Проект №16 - JSON parser – Никола Терзийски ИС гр.4 ФН\_72006

**Стартова точка**

Идеята на проекта е да се опита да представи един текстов файл, който е написан като JSON формат, като JSON обект.JSON обект представлява -> ключ със съответна стойност, чиято стойност може да бъде - масив от стойности или стойност от тип – Int, String, Bool.

Ако е валиден този обект да може да се намира валидни пътища до неговите подобекти или да се търсят ключове. Да се заменят ключове с други дадени от конзолата и да се заменя дадена стойност с друга.

**Функционалност**

Проектът започва при подаване на командата *open <path>.* При подаден невалиден път, към текстов файл, програма отчита грешка и казва на конзолата, че има грешка.

Ако пътят е валиден => програмата стартира и се извиква автоматично метод Engine, който работи с функции, свързани с отворен вече файл.

В моя проект съм направил, при валиден път, да се създаде празен **JsonNode** клас, който има функция validate().

Функцията Engine приема този JsonNode и му извиква validate(), а validate() приема std::string text, int counter.

1.std::string text – подаваме този текст, който всъщност е отворения файл, направен на един текст, за да може по-лесно да валидираме.

2.counter – това е брояч, който следи на кой индекс сме

**Valide() –** основен метод, който винаги се извиква. Направил съм алгоритъм, който проверява дали първо текстовия файл започва с { (отваряща скоба).

1. Ако не започва с { - хвърля error, който го handle-ваме и изписваме грешката.
2. Ако мине първия тест, след това търси двойни кавички, защото всеки ключ е string и трябва да започва с кавички и свършва с кавички. Ако няма кавичка – хвърля error със съответната грешка. След това търсим следващата кавичка. Ако стигне до нов ред => хвърляме грешка, защото ключа не може да бъде на 2 реда.
3. След ключа проверяваме за двуеточие “:”. Ако няма двуеточие хвърляме грешка, защото след всеки ключ има двуеточие за негото value.
4. След двуеточието следва проверка

-Ако следващия символ е кавички търсим като в първа стъпка

-Ако следващия символ е число( проверка по ASCII таблицата) => търсим докато не намерим знак различен от число или нов ред. Ако намери знак различен от число хвърля error

-Третия вариант е ако намери обект, тоест ако value-то на един ключ започва с {

Тогава правим нов обект и викаме на този обект validate(text,counter)

-Четвъртият вариант е булева стойност (true или false)

1. Последния възможен вариант е да бъде масив от value-та, като този елемент може да съдържа неограничен брой елементи, които се намират в точка 4.

Подходил съм да търся елемент по елемент. Ако намеря отваряще се скоба, извиквам на текущия node, който създавам validate(). Така на него ще направи подобекти.

Другите функции

-print – Принтира обекта json – като в зависимост от това, че всеки поделемент е с един space разстояние по-навътре.

-search <key> - Търси обект с ключ подаден от входа. Ако намери да върне този обект, чиито ключ е подадения key и да принтираме този обект. Тоест, извикваме метода и на неговия резултат даваме print().

-set <path> <string> - Подаваме път и след това string, който всъщност е JSON обект. Първо валидираме пътя. Пътя е изграден само от string-ове, Пример : “Dogs”/”Big”

Примера означава – търси ми ключ със стойност “Dogs” и след трябва неговия подобект да има също ключ “Big”. Ако няма – трябва да върне грешка.

Ако съответства на пътя => на подадения JSON string правим обект и го валидираме. След това стигаме до пътя и заместваме неговото value с подадения ни JSON обект от входа.

-create <path> <string>. Както горната функция set, но този път трябва последния елемент на входа с пътя да не съществува, за да може да създаде нов елемент. Ако съществува връща грешка. Валидираме си JSON string от входа, ако е валиден, го заместваме като value на новия ключ.

-delete <path> - Взима път, който го валидира и трие елементите намиращи се на този път

-move <from> <to> - Тази функция използва горните две. Първо взима елемента на път *from*. После трие елемента на път *from*. Накрая добавя елемента или по-точно **set** на път *to*

-save [<path>] – Записва обекта на дадения път в текущия файл, който е отворен

-saveas <file> [<path>] – Записва обекта в новия файл, който е подаден от входа

**Кога приключва**

Когато се правят промени с помощта на по-горните функции, ако не се напише save след това, никакъв прогрес няма да се запази. Просто ще правите промени локално.

Програмата затваря отворения файл с помощта на командата **close.**

За да излезнете от програмата трябва първо да затворите отворения файл (**close**) и след това **exit**.

**Подобрения**

Като цяло, в такъв тип проекти, може да се направи много добър refactoring на кода или по-точно да се увеличи скоростта, с която програмата валидира даден JSON или като цяло всички други изброени функции.

Една добра модификация е, като валидираме обекта да може да го запишем във файл с разширение .json и директно да може да се използва от програми, които използват файлове с разширение .json.

Github папка - <https://github.com/nikolaTerziiski/JsonParser/tree/master/JsonParse>