Guía de ejercicios de repaso

Implementación de Estructuras Dinámicas de Datos

1. Desarrollar un implementación dinámica del TDA Pila.
2. Desarrollar una implementación dinámica del TDA Cola.
3. Implementar para la clase ListaDoblementeEnlazada el método agregarAlFinal.
4. Implementar para la clase ListSimplementeEnlazada el método

```
/* pre : la lista tiene al menos tantos elementos como posicionElemento.
 * post: remueve el elemento localizado en la posición posicionElemento de la lista.
 */
void remover(unsigned int posicionElemento);
```

5. Implementar para la clase ListaDoblementeEnlazada<T> la siguiente operación:

```
/*
 * post: invierte el orden de todos los elementos en la Lista:
 *       1º → Nº, 2º → (N-1)º, 3º → (N-2)º, etc.
 */
void invertir();
```

Uso de Estructuras Dinámicas de Datos

1. Implementar el método buscarConcursosIrregulares de la clase **AuditorDeConcursos** a partir de las siguientes especificaciones:

```
class AuditorDeConcursos {
  public:
    /* post: busca en concursosAuditados aquellos Concursos que tengan entre sus premiados
     *       algún nombre que no corresponda con el de un Participante asociado al Concurso.
     *       Devuelve una nueva Lista con todos los Concursos en esta condición. */
    Lista<Concurso*>* buscarConcursosIrregulares(Lista<Concurso*>* concursosAuditados);
};

class Concurso {
  public:
    /* post: inicializa el consurso sin participantes
     *       asociados, con el nombre indicado.
     */
    Concurso(string nombre);
    /* post: destruye todos los participantes asociados.
     */
    ~Concurso();
    /* post: devuelve el nombre que identifica el consurso.
     */
    string obtenerNombre();
    /* post: devuelve los participantes asociados
     */
    Lista<Participante*>* obtenerParticipante();
    /* post: devuelve los nombres de aquellos participantes
     *       que han sido premiados por su participación.
     */
    Lista<string>* obtenerPremiados();
};

class Participante {

  public:
    /* post: inicializa el participantes con el nombre indicado
     *       y 0 puntos.
     */
    Participante(string nombre);
    /* post: devuelve el nombre que los identifica
     */
    string obtenerNombre();
    /* post: devuelve la cantidad de puntos del participante
     */
    int obtenerPuntos();
    /* post: cambia por cantidad los puntos del participante
     */
    void cambiarPuntos(int cantidad);
};
```

2. Implementar el método seleccionarVideo de la clase Editor a partir de las siguientes especificaciones:

```
class Editor {

  public:

    /* post: devuelve una nueva lista con aquellos videos que tengan la etiqueta indicada
     *       y su promedio de calificaciones sea
     *       superior a calificacionPromedioMinima.
     */
    Lista<Video*>* seleccionarVideos(Lista<Video*>* videos, int calificacioPromedioMinima,
                                     string etiqueta);

};
```

```
class Video {

  public:
    /* post: inicializa el Video con el título y URL.
     */
    Video(string titulo, string url);
    /* post: devuelve el título del Video.
     */
    string obtenerTitulo();
    /* post: devuelve la URL del Video.
     */
    string obtenerUrl();
    /* post: devuelve los comentarios asociadas al Video.
     */
    Lista<Comentario*>* obtenerComentarios();
    /* post: devuelve las etiquetas del video
     /
    List<String>* obtenerEtiquetas();

    ~Video();
};

class Comentario {

  public:
    /* post: inicializa el Comentario con el contenido
     *       y calificación 0.
     */
    Comentario(string contenido);

    string obtenerContenido();
    /* post: devuelve la calificación [1 a 10] asociada,
     *       o 0 si el Comentario no tiene calificación.
     */
    int obtenerCalificacion();
    /* pre : calificacion está comprendido entre 1 y 10
     * post: cambia la calificación del Comentario.
     */
    void calificar(int calificacion);
};
```

3. Implementar la clase BuscadorDeComentarios a partir de las siguientes especificaciones:

```
class FiltroDeNoticias {

  public:

    /* post: remueve de noticias aquellas Noticias que tengan asociada TODAS las categorias
     *       dadas en conCategorias.
     */
    void removerNoticias(Lista<Noticia*>* noticias, Lista<Categoria*>* conCategorias);

};

class Noticia {

  public:
    /* post: inicializa la noticia con el título y cuerpo
     *       indicados, sin categorias asociadas.
     */
    Noticia(string titulo, string cuerpo);
    /* post: devuelve el título de la noticia.
     */
    string getTitulo();
    /* post: devuelve el cuerpo de la noticia.
     */
    string getCuerpo();
    /* post: devuelve las categorias asociadas a la noticia.
```

```
     */
    Lista<Categoria*>* getCategorias();

    ~Noticia();
};

class Categoria {

  public:
    /* post: inicializa la categoria con el nombre y la
     *       descripción indicadas.
     */
    Categoria(string nombre, string descripcion);
    /* post: devuelve el nombre que la identifica.
     */
    string getNombre();
    /* post: devuelve la descripción de la categoria.
     */
    string getDescripcion();
};
```

4. Implementar el método seleccionarVideo de la clase Editor a partir de las siguientes especificaciones:

```
class Editor {

  public:

    /* post: selecciona de videos aquel que tenga por lo menos tantos Comentarios como los
     *       indicados y el promedio de calificaciones sea máximo. Ignora los Comentarios sin
     *       calificación.
     */
    Video* seleccionarVideo(Lista<Video*>* videos, int cantidadDeComentarios);

};

class Video {

  public:
    /* post: inicializa el Video con el título y URL.
     */
    Video(string titulo, string url);
    /* post: devuelve el título del Video.
     */
    string obtenerTitulo();
    /* post: devuelve la URL del Video.
     */
    string obtenerUrl();
    /* post: devuelve los comentarios asociadas al Video.
     */
    Lista<Comentario*>* obtenerComentarios();

    ~Video();
};

class Comentario {

  public:
    /* post: inicializa el Comentario con el contenido
     *       y calificación 0.
     */
    Comentario(string contenido);

    string obtenerContenido();
    /* post: devuelve la calificación [1 a 10] asociada,
     *       o 0 si el Comentario no tiene calificación.
     */
    int obtenerCalificacion();
    /* pre : calificacion está comprendido entre 1 y 10
     * post: cambia la calificación del Comentario.
```

```
     */
    void calificar(int calificacion);
};
```

Tipo de Dato Abstracto

1. Diseñar la especificación e implementar el TDA **Cerrojo**. El Cerrojo posee tres estados posibles: ABIERTO, CERRADO o BLOQUEADO. Un Cerrojo debe crearse con un código de seguridad secreto y debe bloquearse luego de haber realizado más de 10 intentos fallidos consecutivos por abrirlo. Una vez que el Cerrojo está BLOQUEADO no es posible abrirlo nunca más. Debe proveer operaciones para:
    - conocer su estado
    - intentar abrirlo proporcionando un código
    - cerrarlo
    - devolver la cantidad de aperturas exitosas realizadas
    - devolver la cantidad de intentos restantes antes de pasar al estado BLOQUEADO
    - cambiar el código de seguridad (solo en caso de estar ABIERTO)
    - devolver la cantidad máxima de intentos fallidos consecutivos en toda la vida de Cerrojo.
2. Diseñar la especificación e implementar el TDA modelen un **GameDeTenis**. Un GameDeTenis debe proporcionar métodos para:
    - Anotar punto al sacador.
    - Anotar punto al receptor.
    - Saber si terminó.
    - Saber si ganó el receptor.
    - Saber si ganó el sacador.
    - Devolver el marcador actual en formato string.
    - Devolver una lista con los estados en los que estuvo el marcador.
3. Diseñar la especificación e implementar el TDA TarjetaBAJA. La TarjetaBAJA permite pagar pasajes en múltiples medios de transporte (COLECTIVO, TREN y SUBTE). Debe crearse sin saldo o con un saldo inicial y proveer operaciones para:
    - cargar saldo
    - pagar un viaje en un medio de transporte: TREN ($ 3,00), COLECTIVO ($ 1,70), SUBTE ($ 2,50)
    - devolver el saldo
    - devolver la cantidad total de viajes pagados
    - devolver la cantidad de viajes para el medio de transporte indicado
    - Debe permitir pagar viajes en COLECTIVO aún sin saldo, hasta un máximo de $ 10,00.

4. Implementar una clase que modele una **Subasta**:
    - Una Subasta debe ser creada con la descripción del artículo y opcionalmente el precio base de la misma. Si no se especifica, el precio base es $ 0.
    - La Subasta debe poseer métodos para realizar una oferta y conocer cuál es, hasta el momento, la oferta ganadora (la mayor de todas la ofertas realizadas).
    - Inicialmente la Subasta no está abierta (está pendiente, por tanto no recibe ofertas) y debe proveer un método para abrirla (a partir de lo cual recibe ofertas).
    - La Subasta debe proveer un método para cerrarla, a partir de lo cual no recibe más ofertas. Una vez cerrada no puede ser abierta nuevamente.
5. Diseñar la especificación e implementar el TDA **MarcadorDeBasquet**. Un Marcador de Basquet se crea indicando los nombres de los equipos que compiten: local y visitante. Un Marcador puede estar: no iniciado, en el primer cuarto, en el segundo cuarto, en el tercer cuarto, en el cuarto cuarto o terminado. Debe proveer operaciones para:
    - avanzar al siguiente estado
    - devolver el estado en el que se encuentra
    - anotar un tiro libre, doble o triple al equipo local o al visitante
    - devolver la cantidad total de puntos del local o del visitante
    - devolver la cantidad de puntos en un cuarto del local o del visitante
    - devolver la máxima distancia en puntos alcanzada entre los dos equipos