

МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА
ВЕЛИКОГО»**

Институт компьютерных наук и кибербезопасности
Высшая школа технологий искусственного интеллекта

Отчёт по дисциплине «Математическая логика»

Лабораторная работа №1
«Синтаксический анализатор.».
Вариант №14

Студент: _____

Салимли Айзек Мухтар Оглы

Преподаватель: _____

Востров Алексей Владимирович

«____» _____ 20__ г.

Содержание

Введение	3
1 Постановка задачи	4
2 Математическое описание	5
2.1 Математическая модель программы	5
2.2 Грамматика Хомского	5
2.3 Типы грамматик Хомского	5
2.4 Диаграмма	6
3 Программная реализация	7
3.1 Математическая модель программной реализации	7
3.2 Реализация	7
3.3 Lib.hs	7
3.4 Main.hs	9
4 Результаты программы	10
4.1 Варианты	10
4.2 Исключения и ошибки анализа	12
4.3 Особые случаи	15
4.4 Семантика	16
4.5 Выбор текстового файла	16
Заключение	17
Список литературы	18

Введение

В отчете описана реализация web приложения синтаксического анализатора. Цель задачи состоит в реализации лексического анализатора, который будет проводить лексический анализ входного текста в соответствии с заданным вариантом. Программа порождает таблицу лексем с указанием их типов и значений. Реализация была дополнена иными ключевыми словами, операторами и функторами.

Был собран stack проект, код программы написан на языке Haskell2010, с конфигурацией Cabal 3.0.0, GHC 9.12.2 в интегрированной среде разработки visual studio code. Используемые библиотеки:

- base >= 4.14.0.1 && < 5 –стандартная библиотека
- threepenny-GUI –библиотека для создания веб интерфейса
- data.char –библиотека для работы с символами

Указанный вариант - №14.

Правила:

- Входной текст содержит операторы цикла while ... do и do ... while
- Разделитель символом ;
- Операторы условия содержат знаки сравнения =, >, <
- Вещественные числа
- Знак присваивания :=
- Вещественные числа могут начинаться с точки*

Дополнения:

- Монады
- Тип-данных Monad
- Тип-данных String
- Функторы: <>, < \$ >,, map, fmap
- Стрелка Клейсли: >>=
- Лямбда-функции: \n
- Семантика циклов: while ... do и do ... while

Не компилируемые лексемы:

- Комментарии типа: --
- Комментарии типа: //
- Комментарии типа: /* */

1 Постановка задачи

Написать программу, которая выполняет лексический анализ входного текста в соответствии с заданием и порождает таблицу лексем с указанием их типов и значений.

1. Подготовить несколько вариантов программы в виде текста на входном языке.
2. Программа должна выдавать сообщения о наличии во входном тексте ошибок, которые могут быть обнаружены на этапе лексического анализа.
3. Длина идентификатора и строковых констант ограничена 16 символами, только латиница.
4. Программа должна допускать наличие комментариев неограниченной длины во входном файле.
5. Построить синтаксические диаграммы.

2 Математическое описание

2.1 Математическая модель программы

Лексический анализатор принимает на вход строку символов w и выдает последовательность токенов $T = (t_1, t_2, \dots, t_n)$.

$$F_{\text{lexer}} : \Sigma^* \rightarrow T^*$$

, где:

- Σ^* - множество всех возможных строк над алфавитом Σ ;
- T^* - множество всех возможных последовательностей токенов;
- $t_i \in T$ - токены.

Лексический анализатор строится на основе регулярных языков и грамматик Хомского.

2.2 Грамматика Хомского

Формальная грамматика Хомского — это набор $G = (N, \Sigma, P, S)$, состоящий из:

- N - конечного множества нетерминалов;
- Σ - конечного множества терминальных символов (алфавит);
- P - множества продукций (правил);
- $S \in N$ - начального символа.

2.3 Типы грамматик Хомского

Существует четыре типа грамматик Хомского, но в контексте лексического анализа рассматриваются два:

Тип 3 (Регулярные грамматики):

$$A \rightarrow \alpha B \quad \text{или} \quad A \rightarrow \alpha$$

, где A, B — нетерминалы, а α — терминал.

Тип 2 (Контекстно-свободные грамматики):

$$A \rightarrow \gamma, \quad \text{где } A \text{ - нетерминал, } \gamma \text{ - последовательность терминалов и нетерминалов.}$$

Тип 1 (КЗ грамматики):

$$\alpha A \beta \rightarrow \alpha \gamma \beta$$

где A — нетерминал, α, γ, β — строки из $(N \cup \Sigma)^*$, и $|\gamma| \geq |\beta|$.

2.4 Диаграмма

На рисунке 1, представлена синтаксическая диаграмма последующей программы.

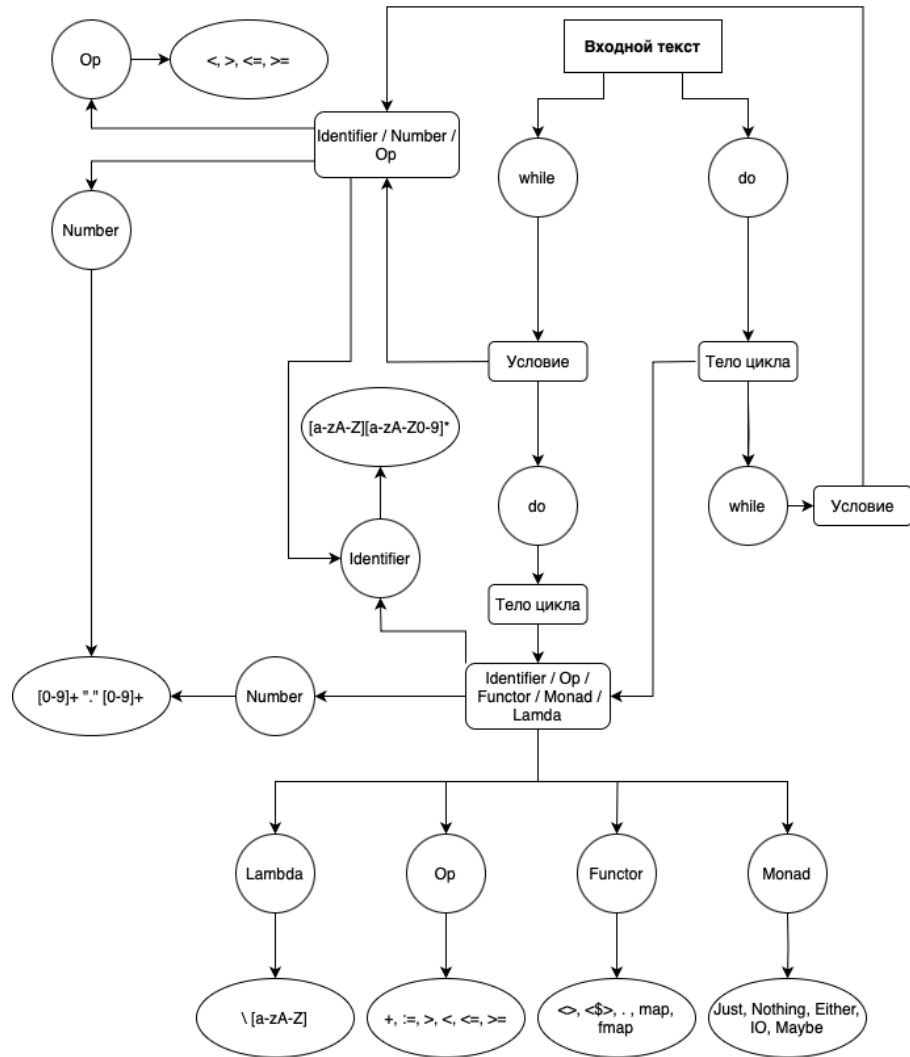


Рис. 1: Синтаксическая диаграмма

3 Программная реализация

3.1 Математическая модель программной реализации

Идея заключается в обозначении токенов типами из библиотеки `Token` из `Hackage`. Для каждого множества грамматики Хомского обозначим такие токены:

- $TKeyword \rightarrow \{ "while" "do" "String" "Monad" \}$
- $TIdentifier \rightarrow [a - zA - Z][a - zA - Z0 - 9]^*$
- $TReal \rightarrow [0 - 9]^+ \cup (\epsilon \mid .[0 - 9]^+)$
- $TOp \rightarrow \{ +, -, *, /, :=, <, >, <=, >=, ==, <> \}$
- $TComment \rightarrow //[\backslash n]^*$
- $TComment \rightarrow /\backslash * (.*?\backslash *)\backslash * /$
- $TSemicolon \rightarrow ;$
- $TString \rightarrow "[]^* "$

3.2 Реализация

Программа была разделена в проекте **stack** на управляющую логику (**Lib.hs**) и веб-интерфейс (**Main.hs**). В качестве веб-интерфейса была выбрана библиотека **threepenny-gui** из `Hackage`.

3.3 Lib.hs

Определение токенов (типов лексем):

```
1 data Token
2   = TWhile
3     | TDo
4     | TKeyword String
5     | TIdentifier String
6     | TReal Double
7     | TAssign
8     | TOp String
9     | TSemicolon
10    | TString String
11    | TComment String
12    | TLambda String
13    deriving (Show, Eq)
```

Функция `tokenLexeme :: Token -> String` преобразует токен в строку:

```
1 tokenLexeme :: Token -> String
2 tokenLexeme t = case t of
3   TWhile      -> "while"
4   TDo         -> "do"
5   TKeyword s  -> s
6   TIdentifier s -> s
7   TReal d     -> show d
8   TAssign     -> " :="
9   TOp op      -> op
10  TSemicolon  -> ";"
11  TString s   -> s
12  TComment s  -> s
13  TLambda s   -> s
```

Функция **lexer** разбирает строку на токены и ошибки, вызывая **lexInternal**:

```
1 lexer :: String -> ([Token], [String])
2 lexer input =
3     let (ts, es) = lexInternal input
4     in if not (null es)
5         then (ts, es)
6         else
7             let e2 = checkNonLatinForNonComment ts
8             in (ts, e2)
```

Функция **lexInternal** рекурсивно разбирает строку на токены:

```
1 lexInternal :: String -> ([Token],[String])
2 lexInternal [] = ([], [])
3 lexInternal (c:cs)
4     | isSpace c = lexInternal cs
5     | c=='/' && take 1 cs == "/" = ...
6     | c=='-' && take 1 cs == "-" = ...
7     | isAlpha c = lexIdentOrKeyword (c:cs)
8     | isDigit c || c=='.' = lexNumber (c:cs)
9     | otherwise =
10         let (toks, errs) = lexInternal cs
11         in ([], ("Unexpected character: " ++ [c]) : errs)
```

Функция **executeProgram** выполняет разбор программы и имитацию циклов:

```
1 executeProgram :: String -> Either String String
2 executeProgram s =
3     let (ts, es) = lexer s
4     in if not (null es)
5         then Left (unlines es)
6         else
7             let noComm = filter (not . isComment) ts
8             in case noComm of
9                 [ TWhile, TIdentifier i1, TOp "<", TReal lim, TDo, TSemicolon
10                  , TIdentifier i2, TAssign, TIdentifier i3, TOp "+", TReal st
11                  , TSemicolon]
12                 | i1==i2 && i2==i3 -> execWhileLoop 0 lim st 0
13
14                 [ TDo, TIdentifier i1, TAssign, TIdentifier i2, TOp "+", TReal
15                  st
16                  , TSemicolon, TWhile, TIdentifier i3, TOp "<", TReal lim,
17                  TSemicolon]
18                 | i1==i2 && i2==i3 -> execDoWhileLoop 0 lim st 0
19
20                 _ -> Left "Unsupported program format"
```


3.4 Main.hs

Содержит UI-функции библиотеки **Threepenny-GUI**. Функция `setup :: Window -> UI()` настраивает интерфейс:

```
1 setup window = do
2   return window # set title "Haskell UI"
3   on UI.click recognizeButton $ \_ -> do
4     txt <- get value inputBox
5     if all isSpace txt
6       then element outputDiv # set text "Enter text or press \"Generate\""
7     else do
8       let (tokens, errors) = lexer txt
9       if not (null errors)
10        then element outputDiv # set text ("Errors:\n" ++ unlines errors)
11      else do
12        let assigns = buildAssignMap tokens
13        let (comments, nonComments) = separateComments tokens
14        let (rows, _) = foldl
15          (\(acc, counter) tk ->
16            let (row3, newC) = tokenToRow assigns tk
17              counter
18            in (acc ++ [row3], newC))
19          ([], 1)
20          nonComments
21        tableMain <- buildMainTable rows
22        tableComm <- buildCommentTable comments
23        element outputDiv # set children [tableMain, tableComm]
```

Для предотвращения заикливания была создана функция сброса:

```
1 on UI.click resetButton $ \_ -> do
2   element inputBox # set value ""
3   element outputDiv # set text "Reset"
```

4 Результаты программы

На рисунках 2 - 13 представлены результаты выполнения лексического анализатора.

4.1 Варианты

```
-- Генерация while
while i < 6.705976273616398 do;
  i := i + 1.7667203630870305;
```

Распознать

Сгенерировать

Очистить

Решить пример

Сброс

Лексема	Тип лексемы	Значение
while	ключевое слово	X1
i	идентификатор	i : i
<	оператор сравнения	-
6.705976273616398	константа	6.705976273616398
do	ключевое слово	X2
;	разделитель	-
i	идентификатор	i : i
:=	знак присваивания	-
i	идентификатор	i : i
+	оператор присваивания	-
1.7667203630870305	константа	1.7667203630870305
;	разделитель	-

Не компилируемые:

Тип	Комментарий
--	- Генерация while

Рис. 2: Стандартный вариант

```
Monad m := IO;
while i <= 2
do \n.n+xi 5 <$> Maybe e := Just f <> fmap . list "Hello";
Either f . f + \xy Left x Right y
// HASKEEEEEEL
```

Распознать

Сгенерировать

Очистить

Решить пример

Сброс

Лексема	Тип лексемы	Значение
Monad	ключевое слово	X1
m	идентификатор	m : IO
:=	знак присваивания	-
IO	монада	X2
;	разделитель	-
while	ключевое слово	X3
i	идентификатор	i : не инициализирован
<=	оператор сравнения	-
2.0	константа	2.0
do	ключевое слово	X4
\n.n+xi	лямбда выражение	X5
5.0	константа	5.0
<\$>	функтор	-
Maybe	монада	X6
e	идентификатор	e : не инициализирован
:=	знак присваивания	-
Just	монада	X7
f	идентификатор	f : не инициализирован
<>	функтор	-
fmap	функтор	-
.	функтор	-
list	идентификатор	list : не инициализирован
Hello	строковая константа	Hello
;	разделитель	-
Either	монада	X8
f	идентификатор	f : не инициализирован
.	функтор	-
f	идентификатор	f : не инициализирован
+	оператор присваивания	-
\xy	лямбда выражение	X9
Left	монада	X10
x	идентификатор	x : не инициализирован
Right	монада	X11
y	идентификатор	y : не инициализирован

Не компилируемые:

Тип	Комментарий
//	/ HASKEEEEEEL

Рис. 3: Расширенный вариант

4.2 Исключения и ошибки анализа

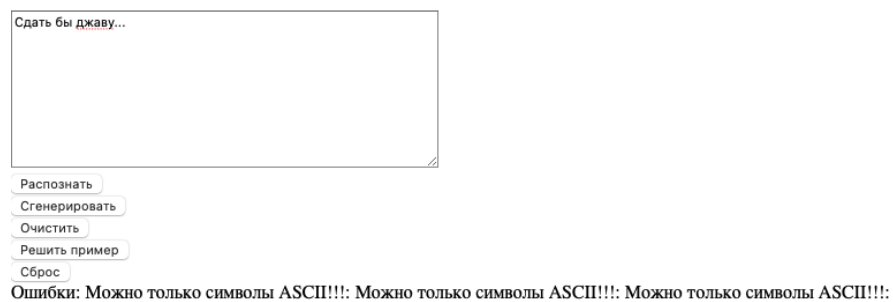


Рис. 4: Ошибка ввода кириллицы

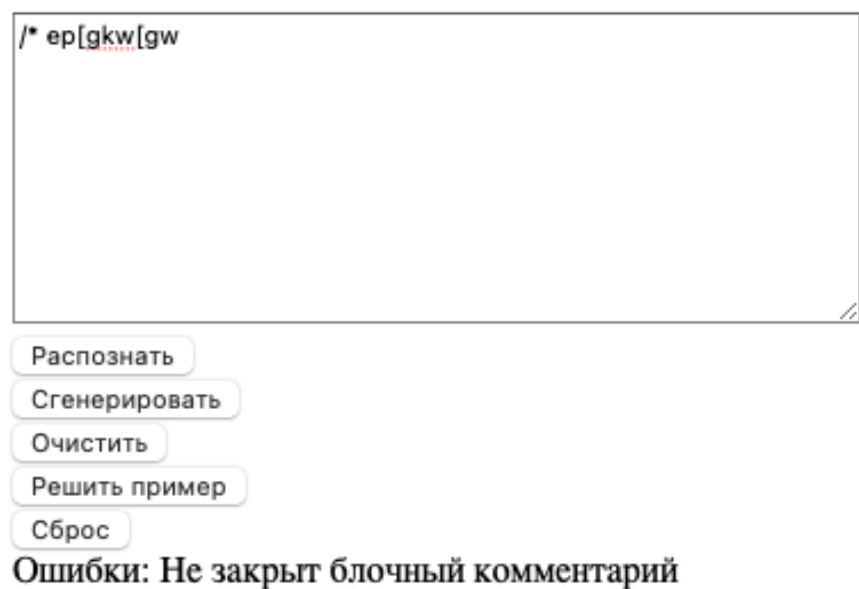


Рис. 5: Ошибка: не закрыт блочный комментарий

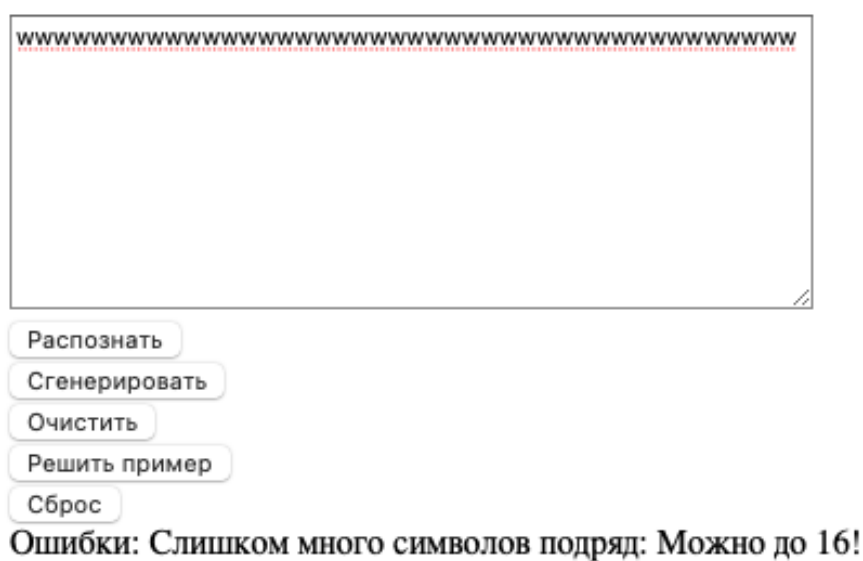


Рис. 6: Ошибка: длина лексемы больше 16



Рис. 7: Ошибка: не закрыта строка



Рис. 8: Ошибка: неверное начало идентификатора

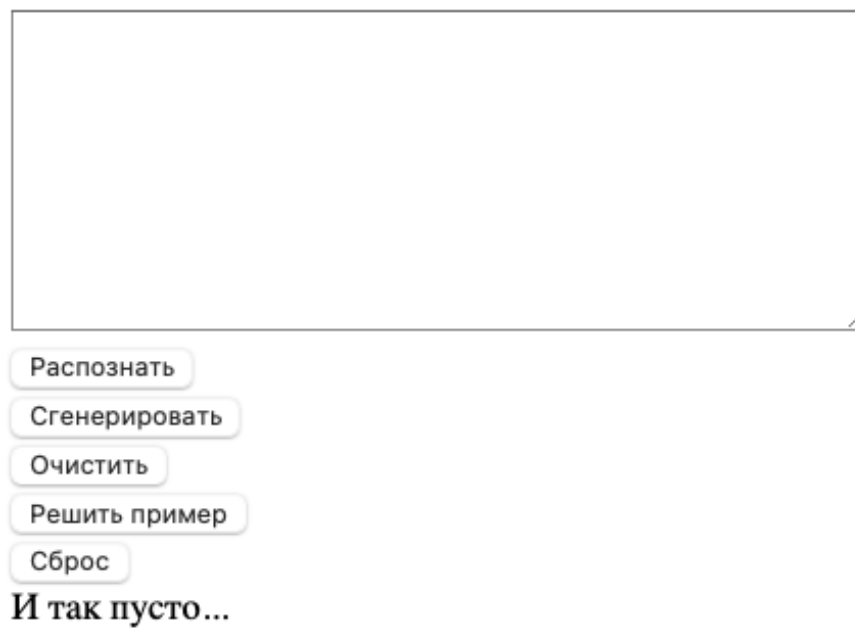


Рис. 9: Ошибка: попытка очистить пустое окно

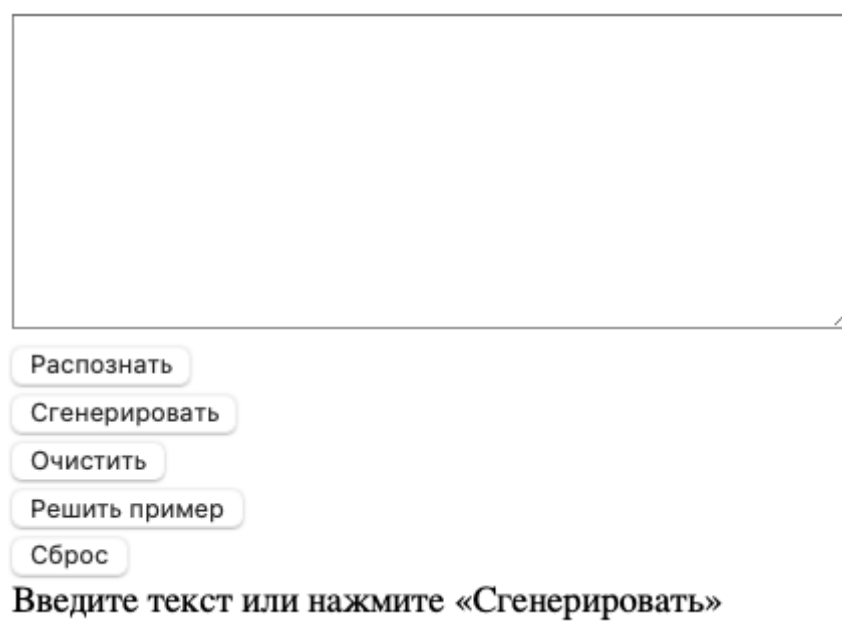


Рис. 10: Ошибка: распознать пустое окно

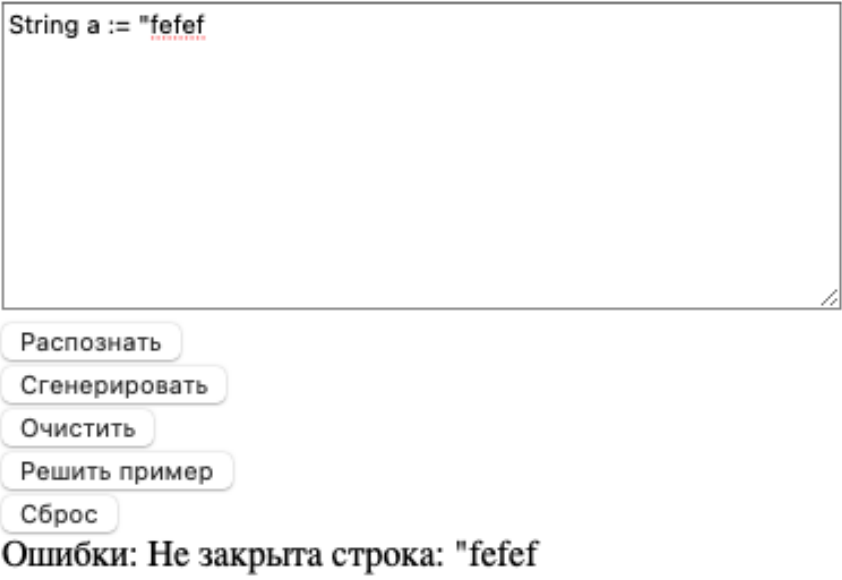


Рис. 11: Ошибка: не закрыта строка

4.3 Особые случаи

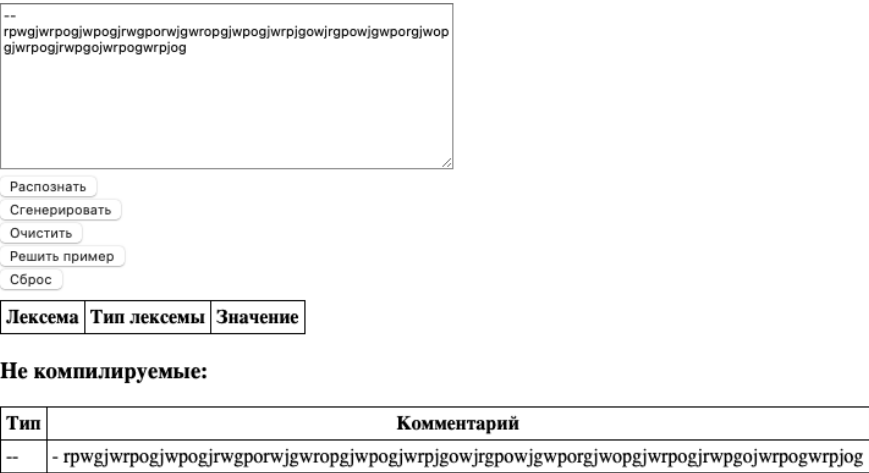


Рис. 12: Разрешен большой комментарий

4.4 Семантика

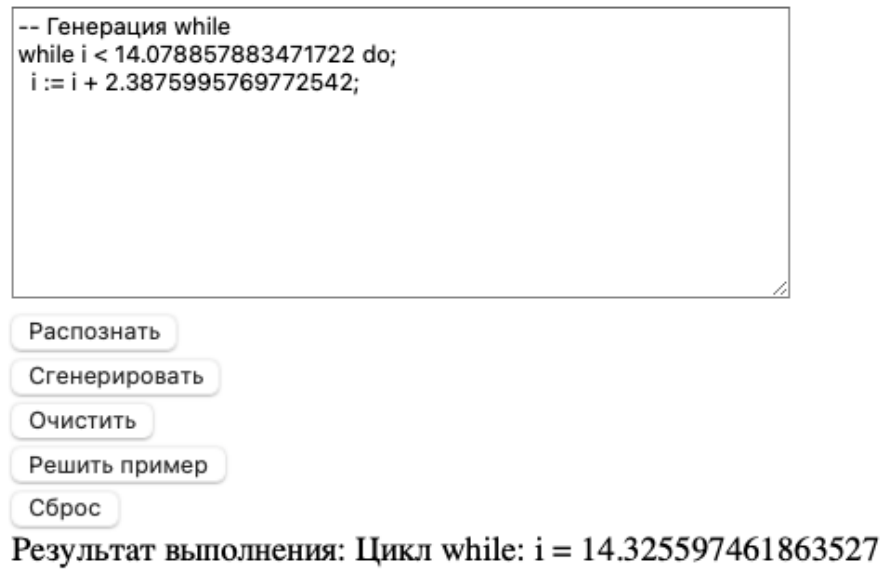


Рис. 13: Семантика цикла

4.5 Выбор текстового файла

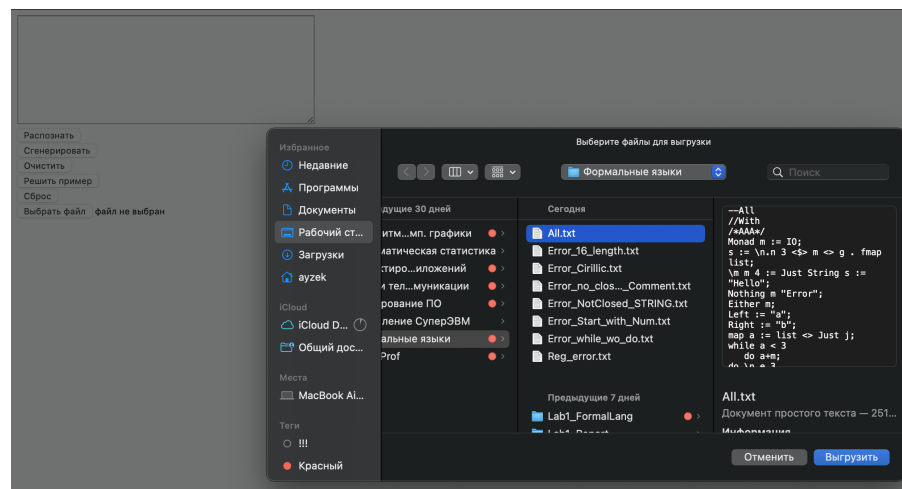


Рис. 14: Выбор текстового файла

Преведены все случаи ошибок и вариантов работы программы.

Заключение

В результате выполнения лабораторной работы №1, был реализован синтаксический (лексический) анализатор, распознающий ключевые слова, вещественные числа, идентификаторы, операторы, функторы, разделители и строки. Лексический анализатор, основан на логике грамматик Хомского, с подстановкой токенов как терминалов и нетерминалов. Так же приведена синтаксическая диаграмма программы.

Список литературы

1. Востров, А. В. Математическая логика [Электронный ресурс]. Режим доступа: <https://tema.spbstu.ru/compiler/> (последний визит: 18.03.2025).
2. Сети, Р.; Ахо, А. Компиляторы: принципы, технологии и инструменты / Р. Сети, А. Ахо. – М.: Издательство «Наука», 2006. – С. 104.