Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

ОТЧЕТ

к практической работе

на тему:

Учебная практика

|  |  |  |  |
| --- | --- | --- | --- |
| Выполнил  Студент гр. 351001 |  | Головко Р.С. | |
| Проверил |  | Фадеева Е. Е. | |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Минск, 2024

**СОДЕРЖАНИЕ**

[1 ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ 3](#_Toc168534917)

[1.1 Теоретические сведения по теме лабораторной работы 3](#_Toc168534918)

[1.2 Постановка задачи 4](#_Toc168534919)

[1.3 Эскиз ожидаемого результата 4](#_Toc168534920)

[2 ВЫПОЛНЕНИЕ 6](#_Toc168534921)

[2.1 Алгоритм работы программы 6](#_Toc168534922)

[2.2 Алгоритм чтения файлов 6](#_Toc168534923)

[2.3 Алгоритм отображения списков 6](#_Toc168534924)

[2.4 Алгоритм добавления элемента в список 6](#_Toc168534925)

[2.5 Алгоритм удаления элемента из списка 7](#_Toc168534926)

[2.6 Алгоритм редактирования элемента списка 7](#_Toc168534927)

[2.7 Алгоритм поиска элементов в списке 7](#_Toc168534928)

[2.8 Алгоритм сортировки списка 7](#_Toc168534929)

[2.9 Алгоритм специальной функции 8](#_Toc168534930)

[2.10 Алгоритм сохранения в файл 8](#_Toc168534931)

[2.11 Определение основных подпрограмм и их описание 8](#_Toc168534932)

[2.12 Выделение основных структур данных 10](#_Toc168534933)

[2.13 Тестирование и отладка программы 11](#_Toc168534934)

[2.14 Руководство по эксплуатации 11](#_Toc168534935)

[ПРИЛОЖЕНИЕ А 17](#_Toc168534936)

[ПРИЛОЖЕНИЕ Б 25](#_Toc168534952)

[ПРИЛОЖЕНИЕ В 35](#_Toc168534953)

# ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ

## Теоретические сведения по теме лабораторной работы

**Запись –** это структура данных, состоящая в общем случае из иерархически упорядоченных разнородных компонентов. В отличие от массивов компоненты записей могут иметь различные типы, и доступ к ним осуществляется не по индексам, а по именам.

Компоненты записей называются **полями.** На тип поля записи ограничений не накладывается, поэтому компонентой записи может быть тоже запись. В этом случае говорят об иерархической записи. Уровень иерархии (вложенности) не должен превышать 9.

**Динамические переменные** – это переменные, которые порождаются и уничтожаются в процессе выполнения программы, размер значений которых (область отводимой памяти) определяется и может изменяться при выполнении программы.

Для работы с динамическими переменными в языке Delphi предусмотрен **ссылочный тип (тип указатель)**. Значением этого типа является **ссылка** на какой-либо элемент программы. По данной ссылке осуществляется непосредственный доступ к этому элементу. В качестве ссылки используется адрес соответствующего элемента в памяти машины.

**Связный список** — базовая динамическая структура данных, состоящая из узлов, содержащих данные и ссылки («связки») на следующий и/или предыдущий узел списка.

**Логический файл** – это произвольная последовательность элементов, длина которой заранее не определена, а конкретизируется в процессе выполнения программы.

**Физический файл** (набор данных) – это поименованная область памяти на внешнем носителе, в которой хранится некоторая информация (файл с точки зрения пользователя).

**Виды переменных файлового типа:**

* Текстовые файлы
* Типизированные файлы
* Не типизированные файлы

**Типизированный файл —** это файл, в котором записаны идентичные структуры. Типизированный файл можно создать только программным путем. Он состоит из записей фиксированной длины. Содержимое файла — это машинное представление данных.

**Текстовый файл ­–** текстовые файлы обеспечивают простой и удобный способ сохранения текстовых данных. Они обеспечивают простой механизм для чтения и записи численных данных, сохраненных как текст.

## Постановка задачи

Мной было разработано техническое задание, представленное в **Приложении А.**

## Эскиз ожидаемого результата

Вид пользовательского меню:

Программа для работы со списками, связанными с музыкой

Меню приложения:

1. Прочитать данные из файла.

2. Просмотреть списки.

3. Сортировать данные.

4. Найти данные в списках.

5. Добавить данные в списки.

6. Удалить данные из списков.

7. Редактировать списки.

8. Создать Playlist.

9. Выйти из программы без сохранения.

10. Выйти из программы с сохранением.

После выбора пункта 1 должно быть отражено подменю чтения файла.

Меню чтения из файла.

1 .\files\Session

Выберите действие:

1. Выбрать существующую сессию.

2. Создать новую сессию.

При выборе пунктов меню, соответствующее действие которых может быть применено к различным объектам (различным спискам) будет выведено подменю выбора соответствующего объекта (списка).

Меню подпункта просмотра списков:

1. Просмотреть список исполнителей.

2. Просмотреть список альбомов.

3. Просмотреть список песен.

0. Выйти из подпункта меню.

Вывод информации об объектах списка должно быть отображено следующим образом (список альбомов):

|-------------|-----------------|----------------------|------------|

| Код альбома | Код исполнителя | Название альбома | Год записи |

|-------------|-----------------|----------------------|------------|

| 1 | 1 | Человеко-часы | 2020 |

| 2 | 1 | Радио огонь | 2016 |

| 3 | 2 | Без правил! | 2017 |

| 4 | 2 | Недошуток | 2015 |

| 5 | 3 | Back In Black | 1980 |

| 6 | 3 | Highway to Hell | 1979 |

| 7 | 4 | Ну и что, что я вор? | 2022 |

| 8 | 4 | Мой байк | 2022 |

| 9 | 5 | In A Time Lapse | 2013 |

| 10 | 5 | Una Mattina | 2004 |

| 11 | 6 | Холостяк | 2015 |

| 12 | 7 | Мокрые кроссы | 2019 |

| 13 | 7 | Альфа и Омега | 2019 |

| 14 | 8 | Night Visions | 2012 |

| 15 | 8 | Origins | 2018 |

| 16 | 8 | Bones | 2022 |

| 17 | 9 | t11 | 1 |

| 18 | 9 | t12 | 6 |

| 19 | 10 | t21 | 2 |

| 20 | 10 | t22 | 7 |

| 21 | 11 | t31 | 3 |

| 22 | 11 | t32 | 8 |

| 23 | 12 | t41 | 4 |

| 24 | 12 | t42 | 9 |

| 25 | 13 | t51 | 5 |

| 26 | 13 | t52 | 10 |

| 27 | 14 | t61 | 6 |

|-------------|-----------------|----------------------|------------|

Выбор способа взаимодействия со списком через значение определенного поля элемента должно быть отображено следующим образом (меню подпункта поиска в списке исполнителей):

Меню подпункта поиска в списке исполнителей:

1. Поиск по коду исполнителя.

2. Поиск по имени исполнителя.

3. Поиск по стране исполнителя.

4. Поиск по направлению исполнителя.

0. Выйти из подпункта меню.

При выборе пункта добавления данных в список будет выведено подменю выбора списка для добавления. Затем информация, которая будет вводится в качестве данных элементов списка будет постепенно выведена на экран (вставка исполнителя в список):

Введите имя исполнителя: Пример

Введите страну исполнителя: Пример

Введите направление песен исполнителя: Пример

При выборе пункта создания playlist-а будет выведено меню для ввода информации, на основе которой необходимо создать playlist-ы. Затем будут выведены и сохранены в текстовый файл количество и все созданные playlist-ы:

ВНИМАНИЕ, ЕСЛИ PLAYLIST УЖЕ БЫЛ СОЗДАН, ТО ЕГО ДАННЫЕ БУДУТ УТЕРЯНЫ.

Меню создания playlist-ов:

Введите направление исполнителя: test

Введите длину Playlist-а в формате: чч:мм:сс.

(Если часы и/или минуты равны 0-ю, 0-и необходимо записать, пример:

00:00:45 / 00:45:00).

Длина: 00:01:00

Введите год, с которого выбирать песни: 1

Количество playlist-ов: 2

|-----------|----------------------|-------------|--------------------|

| Код песни | Название песни | Код альбома | Длительность песни |

|-----------|----------------------|-------------|--------------------|

| 41 | t612 | 27 | 60 |

|-----------|----------------------|-------------|--------------------|

|-----------|----------------------|-------------|--------------------|

| Код песни | Название песни | Код альбома | Длительность песни |

|-----------|----------------------|-------------|--------------------|

| 42 | t613 | 27 | 60 |

|-----------|----------------------|-------------|--------------------|

# ВЫПОЛНЕНИЕ

**Разработка алгоритма по ГОСТ 19.701-90**

Схемы разработанных алгоритмов представлены в **Приложении Б**.

## Алгоритм работы программы

Работа программы построена на основе работы отдельных вспомогательных алгоритмов в зависимости от выбора пункта меню пользователем. После отображения основного меню программа ожидает ввод пользователем номера действия, которое в последствии будет выполнено.

Данный алгоритм представлен в **Приложении Б, схема Б.1.**

## Алгоритм чтения файлов

Выполнение алгоритма выполнения разделено на 2 этапа. Первый этап представляет собой поэлементное чтение файла, второй – создание списка на основе прочитанных данных. В случае уже прочитанного файла повторное чтение запрещено.

Данный алгоритм представлен в **Приложении Б, схема Б.2.**

## Алгоритм отображения списков

В зависимости от выбора пользователем списка для отображения будет выведена вся информация об элементах, находящимся в данном списке. Для этого в цикле, начиная с первого элемента в списке, будет перемещен указатель на последующие элементы, при этом выводя информацию, об элементе, на который ссылается данный указатель.

Данный алгоритм представлен в **Приложении Б, схема Б.3.**

## Алгоритм добавления элемента в список

При добавлении элемента в список выделяется область памяти и заполняется данными, введенными пользователем. Указатель на новый элемент располагается за «головой» (списки исполнителей и песен) или в конце (список альбомов) списка. Указатель на оставшуюся часть списка помещается за вставленным элементом.

Данный алгоритм представлен в **Приложении Б, схемы Б.4, Б.5.**

## Алгоритм удаления элемента из списка

При удалении элемента из списка осуществляется поиск элемента по коду, введенному пользователем. В случае нахождения элемента, указатель предыдущего элемента списка на следующий элемент будет заменен на указатель элемента, следующего за удаляемым, и память, занятая удаляемым элементом, будет освобождена.

Данный алгоритм представлен в **Приложении Б, схемы Б.7, Б8.**

## Алгоритм редактирования элемента списка

При редактировании элемента из списка осуществляется поиск элемента по коду, введенному пользователем. В случае нахождения элемента, полям, выбранным пользователем, будут присвоены новые значения.

Данный алгоритм представлен в **Приложении Б, схема Б.6.**

## Алгоритм поиска элементов в списке

Поиск осуществляется пошаговым прохождение по элементам списка и проверкой на совпадение выбранного пользователем поля и введенного им значения. Информация о всех найденных элементах выводится на экран.

Данный алгоритм представлен в **Приложении Б, схема Б.9.**

## Алгоритм сортировки списка

Алгоритм сортировки списка основан на алгоритме сортировке простым выбором.

Алгоритм сортировки простым выбором:

* Находим номер минимального значения в текущем списке (Отношения между элементами списков определяются переданными функциями).
* Производим обмен этого значения со значением первой неотсортированной позиции.
* Теперь сортируем хвост списка, исключив из рассмотрения уже отсортированные элементы.

Данный алгоритм представлен в **Приложении Б, схема Б.10.**

## Алгоритм специальной функции

Алгоритм начинается с формирования списка из песен, подходящих под критерии, введенные пользователем. Затем происходит формирование playlist-ов. Они формируются следующим образом: проходим по списку и сравниваем, длину текущего playlist-а при вставке текущей песни и заданную пользователем, если длина текущего меньше, то добавляем песню в playlist и устанавливаем флаг для отбрасывания последней песни при переходе к следующей, и снова вызываем процедуру для формирования playlist-ов с текущей песни, если длины совпали то добавляем песню устанавливаем тот же флаг записываем данный playlist в массив playlist-ов, а если больше переходим с следующему шагу; после сравнения длин проверяем значение флага и в зависимости от него просто переходим к следующей песне или сначала извлекаем последнюю песню.

Данный алгоритм представлен в **Приложении Б, схема**

## Алгоритм сохранения в файл

Алгоритм основан на прохождении списков в цикле и записи значений их элементов в типизированный файл, выбор которого зависит от выбора пользователя.

Данный алгоритм представлен в **Приложении Б, схема**

## Определение основных подпрограмм и их описание

Таблица 2.1 – Основные подпрограммы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя подпрограммы | Назначение подпрограммы | Заголовок подпрограммы | Имя параметра | Назначение параметра |
| DeleteAllLists | Удаление всех списков в конце работы | Procedure DeleteAllLists(ArtistList: TAdrOfArtistList; AlbumList: TAdrOfAlbumList; SongList: TAdrOfSongList); | ArtistList | Список исполнителей |
| AlbumList | Список альбомов |
| SongList | Список песен |
| WatchArtistList | Просмотр списка исполнителей | Procedure WatchArtistList(ArtistList: TAdrOfArtistList); | ArtistList | Список исполнителей |
| WatchALbumList | Просмотр списка альбомов | Procedure WatchALbumList(AlbumList: TAdrOfALbumList); | AlbumList | Список альбомов |
| WatchSongList | Просмотр списка песен | Procedure WatchSongList(SongList: TAdrOfSongList); | SongList | Список песен |
| InsertArtist | Вставка нового исполнителя | Procedure InsertArtist(ArtistList: TAdrOfArtistList); | ArtistList | Список исполнителей |
| InsertAlbum | Вставка нового альбома | Procedure InsertAlbum(AlbumList: TAdrOfAlbumList; ArtistList: TAdrOfArtistList); | AlbumList | Список альбомов |
| ArtistList | Список исполнителей |

Продолжение таблицы 2.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя подпрограммы | Назначение подпрограммы | Заголовок подпрограммы | Имя параметра | Назначение параметра |
| InsertSong | Вставка новой песни | Procedure InsertSong(SongList: TAdrOfSongList; AlbumList: TAdrOfAlbumList;  ArtistList: TAdrOfArtistList); | SongList | Список песен |
| AlbumList | Список альбомов |
| ArtistList | Список исполнителей |
| DeleteArtist | Удаление исполнителя и всех его альбомов и песен | Procedure DeleteArtist(ArtistList: TAdrOfArtistList; AlbumList: TAdrOfAlbumList; SongList: TAdrOfSongList); | ArtistList | Список исполнителей |
| AlbumList | Список альбомов |
| SongList | Список песен |
| DeleteAlbum | Удаление альбома и всех песен в этом альбоме | Procedure DeleteAlbum(AlbumList: TAdrOfAlbumList; SongList: TAdrOfSongList;  CheckID: Integer); | AlbumList | Список альбомов |
| SongList | Список песен |
| CheckID | Код для удаления альбома (если не равен 0) |
| DeleteSong | Удаление песни | Procedure DeleteSong(SongList: TAdrOfSongList; CheckID: Integer); | SongList | Список песен |
| CheckID | Код для удаления песни (если не равен 0) |
| SearchArtist | Поиск в списке исполнителей | Procedure SearchArtist(ArtistList: TAdrOfArtistList); | ArtistList | Список исполнителей |
| SearchALbum | Поиск в списке альбомов | Procedure SearchALbum(AlbumList: TAdrOfALbumList); | AlbumList | Список альбомов |
| SearchSong | Поиск в списке песен | Procedure SearchSong(SongList: TAdrOfSongList); | SongList | Список песен |
| EditArtist | Редактирование информации о исполнителе по коду | Procedure EditArtist(ArtistList: TAdrOfArtistList); | ArtistList | Список исполнителей |
| EditAlbum | Редактирование информации о альбоме по коду | Procedure EditAlbum(AlbumList: TAdrOfALbumList); | AlbumList | Список исполнителей |
| EditSong | Редактирование информации о песне по коду | Procedure EditSong(SongList: TAdrOfSongList); | SongList | Список альбомов |
| ReadAllListsFromFiles | Чтение всех списков из файла | Procedure ReadAllListsFromFiles(ArtistList: TAdrOfArtistList; AlbumList: TAdrOfAlbumList; SongList: TAdrOfSongList; var ArtistFile: TArtistFile; var AlbumFile: TAlbumFile; var SongFile: TSongFile); | ArtistList | Список исполнителей |
| AlbumList | Список альбомов |
| SongList | Список песен |
| ArtistFile | Файл с исполнителями |
| AlbumFile | Файл с альбомами |
| SongFile | Файл с песнями |
| ReWriteAllListsInFiles | Запись всех списков в файл | Procedure ReWriteAllListsInFiles( ArtistList: TAdrOfArtistList; AlbumList: TAdrOfAlbumList; SongList: TAdrOfSongList; var ArtistFile: TArtistFile; var AlbumFile: TAlbumFile; var SongFile: TSongFile); | ArtistList | Список исполнителей |
| AlbumList | Список альбомов |
| SongList | Список песен |
| ArtistFile | Файл с исполнителями |
| AlbumFile | Файл с альбомами |
| SongFile | Файл с песнями |

Продолжение таблицы 2.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя подпрограммы | Назначение подпрограммы | Заголовок подпрограммы | Имя параметра | Назначение параметра |
| SelectionSort | Сортировка списка | Procedure SelectionSort(var L; const ArrIn: TArrayOfIndexes;  CompareTo: FCompareTo); | L | Список для сортировки |
| ArrIn | Порядок элементов в другом списке для фукнции CompareTo |
| CompareTo | Функция для сравнения элементов списков |
| MakePlaylist | Создание массива playlist-ов | Procedure MakePlaylist(ListOfAllSong: TAdrOfSongList; const PLength: Integer; var Arr: TArrOfArrOfIndexes); | ListOfAllSong | Список из песен, подходящих по критериям, введенным пользователем |
| PLength | Необходимая длина playlist-а |
| Arr | Массив для хранения всех playlist-ов |

## Выделение основных структур данных

Таблица 2.2 – Основные структуры данных

|  |  |  |
| --- | --- | --- |
| Имя идентификатора структуры | Назначение структуры | Тип структуры |
| TDataString | Строка для полей содержащих текстовые данные. | TDataString = String[20] |
| TArtist | Запись для хранения информации об исполнителе. | TArtist = record  ID: Integer;  Name: TDataString;  Country: TDataString;  Direction: TDirString;  end |
| TArtistList | Запись для храниния информации об элементе списка исполнителей. | TArtistList = record  next: TAdrOfArtistList;  Max\_Id: Integer;  Artist: TArtist;  end |
| TAlbum | Запись для хранения информации об альбоме. | TAlbum = record  ID: Integer;  ID\_Artist: Integer;  Name: TDataString;  Year: Integer;  end |
| TAlbumList | Запись для храниния информации об элементе списка альбомов. | TAlbumList = record  next: TAdrOfAlbumList;  Max\_Id: Integer;  Album: TAlbum;  end |
| TSong | Запись для хранения информации о песне. | TSong = record  ID: Integer;  Name: TDataString;  ID\_Album: Integer;  Length: Integer;  end; |
| TSongList | Запись для храниния информации об элементе списка песен. | TSongList = record  next: TAdrOfSongList;  Max\_Id: Integer;  Song: TSong;  end; |
| TArtistFile | Файл для хранения информации об исполнителе. | TArtistFile = File of TArtist |
| TAlbumFile | Файл для хранения информации об альбоме. | TAlbumFile = File of TAlbum |
| TSongFile | Файл для хранения информации о песне. | TSongFile = File of TSong |

## Тестирование и отладка программы

Таблица 2.3 – Прохождение тестов программы

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Вводимые данные | Ожидаемый результат | Полученный результат |
| 1 | Запуск программы. | Отображение главного меню, согласно пункту 1.3 | Тест пройден успешно. |
| 2 | Чтение из файла (отсутствие файлов). | Предложение создать новую сессию. | Тест пройден успешно. |
| 3 | Чтение из файла. | Данные загружены в списки. | Тест пройден успешно. |
| 4 | Отображение пустых списков. | Вывод на экран пустой таблицы. | Тест пройден успешно. |
| 5 | Отображение заполненных списков. | Вывод на экран заполненной таблицы. | Тест пройден успешно. |
| 6 | Добавление данных в пустой список исполнителей. | Исполнитель добавлен. | Тест пройден успешно. |
| 7 | Добавление данных в пустой список альбомов. | Альбом добавлен. | Тест пройден успешно. |
| 8 | Добавление данных в пустой список песен. | Песня добавлена. | Тест пройден успешно. |
| 9 | Добавление альбома, все списки пусты. | Предложение добавить исполнителя. Добавление исполнителя и альбома. | Тест пройден успешно. |
| 10 | Добавление песни, все списки пусты. | Предложение добавить альбом, затем предложение добавить исполнителя . Добавление исполнителя, альбома и песни. | Тест пройден успешно. |
| 11 | Удаление песни. | Песня удалена. | Тест пройден успешно. |
| 12 | Удаление альбома. | Альбом и все песни связанные с ним удалены. | Тест пройден успешно. |
| 13 | Удаление исполнителя. | Исполнитель и все его альбомы и песни удалены. | Тест пройден успешно. |
| 14 | Удаление всех исполнителей. | Все списки пусты. | Тест пройден успешно. |
| 15 | Редактирование исполнителя. | Имя, страна и направление песен изменены. | Тест пройден успешно. |
| 16 | Редактирование альбома. | Название и год выхода изменены. | Тест пройден успешно. |
| 17 | Редактирование песни. | Название и длительность изменены. | Тест пройден успешно. |
| 18 | Сортировка списков. | Все списки отсортированы в соответствии со специальной функцией. | Тест пройден успешно. |
| 19 | Создание playlist-ов (данные валидны). | Все возможные playlist-ы без повторений созданы. | Тест пройден успешно. |
| 20 | Создание playlist-ов (данные не валидны) | Ни одного playlist-а не было создано. | Тест пройден успешно. |
| 21 | Создание playlist-ов (пустые списки) | Ни одного playlist-а не было создано. | Тест пройден успешно. |

## Руководство по эксплуатации

Для установки данной программы ваш компьютер должен соответствовать требованиям:

1. Операционная система Windows 7 и выше;
2. 1 Гб ОЗУ;
3. 16 Мб свободного дискового пространства;
4. Наличие клавиатуры.

Чтобы установить программу, нужно скачать exe-файл «PlaylistCreator.exe» на жесткий диск компьютера. Для начала работы с программой нужно дважды щелкнуть по файлу «PlaylistCreator.exe», после чего приложение будет запущено.

Взаимодействие пользователя с данной программой происходит через клавиатуру. На экране будут отображены подсказки к возможному выполнению определенных действий.

После запуска программы пользователя будет встречать начальное меню, состоящее из 10 пунктов.

Программа для работы со списками, связанными с музыкой

Меню приложения:

1. Прочитать данные из файла.

2. Просмотреть списки.

3. Сортировать данные.

4. Найти данные в списках.

5. Добавить данные в списки.

6. Удалить данные из списков.

7. Редактировать списки.

8. Создать PlayList.

9. Выйти из программы без сохранения.

10. Выйти из программы с сохранением.

Для выбора определенного пункта меню необходимо ввести его номер. Все действия выбора подтверждаются нажатием кнопки Enter.

**Чтение из файла:**

Для работы с сохраненными результатами необходимо выбрать пункт меню “Прочитать данные из файла.”, после чего будет выведено подменю выбора файла для прочтения.

Меню чтения из файла.

1 .\files\Session

Выберите действие:

1. Выбрать существующую сессию.

2. Создать новую сессию.

1

Введите номер сессии:

**Просмотр списков:**

Для просмотра списков необходимо выбрать пункт меню “Просмотреть списки.”, после чего будет выведено подменю выбора списка для просмотра информации.

Меню подпункта просмотра списков:

1. Просмотреть список исполнителей.

2. Просмотреть список альбомов.

3. Просмотреть список песен.

0. Выйти из подпункта меню.

После выбора соответствующего списка будет отображена вся информация об элементах данного списка.

Пример список исполнителей:

|-----------------|-------------------|--------------------|-------------------------|

| Код исполнителя | Имя исполнителя | Страна исполнителя | Направление исполнителя |

|-----------------|-------------------|--------------------|-------------------------|

| 1 | ИСПОЛНИТЕЛЬ | СТРАНА | НАПРАВЛЕНИЕ |

|-----------------|-------------------|--------------------|-------------------------|

Пример список альбомов:

|-------------|-----------------|----------------------|------------|

| Код альбома | Код исполнителя | Название альбома | Год записи |

|-------------|-----------------|----------------------|------------|

| 1 | 1 | АЛЬБОМ | 2000 |

|-------------|-----------------|----------------------|------------|

Пример список песен:

|-----------|----------------------|-------------|--------------------|

| Код песни | Название песни | Код альбома | Длительность песни |

|-----------|----------------------|-------------|--------------------|

| 1 | ПЕСНЯ | 1 | 100 |

|-----------|----------------------|-------------|--------------------|

**Сортировка данных**

Для сортировки списков необходимо выбрать пункт «Сортировать данные.», после чего будет выведено сообщение:

Данные успешно отсортированы.

**Поиск данных**

Для поиска в списках необходимо выбрать пункт меню «Найти данные в списках.», после чего будет выведено подменю выбора списка для поиска.

Меню подпункта поиска:

1. Искать в списке исполнителей.

2. Искать в списке альбомов.

3. Искать в списке песен.

0. Выйти из подпункта меню.

После выбора списка будет отображено подменю выбора значения поля, по которому пользователь желает произвести поиск.

Пример список исполнителей:

Меню подпункта поиска в списке исполнителей:

1. Поиск по коду исполнителя.

2. Поиск по имени исполнителя.

3. Поиск по стране исполнителя.

4. Поиск по направлению исполнителя.

0. Выйти из подпункта меню.

Пример список альбомов:

Меню подпункта поиска в списке альбомов:

1. Поиск по коду альбома.

2. Поиск по коду исполнителя альбома.

3. Поиск по названию альбома.

0. Выйти из подпункта меню.

Пример список песен:

Меню подпункта поиска в списке песен:

1. Поиск по коду песни.

2. Поиск по коду альбома песни.

3. Поиск по названию песни.

4. Поиск по имени исполнителя.

0. Выйти из подпункта меню.

Затем необходимо ввести значение поля для поиска, после чего будут выведены результаты поиска.

**Добавление элементов в список**

При выборе пункта «Добавить данные в списки.» будет выведено подменю выбора списка для добавления.

Меню подпункта вставки элемента:

1. Вставить элемент в список исполнителей.

2. Вставить элемент в список альбомов.

3. Вставить элемент в список песен.

0. Выйти из подпункта меню.

Затем будет выведен список исполнителей или альбомов (для списков альбомов или песен, для исполнителей выведен не будет), и информация, которая будет вводится в качестве данных элементов списка будет постепенно выведена на экран.

Пример список песен:

|-------------|-----------------|----------------------|------------|

| Код альбома | Код исполнителя | Название альбома | Год записи |

|-------------|-----------------|----------------------|------------|

| 1 | 1 | АЛЬБОМ | 2000 |

|-------------|-----------------|----------------------|------------|

Введите код альбома: 1

Введите название песни: ПЕСНЯ2

Введите длину песни в секундах: 100

**Удаление элементов из списка.**

При выборе пункта «Удалить данные из списков.» будет выведено подменю выбора списка для удаления.

Меню подпункта удаления элемента:

1. Удалить элемент из списка исполнителей.

2. Удалить элемент из списка альбомов.

3. Удалить элемент из списка песен.

0. Выйти из подпункта меню.

После выбора списка, он будет выведен на экран и будет ожидаться ввод кода списка для удаления.

Пример список песен:

|-----------|----------------------|-------------|--------------------|

| Код песни | Название песни | Код альбома | Длительность песни |

|-----------|----------------------|-------------|--------------------|

| 2 | ПЕСНЯ2 | 1 | 100 |

| 1 | ПЕСНЯ | 1 | 100 |

|-----------|----------------------|-------------|--------------------|

Введите код песни для удаления: 2

**Редактирование элемента в списке**

При выборе пункта «Редактировать списки.» будет выведено подменю выбора списка для редактирования.

Меню подпункта редактирования элемента:

1. Редактировать элемент из списка исполнителей.

2. Редактировать элемент из списка альбомов.

3. Редактировать элемент из списка песен.

0. Выйти из подпункта меню.

После выбора списка он будет выведен на экран и будет ожидаться код элемента.

Пример список исполнителей:

|-----------------|-------------------|--------------------|-------------------------|

| Код исполнителя | Имя исполнителя | Страна исполнителя | Направление исполнителя |

|-----------------|-------------------|--------------------|-------------------------|

| 1 | ИСПОЛНИТЕЛЬ | СТРАНА | НАПРАВЛЕНИЕ |

|-----------------|-------------------|--------------------|-------------------------|

Введите код исполнителя для редактирования: 1

И после этого будет выведено подменю для выбора поля для редактирования.

Меню редактирования:

1. Редактировать имя исполнителя.

2. Редактировать страну исполнителя.

3. Редактировать направление исполнителя.

0. Выход из подменю редактирования исполнителя.

1

Введите новое имя исполнителя: ИСПОЛНИТЕЛЬ

**Создание playlist-а**

При выборе пункта «Создать Playlist.» будет ожидаться ввод данных, на основе которых будет создан playlist, а именно направление исполнителя, длительность в формате чч:мм:сс и год, с которого выбирать песни.

ВНИМАНИЕ, ЕСЛИ PLAYLIST УЖЕ БЫЛ СОЗДАН, ТО ЕГО ДАННЫЕ БУДУТ УТЕРЯНЫ.

Меню создания playlist-ов:

Введите направление исполнителя: НАПРАВЛЕНИЕ

Введите длину Playlist-а в формате: чч:мм:сс.

(Если часы и/или минуты равны 0-ю, 0-и необходимо записать, пример:

00:00:45 / 00:45:00).

Длина: 00:01:40

Введите год, с которого выбирать песни: 1

После нажатия Enter будут сохранены в файл «PlayListFile.txt» и выведены на экран все составленные playlist-ы и их количество.

Количество playlist-ов: 1

|-----------|----------------------|-------------|--------------------|

| Код песни | Название песни | Код альбома | Длительность песни |

|-----------|----------------------|-------------|--------------------|

| 1 | ПЕСНЯ | 1 | 100 |

|-----------|----------------------|-------------|--------------------|

**Сохранение**

При работе с приложением возможно сохранение в файл для его последующего чтения.

**Важно!**

1. В течении одной сессии файл может быть прочитан один раз. В случае желания пользователя прочитать другой файл, необходимо завершить программу и прочитать нужный файл.
2. Все данные сохраненной сессии будут сохранены в папке files, располагающейся в одной директории с данной программой.
3. При активации спец. функции результаты будут размещены в папке с названием сессии.

**Обработка ошибок:**

В случае введения некорректных данных будет отображено соответствующее сообщение об ошибке.

**Завершение работы:**

Для выхода из программы пользователю необходимо ввести команду «9» (Выход из программы без сохранения изменений) или «10» (Выход с сохранением изменений) и нажать клавишу Enter.

# ПРИЛОЖЕНИЕ А

**ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

|  |  |
| --- | --- |
| **УТВЕРЖДАЮ**  Руководитель  Фадеева Елена Евгеньевна  «11» марта 2024 г. |  |
| Программное средство для работы с playlist-ами  “Playlist Creator”  наименование вида ИС  Playlist Creator  Сокращенное наименование ИС  Техническое задание  Действует с 10.03.2024 г. | |
| **СОГЛАСОВАНО**  Руководитель  Фадеева Елена Евгеньевна  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г. | **РАЗРАБОТЧИК**  Студент  Головко Роман Сергеевич  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024г. |
| Минск 2024 | |

**СОДЕРЖАНИЕ**

[1. ОБЩИЕ СВЕДЕНИЯ 20](#_Toc168535048)

[2. НАЗНАЧЕНИЕ И ЦЕЛИ СОЗДАНИЯ СИСТЕМЫ 21](#_Toc168535049)

[2.1. Назначение Системы 21](#_Toc168535050)

[2.2. Цели создания Системы 21](#_Toc168535051)

[3. ХАРАКТЕРИСТИКА ОБЪЕКТА АВТОМАТИЗАЦИИ 21](#_Toc168535052)

[3.1. Краткие сведения об объектах автоматизации 21](#_Toc168535053)

[4. TРЕБОВАНИЯ К СИСТЕМЕ 22](#_Toc168535054)

[4.1. Требования к системе в целом 22](#_Toc168535055)

[4.1.1. Требования к структуре и функционированию 22](#_Toc168535056)

[4.1.1.1. Перечень подсистем, их назначение и основные характеристики 22](#_Toc168535057)

[4.1.1.2. Требования к организации обмена информацией между компонентами Системы 22](#_Toc168535058)

[4.2. Функциональные требования 22](#_Toc168535059)

[4.3. Обработка ошибок 23](#_Toc168535060)

[4.3.1. Ошибки загрузки данных из внешних источников 23](#_Toc168535061)

[4.3.2. Внутренние ошибки 24](#_Toc168535062)

[4.4. Интерфейс 24](#_Toc168535063)

[4.4.1. Основные требования 24](#_Toc168535064)

# ОБЩИЕ СВЕДЕНИЯ

Полное наименование Системы и её условное обозначение

Полное наименование системы: Программное средство для работы с playlist-ами“Playlist Creator”

Краткое наименование системы: “Playlist Creator”

**1.1. Наименование и реквизиты Заказчика**

Заказчиком Системы является кафедра ПОИТ (ассистент Фадеева Елена Евгеньевна)

Контактный телефон: +375 44 552 37 38

**1.2. Наименование и реквизиты Исполнителя**

Исполнителем является студент группы 351001 Головко Роман Сергеевич

Адрес электронной почты: rgolovkl6@gmail.com

Телефон: +375 44 751 95 99

**1.3. Основание для разработки**

- Список индивидуальных заданий УП

- Список вариантов к списку индивидуальных заданий

**1.4. Плановые сроки начала и окончания работ**

Начало разработки: 10 марта 2024 года;

Ожидаемое завершение разработки: 31 мая 2024 года

# НАЗНАЧЕНИЕ И ЦЕЛИ СОЗДАНИЯ СИСТЕМЫ

## 2.1. Назначение Системы

Система “Playlist Creator” предназначена для удобной работы с музыкой*.*

## Цели создания Системы

Система " Construction Manager " (СМ) создана с цельюавтоматизациисоздания playlist-ов по пользовательским предпочтениям.

# ХАРАКТЕРИСТИКА ОБЪЕКТА АВТОМАТИЗАЦИИ

## Краткие сведения об объектах автоматизации

В соответствии с пунктом 2.1 можно выделить следующие объекты автоматизации:

1. Списки исполнителей

Каждый элемент этого списка содержит следующую информацию:

* Код исполнителя;
* Имя исполнителя;
* Страну исполнителя;
* Направление исполнителя;

1. Списки альбомов

Каждый элемент этого списка содержит следующую информацию:

* Код альбома;
* Код исполнителя;
* Название альбома;
* Год выпуска альбома;

1. Списки песен

Каждый элемент этого списка содержит следующую информацию:

* Код песни;
* Код альбома;
* Название песни;
* Длительность;

# TРЕБОВАНИЯ К СИСТЕМЕ

## Требования к системе в целом

Простота в использовании, логичный, понятный интерфейс, работа без аварийных завершений.

## Требования к структуре и функционированию

## Перечень подсистем, их назначение и основные характеристики

Программа должна содержать пользовательское меню, состоящее из следующих пунктов:

1.Чтение данных из файла

2.Просмотр всего списка (если несколько списков – выпадает подменю с именами списков для просмотра)

3.Сортировка данных в соответствии с заданием

4.Поиск данных с использованием фильтров

5.Добавление данных в список (если несколько списков – выпадает подменю с именами списков для добавления)

6.Удаление данных из списка (если несколько списков – выпадает подменю с именами списков для удаления)

7.Редактирование данных (если несколько списков – выпадает подменю с именами списков для редактирования)

8.Специальные функции задания

9.Выход из программы без сохранения изменений

10.Выход с сохранением изменений

## Требования к организации обмена информацией между компонентами Системы

**1. Стандартизация данных:** Все компоненты системы должны использовать единые стандарты для представления данных. Это включает в себя стандартизированные форматы данных, соглашения об именовании полей и другие соглашения, которые облегчают интеграцию

**2**. **Интеграция с внешними источниками**: Система должна быть способной интегрировать данные из внешних источников, таких как сторонние файлы, базы данных.

## Функциональные требования

1. Подсистемы организации работы меню и подменю:

Задачи:

* Отрисовка соответствующего меню/подменю;
* Получение данных от пользователя;
* Вызов подсистемы в соответствии с введёнными данными с передачей ей необходимых для работы данных;

1. Подсистемы работы с типизированными и текстовыми файлами:

Задачи:

* Проверка наличия соответствующих файлов;
* Чтение информации из типизированного/текстового файла;
* Запись информации в типизированный/текстовый файл;

1. Подсистемы просмотра выбранного списка:

Задачи:

* Обработка полученных при вызове данных;
* Вывод в консоль выбранного списка;

1. Подсистемы добавления/удаления/редактирования записей в выбранном списке:

Задачи:

* Обработка полученных при вызове данных;
* Изменение выбранного списка (добавление/удаление/редактирование) в соотв. с обработанными данными;

1. Подсистемы поиска записей по выбранному полю:

Задачи:

* Обработка полученных при вызове данных;
* Осуществление поиска в списке в соотв. с обработанными данными;

1. Подсистемы сортировки списков:

Задачи:

* Обработка полученных при вызове данных;
* Сортировка списков;

1. Создание playlist-ов (критерии расписаны в 3.1):

Задачи:

* Обработка полученных при вызове данных;
* Осуществление корректной работы алгоритма по созданию playlist-ов.

## Обработка ошибок

## Ошибки загрузки данных из внешних источников

Ошибки загрузки данных из внешних источников могут возникнуть в следующих случаях:

1. Повреждение рабочих файлов в результате некорректной работы Системы при прошлом запуске;
2. Повреждение рабочих файлов в результате непреднамеренных действий пользователя;
3. Изменение формата/имени рабочих файлов.

При возникновении ошибок данного типа необходимо уведомить пользователя и с его подтверждения создать новые рабочие файлы.

## Внутренние ошибки

Внутренние ошибки могут возникнуть в следующих случаях:

1. Ввод пользователем непредвиденных/некорректных данных. Для предотвращения подобного рода ошибок необходимо проводить проверку всех введённых пользователем данных. Также необходимо оповестить пользователя и позволить повторный ввод данных;
2. Некорректная работа с динамическими структурами данных. Результат подобного рода ошибок непредсказуем.

## Интерфейс

**Пользовательский интерфейс:** должен быть интуитивно понятным и удобным. Разрабатывается простой, но информативный интерфейс, который обеспечивает легкий доступ к основным функциям системы.

## Основные требования

Интерфейс должен быть реализован в консольном режиме.

Пользовательское меню должно содержать следующие пункты:

1. Прочитать данные из файла.
2. Просмотреть списки.
3. Сортировать данные.
4. Найти данные в списках.
5. Добавить данные в списки.
6. Удалить данные из списков.
7. Редактировать списки.
8. Создать Playlist.
9. Выйти из программы без сохранения.
10. Выйти из программы с сохранением.

Для пунктов №2, 3, 4, 5, 6, 7 необходимо реализовать выпадающие подменю для ввода дополнительной информации (фильтры поиска, код удаляемой записи, интересующий список и т.д.).

# ПРИЛОЖЕНИЕ Б

Схема Б.1 - Схема работы программы



Схема Б.2 – Схема чтения списка исполнителей из файла



Схема Б.3 – Отображение списка



Схема Б.4 – Вставка исполнителя в список



Схема Б.5 – Вставка альбома в список



Схема Б.6 – Редактирование элемента в списке исполнителей



Схема Б.7 – Удаление элемента из списка песен



Схема Б.8 – Удаление элемента из списка альбомов



Схема Б.9 – Поиск в списке исполнителей



Схема Б.10 – Сортировка списка



Схема Б.11 – Создание playlist-а



Схема Б.12 – Запись исполнителя в файл



# ПРИЛОЖЕНИЕ В

Программный код

program PlaylistCreator;

uses

AllTypesInProject in 'AllTypesInProject.pas',

WorkWithFiles in 'WorkWithFiles.pas',

WorkWithLists in 'WorkWithLists.pas',

MenuItems in 'MenuItems.pas',

Playlist in 'Playlist.pas';

var

ArtistList: TAdrOfArtistList;

AlbumList: TAdrOfAlbumList;

SongList: TAdrOfSongList;

ArtistFile: TArtistFile;

AlbumFile: TAlbumFile;

SongFile: TSongFile;

Menu: Integer;

StateOfFiles: TStateOfFile;

CurrSession: String = '';

begin

StateOfFiles := NoFileInformation;

New(ArtistList);

ArtistList^.Max\_Id := 0;

ArtistList^.next := nil;

New(AlbumList);

AlbumList^.Max\_Id := 0;

AlbumList^.next := nil;

New(SongList);

SongList^.Max\_Id := 0;

SongList^.next := nil;

WriteLn('Программа для работы со списками, связанными с музыкой');

Repeat

WriteLn('Меню приложения:');

WriteLn(' 1. Прочитать данные из файла.');

WriteLn(' 2. Просмотреть списки.');

WriteLn(' 3. Сортировать данные.');

WriteLn(' 4. Найти данные в списках.');

WriteLn(' 5. Добавить данные в списки.');

WriteLn(' 6. Удалить данные из списков.');

WriteLn(' 7. Редактировать списки.');

WriteLn(' 8. Создать Playlist.');

WriteLn(' 9. Выйти из программы без сохранения.');

WriteLn('10. Выйти из программы с сохранением.');

ReadNum(Menu);

case Menu of

1:

begin

MenuItem1\_ReadLists(CurrSession, ArtistList, AlbumList, SongList,

ArtistFile, AlbumFile, SongFile, StateOfFiles);

end;

2:

begin

MenuItem2\_WatchLists(ArtistList, AlbumList, SongList);

end;

3:

begin

MenuItem3\_Sort(ArtistList, AlbumList, SongList);

end;

4:

begin

MenuItem4\_Search(ArtistList, AlbumList, SongList);

end;

5:

begin

StateOfFiles := ListChanged;

MenuItem5\_Insert(ArtistList, AlbumList, SongList);

end;

6:

begin

MenuItem6\_Delete(ArtistList, AlbumList, SongList);

end;

7:

begin

MenuItem7\_Edit(ArtistList, AlbumList, SongList);

end;

8:

begin

MenuItem8\_Playlist(CurrSession, ArtistList, AlbumList, SongList);

end;

10:

begin

ReWriteAllListsInFiles(CurrSession, ArtistList, AlbumList, SongList,

ArtistFile, AlbumFile, SongFile);

end;

end;

Until (Menu = 10) or (Menu = 9);

DeleteAllLists(ArtistList, AlbumList, SongList);

end.

**unit AllTypesInProject:**

unit AllTypesInProject;

interface

Type

TStateOfFile = (NoFileInformation, FileExist, FileNotExist, FileAlreadyRead,

ListChanged);

TDir = record

ID: Integer;

Dir: String;

end;

TArrOfDir = array of TDir;

TAdrOfArtistList = ^TArtistList;

TAdrOfAlbumList = ^TAlbumList;

TAdrOfSongList = ^TSongList;

TArrayOfIndexes = array of Integer;

TDataString = String[20];

TDirString = String[15];

TArtist = record

ID: Integer;

Name: TDataString;

Country: TDataString;

Direction: TDirString;

end;

TArtistFile = File of TArtist;

TAlbum = record

ID: Integer;

ID\_Artist: Integer;

Name: TDataString;

Year: Integer;

end;

TAlbumFile = File of TAlbum;

TSong = record

ID: Integer;

Name: TDataString;

ID\_Album: Integer;

Length: Integer;

end;

TSongFile = File of TSong;

TArtistList = record

next: TAdrOfArtistList;

Max\_Id: Integer;

Artist: TArtist;

end;

TAlbumList = record

next: TAdrOfAlbumList;

Max\_Id: Integer;

Album: TAlbum;

end;

TSongList = record

next: TAdrOfSongList;

Max\_Id: Integer;

Song: TSong;

end;

TAdrOfList = ^TList;

TTypeList = (Artist, Album, Song);

TList = record

next: TAdrOfList;

Max\_Id: Integer;

case TTypeList of

Artist:

(Artist: TArtist);

Album:

(Album: TAlbum);

Song:

(Song: TSong);

end;

PInput\_Search = Procedure(var ID: Integer; var S: TDataString);

FCondEq\_Search = Function(var Element; var ID: Integer;

var S: TDataString): Boolean;

FCompareTo = Function(Self, o: TAdrOfList;

const ArrIn: TArrayOfIndexes): Boolean;

TArrOfArrOfIndexes = array of TArrayOfIndexes;

TArrOfPlaylists = array of TAdrOfSongList;

Procedure Add10(var Arr: TArrayOfIndexes);

Procedure ReadNum(var n: Integer);

implementation

Procedure Add10(var Arr: TArrayOfIndexes);

var

OldLast, I: Integer;

begin

OldLast := Length(Arr);

SetLength(Arr, Length(Arr) + 10);

for I := OldLast to High(Arr) do

Arr[I] := 0;

end;

Procedure ReadNum(var n: Integer);

var

S: String;

Err: Integer;

begin

Err := 0;

repeat

readln(S);

Val(S, n, Err);

if (Err <> 0) or (n < 0) then

Write('Некорректный ввод. Введите снова: ');

until (Err = 0) and (n >= 0);

end;

end.

**unit WorkWithFiles:**

unit WorkWithFiles;

interface

uses AllTypesInProject;

// ReWrite All Lists in files

Procedure ReWriteAllListsInFiles(var CurrSession: String;

ArtistList: TAdrOfArtistList; AlbumList: TAdrOfAlbumList;

SongList: TAdrOfSongList; var ArtistFile: TArtistFile;

var AlbumFile: TAlbumFile; var SongFile: TSongFile);

Procedure MenuReadFiles(var State: TStateOfFile; var CurrSession: String;

ArtistList: TAdrOfArtistList; AlbumList: TAdrOfAlbumList;

SongList: TAdrOfSongList; var ArtistFile: TArtistFile;

var AlbumFile: TAlbumFile; var SongFile: TSongFile);

implementation

uses SysUtils, IOUtils;

{ \\\\\\\\\\ Work with ArtistFile ////////// }

// Read ArtistList from file

Procedure ReadArtistListFromFile(var CurrSession: String;

ArtistList: TAdrOfArtistList; var ArtistFile: TArtistFile);

begin

Assign(ArtistFile, CurrSession + '\ArtistFile');

Reset(ArtistFile);

if Not(Eof(ArtistFile)) then

begin

Read(ArtistFile, ArtistList^.Artist);

ArtistList^.Max\_Id := ArtistList^.Artist.ID;

while Not(Eof(ArtistFile)) do

begin

New(ArtistList^.next);

ArtistList := ArtistList^.next;

Read(ArtistFile, ArtistList^.Artist);

ArtistList^.next := nil;

end;

end;

CloseFile(ArtistFile);

end;

// ReWrite ArtistList in file

Procedure ReWriteArtistListInFile(var CurrSession: String;

ArtistList: TAdrOfArtistList; var ArtistFile: TArtistFile);

begin

Assign(ArtistFile, CurrSession + '\ArtistFile');

ReWrite(ArtistFile);

ArtistList^.Artist.ID := ArtistList^.Max\_Id;

while ArtistList <> nil do

begin

Write(ArtistFile, ArtistList^.Artist);

ArtistList := ArtistList^.next;

end;

CloseFile(ArtistFile);

end;

{ \\\\\\\\\\ Work with AlbumFile ////////// }

// Read AlbumList from file

Procedure ReadAlbumListFromFile(var CurrSession: String;

AlbumList: TAdrOfAlbumList; var AlbumFile: TAlbumFile);

begin

Assign(AlbumFile, CurrSession + '\AlbumFile');

Reset(AlbumFile);

if Not(Eof(AlbumFile)) then

begin

Read(AlbumFile, AlbumList^.Album);

AlbumList^.Max\_Id := AlbumList^.Album.ID;

while Not(Eof(AlbumFile)) do

begin

New(AlbumList^.next);

AlbumList := AlbumList^.next;

Read(AlbumFile, AlbumList^.Album);

AlbumList^.next := nil;

end;

end;

CloseFile(AlbumFile);

end;

// ReWrite AlbumList in file

Procedure ReWriteAlbumListInFile(var CurrSession: String;

AlbumList: TAdrOfAlbumList; var AlbumFile: TAlbumFile);

begin

Assign(AlbumFile, CurrSession + '\AlbumFile');

ReWrite(AlbumFile);

AlbumList^.Album.ID := AlbumList^.Max\_Id;

while AlbumList <> nil do

begin

Write(AlbumFile, AlbumList^.Album);

AlbumList := AlbumList^.next;

end;

CloseFile(AlbumFile);

end;

{ \\\\\\\\\\ Work with SongFile ////////// }

// Read SongList from file

Procedure ReadSongListFromFile(var CurrSession: String;

SongList: TAdrOfSongList; var SongFile: TSongFile);

begin

Assign(SongFile, CurrSession + '\SongFile');

Reset(SongFile);

if Not(Eof(SongFile)) then

begin

Read(SongFile, SongList^.Song);

SongList^.Max\_Id := SongList^.Song.ID;

while Not(Eof(SongFile)) do

begin

New(SongList^.next);

SongList := SongList^.next;

Read(SongFile, SongList^.Song);

SongList^.next := nil;

end;

end;

CloseFile(SongFile);

end;

// ReWrite SongList in file

Procedure ReWriteSongListInFile(var CurrSession: String;

SongList: TAdrOfSongList; var SongFile: TSongFile);

begin

Assign(SongFile, CurrSession + '\SongFile');

ReWrite(SongFile);

SongList^.Song.ID := SongList^.Max\_Id;

while SongList <> nil do

begin

Write(SongFile, SongList^.Song);

SongList := SongList^.next;

end;

CloseFile(SongFile);

end;

Procedure CheckAllFiles(const CurrSession: String; var State: TStateOfFile);

begin

if FileExists(CurrSession + '\ArtistFile') and

FileExists(CurrSession + '\AlbumFile') and

FileExists(CurrSession + '\SongFile') then

State := FileExist

else

State := FileNotExist;

end;

// Read All Lists from files

Procedure ReadAllListsFromFiles(var State: TStateOfFile;

var CurrSession: String; ArtistList: TAdrOfArtistList;

AlbumList: TAdrOfAlbumList; SongList: TAdrOfSongList;

var ArtistFile: TArtistFile; var AlbumFile: TAlbumFile;

var SongFile: TSongFile);

begin

CheckAllFiles(CurrSession, State);

if State = FileExist then

begin

ReadArtistListFromFile(CurrSession, ArtistList, ArtistFile);

ReadAlbumListFromFile(CurrSession, AlbumList, AlbumFile);

ReadSongListFromFile(CurrSession, SongList, SongFile);

end;

end;

// ReWrite All Lists in files

Procedure ReWriteAllListsInFiles(var CurrSession: String;

ArtistList: TAdrOfArtistList; AlbumList: TAdrOfAlbumList;

SongList: TAdrOfSongList; var ArtistFile: TArtistFile;

var AlbumFile: TAlbumFile; var SongFile: TSongFile);

begin

ReWriteArtistListInFile(CurrSession, ArtistList, ArtistFile);

ReWriteAlbumListInFile(CurrSession, AlbumList, AlbumFile);

ReWriteSongListInFile(CurrSession, SongList, SongFile);

end;

//

Procedure GetAllDirectories(var ArrOfDirectories: TArrOfDir);

var

i: Integer;

S: String;

begin

i := 0;

for S in TDirectory.GetDirectories('.\files') do

begin

Inc(i);

Setlength(ArrOfDirectories, i);

ArrOfDirectories[i - 1].ID := i;

ArrOfDirectories[i - 1].Dir := S;

end;

end;

Function SearchInArr(const Arr: TArrOfDir; const item: Integer): TDir;

var

Flag: Boolean;

i: Integer;

begin

i := Low(Arr);

Flag := true;

result.ID := -1;

while Flag and (i <= High(Arr)) do

begin

if item = Arr[i].ID then

begin

result := Arr[i];

Flag := false;

end;

Inc(i);

end;

end;

Procedure CreateNewSession(var CurrSession: String);

var

NameSession: String;

Flag: Boolean;

i: Integer;

begin

Flag := true;

Writeln('Введите имя новой сессии.');

Writeln('Правила для ввода имени.');

Writeln('Длина должна быть от 1 до 50 символов.');

Writeln('В состав названия могут входить буквы английского');

Writeln('алфавита, числа и пробелы.');

Write('Имя сессии: ');

repeat

if Not(Flag) then

Write('Неверный формат ввода. Введите снова: ');

Readln(NameSession);

Flag := true;

if (Length(NameSession) = 0) or DirectoryExists('.\files\' + NameSession)

then

begin

Flag := false;

end

else

for i := Low(NameSession) to High(NameSession) do

if Not(CharInSet(NameSession[i], [' ', 'A' .. 'Z', 'a' .. 'z',

'0' .. '9'])) then

begin

Flag := false;

break;

end;

until Flag;

CurrSession := '.\files\' + NameSession;

TDirectory.CreateDirectory(CurrSession);

end;

Procedure ChooseSession(const ArrOfDirectories: TArrOfDir;

var CurrSession: String);

var

Flag: Boolean;

IDSession: Integer;

Dir: TDir;

begin

Flag := true;

Write('Введите номер сессии: ');

repeat

if Not(Flag) then

Write('Неверный формат ввода. Введите снова: ');

ReadNum(IDSession);

Flag := true;

Dir := SearchInArr(ArrOfDirectories, IDSession);

if Dir.ID = -1 then

Flag := false;

until Flag;

CurrSession := Dir.Dir;

end;

//

Procedure MenuReadFiles(var State: TStateOfFile; var CurrSession: String;

ArtistList: TAdrOfArtistList; AlbumList: TAdrOfAlbumList;

SongList: TAdrOfSongList; var ArtistFile: TArtistFile;

var AlbumFile: TAlbumFile; var SongFile: TSongFile);

var

i, Menu: Integer;

ArrOfDirectories: TArrOfDir;

begin

if Not(TDirectory.Exists('.\files')) then

TDirectory.CreateDirectory('.\files');

GetAllDirectories(ArrOfDirectories);

Writeln('Меню чтения из файла.');

for i := Low(ArrOfDirectories) to High(ArrOfDirectories) do

Writeln(ArrOfDirectories[i].ID, ' ', ArrOfDirectories[i].Dir);

if (Length(ArrOfDirectories) = 0) or (State = ListChanged) then

begin

CreateNewSession(CurrSession);

end

else

begin

Writeln('Выберите действие:');

Writeln('1. Выбрать существующую сессию.');

Writeln('2. Создать новую сессию.');

ReadNum(Menu);

case Menu of

1:

begin

ChooseSession(ArrOfDirectories, CurrSession);

ReadAllListsFromFiles(State, CurrSession, ArtistList, AlbumList,

SongList, ArtistFile, AlbumFile, SongFile);

end;

2:

begin

CreateNewSession(CurrSession);

end;

end;

end;

end;

end.

**unit WorkWithLists:**

unit WorkWithLists;

interface

uses

AllTypesInProject;

Procedure WatchArtistList(ArtistList: TAdrOfArtistList);

Procedure InsertArtist(ArtistList: TAdrOfArtistList);

Procedure DeleteArtist(ArtistList: TAdrOfArtistList; AlbumList: TAdrOfAlbumList;

SongList: TAdrOfSongList);

Procedure EditArtist(ArtistList: TAdrOfArtistList);

Procedure WatchALbumList(AlbumList: TAdrOfAlbumList);

Procedure InsertAlbum(AlbumList: TAdrOfAlbumList; ArtistList: TAdrOfArtistList);

Procedure DeleteAlbum(AlbumList: TAdrOfAlbumList; SongList: TAdrOfSongList;

CheckID: Integer);

Procedure EditAlbum(AlbumList: TAdrOfAlbumList);

Procedure WatchSongList(SongList: TAdrOfSongList);

Procedure InsertSong(SongList: TAdrOfSongList; AlbumList: TAdrOfAlbumList;

ArtistList: TAdrOfArtistList);

Procedure DeleteSong(SongList: TAdrOfSongList; CheckID: Integer);

Procedure EditSong(SongList: TAdrOfSongList);

Procedure MenuSearchArtist(ArtistList: TAdrOfArtistList);

Procedure MenuSearchAlbum(AlbumList: TAdrOfAlbumList);

Procedure MenuSearchSong(ArtistList: TAdrOfArtistList;

AlbumList: TAdrOfAlbumList; SongList: TAdrOfSongList);

implementation

{ \\\\\\\\\\ Work with ArtistList ////////// }

// Просмотреть список исполнителей.

Procedure WatchArtistList(ArtistList: TAdrOfArtistList);

begin

Writeln('|-----------------|-------------------|--------------------|-------------------------|');

Writeln('| Код исполнителя | Имя исполнителя | Страна исполнителя | Направление исполнителя |');

Writeln('|-----------------|-------------------|--------------------|-------------------------|');

while ArtistList^.next <> nil do

begin

ArtistList := ArtistList^.next;

Writeln('|', ArtistList^.Artist.ID:16, ' |', ArtistList^.Artist.Name:18,

' |', ArtistList^.Artist.Country:19, ' |', ArtistList^.Artist.Direction

:24, ' |');

end;

Writeln('|-----------------|-------------------|--------------------|-------------------------|');

Writeln;

end;

// Проверка на существование исполнителя.

Function IsArtistAlreadyExist(ArtistList: TAdrOfArtistList;

TmpArtist: TArtist): Boolean;

begin

Result := False;

while ArtistList^.next <> nil do

begin

ArtistList := ArtistList^.next;

if (ArtistList^.Artist.Name = TmpArtist.Name) and

(ArtistList^.Artist.Country = TmpArtist.Country) and

(ArtistList^.Artist.Direction = TmpArtist.Direction) then

Result := True;

end;

end;

// Вставить исполнителя в список.

Procedure InsertArtist(ArtistList: TAdrOfArtistList);

var

MaxId: Integer;

Tmp: TAdrOfArtistList;

TmpArtist: TArtist;

begin

Write('Введите имя исполнителя: ');

readln(TmpArtist.Name);

Write('Введите страну исполнителя: ');

readln(TmpArtist.Country);

Write('Введите направление песен исполнителя: ');

readln(TmpArtist.Direction);

if not(IsArtistAlreadyExist(ArtistList, TmpArtist)) then

begin

Inc(ArtistList^.Max\_Id);

MaxId := ArtistList^.Max\_Id;

Tmp := ArtistList^.next;

New(ArtistList^.next);

ArtistList := ArtistList^.next;

TmpArtist.ID := MaxId;

ArtistList^.Artist := TmpArtist;

ArtistList^.next := Tmp;

end

else

Writeln('Такой исполнитель уже существует.');

Writeln;

end;

// Удалить исполнителя из списка.

Procedure DeleteArtist(ArtistList: TAdrOfArtistList; AlbumList: TAdrOfAlbumList;

SongList: TAdrOfSongList);

var

IDForDelete: Integer;

Tmp: TAdrOfArtistList;

TmpAlbumList: TAdrOfAlbumList;

Flag: Boolean;

begin

WatchArtistList(ArtistList);

Write('Введите код исполнителя для удаления: ');

ReadNum(IDForDelete);

Flag := False;

While Not(Flag) and (ArtistList^.next <> nil) do

begin

if ArtistList^.next^.Artist.ID = IDForDelete then

begin

Flag := True;

Tmp := ArtistList^.next;

ArtistList^.next := ArtistList^.next^.next;

Dispose(Tmp);

end;

ArtistList := ArtistList^.next;

end;

if Not(Flag) then

Writeln('Исполнителя с таким кодом нет в списке.')

else

begin

New(TmpAlbumList);

TmpAlbumList^.next := AlbumList;

while AlbumList <> nil do

begin

if (AlbumList^.next <> nil) and

(AlbumList^.next^.Album.ID\_Artist = IDForDelete) then

begin

DeleteAlbum(TmpAlbumList^.next, SongList, AlbumList^.next^.Album.ID);

AlbumList := TmpAlbumList;

end;

AlbumList := AlbumList^.next;

end;

end;

Writeln;

end;

Procedure InputArtistID(var ID: Integer; var S: TDataString);

begin

Write('Введите код исполнителя: ');

ReadNum(ID);

end;

Function ConditionArtistID(var Element; var ID: Integer;

var S: TDataString): Boolean;

begin

Result := TArtist(Element).ID = ID;

end;

Procedure InputArtistName(var ID: Integer; var S: TDataString);

begin

Write('Введите имя исполнителя: ');

readln(S);

end;

Function ConditionArtistName(var Element; var ID: Integer;

var S: TDataString): Boolean;

begin

Result := TArtist(Element).Name = S;

end;

Procedure InputArtistCountry(var ID: Integer; var S: TDataString);

begin

Write('Введите страну исполнителя: ');

readln(S);

end;

Function ConditionArtistCountry(var Element; var ID: Integer;

var S: TDataString): Boolean;

begin

Result := TArtist(Element).Country = S;

end;

Procedure InputArtistDirection(var ID: Integer; var S: TDataString);

begin

Write('Введите направление исполнителя: ');

readln(S);

end;

Function ConditionArtistDirection(var Element; var ID: Integer;

var S: TDataString): Boolean;

begin

Result := TArtist(Element).Direction = S;

end;

// Найти исполнителя в списке.

Procedure SearchArtist(ArtistList: TAdrOfArtistList; Input: PInput\_Search;

Cond: FCondEq\_Search);

var

SearchID: Integer;

SearchString: TDataString;

begin

Input(SearchID, SearchString);

Writeln('|-----------------|-------------------|--------------------|-------------------------|');

Writeln('| Код исполнителя | Имя исполнителя | Страна исполнителя | Направление исполнителя |');

Writeln('|-----------------|-------------------|--------------------|-------------------------|');

while (ArtistList^.next <> nil) do

begin

ArtistList := ArtistList^.next;

if Cond(ArtistList^.Artist, SearchID, SearchString) then

begin

Writeln('|', ArtistList^.Artist.ID:16, ' |', ArtistList^.Artist.Name:18,

' |', ArtistList^.Artist.Country:19, ' |', ArtistList^.Artist.Direction

:24, ' |');

end;

end;

Writeln('|-----------------|-------------------|--------------------|-------------------------|');

Writeln;

end;

// Найти всех исполнителей по имени в списке.

Procedure SearchArtistByName(ArtistList: TAdrOfArtistList;

var ArtistIndexes: TArrayOfIndexes);

var

SearchID, Index: Integer;

SearchString: TDataString;

begin

Index := 0;

InputArtistName(SearchID, SearchString);

while (ArtistList^.next <> nil) do

begin

ArtistList := ArtistList^.next;

if ConditionArtistName(ArtistList^.Artist, SearchID, SearchString) then

begin

if High(ArtistIndexes) < Index then

Add10(ArtistIndexes);

ArtistIndexes[Index] := ArtistList^.Artist.ID;

Inc(Index);

end;

end;

end;

// Меню поиска в списке исполнителей.

Procedure MenuSearchArtist(ArtistList: TAdrOfArtistList);

var

Menu: Integer;

begin

repeat

Writeln('Меню подпункта поиска в списке исполнителей:');

Writeln('1. Поиск по коду исполнителя.');

Writeln('2. Поиск по имени исполнителя.');

Writeln('3. Поиск по стране исполнителя.');

Writeln('4. Поиск по направлению исполнителя.');

Writeln('0. Выйти из подпункта меню.');

ReadNum(Menu);

case Menu of

1:

SearchArtist(ArtistList, InputArtistID, ConditionArtistID);

2:

SearchArtist(ArtistList, InputArtistName, ConditionArtistName);

3:

SearchArtist(ArtistList, InputArtistCountry, ConditionArtistCountry);

4:

SearchArtist(ArtistList, InputArtistDirection,

ConditionArtistDirection);

end;

until Menu = 0;

end;

// Выбор поля для редактирования исполнителя

Procedure EditArtistMenu(ArtistList: TAdrOfArtistList);

var

Menu: Integer;

begin

repeat

Writeln('Меню редактирования:');

Writeln('1. Редактировать имя исполнителя.');

Writeln('2. Редактировать страну исполнителя.');

Writeln('3. Редактировать направление исполнителя.');

Writeln('0. Выход из подменю редактирования исполнителя.');

ReadNum(Menu);

case Menu of

1:

begin

Write('Введите новое имя исполнителя: ');

readln(ArtistList^.Artist.Name);

end;

2:

begin

Write('Введите страну исполнителя: ');

readln(ArtistList^.Artist.Country);

end;

3:

begin

Write('Введите направление исполнителя: ');

readln(ArtistList^.Artist.Direction);

end;

end;

until Menu = 0;

end;

// Редактировать информацию о исполнителе.

Procedure EditArtist(ArtistList: TAdrOfArtistList);

var

Flag: Boolean;

ID: Integer;

begin

WatchArtistList(ArtistList);

Write('Введите код исполнителя для редактирования: ');

readln(ID);

Flag := False;

while (ArtistList^.next <> nil) and not(Flag) do

begin

ArtistList := ArtistList^.next;

if ArtistList^.Artist.ID = ID then

begin

Flag := True;

EditArtistMenu(ArtistList);

end;

end;

if not(Flag) then

Writeln('Исполнителя с таким кодом не существует.');

Writeln;

end;

{ \\\\\\\\\\ Work with AlbumList ////////// }

// Просмотреть список альбомов.

Procedure WatchALbumList(AlbumList: TAdrOfAlbumList);

begin

Writeln('|-------------|-----------------|----------------------|------------|');

Writeln('| Код альбома | Код исполнителя | Название альбома | Год записи |');

Writeln('|-------------|-----------------|----------------------|------------|');

while AlbumList^.next <> nil do

begin

AlbumList := AlbumList^.next;

Writeln('|', AlbumList^.Album.ID:12, ' |', AlbumList^.Album.ID\_Artist:16,

' |', AlbumList^.Album.Name:21, ' |', AlbumList^.Album.Year:11, ' |');

end;

Writeln('|-------------|-----------------|----------------------|------------|');

Writeln;

end;

// Прочитать код артиста, предусмотрев его отсутствие.

Procedure ReadID\_Artist(var ID: Integer; ArtistList: TAdrOfArtistList);

var

Flag: Boolean;

ArtL: TAdrOfArtistList;

Menu: Integer;

begin

repeat

Write('Введите код исполнителя: ');

ReadNum(ID);

Flag := False;

ArtL := ArtistList^.next;

while (ArtL <> nil) and not(Flag) do

begin

if ArtL^.Artist.ID = ID then

Flag := True;

ArtL := ArtL^.next;

end;

if Flag = False then

begin

Writeln('Исполнителя с таким кодом не существует.');

Writeln('Желаете создать нового исполнителя?');

Writeln('1. Да. / 0. Нет (Ввести код заново).');

ReadNum(Menu);

if Menu = 1 then

begin

ID := ArtistList^.Max\_Id + 1;

InsertArtist(ArtistList);

Flag := True;

end;

end;

until Flag;

end;

// Проверка на существование альбома.

Function IsAlbumAlreadyExist(AlbumList: TAdrOfAlbumList;

TmpAlbum: TAlbum): Boolean;

begin

Result := False;

while AlbumList^.next <> nil do

begin

AlbumList := AlbumList^.next;

if (AlbumList^.Album.Name = TmpAlbum.Name) and

(AlbumList^.Album.ID\_Artist = TmpAlbum.ID\_Artist) and

(AlbumList^.Album.Year = TmpAlbum.Year) then

Result := True;

end;

end;

// Вставить альбом в список.

Procedure InsertAlbum(AlbumList: TAdrOfAlbumList; ArtistList: TAdrOfArtistList);

var

MaxId: Integer;

TmpAlbum: TAlbum;

begin

WatchArtistList(ArtistList);

ReadID\_Artist(TmpAlbum.ID\_Artist, ArtistList);

Write('Введите название альбома: ');

readln(TmpAlbum.Name);

Write('Введите год выпуска альбома: ');

ReadNum(TmpAlbum.Year);

if not(IsAlbumAlreadyExist(AlbumList, TmpAlbum)) then

begin

Inc(AlbumList^.Max\_Id);

MaxId := AlbumList^.Max\_Id;

while AlbumList^.next <> nil do

AlbumList := AlbumList^.next;

New(AlbumList^.next);

AlbumList := AlbumList^.next;

TmpAlbum.ID := MaxId;

AlbumList^.Album := TmpAlbum;

AlbumList^.next := nil;

end

else

Writeln('Такой альбом уже существует.');

Writeln;

end;

// Удалить альбом из списка.

Procedure DeleteAlbum(AlbumList: TAdrOfAlbumList; SongList: TAdrOfSongList;

CheckID: Integer);

var

IDForDelete: Integer;

Tmp: TAdrOfAlbumList;

TmpSongList: TAdrOfSongList;

Flag: Boolean;

begin

IDForDelete := CheckID;

if CheckID = 0 then

begin

WatchALbumList(AlbumList);

Write('Введите код альбома для удаления: ');

ReadNum(IDForDelete);

end;

Flag := False;

While Not(Flag) and (AlbumList^.next <> nil) do

begin

if AlbumList^.next^.Album.ID = IDForDelete then

begin

Flag := True;

Tmp := AlbumList^.next;

AlbumList^.next := AlbumList^.next^.next;

Dispose(Tmp);

end;

AlbumList := AlbumList^.next;

end;

if Not(Flag) then

Writeln('Альбома с таким кодом нет в списке.')

else

begin

New(TmpSongList);

TmpSongList^.next := SongList;

while SongList <> nil do

begin

if (SongList^.next <> nil) and

(SongList^.next^.Song.ID\_Album = IDForDelete) then

begin

DeleteSong(TmpSongList^.next, SongList^.next^.Song.ID);

SongList := TmpSongList;

end;

SongList := SongList^.next;

end;

end;

end;

Procedure InputALbumID(var ID: Integer; var S: TDataString);

begin

Write('Введите код альбома: ');

ReadNum(ID);

end;

Function ConditionAlbumID(var Element; var ID: Integer;

var S: TDataString): Boolean;

begin

Result := TAlbum(Element).ID = ID;

end;

Function ConditionAlbumID\_Artist(var Element; var ID: Integer;

var S: TDataString): Boolean;

begin

Result := TAlbum(Element).ID\_Artist = ID;

end;

Procedure InputALbumName(var ID: Integer; var S: TDataString);

begin

Write('Введите название альбома: ');

readln(S);

end;

Function ConditionAlbumName(var Element; var ID: Integer;

var S: TDataString): Boolean;

begin

Result := TAlbum(Element).Name = S;

end;

// Найти альбом в списке.

Procedure SearchAlbum(AlbumList: TAdrOfAlbumList; Input: PInput\_Search;

Cond: FCondEq\_Search);

var

SearchID: Integer;

SearchString: TDataString;

begin

Input(SearchID, SearchString);

Writeln('|-------------|-----------------|----------------------|------------|');

Writeln('| Код альбома | Код исполнителя | Название альбома | Год записи |');

Writeln('|-------------|-----------------|----------------------|------------|');

while (AlbumList^.next <> nil) do

begin

AlbumList := AlbumList^.next;

if Cond(AlbumList^.Album, SearchID, SearchString) then

begin

Writeln('|', AlbumList^.Album.ID:12, ' |', AlbumList^.Album.ID\_Artist:16,

' |', AlbumList^.Album.Name:21, ' |', AlbumList^.Album.Year:11, ' |');

end;

end;

Writeln('|-------------|-----------------|----------------------|------------|');

Writeln;

end;

// Найти все альбомы по коду исполнителя в списке.

Procedure SearchAlbumByID\_Artist(AlbumList: TAdrOfAlbumList;

var ArtistIndexes, AlbumIndexes: TArrayOfIndexes);

var

IndexArtist, IndexAlbum: Integer;

SearchString: TDataString;

Tmp: TAdrOfAlbumList;

begin

IndexAlbum := 0;

IndexArtist := 0;

Tmp := AlbumList;

while (Length(ArtistIndexes) <> 0) and (ArtistIndexes[IndexArtist] <> 0) do

begin

AlbumList := Tmp;

while (AlbumList^.next <> nil) do

begin

AlbumList := AlbumList^.next;

if ConditionAlbumID\_Artist(AlbumList^.Album, ArtistIndexes[IndexArtist],

SearchString) then

begin

if High(AlbumIndexes) < IndexAlbum then

Add10(AlbumIndexes);

AlbumIndexes[IndexAlbum] := AlbumList^.Album.ID;

Inc(IndexAlbum);

end;

end;

Inc(IndexArtist);

end;

end;

// Меню поиска в списке альбомов.

Procedure MenuSearchAlbum(AlbumList: TAdrOfAlbumList);

var

Menu: Integer;

begin

repeat

Writeln('Меню подпункта поиска в списке альбомов:');

Writeln('1. Поиск по коду альбома.');

Writeln('2. Поиск по коду исполнителя альбома.');

Writeln('3. Поиск по названию альбома.');

Writeln('0. Выйти из подпункта меню.');

ReadNum(Menu);

case Menu of

1:

SearchAlbum(AlbumList, InputALbumID, ConditionAlbumID);

2:

SearchAlbum(AlbumList, InputArtistID, ConditionAlbumID\_Artist);

3:

SearchAlbum(AlbumList, InputALbumName, ConditionAlbumName);

end;

until Menu = 0;

end;

// Выбор поля для редактирования альбома

Procedure EditAlbumMenu(AlbumList: TAdrOfAlbumList);

var

Menu: Integer;

begin

repeat

Writeln('Меню редактирования:');

Writeln('1. Редактировать название альбома.');

Writeln('2. Редактировать год издания альбома.');

Writeln('0. Выход из подменю редактирования исполнителя.');

ReadNum(Menu);

case Menu of

1:

begin

Write('Введите новое название альбома: ');

readln(AlbumList^.Album.Name);

end;

2:

begin

Write('Введите год записи: ');

ReadNum(AlbumList^.Album.Year);

end;

end;

until Menu = 0;

end;

// Редактировать информацию о альбоме.

Procedure EditAlbum(AlbumList: TAdrOfAlbumList);

var

Flag: Boolean;

ID: Integer;

begin

WatchALbumList(AlbumList);

Write('Введите код альбома для редактирования: ');

ReadNum(ID);

Flag := False;

while (AlbumList^.next <> nil) and not(Flag) do

begin

AlbumList := AlbumList^.next;

if AlbumList^.Album.ID = ID then

begin

Flag := True;

EditAlbumMenu(AlbumList);

end;

end;

if not(Flag) then

Writeln('Альбома с таким кодом не существует.');

Writeln;

end;

{ \\\\\\\\\\ Work with SongList ////////// }

// Просмотреть список песен.

Procedure WatchSongList(SongList: TAdrOfSongList);

begin

Writeln('|-----------|----------------------|-------------|--------------------|');

Writeln('| Код песни | Название песни | Код альбома | Длительность песни |');

Writeln('|-----------|----------------------|-------------|--------------------|');

while SongList^.next <> nil do

begin

SongList := SongList^.next;

Writeln('|', SongList^.Song.ID:10, ' |', SongList^.Song.Name:21, ' |',

SongList^.Song.ID\_Album:12, ' |', SongList^.Song.Length:19, ' |');

end;

Writeln('|-----------|----------------------|-------------|--------------------|');

Writeln;

end;

// Прочитать код альбома, предусмотрев его отсутствие.

Procedure ReadID\_Album(var ID: Integer; AlbumList: TAdrOfAlbumList;

ArtistList: TAdrOfArtistList);

var

Flag: Boolean;

AlbL: TAdrOfAlbumList;

Menu: Integer;

begin

Menu := 0;

repeat

Write('Введите код альбома: ');

ReadNum(ID);

Flag := False;

AlbL := AlbumList^.next;

while (AlbL <> nil) and not(Flag) do

begin

if AlbL^.Album.ID = ID then

Flag := True;

AlbL := AlbL^.next;

end;

if Flag = False then

begin

Writeln('Альбома с таким кодом не существует.');

Writeln('Желаете создать новый альбом?');

Writeln('1. Да. / 0. Нет (Ввести код заново).');

ReadNum(Menu);

if Menu = 1 then

begin

ID := AlbumList^.Max\_Id + 1;

InsertAlbum(AlbumList, ArtistList);

Flag := True;

end;

end;

until Flag;

end;

// Проверка на существование песни.

Function IsSongAlreadyExist(SongList: TAdrOfSongList; TmpSong: TSong): Boolean;

begin

Result := False;

while SongList^.next <> nil do

begin

SongList := SongList^.next;

if (SongList^.Song.Name = TmpSong.Name) and

(SongList^.Song.ID\_Album = TmpSong.ID\_Album) and

(SongList^.Song.Length = TmpSong.Length) then

Result := True;

end;

end;

// Вставить песню в список.

Procedure InsertSong(SongList: TAdrOfSongList; AlbumList: TAdrOfAlbumList;

ArtistList: TAdrOfArtistList);

var

MaxId: Integer;

Tmp: TAdrOfSongList;

TmpSong: TSong;

begin

WatchALbumList(AlbumList);

ReadID\_Album(TmpSong.ID\_Album, AlbumList, ArtistList);

Write('Введите название песни: ');

readln(TmpSong.Name);

Write('Введите длину песни в секундах: ');

ReadNum(TmpSong.Length);

if not(IsSongAlreadyExist(SongList, TmpSong)) then

begin

Inc(SongList^.Max\_Id);

MaxId := SongList^.Max\_Id;

Tmp := SongList^.next;

New(SongList^.next);

SongList := SongList^.next;

TmpSong.ID := MaxId;

SongList^.Song := TmpSong;

SongList^.next := Tmp;

end

else

Writeln('Такая песня уже существует.');

Writeln;

end;

// Удалить песню из списка.

Procedure DeleteSong(SongList: TAdrOfSongList; CheckID: Integer);

var

IDForDelete: Integer;

Tmp: TAdrOfSongList;

Flag: Boolean;

begin

IDForDelete := CheckID;

if CheckID = 0 then

begin

WatchSongList(SongList);

Write('Введите код песни для удаления: ');

ReadNum(IDForDelete);

end;

Flag := False;

While Not(Flag) and (SongList^.next <> nil) do

begin

if SongList^.next^.Song.ID = IDForDelete then

begin

Flag := True;

Tmp := SongList^.next;

SongList^.next := SongList^.next^.next;

Dispose(Tmp);

end;

SongList := SongList^.next;

end;

if Not(Flag) then

Writeln('Песни с таким кодом нет в списке.');

end;

Procedure InputSongID(var ID: Integer; var S: TDataString);

begin

Write('Введите код песни: ');

ReadNum(ID);

end;

Function ConditionSongID(var Element; var ID: Integer;

var S: TDataString): Boolean;

begin

Result := TSong(Element).ID = ID;

end;

Function ConditionSongID\_Album(var Element; var ID: Integer;

var S: TDataString): Boolean;

begin

Result := TSong(Element).ID\_Album = ID;

end;

Procedure InputSongName(var ID: Integer; var S: TDataString);

begin

Write('Введите название песни: ');

readln(S);

end;

Function ConditionSongName(var Element; var ID: Integer;

var S: TDataString): Boolean;

begin

Result := TSong(Element).Name = S;

end;

// Найти песню в списке.

Procedure SearchSong(SongList: TAdrOfSongList; Input: PInput\_Search;

Cond: FCondEq\_Search);

var

SearchID: Integer;

SearchString: TDataString;

begin

Input(SearchID, SearchString);

Writeln('|-----------|----------------------|-------------|--------------------|');

Writeln('| Код песни | Название песни | Код альбома | Длительность песни |');

Writeln('|-----------|----------------------|-------------|--------------------|');

while (SongList^.next <> nil) do

begin

SongList := SongList^.next;

if (Cond(SongList^.Song, SearchID, SearchString)) then

begin

Writeln('|', SongList^.Song.ID:10, ' |', SongList^.Song.Name:21, ' |',

SongList^.Song.ID\_Album:12, ' |', SongList^.Song.Length:19, ' |');

end;

end;

Writeln('|-----------|----------------------|-------------|--------------------|');

Writeln;

end;

// Найти песню в списке(Специальная функция 2).

Procedure SearchSongByArtist(ArtistList: TAdrOfArtistList;

AlbumList: TAdrOfAlbumList; SongList: TAdrOfSongList);

var

IndexAlbum: Integer;

SearchString: TDataString;

ArtistIndexes, AlbumIndexes: TArrayOfIndexes;

Tmp: TAdrOfSongList;

begin

SearchArtistByName(ArtistList, ArtistIndexes);

SearchAlbumByID\_Artist(AlbumList, ArtistIndexes, AlbumIndexes);

IndexAlbum := 0;

Writeln('|-----------|----------------------|-------------|--------------------|');

Writeln('| Код песни | Название песни | Код альбома | Длительность песни |');

Writeln('|-----------|----------------------|-------------|--------------------|');

Tmp := SongList;

while (Length(AlbumIndexes) <> 0) and (AlbumIndexes[IndexAlbum] <> 0) do

begin

SongList := Tmp;

while (SongList^.next <> nil) do

begin

SongList := SongList^.next;

if (ConditionSongID\_Album(SongList^.Song, AlbumIndexes[IndexAlbum],

SearchString)) then

begin

Writeln('|', SongList^.Song.ID:10, ' |', SongList^.Song.Name:21, ' |',

SongList^.Song.ID\_Album:12, ' |', SongList^.Song.Length:19, ' |');

end;

end;

Inc(IndexAlbum);

end;

Writeln('|-----------|----------------------|-------------|--------------------|');

Writeln;

end;

// Меню поиска в списке альбомов.

Procedure MenuSearchSong(ArtistList: TAdrOfArtistList;

AlbumList: TAdrOfAlbumList; SongList: TAdrOfSongList);

var

Menu: Integer;

begin

repeat

Writeln('Меню подпункта поиска в списке песен:');

Writeln('1. Поиск по коду песни.');

Writeln('2. Поиск по коду альбома песни.');

Writeln('3. Поиск по названию песни.');

Writeln('4. Поиск по имени исполнителя.');

Writeln('0. Выйти из подпункта меню.');

ReadNum(Menu);

case Menu of

1:

SearchSong(SongList, InputSongID, ConditionSongID);

2:

SearchSong(SongList, InputALbumID, ConditionSongID\_Album);

3:

SearchSong(SongList, InputSongName, ConditionSongName);

4:

SearchSongByArtist(ArtistList, AlbumList, SongList);

end;

until Menu = 0;

end;

// Выбор поля для редактирования песни

Procedure EditSongMenu(SongList: TAdrOfSongList);

var

Menu: Integer;

begin

repeat

Writeln('Меню редактирования:');

Writeln('1. Редактировать название песни.');

Writeln('2. Редактировать длительность песни.');

Writeln('0. Выход из подменю редактирования исполнителя.');

ReadNum(Menu);

case Menu of

1:

begin

Write('Введите название песни: ');

readln(SongList^.Song.Name);

end;

2:

begin

Write('Введите длительность: ');

ReadNum(SongList^.Song.Length);

end;

end;

until Menu = 0;

end;

// Редактировать информацию о песне.

Procedure EditSong(SongList: TAdrOfSongList);

var

Flag: Boolean;

ID: Integer;

begin

WatchSongList(SongList);

Write('Введите код песни для редактирования: ');

ReadNum(ID);

Flag := False;

while (SongList^.next <> nil) and not(Flag) do

begin

SongList := SongList^.next;

if SongList^.Song.ID = ID then

begin

Flag := True;

EditSongMenu(SongList);

end;

end;

if not(Flag) then

Writeln('Песни с таким кодом не существует.');

Writeln;

end;

end.

**unit MenuItems:**

unit MenuItems;

interface

uses AllTypesInProject, WorkWithLists, WorkWithFiles, PlayList;

Procedure MenuItem1\_ReadLists(var CurrSession: String;

ArtistList: TAdrOfArtistList; AlbumList: TAdrOfAlbumList;

SongList: TAdrOfSongList; var ArtistFile: TArtistFile;

var AlbumFile: TAlbumFile; var SongFile: TSongFile; var State: TStateOfFile);

Procedure MenuItem2\_WatchLists(ArtistList: TAdrOfArtistList;

AlbumList: TAdrOfAlbumList; SongList: TAdrOfSongList);

Procedure MenuItem3\_Sort(ArtistList: TAdrOfArtistList;

AlbumList: TAdrOfAlbumList; SongList: TAdrOfSongList);

Procedure MenuItem4\_Search(ArtistList: TAdrOfArtistList;

AlbumList: TAdrOfAlbumList; SongList: TAdrOfSongList);

Procedure MenuItem5\_Insert(ArtistList: TAdrOfArtistList;

AlbumList: TAdrOfAlbumList; SongList: TAdrOfSongList);

Procedure MenuItem6\_Delete(ArtistList: TAdrOfArtistList;

AlbumList: TAdrOfAlbumList; SongList: TAdrOfSongList);

Procedure MenuItem7\_Edit(ArtistList: TAdrOfArtistList;

AlbumList: TAdrOfAlbumList; SongList: TAdrOfSongList);

Procedure MenuItem8\_Playlist(const CurrSession: String;

ArtistList: TAdrOfArtistList; AlbumList: TAdrOfAlbumList;

SongList: TAdrOfSongList);

Procedure DeleteAllLists(var ArtistList: TAdrOfArtistList;

AlbumList: TAdrOfAlbumList; SongList: TAdrOfSongList);

implementation

Procedure MenuItem1\_ReadLists(var CurrSession: String;

ArtistList: TAdrOfArtistList; AlbumList: TAdrOfAlbumList;

SongList: TAdrOfSongList; var ArtistFile: TArtistFile;

var AlbumFile: TAlbumFile; var SongFile: TSongFile; var State: TStateOfFile);

var

Flag: Boolean;

begin

Flag := False;

if State = ListChanged then

begin

Flag := True;

State := NoFileInformation;

end;

case State of

NoFileInformation:

begin

if Flag then

State := ListChanged;

MenuReadFiles(State, CurrSession, ArtistList, AlbumList, SongList,

ArtistFile, AlbumFile, SongFile);

if State = FileExist then

Writeln('Данные успешно прочитаны.')

else

Writeln('Файлы не найдены. Были созданы новые в данной сессии. Возможность сохранения доступна.');

State := FileAlreadyRead;

end;

FileAlreadyRead:

begin

Writeln('Данные уже были прочитаны или недоступны.');

end;

end;

end;

Procedure MenuItem2\_WatchLists(ArtistList: TAdrOfArtistList;

AlbumList: TAdrOfAlbumList; SongList: TAdrOfSongList);

var

Menu: Integer;

begin

repeat

Writeln('Меню подпункта просмотра списков:');

Writeln('1. Просмотреть список исполнителей.');

Writeln('2. Просмотреть список альбомов.');

Writeln('3. Просмотреть список песен.');

Writeln('0. Выйти из подпункта меню.');

readNum(Menu);

case Menu of

1:

WatchArtistList(ArtistList);

2:

WatchAlbumList(AlbumList);

3:

WatchSongList(SongList);

end;

until Menu = 0;

end;

Procedure MenuItem3\_Sort(ArtistList: TAdrOfArtistList;

AlbumList: TAdrOfAlbumList; SongList: TAdrOfSongList);

begin

SortAllLists(ArtistList, AlbumList, SongList);

Writeln('Данные успешно отсортированы.');

end;

Procedure MenuItem4\_Search(ArtistList: TAdrOfArtistList;

AlbumList: TAdrOfAlbumList; SongList: TAdrOfSongList);

var

Menu: Integer;

begin

repeat

Writeln('Меню подпункта поиска:');

Writeln('1. Искать в списке исполнителей.');

Writeln('2. Искать в списке альбомов.');

Writeln('3. Искать в списке песен.');

Writeln('0. Выйти из подпункта меню.');

readNum(Menu);

case Menu of

1:

MenuSearchArtist(ArtistList);

2:

MenuSearchAlbum(AlbumList);

3:

MenuSearchSong(ArtistList, AlbumList, SongList);

end;

until Menu = 0;

end;

Procedure MenuItem5\_Insert(ArtistList: TAdrOfArtistList;

AlbumList: TAdrOfAlbumList; SongList: TAdrOfSongList);

var

Menu: Integer;

begin

repeat

Writeln('Меню подпункта вставки элемента:');

Writeln('1. Вставить элемент в список исполнителей.');

Writeln('2. Вставить элемент в список альбомов.');

Writeln('3. Вставить элемент в список песен.');

Writeln('0. Выйти из подпункта меню.');

readNum(Menu);

case Menu of

1:

InsertArtist(ArtistList);

2:

InsertAlbum(AlbumList, ArtistList);

3:

InsertSong(SongList, AlbumList, ArtistList);

end;

until Menu = 0;

end;

Procedure MenuItem6\_Delete(ArtistList: TAdrOfArtistList;

AlbumList: TAdrOfAlbumList; SongList: TAdrOfSongList);

var

Menu: Integer;

begin

repeat

Writeln('Меню подпункта удаления элемента:');

Writeln('1. Удалить элемент из списка исполнителей.');

Writeln('2. Удалить элемент из списка альбомов.');

Writeln('3. Удалить элемент из списка песен.');

Writeln('0. Выйти из подпункта меню.');

readNum(Menu);

case Menu of

1:

DeleteArtist(ArtistList, AlbumList, SongList);

2:

DeleteAlbum(AlbumList, SongList, 0);

3:

DeleteSong(SongList, 0);

end;

until Menu = 0;

end;

Procedure MenuItem7\_Edit(ArtistList: TAdrOfArtistList;

AlbumList: TAdrOfAlbumList; SongList: TAdrOfSongList);

var

Menu: Integer;

begin

repeat

Writeln('Меню подпункта редактирования элемента:');

Writeln('1. Редактировать элемент из списка исполнителей.');

Writeln('2. Редактировать элемент из списка альбомов.');

Writeln('3. Редактировать элемент из списка песен.');

Writeln('0. Выйти из подпункта меню.');

readNum(Menu);

case Menu of

1:

EditArtist(ArtistList);

2:

EditAlbum(AlbumList);

3:

EditSong(SongList);

end;

until Menu = 0;

end;

Procedure MenuItem8\_Playlist(const CurrSession: String;

ArtistList: TAdrOfArtistList; AlbumList: TAdrOfAlbumList;

SongList: TAdrOfSongList);

var

ArrOfPlaylists: TArrOfPlaylists;

begin

if CurrSession = '' then

Writeln('Для создания playlist-а выберите существующую сессию или создайте новую.')

else

begin

Writeln('ВНИМАНИЕ, ЕСЛИ PLAYLIST УЖЕ БЫЛ СОЗДАН, ТО ЕГО ДАННЫЕ БУДУТ УТЕРЯНЫ.');

MakePlayListMenu(ArtistList, AlbumList, SongList, ArrOfPlaylists);

WritePlaylistsInFile(ArrOfPlaylists, CurrSession);

DeletePlaylists(ArrOfPlaylists);

end;

end;

Procedure DeleteAllLists(var ArtistList: TAdrOfArtistList;

AlbumList: TAdrOfAlbumList; SongList: TAdrOfSongList);

var

TmpArt: TAdrOfArtistList;

TmpAlb: TAdrOfAlbumList;

TmpSng: TAdrOfSongList;

begin

while ArtistList <> nil do

begin

TmpArt := ArtistList;

ArtistList := ArtistList^.next;

Dispose(TmpArt);

end;

while AlbumList <> nil do

begin

TmpAlb := AlbumList;

AlbumList := AlbumList^.next;

Dispose(TmpAlb);

end;

while SongList <> nil do

begin

TmpSng := SongList;

SongList := SongList^.next;

Dispose(TmpSng);

end;

end;

end.

**unit Playlist:**

unit Playlist;

interface

uses

WorkWithLists, AllTypesInProject;

Procedure SortAllLists(ArtistList: TAdrOfArtistList; AlbumList: TAdrOfAlbumList;

SongList: TAdrOfSongList);

Procedure MakePlayListMenu(ArtistList: TAdrOfArtistList;

AlbumList: TAdrOfAlbumList; SongList: TAdrOfSongList;

var ArrOfPlayLists: TArrOfPlaylists);

Procedure DeletePlaylists(var Arr: TArrOfPlaylists);

Procedure WritePlaylistsInFile(const ArrOfPlayLists: TArrOfPlaylists;

const CurrSession: String);

implementation

// Бинарный поиск в массиве индексов

Function SearchInArr(const ArrIn: TArrayOfIndexes; const item: Integer)

: Integer;

var

Flag: Boolean;

i: Integer;

begin

i := Low(ArrIn);

Flag := true;

result := -1;

while Flag and (i <= High(ArrIn)) do

begin

if item = ArrIn[i] then

begin

result := i;

Flag := false;

end;

Inc(i);

end;

end;

// Поменять два элемента списка местами

Procedure Swap(var i, J);

var

Tmp: TAdrOfList;

A: TAdrOfList absolute i;

B: TAdrOfList absolute J;

begin

Tmp := A^.next;

A^.next := B^.next;

B^.next := Tmp;

Tmp := A^.next.next;

A^.next.next := B^.next.next;

B^.next.next := Tmp;

end;

// Компоратор исполнителей

Function ArtistCompareTo(Self, o: TAdrOfList;

const ArrIn: TArrayOfIndexes): Boolean;

begin

if o = nil then

result := false

else

result := Self^.Artist.Direction > o^.Artist.Direction;

end;

// Заполнить массив индексов альбомов и исполнителей

Procedure FillArrOfIndexes(var L; var ArrIn: TArrayOfIndexes);

var

List: TAdrOfList;

i: Integer;

begin

List := TAdrOfList(L);

i := 0;

while List^.next <> nil do

begin

List := List^.next;

if i > High(ArrIn) then

Add10(ArrIn);

ArrIn[i] := List^.Artist.ID;

Inc(i);

end;

end;

// Компоратор альбомов

Function AlbumCompareTo(Self, o: TAdrOfList;

const ArrIn: TArrayOfIndexes): Boolean;

var

P: Boolean;

begin

if o = nil then

result := false

else

begin

P := SearchInArr(ArrIn, Self^.Album.ID\_Artist) >

SearchInArr(ArrIn, o^.Album.ID\_Artist);

result := (P) or ((Self^.Album.ID\_Artist = o^.Album.ID\_Artist) and

(Self^.Album.Year > o^.Album.Year));

end;

end;

// Компоратор песен

Function SongCompareTo(Self, o: TAdrOfList;

const ArrIn: TArrayOfIndexes): Boolean;

var

P: Boolean;

begin

if o = nil then

result := false

else

begin

P := SearchInArr(ArrIn, Self^.Song.ID\_Album) >

SearchInArr(ArrIn, o^.Song.ID\_Album);

result := (P) or ((Self^.Song.ID\_Album = o^.Song.ID\_Album) and

(Self^.Song.Length > o^.Song.Length));

end;

end;

// Сортировка прямым выбором для любого списка

Procedure SelectionSort(var L; const ArrIn: TArrayOfIndexes;

CompareTo: FCompareTo);

var

List: TAdrOfList;

TmpEl, TmpList: TAdrOfList;

begin

List := TAdrOfList(L);

while List^.next <> nil do

begin

TmpEl := List;

TmpList := List^.next;

while TmpList^.next <> nil do

begin

if CompareTo(TmpEl^.next, TmpList^.next, ArrIn) then

TmpEl := TmpList;

TmpList := TmpList^.next;

end;

Swap(TmpEl, List);

List := List^.next;

end;

end;

// Процедура для сортировок всех списков разом

Procedure SortAllLists(ArtistList: TAdrOfArtistList; AlbumList: TAdrOfAlbumList;

SongList: TAdrOfSongList);

var

ArrInArtist, ArrInAlbum: TArrayOfIndexes;

begin

SelectionSort(ArtistList, [], ArtistCompareTo);

FillArrOfIndexes(ArtistList, ArrInArtist);

SelectionSort(AlbumList, ArrInArtist, AlbumCompareTo);

FillArrOfIndexes(AlbumList, ArrInAlbum);

SelectionSort(SongList, ArrInAlbum, SongCompareTo);

end;

Procedure ReadTime(var Time: Integer);

var

Ind, ErrH, ErrM, ErrS, Hours, Minutes, Seconds: Integer;

TimeStr, SHours, SMinutes, SSeconds: TDataString;

Flag, Err, Format: Boolean;

begin

repeat

ErrH := 0;

ErrM := 0;

ErrS := 0;

Flag := false;

Readln(TimeStr);

Ind := Pos(':', String(TimeStr));

SHours := TDataString(Copy(TimeStr, 0, Ind - 1));

Delete(TimeStr, 1, Ind);

Ind := Pos(':', String(TimeStr));

SMinutes := TDataString(Copy(TimeStr, 0, Ind - 1));

Delete(TimeStr, 1, Ind);

SSeconds := TimeStr;

Val(String(SHours), Hours, ErrH);

Val(String(SMinutes), Minutes, ErrM);

Val(String(SSeconds), Seconds, ErrS);

Time := Hours \* 3600 + Minutes \* 60 + Seconds;

Err := (ErrH = 0) and (ErrM = 0) and (ErrM = 0);

Format := (Minutes < 60) and (Minutes >= 0) and (Seconds < 60) and

(Seconds >= 0);

if Err and Format then

Flag := true

else

Write('Неверный формат ввода, введите снова: ');

until Flag = true;

end;

// Заполнить массив индексов исполнителей по направлению

Procedure FillArrOfArtistInd(ArtistList: TAdrOfArtistList; Dir: TDataString;

var ArrIn: TArrayOfIndexes);

var

i: Integer;

begin

i := 0;

while (ArtistList^.next <> nil) and

(ArtistList^.next.Artist.Direction <> Dir) do

begin

ArtistList := ArtistList^.next;

end;

while (ArtistList^.next <> nil) and

(ArtistList^.next.Artist.Direction = Dir) do

begin

ArtistList := ArtistList^.next;

if i > High(ArrIn) then

begin

Add10(ArrIn);

end;

ArrIn[i] := ArtistList^.Artist.ID;

Inc(i);

end;

end;

// Заполнить массив индексов альбомов с помощью массива исполнителей

Procedure FillArrOfAlbumInd(AlbumList: TAdrOfAlbumList; Year: Integer;

var ArrIndArt, ArrIndAlb: TArrayOfIndexes);

var

i: Integer;

begin

i := 0;

if Length(ArrIndArt) <> 0 then

begin

while (AlbumList^.next <> nil) and

(AlbumList^.next.Album.ID\_Artist <> ArrIndArt[Low(ArrIndArt)]) do

begin

AlbumList := AlbumList^.next;

end;

while (AlbumList^.next <> nil) and

(SearchInArr(ArrIndArt, AlbumList^.next.Album.ID\_Artist) <> -1) do

begin

AlbumList := AlbumList^.next;

if AlbumList^.Album.Year >= Year then

begin

if i > High(ArrIndAlb) then

begin

Add10(ArrIndAlb);

end;

ArrIndAlb[i] := AlbumList^.Album.ID;

Inc(i);

end;

end;

end;

end;

// Заполнить список песен, которые подходят по году и направлению

Procedure MakeListOfAllSong(SongList, ListOfAllSong: TAdrOfSongList;

var ArrIndAlb: TArrayOfIndexes);

var

MaxID: ^Integer;

begin

MaxID := @ListOfAllSong^.Max\_Id;

if Length(ArrIndAlb) <> 0 then

begin

while (SongList^.next <> nil) and

(SongList^.next.Song.ID\_Album <> ArrIndAlb[Low(ArrIndAlb)]) do

begin

SongList := SongList^.next;

end;

while (SongList^.next <> nil) do

begin

SongList := SongList^.next;

if (SearchInArr(ArrIndAlb, SongList^.Song.ID\_Album) <> -1) then

begin

Inc(MaxID^);

New(ListOfAllSong^.next);

ListOfAllSong := ListOfAllSong^.next;

ListOfAllSong^.Song := SongList^.Song;

ListOfAllSong^.next := nil;

end;

end;

end;

end;

// Процедура создания всех вариаций playlist-ов без повторений

Procedure MakePlaylist(ListOfAllSong: TAdrOfSongList; const PLength: Integer;

var Arr: TArrOfArrOfIndexes);

var

ArrInd: TArrayOfIndexes;

i, Sum: Integer;

{ Так как процедура рекурсивная, то создана внутренная

процедура для уменьшения количества переданных параметров }

Procedure MPlaylist(LSong: TAdrOfSongList; i: Integer);

var

Flag: Boolean;

begin

while LSong^.next <> nil do

begin

Flag := false;

LSong := LSong^.next;

Sum := Sum + LSong^.Song.Length;

if Sum > PLength then

begin

Sum := Sum - LSong^.Song.Length;

end

else if Sum < PLength then

begin

if i > High(ArrInd) then

Add10(ArrInd);

ArrInd[i] := LSong^.Song.ID;

Inc(i);

Flag := true;

MPlaylist(LSong, i);

end

else

begin

if i > High(ArrInd) then

Add10(ArrInd);

ArrInd[i] := LSong^.Song.ID;

SetLength(Arr, Length(Arr) + 1);

Arr[High(Arr)] := Copy(ArrInd);

Inc(i);

Flag := true;

end;

if Flag then

begin

Sum := Sum - LSong^.Song.Length;

Dec(i);

ArrInd[i] := 0;

end;

end;

end;

begin

SetLength(ArrInd, 0);

SetLength(Arr, 0);

i := 0;

Sum := 0;

MPlaylist(ListOfAllSong, i);

end;

// Поиск данных песни по коду

Function FindSong(SongList: TAdrOfSongList; ID: Integer): TSong;

var

Flag: Boolean;

begin

Flag := true;

while (SongList^.next <> nil) and Flag do

begin

SongList := SongList^.next;

if SongList^.Song.ID = ID then

Flag := false;

end;

result := SongList^.Song;

end;

// Процедура создания массива playlist-ов

Procedure MakeArrOfPlaylists(ListOfAllSong: TAdrOfSongList;

var ArrOfPlayLists: TArrOfPlaylists; PlaylistsArr: TArrOfArrOfIndexes);

var

i, J: Integer;

Tmp: TAdrOfSongList;

begin

SetLength(ArrOfPlayLists, Length(PlaylistsArr));

for i := Low(ArrOfPlayLists) to High(ArrOfPlayLists) do

begin

New(ArrOfPlayLists[i]);

ArrOfPlayLists[i]^.Max\_Id := 0;

ArrOfPlayLists[i]^.next := nil;

J := 0;

Tmp := ArrOfPlayLists[i];

while (J < Length(PlaylistsArr[i])) and (PlaylistsArr[i, J] <> 0) do

begin

New(Tmp^.next);

Tmp := Tmp^.next;

Tmp.Song := FindSong(ListOfAllSong, PlaylistsArr[i, J]);

Tmp^.next := nil;

Inc(J);

end;

end;

end;

// Проверка на наличие кода исполнителя в массиве

function IsArrHasID(Arr: TArrayOfIndexes; ID: Integer): Boolean;

var

i: Integer;

begin

i := 0;

result := false;

while Not(result) and (i <= High(Arr)) do

begin

if Arr[i] = ID then

result := true;

Inc(i);

end;

end;

// Запись всех кодов альбомов из кодов песен

Procedure CalcCountOfAlbum(ArrOfPlayLists: TArrOfPlaylists;

var ArrOfArrAlbumIndexes: TArrOfArrOfIndexes);

var

i, J: Integer;

Tmp: TAdrOfSongList;

begin

SetLength(ArrOfArrAlbumIndexes, Length(ArrOfPlayLists));

for i := Low(ArrOfPlayLists) to High(ArrOfPlayLists) do

begin

Tmp := ArrOfPlayLists[i];

J := 0;

while Tmp^.next <> nil do

begin

Tmp := Tmp^.next;

if Not(IsArrHasID(ArrOfArrAlbumIndexes[i], Tmp^.Song.ID\_Album)) then

begin

if J > High(ArrOfArrAlbumIndexes[i]) then

Add10(ArrOfArrAlbumIndexes[i]);

ArrOfArrAlbumIndexes[i, J] := Tmp^.Song.ID\_Album;

Inc(J);

end;

end;

end;

end;

// Найти код исполнителя по коду альбома

Function FindArtistID(AlbumList: TAdrOfAlbumList; ID: Integer): Integer;

var

Flag: Boolean;

begin

Flag := true;

while (AlbumList^.next <> nil) and Flag do

begin

AlbumList := AlbumList^.next;

if AlbumList^.Album.ID = ID then

Flag := false;

end;

result := AlbumList^.Album.ID\_Artist;

end;

// Записать количество исполнителей для сортировки

Procedure WriteCountOfArtist(ArrOfPlayLists: TArrOfPlaylists;

ArrOfArrArtistIndexes: TArrOfArrOfIndexes);

var

i: Integer;

J: Integer;

begin

for i := Low(ArrOfPlayLists) to High(ArrOfPlayLists) do

begin

J := 0;

while ArrOfArrArtistIndexes[i, J] <> 0 do

begin

Inc(ArrOfPlayLists[i]^.Max\_Id);

Inc(J);

end;

end;

end;

// Рассчитать количество исполнителей, по количеству альбомов

Procedure CalcCountOfArtist(ArrOfPlayLists: TArrOfPlaylists;

AlbumList: TAdrOfAlbumList);

var

ArrOfArrArtistIndexes, ArrOfArrAlbumIndexes: TArrOfArrOfIndexes;

i, J, ID, Index: Integer;

begin

CalcCountOfAlbum(ArrOfPlayLists, ArrOfArrAlbumIndexes);

SetLength(ArrOfArrArtistIndexes, Length(ArrOfArrAlbumIndexes));

for i := Low(ArrOfArrArtistIndexes) to High(ArrOfArrArtistIndexes) do

begin

J := 0;

Index := 0;

while (J < Length(ArrOfArrAlbumIndexes[i])) and

(ArrOfArrAlbumIndexes[i, J] <> 0) do

begin

ID := FindArtistID(AlbumList, ArrOfArrAlbumIndexes[i, J]);

if Not(IsArrHasID(ArrOfArrArtistIndexes[i], ID)) then

begin

if index > High(ArrOfArrArtistIndexes[i]) then

Add10(ArrOfArrArtistIndexes[i]);

ArrOfArrArtistIndexes[i, index] := ID;

Inc(Index);

end;

Inc(J);

end;

end;

WriteCountOfArtist(ArrOfPlayLists, ArrOfArrArtistIndexes);

end;

// Сортировка Шелла для массива playlist-ов

Procedure ShellSort(var A: TArrOfPlaylists);

var

gap, i, J, Len: Integer;

Temp: TAdrOfSongList;

begin

Len := High(A);

gap := Len div 2;

while gap > 0 do

begin

for i := gap to Len do

begin

Temp := A[i];

J := i;

while (J >= gap) and (A[J - gap]^.Max\_Id > Temp^.Max\_Id) do

begin

A[J] := A[J - gap];

J := J - gap;

end;

A[J] := Temp;

end;

gap := gap div 2;

end;

end;

// Вывести все playlist-ы

Procedure WatchAllPlaylists(ArrOfPlayLists: TArrOfPlaylists);

var

i: Integer;

begin

for i := Low(ArrOfPlayLists) to High(ArrOfPlayLists) do

WatchSongList(ArrOfPlayLists[i]);

end;

// 2-ой компоратор песен для сортировки по длине

Function SongCompareTo2(Self, o: TAdrOfList;

const ArrIn: TArrayOfIndexes): Boolean;

begin

if o = nil then

result := false

else

begin

result := Self^.Song.Length > o^.Song.Length;

end;

end;

Procedure DeletePlaylists(var Arr: TArrOfPlaylists);

var

i: Integer;

Tmp, TmpSongList: TAdrOfSongList;

begin

for i := Low(Arr) to High(Arr) do

begin

TmpSongList := Arr[i];

Arr[i] := nil;

while TmpSongList <> nil do

begin

Tmp := TmpSongList;

TmpSongList := TmpSongList^.next;

Dispose(Tmp);

end;

end;

SetLength(Arr, 0);

end;

// Меню создания playlist-а

Procedure MakePlayListMenu(ArtistList: TAdrOfArtistList;

AlbumList: TAdrOfAlbumList; SongList: TAdrOfSongList;

var ArrOfPlayLists: TArrOfPlaylists);

var

ArrIndArtist, ArrIndAlbum: TArrayOfIndexes;

Dir: TDirString;

PLength, Year: Integer;

ListOfAllSong: TAdrOfSongList;

PlaylistsInArrForm: TArrOfArrOfIndexes;

begin

SortAllLists(ArtistList, AlbumList, SongList);

Writeln('Меню создания playlist-ов:');

Write('Введите направление исполнителя: ');

Readln(Dir);

Writeln('Введите длину Playlist-а в формате: чч:мм:сс.');

Writeln('(Если часы и/или минуты равны 0-ю, 0-и необходимо записать, пример:');

Writeln('00:00:45 / 00:45:00).');

Write('Длина: ');

ReadTime(PLength);

Write('Введите год, с которого выбирать песни: ');

ReadNum(Year);

FillArrOfArtistInd(ArtistList, Dir, ArrIndArtist);

FillArrOfAlbumInd(AlbumList, Year, ArrIndArtist, ArrIndAlbum);

New(ListOfAllSong);

ListOfAllSong^.next := nil;

ListOfAllSong^.Max\_Id := 0;

MakeListOfAllSong(SongList, ListOfAllSong, ArrIndAlbum);

SelectionSort(ListOfAllSong, [], SongCompareTo2);

SetLength(ArrIndArtist, 0);

SetLength(ArrIndAlbum, 0);

MakePlaylist(ListOfAllSong, PLength, PlaylistsInArrForm);

MakeArrOfPlaylists(ListOfAllSong, ArrOfPlayLists, PlaylistsInArrForm);

SetLength(PlaylistsInArrForm, 0);

CalcCountOfArtist(ArrOfPlayLists, AlbumList);

Writeln('Количество playlist-ов: ', Length(ArrOfPlayLists));

ShellSort(ArrOfPlayLists);

WatchAllPlaylists(ArrOfPlayLists);

end;

Procedure WritePlaylistsInFile(const ArrOfPlayLists: TArrOfPlaylists;

const CurrSession: String);

var

i: Integer;

TmpSongList: TAdrOfSongList;

PlayListFile: TextFile;

begin

AssignFile(PlayListFile, CurrSession + '\PlayListFile.txt');

Rewrite(PlayListFile);

Writeln(PlayListFile, 'Количество playlist-ов: ', Length(ArrOfPlayLists));

for i := Low(ArrOfPlayLists) to High(ArrOfPlayLists) do

begin

TmpSongList := ArrOfPlayLists[i];

Writeln(PlayListFile,

'|-----------|----------------------|-------------|--------------------|');

Writeln(PlayListFile,

'| Код песни | Название песни | Код альбома | Длительность песни |');

Writeln(PlayListFile,

'|-----------|----------------------|-------------|--------------------|');

while TmpSongList^.next <> nil do

begin

TmpSongList := TmpSongList^.next;

Writeln(PlayListFile, '|', TmpSongList^.Song.ID:10, ' |',

TmpSongList^.Song.Name:21, ' |', TmpSongList^.Song.ID\_Album:12, ' |',

TmpSongList^.Song.Length:19, ' |');

end;

Writeln(PlayListFile,

'|-----------|----------------------|-------------|--------------------|');

Writeln(PlayListFile, '');

end;

CloseFile(PlayListFile);

End;

end.