DEPARTAMENTO DE ELECTRONICA

---

**Ayudantía 5: Stack, buffer circular y recursividad**

**Objetivo Principal:** Conocer cómo funcionan e implementar stacks, buffer circular y entender la recursividad.
**Autor:** Kevin Pizarro Aguirre

---

# 1. Stack o pila

La "filosofía" detrás del stack o pila es que podemos meter datos empujando los demás puestos anteriormente, ahora si se desea sacar datos se deberá sacar aquél que fue puesto más recientemente, esto es un filosofía LIFO (Last in, First out). En principio existen dos grandes formas de implementar un stack, la primera es a través de arreglos y la segunda a través de listas simplemente enlazadas, nos centraremos en la segunda por sus ventajas en cuando memoria dinámica.

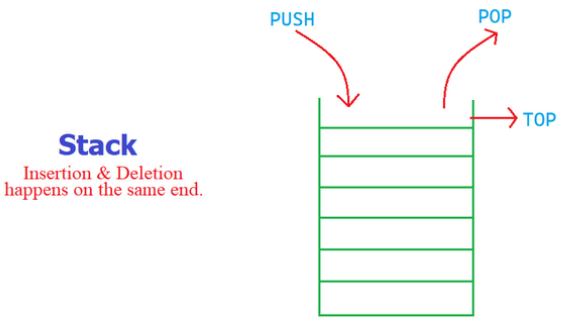


Figura 1: Diagrama para stack.

De la figura 1 se puede notar que nuestra cabeza es nuestro *top*, por lo que cada vez que se monte o ingrese un nuevo elemento entonces el top apuntará a ese nuevo elemento añadido, de manera análoga sucede con la función pop cuando se elimine un elemento. Se pueden crear otras funciones útiles como por ejemplo si nuestro stack está vacío, saber que tan grande es nuestro stack o imprimir la información de todos los elementos almacenados.

## 2. Buffer circular

Al igual que con el stack, un buffer circular puede ser implementado de varias formas, aunque las más comunes son con listas simplemente enlazadas, doblemente enlazadas y arreglos. Lo importante es conocer cual es la filosofía detrás para poder hacer modificaciones a lo que ya se conoce y obtener los resultados que deseamos.

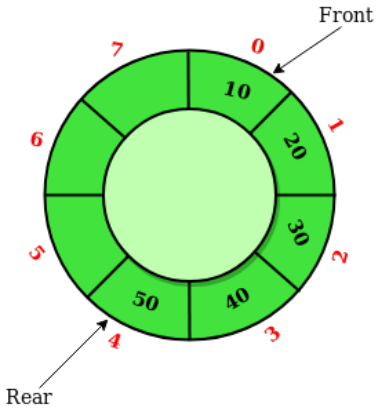


Figura 2: Diagrama para buffer circular.

Tal como se ve en la figura 2, se deberán tener 2 punteros uno para el frente o cabeza y otro para la parte trasera o cola, además se debe especificar el tamaño máximo. Si se deseara colocar más elementos que el tamaño máximo permitido, entonces hay que elegir que hacer con el último elemento, se tienen dos grandes opciones: descartar ese último elemento o sobrescribir el elemento más viejo del buffer. Para la implementación de las funciones para añadir elementos o retirarlos es necesario seguir su filosofía FIFO (First in, first out), el primer elemento que entra es el primero que saldrá.

# 3. Recursividad

## 3.1. Recursividad

### 3.1.1. Recursividad

Imagine operaciones que se repiten hasta una una determinada condición. Algunas de las opciones podrían ser una sumatoria, productoria, cálculo de factorial, cálculo de Fibonacci. En el cálculo del factorial de 6, se debe multiplicar por el entero inmediatamente inferior (5 en primera iteración) hasta que llegue a la multiplicación por 1. De esta forma podemos implementar el cálculo del factorial de un número a través de funciones recursivas, que se llamen a si mismas hasta que cumpla una determinada condición.

```
#include <stdio.h>
int fibo(int n)
{
   if (n <= 1) return n;
   return fibo(n-1) + fibo(n-2);
}
```

Desarrollaremos el ejemplo anterior para n = 4. Entramos con fibo(4).

$$\begin{aligned} \text{fibo}(4) &= \text{fibo}(3) + \text{fibo}(2) \\ &= (\text{fibo}(2) + \text{fibo}(1)) + (\text{fibo}(1) + \text{fibo}(0)) \\ &= (\text{fibo}(1) + \text{fibo}(0) + 1) + (1 + 0) \\ &= (1 + 0 + 1) + (1 + 0) \\ &= 3 \end{aligned}$$

Con la idea anterior se puede realizar múltiples operaciones útiles. Una de las aplicaciones que le daremos será en las implementaciones de funciones en los árboles binarios de búsqueda, aunque también se puede aplicar en cualquier otra estructura o problema que se encuentre.

# 4. Ejercicios

## 4.1. Stack

Usted está trabajando de desarrrollador de videojuegos para una compañía prestigiosa, llamada Mimtendo. Una de las tareas semanales que le han encargado es implementar parte del sistema de backtracking para un minijuego del estilo *Rat in a Maze*.

a) Sea un mapa bidimensional de 5x5. Si el ratón ocupa todas las casillas sólo una vez, entonces ¿de qué tamaño debe ser el stack?

b) Considere que al moverse a una casilla nueva se debe registrar con la función de prototipo void new_move(int x, int y).

c) Si se arrepiente del último movimiento puede desecharlo con la función de prototipo void back().

d) Implemente una función que imprima los puntos que ha recorrido, desde el más reciente hasta el más viejo.

e) Pruebe el flujo del programa con una secuencia del tipo: (0,0) → (0,1) → (0,2) → (1,2) → (2,2) → (2,3) → (2,2) → (3,2) → (4,2) → (4,3) → (4,4).

## 4.2. Buffer circular

Una de las aplicaciones que se les puede dar a los buffer circular es para solucionar el problema del *productor-consumidor*. Donde se tiene un productor que envía datos y luego un consumidor que los lee bajo un almacenamiento que es finito, esto se presenta en las redes de computadores, problemas de producción y otras áreas.

a) Está programando un microcontrolador y desea leer los datos recibidos por comunicación serial. El tamaño máximo es de 20 elementos. Implemente una función de prototipo void new_data(int data) donde simule que ha llegado un nuevo dato al buffer (producción).

b) Cree otra función donde simule la lectura (consumir el dato) con prototipo void read_data().

c) Simule que le llegan datos cada 1 segundo y que los lee cada 2 segundos. Luego simule la situación inversa, llegan datos cada 2 segundos y lee cada 1 segundo. ¿Qué debería suceder cuando se llegue al límite?, ¿la solución es la deseada? y si no es la deseada ¿cuáles serían las potenciales soluciones?

## 4.3. Recursividad

a) Implemente una función recursiva que le permita calcular $\sum_{n=a}^{b} n$, donde a y b son parámetros de la función. Asuma a y b números enteros positivos.

b) Utilice recursividad para encontrar el número más grande dentro de un arreglo. ¿Cómo podría hacer lo mismo para otras estructuras de datos? y si ahora se deseara ordenar de forma ascendente ¿cómo lo implementaría con recursividad?