Университет ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.01 Информатика и вычислительная техника

**Лабораторная работа №2**

по дисциплине «Низкоуровневое программирование»   
Вариант GraphQl

Выполнил:

Студент группы P33302

Овчаренко А.А.

Преподаватель:

Кореньков Юрий Дмитриевич

г. Санкт-Петербург

2023

Содержание

[Задание 3](#_Toc145175054)

[Выполнение работы 4](#_Toc145175055)

[Итог 5](#_Toc145175056)

# Цели

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.

# Задачи

1. Изучить выбранное средство синтаксического анализа
   1. Средство должно поддерживать программный интерфейс совместимый с языком С
   2. Средство должно параметризоваться спецификацией, описывающий синтаксическую структуру разбираемого языка
   3. Средство может функционировать посредством кодогенерации и/или подключения необходимых для его работы дополнительных библиотек
   4. 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
   1. При необходимости добавления новых конструкций в язык, добавить нужные синтаксические конструкции в спецификацию (например, сравнения в GraphQL)
   2. Язык запросов должен поддерживать возможность описания следующих конструкций: порождение нового элемента данных, выборка, обновление и удаление существующих элементов данных по условию
      1. На равенство и неравенство для чисел, строк и булевских значений
      2. На строгие и нестрогие сравнения для чисел
      3. Существование подстроки
   3. Логическую комбинацию произвольного количества условий и булевских значений
   4. В качестве любого аргумента условий могут выступать литеральные значения (константы) или ссылки на значения, ассоциированные с элементами данных (поля, атрибуты, свойства)
   5. Разрешение отношений между элементами модели данных любых условий над сопрягаемыми элементами данных
   6. Поддержка арифметических операций и конкатенации строк не обязательна
3. Реализовать модуль, использующий средство синтаксического анализа для разбора языка запросов
   1. Программный интерфейс модуля должен принимать строку с текстом запроса и возвращать структуру, описывающую дерево разбора запроса или сообщение о синтаксической ошибке
   2. 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:
   1. В части 3 привести описание структур данных, представляющих результат разбора запроса
   2. В части 4 описать, какая дополнительная обработка потребовалась для результата разбора, представляемого средством синтаксического анализа, чтобы сформировать результат работы созданного модуля
   3. В части 5 привести примеры запросов для всех возможностей из п.2.b и результирующий вывод тестовой программы, оценить использование разработанным модулем оперативной памяти

# Описание работы

## Структуры для построения AST-дерева

typedef struct Node {

AstNodeType type;

union {

double double\_val;

int int\_val;

char\* str\_val;

int bool\_val;

} value;

struct Node \*left;

struct Node \*right;

} Node;

Дерево состоит из Node. Каждая Node имеет AstNodeType – тип токена, значение, если оно подразумевается типом токена, правого и левого ребенка.

Типы токенов

typedef enum {

QUERY,

MUTATION,

BOOLEAN,

NULL\_VALUE,

FIELD,

VARIABLE,

NUMBER,

STRING,

PARENTHESIS,

COLON,

EQUAL,

BRAKET\_OPEN,

BRAKET\_CLOSE,

CURLY\_BRACE\_OPEN,

CURLY\_BRACE\_CLOSE,

COMMA,

DOT,

EQUAL\_OPERATOR,

LOWER\_OPERATOR,

GREATER\_OPERATOR,

GREATER\_EQUAL,

LOWER\_EQUAL,

NOT\_EQUAL,

NOT,

OR,

AND,

INTEGER,

DOUBLE,

IDENTIFIER,

PLUS\_OPERATION,

MINUS\_OPERATION,

MULTIPLY\_OPERATION,

DIVIDE\_OPERATION,

RET,

DOUBLE\_DOT,

QOUTE,

ARGUMENT,

FILTER,

SELECTION,

OPERATION,

VALUE,

MATH\_VALUE,

VAR\_VALUE,

NAME\_DEFINITION,

VARIABLE\_DEFINITION,

SELECTION\_SET,

TYPE,

JSON,

JSON\_OBJECT,

JSON\_ELEMENT,

JSON\_ARRAY,

VALUES,

JSON\_ELEMENTS,

ROOT,

AST\_NODE\_TYPE\_COUNT

} AstNodeType;

Для реализации разбора запроса на токены и дальнейшей работы понадобилось реализовать следующие дополнительные методы

Node \*createNode\_V(AstNodeType type, Node \*left, Node \*right);

Node \*createNode\_I(AstNodeType type, int val, Node \*left, Node \*right);

Node \*createNode\_D(AstNodeType type, double val, Node \*left, Node \*right);

Node \*createNode\_B(AstNodeType type, bool val, Node \*left, Node \*right);

Node \*createNode\_S(AstNodeType type, char \*val, Node \*left, Node \*right);

void printRoot();

void addJsonInput(Node \*node);

void addRequest(Node \*node);

const char \*getAstNodeTypeName(AstNodeType type);

void printTree(Node \*node, int depth, int is\_left, int \*parent\_dir);

# Примеры выполнения

Запрос на порождение нового элемента

Запрос:

mutation CreateData($newPersonInput: 'PersonInput') {

# Create a new person

createPerson(input: $newPersonInput) {

id

posts {

id

title

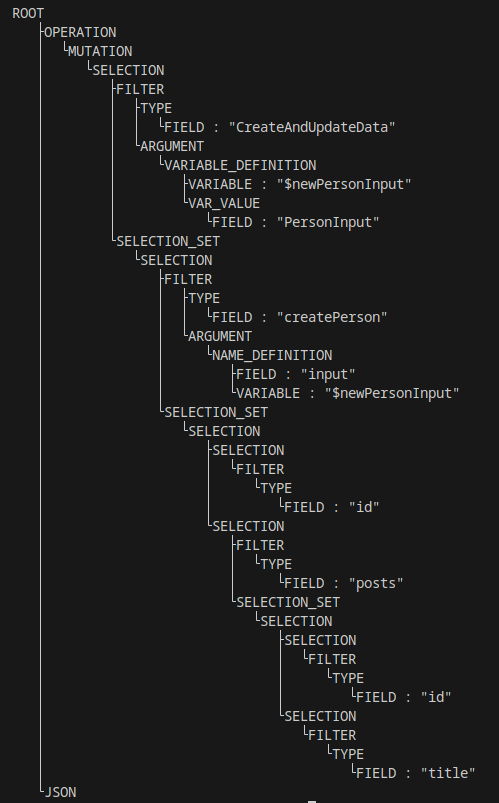
}

}

}

;

AST дерево



Запрос на обновление элемента + демонстрация арифметических операций

{

"glossary": {

"title": "example glossary",

"GlossDiv": "w342"

},

"dfsdf": [123, 231, 4, "23"]

}

mutation UpdateData($updatePersonInput: PersonInput, $postId: ID!) {

updatePerson(input: $updatePersonInput) {

id($postId)

age(age > (1 + 3) \* 8 / 8 \* 10)

lastName

}

}

;

ROOT

├OPERATION

│ └MUTATION

│ └SELECTION

│ ├FILTER

│ │ ├TYPE

│ │ │ └FIELD : "UpdateData"

│ │ └ARGUMENT

│ │ ├ARGUMENT

│ │ │ └VARIABLE\_DEFINITION

│ │ │ ├VARIABLE : "$updatePersonInput"

│ │ │ └VAR\_VALUE

│ │ │ └FIELD : "PersonInput"

│ │ └VARIABLE\_DEFINITION

│ │ ├VARIABLE : "$postId"

│ │ └VAR\_VALUE

│ │ └FIELD : "ID!"

│ └SELECTION\_SET

│ └SELECTION

│ ├FILTER

│ │ ├TYPE

│ │ │ └FIELD : "updatePerson"

│ │ └ARGUMENT

│ │ └NAME\_DEFINITION

│ │ ├FIELD : "input"

│ │ └VARIABLE : "$updatePersonInput"

│ └SELECTION\_SET

│ └SELECTION

│ ├SELECTION

│ │ ├SELECTION

│ │ │ └FILTER

│ │ │ ├TYPE

│ │ │ │ └FIELD : "id"

│ │ │ └ARGUMENT

│ │ │ └VARIABLE\_DEFINITION

│ │ │ └VARIABLE : "$postId"

│ │ └SELECTION

│ │ └FILTER

│ │ ├TYPE

│ │ │ └FIELD : "age"

│ │ └ARGUMENT

│ │ └GREATER\_OPERATOR

│ │ ├FIELD : "age"

│ │ └MATH\_VALUE

│ │ └MULTIPLY\_OPERATION

│ │ ├DIVIDE\_OPERATION

│ │ │ ├MULTIPLY\_OPERATION

│ │ │ │ ├PLUS\_OPERATION

│ │ │ │ │ ├INTEGER : 1

│ │ │ │ │ └INTEGER : 3

│ │ │ │ └INTEGER : 8

│ │ │ └INTEGER : 8

│ │ └INTEGER : 10

│ └SELECTION

│ └FILTER

│ └TYPE

│ └FIELD : "lastName"

└JSON

└JSON\_OBJECT

└JSON\_ELEMENTS

├JSON\_ELEMENT

│ ├VALUE

│ │ └STRING : ""glossary""

│ └JSON\_OBJECT

│ └JSON\_ELEMENTS

│ ├JSON\_ELEMENT

│ │ ├VALUE

│ │ │ └STRING : ""title""

│ │ └VALUE

│ │ └STRING : ""example glossary""

│ └JSON\_ELEMENT

│ ├VALUE

│ │ └STRING : ""GlossDiv""

│ └VALUE

│ └STRING : ""w342""

└JSON\_ELEMENT

├VALUE

│ └STRING : ""dfsdf""

└JSON\_ARRAY

└VALUES

├VALUES

│ ├VALUES

│ │ ├MATH\_VALUE

│ │ │ └INTEGER : 123

│ │ └MATH\_VALUE

│ │ └INTEGER : 231

│ └MATH\_VALUE

│ └INTEGER : 4

└VALUE

└STRING : ""23""

# Итог

В результате выполнения лабораторной работы был изучен инструмент лексического анализа flex и инструмент синтаксического анализа bison. Была разработана программа для анализа запроса на языке GraphQl и построения AST-дерева.