Линк към страницата на приложението: <https://github.com/xhist/Hotel>

**Предназначение на документацията –** документацията служи като помощно средство за вникване в идеята на приложението, както и за усвояване на работата с неговите функционалности. Подходяща е за всички лица, които изпитват интерес както към развитие на проекта, така и към експлоатирането му. Целта за написването на документацията е да представи детайлно софтуерната архитектура на приложението, да разгледа отделните компоненти от приложението, както и техните характеристики и отговорности.

**Структурата на документацията:**

* **Въведение –** има задачата да запознае всички, които имат желание да научат повече за приложението, с целите на проекта, какво се стреми да постигне и как се опитва да помогне в разрешаването на даден проблем.
* **Преглед на предметната област –** включва основни понятия, дефиниции, обвързани с предмета, като база от знания, чрез която приложението се разработва. Посочени са алгоритми, които са били адаптирани според нуждите от приложението. Обръща се внимание на проблеми, възникнали по време на работа по приложението, както и на тяхното методическо и подходящо преодоляване. Изложени са функционални и нефункционални изисквания на приложението.
* **Проектиране –** представя архитектурата на приложението под формата на диаграми, разглеждат се отделните компоненти, връзките между някои от тях, разкрива значението на всеки един отделен компонент с отговорността му към приложението. Визуализира начина на протичане на процесите в приложението.
* **Реализация, тестване –** добавя записки относно избрани моменти, които са били запаметени като ключови за приложението и подобряват механизма на работа на приложението. Описва се разпределението на паметта, както и дейността с нея, отделяме ключови за приложението алгоритми за подробно изследване. Прави се анализ на тестовите сценарии за изпробване на функционалностите на приложението, като подлагат приложението на принципа „проба-грешка”.

1. **Увод**

Идеята на приложението произхожда от необходимостта за предоставяне на механизъм за управление на хотели. Приложението се фокусира върху реализирането на информационна система, която обработва заявки от страна на гости с помощта на командния интерфейс. Целта на приложението е да поднесе удобна за работа с управление на даден хотел система, съпроводена с интуитивен дизайн. Когато говорим за хотел възниква представата за контролиране на всички стаи в него, гостите настанени в тях, преместването на гостите между стаите, отписване на гости и други дейности, обвързани с цялата структура на хотела. Това изглежда на пръв поглед трудоемко за поддържане, ако не съществува система, която да се грижи за установяване на ред в хотела. Приложението разчита на този факт, като то вмъква арсенал с необходимите за управлението на хотела команди, за да опрости оперирането с структурата на хотела. Като част от леснодостъпния си, интуитивен и годен за разширение дизайн, приложението предлага взаимодействие с текстови файлове, в които се съхранява информацията за даден хотел. На фона на това взаимодействие изскачат и операциите за работа с хотела – намиране на подходяща свободна стая за гости, преместване на гости от конкретна стая с цел освобождането и за специални гости и ред други функционалности. Това са и основни задачи, които приложението се стреми да преобразува в код и да може да се използват през конзолата. Зависимостта от това да изберем текстови файлове преносител на информацията за управлението на хотел е лично предпочитание, поради причината, че класовете съдържат собствена логика за четене и записване в файлове.

**Задачи на разработка:**

* **Работа с текстови файлове –** основна задача за приложението. Тя включва отделни подзадачи, които са разработени в рамките на приложението – четене от файл, записване в файл, затваряне на файл.
* **Настаняване на гости в определена стая –** тази операция се грижи за настаняването на дадена бройка гости в указана стая, възниква и въпроса дали съществува такава стая с толкова легла, колкото да побере броя гости. Ако няма такава стая се възстановява операцията, за да изпрати съобщение за грешно предоставена информация.
* **Проверка за свободни за резервация стаи –** извежда списък с всички свободни стаи за посочена дата, които са налични за резервация. Ако не е указана дата, се взема предвид текущата дата.
* **Отписване от стая –** отписва гостите от указана стая, маркира стаята за свободна за резервация.
* **Доклад –** описва всяка стая поотделно и изкарва доклад за това в указан период от време колко дни е била използвана.
* **Намиране на подходяща стая –** открива подходяща свободна за резервация стая в някакъв период от време според даден брой гости.
* **Намиране спешно на стая за специални гости –** търси стая, в която да настани даден брой специални гости, и ако намери такава стая с поне толкова на брой легла колкото са гостите освобождава тази стая за тези специални гости, ако не е свободна за резервация. Подчертава се факта, че стаята, която запазваме за специалните гости, може да не е свободна за резервация затова прибягваме към преместване на гостите в друга подходяща свободна стая, за да освободим текущата.
* **Обяваване на стаята за недостъпна –** стаята става недостъпна за даден период от време като този процес се изпълнява при условие, че в стаята не пребивават гости или няма запазена активна резервация.

Цялото внимание при оформянето на финалния облик на приложението е съсредоточено в осъществяването на разработка на изброените задачи, които са минимумът от операции, които приложението да бъде способно и в състояние да изпълнява.

1. **Преглед на предметната област**

А) Запознаване с предметната област

Преди да се процедира с разработването на едно приложение винаги започваме от въпроса каква е темата, върху която ще се базира приложението. От този въпрос изхождат и други въпроси като какви са понятията, концепциите, задълбаващи в усвояването на темата. Основни понятия като това на какъв принцип да се чете или записва в един файл, каква да е структурата на една стая, от какво да се състои, колко активни резервации да има една стая, как да се осъществява отписването от стаята, са придобити в рамките на приложението. Това, което ни вълнува с особено значение е въпросът за броят на активни резервации на една стая. В контекста на приложението се смята, че една стая може да има една единствена активна резервация.

Б) Проблеми, създаващи препятствия при разработката

В течение на разработката на приложението се застъпват няколко проблема, които попречват на процеса на развитие на приложението. В множеството от проблеми попадат например алгоритъма за намирането спешно на стая за специални гости, изчислението на разликата между две дати в дни, подредбата на периодите от време за всяка една резервация, валидността на данните за хотела, пристигащи от даден файл. Всичките тези изброени проблеми провокират допълнителното разучаване на структурата на всеки един компонент сътворяващ архитектурата на приложението. Като например взимането на разликата между дати е необходимо да се съобрази и годините, които са високосни. Потока от информация, насочен към приложението, съдържа информация за всяка една стая на хотела с нейните резервации. Не се изключва шансът тези резервации да са в разбъркан ред, което би се оказало предизвикателство за приложението, което трябва да изпълни. Това също хвърля поглед върху допълнителни размишления върху структурата на класа Date, които ни довеждат до изменение на общия вид на класа Date – операции за сравнение на дати. Проблемът с най-голяма тежест, който прилага най-голямо въздействие върху мисълта за промяна на общата архитектура на приложението е алгоритъма за спешното намиране на стая за специални гости.

В) Подходи и методи, използвани в рамките на приложението

Като подходи и методи, които се упражняват върху преборването на проблемите натъкващи се при реализацията на приложението, са заимствани от основните алгоритми, които съпровождат реализацията на някои функции в приложението. Подобни алгоритми са обхождане на масив, търсене на елемент в масив, сортиране на масив, изчисления, проведени върху дати, прехвърляне на елементи от един масив в друг.

Г) Функционални и нефункционални изисквания

За употреба на приложението потребителят трябва:

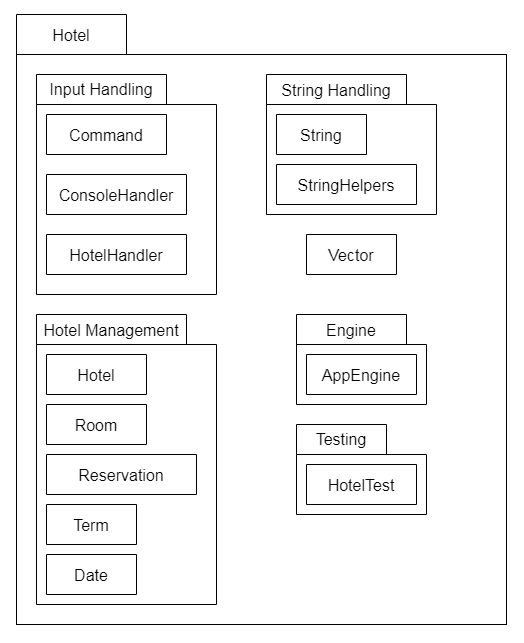
* да записва датата в формат [YYYY]-[MM]-[DD] спазваща стандарта ISO 8601.
* да указва за всяка една резервация в текстовия файл дали е приключила чрез латинските думи “yes”, “no”, като думата се изписва след периода от време, през който тече резервацията, и преди бележката, оставена след осъществяване на резервацията.
* да посочва най-много до една активна бъдеща резервация в текстовия файл с съдържанието на хотела.
* да записва всяка команда с нейните параметри на латински език
* да посочва от колко стаи се състои един хотел, за да се увери коректността на функционалностите.
* да разделя периодите от време на всички резервации, принадлежащи на дадена стая, така че да не се секат.

Като изисквания, които да гарантират качеството и продуктивността на приложението, са :

* когато не се указани колко на брой легла да се заемат при настаняване в конкретна стая, да се избере пълния брой легла, попълващи пространството в стаята.
* при изпускане на дата при проверката на стаи, свободни за резервация, да се използва текущата дата, съвпадаща с тази на устройството, от което се ползва приложението.

1. **Проектиране**

А) Общ вид на архитектурата на приложението



На снимката са илюстрирани всички класове, от които заимства приложението за механизма си на работа. Класовете са обединени под общи модули с различен смисъл за приложението и изпълняващи отговорност към процесите, течащи в приложението.

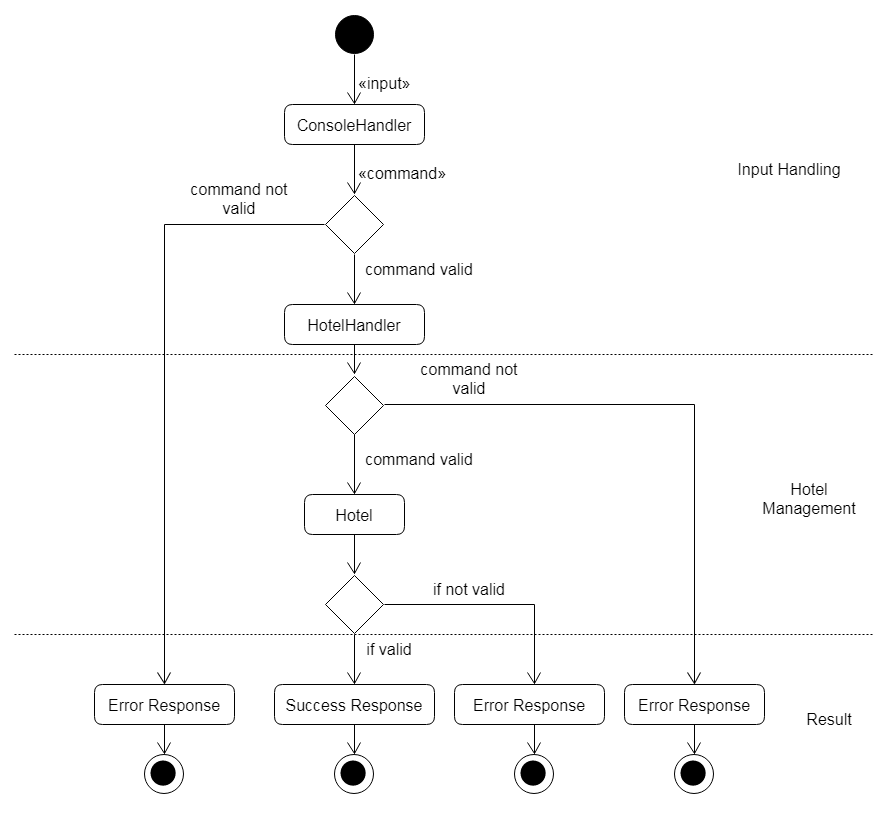
Характеристика на всеки един модул:

* Input Handling – този модул се обвързва с операциите, които работят с въведени през командния интерфейс данни. Част от класовете в този модул са в линейна свързаност помежду си.
* Hotel Management – предназначението на този модул се върти около главната цел на приложението – получаване и обработка на заявки от потребителя.
* String Handling – опростява работата с низове в общността на приложението, въвежда допълнителни операции над низове.
* Engine – от английски означава двигател. На този модул е назначена задачата да образува интуитивният дизайн за работа с приложението. Включва цикълът от действия, които се изпълняват в приложението.
* Testing – на този модул е присъдена процедурата по тестване на изпълнението на приложението. Съдържа тестовите сценарии демонстриращи работата на приложението.

Описание на отделните класове(компоненти) на приложението:

* **Command –** занимава се с организацията на данните, въведени в конзолата, за по опростена работа с тях в останалите компоненти.
* **ConsoleHandler –** поема ролята да се грижи за прочитенето на командите от конзолата, тласка класа Command в употреба при реализацията си.
* **HotelHandler –** заявките при четене от конзолата попадат тук и се изпълнява съответното действие. Служи като преходен компонент между ConsoleHandler и Hotel.
* **Hotel –** основен компонент на приложението. Тук са прехвърлени всички действия, отговарящи на заявките в приложението. Между класовете Room, Reservation, Term, Date съществува йерархична организация, в която се слиза последователно надолу от класа Room до класа Date.
* **Room –** представя една стая на абстрактно като структура от данни и действия с тази стая.
* **Reservation –** моделира образа на резервациите в областта на приложението. Имплементира данните и операциите за взаимодействие с една резервация.
* **Term –** определя как е устроен един период от време чрез приложението на дати.
* **Date –** интегрира дата в приложението, следвайки стандарта за дата, споменат в изискванията на приложението.
* **String –** попълва позицията на примитивните низове в езика C++, добавя допълнителни операции с низове.
* **StringHelpers –** въвежда всички операции с примитивни низове, които се използват като помощни пособия за оформяна на класа String.
* **Vector –** преобразува нормалния масив от елементи в стабилен, разширяващ се масив от елементи, който пази и рекорд за броя на съществуващите елементи в себе си.
* **AppEngine –** ядрото на приложението; обвива цикъла от действия, изпълняващи се приложението, в един метод за лесно достъпване на приложението; задвижва механизма на работа на приложението.
* **HotelTest –** съхранява тестовите сценарии, демонстриращи функционалностите на приложението.

Б) Обща структура на заявките в приложението



Общо описание на структурата на изпълнение на заявките в приложението:

* **Начална/отправна точка** (отбелязана с черна точка) – маркира началото, откъдето започва изпълнението на приложението, както и момента, в който трябва потребителят да поиска каква заявка да изпълни приложението.
* **„Работници”** (изобразени с правоъгълници с заоблени върхове) – причината да се наричат работници е, че те прилагат някакви действия при работата с заявката. Може да се нарекат още преходни състояния на приложението, при които се образува някакъв маршрут на действие.
* **Поток на действие** (изобразен с стрелки) – посочва хода на действие на самия процес на изпълнение на заявката, нарежда по какъв маршрут да тръгне заявката.
* **Условие** (изобразено с ромб) – прави оценка на данните или действието, които пристигат от дадено място, която определя какъв път ще поеме процеса на изпълнението на заявката.
* **Крайна точка** (изобразена с черна точка, вградена в бяла точка) – съобщава на процеса на изпълнението на заявката, че няма повече стъпки и процеса се прекратява.

Преди да се втурнем в използването на приложението първо трябва да се разясни през какви стъпки минава процеса на изпълнение на заявките в приложението. На снимката много чисто и ясно се вижда последователността от стъпки, които приложението спазва. Първо се инициализира конзолата, за да подкрепи потребителя с умението да пише в конзолата. След въвеждане на командата, която съответства на дадена заявка, данните се удостоверяват в следващия етап от процеса на изпълнение на заявката. Този етап е обозначен на снимката чрез класа ConsoleHandler, който засича в данните дали има нередности, при което изпраща сигнал за грешка към конзолата и заявката се прекратява. Ако всички проверки са минали успешно, то тогава се процедира към следващия етап и това e хода на изпълнение на действието, което заявката принуждава да се извика. Това се контролира от класа HotelHandler, който също както класа ConsoleHandler анализира данните за потенциални повреди и се прехвърля към прекратяване на всякакви действия. При успешно проведени проверки информацията, носеща се от класа HotelHandler се изпраща към последния етап от изпълнението на заявката и това е самият хотел, който трябва да изпълни заявката на потребителя, при което при данни, нарушаващи валидността за изпълнение на заявката, се стига до приключване на процеса на изпълнение на заявката. В противен случай действието при изпълнение на заявката минава успешно и се извежда крайния резултат.

1. **Реализация, тестове**

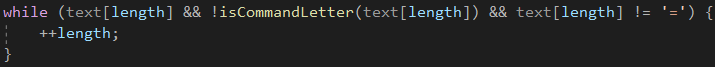
При реализация на приложението често може да се забележи на места даден клас да се отличава или изпъква пред останалите класове или компоненти от архитектурата на приложението с неговите качества, характеристика и значение за приложението. Тези черти за самият клас или компонент на приложението го поставят в ключова за приложението позиция, която му позволява да превърне приложението в удобна за взаимодействие с потребителя среда, както и да премахне граници, които ограничават приложението в затворена област. С други думи тези ключови моменти се стараят да направят приложението отворено за нови промени с цел развитие и да подобрят преживяването при употреба на приложението. Примерни класове, които индикират за това, че в тях са концентрирани ключови роли за приложението, са описани по-долу с техните член-данни и член-функции, за да потвърдят ръководещата роля на класа за приложението.

А) Описание на ключови класове:

1. **Command**

Класа Command се отличава по смисъла си на извличане на информацията, въведена от потребителя през командния интерфейс, като това предоставя на останалите класове лесното жунглиране с тази информация. Класа е поставен на ключова позиция, която също така носи и отговорност към приложението. Отговорността се корени в фактът, че самият клас трябва да пренася информацията, която е взел от потребителя, към останалите класове, за да може процесът на изпълнение на заявките да е завършен.

* член-данни:
  + String name – съхранява информация за името на командата. В полето се идентифицира класът String за подобрената работа с низове.
  + Vector<String> parameters – управлява всички параметри, които една команда приема. Използва се класа Vector, който като елементи приема String. Първопричина за това решение е съкращаване на прекомерната употреба на допълнителни член-данни за построяване на динамичен масив, както и възможност за опростените операции по отношение на динамичните масиви.
* член-функции:
  + int extractCommand(const String& text, Vector<String>& words) – смисълът на тази функция лежи в изискването да разграничим от дългия текст, идващ от конзолата, кой низ отговаря за името на командата и кои думи са параметрите на командата. Обяснено по друг начин, функцията се стреми да извлече от целия текст, въведен от потребителя, името на командата като първи низ и всичко останало след нея са параметрите на командата. Параметрите, участващи в функцията, са познати до момента и се поставя акцент на втория параметър, който сложи за временен контейнер, в който съхраняваме всички отделни думи, на които сме разцепили текста, символизиращ командата.



Самата функция действа, опитвайки се да разчете допустим символ, като заключението за това как разбира дали един символ е допустим е употребата на функцията isCommandLetter(text[length]), която връща като отговор дали един символ е допустим, като това представлява правило, дефинирано от настройката на системата. Друг важен детайл от снимката, представяща част от функцията е символът „=”, чието задължение е да провери кога има дума, която представя името на някакъв параметър, който изисква стойност след „=”, за да може името на параметъра и стойността да са слепени като един общ параметър. Важно е да се отбележи, че цикълът, който е показан на снимката проверява дали символът спада към допустимите и ако не спада да изпълним съдържанието на цикъла, което означава да прескочим с една позиция напред текстът, който е въведен от потребителят.

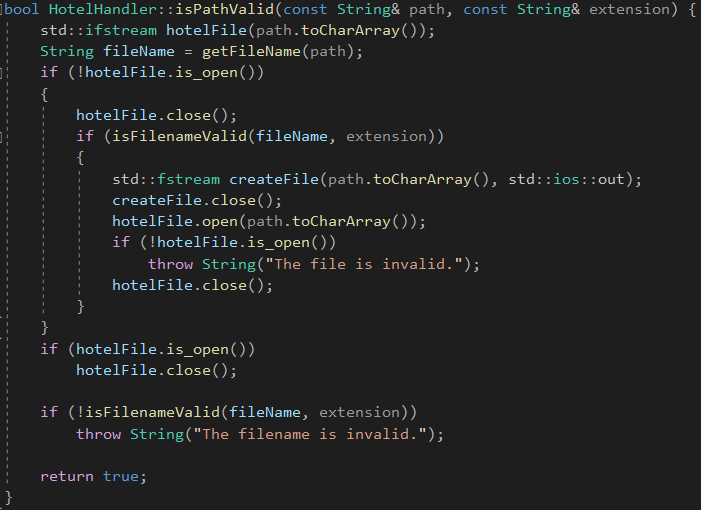


В тази част от кода, формиращ функцията, се открива условието, ако символът, към който сочи текущата позиция в низът, принадлежи към допустимите, записани от системата. Фокусът тук пада върху цикълът, който преглежда последователността от допустими символи, за които се изчислява размера на думата, която се формира от тези символи и която се добавя към векторът от думи, който е параметър на функцията.

1. **HotelHandler**

Съдържанието на този клас е закачено за функционалността на класа Hotel, като връзката не е само насочена към него, но и към класа ConsoleHandler. Класът HotelHandler действа като медиатор между двата класа Hotel и ConsoleHandler, защото осъществява доставянето на информацията от ConsoleHandler до Hotel. Обаче ролята на класът HotelHandler не се изчерпва само с построяването на връзка между класове, но и поема тежестта да обработва данните за безопасна транзиция от единия клас до другия. Това означава, че класът HotelHandler прилага серия от валидационни правила като предпазни мерки срещу фалшива информация. Част от неговите член-данни и член-функции го превръщат в специален за приложението компонент.

* член-данни:
  + String currentOpenFile – при отваряне на нов файл неговото име се запаметява в това поле, като не се допуска отваряне на нов файл, ако текущия не е затворен. Това действа като предпазен механизъм срещу инцидентното използване на командата за зареждане на нов файл.
* член-функции:
  + bool isPathValid(const String& path, const String& extension) – прави задълбочено изследване на името на файла, като претърсва за разширението на файла, което трябва да съвпада с посоченото като параметър на функцията. Приоритет е и проверката за валидността на файла – дали съществува или дали пътят към файла е правилно указан.



Като незаменим детайл открояващ се на снимката е функцията toCharArray(), която е член-функция на класа String. Характерното за тази функция е, че връща като резултат const char\*. Това на пръв поглед изглежда странно, съдейки по факта, че класа String е натоварен със смисъла да може да предизвика липсата на примитивните низове в рамките на приложението. Тук този детайл се появява поради главната причина, че потоците за манипулация на файлове са реализирани чрез примитивните в езика C++ низове и те нямат достъп до съдържанието на персонално създадения низ String.

Прехвърляме се към следващия детайл от картинката и това е функцията getFileName(path), за която се подразбира, че връща като резултат String. Следващите стъпки при процеса на валидиране на файла застъпват създаване на файла, ако не е бил създаден, сравнение на разширението на файла с посоченото при параметрите на функцията. Ако сравнението се е провалило, тогава се излиза от тази функция, за да се изведе съобщение за грешка.

1. **Hotel**

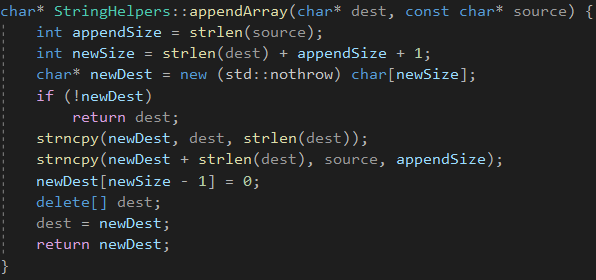
Класът Hotel контролира цялата логика на приложението. По всичко личи, че този клас е най-ключов за приложението, защото без него приложението няма да бъде реализира. Тъй като цялата функционалност на приложението лежи в този клас, то тогава е достатъчно да се концентрира вниманието върху определени елементи от класа, за да се усвои пълната представа за ключовите моменти при реализацията на приложението. Като единствен примерен вариант за ключов момент при реализацията на класа е функцията за спешното намиране на стая за специални гости, тъй като в центъра на тази функция се внедрява почти цялата сложност на приложението. Функцията започва от момента да търси налична свободна стая за специалните гости. Ако не е намерена такава, тогава функцията се прехвърля към втори вариант – да намери стая, която да побере специалните гости и да измести текущите гости в друг налична свободна стая. Ако и този вариант се провали, тогава остава последна възможност – да се направи разместване на гостите от две стаи, като това се изпълнява на принципа – гостите от първата стая се местят във втората, а гостите от втората се местят в третата, която се предполага, че е свободна за резервация. Така по този начин се освобождава първата стая за специалните гости.

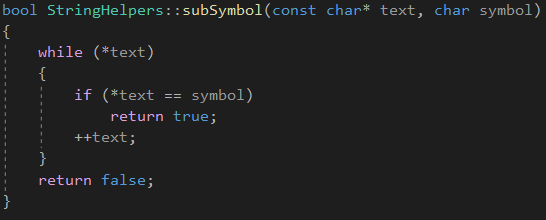
Б) Управление на памет и алгоритми

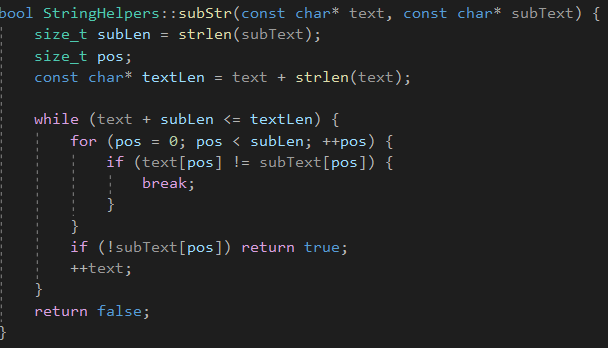
В хода на разработката на приложението винаги се разчита на това да съхраним данните в паметта – дали временно или за постоянно. Отчитайки факта, че се работи не само с статичната памет, в която пребивава съществена част от приложението, но и се работи и с динамична памет. Като примери за това са класовете String и Vector, в които се подготвят блокове от динамична памет за употреба. Обща характеристика на класовете:

1. **String**

Полето, което отговаря за съдържанието на класа, е от примитивен за езика C++ тип и то засвидетелства за използването на динамична памет сред реализацията на класа. Операциите, върху които стъпва реализацията на класа, също използват данни, които се възползват от динамична памет. По този начин класа се стреми да обвие цялата работа с данни, заделени в динамичната памет, в една абстрактна структура. Не трябва да се забравя, че голяма част от алгоритмите, съставляващи приложението, се концентрират именно тук. Приложението разчита на низовете като основен стълб на своята реализация. Почти навсякъде в приложението се употребяват низове и това подтиква написването на персонален низ String. Тъй като класа предоставя на абстрактно ниво работата с низове, то всъщност под цялата логика се крият алгоритми, които са обвързани с динамични примитивни низове. Такива алгоритми са например слепяне на два низа, търсене на низ в друг низ или на символ в низ. Тези алгоритми са показани чрез снимки по-долу, илюстриращи кода им.

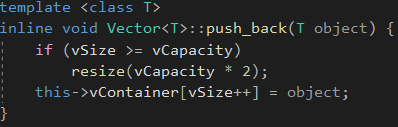


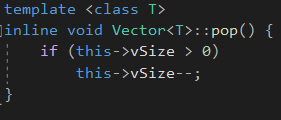


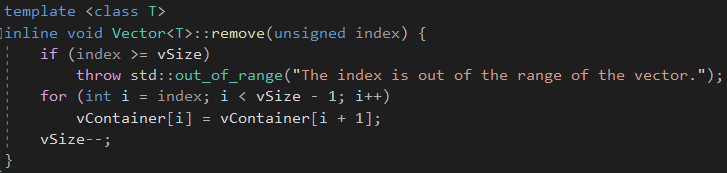


1. **Vector**

От описанието на класа Vector се знае, че замисълът му е да предостави обогатена функционалност за последователност от данни от определен тип. В рамките на класа се наблюдава обширната употреба на динамични данни. Още от член-данните се заключава, че има наличие на динамична памет. Един от първоизточниците на принципа на използване на динамична памет е възможността да се осигури контейнер с променлива дължина. Когато съчетаем такъв контейнер с други данни като например размер и капацитет в класа, приближаваме се до създаването на стабилен и гъвкав контейнер с данни, способен да редуцира съдържанието чрез премахване на елементи или да се разтяга чрез вмъкване на нови елементи. Класа, също както String, се доверява на базисни за приложението алгоритми за реализацията си, засягащи търсенето на елемент в масив, изтриване на елемент от масив, добавянето на елемент в масив. Надолу са изложени снимки, които изразяват алгоритмите чрез код.







В) Тестове и замисълът им

Преди да се стигне до финалната права, където се пуска приложението в продукция, първо се задава въпросът дали функционалностите на приложението работят изправно. Тогава на помощ се притичват така наречените тестове, на които замисълът им е не само да предоставят инструкции за работа с функционалностите, но и да съобщават за грешки при изпълнение на функционалностите, за да се прибегне до обновление на съдържанието на функцията. Тестовите сценарии, приготвени за приложението са:

* void testHotelValidCheckinWithoutActiveReservation() – тества поведението на функцията за настаняване на гости в стая, която е свободна за резервация
* void testHotelInvalidCheckinWithActiveReservation() – тества как се държи функцията за настаняване на гости в стая с вече налична активна резервация
* void testHotelValidCheckoutWithActiveReservation() – тества отписването на гости от дадена стая, когато стаята има активна резервация
* void testHotelInvalidCheckoutWithoutActiveReservation() – тества отписването на гости от дадена стая, когато стаята няма активна резервация
* void testHotelReport() – тества извеждането на списъка от стаи с определен период от дни на ползване на стаите в зададен период от време
* void testHotelAvailability() – тества излагането на списък от стаи, свободни за резервация след дадена дата
* void testHotelValidFindWithValidBedsAndPeriod() – тества изпълнението на функцията за намиране на стая за гости с брой легла по-малък или равен на максималния зададен брой за приложението, както и с валиден период от време разминаващ последния, за който е била направена резервация.
* void testHotelInvalidFindWithInvalidNumberOfBeds() – тества поведението на функцията за намиране на свободна за резервация стая за гости с брой легла, превишаващ максимума, дефиниран за приложението
* void testHotelFindEmergently() – тества функционалността за спешното намиране на стая за специални гости, която реализира алгоритъм за разместване на гостите
* void testHotelValidUnavailabilityWithoutActiveReservation() – тества функционалността за привеждане на статуса на една към недостъпна за гости, когато не съществува активна резервация за текущата стая
* void testHotelInvalidUnavailabilityWithActiveReservation() – тества функцията за превръщане на една стая в недостъпна за гости, когато стаята вече съдържа активна резервация
* void testAllHotelFunctions() – обхваща всички тестови сценарии в един общ подреден сценарий

Все още стои въпросът за подобрения към приложението. Това, в което се убеждаваме след многобройни изпълнени коректно тестови сценарии, е че приложението успешно постига целите, които се проследяват още в началото на документацията. Остава празно място за нови попълнения към функционалността на приложението като например поддръжката на множество от активни резервации, отписването от стая още преди края на периода от време, записване на гостите от стаята за различни дейности, провеждащи се в района на хотела и други.