Проект по „Структури от данни и програмиране“

Тема: “Музикална Колекция”

Изготвил:

Кристина Бориславова Попова, ф.н. 81933, Компютърни науки, трети курс, втори поток, шеста група

1.Увод

1.1. Описание и идея на проекта

Основната идея на проекта “Музикална колекция“ е реализирането на музикална библиотека, която поддържа информация за песни и потребители и позволява създаването и премахването(на вече съществуващи) плейлисти според вкусовете на потребителя. Музикалната система съдържа голям брой функционалности, които улесняват потребителя.

1.2. Цел и задачи на разработката

Целта на проекта е създаването на музикална колекция, която да поддържа следните функционалности, с които позволява на потребителя да:

- влезне в сестемата с потребителско име и парола;
- направи регистрация с неизползвано потребителско име и парола;
- излезне от системата;
- промени своите лични данни, като потребителско име, парола, собствено име, дата на раждане;
- добавя и премахва жанр от своите любими;
- вижда кратка информация за поддържаните от програмата команди;
- добавя песни;
- създава плейлист под дадено име;
- премахва плейлист под дадено име;
- зарежда даден плейлист по име;
- вижда информация за всички песни в тукущия плейлист;
- гласува за дадена песен от музикалната библиотека;
- сзвада плейлист с филтрирани песни преди/след/от дадена година, деден жанр или всички песни от даден жанр освен посочения, с рейтинг по-голям от определена стойност(от 0 до 5), от любимите на потребителя жанрове;

2. Преглед на предметната област

2.1. Основни дефиниции и концепции , които ще бъдат използвани

„Алгоритми+структури от данни=програми“ е заглавието на книгата на швейцарския професор по компютърни науки, автор на езиците Паскал, Модула-2, Оберон – Никлаус Вирт. Двете съставки на програмите са еднакво важни.

Структура от данни е програмна единица, която позволява да се съхранява и обработва множество от еднотипини и/или логически свързани данни чрез компютър. По-точно всяка величина определена в програма, се нарича структура от данни.

Алгоритъм е точен набор от инструкции, описващи реда на действията на изпълнителя за достигане до резултата от решението на задачата за крайно време.

Изборът на подходящи алгоритми и структури от данни е от голяма важност за създаването на качествени и ефективни програми.

Двоично дърво в информатиката се нарича дърво с разклоненост 2. При двоичното дърво всеки възел може да има не повече от двама. Всяко двоично дърво има елемент наречен *корен* (root), на който всички останали са *наследници* (или наследници на наследниците му). Обикновено с двоичното дървото се работи чрез корена му, който позволява да се достъпи всеки друг негов елемент.

Двоичното дърво за претърсване е структура от данни, която служи за съхраняване и намиране на данни по ключ, за който съществува наредба. Данните са разпределени в дървото по следния начин: за всеки връх, всички данни, които се намират в лявото му поддърво имат по-малък ключ, а всички данни, които се намират в дясното поддърво, имат по-голям ключ.

Средната сложност на операциите добавяне и търсене в двоичните дървета за претърсване е O(logN), където N е броят на елементите, добавени в структурата. Съществуват алгоритми, които поддържат структурата балансирана и запазват добрите сложности.

Стекът е линейна структура от данни в информатиката, в която обработката на информация става само от едната страна наречена връх. Стековете са базирани на принципа „последен влязъл пръв излязъл“.

В проекта е използвана ООП парадигмата, която принуждаваща програмистите да преразгледат мисленето си за програмирането, за смисъла на изпълнение на програмата и за това как информацията да бъде структурирана в компютъра. Обектно-ориентираното програмиране е съсредоточено върху обектите, които капсулират състояние и поведение. Тази дефиниция оприличава обектите на променливи

величини от абстрактни типове данни. Възможността за дефиниране на класове позволява да се създават и обработват типове данни, които липсват в езика и да се създават специфични приложения.

3.Проектиране

3.1.Обща архитектура

За проекта „Музикална колекция “ бяха реализирани класовете `User, Song, Playlist, Date, OrderedBinaryTree` и `System.` За работата с командите, беше използван поведенческият шаблон за дизайн - *command pattern,* за който бяха необходими следните класове :

- `Invoker`
- `ICommand`
- `Validator –` в който се валидира въведения вход от потребителя.
- `AddFavGenre, AddPlaylist, AddSong, ChangeBirth, ChangeName,ChangePassword, ChangeUsername, FilterCommand, HelpCommand, LoadPlaylist, LogInCommand, LogOutCommand, PrintUserPlaylist, RateCommand, RemoveFavGenre, RemovePlaylist, SignUpCommand`

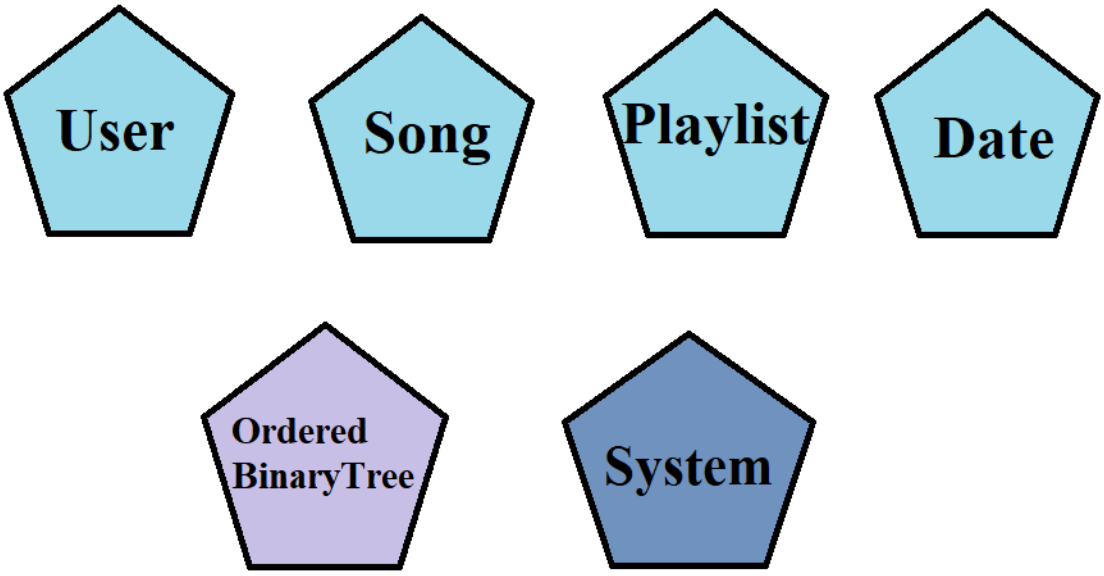
3.2. Диаграми

На фиг.1 са изобразени основните класове използвани в архитектурата на проекта “Музикална колекция”. Класът `User` се използва за представянето на потребителите в музикалната колекция. За всеки потребител се поддържа потребителско име и парола, пълно име, дата на раждане, множество любими жанрове и множество от плейлисти. Класът `Song` поддържа информация за името на песента и нейния изпълнител, жанр, албум, година на издаване и рейтинг. Класът `Playlist` представлява плейлистите, които всеки потребител създава. Всеки плейлист си има име и множество от песни, които съдържа.

В класът `System` се поддържат множество от потребители, множество от песни, които поддържа системата, текущ потребител, текущ зареден плейлист, член данна, с която помним кой потребител вече е гласувал за дадена песен и флаг, който ни казва кога в системата има влезнал потребител.

Класът `OrderedBinaryTree` използваме за сортирането на песните по азбучен ред, когато викаме функцията filter().Използването на наредено двоично дърво ни позволява при всяко добавяне на нова песен да я поставим на правилното й място в дървото. Така всяка песен, която отговаря на исканите изисквания бива добавена в двоичното наредено дърво, като мястото й в него се определя от нейното име. Принтираме дървото чрез обхождането ляво-корен-дясно, като по този начин

принтираме песните по азбучен ред. Клъсът е така реализиран, че ако вече дадена песен се намира в двоичното дърво, тя да не се добави.



фиг.1

4.Реализация и тестване

4.1. Реализация на класове

Класът `Date` използваме за датата на раждане на потребителя и датата, на която дадена песен е издадена. По-важните членфункции реализирани в този клас са:

- `bool operator==(const Date& other)const;` - която връща истина, ако две дати са едни и същи, и лъжа, когато са различни. Тази функционалност е необходима за да можем да проверяваме дали две представления се провеждат на една и съща дата.
- `int getDay()const;` - *гетър* връщащ деня.
- `int getMonth()const;` - *гетър* връщащ месеца. • `int getYear()const;` - *гетър* връщащ годината.

Класът `User` е потребителят, който използва системата. В него бяха реализирани *гетъри,* които ни връщат член данните на класа. Член функции, с които да можем да променяме(*сетъри*) член данните на класа. Останалите член функциите са:

- `void removeFavGenre(const std::string& genre);`- с която премахваме любим жанр;

- `void addFavGenre(const std::string& genre);` - с която добавяме любим жанр;
- `bool createPlaylist(Playlist* playlist);` - с която създаваме нов плейлист;
- `void removePlaylist(Playlist* playlist);` - с която премахваме даден плейлист;
- `void addSongToPlaylist(const std::string& pl, Song* song);` - с който в даден плейлист добавяме нова песен;

Класът `Song` е песента. В него бяха реализирани *гетъри,* които ни връщат член данните на класа и функцията `void rate(int n);`, която викаме, когато даден потребител гласува за дадена песен.

Клъсът `Playlist` прадставлява плейлиста, който всеки потребител създава и добавя песни в него. В него имаме два *гетъра* и член функцията `void addSong(Song* song);`, с която добавяме песни в плейлиста.

В класът `System` се прочитат всички потребители, които имат вече създадени профили в системата, зареждат се всички песни, които поддържа музикалната колекция, помни се текущият влезнал потребител, текущият отворен плейлист, ако има такъв, и дали потребителят е гласувал за определена песен. Член-функциите реализирани в този клас са всички операции, които трябва да поддържа музикалната система.

- `void signup(const std::string& username, const std::string& password, const std::string& name, int day, int month,int year);`
- `void login(const std::string& username, const std::string& password);`
- `void logout();`
- `void changeUsername(const std::string& username);`
- `void changePassword(const std::string& password);`
- `void changeDateOfBirth(int day, int month, int year);`
- `void changeFullName(const std::string& fullName);`
- `void addFavGenre(const std::string& genre);`
- `void removeFavGenre(const std::string& genre);`
- `void addSong(const std::string& playlist, const std::string& name, const std::string& artist, const std::string& genre, const std::string& album, int day, int month, int year);`
- `bool addPlaylist(const std::string& playlist);`
- `void removePlaylist(const std::string& playlist);`
- `void rateSong(const std::string& name,int rate);`

- `void filter(const std::string& input, const std::string& playlistName);`
- `void loadPlaylist(const std::string& playlist);`
- `void help()const;`
- `void printUserPlaylist();`

В private частта на класа System, освен член-данните, са дефинирани и помощни член-функции, които се викат от член-функциите в public частта. Някои от тях са:

- `void loadSongs();` - прочита песните от файла “songs.txt”
- `bool checkUser(const std::string& username, const std::string& password);` - връща истина, ако потребителят вече съществува в системата, и лъжа в противен случай
- `int checkSong(const std::string& n, const std::string& a);` - връща позицията на дадената песен, ако съществува в системата, и -1 в противен случай
- `void updateSongs(const std::string& fileName);` - обновява файла „songs.txt“ при всяко добавяне на нова песен в системата
- `void filterByRate(int rate,OrderedBinaryTree& s);` - филтрира песните над дадена стойност `rate` и ги добавя в двоично наредено дърво, за да може да ги сортира в азбучен ред
- `void filterHelper(std::string input, std::stack<std::string> &filters, std::stack<std::string> &op);` - по даден `input` извлича подадените филтри и операции. Операциите се вкарват с приоритет в стека `op`, като на върха на стека се намират операциите с най-висок приоритет.

4.2. Създаване на тестови сценарии (създаване на примери)

За проверка дали базовите класове са коректни са реализирани unit тестове, с които се проверят член-функциите на класовете `User`, `Playlist` и `Song`. Системата беше тествана ръчно върху следните тестови сценарии:

- Създаване на несъществуващ профил чрез командата signup.
- Създаване на профил с вече съществуващо потребителско име в системата, при което беше върнато подходящо съобщение
- Влзиане в системата с валидно и невалидно потребителско име и парола, при което се връща подходящо съобщение.
- Излизане от системата, при което всички направени промени се запазват успешно.

- Промяна на потребителско име, парола, име на потребителя и дата на раждане.
- Добавяне на нов любим жанр и добавяне на вече съществуващ любим жанр, при което се извежда подходящо съобщение и жанрът не се добавя.
- Премахване на любим жанр и премахване на несъществуващ любим жанр, при което се извежда подходящо съобщение.
- Зареждане на текущия плейлист.
- Добавяне на песен в даден плейлист.
- Гласуване за определена песен. Опит за повторно гласуване, при което системата не позволява на един и същи потребител да гласува два пъти за една и съща песен.
- Филтриране на песни преди/след/от дадена година, деден жанр или всички песни от даден жанр освен посочения, с рейтинг по-голям от определена стойност(от 0 до 5), от любимите на потребителя жанрове;
- Зареждане на текущ плейлист.

5.Заключение

5.1. Обобщение на изпълнението на началните цели

Проектът удовлетворява исканите функционалности. Бяха създадени, разработени и разширени (с допълнителни член-функции и член-данни) исканите класове. Създадоха се и допълнителни класове, с които да се изпълнят поставените задачи. Системата беше тествана и коригирана, когато не отговаряше на изискванията.

5.2. Насоки за бъдещо развитие и усъвършенстване

Нещата, които биха усъвършенствали проекта са:

- Оптимизиране на кода – разбиване на дългите функции на по-малки такива; премахване на повторенията на код, там където са останали такива; разбиване на големите класове на по-малки;
- Оптимизиране на сложността на част от функциите.
- Довършване на функцията filter().

Използвана литература

1. Тодорова, Магдалина, “Обектно-ориентирано програмиране на базата на езика C++”
2. Тодорова, Магдалина, “Структури от данни и програмиране на C++”
3. Glenn W. Rowe, An Introduction to Data Structures and Algorithms with Java, Prentice Hill Europe 1998