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

Язык запросов GraphQL

Студент:

Степанов Михаил
Андреевич

Группа P33312

Преподаватель:

Кореньков Юрий Дмитриевич

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.

Порядок выполнения:

1. Изучить выбранное средство синтаксического анализа
    a. Средство должно поддерживать программный интерфейс совместимый с языком C
    b. Средство должно параметризоваться спецификацией, описывающий синтаксическую структуру разбираемого языка
    c. Средство может функционировать посредством кодогенерации и/или подключения необходимых для его работы дополнительных библиотек
    d. 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
    a. При необходимости добавления новых конструкций в язык, добавить нужные синтаксические конструкции в спецификацию (например, сравнения в GraphQL)
    b. Язык запросов должен поддерживать следующие возможности:
        - Условия
            o На равенство и неравенство для чисел, строк и булевских значений
            o На строгие и нестрогие сравнения для чисел
            o Существование подстроки
        - Логическую комбинацию произвольного количества условий и булевских значений
        - В качестве любого аргумента условий могут выступать литеральные значения (константы) или ссылки на значения, ассоциированные с элементами данных (поля, атрибуты, свойства)
        - Разрешение отношений между элементами модели данных любых условий над сопрягаемыми элементами данных
        - Поддержка арифметических операций и конкатенации строк не обязательна
    c. Разрешается разработать свой язык запросов с нуля, в этом случае необходимо показать отличие основных конструкций от остальных вариантов (за исключением типичных выражений типа инфиксных операторов сравнения)

3. Реализовать модуль, использующий средство синтаксического анализа для разбора языка запросов

a. Программный интерфейс модуля должен принимать строку с текстом запроса и возвращать структуру, описывающую дерево разбора запроса или сообщение о синтаксической ошибке

b. 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:

a. В части 3 привести описание структур данных, представляющих результат разбора запроса

b. В части 4 описать, какая дополнительная обработка потребовалась для результата разбора, представляемого средством синтаксического анализа, чтобы сформировать результат работы созданного модуля

c. 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:

Программа реализована в виде модуля, который запрашивает у пользователя строку и обертку над Ast деревом для возвращения результата. Код возврата представлен значением int. Если возвращен не 0, то программа отработала с ошибкой.

Пример использования:

```
Person(select: true, node_name: "John"){
    id,
    props.cyl
};

Query: Select
class_type: Person
  SelectionSet:
    Arguments:
      Argument: node_name
      Argument_value: "John"
  ResultSet:
    StringConstant: id
```

```
    StringConstant: props.cyl
```

Аспекты реализации:

- *lexel.l* - файл лексера (flex)
- *parser.y* - файл парсера (bison)
- *ast.h, ast.cpp* - реализация узлов дерева запросов

Типы узлов

```
enum node_type {
    QUERY_NODE,
    SELECTION_SET_NODE,
    RESULT_SET_NODE,
    ARGUMENT_WRAPPER_NODE,
    ARGUMENT_NODE,
    OBJECT_WRAPPER_NODE,
    OBJECT_NODE,
    FIELDS_WRAPPER_NODE,
    FIELD_NODE,
    RELATION_WRAPPER_NODE,
    RELATION_NODE,
    SUB_OPERATION_WRAPPER,
    SUB_OPERATION,
    FILTER_NODE,
    LOGICAL_OP_NODE,
    CONSTANT_NODE
};
```

Типы данных

```
enum data_type {
    INT,
    FLOAT,
    STRING,
    BOOL
};
```

Типы операторов сравнения

```
enum filter_operation{
    GT,
    GE,
    LT,
    LE,
```

```
    NE,
    LIKE
};
```

Типы логических операторов

```
enum logical_operation{
    OR,
    AND
};
```

Типы операторов модификации

```
enum sub_operation {
    SET,
    ADD,
    SUB
};
```

Примеры запросов:

- **Базовая выборка**

```
Person(select: true, node_name: "John"){
    id,
    props.cyl
};
Query: Select
class_type: Person
  SelectionSet:
    Arguments:
      Argument: node_name
      Argument_value: "John"
  ResultSet:
    StringConstant: id
    StringConstant: props.cyl
```

- **Выборка с несколькими условиями**

```
Person(select: true, node_name: "John", props.privet: 123){
    id,
    props.cyl
};
Query: Select
class_type: Person
  SelectionSet:
```

```
    Arguments:
      Argument: node_name
        Argument_value: "John"
      Argument: props.privet
        Argument_value: 123
  ResultSet:
    StringConstant: id
    StringConstant: props.cyl
```

- **Выборка с оператором сравнения на условие**

```
Person(select: true, node_name: "John", props.privet: filter{ op:
gt, val: 123}){
    id,
    props.cyl,
    relations.muratu
};
Query: Select
class_type: Person
  SelectionSet:
    Arguments:
      Argument: node_name
        Argument_value: "John"
      Argument: props.privet
        Argument_value: 123
        Filter: GT
  ResultSet:
    StringConstant: id
    StringConstant: props.cyl
    StringConstant: relations.muratu
```

- **Выборка с логическим оператором**

```
Person(select: true, node_name: "John", or{id:5, props.privet:
123}){
    id,
    props.cyl
};
Query: Select
class_type: Person
  SelectionSet:
    Arguments:
      Argument: node_name
        Argument_value: "John"
```

```
      Logical_op: OR
        Arguments:
          Argument: id
            Argument_value: 5
          Argument: props.privet
            Argument_value: 123
  ResultSet:
    StringConstant: id
    StringConstant: props.cyl
```

- **Выборка с комбинацией аргументов**

```
Person(select: true, node_name: "John", or{ and{ id:5,
relations.kiwi: "Ogurec"}, props.cyl: 10}, props.privet: 123){
    id,
    props.cyl
};
Query: Select
class_type: Person
  SelectionSet:
    Arguments:
      Argument: node_name
        Argument_value: "John"
      Logical_op: OR
        Arguments:
          Logical_op: AND
            Arguments:
              Argument: id
                Argument_value: 5
              Argument: relations.kiwi
                Argument_value: "Ogurec"
          Argument: props.cyl
            Argument_value: 10
      Argument: props.privet
        Argument_value: 123
  ResultSet:
    StringConstant: id
    StringConstant: props.cyl
```

- **Вставка элемента**

```
Person(insert: {
    {
        node_name: Volkswagen,
```

```
        props{
            id: 5,
            cyl: 124.5
        }
    }
}){
    id
};
Query: Insert
class_type: Person
  SelectionSet:
    Objects:
      Object:
        Name: Volkswagen
        Fields:
          Field: id
            Value: 5
          Field: cyl
            Value: 124.500000
  ResultSet:
    StringConstant: id
```

- **Удаление элемента**

```
Person(delete: true,
    node_name: "John"){
    id,
    node_name
};
Query: Delete
class_type: Person
  SelectionSet:
    Arguments:
      Argument: node_name
        Argument_value: "John"
  ResultSet:
    StringConstant: id
    StringConstant: node_name
```

- **Обновление элемента**

```
Person(update:[["set", node_name, "Jack"],["set", props.cyl,
"123"]], id: 1){
    id,
```

```
    node_class,
    node_name
};
Query: Update
class_type: Person
  SelectionSet:
    Arguments:
      Argument: id
        Argument_value: 1
    SubOperations:
      Sub_Operation: Person
        Name: node_name
        Value: "Jack"
      Sub_Operation: "Jack"
        Name: props.cyl
        Value: "123"
  ResultSet:
    StringConstant: id
    StringConstant: node_class
    StringConstant: node_name
```

Вывод:

В процессе выполнения лабораторной работы было разработано абстрактное синтаксическое дерево запросов к базе данных из лабораторной работы №1. Разработан модуль считывания и распознавания запросов при помощи технологий Flex и Bison.