Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования
"Национальный Исследовательский Университет ИТМО"

Факультет Программной Инженерии и Компьютерных Технологий

Лабораторная работа №2

по дисциплине

«Низкоуровневое программирование»

Выполнил:
Студент группы P33302
Тюрин Святослав Вячеславович

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

В данной лабораторной работе:

- parser
- view

Считывается запрос с файла, затем он парсится, а после спаршенный запрос выводится в консоль.

Пример работы программы

Добавление элемента

```
Your request:
query {
    add {
        intgr = 10,
        dbl = 10.1,
        bln = 1,
        str = grg
    }
}

Operation: add
    Left operand: intgr
        Condition: =
            Right operand: 10
                Left operand: dbl
                    Condition: =
                        Right operand: 10.1
                            Left operand: bln
                                Condition: =
                                    Right operand: 1
                                        Left operand: str
                                            Condition: =
                                                Right operand: grg
```

Удаление элемента по id

```
Your request:
query {
    delete {
        id = 1
    }
}

Operation: delete
    Left operand: id
        Condition: =
            Right operand: 1
```

Поиск элемента по id

```
Your request:
query {
    find {
        id = 1
    }
}

Operation: find
    Left operand: id
        Condition: =
            Right operand: 1
```

Поиск всех элементов

```
Your request:
query {
    find {
        id = *
    }
}

Operation: find
    Left operand: id
        Condition: =
            Right operand: *
```

Поиск элементов, удовлетворяющих условию

```
Your request:
query {
    find {
        intgr = 10,
        |,
        dbl > 50.1
    }
}

Operation: find
    Left operand: intgr
        Condition: =
            Right operand: 10
                Combined condition: |
                    Left operand: dbl
                        Condition: >
                            Right operand: 50.1
```

Поиск всех элементов, присоединенных к node с id равным 1

```
Your request:
query {
    find {
        id^ = 1,
        id = *
    }
}

Operation: find
    Left operand: id^
        Condition: =
            Right operand: 1
                Left operand: id
                    Condition: =
                        Right operand: *
```

Обновление элемента по id

```
Your request:
query {
    update {
        id = 1,
        intgr = 10
    }
}

Operation: update
    Left operand: id
        Condition: =
            Right operand: 1
                Left operand: intgr
                    Condition: =
                        Right operand: 10
```

Присоединение двух nodes по id

```
Your request:
query {
    connect {
        id_1 = 1,
        id_2 = 2
    }
}

Operation: connect
    Left operand: id_1
        Condition: =
            Right operand: 1
                Left operand: id_2
                    Condition: =
                        Right operand: 2
```

Аспекты реализации

Структура для хранения запроса:

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

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

В request хранится оператор (add, find, delete, update, connect) и хранятся атрибуты. Структура атрибуты построена следующим образом: left – левая часть выражения, right – правая часть выражения, condition – отношение между левой и правой частью выражения, combined_condition – атрибуты могут быть связаны логическим И или ИЛИ, именно здесь хранится этот знак (|, &), если он есть. И структура attribute реализована односвязным списком, благодаря чему удобно итерироваться по ней.

Операции:

- add - добавление
- delete - удаление
- find - поиск
- update – обновление
- connect – присоединение nodes

Отношения между атрибутами и булевы знаки:

- = - равно
- != - не равно
- > - больше
- < - меньше
- >= - больше или равно
- <= - меньше или равно
- & - логическое И
- | - логическое ИЛИ

Выбор элементов:

- * - все элементы
- id^ - родительский элемент

Парсер

Решил написать свой парсер без подключения внешних библиотек.

Функция read_word – считывает символы до пробельного (слово).

```
char *read_word(char **req, int *path_length) {

    char *word = calloc( nmemb: MAX_STRING_SIZE,  size: sizeof(char));;

    int i;
    for (i = 0; i < *path_length; i++) {
        if ((*req)[i] == ' ' || (*req)[i] == ',') {
            if ((*req)[i] == ',') i--;
            break;
        }
        word[i] = (*req)[i];
    }
    i++;

    for (int j = 0; j < i; j++) {
        remove_char( size: path_length,  request_path: req);
    }
    return word;
}
```

Функция new_line – переставляет наш «курсор» на следующую строку.

```
void new_line(char **req, int *path_length) {
    int i;
    for (i = 0; i < *path_length; i++) {
        if ((*req)[i] == '\n') break;
    }
    i++;
    for (; i < *path_length; i++) {
        if ((*req)[i] != ' ') break;
    }

    for (int j = 0; j < i; j++) {
        remove_char( size: path_length,  request_path: req);
    }
}
```

В самом начале делается проверка на кол-во открывающихся и закрывающихся скобочек.

```c
void check_path(char *req, int *path_length) {
    int bracket = 0;

    for (int i = 0; i < *path_length; i++) {
        if (req[i] == '{') bracket++;
        if (req[i] == '}') bracket--;
    }

    if (bracket != 0) {
        print_error();
    }
}
```

Здесь происходит проверка на зарезервированные слова, которые юридически обязаны быть в запросе.

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

    int path_length = strlen( s: req);

    check_path( request_path: req,  path_size: &path_length);

    char *query = read_word(&req, &path_length);
    if (strcmp("query", query) != 0) print_error();

    new_line(&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 ||
            strcmp("connect", operation) == 0
    ))
        print_error();

    request->operation = operation;
```

Далее начинается считывание атрибутов запроса. Рекурсивно читаем строку за строкой, попутно заполняя односвязный список attributes.

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

    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] == ',') {

        new_line(req, path_length);
        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;
        new_line(req, path_length);
        read_attributes(req, path_length, attribute);
    }
    remove_char( size: path_length,  request_path: req);
}
```

Оперативная память

В лабораторной хранится только структура request. По окончании работы программы она отчищается.

Результаты

- Написан парсер
- Написано отображение request
- Приложение протестировано

Вывод: Задание выполнено в полном объёме.