Name : Nupur Suresh Shinde.

Class : TY-IT-A

Roll No : 54

Batch No : B2

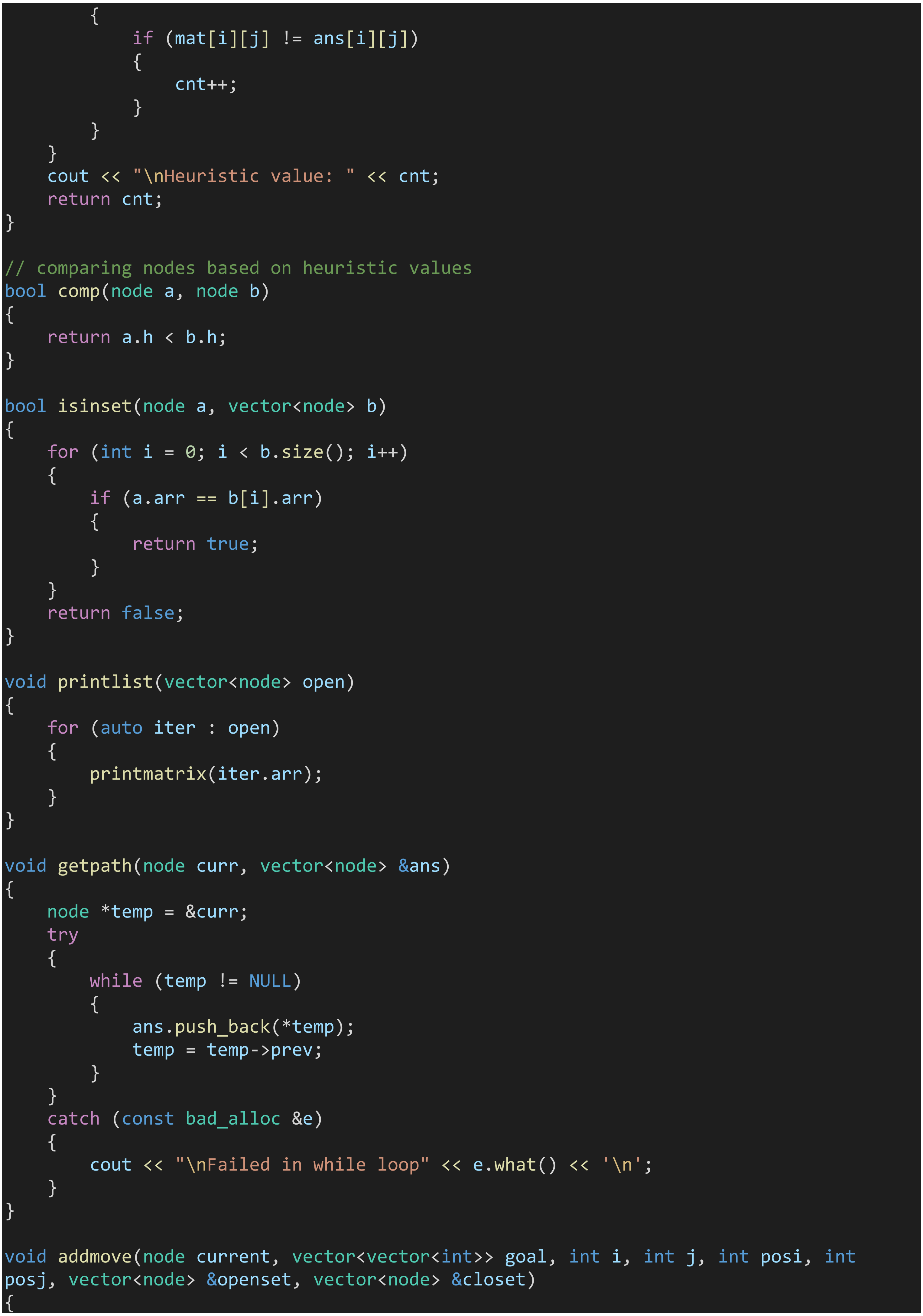
PRN No : 12010610

Lab No : 02 Part B

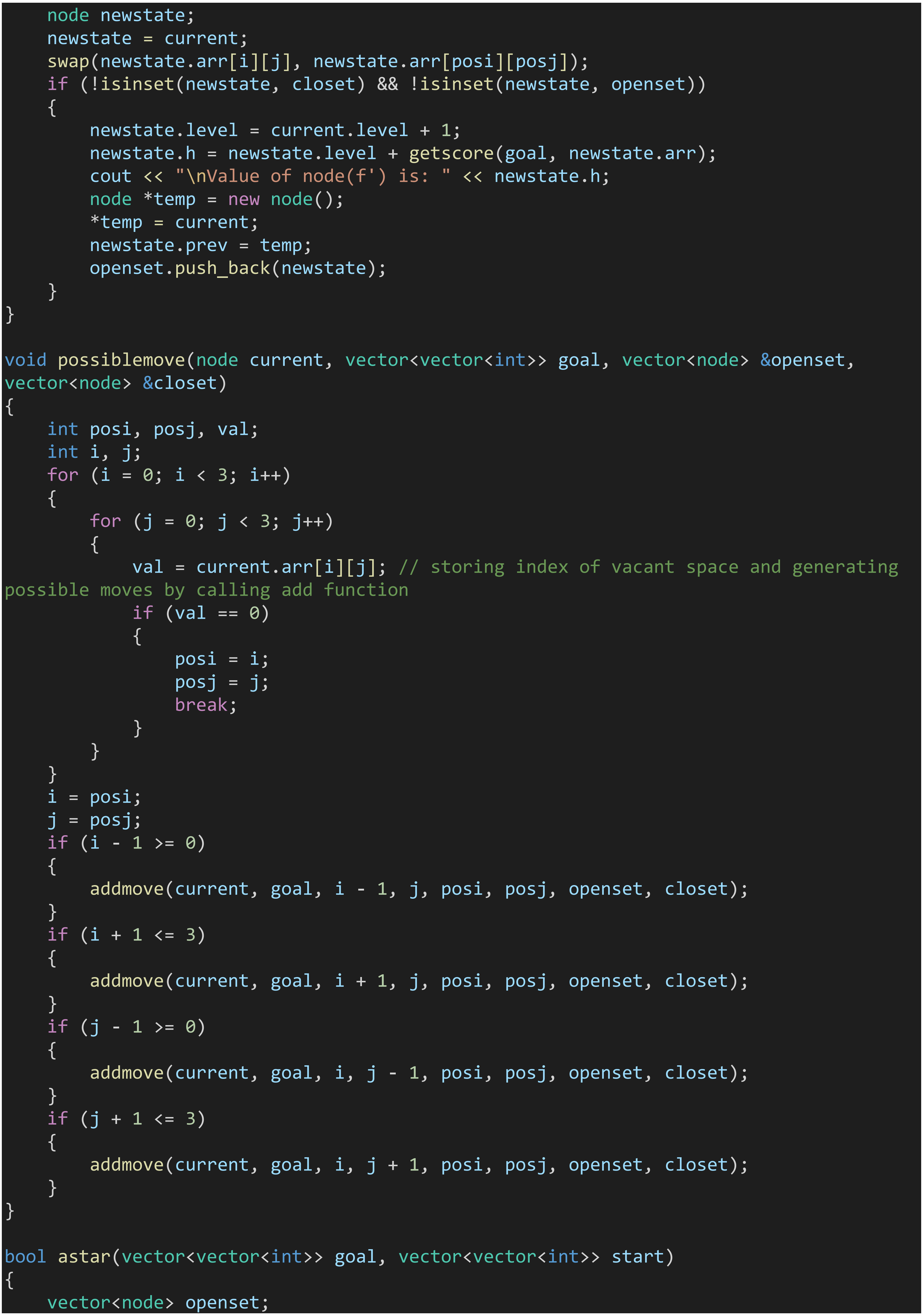
Problem Statement : 8 Puzzle using A* algorithm

Code:

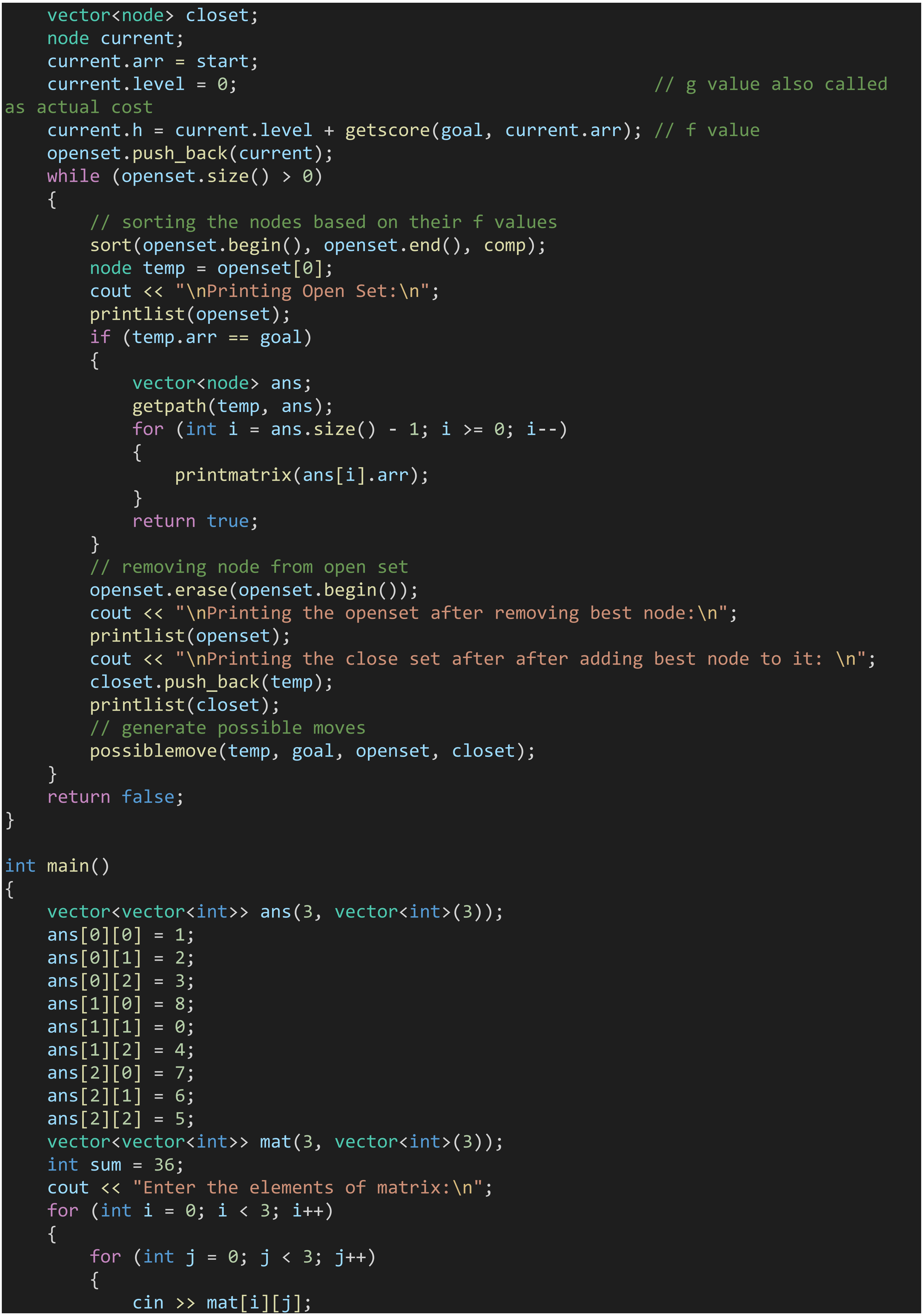
```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

struct node
{
    vector<vector<int>> arr; // matrix
    int level;
    int h;
    node *prev;
    node()
    {
        level = 0;
        prev = NULL;
        h = 0;
    }
};

void printmatrix(vector<vector<int>> mat)
{
    cout << "\nBoard Position:\n";
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            cout << mat[i][j] << "\t";
        }
        cout << endl;
    }
}

// calculate heuristic value
int getscore(vector<vector<int>> &ans, vector<vector<int>> mat)
{
    int cnt;
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
```



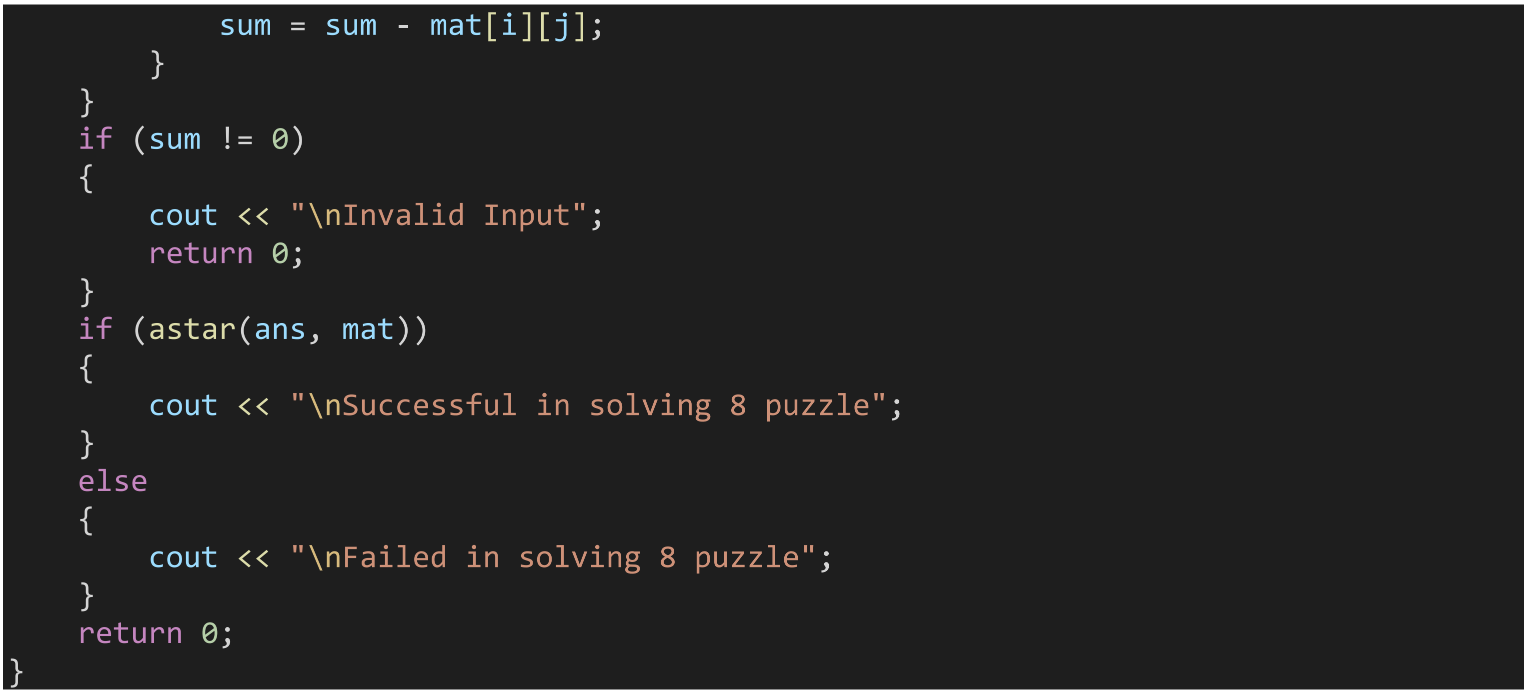
```
        {
            if (mat[i][j] != ans[i][j])
            {
                cnt++;
            }
        }
    }
    cout << "\nHeuristic value: " << cnt;
    return cnt;
}

// comparing nodes based on heuristic values
bool comp(node a, node b)
{
    return a.h < b.h;
}

bool isinset(node a, vector<node> b)
{
    for (int i = 0; i < b.size(); i++)
    {
        if (a.arr == b[i].arr)
        {
            return true;
        }
    }
    return false;
}

void printlist(vector<node> open)
{
    for (auto iter : open)
    {
        printmatrix(iter.arr);
    }
}

void getpath(node curr, vector<node> &ans)
{
    node *temp = &curr;
    try
    {
        while (temp != NULL)
        {
            ans.push_back(*temp);
            temp = temp->prev;
        }
    }
    catch (const bad_alloc &e)
    {
        cout << "\nFailed in while loop" << e.what() << '\n';
    }
}

void addmove(node current, vector<vector<int>> goal, int i, int j, int posi, int
posj, vector<node> &openset, vector<node> &closet)
{
```



```
    node newstate;
    newstate = current;
    swap(newstate.arr[i][j], newstate.arr[posi][posj]);
    if (!isinset(newstate, closet) && !isinset(newstate, openset))
    {
        newstate.level = current.level + 1;
        newstate.h = newstate.level + getscore(goal, newstate.arr);
        cout << "\nValue of node(f') is: " << newstate.h;
        node *temp = new node();
        *temp = current;
        newstate.prev = temp;
        openset.push_back(newstate);
    }
}

void possiblemove(node current, vector<vector<int>> goal, vector<node> &openset,
vector<node> &closet)
{
    int posi, posj, val;
    int i, j;
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
        {
            val = current.arr[i][j]; // storing index of vacant space and generating
possible moves by calling add function
            if (val == 0)
            {
                posi = i;
                posj = j;
                break;
            }
        }
    }
    i = posi;
    j = posj;
    if (i - 1 >= 0)
    {
        addmove(current, goal, i - 1, j, posi, posj, openset, closet);
    }
    if (i + 1 <= 3)
    {
        addmove(current, goal, i + 1, j, posi, posj, openset, closet);
    }
    if (j - 1 >= 0)
    {
        addmove(current, goal, i, j - 1, posi, posj, openset, closet);
    }
    if (j + 1 <= 3)
    {
        addmove(current, goal, i, j + 1, posi, posj, openset, closet);
    }
}

bool astar(vector<vector<int>> goal, vector<vector<int>> start)
{
    vector<node> openset;
```

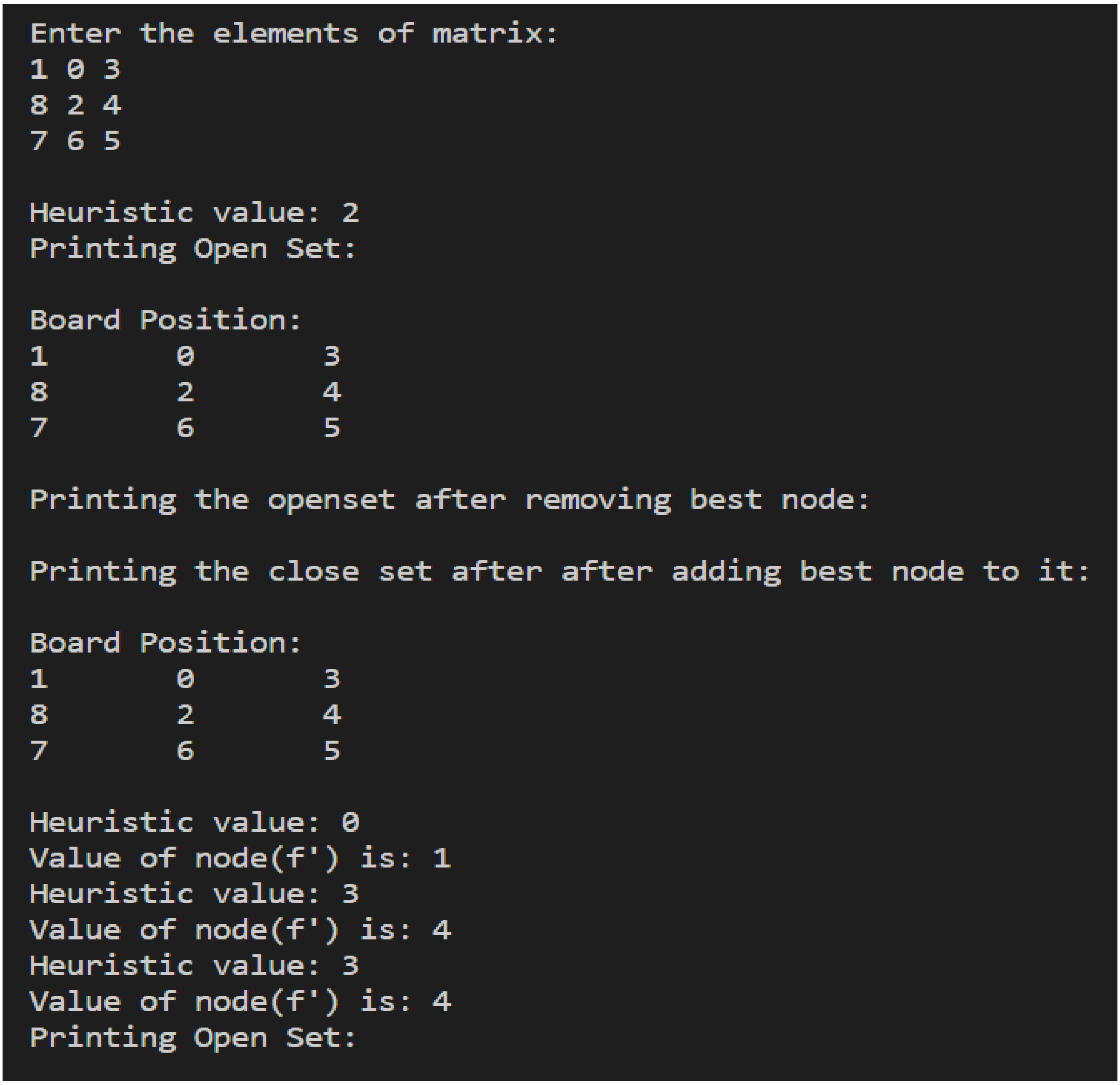


```
    vector<node> closet;
    node current;
    current.arr = start;
    current.level = 0;                                      // g value also called
as actual cost
    current.h = current.level + getscore(goal, current.arr); // f value
    openset.push_back(current);
    while (openset.size() > 0)
    {
        // sorting the nodes based on their f values
        sort(openset.begin(), openset.end(), comp);
        node temp = openset[0];
        cout << "\nPrinting Open Set:\n";
        printlist(openset);
        if (temp.arr == goal)
        {
            vector<node> ans;
            getpath(temp, ans);
            for (int i = ans.size() - 1; i >= 0; i--)
            {
                printmatrix(ans[i].arr);
            }
            return true;
        }
        // removing node from open set
        openset.erase(openset.begin());
        cout << "\nPrinting the openset after removing best node:\n";
        printlist(openset);
        cout << "\nPrinting the close set after after adding best node to it: \n";
        closet.push_back(temp);
        printlist(closet);
        // generate possible moves
        possiblemove(temp, goal, openset, closet);
    }
    return false;
}

int main()
{
    vector<vector<int>> ans(3, vector<int>(3));
    ans[0][0] = 1;
    ans[0][1] = 2;
    ans[0][2] = 3;
    ans[1][0] = 8;
    ans[1][1] = 0;
    ans[1][2] = 4;
    ans[2][0] = 7;
    ans[2][1] = 6;
    ans[2][2] = 5;
    vector<vector<int>> mat(3, vector<int>(3));
    int sum = 36;
    cout << "Enter the elements of matrix:\n";
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            cin >> mat[i][j];
```

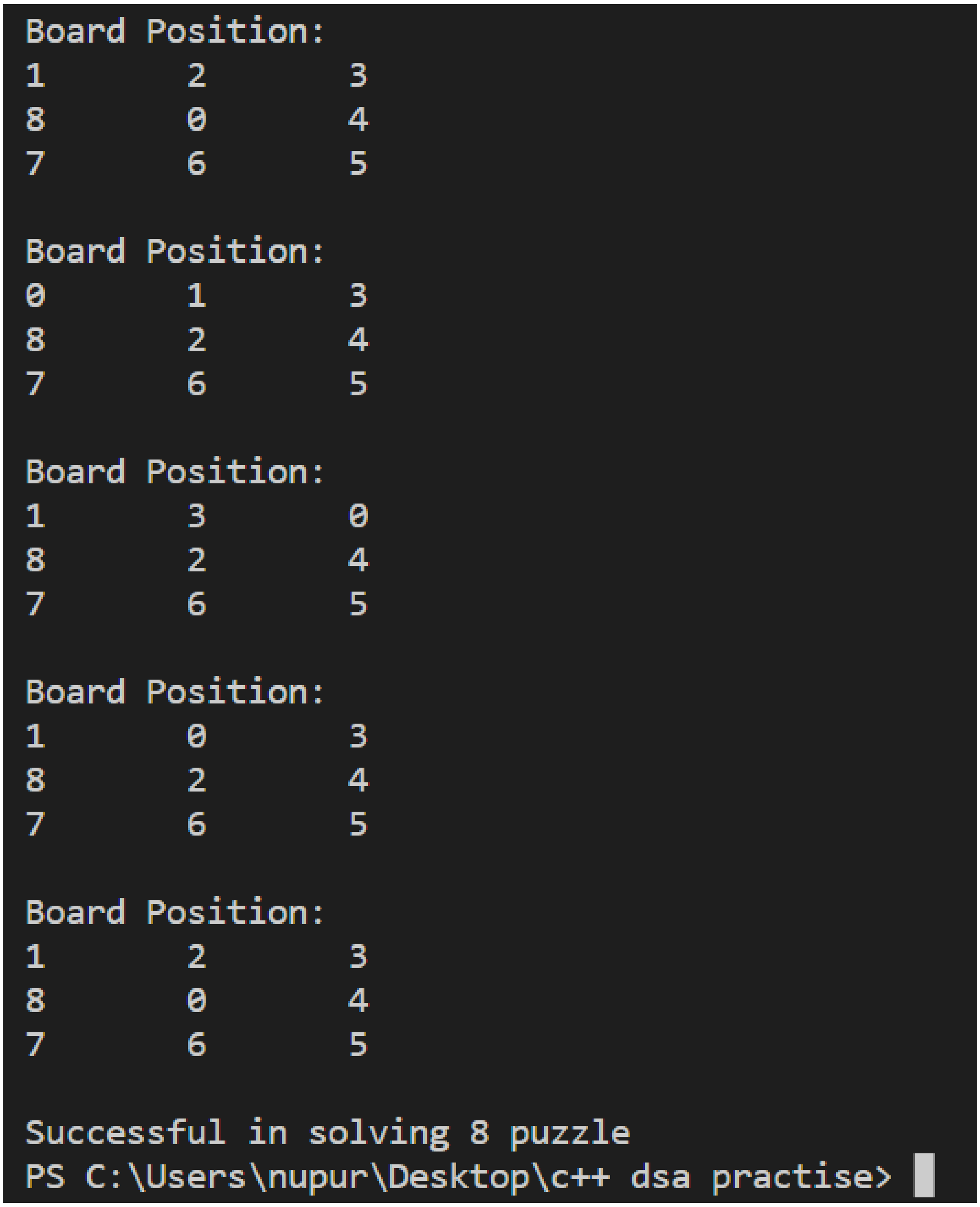


```
            sum = sum - mat[i][j];
        }
    }
    if (sum != 0)
    {
        cout << "\nInvalid Input";
        return 0;
    }
    if (astar(ans, mat))
    {
        cout << "\nSuccessful in solving 8 puzzle";
    }
    else
    {
        cout << "\nFailed in solving 8 puzzle";
    }
    return 0;
}
```

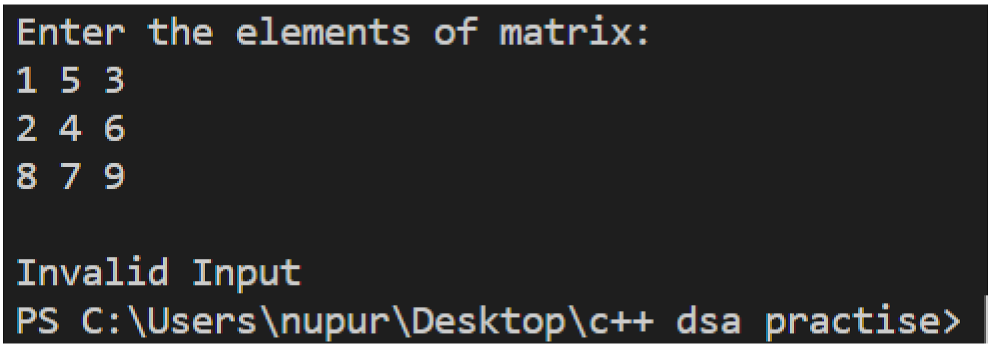
Output:



```
Enter the elements of matrix:
1 0 3
8 2 4
7 6 5

Heuristic value: 2
Printing Open Set:

Board Position:
1       0       3
8       2       4
7       6       5

Printing the openset after removing best node:

Printing the close set after after adding best node to it:

Board Position:
1       0       3
8       2       4
7       6       5

Heuristic value: 0
Value of node(f') is: 1
Heuristic value: 3
Value of node(f') is: 4
Heuristic value: 3
Value of node(f') is: 4
Printing Open Set:
```



```
Board Position:
1       2       3
8       0       4
7       6       5

Board Position:
0       1       3
8       2       4
7       6       5

Board Position:
1       3       0
8       2       4
7       6       5

Board Position:
1       0       3
8       2       4
7       6       5

Board Position:
1       2       3
8       0       4
7       6       5

Successful in solving 8 puzzle
PS C:\Users\nupur\Desktop\c++ dsa practise>
```



```
Enter the elements of matrix:
1 5 3
2 4 6
8 7 9

Invalid Input
PS C:\Users\nupur\Desktop\c++ dsa practise>
```