

**СОФИЙСКИ УНИВЕРСИТЕТ "СВ. КЛИМЕНТ ОХРИДСКИ"**

ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА

Документация на проект

по Обектно ориентирано програмиране

НА ТЕМА:

**Проект 16: Json Parser**

Изготвил:

Кенан Юсеин

фак. № 71947

спец. Информационни системи, 1 курс

Contents

[Глава 1: Увод 3](#_Toc42971331)

[Описание и идея на проекта: 3](#_Toc42971332)

[Цел и задачи на разработката 3](#_Toc42971333)

[Глава 2: Преглед на предметната област 3](#_Toc42971334)

[Какво представлява един JSON файл 3](#_Toc42971335)

[Работа с файл 5](#_Toc42971336)

[Валидация на .json 5](#_Toc42971337)

[Parser 6](#_Toc42971338)

[Глава 3: Проектиране и Реализация 6](#_Toc42971339)

[Class Json 6](#_Toc42971340)

[Методи: 6](#_Toc42971341)

[Класове, които наследяват class Json: 8](#_Toc42971342)

[Main.cpp 8](#_Toc42971343)

[Потребителски интерфейс 8](#_Toc42971344)

[Глава 4: Тестване 9](#_Toc42971345)

[Глава 5: Заключение 9](#_Toc42971346)

# Глава 1: Увод

## Описание и идея на проекта:

[JSON](https://www.json.org/json-bg.html) е популярен текстов формат за описване на данни. Проектът трябва да работи с файлове в такъв формат.

Идеята на проекта е програма, която чете и Parse-ва един JSON файл и запаметява обектите (key-value) в паметта на програмата с цел използването им на по-късен етап.

## Цел и задачи на разработката

Цели:

* Постигане на четим код, отговарящ на принципите на ООП.
* Програмата трябва да може да отваря и чете файлове от тип .json.
* След отварянето на такъв файл, трябва да валидира файла по стандартите на json синтаксиса и максимално точно да уведоми потребителя при грешки в синтаксиса на файла.
* Ако файла е валиден, програмата трябва да Parse-не обектите от файла в паметта на програмата.
* Програмата трябва да предоставя лесен и удобен начин за работа с тези обекти: принтиране, добавяне, премахване и т.н.

# Глава 2: Преглед на предметната област

## Какво представлява един JSON файл

**JSON** (JavaScript Object Notation) е опростен формат за обмяна на данни, удобен за работа както за хората, така и за компютрите. Той е базиран на едно подмножество на езика за програмиране JavaScript.

JSON се състои от две структури:

* Колекция от двойки име/стойност. В различните езици, това се реализира като *обект*, запис, структура, речник, хеш таблица, именован списък, или асоциативен масив.\
* Подреден списък от стойности. В повечето езици, това се реализира чрез *масив*, вектор, списък или последователност.

Това са универсални структури от данни, които се поддържат от всички модерни езици под една или друга форма. Може да се предположи, че формат, подходящ за обмяна на данни между езиците за програмиране, също ще бъде базиран на тези структури.

В JSON, това изглежда по следния начин:

* Един **обект** е неподреден набор от **двойки име/стойност**. Обекта започва с {лява фигурна скоба и завършва с }дясна фигурна скоба. След всяко име се поставя :двуеточие, а двойките име/стойност се разделят със ,запетайка.



* Един ***масив*** е подредена колекция от стойности. Масивът започва с **[***лява квадратна скоба* и завършва с **]***дясна квадратна скоба*. Стойностите се разделят чрез **,***запетайка*



* Една *стойност* може да бъде *стринг* в двойни кавички, *число*, **истина (true)** или **лъжа (false)**, **null**, *обект* или *масив*. Тези структури могат да бъдат вложени.



* Един *стринг* е колекция от от нула или повече Unicode символи, обграден от двойни кавички, използвайки **\***обратна наклонена черта* като escape символ. Символът се представя чрез низ, съдържащ само един символ. Стрингът е подобен на тези в C или Java.
* *Числото* много наподобява числото в C или Java, с изключение на това, че осмичният и шестнадесетичен формат не се използват.
* Между двойките може да бъде вмъквано празно пространство (whitespace). С изключение на някои детайли относно кодирането на символите (encoding), това напълно описва езика.

## Работа с файл

След като приложението отвори даден файл, то трябва да може да извършва посочените по-долу операции, в допълнение на общите операции (open, close, save, save as, help и exit).

## Валидация на .json

За да е валиден един JSON файл той трябва да отговаря на всички изисквания относно синтаксиса описани по горе.

## Parser

Всеки обект се състои от Key – Value, като всеки Key представлява String ограден със кавички, а Value може да бъдат всякакви променливи. (Int, Double, Bool, String, Array of .., Object.)

**Основни проблеми:**

* Да се различи от какъв тип е Value за да може да се съхрани под съответна променлива.
* Трябва да може да се нестват обекти от различни типове в един контейнер, защото Value може да бъде нов обект, който да съдържа дори и други обекти.

# Глава 3: Проектиране и Реализация

След валидиране на JSON файла, програмата ще запазва целият текст прочетен от файла в един глобален std::string. Този String ще бъде подаван на променлива в class Json като се премахват всички “whitespaces” (интервали от стринга).

За реализация на класът JSON се използва STD::MAP, което ни позволява да запазваме двойки (std::pair) от key-value.

За Key се използва String, а за Value се използва Json\* като:

Класът Json се наследява от ред други класове които представляват различните варианти на тип Value.

Ще се използват виртуални функции за достъпване на типовете Value в контейнера Map.

За парсеване на променливите се използва библиотеката #include <sstream> което ни позволява да четем от стринга, който съдържа целия Json текст. std::istringstream

Различните типове Value се определят според синтаксиса на JSON.

## Class Json

## Методи:

#### virtual void print();

**Виртуална функция, която се използва при принтиране на контейнера std::map. Извикват се print() функции на класовете, които наследяват Json и се принтират съответните променливи от тези класове, които отговарят на типа Value, който е бил парсенат.**

#### void searchKey(std::string& sKey, bool & found); // this will search for matches with all keys, even nested ones in objects.

Функцията проверява в дълбочина контейнера за съответствие на търсен ключ. При откриване на такъв той се принтира на екрана. Ако няма намерени съвпадения се извежда съобщение за това.

#### void deleteKey(std::string& dKey, bool & deleted); // deep searches (nested objects aswell) for the key and deletes it if it matches.

Функцията проверява отново в дълбочина контейнера за съвпадения с търсен ключ и изтрива първия намерен такъв от контейнера**.** Извежда съобщение ако не е открит такъв.

#### void create(std::string& newElement);

След проверка за валидност на въведения key-value, функцията използва вече създадената функция parse() за да добави нов елемент в контейнера.

#### void reset();

Изтрива съдържанието на контейнера и стринга, който пази целия json текст. (нулира класа)

#### void loadNparse(const std::string & txt);

Извиква се в main функцията. Зарежда променливата от класа Json, който пази текста и извиква функцията parse() над него.

#### void parse(std::string& jsontext, std::map<std::string, Json\*>& elements);

Функцията използва std::stringstream за да чете от стринга, който пази json текста.

Чете до срещане на „(кавички) и взима под един стринг ключа от тези кавички.

След срещане на : (двоеточие) взима целия Value под нов стринг и създава нов std::stringstream, който се използва за определяне на типа Value. След определянето на типа се създава съответна променлива, която взима Value и я запаметява. След това се създава наследяващ клас чрез конструктор, който взима тази променлива създадена за Value.

Ако има нестване (обект в текущия обект) функцията Parse() се използва рекурсивно:

Взима се целия Value (в случая нов обект, до срещане на } ) и се извиква рекурсивно Parse() над този стринг, съдържащ целия нов обект.

### Класове, които наследяват class Json:

#### Class JsonInt

#### Class JsonDouble

#### Class JsonBool

#### Class JsonString

#### Class JsonIntArray

#### Class JsonDoubleArray

#### Class JsonStringArray

#### Class JsonObjectArray

Всеки клас отговаря на съответния тип, който се определя след парсеването. Класовете се създават по време на изпълнение на програмата и заместват Json\* в std::map като се достъпват чрез виртуални функции.

За Value от тип Array се използват вектори в различните наследяващи класове, които са от съответния тип.

## Main.cpp

### Потребителски интерфейс

Използват се 2 менюта в програмата.

* Menu()

Това е главното меню на програмата, чрез който се работи с файловете. Реализирани са следните функционалности:

void Open(); void Close(); void Save(); voiSaveAs(); void Help(); void exit();

За следена дали вече е отворен/зареден файл се използва string-a filenamе. При затваряне или запаметяване на файл (освобождаване на паметта на програмата) стринга се нулира, което означава, че няма отворен файл.

* UserOptions()

Това е функционалното меню на програмата, чрез който потребителят работи със склада:

void print();

void searchKey(std::string&);

void deleteKey(std::string&);

void create(std::string& newElement);

* Извикват съответните методи от класа JSON, които правят необходимите промени в контейнера на ПАрсера.

# Глава 4: Тестване

За тестване на програмата са изработени три примерни .json файла:

* example.json
* example1.json
* example2.json

{

"Name": "KENAN",

"Age": 20,

"PocketMoney": 10.50,

"Gender": true,

"Mom": {

"Name": "Barie",

"Age": 40

},

"Dad": {

"Name": "Kemal",

"Age": 47

},

"Sister": {

"Name": "Sevdjan",

"Age": 22

},

"Granparent names": [ "GP\_1\_Name", "GP\_2\_Name", "GP\_3\_Name" ],

"GP Ages": [ 70, 72, 60 ],

"Fav Hobbies": [ "Coding", "Gaming", "Volleyball" ],

"Fav Numbers": [ 16, 13, 6 ],

"Fav double numbers": [ 6.6, 4.20, 3.4 ],

"UniversitySubjects": {

"Maths": [ "DIS", "GEOMETRY", "ALGEBRA" ],

"Programming": [ "OOP", "FPR" ],

"Marks": {

"DIS1": 5,

"UP": 6

}

}

}

# Глава 5: Заключение

## GITHUB Link to project’s repository

* <https://github.com/kensevv/FMI-OOP/tree/master/JSON%20parser%20Project%2016>