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

процедурно-ориентированного языка

Пояснительная записка

RU. 643.02068048.0001-01 81 01

На 15 листах

|  |  |  |
| --- | --- | --- |
| Руководитель |  | к.т.н. доцент кафедры ИЗИ Ю.М. Монахов |
|  |
| Исполнитель |  | студент гр. ИСБ-117 И.С. Бедняцкий |

**Владимир 2020**

# АННОТАЦИЯ

В данном программном документе приведён текст компилятора подмножества процедурно-ориентированного языка. Компилятор реализован на языке Java с использованием библиотеки Antlr. Основная функция компилятора – проверка принадлежности исходной цепочки входному языку и генерация выходной цепочки символов на языке ассемблер.

Разработка компилятора подмножества процедурного языка в ассемблер состоит из следующих стадий:

1) построение лексического анализатора;

2) построение синтаксического анализатора;

3) построение генератора объектного кода;

4) построение оптимизатора объектного кода.

Оглавление

[1 ПРОЕКТИРОВАНИЕ КОМПИЛЯТОРА 4](#_Toc41657097)

[1.1Основные требования 4](#_Toc41657098)

[1.2. Создание лексического анализатора 5](#_Toc41657099)

[1.3. Разработка синтаксического анализатора 6](#_Toc41657100)

[1.4. Построение генератора объектного кода 8](#_Toc41657101)

[1.5. Оптимизатор 10](#_Toc41657102)

[2 ПРОВЕРКА НА СООТВЕТСТВИЕ ОСНОВНЫМ ТРЕБОВАНИЯМ 11](#_Toc41657103)

# 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, break, continue);
6. Базовый вывод (строковой литерал, переменная);
7. Типы (целочисленный, вещественный).

* Требования к выходному языку:

1. Ассемблер.

## 1.2. Создание лексического анализатора

Лексический анализатор является первой фазой работы компилятора. Работа лексического анализатора основана на регулярных выражениях. На вход подаётся последовательность символов, выраженная исходным кодом. На выходе образуется поток токенов, лексем исходного языка.

Грамматика синтаксического и лексического анализатора основаны на грамматике языка pl0 с некоторыми изменениями. Лексический и синтаксический анализатор разработаны с помощью Antlr 4 под Java.

На рисунке 1 приведен список регулярных выражения для описания лексем языка.

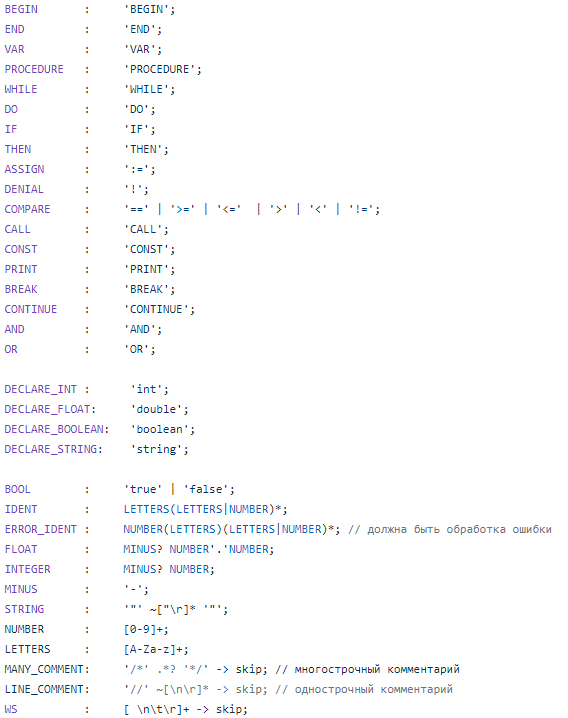


Рисунок Регулярные выражения для описания лексем

## 1.3. Разработка синтаксического анализатора

Второй стадией компилятора является синтаксический анализ. На вход синтаксическому анализатору подаётся набор токенов из лексического анализатора. На основе грамматики языка (рисунок 2-3) строится дерево разбора грамматики.

Все взаимодействия во время построения дерева разбора обрабатываются через класс MyVisitor, исходный код которого можно посмотреть на <https://github.com/iliyB/iliyB>.

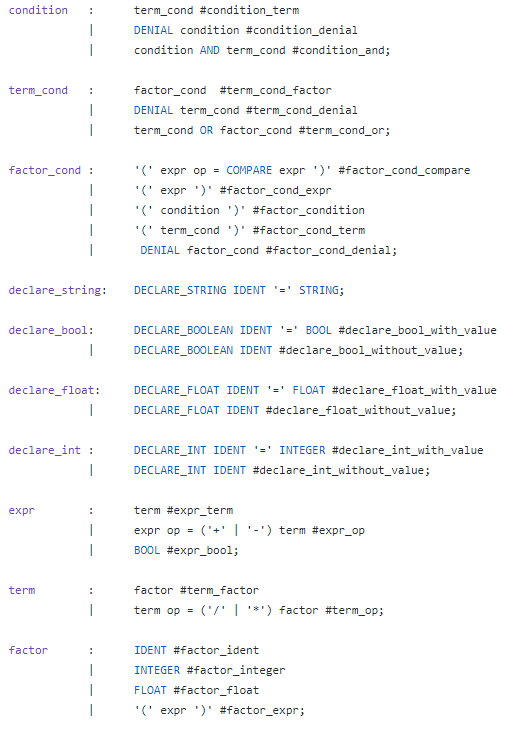


Рисунок Грамматика часть 1

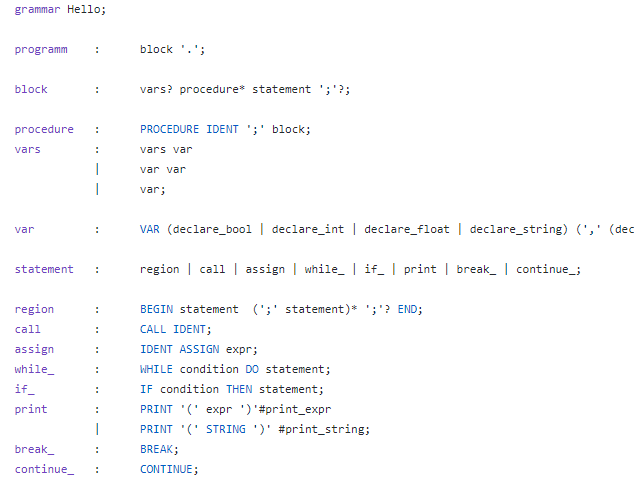


Рисунок Грамматика часть 2

Привила использования грамматики представлены на рисунке 4.

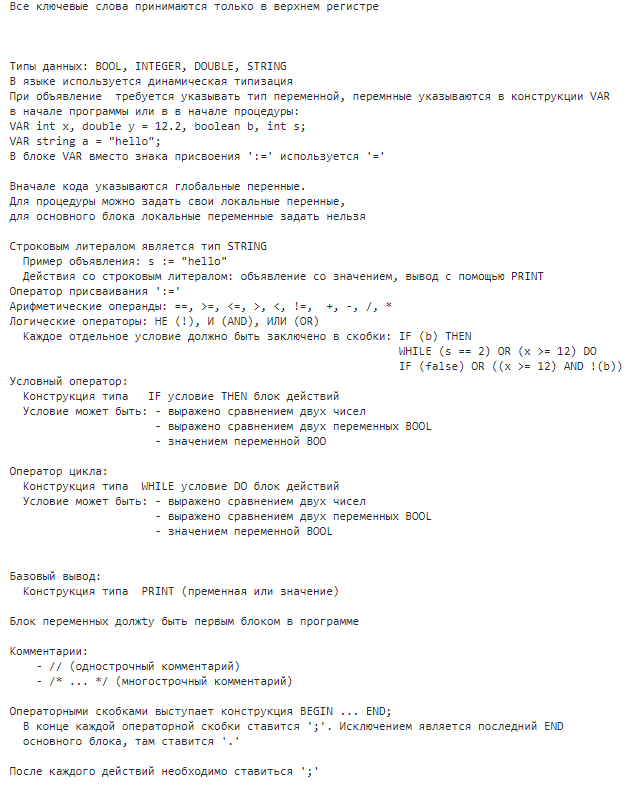


Рисунок Правила использования грамматики

## 1.4. Построение генератора объектного кода

Генерация объектного кода выполняется во время обхода дерева в классе MyVisitor.

Генерация осуществляется путем добавления в общий буфер объектного кода строк в объектном коде, эквивалентных исходному языку.

Буфер записывается в файл с расширением ll и преобразуется с помощью статического компилятора LLC.

Все функции генерации объектного кода представлены в классе GeneratorLLVM, где реализованы следующие функции:

* Generate – вывод буфера в файл ll
* Function\_start
* Function\_end
* Call
* Функции объявления переменных (глобально и локально)
* Функции присвоения значения переменным
* Функции вывода на экран
* Translate\_int\_to\_float
* Функции арифметических действий
* Функции логических действий
* Функции загрузки переменных из памяти
* If\_start
* If\_end
* Функции для цикла while
* Continue
* Break

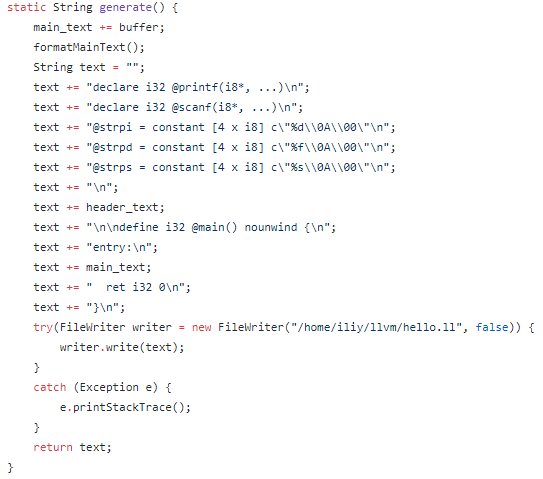


Рисунок Функция Generate

## 1.5. Оптимизатор

Оптимизатор оптимизирует промежуточный код на языке llvm. Оптимизатор реализован в классе OptimizationLLVM. Оптимизация состоит в:

- Удаление неиспользуемых переменных

- Если строка выводится в ходе программы несколько раз, то сохраняется она только один раз.

Оптимизатор работает в две фазы:

- Первая фаза – проход по объектному коду с запоминанием всех переменных и местах использования и инициализации этих переменных

- Вторая фаза – удаление всех упоминаний в коде, связанных с неиспользуемыми переменными.

# ПРОВЕРКА НА СООТВЕТСТВИЕ ОСНОВНЫМ ТРЕБОВАНИЯМ

При проектировании компилятора к основному языку были установлены следующие минимальные требования: наличие операторных скобок, игнорирование пробелов и идентации программы, поддержка многострочных комментариев и вызова функций. Наличие операторов присваивания, условных, цикла, break-continue, арифметических, логических. Должны присутствовать два типа данных. И выходная программа должна быть на ассемблере.

Далее приведено тестирование компилятора.

Рисунки 6-8 – Проверка областей видимости

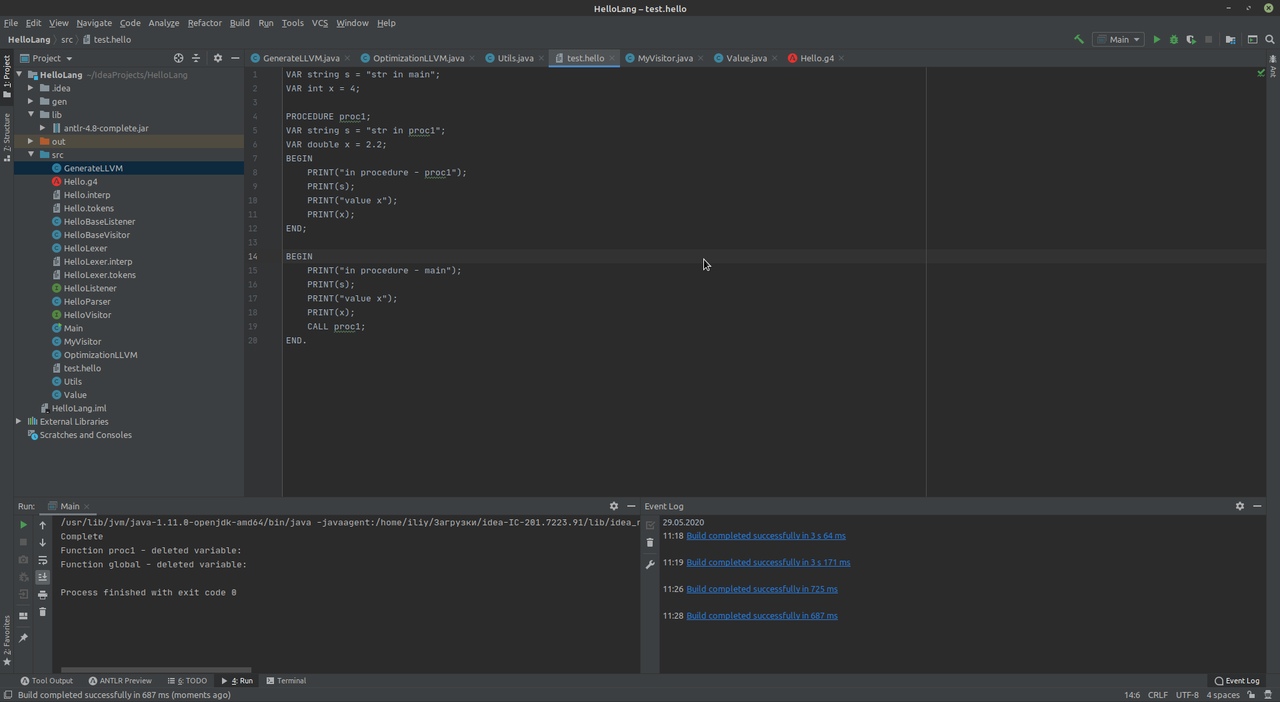


Рисунок Код на исходном языке

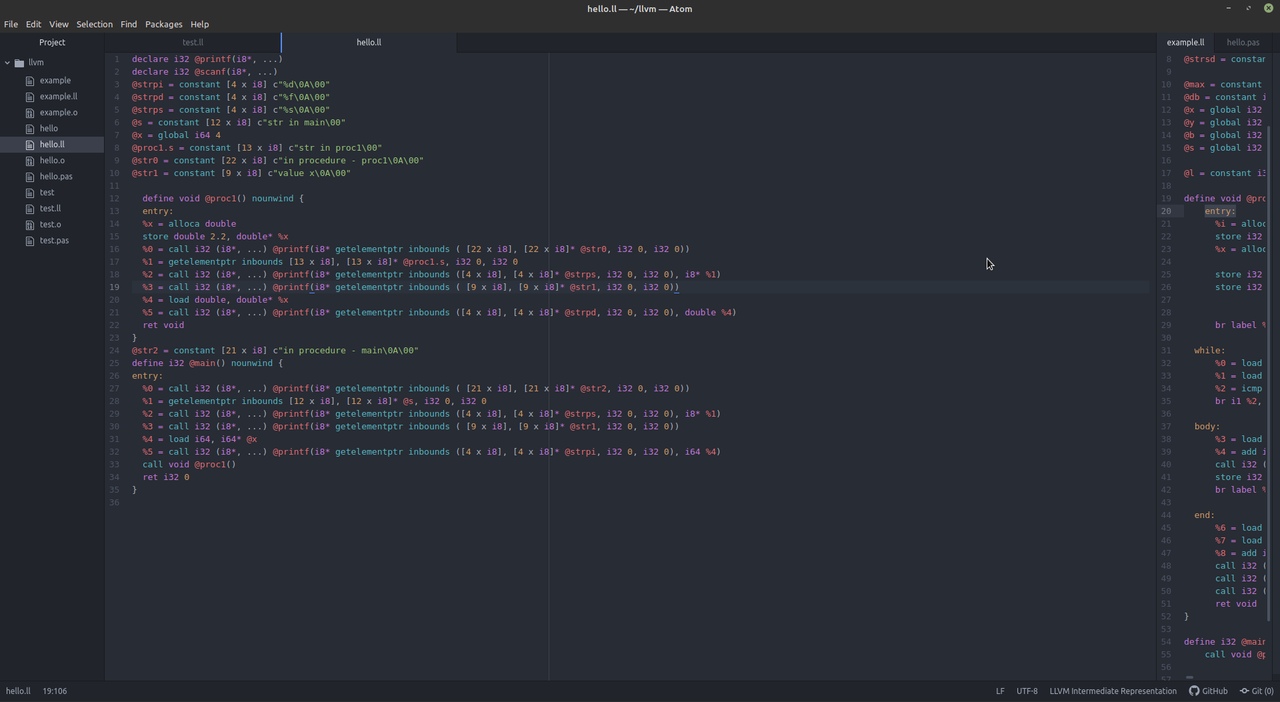


Рисунок Код на языке llvm

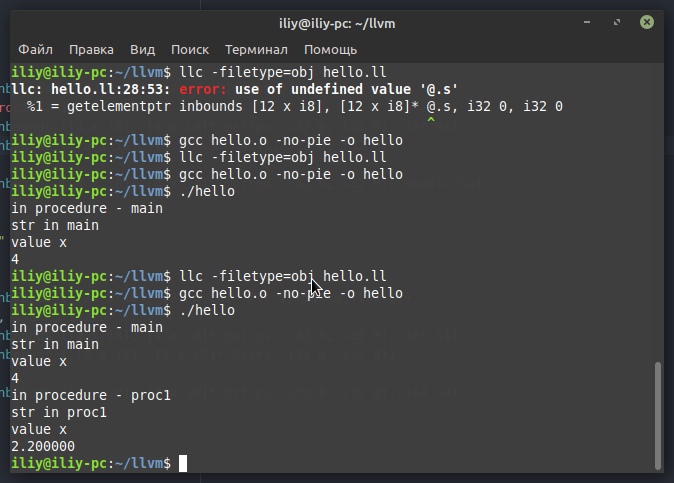


Рисунок Результат программы

Рисунки 9-11 – проверка условного оператора

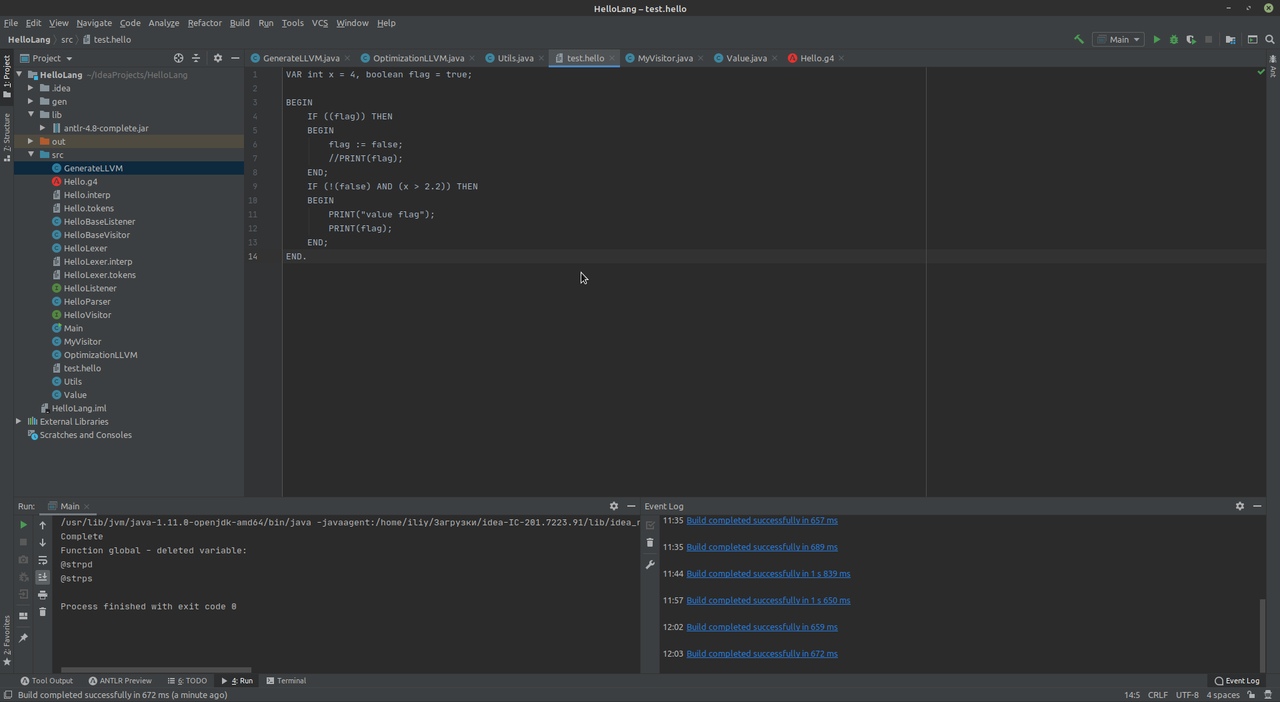


Рисунок Код на исходном языке

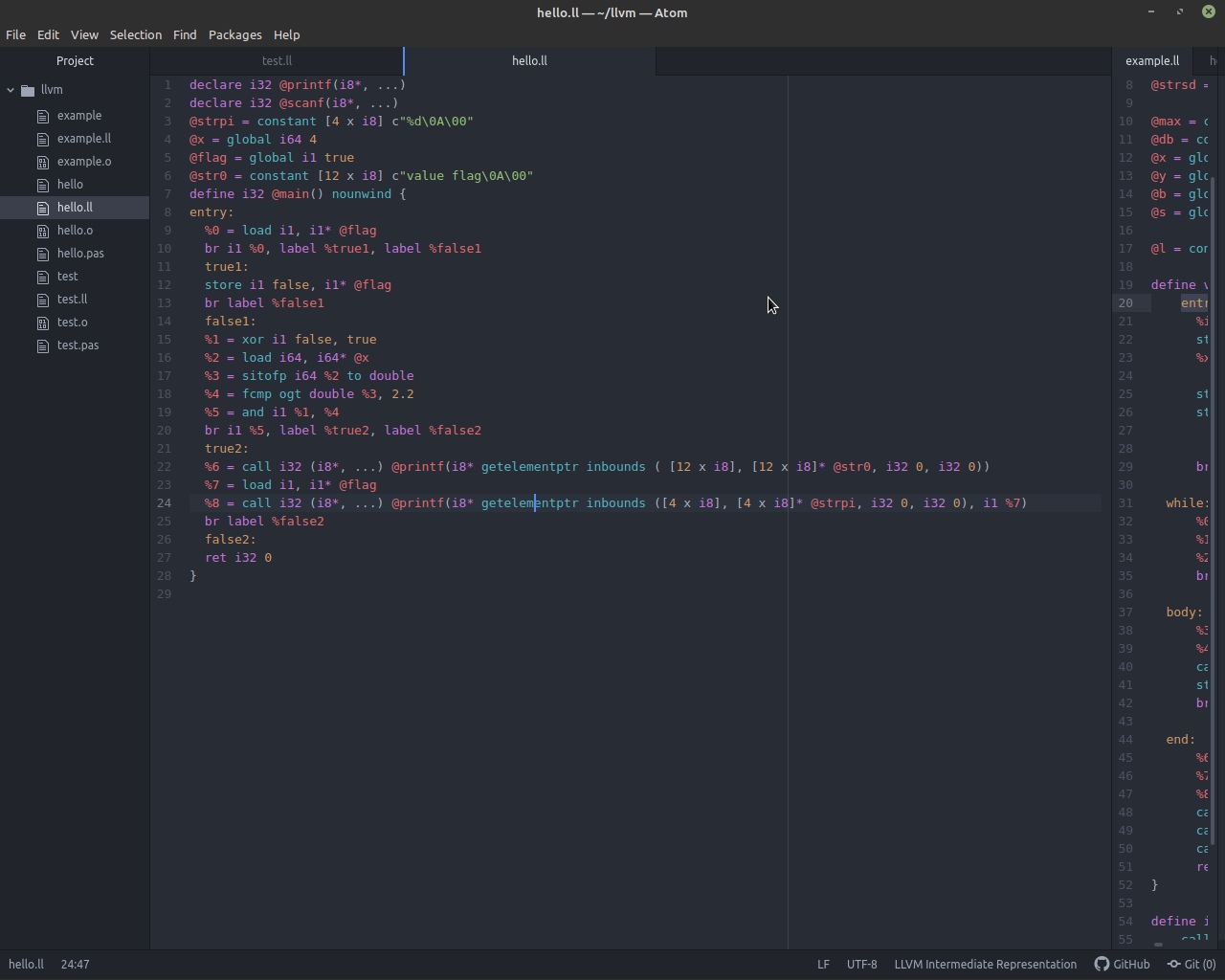


Рисунок Код на языке llvm

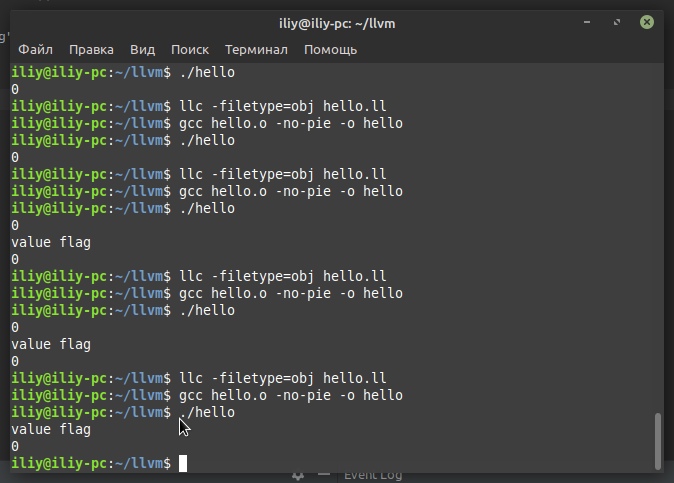


Рисунок Результат выполнения программы

Рисунки 12-14 – проверка цикла while

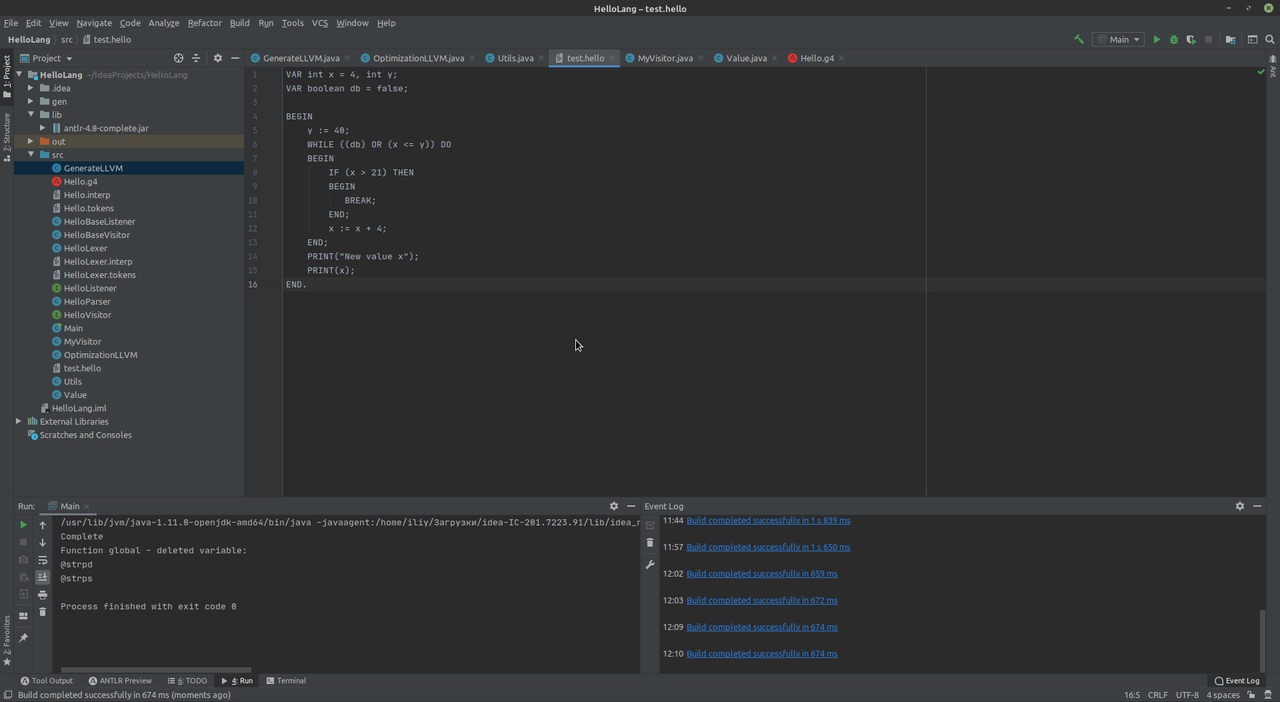


Рисунок Код на исходном языке

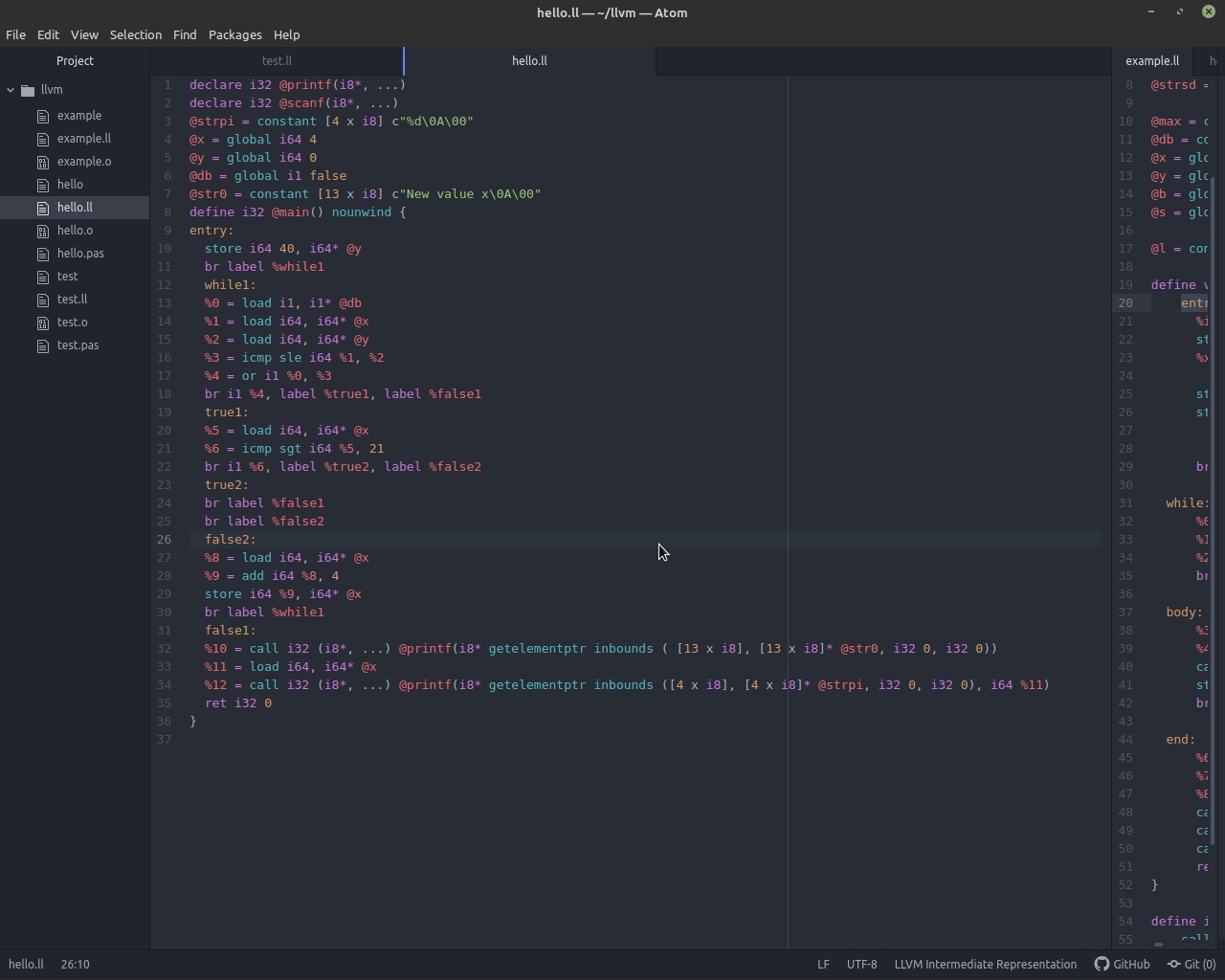


Рисунок Код на языке llvm

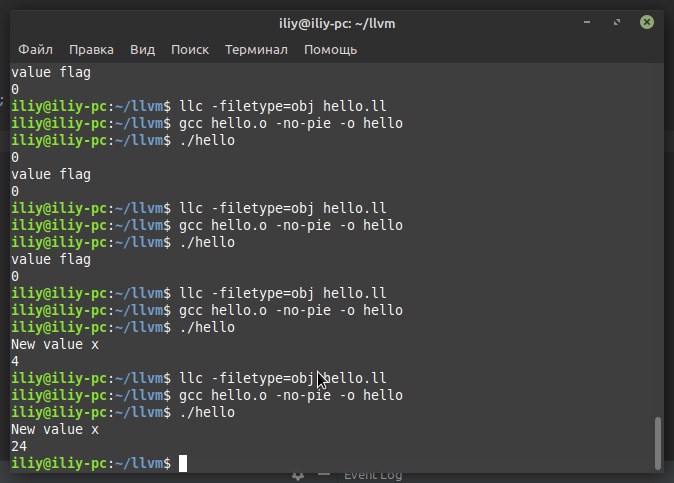


Рисунок Результат выполнения программы

Рисунки 15-17 – тестирование оптимизации

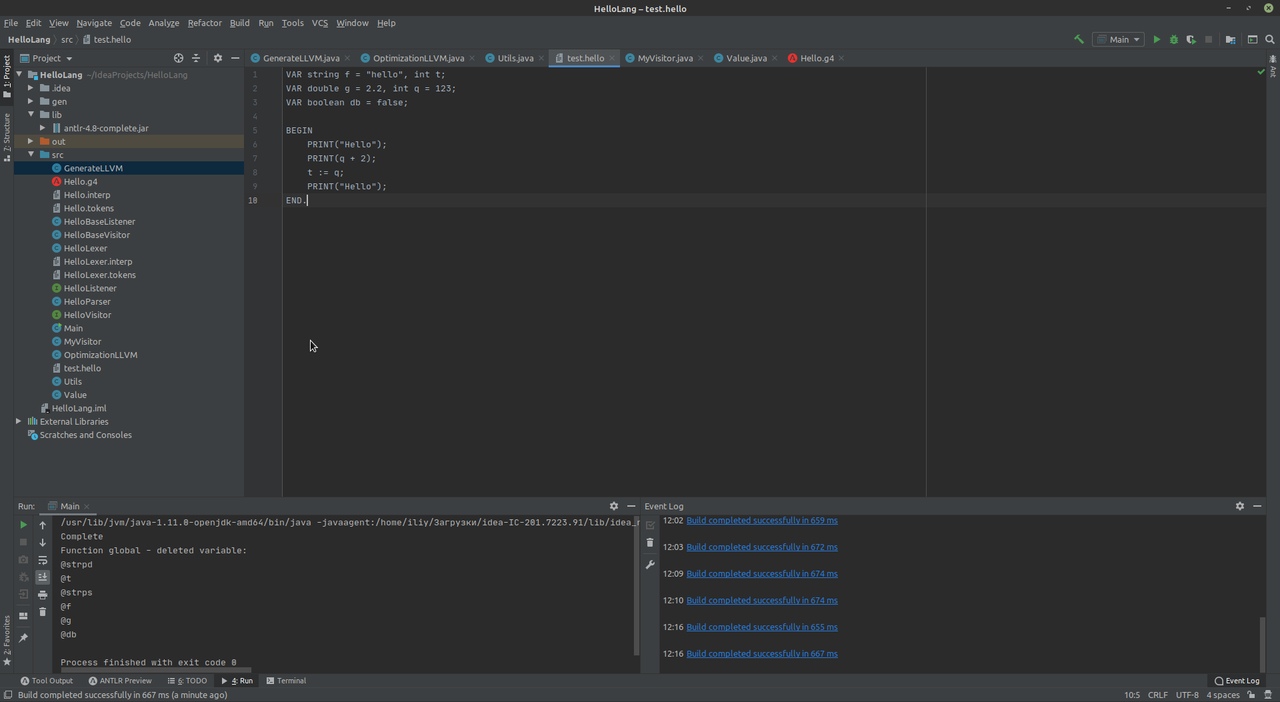


Рисунок Исходный код

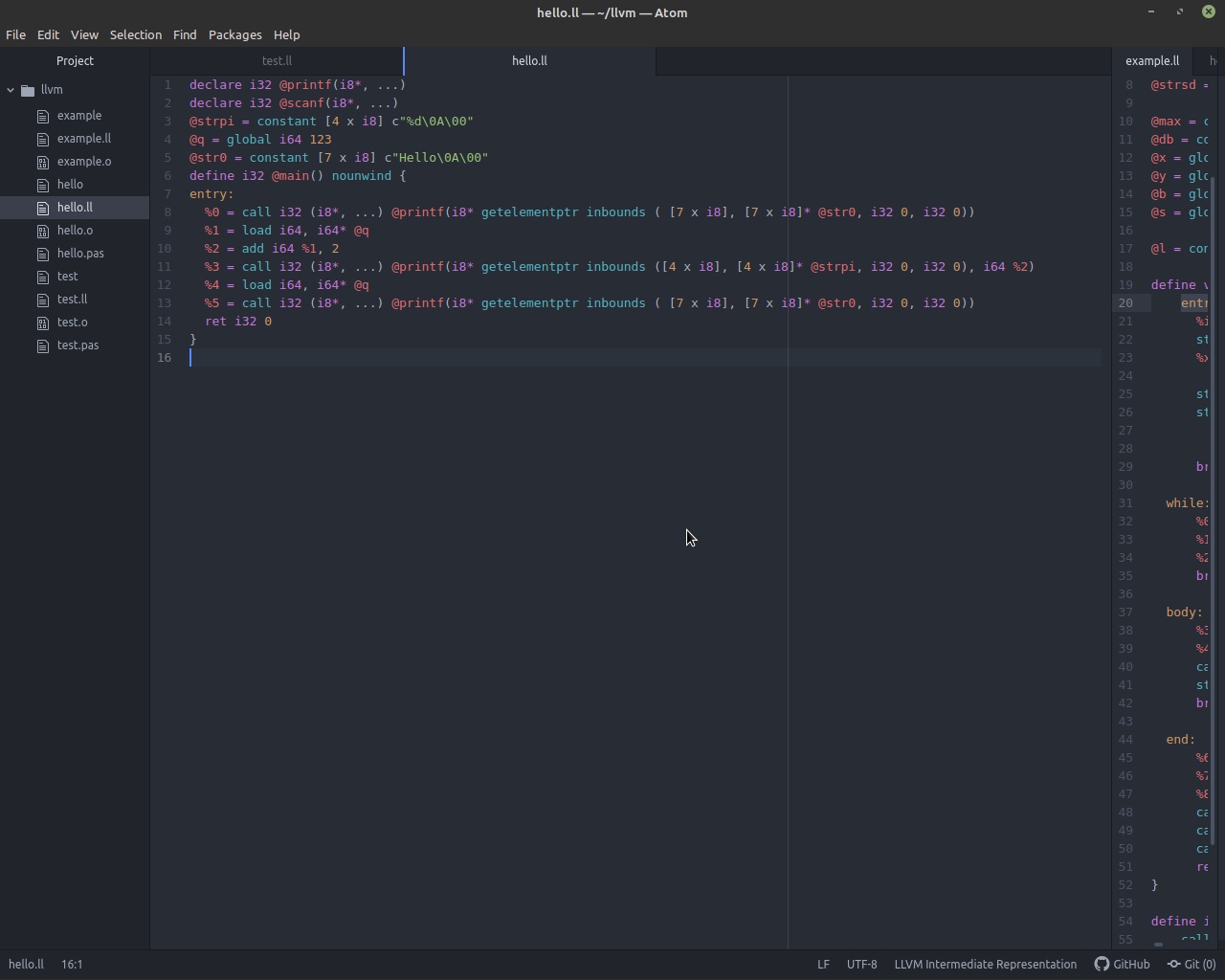


Рисунок Код на языке llvm

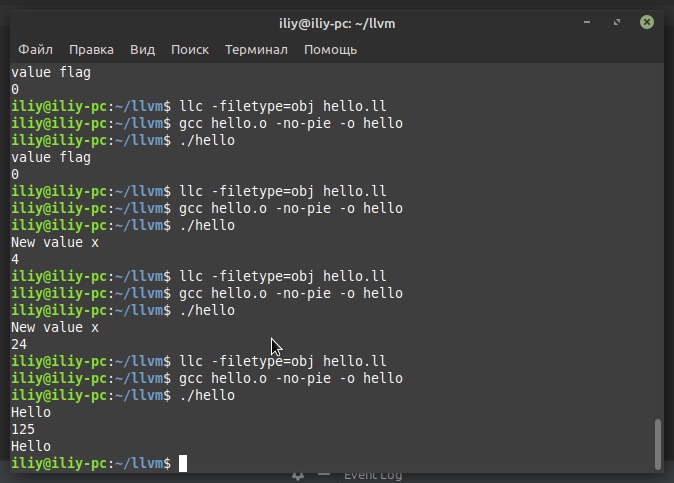


Рисунок Результат выполнения программы

Реквизиты к курсовой работе:

<https://github.com/iliyB/iliyB>