Проект по СДП – JSON парсер

Павел Григоров, СУ-ФМИ, КН, ФН 82038

Връзка към проекта - https://github.com/PKGri/DSP\_Course\_Project\_JSON\_Parser.git

1. Въведение

JSON парсерът е програма, която дава възможността да се работи с данни записани във файловия формат JSON (**J**ava**S**cript **O**bject **N**otation). В действителност тези файлове представляват текстови файлове, в който данните са записани по определена конвенция. Основната функционалност е прочитането на файла и съответно представянето им в паметта на компютъра по начин, който позволява да бъдат извършвани множество действия, включващи, но не ограничени до, извличане на конкретна информация, редактиране на някои от данните, записване на данните отново в JSON файлов формат, както и осъществяването на някои основни JSONPath заявки.

JSONPath заявките са изрази, които указват на програмата каква информация да извлече и да представи на потребителя.

Първата основна задача е прочитането на един JSON файл. За тази цел трябва да се измисли как трябва да бъде представяна в паметта текстовата информация от файла, за да могат след това да бъдат извършват операции с нея.

Следващата задача е да се определи какви точно операции да бъдат поддържани и по какъв начин да се подходи с тяхното осъществяване в кода.

Междинни цели са осъществяването на работа с команден ред, за улеснена работа на потребителя с всичките заредени данни.

Изискванията, които условието задава, са да могат да се четат JSON файлове и да се поддържат команди за избиране на даден елемент, за променяне на съдържанието му и да се записва определено съдържание.

По-долу в тази документация описвам как съм подходил в достигането си на тези цели, някои от проблемите, които са се явили в процеса на разработката и как съм се справил с тях. Ще бъдат описани всичките класове и някои от по-сложните функции, които са част от кода. Разбира се, на края на този документ се намират хипервръзки към отделните източници, които ми бяха от полза в изготвянето на проекта.

1. Преглед на предметната област

Тук ще представя и дам дефиниции на специфичните термини, които ще бъдат срещани надолу в документацията.

**Общи:**

* **Регулярен израз –** последователност от знаци, която дефинира шаблон за търсене. В кода е използвана стандартната библиотека <regex>.

**JSON специфични:**

* **Обект –** Структура, съдържаща двойки <ключ> : <елемент>
* **Масив –** Структура, съдържаща подредени елементи
* **Елемент –** обект, масив, низ, число или булева, в контекста на JSON
* **Ключ –** името на елемент.
* **Елемент –** елементът включва цялата информация от начален таг до съответния затварящ таг
* **Коренов елемент** или **корен –** първоначалният елемент. Той е обект или низ. Файлът може да съдържа само един коренов елемент.
* **Елемент-дете** или **дете –** елемент, който е вложен в друг елемент
* **Родителски елемент** или **родител –** елемент, който има поне едно дете – обекти и масиви.

**JSONPath специфични:**

* **Път –** указаният чрез *JSONPath-*израз път до елемент.
* **Възел –** всяка част от *JSONPath-*израза, която отговаря на елемент

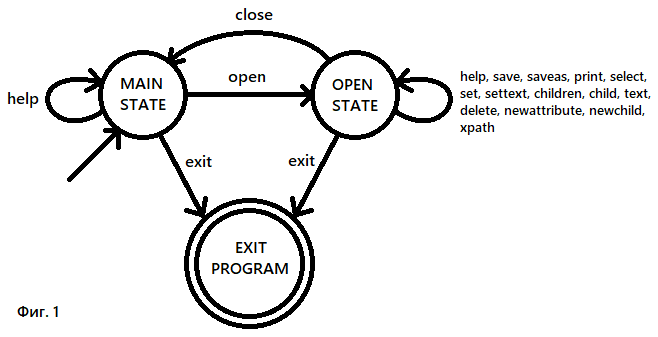
За главната цел, която е четенето на JSON файла, първото, което направих, беше да създам клас за елемент. След това създадох клас за самия файл, така че да са по-добре структурирани прочетените данни.

Написах логиката си за работа с команден ред, за да мога по-лесно да изпробвам различните операции, които трябва да извършвам, след което почнах да реализирам изискваните команди.

1. Проектиране и реализация

В програмата са създадени два класа, които да ми отговарят за различни данни:

* class JSONElement
* class JSONFile

Отделно от тях съм направил функции за работата с команден ред.

Първо ще опиша как реализирах **командния ред**. Като цяло той е базиран на цикъл *do{…}while(…)* с още един такъв цикъл вложен в него*.* Тъй като операцията *open* за четене на XML файл не може да се извърши, ако вече има отворен файл, то може да разглеждаме действието на програмата като детерминиран краен автомат с 3 състояния и преходи – отделните команди възможни в тези състояния. (фиг. 1)

За повече четливост и за да се избегнат повторения на код, използвах функции от по-висок ред. (фиг. 2)

void enterCommand(std::function<bool(std::string)> terminationPredicate,

std::function<bool(std::string)> commandList)

{

do

{

do

{

std::cout << '>';

std::cin >> command;

} while (!commandList(command));

executeCommand(command);

} while (!terminationPredicate(command));

} **Фиг. 2**

Много от функциите, особено тези изискващи JSONPath, изискват вкарването на аргументите една по една, не на куп, поради ползването на getline. Препоръчва се всяка функция да се използва внимателно с вкарване на аргументите един по един, за да се избегнат грешки.

Класът ***JSONElement*** съдържа член-данни за своите:

* тип
* низ за ключа
* указател към своя родител (ако има)
* списък от деца (ако има)
* низ за съдържание (ако е низ, число или булева)
* броя на предшествениците му

Обект от класа ***JSONFile***съдържа цялата заредена информация от един файл. Член-данните му са:

Фиг. 6

* низ за името/директорията на файла
* JSONElement представящ заредената от файл информация
* JSONElement представящ селектиран обект чрез функцията select
* указател към JSONElement, който указва един от двата гореописани елемента

Както предполага името на проекта, може би най-трудната част е извличането на информацията от файл. Множество функции се ползват за тази цел. На края на функцията *open*, която е водещата за четенето, отвореният файл се затваря и информацията остава заредена в паметта на компютъра. При изпълняване на командата *close* се освобождава паметта от заредената информация чрез функцията *clear*.

За записването на обект се използват множество функции, тъй като по условие има много различни случая, които трябва да се имат предвид.

1. JSONPath

Считам, че тази функционалност трябва да бъде отделна глава, защото трябва по-ясно да опиша как се използва.

JSONPath по принцип поддържа два оператора за наследник - **.** (точка) и **[ ]** (правоъгълни скоби).

За улеснение на работата съм избрал нотацията със скобите, защото е по-обща, тъй като скобите така или иначе се използват за достъп до пореден елемент в масив.

Ключ се оказва като е обграден с единични кавички, докато индекс в масива се оказва с цяло число, като броенето почва от 0 (нула). Не се допускат отстояния между елементи. Целият път трябва да бъде въведен наведнъж.

Нека погледнем следният JSON обект:

Ползвайки нотацията със скоби, ако искаме да достъпим втория телефонен номер, трябва да напишем:

{

  "firstName": "John",

  "age"      : 26,

  "phoneNumbers": [

    {

      "number": "0123-4567-8888",

    },

    {

      "type"  : "iPhone",

      "number": "0123-4567-8910"

    }

  ]

}

$[‘phoneNumbers’][1][‘number’]

Допуска се и да бъде пропуснат символът за главен обект $:

[‘phoneNumbers’][1][‘number’]

Ако искаме да достъпим главният обект, изписваме само символът за него:

$

Валидацията на указания JSONPath израз се извършва чрез използването на регулярни изрази.

1. Примери и тестови сценарии

В GitHub репозиторията на проекта съм прикачил и папка с разни JSON файлове, които съм ползвал за тестване на програмата си.

Връзка към репозиторията - <https://github.com/PKGri/OOP_Course_Project_XML_Parser>

1. Заключение

В изпълнението си на проекта съм предоставил начин за осъществяване на исканите операции, като съм добавил и няколко допълнителни.

В бъдеще ще преразгледам функциите, които са по-тежки за изпълнение, дали мога да ги оптимизирам повече и като цяло ще искам да разгледам какво мога да оптимизирам по проекта. JSONPath поддръжката има още много, което може да се допълва по нея, това е най-минималното; Има множество други предикати и оператори с разнообразни действия и ползи.

1. Използвана литература

<http://www.cplusplus.com/> - източник на обща информация относно езика

https://www.json.org/json-en.html - източник относно JSON

<https://support.smartbear.com/alertsite/docs/monitors/api/endpoint/jsonpath.html> - източник относно JSONPath

<https://www.youtube.com/watch?v=9K4N6MO_R1Y> – видеоурок за използването на <regex> в С++