Tarea 2
Listas y árboles
Curso 2025
Segundo semestre

1. Introducción

Esta tarea tiene como principales objetivos el trabajar:

- sobre el manejo dinámico de memoria.
- listas simplemente y doblemente enlazadas.
- árboles binarios de búsqueda.

La fecha límite de entrega es el **miércoles 17 de septiembre 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. Realidad de la tarea

Un grupo de estudiantes de Computación está a cargo del desarrollo de un prototipo de sistema de gestión para la Biblioteca Nacional para el control de usuarios (socios), libros y préstamos. La tarea consiste en la implementación del prototipo de dicho software para que permita mejorar su gestión.

La Biblioteca necesita llevar el control de los libros (TLibro y TABBLibros) que gestiona, las personas socias (Tsocio) que han registrado para retirar libros (TLSESocios) en préstamo (Tprestamo), el registro con la información de los libros que se tiene en préstamo (TLDEPrestamos) y la colección de libros de la biblioteca (TABBLibros).

Descripción general

La Biblioteca Nacional trabaja con libros (TLibro), de los que interesa almacenar su identificador único (ISBN), su título, el nombre y apellido del autor y la fecha de edición.

En relación a los socios de la biblioteca (TSocio) se conoce el nombre, apellido, el nro de CI, la fecha de alta como socio y una colección de elementos que representa sus géneros de lectura favoritos. Los géneros favoritos se registran el momento en que se afilia la persona a la biblioteca a partir de una lista acotada, identificados por un entero. La información de los socios se almacena en una lista simplemente enlazada (TLSESocios), ordenada de menor a mayor por la fecha de alta, para facilitar su gestión.

Cuando se presta un libro se registra el socio (TSocio) que lo retira, el libro (TLibro) que se presta, la fecha de retiro y de devolución. Esta información se almacena en una lista doblemente enlazada (TLDEPrestamos), ordenada por la fecha de préstamo (retiro), para así facilitar la consulta de los préstamos de libros realizados.

La colección de libros gestionados por La Biblioteca se almacena en un árbol binario de búsqueda de elementos de tipo TLibro ordenado por ISBN (TABBLibros).
En las siguientes secciones se describen los distintos módulos y funciones que se solicita implementar: `socio.cpp` (3), `prestamo.cpp`(4), `lseSocios.cpp` (5), `ldePrestamos.cpp` (6) y `abbLibros.cpp` (7). **Tener en cuenta que además deben incluir en el directorio** `src` **los archivos** `fecha.cpp` **y** `libro.cpp` **implementados en la tarea 1.**

3. Módulo socio

En esta sección se describe la implementación del módulo *socio.cpp*. Cada elemento del tipo TSocio almacenará un *identificador* (CI), el *nombre*, *apellido*, la *fecha de alta* como socio y un arreglo con tope de *géneros* favoritos de lectura. Los géneros están acotados por MAX_GENEROS_FAVORITOS.

1. **Implemente** la representación de socio *rep_socio* y las funciones crearTSocio, imprimirTSocio, liberarTSocio, ciTSocio, nombreTSocio, apellidoTSocio y fechaAltaTSocio. Tenga en cuenta que el formato de impresión se especifica en *socio.h*. Ejecute el caso de prueba socio1-crear-imprimir-liberar para verificar el funcionamiento de las operaciones. Foro de dudas.

2. **Implemente** las funciones agregarGeneroFavoritoTSocio, tieneGeneroFavoritoTSocio, y cantidadGenerosFavoritosTSocio. **Ejecute** el test socio2-crear-imprimir-genero-liberar para verificar las funciones. Foro de dudas.

3. **Implemente** la función copiarTSocio. **Ejecute** los tests socio3-crear-imprimir-copiar-liberar y socio4-crear-imprimir-genero-copiar-liberar para verificar la función. Foro de dudas.

4. Módulo prestamo

En esta sección se describe la implementación del módulo *prestamo.cpp*. Cada elemento del tipo TPrestamo representa el préstamo de un libro a un socio de la biblioteca. Cada nodo contiene un elemento del tipo *socio* (*TSocio*) que lo solicita, el *libro* (*TLibro*) que se da en préstamo, la *fecha de retiro* y su *fecha de devolución* (ambos del tipo *Tfecha*).

1. **Implemente** la representación de prestamo *rep_prestamo* y las funciones crearTPrestamo, imprimirTPrestamo, liberarTPrestamo, socioTPrestamo, fechaRetiroTPrestamo, fechaDevolucionTPrestamo y libroTPrestamo. Tenga en cuenta que el formato de impresión se especifica en *prestamo.h*. Ejecute el caso de prueba prestamo1-crear-imprimir-liberar para verificar el funcionamiento de las operaciones. Foro de dudas.

2. **Implemente** la función copiarTPrestamo. **Ejecute** el test prestamo2-copiar para verificar la función. Foro de dudas.

3. **Implemente** las funciones fueRetornadoTPrestamo y actualizarFechaDevolucionTPrestamo. **Ejecute** el tests prestamo3-actualizar para verificar las funciones. Foro de dudas.

5. Lista simplemente enlazada: módulo lseSocios

> Recomendamos que antes de comenzar con la implementación de este módulo, estudie los contenidos asociados a la estructura *Lista simplemente enlazada* en la sección Estructuras lineales de memoria dinámica del eva. La clase de práctico en la que se trabajará sobre listas simplemente enlazadas será la semana del **25 de agosto**.

En esta sección se implementará el módulo *lseSocios.cpp*. Este representa una lista de personas que se registraron como socios de la biblioteca para poder solicitar libros en préstamo. La estructura de tipo TLSESocios almacenará elementos del tipo TSocio, implementada como una **lista simplemente encadenada**, ordenada por fecha de alta (de menor a mayor). Esta estructura lineal es dinámica y permitirá almacenar una cantidad no acotada de socios.

1. **Implemente** la representación de la lista simplemente enlazada *rep_lsesocios*. La representación debe almacenar a un socio y un puntero al siguiente nodo. Foro de dudas.

2. **Implemente** las funciones crearTLSESociosVacia, esVaciaTLSESocios, imprimirTLSESocios y liberarTLSESocios. Recomendamos que la representación de la lista vacía sea simplemente un *puntero a NULL*. **Ejecute** el test lseSocios1-crear-esVacia-imprimir-liberar para verificar el funcionamiento de las funciones. Foro de dudas.

3. **Implemente** la función insertarTLSESocios. Recuerde que las socios se mantienen ordenados de menor a mayor por fecha de alta, y si existen otros socios en la misma fecha, la que se está ingresando queda después. **Ejecute** los tests lseSocios2-crear-insertar-imprimir-liberar y lseSocios3-crear-insertar-imprimir-liberar para verificar el funcionamiento de la función. Foro de dudas.

4. **Implemente** la función existeSocioTLSESocios. **Ejecute** el test lseSocios4-esVacia-existe para verificar el funcionamiento de la función. Foro de dudas.

5. **Implemente** las funciones obtenerSocioTLSESocios, cantidadTLSESocios y obtenerNesimoTLSESocios. **Ejecute** el test lseSocios5-cantidad-obtenerSocio-obtenerNesimo para verificar el funcionamiento de las funciones. Foro de dudas.

6. **Implemente** la función removerSocioTLSESocios. **Ejecute** el test lseSocios6-remover para verificar el funcionamiento de la función. Foro de dudas.

7. **Ejecute** el test lseSocios7-combinado para verificar el funcionamiento conjunto de todas las operaciones implementadas. Foro de dudas.

6. Lista doblemente enlazada: módulo ldePrestamos

Recomendamos que antes de comenzar con la implementación de este módulo, estudie los contenidos asociados a la estructura *Lista doblemente enlazada* en la sección Estructuras lineales de memoria dinámica del eva. La clase de práctico en la que se trabajará sobre listas doblemente enlazadas será la semana del **25 de agosto**.

En esta sección se implementará el módulo *ldePrestamos.cpp*. Este representa una lista doblemente enlazada de prestamos de libros. La estructura de tipo TLDEPrestamos almacenará elementos del tipo TPrestamo y estará implementada como una **lista doblemente encadenada**. Además, se contará con acceso directo (puntero) al inicio y al final de la lista. Esta estructura lineal es dinámica y permitirá almacenar una cantidad no acotada de prestamos. La lista **estará ordenada** según la fecha de préstamo (retiro) (de menor a mayor).

1. **Implemente** la estructura *rep_tldePrestamos* que permita almacenar una lista doblemente enlazada. Para poder cumplir con los órdenes de tiempo de ejecución de las operaciones, recomendamos que la representación sea mediante un **cabezal** con un puntero al nodo *inicial* y otro al nodo *final*. En este sentido, se debe definir además una **representación auxiliar** para los nodos de la lista doblemente enlazada, que tengan un elemento *TPrestamo*, un puntero a un nodo *siguiente* y uno a un nodo *anterior*. Foro de dudas.

2. **Implemente** las funciones crearTLDEPrestamosVacia, insertarTLDEPrestamos y liberarTLDEPrestamos. Verifique el funcionamiento de las funciones ejecutando el test LDEPrestamos1-crear-liberar. Foro de dudas.

3. **Implemente** las funciones imprimirTLDEPrestamos y imprimirInvertidoTLDEPrestamos. **Ejecute** los tests LDEPrestamos2-crear-insertar-imprimir-liberar y LDEPrestamos3-crear-insertar-imprimir-liberar para verificar el funcionamiento de las funciones. Foro de dudas.

4. **Implemente** las funciones cantidadTLDEPrestamos. **Ejecute** el test LDEPrestamos4-crear-insertar-cantidad-liberar para verificar el funcionamiento de la función. Foro de dudas.

5. **Implemente** las funciones obtenerPrimeroTLDEPrestamos, obtenerUltimoTLDEPrestamos y obtenerNesimoTLDEPrestamos. **Ejecute** el test LDEPrestamos5-crear-insertar-obtener-liberar para verificar el funcionamiento de las funciones. Foro de dudas.

6. **Implemente** la función filtrarPrestamosTLDEPrestamos. **Ejecute** el test LDEPrestamos6-crear-insertar-obtener-filtrar-liberar para verificar el funcionamiento de la función. Foro de dudas.

7. **Ejecute** el test LDEPrestamos7-combinado. Foro de dudas.

7. Árbol binario de búsqueda: módulo abbLibros

Recomendamos que antes de comenzar con la implementación de este módulo, estudie los contenidos asociados a la estructura *Árbol Binario de Búsqueda* en la sección Estructuras arborecentes de memoria dinámica del eva. Las clases de práctico en las que se trabajará sobre estructuras árbolescentes de memoria dinámica será a partir de la semana del **1ero de septiembre**.

En esta sección se implementará el módulo *abbLibros.cpp*. La estructura de tipo TABBLibros almacenará elementos del tipo TLibro y estará implementada como un **árbol binario de búsqueda (ABB)**, **ordenado por número ISBN del libro**. La estructura **no aceptará números ISBN repetidos** (se puede asumir que nunca se agregarán repetidos). Se recomienda que las implementaciones sean recursivas.

1. **Implemente** la representación del árbol binario de búsqueda *rep_abbLibros*. La representación debe tener un elemento del tipo *TLibro* y un puntero a un nodo *izquierdo* y a otro nodo *derecho*. Foro de dudas.

2. **Implemente** las funciones crearTABBLibrosVacio, insertarLibroTABBLibros, imprimirTABBLibros en orden ascendente según el número ISBN y liberarTABBLibros. Recomendamos que el árbol vacío se represente mediante un *puntero a NULL*. **Ejecute** los tests abbLibros1-crear-insertar-imprimir-liberar y abbLibros2-crear-insertar-imprimir-liberar para verificar el funcionamiento de las funciones. Foro de dudas.

3. **Implemente** las funciones existeLibroTABBLibros y obtenerLibroTABBLibros. **Ejecute** el test abbLibros3-existe-obtener para verificar el funcionamiento de las funciones. Foro de dudas.

4. **Implemente** la función alturaTABBLibros. **Ejecute** el test abbLibros4-altura para verificar el funcionamiento de la función. Foro de dudas.

5. **Implemente** las funciones maxISBNLibroTABBLibros y removerLibroTABBLibros. **Ejecute** el test abbLibros5-maxISBN-remover para verificar el funcionamiento de las funciones. Foro de dudas.

6. **Implemente** la función cantidadTABBLibros. **Ejecute** el test abbLibros6-cantidad para verificar el funcionamiento de la función. Foro de dudas.

7. **Implemente** la función obtenerNesimoLibroTABBLibros, que obtiene la libro nro. N de un árbol, considerando el orden de ISBN. La descripción detallada se encuentra en abbLibros.h. **Ejecute** el test abbLibros7-obtenerNesimo. Foro de dudas.

8. **Implemente** la función filtradoPorGeneroTABBLibros. La descripción detallada se encuentra en abbLibros.h. **Ejecute** el test abbLibros8-filtrado para verificar el funcionamiento de la función. Foro de dudas.

9. **Ejecute** el test abbLibros9-combinado. Foro de dudas.

10. **Ejecute** el test abbLibros10-tiempo. Este test se debe ejecutar sin valgrind. Para hacerlo, ejecute make tt-abbLibros10-tiempo. Foro de dudas.

11. **Ejecute** el test abbLibros11-tiempo. Este test se debe ejecutar sin valgrind. Para hacerlo, ejecute make tt-abbLibros11-tiempo. 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 --
   1111111111111111111111111111111-
   ```

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

   ```
   fecha5-combinado \
   socio1-crear-imprimir-liberar \
   socio2-crear-imprimir-genero-liberar \
   socio3-crear-imprimir-copiar-liberar \
   socio4-crear-imprimir-genero-copiar-liberar \
   lseSocios1-crear-esVacia-imprimir-liberar \
   lseSocios2-crear-insertar-imprimir-liberar \
   lseSocios3-crear-insertar-imprimir-liberar \
   lseSocios4-esVacia-existe \
   lseSocios5-cantidad-obtenerSocio-obtenerNesimo \
   lseSocios6-remover \
   lseSocios7-combinado \
   prestamo1-crear-imprimir-liberar \
   prestamo2-copiar \
   prestamo3-actualizar \
   LDEPrestamos1-crear-liberar \
   LDEPrestamos2-crear-insertar-imprimir-liberar \
   LDEPrestamos3-crear-insertar-imprimir-liberar \
   LDEPrestamos4-crear-insertar-cantidad-liberar \
   LDEPrestamos5-crear-insertar-obtener-liberar \
   LDEPrestamos6-crear-insertar-obtener-filtrar-liberar \
   LDEPrestamos7-combinado \
   abbLibros1-crear-insertar-imprimir-liberar \
   abbLibros2-crear-insertar-imprimir-liberar \
   abbLibros3-existe-obtener \
   abbLibros4-altura \
   abbLibros5-maxISBN-remover \
   abbLibros6-cantidad \
   abbLibros7-obtenerNesimo \
   abbLibros8-filtrado \
   abbLibros9-combinado \
   abbLibros10-tiempo \
   abbLibros11-tiempo \
   ```

   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 `EntregaTarea2.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 **EntregaTarea2.tar.gz**, que contiene los nuevos módulos a implementar **libro.cpp**, **IseSocios.cpp**, **IdePrestamos.cpp** y **abbLibros.cpp**, además de los módulos **libro.cpp** y **fecha.cpp** implementados en la tarea 1. 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.