Estructuras Recursivas

Listas y Árboles

David Alejandro González Márquez

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

Nota: `typedef`

- En C es posible definir nuevos tipos de datos mediante el uso de `typedef`.

Nota: typedef

- En C es posible definir nuevos tipos de datos mediante el uso de `typedef`.

- Ejemplo:

```
typedef int numero_t
```

Nota: typedef

- En C es posible definir nuevos tipos de datos mediante el uso de `typedef`.

- Ejemplo:

```
typedef int numero_t
```

- Podría ahora declarar una función como:

```
numero_t suma(numero_t a, numero_t b)
```

Nota: typedef

- En C es posible definir nuevos tipos de datos mediante el uso de `typedef`.

- Ejemplo:

```
typedef int numero_t
```

- Podría ahora declarar una función como:

```
numero_t suma(numero_t a, numero_t b)
```

- Para el ejemplo, `numero_t` es un nuevo tipo, sinónimo de `int`.

Nota: `typedef` en `struct`'s

- Podemos usar `typedef` para renombrar `struct`'s

Nota: typedef en struct's

- Podemos usar `typedef` para renombrar `struct's`
- Ejemplo:

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

Nota: typedef en struct's

- Podemos usar `typedef` para renombrar `struct's`
- Ejemplo:

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

- Se escribiría como:

```
typedef struct {
      char* nombre;
      char comision;
      int dni;
} alumno_t;
```

Nota: typedef en struct's

- Podemos usar `typedef` para renombrar `struct's`
- Ejemplo:

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

- Se escribiría como:

```
typedef struct {
    char* nombre;
    char comision;
    int dni;
} alumno_t;
```

- `alumno_t` es un nuevo tipo y puede usarse en remplazo de `struct alumno`

Nota: typedef en tipos de funciones

- En C las funciones en sí mismas tiene un tipo de datos.

Nota: `typedef` en tipos de funciones

- En C las funciones en sí mismas tiene un tipo de datos.
- Vamos a utilizar `typedef` para nombrar al tipo de una función.

Nota: typedef en tipos de funciones

- En C las funciones en sí mismas tiene un tipo de datos.
- Vamos a utilizar `typedef` para nombrar al tipo de una función.
- Ejemplo:

```
int suma(int, int);
```

Nota: typedef en tipos de funciones

- En C las funciones en sí mismas tiene un tipo de datos.
- Vamos a utilizar `typedef` para nombrar al tipo de una función.
- Ejemplo:

```
int suma(int, int);
```

- Se escribiría como:

```
typedef int (*func_suma)(int, int);
```

Nota: typedef en tipos de funciones

- En C las funciones en sí mismas tiene un tipo de datos.
- Vamos a utilizar `typedef` para nombrar al tipo de una función.
- Ejemplo:

```
int suma(int, int);
```

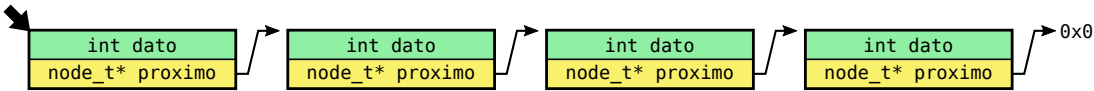
- Se escribiría como:

```
typedef int (*func_suma)(int, int);
```

- Podemos utilizar `func_suma` como el tipo de datos del puntero a una función que toma dos enteros y retorna un entero.

Listas

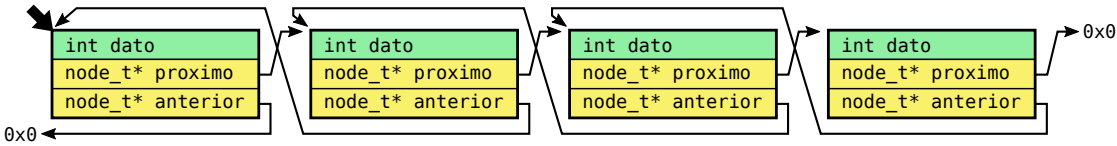
Lista simplemente enlazada:




```
typedef struct {
     int dato;
     node_t* proximo;
} node_t;
```

Listas

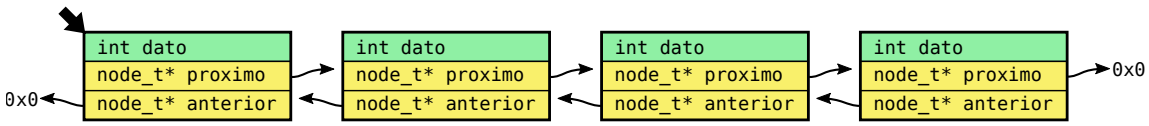
Lista doblemente enlazada:




```
typedef struct {
     int dato;
     node_t* proximo;
     node_t* anterior;
} node_t;
```

Listas

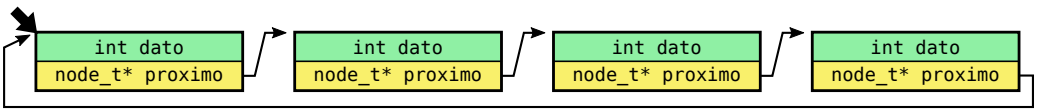
Lista doblemente enlazada (representación simplificada):




```
typedef struct {
      int dato;
      node_t* proximo;
      node_t* anterior;
} node_t;
```

Listas

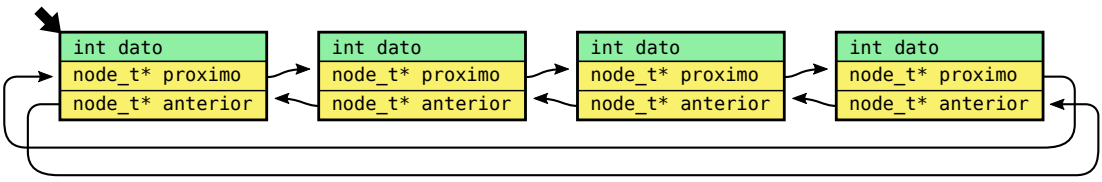
Lista circular simplemente enlazada:




```
typedef struct {
     int dato;
     node_t* proximo;
} node_t;
```

Listas

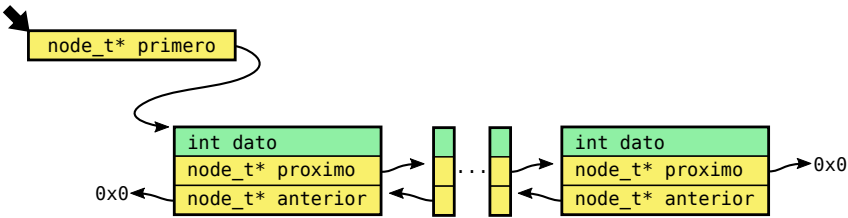
Lista circular doblemente enlazada:




```
typedef struct {
      int dato;
      node_t* proximo;
      node_t* anterior;
} node_t;
```

Listas

Lista con puntero al primer elemento:

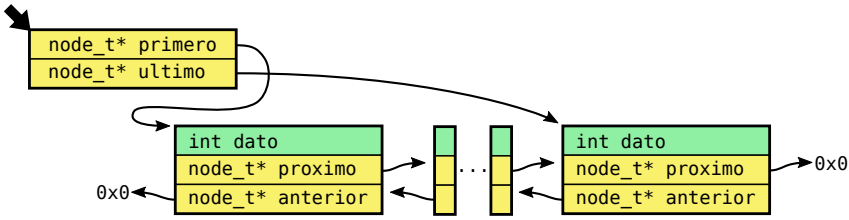



```
typedef struct {
     node_t* primero;
} list_t;
```

```
typedef struct {
     int dato;
     node_t* proximo;
     node_t* anterior;
} node_t;
```

Listas

Lista con puntero al primer y último elemento:

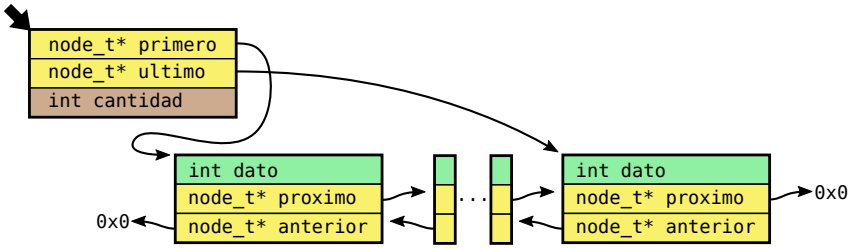



```
typedef struct {
     node_t* primero;
     node_t* ultimo;
} list_t;
```

```
typedef struct {
     int dato;
     node_t* proximo;
     node_t* anterior;
} node_t;
```

Listas

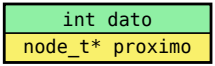
Lista con puntero al primer elemento, último elemento y tamaño:




```
typedef struct {
     node_t* primero;
     node_t* ultimo;
     int cantidad;
} list_t;
```

```
typedef struct {
     int dato;
     node_t* proximo;
     node_t* anterior;
} node_t;
```

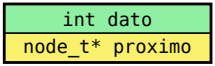
Datos



Un `int` en el nodo.

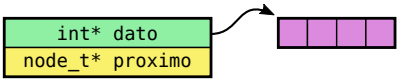
```
typedef struct {
    int dato;
    node_t* proximo;
} node_t;
```

Datos



Un `int` en el nodo.

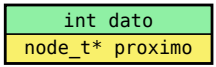
```
typedef struct {
    int dato;
    node_t* proximo;
} node_t;
```



Un puntero a un `int`.

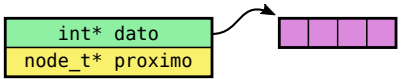
```
typedef struct {
    int* dato;
    node_t* proximo;
} node_t;
```

Datos



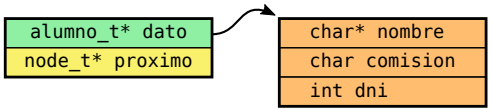
Un `int` en el nodo.

```
typedef struct {
    int dato;
    node_t* proximo;
} node_t;
```



Un puntero a un `int`.

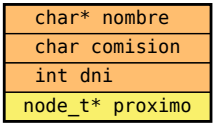
```
typedef struct {
    int* dato;
    node_t* proximo;
} node_t;
```



Un puntero a una estructura.

```
typedef struct {
    alumno_t* dato;
    node_t* proximo;
} node_t;
```

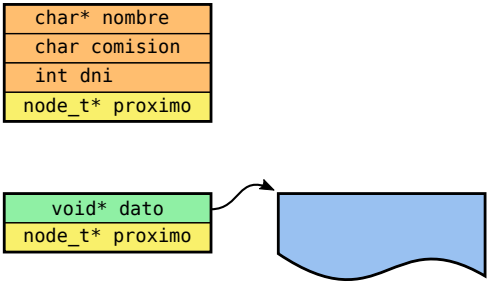
Datos



Una estructura como parte del nodo.

```
typedef struct {
        alumno_t dato;
        node_t* proximo;
} node_t;
```

Datos



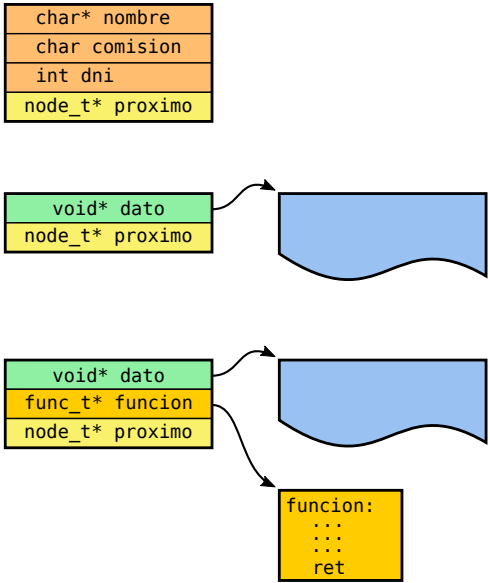
Una estructura como parte del nodo.

```
typedef struct {
        alumno_t dato;
        node_t* proximo;
} node_t;
```

Un puntero a `void` (sin tipo).

```
typedef struct {
        void* dato;
        node_t* proximo;
} node_t;
```

Datos



Una estructura como parte del nodo.

```
typedef struct {
        alumno_t dato;
        node_t* proximo;
} node_t;
```

Un puntero a `void` (sin tipo).

```
typedef struct {
        void* dato;
        node_t* proximo;
} node_t;
```

Un puntero a `void` y un puntero a una función.

```
typedef struct {
        void* dato;
        func_t* funcion;
        node_t* proximo
} node_t;
```

Árboles

Árbol binario:

```
typedef struct {
     int dato
     node_t* derecha
     node_t* izquierda
} node_t;
```

- Cada nodo define un par de punteros, uno a derecha y otro a izquierda.

Árboles

Árbol binario:

```
typedef struct {
     int dato
     node_t* derecha
     node_t* izquierda
} node_t;
```

- Cada nodo define un par de punteros, uno a derecha y otro a izquierda.
- **Árbol binario de busqueda**: Todos los datos a derecha son más grandes que el dato en la raíz y todos los datos a izquierda son menores o iguales al de la raíz.

Árboles

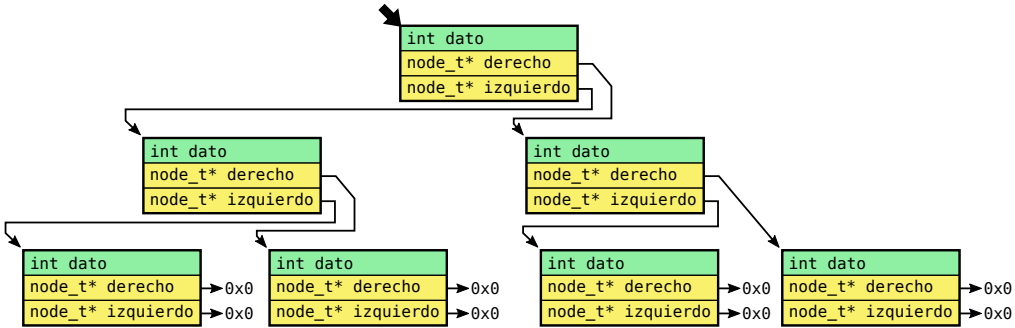
Árbol binario:

```
typedef struct {
     int dato
     node_t* derecha
     node_t* izquierda
} node_t;
```

- Cada nodo define un par de punteros, uno a derecha y otro a izquierda.
- **Árbol binario de busqueda**: Todos los datos a derecha son más grandes que el dato en la raíz y todos los datos a izquierda son menores o iguales al de la raíz.
- **Balanceado**: Para todos los nodos, la cantidad de nodos de cada lado del arbol es equivalente.

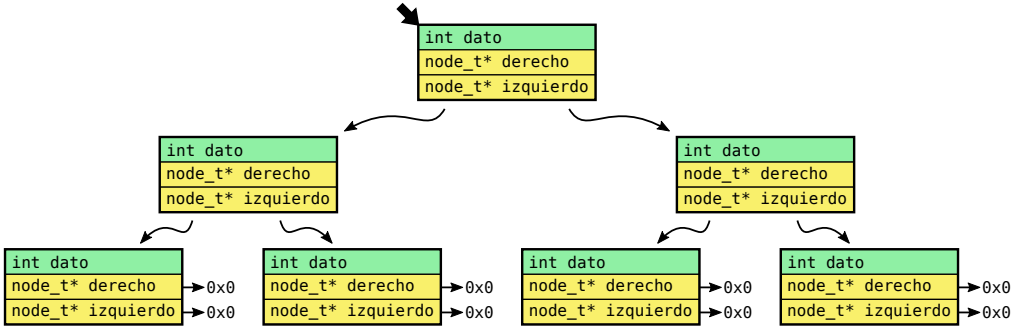
Árboles

Árbol binario:



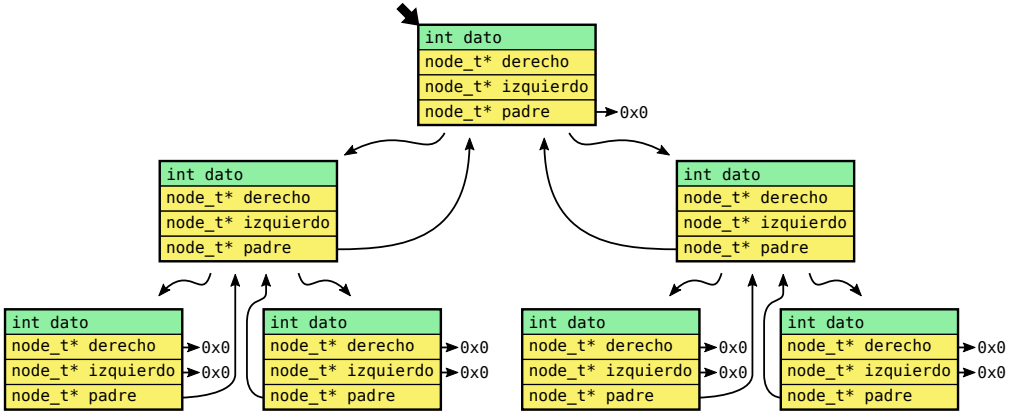
Árboles

Árbol binario (representación simplificada):



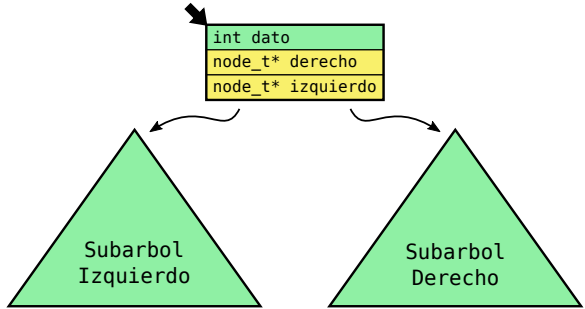
Árboles

Árbol binario con puntero al padre:



Árboles

Representación de un árbol binario:



Árboles n-arios

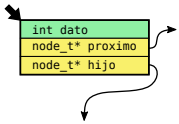
Árbol n-ario:

```
typedef struct {
    int dato
    node_t* proximo
    node_t* hijo
} node_t;
```

- Cada nivel funciona como una lista.
- El puntero hijos indica el nivel inferior.
- Nivel puede tener una cantidad arbitraria de nodos.

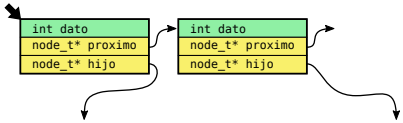
Árboles n-arios

Árbol n-ario:



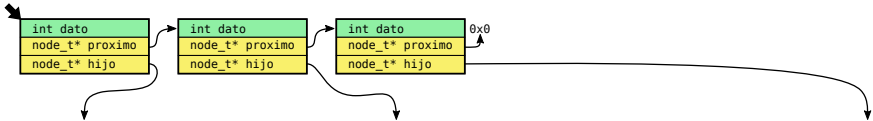
Árboles n-arios

Árbol n-ario:



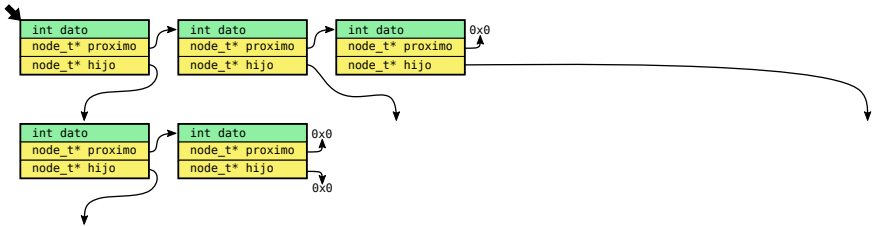
Árboles n-arios

Árbol n-ario:



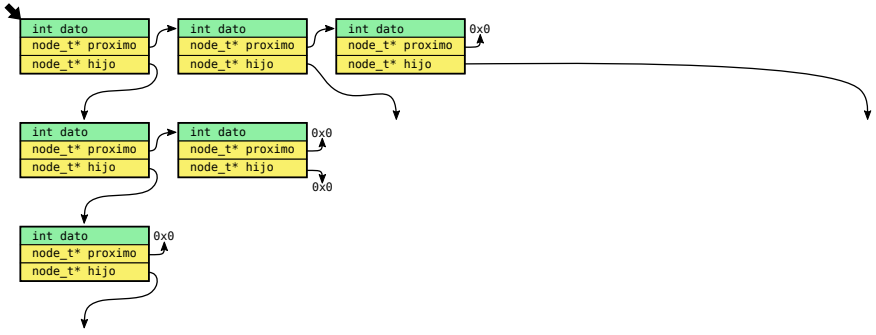
Árboles n-arios

Árbol n-ario:



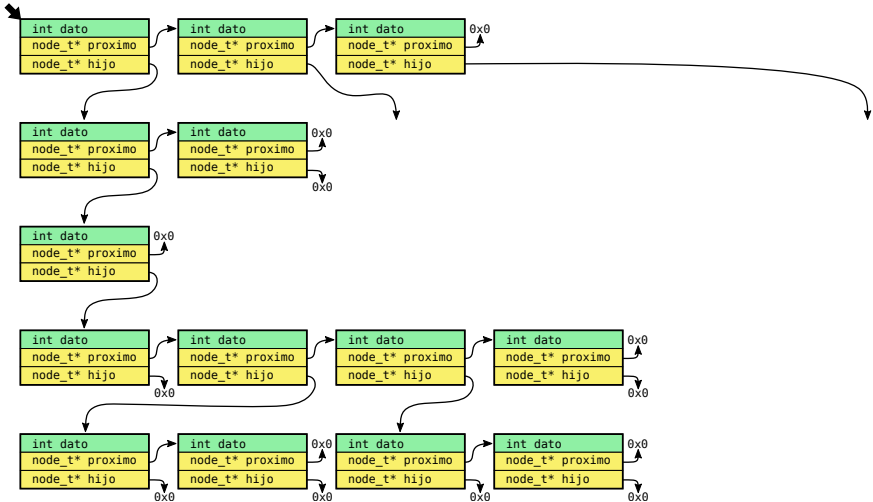
Árboles n-arios

Árbol n-ario:



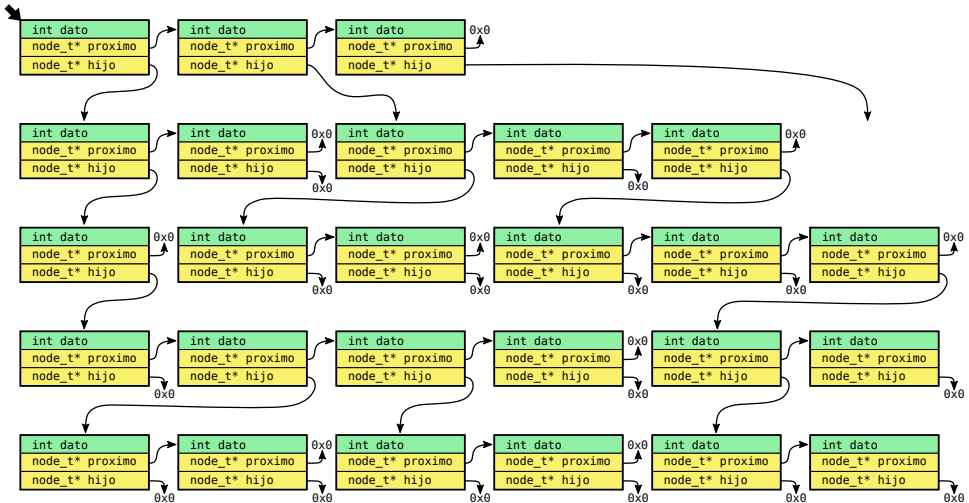
Árboles n-arios

Árbol n-ario:



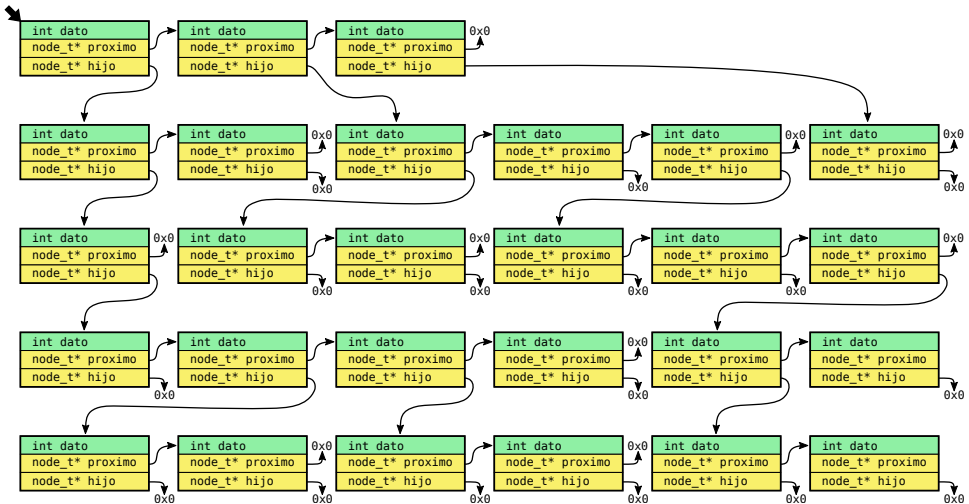
Árboles n-arios

Árbol n-ario:



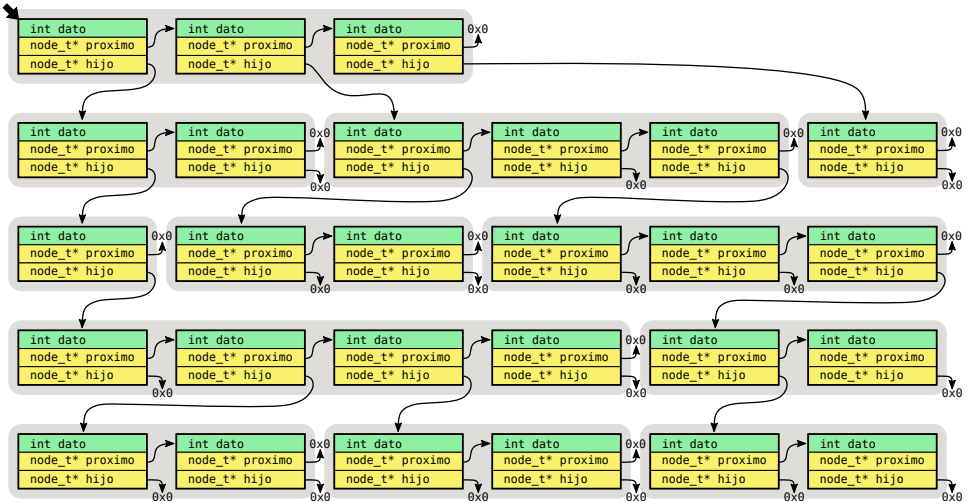
Árboles n-arios

Árbol n-ario:



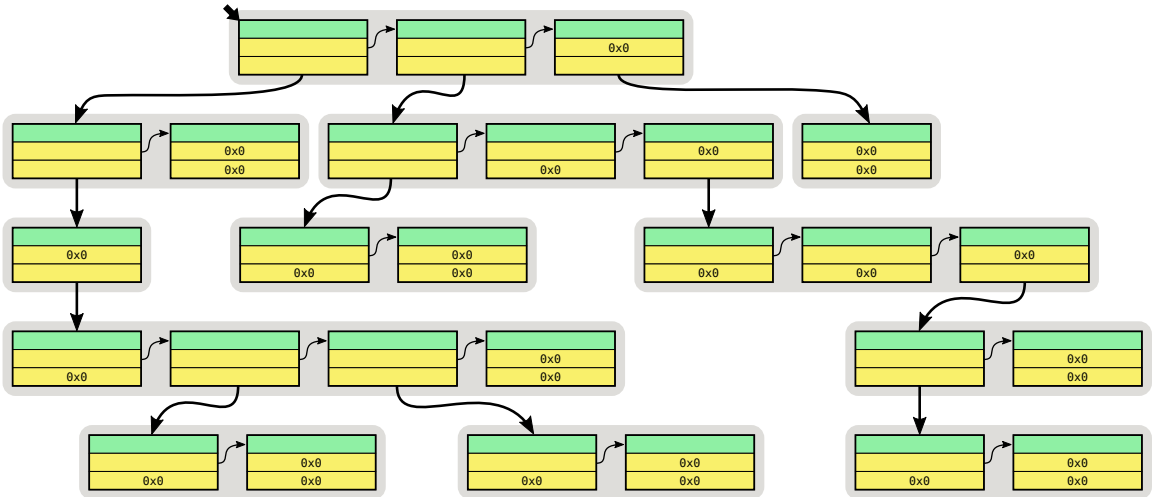
Árboles n-arios

Árbol n-ario:



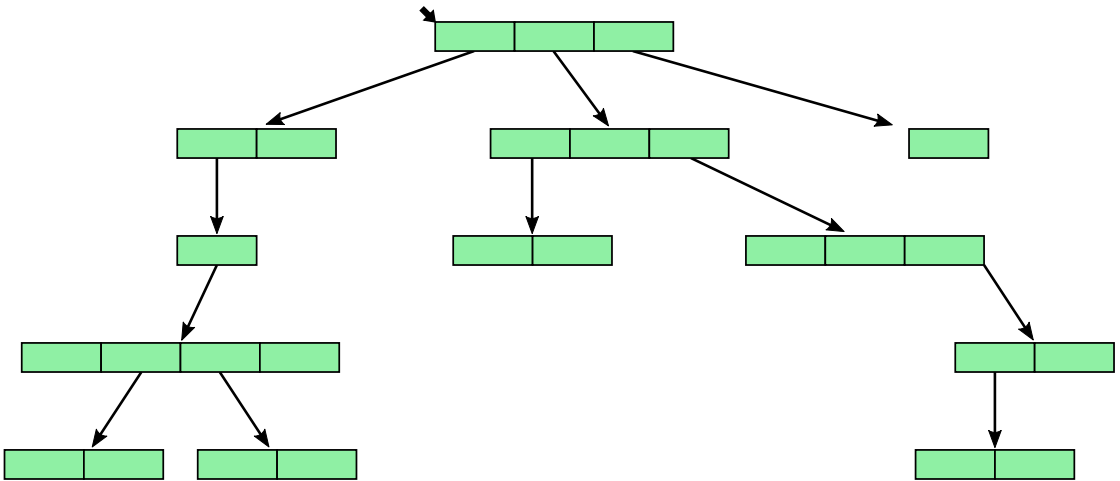
Árboles n-arios

Árbol n-ario (representación simplificada):

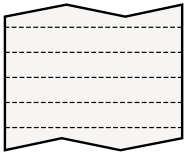


Árboles n-arios

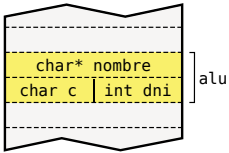
Árbol n-ario (representación simplificada):



Nota sobre doble punteros



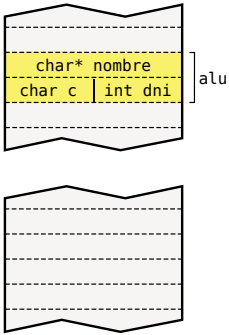
Nota sobre doble punteros



En C
```
alumno_t alu;
```

En ASM
```
push rbp
mov rbp, rsp
sub rsp, 16; variable local en la pila (alumno_t)
```

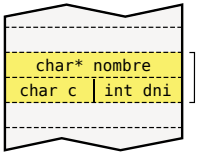
Nota sobre doble punteros



En C
```
alumno_t alu;
```

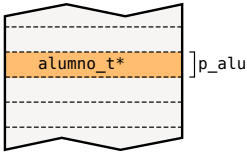
En ASM
```
push rbp
mov rbp, rsp
sub rsp, 16; variable local en la pila (alumno_t)
```

Nota sobre doble punteros



En C `alumno_t alu;`

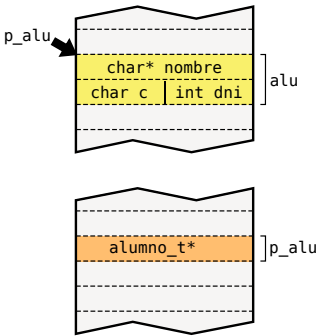
En ASM
```
push rbp
mov rbp, rsp
sub rsp, 24; variable local en la pila (alumno_t y alumno_t*)
```



En C `alumno_t *p_alu = &alu;`

En ASM
```
lea rax, [rbp-16]; obtener la dirección de alu
mov [rbp-24], rax; escribir la dirección en una variable
```

Nota sobre doble punteros



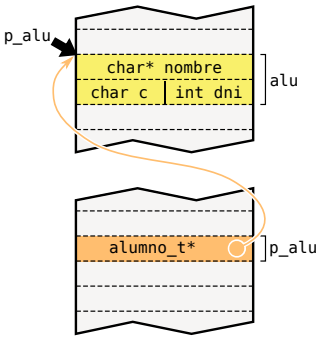
En C `alumno_t alu;`

En ASM
```
push rbp
mov rbp, rsp
sub rsp, 24; variable local en la pila (alumno_t y alumno_t*)
```

En C `alumno_t *p_alu = &alu;`

En ASM
```
lea rax, [rbp-16]; obtener la dirección de alu
mov [rbp-24], rax; escribir la dirección en una variable
```

Nota sobre doble punteros



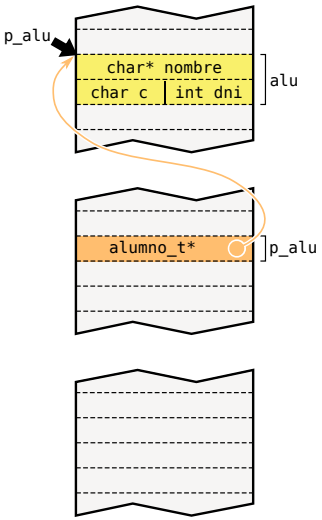
En C `alumno_t alu;`

En ASM
```
push rbp
mov rbp, rsp
sub rsp, 24; variable local en la pila (alumno_t y alumno_t*)
```

En C `alumno_t *p_alu = &alu;`

En ASM
```
lea rax, [rbp-16]; obtener la dirección de alu
mov [rbp-24], rax; escribir la dirección en una variable
```

Nota sobre doble punteros



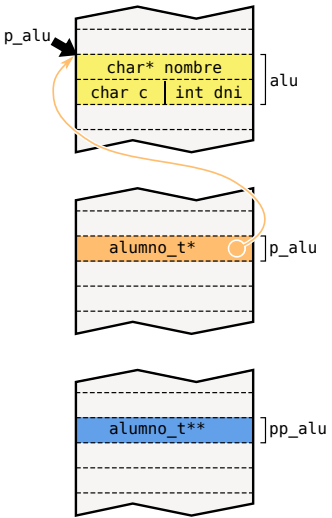
En C `alumno_t alu;`

En ASM
```
push rbp
mov rbp, rsp
sub rsp, 24; variable local en la pila (alumno_t y alumno_t*)
```

En C `alumno_t *p_alu = &alu;`

En ASM
```
lea rax, [rbp-16]; obtener la dirección de alu
mov [rbp-24], rax; escribir la dirección en una variable
```

Nota sobre doble punteros



En C `alumno_t alu;`

En ASM
```
push rbp
mov rbp, rsp
sub rsp, 24; variable local en la pila (alumno_t y alumno_t*)
```

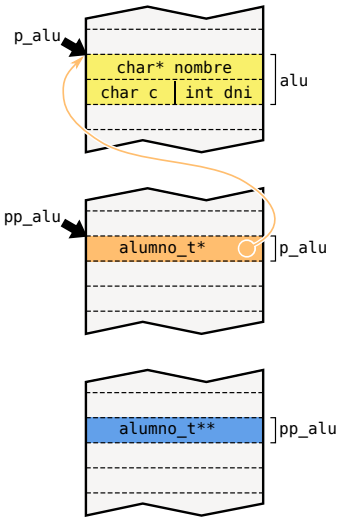
En C `alumno_t *p_alu = &alu;`

En ASM
```
lea rax, [rbp-16]; obtener la dirección de alu
mov [rbp-24], rax; escribir la dirección en una variable
```

En C `alumno_t **pp_alu = &p_alu;`

En ASM
```
lea rdi, [rbp-24]; obtener la dirección de la variable
```

Nota sobre doble punteros



En C `alumno_t alu;`

En ASM
```
push rbp
mov rbp, rsp
sub rsp, 24; variable local en la pila (alumno_t y alumno_t*)
```

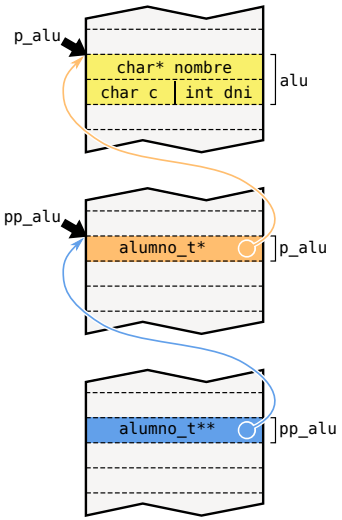
En C `alumno_t *p_alu = &alu;`

En ASM
```
lea rax, [rbp-16]; obtener la dirección de alu
mov [rbp-24], rax; escribir la dirección en una variable
```

En C `alumno_t **pp_alu = &p_alu;`

En ASM
```
lea rdi, [rbp-24]; obtener la dirección de la variable
```

Nota sobre doble punteros



En C `alumno_t alu;`

En ASM
```
push rbp
mov rbp, rsp
sub rsp, 24; variable local en la pila (alumno_t y alumno_t*)
```

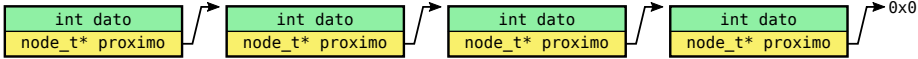
En C `alumno_t *p_alu = &alu;`

En ASM
```
lea rax, [rbp-16]; obtener la dirección de alu
mov [rbp-24], rax; escribir la dirección en una variable
```

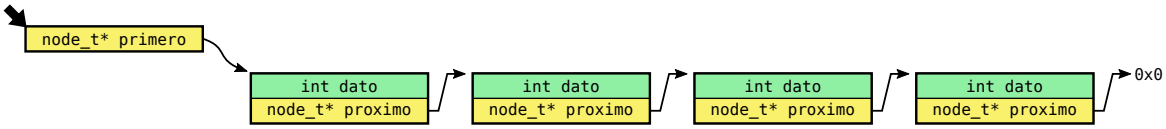
En C `alumno_t **pp_alu = &p_alu;`

En ASM
```
lea rdi, [rbp-24]; obtener la dirección de la variable
```

Nota sobre como recorrer una lista con un doble puntero

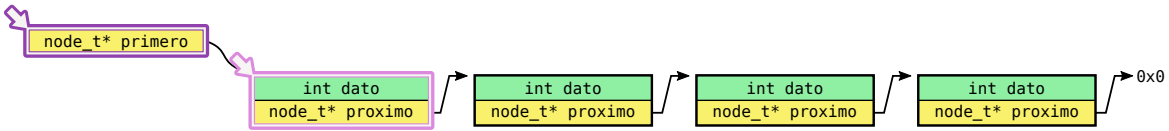


Nota sobre como recorrer una lista con un doble puntero



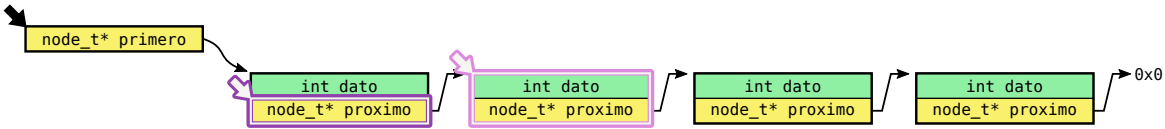
- Suponer un puntero al primer elemento de la lista.

Nota sobre como recorrer una lista con un doble puntero



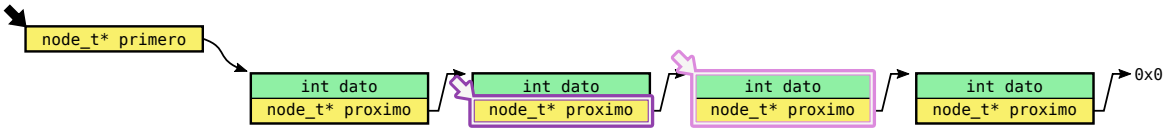
- Suponer un puntero al primer elemento de la lista.
- Para recorrer la lista, obtenemos dos punteros:
  1. Doble puntero al primer nodo
  2. Puntero al primer nodo

Nota sobre como recorrer una lista con un doble puntero



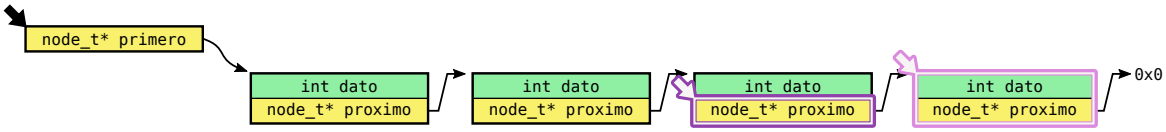
- Suponer un puntero al primer elemento de la lista.
- Para recorrer la lista, obtenemos dos punteros:
  1. Doble puntero al primer nodo
  2. Puntero al primer nodo
- Luego, iteramos moviendo ambos punteros sobre la lista.

Nota sobre como recorrer una lista con un doble puntero



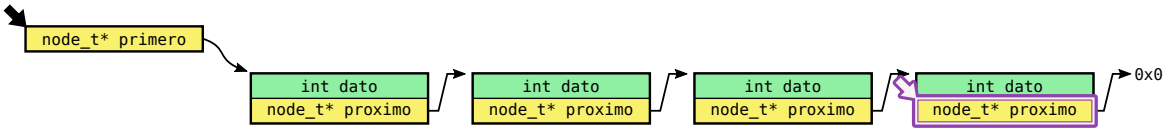
- Suponer un puntero al primer elemento de la lista.
- Para recorrer la lista, obtenemos dos punteros:
  1. Doble puntero al primer nodo
  2. Puntero al primer nodo
- Luego, iteramos moviendo ambos punteros sobre la lista.

Nota sobre como recorrer una lista con un doble puntero



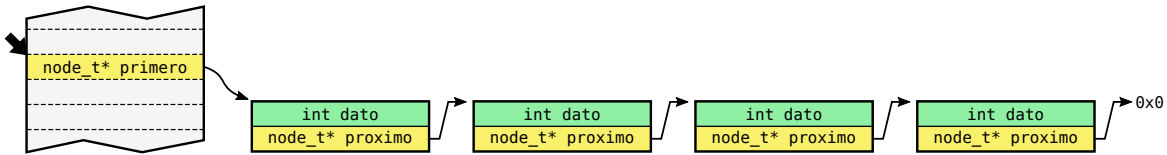
- Suponer un puntero al primer elemento de la lista.
- Para recorrer la lista, obtenemos dos punteros:
  1. Doble puntero al primer nodo
  2. Puntero al primer nodo
- Luego, iteramos moviendo ambos punteros sobre la lista.

Nota sobre como recorrer una lista con un doble puntero



- Suponer un puntero al primer elemento de la lista.
- Para recorrer la lista, obtenemos dos punteros:
  1. Doble puntero al primer nodo
  2. Puntero al primer nodo
- Luego, iteramos moviendo ambos punteros sobre la lista.
- El final de la lista será cuando el doble puntero apunte a `null`

Nota sobre como recorrer una lista con un doble puntero



- Suponer un puntero al primer elemento de la lista.
- Para recorrer la lista, obtenemos dos punteros:
  1. Doble puntero al primer nodo
  2. Puntero al primer nodo
- Luego, iteramos moviendo ambos punteros sobre la lista.
- El final de la lista será cuando el doble puntero apunte a `null`
- Considerar que el doble puntero puede ser una posición en la pila.

Nota sobre estructuras

- Las estructuras pueden ser tan complejas como queramos.

Nota sobre estructuras

- Las estructuras pueden ser tan complejas como queramos.
- Tener en cuenta:
  - ¿Qué es parte de la estructura? y ¿Qué es un puntero?

Nota sobre estructuras

- Las estructuras pueden ser tan complejas como queramos.
- Tener en cuenta:
  - ¿Qué es parte de la estructura? y ¿Qué es un puntero?
  - ¿Qué tamaño tiene y en que lugar de memoria esta?

Nota sobre estructuras

- Las estructuras pueden ser tan complejas como queramos.
- Tener en cuenta:
  - ¿Qué es parte de la estructura? y ¿Qué es un puntero?
  - ¿Qué tamaño tiene y en que lugar de memoria esta?
  - ¿Tiene punteros a funciones?, ¿Cuáles son?, y ¿Para qué sirven?, ¿Está definidas?

Nota sobre estructuras

- Las estructuras pueden ser tan complejas como queramos.
- Tener en cuenta:
  - ¿Qué es parte de la estructura? y ¿Qué es un puntero?
  - ¿Qué tamaño tiene y en que lugar de memoria esta?
  - ¿Tiene punteros a funciones?, ¿Cuáles son?, y ¿Para qué sirven?, ¿Está definidas?
  - Recordar, dibujar las estructuras y los punteros.

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.