**PL1 2023/24 Estructuras de datos**

**Jorge Barroso – 50359997W**

**Luciana Paola Diaz – Y6049376E**

**TAD’S CREADOS**

* Pila [Mesa]
* Cola [Reserva]
* Lista [Pedido]

Siendo Mesa, Reserva y Pedido clases del programa.

**ESPECIFICACIÓN DE LOS TAD’S**

**Especificación del TAD Pila [Mesa]:**

espec PILA[MESA]  
usa BOOLEANOS, RESERVA, NATURALES  
parámetro formal  
 géneros mesa  
fparámetro  
géneros pila  
**operaciones**  
 Pila: -> pila {constructor de pila, equivalente a pilaVacia} (Generadora)  
 apilar: pila mesa -> pila (Generadora)  
 apilarEnOrden: pila mesa -> pila (Modificadora)   
 Parcial desapilar: pila -> pila (Modificadora)  
 vaciarPila: pila -> pila (Modificadora)  
 Parcial getMesaCima: pila-> mesa (Observadora)  
 esVacia: pila -> boolean (Observadora)  
 mostrarPilaMesas: pila  
 buscarMesas: reserva natural -> puntero a array de ^mesa   
**var** p: pila  
**ecuaciones de definitud**  
 esVacia(p) = F -> Def(desapilar(p))   
 esVacia(p) = F -> Def(getMesaCima(p))   
fespec

**Especificación del TAD Cola[Reserva]:**  
 Parcial desencolar: cola -> cola (Modificadora)  
 Parcial inicio: cola -> reserva (Observadora)  
 Parcial fin: cola -> reserva (Observadora)  
 esVacia: cola -> boolean (Observadora)  
 Parcial mostrarCola: cola  
**var** cola: c  
**ecuaciones de definitud**  
esVacia(c) = F -> Def(desencolar(c))  
esVacia(c) = F -> Def(inicio(c))  
esVacia(c) = F -> Def(fin(c))  
esVacia(c) = F -> Def(mostrarCola(c))

**Especificación del TAD Lista[Pedido] (Lista doblemente enlazada):**

**espec LISTA[PEDIDO]**  
**usa BOOLEANOS, PILAS**  
**parámetros formales**  
 **genéros pedido**  
**fparámetro**  
**genéros lista**  
****operaciones****  
 **Lista: -> lista**  **(Generadora)**  
 extenderListaPorIzquierda: lista pedido -> lista (Generadora)  
 extenderListaPorDerecha: lista pedido -> lista (Modificadora)  
 Parcial eliminarPrimero: lista -> lista (Modificadora)  
 Parcial eliminarUltimo: lista -> lista (Modificadora)  
 Parcial completarSiguientesPedidos: pila lista -> lista (Modificadora)  
 esVacia: lista -> boolean (Observadora)  
 Parcial mostrarDatosLista: lista   
 Parcial getPrimero: lista -> NodoLista\* (Observadora)

var l:Lista, p:Pedido , pila:Pila  
**ecuaciones de definitud**  
esVacia(l)=F -> Def(extenderListaPorDerecha(l,p))  
esVacia(l)=F -> Def(eliminarPrimero (l))  
esVacia(l)=F -> Def(eliminarUltimo (l))  
esVacia(l)=F -> Def(completarSiguientesPedidos (pila,l))  
esVacia(l)=F -> Def(mostrarDatosLista (l))  
esVacia(l)=F -> Def(getPrimero (l))   
fespec

**DESCRIPCIÓN DE LAS DIFICULTADES ENCONTRADAS Y SOLUCIÓN ADOPTADA**

A lo largo de la práctica hemos tenido una serie de dificultades con métodos, tipos de datos y las especificaciones de los TAD’s con métodos añadidos:

1. Según los métodos, tuvimos dificultades con la función buscar mesa, porque requería un manejo más complejo de los punteros y de la lógica detrás de la búsqueda de una o dos mesas (según la reserva). Al principio, la función retornaba un booleano si hallaba las mesas necesarias y creaba el pedido correspondiente a la reserva y lo agregaba a la lista de pedidos dentro de la misma función. El problema de esta función radicaba en muchas estructuras lógicas como whiles, ifs y punteros, además de la creación del pedido dentro de la función y no fuera en la clase Gestor que era desde donde se invocaba. Se resolvió con una función más simplificada con un arreglo de combinaciones óptimas basadas en el número de personas, recorriendo la pila de mesas dos veces (en caso de que no se encontrara la primera combinación) y añadiendo las mesas en un array de punteros de mesa, retornando este mismo array. Se presentó algún problema con respecto a sacar la mesa del stack de mesas en memoria dinámica por un inconveniente de eliminar los nodos del stack pero se solucionó.  
     
   Con respecto a procesar las últimas reservas de colas pendientes el problema se enfocaba en no saber cuándo la cola de reservas pendientes había sido recorrida después de gestionar todas las reservas. Por lo que la solución adoptada fue crear una nueva cola de reservas pendientes llamada cola de reservas no gestionadas. Una vez se han simulado todas las reservas y se haya recorrido las reservas de cola pendientes, las reservas que aún estén en cola pendientes pasarán a la cola de reservas no gestionadas, siendo estas las reservas que no se logró encontrar una mesa en ningún momento.
2. En cuanto al problema con los Arrays, era más un problema de sintaxis y de compresión al entender como una función devolviese un array, como se ha mencionado anteriormente, la función mesa devolvía un array de dos puestos con punteros hacia las mesas correspondientes. Por lo que para devolver este array había que cambiar el tipo que devolvía la función y su sintaxis. La solución fue solo entender como funcionaban y como se representaban en el lenguaje utilizado.
3. En cuanto a la especificación de los TAD’s con nuevos métodos, el problema se enfocaba en que algunas funciones no devolvían nada, ya que imprimían por pantalla los datos de la reserva, mesa o pedido. La solución ha sido que estas funciones se toman como funciones auxiliares, porque no devuelven nada ni modifican el TAD.

**DISEÑO DE LA RELACIÓN ENTRE LAS CLASES DE LOS TAD IMPLEMENTADOS**

En nuestro programa hemos implementado las siguientes clases:

* Pila
* NodoPila
* Cola
* NodoCola
* Lista (Doblemente enlazada)
* NodoLista
* Pedido
* Reserva
* Mesa
* Gestor
* main

Para empezar, las tres colecciones implementadas (Pila, Cola y Lista) están formadas por nodos (NodoPila, NodoCola y NodoLista respectivamente) que guardan un valor puntero a otra clase (Mesa, Reserva y Pedido respectivamente) y un puntero al siguiente nodo llamado “siguiente”. Para entenderlo mejor, estos son los atributos de NodoPila:  
 Mesa\* pmesa;  
 NodoPila \*siguiente;  
 friend class Pila;   
Notese que usa friend class para permitir usar estos atributos privados a una clase especificada, en este caso Pila. Cola y Lista realizan lo mismo, como se puede ver en el código.

Además, la clase NodoLista como la lista es doblemente enlazada guarda también un puntero NodoLista al nodo anterior.

De esta forma cada clase colección maneja un tipo distinto de datos y está relacionada con sus mismas clases nodos siendo gráficamente sus relaciones:  
 Pila->NodoPila->Mesa  
 Cola->NodoCola->Reserva  
 Lista->NodoLista->Pedido

Finalmente, en la clase main se crean todas las instancias necesarias de las colecciones (3 colas, una pila y una lista) y no se crean nuevas ni eliminan estas a lo largo del programa, pasándolas siempre como referencia.

**EXPLICACIÓN DEL COMPORTAMIENTO DEL PROGRAMA Y DE LOS MÉTODOS MÁS DESTACADOS**

Antes de proceder a la explicación, debe saberse que todo el código del programa está documentado debidamente para una mayor claridad y legibilidad del programa, por lo que si esta breve explicación no fuese suficiente para entender el programa, se puede leer directamente el código fuente documentado.

La ejecución del programa comienza en el archivo main.cpp, el cual, antes de nada, crea las siguientes colecciones que se usarán a lo largo del programa: Cola colaReservas, Cola colaReservasPdtes, Cola colaReservasNoGestionadas, Lista listaPedidos y Pila pilaMesas, y luego llama a la función int main().

Dentro del main() se entra en un buble en el cual se mostrará el menú principal al usuario con las posibles opciones a realizar, pedirá la elección de una al usuario, la validará y llevará a cabo, y se reiniciará el bucle a menos que se escoja la opción 0.Salir, en cuyo caso el bucle termina y finaliza el programa.

Dependiendo de la opción que se lleve a cabo se llamará a un método estático de la clase Gestor, que llevará a cabo dicha opción. Para cada opción del menú hay un método correspondiente en la clase Gestor. Además, en el caso de elegir la opción 1. Generar la cola de reservas, se pedirá al usuario el número de reservas a generar entre 12 y 50 y se validará antes de pasárselo al método estático de Gestor para generar las reservas. Para las opciones 7,8 y 9 se llamará al método correspondiente de Gestor y después de este se llamará al método mostrarDatos de Gestor que mostrará todos los datos correspondientes tras las simulaciones.

Ahora explicaré brevemente el funcionamiento de las principales funciones (todas estáticas) de Gestor:

**Las 3 primeras opciones** del menú principal corresponden a las funciones para generar, mostrar y vaciar la cola de reservas. La primera, genera la cola de reservas con el número de reservas dado (obtenido del usuario en el menú principal), con la restricción de que habrá al menos 4 para cada periodo horario: 13:30,14:30,15:30, y si la cola ya está llena, antes de generar nuevas reservas, la vacía. La segunda llama al método auxiliar mostrarCola dentro de Cola que usa memoria dinámica para recorrer los nodos de la Cola con un puntero NodoCola y mostrar sus datos. La tercera llama al método auxiliar vaciarCola dentro de Cola que usa memoria dinámica para vaciar la cola.

Análogamente a las 3 primeras, las opciones 4-6 generan la pila de mesas, la muestran y la vacían de la misma forma que las opciones para la cola, pero con la restricción en la generación de pilas de que debe haber al menos 8 mesas con situación de la mesa en Terraza.

La opciónes 7,8 y 9 son relativas a la simulación de la gestión de las reservas y no se llevan a cabo si la cola de reservas está vacía. Todas ellas se apoyan sobre una función auxiliar void Gestor:: procesarReserva que se encarga de procesar una reserva dado un puntero apuntándola, buscando mesas para la reserva y creando el pedido correspondiente si las hay. Si no las hay, si es una reserva común se añade a reservas pendientes y si se trata de la última pasada de la simulación por la cola de reservas pendientes, se añade a la cola de reservas no gestionadas, correspondiente a las reservas que no se han conseguido gestionar a lo largo de la simulación.

La función correspondiente a la **opción 7**, void Gestor:: simularGestionProximaReserva, se encarga de simular la gestión de la próxima reserva, primero simulando las reservas de cola reservas y si no quedan se empiezan a simular las de la cola de reservas pendientes hasta que no quede ninguna. Para simular la gestión, primero se procesa la reserva correspondiente (lo cual aumenta un contador estático de reservas gestionadas) y cada 2 reservas gestionadas de cola reservas se intenta gestionar una de reservas pendientes. Finalmente comprueba si esa reserva era la última a esa hora, y en caso afirmativo, simula el cambio de hora, llamando a void Gestor:: simularCambioHora, que marca los 4 primeros pedidos de la lista pedidos como finalizados y libera sus mesas asignadas añadiendolas de nuevo a la pila de mesas.

La función correspondiente a la **opción 8**, void Gestor::simularGestionReservasProximaHora, se encarga de simular la gestión de la próxima reserva de cola reservas si la hay, y todas las demás de cola reservas que sean a la misma hora. Para ello procesa la siguiente reserva correspondiente en un buble hasta que haya cambio de hora (comprobado mediante la función auxiliar bool comprobarCambioHora). Y al terminar simula el cambio de hora. Además, dentro del bucle se comprueba cada 2 reservas de cola reservas si se puede gestionar una de la cola de reservas pendientes.

La función correspondiente a la **opción 9**, void Gestor::simularGestionReservasTotal, se encarga de gestionar todas las reservas de la simulación, primero las de cola reservas y luego las de cola reservas pendientes. Para conseguirlo es tan sencillo como llamar a la función de la opción 7, simularGestionProximaReserva, en bucle mientras queden reservas en la cola de reservas normales o pendientes, ya que está ya se encarga de toda la funcionalidad de gestionar la próxima reserva individual.

Finalmente, la función correspondiente a la opción 0, void Gestor::salir(), solo muestra por pantalla un mensaje de despedida, y el programa finaliza después ya que tras llamar a esta función se detiene el bucle del menú principal en el int main() de main.cpp.

**BIBLIOGRAFÍA**

* Transparencias de clase: Profesora: Mª José Domínguez Alda, Departamento de Ciencias de la computación, Universidad de Alcalá, Alcalá de Henares, España; **uah.blackboard.com/2023-24: ESTRUCTURAS DE DATOS (2A\_G781) / Teoría/**
  + **TEMA 1-INTRODUCCIÓN TIPOS ABSTRACTOS DE DATOS.**
  + **TEMA 2-TAD PILA** y **TEMA 3-TAD COLAS**
  + **TEMA 4-TAD LISTAS** y **TEMA 4B-TAD LISTA+**
* **Youtube/Neso Academy/ Introduction to Doubly Linked List:** <https://www.youtube.com/watch?v=e9NG_a6Z0mg>
* Other **Youtube/Neso Academy** double linked list videos.