Университет ИТМО

Факультет программной инженерии и компьютерной техники
Направление подготовки 09.03.01 Информатика и вычислительная техника

Лабораторная работа №2

по дисциплине «Низкоуровневое программирование»
Вариант №8 (LINQ)

Выполнил:
Студент группы P33302
Иванов Н.Д.

Преподаватель:
Кореньков Юрий Дмитриевич

г. Санкт-Петербург
2023

Содержание

Цели..... 3
Задачи..... 4
Описание работы.....5
Примеры обработки запросов:.....10
Выводы:..... 13

Цели

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

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)
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

Структуры для хранения информации о дереве:

```
typedef struct AstNode{
  enum AstNodeType type;
  int num_children;
  struct AstNode** children;
  union {
     double double_val;
     int int_val;
     char* str_val;
     int bool_val;
  } value;
} AstNode;
```

Для создания и выведения информации о дереве были реализованы следующие функции:

```
AstNode *createAstNode(enum AstNodeType type, int num_children, ...) {
  AstNode *node = ( AstNode *) malloc(sizeof( AstNode));
  if (node == NULL) {
     fprintf(stderr, "Failed to allocate memory for AstNode\n");
     return NULL;
  }

  node->type = type;
  node->num_children = num_children;

  node->children = ( AstNode **) malloc(num_children * sizeof( AstNode *));
  if (node->children == NULL) {
     fprintf(stderr, "Failed to allocate memory for AstNode children\n");
     return NULL;
  }

  va_list args;
  va_start(args, num_children);

  for (int i = 0; i < num_children; ++i) {
     node->children[i] = va_arg(args,
                                AstNode*);
  }

  switch (type) {
     case DOUBLE_LITERAL:
        node->value.double_val = va_arg(args, double);
        break;
     case INTEGER_LITERAL:
        node->value.int_val = va_arg(args, int);
        break;
     case STRING_LITERAL:
        node->value.str_val = va_arg(args, char*);
        break;
     case BOOLEAN_LITERAL:
        node->value.bool_val = va_arg(args, int);
        break;
     case FIELD_IDENTIFIER:
        node->value.str_val = va_arg(args, char *);
        break;
     default:
        break;
  }

  va_end(args);

  return node;
}
```

```
static void printAstTreeRecursive(AstNode *node, int depth) {

  if (node == NULL) {
    return;
  }

  for (int i = 0; i < depth; ++i) {
    printf("  ");
  }

  switch (node->type) {
    case DOUBLE_LITERAL:
      printf(": %f", node->value.double_val);
      break;
    case INTEGER_LITERAL:
      printf(": %d", node->value.int_val);
      break;
    case STRING_LITERAL:
      printf(": \"%s\"", removeFirstAndLastChar(node->value.str_val));
      break;
    case BOOLEAN_LITERAL:
      printf(": %s", node->value.bool_val ? "true" : "false");
    case FIELD_IDENTIFIER:
      printf (": %s", node->value.str_val);
      break;
    default:
      printf(": %s", getAstNodeTypeName(node->type));
      break;
  }

  printf("\n");

  for (int i = 0; i < node->num_children; ++i) {
    printAstTreeRecursive(node->children[i], depth + 1);
  }
}
```

Необходимо было создать lexer.l

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

%%

"select"                    { return TOKEN_SELECT; } /* operations */
"insert"                    { return TOKEN_INSERT; }
"delete"                    { return TOKEN_DELETE; }
"update"                    { return TOKEN_UPDATE; }

"from"                      { return TOKEN_FROM; } /* keywords */
"in"                        { return TOKEN_IN; }
"equals"                    { return TOKEN_EQUALS; }
"="                         { return TOKEN_EQ; }
"on"                        { return TOKEN_ON; }
"where"                     { return TOKEN_WHERE; }
"set"                       { return TOKEN_SET; }
"into"                      { return TOKEN_INTO; }
"Contains"                  { return TOKEN_CONTAINS; }
"join"                      { return TOKEN_JOIN; }
"values"                    { return TOKEN_VALUES; }


"("                         { return TOKEN_PAR_OPEN; } /* special symbols */
")"                         { return TOKEN_PAR_CLOSE; }
","                         { return TOKEN_COMMA; }
"."                         { return TOKEN_DOT; }

"=="                        { return TOKEN_EQ_OP; } /* comparators */
"<"                         { return TOKEN_LT; }
">"                         { return TOKEN_GT; }
">="                        { return TOKEN_GE; }
"<="                        { return TOKEN_LE; }
"!="                        { return TOKEN_NE; }
"!"                         { return TOKEN_NOT; }

"||"                        { return TOKEN_OR; } /* combinations */
"&&"                        { return TOKEN_AND; }


"true"                      {yylval.bval = 1; return TOKEN_BOOLEAN;}
"false"                     {yylval.bval = 0; return TOKEN_BOOLEAN;}

[-]?[0-9]+                  {yylval.ival = atoi(yytext); return TOKEN_INTEGER;}
[-]?[0-9]*\.?[0-9]+([eE][-+]?[0-9]+)?    {yylval.dval = atof(yytext); return TOKEN_DOUBLE;}
[a-zA-Z]+                   {yylval.sval = strdup(yytext); return TOKEN_IDENTIFIER;}
\"[^\"]*\"                  {yylval.sval = strdup(yytext); return TOKEN_QUOTED_STRING;}


[ \t\n] ;                   /* skip space symbols */

";"                         { return END_OF_STATEMENT; }
```

```
%%
```

Некоторые токены должны проходить предобработку некоторую, а именно сохранение данных в yylval. Для некоторых токенов не нужна предобработка это обусловлено тем, что они должны распознавать константные значения и по факту возвращать токен указывающий на их наличие.

Был создан модуль parser.y, который определяет грамматику языка.
Сначала принимает наиболее общие правила:

```
any
  : select END_OF_STATEMENT {
      setAstRoot($1);
      $$ = $1;
      YYACCEPT;
  }
  | insert END_OF_STATEMENT {
      setAstRoot($1);
      $$ = $1;
      YYACCEPT;
  }
  | delete END_OF_STATEMENT {
      setAstRoot($1);
      $$ = $1;
      YYACCEPT;
  }
  | update END_OF_STATEMENT {
      setAstRoot($1);
      $$ = $1;
      YYACCEPT;
  }
  ;
```

Далее в зависимости от типа запроса (операции) он парсится дальше
Вот пример обработки delete запроса:

```
delete
  : TOKEN_DELETE TOKEN_FROM TOKEN_IDENTIFIER TOKEN_WHERE boolean_expression {
      AstNode *astDelete = createAstNode(DELETE_FROM, 1, createAstNode(FIELD_IDENTIFIER, 0, $3));
      AstNode *astCondition = createAstNode(DELETE_WHERE, 1, $5);
      $$ = createAstNode(DELETE_QUERY, 2, astDelete, astCondition);
  }
  ;
```

Создаем ноды непосредственно на операцию и на условия и тд.

Примеры обработки запросов:

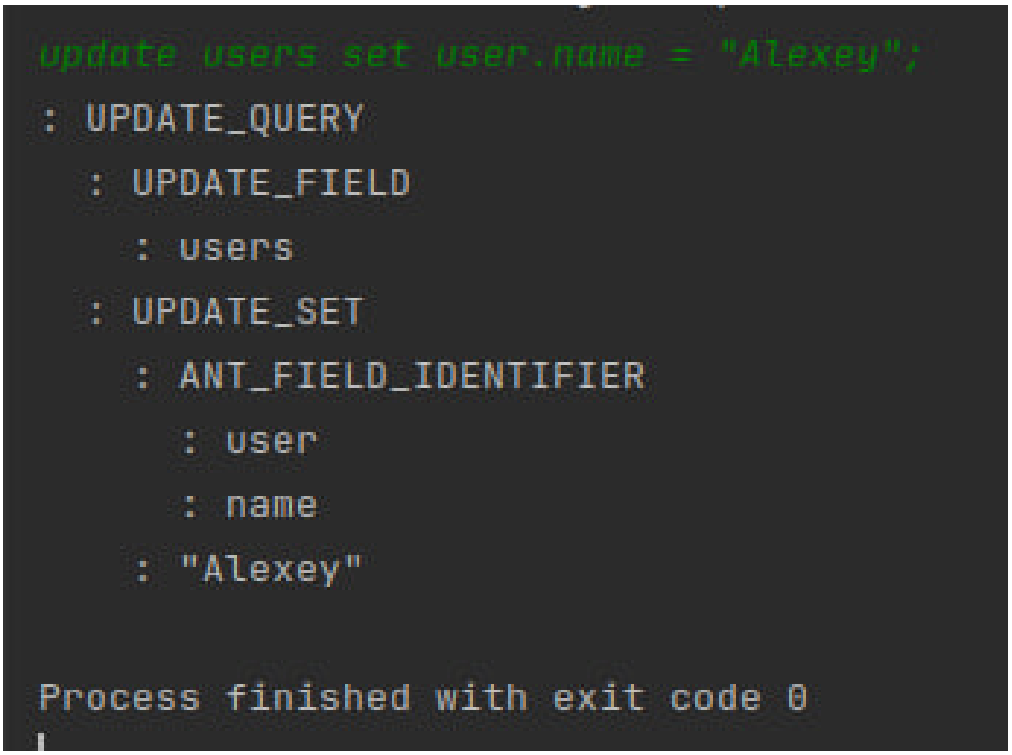
Insert:

```
insert into users values (20, "Nikita", true, 26.72, "Ivanov");
: INSERT_QUERY
  : INSERT_INTO
    : users
  : INSERT_VALUES
    : INSERT_VALUES
      : INSERT_VALUES
        : INSERT_VALUES
          : INSERT_VALUES
            : 20
            : "Nikita"
          : true
        : 26.720000
      : "Ivanov"

Process finished with exit code 0
```

Select:

```
from user in users where age == 20 && name == "Nikita" select user;
: SELECT_QUERY
  : FROM
    : FROM_VARNAME
      : user
    : FROM_COLLECTION_NAME
      : users
  : QUERY_BODY
    : WHERE
      : AND
        : EQ_OP
          : age
          : 20
        : EQ_OP
          : name
          : "Nikita"
    : SELECT
      : user

Process finished with exit code 0
```

Update:



Delete:

Contains:

```
delete from table where name.a.Contains("nov");
: DELETE_QUERY
  : DELETE_FROM
    : table
  : DELETE_WHERE
    : CONTAINS
      : ANT_FIELD_IDENTIFIER
        : name
        : a
      : "nov"

Process finished with exit code 0
```

Join:

```
from invoice in invoiceCollection join invoiceType in invoiceTypeCollection
    on invoice.invoiceTypeId equals invoiceType.id
        where (invoiceType.description.Contains("accounting")) || (invoice.finInstitutionId == 1234 && (invoice.isPassed == true)) select invoice.startTime;
: SELECT_QUERY
  : FROM
    : FROM_VARNAME
      : invoice
    : FROM_COLLECTION_NAME
      : invoiceCollection
  : QUERY_BODY
    : QUERY_BODY_CLAUSES
      : JOIN
        : JOIN_IN
          : invoiceType
          : invoiceTypeCollection
        : JOIN_ON
          : ANT_FIELD_IDENTIFIER
            : invoice
            : invoiceTypeId
          : ANT_FIELD_IDENTIFIER
            : invoiceType
            : id
      : WHERE
        : OR
          : CONTAINS
            : ANT_FIELD_IDENTIFIER
              : invoiceType
              : description
            : "accounting"
          : AND
            : EQ_OP
              : ANT_FIELD_IDENTIFIER
                : invoice
                : finInstitutionId
              : 1234
            : EQ_OP
              : ANT_FIELD_IDENTIFIER
                : invoice
                : isPassed
              : true
    : SELECT
      : ANT_FIELD_IDENTIFIER
        : invoice
        : startTime

Process finished with exit code 0
```

Выводы:

Flex и Bison прекрасно работают в связке, позволяют гибко настроить грамматику языка. Если грамотно продумать и изучить грамматику языка то можно красива ее описать и включить недостающие операции. В LINQ нет возможностей совершать манипуляции над данными, нельзя создавать обновлять и удалять элементы и эту грамматику пришлось взять из другого языка, который наиболее нам всем знаком - SQL. Подобный языковой анализ средствами flex и bison позволяем комбинировать грамматики разных языков и обрабатывать входной поток в соответствии с тем как мы задумали. Безусловно перед тем как все описать необходимо было изучить сам язык, как в этом языке взаимодействуют token-ы друг с другом, какие лексемы существуют.