­­ИНСТИТУТ ТРАНСПОРТА И СВЯЗИ



ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ НАУК И ТЕЛЕКОММУНИКАЦИЙ

Лабораторная работа №1

По курсу «Проектирование компиляторов»

Тема «Программирование лексического анализатора»

Выполнили:

Михаил Красильников

Группа 4701BV

Проверил Ирина Птицина

Рига, 2020

**Содержание**

[Задание на лабораторную работу](#_2et92p0) **3**

[Индивидуальное задание](#_tyjcwt) **4**

[Метод исключения Гаусса с ведущим элементом](#_3dy6vkm) **4**

[Метод Краута-Холецки](#_1t3h5sf) **5**

[Число обусловленности матрицы](#_4d34og8) **6**

[Выводы](#_2s8eyo1) **6**

# Краткое теоретическое обоснование

Лексический анализатор – это компонент, который читает искходный текст программы на лексемы, всего существуют несколько типов лексем, а именно:

* Индетификаторы
* Ключевые слова
* Специальные символы
* Константы

Для построения лексического анализатора требуется построить грамматику языка, грамматику языка можно построить при помощи формы Бэкуса наура или BNF форма

BNF форма состоит ил терминалов и нетерминалов.

терминал состоит из конечного множестве символов из которого может быть построенны более сложные конструкции языка к примеру слово или формула

Терминал – описывает сущность языка(слово или формул

Произвести анализ заданного на языке PASCAL программного фрагмента (индивидуальные варианты фрагментов - [здесь](https://e.tsi.lv/pluginfile.php/81226/mod_assign/intro/Compilers_LW1_IndTasks_BV.pdf), номер индивидуального задания выбирается в соответствии со своим номером по [списку групп](https://e.tsi.lv/pluginfile.php/81226/mod_assign/intro/4601BV_autumn_moodle.pdf)) и выделить все типы имеющихся в нем лексем: ключевые слова, специальные символы (однопозиционные, двухпозиционные разделители), литералы, идентификаторы.

# Для выделенных типов лексем построить регулярную грамматику. Если в заданном программном фрагменте есть комментарии и текстовые (символьные, строковые) литералы, то грамматические правила для них строить не нужно, однако их распознавание должно осуществляться далее конечным автоматом.

# Для полученной грамматики построить диаграмму состояний соответствующего конечного автомата, который сможет распознавать все лексемы всех выделенных типов для заданного программного фрагмента.

# На основе полученного в пунктах 1-3 написать программу лекси­ческого анализатора (сканер), который:

# принимает заданный программный фрагмент, таблицу ключевых слов и таблицу специальных символов (эти две таблицы могут быть объединены в одну таблицу т.н. служебных символов, которые могут задаваться в тексте программы сканера, например, в виде массива или хранится в файле или фиксироваться как-то иначе)

# возможно формирует отдельную таблицу распознанных литералов и таблицу распознанных идентификаторов;

# формирует таблицу т.н. стандартных символов (все распознанные лексемы в порядке их появления в заданном программном фрагменте и их тип, а также, возможно, индекс в таблице соответствующего типа лексем, если таковые формировались);

# сообщать об успешном или неуспешном результате лексического анализа (для этого необходимо внести изменения в исходный фрагмент и проверить, что сканер по-прежнему распознает все лексемы, если лексических ошибок нет, и обнаруживает лексические ошибки, если они есть).

# Подготовить отчет о проделанной работе.

# Текст заданного программного фрагмента индивидуального задания

|  |
| --- |
| while Tmp<=nil do  Begin  if Tmp.Num = StrToInt(S) then i := i+1;  Begin  i := i+1;  Summ := Summ + Tmp^.Mark  tmp:=Tmp^.Next  End;  j:= Summ/i;  Edit5.Text := FloatToStr(j)  End; |

# Грамматические правила для имеющихся в заданном программном фрагменте лексем.

|  |
| --- |
| <identifier> ::= <letter> | < identifier> <letter> | < identifier > <number>  <integer-literal> ::= <number>  <two\_delim> :: = <less> **>** | <colon> **=**  <one\_delim> :: = **=** | **+** | **;** | **.** | **^** | **/** | **(** | **)**  <less>::= <  <colon> ::= :  <letter> ::= 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  <keywords> ::= while | nil |do | if | then |end  <number> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

# 

# 

# Graphical user interface, application, Teams Description automatically generatedДиаграмма состояний соответствующего конечного авто­мата.

# Исходные таблицы лексем (ключевые слова и специальные символы).

**Рисунок 1 Диаграмма состояний**

# Текст программы сканера.

|  |
| --- |
| <?php declare(strict\_types=1);  namespace Krasilnikovm\Compiler;  */\*\*  \* Class Scanner  \*  \* @author Mihail Krasilnikov <mihail.krasilnikov.j@gmail.com>  \*/* final class Scanner {  private const *KEYWORDS* = [  'while',  'nil',  'do',  'if',  'then',  'end',  'begin',  ];   private const *ONE\_DELIMITER* = [  '=',  '+',  ';',  '.',  '^',  '/',  '(',  ')',  ];   private const *TWO\_DELIMITER* = [  '<>',  ':=',  ];   private FileReader $fileReader;  private string $word = '';  private ?string $char = '';   */\*\*  \* Scanner constructor.  \* @param FileReader $fileReader  \*/* public function \_\_construct(FileReader $fileReader)  {  $this->fileReader = $fileReader;  }   public function scan(): \Generator  {  $this->char = $this->fileReader->getChar();   while ($this->char !== null) {  $this->word = '';   if (*ctype\_space*($this->char) || $this->char === '\n') {  $this->char = $this->fileReader->getChar();  continue;  }   switch (true) {  case *ctype\_alpha*($this->char):  yield $this->handleIdentifierState();  break;  case *ctype\_digit*($this->char):  yield $this->handleLiteralState();  break;  case *in\_array*($this->char, [':', '<'], true):  yield $this->handleTwoDelimiterState();  break;  case *in\_array*($this->char, self::*ONE\_DELIMITER*, true):  yield $this->handleOneDelimiterState();  break;  default:  $position = $this->fileReader->getCurrentPosition() - *strlen*($this->word);  yield [$this->char, Constants::*ERROR\_TYPE*, "Unexpected lexem found on line {$this->fileReader->getCurrentLine()} and on position $position"];  $this->char = $this->fileReader->getChar();  }  }   $this->fileReader->closeFile();  }   private function handleIdentifierState(): array  {  do {  $this->word .= $this->char;  $this->char = $this->fileReader->getChar();  } while (*ctype\_alpha*($this->char) || *ctype\_digit*($this->char));   $type = *in\_array*(*strtolower*($this->word), self::*KEYWORDS*) ? Constants::*KEYWORD* : Constants::*IDENTIFIER\_TYPE*;   return [$this->word, $type, null];  }   private function handleLiteralState(): array  {  do {  $this->word .= $this->char;  $this->char = $this->fileReader->getChar();  } while (*ctype\_digit*($this->char));   return [$this->word, Constants::*LITERAL\_TYPE*, null];  }   private function handleTwoDelimiterState(): array  {  $this->word .= $this->char;  $this->char = $this->fileReader->getChar();  $this->word .= $this->char;   $type = *in\_array*($this->word, self::*TWO\_DELIMITER*) ? Constants::*DELIMITER* : Constants::*ERROR\_TYPE*;   if ($type === 'error') {  $position = $this->fileReader->getCurrentPosition() - *strlen*($this->word);  $message = "Unexpected lexem found on line {$this->fileReader->getCurrentLine()} and on position $position";  }   $this->char = $this->fileReader->getChar();  return [$this->word, $type, $message ?? null];  }   private function handleOneDelimiterState(): array  {  $this->word .= $this->char;  $this->char = $this->fileReader->getChar();  return [$this->word, Constants::*DELIMITER*, null];  } } |

# Результаты работы сканера.

# Выводы.