Diego Javier Lemus Girón – 1211621

Sebastian Echeverria - 1138122

Documento PDF que contiene el análisis y diseño (algoritmo) del programa, incluyendo al menos:

(02 pts.) ¿Qué acciones debe poder hacer su programa? Enumérelas.

-Mostrar errores del programa, si está vacío, en formato de línea incorrecta o si hay líneas duplicadas -Agregar canción

-Ver Cola de Reproducción

-Reproducción Actual

-Reproducir Siguiente

-Ordenar canciones

(02 pts.) ¿Con qué datos va a trabajar? ¿Qué información debe pedir al usuario?, defina sus datos de entrada y el tipo de dato que utilizará para los datos principales.

Para empezar, tiene que elegir entre una de estas opciones:

1. Cargar respaldos

Introduce la ruta de los respaldos (sin espacios, para rutas con espacios, use comillas) o si no existe saldrá “No se encontraron archivos en la ruta especificada.”

2. Agregar Canción

-Seleccione un CD:

-Seleccione una canción:

-Canción agregada a la cola.

3. Ver Cola de Reproducción

Si hay algo en la loca, saldrá cada canción y si está vacía saldrá: “Cola de reproducción vacía.”

4. Ordenar Cola de Reproducción

Se puede elegir entre "1. Ordenar por artista", "2. Ordenar por canción", "3. Ordenar por duración" y si se escribe una opción no existente, saldrá que es una “opción no valida”.

5. Reproducir Cancion Actual Reproduciendo:

O la opción de “Reproducción en pausa.”

6. Reproducir Siguiente Canción

Se reproducirá la siguiente canción y si la cola esta vacía, se mostrar un mensaje de “Cola de reproducción vacía.”

7. Salir

(02 pts.) ¿Qué estructuras de datos trabajará para almacenar la información?

Estructuras de datos, árboles, listas, pilas y colas.

(04 pts.) ¿Qué condiciones o restricciones debe tomar en cuenta? ¿Qué cálculos debe hacer?

Que no esté ningún campo vacío cuando se pide algún dato, que este bien escrito la ruta del archivo, que se seleccione una canción correctamente para agregarla, en la opción de ordenar cola de reproducción que se seleccione una opción existente y en reproducir siguiente que si haya alguna canción.

(10 pts.) Algoritmo para mostrar las funciones principales que debe realizar el programa

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
#include <fstream>
#include <filesystem>
#include <fstream>
#include <sstream> using
namespace std;
 class Cancion {
public:     string
nombre;     string
artista;     string
duracion;

    Cancion() {}
    Cancion(string _nombre, string _artista, string _duracion) :
nombre(_nombre), artista(_artista), duracion(_duracion) {}
};  class CD { public:
string nombreCD;
vector<Cancion> canciones;



};
class Reproductor {
vector<CD> CDs;
vector<Cancion> cola;
    Cancion* reproduciendo = nullptr;
 public:
    Reproductor() {}
void cargarRespaldos() {
        cout << "Introduce la ruta de los respaldos (sin espacios, para rutas
con espacios, use comillas): ";         string ruta;
        cin.ignore(); // Limpiar el búfer para getline
getline(cin, ruta);

        vector<string> archivos = listarArchivos(ruta);
if (archivos.empty()) {
            cout << "No se encontraron archivos en la ruta especificada." <<
endl;
            return;
        }
        for (const auto& archivo : archivos) {
            cout << "Procesando archivo: " << archivo << endl; // Mensaje de
depuración
            CD cd = cargarCDDesdeArchivo(ruta + "/" + archivo);
if (!cd.canciones.empty()) {                 CDs.push_back(cd);
                cout << "CD " << cd.nombreCD << " cargado con éxito." << endl;
// Mensaje de confirmación
```

```
            }
else {
                cout << "Error al cargar CD desde archivo: " << archivo <<
endl; // Mensaje de error
            }
        }
    }
    CD cargarCDDesdeArchivo(const string& rutaArchivo) {
ifstream archivo(rutaArchivo);         if
(!archivo.is_open()) {
            throw runtime_error("No se pudo abrir el archivo " + rutaArchivo);
}

        CD cd;
cd.nombreCD =
filesystem::path(rutaArchivo).filename().replace_extension().string();

        string linea;
        while (getline(archivo, linea)) {
size_t posInicio = 0;
            size_t posFin = linea.find("||");

            string nombre, artista, duracion;

            // Verifica si se encontró "||" en la línea
if (posFin != string::npos) {                  //
Extraer nombre
                nombre  =  linea.substr(posInicio,  posFin  -  posInicio);
posInicio = posFin + 2;  // Avanzar el índice de inicio más allá de '||'

                // Encontrar la siguiente ocurrencia de '||'
posFin = linea.find("||", posInicio);

                // Extraer artista
                artista  =  linea.substr(posInicio,  posFin  -  posInicio);
posInicio = posFin + 2;  // Avanzar el índice de inicio más allá de '||'

                // Extraer duración
                duracion = linea.substr(posInicio);
            }

            Cancion cancion;
cancion.nombre = nombre;
cancion.artista = artista;
cancion.duracion = duracion;

            cd.canciones.push_back(cancion);
        }
        archivo.close();
return cd;
    }
```

```
    bool formatoValido(const string& linea) {         int
Count = count(linea.begin(), linea.end(), '|');         return
Count == 4 && linea.find("||") != string::npos;
    }
    void mostrarCDs() {
        for (int i = 0; i < CDs.size(); ++i) {
            cout << i + 1 << ". " << CDs[i].nombreCD << endl;
        }
    }
    void agregarCancion() {
mostrarCDs();         int
eleccion;
        cout << "Seleccione un CD: ";
cin >> eleccion;
        if (eleccion <= 0 || eleccion > CDs.size()) {
cout << "Selección no válida." << endl;
return;
        }

        CD& cdElegido = CDs[eleccion - 1];
        for (int i = 0; i < cdElegido.canciones.size(); ++i) {
            cout << i + 1 << ". " << cdElegido.canciones[i].nombre << endl;
        }
        cout << "Seleccione una canción: ";
cin >> eleccion;
        if (eleccion <= 0 || eleccion > cdElegido.canciones.size()) {
cout << "Selección no válida." << endl;             return;
        }
        cola.push_back(cdElegido.canciones[eleccion - 1]);
cout << "Canción agregada a la cola." << endl;
    }
    void verColaReproduccion() {
        // Implementar lógica para mostrar la cola de reproducción
if (cola.empty()) {
            std::cout << "Cola de reproducción vacía." << std::endl;
return;
        }
        for (const auto& cancion : cola) {
            std::cout << cancion.nombre << " - " << cancion.artista << " - " <<
cancion.duracion << std::endl;
        }
    }
    void reproducirActual() {
        // Implementar lógica para reproducir la canción actual
if (reproduciendo) {
            cout << "Reproduciendo: " << reproduciendo->nombre << " - " <<
reproduciendo->artista << " - " << reproduciendo->duracion << endl;
        }
else {
            cout << "Reproducción en pausa." << endl;
        }
    }

    void reproducirSiguiente() {
```

```
        // Implementar lógica para reproducir la siguiente canción
if (cola.empty()) {
            cout << "Cola de reproducción vacía." << endl;
return;
        }
        if (reproduciendo) {
            // Hacer una copia independiente de la canción antes de eliminarla
de la cola
            *reproduciendo = cola.front();
        }
else {
            reproduciendo = new Cancion(cola.front()); // O usa el operador de
asignación si está definido
        }
        cola.erase(cola.begin());
    }
    void ordenarCola() {
        // Implementar lógica para ordenar la cola
cout << "1. Ordenar por artista\n";         cout
<< "2. Ordenar por canción\n";         cout << "3.
Ordenar por duración\n";         int eleccion;
cin >> eleccion;

        switch (eleccion) {
case 1:
            sort(cola.begin(), cola.end(), [](const Cancion& a, const Cancion&
b) {
                return a.artista < b.artista;
                });
break;          case 2:
            sort(cola.begin(), cola.end(), [](const Cancion& a, const Cancion&
b) {
                return a.nombre < b.nombre;
                });
break;          case 3:
            sort(cola.begin(), cola.end(), [](const Cancion& a, const Cancion&
b) {
                return a.duracion < b.duracion;
                });             break;
default:              cout << "Opción no válida."
<< endl;              break;
        }
}
    vector<string> listarArchivos(const string& ruta) {
vector<string> archivos;

        for (const auto& entry : filesystem::directory_iterator(ruta)) {
            if (entry.is_regular_file() && entry.path().extension() == ".txt")
{
                archivos.push_back(entry.path().filename().string());
            }
        }
        return archivos;
    }
```

```
};
int main() {
    Reproductor reproductor;

    int opcion = 0;
     do
{
        cout << "\n--- Menu ---" << endl;         cout <<
"1. Cargar respaldos" << endl;         cout << "2. Agregar
Cancion" << endl;         cout << "3. Ver Cola de
Reproduccion" << endl;         cout << "4. Ordenar Cola de
Reproduccion" << endl;         cout << "5. Reproducir
Cancion Actual" << endl;         cout << "6. Reproducir
Siguiente Cancion" << endl;         cout << "7. Salir" <<
endl;

        cout << "Seleccione una opcion: ";
while (!(cin >> opcion)) {
            cout << "Por favor, ingrese un número válido: ";
cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        }
        switch (opcion) {         case 1:
reproductor.cargarRespaldos();             break;         case 2:
reproductor.agregarCancion();             break;         case 3:
reproductor.verColaReproduccion();             break;         case 4:
reproductor.ordenarCola();             break;         case 5:
reproductor.reproducirActual();             break;         case 6:
reproductor.reproducirSiguiente();             break;         case 7:
break;         default:             cout << "Opción no válida. Por favor,
intente nuevamente." << endl;             break;
        }

        // Pausar y esperar que el usuario presione enter para continuar
cout << "\nPresione ENTER para continuar...";
cin.ignore(numeric_limits<streamsize>::max(), '\n');         cin.get();

    } while (opcion != 7);

    return 0; }
```

Canción.h

```
#pragma once
#include <iostream>
#include <fstream>
#include <vector>
#include <sstream>
```

```
#include <algorithm>
#include <filesystem>
#include <string> using
namespace std;

class Cancion
{ public:     string
nombre;     string
artista;     string
duracion;

    Cancion(string n, string a, string d) : nombre(n), artista(a), duracion(d)
{}
};
```

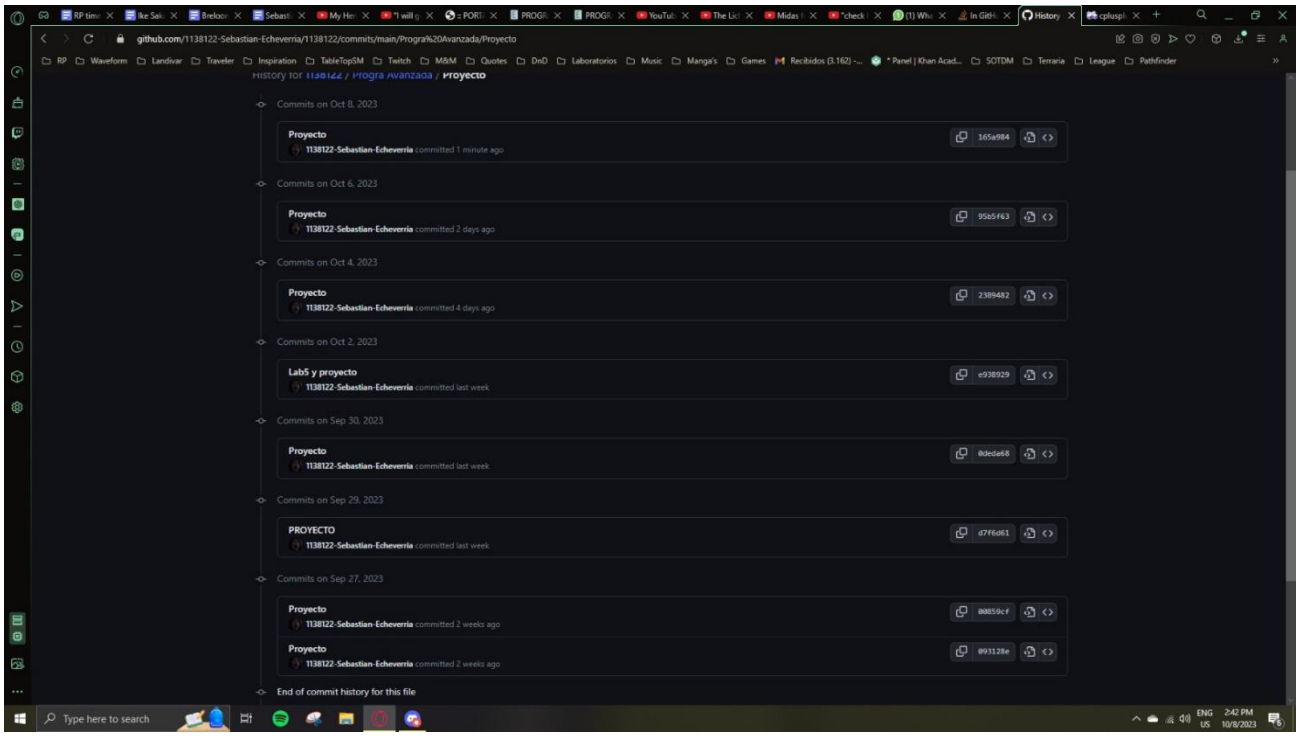
CD.h

```
#pragma once
#include <iostream>
#include <fstream>
#include <vector>
#include <sstream>
#include <algorithm>
#include <filesystem>
#include <string> #include
"Cancion.h" using
namespace std;
 class
CD {
public:
string
nombreCD
;
vector<C
ancion>
cancione
s;

    CD(string n) : nombreCD(n) {}

    void agregarCancion(const Cancion& cancion) {
canciones.push_back(cancion);
    }
};
```

Commits realizados dentro de github:


github.com/1138122-Sebastian-Echeverria/1138122/commits/main/Progra%20Avanzada/Proyecto
History for 1138122 / Progra Avanzada / Proyecto
Commits on Oct 8, 2023
Proyecto
1138122-Sebastian-Echeverria committed 1 minute ago
165a984
Commits on Oct 6, 2023
Proyecto
1138122-Sebastian-Echeverria committed 2 days ago
9505f63
Commits on Oct 4, 2023
Proyecto
1138122-Sebastian-Echeverria committed 4 days ago
2389482
Commits on Oct 2, 2023
Lab5 y proyecto
1138122-Sebastian-Echeverria committed last week
e938929
Commits on Sep 30, 2023
Proyecto
1138122-Sebastian-Echeverria committed last week
0deda68
Commits on Sep 29, 2023
PROYECTO
1138122-Sebastian-Echeverria committed last week
d7f6d61
Commits on Sep 27, 2023
Proyecto
1138122-Sebastian-Echeverria committed 2 weeks ago
80859cf
Proyecto
1138122-Sebastian-Echeverria committed 2 weeks ago
093128e
End of commit history for this file
Type here to search