Tarea 3
TAD conjunto y uso de TAD
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 (TAD).
- Trabajar en el uso de TAD como auxiliares para la resolución de problemas.

La fecha límite de entrega es el **miércoles 23 de octubre 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.
Recuerde **descargar los materiales de la tarea del EVA y descomprimirlos en la carpeta de desarrollo**. Los materiales para realizar cada tarea se extraen de un archivo específico para cada tarea que se encuentra en la sección Laboratorio, Materiales y entrega del sitio del curso.
Para desempaquetar el material se puede usar la utilidad *tar* desde la línea de comandos:

```
$ tar zxvf NombreArchivo.tar.gz
```

2. Descripción de las funcionalidades a implementar

En esta tarea se trabajará fundamentalmente sobre el TAD conjunto, que se utilizará para implementar un conjunto de productos (3), y sobre el concepto de promoción (4). Una promoción de productos tiene validez durante un cierto período de tiempo, esto es, tiene una fecha de inicio y una de fin y comprende algunos de los productos de la cadena de tiendas **Mercado FINGer**.
A su vez, se implementará el módulo historial (6), el cual manejará la planificación y el archivo de las promociones finalizadas, activas y futuras. Para poder identificar qué promociones están finalizadas, cuáles activas y cuáles son futuras, el historial mantiene el valor de la fecha actual.
Finalmente, 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 conjuntoProductos (TAD Conjunto)

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

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

   Foro de dudas.

2. **Implemente** las funciones crearTConjuntoProductos, insertarTConjuntoProductos, imprimirTConjuntoProductos y liberarTConjuntoProducts. Verifique el funcionamiento de las funciones ejecutando el test conjuntoProductos1-crear-insertar-imprimir-liberar. Foro de dudas.

3. **Implemente** las funciones esVacioTConjuntoProductos, cantidadTConjuntoProductos y cantMaxTConjuntoProductos. Verifique el funcionamiento de las funciones ejecutando el test conjuntoProductos2-esvacio-cardinal-cantmax. Foro de dudas.

4. **Implemente** las funciones perteneceTConjuntoProductos y borrarDeTConjuntoProductoss. **Ejecute** el caso de prueba conjuntoProductos3-pertenece-borrar. Foro de dudas.

5. **Implemente** las funciones unionTConjuntoProductos, interseccionTConjuntoProductos y diferenciaTConjuntoProductos. **Ejecute** el caso de prueba conjuntoProductos4-union-interseccion-diferencia. Foro de dudas.

6. **Ejecute** el caso de prueba conjuntoProductos5-combinado. Foro de dudas.

4. Módulo promoción

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

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

2. **Implemente** las funciones crearTPromocion, agregarATPromocion, imprimirTPromocion y liberarTPromocion. Recuerde que en la funcíon liberar también se debe liberar toda la información utilizada por las estructuras auxiliares. **Ejecute** el test promocion1-crear-agregar-imprimir-liberar para verificar el funcionamiento de las funciones. Foro de dudas.

3. **Implemente** las funciones perteneceATPromocion, idTPromocion, fechaInicioTPromocion y fechaFinTPromocion. **Ejecute** el test promocion2-pertenece-id-fechaIni-fechaFin para verificar el funcionamiento de las funciones. Foro de dudas.

4. **Implemente** la función sonPromocionesCompatibles.
   Una promocion es compatible con otra si no comparten productos durante un mismo período de tiempo. O de otra manera, no son compatibles si coinciden en el tiempo y además un mismo producto es parte de ambas promociones. Ejecute el test promocion3-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 promoción puede comenzar antes y terminar después de la siguiente. Foro de dudas.

5. **Ejecute** el test prommocion4-combinado. Foro de dudas.

5. Módulo listaPromociones

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

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

2. **Implemente** las funciones crearTListaPromocionesVacia, agregarPromocionTListaPromociones, imprimirTListaPromociones y liberarTListaPromociones. Verifique el funcionamiento de las funciones ejecutando el test listaPromociones1-crear-agregar-imprimir-liberar. Foro de dudas.

3. **Implemente** las funciones esVaciaTListaPromociones, perteneceTListaPromociones y obtenerPromocionTListaPromociones. **Ejecute** el test listaPromociones2-vacia-pertenece-obtener para verificar el funcionamiento de las funciones. Foro de dudas.

4. **Implemente** las funciones obtenerPromocionesFinalizadas y obtenerPromocionesActivas. **Ejecute** el test listaPromociones3-finalizadas-activas para verificar el funcionamiento de las funciones. Foro de dudas.

5. **Implemente** las funciones esCompatibleTListaPromociones y unirListaPromociones. **Ejecute** el test listaPromociones4-compatible-unir para verificar el funcionamiento de las funciones. Foro de dudas.

6. **Ejecute** el test listaPromociones5-combinado. Foro de dudas.

6. Módulo historial

En esta sección se describe la implementación del módulo *historial.cpp*. El módulo historial permitirá gestionar las promociones.
El historial está compuesto por tres listas de promociones: pasadas, activas y futuras. Además, almacena la fecha actual (representada con una variable de tipo TFecha), para poder determinar cuáles promociones están activas; cuáles, pasadas y cuáles, futuras.

1. **Implemente** la representación de *rep_historial* y la función crearTHistorial. Foro de dudas.

2. **Implemente** las funciones agregarProductoTHistorial y liberarTHistorial. Ejecute el caso de prueba historial1-crear-liberar. Foro de dudas.

3. **Implemente** las funciones agregarPromocionTHistorial y agregarProductoAPromocionTHistorial. Ejecute el caso de prueba historial2-agregarpromocion-agregarproductopromocion. Foro de dudas.

4. **Implemente** las funciones imprimirPromocionesFinalizadasTHistorial, imprimirPromocionesActivasTHistorial, e imprimirPromocionesFuturasTHistorial. Ejecute el caso de prueba historial3-imprimir. Foro de dudas.

5. **Implemente** la función esCompatiblePromocionTHistorial. Ejecute el caso de prueba historial4-compatible. Foro de dudas.

6. **Implemente** la función avanzarAFechaTHistorial. Recuerde que dicha función modifica las listas de promociones según la nueva fecha. Sugerencia: Estudie las funciones provistas por el módulo listaPromociones. Ejecute el caso de prueba historial5-avanzarfecha. Foro de dudas.

7. Ejecute el caso de prueba historial6-combinado. 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 --
1111111111111111111
```

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

```
conjuntoProductos1-crear-insertar-imprimir-liberar
conjuntoProductos2-esvacio-cardinal-cantmax
conjuntoProductos3-pertenece-borrar
conjuntoProductos4-union-interseccion-diferencia
conjuntoProductos5-combinado
promocion1-crear-agregar-imprimir-liberar
promocion2-pertenece-id-fechaIni-fechaFin
promocion3-compatibles
promocion4-combinado
listaPromociones1-crear-agregar-imprimir-liberar
listaPromociones2-vacia-pertenece-obtener
listaPromociones3-finalizadas-activas
listaPromociones4-compatible-unir
listaPromociones5-combinado
historial1-crear-liberar
historial2-agregarpromocion-agregarproductopromocion
historial3-imprimir
historial4-compatible
historial5-avanzarfecha
historial6-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 **conjuntoProductos.cpp**, **promocion.cpp**, **listaPrommociones.cpp** y **historial.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.