

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 (=сериализацию);
 - 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 производилась 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.formal import BinarySerializer, Wrapper
4 from lib.test import TestRunner
5 from os import path
6
7
8 if __name__ == "__main__":
9     INPUT_FILE_SRC = path.abspath(path.join(path.dirname('__file__'), '
input', 'schedule.ini'))
10    BIN_FILE_PATH = path.abspath(path.join(path.dirname('__file__'), '
output', 'schedule_hand_written.bin'))
11
12    print(f"Вариант: {501993 % 132}")
13
14    # Обязательное задание: десериализация в бинарный объект
15    ini_file_content = HandWrittenConvertor.read_file(INPUT_FILE_SRC)
16    deserialized_raw = HandWrittenConvertor.deserialize(file_content=
ini_file_content)
17    BinarySerializer.save_to_file(deserialized_raw, BIN_FILE_PATH)
18
19    # Дополнительное задание 1: загрузка из бинарного + сериализация в RON
20    loaded_obj = BinarySerializer.load_from_file(BIN_FILE_PATH)
21    HandWrittenConvertor.serialize(obj=loaded_obj, format='ron')
22
23    # Дополнительное задание 2: парсинг через библиотеки
24    ini_parser = AdvancedConvertor.read_file(INPUT_FILE_SRC)
25    deserialized_advanced = AdvancedConvertor.deserialize(parse_object=
ini_parser)
26    AdvancedConvertor.serialize(deserialized_advanced, format='ron')
27
28    # Дополнительное задание 3: загрузка из бинарного + сериализация в XML
29    loaded_obj_xml = BinarySerializer.load_from_file(BIN_FILE_PATH)
30    HandWrittenConvertor.serialize(obj=loaded_obj_xml, format='xml')
31
32    # Дополнительное задание 4: 100 итераций тестов
33    TestRunner.run_handwritten_test(INPUT_FILE_SRC, BIN_FILE_PATH, 100)
34    TestRunner.run_advanced_test(INPUT_FILE_SRC, 100)
```

Листинг 1: main.py

Разберем детально решение обязательного задания.

- для парсинга и конвертации 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 секции исходного файла (класс Section), которые в свою очередь состоят из полей разного типа (NestedField - для вложенностей, ValueField - поле ключ+значения, и CommentField - комментариев)

```

1  """Самописный
2  конвертор для обязательного задания, дополнительного задания №1 и №3. Без
3  использования сторонних библиотек осуществляет:
4  - десериализацию данных из формата INI в бинарный объект,
5  - сериализацию из бинарного объекта в формат RON,
6  - сериализацию из бинарного объекта в формат XML
7  """
8
9  import os
10 from .convertor import Convertor
11 from typing import Sequence
12 from .formal import Wrapper, Section, ValueField, CommentField,
    NestedField
13 from .formal import BinarySerializer
14
15
16 class HandWrittenConvertor(Convertor):
17     @staticmethod
18     def read_file(src: os.path):
19         with open(src, 'r', encoding='UTF-8') as f:
20             return list(filter(lambda row: row != "", map(lambda row: row.
strip(), f.readlines()))))
21
22     @staticmethod
23     def deserialize(file_content: Sequence[str]) -> Wrapper:
24         def is_section(row):
25             return row.startswith('[') and row.endswith(']')
26
27         def is_nested(row):
28             prefix = row.split('.')[0]
29             return not is_section(row) and row.find(prefix) + len(prefix)
< row.find('=')
30
31         def find_subfields(prefix: str, fields: Sequence, nested_prefix:
str = "") -> NestedField:
32             if nested_prefix == "":
33                 nested_prefix = prefix
34             nested = NestedField(key=nested_prefix)
35             # Фильтруем поля начинающиеся с префикса
36             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(),
106 field.get_type(), indent + " ")
107                 out.write(f"{indent})\n")
108
109         output = os.path.abspath(os.path.join(os.path.dirname('__file__'), 'output', 'schedule_hand_written.ron'))
110         with open(output, 'w', encoding='utf-8') as out:
111             out.write("Schedule(\n")
112             indent = " "
113             for section in obj.get_sections():
114                 out.write(f"{indent}{section.get_name()}: (\n")
115                 for field in section.get_fields():
116                     write_field(out, field.get_key(), field.get_value
117 (), field.get_type(), indent + " ")
118                 out.write(f"{indent})\n")
119             out.write(")")
120
121     elif format == 'xml':
122         def write_field(out, key, value, type, indent):
123             if type == "value_field":
124                 out.write(f"{indent}<{key}>{value}</{key}>\n")
125                 return
126             if type == "comment_field":
127                 out.write(f"{indent}<!-- {value} -->\n")
128                 return
129             out.write(f"{indent}<{key}>\n")
130             for field in value:
131                 write_field(out, field.get_key(), field.get_value(),
132 field.get_type(), indent + " ")
133             out.write(f"{indent}</{key}>\n")
134
135         output = os.path.abspath(os.path.join(os.path.dirname('__file__'), 'output', 'schedule_hand_written.xml'))
136         with open(output, 'w', encoding='utf-8') as out:
137             out.write("<?xml version='1.0' encoding='UTF-8'?>\n")
138             out.write("<main>\n")
139             indent = " "
140             out.write(f"{indent}<Schedule>\n")
141             for section in obj.get_sections():
142                 out.write(f"{indent*2}<{section.get_name()}>\n")
143                 for field in section.get_fields():
144                     write_field(out, field.get_key(), field.get_value
145 (), field.get_type(), indent * 3)
146                 out.write(f"{indent*2}</{section.get_name()}>\n")
147             out.write(f"{indent}</Schedule>\n")
148             out.write("</main>\n")
149
150     else:

```



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

Листинг 2: hand_written_converter.py

```
1 import pickle
2 from typing import List, Optional
3 from abc import ABC
4
5
6 class Field(ABC):
7     def __init__(self, key: str, value: str, type: str):
8         self.__key = key
9         self.__value = value
10        self.__type = type
11
12    def get_key(self) -> str:
13        return self.__key
14
15    def get_value(self) -> str:
16        return self.__value
17
18    def get_type(self) -> str:
19        return self.__type
20
21    def to_bytes(self) -> bytes:
22        # Сериализация объекта с помощью pickle
23        return pickle.dumps({
24            "key": self.__key,
25            "value": self.__value,
26            "type": self.__type,
27        })
28
29
30 class ValueField(Field):
31     def __init__(self, key: str, value: str):
32         super().__init__(key, value, "value_field")
33
34
35 class CommentField(Field):
36     def __init__(self, value: str):
37         super().__init__("!", value, "comment_field")
38
39
40 class NestedField(Field):
41     def __init__(self, key: str):
42         self.__value: List[Field] = []
43         super().__init__(key, self.__value, "nested_field")
44
45     def add_field(self, field: Field) -> None:
46         self.__value.append(field)
47
48     def get_fields(self) -> List[Field]:
49         return self.__value
50
51     def to_bytes(self) -> bytes:
52         # Сериализовать список вложенных полей
53         return pickle.dumps({
54             "key": self.get_key(),
55             "value": [f.to_bytes() for f in self.__value],
56             "type": self.get_type(),
```

```

57         })
58
59
60     class Section:
61         def __init__(self, name: str, fields: Optional[List[Field]] = None) ->
        None:
62             self.__name = name
63             self.__fields = fields if fields is not None else []
64
65         def add_field(self, field: Field) -> None:
66             self.__fields.append(field)
67
68         def get_fields(self) -> List[Field]:
69             return self.__fields
70
71         def get_name(self) -> str:
72             return self.__name
73
74         def to_bytes(self) -> bytes:
75             # Сериализовать секцию: имя и список полей
76             return pickle.dumps({
77                 "name": self.__name,
78                 "fields": [f.to_bytes() for f in self.__fields],
79             })
80
81
82     class Wrapper:
83         def __init__(self, sections: Optional[List[Section]] = None) -> None:
84             self.__sections = sections if sections is not None else []
85
86         def add_section(self, section: Section) -> None:
87             self.__sections.append(section)
88
89         def get_sections(self) -> List[Section]:
90             return self.__sections
91
92         def to_bytes(self) -> bytes:
93             return pickle.dumps(self)
94
95
96     class BinarySerializer:
97         @staticmethod
98         def save_to_file(obj: object, filename: str) -> None:
99             with open(filename, 'wb') as f:
100                 f.write(obj.to_bytes())
101
102         @staticmethod
103         def load_from_file(filename: str) -> object:
104             with open(filename, 'rb') as f:
105                 data = f.read()
106                 return pickle.loads(data)

```

Листинг 3: formal.py

2. 2. Дополнительные задание 1

Разберем детально решение дополнительного задания 1.

- для конвертации 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 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 кавычками. Библиотека ставит одинарные кавычки, хотя это неверно с точки зрения синтаксиса языка разметки.

```

Schedule(
  Понедельник: {
    lesson1: (
      subject: Основы дискретной математики (базовый уровень)
      type: Лекции
      start: 09:50
      end: 11:20
      teacher: Поляков Владимир Иванович
      room: 2337
      building: Кронверкский пр., д.49, лит.А
    )
  )
)

```

Рис. 1: ручной перевод

```

{
  Понедельник: {
    lesson1.subject: 'Основы дискретной математики (базовый уровень)',
    lesson1.type: 'Лекции',
    lesson1.start: '09:50',
    lesson1.end: '11:20',
    lesson1.teacher: 'Поляков Владимир Иванович',
    lesson1.room: '2337',
    lesson1.building: 'Кронверкский пр., д.49, лит.А',
  }
}

```

Рис. 2: перевод через библиотеку

```

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

```

```

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 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`
- 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 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.

```
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 lib.formal import BinarySerializer
8  from os import path
9
10
11 class TestRunner:
12     @staticmethod
13     def run_handwritten_test(src: path, bin_src: path, iter_num: int=100,):
14         start_time = time.time()
15
16         for _ in range(iter_num):
17             ini_file_content = HandWrittenConvertor.read_file(src)
18             deserialized_raw = HandWrittenConvertor.deserialize(file_content=
ini_file_content)
19             BinarySerializer.save_to_file(deserialized_raw, bin_src)
20             loaded_obj = BinarySerializer.load_from_file(bin_src)
21             HandWrittenConvertor.serialize(obj=loaded_obj, format='ron')
22
23         delta_time = (time.time() - start_time) * 1000
24         TestRunner.print_statistics(delta_time, "Собственный парсер")
25
26
```

```

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 неизвестными 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)