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.formal import BinarySerializer, Wrapper
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'))
    BIN_FILE_PATH = path.abspath(path.join(path.dirname('__file__'), 'output', 'schedule_hand_written.bin'))

    print(f"Вариант: {501993 % 132}")

    # Обязательное задание: десериализация в бинарный объект
    ini_file_content = HandWrittenConvertor.read_file(INPUT_FILE_SRC)
    deserialized_raw = HandWrittenConvertor.deserialize(file_content=ini_file_content)
    BinarySerializer.save_to_file(deserialized_raw, BIN_FILE_PATH)

    # Дополнительное задание 1: загрузка из бинарного + сериализация в RON
    loaded_obj = BinarySerializer.load_from_file(BIN_FILE_PATH)
    HandWrittenConvertor.serialize(obj=loaded_obj, format='ron')

    # Дополнительное задание 2: парсинг через библиотеки
    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: загрузка из бинарного + сериализация в XML
    loaded_obj_xml = BinarySerializer.load_from_file(BIN_FILE_PATH)
    HandWrittenConvertor.serialize(obj=loaded_obj_xml, format='xml')

    # Дополнительное задание 4: 100 итераций тестов
    TestRunner.run_handwritten_test(INPUT_FILE_SRC, BIN_FILE_PATH, 100)
    TestRunner.run_advanced_test(INPUT_FILE_SRC, 100)
```

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, а затем в бинарный файл при помощи класса BinarySerializer 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)

```
"""Самописный
 конвертор для обязательного задания, дополнительного задания №1 и №3.Без
 использования сторонних библиотек осуществляет:
- десериалиацию данных из формата INI в бинарный объект,
- сериализацию из бинарного объекта в формат RON,
- сериализацию из бинарного объекта в формат XML
"""

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


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]) -> Wrapper:
        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)
            # Фильтруем поля начинающиеся с префикса
            relevant_fields = [f for f in fields if f.split('=')[0].
    startswith(prefix)]
```

```
37
38              # Словарь для группировки по следующему уровню вложенности
39              groups = {}
40              for field in relevant_fields:
41                  full_key, value = field.split('=', 1)
42                  sub_key = full_key[len(prefix) + 1:]
43                  if '.' not in sub_key:
44                      nested.add_field(ValueField(key=sub_key, value=value))
45                  else:
46                      group_key = sub_key.split('.')[0]
47                      groups.setdefault(group_key, []).append(field)
48
49              for group_key, group_fields in groups.items():
50                  nested.add_field(find_subfields(prefix=f"{prefix}.{
    group_key}", fields=group_fields, nested_prefix=group_key))
51
52              return nested
53
54
55          wrapper = Wrapper()
56
57          sections = '\n'.join(file_content).split('[')[1:]
58          for section_string in sections:
59              nested_fields = []
60              section_fields = list(filter(lambda r: r != '', section_string
    .split('\n')))
61              section = Section(name=section_fields[0][:-1])
62
63              for i in range(1, len(section_fields)):
64                  row = section_fields[i]
65
66                  if i == len(section_fields) - 1:
67                      for nested in nested_fields:
68                          section.add_field(nested)
69                      wrapper.add_section(section)
70
71                  if row[0] == '#' or row[0] == ';':
72                      section.add_field(CommentField(value=row[1:].strip()))
73                  elif is_nested(row):
74                      prefix = row.split('.')[0]
75                      # Проверяем, есть ли уже NestedField с этим ключом
76                      existing = next((f for f in nested_fields if f.get_key
    () == prefix), None)
77                      if existing is None:
78                          nested_fields.append(find_subfields(prefix=prefix,
     fields=section_fields))
79                  else:
80                      section.add_field(ValueField(*row.split('=', 1)))
81
82              return wrapper
83
84
85      @staticmethod
86      def serialize(obj: Wrapper, format: str):
87          # Используется для правильного отображения строк и чисел в RON файле
88          def wrapp_field_type(indent: str, key: str, value: str) -> str:
89              if value.isnumeric() and value != "true" and value != "false":
90                  return f"{indent}{key}: {value}\n"
91              return f"{indent}{key}: \"{value}\"\n"
92
```

```
93
94          format = format.lower()
95          if format == 'ron':
96              def write_field(out, key, value, type, indent):
97                  if type == "value_field":
98                      out.write(wrapp_field_type(indent, key, value))
99                      return
100                 if type == "comment_field":
101                     out.write(f"{indent}//{value}\n")
102                     return
103                 out.write(f"{indent}{key}: (\n")
104                 for field in value:
105                     write_field(out, field.get_key(), field.get_value(),
    field.get_type(), indent + "  ")
106                 out.write(f"{indent})\n")
107
108             output = os.path.abspath(os.path.join(os.path.dirname('
    __file__'), 'output', 'schedule_hand_written.ron'))
109             with open(output, 'w', encoding='utf-8') as out:
110                 out.write("Schedule(\n")
111                 indent = "  "
112                 for section in obj.get_sections():
113                     out.write(f"{indent}{section.get_name()}: (\n")
114                     for field in section.get_fields():
115                         write_field(out, field.get_key(), field.get_value
    (), field.get_type(), indent + "  ")
116                     out.write(f"{indent})\n")
117                 out.write(")")
118
119         elif format == 'xml':
120             def write_field(out, key, value, type, indent):
121                 if type == "value_field":
122                     out.write(f"{indent}<{key}>{value}</{key}>\n")
123                     return
124                 if type == "comment_field":
125                     out.write(f"{indent}<!-- {value} -->\n")
126                     return
127                 out.write(f"{indent}<{key}>\n")
128                 for field in value:
129                     write_field(out, field.get_key(), field.get_value(),
    field.get_type(), indent + "  ")
130                 out.write(f"{indent}</{key}>\n")
131
132             output = os.path.abspath(os.path.join(os.path.dirname('
    __file__'), 'output', 'schedule_hand_written.xml'))
133             with open(output, 'w', encoding='utf-8') as out:
134                 out.write("""<?xml version="1.0" encoding="UTF-8"?>\n""")
135                 out.write("<main>\n")
136                 indent = "  "
137                 out.write(f"{indent}<Schedule>\n")
138                 for section in obj.get_sections():
139                     out.write(f"{indent*2}<{section.get_name()}>\n")
140                     for field in section.get_fields():
141                         write_field(out, field.get_key(), field.get_value
    (), field.get_type(), indent * 3)
142                     out.write(f"{indent*2}</{section.get_name()}>\n")
143                 out.write(f"{indent}</Schedule>\n")
144                 out.write("</main>\n")
145
146         else:
```

```
            raise ValueError(f"Unknown format: {format}")
```

Листинг 2: hand_written_convertor.py

```
from typing import List, Optional
from abc import ABC


class Field(ABC):
    def __init__(self, key: str, value, type: str):
        self.__key = key
        self.__value = value
        self.__type = type

    def get_key(self) -> str:
        return self.__key

    def get_value(self):
        return self.__value

    def get_type(self) -> str:
        return self.__type

    def to_dict(self) -> dict:
        if self.get_type() == "nested_field":
            return {
                "key": self.__key,
                "value": [f.to_dict() for f in self.__value],
                "type": self.__type,
            }
        else:
            return {
                "key": self.__key,
                "value": self.__value,
                "type": self.__type,
            }

    def to_bytes(self) -> bytes:
        import json
        # Преобразуем dict в JSON, затем в байты
        return json.dumps(self.to_dict()).encode("utf-8")

    @staticmethod
    def from_dict(data: dict) -> 'Field':
        f_type = data["type"]
        if f_type == "value_field":
            return ValueField(data["key"], data["value"])
        elif f_type == "comment_field":
            return CommentField(data["value"])
        elif f_type == "nested_field":
            nested = NestedField(data["key"])
            for fdata in data["value"]:
                nested.add_field(Field.from_dict(fdata))
            return nested
        else:
            raise ValueError(f"Unknown field type {f_type}")

    @staticmethod
    def from_bytes(bdata: bytes) -> 'Field':
        import json
```

```
        data = json.loads(bdata.decode("utf-8"))
        return Field.from_dict(data)


class ValueField(Field):
    def __init__(self, key: str, value: str):
        super().__init__(key, value, "value_field")


class CommentField(Field):
    def __init__(self, value: str):
        super().__init__("!", value, "comment_field")


class NestedField(Field):
    def __init__(self, key: str):
        self.__value: List[Field] = []
        super().__init__(key, self.__value, "nested_field")

    def add_field(self, field: Field) -> None:
        self.__value.append(field)

    def get_fields(self) -> List[Field]:
        return self.__value


class Section:
    def __init__(self, name: str, fields: Optional[List[Field]] = None) ->
    None:
        self.__name = name
        self.__fields = fields if fields is not None else []

    def add_field(self, field: Field) -> None:
        self.__fields.append(field)

    def get_fields(self) -> List[Field]:
        return self.__fields

    def get_name(self) -> str:
        return self.__name

    def to_dict(self) -> dict:
        return {
            "name": self.__name,
            "fields": [f.to_dict() for f in self.__fields],
        }

    def to_bytes(self) -> bytes:
        import json
        return json.dumps(self.to_dict()).encode("utf-8")

    @staticmethod
    def from_dict(data: dict) -> 'Section':
        section = Section(data["name"])
        for fdata in data["fields"]:
            section.add_field(Field.from_dict(fdata))
        return section

    @staticmethod
    def from_bytes(bdata: bytes) -> 'Section':
```

```
        import json
        data = json.loads(bdata.decode("utf-8"))
        return Section.from_dict(data)


class Wrapper:
    def __init__(self, sections: Optional[List[Section]] = None) -> None:
        self.__sections = sections if sections is not None else []

    def add_section(self, section: Section) -> None:
        self.__sections.append(section)

    def get_sections(self) -> List[Section]:
        return self.__sections

    def to_dict(self) -> dict:
        return {
            "sections": [s.to_dict() for s in self.__sections]
        }

    def to_bytes(self) -> bytes:
        import json
        return json.dumps(self.to_dict()).encode("utf-8")

    @staticmethod
    def from_dict(data: dict) -> 'Wrapper':
        wrapper = Wrapper()
        for sdata in data["sections"]:
            wrapper.add_section(Section.from_dict(sdata))
        return wrapper

    @staticmethod
    def from_bytes(bdata: bytes) -> 'Wrapper':
        import json
        data = json.loads(bdata.decode("utf-8"))
        return Wrapper.from_dict(data)


class BinarySerializer:
    @staticmethod
    def save_to_file(obj: object, filename: str) -> None:
        with open(filename, 'wb') as f:
            f.write(obj.to_bytes())

    @staticmethod
    def load_from_file(filename: str) -> object:
        with open(filename, 'rb') as f:
            data = f.read()
        return Wrapper.from_bytes(data)
```

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.
Другой особенностью является правильное выделение строк двойными ковычками. Библиотека ставит одинарные ковычки, хотя это неверно с точки зрения синтаксиса языка разметки.

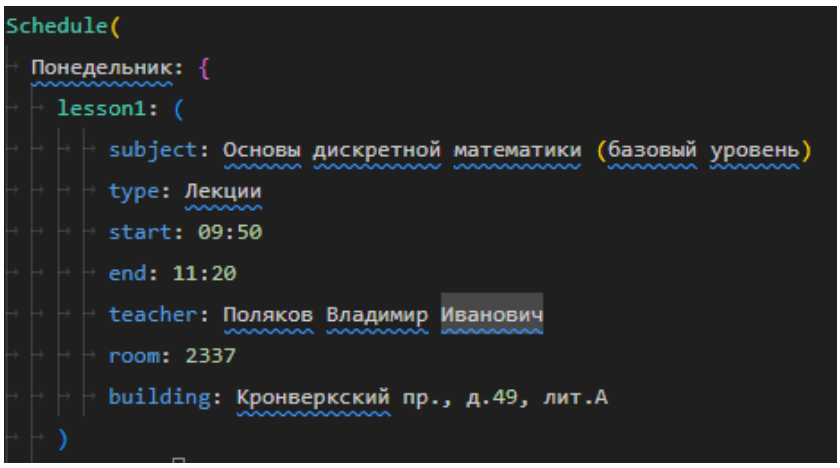

Рис. 1: ручной перевод

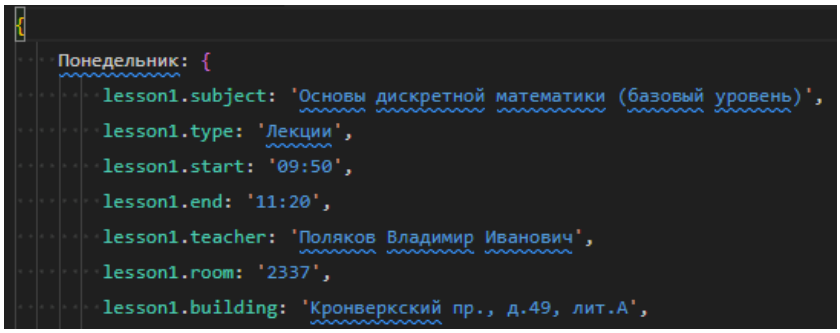

Рис. 2: перевод через библиотеку

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

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 88.102552 мс, 0.881025 мс на 1 итерацию
- для парсера с использованием библиотек - затрачено времени 120.904266 мс, 1.839042 мс на 1 итерацию

Таким образом, парсер с использованием библиотек выполняет перевод приблизительно на 36% медленнее. Это может быть связано с тем,что мой парсер сделан под узкую задачу, без дополнительной универсальной логики и проверок, характерных для библиотек, что уменьшает накладные расходы. Учитываются только нужные структуры и синтаксис, что ускоряет обработку и парсинг.

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

import time

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


class TestRunner:
  @staticmethod
  def run_handwritten_test(src: path, bin_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)
      BinarySerializer.save_to_file(deserialized_raw, bin_src)
      loaded_obj = BinarySerializer.load_from_file(bin_src)
      HandWrittenConvertor.serialize(obj=loaded_obj, format='ron')

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


```

```
27    @staticmethod
28    def run_advanced_test(src: path, iter_num: int=100):
29      start_time = time.time()
30
31      for _ in range(iter_num):
32        ini_parser = AdvancedConvertor.read_file(src)
33        deserialized_advanced = AdvancedConvertor.deserialize(parse_object=
      ini_parser)
34        AdvancedConvertor.serialize(deserialized_advanced, format='ron')
35
36      delta_time = (time.time() - start_time) * 1000
37      TestRunner.print_statistics(delta_time, "С использованием библиотек")
38
39    @staticmethod
40    def print_statistics(delta_time, name):
41      print('='*40)
42      print(f"Результат 100 итераций - {name}")
43      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)