|  |  |  |
| --- | --- | --- |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное автономное  образовательное учреждение высшего образования  «Пермский государственный национальный  исследовательский университет» | | |
|  | Институт компьютерных наук и технологий | |
| **ОТЧЁТ**  по индивидуальной работе №1  по дисциплине «Языки программирования»  Вариант 14 | | |
|  | | Работу выполнил  студент группы ПМИ-9-10-2023 1 курса  \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Тарутин И.А.  «\_\_\_» \_\_\_\_\_\_\_\_2023 г. |
| Работу проверил  \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кнутова Н.С.  «\_\_\_» \_\_\_\_\_\_\_\_\_ 2023 г. |
| Пермь 2023 | | |

СОДЕРЖАНИЕ

[Постановка задачи 3](#_Toc153130027)

[Алгоритм решения 3](#_Toc153130028)

[Тестирование 3](#_Toc153130029)

[Код программы 3](#_Toc153130030)

[Инструкция по применению стилей и оформлению работы 4](#_Toc153130031)

# Постановка задачи

Создать файл записей, в котором хранится информация об аудиозаписях в медиатеке: исполнитель, название трека, альбом, год выпуска, длительность, количество прослушиваний. Разработать и реализовать программу "Медиатека", которая считывает исходную информацию и позволяет на основе неё создавать следующие отчёты:

1.      Список всех аудиозаписей, отсортированный по следующему ключу: исполнитель (по возрастанию) + год выпуска (по убыванию) + количество прослушиваний (по убыванию).

2.      Список всех аудиозаписей конкретного исполнителя (вводится с клавиатуры), отсортированный по следующему ключу: альбом (по убыванию) + название трека (по возрастанию).

3.      Список всех аудиозаписей, выпущенных в период с N1 до N2 года (вводятся с клавиатуры), отсортированный по следующему ключу: год выпуска (по убыванию) + исполнитель (по возрастанию).

Создать базу аудиозаписей, включающую не менее 25 записей и на основе неё сформировать все указанные списки (можно использовать вложенные списки и словари) и на основе неё сформировать все указанные списки). Взаимодействие с базой данных обеспечивается отдельными функциями – добавление/удаление/изменение записи, вывода на экран всей базы и отдельно сформированных списков. Интерфейс пользователя удобнее организовать в виде меню с выбором соответствующего действия. В программе обязательно наличие дружественного интерфейса и защиты от некорректного пользовательского ввода.  База должна содержать такие записи, чтобы во всех списках явно прослеживался заданный вид сортировки по всем ключам. Для сортировки записей использовать метод Хоара.

# Алгоритм решения

1. Инициализация базы данных:

Создаем класс `Song` с атрибутами `artist`, `title`, `album`, `year`, `duration`, и `listens`.

Создаем глобальный список `song\_list`, который будет хранить объекты этого класса.

2. Меню:

Создаем рекурсивную функцию Menu для отображения возможностей программы и вызова соответствующих функций, выбранных пользователем, для взаимодействия с базой данный. После выполнения выбранного действия Меnu будет рекурсивно вызвана для дальнейших указаний

3. Считывание БД:

Создаем функцию `Read` для заполнения списка `song\_list`

4. Добавление записей:

Пишем функцию `AddElement`, которая добавляет новую запись в БД

5. Удаление записей:

Пишем функцию `DeleteElement`, которая удаляет запись из БД (по индексу/названию песни)

6. Изменение записей:

Пишем функцию `СhangeElement`, которая изменяет запись, используя функцию `DeleteElement` для удаления записи, затем `AddElement` для добавления новой на место старой.

\*\* Функции `СhangeElement`, `DeleteElement` используют вспомогательную функцию `ChooseElement` для поиска индекса нужного элемента и передачи его обратно

7. Печать всего списка:

Пишем функцию `PrintList` для печати списка

8. Выход из программы:

Пишем функцию которая будет выкидывать нас из программы, спрашивая сохранить изменения БД или нет

9. Сортировка списка всех аудиозаписей:

Пишем функцию `SortAll`, которая сортирует весь список сначала по исполнителю (по возрастанию), если исполнитель одинаковый, то по году выпуска (по убыванию), если и год одинаковый то по количеству прослушиваний (по убыванию)

10. Список всех аудиозаписей конкретного исполнителя:

Пишем функцию `SortOneSinger`, которая уже принимает подсписок с нужным исполнителем (реализация создания подсписка вне функции) сортирует весь подсписок сначала по альбому (по убыванию), затем если альбомы одинаковые, то по названию трека (по возрастанию).

11. Список всех аудиозаписей, выпущенных в период с N1 до N2 года:

Пишем функцию `SortOneSinger`, которая уже принимает подсписок с нужным периодом лет (реализация создания подсписка вне функции) сортирует весь подсписок сначала по году выпуска (по убыванию), затем если они одинаковые то по исполнителю (по возрастанию).

\*\* Все сортировки выполнены алгоритмом Хоара, сновная идея которого заключается в разделении списка на подсписки, сортировке этих подсписков и последующем объединении отсортированных подсписков в один отсортированный список.

12. \*\*Пример использования:\*\* Создайте несколько записей в базе данных, взаимодействуйте с программой через меню, проверьте корректность вывода и сортировки данных.

# Тестирование

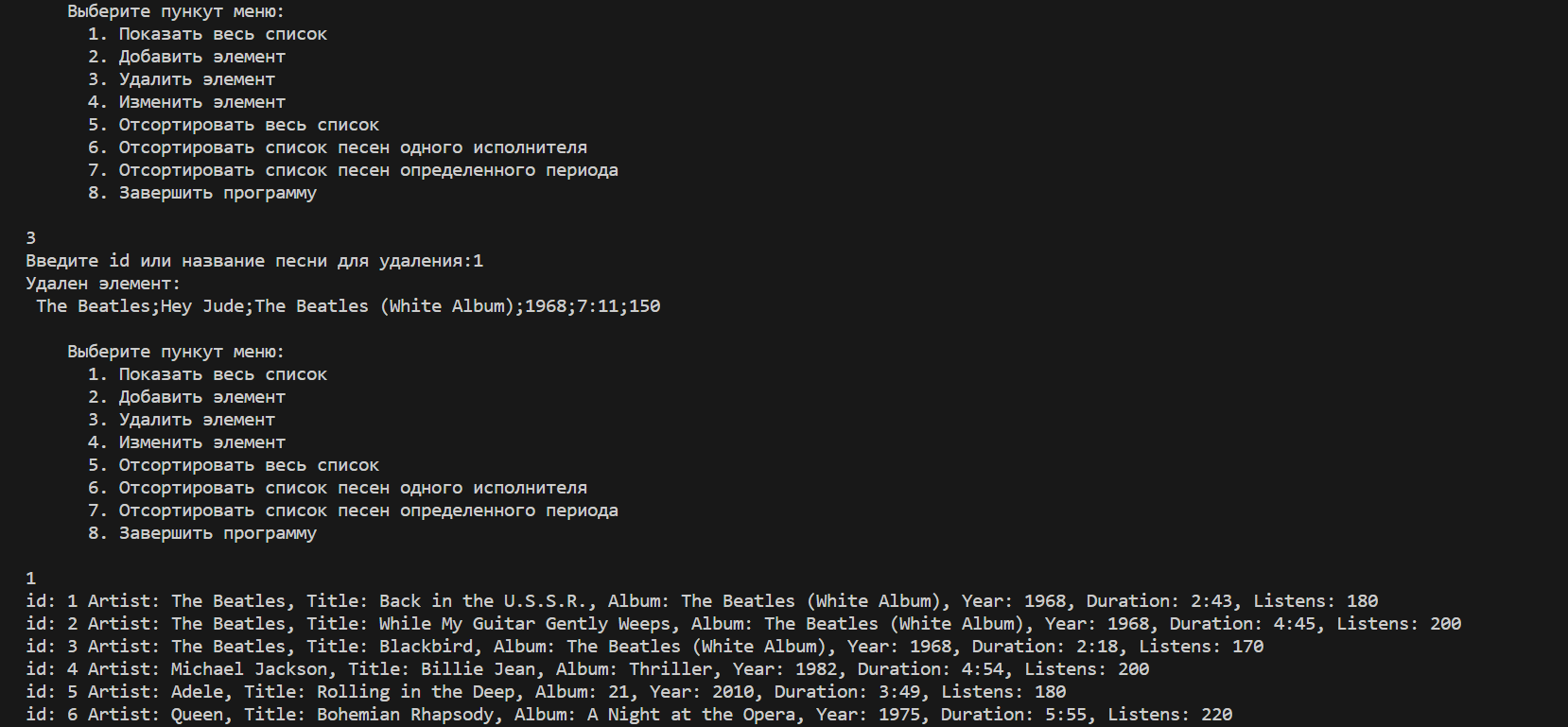
Тестирование всех элементов меню:

1. Вывод всего списка

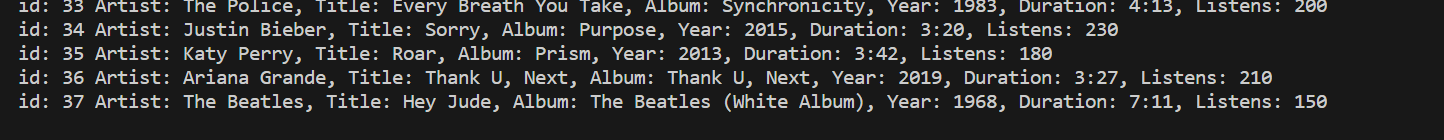
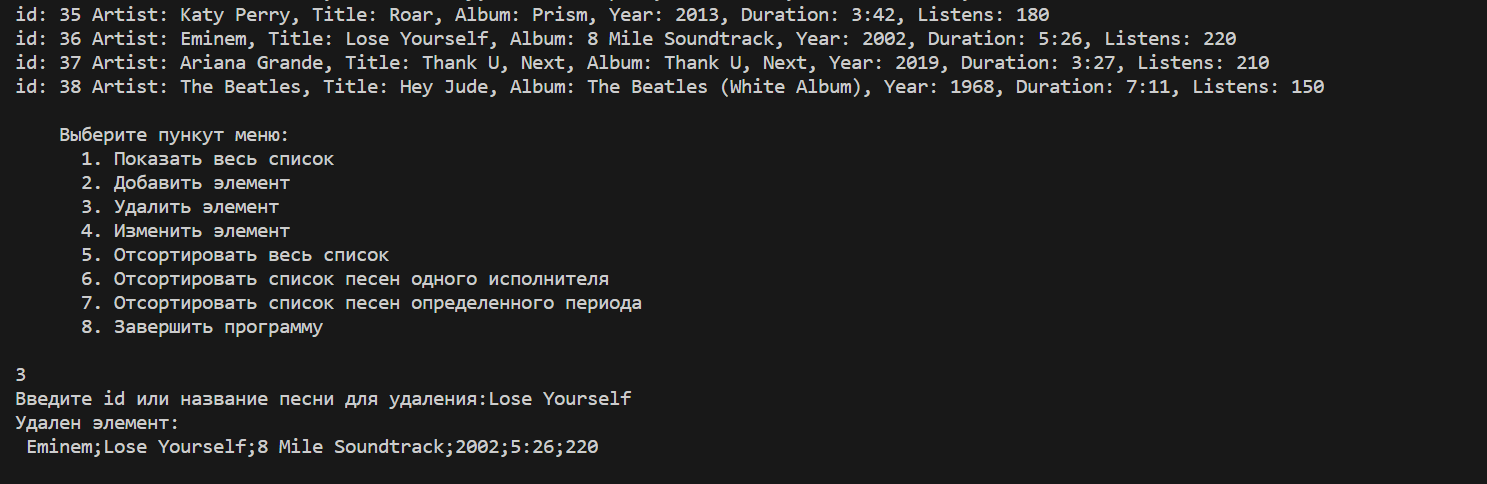


2. Удаление элемента

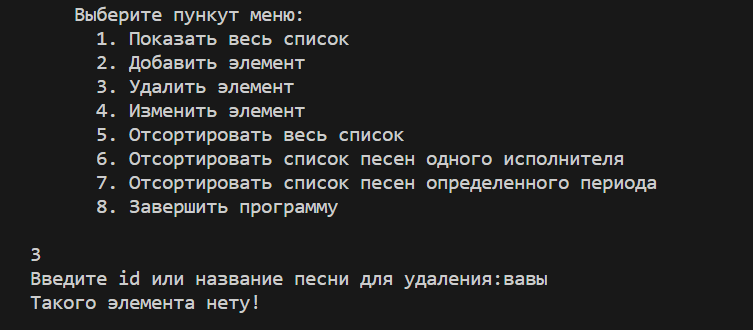
\*\*по индексу



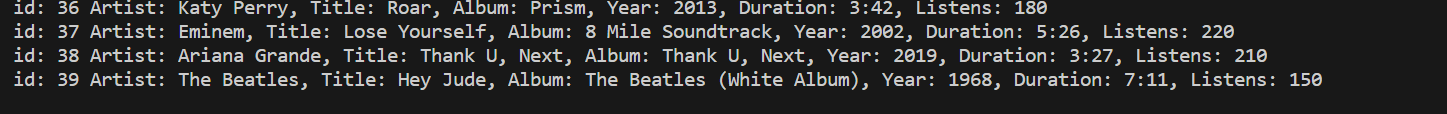
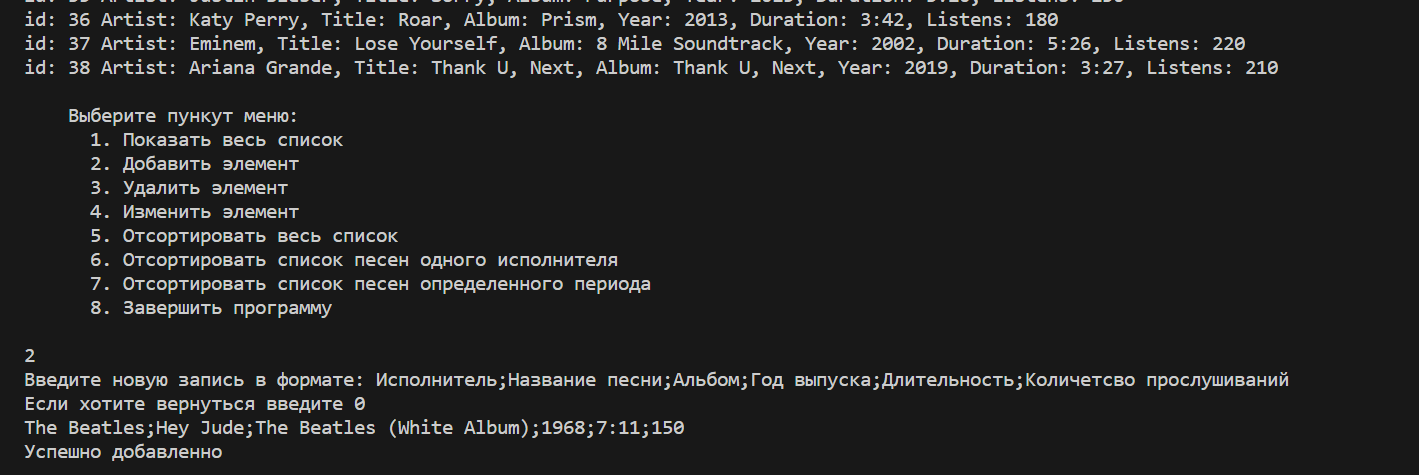
\*\*по названию песни



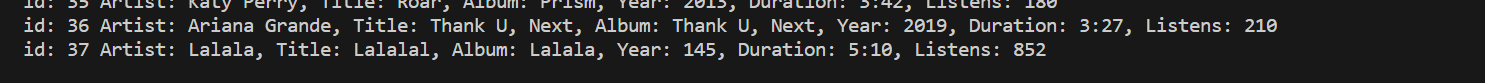
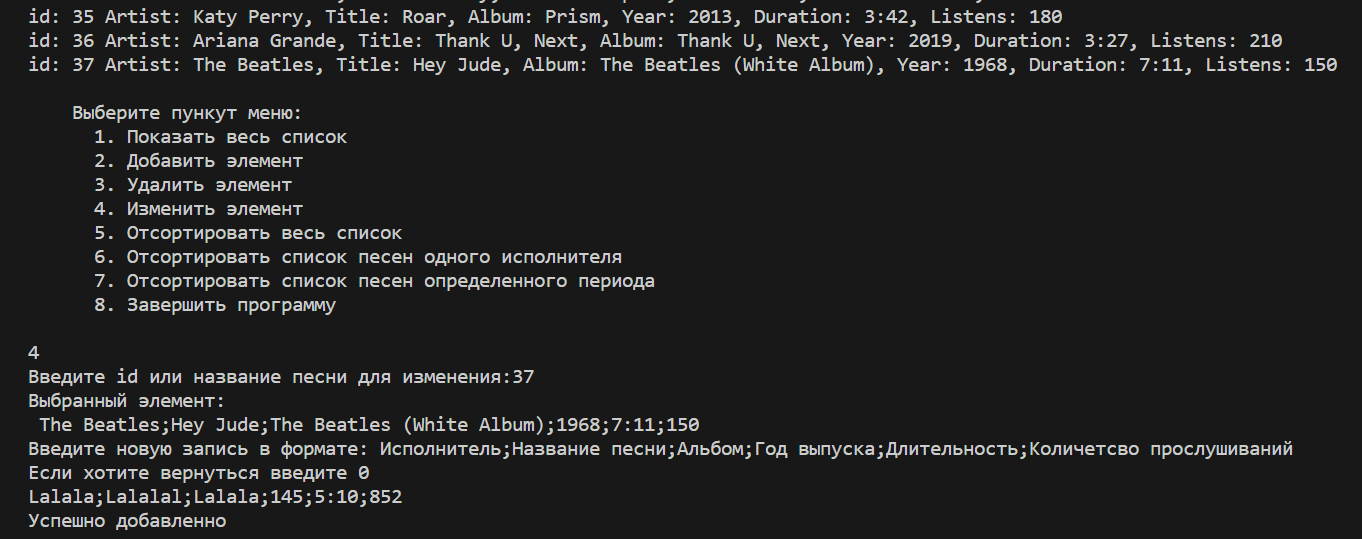
\*\*неверные данные



3. Добавление элемента (На первом скрине видно на что список заканчивался, на втором то на что заканчивается после добавления)

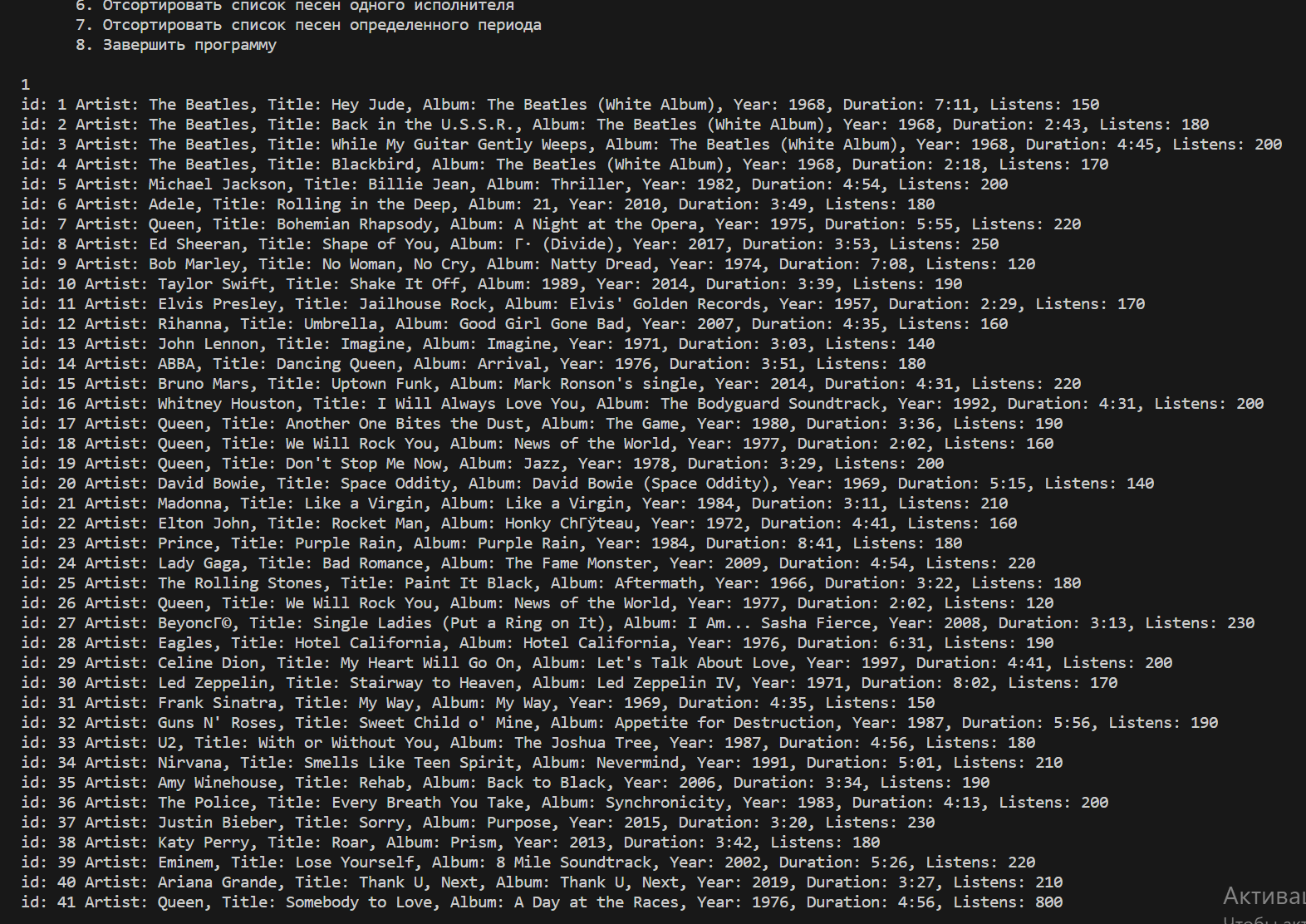


4. Изменение элемента (тк использует функции выше смысла проверять из еще раз нет)

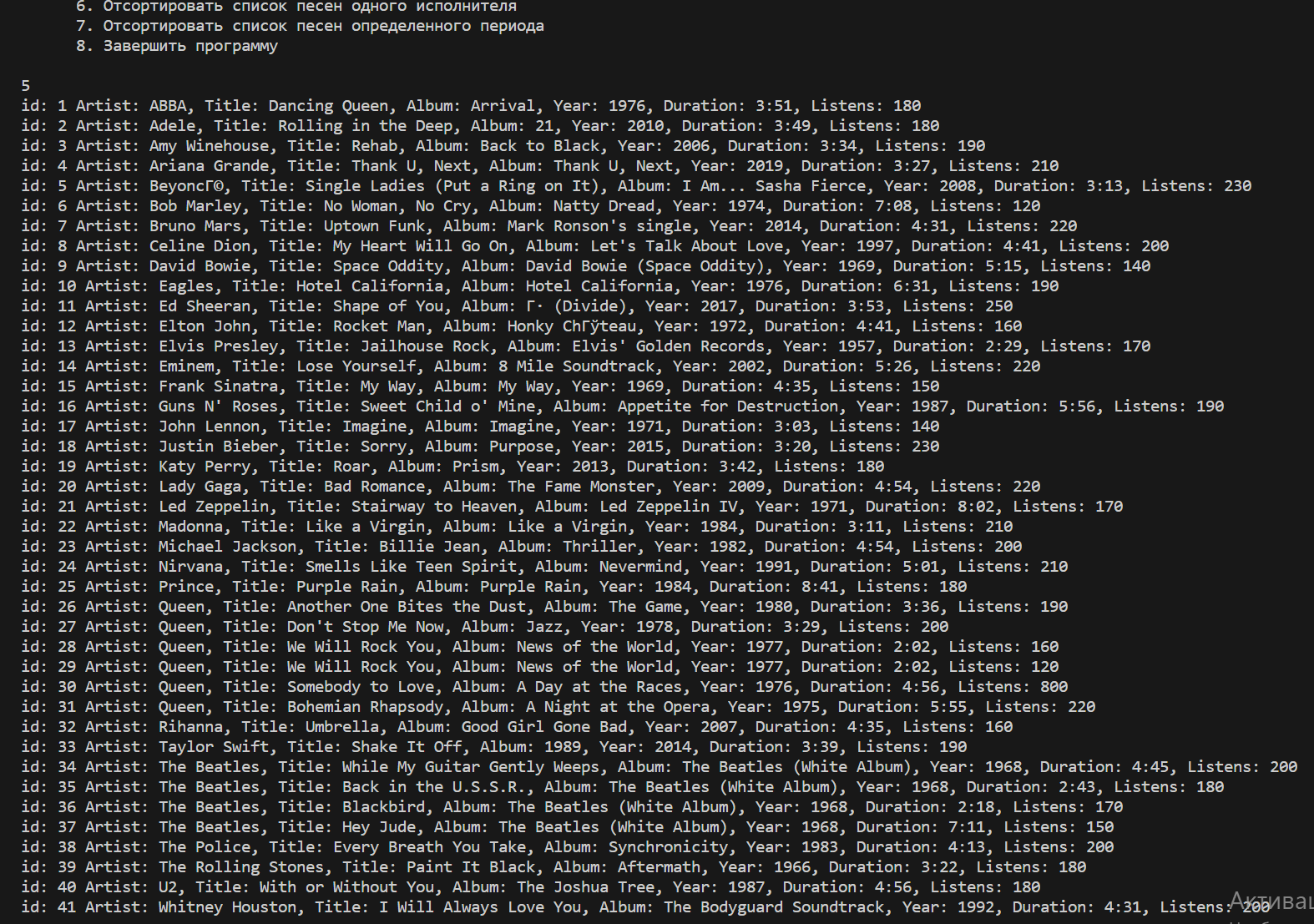


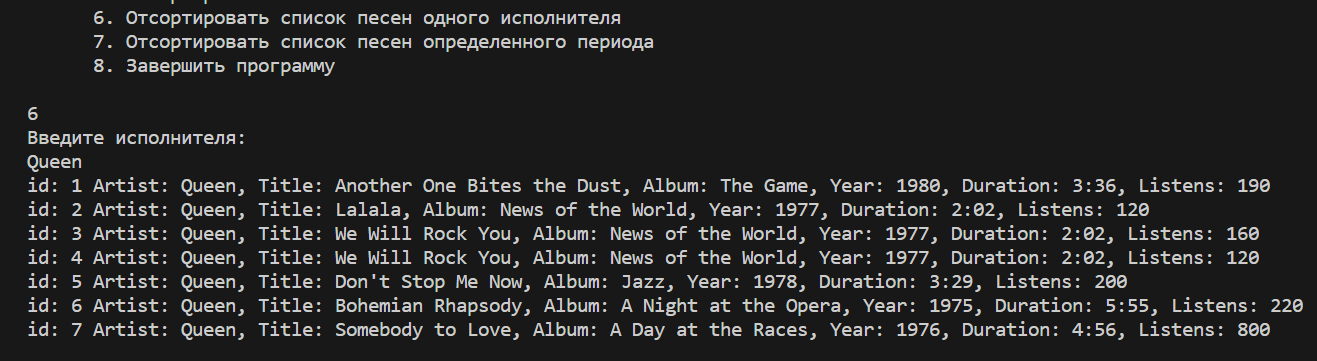
5. Сортировка всего списка (артист (по возраст), год выпуска (по убыванию), количество прослушиваний (по убыванию)):

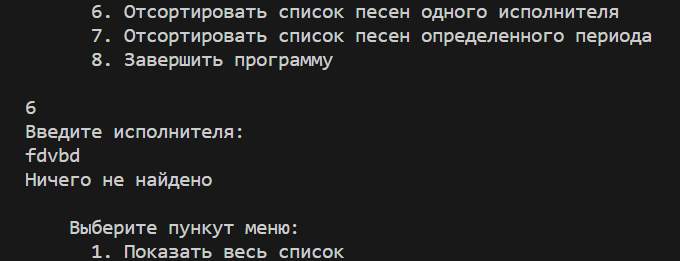
Было (Вся сортировка по всем пунктам видна на группе Queen:



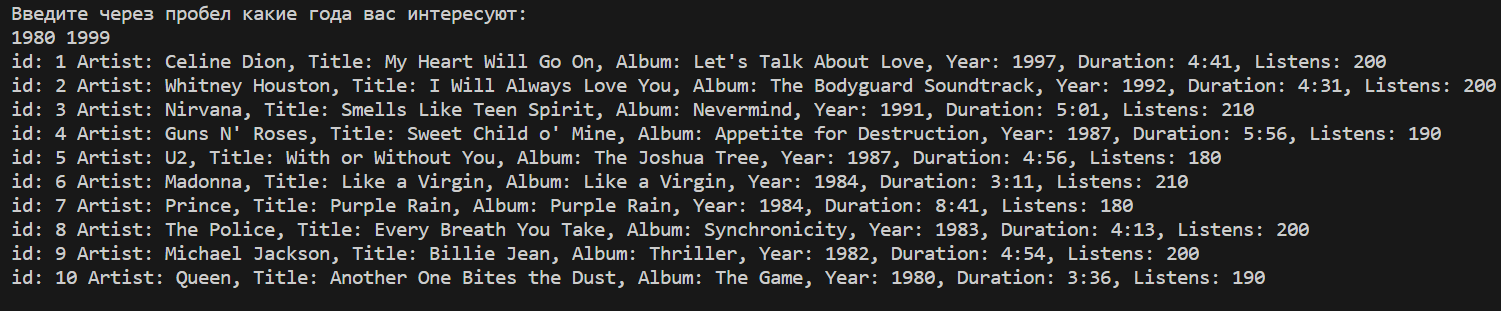
Стало:

6. Сортировка списка всех аудиозаписей конкретного исполнителя (альбом (по убыванию), название трека (по возрастанию)):

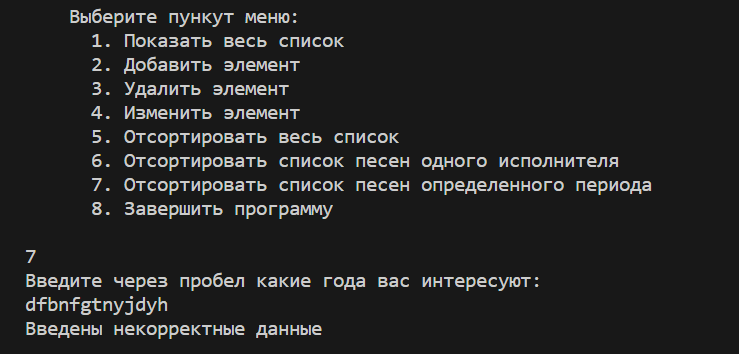
\*\*проверка некорректного ввода:

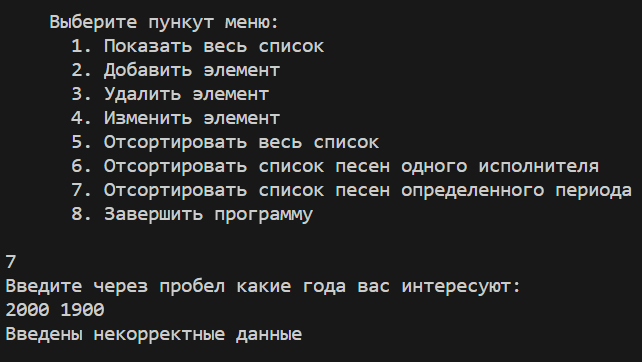


7. Список всех аудиозаписей, выпущенных в период с N1 до N2 года (год выпуска (по убыванию), исполнитель (по возрастанию):

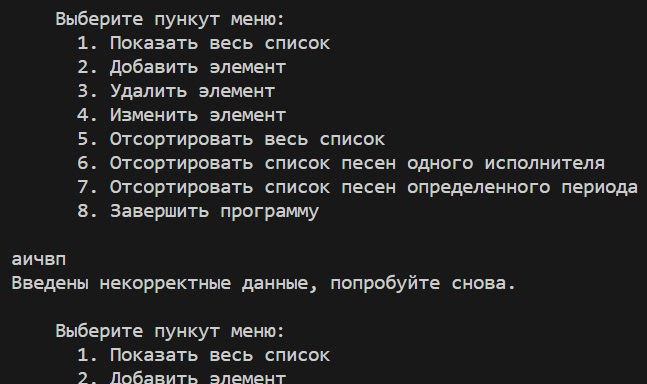


\*\*проверка некорректного ввода:





8. Введены некорректные данные в меню:



# Код программы

song\_list = [] #Хранение всего списка

class Song: #Класс для каждого элемента списка

def \_\_init\_\_(self, artist, title, album, year, duration, listens): #Инициализация элемента

self.artist = artist

self.title = title

self.album = album

self.year = year

self.duration = duration

self.listens = listens

def Menu(): #Рекурсивная функция меню

print ("""

Выберите пункут меню:

1. Показать весь список

2. Добавить элемент

3. Удалить элемент

4. Изменить элемент

5. Отсортировать весь список

6. Отсортировать список песен одного исполнителя

7. Отсортировать список песен определенного периода

8. Завершить программу

""")

try: #Обработка ошибки ввода

x = int(input()) #Считывание выбранного пункта

except:

print ("Введены некорректные данные, попробуйте снова.")

Menu()

while x!=8: #Переключатель пунктов

if x == 1:

PrintList(song\_list)

elif x == 2:

AddElement()

elif x == 3:

DeleteElement(None)

elif x == 4:

СhangeElement()

elif x == 5:

PrintList(SortAll(song\_list))

elif x == 6:

singer = input("Введите исполнителя:\n")

new\_list = []

for x in song\_list: #Создание подсписка нужный исполнителей

if x.artist == singer:

new\_list.append(x)

if len(new\_list)==0:

print("Ничего не найдено")

else:

PrintList(SortOneSinger(new\_list))

elif x == 7:

years = input("Введите через пробел какие года вас интересуют:\n").strip().split()

try:

new\_list = []

for x in song\_list: #Создание подсписка нужного периода времени

if int(years[0]) <= int(x.year) <= int(years[1]):

new\_list.append(x)

if len(new\_list)==0:

print("Введены некорректные данные")

else:

PrintList(SortForYears(new\_list))

except:

print("Введены некорректные данные")

else:

print ("Введены некорректные данные, попробуйте снова.")

Menu() #Рекурсивный вызов меню

CloseProgram() #Если ввели 8 (захотели выйти), то вызов этой функции

def CloseProgram(): #Функция закрывающая программу

wannaSave = input("Сохранить изменения базы данных? (Y/N)")

if wannaSave == "Y": #Обработка сохранения (нужно или нет)

with open('input.txt', 'w') as file:

for i in song\_list:

file.write(f'{i.artist};{i.title};{i.album};{i.year};{i.duration};{i.listens}\n')

exit("Программа завершена с сохранением") #Выход из программы

elif wannaSave == "N":

exit("Программа завершена без сохранения") #Выход из программы

else:

print ("Ввдены некорректные данные") #Если пользователь ввел чтото другое то заново в меню идем

Menu()

def Read (): #Считываем весь файл

with open('input.txt', 'r') as file:

for line in file:

artist, title, album, year, duration, listens = line.strip().split(';')

song\_list.append(Song(artist, title, album, year, duration, listens))

def PrintList (listt): #Печать переданного списка

try:

for index, item in enumerate(listt, start=1): #Используем enumerate для индексирования элементов для удобного вывода

print(f'id: {index} Artist: {item.artist}, Title: {item.title}, Album: {item.album}, Year: {item.year}, Duration: {item.duration}, Listens: {item.listens}')

except:

print("Список пуст")

def AddElement (): #Добавление элемента

song = input("""Введите новую запись в формате: Исполнитель;Название песни;Альбом;Год выпуска;Длительность;Количетсво прослушиваний

Если хотите вернуться введите 0

""")

try: #Если введены корретные данный и можно их засплитить и пихнуть то делаем это

artist, title, album, year, duration, listens = song.strip().split(';')

song\_list.append(Song(artist, title, album, int(year), duration, int(listens)))

print ("Успешно добавленно")

except: #Если не получается

try: #Попытаться понять не хочет ли пользователь просто не добавлять ничего

if int(song) == 0: return 0 #Если не хочет то просто выходим из функции

else:

print ("Ошибка ввода")

return 0

except:

print ("Ошибка ввода")

return 0

def ChooseElement(elem):#Ищет id элемента в списке

try:

if int(elem)-1>=len(song\_list): #Если не влезает в границы списка то выходим

print("Такого элемента нету!")

else: #Если влезает в границы списка то возвращаем индекс

return int(elem)-1

except:

for id, song in enumerate(song\_list, start=1):

if song.title.strip() == elem.strip(): #Если название соответсвует какомуто из списка то возвращаме его id

return id - 1

print("Такого элемента нету!")

def DeleteElement (idElem):#Удаление элемента

if idElem is None: #Проверка на пустоту id и заполнение

idElem = input("Введите id или название песни для удаления:")

try:

elem = song\_list[ChooseElement(idElem)] #Поиск элемента

if ChooseElement(idElem)>=0: #Проверка что вернули не 0

print(f'Удален элемент: \n {elem.artist};{elem.title};{elem.album};{elem.year};{elem.duration};{elem.listens}')

song\_list.pop(ChooseElement(idElem)) #Удаляем элемент

except:

print()

def СhangeElement():#Изменение элемента

idElem = input("Введите id или название песни для изменения:")

try:

elem = song\_list[ChooseElement(idElem)]

print(f'Выбранный элемент: \n {elem.artist};{elem.title};{elem.album};{elem.year};{elem.duration};{elem.listens}')

if AddElement()!=0:

song\_list.pop(ChooseElement(idElem))

except:

print()

def SortAll(new\_list):

# Если список пуст или содержит только один элемент, то он уже отсортирован

if len(new\_list) <= 1:

return new\_list

else:

# Выбираем опорный элемент из середины списка

pivot = new\_list[len(new\_list) // 2]

left, middle, right = [], [], []

# Разделяем список на три части: меньше, равные и больше опорного элемента по имени

for x in new\_list:

if x.artist < pivot.artist:

left.append(x) #Если текущий меньше опорного то добавляем его в левую часть (по возрастанию)

elif x.artist == pivot.artist: #Если имена артистов одинаковые то сравниваем по годам песни

if int(x.year) > int(pivot.year):

left.append(x) #Если текущий больше опорного то добавляем его в левую часть (по убыванию)

elif int(x.year) == int(pivot.year): #Если года песен одинаковые то сравниваем по кол прослушиваний

if int(x.listens) > int(pivot.listens):

left.append(x) #Если текущий больше опорного то добавляем его в левую часть (по убыванию)

elif int(x.listens) == int(pivot.listens):

middle.append(x) #Если текущий равен опорному то добавляем его в равную часть

else:

right.append(x) #Иначе добавляем в правую

else:

right.append(x)

else: #Если текущий больше опорного то добавляем его в правую часть

right.append(x)

# Рекурсивно сортируем подсписки и объединяем их в один отсортированный список

return SortAll(left) + middle + SortAll(right)

def SortOneSinger(new\_list):

# Если список пуст или содержит только один элемент, то он уже отсортирован

if len(new\_list) <= 1:

return new\_list

else:

# Выбираем опорный элемент из середины списка

pivot = new\_list[len(new\_list) // 2]

left, middle, right = [], [], []

# Разделяем список на три части: меньше, равные и больше опорного элемента

for x in new\_list:

if x.album > pivot.album:

left.append(x)

elif x.album == pivot.album:

if x.title < pivot.title:

left.append(x)

elif x.title == pivot.title:

middle.append(x)

else:

right.append(x)

else:

right.append(x)

# Рекурсивно сортируем подсписки и объединяем их в один отсортированный список

return SortOneSinger(left) + middle + SortOneSinger(right)

def SortForYears(new\_list):

# Если список пуст или содержит только один элемент, то он уже отсортирован

if len(new\_list) <= 1:

return new\_list

else:

# Выбираем опорный элемент из середины списка

pivot = new\_list[len(new\_list) // 2]

left, middle, right = [], [], []

# Разделяем список на три части: больше, равные и меньше опорного элемента

for x in new\_list:

if int(x.year) > int(pivot.year):

left.append(x)

elif int(x.year) == int(pivot.year):

if x.artist < pivot.artist:

left.append(x)

elif x.artist == pivot.artist:

middle.append(x)

else:

right.append(x)

else:

right.append(x)

# Рекурсивно сортируем подсписки и объединяем их в один отсортированный список

return SortForYears(left) + middle + SortForYears(right)

Read()

Menu()