Tecnológico de Monterrey, Campus Monterrey

Departamento de Ciencias Computacionales

**ESTRUCTURAS DE DATOS** - Ing. Román Martínez M.

**PRIMER EXAMEN DE PROGRAMACIÓN**  - 3 de Octubre de 2013

# Nombre: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Matrícula: \_\_\_\_\_\_\_\_\_\_\_\_

**Instrucciones:**

* Para resolver este examen deberás utilizar de preferencia el compilador de Visual C++ para programar tus respuestas.
* Se te provee de un código ya implementado que utilizarás y que podrás copiar para implementar tus respuestas. Para esta situación del examen, y con la finalidad de no estar con varios archivos, se programará todo en el mismo archivo, es decir, el contenido de los .h que normalmente realizas, estará concentrado en el mismo código fuente de un solo archivo que llamarás EX1.cpp.
* Documenta el código con los comentarios que consideres importantes y necesarios para explicar lo que programas. No olvides incluir una primera línea con el comentario de tu nombre y matrícula.
* El archivo final de entrega como respuesta de tu examen, deberá subirse a la página del curso en el espacio correspondiente.
* Está estrictamente PROHIBIDO el acceso a códigos previamente realizados y cualquier tipo de comunicación electrónica o presencial. Cualquier situación deshonesta será penalizada con una calificación de DA.

***PROBLEMA I.*** *(50 puntos – Tiempo sugerido para solucionar el problema: 45 min).*

Una COLA PRIORIZADA es una estructura de datos útil para administrar el orden de entrada de los datos como en una FILA (COLA), pero considerando que hay prioridades y que los datos de mayor prioridad deberán salir o procesarse antes de los datos de menor prioridad. Hay muchas situaciones cotidianas y computacionales en las que una cola priorizada tiene aplicación. Por ejemplo, en los parques de diversiones, los pases VIP o similares les dan prioridad a las personas que los poseen en la entrada a alguna atracción; en los bancos, los clientes VIP tienen preferencia en las ventanillas de atención; en una cola de impresión de documentos, hay prioridades de acuerdo al usuario que los manda, etc.

A continuación se muestra la especificación lógica del **ADT Cola Priorizada**

1. Elementos: cualquier tipo de dato válido según la aplicación.
2. Tipo de organización: Lineal
3. Dominio: la cola priorizada puede guardar cualquier cantidad de datos que permita la memoria.
4. Operaciones:

**INSERTA**

Sirve para agregar un dato con su correspondiente prioridad en la Cola priorizada

ENTRADAS: Cola priorizada en donde se desea insertar, dato a insertar y prioridad del dato

SALIDAS: Cola priorizada con el dato agregado.

PRECONDICIÓN: ninguna (se asume que siempre habrá memoria para agregar el dato)

POSTCONDICIÓN: La cola priorizada contiene el dato insertado y está posicionado de acuerdo a su prioridad.

**PROCESA**

Sirve para eliminar el dato que corresponde procesar según en orden de entrada (primero en entrar, primero en salir), pero considerando sea el de mayor prioridad en la Cola priorizada.

ENTRADAS: Cola priorizada en donde se desea procesar

SALIDAS: Cola priorizada con el dato eliminado y dato eliminado.

PRECONDICIÓN: la cola priorizada no está vacía

POSTCONDICIÓN: La cola priorizada contiene un dato menos ya que se eliminó el dato de mayor prioridad.

La precondición de la operación PROCESA obliga a tener una operación más en la especificación lógica, pero esta operación tú la deberás identificar y diseñar.

Para implementar este ADT se puede utilizar un arreglo de filas de tal manera que cada fila del arreglo guarde a los elementos de una misma prioridad. La posición del arreglo representará la prioridad, siendo la posición 0 la de mayor prioridad, la de la posición 1 la siguiente mayor prioridad y así sucesivamente. Con esta representación, la operación PROCESA al ejecutarse sacará el dato de la primera fila que tenga datos siguiendo el orden de las prioridades.

Realiza la implementación del **ADT Cola Priorizada** considerando la especificación lógica indicada y bajo la representación que se ha descrito. Utiliza a las filas de STL para hacer la implementación, y considera además que el usuario de las colas priorizadas puede indicar al declararlas cuántas prioridades manejará la cola. Esto significa que el arreglo de filas será dinámico. Cuando el usuario no indiqué la cantidad de prioridades en la declaración, se tomarán por default 3 prioridades.

Prueba tu implementación con las instrucciones que tu desees en el main.

En este problema, no se te da código ya implementado y deberás partir de cero en la implementación a realizar en el archivo EX1.cpp.

La evaluación de este problema se hará considerando la correcta implementación de los métodos constructores, el método destructor, y los método de las operaciones del ADT.

***PROBLEMA II.*** *(50 puntos – Tiempo sugerido para solucionar el problema: 30 min).*

En el archivo EX1.cpp podrás encontrar el código que corresponde a la implementación de una lista doblemente encadenada circular que hasta el momento sólo tiene implementada la operación de inserción de un dato. Analiza este código y utilízalo para responder a los siguientes casos:

1. Si ésta implementación correspondiera a una PILA, implementa a la operación SACAR-PILA bajo el diseño de que no tiene precondición.
2. Si ésta implementación correspondiera a una FILA, implementa a la operación SACAR-FILA bajo el diseño de que no tiene precondición.
3. Una lista doblemente encadenada circular, tiene la ventaja de que el apuntador que la referencia, puede moverse por los nodos de la lista hacia adelante y hacia atrás, sin perder el acceso a toda la información de la lista. Agrega a la clase de la lista la implementación de la siguiente operación que se especifica:

**ROTAR**

Lleva el apuntador externo de la lista tantas posiciones hacia adelante o hacia atrás como se indique.Un valor positivo significa un movimiento hacia adelante, y un valor negativo significa un movimiento hacia atrás.

ENTRADAS: Lista doblemente encadenada circular y cantidad de posiciones a mover el apuntador externo.

SALIDAS: Información del nodo en donde queda el apuntador después del movimiento.

PRECONDICIÓN: La lista no está vacía

POSTCONDICIÓN: La lista es referenciada a través de la nueva posición del apuntador.

***PROBLEMA III.*** *(puntos extras – Tiempo sugerido para solucionar el problema: 15 min).*

Una DOBLE FILA (conocida en inglés como DEQUE), es una modificación a la típica estructura de una FILA. Esta modificación consiste en que ahora los elementos podrán ser agregados o eliminados de la DOBLE FILA por cualquiera de los dos extremos de la estructura.

El siguiente diagrama muestra una representación gráfica o abstracta de una DOBLE FILA:



Siguiendo los típicos diseños de los ADT Pila y Fila, se puede observar que ahora el **ADT Doblefila** tendrá definidas al menos las siguientes operaciones: METERFRENTE, SACARFRENTE, METERFINAL y SACARFINAL. Estas operaciones se pueden diseñar a nivel lógico de la misma forma en que se han diseñado tradicionalmente en otras estructuras como las FILAS o las PILAS (mismas entradas, salidas, pre y postcondiciones).

Utilizando eficientemente el mecanismo de **herencia** y aplicando sólo las operaciones ya implementadas para la lista doblemente encadenada circular del problema II, implementa una DOBLE FILA.

**ANEXO – CÓDIGO FUENTE PARA TRABAJAR EN EL PROBLEMA II DEL EXAMEN**

template <class T>

class Nodo

{

public:

T info;

Nodo<T> \*sig, \*ant;

Nodo() { sig = ant = NULL; }

Nodo(T dato) { info = dato; sig = ant = NULL; }

};

template <class T>

class ListaDEC

{

private:

Nodo<T>\* externo;

public:

ListaDEC() { externo = NULL; }

void Insertar(T dato);

~ListaDEC();

};

template <class T>

void ListaDEC<T>::Insertar(T dato)

{

Nodo<T>\* aux = new Nodo<T>(dato);

if (aux == NULL) return false;

if (externo == NULL)

{

aux->sig = aux->ant = aux;

externo = aux;

}

else

{

aux->sig = externo->sig;

aux->ant = externo;

externo->sig->ant = aux;

externo->sig = aux;

}

return true;

}

template <class T>

ListaDEC<T>::~ListaDEC()

{

if (externo != NULL)

{

externo->ant->sig= NULL;

Nodo<T> \*aux = externo;

while (aux != NULL)

{

externo = aux->sig;

delete aux;

aux = externo;

}

}

}