Курсова работа

Тема: XML Parser

Изготвена от: Владислав Иванов Георгиев

фак. номер:45699

Увод

1.1 Описание и идея на проект

Целта на проекта е да се създаде програма, реализираща чатене, обработка и работа върху XML файлове. За работа с програмата трябва да се използва команден ред като бъдат предоставено възможнжстите за отваряне, редактиране и записване на XML файлове.

1.2 Цел и задачи на разработка

Програмата трябва да може да зарежда съдържанието на XML файл и да го трансформира в обекти позволяващи лесно добавяне, премахване и редактиране на XML елементи, чрез команди въведени от командния ред.
Поддържаните от програмата характеристики на XML елементите трябва да са:

- идентификатор на елемента
- списък от атрибути и стойности
- списък от вложени XML елементи или текст

Идентификаторите на елементите трябва да бъдат уникални за всеки един елемент във файла. Във входния файл за идентификатор се счита атрибута “id”. Ако XML елемент има атрибут “id” и стойността му е уникална за файла стойността остава непроменена. Ако обаче стойността на този атрибут не е уникална за файла към стойността се конкатерина низ, който да го направи уникален. За всеки елемент от входния файл, който няма посочен атрибут “id” трябва да бъде генериран от програмата (отново уникален).

Главния набор от операции, които трябва да се поддържат от програмата са за вход/изход: open, close, save, save as, help и exit. Описанието на тези функции е тривиално. Тук важното е да се подчертае че **нито една друга команда не може да бъде извикана преди да бъде отворен файл с командата open.**Освен тези команди трябва да бъдат поддържани и следните команди за работа върху съдържанието на XML файла:

- print - Извежда на командния ред прочетената информация от XML файла като извеждането трябва да бъде форматирано в класическия вид на XML документи.
- select <id> <key> - извежда стойността на атрибут с ключ <key> на елемент с идентификатор <id>
- set <id> <key> <value> - присвоява се стойност <value> на атрибута със стойност <key> на елемент със идентификатор <id>
- children <id>- извежда се списък с атрибути на вложените елементи на елемент с идентификатор <id>
- child <id> <n> - връща информация за <n>-тия наследник на елемент с идентификатор <id>
- text <id>- връща текста на елемент с идентификатор <id>
- delete <id> <key> - изтрива атрибут с ключ <key> на елемент идентификатор <id>
- xpath <id> <XPath> - изпълнява XPath заявка на елемент с идентификатор <id>

Минималните XPath оператори които трябва да се поддържат са:

- / - https://www.w3.org/TR/xpath20/#dt-path-expression
- [] - https://www.w3.org/TR/xpath20/#doc-xpath-Predicate
- @ - https://www.w3.org/TR/xpath20/#doc-xpath-AbbrevForwardStep
- опреаратори за сравнание

1.3 Структура на документацията

- Глава 1 - Увод в проекта
- Глава 2 - Преглед на предметната област на проекта
- Глава 3 - Проектиране на отделните компоненти на проекта
- Глава 4 - Начин на реализация и тестване
- Глава 5 - Заключение и бъдещи насоки

Преглед на предметната област

2.1 Основни концепции и алгоритми.

За решаване на задачата са използвани стандартни програмни структури в езика C++ като класове и вектори, символни низове и функционалности като входни/изходни операции, наследяване и полиморфизъм. Дървовидната структура на XML файловете налага използването на често срещани техники за обхождане на такива структури като рекурсивно търсене.

2.2 Проблеми, със които решението на задачата трябва да се справи

Най-важните задачи в проекта са:

- Алгоритъм за обработка на XML файловете, който да преобразува файла в обекти, с които останалата част от програмата да може да работи
- Създаване на командния интерфейс, който да бъде лесен за ползване от останалата част от програмата
- Структурата от xml елементи да бъде лесна за обхождане и търсене в нея
- Потребителя на програмата да има ясна представа със случващото се със програмата и структурата на обекта XML документа, който обработва

2.3 Подходи, методи за решаването

Работата на алгоритъма за обработка на XML файлове създава дърво като резултат от изпълнението се връща обект, който съдържа корена на дървото. Освен това той съдържа няколко помощни функции за достъпване на XML елементите и техните атрибути както принтирането “красиво” на XML елементите в даден изходен поток. По този начин всяка една операция се нуждае само от обекта, за да извърши нужната функция върху него. За реализацията на командния интерфейс е използван модела Command pattern. Този модел позволява лесно добавяне на операции без да се усложнява логиката, на останалата част от програмата. По този начин всяка операция се грижи за това какво вижда потребителя при изпълнението и - останалата част от логиката няма нужда да се грижи за това.

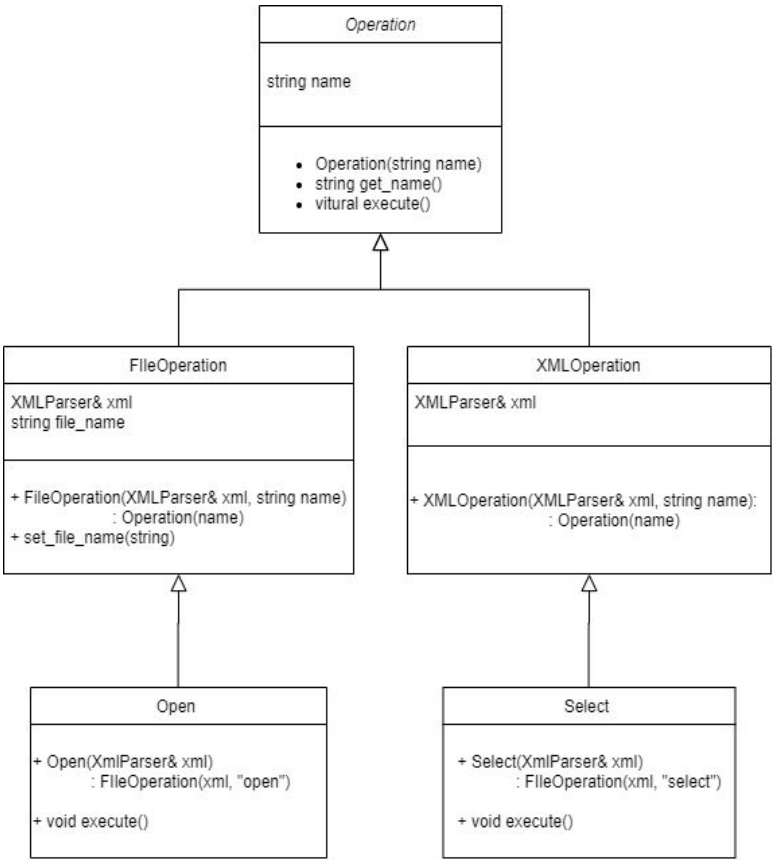
Проектиране

3.1 Обща архитектура реализация на проекта

Главните конструкции в проекта са следните класове:

- XMLParser
- Operation

Първия се грижи за четенето, обработката и достъп на XML съдържанието, а втория е виртуален клас със две функции - едната за задаване на име на операцията и една за виртуална функция, която всяка операция трябва да имплементира. Конструктора на Operation приема референция към XMParser върху, който да извършва операциите. Този подход има своите недостатъци тъй като трябва да се гарантира че “живота” на XMLParser трябва да е по-дълъг от този на Операциите закачени за него. Това е постигнато като XMLParser съдържа вектор със *pointers* сочещи към инстанции на операции. След това в деструктора на XMLParser тези инстанции биват изтрити. По-този начин е гарантирано, че инстанцията на всяка една операция за даден XMLParser няма да “надживее” класа, към които държат референция. Това също улеснява начина на намиране на операции, при четенето но името им от стандартния вход. Класа Operation е наследен от два други класа - FileOperation и XMLOperation. Първият държи информация за в момента отворения файл, а втория за XML класа върху, който да извършва операциите по редактиране и селектиране на XML елементи. Диаграмата по-долу показва начина на наследяване и работа на опрециите:



Отделните XML елементи се репрезентират от класа Node. Той съдържа следните характеристики:

- Идентификатор на елемента
- Колекция от атрибути
- Колекция от вложени XML елементи (“деца”)

Освен това като член-функции той съдържа функции за рекурсивно търсене в XML документа, достъп до даден XML елемент, редактирането му и принтирането на XML документа в даден изходен поток. Рекурсивното търсене се осъществява по идентификатора на елемент по следния алгоритъм:

```
Node* const& XMLParser::recursive_search(Node* const& n, std::string id) {
    if (n->get_id() == id) {
        return n;
    }
    if (!n->has_children()) {
        return nullptr;
    }
    std::vector<Node* > children = n->get_children();
    for (std::vector<Node*>::iterator it = children.begin(); it != children.end(); ++it) {
        Node* temp = recursive_search((*it), id);

        if (temp != nullptr) {
            return temp;
        }
    }
    return nullptr;
}
```

фиг 3.1.1 Имплементация на рекурсивно търсене в XML елементи

По този начин всяка една операция може да намери XML обекта върху, който да се изпълни. Друг важен елемент от програмата е генерирането на уникални идентификатори за весик елемент. Тази процедура се изпълнява след като целия XML документ и прочетен и обработен. Като отново се използва рекурсивен алгоритъм:

```
bool XMLParser::id_exists(std::string const &id, Node* node) {

    Node* search = recursive_search(parent_node, id);

    return search != nullptr && node != search;
}

void XMLParser::generate_ids(Node* node) {

    if (node->get_id() == "") {
        while (id_exists(std::to_string(current_id), node)) {
            current_id++;
        }
        node->set_id(std::to_string(current_id));
    }
    else if(id_exists(node->get_id(), node)) {
        int id_suffix = 1;

        while (id_exists(node->get_id() + "_" + std::to_string(id_suffix), node)) {
            id_suffix++;
        }
        node->set_id(node->get_id() + "_" + std::to_string(id_suffix));
    }

    if (!node->has_children()) {
        return;
    }

    std::vector<Node*> children = node->get_children();

    for (std::vector<Node*>::iterator it = children.begin(); it != children.end(); ++it) {
        generate_ids((*it));
    }


}
```

фиг 3.1.2 Имплементация на алгоритъма за генериране на уникални идентификатори

Важно е да се отбележи, че отделните Nodes и децата им, както и операциите се запазват динамично в XMLParser класа. За Node елементите това се налага, за да могат операциите да се изпълнят върху точно върху определен Node и да не се налага премахване и презаписване на Nodes. За операциите това е полезно, защото така ползването на операции става без да се налага инициализиране на обекти, нуждата да се знае от кой подвид(XMLOperation или FileOperation) е дадената операция както и “замърсяване” на външни контексти.

Всичко това допринася за лесното имплементиране на main функцията на програмата.

Заключение

4.1 Обобщение на изпълнените задачи

Програмата реализира обработка, редакция и записване на XML документи. Всички функционалности от условието са реализирани и изискванията към решението са спазени.