Национальный исследовательский университет
компьютерных технологий, механики и оптики

Факультет ПИиКТ

Низкоуровневое программирование
Лабораторная работа №2
Вариант 4

Работу выполнил: Зубахин Д. С.
Группа: P33312
Преподаватель: Кореньков Ю. Д.

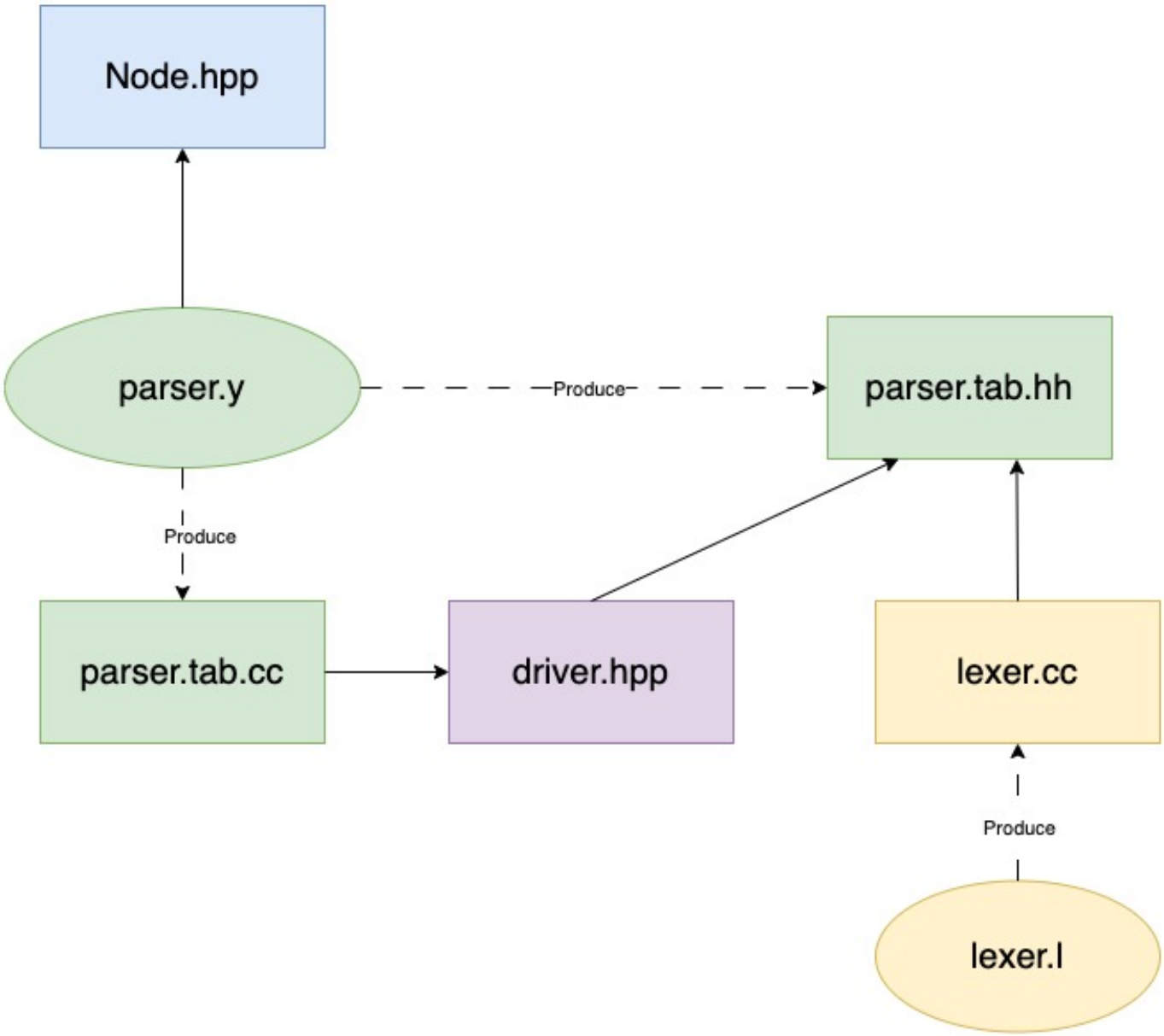
Санкт-Петербург
2022 год

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.

1. Спроектировать архитектуру модуля
2. Провести изучение технологий flex, bison
3. Изучить грамматику языка GraphQL
4. Разработать собственную грамматику
5. Написать тест-кейсы запросов и прогнать через приложение изучив полученный результат

2 Описание работы



Программный комплекс состоит из 4 частей:

1. Лексер - Любезно предоставляет токены интересующие нас
2. Парсер - Составляющий языковые конструкции по заданной грамматике
3. Топология AST - заголовочный файл в котором описаны все структуры в которые должен превратиться текст, пройдя через парсер
4. Драйвер - модуль, который является прослойкой между поступающими токенами и превращением их в грамматические конструкции. Парсер использует драйвер для получения нового токена. Решение принято чтобы ослабить связность и ввести стадию препроцессинга токенов

3 Устройство структур для представления AST

```
enum FieldType
{
    STRING,
    INT32,
    DOUBLE,
    BOOL
};

enum Command
{
    INSERT,
    DELETE,
    SELECT,
    UPDATE
};

enum Cmp
{
    GT = 0,
    GE,
    LT,
    LE,
    IN,
    EQ
};

struct Node
{
protected:
    std::string offset(int level) { return std::string(level, '\t'); }

public:
    virtual std::string repr(int level) { return "Node"; };
};

struct FieldNode : Node
{
    FieldType type;
    std::string name_;
    std::string str_value_;
    int32_t int_value_;
    double double_value_;
    bool bool_value_;
};

struct FieldListNode : public Node
{
    std::vector<FieldNode> fields = {};

    void add(FieldNode &field) { fields.push_back(field); }
};


struct PropertyNode : FieldNode
{
    Cmp cmp_;
};

struct PropertyListNode : Node
{
```

```
    std::vector<PropertyNode> fields = {};

    void add(PropertyNode &field) { fields.push_back(field); }
};

struct WordListNode : Node
{
    std::vector<std::string> fields = {};

    void add(std::string field)
    {
        fields.push_back(field);
    }
};


struct ReprNode : Node
{
    WordListNode wordListNode_;
};

struct ConditionNode : Node
{
    PropertyListNode propertyListNode_;
};

struct EntityNode : Node
{
    std::string nodeName_ = "Root";
    FieldListNode fieldListNode_;
};

struct EntityList : Node
{
    std::vector<EntityNode> entities;

    void add(const EntityNode &node) { entities.push_back(node); }
};

struct ConditionBodyNode : Node
{
    std::string typeName_;
    ConditionNode conditionNode_;
};

struct EntityBodyNode : Node
{
    std::string typeName_;
    EntityList entityList_;
};

struct QueryNode : Node
{
    Command command_;
    ReprNode reprNode_;
};

struct InsertQueryNode : QueryNode
{
    EntityBodyNode entityBodyNode_;
};
```

```
struct SelectQueryNode : QueryNode
{
    ConditionBodyNode conditionBodyNode_;
};

struct DeleteQueryNode : QueryNode
{
    ConditionBodyNode conditionBodyNode_;
};

struct UpdateQueryNode : QueryNode
{
    ConditionBodyNode conditionBodyNode_;
    EntityBodyNode entityBodyNode_;
};
```

Для удобства были убраны все конструкторы и отнаследованные переопределенные методы(в частности repr). Структура следующая - мы имеем структуру Node у которой есть виртуальный метод возвращающий текстовую репрезентацию интересуемой структуры и метод высчитывающий отступ печати. Все остальные структуру наследуют Node и являются типизированными обертками над продукциями.

4 Аспекты реализации

4.1 Реализация лексера

```
%option c++ case-insensitive noyywrap

%{

#include "parser.tab.hh"

%}

WS      [ \t\n\v]+
DIGIT   [0-9]
DIGIT1  [1-9]
WORD    [A-Za-z][A-Za-z0-9_]*

%%

{WS}                                    /* skip blanks and tabs */

GT|GE|LT|LE                             return yy::parser::token_type::CMP;
EQ                                      return yy::parser::token_type::EQ;
IN                                      return yy::parser::token_type::IN;

\"{WORD}\"                              return yy::parser::token_type::STRING;
(-?)(0|{DIGIT1}{DIGIT}*)(\.{DIGIT}+)    return yy::parser::token_type::DOUBLE;
(-?){DIGIT1}{DIGIT}*                    return yy::parser::token_type::INT;
true|false                              return yy::parser::token_type::BOOL;

INSERT                                  return yy::parser::token_type::INSERT;
UPDATE                                  return yy::parser::token_type::UPDATE;
DELETE                                  return yy::parser::token_type::DELETE;
SELECT                                  return yy::parser::token_type::SELECT;

{WORD}                                  return yy::parser::token_type::WORD;

"{"                                     return yy::parser::token_type::LB;
"}"                                     return yy::parser::token_type::RB;
"("                                     return yy::parser::token_type::LP;
")"                                     return yy::parser::token_type::RP;
":"                                     return yy::parser::token_type::COLON;

.                                 return yy::parser::token_type::ERR;

%%
```

Использовался c++ интерфейс. Также я отключил чувствительность к регистру и прописывание вручную yywrap функции.

4.2 Реализация драйвера

```
parser::token_type yylex(parser::semantic_type *yylval)
        {
            parser::token_type tt = static_cast<parser::token_type>(plex_->yylex());

            switch (tt)
            {
            case yy::parser::token_type::WORD:
            case yy::parser::token_type::STRING:
                yylval->as<std::string>() = plex_->YYText();
                break;
            case yy::parser::token_type::INT:
                yylval->as<int>() = std::stoi(plex_->YYText());
                break;
            case yy::parser::token_type::DOUBLE:
                yylval->as<double>() = std::stod(plex_->YYText());
                break;
            case yy::parser::token_type::BOOL:
                yylval->as<bool>() = (strcmp(plex_->YYText(), "true") == 0 || strcmp(plex_->YYText(),
                break;
            case yy::parser::token_type::CMP:
            case yy::parser::token_type::EQ:
            case yy::parser::token_type::IN:
                yylval->as<Cmp>() = cmpTable.find(plex_->YYText())->second;
                break;
            case yy::parser::token_type::INSERT:
            case yy::parser::token_type::DELETE:
            case yy::parser::token_type::SELECT:
            case yy::parser::token_type::UPDATE:
                yylval->as<Command>() = commandTable.find(plex_->YYText())->second;
                break;
            default:
                break;
            }

            return tt;
        }
```

Суть данного модуля заключалась в препроцессинге токенов. Это облегчило модули для парсинга и лексического анализа и уменьшило сложность кода.

Через плюсовый интерфейс лекса мы создавали объект лексера, кидали его в драйвер и использовали как параметр в парсере, все обособлено и красиво.

4.3 Реализация парсера

Использовался c++ интерфейс bison со следующей грамматикой:

```
query_list: insert_query                                { driver->insert($1); }
  | query_list insert_query                             { driver->insert($2); }
  | select_query                                        { driver->insert($1); }
  | query_list select_query                             { driver->insert($2); }
  | delete_query                                        { driver->insert($1); }
  | query_list delete_query                             { driver->insert($2); }
  | update_query                                        { driver->insert($1); }
  | query_list update_query                             { driver->insert($2); }
;

insert_query: INSERT LB entity_body repr RB                 { $$ = InsertQueryNode{$1, $3, $4}; }
;
select_query: SELECT LB condition_body repr RB              { $$ = SelectQueryNode{$1, $3, $4}; }
;
delete_query: DELETE LB condition_body repr RB              { $$ = DeleteQueryNode{$1, $3, $4}; }
```

```
;
update_query: UPDATE LB condition_body entity_body repr RB    { $$ = UpdateQueryNode{$1, $3, $4, $5};
;

entity_body: WORD LP LB entity_list RB RP                     { $$ = EntityBodyNode{$1, $4}; }
;

entity_list: entity                                           { $$.add($1); }
| entity_list entity                                          { $$ = $1; $$.add($2); }
;

entity: WORD COLON LB field_list RB                           { $$ = EntityNode{$1, $4}; }
| field_list                                                  { $$ = EntityNode{$1}; }
;

condition_body: WORD LP condition RP                          { $$ = ConditionBodyNode{$1, $3}; }
;

condition: LB property_list RB                                { $$ = ConditionNode{$2}; }
;

repr: LB word_list RB                                         { $$ = ReprNode{$2};  }
| LB RB                                                       {  }
;

word_list: WORD                                               { $$.add($1); }
| word_list WORD                                              { $$ = $1; $$.add($2); }
;

property_list: property                                       { $$.add($1); }
| property_list property                                      { $$ = $1; $$.add($2); }
;

field_list: field                                             { $$.add($1); }
| field_list field                                            { $$ = $1; $$.add($2); }
;

property: WORD IN STRING                                      { $$ = PropertyNode{$1, $2, $3}; }
| WORD EQ STRING                                              { $$ = PropertyNode{$1, $2, $3}; }
| WORD CMP INT                                                { $$ = PropertyNode{$1, $2, $3}; }
| WORD EQ INT                                                 { $$ = PropertyNode{$1, $2, $3}; }
| WORD CMP DOUBLE                                             { $$ = PropertyNode{$1, $2, $3}; }
| WORD EQ DOUBLE                                              { $$ = PropertyNode{$1, $2, $3}; }
| WORD CMP BOOL                                               { $$ = PropertyNode{$1, $2, $3}; }
| WORD EQ BOOL                                                { $$ = PropertyNode{$1, $2, $3}; }
;

field: WORD COLON STRING                                      { $$ = FieldNode{$1, $3}; }
| WORD COLON INT                                              { $$ = FieldNode{$1, $3}; }
| WORD COLON DOUBLE                                           { $$ = FieldNode{$1, $3}; }
| WORD COLON BOOL                                             { $$ = FieldNode{$1, $3}; }
;
```

5 Результаты

1. INSERT

```
    INSERT {
    K({
        name: "Dmitry"
        height: 150
    }) {
        name
        height
        width
    }
}

INSERT {
    K({
        name: {
            literals: 5
            language: "Russian"
        }
        height: 150
    }) {
        height
        width
    }
}

insert {
    K({
        name: {
            literals: 5
            language: "Russian"
            is_new: true
        }
        height: 150
        width: 50.5
    }) {
    }
}


GQLInsertQuery
GQLCommand 0
GQLEntityBody
        GQLWord K
        GQLEntityList
                GQLEntity
                        GQLEntityName Root
                        GQLFieldList
                                GQLField name: name   value: "Dmitry"
                                GQLField name: height value: 150
GQLRepresentation
        GQLWordList
                GQLWord name
                GQLWord height
                GQLWord width
GQLInsertQuery
GQLCommand 0
GQLEntityBody
        GQLWord K
        GQLEntityList
```

```
                GQLEntity
                        GQLEntityName name
                        GQLFieldList
                                GQLField name: literals       value: 5
                                GQLField name: language       value: "Russian"
                GQLEntity
                        GQLEntityName Root
                        GQLFieldList
                                GQLField name: height value: 150
GQLRepresentation
        GQLWordList
                GQLWord height
                GQLWord width
GQLInsertQuery
GQLCommand 0
GQLEntityBody
        GQLWord K
        GQLEntityList
                GQLEntity
                        GQLEntityName name
                        GQLFieldList
                                GQLField name: literals       value: 5
                                GQLField name: language       value: "Russian"
                                GQLField name: is_new value: 1
                GQLEntity
                        GQLEntityName Root
                        GQLFieldList
                                GQLField name: height value: 150
                                GQLField name: width  value: 50.500000
GQLRepresentation
        GQLWordList
```

2. SELECT

```
    SELECT {
    K({
        name in "Dmitry"
        height gt 150
        width eq 555
    }) {
        name
        height
        width
    }
}

select {
    K({
        name eq "Dmitry"
        height le 150
    }) {
        name
    }
}


GQLSelectQuery
GQLCommand 2
GQLConditionBody
        GQLWord K
        GQLCondition
                GQLPropertyList
                        GQLProperty name: name        Operation: 4    value: "Dmitry"
                        GQLProperty name: height      Operation: 0    value: 150
                        GQLProperty name: width       Operation: 5    value: 555
GQLRepresentation
        GQLWordList
                GQLWord name
                GQLWord height
                GQLWord width
GQLSelectQuery
GQLCommand 2
GQLConditionBody
        GQLWord K
        GQLCondition
                GQLPropertyList
                        GQLProperty name: name        Operation: 5    value: "Dmitry"
                        GQLProperty name: height      Operation: 3    value: 150
GQLRepresentation
        GQLWordList
                GQLWord name
```

3. DELETE

```
    delete {
    K({
        count le 50
    }) {
        name
        height
    }
}
```

```
GQLDeleteQuery
GQLCommand 1
GQLConditionBody
        GQLWord K
        GQLCondition
                GQLPropertyList
                        GQLProperty name: count       Operation: 3    value: 50
GQLRepresentation
        GQLWordList
                GQLWord name
                GQLWord height
```

4. UPDATE

```
    update {
    K({
        name eq "Dmitry"
    })
    L({
        name: "Andrey"
    })
    {
        name
        height
    }
}

update {
    K({
        name in "Dmitry"
        height lt 150
    })
    {
        name: "Andrey"
    }
    {
        name
        height
    }
}


GQLUpdateQuery
GQLCommand 3
GQLConditionBody
        GQLWord K
        GQLCondition
                GQLPropertyList
                        GQLProperty name: name        Operation: 5    value: "Dmitry"
GQLEntityBody
        GQLWord L
        GQLEntityList
                GQLEntity
                        GQLEntityName Root
                        GQLFieldList
                                GQLField name: name   value: "Andrey"
GQLRepresentation
        GQLWordList
                GQLWord name
                GQLWord height
GQLUpdateQuery
GQLCommand 3
GQLConditionBody
        GQLWord K
        GQLCondition
                GQLPropertyList
                        GQLProperty name: name        Operation: 4    value: "Dmitry"
                        GQLProperty name: height      Operation: 2    value: 150
GQLEntityBody
        GQLWord L
        GQLEntityList
                GQLEntity
                        GQLEntityName Root
                        GQLFieldList
```

```
                                GQLField name: name   value: "Andrey"
GQLRepresentation
        GQLWordList
                GQLWord name
                GQLWord height
```