Tarea 4
TADS: Tabla y Cola de Prioridad
Curso 2025
Segundo semestre

1. Introducción

Esta tarea tiene los siguientes objetivos principales:

- Implementar los TADs colaDePrioridad y tabla.
- Utilizar un hash de dispersión abierta en la implementación de la tabla.
- Utilizar los módulos implementados para resolver problemas de la realidad planteada.

La fecha límite de entrega es el **miércoles 19 de noviembre a las 10:00 horas**. El mecanismo específico de entrega se explica en la Sección 5. Por otro lado, para plantear **dudas específicas de cada paso** de la tarea, se deja un link a un **foro de dudas** al final de cada parte.
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 cada función se especifica cuál debe ser el orden del tiempo de ejecución en el **peor caso o caso promedio, según se indique**.

2. Partiendo de la tarea 3

Recuerde colocar todas las implementaciones de los módulos desarrollados hasta el momento en el directorio *src*.
Para realizar esta tarea, vamos a necesitar extender el módulo *socio*, debido a que surgió el requerimiento de rankear (clasificar) a los socios según un rango (grado) de fidelidad a la biblioteca.

Por lo anterior se solicita:

1. **Modifique** la estructura *rep_socio* de forma que incluya el rango del socio. A su vez **implemente** la función rangoTSocio la cual retorna el rango (grado de fidelidad) del socio. **Ejecute** el test socio1-rango para verificar el funcionamiento de la función implementada. Foro de dudas.

3. TAD Cola de Prioridad: módulo colaDePrioridadReservas

> Recomendamos que antes de comenzar con la implementación de este módulo, estudie los contenidos asociados al TAD *Cola de Prioridad* en la sección 9 - Colas de prioridad e implementaciones, incluyendo heap del eva.

En esta sección se implementará el módulo *colaDePrioridadReservas.cpp*, el cual implementa las funciones del TAD Cola de Prioridad. La estructura de tipo TColaDePrioridadReservas almacenará elementos del tipo TReserva, pero en este caso se implementará una prioridad para obtener eficientemente la reserva más prioritaria. Se establece entonces una prioridad por el atributo *rango* de cada socio asociado a la reserva, donde a menor rango mayor prioridad.
De esta forma, si la estructura no es vacía hay una reserva considerada la prioritaria 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 ordenes de tiempo de ejecución de algunas operaciones.

1. **Implemente** la estructura *rep_colaDePrioridadReservas* que permita implementar las funciones del módulo en el orden indicado. Se recomienda estudiar las funciones solicitadas previo a realizar el diseño de la estructura. Foro de dudas.

2. **Implemente** las funciones crearTColaDePrioridadReservas, liberarTColaDePrioridadReservas y estaVaciaTColaDePrioridadReservas Verifique el funcionamiento de las funciones ejecutando el test colaPrioridadReservas1-crear-esVacia-liberar. Foro de dudas.

3. **Implemente** las funciones insertarTColaDePrioridadReservas, eliminarPrioritarioTColaDePrioridadReservas y prioritarioTColaDePrioridadReservas. **Ejecute** el test colaPrioridadReservas2-insertar-eliminar-prioritario para verificar el funcionamiento de las funciones. Foro de dudas.

4. **Implemente** las funciones estaTColaDePrioridadReservas y prioridadTColaDePrioridadReservas. **Ejecute** el test colaPrioridadReservas3-esta-prioridad para verificar el funcionamiento de las funciones implementadas. Foro de dudas.

5. **Implemente** la función invertirPrioridadTColaDePrioridadReservas. **Ejecute** el test colaPrioridadReservas4-invertir para verificar el funcionamiento de la función implementada. Foro de dudas.

6. **Implemente** las funciones copiarTColaDePrioridadReservas y imprimirTColaDePrioridad. **Ejecute** el test colaPrioridadReservas5-copiar-imprimir para verificar el funcionamiento de las funciones implementadas. Foro de dudas.

7. **Ejecute** el test colaPrioridadReservas6-combinado. Foro de dudas.

4. TAD Tabla: módulo tablaReserva

Recomendamos que antes de comenzar con la implementación de este módulo, estudie los contenidos asociados al TAD *Tabla* en la sección 10- Multiconjuntos y Tablas (Funciones parciales) del eva.

En esta sección se implementará el módulo *tablaReserva.cpp*. Este módulo se utilizará para almacenar las reservas de los libros de la biblioteca, de forma tal que se pueda acceder a las reservas de un libro en particular, a partir de su isbn. Para ello, se utiliza un hash, donde cada posición de la tabla contiene una cola de prioridad con las reservas del libro asociado a esa posición. Si bien actualmente los isbns de los libros son acotados, para facilitar cambios futuros el módulo se deberá implementar como una **tabla de dispersión abierta**.

1. **Implemente** la estructura *rep_tablaReserva*. 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 que permitan almacenar elementos de tipo *TAGTablaReserva*. Las estructuras adicionales deben definirse internamente en el módulo. Foro de dudas.

2. **Implemente** las funciones crearTTablaReserva y liberarTTablaReserva. Ejecute el test tablaReservas1-crear-liberar para verificar el funcionamiento de la funciones. Foro de dudas.

3. **Implemente** las funciones insertarTTablaReserva y perteneceTTablaReserva. La función de inserción recibe una TTablaReserva, un isbn y un elemento TReserva. Se asocia en la tabla la TReserva con el isbn del libro. Para calcular la posición en la que se debe insertar la reserva en la tabla de dispersión abierta se debe realizar simplemente (isbnLibro % tamaño), donde el tamaño de la tabla es un parámetro de la función de crearTTablaReserva. La reserva debe ser ubicada en la posición calculada. Ejecute el test tablaReservas2-insertar-pertenece para verificar el funcionamiento de las funciones. Foro de dudas.

4. **Implemente** las funciones obtenerReservaTTablaReserva y obtenerSigReservaTTablaReserva. Ejecute el test tablaReservas3-obtener-siguiente para verificar el funcionamiento de las funciones. Foro de dudas.

5. **Ejecute** el test tablaReservas4-combinado. Foro de dudas.

5. 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:

   ```
   colaPrioridadReservas1-crear-esVacia-liberar
   colaPrioridadReservas2-insertar-eliminar-prioritario
   colaPrioridadReservas3-esta-prioridad
   colaPrioridadReservas4-invertir
   colaPrioridadReservas5-copiar-imprimir
   colaPrioridadReservas6-combinado
   socio1-rango
   tablaReservas1-crear-liberar
   tablaReservas2-insertar-pertenece
   tablaReservas3-obtener-siguiente
   tablaReservas4-combinado
   ```

   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 nuevos módulos a implementar **tablaReserva.cpp** y **colaDePrioridadReservas.cpp**, además de los módulos de tareas anteriores. 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.