Tarea 2
Listas y Árboles
Curso 2024

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 24 de abril 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**.

2. Realidad de la tarea

2.1. Introducción

Un grupo de estudiantes del InCo se interesa mucho por el arte y suelen visitar galerías y museos. Han notado que los administradores de una de sus galerías preferidas utilizan demasiado de su tiempo en anotar y manejar información de inventario y visitas en cuadernos, en lugar de dedicarse a tareas que son más de su interés, como conseguir piezas de arte y decidir su ubicación en las distintas salas de la galería. Con esta motivación es que deciden implementar una aplicación para ayudar a los administradores de su galería amiga. Luego de varias entrevistas, los estudiantes llegan a la siguiente descripción breve de la realidad de la galería.

2.2. Descripción general

En la galería se desarrollan exposiciones de piezas de arte de diferentes tipos. Las personas (visitantes), realizan visitas a las exposiciones en grupos.
Las exposiciones tienen un comienzo y un final que se decide al momento de organizar la exposición. Dado el tamaño de la galería, puede haber más de una exposición al mismo tiempo en diferentes salas. Antes de comenzar la exposición, se decide un conjunto de piezas que es con las que se inicia la exposición.
Dado que la galería es muy famosa y se muestran piezas de autores de renombre, para los administradores es importante llevar registro de las personas que los visitan por cuestiones de seguridad y para limitar el aforo. Por esto es que a los visitantes los registran como un grupo de visita y se les asigna una fecha y un horario en el cual pueden recorrer una exposición determinada. En esta tarea se trabajará en la implementación de las piezas, los visitantes y los grupos.
En las exposiciones se presentan piezas (3) de arte de diversos tipos de las cuales se conoce su nombre, autor y fecha de creación. El sistema debe mantener diferentes listas de piezas, para lo cual se implementará una colección de piezas (4).
Las exposiciones reciben visitantes (5), de los cuales se conoce su nombre, apellido, edad y un identificador asociado. Los visitantes están organizados en grupos (6), los cuales se representan como colecciones de visitantes.
Por último, para registrar qué grupos se accederán durante cada día, se implementará una colección de grupos (7). A los administradores les interesa fomentar las visitas entre las nuevas generaciones, por lo cual se le dará prioridad de entrada a la exposición a los grupos con menor edad promedio.
En las siguientes secciones se describen los distintos módulos y funciones que se solicita implementar.

3. Módulo pieza

En esta sección se describe la implementación del módulo *pieza.cpp*. Cada elemento del tipo TPieza almacenará un *identificador*, un *nombre* para la pieza, el nombre y apellido del autor y una fecha de creación.

1. **Implemente** la representación de pieza *rep_pieza* y las funciones crearTPieza, idTPieza, imprimirTPieza y liberarTPieza. Tenga en cuenta que el formato de impresión se especifica en *pieza.h*. Ejecute el caso de prueba pieza1-crear-liberar y pieza2-crear-imprimir-liberar para verificar el funcionamiento de las operaciones. Foro de dudas.

4. Lista simple: colección de piezas

En esta sección se describe la implementación del módulo *coleccionPiezas.cpp*. La colección de piezas estará implementada mediante una *lista simplemente enlazada*, ordenada por el id de las piezas. Esta estructura lineal es dinámica y permitirá almacenar una cantidad no acotada de piezas de una colección. A la hora de implementar listas hay funciones que se pueden hacer de forma iterativa o recursiva. Recomendamos que las implementaciones sean **iterativas** ya que las recursivas se recomendaran fuertemente en otro módulo.

1. **Implemente** la representación de la lista simplemente enlazada rep_coleccionpiezas. La representación debe almacenar una pieza y un puntero al siguiente nodo, por ejemplo, de la siguiente manera:

```
TPieza pieza;
rep_coleccionpiezas * sig;
```

   Observe que en *coleccionPiezas.h* se encuentra declarado el tipo TColeccionPiezas de la siguiente manera:

```
typedef struct rep_coleccionpiezas *TColeccionPiezas;
```

   Es decir, *TColeccionPiezas* es un *alias* de *rep_coleccionpiezas* * (son equivalentes). Por lo tanto, también se puede definir *rep_coleccionpiezas* de la siguiente manera:

```
TPieza pieza;
TColeccionPiezas sig;
```

   Se debe tener en cuenta que esto es posible ya que el tipo de TColeccionPiezas fue definido antes (en el .h) que rep_coleccionpiezas. Foro de dudas.

2. **Implemente** las funciones crearColeccionPiezasVacia, insertarPiezaColeccionPiezas, imprimirColeccionPiezas y liberarColeccionPiezas. Recomendamos que la representación de la lista vacía sea simplemente un *puntero a NULL*. Recuerde que la colección de piezas mantiene las piezas de forma ordenada de **menor a mayor** por el id de la pieza, sabiendo que estos id's son únicos. Por otro lado, el formato en el que se debe imprimir la colección de piezas es simplemente utilizando de forma secuencial la función *imprimirTPieza*. **Ejecute** el caso de prueba coleccionPiezas1-crear-insertar-imprimir-liberar. Foro de dudas.

3. **Implemente** las funciones esVaciaColeccionPiezas, existePiezaColeccionPiezas y obtenerPiezaColeccionPiezas. **Ejecute** el caso de prueba coleccionPiezas2-vacia-existe-obtener. Foro de dudas.

4. **Implemente** la función removerPiezaColeccionPiezas y **ejecute** el caso de prueba coleccionPiezas3-remover. Foro de dudas.

5. **Ejecute** el caso de prueba coleccionPiezas4-combinado. Foro de dudas.

5. Módulo visitante

En esta sección se implementará el módulo *visitante.cpp*. Cada elemento del tipo visitante almacenará un *id*, la *edad*, su *nombre* y *apellido*.

1. **Implemente** la representación de visitante *rep_visitante* y las funciones crearTVisitante, imprimirTVisitante y liberarTVisitante. Tenga en cuenta que el formato de impresión se especifica en *visitante.h*. Ejecute el caso de prueba visitante1-crear-imprimir-liberar para verificar el funcionamiento de las operaciones. Foro de dudas.

2. **Implemente** las funciones idTVisitante, nombreTVisitante, apellidoTVisitante y edadTVisitante. **Ejecute** el test visitante2-id-nombre-apellido-edad para verificar las funciones. Foro de dudas.

3. **Implemente** la función copiarTVisitante. **Ejecute** el test visitante3-copiar para verificar la función. **Ejecute** el test visitante4-combinado Foro de dudas.

6. Árbol binario de búsqueda: módulo grupoABB

En esta sección se implementará el módulo *grupoABB.cpp*. La estructura de tipo TGrupoABB almacenará elementos del tipo Tvisitante y estará implementada como un **árbol binario de búsqueda (ABB)**, **ordenado por el índice del visitante**. La estructura **no aceptará índices repetidos** (se puede asumir que nunca se agregarán repetidos).

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

2. **Implemente** las funciones crearTGrupoABBVacio, insertarTVisitanteTGrupoABB, imprimirTGrupoABB y liberarTGrupoABB. 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** el test grupoABB1-crear-insertar-imprimir-liberar para verificar el funcionamiento de las funciones. Foro de dudas.

3. **Implemente** las funciones existeTVisitanteTGrupoABB y obtenerTVisitanteTGrupoABB. **Ejecute** el test grupoABB2-existe-obtener para verificar el funcionamiento de las funciones. Foro de dudas.

4. **Implemente** la función alturaTGrupoABB. **Ejecute** el test grupoABB3-altura para verificar el funcionamiento de la función. Foro de dudas.

5. **Implemente** las funciones maxIdTVisitanteTGrupoABB y removerTVisitanteTGrupoABB. **Ejecute** el test grupoABB4-maxid-remover para verificar el funcionamiento de las funciones. Foro de dudas.

6. **Implemente** las funciones cantidadVisitantesTGrupoABB y edadPromedioTGrupoABB. **Ejecute** el test grupoABB5-cantidad-edadPromedio para verificar el funcionamiento de las funciones. Foro de dudas.

7. **Implemente** la función obtenerNesimoVisitanteGrupoABB, que obtiene el visitante nro N de un grupo, considerando el orden de id. La descripción detallada se encuentra en grupo.h. **Ejecute** el test grupoABB6-obtenerNesimo. Foro de dudas.

8. **Ejecute** el test grupoABB7-combinado. Foro de dudas.

7. Lista doblemente enlazada: módulo coleccionGrupos

En esta sección se implementará el módulo *coleccionGrupos.cpp*. El mismo representa una lista de grupos. La estructura de tipo TColeccionGrupos almacenará elementos del tipo TGrupoABB 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 grupos. La lista **estará ordenada** según la edad promedio del grupo (de menor a mayor).

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

2. **Implemente** las funciones crearTColeccionTGruposVacia, insertarGrupoTColeccionTGrupos y liberarTColeccionTGrupos. Puede resultar útil implementar una **función auxiliar** que permita liberar un nodo individual. Verifique el funcionamiento de las funciones ejecutando el test coleccionTGrupos1-crear-insertar-liberar. Foro de dudas.
3. **Implemente** las funciones imprimirTColeccionTGrupos y imprimirInvertidoTColeccionTGrupos. **Ejecute** el test coleccionTGrupos2-imprimir para verificar el funcionamiento de las funciones. Foro de dudas.
4. **Implemente** las funciones cantidadTGruposColeccionTGrupos, obtenerPrimeroColeccionTGrupos y obtenerNesimoColeccionTGrupos. **Ejecute** el test coleccionTGrupos3-cantidad-primero-enesimo para verificar el funcionamiento de las funciones. Foro de dudas.
5. **Implemente** las funciones removerUltimoColeccionTGrupos y removerNesimoColeccionTGrupos. **Ejecute** el test coleccionTGrupos4-removerultimo-removerenesimo para verificar el funcionamiento de las funciones. Foro de dudas.
6. **Implemente** las funciones visitanteMasRepetido y obtenerVisitantesRepetidos. **Ejecute** el test coleccionTGrupos5-repetido para verificar el funcionamiento de las funciones. Foro de dudas.
7. **Ejecute** el test coleccionTGrupos6-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 --
   11111111111111111111
   ```

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

   ```
   pieza1-crear-liberar
   pieza2-crear-imprimir-liberar
   coleccionPiezas1-crear-insertar-imprimir-liberar
   coleccionPiezas2-vacia-existe-obtener
   coleccionPiezas3-remover
   coleccionPiezas4-combinado
   visitante1-crear-imprimir-liberar
   visitante2-id-nombre-apellido-edad
   visitante3-copiar
   visitante4-combinado
   grupoABB1-crear-insertar-imprimir
   grupoABB2-existe-obtener
   grupoABB3-altura-liberar
   grupoABB4-maxid-remover
   grupoABB5-cantidad-edadPromedio
   grupoABB6-obtenerNesimo
   grupoABB7-combinado
   coleccionTGrupos1-crear-insertar-liberar
   coleccionTGrupos2-imprimir
   ```

```
coleccionTGrupos3-cantidad-primero-enesimo
coleccionTGrupos4-removerultimo-removerenesimo
coleccionTGrupos5-repetidos
coleccionTGrupos6-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 `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 módulos a implementar **pieza.cpp**, **coleccionPiezas.cpp**, **visitante.cpp**, **grupoABB.cpp** y **coleccionTGrupos.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.