Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

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 Е.Е. Калитин |

**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;

Оглавление

[АННОТАЦИЯ 2](#_Toc41845436)

[1 ПРОЕКТИРОВАНИЕ КОМПИЛЯТОРА 4](#_Toc41845437)

[1.1 Основные требования 4](#_Toc41845438)

[1.2. Создание лексического анализатора 5](#_Toc41845439)

[1.3. Разработка синтаксического анализатора 6](#_Toc41845440)

[1.4. Построение генератора объектного кода 10](#_Toc41845441)

[2 ПРОВЕРКА НА СООТВЕТСТВИЕ ОСНОВНЫМ ТРЕБОВАНИЯМ 11](#_Toc41845442)

# 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

За основу грамматики синтаксического и лексического анализаторов взят язык pl0 с добавлением необходимых для выполнения требований ТЗ функций. Лексический и синтаксический анализатор разработаны с помощью Antlr 4 на ЯП Java.

Начальная фаза компилятора – лексический анализ. Читается поток символов, составляющих исходную программу, и группируется в значащие последовательности, называющиеся лексемами. Для каждой лексемы анализатор строит выходной токен. Этот токен передается следующей фазе – синтаксическому анализтору.

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

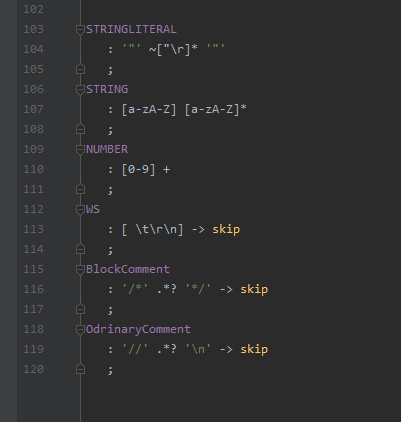


Рисунок 1 Регулярные выражения для описания лексем

## 1.3. Разработка синтаксического анализатора

Второй стадией компилятора – синтаксический анализ.

На вход синтаксическому анализатору подаётся набор токенов из лексического анализатора.  
Преобразование токенов, полученных при лексическом анализе, проводится с целью создания древовидного представления, которое описывает грамматическую структуру потока токенов.

Технические моменты при построение дерева и прочие взаимодействия обрабатываются в классе MyVisitor.



Рисунок 2 Грамматика часть 1

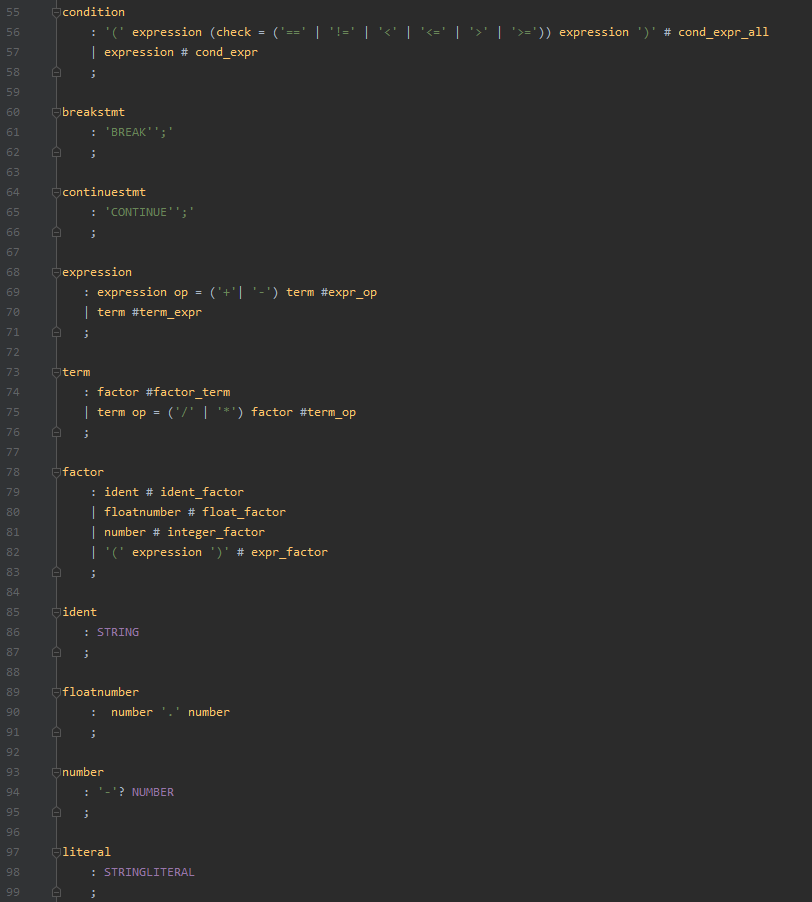


Рисунок 3 Грамматика часть 2

Привила использования грамматики представлены на рисунке 4.

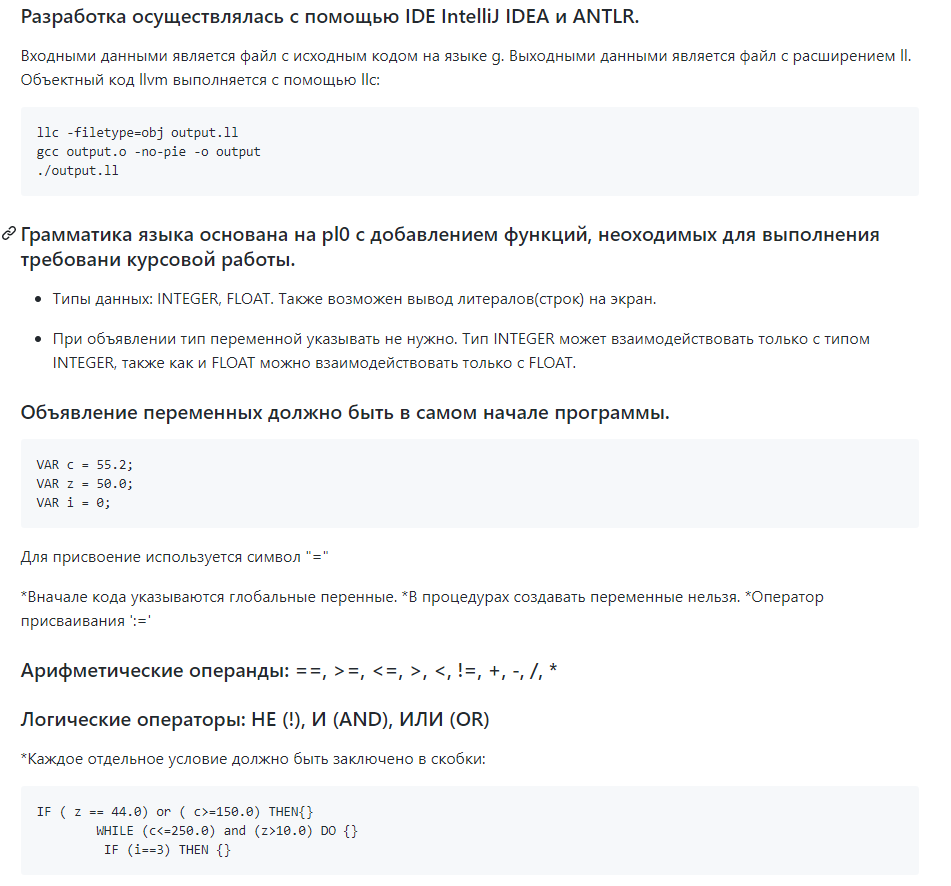


Рисунок 4 Правила использования грамматики часть 1.

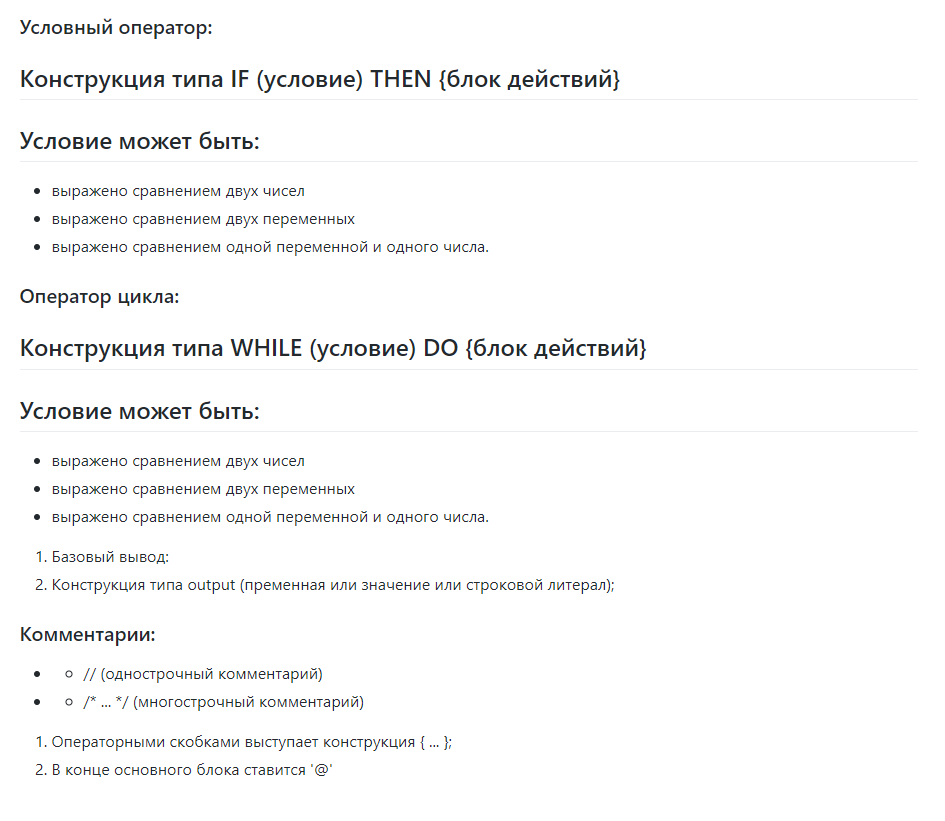


Рисунок 4 Правила использования грамматики часть 2.

## 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 в консоль
* Функции арифметических действий
* Функции логических действий
* Функции загрузки переменных из памяти
* If\_start
* If\_end
* Функции для цикла while
* Continue
* Break



Рисунок 5 Функция Generate

# 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.

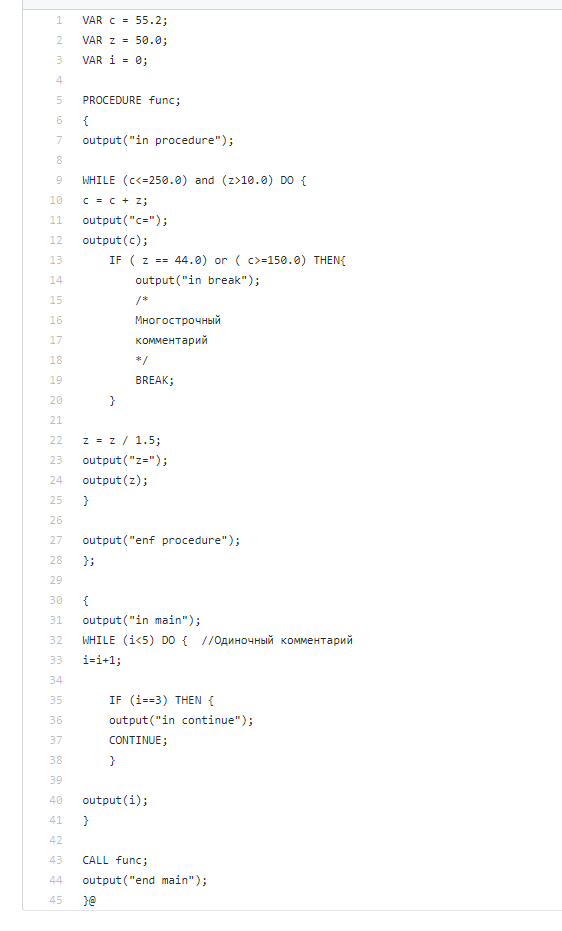


Рисунок 6. Тестовая программа, покрывающая все возможности языка.

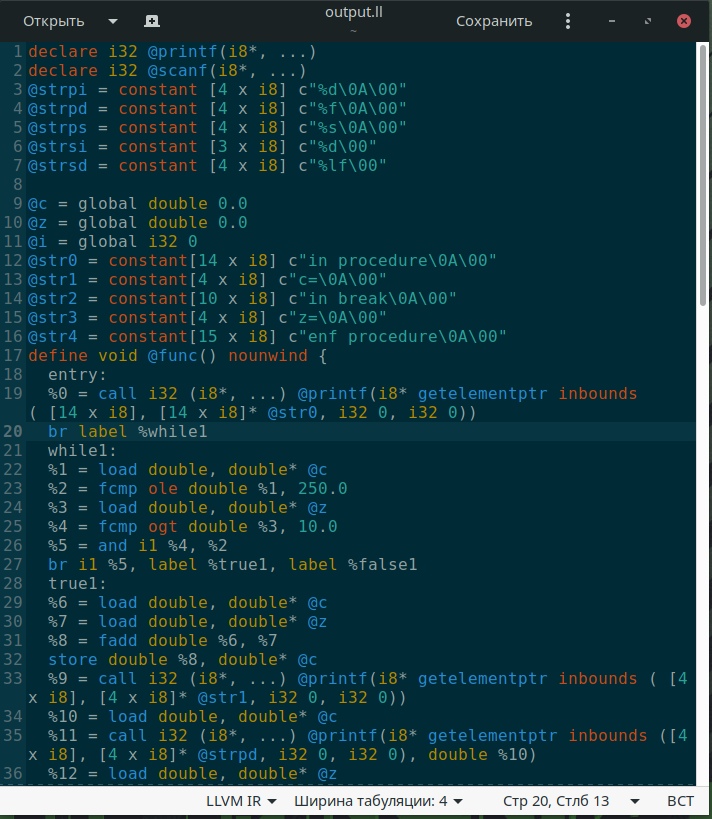


Рисунок 7. Сгенерированный код. Часть1.



Рисунок 8. Сгенерированный код. Часть 2.

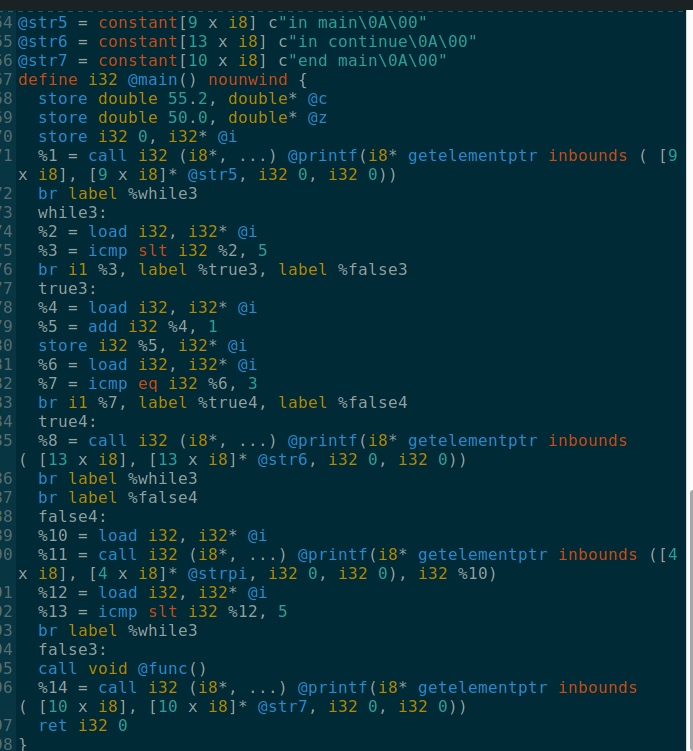


Рисунок 9. Сгенерированный код. Часть 3.

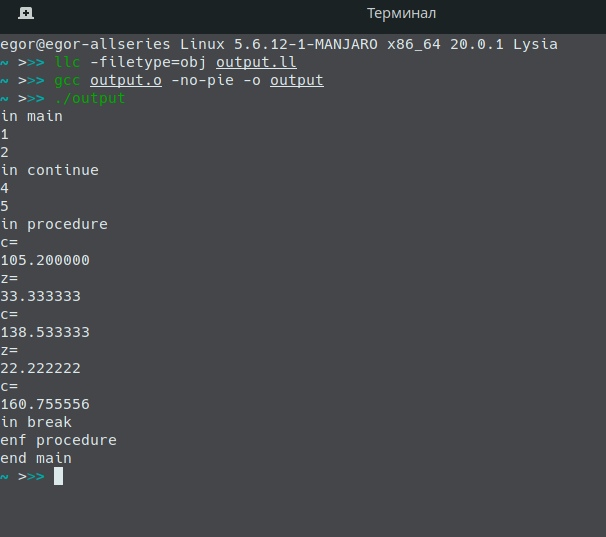


Рисунок 10. Результат выполнения сгенерированного кода.

Реквизиты к курсовой работе:

https://github.com/Catharsiz/kyrsach2020