Sumario Pilas y Filas

Parcial	AutoEstudio

Pila (stack)

- **LIFO (Last In, First Out)**: El último que entra es el primero en salir.
- Se utiliza un arreglo y un entero `top` que indica la cima de la pila.
- Inicialmente, `top = -1` significa pila vacía.
- Se usa para **deshacer acciones**, **paréntesis**, **recursión**, etc.

```
#include <stdio.h>

#define MAX 5

// Estructura para la pila
typedef struct {
    int datos[MAX];  // Arreglo que guarda los datos
    int top;         // Índice del tope de la pila
} Stack;

// Inicializa la pila vacía
void init(Stack *pila) {
    pila->top = -1; //pila vacia
}

// Verifica si la pila está vacía
int isEmpty(Stack *pila) {
    return pila->top == -1;
}

// Verifica si la pila está llena
```

```
int isFull(Stack *pila) {
    return pila->top == MAX - 1;
}

// Agrega un valor a la pila
void push(Stack *pila, int valor) {
    if (!isFull(pila)) {
        pila->top++;
        pila->datos[pila->top] = valor;
    } else {
        printf("La pila está llena\n");
    }
}

// Elimina y retorna el valor del tope
int pop(Stack *pila) {
    if (!isEmpty(pila)) {
        int val = pila->datos[pila->top];
        pila->top--;
        return val;
    } else {
        printf("La pila está vacía\n");
        return -1;
    }
}

// Muestra el valor del tope sin eliminarlo
int peek(Stack *pila) {
    if (!isEmpty(pila)) {
        return pila->datos[pila->top];
    } else {
        return -1;
    }
}

// Imprime el contenido actual de la pila
```

```
void print(Stack *pila) {
    if (isEmpty(pila)) {
        printf("La pila está vacía\n");
    } else {
        printf("Pila: ");
        for (int i = pila->top; i >= 0; i--) {
            printf("%d ", pila->datos[i]);
        }
        printf("\n");
    }
}
```

Fila circular (Queue)

- **FIFO (First In, First Out)**: El primero que entra es el primero en salir.
- Se usan dos índices: `front` y `rear`.
- El arreglo es circular con `%` (módulo): si se llega al final, se vuelve al inicio.
- Condición de **fila vacía**: `front == -1`
- Condición de **fila llena**: `(rear + 1) % MAX == front`

```
#include <stdio.h>

#define MAX 5

// Estructura para la fila circular
typedef struct {
    int datos[MAX]; // Arreglo circular
    int front;      // Índice del primer elemento
    int rear;       // Índice del último elemento
} Queue;

// Inicializa la fila vacía
void initQueue(Queue *q) {
```

```
    q->front = -1;
    q->rear = -1;
}

// Verifica si la fila está vacía
int isEmpty(Queue *q) {
    return q->front == -1;
}

// Verifica si la fila está llena (modo circular)
int isFull(Queue *q) {
    return (q->rear + 1) % MAX == q->front;
}

// Inserta un valor al final de la fila
void enqueue(Queue *q, int valor) {
    if (isFull(q)) {
        printf("La fila está llena\n");
        return;
    }

    if (isEmpty(q)) {
        q->front = 0;
        q->rear = 0;
    } else {
        q->rear = (q->rear + 1) % MAX;
    }

    q->datos[q->rear] = valor;
}

// Elimina y retorna el valor del frente de la fila
int dequeue(Queue *q) {
    if (isEmpty(q)) {
        printf("La fila está vacía\n");
        return -1;
```

```
    }

    int val = q->datos[q->front];

    if (q->front == q->rear) {
        // Solo había un elemento
        q->front = -1;
        q->rear = -1;
    } else {
        q->front = (q->front + 1) % MAX;
    }

    return val;
}

// Muestra el valor al frente de la fila
int peek(Queue *q) {
    if (!isEmpty(q)) {
        return q->datos[q->front];
    }
    return -1;
}

// Imprime todos los elementos de la fila circular
void printQueue(Queue *q) {
    if (isEmpty(q)) {
        printf("Fila vacía\n");
        return;
    }

    printf("Fila: ");
    int i = q->front;
    while (1) {
        printf("%d ", q->datos[i]);
        if (i == q->rear) break;
        i = (i + 1) % MAX;
```

```
    }
    printf("\n");
}
```

Recursividad

- Una función se llama a sí misma para resolver un problema en partes.
- Siempre necesita:
    1. **Caso base**: condición para detenerse.
    2. **Llamada recursiva**: repetir con menor tamaño.

Factorial

```
int factorial(int n) {
    if (n <= 1) return 1;  // Caso base
    return n * factorial(n - 1); // Paso recursivo
}
```

Suma de numeros de 1 a n

```
int suma(int n) {
    if (n == 0) return 0;
    return n + suma(n - 1);
}
```

Recorrer lista enlazada recursivamente

```
void imprimirRecursivo(Nodo *n) {
    if (n == NULL) {
        printf("NULL\n");
        return;
    }
    printf("%d → ", n→valor);
    imprimirRecursivo(n→siguiente);
}
```

Notas utiles para el examen

- `*` se usa para **declarar o acceder** a punteros.
- `→` se usa para acceder a miembros de una estructura a través de un puntero.
- `%` se usa en fila circular para **volver al inicio del arreglo** (circularidad).