МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

ЛАБОРАТОРНАЯ РАБОТА №2

по дисциплине

“Низкоуровневое программирование”

Вариант 3 (GraphQL)

Выполнил:
Студент группы P33302

Добрягин Михаил
Александрович

Преподаватель:
Кореньков Юрий Дмитриевич

Санкт-Петербург, 2023

1. Цель

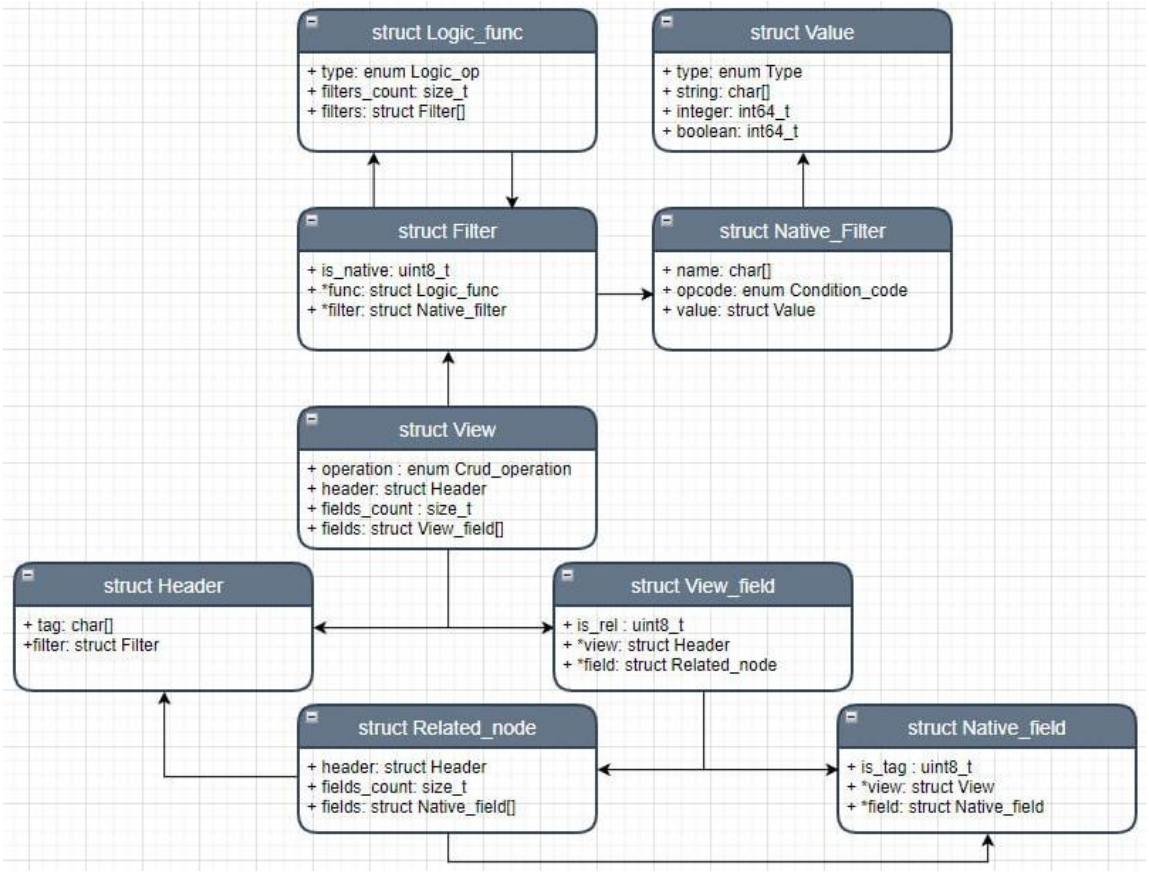
Реализовать модуль для разбора некоторого достаточного подмножества языка запросов по выбору в соответствии с вариантом формы данных.

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

2. Основные структуры

(сами структуры описаны в файле signatures.h)



3. Выполнение

3.1. Аспекты реализации:

Модуль реализован на основе Lex (lex.l) + Yacc (graphQL.y)

Операции: insert (добавление элемента), delete (удаление), query (поиск элемента), update (обновление элемента)

3.2. Особенности синтаксиса:

or: [filter1, filter2] – логическое ИЛИ

and: [filter1, filter2] – логическое И

not: [filter1] – логическое НЕ

eq – сравнивание элементов

lt – Less than

le – Less or equals than

gt – Greater than

gt – Greater or equals than

3.3. Базовые правила Yacc

```
graphQL: operation OPCBRACE body CLCBRACE
        {print_tree(tree); print_ram();};

operation: QUERY {set_opcode(0);}
        | DELETE {set_opcode(1);}
        | INSERT {set_opcode(2);}
        | UPDATE {set_opcode(3);};

body: STRING OPBRACE root_condition CLBRACE OPCBRACE fields CLCBRACE
        {memcpy(&tree.header.tag, $1, strlen($1));};
```

Сразу видна схожесть сигнатур представленных команд (правило body общее для всех CRUD’ов). Единственное, что делают конкретно эти правила – проставляют в глобальную переменную struct View tree номер команды и имя тега, а в конце печатают в стандартный поток вывода получившееся дерево, а также затраченную оперативную память. Заполнением непосредственно фильтров, тегов 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 ещё несколько глобальных переменных.

3.4. Пример Lex-правил

```
[a-zA-Z][a-zA-Z_0-9]*        {yylval.string = strdup(yytext); return STRING;}
[-]?[0-9]+                    {yylval.num = atoi(yytext); return INT_NUMBER;}
[ \t\n\r]                  ;
```

Для поиска строк и чисел в запросе были использованы регулярные выражения

4. Результаты

```
query{Entity(name:{eq:"Deutsche Bank AG"}){Country(name:{ge:"S"}){}}}
---END---
Query -> query{Entity(name:{eq:"Deutsche Bank AG"}){Country(name:{ge:"S"}){}}}

[COMMAND]: query

[TAG]: Entity

[CONDITIONS]:
name == Deutsche Bank AG

[NATIVE FIELDS]:

[RELATED NODES]:

        [RELATED NODE 0]:
        |TAG|: Country
        |CONDITIONS|:
        name >= S
```

5. Вывод

В результате выполнения лабораторной работы был разработан модуль, производящий синтаксический анализ и разбор GraphQL-запроса. Помимо этого, я познакомился с синтаксисом Lex и Yacc, а именно разобрался в понятиях правил, секций – их связь с генерируемым исходным кодом.