Министерство образования и науки 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. 02068048.502900.01.01

#### АННОТАЦИЯ

В данном программном документе приведён текст компилятора подмножества процедурно-ориентированного языка. Текст программы реализован на языке python в интегрированной среде разработки PyCharm Community edition 2020.1.2. Основная функция компилятора — это проверка исходной цепочки символов на принадлежность к входному языку и генерация выходной цепочки символов на языке машинных команд или ассемблере.

Можно выделить основные части компилятора подмножества процедурно-ориентированного языка:

1) лексический анализатор

2) синтаксический анализатор

3) Генератор кода

**СОДЕРЖАНИЕ**

СОДЕРЖАНИЕ 3

1 ПРОЕКТИРОВАНИЕ КОМПИЛЯТОРА 4

1.1 ОСНОВНЫЕ ТРЕБОВАНИЯ 4

1.2 СОЗДАНИЕ ЛЕКСИЧЕСКОГО АНАЛИЗАТОРА 4

1.3 РАЗРАБОТКА СИНТАКСИЧЕСКОГО АНАЛИЗАТОРА 11

1.4 РАЗРАБОТКА ТРАНСЛЯТОРА 12

2 ОБЩАЯ ИНФОРМАЦИЯ И ТЕСТЫ 14

2.1 УПАКОВКА ПРИЛОЖЕНИЯ 15

2.2 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 15

2.3 ТЕСТЫ 15

3 ЗАКЛЮЧЕНИЕ 15

4 ПРИЛОЖЕНИЕ 15

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ 16

**1 ПРОЕКТИРОВАНИЕ КОМПИЛЯТОРА**

**1.1 Основные требования**

Разработка будет производиться в соответствии со следующими требованиями:

* Требования к входному языку:

1. Должны присутствовать операторные скобки;
2. Должна игнорироваться идентация 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, логические операторы;
3. Условный оператор;
4. Операторы цикла (break, continue);
5. Базовый вывод;
6. Типы (целочисленный, вещественный).

* Требования к выходному языку:

1. Ассемблер для архитектуры x86.
   1. **Создание лексического анализатора**

Лексический анализатор является первой фазой работы компилятора. Работа лексического анализатора основана на регулярных выражениях. На вход подаются строки исходной программы. В самом анализаторе присутствует набор функций для получения результата анализа строки. В результате работы лексера и парсера на выходе получается дерево разбора. Для генерации оных используются библиотека, грамматика и плагин для языка pl0 от ANTLR4.

В Листинге 1 приведен список регулярных выражений.

grammar pl0;

program

: block '.'

;

block

: consts? var? procedure\* statement

;

consts

: CONST ident '=' number (',' ident '=' number)\* ';'

;

var

: VAR ident (',' ident)\* ';'

;

procedure

: PROCEDURE ident ';' block ';'

;

statement

: (assignstmt | callstmt | writestmt | qstmt | bangstmt | beginstmt | ifstmt | whilestmt)?

;

assignstmt

: ident ':=' expression

;

callstmt

: CALL ident

;

writestmt

: WRITE ident

;

qstmt

: '?' ident

;

bangstmt

: '!' expression

;

beginstmt

: BEGIN statement (';' statement)\* END

;

ifstmt

: IF condition THEN statement

;

whilestmt

: WHILE condition DO statement

;

condition

: ODD expression

| expression ('=' | '#' | '<' | '<=' | '>' | '>=') expression

;

expression

: ('+' | '-')? term (('+' | '-') term)\*

;

term

: factor (('\*' | '/') factor)\*

;

factor

: ident

| number

| '(' expression ')'

;

ident

: STRING

;

number

: NUMBER

;

WRITE

: W R I T E

;

WHILE

: W H I L E

;

DO

: D O

;

IF

: I F

;

THEN

: T H E N

;

ODD

: O D D

;

BEGIN

: B E G I N

;

END

: E N D

;

CALL

: C A L L

;

CONST

: C O N S T

;

VAR

: V A R

;

PROCEDURE

: P R O C E D U R E

;

fragment A

: ('a' | 'A')

;

fragment B

: ('b' | 'B')

;

fragment C

: ('c' | 'C')

;

fragment D

: ('d' | 'D')

;

fragment E

: ('e' | 'E')

;

fragment F

: ('f' | 'F')

;

fragment G

: ('g' | 'G')

;

fragment H

: ('h' | 'H')

;

fragment I

: ('i' | 'I')

;

fragment J

: ('j' | 'J')

;

fragment K

: ('k' | 'K')

;

fragment L

: ('l' | 'L')

;

fragment M

: ('m' | 'M')

;

fragment N

: ('n' | 'N')

;

fragment O

: ('o' | 'O')

;

fragment P

: ('p' | 'P')

;

fragment Q

: ('q' | 'Q')

;

fragment R

: ('r' | 'R')

;

fragment S

: ('s' | 'S')

;

fragment T

: ('t' | 'T')

;

fragment U

: ('u' | 'U')

;

fragment V

: ('v' | 'V')

;

fragment W

: ('w' | 'W')

;

fragment X

: ('x' | 'X')

;

fragment Y

: ('y' | 'Y')

;

fragment Z

: ('z' | 'Z')

;

STRING

: [a-zA-Z] [a-zA-Z]\*

;

NUMBER

: [0-9] +

;

WS

: [ \t\r\n] -> skip;

* 1. **Разработка синтаксического анализатора**

Вторая фаза компилятора – это синтаксический анализ. На вход синтаксического анализатора подаются строки из первичного дерева. Данный компонент принимает список токенов на входе и создает на выходе дерево разбора. Как было описано выше, здесь используется библиотека/плагин ANTLR4. При помощи двух компонентов парсера создаём работающий компилятор, интерпретирующий язык. Метод parse считывает изначальный токен из списка и сравнивает его с ожидаемым шаблоном объявления программы, и далее управление передается правилу program (). В Листинге 2 приведен код элемента парсера.

Листинг 2- Код элемента синтаксического анализатора

def program(self):

localctx = pl0Parser.ProgramContext(self, self.\_ctx, self.state)

self.enterRule(localctx, 0, self.RULE\_program)

try:

self.enterOuterAlt(localctx, 1)

self.state = 40

self.block()

self.state = 41

self.match(pl0Parser.T\_\_0)

except RecognitionException as re:

localctx.exception = re

self.\_errHandler.reportError(self, re)

self.\_errHandler.recover(self, re)

finally:

self.exitRule()

return localctx

На рисунке 1 показан результат работы синтаксического анализатора.

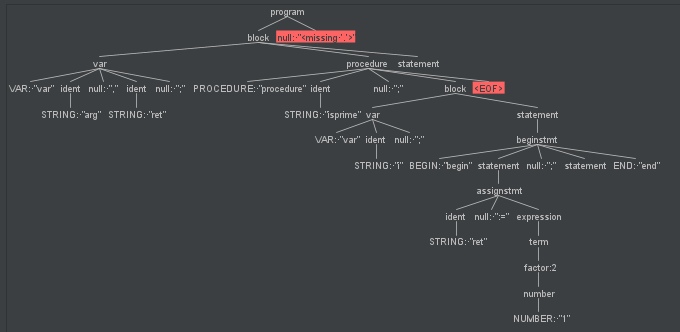


Рис.1

**1.4Транслятор**

При проектировании транслятора использовалась библиотека llvmlite, которая представляет возможность для создания промежуточного кода ir-llvm.  
Частичный исходный код инструкций:  
class GeneratedIR():  
reg = 1  
  
header\_text = ""  
main\_text = ""  
buffer = ""  
str\_i = 0  
main\_reg = 1  
br = 0  
stack\_pop = True  
  
br\_stack = []  
br\_loop = []  
  
def String\_generate(self):  
self.main\_text += self.buffer  
self.formatMainText()  
text = ""  
text += "\n"  
text += self.header\_text  
text += "define i32 @main() nounwind {\n"  
text += str(self.main\_text)  
text += " ret i32 0\n"  
text += "}\n"  
file = open("../lexanalyz/output.ll", 'w')  
if file:  
file.write(text)  
else:  
print("String\_generate ERROR!")  
return text  
  
def function\_start(self, id):  
self.main\_text += self.buffer  
main\_reg = self.reg  
self.buffer = "define main @ " + id + " () nounwind {\n"  
self.reg = 1  
  
def function\_end(self):  
self.buffer += "ret void\n"  
self.formatself()  
self.buffer += "}\n\n"  
self.header\_text += str(self.buffer)  
self.buffer = ""  
self.reg = self.main\_reg  
  
def call(self, id):  
self.buffer += "call void @" + id + "()\n"  
  
Также в этом модуле происходит обьединение всех используемых инструкций в один файл \*.ll

**2. ОБЩАЯ ИНФОРМАЦИЯ И ТЕСТЫ**

# 2.1 Упаковка приложения

Приложение находится в папке src/ в виде нескольких файлов python-проекта, с подкаталогом gen, где лежат файлы парсера и лексера, и отдельно от главной папки библиотека antlr-ast.

# 2.2 Руководство пользователя

Для запуска компилятора необходимо заполнить файл test.txt тестовой программой. Далее запустить файл Main.py. После этого будет создан файл output.ll, содержащий в себе промежуточный ir-llvm код.

# 2.3 Тесты

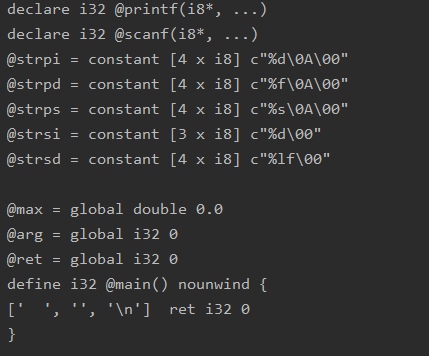
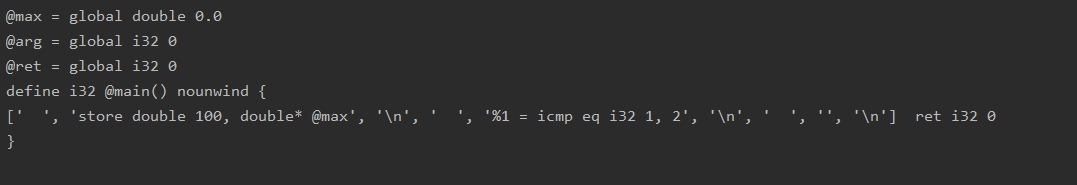


Рис.2 Объявление переменных

Рис.3 Операция сложения

**3.ЗАКЛЮЧЕНИЕ**

В данной программе был реализован лексер и парсер подмножества языка pl0. В результате работы возникало множество сложностей, в следствие которых задача могла быть реализована не в срок и в недостаточном объеме.

**4.ПРИЛОЖЕНИЕ**

<https://github.com/Beesmark/Compiler>

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |