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 ИТР

Кафедра ПИн

Лабораторная работа

По Лексический анализ

Тема Теория автоматов и формальных языков.

Руководитель:

Кульков Я.Ю.
(фамилия,инициалы)

(подпись) (дата)

Студент ПИнз - 120
(группа)

Чернышев А.Е.
(фамилия, инициалы)

(подпись) (дата)

Муром 2023

Лабораторная работа 1

Цель работы: Ознакомление с назначением и принципами работы лексических анализаторов, получение практических навыков построения сканера на примере заданного входного языка.

Задание:

1. Написать программу, которая выполняет лексический анализ входного текста, подготовленного в соответствии с заданием и порождает список токенов. Текст на входном языке задаётся в виде символьного (текстового) файла, либо читается из текстового поля на форме.

2. Программа должна выдавать сообщения о наличие во входном тексте ошибок, которые могут быть обнаружены на этапе лексического анализа. Длину идентификаторов ограничить 8 символами.

3. Выводить на форму лексемы с указанием типа каждой из них.

4. Составить тестовые наборы данных и проверить на них работу программы.

Ход работы:

```
enum TokenType {
  var = 'var',
  integer = 'integer',
  real = 'real',
  string = 'string',
  begin = 'begin',
  end = 'end',
  repeat = 'repeat',
  until = 'until',
  comma = ',',
  dot = '.',
  colon = ':',
  semicolon = ';',
  gt = '>',
  lt = '<',
  equals = '=',
  lpar = '(',
  rpar = ')',
  plus = '+',
  minus = '-',
  multy = '*',
  delimiter = '/',
  id = 'id',
  literal = 'literal',
```

```
}

export class Token {
  public id: string
  constructor(public type: TokenType, public value: string = '') {
    this.id = new Date() + value
  }
  toString() {
    return `${this.type}, ${this.value}`
  }
}

const Delimiters: TokenType[] = [
  TokenType.comma,
  TokenType.dot,
  TokenType.semicolon,
  TokenType.gt,
  TokenType.lt,
  TokenType.equals,
  TokenType.lpar,
  TokenType.rpar,
  TokenType.plus,
  TokenType.minus,
  TokenType.minus,
  TokenType.delimiter,
]

const SpecialWords: Record<string, TokenType> = {
  var: TokenType.var,
  integer: TokenType.integer,
  real: TokenType.real,
  string: TokenType.string,
  begin: TokenType.begin,
  end: TokenType.end,
  repeat: TokenType.repeat,
  untill: TokenType.until,
}
const SpectialSymbols: Record<string, TokenType> = {
  ',': TokenType.comma,
  ':': TokenType.colon,
  '.': TokenType.dot,
  ';': TokenType.semicolon,
  '>': TokenType.gt,
  '<': TokenType.lt,
  '=': TokenType.equals,
  '(': TokenType.lpar,
  ')': TokenType.rpar,
  '+': TokenType.plus,
  '-': TokenType.minus,
  '*': TokenType.multy,
```

```
  / : TokenType.delimiter;
}

class EndException extends Error {
  /**
   *
   */
  constructor() {
    super()
  }
}
class UndefinedSymbolException extends Error {
  constructor(m: string, buffer: string) {
    super(m + ': ' + buffer)
  }
}
class LexicalException extends Error {
  constructor(m: string, buffer: string) {
    super(m + ': ' + buffer)
  }
}

export class LexicalParser {
  constructor(
    private symbols: string[] = [],
    private buffer: string = '',
    private active: string | undefined = '',
    public table: Token[] = [],
    public errors: string[] = []
  ) {}
  init(input: string) {
    this.symbols = input.split('')
    try {
      this.next()
      this.add()
      this.s()
    } catch (err) {
      const error = err as Error
      const ctor = Object.getPrototypeOf(err).constructor.name
      if (ctor === UndefinedSymbolException.name) {
        return this.errors.push(error.message)
      }
      if (ctor === LexicalException.name) {
        return this.errors.push(error.message)
      }
      if (ctor === EndException.name) {
        return
      }
      throw err
    }
```

```
}
reset() {
  this.symbols = []
  this.buffer = ''
  this.active = ''
  this.table = []
}

add() {
  const v = this.symbols.shift()
  if (v === undefined) {
    if (this.buffer.length > 0) this.out()
    throw new EndException()
  } else {
    this.active = '' + v
  }
}
next() {
  this.buffer += this.active
}
out() {
  if (this.isSpecialWord(this.buffer)) {
    return this.table.push(new Token(SpecialWords[this.buffer]))
  }
  if (this.isSpecialSymbol(this.buffer)) {
    return this.table.push(new Token(SpectialSymbols[this.buffer]))
  }

  if (/^[a-zA-Z_][a-zA-Z0-9_]*/gm.test(this.buffer)) {
    return this.table.push(new Token(TokenType.id, this.buffer))
  }
  if (/[0-9]*/.test(this.buffer)) {
    return this.table.push(new Token(TokenType.literal, this.buffer))
  }
}
clear() {
  this.buffer = ''
}

s(): any {
  this.log('state s')
  if (this.active === undefined) return
  if (/[a-zA-Z_]/.test(this.active)) {
    return this.i()
  }
  if (/[0-9]/.test(this.active)) {
    return this.d()
  }
  if (this.active === ' ' || this.active === '\n') {
    this.clear()
```

```
    this.add()
    return this.s()
  }
  if (/:|,|\.|;|>|<|=|\(|\)|\+|-|\*|\//gm.test(this.active)) {
    return this.r()
  }
  throw new UndefinedSymbolException('Неккоректный символ', this.active)
}
end() {
  return
}

i(): any {
  this.next()
  this.add()
  this.log('State I')
  if (/[a-zA-Z_]/.test(this.active as string)) {
    return this.i()
  }
  if (/[0-9]/.test(this.active as string)) {
    return this.i()
  }
  if (!/:|,|\.|;|>|<|=|\(|\)|\+|-|\*|\/|\s/gm.test(this.active as string)) {
    throw new LexicalException(
      'Неккоректное значение',
      (this.buffer + this.active) as string
    )
  }
  this.out()
  this.clear()
  this.s()
}

d(): any {
  this.next()
  this.add()
  this.log('State D')
  if (/[0-9]/.test(this.active as string)) {
    return this.d()
  }
  if (/[a-zA-Z]/.test(this.active as string)) {
    throw new LexicalException(
      'Неккоректное описание ',
      this.buffer + this.active
    )
  }
  this.out()
  this.clear()
  this.s()
}
```

```
  r() {
    this.next()
    this.add()
    this.log('State R')
    this.out()
    this.clear()
    this.s()
  }

  log(name = '') {
    // console.log(name, {
    //   buffer: this.buffer,
    //   activeSymbol: this.active,
    //   table: [...this.table],
    //   symbols: this.symbols,
    // })
  }

  isDelimiter(token: Token): boolean {
    return Delimiters.includes(token.type)
  }
  isSpecialWord(word: string): boolean {
    if (word === '') return false
    return Object.keys(SpecialWords).includes(word)
  }
  isSpecialSymbol(word: string): boolean {
    return Object.keys(SpectialSymbols).includes(word)
  }
}
```

Вывод: В ходе лабораторной работы мы ознакомились с назначением и принципами работы лексических анализаторов, получение практических навыков построения

сканера на примере заданного входного языка.