Университет ИТМО
Факультет программной инженерии и компьютерной техники

Лабораторная работа №2

«Низкоуровневое программирование»

Вариант 2

Выполнил:
Студент группы P33102
Гулямов Т.И.

Преподаватель:
Кореньков Ю.Д.

Санкт-Петербург
2023

Цель:

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
    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
    b. Язык запросов должен поддерживать возможность описания следующих конструкций: порождение нового элемента данных, выборка, обновление и удаление существующих элементов данных по условию
        i. Условия
            1. На равенство и неравенство для чисел, строк и булевских значений
            2. На строгие и нестрогие сравнения для чисел
            3. Существование подстроки
        ii. Логическую комбинацию произвольного количества условий и булевских значений
        iii. В качестве любого аргумента условий могут выступать литеральные значения (константы) или ссылки на значения, ассоциированные с элементами данных (поля, атрибуты, свойства)
        iv. Разрешение отношений между элементами модели данных любых условий над сопрягаемыми элементами данных
        v. Поддержка арифметических операций и конкатенации строк не обязательна
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

Описание работы:

Сборка и запуск:

```
git clone git@github.com:tplaymeow/itmo-low-level-programming-lab2.git
git submodule init
git submodule update
```

Для сборки и запуска необходимо использовать Cmake.

Тестовая программа

```
cmake -B build -DCMAKE_BUILD_TYPE=Release
cmake --build build
build/itmo_low_level_programming_lab2
```

Модуль:

- sql_ast.h, sql_ast.c – структуры данных и вспомогательные функции для представления абстрактного синтаксического дерева запросов нашего SQL-подобного языка
- serialization.h, serialization.c – функции для вывода абстрактного синтаксического дерева в JSON формате
- sql_lexer.l – правила для лексического анализа запросов. На основание этого файла с помощью пакета FLEX генерируется код лексера. В результате работы лексера исходный текст разбивается на последовательность токенов.
- sql_parser.y – правила для синтаксического анализа запросов. На основание этого файла с помощью пакета BISON генерируется код парсера. В результате работы парсера из последовательности токенов формируется абстрактное синтаксическое дерево.

Структуры данных:

Основные структуры данных представляющие разобранный запрос:

```
struct sql_ast_create_statement {
  char *table_name;
  struct sql_ast_column_with_type_list *columns;
};

struct sql_ast_drop_statement {
  char *table_name;
};

struct sql_ast_insert_statement {
  char *table_name;
  struct sql_ast_literal_list *values;
};

struct sql_ast_select_statement {
  char *table_name;
  struct sql_ast_join_optional join;
  struct sql_ast_filter filter;
};

struct sql_ast_delete_statement {
  char *table_name;
  struct sql_ast_filter filter;
};

struct sql_ast_update_statement {
  char *table_name;
  struct sql_ast_filter filter;
  struct sql_ast_column_with_literal_list *set;
};

enum sql_ast_statement_type {
  SQL_AST_STATEMENT_TYPE_CREATE,
  SQL_AST_STATEMENT_TYPE_DROP,
  SQL_AST_STATEMENT_TYPE_INSERT,
  SQL_AST_STATEMENT_TYPE_SELECT,
  SQL_AST_STATEMENT_TYPE_DELETE,
  SQL_AST_STATEMENT_TYPE_UPDATE
};

union sql_ast_statement_value {
  struct sql_ast_create_statement create;
  struct sql_ast_drop_statement drop;
  struct sql_ast_insert_statement insert;
  struct sql_ast_select_statement select;
  struct sql_ast_delete_statement delete;
  struct sql_ast_update_statement update;
};
```

```
struct sql_ast_statement {
  enum sql_ast_statement_type type;
  union sql_ast_statement_value value;
};
```

Вспомогательные структуры данных:

```
enum sql_ast_data_type {
  SQL_AST_DATA_TYPE_INTEGER,
  SQL_AST_DATA_TYPE_FLOATING_POINT,
  SQL_AST_DATA_TYPE_BOOLEAN,
  SQL_AST_DATA_TYPE_TEXT
};

struct sql_ast_column_with_type {
  char *name;
  enum sql_ast_data_type type;
};

struct sql_ast_column_with_type_list {
  struct sql_ast_column_with_type item;
  struct sql_ast_column_with_type_list *next;
};

union sql_ast_literal_value {
  int32_t integer;
  double floating_point;
  bool boolean;
  char *text;
};

struct sql_ast_literal {
  enum sql_ast_data_type type;
  union sql_ast_literal_value value;
};

struct sql_ast_literal_list {
  struct sql_ast_literal item;
  struct sql_ast_literal_list *next;
};

enum sql_ast_operand_type {
  SQL_AST_OPERAND_TYPE_COLUMN,
  SQL_AST_OPERAND_TYPE_LITERAL
};

union sql_ast_operand_value {
  char *column;
  struct sql_ast_literal literal;
};
```

```
struct sql_ast_operand {
  enum sql_ast_operand_type type;
  union sql_ast_operand_value value;
};

union sql_ast_text_operand_value {
  char *column;
  char *literal;
};

struct sql_ast_text_operand {
  enum sql_ast_operand_type type;
  union sql_ast_text_operand_value value;
};

enum sql_ast_comparison_operator {
  SQL_AST_COMPARISON_OPERATOR_EQUAL,
  SQL_AST_COMPARISON_OPERATOR_NOT_EQUAL,
  SQL_AST_COMPARISON_OPERATOR_GREATER,
  SQL_AST_COMPARISON_OPERATOR_GREATER_OR_EQUAL,
  SQL_AST_COMPARISON_OPERATOR_LESS,
  SQL_AST_COMPARISON_OPERATOR_LESS_OR_EQUAL
};

struct sql_ast_comparison {
  enum sql_ast_comparison_operator operator;
  struct sql_ast_operand left;
  struct sql_ast_operand right;
};

struct sql_ast_contains {
  struct sql_ast_text_operand left;
  struct sql_ast_text_operand right;
};

enum sql_ast_logic_binary_operator {
  SQL_AST_LOGIC_BINARY_OPERATOR_AND,
  SQL_AST_LOGIC_BINARY_OPERATOR_OR
};

struct sql_ast_logic {
  enum sql_ast_logic_binary_operator operator;
  struct sql_ast_filter *left;
  struct sql_ast_filter *right;
};

enum sql_ast_filter_type {
  SQL_AST_FILTER_TYPE_ALL,
  SQL_AST_FILTER_TYPE_COMPARISON,
  SQL_AST_FILTER_TYPE_CONTAINS,
  SQL_AST_FILTER_TYPE_LOGIC
```

```
};

union sql_ast_filter_value {
  struct sql_ast_comparison comparison;
  struct sql_ast_contains contains;
  struct sql_ast_logic logic;
};

struct sql_ast_filter {
  enum sql_ast_filter_type type;
  union sql_ast_filter_value value;
};

struct sql_ast_join {
  char *join_table;
  char *table_column;
  char *join_table_column;
};

struct sql_ast_join_optional {
  bool has_value;
  struct sql_ast_join value;
};

struct sql_ast_column_with_literal {
  char *name;
  struct sql_ast_literal literal;
};

struct sql_ast_column_with_literal_list {
  struct sql_ast_column_with_literal item;
  struct sql_ast_column_with_literal_list *next;
};
```

Аспекты реализации:

Лексический анализ ключевых слов:

```
"create" {return CREATE;}
"drop" {return DROP;}
"select" {return SELECT;}
"insert" {return INSERT;}
"delete" {return DELETE;}
"update" {return UPDATE;}
"table" {return TABLE;}
"from" {return FROM;}
"where" {return WHERE;}
"into" {return INTO;}
"set" {return SET;}
"integer" {return INTEGER_TYPE;}
"float" {return FLOATING_TYPE;}
```

```
"bool" {return BOOLEAN_TYPE;}
"text" {return TEXT_TYPE;}
"and" {return AND;}
"or" {return OR;}
"contains" {return CONTAINS;}
"join" {return JOIN;}
"on" {return ON;}
"=" {return ASSIGN;}
"(" {return LEFT_BRACKET;}
")" {return RIGHT_BRACKET;}
";" {return SEMICOLON;}
"," {return COMMA;}
"==" {return COMPARISON_OPERATOR_EQUAL;}
"!=" {yylval.comparison_operator_val = SQL_AST_COMPARISON_OPERATOR_NOT_EQUAL;
return COMPARISON_OPERATOR;}
">" {yylval.comparison_operator_val = SQL_AST_COMPARISON_OPERATOR_GREATER;
return COMPARISON_OPERATOR;}
">=" {yylval.comparison_operator_val =
SQL_AST_COMPARISON_OPERATOR_GREATER_OR_EQUAL; return COMPARISON_OPERATOR;}
"<" {yylval.comparison_operator_val = SQL_AST_COMPARISON_OPERATOR_LESS; return
COMPARISON_OPERATOR;}
"<=" {yylval.comparison_operator_val =
SQL_AST_COMPARISON_OPERATOR_LESS_OR_EQUAL; return COMPARISON_OPERATOR;}
"true" {yylval.boolean_val = true; return BOOLEAN_VAL;}
"false" {yylval.boolean_val = false; return BOOLEAN_VAL;}
```

Лексический анализ целочисленных литералов:

```
[0-9]+ {yylval.integer_val = atoi(yytext); return INTEGER_VAL;}
```

Лексический анализ литералов чисел с плавающей точкой:

```
[0-9]+\.[0-9]+     {yylval.floating_val = atof(yytext); return FLOATING_VAL;}
```

Лексический анализ строковых литералов:

```
\"([^\\\"]|\\.)*\"  {
    size_t size = strlen(yytext);
    yylval.text_val = malloc(size - 1);
    strncpy(yylval.text_val, yytext + 1, size - 2);
    yylval.text_val[size - 2] = '\0';
    return TEXT_VAL;
}
```

Лексический анализ идентификаторов:

```
[a-zA-Z_][a-zA-Z0-9_]* {yylval.identifier_val = strdup(yytext); return
IDENTIFIER;}
```

Парсинг выражений:

```
input
    : statement SEMICOLON {
        struct serialization_context *ctx = serialization_context_create();
        serialize(ctx, $1);
        serialization_context_destroy(ctx);
        sql_ast_statement_free($1);
    }
    ;

statement
    : drop_statement {
        $$ = (struct sql_ast_statement) {
            .type = SQL_AST_STATEMENT_TYPE_DROP,
            .value.drop = $1
        };
    }
    | create_statement {
        $$ = (struct sql_ast_statement) {
            .type = SQL_AST_STATEMENT_TYPE_CREATE,
            .value.create = $1
        };
    }
    | insert_statement {
        $$ = (struct sql_ast_statement) {
            .type = SQL_AST_STATEMENT_TYPE_INSERT,
            .value.insert = $1
        };
    }
    | select_statement {
        $$ = (struct sql_ast_statement) {
            .type = SQL_AST_STATEMENT_TYPE_SELECT,
            .value.select = $1
        };
    }
    | update_statement {
        $$ = (struct sql_ast_statement) {
            .type = SQL_AST_STATEMENT_TYPE_UPDATE,
            .value.update = $1
        };
    }
    | delete_statement {
        $$ = (struct sql_ast_statement) {
            .type = SQL_AST_STATEMENT_TYPE_DELETE,
            .value.delete = $1
        };
    }
    ;

create_statement
    : CREATE TABLE IDENTIFIER column_with_type_list {
        $$ = (struct sql_ast_create_statement) {
```

```
            .table_name = $3,
            .columns = $4
        };
    }
    ;

drop_statement
    : DROP TABLE IDENTIFIER {
        $$ = (struct sql_ast_drop_statement) {
            .table_name = $3
        };
    }
    ;

insert_statement
    : INSERT INTO IDENTIFIER literal_list {
        $$ = (struct sql_ast_insert_statement) {
            .table_name = $3,
            .values = $4
        };
    }
    ;

select_statement
    : SELECT FROM IDENTIFIER join where {
        $$ = (struct sql_ast_select_statement) {
            .table_name = $3,
            .join = $4,
            .filter = $5
        };
    }
    ;

update_statement
    : UPDATE IDENTIFIER SET column_with_literal_list where {
        $$ = (struct sql_ast_update_statement) {
            .table_name = $2,
            .filter = $5,
            .set = $4
        };
    }

delete_statement
    : DELETE FROM IDENTIFIER where {
        $$ = (struct sql_ast_delete_statement) {
            .table_name = $3,
            .filter = $4
        };
    }
    ;
```

Результаты:

Примеры:

```
drop table Users;

{
  "drop": {
    "table_name": "Users"
  }
}

total heap usage: 7 allocs, 4 frees, 24,672 bytes allocated
```

```
CREATE TABLE Users
(
    name text,
    age integer,
    rating float,
    is_admin bool
);

{
  "create": {
    "table_name": "Users",
    "columns": [
      {
        "name": "is_admin",
        "type": "bool"
      },
      {
        "name": "rating",
        "type": "float"
      },
      {
        "name": "age",
        "type": "integer"
      },
      {
        "name": "name",
        "type": "text"
      }
    ]
  }
}
```

```
total heap usage: 15 allocs, 12 frees, 24,793 bytes allocated
```

```
update Users
set rating = 5.0
where (age > 18 and age <= 50) or is_admin == true;

{
  "update": {
    "table_name": "Users",
    "filter": {
      "logic": {
        "operator": "OR",
        "left": {
          "logic": {
            "operator": "AND",
            "left": {
              "comparison": {
                "operator": "GREATER",
                "left": {
                  "column": {
                    "column_name": "age"
                  }
                },
                "right": {
                  "literal": {
                    "type": "integer",
                    "value": 18
                  }
                }
              }
            },
            "right": {
              "comparison": {
                "operator": "LESS_OR_EQUAL",
                "left": {
                  "column": {
                    "column_name": "age"
                  }
                },
                "right": {
                  "literal": {
                    "type": "integer",
                    "value": 50
                  }
                }
```

```
              }
            }
          }
        },
        "right": {
          "comparison": {
            "operator": "EQUAL",
            "left": {
              "column": {
                "column_name": "is_admin"
              }
            },
            "right": {
              "literal": {
                "type": "bool",
                "value": true
              }
            }
          }
        }
      }
    },
    "values": [
      {
        "name": "rating",
        "literal": {
          "type": "float",
          "value": 5.000000
        }
      }
    ]
  }
}

total heap usage: 16 allocs, 13 frees, 24,984 bytes allocated
```

```
delete from Users
where name == "Timur";

{
  "delete": {
    "table_name": "Users",
    "filter": {
      "comparison": {
        "operator": "EQUAL",
```

```
        "left": {
          "column": {
            "column_name": "name"
          }
        },
        "right": {
          "literal": {
            "type": "text",
            "value": "Timur"
          }
        }
      }
    }
  }
}

total heap usage: 9 allocs, 6 frees, 24,683 bytes allocated
```

```
insert into Users
("Timur", false, false, false, 1, 2, 3, 4.0);

{
  "insert": {
    "table_name": "Users",
    "values": [
      {
        "type": "float",
        "value": 4.000000
      },
      {
        "type": "integer",
        "value": 3
      },
      {
        "type": "integer",
        "value": 2
      },
      {
        "type": "integer",
        "value": 1
      },
      {
        "type": "bool",
        "value": false
      },
```

```
      {
        "type": "bool",
        "value": false
      },
      {
        "type": "bool",
        "value": false
      },
      {
        "type": "text",
        "value": "Timur"
      }
    ]
  }
}

total heap usage: 16 allocs, 13 frees, 24,870 bytes allocated
```

```
select from Users
where (age > 18 and age <= 50) or (is_admin == true or name == "Admin"
and age != 10);

{
  "select": {
    "table_name": "Users",
    "filter": {
      "logic": {
        "operator": "OR",
        "left": {
          "logic": {
            "operator": "AND",
            "left": {
              "comparison": {
                "operator": "GREATER",
                "left": {
                  "column": {
                    "column_name": "age"
                  }
                },
                "right": {
                  "literal": {
                    "type": "integer",
                    "value": 18
                  }
                }
```

```
          }
        },
        "right": {
          "comparison": {
            "operator": "LESS_OR_EQUAL",
            "left": {
              "column": {
                "column_name": "age"
              }
            },
            "right": {
              "literal": {
                "type": "integer",
                "value": 50
              }
            }
          }
        }
      }
    },
    "right": {
      "logic": {
        "operator": "OR",
        "left": {
          "comparison": {
            "operator": "EQUAL",
            "left": {
              "column": {
                "column_name": "is_admin"
              }
            },
            "right": {
              "literal": {
                "type": "bool",
                "value": true
              }
            }
          }
        },
        "right": {
          "logic": {
            "operator": "AND",
            "left": {
              "comparison": {
                "operator": "EQUAL",
                "left": {
```

```
                    "column": {
                      "column_name": "name"
                    }
                  },
                  "right": {
                    "literal": {
                      "type": "text",
                      "value": "Admin"
                    }
                  }
                }
              },
              "right": {
                "comparison": {
                  "operator": "NOT_EQUAL",
                  "left": {
                    "column": {
                      "column_name": "age"
                    }
                  },
                  "right": {
                    "literal": {
                      "type": "integer",
                      "value": 10
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}

total heap usage: 21 allocs, 18 frees, 25,216 bytes allocated
```

Тестирование:
Для тестирования приложения в разных конфигурациях используется GitHub Actions. [Результаты](#).

Создан workflow, который проверяет основную функциональность приложения в следующих конфигурациях:

- Ubuntu, GCC, Release

- Ubuntu, Clang, Release
- MacOS, GCC, Release
- MacOS, Clang, Release
- Ubuntu, GCC, Debug + Sinitizers
- Ubuntu, Clang, Debug + Sinitizers
- MacOS, GCC, Debug + Sinitizers
- MacOS, Clang, Debug + Sinitizers

Результаты:

Во время выполнения лабораторной работы, познакомился с популярными инструментами для лексического анализа (Flex) и синтаксического анализа (Bison), попрактиковался в составление грамматик и моделирование AST, углубил свои знания в CMake. Получил бесценный опыт, который, несомненно, пригодится мне в будущем!