```
/*WAP to implement graph.*/
#include <iostream>//Without STL (Standard Template Library)
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;
}
```

```
/*WAP to implement graph.*/
#include <iostream>//With STL (Standard Template Library)
#include <vector>
using namespace std;
// data structure to store graph edges
struct Edge
{
   int src, dest;
```

```
};
// class to represent a graph object
class Graph
{
public:
   // construct a vector of vectors to represent an adjacency list
   vector<vector<int>> adjList;
   // Graph Constructor
   Graph(vector<Edge> const &edges, int N)
   {
      // resize the vector to N elements of type vector<int>
      adjList.resize(N);
      // add edges to the directed graph
      for (auto &edge: edges)
      {
         // insert at the end
         adjList[edge.src].push_back(edge.dest);
         // Uncomment below line for undirected graph
         // adjList[edge.dest].push_back(edge.src);
      }
   }
};
// print adjacency list representation of graph
void printGraph(Graph const& graph, int N)
{
   for (int i = 0; i < N; i++)
   {
      // print current vertex number
      cout << i << " --> ";
      // print all neighboring vertices of vertex i
      for (int v : graph.adjList[i])
         cout << v << " ";
      cout << endl;
   }
}
// Graph Implementation using STL
int main()
```

```
{
    // vector of graph edges as per above diagram.
    // Please note that the initialization vector in below format will
    // work fine in C++11, C++14, C++17 but will fail in C++98.
    vector<Edge> edges =
    {
        { 0, 1 }, { 1, 2 }, { 2, 0 }, { 2, 1 },
        { 3, 2 }, { 4, 5 }, { 5, 4 }
    };
    // Number of nodes in the graph
    int N = 6;
    // construct graph
    Graph graph(edges, N);
    // print adjacency list representation of graph
    printGraph(graph, N);
    return 0;
}
```