CUCEI

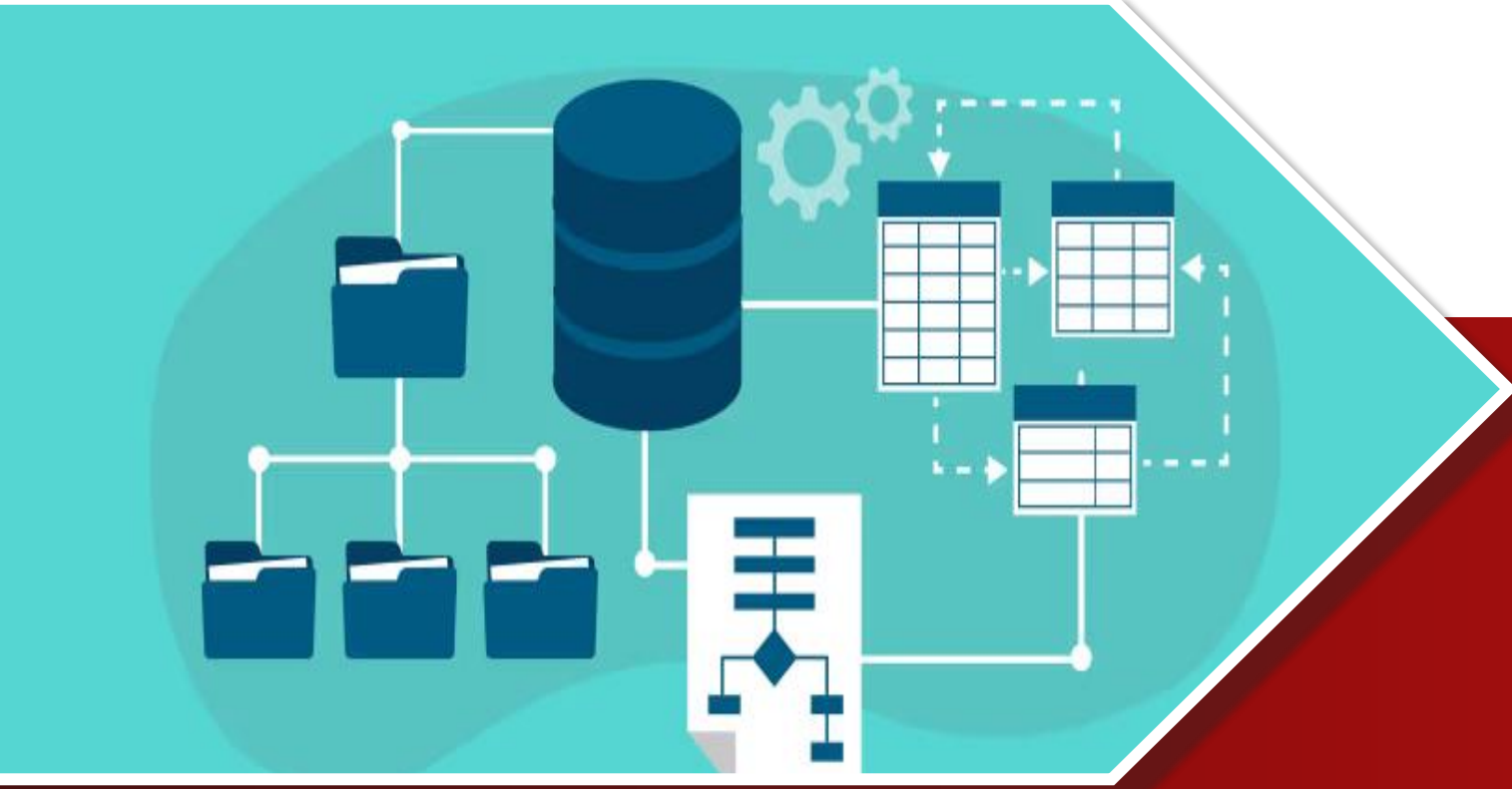
Actividad de Aprendizaje 10

La Lista: Implementación Dinámica Simplemente Ligada

Centro Universitario de Ciencias Exactas e Ingenierías

Materia: Estructuras de Datos

Clave: V0731 ***Sección***: D02



Alumno: Mariscal Rodríguez Omar Jesús

Código: 220858478

Profesor: Gutiérrez Hernández Alfredo

Fecha: 19 de Octubre de 2025

UNIVERSIDAD DE GUADALAJARA

PIENSA Y TRABAJA

Contenido

Test de Autoevaluación 3
Introducción y Abordaje del Problema 4
Planteamiento del Problema 4
Código Fuente 6
Carpeta Include 6
list.hpp 6
menu.hpp 23
name.hpp 25
ownexceptions.hpp 26
song.hpp 28
Carpeta src 30
main.cpp 30
menu.cpp 31
name.cpp 54
song.cpp 56
Ejecución del Programa 60
Conclusiones 65

Test de Autoevaluación

Autoevaluación			
Concepto	***Sí***	***No***	***Acumulado***
Bajé el trabajo de internet o alguien me lo pasó (aunque sea de forma parcial)	***-100 pts***	***0 pts***	***0***
Incluí el código fuente ***en formato de texto (sólo si funciona cumpliendo todos los requerimientos)***	***+25 pts***	***0 pts***	***25***
Incluí las ***impresiones de pantalla (sólo si funciona cumpliendo todos los requerimientos)***	***+25 pts***	***0 pts***	***25***
Incluí una ***portada*** que identifica mi trabajo (nombre, código, materia, fecha, título)	***+25 pts***	***0 pts***	***25***
Incluí una ***descripción y conclusiones*** de mi trabajo	***+25 pts***	***0 pts***	***25***
		Suma:	***100***

Introducción y Abordaje del Problema

El propósito de esta actividad fue activar los conocimientos que obtuvimos con apuntadores para la lista dinámica simplemente ligada, recuperando la actividad de las canciones, adaptando una lista que implementamos en clase a esta actividad

Planteamiento del Problema

Adaptar una lista en un principio creí que sería como las demás actividades, pero fue un poco más interesante, y me refería a que creía que con la lista programada su implementación sería más fácil como las demás, pero no había considerando el hecho de que con la lista simplemente ligada sin encabezados utilizamos una clase interna y manejábamos las posiciones con aquella, lo que cambió la manera de implementación y ciertas firmas, así, que para que la lista funcionara hubo que realizar ciertas adaptaciones para que el menú pueda seguir con índices lo máximo posible, y sea más adaptable, ello requirió de firmas adicionales e implementaciones auxiliares.

También existían ciertos dilemas, como la búsqueda binaria u ordenamientos que requerían de posiciones de un arreglo, esto es muy conflictivo; para la práctica, se iba a utilizar la versión más actualizada del programa y con las mayores funcionalidades, aquí entran ordenamientos y búsquedas de diversos tipos, sin embargo, por la misma naturaleza de una lista enlazada, el acceso a los "índices" (posiciones en este caso) requieren de un recorrido de lista, y esto provoca ineficiencias varias en ciertos métodos que por modelo requieren de un acceso a cierto índice de manera directa como un Shell sort o una búsqueda binaria, aún así, para la práctica se decidió adaptar estas funcionalidades, aunque resaltando fuertemente que NO SON EFICIENTES, no deberían ni utilizarse ni considerarse en un caso real de programación.

Ejemplo de esto es la búsqueda binaria; aquella requiere de un acceso inmediato a la mitad de la lista, y para ello, se requiere conocer la totalidad del arreglo y poder acceder a él, una búsqueda binaria sin acceso directo requiere de un recorrido donde ya se pudo haber encontrado en arreglo, esto NO ES IDEAL, no debe implementarse en casos reales, pero fueron programados a fin de practicar con apuntadores y adaptar una lista, pero NO SON EJEMPLOS REALES.

Recalcado aquello, se utilizaron recursos de más para la adaptación y que el usuario pueda seguir seleccionando una “búsqueda binaria” o un “Shell Sort” por ejemplo.

Código Fuente

Carpeta Include

list.hpp

```
#ifndef __LIST_H__
#define __LIST_H__

#include <string>

#include "ownexceptions.hpp"

using namespace std;

// Definición
template <class T>
class List {
  class Node;

 public:
  typedef Node* Position;

 private:
  class Node {
   private:
    T data;
    Position next = nullptr;

   public:
    Node();
    Node(const T&);

    T& getData();
    Position getNext() const;

    void setData(const T&);
    void setNext(const Position&);
  };

  bool isValidPosition(const Position&) const;
  void add(const List<T>&);

  Position anchor = nullptr;

 public:
  List();
  List(const List<T>&);
```

```
  ~List();

  bool isEmpty() const;

  void insertData(const T&, const Position&);
  void insertData(const T&, const int&);

  void deleteData(const Position&);
  void deleteData(const int&);

  Position getFirstPos() const;
  Position getLastPos() const;
  int getLastPosition() const;
  Position getPrevPos(const Position&) const;
  Position getNextPos(const Position&) const;

  Position findData(const T&) const;
  int findDataL(const T&) const;
  int findDataL(const T&, int(const T&, const T&)) const;
  int findDataB(const T&, int(const T&, const T&)) const;

  T& retrieve(const Position&);
  T* retrieve(const int&);

  string toString() const;
  void insertSortedData(const T&);

  /*Existen Aquí ordenamientos muy ineficientes que sólo fueron adaptados
para
     mantener las funcionaledes del programa que ya tenía como la
búsqueda
     binaria o un sortDataShell, que requirieron de implementaciones poco
     eficientes se implementaron SOLO CON LA FINALIDAD DE QUE EL PROGRAMA
NO
     PERDIERA DICHAS FUNCIONES pero en casos de implementación real NO
DEBEN DE
     IMPLEMENTARSE EN UNA LISTA CON ESTAS CONDICIONES Dicho aquello y
     considerando que se buscó la práctica y dejar el programa con las
maayores
     funcionalidades intactas, seguimos
  */
  void sortDataBubble();
  void sortDataInsert();
  void sortDataSelect();
  void sortDataShell();

  void deleteAll();
```

```
    List<T>& operator=(const List<T>&);

    void sortDataBubble(int(const T&, const T&));
    void sortDataInsert(int(const T&, const T&));
    void sortDataSelect(int(const T&, const T&));
    void sortDataShell(int(const T&, const T&));
    bool isValidPosition(const int&) const;
    Position getNodeAt(int) const;

    template <class U>
    friend std::ostream& operator<<(std::ostream&, const List<U>&);
    template <class U>
    friend std::istream& operator>>(std::istream&, List<U>&);
};

// Implementación
// Node
template <class T>
List<T>::Node::Node() {}

template <class T>
List<T>::Node::Node(const T& element) : data(element) {}

template <class T>
T& List<T>::Node::getData() {
    return this->data;
}

template <class T>
typename List<T>::Position List<T>::Node::getNext() const {
    return this->next;
}

template <class T>
void List<T>::Node::setData(const T& data) {
    this->data = data;
}

template <class T>
void List<T>::Node::setNext(const typename List<T>::Position& pointer) {
    this->next = pointer;
}

// Lista

template <class T>
bool List<T>::isValidPosition(const Position& pointer) const {
    Position aux = this->anchor;
```

```
  while (aux != nullptr) {
    if (aux == pointer) {
      return true;
    }
    aux = aux->getNext();
  }

  return false;
}

template <class T>
bool List<T>::isValidPosition(const int& index) const {
  if (index < 0) {
    return false;
  }

  Position aux = this->anchor;
  int count = 0;

  while (aux != nullptr) {
    if (count == index) {
      return true;
    }
    aux = aux->getNext();
    count++;
  }

  return false;
}

template <class T>
void List<T>::add(const List<T>& other) {
  Position aux(other.anchor), lastInsert(this->getLastPos()),
newNode(nullptr);

  while (aux != nullptr) {
    newNode = new Node(aux->getData());
    if (newNode == nullptr)
      throw DataContainersExceptions::MemoryDeficiency("Memoria no
Disponible");

    if (lastInsert == nullptr)
      this->anchor = newNode;
    else
      lastInsert->setNext(newNode);

    lastInsert = newNode;
```

```
    aux = aux->getNext();
  }
}

template <class T>
List<T>::List() {}

template <class T>
List<T>::List(const List<T>& other) {
  this->add(other);
}

template <class T>
List<T>::~List() {
  this->deleteAll();
}

template <class T>
bool List<T>::isEmpty() const {
  return this->anchor == nullptr;
}

template <class T>
void List<T>::insertData(const T& element,
                         const typename List<T>::Position& position) {
  if (position != nullptr && !this->isValidPosition(position))
    throw DataContainersExceptions::InvalidPosition();

  Position newNode(new Node(element));
  if (newNode == nullptr)
    throw DataContainersExceptions::MemoryOverflow("Memoria No
Disponible");

  if (position == nullptr) {  // Insertar al Principio
    newNode->setNext(this->anchor);
    this->anchor = newNode;
  }

  else {  // Cualquier otra
    newNode->setNext(position->getNext());
    position->setNext(newNode);
  }
}

template <class T>
void List<T>::deleteData(const typename List<T>::Position& position) {
  if (!this->isValidPosition(position))
    throw DataContainersExceptions::InvalidPosition();
```

```
  if (position == this->anchor)  // Eliminar el primero
    this->anchor = position->getNext();

  else  // Cualquier otra Posición
    this->getPrevPos(position)->setNext(position->getNext());

  delete position;
}

template <class T>
typename List<T>::Position List<T>::getFirstPos() const {
  return this->anchor;
}

template <class T>
typename List<T>::Position List<T>::getLastPos() const {
  if (this->isEmpty())
    return nullptr;

  Position aux(this->anchor);
  while (aux->getNext() != nullptr)
    aux = aux->getNext();

  return aux;
}

template <class T>
typename List<T>::Position List<T>::getPrevPos(
    const typename List<T>::Position& position) const {
  if (position == nullptr)
    return nullptr;

  Position aux(this->anchor);
  while (aux != nullptr && aux->getNext() != position)
    aux = aux->getNext();

  return aux;
}
template <class T>
typename List<T>::Position List<T>::getNextPos(
    const typename List<T>::Position& position) const {
  return this->isValidPosition(position) ? position->getNext() : nullptr;
}

template <class T>
typename List<T>::Position List<T>::findData(const T& dataSearched) const
{
```

```
  Position aux(this->anchor);

  while (aux != nullptr && aux->getData() != dataSearched)
    aux = aux->getNext();
  return aux;
}

template <class T>
T& List<T>::retrieve(const typename List<T>::Position& p) {
  if (!this->isValidPosition(p))
    throw DataContainersExceptions::InvalidPosition();
  return p->getData();
}

template <class T>
string List<T>::toString() const {
  ostringstream oss;
  Position aux(this->anchor);
  int cont;
  while (aux != nullptr) {
    oss << "| " << std::to_string(cont)
        << std::setw(11 - std::to_string(cont).size()) << ""
        << aux->getData().toString() << "\n";
    cont++;
    aux = aux->getNext();
  }

  return oss.str();
}

template <class T>
void List<T>::deleteAll() {
  Position aux;

  while (this->anchor != nullptr) {
    aux = this->anchor;
    this->anchor = aux->getNext();
    delete aux;
  }
}

template <class T>
void List<T>::sortDataBubble() {
  if (this->anchor == nullptr || this->anchor->getNext() == nullptr)
    return;

  bool swapped;
  Position ptr1;
```

```
  Position lptr = nullptr;

  do {
    swapped = false;
    ptr1 = this->anchor;

    while (ptr1->getNext() != lptr) {
      if (ptr1->getData() > ptr1->getNext()->getData()) {
        T temp = ptr1->getData();
        ptr1->setData(ptr1->getNext()->getData());
        ptr1->getNext()->setData(temp);
        swapped = true;
      }
      ptr1 = ptr1->getNext();
    }
    lptr = ptr1;
  } while (swapped);
}

template <class T>
void List<T>::sortDataInsert() {
  if (this->anchor == nullptr || this->anchor->getNext() == nullptr)
    return;

  Position sorted = this->anchor;
  Position current = this->anchor->getNext();
  sorted->setNext(nullptr);

  while (current != nullptr) {
    Position next = current->getNext();

    if (current->getData() < sorted->getData()) {
      current->setNext(sorted);
      sorted = current;
    } else {
      Position temp = sorted;
      while (temp->getNext() != nullptr &&
             temp->getNext()->getData() <= current->getData()) {
        temp = temp->getNext();
      }
      current->setNext(temp->getNext());
      temp->setNext(current);
    }

    current = next;
  }

  this->anchor = sorted;
```

```
}

template <class T>
void List<T>::sortDataSelect() {
  if (this->anchor == nullptr || this->anchor->getNext() == nullptr) {
    return;
  }

  Position current = this->anchor;
  while (current != nullptr) {
    Position minNode = current;
    Position scanner = current->getNext();

    while (scanner != nullptr) {
      if (scanner->getData() < minNode->getData()) {
        minNode = scanner;
      }
      scanner = scanner->getNext();
    }

    if (minNode != current) {
      T temp = current->getData();
      current->setData(minNode->getData());
      minNode->setData(temp);
    }

    current = current->getNext();
  }
}

template <class T>
void List<T>::sortDataShell() {
  if (this->anchor == nullptr || this->anchor->getNext() == nullptr) {
    return;
  }

  int series[] = {4181, 2584, 1597, 987, 610, 377, 233, 144, 89, 55,
                  34,   21,   13,   8,   5,   3,   2,   1,   0};

  int size = 0;
  Position temp = this->anchor;
  while (temp != nullptr) {
    size++;
    temp = temp->getNext();
  }

  int pos = 0;
  int gap = series[pos];
```

```
  while (gap > size) {
    gap = series[++pos];
  }

  while (gap > 0) {
    for (int i = gap; i < size; i++) {
      Position nodeI = this->getNodeAt(i);
      T tempData = nodeI->getData();

      int j = i;
      while (j >= gap) {
        Position nodeJMinusGap = this->getNodeAt(j - gap);

        if (nodeJMinusGap->getData() > tempData) {
          Position nodeJ = this->getNodeAt(j);
          nodeJ->setData(nodeJMinusGap->getData());
          j -= gap;
        } else {
          break;
        }
      }

      Position nodeJ = this->getNodeAt(j);
      nodeJ->setData(tempData);
    }

    gap = series[++pos];
  }
}

template <class T>
List<T>& List<T>::operator=(const List<T>& other) {
  this->deleteAll();
  this->add(other);

  return *this;
}

template <class T>
void List<T>::sortDataBubble(int cmp(const T&, const T&)) {
  if (this->anchor == nullptr || this->anchor->getNext() == nullptr) {
    return;
  }

  bool flag;
  Position lptr = nullptr;
```

```
    do {
      flag = false;
      Position ptr1 = this->anchor;

      while (ptr1->getNext() != lptr) {
        if (cmp(ptr1->getData(), ptr1->getNext()->getData()) > 0) {
          T temp = ptr1->getData();
          ptr1->setData(ptr1->getNext()->getData());
          ptr1->getNext()->setData(temp);
          flag = true;
        }
        ptr1 = ptr1->getNext();
      }
      lptr = ptr1;
    } while (flag);
}

template <class T>
void List<T>::sortDataInsert(int cmp(const T&, const T&)) {
  if (this->anchor == nullptr || this->anchor->getNext() == nullptr)
    return;

  Position sorted = this->anchor;
  Position current = this->anchor->getNext();
  sorted->setNext(nullptr);

  while (current != nullptr) {
    Position next = current->getNext();

    if (cmp(current->getData(), sorted->getData()) < 0) {
      current->setNext(sorted);
      sorted = current;
    } else {
      Position temp = sorted;
      while (temp->getNext() != nullptr &&
             cmp(temp->getNext()->getData(), current->getData()) <= 0) {
        temp = temp->getNext();
      }
      current->setNext(temp->getNext());
      temp->setNext(current);
    }

    current = next;
  }

  this->anchor = sorted;
}
```

```
template <class T>
void List<T>::sortDataSelect(int cmp(const T&, const T&)) {
  if (this->anchor == nullptr || this->anchor->getNext() == nullptr) {
    return;
  }

  Position current = this->anchor;

  while (current != nullptr) {
    Position minNode = current;
    Position scanner = current->getNext();

    while (scanner != nullptr) {
      if (cmp(scanner->getData(), minNode->getData()) < 0) {
        minNode = scanner;
      }
      scanner = scanner->getNext();
    }

    if (minNode != current) {
      T temp = current->getData();
      current->setData(minNode->getData());
      minNode->setData(temp);
    }

    current = current->getNext();
  }
}

template <class T>
void List<T>::sortDataShell(int cmp(const T&, const T&)) {
  if (this->anchor == nullptr || this->anchor->getNext() == nullptr) {
    return;
  }

  int series[] = {4181, 2584, 1597, 987, 610, 377, 233, 144, 89, 55,
                  34,   21,   13,   8,   5,   3,   2,   1,   0};

  int size = 0;
  Position temp = this->anchor;
  while (temp != nullptr) {
    size++;
    temp = temp->getNext();
  }

  int pos = 0;
  int gap = series[pos];
```

```
  while (gap > size) {
    gap = series[++pos];
  }

  while (gap > 0) {
    for (int i = gap; i < size; i++) {
      Position nodeI = this->getNodeAt(i);
      T tempData = nodeI->getData();

      int j = i;
      while (j >= gap) {
        Position nodeJMinusGap = this->getNodeAt(j - gap);

        if (cmp(nodeJMinusGap->getData(), tempData) > 0) {
          Position nodeJ = this->getNodeAt(j);
          nodeJ->setData(nodeJMinusGap->getData());
          j -= gap;
        } else {
          break;
        }
      }

      Position nodeJ = this->getNodeAt(j);
      nodeJ->setData(tempData);
    }

    gap = series[++pos];
  }
}

template <class T>
typename List<T>::Position List<T>::getNodeAt(int index) const {
  Position current = this->anchor;
  int count = 0;

  while (current != nullptr && count < index) {
    current = current->getNext();
    count++;
  }

  return current;
}

template <class T>
int List<T>::getLastPosition() const {
  if (this->isEmpty()) {
    return -1;
  }
```

```
  int count = 0;
  Position aux = this->anchor;
  while (aux->getNext() != nullptr) {
    aux = aux->getNext();
    count++;
  }
  return count;
}

// insertData con índice
template <class T>
void List<T>::insertData(const T& element, const int& index) {
  if (index < 0) {
    throw DataContainersExceptions::InvalidPosition();
  }

  if (index == 0) {
    this->insertData(element, nullptr);
    return;
  }

  Position prev = this->getNodeAt(index - 1);
  if (prev == nullptr) {
    throw DataContainersExceptions::InvalidPosition();
  }

  this->insertData(element, prev);
}

// deleteData con índice
template <class T>
void List<T>::deleteData(const int& index) {
  Position target = this->getNodeAt(index);
  if (target == nullptr) {
    throw DataContainersExceptions::InvalidPosition();
  }
  this->deleteData(target);
}

// retrieve con índice
template <class T>
T* List<T>::retrieve(const int& index) {
  Position target = this->getNodeAt(index);
  if (target == nullptr) {
    throw DataContainersExceptions::InvalidPosition();
  }
  return &(target->getData());
```

```
}

// findDataL - Búsqueda lineal simple
template <class T>
int List<T>::findDataL(const T& dataSearched) const {
  Position aux = this->anchor;
  int index = 0;

  while (aux != nullptr) {
    if (aux->getData() == dataSearched) {
      return index;
    }
    aux = aux->getNext();
    index++;
  }

  return -1;
}

// findDataL con comparador
template <class T>
int List<T>::findDataL(const T& dataSearched,
                       int cmp(const T&, const T&)) const {
  Position aux = this->anchor;
  int index = 0;

  while (aux != nullptr) {
    if (cmp(aux->getData(), dataSearched) == 0) {
      return index;
    }
    aux = aux->getNext();
    index++;
  }

  return -1;
}

// findDataB - Búsqueda binaria
template <class T>
int List<T>::findDataB(const T& dataSearched,
                       int cmp(const T&, const T&)) const {
  if (this->isEmpty()) {
    return -1;
  }

  int left = 0;
  int right = this->getLastPosition();
```

```
    while (left <= right) {
      int mid = left + (right - left) / 2;
      Position midNode = this->getNodeAt(mid);

      int comparison = cmp(midNode->getData(), dataSearched);

      if (comparison == 0) {
        return mid;
      } else if (comparison < 0) {
        left = mid + 1;
      } else {
        right = mid - 1;
      }
    }

    return -1;
}

// insertSortedData - Inserción en orden
template <class T>
void List<T>::insertSortedData(const T& element) {
    Position newNode = new Node(element);
    if (newNode == nullptr) {
      throw DataContainersExceptions::MemoryOverflow("Memoria No
Disponible");
    }

    if (this->anchor == nullptr || element < this->anchor->getData()) {
      newNode->setNext(this->anchor);
      this->anchor = newNode;
      return;
    }

    Position current = this->anchor;
    while (current->getNext() != nullptr &&
           current->getNext()->getData() < element) {
      current = current->getNext();
    }

    newNode->setNext(current->getNext());
    current->setNext(newNode);
}

template <class U>
std::ostream& operator<<(std::ostream& os, const List<U>& list) {
    typename List<U>::Position aux = list.anchor;
    int index = 0;
```

```
    while (aux != nullptr) {
      os << aux->getData();
      aux = aux->getNext();
      index++;
    }

    return os;
}

template <class U>
std::istream& operator>>(std::istream& is, List<U>& list) {
    U element;
    try {
      while (is >> element) {
        list.insertSortedData(element);
      }
    } catch (const std::invalid_argument& ex) {
    }

    return is;
}

#endif  // __LIST_H__
```

menu.hpp

```
#ifndef __MENU_H__
#define __MENU_H__

#include <fstream>
#include <iomanip>
#include <iostream>
#include <sstream>
#include <string>

#include "list.hpp"
#include "name.hpp"
#include "song.hpp"

class Menu {
 private:
  List<Song>& songList;

  void enterToContinue();
  int readInteger(std::string, const int&, const int&);
  Name readName(std::string);
  std::string readLinePrompt(const std::string&, bool = false);
  char readChar(const std::string&, const char*);

  bool handleOption(const std::string&);
  std::string windowHeader(const int&, const std::string&) const;
  std::string songTable(const int& = 10,
                        const int& = 35,
                        const int& = 30,
                        const int& = 25) const;

  void noDataMessage();

  void mainMenu();
  void insertSong();
  void deleteSong(const int&);
  void deleteAllSongs();
  void editSong(const int&);
  void exitProgram();

  void searchMenu();
  void searchBySongName();
  void searchByIntepreter();

  void sortMenu();
  void sortBySongName();
  void sortByInterpreter();
  void sortByRanking();
```

```
    void saveToDisk();
    void readFromDisk();

  public:
    Menu();
    Menu(const Menu&);
    Menu(List<Song>&);
};

#endif  // __MENU_H__
```

name.hpp

```
#ifndef __NAME_H__
#define __NAME_H__

#include <fstream>
#include <iostream>
#include <string>

#include "ownexceptions.hpp"

class Name {
 private:
  std::string first;
  std::string last;

 public:
  Name();
  Name(const Name&);
  Name(const std::string&, const std::string&);

  // Interfaz
  // Setter's
  void setFirst(const std::string&);
  void setLast(const std::string&);

  // Getter's
  std::string getFirst() const;
  std::string getLast() const;

  std::string toString() const;

  Name& operator=(const Name&);

  bool operator==(const Name&) const;
  bool operator!=(const Name&) const;
  bool operator<(const Name&) const;
  bool operator>(const Name&) const;
  bool operator<=(const Name&) const;
  bool operator>=(const Name&) const;

  int compareTo(const Name&) const;
  int static compare(const Name&, const Name&);

  friend std::ostream& operator<<(std::ostream&, const Name&);
  friend std::istream& operator>>(std::istream&, Name&);
};
#endif  // __NAME_H__
```

ownexceptions.hpp

```
#ifndef __OWNEXCEPTIONS_H__
#define __OWNEXCEPTIONS_H__

#include <stdexcept>
#include <string>

namespace DataContainersExceptions {
class MemoryDeficiency : public std::runtime_error {
 public:
  explicit MemoryDeficiency(const std::string& msg = "Insuficiencia de
Memoria")
      : std::runtime_error(msg) {}
};

class MemoryOverflow : public std::runtime_error {
 public:
  explicit MemoryOverflow(const std::string& msg = "Desbordamiento de
Memoria")
      : std::runtime_error(msg) {}
};

class InvalidPosition : public std::runtime_error {
 public:
  explicit InvalidPosition(
      const std::string& msg = "La posicion Ingresada es Invalida")
      : std::runtime_error(msg) {}
};
}  // namespace DataContainersExceptions

namespace InputExceptions {
class InvalidOption : public std::runtime_error {
 public:
  explicit InvalidOption(
      const std::string& msg = "La opcion ingresada esta fuera de rango")
      : runtime_error(msg) {}
};

class EmptyString : public std::runtime_error {
 public:
  explicit EmptyString(
      const std::string& msg = "El string no puede estar vacio")
      : runtime_error(msg) {}
};

class OperationCanceledException : public std::runtime_error {
 public:
  explicit OperationCanceledException(
```

```
        const std::string msg = "Operacion Cancelada")
      : runtime_error(msg) {}
};

class NumberNotPositive : public std::runtime_error {
 public:
  explicit NumberNotPositive(
      const std::string& msg = "Esta entrada debe ser mayor a 0")
      : runtime_error(msg) {}
};
}  // namespace InputExceptions

namespace DateExceptions {
class InvalidDate : public std::runtime_error {
 public:
  explicit InvalidDate(const std::string& msg = "La fecha es invalida")
      : runtime_error(msg) {}
};
}  // namespace DateExceptions

namespace RecipeExceptions {
class RepeatedIngredient : public std::runtime_error {
 public:
  explicit RepeatedIngredient(
      const std::string& msg = "El ingrediente ya esta registrado en la
lista.")
      : runtime_error(msg) {}
};

class NonExistenIngredient : public std::runtime_error {
 public:
  explicit NonExistenIngredient(
      const std::string& msg = "El ingrediente no existe en la lista.")
      : runtime_error(msg) {}
};
}  // namespace RecipeExceptions

namespace MenuExceptions {
class InvalidInsertCategory : std::runtime_error {
 public:
  explicit InvalidInsertCategory(
      const std::string& msg = "La categoria de inserción es inválida")
      : runtime_error(msg) {}
};
}  // namespace MenuExceptions

#endif  // __OWNEXCEPTIONS_H__
```

song.hpp

```
#ifndef __SONG_H__
#define __SONG_H__

#include <fstream>
#include <iomanip>
#include <iostream>
#include <sstream>
#include <string>

#include "name.hpp"

class Song {
 private:
  int ranking;
  std::string songName;
  Name author;
  Name interpreter;
  std::string mp3Name;

 public:
  Song();
  Song(const Song&);

  /// @brief
  /// @param  Ranking
  /// @param  NombreCancion
  /// @param  NombreAutor
  /// @param  NombreInterprete
  /// @param  NombreMP3
  Song(const int&,
       const std::string&,
       const Name&,
       const Name&,
       const std::string&);

  // Interfaz:
  // Setter's
  void setRanking(const int&);
  void setSongName(const std::string&);
  void setAuthor(const Name&);
  void setInterpreter(const Name&);
  void setMp3Name(const std::string&);

  // Getter's
  int getRanking() const;
  std::string getSongName() const;
  Name getAuthor() const;
```

```
    Name getInterpreter() const;
    std::string getMp3Name() const;

    /// @brief Función toString para la lsit.hpp
    /// @param  widthRanking
    /// @param  widthSongName
    /// @param  widthName
    /// @param  widthMP3
    std::string toString(const int& = 10,
                         const int& = 35,
                         const int& = 30,
                         const int& = 25) const;  // Para impresiones en
list.hpp

    /// @brief Función de 1 sola canción
    /// @param widthBorder
    /// @return
    std::string toStringOnly(
        const int& = 60) const;  // Para impresiones de solo 1 canción

    Song& operator=(const Song&);

    // Operadores Relacionales que utilizan el ranking como compardor
    bool operator==(const Song&) const;
    bool operator!=(const Song&) const;
    bool operator<(const Song&) const;
    bool operator>(const Song&) const;
    bool operator<=(const Song&) const;
    bool operator>=(const Song&) const;

    int compareTo(const Song&) const;
    static int compare(const Song&, const Song&);

    static int compareBySongName(const Song&, const Song&);
    static int compareByAutor(const Song&, const Song&);
    static int compareByInterpreter(const Song&, const Song&);
    static int compareByMP3Name(const Song&, const Song&);

    friend std::ostream& operator<<(std::ostream&, const Song&);
    friend std::istream& operator>>(std::istream&, Song&);
};
#endif  // __SONG_H__
```

Carpeta src

main.cpp

```
#include "menu.hpp"

int main() {
  new Menu(*new List<Song>);

  return 0;
}
```

menu.cpp

```
#include "menu.hpp"

using namespace std;

Menu::Menu() : songList(*new List<Song>) {
  mainMenu();
}

Menu::Menu(const Menu& other) : songList(other.songList) {
  mainMenu();
}

Menu::Menu(List<Song>& s) : songList(s) {
  mainMenu();
}

void Menu::enterToContinue() {
  cout << "[Enter] para continuar..." << endl;
  getchar();
}

int Menu::readInteger(string oss,
                      const int& lowerLimit,
                      const int& upperLimit) {
  string aux("");
  int result;
  while (true) {
    try {
      system("CLS");
      cout << oss;
      getline(cin, aux);
      result = stoi(aux);

      if (result > upperLimit || result < lowerLimit)
        throw InputExceptions::InvalidOption("Numero Fuera de Rango");
      break;
    } catch (const std::invalid_argument& ex) {
      system("CLS");
      cout << "Entrada invalida" << endl;
      cout << "Intente nuevamente" << endl;
      enterToContinue();
    } catch (const InputExceptions::InvalidOption& msg) {
      system("CLS");
      cout << msg.what() << endl;
      enterToContinue();
    }
  }
```

```
  return result;
}

Name Menu::readName(string prompt) {
  Name result;
  result.setFirst(readLinePrompt(prompt));
  prompt += result.getFirst() + "\n";
  result.setLast(readLinePrompt(prompt + "Ingrese el Apellido: "));

  return result;
}

string Menu::readLinePrompt(const string& prompt, bool allowEmpty) {
  string result;
  while (true) {
    system("CLS");
    cout << prompt;
    getline(cin, result);
    if (!allowEmpty && result.empty()) {
      system("CLS");
      cout << "No puede estar vacio.\nIntentelo nuevamente." << endl;
      enterToContinue();
      continue;
    }
    return result;
  }
}

char Menu::readChar(const std::string& prompt, const char* posibilities)
{
  char result, comparation;

  while (true) {
    int i = 0;
    system("CLS");
    cout << prompt;
    cin >> result;

    result = toupper(result);
    do {
      comparation = *(posibilities + i);
      if (result == comparation)
        return result;
      i++;
    } while (comparation != '\0');

    system("CLS");
```

```
      cout << "Opcion Invalida" << endl;
      cout << "Intentelo Nuevamente" << endl;
      system("PAUSE");
    }
}

string Menu::windowHeader(const int& widthBorder, const string& prompt)
const {
  ostringstream oss;

  oss << left << setfill('=') << setw(widthBorder) << "" << endl;
  oss << setfill(' ');

  // Título de Ventana
  oss << setw(widthBorder / 2 - (prompt.size() / 2)) << "| " << prompt
      << setw((widthBorder / 2) - (prompt.size() / 2) - 2) << "" << "|"
<< endl;
  oss << setfill('-') << setw(widthBorder) << "" << endl;
  oss << setfill(' ');

  return oss.str();
}

bool Menu::handleOption(const std::string& promt) {
  string response;

  system("CLS");
  cout << promt;
  getline(cin, response);

  if (response.empty())
    return true;

  // Hacer las letras mayúsculas
  char option =
      static_cast<char>(std::toupper(static_cast<unsigned
char>(response[0])));

  // buscar primer dígito después de la letra (saltando espacios)
  std::size_t pos = 1;
  while (pos < response.size() &&
         std::isspace(static_cast<unsigned char>(response[pos])))
    ++pos;

  bool hasNumber = false;
  int index = -1;
  if (pos < response.size() &&
      std::isdigit(static_cast<unsigned char>(response[pos]))) {
```

```
    std::size_t start = pos;
    std::size_t end = start;
    while (end < response.size() &&
           std::isdigit(static_cast<unsigned char>(response[end])))
      ++end;
    std::string numstr = response.substr(start, end - start);
    try {
      index = std::stoi(numstr);
      hasNumber = true;
    } catch (...) {
      hasNumber = false;
    }
  }

  switch (option) {
    case 'A':
      this->insertSong();
      break;

    case 'B':
      if (!hasNumber) {
        system("CLS");
        std::cout << "Falta numero de posicion. Ej: B2\n";
        this->enterToContinue();
        break;
      }
      if (!this->songList.isValidPosition(index)) {
        system("CLS");
        std::cout << "Posicion de lista invalida\n";
        this->enterToContinue();
        break;
      }
      this->editSong(index);
      break;

    case 'C':
      if (!hasNumber) {
        system("CLS");
        std::cout << "Falta numero de posicion. Ej: C12\n";
        this->enterToContinue();
        break;
      }
      if (!this->songList.isValidPosition(index)) {
        system("CLS");
        std::cout << "Posicion de lista invalida\n";
        this->enterToContinue();
        break;
      }
```

```
        this->deleteSong(index);
      break;

    case 'D':
      if (!hasNumber) {
        system("CLS");
        std::cout << "Falta numero de posiciin. Ej: D12\n";
        this->enterToContinue();
        break;
      }
      if (!this->songList.isValidPosition(index)) {
        system("CLS");
        std::cout << "Posicion de lista invalida\n";
        this->enterToContinue();
        break;
      }
      system("CLS");
      {
        List<Song>::Position pos = this->songList.getNodeAt(index);
        if (pos != nullptr) {
          Song& s = this->songList.retrieve(pos);
          std::cout << s.toStringOnly();
        } else {
          std::cout << "Cancion no encontrada\n";
        }
      }
      this->enterToContinue();
      break;
    case 'E':
      this->deleteAllSongs();
      break;
    case 'F':
      this->saveToDisk();
      break;
    case 'G':
      this->readFromDisk();
      break;
    case 'H':
      this->searchMenu();
      break;
    case 'I':
      this->sortMenu();
      break;
    case 'J':
      this->exitProgram();
      return false;

    default:
```

```
        system("CLS");
        std::cout << "Comando invalido\nIntentelo nuevamente.\n";
        enterToContinue();
        break;
    }  // switch

    return true;
}

std::string Menu::songTable(const int& widthRanking,
                            const int& widthSongName,
                            const int& widthName,
                            const int& widthMP3) const {
    ostringstream oss;

    int widthBorder =
        widthRanking + widthSongName + (widthName * 2) + widthMP3 + 16;

    oss << windowHeader(widthBorder, "LISTA DE EXITOS");
    oss << "| " << "# En Lista" << setw(3) << "| " << "Ranking"
        << setw(widthRanking - 7) << "| " << "Nombre de la Cancion"
        << setw(widthSongName - 20) << "| " << "Nombre del Artista"
        << setw(widthName - 18) << "| " << "Nombre del Interprete"
        << setw(widthName - 21) << "| " << "Nombre del MP3" <<
setw(widthMP3 - 14)
        << "|" << endl;

    oss << setfill('-');
    oss << setw(widthBorder) << " ";
    oss << setfill(' ') << endl;

    oss << this->songList.toString();

    oss << setfill('-');
    oss << setw(widthBorder) << " ";
    oss << setfill(' ');
    oss << endl;

    return oss.str();
}

void Menu::noDataMessage() {
    cout << "+------------------------------------------------------+" <<
endl;
    cout << "+            No hay Canciones Registradas Aun          +" <<
endl;
    cout << "+              Regresando al Menu...                   +" <<
endl;
```

```
    cout << "+-------------------------------------------------------+" <<
endl;
  this->enterToContinue();
}

void Menu::mainMenu() {
  ostringstream oss;
  bool running = true;

  while (running) {
    system("CLS");

    // Limpiar el ostringstream
    oss.str("");
    oss.clear();

    oss << this->songTable();
    oss << "Opciones: \n";
    oss << "[A] Agregar una Cancion.  [B<n>] Editar una Cancion [C<n>] "
           "Eliminar "
           "una Cancion. [D<n>]  Mostrar Detalles de
Cancion.  [E]    Eliminar "
           "Todas las Canciones. \n\n"
           "[F] Guardar la Database   [G] Leer del Disco   "
           "[H] Buscar una Cancion    [I] Ordenar Lista  [J] Salir.\n\n";
    oss << "Seleccione un Comando: ";

    running = handleOption(oss.str());
  }
}

void Menu::insertSong() {
  int widthBorder = 100;
  Song newSong;
  string myString("");
  int myInt(0);
  Name myName;
  ostringstream oss;

  do {
    system("CLS");
    // Linea Exterior
    oss << windowHeader(widthBorder, "INSERTAR EXITO");

    oss << "Ingrese el Nombre de la Cancion: ";
    myString = this->readLinePrompt(oss.str(), false);
    newSong.setSongName(myString);
    oss << newSong.getSongName() << endl;
```

```
oss << "Ingrese el Ranking de la Cancion: ";

while (true) {
  try {
    system("CLS");
    myInt = readInteger(oss.str(), 0, 3000);
    newSong.setRanking(myInt);
    if (songList.findData(newSong) != nullptr)
      throw std::invalid_argument("Ranking ya utilizado");
    break;
  } catch (const InputExceptions::InvalidOption& msg) {
    system("CLS");
    cout << msg.what() << endl;
    enterToContinue();
  } catch (const std::invalid_argument& msg) {
    system("CLS");
    cout << msg.what() << endl;
    enterToContinue();
  }
}

oss << newSong.getRanking() << endl;
oss << "Ingrese el Nombre del Autor: ";
myName = readName(oss.str());
oss << myName.getFirst() << endl;
oss << "Ingrese el Apellido: " << myName.getLast() << endl;

newSong.setAuthor(myName);

oss << "Ingrese el Nombre del Interprete: ";
myName = readName(oss.str());
oss << myName.getFirst() << endl;
oss << "Ingrese el Apellido: " << myName.getLast() << endl;

newSong.setInterpreter(myName);

oss << "Ingrese el nombre del Archivo MP3: ";
myString = this->readLinePrompt(oss.str(), false);
newSong.setMp3Name(myString);
oss << newSong.getMp3Name() << endl;

if (songList.isEmpty())
  songList.insertData(newSong, 0);
else {
  oss << "Ingrese la posicion en la lista que tendra la cancion: ";
  while (true) {
    try {
```

```
            myInt = readInteger(oss.str(), 0, 49);
            songList.insertData(newSong, myInt);
            oss << myInt << endl;
            break;
          } catch (const DataContainersExceptions::InvalidPosition& msg) {
            system("CLS");
            cout << msg.what() << endl;
            cout << "Intente Nuevamente." << endl;
            enterToContinue();
          } catch (const DataContainersExceptions::MemoryOverflow& msg) {
            system("CLS");
            cout << msg.what() << endl;
            cout << "Regresando..." << endl;
            enterToContinue();
            return;
          }
        }
      }

      oss << "Cancion Agregada con Exito!." << endl;
      oss << "Desea Agregar Otra Cancion? (1. Si / 2. No): ";

      myInt = readInteger(oss.str(), 1, 2);

      oss.str("");
      oss.clear();
    } while (myInt != 2);
}

void Menu::deleteSong(const int& position) {
    system("CLS");
    ostringstream oss;
    int response;

    List<Song>::Position pos = songList.getNodeAt(position);
    if (pos == nullptr) {
      cout << "Posicion invalida" << endl;
      enterToContinue();
      return;
    }

    Song& target = songList.retrieve(pos);
    oss << target.toStringOnly();
    oss << "Esta seguro que desea eliminar esta cancion? (1. Si/ 2. No): ";
    response = readInteger(oss.str(), 1, 2);

    if (response == 1) {
      songList.deleteData(pos);
```

```
      oss << endl << "Cancion Eliminada con Exito!" << endl;
    } else {
      oss << endl << "Operacion Cancelada" << endl;
    }

    system("CLS");
    cout << oss.str();
    enterToContinue();
}

void Menu::deleteAllSongs() {
    system("CLS");
    if (songList.getLastPosition() == -1) {
      cout << "Aun no hay canciones para eliminar" << endl;
      enterToContinue();
      return;
    }

    ostringstream oss;
    int widhtBorder = 50;

    oss << windowHeader(widhtBorder, "ELIMINAR TODAS LAS CANCIONES");

    oss << "Esta seguro que desea eliminar las " <<
songList.getLastPosition() + 1
        << " canciones? (1. Si/ 2. No): ";
    int response = readInteger(oss.str(), 1, 2);
    system("CLS");
    if (response == 1) {
      songList.deleteAll();
      cout << "Canciones eliminadas con Exito!" << endl;
      cout << "Base de Datos Vacia." << endl;
    } else {
      cout << "Operacion Cancelada." << endl;
    }
    enterToContinue();
}

void Menu::editSong(const int& position) {
    ostringstream oss;

    List<Song>::Position pos = songList.getNodeAt(position);
    if (pos == nullptr) {
      system("CLS");
      cout << "Posicion invalida" << endl;
      enterToContinue();
      return;
    }
```

```
Song& target = songList.retrieve(pos);
int editOption, newRanking;
string dataString;
Name newName;
Song ver;

oss << target.toStringOnly();
oss << "5 Salir\n";

editOption = readInteger(
    oss.str() + "Elige el atributo que quieras cambiar (1-5): ", 1, 5);

switch (editOption) {
  case 1:
    oss << "Ingrese el Nuevo Ranking de la Cancion: ";
    newRanking = readInteger(oss.str(), 1, 999999);
    ver.setRanking(newRanking);
    if (this->songList.findData(ver) != nullptr) {
      system("CLS");
      cout << "El ranking ya esta ocupado" << endl;
      enterToContinue();
      break;
    }
    target.setRanking(newRanking);
    cout << "Cambio hecho con Exito!";
    break;
  case 2:
    oss << "Ingrese el nuevo nombre de la cancion: ";
    dataString = readLinePrompt(oss.str());
    target.setSongName(dataString);
    cout << "Cambio hecho con Exito!";
    enterToContinue();
    break;
  case 3:
    oss << "Ingrese el nuevo autor de la cancion: ";
    newName = readName(oss.str());
    target.setAuthor(newName);
    cout << "Cambio hecho con Exito!";
    break;
  case 4:
    oss << "Ingrese el nuevo interprete de la cancion: ";
    newName = readName(oss.str());
    target.setInterpreter(newName);
    cout << "Cambio hecho con Exito!";
    break;
  case 5:
    return;
```

```
    default:
      break;
  }
  enterToContinue();
}

void Menu::exitProgram() {
  system("CLS");
  int response;
  ostringstream oss;
  if (!this->songList.isEmpty()) {
    oss << windowHeader(50, "SALIR SIN GUARDAR?");
    response = readInteger(
        oss.str() +
            "Desea Guardar las canciones antes de Salir? (1. Si/ 2. No):
",
        1, 2);
    if (response == 1)
      saveToDisk();
  }

  system("CLS");
  std::cout << "Saliendo del Programa.\nTenga un Lindo Dia :D\n";
  enterToContinue();
}

void Menu::searchMenu() {
  system("CLS");
  if (this->songList.isEmpty()) {
    this->noDataMessage();
    return;
  }

  ostringstream oss;
  char op;
  oss << windowHeader(50, "BUSCAR CANCION");
  oss << "Existen un total de: " << this->songList.getLastPosition() + 1
      << " registradas." << endl;
  oss << "A continuacion se muestran las opciones de busqueda: " << endl;
  oss << "[A] Buscar por Nombre de Cancion" << endl
      << "[B] Buscar por Nombre del Inteprete" << endl
      << "[R] Regresar." << endl
      << "Seleccione una Opcion: ";

  while (op != 'R') {
    system("CLS");
    cout << oss.str();
    cin >> op;
```

```
    cin.ignore();

    op = toupper(op);

    switch (op) {
      case 'A':
        this->searchBySongName();
        break;
      case 'B':
        this->searchByIntepreter();
        break;
      case 'R':
        system("CLS");
        cout << "Regresando...";
        enterToContinue();
        break;
      default:
        system("CLS");
        cout << "Opcion invalida" << endl;
        cout << "Intentelo nuevamente" << endl;
        enterToContinue();
        break;
    }
  }
}

void Menu::searchBySongName() {
  List<Song> songWithTheName, auxList = this->songList;
  ostringstream oss;
  char response, options[2] = {'B', 'L'};
  string songName;
  Song searchedSong;
  int position, repeat;

  do {
    oss.str("");
    oss.clear();
    system("CLS");
    oss << windowHeader(50, "BUSCAR POR NOMBRE DE CANCION");
    oss << "Ingrese el nombre de la cancion que quiera buscar: ";

    songName = readLinePrompt(oss.str(), false);
    searchedSong.setSongName(songName);
    oss << songName << endl;

    oss << "Desea Realizar Una Busqueda Binaria o Lineal? (L/B): ";

    response = readChar(oss.str(), options);
```

```
     cin.ignore();
    oss << response << endl;

    if (response == 'L') {
      try {
        while (true) {
          position = auxList.findDataL(searchedSong,
Song::compareBySongName);
          List<Song>::Position pos = auxList.getNodeAt(position);
          songWithTheName.insertSortedData(auxList.retrieve(pos));
          auxList.deleteData(position);
        }
      } catch (const DataContainersExceptions::InvalidPosition& ex) {
        // No hacer nada
      }
    } else {
      try {
        while (true) {
          position = auxList.findDataB(searchedSong,
Song::compareBySongName);
          songWithTheName.insertSortedData(*auxList.retrieve(position));
          auxList.deleteData(position);
        }
      } catch (const DataContainersExceptions::InvalidPosition& ex) {
        // No hacer nada
      }
    }

    if (songWithTheName.isEmpty())
      oss << "\nNo existe un registro de una cancion llamada: " <<
songName
          << endl;
    else {
      oss.str("");
      oss.clear();
      oss << windowHeader(146, "LISTA DE EXITOS");
      oss << "| " << "# En Lista" << setw(3) << "| " << "Ranking" <<
setw(3)
          << "| " << "Nombre de la Cancion" << setw(15) << "| "
          << "Nombre del Artista" << setw(12) << "| " << "Nombre del
Interprete"
          << setw(9) << "| " << "Nombre del MP3" << setw(11) << "|" <<
endl;

      oss << setfill('-');
      oss << setw(146) << " ";
      oss << setfill(' ') << endl;
      oss << songWithTheName.toString();
```

```
      oss << setfill('-');
      oss << setw(146) << "";
      oss << setfill(' ') << endl;
    }

    songWithTheName.deleteAll();
    oss << "Desea Realizar Otra Busqueda?: (1.Si / 2.No): ";

    repeat = readInteger(oss.str(), 1, 2);

  } while (repeat != 2);
  system("CLS");
  cout << "Regresando..." << endl;
  enterToContinue();
}

void Menu::searchByIntepreter() {
  List<Song> songsOfInterpreter;
  List<Song> auxList = this->songList;
  ostringstream oss;
  char response, options[2] = {'B', 'L'};
  string dataString;
  Name searchedName;
  Song searchedSong;
  int position, repeat;

  do {
    oss.str("");
    oss.clear();
    system("CLS");
    oss << windowHeader(70, "BUSCAR POR INTERPRETE DE LA CANCION");

    oss << "Ingrese el Nombre del Interprete: ";
    searchedName = readName(oss.str());
    oss << searchedName.getFirst() << endl;
    oss << "Ingrese el Apellido: " << searchedName.getLast() << endl;

    searchedSong.setInterpreter(searchedName);

    oss << "Desea Realizar Una Busqueda Binaria o Lineal? (L/B): ";

    response = readChar(oss.str(), options);
    cin.ignore();
    oss << response << endl;

    if (response == 'L') {
      try {
```

```
          while (true) {
            position =
                auxList.findDataL(searchedSong,
Song::compareByInterpreter);
            songsOfInterpreter.insertSortedData(*auxList.retrieve(position)
);
            auxList.deleteData(position);
          }
        } catch (const DataContainersExceptions::InvalidPosition& ex) {
          // No hacer nada
        }
      } else {
        try {
          while (true) {
            position =
                auxList.findDataB(searchedSong,
Song::compareByInterpreter);
            songsOfInterpreter.insertSortedData(*auxList.retrieve(position)
);
            auxList.deleteData(position);
          }
        } catch (const DataContainersExceptions::InvalidPosition& ex) {
          // No hacer nada
        }
      }

      if (songsOfInterpreter.isEmpty())
        oss << "\nNo existe un registro de una cancion del interprete: "
            << searchedName.toString() << endl;
      else {
        oss.str("");
        oss.clear();
        oss << windowHeader(146, "LISTA DE EXITOS");
        oss << "| " << "# En Lista" << setw(3) << "| " << "Ranking" <<
setw(3)
            << "| " << "Nombre de la Cancion" << setw(15) << "| "
            << "Nombre del Artista" << setw(12) << "| " << "Nombre del
Interprete"
            << setw(9) << "| " << "Nombre del MP3" << setw(11) << "|" <<
endl;

        oss << setfill('-');
        oss << setw(146) << " ";
        oss << setfill(' ') << endl;
        oss << songsOfInterpreter.toString();

        oss << setfill('-');
        oss << setw(146) << "";
```

```
        oss << setfill(' ') << endl;
      }
      songsOfInterpreter.deleteAll();
      oss << "Desea Realizar Otra Busqueda?: (1.Si / 2.No): ";

      repeat = readInteger(oss.str(), 1, 2);

    } while (repeat != 2);
    system("CLS");
    cout << "Regresando..." << endl;
    enterToContinue();
}

void Menu::sortMenu() {
    system("CLS");
    if (this->songList.isEmpty()) {
      this->noDataMessage();
      return;
    }

    if (this->songList.getLastPosition() == 0) {
      cout << "Solo hay 1 cancion registrada." << endl;
      cout << "No se requiere Ordenamiento." << endl;
      enterToContinue();
      return;
    }

    ostringstream oss;
    char op;
    oss << windowHeader(50, "ORDENAR EXITOS");
    oss << "Existen un total de: " << this->songList.getLastPosition() + 1
        << " registradas." << endl;
    oss << "A continuacion se muestran las opciones sobre las cuales
ordenar las "
           "canciones: "
        << endl;
    oss << "[A] Ordenar por Nombre de Cancion" << endl
        << "[B] Ordenar por Nombre del Inteprete" << endl
        << "[C] Ordenar por Numero de Ranking" << endl
        << "[R] Regresar." << endl
        << "Seleccione una Opcion: ";

    while (op != 'R') {
      char charsValid[] = {'A', 'B', 'C', 'R'};
      op = readChar(oss.str(), charsValid);
      cin.ignore();

      switch (op) {
```

```
        case 'A':
          this->sortBySongName();
          break;
        case 'B':
          this->sortByInterpreter();
          break;
        case 'C':
          this->sortByRanking();
          break;
        case 'R':
          system("CLS");
          cout << "Regresando...";
          enterToContinue();
          break;
      }
    }
}

void Menu::sortBySongName() {
  system("CLS");
  ostringstream oss;
  char op, validOptions[5] = {'A', 'B', 'C', 'D', 'E'};

  oss << windowHeader(100, "Ordenar por Nombre de la Cancion");
  oss << "A continuacion, eliga un algoritmo de ordenamiento para la
lista: "
      << endl;
  oss << "[A] Ordenamiento por Burbuja" << endl;
  oss << "[B] Ordenamiento por InsertSort" << endl;
  oss << "[C] Ordenamiento por SelectSort" << endl;
  oss << "[D] Ordenamiento por ShellSort" << endl;
  oss << "[E] Regresar" << endl << endl;
  oss << "Ingrese una Opcion: ";

  op = this->readChar(oss.str(), validOptions);
  cin.ignore();

  oss << op << endl;
  switch (op) {
    case 'A':
      this->songList.sortDataBubble(Song::compareBySongName);
      oss << "Ordenamiento por Burbuja hecho correctamente!." << endl;
      break;
    case 'B':
      this->songList.sortDataInsert(Song::compareBySongName);
      oss << "Ordenamiento por InserSor hecho correctamente!" << endl;
      break;
    case 'C':
```

```
        this->songList.sortDataSelect(Song::compareBySongName);
        oss << "Ordenamiento por DataSelect hecho correctamente!" << endl;
        break;
      case 'D':
        this->songList.sortDataShell(Song::compareBySongName);
        oss << "Ordenamiento por ShellSort hecho correctamente!" << endl;
        break;
      case 'E':
        system("CLS");
        cout << "Regresando..." << endl;
        enterToContinue();
        return;
        break;
    }

    system("CLS");
    oss << endl << "Regresando..." << endl << endl;
    cout << oss.str();
    enterToContinue();
}

void Menu::sortByInterpreter() {
    system("CLS");
    ostringstream oss;
    char op, validOptions[5] = {'A', 'B', 'C', 'D', 'E'};

    oss << windowHeader(100, "Ordenar por Interprete de la Cancion");
    oss << "A continuacion, eliga un algoritmo de ordenamiento para la
lista: "
        << endl;
    oss << "[A] Ordenamiento por Burbuja" << endl;
    oss << "[B] Ordenamiento por InsertSort" << endl;
    oss << "[C] Ordenamiento por SelectSort" << endl;
    oss << "[D] Ordenamiento por ShellSort" << endl;
    oss << "[E] Regresar" << endl << endl;
    oss << "Ingrese una Opcion: ";

    op = this->readChar(oss.str(), validOptions);
    cin.ignore();

    oss << op << endl;
    switch (op) {
      case 'A':
        this->songList.sortDataBubble(Song::compareByInterpreter);
        oss << "Ordenamiento por Burbuja hecho correctamente!." << endl;
        break;
      case 'B':
        this->songList.sortDataInsert(Song::compareByInterpreter);
```

```
        oss << "Ordenamiento por InserSor hecho correctamente!" << endl;
        break;
      case 'C':
        this->songList.sortDataSelect(Song::compareByInterpreter);
        oss << "Ordenamiento por DataSelect hecho correctamente!" << endl;
        break;
      case 'D':
        this->songList.sortDataShell(Song::compareByInterpreter);
        oss << "Ordenamiento por ShellSort hecho correctamente!" << endl;
        break;
      case 'E':
        system("CLS");
        cout << "Regresando..." << endl;
        enterToContinue();
        return;
        break;
    }

    system("CLS");
    oss << endl << "Regresando..." << endl << endl;
    cout << oss.str();
    enterToContinue();
}

void Menu::sortByRanking() {
    system("CLS");
    ostringstream oss;
    char op, validOptions[5] = {'A', 'B', 'C', 'D', 'E'};

    oss << windowHeader(100, "Ordenar por Ranking de la Cancion");
    oss << "A continuacion, eliga un algoritmo de ordenamiento para la
lista: "
        << endl;
    oss << "[A] Ordenamiento por Burbuja" << endl;
    oss << "[B] Ordenamiento por InsertSort" << endl;
    oss << "[C] Ordenamiento por SelectSort" << endl;
    oss << "[D] Ordenamiento por ShellSort" << endl;
    oss << "[E] Regresar" << endl << endl;
    oss << "Ingrese una Opcion: ";

    op = this->readChar(oss.str(), validOptions);
    cin.ignore();

    oss << op << endl;
    switch (op) {
      case 'A':
        this->songList.sortDataBubble();
        oss << "Ordenamiento por Burbuja hecho correctamente!." << endl;
```

```
        break;
      case 'B':
        this->songList.sortDataInsert();
        oss << "Ordenamiento por InserSor hecho correctamente!" << endl;
        break;
      case 'C':
        this->songList.sortDataSelect();
        oss << "Ordenamiento por DataSelect hecho correctamente!" << endl;
        break;
      case 'D':
        this->songList.sortDataShell();
        oss << "Ordenamiento por ShellSort hecho correctamente!" << endl;
        break;
      case 'E':
        system("CLS");
        cout << "Regresando..." << endl;
        enterToContinue();
        return;
        break;
    }

    system("CLS");
    oss << endl << "Regresando..." << endl << endl;
    cout << oss.str();
    enterToContinue();
}

void Menu::saveToDisk() {
    system("CLS");
    ostringstream oss;

    if (this->songList.isEmpty()) {
        cout << "+-----------------------------------------------------+"
<< endl;
        cout << "+            No hay Canciones Registradas Aun         +"
<< endl;
        cout << "+              Regresando al Menu...                  +"
<< endl;
        cout << "+-----------------------------------------------------+"
<< endl;
        enterToContinue();
        return;
    }

    int widthBorder = 50;
    string fileName("");
    ofstream file;
```

```
    oss << windowHeader(widthBorder, "GUARDAR DATABASE");

    oss << "Ingrese el Nombre que Tendra el Archivo: ";
    fileName = readLinePrompt(oss.str());

    file.open(fileName, ios_base::trunc);

    if (!file.is_open())
      oss << "No se permite la creacion de archivos." << endl;
    else {
      file << this->songList;
      oss << "Database guardada con Exito!" << endl;
    }

    system("CLS");
    cout << oss.str();
    enterToContinue();
}

void Menu::readFromDisk() {
    system("CLS");
    ostringstream oss;
    int widthBorder = 100;
    ifstream file;
    string fileName;

    oss << windowHeader(widthBorder, "LEER ARCHIVO");

    oss << "Tenga en Cuenta que los Archivos se Sobreescribiran" << endl;
    oss << "Ingrese el Nombre del Archivo a Cargar sus Datos: ";

    fileName = readLinePrompt(oss.str());
    oss << fileName << endl;

    file.open(fileName);

    if (!file.is_open())
      oss << "El archivo no existe o no pudo ser abierto" << endl;
    else {
      this->songList.deleteAll();
      file >> this->songList;
      oss << "Archivos Cargados Con Exito!" << endl;
    }

    oss << setfill('=') << setw(widthBorder) << "" << endl;
    oss << setfill(' ');

    system("CLS");
```

```
    cout << oss.str();
    enterToContinue();
}
```

name.cpp

```
#include "name.hpp"

Name::Name() : first("default"), last("default") {}

Name::Name(const Name& other) : first(other.first), last(other.last) {}

Name::Name(const std::string& f, const std::string& l) : first(f),
last(l) {}

void Name::setFirst(const std::string& first) {
  if (first.empty())
    throw InputExceptions::EmptyString(
        "Nombre no puede estar vacío, setFirst(Name)");
  this->first = first;
}

void Name::setLast(const std::string& last) {
  if (last.empty())
    throw InputExceptions::EmptyString(
        "Apellido no puede estar vacío, setLast(Name)");
  this->last = last;
}

std::string Name::getFirst() const {
  return this->first;
}

std::string Name::getLast() const {
  return this->last;
}

std::string Name::toString() const {
  return this->first + " " + this->last;
}

Name& Name::operator=(const Name& other) {
  this->first = other.first;
  this->last = other.last;

  return *this;
}

bool Name::operator==(const Name& other) const {
  return this->toString() == other.toString();
}

bool Name::operator!=(const Name& other) const {
```

```
    return !(*this == other);
}

bool Name::operator<(const Name& other) const {
  return this->toString() < other.toString();
}

bool Name::operator>(const Name& other) const {
  return this->toString() > other.toString();
}

bool Name::operator<=(const Name& other) const {
  return (*this < other) || (*this == other);
}

bool Name::operator>=(const Name& other) const {
  return (*this > other) || (*this == other);
}

int Name::compareTo(const Name& other) const {
  return this->toString().compare(other.toString());
}

int Name::compare(const Name& nameA, const Name& nameB) {
  return nameA.toString().compare(nameB.toString());
}

std::ostream& operator<<(std::ostream& os, const Name& name) {
  os << name.first << "," << name.last;

  return os;
}

std::istream& operator>>(std::istream& is, Name& name) {
  std::string dataString;
  getline(is, dataString, ',');
  name.first = dataString;
  getline(is, dataString, ',');
  name.last = dataString;

  return is;
}
```

song.cpp

```cpp
#include "song.hpp"

using namespace std;

Song::Song()
    : ranking(-1),
      songName("default"),
      author(),
      interpreter(),
      mp3Name("default") {}

Song::Song(const Song& other)
    : ranking(other.ranking),
      songName(other.songName),
      author(other.author),
      interpreter(other.interpreter),
      mp3Name(other.mp3Name) {}

Song::Song(const int& r,
           const std::string& n,
           const Name& a,
           const Name& i,
           const std::string& m)
    : ranking(r), songName(n), author(a), interpreter(i), mp3Name(m) {}

void Song::setRanking(const int& ranking) {
  if (ranking <= 0)
    throw InputExceptions::InvalidOption("El ranking debe ser positivo");
  this->ranking = ranking;
}

void Song::setSongName(const std::string& songName) {
  if (songName.empty())
    throw InputExceptions::EmptyString("El nombre no puede estar
vacio.");
  this->songName = songName;
}

void Song::setAuthor(const Name& author) {
  this->author = author;  // Name tiene sus propias validaciones
}

void Song::setInterpreter(const Name& interpreter) {
  this->interpreter = interpreter;
}

void Song::setMp3Name(const std::string& mp3Name) {
```

```
    if (mp3Name.empty())
      throw InputExceptions::EmptyString("El nombre no puede estar vacio");
    this->mp3Name = mp3Name;
}

int Song::getRanking() const {
    return this->ranking;
}

std::string Song::getSongName() const {
    return this->songName;
}

Name Song::getAuthor() const {
    return this->author;
}

Name Song::getInterpreter() const {
    return this->interpreter;
}

std::string Song::getMp3Name() const {
    return this->mp3Name;
}

std::string Song::toStringOnly(const int& widthBorder) const {
    ostringstream oss;

    oss << setfill('=') << setw(widthBorder) << "" << endl;
    oss << setfill(' ');

    oss << "| " << setw((widthBorder / 2) + 10) << "INFORMACION DE LA
CANCION"
        << setw((widthBorder / 2) - 12) << "|" << endl;

    oss << setfill('-') << setw(widthBorder) << "" << endl;
    oss << setfill(' ');

    oss << "Posicion en el Ranking: " << ranking << endl;
    oss << "Nombre de la Cancion: " << songName << endl;
    oss << "Nombre del Autor: " << author.toString() << endl;
    oss << "Nombre del Inteprete: " << interpreter.toString() << endl;
    oss << "Nombre del Archivo MP3: " << mp3Name << endl;

    oss << endl << setfill('=') << setw(widthBorder) << "" << endl;
    oss << setfill(' ');

    return oss.str();
```

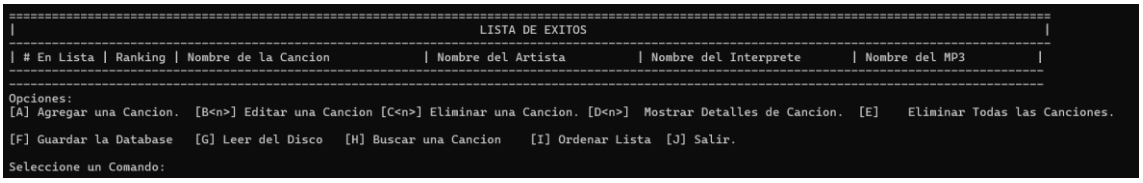
```
}

std::string Song::toString(const int& widthRanking,
                           const int& widthSongName,
                           const int& widthName,
                           const int& widthMP3) const {
  ostringstream oss;

  oss << "| " << this->ranking
      << setw(widthRanking - to_string(this->ranking).size()) << "| "
      << this->songName << setw(widthSongName - this->songName.size()) <<
"| "
      << this->author.toString()
      << setw(widthName - this->author.toString().size()) << "| "
      << this->interpreter.toString()
      << setw(widthName - this->interpreter.toString().size()) << "| "
      << this->mp3Name << setw(widthMP3 - this->mp3Name.size()) << "|";

  return oss.str();
}

Song& Song::operator=(const Song& other) {
  this->ranking = other.ranking;
  this->songName = other.songName;
  this->author = other.author;
  this->interpreter = other.interpreter;
  this->mp3Name = other.mp3Name;

  return *this;
}

bool Song::operator==(const Song& other) const {
  return this->ranking == other.ranking;
}

bool Song::operator!=(const Song& other) const {
  return !(*this == other);
}

bool Song::operator<(const Song& other) const {
  return this->ranking < other.ranking;
}

bool Song::operator>(const Song& other) const {
  return this->ranking > other.ranking;
}

bool Song::operator<=(const Song& other) const {
```

```
    return !(*this > other);
}

bool Song::operator>=(const Song& other) const {
  return !(*this < other);
}

int Song::compareTo(const Song& other) const {
  return this->ranking - other.ranking;
}

int Song::compare(const Song& songA, const Song& songB) {
  return songA.ranking - songB.ranking;
}

int Song::compareBySongName(const Song& songA, const Song& songB) {
  return songA.songName.compare(songB.songName);
}

int Song::compareByAutor(const Song& songA, const Song& songB) {
  return songA.author.compareTo(songB.author);
}

int Song::compareByInterpreter(const Song& songA, const Song& songB) {
  return songA.interpreter.compareTo(songB.interpreter);
}

int Song::compareByMP3Name(const Song& songA, const Song& songB) {
  return songA.mp3Name.compare(songB.mp3Name);
}

std::ostream& operator<<(std::ostream& os, const Song& song) {
  os << song.ranking << "," << song.songName << "," << song.author << ","
     << song.interpreter << "," << song.mp3Name;

  return os;
}

std::istream& operator>>(std::istream& is, Song& song) {
  string dataString;
  getline(is, dataString, ',');
  song.ranking = stoi(dataString);
  getline(is, song.songName, ',');
  is >> song.author;
  is >> song.interpreter;
  getline(is, song.mp3Name);

  return is;}
```

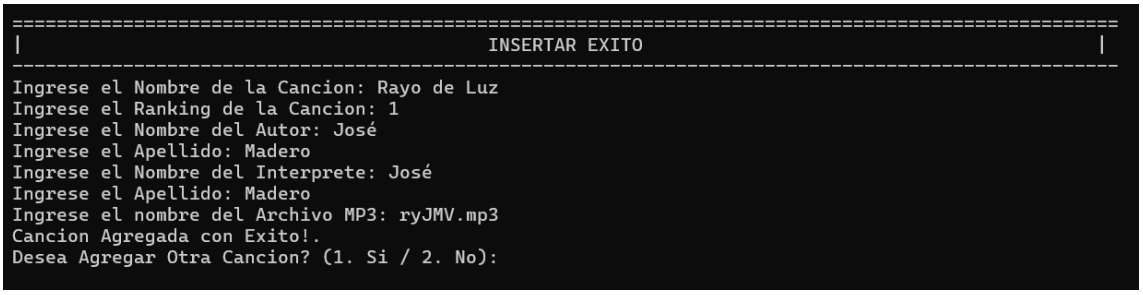
Ejecución del Programa

La ejecución de este programa para el usuario no cambia, así que las tomas de pantalla serán similares a tareas anteriores, pero muestran que el programa funciona adecuadamente con las modificaciones hechas en la lista.

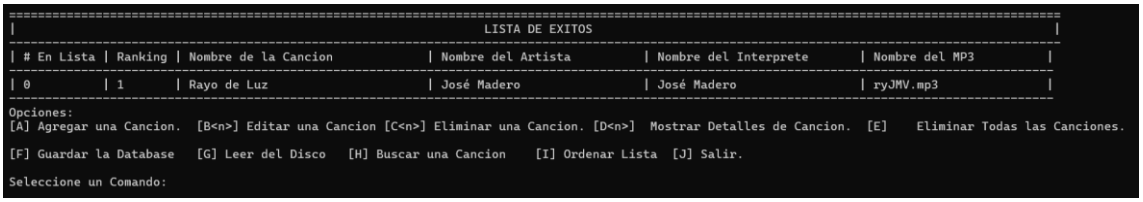
Ejecución del menú principal



El proceso de agregar una canción sigue correcto:



Y se refleja al ir al menú principal:



Para leer una database sigue reconociendo el mismo formato:

Que sí se reflejan correctamente:

```
=================================================================================================================
|                                                LISTA DE EXITOS                                                |
-----------------------------------------------------------------------------------------------------------------
| # En Lista | Ranking | Nombre de la Cancion          | Nombre del Artista | Nombre del Interprete | Nombre del MP3          |
-----------------------------------------------------------------------------------------------------------------
| 0          | 235     | Karmadame                     | León Larregui      | Zoé Zoé               | karmadame_zoe.mp3,      |
| 1          | 302     | Luna                          | León Larregui      | Zoé Zoé               | luna_zoe.mp3,           |
| 2          | 315     | Labios Rotos                  | León Larregui      | Zoé Zoé               | labiosrotos_zoe.mp3,    |
| 3          | 456     | Narcisista por Excelencia     | José Madero        | José Madero           | narc_ex_jm.mp3,         |
| 4          | 789     | El Duelo                      | León Larregui      | Zoé Zoé               | elduelo_zoe.mp3,        |
| 5          | 904     | Los Malaventurados No Lloran  | José Madero        | José Madero           | malav_jm.mp3,           |
| 6          | 1023    | Mercedes                      | José Madero        | José Madero           | mercedes_jm.mp3,        |
| 7          | 1222    | Rayo de Luz                   | José Madero        | José Madero           | rayoluz_jm.mp3,         |
| 8          | 1421    | Fiebre                        | León Larregui      | Zoé Zoé               | fiebre_zoe.mp3,         |
| 9          | 1502    | No Hay Mal Que Dure           | León Larregui      | Zoé Zoé               | nomal_dure_zoe.mp3,     |
| 10         | 1551    | Love                          | León Larregui      | Zoé Zoé               | love_zoe.mp3,           |
| 11         | 1582    | Nada Personal                 | Luis Hernández     | Luis Hernández        | nada_pers.mp3,          |
| 12         | 1673    | Día de Mayo                   | José Madero        | José Madero           | diademayo_jm.mp3,       |
| 13         | 1678    | Tokyo                         | Laura Pergolizzi   | LP Pergolizzi         | tokyo_lp.mp3,           |
| 14         | 1742    | Monster (Zoé cover)           | León Larregui      | Zoé Zoé               | monster_zoe.mp3,        |
| 15         | 1755    | Hablemos del Campo            | José Madero        | José Madero           | habcamp_jm.mp3,         |
| 16         | 1760    | Earned It                     | Abel Tesfaye       | The Weeknd            | earned_tw.mp3,          |
| 17         | 1764    | Save Your Tears               | Abel Tesfaye       | The Weeknd            | save_tw.mp3,            |
| 18         | 1876    | Quince Mil Días               | José Madero        | José Madero           | quincemil_jm.mp3,       |
| 19         | 1888    | Monster                       | Abel Tesfaye       | The Weeknd            | monster_tw.mp3,         |
| 20         | 1901    | Gasolina (cover)              | Laura Pergolizzi   | LP Pergolizzi         | gasolina_lp.mp3,        |
| 21         | 1977    | Belong To The World (cover)   | Laura Pergolizzi   | LP Pergolizzi         | belong_lp.mp3,          |
| 22         | 1987    | Lunes 28                      | José Madero        | José Madero           | lunes28_jm.mp3,         |
| 23         | 1988    | Callaíta (cover)              | Abel Tesfaye       | The Weeknd            | callaita_tw.mp3,        |
| 24         | 1989    | Lost On You                   | Laura Pergolizzi   | LP Pergolizzi         | lost_lp.mp3,            |
| 25         | 2022    | Midnight City (cover)         | Laura Pergolizzi   | LP Pergolizzi         | midnight_lp.mp3,        |
| 26         | 2099    | Gardenias 87                  | José Madero        | José Madero           | gardenias_jm.mp3,       |
| 27         | 2111    | Strange                       | Laura Pergolizzi   | LP Pergolizzi         | strange_lp.mp3,         |
| 28         | 2115    | Nueve Vidas                   | José Madero        | José Madero           | nuevevidas_jm.mp3,      |
| 29         | 2122    | The Hills                     | Abel Tesfaye       | The Weeknd            | hills_tw.mp3,           |
| 30         | 2177    | Hit Me Hard                   | The Weeknd         | The Weeknd            | hitme_tw.mp3,           |
| 31         | 2203    | Campeones del Mundo           | José Madero        | José Madero           | campeones_jm.mp3,       |
| 32         | 2228    | Home                          | The Weeknd         | The Weeknd            | home_tw.mp3,            |
| 33         | 2244    | Popular                       | León Larregui      | Zoé Zoé               | popular_zoe.mp3,        |
| 34         | 2250    | In Your Eyes                  | Abel Tesfaye       | The Weeknd            | inyoureyes_tw.mp3,      |
| 35         | 2307    | What You Need                 | The Weeknd         | The Weeknd            | whatneed_tw.mp3,        |
```

Editar una canción sigue el mismo proceso:

```
=============================================================
|                  INFORMACION DE LA CANCION                |
-------------------------------------------------------------
Posicion en el Ranking: 235
Nombre de la Cancion: Karmadame
Nombre del Autor: León Larregui
Nombre del Inteprete: Zoé Zoé
Nombre del Archivo MP3: karmadame_zoe.mp3,

=============================================================
5 Salir
Elige el atributo que quieras cambiar (1-5):
```

```
=============================================================
|                  INFORMACION DE LA CANCION                |
-------------------------------------------------------------
Posicion en el Ranking: 235
Nombre de la Cancion: Karmadame
Nombre del Autor: León Larregui
Nombre del Inteprete: Zoé Zoé
Nombre del Archivo MP3: karmadame_zoe.mp3,

=============================================================
5 Salir
Ingrese el Nuevo Ranking de la Cancion: 5
Cambio hecho con Exito![Enter] para continuar...
```

Que sí se reflejan en el menú:

```
================================================================================================================================
|                                                        LISTA DE EXITOS                                                       |
--------------------------------------------------------------------------------------------------------------------------------
| # En Lista | Ranking | Nombre de la Cancion            | Nombre del Artista | Nombre del Interprete | Nombre del MP3          |
--------------------------------------------------------------------------------------------------------------------------------
| 0          | 5       | Karmadame                       | León Larregui      | Zoé Zoé               | karmadame_zoe.mp3,      |
| 1          | 302     | Luna                            | León Larregui      | Zoé Zoé               | luna_zoe.mp3,           |
| 2          | 315     | Labios Rotos                    | León Larregui      | Zoé Zoé               | labiosrotos_zoe.mp3,    |
| 3          | 456     | Narcisista por Excelencia       | José Madero        | José Madero           | narc_ex_jm.mp3,         |
| 4          | 789     | El Duelo                        | León Larregui      | Zoé Zoé               | elduelo_zoe.mp3,        |
| 5          | 904     | Los Malaventurados No Lloran    | José Madero        | José Madero           | malav_jm.mp3,           |
```

Para mostrar información imprime correctamente:

```
============================================================
|                 INFORMACION DE LA CANCION                |
------------------------------------------------------------
Posicion en el Ranking: 904
Nombre de la Cancion: Los Malaventurados No Lloran
Nombre del Autor: José Madero
Nombre del Inteprete: José Madero
Nombre del Archivo MP3: malav_jm.mp3,

============================================================
[Enter] para continuar...
```

El guardado al disco:

```
================================================
|               GUARDAR DATABASE               |
------------------------------------------------
Ingrese el Nombre que Tendra el Archivo: Database guardada con Exito!
[Enter] para continuar...
```

La búsqueda:

```
================================================
|                 BUSCAR CANCION               |
------------------------------------------------
Existen un total de: 45 registradas.
A continuacion se muestran las opciones de busqueda:
[A] Buscar por Nombre de Cancion
[B] Buscar por Nombre del Inteprete
[R] Regresar.
Seleccione una Opcion:
```

```
==================================================
|           BUSCAR POR NOMBRE DE CANCION         |
--------------------------------------------------
Ingrese el nombre de la cancion que quiera buscar: Soñé
Desea Realizar Una Busqueda Binaria o Lineal? (L/B): L
```

(Aún funcionando con la binaria con sus debidas advertencias):

```
=================================================================================================================
|                                                LISTA DE EXITOS                                                |
-----------------------------------------------------------------------------------------------------------------
| # En Lista | Ranking | Nombre de la Cancion            | Nombre del Artista          | Nombre del Interprete      | Nombre del MP3          |
-----------------------------------------------------------------------------------------------------------------
| 0          | 2503    | Soñé                            | León Larregui               | Zoé Zoé                    | sone_zoe.mp3,           |
-----------------------------------------------------------------------------------------------------------------
Desea Realizar Otra Busqueda?: (1.Si / 2.No):
```

El ordenamiento:

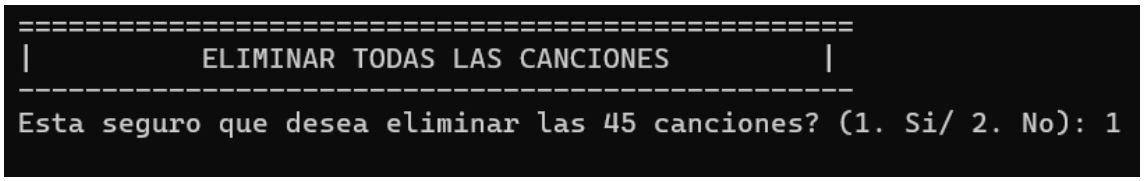
```
==================================================
|                  ORDENAR EXITOS                |
--------------------------------------------------
Existen un total de: 45 registradas.
A continuacion se muestran las opciones sobre las cuales ordenar las canciones:
[A] Ordenar por Nombre de Cancion
[B] Ordenar por Nombre del Inteprete
[C] Ordenar por Numero de Ranking
[R] Regresar.
Seleccione una Opcion:
```

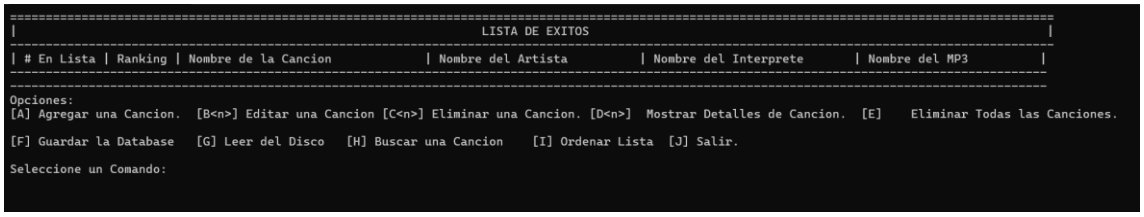
```
=================================================================================================================
|                                    Ordenar por Nombre de la Cancion                                           |
-----------------------------------------------------------------------------------------------------------------
A continuacion, eliga un algoritmo de ordenamiento para la lista:
[A] Ordenamiento por Burbuja
[B] Ordenamiento por InsertSort
[C] Ordenamiento por SelectSort
[D] Ordenamiento por ShellSort
[E] Regresar

Ingrese una Opcion: A
Ordenamiento por Burbuja hecho correctamente!.

Regresando...

[Enter] para continuar...
```

Que sí se relejan:

```
=================================================================================================================
|                                                LISTA DE EXITOS                                                |
-----------------------------------------------------------------------------------------------------------------
| # En Lista | Ranking | Nombre de la Cancion            | Nombre del Artista          | Nombre del Interprete      | Nombre del MP3          |
-----------------------------------------------------------------------------------------------------------------
| 0          | 2999    | After Hours                     | Abel Tesfaye                | The Weeknd                 | afterh_tw.mp3,          |
| 1          | 1977    | Belong To The World (cover)     | Laura Pergolizzi            | LP Pergolizzi              | belong_lp.mp3,          |
| 2          | 2345    | Blinding Lights                 | Abel Tesfaye                | The Weeknd                 | blinding_tw.mp3,        |
| 3          | 1988    | Callaita (cover)                | Abel Tesfaye                | The Weeknd                 | callaita_tw.mp3,        |
| 4          | 2203    | Campeones del Mundo             | José Madero                 | José Madero                | campeones_jm.mp3,       |
| 5          | 2420    | Dafne                           | José Madero                 | José Madero                | dafne_jm.mp3,           |
| 6          | 1673    | Día de Mayo                     | José Madero                 | José Madero                | diademayo_jm.mp3,       |
| 7          | 1760    | Earned It                       | Abel Tesfaye                | The Weeknd                 | earned_tw.mp3,          |
| 8          | 789     | El Duelo                        | León Larregui               | Zoé Zoé                    | elduelo_zoe.mp3,        |
| 9          | 1421    | Fiebre                          | León Larregui               | Zoé Zoé                    | fiebre_zoe.mp3,         |
| 10         | 2433    | Friends                         | Abel Tesfaye                | The Weeknd                 | friends_tw.mp3,         |
| 11         | 2320    | Gardenias                       | José Madero                 | José Madero                | gardenias2_jm.mp3,      |
| 12         | 2099    | Gardenias 87                    | José Madero                 | José Madero                | gardenias_jm.mp3,       |
| 13         | 1901    | Gasolina (cover)                | Laura Pergolizzi            | LP Pergolizzi              | gasolina_lp.mp3,        |
| 14         | 1755    | Hablemos del Campo              | José Madero                 | José Madero                | habcamp_jm.mp3,         |
| 15         | 2410    | Heartless                       | Abel Tesfaye                | The Weeknd                 | heartless_tw.mp3,       |
| 16         | 2177    | Hit Me Hard                     | The Weeknd                  | The Weeknd                 | hitme_tw.mp3,           |
| 17         | 2228    | Home                            | The Weeknd                  | The Weeknd                 | home_tw.mp3,            |
| 18         | 2250    | In Your Eyes                    | Abel Tesfaye                | The Weeknd                 | inyoureyes_tw.mp3,      |
| 19         | 5       | Karmadame                       | León Larregui               | Zoé Zoé                    | karmadame_zoe.mp3,      |
| 20         | 315     | Labios Rotos                    | León Larregui               | Zoé Zoé                    | labiosrotos_zoe.mp3,    |
| 21         | 2860    | Liminal (Zoé)                   | León Larregui               | Zoé Zoé                    | liminal_zoe.mp3,        |
| 22         | 904     | Los Malaventurados No Lloran    | José Madero                 | José Madero                | malav_jm.mp3,           |
```

La eliminación completa:





En general, el programa tiene el comportamiento esperado

Conclusiones

Realizare esta actividad representa un paso importante en la comprensión profunda de las estructuras de datos dinámicas, en especial de la Lista Simplemente Ligada, uno de los fundamentos más importantes de las estructuras de datos. El diseño e implementación de este programa fue interesante por el hecho del manejo de la posición como antes no lo había hecho y por una manera diferente de ver la lista

Desde una perspectiva técnica, la implementación dinámica aporta ventajas sustanciales frente a estructuras estáticas, como la posibilidad de gestionar eficientemente el crecimiento del número de canciones sin necesidad de conocer su tamaño de antemano, un requerimiento esencial en el problema planteado.

En cuanto a la funcionalidad del sistema se mantuvo a pesar de las repetidas advertencias a ciertas funcionalidades del programa, pero esto solo demuestra que por que algo pueda implementarse no significa que deba, o que algoritmos más sencillos pueden ser mejores en ciertos casos que los que nos dicen que son los que dominan.

Finalmente, esta actividad fue enriquecedora, pero aún falta mucho por conocer de las listas, desde practicar más con las implementaciones circulares de una lista hasta conocer los headers y las listas doblemente enlazadas para solucionar y hacer más eficientes muchas de las prácticas que ya tenemos en las listas.