Национальный исследовательский университет ИТМО

Факультет ПИиКТ

Лабораторная работа №2 по дисциплине «Низкоуровневое программирование»

Вариант: 4 (GraphQL)

Работу выполнил:

Асташин С. С.

Группа:

P33302

Преподаватель:

Кореньков Ю. Д.

Санкт-Петербург,

2023

Цель работы

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.

Задачи

1. Изучить выбранное средство синтаксического анализа
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:
        i. Условия:
            1. На равенство и неравенство для чисел, строк и булевских значений
            2. На строгие и нестрогие сравнения для чисел
            3. Существование подстроки
    c. Логическую комбинацию произвольного количества условий и булевских значений
    d. В качестве любого аргумента условий могут выступать литеральные значения (константы) или ссылки на значения, ассоциированные с элементами данных (поля, атрибуты, свойства)
    e. Разрешение отношений между элементами модели данных любых условий над сопрягаемыми элементами данных
3. Реализовать модуль, использующий средство синтаксического анализа для разбора языка запросов
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

Исходный код

https://github.com/Gramdel/llp_lab2

Описание работы

Список модулей:

- lexer.l – список правил для flex
- parser.y – список грамматики для bison
- main.c – вызов парсера (запуск программы)
- graphql_ast.c – создание, уничтожение и вывод дерева разбора

Начнём с лексического анализа. Вот такие токены распознаёт модуль:

```
%option yylineno noyywrap nounput noinput

%{
#include <stdio.h>
#include <stdbool.h>
#include "graphql_ast.h"
#include "parser.tab.h"
%}

%%
\(                      { return L_PARENTHESIS; }
\)                      { return R_PARENTHESIS; }
\{                      { return L_BRACE; }
\}                      { return R_BRACE; }
\[                      { return L_BRACKET; }
\]                      { return R_BRACKET; }
,                       { return COMMA; }
:                       { return COLON; }
select                  { return SELECT; }
insert                  { return INSERT; }
update                  { return UPDATE; }
delete                  { return DELETE; }
values                  { return VALUES; }
filter                  { return FILTER; }
eq                      { yylval.opType = OP_EQ_NODE; return COMPARE_OP; }
neq                     { yylval.opType = OP_NEQ_NODE; return COMPARE_OP; }
gt                      { yylval.opType = OP_GT_NODE; return COMPARE_OP; }
gte                     { yylval.opType = OP_GTE_NODE; return COMPARE_OP; }
le                      { yylval.opType = OP_LE_NODE; return COMPARE_OP; }
lee                     { yylval.opType = OP_LEE_NODE; return COMPARE_OP; }
like                    { yylval.opType = OP_LIKE_NODE; return LIKE_OP; }
and                     { yylval.opType = OP_AND_NODE; return LOGICAL_BOP; }
or                      { yylval.opType = OP_OR_NODE; return LOGICAL_BOP; }
not                     { yylval.opType = OP_NOT_NODE; return LOGICAL_UOP; }
true                    { yylval.boolVal = true; return BOOL; }
false                   { yylval.boolVal = false; return BOOL; }
[+-]?[0-9]+             { yylval.intVal = atoi(yytext); return INT; }
[+-]?[0-9]*\.[0-9]*     { yylval.doubleVal = atof(yytext); return DOUBLE; }
\"[^\"]*\"              { yylval.strVal = strdup(yytext); return STRING; }
[A-Za-z_][A-Za-z0-9_]*  { yylval.strVal = strdup(yytext); return NAME; }
\n                      { /* Считываем перенос строки, чтобы работал yylineno */ }
.                       { /* Игнорируем всё остальное */ }
%%
```

Потом bison, используя грамматику, составленную из этих токенов, собирает программу-автомат. Тут уже точно весь список не вставить, поэтому вставлю наиболее интересные правила – правила для команд insert и update:

```
insert_or_select_next: insert_next %prec COMMA { $$ = $1; }
                     | select_object L_BRACE insert_or_select_next R_BRACE { $$ = newQuerySetNode(newQueryNode(NESTED_QUERY_NODE, $1, $3)); }
                     | insert_or_select_next COMMA insert_or_select_next { addNextQueryToSet($1, $3); $$ = $1; }

insert_next: mutate_object { $$ = newQuerySetNode(newQueryNode(NESTED_QUERY_NODE, $1, NULL)); }
           | mutate_object L_BRACE insert_next R_BRACE { $$ = newQuerySetNode(newQueryNode(NESTED_QUERY_NODE, $1, $3)); }
           | insert_next COMMA insert_next { addNextQueryToSet($1, $3); $$ = $1; }

update_next: mutate_object { $$ = newQuerySetNode(newQueryNode(NESTED_QUERY_NODE, $1, NULL)); }
           | mutate_object L_BRACE update_next R_BRACE { $$ = newQuerySetNode(newQueryNode(NESTED_QUERY_NODE, $1, $3)); }
           | select_object L_BRACE update_next R_BRACE { $$ = newQuerySetNode(newQueryNode(NESTED_QUERY_NODE, $1, $3)); }
           | update_next COMMA update_next { addNextQueryToSet($1, $3); $$ = $1; }

select_object: schema_name { $$ = newObjectNode($1, NULL, NULL); }
             | schema_name L_PARENTHESIS filter R_PARENTHESIS { $$ = newObjectNode($1, NULL, $3); }

mutate_object: schema_name L_PARENTHESIS values R_PARENTHESIS { $$ = newObjectNode($1, $3, NULL); }
             | schema_name L_PARENTHESIS values COMMA filter R_PARENTHESIS { $$ = newObjectNode($1, $3, $5); }
```

Вставка и обновление

Не буду подробно описывать рекурсию, но главный аспект здесь в том, что insert должен до какого-то момента делать то же самое, что и select (то есть мы какое-то время спускаемся туда, куда надо делать вставку), и потом только вставлять, а update может как бы чередовать select и update, с единственным ограничением, что в самом низу запроса обязательно должно произойти обновление данных.

Наверно, на этом этапе можно переходить к дереву разбора. Оно состоит из узлов вида:

```
typedef struct astNode {
    astNode* left;
    astNode* right;
    nodeType type;
    union {
        int32_t intVal;
        double doubleVal;
        bool boolVal;
        char* strVal;
    };
} astNode;
```

Структура для узла

Свойствами узла являются тип и значение (union). Типы могут быть такие:

```
typedef enum nodeType {
    SELECT_QUERY_NODE,
    INSERT_QUERY_NODE,
    UPDATE_QUERY_NODE,
    DELETE_QUERY_NODE,
    NESTED_QUERY_NODE,
    QUERY_SET_NODE, //
    OBJECT_NODE,
    VALUES_NODE,
    ELEMENT_SET_NODE, /
    ELEMENT_NODE,
    KEY_NODE,
    INT_VAL_NODE,
    DOUBLE_VAL_NODE,
    BOOL_VAL_NODE,
    STR_VAL_NODE,
    FILTER_NODE,
    OP_EQ_NODE,
    OP_NEQ_NODE,
    OP_GT_NODE,
    OP_GTE_NODE,
    OP_LE_NODE,
    OP_LEE_NODE,
    OP_LIKE_NODE,
    OP_AND_NODE,
    OP_OR_NODE,
    OP_NOT_NODE,
} nodeType;
```

Тип узла

Отдельно отмечу, что типы QUERY_SET_NODE и ELEMENT_SET_NODE являются обёртками над узлами типа QUERY_NODE и ELEMENT_NODE соответственно. Они нужны для составления списков запросов и элементов (node->left в обёртке хранит указатель на то, что она оборачивает, а node->right – на следующую обёртку).

Прочие аспекты реализации

В принципе, большинство проверок уже “вшиты” в грамматику. Единственное, что пришлось добавить дополнительно – проверка на то, что названия схем и ключей не длиннее 12 символов (ограничение из первой лабораторной). Сделано это так:

```
schema_name: NAME { if(strlen($1) > MAX_NAME_LENGTH) { yyerror("name of schema is too long"); YYABORT; } $$ = $1; }

key: NAME { if(strlen($1) > MAX_NAME_LENGTH) { yyerror("key is too long"); YYABORT; } $$ = newKeyNode($1); }
```

Результаты

```
user@llp-ubuntu:~/Desktop/llp/llp_lab2$ ./main
insert {
    root(values: [{rootInt1 : 111}, {rootInt2 : -999}], filter: eq(rootInt1, "value")) {
        child
    }
}
Error on line 4: syntax error
```

Неверный insert (на нижнем уровне ничего не вставляется)

```
user@llp-ubuntu:~/Desktop/llp/llp_lab2$ ./main
insert {
    root(values: [{rootInt1 : 111}, {rootInt2 : -999}], filter: eq(rootInt1, "value")) {
        child(values: [{childInt1 : 111}])
    }
}
QueryType: Insert
QuerySet:
  Query:
    Object:
      SchemaName: root
      NewValues:
        ElementSet:
          Element:
            Key: rootInt1
            Value:
              ValueType: Integer
              Data: 111
          Element:
            Key: rootInt2
            Value:
              ValueType: Integer
              Data: -999
      Filter:
        Operation:
          OperationType: Equal
          Key: rootInt1
          Value:
            ValueType: String
            Data: "value"
    QuerySet:
      Query:
        Object:
          SchemaName: child
          NewValues:
            ElementSet:
              Element:
                Key: childInt1
                Value:
                  ValueType: Integer
                  Data: 111
          Filter: <undefined>
        QuerySet: <undefined>
```

Верный insert

```
user@llp-ubuntu:~/Desktop/llp/llp_lab2$ ./main
select {
    root(values: [{childBool : true}], filter: eq(rootInt1, "value"))
}
Error on line 2: syntax error
```

Неверный select (при выборке нельзя передать данные для вставки)

```
user@llp-ubuntu:~/Desktop/llp/llp_lab2$ ./main
select {
    root {
        child
    },
    grChild
}
QueryType: Select
QuerySet:
  Query:
    Object:
      SchemaName: root
      NewValues: <undefined>
      Filter: <undefined>
    QuerySet:
      Query:
        Object:
          SchemaName: child
          NewValues: <undefined>
          Filter: <undefined>
        QuerySet: <undefined>
  Query:
    Object:
      SchemaName: grChild
      NewValues: <undefined>
      Filter: <undefined>
    QuerySet: <undefined>
```

Верный select

Обратите внимание: запрос выше является верным, поскольку в данном случае по сути выполняется два запроса – “достать child из корня root” и “достать корень grChild”). Очевидно, что один из этих подзапросов не выполнится (поскольку название у корня только одно), но с точки зрения синтаксиса – запрос верен.

```
user@llp-ubuntu:~/Desktop/llp/llp_lab2$ ./main
update {
    root {
        child(values: [{childBool : true}])
    }
}
QueryType: Update
QuerySet:
  Query:
    Object:
      SchemaName: root
      NewValues: <undefined>
      Filter: <undefined>
    QuerySet:
      Query:
        Object:
          SchemaName: child
          NewValues:
            ElementSet:
              Element:
                Key: childBool
                Value:
                  ValueType: Boolean
                  Data: true
          Filter: <undefined>
        QuerySet: <undefined>
```

Верный update

```
user@llp-ubuntu:~/Desktop/llp/llp_lab2$ ./main
update {
    root(values: [{rootInt1 : 111}, {rootInt2 : -999}], filter: eq(rootInt1, "value")) {
        child
    }
}
Error on line 4: syntax error
```

Неверный update (на нижнем уровне ничего не обновляется)

```
user@llp-ubuntu:~/Desktop/llp/llp_lab2$ ./main
delete {
    root {
        child
    },
    grChild
}
QueryType: Delete
QuerySet:
  Query:
    Object:
      SchemaName: root
      NewValues: <undefined>
      Filter: <undefined>
    QuerySet:
      Query:
        Object:
          SchemaName: child
          NewValues: <undefined>
          Filter: <undefined>
        QuerySet: <undefined>
  Query:
    Object:
      SchemaName: grChild
      NewValues: <undefined>
      Filter: <undefined>
    QuerySet: <undefined>
```

Верный delete

Выводы

В ходе выполнения данной лабораторной работы были изучены (на базовом уровне) Flex и Bison. На основе грамматики GraphQL была построена своя грамматика для модуля. Основной проблемой, которую пришлось решить на этом этапе работы, были shift/reduce конфликты, в основном связанные со спецификой запросов insert и update (как я уже упомянул выше, там есть ограничения на то, какое действие должно идти за каким). Также, был реализован модуль, строящий и выводящий дерево разбора полученного запроса. С ним, в целом, проблем не возникло, хотя пришлось придумать, как хранить списки, сохраняя при этом бинарность дерева.