Tarea 2
Listas y árboles
Curso 2025

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 23 de abril a las 16:00 horas**. El mecanismo específico de entrega se explica en la Sección 7. 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 tiene como cometido desarrollar un prototipo de un sistema de gestión para la fundación **FINGHogar**, dedicada al rescate, cuidado y adopción de perros abandonados. La fundación busca sistematizar sus procesos para mejorar la eficiencia en la colocación de los animales en hogares adecuados y hacer seguimiento de las adopciones realizadas.

La fundación necesita llevar el control de las personas interesadas en adoptar (TPersona), los perros disponibles para adopción (TPerro), y el registro de las adopciones realizadas.

2.1. Descripción general

La fundación **FINGHogar** trabaja con perros rescatados, de los que interesa conocer su identificador único, su nombre, su edad, su estado vital, una descripción y su fecha de ingreso al refugio.

Los perros disponibles para adopción se organizan en una lista doblemente enlazada (5) para facilitar su gestión, ordenados según su edad. Esta estructura permite mantener un registro actualizado de los animales que ingresan y salen del refugio a través de las adopciones.

Además, se conoce la información de las personas (3) interesadas en adoptar, de las cuales se sabe su cédula de identidad, nombre, apellido, fecha de nacimiento y sus perros.

Cuando una persona decide adoptar un perro, se registra esta información en una lista simplemente enlazada de adopciones (4), que mantiene un histórico cronológico de las adopciones realizadas, incluyendo la fecha en que se concretó cada adopción.

Para facilitar la búsqueda de información sobre los adoptantes, la fundación implementa un árbol binario de búsqueda de personas (6), ordenado por el número de cédula de identidad. Esta estructura permite localizar rápidamente la información de cualquier persona registrada en el sistema.

En las siguientes secciones se describen los distintos módulos y funciones que se solicita implementar: `persona.cpp`, `lseAdopciones.cpp`, `ldePerros.cpp` y `abbPersonas.cpp`. **Tener en cuenta que además deben incluir en el directorio** `src` **los archivos** `fecha.cpp` **y** `perro.cpp` **implementados en la tarea 1.**

3. Módulo persona

En esta sección se describe la implementación del módulo *persona.cpp*. Cada elemento del tipo TPersona almacenará un *identificador*, un *nombre*, un *apellido*, su *edad* y un arreglo con tope de *perros* (los que lleva adoptados).

1. **Implemente** la representación de persona *rep_persona* y las funciones crearTPersona, imprimirTPersona, liberarTPersona, ciTPersona, nombreTPersona, apellidoTPersona y fechaNacimientoTPersona. Tenga en cuenta que el formato de impresión se especifica en *persona.h*. Ejecute el caso de prueba persona1-crear-imprimir-liberar para verificar el funcionamiento de las operaciones. Foro de dudas.

2. **Implemente** las funciones agregarPerroTPersona, pertenecePerroTPersona, y cantidadPerrosTPersona. **Ejecute** el test persona2-crear-imprimir-perro-liberar para verificar las funciones. Foro de dudas.

3. **Implemente** la función copiarTPersona. **Ejecute** los tests persona3-crear-imprimir-copiar-liberar y persona4-crear-imprimir-perro-copiar-liberar para verificar la función. Foro de dudas.

4. Lista simplemente enlazada: módulo lseAdopciones

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 es la del **martes 1ro de abril**.

En esta sección se implementará el módulo *lseAdopciones.cpp*. Este representa una lista con las adopciones de perros. La estructura de tipo TLSEAdopciones almacenará elementos del tipo TPersona, TPerro y TFecha para representar que una persona adoptó un perro en una fecha determinada. Estará implementada como una **lista simplemente encadenada**, ordenada por fecha de adopción (de menor a mayor). Esta estructura lineal es dinámica y permitirá almacenar una cantidad no acotada de adopciones.

1. **Implemente** la representación de la lista simplemente enlazada *rep_lseadopciones*. La representación debe almacenar una persona, un perro, una fecha y un puntero al siguiente nodo. Foro de dudas.

2. **Implemente** las funciones crearTLSEAdopcionesVacia, esVaciaTLSEAdopciones, imprimirTLSEAdopciones y liberarTLSEAdopciones. Recomendamos que la representación de la lista vacía sea simplemente un *puntero a NULL*. **Ejecute** el test lseAdopciones1-crear-esVacia-imprimir-liberar para verificar el funcionamiento de las funciones. Foro de dudas.

3. **Implemente** la función insertarTLSEAdopciones. Recuerde que las adopciones se mantienen ordenadas de menor a mayor por fecha, y si existen otras adopciones en la misma fecha, la que se está ingresando queda después. **Ejecute** los tests lseAdopciones2-crear-insertar-imprimir-liberar y lseAdopciones3-crear-insertar-imprimir-liberar para verificar el funcionamiento de la función. Foro de dudas.

4. **Implemente** la función existeAdopcionTLSEAdopciones. **Ejecute** el test lseAdopciones4-esVacia-existe para verificar el funcionamiento de la función. Foro de dudas.

5. **Implemente** la función removerAdopcionTLSEAdopciones. **Ejecute** el test lseAdopciones5-remover para verificar el funcionamiento de la función. Foro de dudas.

6. **Ejecute** el test lseAdopciones6-combinado para verificar el funcionamiento conjunto de todas las operaciones implementadas. Foro de dudas.

5. Lista doblemente enlazada: módulo ldePerros

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 es la

del **jueves 3 de abril**.

En esta sección se implementará el módulo *ldePerros.cpp*. Este representa una lista doblemente enlazada de perros. La estructura de tipo TLDEPerros almacenará elementos del tipo TPerro 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 perros. La lista **estará ordenada** según la edad de los perros (de menor a mayor).

1. **Implemente** la estructura *rep_tldeperros* 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 *TPerro*, un puntero a un nodo *siguiente* y uno a un nodo *anterior*. Foro de dudas.

2. **Implemente** las funciones crearTLDEPerrosVacia, insertarTLDEPerros y liberarTLDEPerros. Puede resultar útil implementar una **función auxiliar** que permita liberar un nodo individual. Verifique el funcionamiento de las funciones ejecutando el test LDEPerro1-crear-liberar. Foro de dudas.

3. **Implemente** las funciones imprimirTLDEPerros, imprimirInvertidoTLDEPerros. **Ejecute** los tests LDEPerro2-crear-insertar-imprimir-liberar y LDEPerro3-crear-insertar-imprimir-liberar para verificar el funcionamiento de las funciones. Foro de dudas.

4. **Implemente** las funciones cantidadTLDEPerros y removerPerroTLDEPerros. **Ejecute** el test LDEPerro4-crear-insertar-remover-cantidad-liberar para verificar el funcionamiento de la función. Foro de dudas.

5. **Implemente** las funciones obtenerPrimeroTLDEPerros, obtenerUltimoTLDEPerros y obtenerNesimoTLDEPerros. **Ejecute** el test LDEPerro5-crear-insertar-obtener-liberar para verificar el funcionamiento de las funciones. Foro de dudas.

6. **Implemente** la función existePerroTLDEPerros. **Ejecute** el test LDEPerro6-crear-insertar-obtener-existe-liberar para verificar el funcionamiento de la función. Foro de dudas.

7. **Ejecute** el test LDEPerro7-combinado. Foro de dudas.

6. Árbol binario de búsqueda: módulo abbPersonas

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 árboles binarios de búsqueda son la del **martes 8 de abril** (funciones básicas) y la del **jueves 10 de abril** (funciones más complejas).

En esta sección se implementará el módulo *abbPersonas.cpp*. La estructura de tipo TABBPersonas almacenará elementos del tipo TPersona y estará implementada como un **árbol binario de búsqueda (ABB)**, **ordenado por la cédula de identidad (CI) de la persona**. La estructura **no aceptará CIs repetidas** (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_abbPersonas*. La representación debe tener un elemento del tipo *TPersona* y un puntero a un nodo *izquierdo* y a otro nodo *derecho*. Foro de dudas.

2. **Implemente** las funciones crearTABBPersonasVacio, insertarTPersonaTABBPersonas, imprimirTABBPersonas y liberarTABBPersonas. Recomendamos que el árbol vacío se represente mediante un *puntero a NULL*. Por otro lado, recomendamos crear una **función auxiliar** para liberar un nodo individual. **Ejecute** los tests abbPersonas1-crear-insertar-imprimir-liberar y abbPersonas2-crear-insertar-imprimir-liberar para verificar el funcionamiento de las funciones. Foro de dudas.

3. **Implemente** las funciones existeTPersonaTABBPersonas y obtenerTPersonaTABBPersonas. **Ejecute** el test abbPersonas3-existe-obtener para verificar el funcionamiento de las funciones. Foro de dudas.

4. **Implemente** la función alturaTABBPersonas. **Ejecute** el test abbPersonas4-altura para verificar el funcionamiento de la función. Foro de dudas.
5. **Implemente** las funciones maxCITPersonaTABBPersonas y removerTPersonaTABBPersonas. **Ejecute** el test abbPersonas5-maxCI-remover para verificar el funcionamiento de las funciones. Foro de dudas.
6. **Implemente** la función cantidadTABBPersonas. **Ejecute** el test abbPersonas6-cantidad para verificar el funcionamiento de la función. Foro de dudas.
7. **Implemente** la función obtenerNesimaPersonaTABBPersonas, que obtiene la persona nro. N de un árbol, considerando el orden de CI. La descripción detallada se encuentra en abbPersonas.h. **Ejecute** el test abbPersonas7-obtenerNesimo. Foro de dudas.
8. **Implemente** la función filtradoPorFechaDeNacimientoTABBPersonas. La descripción detallada se encuentra en abbPersonas.h. **Ejecute** el test abbPersonas8-filtrado para verificar el funcionamiento de la función. Foro de dudas.
9. **Ejecute** el test abbPersonas9-combinado. Foro de dudas.
10. **Ejecute** el test abbPersonas10-tiempo. Este test se debe ejecutar sin valgrind. Para hacerlo, ejecute make tt-abbPersonas10-tiempo. Foro de dudas.
11. **Ejecute** el test abbPersonas11-tiempo. Este test se debe ejecutar sin valgrind. Para hacerlo, ejecute make tt-abbPersonas11-tiempo. 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 --
1111111111111111111111111
```

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

```
fecha5-combinado
persona1-crear-imprimir-liberar
persona2-crear-imprimir-perro-liberar
persona3-crear-imprimir-copiar-liberar
persona4-crear-imprimir-perro-copiar-liberar
lseAdopciones1-crear-esVacia-imprimir-liberar
lseAdopciones2-crear-insertar-imprimir-liberar
lseAdopciones3-crear-insertar-imprimir-liberar
lseAdopciones4-esVacia-existe
lseAdopciones5-remover
lseAdopciones6-combinado
LDEPerro1-crear-liberar
LDEPerro2-crear-insertar-imprimir-liberar
LDEPerro3-crear-insertar-imprimir-liberar
```

```
LDEPerro4-crear-insertar-remover-cantidad-liberar
LDEPerro5-crear-insertar-obtener-liberar
LDEPerro6-crear-insertar-obtener-existe-liberar
LDEPerro7-combinado
abbPersonas1-crear-insertar-imprimir-liberar
abbPersonas2-crear-insertar-imprimir-liberar
abbPersonas3-existe-obtener
abbPersonas4-altura
abbPersonas5-maxCI-remover
abbPersonas6-cantidad
abbPersonas7-obtenerNesimo
abbPersonas8-filtrado
abbPersonas9-combinado
abbPersonas10-tiempo
abbPersonas11-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 **persona.cpp**, **lseAdopciones.cpp**, **ldePerros.cpp** y **abbPersonas.cpp**, además de los módulos **perro.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.