```python
MATRIX_FILE = "matrixFile.txt"
"""
Name: Samuel Haberkorn
Class: CS 2300
Date: 09/12/2020
Assignment: Project 1
"""

def main():
    with open(MATRIX_FILE) as file:
        matrix_a = read_matrix(file)
        matrix_b = read_matrix(file)

    # STEP 1
    write_matrix(matrix_a[1], matrix_a[0], open("FirstMatrix", "w+"))
    write_matrix(matrix_b[1], matrix_b[0], open("SecondMatrix", "w+"))

    # STEP 2
    print(f"Matrix {matrix_a[0]}: ")
    print_matrix(matrix_a[1])
    print("\n")
    print(f"Matrix {matrix_b[0]}: ")
    print_matrix(matrix_b[1])
    print("\n")

    # STEP 3
    print(f"1.5{matrix_b[0]} - 2.5{matrix_a[0]}: ")

    final_matrix = subtract_matrices(
        multiply_matrix(matrix_b[1], 1.5),
        multiply_matrix(matrix_a[1], 2.5)
    )
    print_matrix(final_matrix)
    write_matrix(final_matrix, "m", open("calcMatrix", "w+"))
    print("\n")

    # step 4
    print(f"Transposed Matrix {matrix_b[0]}:")
    transposed_matrix = transpose_matrix(matrix_b[1])
    print_matrix(transposed_matrix)
    write_matrix(transposed_matrix, "t", open("
transposedMatrix", "w+"))


def transpose_matrix(matrix):
    """
    Transposes a matrix from (x, y) to (y, x)
    :param matrix: Matrix
    :return: Matrix
    """
    new_matrix = []
    for i in range(len(matrix[0])):
        new_matrix.append(([None] * len(matrix)).copy
())

    for row_num, row in enumerate(matrix):
        for col_num, col in enumerate(row):
            new_matrix[col_num][row_num] = col

    return new_matrix


def subtract_matrices(matrix_a, matrix_b):
    """
    Subtracts first matrix from second matrix
    :param matrix_a: Matrix (2D-List)
    :param matrix_b: Matrix (2D-List)
    :return: Matrix (2D-List)
    """
    new_matrix = []
    for rowA, rowB in zip(matrix_a, matrix_b):
        temp = []
        for colA, colB in zip(rowA, rowB):
            temp.append(colA - colB)
        new_matrix.append(temp)
    return new_matrix


def multiply_matrix(matrix, multiplier):
    """
    Multiplies a matrix by a constant value
    :param matrix: Matrix to multiply
    :param multiplier: multiplier
    :return: a new matrix (2D-List)
    """
    new_matrix = []
    for row in matrix:
        temp = []
        for col in row:
            temp.append(col * multiplier)
        new_matrix.append(temp)
    return new_matrix


def print_matrix(matrix):
    """
    Prints matrix to stdout with 5 characters per cell
    :param matrix: matrix to write
    :return: None
    """
    for row in matrix:
        print(" ".join(map(lambda n:  f"{n:>5}", row)))


def write_matrix(matrix, character, file):
    """
    Writes matrix to file
    :param matrix: Matrix to write
    :param character: Character name of matrix
    :param file: file to write to (must be open)
    :return: None
    """
    file.write(f"{character} {len(matrix)} {len(matrix[0])} ")
    for row in matrix:
        file.write(" ".join(map(str, row)))
        file.write(" ")


def read_matrix(file):
    """
    Reads a matrix from a file. Returns a tuple with the character name and the actual matrix
    """
    matrix = []

    char = get_next_char(file)
    rows = int(get_next_char(file))
    cols = int(get_next_char(file))

    for row in range(rows):
        temp = []
        for col in range(cols):
            temp.append(int(get_next_char(file)))
        matrix.append(temp)
    return char, matrix


'''
Recursive function to read in each set of characters.
If no character or whitespace is read, return the read characters or keep reading (if no characters are read)
'''


def get_next_char(file, prev_chars=""):
    """
    Recursive function to read in each set of characters.
    If no character or whitespace is read, return the read characters or keep reading (if no characters are read)
    :param file: file
    :param prev_chars: str
    :return: str
    """
    char = file.read(1)

    if not len(char) == 0 and char not in [None, " ", "\n"]:
        return get_next_char(file, prev_chars+char)
    return prev_chars if prev_chars != "" else get_next_char(file)


if __name__ == '__main__':
    main()

```