Linear Algebra (0031)
Project 0

Yulwon Rhee (202211342)

Department of Computer Science and Engineering, Konkuk University

1. (a) `transposeMatrix(A, m, n)`: transpose the $m \times n$ matrix A and return the result

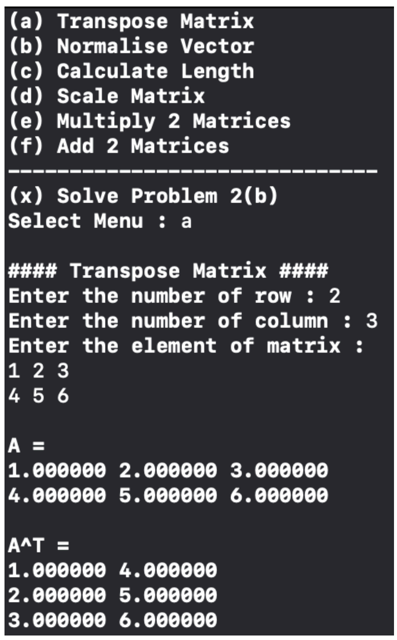
Let $A^T = B$. Since $B_{ij} = A_{ji}$, the code below returns transpose of matrix A.

```
double** transposeMatrix(double **A, int m, int n) {
    double** B = allocateMemory(n, m);

    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
            B[j][i] = A[i][j];

    return B;
}
```

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

$$\therefore A^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

(a) Equation



(b) Result Image

Fig. 1: `transposeMatrix()`

(b) `normalizeVector(v, n)`: normalise the n-dimensional vector $\mathbf{v}$ and return the result

Since normalised vector is calculated by dividing all entries by its length, the code below returns the normalised vector $\mathbf{v}$.

```
double** normalizeVector(double** v, int n) {
    double** w;
    double len = calculateLength(v, n);

    w = allocateMemory(n, 1);
    for (int i = 0; i < n; i++)
        w[i][0] = v[i][0] / len;

    return w;
}
```

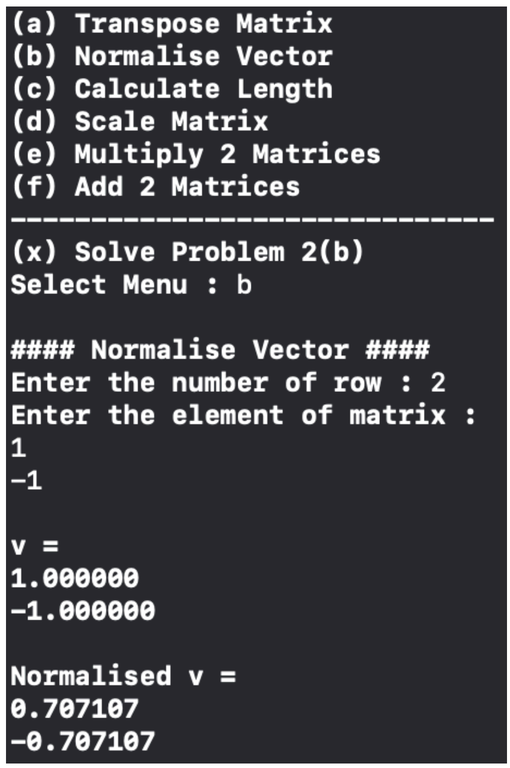
$$\mathbf{v} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

$$||\mathbf{v}|| = \sqrt{(1)^2 + (-1)^2}$$

$$\therefore \hat{\mathbf{v}} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix}$$

$$\hat{\mathbf{v}} \approx \begin{bmatrix} 0.707107 \\ -0.707107 \end{bmatrix}$$

(a) Equation



(b) Result Image

Fig. 2: `normalizeVector()`

(c) `calculateLength(v, n)`: calculate the length of the n-dimensional vector **v** and return the result

Since length of vector is calculated by square root of the sum of the square of all entries, the code below returns the length of vector **v**.

```
double calculateLength(double** v, int n) {
    double len = 0.0;

    for (int i = 0; i < n; i++) {
        len += v[i][0] * v[i][0];
    }
    len = sqrt(len);

    return len;
}
```

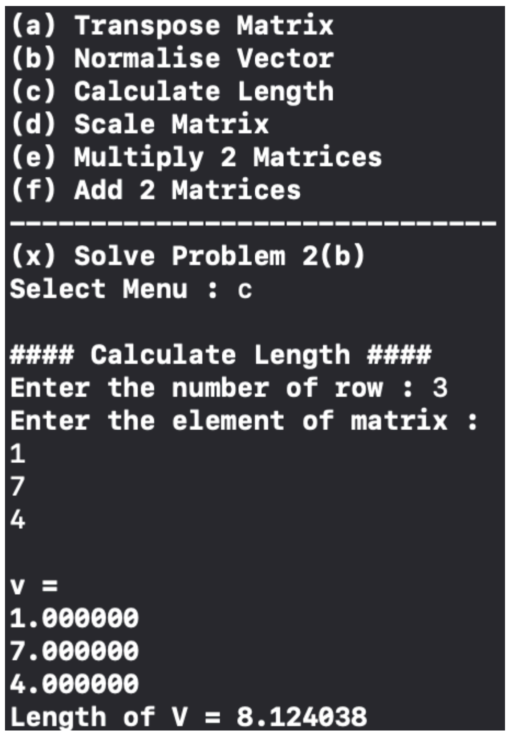
$$\mathbf{v} = \begin{bmatrix} 1 \\ 7 \\ 4 \end{bmatrix}$$

$$||\mathbf{v}|| = \sqrt{1^2 + 7^2 + 4^2}$$

$$||\mathbf{v}|| = \sqrt{66}$$

$$||\mathbf{v}|| \approx 8.124038$$

(a) Equation



(b) Result Image

Fig. 3: `calculateLength()`

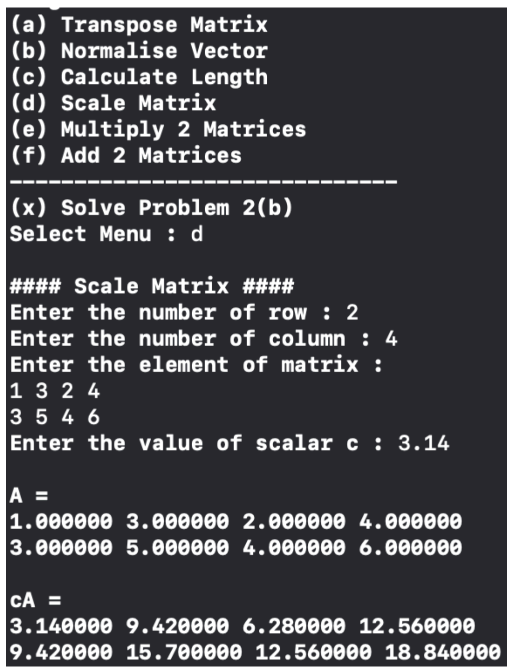
(d) `scaleMatrix(A, m, n, c)`: scale the $m \times n$ matrix A with scalar c

The code below returns the matrix A scaled by c by multiplying all entries in A by c.

```c
double** scaleMatrix(double** A, int m, int n, double c) {
    double** cA = allocateMemory(m, n);
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            cA[i][j] = c * A[i][j];
        }
    }

    return cA;
}
```

$$3.14 \begin{bmatrix} 1 & 3 & 2 & 4 \\ 3 & 5 & 4 & 5 \end{bmatrix} = \begin{bmatrix} 3.14 & 9.42 & 6.28 & 12.56 \\ 9.42 & 15.7 & 12.56 & 18.84 \end{bmatrix}$$

(a) Equation



(b) Result Image

Fig. 4: `scaleMatrix()`

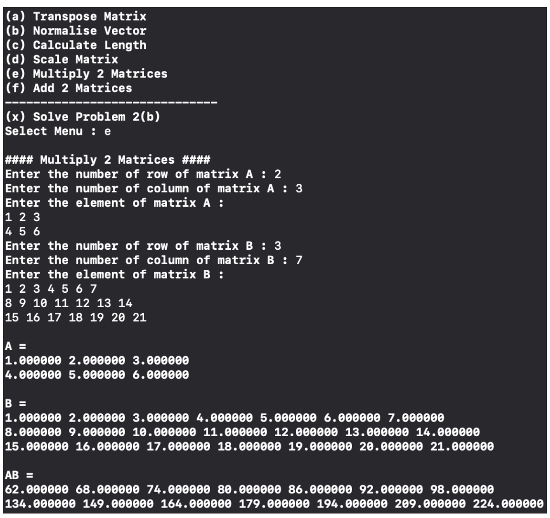
(e) `multiplyTwoMatrices(A, m, n, B, l, k)`: for $m \times n$ matrix A and $l \times k$ matrix B, calculate and return AB. Return `null` if multiplication is impossible.

The code below returns the multiplication between matrix A and B or `NULL` if multiplication is impossible.

```
double** multiplyTwoMatrices(double** A, int m, int n, double** B, int p, int q) {
    if (n != p) return NULL;

    double** AB = allocateMemory(m, n);

    for (int i = 0; i < m; i++) {
        for (int j = 0; j < q; j++) {
            AB[i][j] = 0;
            for (int k = 0; k < p; k++) {
                AB[i][j] += A[i][k] * B[k][j];
            }
        }
    }

    return AB;
}
```

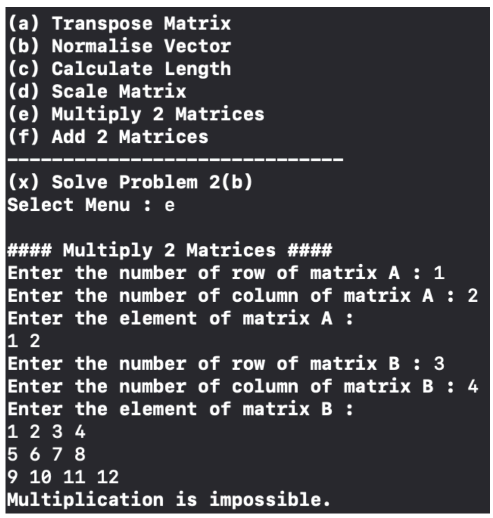
$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 & 12 & 13 & 14 \\ 15 & 16 & 17 & 18 & 19 & 20 & 21 \end{bmatrix} = \begin{bmatrix} 62 & 68 & 74 & 80 & 86 & 92 & 98 \\ 134 & 149 & 164 & 179 & 194 & 209 & 224 \end{bmatrix}$$

(a) Equation



```
(a) Transpose Matrix
(b) Normalise Vector
(c) Calculate Length
(d) Scale Matrix
(e) Multiply 2 Matrices
(f) Add 2 Matrices
------------------------------
(x) Solve Problem 2(b)
Select Menu : e

#### Multiply 2 Matrices ####
Enter the number of row of matrix A : 2
Enter the number of column of matrix A : 3
Enter the element of matrix A :
1 2 3
4 5 6
Enter the number of row of matrix B : 3
Enter the number of column of matrix B : 7
Enter the element of matrix B :
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21

A =
1.000000 2.000000 3.000000
4.000000 5.000000 6.000000

B =
1.000000 2.000000 3.000000 4.000000 5.000000 6.000000 7.000000
8.000000 9.000000 10.000000 11.000000 12.000000 13.000000 14.000000
15.000000 16.000000 17.000000 18.000000 19.000000 20.000000 21.000000

AB =
62.000000 68.000000 74.000000 80.000000 86.000000 92.000000 98.000000
134.000000 149.000000 164.000000 179.000000 194.000000 209.000000 224.000000
```

(b) Result Image



```
(a) Transpose Matrix
(b) Normalise Vector
(c) Calculate Length
(d) Scale Matrix
(e) Multiply 2 Matrices
(f) Add 2 Matrices
------------------------------
(x) Solve Problem 2(b)
Select Menu : e

#### Multiply 2 Matrices ####
Enter the number of row of matrix A : 1
Enter the number of column of matrix A : 2
Enter the element of matrix A :
1 2
Enter the number of row of matrix B : 3
Enter the number of column of matrix B : 4
Enter the element of matrix B :
1 2 3 4
5 6 7 8
9 10 11 12
Multiplication is impossible.
```

(c) Result Image When Multiplication is Impossible

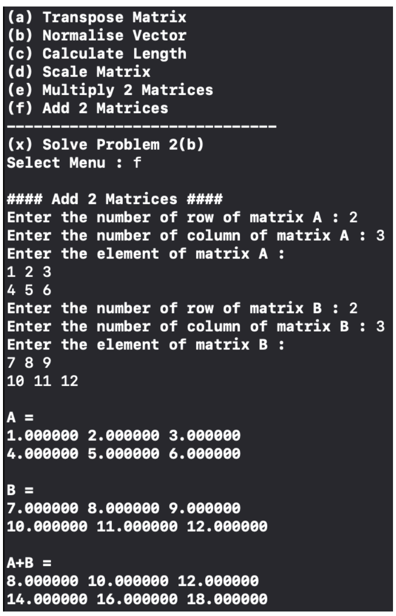
Fig. 5: `multiplyTwoMatrices()`

(f) `addTwoMatrices(A, m, n, B, l, k)`: for $m \times n$ matrix A and $l \times k$ matrix B, calculate and return $A+B$. Return `null` if addition is impossible.
The code below returns the addition between matrix A and B or `NULL` if addition is impossible.

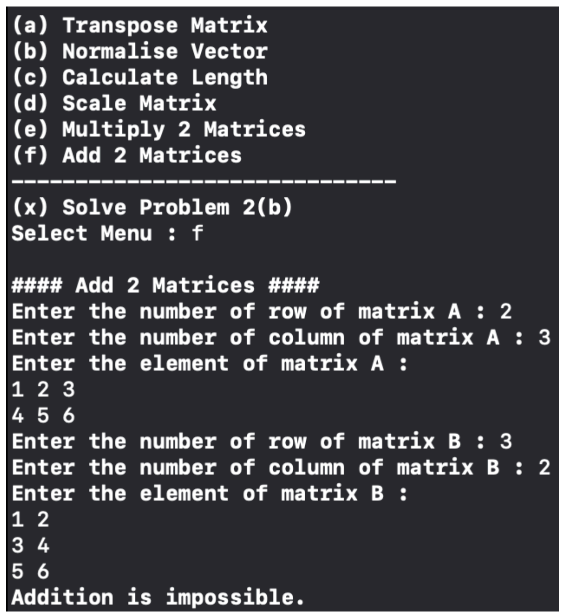
```c
double** addTwoMatrices(double** A, int m, int n, double** B, int l, int k) {
    if (m != l || n != k) return NULL;

    double** C = allocateMemory(m, n);
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            C[i][j] = A[i][j] + B[i][j];
        }
    }

    return C;
}
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix} = \begin{bmatrix} 8 & 10 & 12 \\ 14 & 16 & 18 \end{bmatrix}$$

(a) Equation



(b) Result Image



(c) Result Image When Addition is Impossible

Fig. 6: `addTwoMatrices()`

2. (a) Test the correctness of each of the function you wrote in 1.
Already done in above.

(b) For given $n \times n$ matrices A and $\tilde{H}$, normalize each column of $\tilde{H}$ (let H be this normalized matrix). Then, calculate $B = H^T A^H$, and then, $C = HBH^T$.

```
void problem2b() {
    double a[2][2] = {
        {1, 2},
        {3, 4}
    };

    double tildeH[2][2] = {
        {1, 1},
        {1, -1}
    };

    double** A = allocateMemory(2, 2);
    for (int i = 0; i < 2; i++)
        for (int j = 0; j < 2; j++)
            A[i][j] = (double) a[i][j];
    printMatrix(A,2,2,"A");

    double** TildeH = allocateMemory(2, 2);
    for (int i = 0; i < 2; i++)
        for (int j = 0; j < 2; j++)
            TildeH[i][j] = (double) tildeH[i][j];
    printMatrix(TildeH,2,2,"Tilde H");

    double** H = normalizeMatrix(TildeH, 2, 2);
    printMatrix(H, 2, 2, "H");

    double** HT = transposeMatrix(H, 2, 2);

    double** B = multiplyTwoMatrices(HT, 2, 2, A, 2, 2);
    B = multiplyTwoMatrices(B, 2, 2, H, 2, 2);
    printMatrix(B, 2, 2, "B");

    double** C = multiplyTwoMatrices(H, 2, 2, B, 2, 2);
    C = multiplyTwoMatrices(C, 2, 2, HT, 2, 2);
    printMatrix(C, 2, 2, "C");

    releaseMemory(A, 2);
    releaseMemory(TildeH, 2);
    releaseMemory(H, 2);
```

```
    releaseMemory(HT, 2);
    releaseMemory(B, 2);
    releaseMemory(C, 2);
}
```

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

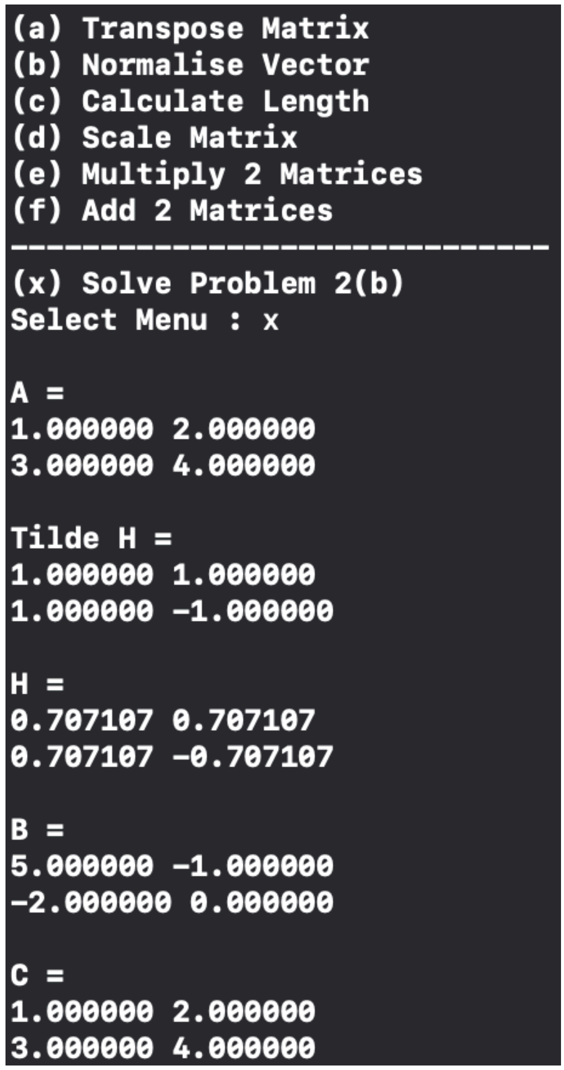
$$\tilde{H} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$H = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$$

$$\begin{aligned} B &= H^T A H \\ &= \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \\ &= \begin{bmatrix} 5 & -1 \\ -2 & 0 \end{bmatrix} \end{aligned}$$

$$\begin{aligned} C &= H B H^T \\ &= \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 5 & -1 \\ -2 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \\ &= \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \end{aligned}$$

(a) Equation



(b) Result Image

Fig. 7: `problem2b()`