**Документация**

**на проект XML Parser**

Изготвена от Тодор Тонев Димитров 81969

1. **Увод**

Гитхъб репо на проекта: <https://github.com/todorakisa/XML-Parser>

* 1. **Описание и идея на проекта**

Идеята на проекта е да се направи приложение, което да може да отваря *xml* файлове и да ги обработва по определен начин. След като са извършени дадени операции по файла, приложението трябва да може да ги запази в оригиналния файл или нов такъв по избор на потребителя.

* 1. **Цел и задачи на разработката**

Целта на приложението е правилно да чете информацията от даден *xml* файл и да я пази в подходяща структора. На тази стуктора трябва да се извършват дадени операции които я променят и след това трябва да може всичката тази инфорамция да се запази в дадения първоначално файл. В това приложение остновните задачи са: Да се измисли подходяща структора в която да пазим данните(*xml* елементи) от *xml* файла; Да се измисли подходящ начин по който да се търси *xml* елемент в структурата; Да се измисли начин по който може да променяме даден елемент; Да се измисли начин по който да се запазва информацията в същия файл. Трябва и да се направят редица команди с които потребителя да модифицира данните, да отваря файлове и да запазва файлове. Ако потребителя не запази модифицирания файл то програмата не трябва да променя оригиналния файл.

* 1. **Структура на документацията**

В документацията има пет основни глави: 1 Увод 2 Преглед на предметната област 3 Проектиране 4 Реализация и тестване и 5 Заключение. Всяка глава си има различни аспекти за разглеждане на проекта.

1. **Преглед на предметната област**
   1. **Основни дефиниции, концепции и алгоритми, които ще бъдат използвани**

За да може да запазваме инфорамцията от *xml* файл трябва да се направи дървовидна структора за запазване на xml елементите поради самото естество на XML. В даден файл от тозитип има един главен елемент и всички останали са негови деца. Това прави едно кореново дърво от елементи като първия елемент във файла е неговия корен. Също така за да може да се обходи подобен тип структура то трябва да се използва рекусрсия. Това означава че за да работим с дървото, търсенето в него, принтирането в конзолата и самото му запазване в даден файл трябва да е рекурсивно. Също така за да може правилно да вземем всичката информация от даден файл трябва да се измисли алгоритъм който да чете от *xml* файл.

* 1. **Дефиниране на проблеми и сложност на поставената задача**

Има няколко основни проблема в задачата и те са:

1. **Четенето от *xml* файла.**

Поради факта че когато се отвори един файл се чете подред всеки ред или символ се създава проблем. Самата дървовидна структора изисква да се добавят наследници на праилния елемент, а когато се чете ред по ред не винаги наследниците ще са на един и същи елемент.

1. **Търсене в дървовидна структура**

Заради самия вид на сруктурата традиционното търсене неможе да свърши работа когато търсим определен елемент.

1. **Запазване на промените във оригиналния файл**

Тук отново, заради вида на структората, традиционния запис във файл няма да свърши работа.

* 1. **Подходи, методи за решаване на поставените проблемите**

За да решим основните проблеми трябва да изпълним няколко неща. За да можем да търсим даден елемент в дървото или да гозаписваме в даден файл ще използваме рекурсия. За четенето от файла ще трябва да се използва така наречения контейнер Stack. Чрез използването на този контейнер ще може правилно да се добавят децата на основния елемент във файла. Също така когато се запазват промените в оригиналния файл програмата ще го изтрива и ще прави нов със същото име, понеже цялата информация от файла вече е запазена в програмата и няма загуба на информация от изтриването.

* 1. **Потребителски и качествени изисквания**

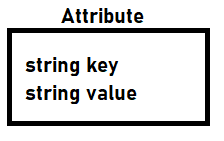
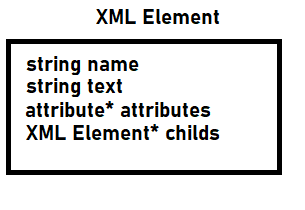
За да може потребителя да работи с програмата трябва да се реализират команди с които дой да управлява какво да се случи с файла. Също така когато се отвори даден *xml* трябва на всеки елемент да се даде уникално *id ,*а ако той има такова да се промени ако не е уникално. Това се прави с цел по добро търсене в елементите. За да може да не се получава объркване с командите, потребителя не трябва да може да ги използва ако не е отворен даден файл и съответно неможе да отваря друг файл преди да е затворил предишния.

1. **Проектиране**
   1. **Обща архитектура – ООП дизайн**

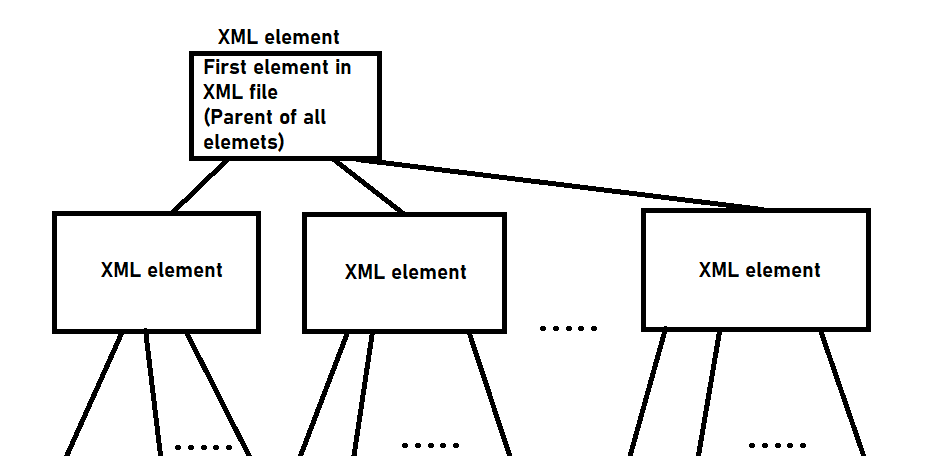
За да може да се запази цялата информация от файла, има един основен *xml* елемент. За целта трява да се реализира клас *xml* елемент(фиг. 3.2.2). По стандарт *xml* файловете имат един *parent* елемент. Затова и всички останали елементи са негови деца. Така в програмата ще имаме един елемент който съдържа в себе си всички останали елементи като негови деца(фиг. 3.2.3). Също така за да може да запазваме атрибутите на елементите, всеки елемент ще трябва да има масив от атрибути. За целта тряба да бъде имплементиран клас атрибут който да съдържа в себе си *key* и *value* като стойности(фиг. 3.2.1). За да се реализира *xpath* функционалността трябва да се направи клас, който да съдържа основните функции на *xpath* и той да има метод *execute* по този начи ще подаваме на класа дадената *xpath* команда и след това той с даден алгоритъм ще я изпълни.

За да може да чете правилно елементите от файла програмата използва Stack. Stack използва така наречената концепция LIFO (Last input First output) (фиг. 3.2.4). По този начин правилно се добавят децата на дадения елемент. Първия елемент се слага с Stack и след това всеки следващ се слага. Ако програмата срещне затварящ таг на xml елемет то той се маха от Stack и се добавя като дете на предишния(фиг. 3.2.5).

* 1. **Диаграми**

**** 

Фиг. 3.2.1 Клас атрибут Фиг. 3.2.2 Клас XML Element



Фиг. 3.2.3 Кореново дърво получаващо се от елементите на *xml* файла



Фиг. 3.2.4 Пример за добавяне и изваждане от Stack



Фиг. 3.2.5 Част от алгоритъм за четене от файл, където е използван вектор “child\_order” като Stack за да може правилно да се записват елементите.

1. **Реализация и тестване**
   1. **Реализация на класове**

Реализирани са класовете: fileXML, elementXMl, attribute, xpath. Attribute съдържа цялата информация за един атрибут. ElementXML(фиг. 4.1.1) съдържа в себе си име, текст, деца(които също са клас ElementXML ) и атрибути. Той има няколко важни метода в себе си. Освен основните конструктори(копиращ и default) и деструктор и оператор =, той съдържа рекурсивен метод принт(фиг. 4.1.2). Има и редица методи за работа с атрибути и id. Също така има и метод за добавяне на деца.



Фиг.4.1.1 Клас елемент



Фиг. 4.1.2 Рекурсивен метод за принтиране в конзолата.



Фиг. 4.1.3 Методи за атрибути.

fileXML е класът който се занимава със файла. В него са и имплементирани и решенията на основните проблеми изброени в по-горна глава. Един от най-важните му методи е fill\_content(). Този метод отваря файла и с помощта на Stack започва да чете отварящите и затваеящи тагове. За да може програмата да чете лошо написани, новалидни *xml* файлове, четенето се извършва симнвол по символ. Така ще можем да избегнем много whitespace които пречат на четенето. Един от ключовите моменти е запазватето във файла. То се случва рекурсивно с метода write\_in\_file(фиг. 4.1.4).



Фиг. 4.1.4 Рекурсивно писане във файл.

В главния main метод на програмата са реализирани всички команди с които юзъра да модифицира съдържанието на даден файл.

* 1. **Управление на паметта и алгоритми. Оптимизации.**

Всеки от класовете реализирани в проекта има съответен деструктор който да се погриже за заделената памент по време на изпълнение на програмата. По време на изпълнение на един от най-големите алгоритми, а именно четенето от файла, се заделя най-много памет. За да се оптимизира, накрая на изпълнението на алгоритъма се изтрива масива ненужните елементи и се запазва само най-първия елемент а именно родителя на всички елементи. Той съдържа в себе си всички елементи и затова няма нужда да пазим останалите елементи в масив.

* 1. **Планиране, описание и създаване на тестови сценарии**

Проекта е разработван на Linux и заради това предимство е използвана програма наречена “valgrind” която следи за неосвободена памет в програмата. След тестване на проложението с нея са премахнати всички “memory leaks”.

За да се тества програмата напълно, то трябва да се тества дали раздава правилно id на всички елементи. В репото на проекта е включен файл Example.xml, с който е тествано дали раздава правилно. Има и много други тестове, но понеже четенето е направено символ по символ повечето лошо написан, но валиден xml се прочита правилно от програмата.

1. **Заключение**
   1. **Обобщение на изпълнението на началните цели**

Изпълнени са основните цели на проекта, както и са разрешени основните проблеми при изпълнавянето на задачите които цели да изпълни това приложение.

* 1. **Насоки за бъдещо развитие и усъвършенстване**

Като бъдещо развитие може да се счита развиването на xpath заявките. В момента изпълнява само три от основните. Има много други които могат да бъдат имплементирани в проекта