МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное

учреждение высшего образования

«Севастопольский государственный университет»

Кафедра информационных технологий

и компьютерных систем

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА к курсовому проекту по дисциплине «Системное программное обеспечения» на тему: «Разработка, отладка и тестирование транслятора

для подмножества учебного языка высокого уровня»

№ варианта 17

Выполнил:

ст.гр. ИВТб-32 \_\_\_\_\_\_\_\_\_\_\_\_\_ Теличко Д.В.

(подпись) (Ф.И.О.)

Руководитель:

Доцент \_\_\_\_\_\_\_\_\_\_\_\_\_ Фисун С. Н.

(ученая степень, должность) (подпись) (Ф.И.О.)

Дата защиты \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Севастополь 2018

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc515622487)

[1. ПОСТАНОВКА ЗАДАЧИ 6](#_Toc515622488)

[2. ОПИСАНИЕ МЕТОДОВ РЕШЕНИЯ ЗАДАЧИ 8](#_Toc515622489)

[2.1 Описание лексического анализатора 8](#_Toc515622490)

[2.2 Описание синтаксического анализатора 10](#_Toc515622491)

[2.3 Трансляция выражений в ПОЛИЗ 11](#_Toc515622491)

[2.4 Формирование таблицы триад 12](#_Toc515622491)

[2.5 Формирование кода на языке ASSEMBLER 12](#_Toc515622491)

[2.6 Алгоритмы решения задачи 14](#_Toc515622492)

[2.7 Преобразование грамматики 12](#_Toc515622493)

[2.8 Построение управляющей таблицы 16](#_Toc515622494)

[3. ОПИСАНИЕ ПРОГРАММЫ 18](#_Toc515622495)

[3.1 Назначение программы 18](#_Toc515622496)

[3.2 Описание структуры программы 18](#_Toc515622497)

[3.3 Описание классов 19](#_Toc515622497)

[3.4 Входные данные. 20](#_Toc515622498)

[3.5 Выходные данные 20](#_Toc515622499)

[4. ТЕСТОВЫЕ ПРИМЕРЫ 21](#_Toc515622500)

[ЗАКЛЮЧЕНИЕ 24](#_Toc515622501)

[БИБЛИОГРАФИЧЕСКИЙ СПИСОК 25](#_Toc515622502)

[ПРИЛОЖЕНИЕ А 26](#_Toc515622503)

[ПРИЛОЖЕНИЕ Б – Таблица лексем 27](#_Toc515622504)

[ПРИЛОЖЕНИЕ В – Управляющая таблица 28](#_Toc515622505)

[ПРИЛОЖЕНИЕ Г – Текст программы 29](#_Toc515622506)

# ВВЕДЕНИЕ

Системное программное обеспечение изучает комплекс программ, которые обеспечивают управление компонентами [компьютерной системы](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), такими как [процессор](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80), [оперативная память](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C), [устройства ввода-вывода](https://ru.wikipedia.org/wiki/%D0%A3%D1%81%D1%82%D1%80%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%B0_%D0%B2%D0%B2%D0%BE%D0%B4%D0%B0-%D0%B2%D1%8B%D0%B2%D0%BE%D0%B4%D0%B0), [сетевое оборудование](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%BE%D0%B5_%D0%BE%D0%B1%D0%BE%D1%80%D1%83%D0%B4%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), выступая как «межслойный интерфейс», с одной стороны которого аппаратура, а с другой — приложения пользователя. В отличие от [прикладного программного обеспечения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), системное не решает конкретные практические задачи, а лишь обеспечивает работу других программ, предоставляя им сервисные функции, абстрагирующие детали аппаратной и [микропрограммной](https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BA%D1%80%D0%BE%D0%BA%D0%BE%D0%B4) реализации вычислительной системы, управляет аппаратными ресурсами вычислительной системы.

В данном курсовом проекте разрабатывается  модуль транслятора. Транслятор это  обрабатывающая программа, предназначенная  для преобразования исходной программы  в объектный модуль. Транслятор обычно выполняет также диагностику ошибок и формирует словари идентификаторов. По способу  обработки входных данных  трансляторы  разделяются на компиляторы и интерпретаторы. В рамках данной курсовой работы модуль транслятора  состоит из блока лексического, синтаксического и семантического анализатора, блока интерпретации и построения  кода на языке ASSEMBLER.

# ПОСТАНОВКА ЗАДАЧИ

Разработать программу транслятора языка ASPLE1 c заданной грамматикой. При реализации синтаксического анализатора использовать алгоритм разбора рекурсивный спуск. Реализовать модуль трансляции выражений в ПОЛИЗ, формирование таблиц триад языка программирования ASPLE1, формирование кода на языке ASSEMBLER.Исходными данными является программа на языке ASPLE1.

Грамматика языка ASPLE1

<программа>::=BEGIN<посл. описаний>;<посл. операторов>END   
<посл. описаний>::=<описание>|<описание>;<посл. описаний>   
<посл. операторов>::=<оператор>|<оператор>;<посл. операторов>   
<описание>::=INT<список идент.>   
<список идент.>::=<идент.>|<идент.>,<список идент.>   
<оператор>::=<оператор присв.>|<оператор цикла>|   
<оператор обмена>   
<оператор присв.>::= <идент.>:=<выражение>   
<оператор цикла>::=WHILE<выражение>DO<посл. операторов>END   
<оператор обмена>::=INPUT<идент.>|OUTPUT<выражение>   
<выражение>::=<фактор>|<выражение>+<фактор>|<выражение>-<фактор>   
<фактор>::=<первичное>|<фактор>\*<первичное>|   
<фактор>DIV<первичное>|<фактор>MOD<первичное>   
<первичное>::=<идент.>|<константа>|(<выражение>)|(<сравнение>)   
<сравнение>::=<выражение>==<выражение>|   
<выражение><><выражение>   
<константа>::=<целая константа>   
<целая константа>::=<число>|(<число>|-<число>   
<число>::=<цифра>|<число><цифра>   
<цифра>::=0|1|2|3|4|5|6|7|8|9   
<идент.>::=<буква>|<идент.><буква>   
<буква>::=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

# ОПИСАНИЕ МЕТОДОВ РЕШЕНИЯ ЗАДАЧИ

## 2.1 Описание лексического анализатора

Лексический анализ — процесс аналитического разбора входной последовательности символов с целью получения на выходе последовательности символов, называемых «токенами» (подобно группировке букв в слова). Группа символов входной последовательности, идентифицируемая на выходе процесса как токен, называется лексемой. В процессе лексического анализа производится распознавание и выделение лексем из входной последовательности символов.

Традиционно принято организовывать процесс лексического анализа, рассматривая входную последовательность символов как поток символов. При такой организации процесс самостоятельно управляет выборкой отдельных символов из входного потока.

Лексический анализатор сам формирует таблицы объектов и выдает тип лексемы и указатель на соответствующий вход в таблице объектов. Эта информация впоследствии используется для синтаксического анализатора.

В основе ЛА лежит диаграмма переходов соответствующего конечного автомата. Отдельная проблема здесь - анализ ключевых слов. Как правило, лексический анализ производится с точки зрения определённого формального языка или набора языков. Язык, а точнее его грамматика, задаёт определённый набор лексем, которые могут встретиться на входе процесса. Как правило, ключевые слова - это выделенные идентификаторы.

Возможны два основных способа распознавания ключевых слов: либо очередная лексема сначала диагностируется на совпадение с каким-либо ключевым словом и в случае неуспеха делается попытка выделить лексему из какого-либо класса, либо, наоборот, после выборки лексемы идентификатора происходит обращение к таблице ключевых слов на предмет сравнения. Если соответствия не найдено, то символ пропускается. Отметим, что на вход лексического анализатора поступает текст программы без комментария.

Таблица лексем, разработанная для данного курсового проекта представлена в Приложение А. Отдельной лексемой, считались цифры, буквы (в нижнем регистре), зарезервированные слова (в верхнем регистре), знаки операций и отношений. Если зарезервированное слово написано нижним регистром, то оно считается как последовательность символов. Таблица лексем не выводится в графический интерфейс и служит для оптимизации кода.

Каждый язык программирования описывается с помощью набора правил, определяющих структуру правильной программы.

Грамматики одинаково помогают решать задачи как программистов, использующих язык, так и создателей компиляторов для данного языка:

• Грамматика предоставляет точную и достаточно легкую для понимания синтаксическую спецификацию языка программирования.

• Для некоторых классов грамматик мы можем автоматически сконструировать эффективный анализатор, который определяет, является ли исходная программа синтаксически правильной.

• Аккуратно созданная грамматика может придать языку программирования такую структуру, которая будет полезна и при трансляции исходной программы в правильный объектный код, и при определении ошибок.

• Компиляторы, разработанные на базе грамматик, могут быть достаточно легко расширены (это особенно полезно для добавления новых конструкций, появившихся как результат развития языка)

Еще раз подчеркнем, что с помощью контекстно-свободных грамматик определяется только так называемая контекстно-свободная составляющая языка программирования, то есть только то, каким образом записывается та или иная конструкция языка.

## 2.2 Описание синтаксического анализатора

По иерархии грамматик Хомского выделяют 4 основные группы языков (и описывающих их грамматик). При этом наибольший интерес представляют регулярные и контекстно-свободные (КС) грамматики и языки. Они используются при описании синтаксиса языков программирования. С помощью регулярных грамматик можно описать лексемы языка - идентификаторы, константы, служебные слова и прочие. На основе КС-грамматик строятся более крупные синтаксические конструкции - описания типов и переменных, арифметические и логические выражения, управляющие операторы, и, наконец, полностью вся программа на исходном языке.

Входные цепочки регулярных языков распознаются с помощью конечных автоматов (КА). Они лежат в основе сканеров, выполняющих лексический анализ и выделение слов в тексте программы на входном языке. Результатом работы сканера является преобразование исходной программы в список или таблицу лексем. Дальнейшую ее обработку выполняет другая часть компилятора - синтаксический анализатор. Его работа основана на использовании правил КС-грамматики, описывающих конструкции исходного языка.

Таким образом, на вход синтаксического анализатора получаем последовательность лексем после обработки лексическим анализатором, а на после работы синтаксического анализатора получаем дерево разбора.

2.3 Трансляция выражений в ПОЛИЗ

Обратная польская нотация (ОПН) — форма записи математических и логических выражений, в которой операнды расположены перед знаками операций. Также именуется как обратная польская запись, обратная бесскобочная запись (ОБЗ), постфиксная нотация, бесскобочная символика Лукасевича, польская инверсная запись, ПОЛИЗ.

В настоящее время широко используется в трансляторах метод трансляции выражений, основанный на использовании ПОЛИЗ (польской инверсной записи выражений).

Эта запись обладает следующими свойствами:

1. Операнды в ПОЛИЗ располагаются в том порядке, что и в обычной (инфиксной) записи;

2. Знаки операций расположены в том порядке, в котором нужно выполнить соответствующие действия;

3. Знак каждой операции записывается после соответствующих операндов;

При формировании ПОЛИЗа требует ввести новые типы операций и лексем, а именно:

\* условный переход по «false», сокращенно УПЛ, определяет, что по соответствующему значению логической переменной следует перейти к указанной метке;

\* безусловный переход, сокращенно БП, определяет, что следует перейти к указанной метке в обязательном порядке;

\* метка – которая позволяет указать ту часть ПОЛИЗа, к которой следует перейти.

Входящие данные: список лексем. Выходные данные: в поток вывода печатается таблица идентификаторов и операторы в формате ПОЛИЗ.

2.4 Формирование таблицы триад

Триады представляют собой запись операций в форме из четырех составляющих <Номер триады> <операция> <оп1> <оп2>, при этом один или оба операнда могут быть ссылками на другую триаду в том случае, если в качестве операнда данной триады выступает результат выполнения другой триады. Поэтому триады, при записи, последовательно номеруют для удобства указания ссылок одних триад на другие. Такой тип представления триад называют «Тройками». Являются более короткой и удобной для представления выражений в виде таблицы триад.

2.5 Формирование кода на языке ASSEMBLER

После получения таблицы триад необходимо сформировать код на языке ASSEMBLER, при этом стоит учитывать особенности и возможности языка, особенности описания циклов в нем, а так же ситуации, когда в триаде операндами являются другие триады, для этого была использована схема су-перевода на язык ASSEMBLER, которая была приведена в лекция, отдельно учтены свойства операций, для которых возможна перестановка операндов без изменения результатов, это позволило уменьшить количество команд на языке ASSEMBLER.

2.6 Алгоритмы решения задачи

Грамматика, таблица анализа которой не имеет множественных записей, называется LL(1). Первое "L" означает просмотр входного потока слева направо, второе "L"— левое порождений а "1" — просмотр одного символа из входного потока на каждом шаге для принятия решения о дальнейших действиях. LL(1)-грамматики имеют ряд отличительных свойств. Такая грамматика не может быть неоднозначной или леворекурсивной. Можно также показать, что грамматика G является LL(1)-грамматикой тогда и только тогда, когда для любых двух различных ее продукций А —> *α* | *β* выполняются следующие условия.

1. Не существует такого терминала *а*, для которого и *α*, и *β* порождают строку, начинающуюся с *а*.

2. Пустую строку может порождать только одна из продукций *α* или *β*

3. Если *β* =>ε, то *α* не порождает ни одну строку, начинающуюся с терминала из FOLLOW(А).

Синтаксический анализатор, управляемый таблицей, имеет входной буфер, стек, таблицу разбора и выходной поток. Входной буфер содержит анализируе­мую строку с маркером ее правого конца — специальным символом. Стек содержит по­следовательность символов грамматики с $ на дне. Изначально стек содержит стартовый символ грамматики непосредственно над символом $. Таблица разбора представляет со­бой двухмерный массив М[А, а], где А — не терминал, *а* — терминал или символ $.

Синтаксический анализатор управляется программой, которая работает следующим образом. Программа рассматривает X— символ на вершине стека, и *а*, текущий входной символ. Эти два символа определяют действия синтаксического анализатора. Имеется три варианта.

1. Если Х=*а*=$> синтаксический анализатор прекращает работу и сообщает об ус­пешном завершении разбора.

2. Если X = *а* ≠ $, синтаксический анализатор снимает со стека X и перемещает указа­тель входного потока к следующему символу.

3. Если X представляет собой не терминал, программа рассматривает запись М[Х, а) из таблицы разбора М. Эта запись представляет собой либо X-продукцию грамматики, либо запись об ошибке. Если, например, М[Х, а) = {X—> *UVW*), синтаксический ана­лизатор замешает X на вершине стека на *WVU* (с *U* на вершине стека). Мы полагаем, что в качестве выхода синтаксический анализатор просто выводит использованную продукцию, но, конечно же, здесь может выполняться любой необходимый код. Ес­ли М[Х, а] = **error**, синтаксический анализатор вызывает программу восстановления после ошибки.

Поведение синтаксического анализатора может описываться его конфигурациями, которые дают содержимое стека и оставшийся входной поток.

## 2.7 Преобразование грамматики

Правила которые остаются из грамматики языка ASPLE1:

1. <программа>→BEGIN<посл. описаний>;<посл. операторов>END
2. <посл. описаний>→<описание>
3. <посл. описаний>→<описание>;<посл. описаний>
4. <посл. операторов>→<оператор>
5. <посл. операторов>→<оператор>;<посл. операторов>
6. <описание>→INT<список идент.>
7. <список идент.> →<идент.>
8. <список идент.> →<идент.> ,<список идент.>
9. <оператор>→<оператор присв.>
10. <оператор>→ <оператор цикла>
11. <оператор>→<оператор обмена>
12. <оператор присв.> →<идент.>:=<выражение>
13. <оператор цикла>→WHILE<выражение>DO<посл. операторов>END
14. <оператор обмена>→INPUT<идент.>
15. <оператор обмена>→OUTPUT<выражение>
16. <выражение>→<фактор>
17. <выражение>→<выражение>+<фактор>
18. <выражение>→<выражение>-<фактор>
19. <фактор>→<первичное>
20. <фактор>→<фактор>\*<первичное>
21. <фактор>→<фактор>DIV<первичное>
22. <фактор>→<фактор>MOD<первичное>
23. <первичное>→<идент.>
24. <первичное>→<константа>
25. <первичное>→(<выражение>)
26. <первичное>→(<сравнение>)
27. <сравнение>→<выражение>==<выражение>
28. <сравнение>→<выражение><><выражение>
29. <константа>→<целая константа>
30. <целая константа>→<число>
31. < целая константа>→(<число>
32. <целая константа>→-<число>
33. <число>→<цифра>
34. <число>→<число><цифра>
35. <цифра>→0|1|2|3|4|5|6|7|8|9
36. <идент.>→<буква>
37. <идент.>→<идент.><буква>

<буква>→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

Для разработки синтаксического анализатора, нетерминалы <идент.> и <число> будут заменены на терминалы идент и число.   
Грамматика не соблюдает условиям применимости рекурсивного спуска:

Правила

17, 18, 20, 21, 22,– левая рекурсия;

2 и 3, 4 и 5, 7 и 8, 9 и 10 и 11, 16 и 17 и 18, 19 и 20 и 21 и 22, 23 и 24, 27 и 28 неоднозначны - начинаются с нетерминалов.

25 и 26 неоднозначны – начинаются с одинаковых терминалов

Преобразуем грамматику в эквивалентную с устранением этих несоблюдений.

Конечная грамматика ASPLE1 после преобразований:

1. <программа>→BEGIN<посл. описаний>.<посл. операторов>END
2. <посл. описаний>→<описание><A>
3. <A>→;<посл. описаний>
4. <A>→ɛ
5. <посл. операторов>→<оператор><B>
6. <B>→;<посл. операторов>
7. <B>→ɛ
8. <описание>→INT<список идент.>
9. <список идент.> →идент. <C>
10. <C>→,<список идент.>
11. <C>→ ɛ
12. <оператор>→идент. := <выражение>
13. <оператор>→WHILE<выражение>DO<посл. операторов>END
14. <оператор>→INPUT идент.
15. <оператор>→OUTPUT<выражение>
16. <выражение>→<фактор><G>
17. <G>→+<выражение >
18. <G>→-<выражение>
19. <G>→ ɛ
20. <фактор>→<первичное><E>
21. <E>→\*<фактор >
22. <E>→ DIV<фактор>
23. <E>→ MOD<фактор>
24. <E>→ ɛ
25. <первичное>→идент.
26. <первичное>→число|-число
27. <первичное>→(<F>)
28. <F>→<сравнение>
29. <F>→число
30. <сравнение>→<выражение><J>
31. <J>→==<выражение>
32. <J>→<><выражение>
33. <J>→ ɛ

## 2.8 Построение управляющей таблицы

В пункте 2.1 была получена грамматика и по ней построена таблица лексем. Далее будут представлены все правила с применением таблицы лексем (см. приложение Б).

1. <П>::= BEGIN <ПО>;<ПОП>END
2. <ПО>::=<ОП> <A >
3. <A>::=;<ОП>
4. <A>::= E
5. <ПОП>::= <ОПЕР><B>
6. <B>::= ;<ПОП>
7. <B>::= E
8. <ОП>::= INT<СИД>
9. <СИД>::= ИД<C>
10. <С>::= ,<СИД>
11. <С>::= E
12. <ОПЕР>::= ИД:=<ВЫР>
13. <ОПЕР>::= WHILE<ВЫР>DO<ПОП >END
14. <ОПЕР>::= INPUT ИД
15. <ОПЕР>::= OUTPUT<ВЫР>
16. <ВЫР>::=<ФАКТ><G>
17. <G>::= +<ВЫР >
18. <G>::= -<ВЫР>
19. <G>::= E
20. <ФАКТ>::=<П><E>
21. <E>::= \*<ФАКТ>
22. <E>::= DIV<ФАКТ>
23. <E>::= MOD<ФАКТ>
24. <П>::= ИД
25. <П>::= ЧИСЛО
26. <П>::= -ЧИСЛО
27. <П>::= (<F>
28. <F>::= <СР>)
29. <F>::= ЧИСЛО
30. <СР>::= <ВЫР><J>
31. <J>:= ==<ВЫР>
32. <J>:= <><ВЫР>
33. <J>:= E

Управляющая таблица LL1 представлена в приложении В.

# 3. ОПИСАНИЕ ПРОГРАММЫ

Программа выполняет синтаксический анализ заданной грамматики языка ASPLE1. Для разработки синтаксического анализа был взят язык программирования java. Текст программы приведен в Приложении Д.

## 3.1 Назначение программы

Программа предназначена для первичной обработки исходного текста программы на языке высокого уровня ASPLE1, выделения в тексте терминальных символов, формирования ПОЛИЗа, таблицы триад, кода на языке ASSEMBLER. После работы лексического анализатора из исходного текста программы удаляются комментарии и выделяются лексемы. Синтаксический анализатор проверяет соответствие текста программы грамматике.

## 3.2 Описание структуры программы

Процесс трансляции, от написания кода программы на языке грамматики ASPLE1 , содержит в себе следующие этапы:

1) Лексический анализ, выделение типовой подстроки в исходной программе с целью проверки исходного текста и формирования кода программы в виде цепочки лексем, в которой отсутствуют лишние данные. На данном этапе могут возникать ошибки, которые говорят о том, что к дальнейшим этапам приступать нельзя;

2) Синтаксический анализ, на этапе которого проверяется соответствие цепочки лексем правилам исходной грамматики, если цепочка не соответствует правилам, к дальнейшим этапам приступать нельзя;

3) Формирование цепочки лексем и операндов, в которой операции встречаются в порядке их выполнения;

4) Формирование таблицы триад, в которой выделена отдельно каждая последующая операция

5) Формирование кода на языке ASSEMBLER, данный текст можно скомпилировать и получить итоговую программу, изначально написанную на ASPLE1

3.3 Описание классов

Класс Main– главный класс программы. Позволяет пользователю ввести исходную программу с помощью файла input.txt с текстом программы.

main(String[] args) – основной метод главного класса.

Класс LA–класс программы лексического анализатора. Позволяет пользователю получить таблицу лексем из введённой программы.

public static String analyze(String input) –анализпрограммы. Вход: код программы ASPLE. Вывод: таблица лексем.

Класс SA– класс программы синтаксического анализатора. Позволяет пользователю проверить правильность цепочки лексем, полученной на этапе лексического анализа.

public static boolean analyze(String input) – анализ таблицы лексем. Вход: код программы таблица лексем. Вывод: допуск или ошибка в зависимости от правильности введённого кода.

void error() – Метод вызова ошибки. Останавливает выполнение программы.

Класс Poliz.java – класс, разработанный для формирования ПОЛИЗА из кодированной строки лексем.

public static String buildPoliz(String input) – построение ПОЛИЗа. Вход: кодированная строка лексем. Вывод: ПОЛИЗ программы.

Класс TT.java – класс, разработанный для формирования таблицы триад из ПОЛИЗа программы.

public static String BuildTriads(String input) – построение таблицы триад. Вход: ПОЛИЗ программы. Вывод: таблица триад.

Класс Asembling.java – класс, разработанный для формирования кода на языке ASSEMBLER из таблицы триад.

public static String GetAsm(String table) – формирование кода на языке ASSEMBLER. Вход: таблица триад. Вывод: код на языке ASSEMBLER.

## 3.4 Входные данные.

Входными данными является строка соответствующая подмножеству 1 языка ASPLE1. Т.е. любая комбинация таких символов: “П”, “ПО”, “A”, “ПОП”, “B”, “ОП”, “СИД”, “С”, “OПЕР”, “ВЫР”, “G”, “ФАКТ”,“E”, “П”, “F”, “CP”, “J”, а также для терминала “переменная” используется буква, для терминала “число” используется только одна цифра.

## 3.5 Выходные данные

Выходные данные представляют собой – код на языке ASSEMBLER.

# ТЕСТОВЫЕ ПРИМЕРЫ

Для тестирования программы необходимо написать несколько тестов на корректную работу программы и некорректную (когда в коде, входная грамматика, содержит различные ошибки).

Тестовый пример 1. Программа работает успешно, цепочка принадлежит грамматике языка ASPLE1.

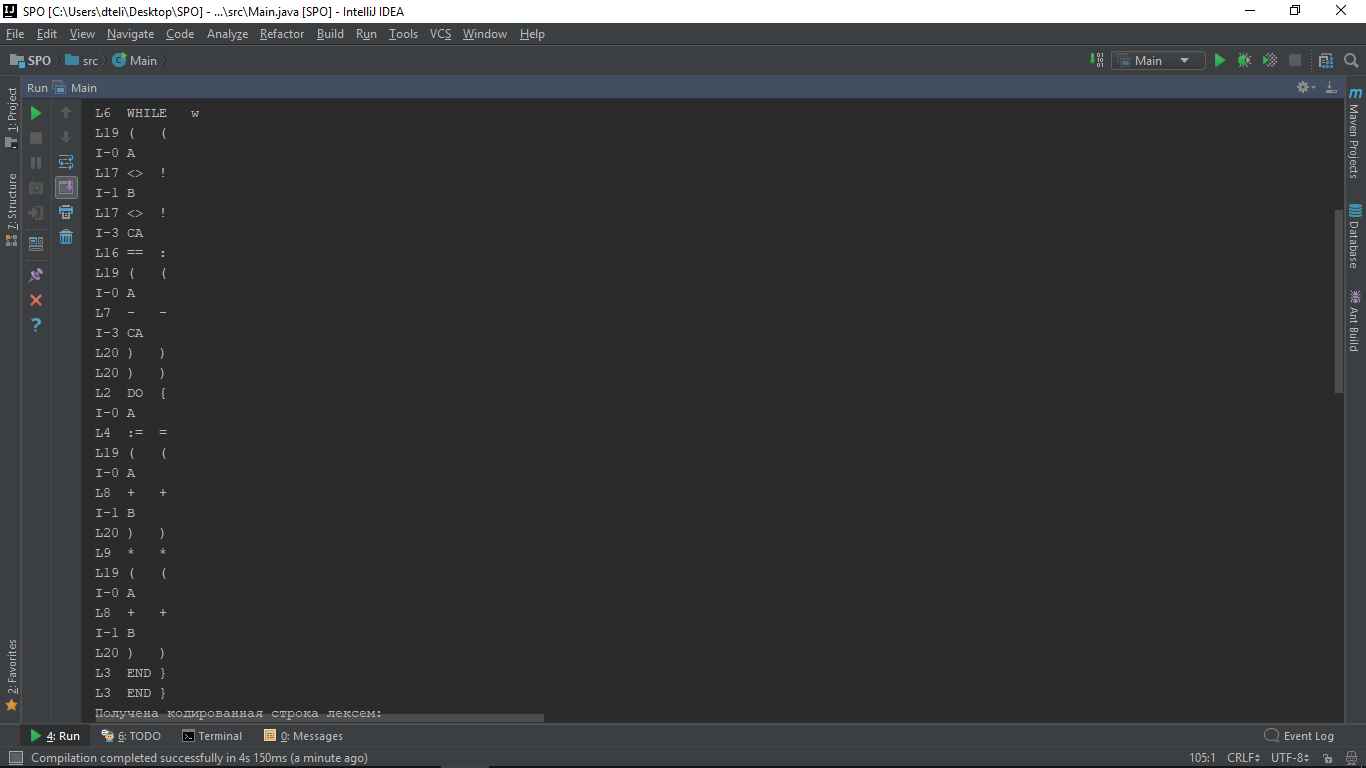
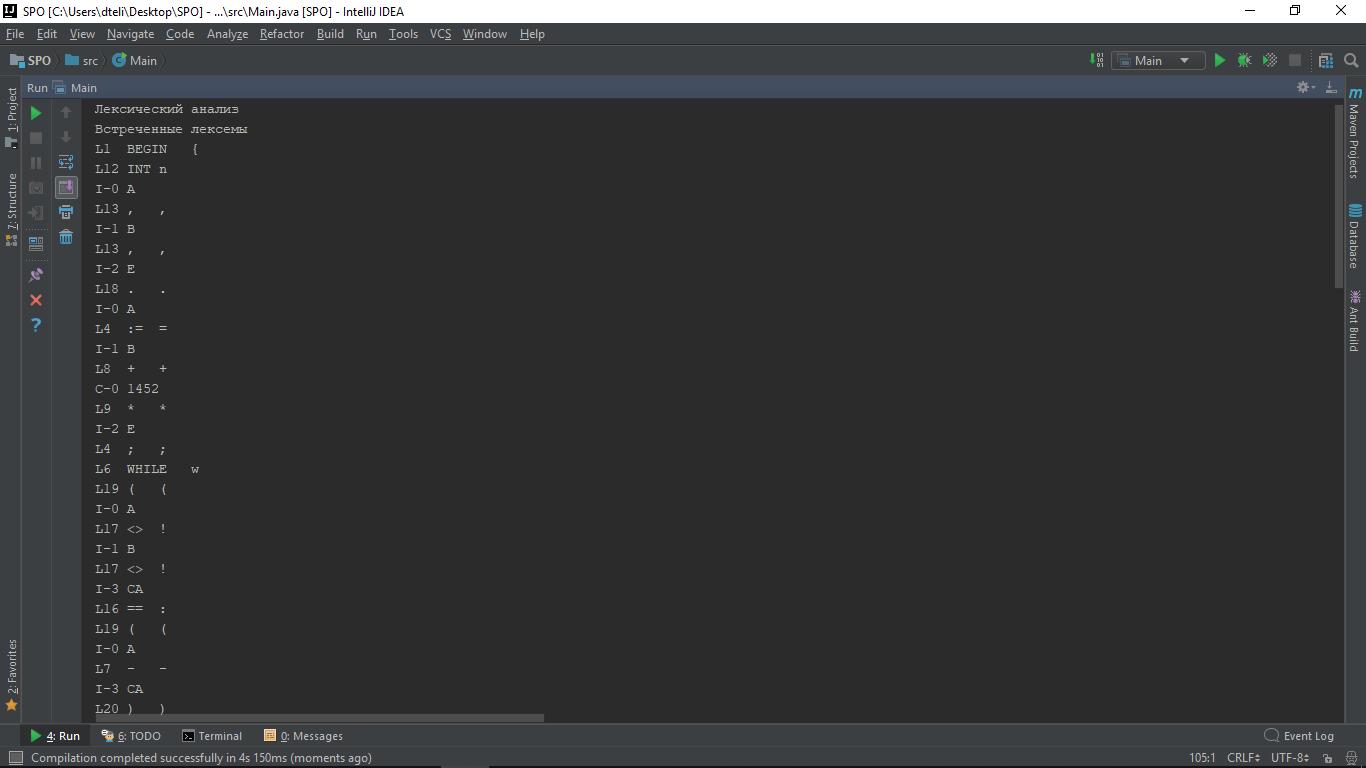
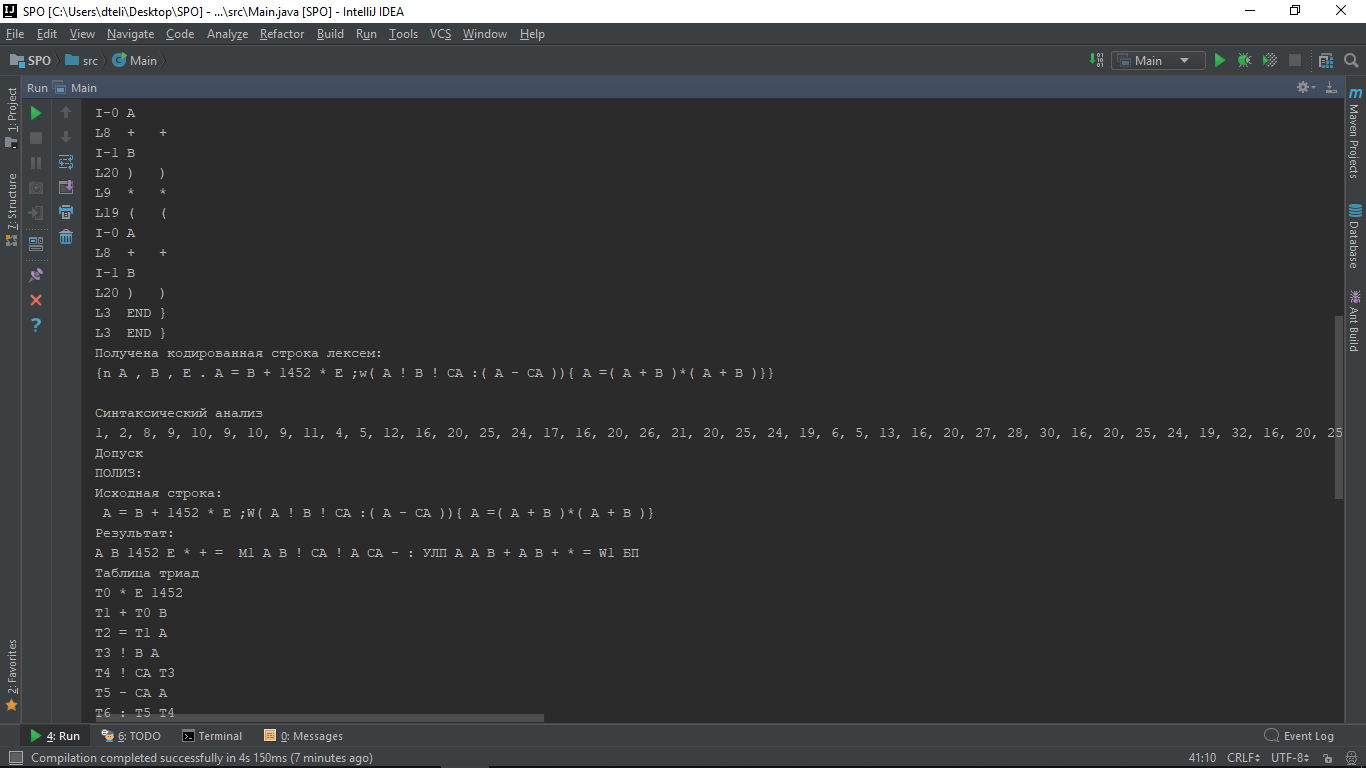


Рисунок 1 – тест №1

Рисунок 2 – тест №1

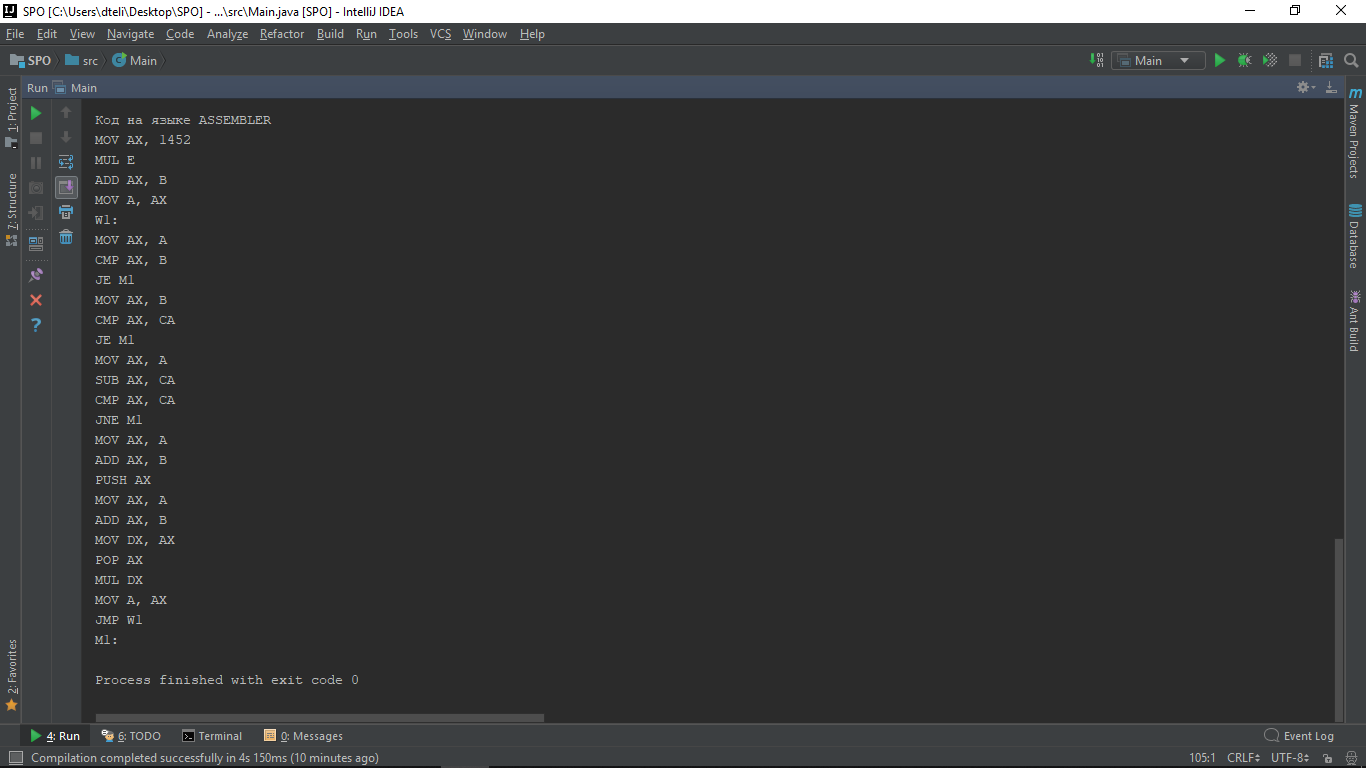
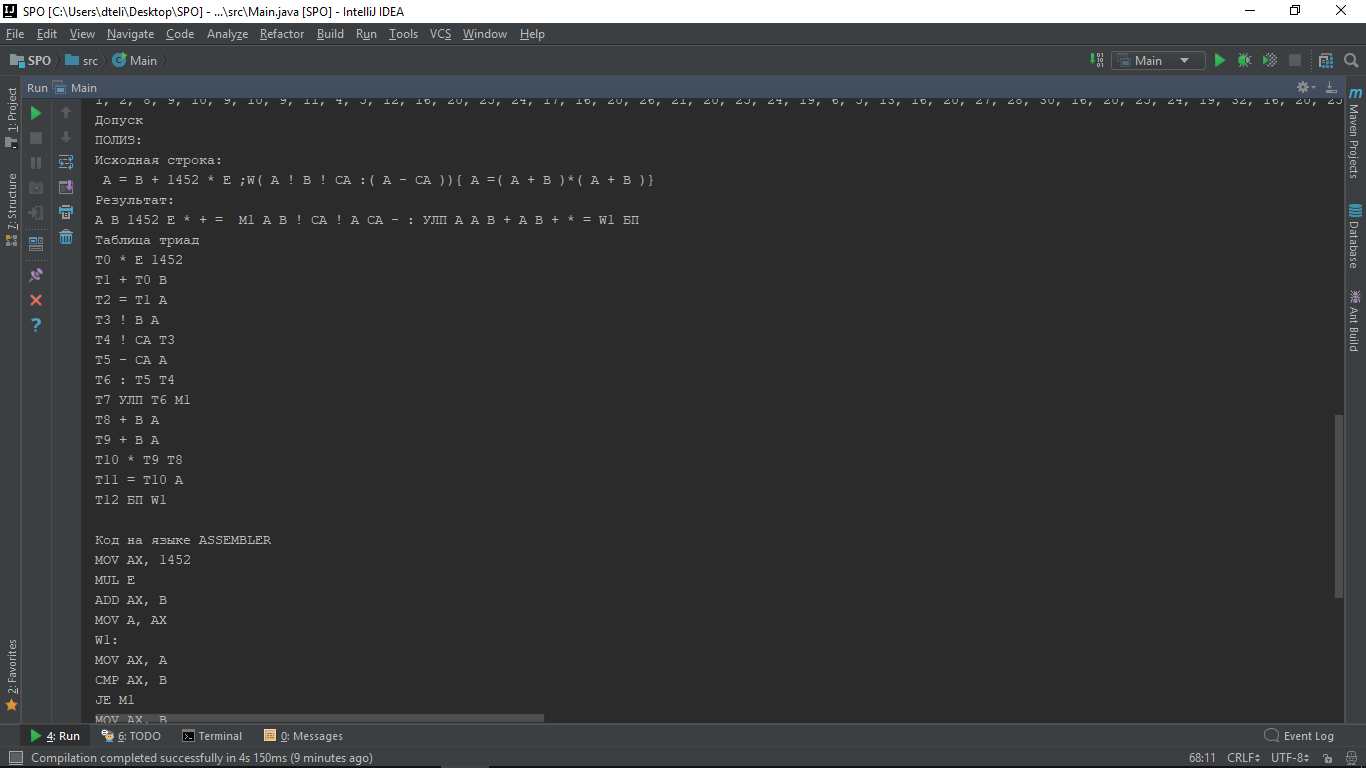


Рисунок 3 – тест №1

Тестовый пример 2. Программа работает успешно, лексически неправильные входные данные.

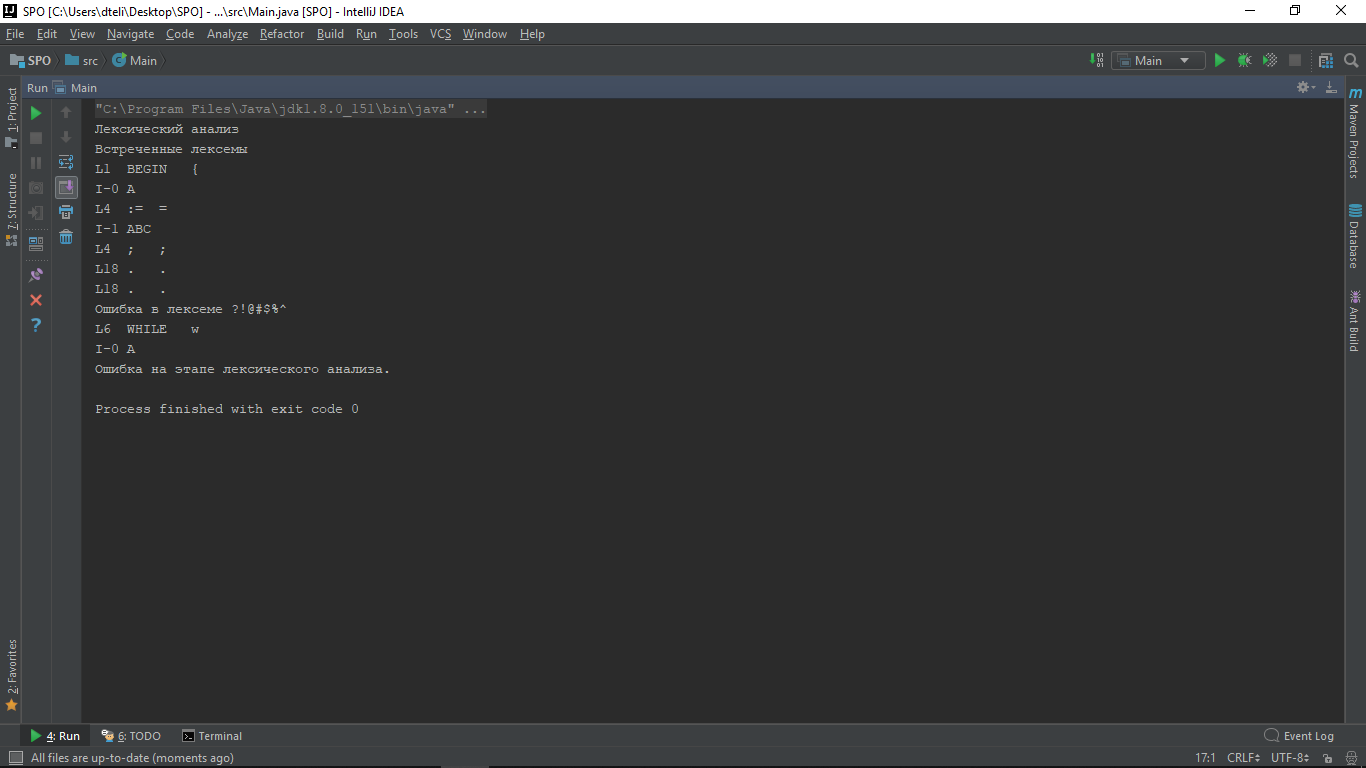


Рисунок 4 – Тест №2

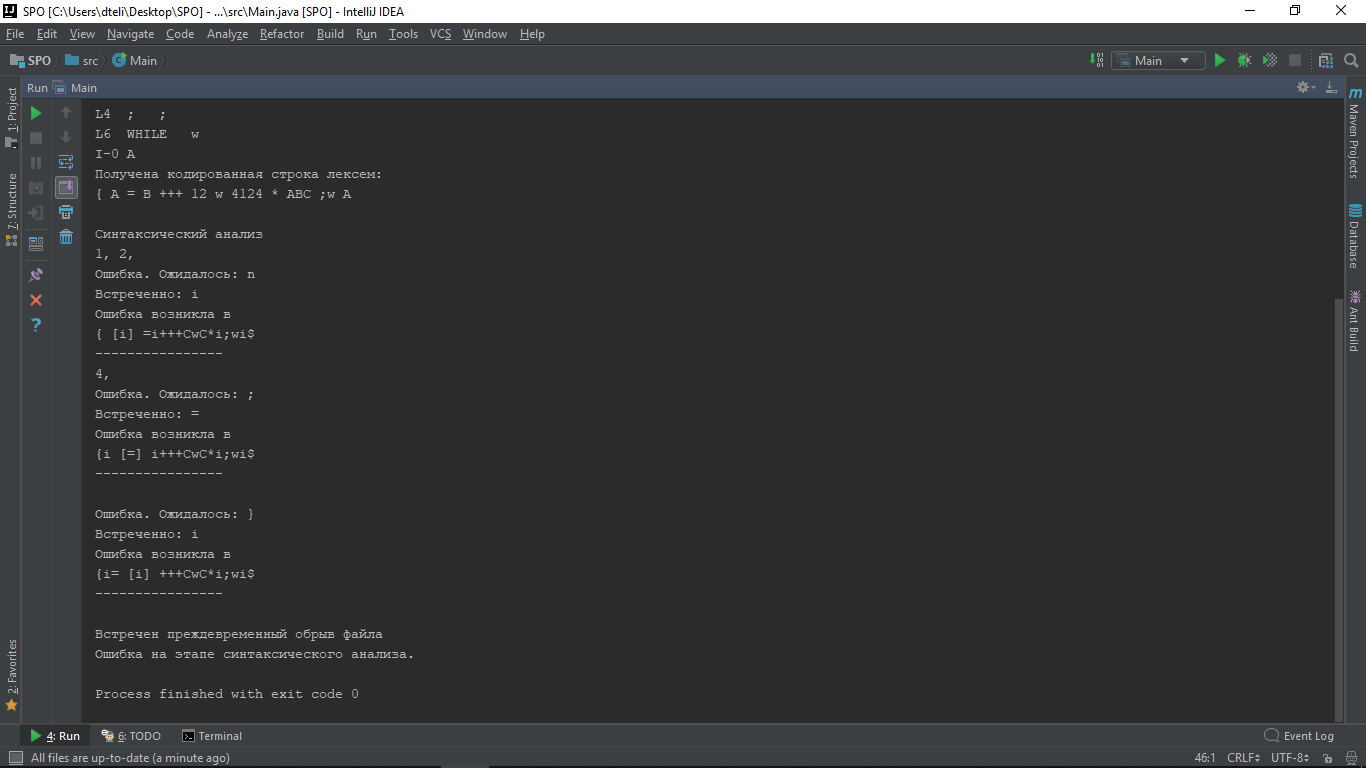
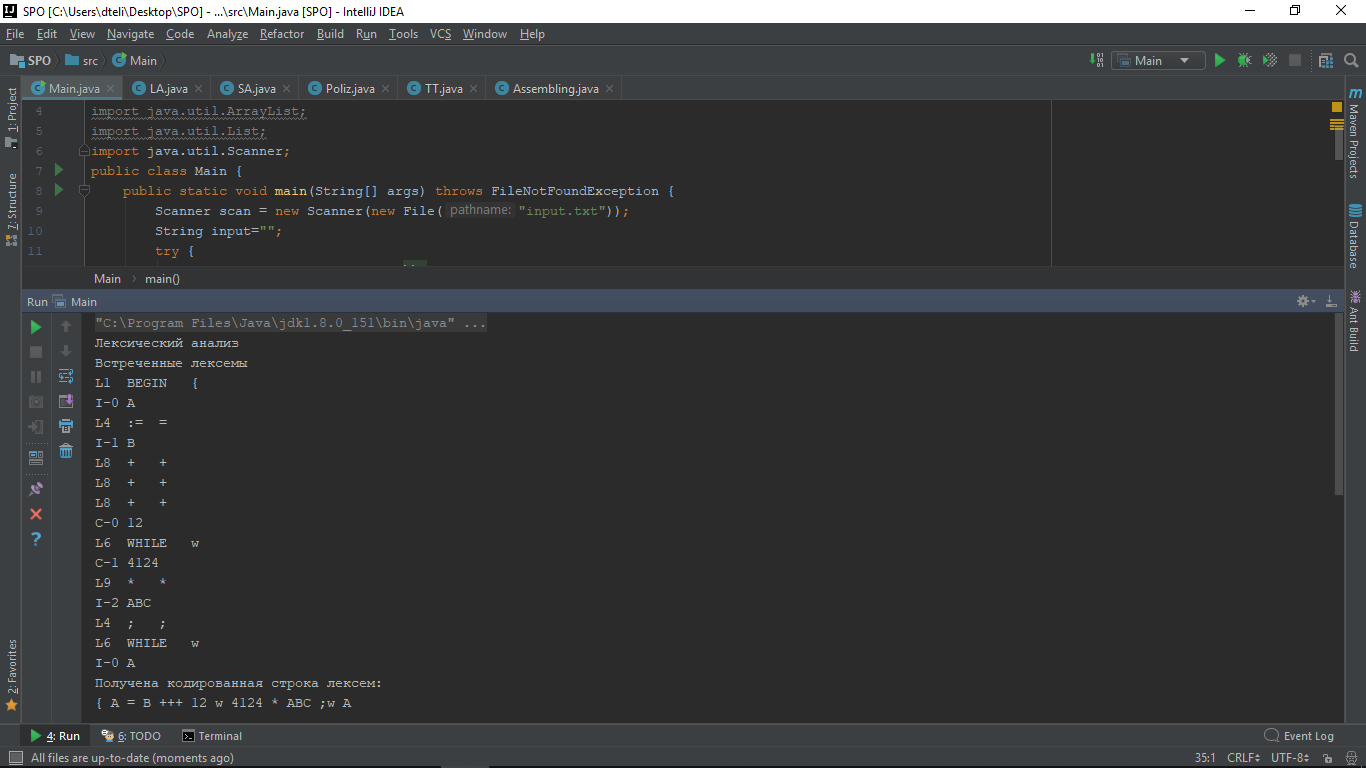
Тестовый пример 3. Программа работает успешно, синтаксически неправильные входные данные.  


Рисунок 3 – Тест №3

# ЗАКЛЮЧЕНИЕ

В ходе курсового проекта был разработан модуль на языке Java , который реализует блок лексического синтаксического анализа. Программный продукт прошёл тестирование на многочисленных тестовых примерах. На всех тестах модуль работал корректно, из чего можно сделать вывод, что цель данного курсового проекта достигнута. Так же был приобретен опыт разработки программного обеспечения в команде, распределения нагрузки на каждого участника команды разработчиков. Созданную в результате выполнения работы программную модель можно использовать как экспериментальный макет для изучения методов трансляции описаний учебных языков высокого уровня.

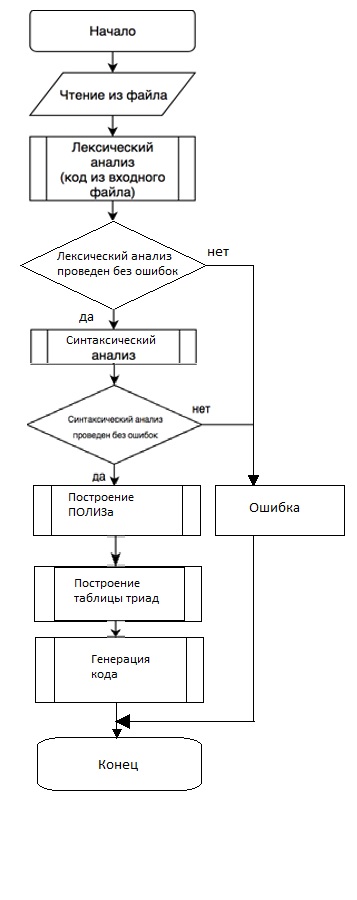
# БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Методические указания по дисциплине «Системное программное обеспечение» для студентов всех форм обучения / Сост. Г.Г.Сергеев, С.Н. Фисун. – Севастополь: Изд-во СевГУ, 2016. – 46с.

2. Лебедев В.Н. Введение в системы программирования – М.: Статистика, 1975. - 312с.

3. Альфред В. Ахо, Моника С. Лам, Рави Сети, Джеффри Д. Ульман Компиляторы: принципы, технологии и инструментарий, 2 издание = Compilers: Principles, Techniques, andTools. — 2 изд. — М.: [Вильямс](http://ru.wikipedia.org/w/index.php?title=Вильямс_(издательство)&action=edit&redlink=1), 2008. — [ISBN 978-5-8459-1349-4](http://ru.wikipedia.org/wiki/Служебная:BookSources/9785845913494).

# ПРИЛОЖЕНИЕ А



Изм.

Лист

№ докум.

Подпись

Дата

Лист

1

СевГУ.ИВТб-32о.В-17

Разраб.

Теличко

Провер.

Фисун С.Н.

Реценз.

Н. Контр.

Утверд.

ПРИЛОЖЕНИЕ А

Схема транслятора

Лит.

Листов

1

ИВТб – 32о

# ПРИЛОЖЕНИЕ Б – Таблица лексем

|  |  |  |
| --- | --- | --- |
| Номер правила | Запись в грамматике | Запись в управляющей таблице |
| 1 | <Программа> | <П> |
| 2 | < посл. описаний > | <ПО> |
| 3 | < A > | <A> |
| 4 | < B > | <B> |
| 5 | < описание > | <ОП> |
| 6 | < список идент. > | <СИД> |
| 7 | < C > | <C> |
| 8 | < оператор > | <ОПЕР> |
| 9 | <выражение> | <ВЫР> |
| 10 | <G> | <G> |
| 11 | <фактор > | <ФАКТ> |
| 12 | <E> | <E> |
| 13 | < первичное > | <П> |
| 14 | <F> | <F> |
| 15 | < сравнение > | <СР> |
| 16 | <J> | <J> |
| 17 | BEGIN | { |
| 18 | DO | { |
| 19 | END | } |
| 20 | ; | ; |
| 21 | := | **=** |
| 22 | while | ww |
| 23 | - | - |
| 24 | + | + |
| 25 | \* | \* |
| 26 | DIV | d |
| 27 | MOD | m |
| 28 | INT | n |
| 29 | , | , |
| 30 | OUTPUT | o |
| 31 | INPUT | r |
| 32 | == | : |
| 33 | <> | ! |
| 34 | . | . |
| 35 | ( | ( |
| 36 | ) | ) |
| 37 | идент | I |
| 38 | число | C |

# ПРИЛОЖЕНИЕ В – Управляющая таблица

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | { | } | ; | | | = | W | | | | - | + | | \* | d | m | | n | , | o | r | : | ! | . | ( | ) | | I | | E | | C | |
| <П> | 1 | 1 |  | | |  |  | | | |  |  | |  |  |  | |  |  |  |  |  |  | 1 |  |  | |  | |  | |  | |
| <ПО> | 2 | 2 | 2 | | | 2 | 2 | | | | 2 | 2 | | 2 | 2 | 2 | | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | | 2 | | 2 | | 2 | |
| <A> |  |  | 3 | | |  |  | | | |  |  | |  |  |  | |  |  |  |  |  |  |  |  |  | |  | | 4 | |  | |
| <ПОП> | 5 | 5 | 5 | | | 5 | 5 | | | | 5 | 5 | | 5 | 5 | 5 | | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | | 5 | | 5 | | 5 | |
| <B> |  |  | 6 | | |  |  | | | |  |  | |  |  |  | |  |  |  |  |  |  |  |  |  | |  | | 7 | |  | |
| <ОП> |  |  |  | | |  |  | | | |  |  | |  |  |  | | 8 |  |  |  |  |  |  |  |  | |  | |  | |  | |
| <СИД> |  |  |  | | |  |  | | | |  |  | |  |  |  | |  |  |  |  |  |  |  |  |  | | 9 | |  | |  | |
| <C> |  |  |  | | |  |  | | | |  |  | |  |  |  | |  | 10 |  |  |  |  |  |  |  | |  | | 11 | |  | |
| <ОПЕР> | 13 | 13 |  | | |  | 13 | | | |  |  | |  |  |  | |  |  | 15 | 14 |  |  |  |  |  | | 12 | |  | |  | |
| <ВЫР> | 16 | 16 | 16 | | | 16 | 16 | | | | 16 | 16 | | 16 | 16 | 16 | | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | | 16 | | 16 | | 16 | |
| <G> |  |  |  | | |  |  | | | | 18 | 17 | |  |  |  | |  |  |  |  |  |  |  |  |  | |  | | 19 | |  | |
| <ФАКТ> | 20 | 20 | 20 | | | 20 | 20 | | | | 20 | 20 | | 20 | 20 | 20 | | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | | 20 | | 20 | | 20 | |
| <E> |  |  |  | | |  |  | | | |  |  | | 21 | 22 | 23 | |  |  |  |  |  |  |  |  |  | |  | | 24 | |  | |
| <П> |  |  |  | | |  |  | | | | 26 |  | |  |  |  | |  |  |  |  |  |  |  | 27 |  | | 25 | |  | | 26 | |
| <F> | 28 | 28 | 28 | | | 28 | 28 | | | | 28 | 28 | | 28 | 28 | 28 | | 28 | 28 | 28 | 28 | 28 | 28 | 28 | 28 | 28 | | 29 | | 28 | | 28 | |
| <СР> | 30 | 30 | 30 | | | 30 | 30 | | | | 30 | 30 | | 30 | 30 | 30 | | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | | 30 | | 30 | | 30 | |
| <J> |  |  |  | | |  |  | | | |  |  | |  |  |  | |  |  |  |  | 31 | 32 |  |  |  | |  | | 33 | |  | |
| { | c |  |  | | |  |  | | | |  |  | |  |  |  | |  |  |  |  |  |  |  |  |  | |  | |  | |  | |
| } |  | c | |  |  | | |  | |  | | |  |  |  | |  |  |  |  |  |  |  |  |  | |  | |  | |  | |  |
| ; |  |  | | c |  | | |  | |  | | |  |  |  | |  |  |  |  |  |  |  |  |  | |  | |  | |  | |  |
| **=** |  |  | |  | c | | |  | |  | | |  |  |  | |  |  |  |  |  |  |  |  |  | |  | |  | |  | |  |
| w |  |  | |  |  | | | c | |  | | |  |  |  | |  |  |  |  |  |  |  |  |  | |  | |  | |  | |  |
| - |  |  | |  |  | | |  | | c | | |  |  |  | |  |  |  |  |  |  |  |  |  | |  | |  | |  | |  |
| + |  |  | |  |  | | |  | |  | | | c |  |  | |  |  |  |  |  |  |  |  |  | |  | |  | |  | |  |
| \* |  |  | |  |  | | |  | |  | | |  | c |  | |  |  |  |  |  |  |  |  |  | |  | |  | |  | |  |
| d |  |  | |  |  | | |  | |  | | |  |  | c | |  |  |  |  |  |  |  |  |  | |  | |  | |  | |  |
| m |  |  | |  |  | | |  | |  | | |  |  |  | | c |  |  |  |  |  |  |  |  | |  | |  | |  | |  |
| n |  |  | |  |  | | |  | |  | | |  |  |  | |  | c |  |  |  |  |  |  |  | |  | |  | |  | |  |
| , |  |  | |  |  | | |  | |  | | |  |  |  | |  |  | c |  |  |  |  |  |  | |  | |  | |  | |  |
| o |  |  | |  |  | | |  | |  | | |  |  |  | |  |  |  | c |  |  |  |  |  | |  | |  | |  | |  |
| r |  |  | |  |  | | |  |  | | | |  |  |  | |  |  |  |  | c |  |  |  |  | |  | |  | |  | |  |
| : |  |  | |  |  | | |  |  | | | |  |  |  | |  |  |  |  |  | c |  |  |  | |  | |  | |  | |  |
| ! |  |  | |  |  | | |  |  | | | |  |  |  | |  |  |  |  |  |  | c |  |  | |  | |  | |  | |  |
| . |  |  | |  |  | | |  |  | | | |  |  |  | |  |  |  |  |  |  |  | c |  | |  | |  | |  | |  |
| ( |  |  | |  |  | | |  |  | | | |  |  |  | |  |  |  |  |  |  |  |  | c | |  | |  | |  | |  |
| ) |  |  | |  |  | | |  |  | | | |  |  |  | |  |  |  |  |  |  |  |  |  | | c | |  | |  | |  |
| I |  |  | |  |  | | |  |  | | | |  |  |  | |  |  |  |  |  |  |  |  |  | |  | | c | |  | |  |
| E |  |  | |  |  | | |  |  | | | |  |  |  | |  |  |  |  |  |  |  |  |  | |  | |  | | c | |  |
| C |  |  | |  |  | | |  |  | | | |  |  |  | |  |  |  |  |  |  |  |  |  | |  | |  | |  | | c |

# ПРИЛОЖЕНИЕ Д – Текст программы

**import** java.io.\*;  
**import** java.nio.charset.Charset;  
**import** java.nio.file.\*;  
**import** java.util.ArrayList;  
**import** java.util.List;  
**import** java.util.Scanner;  
**public class** Main {  
 **public static void** main(String[] args) **throws** FileNotFoundException {  
 Scanner scan = **new** Scanner(**new** File(**"input.txt"**));  
 String input=**""**;  
 **try** {  
 input = scan.useDelimiter(**"\\A"**).next();  
 } **catch** (Exception e){  
 System.***out***.println(**"Ошибка, использованы кириллические символы"**);  
 }  
 scan.close();  
*// Игнорируем регистр* input = input.toUpperCase();  
 input = input.replaceAll(**"[\r|\n]"**, **" "**);  
 System.***out***.println(**"Лексический анализ"**);  
 String lexemStr = LA.*analyze*(input);  
 **if** (lexemStr.equals(**""**)) {  
 System.***out***.println(**"Ошибка на этапе лексического анализа."**);  
 **return**;  
 }  
 System.***out***.println(**"Получена кодированная строка лексем:"**);  
 System.***out***.println(lexemStr);  
 System.***out***.println();  
 System.***out***.println(**"Синтаксический анализ"**);  
 **if** (SA.*analyze*(lexemStr + **"$"**)) {  
 System.***out***.println(**"\nДопуск"**);  
 String c = Poliz.*buildPoliz*(lexemStr.substring(0, lexemStr.length()-1));  
 System.***out***.println(c);  
 String tt=TT.*BuildTriads*(c);  
 System.***out***.println(tt);  
 System.***out***.println(Assembling.*GetAsm*(tt));  
 } **else** {  
 System.***out***.println(**"Ошибка на этапе синтаксического анализа."**);  
 }  
 }  
}

**import** java.util.ArrayList;  
**import** java.util.regex.Matcher;  
**import** java.util.regex.Pattern;  
**public class** LA {  
 **static** ArrayList<String> *idTable* = **new** ArrayList<String>(); *// Таблица имён* **static** ArrayList<String> *nTable* = **new** ArrayList<String>(); *// Таблица числовых констант  
// Таблица лексем* **static final** String[] ***lexemTable*** = { **"BEGIN"**, **"DO"**,**"END"**, **";"**, **":="**, **"WHILE"**, **"-"**, **"+"**, **"\*"**, **"DIV"**, **"MOD"**,**"INT"**,**","**,**"OUTPUT"**,**"INPUT"**,**"=="**,**"<>"**,**"."**,**"("**,**")"**};  
*// Таблица КОДОВ лексем* **static final** String[] ***codedTable*** = { **"{"**,**"{"**, **"}"**, **";"**, **"="**, **"w"**, **"-"**, **"+"**, **"\*"**,**"d"**,**"m"**,**"n"**,**","**,**"o"**,**"r"**,**":"**, **"!"**,**"."**,**"("**,**")"**};  
  
*//Анализ программы* **public static** String analyze(String input) {  
 **boolean** gotErrors = **false**;  
 System.***out***.println(**"Встреченные лексемы"**);  
 String outStr = **""**;  
 **while** (input.length() > 0) {  
 **if** (input.startsWith(**" "**)) {  
 input = input.substring(1);  
 **continue**;  
 }  
 **if** (input.startsWith(**":="**)) {  
 System.***out***.println(**"L4\t:=\t="**);  
 outStr += **'='**;  
 input = input.substring(2);  
 **continue**;  
 }  
 **int** wordInd = input.indexOf(**' '**); *// конец слова  
// текущее слово* String currentWord = wordInd > -1 ? input.substring(0, wordInd) : input;  
  
*// Поиск лексемы в таблице* **boolean** found = **false**;  
 **for** (**int** i = 0; i < ***lexemTable***.**length**; i++) {  
 **if** (***lexemTable***[i].equals(currentWord) ||  
 ***lexemTable***[i].equals(input.charAt(0) + **""**)) {  
 System.***out***.print(**"L"** + (i+1));  
 System.***out***.print(**"\t"** + ***lexemTable***[i]);  
 System.***out***.println(**"\t"** + ***codedTable***[i]);  
 outStr += ***codedTable***[i];  
 input = input.substring(***lexemTable***[i].length());  
 found = **true**;  
 **break**;  
 }  
 }  
 **if** (found)  
 **continue**;  
*// Если встретили число любой длины* Matcher m = Pattern.*compile*(**"^(\\d+)(.\*)"**).matcher(input);  
 **if** (m.find()) {  
*// Извленкаем это число из строки* String number = m.group(1);  
*// Добавляем в таблицу констант  
 addToTable*(*nTable*, number);  
 System.***out***.print(**"C-"**);  
 System.***out***.print(*nTable*.indexOf(number));  
 System.***out***.println(**"\t"** + number);  
*// И удаляем его из входной строки* input = m.group(2);  
 outStr += **" "** + number + **" "**;  
 **continue**;  
 }  
*// Если встретили имя переменной* m = Pattern.*compile*(**"^(\\w+)(.\*)"**).matcher(input);  
 **if** (m.find()) {  
*// Извленкаем это имя из строки* String var = m.group(1);  
*// Добавляем в таблицу переменных  
 addToTable*(*idTable*, var);  
 System.***out***.print(**"I-"**);  
 System.***out***.print(*idTable*.indexOf(var));  
 System.***out***.println(**"\t"** + var);  
*// И удаляем её из входной строки* input = m.group(2);  
 outStr += **" "** + var + **" "**;  
 **continue**;  
 }  
 System.***out***.println(**"Ошибка в лексеме "** + currentWord);  
 gotErrors = **true**;  
 input = input.substring(currentWord.length());  
 }  
 **if** (!gotErrors)  
 **return** outStr;  
 **else  
 return ""**;  
 }  
  
 **static void** addToTable(ArrayList<String> table, String item) {  
 **if** (!table.contains(item)) {  
 table.add(item);  
 }  
 }  
}

**import** java.io.File;  
**import** java.io.FileNotFoundException;  
**import** java.util.Scanner;  
  
**public class** SA {  
 **static** String *input*; *// Входные данные* **static int** *i*; *// Указатель на текущий символ* **public** SA(String input) {  
 **this**.*input* = input;  
 }  
  
 **public static boolean** analyze(String \_input) {  
 *input* = \_input;  
 *input* = *input*.replaceAll(**" [0-9]+ "**, **"C"**);  
 *input* = *input*.replaceAll(**" [A-Z]+ "**, **"i"**);  
 **return** *S*();  
 }  
  
 *// Названия нетерминалов сохранены как в грамматике* **static boolean** S() {  
 *программа*();  
 **if** (*input*.charAt(*i*) == **'$'**) {  
 **return true**;  
 }  
 System.***out***.println(**"\nВстречен преждевременный обрыв файла"**);  
 **return false**;  
 }  
  
 **static void** программа() {  
 **if** (*input*.charAt(*i*) == **'{'**) {  
 *i*++;  
 System.***out***.print(**"1, "**);  
 *описания*();  
 **if** (*input*.charAt(*i*) == **'.'**) {  
 *i*++;  
 *операторы*();  
 } **else** {  
 *error*(**";"**);  
 *i*++;  
 }  
  
 **if** (*input*.charAt(*i*) != **'}'**)  
 *error*(**"}"**);  
 *i*++;  
 } **else** *error*(**"{"**);  
 }  
  
 **static void** описания() {  
 System.***out***.print(**"2, "**);  
 *описание*();  
 *A*();  
 }  
  
 **static void** A() {  
 **if** (*input*.charAt(*i*) == **';'**) {  
 *i*++;  
 System.***out***.print(**"3, "**);  
 *описания*();  
 } **else** {  
 System.***out***.print(**"4, "**);  
 }  
 }  
  
 **static void** операторы() {  
 System.***out***.print(**"5, "**);  
 *оператор*();  
 *B*();  
 }  
  
 **static void** B() {  
 **if** (*input*.charAt(*i*) == **';'**) {  
 *i*++;  
 System.***out***.print(**"6, "**);  
 *операторы*();  
 } **else** {  
 System.***out***.print(**"7, "**);  
 }  
 }  
  
 **static void** описание() {  
 **if** (*input*.charAt(*i*) == **'n'**) {  
 *i*++;  
 System.***out***.print(**"8, "**);  
 *идентификаторы*();  
 } **else** {  
 *error*(**"n"**);  
 *i*++;  
 }  
 }  
  
 **static void** идентификаторы() {  
 **if** (*input*.charAt(*i*) == **'i'**) {  
 *i*++;  
 System.***out***.print(**"9, "**);  
 *C*();  
 } **else** {  
 *error*(**"i"**);  
 *i*++;  
 }  
 }  
  
 **static void** C() {  
 **if** (*input*.charAt(*i*) == **','**) {  
 *i*++;  
 System.***out***.print(**"10, "**);  
 *идентификаторы*();  
 } **else** {  
 System.***out***.print(**"11, "**);  
 }  
 }  
  
 **static void** оператор() {  
 **if** (*input*.charAt(*i*) == **'i'**) {  
 *i*++;  
 **if** (*input*.charAt(*i*) == **'='**) {  
 *i*++;  
 System.***out***.print(**"12, "**);  
 *выражение*();  
 } **else** {  
 *error*(**"="**);  
 *i*++;  
 }  
 } **else if** (*input*.charAt(*i*) == **'w'**) {  
 *i*++;  
 System.***out***.print(**"13, "**);  
 *выражение*();  
 **if** (*input*.charAt(*i*) == **'{'**) {  
 *i*++;  
 *операторы*();  
 } **else** {  
 *error*(**"{"**);  
 *i*++;  
 }  
 **if** (*input*.charAt(*i*) != **'}'**) {  
 *error*(**"{"**);  
 }  
 *i*++;  
 } **else if** (*input*.charAt(*i*) == **'r'**) {  
 *i*++;  
 System.***out***.print(**"14, "**);  
 **if** (*input*.charAt(*i*) != **'i'**) {  
 *error*(**"i"**);  
 *i*++;  
 }  
 } **else if** (*input*.charAt(*i*) == **'o'**) {  
 *i*++;  
 System.***out***.print(**"15, "**);  
 *выражение*();  
 } **else** {  
 *error*(**"i/w/r/o"**);  
 *i*++;  
 }  
 }  
  
 **static void** выражение() {  
 System.***out***.print(**"16, "**);  
 *фактор*();  
 *G*();  
 }  
  
 **static void** G() {  
 **if** (*input*.charAt(*i*) == **'+'**) {  
 *i*++;  
 System.***out***.print(**"17, "**);  
 *выражение*();  
 } **else if** (*input*.charAt(*i*) == **'-'**) {  
 *i*++;  
 System.***out***.print(**"18, "**);  
 *выражение*();  
 } **else** System.***out***.print(**"19, "**);  
 }  
  
 **static void** фактор() {  
 System.***out***.print(**"20, "**);  
 *первичное*();  
 *E*();  
 }  
  
 **static void** E() {  
 **if** (*input*.charAt(*i*) == **'\*'**) {  
 *i*++;  
 System.***out***.print(**"21, "**);  
 *фактор*();  
 } **else if** (*input*.charAt(*i*) == **'d'**) {  
 *i*++;  
 System.***out***.print(**"22, "**);  
 *фактор*();  
 } **else if** (*input*.charAt(*i*) == **'m'**) {  
 *i*++;  
 System.***out***.print(**"23, "**);  
 *фактор*();  
 } **else** {  
 System.***out***.print(**"24, "**);  
 }  
 }  
  
 **static void** первичное() {  
 **if** (*input*.charAt(*i*) == **'i'**) {  
 *i*++;  
 System.***out***.print(**"25, "**);  
 } **else if** (*input*.charAt(*i*) == **'C'**) {  
 *i*++;  
 System.***out***.print(**"26, "**);  
 } **else if** (*input*.charAt(*i*) == **'-'**) {  
 *i*++;  
 System.***out***.print(**"26, "**);  
 **if** (*input*.charAt(*i*) != **'C'**) {  
 *error*(**"C"**);  
 }  
 *i*++;  
 } **else if** (*input*.charAt(*i*) == **'('**) {  
 *i*++;  
 System.***out***.print(**"27, "**);  
 *F*();  
 **if** (!(*input*.charAt(*i*) == **')'**))  
 *error*(**"C"**);  
 *i*++;  
 }  
 }  
  
 **static void** F() {  
 **if** (*input*.charAt(*i*) == **'C'**) {  
 *i*++;  
 System.***out***.print(**"29, "**);  
 } **else** {  
 System.***out***.print(**"28, "**);  
 *сравнение*();  
 }  
 }  
  
 **static void** сравнение() {  
 System.***out***.print(**"30, "**);  
 *выражение*();  
 *J*();  
 }  
  
 **static void** J() {  
 **if** (*input*.charAt(*i*) == **':'**) {  
 *i*++;  
 System.***out***.print(**"31, "**);  
 *выражение*();  
 *J*();  
 } **else if** (*input*.charAt(*i*) == **'!'**) {  
 *i*++;  
 System.***out***.print(**"32, "**);  
 *выражение*();  
 *J*();  
 } **else** System.***out***.print(**"33, "**);  
 }  
  
  
  
  
 **static void** error(String awaited) {  
 System.***out***.println();  
 System.***out***.println(**"Ошибка. Ожидалось: "** + awaited);  
 System.***out***.println(**"Встреченно: "** + *input*.charAt(*i*));  
 System.***out***.println(**"Ошибка возникла в"**);  
 System.***out***.print(*input*.substring(0, *i*));  
 System.***out***.print(**" ["** + *input*.charAt(*i*) + **"] "**);  
 System.***out***.println(*input*.substring(*i* + 1));  
 System.***out***.println(**"----------------"**);  
 }  
}

**import** java.util.Stack;  
  
**public class** Poliz {  
*// Однооперандные команды* **final static char**[] ***singleOp*** = { **'c'**, **'v'**, **'p'**, **'a'**, **'i'**, **'o'** };  
 **static** String *priorityTable*[][] = {  
 { **"("**, **"B"**, **"W"**, **"C"**, **"V"**, **"A"**, **"I"**, **"O"**},  
 { **")"**, **"E"**, **"D"**,**"{"**},  
 { **":"**, **"!"**,**"="**},  
 { **"+"**, **"-"** },  
 { **"\*"**, **"d"**,**"m"**},  
 { **"r"**, **"o"** }  
 };  
  
 **static int** getPriority(String op) {  
 **for** (**int** i = 0; i < *priorityTable*.**length**; i++) {  
 **for** (**int** j = 0; j < *priorityTable*[i].**length**; j++) {  
 **if** (op.equalsIgnoreCase(*priorityTable*[i][j]))  
 **return** i;  
 }  
 }  
 **return** -1;  
 }  
  
 **public static** String buildPoliz(String input) {  
 System.***out***.println(**"ПОЛИЗ:"**);  
 input = input.toUpperCase();  
 **return new** Poliz().getPoliz(input);  
 }  
  
 **int pointerNumber** = 0; *// Указатель для циклов и условий* **static** Stack<String> *workSt*; *// Рабочий стек* **static** String *output*; *//выход стр* **static** String *input*; *//выход стр* **static int** *i*=0; *//указатель на текущий символ  
  
//Вывод имени переменной в выходную стр.* **void** nameToOut() {  
 String name = *input*.substring(*i*-1).split(**" "**)[1];  
*//Переменные выводятся сразу в выводную стр.  
 output* += name + **" "**;  
*//Сдвиг указателя на длину имени и два пробела  
 i* += name.length() + 1;  
 }  
  
*// Вытолкнуть всё исключительно* **void** popUntil(String symbol) {  
  
 **while** (!*workSt*.isEmpty() && !*workSt*.peek().equals(symbol)) {  
 String p = *workSt*.pop();  
 **if** (!p.equals(**"W"**) && !p.equals(**"F"**))  
 *output* += p + **" "**;  
 }  
 }  
  
*// Вытолкнуть для точки с зяпятой* **void** popSemicolum() {  
 **while** (!*workSt*.isEmpty()  
 && !*workSt*.peek().equals(**"("**)  
 && !*workSt*.peek().equals(**"B"**)  
 && !*workSt*.peek().equals(**"W"**)  
 && !*workSt*.peek().equals(**"F"**)) {  
 *output* += *workSt*.pop() + **" "**;  
 }  
 **if** (*workSt*.isEmpty())  
 **return**;  
*//полиз циклов* String top = *workSt*.peek();  
 **if** (top.equals(**"W"**)) {  
 **pointerNumber**--;  
 *output* += **"M"** + **pointerNumber** + **" БП "**;  
 **pointerNumber**++;  
 *output* += **"M"** + **pointerNumber** + **": "**;  
 **pointerNumber**--;  
 *workSt*.pop();  
 }  
 **else if** (top.equals(**"F"**)) {  
 *output* += **"M"** + **pointerNumber** + **": "**;  
 **pointerNumber**--;  
 *workSt*.pop();  
 }  
 **else if** (isSingleOp(top)) {  
 *output* += top;  
 *workSt*.pop();  
 }  
 }  
  
*// Вытолкнуть для зяпятой* **void** popComa() {  
 **while** (!*workSt*.isEmpty()  
 && !*workSt*.peek().equals(**"C"**)  
 && !*workSt*.peek().equals(**"V"**)  
 && !*workSt*.peek().equals(**"I"**)  
 ) {  
 *output* += *workSt*.pop() + **" "**;  
 }  
 **if** (!*workSt*.isEmpty()) {  
 *output* += *workSt*.peek() + **" "**;  
 }  
 }  
  
 **boolean** isSingleOp(String op) {  
 **for** (**int** i = 0; i < ***singleOp***.**length**; i++) {  
 **if** (op.equalsIgnoreCase(***singleOp***[i] + **""**))  
 **return true**;  
 }  
 **return false**;  
 }  
  
*// Основной метод - ввод: выражения, вывод: полиз* **private** String getPoliz(String \_input) {  
 *input* = \_input.substring(\_input.indexOf(**"."**)+1);  
 System.***out***.println(**"Исходная строка:"**);  
 System.***out***.println(*input*);  
 System.***out***.println(**"Результат:"**);  
 **if** (!*input*.endsWith(**"."**)) {  
 *input* += **"."**;  
 }  
 *workSt* = **new** Stack<String>(); *// Рабочий стек  
 output* = **""**;  
 **while** (*i* < *input*.length()) {  
 String c = *input*.charAt(*i*++) + **""**;  
  
 **if** ( (c.equals(**"="**) && *input*.charAt(*i*) == **'='**)  
 || c.equals(**"<"**) || c.equals(**">"**)) {  
 c += *input*.charAt(*i*++);  
 }  
 **if** (c.equals(**"$"**)) {  
 **continue**;  
 }  
 **if** (c.equals(**"."**)) {  
 popSemicolum();  
 **return** *output*;  
 }  
*// Пробел - признак начала имени переменной* **if** (c.equals(**" "**)) {  
 nameToOut();  
 **continue**;  
 }  
*// Особенности обработки скобок* **if** (c.equals(**")"**)) {  
 popUntil(**"("**);  
 *workSt*.pop();  
 **continue**;  
 }  
*// Особенности обработки скобок* **if** (c.equals(**"E"**)) {  
 popUntil(**"B"**);  
 *workSt*.pop();  
 **continue**;  
 }  
*// Особенности обработки точки с запятой* **if** (c.equals(**";"**)) {  
 popSemicolum();  
 **continue**;  
 }  
*// Особенности обработки запятой* **if** (c.equals(**","**)) {  
 popComa();  
 **continue**;  
 }  
 **if** (c.equals(**"W"**)) {  
 \_while();  
 **continue**;  
 } **else if** (c.equals(**"{"**)) {  
 \_do();  
 **continue**;  
 } **else if** (c.equals(**"}"**)) {  
 \_end();  
 **continue**;  
 }  
 **int** priority = *getPriority*(c);  
*// Если приоритет входного знака равен нулю или больше приоритета знака,  
// находящегося в вершине стека, то новый знак добавляется к вершине стека* **if** (priority == 0 || (!*workSt*.isEmpty() && priority > *getPriority*(*workSt*.peek()))) {  
 *workSt*.push(c);  
 **continue**;  
 }  
*// В противном случае из стека выталкивается и  
// переписывается в выходную строку знак, находящийся в вершине, а также  
// следующие за ним знаки с приоритетами большими или равными приоритету  
// входного знака. После этого входной знак добавляется к вершине стека.* **else** {  
 **while** (!*workSt*.isEmpty() && *getPriority*(*workSt*.peek()) >= priority) {  
 *output* += *workSt*.pop()+**" "**;  
 }  
 *workSt*.push(c);  
 **continue**;  
 }  
*// Так говорил Лебедев.* }  
 **return ""**;  
 }  
  
 **void** \_while() {  
 **pointerNumber**++;  
 *output* += **" M"** + **pointerNumber** + **" "**;  
 *workSt*.push(**"УЛП"**);  
 }  
  
 **void** \_do() {  
 popUntil(**"УЛП"**);  
 *output* +=*workSt*.peek()+**" "**;  
*//pointerNumber--;* }  
  
 **void** \_end(){  
 popUntil(**"УЛП"**);  
 *workSt*.pop();  
 *output* +=**"W1 БП"**;  
 }  
  
 **void** \_if() {  
 *workSt*.push(**"F"**);  
 }  
  
 **void** \_then() {  
 **pointerNumber**++;  
 *output* += **"M"** + **pointerNumber** + **" УПЛ "**;  
 popUntil(**"F"**);  
 }  
}

**import** java.util.Stack;  
  
**public class** TT {  
 **final static** String[] ***singleOp*** = {**"БП"**, **"O"**, **"R"**};  
 **final static** String[] ***operators*** = {**"\*"**, **"D"**, **"M"**, **"+"**, **"-"**, **":"**, **"!"**, **"="**, **"УЛП"**, **"БП"**, **"O"**, **"R"**};  
 **static** String *triadTable* = **""**; *//таблица триад* **static int** *triN* = 0;*//номер триады* **static** Stack<String> *st* = **new** Stack<>();  
  
 **static boolean** IsOperator(String s) {  
 **for** (**int** i = 0; i < ***operators***.**length**; i++) {  
 **if** (***operators***[i].equals(s))  
 **return true**;  
 }  
 **return false**;  
 }  
  
 **static boolean** IsSingleOp(String s) {  
 **for** (**int** i = 0; i < ***singleOp***.**length**; i++) {  
 **if** (s.equals(***singleOp***[i] + **""**))  
 **return true**;  
 }  
 **return false**;  
 }  
  
 **public static** String BuildTriads(String input) {  
 String[] ops = input.split(**" "**);  
 **for** (**int** i = 0; i < ops.**length**; i++) {  
 **if** (*IsOperator*(ops[i])) {  
 String triad = **"T"** + *triN* + **" "** +ops[i]+ **" "** + *st*.pop();  
 **if** (!*IsSingleOp*(ops[i])) {  
 triad += **" "** + *st*.pop();  
 }  
 *st*.push(**"T"** + *triN*);  
 *triadTable* += triad + **"\n"**;  
 *triN*++;  
 } **else** *st*.push(ops[i]);  
 }  
 **return** *triadTable*;  
 }  
}

**import** java.util.regex.Pattern;  
  
**public class** Assembling {  
 **static** String *asm* = **""**;  
 **static** String[] *tab*;  
 **static** String[] *str*;  
 **static** String *ja* = **""**;  
  
 **public static** String GetAsm(String table) {  
 *tab* = table.split(**"\n"**);  
 String regexp = **"T[0-9]{1,}"**;  
 String t = **" "**;  
 **boolean** fl = **false**;  
 **for** (**int** i = 0; i < *tab*.**length**; i++) {  
 *str* = *tab*[i].split(**" "**);  
 **if** (*str*.**length** == 4) {  
 **if** ((Pattern.*matches*(regexp, *str*[3]) && Pattern.*matches*(regexp, *str*[2])) & !*str*[1].equals(**":"**) & !*str*[1].equals(**"!"**)) {  
 t += *str*[3] + **" "**;  
 }  
 }  
 }  
 **for** (**int** i = 0; i < *tab*.**length**; i++) {  
 *str* = *tab*[i].split(**" "**);  
 **if** (*str*[1].equals(**"!"**) | *str*[1].equals(**":"**)) {  
 *cons*(*str*);  
 **if** (!fl) {  
 fl = **true**;  
 **int** j = i;  
 String[] str1 = *str*;  
 **while** ((j < *tab*.**length**) && !Pattern.*matches*(**"M[0-9]{1,}"**, str1[3])) {  
 j++;  
 str1 = *tab*[j].split(**" "**);  
 }  
 **if** (Pattern.*matches*(**"M[0-9]{1,}"**, str1[3])) {  
 *asm* += str1[3].replace(**"M"**, **"W"**) + **":\n"**;  
 *ja* = str1[3];  
 }  
 }  
 }  
 **boolean** op1 = **false**;  
 **if** (*str*.**length** == 4) op1 = Pattern.*matches*(regexp, *str*[3]);  
 **boolean** op2 = Pattern.*matches*(regexp, *str*[2]);  
 **if** (*str*[1].equals(**"УЛП"**)) {  
 fl = **false**;  
 **continue**;  
 } **else if** (*str*[1].equals(**"БП"**)) {  
 *asm* += **"JMP "** + *str*[2] + **"\n"** + *str*[2].replace(**"W"**, **"M"**) + **":"**;  
 **continue**;  
 } **else if** ((!op1) & (!op2)) {  
 *asm* += **"MOV AX, "** + *str*[3] + **"\n"** + *act*(*str*[1], *str*[2]);  
 } **else if** ((!op1) && (op2)) {  
 **if** (*str*[1].equals(**"D"**) || *str*[1].equals(**"M"**) || *str*[1].equals(**"-"**)) {  
 *asm* += **"MOV DX, AX\nMOV AX, "** + *str*[3] + **"\n"** + *act*(*str*[1], **"DX"**);  
 } **else** {  
 *asm* += *act*(*str*[1], *str*[3]);  
 }  
 } **else if** ((op1) & (!op2)) {  
 *asm* += *act*(*str*[1], *str*[2]);  
 } **else** {  
 *asm* += **"MOV DX, AX\nPOP AX\n"** + *act*(*str*[1], **"DX"**);  
 }  
 **if** (t.contains(**" "** + *str*[0] + **" "**)) {  
 *asm* += **"PUSH AX\n"**;  
  
 }  
  
 }  
  
 **return** *asm*;  
 }  
  
 **static** String act(String act, String op2) {  
 *//{"\*", "D", "M", "+", "-", ":", "!", "=", "O", "R"}* String str = **""**;  
 **if** (act.equals(**"\*"**)) {  
 **if** (Pattern.*matches*(**"[A-Z]{1,}"**, op2)) {  
 str = **"MUL "** + op2 + **"\n"**;  
 } **else** {  
 str += **"MOV BL, "** + op2 + **"\nMUL BL\n"**;  
 }  
 } **else if** (act.equals(**"D"**)) {  
 **if** (Pattern.*matches*(**"[A-Z]{1,}"**, op2)) {  
 str += **"DIV"** + op2 + **"\n"**;  
 } **else** {  
 str += **"MOV bl, "** + op2 + **"\nDIV BL\n"**;  
 }  
 } **else if** (act.equals(**"M"**)) {  
 str = **"MOV CX,AX\n "**;  
 **if** (Pattern.*matches*(**"[A-Z]{1,}"**, op2)) {  
 str += **"DIV"** + op2 + **"\n"**;  
 } **else** {  
 str += **"MOV BL, "** + op2 + **"\nDIV BL\n"**;  
 }  
 str += **"MUL "** + op2 + **"\nMOV BX, AX\nMOV AX, CXSUB AX, BX\n"**;  
 str += **"MUL "** + op2 + **"\nMOV BX, AX\nMOV AX, CXSUB AX, BX\n"**;  
 } **else if** (act.equals(**"+"**)) {  
 str = **"ADD AX, "** + op2 + **"\n"**;  
 } **else if** (act.equals(**"-"**)) {  
 str = **"SUB AX, "** + op2 + **"\n"**;  
 } **else if** (act.equals(**":"**)) {  
 str = **"CMP AX, "** + op2 + **"\nJNE "** + *ja* + **"\n"**;  
 } **else if** (act.equals(**"!"**)) {  
 str = **"CMP AX, "** + op2 + **"\nJE "** + *ja* + **"\n"**;  
 } **else if** (act.equals(**"="**)) {  
 str = **"MOV "** + op2 + **", AX\n"**;  
 } **else if** (act.equals(**"R"**)) {  
 str = **"INPUT "** + op2 + **"\n"**;  
 } **else if** (act.equals(**"O"**)) {  
 str = **"PUSH AX\nMOV "** + op2 + **", AX\nMOV DL, AH\nMOV AH, 2\nINT 21H\nMOV DL, AL\nINT 21H\n"**;  
 }  
  
 **return** str;  
 }  
  
 **static** String cons(String[] str) {  
 **if** ((str[1].equals(**"!"**) || str[1].equals(**":"**)) & (Pattern.*matches*(**"T[0-9]{1,}"**, str[2]) || Pattern.*matches*(**"T[0-9]{1,}"**, str[3]))) {  
 str[3] = *cons*(*tab*[Integer.*parseInt*(str[3].substring(1))].split(**" "**));  
 }  
 **return** str[2];  
 }  
}