Tarea 3
TADS: Árbol General, Pila, Cola y Conjunto
Curso 2025

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 arbol general.
- Implementar los TADs pila, cola y conjunto.
- Implementar funciones complejas haciendo uso de los TADs implementados.

La fecha límite de entrega es el **miércoles 28 de mayo a las 16:00 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**.
IMPORTANTE: notar que a diferencia de las tareas anteriores, los requisitos de tiempo e implementación de los módulos solicitados se encuentran en el archivo *.cpp* correspondiente al módulo a implementar.

2. Partiendo de la tarea 2

Recuerde colocar sus implementaciones de los modulos fecha, perro, persona, ldePerros, lseAdopciones y abbPersonas en el directorio src. En este caso, notar que se agregaron 2 funciones nuevas al módulo ldePerros que debe implementar.

1. En el módulo *ldePerros*, implemente las funciones adicionales removerPrimeroTLDEPerros y removerUltimoTLDEPerros. Ejecute el test: ldePerros1-rprimero-rultimo para verificar el funcionamiento de las nuevas funciones. Foro de dudas.
2. Ejecute el test ldePerros2-regresion para verificar que las funciones de la tarea anterior continúan funcionando correctamente. Foro de dudas.

3. Árbol General: módulo agFichaVacunación

> 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 *agFichaVacunacion.cpp*, el cual representa una ficha de vacunación. Una vacuna se representa por su código (int). Dado que las vacunas indicadas por el refugio pueden ser aplicadas solo cuando sus predecesoras hayan sido aplicadas, una estructura arborescente permite representarlas adecuadamente. Un elemento de tipo TAGFichaV acunación estará implementado como un **árbol general (AG)**. La estructura **no aceptará vacunas repetidas** (se puede asumir que nunca se agregarán repetidos). Se recomienda que las implementaciones sean recursivas.

1. **Implemente** la representación del árbol general *rep_agFichaVacunacion*. Se sugiere una implementación primer hijo, siguiente hermano. Foro de dudas.

2. **Implemente** las funciones crearTAGFichaVacunacion, insertarVacunaTAGFichaVacunacion, imprimirTAGFichaVacunacion y liberarTAGFichaVacunacion.

   La impresión de la ficha de vacunación se realiza un nodo por línea, dejando espacios a la izquierda según la altura del nodo. Por ejemplo, un árbol dado por un nodo raiz de valor 1, con dos hijos 12 y 11, cada uno con hijos 123, 122, 121 y 113, 112, 111 respectivamente, se vería como se indica a continuación:

```
1
    12
        123
        122
        121
    11
        113
        112
        111
```

   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 agFichaVacunacion1-crear-liberar y agFichaVacunacion2-insertar-imprimir para verificar el funcionamiento de las funciones. Foro de dudas.

3. **Implemente** las funciones existeVacunaTAGFichaVacunacion, alturaTAGFichaVacunacion y cantidadTAGFichaVacunacion. **Ejecute** el test agFichaVacunacion3-existe-altura-cantidad para verificar el funcionamiento de las funciones. Foro de dudas.

4. **Implemente** la función removerVacunaTAGFichaVacunacion. **Ejecute** el test agFichaVacunacion4-remover para verificar el funcionamiento de la función. Foro de dudas.

5. **Implemente** la función igualesTAGFichaVacunacion. **Ejecute** el test agFichaVacunacion5-iguales para verificar el funcionamiento de la función. Foro de dudas.

6. **Ejecute** el test agFichaVacunacion6-combinado. Foro de dudas.

4. TAD Pila: módulo pila

Recomendamos que antes de comenzar con la implementación de este módulo, estudie los contenidos asociados al TAD *Pila* en la sección Introducción a TADs. Lista, Pila y Cola del eva. Las clases de práctico en las que se trabajará sobre el TAD Pila son las de la semana del **5 de mayo**.

En esta sección se implementará el módulo *pila.cpp*, el cual implementa las funciones del TAD Pila. La estructura de tipo TPila almacenará elementos del tipo int. Esta estructura lineal es dinámica y permitirá almacenar una cantidad no acotada de elementos.

1. **Implemente** la estructura *rep_pila* que permita implementar las funciones del módulo en el orden indicado. Se recomienda utilizar estructuras auxiliares. Foro de dudas.

2. **Implemente** las funciones crearTPila y liberarTPila. Verifique el funcionamiento de las funciones ejecutando el test pila1-crear-liberar. Foro de dudas.

3. **Implemente** las funciones apilarTPila, e imprimirTPila. **Ejecute** los tests pila2-apilar-imprimir para verificar el funcionamiento de las funciones. Foro de dudas.

4. **Implemente** las funciones cantidadTPila, cimaTPila y desapilarTPila. **Ejecute** el test pila3-cantidad-cima-desapilar para verificar el funcionamiento de las funciones. Foro de dudas.

5. **Ejecute** el test pila4-combinado. Foro de dudas.

5. TAD Cola: módulo colaPerros

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. Las clases de práctico en las que se trabajará sobre el TAD Cola son las de la semana del **5 de mayo**.

El refugio mantiene un servicio de paseos. Como usualmente hay demasiados perros alojados como para pasearlos a todos juntos, los pasea de a tandas. Para lograr un trato equitativo, la información de los siguientes perros a pasear se almacena en una cola.
En esta sección se implementará el módulo *colaPerros.cpp*, el cual implementa las funciones del TAD Cola. La estructura de tipo TColaPerros almacenará elementos del tipo TPerro. Esta estructura lineal es dinámica y permitirá almacenar una cantidad no acotada de perros.

1. **Implemente** la estructura *rep_colaPerros* que permita implementar las funciones del módulo en el orden indicado. Se recomienda utilizar estructuras auxiliares. Foro de dudas.
2. **Implemente** las funciones crearTColaPerros y liberarTColaPerros. Verifique el funcionamiento de las funciones ejecutando el test colaPerros1-crear-liberar. Foro de dudas.
3. **Implemente** las funciones encolarTColaPerros, e imprimirTColaPerros. **Ejecute** los tests colaPerros2-encolar-imprimir para verificar el funcionamiento de las funciones. Foro de dudas.
4. **Implemente** las funciones cantidadTColaPerros, frenteTColaPerros y desencolarTColaPerros. **Ejecute** el test colaPerros3-cantidad-frente-desencolar para verificar el funcionamiento de las funciones. Foro de dudas.
5. **Ejecute** el test colaPerros4-combinado. Foro de dudas.

6. TAD Conjunto: módulo conjuntoPerros

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. Las clases de práctico en las que se trabajará sobre el TAD Conjunto son las de la semana del **12 de mayo**.

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

1. **Implemente** la estructura *rep_conjuntoperros*, que almacena un conjunto acotado de enteros y que permita satisfacer los órdenes de tiempo de ejecución solicitados en conjuntoPerros.h

   Foro de dudas.
2. **Implemente** las funciones crearTConjuntoPerros, insertarTConjuntoPerros, imprimirTConjuntoPerros y liberarTConjuntoPerros. Verifique el funcionamiento de las funciones ejecutando el test conjuntoPerros1-crear-insertar-imprimir-liberar. Foro de dudas.
3. **Implemente** las funciones esVacioTConjuntoPerros, cardinalTConjuntoPerros y cantMaxTConjuntoPerros. Verifique el funcionamiento de las funciones ejecutando el test conjuntoPerros2-esvacio-cardinal-cantmax. Foro de dudas.
4. **Implemente** las funciones perteneceTConjuntoPerros y borrarDeTConjuntoPerros. **Ejecute** el caso de prueba conjuntoPerros3-pertenece-borrar. Foro de dudas.
5. **Implemente** las funciones unionTConjuntoPerros, interseccionTConjuntoPerros y diferenciaTConjuntoPerros. **Ejecute** el caso de prueba conjuntoPerros4-union-interseccion-diferencia. Foro de dudas.
6. **Ejecute** el caso de prueba conjuntoPerros5-combinado. Foro de dudas.
7. **Ejecute** el caso de prueba conjuntoPerros6-tiempo. Recuerde ejecutarlo con el comando make tt-conjuntoPerros6-tiempo para ejecutar sin valgrind. Foro de dudas.

7. Módulo Aplicaciones

En esta sección se solicitará la implementación de funciones adicionales, ajenas a la realidad el la tarea, que requieren el *uso* de los TADs definidos anteriormente para su resolución.
Las funciones solicitadas en esta sección se encuentran en el módulo aplicaciones (archivo aplicaciones.cpp).

1. **Implemente** la función mismosElementos. Verifique el funcionamiento ejecutando el test aplicaciones1-mismos. Foro de dudas.
2. **Implemente** la función menoresQueElResto. Verifique el funcionamiento ejecutando el test aplicaciones2-menores. Foro de dudas.
3. **Implemente** la función sumaPares. Verifique el funcionamiento ejecutando el test aplicaciones3-suma. 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 --
   11111111111111111111111111
   ```

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

   ```
   ldePerros1-rprimero-rultimo
   ldePerros2-regresion
   agFichaVacunacion1-crear-liberar
   agFichaVacunacion2-insertar-imprimir
   agFichaVacunacion3-existe-altura-cantidad
   agFichaVacunacion4-remover
   agFichaVacunacion5-iguales
   agFichaVacunacion6-combinado
   pila1-crear-liberar
   pila2-apilar-imprimir
   pila3-cantidad-cima-desapilar
   pila4-combinado
   colaPerros1-crear-liberar
   colaPerros2-encolar-imprimir
   colaPerros3-cantidad-frente-desencolar
   colaPerros4-combinado
   conjuntoPerros1-crear-insertar-imprimir-liberar
   conjuntoPerros2-esvacio-cardinal-cantmax
   conjuntoPerros3-pertenece-borrar
   conjuntoPerros4-union-interseccion-diferencia
   conjuntoPerros5-combinado
   conjuntoPerros6-tiempo
   aplicaciones1-mismos
   ```

```
aplicaciones2-menores
aplicaciones3-suma
```

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 **agFichaVacunacion.cpp**, **pila.cpp**, **colaPerros.cpp**, **conjuntoPerros.cpp** y **aplicaciones.cpp**, el módulo modificado **IdePerros.cpp**, además de los módulos **persona.cpp**, **lseAdopciones.cpp**, **abbPersonas.cpp**, **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.