|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |
| Институт информационных технологий(ИТ)  Кафедра вычислительной техники(ВТ) | | |

**КУРСОВАЯ РАБОТА**

|  |  |
| --- | --- |
| по дисциплине | Теория формальных языков |
|  |  |
|  | *(наименование дисциплины)* |

|  |  |
| --- | --- |
| **Тема курсовой работы** | Разработка распознавателя модельного языка |
|  | программирования |
| *(наименование темы)* | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Студент группы** | ИКБО-15-21 | Сырых Я.И. |  |  |
|  | *(учебная группа) (Фамилия И.О.)* | |  | *(подпись студента)* |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Руководитель курсовой работы** | доцент каф. ВТ, к.т.н. | Унгер А.Ю. |  |  |
|  |  | |  | *(подпись руководителя)* |
| **Консультант** |  |  |  |  |
|  |  | |  | *(подпись консультанта)* |

|  |  |  |  |
| --- | --- | --- | --- |
| Работа представлена к защите | « » |  | 202\_ г. |
|  |  |  |  |
| Допущен к защите | « » |  | 202\_ г. |

Москва 202\_

Оглавление

[Введение 3](#_Toc126093773)

[Постановка задачи 5](#_Toc126093774)

[Порядок выполнения 7](#_Toc126093775)

[Грамматика языка 8](#_Toc126093776)

[Разработка лексического анализатора 10](#_Toc126093777)

[Разработка синтаксического анализатора 11](#_Toc126093778)

[Семантический анализ 13](#_Toc126093779)

[Тестирование программы 14](#_Toc126093780)

[Заключение 19](#_Toc126093781)

[Список использованной литературы 20](#_Toc126093783)

[Приложение 1 21](#_Toc126093784)

# Введение

История вычислительной техники насчитывает уже более полувека, и ведет свой отсчет с 1957 года. В этот год американский ученый Джон Бэкус разработал первый компилятор языка Фортран. При разработке он применил теорию формальных языков, опирающуюся на работы известного ученого-лингвиста Н. Хомского – автора классификации формальных языков. Формальные языки – это искусственно созданные языки для профессионального применения. Они, как правило, носят международный характер и имеют письменную форму. Примерами таких языков являются язык математики, язык химических формул, нотная грамота – язык музыки и др. Хомский в основном занимался изучением естественных языков, Бекус применил его теорию для разработки языка программирования. Это дало толчок к разработке сотен языков программирования. Несмотря на наличие большого количества алгоритмов, позволяющих автоматизировать процесс написания транслятора для формального языка, создание нового языка требует в большей степени творческого подхода. В основном это относится к синтаксису языка, который, с одной стороны, должен быть удобен в прикладном программировании, а с другой, должен укладываться в область контекстно-свободных языков, для которых существуют развитые методы анализа.

Целью данной курсовой работы является разработка распознавателя модельного языка программирования, согласно заданной формальной грамматике, основанной на индивидуальном варианте.

А также такие общие задачи, как:

-Освоение основных методов разработки распознавателей формальных языков на примере модельного языка программирования;

-Приобретение практических навыков написания транслятора языка программирования; закрепление практических навыков самостоятельного решения инженерных задач, умения пользоваться справочной литературой и технической документацией.

-Оттачивание практических навыков написания комплексных программ.

-Закрепления навыков написания программ на конкретных языках программирования.

# Постановка задачи

Разработать распознаватель модельного языка программирования

согласно заданной формальной грамматике. Распознаватель представляет собой специальных алгоритм, позволяющий вынести решение и принадлежности цепочки символов некоторому языку.

Распознаватель можно схематично представить в виде совокупности входной ленты, читающей головки, которая указывает на очередной символ на ленте, устройства управления (УУ) и дополнительной памяти (стек).

Конфигурацией распознавателя является:

- состояние УУ;

- содержимое входной ленты;

- положение читающей головки;

- содержимое дополнительной памяти (стека).

Трансляция исходного текста программы происходит в несколько этапов.

Основными этапами являются следующие:

- лексический анализ;

- синтаксический анализ;

- семантический анализ;

- генерация целевого кода.

Лексический анализ является наиболее простой фазой и выполняется с

помощью регулярной грамматики. Регулярным грамматикам соответствуют

конечные автоматы, следовательно, разработка и написание программы

лексического анализатора эквивалентна разработке конечного автомата и его

диаграммы состояний (ДС).

Синтаксический анализатор строится на базе контекстно-свободных

(КС) грамматик. Задача синтаксического анализатора – провести разбор текста

программы и сопоставить его с формальным описание языка.

Семантический анализ позволяет учесть особенности языка программирования, которые не могут быть описаны правилами КС- грамматики. К таким особенностям относятся:

- обработка описаний;

- анализ выражений;

- проверка правильности операторов.

Обработки описаний позволяет убедиться в том, что каждая переменная в программе описана и только один раз. Анализ выражений заключается в том, чтобы проверить описаны ли переменные, участвующие в выражении, и соответствуют ли типы операндов друг другу и типу операции.

# Порядок выполнения

1. В соответствии с номером варианта составить описание модельного языка программирования в виде правил вывода формальной грамматики;

2. Составить таблицу лексем и нарисовать диаграмму состояний для распознавания и формирования лексем языка;

3. Разработать процедуру лексического анализа исходного текста программы на языке высокого уровня;

4. Разработать процедуру синтаксического анализа исходного текста методом рекурсивного спуска на языке высокого уровня;

5. Построить программный продукт, читающий текст программы, написанной на модельном языке, в виде консольного приложения;

6. Протестировать работу программного продукта с помощи серии тестов, демонстрирующих все основные особенности модельного языка программирования, включая возможные лексические и синтаксические ошибки.

# Грамматика языка

Согласно индивидуальному варианту задания на курсовую работу грамматика языка включает следующие синтаксические конструкции:

1. <операции\_группы\_отношения> ::= <> | = | < | <= | > | >=

2. <операции\_группы\_сложения> ::= + | - | or

3. <операции\_группы\_умножения> ::= \* | / | and

4. <унарная\_операция> ::= not

5. <программа> ::= program var <описание> begin <оператор> {;<оператор>} end

6. <описание> ::= {<идентификатор> {, <идентификатор> } : <тип>;}

7. <тип> ::= % | ! | $

8. <оператор>::= <составной> | <присваивания> | <условный> | <фиксированного\_цикла> | <условного\_цикла> | <ввода> | <вывода>

9. <составной>::= «[» <оператор> { ( : | перевод строки) <оператор> } «]»

10. <присваивания>::= <идентификатор> as <выражение>

11. <условный>::= if <выражение> then <оператор> [ else <оператор>]

12. <фиксированного\_цикла>::= for <присваивания> to <выражение> do <оператор>

13. <условного\_цикла>::= while <выражение> do <оператор>

14. <ввода>::= read «(»<идентификатор> {, <идентификатор> } «)»

15. <вывода>::= write «(»<выражение> {, <выражение> } «)»

16. <выражение> ::= <операнд> {<операции\_группы\_отношения> <операнд>}

17. <операнд> ::= <слагаемое> {<операции\_группы\_сложения> <слагаемое>}

18. <слагаемое> ::= <множитель> {<операции\_группы\_умножения> <множитель>}

19. <множитель> ::= <идентификатор> | <число> | <логическая\_константа> | <унарная\_операция> <множитель> | «(» <выражение> «)»

20. <логическая\_константа> ::= true | false

21. <идентификатор>::= <буква> {<буква> | <цифра>}

22. <число>::= <целое> | <действительное>

23. <целое>::= <двоичное> | <восьмеричное> | <десятичное> | <шестнадцатеричное>

24. <двоичное>::= {/ 0 | 1 /} (B | b)

25. <восьмеричное>::= {/ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 /} (O | o)

26. <десятичное>::= {/ <цифра> /} [D | d]

27. <шестнадцатеричное>::= <цифра> {<цифра> | A | B | C | D | E | F | a | b | c | d | e | f} (H | h)

28 <действительное>::= <числовая\_строка> <порядок> |[<числовая\_строка>] . <числовая\_строка> [порядок]

29. <числовая\_строка>::= {/ <цифра> /}

30. <порядок>::= ( E | e )[+ | -] <числовая\_строка>

31. <буква> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v

| w | x | y | z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U |

V | W | X | Y | Z

32. <цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

\*для записи правил грамматики используется форма Бэкуса-Наура(БНФ). В записи БНФ левая и правая части порождения разделяются символом «::=», нетерминалы заключены в угловые скобки, а терминалы – просто символы, используемые в языке.

# Разработка лексического анализатора

Лексический анализатор – программа(или модуль/часть программы), которая принимает на вход исходный текст программы и выдает последовательность лексем – минимальных элементов программы, несущих смысловую нагрузку. В модельном языке программирования выделяют следующие типы лексем:

- ключевые слова;

- ограничители;

- числа;

- идентификаторы.

При разработке лексического анализатора, ключевые слова и ограничителя известны заранее, идентификаторы и числовые константы – вычисляются в момент разбора исходного текста. Для каждого типа лексем предусмотрена отдельная таблица. Таким образом, внутреннее представление лексемы – пара чисел (n, k), где n – номер таблицы лексем, k – номер лексемы в таблице.

Кроме того, в исходном коде программы кроме ключевых слов, идентификаторов и числовых констант может находиться произвольное число пробельных символов («пробел», «табуляция», «перенос строки», «возврат каретки») и комментариев, заключенных в фигурные скобки.

Лексический анализ текста проводится по регулярной грамматике. Известно, что регулярная грамматика эквивалентна конченому автомату, следовательно, для написания лексического анализатора необходимо построить диаграмму состояний, соответствующего конечного автомата (рис. 1).

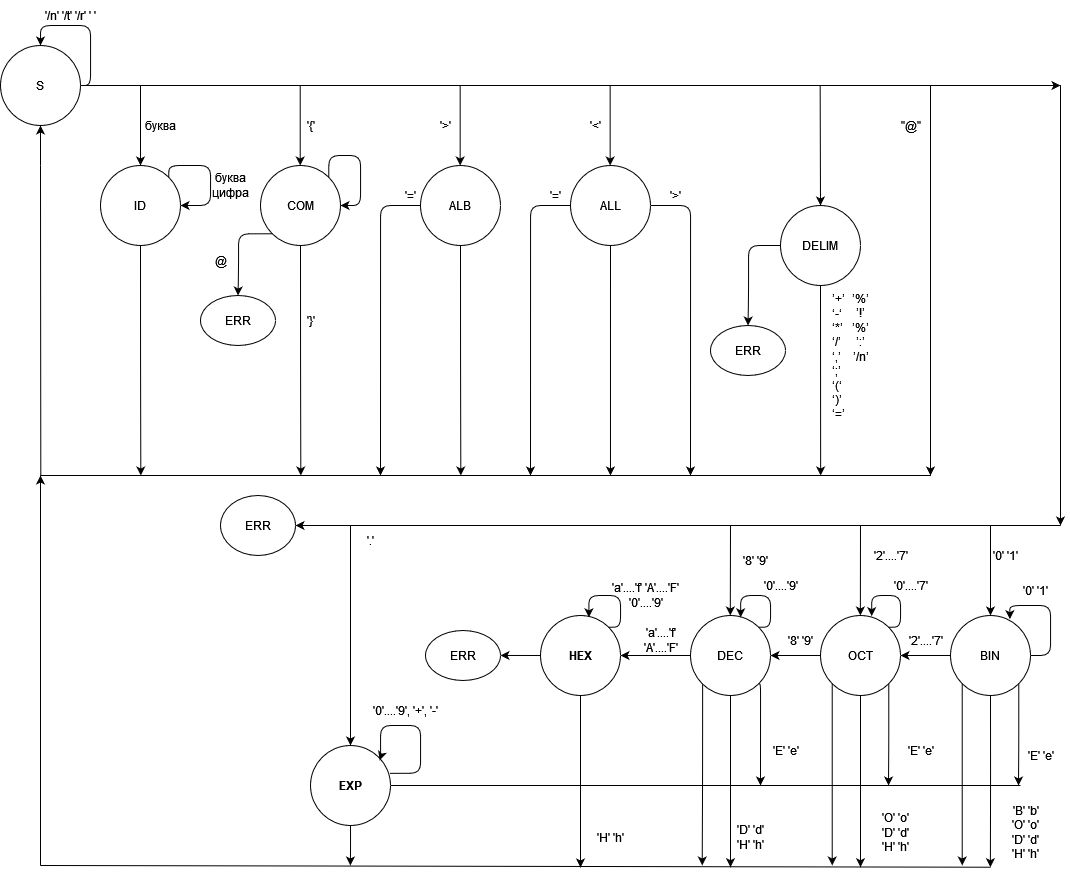


Рис. 1. Диаграмма состояний лексического анализатора

# Разработка синтаксического анализатора

Синтаксический анализатор принимает последовательность лексем, полученную в результате работы лексического анализатора, и проверяет правильность полученной последовательности лексем. Таким образом, разбор исходного текста программы идет последовательно, сохраняя перечень полученных лексем. Разработку синтаксического анализатора проведем с помощью метода рекурсивного спуска (далее сокращение РС). В основе метода лежит тот факт, что каждому нетерминалу ставится в соответствие рекурсивная функция. Для того чтобы в явном виде представить множество рекурсивных функций, перепишем грамматические правила следующим образом:

<программа> 𝑃 → 𝒑𝒓𝒐𝒈𝒓𝒂𝒎 𝐷1; 𝐵 ⊥

<описание> 𝐷1 → 𝒗𝒂𝒓 𝐷 {, 𝐷}

𝐷 → 𝐼 {, 𝐼}**:** **[% | ! | $**]

<оператор> 𝐵 → 𝒃𝒆𝒈𝒊𝒏 𝑆 {; 𝑆} 𝒆𝒏𝒅

𝑆 → So | Pr | U | Fc | Uc| Re | Wr

<составной> So → “[“S {( **:** | \n) S}”]”

<присваивания> Pr → I **as** E

<условный> U → **if** E **then** S [**else** S]

<фиксированного\_цикла> Fc → **for** Pr **to** E **do** S

<условного\_цикла> Uc → **while** E **do** S

<ввода> Re → **read**”(“I{,I}”)”

<вывода> Wr → **write**”(“E{,E}”)”

<выражение> 𝐸 → 𝐸1{[<> | = | < | <= | > | >=]𝐸1}

<операнд> 𝐸1 → 𝑇{[ + | − | 𝒐𝒓 ] 𝑇}

<слагаемое> 𝑇 → 𝐹{[ ∗ | / | 𝒂𝒏𝒅 ] 𝐹}

<множитель> 𝐹 → 𝐼 | 𝑁 | 𝐿| 𝒏𝒐𝒕 𝐹 | (𝐸)

<идентификатор> 𝐼 → C {C | R}

<число> N → N1 | N2

<целое> N1 → Nd | Ne | Nt | Ns

<двоичное> Nd → {/ 0 | 1 /} (‘B’ | ‘b’)

<восьмеричное> Ne→ {/ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 /} (‘O’ | ‘o’)

<десятичное> Nt → {/ R /} [‘D’ | ‘d’]

<шестнадцатеричное>Ns → R {R | C} (‘H’ | ‘h’)

<действительное> N2 → Rs Po | [Rs] . Rs [Po]

<числовая\_строка> Rs → {/ R /}

<порядок> Po → ( E | e )[+ | -] Rs

<буква>C → a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v

| w | x | y | z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U |

V | W | X | Y | Z

<цифра>R → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<логическая\_константа>L → **true** | **false**

Отметим, что правила для всех нетерминалов, начиная с I описаны на этапе лексического разбора. Следовательно, их описание на данном этапе не требуется.

# Семантический анализ

К сожалению. некоторые особенности модельного языка не могут быть

описаны контекстно-свободной грамматикой. К таким правилам относятся:

- любой идентификатор, используемый в программном коде, должен быть описан;

- повторное описание одного и того же идентификатора неразрешается.

Указанные особенности языка разбираются на этапе семантического анализа. Удобно процедуры семантического анализа совместить с процедурами синтаксического анализа. На практике это означает, что в рекурсивные функции встраиваются дополнительные контекстно-зависимые проверки.

Например, на этапе лексического анализа в таблицу идентификаторов заносятся данные обо всех лексемах- идентификаторах, которые встречаются в тексте программы. На этапе синтаксического анализа при получении каждого идентификатора информация о нем заносится в буферную память, если он еще не объявлен. В случае использования идентификатора, отсутствующего в буферной памяти, ошибка об этом выводится пользователю. В случае повторной инициализации идентификатора, уже присутствующего в буферной памяти, ошибка об этом также выводится пользователю.

# Тестирование программы

Ниже приведём результаты некоторого количества тестов, демонстрирующих работу написанной в рамках данной курсовой работы программы.

Представленные рисунки показывают общую работоспособность программы, а также отдельные аспекты работы программы и обработку лексических и семантических ошибок, в частности.

Листинг 1 – Корректный исходный код программы.

|  |
| --- |
| program  var  S1 : %,  B1 : $,  I1,I2, I3: !;  begin  I2 as 12E-5;  I2 as 100;  I2 as 100010B;  S1 as true;  [read(I1) : read(I1) : if B1 then write(I2) : I1 as fffH];  read(I1);  while I1 > I2 do write(I2)  end  @ |

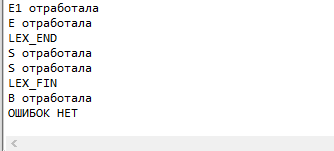


Рис. 2. Демонстрация работы программы, принимающей

на вход корректный код.

Таблица 1 – Полный вывод программы в отформатированном виде.

|  |  |  |  |
| --- | --- | --- | --- |
| Список ключевых слов(лексем) | Вывод считанных лексем | Считанные идентификаторы | Последовательность синтаксического анализа |
| 0 LEX\_NULL NULL  1 LEX\_AND and  2 LEX\_BEGIN begin  3 LEX\_DO do  4 LEX\_ELSE else  5 LEX\_END end  6 LEX\_AS as  7 LEX\_FOR for  8 LEX\_TO to  9 LEX\_IF if  10 LEX\_FALSE false  11 LEX\_DEC dec  12 LEX\_OCT oct  13 LEX\_BIN bin  14 LEX\_HEX hex  15 LEX\_EXP exp  16 LEX\_NOT not  17 LEX\_OR or  18 LEX\_PROGRAM program  19 LEX\_READ read  20 LEX\_THEN then  21 LEX\_TRUE true  22 LEX\_VAR var  23 LEX\_WHILE while  24 LEX\_WRITE write  25 LEX\_INT %  26 LEX\_FLOAT !  27 LEX\_BOOL $  28 LEX\_FIN @  29 LEX\_SEMICOLON ;  30 LEX\_COMMA ,  31 LEX\_COLON :  32 LEX\_LPAREN (  33 LEX\_RPAREN )  34 LEX\_EQ =  35 LEX\_LSS <  36 LEX\_GTR >  37 LEX\_PLUS +  38 LEX\_MINUS -  39 LEX\_TIMES \*  40 LEX\_SLASH /  41 LEX\_LEQ <=  42 LEX\_NEQ <>  43 LEX\_GEQ >=  44 LEX\_NUM num  45 LEX\_ID id  46 LEX\_LSSK [  47 LEX\_RSSK ] | ВЫВОД СПИСКА ЛЕКСЕМ:  LEX\_PROGRAM 18  LEX\_VAR 22  LEX\_ID 1  LEX\_COLON 31  LEX\_INT 25  LEX\_COMMA 30  LEX\_ID 2  LEX\_COLON 31  LEX\_BOOL 27  LEX\_COMMA 30  LEX\_ID 3  LEX\_COMMA 30  LEX\_ID 4  LEX\_COMMA 30  LEX\_ID 5  LEX\_COLON 31  LEX\_FLOAT 26  LEX\_SEMICOLON 29  LEX\_BEGIN 2  LEX\_ID 4  LEX\_AS 6  LEX\_EXP 12E-5  LEX\_SEMICOLON 29  LEX\_ID 4  LEX\_AS 6  LEX\_DEC 100  LEX\_SEMICOLON 29  LEX\_ID 4  LEX\_AS 6  LEX\_BIN 100010B  LEX\_SEMICOLON 29  LEX\_ID 1  LEX\_AS 6  LEX\_TRUE 21  LEX\_SEMICOLON 29  LEX\_LSSK 46  LEX\_READ 19  LEX\_LPAREN 32  LEX\_ID 3  LEX\_RPAREN 33  LEX\_COLON 31  LEX\_READ 19  LEX\_LPAREN 32  LEX\_ID 3  LEX\_RPAREN 33  LEX\_COLON 31  LEX\_IF 9  LEX\_ID 2  LEX\_THEN 20  LEX\_WRITE 24  LEX\_LPAREN 32  LEX\_ID 4  LEX\_RPAREN 33  LEX\_COLON 31  LEX\_ID 3  LEX\_AS 6  LEX\_ID 6  LEX\_RSSK 47  LEX\_SEMICOLON 29  LEX\_READ 19  LEX\_LPAREN 32  LEX\_ID 3  LEX\_RPAREN 33  LEX\_SEMICOLON 29  LEX\_WHILE 23  LEX\_ID 3  LEX\_GTR 36  LEX\_ID 4  LEX\_DO 3  LEX\_WRITE 24  LEX\_LPAREN 32  LEX\_ID 4  LEX\_RPAREN 33  LEX\_END 5  LEX\_FIN 28 | ВЫВОД СПИСКА ИДЕНТИФИКАТОРОВ:  S1  B1  I1  I2  I3 | Вхождение в P  Вхождение в D1  Вхождение в D  D отработала  Вхождение в D  D отработала  Вхождение в D  D отработала  D1 отработала  Вхождение в B  Вхождение в S  Вхождение в E  Вхождение в E1  Вхождение в T  Вхождение в F  F отработала  T отработала  E1 отработала  E отработала  S отработала  Вхождение в S  Вхождение в E  Вхождение в E1  Вхождение в T  Вхождение в F  F отработала  T отработала  E1 отработала  E отработала  S отработала  Вхождение в S  Вхождение в E  Вхождение в E1  Вхождение в T  Вхождение в F  F отработала  T отработала  E1 отработала  E отработала  S отработала  Вхождение в S  Вхождение в E  Вхождение в E1  Вхождение в T  Вхождение в F  F отработала  T отработала  E1 отработала  E отработала  S отработала  Вхождение в S  Вхождение в S  S отработала  Вхождение в S  S отработала  Вхождение в S  Вхождение в E  Вхождение в E1  Вхождение в T  Вхождение в F  F отработала  T отработала  E1 отработала  E отработала  Вхождение в S  Вхождение в E  Вхождение в E1  Вхождение в T  Вхождение в F  F отработала  T отработала  E1 отработала  E отработала  S отработала  S отработала  Вхождение в S  Вхождение в E  Вхождение в E1  Вхождение в T  Вхождение в F  F отработала  T отработала  E1 отработала  E отработала  S отработала  S отработала  Вхождение в S  S отработала  Вхождение в S  Вхождение в E  Вхождение в E1  Вхождение в T  Вхождение в F  F отработала  T отработала  E1 отработала  Вхождение в E1  Вхождение в T  Вхождение в F  F отработала  T отработала  E1 отработала  E отработала  Вхождение в S  Вхождение в E  Вхождение в E1  Вхождение в T  Вхождение в F  F отработала  T отработала  E1 отработала  E отработала  S отработала  S отработала  B отработала |
| **ОШИБОК НЕТ** | | | |

Листинг 2 – Bсходный код программы c синтаксической ошибкой.

(\* вместо лексемы типа)

|  |
| --- |
| program  var  S1 : **\***,  B1 : $,  I1,I2, I3: !;  begin  I2 as 12E-5;  I2 as 1000;  I2 as 100010B;  S1 as true;  [read(I1) : read(I1) : if B1 then write(I2) : I1 as FFFH];  read(I1);  while I1 > I2 do write(I2)  end  @ |

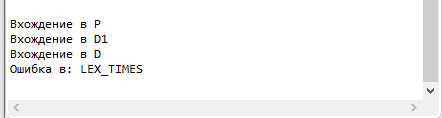


Рис. 3. Демонстрация работы программы, принимающей

на вход код с лексической ошибкой.

Листинг 3 – Bсходный код программы c лексической ошибкой.

(не корректное название переменной)

|  |
| --- |
| program  var  S**?**1 : %,  B1 : $,  I1,I2, I3: !;  begin  I2 as 12E-5;  I2 as 1000;  I2 as 100010B;  S1 as true;  [read(I1) : read(I1) : if B1 then write(I2) : I1 as FFFH];  read(I1);  while I1 > I2 do write(I2)  end  @ |

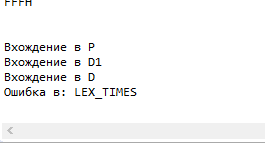


Рис. 4. Демонстрация работы программы, принимающей

на вход код с лексической ошибкой.

Листинг 4 – Bсходный код программы c лексической ошибкой.

(целочисленной переменной присваивается логическая константа)

|  |
| --- |
| program  var  S1 : %,  B1 : $,  I1,I2, I3: !;  begin  I2 as 12E-5;  **I2 as true**;  I2 as 100010B;  S1 as true;  [read(I1) : read(I1) : if B1 then write(I2) : I1 as FFFH];  read(I1);  while I1 > I2 do write(I2)  end  @ |

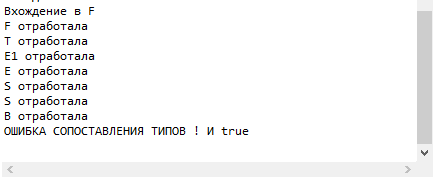


Рис. 6. Демонстрация работы программы, принимающей

на вход код с семантической ошибкой.

# Заключение

В результате выполнения курсовой работы мы закрепили знания, полученные в ходе учебного курса по соответствующим дисциплинам, закрепили на практике знания о алгоритмах работы и методах проектирования трансляторов, лексических, синтаксических и семантических анализаторов. Были закреплены, а также по необходимости дополнительно получены знания об особенностях языка программирования Java. Также в ходе работы изучены основные принципы построения интеллектуальных систем на основе теории автоматов и формальных грамматик, приобретены навыки лексического, синтаксического и семантического анализа предложений языков программирования в общем виде, на примере заданного персональным вариантом модельного языка программирования.

# Список использованной литературы

1. Свердлов С. З. Языки программирования и методы трансляции: учебное пособие. – Санкт-Петербург: Лань, 2019.
2. Малявко А. А. Формальные языки и компиляторы: учебное пособие для вузов. – М.: Юрайт, 2020.
3. Миронов С. В. Формальные языки и грамматики: учебное пособие для студентов факультета компьютерных наук и информационных технологий. – Саратов: СГУ, 2019.
4. Антик М. И., Казанцева Л. В. Теория формальных языков в проектировании трансляторов: учебное пособие. – М.: МИРЭА, 2020.
5. Ахо А. В., Лам М. С., Сети Р., Ульман Дж. Д. Компиляторы: принципы, технологии и инструментарий. – М.: Вильямс, 2008.
6. Ишакова Е.Н. Теория языков программирования и методов трансляции: учебное пособие. – Оренбург: ИПК ГОУ ОГУ, 2007.

# Приложение 1

**Листинг кода созданной в рамках курсовой работы программы.**

*import java.util.\*;*

*import java.io.\*;*

*enum lex\_type*

*{*

*LEX\_NULL, // 0*

*// КЛЮЧЕВЫЕ СЛОВА*

*LEX\_AND, //1*

*LEX\_BEGIN, //2*

*LEX\_DO, //3*

*LEX\_ELSE, //4*

*LEX\_END, //5*

*LEX\_AS, //6*

*LEX\_FOR, //7*

*LEX\_TO, //8*

*LEX\_IF, //9*

*LEX\_FALSE, //10*

*LEX\_DEC, //11*

*LEX\_OCT, //12*

*LEX\_BIN, //13*

*LEX\_HEX, //14*

*LEX\_EXP, //15*

*LEX\_NOT, //16*

*LEX\_OR, //17*

*LEX\_PROGRAM, //18*

*LEX\_READ, //19*

*LEX\_THEN, //20*

*LEX\_TRUE, //21*

*LEX\_VAR, //22*

*LEX\_WHILE, //23*

*LEX\_WRITE, //24*

*LEX\_INT,*

*LEX\_FLOAT,*

*LEX\_BOOL,*

*// МАРКЕР КОНЦА ТЕКСТА ПРОГРАММЫ*

*LEX\_FIN, //25*

*// ОПЕРАТОРЫ И РАЗДЕЛИТЕЛИ*

*LEX\_SEMICOLON, //26*

*LEX\_COMMA, //27*

*LEX\_COLON, //28*

*LEX\_LPAREN, //29*

*LEX\_RPAREN, //30*

*LEX\_EQ, //31*

*LEX\_LSS, //32*

*LEX\_GTR, //33*

*LEX\_PLUS, //34*

*LEX\_MINUS, //35*

*LEX\_TIMES, //36*

*LEX\_SLASH, //37*

*LEX\_LEQ, //38*

*LEX\_NEQ, //39*

*LEX\_GEQ, //40*

*LEX\_NUM, //41*

*LEX\_ID, //42*

*LEX\_LSSK, //41*

*LEX\_RSSK //42*

*};*

*class Lex*

*{*

*public*

*lex\_type type;*

*String value;*

*Lex(lex\_type t, String string)*

*{*

*type = t;*

*value = string;*

*}*

*Lex()*

*{*

*lex\_type t = lex\_type.LEX\_NULL; String v = "0";*

*type = t;*

*value = v;*

*}*

*lex\_type getType()*

*{*

*return type;*

*}*

*String getValue()*

*{*

*return value;*

*}*

*};*

*//Класс "Идентификатор"*

*class Id*

*{*

*public*

*String name;*

*boolean declared;*

*boolean assigned;*

*lex\_type type;*

*String value;*

*Id(String N)*

*{*

*name = N;*

*declared = false;*

*assigned = false;*

*type = lex\_type.LEX\_ID;*

*}*

*String getName()*

*{*

*return name;*

*}*

*void setName(String Nname)*

*{*

*name = Nname;*

*}*

*lex\_type getType()*

*{*

*return type;*

*}*

*void setType(lex\_type Ntype)*

*{*

*type = Ntype;*

*}*

*boolean isDeclared()*

*{*

*return declared;*

*}*

*void setDeclared()*

*{*

*declared = true;*

*}*

*boolean isAssigned()*

*{*

*return assigned;*

*}*

*void setAssigned()*

*{*

*assigned = true;*

*}*

*String getValue()*

*{*

*return value;*

*}*

*void setValue(String Nvalue)*

*{*

*value = Nvalue;*

*}*

*};*

*public class Main {*

*public static lex\_type[] keys = new lex\_type[]{*

*lex\_type.LEX\_NULL, // 0*

*// КЛЮЧЕВЫЕ СЛОВА*

*lex\_type.LEX\_AND, //1*

*lex\_type.LEX\_BEGIN, //2*

*lex\_type.LEX\_DO, //3*

*lex\_type.LEX\_ELSE, //4*

*lex\_type.LEX\_END, //5*

*lex\_type.LEX\_AS, //6*

*lex\_type.LEX\_FOR, //7*

*lex\_type.LEX\_TO, //8*

*lex\_type.LEX\_IF, //9*

*lex\_type.LEX\_FALSE, //10*

*lex\_type.LEX\_DEC, //11*

*lex\_type.LEX\_OCT, //12*

*lex\_type.LEX\_BIN, //13*

*lex\_type.LEX\_HEX, //14*

*lex\_type.LEX\_EXP, //15*

*lex\_type.LEX\_NOT, //16*

*lex\_type.LEX\_OR, //17*

*lex\_type.LEX\_PROGRAM, //18*

*lex\_type.LEX\_READ, //19*

*lex\_type.LEX\_THEN, //20*

*lex\_type.LEX\_TRUE, //21*

*lex\_type.LEX\_VAR, //22*

*lex\_type.LEX\_WHILE, //23*

*lex\_type.LEX\_WRITE, //24*

*lex\_type.LEX\_INT,*

*lex\_type.LEX\_FLOAT,*

*lex\_type.LEX\_BOOL,*

*// МАРКЕР КОНЦА ТЕКСТА ПРОГРАММЫ*

*lex\_type.LEX\_FIN, //25*

*// ОПЕРАТОРЫ И РАЗДЕЛИТЕЛИ*

*lex\_type.LEX\_SEMICOLON, //26*

*lex\_type.LEX\_COMMA, //27*

*lex\_type.LEX\_COLON, //28*

*lex\_type.LEX\_LPAREN, //29*

*lex\_type.LEX\_RPAREN, //30*

*lex\_type.LEX\_EQ, //31*

*lex\_type.LEX\_LSS, //32*

*lex\_type.LEX\_GTR, //33*

*lex\_type.LEX\_PLUS, //34*

*lex\_type.LEX\_MINUS, //35*

*lex\_type.LEX\_TIMES, //36*

*lex\_type.LEX\_SLASH, //37*

*lex\_type.LEX\_LEQ, //38*

*lex\_type.LEX\_NEQ, //39*

*lex\_type.LEX\_GEQ, //40*

*lex\_type.LEX\_NUM, //41*

*lex\_type.LEX\_ID, //42*

*lex\_type.LEX\_LSSK, //41*

*lex\_type.LEX\_RSSK //42*

*};*

*public static String[] KeyWord = new String[] {"NULL", "and", "begin", "do", "else", "end", "as","for","to","if","false","dec","oct","bin","hex","exp","not","or","program","read","then","true","var","while","write","%",*

*"!","$","@",";",",",":","(",")","=","<",">","+","-","\*","/","<=","<>",">=","num","id","[","]"};*

*static ArrayList<Id> IDs = new ArrayList<Id>();*

*static ArrayList<Lex> Lexems = new ArrayList<Lex>();*

*static ArrayList<Character> BUFF = new ArrayList<Character>();*

*static int look() {*

*char[] charArray = new char[BUFF.size()];*

*for(int i = 0; i<BUFF.size();i++) {*

*charArray[i] = BUFF.get(i);*

*}*

*//System.out.print(String.copyValueOf(charArray)); System.out.println(" "+KeyWord.length);*

*for (int i=0; i<KeyWord.length;i++) {*

*String S1 = KeyWord[i];*

*String S2 = String.copyValueOf(charArray);*

*//System.out.println(S1 + " " + S2 +" " + (S1.equals(S2)));*

*if(S1.equals(S2)) {*

*return i;*

*}*

*}*

*return 0;*

*}*

*static char read(FileReader reader) throws IOException {*

*char res = (char) reader.read();*

*//System.out.println("Прочитан символ: " + res);*

*return res;*

*}*

*public static char c= ' ';*

*static Lex lexer(FileReader reader) {*

*try*

*{*

*String CS = "S";*

*boolean flag;*

*int j;*

*do {*

*switch(CS) {*

*case("S"):*

*if (c == ' ' || c == '\n' || c == '\r' || c == '\t' || (c + 0) == 13) {*

*c=read(reader);*

*}*

*else if (Character.isLetter(c)) {*

*BUFF.clear();*

*BUFF.add(c);*

*c=read(reader);*

*CS = "ID";*

*}*

*else if (Character.isDigit(c)) {*

*BUFF.clear();*

*BUFF.add(c);*

*if(c =='1'|c =='0') {*

*CS = "BIN";}*

*else if(c =='2'|c =='3'|c =='4'|c =='5'|c =='6'|c =='7') {*

*CS = "OCT";}*

*else if(c =='8'|c =='9') {*

*CS = "DEC";*

*}*

*c=read(reader);*

*}*

*else if (c == '{') {*

*c=read(reader);*

*CS = "COM";*

*}*

*else if (c == '.') {*

*BUFF.clear();*

*BUFF.add(c);*

*c=read(reader);*

*CS = "EXP";*

*}*

*else if (c == '>') {*

*c=read(reader);*

*CS = "ALB";*

*}*

*else if (c == '<') {*

*c=read(reader);*

*CS = "ALL";*

*}*

*else if (c == '@') {*

*c=read(reader);Lex T1 = new Lex(lex\_type.LEX\_FIN, "28");*

*return T1;*

*}*

*else {*

*CS = "DELIM";*

*}*

*break;*

*case("ID"):*

*if (Character.isLetter(c) || Character.isDigit(c)) {*

*BUFF.add(c);*

*c=read(reader);*

*}*

*else {*

*j = look();*

*if (j!=0) {*

*Lex T1 = new Lex(keys[j], Integer.toString(j));*

*return T1;*

*}*

*else {*

*int I = 0;*

*boolean FL = false;*

*char[] charArray = new char[BUFF.size()];*

*for(int i = 0; i<BUFF.size();i++) {*

*charArray[i] = BUFF.get(i);*

*}*

*Id T = new Id(String.copyValueOf(charArray));*

*for(int i = 0; i<IDs.size();i++) {*

*if(T.name.equals(IDs.get(i).name)) {*

*FL=true;*

*I = i+1;*

*}*

*}*

*if(FL) {*

*Lex T1 = new Lex(lex\_type.LEX\_ID, Integer.toString(I));*

*return T1;*

*}else {*

*IDs.add(T);*

*j = IDs.size();*

*Lex T1 = new Lex(lex\_type.LEX\_ID, Integer.toString(j));*

*return T1;}*

*}*

*}*

*break;*

*case("COM"):*

*if (c == '}') {*

*c=read(reader);*

*CS = "S";*

*break;*

*}*

*else if (c == '@' || c == '{') {*

*break;*

*}*

*else {*

*c=read(reader);*

*break;*

*}*

*case("ALB"):*

*if (c == '=') {*

*c=read(reader);Lex T1 = new Lex(lex\_type.LEX\_GEQ, "43");*

*return T1;*

*} else {*

*c=read(reader);Lex T1 = new Lex(lex\_type.LEX\_GTR, "36");*

*return T1;*

*}*

*case("ALL"):*

*if (c == '=') {*

*c=read(reader);Lex T1 = new Lex(lex\_type.LEX\_LEQ, "41");*

*return T1;*

*} else if (c == '>') {*

*c=read(reader);Lex T1 = new Lex(lex\_type.LEX\_NEQ, "42");*

*return T1;*

*} else {*

*c=read(reader);Lex T1 = new Lex(lex\_type.LEX\_LSS, "35");*

*return T1;*

*}*

*case("BIN"):*

*if (c == '0' | c =='1') {*

*} else if (c == '2' | c =='3' | c =='4' | c =='5' | c =='6' | c =='7') {*

*CS = "OCT";*

*} else if (c == '8' | c =='9' ) {*

*CS = "DEC";*

*}else if (c == 'A' | c =='a' | c =='C' | c =='c' | c =='F' | c =='f') {*

*CS = "HEX";*

*}else if (c == 'e' | c =='E' ) {*

*CS = "EXP";*

*}else if (c == 'e' | c =='E' ) {*

*CS = "EXP";*

*}else if (c == 'B' | c =='b') {*

*BUFF.add(c);*

*char[] charArray = new char[BUFF.size()];*

*for(int i = 0; i<BUFF.size();i++) {*

*charArray[i] = BUFF.get(i);*

*}*

*c=read(reader);Lex T1 = new Lex(lex\_type.LEX\_BIN, String.copyValueOf(charArray));*

*return T1;*

*} else if (c == 'O' | c =='o') {*

*BUFF.add(c);*

*char[] charArray = new char[BUFF.size()];*

*for(int i = 0; i<BUFF.size();i++) {*

*charArray[i] = BUFF.get(i);*

*}*

*c=read(reader);Lex T1 = new Lex(lex\_type.LEX\_OCT, String.copyValueOf(charArray));*

*return T1; }*

*else if (c == 'D' | c =='d') {*

*BUFF.add(c);*

*char[] charArray = new char[BUFF.size()];*

*for(int i = 0; i<BUFF.size();i++) {*

*charArray[i] = BUFF.get(i);*

*}*

*c=read(reader);Lex T1 = new Lex(lex\_type.LEX\_DEC, String.copyValueOf(charArray));*

*return T1; } else if (c == 'h' | c =='H') {*

*BUFF.add(c);*

*char[] charArray = new char[BUFF.size()];*

*for(int i = 0; i<BUFF.size();i++) {*

*charArray[i] = BUFF.get(i);*

*}*

*c=read(reader);Lex T1 = new Lex(lex\_type.LEX\_HEX, String.copyValueOf(charArray));*

*return T1; } else {*

*char[] charArray = new char[BUFF.size()];*

*for(int i = 0; i<BUFF.size();i++) {*

*charArray[i] = BUFF.get(i);*

*}*

*Lex T1 = new Lex(lex\_type.LEX\_DEC, String.copyValueOf(charArray));*

*return T1; }*

*BUFF.add(c);*

*c=read(reader);*

*break;*

*case("OCT"):*

*if (c == '0' |c == '1' | c == '2' | c =='3' | c =='4' | c =='5' | c =='6' | c =='7') {*

*} else if (c == '8' | c =='9' ) {*

*CS = "DEC";*

*}else if (c == 'A' | c =='a' | c == 'B' | c =='b' | c =='C' | c =='c' | c =='F' | c =='f') {*

*CS = "HEX";*

*}else if (c == 'e' | c =='E' ) {*

*CS = "EXP";*

*} else if (c == 'O' | c =='o') {*

*BUFF.add(c);*

*char[] charArray = new char[BUFF.size()];*

*for(int i = 0; i<BUFF.size();i++) {*

*charArray[i] = BUFF.get(i);*

*}*

*c=read(reader);Lex T1 = new Lex(lex\_type.LEX\_OCT, String.copyValueOf(charArray));*

*return T1; } else if (c == 'D' | c =='d') { //1111111111-*

*BUFF.add(c);*

*char[] charArray = new char[BUFF.size()];*

*for(int i = 0; i<BUFF.size();i++) {*

*charArray[i] = BUFF.get(i);*

*}*

*c=read(reader);Lex T1 = new Lex(lex\_type.LEX\_DEC, String.copyValueOf(charArray));*

*return T1; } else if (c == 'h' | c =='H') {*

*BUFF.add(c);*

*char[] charArray = new char[BUFF.size()];*

*for(int i = 0; i<BUFF.size();i++) {*

*charArray[i] = BUFF.get(i);*

*}*

*c=read(reader);Lex T1 = new Lex(lex\_type.LEX\_HEX, String.copyValueOf(charArray));*

*return T1; } else {*

*char[] charArray = new char[BUFF.size()];*

*for(int i = 0; i<BUFF.size();i++) {*

*charArray[i] = BUFF.get(i);*

*}*

*Lex T1 = new Lex(lex\_type.LEX\_DEC, String.copyValueOf(charArray));*

*return T1; }*

*BUFF.add(c);*

*c=read(reader);*

*break;*

*case("DEC"):*

*if (c == '0' |c == '1' | c == '2' | c =='3' | c =='4' | c =='5' | c =='6' | c =='7' | c == '8' | c =='9') {*

*} else if (c == 'A' | c =='a' | c == 'B' | c =='b' | c =='C' | c =='c' | c =='F' | c =='f') {*

*CS = "HEX";*

*}else if (c == 'e' | c =='E' ) {*

*CS = "EXP";*

*} else if (c == 'D' | c =='d') { //1111111111-*

*BUFF.add(c);*

*char[] charArray = new char[BUFF.size()];*

*for(int i = 0; i<BUFF.size();i++) {*

*charArray[i] = BUFF.get(i);*

*}*

*c=read(reader);Lex T1 = new Lex(lex\_type.LEX\_DEC, String.copyValueOf(charArray));*

*return T1; } else if (c == 'h' | c =='H') {*

*BUFF.add(c);*

*char[] charArray = new char[BUFF.size()];*

*for(int i = 0; i<BUFF.size();i++) {*

*charArray[i] = BUFF.get(i);*

*}*

*c=read(reader);Lex T1 = new Lex(lex\_type.LEX\_HEX, String.copyValueOf(charArray));*

*return T1; } else {*

*char[] charArray = new char[BUFF.size()];*

*for(int i = 0; i<BUFF.size();i++) {*

*charArray[i] = BUFF.get(i);*

*}*

*Lex T1 = new Lex(lex\_type.LEX\_DEC, String.copyValueOf(charArray));*

*return T1; }*

*BUFF.add(c);*

*c=read(reader);*

*break;*

*case("EXP"):*

*if (c == '0' |c == '1' | c == '2' | c =='3' | c =='4' | c =='5' | c =='6' | c =='7' | c == '8' | c =='9') {*

*} else if (c == '+' | c =='-' ) {*

*if(BUFF.contains("+")||BUFF.contains("-")) {*

*};*

*} else if (c == 'e' | c =='E' ) {*

*if(BUFF.contains("e")||BUFF.contains("E")) {*

*}}else {*

*char[] charArray = new char[BUFF.size()];*

*for(int i = 0; i<BUFF.size();i++) {*

*charArray[i] = BUFF.get(i);*

*}*

*Lex T1 = new Lex(lex\_type.LEX\_EXP, String.copyValueOf(charArray)); BUFF.clear();*

*return T1; }*

*BUFF.add(c);*

*c=read(reader);*

*break;*

*case("DELIM"):*

*BUFF.clear();*

*BUFF.add(c);*

*j = look();*

*if (j!=0) {*

*c=read(reader);Lex T1 = new Lex(keys[j], Integer.toString(j));*

*return T1;*

*}*

*else {*

*Id T = new Id(BUFF.toString());*

*IDs.add(T);*

*j = IDs.size();*

*c=read(reader);Lex T1 = new Lex(lex\_type.LEX\_ID, Integer.toString(j));*

*return T1;*

*}*

*}}*

*while(true);}*

*catch(IOException ex){*

*System.out.println(ex.getMessage());*

*}*

*return null;*

*}*

*/////*

*static Lex CL = new Lex();*

*static int counter =-1;*

*static boolean ERRF = false;*

*static void gl()*

*{*

*counter++;*

*CL = Lexems.get(counter);*

*//System.out.println(CL.type);*

*}*

*void gl(int i)*

*{*

*CL = Lexems.get(i);*

*}*

*static void P() {*

*System.out.println("Вхождение в P");*

*try {*

*if (CL.type == lex\_type.LEX\_PROGRAM) {*

*gl();*

*}*

*else {*

*throw new RuntimeException("Ошибка в: " + CL.type.toString());*

*}*

*D1();*

*if (CL.type == lex\_type.LEX\_SEMICOLON) {*

*gl();*

*}*

*else {*

*throw new RuntimeException("Ошибка в: " + CL.type.toString());*

*}*

*B();*

*if (CL.type != lex\_type.LEX\_FIN) {*

*throw new RuntimeException("Ошибка в: " + CL.type.toString());*

*}*

*} catch (RuntimeException e) {*

*System.out.println(e.getMessage());*

*ERRF = true;*

*}*

*if(!ERRF) System.out.println("ОШИБОК НЕТ");*

*};*

*static void D1() {*

*System.out.println("Вхождение в D1");*

*if (CL.type != lex\_type.LEX\_VAR) {*

*throw new RuntimeException("Ошибка в: " + CL.type.toString());*

*}*

*gl();*

*D();*

*while (CL.type == lex\_type.LEX\_COMMA) {*

*gl();*

*D();*

*}*

*System.out.println("D1 отработала");*

*};*

*static void D() {*

*System.out.println("Вхождение в D");*

*//reset();*

*if (CL.type != lex\_type.LEX\_ID) {*

*throw new RuntimeException("Ошибка в: " + CL.type.toString());*

*}*

*//push(c\_val);*

*gl();*

*while (CL.type == lex\_type.LEX\_COMMA) {*

*gl();*

*if (CL.type != lex\_type.LEX\_ID) {*

*throw new RuntimeException("Ошибка в: " + CL.type.toString());*

*}*

*else {*

*//push(c\_val);*

*gl();*

*}*

*}*

*if (CL.type != lex\_type.LEX\_COLON) {*

*throw new RuntimeException("Ошибка в: " + CL.type.toString());*

*}*

*gl();*

*if (CL.type == lex\_type.LEX\_INT) {*

*//this->dec(LEX\_INT);*

*gl();*

*}*

*else if (CL.type == lex\_type.LEX\_FLOAT) {*

*//this->dec(LEX\_BOOL);*

*gl();*

*}*

*else if (CL.type == lex\_type.LEX\_BOOL) {*

*//this->dec(LEX\_BOOL);*

*gl();*

*}*

*else {*

*throw new RuntimeException("Ошибка в: " + CL.type.toString());*

*}*

*System.out.println("D отработала");*

*};*

*static void B() {*

*System.out.println("Вхождение в B");*

*if (CL.type != lex\_type.LEX\_BEGIN) {*

*throw new RuntimeException("Ошибка в: " + CL.type.toString());*

*}*

*gl();*

*S();*

*while (CL.type == lex\_type.LEX\_SEMICOLON) {*

*gl();*

*S();*

*}*

*if (CL.type == lex\_type.LEX\_END) {*

*gl();*

*}*

*else {*

*throw new RuntimeException("Ошибка в: " + CL.type.toString());*

*}*

*System.out.println("B отработала");*

*};*

*static void S() {*

*System.out.println("Вхождение в S");*

*if (CL.type == lex\_type.LEX\_LSSK) {*

*gl();*

*S();*

*while (CL.type == lex\_type.LEX\_COLON ) {*

*gl();*

*S();*

*}*

*if (CL.type != lex\_type.LEX\_RSSK) {*

*throw new RuntimeException("Ошибка в: " + CL.type.toString());*

*}*

*gl();*

*}else if(CL.type == lex\_type.LEX\_ID){*

*gl();*

*if (CL.type != lex\_type.LEX\_AS) {*

*throw new RuntimeException("Ошибка в: " + CL.type.toString());*

*}*

*gl();*

*E();*

*}else if(CL.type == lex\_type.LEX\_IF){*

*gl();*

*E();*

*if (CL.type == lex\_type.LEX\_THEN) {*

*gl();*

*S();*

*if (CL.type == lex\_type.LEX\_ELSE) {*

*gl();*

*S();*

*}*

*}*

*} else if (CL.type == lex\_type.LEX\_FOR) {*

*if(CL.type == lex\_type.LEX\_ID){*

*gl();*

*if (CL.type != lex\_type.LEX\_AS) {*

*throw new RuntimeException("Ошибка в: " + CL.type.toString());*

*}*

*E();*

*}*

*if (CL.type != lex\_type.LEX\_TO) {*

*throw new RuntimeException("Ошибка в: " + CL.type.toString());*

*}*

*gl();*

*E();*

*if (CL.type != lex\_type.LEX\_DO) {*

*throw new RuntimeException("Ошибка в: " + CL.type.toString());*

*}*

*gl();*

*S();*

*} else if (CL.type == lex\_type.LEX\_WHILE) {*

*gl();*

*E();*

*if (CL.type != lex\_type.LEX\_DO) {*

*throw new RuntimeException("Ошибка в: " + CL.type.toString());*

*}*

*gl();*

*S();*

*} else if (CL.type == lex\_type.LEX\_READ) {*

*gl();*

*if (CL.type != lex\_type.LEX\_LPAREN) {*

*throw new RuntimeException("Ошибка в: " + CL.type.toString());*

*}*

*gl();*

*if (CL.type != lex\_type.LEX\_ID) {*

*throw new RuntimeException("Ошибка в: " + CL.type.toString());*

*}*

*while (CL.type == lex\_type.LEX\_COMMA) {*

*gl();*

*if (CL.type != lex\_type.LEX\_ID) {*

*throw new RuntimeException("Ошибка в: " + CL.type.toString());*

*}*

*}*

*gl();*

*if (CL.type != lex\_type.LEX\_RPAREN) {*

*throw new RuntimeException("Ошибка в: " + CL.type.toString());*

*}*

*gl();*

*} else if (CL.type == lex\_type.LEX\_WRITE) {*

*gl();*

*if (CL.type != lex\_type.LEX\_LPAREN) {*

*throw new RuntimeException("Ошибка в: " + CL.type.toString());*

*}*

*gl();*

*E();*

*while (CL.type == lex\_type.LEX\_COMMA) {*

*gl();*

*E();*

*}*

*//gl();*

*if (CL.type != lex\_type.LEX\_RPAREN) {*

*throw new RuntimeException("Ошибка в: " + CL.type.toString());*

*}*

*gl();*

*} else {*

*throw new RuntimeException("Ошибка в: " + CL.type.toString());*

*}*

*System.out.println("S отработала");*

*};*

*static void E() {*

*System.out.println("Вхождение в E");*

*E1();*

*if (CL.type == lex\_type.LEX\_EQ || CL.type == lex\_type.LEX\_LSS ||*

*CL.type == lex\_type.LEX\_GTR || CL.type == lex\_type.LEX\_LEQ ||*

*CL.type == lex\_type.LEX\_GEQ || CL.type == lex\_type.LEX\_NEQ)*

*{*

*gl();*

*E1();*

*} System.out.println("E отработала");*

*}*

*static void E1() {*

*System.out.println("Вхождение в E1");*

*T();*

*while (CL.type == lex\_type.LEX\_PLUS || CL.type == lex\_type.LEX\_MINUS || CL.type == lex\_type.LEX\_OR) {*

*gl();*

*T();*

*} System.out.println("E1 отработала");*

*};*

*static void T() {*

*System.out.println("Вхождение в T");*

*F();*

*while (CL.type == lex\_type.LEX\_TIMES || CL.type == lex\_type.LEX\_SLASH || CL.type == lex\_type.LEX\_AND) {*

*gl();*

*F();*

*} System.out.println("T отработала");*

*};*

*static void F() {*

*System.out.println("Вхождение в F");*

*if (CL.type == lex\_type.LEX\_ID) {*

*gl();*

*}*

*else if (CL.type == lex\_type.LEX\_EXP | CL.type == lex\_type.LEX\_DEC | CL.type == lex\_type.LEX\_OCT |*

*CL.type == lex\_type.LEX\_BIN | CL.type == lex\_type.LEX\_HEX) {*

*gl();*

*}*

*else if (CL.type == lex\_type.LEX\_TRUE || CL.type == lex\_type.LEX\_FALSE) {*

*gl();*

*}*

*else if (CL.type == lex\_type.LEX\_NOT) {*

*gl();*

*F();*

*}*

*else if (CL.type == lex\_type.LEX\_LPAREN) {*

*gl();*

*E();*

*if (CL.type != lex\_type.LEX\_RPAREN) {*

*throw new RuntimeException("Ошибка в: " + CL.type.toString());*

*}*

*gl();*

*}*

*else {*

*throw new RuntimeException("Ошибка в: " + CL.type.toString());*

*} System.out.println("F отработала");*

*};*

*public static void main (String[] args) throws IOException*

*{*

*FileReader reader = new FileReader("C:/TXT/code2.txt");*

*lex\_type[] types = lex\_type.values();int i = 0; for (lex\_type s : types) { System.out.print(Integer.toString(i)+" "+ s);System.out.print(" ");System.out.println(KeyWord[i]); i++;}*

*Lex res;*

*while(true) {*

*res = lexer(reader);*

*Lexems.add(res);*

*if(Lexems.get(Lexems.size()-1).type == lex\_type.LEX\_FIN){*

*break;};*

*}*

*reader.close();*

*System.out.println("\n\nВЫВОД СПИСКА ЛЕКСЕМ: ");*

*for (Lex s : Lexems) {System.out.print(s.type.toString()+" ");*

*System.out.println(s.value);*

*};*

*System.out.println("\n\nВЫВОД СПИСКА ИДЕНТИФИКАТОРОВ: ");*

*for (Id d : IDs) {System.out.println(d.name + " ");*

*};*

*System.out.println("\n");*

*gl();*

*P();*

*return;*

*}*

*}*