Министерство науки и высшего образования Российской Федерации
Муромский институт (филиал)
Федерального государственного бюджетного образовательного учреждения высшего образования
«Владимирский государственный университет имени Александра Григорьевича и Николая Григорьевича Столетовых»
Факультет Информационных Технологий и Радиоэлектроники Кафедра программной инженерии

Курсовая работа

По дисциплине: «Теория автоматов и формальных языков»
Тема работы: «Транслятор с подмножества языка С»

Группа

ПИН – 121

Студент

Зубова И. О.

Цели и задачи курсовой работы

Целью курсовой работы является написание приложения – транслятора подмножества языка С.

Задачи курсовой работы:

- Составление грамматики языка;
- Реализация лексического и синтаксического анализатора, а также подпрограммы разбора сложных логических выражений;
- Разработка формы для взаимодействия с пользователем.

Описание языка

Алфавит языка С включает следующий набор символов:

- прописные и строчные буквы латинского алфавита;
- цифры от 0 до 9;
- знаки арифметических операций: «+, -, *, /, |, ^»;
- знаки операций отношения: «=, <,>, <>, <=, >=»;
- знаки препинания и разделители: «, . : ; ( )»;
- символ подчеркивания «_»;
- пробельные символы(пробел, табуляция, переход на новую строку).

В алфавит языка входят также зарезервированные слова, которые не могут быть использованы в качестве имен переменных или процедур.

В С существуют категории операций:

- объявление переменных,
- присвоение значений,
- арифметический оператор,
- логический оператор,
- условный оператор

Разработанная грамматика языка

G=T,N,P,<программа>

T={main, (,), {, int, =, ;, if, else, >, <, +, &&,}

N={<программа>,<спис_опис>,<опис>,<тип>,<спис_перем>,<опер>,

<условн.>,<блок опер.>,<присв.>,<знак>,<операнд>}

P = {

<программа> ::= main() { <спис_опис> <спис_опер> }

<спис_опис> ::= <опис> | <спис_опис>; <опис.>

<опис> ::= <тип> <спис_перем>

<тип> ::= int | bool | string

<спис_перем> ::= id | <спис_перем>, id

<спис_опер> ::= <опер> | <опер> <спис_опер>

<опер.> ::= <условн.> | <присв.>

<условн> ::= if expr <блок. опер.> | if expr <блок_опер> else <блок_опер>

<блок_опер> ::= <опер> | { <спис_опер> }

<присв> ::= id = <операнд> <знак> <операнд>; | id = <операнд>;

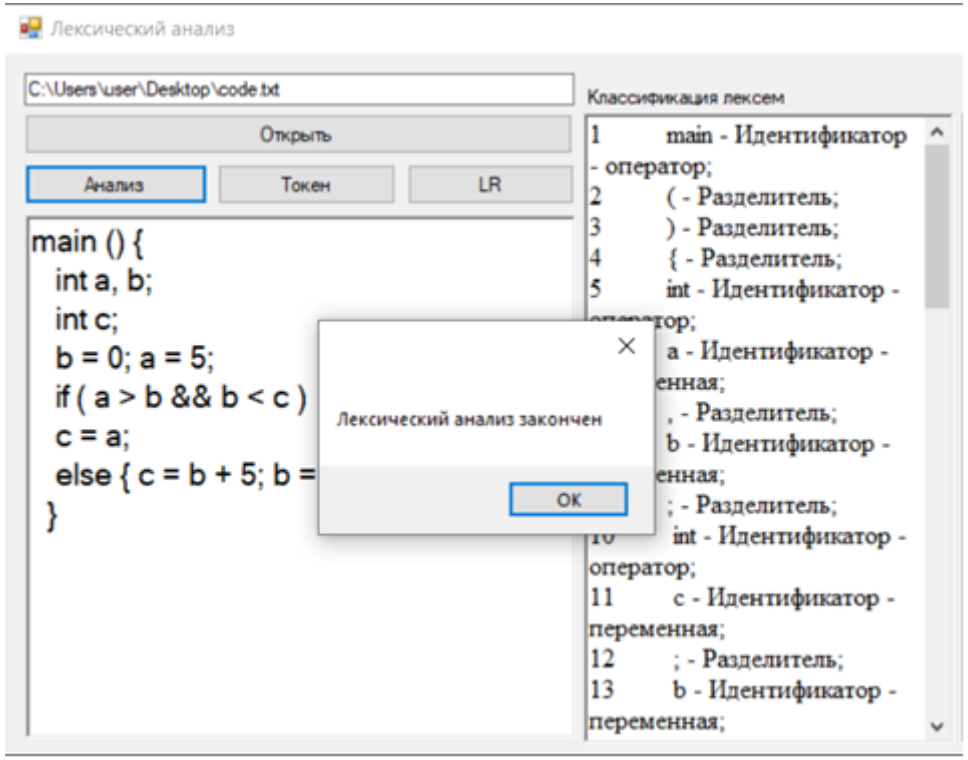
<знак> ::= + | - | \ | *

<операнд> ::= id |lit

}

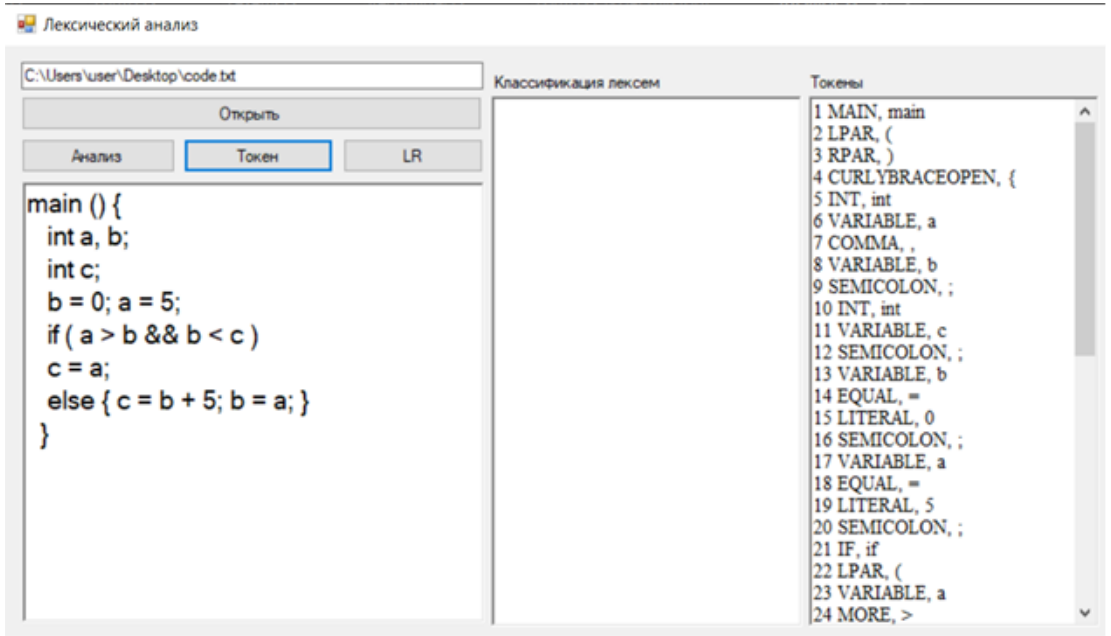
Лексический анализатор

Выделяет из текста лексемы различных типов: идентификаторы, литералы (числовые и символьные константы), разделители. Выделение (сборка) лексемы сопровождается проверкой её правильности.



Результат работы лексического анализатора

Реализация лексического анализатора



Результат работы лексического анализатора

```
foreach (char s in box2.Text)
        {
           //try
           {
              if (Lexems.IsOperator(subText) && (s == '(' || s == ' ' || s == '<' || s
== '>' || s == ';'))
              {
                 i++;
                 listBuf.Add(subText + " "); // - Идентификатор - оператор;
                 forToken.Add("I");
                 forChar.Add(' ');
                 subText = "";
              }
              else if (Lexems.IsLiteral(subText) && (s == ' ' || s == ';' || s == ')'))
              {
                 i++;
                 listBuf.Add(subText + " "); // - Литерал;
                 forToken.Add("D");
                 forChar.Add(' ');
                 subText = "";
              }
            }
            }
```

Синтаксический анализатор

Определяет порождается ли строка лексем данной грамматикой.

Восходящий синтаксический анализ для грамматики G = (T, N, P, ,<программа>) начинает разбор с конкретных слов (лексем) из множества T, связывая сначала пары слов, затем подсоединяет к этим парам новые слова или другие связанные пары образуя нетерминалы из множества N. Постепенно процесс связывания доходит до начального нетерминала, <программа> - то есть все лексемы оказываются связаны в единую структуру.

Граф состояний автомата

Для определения принадлежности грамматики к классу LR(k) были построены граф состояний автомата и решающая таблица детерминированного автомата.

Сост	Пред	Правила	переход
0	-	<программа> ::= • main() { <спис_опис>; <спис_опер> }	1
1	0	<программа> ::= main• () { <спис_опис>; <спис_опер> }	2
2	1	<программа> ::= main (•) { <спис_опис>; <спис_опер> }	3
3	2	<программа> ::= main ()•{ <спис_опис>; <спис_опер> }	4
4	3	<программа> ::= main (){•<спис_опис>; <спис_опер> } <спис_опис>::=• <опис> <опис> ::= • <тип> <спис_перем> <тип> ::= • int <тип> ::= • bool <тип> ::= • string <спис_опис> ::= • <спис_опис>; <опис>	5 6 7 8 9 10 5
5	4	<программа> ::= main (){<спис_опис>•; <спис_опер> } <спис_опис> ::= <спис_опис>•; <опис>	11 11

Реализация восходящего анализатора

Состояние	Стек разбора	Вход	Действие
24	<операнд> id Lit		Переход 31 Переход 32 Переход 33

```
private void State24() {
        switch (lexemStack.Peek().Type)
        {
            case TokenType.NETERM:
                switch (lexemStack.Peek().Value)
                {
                    case "<операнд>":
                        GoToState(31);
                        break;
                    default:
                        throw new Exception($"State24\n String: {nextLex} Ожидалось
нетерминал <операнд>, а получено {lexemStack.Peek().Type}
{lexemStack.Peek().Value}");
                }
                  case TokenType.LITERAL:
                    GoToState(33);
                    break;
            case TokenType.VARIABLE:
                GoToState(32);
                break;
            default:
                throw new Exception($"State23\n String: {nextLex} Ожидалось
терминал VARIABLE, LITERAL, но было получено {lexemStack.Peek().Type}
{lexemStack.Peek().Value}");
        }
    }
```

Реализация восходящего анализатора

Состояние	Стек разбора	Вход	Действие
12	<спис_перем.> <спис_перем.> ,	; , 	Свёртка(-2, <опис.>) Сдвиг Переход 21

```
private void State12() {
        switch (lexemStack.Peek().Type)  {
           case TokenType.NETERM: {
                 switch (lexemStack.Peek().Value) {
                    case "<спис_перем>":
                       switch (GetLexeme(nextLex).Type)  {
                          case TokenType.COMMA:
                             if (nextLex == tokens.Count)
                          throw new Exception($"State12\n String: {nextLex} ");
                             Shift();
                             break;
                          case TokenType.SEMICOLON:
                             Reduce(2, "<опис>");
                             break;
                          default:
                             throw new Exception($"State12\n String: {nextLex} Ожидалось SEMICOLON,
но было получено {lexemStack.Peek().Type}{lexemStack.Peek().Value}");
                       }
                       break;
                    default:
                       throw new Exception($"State12\n String: {nextLex} Ожидалось, <спис_перем>,
но было получено {lexemStack.Peek().Type}{lexemStack.Peek().Value}");
                 }
              }
              break;
           case TokenType.COMMA:
              GoToState(21);
              break;
           default:
              throw new Exception($"State12\n String: {nextLex} Ожидалось терминал COMMA, но
было получено {lexemStack.Peek().Type} {lexemStack.Peek().Value}");
        }
     }
```

Реализация восходящего анализатора

Состояние	Стек разбора	Вход	Действие
6	<опис.>		Свёртка(-1,< спис_опис >)

```
private void State6()
      {
         if (lexemStack.Peek().Type == TokenType.NETERM && lexemStack.Peek().Value == "<опис>")
            Reduce(1, "<спис_опис>");
         else
            throw new Exception($"State6\n String: {nextLex} Ожидалось нетерминал <опис>, но
было получено {lexemStack.Peek().Type} {lexemStack.Peek().Value}");
      }
```

Разбор сложных логических выражений

В данной курсовой работе требовалось выполнять разбор сложных логических выражений методом Дейктры.
Алгоритм метода Дейкстры состоит из следующих этапов:

- Просматриваем входную строку слева направо;
    - Если символ является операндом, то он добавляется к выходной строке;
    - Если символ является открывающейся скобкой, она помещается в стек;
    - Если символ является закрывающейся скобкой, выталкиваются элементы из стека в выходную строку до тех пор, пока на вершине стека не окажется открывающаяся скобка.
    - если символ операция, то
        - если приоритет операции равен нулю или больше приоритета операции, находящейся на вершине стека или стек пуст, то входная операция записывается в стек,
        - - иначе из стека выталкивается и переписывается в выходную строку операция, находящаяся на вершине, а также следующие за ней операции с приоритетами большими или равными приоритету входной операции. После этого входная операция добавляется к вершине стека.
- После просмотра всех символов входной строки происходит выталкивание всех оставшихся в стеке операций и дописывание их к выходной строке.

Таблица приоритетов операций.

Операция	Приоритет
(	0
)	1
||	2
&&	3
>, <, >=, <=, =, <>	4
+, -	5
*, /	6

Перевод обратной Польской нотации в матричный вид

Далее осуществляется перевод обратной Польской нотации в матричный вид по правилам:

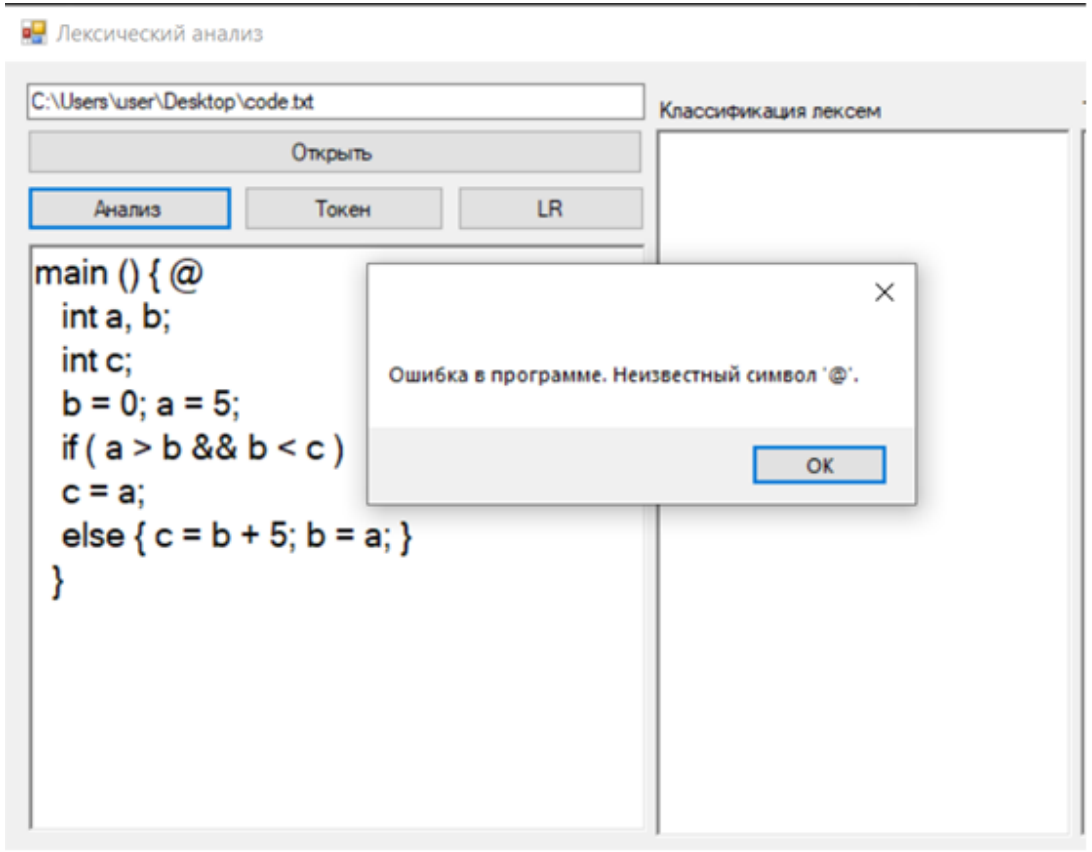
1. ОПН просматривается слева направо;
2. При чтении операнда – он записывается в стек операндов;
3. При чтении операции:

- из стека извлекается два верхних операнда;
- формируется тройка (операция, операнды) и записывается в матрицу операций;
- в стек записывается обозначение строки матрицы с полученной операцией.

Реализация подпрограммы разбора сложных логических выражений

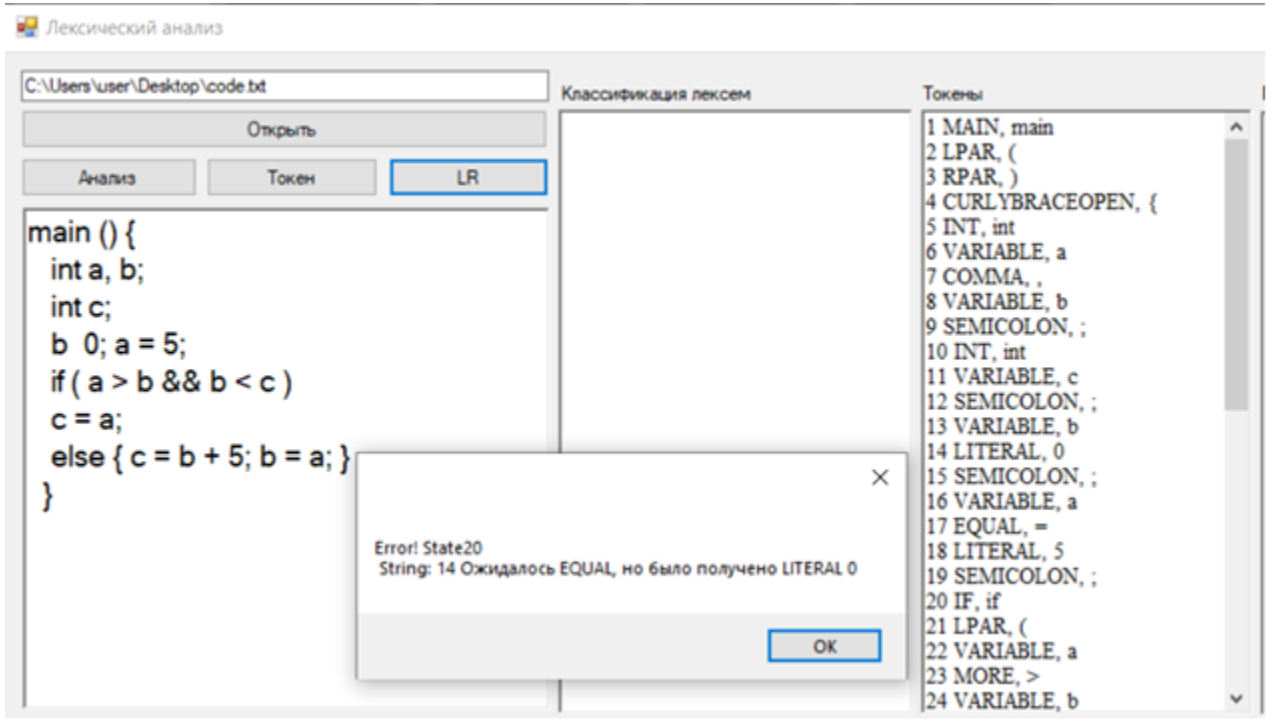
```
private void Decstra()
        if (logicExpressionStack[index].Type == Token.TokenType.LPAR)
{
            stackOfOperations.Push(logicExpressionStack[index].Value);
            priorityStack.Push(0);
            index++;
            //index++;
            while (index != logicExpressionStack.Count())
             {
               if (logicExpressionStack[index].Type ==
Token.TokenType.LITERAL || logicExpressionStack[index].Type ==
Token.TokenType.VARIABLE)
                {
                   output += logicExpressionStack[index].Value + " ";
                   index++;
                }
}
```

Демонстрация работы программы



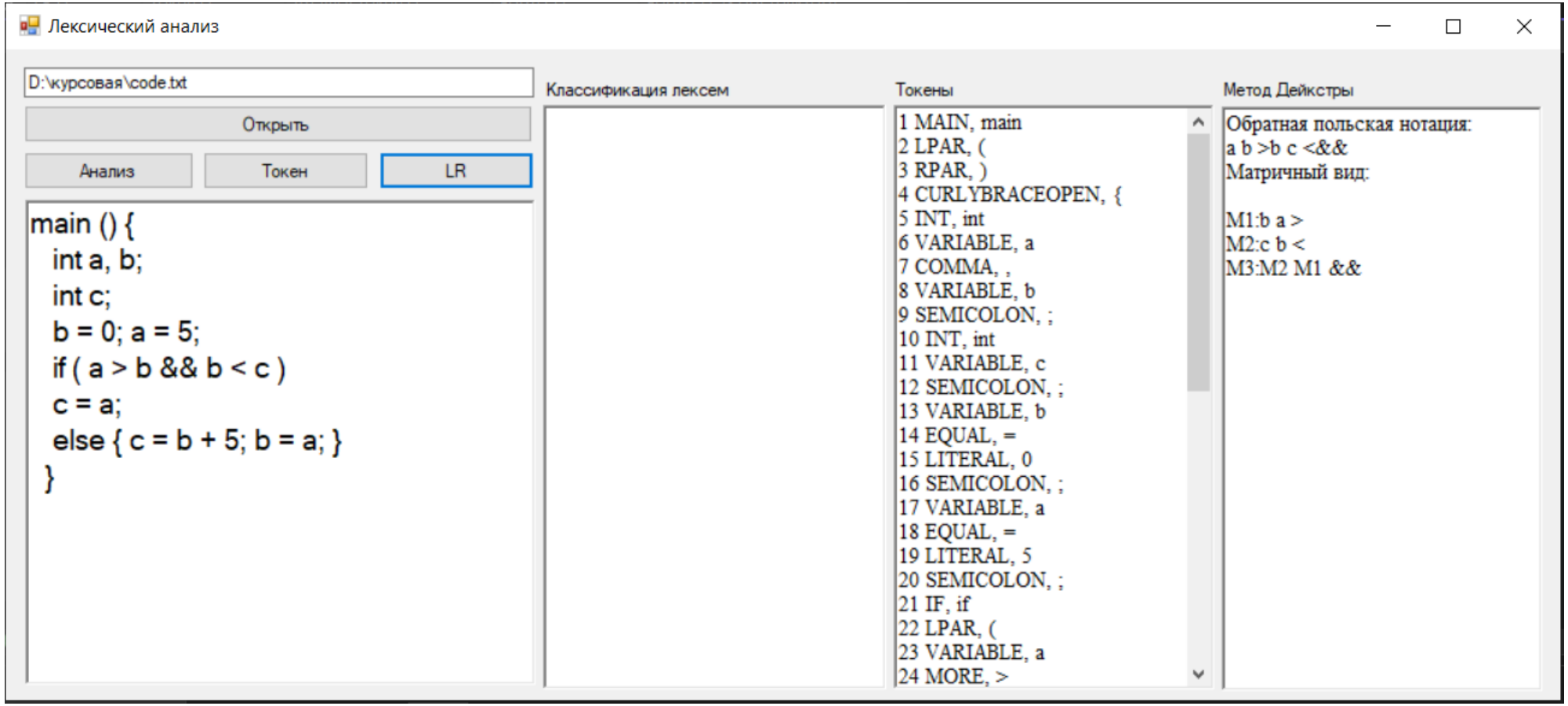
Обнаружение ошибки лексическим анализатором

Демонстрация работы программы



Обнаружение ошибки синтаксическим анализатором

Демонстрация работы программы



Результат успешного выполнения разбора
кода синтаксическим анализатором

Заключение

В результате работы создан транслятор программы на подмножестве языка С. Данная программа может выполнять лексический, синтаксический анализы, а также разбор сложного логического выражения.

Для синтаксического разбора был использован метод LR(k)-грамматик. Для разбора сложного арифметического выражения использован метод Дейкстры.

В ходе работы была составлена грамматика подмножества языка С, реализованы методы лексического и синтаксического разбора полученного кода, а также выполнено тестирование программы.

Для более подробного ознакомления с приложением можно воспользоваться ссылкой на репозиторий данного проекта:

https://github.com/plkkz/TranslyatorC

Подводя итоги, можно считать, что разработанный транслятор соответствует требованиям технического задания.