|  |  |
| --- | --- |
| Gerb-BMSTU_01 | ***«*Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ \_\_\_ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ\_\_\_\_\_\_\_\_

КАФЕДРА \_\_\_\_\_КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ОТЧЕТ**

**По домашнему заданию № 2**

**Дисциплина: Машинно-зависимые языки и основы компиляции**

**Название лабораторной работы: Лексические и синтаксические анализаторы**

Вариант 2.21

Студент гр. ИУ6-42Б  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.С. Твердюк**

(Подпись, дата) (И.О. Фамилия)

Преподаватель  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С.С. Данилюк**

(Подпись, дата) (И.О. Фамилия)

Москва, 2023

***Цель работы:*** закрепление знаний теоретических основ и основных методов приемов разработки лексических и синтаксических анализаторов регулярных и контекстно-свободных формальных языков.

***Ход работы:***

***Задание.*** Разработать грамматику и распознаватель описания языка программирования C++, включающего оператор присваивания и оператор Switch. Считать, что выбор происходит по значению переменной целого типа, альтернативы определяются константами целого типа, а оператор присваивания в правой части содержит только идентификаторы или целые константы. Например:

**switch (nday) { case 1: case 2: case: 3 case 4: case 5: a=b; break; case 6: swich (j) {case 7:a=d; case 8: a=h;} case 7: a=0; }**

**Описание грамматики в форме Бэкуса-Наура**

<Идентификатор> ::= <Буква> |

<Идентификатор><Буква> | <Идентификатор><Цифра>

<Буква> ::= a|b|…|z|A|B|…|Z

<Цифра> ::= 0|1|2|…|9

<Оператор> ::= <Присваивание> | <Оператор switch>

<Присваивание> ::= <Идентификатор> = <Выражение>

<Оператор switch> ::= switch (<Условие>) { <Конструкция case> }

<Выражение> ::= <Целое> | <Идентификатор>

<Целое> ::= <Знак><Целое без знака> | <Целое без знака>

<Целое без знака> ::= <Цифра><Целое без знака> | <Цифра>

<Знак> ::= +|-

<Условие> ::= <Идентификатор>

<Конструкция case> ::= case <Целое>: <Конструкция case> <Оператор>; break; | case <Целое>: <Оператор>; break; |

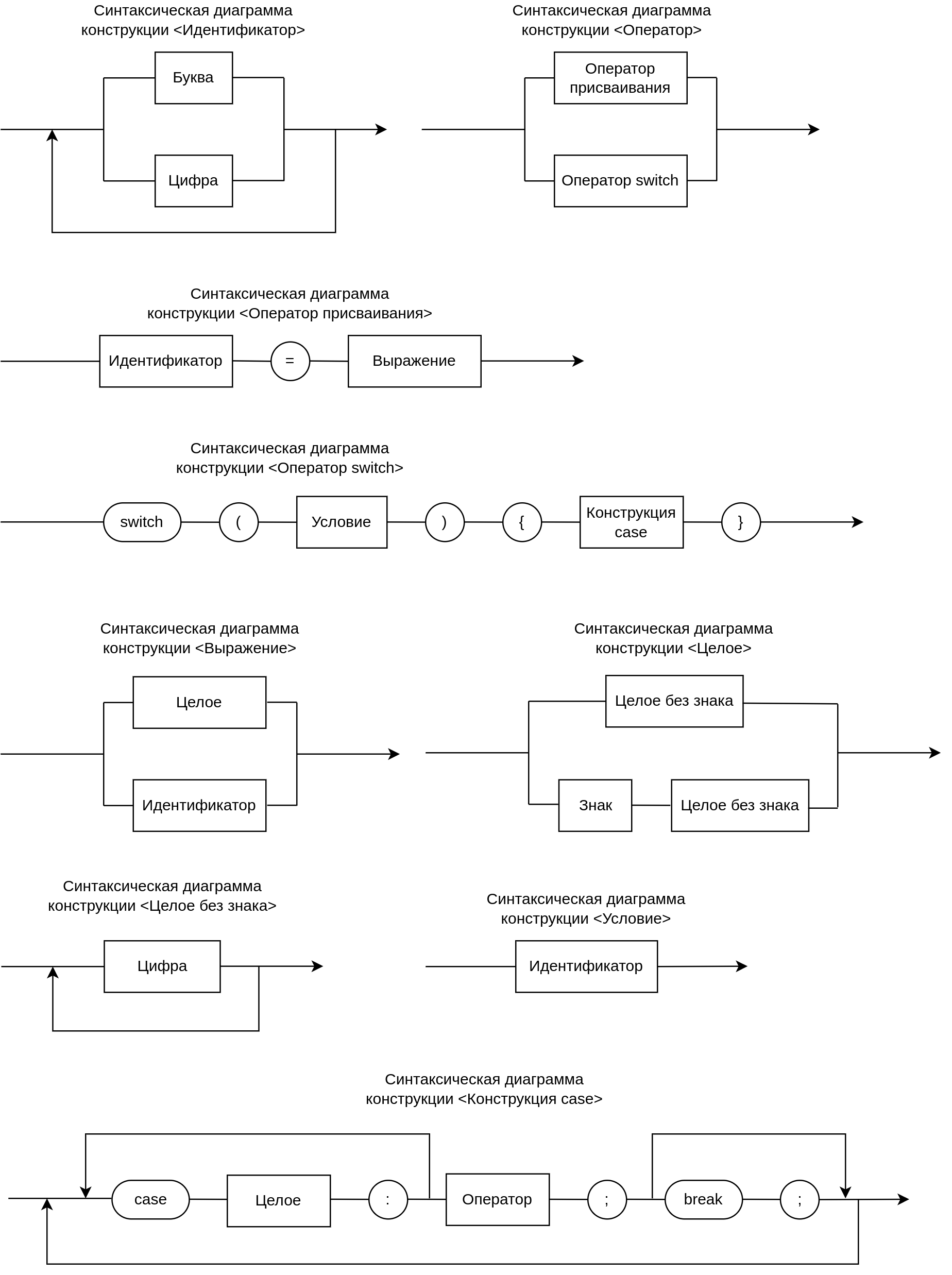
case <Целое>: <Конструкция case> <Оператор>; |

case <Целое>: <Оператор>;

Данная грамматика по классификации Хомского относится к грамматикам 2 типа, так как в самом сложном правиле слева всегда единственный нетерминал и допустима только вложенная рекурсия.

**Синтаксический анализ**

Синтаксические диаграммы используемых конструкций приведены на рисунке 1 и 2.

РРисунок 1 - Синтаксические диаграммы

**Программа DZ2.py**

import re

identifier\_pattern = r'^[a-zA-Z\_][a-zA-Z0-9\_-]\*$'

number\_pattern = r'[-+]?\d+$'

splitters = {' ', '=', ';', ':', '(', ')', '{', '}'}

error\_status = False

def get\_tokens(string):

string += ''

buf = ''

result = []

array = ['switch', 'case', 'break']

for i in range(len(string)):

check = False

if string[i] not in splitters:

buf += string[i]

else:

if i == len(string) - 1:

check = True

j = 0

while j < 3 and buf != array[j]:

j += 1

if j != 3:

print(f"Служебное слово: {buf}")

if buf[0] == 's':

result += ['sw']

elif buf[0] == 'c':

result += ['ca']

elif buf[0] == 'b':

result += ['br', ';\_']

else:

if re.match(identifier\_pattern, buf):

print(f"Идентификатор: {buf}")

result += ['V\_']

elif re.match(number\_pattern, buf):

print(f"Целое число: {buf}")

result += ['V\_']

else:

if buf != '':

print(f"Неверный формат идентификатора: {buf}")

if string[i] != ' ':

print(f"Служебное слово: {string[i]}")

result += [string[i] + '\_']

buf = ''

return result

def operator\_switch():

global tokens

if ''.join(tokens[0:5]) == 'sw(\_V\_)\_{\_':

tokens = tokens[5:]

r = cons\_case()

if r and tokens[0] == '}\_':

tokens = tokens[1:]

elif not r:

print\_error('Неправильная конструкция case')

else:

r = False

print\_error('Неправильная конструкция switch')

else:

r = False

print\_error('Неправильная конструкция switch')

return r

def cons\_case():

global tokens

if ''.join(tokens[0:3]) == 'caV\_:\_':

tokens = tokens[3:]

r = cons\_case()

else:

r = operator()

if r and ''.join(tokens[0:3]) == ';\_br;\_':

tokens = tokens[3:]

if tokens[0] == 'ca':

r = cons\_case()

elif r and tokens[0] == ';\_':

tokens = tokens[1:]

if tokens[0] == 'ca':

r = cons\_case()

elif not r:

print\_error("Неправильный оператор")

else:

r = False

return r

def operator():

global tokens

if tokens[0] == 'V\_':

r = assignment()

elif tokens[0] == 'sw':

r = operator\_switch()

else:

r = False

return r

def assignment():

global tokens

if ''.join(tokens[0:3]) == 'V\_=\_V\_':

tokens = tokens[3:]

return True

else:

print\_error("Неправильное присваивание")

return False

def print\_error(error):

global error\_status

if not error\_status:

print(f"{error}: {' '.join(tokens)}")

error\_status = True

tokens = get\_tokens(input())

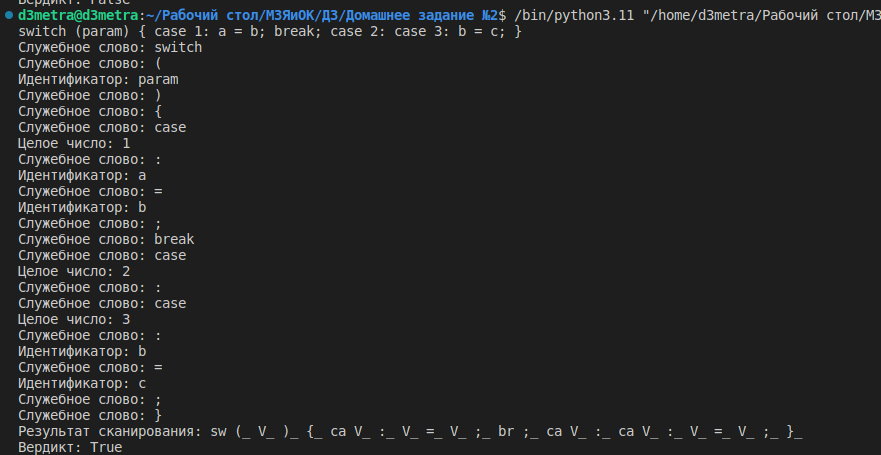
print(f"Результат сканирования: {' '.join(tokens)}")

answer = operator\_switch()

print(f"Вердикт: {answer}")

Запустим программу. Результат выполнения программы представлен на рисунке 2.

Проведём тестирование программы. Результаты представлены в таблице 1.

Рисунок 2 - Выполнение программы

*Таблица 1 — Результаты тестирования программы*

|  |  |  |
| --- | --- | --- |
| **Исходные данные** | **Ожидаемый результат** | **Полученный результат** |
| abc | false | false |
| switch (param) { case 1: a = b; break; case 2: case 3: b = c; } | true | true |
| switch (pop) { case 1: a = b; } | true | true |

***Контрольные вопросы***

1. Дайте определение формального языка и формальной грамматики.

Формальным языком называют множество цепочек, составленных из символов некоторого конечного алфавита. Формальной грамматикой называют систему, которая определяет язык с помощью правил продукции и определяется как четверка: G = (VT, VN, P, S), где VT – алфавит языка; VN – вспомогательный алфавит, символы которого обозначают допустимые понятия языка; P – множество порождающих правил; S – аксиома грамматики.

2. Как определяется тип грамматики по Хомскому?

В зависимости от свойств грамматики определяется ее тип. Если грамматики «без ограничений», то это тип 0 Если контекстно-зависимые грамматики, то это тип 1 Если контекстно-свободные грамматики, то это тип 2. Если регулярные грамматики, то это тип 3

3. Поясните физический смысл и обозначения формы Бэкуса–Наура.

Форма Бэкуса – Наура дает возможность более компактно написать грамматику, что значительно облегчает её понимание и чтение. Она связывает терминальные и нетерминальные символы и также применяет две операции: «можно заменить на» или «или».

4. Что такое лексический анализ? Какие методы выполнения лексического анализа вы знаете?

Лексический анализ – первый этап процесса компиляции текста, написанного на формальном языке программирования. Во время выполнения лексического анализа текст разделяют на «предложения» - операторы языка, а операторы – на «слова» (лексемы).

5. Что такое синтаксический анализ? Какие методы синтаксического анализа вы знаете? К каким грамматикам применяются перечисленные вами методы?

Синтаксический анализ – второй этап работы компилятора. Это процесс распознавания в строке токенов конструкций языка. В зависимости от типа грамматики определяют способ построения анализатора. Для грамматик третьего типа используют конечные автоматы. Для грамматик второго типа используют автоматы с магазинной памятью.

6. Что является результатом лексического анализа?

Результатом лексического анализа является таблица токенов.

7. Что является результатом синтаксического анализа?

Результатом синтаксического анализа являются распознанные конструкции языка.

8. В чем заключается метод рекурсивного спуска?

Метод рекурсивного спуска заключается в построении синтаксической диаграммы и написание основной программы, которая начинает вызов функций с функции, реализующей аксиому языка.

9. Что такое таблица предшествования и для чего она строится?

Таблица предшествования отображает отношения предшествования между терминальными символами строки. Таблица предшествования строится для определения основы, которая должна быть свернута на следующем этапе грамматического разбора.

10.Как с использованием таблицы предшествования осуществляют синтаксический анализ?

Использование таблиц предшествования помогает однозначно определить начало основы, конец основы и принадлежность одной основе. Также для определения, сформированы ли соответствующие комбинации. Далее выполняется операция свертки, когда соответствующая часть основы получена полностью. В конце получаем разбор предложения на отдельные операции.

***Вывод***: изучены теоретические основы и основные методы приемов разработки лексических и синтаксических анализаторов регулярных и контекстно-свободных формальных языков. Разработанная программа на заданных исходных данных работает корректно.