# Latex

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define fi first
#define se second
const int N = 1e6 + 9;
const int N2 = N * 10;
const int mod = 1e9 + 7;
const int inf = LLONG_MAX;

signed main(){
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    if (fopen("TASK.INP", "r")){
    freopen("TASK.INP", "r", stdin);
    freopen("TASK.OUT", "w", stdout);}

    string s;
    cin >> s;
    stack<int> st;
    bool flag = 0;

    for (int i=0; i<s.length(); i++){
        if (s[i] == ']'){
            if (!st.empty() && st.top() == '['){
                st.pop();
            }
            else{
                flag = 1;
                break;
            }
        }
        if (s[i] == ')'){
            if (!st.empty() && st.top() == '('){
                st.pop();
            }
            else{
                flag = 1;
                break;
            }
        }
        if (s[i] == '}'){
            if (!st.empty() && st.top() == '{'){
                st.pop();
            }
            else{
                flag = 1;
                break;
            }
        }
        if (s[i] == '[' || s[i] == '{' || s[i] == '('){
            st.push(s[i]);
        }
    }

    if (!st.empty() || flag){
        cout << 0;
    }
    else cout << 1;

}
```

## LinkedList-Insertion

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

class SinglyLinkedListNode {
    public:
        int data;
        SinglyLinkedListNode *next;

        SinglyLinkedListNode(int node_data) {
            this->data = node_data;
            this->next = nullptr;
        }
};

class SinglyLinkedList {
    public:
        SinglyLinkedListNode *head;
        SinglyLinkedListNode *tail;

        SinglyLinkedList() {
            this->head = nullptr;
            this->tail = nullptr;
        }

        void insert_node(int node_data) {
            SinglyLinkedListNode* node = new SinglyLinkedListNode(node_data);
            if (!this->head) {
                this->head = node;
            } else {
                this->tail->next = node;
            }

            this->tail = node;
        }
};

void free_singly_linked_list(SinglyLinkedListNode* node) {
    while (node) {
        SinglyLinkedListNode* temp = node;
        node = node->next;

        free(temp);
    }
}

void printLinkedList(SinglyLinkedListNode* head) {
    while (head != NULL){
        cout<< head->data << ' ';
        head = head->next;
    }
}


// Complete the insertSortedLinkedList function below.

/*
 * For your reference:
 *
 * SinglyLinkedListNode {
 *     int data;
 *     SinglyLinkedListNode* next;
 * };
 *
 */
SinglyLinkedListNode* insertSortedLinkedList(SinglyLinkedListNode* head, int x) {

    SinglyLinkedListNode* newNode = new SinglyLinkedListNode(x);
    if (head == NULL || head->data >= x) {
        newNode->next = head;
        return newNode;
    }

    SinglyLinkedListNode* current = head;
    while (current->next != NULL && current->next->data < x) {
        current = current->next;
    }
    newNode->next = current->next;
    current->next = newNode;

    return head;
}

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    if (fopen("TASK.INP", "r")){
    freopen("TASK.INP", "r", stdin);
    freopen("TASK.OUT", "w", stdout);}
    SinglyLinkedList* llist = new SinglyLinkedList();
    int llist_count;
    int x;

    cin >> llist_count;
    cin.ignore(numeric_limits<streamsize>::max(), '\n');

    for (int i = 0; i < llist_count; i++) {
        int llist_item;
        cin >> llist_item;
        cin.ignore(numeric_limits<streamsize>::max(), '\n');

        llist->insert_node(llist_item);
    }
    cin >> x;
    llist->head = insertSortedLinkedList(llist->head, x);
    printLinkedList(llist->head);

    return 0;
}
```

## LinkedList-NhapDaThuc

```
#include <bits/stdc++.h>
using namespace std;


struct DONTHUC{
    int somu;
    double heso;

    DONTHUC(double _heso = 0,int _somu=0){
        heso = _heso;
        somu  = _somu;
    }

    DONTHUC& operator = (const DONTHUC &rhs){
        if (this == &rhs) return *this;
        this->heso = rhs.heso;
        this->somu = rhs.somu;
        return *this;
    }
};


struct Node{
    DONTHUC* data;
    Node* next;

    Node(DONTHUC* _data = nullptr){
        this->data = _data;
        this->next = nullptr;
    }

};

struct DATHUC{
    Node* head;
    Node* tail;
    DATHUC(){
        this->head = nullptr;
        this->tail = nullptr;
    }
};

void Nhap(DATHUC &B, double heso, int somu){
    if (heso == 0){
        return;
    }
    DONTHUC* newNode = new DONTHUC(heso,somu);
    Node* node = new Node(newNode);
    if (B.head == nullptr){
        B.head = node;
        B.tail = node;
    }
    else{
        B.tail->next = node;
        B.tail = node;
    }
}
```

```
void Xuat(DATHUC B){
    Node* cur = B.head;
    if (B.head == nullptr){
        cout << "0";
        return;
    }
    while (cur != nullptr){
        if (cur->data->heso > 0 && cur != B.head) cout << "+";
        else if (cur->data->heso < 0) cout << "-";
        if (cur->data->heso == 0){
            cur = cur->next;
            continue;
        }
        if (cur->data->somu == 0){
            cout << abs(cur->data->heso);
            cur = cur->next;
            continue;
        }
        if (cur->data->somu == 1){
            cout << abs(cur->data->heso) << "x";
            cur = cur->next;
            continue;
        }
        if (abs(cur->data->heso) == 1){
            cout << "x^" << cur->data->somu;
            cur = cur->next;
            continue;
        }
        cout << abs(cur->data->heso) << "x^" << cur->data->somu;
        cur = cur->next;
    }
}

double TinhDaThuc(DATHUC B, double x){
    if (!B.head) return 0;
    double ans = 0;

    Node* cur = B.head;
    while (cur != nullptr){
        // cout << cur->data->heso << " " << cur->data->somu << "\n";
        ans += cur->data->heso * pow(x, cur->data->somu);
        cur = cur->next;
    }

    return ans;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    if (fopen("TASK.INP", "r")){
    freopen("TASK.INP", "r", stdin);
    freopen("TASK.OUT", "w", stdout);}
    DATHUC B;
    int N;

    cin >> N;
    for (int test = 0; test < N; test++){
        double heso; int somu;
        cin >> heso >> somu;
        Nhap(B,heso,somu);
    }


    cout << "Da thuc vua nhap la: "; Xuat(B);
    double x; cin >> x;
    cout << "\nVoi x=" << x << ", gia tri da thuc la: "
         << setprecision(2) << fixed << TinhDaThuc(B, x);
    return 0;
}
```

## LinkedList-Reverse

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

class SinglyLinkedListNode {
    public:
        int data;
        SinglyLinkedListNode *next;

        SinglyLinkedListNode(int node_data) {
            this->data = node_data;
            this->next = nullptr;
        }
};

class SinglyLinkedList {
    public:
        SinglyLinkedListNode *head;
        SinglyLinkedListNode *tail;

        SinglyLinkedList() {
            this->head = nullptr;
            this->tail = nullptr;
        }

};




// Complete the insertSortedLinkedList function below.

/*
 * For your reference:
 *
 * SinglyLinkedListNode {
 *     int data;
 *     SinglyLinkedListNode* next;
 * };
 *
 */

void insert_node(SinglyLinkedList* llist, int node_data) {
    SinglyLinkedListNode* newNode = new SinglyLinkedListNode(node_data);

    if (!llist->head) {
        llist->head = newNode;
    } else {
        llist->tail->next = newNode;
    }

    llist->tail = newNode;
}

void reverseLinkedList(SinglyLinkedList* llist) {
    SinglyLinkedList rev = SinglyLinkedList();
    SinglyLinkedListNode* current = llist->head;
    while (current != NULL) {
        SinglyLinkedListNode* newNode = new SinglyLinkedListNode(current->data);
        newNode->next = rev.head;
        rev.head = newNode;
        current = current->next;
    }
    llist->head = rev.head;
    llist->tail = rev.tail;
    SinglyLinkedListNode* currentTail = rev.head;
    while (currentTail->next != NULL) {
        currentTail = currentTail->next;
    }
}
```

```
void printLinkedList(SinglyLinkedList *list) {
    SinglyLinkedListNode* current = list->head;
    while (current != NULL) {
        cout << current->data << " ";
        current = current->next;
    }
    cout << endl;
}

 int main()
 {
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    if (fopen("TASK.INP", "r")){
    freopen("TASK.INP", "r", stdin);
    freopen("TASK.OUT", "w", stdout);}
     SinglyLinkedList* llist = new SinglyLinkedList();
     int llist_count;


     cin >> llist_count;

     for (int i = 0; i < llist_count; i++) {
         int llist_item;
         cin >> llist_item;

         insert_node(llist,llist_item);
     }


     reverseLinkedList(llist);
     printLinkedList(llist);

     return 0;
 }
```

## Dec to Bin

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define fi first
#define se second
const int N = 1e6 + 9;
const int N2 = N * 10;
const int mod = 1e9 + 7;
const int inf = LLONG_MAX;
```

```
string dectobin(int n){
    if (n == 0) return "";
    dectobin(n / 2);
    return dectobin(n / 2) + to_string(n % 2);
}

signed main(){
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    if (fopen("TASK.INP", "r")){
    freopen("TASK.INP", "r", stdin);
    freopen("TASK.OUT", "w", stdout);}

    int n;
    cin >> n;
    string s = dectobin(n);
    cout << s;

}
```

## LinkedList: MergetwoSortedLinkedList

```
#include <bits/stdc++.h>
using namespace std;

struct SinglyLinkedListNode {
    int data;
    SinglyLinkedListNode *next;

    SinglyLinkedListNode(int node_data) {
        this->data = node_data;
        this->next = nullptr;
    }
};

struct SinglyLinkedList {
    SinglyLinkedListNode *head;
    SinglyLinkedListNode *tail;

    SinglyLinkedList() {
        this->head = nullptr;
        this->tail = nullptr;
    }

    void insert_node(int node_data);
```

```
};

void SinglyLinkedList::insert_node(int node_data)
{
        SinglyLinkedListNode* node = new SinglyLinkedListNode(node_data);

        if (!this->head) {
            this->head = node;
        } else {
            this->tail->next = node;
        }

        this->tail = node;
}



void print_singly_linked_list(SinglyLinkedListNode* node,char sep=' ') {
    while (node) {
        cout << node->data;

        node = node->next;

        if (node) {
            cout << sep;
        }
    }
    cout << '\n';
}

void free_singly_linked_list(SinglyLinkedListNode* node) {
    while (node) {
        SinglyLinkedListNode* temp = node;
        node = node->next;

        free(temp);
    }
}



SinglyLinkedListNode* mergeLists(SinglyLinkedListNode* head_list1,SinglyLinkedListNode* h

    SinglyLinkedListNode* curNode = new SinglyLinkedListNode(0);
    SinglyLinkedListNode* headNode = curNode;
    SinglyLinkedListNode* tmp1Node = head_list1;
    SinglyLinkedListNode* tmp2Node = head_list2;
    while (tmp1Node != NULL && tmp2Node != NULL){
        SinglyLinkedListNode* newInsertNode = new SinglyLinkedListNode(0);
        if (tmp1Node->data < tmp2Node->data){
            newInsertNode->data = tmp1Node->data;
            tmp1Node = tmp1Node->next;
        }
        else{
            newInsertNode->data = tmp2Node->data;

            tmp2Node = tmp2Node->next;
        }
        curNode->next = newInsertNode;
        curNode = curNode->next;
    }
    while (tmp1Node != NULL){
        SinglyLinkedListNode* newInsertNode = new SinglyLinkedListNode(0);
        newInsertNode->data = tmp1Node->data;
        curNode->next = newInsertNode;
        curNode = curNode->next;
        tmp1Node = tmp1Node->next;
    }
    while (tmp2Node != NULL){
        SinglyLinkedListNode* newInsertNode = new SinglyLinkedListNode(0);
        newInsertNode->data = tmp2Node->data;
        curNode->next = newInsertNode;
        curNode = curNode->next;
        tmp2Node = tmp2Node->next;
    }
    curNode->next = NULL;
    headNode = headNode->next;
    return headNode;
}




int main (){
    cin.tie(0); std::ios::sync_with_stdio(false);
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    if (fopen("TASK.INP", "r")){
    freopen("TASK.INP", "r", stdin);
    freopen("TASK.OUT", "w", stdout);}
    int tests;
    cin >> tests;




    for (int t = 0; t < tests; t++){
        int llist1_num, llist2_num;
        cin >> llist1_num >> llist2_num;

        SinglyLinkedList* llist1 = new SinglyLinkedList();


        for (int i = 0; i < llist1_num; i++){
            int llist1_item;
            cin >> llist1_item;

            llist1->insert_node(llist1_item);
        }

        SinglyLinkedList* llist2 = new SinglyLinkedList();


        for (int i = 0; i < llist2_num; i++){
            int llist2_item;
            cin >> llist2_item;

            llist2->insert_node(llist2_item);
        }

        SinglyLinkedListNode* llist3 = mergeLists(llist1->head, llist2->head);

        print_singly_linked_list(llist3);

        free_singly_linked_list(llist3);


    }
}
```

## Tree: Preorder Traversal (NLR) - Duyệt cây BST theo NLR

```
#include <bits/stdc++.h>
using namespace std;


class Node {
    public:
        int data;
        Node *left;
        Node *right;
        Node(int d) {
            data = d;
            left = NULL;
            right = NULL;
        }
};

class Solution {
    public:
        Node* insert(Node* root, int data) {
            if(root == NULL) {
                return new Node(data);
            } else {
                Node* cur;
                if(data <= root->data) {
                    cur = insert(root->left, data);
                    root->left = cur;
                } else {
                    cur = insert(root->right, data);
                    root->right = cur;
                }

               return root;
           }
        }

/* you only have to complete the function given below.
Node is defined as

class Node {
    public:
        int data;
        Node *left;
        Node *right;
        Node(int d) {
            data = d;
            left = NULL;
            right = NULL;
        }
};

*/

    void preOrder(Node *root) {
        // Preorder traversal: root -> left -> right
        if (root == NULL) {
            return;
        }
        cout << root->data << " ";
        preOrder(root->left);
        preOrder(root->right);
    }

}; //End of Solution

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    if (fopen("TASK.INP", "r")){
    freopen("TASK.INP", "r", stdin);
    freopen("TASK.OUT", "w", stdout);}
    Solution myTree;
    Node* root = NULL;

    int t;
    int data;

    std::cin >> t;

    while(t-- > 0) {
        std::cin >> data;
        root = myTree.insert(root, data);
    }

    myTree.preOrder(root);

    return 0;
}
```

## Tree: Preorder Traversal (NLR) II - Duyệt cây BST theo NLR không đệ quy

```
#include <bits/stdc++.h>
using namespace std;


class Node {
    public:
        int data;
        Node *left;
        Node *right;
        Node(int d) {
            data = d;
            left = NULL;
            right = NULL;
        }
```

```
};

class Solution {
    public:
        Node* insert(Node* root, int data) {
            if(root == NULL) {
                return new Node(data);
            } else {
                Node* cur;
                if(data <= root->data) {
                    cur = insert(root->left, data);
                    root->left = cur;
                } else {
                    cur = insert(root->right, data);
                    root->right = cur;
                }

               return root;
           }
        }

/* you only have to complete the function given below.
Node is defined as

class Node {
    public:
        int data;
        Node *left;
        Node *right;
        Node(int d) {
            data = d;
            left = NULL;
            right = NULL;
        }
};

*/

    void preOrder(Node *root) {
        // Preorder traversal: root -> left -> right

        stack<Node*> s;
        s.push(root);

        while(!s.empty()) {
            Node* current = s.top();
            s.pop();
            cout << current->data << " ";
            if(current->right) {
                s.push(current->right);
            }
            if(current->left) {
                s.push(current->left);
            }
        }

    }

}; //End of Solution

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    if (fopen("TASK.INP", "r")){
    freopen("TASK.INP", "r", stdin);
    freopen("TASK.OUT", "w", stdout);}
    Solution myTree;
    Node* root = NULL;

    int t;
    int data;

    std::cin >> t;

    while(t-- > 0) {
        std::cin >> data;
        root = myTree.insert(root, data);
    }

    myTree.preOrder(root);

    return 0;
}
```

## LinkedList: Tìm nút chung của 2 danh sách liên kết đơn

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

class SinglyLinkedListNode {
    public:
        int data;
        SinglyLinkedListNode *next;
        SinglyLinkedListNode(int node_data) {
            this->data = node_data;
            this->next = nullptr;
        }
};

class SinglyLinkedList {
    public:
        SinglyLinkedListNode *head;
        SinglyLinkedListNode *tail;

        SinglyLinkedList() {
            this->head = nullptr;
            this->tail = nullptr;
        }

        void insert_node(int node_data) {
            SinglyLinkedListNode* node = new SinglyLinkedListNode(node_data);

            if (!this->head) {
                this->head = node;
            } else {
                this->tail->next = node;
            }
            this->tail = node;
        }
        void printLinkedList() {
            SinglyLinkedListNode* p;
            p = head;
            while (p != NULL){
                cout<<p->data<<endl;
                p = p->next;
            }
        }

};

// Complete the SinglyLinkedListNode* findMergeNode(SinglyLinkedListNode* head1, SinglyLi
/*
     * For your reference:
     *
     * SinglyLinkedListNode {
     *     int data;
     *     SinglyLinkedListNode* next;
     * };
     * SinglyLinkedList {
     *    SinglyLinkedListNode *head;
     *    SinglyLinkedListNode *tail;
     *
     */
SinglyLinkedListNode* findMergeNode(SinglyLinkedListNode* head1, SinglyLinkedListNode* he

    SinglyLinkedListNode* p1 = head1;
    SinglyLinkedListNode* p2 = head2;

    while (p1 != p2) {
        p1 = (p1 == nullptr) ? head2 : p1->next;
        p2 = (p2 == nullptr) ? head1 : p2->next;
    }

    return p1;

}

void free_singly_linked_list(SinglyLinkedListNode* node) {
    while (node) {
        SinglyLinkedListNode* temp = node;
        node = node->next;
        free(temp);
    }
}



int main()
{
    SinglyLinkedList* llist1 = new SinglyLinkedList();
    SinglyLinkedList* llist2 = new SinglyLinkedList();
    int llist_count;
    int x;

    cin >> llist_count;
    cin.ignore(numeric_limits<streamsize>::max(), '\n');

    for (int i = 0; i < llist_count; i++) {
        int llist_item;
        cin >> llist_item;
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        llist1->insert_node(llist_item);
    }

    cin >> llist_count;
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
    for (int i = 0; i < llist_count; i++) {
        int llist_item;
        cin >> llist_item;
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
```

```
            llist2->insert_node(llist_item);
        }

        cin >> llist_count;
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        if (llist_count>0){
            int llist_item;
            cin >> llist_item;
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            llist1->insert_node(llist_item);
            if (llist2->head != nullptr)  llist2->tail->next = llist1->tail;
            else llist2->head = llist2->tail = llist1->tail;
        }
        for (int i = 1; i < llist_count; i++) {
            int llist_item;
            cin >> llist_item;
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            llist1->insert_node(llist_item);
        }
        llist2->tail = llist1->tail;
        SinglyLinkedListNode* p;
        p = findMergeNode(llist1->head,llist2->head);
        if (p == nullptr)
            cout<<"NA";
        else
            cout<<p->data;

        //free_singly_linked_list(llist1->head);
        //free_singly_linked_list(llist2->head);

        return 0;
    }
```

## Binary Search Tree: Nút tổ tiên thấp nhất (tiếng Việt)

```
#include <bits/stdc++.h>

using namespace std;

class Node {
    public:
        int data;
        Node *left;
        Node *right;
        Node(int d) {
            data = d;
            left = NULL;
            right = NULL;
        }
};

class Solution {
    public:
        Node* insert(Node* root, int data) {
            if(root == NULL) {
                return new Node(data);
            } else {
                Node* cur;
                if(data <= root->data) {
                    cur = insert(root->left, data);
                    root->left = cur;
                } else {
                    cur = insert(root->right, data);
                    root->right = cur;
               }

               return root;
           }
        }

/*The tree node has data, left child and right child
class Node {
    int data;
    Node* left;
    Node* right;
};

*/

    Node *lca(Node *root, int v1,int v2) {
        if (root == NULL) {
            return NULL;
        }
        if (root->data == v1 || root->data == v2) {
            return root;
        }
        Node* left_lca = lca(root->left, v1, v2);
        Node* right_lca = lca(root->right, v1, v2);
        if (left_lca && right_lca) {
            return root;
        }
        return (left_lca != NULL) ? left_lca : right_lca;
    }

}; //End of Solution
```

```
int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    if (fopen("TASK.INP", "r")){
    freopen("TASK.INP", "r", stdin);
    freopen("TASK.OUT", "w", stdout);}
    Solution myTree;
    Node* root = NULL;

    int t;
    int data;

    std::cin >> t;

    while(t-- > 0) {
        std::cin >> data;
        root = myTree.insert(root, data);
    }

    int v1, v2;
    std::cin >> v1 >> v2;

    Node *ans = myTree.lca(root, v1, v2);

    std::cout << ans->data;

    return 0;
}
```