

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: Р3109

Проверил:

Малышева Т. А.

Санкт-Петербург, 2025

Содержание

1. Задание	2
2. Основные этапы вычислений	4
1. Обязательное задание	4
2. Дополнительные задание 1	10
3. Дополнительные задание 2	10
4. Дополнительные задание 3	12
5. Дополнительные задание 4	13
3. Вывод	15
4. Литература	16

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 использует 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 бинарного объекта, полученного в обязательном задании, в новый формат (=серIALIZАЦИЮ);
 - для решения задачи использует формальные грамматики;

- не использует 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 сериализация происходила 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, находится в файле 1:

```
1 from lib.advanced_convertor import AdvancedConvertor
2 from lib.hand_written_convertor import HandWrittenConvertor
3 from lib.test import TestRunner
4 from os import path
5
6 if __name__ == "__main__":
7     INPUT_FILE_SRC = path.abspath(path.join(path.dirname('__file__'), 'input',
8         'schedule.ini'))
9
10    print(f"Вариант: {501993 % 132}\n-----|")
11
12    # Обязательное задание
13    ini_file_content = HandWrittenConvertor.read_file(INPUT_FILE_SRC)
14    deserialized_raw = HandWrittenConvertor.deserialize(file_content=
15        ini_file_content)
16
17    # Дополнительное задание 1
18    HandWrittenConvertor.serialize(object=deserialized_raw, format='ron')
19
20    # Дополнительное задание 2
21    # ini_parser = AdvancedConvertor.read_file(INPUT_FILE_SRC)
22    # deserialized_advanced = AdvancedConvertor.deserialize(parse_object=
23        ini_parser)
24    # AdvancedConvertor.serialize(deserialized_advanced, format='ron')
25
26    # Дополнительное задание 3
27    HandWrittenConvertor.serialize(object=deserialized_raw, format='xml')
28
29    # Дополнительное задание 4
30    # TestRunner.run_handwritten_test(INPUT_FILE_SRC)
31    # 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 секции исходного файла (класс Section), которые в свою очередь состоят из полей разного типа (NestedField - для вложенных, ValueField - поле ключ+значения, и CommentField - комментарий)

```

1 """ Конвертор
2 для обязательного задания, дополнительного задания №1 и №3. Без
3 использования сторонних библиотек осуществляет:
4 - десериализацию данных в формате INI,
5 - сериализацию в формат RON,
6 - сериализацию в формат XML
7 """
8
9 from .convertor import Convertor
10 from .formal import Wrapper, Section, ValueField, CommentField,
11     NestedField
12 from typing import Sequence
13 import os
14
15 class HandWrittenConvertor(Convertor):
16     @staticmethod
17     def read_file(src: os.path):
18         with open(src, 'r', encoding='UTF-8') as f:
19             return list(filter(lambda row: row != "", map(lambda row: row.strip(),
20 (), f.readlines())))
21
22     @staticmethod
23     def deserialize(file_content: Sequence[str]):
24
25         def is_section(row):
26             return row.startswith('[') and row.endswith(']')
27
28         def is_nested(row):
29             prefix = row.split(',') [0]
30             return not is_section(row) and row.find(prefix)+len(prefix) < row.
31             find(',')
32
33         def find_subfields(prefix: str, fields: Sequence, nested_prefix: str="") -> NestedField:
34             if nested_prefix == "":
35                 nested_prefix = prefix
36             nested = NestedField(key=nested_prefix)
37             fields = list(filter(lambda f: f.split('=')[0].startswith(prefix),
38                                 fields))
39             for field in fields:
40                 full_field_key, value = field.split('=')[1:]
41                 actual_field_key = full_field_key[len(prefix)+1:]
42
43                 if not '.' in actual_field_key:
44                     nested.add_field(ValueField(key=full_field_key.split(',')[-1],
45                                         value=value))
46                 else:
47                     nested.add_field(find_subfields(full_field_key, fields,
48                                         actual_field_key.split(',') [0]))

```

```

44     return nested
45
46
47     wrapper = Wrapper()
48
49     sections = '\n'.join(file_content).split(',') [1:]
50     for section_string in sections:
51         nested_fields = list()
52         section_fields = list(filter(lambda r: r != '', section_string.split(
53             ('\n'))))
54         section = Section(name=section_fields[0][:-1])
55
56         for i in range(1, len(section_fields)):
57             row = section_fields[i]
58             if i == len(section_fields) - 1:
59                 for nested in nested_fields:
60                     section.add_field(nested)
61                 nested_fields = []
62                 wrapper.add_section(section)
63
63             if row[0] == '#' or row[0] == ';':
64                 section.add_field(CommentField(value=row[1:].strip()))
65
66             elif is_nested(row):
67                 if row.split('.')[0] in [field.get_key() for field in
68                     nested_fields]:
69                     continue
70                 nested_fields.append(find_subfields(prefix=row.split('.')[0],
71                     fields=section_fields))
72
73             else:
74                 section.add_field(ValueField(*row.split('=')))
75
76     # sections = wrapper.get_sections()
77     # for sec in sections:
78     #     print(sec.get_name())
79     #     for field in sec.get_fields():
80     #         print(field.get_key(), [f"key={f.get_key()} value={f.get_value()}"
81     #             type={f.get_type()} for f in field.get_value()])
82     #     print('-'*40+'<')
83
84     return wrapper
85
86
87
88     @staticmethod
89     def serialize(object: Wrapper, format: str):
90         if format.lower() == 'ron':
91
92             def write_field(out, key, value, type, indent):
93                 if type == "value_field":
94                     out.write(f"{indent}{key}: {value}\n")
95                     return
96                 if type == "comment_field":
97                     out.write(f"{indent}//{value}\n")
98                     return
99                 out.write(f"{indent}{key}: {\n")
100                for field in value:
101                    write_field(out, field.get_key(), field.get_value(),
102                        field.get_type(), indent*2)
103                out.write(f"{indent}}}\n")

```

```

99
100     output = os.path.abspath(os.path.join(os.path.dirname('__file__'), ' '
101                                     , 'schedule_hand_written.ron'))
102     with open(output, 'w', encoding='utf-8') as out:
103         out.write("Schedule(\n")
104
104     indent = "    "
105     for section in object.get_sections():
106         out.write(f"{indent}{section.get_name()}: {{\n")
107
108     for field in section.get_fields():
109         field_type = field.get_type()
110         field_key = field.get_key()
111         field_value = field.get_value()
112         write_field(out, field_key, field_value, field_type, indent*2)
113
114         out.write(f"{indent}}}\n")
115
116     out.write(")")
117
118 elif format.lower() == 'xml':
119
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.
130             get_type(), indent*2)
131             out.write(f"{indent}</{key}>\n")
132
132     output = os.path.abspath(os.path.join(os.path.dirname('__file__'), ' '
133                                     , 'schedule_hand_written.xml'))
134     with open(output, 'w', encoding='utf-8') as out:
135         out.write("xml version="1.0" encoding="UTF-8"?&gt;\n")
135         out.write("&lt;main&gt;\n")
136
137         indent = "    "
138         out.write(f"{indent}&lt;Schedule&gt;\n")
139
140         for section in object.get_sections():
141             out.write(f"{indent*2}&lt;{section.get_name()}&gt;\n")
142
143             for field in section.get_fields():
144                 field_type = field.get_type()
145                 field_key = field.get_key()
146                 field_value = field.get_value()
147                 write_field(out, field_key, field_value, field_type, indent*3)
148                 out.write(f"{indent*2}&lt;/{section.get_name()}&gt;\n")
149
150         out.write(f"{indent}&lt;/Schedule&gt;\n")
151         out.write("&lt;/main&gt;\n")
</pre

```

Листинг 2: hand_written_convertor.py

¹ """Классы обертки

```

2 - для десериализации данных: удобны
3 для промежуточного представления при переводе из одного формата файлов в другой.
4 """
5
6 from typing import Type, List
7 from abc import ABC
8
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 бинарного 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 добавляет соответствующий 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 `read_file`, `deserialize` и `serialize` аналогичные 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 библиотек имеют некоторые сходства и различия. 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 полей точкой), а библиотека configparser этого не делает.

```

Schedule(
+ Понедельник: {
+   lesson1: {
+     subject: Основы дискретной математики (базовый уровень)
+     type: Лекции
+     start: 09:50
+     end: 11:20
+     start.test1: {
+       : 11111
+     }
+     end.test2: {
+       : 22222
+       lol: 33
+       kek: 44
+     }
+   }
}

```

Рис. 1: ручной перевод

```

{
Понедельник: {
  lesson1.subject: 'Основы дискретной математики (базовый уровень)',
  lesson1.type: 'Лекции',
  lesson1.start: '09:50',
  lesson1.end: '11:20',
  lesson1.start.test1: '11111',
  lesson1.end.test2: '22222',
  lesson1.end.test2.lol: '33',
  lesson1.end.test2.kek: '44',
  lesson1.teacher: 'Поляков Владимир Иванович',
  lesson1.room: '2337',
  lesson1.building: 'Кронверкский пр., д.49, лит.А',
  lesson2.subject: 'Основы дискретной математики (базовый уровень)',
  lesson2.type: 'Лекции',
  lesson2.start: '11:30',
  lesson2.end: '13:00',
  lesson2.teacher: 'Поляков Владимир Иванович',
  lesson2.room: '2337',
  lesson2.building: 'Кронверкский пр., д.49, лит.А',
}

```

Рис. 2: перевод через библиотеку

```

1 """ Конвертор
2 для дополнительного задания №2.С
3 использованием сторонних библиотек осуществляется:
4 - десериализацию данных в формате INI ,
5 - сериализацию в формат RON
"""
6
7
8 from .convertor import Convertor
9 import configparser
10 import xml.etree.ElementTree as ET
11 import os
12
13

```

```

14 class AdvancedConvertor(Convertor):
15     @staticmethod
16     def deserialize(parse_object: configparser.ConfigParser) -> dict:
17         return {section: dict(parse_object.items(section)) for section in
18                 parse_object.sections()}
19
20     @staticmethod
21     def serialize(object: dict, format: str, indent: int=0, return_value:
22                   bool=False):
23         if format.lower() == 'ron':
24             ron = "{\n"
25             for key, value in object.items():
26                 if isinstance(value, dict):
27                     ron += " " * (indent + 4) + f"{key}: {AdvancedConvertor.
28                     serialize(object=value, format='ron', indent=indent + 4, return_value=
29                     True).strip()}\n"
30                 else:
31                     ron += " " * (indent + 4) + f"{key}: {repr(value)}},\n"
32             ron += " " * indent + "}"
33
34             if return_value:
35                 return ron
36             output = os.path.abspath(os.path.join(os.path.dirname('__file__'), ,
37             'output', 'schedule_advanced.ron'))
38             with open(output, "w", encoding='utf-8') as out:
39                 out.write(ron)
40
41             elif format.lower() == 'xml':
42                 root_name = 'main'
43                 root = ET.Element(root_name)
44                 for section, items in object.items():
45                     section_elem = ET.SubElement(root, section)
46                     for key, value in items.items():
47                         item_elem = ET.SubElement(section_elem, key)
48                         item_elem.text = value
49                     if return_value:
50                         return root
51                     output = os.path.abspath(os.path.join(os.path.dirname('__file__'), ,
52                     'output', 'schedule_advanced.xml'))
53                     tree = ET.ElementTree(root)
54                     tree.write(output, encoding="utf-8", xml_declaration=True)
55
56     @staticmethod
57     def read_file(src: os.path) -> configparser.ConfigParser:
58         object = configparser.ConfigParser()
59         object.read(src, encoding='utf-8')
60         return object

```

Листинг 4: advanced_convertor.py

2. 4. Дополнительные задание 3

Разберем детально решение дополнительного задания 3.

- для конвертацию бинарного 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 соответственно

Результаты 100 итераций таковы:

- для рукописного парсера - затрачено времени 728.8212776184082 мс, 7.288212776184082 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 8-9 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.

```
1 """ Тесты для дополнительного задания №4 """
2
3 import time
4
5 from lib.advanced_convertor import AdvancedConvertor
6 from lib.hand_written_convertor import HandWrittenConvertor
7 from os import path
8
9
10 class TestRunner:
11     @staticmethod
12     def run_handwritten_test(src: path, iter_num: int=100):
13         start_time = time.time()
```

```
14
15     for _ in range(iter_num):
16         ini_file_content = HandWrittenConvertor.read_file(src)
17         deserialized_raw = HandWrittenConvertor.deserialize(file_content=
18             ini_file_content)
19         HandWrittenConvertor.serialize(object=deserialized_raw, format='ron')
20
21     delta_time = (time.time() - start_time) * 1000
22     TestRunner.print_statistics(delta_time, "Собственный парсер")
23
24 @staticmethod
25 def run_advanced_test(src: path, iter_num: int=100):
26     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=
31             ini_parser)
32         AdvancedConvertor.serialize(deserialized_advanced, format='ron')
33
34     delta_time = (time.time() - start_time) * 1000
35     TestRunner.print_statistics(delta_time, "С использованием библиотек")
36
37 @staticmethod
38 def print_statistics(delta_time, name):
39     print('='*40)
40     print(f"Результат 100 итераций - {name}")
41     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 ранее 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)