CS-110: Lab 10

Two-Dimensional Arrays

https://github.com/mmujtaba25/CS-110

Muhammad Mujtaba

CMD ID: 540040

mmujtaba.bese25seecs@seecs.edu.pk

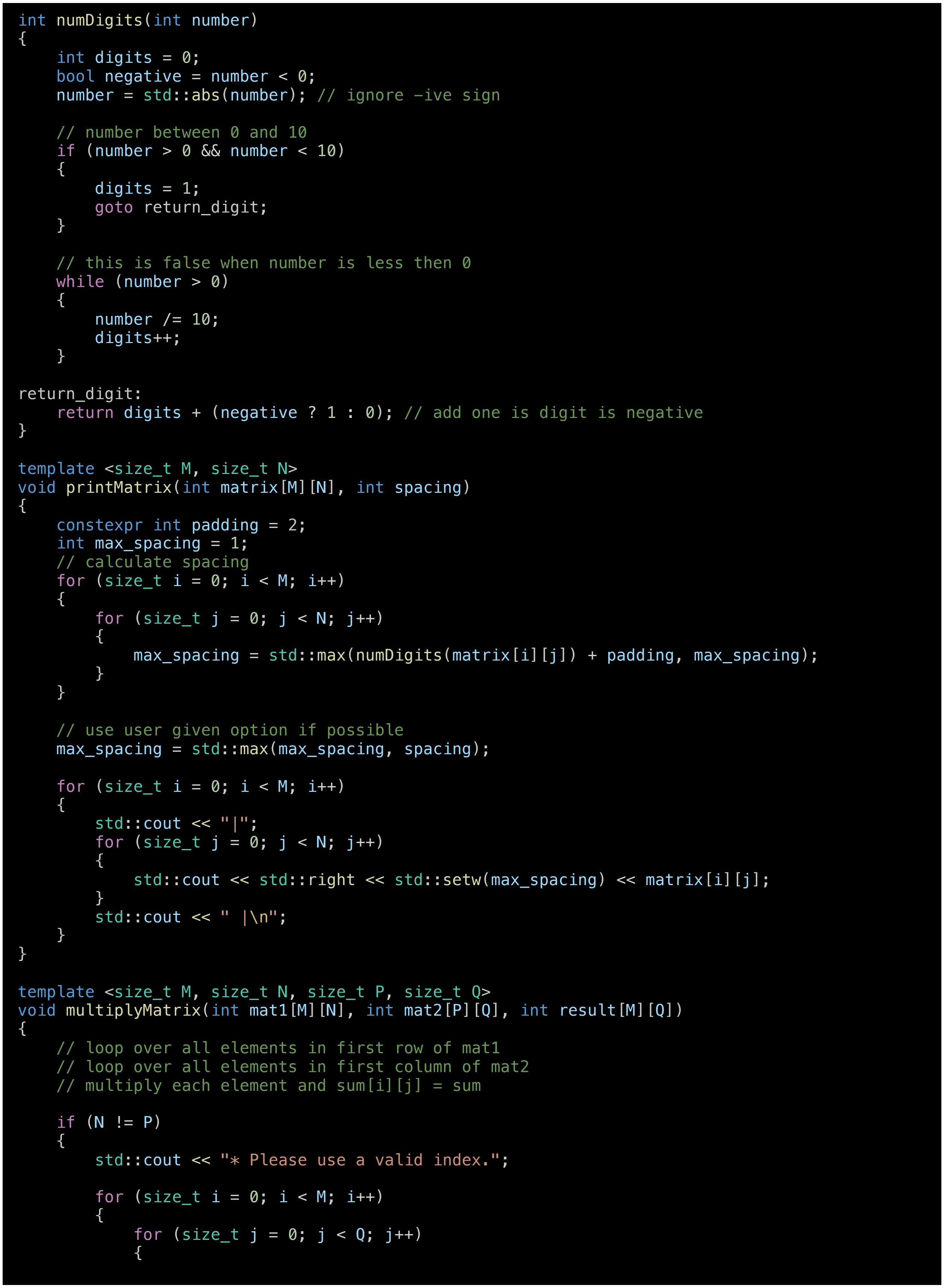
Class: BESE 16B

Batch: 2k25

Task 1: [CLO1]

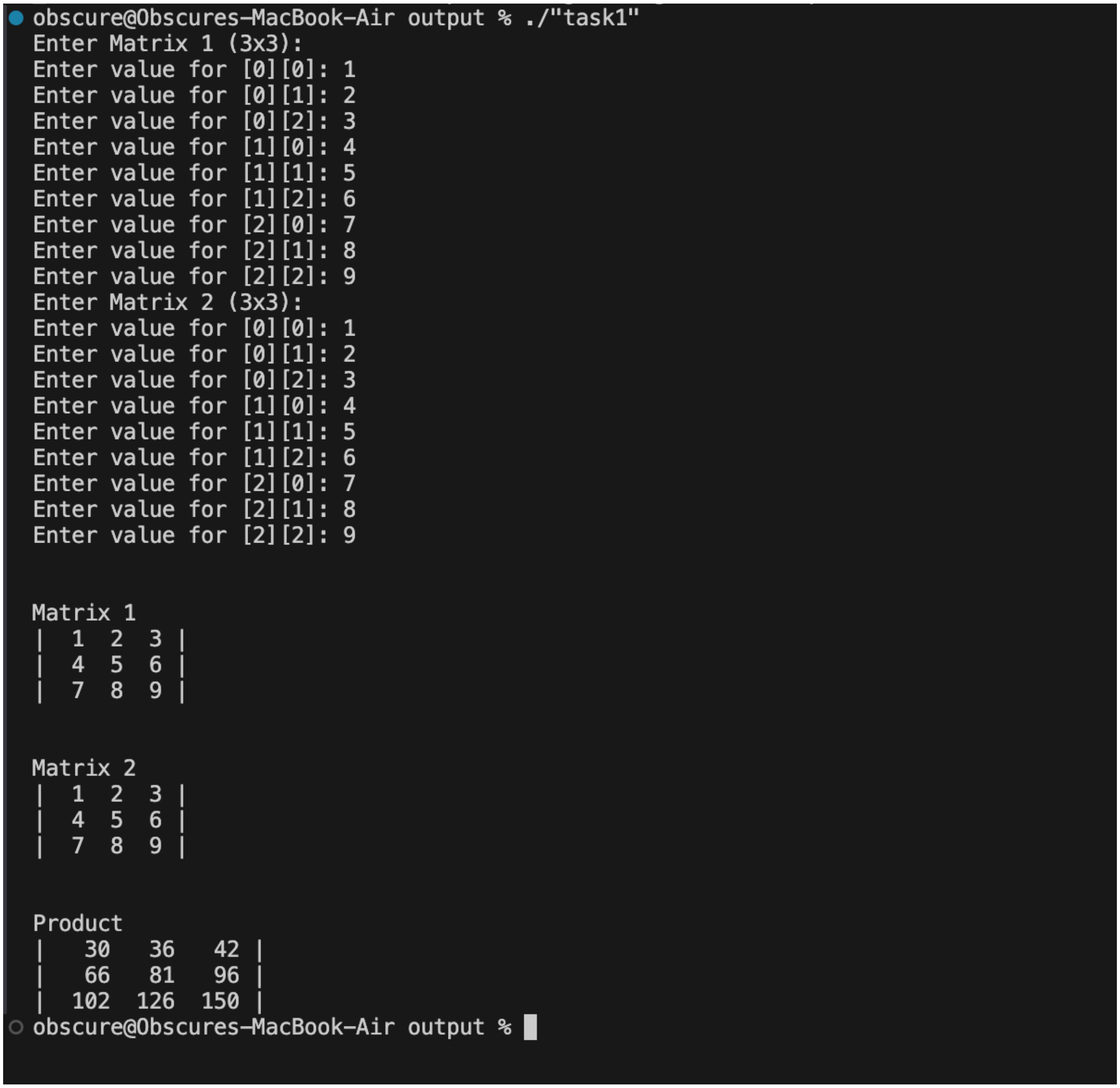
CODE:

```
#include <iostream>
#include <iomanip>

int numDigits(int number);

// templates prototypes works, because definations are given later

template <size_t M, size_t N>
void printMatrix(int matrix[M][N], int spacing = 0);

template <size_t M, size_t N, size_t P, size_t Q>
void multiplyMatrix(int mat1[M][N], int mat2[P][Q], int result[M][Q]);

template <size_t
 M, size_t N>
void readMatrix(int matrix[M][N]);

int main()
{
    constexpr int M = 3;
    constexpr int N = 3;
    constexpr int P = 3;
    constexpr int Q = 3;

    int mat1[M][N], mat2[P][Q];
    int prod[M][Q]; // for MxN * PxQ order is MxQ

    // init mat1
    std::cout << "Enter Matrix 1 (" << M << "x" << N << "):\n";
    readMatrix<M, N>(mat1);

    // init mat2
    std::cout << "Enter Matrix 2 (" << P << "x" << Q << "):\n";
    readMatrix<P, Q>(mat2);

    // init prod -> 0
    for (int i = 0; i < M; i++)
    {
        for (int j = 0; j < Q; j++)
        {
            prod[i][j] = 0;
        }
    }

    multiplyMatrix<M, N, P, Q>(mat1, mat2, prod);

    // print mat1 matrix
    std::cout << "\n\nMatrix 1\n";
    printMatrix<M, N>(mat1);

    // print mat2 matrix
    std::cout << "\n\nMatrix 2\n";
    printMatrix<P, Q>(mat2);

    // print sum matrix
    std::cout << "\n\nProduct\n";
    printMatrix<M, Q>(prod);

    return 0;
}
```



```
int numDigits(int number)
{
    int digits = 0;
    bool negative = number < 0;
    number = std::abs(number); // ignore -ive sign

    // number between 0 and 10
    if (number > 0 && number < 10)
    {
        digits = 1;
        goto return_digit;
    }

    // this is false when number is less then 0
    while (number > 0)
    {
        number /= 10;
        digits++;
    }

return_digit:
    return digits + (negative ? 1 : 0); // add one is digit is negative
}

template <size_t M, size_t N>
void printMatrix(int matrix[M][N], int spacing)
{
    constexpr int padding = 2;
    int max_spacing = 1;
    // calculate spacing
    for (size_t i = 0; i < M; i++)
    {
        for (size_t j = 0; j < N; j++)
        {
            max_spacing = std::max(numDigits(matrix[i][j]) + padding, max_spacing);
        }
    }

    // use user given option if possible
    max_spacing = std::max(max_spacing, spacing);

    for (size_t i = 0; i < M; i++)
    {
        std::cout << "|";
        for (size_t j = 0; j < N; j++)
        {
            std::cout << std::right << std::setw(max_spacing) << matrix[i][j];
        }
        std::cout << " |\n";
    }
}

template <size_t M, size_t N, size_t P, size_t Q>
void multiplyMatrix(int mat1[M][N], int mat2[P][Q], int result[M][Q])
{
    // loop over all elements in first row of mat1
    // loop over all elements in first column of mat2
    // multiply each element and sum[i][j] = sum

    if (N != P)
    {
        std::cout << "* Please use a valid index.";

        for (size_t i = 0; i < M; i++)
        {
            for (size_t j = 0; j < Q; j++)
            {
                // set all to -1 in case of error
                result[i][j] = -1;
            }
        }
        return;
    }

    for (size_t i = 0; i < M; i++)
    {
        for (size_t j = 0; j < Q; j++)
        {
            int sum = 0;

            // sum[i][j] = summation(mat1[i][k] * mat2[k][j])
            // k goes from (0 -> N) where N in MxN * PxQ, note that P is same as N in this scenario
            for (size_t k = 0; k < N; k++)
            {
                sum += mat1[i][k] * mat2[k][j];
            }

            result[i][j] = sum;
        }
    }
}

template <size_t M, size_t N>
void readMatrix(int matrix[M][N])
{
    for (size_t i = 0; i < M; i++)
    {
        for (size_t j = 0; j < N; j++)
        {
            int value;

            while (true)
            {
                std::cout << "Enter value for [" << i << "][" << j << "]: ";

                if (std::cin >> value)
                    break; // valid integer

                // invalid input
                std::cout << "* Invalid input. Please enter an integer.\n";

                // silently discard invalid input
                std::cin.clear();
                std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            }

            matrix[i][j] = value;
        }
    }
}
```

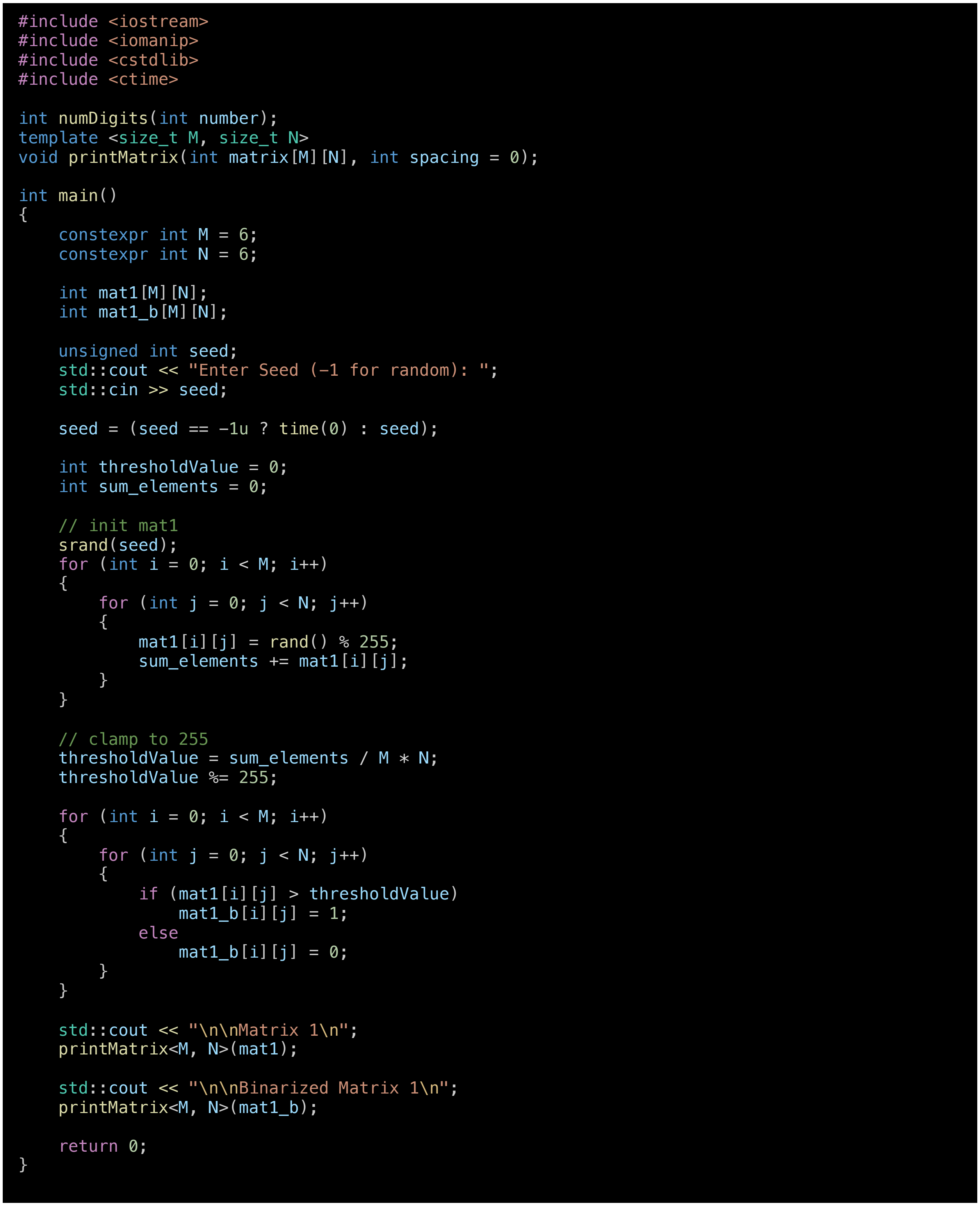
OUTPUT:



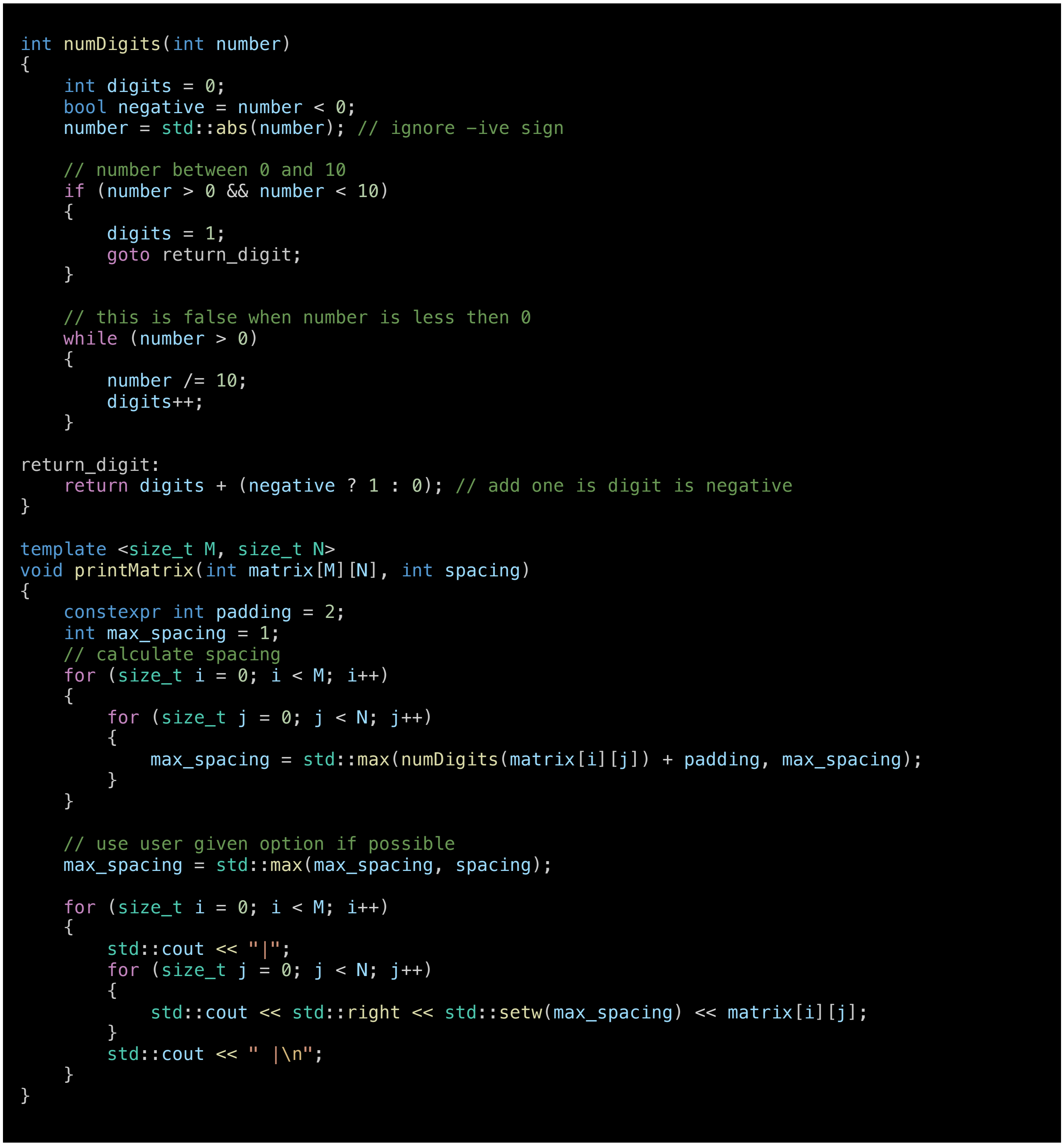
```
obscure@Obscures-MacBook-Air output % ./"task1"
Enter Matrix 1 (3x3):
Enter value for [0][0]: 1
Enter value for [0][1]: 2
Enter value for [0][2]: 3
Enter value for [1][0]: 4
Enter value for [1][1]: 5
Enter value for [1][2]: 6
Enter value for [2][0]: 7
Enter value for [2][1]: 8
Enter value for [2][2]: 9
Enter Matrix 2 (3x3):
Enter value for [0][0]: 1
Enter value for [0][1]: 2
Enter value for [0][2]: 3
Enter value for [1][0]: 4
Enter value for [1][1]: 5
Enter value for [1][2]: 6
Enter value for [2][0]: 7
Enter value for [2][1]: 8
Enter value for [2][2]: 9


Matrix 1
|  1  2  3 |
|  4  5  6 |
|  7  8  9 |


Matrix 2
|  1  2  3 |
|  4  5  6 |
|  7  8  9 |


Product
|   30   36   42 |
|   66   81   96 |
|  102  126  150 |
obscure@Obscures-MacBook-Air output % 
```

Task 2: [CLO 2]

CODE:

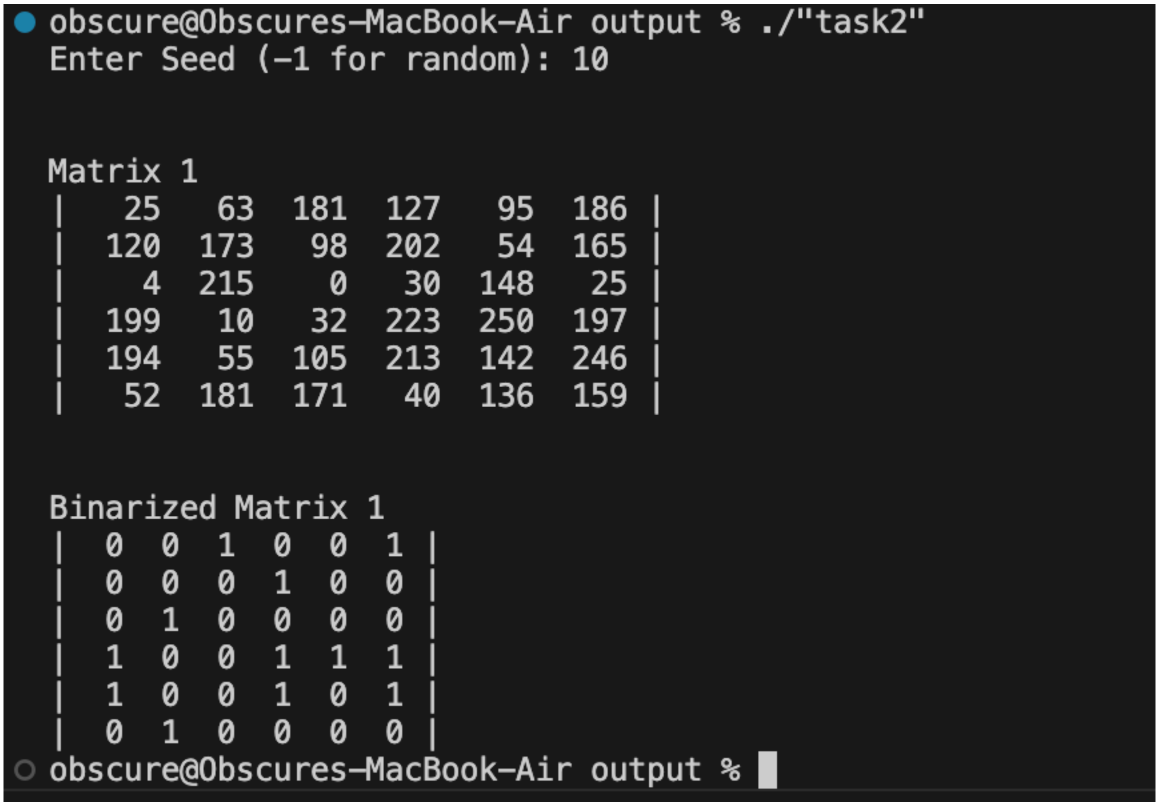


```cpp
#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <ctime>

int numDigits(int number);
template <size_t M, size_t N>
void printMatrix(int matrix[M][N], int spacing = 0);

int main()
{
    constexpr int M = 6;
    constexpr int N = 6;

    int mat1[M][N];
    int mat1_b[M][N];

    unsigned int seed;
    std::cout << "Enter Seed (-1 for random): ";
    std::cin >> seed;

    seed = (seed == -1u ? time(0) : seed);

    int thresholdValue = 0;
    int sum_elements = 0;

    // init mat1
    srand(seed);
    for (int i = 0; i < M; i++)
    {
        for (int j = 0; j < N; j++)
        {
            mat1[i][j] = rand() % 255;
            sum_elements += mat1[i][j];
        }
    }

    // clamp to 255
    thresholdValue = sum_elements / M * N;
    thresholdValue %= 255;

    for (int i = 0; i < M; i++)
    {
        for (int j = 0; j < N; j++)
        {
            if (mat1[i][j] > thresholdValue)
                mat1_b[i][j] = 1;
            else
                mat1_b[i][j] = 0;
        }
    }

    std::cout << "\n\nMatrix 1\n";
    printMatrix<M, N>(mat1);

    std::cout << "\n\nBinarized Matrix 1\n";
    printMatrix<M, N>(mat1_b);

    return 0;
}
```

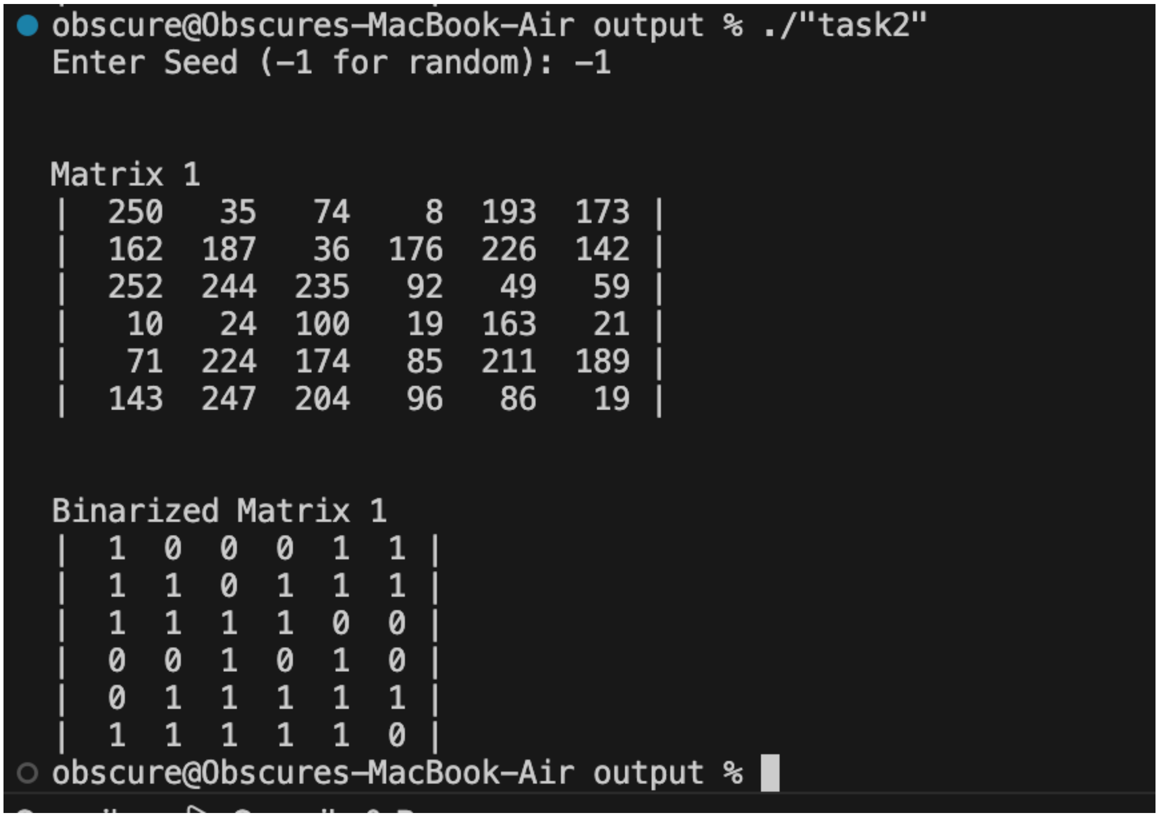


```cpp
int numDigits(int number)
{
    int digits = 0;
    bool negative = number < 0;
    number = std::abs(number); // ignore -ive sign

    // number between 0 and 10
    if (number > 0 && number < 10)
    {
        digits = 1;
        goto return_digit;
    }

    // this is false when number is less then 0
    while (number > 0)
    {
        number /= 10;
        digits++;
    }

return_digit:
    return digits + (negative ? 1 : 0); // add one is digit is negative
}

template <size_t M, size_t N>
void printMatrix(int matrix[M][N], int spacing)
{
    constexpr int padding = 2;
    int max_spacing = 1;
    // calculate spacing
    for (size_t i = 0; i < M; i++)
    {
        for (size_t j = 0; j < N; j++)
        {
            max_spacing = std::max(numDigits(matrix[i][j]) + padding, max_spacing);
        }
    }

    // use user given option if possible
    max_spacing = std::max(max_spacing, spacing);

    for (size_t i = 0; i < M; i++)
    {
        std::cout << "|";
        for (size_t j = 0; j < N; j++)
        {
            std::cout << std::right << std::setw(max_spacing) << matrix[i][j];
        }
        std::cout << " |\n";
    }
}
```

OUTPUT:



```
● obscure@Obscures-MacBook-Air output % ./"task2"
  Enter Seed (-1 for random): 10


  Matrix 1
  |   25   63  181  127   95  186 |
  |  120  173   98  202   54  165 |
  |    4  215    0   30  148   25 |
  |  199   10   32  223  250  197 |
  |  194   55  105  213  142  246 |
  |   52  181  171   40  136  159 |


  Binarized Matrix 1
  |   0   0   1   0   0   1 |
  |   0   0   0   1   0   0 |
  |   0   1   0   0   0   0 |
  |   1   0   0   1   1   1 |
  |   1   0   0   1   0   1 |
  |   0   1   0   0   0   0 |
○ obscure@Obscures-MacBook-Air output % █
```

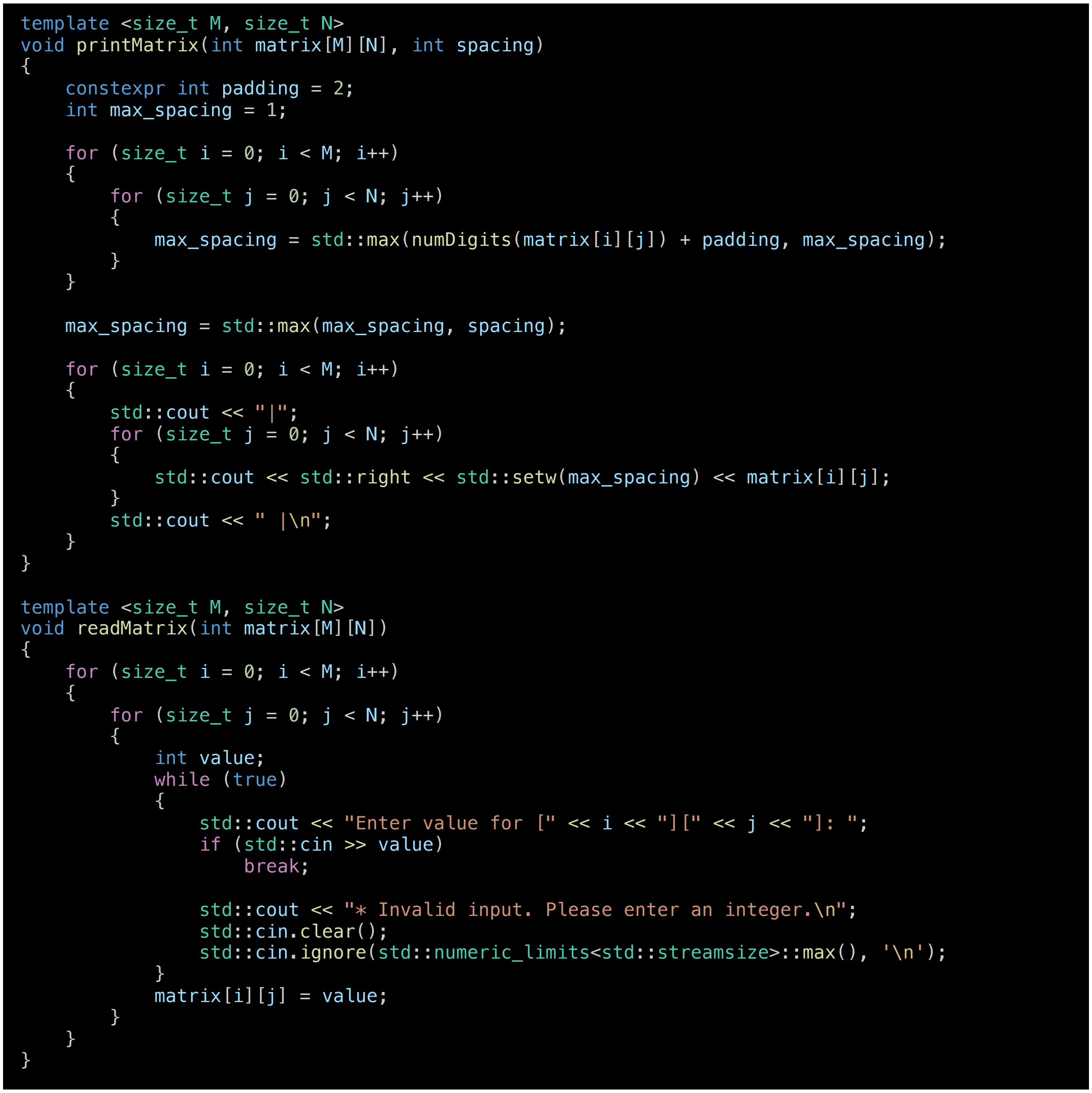


```
● obscure@Obscures-MacBook-Air output % ./"task2"
  Enter Seed (-1 for random): -1


  Matrix 1
  |  250   35   74    8  193  173 |
  |  162  187   36  176  226  142 |
  |  252  244  235   92   49   59 |
  |   10   24  100   19  163   21 |
  |   71  224  174   85  211  189 |
  |  143  247  204   96   86   19 |


  Binarized Matrix 1
  |   1   0   0   0   1   1 |
  |   1   1   0   1   1   1 |
  |   1   1   1   1   0   0 |
  |   0   0   1   0   1   0 |
  |   0   1   1   1   1   1 |
  |   1   1   1   1   1   0 |
○ obscure@Obscures-MacBook-Air output % █
```

Task 3: [CLO 3]

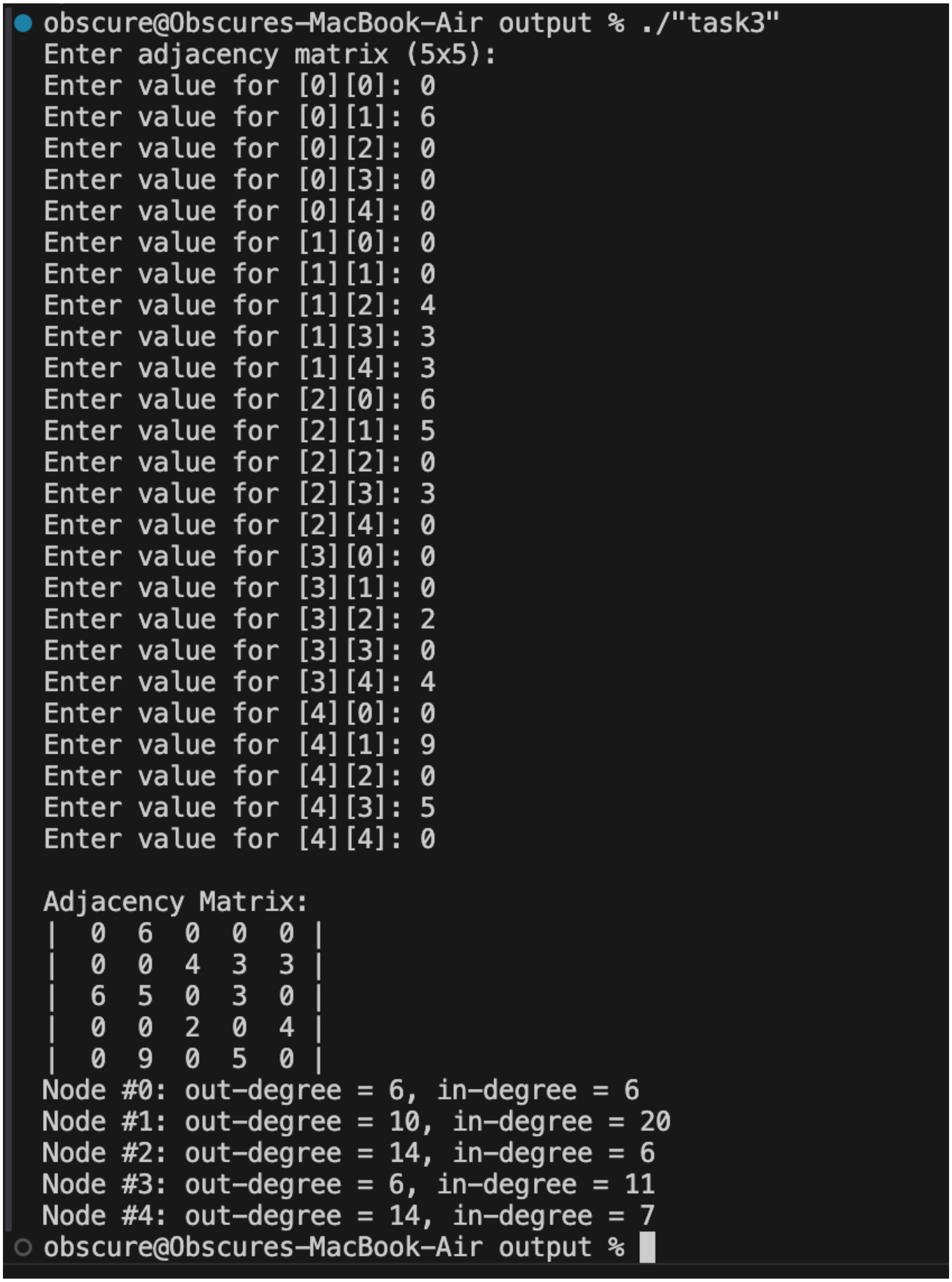
CODE:

```
#include <iostream>
#include <iomanip>
int numDigits(int number);
template <size_t M, size_t N>
void printMatrix(int matrix[M][N], int spacing = 0);
template <size_t M, size_t N>
void readMatrix(int matrix[M][N]);
int main()
{
    constexpr int N = 5;
    int adjacency[N][N];

    std::cout << "Enter adjacency matrix (" << N << "x" << N << "):\n";
    readMatrix<N, N>(adjacency);

    int inDegrees[N] = {0};
    int outDegrees[N] = {0};

    // compute degrees
    for (size_t i = 0; i < N; i++)
    {
        for (size_t j = 0; j < N; j++)
        {
            outDegrees[i] += adjacency[i][j]; // sum of row = out-degree
            inDegrees[j] += adjacency[i][j];  // sum of column = in-degree
        }
    }

    std::cout << "\nAdjacency Matrix:\n";
    printMatrix<N, N>(adjacency);

    for (size_t i = 0; i < N; i++)
    {
        std::cout << "Node #" << i << ": out-degree = " << outDegrees[i]
                  << ", in-degree = " << inDegrees[i] << "\n";
    }

    return 0;
}

int numDigits(int number)
{
    int digits = 0;
    bool negative = number < 0;
    number = std::abs(number);

    if (number > 0 && number < 10)
    {
        digits = 1;
        goto return_digit;
    }

    while (number > 0)
    {
        number /= 10;
        digits++;
    }

return_digit:
    return digits + (negative ? 1 : 0);
}
```



```
template <size_t M, size_t N>
void printMatrix(int matrix[M][N], int spacing)
{
    constexpr int padding = 2;
    int max_spacing = 1;

    for (size_t i = 0; i < M; i++)
    {
        for (size_t j = 0; j < N; j++)
        {
            max_spacing = std::max(numDigits(matrix[i][j]) + padding, max_spacing);
        }
    }

    max_spacing = std::max(max_spacing, spacing);

    for (size_t i = 0; i < M; i++)
    {
        std::cout << "|";
        for (size_t j = 0; j < N; j++)
        {
            std::cout << std::right << std::setw(max_spacing) << matrix[i][j];
        }
        std::cout << " |\n";
    }
}

template <size_t M, size_t N>
void readMatrix(int matrix[M][N])
{
    for (size_t i = 0; i < M; i++)
    {
        for (size_t j = 0; j < N; j++)
        {
            int value;
            while (true)
            {
                std::cout << "Enter value for [" << i << "][" << j << "]: ";
                if (std::cin >> value)
                    break;

                std::cout << "* Invalid input. Please enter an integer.\n";
                std::cin.clear();
                std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            }
            matrix[i][j] = value;
        }
    }
}
```

OUTPUT:



```
● obscure@Obscures-MacBook-Air output % ./"task3"
  Enter adjacency matrix (5x5):
  Enter value for [0][0]: 0
  Enter value for [0][1]: 6
  Enter value for [0][2]: 0
  Enter value for [0][3]: 0
  Enter value for [0][4]: 0
  Enter value for [1][0]: 0
  Enter value for [1][1]: 0
  Enter value for [1][2]: 4
  Enter value for [1][3]: 3
  Enter value for [1][4]: 3
  Enter value for [2][0]: 6
  Enter value for [2][1]: 5
  Enter value for [2][2]: 0
  Enter value for [2][3]: 3
  Enter value for [2][4]: 0
  Enter value for [3][0]: 0
  Enter value for [3][1]: 0
  Enter value for [3][2]: 2
  Enter value for [3][3]: 0
  Enter value for [3][4]: 4
  Enter value for [4][0]: 0
  Enter value for [4][1]: 9
  Enter value for [4][2]: 0
  Enter value for [4][3]: 5
  Enter value for [4][4]: 0

  Adjacency Matrix:
  |  0  6  0  0  0 |
  |  0  0  4  3  3 |
  |  6  5  0  3  0 |
  |  0  0  2  0  4 |
  |  0  9  0  5  0 |
  Node #0: out-degree = 6, in-degree = 6
  Node #1: out-degree = 10, in-degree = 20
  Node #2: out-degree = 14, in-degree = 6
  Node #3: out-degree = 6, in-degree = 11
  Node #4: out-degree = 14, in-degree = 7
○ obscure@Obscures-MacBook-Air output %
```