Университет ИТМО

Факультет программной инженерии и компьютерной техники
Направление подготовки 09.03.04 Информатика и вычислительная техника
Дисциплина «Низкоуровневое программирование»

Отчет
По лабораторной работе №2
Вариант 7

Выполнил:
Казаченко Р. О.
Преподаватель:
Кореньков Ю. Д.

Санкт-Петербург, 2022 г.

Цель: реализовать модуль для разбора некоторого достаточного подмножества языка запросов по выбору в соответствии с вариантом формы данных.

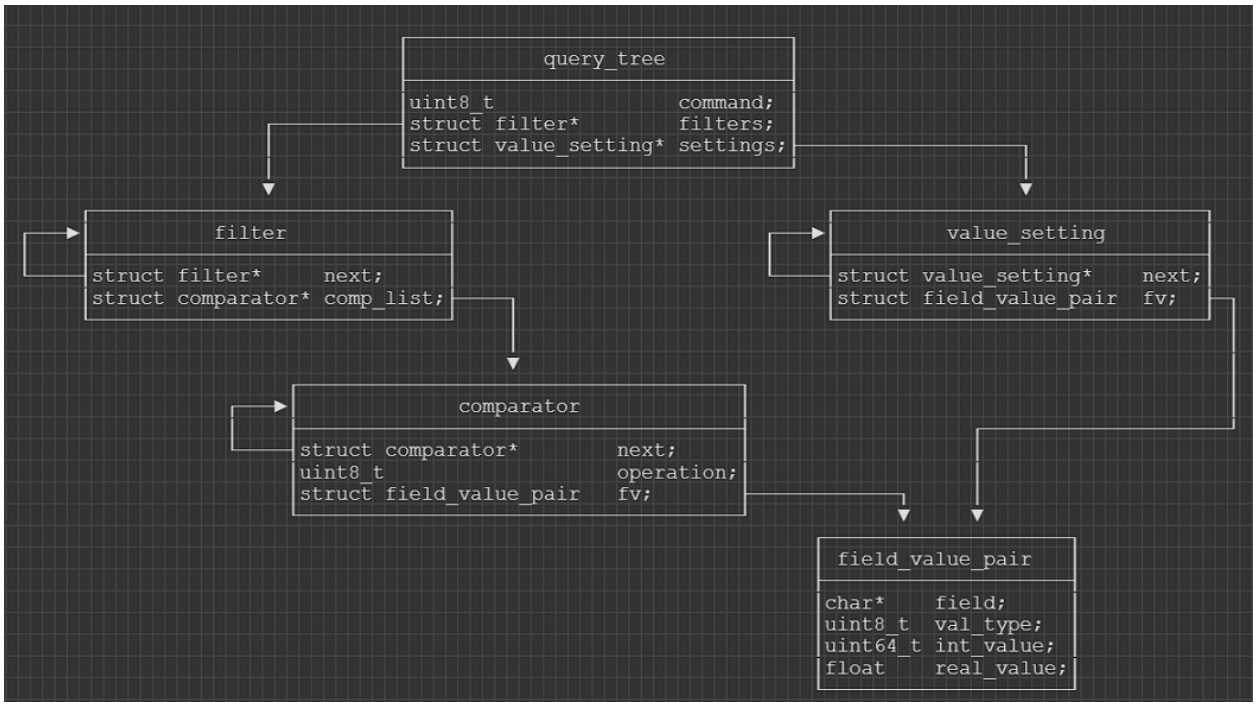
Задачи:

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

Описание работы:

Тестовая программа принимает на стандартные вход один запрос и выводит результат разбора.

Описание структур (сами структуры описаны в signatures.h):



Аспекты реализации:

Модуль реализован на основе Lex + YACC (файлы lex.l и mongo.y соответственно)
build.sh – скрипт сборки
query.ms – входной запрос

Операции над элементами:

- insert -- добавление элемента
- delete -- удаление элемента
- find – поиск элемента
- update -- изменение элемента

Особенности синтаксиса:

- $or{filter1, filter2} – логическое ИЛИ
- $lt : N – Less Than
- $let – Less or Equals Than
- $gt – Greater Than
- $get – Greater or Equals Than
- $ne – Not Equals

Базовое правило YACC:

```
mongosh: DB FIND OPBRACE OPCBRACE filters CLCBRACE CLBRACE {set_command(0);}
        |
        DB DELETE OPBRACE OPCBRACE filters CLCBRACE CLBRACE {set_command(1);}
        |
        DB INSERT OPBRACE parent_def COMMA vals_def CLBRACE {set_command(2);}
        |
        DB UPDATE OPBRACE OPCBRACE filters CLCBRACE COMMA DOLLAR SET COLON vals_def CLBRACE {set_command(3);}
        ;
```

По нему хорошо видно различие сигнатуры представленных команд. Единственное, что делают конкретно эти правила – устанавливают в глобальную переменную `struct query_tree tree` номер команды. Заполнением непосредственно фильтров, компараторов и т. п. занимаются соответствующие им правила, доставая значения из подзапроса посредством ‘$N’, где N – порядковый номер нужного аргумента подзапроса.

Помимо глобальной переменной, содержащей само дерево, было принято решение организовать взаимодействие некоторых правил через еще пару глобальных переменных, так как я не нашел, как на уровень выше передавать больше одного аргумента (имеется ввиду конструкция $$ = $N, которая запоминает лишь один аргумент).

Примеры правил Lex:

```
[a-zA-Z][a-zA-Z_0-9]*       {yylval.string = strdup(yytext); return STRING;}
[-]?([0-9]+)?\.[0-9]+       {yylval.fnum = strtod(yytext, NULL); return FLOAT_NUMBER;}
[-]?[0-9]+                  {yylval.num = atoi(yytext); return INT_NUMBER;}
```

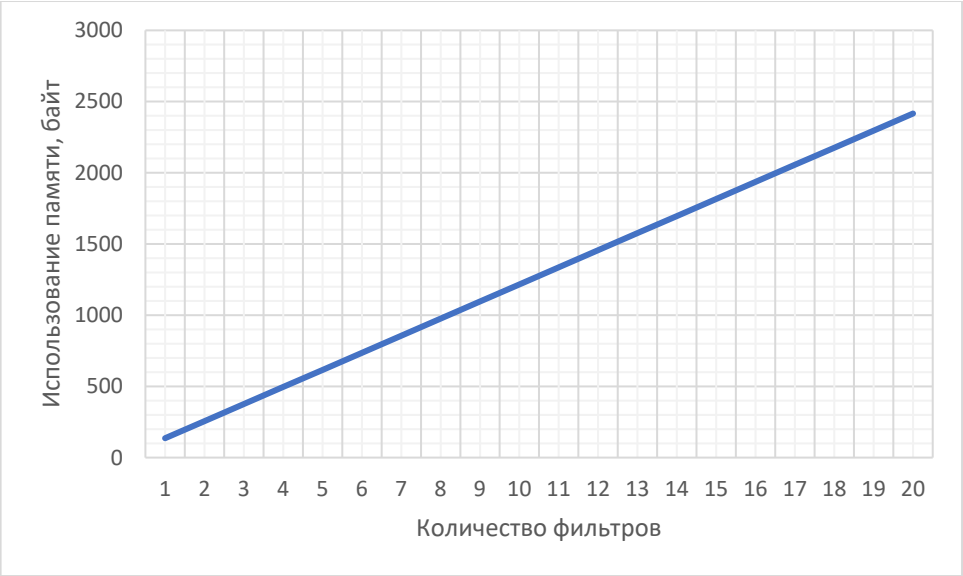
Для поиска строк и чисел в запросе были использованы регулярные выражения, причем строковые значение должны обязательно начинаться с буквы, чтобы не возникло коллизии с числовыми значениями. Стоит учитывать, что порядок правил в Lex имеет значение.

Результаты:

```
db.update({name:"slavik", male: True, balance: {$ne:42.3}, $or[x:1, y: {$lt:-13}]},
$set:{name: "stanislavik"})
--------------------------------------------------------------------------

      COMMAND: 3
       FILTERS:
        FILTER 0:
         COMPARATOR 0:
          FIELD 'name'
          OPERATION '0'
          VALUE 'slavik'
        FILTER 1:
         COMPARATOR 0:
          FIELD 'male'
          OPERATION '0'
          VALUE '1'
        FILTER 2:
         COMPARATOR 0:
          FIELD 'balance'
          OPERATION '5'
          VALUE '0.000000'
        FILTER 3:
         COMPARATOR 0:
          FIELD 'y'
          OPERATION '1'
          VALUE '-13'
         COMPARATOR 1:
          FIELD 'x'
          OPERATION '0'
          VALUE '1'
       SETTINGS:
        FIELD 'name'
        VALUE 'stanislavik'

      RAM USAGE: 640 bytes

db.insert({parent: 123}, {name:"slavik", male: True, balance: 999, height: -1.111})
--------------------------------------------------------------------------
           COMMAND: 2
            FILTERS:
             FILTER 0:
              COMPARATOR 0:
               FIELD 'parent'
               OPERATION '0'
               VALUE '123'
            SETTINGS:
             FIELD 'height'
             VALUE '-1.111000'

             FIELD 'balance'
             VALUE '999'

             FIELD 'male'
             VALUE '1'

             FIELD 'name'
             VALUE 'slavik'

           RAM USAGE: 296 bytes
```

```
db.insert({parent: 123}, {name:"slavik", male: True, })
--------------------------------------------------------------------------------
syntax error

db.insert({name:"slavik", male: True}) // <- не указали parent
--------------------------------------------------------------------------------
syntax error
```

График использования оперативной памяти



Как можно заметить, использование оперативной памяти линейно зависит от размера заполненного дерева запроса (например, от количества фильтров).

Выводы:

Был реализован модуль производящий синтаксический анализ и разбор запроса MongoShell. Помимо этого, я познакомился с синтаксисом Lex & YACC, а именно понятия правил, секций и их связь с кодом программы. По результатам тестов видно, что программа тратит память только на заполнение дерева запроса. Проблемой в процессе выполнения была организация взаимодействия в иерархии правил, для чего пришлось ввести глобальные переменные.