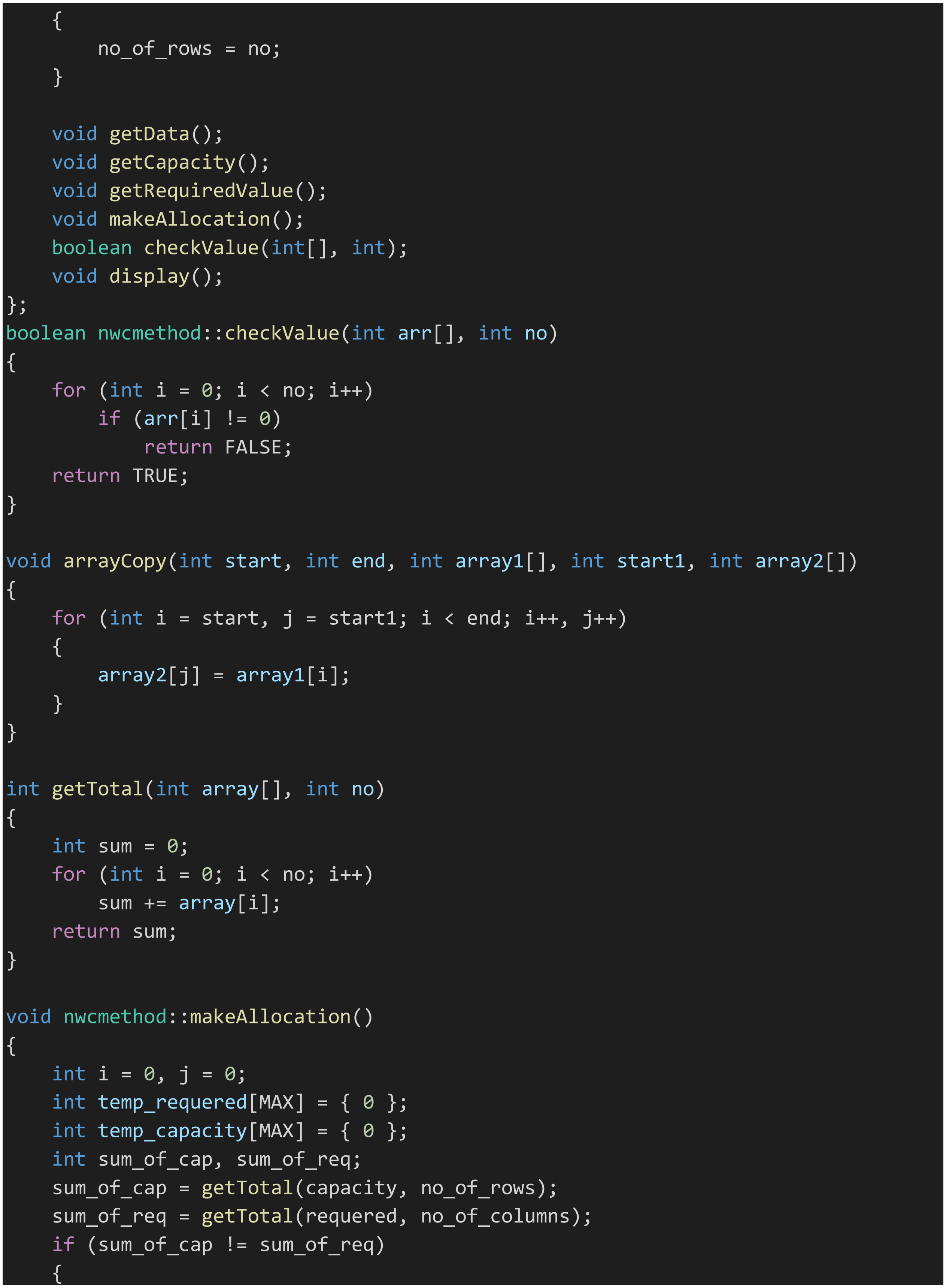
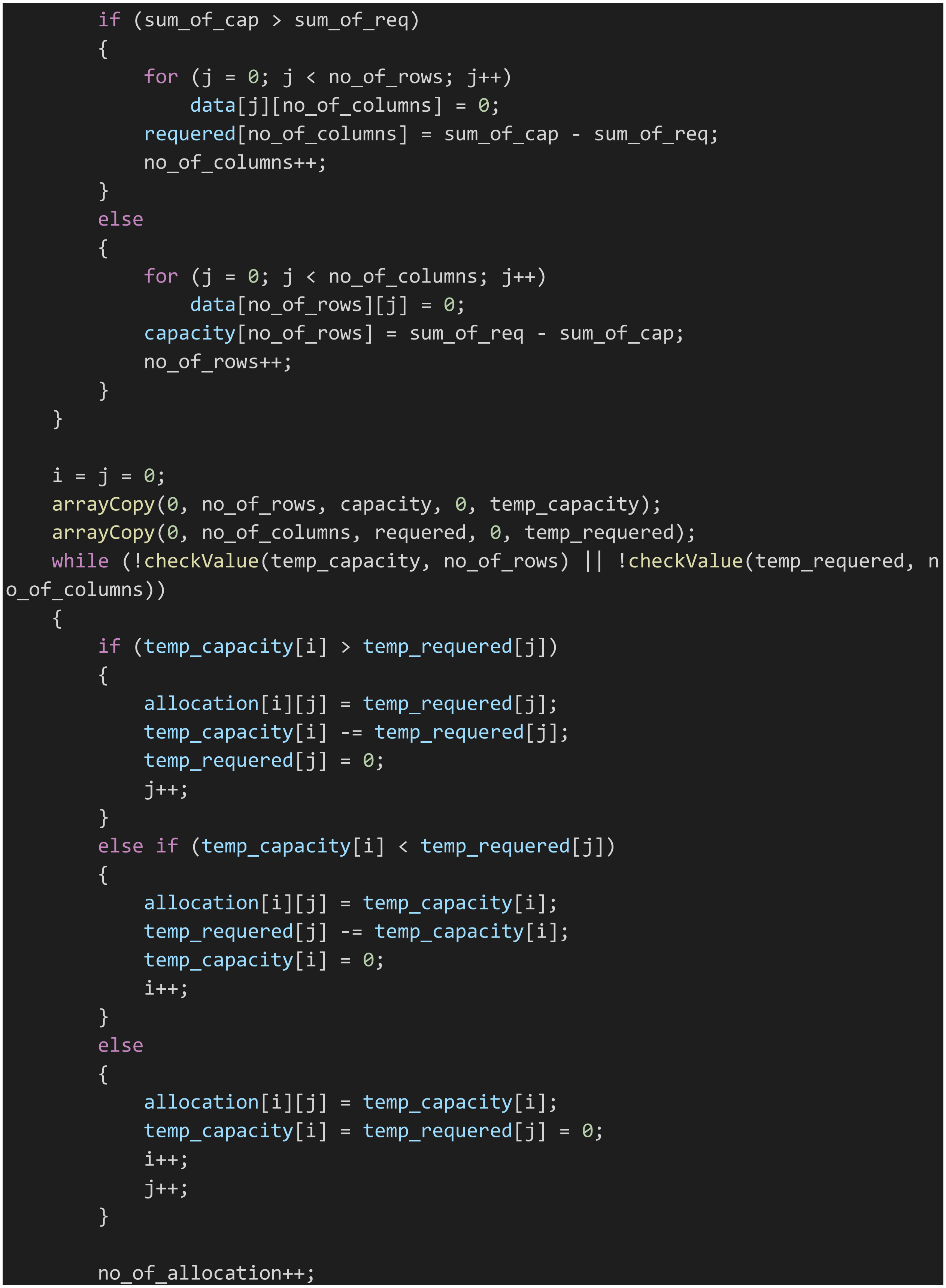
NORTH WEST CORNER METHOD – (C++ LANGUAGE)

CODE

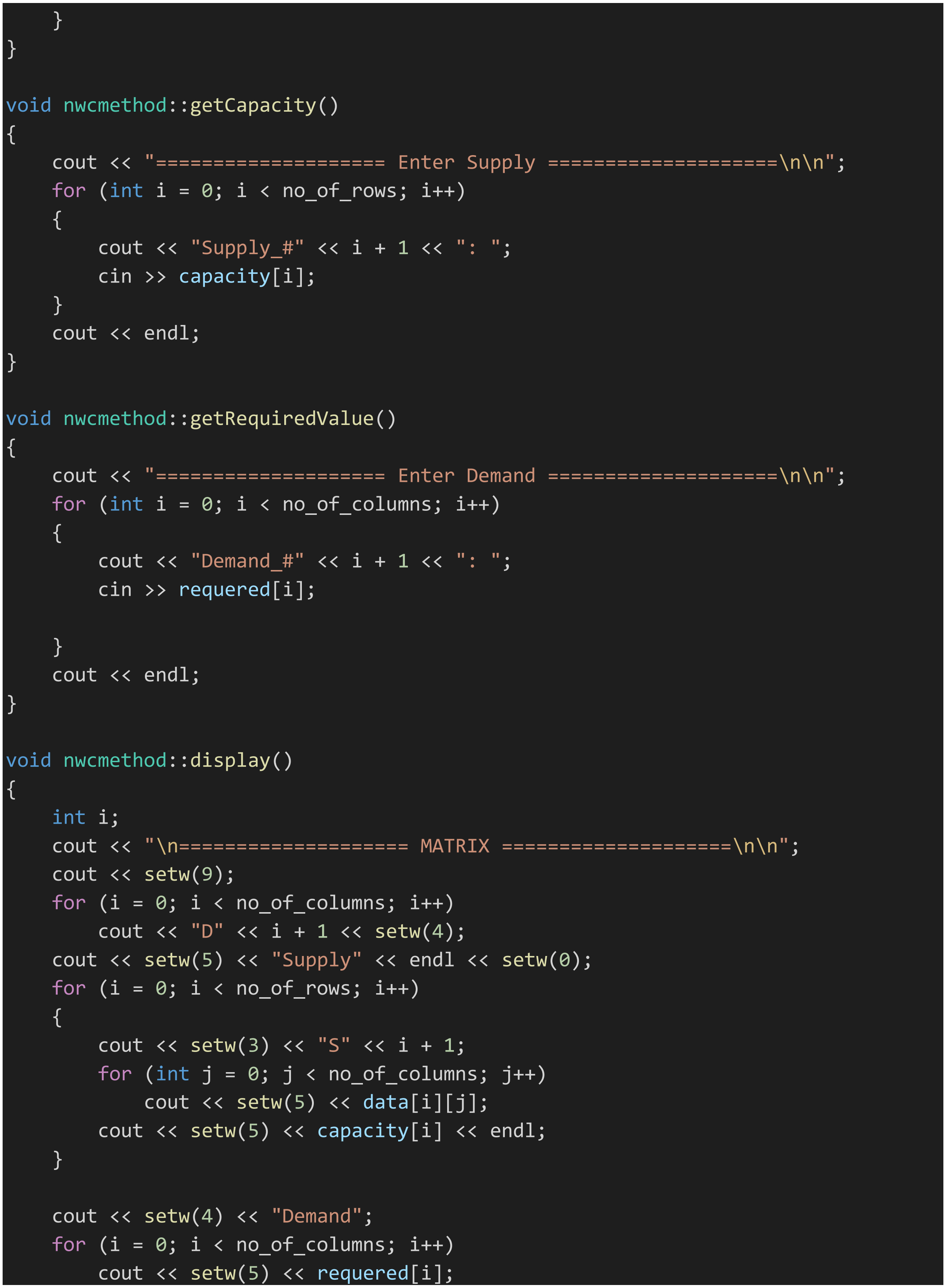
```
#include <iostream>
#include <stdio.h>
#include <conio.h>
#include <iomanip>
#include <stdlib.h>
#define MAX 50
using namespace std;
enum boolean
{
    FALSE,
    TRUE
};
class nwcmethod
{
    int data[MAX][MAX];
    int requered[MAX];
    int capacity[MAX];
    int allocation[MAX][MAX];
    int no_of_rows, no_of_columns, no_of_allocation;
    public:
        nwcmethod()
        {
            for (int i = 0; i < MAX; i++)
            {
                capacity[i] = 0;
                requered[i] = 0;
                for (int j = 0; j < MAX; j++)
                {
                    data[i][j] = 0;
                    allocation[i][j] = 0;
                }
            }

            no_of_rows = no_of_columns = no_of_allocation = 0;
        }

    void setColumn(int no)
    {
        no_of_columns = no;
    };
    void setRow(int no)
```



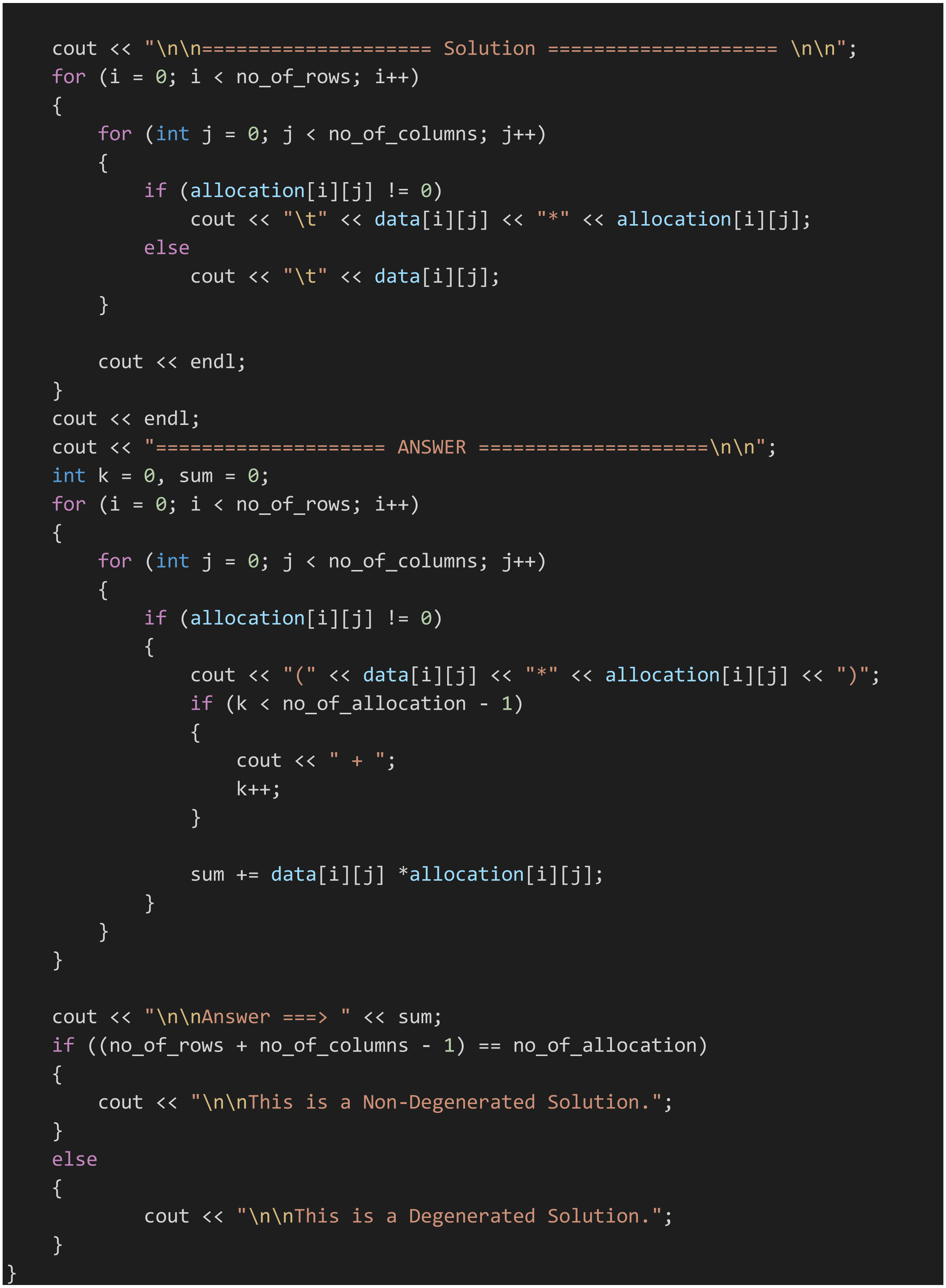
```
    {
        no_of_rows = no;
    }

    void getData();
    void getCapacity();
    void getRequiredValue();
    void makeAllocation();
    boolean checkValue(int[], int);
    void display();
};
boolean nwcmethod::checkValue(int arr[], int no)
{
    for (int i = 0; i < no; i++)
        if (arr[i] != 0)
            return FALSE;
    return TRUE;
}

void arrayCopy(int start, int end, int array1[], int start1, int array2[])
{
    for (int i = start, j = start1; i < end; i++, j++)
    {
        array2[j] = array1[i];
    }
}

int getTotal(int array[], int no)
{
    int sum = 0;
    for (int i = 0; i < no; i++)
        sum += array[i];
    return sum;
}

void nwcmethod::makeAllocation()
{
    int i = 0, j = 0;
    int temp_requered[MAX] = { 0 };
    int temp_capacity[MAX] = { 0 };
    int sum_of_cap, sum_of_req;
    sum_of_cap = getTotal(capacity, no_of_rows);
    sum_of_req = getTotal(requered, no_of_columns);
    if (sum_of_cap != sum_of_req)
    {
```



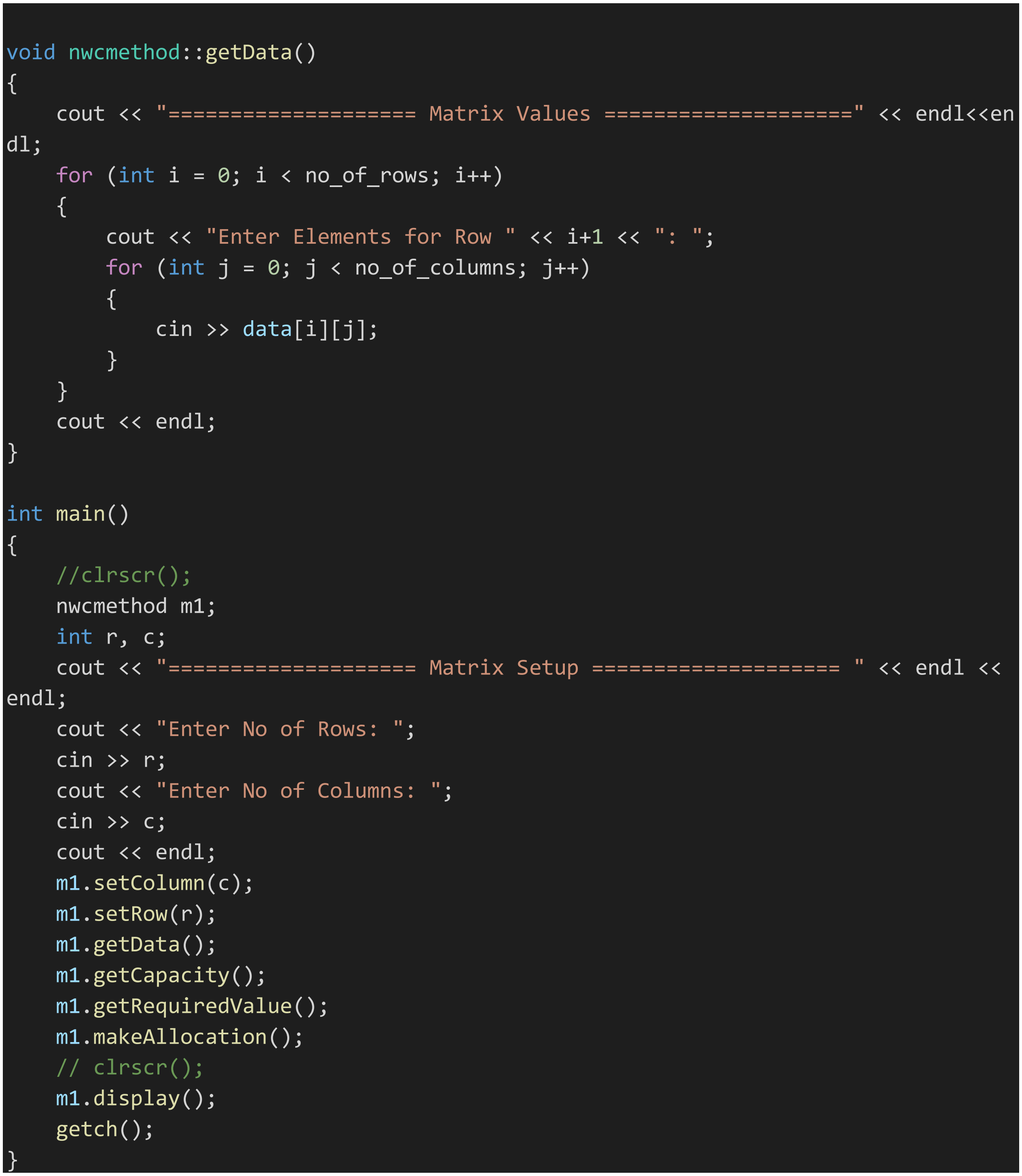
```
        if (sum_of_cap > sum_of_req)
        {
            for (j = 0; j < no_of_rows; j++)
                data[j][no_of_columns] = 0;
            requered[no_of_columns] = sum_of_cap - sum_of_req;
            no_of_columns++;
        }
        else
        {
            for (j = 0; j < no_of_columns; j++)
                data[no_of_rows][j] = 0;
            capacity[no_of_rows] = sum_of_req - sum_of_cap;
            no_of_rows++;
        }
    }

    i = j = 0;
    arrayCopy(0, no_of_rows, capacity, 0, temp_capacity);
    arrayCopy(0, no_of_columns, requered, 0, temp_requered);
    while (!checkValue(temp_capacity, no_of_rows) || !checkValue(temp_requered, n
o_of_columns))
    {
        if (temp_capacity[i] > temp_requered[j])
        {
            allocation[i][j] = temp_requered[j];
            temp_capacity[i] -= temp_requered[j];
            temp_requered[j] = 0;
            j++;
        }
        else if (temp_capacity[i] < temp_requered[j])
        {
            allocation[i][j] = temp_capacity[i];
            temp_requered[j] -= temp_capacity[i];
            temp_capacity[i] = 0;
            i++;
        }
        else
        {
            allocation[i][j] = temp_capacity[i];
            temp_capacity[i] = temp_requered[j] = 0;
            i++;
            j++;
        }

        no_of_allocation++;
```



```
    }
}

void nwcmethod::getCapacity()
{
    cout << "==================== Enter Supply ====================\n\n";
    for (int i = 0; i < no_of_rows; i++)
    {
        cout << "Supply_#" << i + 1 << ": ";
        cin >> capacity[i];
    }
    cout << endl;
}

void nwcmethod::getRequiredValue()
{
    cout << "==================== Enter Demand ====================\n\n";
    for (int i = 0; i < no_of_columns; i++)
    {
        cout << "Demand_#" << i + 1 << ": ";
        cin >> requered[i];

    }
    cout << endl;
}

void nwcmethod::display()
{
    int i;
    cout << "\n==================== MATRIX ====================\n\n";
    cout << setw(9);
    for (i = 0; i < no_of_columns; i++)
        cout << "D" << i + 1 << setw(4);
    cout << setw(5) << "Supply" << endl << setw(0);
    for (i = 0; i < no_of_rows; i++)
    {
        cout << setw(3) << "S" << i + 1;
        for (int j = 0; j < no_of_columns; j++)
            cout << setw(5) << data[i][j];
        cout << setw(5) << capacity[i] << endl;
    }

    cout << setw(4) << "Demand";
    for (i = 0; i < no_of_columns; i++)
        cout << setw(5) << requered[i];
```

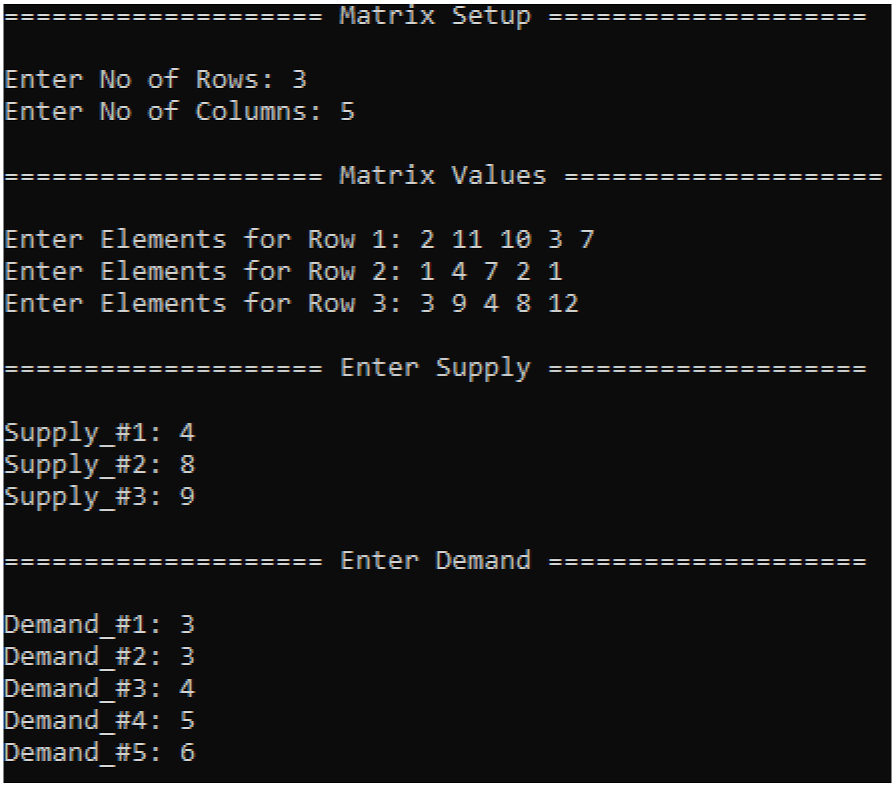


```
    cout << "\n\n==================== Solution ==================== \n\n";
    for (i = 0; i < no_of_rows; i++)
    {
        for (int j = 0; j < no_of_columns; j++)
        {
            if (allocation[i][j] != 0)
                cout << "\t" << data[i][j] << "*" << allocation[i][j];
            else
                cout << "\t" << data[i][j];
        }

        cout << endl;
    }
    cout << endl;
    cout << "==================== ANSWER ====================\n\n";
    int k = 0, sum = 0;
    for (i = 0; i < no_of_rows; i++)
    {
        for (int j = 0; j < no_of_columns; j++)
        {
            if (allocation[i][j] != 0)
            {
                cout << "(" << data[i][j] << "*" << allocation[i][j] << ")";
                if (k < no_of_allocation - 1)
                {
                    cout << " + ";
                    k++;
                }

                sum += data[i][j] *allocation[i][j];
            }
        }
    }

    cout << "\n\nAnswer ===> " << sum;
    if ((no_of_rows + no_of_columns - 1) == no_of_allocation)
    {
        cout << "\n\nThis is a Non-Degenerated Solution.";
    }
    else
    {
            cout << "\n\nThis is a Degenerated Solution.";
    }
}
```

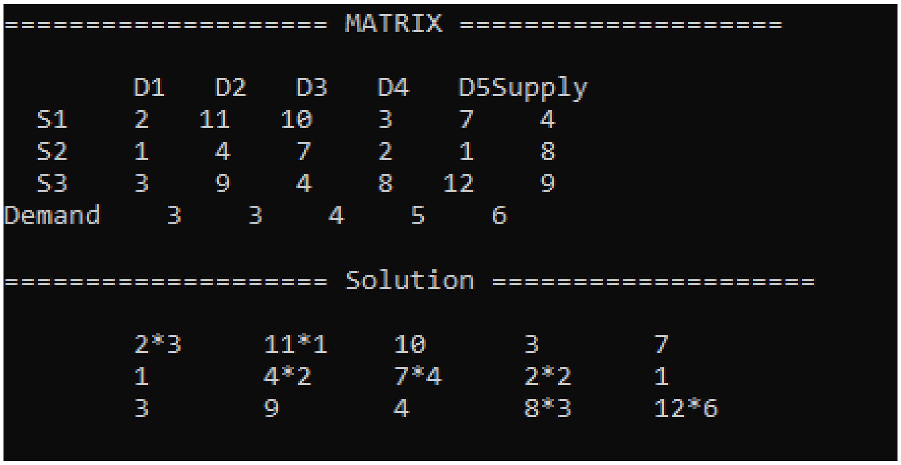


```cpp
void nwcmethod::getData()
{
    cout << "==================== Matrix Values ====================" << endl<<en
dl;
    for (int i = 0; i < no_of_rows; i++)
    {
        cout << "Enter Elements for Row " << i+1 << ": ";
        for (int j = 0; j < no_of_columns; j++)
        {
            cin >> data[i][j];
        }
    }
    cout << endl;
}

int main()
{
    //clrscr();
    nwcmethod m1;
    int r, c;
    cout << "==================== Matrix Setup ==================== " << endl <<
endl;
    cout << "Enter No of Rows: ";
    cin >> r;
    cout << "Enter No of Columns: ";
    cin >> c;
    cout << endl;
    m1.setColumn(c);
    m1.setRow(r);
    m1.getData();
    m1.getCapacity();
    m1.getRequiredValue();
    m1.makeAllocation();
    // clrscr();
    m1.display();
    getch();
}
```

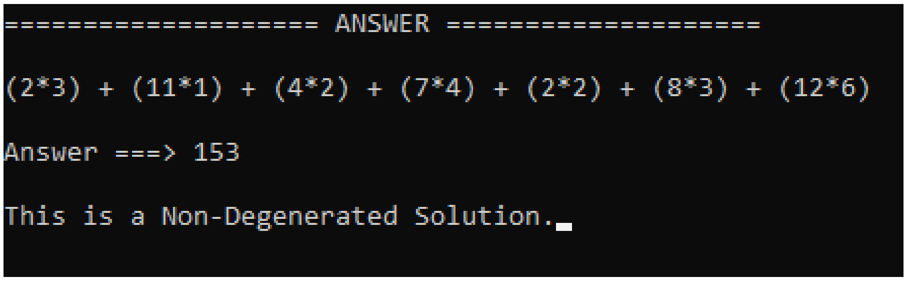
OUTPUT



```
==================== Matrix Setup ====================

Enter No of Rows: 3
Enter No of Columns: 5

==================== Matrix Values ====================

Enter Elements for Row 1: 2 11 10 3 7
Enter Elements for Row 2: 1 4 7 2 1
Enter Elements for Row 3: 3 9 4 8 12

==================== Enter Supply ====================

Supply_#1: 4
Supply_#2: 8
Supply_#3: 9

==================== Enter Demand ====================

Demand_#1: 3
Demand_#2: 3
Demand_#3: 4
Demand_#4: 5
Demand_#5: 6
```



```
==================== MATRIX ====================

        D1   D2   D3   D4   D5Supply
  S1    2   11   10    3    7    4
  S2    1    4    7    2    1    8
  S3    3    9    4    8   12    9
Demand     3    3    4    5    6

==================== Solution ====================

        2*3     11*1    10      3       7
        1       4*2     7*4     2*2     1
        3       9       4       8*3     12*6
```



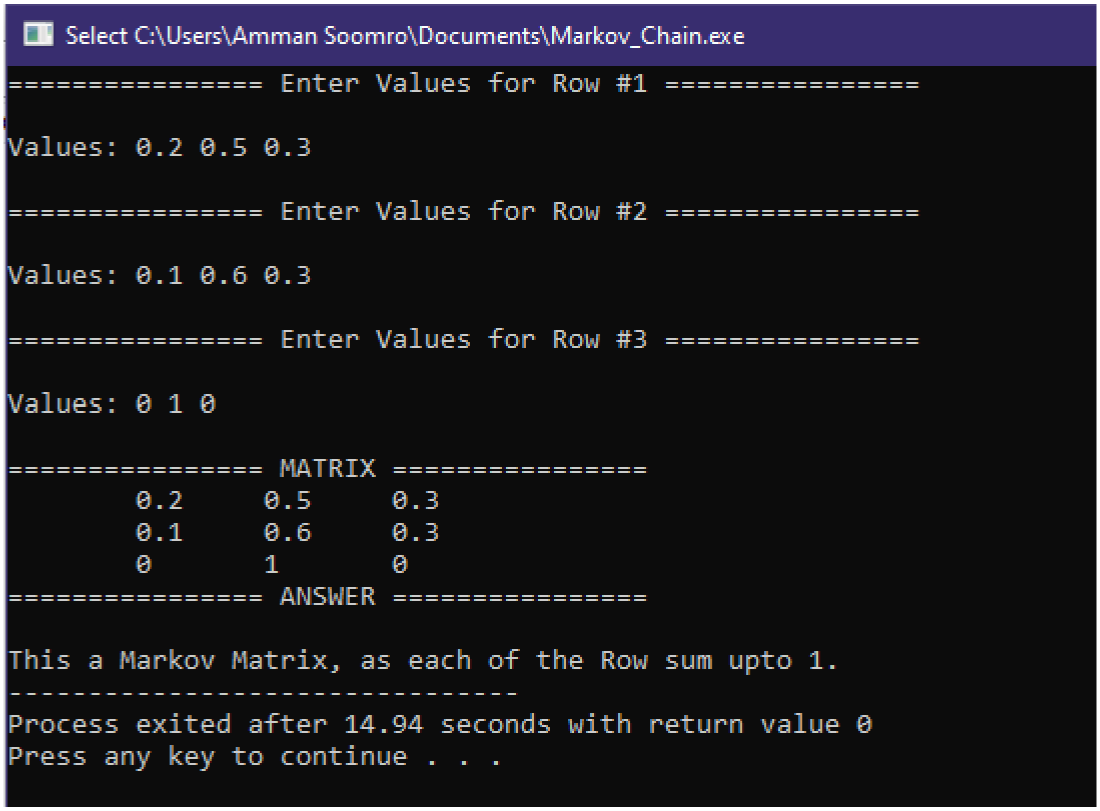
```
==================== ANSWER ====================

(2*3) + (11*1) + (4*2) + (7*4) + (2*2) + (8*3) + (12*6)

Answer ===> 153

This is a Non-Degenerated Solution.
```

MARKOV'S CHAIN - (C++ LANGUAGE)

CODE

```cpp
#include <iostream>
using namespace std;
#define n 3
bool checkMarkov(double m[][n])
{
    for (int i = 0; i <n; i++) {
        double sum = 0;
        for (int j = 0; j < n; j++){
        sum = sum + m[i][j];}
        if (sum != 1)
        return false;
    }
    return true;}
int main(){
    double m[3][3];
    for(int i=0; i<3;i++){
        cout << "================ Enter Values for Row #" << i+1 <<" ================" <<endl;
        cout << "\nValues: ";
        for(int j=0; j<3; j++){
            cin >> m[i][j];}
        cout << endl;}
    cout << "================ MATRIX ================ " <<endl;
    cout <<"\t";
     for(int i=0; i<3;i++)
    {
        for(int j=0; j<3; j++)
        {
            cout << m[i][j] <<"\t";
        }
        cout << endl;
        if(i!=2)
        {
        cout <<"\t";
        }
    }
    cout << "================ ANSWER ================ " <<endl;
    if (checkMarkov(m)){
        cout << "\nThis a Markov Matrix, as each of the Row sum upto 1.";}
    else{
        cout << "\nThis is not a Markov Matrix.";}}
```

OUTPUT



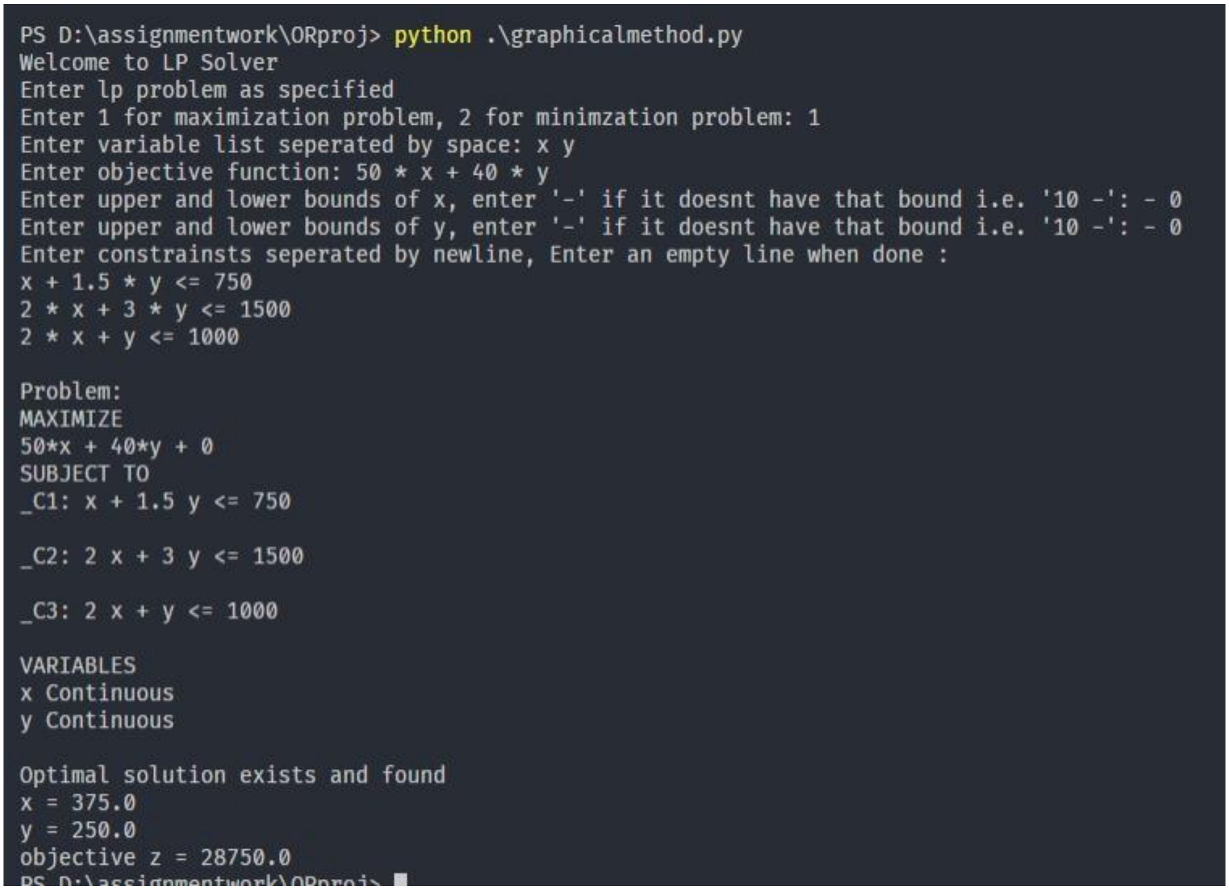
```
================ Enter Values for Row #1 ================

Values: 0.2 0.5 0.3

================ Enter Values for Row #2 ================

Values: 0.1 0.6 0.3

================ Enter Values for Row #3 ================

Values: 0 1 0

================ MATRIX ================
        0.2     0.5     0.3
        0.1     0.6     0.3
        0       1       0
================ ANSWER ================

This a Markov Matrix, as each of the Row sum upto 1.
--------------------------------
Process exited after 14.94 seconds with return value 0
Press any key to continue . . .
```

LPP SOLVER WITH OPTIMAL SOLUTION (PTYHON LANGUAGE)

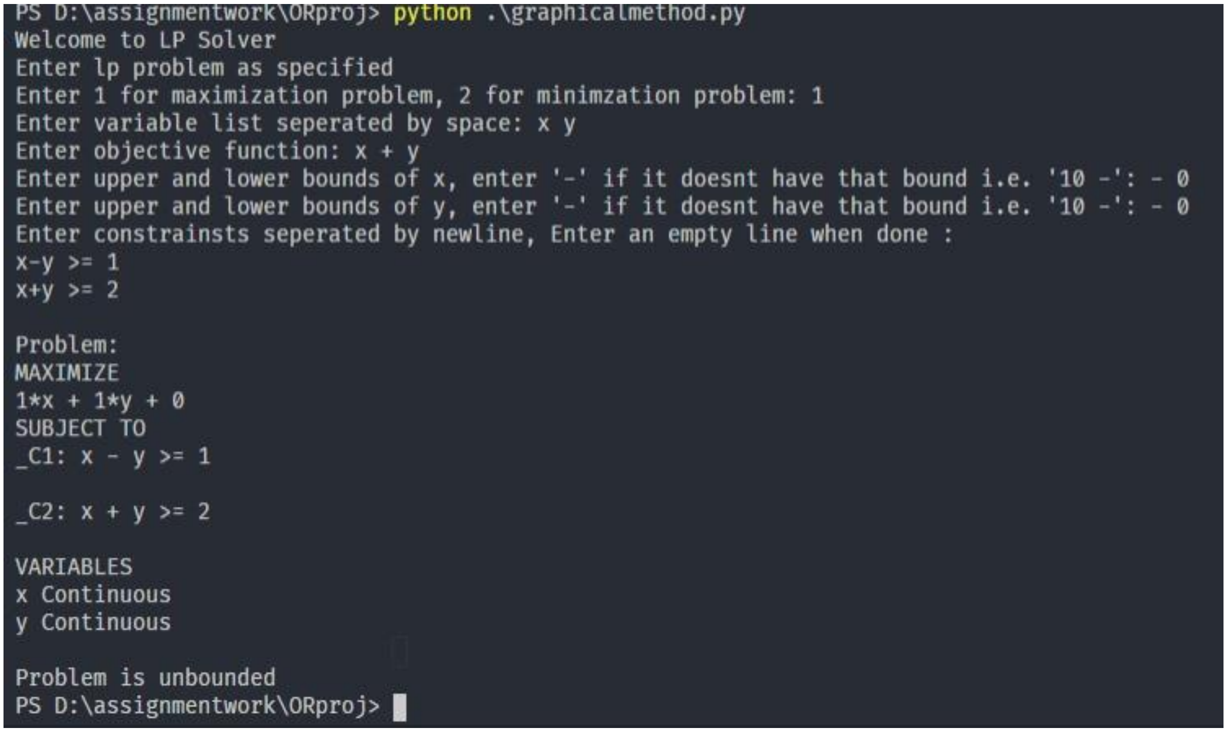
CODE

```
import pulp as p
import string
import re

from pulp.apis.coin_api import PULP_CBC_CMD

allowed_words = list(string.ascii_lowercase)

def string2func(string):
    for word in re.findall('[a-zA-Z_]+', string):
        if word not in allowed_words:
            raise ValueError(
                '"{}" is forbidden to use in math expression'.format(word)
            )

    def func(dict):
        for key in list(dict.keys()):
            lcl = locals()
            lcl[key] = dict[key]
            if key == list(dict.keys())[-1]:
                return eval(string)

    return func


print("Welcome to LP Solver\nEnter lp problem as specified")
probSelection = int(input("Enter 1 for maximization problem, 2 for minimzation problem: "))
varlist = input("Enter variable list seperated by space: ").split(" ")
objectiveFunc = string2func(input("Enter objective function: "))
bounds = dict()
for var in varlist:
    upper, lower = [(float(i) if i != '-' else None) for i in input("Enter upper and lower bounds of {}, enter '-' if it doesnt have that bound i.e. '10 -': ".format(var)).split(" ")]
    bounds[var] = [upper, lower]
print('Enter constrainsts seperated by newline, Enter an empty line when done : ')
constraints = list()
while(True):
    inp = input()
    if inp == "":
        break
    constraints.append(string2func(inp))

Lp_prob = p.LpProblem('Problem', p.LpMaximize if probSelection == 1 else p.LpMinimize)

lpvars = dict()
for var in varlist:
    lpvars[var] = p.LpVariable(var, upBound = bounds[var][0] if bounds[var][0] != None else None,lowBound = bounds[var][1] if bounds[var][1] != None else None)

Lp_prob += objectiveFunc(lpvars)

for constraint in constraints:
    Lp_prob += constraint(lpvars)

print(Lp_prob)

status = Lp_prob.solve(PULP_CBC_CMD(msg=0))
solution = p.LpStatus[status]
if solution == "Optimal":
    print("Optimal solution exists and found")
    for var in varlist:
        print( var + " = {}".format(p.value(lpvars[var])))
    print("objective z = {}".format(p.value(Lp_prob.objective)))
elif solution == "Infeasible":
    print("Problem has no feasible solution")
elif solution == "Unbounded":
    print("Problem is unbounded")
elif solution == "Undefined":
    print("Problem is undefined, solution may exist but can not be found")
```

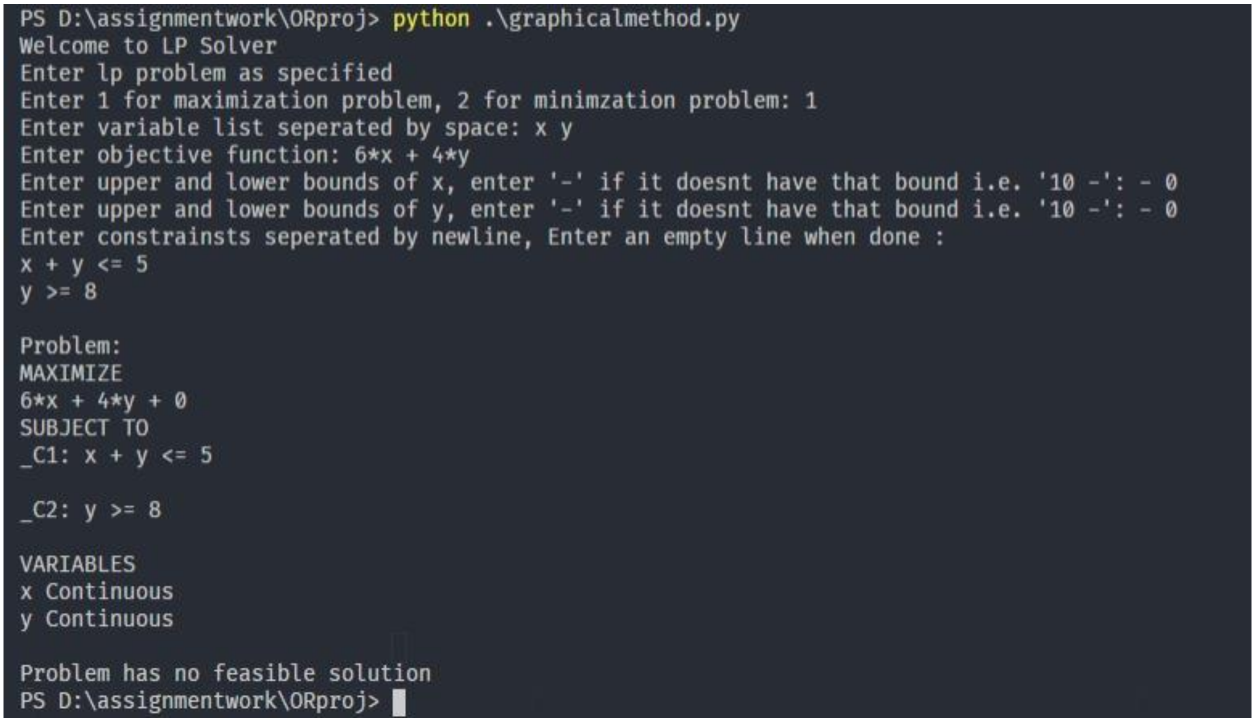
OUTPUT



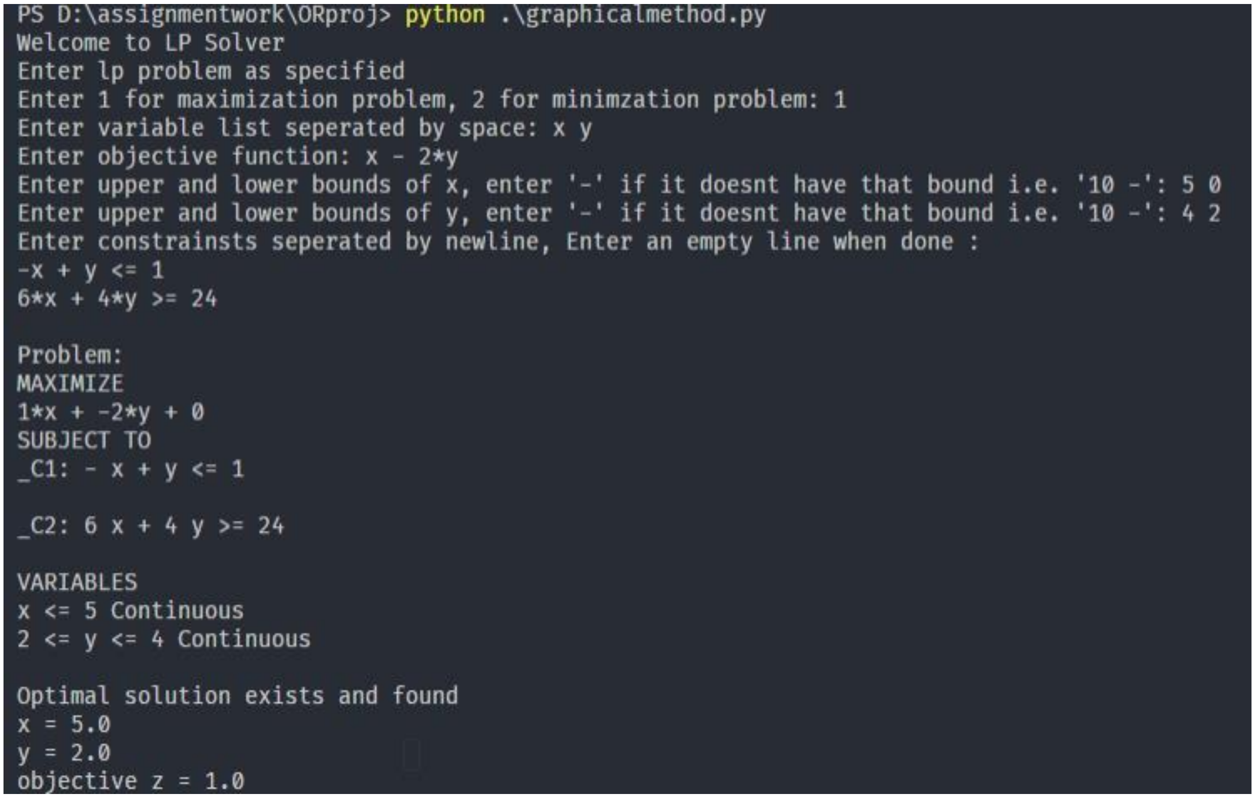
```
PS D:\assignmentwork\ORproj> python .\graphicalmethod.py
Welcome to LP Solver
Enter lp problem as specified
Enter 1 for maximization problem, 2 for minimzation problem: 1
Enter variable list seperated by space: x y
Enter objective function: 50 * x + 40 * y
Enter upper and lower bounds of x, enter '-' if it doesnt have that bound i.e. '10 -': - 0
Enter upper and lower bounds of y, enter '-' if it doesnt have that bound i.e. '10 -': - 0
Enter constrainsts seperated by newline, Enter an empty line when done :
x + 1.5 * y <= 750
2 * x + 3 * y <= 1500
2 * x + y <= 1000

Problem:
MAXIMIZE
50*x + 40*y + 0
SUBJECT TO
_C1: x + 1.5 y <= 750

_C2: 2 x + 3 y <= 1500

_C3: 2 x + y <= 1000

VARIABLES
x Continuous
y Continuous

Optimal solution exists and found
x = 375.0
y = 250.0
objective z = 28750.0
PS D:\assignmentwork\ORproj>
```



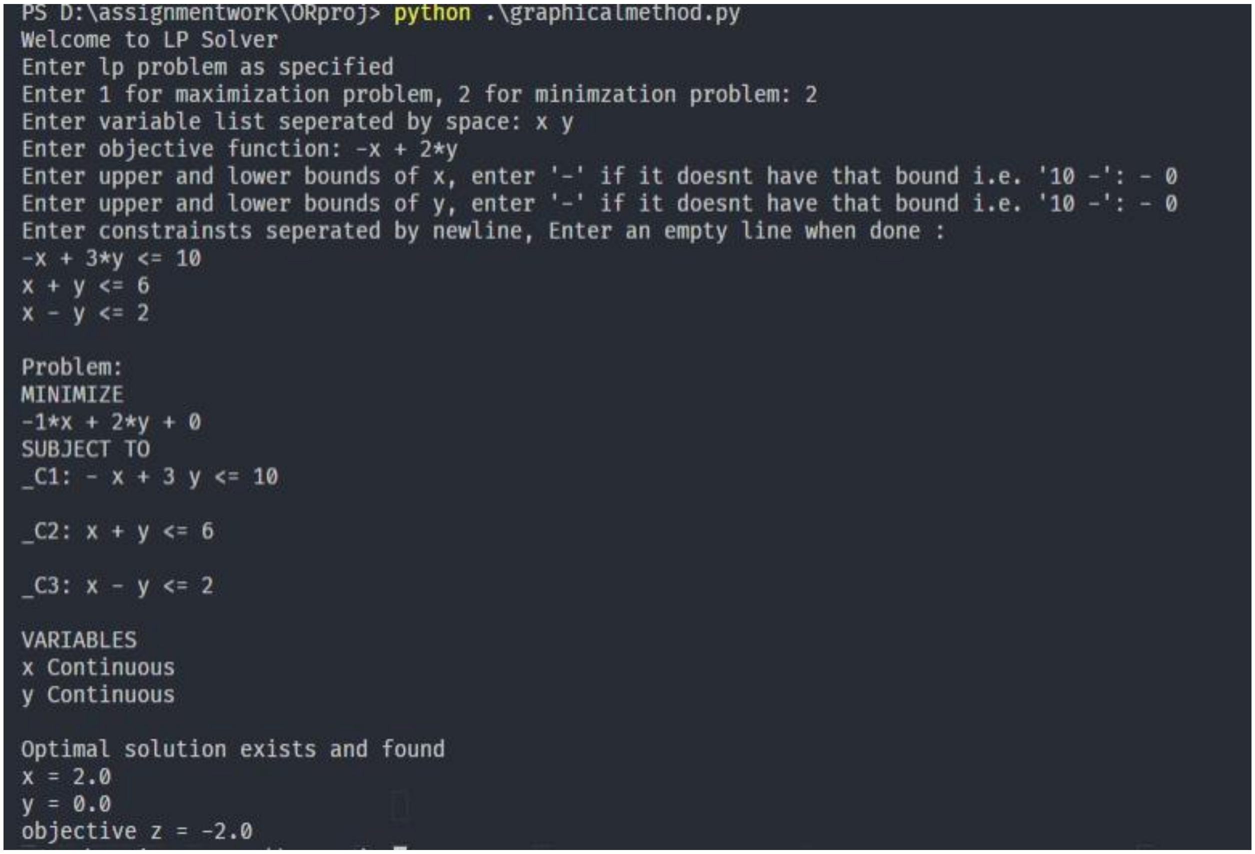
```
PS D:\assignmentwork\ORproj> python .\graphicalmethod.py
Welcome to LP Solver
Enter lp problem as specified
Enter 1 for maximization problem, 2 for minimzation problem: 1
Enter variable list seperated by space: x y
Enter objective function: x + y
Enter upper and lower bounds of x, enter '-' if it doesnt have that bound i.e. '10 -': - 0
Enter upper and lower bounds of y, enter '-' if it doesnt have that bound i.e. '10 -': - 0
Enter constrainsts seperated by newline, Enter an empty line when done :
x-y >= 1
x+y >= 2

Problem:
MAXIMIZE
1*x + 1*y + 0
SUBJECT TO
_C1: x - y >= 1

_C2: x + y >= 2

VARIABLES
x Continuous
y Continuous

Problem is unbounded
PS D:\assignmentwork\ORproj>
```



```
PS D:\assignmentwork\ORproj> python .\graphicalmethod.py
Welcome to LP Solver
Enter lp problem as specified
Enter 1 for maximization problem, 2 for minimzation problem: 1
Enter variable list seperated by space: x y
Enter objective function: 6*x + 4*y
Enter upper and lower bounds of x, enter '-' if it doesnt have that bound i.e. '10 -': - 0
Enter upper and lower bounds of y, enter '-' if it doesnt have that bound i.e. '10 -': - 0
Enter constrainsts seperated by newline, Enter an empty line when done :
x + y <= 5
y >= 8

Problem:
MAXIMIZE
6*x + 4*y + 0
SUBJECT TO
_C1: x + y <= 5

_C2: y >= 8

VARIABLES
x Continuous
y Continuous

Problem has no feasible solution
PS D:\assignmentwork\ORproj> 
```



```
PS D:\assignmentwork\ORproj> python .\graphicalmethod.py
Welcome to LP Solver
Enter lp problem as specified
Enter 1 for maximization problem, 2 for minimzation problem: 1
Enter variable list seperated by space: x y
Enter objective function: x - 2*y
Enter upper and lower bounds of x, enter '-' if it doesnt have that bound i.e. '10 -': 5 0
Enter upper and lower bounds of y, enter '-' if it doesnt have that bound i.e. '10 -': 4 2
Enter constrainsts seperated by newline, Enter an empty line when done :
-x + y <= 1
6*x + 4*y >= 24

Problem:
MAXIMIZE
1*x + -2*y + 0
SUBJECT TO
_C1: - x + y <= 1

_C2: 6 x + 4 y >= 24

VARIABLES
x <= 5 Continuous
2 <= y <= 4 Continuous

Optimal solution exists and found
x = 5.0
y = 2.0
objective z = 1.0
```



```
PS D:\assignmentwork\ORproj> python .\graphicalmethod.py
Welcome to LP Solver
Enter lp problem as specified
Enter 1 for maximization problem, 2 for minimzation problem: 2
Enter variable list seperated by space: x y
Enter objective function: -x + 2*y
Enter upper and lower bounds of x, enter '-' if it doesnt have that bound i.e. '10 -': - 0
Enter upper and lower bounds of y, enter '-' if it doesnt have that bound i.e. '10 -': - 0
Enter constrainsts seperated by newline, Enter an empty line when done :
-x + 3*y <= 10
x + y <= 6
x - y <= 2

Problem:
MINIMIZE
-1*x + 2*y + 0
SUBJECT TO
_C1: - x + 3 y <= 10

_C2: x + y <= 6

_C3: x - y <= 2

VARIABLES
x Continuous
y Continuous

Optimal solution exists and found
x = 2.0
y = 0.0
objective z = -2.0
```