

Mémoire dinámica

Parte 2

Estructuras de Datos

Dpto. Lenguajes y Ciencias de la
Computación.

Universidad de Málaga



Modularidad en C

- La modularidad se logra mediante la separación del código en archivos de encabezado (*.h) y archivos de implementación (*.c).
- Los archivos de encabezado contienen las declaraciones de las funciones y estructuras que se utilizarán en otros archivos
- Los archivos de implementación contienen las definiciones de esas funciones y estructuras.

Modularidad en C: ventajas

- Facilita la reutilización del código
- Mejora la legibilidad y el mantenimiento del código
- Ayuda a evitar la duplicación de código y a mantener una estructura organizada del proyecto.

Modularidad en C: librerías

```
// Incluye el archivo de encabezado de la biblioteca estándar de C
#include <stdio.h>

// Incluye el archivo de encabezado local, comillas dobles
#include "miarchivo.h"

int main(void) {
    // Código del programa
    return 0;
}
```

Ejercicio: Punto.zip

Modularidad en C: cabeceras

```
#ifndef PUNTO_H // Evita la inclusión múltiple del archivo de encabezado
#define PUNTO_H

struct Punto{
    int x;
    int y;
}; // Declaración de una estructura

struct Punto suma(struct Punto a, struct Punto b); // Declaración de una función

#endif
```

Modularidad en C: implementación

```
#include "punto.h" // Incluye el archivo de encabezado local

struct Punto suma(struct Punto a, struct Punto b) {

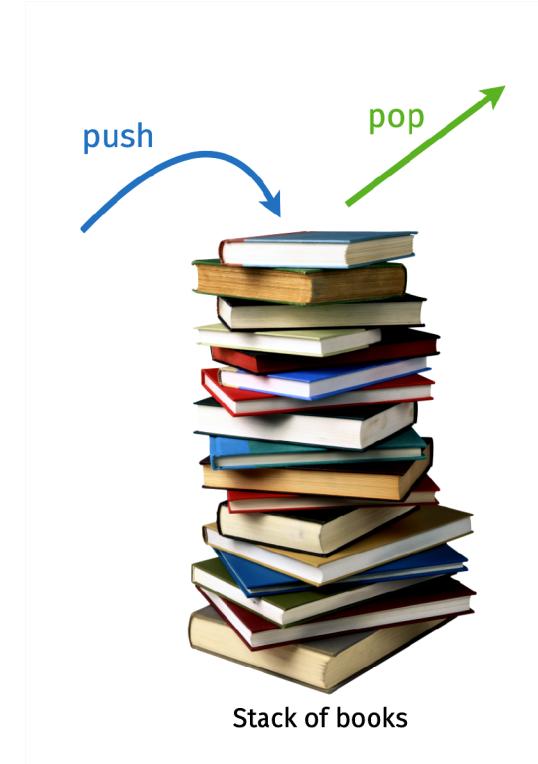
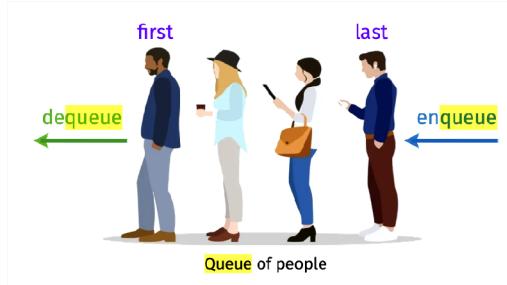
    a.x += b.x;
    a.y += b.y;
    return a;
}
```

Compilar código con librerías

```
gcc -g ./Principal.c ./punto.c -std=c17 -Wall -Wextra -Wpedantic -o a.out
```

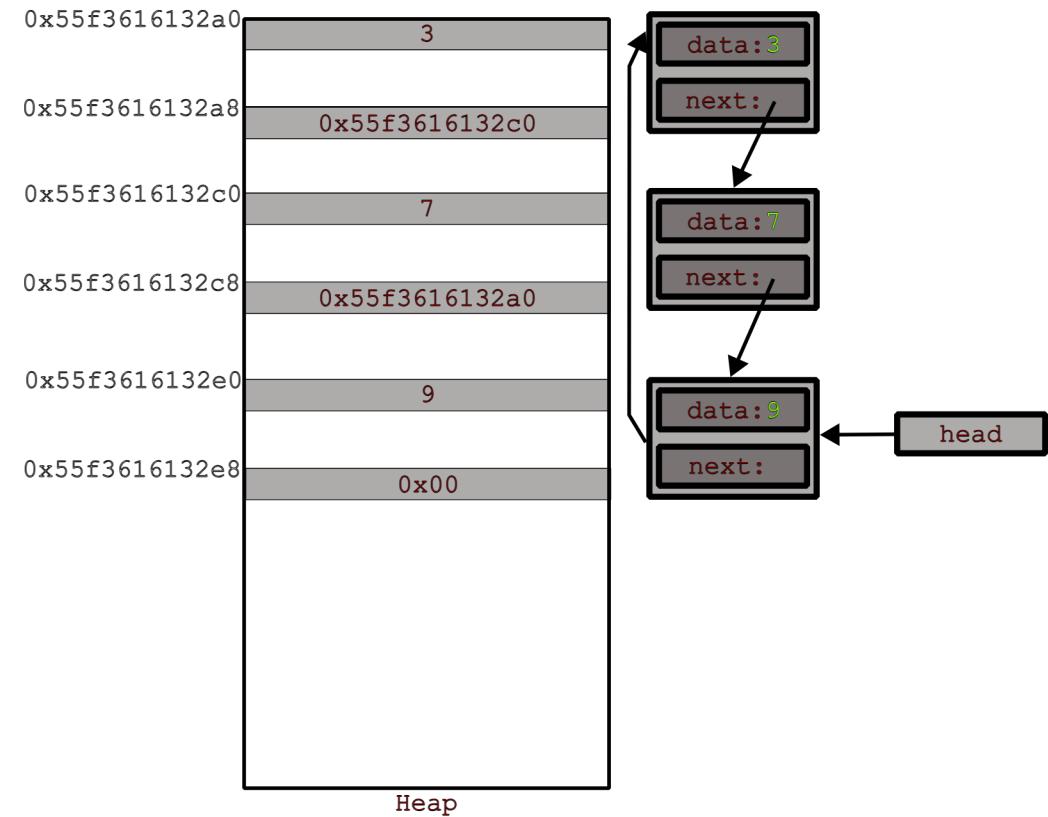
Ejercicios para practicar la modularidad y las estructuras enlazadas

- Cola (Estructura FIFO *First-In-First-Out*)
- Pila (Estructura LIFO *Last-In-First-Out*)



Ejercicio Cola

- Es una estructura *First-In-First-Out*, por lo que se insertará en la cola y se extraerá de la cabeza de la lista.
- Vamos a implementarlo con una **lista circular**.
- *head* apunta al último elemento insertado (cola de la lista).
- *head->next* es el primer elemento insertado (cabeza de la lista).

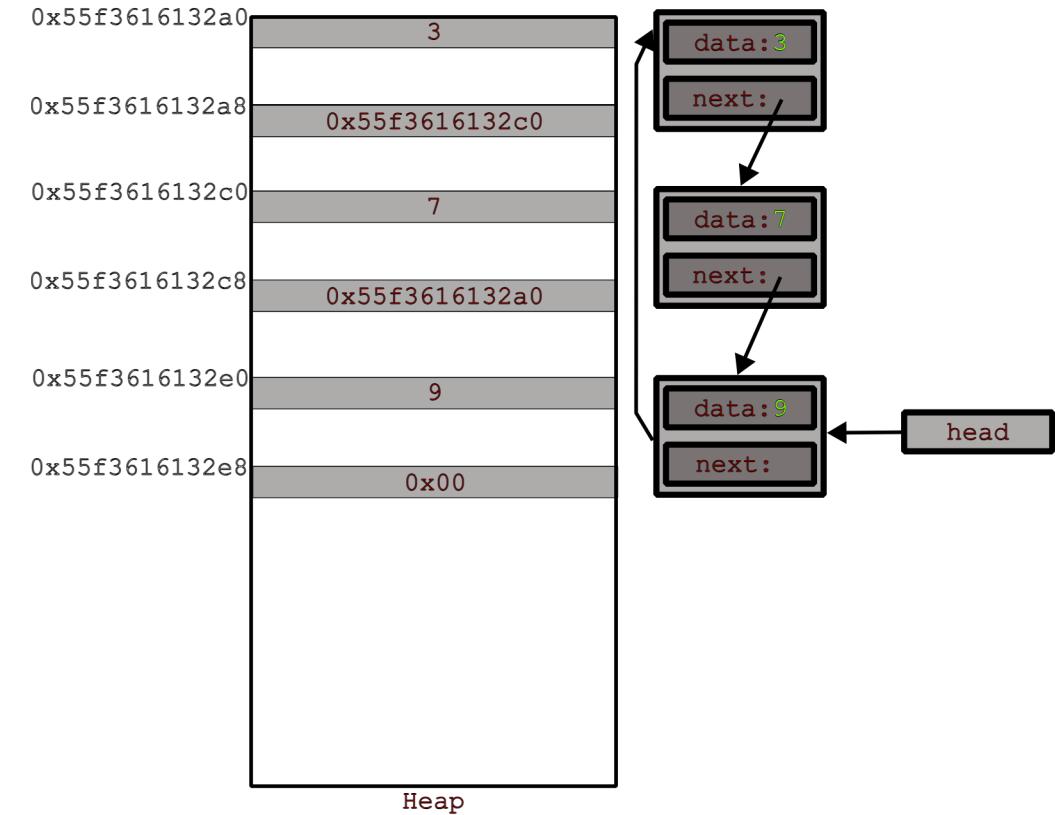


Cabecera

```
typedef struct{
    char *name;
    int age;
} Person;

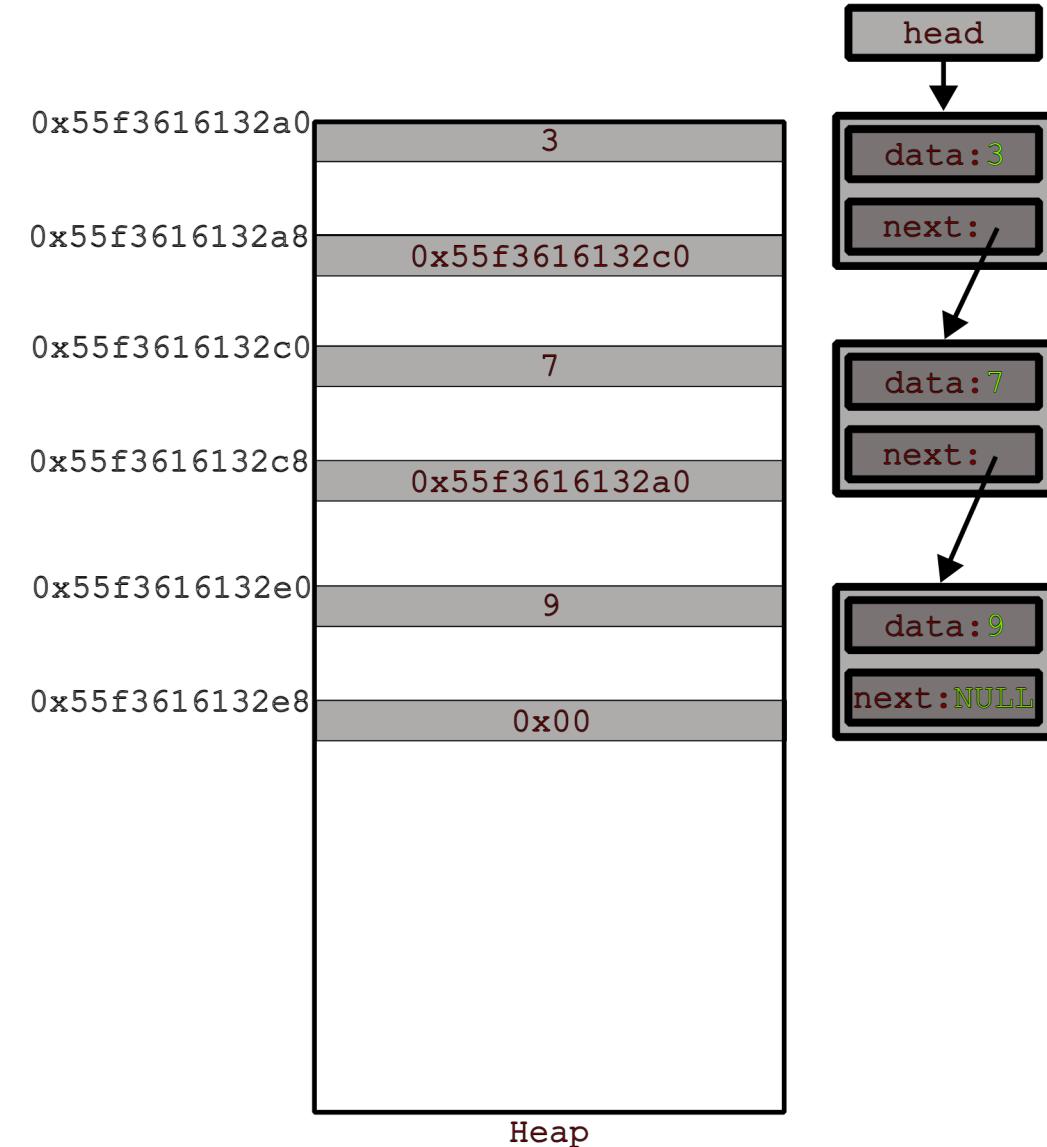
typedef struct Node *Queue;
struct Node{
    Person person;
    struct Node *next;
};

Person *createPerson(char *name, int age);
void createQueue(Queue *ptrqueue);
bool isEmpty(Queue queue);
int size(Queue queue);
void enqueue(Queue *ptrqueue, Person person);
void dequeue(Queue *ptrqueue);
Person first(Queue queue);
void clear(Queue *ptrqueue);
void display(Queue queue);
```



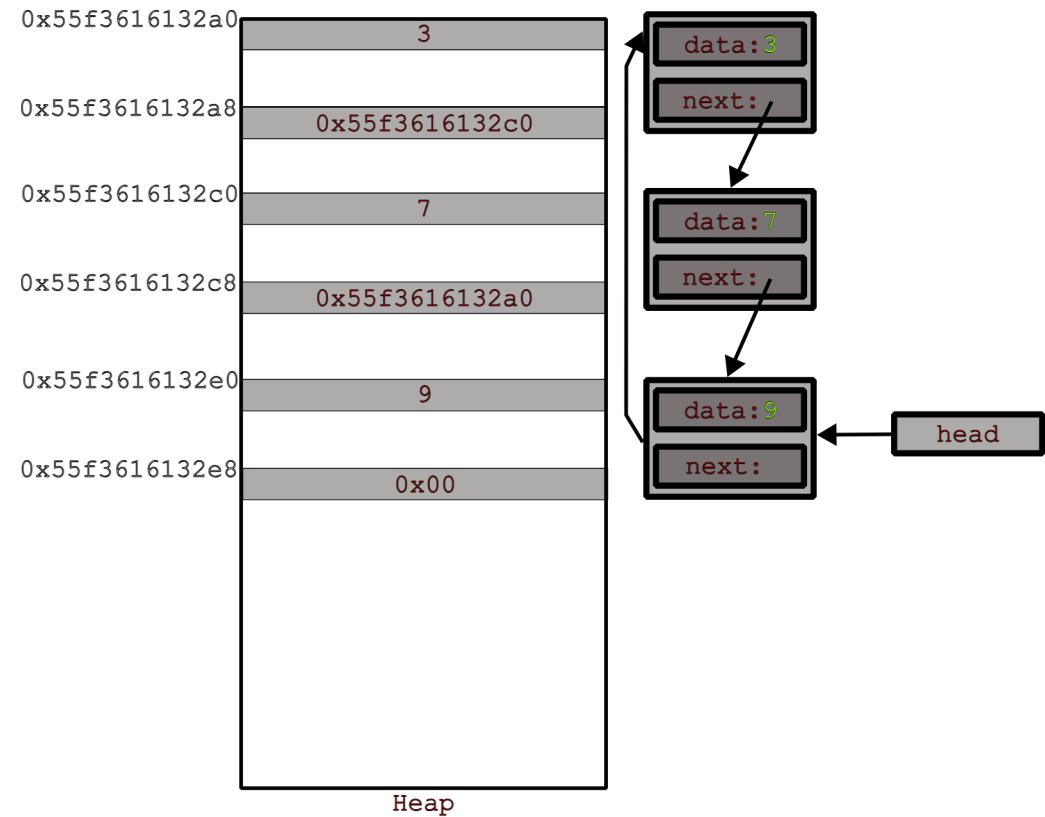
Ejercicio Pila

- Es una estructura *Last-In-First-Out*, por lo que se insertará en la cabeza y se extraerá de la cabeza de la lista.
- Vamos a implementarlo con una **lista enlazada**.
- *head* apunta al último elemento insertado (cabeza de la lista).



Cabecera

```
enum Tipo {PLASTICO, METAL, MADERA};  
typedef struct Node *PtrNode;  
  
struct Node{  
    int color;  
    float capacidad;  
    enum Tipo material;  
    struct Node *next;  
} Node;  
  
void push(PtrNode *ptrptrNode, struct Node element);  
PtrNode top(PtrNode pila);  
bool pop(PtrNode *ptrptrNode);  
void show(PtrNode pila);  
int size(PtrNode pila);  
bool isEmpty(PtrNode pila);  
void clear(PtrNode *ptrptrNode);
```



Ejercicio: Pila.zip