Министерство образования и науки Российской Федерации

Севастопольский государственный университет

Институт информационных технологий

Кафедра ИС

# ОТЧЁТ

по лабораторной работе №2

ИССЛЕДОВАНИЕ СКАНЕРА ПРИ АНАЛИЗЕ ПРОСТЫХ ЯЗЫКОВЫХ КОНСТРУКЦИЙ

Выполнил:

ст. гр. ИС/б-21-2-о

Шевелёв К. С.

Проверил:

Карлусов В. Ю.

Севастополь

2024

# ЦЕЛЬ РАБОТЫ

Изучить принципы построения и программирования лексического анализатора для простых языковых конструкций. Получить навыки практического построения лексического анализатора (сканера) на основе теории конечных автоматов. Освоить приёмы составления регулярных выражений для описания лексем. Закрепить навыки построения минимального КА, осуществляющего сканирование текста программ.

# Задание

* 1. Согласно варианту задания составить регулярное выражение, описывающее КА, который осуществляет лексический анализ. Составить эскизные выражения для описания дизъюнктивных компонент. Построить управляющую таблицу МДКА, редактировать программу из предыдущей работы под нужды лексического анализа и текст диагностических сообщений, генерируемых в его ходе. Разработать схему кодирования лексем;
  2. Согласовать текст диагностических сообщений с преподавателем, сконструировать текст программы сканера на основании программы МДКА. Выявить и устранить синтаксические ошибки;
  3. Разработать тестовые примеры правильных и неправильных лексем языка, уточнить диагностические сообщения. Подготовить мероприятия по хронометрированию работы сканера. Продумать структуру выходной информации программы на предмет использования другими программными модулями компилятора;
  4. Тестировать и отладить программу. Экспериментально определить время работы сканера для качественно различных структур строк и характера ошибок;

# Вариант Задания

Вариант задания – 48.

**Тип идентификатора** − в идентификаторе не встречается более трех букв, идущих подряд.

**Тип константы** – X. Шестнадцатеричное число. Формат 0x012…9a…f.

Таблица 1 – Фрагмент программы для анализа

|  |  |  |  |
| --- | --- | --- | --- |
| **Служебные слова** | **Разделители** | | **Фрагмент программы для анализа** |
| однолитерные | двулитерные |
| DO  WHILE  END | =  –  +  ,  (  ) | <=  >= | WHILE (K<=0o171) DO  K=K+1,  A=A–1,  END |

# ХОД РАБОТЫ

## Определение лексем и назначение им кодов

Всем лексемам были назначены коды относительно их классов и уникальности. Класс однолитерных лексем получил префикс 6, класс двулитерных − префикс 7.

* DO – 100
* WHILE – 200
* END – 300
* идентификатор, <iden> – 400
* константа, <data> – 500
* = – 601
* - – 602
* + – 603
* , – 604
* ( – 605
* ) – 606
* <= – 701
* >= – 702

## Построение регулярного выражения по варианту исходных данных

Общая структура регулярного выражения для описания КА, дополненного функциями сканера, примерно такова:

Обозначения указаны в методических указаниях.

**Служебные слова** являются самоопределяющимися цепочками, составленными из литер {D, O, W, H, I, L, E, N} – DO, WHILE, END.

Эскизно **идентификатор** (переменная) выглядит так:

*(Б* *˅ ББ* *˅ БББ) {Ц ˅ Ц (Б* *˅ ББ* *˅ БББ)},*

где Б – любая латинская буква, Ц – любая десятичная цифра.

Шестнадцатеричная **константа** описывается следующим образом:

*0x (ц ˅ б) {ц ˅ б},*

где ц – любая десятичная цифра, б – любая латинская буква в диапазоне a-f.

С учётом того, что конечный автомат должен отличать отдельные буквы и цифры на фоне других, имеем:

1. **Множество букв** , где δ − буква латинского алфавита, не совпадающая с другими литерами;
2. **Множество цифр** , где ц – любая цифра в диапазоне 1-9;
3. **Множество «не буква, не цифра»** , где – литера, не принадлежащая алфавиту конечного автомата;
4. **Множество «не 16-ричная цифра»** .

Окончательно получаем дизъюнктивные члены выражения, описывающие конечный автомат:

1. DO L1 – первое служебное слово;
2. WHILE L1 – второе служебное слово;
3. END L1 – третье служебное слово;
4. ((A ˅ B ˅ C ˅ D ˅ E ˅ F ˅ O ˅ W ˅ H ˅ I ˅ L ˅ N ˅ X ˅ δ) ˅ (A ˅ B ˅ C ˅ D ˅ E ˅ F ˅ O ˅ W ˅ H ˅ I ˅ L ˅ N ˅ X ˅ δ) (A ˅ B ˅ C ˅ D ˅ E ˅ F ˅ O ˅ W ˅ H ˅ I ˅ L ˅ N ˅ X ˅ δ) ˅ (A ˅ B ˅ C ˅ D ˅ E ˅ F ˅ O ˅ W ˅ H ˅ I ˅ L ˅ N ˅ X ˅ δ) (A ˅ B ˅ C ˅ D ˅ E ˅ F ˅ O ˅ W ˅ H ˅ I ˅ L ˅ N ˅ X ˅ δ)(A ˅ B ˅ C ˅ D ˅ E ˅ F ˅ O ˅ W ˅ H ˅ I ˅ L ˅ N ˅ X ˅ δ)) {0 ˅ ц ˅ (0 ˅ ц)((A ˅ B ˅ C ˅ D ˅ E ˅ F ˅ O ˅ W ˅ H ˅ I ˅ L ˅ N ˅ X ˅ δ) ˅ (A ˅ B ˅ C ˅ D ˅ E ˅ F ˅ O ˅ W ˅ H ˅ I ˅ L ˅ N ˅ X ˅ δ)(A ˅ B ˅ C ˅ D ˅ E ˅ F ˅ O ˅ W ˅ H ˅ I ˅ L ˅ N ˅ X ˅ δ) ˅ (A ˅ B ˅ C ˅ D ˅ E ˅ F ˅ O ˅ W ˅ H ˅ I ˅ L ˅ N ˅ X ˅ δ)(A ˅ B ˅ C ˅ D ˅ E ˅ F ˅ O ˅ W ˅ H ˅ I ˅ L ˅ N ˅ X ˅ δ)(A ˅ B ˅ C ˅ D ˅ E ˅ F ˅ O ˅ W ˅ H ˅ I ˅ L ˅ N ˅ X ˅ δ))} L1 – переменная;
5. 0 X {0 ˅ ц ˅ A ˅ B ˅ C ˅ D ˅ E ˅ F} (0 ˅ ц ˅ A ˅ B ˅ C ˅ D ˅ E ˅ F) L2 – константа;
6. – разделители.

Дизъюнктивные члены были подставлены в общее регулярное выражение.

## Построение функции переходов МДКА

Была построена разметка регулярного выражения (рисунок 1). Предостовные места для минимизации были отмечены жёлтым. Затем разметка была минимизирована (рисунок 2).



Рисунок 1 – Разметка регулярного выражения



Рисунок 2 – Минимизация разметки

По разметке была построена таблица переходов конечного автомата (рисунок 3). Автомат недетерминированный, неоднозначные переходы были выделены жёлтым.

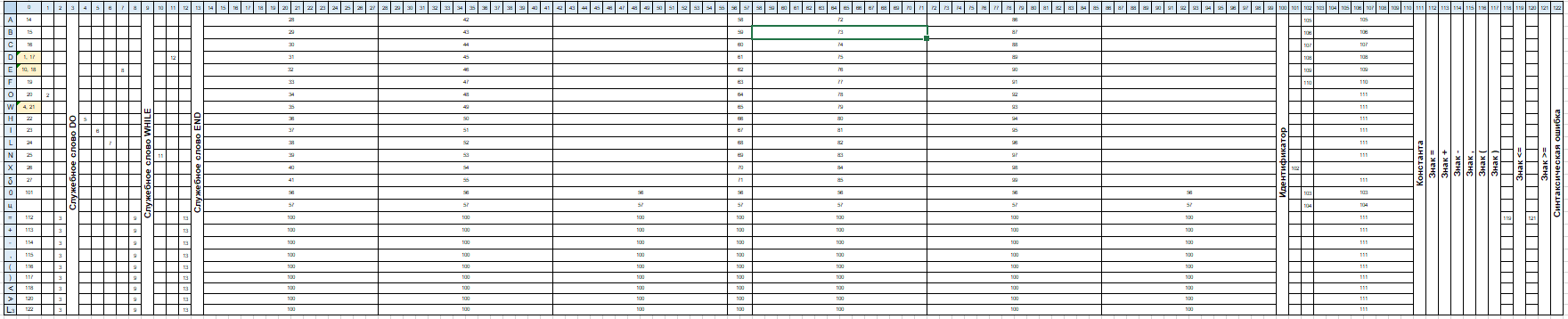


Рисунок 3 – Таблица переходов НДКА

Для преобразования НДКА в ДКА был построен граф фрагмента КА, относящегося к служебным словам (рисунок 4).

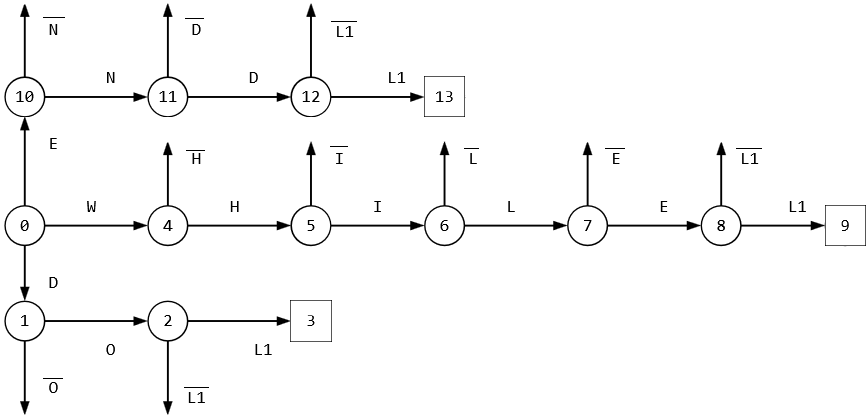


Рисунок 4 – Фрагмент КА, относящийся к разбору служебных слов

С помощью графов таблица переходов была дополнена таким образом, чтобы исключить недетерминированность автомата (рисунок 5).

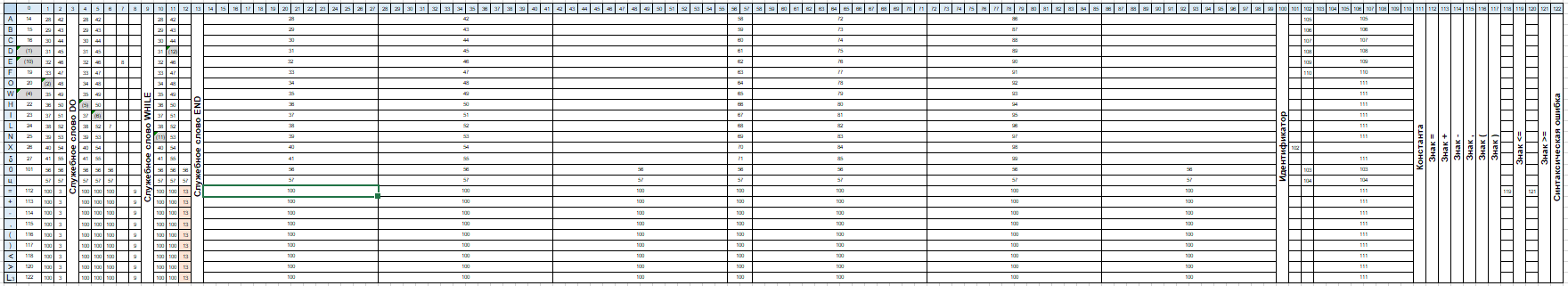


Рисунок 5 – Таблица переходов ДКА

Таблица переходов была минимизирована. Для этого сходные состояния были объединены между собой в одно, заключительным состояниям был установлен уникальный сигнал относительно кода лексемы (рисунок 6).

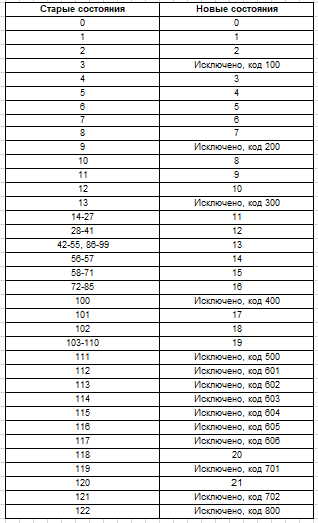


Рисунок 6 – Соответствие старых и новых состояний МДКА

Для случаев, когда образ функции переходов КА пуст, были введены коды ошибок, которые в ходе работы сканера помогут генерировать осмысленные диагностические сообщения:

1. Некорректный фрагмент цепочки. Код 801;
2. Ошибка в служебном слове. Код 802;
3. Ошибка в написании имени переменной. Код 803;
4. Ошибка в константе. Код 804;
5. Ошибка записи двулитерных операторов. Код 805.

Также для упрощения подготовки тестовых данных с позиции их наглядности в алфавит входных символов конечного автомата был введён пробел, пополняющий множество L1 (рисунок 7).

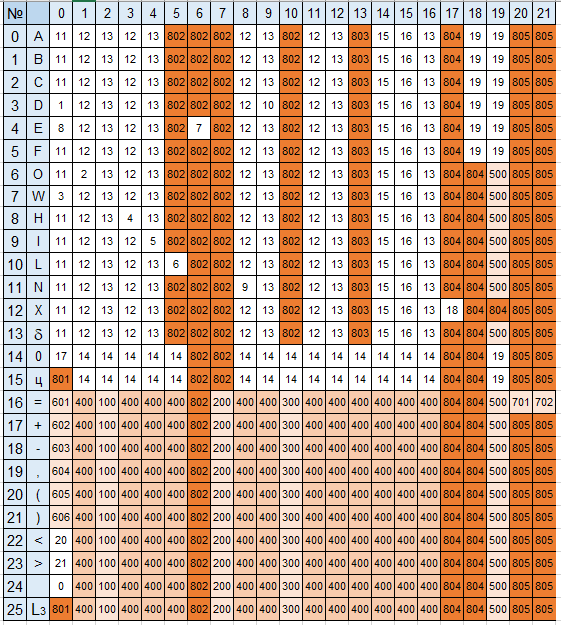


Рисунок 7 – Таблица переходов МДКА

## Разработка программы сканера

На основе полученных данных была написана программа лексического анализатора (сканера) на языке Python (листинг 1). Она считывает таблицу (матрицу) переходов и текст анализируемой программы из файлов.

Листинг 1 – Программа лексического анализатора

import csv

# ГЛОБАЛЬНЫЕ ПЕРЕМЕННЫЕ

table = None

separators = ',=+-<>()'

def read\_table(filename):

"""Чтение таблицы переходов из файла"""

global table

try:

with open(filename, "r") as file:

reader = csv.reader(file, delimiter=';')

n, m = map(int, next(reader))

table = []

for \_ in range(n):

row = list(map(int, next(reader)))

table.append(row)

except Exception as e:

print(f"Ошибка при обработке файла {filename}: {e}")

exit(0)

def scan\_file(filename):

"""Сканирование программы из файла"""

global table, separators

flag = False

try:

with open(filename, "r") as file:

N\_sost = 0 # текущее состояние КА (столбец таблицы состояний)

N\_str = 0 # код лексемы (строка таблицы состояний)

word = "" # текущее слово

c = file.read(1) # текущая литера на входе КА

while True:

if c == '\n': # абзац не учитывается

c = file.read(1)

continue

if c.isalpha(): # Это буква?

match c.upper():

case 'A':

N\_str = 0

case 'B':

N\_str = 1

case 'C':

N\_str = 2

case 'D':

N\_str = 3

case 'E':

N\_str = 4

case 'F':

N\_str = 5

case 'O':

N\_str = 6

case 'W':

N\_str = 7

case 'H':

N\_str = 8

case 'I':

N\_str = 9

case 'L':

N\_str = 10

case 'N':

N\_str = 11

case 'X':

N\_str = 12

case letter if letter in 'GJKMPQRSTUVYZ':

N\_str = 13

case \_:

N\_str = 25

elif c.isdigit(): # Это цифра?

match int(c):

case 0:

N\_str = 14

case num if 1 <= num <= 9:

N\_str = 15

else: # Это ни буква, ни цифра?

match c.upper():

case '=':

N\_str = 16

case '+':

N\_str = 17

case '-':

N\_str = 18

case ',':

N\_str = 19

case '(':

N\_str = 20

case ')':

N\_str = 21

case '<':

N\_str = 22

case '>':

N\_str = 23

case ' ':

N\_str = 24

case \_:

N\_str = 25

if not word and not c: break # проверка конца файла

N\_sost = table[N\_str][N\_sost] # новое состояние по таблице переходов

if N\_sost: # запоминание лекс. единицы

word += c if c else '.'

#print(f"Литера - '{c}', слово - '{word}'")

# обработка состояний

if 100 <= N\_sost < 600: # ключевое слово

handle\_keyword\_state(N\_sost, word[:-1])

flag = True

elif 600 <= N\_sost < 800: # разделитель

handle\_separator\_state(N\_sost)

elif N\_sost >= 800: # ошибка

handle\_error\_state(N\_sost, word)

# if (N\_sost == 802 or N\_sost == 803) and c not in separators:

# continue

if N\_sost > 801 and not ((N\_sost == 802 or N\_sost == 803) and c not in separators):

flag = True

# обнуление состояния

if N\_sost >= 100:

N\_sost = 0

word = ''

if not c: break; # конец файла

if not flag:

c = file.read(1)

else:

flag = False

except Exception as e:

print(f"Ошибка при обработке файла {filename}: {e}")

exit(0)

def handle\_keyword\_state(state, keyword):

"""Обработка состояния ключевых слов"""

keywords = {

100: "Служебное слово",

200: "Служебное слово",

300: "Служебное слово",

400: "Идентификатор",

500: "Константа"

}

print(f"{keywords.get(state, 'Неизвестное ключевое слово')} '{keyword}'. Cостояние {state}")

def handle\_separator\_state(state):

"""Обработка состояний разделителей"""

separators = {

601: "Разделитель '='",

602: "Разделитель '+'",

603: "Разделитель '-'",

604: "Разделитель ','",

605: "Разделитель '('",

606: "Разделитель ')'",

701: "Разделитель '<='",

702: "Разделитель '>='",

}

print(f"{separators.get(state, 'Неизвестный разделитель')}. Cостояние - {state}")

def handle\_error\_state(state, word):

"""Обработка ошибок"""

if state > 803:

word = word[:-1]

errors = {

801: "Некорректный фрагмент цепочки",

802: "Ошибка в служебном слове",

803: "Ошибка в написании имени переменной",

804: "Ошибочная константа",

805: "Ошибка записи разделителя"

}

print(f"{errors.get(state, 'Неизвестная входная литера')} '{word}'. Cостояние - {state}")

# ОСНОВНОЙ КОД

try:

read\_table("transitions.csv")

scan\_file("data.txt")

except Exception as e:

print(f"\nОшибка: {e}")

Программа была запущена для фрагмента программы из варианта задания и абсолютно верно определила лексемы и лишние символы (рисунок 9).

Входная последовательность:

WHILE (K<=0o171) DO

K=K+1,

A=A-1,

END

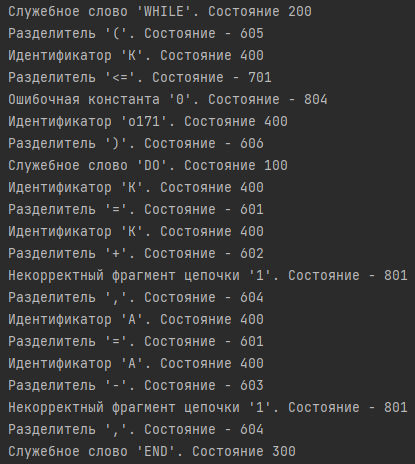


Рисунок 9 – Тестирование программы

Далее были написаны другие примеры.

Входная последовательность:

DEAD DO,

WHILY WHIL(0xABBA)

/=\END

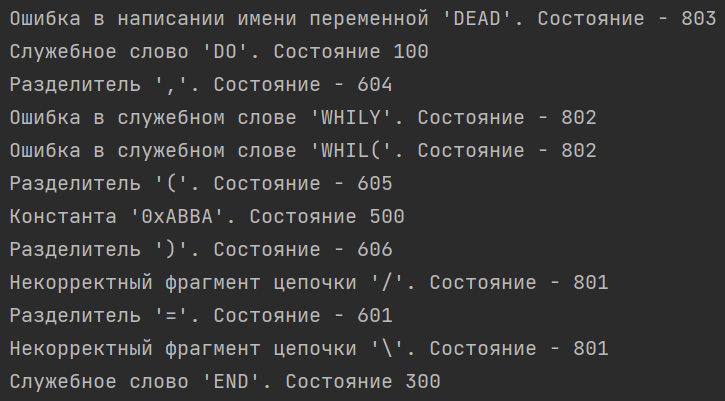


Рисунок 10 – Тестирование программы

Входная последовательность:

DO=

0x5 WHILE,

0xG>ENDD

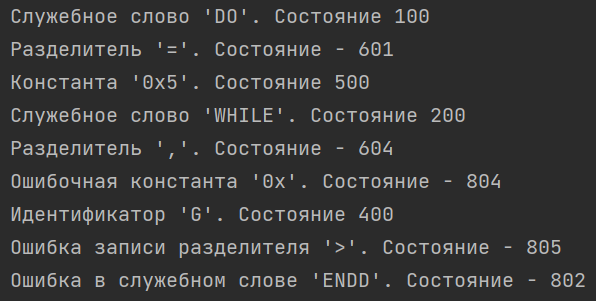


Рисунок 11 – Тестирование программы

# ВЫВОД

В ход работы были изучены принципы построения и программирования лексического анализатора для простых языковых конструкций. Получены навыки практического построения лексического анализатора (сканера) на основе теории конечных автоматов.