```
//
//  common.h
//  LeetCode
//
//  Created by pulinghao on 2021/6/4.
//

#ifndef common_h
#define common_h

#include <iostream>
#include <string>
#include <set>
#include <vector>
#include <sstream>
#include <algorithm>
#include <unordered_map>
#include <unordered_set>
#include <queue>
#include <stack>
#include <numeric>


using namespace std;

struct ListNode {
     int val;
     ListNode *next;
     ListNode() : val(0), next(nullptr) {}
     ListNode(int x) : val(x), next(nullptr) {}
     ListNode(int x, ListNode *next) : val(x), next(next) {}
};


/**
 * Definition for a binary tree node.
 */
struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode() : val(0), left(nullptr), right(nullptr) {}
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
    TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right)
{}
 };

// Definition for a Node.
```

```cpp
class Node {
public:
    int val;
    vector<Node*> children;

    Node() {}

    Node(int _val) {
        val = _val;
    }

    Node(int _val, vector<Node*> _children) {
        val = _val;
        children = _children;
    }
};


int stringToInteger(string input) {
    return stoi(input);
}

void trimLeftTrailingSpaces(string &input) {
    input.erase(input.begin(), find_if(input.begin(), input.end(), [](int ch) {
        return !isspace(ch);
    }));
}

void trimRightTrailingSpaces(string &input) {
    input.erase(find_if(input.rbegin(), input.rend(), [](int ch) {
        return !isspace(ch);
    }).base(), input.end());
}

vector<int> stringToIntegerVector(string input) {
    vector<int> output;
    trimLeftTrailingSpaces(input);
    trimRightTrailingSpaces(input);
    input = input.substr(1, input.length() - 2);
    stringstream ss;
    ss.str(input);
    string item;
    char delim = ',';
    while (getline(ss, item, delim)) {
        output.push_back(stoi(item));
    }
    return output;
}
```

```
vector<vector<int>> stringToIntegerVectors(string input){
    vector<vector<int>> output;
    trimLeftTrailingSpaces(input);
    trimRightTrailingSpaces(input);
    input = input.substr(1, input.length() - 2);

    string tempStr;
    stack<char> st;
    for (int i = 0; i < input.size(); i++) {
        if (input[i] == '[') {
            tempStr.push_back(input[i]);
        } else if(isnumber(input[i])){
            tempStr.push_back(input[i]);
        } else if(input[i] == ']'){
            tempStr.push_back(input[i]);
            vector<int> outLine = stringToIntegerVector(tempStr);
            output.push_back(outLine);
            tempStr.clear();
        } else {
            if (input[i] == ',') {
                if (tempStr.size() == 0) {
                    continue;
                } else {
                    tempStr.push_back(input[i]);
                }
            }
        }
    }

    return output;
}

ListNode* stringToListNode(string input) {
    // Generate list from the input
    vector<int> list = stringToIntegerVector(input);

    // Now convert that list into linked list
    ListNode* dummyRoot = new ListNode(0);
    ListNode* ptr = dummyRoot;
    for(int item : list) {
        ptr->next = new ListNode(item);
        ptr = ptr->next;
    }
    ptr = dummyRoot->next;
    delete dummyRoot;
    return ptr;
}

ListNode* vectorToListNode(vector<int> list){
    ListNode* dummyRoot = new ListNode(0);
    ListNode* ptr = dummyRoot;
    for(int item : list) {
        ptr->next = new ListNode(item);
        ptr = ptr->next;
    }
    ptr = dummyRoot->next;
    delete dummyRoot;
    return ptr;
}



TreeNode *vectorToTreeNode(vector<string> input){
    if (input.size() == 0) {
        return NULL;
    }

    TreeNode *root = (TreeNode *)malloc(sizeof(TreeNode));
    root->val = stringToInteger(input[0]);

    deque<TreeNode *> queue;
    int index = 1;
    int front = 0;
    queue.push_back(root);
    while(index < input.size()){
        TreeNode *node = queue[front];
        front += 1;
        string value = input[index];
        index += 1;
        if (value != "#") {
            int leftValue = stringToInteger(value);
            TreeNode *p = new TreeNode();
            p->val = leftValue;
            node->left = p;
            queue.push_back(node->left);
        } else {
            node->left = NULL;
        }

        if (index >= input.size()){
            break;
        }

        value = input[index];
        index += 1;
        if (value != "#") {
            int rightValue = stringToInteger(value);
            TreeNode *p = new TreeNode();
            p->val = rightValue;
            node->right = p;
            queue.push_back(node->right);
        } else {
            node->right = NULL;
        }
    }
    return root;
}


TreeNode *stringToTreeNode(string input){
    string subString = input.substr(1, input.size() - 2);
    vector<string> arr;
    int start = 0;
    for (int i = 0; i < subString.size(); i++) {
        if (subString[i] == ',') {
            string tempSub = subString.substr(start,i - start);
            if (tempSub == "null") {
                arr.push_back("#");
            } else {
                arr.push_back(tempSub);
            }
            start = i + 1;
        }
    }

    string tempSub = subString.substr(start,subString.size() - start);
    if (tempSub == "null") {
        arr.push_back("#");
    } else {
        arr.push_back(tempSub);
    }

    return vectorToTreeNode(arr);
}

string treeNodeToString(TreeNode *root){
    string res;
    if (!root) {
        return "[]";
    }

    deque<TreeNode *> queue;
    queue.push_back(root);
    while(!queue.empty()){
        TreeNode *front = queue.front();

        queue.pop_front();
        if (front) {
            int val = front->val;
            res.append(to_string(val));
            res += ", ";
        } else {
            res += "null, ";
        }

        if (front) {
            queue.push_back(front->left);
            queue.push_back(front->right);
        }
    }
    return "[" + res.substr(0,res.length() - 2) + "]";
}


string listNodeToString(ListNode* node) {
    if (node == nullptr) {
        return "[]";
    }

    string result;
    while (node) {
        result += to_string(node->val) + ", ";
        node = node->next;
    }
    return "[" + result.substr(0, result.length() - 2) + "]";
}


/// 随机分治
/// @param nums <#nums description#>
int randomPartion(vector<int>& nums,int start, int end){
    if (nums.size() == 0) {
        return -1;
    }

    if (start < 0 || end >= nums.size()) {
        return -1;
    }

    int index = rand() % (end - start + 1 ) + start ;
    // 指向比索引数大的数
    int small = start - 1;
    // 交换索引数
    swap(nums[index], nums[end]);

    for(index = start; index < end; index ++){
        if (nums[index] < nums[end]) {
            small++;
            if (small != index) {
                swap(nums[small], nums[index]);
            }
        }
    }
    small++;
    swap(nums[small], nums[end]);
    return small;
}

/// 随机分治
/// @param nums <#nums description#>
int randomPartion2(vector<int>& nums,int start, int end){
    if (nums.size() == 0) {
        return -1;
    }

    if (start < 0 || end >= nums.size()) {
        return -1;
    }

    int index = 0 ;
    // 指向比索引数大的数
    int indexNum = nums[index];
    int i = start;
    int j = end;
    swap(nums[start],nums[index]);
    while(i < j){
        while(i < j && nums[j] >= indexNum) j--;
        while(i < j && nums[i] <= indexNum) i++;
        swap(nums[i], nums[j]);
    }
    swap(nums[i], nums[index]);
    return i;
}

int partion(vector<int> &nums,int l,int r){
    int i = l;
    int j = r;
    while(i < j){
        while(i < j && nums[j] >= nums[l]) j--;
        while(i < j && nums[i] <= nums[l]) i++;
        swap(nums[i], nums[j]);
    }
    swap(nums[i], nums[l]);
    return i;
}
```

```
/// 查找链表的中点
/// @param head <#head description#>
ListNode* middleNode(ListNode* head) {
    ListNode* slow = head;
    ListNode* fast = head;
    while (fast->next != nullptr && fast->next->next != nullptr) {
        slow = slow->next;
        fast = fast->next->next;
    }
    return slow;
}


/// 链表反转
/// @param head <#head description#>
ListNode* reverseList(ListNode* head) {
    ListNode* prev = nullptr;
    ListNode* curr = head;
    while (curr != nullptr) {
        ListNode* nextTemp = curr->next;
        curr->next = prev;
        prev = curr;
        curr = nextTemp;
    }
    return prev;
}


/// 按照L1，L2的顺序合并链表
/// @param l1 <#l1 description#>
/// @param l2 <#l2 description#>
void mergeList(ListNode* l1, ListNode* l2) {
    ListNode* l1_tmp;
    ListNode* l2_tmp;
    while (l1 != nullptr && l2 != nullptr) {
        l1_tmp = l1->next;
        l2_tmp = l2->next;

        l1->next = l2;
        l1 = l1_tmp;

        l2->next = l1;
        l2 = l2_tmp;
    }
}

std::string SubString(const std::string& string, int beginIndex, int endIndex) {
    int size = (int)string.size();
    if (beginIndex < 0 || beginIndex > size - 1)
        return "-1"; // Index out of bounds
    if (endIndex < 0 || endIndex > size - 1)
        return "-1"; // Index out of bounds
    if (beginIndex > endIndex)
        return "-1"; // Begin index should not be bigger that end.

    std::string substr;
    for (int i = 0; i < size; i++)
        if (i >= beginIndex && i <= endIndex)
            substr += (char)string[i];
    return substr;
}

#endif /* common_h */
```