Tarea 4
TADS: Tabla y Cola de Prioridad
Curso 2025

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 26 de junio a las 16:00 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**.
IMPORTANTE: notar que al igual que en la 3er tarea, los requisitos de tiempo e implementación de los módulos solicitados se encuentran en el archivo *.cpp* correspondiente al módulo a implementar.

2. TAD Tabla: módulo tablaFichaVacunacion

> 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. Las clases de práctico en las que se trabajará sobre el TAD Tabla son las de la semana del **2 de junio**.

En esta sección se implementará el módulo *tablaFichaVacunacion.cpp*. Este módulo se utilizará para almacenar las fichas de vacunación de los perros. Si bien actualmente los IDs de los perros son acotados, para facilitar cambios futuros el módulo se deberá implementar como una **tabla de dispersión abierta**.

1. **Implemente** la estructura *rep_tablaFichaVacunacion*. 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 *TAGFichaVacunacion*. Las estructuras adicionales deben definirse internamente en el módulo. Foro de dudas.
2. **Implemente** las funciones crearTTablaFichaVacunacion y liberarTTablaFichaVacunacion. Ejecute el test tabla1-crear-liberar para verificar el funcionamiento de la funciones. Foro de dudas.
3. **Implemente** las funciones insertarTTablaFichaVacunacion e imprimirTTablaFichaVacunacion. La función de inserción recibe una TTablaFichaVacunacion, un idPerro y un elemento TAGFichaVacunacion. Se asocia en la tabla al TAGFichaVacunacion con el id del perro. Para calcular la posición en la que se debe insertar la ficha en la tabla de dispersión abierta se debe realizar simplemente (idPerro % tamañoTabla) (donde el tamaño de la tabla es un parámetro de la función de crearTTablaFichaVacunacion) La ficha de vacunación debe ser ubicada en la posición calculadada. Por convención, si dicha posición ya se encuentra utilizada se deberá insertar la ficha de vacunación al inicio de la lista definida para dicha posición de la tabla. La función imprimirTTablaFichaVacunacion debe imprimir cada ficha de vacunación de la tabla, en orden creciente de posiciones asociadas en la tabla. En caso de que haya más de un elemento en la misma posición, se deben imprimir en el orden inverso al que fueron agregados (que será el orden natural en que se recorrerá la lista). En el archivo tablaFichaVacunacion.h se encuentra una descripción más detallada del formato de impresión. Ejecute el test tabla2-insertar-imprimir para verificar el funcionamiento de las funciones. Foro de dudas.
4. **Implemente** las funciones perteneceTTablaFichaVacunacion, obtenerFichaTTablaFichaVacunacion y eliminarDeTTablaFichaVacunacion. Ejecute el test tabla3-eliminar-pertenece-obtener. Foro de dudas.
5. **Ejecute** el test tabla4-combinado. Foro de dudas.

3. TAD Cola de Prioridad: módulo colaDePrioridadPerros

Recomendamos que antes de comenzar con la implementación de este módulo, estudie los contenidos asociados al TAD *Cola de Prioridad* en la sección 09 - Colas de prioridad e implementaciones, incluyendo heap del eva. Las clases de práctico en las que se trabajará sobre el TAD Cola de Prioridad son las de la semana del **9 de junio**.

En esta sección se implementará el módulo *colaDePrioridadPerros.cpp*, el cual implementa las funciones del TAD Cola de Prioridad. La estructura de tipo TColaDePrioridadPerros almacenará elementos del tipo TPerro, pero en este caso se implementará una prioridad para obtener eficientemente el perro más prioritario. Esta estructura será utilizada en el módulo *refugio* para obtener eficientemente el perro con menor vitalidad, a fin de ser vacunado. Se establece entonces una prioridad por *vitalidad* de cada perro.
De esta forma, si la estructura no es vacía hay un perro 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 ordenes de tiempo de ejecución de algunas operaciones.

1. **Implemente** la estructura *rep_colaDePrioridadPerros* 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 crearTColaDePrioridadPerros, estaVaciaTColaDePrioridadPerros, insertarTColaDePrioridadPerros, estaTColaDePrioridadPerros y liberarTColaDePrioridadPerros. Verifique el funcionamiento de las funciones ejecutando el test CP1-crear-liberar-vacia-esta-insertar. Foro de dudas.
3. **Implemente** las funciones prioridadTColaDePrioridadPerros y prioritarioTColaDePrioridadPerros. **Ejecute** el test CP2-prioritaria-prioridad-insertar para verificar el funcionamiento de las funciones. Foro de dudas.
4. **Implemente** la función eliminarPrioritarioTColaDePrioridadPerros. **Ejecute** el test CP3-eliminar para verificar el funcionamiento de la función implementada. Foro de dudas.
5. **Implemente** la función invertirPrioridadTColaDePrioridadPerros. **Ejecute** el test CP4-invertirPrioridad para verificar el funcionamiento de la función implementada. Foro de dudas.
6. **Ejecute** el test CP5-tiempo. Recuerde que los test de tiempo se deben verificar sin valgrind, ejecutando *tt-CP5-tiempo* (con doble t). Foro de dudas.
7. **Ejecute** el test CP6-combinado. Foro de dudas.
8. **Ejecute** el test CP7-combinado-invertir. Foro de dudas.

4. Módulo refugio

En esta última sección se implementará el módulo *refugio*, que integrará todas las estructuras creadas hasta el momento y que brindará funciones necesarias para mantener el refugio en funcionamiento.
El refugio mantiene varias estructuras de datos para poder ejecutar sus funciones, un elemento de tipo TRefugio incluye:

- Un elemento de tipo ABBPersonas para registrar las personas asociadas al refugio.
- Un elemento de tipo lseAdopciones para listar todas las adopciones realizadas.
- Un elemento de tipo ldePerros para llevar constancia de los perros que son o fueron parte del refugio.
- Un elemento de tipo agFichaVacunacion para indicar una ficha de vacunación completa para los perros.
- Un elemento de tipo conjuntoPerros, para indicar los perros que fueron adoptados.
- Un elemento de tipo colaPerros, para representar el orden de los paseos.

- Un elemento de tipo tablaFichaVacunacion, para almacenar y acceder a las fichas de vacunación de los perros.

1. **Implemente** la estructura *rep_refugio* de forma que incluya las estructuras descriptas anteriormente. Foro de dudas.

2. **Implemente** las funciones crearTRefugio y liberarTRefugio de forma que inicialicen y liberen las estructuras utilizadas por el refugio. Se define la constante *MAX_PERROS_REFUGIO* para indicar la cantidad máxima de perros en el refugio. A su vez, la función crearTRefugio recibe una cantidad estimada de perros a ser utilizada en la inicialización de la tabla de fichas de vacunación. Verifique el funcionamiento de las funciones ejecutando el test refugio1-crear-liberar. Foro de dudas.

3. **Implemente** las funciones agregarVacunaTRefugio e imprimirEsquemaVacunacionTRefugio, las cuales permiten administrar el esquema de vacunaciones completo para el refugio. **Ejecute** el test refugio2-agregarVacuna-imprimirVacunas para verificar el funcionamiento de las funciones. Foro de dudas.

4. **Implemente** las funciones registrarPersonaTRefugio e imprimirPersonasTRefugio. **Ejecute** el test refugio3-registrarPersona-imprimirPersonas para verificar el funcionamiento de las funciones implementadas. Foro de dudas.

5. En este punto se implementarán las funciones que permiten manejar el ingreso de perros. **Implemente** las funciones ingresarPerroTRefugio, imprimirColaPaseosTRefugio e imprimirPerrosTRefugio. Tenga en cuenta que al ingresar un perro en el refugio, si las estructuras del refugio comparten la memoria de dicho perro, es posible que al liberar el refugio se intente liberar múltiples veces la memoria de la misma variable. Analice el comportamiento de las funciones de inserción invocadas para determinar cuándo se debe realizar copias del perro parámetro, teniendo también en cuenta los requerimientos de la propia función. **Ejecute** el test refugio4-ingresarPerro-imprimirColaPaseos-imprimirPerros para verificar el funcionamiento de las funciones. Foro de dudas.

6. **Implemente** la función pasearPerrosTRefugio, la cual pasea los primeros N perros de la cola de paseos (siendo N parámetro de la función). Ejecute el test refugio5-pasearPerros para verificar el funcionamiento de la función implementada. Foro de dudas.

7. En este punto se implementarán las funciones que permiten manejar la adopción de perros. **Implemente** las funciones adoptarPerroTRefugio e imprimirAdopcionesTRefugio.

   La función de adopción de un perro realiza las siguientes acciones:

   - Inserta una nueva adopción en la lista de adopciones para la fecha, perro y persona.
   - Agrega el perro en los perros adoptados de la persona.
   - Agrega al perro en el conjuntoPerros de perros adoptados.
   - Agrega la asociación idPerro, ciPersona en la tabla de adopciones.
   - Remueve el perro de la cola de paseos, manteniendo el orden de la cola para todos los demás perros.

   Al igual que al agregar perros al refugio, se recomienda prestar particular atención a la memoria utilizada al crear asociaciones para evitar liberar la misma memoria múltiples veces.

   Ejecute el test refugio6-adoptarPerro-imprimirAdopciones para verificar el funcionamiento de las funciones. Notar que el test solo imprime la lista de adopciones, el testing del resto de las acciones de la función adoptarPerroTRefugio debe ser realizado en tests adicionales diseñados por cada estudiante. Los casos privados verificarán el comportamiento completo de la función. Foro de dudas.

8. **Implemente** las funciones vacunarPerroTRefugio e imprimirFichaVacunacionPerroTRefugio. **Ejecute** el test refugio7-vacunarPerro-imprimirVacunasPerro para verificar el funcionamiento de las funciones implementadas. Foro de dudas.

9. **Implemente** la función obtenerPerrosSinVacunacionTRefugio. **Ejecute** el test refugio8-obtenerPerrosSinVacunas para verificar el funcionamiento de dicha función. Foro de dudas.

10. **Ejecute** el test refugio9-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 --
   11111111111111111111111111
   ```

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

   ```
   tabla1-crear-liberar
   tabla2-insertar-imprimir
   tabla3-eliminar-pertenece-obtener
   tabla4-combinado
   CP1-crear-liberar-vacia-esta-insertar
   CP2-prioritaria-prioridad-insertar
   CP3-eliminar
   CP4-invertirPrioridad
   CP5-tiempo
   CP6-combinado
   CP7-combinado-invertir
   refugio1-crear-liberar
   refugio2-agregarVacuna-imprimirVacunas
   refugio3-registrarPersona-imprimirPersonas
   refugio4-ingresarPerro-imprimirColaPaseos-imprimirPerros
   refugio5-pasearPerros
   refugio6-adoptarPerro-imprimirAdopciones
   refugio7-vacunarPerro-imprimirVacunasPerro
   refugio8-obtenerPerrosSinVacunas
   refugio9-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 **tablaFichaVacunacion.cpp**, **colaDePrioridadPerros.cpp** y **refugio.cpp**, además de los módulos **persona.cpp**, **IseAdopciones.cpp**, **abbPersonas.cpp**, **agFichaVacunacion.cpp**, **IdePerros.cpp**, **perro.cpp**, **fecha.cpp**, **conjuntoPerros.cpp**, **colaPerros.cpp**, **pila.cpp** y **aplicaciones.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.