# Tarea 4
# TAD Tabla y Cola de Prioridad
## Curso 2024

## 1. Introducción

Esta tarea tiene como principales objetivos:

- Continuar trabajando sobre el manejo dinámico de memoria.
- Continuar trabajando con el concepto de tipo abstracto de datos (TAD).
- Continuar trabajando en el uso de TADs como auxiliares para la resolución de problemas.
- Trabajar en la implementación de TAD a partir de su especificación y utilizando nuevas estructuras de datos (hash y heap).

La fecha límite de entrega es el **miércoles 20 de noviembre a las 16.00 horas**. El mecanismo de entrega se explica en la Sección 7.
Al igual que en tareas anteriores se dejan enlaces a los **foros** para plantear **dudas específicas de cada módulo**. Además, se disponibiliza un foro para dudas generales.

## 2. Descripción de las funcionalidades a implementar

En esta tarea, y en base a lo construido en tareas anteriores, se completarán las funcionalidades del MercadoFinger. Se agrega el manejo de la cadena de tiendas y la gestión de una tabla de quejas y una cola de envíos.
La cadena manejará las quejas recibidas (ver Sección 3) que posteriormente almacenará en una estructura de hash (ver Sección 4).
Adicionalmente, la cadena de tiendas manejará los envíos pendientes de todas sus tiendas (ver Sección 5), los cuales se gestionarán mediante una cola de prioridad según la fecha de envío más próxima (ver Sección 6) .

A continuación se presenta una **guía** que deberá **seguir paso a paso** para resolver la tarea.
Tenga en cuenta que la especificación de cada función se encuentra en el **.h** respectivo, y para la mayoría de las funciones se especifica cuál debe ser el orden del tiempo de ejecución en el **peor caso** o el **caso promedio**.

## 3. Módulo Queja

En esta sección se implementa el módulo (*queja.cpp*) que representa las quejas que los clientes realizan sobre los productos. Se pide:

1. **Implemente** la estructura *rep_queja* que almacenar el cliente que realiza la queja, el producto sobre el cual se realiza la queja, un comentario que describe la queja (ejemplo: ."el producto es de mala calidad") y la fecha en que se realiza la queja. Foro de dudas.
2. **Implemente** las funciones crearTQueja,fechaTQueja, imprimirTQueja y liberarTQueja. Tenga en cuenta que el formato de impresión se verifica en *queja.h*. Ejecute el test queja1-crear-consultar-liberar para verificar el funcionamiento de las funciones. Foro de dudas.

# 4. Módulo Tabla de Quejas (TAD Tabla)

En esta sección se implementará el módulo *TablaQuejas.cpp*. Este TAD se utilizará para almacenar las quejas, y consiste en una tabla no acotada cuyo dominio son las *fechas* de las quejas y el codominio son elementos del tipo *TQueja*. La tabla debe ser implementada mediante una **tabla de dispersión abierta**. Asuma que no hay quejas con la misma fecha.

1. **Implemente** la estructura *rep_tablaquejas*. Para esto, recomendamos revisar las operaciones del TAD solicitadas y realizar un diseño de la/s estructura/s necesaria/s. La representación debe implementar listas del tipo *TQueja*, a definirse internamente en el módulo. Foro de dudas.

2. **Implemente** las funciones crearTablaQuejas, liberarTablaQuejas, agregarQuejaTablaQuejas e imprimirTablaQuejas. La función de inserción recibe una Tabla de Quejas (TablaQuejas) y una TQueja, y asocia en la tabla a la TQueja con su fecha. Para calcular la posición en la que se debe insertar a la TQueja en la tabla de dispersión abierta se debe utilizar la función brindada **funcionHash** (la función está declarada al comienzo de TablaQuejas.cpp). La queja debe ser ubicada en la la posición de la tabla indicada por dicha función. Por convención, se deberá insertar la queja al inicio de la lista definida para dicha posición de la tabla. La función imprimirTablaQuejas debe imprimir cada queja de la tabla, en orden creciente de posiciones asociadas en la tabla. En caso de que haya más de una queja en la misma posición, se deben imprimir en el orden inverso al que fueron agregadas (que será el orden natural en que se recorrerá la lista). En el archivo *hashListaQuejas.h* se encuentra una descripción más detallada del formato de impresión. Ejecute los tests tablaQuejas1-crear-consultar-liberar y tablaQuejas2-crear-consultar-liberar-muchas-veces para verificar el funcionamiento de las funciones. Foro de dudas.

3. **Implemente** las funciones perteneceQuejaTablaQuejas y obtenerQuejaTablaQuejas. Ejecute el test tablaQuejas3-obtener-pertenece. Foro de dudas.

4. **Ejecute** el test tablaQuejas4-obtener-pertenece-muchas-veces. Foro de dudas.

# 5. Módulo Envío

En esta sección se implementará el módulo *Envío*, que representa el envío a un cliente. Los envíos están formados por un carrito de productos, con los productos comprados por el cliente, y la fecha en que deberá hacerse el envío.

1. **Implemente** la representación de envio *rep_envio* según la descripción brindada en el párrafo anterior. Foro de dudas.

2. **Implemente** la función crearTEnvio para crear un elemento de tipo TEnvio, la cual recibirá como parámetro el carrito de productos asociado al envío y la fecha en la cual deberá realizarse el envío. Foro de dudas.

3. **Implemente** la función liberarTEnvio para liberar la memoria asignada. Foro de dudas.

4. **Implemente** las funciones obtenerCarritoProductosTEnvio, que retorna el carrito de productos asociado al envío, y obtenerFechaTEnvio, que retorna la fecha de un envío dado. Foro de dudas.

5. **Implemente** el procedimiento imprimirTEnvio que imprime la información de un envío. El procedimiento imprime el carrito de productos y la fecha del envío, según el formato dado en envio.h. Foro de dudas.

6. **Ejecute** el test envio1-crear-consultar-liberar para verificar que el funcionamiento de las funciones y procedimientos implementados es el correcto. Foro de dudas.

   En las siguientes secciones se trabajará sobre los envíos pendientes.

# 6. Módulo Cola de Envíos (TAD Cola de Prioridad)

En esta sección se implementará en *colaEnvios.cpp* el TAD Cola de Prioridad a partir de la especificación dada en *colaEnvios.h*.
La cola de envíos almacena elementos del tipo TEnvio en una cola de tamaño máximo N. El criterio por defecto para establecer la prioridad entre envíos es que un envío e1 es prioritario ante otro envio e2 si la fecha de envío de e1 es menor que la de e2. Este criterio se puede modificar (con la función invertirPrioridadTColaEnvios), con lo que se logra que el envío prioritario sea el de fecha de envío mayor. Asuma que no hay envíos con la misma fecha.
Si la cola de envíos no es vacía, hay un envío considerado el prioritario según el criterio de prioridad. Para esta implementación se recomienda utilizar la estructura de *heap* (montículo binario) vista en el curso. Además, podrá ser necesario considerar estructuras auxiliares para cumplir con los órdenes de tiempo de ejecución de algunas operaciones.

1. **Implemente** la estructura *rep_cola_envios*. Para esto, recomendamos revisar las operaciones del TAD solicitadas y realizar un diseño de la/s estructura/s necesaria/s. Foro de dudas.

2. **Implemente** las funciones crearTColaEnvios, liberarTColaEnvios, imprimirTColaEnvios y encolarEnvioTColaEnvios. Recomendamos que declare e implemente la función auxiliar *void filtradoAscendente(int pos, TColaEnvios &colaEnvios)* que realiza el filtrado ascendente en el heap. **Ejecute** el test colaEnvios1-crear-consultar-liberar para verificar las funciones. Foro de dudas.

3. **Implemente** la función desencolarTColaEnvios y cantidadTColaEnvios. Recomendamos que implemente la función auxiliar *void filtradoDescendente(int pos, TColaEnvios &colaEnvios)* que realiza el filtrado descendente en el heap. **Ejecute** el test colaEnvios2-encolar-desencolar-cantidad para verificar las funciones. Foro de dudas.

4. **Ejecute** el test colaEnvios3-encolar-desencolar-cantidad-muchas-veces respetando los tiempos de ejecución solicitados. Foro de dudas.

5. **Implemente** la funcion masPrioritarioTColaEnvios, invertirPrioridadTColaEnvios y maxTColaEnvios . La función invertirPrioridadTColaEnvios debe modificar la cola de forma de que se respete el nuevo criterio de prioridad. **Ejecute** el test colaEnvios4-invertir-max-mas-prioritaria para verificar las funciones. Foro de dudas.

6. **Ejecute** el test colaEnvios5-invertir-max-mas-prioritaria-muchas-veces respetando los tiempos de ejecución solicitados. Foro de dudas.

# 7. Test final y entrega de la tarea

Para finalizar con la prueba del programa utilice la regla *testing* del Makefile y verifique que no hay errores en los tests públicos. Esta regla se debe utilizar **únicamente luego de realizados todos los pasos anteriores (instructivo especial para PCUNIX en paso 3)**.

1. **Ejecute**:

   ```
   $ make testing
   ```

   Si la salida no tiene errores, al final se imprime lo siguiente:

   ```
   -- RESULTADO DE CADA CASO --
   11111111111
   ```

   donde un 1 simboliza que no hay error y un 0 simboliza un error en un caso de prueba, en este orden:

```
envio1-crear-consultar-liberar
queja1-crear-consultar-liberar
tablaQuejas1-crear-consultar-liberar
tablaQuejas2-crear-consultar-liberar-muchas-veces
tablaQuejas3-obtener-pertenece
tablaQuejas4-obtener-pertenece-muchas-veces
colaEnvios1-crear-consultar-liberar
colaEnvios2-encolar-desencolar-cantidad
colaEnvios3-encolar-desencolar-cantidad-muchas-veces
colaEnvios4-invertir-max-mas-prioritaria
colaEnvios5-invertir-max-mas-prioritaria-muchas-veces
```

Foro de dudas.

2. **Prueba de nuevos tests**. Si se siguieron todos los pasos anteriores, el programa creado debería ser capaz de ejecutar todos los casos de uso presentados en los tests públicos. Para asegurar que el programa es capaz de ejecutar correctamente ante nuevos casos de uso es importante realizar tests propios, además de los públicos. Para esto **cree un nuevo archivo en la carpeta test**, con el nombre *test_propio.in*, y **escriba una serie de comandos** que permitan probar casos de uso que no fueron contemplados en los casos públicos. **Ejecute el test** mediante el comando:

   ```
   $ ./principal < test/test_propio.in
   ```

   y verifique que la salida en la terminal es consistente con los comandos ingresados. La creación y utilización de casos de prueba propios, es una forma de robustecer el programa para la prueba de los casos de test privados. Foro de dudas.

3. **Prueba en pcunix**. Es importante probar su resolución de la tarea con los materiales más recientes y en una pcunix, que es el ambiente en el que se realizarán las correcciones. Para esto siga el procedimiento explicado en Sugerencias al entregar.

   **IMPORTANTE:** Debido a un problema en los *pcunix*, al correrlo en esas máquinas se debe iniciar valgrind **ANTES** de correr *make testing* como se indica a continuación:

   **Ejecutar los comandos**:

   ```
   $ make
   $ valgrind ./principal
   ```

   Aquí se debe **ESPERAR** hasta que aparezca:

   ```
   $ valgrind ./principal
   ==102508== Memcheck, a memory error detector
   ==102508== Copyright (C) 2002-2022, and GNU GPL'd, by Julian $ Seward et al.
   ==102508== Using Valgrind-3.20.0 and LibVEX; rerun with -h for copyright info
   ==102508== Command: ./principal
   ==102508==
   $ 1>
   ```

   Luego se debe ingresar el comando **Fin** y recién luego ejecutar:

   ```
   $ make testing
   ```

   Foro de dudas.

4. **Armado del entregable**. El archivo entregable final debe generarse mediante el comando:

```
$ make entrega
```

Con esto se empaquetan los módulos implementados y se los comprime generando el archivo `EntregaTarea4.tar.gz`.

El archivo a entregar **DEBE** ser generado mediante este procedimiento. Si se lo genera mediante alguna otra herramienta (por ejemplo, usando un entorno gráfico) **la tarea no será corregida**, independientemente de la calidad del contenido. Tampoco será corregida si el nombre del archivo se modifica en el proceso de entrega. Foro de dudas.

5. **Subir la entrega al receptor**. Se debe entregar el archivo **EntregaTarea4.tar.gz**, que contiene los módulos a implementar **colaEnvios.cpp**, **hashListaQuejas.cpp**, **envio.cpp**, **queja.cpp**. Una vez generado el entregable según el paso anterior, es necesario subirlo al receptor ubicado en la sección Laboratorio del EVA del curso. **Recordar que no se debe modificar el nombre del archivo generado mediante** `make entrega`. Para verificar que el archivo entregado es el correcto se debe acceder al receptor de entregas y hacer click sobre lo que se entregó para que automáticamente se descargue la entrega.
**IMPORTANTE:** Se puede entregar **todas las veces que quieran** hasta la fecha final de entrega. La última entrega **reemplaza a la anterior** y es la que será tenida en cuenta. Foro de dudas.