Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное
образовательное учреждение высшего образования
“Национальный исследовательский университет ИТМО”
ФКТиУ, Кафедра Информатики и вычислительной техники

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

По предмету:
«Низкоуровневое программирование»

Выполнили:
Студент группы P33302
Стуков Е. А.

Преподаватель:
Кореньков Ю. Д.

Санкт-Петербург
2022 г.

Задание:

Вариант: Gremlin

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
            - На равенство и неравенство для чисел, строк и булевских значений
            - На строгие и нестрогие сравнения для чисел
            - 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

Выполнение:

https://github.com/ZloyEgor/LLP-Lab-2

Часть 1. Выбор средства синтаксического анализа

В качестве инструмента синтаксического анализа были выбраны утилиты Flex и Bison. Flex используется для разбиения входного потока данных на лексемы, набор которых обрабатывается согласно правилам, описанным в программе для Bison.

Часть 2. Синтаксис языка запросов. Запись спецификации.

Примеры составления запросов на языке Gremlin:
https://docs.janusgraph.org/getting-started/gremlin/

Были составлены следующие типы запросов:

- Создание файла

```
create(“filename.txt”);
```

- Открытие существующего файла

```
open(“filename.txt”);
```

- Закрытие файла

```
close();
```

- Добавление схемы

```
addSchema(“schema_name”, “1st_attr”, <attr_type>, “2nd_attr”, <attr_type>, …);
```

- Удаление схемы

```
deleteSchema(“schema_name”);
```

- Добавление записи

```
addVertex(“schema_name”, “attr_name”, <attr_value>, …);
```

- Получение всех элементов схемы

```
V("schema_name");
```

- Получение элементов схемы, соответствующих набору условий значений атрибутов

```
V("schema_name").has("attr_name", <select_option>(<select_value>), …);
```

- Получение записей, ассоциированных с ключевой схемой по ребру

```
V("schema_name").out("edge_name");
```

- Удаление элементов схемы

```
V("schema_name").delete();
```

Примечание:

Запрос может содержать чередование условий выборки элементов с соединением по ребрам. Команда удаления элементов может быть использована в сочетании с выборкой по условиям.

Поддерживаются следующие условия выборки элементов данных:

- Равенство

  ```
  eq(<attr_value>)
  ```

- Строго больше

  ```
  gt(<attr_value>)
  ```

- Больше или равно

  ```
  gte(<attr_value>)
  ```

- Строго меньше

  ```
  lt(<attr_value>)
  ```

- Меньше или равно

  ```
  lte(<attr_value>)
  ```

- Неравенство

  ```
  neq(<attr_value>)
  ```

- Включение подстроки

  ```
  like(<substr>)
  ```

Часть 3. Реализация модуля разбора языка запросов.

Реализация находится в файлах lexer.l и parser.y.

Структура дерева составления запросов описана в файле request_tree.h:

```
typedef enum request_type {
    UNDEFINED,
    REQUEST_OPEN,
    REQUEST_CREATE,
    REQUEST_CLOSE,
    REQUEST_ADD_SCHEMA,
    REQUEST_DELETE_SCHEMA,
    REQUEST_ADD_NODE,
    REQUEST_SELECT
} request_type;

typedef enum attr_type {
    ATTR_TYPE_INTEGER = 0,
    ATTR_TYPE_BOOLEAN,
    ATTR_TYPE_FLOAT,
    ATTR_TYPE_STRING,
    ATTR_TYPE_REFERENCE
} attr_type;

typedef struct file_work_struct {
    char *filename;
} file_work_struct;

typedef struct attribute_declaration {
    char *attr_name;
    attr_type type;
    char *schema_ref_name;
} attribute_declaration;

typedef struct add_schema_struct {
    char *schema_name;
    arraylist *attribute_declarations;
} add_schema_struct;

typedef struct delete_schema_struct {
    char *schema_name;
} delete_schema_struct;

union value {
    int integer_value;
    bool bool_value;
    char* string_value;
    float float_value;
};

typedef struct attr_value {
    char *attr_name;
```

```
    attr_type type;
    union value value;
} attr_value;

typedef struct add_node_struct {
    char* schema_name;
    arraylist *attribute_values;
} add_node_struct;

typedef enum select_option {
    OPTION_EQUAL,
    OPTION_GREATER,
    OPTION_GREATER_EQUAL,
    OPTION_LESS,
    OPTION_LESS_EQUAL,
    OPTION_NOT_EQUAL,
    OPTION_LIKE
} select_option;

typedef struct select_condition {
    char *attr_name;
    select_option option;
    attr_type type;
    union value value;
} select_condition;

typedef enum statement_type {
    SELECT_CONDITION,
    OUT,
    DELETE
} statement_type;

typedef struct statement {
    statement_type type;
    union {
        arraylist *conditions;
        char *attr_name;
    };
} statement;

typedef struct request_tree {
    request_type type;
    char* schema_name;
    union {
        file_work_struct file_work;
        add_schema_struct add_schema;
        delete_schema_struct delete_schema;
        add_node_struct add_node;
        arraylist *statements;
    };
} request_tree;
```

Часть 4. Демонстрация работоспособности реализованного модуля.

Реализованная программа принимает введенный запрос и производит составление дерева запроса, записывая его в структуру request_tree. После составление содержимое структуры печатается на консольный вывод. Помимо этого, происходит вывод размера составленного дерева в байтах.
Примечание: в рамках реализации дерева для обеспечения произвольного количества условий выбора атрибутов, количества атрибутов при создании схемы, а также произвольного количества операций типа “out” и “has”, используется структура arraylist. В следствие чего для хранения элементов выделяется массив, зачастую больший по размеру, чем требуемое количество хранимых элементов. При подсчете размера дерева неиспользуемые, но аллоцированные ячейки памяти также считаются наравне с используемыми. Помимо этого, учитывается размер аллоцированных строк для хранения имен атрибутов, схем и файлов.

```
open("filename.txt");
Open file: "filename.txt"
Tree size: 44 bytes
```

```
create("filename.txt");
Create file: "filename.txt"
Tree size: 44 bytes
```

```
addSchema("test_schema_0");
Add schema: "test_schema_0"
No attributes
Tree size: 45 bytes
```

```
addSchema("test_schema_1", "first", integer, "second", float, "third", boolean);
Add schema: "test_schema_1"
"first": integer
"second": float
"third": boolean
Tree size: 173 bytes
```

```
addSchema("test_schema_2", "first", integer, "ref_attr", reference("test_schema_1"));
Add schema: "test_schema_2"
"first": integer
"ref_attr": reference to test_schema_1
Tree size: 183 bytes
```

```
addVertex("test_schema_1", "first", 123, "second", 32.1, "third", true, "forth", "Hello_world");
Add node of schema: test_schema_1
"first": 123
"second": 32.099998
"third": true
"forth": Hello_world
Tree size: 146 bytes
```

```
V("test_schema1").has("first", eq(123), "second", gte(21.2));
Select nodes: "test_schema1"
* Condition of selection:
  "first" = 123
  "second" >= 21.2000
Tree size: 235 bytes
```

```
V("test_schema3").has("first", eq(123)).has("second", like("example"));
Select nodes: "test_schema3"
* Condition of selection:
  "first" = 123
* Condition of selection:
  "second" like example
Tree size: 347 bytes
```

```
V("test_schema3").has("first", eq(123)).out("ref");
Select nodes: "test_schema3"
* Condition of selection:
  "first" = 123
* Out nodes by "ref"
Tree size: 232 bytes
```

```
V("test_schema3").has("first", eq(123)).out("ref").has("second", eq(5), "third", lte(37.34)).delete();
Select nodes: "test_schema3"
* Condition of selection:
  "first" = 123
* Out nodes by "ref"
* Condition of selection:
  "second" = 5
  "third" <= 37.3400
* Delete nodes
Tree size: 355 bytes
```

На основе сделанных тестов, можно видеть, что размер составляемого дерева запроса зависит от количества используемых в запросе конструкций и условий, и чем больше в запросе условий выборки / перечисления атрибутов, тем больше размер дерева запроса.

Выводы:

В ходе выполнения данной лабораторной работы были задействованы такие инструменты, как Flex и Bison. Была рассмотрена грамматика языка запросов Gremlin, на основе которой была записана спецификация для утилит синтаксического анализа. Спецификация позволяет составлять запросы на работу с файлом данных (открытие, создание, удаление), работу со схемами (добавление/удаление схем), на добавление элементов данных с указанием значений атрибутов, а также на поиск элементов данных по заданным условиям значения указанных в запросе атрибутов с возможностью вывода информации о соединенных ребрами элементов данных и удаления перечисляемых узлов.

Был реализован модуль, способный производить синтаксический разбор запроса и составлять его на основе дерево запроса. Полученное дерево может быть выведено в текстовом виде.