МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

ЛАБОРАТОРНАЯ РАБОТА №2

по дисциплине
“Низкоуровневое программирование”

Вариант 9

Cypher

Студент:
Смирнова А. Ю.
Группа P33301

Преподаватель:
Кореньков Юрий Дмитриевич

Санкт-Петербург, 2023

Задание

Задание 2

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)

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

Выполнение

Программа реализована с помощью средств лексического и синтаксического анализа flex и bison.

Для начала я создала лексер lexer.l, где перечислила валидные токены.

```
WS        [ \t\n]+
NAME      [A-Za-z][A-Za-z0-9_]*
INTEGER   -?[1-9][0-9]*
FLOAT     -?[0-9]+\.[0-9]+
STRING    \"[^"]*\"
COMMENT \/\/.*\n

%%
"MATCH"|"match"         {return MATCH_KEYWORD       ;}
"WHERE"|"where"         {return WHERE_KEYWORD       ;}
"RETURN"|"return"       {return RETURN_KEYWORD      ;}
"CREATE"|"create"       {return CREATE_KEYWORD      ;}
"DELETE"|"delete"       {return DELETE_KEYWORD      ;}
"SET"|"set"             {return SET_KEYWORD         ;}
"AND"|"and"             {return AND_KEYWORD         ;}
"OR"|"or"               {return OR_KEYWORD          ;}
"NOT"|"not"             {return NOT_KEYWORD         ;}
">"                     {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        ;}
{INTEGER}               {yylval.integer = atoi(yytext); return INT_LITERAL       ;}
{FLOAT}                 {yylval.real = atof(yytext);    return FLOAT_LITERAL     ;}
{STRING}                {yylval.string = strdup(yytext);return STRING_LITERAL    ;}
{NAME}                  {yylval.string = strdup(yytext);        return NAME      ;}
<<EOF>>                 {return END_OF_FILE         ;}
{COMMENT}               // dp nothing
{WS}                    // do nothing

. {
    return (int) yytext[0];
}
```

В файле parser.y были описаны грамматические правила, по которым необходимо парсить получаемый запрос.

```
%union
{
    INode                   *iNode;
    RequestNode             *requestNode;
    ExpressionNode          *expressionNode;
    MatchExpressionNode     *matchExpressionNode;
    CreateExpressionNode    *createExpressionNode;
    SetExpressionNode       *setExpressionNode;
    ReturnExpressionNode    *returnExpressionNode;
    DeleteExpressionNode    *deleteExpressionNode;
    VariableMatchNode       *variableMatchNode;
    RelationMatchNode       *relationMatchNode;
    ValueNode               *value;
    FilterNode              *filterNode;
    PredicateNode           *predicateNode;
    LogicalExpressionNode   *logicalExpressionNode;
    AttributeListNode       *attributeListNode;
    char                    *name;
    char                    *string;
    int                     integer;
    float                   real;
    bool                    boolean;
}

%token
    MATCH_KEYWORD
    WHERE_KEYWORD
    RETURN_KEYWORD
    CREATE_KEYWORD
    DELETE_KEYWORD
    SET_KEYWORD
    AND_KEYWORD
    OR_KEYWORD
    NOT_KEYWORD
    GREATER_CMP
    GREATER_OR_EQUAL_CMP
    LESS_CMP
    LESS_OR_EQUAL_CMP
    EQUAL_CMP
    CONTAINS_OP
    ASSIGNMENT
    DASH
    RIGHT_ARROW
    LEFT_ARROW
    DOUBLE_DASH
    COLON
    SCOLON
    PERIOD
    COMMA
    LPAR
    RPAR
    LBRACKET
    RBRACKET
```

```
    LBRACE
    RBRACE
    END_OF_FILE
;

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

%nterm <requestNode>              REQUEST
%nterm <requestNode>              REQUEST_B
%nterm <matchExpressionNode>      MATCH_EXPRESSION
%nterm <variableMatchNode>        VARIABLE_MATCH
%nterm <relationMatchNode>        ANY_RELATION_MATCH
%nterm <relationMatchNode>        RELATION_MATCH
%nterm <returnExpressionNode>     RETURN_EXPRESSION
%nterm <value>                    VALUE
%nterm <createExpressionNode>     CREATE_EXPRESSION
%nterm <setExpressionNode>        SET_EXPRESSION
%nterm <deleteExpressionNode>     DELETE_EXPRESSION
%nterm <filterNode>               FILTER
%nterm <predicateNode>            PREDICATE
%nterm <logicalExpressionNode>    LOGICAL_EXPRESSION
%nterm <attributeListNode>        ATTRIBUTE_LIST

%left OR_KEYWORD
%left AND_KEYWORD
%left NOT_KEYWORD

%start REQUEST

%%
REQUEST: REQUEST_B SCOLON              { insert($1); return 0; }
       | REQUEST_B END_OF_FILE         { insert($1); return 0; }
;


REQUEST_B: MATCH_EXPRESSION            { $$ =
RequestNode_new((ExpressionNode*)$1); }
         | CREATE_EXPRESSION           { $$ =
RequestNode_new((ExpressionNode*)$1); }
         | REQUEST_B MATCH_EXPRESSION  { $$ = $1; RequestNode_addExpr($$,
(ExpressionNode*)$2); }
         | REQUEST_B SET_EXPRESSION    { $$ = $1; RequestNode_addExpr($$,
(ExpressionNode*)$2); }
         | REQUEST_B CREATE_EXPRESSION { $$ = $1; RequestNode_addExpr($$,
(ExpressionNode*)$2); }
         | REQUEST_B DELETE_EXPRESSION { $$ = $1; RequestNode_addExpr($$,
(ExpressionNode*)$2); }
         | REQUEST_B RETURN_EXPRESSION { $$ = $1; RequestNode_addExpr($$,
(ExpressionNode*)$2); }
;

MATCH_EXPRESSION: MATCH_KEYWORD VARIABLE_MATCH
{ $$ = MatchExpressionNode_new($2);         }
```

```
                | MATCH_KEYWORD VARIABLE_MATCH RELATION_MATCH VARIABLE_MATCH
{ $$ = MatchExpressionNode_new3($2, $4, $3); }
                | MATCH_KEYWORD VARIABLE_MATCH ANY_RELATION_MATCH VARIABLE_MATCH
{ $$ = MatchExpressionNode_new2($2, $4); }
;

VARIABLE_MATCH: LPAR NAME COLON NAME PREDICATE RPAR                    { $$ =
(VariableMatchNode *)VariableFilterMatchNode_new($2, $4, $5);  }
              | LPAR NAME COLON NAME LBRACE ATTRIBUTE_LIST RBRACE RPAR { $$ =
(VariableMatchNode *)VariablePatternMatchNode_new($2, $4, $6); }
              | LPAR NAME COLON NAME RPAR                              { $$ =
VariableMatchNode_new($2, $4); }
              | LPAR NAME RPAR                                         { $$ =
VariableMatchNode_new($2, ""); }
;

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

ANY_RELATION_MATCH: DOUBLE_DASH { $$ = RelationMatchNode_new("", "", ANY); }
;

PREDICATE: WHERE_KEYWORD LOGICAL_EXPRESSION { $$ = PredicateNode_new($2); }
;

LOGICAL_EXPRESSION: LOGICAL_EXPRESSION AND_KEYWORD LOGICAL_EXPRESSION { $$ =
(LogicalExpressionNode *)AndOperationNode_new($1, $3); }
                  | LOGICAL_EXPRESSION OR_KEYWORD LOGICAL_EXPRESSION  { $$ =
(LogicalExpressionNode *)OrOperationNode_new($1, $3);  }
                  | NOT_KEYWORD LOGICAL_EXPRESSION                    { $$ =
(LogicalExpressionNode *)NotOperationNode_new($2);     }
                  | FILTER                                            { $$ =
(LogicalExpressionNode *)FilterByPassNode_new($1);     }
;

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

SET_EXPRESSION: SET_KEYWORD NAME PERIOD NAME ASSIGNMENT VALUE { $$ =
SetExpressionNode_new(VariableValueNode_new($2, $4), $6); }
;

DELETE_EXPRESSION: DELETE_KEYWORD NAME { $$ = DeleteExpressionNode_new($2); }
```

```
;

RETURN_EXPRESSION: RETURN_EXPRESSION COMMA VALUE { $$ = $1;
ReturnExpressionNode_addElement($$, $3);                          }
                 | RETURN_KEYWORD VALUE          { $$ =
ReturnExpressionNode_new(); ReturnExpressionNode_addElement($$, $2); }

ATTRIBUTE_LIST: NAME COLON VALUE COMMA ATTRIBUTE_LIST { $$ = $5;
AttributeListNode_addAttribute($$, $1, $3);                       }
              | NAME COLON VALUE                    { $$ =
AttributeListNode_new(); AttributeListNode_addAttribute($$, $1, $3); }
;

CREATE_EXPRESSION: CREATE_KEYWORD VARIABLE_MATCH { $$ =
CreateExpressionNode_new($2); }
                 | CREATE_KEYWORD VARIABLE_MATCH RELATION_MATCH VARIABLE_MATCH {
$$ = CreateExpressionNode_new2($2, $4, $3); }
;

VALUE: NAME             { $$ = (ValueNode *)VariableValueNode_new($1, ""); }
     | BOOL_LITERAL     { $$ = (ValueNode *)BoolLiteralNode_new($1);       }
     | INT_LITERAL      { $$ = (ValueNode *)IntLiteralNode_new($1);        }
     | FLOAT_LITERAL    { $$ = (ValueNode *)FloatLiteralNode_new($1);      }
     | STRING_LITERAL   { $$ = (ValueNode *)StringLiteralNode_new($1);     }
     | NAME PERIOD NAME { $$ = (ValueNode *)VariableValueNode_new($1, $3); }
;
```

В файле ast.h описаны структуры и методы для построения AST. Были реализованы наследование и методы, которые имитируют конструкторы. Приведу в пример лишь часть струтур, их там много.

```
ypedef struct INode {
    void (*print)(struct INode* node, int level, FILE* out);
} INode;

typedef struct ExpressionNode {
    INode base;
    void (*print)(struct INode* node, int level, FILE* out);
} ExpressionNode;

typedef struct ValueNode {
    INode base;
    void (*print)(struct INode* node, int level, FILE* out);
} ValueNode;

typedef struct VariableValueNode {
    ValueNode base;
    char* VariableName;
    char* FieldName;
    void (*print)(int level, const char* VariableName, const char* FieldName);
} VariableValueNode;

VariableValueNode* VariableValueNode_new(char* VName, char* FName);
```

```
typedef struct StringLiteralNode {
    ValueNode base;
    char* Value;
    void (*print)(int level, const char* Value);
} StringLiteralNode;

typedef struct DeleteExpressionNode {
    ExpressionNode base;
    char* VariableName;
    void(*print)(int level, const char* VariableName);
} DeleteExpressionNode;

DeleteExpressionNode* DeleteExpressionNode_new(char* Name);

typedef struct CreateExpressionNode {
    ExpressionNode base;
    VariableMatchNode* LeftNode;
    VariableMatchNode* RightNode;
    RelationMatchNode* Relation;
    void(*print)(int level, const char* LeftNode, const char* Relation, const
char* RightNode);

} CreateExpressionNode;

CreateExpressionNode* CreateExpressionNode_new(VariableMatchNode* Node);

CreateExpressionNode* CreateExpressionNode_new2(VariableMatchNode* Left,
VariableMatchNode* Right, RelationMatchNode* Rel);

typedef enum RelationDirection {
    FORWARD,
    REVERSE,
    ANY
} RelationDirection;

typedef struct RelationMatchNode {
    INode base;
    char* VariableName;
    char* RelationName;
    void (*print)(int level, const char* VariableName, const char* RelationName,
const char* Direction);
    RelationDirection Direction;
} RelationMatchNode;
```

Для представления результата в виде дерева использованы методы, описанные в файле printer.h

```
void BoolLiteralNode_print(int level, int Value);
void IntLiteralNode_print(int level, int Value);
void FloatLiteralNode_print(int level, float Value);
void FilterNode_print(int level, const char* Operation, const char* LHS, const
char* RHS);
void FilterByPassNode_print(int level, const char* Wrapped);
void NotOperationNode_print(int level, const char* Operand);
void AndOperationNode_print(int level, const char* LHS, const char* RHS);
void OrOperationNode_print(int level, const char* LHS, const char* RHS);
void PredicateNode_print(int level, const char* Body);
void AttributeListNode_print(int level, const char* AttrList);
void VariableMatchNode_print(int level, const char* VariableName, const char*
SchemeName);
void VariablePatternMatchNode_print(int level, const char* VariableName, const
char* SchemeName, const char* Pattern);
void VariableFilterMatchNode_print(int level, const char* VariableName, const
char* SchemeName, const char* Predicate);
void RelationMatchNode_print(int level, const char* VariableName, const char*
RelationName, const char* Direction);
void MatchExpressionNode_print(int level, const char* LeftNode, const char*
Relation, const char* RightNode);
void ReturnExpressionNode_print(struct INode* node, int level, FILE* out);
void DeleteExpressionNode_print(int level, const char* VariableName);
void CreateExpressionNode_print(int level, const char* LeftNode, const char*
Relation, const char* RightNode);
void SetExpressionNode_print(int level, const char* Dest, const char* Src);
void RequestNode_print(struct INode* node, int level, FILE* out);
void insert(RequestNode *v);
```

```
void insert(RequestNode *v) {
    printf("Input received. Check results\n");
    v->base.print = RequestNode_print;
    INode_print(&(v->base), 0, stdout);
}
```

И в файле parser.y

```
int main(int argc, char *argv[]) {
  --argc;
  ++argv;
  if (argc == 0) {
    printf("Usage: ./parser <file.txt>\n");
    return 1;
  }

  FILE *file = fopen(argv[0], "r");
  if (file == NULL) {
    printf("Unable to open file: %s\n", argv[0]);
    return 1;
  }
  yyin = file;
  if (yyparse()) {
    printf("Parsing failed\n");
```

```
  }
  else {
    printf("No syntax errors\n");
  }
}
```

Результаты

Равенство и неравенство чисел, строк, булевских значений

```
MATCH (a:Actor WHERE a.birth_date > 2000)
RETURN a
```

```
Request: {
    Match expression: {
        Left node: Variable filter match: {
            Variable name: a
            Scheme name: Actor
            Filter: Predicate: {
                expr: Filter bypass: {
                    Wrapped filter: Filter node: {
                        Left part: Variable value node: {
                            Variable name: a
                            Field name: birth_date
                        }
                        Operation: GREATER
                        Right part: Int Literal node: {
                            Value: 2000
                        }
                    }
                }
            }
        }
    }
    Return Expression: {
        Variable value node: {
            Variable name: a
            Field name: name
        }
    }
}
```

```
MATCH (a:Actor {name: "Anisia"})
RETURN a
```

```
Request: {
    Match expression: {
        Left node: Variable pattern match: {
            Variable name: a
            Scheme name: Actor
            Pattern: Attribute List: {
                Attribute name: name
                Value: String Literal node: {
                    Value: "Anisia"
                }
            }
        }
    }
    Return Expression: {
        Variable value node: {
            Variable name: a
            Field name:
        }
    }
}
```

```
MATCH (m:Movie {isPopular: true})
RETURN m
```

```
Request: {
    Match expression: {
        Left node: Variable pattern match: {
            Variable name: m
            Scheme name: Movie
            Pattern: Attribute List: {
                Attribute name: isPopular
                Value: Bool Literal node: {
                    Value: 1
                }
            }
        }
    }
    Return Expression: {
        Variable value node: {
            Variable name: m
            Field name:
        }
    }
}
```

Существование подстроки

```
MATCH (m:Movie WHERE m.name contains "Love")<-[s:STARRED]-(actors)
RETURN actors.name
```

```
Request: {
    Match expression: {
        Left node: Variable filter match: {
            Variable name: m
            Scheme name: Movie
            Filter: Predicate: {
                expr: Filter bypass: {
                    Wrapped filter: Filter node: {
                        Left part: Variable value node: {
                            Variable name: m
                            Field name: name
                        }
                        Operation: CONTAINS
                        Right part: String Literal node: {
                            Value: "Love"
                        }
                    }
                }
            }
        }
        Relation: Relation match: {
            Variable name: s
            Relation: STARRED
            Direction: REVERSE
        }
        Right node: Any variable match: {
            Variable name: actors
            Scheme name:
        }
    }
    Return Expression: {
        Variable value node: {
            Variable name: actors
            Field name: name
        }
    }
}
```

Логическая комбинация условий и булевских значений

```
MATCH (p:Person WHERE age > 27 AND NOT children < 2)
RETURN p
```

```
Request: {
    Match expression: {
        Left node: Variable filter match: {
            Variable name: p
            Scheme name: Person
            Filter: Predicate: {
                expr: AND: {
                    Left operand: Filter bypass: {
                        Wrapped filter: Filter node: {
                            Left part: Variable value node: {
                                Variable name: age
                                Field name:
                            }
                            Operation: GREATER
                            Right part: Int Literal node: {
                                Value: 27
                            }
                        }
                    }
                    Right operand: NOT: {
                        Operand: Filter bypass: {
                            Wrapped filter: Filter node: {
                                Left part: Variable value node: {
                                    Variable name: children
                                    Field name:
                                }
                                Operation: LESS
                                Right part: Int Literal node: {
                                    Value: 2
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    Return Expression: {
        Variable value node: {
            Variable name: p
            Field name:
        }
    }
}
```

Вставка

```
MATCH (p:Person {name: "Charlie Sheen"})
CREATE (pa:PotentialActor {name: p.name, age: p.age})
```

```
Request: {
    Match expression: {
        Left node: Variable pattern match: {
            Variable name: p
            Scheme name: Person
            Pattern: Attribute List: {
                {
                    Attribute name: name
                    Value: String Literal node: {
                        Value: "Charlie Sheen"
                    }
                }
            }
        }
    }
    Create Expression: {
        Left node: Variable pattern match: {
            Variable name: pa
            Scheme name: PotentialActor
            Pattern: Attribute List: {
                {
                    Attribute name: name
                    Value: Variable value node: {
                        Variable name: p
                        Field name: name
                    }
                },
                {
                    Attribute name: name
                    Value: Variable value node: {
                        Variable name: p
                        Field name: age
                    }
                }
            }
        }
    }
}
```

Удаление

```
MATCH (m:Movie WHERE banned == true)
DELETE m
```

```
Request: {
    Match expression: {
        Left node: Variable filter match: {
            Variable name: m
            Scheme name: Movie
            Filter: Predicate: {
                expr: Filter bypass: {
                    Wrapped filter: Filter node: {
                        Left part: Variable value node: {
                            Variable name: banned
                            Field name:
                        }
                        Operation: EQUAL
                        Right part: Bool Literal node: {
                            Value: 1
                        }
                    }
                }
            }
        }
    }
    Delete Expression: {
        Variable: m
    }
}
```

Вывод

Был реализован модуль разбора подмножества языка запросов Cypher. Я научилась работать с bison и flex, строить синтаксические деревья.