Tarea 3
TADS: Árbol General Cola y Conjunto
Curso 2025
Segundo semestre

1. Introducción

Esta tarea tiene los siguientes objetivos principales:

- Continuar trabajando sobre el manejo dinámico de memoria.
- Profundizar en estructuras arborescentes y funciones recursivas a través de la implementación de un árbol general.
- Implementar los TADs pila y cola.
- Implementar funciones complejas haciendo uso de los TADs implementados.

La fecha límite de entrega es el **miércoles 22 de octubre a las 10:00 horas**. El mecanismo específico de entrega se explica en la Sección 8. 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**.

2. Partiendo de la tarea 2

Recuerde colocar sus implementaciones de los modulos fecha, libro, socio, prestamo, ldePrestamos, lseSocios y abbLibros en el directorio *src*. Notar que se agregaron 2 funciones adicionales que se deben implementar en los módulos *ldePrestamos* y *prestamos*. La firma de ambas funciones se declaran en los archivos *ldePrestamos.h* y *prestamos.h* de estos módulos.

1. En el módulo *ldePrestamos*, implemente la función adicional liberarTLDEPrestamosSoloEstructura. Esta función hace uso de otra, presente en el modulo *prestamos*, llamada liberarTPrestamoSoloEstructura. Estas dos nuevas funciones serán empleadas en los test del modulo *biblioteca* (7) y se emplearan para liberar únicamente la memoria de la estructura del préstamo, pero sin liberar la memoria del socio ni del libro a los que apuntan. Foro de dudas.

3. Módulo reserva

En esta sección se describe la implementación del módulo *reserva.cpp*. Los elementos del tipo TReserva representan una reserva de un socio de un determinado libro de la Biblioteca. Cada nodo contiene un elemento de tipo TSocio y otro TLibro.

1. **Implemente** la representación de reserva *rep_reserva* y las funciones crearTReserva,liberarTReserva e imprimirTReserva. Tenga en cuenta que el formato de impresión se especifica en *reserva.h*. Ejecute el caso de prueba reserva1-crear-eliminar para verificar el funcionamiento de las operaciones. Foro de dudas.
2. **Implemente** las funciones socioTReserva, libroTReserva y copiarTReserva. Ejecute el caso de prueba reserva2-socio-libro-copiar para verificar el funcionamiento de las operaciones. Foro de dudas.
3. **Implemente** le función liberarTReservaSoloEstructura que libera únicamente la memoria de la estructura de la reserva, pero **no** libera la memoria del socio ni del libro a los que apunta. Esta función sera empleadas en los test del modulo *biblioteca* (7). Foro de dudas.

4. TAD Conjunto: módulo conjuntoGeneros

Recomendamos que antes de comenzar con la implementación de este módulo, estudie los contenidos asociados al TAD *Conjunto* en la sección Conjuntos e implementaciones del eva.

En esta sección se describe la implementación del módulo *conjuntoGeneros.cpp*. El módulo ofrece las operaciones típicas del TAD Conjunto. Consiste en un conjunto acotado de identificadores de géneros, que cumplen 0 <= id <cantMax, donde cantMax es el id máximo que pueden tener los subgéneros de un género y por lo tanto también indica la máxima cantidad de elementos en el conjunto.

1. **Implemente** la estructura *rep_conjuntogeneros*, que almacena un conjunto acotado de enteros y que permita satisfacer los órdenes de tiempo de ejecución solicitados en conjuntoGeneros.h Foro de dudas.
2. **Implemente** las funciones crearTConjuntoGeneros, insertarTConjuntoGeneros, imprimirTConjuntoGeneros y liberarTConjuntoGeneros. Verifique el funcionamiento de las funciones ejecutando el test conjuntoGeneros1-crear-insertar-imprimir-liberar. Foro de dudas.
3. **Implemente** las funciones esVacioTConjuntoGeneros, cardinalTConjuntoGeneros y cantMaxTConjuntoGeneros. Verifique el funcionamiento de las funciones ejecutando el test conjuntoGeneros2-esvacio-cardinal-cantmax. Foro de dudas.
4. **Implemente** las funciones perteneceTConjuntoGeneros y borrarDeTConjuntoGeneros. **Ejecute** el caso de prueba conjuntoGeneros3-pertenece-borrar. Foro de dudas.
5. **Implemente** las funciones unionTConjuntoGeneros, interseccionTConjuntoGeneros y diferenciaTConjuntoGeneros. **Ejecute** el caso de prueba conjuntoGeneros4-union-interseccion-diferencia. Foro de dudas.
6. **Ejecute** el caso de prueba conjuntoGeneros5-combinado. Foro de dudas.
7. **Ejecute** el caso de prueba conjuntoGeneros6-tiempo. Recuerde ejecutarlo con el comando make ttconjuntoGeneros6-tiempo para ejecutar sin valgrind. Foro de dudas.

5. Árbol General: módulo agGeneros

Recomendamos que antes de comenzar con la implementación de este módulo, estudie los contenidos asociados a la estructura *Árbol General* en la sección Estructuras arborecentes de memoria dinámica del eva. Las clases de práctico del tema árboles generales fueron durante la primer mitad del semestre.

En esta sección se implementará el módulo *agGeneros.cpp*, el cual representa la estructura y jerarquía de géneros y subgéneros literarios. Los nodos del arbol van a almacenar el código (int) identificador y el nombre del genero, y van a tener asociado como nodos hijos los subgeneros. Un elemento de tipo TAGGeneros estará implementado como un **árbol general (AG)**. Se recomienda que las implementaciones sean recursivas.

1. **Implemente** la representación del árbol general *rep_agGeneros*. Se sugiere una implementación primer hijo, siguiente hermano. Foro de dudas.
2. **Implemente** las funciones crearTAGGeneros, insertarGeneroTAGGeneros, imprimirTAGGeneros y liberarTAGGeneros.

   La impresión de la información del árbol general de géneros se realiza imprimiendo el código del género un nodo por línea, dejando espacios a la izquierda según la altura del nodo. Por ejemplo, un árbol dado por un nodo raíz de valor 1 y genero Terror, con dos hijos 12 y 11 uno con genero Comedia y otro Accion respectivamente, cada uno con hijos 123, 122 con generos Drama e Historica y 113 con genero Romance respectivamente, se vería como se indica a continuación:

```
1 - Terror
        12 - Comedia
                123 - Drama
                122 - Historica
        11 - Accion
                113 - Romance
```

La especificación completa, en particular respecto al orden de impresión de los elementos y el formato de los espacios, se indica en el correspondiente archivo .h.

Ejecute los tests agGeneros1-crear-liberar y agGeneros2-insertar-imprimir para verificar el funcionamiento de las funciones. Foro de dudas.

3. **Implemente** las funciones existeGeneroTAGGeneros, alturaTAGGeneros y cantidadTAGGeneros. **Ejecute** el test agGeneros3-existe-altura-cantidad para verificar el funcionamiento de las funciones. Foro de dudas.
4. **Implemente** las funciones removerGeneroTAGGeneros y nombreGeneroTAGGeneros. **Ejecute** el test agGeneros4-remover para verificar el funcionamiento de las funciones. Foro de dudas.
5. **Implemente** las funciones obtenerSubarbolTAGGeneros y obtenerMaxTAGGeneros. **Ejecute** el test agGeneros5-max-subarbol para verificar el funcionamiento de las funciones. Foro de dudas.
6. **Implemente** la función obtenerConjuntoGeneros. **Ejecute** el test agGeneros6-conjunto para verificar el funcionamiento de las funciones. Foro de dudas.
7. **Ejecute** el test agGeneros7-combinado. Foro de dudas.

6. TAD Cola: módulo colaReservas

Recomendamos que antes de comenzar con la implementación de este módulo, estudie los contenidos asociados al TAD *Cola* en la sección Introducción a TADs. Lista, Pila y Cola del eva.

La biblioteca mantiene un registro con las reservas de libros de los socios. Cuando un socio solicita un libro en préstamo que no fue aún devuelto, el pedido de préstamo es registrado en una cola de reservas.
En esta sección se implementará el módulo *colaReservas.cpp*, el cual implementa las funciones del TAD Cola. La estructura de tipo TColaReservas almacenará elementos del tipo TReserva. Esta estructura lineal es dinámica y permitirá almacenar una cantidad no acotada de reservas de libros.

1. **Implemente** la estructura *rep_colaReservas* que permita implementar las funciones del módulo en el orden indicado. Se recomienda utilizar estructuras auxiliares. Foro de dudas.
2. **Implemente** las funciones crearTColaReservas y liberarTColaReservas. Verifique el funcionamiento de las funciones ejecutando el test colaReserva1-crear-liberar. Foro de dudas.
3. **Implemente** las funciones encolarTColaReservas, e imprimirTColaReservas. **Ejecute** los tests colaReserva2-encolar-imprimir para verificar el funcionamiento de las funciones. Foro de dudas.
4. **Implemente** las funciones cantidadTColaReservas, frenteTColaReservas y desencolarTColaReservas. **Ejecute** el test colaReserva3-cantidad-frente-desencolar para verificar el funcionamiento de las funciones. Foro de dudas.
5. **Implemente** las función liberarTColaReservasSoloEstructura que libera únicamente la memoria asociada a todos los nodos y a las reservas que contienen, pero **no** libera la memoria del socio ni del libro a los que apunta. Ademas, implemente la función extraerFrenteTColaReservas que quita el nodo del frente de la cola y devuelve el TReserva que contenía, liberando únicamente la memoria del nodo. Esta función no libera la reserva. Estas funciones serán empleadas en los test del modulo *biblioteca* (7). Foro de dudas.
6. **Ejecute** el test colaReserva4-combinado. Foro de dudas.

7. Módulo Biblioteca

En esta sección se solicitará la implementación del módulo que implementa al TAD biblioteca. Este utilizará los distintos módulos y TADs definidos anteriormente en la tarea para la implementación de las funciones de éste módulo.
Las funciones solicitadas en esta sección se encuentran en el archivo *biblioteca.h* del módulo biblioteca.

1. **Implemente** las funciones crearTBiblioteca, liberarTBiblioteca, agregarLibroTBiblioteca, agregarSocioTBiblioteca, imprimirSociosTBiblioteca y imprimirLibrosTBiblioteca. Verifique el funcionamiento ejecutando el test biblioteca1-crear-cargar-liberar. Foro de dudas.
2. **Implemente** las funciones prestarLibroTBiblioteca, disponibleLibroTBiblioteca y imprimirPrestamosTBiblioteca. Verifique el funcionamiento ejecutando el test biblioteca2-prestar-disponibilidad. Foro de dudas.
3. **Implemente** las funciones imprimirReservasTBiblioteca y devolverLibroTBiblioteca. Verifique el funcionamiento ejecutando el test biblioteca3-devolucion. Foro de dudas.
4. **Implemente** las funciones agregarGeneroABiblioteca y obtenerLibrosDeGenero. Verifique el funcionamiento ejecutando el test biblioteca4-obtener-libros-genero. Foro de dudas.

8. 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 --
   1111111111111111
   ```

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

   ```
   reserva1-crear-eliminar
   reserva2-socio-libro-copiar
   agGeneros1-crear-liberar
   agGeneros2-insertar-imprimir
   agGeneros3-existe-altura-cantidad
   agGeneros4-remover
   agGeneros5-max-subarbol
   agGeneros6-combinado
   colaReserva1-crear-liberar
   colaReserva2-encolar-imprimir
   colaReserva3-cantidad-frente-desencolar
   colaReserva4-combinado
   conjuntoGeneros1-crear-insertar-imprimir-liberar
   conjuntoGeneros2-esvacio-cardinal-cantmax
   conjuntoGeneros3-pertenece-borrar
   conjuntoGeneros4-union-interseccion-diferencia
   conjuntoGeneros5-combinado
   conjuntoGeneros6-tiempo
   biblioteca1-crear-cargar-liberar
   biblioteca2-prestar-disponibilidad
   biblioteca3-devolucion
   biblioteca4-obtener-libros-genero
   ```

   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 `EntregaTarea3.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 **EntregaTarea3.tar.gz**, que contiene los nuevos módulos a implementar **agGeneros.cpp**, **colaReservas.cpp**, **reserva.cpp** y **biblioteca.cpp**, los módulos modificados **IdePrestamos.cpp** y **prestamo.cpp**, además de los módulos implementados en la tarea 1 y 2. 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.