



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

*Курсова работа по Обектно-ориентирано програмиране  
специалност Компютърни науки*

## Тема №15 Личен календар

## Съдържание

1. Увод.....	3
1.1. Описание и идея на проекта	
1.2. Цели и задачи на разработката	
1.3. Връзка към хранилището в Github	
2. Преглед на предметната област.....	4
2.1. Основни дефиниции, концепции и алгоритми, използвани в проекта	
2.2. Подходи и методи за решаване на поставените проблеми	
3. Проектиране	
3.1. Архитектура и реализация на класовете.....	5
3.2. Диаграми (най-важните извадки от кода).....	8
4. Реализация и тестване.....	10
4.1. Управление на паметта и алгоритми	
4.2. Тестове	
5. Заключение.....	11
5.1. Бъдещо развитие и усъвършенстване	

## **1. Увод**

### **1.1. Описание и идея на проекта**

Проектът „Личен календар“ реализира виртуален календар, в който потребителят може да създава срещи със съответно име и коментари на определената от него дата за желано време. Идеята е да се напише програма, реализираща информационна система, която поддържа личен календар, а информацията за него се записва във файл.

### **1.2. Цели и задачи на разработката**

Целта на проекта е да се създаде оптимална програма с максимално разнообразна функционалност, която да отговаря на поставената задача. Проектът е изграден съгласно добрите принципи на ООП.

### **1.3. Връзка към хранилището в Github: <https://github.com/bobivk/Calendar>**

## 2. Преглед на предметната област

### 2.1. Основни дефиниции, концепции и алгоритми, използвани в проекта

Данните в проекта са разпределени в различни класове, следвайки логиката на задачата. Информацията за класовете е разделена в *header* и *cpp* файлове за по-голямо улеснение при четене на кода, подредба и достъпност, като във всеки *header* се съдържа отделен клас, обединяващ необходимите за неговата имплементация член-данни и член-функции, които са разпределени в *public* и *private* секциите в зависимост от необходимия достъп до тях, а в *cpp* файла се намират дефинициите на декларираните в *header* член-функции. Също така са използвани външни функции, които не са методи на класовете. Използвана е конвенцията за именуване на функции и променливи *camelCase*, като функциите и променливите започват с малка буква, а класовете с главна.

### 2.2. Подходи и методи за решаване на поставените проблеми

Реализирането на класовете става посредством така нареченото правило на „Голямата 4-ка“, т.е. в *public* секцията им са деклариран *Constructor*, *Destructor*, *Copy Constructor* и *Copy Assignment Operator*, които помагат за правилното функциониране на програмата. В конструктора член-данните са инициализирани чрез така наречените инициализиращи списъци. Понеже има обекти, които не могат да се създадат чрез директно присвояване на член-данните, е дефиниран *Copy Constructor*. При изготвянето на проекта не се е наложило използване на наследяването, тъй като няма класове с общи компоненти и поведение като тези на вече дефиниран клас. Употребени са и ламбда функции, за да ползваме стандартните алгоритми, като `std::find_if()`;

Използвани са следните библиотеки:

- `<algorithm>` `//std::sort()`;
- `<cassert>` `//std::assert()`;
- `<fstream>` `//библиотека за работа с файлове`
- `<iostream>` `//стандартната библиотека за вход/изход`
- `<sstream>` `//stringstream`
- `<string>`
- `<vector>`

### 3. Проектиране

#### 3.1. Архитектура и реализация на класовете

Създадени са следните класове:

- Appointment
- Calendar
- Day

и структурите:

- Date
  - Parser
  - TimeInterval
  - TimePair
- направени са структури, а не класове, защото  
искаме да имаме директен достъп до тях  
вместо да ползваме селектори и мутатори

I. Клас Appointment се характеризира с член-данните и член-функциите в секцията със спецификатор за достъп *private*:

- `string name;` име на срещата
- `string comment;` коментар към срещата
- `TimeInterval timeInterval;` диапазон от време
- `void swap(Appointment& other);` помощна функция за *Copy Assignment Operator*

В *public* секцията се намират методите по *Rule of Three*, два конструктора (един по подразбиране и един, приемащ три аргумента – член-данните на класа), необходимите селектори за достъп до *private* данните на класа, предефиниран оператор `==` и функцията `void print(std::ostream&) const`, извеждаща информацията за създадената от потребителя среща и е направена така, че да работи с всякакъв вид поток (файлов или конзолен).

II. Клас Calendar съдържа член-данната `vector<Day> days` в *private* секцията. В *public* секцията са дефинирани:

- нужните конструктор и деструктор, в които съответно се извикват функциите за четене на файлове `void load (std::string fileName, bool merging)` и писане в тях `void save(std::string fileName)`
- `void book()` и `void unbook()`, които отговарят съответно за запазването на среща с посочени име, коментар, начален и краен час и отменяне на срещата на избраните от потребителя дата и час
- `void find()` намира и извежда данните за срещата по подадено име или коментар, а `void findSlot(Date, unsigned)` намира свободно място за срещата по подадени дата, продължителност на срещата, като го търси в

рамките на работния ден от 8.00 до 17.00 ч., ако няма свободен времеви интервал в този ден, преминава на следващия.

- `vector<Day>::iterator searchDay(Date date)` използва `std::find_if()`, за да намери ден във вектора `days` по зададена дата. Ако не съществува ден с тази дата, прави нов такъв и го добавя към `days`. И в двата случая връща `std::vector<Day>::iterator` към деня от `days` с тази дата.
- `void mergeWith(string)` от командния ред синхронизира информацията от два календара, като при застъпване на срещите в конкретни часове, потребителят може да избере кое от тях да остави или да напише ключовата дума *keep*, като по този начин запазва и двете събития

III. Клас Day се състои от член-данните в *private* секцията:

- `Date date;`
- `vector<Appointment> appointments;` вектор от сортирани по продължителност срещи
- `bool isHoliday;` булева променлива, която пази дали един ден е почивен или не
- `void sortAppointments();` сортира срещите по заемане на интервал от време за съответния ден с помощта на `std::sort()` `bool` и `compareAppointments(Appointment, Appointment);`

В *public* секцията се намират методите по *Rule of Three*, съответните селектори и мутатори,

- помощни булеви функции `bool isTimeIntervalFree(TimeInterval)` `const` и `bool appointmentExists(Appointment) const`
- `void addAppointment(Appointment)` проверява дали посоченият от потребителя диапазон от време е свободен, за да се добави срещата и съответно сортира с другите за деня, а ако не е свободен, се извежда подходящо съобщение в конзолата
- `void removeAppointment(TimeInterval)` търси определената среща по подаден от потребителя интервал от време и я премахва от вектора със срещи, като извежда съобщение, с което показва коя среща е премахната; ако не намери такава среща, извежда съответното съобщение
- `TimeInterval findFreeInterval(unsigned)` търси свободен интервал в периода от 8.00 до 17.00 ч., като работи с подадена дължина на интервала като аргумент в минути; ако при проверката се достигне до час, по-голям от 17.00ч. (1020мин.) или денят е почивен, се преминава към следващия ден, ако намери свободен, се връща празен интервал
- `void printAppointments(std::ostream&)` и `void print(std::ostream&)` извеждат необходимата информация за дните и съответните срещи, запазени в тях; направени са да работят с всякакъв вид поток (файлов или конзолен)

IV. Структура Date съдържа:

- член-данни от тип *unsigned* за ден, месец и година
- конструктор по подразбиране и  
`Date(unsigned day, unsigned month, unsigned year);`
- *Copy Assignment Operator*: `Date(const Date& other);`
- предефинирани оператор==, оператор<, оператор++ и оператор-- , за правилното функциониране на `void findSlot(Date, unsigned)` метода
- `void print(std::ostream&) const` извежда датата, като ако въведените ден и/или месец са число, по-малко от 10, се записват във формат например 05.05.2020

V. Структура Parser се състои от статични член-функции и се използва за превръщане на подадения от потребителя низ в конзолата съответно в дата и интервал от време в общоприетия формат

VI. Структура TimePair, съдържаща като член-данни часове и минути от тип *unsigned*, се ползва при метода `TimePair minutesToHours(unsigned minutes) const` в структурата `TimeInterval`, за да може да върне една променлива от две стойности

VII. Структура TimeInterval има:

- член-данните `start` и `end` от тип *unsigned* (все пак времето не може да е отрицателно число)
- метода `TimePair minutesToHours(unsigned minutes) const`, превръщащ минутите в часове, като проверява с функцията `std::assert()` дали минутите са в рамките на денонщието (1440 минути = 24 часа) и връща обект от тип `TimePair`
- съответните конструктор, копи конструктор и оператор=, заедно с помощната за имплементирането му функция `void swap(TimeInterval&);`
- предефиниран оператор==
- `void print(std::ostream&) const`, която извежда информация за времевия интервал във формат например 10:10 – 21:21

VIII. Main.cpp съдържа:

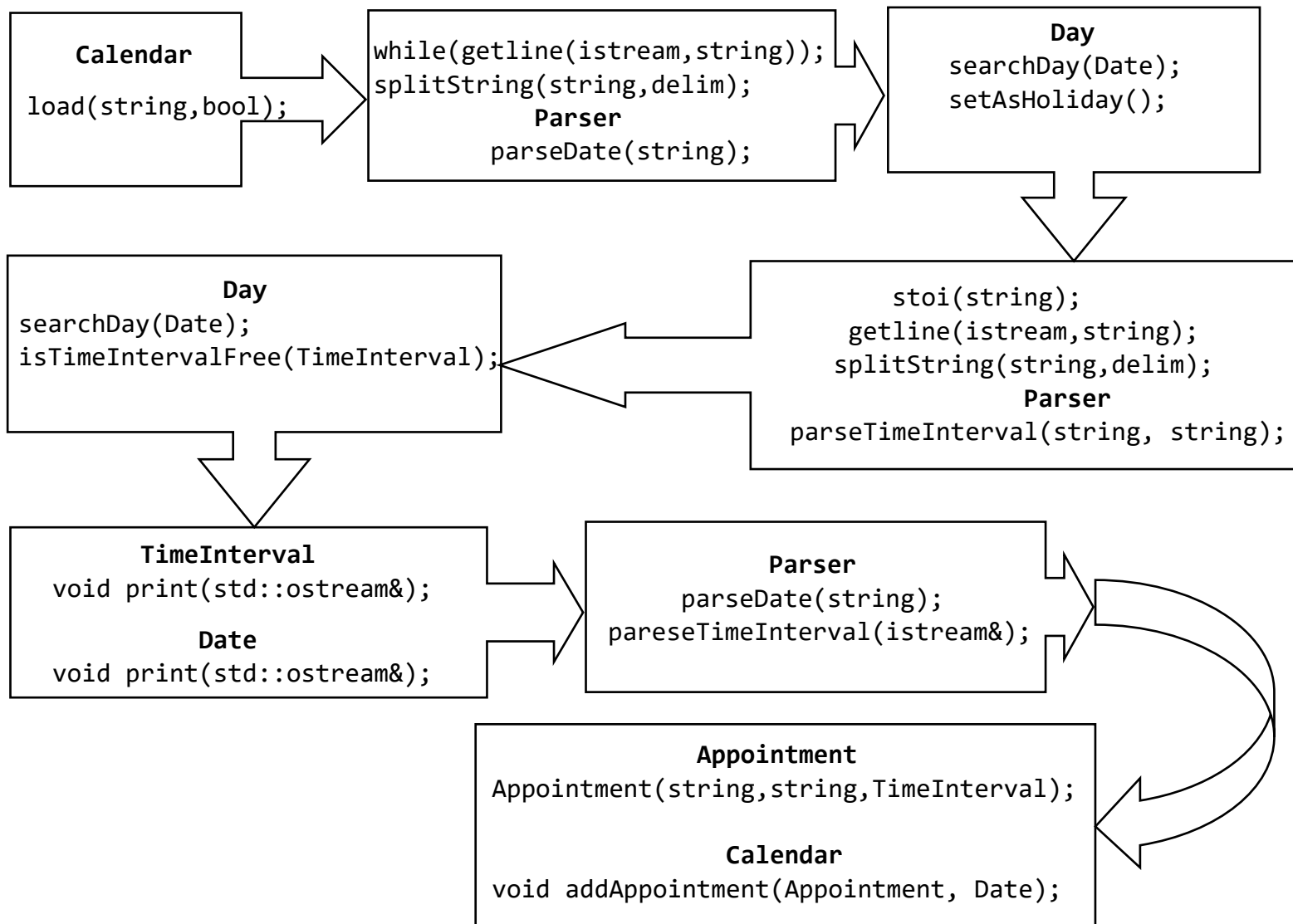
- `void help()` методът изкарва в конзолата помощното меню, което извежда поддържаните от програмата функции както и начина им на правилно изписване заедно с необходимите параметрите, които потребителят може да въведе от конзолата, за да работи с програмата
- `void findSlotWith(Calendar& first, string secondFileName, Date fromDate, unsigned minutes)` създава временен календар и го

`void findSlot(Date, unsigned)` на синхронизирания календар

- `void runProgram(Calendar&)` е функцията, в която се проверява въведената от потребителя команда под формата на низ дали съответства с някоя от поддържаните от програмата команди; ако въведената команда е правилна, се извиква съответния метод, изпълняващ посоченото от потребителя, иначе се извежда съобщение за грешна команда
- тестове, проверяващи функционалността на програмата

### 3.2. Диаграми

На фигура 3.2.1 обобщено е показана функционалността на метода `void Calendar::load(string,bool);`.



Фигура 3.2.1



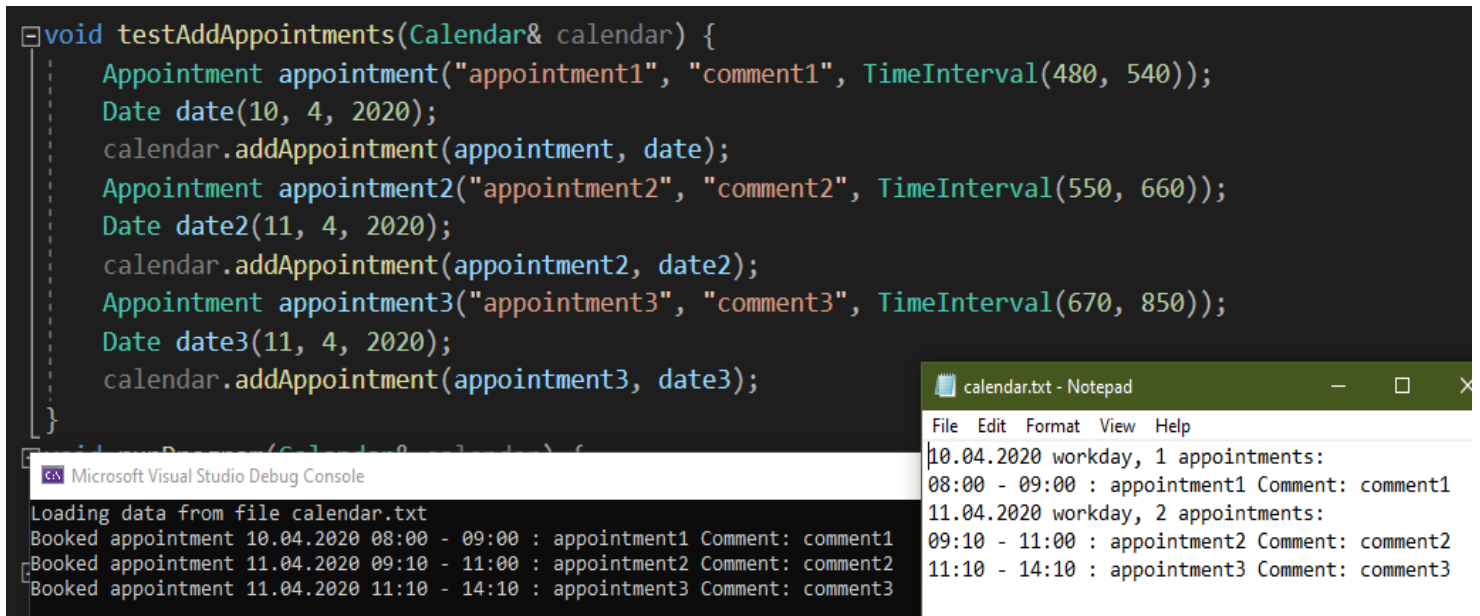
## 4. Реализация и тестване

### 4.1. Управление на паметта и алгоритми

Навсякъде паметта автоматично се заделя и унищожава от класа *vector* в стандартната библиотека. Повечето използвани алгоритми имат сложност  $O(n)$ .

### 4.2. Тестване

За проверка на правилното функциониране на програмата в *Main.cpp* са създадени съответните тест методи. Пример за такъв може да се види на фигура 4.2.1.



```
void testAddAppointments(Calendar& calendar) {
    Appointment appointment("appointment1", "comment1", TimeInterval(480, 540));
    Date date(10, 4, 2020);
    calendar.addAppointment(appointment, date);
    Appointment appointment2("appointment2", "comment2", TimeInterval(550, 660));
    Date date2(11, 4, 2020);
    calendar.addAppointment(appointment2, date2);
    Appointment appointment3("appointment3", "comment3", TimeInterval(670, 850));
    Date date3(11, 4, 2020);
    calendar.addAppointment(appointment3, date3);
}
```

Microsoft Visual Studio Debug Console

```
Loading data from file calendar.txt
Booked appointment 10.04.2020 08:00 - 09:00 : appointment1 Comment: comment1
Booked appointment 11.04.2020 09:10 - 11:00 : appointment2 Comment: comment2
Booked appointment 11.04.2020 11:10 - 14:10 : appointment3 Comment: comment3
```

calendar.txt - Notepad

```
File Edit Format View Help
10.04.2020 workday, 1 appointments:
08:00 - 09:00 : appointment1 Comment: comment1
11.04.2020 workday, 2 appointments:
09:10 - 11:00 : appointment2 Comment: comment2
11:10 - 14:10 : appointment3 Comment: comment3
```

Фигура 4.2.1

## **5. Заключение**

### **5.1. Бъдещо развитие и усъвършенстване**

Програмата може да бъде допълнително разработвана и към нея да бъдат добавяни различни функционалности – например потребителят да записва аларми за някоя среща или да праща известие за нея на даден email адрес. Към този код може да се разработи и графичен интерфейс за телефон, таблет или компютър или да бъде направен на уеб приложение, макар вече да съществуват множество такива. Възможностите са безкрайни!