МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Тверской государственный технический университет»

(ТвГТУ)

Кафедра “Программного обеспечения”

**Лабораторная 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

[Цель работы 3](#_Toc53749115)

[БНФ реализуемого языка 3](#_Toc53749116)

[Создание каркаса транслятора 3](#_Toc53749117)

[Разбор исходного файла 4](#_Toc53749118)

[Разработка лексического анализатора 5](#_Toc53749119)

[Список классов лексем реализуемого языка 5](#_Toc53749120)

[Реализация в программе 5](#_Toc53749121)

[Пример работы анализатора 6](#_Toc53749122)

[Таблица имен 6](#_Toc53749123)

[Реализация элемента таблицы 6](#_Toc53749124)

[Общие принципы хранения идентификаторов внутри таблицы 6](#_Toc53749125)

[Реализация методов регистрации идентификатора, получения идентификатора из таблицы по имени 8](#_Toc53749126)

[Синтаксический анализатор 8](#_Toc53749127)

[Общая схема разбора 8](#_Toc53749128)

[Разбор инструкции присваивания 9](#_Toc53749129)

[Схема разбора сложных выражений 10](#_Toc53749130)

[Схема разбора инструкций ветвления 11](#_Toc53749131)

[Генератор кода 13](#_Toc53749132)

[Генерирование общих фрагментов кода, объявления переменных 13](#_Toc53749133)

[Генерация кода для логических операций, работа с ассемблерным стеком 14](#_Toc53749134)

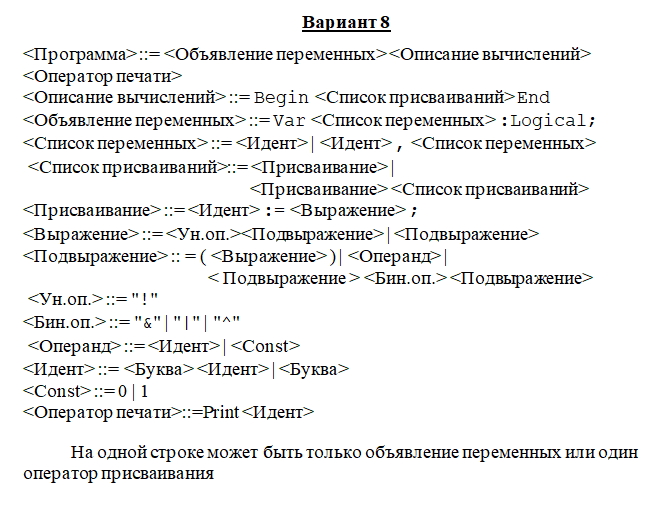
[Разбор и генерация оператора печати 15](#_Toc53749135)

[Общие принципы организации условного перехода в ассемблере. Инструкция *cmp*. Создание меток перехода, генерирование условных переходов для реализации инструкций *while* или *if*. 16](#_Toc53749136)

# Цель работы

Основной целью лабораторных работ по разработке трансляторов является получение практических навыков, позволяющих разрабатывать трансляторы языков программирования.

# БНФ реализуемого языка



# Создание каркаса транслятора

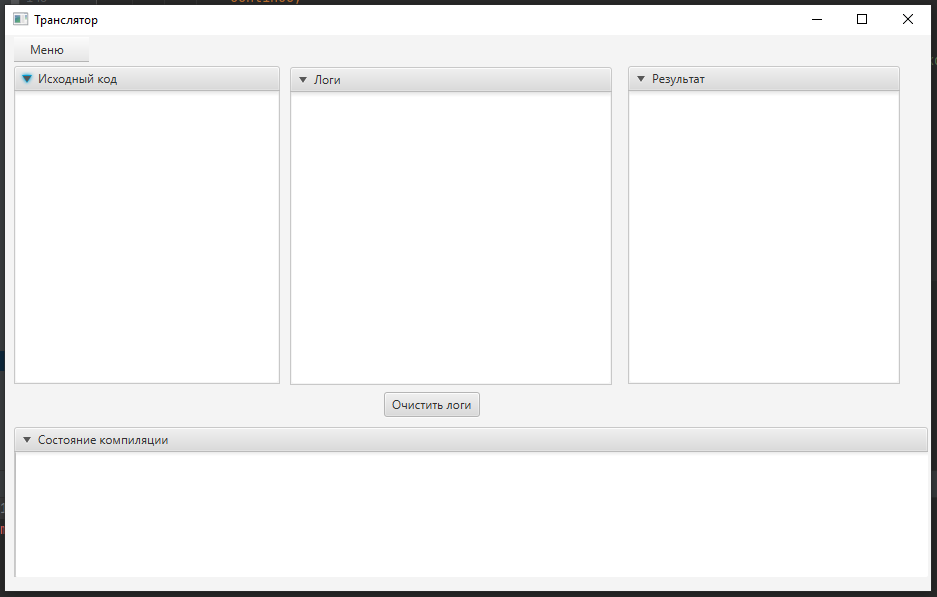


Рис. Вариант интерфейса

Используемые технология:

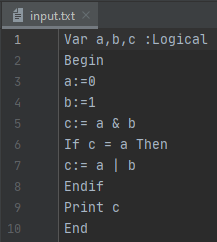
* Java 8
* Java FX

# Разбор исходного файла

Класс чтения из файла



Входной файл



# Разработка лексического анализатора

## 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)

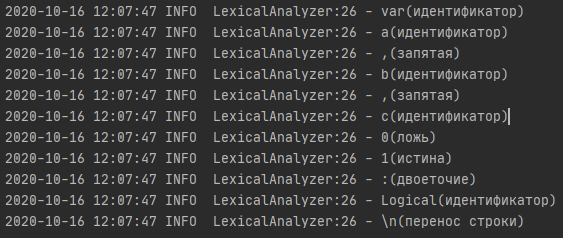
## Реализация в программе

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

## Пример работы анализатора

 - входной текст

 - результат

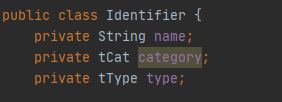
# Таблица имен

## Реализация элемента таблицы

Реализована следующая структура

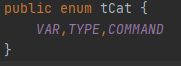
|  |  |  |
| --- | --- | --- |
| Имя | Категория | Тип |
| Строка | Переменная, тип данных, команда | Logical |

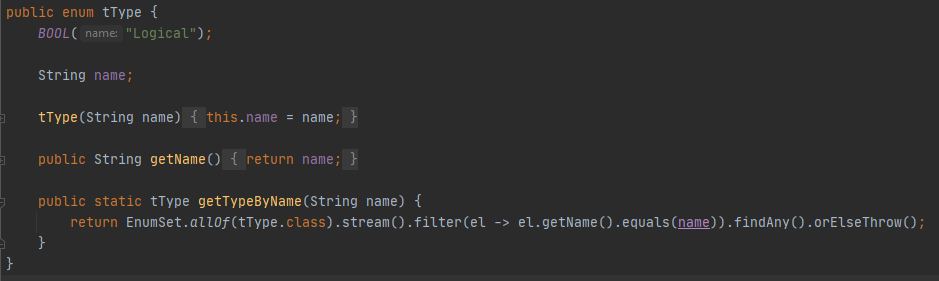
## Общие принципы хранения идентификаторов внутри таблицы



Идентификаторы хранятся в структуре «Идентификатор».

У каждого идентификатора есть имя, категория и тип.





Перечисление tType хранит тип данных исходного языка: bool.

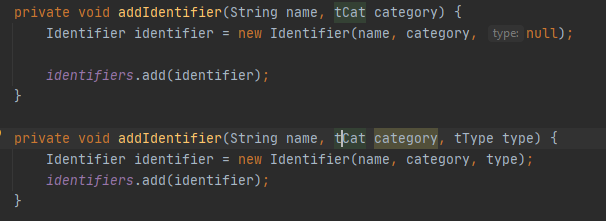
tCat - категории идентификаторов: константа, переменная, тип данных.

Хранение идентификаторов внутри таблицы имен осуществляется в структуре Set, ниже приведена реализация.

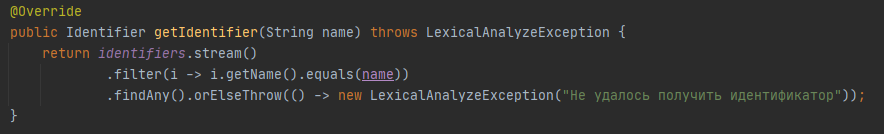


## Реализация методов регистрации идентификатора, получения идентификатора из таблицы по имени

Регистрация идентификатора



Получение идентификатора из таблицы по имени



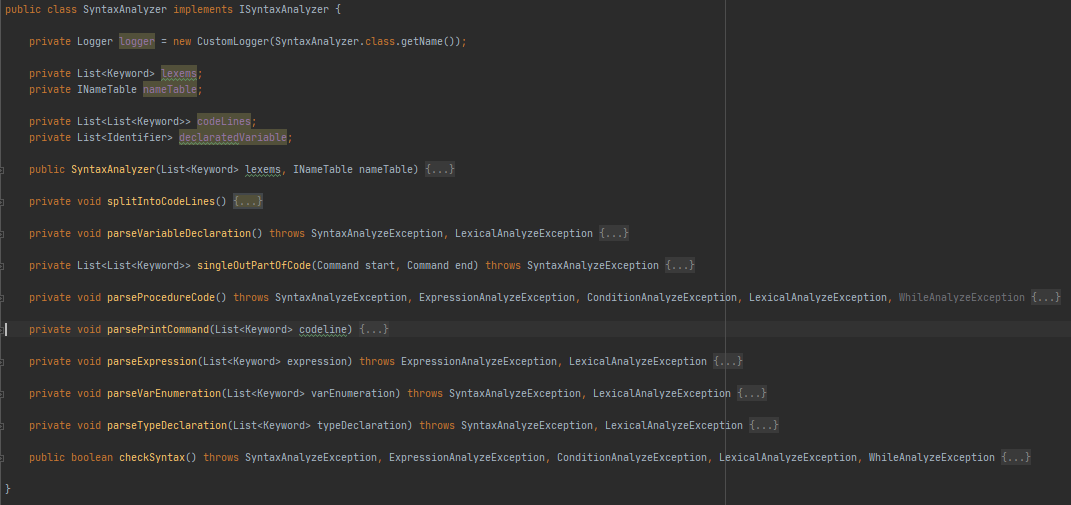
# Синтаксический анализатор

## Общая схема разбора

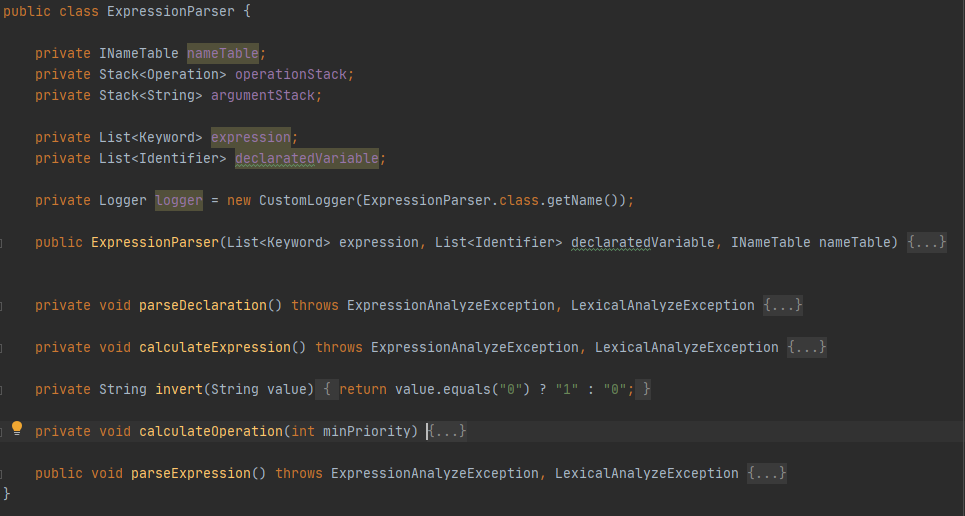
Разбор исходного кода состоит из:

* Разбора объявления переменных
  + Разбор перечисления переменных
  + Разбор объявления типа данных
* Разбора основного блока кода
  + Разбор секции от BEGIN до END
  + Разбор команды PRINT
  + Разбор условия IF
  + Разбор выражения
    - Разбор присвоения значения
    - Разбор присвоения выражения

Класс SyntaxAnalyzer



Класс ExpressionParser

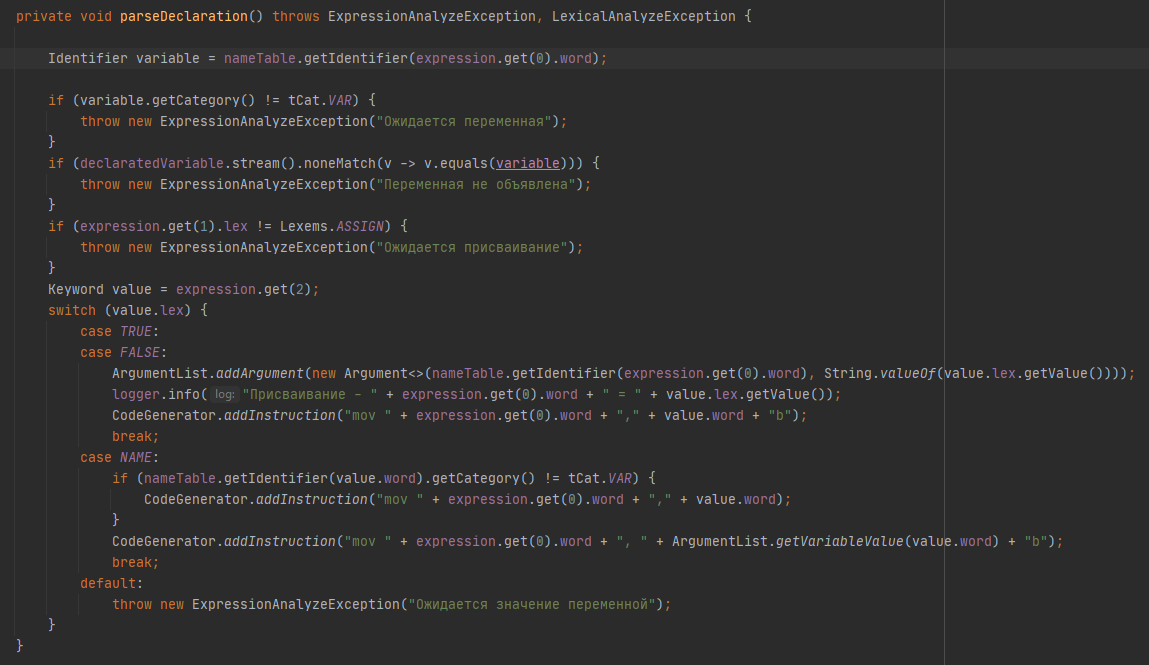


## Разбор инструкции присваивания

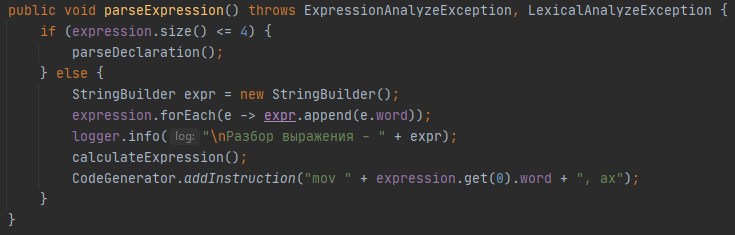
Виды присваивания:

* Присваивание значения (пример, a = 0)
* Присваивание выражения (пример, a = b | c)

Метод parseDeclaration(), разбор присваивания значения



Метод parseExpression(), разбор присваивания выражения



## Схема разбора сложных выражений

Схема разбора состоит из:

* стека аргументов содержит булевы значения
* стек операций содержит знак операции (символ) и её приоритет (целое число).

Приоритеты операций:

* у логического ИЛИ и у исключающего ИЛИ — 1;
* у умножения и деления — 2.

Пример

|  |  |  |
| --- | --- | --- |
| Выражение | Стек операций | Стек аргументов |
| a := 0  b := 1  c := a | b & b | & (2)  | (1) | 1  1  0 |

Вычисление

|  |  |  |
| --- | --- | --- |
| Выражение | Стек операций | Стек аргументов |
| 0 | 1 & 1 | & (2)  | (1) | 1  1  0 |

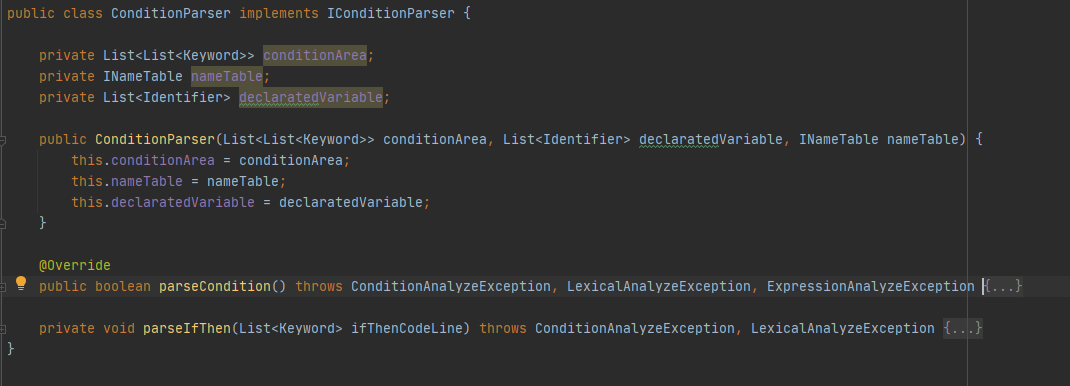
Вычисляем 1 & 1 так как у следующей операции меньший приоритет

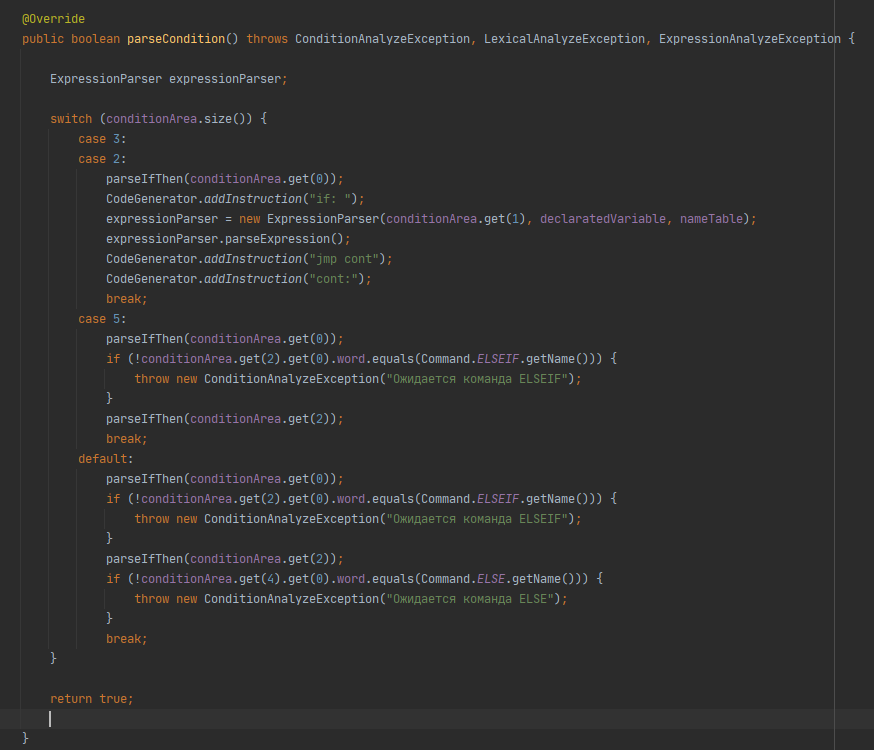
1 & 1 = 1, кладем новое значение в стек

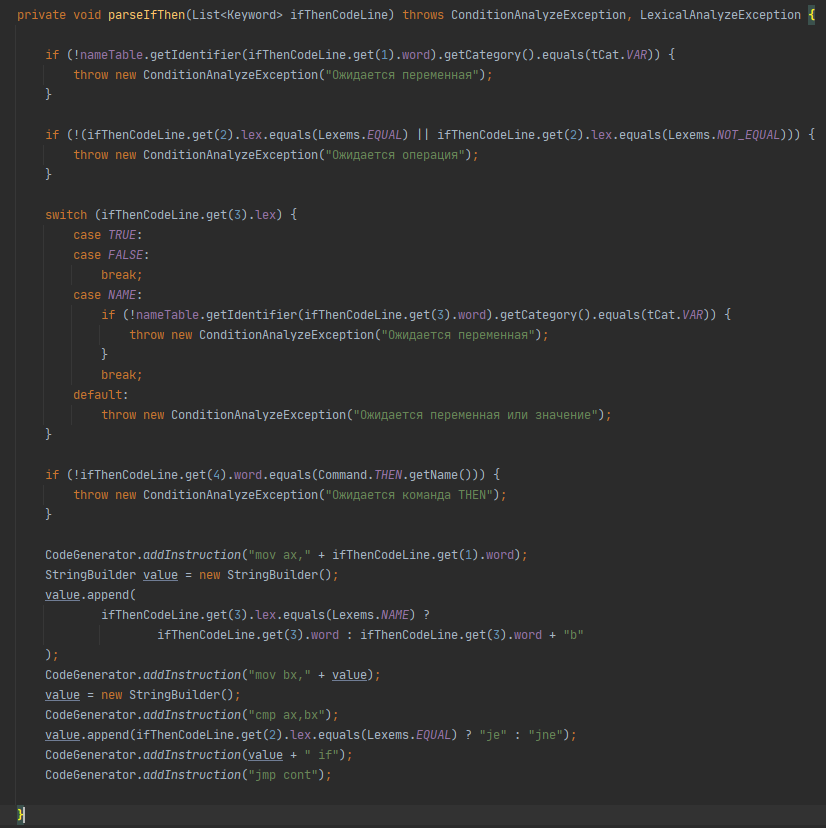
|  |  |  |
| --- | --- | --- |
| Выражение | Стек операций | Стек аргументов |
| 0 | 1 & 1 | | (1) | 1  0 |

Вычисляем 1 | 0, результат **1**

## Схема разбора инструкций ветвления







# Генератор кода

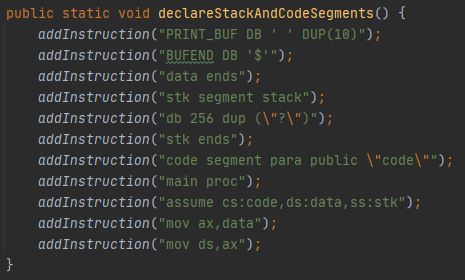
## Генерирование общих фрагментов кода, объявления переменных

Генерация общих фрагментов состоит из

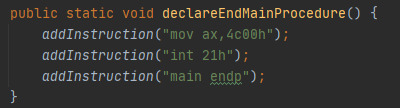
* Генерации сегмента данных



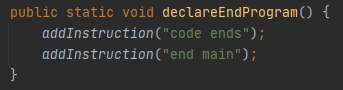
* Генерация сегментов стека и кода



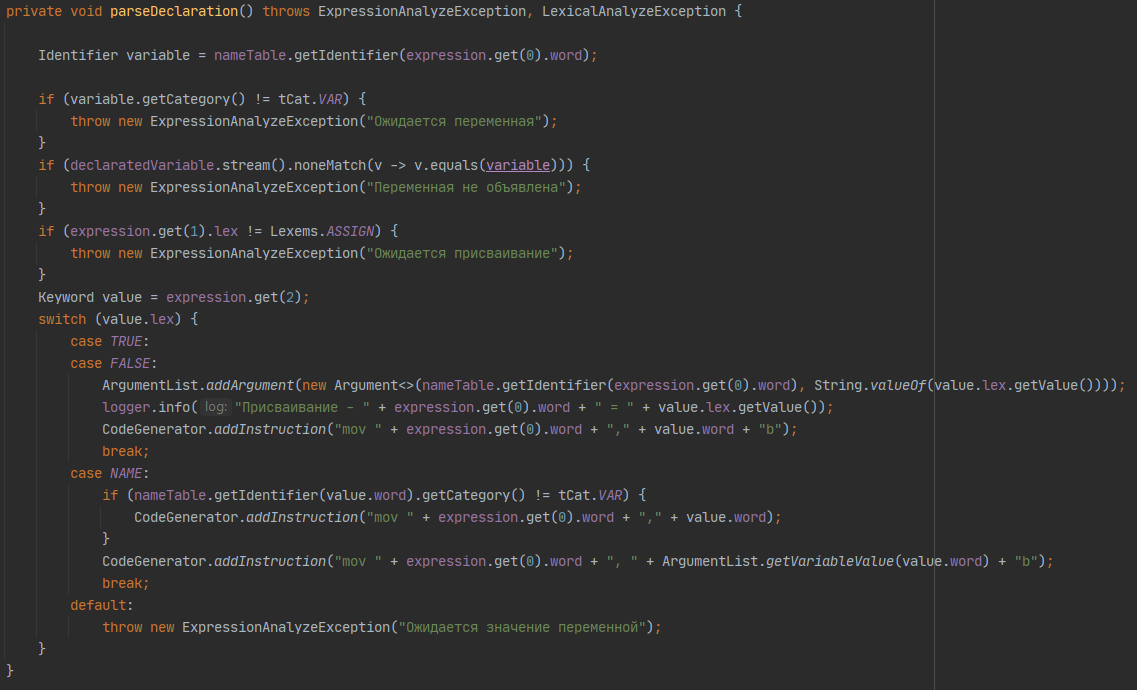
* Генерация завершение основной процедуры



* Генерация завершения кода



* Генерация объявления переменных

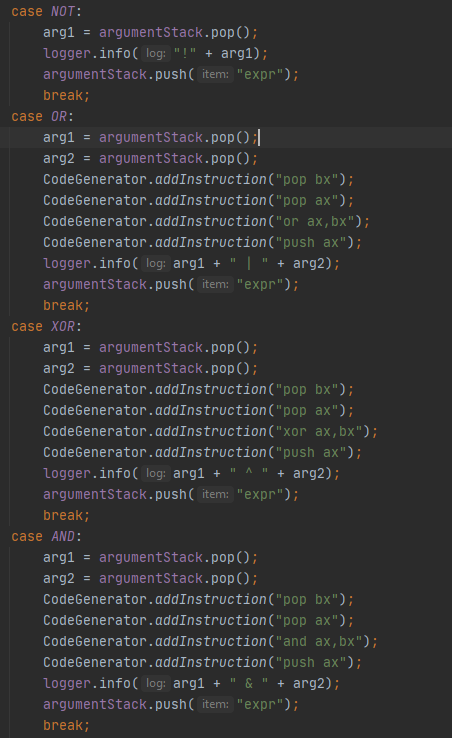


## Генерация кода для логических операций, работа с ассемблерным стеком

|  |  |
| --- | --- |
| Операция | Инструкции ассемблера |
| & | pop bx  pop ax  and ax, bx  push ax |
| | | pop bx  pop ax  or ax, bx  push ax |
| ^ | pop bx  pop ax  xor ax, bx  push ax |

Порядок работы с ассемблерным стеком

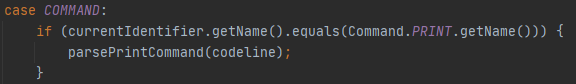
1. Из стека достаются и удаляются два значения
2. Совершается операция
3. В стек кладется новое, вычисленное значение

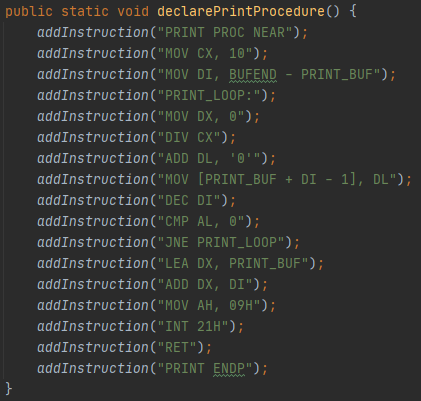


## Разбор и генерация оператора печати

При разборе команды PRINT происходит следующее:

1. В регистр ax заносится переменная, которая должна вывестись на печать
2. Исполняется процедура генерации печати значения регистра ax





## Общие принципы организации условного перехода в ассемблере. Инструкция *cmp*. Создание меток перехода, генерирование условных переходов для реализации инструкций *while* или *if*.

|  |  |  |  |
| --- | --- | --- | --- |
| Инструкция | Пояснение | Код на «нашем» языке | Код на ассемблере |
| je | jump equal | a == b | mov ax, a  mov bx, b  cmp ax, bx  jne label |
| jne | jump not equal | a != b | mov ax, a  mov bx, b  cmp ax, bx  je label |

Пример

|  |  |
| --- | --- |
| Исходный код | Инструкции ассемблера |
| If c = a Then  c:= a | b  Endif | cmp ax,bx **– сравниваем c и a**  je if **– если равны идем на метку if**  jmp cont **– если не равны идем на метку cont**  if: **- метка 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 **– метка cont**  cont: |