ex3.cpp

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

// Data structure to store Adjacency list nodes
struct Node {
    int val, cost;
    Node* next;
};

// Data structure to store graph edges
struct Edge {
    int src, dest, weight;
};

class Graph
{
    // Function to allocate new node of Adjacency List
    Node* getAdjListNode(int value, int weight, Node* head)
    {
        Node* newNode = new Node;
        newNode->val = value;
        newNode->cost = weight;

        // point new node to current head
        newNode->next = head;

        return newNode;
    }

    int N;  // number of nodes in the graph

    public:

    // An array of pointers to Node to represent
    // adjacency list
    Node **head;

    // Constructor
    Graph(Edge edges[], int n, int N)
    {
        // allocate memory
        head = new Node*[N]();
        this->N = N;
```

```
        // initialize head pointer for all vertices
        for (int i = 0; i < N; ++i)
            head[i] = nullptr;

        // add edges to the directed graph
        for (unsigned i = 0; i < n; i++)
        {
            int src = edges[i].src;
            int dest = edges[i].dest;
            int weight = edges[i].weight;

            // insert in the beginning
            Node* newNode = getAdjListNode(dest, weight, head[src]);

            // point head pointer to new node
            head[src] = newNode;

            // Uncomment below lines for undirected graph

            /*
               newNode = getAdjListNode(src, weight, head[dest]);

            // change head pointer to point to the new node
            head[dest] = newNode;
            */
        }
    }

    // Destructor
    ~Graph() {
        for (int i = 0; i < N; i++)
            delete[] head[i];

        delete[] head;
    }
};

// print all neighboring vertices of given vertex
void printList(Node* ptr, int i)
{
    while (ptr != nullptr)
    {
        cout << "(" << i << ", " << ptr->val
            << ", " << ptr->cost << ") ";

        ptr = ptr->next;
```

```
    }

    cout << endl;
}

// Graph Implementation in C++ without using STL
int main()
{
    // array of graph edges as per above diagram.
    Edge edges[] =
    {
        // (x, y, w) -> edge from x to y having weight w
        { 0, 1, 6 }, { 1, 2, 7 }, { 2, 0, 5 }, { 2, 1, 4 },
        { 3, 2, 10 }, { 4, 5, 1 }, { 5, 4, 3 }
    };

    // Number of vertices in the graph
    int N = 6;

    // calculate number of edges
    int n = sizeof(edges)/sizeof(edges[0]);

    // construct graph
    Graph graph(edges, n, N);

    // print adjacency list representation of graph
    for (int i = 0; i < N; i++)
    {
        // print all neighboring vertices of vertex i
        printList(graph.head[i], i);
    }

    return 0;
}
```

Output:

```
(0, 1, 6)
(1, 2, 7)
(2, 1, 4) (2, 0, 5)
(3, 2, 10)
(4, 5, 1)
(5, 4, 3)
```

ex2.cpp

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

```
// Data structure to store Adjacency list nodes
struct Node {
    int val;
    Node* next;
};

// Data structure to store graph edges
struct Edge {
    int src, dest;
};

class Graph
{
    // Function to allocate new node of Adjacency List
    Node* getAdjListNode(int dest, Node* head)
    {
        Node* newNode = new Node;
        newNode->val = dest;

        // point new node to current head
        newNode->next = head;

        return newNode;
    }

    int N;  // number of nodes in the graph

    public:

    // An array of pointers to Node to represent
    // adjacency list
    Node **head;

    // Constructor
    Graph(Edge edges[], int n, int N)
    {
        // allocate memory
        head = new Node*[N]();
        this->N = N;

        // initialize head pointer for all vertices
        for (int i = 0; i < N; i++)
            head[i] = nullptr;

        // add edges to the directed graph
```

```
        for (unsigned i = 0; i < n; i++)
        {
            int src = edges[i].src;
            int dest = edges[i].dest;

            // insert in the beginning
            Node* newNode = getAdjListNode(dest, head[src]);

            // point head pointer to new node
            head[src] = newNode;

            // Uncomment below lines for undirected graph

            /*
               newNode = getAdjListNode(src, head[dest]);

            // change head pointer to point to the new node
            head[dest] = newNode;
            */
        }
    }

    // Destructor
    ~Graph() {
        for (int i = 0; i < N; i++)
            delete[] head[i];

        delete[] head;
    }
};

// print all neighboring vertices of given vertex
void printList(Node* ptr)
{
    while (ptr != nullptr)
    {
        cout << " -> " << ptr->val << " ";
        ptr = ptr->next;
    }
    cout << endl;
}

// Graph Implementation in C++ without using STL
int main()
{
    // array of graph edges as per above diagram.
```

```
    Edge edges[] =
    {
        // pair (x, y) represents edge from x to y
        { 0, 1 }, { 1, 2 }, { 2, 0 }, { 2, 1 },
        { 3, 2 }, { 4, 5 }, { 5, 4 }
    };

    // Number of vertices in the graph
    int N = 6;

    // calculate number of edges
    int n = sizeof(edges)/sizeof(edges[0]);

    // construct graph
    Graph graph(edges, n, N);

    // print adjacency list representation of graph
    for (int i = 0; i < N; i++)
    {
        // print given vertex
        cout << i << " --";

        // print all its neighboring vertices
        printList(graph.head[i]);
    }

    return 0;
}
```

Output:

```
0 -- -> 1
1 -- -> 2
2 -- -> 1  -> 0
3 -- -> 2
4 -- -> 5
5 -- -> 4
```

ex1.cpp

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

// A utility function to add an edge in an
// undirected graph.
void addEdge(vector<int> adj[], int u, int v) {
```

```
    adj[u].push_back(v);
    adj[v].push_back(u);
}
// A utility function to print the adjacency list
// representation of graph
void printGraph(vector<int> adj[], int V) {
    for (int v = 0; v < V; ++v) {
        cout << "\n Adjacency list of vertex " << v << "< " "\n head ";
        for (auto x : adj[v])
            cout << "- > " << x; printf("\n");
    }
}

// Driver code
int main() { int V = 5;
    vector<int> adj[V];
    addEdge(adj, 0, 1);
    addEdge(adj, 0, 4);
    addEdge(adj, 1, 2);
    addEdge(adj, 1, 3);
    addEdge(adj, 1, 4);
    addEdge(adj, 2, 3);
    addEdge(adj, 3, 4);
    printGraph(adj, V);
    return 0;
}
```

Output:

```
 Adjacency list of vertex 0<
 head - > 1- > 4

 Adjacency list of vertex 1<
 head - > 0- > 2- > 3- > 4

 Adjacency list of vertex 2<
 head - > 1- > 3

 Adjacency list of vertex 3<
 head - > 1- > 2- > 4

 Adjacency list of vertex 4<
 head - > 0- > 1- > 3
```