Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Факультет систем управления (ФСУ)

Кафедра автоматизированных систем управления (АСУ)

СИНТАКСИЧЕСКИЙ АНАЛИЗ С ИСПОЛЬЗОВАНИЕМ КОНЕЧНЫХ АВТОМАТОВ

Лабораторная работа №1 по дисциплине

«Теория языков программирования и методы трансляции»

Выполнил: студент гр. 430-2

_____А.А. Лузинсан

«____»_____2023 г.

Проверил: к.т.н., доц. каф. АСУ ТУСУР

_____ В.В. Романенко

«____»_____2023 г.

Томск 2023

Сокращения, обозначения, термины и определения

ДКА — детерминированный конечный автомат;

$\perp$ - маркер конца цепочки, который должен присутствовать в конце каждой анализируемой цепочки;

HALT — специальный символ, обозначающий успешное завершение разбора.

Оглавление

Введение

Цель: научиться применять на практике такие средства синтаксического анализа, как детерминированные конечные автоматы (ДКА).

Задание: написать программу, которая должна читать входные данные из текстового файла (например, имеющего имя «input.txt»), и выдавать результат работы в текстовый файл (например, имеющий имя «output.txt»). Для ввода и вывода данных допускается использование в программе визуального интерфейса вместо файлового ввода/вывода.

Вариант 1. На вход программы подается описание переменных на выбранном языке (Pascal, C++, C# и т.д.). Программа должна проанализировать его при помощи ДКА или ДМПА и выдать результат проверки. Это может быть:

1 Сообщение о том, что описание корректное.

2 Сообщение о синтаксической ошибке. Указывать тип ошибки не обязательно, требуется только указать строку и позицию в строке входного файла, где наблюдается ошибка. Достаточно находить только первую ошибку в описании.

3 Сообщение о дублировании имен переменных. В этом случае на выходе программы необходимо указать имя дублируемой переменной, а также строку и позицию в строке, где встретился дубликат.

1 ХОД РАБОТЫ

1.1 Краткая теория

В данном разделе приведена краткая теория, которая использовалась в ходе выполнения программной части лабораторной работы.

1.1.1 Синтаксис описания переменных

В качестве языка программирования был выбран C++, описание переменных которого подавалось на вход программы. Правила описания переменных в таком случае включают следующие аспекты:

- имя переменной может состоять только из латинских букв, цифр и символа подчеркивания;
- имя переменной не может начинаться с цифры;
- имя переменной не может повторяться, то есть нельзя объявить две переменные с одним именем;
- в качестве имени переменной не могут использоваться ключевые слова языка C++. Поддерживаемый список таких слов следующий: int, double, long, short, char, float.

Таким образом, описание состоит из указания типа данных со следующими за ним списком имён переменных. Помимо этого поддерживаются модификаторы размера типа: long, short. При этом модификатор может использоваться без указания базового типа. В этом случае в качестве базового типа подразумевается тип int. Модификатор long может использоваться с типами int и double, модификатор short — только с типом int. Без данных модификаторов используются типы float, char. Также, в программе поддерживается описание многомерных статических массивов, в качестве размеров которого могут указывается только натуральные целые числа.

1.1.2 Построение детерминированного конечного автомата

Первым делом было составлено сокращённое формальное описание конечного автомата, представленное в формуле 1.1.

$$\mathrm{M} = (\mathrm{Q},\ \Sigma, \delta, q_0), \tag{1.1}$$

где:

• Q — конечное множество состояний. Мощность этого множества будет определено в процессе составления функции переходов;

• Σ — конечное множество входных символов (алфавит). Для описания переменных алфавит состоит из: прописных и строчных букв латинского алфавита, символа подчёркивания (_), цифр, а также символов «,; []»;

• δ — функция переходов, отображение множества $\mathrm{Q} \times(\Sigma \cup \{\perp\})$ во множество Q;

• $\mathrm{q}_o \in Q$ — начальное состояние;

Положем, что на вход программе подаётся входная цепочка $\alpha = a_1 a_2 .. a_n \perp$. Тогда алгоритм работы ДКА представляет собой следующую последовательность действий:

1 Инициализируем $\mathrm{q} \leftarrow q_0, k \leftarrow 1$.

2 Находим значение отображения $\delta(q, a)$, где $\mathrm{a} = \mathrm{a}_k$.

3 Если $\delta(q, a)$ не определена, то выбрасывается исключение с позицией k. Если $\delta(q, a) = q$‘, то:

3.1 Переходим в новое состояние $\mathrm{q} \leftarrow q$‘.

3.2 Переходим к следующему символу $\mathrm{k} \leftarrow k + 1$.

4 Если $\delta(q, a) = HALT$, то разбор успешно завершен.

5 Если $\delta(q, a) = ERROR$, то имеем во входной цепочке синтаксическую ошибку в позиции k.

6 Иначе возвращаемся на шаг 2.

1.1.3 Определение функции переходов

Данный этап конкретизации конечного автомата проще всего провести посредством построения графа переходов. Для этого первым делом рассматривается начальное состояние.

Начальное состояние может являться также конечным, поэтому оно помечается окружностью с двойной границей.

Для каждого состояния графа q_i определяем, есть ли из данного состояния такие переходы, которые соответствуют допустимому символу a из входной цепочки, которые пока ещё отсутствуют в графе. Если есть, то проверяем, ведёт ли данный переход в уже умеющееся состояние q_j. Если да, то добавляем в граф только новый переход $\delta(q_i, a) = q_j$. Если нет, то добавляем в граф новое состояние q_k и переход в него $\delta(q_i, a) = q_k$. Если новое состояние может являться конечным, помечаем его двойной границей.

Если в процессе вышеописанных действий в графе появились новые состояния или переходы, то повторяем оную процедуру, иначе граф переходов построен. Построенный граф представлен в приложении А.1.

Помимо этого в графе были определены внедрённые действия:

- A_1 — добавление символа к временному буферу.
- A_2 — конец считывания идентификатора и его проверка на совпадение с контейнером идентификаторов. Очистка буфера.

Далее граф переходов был оформлен в табличной форме, представленной в приложении В.1. Строки данной таблицы представляют собой номера состояний, а столбцы — шаблоны переходов для реализации смешанного разбора. Пустые ячейки в данном случае обозначают значение ERROR. Ячейки, содержащие 2 числа, означают наличие внедрённого действия при переходе. Внедрённое действие при этом прописывается вторым значением.

1.1.4 Программная реализация

В качестве языка программирования был выбран python версии 3.11. Вспомогательной библиотекой, реализующей управляющее устройство, явилась билиотека pandas. Графический интерфейс, поддерживающий считывание входного сообщения как вручную, так и из файла, был обеспечен библиотекой dearpygui.

Программная реализация управляющего устройства заключается в считывании файла с функцией переходов в формате ods функцией read_excel() библиотеки pandas. Алфавит конечного автомата определяется в таблице с функцией переходов.

В ходе реализации функции переходов был выбран смешанный вариант разбора цепочки, который подразумевает разбор как по символам, так и по лексемам. Посимвольный автомат считывает входную цепочку посимвольно, т. е. Считывающая головка передвигается только на один символ за один такт работы автомата. Тогда как полексемный разбор подразумевает наличие в алфавите некоторых лексем (ключевых слов), и соответственно автомат за один такт считывает одну лексему. Таким образом смешанный разбор подразумевает под собой полексемное считывание ключевых слов и посимвольный разбор идентификаторов.

Чтобы реализовать данный подход была использована встроенная библиотека с регулярными выражениями re. Из заголовка таблицы переходов автоматически составляется регулярное выражение, содержащее количество нумерованных групп, совпадающее с количеством столбцов в таблице переходов. Содержимое этих групп также совпадает с содержимым заголовков столбцов. Далее к регулярному выражению применяется метод match(), суть которого отыскать совпадения по группам шаблона от начала строки, указываемой в аргументе. Задавая ключевому параметру pos индекс считывающей головки, метод match() выполняет поиск не с начала строки, а

с указываемой позиции во входной строке input_string.

Далее, в случае совпадения, метод group() возвращает сопоставленный символ или лексему. Для того чтобы узнать номер столбца для функции переходов используется проход по сопоставленным группам в генераторе списков и возврат нулевого элемента.

Наконец, свойство iloc возвращает объект, содержащий: строку со значением „HALT“, строку с данными для перехода или None. В случае „HALT“ ядро возвращает значение True, иначе, если это также строка, ядро определяет количество элементов в строке. Два значения парсятся в значение нового состояния и внедрённого действия. В этой же ветке применяется внедрённое действие. В противном случае строка конвертируется в целочисленное значение и определяет новое состояние. Далее считывающая головка сдвигается на количество позиций, равное длине сопоставленной группы.

Исключения могут возникнуть в трёх случаях:

• Неуникальный идентификатор. Возникает в функции внедрённых действий и возвращает текущий буффер, контейнер идентификаторов, а также номер строки и позиция в строке, где возникло исключение.

• Синтаксическая ошибка, заключающаяся в отсутствии символа или лексемы в поддерживаемом алфавите ДКА. В таком случае возвращается только номер строки и позиция с строке.

• Синтаксическая ошибка, заключающаяся в неверном переходе и возврате None значения, подразумевающее значение ERROR. В данном случае возвращается номер строки, позиция в строке, лексема или символ под считывающей головкой, временный буфер и контейнер идентификаторов.

Реализация лабораторной работы представлена в листинге В.1.

1.2 Тестирование программы

В ходе тестирования программы было проверено три ситуации, результаты которых представлены на рисунках 1.1-1.3, при которых вызываются исключения, описанные в пункте 1.1.4, а также ситуация с корректно описанными переменными и статическими массивами, изображенная на рисунке 1.4.

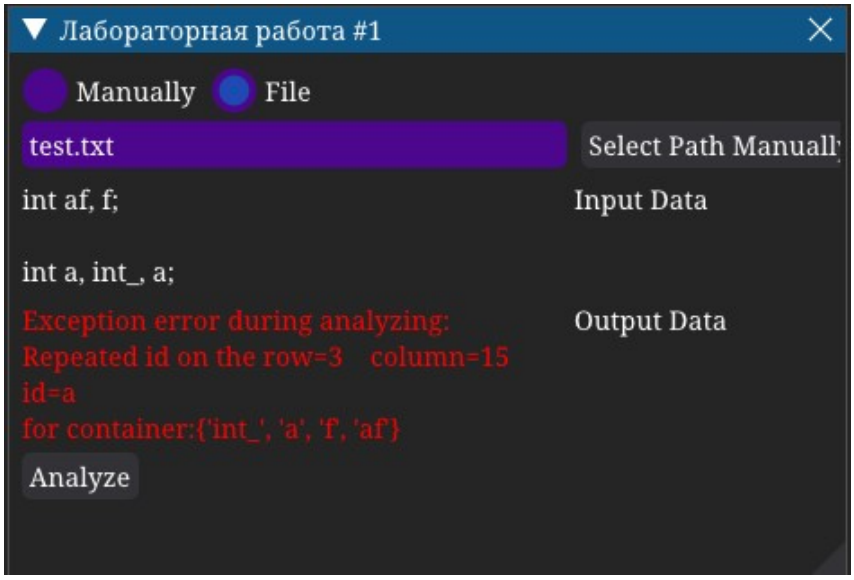

Рисунок 1.1 — Неуникальный идентификатор „a“

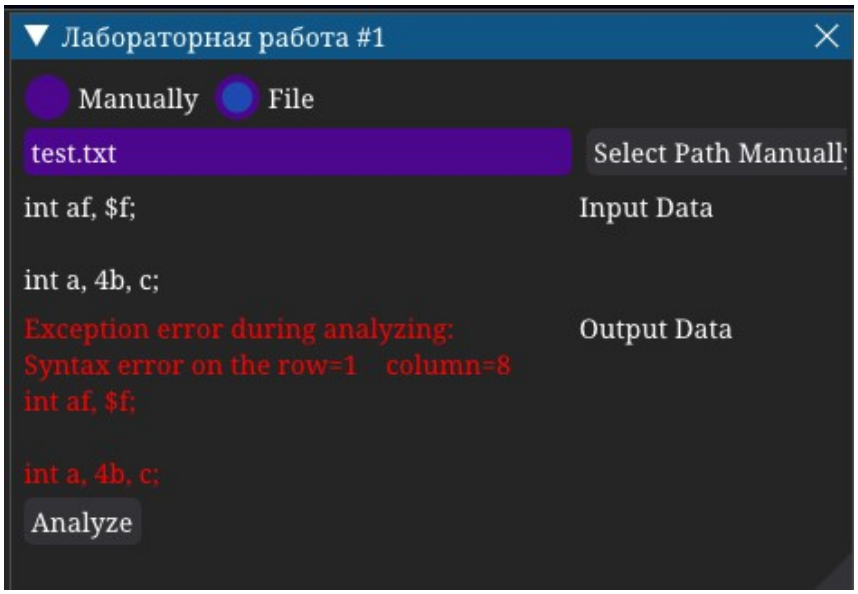

Рисунок 1.2 - Синтаксическая ошибка при отсутствии символа в алфавите

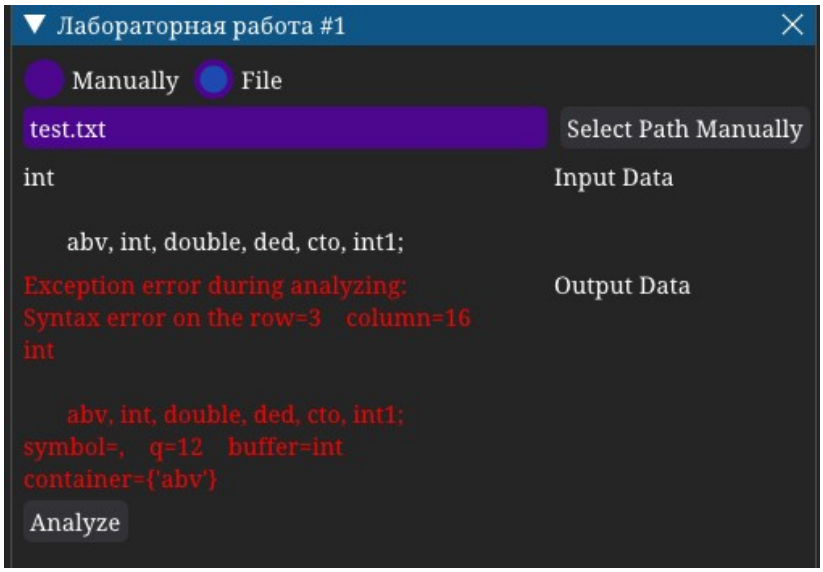

Рисунок 1.3 - Синтаксическая ошибка при неожиданном переходе

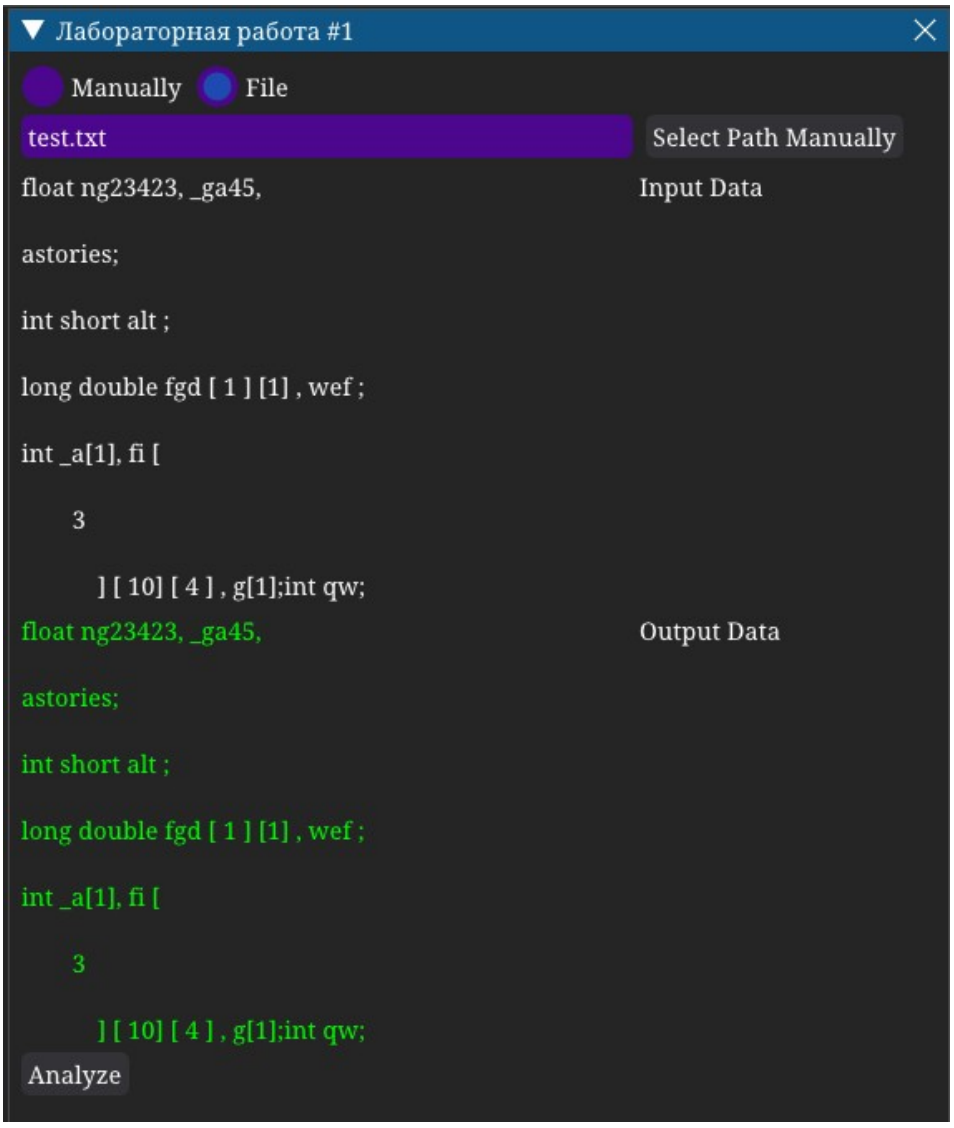

Рисунок 1.4 — Успешное завершение анализа описания переменных

Заключение

В результате выполнения лабораторной работы я научилась применять на практике такие средства синтаксического анализа, как детерминированные конечные автоматы (ДКА).

Приложение А
(обязательное)
Граф переходов

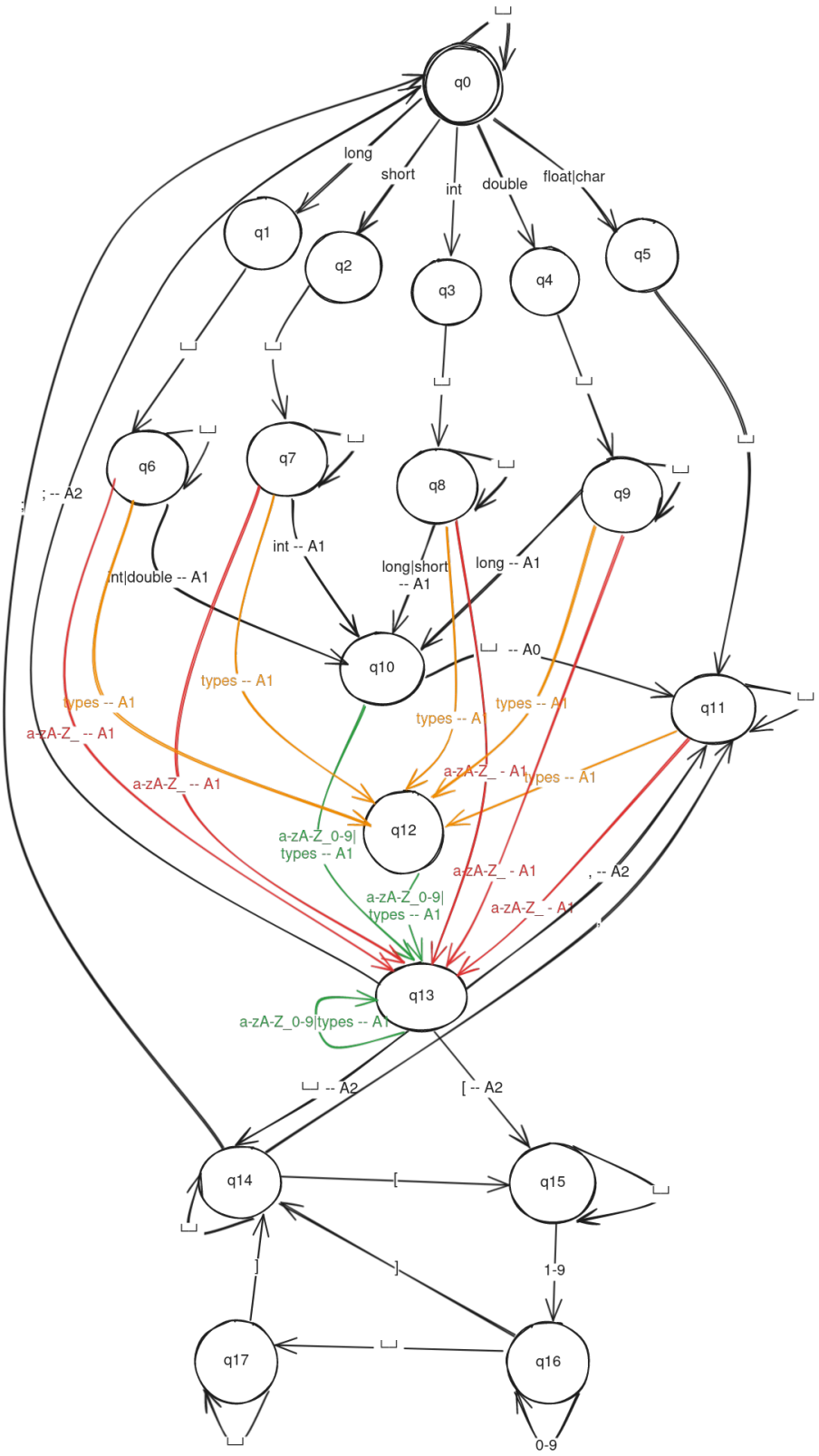

Рисунок А.1 — Граф переходов

Приложение Б
(обязательное)
Функция переходов в табличном виде

Таблица Б.1 — Функция переходов

	long	short	int	double	float|char	[a-zA-Z_]	0	[1-9]	\s	,	\[	\]	;	\0
0	1	2	3	4	5				0					HALT
1									6					
2									7					
3									8					
4									9					
5									11					
6	12 1	12 1	10 1	10 1	12 1	13 1			6					
7	12 1	12 1	10 1	12 1	12 1	13 1			7					
8	10 1	10 1	12 1	12 1	12 1	13 1			8					
9	10 1	12 1	12 1	12 1	12 1	13 1			9					
10	13 1	13 1	13 1	13 1	13 1	13 1	13 1	13 1	11 0					
11	12 1	12 1	12 1	12 1	12 1	13 1			11					
12	13 1	13 1	13 1	13 1	13 1	13 1	13 1	13 1						
13	13 1	13 1	13 1	13 1	13 1	13 1	13 1	13 1	14 2	11 2	15 2		0 2	
14									14	11	15		0	
15								16	15					
16							16	16	17			14		
17									17			14		

Приложение В
(обязательное)
Листинг программы

Листинг В.1 — Содержимое файла lr1.py

```
import pandas as pd
import dearpygui.dearpygui as dpg
import re
from __init__ import initialize
pd.set_option('display.max_columns', None)

class DFSM:
    __DELTA__: pd.DataFrame
    __container__: set
    __buffer__: str
    __header__: re

    def __init__(self, file_transition: str = 'transition.ods'):
        self.__DELTA__ = pd.read_excel(file_transition, index_col=0, engine="odf", dtype=str)
        self.__container__ = set()
        self.__buffer__ = str()
        self.__header__ = re.compile('(' + ')|('.join([str(column) for column in self.__DELTA__.columns]) + ')')

    def func(self, _func: int, row: int, column: int, symbol: str):
        match _func:
            case 1:
                self.__buffer__ += symbol
```

```
            case 2:
                if self.__buffer__ in self.__container__:
                    raise ValueError(f'Repeated id on the row={row}\tcolumn={column}'
                                     f'\nid={self.__buffer__}'
                                     f'\nfor container:{self.__container__}')
                self.__container__ |= {self.__buffer__}
                self.__buffer__ = ''

    def analyze(self, input_string: str):
        q, row, column = 0, 1, 0
        input_string += '\0'
        index = 0
        while True:
            symbol = input_string[index]
            if symbol == '\n':
                column = 0
                row += 1
            match = self.__header__.match(input_string, pos=index)
            if match:
                symbol = match.group()
                len_shift = len(symbol)
                match = [i for i, val in enumerate(match.groups()) if val is not None][0]
                    print('column: ', self.__DELTA__.columns[match], ' index of col: ',
match)
                data_zp = self.__DELTA__.iloc[int(q), int(match)]
                if data_zp == 'HALT':
                    return True
                elif type(data_zp) is str:
                    if len(data_zp.split()) == 2:
```

```
                    q, _func = map(int, data_zp.split())
                    self.func(_func, row, column, symbol)
                else:
                    q = int(data_zp)
                index += len_shift
                column += len_shift
            else:
                raise ValueError(f'Syntax error on the row={row}\tcolumn={column}'
                                 f'\n{input_string[:-1]}'
                                   f'\nsymbol={symbol}\tq={q}\tbuffer={self.__buffer__}\t\
ncontainer={self.__container__}')
        else:
            raise ValueError(f'Syntax error on the row={row}\tcolumn={column}'
                             f'\n{input_string[:-1]}')

def get_input_data():
    if dpg.get_value('input_method') == 'File':
        file_path = dpg.get_value('file')
        file = open(file_path, 'r', encoding="utf-8")
        input_data = '\n'.join(file.readlines())
        file.close()
        return input_data
    else:
        return dpg.get_value('Manually')

def main():
    dpg.show_item('Analyzing')
    input_data = get_input_data()
    dpg.set_value('input data', value=input_data)
```

```
    try:
        engine: DFSM = DFSM()
        engine.analyze(input_data)
        dpg.configure_item('test', default_value=input_data, color=(0, 255, 0, 255))
    except ValueError as err:
         dpg.configure_item('test', default_value=f"Exception error during analyzing:\
n{err}", color=(255, 0, 0, 255))

def initialize_lr1():
        with dpg.window(label="Лабораторная работа #1", tag='lr1', show=True,
width=500, height=700, pos=(100, 100),
                    on_close=lambda: dpg.delete_item('lr1')):
        initialize()
        dpg.add_button(label="Analyze", callback=main, show=False, tag='continue')
```