Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №2

по «Низкоуровневому программированию»

Выполнил:

Студент группы P33302
Верзаков А.Ю.

Преподаватель:
Кореньков Ю.Д.

Санкт-Петербург

2023

Вариант: 7 (Mongo shell)

Задание:

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 разработанного модуля
5. Представить результаты тестирования в отчёте

Для реализации модуля использовались программы lex и yacc. Эти программы часто используются вместе для создания синтаксических анализаторов языков.

Основные доступные операции над элементами:

- receive – получить выборку элементов по условию
- insert – вставить элемент
- update – обновить существующий элемент
- delete – удалить существующий элемент

Условные операторы:

- lo_t – (**lo**wer **t**han) – меньше чем
- lo_eq_t – (**lo**wer **eq**uals **t**hen) – меньше или равно чем
- gr_t – (**gr**eater **t**han) – больше чем
- gr_eq_t – (**gr**eater **eq**uals **t**han) – больше или равно чем
- no_eq – (**no eq**uals) – не равно

Все условные операторы начинаются со знака $.

Строка запроса помещается в файл query.mso, результирующее дерево сохраняется в глобальную переменную query и распечатывается в поток вывода.

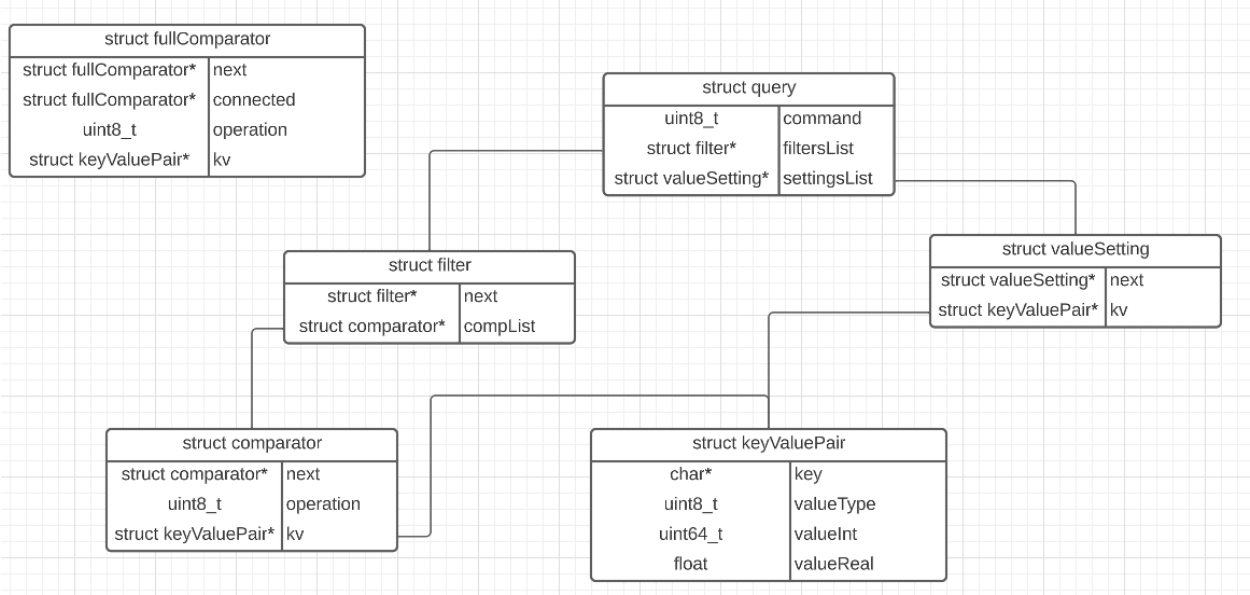
Правила lex (пример):

```
"receive"    return RECEIVE;
"insert"     return INSERT;
"delete"     return DELETE;
"update"     return UPDATE;
```

Правила yacc:

```
mongosh: STORAGE RECEIVE OP_BRACE OP_C_BRACE filters CL_C_BRACE CL_BRACE {setCommand(0);}
|
STORAGE DELETE OP_BRACE OP_C_BRACE filters CL_C_BRACE CL_BRACE {setCommand(1);}
|
STORAGE INSERT OP_BRACE parent_def COMMA vals_def CL_C_BRACE {setCommand(2);}
|
STORAGE UPDATE OP_BRACE OP_C_BRACE filters CL_C_BRACE COMMA DOLLAR SET COLON vals_def CL_BRACE {setCommand(3);};
```

Структура результирующего дерева запроса:



Заполнением структурой запроса (query) управляют правила yacc (файл main.y), подобные тем, которые показаны на примере. Из правил определяются сигнатуры доступных команд.

Для определения строкового или числового аргумента используются регулярные выражения (правила lex в файле rules.lex).

Для организации взаимодействий правил и отображения дерева были введены глобальные переменные, помимо query.

Примеры запросов:

Запрос:

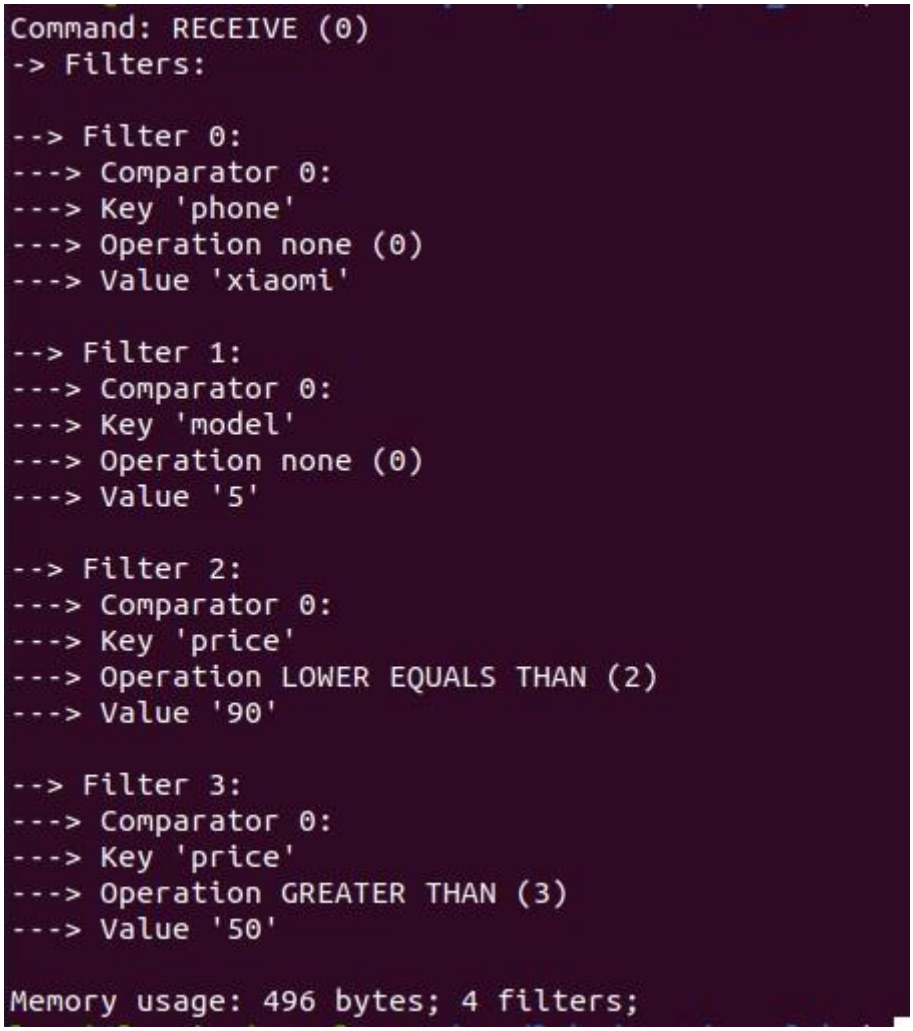
storage.insert({parent: 0}, {phone:"xiaomi", model: 5, price: 99})

Результат:

Запрос:

storage.receive({phone:"xiaomi", model: 5, price: {$lo_eq_t: 90}, price: {$gr_t: 50}})

Результат:



Запрос:

storage.update({phone:"xiaomi", model: 5, $or[price: {$gr_t: 100}, price: {$lo_t: 150}]}, $set:{phone: "iphone"})

Результат:

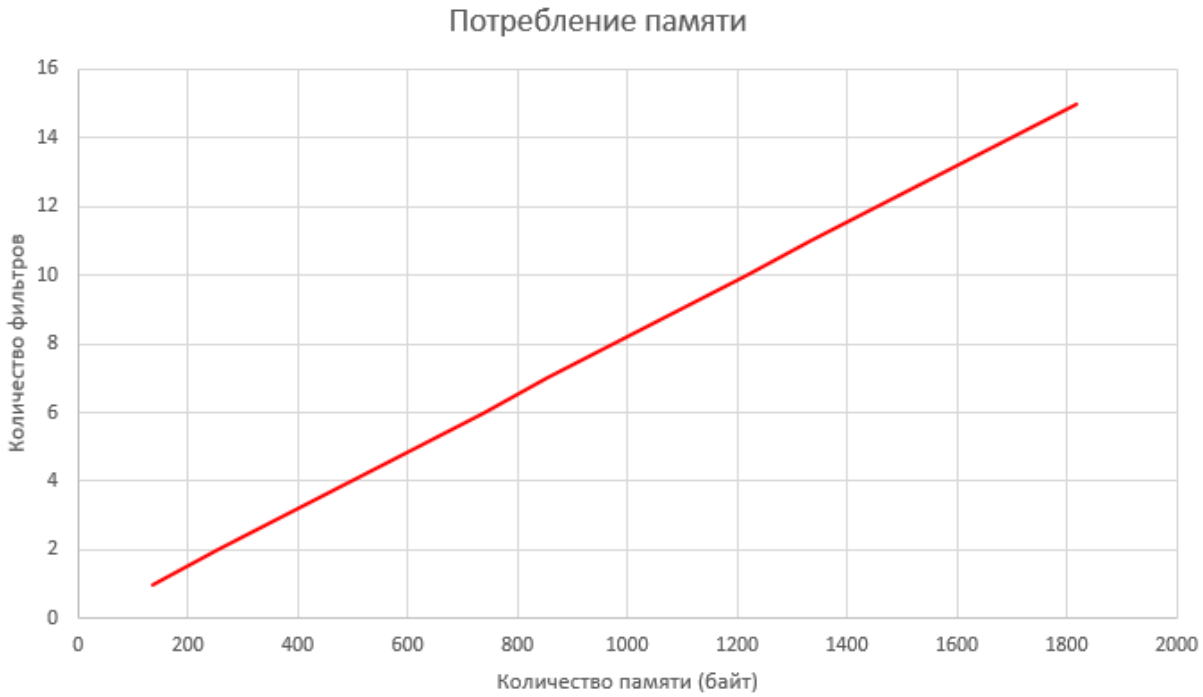
```
Command: UPDATE (3)
-> Filters:

--> Filter 0:
---> Comparator 0:
---> Key 'phone'
---> Operation none (0)
---> Value 'xiaomi'

--> Filter 1:
---> Comparator 0:
---> Key 'model'
---> Operation none (0)
---> Value '5'

--> Filter 2:
---> Comparator 0:
---> Key 'price'
---> Operation LOWER THAN (1)
---> Value '150'
---> Comparator 1:
---> Key 'price'
---> Operation GREATER THAN (3)
---> Value '100'

-> Settings:
--> Key 'phone'
--> Value 'iphone'

Memory usage: 520 bytes; 3 filters;
```

Запрос:

storage.delete({phone:"xiaomi", model: 5, price: {$no_eq: 99}, $or[color:"red", color:"black"]})

Результат:

Использование модулем памяти:



Как видно из графика, память растёт линейно, пропорционально количеству используемых фильтров

Вывод:

Был реализован модуль для разбора запросов языка MongoShell. В ходе работы я познакомился с программами lex и yacc, познакомился с организацией правил, понял, как можно включить синтаксический анализатор в свои программы.