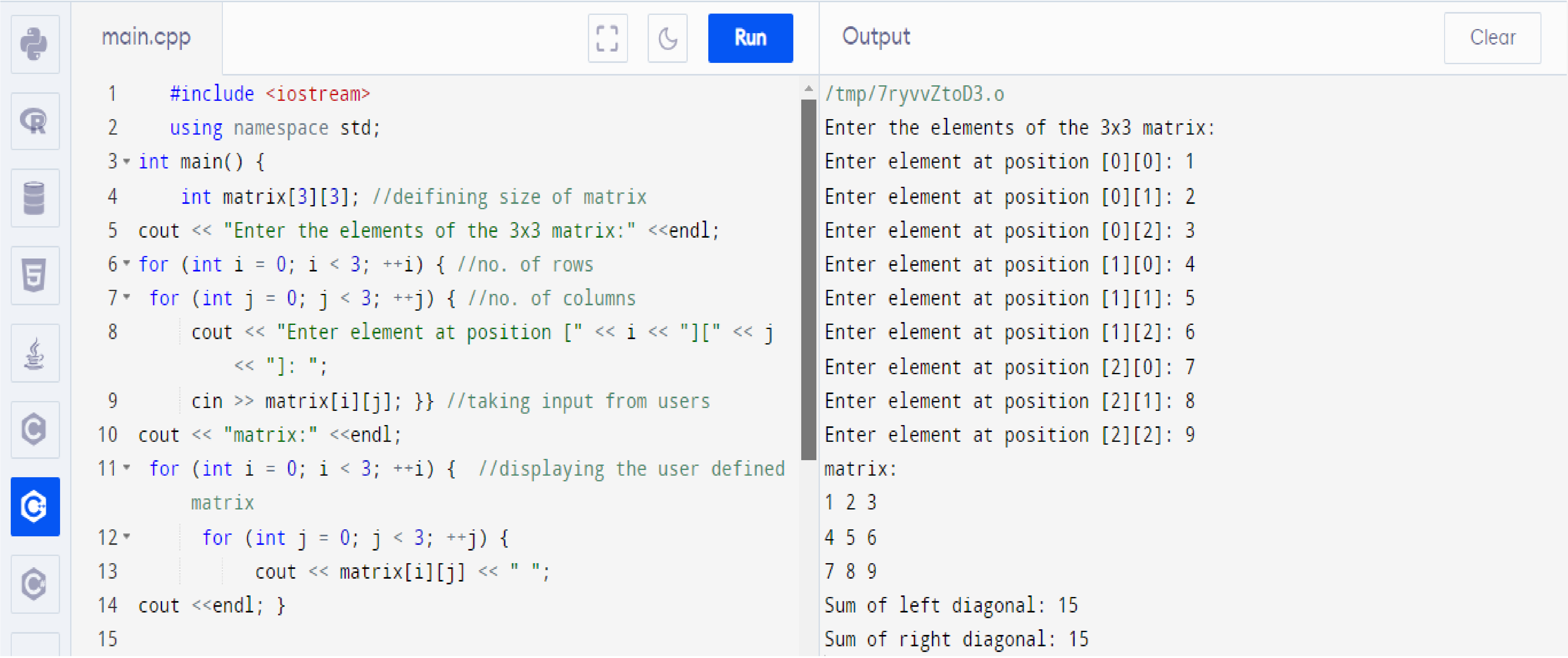
FUNDAMENTALS OF PROGRAMMING (LAB)

LAB MANUAL 9 (LAB +HOME TASKS)

SUBMITTED BY: - ABEER ZAHRA JAFARI (476474) ME-15(C)

TASK # 01

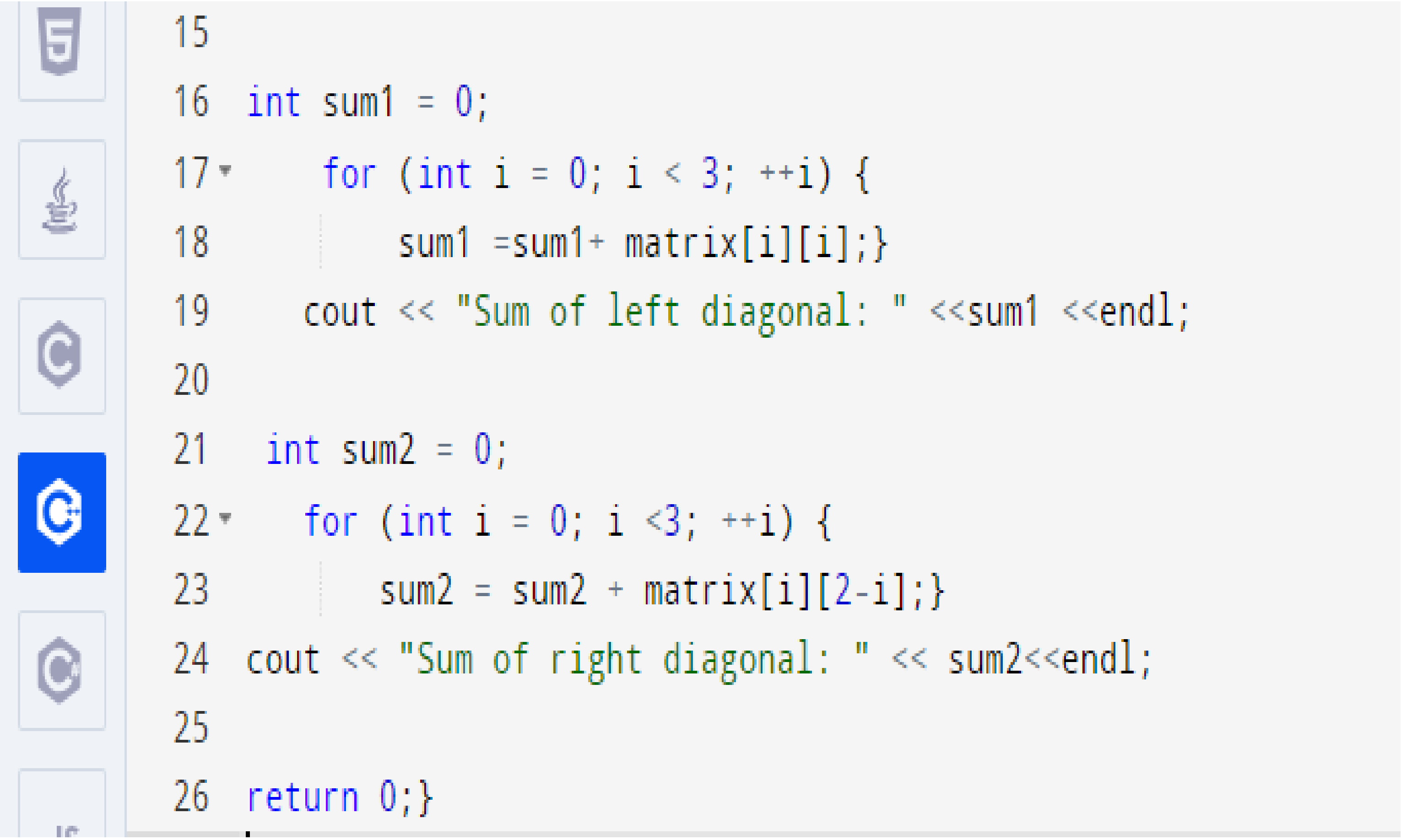
- Make 2D Array in C++ and print left diagonal and right diagonal sum of a 3x3 matrix.



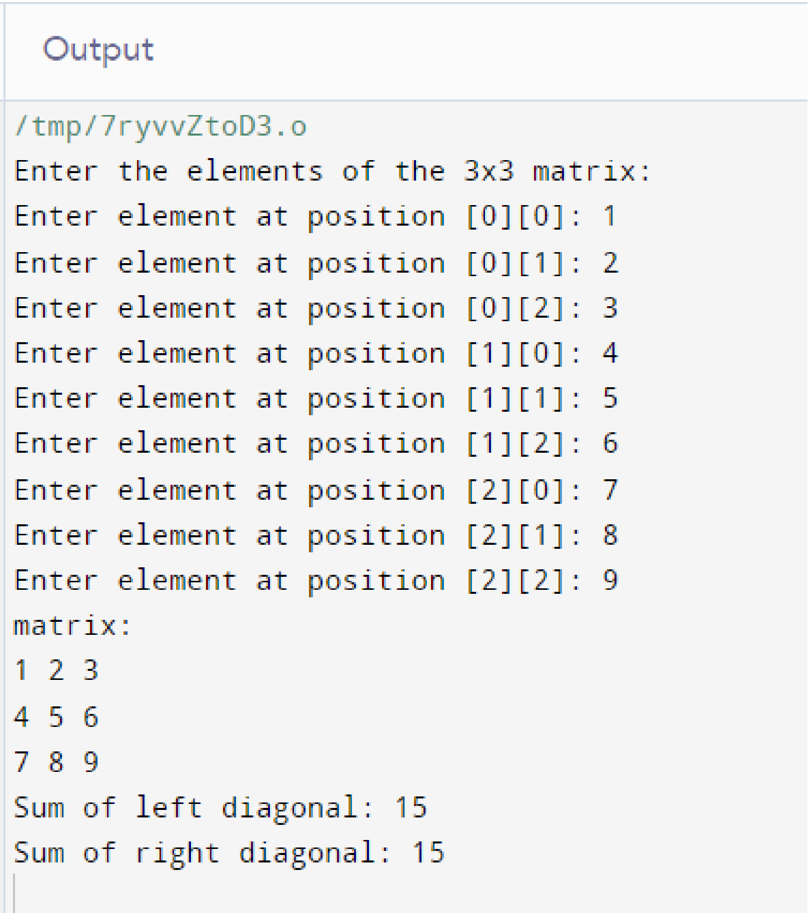
```cpp
  #include <iostream>
  using namespace std;
int main() {
    int matrix[3][3]; //deifining size of matrix
cout << "Enter the elements of the 3x3 matrix:" <<endl;
for (int i = 0; i < 3; ++i) { //no. of rows
 for (int j = 0; j < 3; ++j) { //no. of columns
     cout << "Enter element at position [" << i << "][" << j << "]: ";
     cin >> matrix[i][j]; }} //taking input from users
cout << "matrix:" <<endl;
 for (int i = 0; i < 3; ++i) {  //displaying the user defined matrix
      for (int j = 0; j < 3; ++j) {
           cout << matrix[i][j] << " ";
cout <<endl; }

```

```
/tmp/7ryvvZtoD3.o
Enter the elements of the 3x3 matrix:
Enter element at position [0][0]: 1
Enter element at position [0][1]: 2
Enter element at position [0][2]: 3
Enter element at position [1][0]: 4
Enter element at position [1][1]: 5
Enter element at position [1][2]: 6
Enter element at position [2][0]: 7
Enter element at position [2][1]: 8
Enter element at position [2][2]: 9
matrix:
1 2 3
4 5 6
7 8 9
Sum of left diagonal: 15
Sum of right diagonal: 15
```

TASK # 01 (contd)



```

int sum1 = 0;
    for (int i = 0; i < 3; ++i) {
        sum1 =sum1+ matrix[i][i];}
  cout << "Sum of left diagonal: " <<sum1 <<endl;

 int sum2 = 0;
   for (int i = 0; i <3; ++i) {
       sum2 = sum2 + matrix[i][2-i];}
cout << "Sum of right diagonal: " << sum2<<endl;

return 0;}
```

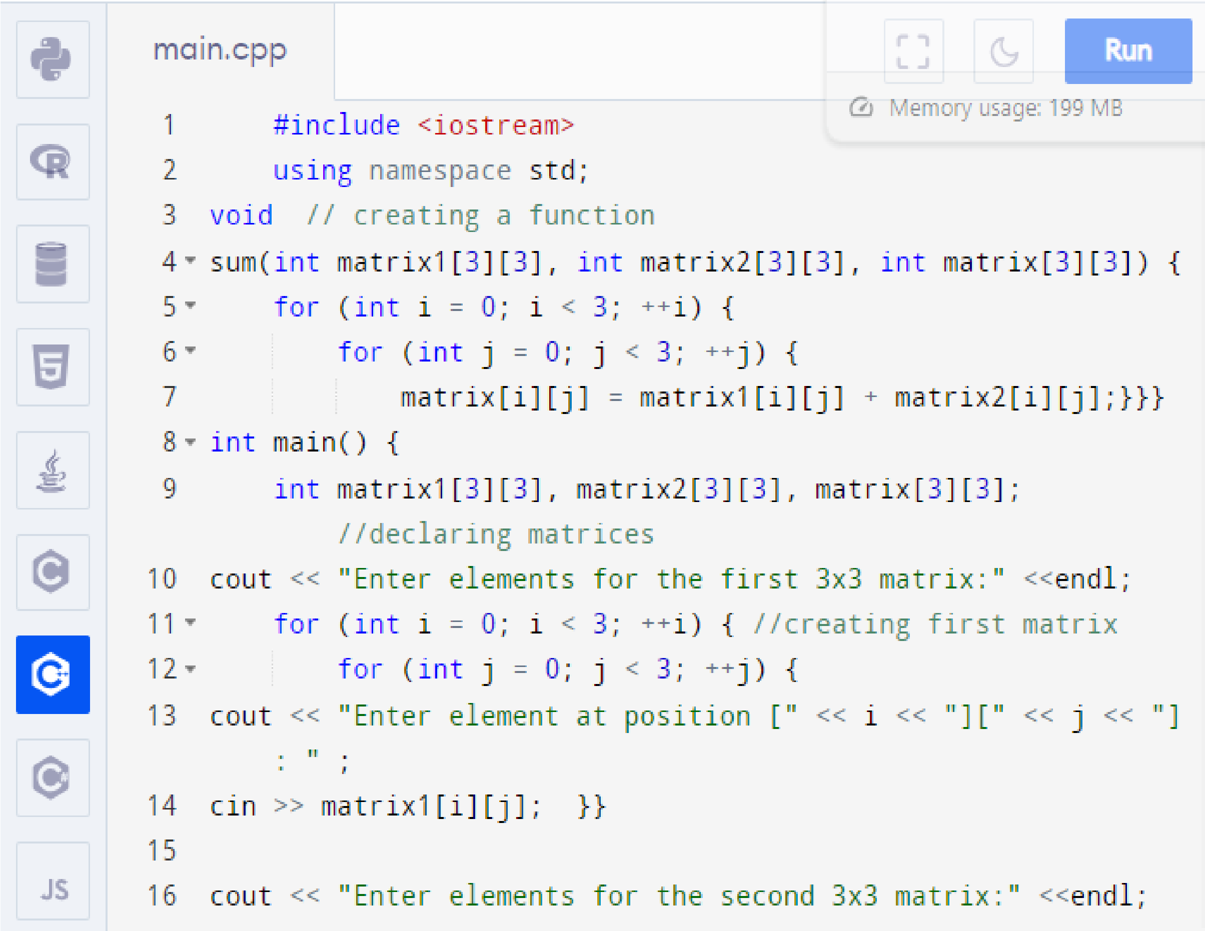


Output

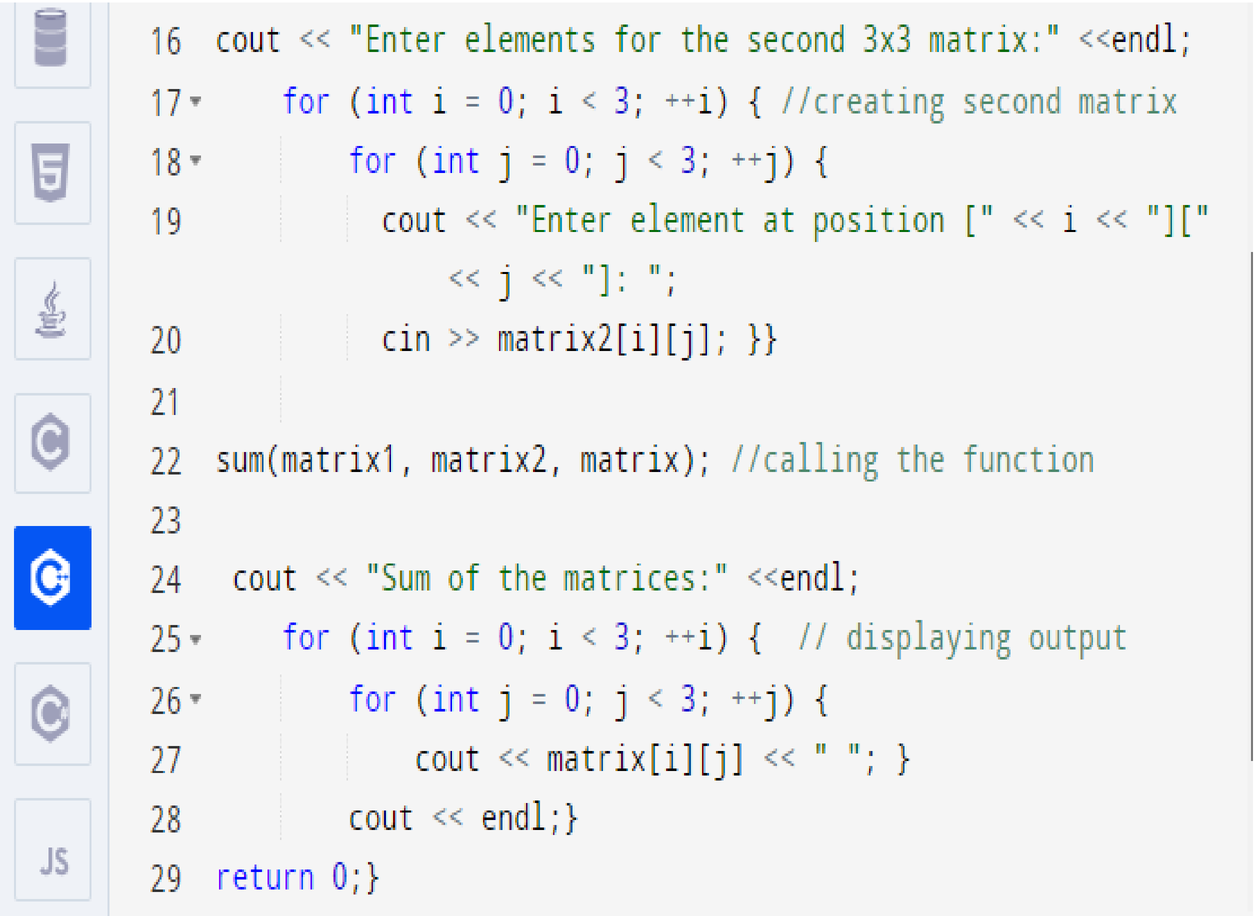
```
/tmp/7ryvvZtoD3.o
Enter the elements of the 3x3 matrix:
Enter element at position [0][0]: 1
Enter element at position [0][1]: 2
Enter element at position [0][2]: 3
Enter element at position [1][0]: 4
Enter element at position [1][1]: 5
Enter element at position [1][2]: 6
Enter element at position [2][0]: 7
Enter element at position [2][1]: 8
Enter element at position [2][2]: 9
matrix:
1 2 3
4 5 6
7 8 9
Sum of left diagonal: 15
Sum of right diagonal: 15
```

TASK # 02

- Write a function to add two 2D arrays of size 3x3.



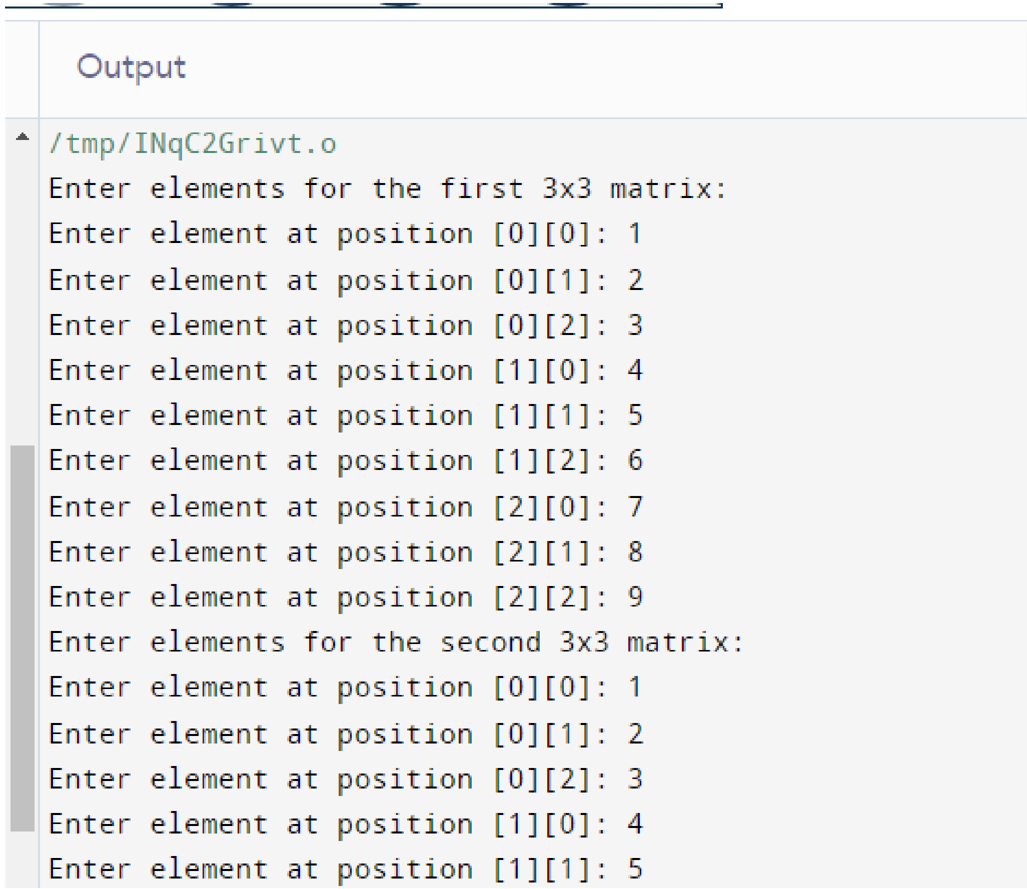
```
    #include <iostream>
    using namespace std;
void  // creating a function
sum(int matrix1[3][3], int matrix2[3][3], int matrix[3][3]) {
    for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 3; ++j) {
            matrix[i][j] = matrix1[i][j] + matrix2[i][j];}}}
int main() {
    int matrix1[3][3], matrix2[3][3], matrix[3][3];
        //declaring matrices
cout << "Enter elements for the first 3x3 matrix:" <<endl;
    for (int i = 0; i < 3; ++i) { //creating first matrix
        for (int j = 0; j < 3; ++j) {
cout << "Enter element at position [" << i << "][" << j << "]
    : " ;
cin >> matrix1[i][j];  }}

cout << "Enter elements for the second 3x3 matrix:" <<endl;
```



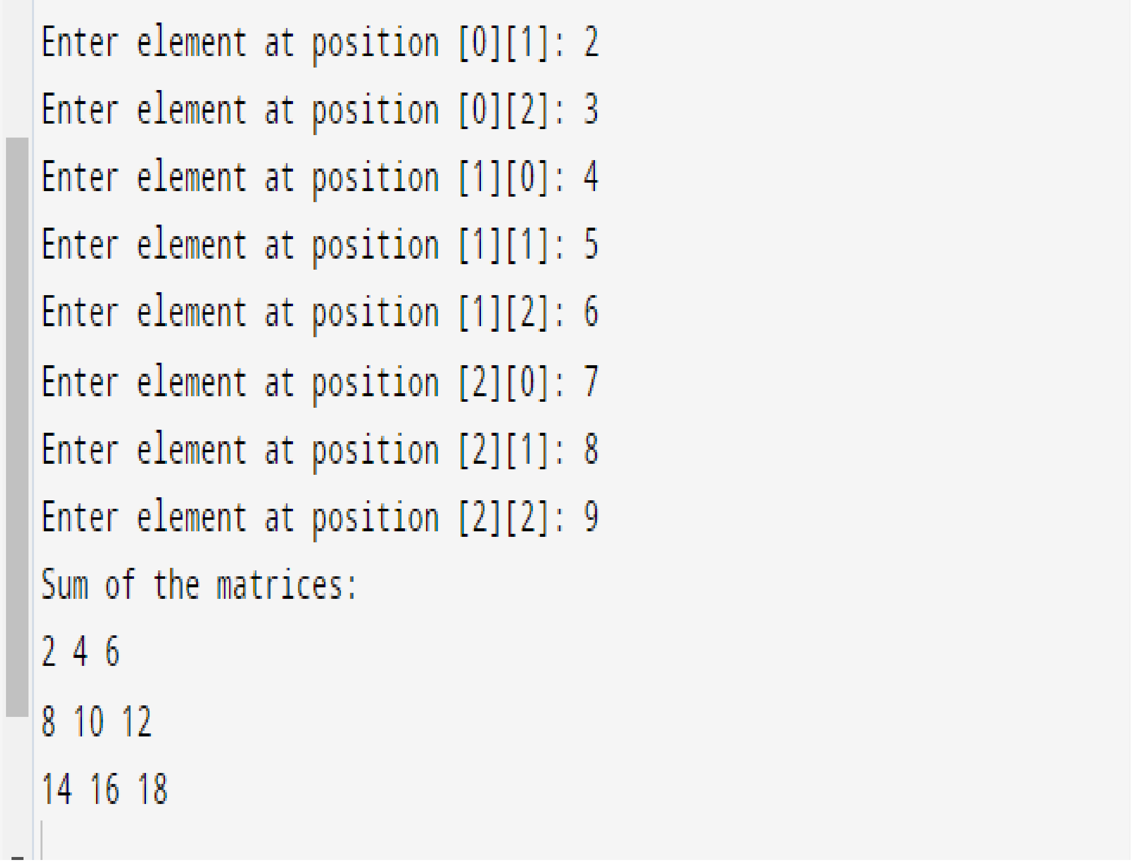
cout << "Enter elements for the second 3x3 matrix:" <<endl;

```
    for (int i = 0; i < 3; ++i) { //creating second matrix
        for (int j = 0; j < 3; ++j) {
            cout << "Enter element at position [" << i << "]["
                << j << "]: ";
            cin >> matrix2[i][j]; }}


sum(matrix1, matrix2, matrix); //calling the function


 cout << "Sum of the matrices:" <<endl;
    for (int i = 0; i < 3; ++i) {  // displaying output
        for (int j = 0; j < 3; ++j) {
            cout << matrix[i][j] << " "; }
        cout << endl;}
return 0;}
```

TASK # 02 (OUTPUT)



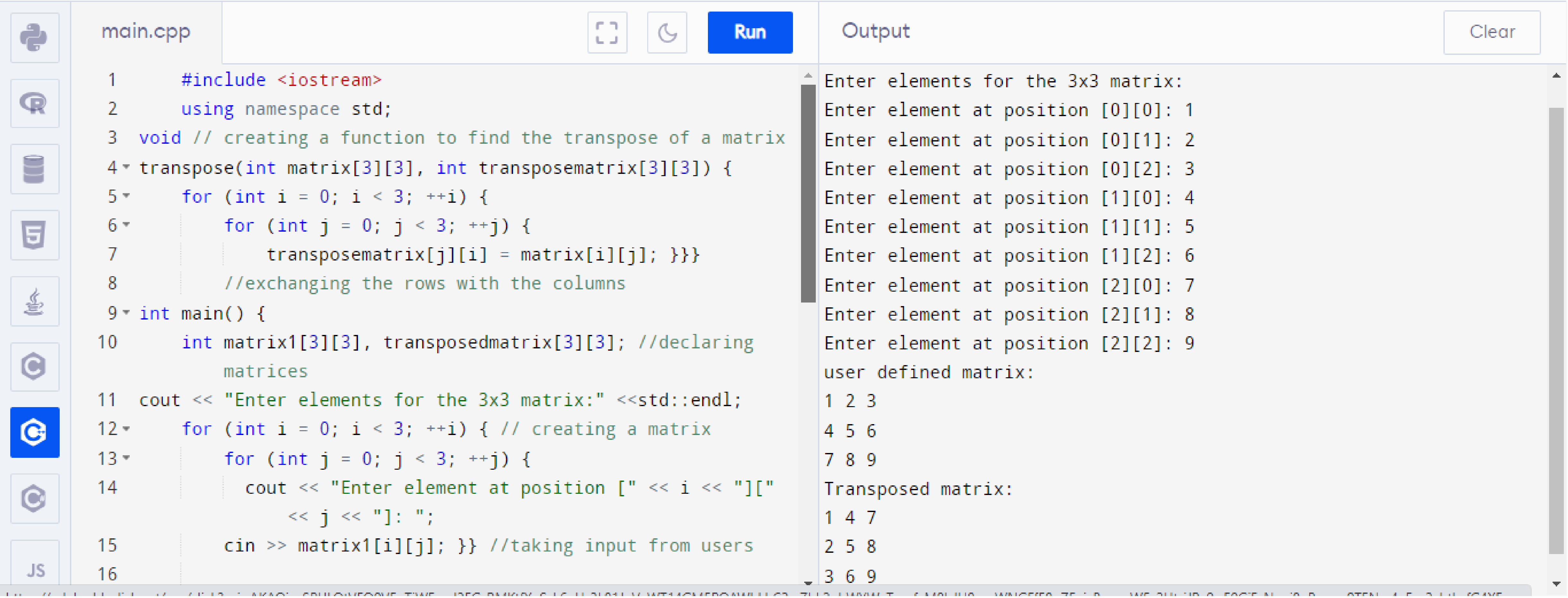
```
/tmp/INqC2Grivt.o
Enter elements for the first 3x3 matrix:
Enter element at position [0][0]: 1
Enter element at position [0][1]: 2
Enter element at position [0][2]: 3
Enter element at position [1][0]: 4
Enter element at position [1][1]: 5
Enter element at position [1][2]: 6
Enter element at position [2][0]: 7
Enter element at position [2][1]: 8
Enter element at position [2][2]: 9
Enter elements for the second 3x3 matrix:
Enter element at position [0][0]: 1
Enter element at position [0][1]: 2
Enter element at position [0][2]: 3
Enter element at position [1][0]: 4
Enter element at position [1][1]: 5
```



```
Enter element at position [0][1]: 2
Enter element at position [0][2]: 3
Enter element at position [1][0]: 4
Enter element at position [1][1]: 5
Enter element at position [1][2]: 6
Enter element at position [2][0]: 7
Enter element at position [2][1]: 8
Enter element at position [2][2]: 9
Sum of the matrices:
2 4 6
8 10 12
14 16 18
```

TASK # 03

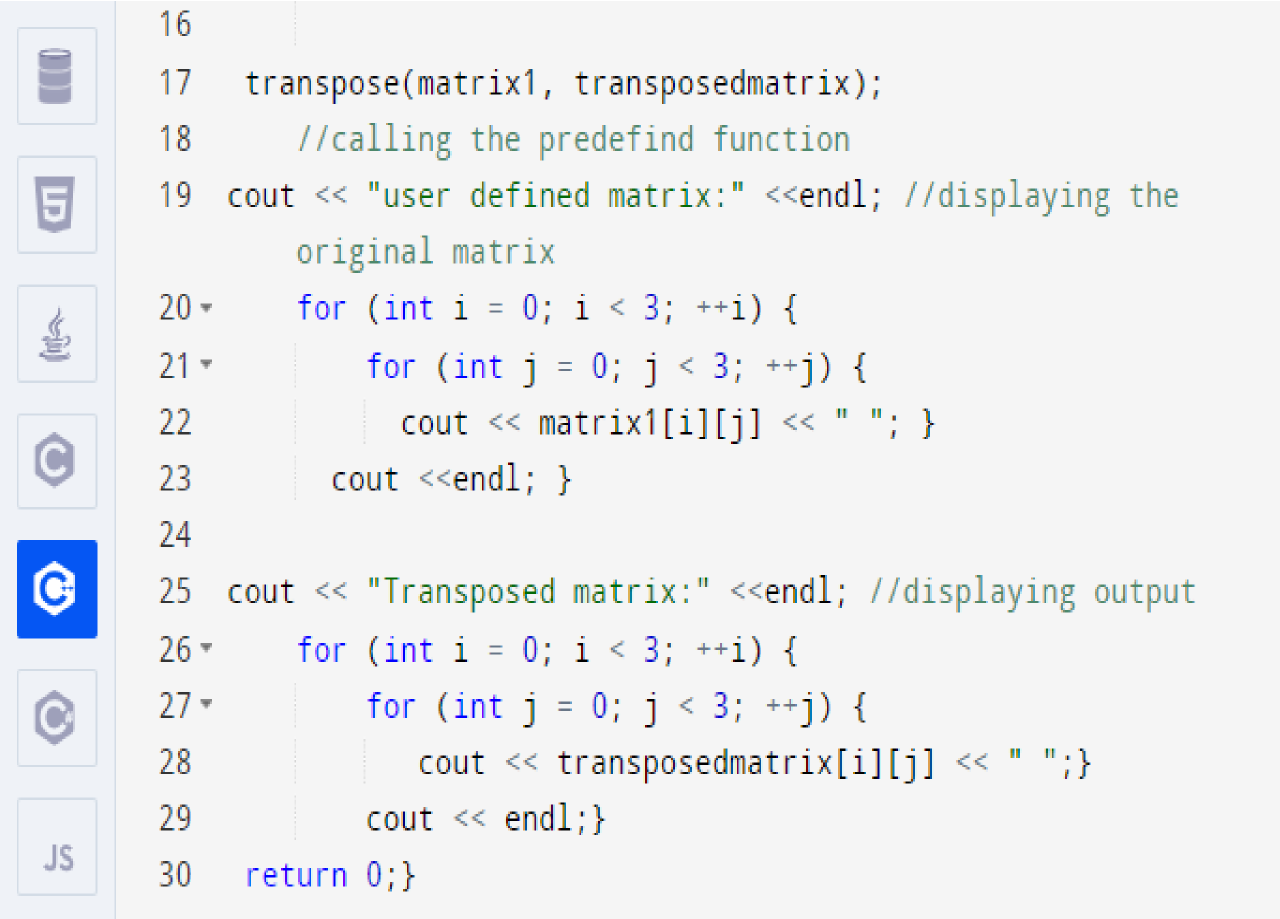
- Using 2D arrays in C++, take transpose of a 3x3 matrix. Make a transpose function.



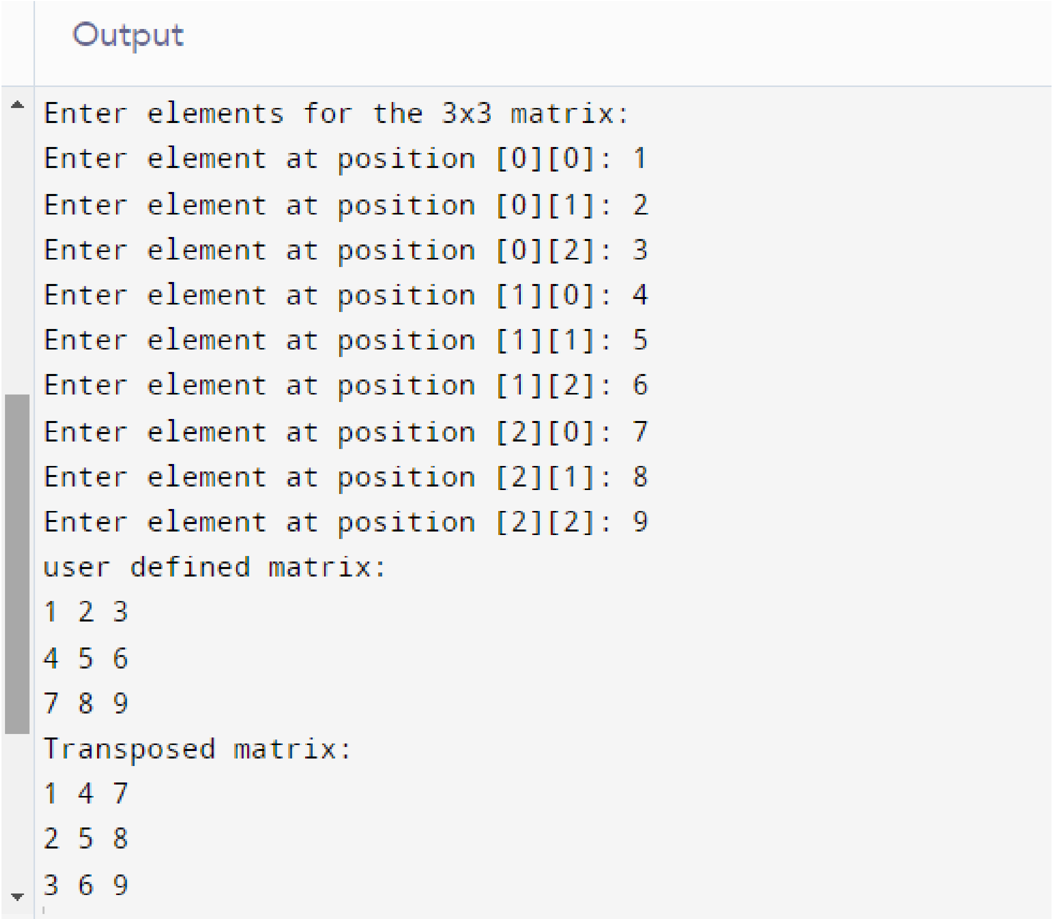
```cpp
    #include <iostream>
    using namespace std;
void // creating a function to find the transpose of a matrix
transpose(int matrix[3][3], int transposematrix[3][3]) {
    for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 3; ++j) {
            transposematrix[j][i] = matrix[i][j]; }}}
        //exchanging the rows with the columns
int main() {
    int matrix1[3][3], transposedmatrix[3][3]; //declaring matrices
cout << "Enter elements for the 3x3 matrix:" <<std::endl;
    for (int i = 0; i < 3; ++i) { // creating a matrix
        for (int j = 0; j < 3; ++j) {
          cout << "Enter element at position [" << i << "][" << j << "]: ";
        cin >> matrix1[i][j]; }} //taking input from users

```

```
Enter elements for the 3x3 matrix:
Enter element at position [0][0]: 1
Enter element at position [0][1]: 2
Enter element at position [0][2]: 3
Enter element at position [1][0]: 4
Enter element at position [1][1]: 5
Enter element at position [1][2]: 6
Enter element at position [2][0]: 7
Enter element at position [2][1]: 8
Enter element at position [2][2]: 9
user defined matrix:
1 2 3
4 5 6
7 8 9
Transposed matrix:
1 4 7
2 5 8
3 6 9
```

TASK # 03 (contd)



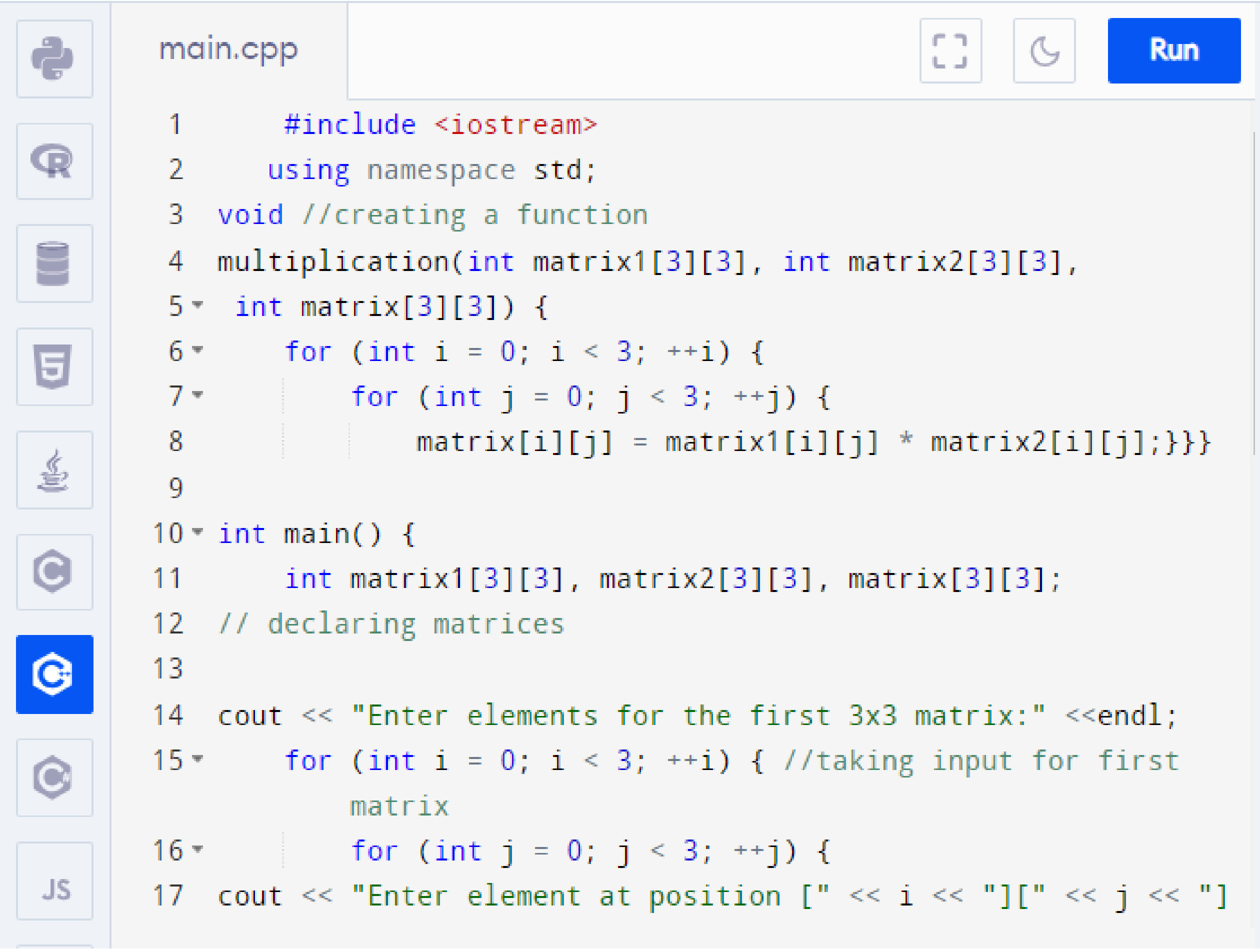
```cpp

 transpose(matrix1, transposedmatrix);
    //calling the predefind function
cout << "user defined matrix:" <<endl; //displaying the original matrix
    for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 3; ++j) {
          cout << matrix1[i][j] << " "; }
      cout <<endl; }

cout << "Transposed matrix:" <<endl; //displaying output
    for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 3; ++j) {
           cout << transposedmatrix[i][j] << " ";}
        cout << endl;}
 return 0;}
```



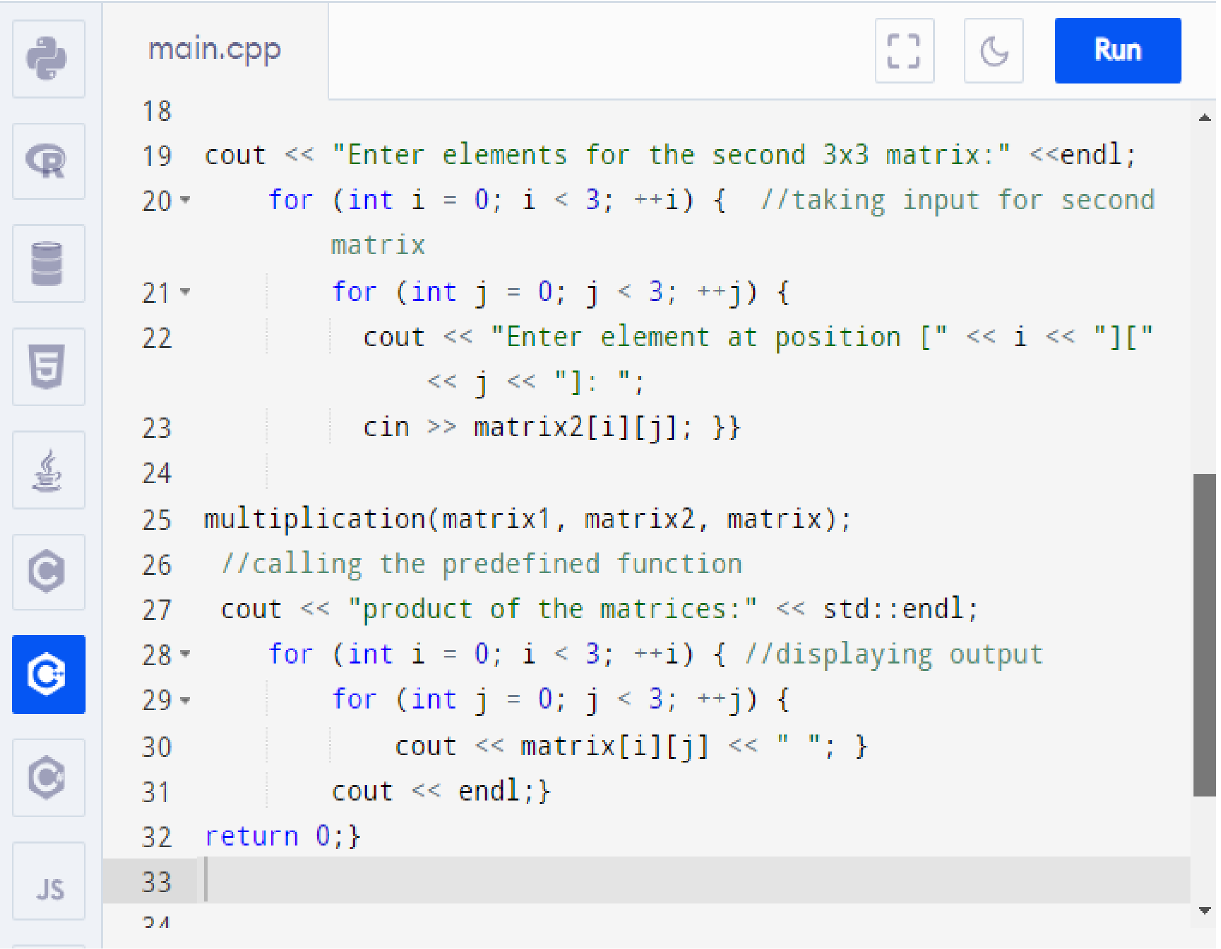
```
Enter elements for the 3x3 matrix:
Enter element at position [0][0]: 1
Enter element at position [0][1]: 2
Enter element at position [0][2]: 3
Enter element at position [1][0]: 4
Enter element at position [1][1]: 5
Enter element at position [1][2]: 6
Enter element at position [2][0]: 7
Enter element at position [2][1]: 8
Enter element at position [2][2]: 9
user defined matrix:
1 2 3
4 5 6
7 8 9
Transposed matrix:
1 4 7
2 5 8
3 6 9
```

TASK # 04

- Using 2D arrays in C++, implement 3x3 matrix multiplication. Make a function.

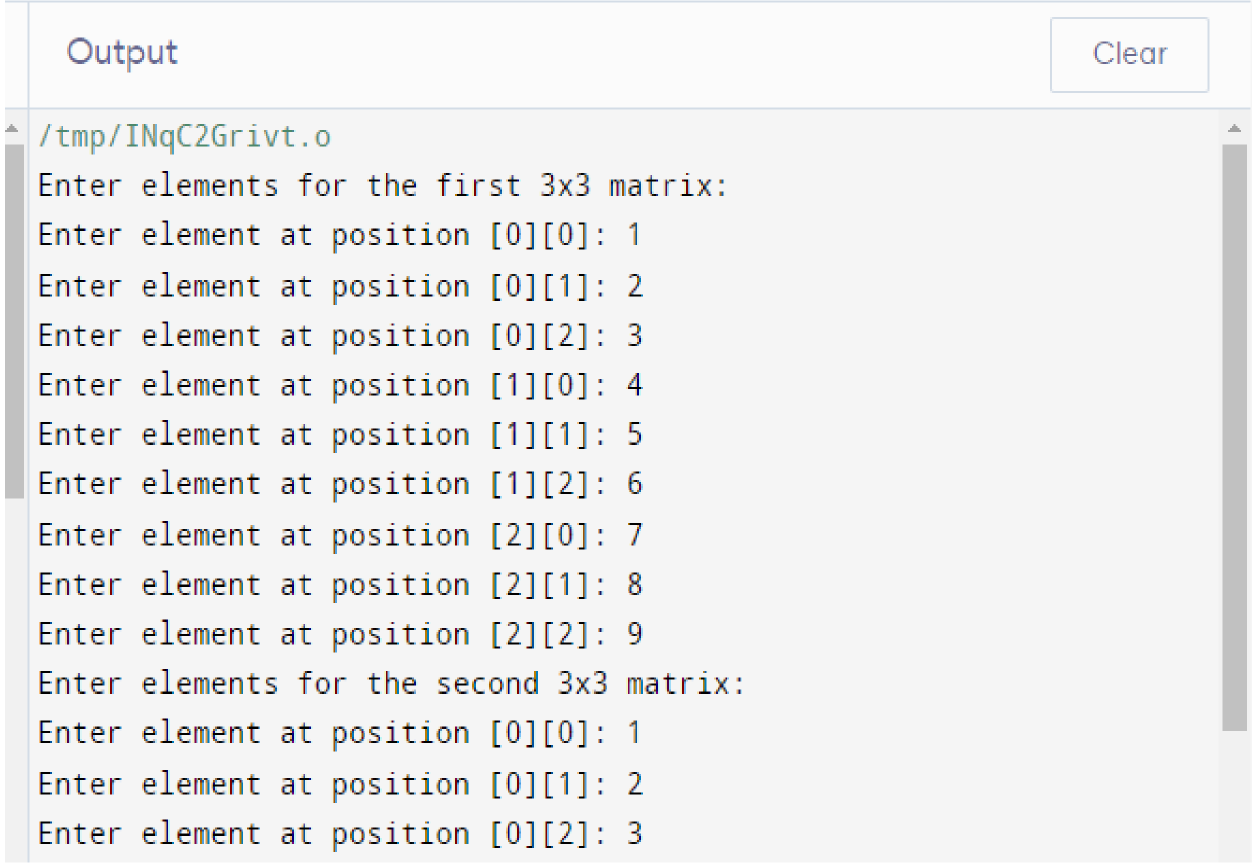


```cpp
    #include <iostream>
   using namespace std;
void //creating a function
multiplication(int matrix1[3][3], int matrix2[3][3],
 int matrix[3][3]) {
    for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 3; ++j) {
            matrix[i][j] = matrix1[i][j] * matrix2[i][j];}}}

int main() {
    int matrix1[3][3], matrix2[3][3], matrix[3][3];
// declaring matrices

cout << "Enter elements for the first 3x3 matrix:" <<endl;
    for (int i = 0; i < 3; ++i) { //taking input for first matrix
        for (int j = 0; j < 3; ++j) {
cout << "Enter element at position [" << i << "][" << j << "]
```

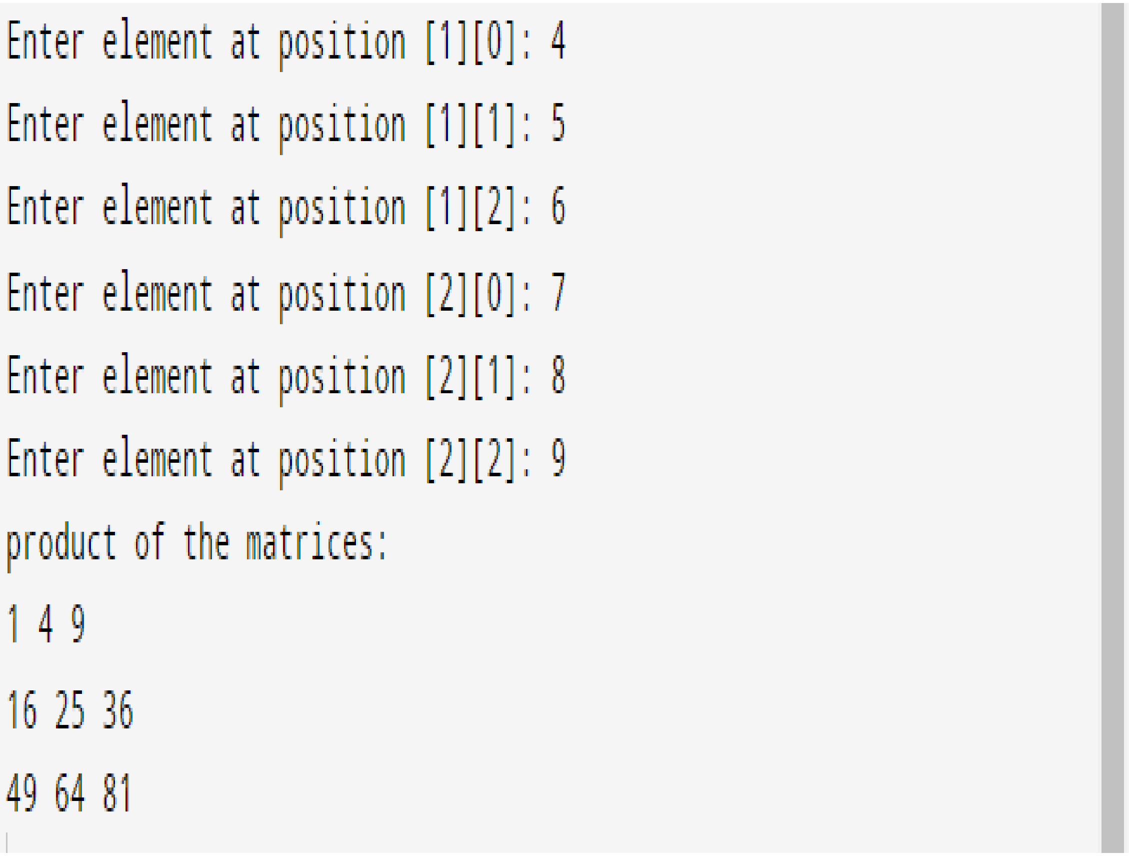


```cpp

cout << "Enter elements for the second 3x3 matrix:" <<endl;
    for (int i = 0; i < 3; ++i) {  //taking input for second matrix
        for (int j = 0; j < 3; ++j) {
         cout << "Enter element at position [" << i << "][" << j << "]: ";
         cin >> matrix2[i][j]; }}

multiplication(matrix1, matrix2, matrix);
 //calling the predefined function
 cout << "product of the matrices:" << std::endl;
    for (int i = 0; i < 3; ++i) { //displaying output
        for (int j = 0; j < 3; ++j) {
            cout << matrix[i][j] << " "; }
        cout << endl;}
return 0;}

```

TASK # 04 (OUTPUT)



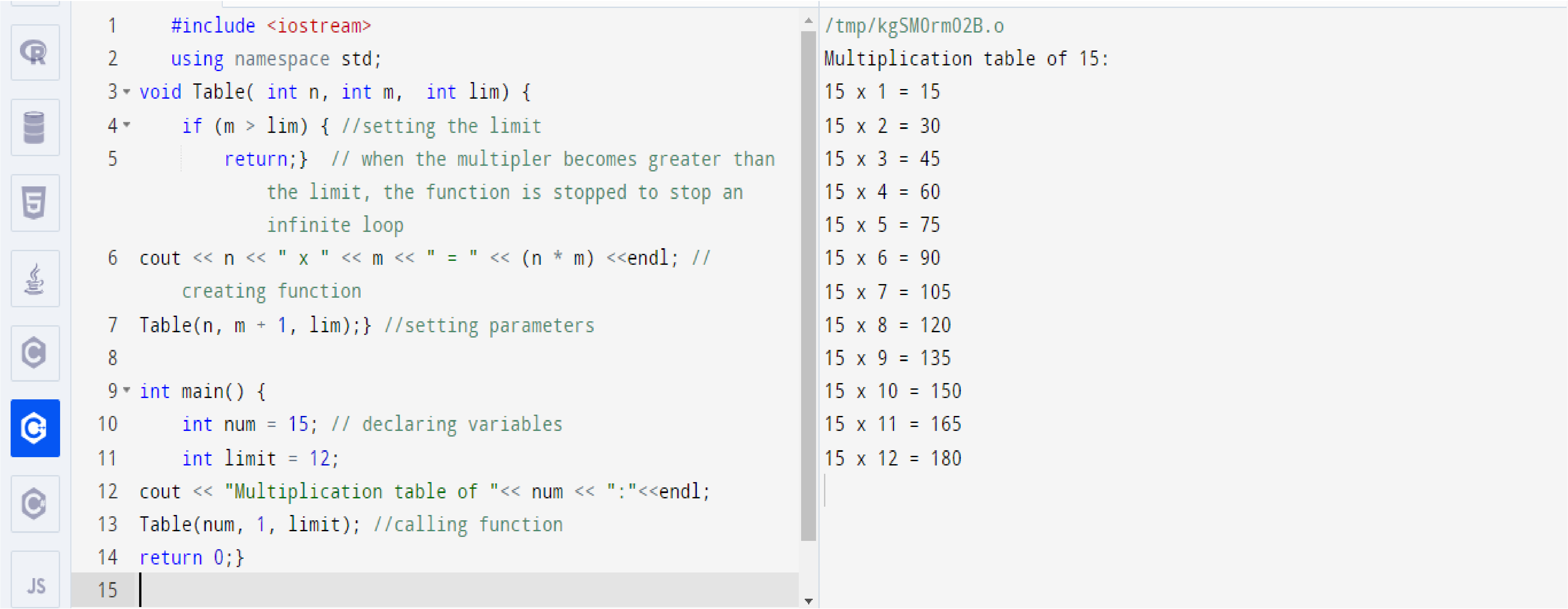
```
/tmp/INqC2Grivt.o
Enter elements for the first 3x3 matrix:
Enter element at position [0][0]: 1
Enter element at position [0][1]: 2
Enter element at position [0][2]: 3
Enter element at position [1][0]: 4
Enter element at position [1][1]: 5
Enter element at position [1][2]: 6
Enter element at position [2][0]: 7
Enter element at position [2][1]: 8
Enter element at position [2][2]: 9
Enter elements for the second 3x3 matrix:
Enter element at position [0][0]: 1
Enter element at position [0][1]: 2
Enter element at position [0][2]: 3
```



```
Enter element at position [1][0]: 4
Enter element at position [1][1]: 5
Enter element at position [1][2]: 6
Enter element at position [2][0]: 7
Enter element at position [2][1]: 8
Enter element at position [2][2]: 9
product of the matrices:
1 4 9
16 25 36
49 64 81
```

TASK # 05

- Print the multiplication table of 15 using recursion.

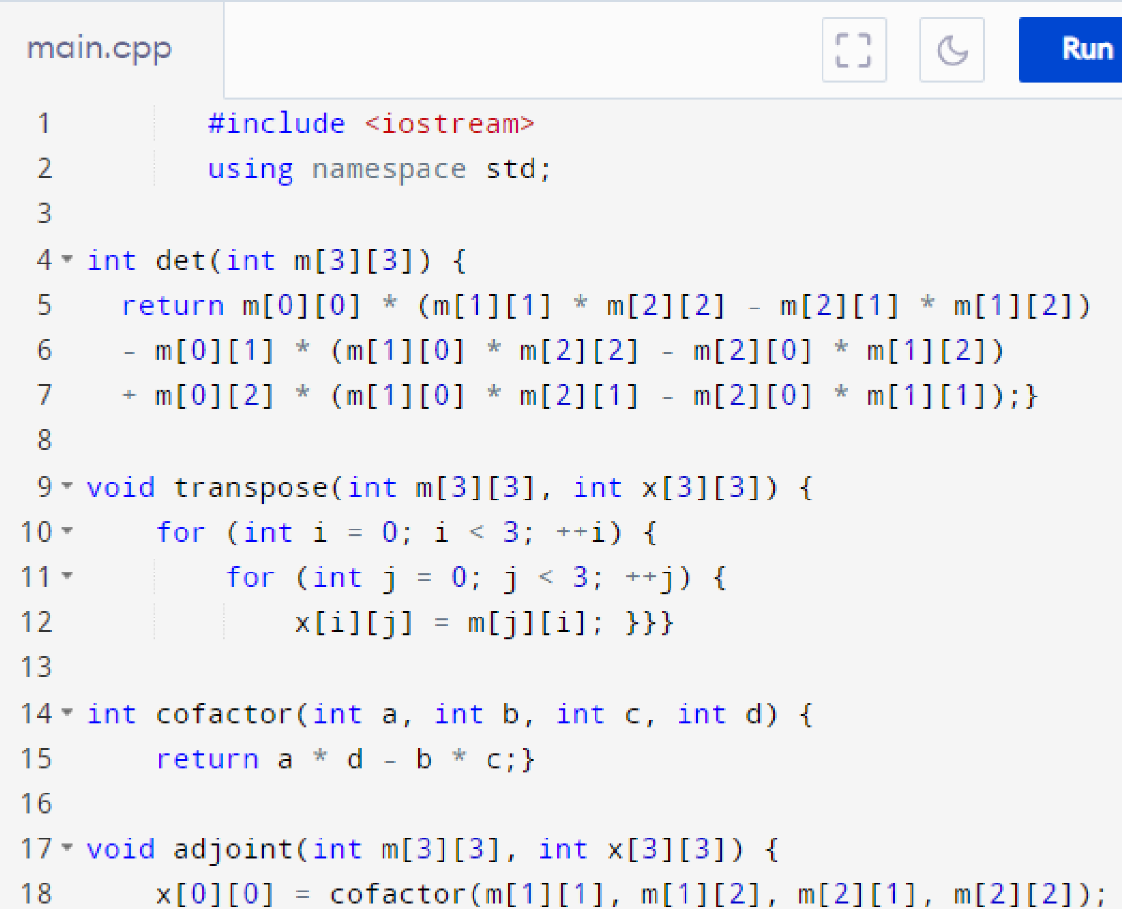


```cpp
   #include <iostream>
   using namespace std;
void Table( int n, int m,  int lim) {
    if (m > lim) { //setting the limit
        return;}  // when the multipler becomes greater than the limit, the function is stopped to stop an infinite loop
cout << n << " x " << m << " = " << (n * m) <<endl; // creating function
Table(n, m + 1, lim);;} //setting parameters

int main() {
    int num = 15; // declaring variables
    int limit = 12;
cout << "Multiplication table of "<< num << ":"<<endl;
Table(num, 1, limit); //calling function
return 0;}
```

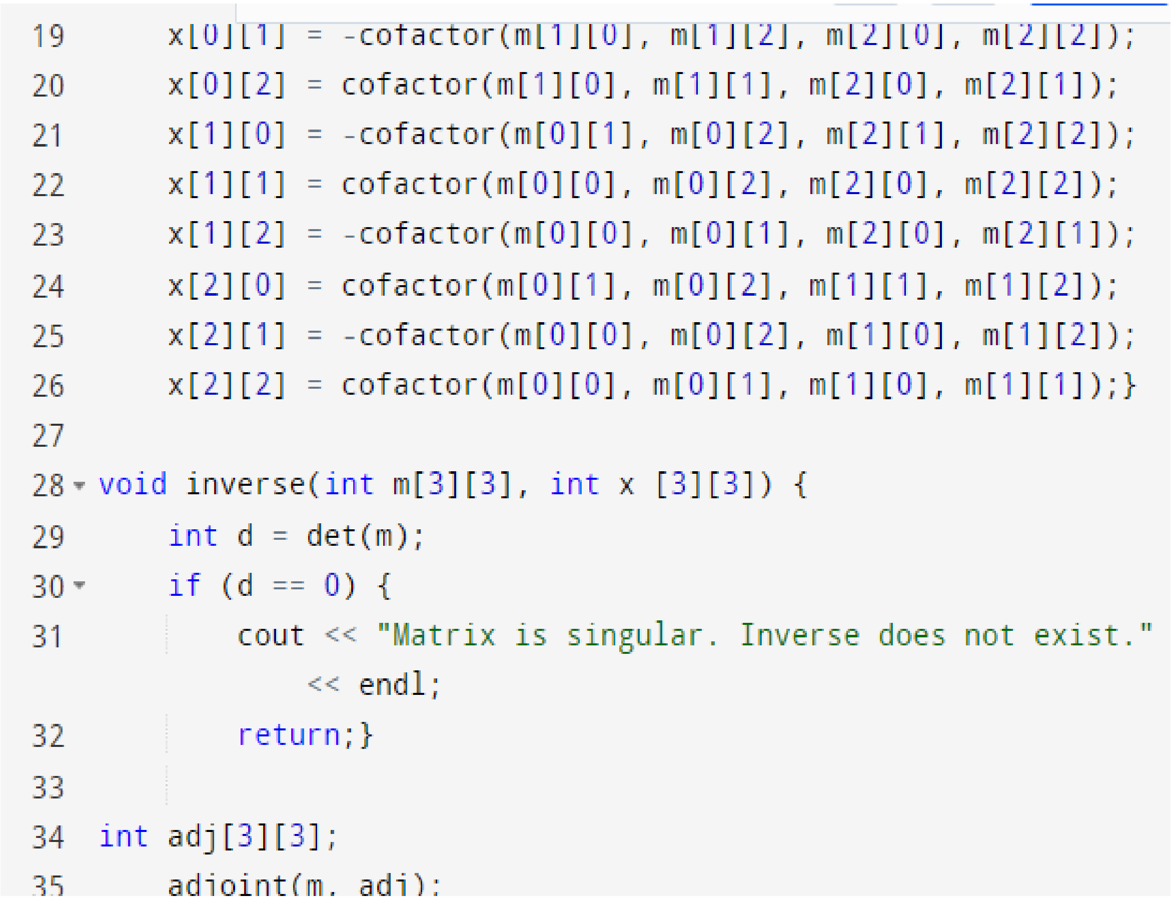
```
/tmp/kgSMOrm02B.o
Multiplication table of 15:
15 x 1 = 15
15 x 2 = 30
15 x 3 = 45
15 x 4 = 60
15 x 5 = 75
15 x 6 = 90
15 x 7 = 105
15 x 8 = 120
15 x 9 = 135
15 x 10 = 150
15 x 11 = 165
15 x 12 = 180
```

TASK # 01 (HOME)

- Write a C++ program to take inverse of a 3x3 matrix using its determinant and adjoint.

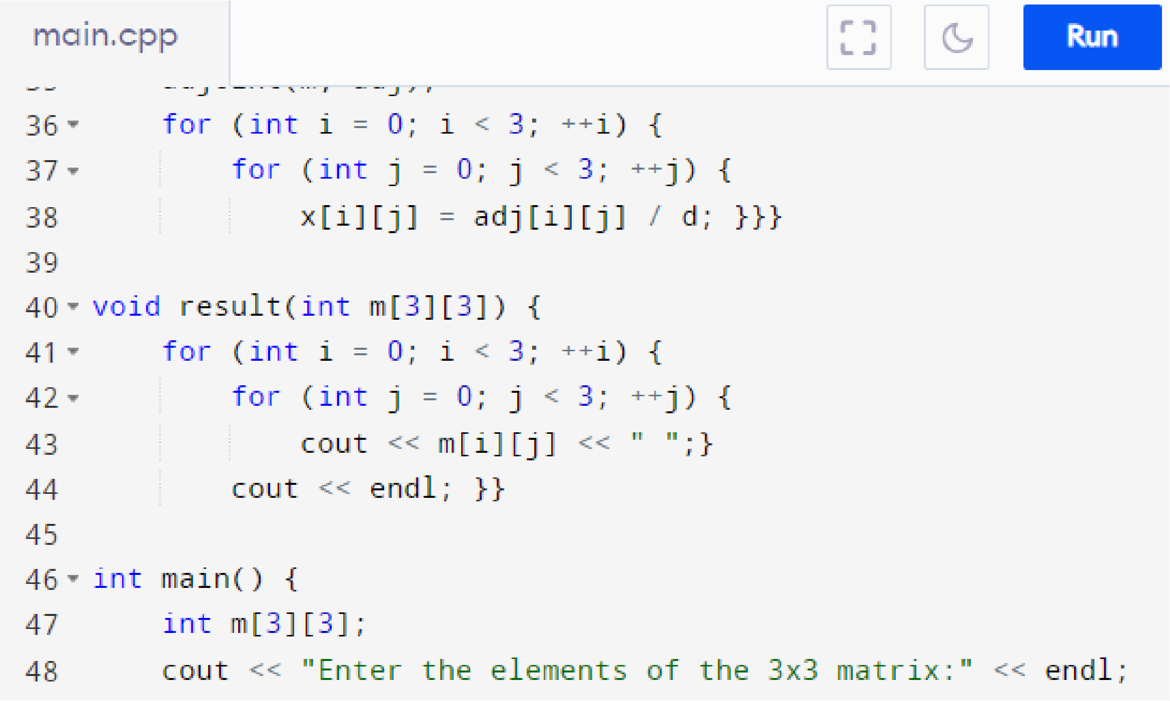


```cpp
        #include <iostream>
        using namespace std;

int det(int m[3][3]) {
  return m[0][0] * (m[1][1] * m[2][2] - m[2][1] * m[1][2])
  - m[0][1] * (m[1][0] * m[2][2] - m[2][0] * m[1][2])
  + m[0][2] * (m[1][0] * m[2][1] - m[2][0] * m[1][1]);}

void transpose(int m[3][3], int x[3][3]) {
    for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 3; ++j) {
            x[i][j] = m[j][i]; }}}

int cofactor(int a, int b, int c, int d) {
    return a * d - b * c;}

void adjoint(int m[3][3], int x[3][3]) {
    x[0][0] = cofactor(m[1][1], m[1][2], m[2][1], m[2][2]);
```

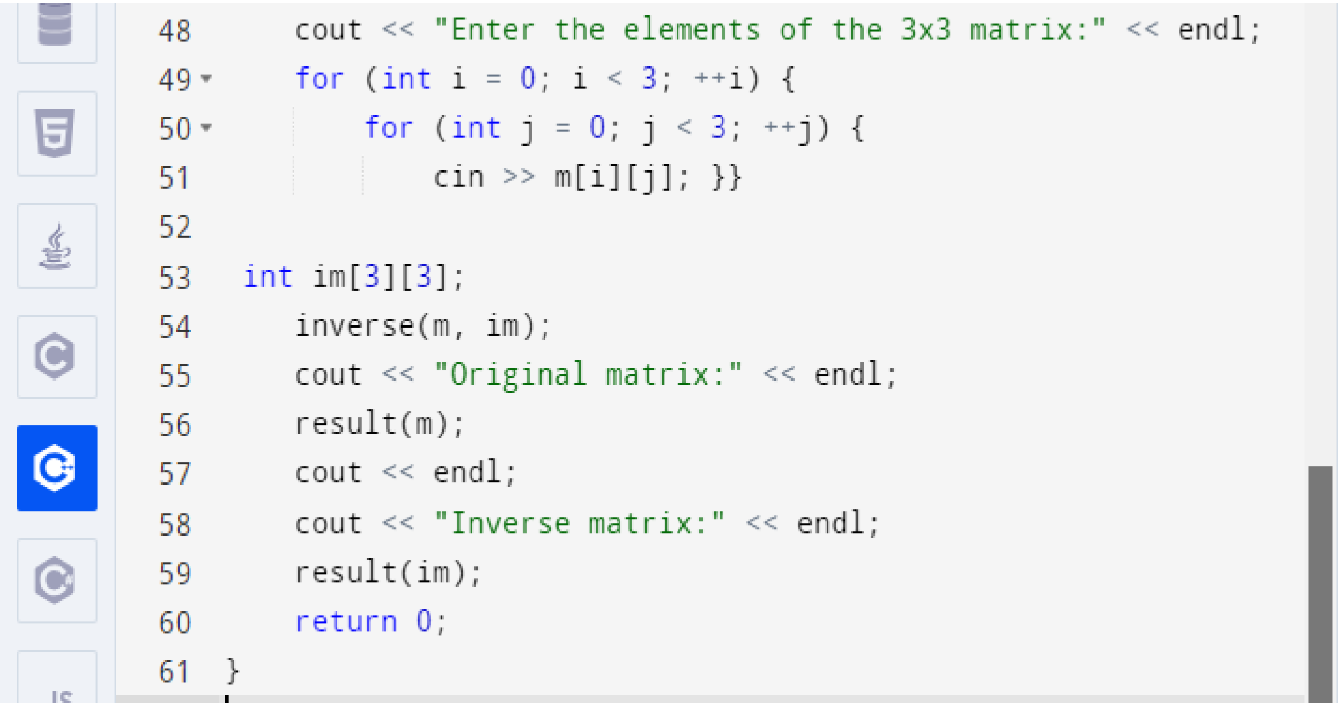


```cpp
    x[0][1] = -cofactor(m[1][0], m[1][2], m[2][0], m[2][2]);
    x[0][2] = cofactor(m[1][0], m[1][1], m[2][0], m[2][1]);
    x[1][0] = -cofactor(m[0][1], m[0][2], m[2][1], m[2][2]);
    x[1][1] = cofactor(m[0][0], m[0][2], m[2][0], m[2][2]);
    x[1][2] = -cofactor(m[0][0], m[0][1], m[2][0], m[2][1]);
    x[2][0] = cofactor(m[0][1], m[0][2], m[1][1], m[1][2]);
    x[2][1] = -cofactor(m[0][0], m[0][2], m[1][0], m[1][2]);
    x[2][2] = cofactor(m[0][0], m[0][1], m[1][0], m[1][1]);}

void inverse(int m[3][3], int x [3][3]) {
    int d = det(m);
    if (d == 0) {
        cout << "Matrix is singular. Inverse does not exist."
            << endl;
        return;}

int adj[3][3];
    adjoint(m, adj);
```

TASK # 01 (contd)

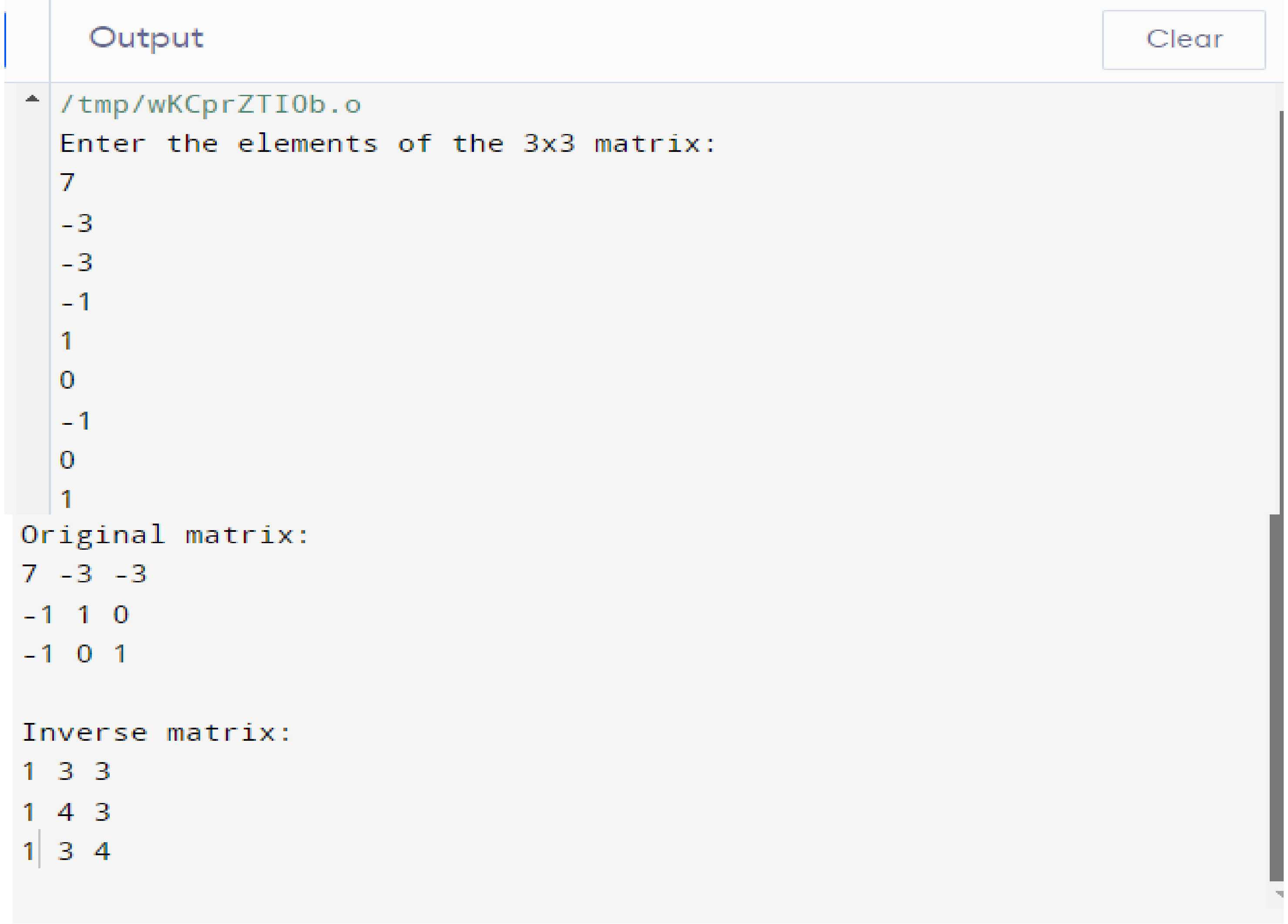


```cpp
    for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 3; ++j) {
            x[i][j] = adj[i][j] / d; }}}

void result(int m[3][3]) {
    for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 3; ++j) {
            cout << m[i][j] << " ";}
        cout << endl; }}

int main() {
    int m[3][3];
    cout << "Enter the elements of the 3x3 matrix:" << endl;
```



```cpp
    cout << "Enter the elements of the 3x3 matrix:" << endl;
    for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 3; ++j) {
            cin >> m[i][j]; }}

 int im[3][3];
    inverse(m, im);
    cout << "Original matrix:" << endl;
    result(m);
    cout << endl;
    cout << "Inverse matrix:" << endl;
    result(im);
    return 0;
}
```

TASK # 01 (OUTPUT)



```
/tmp/wKCprZTIOb.o
Enter the elements of the 3x3 matrix:
7
-3
-3
-1
1
0
-1
0
1
Original matrix:
7 -3 -3
-1 1 0
-1 0 1

Inverse matrix:
1 3 3
1 4 3
1 3 4
```