

Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №2

По дисциплине "Низкоуровневое программирование"

Выполнил:

Русинов Дмитрий Станиславович

Преподаватель:

Кореньков Юрий Дмитриевич

Санкт-Петербург, 2023

Задание

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.

Вариант - LINQ

Результат выполнения

Лабораторная работа была выполнена на языке C с использованием средств bison и flex.

Структура для хранения дерева разбора

```
struct AstNode {
    enum AstNodeType type;
    union {
        char *string_value;
        int int_value;
        float float_value;
        bool bool_value;
    } value;
    size_t children_count;
    struct AstNode **children;
};
```

```
#define FOREACH_AST_NODE_TYPE(AST_NODE_TYPE) \
    AST_NODE_TYPE (ANT_QUERY) \
    AST_NODE_TYPE (ANT_SELECT) \
    AST_NODE_TYPE (ANT_FROM) \
    AST_NODE_TYPE (ANT_FROM_VARNAME) \
    AST_NODE_TYPE (ANT_FROM_COLLECTION_NAME) \
    AST_NODE_TYPE (ANT_WHERE) \
    AST_NODE_TYPE (ANT_AND) \
    AST_NODE_TYPE (ANT_OR) \
    AST_NODE_TYPE (ANT_NOT) \
    AST_NODE_TYPE (ANT_EQ_OP) \
    AST_NODE_TYPE (ANT_NE) \
    AST_NODE_TYPE (ANT_LT) \
```

```

AST_NODE_TYPE (ANT_GT) \
AST_NODE_TYPE (ANT_LE) \
AST_NODE_TYPE (ANT_GE) \
AST_NODE_TYPE (ANT_PLUS) \
AST_NODE_TYPE (ANT_MINUS) \
AST_NODE_TYPE (ANT_TIMES) \
AST_NODE_TYPE (ANT_DIVIDE) \
AST_NODE_TYPE (ANT_IDENTIFIER) \
AST_NODE_TYPE (ANT_STRING) \
AST_NODE_TYPE (ANT_INTEGER) \
AST_NODE_TYPE (ANT_DOUBLE) \
AST_NODE_TYPE (ANT_BOOLEAN) \
AST_NODE_TYPE (ANT_JOIN) \
AST_NODE_TYPE (ANT_JOIN_IN) \
AST_NODE_TYPE (ANT_JOIN_ON) \
AST_NODE_TYPE (ANT_SELECT_QUERY_BODY) \
AST_NODE_TYPE (ANT_QUERY_BODY_CLAUSES) \
AST_NODE_TYPE (ANT_QUERY_BODY) \
AST_NODE_TYPE (ANT_SELECT_QUERY) \
AST_NODE_TYPE (ANT_FIELD_IDENTIFIER) \
AST_NODE_TYPE (ANT_CONTAINS) \
AST_NODE_TYPE (ANT_INSERT_QUERY) \
AST_NODE_TYPE (ANT_INSERT_INTO) \
AST_NODE_TYPE (ANT_INSERT_VALUES) \
AST_NODE_TYPE (ANT_UPDATE_QUERY) \
AST_NODE_TYPE (ANT_UPDATE_FIELD) \
AST_NODE_TYPE (ANT_UPDATE_SET) \
AST_NODE_TYPE (ANT_DELETE_QUERY) \
AST_NODE_TYPE (ANT_DELETE_FROM) \
AST_NODE_TYPE (ANT_DELETE_WHERE) \

#define GENERATE_ENUM(ENUM) ENUM,
#define GENERATE_STRING(String) #String,

enum AstNodeType {FOREACH_AST_NODE_TYPE (GENERATE_ENUM) };
static const char *AstNodeTypeString[] =
{FOREACH_AST_NODE_TYPE (GENERATE_STRING) };

```

Для иерархического представления запроса, т.е. в виде дерева разбора была использована структура, которая хранит в себе некоторый тип текущей ноды и ссылки на детей. Также в ноде хранится значение в тех случаях, когда она описывает литерал или идентификатор какой-либо переменной (поля).

Дополнительная обработка

Необходимо было создать файл `lexer.l`, в котором описаны правила создания токенов

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

identifier [a-zA-Z]+
double [-+]?[0-9]*\.[0-9]+([eE][-+]?[0-9]+)?
integer [-+]?[0-9]+
boolean true|false
quoted_string \"[^\"]*\"

%%

"from" { return TOKEN_FROM; }
"in" { return TOKEN_IN; }
"=" { return TOKEN_EQ; }
"==" { return TOKEN_EQ_OP; }
"where" { return TOKEN_WHERE; }
"join" { return TOKEN_JOIN; }
"on" { return TOKEN_ON; }
"equals" { return TOKEN_EQUALS; }
"into" { return TOKEN_INTTO; }
"select" { return TOKEN_SELECT; }
"Contains" { return TOKEN_CONTAINS; }
"insert" { return TOKEN_INSERT; }
"values" { return TOKEN_VALUES; }
"update" { return TOKEN_UPDATE; }
"set" { return TOKEN_SET; }
"delete" { return TOKEN_DELETE; }
```

```
"(" { return TOKEN_PAR_OPEN; }
")" { return TOKEN_PAR_CLOSE; }
"," { return TOKEN_COMMA; }

"|" { return TOKEN_OR; }
"&&" { return TOKEN_AND; }
"=" { return TOKEN_EQ; }
"!=" { return TOKEN_NE; }
"<" { return TOKEN_LT; }
">" { return TOKEN_GT; }
"<=" { return TOKEN_LE; }
">=" { return TOKEN_GE; }
"+" { return TOKEN_PLUS; }
"-" { return TOKEN_MINUS; }
"*" { return TOKEN_TIMES; }
"/" { return TOKEN_DIVIDE; }
"!" { return TOKEN_NOT; }
"." { return TOKEN_DOT; }

{boolean} {
    yylval.bval = (strcmp(yytext, "true") == 0);
    return TOKEN_BOOLEAN;
}

{identifier} {
    yylval.sval = strdup(yytext);
    return TOKEN_IDENTIFIER;
}

{integer} {
    yylval.ival = atoi(yytext);
    return TOKEN_INTEGER;
}

{double} {
    yylval.dval = atof(yytext);
    return TOKEN_DOUBLE;
}

{quoted_string} {
    yylval.sval = strdup(yytext);
    return TOKEN_QUOTED_STRING;
}
```

```
[ \t\n] ;

";" { return END_OF_STATEMENT; }

. { printf("Unrecognized character: %s\n", yytext); }
```

Причем часть из этих токенов не хранит в себе значение, а для части - его нужно сохранить.

Также был создан файл parser.y, в котором описывается непосредственная грамматика запросов и логика их обработки. В логику их обработки я и включил создание дерева разбора. По сути, после обработки каждого "внутреннего" правила, нода передается наверх, объединяется с другими нодами с общим родителем и в корне сохраняется в переменную.

Также стоит заметить, что LINQ - это query language, поддерживающий только запросы выборки. Поэтому я поддержал запросы на update, delete, insert в sql-like стиле.

Пример обработки правила:

```
join_clause
: TOKEN_JOIN field_or_simple_identifier TOKEN_IN expression TOKEN_ON
expression TOKEN_EQUALS expression {
    struct AstNode *join_in = create_ast_node(ANT_JOIN_IN, 2, $2, $4);
    struct AstNode *join_on = create_ast_node(ANT_JOIN_ON, 2, $6, $8);
    $$ = create_ast_node(ANT_JOIN, 2, join_in, join_on);
}
;
```

Примеры работы программы

Примеры можно найти в файле example.txt

```

ruskaof@asus-zenbook:~/CLionProjects/llp_bison_parser$ ./cmake-build/llp_bison_parser
from varname in collection join anothervar in anothercoll on varname.x equals anothervar.y where varname.f > 3 && varname.a.Contains("aaa") select anothervar.hhh;
parse complete
ANT_SELECT_QUERY
  ANT_FROM
    ANT_FROM_VARNAME
      varname
    ANT_FROM_COLLECTION_NAME
      collection
  ANT_QUERY_BODY
    ANT_QUERY_BODY_CLAUSES
      ANT_JOIN
        ANT_JOIN_IN
          anothervar
          anothercoll
        ANT_JOIN_ON
          ANT_FIELD_IDENTIFIER
            varname
          x
          ANT_FIELD_IDENTIFIER
            anothervar
            y
      ANT_WHERE
        ANT_AND
          ANT_GT
            ANT_FIELD_IDENTIFIER
              varname
              f
            3
          ANT_CONTAINS
            ANT_FIELD_IDENTIFIER
              varname
              a
            "aaa"
  ANT_SELECT
    ANT_FIELD_IDENTIFIER
      anothervar
      hhh

```

```

ruskaof@asus-zenbook:~/CLionProjects/llp_bison_parser$ ./cmake-build/llp_bison_parser
from person in persons where true || (false && true) select person;
parse complete
ANT_SELECT_QUERY
  ANT_FROM
    ANT_FROM_VARNAME
      person
    ANT_FROM_COLLECTION_NAME
      persons
  ANT_QUERY_BODY
    ANT_WHERE
      ANT_OR
        true
      ANT_AND
        false
        true
  ANT_SELECT
    person

```

```

ruskaof@asus-zenbook:~/CLionProjects/llp_bison_parser$ ./cmake-build/llp_bison_parser
insert into student values (1, "string", true, 3.14);
parse complete
ANT_INSERT_QUERY
  ANT_INSERT INTO
    student
  ANT_INSERT_VALUES
    ANT_INSERT_VALUES
      ANT_INSERT_VALUES
        ANT_INSERT_VALUES
          1
          "string"
        true
      3.140000

```

```

ruskaof@asus-zenbook:~/CLionProjects/llp_bison_parser$ ./cmake-build/llp_bison_parser
update pets set pet.name = "Tom";
parse complete
ANT_UPDATE_QUERY
  ANT_UPDATE_FIELD
    pets
  ANT_UPDATE_SET
    ANT_FIELD_IDENTIFIER
      pet
      name
    "Tom"

```

```
ruskaof@asus-zenbook:~/CLionProjects/llp_bison_parser$ ./cmake-build/llp_bison_parser
delete from students where student.age >= 30;
parse complete
ANT_DELETE_QUERY
  ANT_DELETE_FROM
    students
  ANT_DELETE_WHERE
    ANT_GE
      ANT_FIELD_IDENTIFIER
        student
        age
      30
```

Выводы

После выполнения лабораторной работы я могу сказать, что такие средства, как bison и flex - действительно мощные, так как они позволяют обрабатывать некоторый текст в любой грамматике по сути любым образом. Это делает их очень гибкими в использовании.