Стефани Арабаджиева
ф.н. 81888
30.05.2020 г.
https://github.com/sarabadjieva/Hotel-Managment

Проект „Хотел”

курс Обектно-ориентирано програмиране
за специалности Информатика, Информационни системи и Компютърни науки
летен семестър 2019/2020 г.

1. Увод

Проектът представлява разработка на функционираща информационна система за управление на хотел. За изпълнението на проекта се използва *Visual Studio* и езикът за програмиране C++, контейнери от стандартната библиотека с шаблони, библиотека за работа с файлове и за работа с потоци. Данните се запазват в текстов файл. Датите се изписват в международния стандарт за обмен на дати и данни за време *ISO 8601*.

Целта на проекта е да се реализира система, позволяваща улеснено управление на хотел чрез различни функции. Поддържат се различни операции за резервиране на стая, за проверяване дали е заета, за намиране на свободна и други.

Задачата на проекта е да предостави на потребителя различни възможности за работа с файлове, т.е. да може да се избира с кои файлове да се работи, дали новите данни да бъдат изтрити(close), запазени в същия файл(save) или друг(save as), чието име е въведено от потребителя. Другата част от задачата е проектиране на функции за работа с данните, намиращи се във файла. Те представляват: добавяне на резервация или на период от време, в който една дадена стая не може да бъде наета; проверяване дали дадена стая е свободна; освобождаване на заета стая; намиране на свободна стая; извеждане на справки.

В документацията ще бъде направен преглед на предметната област (основните дефиниции, концепции и алгоритми, които са използвани; проблеми и сложност на поставената задача и методи за решаването им). Следващото нещо, което ще се разгледа, е проектирането на задачата, където са включени основните класове и връзката между тях. Приложена е таблица с името на класовете, съответните им член-данни и по-важните методи. После се преминава към реализацията и тестването на проекта, приложени са части код, обяснени са в дълбочина по-важните функции и работата на програмата. Създадени са и няколко примера, показващи некоректно въвеждане на информацията. В заключителната част са дадени идеи за бъдещо развитие на проекта и подобрението на потребителския интерфейс.

2. Преглед на предметната област

Подходът за реализиране на проекта е чрез обектно-ориентирано програмиране. Различните части от задачата се представят като обекти, чиито класове включват в себе

си данни и методи(конструктори, селектори, мутатори и др.). Използвани са основните принципи абстракция, капсулация, както и рекурсия. Всички класове са дефинирани глобално. В програмата са включени алгоритми за търсене, рекурсия, динамично програмиране.

Един от проблемите е създаването на четим файл и въвеждането на информация в него. За да е разбираем за човешкото око това няма как да се осъществи чрез използването на двоичен файл. В класа *Room* и *Date* са предефинирани операторите за вход и изход, които позволяват добре структуриран текстов файл, в който ясно са въведени данните.

Различните възможности за запазване на файл са програмирани чрез създаване на временен файл, в който се копира първоначалното съдържание. В края на програмата този файл се изтрива, а информацията в него може да се презапише в друг файл (оригиналния или нов) или да не се запазва.

Системата е удобна и скалируема, защото позволява да се работи с неограничен брой стаи и резервации. Всеки потребител може да работи със системата без значение неговите права и роля. Той трябва да отговаря на различни въпроси или пише операции за промяна на данните, като въвежда всичко в терминала от клавиатурата. Проектът изисква обектно-ориентиран стил, за да може да се чете по-лесно и да могат да се правят промени бързо.

3. Проектиране

В проекта се съдържат три основни класа: *Manager, Room, Date.* Класът *Manager* съдържа контейнер с екземпляри от класа *Room*. Всяка стая съдържа брой легла; свой номер, прочетен от предварително зададените данни; число, използвано за разграничаване на различните периоди, в които стаята не е свободна. Член-данни са и три асоциативни контейнера, описващи различните видове заетост на стая. Те описват период от дати, брой гости и бележки. Работят по следния начин: ключа на стаята служи за ключ на всеки от тези асоциативни контейнера. Него може да си го представим като брояч. Случаите, в които една стая е заета са два: или е направена резервация, или е обявена за недостъпна. И при двата се записват бележки, но само при резервацията има брой гости. И в двата случая се записва желания интервал от дати в контейнера *reservations* и се добавя бележката в *notes,* където ключ и на двете стойности е член-данната *key.* Ако искаме да добавим резервация, в контейнера *guests* слагаме желания брой гости с ключ – текущата стойност на променливата в класа. След всички тези операции добавяме към *key* 1. Така когато искаме да разберем дали стая е обявена на едни дати за резервирана или недостъпна, е достатъчно да проверим ключа на съответните дати дали го има в контейнера за гости. Ако го няма, е обявена за недостъпна.

В следващата таблица са представени накратко всички класове в проекта - техните член-данни и по-важни функции. Кратко обяснение се съдържа в полето, отдясно на тях. Стрелките посочват пътя на създаване на програмата, като най-отдолу е класът *Date,* който помага за създаването на по-нататъшни операции. Над него е класът за стаи, който е ключов за цялата програма. Мениджър класа е частта, до която „достига” обикновеният потребител и работи с вход и изход на данни.

Manager
file: fstream temp: fstream filename: char array tempName: char array events: vector<Room*>
callManager(): void readRooms(): void writeRooms(): void workwithfile(): void

Класът *Manager* служи за контролиране на всички процеси и управление на потребителския вход. В него се запазват множеството от стаи, с които се работи. Те биват прочетени от файл чрез функцията *readRooms()*. При всяка промяна на данните се изтрива информацията от началния файл и се копира тази, запазена във временния файл. Той ”живее” само по време на работа с програмата. При нейното спиране автоматично се изтрива.

Room
beds: short unsigned roomNumber: unsigned integer key: integer reservations: map<int, vector<Date> > guests: map<int, short unsigned> notes: map<int, const char*>
addReservation(const Date&, const Date&, const char*, integer): bool checkout(const Date& d): bool unavailable(const Date&, const Date&, const char*): void getReservations(const Date&, const Date&): vector<stringstream*> report(const Date&, const Date&): int

Класът *Room* представя стаите и в него се намира основната информация за проекта. Съдържа се всичко необходимо, създадено чрез три асоциативни вектора и едно число (ключ), които ще бъдат разгледани по-надолу в документацията. Кодирани са функции, които позволяват управлението на заетостта на дадена стая в даден за някакъв период от време.

Date
year: unsigned integer month: unsigned integer date: unsigned integer validDate: bool
daysInMonth(unsigned integer, unsigned integer): unsigned integer nextDate(): Date operator==(Date): bool operator<(const Date&): bool operator>(Date): bool getDatesBetween(Date, Date): vector<Date>

Класът *Date* съдържа основните операции, които могат да се извършват с дати и биха били полезни. Член-данна е булева, която съдържа информация относно валидността на въведената дата, т.е. дали месецът е между 0 и 12, дали денят отговаря на истинска дата. В случай, когато датата е създадена без да се подават параметри, новосъздадената дата е днешната.

4. Реализация, тестване

При стартиране на програмата потребителят трябва да избере дали иска да работи с файла по подразбиране или с друг, създаден вече. Програмата очаква вход от потребителя и след получаването на такъв - информацията се обработва. Според въведените данни се избира кой конструктор на класа *Manager* да се използва. След това се влиза в цикъл, позволяващ на потребителя да избере по какъв начин иска да работи с файла.

```
void Manager::callManager()
{
	std::cout << "Choose what you want to do:\nopen, close, save, saveas, help,
exit.\n";
	char choice[20];
	std::cin >> choice;
	std::cin.clear();
	std::cin.ignore();
	if (strcmp(choice, "open") == 0)
	{
		open();
		workwithfile();
	}
	else if (strcmp(choice, "close") == 0)
	{
		close();
	}
	else if (strcmp(choice, "save") == 0)
	{
		save();
		callManager();
	}
	else if (strcmp(choice, "saveas") == 0)
	{
		saveas();
		callManager();
	}
	else if (strcmp(choice, "help") == 0)
	{
		help();
		callManager();
	}
	else if (strcmp(choice, "exit") == 0)
	{
		exitFunction();
	}
	else callManager();
}
```

При избор на *open* се отваря избраният файл за работа с билети, като се прави негово копие във временен текстов файл (*"temp.txt"*). Ако при отварянето на някой файл се появи грешка, програмата принтира съобщение и извиква функцията *callManager()*. Ако няма проблеми с файловете се извиква се функция *workwithfile()*, с чиято помощ се извършват операции върху данните, прочетени от файла. При потребителски вход *close* програмата затваря файла с данните, без да запазва създадените промени по време на процеса на работа. *Save* запазва направените промени в същия файл, от който са били прочетени данните, а *saveas* ги записва в файл, чието име трябва да се укаже от потребителя. Информацията се извежда с помощта на *help*, а *exit* излиза от програмата.

При влизане във функцията *workwithfile()*, потребителят трябва да въведе желаната функция за работа с данните. *Workwithfile()* работи на същия принцип като *callManager()* – очаква се вход от потребителя, който се конвертира в поток; програмата брои въведените думи и сравнява първата от тях с възможните операции. В края на функцията програмата пита потребителя дали иска да продължи да работи с данните от файла, ако не отново се вика *callManager()*. Приложен е част от кода, показващ главната функционалност на *workwithfile()*, като са пропуснати различните *if* проверки, които четат първата дума от потребителския вход.

```
void Manager::workwithfile()
{
	std::cout << "\nPlease, enter function with parameters:";
	char function[1000];
	std::cin.getline(function, 1000);
	std::stringstream ss(function);

	char word[MAX_NOTE_LENGTH];
	int wordCount = 0;
	while (ss >> word)
	{
		wordCount++;
	}
	if (wordCount == 0)
	{
		std::cerr << "Please write something!\n";
		workwithfile();
	}

	ss.clear();
	ss.seekg(0, std::ios::beg);
	char functionName[20];
	ss.getline(functionName, 20, DELIMITER);

	if (strcmp(functionName, "checkin") == 0)
	{
		unsigned rn;
		Date d1, d2;
		char note[MAX_NOTE_LENGTH];

		if (wordCount < 5)
		{
			std::cout << "Wrong parameters!\n";
			workwithfile();
		}

		ss >> rn >> d1 >> d2;
		ss.get();

		char digits[11] = "0123456789";
		char c = ss.str().back();
		if (strchr(digits,c) != NULL)
		{
			ss.getline(note, MAX_NOTE_LENGTH, DELIMITER);
			wordCount -= 5;

			char temp[MAX_NOTE_LENGTH];
			while (wordCount != 1)
			{
				ss.getline(temp, MAX_NOTE_LENGTH - strlen(note), DELIMITER
				strncat_s(note, &DELIMITER, 1);
				strncat_s(note, temp, strlen(temp));
				wordCount--;
			}

			unsigned guests;
			ss >> guests;
			checkin(rn, d1, d2, note, guests);
		}
		else
		{
			ss.getline(note, MAX_NOTE_LENGTH);
			checkin(rn, d1, d2, note);
		}

		writeRooms();
	}

...
```

```
else
	{
		std::cout << "Wrong input!\n\n";
	}

	std::cout << "\nDo you want to work with the same file more(y/n)\n";
	char answer;
	std::cin >> answer;
	std::cin.clear();
	std::cin.ignore();
	if (answer == 'y')
	{
		workwithfile();
	}
	else callManager();
}
```

Работи по следния начин: след прочитане на първата дума, а именно името на функцията, програмата отчита валидността на следващите подадени параметри, като се проверява дали е изпълнено условието за брой на думи (например на *checkin* трябва да се подадат най-малко още четири думи освен името на функцията, за да има достатъчен брой данни, с които да се работи). Четат се параметрите от останалата част от потока, който вече е с една дума по-малко (първата, защото е взета за проверка), и се вика съответната операция, която се изисква.

При избор на *checkin* първо се чете номер на стаята и след него две дати. Прави се проверка дали последната дума от реда е число, за да може да се извлекат правилно параметрите. Единият от тях е бележка, която може да бъде с повече от една дума, а другият, който би могъл да бъде пропуснат, са броят гости. След разрешаване на този проблем се вика *checkin(unsigned room number, const Date&, const Date&, const char* note, int guests),* която първоначално проверява дали датите са правилни, бъдещи и една след друга; гледа се и дали гостите са положително число над нула (в случай че стойността на този параметър е -1, се приема, че не е подаден конкретен брой гости, и се заемат всички легла). Ако всички тези са изпълнени, програмата продължава напред и търси съответствие в контейнера от стаи и търси подадения номер на стая. Ако бъде намерено, се извиква булевата функция *addReservation(…)* от класа *Room.* В зависимост от нейната стойност се извежда съобщение за успешно/неуспешно извършена операция.

availability(const Date& d = Date()) намира свободните стаи за дадена дата. Проверява дали датата е правилно изписана, като няма значение дали датата е изминала, днешната или бъдеща. Влиза се в цикъл, който за всяка стая извиква булевата функцията *getAvailabilityAt(const Date&)* от *Room.* Ако нейната стойност е *true,* на екрана се принтира съобщение с номера на стаята.

checkout(unsigned rn) освобождава заета стая, като преминава през цикъл, търсещ стаята с подадения номер. При намирането ѝ се извиква булевата функция *checkout()* от класа за стая. Тя връща истина, когато освобождаването е успешно. В този случай се

принтира съобщение на екрана, уведомяващо потребителя за това. Ако функцията върне лъжа или стаята не бъде намерена – програмата извежда съобщение за грешка.

report(const Date&, const Date&) извежда се списък, в който за всяка стая, използвана в дадения период, се извежда и броя на дните, в които е била използвана. Проверява се дали въведените дати са валидни и не са изминали, след което се влиза в цикъл, който проверява за какъв брой дена е била заета стаята. Извеждат се стаите, които са използвани над 0 дена.

bool find(short unsigned b, const Date&, const Date&) булева функция, намираща свободна стая по подадените параметри. След проверките за дати се влиза в цикъл, който проверява за всяка стая с достатъчен брой легла дали е свободна на всяка една дата в подадения период. Когато отговаря на критериите, тази стая се запазва във вектор, който се използва за четливо принтиране на стаите, които са подходящи.

find_(short unsigned g, const Date&, const Date&) функцията намира стая за специален гост, като може да се извършат до две премествания от тази в стая в друга. Функцията прави проверки за датите, намира всички стаи с достатъчен брой легла и ги слага в контейнер, след което влиза в цикъл и проверява заетостта на всяка за дадения период. Резервациите в дадена стая за определен период от време се намират чрез функцията *getReservations(const Date&, const Date&),* която връща контейнер от потоци, които съдържат информация за периода, в който стаята е била заета, и броя гости. Главната функция продължава като изключва стаи с над две резервации в този период и такива, записани за недостъпни за някоя от желаните дати. За останалите стаи и техните резервации се извиква функцията find(...), която връща истина, ако хората от желаната стая могат да бъдат преместени в друга. Когато не може – стаята отново се изключва. Накрая остават само тези, които могат да бъдат резервирани. Те се принтират и потребителят трябва да избере дали иска да запази някоя от тях.

unavailable(unsigned rn, const Date&, const char)* обявява една стая за недостъпна, като първо проверява датите, след което влиза в цикъл, докато намери номера на желаната стая. Ако търсенето бъде успешно, за стаята се извиква функцията *unavailable(const Date&, const Date&, const char* note)* от *Room.* Извежда се съобщение за несъществуваща стая, ако не бъде намерена.

Ще бъдат разгледани накратко споменатите функции от други класове.

bool Room::addReservation(const Date&, const Date&, const char, int guests)* прави проверка за брой легла, след което се създава вектор от датите за желания период с функцията *getDatesBetween(...),* която връща празен контейнер при грешно въведени данни. Ако функцията е върнала вектор с големина 0 или 1, не е възможно да се направи резервация и функцията връща *false.* Проверява се дали стаята е свободна на всяка една дата и добавя резервация, ако не е заета. В противен случай връща грешка. Ако не е указан броят на гостите, се заемат всички легла.

bool Room::checkout(const Date& d) по подразбиране работи с текущата дата. Проверява се дали стаята, която трябва да се освободи, е заета. Ако не е, се връща грешка. В противен случай се влиза в цикъл, който проверява всички резервации от

контейнера. Следват три проверки: 1) ако датата е между началната и крайната дата на резервация, 2)ако съвпада с началната, 3) ако съвпада с последната. В първия случай се махат последните дати от резервацията, докато не стигнем до желаната дата за освобождаване. Във вторият случай се изтрива цялата резервация. В третия не се променя нищо. Във всеки един от трите случая се проверява първо дали тези дати са обявени за недостъпни на дадена стая. Ако да, се връща лъжа, иначе - истина.

Room::unavailable(const Date&, const Date&, const char)* обявява една стая за недостъпна в даден период от време. Ако има резервации в този период, то те се освобождават и се принтира съобщение, което заявява, на коя дата е трябвало да се освободи стаята.

vector<stringstream> Room::getReservations(const Date&, const Date&)* създава вектор от потоци. След което влиза в цикъл, който проверява за всяка една стая дали е била заета в дадения период. Правят се различни проверки за датите и, ако бъде намерена стая, заета в този период, то за нея се проверява дали е резервирана или обявена за недостъпна. Ако е недостъпна, във векторът се добавя поток „*unavailable*", ако е резервирана „<дата1> <дата2> <брой_гости>". Връща се създаденият вектор.

int Room::report(const Date&, const Date&) проверява дните в даден интервал от време дали са били заети. Първо е създаден брояч с начална стойност 0. При всяко намиране на заета дата към този брояч се добавя 1. Връща се неговата стойност.

Всеки клас съдържа в себе си конструктор и деструктор, а всички без мениджър класа съдържат и конструктор за копиране и присвояване. В класовете са дефинирани и селектори и мутатори. Деструкторите позволяват освобождаване на заетата памет по време на работенето с програмата. Създадени са глобални променливи, представители на типовете *char* и *int,* за по-лесно програмиране и четене на файла от други хора.

Тестовете, които трябва да се извършат, за да се провери коректността на програмата са различни. Един от тях е да се въведат по-малък на брой думи, отколкото функцията изисква и следователно липсват данни, с които да се работи. Валидността на датата също е от съществено значение за обработване на информацията. Въвеждане на данните в грешен ред. Примерите по-долу показват грешно въведени данни от потребителя.

$find 2 2020-13-02 2020-12-01 – грешно въведени данни (неправилна дата; първата дата е след втората)

$find 2020-02-02 2020-03-02 (2) – неправилна подредба или липса на информация

Датите се изписват в международния стандарт за обмен на дати и данни за време *ISO 8601.* Ако потребителят се опита да ги въведе в друг формат, програмата ще отчете датата за невалидна.

5. Заключение

При работа с програмата потребителят има възможност да изпълнява всички функционалности при въвеждане на валидни данни. Програмата може лесно да бъде модифицирана с цел скалируемост. Освен че може да работи с неограничен брой стаи и резервации, лесно могат да бъдат добавени данни за цена на нощувка, хабене на вода, ток и т.н. По този начин биха могли да се наблюдават приходите от всяка стая и хотел. Възможност за намалена цена за деца също би била удобна, защото в мнозинството от хотели е съществува такова правило. Създаване на обект „мини хладилник”, който съдържа в себе си различни продукти, които могат да се взимат от гостите спрямо дадена цена.

За улеснение на потребителя е ключово и създаването на графичен потребителски интерфейс, в който командите не трябва да се пишат директно от потребителя, а ще са предоставени елементи за управление във вид на графични изображения.