Университет ИТМО

Факультет программной инженерии и компьютерной техники

**Лабораторная работа №2**

по «Низкоуровневое программирование»

Вариант XPath

Выполнил:

Студент группы P33301

Акимов Роман Иванович

Преподаватель:

Кореньков Юрий Дмитриевич

Санкт-Петербург

2023

Задачи

Использовать средство синтаксического анализа по выбору, реализовать модуль для разбора некоторого достаточного подмножества языка запросов по выбору в соответствии с вариантом формы данных. Должна быть обеспечена возможность описания команд создания, выборки, модификации и удаления элементов данных.

Порядок выполнения:

1) Изучить выбранное средство синтаксического анализа

a. Средство должно поддерживать программный интерфейс совместимый с языком С

b. Средство должно параметризоваться спецификацией, описывающий синтаксическую структуру разбираемого языка

c. Средство может функционировать посредством кодогенерации и/или подключения

необходимых для его работы дополнительных библиотек

d. Средство может быть реализовано с нуля, в этом случае оно должно быть основано на

обобщённом алгоритме, управляемом спецификацией

2) Изучить синтаксис языка запросов и записать спецификацию для средства синтаксического анализа

a. При необходимости добавления новых конструкций в язык, добавить нужные синтаксические конструкции в спецификацию (например, сравнения в GraphQL)

b. Язык запросов должен поддерживать возможность описания следующих конструкций:

порождение нового элемента данных, выборка, обновление и удаление существующих

элементов данных по условию

• Условия

* На равенство и неравенство для чисел, строк и булевских значений
* На строгие и нестрогие сравнения для чисел
* Существование подстроки

• Логическую комбинацию произвольного количества условий и булевских значений

• В качестве любого аргумента условий могут выступать литеральные значения

(константы) или ссылки на значения, ассоциированные с элементами данных

(поля, атрибуты, свойства)

• Разрешение отношений между элементами модели данных любых условий над

сопрягаемыми элементами данных

• Поддержка арифметических операций и конкатенации строк не обязательна

c. Разрешается разработать свой язык запросов с нуля, в этом случае необходимо показать

отличие основных конструкций от остальных вариантов (за исключением типичных выражений

типа инфиксных операторов сравнения)

3) Реализовать модуль, использующий средство синтаксического анализа для разбора языка запросов

a. Программный интерфейс модуля должен принимать строку с текстом запроса и возвращать

структуру, описывающую дерево разбора запроса или сообщение о синтаксической ошибке

b. Результат работы модуля должен содержать иерархическое представление условий и других выражений, логически представляющие собой иерархически организованные данные, даже если на уровне средства синтаксического анализа для их разбора было использовано линейное представление

4 Реализовать тестовую программу для демонстрации работоспособности созданного модуля,

принимающую на стандартный ввод текст запроса и выводящую на стандартный вывод

результирующее дерево разбора или сообщение об ошибке

5 Результаты тестирования представить в виде отчёта, в который включить:

a. В части 3 привести описание структур данных, представляющих результат разбора запроса

b. В части 4 описать, какая дополнительная обработка потребовалась для результата разбора, представляемого средством синтаксического анализа, чтобы сформировать результат работы созданного модуля

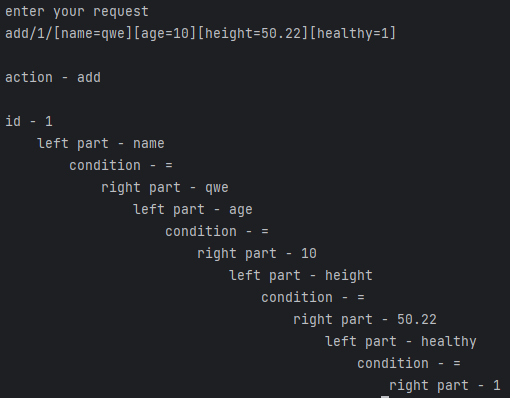
c. В части 5 привести примеры запросов для всех возможностей из п.2.b и результирующий вывод тестовой программы, оценить использование разработанным модулем оперативной памяти

Ход работы

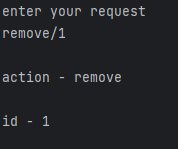
Модуль parser отвечает за парсинг запроса и упаковку в структуру request. Модуль view отвечает за отображение этой структуры в консоль.

Пример работы программы

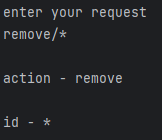
Добавление элемента



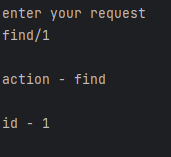
Удаление элемента по id

~~~~

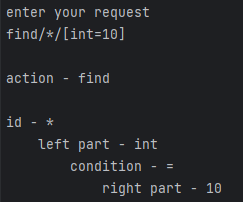
Удалить все элементы



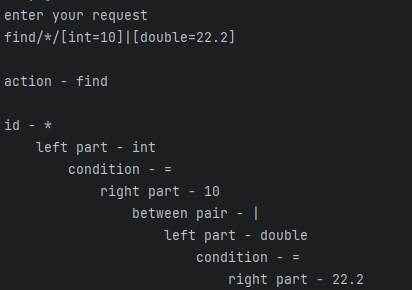
Поиск элемента по id

~~~~

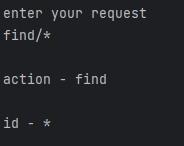
Поиск всех элементов, удовлетворяющих условию

~~~~

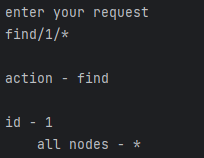
Поиск элементов, удовлетворяющих условию с булевым объединением

~~~~

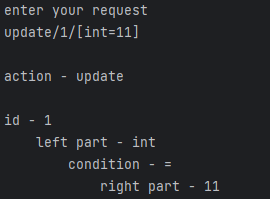
Поиск всех элементов

~~~~

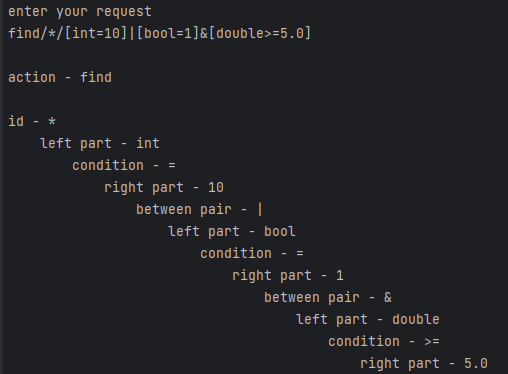
Поиск всех элементов, родитель id которого равен

~~~~

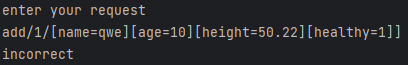
Обновление элемента по id

~~~~

Поиск элемента, удовлетворяющего условиям

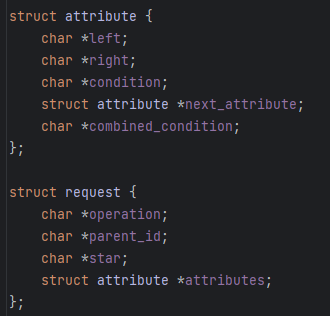


Пример некорректного запроса



Аспекты реализации

Структура request:

~~~~

В этой структуре хранятся операция (add, remove, find, update), id родителя, маркер для поиска всех элементов, а также атрибуты. Атрибуты представлены односторонним списком для удобного переключения между ними. Сама структура attribute хранит в себе левую часть выражения, то есть condition, и правую часть выражения name = roma -> name (левая) = (condition) roma (правая). Также она хранит булевое сплетение выражений (| или &).

Операции:

* add - добавление
* remove - удаление
* find - поиск
* update – обновление

Отношения между атрибутами и булевы знаки:

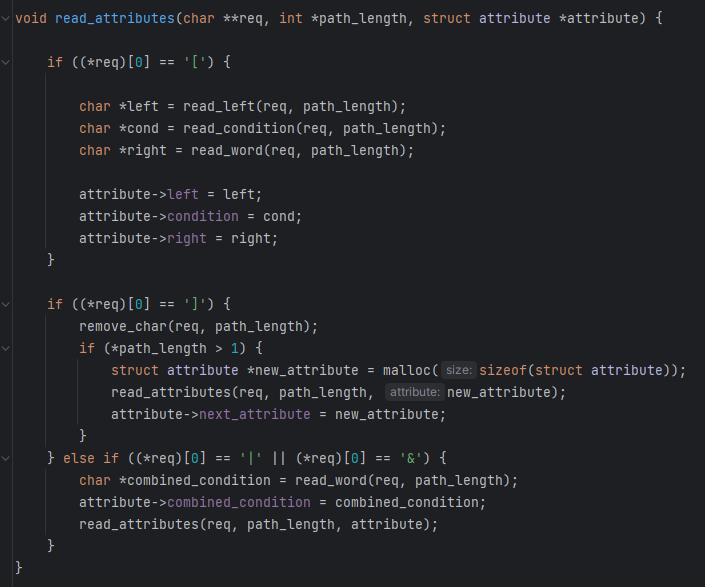
* = - равно
* != - не равно
* > - больше
* < - меньше
* >= - больше или равно
* <= - меньше или равно
* & - логическое И
* | - логическое ИЛИ
* \* - маркер «все элементы»

Парсер примитивный. Он посимвольно идет по строке и заполняет структуру request.

Посмотрим на реализацию парсера:

~~~~

Рекурсивно заполняем список атрибутов.

~~~~

Вывод:

В процессе выполнения работы выяснилось, что оперативная память расходуется только на хранение структуры request. Я написал модули parser и view, а также протестировал написанную программу.