Tarea 3
TADs Cola, Conjunto y uso de TADs
Curso 2024

1. Introducción

Esta tarea tiene como principales objetivos:

- Continuar trabajando sobre el manejo dinámico de memoria.
- Trabajar con el concepto de tipo abstracto de datos (TADs).
- Trabajar en el uso de TADs como auxiliares para la resolución de problemas.

La fecha límite de entrega es el **miércoles 5 de junio 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.

2. Descripción de las funcionalidades a implementar

En esta tarea se trabajará fundamentalmente sobre el concepto de exposición (5). Una exposición ocurre durante un cierto período de tiempo (tiene una fecha de inicio y una de fin) y exhibe algunas de las piezas de la galería.
A su vez se implementará el módulo galería (7), el cual manejará una lista de todas las piezas de arte disponibles, así como la planificación y archivo de exposiciones finalizadas, activas y futuras. Para poder identificar qué exposiciones están finalizadas, cuáles activas y cuáles son futuras, la galería mantiene el valor de la fecha actual.
Finalmente, también se trabajará en el módulo visitaDia (3), el cual representa los diferentes grupos que asisten a la galería en un determinado día.
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**.

3. Módulo visitaDia (TAD Cola)

En esta sección se implementará el módulo *visitaDia.cpp*. Cada elemento del tipo visitaDia almacena una *fecha* y una colección de *visitantes* (una instancia de GrupoABB), sobre los cuales se brindan algunas de las funciones típicas del TAD Cola.

1. **Implemente** la representación de visitaDia *rep_visitadia* y las funciones crearTVisitaDia, encolarGrupoTVisitaDia, imprimirVisitaDia y liberarTVisitaDia. Tenga en cuenta que el formato de impresión se especifica en *visitaDia.h*. Ejecute el caso de prueba visitaDia1-crear-insertar-imprimir-liberar para verificar el funcionamiento de las operaciones. Foro de dudas.
2. **Implemente** las funciones cantidadGruposTVisitaDia y desencolarGrupoTVisitaDia. **Ejecute** el test visitaDia2-cantidad-desencolar para verificar las funciones. Foro de dudas.
3. **Ejecute** el test visitaDia3-combinado Foro de dudas.

4. Módulo conjuntoPiezas (TAD Conjunto)

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

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

   Foro de dudas.

2. **Implemente** las funciones crearTConjuntoPiezas, insertarTConjuntoPiezas, imprimirTConjuntoPiezas y liberarTConjuntoPiezas. Verifique el funcionamiento de las funciones ejecutando el test conjuntoPiezas1-crear-insertar-imprimir-liberar. Foro de dudas.

3. **Implemente** las funciones esVacioTConjuntoPiezas, cardinalTConjuntoPiezas y cantMaxTConjuntoPiezas. Verifique el funcionamiento de las funciones ejecutando el test conjuntoPiezas2-esvacio-cardinal-cantmax. Foro de dudas.

4. **Implemente** las funciones perteneceTConjuntoPiezas y borrarDeTConjuntoPiezas. **Ejecute** el caso de prueba conjuntoPiezas3-pertenece-borrar. Foro de dudas.

5. **Implemente** las funciones unionTConjuntoPiezas, interseccionTConjuntoPiezas y diferenciaTConjuntoPiezas. **Ejecute** el caso de prueba conjuntoPiezas4-union-interseccion-diferencia. Foro de dudas.

6. **Ejecute** el caso de prueba conjuntoPiezas5-combinado. Foro de dudas.

5. Módulo exposición

En esta sección se implementará el módulo *exposicion.cpp*. La estructura almacena un id de tipo entero, dos fechas para representar el inicio y el fin de la exposición, así como la información de qué piezas forman parte de la exposición. Para almacenar dicha información se recomienda utilizar el módulo conjuntoPiezas.

1. **Implemente** la estructura *rep_exposicion* incluyendo los elementos mencionados en el párrafo anterior. Foro de dudas.

2. **Implemente** las funciones crearTExposicion, agregarATExposicion, imprimirTExposicion y liberarTExposicion. Recuerde que en la funcíon liberar también se debe liberar toda la información utilizada por las estructuras auxiliares. **Ejecute** el test exposicion1-crear-agregar-imprimir-liberar para verificar el funcionamiento de las funciones. Foro de dudas.

3. **Implemente** las funciones perteneceATExposicion, idTExposicion, fechaInicioTExposicion y fechaFinTExposicion. **Ejecute** el test exposicion2-pertenece-id-fechaIni-fechaFin para verificar el funcionamiento de las funciones. Foro de dudas.

4. **Implemente** la función sonExposicionesCompatibles. Ejecute el test exposicion3-compatibles.

   Importante: el test es intencionalmente simple. El caso de prueba privado para verificar esta función será mucho más completo y verificará casos de borde, los cuales existen varios. **Debe** verificar esta función con tests adicionales propios. Sugerencia: piense bien todos los posibles casos en el solapamiento de las fechas, considerando también que una exposición puede comenzar antes y terminar después que la segunda. Foro de dudas.

5. **Ejecute** el test exposicion4-combinado. Foro de dudas.

6. Módulo listaExposiciones

En esta sección se implementará el módulo *listaExposiciones.cpp*. Este módulo permite manejar una colección de exposiciones y se proveen la mayoría de las operaciones del TAD Lista. La estructura de tipo TListaExposiciones almacenará elementos del tipo TExposición y se recomienda implementarla como una lista simplemente enlazada. La lista debe estar ordenada por fecha de inicio de la exposición.

1. **Implemente** la estructura *rep_listaexposiciones* que permita implementar las operaciones con los órdenes solicitados. Foro de dudas.

2. **Implemente** las funciones crearTListaExposicionesVacia, agregarExposicionTListaExposiciones, imprimirTListaExposiciones y liberarTListaExposiciones. Verifique el funcionamiento de las funciones ejecutando el test listaExposiciones1-crear-agregar-imprimir-liberar. Foro de dudas.

3. **Implemente** las funciones esVaciaTListaExposiciones, perteneceExposicionTListaExposiciones y obtenerExposicionTListaExposiciones. **Ejecute** el test listaExposiciones2-vacia-pertenece-obtener para verificar el funcionamiento de las funciones. Foro de dudas.

4. **Implemente** las funciones obtenerExposicionesFinalizadas y obtenerExposicionesActivas. **Ejecute** el test listaExposiciones3-finalizadas-activas para verificar el funcionamiento de las funciones. Foro de dudas.

5. **Implemente** las funciones esCompatibleTListaExposiciones y unirListaExposiciones. **Ejecute** el test listaExposiciones4-compatible-unir para verificar el funcionamiento de las funciones. Foro de dudas.

6. **Ejecute** el test listaExposiciones5-combinado. Foro de dudas.

7. Módulo galería

En esta sección se describe la implementación del módulo *galería.cpp*. El módulo galería ofrece las funciones de más alto nivel de la aplicación, es decir, los comandos principales. Para esta tarea, la galería permitirá gestionar las exposiciones y las piezas.
La galería maneja tres listas de exposiciones: pasadas, activas y futuras. Además, almacena la fecha actual (representada con una variable de tipo TFecha), para poder determinar cuáles exposiciones están activas, cuáles pasadas y cuáles futuras.
Finalmente, la galería mantiene una colección de piezas (implementada en la Tarea 2) para registrar todas las piezas disponibles en la galería y que permite determinar cuáles piezas pueden estar en exposición.

1. **Implemente** la representación de pieza *rep_galeria* y la función crearTGaleria. Foro de dudas.

2. **Implemente** las funciones agregarPiezaTGaleria y liberarTGaleria. Ejecute el caso de prueba galeria1-crear-agregarpieza-liberar. Foro de dudas.

3. **Implemente** las funciones agregarExposicionTGaleria y agregarPiezaAExposicionTGaleria. Ejecute el caso de prueba galeria2-agregarexpo-agregarpiezaexpo. Foro de dudas.

4. **Implemente** las funciones imprimirExposicionesFinalizadasTGaleria, imprimirExposicionesActivasTGaleria, e imprimirExposicionesFuturasTGaleria. Ejecute el caso de prueba galeria3-imprimir. Foro de dudas.

5. **Implemente** la función esCompatibleExposicionTGaleria. Ejecute el caso de prueba galeria4-compatible. Foro de dudas.

6. **Implemente** la función avanzarAFechaTGaleria. Recuerde que dicha función modifica las listas de exposiciones según la nueva fecha. Sugerencia: Estudie las funciones provistas por el módulo listaExposiciones. Ejecute el caso de prueba galeria5-avanzarfecha. Foro de dudas.

7. Ejecute el caso de prueba galeria6-combinado. 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 --
11111111111111111111111
```

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

```
visitaDia1-crear-insertar-imprimir-liberar
visitaDia2-cantidad-desencolar
visitaDia3-combinado
conjuntoPiezas1-crear-insertar-imprimir-liberar
conjuntoPiezas2-esvacio-cardinal-cantmax
conjuntoPiezas3-pertenece-borrar
conjuntoPiezas4-union-interseccion-diferencia
conjuntoPiezas5-combinado
exposicion1-crear-agregar-imprimir-liberar
exposicion2-pertenece-id-fechaIni-fechaFin
exposicion3-compatibles
exposicion4-combinado
listaExposiciones1-crear-agregar-imprimir-liberar
listaExposiciones2-vacia-pertenece-obtener
listaExposiciones3-finalizadas-activas
listaExposiciones4-compatible-unir
listaExposiciones5-combinado
galeria1-crear-agregarpieza-liberar
galeria2-agregarexpo-agregarpiezaexpo
galeria3-imprimir
galeria4-compatible
galeria5-avanzarfecha
galeria6-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 `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 módulos a implementar **visitaDia.cpp**, **conjuntoPiezas.cpp**, **exposicion.cpp**, **listaExposiciones.cpp** y **galeria.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.