Aim ↬ To implement Matrix using arrays.

Objectives ↬

i. Write a program to create two matrices of size 4x4 and calculate and print the product matrix.
ii. Write a program to create a matrix of size 5x5. Find whether it is orthogonal or not. (i.e. $\mathbf{A}^T\mathbf{A} = I$ or not).

Software Required ↬ Visual Studio Code

Code 1 ↬

```
#include<stdio.h>
int main(){
   int a[4][4], b[4][4], m[4][4];
   printf("\nEnter elements in first matrix:\n");
   for(int i=1; i<=4; i++){
      for(int j=1; j<=4; j++){
         printf("Element at index %d%d: ",i, j);
         scanf("%d", &a[i-1][j-1]);}
   }
   printf("\nEnter elements in second matrix:\n");
   for(int i=1; i<=4; i++){
      for(int j=1; j<=4; j++){
         printf("Element at index %d%d: ",i, j);
         scanf("%d", &b[i-1][j-1]);}
   }
   printf("\nMatrix a:\n");
   for(int i=1; i<=4; i++){
      for(int j=1; j<=4; j++){
         printf("%d ",a[i-1][j-1]);
         if(j==4) printf("\n\n");}
   }
   printf("Matrix b:\n");
   for(int i=1; i<=4; i++){

      for(int j=1; j<=4; j++){
         printf("%d ",b[i-1][j-1]);
         if(j==4) printf("\n\n");}
```

```
    }
    for(int i=0; i<4; i++){
        for(int k=0; k<4; k++){
            int pro=0;
            for(int j=0; j<4; j++)
                pro = pro + a[i][j]*b[j][k];
            m[i][k] = pro;
        }
    }
    printf("Matrix m:\n");
    for(int i=1; i<=4; i++){
        for(int j=1; j<=4; j++){
            printf("%d ",m[i-1][j-1]);
            if(j==4) printf("\n\n");
        }
    }
}
```

Output ↬

```
Enter elements in first matrix:
Element at index 11: 2
Element at index 12: 4
Element at index 13: 3
Element at index 14: 1
Element at index 21: 3
Element at index 22: 6
Element at index 23: 5
Element at index 24: 1
Element at index 31: 2
Element at index 32: 4
Element at index 33: 3
Element at index 34: 4
Element at index 41: 2
Element at index 42: 1
Element at index 43: 1
Element at index 44: 3
Enter elements in second matrix:
Element at index 11: 1
Element at index 12: 3
Element at index 13: 4
Element at index 14: 2
Element at index 21: 6
Element at index 22: 5
Element at index 23: 1
Element at index 24: 4
```

```
Element at index 31: 2
Element at index 32: 5
Element at index 33: 3
Element at index 34: 4
Element at index 41: 1
Element at index 42: 2
Element at index 43: 5
Element at index 44: 1
Matrix a:
2 4 3 1

3 6 5 1

2 4 3 4

2 1 1 3

Matrix b:
1 3 4 2

6 5 1 4

2 5 3 4

1 2 5 1
```

```
Matrix m:
33 43 26 33

50 66 38 51

36 49 41 36

13 22 27 15
```

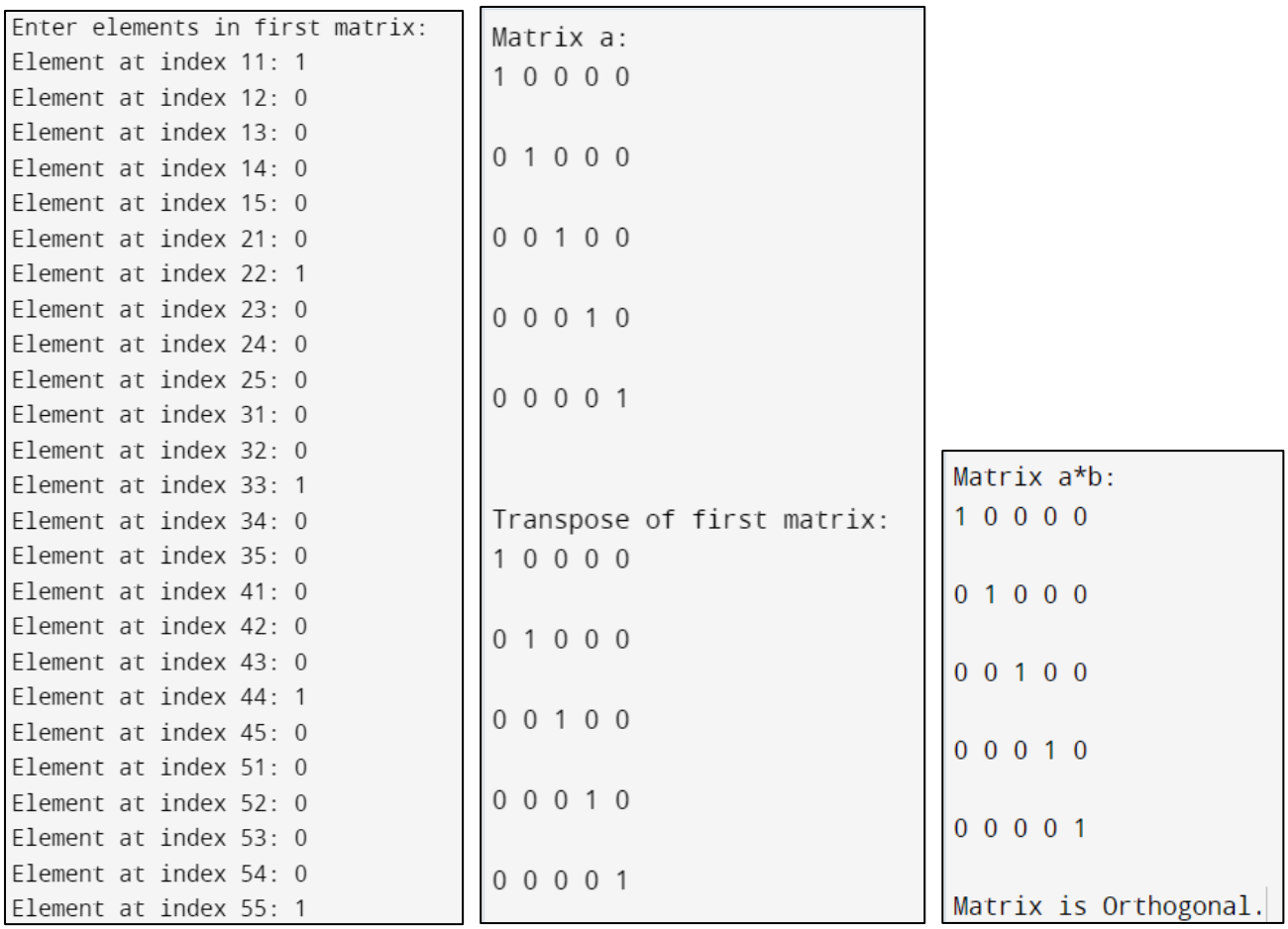
Code 2 ↬

```
#include<stdio.h>
int main(){
   int a[5][5], b[5][5], id[5][5], flag = 1;
   printf("\nEnter elements in first matrix:\n");
   for(int i=1; i<=5; i++){
      for(int j=1; j<=5; j++){
         printf("Element at index %d%d: ",i, j);
         scanf("%d", &a[i-1][j-1]);
      }
   }
   for(int i=0; i<5; i++){
      for(int j=0; j<5; j++){
         b[i][j] = a[j][i];
      }
   }
   printf("\nMatrix a:\n");
   for(int i=1; i<=5; i++){
      for(int j=1; j<=5; j++){
         printf("%d ",a[i-1][j-1]);
         if(j==5) printf("\n\n");
      }
   }
   printf("\nTranspose of first matrix:\n");
   for(int i=1; i<=5; i++){
      for(int j=1; j<=5; j++){
         printf("%d ",b[i-1][j-1]);
         if(j==5) printf("\n\n");
      }
   }
   for(int i=0; i<5; i++){
      for(int k=0; k<5; k++){
         int pro=0;
         for(int j=0; j<5; j++)
            pro = pro + a[i][j]*b[j][k];
         id[i][k] = pro;
      }
```

```
    }
    printf("Matrix a*b:\n");
    for(int i=1; i<=5; i++){
        for(int j=1; j<=5; j++){
            printf("%d ",id[i-1][j-1]);
            if(j==5) printf("\n\n");
        }
    }
    for(int i=1; i<=5; i++){
        for(int j=1; j<=5; j++){
            if (i == j){
                if (id[i-1][j-1]==1) flag = 1;
                else{
                    flag = 0;
                    break;
                }
            }
            else {
                if (id[i-1][j-1]==0) flag = 1;
                else{
                    flag = 0;
                    break;


                }
            }
        }
    }
    if (flag == 1) printf("Matrix is Orthogonal.");
    else printf("Matrix is not Orthogonal.");
}
```

Output ↬



```
Enter elements in first matrix:
Element at index 11: 1
Element at index 12: 0
Element at index 13: 0
Element at index 14: 0
Element at index 15: 0
Element at index 21: 0
Element at index 22: 1
Element at index 23: 0
Element at index 24: 0
Element at index 25: 0
Element at index 31: 0
Element at index 32: 0
Element at index 33: 1
Element at index 34: 0
Element at index 35: 0
Element at index 41: 0
Element at index 42: 0
Element at index 43: 0
Element at index 44: 1
Element at index 45: 0
Element at index 51: 0
Element at index 52: 0
Element at index 53: 0
Element at index 54: 0
Element at index 55: 1
```

```
Matrix a:
1 0 0 0 0

0 1 0 0 0

0 0 1 0 0

0 0 0 1 0

0 0 0 0 1


Transpose of first matrix:
1 0 0 0 0

0 1 0 0 0

0 0 1 0 0

0 0 0 1 0

0 0 0 0 1
```

```
Matrix a*b:
1 0 0 0 0

0 1 0 0 0

0 0 1 0 0

0 0 0 1 0

0 0 0 0 1

Matrix is Orthogonal.
```

Result ↬

The programs were successfully implemented to perform matrix operations, including the multiplication of two 4x4 matrices and checking the orthogonality of a 5x5 matrix. The outputs confirmed the correct execution of matrix multiplication and orthogonality checking, demonstrating accurate results.

Conclusion ↬

The experiment effectively demonstrated matrix manipulation using arrays, focusing on matrix multiplication and orthogonality testing. These operations highlight fundamental concepts in matrix algebra, enhancing the understanding of matrix calculations and their implementation in C programming.

Precautions ↬

- Ensure the input values for matrices are correctly entered, as incorrect inputs can lead to invalid results, especially in orthogonality checks.

- Validate the matrix dimensions before performing operations to ensure compatibility for multiplication.
- Handle large values carefully to avoid overflow during matrix multiplication, and double-check calculations when determining orthogonality to prevent logical errors.