**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**

Пояснительная записка

На 16 листах

|  |  |  |
| --- | --- | --- |
| Руководитель |  | к.т.н. доцент кафедры ИЗИ Ю.М. Монахов |
| Исполнитель |  | студент гр. ИБ-118 С.В Корочкин |

**Владимир 2021**

**Оглавление**

[**Аннотация** 3](#_Toc72709985)

[**1 ПРОЕКТИРОВАНИЕ КОМПИЛЯТОРА** 4](#_Toc72709986)

[1.1 Основные требования 4](#_Toc72709987)

[1.2 Лексический анализатор 5](#_Toc72709988)

[1.3 Синтаксический анализатор 6](#_Toc72709989)

[1.4 Таблица символов 8](#_Toc72709990)

[1.5 Построение генератора промежуточного кода 9](#_Toc72709994)

[1.6 Построение генератора объектного кода 11](#_Toc72709995)

[**2** **ПРОВЕРКА НА СООТВЕТСТВИЕ ОСНОВНЫМ ТРЕБОВАНИЯМ** 13](#_Toc72709996)

[**3. СПИСОК ИСПОЛЬЗОВАННЫХ МАТЕРИАЛОВ** 17](#_Toc72709997)

# **Аннотация**

В данном программном документе приведено описание компилятора подмножества процедурно-ориентированного языка Pascal. Компилятор реализован на языке Python с использованием библиотек ply и llvmlite. Основная функция компилятора – проверка принадлежности исходной цепочки входному языку и генерация выходной цепочки символов виде llvm-кода.

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

* Реализация лексического анализатора
* Реализация синтаксического анализатора
* Реализация таблицы символов
* Реализация генератора промежуточного кода (трехадресного кода)
* Реализация транслятора трехадресного кода в код llvm

Реквизиты к курсовой работе: <https://github.com/KorochkinStepan/CompilerLLVM>

# **1 ПРОЕКТИРОВАНИЕ КОМПИЛЯТОРА**

## Основные требования

Разработка будет производиться в соответствии со следующими требованиями:

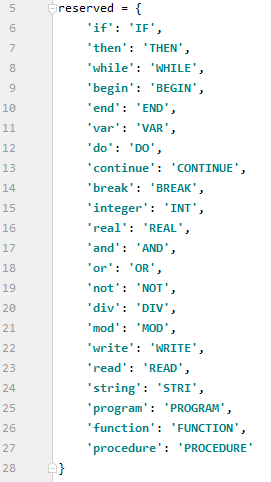
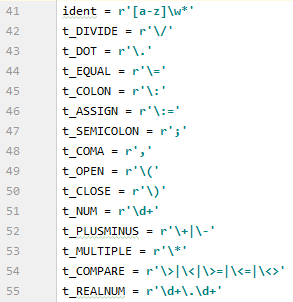
* Требования к входному языку:

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 32 бита, вещественный 32 бита).

## Лексический анализатор

Лексический анализатор был реализован при помощи библиотеки ply. Для его работы необходим набор зарезервированных слов и набора регулярных выражений, при помощи которых лексер будет разбивать исходный код программы на токены

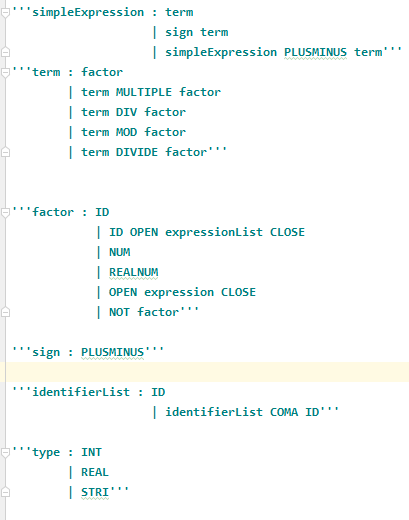
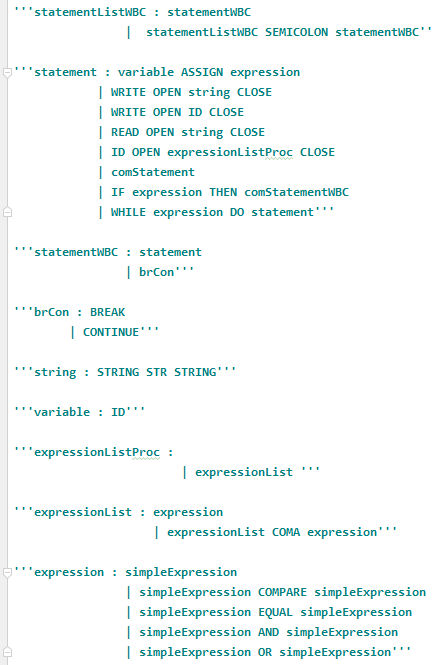
*рис. 1 Список зарезервированных слов*

*рис. 2 Регулярные выражения токенов*

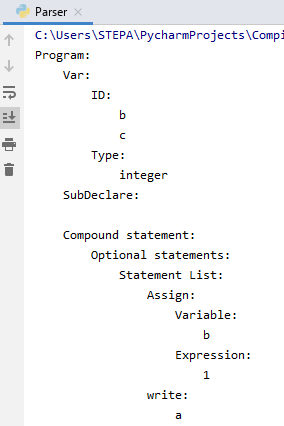
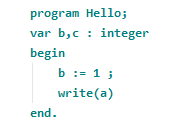
## Синтаксический анализатор

Следующим этапом курсовой работы является реализация синтаксического анализатора. Для данного этапа работы была использована библиотека ply. На вход синтаксическому анализатору подается исходный код, поделенный на токены лексером. Результатом работы синтаксического анализатора является синтаксическое дерево разбора. Ниже представлена КС грамматика, по которой синтаксический анализатор обрабатывает поступающие токены (также находится в файле Parser.py) :

*рис 3. КС грамматика языка ч.1*

*рис 4. КС грамматика языка ч.2*

*рис. 5 КС грамматика языка ч.3*

В итоге работы синтаксического анализатора получаем синтаксическое дерево разбора, пример которого представлен ниже:

*рис. 6 Пример синтаксического дерева разбора*

**=**

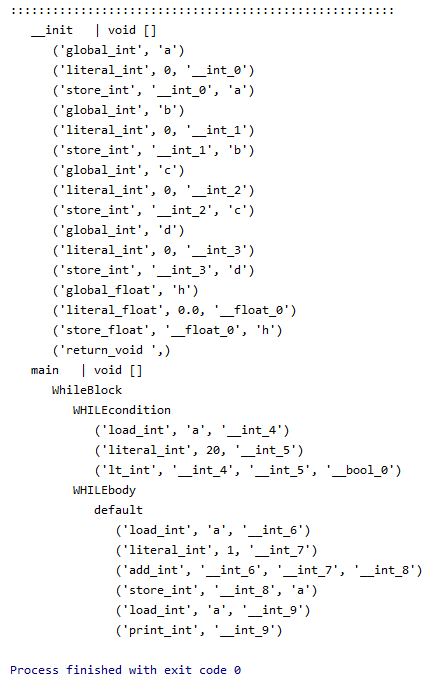
## Таблица символов

## Для генерации таблицы символов используется обход синтаксического дерева, при котором тип аргументов и переменных записываются в словарь.

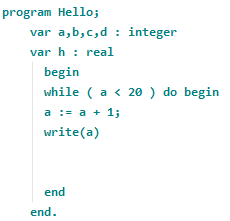
## 

## *рис. 7 Пример символьной таблицы*

## Построение генератора промежуточного кода

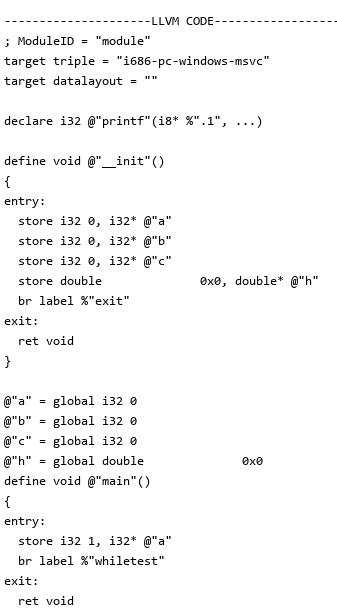
Чтобы сгенерировать промежуточный код мною был реализован алгоритм обхода синтаксического дерева, который встречая определенные узлы дерева создавал для них трехадресные инструкции. Для хранения трехадресного кода был реализован класс Block, который хранит в себе списки инструкций, название блока инструкций, его возвращаемый тип и параметры (если блок используется для хранения трехадресного кода функции) и может состоять из других вложенных в него блоков. На рисунке 8 представлен сгенерированный трехадресный код для программы из рисунка 9.

*рис.8 Пример генерации трехадресного кода*

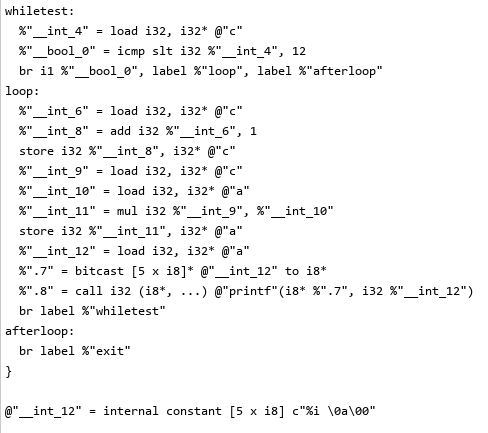
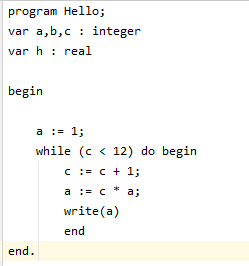


*рис. 9 Пример программы*

## 1.6 Построение генератора объектного кода

Для реализации трансляции кода в машинный код мною была использована библиотека llvmlite. Получая трехадресные инструкции из генератора трехадресного кода, транслятор обрабатывает их и создает файл с промежуточным кодом расширения ll.

*рис. 10 пример промежуточного кода*



*рис. 12 Исходный код программы*

*рис. 11 пример промежуточного кода*

# **ПРОВЕРКА НА СООТВЕТСТВИЕ ОСНОВНЫМ ТРЕБОВАНИЯМ**

Пример №1 выводит первые a чисел последовательности Фибоначчи

показывает выполнение следующих требований:

- Операторы присваивания

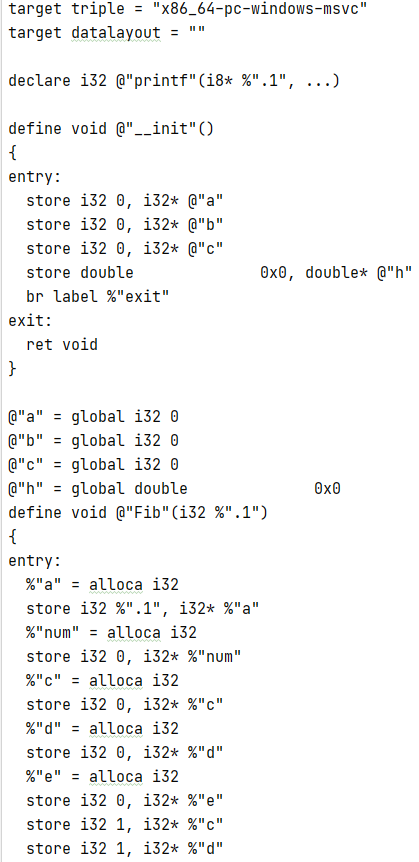
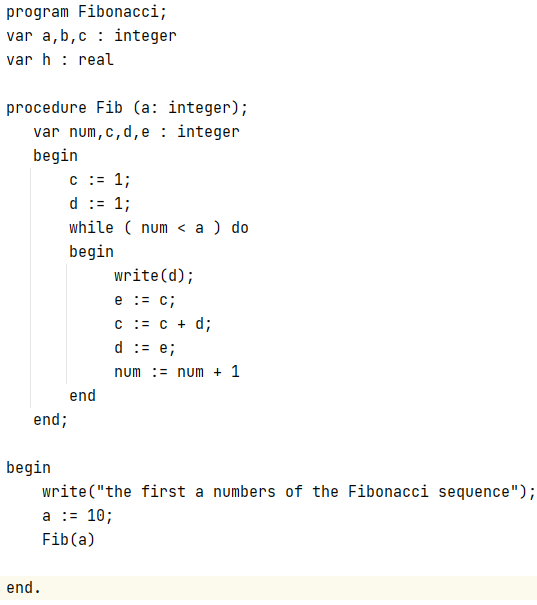
- Арифметика

- Операторы цикла (while)

- Базовый вывод переменных и строковых литералов

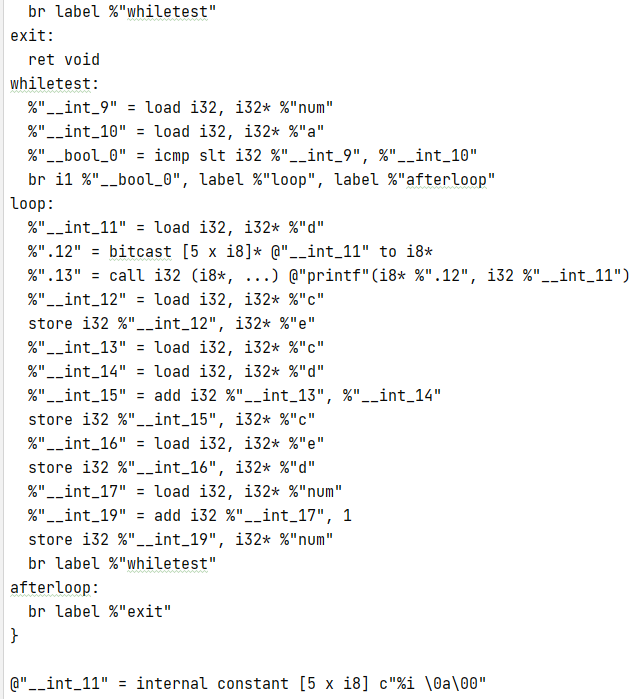
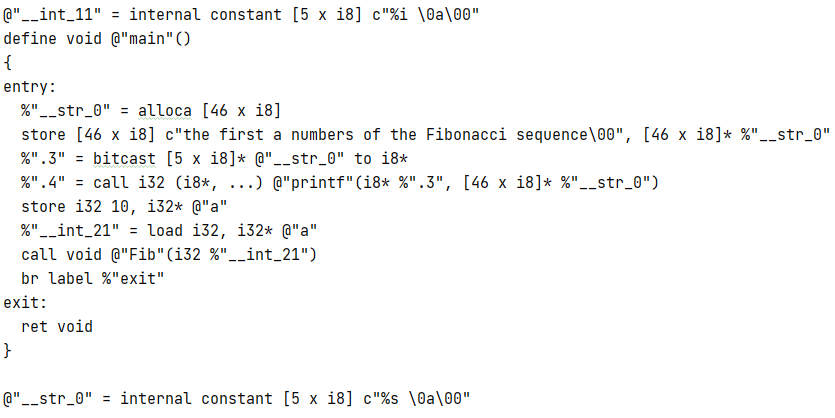
- Поддержка вызова функций и процедур

- Игнорирование индентации программы

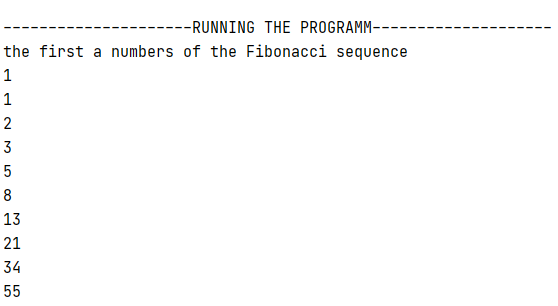
- Операторные скобки (begin end).

*рис. 13 Код исходной программы*

*рис. 14 Сгенерированный объектный код llvm ч.1*

*рис. 14 объектный код ч.2*

*рис. 16 объектный код ч.3*



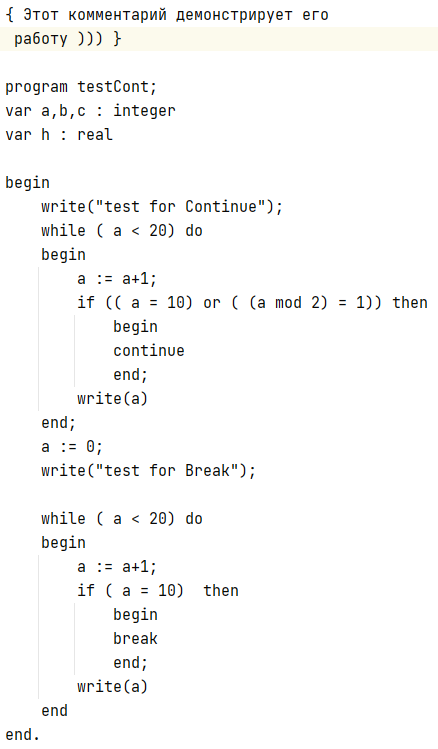
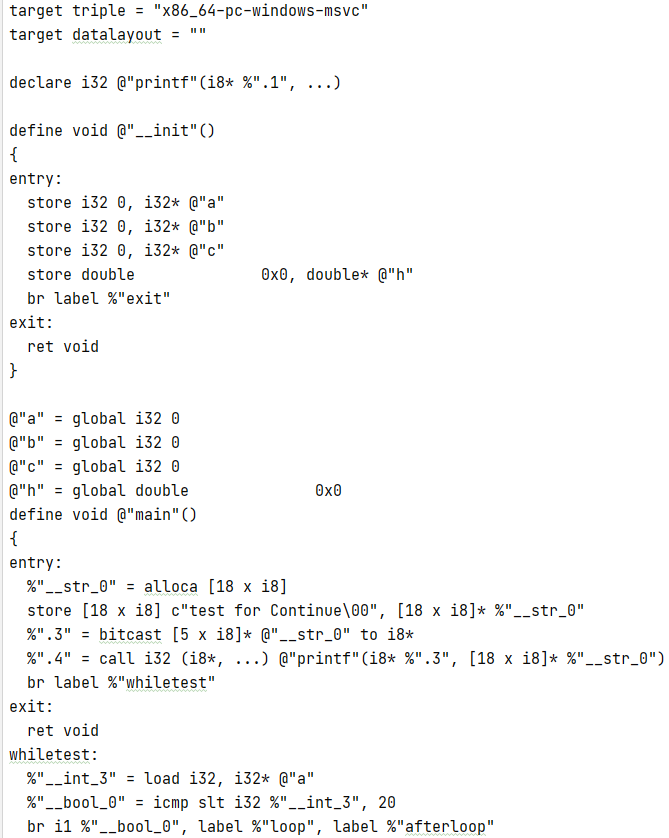
*рис. 17 Результат компиляции Примера №1*

Пример №2 выводит только четные числа и игнорирует число 10. Пример показывает выполнение следующих требований, которые не показаны в примере №1:

- Условный оператор ЕСЛИ

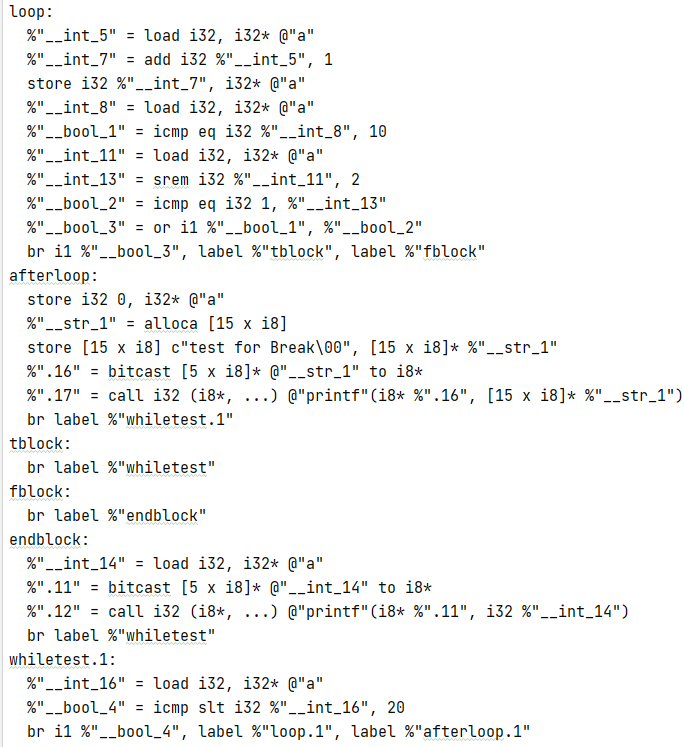
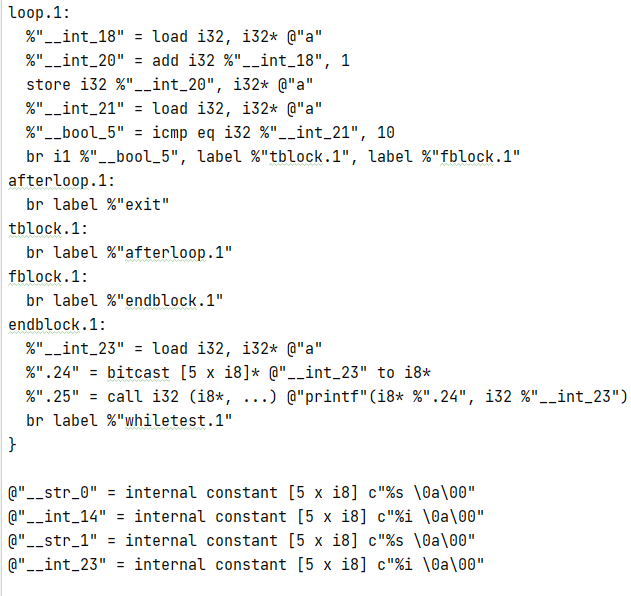
- Операторы цикла (while, break , continue)

- Комментарии любой длины

- Логические операции

*рис. 19 Объектный код ч.1*

*рис. 18 Исходный код программы*



*рис. 21 Объектный код ч.3*

*рис. 20 Объектный код ч.2*

# **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. Компилятор на Python под LLVM с использованием библиотек PLY и LLVMLITE // GitHub URL: https://github.com/emmett9001/gone-compiler (дата обращения: 13.05.21).