Университет ИТМО

Факультет программной инженерии и компьютерной техники

Низкоуровневое программирование

Лабораторная работа №2

Выполнил:

Студент группы P33302

Гониченко Н.И.

Преподаватель:
Кореньков Ю.Д.

Санкт-Петербург

2023

Задание:

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.

Вариант: 1 XPath

Ход работы:

1. Изучить выбранное средство синтаксического анализа
2. Изучить синтаксис языка запросов и записать спецификацию для средства синтаксического анализа
3. Реализовать модуль, использующий средство синтаксического анализа для разбора языка запросов
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

Аспекты реализации:

Варианты запроса:

- + добавление элемента
- - удаление элемента
- ? поиск элемента
- = изменение элемента

Выбор элемента:

- все элементы на данном уровне
- / следующий оператор
- <tuple_name> узел с данным именем

Предикаты операторов:

- > больше
- < меньше
- = равно
- >= больше или равно
- <= меньше или равно
- ! инвертировать

Для объединения атрибутов:

- | или
- & и

Описание работы программы:

Программа принимает на вход запрос из стандартного потока ввода (stdin), затем передаёт строку в модуль парсинга (parser.c). В модуле парсинга происходит заполнение структуры struct request (все структуры хранятся в model.h).
После этого происходит вывод с помощью модуля print.c.

Описание структуры данных:

Запрос – хранит в себе операцию и список узлов

```
struct request {
        char operation;
        struct tuple* tuple_list;
};
```

Узел – хранит в себе имя узла, список атрибутов и ссылку на следующий узел

```
struct tuple {
        char* name;
        struct attribute* attribute_list;
        struct tuple* next;
};
```

Атрибут – хранит в себе имя поля, значение поля, условие выборки, условие соединения (может быть null), ссылку на связанный атрибут (может быть null), ссылку на следующий атрибут

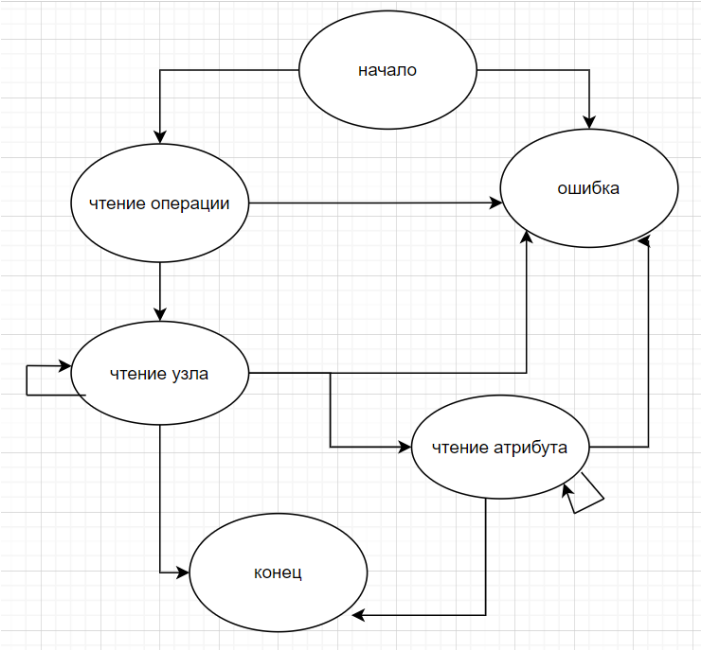
```
struct attribute {
        char* field;
        char* value;
        enum attribute_condition condition;
        enum bool_condition bool_condition;
        struct attribute* composite_attribute;
        struct attribute* next;
};
```

Используемые перечисления:

```
enum attribute_condition {
        MORE = '>',
        LESS = '<',
        EQUAL = '=',
        NEG = '!',
        GOQ = GREATER_OR_EQUAL,
        LOQ = LESS_OR_EQUAL
};

enum bool_condition {
        AND = '&',
        OR = '|',
        NOT = '!'
};
```

Выбор синтаксического средства: я решил написать собственный парсер, потому что мне показалось, что так будет проще и быстрее. Для реализации был придуман следующий конечный автомат:



Оценка используемой оперативной памяти: оперативная память используется только для хранения структуры запроса (struct request)

Примеры запросов:

Запрос с двумя иерархическими уровнями
Первый уровень – root (Deep 1)

Второй уровень – subject (Deep 2)

```
Enter request:
+/root/subject[name="subject"&hour>1]

Add operation

Deep: 1
Where tuple name = root

Deep: 2
Where tuple name = subject
Field name: name
Condition: = : equally
Value: "subject"
Condition: &
Field name: hour
Condition: > : more
Value: 1
```

Запрос с двумя иерархическими уровнями
Первый уровень – любой элемент (Deep 1)
Второй уровень – computer (Deep 2)

```
Enter request:
?/*/computer[name="HP"&year<2023]

Get operation

Deep: 1
* - For all tuples

Deep: 2
Where tuple name = computer
Field name: name
Condition: = : equally
Value: "HP"
Condition: &
Field name: year
Condition: < : less
Value: 2023
```

Некорректный запрос:

```
Enter request:
()
Invalid request
```

Выводы:

В ходе выполнения данной лабораторной работы я реализовал синтаксический анализатор для запросов XPath (в немного изменённом виде). Как писалось выше, я реализовал свой парсер запросов, потому что посчитал, что так будет быстрее и удобнее. Иерархическая структура показана в «Примеры запросов» в первых двух примерах.