**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

Факультет инфокоммуникационных сетей и систем (ИКСС)

Лабораторная работа №6

по курсу

«Теория автоматов и формальных языков»

Группа: ИКПИ-14

Выполнил студент: Хохлов Т. В.

Принял преподаватель: Леонова М.Д.

Место для подписи \_\_\_\_\_\_\_\_\_\_\_\_\_

Санкт-Петербург

2023 г.

**Задание:** реализовать синтаксический анализатор для проверки правильности конструкции цикла с условием (while) на выбранном языке программирования. Произвести разбор входной строки на лексемы, провести синтаксический анализ и проверить соответствие строки правилам грамматики.

Синтаксический анализ — процесс сопоставления последовательности токенов с правилами грамматики. В результате анализа строится структура данных, представляющая структуру входной строки, либо обнаруживаются ошибки в синтаксисе.

**Постановка задачи**

1. Составить грамматику, описывающую синтаксис цикла с условием (while).
2. Разработать лексический и синтаксический анализатор для разбора кода с использованием созданной грамматики.
3. Проверить, соответствует ли входная строка правилам грамматики для цикла while.

**Грамматика цикла while**

<S> ::= while (<condition>) { <body> }

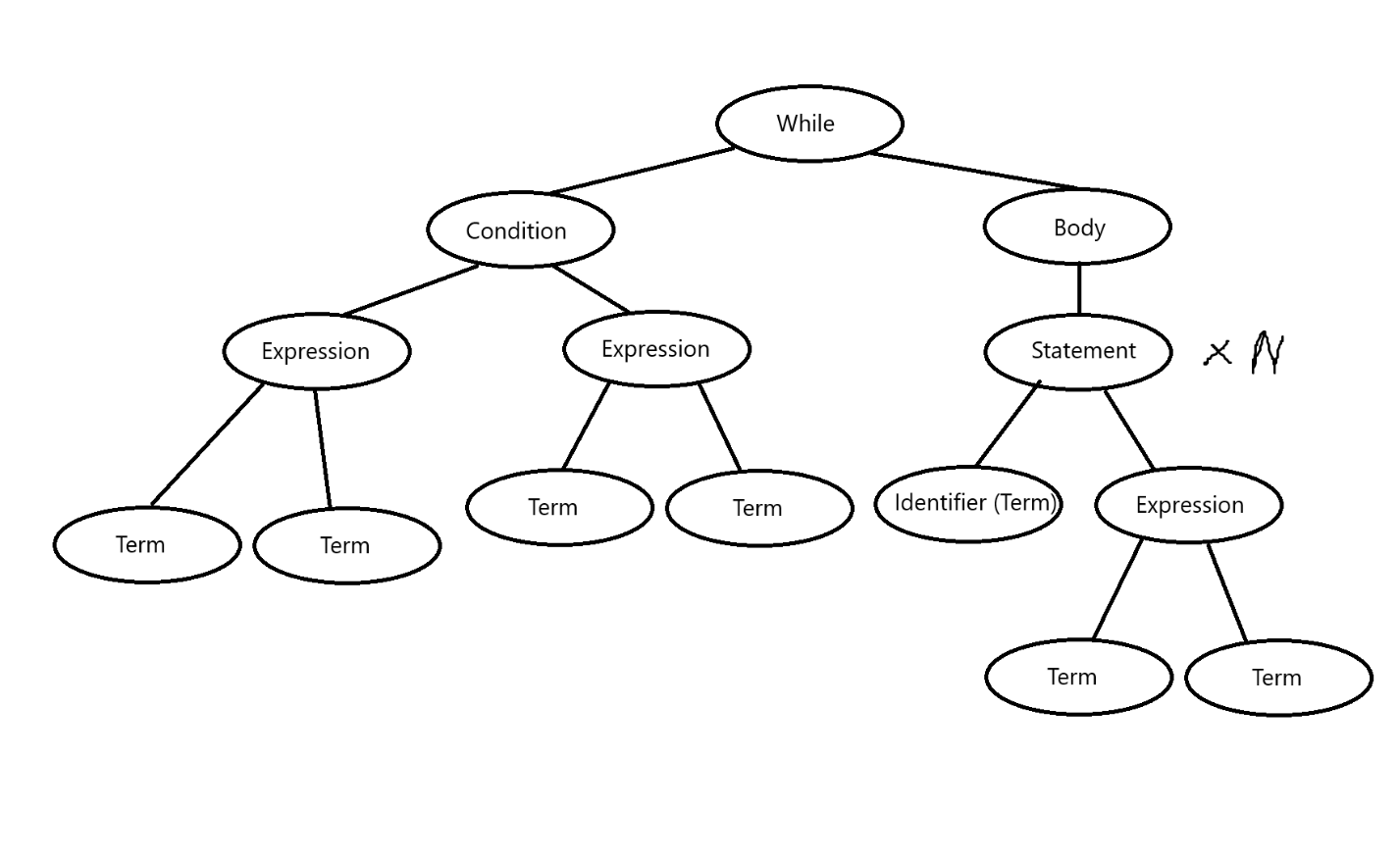
<condition> ::= <expression> (== | != | < | > | <= | >=) <expression>

<expression> ::= <term> ( ("+" | "-") <term> )\*

<term> ::= <identifier> | <number>

<body> ::= <statement> x N

<statement> ::= <identifier> ("=" | "+=" | "-=" | "\*=" | "/=") <expression> ;



Здесь <S> — это правило, описывающее цикл while, с условием в круглых скобках и телом цикла в фигурных. <condition> описывает условие с использованием операторов сравнения, а <body> — тело цикла, состоящее из одного или нескольких выражений.

**Код программы**

import re

token\_specification = [

('preprocessor', r'#\b(?:include|define|if|endif|ifdef|ifndef|pragma)\b'),

('header', r'<[A-Za-z\_][A-Za-z0-9\_]\*>'),

('keyword', r'\b(?:if|else|while|for|return|int|float|double|char|bool|void|class)\b'),

('WHILE', r'while'),

('LBRACKET', r'\('),

('RBRACKET', r'\)'),

('LBRACE', r'\{'),

('RBRACE', r'\}'),

('OP', r'==|!=|<=|>=|<|>'),

('ID', r'[A-Za-z\_][A-Za-z0-9\_]\*'),

('NUM', r'\b\d+(\.\d\*)?|\b0x[0-9A-Fa-f]+\b'),

('literal\_string', r'"(?:\\.|[^"\\])\*"|\'(?:\\.|[^\'\\])\*'),

('ASSIGN', r'='),

('COMPOUND\_ASSIGN', r'\+=|-=|\\*=|/='),

('SEMICOLON', r';'),

('ARITH\_OP', r'[+\-\*/]'),

('separator', r'[{}();,]'),

('comment', r'\/\\*[\s\S]\*?\\*\/|\/\/.\*'),

('WHITESPACE', r'\s+'),

]

token\_regex = '|'.join(f'(?P<{name}>{pattern})' for name, pattern in token\_specification)

get\_token = re.compile(token\_regex)

def lex\_analysis(text):

tokens = []

for match in re.finditer(get\_token, text):

token\_type = match.lastgroup

token\_value = match.group(token\_type)

if token\_type != 'WHITESPACE' and token\_type != 'comment':

tokens.append((token\_type, token\_value))

return tokens

**Синтаксический анализатор**

class Parser:

def \_\_init\_\_(self, tokens):

self.tokens = tokens

self.pos = 0

def consume(self, expected\_type):

if self.pos < len(self.tokens) and self.tokens[self.pos][0] == expected\_type:

self.pos += 1

else:

raise SyntaxError(f"Expected {expected\_type} at position {self.pos}")

def parse(self):

self.while\_stmt()

if self.pos < len(self.tokens):

raise SyntaxError("Extra input after end of while statement")

def while\_stmt(self):

self.consume('WHILE')

self.consume('LBRACKET')

self.condition()

self.consume('RBRACKET')

self.consume('LBRACE')

self.body()

self.consume('RBRACE')

def condition(self):

self.expression()

self.consume('OP')

self.expression()

def expression(self):

self.term()

while self.pos < len(self.tokens) and self.tokens[self.pos][0] in ('ARITH\_OP', 'COMPOUND\_ASSIGN'):

self.pos += 1

self.term()

def term(self):

if self.pos < len(self.tokens) and self.tokens[self.pos][0] in ('ID', 'NUM'):

self.pos += 1

else:

raise SyntaxError(f"Expected identifier or number at position {self.pos}")

def body(self):

while self.pos < len(self.tokens) and self.tokens[self.pos][0] != 'RBRACE':

self.statement()

def statement(self):

self.consume('ID')

if self.tokens[self.pos][0] == 'COMPOUND\_ASSIGN':

self.consume('COMPOUND\_ASSIGN')

else:

self.consume('ASSIGN')

self.expression()

self.consume('SEMICOLON')

**Тестовый код для проверки:**

input\_code = """

while (x != a) {

x -= 2;

}

"""

tokens = lex\_analysis(input\_code)

# Вывод токенов

print("Токены:")

for i, token in enumerate(tokens, start=1):

print(f"{i}: {token}")

# Запуск синтаксического анализатора

parser = Parser(tokens)

try:

parser.parse()

print("Строка соответствует грамматике цикла while.")

except SyntaxError as e:

print(f"Синтаксическая ошибка: {e}")

**Пример работы программы:**

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

**Вывод:**

Разработанный синтаксический анализатор успешно проверяет соответствие строки правилам грамматики для цикла while. Лексический анализатор корректно разделяет код на токены, пропуская пробелы и комментарии. Синтаксический анализатор успешно определяет, соответствует ли структура кода правилам, заданным грамматикой.