НЕГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ ЧАСТНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «МОСКОВСКИЙ ФИНАНСОВО-ПРОМЫШЛЕННЫЙ УНИВЕРСИТЕТ “СИНЕРГИЯ”»

Факультет/Институт Информационных технологий
(наименование факультета/ Института)

Направление/специальность подготовки: 09.02.07 Информационные системы и программирование
(код и наименование направления /специальности подготовки)

Форма обучения: очная
(очная, очно-заочная, заочная)

Отчет по лабораторной работе № 2

на тему Разработка тестовых пакетов
(наименование темы)

по дисциплине Тестирование информационных систем
(наименование дисциплины)

Обучающийся Грачев Дмитрий Александрович _____
(ФИО) (подпись)

Группа ДКИП-312

Преподаватель Авдеенков Владимир Александрович _____
(ФИО) (подпись)

Москва 2024 г

Лабораторная работа №2. «Разработка тестовых пакетов»

Задание 1:

В Древней Греции (II в. до н.э.) был известен шифр, называемый "квадрат Полибия". Шифровальная таблица представляла собой квадрат с пятью столбцами и пятью строками, которые нумеровались цифрами от 1 до 5. В каждую клетку такого квадрата записывалась одна буква. В результате каждой букве соответствовала пара чисел, и шифрование сводилось к замене буквы парой чисел. Для латинского алфавита квадрат Полибия имеет вид:

	1	2	3	4	5
1	A	B	C	D	E
2	F	G	H	I, J	K
3	L	M	N	O	P
4	Q	R	S	T	U
5	V	W	X	Y	Z

Пользуясь вышеизложенным способом написать программу, которая:

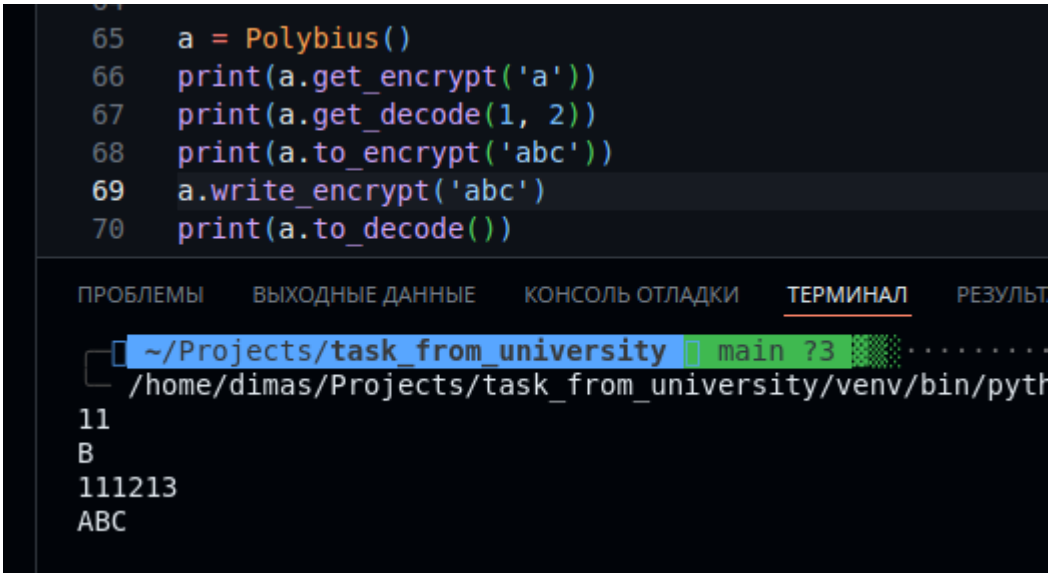
а) зашифрует введенный текст и выведет на экран;

б) считает зашифрованный текст и расшифрует данный текст (*пункт б выполнить по желанию и возможностям*).

1. Код программы на языке Python:

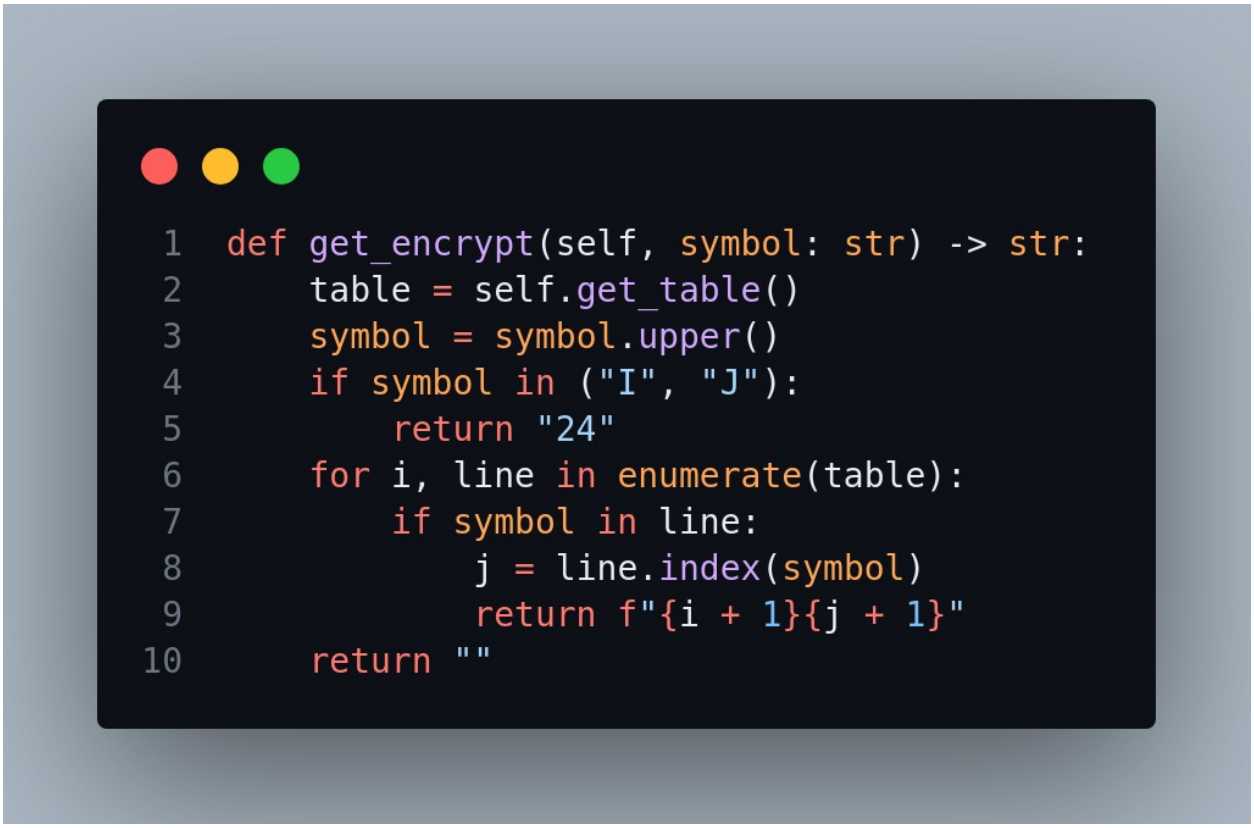
```python
from typing import Tuple


class Polybius:
    def __init__(self, filename: str = "crypt.txt") -> None:
        self.filename = filename


    @staticmethod
    def get_table() -> Tuple[Tuple[str, ...], ...]:
        return (
            ("A", "B", "C", "D", "E"),
            ("F", "G", "H", "IJ", "K"),
            ("L", "M", "N", "O", "P"),
            ("Q", "R", "S", "T", "U"),
            ("V", "W", "X", "Y", "Z")
        )


    def get_encrypt(self, symbol: str) -> str:
        table = self.get_table()
        symbol = symbol.upper()
        if symbol in ("I", "J"):
            return "24"
        for i, line in enumerate(table):
            if symbol in line:
                j = line.index(symbol)
                return f"{i + 1}{j + 1}"
        return ""


    def to_encrypt(self, text: str) -> str:
        crypt_text = ""
        for symbol in text:
            encrypted = self.get_encrypt(symbol)
            crypt_text += encrypted if encrypted else symbol
        return crypt_text


    def write_encrypt(self, text: str) -> None:
        with open(self.filename, "w") as file:
            file.write(self.to_encrypt(text))


    def read_encrypt(self) -> str:
        with open(self.filename) as file:
            return file.read()


    def get_decode(self, i: int, j: int) -> str:
        table = self.get_table()
        return table[i-1][j-1] if 0 <= i - 1 <= 5 and 0 <= j - 1 <= 5 else ""


    def to_decode(self) -> str:
        encrypt_text = self.read_encrypt()
        text = ""
        while len(encrypt_text) >= 2:
            i = int(encrypt_text[0])
            j = int(encrypt_text[1])
            text += self.get_decode(i, j)
            encrypt_text = encrypt_text[2:]
        return text
```

Примеры работы программы



Задание 2-3:

1) Проверка шифрования символа



```
1  def get_encrypt(self, symbol: str) -> str:
2      table = self.get_table()
3      symbol = symbol.upper()
4      if symbol in ("I", "J"):
5          return "24"
6      for i, line in enumerate(table):
7          if symbol in line:
8              j = line.index(symbol)
9              return f"{i + 1}{j + 1}"
10     return ""
```

полное покрытие данного блока возможно достичь за счет перебора всех возможных аргументов и соответствующих им значений

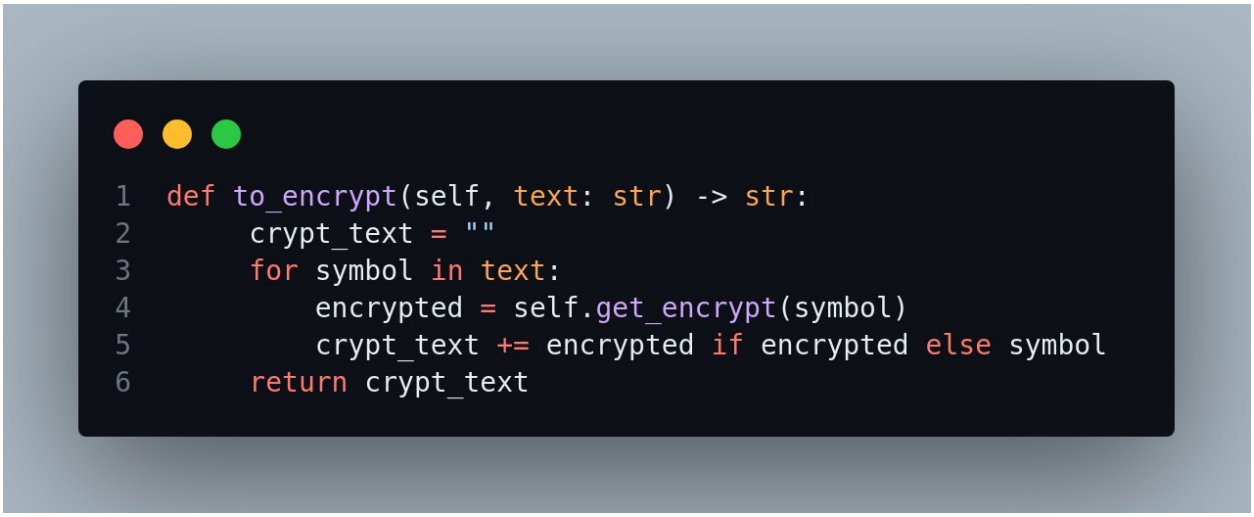
Тест	Ожидаемый результат	Фактический результат	Результат
Символ A	11	11	успех
Символ B	12	12	успех
Символ C	13	13	успех
Символ D	14	14	успех
Символ E	15	15	успех
Символ F	21	21	успех
Символ G	22	22	успех
Символ H	23	23	успех
Символ I	24	24	успех
Символ J	24	24	успех
Символ K	25	25	успех
Символ L	31	31	успех
Символ M	32	32	успех
Символ N	33	33	успех
Символ O	34	34	успех
Символ P	35	35	успех
Символ Q	41	41	успех
Символ R	42	42	успех

Символ S	43	43	успех
Символ T	44	44	успех
Символ U	45	45	успех
Символ V	51	51	успех
Символ W	52	52	успех
Символ X	53	53	успех
Символ Y	54	54	успех
Символ Z	55	55	успех

Тестовый сценарий Pytest

```
data_get_encrypt = (
    ("a", "11"), ("b", "12"), ("c", "13"), ("d", "14"), ("e", "15"),
    ("f", "21"), ("g", "22"), ("h", "23"), ("i", "24"), ("j", "24"),
    ("k", "25"), ("l", "31"), ("m", "32"), ("n", "33"), ("o", "34"),
    ("p", "35"), ("q", "41"), ("r", "42"), ("s", "43"), ("t", "44"),
    ("u", "45"), ("v", "51"), ("w", "52"), ("x", "53"), ("y", "54"),
    ("z", "55"), ("", ""), (" ", ""), ("1", ""), ("@", "")
)


@pytest.mark.parametrize('symbol, result', data_get_encrypt)
def test_get_encrypt(symbol, result):
    """Тестирование кодирования символов"""
    assert polybius.get_encrypt(symbol) == result
```

2) Проверка шифрования строки



```
def to_encrypt(self, text: str) -> str:
    crypt_text = ""
    for symbol in text:
        encrypted = self.get_encrypt(symbol)
        crypt_text += encrypted if encrypted else symbol
    return crypt_text
```

Для проверки достаточно любой комбинации символов в верхнем и в нижнем регистре, например все возможные символы в одной строке (кодирование каждого символа проверено ранее)

Тест	Ожидаемый результат	Фактический результат	Результат
"abc"	"111213"	"111213"	успех
"xyz"	"535455"	"535455"	успех
"a b"	"11 12"	"11 12"	успех
"i j"	"24 24"	"24 24"	успех
"!@#"	"!@#"	"!@#"	успех
" "	" "	" "	успех
"123"	"123"	"123"	успех
"A"	"11"	"11"	успех
"B"	"12"	"12"	успех
"C"	"13"	"13"	успех
"I"	"24"	"24"	успех

Сценарий Pytest

```python
data_to_encrypt = (
    ("abc", "111213"), ("xyz", "535455"), ("a b", "11 12"),
    ("i j", "24 24"), ("!@#", "!@#"), ("  ", "  "),
    ("123", "123"), ("A", "11"), ("B", "12"),
    ("C", "13"), ("I", "24")
)

@pytest.mark.parametrize('text, result', data_to_encrypt)
def test_to_encrypt(text, result):
    """Тестирование кодирования текста"""
    assert polybius.to_encrypt(text) == result

```

3)Проверка записи в файл, чтения из файла

```python
def write_encrypt(self, text: str) -> None:
        with open(self.filename, "w") as file:
            file.write(self.to_encrypt(text))


    def read_encrypt(self) -> str:
        with open(self.filename) as file:
            return file.read()
```

Тест	Ожидаемый результат	Фактический результат	Результат
Запись\Чтение	Запись в файл	Запись в файл	успех

Запись\Чтение	Чтение из…	Чтение из…	успех

Сценарий Pytest

```python
data_write_read = (
    ("a", "A"), ("b", "B"), ("c", "C"), ("d", "D"), ("e", "E"),
    ("f", "F"), ("g", "G"), ("h", "H"), ("i", "I"), ("j", "J"),
    ("k", "K"), ("l", "L"), ("m", "M"), ("n", "N"), ("o", "O"),
    ("p", "P"), ("q", "Q"), ("r", "R"), ("s", "S"), ("t", "T"),
    ("u", "U"), ("v", "V"), ("w", "W"), ("x", "X"), ("y", "Y"),
    ("z", "Z"), ("", ""), (" ", ""), ("1", ""), ("@", "")
)


@pytest.mark.parametrize('symbol, result', data_write_read)
def test_write_read(symbol, result):
    """Тестирование шифрования и декодирования файла"""
    text = polybius.write_encrypt(symbol)
    assert polybius.to_decode() == result
```

4) Декодирование

```python
def get_decode(self, i: int, j: int) -> str:
        table = self.get_table()
        return table[i-1][j-1] if 0 <= i - 1 <= 5 and 0 <= j - 1 <= 5 else ""


    def to_decode(self) -> str:
        encrypt_text = self.read_encrypt()
        text = ""
        while len(encrypt_text) >= 2:
            i = int(encrypt_text[0])
            j = int(encrypt_text[1])
            text += self.get_decode(i, j)
            encrypt_text = encrypt_text[2:]
        return text
```

полное покрытие данного блока возможно достичь за счет перебора всех возможных аргументов и соответствующих им значений (для символов)

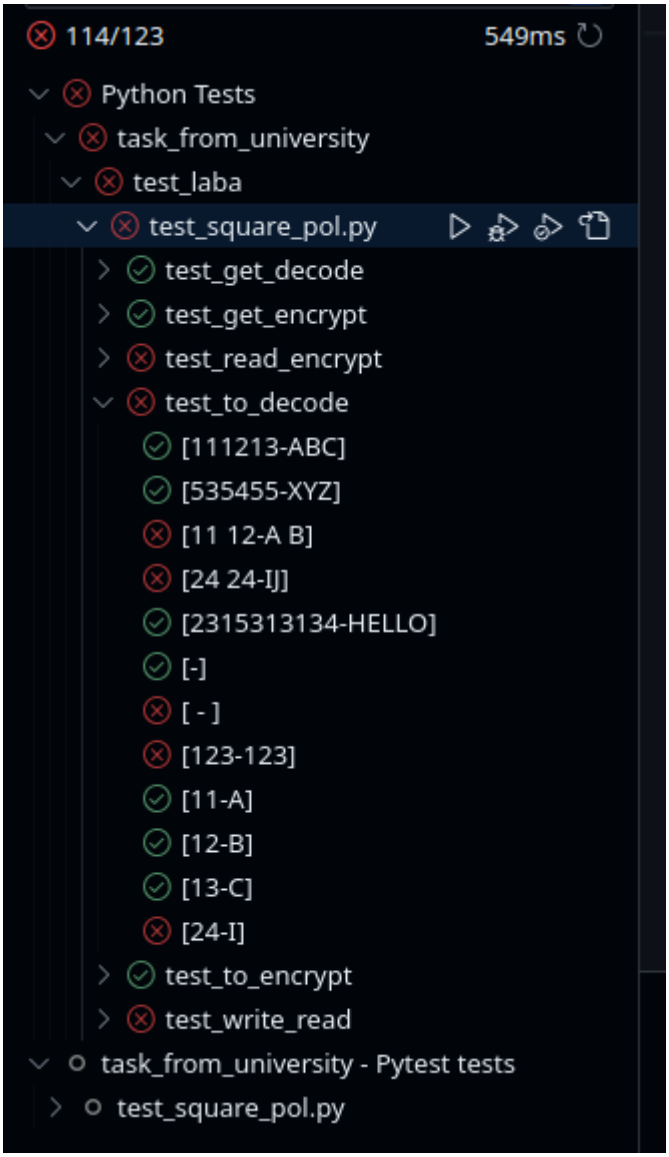
Тест	Ожидаемый результат	Фактический результат	Результат
11	A	A	успех
12	B	B	успех
13	C	C	успех
14	D	D	успех
15	E	E	успех
21	F	F	успех
22	G	G	успех
23	H	H	успех
24	I	I, J	Не успех
24	J	I, J	Не успех
25	K	K	успех
31	L	L	успех
32	M	M	успех
33	N	N	успех
34	O	O	успех
35	P	P	успех

41	Q	Q	успех
42	R	R	успех
43	S	S	успех
44	T	T	успех
45	U	U	успех
51	V	V	успех
52	W	W	успех
53	X	X	успех
54	Y	Y	успех
55	Z	Z	успех

Сценарий Pytest

```python
data_get_decode = (
    (1, 1, "A"), (1, 2, "B"), (1, 3, "C"), (1, 4, "D"), (1, 5, "E"),
    (2, 1, "F"), (2, 2, "G"), (2, 3, "H"), (2, 4, "IJ"), (2, 5, "K"),
    (3, 1, "L"), (3, 2, "M"), (3, 3, "N"), (3, 4, "O"), (3, 5, "P"),
    (4, 1, "Q"), (4, 2, "R"), (4, 3, "S"), (4, 4, "T"), (4, 5, "U"),
    (5, 1, "V"), (5, 2, "W"), (5, 3, "X"), (5, 4, "Y"), (5, 5, "Z"),
    (0, 0, ""), (6, 0, ""), (0, 6, "")
)

@pytest.mark.parametrize('i, j, result', data_get_decode)
def test_get_decode(i, j, result):
    """Тестирование расшифровки символов"""
    assert polybius.get_decode(i, j) == result
```

Итог автоматического тестирования на тестовом пакете



Вывод:

В ходе лабораторной работы написана программа, реализующая алгоритм шифрования «Квадрат Полибия», был разработан пакет тестовых сценариев, который полностью охватывает код программы. В ходе тестирования удалось обнаружить недостаток алгоритма при дешифровании, это связано с тем, что двум буквам (I, J) соответствует одна уникальная пара индексов таблицы (2, 4), и при обратном процессе - шифру 24 соответствуют два значения (I и J). Все остальные тесты пройдены успешно.