PEL1 ~ Curso 2024/25

Práctica de Pilas y Colas

PRIMER CUATRIMESTRE

Estructura de Datos – María José Domínguez 27/10/2024

Luis Miguel Herrá Alpuente – luis.herra@edu.uah.es – 06612001F

Jairo Moraga Monterroso - jairo.moraga@edu.uah.es – 49226007S

Índice

Parte 1 TADs creados:

A través del enunciado y la información compartida para la realización de la práctica, hemos decidido la creación de los siguientes tipos abstractos de datos.

Box:

Representa un mostrador donde el pasajero será atendido hasta poder acceder al aeropuerto.

Atributos

ID: Identificador único del Box (int)

Ocupado: Estado de ocupación del Box (bool)

Pasajero: Pasajero que está ocupando el box (pasajero).

Pasajero:

Representa a aquellos viajeros que acuden al aeropuerto.

Atributos

ID: Identificador único del pasajero (int)

Hora: hora de entrada del pasajero a la cola (int)

Duración: tiempo que permanecerá atendido en el box (int)

Prioridad: prioridad de atención del pasajero (int)

Prioridad hace referencia a la prioridad del destino al que viaja el pasajero, el nombre del país es indiferente nuestro único interés reside en la prioridad de atención.

Pila Numérica*:

Pila que trata con valores primitivos enteros, su implementación es totalmente idéntica a la pila de pasajeros. Su finalidad es evitar trabajar con estructuras de datos ajenas a las pilas y colas en esta primera parte.

Parte 1 operaciones de TADs I:

Clase Box

Box.cpp

Aquí, se gestiona una cola de pasajeros, permitiendo agregar y eliminar pasajeros, verificar el estado de la cola y mostrar su contenido.

Para comenzar el constructor de la clase Box que inicializa los miembros de la clase: **ID, ocupado y pasajero** usando los valores proporcionados como parámetros.

A continuación, contamos con diferentes métodos para cada miembro de la clase:

- Para ID
    - El método **Get** devuelve el valor del ID del Box. El método es const, lo que significa que no modifica el estado del objeto.
    - El método **Set** establece un nuevo ID para el Box. Recibe un entero como parámetro.
- Para Ocupado
    - El método **isOcupado**() que devuelve el estado booleano de si el Box está ocupado o no. También es const, ya que no modifica el objeto.
    - El método **Set** establece el estado de ocupado del Box
- Para Pasajero
    - Método **Get** que devuelve una copa del objeto Pasajero asociado al Box.
    - Método **Set** que establece un nuevo objeto Pasajero para el Box.

Box.h

Para empezar, se incluye mediante #include "Pasajero.h" la definición de la clase Pasajero, que es necesaria para declarar un miembro pasajero dentro del Box.

A continuación, se declaran los atributos privados de la clase: **ID, ocupado y pasajero**.

Cuenta con los siguientes métodos públicos:

- **Constructor** de la clase que inicializa los atributos ID, ocupado y pasajero con valores por defecto.
- Métodos para obtener y establecer el valor de ID (**get y set**). El método getID() es const, lo que significa que no modifica el estado del objeto.
- Métodos para obtener y establecer el valor de Ocupado (**get y set**).
- Métodos para obtener y establecer el valor de Pasajero (**get y set**).

Parte 1 operaciones de TADs II:

Cola.cpp

Esta clase establece la estructura y los métodos para gestionar una cola de Pasajero utilizando nodos enlazados. Permite agregar, eliminar y acceder a pasajeros, así como verificar el estado y la longitud de la cola.

Encontramos el **Constructor** de Cola que inicializa una cola vacía, con primero y ultimo apuntando a nullptr y longitud establecida en 0. Por otro lado tenemos el Destructor que libera la memoria utilizada por la cola, desencolando todos los elementos hasta que la cola esté vacía.

Método Encolar: Agrega un nuevo pasajero a la cola. Si la cola está vacía, el nuevo nodo se convierte en el primero y último. De lo contrario, se agrega al final de la cola y se actualiza último.

Método Desencolar: Elimina y devuelve el primer pasajero de la cola. Si la cola está vacía, lanza una excepción.

Método Inicio: Devuelve el primer pasajero sin desencolarlo. Si la cola está vacía, lanza una excepción.

Método Fin: Devuelve el último pasajero sin desencolarlo. Si la cola está vacía, lanza una excepción.

Método Es_Vacía: Comprueba si la cola está vacía.

Método mostrarCola: Muestra todos los pasajeros en la cola.

Método getLongitud: Devuelve la longitud actual de la Cola

Cola.h

Para comenzar, se incluyen los encabezados para las clases NodoCola y Pasajero, que son necesarios para declarar la clase Cola.

A continuación, se produce la Declaración de la Clase definiendo a su ves los atributos privados de la clase:

- **primero**: Puntero al primer nodo de la cola.
- **ultimo**: Puntero al último nodo de la cola.
- **longitud**: Almacena la longitud actual de la cola.

En cuanto a los métodos, se declaran los mismos métodos que se implementan en el archivo .cpp

Parte 1 operaciones de TADs III:

NodoCola.cpp

La clase NodoCola actúa como un contenedor para un Pasajero y un puntero al siguiente NodoCola en la cola. Esto permite la creación de una estructura de datos de cola enlazada, donde cada nodo apunta al siguiente.

Incluye los encabezados necesarios para la declaración de la clase NodoCola y la clase Pasajero.

El **constructor** de la clase NodoCola inicializa un nodo con dos parámetros:

- **e**: Un objeto de tipo Pasajero.
- **sig**: Un puntero al siguiente nodo en la cola.

Este constructor utiliza una lista de inicialización para establecer los valores de elemento (el pasajero que contiene el nodo) y siguiente (el siguiente nodo en la cola).

También cuenta con un **Destructor** para liberar recursos cuando el nodo se destruye.

NodoCola.h

Este archivo declara la estructura básica de un nodo en una cola enlazada, con un pasajero y un puntero al siguiente nodo

Primero se define la clase NodoCola, que es un nodo de una estructura de cola, diseñada para contener objetos Pasajero.

Incluye la definición de la clase Pasajero, necesaria para declarar un miembro Pasajero dentro de NodoCola.

Declaración de la Clase NodoCola: define la clase y declara a Cola como amiga. Esto permite que la clase Cola acceda a los miembros privados de NodoCola, facilitando la gestión de la estructura de la cola.

Se declaran los métodos públicos de Constructor y Destructor de la clase .cpp

Parte 1 operaciones de TADs IV:

Pila.cpp

La clase Pila implementa una pila de pasajeros, proporcionando métodos para apilar, desapilar, verificar si está vacía, mostrar su contenido y obtener el pasajero en la cima sin eliminarlo. Sus métodos son los siguientes:

- **Constructor**: Inicializa una pila vacía con cima apuntando a NULL.
- **Destructor**: Libera la memoria utilizada por la pila, desapilando todos los elementos hasta que la pila esté vacía.
- **Método esVacia**: Comprueba si la pila está vacía.
- **Método apilar**: Agrega un nuevo pasajero en la cima de la pila. Crea un nuevo nodo y lo establece como la nueva cima.
- **Método desapilar**: Elimina el pasajero en la cima de la pila. Si la pila no está vacía, elimina el nodo en la cima y actualiza la cima al siguiente nodo.
- **Método mostrarPila**: Muestra todos los pasajeros en la pila, útil para depuración. Recorre la pila desde la cima hasta el fondo.
- **Método getCima**: Devuelve el pasajero en la cima de la pila sin eliminarlo. Si la pila está vacía, lanza una excepción.

Pila.h

En este archivo, se proporcionan los métodos necesarios para gestionar una pila de pasajeros de manera eficiente y organizada.

Primero, se incluye los encabezados necesarios para la definición de las clases NodoPila y Pasajero, que son componentes esenciales de la clase Pila.

A continuación, se produce la declaración de la clase Pila con un miembro privado:

- **cima**: Un puntero al nodo en la cima de la pila. pnodo es probablemente un tipo definido para NodoPila.

Por último, se declaran los métodos públicos explicados en el archivo pila.cpp

Parte 1 operaciones de TADs V:

NodoPila.cpp

Este archivo implementa la lógica para inicializar nodos en una pila, gestionando tanto la creación con valores por defecto como con valores específicos, y proporcionando un destructor para limpieza.

Primero encontramos un **Constructor** por Defecto que inicializa un nodo con valores por defecto:

- **valor(Pasajero()):** Inicializa el valor del nodo con un objeto Pasajero por defecto.
- **siguiente(NULL)**: Establece el puntero al siguiente nodo como NULL, indicando que no hay nodo siguiente.

Después, tenemos un **constructor** que acepta parámetros inicializando un nodo con valores proporcionados:

- **valor(v):** Inicializa el valor del nodo con el objeto Pasajero proporcionado.
- **siguiente(sig):** Establece el puntero al siguiente nodo con el puntero proporcionado.

Por último, tenemos el **Destructor** para liberar recursos cuando el nodo se destruye.

NodoPila.h

Este archivo ayuda a gestionar la pila de objetos Pasajero proporcionando los constructores y el destructor necesarios para la inicialización y limpieza, y definiendo cómo los nodos están enlazados en la estructura de la pila.

Primero se incluyen las bibliotecas necesarias. Pasajero.h es necesario para declarar el miembro Pasajero.

A continuación, se produce la Declaración de la Clase con los siguientes atributos privados creados:

- **valor**: Un objeto Pasajero almacenado en el nodo.
- **siguiente**: Un puntero al siguiente nodo en la pila.
- **friend class** Pila: Permite que la clase Pila acceda a los miembros privados de NodoPila.

Por último, se declaran los métodos públicos explicados en el archivo NodoPila.cpp

Parte 1 operaciones de TADs VI:

Pasajero.cpp

En primer lugar, tenemos la declaración del Constructor con Parámetros de la clase. Inicializa un objeto Pasajero con los parámetros:

- **id**: Identificador único del pasajero.
- **hora**: Hora de llegada del pasajero.
- **duracion**: Duración estimada de la estancia del pasajero.
- **prioridad**: Nivel de prioridad del pasajero.

En segundo lugar, tenemos los siguientes métodos:

- **Getters**: Estos métodos (getId, getHora, getDuracion, getPrioridad) devuelven copias de los datos miembro del Pasajero. Están marcados como const, lo que significa que no modifican el estado del objeto.
- **Setters**: Estos métodos (setId, setHora, setDuracion, setPrioridad) establecen nuevos valores para los datos miembro del Pasajero. Utilizan la palabra clave this para referirse al objeto actual y actualizar sus miembros.

Pasajero.h

Para comenzar tenemos la declaración de la clase con los mismos atributos privados que encontramos en el constructor del archivo Pasajeros.cpp que son: **id, hora, duración y prioridad.**

Después, se declaran los Métodos Públicos que son: el **Constructor**, y los **Getters** y los **Setters** explicados en el apartado anterior.

Parte 1 operaciones de TADs VII:

Utilidades.cpp

En este archivo se encuentran algunas funciones necesarias para el corrector funcionamiento del programa. Además, se incluyen las clases y encabezados necesarios para la implementación de las funciones. A continuación, se enumeran las funciones:

- **Función desencolarMaxPrioridad**: Esta función desencola el pasajero con la prioridad más alta de colaPasajeros:

    1. Desencola todos los pasajeros en una cola temporal.
    2. Encuentra el pasajero con la máxima prioridad.
    3. Reconstruye la cola original sin el pasajero de máxima prioridad.
    4. Devuelve el pasajero de máxima prioridad.

- **Función ordenarColaPorPrioridad**: Esta función ordena colaPasajeros por prioridad utilizando desencolarMaxPrioridad. Desencola el pasajero con mayor prioridad y lo encola en colaOrdenada. Reconstruye colaPasajeros con los pasajeros ordenados.
- **Función boxesLibres**: Comprueba si todos los Box están libres.
- **Función simular_min**: Simula el paso de un minuto, gestionando el movimiento de pasajeros entre la pila, la cola y los boxes.

    1. Incrementa el tiempo.
    2. Mueve pasajeros de la pila a la cola si su hora coincide con el tiempo actual.
    3. Procesa cada box, decrementando el tiempo de proceso de los pasajeros y moviéndolos al aeropuerto si han terminado.

- **Función calcularMediaPila**: Calcula la media de los valores en una pila, utilizando una pila temporal para no modificar la pila original.
- **Función simular_hasta_final**: Simula el proceso completo hasta que no queden pasajeros en la pila, cola, ni boxes. Utiliza simular_min en un bucle hasta que todas las estructuras estén vacías.

Utilidades.h

Incluye los encabezados necesarios para definir las clases Cola, Pila y Box, que se utilizan en las funciones declaradas en Utilidades.h. Después, declara las funciones explicadas en el archivo utilidades.cpp

Parte 2 operaciones de TADs I:

Lista.cpp

Constructor de la Lista
El constructor de Lista inicializa un objeto Lista con los siguientes atributos:

Destructor de la Lista
Libera la memoria de todos los nodos de la lista mediante un ciclo que recorre la lista, eliminando cada nodo uno por uno.

Método insertar(Box& b)
Inserta un nuevo Box al final de la lista. Si la lista está vacía, el nuevo Box se convierte en el primer elemento. Si no, se recorre hasta el final de la lista y se enlaza el nuevo nodo al último elemento.

Método eliminarVacios()
Elimina los Box vacíos si existen al menos dos Box vacíos en la lista. Primero, cuenta cuántos Box están vacíos; si hay dos o más, los elimina hasta que queden menos de dos Box vacíos en la lista.

Método obtenerMenorCola()
Retorna el Box con la cola más corta. Si la lista está vacía, lanza una excepción. Este método recorre todos los nodos y compara la longitud de la cola de cada Box para encontrar la mínima.

Método todosMayorDos()
Verifica si todos los Box tienen más de dos pasajeros en la cola. Si algún Box tiene dos o menos pasajeros, retorna false; en caso contrario, retorna true.

Método boxesVacios()
Verifica si todos los Box están vacíos. Retorna true si todos están vacíos y false si alguno tiene pasajeros.

Método estaVacia()
Retorna true si la lista está vacía (primero es nullptr) y false en caso contrario.

Método mostrarLista()
Muestra en consola información de todos los Box en la lista. Para cada Box, muestra su ID, si está ocupado, y el largo de la cola.

Método getLongitud()
Retorna el número total de nodos en la lista (longitud).

Método getPrimero()
Retorna el puntero al primer nodo de la lista (primero). Este método está marcado como const, lo que garantiza que no modifica el estado del objeto Lista.

Método obtenerBoxEnPosicion(int indice)
Retorna un puntero al Box en la posición especificada por indice. Si el índice está fuera de rango, lanza una excepción. Recorre la lista hasta llegar al nodo deseado.

Parte 2 operaciones de TADs II:

Lista.h

Define la clase Lista con los atributos privados primero y longitud, y declara los métodos públicos que se implementan en Lista.cpp, incluyendo el constructor, destructor y los métodos descritos anteriormente.

NodoLista.cpp

Este archivo implementa la lógica para inicializar nodos en una lista enlazada, gestionando tanto la creación con valores específicos como la referencia al siguiente nodo en la lista, y proporcionando un destructor para la limpieza.

Primero, encontramos el **Constructor con Parámetros** que inicializa un nodo con valores proporcionados:

- **box(b)**: Inicializa el Box contenido en el nodo con el objeto Box proporcionado.
- **siguiente(sig)**: Establece el puntero al siguiente nodo como sig, que puede apuntar a otro nodo o ser nullptr, indicando el final de la lista.

Después, tenemos el **Destructor**, que libera recursos cuando el nodo se destruye. Este destructor no realiza ninguna operación adicional, ya que la eliminación del nodo no requiere manejo especial de memoria.

Finalmente, tenemos los **Métodos Getters**:

- **getSiguiente()**: Este método retorna el puntero siguiente, permitiendo acceder al nodo que sigue en la lista. Está marcado como const para garantizar que no se modifique el estado del nodo.
- **getBox()**: Este método retorna una referencia al Box almacenado en el nodo, permitiendo el acceso y modificación directa del objeto Box contenido.

NodoLista.h

Define la clase NodoLista con los atributos privados box y siguiente, y declara los métodos públicos que se implementan en NodoLista.cpp, incluyendo el constructor, destructor y los métodos getSiguiente y getBox descritos anteriormente.

Parte 2 operaciones de TADs III:

Utilidades.cpp

Este archivo contiene varias funciones necesarias para el correcto funcionamiento del programa. A continuación, se describen las funciones implementadas:

- **Función desencolarMaxPrioridad**: Esta función se encarga de desencolar el pasajero con la máxima prioridad de colaPasajeros:
    1. Desencola todos los pasajeros en una cola temporal.
    2. Encuentra el pasajero con la prioridad más alta.
    3. Reconstruye la cola original sin el pasajero de máxima prioridad.
    4. Devuelve el pasajero de máxima prioridad.
- **Función ordenarColaPorPrioridad**: Esta función ordena colaPasajeros por prioridad utilizando desencolarMaxPrioridad. Desencola el pasajero con mayor prioridad y lo encola en colaOrdenada, luego reconstruye colaPasajeros con los pasajeros ordenados.
- **Función boxesLibres**: Comprueba si todos los Box están libres.
- **Función simular_min**: Simula el paso de un minuto, gestionando el movimiento de pasajeros entre la pila, la cola y los boxes:
    1. Verifica si la simulación debe terminar comprobando si la pila y los boxes están vacíos.
    2. Incrementa el tiempo.
    3. Mueve pasajeros de la pila a la cola si su hora coincide con el tiempo actual.
    4. Procesa cada box, decrementando el tiempo de proceso de los pasajeros y moviéndolos al aeropuerto si han terminado.
    5. Verifica el número de boxes ocupados y, si hay al menos dos boxes libres, elimina los boxes vacíos.
    6. En caso de haber dos en cada cola crea un nuevo box con el ID
- **Función calcularMediaPila**: Calcula la media de los valores en una pila, utilizando una pila temporal para no modificar la pila original.
- **Función simular_hasta_final**: Simula el proceso completo hasta que no queden pasajeros en la pila, cola, ni boxes. Utiliza simular_min en un bucle hasta que todas las estructuras estén vacías.

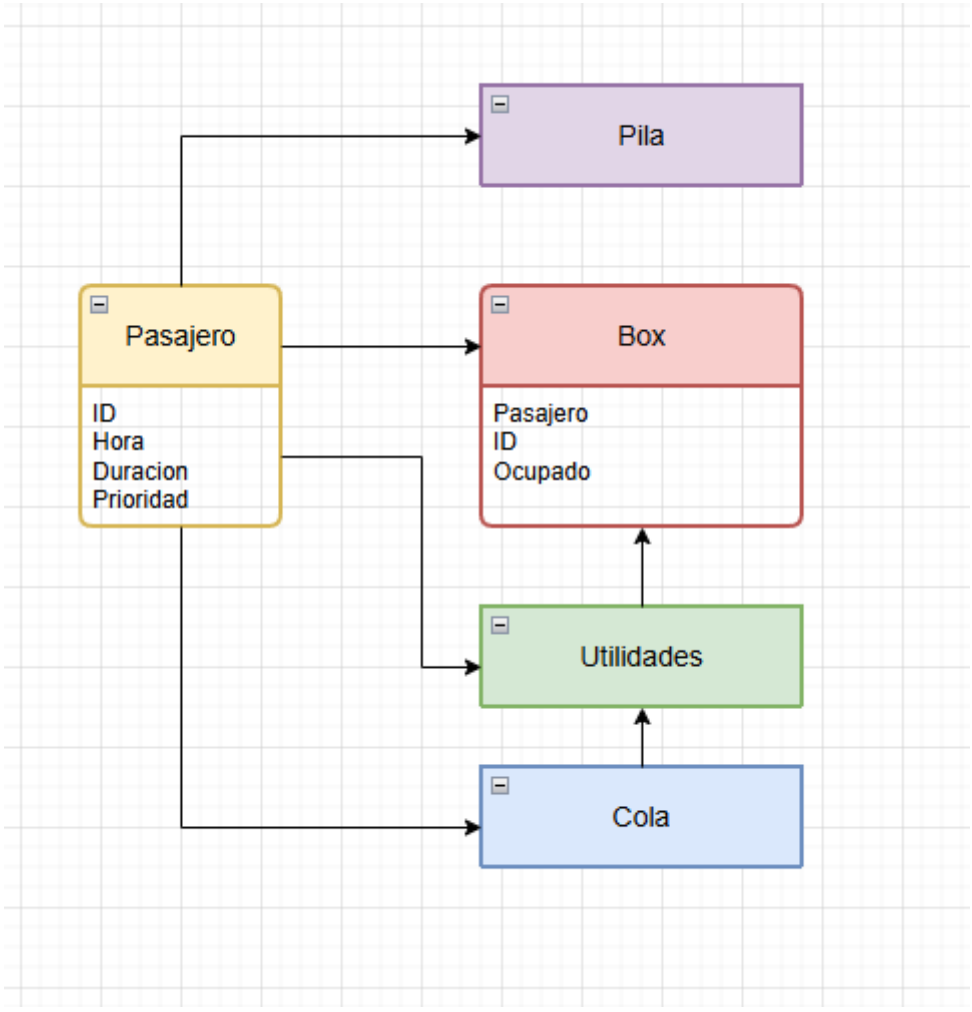
Utilidades.h

Este archivo incluye los encabezados necesarios para definir las clases Cola, Pila y Box, que se utilizan en las funciones declaradas a continuación. También declara las funciones descritas en Utilidades.cpp.

Dificultades encontradas:

El desarrollo del trabajo ha sido marcado por un mayor tiempo de duración del proceso de análisis; esto ha favorecido un mejor entendimiento de los requisitos del sistema. Sin embargo, hemos experimentado problemas con el paso por referencia y valor. Problemas solucionados gracias al contenido mostrado en la bibliografía.

Modelo Relacional de TADs 1:



Este diseño busca centralizar las operaciones en la clase utilidades. Aunque esta forma de trabajo pueda aumentar el acoplamiento, creemos más legible y cómodo para el desarrollo la centralización de funciones en una clase.

Explicación del comportamiento del programa P1:

El comportamiento del programa sigue los criterios establecidos en la práctica con la implementación de un menú con las siguientes funcionalidades.

La función principal (main) **inicializa** las estructuras necesarias para el simulador:

Cola y Pila de Pasajeros: La cola colaPasajeros y la pila pilaPasajeros almacenan y manejan los pasajeros en espera.

PilaEnteros aeropuerto: Se usa como un registro de tiempo por pasajero.

Boxes b1, b2 y b3: Representan los espacios de atención que indican si están ocupados o no.

El bucle while ejecuta el programa hasta que el usuario seleccione la opción de salir (opción 8).

Caso 1: Crear una pila de pasajeros.

Se crean objetos de tipo Pasajero con ciertos parámetros (probablemente identificador, prioridad, tiempo de atención, y nivel) y se apilan en pilaPasajeros. Esto simula la llegada de nuevos pasajeros al aeropuerto.

Caso 2: Mostrar la pila de pasajeros.

Llama a pilaPasajeros.mostrarPila(), lo que debería mostrar los datos de cada pasajero en la pila, permitiendo visualizar quién está en espera.

Caso 3: Borrar la pila de pasajeros.

Utiliza un bucle while para desapilar (desapilar()) cada elemento hasta que la pila esté vacía. Esto limpia la pila por completo.

Caso 4: Mostrar la cola de espera.

Llama a colaPasajeros.mostrarCola(), que debería mostrar los pasajeros en la cola de espera.

Caso 5: Mostrar el estado de los boxes.

Imprime el estado (ocupado o no) de cada uno de los tres boxes b1, b2, y b3 usando su método isOcupado(), que probablemente devuelve un booleano indicando si el box está ocupado.

Caso 6: Simular el paso de N minutos.

Permite al usuario ingresar un número N de minutos a simular. En cada minuto, llama a simular_min, una función que realiza las operaciones de simulación durante un minuto. Esta función toma los objetos colaPasajeros, pilaPasajeros, el contador tiempo, aeropuerto, y los boxes b1, b2, y b3 como parámetros, y probablemente gestiona el movimiento de pasajeros y la ocupación de los boxes.

Caso 7: Simular hasta que todos sean atendidos.

Ejecuta la función simular_hasta_final, que simula el proceso hasta que todos los pasajeros hayan sido atendidos. Al final, calcula y muestra el tiempo medio de atención por pasajero llamando a calcularMediaPila(aeropuerto).

Explicación del comportamiento del programa P2

El programa simula la gestión de pasajeros en un aeropuerto, utilizando un menú interactivo con las siguientes opciones:

1. **Crear Pila de Pasajeros**: Se generan y apilan pasajeros con atributos como ID, prioridad, tiempo de atención y nivel.
2. **Mostrar Pila de Pasajeros**: Muestra todos los pasajeros apilados.
3. **Borrar Pila de Pasajeros**: Desapila todos los pasajeros hasta vaciar la pila.
4. **Mostrar Cola de Espera**: Permite seleccionar un box específico y muestra la cola de espera asociada.
5. **Mostrar Estado de Boxes**: Imprime el estado (ocupado/no ocupado) de todos los boxes disponibles.
6. **Simular N Minutos**: Simula el paso de un número especificado de minutos, ejecutando la función de simulación correspondiente.
7. **Simular Hasta que Todos Sean Atendidos**: Realiza una simulación completa hasta atender a todos los pasajeros.
8. **Mostrar Box Menos Ocupado**: Identifica y muestra el box con menos ocupación.
9. **Contar Boxes Operativos**: Cuenta y muestra cuántos boxes están operativos.
10. **Salir**: Termina la ejecución del programa.

El bucle principal se repite hasta que el usuario elige la opción de salir, permitiendo una gestión dinámica de los pasajeros y los recursos del aeropuerto.

Bibliografía

Tutoriales útiles seguidos

https://www.youtube.com/watch?v=y3K3jb3wv2I

https://www.youtube.com/watch?v=_pcfFMFs9-g

https://www.youtube.com/watch?v=bgfH_HB341M

https://www.youtube.com/watch?v=slzcWKWCMBg

Páginas de aprendizaje online

https://www.w3schools.com/cpp/cpp_pointers.asp

Documentos PDF subidos

Especificación de Listas, Listas 2, Lista extendida

Especificación de Pila, Cola