Национальный исследовательский университет ИТМО

Факультет программной инженерии и компьютерной техники

Направление программная инженерия
Образовательная программа системное и прикладное программное обеспечение

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 4

курса «Информатика»

Вариант № 129

Выполнил студент:
Бых Даниил Максимович
группа: P3109

Проверил:
Малышева Т. А.

Санкт-Петербург, 2025

Содержание

1. Задание

1. Определить номер варианта как остаток деления на 132 своего идентификационного номера в ISU: например, 125598 / 132 = 16. В случае, если в обоих указанных днях недели нет занятий, то увеличить номер варианта на восемь. В случае, если занятий нет и в новом наборе дней, то продолжать увеличивать на восемь.
2. Изучить форму Бэкуса-Наура.
3. Изучить основные принципы организации формальных грамматик.
4. Изучить особенности языков разметки/форматов JSON, RON, HCL, YAML, TOML, INI, XML
5. Понять устройство страницы с расписанием на примере расписания лектора: https://itmo.ru/ru/schedule/3/125598/raspisanie_zanyatiy.htm
6. Исходя из структуры расписания конкретного дня, сформировать файл с расписанием в формате, указанном в задании в качестве исходного. При этом необходимо, чтобы хотя бы в одной из выбранных дней было не менее двух занятий (можно использовать своё персональное). В случае, если в данный день недели нет таких занятий, то увеличить номер варианта ещё на восемь.
7. Обязательное задание (позволяет набрать до 50 процентов от максимального числа баллов БаРС за данную лабораторную). Написать программу на языке Python 3.x или любом другом, которая:
    - осуществляет парсинг и конвертацию исходного файла в бинарный объект (=десериализацию);
    - для решения задачи использует формальные грамматики; то есть ваш код должен уметь осуществлять парсинг и конвертацию любых данных, представленных в исходном формате, в данные, представленные в результирующем формате (как с готовыми библиотеками из дополнительного задания №2);
    - не использовует готовые библиотеки, в том числе регулярные выражения в Python и библиотеки для загрузки файлов.
8. Дополнительное задание №1 (позволяет набрать +15 процентов от максимального числа баллов БаРС за данную лабораторную). Написать программу на языке Python 3.x или любом другом, которая:
    - осуществляет парсинг и конвертацию бинароного объекта, полученного в обязательном задании, в новый формат (=сериализацию);
    - для решения задачи использует формальные грамматики;

- не использовует готовые библиотеки, в том числе регулярные выражения в Python и библиотеки для загрузки файлов.

9. Дополнительное задание №2 (позволяет набрать +10 процентов от максимального числа баллов БаРС за данную лабораторную).
    - Найти готовые библиотеки, осуществляющие аналогичный парсинг и конвертацию файлов (десериализацию и сериализацию).
    - Переписать исходный код и код из дополнительного задания №1, применив найденные библиотеки. Регулярные выражения также нельзя использовать.
    - Сравнить полученные результаты и объяснить их сходство/различие. Объяснение должно быть отражено в отчёте.
10. Дополнительное задание №3 (позволяет набрать +20 процентов от максимального числа баллов БаРС за данную лабораторную). Переписать код из дополнительного задания №1, чтобы серализация происводилась в XML файл.
11. Дополнительное задание №4 (позволяет набрать +5 процентов от максимального числа баллов БаРС за данную лабораторную).
    - Используя свою исходную программу из обязательного задания и программы из дополнительных заданий, сравнить стократное время выполнения парсинга + конвертации в цикле.
    - Проанализировать полученные результаты и объяснить их сходство/различие. Объяснение должно быть отражено в отчёте
12. Проверить, что все пункты задания выполнены и выполнены верно
13. Написать отчёт о проделанной работе.
14. Подготовиться к устным вопросам на защите.

2. Основные этапы вычислений

2. 1. Обязательное задание

Так как мой номер ISU - 501993, то итоговый вариант 501993 % 132 = 129 (129 вариант). Необходимо было выполнить следующие преобразования:

- Перевести данные из файла в формате INI в файл в формате RON.
- В качестве дней недели взять понедельник и субботу

Код, являющийся точкой входа в программу для решения как обязательного, так и дополнительных заданий, находиться в файле 1:

```
from lib.advanced_convertor import AdvancedConvertor
from lib.hand_written_convertor import HandWrittenConvertor
from lib.test import TestRunner
from os import path

if __name__ == "__main__":
  INPUT_FILE_SRC = path.abspath(path.join(path.dirname('__file__'), 'input
   ', 'schedule.ini'))

  print(f"Вариант: {501993 % 132}\n------------|")

  # Обязательное задание
  ini_file_content = HandWrittenConvertor.read_file(INPUT_FILE_SRC)
  deserialized_raw = HandWrittenConvertor.deserialize(file_content=
   ini_file_content)

  # Дополнительное задание 1
  HandWrittenConvertor.serialize(object=deserialized_raw, format='ron')

  # Дополнительное задание 2
  ini_parser = AdvancedConvertor.read_file(INPUT_FILE_SRC)
  deserialized_advanced = AdvancedConvertor.deserialize(parse_object=
   ini_parser)
  AdvancedConvertor.serialize(deserialized_advanced, format='ron')

  # Дополнительное задание 3
  HandWrittenConvertor.serialize(object=deserialized_raw, format='xml')

  # Дополнительное задание 4
  TestRunner.run_handwritten_test(INPUT_FILE_SRC)
  TestRunner.run_advanced_test(INPUT_FILE_SRC)
```

Листинг 1: main.py

Разберем детально решение обязательного задания.

- для парсинга и конвертацию исходного файла в бинарный объект был создан класс HandWrittenConvertor в файле 2.
- в нем был создан метод read_file, осуществляющий чтение исходного файла и удаление лишних символов табуляции.

- также в нем был создан метод deserialize, осуществляющий десериализацию исходного файла в объект класса Wrapper 3.

- класс Wrapper определяет бинарный объект, содержащий секциии исходного файла (класс Section), которые в свою очередь состоят из полей разного типа (NestedField - для вложенностей, ValueField - поле ключ+значения, и CommentField - комментарий)

```
"""Конвертор
 для обязательного задания, дополнительного задания №1 и №3.Без
 использования сторонних библиотек осуществляет:
- десериалиацию данных в формате INI,
- сериализацию в формат RON,
- сериализацию в формат XML
"""

from .convertor import Convertor
from .formal import Wrapper, Section, ValueField, CommentField,
    NestedField
from typing import Sequence
import os


class HandWrittenConvertor(Convertor):
  @staticmethod
  def read_file(src: os.path):
    with open(src, 'r', encoding='UTF-8') as f:
      return list(filter(lambda row: row != "", map(lambda row: row.strip
    (), f.readlines())))

  @staticmethod
  def deserialize(file_content: Sequence[str]):

    def is_section(row):
      return row.startswith('[') and row.endswith(']')

    def is_nested(row):
      prefix = row.split('.')[0]
      return not is_section(row) and row.find(prefix)+len(prefix) < row.
    find('=')

    def find_subfields(prefix: str, fields: Sequence, nested_prefix: str="
    ") -> NestedField:
      if nested_prefix == "":
        nested_prefix = prefix
      nested = NestedField(key=nested_prefix)
      fields = list(filter(lambda f: f.split('=')[0].startswith(prefix),
    fields))
      for field in fields:
        full_field_key, value = field.split('=')[:2]
        actual_field_key = full_field_key[len(prefix)+1:]

        if not '.' in actual_field_key:
          nested.add_field(ValueField(key=full_field_key.split('.')[-1],
    value=value))
        else:
          nested.add_field(find_subfields(full_field_key, fields,
    actual_field_key.split('.')[0]))
```

```
      return nested


    wrapper = Wrapper()

    sections = '\n'.join(file_content).split('[')[1:]
    for section_string in sections:
      nested_fields = list()
      section_fields = list(filter(lambda r: r != '', section_string.split
    ('\n')))
      section = Section(name=section_fields[0][:-1])

      for i in range(1, len(section_fields)):
        row = section_fields[i]
        if i == len(section_fields) - 1:
          for nested in nested_fields:
            section.add_field(nested)
          nested_fields = []
          wrapper.add_section(section)

        if row[0] == '#' or row[0] == ';':
          section.add_field(CommentField(value=row[1:].strip()))
          print()

        elif is_nested(row):
          if row.split('.')[0] in [field.get_key() for field in
    nested_fields]:
            continue
          nested_fields.append(find_subfields(prefix=row.split('.')[0],
    fields=section_fields))

        else:
          section.add_field(ValueField(*row.split('=')))

    return wrapper


  @staticmethod
  def serialize(object: Wrapper, format: str):
    if format.lower() == 'ron':

      def write_field(out, key, value, type, indent):
        if type == "value_field":
          out.write(f"{indent}{key}: {value}\n")
          return
        if type == "comment_field":
          out.write(f"{indent}//{value}\n")
          return
        out.write(f"{indent}{key}: (\n")
        for field in value:
          write_field(out, field.get_key(), field.get_value(), field.
    get_type(), indent*2)
        out.write(f"{indent})\n")

      output = os.path.abspath(os.path.join(os.path.dirname('__file__'), '
    output', 'schedule_hand_written.ron'))
      with open(output, 'w', encoding='utf-8') as out:
        out.write("Schedule(\n")

        indent = "  "
```

```
        for section in object.get_sections():
          out.write(f"{indent}{section.get_name()}: {{\n")

          for field in section.get_fields():
            field_type = field.get_type()
            field_key = field.get_key()
            field_value = field.get_value()
            write_field(out, field_key, field_value, field_type, indent*2)

          out.write(f"{indent}}}\n")

        out.write(")")

    elif format.lower() == 'xml':

      def write_field(out, key, value, type, indent):
        if type == "value_field":
          out.write(f"{indent}<{key}>{value}</{key}>\n")
          return
        if type == "comment_field":
          out.write(f"{indent}<!-- {value} -->\n")
          return
        out.write(f"{indent}<{key}>\n")
        for field in value:
          write_field(out, field.get_key(), field.get_value(), field.
    get_type(), indent*2)
        out.write(f"{indent}</{key}>\n")

      output = os.path.abspath(os.path.join(os.path.dirname('__file__'), '
    output', 'schedule_hand_written.xml'))
      with open(output, 'w', encoding='utf-8') as out:
        out.write("""<?xml version="1.0" encoding="UTF-8"?>\n""")
        out.write("<main>\n")

        indent = "  "
        out.write(f"{indent}<Schedule>\n")

        for section in object.get_sections():
          out.write(f"{indent*2}<{section.get_name()}>\n")

          for field in section.get_fields():
            field_type = field.get_type()
            field_key = field.get_key()
            field_value = field.get_value()
            write_field(out, field_key, field_value, field_type, indent*3)
          out.write(f"{indent*2}</{section.get_name()}>\n")

        out.write(f"{indent}</Schedule>\n")
        out.write("</main>\n")
```

Листинг 2: hand_written_convertor.py

```
"""Классыобертки
- для десериализации данных:удобны
 для промежуточного представления при переводе из одного формата файлов в другой.
"""

from typing import Type, List
from abc import ABC

```

```
9
10 class Wrapper:
11   def __init__(self, sections: List[Section]=[]) -> None:
12     self.__sections = sections
13
14   def add_section(self, section: Section) -> None:
15     self.__sections.append(section)
16
17   def get_sections(self) -> list[Section]:
18     return self.__sections
19
20
21 class Section:
22   def __init__(self, name: str, fields: List[Type[Field]]=[]) -> None:
23     self.__name = name
24     self.__fields = fields.copy()
25
26   def add_field(self, field: Field) -> None:
27     self.__fields.append(field)
28
29   def get_fields(self) -> list[Field]:
30     return self.__fields
31
32   def get_name(self) -> str:
33     return self.__name
34
35
36 class Field(ABC):
37   def __init__(self, key, value, type):
38     self.__key = key
39     self.__value = value
40     self.__type = type
41
42   def get_key(self) -> dict:
43     return self.__key
44
45   def get_value(self) -> dict:
46     return self.__value
47
48   def get_type(self) -> str:
49     return self.__type
50
51
52 class NestedField(Field):
53   def __init__(self, key: str):
54     self.__value: List[Type[Field]] = []
55     super().__init__(key, self.__value, "nested_field")
56
57   def add_field(self, field: ValueField) -> None:
58     self.__value.append(field)
59
60
61 class ValueField(Field):
62   def __init__(self, key: str, value: str):
63     super().__init__(key, value, "value_field")
64
65
66 class CommentField(Field):
67   def __init__(self, value: str):
68     super().__init__("!", value, "comment_field")
```

Листинг 3: formal.py

2. 2. Дополнительные задание 1

Разберем детально решение дополнительного задания 1.

- для конвертацию бинароного объекта, полученного в обязательном задании, в формат RON в классе HandWrittenConvertor был создан метод serialize 2

- данный метод проходит в цикле по каждой секции бинарного объекта, определяет тип поля, и в зависимости от него добавялет соотвествующий элемент RON

  Результат: ../src/output/schedule_hand_written.ron

2. 3. Дополнительные задание 2

Разберем детально решение дополнительного задания 2.

- для сериализации и десериализации исходных данных был создан класс AdvancedConvertor в файле 4

- данный класс содержит методы методы read_file, deserialize и serialize аналочичные HandWrittenConvertor, но написанные при помощи библиотек "configparser"для RON и "xml"для XML

  Результат ручного перевода: ../src/output/schedule_hand_written.ron

  Результат с использованием библиотек: ../src/output/schedule_advenced.ron

Результаты, полученные при помощи самописного парсера и парсера с использованием готовых бибилиотек имеют некторые сходства и раличия. Сходства состоят в том, что оба класса одинаково хорошо справляются с десериализацией исходного файла в бинарный объект.
Из различий можно выделить то, что самописный парсер добавляет необязательное имя структуры Shedule в начале RON файла (1), в то время как библиотека configparser этого не делает (2). Также самописный парсер позволяет обрабатывать вложенность в INI файле (разделение точкой), а библиотека configparser этого не делает.
Кроме того, самописный парсер умеет обрабатывать строки с комментариями, а configparser нет.

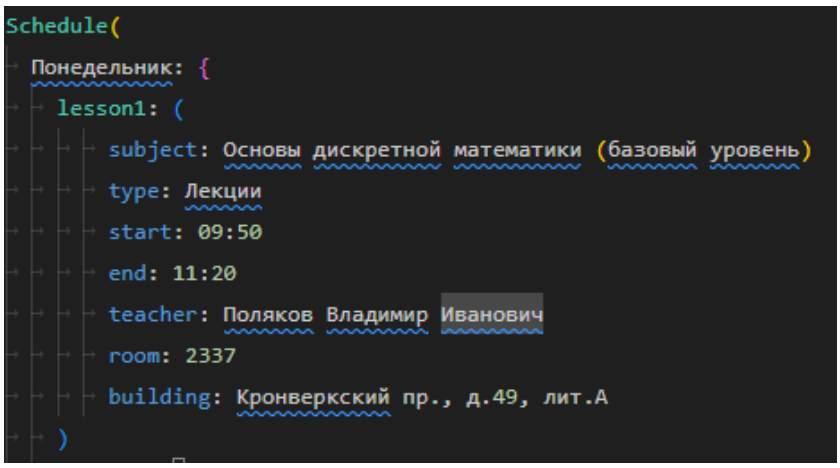

Рис. 1: ручной перевод

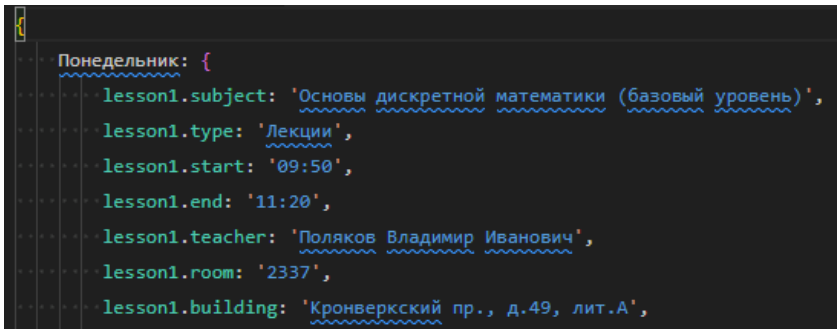

Рис. 2: перевод через библиотеку

```python
"""Конвертор
 для дополнительного задания №2.С
 использованием сторонних библиотек осуществляет:
- десериалиацию данных в формате INI ,
- сериализацию в формат RON
"""

from .convertor import Convertor
import configparser
import xml.etree.ElementTree as ET
import os


class AdvancedConvertor(Convertor):
  @staticmethod
  def deserialize(parse_object: configparser.ConfigParser) -> dict:
    return {section: dict(parse_object.items(section)) for section in
   parse_object.sections()}


  @staticmethod
  def serialize(object: dict, format: str, indent: int=0, return_value:
   bool=False):
    if format.lower() == 'ron':
      ron = "{\n"
      for key, value in object.items():
        if isinstance(value, dict):
```

```
26              ron += " " * (indent + 4) + f"{key}: {AdvancedConvertor.
    serialize(object=value, format='ron', indent=indent + 4, return_value=
    True).strip()}\n"
27          else:
28              ron += " " * (indent + 4) + f"{key}: {repr(value)},\n"
29      ron += " " * indent + "}"
30
31      if return_value:
32        return ron
33      output = os.path.abspath(os.path.join(os.path.dirname('__file__'), '
    output', 'schedule_advanced.ron'))
34      with open(output, "w", encoding='utf-8') as out:
35        out.write(ron)
36
37    elif format.lower() == 'xml':
38      root_name = 'main'
39      root = ET.Element(root_name)
40      for section, items in object.items():
41        section_elem = ET.SubElement(root, section)
42        for key, value in items.items():
43            item_elem = ET.SubElement(section_elem, key)
44            item_elem.text = value
45      if return_value:
46        return root
47      output = os.path.abspath(os.path.join(os.path.dirname('__file__'), '
    output', 'schedule_advanced.xml'))
48      tree = ET.ElementTree(root)
49      tree.write(output, encoding="utf-8", xml_declaration=True)
50
51  @staticmethod
52  def read_file(src: os.path) -> configparser.ConfigParser:
53    object = configparser.ConfigParser()
54    object.read(src, encoding='utf-8')
55    return object
```

Листинг 4: advanced_convertor.py

2. 4. Дополнительные задание 3

Разберем детально решение дополнительного задания 3.

- для конвертацию бинароного объекта, полученного в обязательном задании, в формат XML в классе HandWrittenConvertor был изменен метод serialize 2

- теперь данный метод принимает аргумент "format значение которого определяет перевод в нужный формат

- логика сериализации в XML практически не отличается от таковой в случае RON, за исключением видоизмененных полей файла в соответствии с форматом данных.

Результат: ../src/output/schedule_hand_written.ron

2. 5. Дополнительные задание 4

Разберем детально решение дополнительного задания 4.

- для проведения тестов был создан класс TestRunner 5
- данный класс содержит методы run_handwritten_test и run_advanced_test, которые осуществляют 100 итераций перевода при помощи рукописного парсера и парсера с использованием библиотек соотвественно

Результаты 100 итераций таковы:

- для рукописного парсера - затрачено времени 266.10255241394043 мс, 2.6610255241394043 мс на 1 итерацию
- для парсера с использованием библиотек - затрачено времени 83.90426635742188 мс, 0.8390426635742188 мс на 1 итерацию

Таким образом, парсера с использованием библиотек выполняет перевод в приблизительно 3-4 раз быстрее. Это может быть связано с тем, библиотеки используют оптимизированные алгоритмы, часто написанные на C/C++ или с использованием специализированных структур данных, что значительно ускоряет обработку.

```
""" Тесты для дополнительного задания №4 """

import time

from lib.advanced_convertor import AdvancedConvertor
from lib.hand_written_convertor import HandWrittenConvertor
from os import path


class TestRunner:
  @staticmethod
  def run_handwritten_test(src: path, iter_num: int=100):
    start_time = time.time()

    for _ in range(iter_num):
      ini_file_content = HandWrittenConvertor.read_file(src)
      deserialized_raw = HandWrittenConvertor.deserialize(file_content=
   ini_file_content)
      HandWrittenConvertor.serialize(object=deserialized_raw, format='ron'
   )

    delta_time = (time.time() - start_time) * 1000
    TestRunner.print_statistics(delta_time, "Собственный парсер")


  @staticmethod
  def run_advanced_test(src: path, iter_num: int=100):
    start_time = time.time()
```

```
27
28      for _ in range(iter_num):
29          ini_parser = AdvancedConvertor.read_file(src)
30          deserialized_advanced = AdvancedConvertor.deserialize(parse_object=
        ini_parser)
31          AdvancedConvertor.serialize(deserialized_advanced, format='ron')
32
33      delta_time = (time.time() - start_time) * 1000
34      TestRunner.print_statistics(delta_time, "С использованием библиотек")
35
36  @staticmethod
37  def print_statistics(delta_time, name):
38      print('='*40)
39      print(f"Результат 100 итераций - {name}")
40      print(f"Затрачено времени {delta_time} мс, {delta_time/100} мс на 1
        итерацию")
```

Листинг 5: test.py

3. Вывод

В процессе выполнения лабораторной работы по информатике я ознакомился с формой Бэкуса-Наура, изучил основные принципы организации формальных грамматик [1], а также ознакомился и научился работать с незнакомыми мне раннее языками разметки, INI, XML и RON [2].

4. Литература

[1] Орлов С. А., Цилькер Б. Я. Организация ЭВМ и систем: Учебник для вузов. 2-е изд. – СПб.: Питер, 2011. – 688 с.: ил., Приложение А «Арифметические основы вычислительных машин». URL: https://bit.ly/4dzgo3u (Дата обращения: 10.09.25)

[2] Алексеев Е.Г., Богатырев С.Д. Информатика. Мультимедийный электронный учебник. Раздел 3 «Системы счисления». URL: http://inf.e-alekseev.ru/text/Schisl.html (Дата обращения: 10.09.25)