

**Documentación PECL1**

**Universidad de Alcalá**

**Grado en Ingeniería en Sistemas de Información (GISI)**

**Estructuras de Datos**

**Tutor: José Miguel Alonso Martínez**

**David Sánchez Sánchez**

**Rodrigo Palomo Cuenca**

Universidad de Alcalá

Madrid, Spain

d.sanchezsanchez@edu.uah.es → 027551903E

rodrigo.palomo@edu.uah.es → 03220316V

****

**Índice**

[Detalles y justificación de implementación 3](#_Toc149436738)

[Especificación de interfaz de los TAD’s 3](#_Toc149436739)

[TAD’s creados 3](#_Toc149436740)

[Definición de las operaciones de los TAD’s 3](#_Toc149436741)

[Solución adoptada 8](#_Toc149436742)

[Dificultades encontradas 8](#_Toc149436743)

[Diseño relación entre las clases 8](#_Toc149436744)

[Explicación métodos más destacados 8](#_Toc149436745)

[Explicación del comportamiento del programa 9](#_Toc149436746)

[Bibliografía 10](#_Toc149436747)

# Detalles y justificación de implementación

## Especificación de interfaz de los TAD’s

### TAD’s creados

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Archivo al que pertenece | Con relación a | Descripción |
| Reserva | reserva.cpp & reserva.h | Reservas (Pág. 1 Pt.1) | TAD encargado de almacenar todos los atributos de una reserva |
| Mesa | mesa.h | Mesa (Pág. 1 Pt. 2) | TAD encargado de almacenar todos los atributos de mesa |
| NodoR | nodoReserva.h | cReservas y cPendientes (Pág 1 Pt.1) | Unidad mínima para guardar mesas en estructuras más complejas como Colas y que acepten el TAD Reserva |
| NodoM | nodoMesa.h | pMesas (Pág. 1 Pt. 2) | Unidad mínima para guardar mesas en estructuras más complejas como Pilas y que acepten el TAD Mesa |
| Pedido | Pedido.h | Pedido (Pag.1 Pt.3) | TAD encargado de almacenar todos los atributos de un pedido |
| NodoPedido | NodoPedido.h | cPedidos (Pag.1 Pt.3) | Unidad mínima para guardar pedidos en estructuras más complejas como Colas y que acepten el TAD Pedido |
| cReserva | colaReserva.h | cReservas y cPendientes (Pág 1 Pt.1) | TAD creado para poder tener un control de todas las reservas solicitadas y pendientes de dar (Seguimiento a través de punteros). |
| cPedidos | colaPedidos.h | cPedidos (Pag.1 Pt.3) | TAD creado para llevar un seguimiento de todas las reservas asignadas a una mesa (Seguimiento a través de punteros). |
| pMesa | pilaMesa.h | pMesas (Pág. 1 Pt. 2) | TAD creado para tener un seguimiento de todas las mesas libres (Seguimiento a través de punteros). |

### Definición de las operaciones de los TAD’s

#### Reserva

Reserva (string nombre, string situ, int num, string hora, string pref) 🡪 Constructor de la clase

Reserva () 🡪 Constructor vacío.

string toString() 🡪 Devuelve un string de todos los elementos de la reserva

string toStringMini() 🡪Devuelve un string de los elementos más importantes de la reserva

string getSituacion() 🡪 Devuelve un string del atributo situación

int getNumPersonas() 🡪 Devuelve un int del número de personas que hay en la reserva

string getNombre() 🡪 Devuelve el nombre al que está asociado la reserva

string getPrefMenu() 🡪 Devuelve un string con la preferencia del menú

void setSituacion(string situacion) 🡪 Establece una nueva situación para la reserva

void setNumPersonas(int \_numPersonas) 🡪 Establece un nuevo número de personas para la reserva

string getHoraReserva() 🡪 Devuelve en un string la hora programada para la reserva.

#### Pedido

Pedido(int \_numMesa, string \_nombreReserva, int \_numPersonas, string \_preferenciaMenu, string \_situacion, bool \_finalizado,Mesa \_mesadeReserva) 🡪 Constructor completo de la clase

Pedido() 🡪 Constructor vacío.

~Pedido() 🡪 Destructor del objeto.

string toString() 🡪 Devuelve un string con todos los datos del pedido.

string toStringMINI() 🡪 Devuelve un string con los datos más importantes del pedido.

int getNumMesa () 🡪 Devuelve el número de mesa al que está asociado el pedido.

getMesadeReserva () 🡪 Devuelve la mesa a la que está asociada el pedido.

setFinalizado(bool \_finalizado) 🡪 Establece un nuevo booleano para el atributo finalizado.

getReserva() 🡪 Devuelve la reserva asociada al pedido.

#### Mesa

Mesa(int nMesas, int capac, string situ) 🡪 Constructor completo de la clase Mesa

Mesa() 🡪 Concstructor vacío.

string toString() 🡪 Devuelve un string con todos los datos de la mesa.

string toStringMINI() 🡪 Devuelve un string con los datos más importantes de la mesa.

Reserva\* getReservaMesa() 🡪 Devuelve la reserva que está asociada a esa mesa.

void setReservaMesa(Reserva\* \_reservaMesa) 🡪 Establece una reserva a la mesa.

string getSituacion() 🡪 Devuelve un string que contiene la situación de la mesa.

void setSituacion(string \_situacion) 🡪 Establece una nueva situación a la mesa.

int getCapacidad() 🡪 Devuelve un int que contiene la capacidad de la mesa.

void setCapacidad(int \_capacidad) 🡪 Establece la capacidad de la mesa.

void setReserva(Reserva\* \_reservaMesa) 🡪 Establece la reserva que está asociada a la mesa.

Reserva\* getReserva() 🡪 Devuelve el puntero de la mesa que está asociada a esa mesa.

int getNumeroMesas() 🡪 Devuelve un entero con el número de mesa.

#### NodoReserva

NodoR(Reserva reserva, NodoR\* sig=nullptr) 🡪 Constructor de la clase.

~NodoR() 🡪 Destructor de la clase.

Reserva getValor() 🡪 Devuelve el contenido del nodo.

NodoR\* getSiguiente() 🡪 Devuelve un puntero al que apunta el Nodo.

void setSiguiente(NodoR\* \_siguiente) 🡪 Establece el puntero al elemento al que va a apuntar ahora el nodo.

#### NodoPedido

NodoPedido(Reserva reserva, NodoPedido\* sig=nullptr) 🡪 Constructor de la clase.

~NodoPedido() 🡪 Destructor de la clase.

Pedido getValor() 🡪 Devuelve el contenido del nodo.

NodoPedido\* getSiguiente() 🡪 Devuelve un puntero al que apunta el Nodo.

void setSiguiente(NodoPedido\* \_siguiente) 🡪 Establece el puntero al elemento al que va a apuntar ahora el nodo.

void setValor() 🡪 Establece un nuevo valor, que será un pedido, para el nodo-

#### NodoMesa

NodoM(Reserva reserva, NodoM\* sig=nullptr) 🡪 Constructor de la clase.

~NodoM() 🡪 Destructor de la clase.

Pedido getValor() 🡪 Devuelve el contenido del nodo.

NodoM\* getSiguiente() 🡪 Devuelve un puntero al que apunta el Nodo.

void setSiguiente(NodoM\* \_siguiente) 🡪 Establece el puntero al elemento al que va a apuntar ahora el nodo.

#### PilaMesa

PilaM() 🡪 Constructor de la pila.

~PilaM() 🡪 Destructor de la pila.

void insertar(Mesa &v) 🡪 Inserta un elemento en la pila (en la cima).

Mesa extraer() 🡪 Extrae un elemento de la pila (la cima).

void mostrar() 🡪 Muestra todos los elementos contenidos en la pila.

void mesasNuevasRandom(int n) 🡪 Genera un numero indicado de mesas y las inserta en la pila.

void inicializar() 🡪 Inicializa la pila metiendo mesas generadas de manera aleatoria.

bool estaVacia() 🡪 Devuelve un bool indicando si la pila está vacía o no.

void reservaRandPg2P4 () 🡪 Genera mesas en función de lo que se pide en el enunciado de la práctica.

NodoM\* getUltimo() 🡪 Devuelve el elemento que este más arriba de la pila(la cima).

#### ColaReserva

ColaR() 🡪 Constructor de la cola.

~ColaR() 🡪 Destructor de la cola.

bool estaVacia() 🡪 Devuelve un bool indicando si la cola está vacía o no.

void insertar(Reserva v) 🡪 Inserta un elemento en la cola

void eliminar() 🡪 Elimina el elemento posicionado en primera posición de la cola.

void mostrar() 🡪 Muestra todos los elementos de la cola.

void reservasNuevasRandom(int n) 🡪 Genera un numero de reservas indicado y la mete en la cola.

void reservaRandPg2P1() 🡪 Genera reservas en función de cómo se pide en el enunciado y las mete a la cola.

NodoR\* getPrincipio() 🡪 Devuelve el nodo del primer elemento de la cola.

tamañoCola() 🡪 Devuelve la cantidad de elementos que hay en la cola.

#### ColaPedido

ColaP() 🡪Constructor de la cola

~ColaP() 🡪 Destructor de la cola.

bool estaVacia() 🡪 Devuelve un bool indicando si la cola está vacía.

void eliminar() 🡪 Elimina el elemento que este en la primera posición de la cola.

void insertar(Pedido pedido) 🡪 Inserta un pedido al final de la cola.

void mostrar() 🡪 Muestra todos los pedidos de la cola.

NodoPedido\* getPrimero() 🡪 Devuelve el nodo que este posicionado al principio de la cola.

NodoPedido\* getUltimo () 🡪 Devuelve el nodo que este posicionado al final de la cola .

tamañoCola() 🡪 Devuelve la cantidad de elementos que hay en la cola.

## Solución adoptada

### Consideraciones del enunciado

1. *"Una vez finalizadas todas las reservas de una hora, y antes de comenzar con las reservas de la siguiente, se marcarán como finalizados la mitad de los pedidos de cPedidos, simulando que quedan libres las mesas correspondientes (que se apilarán en la pila pMesas)."*
   1. Hemos interpretado que se marcarán la mitad DEL NÚMERO DE LA TANDA INTRODUCIDA, es decir, si en la cola de reservas hay 4 de cada hora, el número de cada tanda será 4 y por tanto se marcarán como finalizados 2 pedidos.
2. También en esta cuestión hemos interpretado que los pedidos al ser marcados no se eliminan de la cola Pedidos, permanecen allí como una especie de histórico.
3. En la asignación de mesas para una reserva hemos decidido que una reserva de menos de 5 personas no podrá obtener una de 8 aunque no haya de 4, pasarán a pendientes.
4. A la hora de mostrar el resultado, sobre todo en el ejercicio 9, hemos añadido símbolos y flags para que se lea correctamente el trascurso del programa.

### Dificultades encontradas

En cuanto a las dificultades encontradas han sido varias, lo primero y que más problemas nos ha ocasionado ha sido el tema de errores ya que c++ no aporta mucha información de los errores ocurrido en el código.

Luego ha sido muy difícil solucionar errores de punteros ya que al estar todo almacenado en memoria dinámica el concepto de abstracción para saber en qué momento pasaban las cosas y donde ha sido muy complicado de asimilar y muchas veces no se ha podido llegar a asimilar del todo.

El uso de pilas y colas ha sido fácil de entender como concepto y de implementar de una manera básica, pero a la hora de meter restricciones y métodos más complejos, el nivel de abstracción sube, los errores aumentan y a su vez es más difícil de resolverlos, y todo es más difícil de implementar.

En nuestro punto de vista la mayor complicación de la práctica se ha condensado en los punteros, no en el entendimiento y el desarrollo, si no a la hora de implementarlo en tantas variables y clase ya que no sabes “donde están las cosas” por la cantidad de memoria dinámica procesada y creada en el proyecto.

## Diseño relación entre las clases

### Explicación métodos más destacados

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Archivo al que pertenece | En relación con | Descripción |
| asignacionMesa() | Metodos.h | Pág. 2 Pt.7 | método encargado de hacer todo lo que se pide en el punto 7 de la página 2 con todas las restricciones que se detallan en el documento, informa al programador de que hay una mesa disponible para esa reserva |
| mesadeReserva() | Metodos.h | Pág. 2 Pt.7 | Asigna la mesa a una reserva sabiendo con anterioridad que esta disponible una reserva para ese tipo de mesa |
| reservaCorrespondeMesa() | Metodos.h | Pág. 2 Pt.7 | Se usa para saber si la reserva coincide con el tipo de características de la mesa para posteriormente poder asignarla |
| extraerMesaDePila | Metodos.h | Pág. 2 Pt.7 | Usado para eliminar la mesa de la pila cuando una reserva se asocia a esa mesa (Ya que esa mesa no estará disponible más) |
| abrirMenu() | Menu.h | Pág. 2 | método encargado de gestionar todo el menú de la aplicación y todas las funciones que se exigen en el enunciado. |

### Explicación del comportamiento del programa

Primero, el main del programa realmente no hace gran cosa, donde realmente adopta importancia es cuando se ejecuta la función abrirMenu(), ya que esta va a mostrar al usuario las 9 opciones que se consideran en la práctica, siendo capaz de tratar todo tipo de errores que puedan surgir por la parte del usuario que usa el programa

##### Generar 12 reservas

Genera 12 reservas teniendo en cuenta las condiciones del enunciado y también algunas restricciones, si ya hay reservas creadas las elimina.

##### Mostrar todas las reservas

Muestra todas las reservas que estén en la cola de reservas teniendo en cuenta algunas restricciones, si no hay nada en la cola de reservas te salta un mensaje avisando de que no hay reservas en la cola.

##### Borrar todas las reservas

Borra todas las reservas que estén en la cola de reservas teniendo en cuenta algunas restricciones, si no hay reservas en la cola te avisa de que está vacía y no hay nada que borrar.

##### Generador 20 mesas

Genera 20 mesas siguiendo las restricciones del enunciado y comprobando que, si hay ya mesas, estas las elimina primero y luego crea las 20 que se le piden.

##### Mostrar mesas

Muestra todas las mesas que estén en la pila de mesas, teniendo en cuenta que si no hay mesas en la pila saltara un mensaje avisando de que o hay mesas aun apiladas.

##### Borrar mesas

Borra todas las mesas que estén en la pila teniendo en cuenta de que si no hay mesas apiladas saltara un mensaje al usuario diciendo que no hay mesas que borrar todavía.

##### Gestión Completa de 1 reserva

Lo primero que se comprueba es que haya reservas en cola y mesas en la pila, de lo contraria, lo genera todo.

después de esa comprobación ejecutara el método asignacionMesa(), el cual por dentro asignara una reserva a una mesa y devolviéndonos un booleano true. Tras devolver ese valor se crear el pedido correspondiente con los datos de la reserva y de la mesa y se insertaría en la cola “cPedidos”. Finalmente, se le avisaría al usuario de lo sucedido y se eliminará la reserva de la cola “cReservas” y la mesa de la pila “pMesas”.

Si no asignase la reserva a la mesa nos devolvería un booleano false, se avisaría al usuario de lo sucedido y se asignará esa reserva a la cola de “cPendientes”.

Para finalizar se muestra la cola de reservas (“cReservas”) por pantalla al usuario, la cola de pendientes (“cPendientes”), la pila de las mesas disponibles (“pMesas”) y por último la cola de los pedidos asignados (“cPedidos”).

##### Definir simulación 2

##### Definir simulación 3

##### Salir del programa

El menú se ejecuta sobre un bucle while que en cada iteración comprueba si el usuario ha introducido como opción la opción 0 (Salir del programa). Si es así, el método abrirMenu() se acabaría dejando al main llegar al final borrando toda la memoria dinámica y acabando el programa finalmente.

# Bibliografía

<https://stackoverflow.com/> 🡪Para tema de errores de compilación y reglas de c++ a la hora de programar código ha sido muy útil.