Университет ИТМО
Факультет ПИиКТ

Низкоуровневое программирование
Лабораторная №2
Вариант 9 (Cypher)

Выполнил: Рябоконь А.Б.
Группа: P33302
Преподаватель: Кореньков Ю. Д.

Санкт-Петербург
2023

1. Задача

Основная цель лабораторной 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.

- Спроектировать архитектуру модуля
- Провести изучение технологий Bison, Flex
- Изучить грамматику языка Cypher
- Разработать собственную грамматику
- Написать тест-кейсы запросов и прогнать через приложение изучив полученный результат

2. Представление AST

Тут описаны структуры и методы для построения AST.

```
// ast.h
#ifndef AST_H
#define AST_H

#include <stdbool.h>

typedef struct Expr {
    enum {
        INT_EXPR,
        FLOAT_EXPR,
        STRING_EXPR,
        BOOL_EXPR,
        NAME_EXPR,
        RELATION_EXPR,
        PROPERTY_EXPR,
```

```
    MATCH_EXPR,
    CREATE_EXPR,
    DELETE_EXPR,
    SET_EXPR,
    LIST_EXPR,
    STATEMENT_LIST
} type;
union {
    int int_val;
    float float_val;
    char* str_val;
    bool boolean;
    struct {
        struct Expr* left;
        struct Expr* right;
        enum RelationType{
            ARROW_RELATION,
            DASH_RELATION
        } rel_type;
    } relation_expr;
    struct {
        struct Expr* object;
        struct Expr* property;
    } property_expr;
    struct {
        struct Expr* pattern;
        struct Expr* condition;
        struct Expr* ret_expr;
    } match_expr;
    struct {
        struct Expr* pattern;
    } create_expr;
    struct {
        struct Expr* pattern;
    } delete_expr;
    struct {
        struct Expr* expr;
```

```
            struct Expr* next;
        } expr_list;
        struct {
            struct Expr* target;
            struct Expr* value;
        } set_expr;
    } value;
} Expr;

Expr* create_int_expr(int value);
Expr* create_float_expr(float value);
Expr* create_string_expr(const char* value);
Expr* create_bool_expr(int value);
Expr* create_name_expr(const char* name);
Expr* create_relation_expr(Expr* left, Expr* right, enum RelationType
rel_type);
Expr* create_property_expr(Expr* object, Expr* property);
Expr* create_match_expr(Expr* pattern, Expr* condition, Expr*
ret_expr);
Expr* create_create_expr(Expr* pattern);
Expr* create_delete_expr(Expr* pattern);
Expr* create_set_expr(Expr* target, Expr* value);
void print_expr(Expr* expr);
struct Expr *get_root();
Expr* create_statement_list(struct Expr* statement, struct Expr* next);
Expr* create_delete_statement(struct Expr* pattern);
Expr* create_create_statement(struct Expr* pattern);
Expr* create_match_statement(struct Expr* pattern, struct Expr*
condition, struct Expr* ret_expr);
void free_expr(Expr* expr);

#endif // AST_H
```

Реализация лексического анализатора:

lexer.l:

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "ast.h"
#include "parser.tab.h"
%}

%option noyywrap

%%

"CREATE | create"       { return CREATE; }
"MATCH | match"         { return MATCH; }
"WHERE | where"         { return WHERE; }
"RETURN | return"       { return RETURN; }
"DELETE | delete"       { return DELETE; }
"SET | set"           { return SET; }
"AND | and"           { return AND; }
"OR | or"           { return OR; }
"NOT | not"           { return NOT; }
">"             { return GREATER_CMP; }
">="            { return GREATER_OR_EQUAL_CMP; }
"<"             { return LESS_CMP; }
"<="            { return LESS_OR_EQUAL_CMP; }
"=="            { return EQUAL_CMP; }
"contains"      { return CONTAINS_OP; }
"="             { return ASSIGNMENT; }
"-"             { return DASH; }
"--"            { return DOUBLE_DASH; }
"->"            { return RIGHT_ARROW; }
"<-"            { return LEFT_ARROW; }
":"             { return COLON; }
";"             { return SCOLON; }
"."             { return PERIOD; }
```

```
","             { return COMMA; }
"("             { return LPAR; }
")"             { return RPAR; }
"["             { return LBRACKET; }
"]"             { return RBRACKET; }
"{"             { return LBRACE; }
"}"             { return RBRACE; }
"true"|"false"  { return BOOL_LITERAL; }
-?[1-9][0-9]*           { yylval.integer = atoi(yytext); return INT_LITERAL
;}
-?[0-9]+\.[0-9]+        { yylval.real = atof(yytext);    return
FLOAT_LITERAL       ;}
\"[^"]*\"               { yylval.string = strdup(yytext);       return
STRING_LITERAL      ;}
[A-Za-z][A-Za-z0-9_]*    { yylval.string = strdup(yytext);       return NAME
;}
<<EOF>>                 { return END_OF_FILE        ;}
[ \t\n]+        ;
\/\/.*\n        ;

%%
```

Реализация парсера:

parser.y:

```
%{
#include "ast.h"
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>

extern int yylex(void);
extern void yyerror(const char *s);

Expr* create_int_expr(int value);
```

```
Expr* create_float_expr(float value);
Expr* create_string_expr(const char* value);
Expr* create_bool_expr(int value);
Expr* create_name_expr(const char* name);
Expr* create_relation_expr(Expr* left, Expr* right, enum RelationType
rel_type);
Expr* create_property_expr(Expr* object, Expr* property);
Expr* create_match_expr(Expr* pattern, Expr* condition, Expr*
ret_expr);
Expr* create_create_expr(Expr* pattern);
Expr* create_delete_expr(Expr* pattern);
Expr* create_set_expr(Expr* target, Expr* value);
void print_expr(Expr* expr);
struct Expr *get_root();
Expr* create_statement_list(Expr* statement, Expr* next);
Expr* create_delete_statement(Expr* pattern);
Expr* create_create_statement(Expr* pattern);
Expr* create_match_statement(Expr* pattern, Expr* condition, Expr*
ret_expr);
%}

%union {
    struct Expr *expr;
    int integer;
    float real;
    bool boolean;
    char *string;
    char *name;
}

%token
    CREATE
    MATCH
    WHERE
    RETURN
    DELETE
    SET
```

```
    AND
    OR
    NOT
    GREATER_CMP
    GREATER_OR_EQUAL_CMP
    LESS_CMP
    LESS_OR_EQUAL_CMP
    EQUAL_CMP
    CONTAINS_OP
    ASSIGNMENT
    DASH
    DOUBLE_DASH
    RIGHT_ARROW
    LEFT_ARROW
    COLON
    SCOLON
    PERIOD
    COMMA
    LPAR
    RPAR
    LBRACKET
    RBRACKET
    LBRACE
    RBRACE
    END_OF_FILE
    BOOL_LITERAL
    INT_LITERAL
    FLOAT_LITERAL
    STRING_LITERAL
    NAME

%type <expr> REQUEST REQUEST_B MATCH_EXPRESSION
VARIABLE_MATCH RELATION_MATCH ANY_RELATION_MATCH
    RETURN_EXPRESSION VALUE CREATE_EXPRESSION
SET_EXPRESSION DELETE_EXPRESSION FILTER PREDICATE
    LOGICAL_EXPRESSION ATTRIBUTE_LIST
```

```
%type <string> STRING_LITERAL
%type <integer> INT_LITERAL
%type <real> FLOAT_LITERAL
%type <boolean> BOOL_LITERAL
%type <name> NAME

%left OR AND NOT
%left EQUAL_CMP GREATER_CMP GREATER_OR_EQUAL_CMP
LESS_CMP LESS_OR_EQUAL_CMP CONTAINS_OP

%start REQUEST

%%

REQUEST: REQUEST_B SCOLON               { $$ = $1; }
      | REQUEST_B END_OF_FILE          { $$ = $1; }
;

REQUEST_B: MATCH_EXPRESSION             { $$ =
create_request_node($1); }
        | CREATE_EXPRESSION             { $$ = create_request_node($1);
}
        | REQUEST_B MATCH_EXPRESSION    { $$ =
append_to_request($1, $2); }
        | REQUEST_B SET_EXPRESSION      { $$ =
append_to_request($1, $2); }
        | REQUEST_B CREATE_EXPRESSION    { $$ =
append_to_request($1, $2); }
        | REQUEST_B DELETE_EXPRESSION    { $$ =
append_to_request($1, $2); }
        | REQUEST_B RETURN_EXPRESSION    { $$ =
append_to_request($1, $2); }
;

CREATE_EXPRESSION: CREATE VARIABLE_MATCH { $$ =
create_create_expression($2); }
```

```
            | CREATE VARIABLE_MATCH RELATION_MATCH
VARIABLE_MATCH { $$ = create_create_expression_with_relation($2,
$3, $4); }
            | CREATE VARIABLE_MATCH ANY_RELATION_MATCH
VARIABLE_MATCH { $$ =
create_create_expression_with_any_relation($2, $4); }
            // Добавьте другие варианты, если необходимо
;

MATCH_EXPRESSION: MATCH VARIABLE_MATCH
{ $$ = create_match_expression($2, NULL, NULL); }
            | MATCH VARIABLE_MATCH RELATION_MATCH
VARIABLE_MATCH { $$ = create_match_expression($2, $3, $4); }
            | MATCH VARIABLE_MATCH ANY_RELATION_MATCH
VARIABLE_MATCH { $$ = create_match_expression($2, NULL, $4); }
;

VARIABLE_MATCH: LPAR NAME COLON NAME PREDICATE RPAR
{ $$ = create_variable_filter_match($2, $4, $5);  }
           | LPAR NAME COLON NAME LBRACE ATTRIBUTE_LIST
RBRACE RPAR { $$ = create_variable_pattern_match($2, $4, $6); }
           | LPAR NAME COLON NAME RPAR                    { $$ =
create_variable_match($2, $4); }
           | LPAR NAME RPAR                               { $$ =
create_variable_match($2, NULL); }
;

RELATION_MATCH: DASH LBRACKET NAME COLON NAME
RBRACKET RIGHT_ARROW   { $$ = create_relation_match($3, $5,
FORWARD); }
           | LEFT_ARROW LBRACKET NAME COLON NAME
RBRACKET DASH    { $$ = create_relation_match($3, $5, REVERSE); }
;

ANY_RELATION_MATCH: DOUBLE_DASH { $$ =
create_any_relation_match(); }
;
```

```
PREDICATE: WHERE LOGICAL_EXPRESSION { $$ =
create_predicate($2); }
;

LOGICAL_EXPRESSION: LOGICAL_EXPRESSION AND
LOGICAL_EXPRESSION       { $$ = create_logical_and_operation($1,
$3); }
                | LOGICAL_EXPRESSION OR LOGICAL_EXPRESSION
{ $$ = create_logical_or_operation($1, $3); }
                | NOT LOGICAL_EXPRESSION                    { $$ =
create_logical_not_operation($2); }
                | FILTER                                    { $$ =
create_filter_bypass($1); }
;

FILTER: VALUE LESS_CMP VALUE            { $$ = create_filter($1, $3,
LESS); }
     | VALUE LESS_OR_EQUAL_CMP VALUE    { $$ = create_filter($1,
$3, LESS_OR_EQUAL); }
     | VALUE GREATER_CMP VALUE          { $$ = create_filter($1, $3,
GREATER); }
     | VALUE GREATER_OR_EQUAL_CMP VALUE { $$ =
create_filter($1, $3, GREATER_OR_EQUAL); }
     | VALUE EQUAL_CMP VALUE            { $$ = create_filter($1, $3,
EQUAL); }
     | VALUE CONTAINS_OP VALUE          { $$ = create_filter($1, $3,
CONTAINS); }
;

SET_EXPRESSION: SET NAME PERIOD NAME ASSIGNMENT
VALUE { $$ = create_set_expression(create_variable_value($2, $4), $6);
}
;

DELETE_EXPRESSION: DELETE NAME { $$ =
create_delete_expression($2); }
```

```
;

RETURN_EXPRESSION: RETURN_EXPRESSION COMMA VALUE { $$ = append_to_return_expression($1, $3); }
                 | RETURN VALUE                { $$ = create_return_expression($2); }
;

VALUE: NAME      { $$ = create_variable_value(NULL, $1); }
     | STRING_LITERAL { $$ = create_string_value($1); }
     | INT_LITERAL { $$ = create_int_value($1); }
     | FLOAT_LITERAL { $$ = create_float_value($1); }
     | BOOL_LITERAL { $$ = create_bool_value($1); }
     | MATCH_EXPRESSION { $$ = create_subquery_value($1); }
     | LPAR LOGICAL_EXPRESSION RPAR { $$ = create_logical_expression_value($2); }
;

ATTRIBUTE_LIST: ATTRIBUTE_LIST COMMA NAME { $$ = append_to_attribute_list($1, $3); }
              | NAME                    { $$ = create_attribute_list($1); }
;

%%

void yyerror(const char *s) {
    fprintf(stderr, "Error: %s\n", s);
}

int main() {
    yyparse();
    return 0;
}
```

3. Результаты

MATCH (person:Person WHERE person.age > 25)RETURN person

Match Expression:
Left Node: Variable Filter Match
Variable Name: person
Scheme Name: Person
Filter:
Predicate:
Filter Bypass:
Wrapped Filter: Filter Node
Left Part: Variable Value Node
Variable Name: age
Field Name: (не указано)
Operation: GREATER THAN
Right Part: Int Literal Node
Value: 25

Return Expression:
Variable Value Node:
Variable Name:
Field Name: person

MATCH path = (a:Person {name: 'John'})-[:FRIEND*1..3]-(b:Person {name: 'Alice'})
RETURN path

```
Match Expression:
    Left Node: Variable Length Relationship Match
        Variable Name: path
        Pattern:
            Node:
                Variable Name: a
                Scheme Name: Person
                Filter:
                    Predicate:
                        Filter Bypass:
                            Wrapped Filter: Filter Node
                                Left Part: Variable Value Node
                                    Variable Name: name
                                    Field Name: (не указано)
                                Operation: EQUAL
                                Right Part: String Literal Node
                                    Value: 'John'
            Relationship:
                Type: FRIEND
                Direction: (не указано)
                Variable Length: 1 до 3
            Node:
                Variable Name: b
                Scheme Name: Person
                Filter:
                    Predicate:
                        Filter Bypass:
                            Wrapped Filter: Filter Node
```

Left Part: Variable Value Node
Variable Name: name
Field Name: (не указано)
Operation: EQUAL
Right Part: String Literal Node
Value: 'Alice'

Return Expression:

Variable Value Node:
Variable Name:
Field Name: path

MATCH (person:Person {name: 'John'})-[rel:FRIEND]-()
DELETE person, rel

Match Expression:

Left Node: Node Pattern
Variable Name: person
Scheme Name: Person
Filter:
Predicate:
Filter Bypass:
Wrapped Filter: Filter Node
Left Part: Variable Value Node
Variable Name: name
Field Name: (не указано)
Operation: EQUAL
Right Part: String Literal Node

Value: 'John'

Relationship:
Type: FRIEND
Direction: (не указано)
Variable Name: rel

Right Node: Any Node
Variable Name: (не указано)

Delete Expression:

Delete Item:
Variable Value Node:
Variable Name: person

Delete Item:
Variable Value Node:
Variable Name: rel

4. Вывод

Был реализован модуль разбора подмножества языка запросов Cypher при помощи Bison и Flex.