Высшее государственное образовательное учреждение

«Санкт-Петербургский Государственный Морской Технический Университет»

Лабораторная работа №2 по программированию

9 вариант

Работу выполнил

Студент 1 курса факультета

цифровых промышленных технологий

группы 20121 Красноперов Данил

Проверил

Поделенюк Павел Петрович

Санкт-Петербург

2023

# 1.Цели работы

При работе над данной лабораторной работой будут затронуты следующие вопросы:

1. Работа с текстовыми файлами,
2. Работа с локальными и глобальными переменными,
3. Работа с несколькими функциями, а не только с main,
4. Нисходящий метод разработки программ,
5. Анализ ходов.

# 2.Ход работы

1. Создадим основную функцию main()
2. Создадим файл Input.txt, запишем в него вводные данные и считаем их(Считываются значения N, L, K из первой строки файла.)
3. Создадим пустое множество solutions, в которое будут добавляться все возможные решения.
4. Далее считываются координаты уже стоящих фигур из файла и добавляются в список s.
5. Выводится информация о размере доски, количестве нужных фигур и уже стоящих фигурах.
6. Вызывается рекурсия recursion() для поиска всех возможных решений.
7. Функция alfil\_steps(x, y) принимает координаты фигуры и высчитывает все возможные шаги фигуры Алфил. Возвращает множество возможных шагов.
8. Функция matrix\_board(N) создает и возвращает матрицу размером N x N, заполненную нулями.
9. Функция mark\_solutions(x, y, matrix) принимает координаты фигуры и матрицу доски. Отмечает позицию фигуры на доске и отмечает все возможные ходы фигуры вокруг этой позиции.
10. Функция create\_matrix(matrix, posing) создает и возвращает новую матрицу, копируя исходную матрицу и отмечая на ней позиции фигур.
11. Рекурсия recursion(N, L, solutions, current\_solution, current):

* Если текущее количество фигур равно нужному количеству L, то текущее решение добавляется в множество solutions.
* Если в множестве solutions есть решения, то выводится первое найденное решение на экран, с помощью функции create\_matrix() и output\_matrix().
* Записываются все решения в файл "output.txt".
* В цикле по каждой клетке доски проверяется, можно ли поставить фигуру в данную клетку.
* Если фигуры еще нет в текущем решении и нет пересечения шагов с уже стоящими фигурами, то создается новое решение с добавлением этой фигуры и вызывается рекурсивная функция recursion() с новым решением и увеличенным текущим количеством фигур на 1.

1. Если есть решения, то выводится их количество и записываются в файл "output.txt".
2. Вызываем функцию main()
3. Измерим время выполнения программы используя библиотеку time

# 3.Результат работы

Наш файл Input.txt выглядит следующим образом:

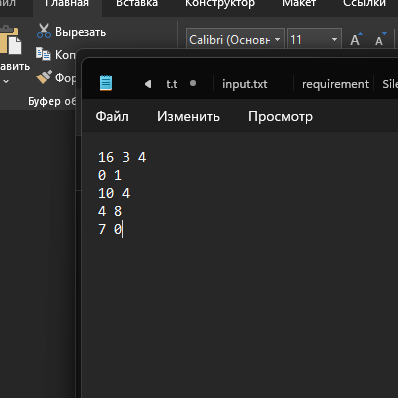


Рисунок 1 Файл Input.txt

После его выполнения в папке с программой появляется файл Output.txt, содержащий наше решение, а на консоль выводится следующее:

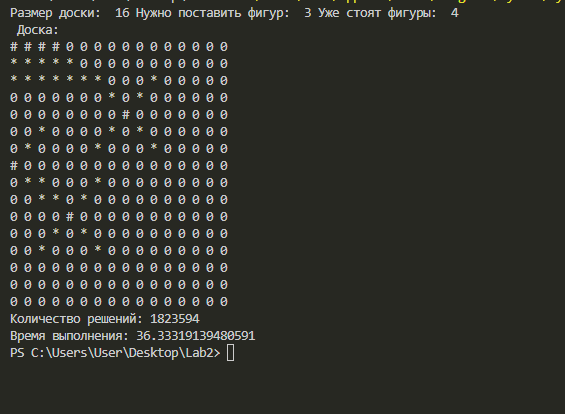


Рисунок 2 Консоль после выполнения программы

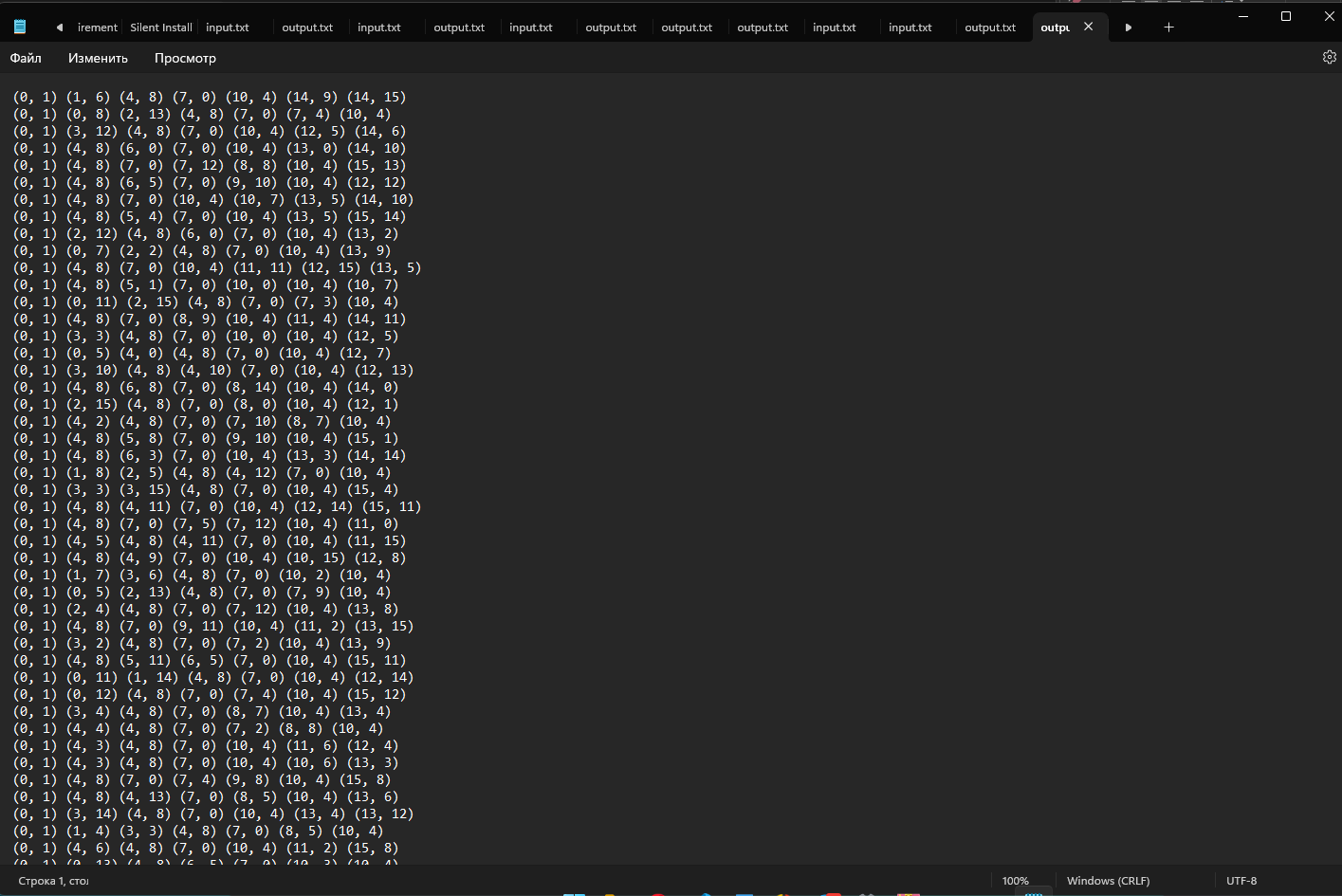
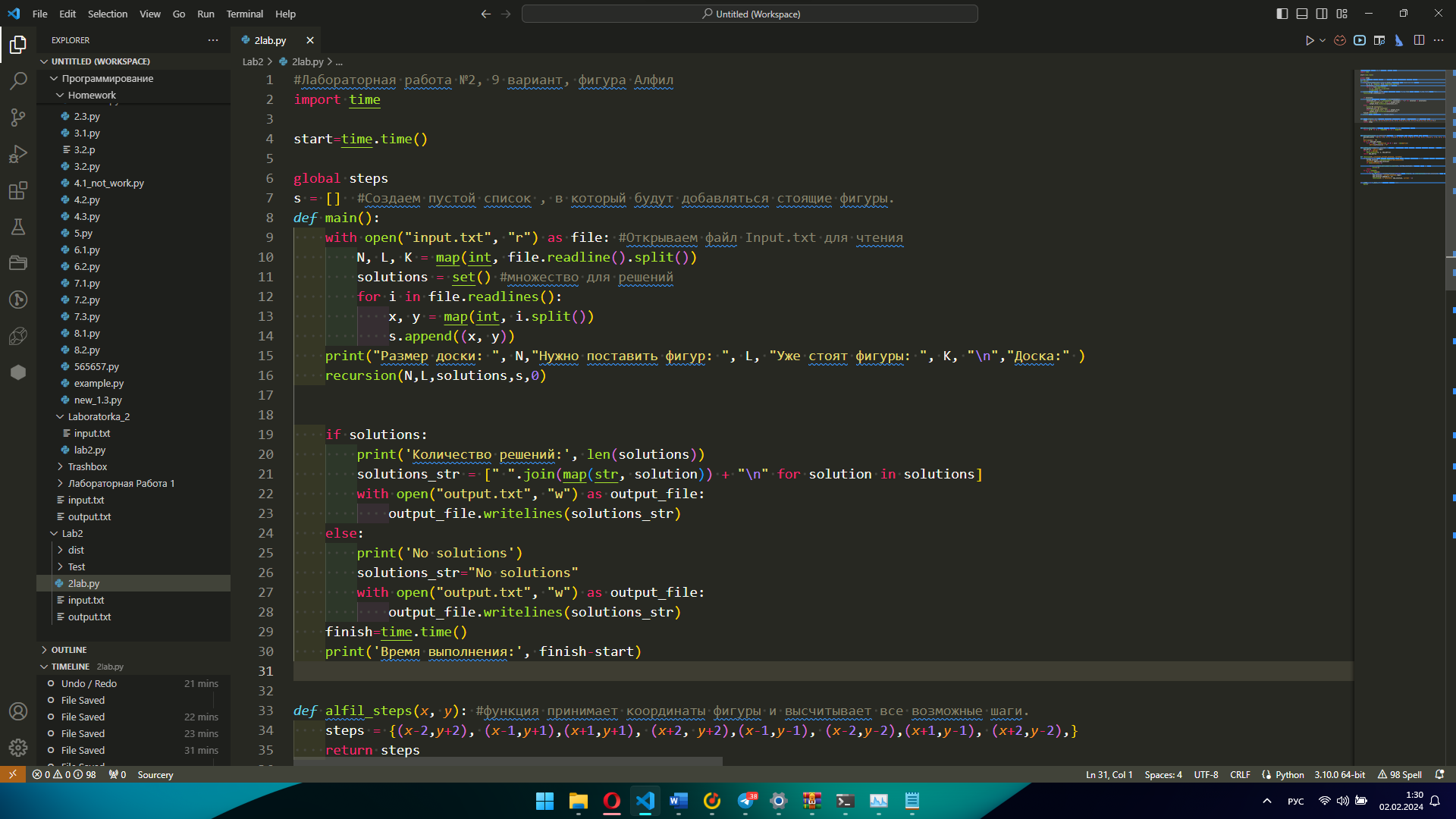


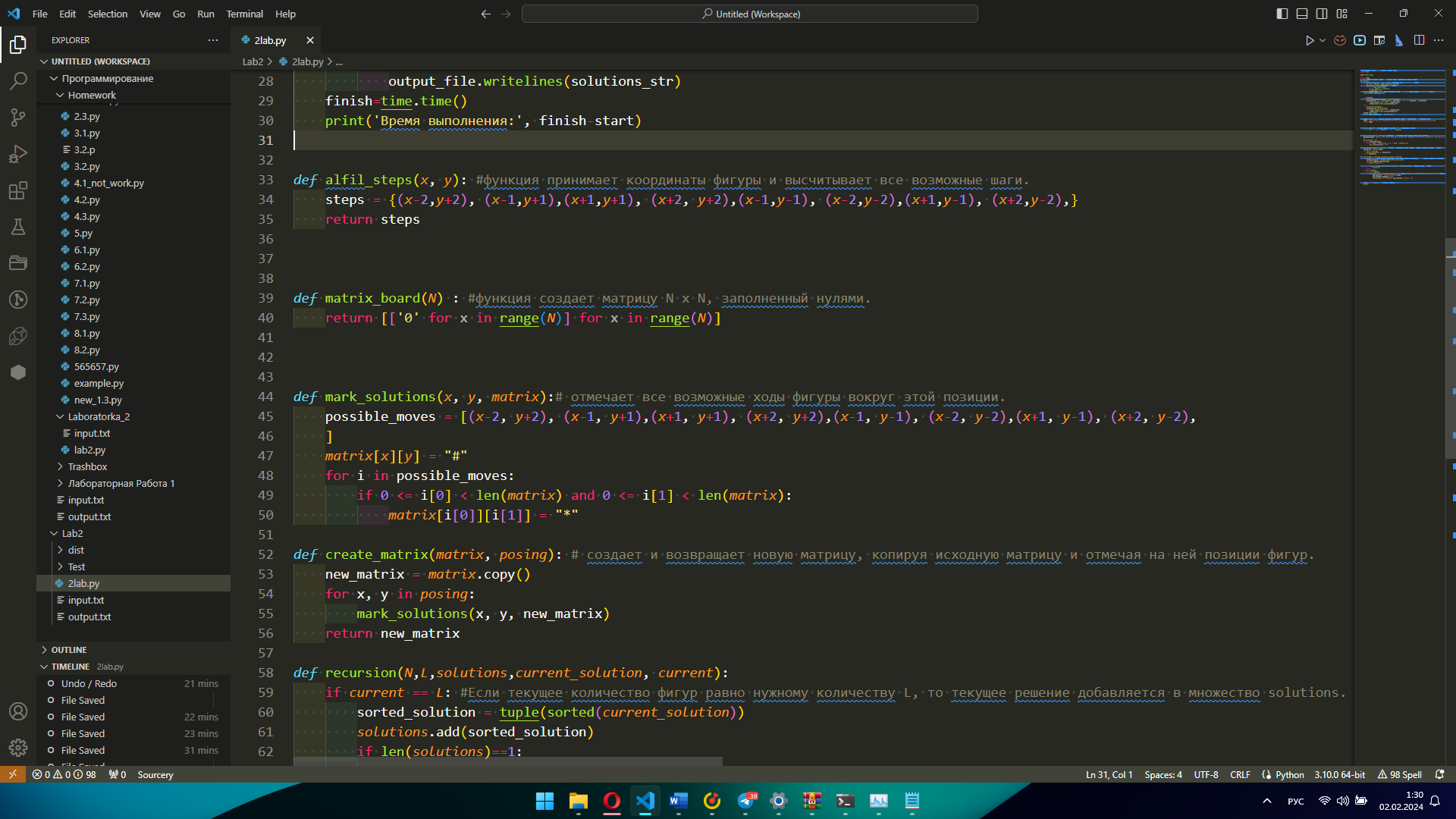
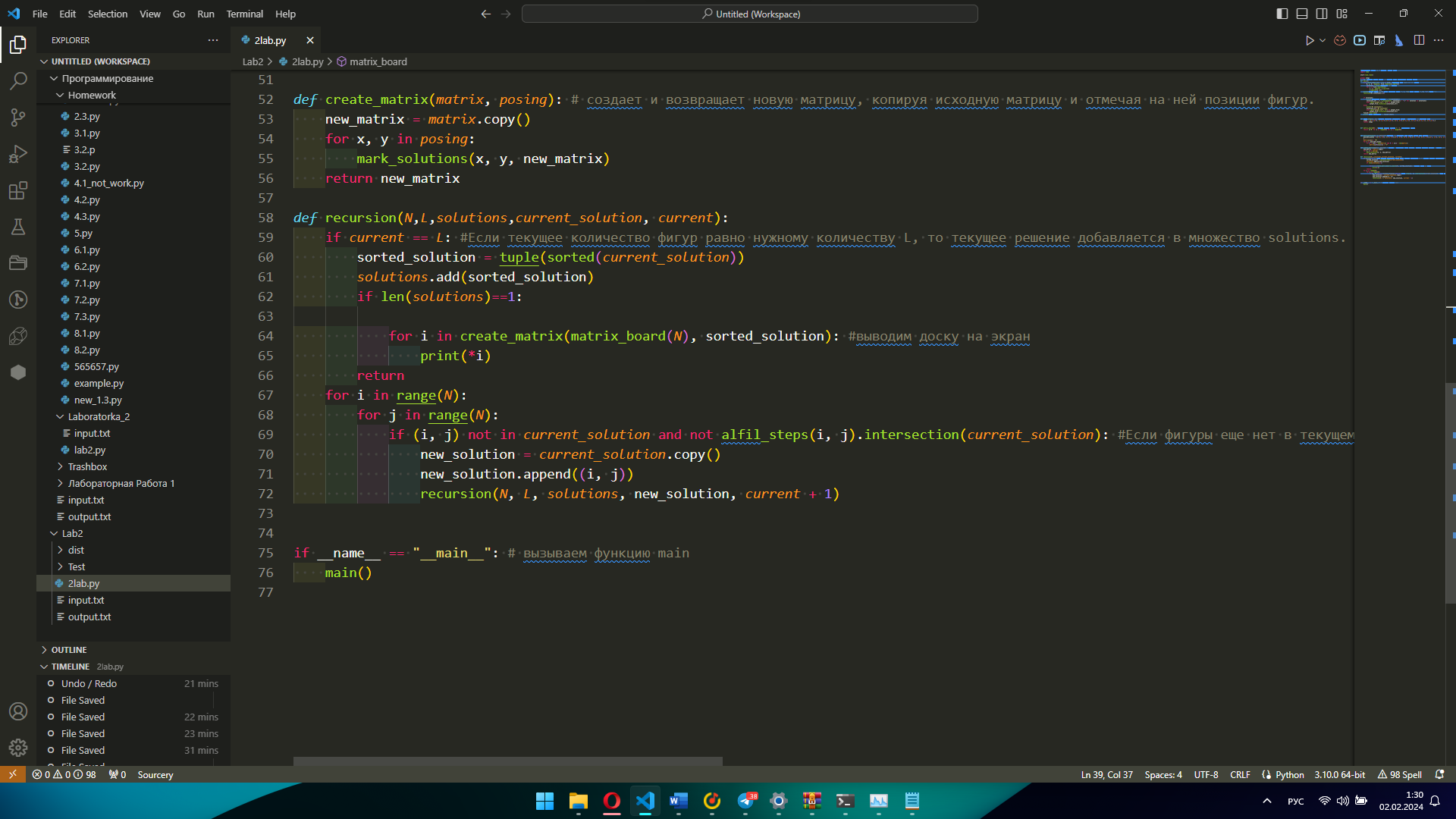
Рисунок 2 Файл Output.txt

# 4.Вывод

При работе над данной лабораторной работой я получил навыки работы с текстовыми файлами, с локальными и глобальными переменными, с несколькими функциями, а не только с main, познакомился с нисходящим методом разработки программ и научился анализировать ходы.

# 5.Листинг кода



#Лабораторная работа №2, 9 вариант, фигура Алфил

import time

start=time.time()

global steps

s = []  #Создаем пустой список , в который будут добавляться стоящие фигуры.

*def* main():

    with open("input.txt", "r") as file: #Открываем файл Input.txt для чтения

        N, L, K = map(int, file.readline().split())

        solutions = set() #множество для решений

        for i in file.readlines():

            x, y = map(int, i.split())

            s.append((x, y))

    print("Размер доски: ", N,"Нужно поставить фигур: ", L, "Уже стоят фигуры: ", K, "\n","Доска:" )

    recursion(N,L,solutions,s,0)

    if solutions:

        print('Количество решений:', len(solutions))

        solutions\_str = [" ".join(map(str, solution)) + "\n" for solution in solutions]

        with open("output.txt", "w") as output\_file:

            output\_file.writelines(solutions\_str)

    else:

        print('No solutions')

        solutions\_str="No solutions"

        with open("output.txt", "w") as output\_file:

            output\_file.writelines(solutions\_str)

    finish=time.time()

    print('Время выполнения:', finish-start)

*def* alfil\_steps(*x*, *y*): #функция принимает координаты фигуры и высчитывает все возможные шаги.

    steps = {(*x*-2,*y*+2), (*x*-1,*y*+1),(*x*+1,*y*+1), (*x*+2, *y*+2),(*x*-1,*y*-1), (*x*-2,*y*-2),(*x*+1,*y*-1), (*x*+2,*y*-2),}

    return steps

*def* matrix\_board(*N*) : #функция создает матрицу N х N, заполненный нулями.

    return [['0' for x in range(*N*)] for x in range(*N*)]

*def* mark\_solutions(*x*, *y*, *matrix*):# отмечает все возможные ходы фигуры вокруг этой позиции.

    possible\_moves = [(*x*-2, *y*+2), (*x*-1, *y*+1),(*x*+1, *y*+1), (*x*+2, *y*+2),(*x*-1, *y*-1), (*x*-2, *y*-2),(*x*+1, *y*-1), (*x*+2, *y*-2),

    ]

*matrix*[*x*][*y*] = "#"

    for i in possible\_moves:

        if 0 <= i[0] < len(*matrix*) and 0 <= i[1] < len(*matrix*):

*matrix*[i[0]][i[1]] = "\*"

*def* create\_matrix(*matrix*, *posing*): # cоздает и возвращает новую матрицу, копируя исходную матрицу и отмечая на ней позиции фигур.

    new\_matrix = *matrix*.copy()

    for x, y in *posing*:

        mark\_solutions(x, y, new\_matrix)

    return new\_matrix

*def* recursion(*N*,*L*,*solutions*,*current\_solution*, *current*):

    if *current* == *L*: #Если текущее количество фигур равно нужному количеству L, то текущее решение добавляется в множество solutions.

        sorted\_solution = tuple(sorted(*current\_solution*))

*solutions*.add(sorted\_solution)

        if len(*solutions*)==1:

            for i in create\_matrix(matrix\_board(*N*), sorted\_solution): #выводим доску на экран

                print(\*i)

        return

    for i in range(*N*):

        for j in range(*N*):

            if (i, j) not in *current\_solution* and not alfil\_steps(i, j).intersection(*current\_solution*): #Если фигуры еще нет в текущем решении и нет пересечения шагов с уже стоящими фигурами, то создается новое решение с добавлением этой фигуры и вызывается рекурсия с новым решением и увеличенным текущим количеством фигур на 1.

                new\_solution = *current\_solution*.copy()

                new\_solution.append((i, j))

                recursion(*N*, *L*, *solutions*, new\_solution, *current* + 1)

if \_\_name\_\_ == "\_\_main\_\_": # вызываем функцию main

    main()