Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №2

по «Низкоуровневое программирование»

Вариант XPath

Выполнил:

Студент группы P33301

Акимов Роман Иванович

Преподаватель:

Кореньков Юрий Дмитриевич

Санкт-Петербург

2023

Задачи

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

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

Ход работы

Модуль parser отвечает за парсинг запроса и упаковку в структуру request. Модуль view отвечает за отображение этой структуры в консоль.

Пример работы программы

Добавление элемента

```
enter your request
add/1/[name=qwe][age=10][height=50.22][healthy=1]

action - add

id - 1
    left part - name
        condition - =
            right part - qwe
                left part - age
                    condition - =
                        right part - 10
                            left part - height
                                condition - =
                                    right part - 50.22
                                        left part - healthy
                                            condition - =
                                                right part - 1
```

Удаление элемента по id

```
enter your request
remove/1

action - remove

id - 1
```

Удалить все элементы

```
enter your request
remove/*

action - remove

id - *
```

Поиск элемента по id

```
enter your request
find/1

action - find

id - 1
```

Поиск всех элементов, удовлетворяющих условию

```
enter your request
find/*/[int=10]

action - find

id - *
    left part - int
        condition - =
            right part - 10
```

Поиск элементов, удовлетворяющих условию с булевым объединением

```
enter your request
find/*/[int=10]|[double=22.2]

action - find

id - *
    left part - int
        condition - =
            right part - 10
                between pair - |
                    left part - double
                        condition - =
                            right part - 22.2
```

Поиск всех элементов

```
enter your request
find/*

action - find

id - *
```

Поиск всех элементов, родитель id которого равен

```
enter your request
find/1/*

action - find

id - 1
    all nodes - *
```

Обновление элемента по id

```
enter your request
update/1/[int=11]

action - update

id - 1
    left part - int
        condition - =
            right part - 11
```

Поиск элемента, удовлетворяющего условиям

```
enter your request
find/*/[int=10]|[bool=1]&[double>=5.0]

action - find

id - *
    left part - int
        condition - =
            right part - 10
                between pair - |
                    left part - bool
                        condition - =
                            right part - 1
                                between pair - &
                                    left part - double
                                        condition - >=
                                            right part - 5.0
```

Пример некорректного запроса

```
enter your request
add/1/[name=qwe][age=10][height=50.22][healthy=1]]
incorrect
```

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;
    char *star;
    struct attribute *attributes;
};
```

В этой структуре хранятся операция (add, remove, find, update), id родителя, маркер для поиска всех элементов, а также атрибуты. Атрибуты представлены односторонним списком для удобного переключения между ними. Сама структура attribute хранит в себе левую часть выражения, то есть condition, и правую часть выражения name = roma -> name (левая) = (condition) roma (правая). Также она хранит булевое сплетение выражений (| или &).

Операции:

- add - добавление
- remove - 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 примитивный. Он посимвольно идет по строке и заполняет структуру request. Посмотрим на реализацию парсера:

```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);
    request->operation = operation;

    if (!(
            strcmp("add", operation) == 0 ||
            strcmp("find", operation) == 0 ||
            strcmp("remove", operation) == 0 ||
            strcmp("update", operation) == 0
    ))
        print_error();

    char *id = read_word(&req, &path_length);
    request->parent_id = id;

    if (req[0] == '[') {
        struct attribute *attribute = malloc( size: sizeof(struct attribute));
        request->attributes = attribute;
        read_attributes(&req, &path_length, attribute);
    } else if (req[0] == '*') request->star = "*";

    return PARSE_OK;
}
```

Рекурсивно заполняем список атрибутов.

```c
void read_attributes(char **req, int *path_length, struct attribute *attribute) {

    if ((*req)[0] == '[') {

        char *left = read_left(req, path_length);
        char *cond = read_condition(req, path_length);
        char *right = read_word(req, path_length);

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

    if ((*req)[0] == ']') {
        remove_char(req, path_length);
        if (*path_length > 1) {
            struct attribute *new_attribute = malloc( size: sizeof(struct attribute));
            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;
        read_attributes(req, path_length, attribute);
    }
}
```

Вывод:

В процессе выполнения работы выяснилось, что оперативная память расходуется только на хранение структуры request. Я написал модули parser и view, а также протестировал написанную программу.