Проект

Тема: JSON Parser

Изтовил: Семир Балджиев

Глава 1

Увод

1.1 Описание и идея на проекта

Проектът JSON Parser се основава на създаването на програма която имплементира парсване и обработка на файлове в JSON формат.

JSON е популярен формат за обмен на данни, който се използва в широк кръг от приложения. Идеята на проекта е да се разработи JSON Parser, който да може да прочете, анализира и представи структурата на JSON данни в паметта, което позволява по-лесно манипулиране и използване на информацията.

1.2 Цели и задачи на разработката

Целта на проекта JSON Parser е да предостави функционалност за парсване на JSON данни и изграждане на подходящи обекти в паметта, които представят структурата на JSON формата. Основните задачи на разработката включват:

- Разработване на архитектурата на JSON Parser, базирана на обектно-ориентиран подход.
- Имплементиране на класове и методи за четене и парсване на JSON данни.
- Обработка на различни типове JSON елементи, като обекти, масиви, числа, низове и булеви стойности.
- Обработка на вложени структури и извличане на информация от JSON обекти и масиви.

1.3 Структура на документацията

В документацията са разгледани най-важните аспекти от създаването на проекта, като: Предметна област, Проектиране ООП дизайн, Реализация и тестове

Глава 2

Преглед на предметната област

2.1 Основни дефиниции, концепции и алгоритми

JSON (JavaScript Object Notation) е текстов формат за обмен на данни, базиран на синтаксиса на JavaScript.

Парсването е процесът на анализиране и преобразуване на текст или символен низ в структурирано представяне. В случая на JSON Parser, парсването включва прочитане и анализиране на JSON данни за изграждане на подходяща структура в паметта. Като преди това се валидира JSON файла и се съобщава за решки в синтаксиса.

2.2 Дефиниране на проблеми и сложност на поставената задача

Проблемите които възникват при решаване на проблема са няколко. Първият е да се валидира JSON файла за синтактични грешки като: скоби, кавички, запетаи и още много други символи, които могат да направят файла синтактически некоректен. Допуснал съм някои улеснения при валидиране, като например не проверявам за нови редове и специални символи в стойностите на обектите.(В условието на задачата се допуска такова усленение).

След валидацията идва ред и на парсването на текста и създаване на правилна структура в паметта, която да позволи лесното използване на обекта и извършване на манипулации в последствие.

2.3 Подходи и методи за решаване на задачата

За решение на проблема с валидациата проверявам за символи които са характерни за JSON формата. Самото обхождане на файла е до накаква степен рекурсивно, зашото в масивите и обектите може да имаме произволни типове от всички при което най-често се образува вложена йерархия.

За създаване на структурата и съхраняването и в паметта използвам шаблон за дизайм Фабрика (Factory) чрез който създавам необходимите обекти от заредения файл.

Глава 3

Проектиране

3.1 Обща архитектура ООП дизайн

При проектирането на JSON Parser се използва обектно-ориентиран (ООП) дизайн, който цели да предостави модулна и разширяема архитектура. ООП подходът използва обекти, които представляват конкретни елементи от системата и имат свойства и методи за манипулация с тези данни.

В изгращдането на архитектурата са използвани добри ооп практики и парадигми които включват полиморфизъм, наследяванем, абстракция, енкапсулация.

В рамките на общата архитектура на JSON Parser се разглеждат следните ключови елементи:

- JsonParser: Това е основният клас, отговарящ за парсването и обработката на JSON данни. Той включва методи за четене на JSON низ, анализиране на синтаксиса и изграждане на подходяща структура от обекти, представящи JSON елементите.
- JsonObject: Този клас представя JSON обектите и съдържа свойства и методи за достъп и манипулация на данните в обекта. Той може да съдържа вложени обекти или масиви и предоставя методи за извличане на стойности по ключове.
- JsonArray: Този клас представя JSON масивите и предлага методи за достъп и манипулация на елементите в масива. Той може да съдържа различни типове данни като числа, низове, булеви стойности и други.
- JsonType: Абстрактен клас, който служи като базов клас за представяне на стойностите в JSON формат. Той се наследява от класовете, представящи конкретните типове данни като числа, низове и булеви стойности тн.
- KVPair: Това е клас който се съдържа в обекта, тоест елементите на всеки обект са колекция от двойка ключ стойност. Като в него има имлементирани медоти за работа с такава структура.

Глава 4

Реализация и тестове

4.1 Реализация на класовете

Базовият клас за типовете - JsonType

Наследници:

- JsonObejct
- JsonArray
- JsonNumber
- JsonString
- JsonBool
- JsonNull

KVPair - клас описващ двойка ключ стойност

Допълнителни класове:

- JsonValidator - грижи се за логиката по валидирането на файла спрямо JSON формата
- JsonParser - прочитане и парсване на валиден файл
- JsonFactory - създаване на йерархичната структура от обекти
- CustomJsonExceptions - клас в който са дефинирани *custom* изключения за по-описателни съобщения при грешки
- Json - описва JSON обект като в него се пази референция към най-главния обект в йерархията и през него се манипулират останалите.
- JsonApp - Клас който се грижи за изпълнението на командите описани в условието на проекта.

4.2 Управление на памет и алгоритми

Валидациатя и парсването съм ги направил рекурсивно за обеката и масива тъй като в тях може да има всякакъв ти елементи от JSON формата. Като ако е необходимо се викат методите за парсване и валидация на останалите обекти в тези на обекта и масива.

4.3. Планиране, описание и създаване на тестови сценарии

В проекта съм създал тестови файлове които да се подават на програмата при стартирането и.

Осовните команди които се подържат за работа със създадената структура в паметта са:

- open <filePath> - отваря файла на подадения път ако съществува такъв
- validate - валидира файла за коректост
- search <key> - търси в структурата елементи с такъв ключ ако има повече елементи връща масив от всички.
- contains <value> - Проверява дали *value* се съдържа в стойността на някой елемент.
- set <path> <string> - Променя стойноста на зададения елемент по оодадения път.
- create <path> <string> - Създава елемент на подадения път.
- delete <path> - Изтрива елемента на подадения път.
- move <fromPath> <toPath> - Премества стойноста от единия път на другия.
- save [<path>] - Записва структурата във файл ако не е подаден път, ако е записва елемента на пътя.
- saveas <filePath> [<path>] - Записва цялата структура в нов файл ако не е подаден път, иначе записва стойноста на пътя в новия файл.

Глава 5

Заключение

5.1. Обобщение на изпълнението на началните цели.

Всички задължителни изисквания от условието на проекта и начлно поствени цели от мен са покрити.

5.2. Насоки за бъдещо развитие и усъвършенстване.

За бъдеще може да се направи командите да са в отделни класове във някаква йерархия от наследяване на общ базов клас за комада и да се прегледа по-обстойно кода за гранични случаи.

Използвана литература

https://www.json.org/json-en.html - официален сайт на JSON стандарта.

https://cplusplus.com/ - Докуметация за C++

Всякаква литература от курса по ООП като лекции, презентации, примерен код