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

RU. 643.02068048.0001-01 81 01

На 11 листах

|  |  |  |
| --- | --- | --- |
| Руководитель |  | к.т.н. доцент кафедры ИЗИ Ю.М. Монахов |
|  |
| Исполнитель |  | студентка гр. ИБ-117 А.А. Мохова |

**Владимир 2020**

АННОТАЦИЯ

В данном документе представлены текст компилятора подмножества процедурно-ориентированного языка, описание основных функций и тестирование. Компилятор реализован на языке Java c использованием генератора формальных языков ANTLR.

Компилятор – это специальная программа, которая переводит текст программы, написанный на языке программирования, в набор машинных кодов.

Выполнение курсовой работы состоит из следующих этапов:

1. Проектирование лексического анализатора
2. Построение синтаксического анализатора
3. Проектирование и оптимизация генератора объектного кода

СОДЕРЖАНИЕ

1 ОПИСАНИЕ РАБОТЫ 4

1.1 Основные требования 4

1.2 Проектирование лексического анализатора 4

1.3 Построение синтаксического анализатора 6

1.4 Проектирование и оптимизация генератора объектного кода 7

1.5 Тестирование 8

ЗАКЛЮЧЕНИЕ 12

ПРИЛОЖЕНИЕ А 13

1 ОПИСАНИЕ РАБОТЫ

* 1. **Основные требования**

В следующих подразделах будут описаны основные решения по реализации требований, изложенных в Техническом Задании на курсовую работу. Они разделены на два блока: требования к входному языку и требования к операторам.

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. **Проектиррование лексического анализатора**

Этап проектирования лексического анализатора заключается в написании регулярных выражений, с помощью которых компилятор выделяет в тексте исходной программы лексемы.

Для построения анализатора использовался генератор формальных языков ANTLR, а грамматика основана на грамматике pl0.

На рисунке 1 представлены некоторые из регулярных выражений, полный перечень которых расположен в файле MAA.g4, который можно будет посмотреть, перейдя по ссылке, приведенной в Приложении А.

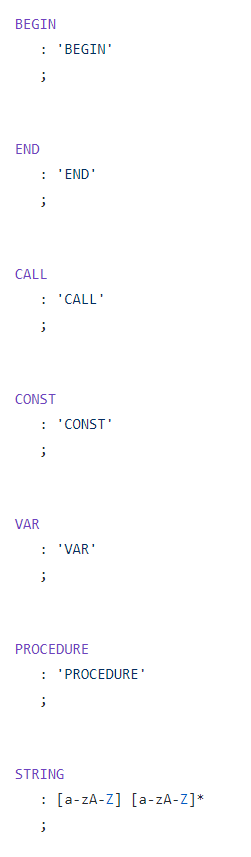
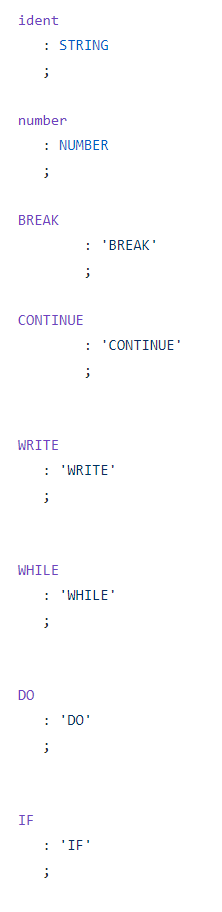


Рис.1

* 1. **Построение синтаксического анализатора**

Этап построения синтаксического анализатора разделяется на несколько этапов: внесение дополнительных правил грамматики, которые относятся к выполенению различных функций программы, генерацию дерева разбора и на анализ пришедшего поддерева. Для последней из перечисленных целей используется шаблон Visitor, методы которого описываются в классе MyVisitor.

Частично правила грамматики приведены на рисунке 2. Полное описание можно посмотреть так же в файле MAA.g4, перейдя по ссылке в Приложении А.

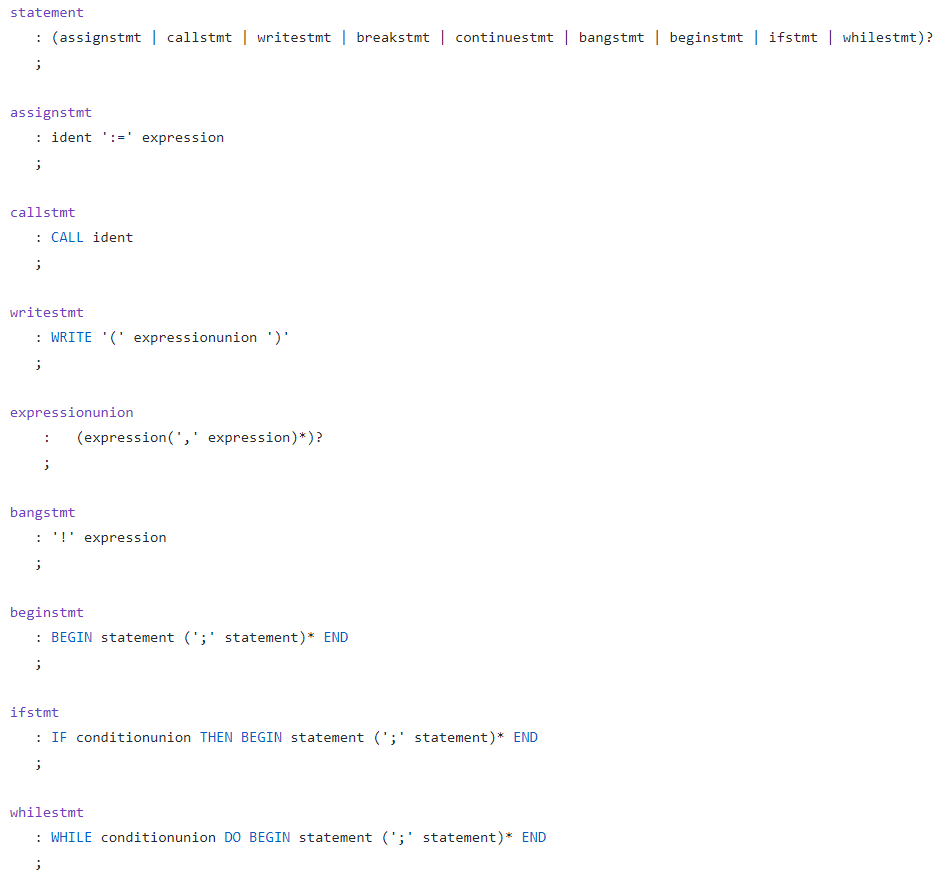


Рис.2

Краткое описание возможностей и правил использования:

1. Все ключевые слова принимаются только в верхнем регистре.
2. Типы данных: int, float.

Объявление переменных происходит перед главным блоком программы, при этом обязательно указываются тип, имя переменной и ее значение.

1. Операторными скобками выступает конструкция BEGIN…END.
2. Индентация программы игнорируется.
3. Поддерживаются коментарии двух видов:

однострочный - //

многострочный - /\*...\*/

1. Оператор присваивания: (:=), арифметические операции: (+, -, \*, /), сравнение: (=, !=, <, <=, >, >=).
2. Условным оператором выступает конструкция IF...THEN, оператором цикла WHILE…DO.
3. Базовый вывод: WRITE.
   1. **Проектирование и оптимизация генератора объектного кода**

Генерация объектного кода – это перевод компилятором внутреннего представления исходной программы в результирующую объектную программу на языке ассемблера или непосредственно на машинном языке (машинных кодах).

Генерация объектного кода выполняется после того, как выполнен синтаксический анализ программы и все необходимые действия по подготовке к генерации кода: распределено адресное пространство под функции и переменные, проверено соответствие имен и типов переменных, констант и функций в синтаксических конструкциях исходной программы и т.д.

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 GLLVM. Основная функция – рисунок 3.

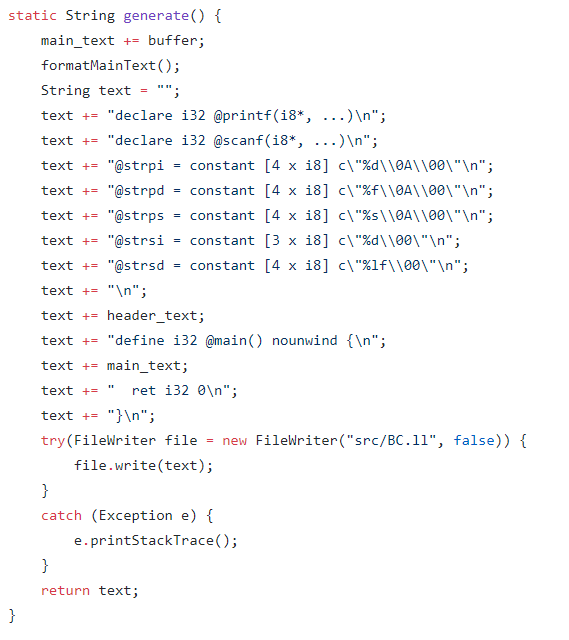


Рис.3

* 1. **Тестирование**

Объектный код llvm выполняется с помощью llc. Для этого была поставлена ВМ c ОС Debian (Linux). Для запуска объектного кода выполняется последовательность команд:

llc –filetype=obj BC.ll  
gcc BC.o –no-pie –o BC

./BC

Далее, на рисунках 4 - 11, приведены тесты.

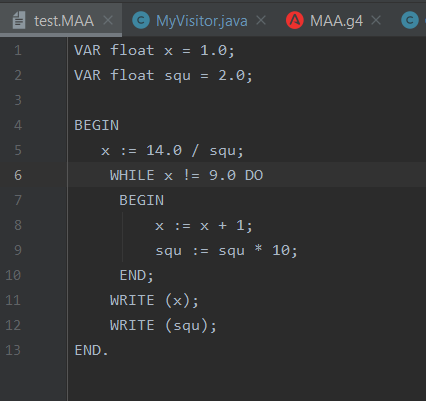


Рис.4 «Исходный текст программы»

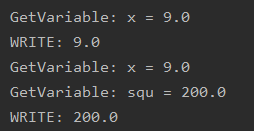


Рис.5 «Результат работы интерпретатора»

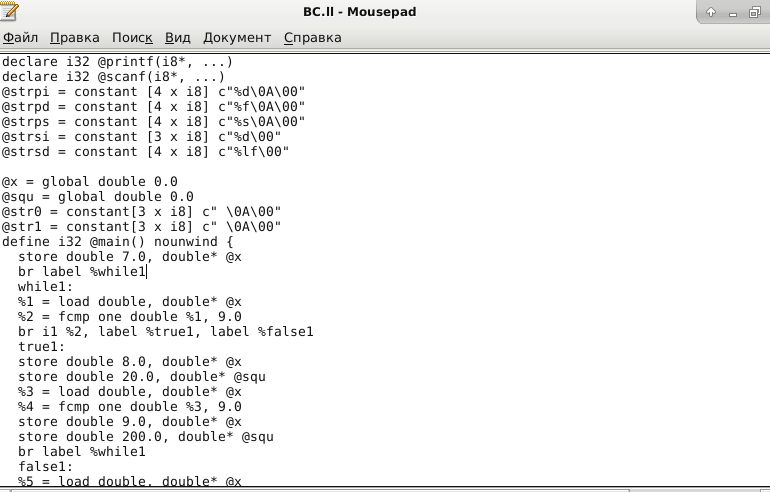


Рис.6 «Объектный код»

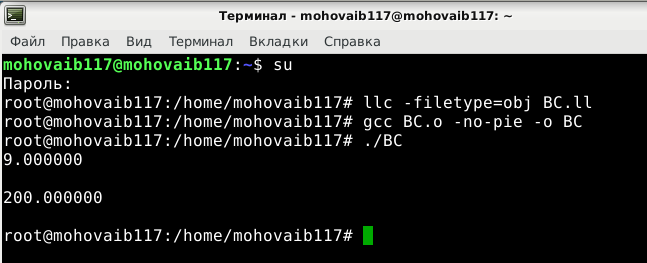


Рис.7 «Результат работы компилятора»

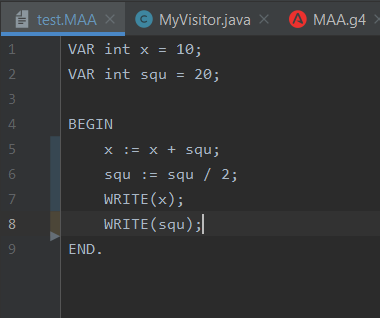


Рис.8 «Исходный текст программы»

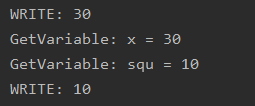


Рис.9 «Результат работы интерпретатора»

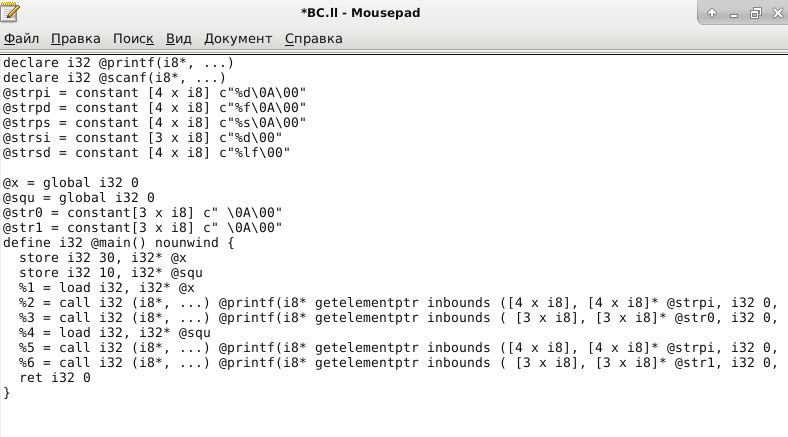


Рис.10 «Объектный код»

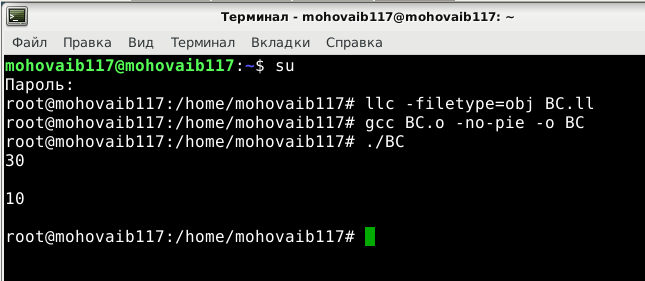


Рис.11 «Результат работы компилятора»

ЗАКЛЮЧЕНИЕ

В заключении хотелось бы отметить, что все три поставленные задачи были выполнены. Проект удовлетворяет требованиям, изложенным в Техническом Задании.

ПРИЛОЖЕНИЕ А

Ссылка на репозиторий, в котором хранятся проект и различные документы:

<https://github.com/amohovaa/MAA>