 *Софийски университет „Св. Климент Охридски“  
Факултет по математика и информатика*

КУРСОВ ПРОЕКТ ПО ОБЕКТНО-ОРИЕНТИРАНО ПРОГРАМИРАНЕ

летен семестър 2019/2020г.

Проект 11 - Хотел

Изготвил: Атанас Миленов Иванов

Специалност: Информационни системи I курс

Група: 3

Юни 2020

гр.София

**Глава 1. Увод**

1.1. Описание и идея на проекта……………………………………………..…………..

1.2. Цел и задачи на разработката ……………………………………………………….

**Глава 2. Преглед на предметната област**

2.1. Основни дефиниции, концепции и алгоритми…………………………………….

2.2. Дефиниране на проблеми и сложност на поставената задача……………………..

2.3. Подходи, методи (евентуално модели и стандарти) за решаване на поставените проблемите ………………………………………………………………………………...

**Глава 3. Проектиране**

3.1. Обща архитектура – ООП дизайн……………………………………………………..

3.2. Диаграми (на структура и поведение - по обекти, слоеве с най-важните извадки от кода)…………………………………………………………………………………………..

**Глава 4. Реализация, тестване**

4.1. Реализация на класове ----------------------------------------------------------------------

4.2. Управление на паметта и алгоритми. Оптимизации.------------------------------------

4.3. Планиране, описание и създаване на тестови сценарии --------------------------------

**Глава 5. Заключение**

5.1. Обобщение на изпълнението на началните цели-----------------------------------------

5.2. Насоки за бъдещо развитие и усъвършенстване-------------------------------------------

**Глава 6. Използвана литература**

За създаването на диаграмите:

<https://app.diagrams.net/>

https://www.stackoverflow.com/

**Важно**: НЕ са използвани std::vector и std::string, а са употребени ръчно написани.

1. Увод
   1. **Описание и идея на проекта**

Представям ви информационната система Хотел. Идеята на проекта е да обслужва хотел, като съхранява и обработва данни за стаи, настанявания, гости и други. Потребителят има пълна информация за всяка отделна стая, колко гости са настанени и колко легла има и чрез използването на прости команди и операции може да модифицира данните в нея. Предимставата на системата са, че се грижи за управлението на файловете, тоест всяка промяна по хотела може да бъде запазена в текстов файл, до който потребителят има достъп и е добре структурирана, така че всяка операция да има точно определена задача.

* 1. **Цел и задачи на разработката**

Във времената на разрастващите се софтуерните технологии, много компании и различни предприятия се нуждаят от система, която им улеснява работата, така че да бъде по-ефективна и по-оптимизирана. Главаната цел на настоящия проект е точно такава – предоставяне на програмата, която улеснява задачите при работата с клиенти от страна на хотела. Без нужда от познания по разработката на софтуер администрацията може да управлява своя хотел по бърз и ефикасен начин.

Основните задачи, които изпълнява информационната система са:

* Зареждане на всички стаи в хотела (константен брой)
* Регистриране в стая с номер за даден период от време, както и броя на гости, които ще бъдат настанени (**checkin**)
* Извеждане на всички свободни стаи на определена дата (**availability**)
* Освобождаване на заета стая с номер (**checkout**)
* Справка за използването на стаи в зададен период (**report**)
* Намиране на подходяща стая със зададен брой легла или повече в даден период. При наличие на повече свободни стаи се предпочита тази с по-малък брой легла. (**find**)
* Алгоритъм за спешно намиране на на стая за важен гост в случай на липса на совбодни стаи за даден период. Алгоритъмът служи за разместване на настанените от най-много две стаи (**find!**)
* Обявяване на стая за временно недостъпна за даден период от време и се добавя бележка (**unavailable**)

Основните компоненти на системата са Hotel, Date, Period и Room.

Хотелът съдържа:

* Име (name)
* Стаи (Vector<Room> rooms)
* Брой на стаи (const numOfRooms)

Периодът съдържа:

* Двойка дати „от-до“ (Date from, Date to)
* Всички дати в периода (Vector<Date> datesInPeriod)

Стаята съдържа:

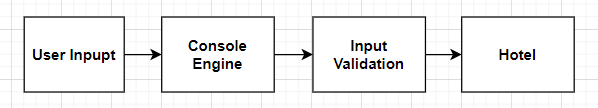
* Номер (roomNum)
* Гости (roomGuests)
* Дали е свободна (free)
* Легла (beds)
* Заети дати (scheduledDates)

Датата е стандартна във формат YY-DD-MM (2020-02-06).

2. Преглед на предметната област

2.1. **Oсновни дефиниции, концепции и алгоритми**

Основният компонент който задвижва програмата е ConsoleEngine, който позволява нашата система да работи като конзола, която изпълнява операции.



Фиг.1 Как програмата обработва входа на потребителя

При зареждане на приложението, се отваря конзола, която очаква вход от потребителя или по-скоро валидна операция (една от изброените в глава 1.2). Нека вземем за пример операцията **availability**. Потребителят въвежда “availability 2020-15-10” (на един ред) и както в една обикновена конзола, тя разбира че иска да се изпълни операцията availability с подаден параметър датата 2020-15-10. На фиг. 1 се вижда нагледно как става процесът. Първоначално входа на потребителя представлява един символен низ, който чрез помощта на функциите в ConsoleЕngine бива обработен по подходящ начин. Става дума за следните функции:

* **countLetters**
* **indexOf**
* **substring**
* **split**
* **toInt**

Използвайки ги, въведеният символен низ се разделя съответно на отделни думи, които първо биват валидирани и след това извикват правилните функции от хотела. Подробното им действие е описано в следващата глава.

2.2. **Основни проблеми, решения и определяне на сложност на задачата**

Първият проблем, който възниква, е именно разделянето на символния низ на отделни думи, както беше споменато горе. Тук идва на помощ функцията **split**, която е известна и в други програмни езици (разделя символен низ на отделни думи). Следващият въпрос е как програмата разбира коя операция да изпълни? Чрез един switch блок, и с **checkOperation** става ясно коя точно операция трябва да бъде използвана. За правилната работа на ConsoleEngine, самият хотел е инициализиран в задейставащата приложението команда oт **ConsoleEngine - run()**.

Важно е да се спомене проблема с невалидния вход от потребителя. Програмата трябва да разбира кога някой се опитва да въвежда грешни неща и съответно или да го предупреждава или да спира действието и, но да не предизвиква bug-ове или crash-ове.

Нека първо разгледаме няколко невалидни входа:

- Подаване на текст, там където се иска число

- Подаване на повече или по-малък брой параметри отколкото е нужно

- Подаване на грешна команда

- Оставяне на повече място от нужното между 2 от аргументите или неоставяне на никакво място между тях

- Въвеждането на невалидна дата (може би най-сложната валидация в приложението)

Ако се стигне до някой от тези случаи, потребителят получава съобщение за грешка, което не прекратява програмата, а му позволява да въведе правилна команда.

От критично значение беше справянето с йерархичната структура на класовете, особено за датите в даден период и за по-лесното им управление и ползване във функционалностите беше важно да бъдат включени във всяка стая като резервирани дати. И към датите имаше проблем с взимането на всички такива в даден период (пр. 2020-15-05 до 2020-02-06), да бъдат валидирани и използвани правилно.

2.3. **Използвани подходи за решаването на проблемите**

https://i.gyazo.com/0299040e4e1dfff850a4240d9571be10.pngНека първо видим как става разделянето на отделни думи при въвеждането на символния низ.

Фиг.2 Команда за разделяне на символен низ

Командата приема символен низ и ни връща вектор от символни низове. Това е доста удобно, понеже ще имаме достъп до всяка една данна, която е необходима за програмата да разбере коя операция да извърши. А как всъщност се решава това?

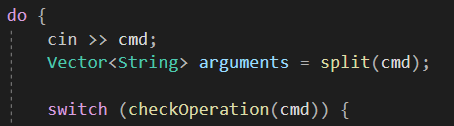
https://i.gyazo.com/c4631bef252eb1c735482cf2fcd1243c.png

Фиг.3 Всички валидни операции

https://i.gyazo.com/d7b405644e8708c064fdc4b6b5912cc1.png

Фиг.4 Фунцкия за определяне на правилната команда

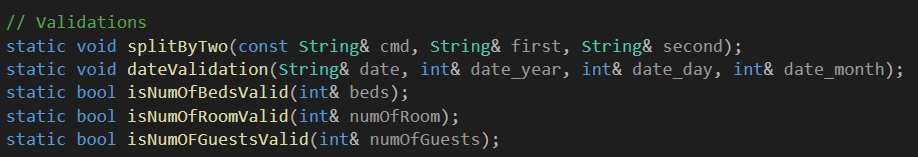
Използвайки функцията **checkOperation** ние разделяме със split подадения символен низ на вектор от думи и проверяваме дали първият елемент на този вектор отговаря на някоя от командите от Фиг.3. Ако съвпада ние връщаме число което е с едно по-голямо от позицията на тази команда в масива от Фиг.3. Така влизаме в някои от слуачите на switch блока в командата **ConsoleEngine::run()**.



Фиг.5 Задействането на run()

Нека сега видим някои от валидациите, които предотвратяват грешен вход от потребителя. Като за начало, ако входът на потребителя НЕ започва с някоя от операциите излиза грешка. Във всяка от отделните валидации има проверка за разстоянието между думите така че то също да бъде валидно.

За улеснение на работата е създаден клас **Operation**, който с служи за въвеждане на команди, изчистване на данни, валидиране и други. Той е изцяло статичен.



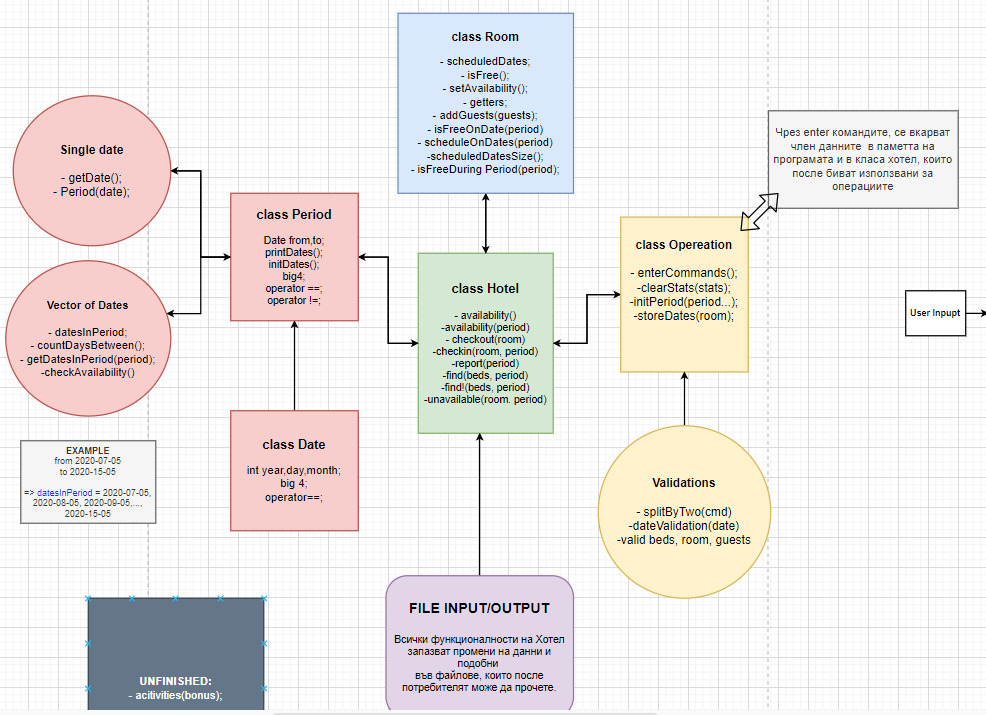
Фиг.6 Някои от валидациите

https://i.gyazo.com/df868cc54da04abdcd6948b68509f585.pngИнициализирането на дати също беше един от най-основните проблеми.

Фунцкията **initDates** добавя всички дати в даден период във вектор от дати, предотвратява грешки, като това дава лесен достъп до всички от тях. Тези дати са ключови за цялостната работа на програмата, понеже се съдържат в почти всички от основните операции. Именно те се използват за запазване на дадена стая в определен период например, или проверяване на наличност в даден период.

3. Проектиране

3.1. **Обща архитектура – ООП Дизайн**



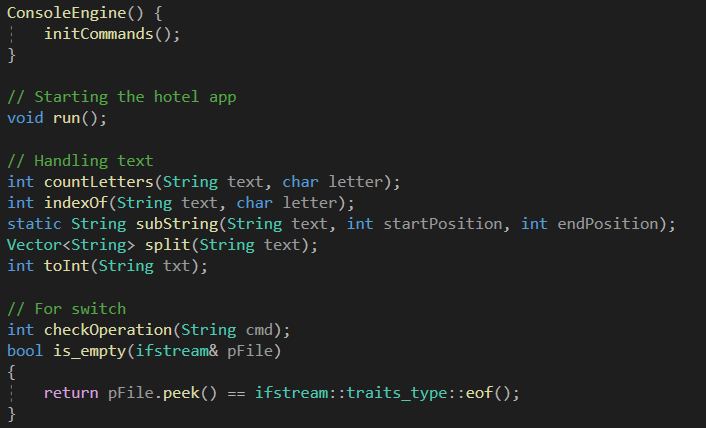
Фиг. 7 Архитектура на проекта

На фигурата са показани главните класове, заедно с член данните им и как си взаимодействат помежду си. Главният е класа Hotel, който съдържа всички основни операции.

Абстрактен тук е класа Operation, поради по-лесната му употреба. Методите му разчитат по-скоро на параметрите, които са му подадени, отколкото на член данни.

3.2 **Структури и поведение**

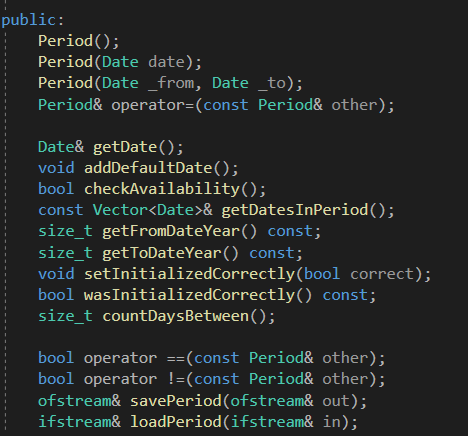
Извадките представляват използваните методи за реализирането на конзолната работа и периода (основните двигатели на приложението). Пълните дефиниции и подробната работа на тези методи може да се намери в кода.



Фиг. 8 Декларации на методи в конзолата

Този клас има член данни:

* String cmd – въведената команда от потребителя
* Vector<String> commands – вектор от възможните команди
* String commands\_arr[13] – списък с правилните команди
* void initCommands() - инициализирането на commands\_arr[13]



Фиг.9 Дефинициите на методите в класа Period

Това е може би най-същественият клас в програмата, както обясних по-горе в документацията. Той е най-често използван в **Hotel**. Съдържа следните член данни:

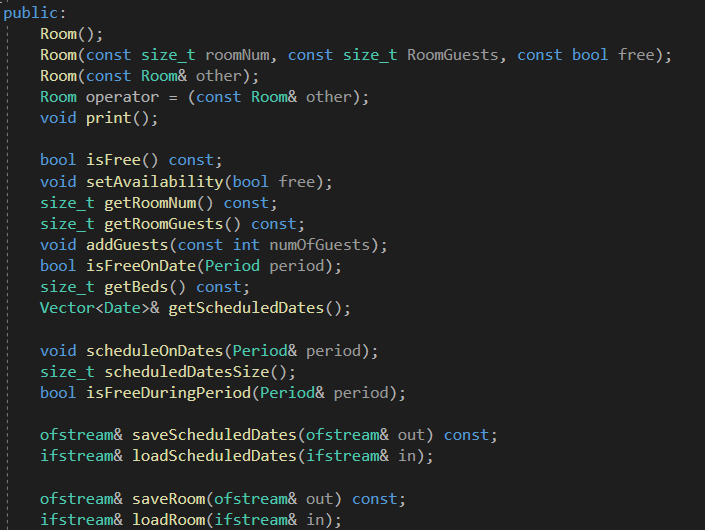
* Date from, to – датите, нужни за хотела от-до
* Vector<Dates> datesInPeriod – съхранява всички дати в интервала
* bool initializedCorrectly – помага за валидирането
* void printDates() – принтира датите в периода
* void initDates() – инициализиране datesInPeriod

За правилната работа с файла hotel.txt, всеки клас си има съответно **save** и **load** методи, които служат за записването в него и четенето от него. Такива са save/load Period, save/load Date, save/load Hotel.

4. Реализация и тестване

4.1. **Реализация на класовете**

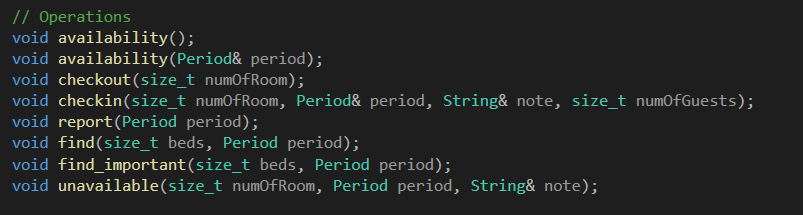
Ще започнем с класа Room.



Фиг.9 Декларации на методи на стая

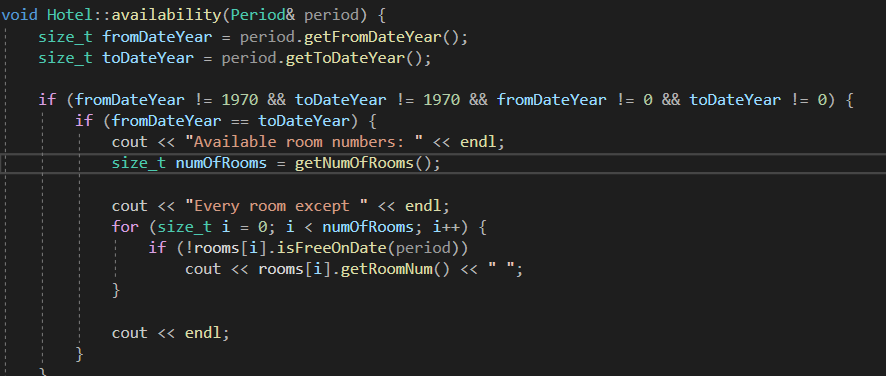
Стаята съдържа номер, гости, легла, информация дали е заета и списък от заети дати. Тя притежава нужните сетъри и гетъри и методи, които променят заетите стаи по правилен начин, така че в последствие потребителят да има достъп до правилно заредена информация. Има и save/load ScheduledDates, които записват и четат от главния файл на хотела заетите дати.

Нека разгледаме реализацията на основните операции в Хотел.



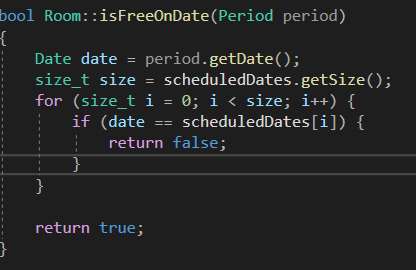
Фиг. 10 Декларации на основните методи в Хотел

Реализациите на някои от по-важните функции са на **availability**, **checkin** и **checkout**.

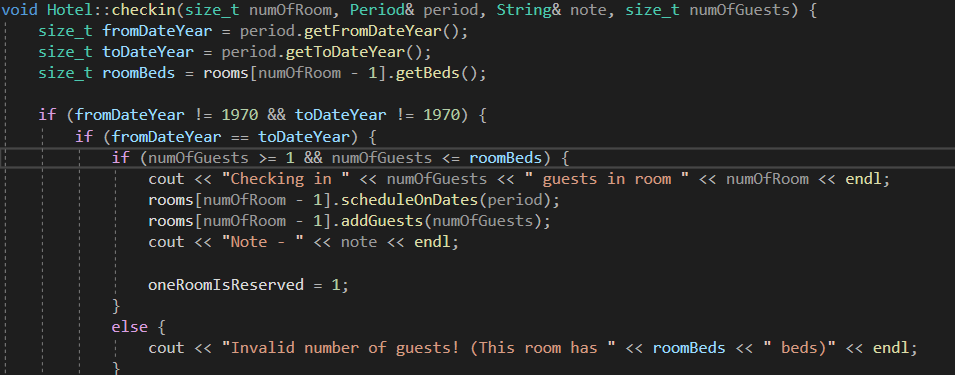


Фиг. 11 Дефиниция на availability

Този метод получава период като параметър и работи около него. Първият if служи за проверка дали годините на датата са инициализирани правилно, а не са 1970, която е default стойността. След това ако двете години са равни (понеже не може да се прави резервация за по-голям период от 3 месеца) тръгва цикъл, който обхожда всички стаи. Понеже хотелът първоначално има повече свободни, отколкото заети стаи, реших за удобно да се показват стаите които НЕ са заети за по-четим код. Съответно викаме функцията isFreeOnDate(period) на всяка една от стаите и ако не е свободна, да се изведе на потребителя. Тук идва въпросът как е реализирана isFreeOnDate?

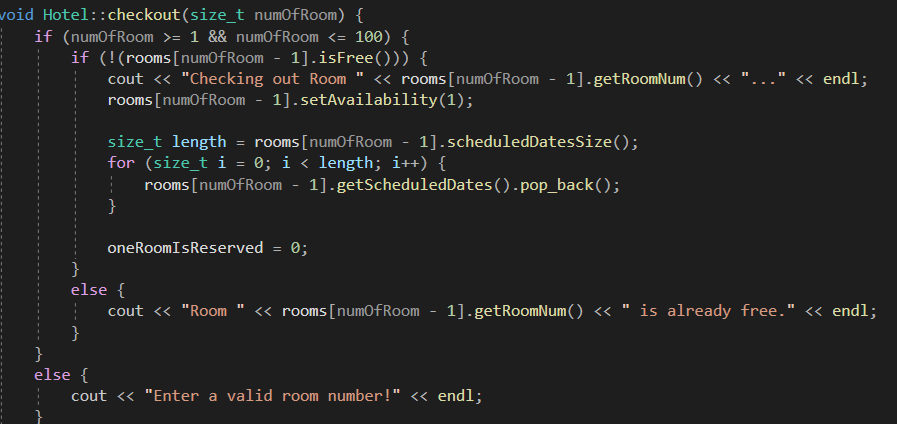


Обхождаме всички дати, за които съответната стая е заета. Нека например те са 10 на брой. Ако датата (или периода), който сме подали като параметър се съдържа в този вектор от заети стаи, то съответно стята не е свободна за този период дори и само една от тези 10 дати отговаря на подадената като параметър.



Фиг. 12 Дефиниция на checkin

Използваме парамтерите, като вземем годината на началната и крайната дата за резервиране и броя на леглата в стаята, която искаме да настаним гост. if-а е използван пак за валидиране, но този път и за броя на леглата, които се генерират на случаен принцип от 1 до 5 във всяка стая. Ако всичко е наред запазваме датите в периода чрез scheduleOnDates, които добавя всяка една дата в периода във вектора от резервирани и добавяме броя на гостите в стаята.



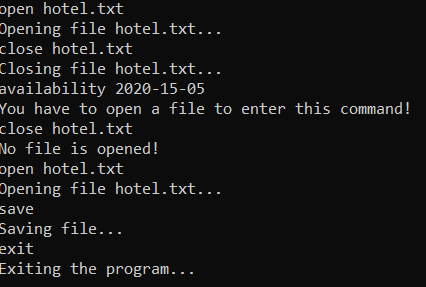
Фиг.13 Дефиниция на checkout

Тази функция получава като параметър номера на стаята, която искаме да освободим. Чрез !isFree() проверяваме дали стаята не е свободна и при такъв случай, променяме булевата стойност за достъпност на стаята до True. След това обхождаме векторът от заети дати за съответната стая и чрез pop\_back() премахваме всяка една дата от вектора, така че да се освободи.

4.2. **Управление на памет**

Динамичната памет се контролира най-вече от Vector и от String, които са ръчно написани и се справят доста добре именно с това. Имаше проблеми, които се появяваха по време на тестване на програмата, но бяха изчистени във методите най-вече на вектора. Основно програмата работи със статична памет (в stack-a), като за целта функционалностите на приложението са оптимизирани напълно.

4.3 **Примери**



Тук се вижда работата с файлове. Open отваря поток към даден

файл, а ако такъв не съществува, се създава автоматично. Close

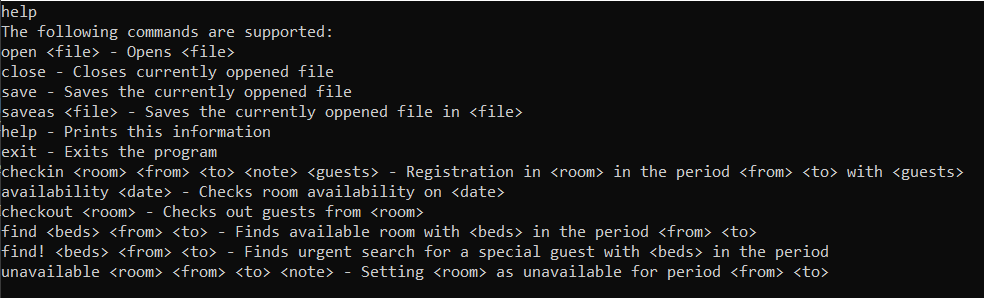
съответно затваря файла, ако е отворен. Ако се пробваме да

изпълним някоя команда без да има отворен файл, излиза грешка.

Save запазва направените промени по отворения файл и exit спира

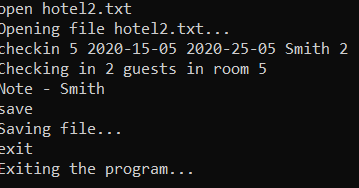
изпълнението на програмата.

**help**

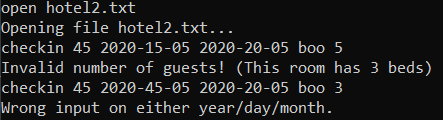


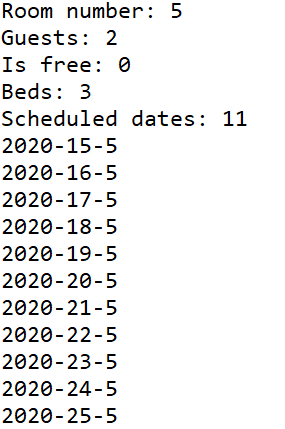
Командата help дава възможност на потребителя да види валидните операции.

**checkin**



Когато използваме checkin първо подаваме номер на стаята, в която ще резервираме, после периода, бележка и колко гости ще бъдат настанени. Естествено, ако някой от тези аргументи бъде изпуснат или написан в грешен формат излиза грешка.



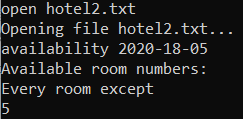
Пр. В стаята не могат да бъдат настанени 5 госта, тъй като има само 3 легла. При второто изивкване първата дата е невалидна (няма 45-ти ден).

Когато гостите са настанени правилно във файла излиза информацията.

**Важно: Тези валидации ги има за всяка една от функциите, затова няма да ги споменавам за при описанието на всички операции.**

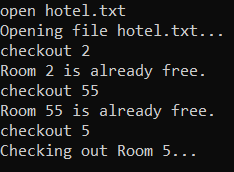
**аvailability**

При зареждането на същия файл, информацията бива прочитана и съответно излиза че стая номер **5** е резервирана на дата, която е запазена за тази стая, поради използването на checkin в миналия пример.



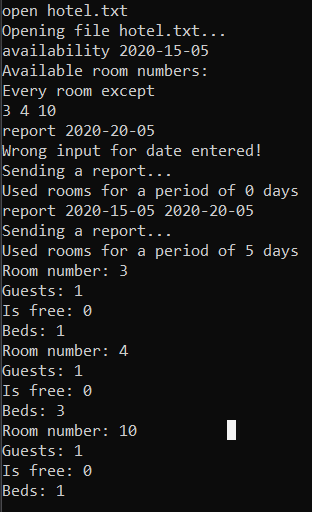
**checkout**

Ще освободим стая номер 5. Ако се пробваме да освободим свободна стая, излиза подходящо съобщение.



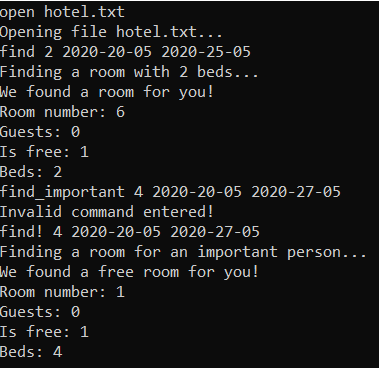
**report**

Извежда справка за използването на стаи в период (две дати).



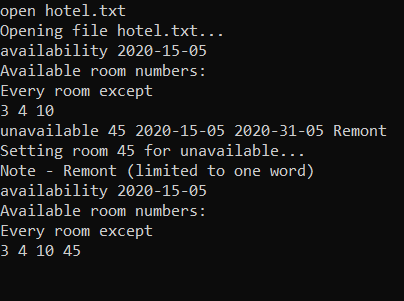
**find & find\_important**

Двете функции са доста идентични с разликчата че при find\_important се реализира разместващ алгоритъм за намиране на стая.



**unavailable**

Обявяваме дадена стая за неизползваема. Естествено информацията отива във файла и се прочита обратно.



5. Заключение

Преглеждайки функционалностите на приложението, става ясно, че то наистина е доста удобно, оптимизирано и лесно за използване. Предвид нарастващата необходимост от софтуер в почти всяка индустрия, включително и хотелиерството, едно такова приложение би могло да бъде от голяма полза или да даде основите на още по-развита и функционална информационна система. В бъдещете към приложението може да бъде добавена опция за резервации от по-ранен период, също и добавяне на предпочитани дестинации или сезон на пътуване. В допълнение, може да бъде изграден привлекателен потребителски интърфейс за по-приятна работа със системата.