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

Иванов Р.В

Проверил:

Калабин А.Л

Тверь 2020

Оглавление

[Введение 3](#_Toc56294504)

[Цель работы 3](#_Toc56294505)

[Используемые технологии 3](#_Toc56294506)

[БНФ реализуемый язык 4](#_Toc56294507)

[Создание транслятора 5](#_Toc56294508)

[Разработка пользовательского интерфейс 5](#_Toc56294509)

[Разбор исходного файла 5](#_Toc56294510)

[Разработка лексического анализатора 6](#_Toc56294511)

[Список классов лексем реализуемого языка 6](#_Toc56294512)

[Реализация в программе 7](#_Toc56294513)

[Пример работы анализатора 7](#_Toc56294514)

[Таблица имен 7](#_Toc56294515)

[Реализация элемента таблицы 7](#_Toc56294516)

[Общие принципы хранения идентификаторов внутри таблицы 8](#_Toc56294517)

[Реализация методов регистрации идентификатора, получения идентификатора из таблицы по имени 9](#_Toc56294518)

[Синтаксический анализатор 10](#_Toc56294519)

[Общая схема разбора 10](#_Toc56294520)

[Разбор инструкции присваивания 11](#_Toc56294521)

[Схема разбора сложных выражений 12](#_Toc56294522)

[Схема разбора инструкций ветвления 13](#_Toc56294523)

[Генератор кода 14](#_Toc56294524)

[Генерирование общих фрагментов кода, объявления переменных 14](#_Toc56294525)

[Генерация кода для логических операций, работа с ассемблерным стеком 16](#_Toc56294526)

[Разбор и генерация оператора печати 18](#_Toc56294527)

[Дополнительное задание 19](#_Toc56294528)

[Реализация условного оператора if else 19](#_Toc56294529)

[Пример работы программы и тестовые примеры 20](#_Toc56294530)

[Тестовый пример 1 20](#_Toc56294531)

[Тестовый пример 2 23](#_Toc56294532)

[Тестовый пример 3 24](#_Toc56294533)

[Тестовый пример 4 25](#_Toc56294534)

[Тестовый пример 5 25](#_Toc56294535)

[Заключение 27](#_Toc56294536)

[Источники 27](#_Toc56294537)

# Введение

Транслятор – это программа, которая переводит входной текст (например, программу, написанную на языке программирования высокого уровня, таком, как Паскаль или С) в машинный язык. Для построения транслятора применяют идею синтаксически-ориентированной трансляции, которая состоит в том, что для преобразования исходной информации, представленной в виде предложения входного языка, транслятор должен использовать структуру входного текста. Такой подход является обычным в науке: анализ и понимание сложной системы всегда явно или неявно опираются на структуру этой системы.

Трансляция программы — преобразование программы, представленной на одном из языков программирования, в программу на другом языке и, в определённом смысле, равносильную первой. Язык, на котором представлена входная программа, называется исходным языком, а сама программа — исходным кодом. Выходной язык называется целевым языком или объектным кодом. Понятие трансляции относится не только к языкам программирования, но и к другим компьютерным языкам, вроде языков разметки, аналогичных HTML, и к естественным языкам, вроде английского или русского.

# Цель работы

Основной целью курсовой работы является разработка транслятора, который переводит из сходного БНФ реализуемого языка в машинный ассемблер код

# Используемые технологии

* Java 11
* Java FX

# БНФ реализуемый язык

**Вариант 9**

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

<Описание вычислений> ::= Begin <Список присваиваний> End

<Объявление переменных> ::= Var <Список переменных> :Boolean; <Список переменных> ::= <Идент> | <Идент> , <Список переменных>

<Список присваиваний>::= <Присваивание> |

<Присваивание> <Список присваиваний> <Присваивание> ::= <Идент> = <Выражение> ;

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

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

<Ун.оп.> ::= ".NOT."

<Бин.оп.> ::= ".AND." | ".OR." | ".XOR."

<Операнд> ::= <Идент> | <Const>

<Идент> ::= <Буква> <Идент> | <Буква>

<Const> ::= 0 | 1

<Оператор печати>::=Print <Идент>

На одной строке может быть только объявление переменных или один оператор присваивания

# Создание транслятора

## Разработка пользовательского интерфейс

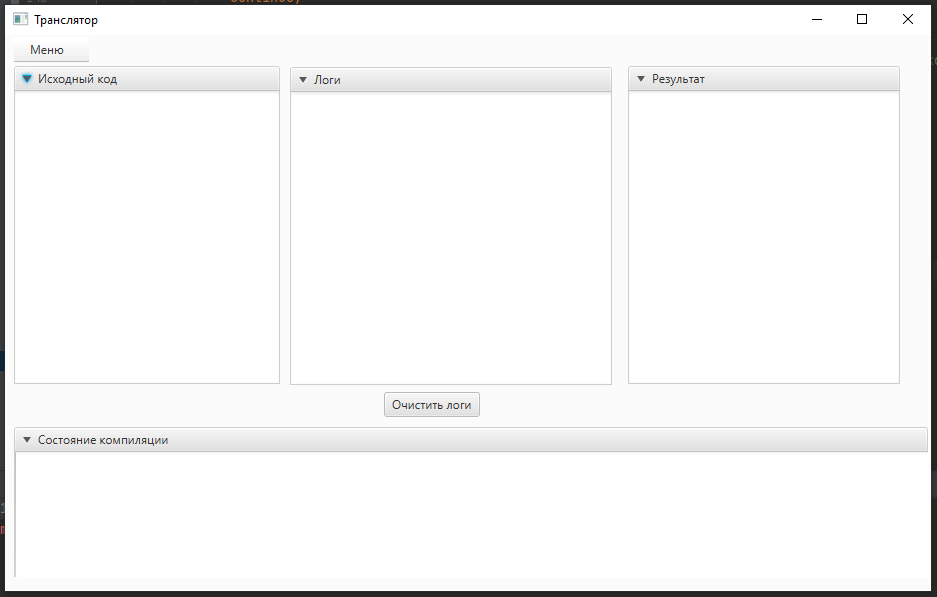


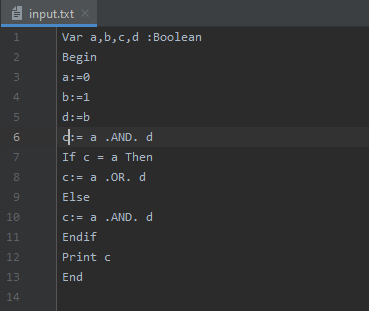
Рис. Вариант интерфейса

## 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 (‘.NOT.’ в программе)
* AND – логический оператор ('.AND.' в программе)
* OR – логический оператор ('.OR.' в программе)
* XOR– логический оператор ('.XOR.' 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)

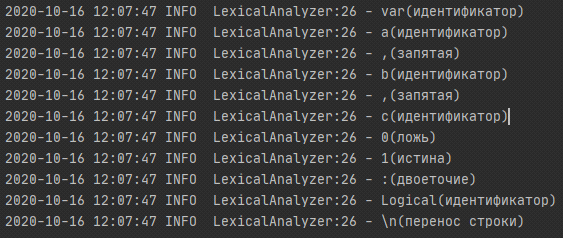
### Реализация в программе

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

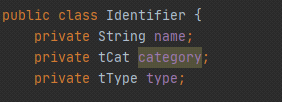
## Таблица имен

### Реализация элемента таблицы

Реализована следующая структура

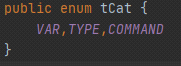
|  |  |  |
| --- | --- | --- |
| Имя | Категория | Тип |
| Строка | Переменная, тип данных, команда | Logical |

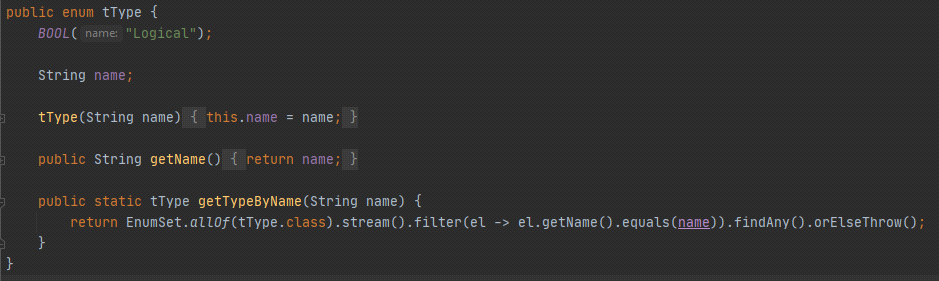
### Общие принципы хранения идентификаторов внутри таблицы



Идентификаторы хранятся в структуре «Идентификатор».

У каждого идентификатора есть имя, категория и тип.





Перечисление tType хранит тип данных исходного языка: bool.

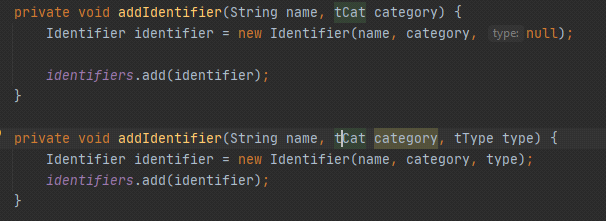
tCat - категории идентификаторов: константа, переменная, тип данных.

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

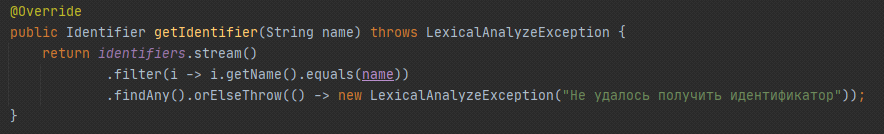


### Реализация методов регистрации идентификатора, получения идентификатора из таблицы по имени

Регистрация идентификатора



Получение идентификатора из таблицы по имени



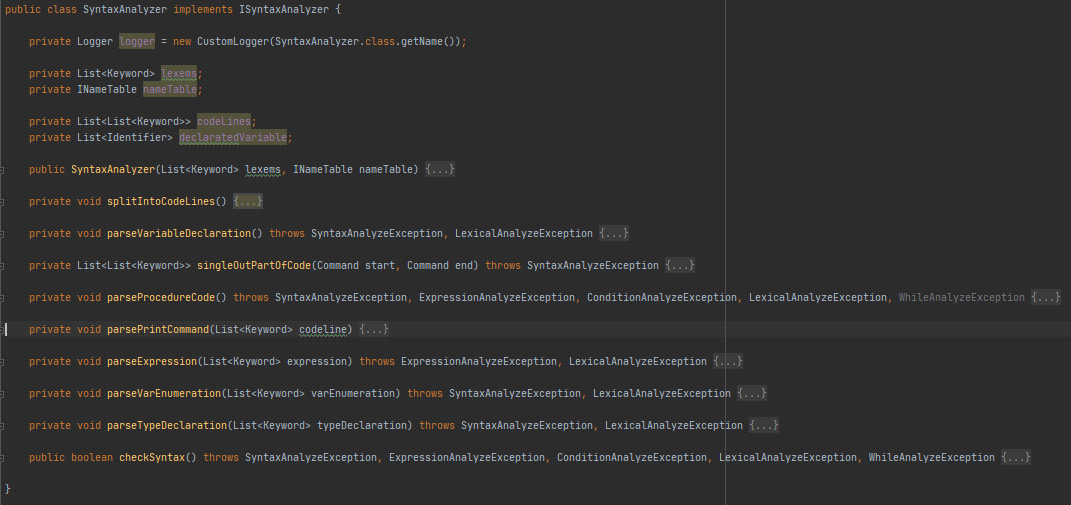
## Синтаксический анализатор

### Общая схема разбора

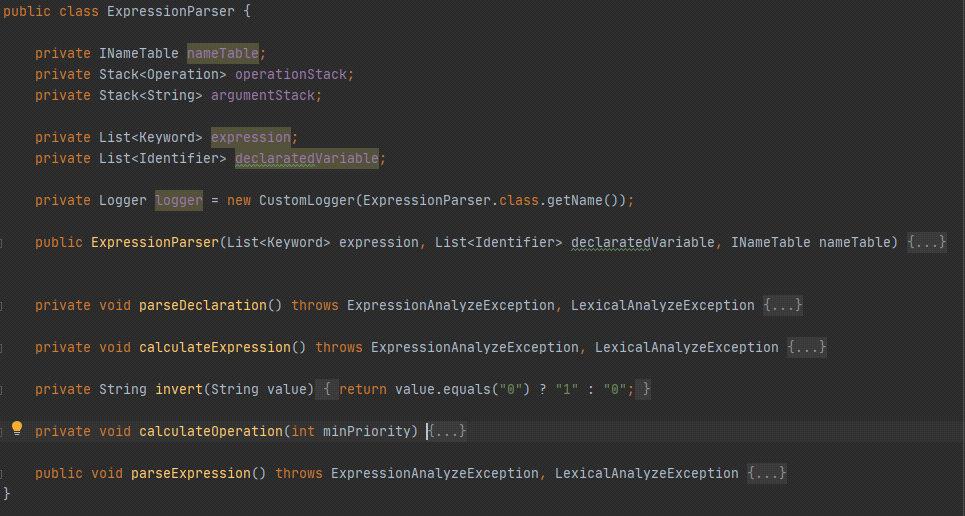
Разбор исходного кода состоит из:

* 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

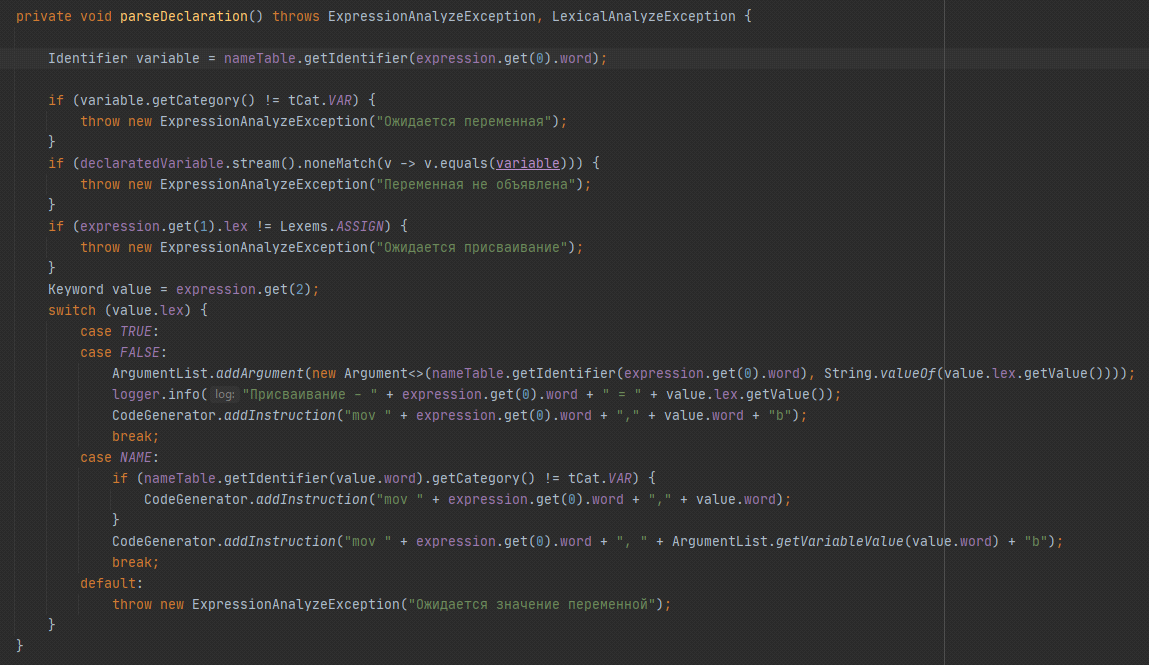


### Разбор инструкции присваивания

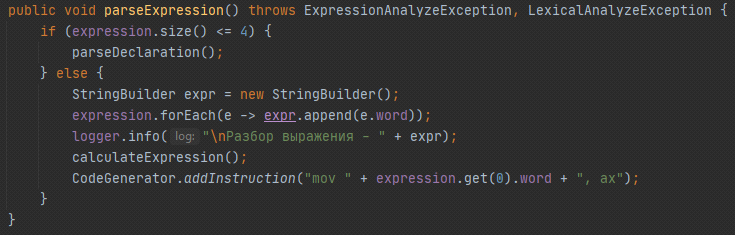
Виды присваивания:

* 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 .OR. b .AND. b | .OR. (2)  .AND. (1) | 1  1  0 |

Вычисление

|  |  |  |
| --- | --- | --- |
| Выражение | Стек операций | Стек аргументов |
| 0 .OR. 1 .AND. 1 | .AND. (2)  .OR. (1) | 1  1  0 |

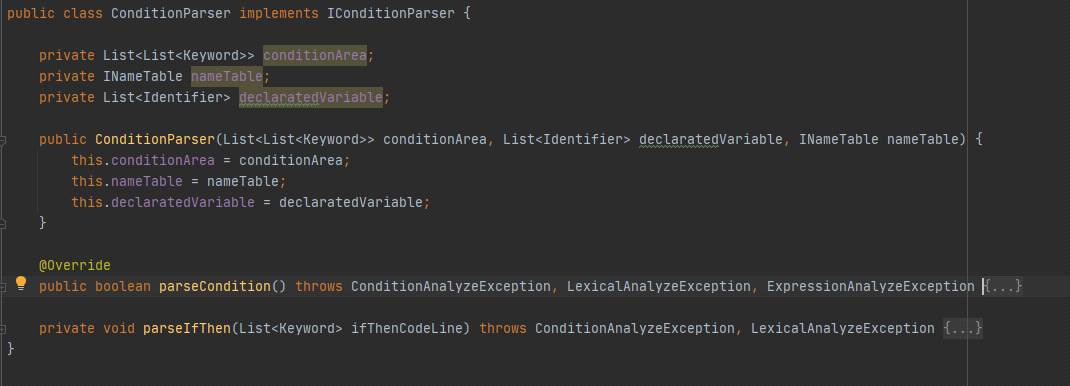
Вычисляем 1 .AND. 1 так как у следующей операции меньший приоритет

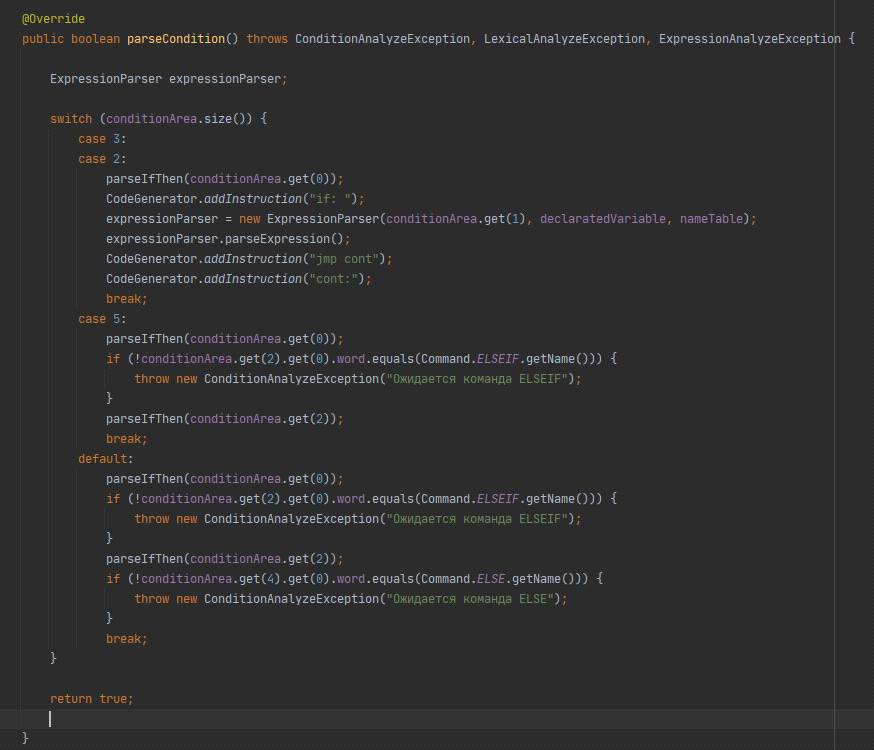
1 .AND. 1 = 1, кладем новое значение в стек

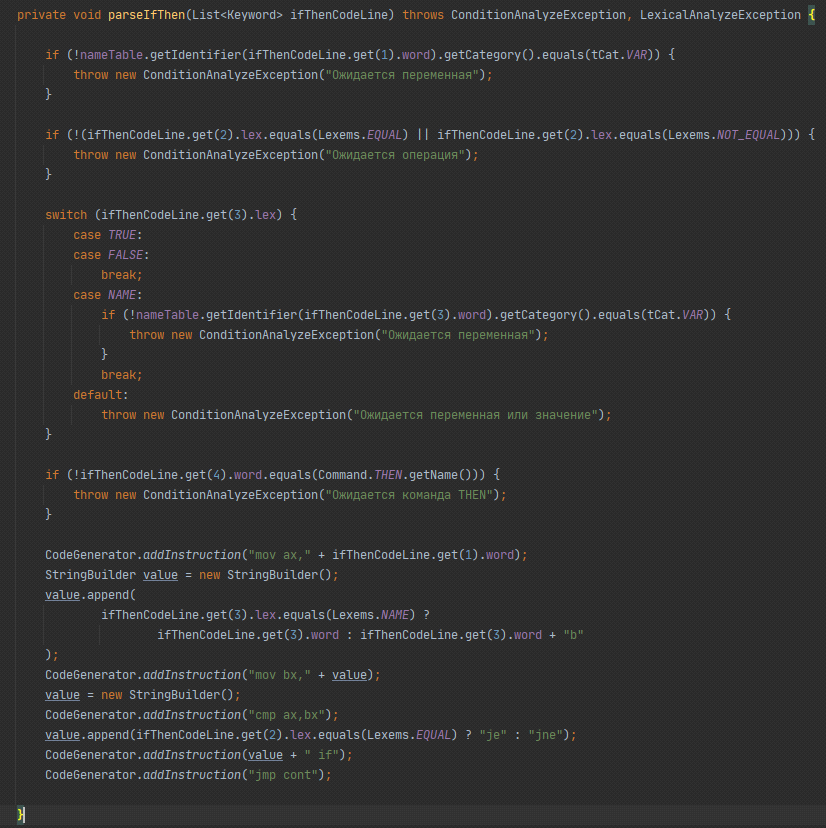
|  |  |  |
| --- | --- | --- |
| Выражение | Стек операций | Стек аргументов |
| 0 .OR. 1 .AND. 1 | .OR. (1) | 1  0 |

Вычисляем 1 .OR. 0, результат **1**

### Схема разбора инструкций ветвления







## Генератор кода

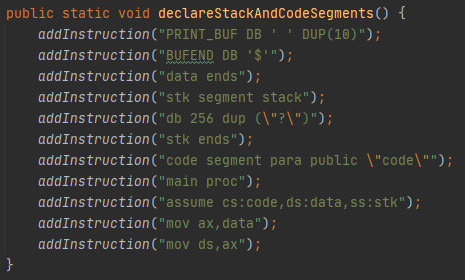
### Генерирование общих фрагментов кода, объявления переменных

Генерация общих фрагментов состоит из

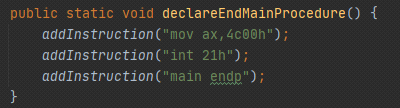
* Генерации сегмента данных



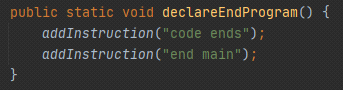
* Генерация сегментов стека и кода



* Генерация завершение основной процедуры



* Генерация завершения кода



* Генерация объявления переменных

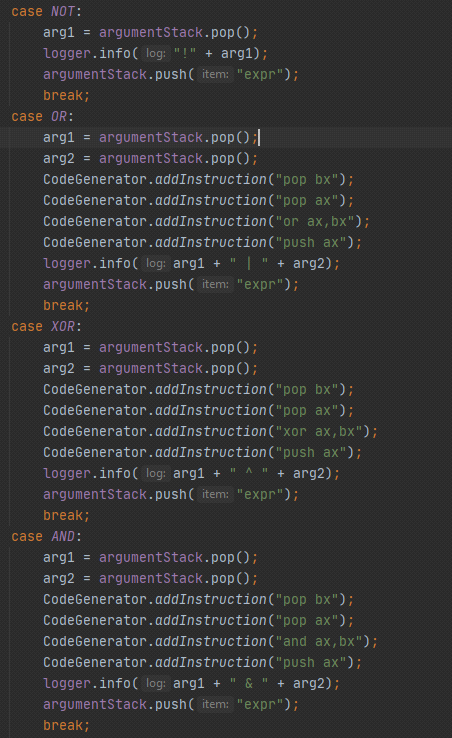


### Генерация кода для логических операций, работа с ассемблерным стеком

|  |  |
| --- | --- |
| Операция | Инструкции ассемблера |
| .AND. | pop bx  pop ax  and ax, bx  push ax |
| .OR. | pop bx  pop ax  or ax, bx  push ax |
| .XOR. | pop bx  pop ax  xor ax, bx  push ax |

Порядок работы с ассемблерным стеком

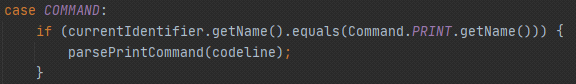
* Из стека достаются и удаляются два значения
* Совершается операция
* В стек кладется новое, вычисленное значение

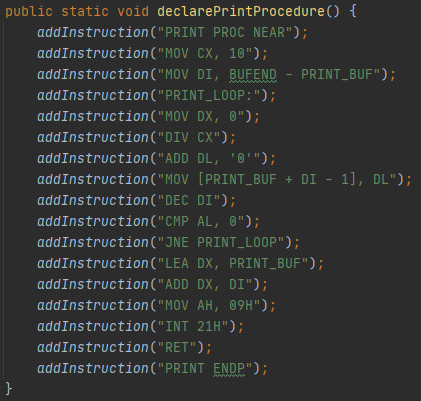


### Разбор и генерация оператора печати

При разборе команды PRINT происходит следующее:

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





## Дополнительное задание

### Реализация условного оператора if else

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 |

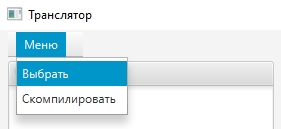
Пример

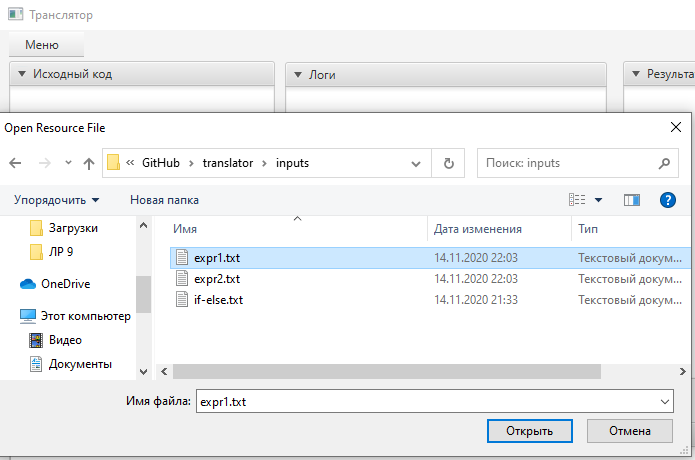
|  |  |
| --- | --- |
| Исходный код | Инструкции ассемблера |
| If c = a Then  c:= a .OR. b  Endif | cmp ax,bx **– сравниваем c и a**  je if **– если равны идем на метку if**  jmp cont **– если не равны идем на метку cont**  if: **- метка if**  mov ax,0b  push ax  mov ax,1b  push ax  pop bx  pop ax  or ax,bx  push ax  mov c, ax  jmp cont **– метка cont**  cont: |

## Пример работы программы и тестовые примеры

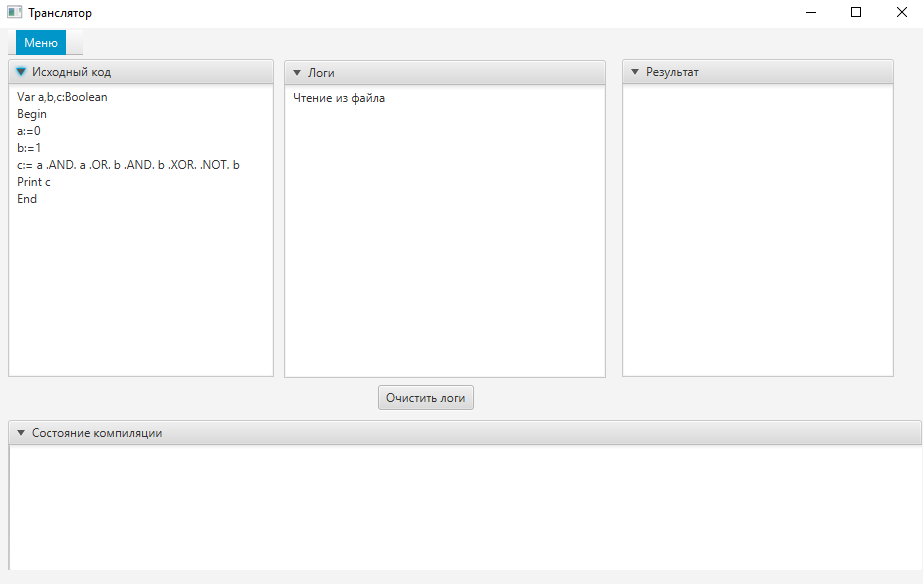
### Тестовый пример 1

Запускаем программу и выбираем фал с тестовым примером

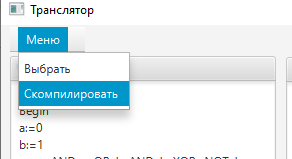




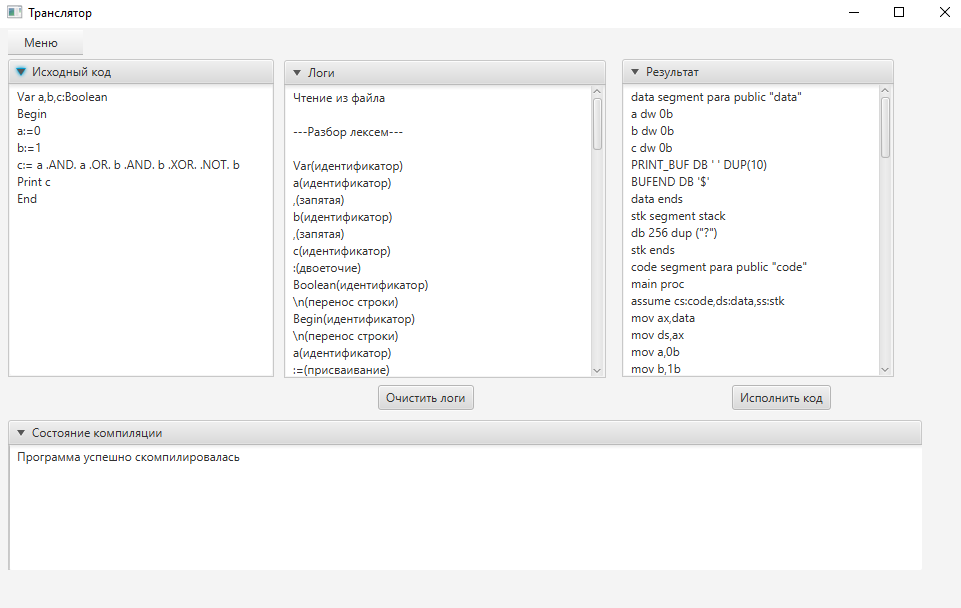
После выбора файла мы получаем данное окно



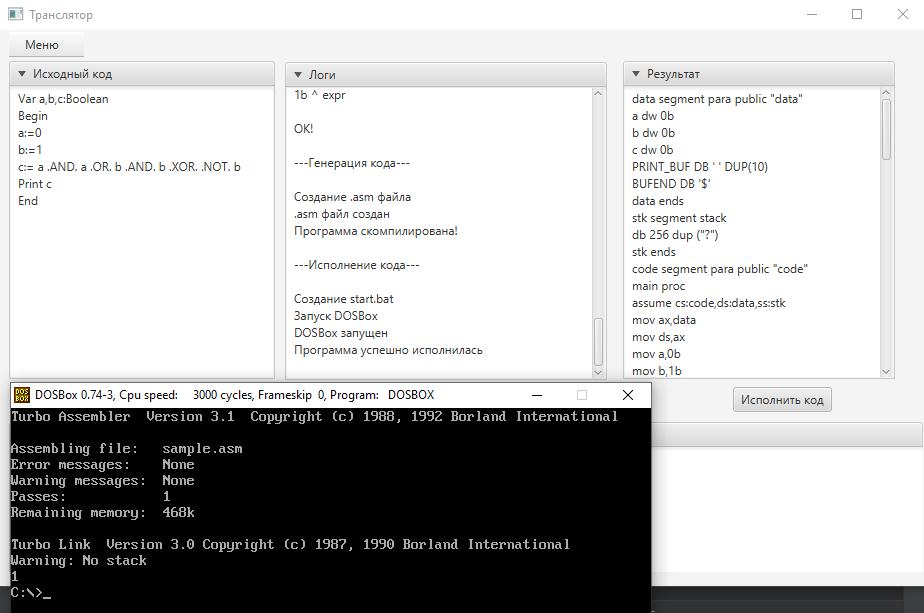
Открываем меню и жмем на компиляцию



Получаем исходный код и лог



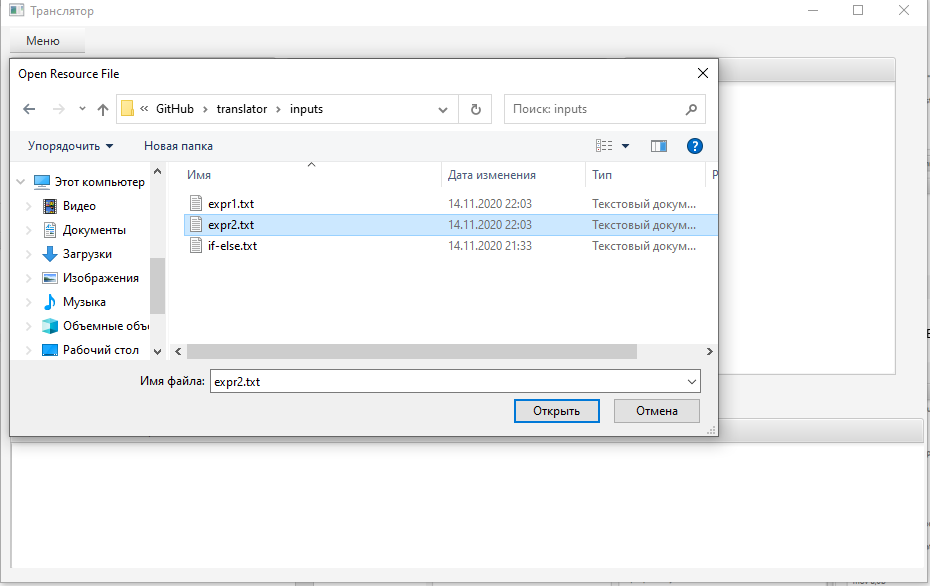
Далее исполняем наш код, путем нажатия на кнопку исполнить код



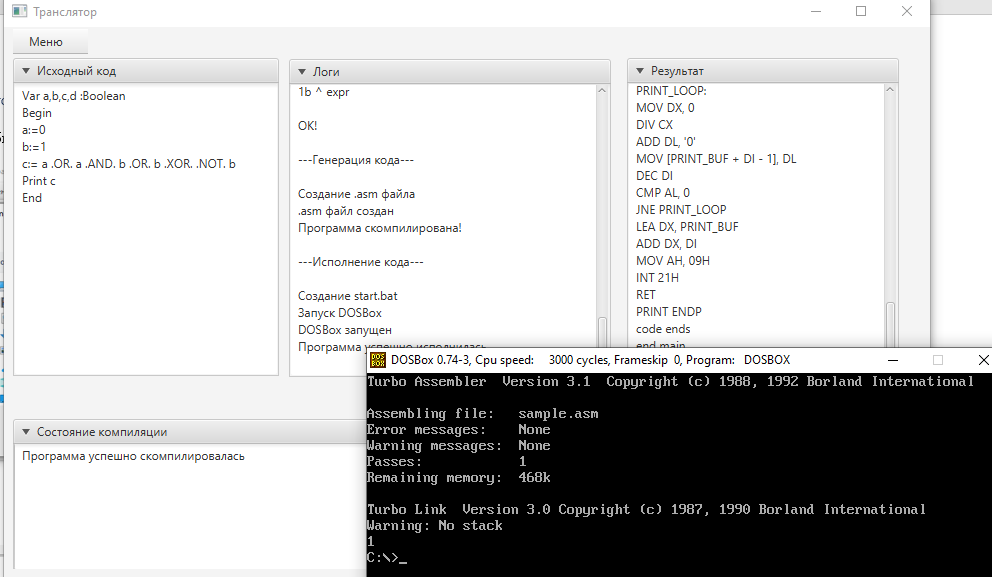
Программа сама конвертирует полученный нами код в фал с расширением .obj и запускает DosBox для вывода результата

### Тестовый пример 2

Выбираем файл со вторым примером

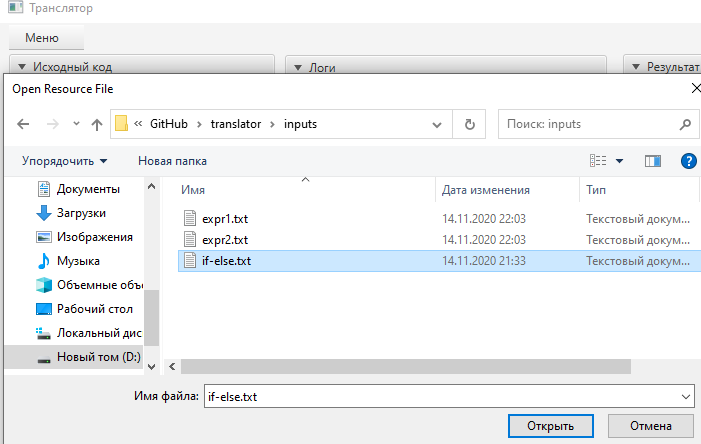


Компилируем полученный код и исполняем код

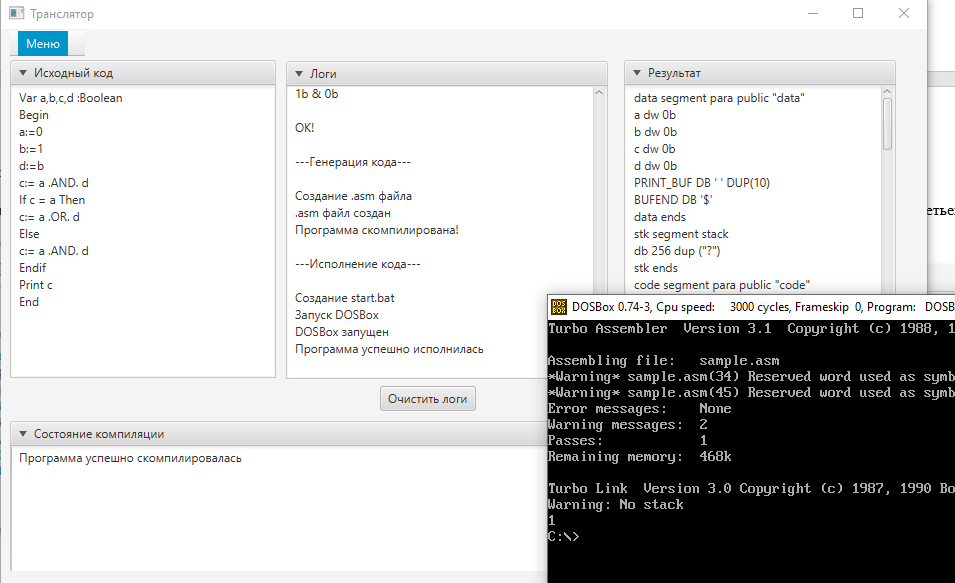


### Тестовый пример 3

Выбираем файл со третьем примером



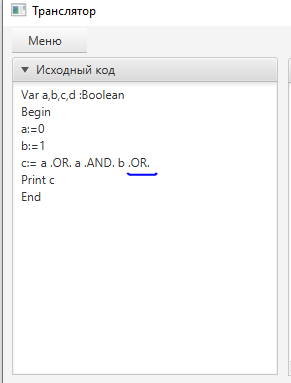
Компилируем полученный код и исполняем код



### Тестовый пример 4

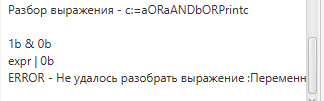
Лексическая ошибка

Входные данные:



Ожидаемый результат: обработчик ошибок должен обнаружить неизвестный символ

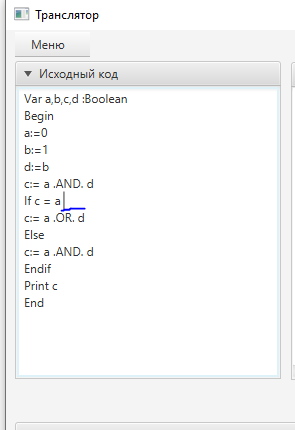
Результат:



### Тестовый пример 5

Синтаксическая ошибка

Входные данные:



Ожидаемый результат: обработчик ошибок должен обнаружить отсутствие команды Then в условии

Результат:



# Заключение

В ходе выполнения курсовой работы был разработан транслятор для конкретного бнф языка , реализован условный оператор if else , так же были проведены тесты данного транслятора.

# Источники

Методы построения трансляторов - Карпов Юрий Глебович С.Петербург, 2001г.