**Министерство науки и высшего образования Российской Федерации**

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Владимирский государственный университет

имени Александра Григорьевича и Николая Григорьевича Столетовых» (ВлГУ)

**РАЗРАБОТКА КОМПИЛЯТОРА ПОДМНОЖЕСТВА ПРОЦЕДУРНОГО ЯЗЫКА В АССЕМБЛЕР**

Пояснительная записка

На 16 листах

|  |  |  |
| --- | --- | --- |
| Руководитель |  | к.т.н. доцент кафедры ИЗИ  Ю.М. Монахов |
| Исполнитель |  | студент гр. ИСБ-118  А.С. Емельянов |

**Владимир 2021**

**Оглавление**

[**Аннотация** 3](#_Toc72672314)

[**1. ПРОЕКТИРОВАНИЕ КОМПИЛЯТОРА** 4](#_Toc72672315)

[1.1 Основные требования 4](#_Toc72672316)

[1.2 Лексический и синтаксический анализатор 5](#_Toc72672317)

[1.3 Построение генератора объектного кода 9](#_Toc72672318)

[**2. ПРОВЕРКА НА СООТВЕТСТВИЕ ОСНОВНЫМ ТРЕБОВАНИЯМ** 10](#_Toc72672319)

[**3. СПИСОК ИСПОЛЬЗОВАННЫХ МАТЕРИАЛОВ** 12](#_Toc72672320)

[**ПРИЛОЖЕНИЕ А** 14](#_Toc72672321)

# **Аннотация**

В данном документе приведён текст компилятора подмножества процедурно-ориентированного языка. Компилятор реализован на языке Java с использованием библиотек ANTLR и ASM.

Основная функция компилятора – проверка принадлежности исходной цепочки входному языку и генерация выходной цепочки символов в виде JVM-кода.

Стадии разработки компилятора подмножества процедурного языка в ассемблер:

* построение лексического и синтаксического анализатора;
* построение генератора JVM-кода

# **1. ПРОЕКТИРОВАНИЕ КОМПИЛЯТОРА**

## Основные требования

Разработка будет производиться в соответствии со следующими требованиями:

* Требования к входному языку:

1. Должны присутствовать операторные скобки;
2. Должна игнорироваться индентация программы;
3. Должны поддерживаться комментарии любой длины;
4. Входная программа должна представлять собой единый модуль, но поддерживать вызов функций.

* Требования к операторам:

1. Оператор присваивания;
2. Арифметические операторы;
3. Логические операторы (И, ИЛИ, НЕ);
4. Условный оператор (ЕСЛИ);
5. Оператор цикла (while);
6. Базовый вывод (строковой литерал, переменная);
7. Типы (целочисленный 32 бита, с плавающей запятой 32 бита).

## Лексический и синтаксический анализатор

Лексический анализатор является первой фазой компилятора. Он преобразует входной поток символов в поток токенов. На основе этих токенов строится дерево разбора грамматики.

Грамматика языка реализована с использованием библиотеки ANTLR. Грамматика языка включает в себя элементы из популярных языков, таких как Python, Java и др.

Список зарезервированных слов представлен на рисунке 1:

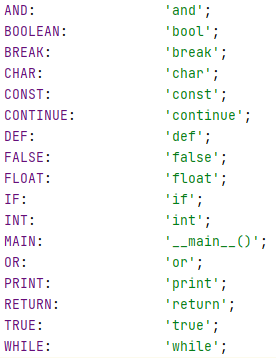


Рисунок 1

Также используются символы : *“{}”, “=”, “>”* “<”, “==”, “!=”,”<=” и т.д.

Дерево разбора строится также встроенными средствами ANTLR

На рисунках 2 и 3 представлены пример входной программы и сгенерированное дерево разбора соответственно.

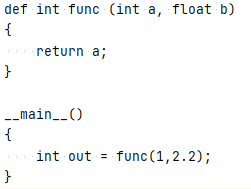


Рисунок 2

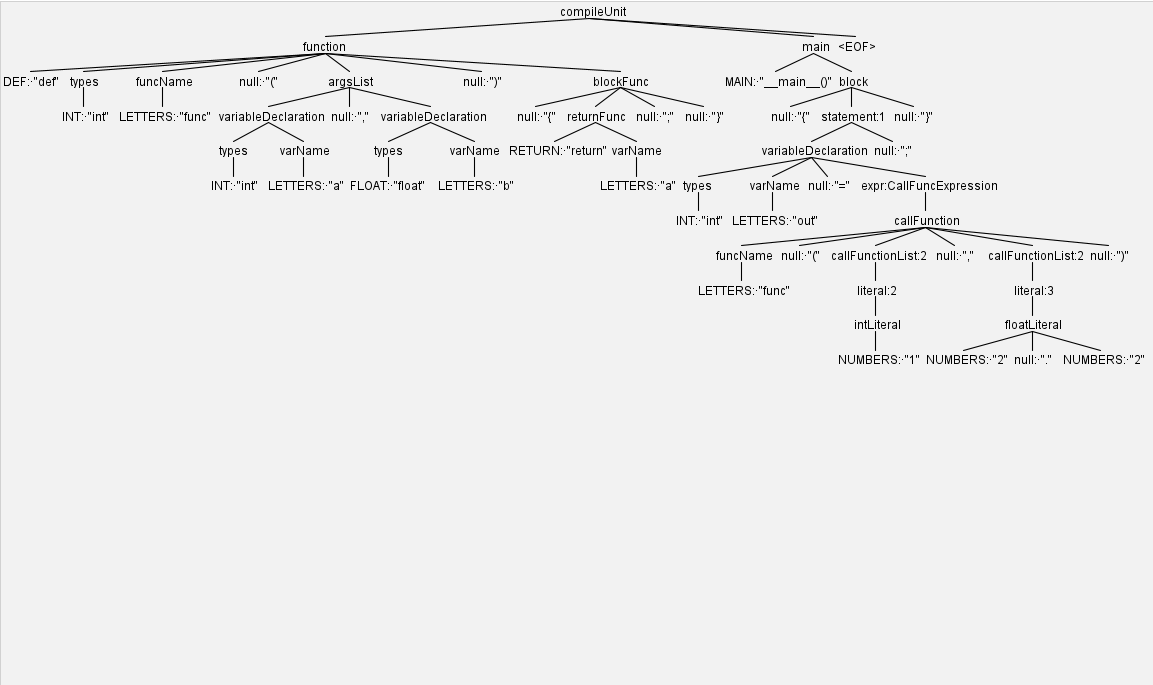


Рисунок 3

**КС-грамматика языка** представлена файлом LangSi.g4, расположенным в каталоге CompilerCourse/src/LangSi/ . Ссылка на репозиторий, содержащий все материалы, будет указана в конце документа.

Правила использования грамматики представлены ниже.

**Объявление функции** происходит посредством указания ключевого слова def, типа возвращаемого значения, названия функции, аргументов, исполняемого блока, включающего ключевое слово return и возвращаемую переменную.

def int func (int a, float b) {…return a}

**Объявление главного блока main** происходит посредством указания ключевого набора символов \_\_main\_\_() и исполняемого блока инструкций.

\_\_main\_\_ () {…}

**Объявление переменной** состоит из указания типа, названия переменной, опциональна инициализация с указанным значением.

int a=1; int b; int c = 3+1;

**Присваивание** происходит посредством указания названия переменной и значения, либо выражения.

a = 2; a = func(a); a = a + a;

**Блок сравнения** определяется ключевым словом if, логическим выражением в скобках, исполняемым блоком инструкций. Если блок находится внутри цикла, возможно применение операторов break и continue, которые реализуют выход из цикла и переход на следующею итерацию соответственно. Возможно сочетание нескольких логических условий посредством операторов or или and.

if (a >b) {…} | if (a >b and a<c) {…} | if (a >b or a<c) {…}

**Блок цикла** определяется аналогично блоку сравнения. Вместо if используется while.

**Базовый вывод** определяется ключевым словом print и указанием одной переменной или одного символа в скобках.

print (a); print(‘a’);

**Вызов определенной функции**  происходит посредством указания названия функции и передачи аргументов в скобках через запятую. Вызов функции возможен только посредством присваивания её значения переменной.

int a = func (a,b);

## Построение генератора объектного кода

Генерация объектного кода выполняется во время обхода дерева в классе VisitImplement, с помощью создания дерева элементов, каждый из которых имеет метод генерации байт-кода. Сформированное дерево передается в управляющий класс Compile, который вызывает метод генерации байт-кода у корня дерева.

Реализовано два главных наследуемых класса Expr – выражения и Stmt – инструкции. Все остальные классы наследуют один из них.

Пример дерева разбора (рисунок 4) для программы:

int a = 0;

a = 3 + 5;

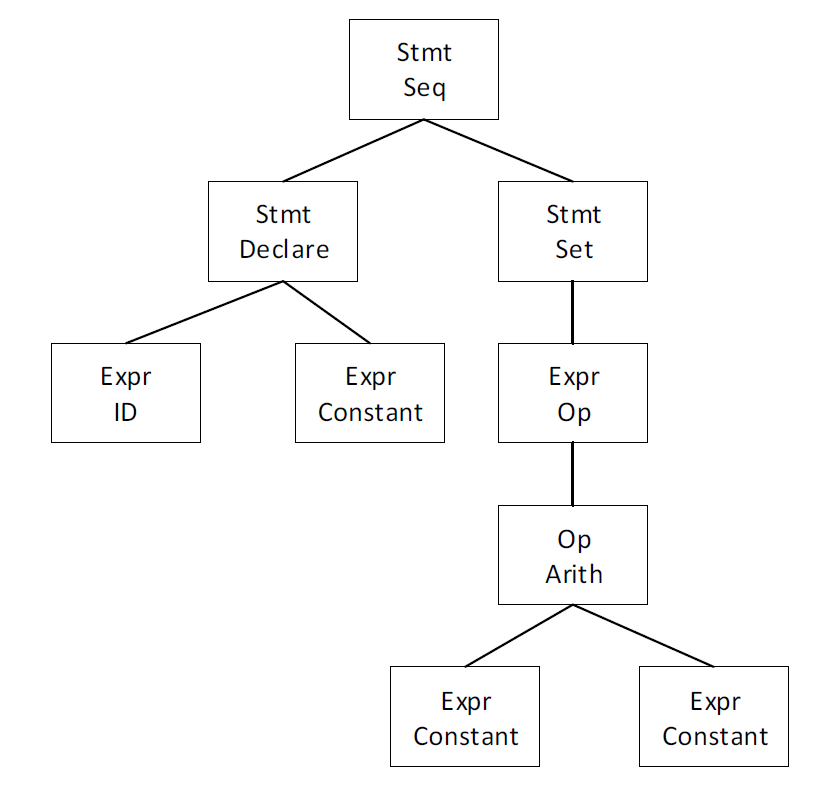


Рисунок 4

# **2. ПРОВЕРКА НА СООТВЕТСТВИЕ ОСНОВНЫМ ТРЕБОВАНИЯМ**

Пример 1 (рисунок 5 и 6) показывает выполнение следующих требований:

* Операторы(присваивания, \*, /, -, +, =, цикла, базового вывода переменной, типы целочисленный и с плавающей запятой)
* Требования ко входному языку (операторные скобки, игнорирование индентации, комментарии любой длины, поддержка вызова функций)

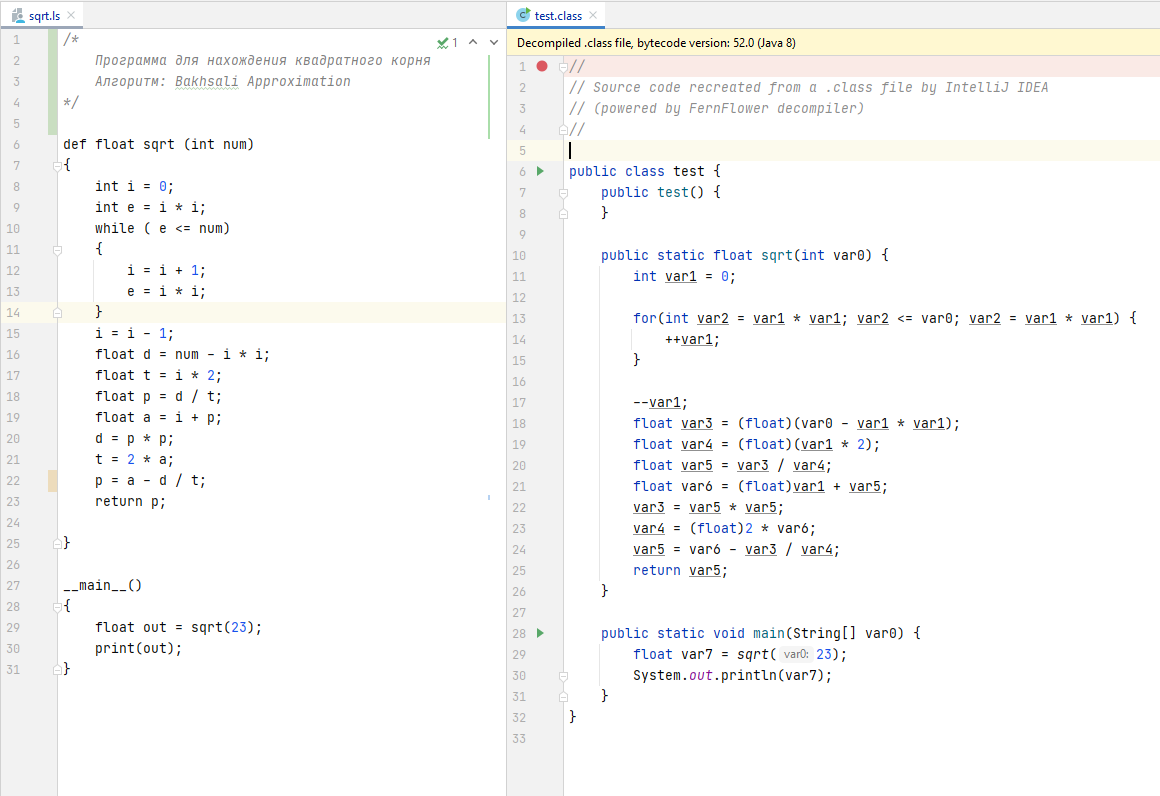


Рисунок 5

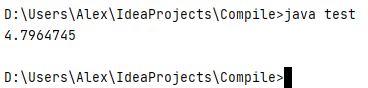


Рисунок 6

Пример 2 (рисунок 7 и 8) показывает выполнение следующих требований, не показанных на предыдущем примере:

* Операторы( <, >, break, continue, базового вывода строкового литерала)
* Требования ко входному языку (операторные скобки, игнорирование индентации, комментарии любой длины, поддержка вызова функций)

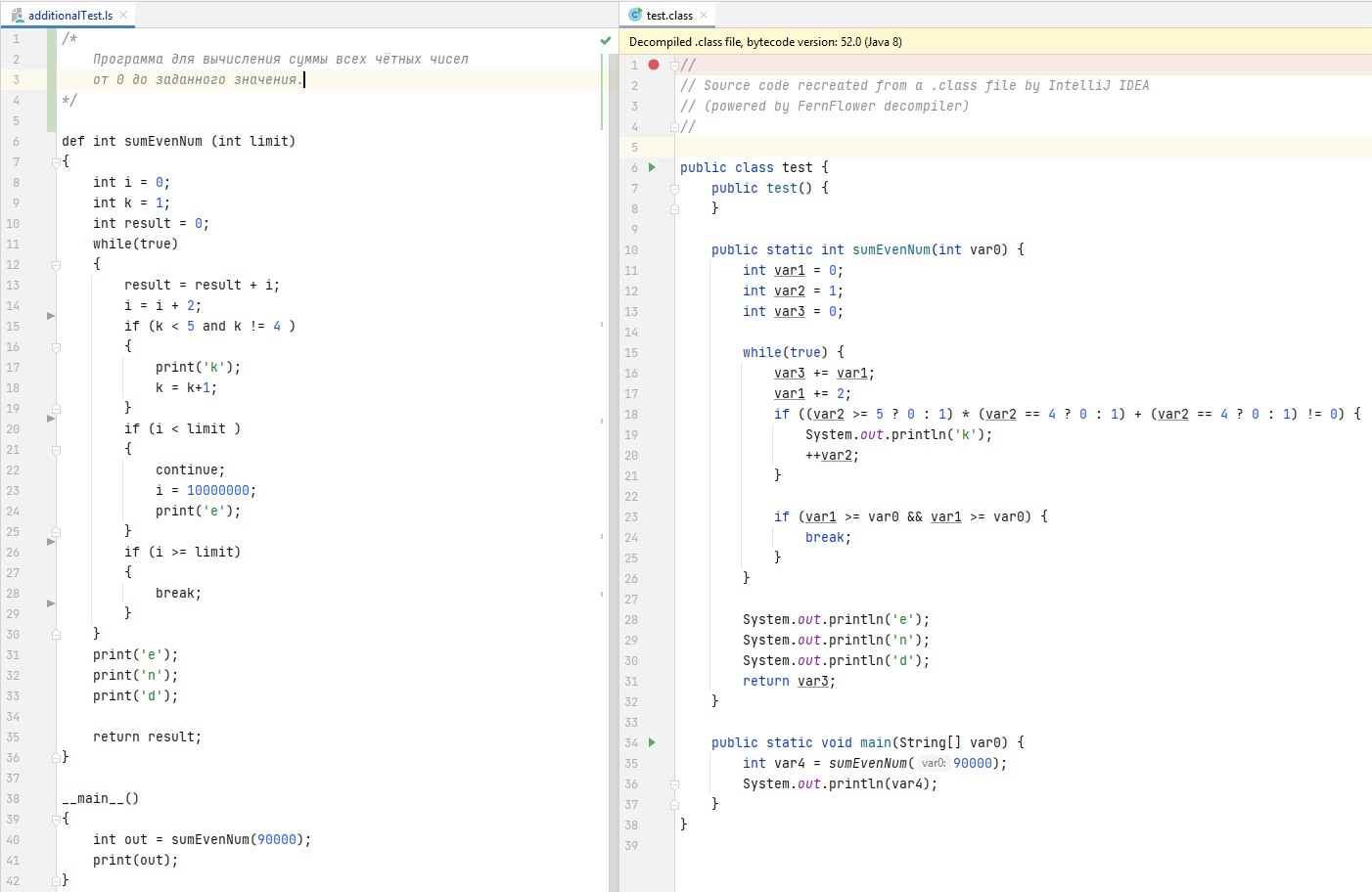


Рисунок 7

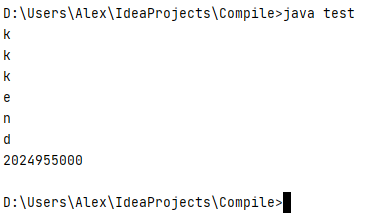


Рисунок 8

# **3. СПИСОК ИСПОЛЬЗОВАННЫХ МАТЕРИАЛОВ**

1. Ульман Д. Д. Компиляторы: принципы, технологии и инструментарий, 2-е изд./ Ульман Д. Д. , Ахо А.В., Лам С.М., Сети Р.-М. :ООО «И.Д. Вильямс», 2008.- 1184 с.
2. ASM userguide // ASM URL: https://asm.ow2.io/asm4-guide.pdf (дата обращения: 21.03.21).
3. The Definitive ANTLR 4 Reference // GitHub URL: https://github.com/joaoBordalo/feup-COMP/blob/master/The%20Definitive%20ANTLR%204%20Reference.pdf (дата обращения: 15.02.21).
4. Компилятор на Java под JVM с использованием библиотек ANTLR и ASM // GitHub URL: https://github.com/irlyk/Compiler (дата обращения: 15.02.21).

Дополнительные примеры в приложении A

Реквизиты к курсовой работе:

<https://github.com/Dertiende/CompilerCourse>

# **ПРИЛОЖЕНИЕ А**

Дополнительные примеры работы компилятора:

Нахождение факториала числа (рисунок 9 и 10):

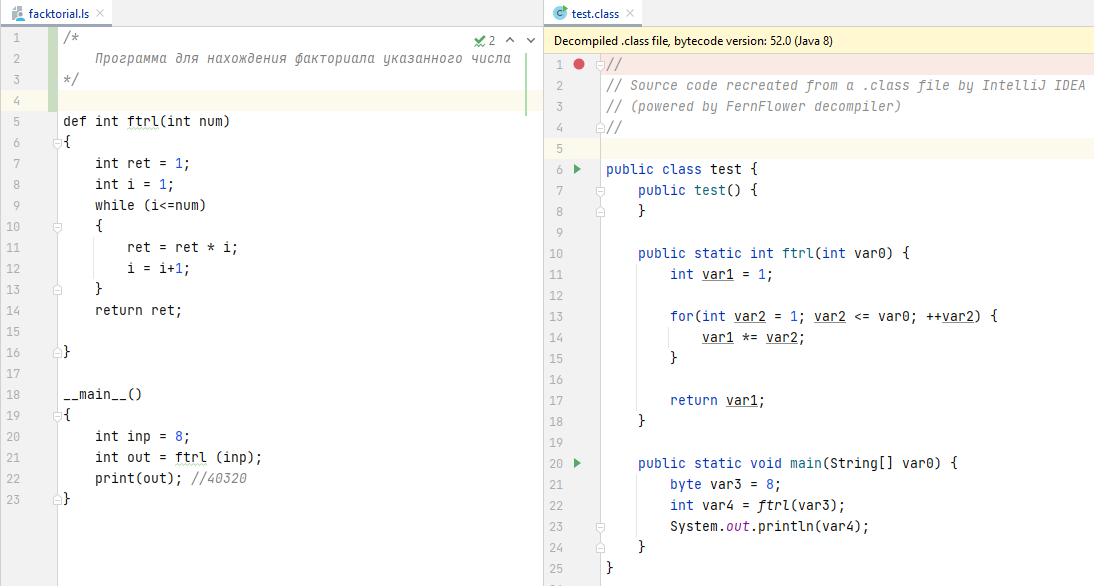


Рисунок 9

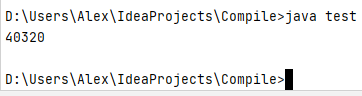


Рисунок 10

Поиск числа из последовательности Фибоначчи с указанным номером (рисунок 11 и 12):

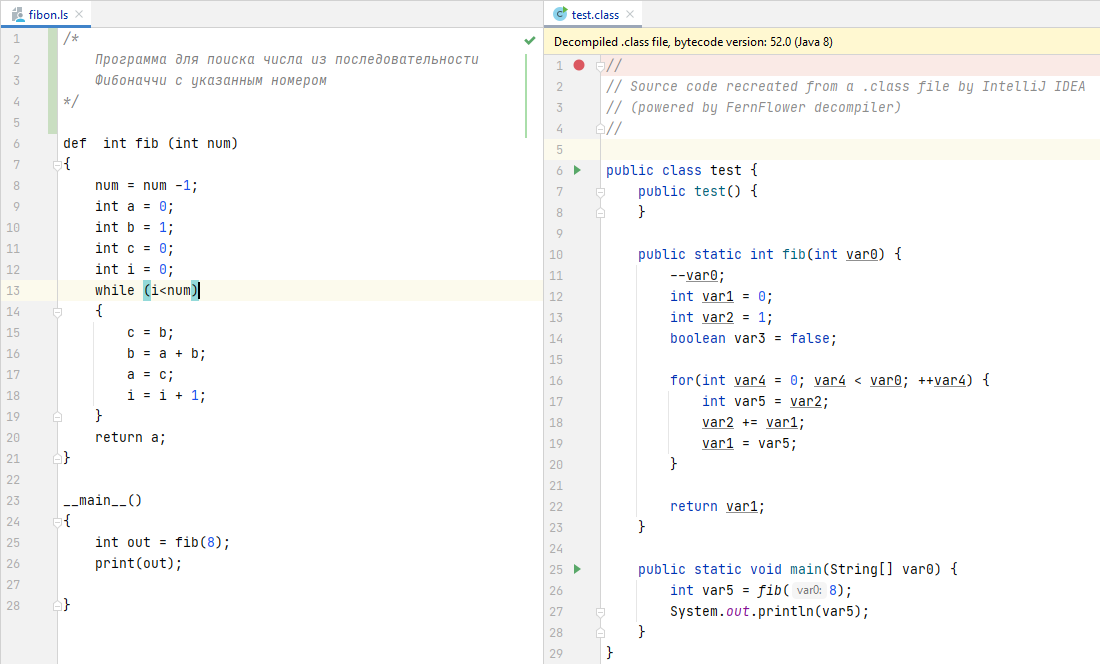


Рисунок 11

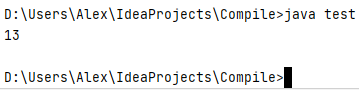


Рисунок 12

Поиск логарифма числа с указанным основанием (рисунок 13 и 14):

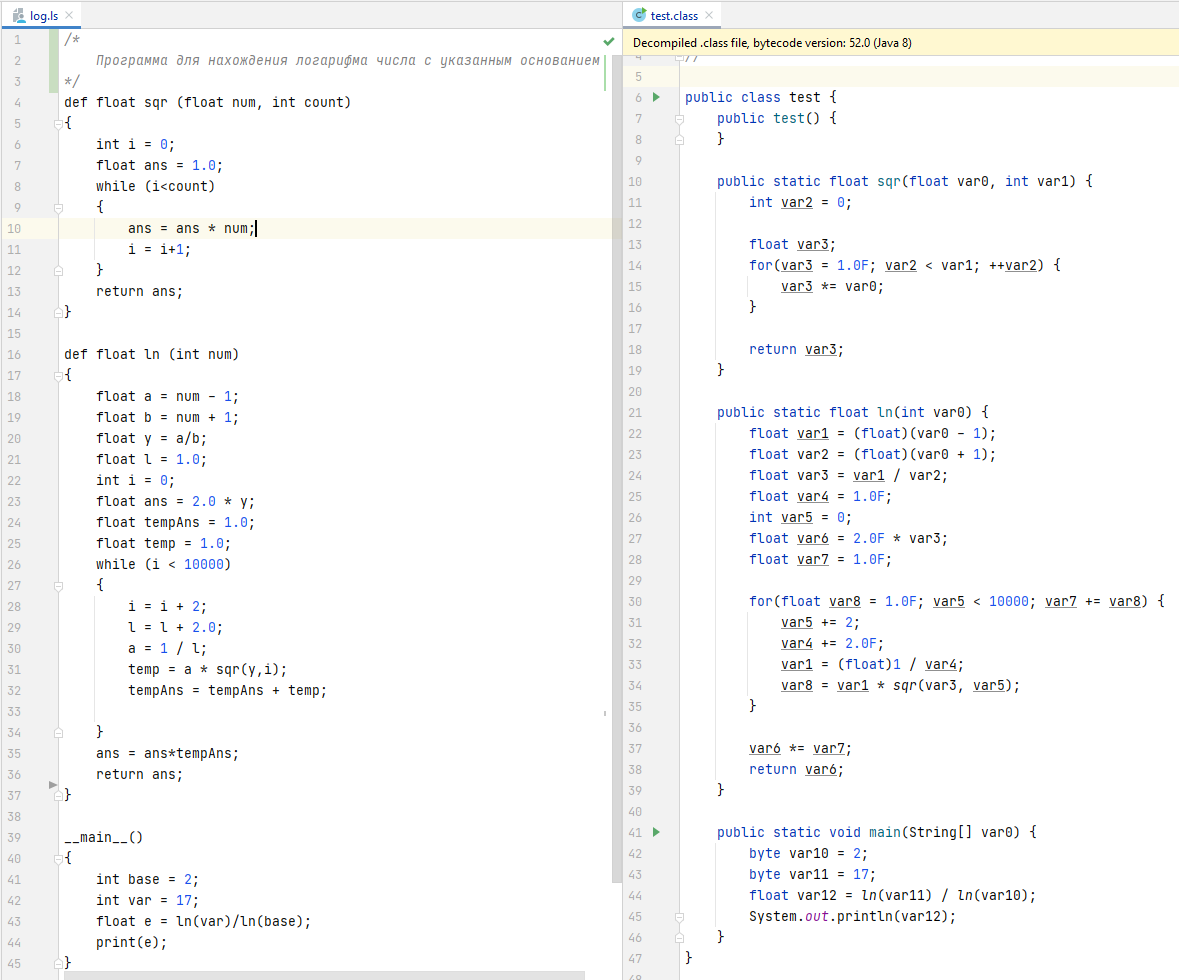


Рисунок 13

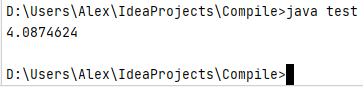


Рисунок 14