Pontificia Universidad Javeriana
Departamento de Ingeniería de Sistemas
Estructuras de Datos, 2023-10
Parcial 1 - 22 de febrero de 2023

Reglas y recomendaciones

Durante el parcial, deben observarse las siguientes reglas:

1. El parcial se debe desarrollar en el computador, escribiendo las respuestas en los campos designados dentro del cuestionario de BrightSpace.

2. Archivos adicionales se aceptarán sólo en el caso del diseño, de acuerdo a las instrucciones que se encuentran en el enunciado del parcial. Guarde regularmente su trabajo (para evitar posibles pérdidas de información), con los nombres de archivo
`par1_apellido1_apellido2_nombre1_nombre2.txt` (para el diseño de los TADs) y
`par1_apellido1_apellido2_nombre1_nombre2_dis.pdf` (para el diagrama de relación entre TADs)
Tenga en cuenta que los nombres y apellidos van todos en minúsculas, los espacios se reemplazan por guiones bajos y no se utilizan ni tildes ni 'ñ'. Estos archivos deben enviarse únicamente dentro de la pregunta correspondiente en el cuestionario de BrightSpace.

3. El parcial tiene una duración de dos horas, contadas a partir del inicio normal de la clase. El cuestionario en BrightSpace está programado para cerrarse al terminar las dos horas, por lo que no enviarlo dentro de ese lapso de tiempo se considerará como parcial no presentado y tendrá una calificación de 0.0.

4. El parcial es estrictamente individual y se debe desarrollar únicamente en la sala de computadores del día.

5. Se recomienda no utilizar compiladores, intérpretes de lenguajes de programación o entornos de desarrollo de cualquier tipo.

6. Puede utilizar sus apuntes, libros, e Internet para obtener la información que necesite para el parcial. Puede utilizar también herramientas de inteligencia artificial (tipo ChatGPT), bajo su propio riesgo y corroborando la información obtenida.

7. Sin embargo, está absolutamente prohibido comunicarse con cualquier otro ser humano para obtener información sobre el parcial, a través de cualquier medio (conversación directa, Teams, Skype, Zoom, Gtalk, Whatsapp, etcétera). También está totalmente prohibido copiar o transcribir texto o código de enunciados y respuestas de parciales anteriores, o de diseños e implementaciones que no sean propias.

8. Los celulares deben permanecer apagados, y no se debe enviar ni recibir ningún mensaje de texto.

9. La única excepción a lo anterior son los profesores y monitores de la asignatura, quienes sólo responderán consultas respecto a la claridad de las preguntas del parcial y no responderán consultas sobre la materia.

10. Si el estudiante incumple con cualquiera de las reglas, será evaluado con nota 0.0

11. **Recuerde que diseñar no implica implementar.**

1 (15%) Análisis de código

Considere la siguiente función (se garantiza que los argumentos contienen la cantidad de elementos necesarios para la correcta ejecución de la función):

```
typedef std::vector< float > TVector;
typedef std::vector< TVector > TVector2;

void func( TVector2 ml ) {

  TVector2::iterator mIt = ml.begin();
  TVector::iterator lIt = mIt->begin();

  for( TVector2::iterator i = ml.begin(); i != ml.end(); i++ ) {

    for( TVector::iterator j = i->begin(); j != i->end(); j++ ) {
      if ( *j < *lIt ) {
        mIt = i;
        lIt = j;
      }
    }

  }

  mIt->erase( lIt );
}
```

1.1 (7%) ¿Qué hace «func»? (descríbala en 12 palabras o menos)

Elimina el menor (el más pequeño) elemento de la multilista.

1.2 (8%) ¿Cuál es el orden de complejidad de «func»? Justifique brevemente su respuesta.

$O(n^2)$, el ciclo externo se repite tantas veces como listas hay en la multilista, el ciclo interno se repite por la longitud de cada sublista; ambos valores son desconocidos de antemano.

2 (24%) Selección múltiple con única respuesta

2.1 (8%) Para revisar los elementos almacenados en una cola es necesario:

1. definir un iterador que permita recorrer el contenedor y almacene los elementos en un contenedor auxiliar.
2. usar el acceso aleatorio para acceder a cada elemento.
3. extraer uno a uno los elementos del contenedor, y posiblemente almacenarlos en un contenedor auxiliar.
4. definir un apuntador que utilice las direcciones de memoria de cada elemento.

2.2 (8%) De los contenedores básicos de la STL (`vector`, `deque`, `list`), ¿cuáles de ellos pueden utilizarse como base para implementar una cola de forma eficiente?

1. sólo `vector`.
2. sólo `list`.
3. `vector` y `deque`.
4. `deque` y `list`.

2.3 (8%) Se ha identificado que un algoritmo tiene una complejidad de $O(n^2 \log n)$. ¿Qué tareas podría estar realizando el algoritmo?

1. Para cada lista de una multilista, ordenarla con *BubbleSort* y buscar un elemento en ella con búsqueda lineal.
2. Rellenar un vector a partir de una multilista, ordenar el vector y luego buscar un elemento allí con búsqueda binaria.
3. Rellenar una lista a partir de una multilista, ordenar la lista y luego buscar un elemento en la lista con búsqueda lineal.
4. Para cada lista de una multilista, ordenarla con *MergeSort* y buscar un elemento en ella con búsqueda binaria.

3 (61%) Diseño e Implementación de TADs

Desde que el sonido pudo generarse y almacenarse en formato digital, los reproductores de música han evolucionado de forma importante, mucho más cuando se abrió la posibilidad de tener los archivos de música en la nube. Antes, era necesario almacenar localmente (computador, celular, reproductor de música) los archivos con las canciones, normalmente en formato MP3. Ahora, sólo se necesita tener una cuenta en un servicio de música en línea (como Spotify, Deezer y similares) y contar con una buena conexión a Internet, lo cual da acceso a millones de canciones en todo el mundo, e incluso a otros contenidos de audio como *podcasts*.

Para mantener organizados estos archivos de música, lo más común es tener una estructura basada en los artistas, sus álbumes y las canciones que los componen. De esta forma, de cada canción es posible almacenar su título, su duración en segundos y la cantidad de veces que ha sido reproducida por el usuario. La duración en segundos de cada canción se transforma al formato horas - minutos - segundos sólo al momento de visualizar la información de la canción en el reproductor. Cada álbum se compone entonces de un conjunto de canciones, y ese álbum también tiene asociado un título y un año de lanzamiento. Finalmente, de cada artista se almacena su nombre, su nacionalidad y el conjunto de álbumes que ha lanzado, usualmente ordenados de acuerdo al año de lanzamiento (el álbum más reciente primero, pues suele ser el que más reproducciones demanda).

El reproductor de música, además de tener acceso a esta información, dispone de una cola de reproducción, en donde el usuario va encolando o agregando las canciones que desea escuchar en un momento dado. Estas canciones se van reproduciendo en el orden en el que el usuario las va agregando a la cola. Para poder tener acceso a las canciones bajo el esquema descrito anteriormente cuando no se dispone de acceso a internet, el reproductor usualmente mantiene una copia local de las canciones en la cola, facilitando así que el usuario pueda escuchar su música sin interrupciones.

Se quiere enriquecer el reproductor con algunas opciones adicionales, que faciliten al usuario escuchar su música favorita bajo ciertos parámetros. Para esto, se asumirá que el reproductor tiene acceso local a la estructura de artistas, álbumes y canciones global (sólo por cuestiones de implementación), con lo cual se desea implementar las siguientes operaciones:

1. Actualizar la estructura de artistas, álbumes y canciones agregando una nueva canción de un artista y álbum dado. Del artista sólo se conoce su nombre, del álbum sólo se conoce su título, pero de la canción si se tiene toda la información necesaria. Si el álbum dado no existe en el sistema, es necesario crearlo, utilizando como año de lanzamiento el año actual. Si el artista dado no existe en el sistema, es necesario crearlo, usando como nacionalidad "Desconocida" y creando el correspondiente álbum como se indicó anteriormente. Al agregar la canción al sistema, también debe agregarse de forma automática a la cola de reproducción actual.

2. Ordenar la cola de reproducción actual para acomodar las canciones de acuerdo a la cantidad de veces que han sido escuchadas por el usuario actual. La idea es dejar adelante en la cola aquellas canciones más escuchadas, mientras que las menos escuchadas quedan hacia el final de la cola de reproducción.

Se le pide entonces diseñar e implementar (en C++) los componentes ya descritos del sistema reproductor de música.

3.1 (16%) Diseño

Diseñe el sistema y el (los) TAD(s) solicitado(s). Utilice la plantilla de especificación de TADs vista en clase para el diseño. Recuerde que diseñar es un proceso previo a la implementación, por lo que no debería contener ninguna referencia a lenguajes de programación (es decir, si escribe encabezados o código fuente, el punto no será evaluado y tendrá una calificación de cero). Para simplicidad del diseño, no es necesario incluir los métodos obtener y fijar (get/set) del estado de cada TAD.

TAD Cancion
Datos mínimos:
- titulo, cadena de caracteres, representa el título de la canción.
- duracion, entero largo, representa la duración total en segundos de la canción.
- reps, entero, representa la cantidad de veces que la canción ha sido escuchada o reproducida.

Operaciones:
- ObtenerTitulo(), retorna el título actual de la canción.
- ObtenerDuracion(), retorna la duración actual de la canción.
- ObtenerReps(), retorna la cantidad de reproducciones actual de la canción.
- FijarTitulo(ntitulo), cambia el título de la canción a ntitulo.
- FijarDuracion(nduracion), cambia la duración de la canción a nduración.
- FijarReps(nreps), cambia la cantidad de reproducciones de la canción a nreps.

TAD Album
- titulo, cadena de caracteres, representa el título del álbum.
- anno, entero, representa el año de lanzamiento del álbum.
- canciones, lista de Cancion, representa las canciones que hacen parte del álbum.

Operaciones:
- ObtenerTitulo(), retorna el título actual del álbum.
- ObtenerAnno(), retorna el año de lanzamiento actual del álbum.
- ObtenerCanciones(), retorna el conjunto de canciones actual del álbum.
- FijarTitulo(ntitulo), cambia el título del álbum a ntitulo.
- FijarAnno(nanno), cambia el año de lanzamiento del álbum a nanno.
- FijarCanciones(ncanciones), cambia el conjunto de canciones del álbum a ncanciones.

TAD Artista
- nombre, cadena de caracteres, representa el nombre del artista.
- nacion, cadena de caracteres, representa la nacionalidad del artista.
- albumes, lista de Album, representa los álbumes lanzados por el artista.

Operaciones:
- ObtenerNombre(), retorna el nombre actual del artista.
- ObtenerNacion(), retorna la nacionalidad actual del artista.
- ObtenerAlbumes(), retorna el conjunto de álbumes actual del artista.
- FijarNombre(nnombre), cambia el nombre del artista a nnombre.
- FijarNacion(nnacion), cambia la nacionalidad del artista a nnacion.
- FijarAlbumes(nalbumes), cambia el conjunto de álbumes del artista a nalbumes.

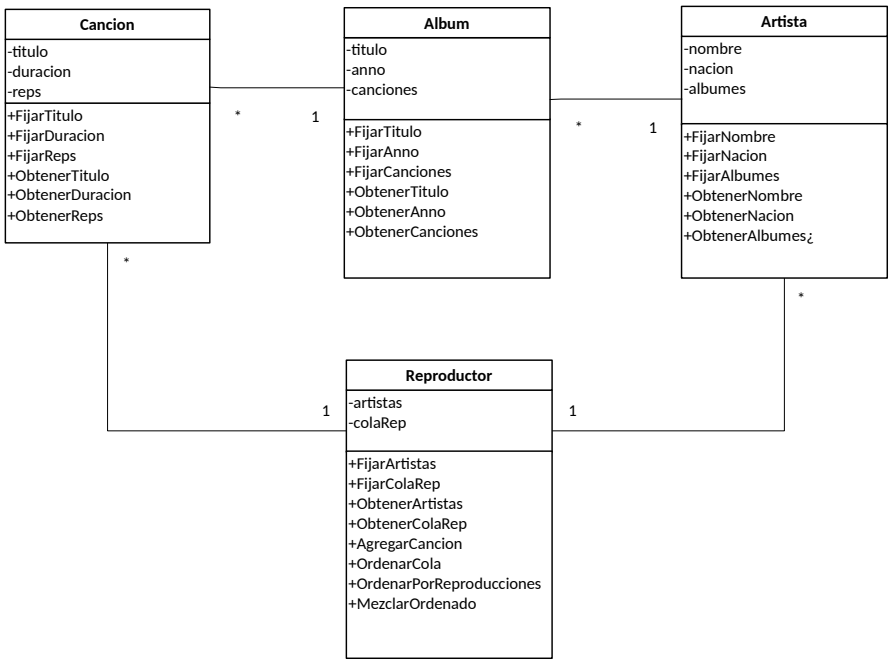
TAD Reproductor
- artistas, lista de Artista, representa la información de artistas, álbumes y canciones que posee el sistema reproductor de música.
- colaRep, cola de Cancion, representa la cola de reproducción de canciones del usuario.

Operaciones:
- ObtenerArtistas(), retorna el conjunto de artistas actual del reproductor.
- ObtenerColaRep(), retorna la cola de reproducción actual del reproductor.
- FijarArtistas(nartistas), cambia el conjunto de artistas del reproductor a nartistas.
- FijarColaRep(ncola), cambia la cola de reproducción del reproductor a ncola.
- AgregarCancion(artista,album,titulo,duracion), agrega una nueva canción dentro de la estructura de artistas, álbumes y canciones del reproductor. Se utiliza el artista y album para ubicar la canción, y se agrega inicializando el contador de reproducciones

```
de la canción en 0.
- OrdenarCola(), reorganiza las canciones en la cola de reproducción para ordenarlas en
  forma descendente de acuerdo a la cantidad de veces que cada una ha sido escuchada.
- OrdenarPorReproducciones(lista,inicio,final), ordena una lista de canciones desde la
  posición de inicio hasta la posición final.
- MezclerOrdenado(lista,inicio,medio,final), mezcla de forma ordenada dos partes de la
  lista dada, la primera entre inicio y medio, y la segunda entre medio y final.
```

3.2 (7%) Diagrama de relación

El diseño incluye el diagrama de relación entre TADs, cuando se definen dos o más TADs en el punto anterior. En ese caso, adjúntelo en formato PDF, JPG o PNG como parte de su entrega.



3.3 (21%) Operación 1: Agregar canción.

Dado el (los) TAD(s) ya diseñado(s), escriba la implementación en C++ del algoritmo que permite agregar una nueva canción al sistema y a la cola de reproducción, dado el artista, el álbum y la información completa de la canción. La implementación deberá tener en cuenta:

- la definición apropiada de los prototipos de los métodos/funciones (i.e. recibir/retornar los datos suficientes y necesarios para su correcta ejecución),
- el NO uso de salidas/entradas por pantalla/teclado (i.e. paso/retorno correcto de valores y/o objetos),
- el correcto uso del diseño definido en el punto anterior, y
- la escritura de todo el código que pueda llegar a necesitar que no esté incluido en la STL.

```
// Pasos a seguir:
// 1. Ubicar el artista dado dentro de la información del reproductor. Si no existe,
//    crearlo con nacionalidad ‘‘Desconocida’’.
// 2. Para ese artista, ubicar el álbum dado. Si no existe, crearlo con año de
```

```
//    lanzamiento el actual.
// 3. Dentro de ese álbum, crear la canción con cantidad de repeticiones en 0 y
//    agregarla a la lista de canciones del álbum.
// 4. Agregar esa misma canción a la cola de reproducción actual.

bool Reproductor::AgregarCancion ( string nartista, string talbum,
                                   string titulo, long duracion ) {
  bool encArtista, encAlbum;
  bool agregado = false;

  // 1. Encontrar artista o agregarlo si no existe
  std::list<Artista>::iterator itAr = artistas.begin();
  encArtista = false;
  while (itAr != artistas.end() && !encArtista) {
    if (itAr->ObtenerNombre() == nartista)
      encArtista = true;
    else
      itAr++;
  }
  if (!encArtista && itAr == artistas.end()) {
    Artista nuevoArt = new Artista;
    nuevoArt.FijarNombre(nartista);
    nuevoArt.FijarNacion(“Desconocida”);
    artistas.push_back(nuevoArt);
    itAr = std::prev(artistas.end()); // posición del artista que acabo de insertar
  }

  // 2. Encontrar album o agregarlo si no existe
  std::list<Album> aalbumes = itAr->ObtenerAlbumes();
  std::list<Album>::iterator itAl = aalbumes.begin();
  encAlbum = false;
  while (itAl != aalbumes.end() && !encAlbum) {
    if (itAl->ObtenerTitulo() == talbum)
      encAlbum = true;
    else
      itAl++;
  }
  if (!encAlbum && itAl == aalbumes.end()) {
    Album nuevoAlb = new Album;
    nuevoAlb.FijarTitulo(talbum);
    nuevoAlb.FijarAnno(2023);
    aalbumes.push_back(nuevoAlb);
    itAl = std::prev(aalbumes.end()); // posición del album que acabo de insertar
  }

  // 3. Crear la canción con sus datos
  std::list<Cancion> acanciones = itAl->ObtenerCanciones();
  Cancion nuevaCanc = new Cancion;
  nuevaCanc.FijarTitulo(titulo);
  nuevaCanc.FijarDuracion(duracion);
```

```
  nuevaCanc.FijarReps(0);
  acanciones.push_back(nuevaCanc);

  itAl->FijarCanciones(acanciones);
  itAr->FijarAlbumes(aalbumes);
  agregado = true;

  // 4. Agregar a la cola de reproducción actual
  colaRep.push(nuevaCan);

  return agregado;
}
```

3.4 (17%) Operación 2: Ordenar la cola de reproducción.

Dado el (los) TAD(s) ya diseñado(s), escriba la implementación en C++ del algoritmo que permite ordenar la cola de reproducción de acuerdo a la cantidad de veces que las canciones han sido escuchadas. Así como en el punto anterior, la implementación deberá tener en cuenta:

- la definición apropiada de los prototipos de los métodos/funciones (i.e. recibir/retornar los datos suficientes y necesarios para su correcta ejecución),
- el NO uso de salidas/entradas por pantalla/teclado (i.e. paso/retorno correcto de valores y/o objetos),
- el correcto uso del diseño definido en el punto anterior, y
- la escritura de todo el código que pueda llegar a necesitar que no esté incluido en la STL.

```
// Pasos a seguir:
// 1. Extraer todas las canciones de la cola de reproducción a una estructura de
//    acceso aleatorio.
// 2. Aplicar un algoritmo de ordenamiento sobre las canciones en la nueva estructura,
//    utilizando como criterio la cantidad de reproducciones de cada canción.
// 3. Poner en la cola las canciones ordenadas.

bool Reproductor::OrdenarCola ( ) {
  bool ordenada = false;

  // 1. Extraer de la cola a un vector
  std::vector<Cancion> lCanciones;
  Cancion unaCanc;
  while (!colaRep.empty()) {
    unaCanc = colaRep.front();
    colaRep.pop();
    lCanciones.push_back(unaCanc);
  }

  // 2. Ordenar el vector por número de reproducciones
  OrdenarPorReproducciones(lCanciones,0,lCanciones.size()-1);

  // 3. Poner nuevamente en la cola
```

```
  for (int i = 0; i < lCanciones.size(); i++) {
    colaRep.push(lCanciones[i]);
  }
  ordenada = true;

  return ordenada;
}

void Reproductor::OrdenarPorReproducciones ( std::vector<Cancion>& lcanc,
                                             int inicio, int final ) {
  if (inicio >= final)
    return;
  int medio = inicio + (final - inicio) / 2;
  OrdenarPorReproducciones(lcanc,inicio,medio);
  OrdenarPorReproducciones(lcanc,medio+1,final);
  MezclarOrdenado(lcanc,inicio,medio,final);
}

void Reproductor::MezclarOrdenado ( std::vector<Cancion>& lcanc,
                                    int inicio, int medio, int final ) {
  std::vector<Cancion> parte1, parte2;
  std::copy(lcanc.begin()+inicio,lcanc.begin()+(medio-inicio+1),parte1.begin());
  std::copy(lcanc.begin()+medio,lcanc.begin()+(final-medio),parte2.begin());
  int i = 0;
  int j = 0;
  int k = inicio;
  while ( i < parte1.size() && j < parte2.size() ) {
    if ( parte1[i].ObtenerReps() >= parte2[j].ObtenerReps() ) {
      lcanc[k] = parte1[i];
      i++;
    } else {
      lcanc[k] = parte2[j];
      j++;
    }
    k++;
  }
  while ( i < parte1.size() ) {
    lcanc[k] = parte1[i];
    k++;
    i++;
  }
  while ( j < parte2.size() ) {
    lcanc[k] = parte2[j];
    k++;
    j++;
  }
}
```