Traveller’s app

(Документация)

Глава 1: Увод

* 1. Описание и идея на проекта

Проектът, Travller’s app, е насочен към любителите пътешественици. Основната цел на приложението е да може потребителите лесно и бързо да споделят снимки, изживавания и впечатления от пътуванията си. От друга страна, също да могат да проверят впечатленията на останалите потребители, които са посетили дадена дестинация. Хората, използващи приложението, могат да се сприятеляват и така по-лесно и дастъпно да проверват отзивите и информаццията за дадена дестинация, която е била посетена от техен приятел.

* 1. Цели и задачи за разработка

Основната цел е разработване на програма, която поддържа база данни с потребители на приложението, които имат собствена лична база данни, и списък с всички дестинации, които са били посетени от потребителите.

За реализацията на проекта е необходимо бъде изграден хетерогенен тип данни, който да представлява потребителя в паметта на компютъра. Всеки потребител се характеризира с потребителско име, което е уникално за всеки, парола, имейл и списък с информация за всичките му пътувания, което налага изграждането на още един хетерогенен тип данни, който да служи за описание на пътуванията. В базата данни с потребителите се съхранява информацията за всеки потребител- потребителско име, парола и имейл. В личната база данни за всеки потребител се съдържа информацията за всяко едно негово пътуване. Личната база данни се зарежда (отваря), когато потребителят се впише, след правилно въведени потребителско име и парола.

За да бъде добре предсатвена информацията за пътуванията, се налага тя да бъде разделена на няколко части- дестинация, която се харектеризира с име, брой посещения, средна оценка и списък с потребители, които са я посетили, период на посещение- дата на пристигане и дата на напускане, оценка от потребителя, коментар и множество снимки. За тази цел трябва да бъдат реализирани типове данни за дестинация, дата и снимки.

Всяка дата се определя с ден, месец и година. Но за да се провери дали въведена дата е валидна е необходимо да се провери дали въведената година е високосна или не, дали определен месец е с 30, 31, 28 или 29 дни. От друга страна трябва да се проверява дали датата на пристигане е по-ранна или съвпадаща с дата на заминаване.

Всяка снимка се характеризира с име на файла, който съдържа снимката. За да бъде валидно името на файла, то трябда да е с разширение *jpeg* или *png*, съставено от малки и големи латиски букви и символа ‘\_’. Ако не е изпълнено това успловие, то името на файла е невалидно и не може да бъде създадена такава снимка.

Поради необихадимостта всички тези данни да бъдат записани във файл, който да съдържа цялата информация за всички пътувания на всеки един потребител (лична база данни на всеки потребител), то трябва всеки от гореописаните класове да има методи за записване и четене на информацията в и от бинарен файл. Личната база данни, както и базата данни, съдаржаща информация за пътуванията, са бинарни, защото тяхното разшерение е *db* и за да бъде прочетен един такъв файл е нужен специален редакатор.

За да бъде стартирана програма ще трбява да се реализира клас, който да съдържа потребителите на приложинието и всикчки дестинации. Винаги първо преди един потреител да се впише, трябва да има отворен зареден файл, чието съдържание да е или за потребителите на системата, или за дестинациите. След това има две възможности- регистрация на нов потребител или вписване на вече съществуващ такъв. След вписване на потребителя се отваря, зарежда, личната му база данни.

Глава 2: Преглед на предметната област

2.1. Основни дефиниции, концепиции и алгоритми, които ще бъдат използвани

За реализирането на проекта е необходимо да бъдат дефинирани класове, които са основен инструмент на ООП и са средство за дефиниране на абстракти типове данни. Класовете представляват синтактична конструкция, която позволава да бъдат логически групирани данни и операциите над тях. За описанието на това, какво са потребител, дата, дестинация, снимка, информация за пътуване, ще бъдат използвани класове, които ще бъдат свързани помежду си.

2.2. Дефиниране на проблеми и сложност на поставената задача. Подходи, методи за решаване на поставените проблеми.

Така поставената задача изисква всяка дестинация да съдаржа списък с потребители, които са я посетили, общата сума от оценките на всеки един потребител и броят на посещенията, за да може, когато човек иска да провери информацията за дадена дестинация, която въвежда по име, да излезе списък с потребителите, които са я посетили и тяхната средна оценка. Средната оценка ще се изчеслява като средноаритметично на сумата от всички оценки и броят на посещенията.

От друга страна, всеки потребител има спъсък с приятели. Потребитялят трябва да има възможност да проверява дали негов приятел е посетил дадена дестинация и какви са неговите отзиви. За да се знае, кой е текущо влезлият потребител в системата, след като се впише, в една променлива *posLoggedUser*  ще се запазва позицията на потребителя, който се е вписал. В класа Manager, който управлява програмата има дефинирана функция, с декларация:

void checkFriendDestination(char\* friendName, char\* destinationName);

която проверавя дали приятел с име friendName e посетил дестинация с име destinationName . В началото функция с помощта на друга фукция:

bool isUsed(char\*);

проверява дали потребител с въведеното име съществува в програмата и ако е изпълнено това условие се проверява дали той е приятел на потребителя, който е влязъл. Ако те са приятели, се проверява дали е посетене дестинацията с въведното име, ако е е изпълнено се извежда информацията, която потербетелят приятел е дал, в противен случай се извежда съатбещение, че приятелят не е посетил тази дестинация. Ако потребителят с въведеното име не съществува или не е приятел на вписания, то се извежда подходящо съобщение. По този начин се решава проблема с това потребителите на приложението да проверяват отзивите за дадена дестинация и също така, да търсят сведения кой потребител е посетил дестинацията, каква оценка е дал, както и средната оценка от всички посетители.

2.3. Потребителски и качествени изаисквания

Всеки потребител има правото да добавя нова дестинация, която е посетил, да добавя приятел, да блокира приятел, да проверява информация за дадено пътешествие, да проверява отзивите на други потребители за дадена дестинация, да вижда всички дестинации, които е посетил негов приятел, да проверява отзивите на приятел за дестниация, която той е посетил, да вижда списък от всички дестинации, които са били посетини от потребителите, да вижда целият списък от потребители в системата.

Програма трябва да работи винаги с коректни данни за това има множество от функции, които проверяват дали даден потребител съществува, дали потребител е прител на вписания такъв, дали една дата е валидна, дали една година е високосна или не, провеки за съществуването на дадена дестинация, функции, които преоразмеряват списъците с потребители, дестинации и снимки. Целта на всички тези проверки е да може програмата да работи оптимално най-добре при всякъв вход данни.

Глава 3: Проектиране

3.1. Обща арихитектура- ООП дизайн

Първоначално трябва да бъде изграден класът за потребителя ( class User). Той съдържа член-данни за потребителското име, имейла и паролата, масив от указатели към класа User (User\* m\_friendsList), който ще представлява списък с приятели, член-данни за броя и капацитета на спъсъка с приятели. Аналогично е предтавен и списъкът с пътуванията- TravelInformation\* m\_travels, и отново ще има член-данни за броя и капацитета на списъка с пътувания, и една статична константа, коята ще служи за начално инициялизиране на капацитета на списъка с приятели и списъка с пътувания. Класът User има методи:

- мутатори

- селектори

- копиране на член-данните

- изтриване на заделената памет

- конструктори с и без параметри

- конструктор за копиране

- конструктор за присвояване

- деструктор

- за заделяне на памет за списъка с приятели

- за заделяне на памет за списъка с пътувания

- записване и четене от база данните за потребителите

- записване и четене от личната база данни

- добавяне и блокиране на приятел

- преоразмеряване на списъка с приятели и списъка с пътувания

- проверка дали потребител с дадено име е приятел

- извеждане на информацията за потребителя и неговите пътувания

- добавяне на нова дестинация

Класът за снимките (class Photo) има едно член-данна за името на файла на снимката и една булева променлива, която пази дали е създадена снимка с валидно име. Класът притежава голяма четворка, мутатори, селктори, метод за записване в бинарен файл, метод за четене от бинарен файл и метод, който проверява дали името на файла със снимката е коректно.

Класът, който описва дестинация (class Destination), има член-данни за името на дестинацията, брой посещения, сума от всички оценки, дадени от потребителите, които са я посетили, списък с потребители (User\* m\_users), брой потребители и капацитет на списъка с потребители. Класът има голяма четворка, селектори, мутатори, метод за добавяне на потребител, метод за преоразмеряване на списъка с потребители, запис в бинарен файл и четене от бинарен файл.

Класът Data има член-данни за ден, месец, година. Притежава само един конструктор и има методи за извеждане на дата, записване в бинарен файл, четене от бинарен файл, проверка дали годинате е високосна, мутатори и селектори.

Класът, който описва едно пътуване се нарича TravelInformation. Той има като член-данни указател към Destination, дата на пристигане, дата на напускане, оценка от потребителя, коментар, списък със снимки, брой снимки, капацитет на снимките, булева променлива, която е със стоъност истина, ако успешно е създадено пътуването. Класът има и една статично константа, която служи за инициализиране на капацитетета от снимките. Реализирана е голяма четворка, с помощта на две помщни функции-

void copy(const TravelInformation&) и void clean(), първата копира член-данните, а другата функция изтрива заделената памет. Аналогично на останалите класове и този клас има функции за четене и писане от и в бинарен файл. Притежава метод за добавяне на нова снимка, както и метод за преоразмеряване на масива от снимки.

Класът Manager, е този който стартира програмата. Той притежава списък с потребители на приложението и списък с дестинации. Също така има и две дефинирани статични константи, една която служи за инициализирането на началниа капацитет от потребители и от дестинации, а другата служи като констната при инициализирането на низове. Класът има реализирана голяма четворка. Като конструкторът за копиране и присвояване са „забранени“, защото не се желае копиране на програмата.

Manager(const Manager&) = delete;

Manager& operator=(const Manager&) = delete;

Действието на всикчки методи на класа са описани чрез коментари в кода.

Като private член-данни класът има една променлива, която пази името на последният влязъл потребител, за да може да се създаде името на личната му база данни и да може да се отвори веднага при вписване на потребителя.

3.2. Диаграми (на структора и поведение)

Глава 4: Реализация и тестване

4.1.Реализация на класове

Основен проблем при реализиране на част от класовете беше това, че за да се създаде един клас, то той зависи от друг клас, който пък зависи от класа, който трябва да се създаде. Например, за да се създаде класа User, то той трябва да използва класа TravelInformation, който пък от своя страна трябва да използва класа User. Решението на проблема беше като преди декларацията на класа User, трабваше да се декларира, че съществува използва класа TravelInformation.

class TravelInformation;

class User {

…

}

Аналогичен беше пробелемът и при класовете TravelInformation, Destination.

class Destination;

class TravelInformation {

…

}

class User;//За да може да се създаде правилно class Destination

class Destination {

private:

…

}

В конструктора по подразбиране на класа User, не се заделя памет за списъка с приятели и списъка с пътувания, защото ще се извика консрукторът, който още не е създаден. За това на местата, които се извеква този конструктор се извикват след него функциите за заделяне на памет:

void memoryAllocFriendsList();

void memoryAllocTravelInfo();

4.2. Управление на паметта и алгоритми. Оптимизации.

Във всеки един от реализираните класове се заделя динамична памет в конструкторите. За да няма повтраяне на код, всеки от класовете притежава функции, които заделят памет и освобождават, заделената памет, когато е нужно. Такива функции например са:

//Функция за копиране

void Destination:: copy(const Destination&);

//Функция за изтриване на паметта.

void Destination::cleanMemory();

Аналогични на тези функции, притежават и остналите класове.

Класове, които притежават масиви от указатели към други обекти на класове, притежават функция, която преоразмерява този масив,когато е дастигнат капацитета му. Обикновено преорзамеряването става при добавянето на нов обект към масива. Пример за такава функция:

void Destination::resize()

{

unsigned newCapacity = m\_capacityOfUsers \* 2;

User\* newUsers = new (std::nothrow) User[newCapacity];

if (newUsers == nullptr) {

std::cout << "Not enought memory for resize! Error!" << std::endl;

return;

}

for (unsigned i = 0; i < m\_capacityOfUsers; ++i) {

newUsers[i] = m\_users[i];

}

m\_capacityOfUsers = newCapacity;

delete[] m\_users;

m\_users = newUsers;

}

void Destination::addUser(const User & newUser)

{

if (m\_numberOfUsers >= m\_capacityOfUsers) {

resize();

}

m\_users[m\_numberOfUsers] = newUser; //User\* m\_users=new(std::nothrow)

//User[m\_capacityOfUsers]

++m\_numberOfUsers;

}

При създаването на класовете TravelInformation, User, Destination, бяха необходими константи за инициализирането на капацитет или на константен низ. Ако бяха дефинирани като глабални константи за цялата програма, то е възможно тези константи да бъдат промени и да се наруши смисъла и реализацията на програмата. За това, когато са необходими константи за създаването на обект от даден клас, то в този клас константите са статични и по този начин те са „глобални“ за класа, но няма външен дастъп до тях и не могат да бъдат променени.

4.3. Планиране, описание и създаване на тестови сценарии.

По време на създаване на класовете в програмата бяха създадени тестове, които да проверят дали правилно се създават обекти от различните класове, дали методите им работят правилно, дали е направена пълна валидация на данните, които постъпват преди да се създаде обект от даден клас. Всички тестове се намаират в заглавен файл с име “Tests.hpp”.

При тесването на класа User са тествани, конструкторите, методите за добавяне на приятел, за премахване на приятел, за извеждане на информацията, за добавяне на дестинация, за извеждане на инфоламицията за пътуванията на потребителя.

void testUser() {

User u1("Ivan", "123", "ivan@abv.bg");

User u2;

u2.memoryAllocFriendsList();//Защото в конструктора по подразбиране не се заделя памет за приятелите и пътуванията.

u2.memoryAllocTravelInfo();

u2 = u1;//Тест конструктор за присвояване.

u2.printInformation();

User user1("User1", "123", "user1mail");

User user2("User2", "123", "user2mail");

User user3("User3", "123", "user3mail");

//Различни тестове за добавяне на приятел.

user1.addFriend(user1);

user1.addFriend(user2);

user1.addFriend(user3);

user1.removeFriend(user2);

std::cout << user1.isFriend("User1") << std::endl;

std::cout << user1.isFriend("User2") << std::endl;

std::cout << user1.isFriend("User3") << std::endl;

user1.printInformation();

user2.printInformation();

user3.printInformation();

Destination\* dest = new Destination("Rome");

Date a(1, 3, 2020);

Date b(10, 3, 2020);

unsigned eval = 4;

unsigned const m = 5;

char comment[m];

std::cin.getline(comment, m);

TravelInformation t1(dest, a, b, eval, comment);

user1.addNewTravel(t1);

user1.printTravelInformation();

user1.printDestinationInfoByName("Rome");

user1.printDestinationInfoByName("Ibisa");

user1.prinDestinationEvaluation("Rome");

}

За класа Date трябва също да се тестват конструктора и методите му за писане и четене от бинарен файл, както и дали валидацията работи успешно. Когато потребител въведе ден, който е извън рамките на месецът, който е избрал, програмата конвертира дата до последния ден от месеца, който е посочил потребителя. Ако той не е доволен от това конвертиране може да промени датата.

void testDate() {

std::ofstream ofs("date.bin", std::ios::out | std::ios::binary);

Date date1(3, 2, 2020);//Тестване на конструкторите

Date date2(1, 3, 2020);

Date invalidDate(29, 2, 2018);//Съобщение за невалидна дата.

std::cout << "date1: ";

date1.printDate();//Извеждане на дата

date1.serialize(ofs);//Сериализиране.

ofs.close();

std::cout << std::endl << "date2: ";

date2.printDate();

Date date3;

std::ifstream ifs("date.bin", std::ios::in | std::ios::binary);

date3.deserialize(ifs);//Десериализиране.

std::cout << "Date 3: ";

date3.printDate();

ifs.close();

Date d1(29, 2, 2019);

std::cout << d1.getDay() << "/" << d1.getMonth() << "/" << d1.getYear() << std::endl;

}

Важно Destination е да се при тестването на класа проверят дали конструкторът работи правилно, изчислява ли се правилно средната оценка и дали се извежда като дробно число, а не като цяло. Отново както при другите класове трябва да се тестват и функциите за записване и четене от файл.

void testDestination() {

Destination d("Sofia", 7, 25);

//Извеждане на общата оценка за дестинацията.

std::cout << "Average evaluation of destination is: " << std::setprecision(2) << d.getAverage() << std::endl;

std::ofstream dest("destination.bin", std::ios::out | std::ios::binary);

d.serialize(dest);

dest.close();

std::ifstream des("destination.bin", std::ios::in | std::ios::binary);

Destination d2;

d2.deserialize(des);

std::cout << d2.getDestination() << ", " << d2.getNumberOfVisits() << std::endl;

des.close();

}

Подобно на останалите класове и класът TravelInformation е тестван.

void testTravelInfo() {

Date d1(1, 7, 2019);

Date d2(7, 7, 2019);

Destination\*destination = new Destination("Nesebar", 100, 321);

unsigned eval = 3;;

char comment[100];

std::cout << "enter a coment: ";

std::cin.getline(comment, 100);

Photo\* p1 = new(std::nothrow) Photo[1];

p1->setPhoto("myPhoto.jpeg");

TravelInformation t1(destination, d1, d2, eval, comment, p1, 1, 2);

destination->setNumberOfVisits(destination->getNumberOfVisits() + 1);

destination->setSumOfAll(destination->getSumOfALL() + eval);

std::cout << "Print info: " << std::endl;

t1.print();

std::cout << "Destination: " << destination->getAverage() << ", " << destination->getNumberOfVisits() << ", " << destination->getSumOfALL() << std::endl;

std::cout << std::endl;

//Тестване дали правилно е добавено ново пътуване.

User u1("Ivan", "123", "ivan@abv.bg");

std::cout << "User1: " << u1.getNumberOfFriends() << "," << u1.getCapacityOfFriends()

<< '/' << u1.getNumberOfTravels();

u1.addNewTravel(t1); //Успешно преминат

std::cout << "Add new travel for u1: " << std::endl;

u1.printTravelInformation();

std::ofstream ofs("travelInfo.bin", std::ios::out | std::ios::binary);

//t1.serialize(ofs);

ofs.close();

std::ifstream ifs("travelInfo.bin", std::ios::in | std::ios::binary);

TravelInformation t2;

t2.deserialize(ifs);

std::cout << t2.getComment() << std::endl;

ifs.close();

}

При класа Photo също трябваше да се тества дали конструкторите работят и методите за писане и четене от бинарен файл.

void testPhoto() {

Photo p("myFristPhoto.jpeg");

std::ofstream ofs("photos.bin", std::ios::out | std::ios::binary);

p.serialize(ofs);

ofs.close();

Photo p1;

std::ifstream ifs("photos.bin", std::ios::in | std::ios::binary);

p1.deserialize(ifs);

std::cout << "Photo name is: " << p1.getPhoto() << std::endl;

Photo p2("img.jpeg");

std::cout << p2.getPhoto() << std::endl;

Photo p3 = p2;

std::cout << p3.getPhoto() << std::endl;

Photo p4;

p4 = p1;

std::cout << p4.getPhoto() << std::endl;

Photo p5("error.txt");

std::cout << p5.getPhoto() << std::endl;

}

Методите на класа Manager също са тествани.

void testMenagerClass() {

Manager m;

m.registrerNewUser();

m.registrerNewUser();

m.login();

Destination\* dest = new Destination("Melnik");

Date a(1, 3, 2020);

Date b(2, 3, 2020);

unsigned eval = 4;

char comment[3];

std::cin.getline(comment, 3);

TravelInformation t1(dest, a, b, eval, comment);

m.addDestination(t1);

m.printUser();

m.printDestinations();

}

Глава 5: Заключение

5.1. Обобщение на изпълнението на началните цели

Началните цели на проекта са изпълнени, използвайки знанията получени от курса по ООП.

5.2. Насоки за бъдещо развитие и усъвършенстване

В бъдеще проектът може да бъде усъвършенстван. В функцията void Manager::addDestination(const TravelInformation & newTravel), кагото се създава нова дестинация, да се добавя потребителят, който я е посетил, и така функцията void checkDestination(char\*) на същия клас да извежда спъсъка с потребители, които са я посетили. Има възможност, част от някой функции да се оптимизират, така че да работят еднакво за различни типове(може би финкцията resize и тази, която заделя памет). Също така при регистриране на нов потребител може да се направи валидация, дали паролата е лесна, дали има вече потребител със същият имейл и така потребителят, който иска да се регистрира да промени данните си. Възможно е при въвеждането на парола на екрана да се изписва някакъв символ, за да остане паролата скрита.

Линк към Github хранилище: <https://github.com/TerezaTrenadafilova/Traveller-s-app>