Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский университет ИТМО»

Факультет Программной Инженерии и Компьютерной Техники

Дисциплина: Низкоуровневое программирование

Лабораторная работа 2

Вариант 7

Выполнил:

Гурьянов Кирилл Алексеевич

Группа: P33302

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 возможность описания следующих конструкций:
порождение нового элемента данных, выборка, обновление и удаление существующих элементов данных по условию

· 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

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

Ход работы

В работе есть 2 модуля, parser и view. Первый модуль отвечает за парсинг запроса и упаковку в структуру request (в аспектах реализации). Он содержит функции, которые читают запрос и формируют соответствующую структуру. Модуль view отвечает за отображение этой структуры в консоль.

Пример работы программы

Добавление элемента

```
enter your request
add({string : qwe, int : 10, double : 13.4, boolean : 0});

operation - add

    operand name - string
        condition - :
            operand value - qwe
                operand name - int
                    condition - :
                        operand value - 10
                            operand name - double
                                condition - :
                                    operand value - 13.4
                                        operand name - boolean
                                            condition - :
                                                operand value - 0
```

Удаление элемента по id

```
enter your request
delete( { id : 1 } );

operation - delete

    operand name - id
        condition - :
            operand value - 1
```

Поиск элемента по id

```
enter your request
find({id : 2});

operation - find

    operand name - id
        condition - :
            operand value - 2
```

Поиск элементов, удовлетворяющих условию с булевым объединением

```
enter your request
find({int > 20 | boolean : 1});

operation - find

    operand name - int
        condition - >
            operand value - 20
                between attributes - |
                    operand name - boolean
                        condition - :
                            operand value - 1
```

Поиск всех элементов

```
enter your request
find({id : *})

operation - find

    operand name - id
        condition - :
            operand value - *
```

Поиск всех элементов, родитель id которого равен

```
enter your request
find({id : 1 {id : *}});

operation - find

father id - 1
    operand name - id
        condition - :
            operand value - *
```

Поиск элемента, родитель которого 1 и некие условия

```
enter your request
find({id : 1 {int > 10, double <: 25.5}});

operation - find

father id - 1
    operand name - int
        condition - >
            operand value - 10
                operand name - double
                    condition - <:
                        operand value - 25.5
```

Обновление элемента по id

```
enter your request
update({id : 2, int : 18});

operation - update

    operand name - id
        condition - :
            operand value - 2
                operand name - int
                    condition - :
                        operand value - 18
```

Пример некорректного запроса

```
enter your request
find({{{id : *})
PARSE ERROR
```

Аспекты реализации

Структура request:

```
struct attribute {
    char *left;
    char *right;
    char *condition;
    struct attribute *next_attribute;
    char *combined_condition;
};

struct request {
    char *operation;
    char *parent_id;
    struct attribute *attributes;
};
```

В этой структуре хранится действие (add, remove, find, update), id родителя, и маркер для поиска всех элементов (опционально). Действие представляет представляет собой операцию, которую необходимо выполнить с данными. Это ключевое слово определяет желаемое действие с данными. parant_id — это идентификатор, который связывает текущий запрос с другим элементом данных, обычно считается "родительским" элементом. Это помогает организовать иерархию данных или запросов в системе. Также в этой структуре хранятся атрибуты.

Атрибуты представлены односторонним списком, что обеспечивает удобный доступ и манипуляции с ними. Левая часть выражения (Left): Это часть атрибута, которая указывает на свойство или характеристику данных, например, "имя".Правая часть выражения (Right): Это значение или данные, с которыми сравнивается левая часть, например, "Roma". Condition (Условие): Это оператор сравнения, определяющий отношение между левой и правой частью выражения, такой как "=", ">", "<" и т.д. Булево сплетение выражений: Это оператор, который объединяет различные атрибуты или их условия, позволяя создавать более сложные логические выражения, например, "|" (ИЛИ) или "&" (И).

Эта структура данных позволяет компактно организовать запросы и их атрибуты, что удобно для фильтрации данных или выполнения операций над ними в зависимости от заданных условий и требований. Отдельные атрибуты представлены в форме выражений, а булево сплетение их условий позволяет создавать сложные правила фильтрации или поиска данных.

Операции:

- add - добавление
- delete - удаление
- find - поиск
- update – обновление

Отношения между атрибутами и булевы знаки:

- : - равно
- !: - не равно
- > - больше
- < - меньше
- >: - больше или равно
- <: - меньше или равно
- & - логическое И
- | - логическое ИЛИ
- * - маркер «все элементы»

Парсер:

```c
enum parser_status parse_request(char *req, struct request *request) {


    int path_length = strlen( s: req);

    check_path(req, &path_length);

    char *operation = read_word(&req, &path_length);
    if (!(
            strcmp("add", operation) == 0 ||
            strcmp("find", operation) == 0 ||
            strcmp("delete", operation) == 0 ||
            strcmp("update", operation) == 0
    ))
        print_error();

    request->operation = operation;

    struct attribute *attribute = malloc( size: sizeof(struct attribute));
    request->attributes = attribute;
    int flag = read_attributes(&req, &path_length, attribute);

    struct attribute *attr = request->attributes;
```

Рекурсивное заполнение списка атрибутов

```c
int read_attributes(char **req, int *path_length, struct attribute *attribute) {
    int flag = 0;

    if ((*req)[0] != '|' && (*req)[0] != '&') {

        char *left = read_word(req, path_length);
        char *condition = read_word(req, path_length);
        char *right = read_word(req, path_length);

        attribute->left = left;
        attribute->condition = condition;
        attribute->right = right;

        if ((*req)[0] == '{') {
            flag++;
            struct attribute *new_attribute = malloc( size: sizeof(struct attribute));
            flag += read_attributes(req, path_length, attribute: new_attribute);
            attribute->next_attribute = new_attribute;
        }
    }

    if ((*req)[0] == ',') {
        remove_char( size: path_length, request_path: req);
        remove_char( size: path_length, request_path: req);

        struct attribute *new_attribute = malloc( size: sizeof(struct attribute));
        flag += read_attributes(req, path_length, attribute: new_attribute);
        attribute->next_attribute = new_attribute;

    } else if ((*req)[0] == '|' || (*req)[0] == '&') {
        char *combined_condition = read_word(req, path_length);

        attribute->combined_condition = combined_condition;

        struct attribute *new_attribute = malloc( size: sizeof(struct attribute));
        flag += read_attributes(req, path_length, attribute: new_attribute);
        attribute->next_attribute = new_attribute;
    }

    remove_char( size: path_length, request_path: req);
```

Вывод

В ходе выполнения лабораторной работы мною был разработан модуль для разбора некоторого достаточного подмножества языка запросов по выбору в соответствии с вариантом формы данных. Кажется, получилось довольно эффективно использовать память, ограничившись только хранением структуры request. Я разработал и успешно реализовал parser и view, что позволяет обрабатывать и визуализировать информацию из этой структуры. Лабораторная успешно протестирована, и всё выглядит в порядке. Это был интересный процесс, где каждый этап привел к успешному результату.