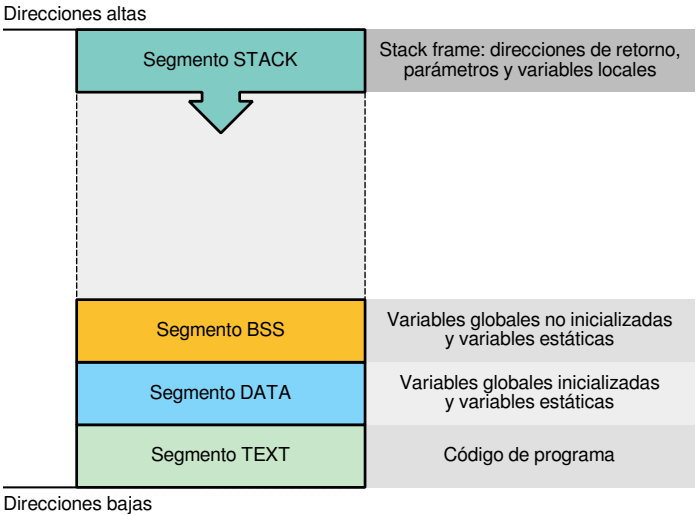
Memoria



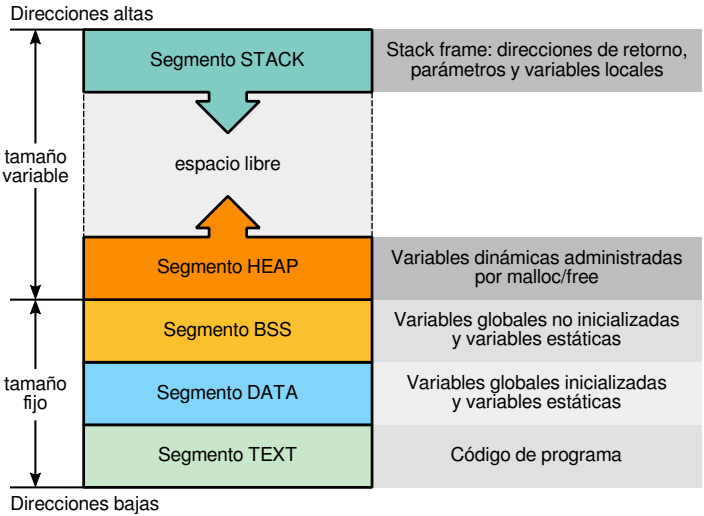
Memoria


Direcciones altas
Segmento DATA
Variables globales inicializadas y variables estáticas
Segmento TEXT
Código de programa
Direcciones bajas

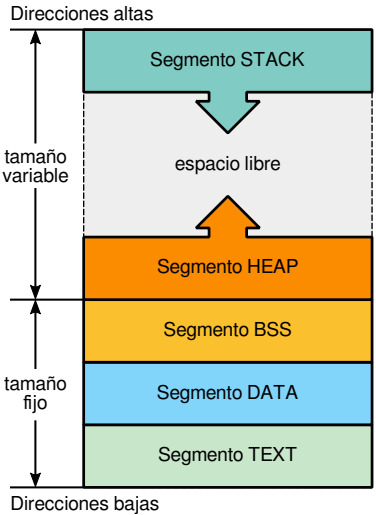
Memoria


Direcciones altas
Segmento STACK
Stack frame: direcciones de retorno, parámetros y variables locales
Segmento BSS
Variables globales no inicializadas y variables estáticas
Segmento DATA
Variables globales inicializadas y variables estáticas
Segmento TEXT
Código de programa
Direcciones bajas

Memoria

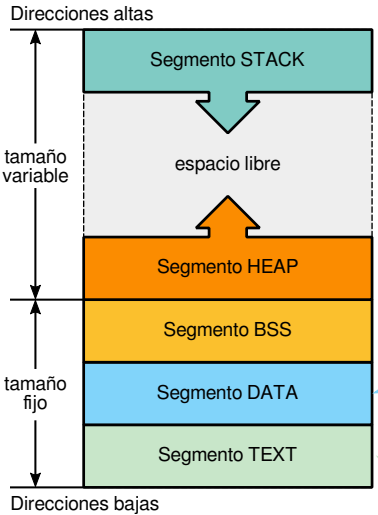

Direcciones altas
Segmento STACK
Stack frame: direcciones de retorno, parámetros y variables locales
tamaño variable
espacio libre
Segmento HEAP
Variables dinámicas administradas por malloc/free
Segmento BSS
Variables globales no inicializadas y variables estáticas
tamaño fijo
Segmento DATA
Variables globales inicializadas y variables estáticas
Segmento TEXT
Código de programa
Direcciones bajas

Memoria



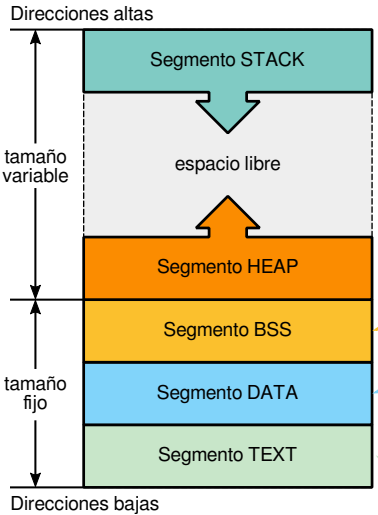

```c
#include <stdio.h>
#include <stdlib.h>

int x;
int y = 35;

int main(int argc, char *argv[]) {

    int *data;
    int i;

    data = (int*)malloc(sizeof(int)*y);

    ...

    return 0;
}
```

Memoria



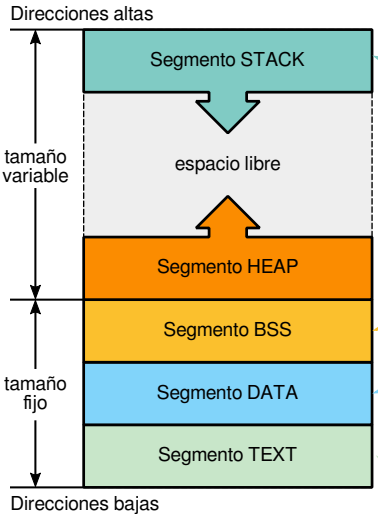

```c
#include <stdio.h>
#include <stdlib.h>

int x;
int y = 35;

int main(int argc, char *argv[]) {

    int *data;
    int i;

    data = (int*)malloc(sizeof(int)*y);

    ...

    return 0;
}
```

Memoria



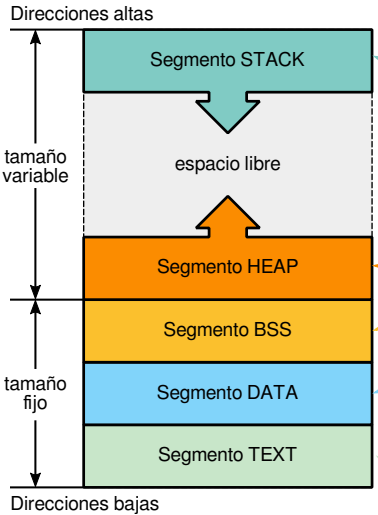

```c
#include <stdio.h>
#include <stdlib.h>

int x;
int y = 35;

int main(int argc, char *argv[]) {

    int *data;
    int i;

    data = (int*)malloc(sizeof(int)*y);

    ...

    return 0;
}
```

Memoria




```c
#include <stdio.h>
#include <stdlib.h>

int x;
int y = 35;

int main(int argc, char *argv[]) {

    int *data;
    int i;

    data = (int*)malloc(sizeof(int)*y);

    ...

    return 0;
}
```

Memoria




```c
#include <stdio.h>
#include <stdlib.h>

int x;
int y = 35;

int main(int argc, char *argv[]) {

    int *data;
    int i;

    data = (int*)malloc(sizeof(int)*y);

    ...

    return 0;
}
```

Memoria Dinámica

Solicitar memoria

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

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

Memoria Dinámica

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

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
```

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)

Memoria Dinámica - IMPORTANTE -

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

Memoria Dinámica - IMPORTANTE -

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

Caso contrario se PIERDE MEMORIA

Memoria Dinámica - IMPORTANTE -

Si se solicita memoria utilizando `malloc`, 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`

Memoria Dinámica - IMPORTANTE -

Si se solicita memoria utilizando `malloc`, 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

Uso:

```
$ valgrind --leak-check=full --show-leak-kinds=all -v ./holamundo
```

Instalación:

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

Listas

Estructuras:

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

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

Listas

Estructuras: → SIZE

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

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

Listas

Estructuras: → SIZE ⇒ OFFSET

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

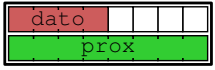
```
struct nodo {
 int dato;        → 4   ⇒ 0
 nodo *prox;      → 8   ⇒ 8
};                      ⇒ 16
```

Listas

Estructuras: → SIZE ⇒ OFFSET

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

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

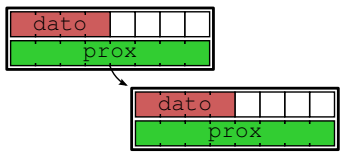


Listas

Estructuras: → SIZE ⇒ OFFSET

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

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

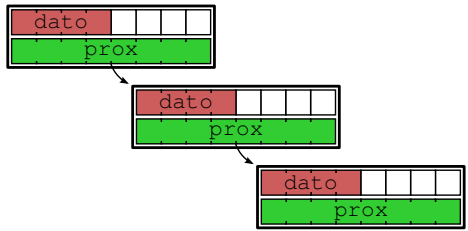


Listas

Estructuras: → SIZE ⇒ OFFSET

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

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

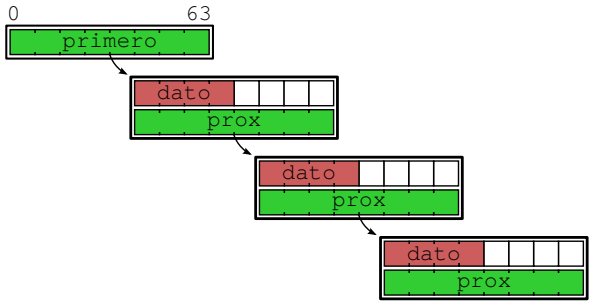


Listas

Estructuras: → SIZE ⇒ OFFSET

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

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

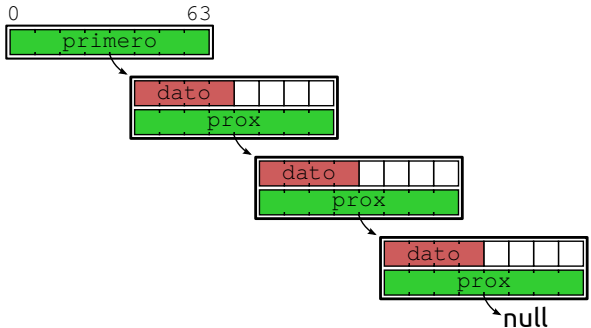


Listas

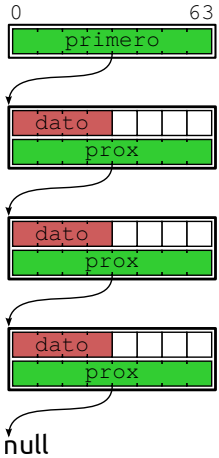
Estructuras: → SIZE ⇒ OFFSET

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

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



Listas - Agregar

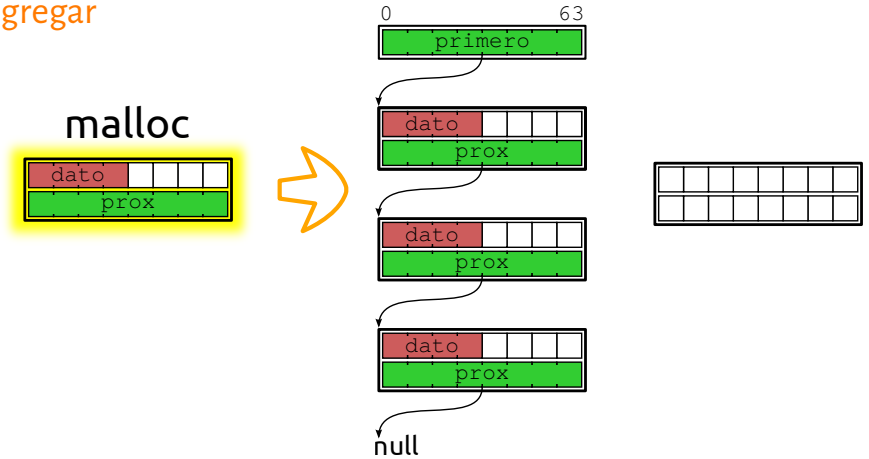


A Crear el nuevo nodo usando `malloc` y asignar su contenido

B Conectar el nuevo nodo a su siguiente en la lista

C Conectar el puntero anterior en la lista al nuevo nodo

Listas - Agregar

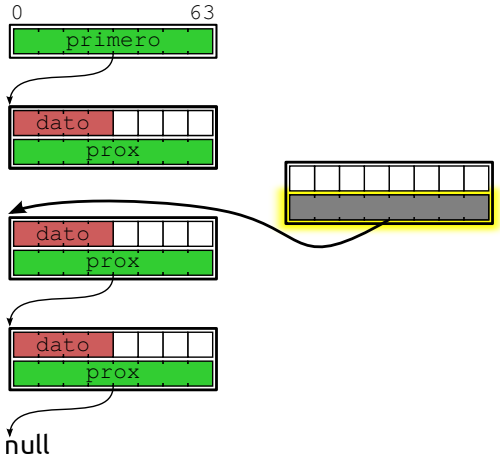


A Crear el nuevo nodo usando `malloc` y asignar su contenido

B Conectar el nuevo nodo a su siguiente en la lista

C Conectar el puntero anterior en la lista al nuevo nodo

Listas - Agregar

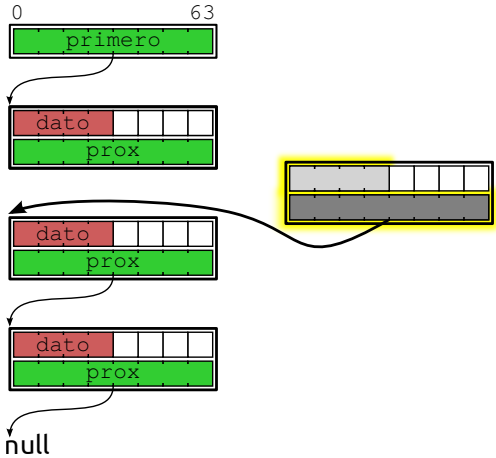


A Crear el nuevo nodo usando `malloc` y asignar su contenido

B Conectar el nuevo nodo a su siguiente en la lista

C Conectar el puntero anterior en la lista al nuevo nodo

Listas - Agregar

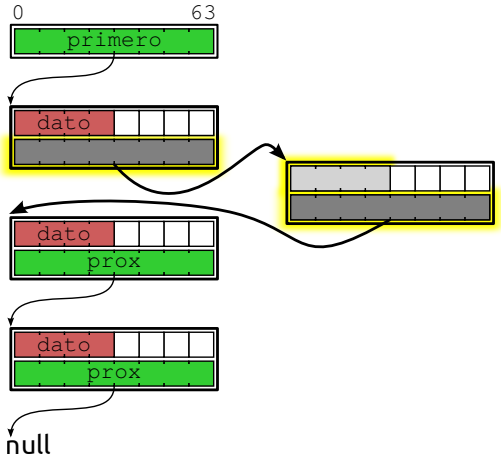


A Crear el nuevo nodo usando `malloc` y asignar su contenido

B Conectar el nuevo nodo a su siguiente en la lista

C Conectar el puntero anterior en la lista al nuevo nodo

Listas - Agregar

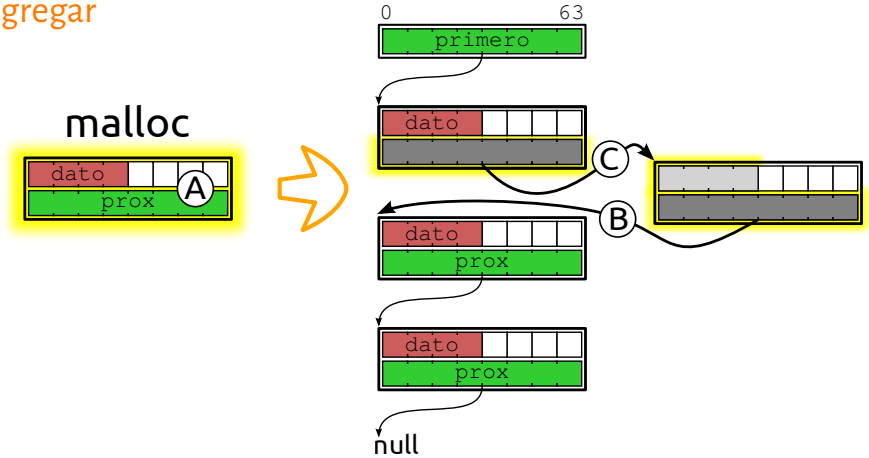


A Crear el nuevo nodo usando `malloc` y asignar su contenido

B Conectar el nuevo nodo a su siguiente en la lista

C Conectar el puntero anterior en la lista al nuevo nodo

Listas - Agregar

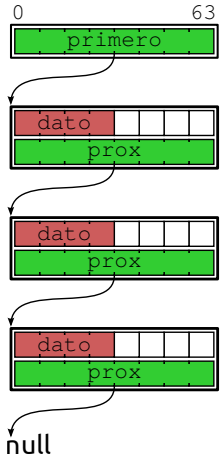


A Crear el nuevo nodo usando `malloc` y asignar su contenido

B Conectar el nuevo nodo a su siguiente en la lista

C Conectar el puntero anterior en la lista al nuevo nodo

Listas - Borrar

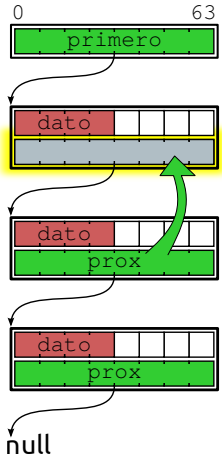


A Leer el valor del puntero al siguiente nodo

B Conectar el nodo anterior al siguiente del nodo a borrar

C Borrar el nodo usando `free`

Listas - Borrar

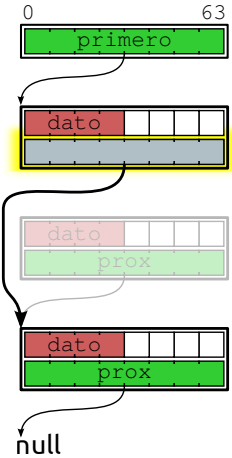


A Leer el valor del puntero al siguiente nodo

B Conectar el nodo anterior al siguiente del nodo a borrar

C Borrar el nodo usando `free`

Listas - Borrar

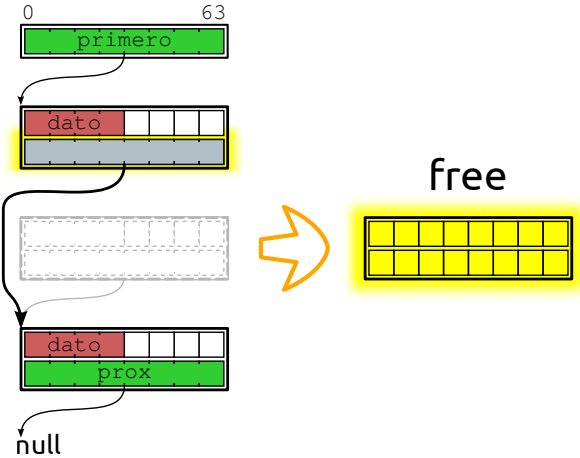


A Leer el valor del puntero al siguiente nodo

B Conectar el nodo anterior al siguiente del nodo a borrar

C Borrar el nodo usando `free`

Listas - Borrar

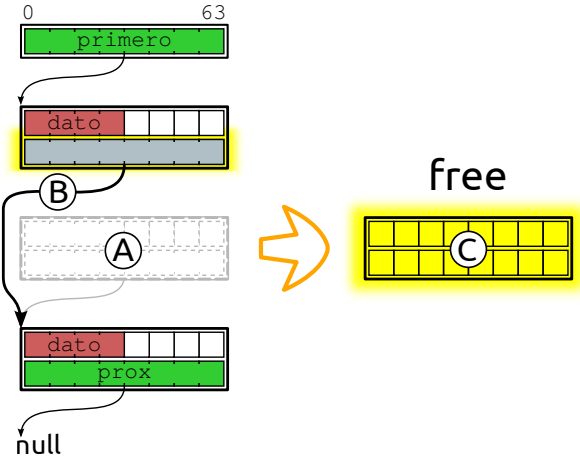


A Leer el valor del puntero al siguiente nodo

B Conectar el nodo anterior al siguiente del nodo a borrar

C Borrar el nodo usando `free`

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

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

- `void agregarPrimero(lista* unaLista, int unInt);`
  Agrega un nuevo nodo en la primera posición de la lista. Su dato será el parámetro `unInt`.
- `void agregarUltimo(lista* unaLista, int unInt);`
  Agrega un nuevo nodo en la última posición de la lista. Su dato será el parámetro `unInt`.
- `void borrarUltimo(lista *unaLista);`
  Borra el último nodo de la lista, si existe.
- `void borrarPrimero(lista *unaLista);`
  Borra el primer nodo de la lista, si existe.

Bibliografía: Fuentes y material adicional

- Convenciones de llamados a función en x86:
  `https://en.wikipedia.org/wiki/X86_calling_conventions`
- Notas sobre System V ABI:
  `https://wiki.osdev.org/System_V_ABI`
- Documentación de NASM:
  `https://nasm.us/doc/`
- Artículo sobre el flag `-pie`:
  `https://eklitzke.org/position-independent-executables`
- Documentación de System V ABI:
  `https://uclibc.org/docs/psABI-x86_64.pdf`
- Manuales de Intel:
  `https://software.intel.com/en-us/articles/intel-sdm`

¡Gracias!

Recuerden leer los comentarios al final de este video por aclaraciones o fe de erratas.