```cpp
#include "stdafx.h"
#include <iostream>
#include <vector>
using namespace std;
class node {
public:
	int value;
	int r_position;
	int c_position;
	node * r_next;
	node * c_next;
	node() { value = -9999; r_position = c_position = -1; r_next = c_next = nullptr; }
	node(int v, int r, int c) { value = v; r_position = r; c_position = c; r_next =
c_next = nullptr; }
};
class my_matrix {
public:
	int num_rows;
	int num_cols;
	vector<node *> r_vec;
	vector<node *> c_vec;
	my_matrix() {}
	my_matrix(int r, int c);

	my_matrix(int r, int c, int n, int k);

	void print();
	void add_node(int v, int r, int c);

	my_matrix operator+(my_matrix M);//matrix addition
	my_matrix operator*(my_matrix M); //matrix multiplication
};
my_matrix::my_matrix(int r, int c) {
	num_rows = r;
	num_cols = c;
	r_vec.resize(r, nullptr);
	c_vec.resize(c, nullptr);
}
my_matrix::my_matrix(int r, int c, int n, int k) {
	num_rows = r;
	num_cols = c;
	r_vec.resize(r, nullptr);
	c_vec.resize(c, nullptr);
	for (int i = 0; i < n; i++) {
		int vv = rand() % (2 * k - 1) - (k - 1);
		int rr = rand() % r;
		int cc = rand() % c;
		add_node(vv, rr, cc);
	}
}
void my_matrix::add_node(int v, int r, int c) {
	int first_run = 0;
	bool head_swap = false;
	bool rnot_found = true;
	bool cnot_found = true;
	bool col_add = true;
	node * head = new node();
	node * temp_node = new node();
```

```
	node * n = new node();
	node * pre_node = new node();
	node * n3 = new node();
	if (v != 0)
	{
		//Add the element to the row vector of the matrix
		if (r_vec[r] == nullptr)
		{
			node * n1 = new node(v, r, c);
			r_vec[r] = n1;
			rnot_found = false;
		}
		else
		{
			pre_node->value = 0;
			pre_node->r_next = r_vec[r];
			while (pre_node->r_next != nullptr && pre_node->r_next->c_position
<= c)
			{
				if (pre_node->r_next->c_position == c)
				{
					rnot_found = false;
					pre_node->r_next->value = v;
				}
				pre_node = pre_node->r_next;
			}
		}
		if (rnot_found)
		{
			node * n2 = new node(v, r, c);
			if (pre_node->r_next == nullptr)
			{
				pre_node->r_next = n2;
			}
			else if (pre_node->r_next->c_position > c)
			{
				if (pre_node->r_next == r_vec[r]) { head_swap = true; }
				temp_node = pre_node->r_next;
				pre_node->r_next = n2;
				n2->r_next = temp_node;
				if (head_swap)
				{
					r_vec[r] = n2;
				}
			}
			else
			{
			}
		}
		//Add the element to the column vector of the matrix
		if (c_vec[c] == nullptr)
		{
			node * n1 = new node(v, r, c);
			c_vec[c] = n1;
			cnot_found = false;
		}
		else
		{
```

```
                        n3->value = 0;
                        n3->c_next = c_vec[c];
                        while (n3->c_next != nullptr && n3->c_next->r_position <= r)
                        {
                                if (n3->c_next->r_position == r)
                                {
                                        cnot_found = false;
                                        n3->c_next->value = v;
                                }
                                n3 = n3->c_next;
                        }
                }
                if (cnot_found)
                {
                        node * n2 = new node(v, r, c);
                        if (n3->c_next == nullptr)
                        {
                                n3->c_next = n2;
                        }
                        else if (n3->c_next->r_position > r)
                        {
                                head_swap = false;
                                if (n3->c_next == c_vec[c]) { head_swap = true; }
                                temp_node = n3->c_next;
                                n3->c_next = n2;
                                n2->c_next = temp_node;
                                if (head_swap)
                                {
                                        c_vec[c] = n2;
                                }
                        }
                        else
                        {
                        }
                }
        }
}
my_matrix my_matrix::operator+(my_matrix M) //overloading +operator for the addition of
two matrices
{
        node *n1 = new node();
        node *n2 = new node();
        bool n1_big = true;
        int vv, rr, cc, rows, cols, diff;
        if (num_rows > M.num_rows)
        {
                rows = num_rows;
                diff = num_rows - M.num_rows;
        }
        else
        {
                rows = M.num_rows;
                diff = M.num_rows - num_rows;
        }
        if (num_cols > M.num_cols)
                cols = num_cols;
        else
                cols = M.num_cols;
```

```
my_matrix result(rows, cols);
for (int i = 0; i < rows; i++)
{
	if (r_vec[i] == nullptr)
	{
		if (M.r_vec[i] != nullptr)
		{
			n2 = M.r_vec[i];
			while (n2 != nullptr)
			{
				rr = n2->r_position;
				cc = n2->c_position;
				vv = n2->value;
				result.add_node(vv, rr, cc);
				n2 = n2->r_next;
			}
		}
	}
	else if (M.r_vec[i] == nullptr)
	{
		if (r_vec[i] != nullptr)
		{
			n1 = r_vec[i];
			while (n1 != nullptr)
			{
				rr = n1->r_position;
				cc = n1->c_position;
				vv = n1->value;
				result.add_node(vv, rr, cc);
				n1 = n1->r_next;
			}
		}
	}
	else if (r_vec[i] != nullptr && M.r_vec[i] != nullptr)
	{
		n1 = r_vec[i];
		n2 = M.r_vec[i];
		while (n1 != nullptr && n2 != nullptr)
		{
			if (n1->c_position == n2->c_position)
			{
				rr = n1->r_position;
				cc = n1->c_position;
				vv = n1->value + n2->value;
				result.add_node(vv, rr, cc);
				n1 = n1->r_next;
				n2 = n2->r_next;
			}
			else if (n1->c_position < n2->c_position)
			{
				rr = n1->r_position;
				cc = n1->c_position;
				vv = n1->value;
				result.add_node(vv, rr, cc);
				n1 = n1->r_next;
			}
			else if (n1->c_position > n2->c_position)
			{
```

```
                                        rr = n2->r_position;
                                        cc = n2->c_position;
                                        vv = n2->value;
                                        result.add_node(vv, rr, cc);
                                        n2 = n2->r_next;
                                }
                        }
                        if (n1 == nullptr)
                        {
                                if (n2 != nullptr)
                                {
                                        while (n2 != nullptr)
                                        {
                                                rr = n2->r_position;
                                                cc = n2->c_position;
                                                vv = n2->value;
                                                result.add_node(vv, rr, cc);
                                                n2 = n2->r_next;
                                        }
                                }
                        }
                        else if (n2 == nullptr)
                        {
                                if (n1 != nullptr)
                                {
                                        while (n1 != nullptr)
                                        {
                                                rr = n1->r_position;
                                                cc = n1->c_position;
                                                vv = n1->value;
                                                result.add_node(vv, rr, cc);
                                                n1 = n1->r_next;
                                        }
                                }
                        }
                }
        }
        return result;
}
my_matrix my_matrix::operator*(my_matrix M) //overloading the * operator for multiplying
two matrices
{
        node *n1 = new node();
        node *n2 = new node();
        bool n1_big = true;
        int vv, rr, cc, rows, cols, diff;
        if (num_rows > M.num_rows)
        {
                rows = num_rows;
                diff = num_rows - M.num_rows;
        }
        else
        {
                rows = M.num_rows;
                diff = M.num_rows - num_rows;
        }
        if (num_cols > M.num_cols)
                cols = num_cols;
```

```
	else
		cols = M.num_cols;
	my_matrix result(rows, cols);
	for (int i = 0; i < num_rows; i++)
	{
		if (r_vec[i] != nullptr)
		{
			n1 = r_vec[i];
			for (int j = 0; j < M.num_cols; j++)
			{
				if (M.c_vec[j] != nullptr)
				{
					n1 = r_vec[i];
					n2 = M.c_vec[j];
					vv = 0;
					while (n1 != nullptr && n2 != nullptr)
					{
						if (n1->c_position == n2->r_position)
						{
							vv += ((n1->value)*(n2->value));
							n1 = n1->r_next;
							n2 = n2->c_next;
						}
						else if (n1->c_position < n2->r_position)
						{
							n1 = n1->r_next;
						}
						else if (n2->r_position < n1->c_position)
						{
							n2 = n2->c_next;
						}
					}
					rr = i;
					cc = j;
					result.add_node(vv, rr, cc);
				}
			}
		}
	}
	return result;
}
void my_matrix::print() {
	cout << endl;
	for (int i = 0; i < num_rows; i++) {
		node * p = r_vec[i];
		cout << endl;
		while (p != nullptr) {
			cout << p->value << " " << p->r_position << " " << p->c_position <<
"   ";
			p = p->r_next;
		}
	}
}
int main() {
	my_matrix M1(7, 5, 11, 8), M2(7, 5, 10, 8), M3(7, 5), M4(5, 6, 13, 9), M5(7, 6);
	M1.print();
	M2.print();
	M3 = M1 + M2;
```

```
        M3.print();
        M1.print();
        M4.print();
        M5 = M1 * M4;
        M5.print();
        getchar();
        getchar();
        return 0;
}
```