```cpp
//TOPO SORT

//(1,2)(1,3)(2,3)(6,9)(5,4)(3,7)(0,7)(9,8)(3,0)(5,0)(2,6)(1,8) => 5 4 1 2 3 0 7 6 9 8
//(9,1)(5,6)(5,4)(4,8)(0,1)(7,2)(7,3)(9,4)(5,7)(0,2)(1,3)(0,6) => 0 5 6 7 2 9 1 3 4 8

#include <iostream>
#include <fstream>
#include <vector>

// Forward declaration of Leader and Trailer structures
typedef struct Leader* lref;
typedef struct Trailer* tref;

// Structure representing a Leader node
struct Leader {
    int key;        // Key of the leader
    int count;      // Number of incoming precedences
    lref next;      // Pointer to the next leader node in the list
    tref trails;    // Pointer to the list of trailers
};

// Structure representing a Trailer node
struct Trailer {
    lref id;        // Pointer to the leader node
    tref next;      // Pointer to the next trailer node in the list
};

// Function to find the leader with key x; if not exist yet, add to the end of the leader list
lref findLeader(lref& head, lref& tail, int x) {
    lref p = head;

    tail->key = x;

    while (p->key != x) {
        p = p->next;
    }

    if (p == tail) {
        tail = new Leader;

        p->count = 0;
        p->trails = nullptr;
        p->next = tail;
    }

    return p;
```

```cpp
}

// Function to split leaders with no precedences from the leader list
void splitLeaderWithNoPrecedence(lref& head, lref& tail) {
    lref p = head;
    head = nullptr;

    while (p != tail) {
        lref tmp = p->next;

        if (p->count == 0) {
            p->next = head;
            head = p;
        }

        p = tmp;
    }
}

// Function to add a new order x < y
void addOrder(lref& head, lref& tail, int x, int y) {
    lref xNode = findLeader(head, tail, x);
    lref yNode = findLeader(head, tail, y);

    tref xTrail = new Trailer{ yNode, xNode->trails };
    xNode->trails = xTrail;

    // Increase the number of precedences
    yNode->count++;
}

// Function to create leaders from pairs of orders
void createLeadersFromPairs(lref& head, lref& tail, 
  std::vector<std::pair<int, int>> orders) {
    head = new Leader{ -1, 0, nullptr, nullptr };
    tail = head;

    for (int i = 0; i < orders.size(); i++) {
        addOrder(head, tail, orders[i].first, orders[i].second);
    }
}

// Function to perform topological sort based on the given orders
void topoSort(std::vector<std::pair<int, int>> orders) {
    lref head, tail;
    createLeadersFromPairs(head, tail, orders);

    splitLeaderWithNoPrecedence(head, tail);

    lref p = head;

    while (p) {
        std::cout << p->key << " ";

        tref t = p->trails;

        p = p->next;

        for (tref q = t; q; q = q->next) {
            lref succNode = q->id;

            succNode->count--;

            if (succNode->count == 0) {
                succNode->next = p;

                p = succNode;
            }
        }
    }
}

// Function to parse orders from a file and return a vector of pairs
std::vector<std::pair<int, int>> parseFile(std::string fileName) {
    std::vector<std::pair<int, int>> orders;

    std::pair<int, int> p;
    char ch1, ch2, ch3;

    std::ifstream inFile;
    inFile.open(fileName);

    while (!inFile.eof()) {
        inFile >> ch1 >> p.first >> ch2 >> p.second >> ch3;

        if (inFile.eof()) {
            break;
        }

        orders.push_back(p);
    }

    inFile.close();

    return orders;
}

// Main function
```

```cpp
int main() {
    // Parse orders from the input file
    std::vector<std::pair<int, int>> orders = parseFile("input.txt");

    // Perform topological sort and print the result
    topoSort(orders);

    return 0;
}

```