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. **643.02068048.00001**

АННОТАЦИЯ

В данном программном документе приведён текст компилятора подмножества процедурно-ориентированного языка. Компилятор реализован на языке Java с использованием библиотеки Antlr. Реализация объектного кода реализована при помощи llvm.

Существует несколько стадий для разработки компилятора подмножества процедурного языка в ассемблер, перечислим их:

1. лексический анализатор;
2. синтаксический анализатор;
3. генератор объектного кода.

**Содержание**

Оглавление

[1. ПРОЕКТИРОВАНИЕ КОМПИЛЯТОРА 4](#_Toc41682700)

[1.1. Основные требования 4](#_Toc41682701)

[1.2. Создание лексического анализатора 5](#_Toc41682702)

[1.3. Разработка синтаксического анализатора 6](#_Toc41682703)

[1.4. Построение генератора объектного кода 9](#_Toc41682704)

[2. ПРОВЕРКА НА СООТВЕТСТВИЕ ОСНОВНЫМ ТРЕБОВАНИЯМ 13](#_Toc41682705)

1. ПРОЕКТИРОВАНИЕ КОМПИЛЯТОРА
   1. Основные требования

Существуют некоторые требования, в соответствии с которыми мы будем разрабатывать программу, рассмотрим их:

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 также должна быть поддержка вызова функций.

Операторы:

1. Оператор присваивания.
2. Арифметика (\*, /, +, -, >, <, =).
3. Логические операторы (И, ИЛИ, НЕ).
4. Условный оператор (ЕСЛИ).
5. Операторы цикла (while, break, continue).
6. Базовый вывод (строковый литерал, переменная).
7. Типы (целочисленный 32 бита, с плавающей запятой 32 бита).

Требования к выходному языку:

1. В ассемблере.

## 1.2. Создание лексического анализатора

Одной из первых фаз в работе компилятора является создание лексического анализатора, работа которого основана на регулярных выражениях, которые подаются последовательностью символов на вход. Результатом работы лексического анализатора является поток лексем исходного языка, которые потом выстраиваются в поток выходных токенов, передающих на следующий этап построения компилятора.

Стоит учесть, что грамматика основана на грамматике языка pl/0 с некоторыми дополнениями, которые реализованы в соответствии с требованиями.

Также разработка этой части, как и следующей, реализована на языке Java с помощью библиотеки antlr4.

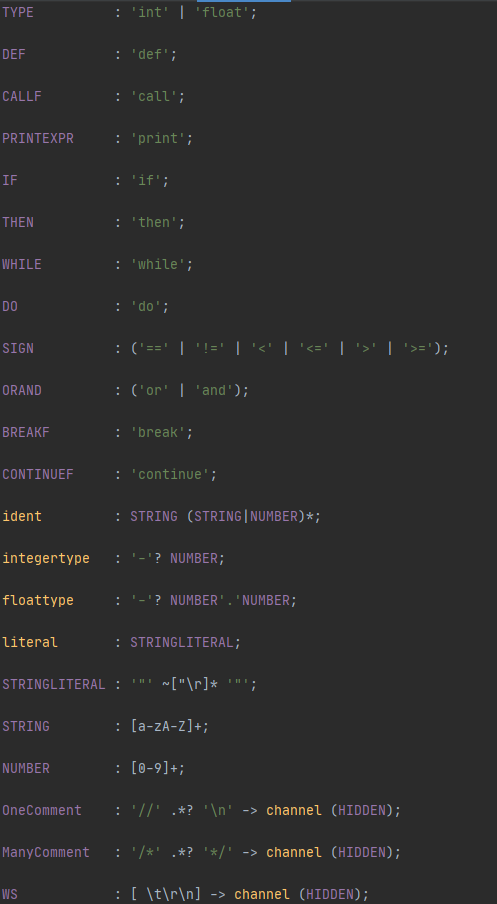


Рисунок 1. Список регулярных выражений.

## 1.3. Разработка синтаксического анализатора

Следующей стадией компилятора является создание синтаксического анализатора, на вход которого подается набор токенов из лексического анализатора. На основе грамматики, написанной нами, строится синтаксическое дерево разбора грамматики, все связи в построении дерева обрабатываются в классе MyVisitor.

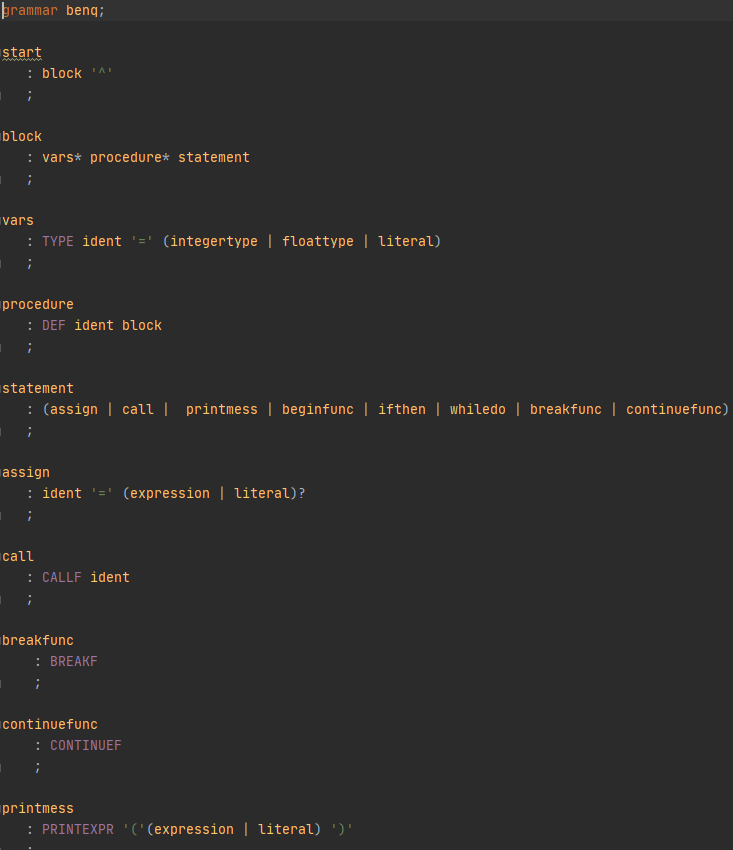


Рисунок 2. Грамматика.

## 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, эквивалентных исходному языку.

Сохраненный буфер со сгенерированным объектным кодом записывается в файл с расширением ll и может быть преобразован с помощью статического компилятора LLC.

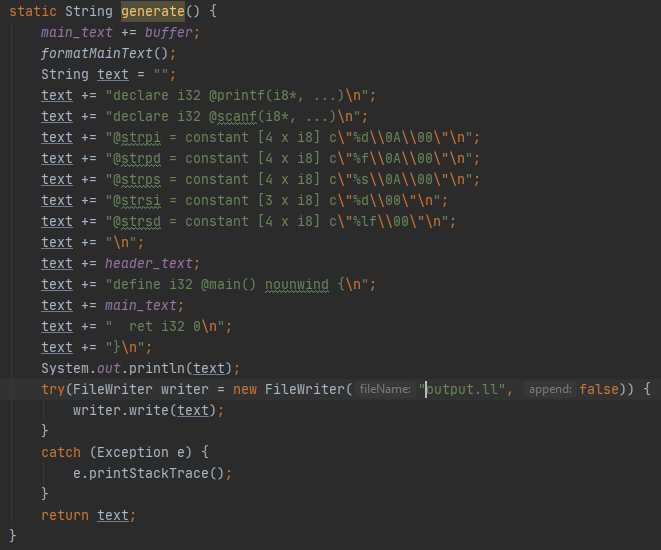


Рисунок 3. Generate.

1. ПРОВЕРКА НА СООТВЕТСТВИЕ ОСНОВНЫМ ТРЕБОВАНИЯМ

Проверка работы функций, вызовов функций и принтов. На данных скриншотах расположены пример на исходном языке, пример с использованием объектного кода и результат выполнения программы. В дальнейших тестированиях мы будем так же использовать 3 разных вида демонстрации примера.

Как мы видим, требования были соблюдены, а именно: игнорирование пробелов, наличие операторных скобок, поддержка многострочных комментариев и вызова функций, наличие операторов присваивания, цикла, break-continue, арифметических, логических (И, ИЛИ, НЕ). Необходимо присутствие двух типов данных (в данном случае Integer и Float).

.

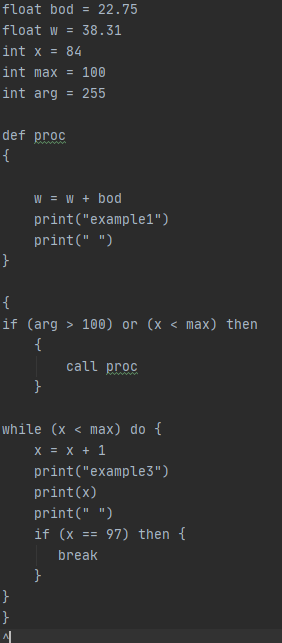


Рисунок 4. Пример 1

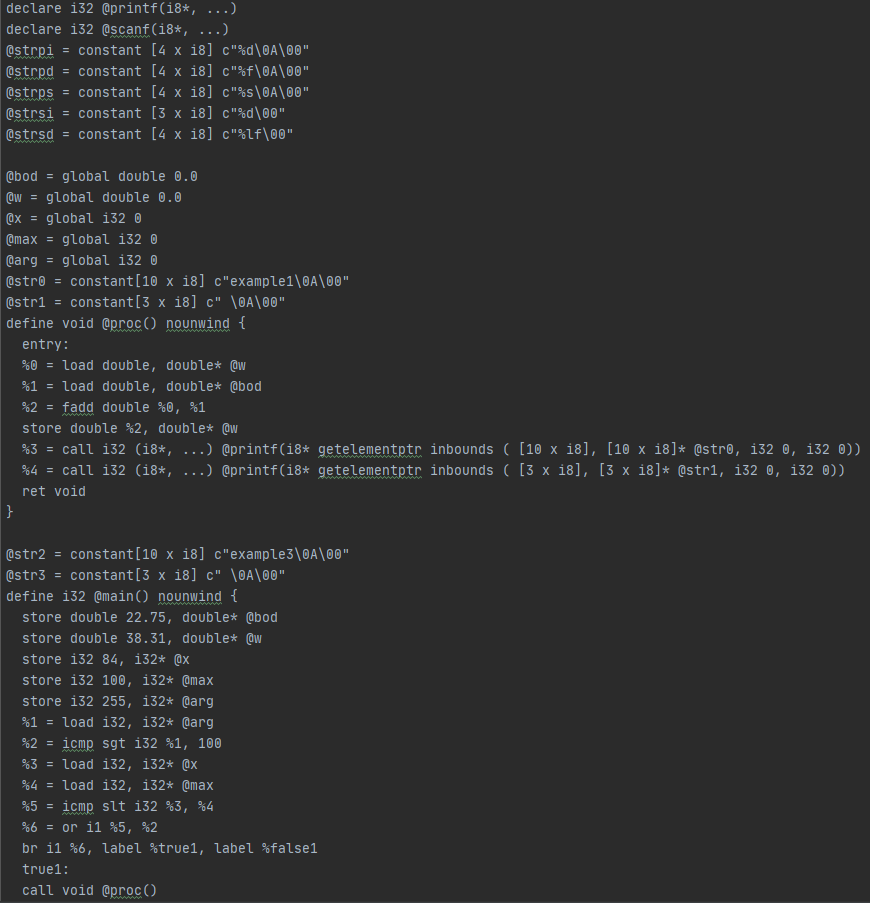


Рисунок 5. Пример 2.

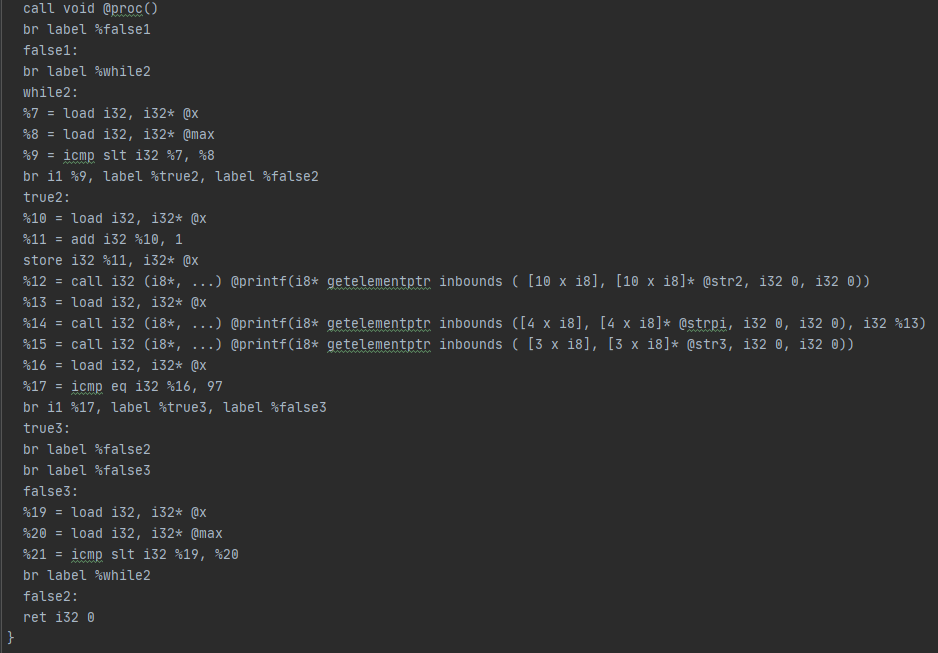


Рисунок 6. Пример 3.

Весь исходный код, реализованный в этой программе, загружен на github.

https://github.com/Higara33/compil