Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Владимирский государственный университет

имени Александра Григорьевича и Николая Григорьевича Столетовых»

(ВлГУ)

Компилятор подмножества процедурного языка в ассемблер

Пояснительная записка

RU. 0000000.000000.0.00

#### АННОТАЦИЯ

В данном программном документе приведён код компилятора подмножества процедурного языка. Код программы реализован на языке программирования **Python**.

Разработанная программа состоит из лексического анализатора, синтаксического анализатора, генератора таблицы символов, генератора трёхадресного кода и генератора объектного кода.

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.

**СОДЕРЖАНИЕ**

1. Лексический анализатор ..4

2. Синтаксический анализатор ..6

3. Построение таблицы символов ..8

4. Генерация трёхадресного кода ..9

5. Генерация объектного кода 10

6. Проверка работоспособности компилятора..………………………………………..12

Лист регистрации изменений 14

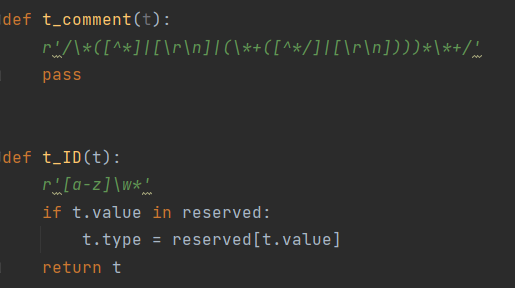
**1. ЛЕКСИЧЕСКИЙ АНАЛИЗАТОР**

Лексический и синтаксический анализатор реализован с помощью библиотек ply и re. В программе заранее определены лексемы и токены, которые в дальнейшем будут использоваться в синтаксическом анализаторе.

На начальном этапе работы компилятора содержимое файла программы ***TestCode2.ok*** разбивается на токены в соответствии с описанными правилами лексического анализатора. Результатом работы является последовательный список всех найденных лексем.

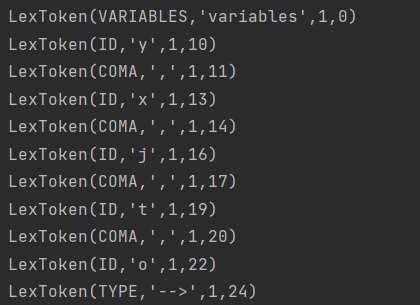
В файле конечной программы имеются операторы присваивания, цикла, условные и логические, арифметика, типы и функции. Также присутствуют многострочные комментарии, которые игнорируются при разборе.

На рисунке 1 представлен небольшой пример определения лексем внутри лексического анализатора с помощью регулярных выражений.



*Рисунок 1 – определение лексем*

На рисунке 2 представлен пример полученных токенов.



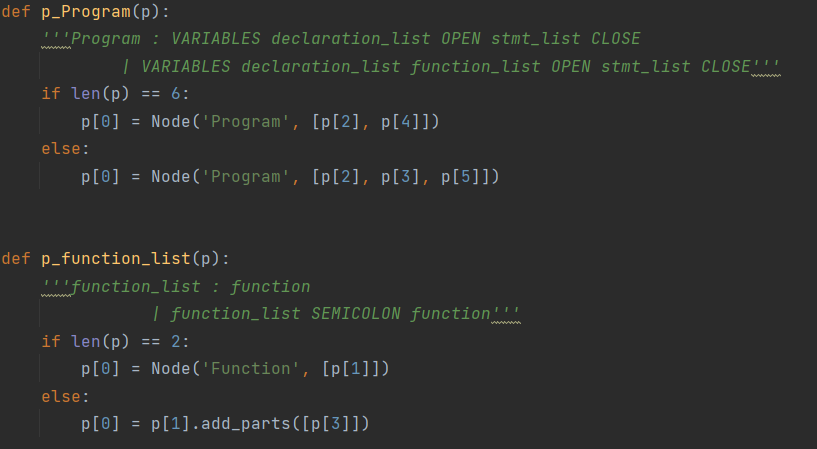
*Рисунок 2 – полученные токены*

1. **СИНТАКСИЧЕСКИЙ АНАЛИЗАТОР**

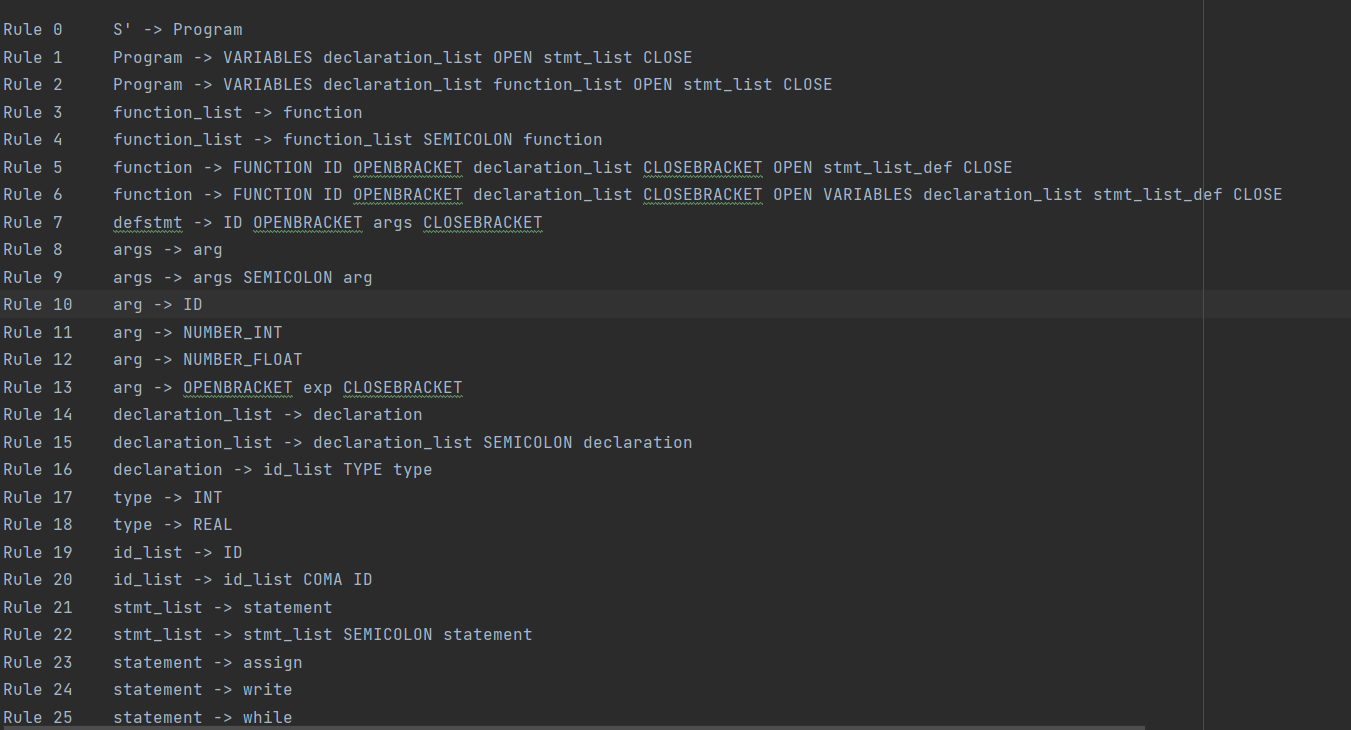
Второй стадией компилятора является синтаксический анализ. Когда синтаксический анализатор принимает поток токенов, то он сразу начинает сопоставлять их уже с новыми с правилами и строит дерево разбора грамматики.

Для каждого токена описаны определённые наборы правил, по котором синтаксический анализатор их может обработать. При помощи этих правил, токены преобразуются в синтаксическое дерево разбора. После завершения, итоговый результат отправляется транслятору для дальнейших преобразований.

На рисунке 3 представлено одно из правил обработки потока токенов.



*Рисунок 3 – правило обработки токенов*



*Рисунок 4 – грамматика языка*

На рисунке 5 изображён пример генерации дерева синтаксическим анализатором:

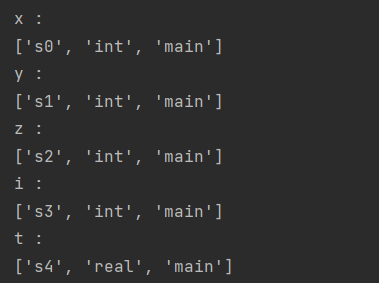
**

*Рисунок 5 – пример сгенерированного дерева*

1. **ПОСТРОЕНИЕ ТАБЛИЦЫ СИМВОЛОВ**

Третьей фазой разработкой компилятора является построение таблицы символов.

Она используется при дальнейшей трансляции. Содержит в себе все объявления переменных в конечной программе, область видимости и их типы. Строится обходом дерева в глубину, сформированным синтаксическим анализатором.



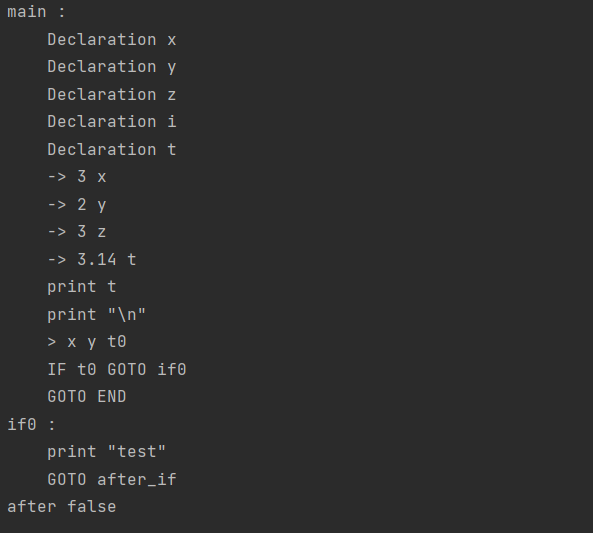
*Рисунок 6 – таблица символов*

1. **ГЕНЕРАЦИЯ ТРЁХАДРЕСНОГО КОДА**

Четвёртой фазой разработкой компилятора является построение таблицы символов.

Реализуется для более удобной трансляции в машинный код без дальнейшего обхода дерева. Содержит в себе записи о объявлении и инициализации переменных, вызове встроенных функций, условных блоков, временных переменных и т.д.

На рисунке 7 представлен трёхадресный код конечной программы.

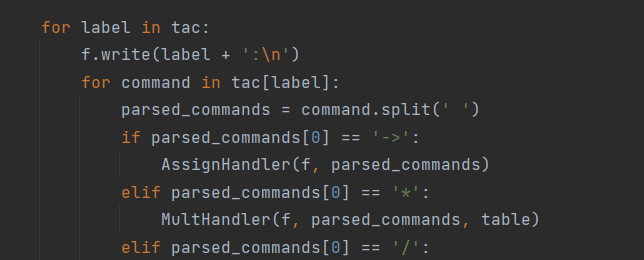
****

*Рисунок 7 – трёхадресный код*

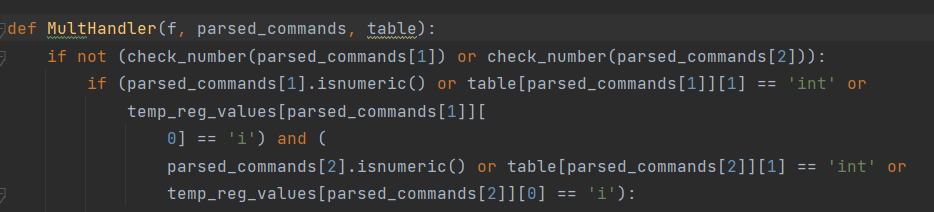
1. **ГЕНЕРАЦИЯ ОБЪЕКТНОГО КОДА**

Пятой фазой разработкой компилятора является генерация объектного кода. Конечной платформой является **MIPS**.

Реализуется с использованием таблицы символов и трёхадресного кода. С помощью алгоритма обхода словаря (трёхадресного кода) по заданным параметрам обрабатываются заранее определённые в алгоритме конструкции и с командами, включая нужные регистры, построчно записываются в файл вывода.

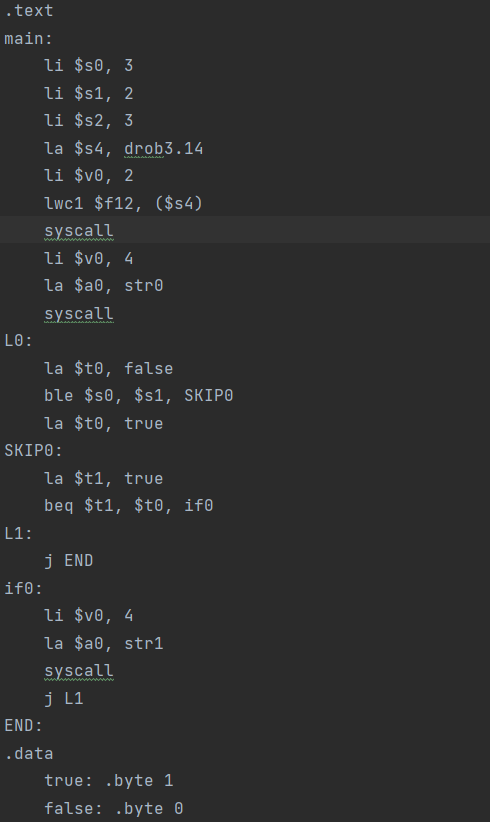


*Рисунок 8 – пример кода*

**

*Рисунок 9 – пример обработчика умножений*

На рисунке 10 представлена конечная компиляция программы на исходном языке в подмножества Assembler MIPS:

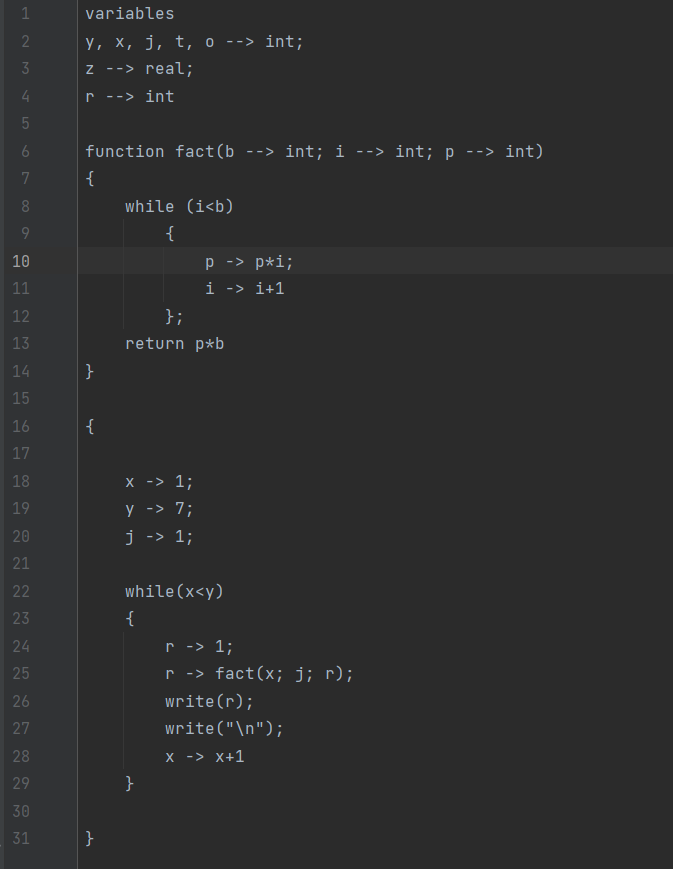


*Рисунок 10 – результат работы компилятор*

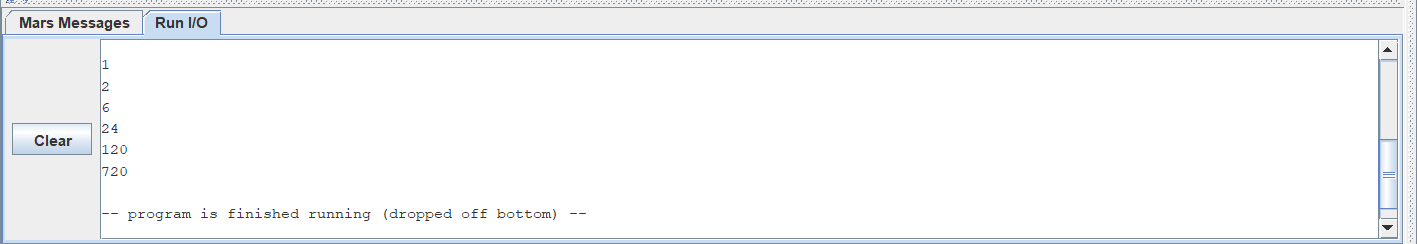
1. **ПРОВЕРКА РАБОТОСПОСОБНОСТИ КОМПИЛЯТОРА**

Ниже приведены примеры тестовых программ и работы компилятора.

**Факториал числа 6:**

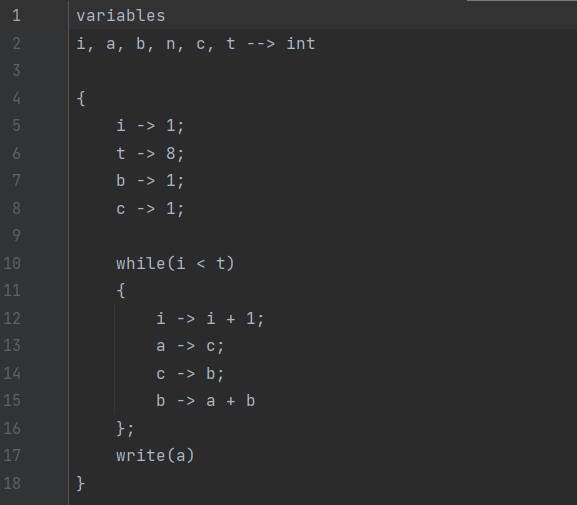


*Рисунок 11 – код программы*

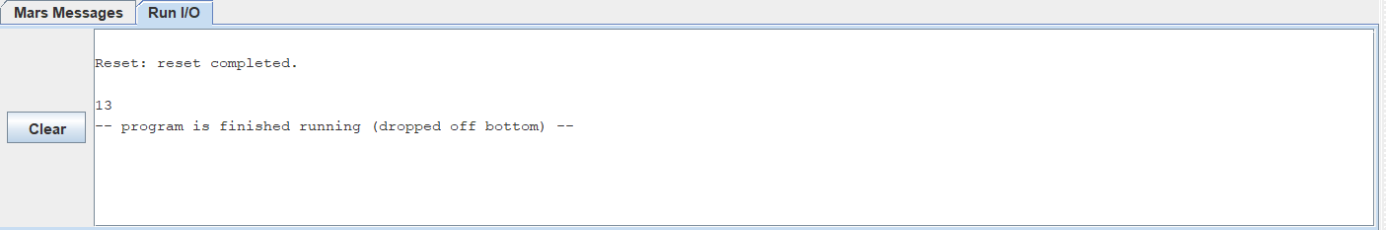


*Рисунок 12 – результат выполнения программы в MARS 4.5*

**По данному числу n (= 7) определить n-е число Фибоначчи:**

****

*Рисунок 13 – код программы*



*Рисунок 14 – результат выполнения программы в MARS 4.5*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | |
| Номера листов (страниц) | | | | | Всего  листов  (страниц)  в докум | №  документа | Входящий  № сопрово  дительного  документа  и дата | Подп. | Дата |
| Изм | изменен  ных | заме  ненных | новых | Анулиро  ванных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |