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.

Санкт-Петербург – 2025

Содержание

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 пораждает 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. в интегрированной среде разработки visual studio code.

Указанный вариант - №14.
Правила:

- Входной текст содержит операторы цикла while ... do и do ... while
- Разделитель символом ;
- Операторы условия содержат знаки сравнения =, >, <
- Вещественные числа
- Знак присваевания :=
- Вещественные числа могут начинатся 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 компелируемые 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 наличие 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 $T = (t_1, t_2, \ldots, t_n)$.

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

,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):

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

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

Тип 2 (Контекстно-свободные грамматики):

$A \to \gamma$, где A - нетерминал, γ - последовательность терминалов и нетерминалов.

Тип 1 (КЗ грамматики):

$$\alpha A \beta \to \alpha \gamma \beta$$

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.

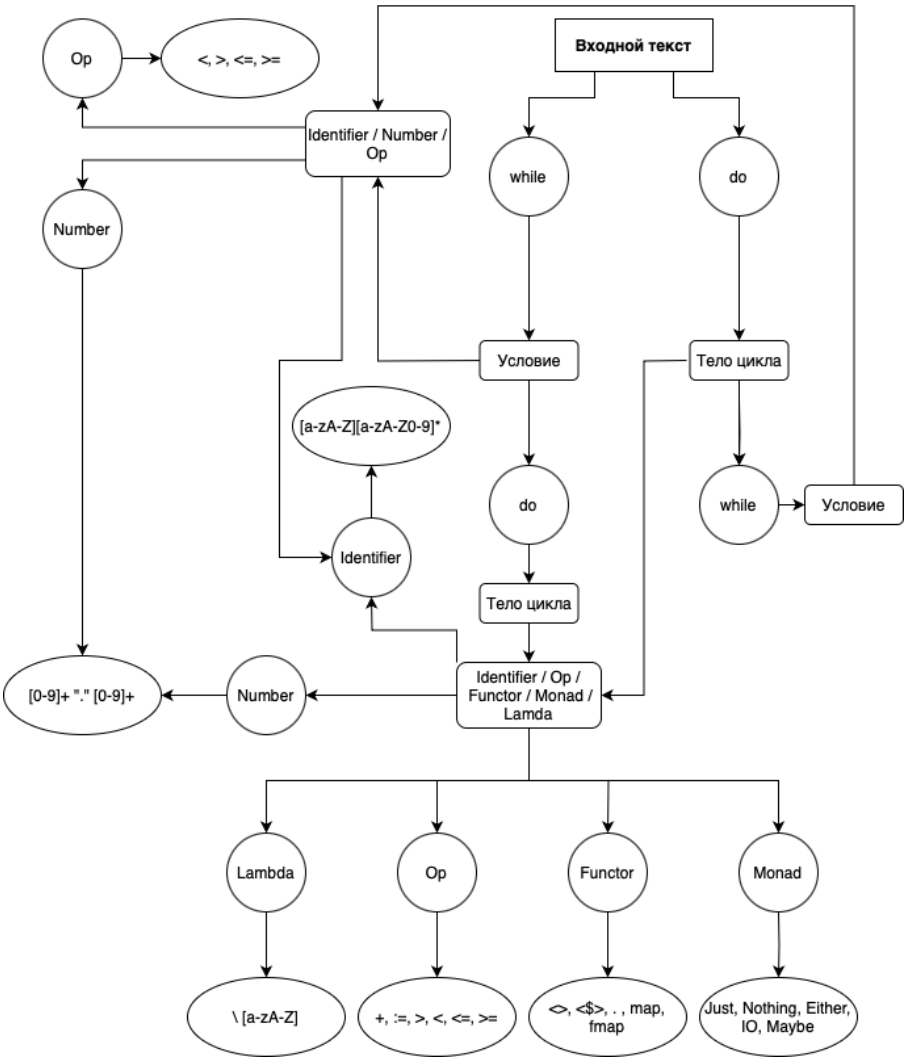

Рис. 1: Синтаксическая диаграмма

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:

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

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):

```haskell
data Token
    = TWhile
    | TDo
    | TKeyword String
    | TIdentifier String
    | TReal Double
    | TAssign
    | TOp String
    | TSemicolon
    | TString String
    | TComment String
    | TLambda String
    deriving (Show, Eq)
```

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

```haskell
tokenLexeme :: Token -> String
tokenLexeme t = case t of
  TWhile        -> "while"
  TDo           -> "do"
  TKeyword s    -> s
  TIdentifier s -> s
  TReal d       -> show d
  TAssign       -> ":="
  TOp op        -> op
  TSemicolon    -> ";"
  TString s     -> s
  TComment s    -> s
  TLambda s     -> s
```

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

```
lexer :: String -> ([Token], [String])
lexer input =
    let (ts, es) = lexInternal input
    in if not (null es)
        then (ts, es)
        else
          let e2 = checkNonLatinForNonComment ts
          in (ts, e2)
```

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

```
lexInternal :: String -> ([Token],[String])
lexInternal [] = ([], [])
lexInternal (c:cs)
    | isSpace c = lexInternal cs
    | c=='/' && take 1 cs == "/" = ...
    | c=='-' && take 1 cs == "-" = ...
    | isAlpha c = lexIdentOrKeyword (c:cs)
    | isDigit c || c=='.' = lexNumber (c:cs)
    | otherwise =
        let (toks, errs) = lexInternal cs
        in ([], ("Unexpected character: " ++ [c]) : errs)
```

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

```
executeProgram :: String -> Either String String
executeProgram s =
    let (ts, es) = lexer s
    in if not (null es)
        then Left (unlines es)
        else
          let noComm = filter (not . isComment) ts
          in case noComm of
              [ TWhile, TIdentifier i1, TOp "<", TReal lim, TDo, TSemicolon
              , TIdentifier i2, TAssign, TIdentifier i3, TOp "+", TReal st
              , TSemicolon]
                  | i1==i2 && i2==i3 -> execWhileLoop 0 lim st 0

              [ TDo, TIdentifier i1, TAssign, TIdentifier i2, TOp "+", TReal
                  st
              , TSemicolon, TWhile, TIdentifier i3, TOp "<", TReal lim,
                 TSemicolon]
                  | i1==i2 && i2==i3 -> execDoWhileLoop 0 lim st 0

              _ -> Left "Unsupported program format"
```

3.4 Main.hs

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

```
setup window = do
  return window # set title "Haskell UI"
  on UI.click recognizeButton $ \_ -> do
    txt <- get value inputBox
    if all isSpace txt
      then element outputDiv # set text "Enter text or press \"Generate\""
      else do
        let (tokens, errors) = lexer txt
        if not (null errors)
          then element outputDiv # set text ("Errors:\n" ++ unlines errors)
          else do
            let assigns = buildAssignMap tokens
            let (comments, nonComments) = separateComments tokens
            let (rows, _) = foldl
                                (\(acc,counter) tk ->
                                    let (row3, newC) = tokenToRow assigns tk
                                       counter
                                    in (acc ++ [row3], newC))
                                ([], 1)
                                nonComments

            tableMain <- buildMainTable rows
            tableComm <- buildCommentTable comments
            element outputDiv # set children [tableMain, tableComm]
```

Для предотвращения зацикливания была создана функция сброса:

```
on UI.click resetButton $ \_ -> do
  element inputBox  # set value ""
  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
// HASKEEEEEL
```

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
//	/ HASKEEEEEL

Рис. 3: Расширенный вариант

4.2 Исключения и ошибки анализа

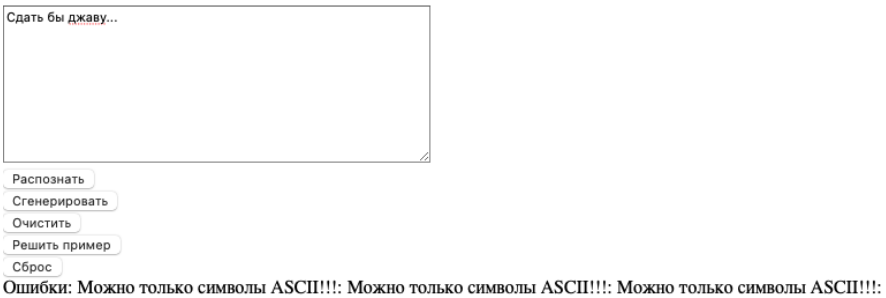

Рис. 4: Ошибка ввода кириллицы

Рис. 5: Ошибка: не закрыт блочный комментарий

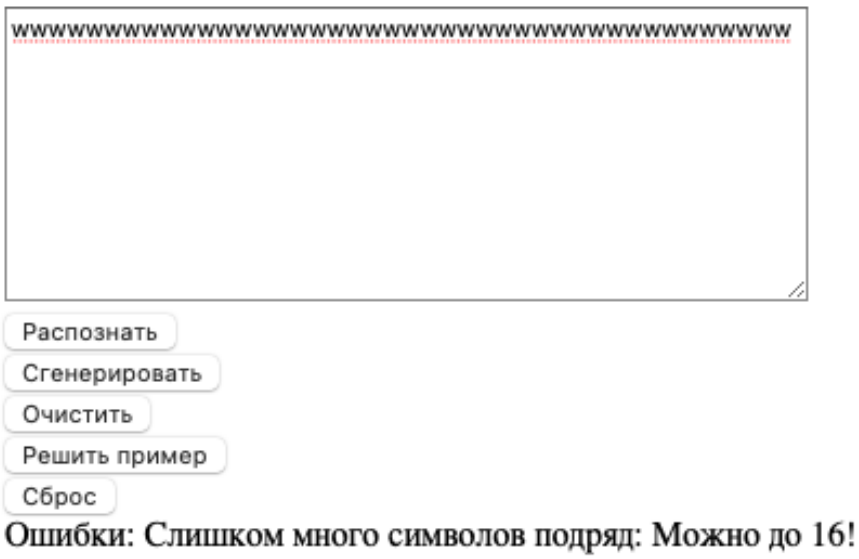

Рис. 6: Ошибка: длина лексемы больше 16

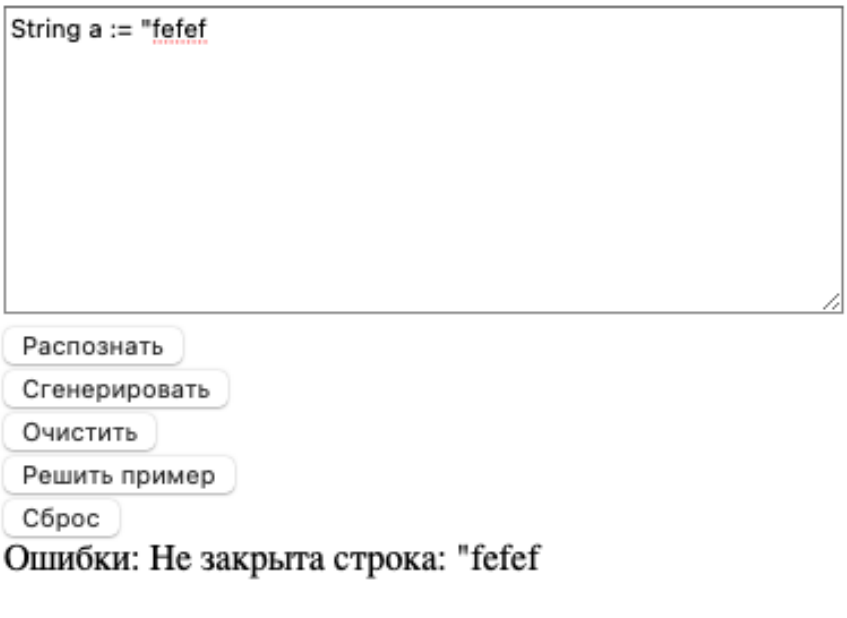

Рис. 7: Ошибка: не закрыта строка

Рис. 8: Ошибка: неверное начало индентификатора

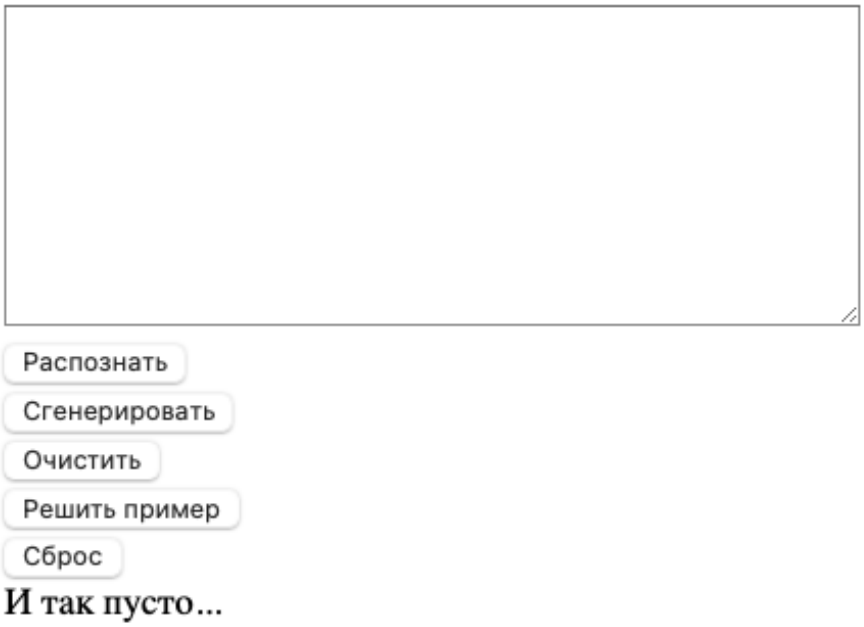

Рис. 9: Ошибка: попытка очистить пустое окно

Рис. 10: Ошибка: респознать пустое окно

Рис. 11: Ошибка: не закрыта строка

4.3 Особые случаи

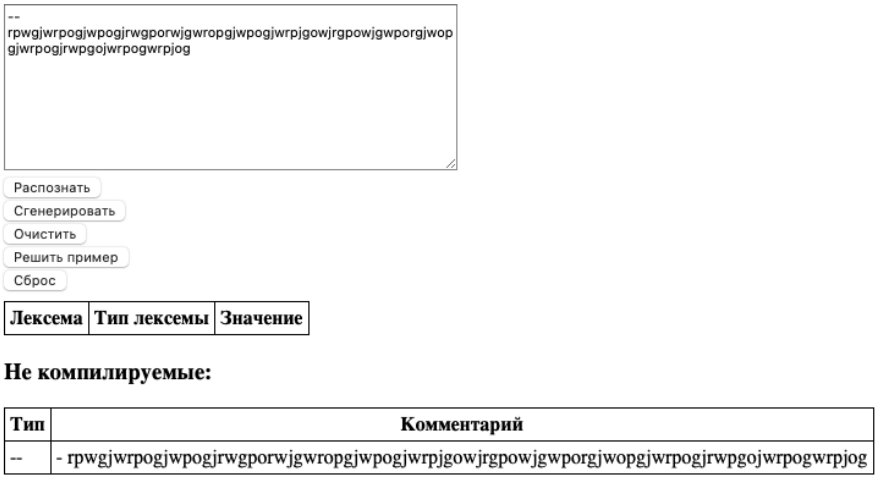

Рис. 12: Разрешен большой комментарий

4.4 Семантика

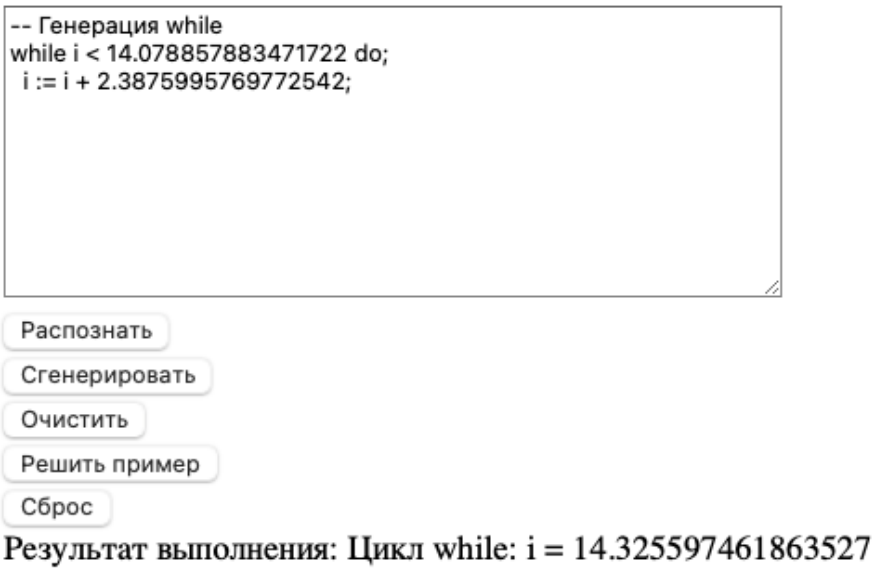

Рис. 13: Семантика цикла

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 приведина синтаксическая диаграмма программы.

Список литературы

1. Востров, А. В. Математическая логика [Электронный ресурс]. Режим доступа: https://tema.spbstu (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.