Tarea 2
Listas y árboles
Curso 2024

1. Introducción

Esta tarea tiene como principal objetivos trabajar sobre:

- el manejo dinámico de memoria,
- listas simplemente y doblemente enlazadas y
- árboles binarios de búsqueda.

La fecha límite de entrega es el **miércoles 18 de septiembre 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.

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
```

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 de Computación tiene como cometido desarrollar un prototipo de un sistema de gestión de productos y ventas de una cadena de tiendas llamada **Mercado FINGer**, que vende productos de alimentación. Cada tienda es una sucursal de la cadena y tiene asociado un número identificador.

La cadena necesita llevar el control de los productos que tiene a la venta (TProducto) y de los clientes registrados que visitan sus distintas sucursales para adquirir productos en ellas (TCliente).

2.2. Descripción general

Las sucursales tienen a la venta productos (3), de los que nos interesa conocer su identificador, su nombre, su fecha de ingreso y su precio.

Cuando se realiza una compra, el cliente agrega los distintos productos a un carrito de compras (TCarrito) hasta que se realiza el pago. El sistema a desarrollar debe llevar el control de los productos que un cliente pone en su carro en una lista de productos (4).

Además, se conoce la información de los clientes (5) de **Mercado FINGer**, de los cuales se sabe su nombre, apellido, edad y un identificador único.

Cada sucursal de **Mercado FINGer** lleva el registro del grupo de clientes que compra en ella (TClientesABB) (6), ordenados según su identificador.

Por último, para llevar el control de los clientes de las distintas sucursales, se implementa una colección de los grupos de clientes de cada sucursal (TClientesSucursalesLDE) (7). A los propietarios de **Mercado FIN-Ger** les interesa desarrollar promociones, facilidades y campañas de fidelización dirigidas a distintas franjas etarias, por lo cual se desea, por ejemplo, conocer los grupos de clientes con menor edad promedio.

En las siguientes secciones se describen los distintos módulos y funciones que se solicita implementar.

3. Módulo producto

En esta sección se describe la implementación del módulo *producto.cpp*. Cada elemento del tipo TProducto almacenará un *identificador*, un *nombre* para el producto, su *fecha de ingreso* y el *precio* de venta.

1. **Implemente** la representación de producto *rep_producto* y las funciones crearTProducto, idTProducto, imprimirTProducto y liberarTProducto. Tenga en cuenta que el formato de impresión se especifica en *producto.h*. Ejecute el caso de prueba producto1-crear-liberar y producto2-crear-imprimir-liberar para verificar el funcionamiento de las operaciones. Foro de dudas.

4. Lista simple: carrito de productos

En esta sección se describe la implementación del módulo *carritoProductos.cpp*. Los productos que son puestos en el carrito de compra estarán implementados mediante una *lista simplemente enlazada*, ordenada por el id. de los productos. Esta estructura lineal es dinámica y permitirá almacenar una cantidad no acotada de productos de un carrito de compra. 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 recomendarán fuertemente en otro módulo.

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

```
TProducto producto;
rep_carritoProductos * sig;
```

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

```
typedef struct rep_carritoProductos *TCarritoProductos;
```

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

```
TProducto producto;
TCarritoProductos sig;
```

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

2. **Implemente** las funciones crearCarritoProductosVacio, insertarProductoCarritoProductos, imprimirCarritoProductos y liberarCarritoProductos. Recomendamos que la representación de la lista vacía sea simplemente un *puntero a NULL*. Recuerde que el carrito de productos mantiene los productos de forma ordenada de **menor a mayor** por el id de producto, sabiendo que estos id's son únicos. Por otro lado, el formato en el que se debe imprimir el contenido de un carrito de productos es simplemente utilizando de forma secuencial la función *imprimirTProducto*. **Ejecute** el caso de prueba carritoProductos1-crear-insertar-imprimir-liberar. Foro de dudas.

3. **Implemente** las funciones esVacioCarritoProductos, existeProductoCarritoProductos y obtenerProductoCarritoProductos. **Ejecute** el caso de prueba carritoProductos2-vacia-existe-obtener. Foro de dudas.

4. **Implemente** la función removerProductoCarritoProductos y **ejecute** el caso de prueba carritoProductos3-remover. Foro de dudas.

5. **Ejecute** el caso de prueba carritoProductos4-combinado. Foro de dudas.

5. Módulo cliente

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

1. **Implemente** la representación de cliente *rep_cliente* y las funciones crearTCliente, imprimirTCliente y liberarTCliente. Tenga en cuenta que el formato de impresión se especifica en *cliente.h*. Ejecute el caso de prueba cliente1-crear-imprimir-liberar para verificar el funcionamiento de las operaciones. Foro de dudas.

2. **Implemente** las funciones idTCliente, nombreTCliente, apellidoTCliente y edadTCliente. **Ejecute** el test cliente2-id-nombre-apellido-edad para verificar las funciones. Foro de dudas.

3. **Implemente** la función copiarTCliente. **Ejecute** el test cliente3-copiar para verificar la función. **Ejecute** el test cliente4-combinado Foro de dudas.

6. Árbol binario de búsqueda: módulo clientesABB

En esta sección se implementará el módulo *clientesABB.cpp*. La estructura de tipo TClientesABB almacenará elementos del tipo TCliente y estará implementada como un **árbol binario de búsqueda (ABB)**, **ordenado por el identificador del cliente**. La estructura **no aceptará identificadores repetidos** (se puede asumir que nunca se agregarán repetidos). Se recomienda que las implementaciones sean recursivas.

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

2. **Implemente** las funciones crearTClientesABBVacio, insertarTclienteTClientesABB, imprimirTClientesABB y liberarTClientesABB. 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 clientesABB1-crear-insertar-imprimir-liberar para verificar el funcionamiento de las funciones. Foro de dudas.

3. **Implemente** las funciones existeTclienteTClientesABB y obtenerTClienteTClientesABB. **Ejecute** el test clientesABB2-existe-obtener para verificar el funcionamiento de las funciones. Foro de dudas.

4. **Implemente** la función alturaTClientesABB. **Ejecute** el test clientesABB3-altura para verificar el funcionamiento de la función. Foro de dudas.

5. **Implemente** las funciones maxIdTClienteTClientesABB y removerTClienteTClientesABB. **Ejecute** el test clientesABB4-maxid-remover para verificar el funcionamiento de las funciones. Foro de dudas.

6. **Implemente** las funciones cantidadClientesTClientesABB y edadPromedioTClientesABB. **Ejecute** el test clientesABB5-cantidad-edadPromedio para verificar el funcionamiento de las funciones. Foro de dudas.

7. **Implemente** la función obtenerNesimoClienteTClientesABB, que obtiene el cliente nro. N de un grupo, considerando el orden de id. La descripción detallada se encuentra en clientesABB.h. **Ejecute** el test clientesABB6-obtenerNesimo. Foro de dudas.

8. **Ejecute** el test clientesABB7-combinado. Foro de dudas.

7. Lista doblemente enlazada: módulo clientesSucursales

En esta sección se implementará el módulo *clientesSucursalesLDE.cpp*. Este representa una lista con los grupos de clientes (clientesABB) de las sucursales de **Mercado FINGer**. La estructura de tipo TClientesSucursalesLDE almacenará elementos del tipo TClientesABB 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 clientesABB. La lista **estará ordenada** según la edad promedio de clientes de las sucursales (clientesABB) (de menor a mayor).

1. **Implemente** la estructura *rep_clientesSucursalesLDE* 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 *TClientesABB*, el identificador de la sucursal (*idSucursal*) un puntero a un nodo *siguiente* y uno a un nodo *anterior*. Foro de dudas.

2. **Implemente** las funciones crearTClientesSucursalesLDEVacia, insertarClientesABBTClientesSucursalesLDE y liberarTClientesSucursalesLDE. Puede resultar útil implementar una **función auxiliar** que permita liberar un nodo individual. Verifique el funcionamiento de las funciones ejecutando el test clientesSucursalesLDE1-crear-insertar-liberar. Foro de dudas.

3. **Implemente** las funciones imprimirTClientesSucursalesLDE y imprimirInvertidoTClientesSucursalesLDE. **Ejecute** el test clientesSucursalesLDE2-imprimir para verificar el funcionamiento de las funciones. Foro de dudas.

4. **Implemente** las funciones cantidadTClientesABBClientesSucursalesLDE, obtenerPrimeroClientesSucursalesLDE y obtenerNesimoClientesSucursalesLDE. **Ejecute** el test clientesSucursalesLDE3-cantidad-primero-enesimo para verificar el funcionamiento de las funciones. Foro de dudas.

5. **Implemente** las funciones removerUltimoClientesSucursalesLDE y removerNesimoClientesSucursalesLDE. **Ejecute** el test clientesSucursalesLDE4-removerultimo-removerenesimo para verificar el funcionamiento de las funciones. Foro de dudas.

6. **Implemente** la función clienteMasRepetido. **Ejecute** el test clientesSucursalesLDE5-repetidos para verificar el funcionamiento la función. Foro de dudas.

7. **Ejecute** el test clientesSucursalesLDE6-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:

```
producto1-crear-liberar \
producto2-crear-imprimir-liberar \
carritoProductos1-crear-insertar-imprimir-liberar \
carritoProductos2-vacia-existe-obtener \
carritoProductos3-remover \
carritoProductos4-combinado \
cliente1-crear-imprimir-liberar \
cliente2-id-nombre-apellido-edad \
cliente3-copiar \
cliente4-combinado \
clientesABB1-crear-insertar-imprimir-liberar \
clientesABB2-existe-obtener \
clientesABB3-altura \
clientesABB4-maxid-remover \
clientesABB5-cantidad-edadPromedio \
clientesABB6-obtenerNesimo \
clientesABB7-combinado \
clientesABB8-tiempo \
clientesSucursalesLDE1-crear-insertar-liberar \
clientesSucursalesLDE2-imprimir \
clientesSucursalesLDE3-cantidad-primero-enesimo \
clientesSucursalesLDE4-removerultimo-removerenesimo \
clientesSucursalesLDE5-repetidos \
clientesSucursalesLDE6-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 **producto.cpp**, **carritoProductos.cpp**, **cliente.cpp**, **clientesABB.cpp** y **clientesSucursales.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.