**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 LLVM**

Пояснительная записка

На 12 листах

|  |  |  |
| --- | --- | --- |
| Руководитель |  | Ю. М. Монахов |
| Исполнитель |  | студент гр. ИБ-121 Д.И. Махров |
|  |  |  |

Владимир 2024

# АННОТАЦИЯ

В данном программном документе приведён код компилятора подмножества процедурного языка. Код программы реализован на языке программирования Python 3.10.6 в интегрированной среде разработки PyCharm.

Используемые библиотеки:

* Стандартные (abc, enum, json, time, typing, re).
* Подключаемые (llvmlite).

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

1. Проектирование лексического анализатора.
2. Построение синтаксического анализатора.
3. Построение символьной таблицы.
4. Проектирование генератора объектного кода.
5. Построение оптимизатора.

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, while do, for).

6. Базовый вывод (строковый литерал, переменная).

7. Типы (целочисленный 32 бита, с плавающей запятой 32 бита).

Требования к выходному языку:

1. Перевод в LLVM.

ОГЛАВЛЕНИЕ

Оглавление

[АННОТАЦИЯ 2](#_Toc168503212)

[1 ЛЕКСИЧЕСКИЙ АНАЛИЗАТОР 4](#_Toc168503213)

[2 СИНТАКСИЧЕСКИЙ АНАЛИЗАТОР 5](#_Toc168503214)

[3 ГЕНЕРАТОР ЦЕЛЕВОГО КОДА 8](#_Toc168503215)

[4 ОПТИМИЗАТОР 9](#_Toc168503216)

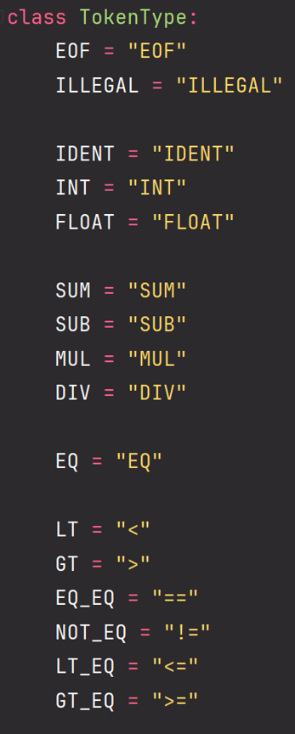
[5 РАБОТОСПОСОБНОСТЬ 11](#_Toc168503217)

[6 КОД 12](#_Toc168503218)

# 1 ЛЕКСИЧЕСКИЙ АНАЛИЗАТОР

На начальном этапе содержимое текстового файла code.txt разбивается на токены в соответствии с описанными правилами лексического анализатора. Результатом работы является последовательный список всех найденных лексем.

Лексический анализатор определяет такие токены как операторы присваивания, цикла, условные, логические, арифметические, также типы и функции, токен обозначающий конец файла и токен недопустимого символа.



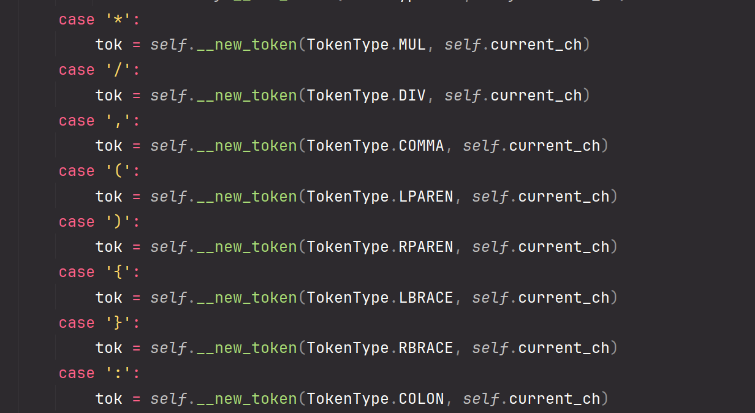


Рисунок 1,2 – Разбор лексем

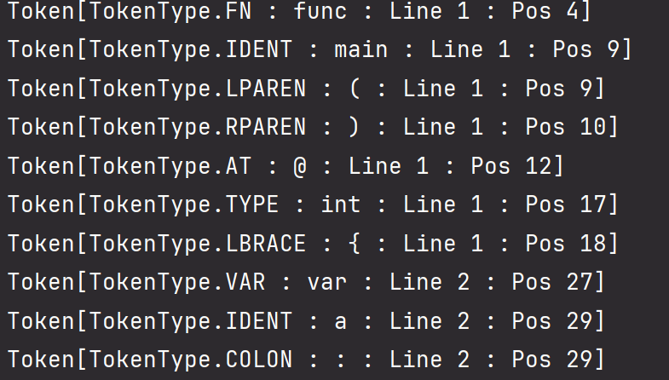


Рисунок 3 – Полученные токены

# 2 СИНТАКСИЧЕСКИЙ АНАЛИЗАТОР

Синтаксический анализатор принимает поток токенов и сопоставляет их с правилами грамматики.

Для каждого токена описаны определённые наборы правил, по котором синтаксический анализатор их может обработать, и в случае нарушения пользователем этих правил выдать ошибку. Итоговым результатом работы парсера является синтаксическое дерево разбора, которое в дальнейшем отправляется генератору кода.



Рисунок 4 – Функции обработки токенов

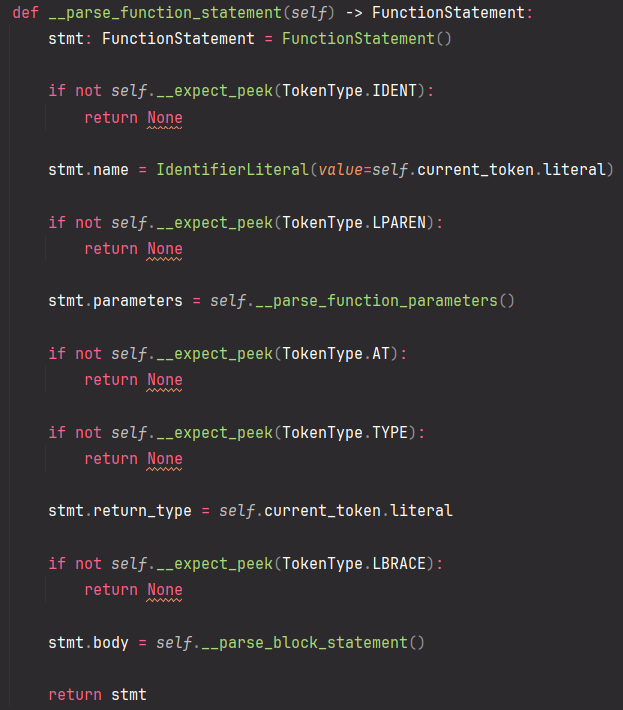
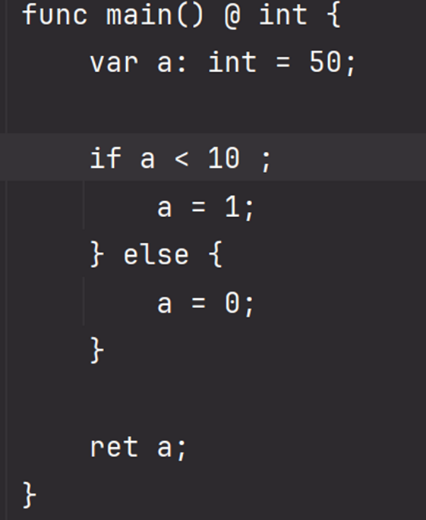
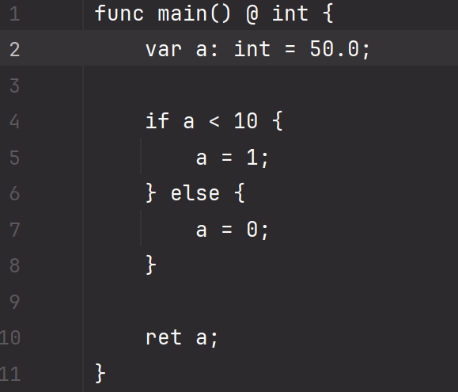


Рисунок 5 – Пример разбора объявления функции



Рисунок 6 – Полученное синтаксическое дерево

Синтаксические ошибки обрабатываются с помощью функции проверки следующего токена на синтаксические правила, семантические ошибки «‎поднимаются» на этапе генерации кода





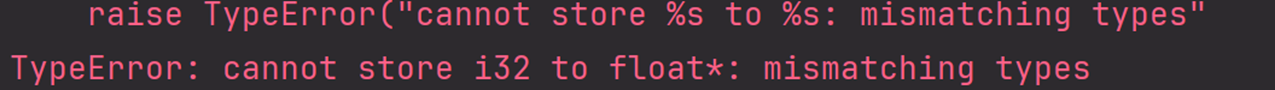


Рисунок 7 – Синтаксические и семантические ошибки

# 3 ГЕНЕРАТОР ЦЕЛЕВОГО КОДА

Выходные данные из синтаксического анализатора передаются в генератор целевого кода, который продолжает обработку при помощи библиотеки «llvmlite». Итогом работы является файл code.ll, в который записывается промежуточный код «LLVM IR».

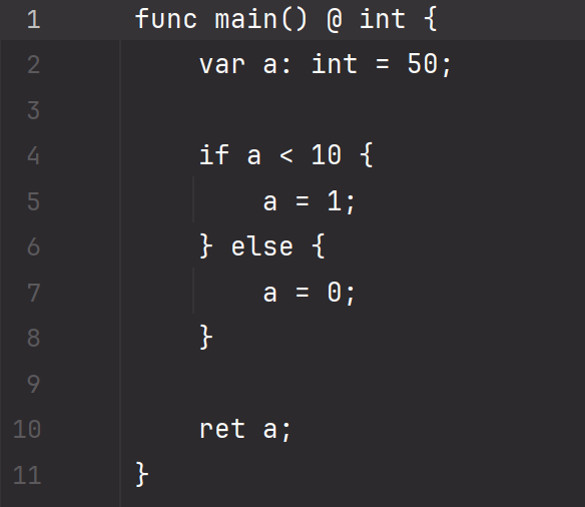


Рисунок 8 – Код исходной программы

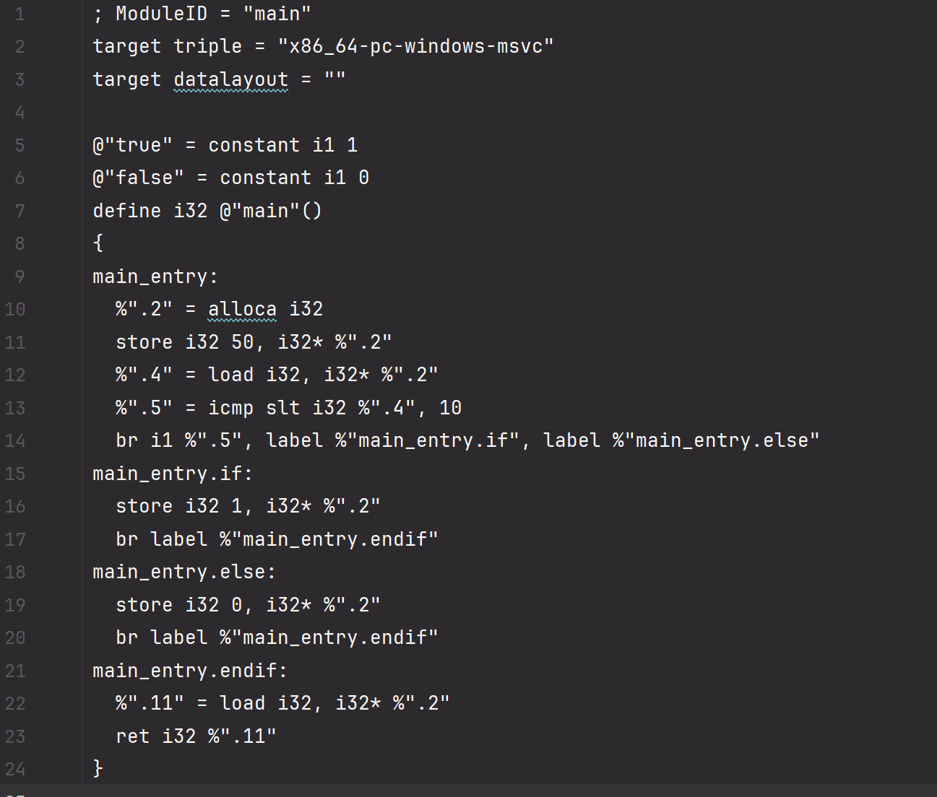


Рисунок 9 – Промежуточное представление

# 4 ОПТИМИЗАТОР

Оптимизация программы является важным этапом при разработке компилятора, так как более эффективного используются ресурсы процессора, уменьшается размер кода, улучшается время компиляции и т.д. В рамках курсовой работы были написаны такие оптимизационные функции как: проблема неиспользуемых функций и проблема переопределения переменных.

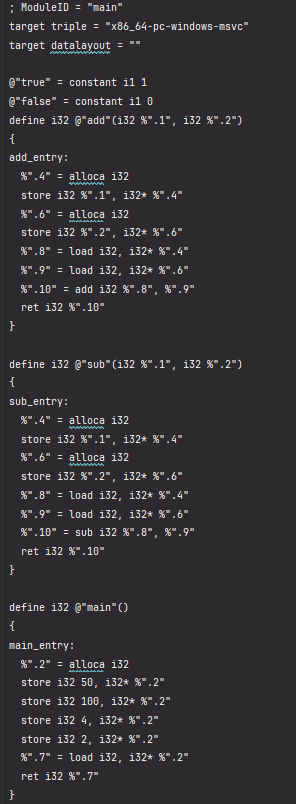


Рисунок 10 – Неоптимизированный код

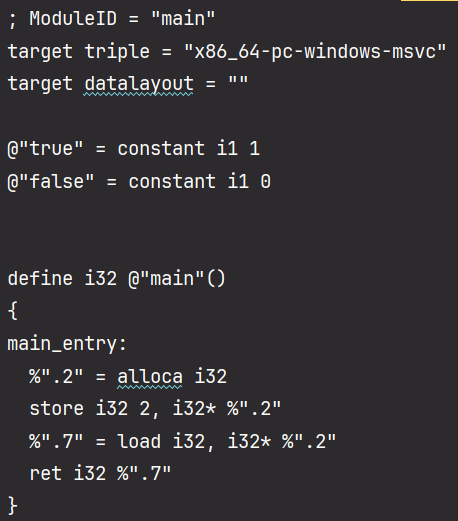


Рисунок 11 – Оптимизированный код

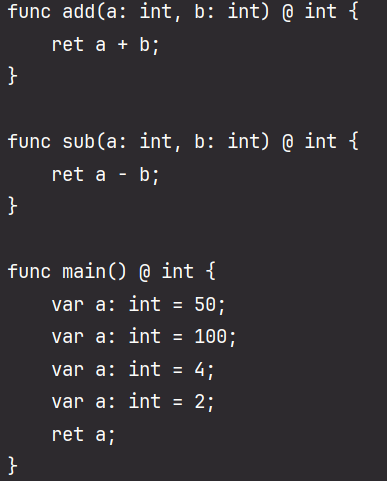


Рисунок 12 – Исходный код программы.

# 5 РАБОТОСПОСОБНОСТЬ

В качестве проверки работоспособности программы был написан рекурсивный алгоритм Фибоначчи:

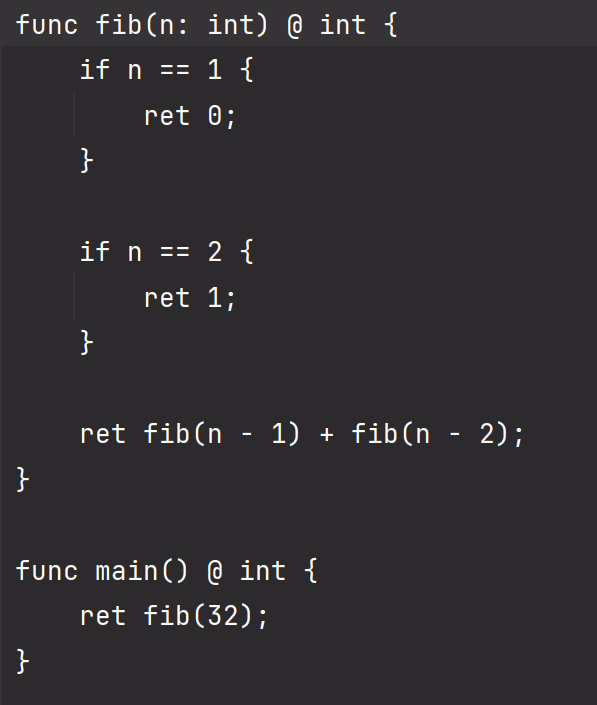


Рисунок 13 – Код исходной программы

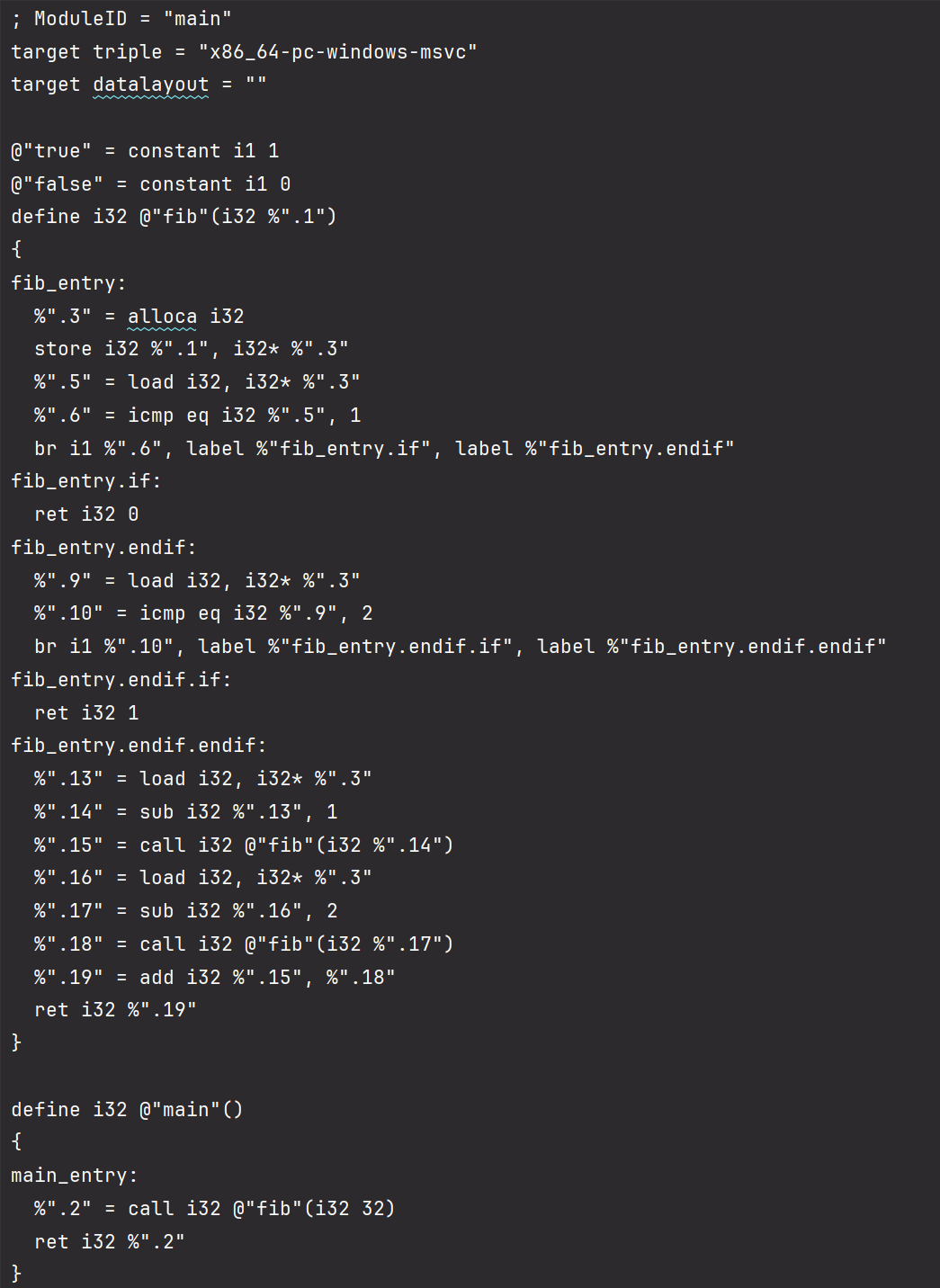


Рисунок 14 – Промежуточное представление

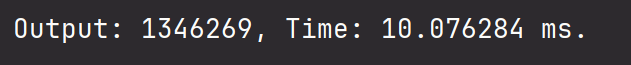


Рисунок 15 – Результат

# 6 КОД

Код работы доступен на Github: https://github.com/makhrov03/coursework6sem