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 РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ

Лабораторная работа №2 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

Оглавление

Введение

Цель: научиться применять на практике такие средства синтаксического анализа, регулярные выражения.

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 регулярного выражения и выдать результат проверки. При этом программа может быть написана на одном языке программирования, но проверять правильность описания переменных на другом языке. 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 КРАТКАЯ 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 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.

1.2 Построение регулярного выражения

В качестве способа определения языка был выбран метод непосредственной записи регулярного выражения, вместо его получения из решения системы уравнений с регулярными коэффициентами. Выбор был сделан по той причине, что количество состояний, полученное в ходе составления функции переходов ДКА составило 17 состояний, что затрудняет решение системы уравнений вручную.

Первым делом была определена начальная конструкция, которая включает в себя именованные группы с правильными цепочками:

- (?<type>int(?:\s+long|\s+short)?|double(?:\s+long)?|long(?:\s+int|\ s+double)?|short(?:\s+int)?|float|char) — именованная группа type, которая захватывает все правильные комбинации типа данных и его модификатора;
- (?<id>[a-zA-Z_][a-zA-Z_0-9]*) — именованная группа id, захватывающая все правильные цепочки имени идентификатора;
- (?<op>[1-9]\d*) - именованная группа op, которая захватывает правильно указанный размер массива.

Далее были добавлены группы, которые должны захватывать неправильные цепочки символов:

- (?<error_type>\S*?) - именованная группа error_type, которая захватывает любые символы, кроме пробельных, в случае, если в группу type не попал ни один захват;
- (?<error_id_like_type>(?&type)) — именованная группа error_id_like_type, захватывающая все идентификаторы, имя которых совпадает с зарезервированными ключевыми словами из группы type;
- (?<error_id>[^;]+?) - именованная группа error_id, которая захватывает остальные краевые случаи, которые не попали в группы error_id_like_type и id. Например, если в указанном идентификаторе содержатся символы, не входящие в регулярное множество \w: «!@#$

%^&*()»

• (?<error_op>[^,;]+?) - именованная группа error_op, захватывающая все неправильно указанные размерности массива, за исключением символов «,;»;

• (?<error_punc>\,\s*\;) - именованная группа error_punc, захватывающая подряд идущие символы пунктуации, для исключения краевых случаев.

В ходе составления регулярного выражения использовались следующие конструкции и регулярные множества:

• (?:…) - незахватываемая группа, которая применяет квантификаторы к части регулярного выражения, но не назначает id;

• (?<name>…) - захватываемая группа, которая использует вместо сгенерированного id пользовательское имя;

• (?&name) — рекурсивный паттерн, который был реализован в библиотеке regex (python);

• (?=...) - позитивный просмотр вперед, который является незахватываемой группой, но от успешности совпадения которого зависит то, захватится ли предыдущая цепочка, или нет;

• ?, ?? - захват нулевой или единичной длины. Добавочный символ «?» переключает шаблон в ленивый режим, в котором будет захватываться минимально допустимое количество символов;

• +, +? - захват длины один и более. Значение добавочного символа «?» аналогично;

• *, *? - захват длины ноль и более. Значение добавочного символа «?» аналогично;

• [^…] - захват всех символов, кроме указанных после символа «^».

Таким образом, построенное регулярное выражение представлено в приложении А.1.

2 РЕЗУЛЬТАТЫ РАБОТЫ

2.1 Программная реализация

В качестве языка программирования был выбран python версии 3.11. Вспомогательной библиотекой, дополняющей стандартную библиотеку с регулярными выражениями re, явилась билиотека regex. Графический интерфейс, поддерживающий считывание входного сообщения как вручную, так и из файла, был обеспечен библиотекой dearpygui.

После считывания данных для анализа, они отправляются в функцию analyze, в которой происходит итерация по соответствиям шаблону, возвращённым методом finditer библиотеки regex. Уже внутри данного цикла запускается цикл, итерирующий по ошибочным группам (с приставкой error_) и группе id. Тело этого вложенного цикла включает вызов метода spans к объекту match, которому передаётся название группы, и которое возвращает список кортежей. В данном случае каждый кортеж представляет собой начальную и конечную позицию захвата (capture) в соответствии (match). Если возвращённый список не пуст, вызывается метод catch_error(), которая, как следует из названия, ловит ошибки и записывает результат в переменную класса. В самом методе происходит итерация по списку захватов некоторой группы. Метод, помимо непосредственной записи в строку ошибок, вычисляет также номер строки, в которой произошла та или иная ошибка и позицию начала группы с ошибкой, относительно этой строки.

Помимо этого в программе происходит проверка уникальности идентификаторов. Данное ограничение проверяется перед вызовом метода catch_errors в методе get_dupl_ids() за счёт удаления из списка захватов группы id всех уникальных идентификаторов. При этом учитываются идентификаторы, считанные в предыдущих соответствиях (match). Однако слабым местом программы является тот случай, когда дублирование идентификаторов происходит в одном соответствии. В этой ситуации будет

показан последний встреченный дубликат.

Исключения, собираемые в процессе работы программы, соответствуют группам с приставкой error_, которые были описаны ранее. В добавок следует лишь упомянуть в группе id, вывод которой означает дублирование переменных.

Реализация лабораторной работы представлена в листинге Б.1.

2.2 Тестирование программы

В ходе тестирования программы был проанализирован тестовый файл, содержащий различного рода ошибки, анализ которых привёл к результату, представленному на рисунке 2.1. Корректное описание переменных также было протестировано, как показано на рисунке 2.2.

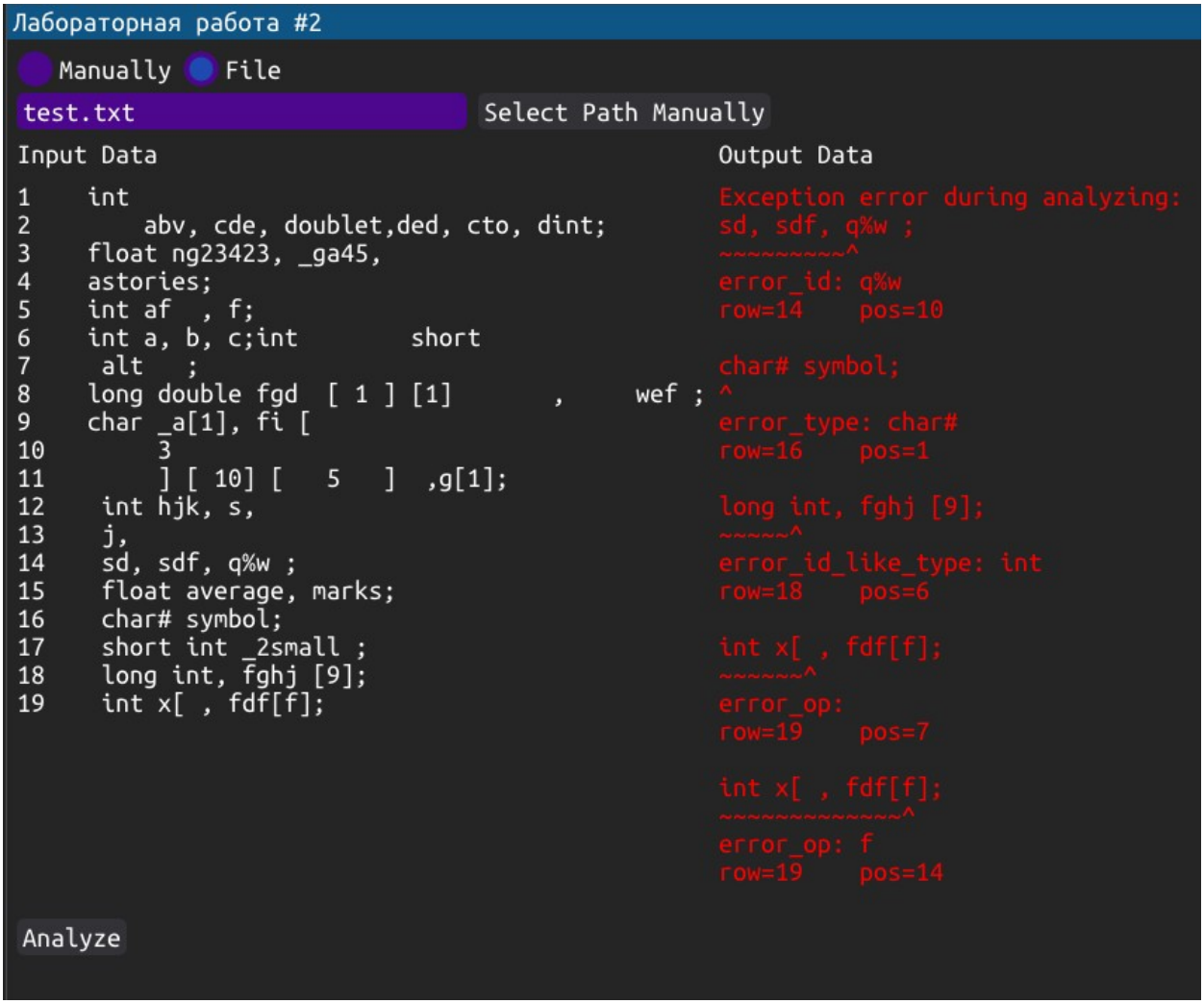

Рисунок 2.1 — Тестирование файла с ошибочным описанием переменных

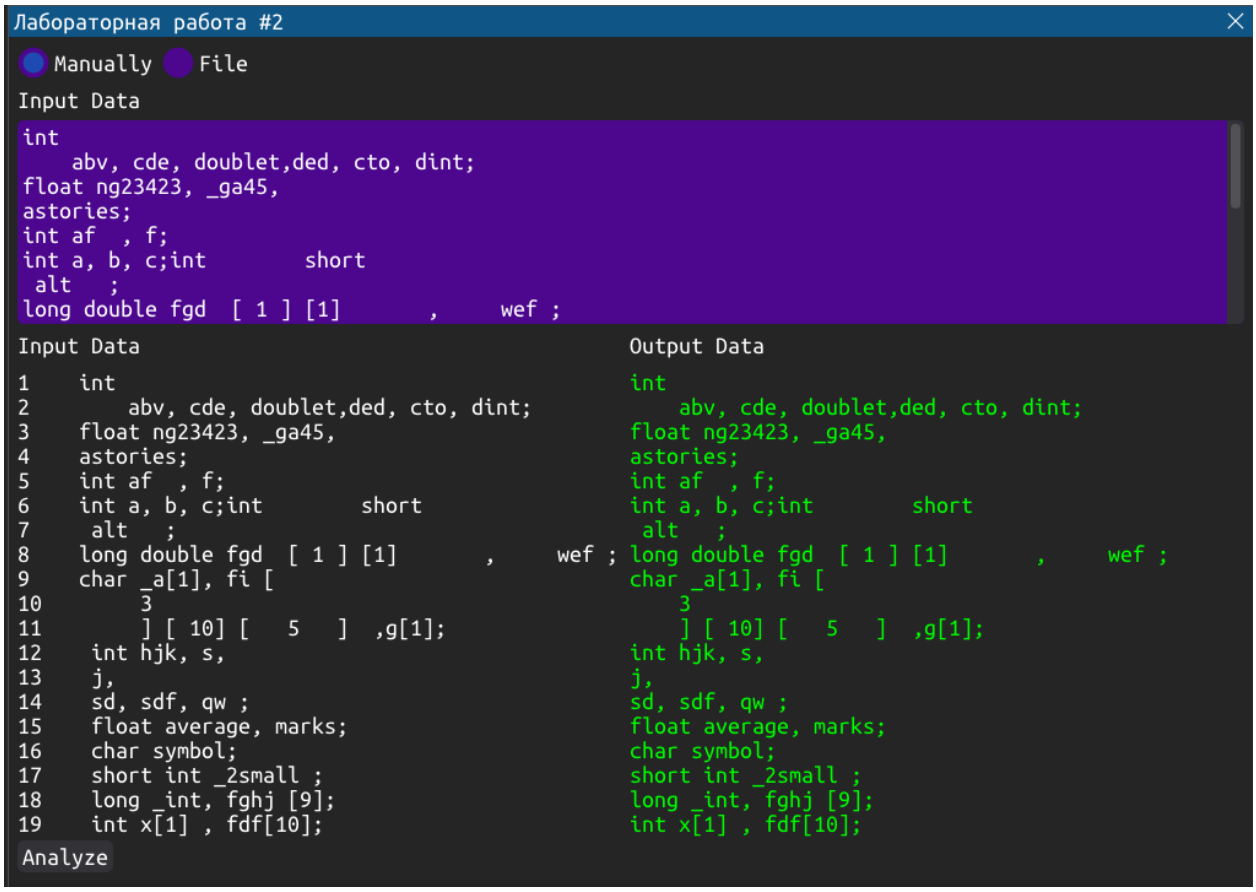

Рисунок 2.2 — Успешное завершение анализа описания переменных

Заключение

В результате выполнения лабораторной работы я научилась применять на практике такие средства синтаксического анализа, как регулярные выражения.

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)
Регулярное выражение

Листинг А.1 — Исходный текст регулярного выражения

```
\s*
(?:
  (?<type>int(?:\s+long|\s+short)?|double(?:\s+long)?|
    long(?:\s+int|\s+double)?|short(?:\s+int)?|float|char)
  |
  (?<error_type>\S*?)
)
\s+
(?:
  (?:
    (?:
      \b(?<error_id_like_type>(?&type))\b
      |
      (?<id>[a-zA-Z_][a-zA-Z_0-9]*)\b
      |
      (?<error_id>[^;]+?)
    )
    (?:\s*\[
              (?:
               \s*(?<op>[1-9]\d*)\s*\]
                |(?<error_op>[^,;]+?)\s*\]??
              )\s*)*
  )
  \s*
  (?:
    (?<error_punc>\,\s*\;)
    |(?:\,\s*)
    |(?=\;)
  )
)+\s*\;\s*
```

Приложение Б
(обязательное)
Листинг программы

Листинг Б.1 — Содержимое файла lr2.py

```
import dearpygui.dearpygui as dpg
import regex as re
from __init__ import initialize

class RegexAnalyze:
    pattern: re
    __container__: set
    __errors__: str

    def __init__(self, file_path_regex: str = "lr2/regex.txt"):
        with open(file_path_regex) as file:
            self.pattern = re.compile(''.join(file.readlines()).replace(" ", '').replace("\n",
''))
        self.__container__ = set()
        self.__errors__ = ''

    def get_dupl_ids(self, ids: list, match_spans):
        dupl_ids = list(dict.fromkeys([ii for n, ii
                                       in enumerate(ids)
                                       if (ids + list(self.__container__)).count(ii) > 1]))
        match_spans = list({ii: match_spans[n]
                            for n, ii
                            in enumerate(ids)
                            if (ids + list(self.__container__)).count(ii) > 1}.values())
        self.__container__.update(set(dict.fromkeys(ids)))
        return dupl_ids, match_spans

    def catch_errors(self, error: str, captures: list,
                                       curr_match: str, spans: list, start_pos: int, curr_row: int):
        for index, capture in enumerate(captures):
            pos = spans[index][0] - start_pos
            row_error = curr_row
            start_row = 0
            if re.search('\n', curr_match, endpos=pos):
                spans = [_match for _match in re.finditer('\n', curr_match, endpos=pos)]
                row_error = curr_row + len(spans)
                start_row = spans[-1].end()
```

```
            pos -= start_row
        last_n = re.search('\n', curr_match, pos=start_row)
        str_row = curr_match[start_row:last_n.start()]
                    if last_n else curr_match[start_row:]
        self.__errors__ += f'{str_row}\n' \
                        + f'{"~" * pos}^\n' \
                        + f'{error}: {capture}\n' \
                        + f'row={row_error}\tpos={pos + 1}\n\n'

    def analyze(self, input_string: str):
        row = 1
        for match in re.finditer(self.pattern, input_string, partial=True):
            print('capturesdict: ', match.capturesdict())
            capture_dict = match.capturesdict()
            for error in ['error_type', 'error_id_like_type',
                          'error_id', 'error_op', 'error_punc', 'id']:
                match_spans = match.spans(error)
                if len(capture_dict[error]):
                    if error == 'id':
                        capture_dict[error], match_spans = \
                                        self.get_dupl_ids(capture_dict[error], match_spans)
                    self.catch_errors(error, capture_dict[error], match[0],
                                      match_spans, match.start(), row)
            row += len(re.findall('\n', match[0]))
        if self.__errors__ != '':
            raise SyntaxError(self.__errors__)

def main():
    dpg.show_item('Analyzing')
    input_data = get_input_data()
    data_with_numering = '\n'.join([f'{index}\t{row}'
                                    for index, row
                                    in enumerate(input_data.split('\n'), 1)])
    dpg.set_value('input data', value=data_with_numering)
    try:
        engine: RegexAnalyze = RegexAnalyze()
        engine.analyze(input_data)
        dpg.configure_item('test', default_value=input_data, color=(0, 255, 0, 255))
    except BaseException as err:
        dpg.configure_item('test',
                           default_value=f"Exception error during analyzing:\n{err}",
                           color=(255, 0, 0, 255))
```

```
def initialize_lr2():
    with dpg.window(label="Лабораторная работа #2",
                            tag='lr2', show=True, autosize=True, min_size=(1000, 800),
                            modal=True, pos=(480, 0),
                            on_close=lambda: dpg.delete_item('lr2')):
        initialize()
        dpg.add_button(label="Analyze", callback=main, show=False, tag='continue')


def get_input_data():
    if dpg.get_value('input_method') == 'File':
        file_path = dpg.get_value('file')
        file = open(file_path, 'r')
        input_data = ''.join(file.readlines())
        file.close()
        return input_data
    else:
        return dpg.get_value('Manually_text')
```