Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №2

по «Низкоуровневое программирование»

Вариант – 6(Gremlin)

Выполнил:

Студент группы P3214

Голощапов И.А.

Преподаватели:

Кореньков Юрий Дмитриевич

Санкт-Петербург

2023

Задание 2

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

Исходный код: https://github.com/Ja-Vani/LLP2/tree/main

Часть 1 Использовались Flex и Bison

На основе просмотра языка Gremlin были созданы следующие типы запросов.

Create(“file”) – создания файла

Open(“file”) – открытие файла

Close() – закрытие файла

addVertex(“name”, “att1”, type1, …) – создание вершины

addEdge(“name”, int1, int2) – Создание связи между вершинами

V(“name”) – получение вершин типа name

V() -получение всех вершин

V().has(“att1”, <option>, ….) – получение вершин по условию

V().out(“name”) – получение связей

V().delete() – удаление вершин

deleteEdge(“name”, int1, int2) – удаление вершины

Условия выборки:

eq() равенство;

neq() неравенство;

gt() строго больше

gte() больше или равно

lt() строго меньше

lte() меньше или равно

like() включение подстроки

Описание структур данных:

```
typedef enum request_type {
    UNDEFINED,
    REQUEST_OPEN,
    REQUEST_CREATE,
    REQUEST_CLOSE,
    REQUEST_ADD_NODE,
    REQUEST_SELECT,
    REQUEST_ADD_EDGE,
    REQUEST_DELETE_EDGE
} 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;

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_edge_struct {
    char* schema_name;
    int node1;
    int node2;
} add_edge_struct;

typedef struct delete_edge_struct {
    char* schema_name;
    int node1;
    int node2;
} delete_edge_struct;

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_edge_struct add_edge;
        delete_edge_struct delete_edge;
        add_node_struct add_node;
        arraylist *statements;
    };
} request_tree;
```

Результаты запросов:

```
deleteEdge("name", 123, 123);
Delete edge name of nodes: 123 123
Tree size: 44 bytes
```

```
addEdge("name", 123, 123);
Add edge name of nodes: 123 123
Tree size: 44 bytes
```

```
Open file: "filename.txt"
Tree size: 44 bytes
```

```
addVertex("name", "Name", "Ivan", "age", 21, "weight", 58.5, "LLP", false);
Add node of type: name
"Name": Ivan
"age": 21
"weight": 58.500000
"LLP": false
Tree size: 132 bytes
```

```
V().has("name", eq("Ivan"));
Select all nodes
* Condition of selection:
        "name" = Ivan
Tree size: 228 bytes
```

```
V("person").has("age", gte(18));
Select nodes: "person"
* Condition of selection:
        "age" >= 18
Tree size: 227 bytes
```

```
V().has("name", like("Ivan"), "age", lt(40), "weight", eq(40));
Select all nodes
* Condition of selection:
        "name" like Ivan
        "age" < 40
        "weight" = 40
Tree size: 237 bytes
```

```
V().has("name", eq("Ivan")).out("person").out("person").delete();
Select all nodes
* Condition of selection:
        "name" = Ivan
* Out nodes by "person"
* Out nodes by "person"
* Delete nodes
Tree size: 240 bytes
```

По запросам видно, что размер дерева зависит от количества используемых в дереве конструкций.

Вывод:

В ходе выполнения работы я ознакомился с языком запросов Gremlin, и сделал модель для разбора некоторых функция из данного языка. С учётом специфики языка были внесены некоторые изменения в систему команд.