



INSTITUTO POLITECNICO NACIONAL

Escuela Superior de Computo

Unidad de aprendizaje:

Algoritmos y estructura de datos

Practica 04

Simulaciones con el TAD Cola

Alumnos:

Murillo Barrientos Alexis Adrian

De La Peña Salazar Ian Rafael

Grupo: 2CM7

N° de boleta:

202463010



2024630393



Entrega para 9-junio-2024



Introducción

Los TAD (Tipos Abstractos de Datos) son herramientas fundamentales en la programación, ya que permiten modelar y manipular datos de forma organizada y eficiente. Entre los TAD más utilizados se encuentran las colas, estructuras de datos que siguen el principio **Primero en Entrar, Primero en Salir (FIFO)**.

Las colas se caracterizan por dos operaciones básicas:

1. **Encolar (enqueue):** Inserta un nuevo elemento al final de la cola.
2. **Desencolar (dequeue):** Elimina el primer elemento de la cola y lo devuelve.

Adicionalmente, suelen incluir operaciones como:

- **Vacía (empty):** Indica si la cola está vacía.
- **Primero (front):** Devuelve el primer elemento de la cola sin eliminarlo.
- **Tamaño (size):** Devuelve el número de elementos en la cola.

Entre otras muchas operaciones que faciliten el manejo de los TAD.

Las colas tienen una amplia gama de aplicaciones en diversos campos, como:

- **Simulación de sistemas de espera:** Modelos de colas se utilizan para analizar el comportamiento de sistemas como filas de espera en bancos, aeropuertos o supermercados.
- **Procesamiento de tareas:** En sistemas operativos, las colas se emplean para gestionar la ejecución de tareas en orden de llegada.
- **Comunicación entre procesos:** Las colas permiten la comunicación entre procesos concurrentes de forma asíncrona.
- **Recopilación de datos:** Las colas se utilizan para almacenar temporalmente datos antes de procesarlos o almacenarlos de forma permanente.

Las colas son herramientas valiosas para simular sistemas de atención en los que los elementos llegan y se procesan en un orden específico, como en supermercados, bancos u oficinas de atención al cliente. En este contexto, las colas permiten modelar el flujo de clientes o usuarios, la asignación de recursos y la gestión de tiempos de espera.

Simulación de la atención en un supermercado

En un supermercado, las colas se pueden utilizar para simular la espera de los clientes en las cajas de pago. Cada cliente se representa como un elemento en la cola, y el cajero atiende a los clientes uno por uno, desencilándolos de la cola. La simulación puede considerar aspectos como la cantidad de cajas abiertas, la velocidad de atención de cada cajero y la distribución de los productos que compran los clientes.

Simulación de la ejecución de procesos en un sistema operativo

En un sistema operativo, las colas se pueden utilizar para simular la ejecución de procesos. Cada proceso se representa como un elemento en la cola, y el procesador ejecuta los procesos uno por uno, desencilándolos de la cola. La simulación puede considerar aspectos como la prioridad de los procesos, el tiempo de ejecución de cada proceso y la disponibilidad de recursos del sistema.

Simulación de la atención en un banco cuidando las políticas de atención a clientes con prioridad

En un banco, las colas se pueden utilizar para simular la atención de clientes, considerando las políticas de atención a clientes con prioridad. Se pueden crear diferentes colas para clientes VIP, clientes regulares y clientes con necesidades especiales. La simulación puede considerar aspectos como el tiempo de espera de cada tipo de cliente, la cantidad de personal disponible para cada tipo de cola y la tasa de llegada de clientes.

Ventajas del uso de TAD Cola para simulación

El uso de TAD Cola para simular sistemas de atención ofrece diversas ventajas:

- **Modelado preciso del flujo de elementos:** Las colas permiten modelar de forma precisa el orden de llegada y atención de los elementos en el sistema.
- **Análisis del comportamiento del sistema:** La simulación permite analizar el comportamiento del sistema bajo diferentes condiciones, como cambios en

la cantidad de recursos, la tasa de llegada de elementos o las políticas de atención.

- **Identificación de cuellos de botella:** La simulación puede ayudar a identificar los cuellos de botella en el sistema, es decir, los puntos donde se acumula la mayor cantidad de elementos y se generan los tiempos de espera más largos.
- **Evaluación de diferentes estrategias:** Se pueden evaluar diferentes estrategias de atención al cliente o de gestión de recursos mediante la simulación, comparando su impacto en el tiempo de espera, la utilización de recursos y la satisfacción de los usuarios.

Marco Teórico:

El concepto de cola data de la antigüedad, con aplicaciones en diversos ámbitos como la gestión de recursos y la organización de tareas. Sin embargo, su formalización como estructura de datos se atribuye a **Edsger Dijkstra** en 1959, en su trabajo sobre la sincronización de semáforos.

Desde entonces, las colas han sido objeto de un amplio estudio y desarrollo, dando lugar a diversas implementaciones y variantes. Entre las implementaciones más comunes se encuentran:

- **Cola lineal:** Una cola lineal es la estructura de cola más básica. Se implementa como una lista enlazada o un arreglo, donde los elementos se insertan en un extremo (final) y se eliminan por el otro (frente).

Las colas lineales son eficientes para almacenar y procesar elementos en orden de llegada. Se utilizan comúnmente en sistemas de colas de espera, procesamiento de tareas y algoritmos de ordenación.

- **Cola circular:** Una cola circular es una variante de la cola lineal que optimiza el uso de la memoria. En lugar de almacenar los elementos en un arreglo lineal, la cola circular utiliza un arreglo circular, donde el final de la cola se envuelve al principio. De esta manera, se evita la reasignación de memoria a medida que se insertan y eliminan elementos.

Las colas circulares son especialmente útiles en situaciones donde la memoria es limitada o donde se requiere un procesamiento rápido de

elementos. Se utilizan comúnmente en sistemas embebidos, búferes de datos y algoritmos de red.

- **Cola con prioridad:** Una cola de prioridad es un tipo de cola que ordena los elementos según una prioridad asignada. A diferencia de las colas lineales y circulares, donde los elementos se procesan en orden de llegada, las colas de prioridad procesan los elementos con mayor prioridad primero.

Las colas de prioridad son esenciales en aplicaciones donde se requiere un procesamiento rápido de elementos según su importancia. Se utilizan comúnmente en algoritmos de búsqueda, planificación de tareas y sistemas de simulación.

Las colas también han sido objeto de análisis en cuanto a su eficiencia y rendimiento. Se han desarrollado algoritmos para realizar operaciones básicas como encolar y desencolar en tiempo constante o amortizado logarítmico.

Especificación genérica de los TAD Cola:

Cabecera

nombre: Cola(QUEUE)

lista de operaciones:

- **Inicializar cola(Initialize):** recibe una cola y la inicializa para su trabajo normal.
- **Encolar(Queue):** recibe una cola y un elemento y agrega el elemento al final de ella.
- **Desencolar(Dequeue):** recibe una cola y remueve el elemento del frente retornándolo.
- **Es vacía(Empty):** recibe la cola y devuelve verdadero si está vacía.
- **Frente(Front):** recibe una cola y retorna el elemento del frente.
- **Final(Final):** recibe una cola y retorna el elemento del final.
- **Elemento(Element):** recibe una cola y un número de elemento de uno al tamaño de la cola y retorna el elemento de esa posición.
- **Eliminar cola(Destroy):** recibe una cola y la libera completamente.
- **Tamaño(Size):** recibe una cola y devuelve el tamaño de ésta.

Planteamiento del problema

En esta práctica se busca el manejo de los TAD cola y diseñar visualmente 3 casos distintos los cuales serán simular atenciones de clientes de un supermercado y un banco este último tomando en cuenta las políticas de prioridad en su clientela, así como, la simulación de ejecución de procesos en un sistema operativo.

Simular la atención de clientes en un supermercado

Objetivo:

Desarrollar un simulador que utilice un TAD (Tipo Abstracto de Datos) cola para modelar la atención de clientes en un supermercado. El simulador debe considerar los siguientes aspectos:

- **Entrada:**
 - Nombre del supermercado
 - Número de cajeros disponibles
 - Tiempo de atención promedio por cada cajero (en milisegundos)
 - Tiempo promedio de llegada de nuevos clientes (en milisegundos)
- **Salida:**
 - Llegada de los clientes a las colas de las cajas.
 - Clientes en espera de cada cola.
 - Cliente que es atendido en cada caja.
 - Número total de clientes atendidos.
 - Momento en que se puede cerrar la tienda (cuando se atienden 100 clientes y no hay clientes en espera).

Restricciones:

- El tiempo de atención y de llegada de los clientes deben ser múltiplos de 10 milisegundos.
- Los identificadores de los clientes son un número consecutivo único para cada uno.

- Al iniciar el número de clientes en cada fila es 0.
- número de cajas $0 < n < 11$.
- Cuando un cliente llegue a formarse, este selecciona aleatoriamente una cola para formarse y no cambio de cola hasta que es atendido.
- La simulación debe continuar hasta que se atiendan 100 clientes o hasta que no haya clientes en espera y se hayan atendido 100 clientes.
- Se debe registrar el nombre del supermercado, el número de cajeros, el número total de clientes atendidos y el momento en que se puede cerrar la tienda.

Metodología:

1. **Desarrollo del simulador:** Crear un programa que permita ingresar los datos de entrada (nombre del supermercado, número de cajeros, tiempo promedio de atención, tiempo promedio de llegada) e inicializar la simulación visual.
2. **Simulación de llegada de clientes:** Generar aleatoriamente la llegada de nuevos clientes. Cada cliente debe tener un nombre.
3. **Asignación de clientes a cajeros:** Utilizar un algoritmo de colas para asignar los clientes a las cajas, este seleccionando aleatoriamente una cola para formarse.
4. **Simulación del tiempo de atención:** Simular el tiempo de atención de cada cliente. Durante este tiempo, el cliente debe permanecer en la cola de atención del cajero asignado.
5. **Actualización de la cola de clientes en espera:** Una vez que un cliente es atendido, se debe eliminar de la cola de atención del cajero correspondiente. Si la cola de clientes en espera está vacía, se debe registrar el momento en que se puede cerrar la tienda.
6. **Recopilación de resultados:** Registrar el número total de clientes atendidos y el momento en que se puede cerrar la tienda.
7. **Presentación de resultados:** Mostrar los resultados de la simulación en un formato claro y conciso visualmente, incluyendo el nombre del supermercado, el número de cajeros, el número total de clientes atendidos y el momento en que se puede cerrar la tienda.

Simular la ejecución de procesos en el sistema operativo

Objetivo:

Se creará un simulador que modele la ejecución de procesos en un equipo monoprocesador sin manejo de prioridades. El simulador leerá la entrada que contiene la información de los procesos y los almacenará en la cola de listos. Luego, simulará la ejecución de los procesos uno a la vez, desencilándolos de la cola de listos, ejecutándolos durante su tiempo de ejecución y finalmente encolándolos en la cola de terminados. El simulador debe considerar los siguientes aspectos:

- **Entrada:**
 - Nombre del proceso hola.
 - Actividad
 - ID
 - tiempo
- **Salida:**
 - Proceso en ejecución actual y sus datos.
 - ID y nombre del último proceso.
 - ID y nombre del proceso siguiente.
 - Cuando un proceso termina.
 - Cuando terminen todos los procesos.

Restricciones:

- El tiempo en cada quantum de tiempo para despacharlos es de un segundo.
- El proceso que se encontraba en ejecución se encola y se coloca al proceso desencilado en ejecución.

Metodología

La metodología para abordar este problema se basa en los siguientes pasos:

1. **Implementación del TAD cola:** Se implementarán las funciones del TAD cola utilizando un lenguaje de programación adecuado, como C o Python. Se asegurará la eficiencia y corrección de las implementaciones.
2. **Diseño del simulador:** Se diseñará la arquitectura del simulador, incluyendo los componentes principales y sus interacciones. Se definirán las estructuras de datos para almacenar la información de los procesos y el estado del sistema.
3. **Implementación del simulador:** Se implementará el simulador utilizando el TAD cola y las estructuras de datos diseñadas. Se incluirán las funciones para leer la entrada, simular la ejecución de los procesos y generar los resultados de la simulación.

Simular la atención de Clientes de un Banco cuidando las políticas de atención.

Objetivo:

El objetivo de este proyecto es desarrollar una simulación que modele la atención de personas en un banco, considerando las diferentes políticas de atención establecidas. La simulación utilizará el TAD cola para gestionar las filas de espera de clientes, usuarios y clientes preferentes.

Descripción del problema

El problema consiste en simular el funcionamiento de un banco con múltiples cajeros, atendiendo a clientes, usuarios y clientes preferentes, de acuerdo con las siguientes políticas:

- **Clientes del banco:** Son atendidos por cualquier cajero disponible y nunca dejan de ser atendidos.
- **Usuarios del banco:** Son atendidos según la disponibilidad de alguna caja, pero no pueden esperar más de 5 personas en la fila de clientes antes de ser atendidos.
- **Clientes preferentes:** Son atendidos por cualquier cajero disponible con mayor prioridad que los clientes y usuarios.

Parámetros de entrada

La simulación tendrá como parámetros de entrada los siguientes:

- **Número de cajeros en el banco:** Define la cantidad de cajeros disponibles para atender a los clientes.
- **Tiempo de atención de cada cajero:** Representa el tiempo que un cajero tarda en atender a un cliente.
- **Tiempo de llegada de los clientes del banco:** Indica el tiempo en el que llega un cliente.
- **Tiempo de llegada de los usuarios del banco:** Indica el tiempo en el que llega un usuario.
- **Tiempo de llegada de los clientes preferentes:** Indica el tiempo en el que llega un cliente.

Salida esperada

La simulación debe generar como salida la siguiente información:

- **Llegada de los clientes a las 3 filas del banco.**
- **Cliente en espera de cada fila.**
- **Cliente que es atendido en cada caja y su tipo(cliente, preferente o usuario).**
- **Cajeros sin realizar ninguna atención en caso de que así sea.**

Restricciones

- Al iniciar el número de personas en cada fila es 0.
- Número de cajeros $0 < n < 11$
- los identificadores de las personas son un número consecutivo único por tipo
 - P1(Preferente 1), C1(Cliente 1), U1(Usuario 1), etc.
- Importante considerar los tiempos dados en múltiplos de 10 ms.
- El banco nunca cierra

Metodología:

1. **Definición del TAD Cola:** Implementar el TAD Cola para gestionar las filas de espera de cada tipo de cliente.

2. **Simulación de la atención:**

Inicializar el sistema con los parámetros de entrada (número de cajeros, tiempos de atención y llegada).

Crear las colas para clientes, usuarios y clientes preferentes.

Simular la llegada de nuevos clientes de acuerdo con el tipo de cliente.

- Encolar al nuevo cliente en la cola correspondiente.

Simular la atención de clientes por parte de los cajeros disponibles:

- Si hay un cliente preferente en la cola, atenderlo primero.
- Si no hay clientes preferentes, atender al siguiente cliente en la cola de clientes o usuarios, siguiendo la política de atención.
- Simular el tiempo de atención del cliente atendido.
- Desencolar al cliente atendido.

Diseño y funcionamiento de la solución

//descripción de la abstracción del problema y su solución, apoyándose de diagramas y figuras en un lenguaje claro de cada simulación.

Simulación 1 “Supermercado”

Entidades:

- **Supermercado**
- **Cliente**
- **Caja**

Atributos:

- **Supermercado:**
 - **Nombre:** Nombre del supermercado.

- **Número de cajas:** Número de cajas disponibles para atender clientes.
- **Clientes atendidos:** Número total de clientes atendidos en el día simulado.
- **Tiempo de atención por caja:** Tiempo promedio de atención en cada caja (en milisegundos).
- **Tiempo de llegada de clientes:** Tiempo promedio de llegada de nuevos clientes (en milisegundos).
- **Cliente:**
 - **ID:** id único y consecutivo al anterior.
 - **Tiempo de llegada:** Momento en que el cliente llega al supermercado (en milisegundos).
 - **Tiempo de espera:** Tiempo que el cliente espera en la fila antes de ser atendido (en milisegundos).
 - **Tiempo de atención:** Tiempo que el cliente es atendido en una caja (en milisegundos).
- **Caja:**
 - **Cliente actual:** Almacena la referencia al cliente que está siendo atendido actualmente.
 - **Tiempo de atención:** Tiempo que le queda a la caja para finalizar la atención del cliente actual (en milisegundos).

Operaciones:

- **Llegada de cliente:** Simula la llegada de un nuevo cliente al supermercado, agregándolo a una fila de forma aleatoria.
- **Inicio de atención:** Se verifica en cada caja si hay clientes en espera, se selecciona al primer cliente de la fila y se inicia su atención.
- **Fin de atención:** Se simula la finalización de la atención de un cliente, actualizando los contadores y liberando la caja.
- **Cierre de simulación:** Se verifica si se ha atendido al menos 100 clientes y no hay clientes en espera. Si se cumple esta condición, se finaliza la simulación y se muestran los resultados.

Algoritmo de simulación

El algoritmo de simulación se puede implementar de la siguiente manera:

1. **Inicialización:** Se definen los parámetros de la simulación (nombre del supermercado, número de cajas, tiempos de atención y llegada de clientes). Se inicializan las colas de espera para cada caja y los contadores de clientes atendidos.
2. **Ciclo de simulación:**
 - Se genera un nuevo cliente con un ID único y consecutivo al anterior y un tiempo de llegada.
 - Se le asigna una caja y se agrega el cliente a su cola de espera.
 - Para cada caja ocupada:
 - Se decrementa el tiempo de atención restante.
 - Si el tiempo de atención restante llega a cero, se finaliza la atención del cliente actual y se actualiza el contador de clientes atendidos.
 - Si la caja queda libre, se verifica si hay clientes en espera y se inicia la atención del siguiente cliente de la fila.

Simulación 2 “Procesos del sistema operativo”

Entrada:

- Cantidad de procesos
- Propiedades de cada proceso:
 - Nombre (cadena)
 - Actividad (cadena)
 - ID (entero)
 - Tiempo de ejecución (segundos)

Salida:

- Simulación de la ejecución de los procesos, mostrando:

- Proceso en ejecución actual y sus datos (nombre, actividad, ID, tiempo total de ejecución)
- ID y nombre del último proceso que terminó
- ID y nombre del siguiente proceso a ejecutar
- Momento en que termina cada proceso
- Momento en que termina la ejecución de todos los procesos

Consideraciones:

- El tiempo de cada quantum de tiempo para despachar los procesos es de 1 segundo.
- Cuando un quantum de tiempo termina:
 - El proceso que se encontraba en ejecución se encola en la cola de listos.
 - El proceso que se desencola de la cola de listos se coloca en ejecución.

Abstracción del problema:

El problema se puede modelar utilizando dos colas:

- **Cola de listos:** Almacena los procesos que están esperando para ser ejecutados.
- **Cola de terminados:** Almacena los procesos que ya han terminado su ejecución.

Se define un ciclo principal que simula el paso del tiempo en quantum de 1 segundo. En cada ciclo, se realizan las siguientes acciones:

1. Se verifica si la cola de listos está vacía:

- Si está vacía, significa que todos los procesos han terminado y la simulación finaliza.
- Si no está vacía, se continúa al siguiente paso.

2. Se desencola el siguiente proceso de la cola de listos:

- Este proceso se considera el proceso en ejecución actual.

3. Se simula la ejecución del proceso actual durante 1 segundo:

- Se actualiza el tiempo total de ejecución del proceso.
- Se verifica si el proceso ha terminado de ejecutarse:
 - Si ha terminado, se pasa al siguiente paso.
 - Si no ha terminado, se continúa al siguiente ciclo.

4. Si el proceso actual ha terminado:

- Se envía el proceso a la cola de terminados.
- Si la cola de listos no está vacía, se desencola el siguiente proceso y se coloca en ejecución (se vuelve al paso 2).
- Si la cola de listos está vacía, significa que todos los procesos han terminado y la simulación finaliza.

Implementación de la solución

Simulación 1 “Supermercado”

Descripción de la función:

Este código implementa un simulador de supermercado en C. Te explico paso a paso lo que hace:

Se definen las variables necesarias, incluyendo variables para el tiempo, el número de clientes, el número de cajeras, y otras variables relacionadas con el manejo de colas y tiempos de atención.

Se solicita al usuario que ingrese el número de cajeras.

Se crean las colas para las cajeras y se solicitan los tiempos de atención para cada cajera.

Se solicita al usuario el tiempo de recepción de nuevos clientes.

En el ciclo principal, se simula el paso del tiempo. Se revisa si es momento de atender a un cliente en alguna de las cajas y si es momento de que llegue un nuevo cliente al supermercado.

Se manejan los clientes en las colas, atendiéndolos si es necesario y agregando nuevos clientes a las colas cuando llegan al supermercado.

Se muestra el estado de las colas en cada iteración del ciclo principal.

Cuando se han atendido 100 clientes y todas las colas están vacías, se cierra el supermercado.

Finalmente, se libera la memoria reservada para las colas y los tiempos de atención.

Este código utiliza las funciones de una estructura de datos de cola (Queue, Dequeue, Initialize, Empty, Size, Element) que no están definidas en el fragmento que proporcionaste. Presumo que estas funciones están definidas en algún otro lugar del código.

Simulación 2 “Procesos del sistema operativo”

Descripción de la función:

Este código en C simula la ejecución de procesos utilizando tres colas: listos, ejecución, y finalizados. Cada proceso es representado por una estructura elemento que contiene el nombre, tiempo de ejecución, actividad, y ID del proceso. El código realiza las siguientes operaciones:

- **Inicialización de Colas:** Se inicializan tres colas (listos, ejecucion, finalizados) para manejar los procesos en diferentes estados.
- **Ingreso de Procesos:** En un bucle while, el programa solicita al usuario ingresar los detalles de cada proceso (nombre, tiempo, actividad, ID) hasta que el usuario decida no añadir más procesos. Cada nuevo proceso se encola en listos.
- **Procesamiento de Entrada:** Para cada entrada de texto (fgets) y número (scanf), se realiza una limpieza de la entrada estándar para evitar problemas con caracteres de nueva línea residuales.
- **Simulación de Ejecución:** En otro bucle while, el programa simula la ejecución de los procesos. Mueve procesos de listos a ejecucion, decrementa el tiempo de ejecución, y luego, basado en el tiempo restante, mueve el proceso a finalizados o lo encola de nuevo en listos para continuar su ejecución.
- **Control de Tiempo:** Se incrementa una variable tiempo en cada iteración del bucle de ejecución para simular el paso del tiempo. Además, se utiliza Sleep(TIEMPO_BASE) para añadir un retraso real en la ejecución y hacer la simulación más perceptible.
- **Finalización:** Una vez que todos los procesos han sido movidos a finalizados, el programa imprime un mensaje indicando que todos los procesos han terminado y finaliza.

Aspectos importantes para considerar:

La estructura elemento y las funciones Initialize, Queue, Dequeue, Empty, y Front sugieren que se está utilizando una implementación de colas específica,

posiblemente definida por el usuario, pero el código de estas funciones y estructuras no se muestra.

TIEMPO_BASE es una constante que define el retraso en la simulación de ejecución de procesos, pero su valor no se especifica en el fragmento de código proporcionado.

La función Sleep se utiliza para añadir un retraso, lo cual indica que el código está destinado a ser ejecutado en un entorno Windows, ya que Sleep es una función específica de la API de Windows. En entornos Unix o Linux, se usaría sleep, que tiene una semántica diferente (segundos en lugar de milisegundos).

El manejo de entrada y salida estándar (scanf, fgets, printf, puts) se utiliza para interactuar con el usuario, recoger datos de los procesos, y mostrar información sobre la ejecución de los mismos.

Simulación 3 “Banco con política de atención”

Descripción de la función:

El programa es un ejemplo de simulación de atención de colas de clientes en un banco. Está escrito en lenguaje C99 y utiliza las librerías estándar stdio.h, windows.h y time.h. También incluye la implementación de una cola dinámica en el archivo "TADColaDin.h".

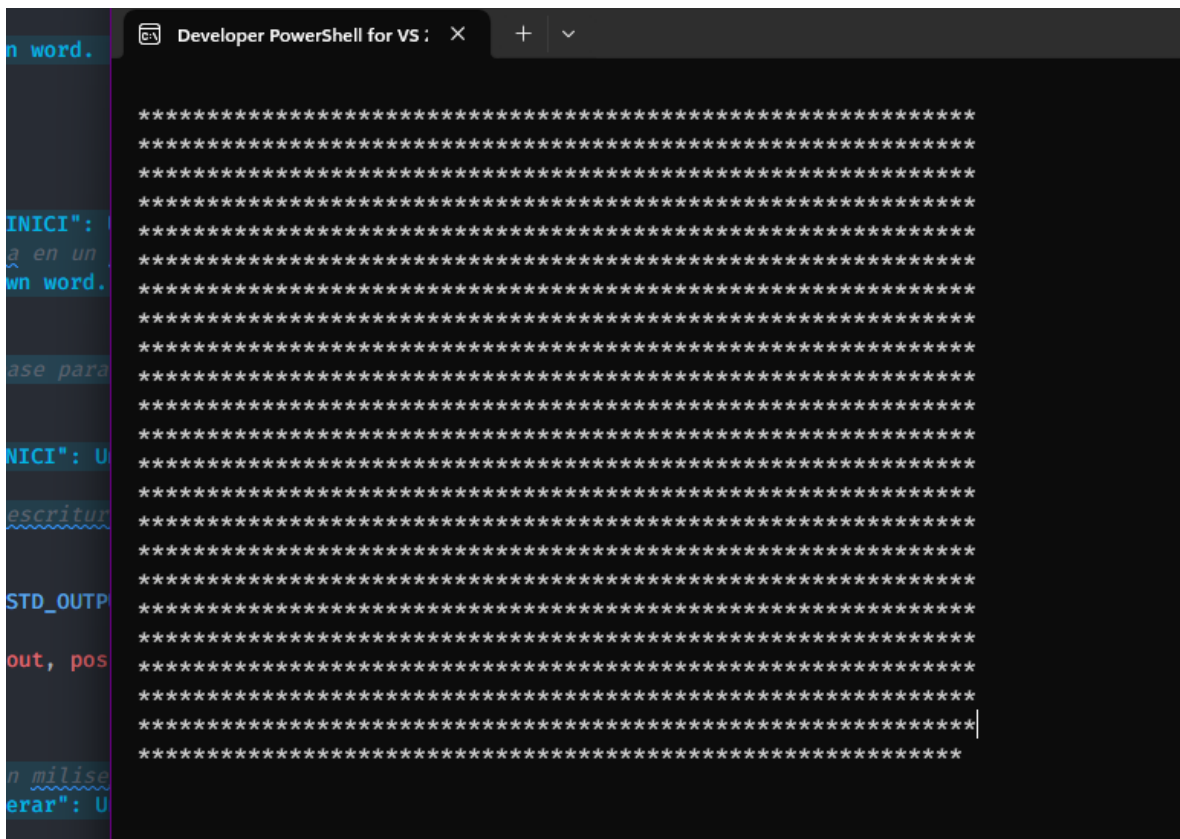
El programa simula la llegada de clientes, usuarios y preferentes al banco en intervalos de tiempo específicos. Cada tipo de cliente se forma en una cola separada: preferentes, clientes y usuarios. El programa muestra el número de clientes en cada cola en cada iteración.

Luego, el programa asigna a cada caja disponible un cliente para ser atendido. Primero, se atienden los clientes preferentes, luego los clientes regulares y finalmente los usuarios. Si no hay clientes en ninguna de las colas, se muestra un mensaje indicando que la caja está vacía.

Después de un tiempo de atención determinado, la caja se vacía y se vuelve a poner en estado disponible.

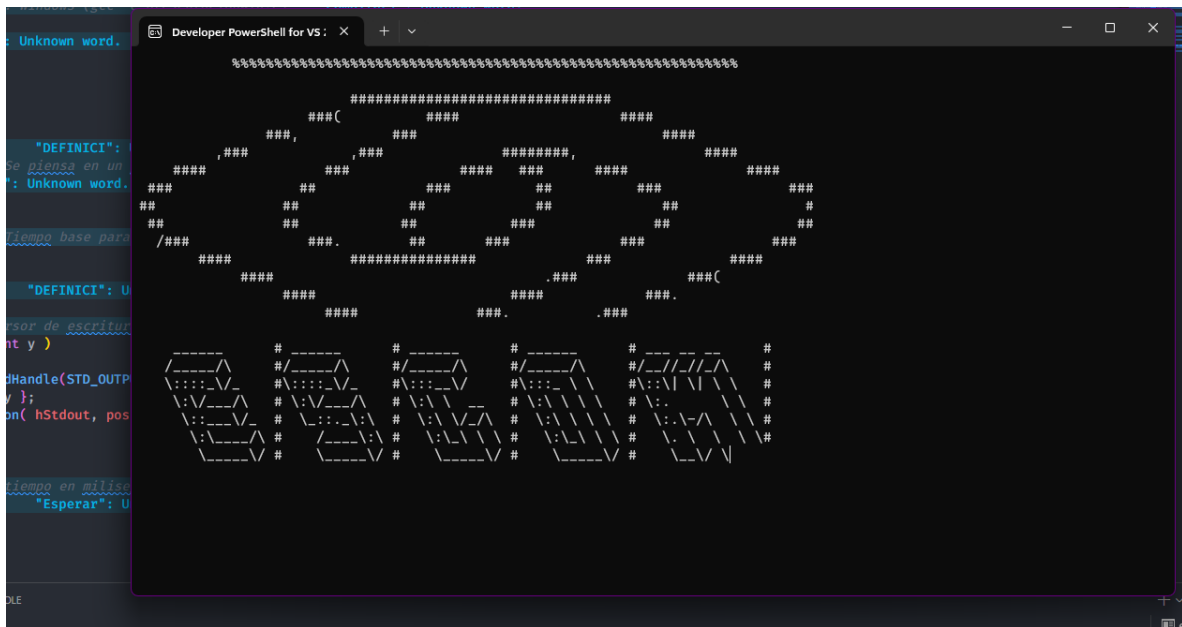
El programa se ejecuta en un ciclo infinito, simulando la llegada continua de clientes y la atención en las cajas.

Funcionamiento



The image shows a PowerShell console window titled "Developer PowerShell for VS : X". The window contains a large block of text consisting of many lines of asterisks, likely representing a simulation or a large output. On the left side, there is a vertical sidebar with some text, including "n word.", "INICI": U", "escriitur", "STD_OUTP", "out, pos", "n milise", and "erar": U".

Pantalla de inicialización



The screenshot shows a Developer PowerShell window for Visual Studio. The window title is "Developer PowerShell for VS: x". The terminal displays the output of a C program. At the top, there is a large ASCII art logo of the letter 'C' composed of hash symbols (#). Below the logo, the program prints several lines of text, including "DEFINICI: U" and "Esperar: U". The background of the terminal is dark, and the text is light gray.

Pantalla de presentación de escom



The screenshot shows a Developer PowerShell window for Visual Studio. The window title is "wsExmpl". The terminal displays the output of a C program. The program prints the following text: "INSTITUTO POLITECNICO NACIONAL", "Escuela Superior de Computo", "Unidad de aprendizaje: Algoritmos y estructura de datos", "Practica 04 : Simulaciones con el TAD Cola", "Alumnos:", "Murillo Barrientos Alexis Adrian", "De La Pe| a Salazar Ian Rafael", "Grupo: 2CM7", "N de boleta:", "2024630393", "2024630107", and "=====|". The background of the terminal is dark, and the text is light gray.

Pantalla de presentación autores



Agradecimiento

Errores detectados

A la hora de imprimir caracteres de como la línea invertida o que estas fuera del estándar ancii, tienden a imprimirse caracteres no deseados o no imprimir nada, lo cual resulta en ocasiones en palabras no entendibles o los formatos de cada transición de pantalla, salieran descuadrados

Posibles mejoras

1. Se puede implementar la librería de caracteres extendidos de C lo cual con un formateo especial en la función printf, se puede evitar los problemas gráficos.

Conclusiones

De La Peña Salazar Ian Rafael

En esta practica se logra apreciar la utilidad de los TAD de tipo Cola, en estos tres casos, que comparten la similitud de ser “servicios” donde se atiende a “clientes”, esto referido por la simulación de procesos de un sistema operativo donde se podría

decir que hay un cliente en espera y el sistema operativo es el del servicio, si bien nada mas se refuerza el aprendizaje en cuanto a los TAD Cola y su manejo también nos ayuda a manejar otras cosas como las librerías windows.h y unistd.h para el manejo de tiempo y el posicionamiento en la consola haciendo referencia a la función gotoxy(x, y) de conio.h, esto para dejar libertad en el diseño visual de los problemas.

Murillo Barrientos Alexis Adrian

Esta práctica me permitió practicar y entender mas sobre el funcionamiento de la librería grafica de Windows para C aunque dentro de lo que cabe es deficiente, es muy útil para hacer aplicaciones de consola , lo cual desde mi punto de vista me abrió un campo de oportunidad para el desarrollo de aplicaciones de consola, para múltiples usos, ya que no conocía que el uso de consola en C podría enriquecerse con windows.h, por otro lado, considero que en el tema de colas o queue, no fue un tema que realmente sintiera avance en su entendimiento , ya que no fueron problemas o ejercicios que me replantearán mi forma de usar o ver la aplicación de la cola.

Anexo

Códigos

- pruebasup.c

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <time.h>
#include "TADColaDin.h" // Assuming this is the correct include for
dynamic queue implementation

//DEFINICION DE CONSTANTES
#define TIEMPO_BASE 200 // Tiempo base en ms

int main(void) {
    unsigned int tiempo = 0;
    unsigned int cliente = 0;
    int i, fila, n, count = 0, cerrar = 0, tcliente;
    elemento e;

    // Inicializar la función rand
    srand(time(NULL));

    // Pedir el número de cajeras
    printf("\nEl numero de cajeras: ");
    scanf("%d", &n);
```

```

// Crear n colas y arreglo para los tiempos de atención
cola *cajera = malloc(sizeof(cola) * n);
int *tiempos = malloc(sizeof(int) * n);

// Inicializar ambas colas
for (int j = 0; j < n; j++) {
    Initialize(&cajera[j]);
}

// Pedir los tiempos de atención para cada cajera
printf("\nIngrese los tiempos de atención para cada cajera:\n");
for (int i = 0; i < n; i++) {
    printf("Cajera %d: ", i);
    scanf("%d", &tiempos[i]);
}

// Pedir el tiempo de recepción de clientes
printf("\nTiempo de recepcion de clientes: ");
scanf("%d", &tcliente);

// Ciclo principal
while (!cerrar) {
    Sleep(TIEMPO_BASE); // Esperar el tiempo base
    tiempo++;           // Incrementar el contador de tiempo

    // Revisar si es tiempo de atender a un cliente en alguna de las
    cajeras
    for (int tmp = 0; tmp < n; tmp++) {
        if (tiempo % tiempos[tmp] == 0) {
            if (!Empty(&cajera[tmp])) {
                e = Dequeue(&cajera[tmp]);
                printf("\n\nAtendi a: %d en caja %d", e.n, tmp);
                count++;
            } else {
                printf("\n\nNo hay alguien por atender en caja %d",
tmp);
            }
        }
    }

    // Si es tiempo de llegada de un nuevo cliente
    if (tiempo % tcliente == 0) {
        cliente++; // Incrementar el número de clientes
        e.n = cliente; // Asignar el número que identifica al
cliente
        fila = rand() % n; // Escoger la fila para formarse
aleatoriamente
        Queue(&cajera[fila], e); // Formarse en la fila seleccionada
        printf("\n\nLlego el cliente: %d a la cola de la caja %d",
e.n, fila);
    }

    // Mostrar los clientes en cada cola
    for (int l = 0; l < n; l++) {
        printf("\n\n%d clientes en cola %d: [", Size(&cajera[l]), l);
        for (i = 1; i <= Size(&cajera[l]); i++) {
            e = Element(&cajera[l], i);

```

```

        printf("%d\t", e.n);
    }
    printf("]");
}

// Si ya se atendieron 100 clientes, verificar si todas las colas
están vacías
if (count >= 100) {
    cerrar = 1; // Suponer que se puede cerrar
    for (int i = 0; i < n; i++) {
        if (!Empty(&cajera[i])) {
            cerrar = 0; // Si alguna cola no está vacía, no se
puede cerrar
            break;
        }
    }
}

// Liberar la memoria reservada
printf("Cerrando el supermercado...");
free(cajera);
free(tiempos);

return 0;
}

```

- procesosprueba.c

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <windows.h>
#include <time.h>
#include "TADColaDin.h"

// DEFINICION DE CONSTANTES
#define TIEMPO_BASE 200 // Tiempo base en ms

int main(void) {
    int tiempo = 0;
    char opcion = 'S';
    elemento prog;
    cola listos, ejecucion, finalizados;
    Initialize(&listos);
    Initialize(&ejecucion);
    Initialize(&finalizados);

    while (opcion == 'S' || opcion == 's') {
        printf("\nIntroducir el nombre del proceso: ");
        fgets(prog.nombre, 200, stdin);
        prog.nombre[strcspn(prog.nombre, "\n")] = '\0'; // Remove
newline character

        printf("\nIntroducir el tiempo del proceso: ");
    }
}

```



```

scanf("%d", &prog.tiempo);

getchar(); // Clear the newline left by scanf
printf("\nIntroducir la actividad del proceso: ");
fgets(prog.actividad, 200, stdin);
prog.actividad[strcspn(prog.actividad, "\n")] = '\0'; // Remove
newline character

printf("\nIntroducir el ID del proceso: ");
fgets(prog.id, 20, stdin);
prog.id[strcspn(prog.id, "\n")] = '\0'; // Remove newline
character

Queue(&listos, prog);

printf("Deseas añadir otro programa? S/N: ");
scanf(" %c", &opcion);
getchar(); // Clear the newline left by scanf
}

printf("\nIniciando procesos de ejecucion...\n");

while (!Empty(&listos)) {
    Queue(&ejecucion, Dequeue(&listos));
    tiempo++;
    printf("\nEjecutando el programa: ");
    puts(Front(&ejecucion).nombre);

    elemento e = Dequeue(&ejecucion);
    e.tiempo--;

    if (e.tiempo > 0) {
        Queue(&listos, e);
    } else {
        Queue(&finalizados, e);
        printf("\nEl programa de id %s ha finalizado en un tiempo de
%d.\n", e.id, tiempo);
    }

    Sleep(TIEMPO_BASE); // Adding a delay for simulation purposes
}

printf("\nTodos los procesos han finalizado.\n");
return 0;
}

```

- banco.c

AUTOR ORIGINAL: Edgardo Adrián Franco Martínez (C) Septiembre 2012

AUTOR MODIFICADOR: EQUIPO DINO y EQUIPO YINYAN

VERSIÓN: 1.4

DESCRIPCIÓN: Ejemplo de atención de colas de clientes, ususaruiso y preferenciales en un banco, la simulación de tiempo se reliza con la función Sleep(Solo funciona en Windows), cada X tiempo llega un cliente y escoje por prioridad una cola y el tiempo

de atención es Y.

OBSERVACIONES: Se puede emplear la libreria TADColaEst.h, TADPilaEstCir.h o TADColaDin.h implementadas en clase. Ambas estructuras elemento, ya sea la de las implementaciones estáticas o dinámica deberán tener un campo int n;

COMPILACIÓN: gcc -o Cajeras Cajeras.c
TADCola/TADCola(Din|Est|EstCirc).o (Si se tiene el objeto de la implementación)

gcc -o Cajeras Cajeras.c
TADCola/TADCola(Din|Est|EstCirc).c (Si se tiene el fuente de la implementación)

EJECUCIÓN: Cajeras.exe (En Windows)

*/

```
//LIBRERAS
#include <stdio.h>
#include <windows.h>
#include <time.h> //Funciona unicamente en Windows para
usar la función Sleep()
//#include "TADCola/TADColaEst.h" //Si se usa la implemtentación
dinámica (TADColaDin.c)
#include "TADColaDin.h" //Si se usa la implemtentación estática
(TADColaEst.c|TADColaEstCirc.c)

//DEFINICION DE CONSTANTES
#define TIEMPO_BASE 10 //Tiempo base en ms
#define MAX_PRE_CLI 5 //Tiempo maximos de preferentes y
clientes antes de un usuario
#define VACIA 0
#define OCUPADAP 1
#define OCUPADAC 2
#define OCUPADAU 3

int main(void)
{
    unsigned int tiempo = 0;
    unsigned int t_caja[10];
    unsigned int preferentes = 0;
    unsigned int cliente = 0;
    unsigned int usuarios = 0;

    int i,n,filas,p,c,u;
    int
TIEMPO_ATENCION,TIEMPO_USUARIOS,TIEMPO_CLIENTE,TIEMPO_PREFERENTES;
    int cont=0;
    elemento e;
    //Crear 3 colas preferentes '0', clientes '1', usuarios '2'
    cola fila[3];

    //Inicializar tres colas
    Initialize(&fila[0]); //preferentes
    Initialize(&fila[1]); //clientes
    Initialize(&fila[2]); //usuarios
```

```

    printf("¿Cuántas cajas tiene abierto el banco? 'Un máximo de
10'\n");
    scanf("%d",&n);
    int caja[n];
    printf("Tiempo de atención en milisegundos de cada cajero 'Solo
múltiplos de 10'\n");
    scanf("%d",&TIEMPO_ATENCION);
    printf("Tiempo de atención en milisegundos de llegada de los
usuarios del banco 'Solo múltiplos de 10'\n");
    scanf("%d",&TIEMPO_USUARIOS);
    printf("Tiempo de atención en milisegundos de llegada de los
clientes del banco 'Solo múltiplos de 10'\n");
    scanf("%d",&TIEMPO_CLIENTE);
    printf("Tiempo de atención en milisegundos de llegada de los
preferentes del banco 'Solo múltiplos de 10'\n");
    scanf("%d",&TIEMPO_PREFERENTES);

    for (i=0;i<n;i++) //RELLENAMOS LAS CAJAS EN VACIAS y sus tiempos de
atención en 0
    {
        caja[i]=VACIA;
        t_caja[i]=0;
    }
    //Ciclo infinito
    while (1)
    {
        Sleep(TIEMPO_BASE);           //Esperar el tiempo base
        tiempo++;                     //Incrementar el contador de
tiempo

        //Si el tiempo es múltiplo del de llegada de los clientes
        if (tiempo % TIEMPO_CLIENTE == 0)
        {
            cliente++;                //Incrementar el número de
clientes
            e.n = cliente;            //Dar el número que
identifica al cliente
            Queue(&fila[1], e);       //Formarse en la fila
seleccionada
            printf("\n\nLlegó el cliente: %d a la cola de la fila
de clientes", e.n);
        }
        if (tiempo % TIEMPO_USUARIOS == 0)
        {
            usuarios++;              //Incrementar el número de
usuarios
            e.n = usuarios;          //Dar el número que
identifica al usuario
            Queue(&fila[2], e);       //Formarse en la fila
seleccionada
            printf("\n\nLlegó el usuario: %d a la cola de la fila
de usuarios", e.n);
        }
        if (tiempo % TIEMPO_PREFERENTES == 0)
        {

```

```

        preferentes++; //Incrementar el
numero de preferentes
        e.n = preferentes; //Dar el numero que
identifica al preferente
        Queue(&fila[0], e); //Formarse en la fila
seleccionada
        printf("\n\nLlego el preferente: %d a la cola de la
fila de preferentes", e.n);
    }

    //Mostrar los clientes en cada cola
    printf("\n\n%d usuarios en la fila de usuarios:
[\",Size(&fila[2]));
    for (i=1;i<=Size(&fila[2]);i++)
    {
        e=Element(&fila[2],i);
        printf("%d\t", e.n);
    }
    printf("]");

    printf("\n\n%d clientes en la fila de clientes:
[\",Size(&fila[1]));
    for (i=1;i<=Size(&fila[1]);i++)
    {
        e=Element(&fila[1],i);
        printf("%d\t", e.n);
    }
    printf("]");

    printf("\n\n%d preferentes en la fila de preferentes:
[\",Size(&fila[0]));
    for (i=1;i<=Size(&fila[0]);i++)
    {
        e=Element(&fila[0],i);
        printf("%d\t", e.n);
    }
    printf("]");
    //PASAMOS A LAS CAJAS
    for(i=0;i<n;i++)
    {
        if (caja[i]==VACIA)
        {
            if ((!Empty(&fila[0])) && (cont<MAX_PRE_CLI)) //si
la fila de preferentes tiene elementos y su contador de atención es
            {
                e=Dequeue(&fila[0]); //La caja atendera al
cliente de la fila de preferentes
                caja[i]=OCUPADAP; //la caja estara ocupada
con ese preferente

                p=e.n;
                printf("\n La caja %d esta ocupada con el
preferencial %d \n",i+1,p);
                cont++;
            }
            else if ((!Empty(&fila[1])) &&
(cont<MAX_PRE_CLI)) // si la fila de clientes tiene elementos

```

```

        {
            e=Dequeue(&fila[1]); //La caja atendera al
cliente de la fila de clientes
            caja[i]=OCUPADAC; //la caja estara ocupada
con ese cliente
            c=e.n;
            printf("\n La caja %d esta ocupada con el
cliente %d \n",i+1,c);
            cont++;
        }
        else if(!Empty(&fila[2])) // si la fila de
usuarios tiene elementos
        {
            e=Dequeue(&fila[2]); //La caja atendera al
cliente de la fila de usuario
            caja[i]=OCUPADAU; //la caja estara ocupada
con ese usuario
            u=e.n;
            printf("\n La caja %d esta ocupada con
usuario %d \n",i+1,u);
            cont=0;
        }
        else
        {
            printf("\n La caja %d esta sin ningun
cliente porque no hay nadie formado\n",i+1);
        }
    }
    else if (caja[i]==OCUPADAU)
    {
        printf("\n\nLa caja %d esta ocupada con el usuario
%d \n",i+1,u);
    }
    else if (caja[i]==OCUPADAC)
    {
        printf("\n\nLa caja %d esta ocupada con el cliente
%d \n",i+1,c);
    }
    else if (caja[i]==OCUPADAP)
    {
        printf("\n\nLa caja %d esta ocupada con el
preferencial %d \n",i+1,p);
    }
}
//Para vaciar la caja despues del tiempo de atencion
for(i=0;i<n;i++)
{
    if (caja[i]==OCUPADAC || caja[i]==OCUPADAP ||
caja[i]==OCUPADAU)
    {
        t_caja[i]++;
        if (t_caja[i] % TIEMPO_ATENCION == 0)
        {
            caja[i]=VACIA;
        }
    }
}
}

```

```

    }
    return 0;
}

```

- Main_graifico.c

```

#include <windows.h>
#include "graficos.h"
#include "estilos.h"

int main()
{
    inicialiar_pantalla();
    EsperarMiliSeg(1000);
    BorrarPantalla();
    dibujar_presentacion(pantalla_de_inicio);
    EsperarMiliSeg(1000);
    BorrarPantalla();
    dibujar_presentacion(pantalla_de_presentacion);

    EsperarMiliSeg(1000);
    BorrarPantalla();

    dibujar_presentacion(pantalla_de_error);
    EsperarMiliSeg(1000);
    BorrarPantalla();
    dibujar_presentacion(pantalla_de_menu);
    EsperarMiliSeg(1000);
    BorrarPantalla();

    dibujar_presentacion(pantalla_de_final);
    return 0;
}

```

- Graficos.c

```

/*
Autor: Edgardo Adrián Franco Martínez

```

Versión 1.1 (02 de Octubre de 2013)

Descripción: Cabecera de la librería para recrear presentaciones más agradables al usuario en el modo consola bajo Windows

Observaciones: Esta implementación de la librería solo es compatible con Windows y el compilador MinGW ya que utiliza la librería "windows.h", la cual no es estándar.

Compilación de la librería: Windows (gcc -c presentacionWin.c)

*/

//LIBRERIAS

#include <windows.h>

#include "graficos.h"

#include "estilos.h"

#include <stdio.h>

//DEFINICIÓN DE CONSTANTES

#define ALTO 24 //Se piensa en un pantalla de 24 filas x 79 columnas

#define ANCHO 79

#define TIEMPO_BASE 1 //Tiempo base para la espera en milisegundos

//DEFINICIÓN DE FUNCIONES

//Función para mover el cursor de escritura de pantalla, simulación de la función gotoxy() que se tenía en borland 3.0 en la librería conio.h

void MoverCursor(int x, int y)

{

HANDLE hStdout = GetStdHandle(STD_OUTPUT_HANDLE);

COORD position = { x, y };

SetConsoleCursorPosition(hStdout, position);

return;

};

//Función para esperar un tiempo en milisegundos, simulación de la función delay() que se tenía en borland 3.0 en la librería conio.h

void EsperarMiliSeg(int t)

{

Sleep(t);

return;

}

//Funci3n para borrar la pantalla de la consola, simulaci3n de la funci3n clrscr() que se tenia en borland 3.0 en la libreria conio.h

```
void BorrarPantalla(void)
```

```
{  
    system("cls");  
    return;  
}
```

```
void dibujar_presentacion(estilos_t estilo)
```

```
{  
    int i;  
    BorrarPantalla();  
    for(i = 0; i < estilo.height; i++)  
    {  
        int j;  
        for(j = 0; j < estilo.width; j++)  
        {  
            MoverCursor(j, i);  
            printf("%c", estilo.fondo_letra[i][j]);  
            EsperarMiliSeg(TIEMPO_BASE);  
        }  
    }  
}
```

```
void inicialiar_pantalla(void)
```

```
{  
    BorrarPantalla();  
    int columna, fila;  
  
    for(columna=1;columna<ANCHO;columna++)  
    {  
        //Recorrer cada fila  
        for(fila=1;fila<ALTO;fila++)  
        {  
            //Mover el cursor, dibujar un * y esperar TIEMPO_BASE  
            milisegundos  
            MoverCursor(columna,fila);  
            printf("*");  
            EsperarMiliSeg(TIEMPO_BASE);  
        };  
    };  
}
```


- Gráficos.h

```
#ifndef GRAFICOS_H
#define GRAFICOS_H

void MoverCursor( int, int );

//Funci3n para esperar un tiempo en milisegundos, simulaci3n de la
funci3n delay() que se tenia en borland 3.0 en la libreria conio.h
void EsperarMiliSeg(int );

//Funci3n para borrar la pantalla de la consola, simulaci3n de la
funci3n clrscr() que se tenia en borland 3.0 en la libreria conio.h
void BorrarPantalla(void);

void dibujar_presentacion(estilos_t);

void inicialiar_pantalla(void);

#endif // GRAFICOS_H
```

- Estilos.c

```
#include "estilos.h"

estilos_t pantalla_de_inicio = {
    .nombre = "pantalla_de_inicio",
    .width = 80,
    .height = 24,
    .fondo_letra = {
        "
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
",
        "
",
        "
",
        "#####"
```


[illegible]

[illegible]

[illegible]

```
};
```

- Estilos.h

```
#ifndef ESTILOS_H  
#define ESTILOS_H
```

```
typedef struct estilos_t  
{  
    char* nombre;  
    int width;  
    int height;  
    char fondo_letra[24][80];  
}estilos_t;
```

```
extern estilos_t pantalla_de_inicio;  
extern estilos_t pantalla_de_presentacion;  
extern estilos_t pantalla_de_final;  
extern estilos_t pantalla_de_menu;  
extern estilos_t pantalla_de_error;
```

```
#endif
```

Bibliografía

- [1] T. H. a. C. E. L. a. R. L. R. a. C. S. Cormen, Introduction to algorithms, MIT Press, 2009.
- [2] GeeksforGeeks, «Queue Data Structure,» 11 Mayo 2024. [En línea]. Available: <https://www.geeksforgeeks.org/queue-data-structure/>. [Último acceso: 05 Junio 2024].
- [3] J. E. H. & J. D. U. Alfred V. Aho, Estructuras de Datos y Algoritmos, México, DF:

PEARSON Addison-Wesley, 1999.