**Увод**

Целта на проекта Traveller’s App е да създаде приложение, в което потребители да споделят свойте пътувания. Създавайки нов акаунт, един потребител получава достъп до множество команди, чрез които той може да манипулира информацията, която желае да сподели. Подсигурено е, че потребителите нямат пряк достъп до множеството класове и функции, необходими за реализиране на задачата, единствено обособените команди, за които може да разбере повече чрез командата help. Всички команди се активират текстово в конзолата, въведени от клавиатура на потребителя. Цялата информация се записва във файлове, за да бъде подсигурено, че тя няма да бъде загубена. Когато потребителят стартира приложението, то автоматично ще зареди всички налични данни.

**Архитектура на проекта**

За реализацията на приложението ще бъдат използвани добрите практики на ООП - целият код организиран под формата на класове и техните член-данни & методи. Както се очаква, всеки клас е разделен в два файла, злаглавен и изходен - съответно съдържайки декларациите и реализациите им. Цялата информация се записва в .db файлове под текстови формат, с възможност в бъдеще да бъдат променени в бинарен формат, но тъй като почти всичко се случва в динамичната памет, това не би било препоръчано.

В основата на проекта стоят класовете User и Travels, събеседвани от Utilities, Date, Command, Vector (не stl-ският) и няколко наследника на Vector – Userdb & TravelDest. Всички низове, използвани в тях са запазени чрез указатели към типа char. Използването на типа string бе възможно, но предпочетох да работя без него.

**Структура на класовете**

Utilities - това е по-скоро namespace от колкото клас. В него се съдържат следните функции:

* bool isEscapeSeq(const char);
* bool isDivider(const char);
* bool isCharacter(const char);
* int strToInt(const char\*);
* bool validateName(const char\*);
* bool validatePass(const char\*);
* bool validateEmail(const char\*);
* bool validateFileImage(const char\*);
* void setString(char\* &, const char\*);

Това са помощни функции, които позволяват по-лесна обработка на данни. Името на всяка от тях интуитивно подсказва за какво служи.

Vector, и неговите два наследника Userdb & TravelDest, служат като контейнери, в които информацията да бъде запазена. Userdb създава инстанция на вектор, която работи с класът User и също така добавя метода IsEmailAvailable(), чрез който се проверява дали въведеният емайл адрес е свободен. Тъй като емайлът е уникален за всеки потребител, този метод е много полезен при регистрация на нов потребител. Освен този метод, Userdb също предоставя методи за запазване и четене на информацията във файлове, както текстови така и бинарни (въпреки, че методите за бинарните файлове не работят винаги, затова и не ги ползвам). За разлика от него, класът TravelDest не променя или добавя към функционалноста на вектора, той просто създава инстанция на него, която работи с указатели към класът Travels.

Класът Data е помощен клас, който обединява три променливи от тип int, за да създаде обект за съхранение на дати. Работи по указаният формат ISO 8601. Чрез него се узначават датите в класовете, в които има такива (именно Travels) като позволява сравнение на дати и валидиране.

Основният Travels съдържа масив от две дати, цяло число без знак за оценка както и три указателя към char, в които са съответно дестинацията, коментар към пътуването и имена на файлове - снимки от пътуването:

class Travels {

char\* destination;

char\* comment;

char\* photos;

Date timePeriod[2];

unsigned short grade;

}

Тук ми хрумна дали няма да е по-добре снимките да са запазени под формата на контейнер от char\*, вместо просто един указател, тъй като това би направило валидирането на формата им доста по-лесно, но структурата на всичко, което вече бях направил би била нарушена и затова остана като проект за бъдеще.

User - класът около който всичко се върти. Всеки потребител има 3 низа, за име, парола и емайл, отново запазени под формата на указатели към char. Освен тях, съчетава и обект от типа TravelDest, в който се пазят всички пътувания, както и вектор от указатели към User за приятели на потребителя:

class User {

char\* name;

char\* pass;

char\* email;

TravelDest travels;

Vector<User\*> frl;

}

И накрая - “двигателя” на приложението” - класът Commands. Както се предполага, в него са всички команди, подържани от приложението. Има само три член данни - низ, чрез който да се чете от конзолата; статично цяло число, съдържащо максималната дължина на буфера; и инстанция на класа Userdb, към която всички регистрирани потребители да бъдат добавени.

Най-важният метод на този клас е функцията run() - тя зарежда информацията за потребителите, ако има такава, и след това прочита вход от конзолата, с който да извика менюто от команди.

Следните команди са подържани:

* exit - Излиза от приложението
* help - Показва съобщение, което съдъжа информация за командите
* login <емайл> <парола> - Влиза в акаунт, регистриран в базата данни
* register <име> <парола> <емайл> - Добавя нов потребител към базата данни
* show - Показва всички регистрирани потребители

Допълнителна информация за имената, паролите и емайлите:

* Имената могат да съдържат само малки и големи латински букви
* Паролите трябва да бъдат под 15 символа, без ограничения за какви да са те.
* Емайлите трябва да са валидни - пр. integral@fmi-sofia.bg

След като един потребител бъде регистриран, той може да влезе в системата

чрез емайла и паролата си. Това ще му даде достъп до нови команди:

* logout - Излиза от текущия акаунт
* add\_friend <емайл> - Добавя приятел по емайл към списъка от приятели на текущия акаунт
* add\_travel - Добавя пътуване използвайки подходящи подкани за напомняне на какво се очаква от потребителя
* show\_travels - Показва всички пътувания на текущия потребител и информацията за тях

**Заключение**

Твърдя, че задачата бе изпълнена пълноценно. Потребителят има достъп само до един метод - стартиране на приложението. Целият вход на информация става чрез конзолата, и потребителят може да въвежда каквото иска през клавиатурата.

Функционалноста е както е посочена в условието (с много малки изключения), с възможността за множество разширения. От гледна точка на развитие със сигурност бих искал да се използват бинарни файлове, но това се указа по-голям проблем от колкото бях предвидил.

Линк към github - <https://github.com/asenovd/denis-OOP-pract>

(в репозиторито има и някои задачи от практикума по ООП, но проектът е обособен в отделна папка)