Министерство науки и образования РФ

Федеральное государственное бюджетное учреждение

высшего образования

**«Тверской государственный технический университет»**

(ТвГТУ)

Кафедра программного обеспечения

**Курсовая работа**

по дисциплине: «Теория алгоритмов»

Тема: «Машина Поста (работа с серией программ на входе)»

|  |
| --- |
| Выполнили:  студенты группы  Б.ПИН.РИС-21.06 |
| Проверила:  старший преподаватель кафедры ПО  Корнеева Е.И. |

Тверь 2023

**Цель проекта:** Разработать программу для работы с серией программ на входе, используя Машину Поста.

**Ключевые требования:**

- Разработка программы должна быть выполнена на языке Python;

- Программа должна иметь графический интерфейс пользователя (GUI);

- Программа должна содержать модульные тесты;

- Программа должна иметь техническую документацию

**Роли в проекте:**

1. Team Lead - Отвечает за общее руководство проектом, планирование и контроль выполнения работ, координацию команды.

2. Classes lead - Отвечает за разработку классов и методов для работы Машины Поста.

3. GUI lead - Отвечает за разработку графического интерфейса пользователя.

4. Test lead - Отвечает за разработку и проведение модульных тестов.

5. Docs lead - Отвечает за разработку документации.

**Теоретическая справка:**

Машина Поста состоит из каретки (или считывающей и записывающей головки) и разбитой на ячейки бесконечной в обе стороны ленты. Каждая ячейка ленты может находиться в 2 состояниях — быть либо пустой — 0, либо помеченной меткой 1. За такт работы машины каретка может сдвинуться на одну позицию влево или вправо, считать, изменить символ в своей текущей позиции.

Работа машины Поста определяется программой, состоящей из конечного числа строк. Для работы машины нужно задать программу и её начальное состояние (то есть состояние ленты и позицию каретки). Кареткой управляет программа, состоящая из пронумерованных не обязательно упорядоченных строк команд, если в каждой команде указана строка, на которую нужно перейти. Обычно принимается, что если в команде переход не указан, то переход происходит на следующую строку.

**Алгоритм Машины Поста:**

1. Задать начальное состояние ленты и позицию каретки.

2. Задать программу, состоящую из конечного числа строк команд.

3. Установить текущую команду на первую строку программы.

4. Если текущая команда - ".", то остановить работу машины.

5. Если текущая команда - "V j", то поставить метку в текущей позиции и перейти к j-й строке программы.

6. Если текущая команда - "X j", то стереть метку в текущей позиции и перейти к j-й строке программы.

7. Если текущая команда - "← j", то сдвинуться влево на одну позицию и перейти к j-й строке программы.

8. Если текущая команда - "→ j", то сдвинуться вправо на одну позицию и перейти к j-й строке программы.

9. Если текущая команда - "? j1; j2", то если в текущей позиции нет метки, то перейти к j2-й строке программы, иначе перейти к j1-й строке программы.

10. Установить текущую команду на следующую строку программы.

11. Повторять шаги 4-10, пока не будет выполнена команда "!" или не будет достигнут максимальный лимит шагов.

**Classes lead**

Данный код представляет собой два класса. Класс Tape представляет работу с лентой (tape) машины Поста, а класс PostMachine - саму машину Поста.

Методы класса **Tape:**

- init(self): инициализация ленты

- right(self): перемещение каретки вправо

- left(self): перемещение каретки влево

- num(self, a): запись числа в ячейку

- checknum(self): чтение числа из ячейки

Методы класса **PostMachine:**

- init(self): инициализация программы машины Поста

- add\_command(self, command): добавление команды в программу машины

- run(self): выполнение программы машины

- get\_tape(self): вывод всей ленты

Класс PostMachine описывает работу машины Поста, которая состоит из команд перемещения по ленте (вправо/влево), записи чисел в ячейку, проверки числа и завершения программы.

class PostMachine:

    def \_\_init\_\_(self):

        self.program = []  # Программа машины Поста

        self.current\_line = 0  # Текущая строка программы

        self.tape = Tape()

    def add\_command(self, command): #добавляем команду в машину

        self.program.append(command)

    def run(self):

        while self.current\_line < len(self.program): #пока не достигли конца программы

            command = self.program[self.current\_line]

            cmd, \*args = command.split()

            if cmd == ">":

                self.tape.right()

            elif cmd == "<":

                self.tape.left()

            elif cmd == "0":

                self.tape.num(0)

            elif cmd == "1":

                self.tape.num(1)

            elif cmd == "?":

                if self.tape.checknum != 0:

                    self.current\_line = int(args[1]) #- 1

                else:

                    self.current\_line = int(args[0]) #- 1

            elif cmd == ".":

                break

            if cmd != "?":

                if args is not None:

                    self.current\_line = int(args[0])

                else:

                    self.current\_line += 1

    def get\_tape(self): #вывод всей ленты

        t = list(reversed(self.tape.left\_tape))

        return "".join(map(str, t)) + "".join(map(str, self.tape.right\_tape))

class Tape:

    def \_\_init\_\_(self):

        self.left\_tape = [] # Левая часть ленты(она будет начинаться с -1)

        self.right\_tape = [0]  # Правая часть ленты(а она с 0)

        self.pos\_carriage = 0 #позиция каретки

    def right(self):

        self.pos\_carriage += 1

        if self.pos\_carriage > 0:

            if self.pos\_carriage + 1 > len(self.right\_tape): # если ленты не хватает добавляем ячейку

                self.right\_tape.append(0)

    def left(self):

        self.pos\_carriage -= 1

        if self.pos\_carriage == -1:

            if len(self.left\_tape) == 0: # добавляем элемент так как изначальна список пуст

                self.left\_tape.append(0)

        if self.pos\_carriage < -1:

            if abs(self.pos\_carriage) >= len(self.left\_tape): # если ленты не хватает добавляем ячейку

                self.left\_tape.append(0)

    def num (self, a):

        if self.pos\_carriage < 0:

            self.left\_tape[abs(self.pos\_carriage) - 1] = a

        else:

            self.right\_tape[self.pos\_carriage] = a

    def checknum(self):

        if self.pos\_carriage < 0:

            return self.left\_tape[abs(self.pos\_carriage) - 1]

        else:

            return self.right\_tape[self.pos\_carriage]# чтение числа из ячейки

**Test lead**

import unittest

from post import PostMachine

class TestPostMachine(unittest.TestCase):

    def setUp(self):

        self.machine = PostMachine() #Создаём глобальный экземпляр класса Машины Поста

# Проверяется, что программа машины содержит только одну команду ">".

    def test\_add\_command(self):

        self.machine.add\_command(">") # вывается метод add\_command с аргументом ">", который добавляет команду ">" в программу машины.

        self.assertEqual(self.machine.program, [">"])

    def test\_right(self):

        self.machine.tape.right()  #Вызывается метод right у ленты машины, который перемещает каретку на одну позицию вправо.

        self.assertEqual(self.machine.tape.pos\_carriage, 1) #Проверяется, что позиция каретки равна 1.

    def test\_left(self): #Аналогично с тестом test\_right, только в другую сторону. Проверка, что будет -1

        self.machine.tape.left()

        self.assertEqual(self.machine.tape.pos\_carriage, -1)

if \_\_name\_\_ == "\_\_main\_\_":

    unittest.main()

Код представляет собой набор тестов для класса PostMachine. В каждом тесте создается экземпляр класса PostMachine, затем вызываются различные методы этого класса, и их результаты сравниваются с ожидаемыми значениями с помощью методов из библиотеки unittest.

Тест test\_add\_command: Проверяет, что метод add\_command() класса PostMachine корректно добавляет команду (в данном случае ">", предположительно команда для машины Поста) в программу машины.

Тест test\_right: Проверяет, что метод right() объекта tape (ленты) класса PostMachine корректно перемещает каретку на одну позицию вправо.

Тест test\_left: Аналогично test\_right, только проверяет перемещение каретки на одну позицию влево.

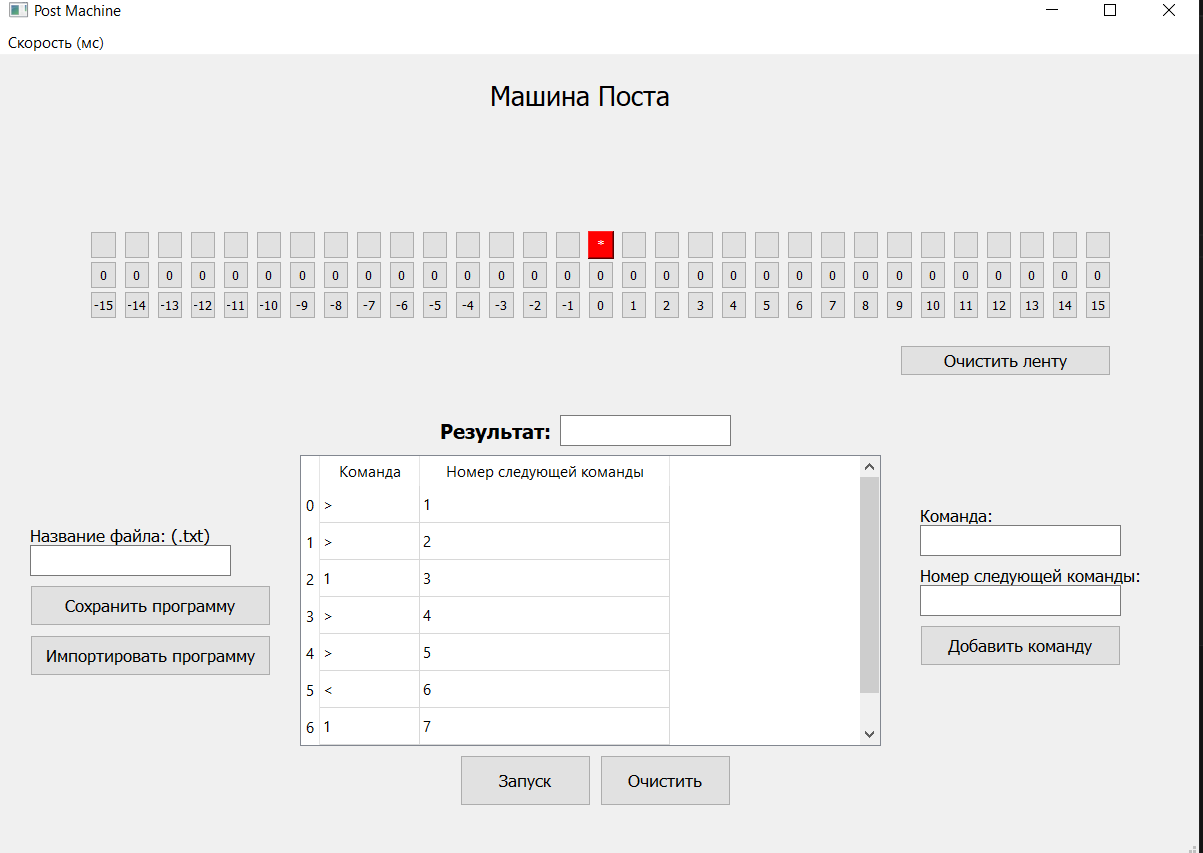
Тест test\_num: Убеждается, что метод num() объекта tape класса PostMachine записывает указанное значение (в данном случае 0) на текущую позицию ленты.

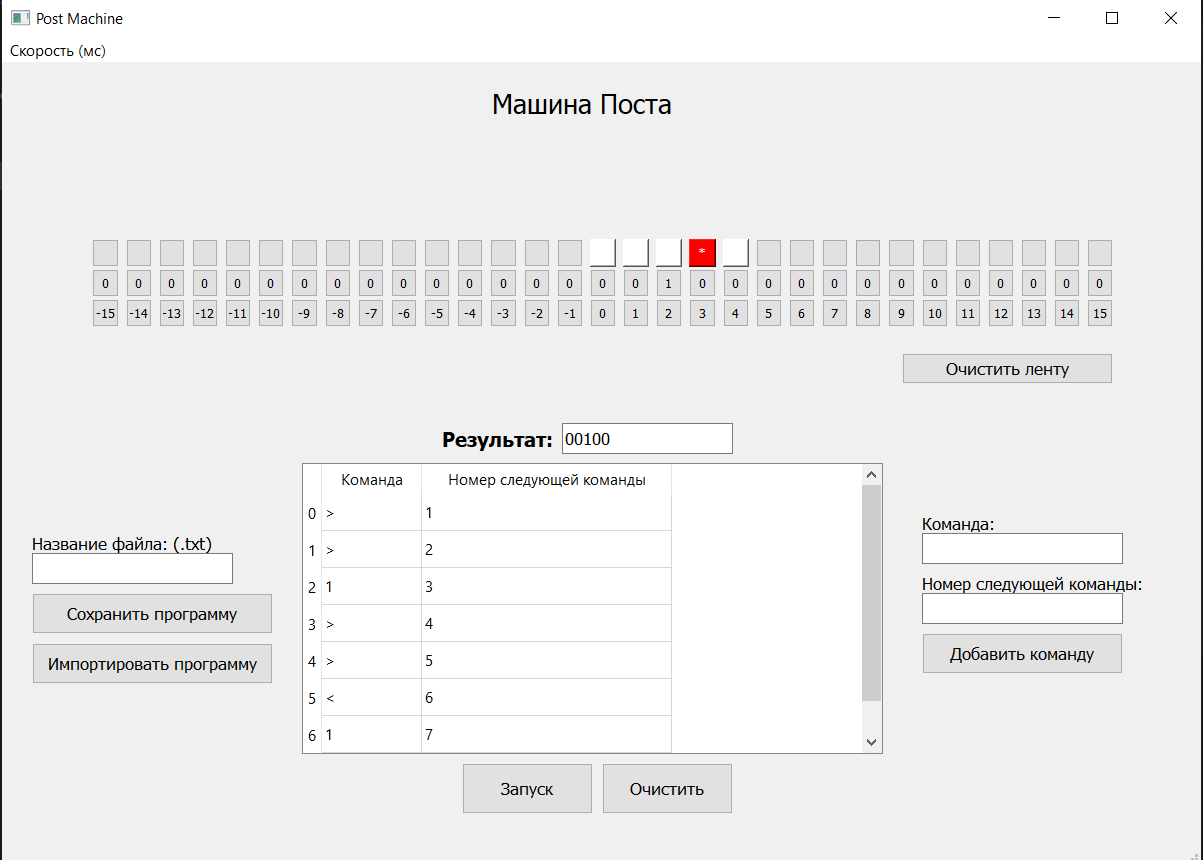
Тест test\_checknum: Устанавливает значение 1 на текущей позиции ленты с помощью метода num(), а затем использует метод checknum() объекта tape, чтобы проверить, что значение на текущей позиции соответствует ожидаемому (равно 1).

Затем, если код запущен как основная программа (if name == "main"), выполняется запуск всех тестов с помощью unittest.main().

**GUI lead**

Интерфейс проекта. Лента. Машина поста.





Графический интерфейс написан на PyQt5 с использование редактора QtDesigner. Были использованы следующие библиотеки: PyQt, post (класс логики), tkinter.

Функционал приложения (графического интерфейса):

* Запись команд;
* Проверка корректности введённых команд;
* Просмотр команд;
* Редактирование команд;
* Сохранение записанных программ в текстовый файл;
* Проверка корректности названия сохраняемой программы;
* Загрузка программ из текстового файла;
* Запуск программы;
* Вывод результата;
* Очистка программы;
* Анимированное выполнение программы;
* Очистка ленты;
* Выбор скорости анимации выполнения программы.

**Docs lead**

**Пользовательская документация к программе.**

Программа “Post Machine” позволяет решать различные задачи с помощью инструкций состоящих из команд и каретки, которая движется по бесконечной ленте. В составлении инструкций и заключается основная работа пользователя.

Инструкции могут состоять из следующих команд, которые воспринимает программа:

"." - остановить работу машины.

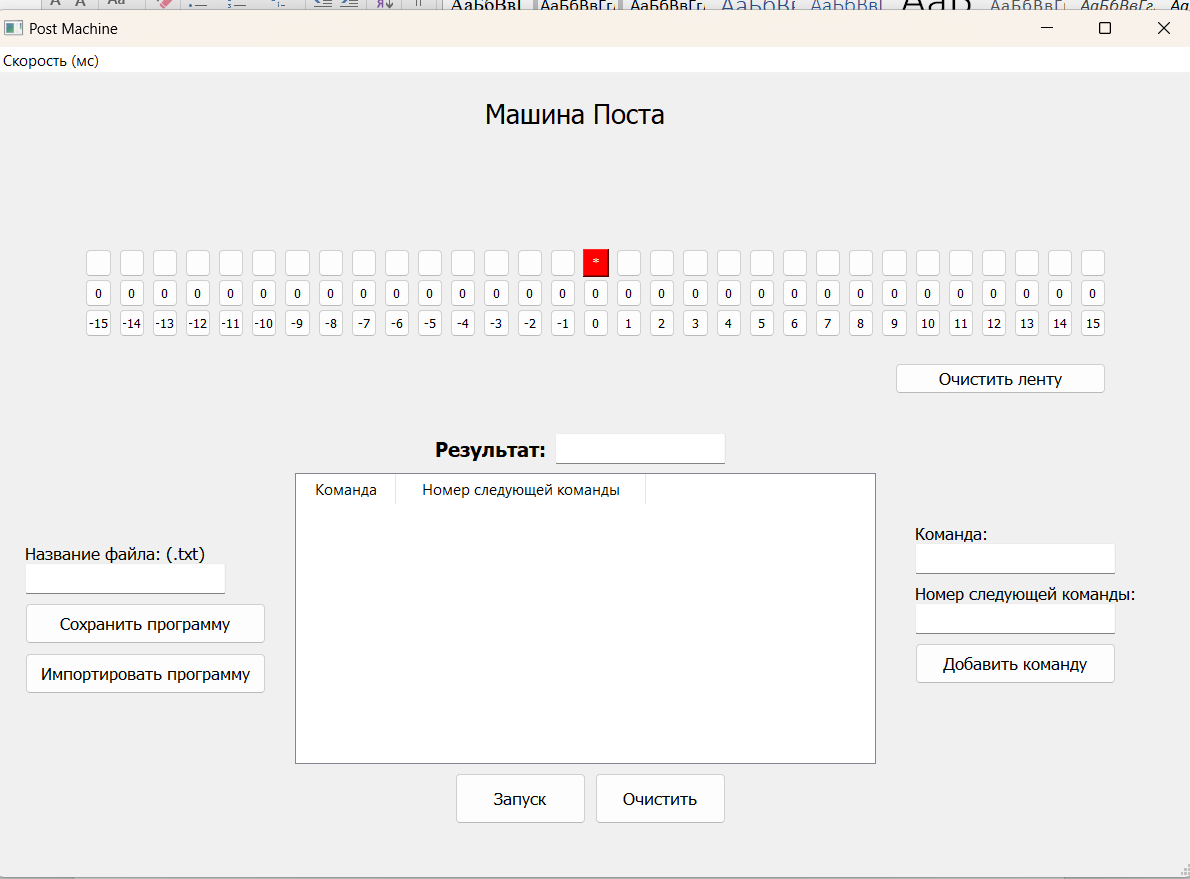
"V j" - поставить метку в текущей позиции и перейти к j-й строке программы.

"X j" - стереть метку в текущей позиции и перейти к j-й строке программы.

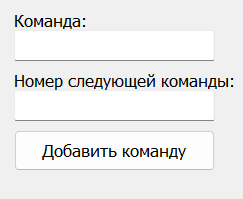
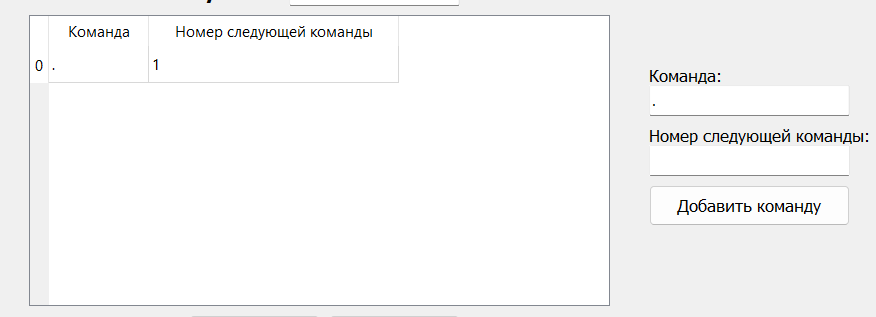
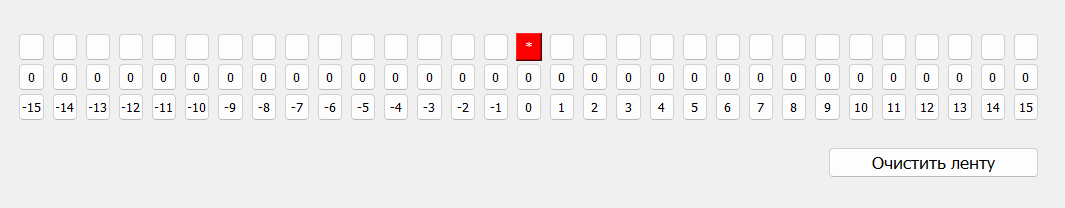
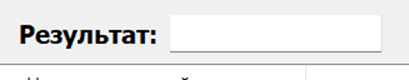
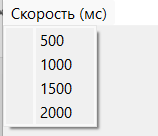
"← j" - сдвинуться влево на одну позицию и перейти к j-й строке программы.

"→ j" - сдвинуться вправо на одну позицию и перейти к j-й строке программы.

"? j1; j2" - если в текущей позиции нет метки, то перейти к j2-й строке программы, иначе перейти к j1-й строке программы.

После запуска программы нас встречает элегантный интерфейс:

Основные функции программы:

1. Текстовые поля “Команда” и “Номер следующей команды” а так же кнопка “Добавить команду” позволяют добавить соответствующую команду в список команд.  
     
   Для того чтобы добавить команду требуется записать в поле “Команда” одну из 6 команд на выбор без кавычек и нажать кнопку “Добавить команду”. Если в это поле написать символ отличный от команды или оставить пустым, то автоматически будет выбрана команда «.». В поле “Номер следующей команды” следует добавить номер той команды, которая будет выполняться следующей. Если этого не сделать то команда будет ссылаться на следующую команду которой не существует.
2. Добавляемые команды будут отображаться в большом белом поле слева:  
   Команды будут добавляться сверху-вниз. Слева от каждой команды находится её номер. Именно с помощью него можно переходить от одной команды на другую.  
   Сами команды в списке можно изменять. Для этого просто нажмите два раза на ячейку и измените её значение на то которое вам нужно.  
   Чтобы очистить список команд, достаточно нажать на кнопку “Очистить” которая находится ниже.
3. Для запуска программы требуется нажать на кнопку “Запуск”. После этого инструкция начнёт выполняться, а её прогресс будет отображаться на анимируемых каретке и ленте:  
   Верхние ячейки являются отображением положения каретки. Средние ячейки являются значением данной ячейки ленты. Нижние ячейки хранят в себе номер ячейки. С помощью кнопки “Очистить ленту” можно вернуть каретку в исходное положение и очистить значения ячеек.
4. После завершения работы вашей инструкции, результат будет выведен в поле “Результат”:  
   
5. При выполнении программы, между каждой командой есть задержка, которая нужна для наглядности выполнения. Время этой задержки можно выставить при помощи кнопки “Скорость (мс)” которая находится в левом верхнем углу программы:  
   
6. Также программа предоставляет возможность сохранить и загрузить инструкцию в виде текстового файла формата “.txt”. Для сохранения инструкции требуется ввести название файла вида “пример.txt” и нажать на кнопку “Сохранить программу”.   
   Для загрузки программы требуется нажать на кнопку “Импортировать программу”. После этого появится диалоговое окно проводника, в котором вам нужно выбрать текстовый файл который вы сохраняли ранее.

Ссылка на проект в GitHub: <https://github.com/DimaChesnokov/PostMachine>

Список литературы:

<https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%B0_%D0%9F%D0%BE%D1%81%D1%82%D0%B0>  
  
<https://pythonworld.ru/moduli/modul-unittest.html>