МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Тверской государственный технический университет»

(ТвГТУ)

Кафедра “Программного обеспечения”

**Курсовая работа**

по дисциплине “ ТЕОРИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ И МЕТОДЫ ТРАНСЛЯЦИИ”

Тема: «Разработка транслятора»

Выполнил: студент группы

ПИН 17.06

Завгороднев Е. Ю

Проверил:

Калабин А.Л

Тверь 2020

Оглавление

[Введение 3](#_Toc53939988)

[Описание синтаксиса реализуемого языка в форме Бэкуса-Наура 3](#_Toc53939989)

[Лексический анализатор 4](#_Toc53939990)

[Список классов лексем реализуемого языка 4](#_Toc53939991)

[Основные процедуры и функции 4](#_Toc53939992)

[Тестовые примеры 4](#_Toc53939993)

[Синтаксический анализатор 5](#_Toc53939994)

[Основные процедуры и функции 5](#_Toc53939995)

[Тестовые примеры 5](#_Toc53939996)

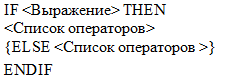
[Генератор кода 7](#_Toc53939997)

[Краткое описание 7](#_Toc53939998)

[Тестовые примеры 7](#_Toc53939999)

# Введение

К транслятору, полученному в ходе лабораторных работ, я добавил следующую конструкцию высокого уровня



# Описание синтаксиса реализуемого языка в форме Бэкуса-Наура

<Программа> ::= <Объявление переменных> <Описание вычислений> <Оператор печати>

<Описание вычислений> ::= **Begin <Список присваиваний><Условная конструкция> End** | Begin <Список присваиваний> End

<Объявление переменных> ::= Var <Список переменных> :Logical

<Список переменных> ::= <Идент> | <Идент> , <Список переменных>

<Список присваиваний>::= <Присваивание> | <Присваивание> <Список присваиваний>

<Присваивание> ::= <Идент> := <Выражение>

**<Условная конструкция> ::= If <Выражение> Then <Присваивание> Else <Присваивание>**

<Выражение> ::= <Ун.оп.><Подвыражение> | <Подвыражение> <Подвыражение> :: = ( <Выражение> ) | <Операнд> |

< Подвыражение > <Бин.оп.> <Подвыражение>

<Ун.оп.> ::= "!"

<Бин.оп.> ::= "&" | "|" | "^"

<Операнд> ::= <Идент> | <Const>

<Идент> ::= <Буква> <Идент> | <Буква>

<Const> ::= 0 | 1

<Оператор печати>::=Print <Идент>

# Лексический анализатор

## Список классов лексем реализуемого языка

* NAME – текст
* TRUE – булево значение (‘1’ в программе)
* FALSE – булево значение (‘0’ в программе)
* NOT – унарный логический оператор (‘!’ в программе)
* AND – логический оператор ('&' в программе)
* XOR– логический оператор (''^' в программе)
* ASSIGN – оператор присваивания ('':=' в программе)
* EQUAL – оператор сравнения (''=' в программе)
* NOT\_EQUAL – оператор сравнения (''!=' в программе)
* SEMI – запятая (',' в программе)
* COLON – двоеточие (':' в программе)
* SPLITTER – разделитель строк ('\n' в программе)

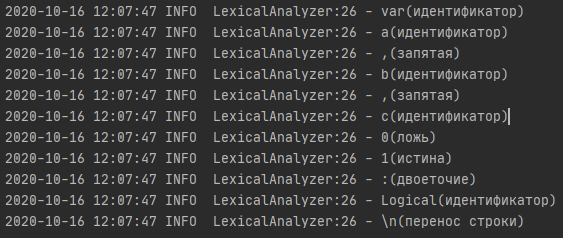
## Основные процедуры и функции

Класс Lexical Analyzer:

* void addKeyword (String word, Lexems lexem) – добавление лексемы в список
* Lexems getIdentifierLexem(String identifier) – получение типа лексемы по строке
* List<Keyword> recognizeAllLexem(String data) – распознавание всех лексем в исходном коде

## Тестовые примеры

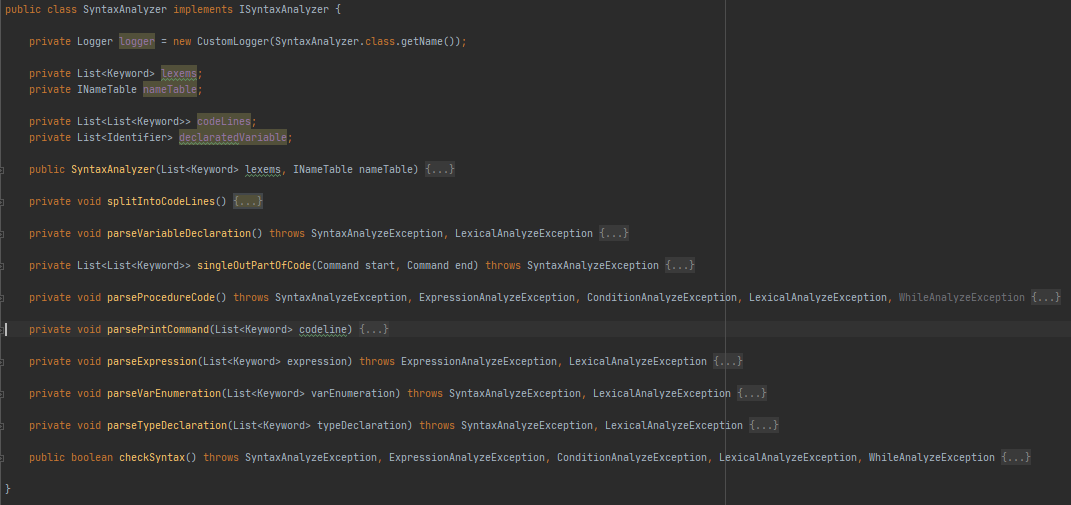
 - входной текст

 - результат

# Синтаксический анализатор

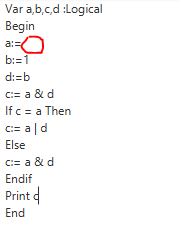
## Основные процедуры и функции

Класс SyntaxAnalyzer:

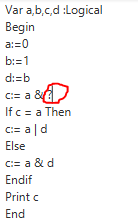


* splitIntoCodeLine() – разбивка всего кода на строки кода
* parseVariableDeclaration() - Разбор объявления переменных
* singleOutPartOfCode() - Разбор секции от BEGIN до END, от IF до ENDIF
* parseProcedureCode() - Разбор основного блока кода
* parsePrintCommand() - Разбор команды PRINT
* parseExpression() - Разбор выражения
* checkSyntax() – Проверка корректности синтаксиса

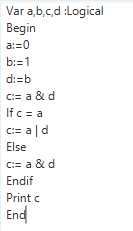
## Тестовые примеры



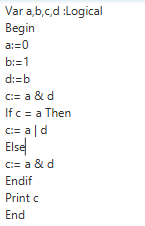














# Генератор кода

## Краткое описание

Генерация кода в данном приложении – процесс создания ассемблерного кода.

Генерация кода состоит из

* Генерации сегмента данных
* Генерация сегментов стека и кода
* Генерация завершение основной процедуры
* Генерация завершения кода
* Генерация объявления переменных
* Генерация кода для логических операций
* Генерация оператора печати
* Генерация условных переходов

## Тестовые примеры

Пример 1 – выражение

|  |  |
| --- | --- |
| Var a,b,c,d :Logical  Begin  a:=0  b:=1  d:=b  c:= a & d  Print c  End | data segment para public "data"  a dw 0b  b dw 0b  c dw 0b  d dw 0b  PRINT\_BUF DB ' ' DUP(10)  BUFEND DB '$'  data ends  stk segment stack  db 256 dup ("?")  stk ends  code segment para public "code"  main proc  assume cs:code,ds:data,ss:stk  mov ax,data  mov ds,ax  mov a,0b  mov b,1b  mov d,1b  mov ax,0b  push ax  mov ax,1b  push ax  pop bx  pop ax  and ax,bx  push ax  mov c, ax  push ax  mov ax, c  CALL PRINT  pop ax  mov ax,4c00h  int 21h  main endp  PRINT PROC NEAR  MOV CX, 10  MOV DI, BUFEND - PRINT\_BUF  PRINT\_LOOP:  MOV DX, 0  DIV CX  ADD DL, '0'  MOV [PRINT\_BUF + DI - 1], DL  DEC DI  CMP AL, 0  JNE PRINT\_LOOP  LEA DX, PRINT\_BUF  ADD DX, DI  MOV AH, 09H  INT 21H  RET  PRINT ENDP  code ends  end main |

Пример 2 – конструкция IF

|  |  |
| --- | --- |
| Var a,b,c,d :Logical  Begin  a:=0  b:=1  d:=b  c:= a & d  If c = a Then  c:= a | d  Endif  Print c  End | …  mov ax,c  mov bx,a  cmp ax,bx  je if  jmp cont  if:  mov ax,0b  push ax  mov ax,1b  push ax  pop bx  pop ax  or ax,bx  push ax  mov c, ax  jmp cont  cont:  push ax  mov ax, c  CALL PRINT  … |

Пример 3 – конструкция IF-ELSE

|  |  |
| --- | --- |
| Var a,b,c,d :Logical  Begin  a:=0  b:=1  d:=b  c:= a & d  If c = a Then  c:= a | d  Else  c:= a & d  Endif  Print c  End | …  mov ax,c  mov bx,a  cmp ax,bx  je if  jne else  if:  mov ax,0b  push ax  mov ax,1b  push ax  pop bx  pop ax  or ax,bx  push ax  mov c, ax  jmp cont  else:  mov ax,0b  push ax  mov ax,1b  push ax  pop bx  pop ax  and ax,bx  push ax  mov c, ax  jmp cont  cont:  push ax  mov ax, c  CALL PRINT  … |