Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

СИНТАКСИЧЕЙСКИЙ АНАЛИЗ С ИСПОЛЬЗОВАНИЕМ КОНЕЧНЫХ АВТОМАТОВ

Отчет по лабораторной работе №1

По дисциплине

«Теория языков программирования и методы трансляции»

Студент гр. 431-3

\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.П. Бекиш

(подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

Руководитель:

к.т.н., доц. каф. АСУ ТУСУР

\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Романенко

(подпись)

Томск 2024

**Сокращения, обозначения, термины и определения**

ДКА — детерминированный конечный автомат;

14§latex§\bot§png§600§FALSE§ - маркер конца цепочки, который должен присутствовать в конце каждой анализируемой цепочки;

HALT — специальный символ, обозначающий успешное завершение разбора.

**Оглавление**

[Введение 4](#_Toc182915595)

[1 ТЕОРИЯ 5](#_Toc182915596)

[1.1 Синтаксис описания переменных 5](#_Toc182915597)

[1.2 Построение детерминированного конечного автомата 5](#_Toc182915598)

[1.3 Определение функции переходов 6](#_Toc182915599)

[2 РЕЗУЛЬТАТЫ РАБОТЫ 8](#_Toc182915600)

[2.1 Программная реализация 8](#_Toc182915601)

[2.2 Тестирование программы 10](#_Toc182915602)

[Заключение 12](#_Toc182915603)

[Список использованных источников 13](#_Toc182915604)

[Приложение А (обязательное) Граф переходов 14](#_Toc182915605)

[Приложение Б (обязательное) Функции переходов в табличном виде 15](#_Toc182915606)

[Приложение В (обязательное) Листинг программы 16](#_Toc182915607)

# 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 указать строку и позицию в строке входного файла, где находится ошибка.
3. Сообщение о дублирование имен переменных. В этом случае на выходе программы необходимо указать имя дублируемой переменной, строку и позицию в строке, где находится дубликат.

# ТЕОРИЯ

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

## 1.1 Синтаксис описания переменных

В качестве языка программирования был выбран С++, 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.

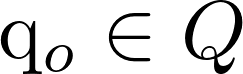
Таким образом, описание состоит из указания типа данных со следующими за ним списком имён переменных. Помимо этого, поддерживаются модификаторы размера типа: long, short. При этом модификатор может использоваться без указания базового типа. В этом случае в качестве базового типа подразумевается тип int. 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.

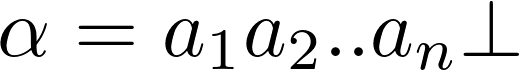
## 1.2 Построение детерминированного конечного автомата

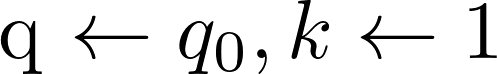
Первым делом было составлено сокращённое формальное описание конечного автомата, представленное в формуле 1.1.

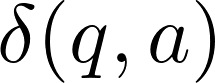
 (1.1)

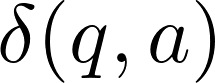
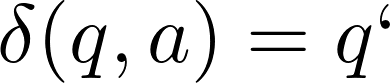
где:

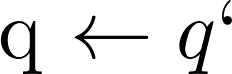
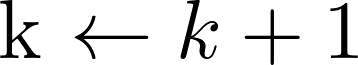
* Q — конечное множество состояний. Мощность этого множества будет определено в процессе составления функции переходов;
* 14§latex§\Sigma§png§600§FALSE§ — конечное множество входных символов (алфавит). Для описания переменных алфавит состоит из: прописных и строчных букв латинского алфавита, символа подчёркивания (\_), цифр, а также символов «,;[]»;
* 14§latex§\delta
  §png§600§FALSE§ — функция переходов, отображение множества  во множество 14§latex§Q§png§600§FALSE§;
*  — начальное состояние;

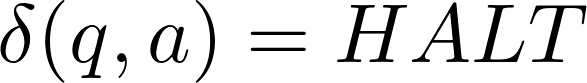
Пусть на вход программе подаётся входная цепочка . Тогда алгоритм работы ДКА представляет собой следующую последовательность действий:

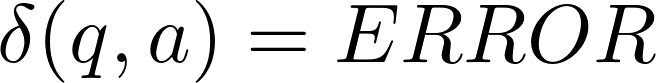
1. Инициализируем .

Находим значение отображения , где 14§latex§a = a_k§png§600§FALSE§.

Если  не определена, то выбрасывается исключение с позицией 14§latex§k§png§600§FALSE§. Если , то:

* 1. Переходим в новое состояние .
  2. Переходим к следующему символу .

Если , то разбор успешно завершен.

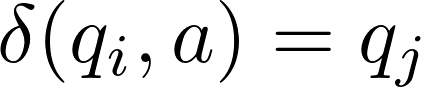
Если , то имеем во входной цепочке синтаксическую ошибку в позиции k.

1. Иначе возвращаемся на шаг 2.

## 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 14§latex§q_i§png§600§FALSE§ определяем, есть ли из данного состояния такие переходы, которые соответствуют допустимому символу 14§latex§a§png§600§FALSE§ из входной цепочки, которые пока ещё отсутствуют в графе. Если есть, то проверяем, ведёт ли данный переход в уже существующие состояние 14§latex§q_j§png§600§FALSE§. Если да, то добавляем в граф только новый переход . Если нет, то добавляем в граф новое состояние 12§latex§q_k§png§600§FALSE§ и переход в него . 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 A.1.

Так же в графе были определенны дополнительные действия:

* 1 – добавление символа в буфер;
* 2 – окончание добавления символов в буфер, которые формируют идентификаторы, после следует проверка на совпадение с контейнером идентификаторов и буфер очищается.

Далее граф переходов был оформлен в табличной форме, представленной в приложении B.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.

# 2 РЕЗУЛЬТАТЫ РАБОТЫ

## 2.1 Программная реализация

В качестве языка программирования был выбран python версии 3.12.7. Вспомогательной библиотекой, реализующей управляющее устройство, явилась библиотека pandas. Графический интерфейс, поддерживающий считывание входного сообщения как вручную, так и из файла, был обеспечен библиотекой streamlit.

Программная реализация управляющего устройства заключается в считывании файла с функцией переходов 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 text.

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.

## 2.2 Тестирование программы

В ходе тестирования программы было проверено три ситуации, результаты которых представлены на рисунках 2.1 – 2.3, где вызываются исключения, а так же и корректные ситуации.

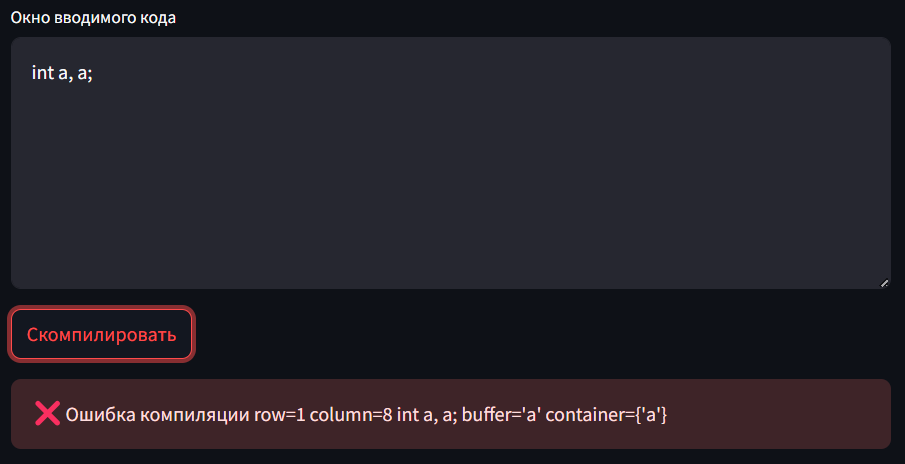


Рисунок 2.1 — Неуникальный идентификатор „a“

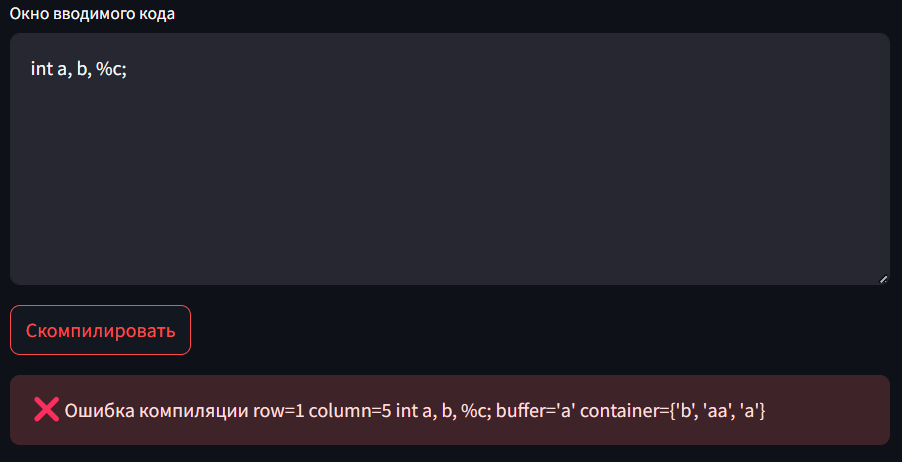


Рисунок 2.2 - Синтаксическая ошибка при отсутствии символа в алфавите

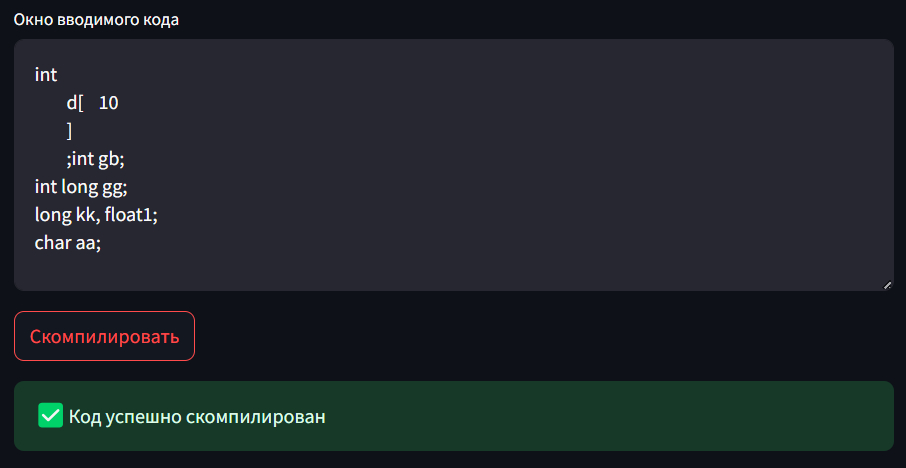


Рисунок 2.3 — Успешное завершение анализа описания переменных

# Заключение

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

# Список использованных источников

1. Романенко, В. В. Теория языков программирования и методы трансляции: Учебное пособие [Электронный ресурс] / В. В. Романенко, В. Т. Калайда. — Томск: ТУСУР, 2019. — 264 с. — Режим доступа: https://edu.tusur.ru/publications/9043
2. Романенко, В. В. Теория языков программирования и методы трансляции: Учебно-методическое пособие по выполнению лабораторных работ [Электронный ресурс] / В. В. Романенко, В. Т. Калайда. — Томск: ТУСУР, 2019. — 122 с. — Режим доступа: https://edu.tusur.ru/publications/9044
3. Образовательный стандарт вуза ОС ТУСУР 01-2021. Работы студенческие по направлениям подготовки и специальностям технического профиля. Общие требования и правила оформления от 25.11.2021 [Электронный ресурс]. – Режим доступа: <https://regulations.tusur.ru/documents/70>.

# Приложение А (обязательное) Граф переходов

Рисунок А.1 — Граф переходов

# Приложение Б (обязательное) Функции переходов в табличном виде

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | long | int | short | 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 | 10 1 | 12 1 | 10 1 | 12 1 | 12 1 | 13 1 |  |  | 7 |  |  |  |  |  |
| 8 | 12 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 |  |  |  |  |  |
| 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 main.py

import pandas as pd

import re

buffer = ''

container = set()

def func\_transition(file\_excel):

\_\_func\_transition = pd.read\_excel(

io=file\_excel,

index\_col=0,

dtype=str

)

return \_\_func\_transition

def header(func\_transition):

return re.compile(f'({')|('.join([str(column) for column in func\_transition.columns])})')

def analyze\_syntax(text, header, func\_transition):

q = 0 # состояние

row = 1

column = 0

index = 0

text += '\0'

while True:

symbol = text[index]

print(f'Текущий символ: {symbol}')

if symbol == '\n':

column = 0

row += 1

print(f'Текущая строка: {text[index:]}')

match = header.match(text, pos=index) # совпадение

print(f'{match=}')

if match:

print(f'{match.groups()=}')

symbol = match.group()

print(f'Просмотр символа для перехода: {symbol}')

shift = len(symbol)

match = [i for i, val in enumerate(match.groups()) if val is not None][0]

print(f'column: {func\_transition.columns[match]}\tcolumns index: {match}')

next\_q = func\_transition.iloc[int(q), int(match)]

print(f'Следующие состояние перехода: {next\_q}\tТекущие состояние: {q}')

if next\_q == 'HALT':

return True

elif type(next\_q) is str:

if len(next\_q.split()) == 2:

q, check = map(int, next\_q.split())

repeat = check\_var(check, symbol, row, column, text)

if repeat:

return repeat

else:

q = int(next\_q)

#q = int(next\_q)

index += shift

column += shift

else:

return f'Ошибка компиляции\nrow={row}\tcolumn={column}\n{text[:-1].split('\n')[row - 1]}'

else:

return f'Ошибка компиляции\nrow={row}\tcolumn={column}\n{text[:-1].split('\n')[row - 1]}'

def check\_var(check, symbol, row, column, text):

global buffer, container

if check == 1: # добавление символа к временному буферу

buffer += symbol

# конец считывания идентификатора и его проверка на совпадение

# с контейнером идентификаторов, буфер очищается

elif check == 2:

if buffer in container:

return f'Ошибка компиляции\nrow={row}\tcolumn={column}\n{text[:-1].split('\n')[row - 1]}\n{buffer=}\n{container=}'

container.add(buffer)

buffer = ''

print(f'{buffer=}\n{container=}')

#table\_func\_transition = func\_transition('func\_transition.xlsx')

#print(table\_func\_transition)

#\_\_header = header(table\_func\_transition)

#print(\_\_header)

#file = open('с++.txt').read()

#print(analyze\_syntax(file, \_\_header, table\_func\_transition))