

# ENTREGA 1: ESTRUCTURAS LINEALES

Simulacion de un supermercado con  
atencion prioritaria



# OBJETIVOS DEL PROGRAMA



## Simular un comportamiento real

Simular el flujo de atención de clientes en un supermercado, aplicando estructuras de datos lineales para modelar el comportamiento real.



## Servicio prioritario

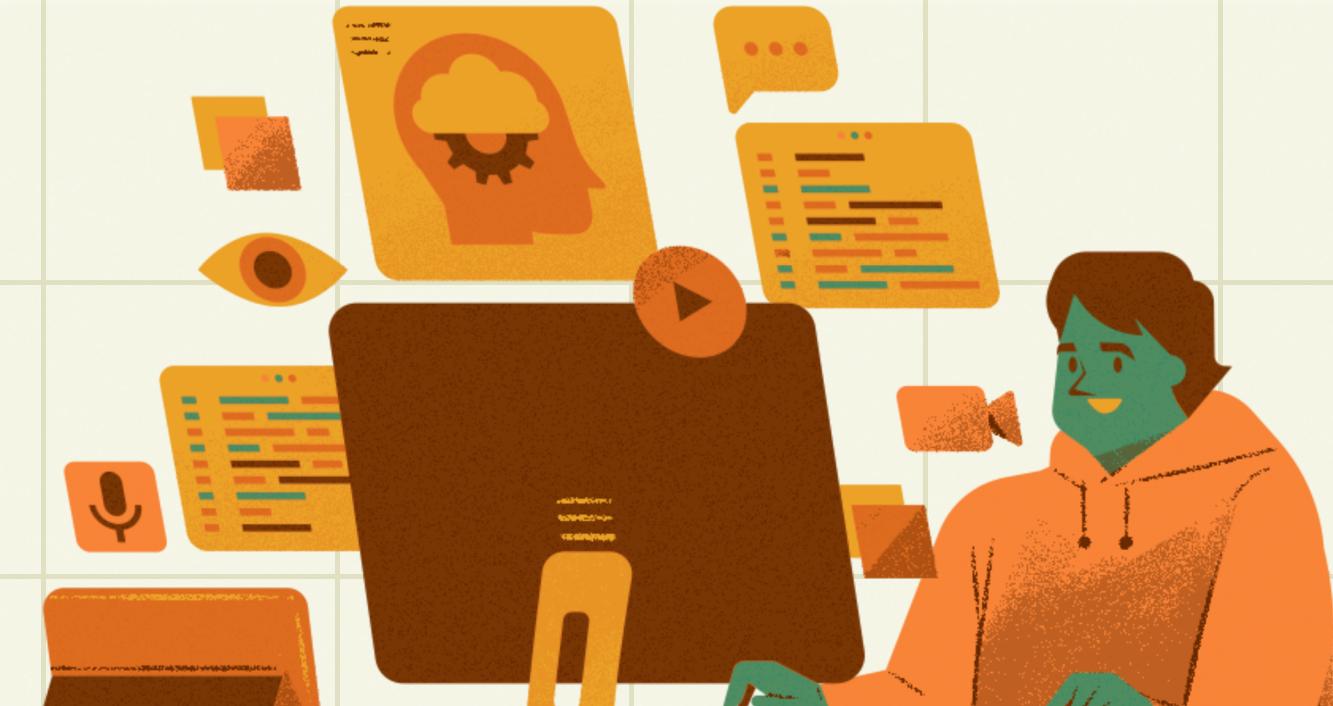
Otorgar prioridad a ciertos tipos de clientes en la fila (discapacidad, adultos mayores o embarazadas).



## Guardar registro

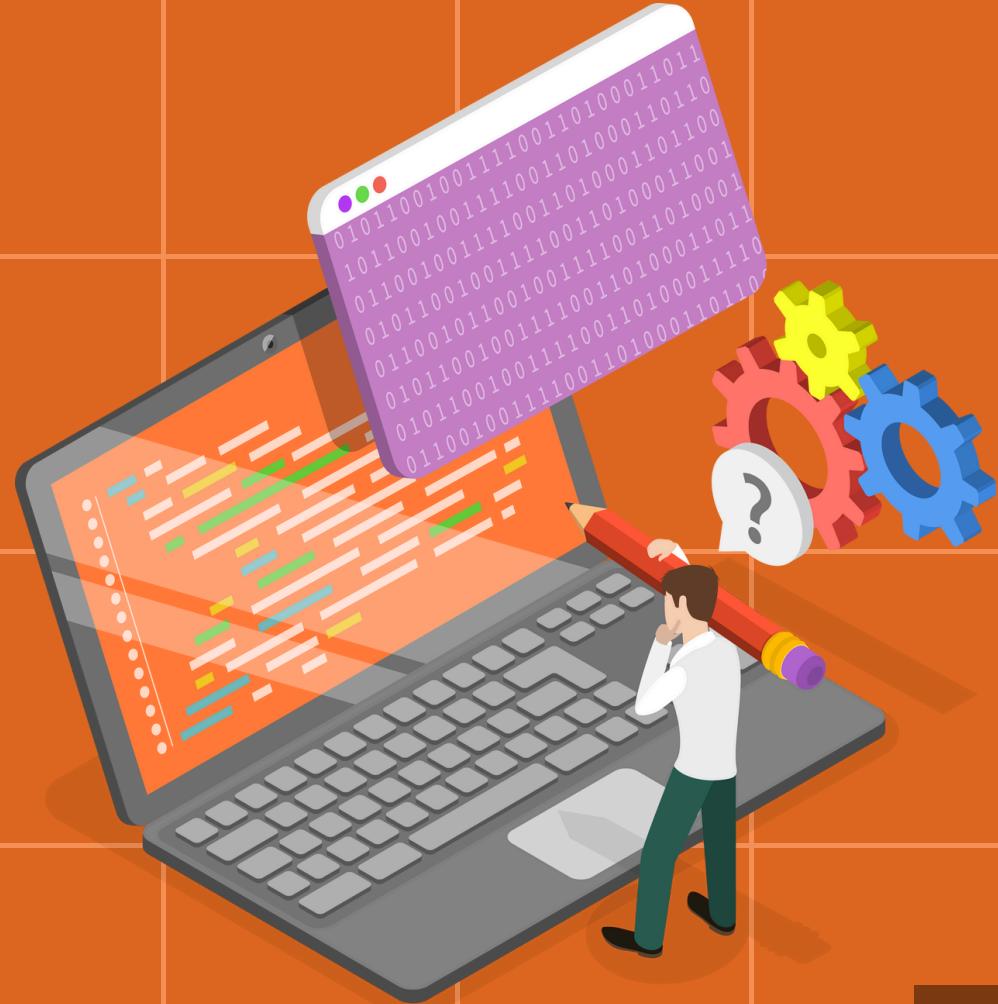
Incorporar estructuras lineales vistas en clase con el fin

# LIBRERIAS USADAS



```
#include <iostream>      // Estandar, entrada y salida de datos mediante cin y cout
#include <stack>         // Estructura Pila
#include <queue>         // Estructura de cola y cola con prioridad
#include <string>         // Para usar los string
#include <vector>         // Permite usar vectores dinamicos
#include <cstdlib>        // rand(), srand() y otras funciones de utilidad
#include <ctime>          // Sirve para usar el time()
#include <thread>         // Permite generar hilos de ejecucion
#include <chrono>         // Se usa en conjunto con la libreria thread para pausas y tiempos
#include <cctype>          // Sirve para usar la funcion toupper y tolower
#include <limits>          // Sirve para usar la funcion numeric_limits para limpiar cin
#include <tuple>           // Para usar tuplas
```

# ESTRUCTURAS DE DATOS USADAS



## Cola prioritaria

Clase ColaPrioritariaD1  
Gestiona el orden de atención de los clientes.  
Usa el ComparadorPrioridad para decidir quién pasa primero.

## Pila

Clase CarritoDeCompras  
Almacena los productos del cliente (LIFO).  
Métodos: push, pop, mostrarProductos.

## Arreglo dinámico

Se usa para almacenar las facturas generadas.  
Permite mostrar el historial de atención en orden.

# LOGICA DE PRIORIDAD

Se define una estructura Cliente con atributos que definen su prioridad: discapacidad, adultoMayor, embarazada, ordenLlegada.

La cola de prioridad se define con el tipo de datos (Clientes), donde se almacenaran (Un vector) y el comparador de su prioridad.

El comparador (ComparadorPrioridad) asigna una puntuación a 2 clientes:  
3 → prioridad alta (condiciones especiales)  
2 → prioridad media (menos de 5 productos)  
1 → prioridad normal

Si dos clientes tienen la misma prioridad, se atiende el que llegó primero.



# CODIGO DEL COMPARADOR

```
struct ComparadorPrioridad {
    bool operator()(const Cliente& a, const Cliente& b) const {
        auto prioridad = [](const Cliente& c) {
            if (c.discapacidad || c.adultoMayor || c.embarazada) return 3;
            if (c.carrito.size() < 5) return 2;
            return 1;
        };

        int pa = prioridad(a);
        int pb = prioridad(b);

        if (pa != pb) return pa < pb;
        return a.ordenLlegada > b.ordenLlegada;
    }
};
```



# PROCESAMIENTO DEL CARRITO

Asigna un precio aleatorio (1000–20000) a cada producto, calcula el total de la compra, y, muestra los productos y sus precios en consola.

(En el futuro puede aparecer como un diccionario con la <key> siendo el nombre del producto y la <value> siendo el precio)

Los resultados se almacenan como una Factura:  
Atributos: nombre, productos, total y fecha/hora.

Las facturas se encolan en colaFacturas para mostrar el historial final.

```
int procesarCarrito(const string& nombreCliente, stack<string> carrito) {
    srand(static_cast<unsigned>(time(0)) + rand());
    int total = 0;
    vector<pair<string, int>> productosFactura;

    cout << ANS_YELLOW << "Procesando carrito...\n" << ANS_RESET;
    while (!carrito.empty()) {
        string producto = carrito.top();
        carrito.pop();

        int precio = 1000 + rand() % 19001;
        cout << " - " << producto << ":" $" << precio << endl;
        total += precio;

        productosFactura.push_back({producto, precio});
    }
}
```

```
Factura nueva(nombreCliente, productosFactura, total, fechaHora);
colaFacturas.push(nueva);

return total;
```

# CODIGO DEL PROCESAMIENTO

# INTERFAZ VISUAL EN CONSOLA

Uso de códigos ANSI para aplicar colores, negrita y limpieza de pantalla.

Funciones auxiliares:

- `slowPrint()` → imprime texto con efecto de escritura.
- `clearScreen()` → limpia la pantalla.
- `waitForEnter()` → pausa el programa.
- `enableVirtualTerminalProcessingOnWindows()` → permite colores en Windows.



# EJEMPLO CODIGO INTERFAZ

```
const string ANS_RESET = "\033[0m";
const string ANS_BOLD = "\033[1m";
const string ANS_RED = "\033[31m";
const string ANS_GREEN = "\033[32m";
const string ANS_YELLOW = "\033[33m";
const string ANS_BLUE = "\033[34m";
const string ANS_MAGENTA = "\033[35m";
const string ANS_CYAN = "\033[36m";
const string ANS_WHITE = "\033[37m";

void slowPrint(const string &s, int ms = 3) {
    for (char c : s) {
        cout << c << flush;
        if (ms > 0) this_thread::sleep_for(chrono::milliseconds(ms));
    }
}
```

```
void enableVirtualTerminalProcessingOnWindows() {
#ifdef _WIN32
    HANDLE hout = GetStdHandle(STD_OUTPUT_HANDLE);
    if (hout == INVALID_HANDLE_VALUE) return;
    DWORD dwMode = 0;
    if (!GetConsoleMode(hout, &dwMode)) return;
    dwMode |= ENABLE_VIRTUAL_TERMINAL_PROCESSING;
    SetConsoleMode(hout, dwMode);
#endif
}

void waitForEnter() {
    cout << ANS_BOLD << ANS_CYAN << "\nPresiona Enter para continuar..." << ANS_RESET;
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
```

# GRACIAS POR SU ATENCION

Juan Bohorquez  
Julián Quintero  
Santiago Herrera

