

CÁCH ĐÁNH GIÁ ĐIỂM THỰC HÀNH

HỌC PHẦN: IT3150 – Project 1- 2023.1

I. Quy định, yêu cầu:

- Tài liệu và nội dung thực hành chấm điểm trên hệ thống:
<https://lab.soict.hust.edu.vn/>
- Bài tập trên lớp chấm điểm tự động (các bài không chấm trên hệ thống làm vào máy tính → làm báo cáo thực hành – Theo mẫu).
- Hạn nộp báo cáo trên Teams (Bài tập trên lớp + Bài tập về nhà): 1 tuần.

II. Đánh giá điểm thực hành

- Chuyên cần (đúng giờ, nghiêm túc trong giờ học) - Điểm danh trên Teams: 10%
- Báo cáo thực hành (bài tập trên lớp + Về nhà) theo mẫu nộp trên Teams: 40%
- Trắc nghiệm – Form trên Teams: 10%
- Kiểm tra thực hành: 40%. (Tiết 2,3 buổi thực hành thứ 5).

Điểm thưởng: 5% → 10% (Cho Mục 1,2 điểm TB từ 9-10).

Tham gia thực hành đúng giờ đầy đủ theo thời khóa biểu (nếu có lý do không đi thực hành đúng giờ được thì gửi mail xin phép thực hành bù trước 1 ngày qua mail hoalt@soict.hust.edu.vn, Tiêu đề: đăng ký học bù – IT3040 – MaLopTH.

Các kíp có thể bù:

TT	Thời gian, địa điểm, Tuần học	Mã nhóm	Mã lớp
1			
2			
3			
4			
5			
6			
7			

Nếu nghỉ không có lý do 3 buổi, không thực hành bù thì điểm chuyên cần, báo cáo và BTVN coi như 0 điểm thực hành.

Contents

Bài thực hành tuần số 3	
Bài tập 1.....	3
Bài tập 2.....	6
Bài tập 3.....	8
Bài tập 4.	11
Bài tập 5.....	16
Bài tập 6.....	22
Bài tập 7.....	28
Bài tập 8.....	32
Bài tập 9.....	36

Vấn đề: Ngăn xếp mô phỏng

Thực hiện một chuỗi các thao tác trên ngăn xếp, mỗi phần tử là một số nguyên:

- PUSH v: đẩy giá trị v vào ngăn xếp
- POP: xóa một phần tử ra khỏi ngăn xếp và in phần tử này ra thiết bị xuất chuẩn (in NULL nếu ngăn xếp trống)

Đầu vào

Mỗi dòng chứa một lệnh (thao tác) thuộc loại

- PUSH
- POP

đầu ra

- Viết kết quả của phép toán POP (mỗi kết quả ghi một dòng)

Source code

```
//Mai Minh Hoàng
//20215381
#include <bits/stdc++.h>
using namespace std;

int main() {
    stack<int> stacks; // Tạo một ngăn xếp (stack) chứa các số nguyên
    string s; // Biến để lưu trữ dòng đầu vào

    while (getline(cin, s)) { // Đọc dữ liệu dưới dạng dòng
        if (s[0] == '#') {
            break; // Nếu dòng bắt đầu bằng "#", kết thúc chương trình
        }
        if (s[0] == 'P' && s[1] == 'U') { //nếu lệnh là PUSH
            stringstream ss(s); // Tạo một stringstream để phân tích chuỗi
            string s1;
            int a; // Biến lưu trữ số nguyên
            ss >> s1 >> a; // Đọc số nguyên từ chuỗi
            stacks.push(a); // Thêm số nguyên vào ngăn xếp
        }
        else {
            if (!stacks.empty()) {
                cout << stacks.top() << "\n"; // In số ở đỉnh ngăn xếp
                stacks.pop(); // Loại bỏ số ở đỉnh ngăn xếp
            }
        }
    }
}
```

```

string s1;
int a; // Biến lưu trữ số nguyên
ss >> s1 >> a; // Đọc số nguyên từ chuỗi
stacks.push(a); // Thêm số nguyên vào ngăn xếp
}
else {
    if (!stacks.empty()) {
        cout << stacks.top() << "\n"; // In số ở đỉnh ngăn xếp
        stacks.pop(); // Loại bỏ số ở đỉnh ngăn xếp
    }
    else {
        cout << "NULL" << "\n"; // Nếu ngăn xếp rỗng, in "NULL"
    }
}
}

return 0;
}

```

Test:

Input

```

PUSH 1
PUSH 2
PUSH 3
POP
POP
PUSH 4
PUSH 5
POP
#

```

Correct output

```

3
2
5

```

User output

```

3
2
5

```

Input

POP
#

Correct output

NULL

User output

NULL

Vấn đề: Hàng đợi mô phỏng

Thực hiện một chuỗi các thao tác trên một hàng đợi, mỗi phần tử là một số nguyên:

- PUSH v: đẩy giá trị v vào hàng đợi
- POP: xóa một phần tử ra khỏi hàng đợi và in phần tử này ra thiết bị xuất chuẩn (in NULL nếu hàng đợi trống)

Đầu vào

Mỗi dòng chứa một lệnh (thao tác) thuộc loại

- Push v
- POP

đầu ra

- Viết kết quả của phép toán POP (mỗi kết quả ghi một dòng)

Source code

```

//C++
//Mai Minh Hoàng
//20215381
#include <bits/stdc++.h>

using namespace std;

int main() {
    queue<int> queues_81; // Tạo một hàng đợi (queue) chứa các số nguyên
    string s_81; // Biến để lưu trữ dòng đầu vào

    queue<int> queues_81; // Tạo một hàng đợi (queue) chứa các số nguyên
    string s_81; // Biến để lưu trữ dòng đầu vào

    while (getline(cin, s_81)) { // Đọc dữ liệu dạng dòng
        if (s_81[0] == '#') {
            break; // Nếu dòng bắt đầu bằng "#", kết thúc chương trình
        }
        if (s_81[0] == 'P' && s_81[1] == 'U') {
            stringstream ss_81(s_81); // Tạo một stringstream để phân tích chuỗi
            string s1_81; // Biến lưu trữ từ "PUSH" trong dòng
            int a_81; // Biến lưu trữ số nguyên
            ss_81 >> s1_81 >> a_81; // Đọc số nguyên từ chuỗi
            queues_81.push(a_81); // Thêm số nguyên vào hàng đợi
        }
        else {
            if (!queues_81.empty()) {
                cout << queues_81.front() << "\n"; // In số ở đầu hàng đợi
                queues_81.pop(); // Loại bỏ số ở đầu hàng đợi
            }
            else {
                cout << "NULL" << "\n"; // Nếu hàng đợi rỗng, in "NULL"
            }
        }
    }

    return 0;
}

```

Test:

Input

```
POP
#
```



Correct output

```
NULL
```



User output

```
NULL
```



Input

```
PUSH 1
PUSH 2
PUSH 3
POP
POP
PUSH 4
PUSH 5
POP
#
```



Correct output

```
1
2
3
```



User output

```
1
2
3
```

**Vấn đề: Kiểm tra dấu ngoặc đơn**

Cho một chuỗi chỉ chứa các ký tự (,), [,] {, }. Viết chương trình kiểm tra xem chuỗi biểu thức có đúng hay không.

Ví dụ

- ([{}])(): Chính xác
- ([{}])()(): không đúng

Đầu vào

- Một dòng chứa chuỗi (độ dài của chuỗi nhỏ hơn hoặc bằng 10^6) Một dòng chứa chuỗi (độ dài của chuỗi nhỏ hơn hoặc bằng 106)

đầu ra

- Viết 1 nếu dãy đúng, viết 0 nếu ngược lại

Source code

```
//C++
//Mai Minh Hoàng
//20215381
#include <bits/stdc++.h>
using namespace std;

stack<char> stacks_81; // Tạo một ngăn xếp (stack) chứa các ký tự
bool check_81(char t) { //hàm kiểm tra xem các ngoặc có đóng đúng không
    if (stacks_81.top() == '(' && t == ')') { //kiểm tra nếu t là )
        return true;
    }
    if (stacks_81.top() == '[' && t == ']') { //kiểm tra nếu t là ]
        return true;
    }
    if (stacks_81.top() == '{' && t == '}') { //kiểm tra nếu t là }
        return true;
    }
    return false;
}
```

```

int main() {
    string s_81; // Biến để lưu trữ chuỗi ký tự đầu vào
    cin >> s_81;
    int j_81 = 1; // Biến kiểm tra tính hợp lệ của dãy ký tự với 1 là đúng 0 là sai
    //nếu số ngoặc trong dãy là lẻ thì sai
    if (s_81.length() % 2 == 1) {
        j_81 = 0;
    }
    else {
        for (int i_81 = 0; i_81 < s_81.length(); i_81++) {
            if (s_81[i_81] == '(' || s_81[i_81] == '[' || s_81[i_81] == '{') { //nếu ký tự đang xét là ngoặc đóng
                if (check_81(s_81[i_81])) { //nếu ngoặc đóng đúng thì đưa ngoặc mở của nó ra khỏi stack
                    if (stacks_81.empty()) {
                        j_81 = 0;
                    }
                    stacks_81.pop();
                }
            }
            else {
                j_81 = 0; //ngược lại thì sai
            }
        }
        else {
            stacks_81.push(s_81[i_81]); //nếu là ngoặc mở thì thêm vào stack
        }
    }
}

```

```

if (stacks_81.size() > 0) { //nếu kết thúc mà stack còn ngoặc thì sai
    j_81 = 0;
}

cout << j_81; // In kết quả kiểm tra tính hợp lệ
return 0;
}

```


Source code

```
//Mai Minh Hoàng
//20215381
#include <bits/stdc++.h>

using namespace std;
//tạo class chứa trạng thái của 2 bình nước
class Jug_81 {
public:
    int a;
    int b;
    //hàm tạo
    Jug_81(int a, int b) {
        this->a = a;
        this->b = b;
    }
};

int A_81, B_81, c_81; //3 số đầu vào
queue<Jug_81*> queue_81[700]; //mảng các hàng đợi chứa trạng thái của 2 bình, chỉ số của mảng là số bước nhỏ nhất để
int dem_81 = 0; //đếm số bước nhỏ nhất
int status_81[1000][1000]; //mảng 2 chiều lưu trạng thái đã tồn tại của 2 bình nước

queue<Jug_81*> createJug_81(Jug_81* jug) { //hàm tạo tất cả trạng thái con của trạng thái hiện tại
    queue<Jug_81*> queueu_81; //hàng đợi để lưu các trạng thái con
```

```

if (status_81[A_81][jug->b] == 0) {//trạng thái con có thể thứ 1
    queuu_81.push(new Jug_81(A_81, jug->b));
    status_81[A_81][jug->b] = 1;
}
if (status_81[0][jug->b] == 0) {//trạng thái con có thể thứ 2
    queuu_81.push(new Jug_81(0, jug->b));
    status_81[0][jug->b] = 1;
}
if (status_81[jug->a][B_81] == 0) {//trạng thái con có thể thứ 3
    queuu_81.push(new Jug_81(jug->a, B_81));
    status_81[jug->a][B_81] = 1;
}
if (status_81[jug->a][0] == 0) {//trạng thái con có thể thứ 4
    queuu_81.push(new Jug_81(jug->a, B_81));
    status_81[jug->a][B_81] = 1;
}
if ((A_81 - jug->a) > jug->b) {//trạng thái con có thể thứ 5
    if (status_81[jug->a + jug->b][0] == 0) {
        queuu_81.push(new Jug_81(jug->a + jug->b, 0));
        status_81[jug->a + jug->b][0] = 1;
    }
}
if ((B_81 - jug->b) > jug->a) {//trạng thái con có thể thứ 6
    if (status_81[0][jug->a + jug->b] == 0) {
        queuu_81.push(new Jug_81(0, jug->a + jug->b));
        status_81[0][jug->a + jug->b] = 1;
    }
}
}

```

```

}
if ((A_81 - jug->a) < jug->b) {//trạng thái con có thể thứ 7
    if (status_81[A_81][jug->b - (A_81 - jug->a)] == 0) {
        queuu_81.push(new Jug_81(A_81, jug->b - (A_81 - jug->a)));
        status_81[A_81][jug->b - (A_81 - jug->a)] = 1;
    }
}
if ((B_81 - jug->b) < jug->a) {//trạng thái con có thể thứ 8
    if (status_81[jug->a - (B_81 - jug->b)][B_81] == 0) {
        queuu_81.push(new Jug_81(jug->a - (B_81 - jug->b), B_81));
        status_81[jug->a - (B_81 - jug->b)][B_81] = 1;
    }
}
return queuu_81;
}
//hàm để tìm theo chiều rộng
void find_81(int step) {
    if (queu_81[step].size() == 0) {//nếu bước này không có bất kỳ trạng thái chưa xét nào thì kết thúc, không tồn tại
        cout << -1;
        return;
    }
    while (!queu_81[step].empty()) {//khi trong hàng đợi còn có trạng thái chưa xét
        Jug_81* j = queu_81[step].front();//lấy ra trạng thái trên cùng
        if ((j->a == c_81) || (j->b == c_81)) {//nếu 1 trong 2 bình có giá trị mục tiêu
            if (dem_81 == 0) {//nếu trước đó chưa tìm được trạng thái nào

```

```

    }
    queu_81[step].pop(); // dù thế nào cũng lấy khỏi hàng đợi
    queue<Jug_81*> newq_81 = createJug_81(j); // tạo các trạng thái con của trạng thái năy
    while (newq_81.size() > 0) { // đệ quy với các trạng thái con
        queu_81[step + 1].push(newq_81.front());
        newq_81.pop();
    }
}
}
if (dem_81 != 0) {
    cout << dem_81; // in kết quả
}
else {
    find_81(step + 1);
}
}
}

int main() {
    cin >> A_81 >> B_81 >> c_81; // nhập
    if (A_81 < B_81) { // để giá trị A luôn nhỏ hơn B để hạn chế số trường hợp ở hàm create
        int temp_81 = A_81;
        A_81 = B_81;
        B_81 = temp_81;
    }
    Jug_81* root_81 = new Jug_81(0, 0); // trạng thái gốc
    queu_81[0].push(root_81);
    status_81[0][0] = 1;
    find_81(0); // tìm và in
    return 0;
}

```

Test:

Input

16 98 12



Correct output

28



User output

28



Input

136 288 24



Correct output

42



User output

42



Input

536 978 24



Correct output

676



User output

676



Vấn đề: Thao tác và truyền tải cây

Mỗi nút trên cây có id trường (mã định danh) là một số nguyên (id của các nút trên cây trùng lặp khác nhau)

Thực hiện 1 chuỗi hành động sau đây bao gồm các thao tác liên quan đến xây dựng cây và trình duyệt cây

- MakeRoot u: Tạo ra nút gốc của cây
- Insert uv: tạo mới 1 nút u và chèn vào danh sách nút cuối cùng của nút v (nếu nút có id bằng v không tồn tại hoặc nút có id bằng u đã tồn tại thì không chèn thêm mới)
- PreOrder: in ra thứ tự các nút trong trình duyệt cây theo thứ tự trước đó
- InOrder: in order các nút trong trình duyệt cây được phép theo thứ tự giữa

- PostOrder: in order các nút trong trình duyệt cây theo thứ tự sau

Dữ liệu : bao gồm các dòng, mỗi dòng là 1 trong số các hành động được mô tả ở trên, dòng cuối cùng được sử dụng là * (dấu kết thúc của dữ liệu).

Kết quả : ghi ra trên mỗi dòng, các nút thứ tự được phép truy cập theo thứ tự trước, giữa, sau của các hành động PreOrder, InOrder, PostOrder tương ứng được đọc từ đầu vào dữ liệu

Source code

```
//Mai Minh Hoàng
//20215381
#include <bits/stdc++.h>
using namespace std;

class Node_81 { //lớp Node chứa từng nút trên cây
public:
    int key_81; //giá trị của nó
    Node_81* child_81; //con đầu tiên của nó nếu có
    Node_81* right_81; //anh em của nó nếu nó là con đầu tiên
    Node_81(int key_81) {
        this->key_81 = key_81;
        this->child_81 = NULL;
        this->right_81 = NULL;
    }
};

Node_81* root_81; //nút gốc
map<int, Node_81*> maps_81; //tạo map để lưu khóa và giá trị không bị trùng và tiện kiểm tra tồn tại chưa

// Hàm chèn một nút mới
void insert_81(int u_81, int v_81) {
    auto it_81 = maps_81.find(u_81); //iterator để kiểm 1 khóa
    auto it1_81 = maps_81.find(v_81);
    if (it_81 != maps_81.end() || it1_81 == maps_81.end()) {
        return; // Nếu nút u_81 đã tồn tại hoặc nút v_81 không tồn tại, không chèn nút mới.
    }
}
```

```

auto it1_81 = maps_81.find(v_81);
if (it_81 != maps_81.end() || it1_81 == maps_81.end()) {
    return; // Nếu nút u_81 đã tồn tại hoặc nút v_81 không tồn tại, không chèn nút mới.
}
Node_81* node_81 = new Node_81(u_81);
if (it1_81->second->child_81 != NULL) { //nếu nút con của nó tồn tại
    Node_81* temp_81 = it1_81->second->child_81;
    while (true) { //thêm vào cuối danh sách con của nó
        if (temp_81->right_81 != NULL) {
            temp_81 = temp_81->right_81;
        }
        else {
            temp_81->right_81 = node_81; // Chèn nút u_81 vào danh sách các nút con của nút v_81.
            break;
        }
    }
}
else {
    it1_81->second->child_81 = node_81; //nếu chưa tồn tại thì node_81 là nút con đầu
}
maps_81.insert({ u_81, node_81 });
}

// Hàm thực hiện duyệt cây theo thứ tự trước (PreOrder)
void preOrder_81(Node_81* node_81) {
    if (node_81 == NULL) return; // Nếu nút là NULL, thoát khỏi hàm.
    cout << node_81->key_81 << " "; // In ra giá trị của nút.
    Node_81* clone_81 = node_81->child_81; // Xem xét nút con đầu tiên.

```

```
Node_81* clone_81 = node_81->child_81; // Xem xét nút con đầu tiên.

while (clone_81 != NULL) {
    preOrder_81(clone_81); // Tiếp tục duyệt cây trở lên nút con.
    clone_81 = clone_81->right_81; // Di chuyển đến nút con kế tiếp trong danh sách nút con.
}
}

// Hàm thực hiện duyệt cây theo thứ tự giữa (InOrder)
void inOrder_81(Node_81* node_81) {
    if (node_81 == NULL) return; // Nếu nút là NULL, thoát khỏi hàm.
    Node_81* clone_81 = node_81->child_81;
    if (clone_81 != NULL) {
        inOrder_81(clone_81); // Tiếp tục duyệt cây vào nút con đầu tiên.
    }

    cout << node_81->key_81 << " "; // In ra giá trị của nút.

    while (clone_81 != NULL && clone_81->right_81 != NULL) {
        clone_81 = clone_81->right_81; // Di chuyển đến nút con kế tiếp trong danh sách nút con.
        inOrder_81(clone_81); // Tiếp tục duyệt cây trở lên nút con.
    }
}

// Hàm thực hiện duyệt cây theo thứ tự sau (PostOrder)
void postOrder_81(Node_81* node_81) {
    if (node_81 == NULL) return; // Nếu nút là NULL, thoát khỏi hàm.
    Node_81* clone_81 = node_81->child_81;
```

```

// Hàm thực hiện duyệt cây theo thứ tự sau (PostOrder)
void postOrder_81(Node_81* node_81) {
    if (node_81 == NULL) return; // Nếu nút là NULL, thoát khỏi hàm.
    Node_81* clone_81 = node_81->child_81;
    while (clone_81 != NULL) {
        postOrder_81(clone_81); // Tiếp tục duyệt cây trở lên nút con.
        clone_81 = clone_81->right_81; // Di chuyển đến nút con kế tiếp trong danh sách nút con.
    }
    cout << node_81->key_81 << " "; // In ra giá trị của nút.
}

int main() {
    string s_81;

    while (getline(cin, s_81)) {
        if (s_81 == "") {
            break;
        }
        if (s_81[0] == 'M') {
            stringstream ss_81(s_81);
            string a_81;
            int b_81;
            ss_81 >> a_81 >> b_81;
            root_81 = new Node_81(b_81);
            maps_81.insert({ b_81, root_81 });
        }
    }
}

```

```

    }
    else if (s_81[0] == 'P') {
        if (s_81[1] == 'r') {
            preOrder_81(root_81); // Thực hiện duyệt cây theo thứ tự trước và in ra.
            cout << "\n";
        }

        else {
            postOrder_81(root_81); // Thực hiện duyệt cây theo thứ tự sau và in ra.
            cout << "\n";
        }
    }
    else if (s_81[0] == 'I' && s_81[2] == 'O') {
        inOrder_81(root_81); // Thực hiện duyệt cây theo thứ tự giữa và in ra.
        cout << "\n";
    }

    else {
        stringstream ss_81(s_81);
        string a_81;
        int b_81, c_81;
        ss_81 >> a_81 >> b_81 >> c_81;
        insert_81(b_81, c_81); // Thêm một nút mới và kết nối nút con với nút cha.
    }
}
return 0;
}

```

Test:

Input

```

MakeRoot 10
Insert 11 10
Insert 1 10
Insert 3 10
InOrder
Insert 5 11
Insert 4 11
Insert 8 3
PreOrder
Insert 2 3
Insert 7 3
Insert 6 4
Insert 9 4
InOrder
PostOrder
*

```

Correct output

```
11 10 1 3
10 11 5 4 1 3 8
5 11 6 4 9 10 1 8 3 2 7
5 6 9 4 11 1 8 2 7 3 10
```

User output

```
11 10 1 3
10 11 5 4 1 3 8
5 11 6 4 9 10 1 8 3 2 7
5 6 9 4 11 1 8 2 7 3 10
```

```
Insert 5 11
Insert 4 11
Insert 8 3
Insert 2 3
Insert 7 3
Insert 6 4
Insert 9 4
InOrder
PostOrder
PreOrder
*
```

Correct output

```
5 11 6 4 9 10 1 8 3 2 7
5 6 9 4 11 1 8 2 7 3 10
10 11 5 4 6 9 1 3 8 2 7
```

User output

```
5 11 6 4 9 10 1 8 3 2 7
5 6 9 4 11 1 8 2 7 3 10
10 11 5 4 6 9 1 3 8 2 7
```

Vấn đề: Cây gia phả

Cho một cây phả hệ được biểu thị bằng quan hệ con-cha mẹ (c,p) trong đó c là con của p. Thực hiện các truy vấn về cây gia phả:

- hậu duệ <name>: trả về số hậu duệ của <name> đã cho
- thế hệ <name>: trả về số thế hệ con cháu của <name> đã cho

Lưu ý: tổng số người trong gia đình nhỏ hơn hoặc bằng 104

Đầu vào

Chứa hai khối. Khối đầu tiên chứa thông tin về cha-con, bao gồm các dòng (kết thúc bằng một dòng chứa ***), mỗi dòng chứa: <child> <parent> trong đó <child> là một chuỗi thể hiện tên của con và <parent> là một chuỗi đại diện cho tên của cha mẹ. Khối thứ hai chứa các dòng (kết thúc bằng một dòng chứa ***), mỗi dòng chứa hai chuỗi <cmd> và <param> trong đó <cmd> là lệnh (có thể là hậu duệ hoặc thế hệ) và <param> là chuỗi đã cho tên của người tham gia truy vấn.

đầu ra

Mỗi dòng là kết quả của một truy vấn tương ứng.

Source code

```

//Mai Minh Hoàng
//20215381
#include <bits/stdc++.h>
using namespace std;

class FTree_81 { //lớp để lưu 1 người
public:
    string name_81; //tên
    vector<FTree_81*> descendants_81; //con trực tiếp
    FTree_81* parent_81; //cha mẹ (nếu có)
    FTree_81(string name_81) {
        this->name_81 = name_81;
    }
};

vector<FTree_81*> Persons_81; //lưu tất cả người đã tồn tại từ lúc duyệt

// Hàm tìm một người trong cây gia đình bằng tên
FTree_81* findPerson_81(string name_81) { //tìm xem người đó tồn tại chưa
    if (!Persons_81.empty()) {
        for (int i = 0; i < Persons_81.size(); i++) {
            if (Persons_81[i]->name_81 == name_81) {
                return Persons_81[i]; //nếu tồn tại thì trả về đối tượng chứa người đó
            }
        }
    }
}

```


// Hàm đệ quy để đếm số người con

```
int printDescentdants_81(FTree_81* p_81) {
    int dem_81 = 0;
    if (p_81->descendants_81.size() == 0 || p_81->descendants_81.empty()) {
        return 1;
    }
    for (int i = 0; i < p_81->descendants_81.size(); i++) {
        dem_81 = dem_81 + printDescentdants_81(p_81->descendants_81[i]);
        if (p_81->descendants_81[i]->descendants_81.size() > 0) {
            dem_81++;
        }
    }
    return dem_81;
}
```

```
int maxs_81 = 0;
```

// Hàm đệ quy để tìm số thế hệ tối đa

```
void printGeneration_81(FTree_81* p_81, int j_81) {
    if (p_81->descendants_81.size() == 0 || p_81->descendants_81.empty()) {
        if (j_81 > maxs_81) {
            maxs_81 = j_81;
        }
        return;
    }
    vector<FTree_81*> l_81;
    for (int i = 0; i < p_81->descendants_81.size(); i++) {
```

```

vector<FTree_81*> l_81;
for (int i = 0; i < p_81->descendants_81.size(); i++) {
    printGeneration_81(p_81->descendants_81[i], j_81 + 1);
}
}

int main() {
    string s_81;
    while (getline(cin, s_81)) { //hàm nhập đầu vào
        if (s_81 == "***") { //gặp dòng này thì dừng
            break;
        }
        stringstream ss_81(s_81); //phân tách 1 dòng
        string child_81; //tên con
        string parent_81; //tên cha
        ss_81 >> child_81 >> parent_81;

        FTree_81* childd_81 = findPerson_81(child_81); //xem coi người này tồn tại chưa
        FTree_81* parentt_81 = findPerson_81(parent_81);
        if (childd_81 == NULL) { //nếu người có tên này chưa tồn tại thì tạo người mới
            childd_81 = new FTree_81(child_81);
            Persons_81.push_back(childd_81); //thêm vào vector
        }
        if (parentt_81 == NULL) { //nếu người có tên này chưa tồn tại thì tạo người mới
            parentt_81 = new FTree_81(parent_81);
            Persons_81.push_back(parentt_81); //thêm vào vector chứa tất cả mọi người
        }
        parentt_81->descendants_81.push_back(childd_81); //thêm đối tượng con vào vector các con của cha
    }
}

```

```

parentt_81->descendants_81.push_back(childd_81);//thêm đối tượng con vào vector các con của cha

    childd_81->parent_81 = parentt_81;
}

while (getline(cin, s_81)) //nhập vào các lệnh in
    if (s_81 == "***") {
        break;
    }
    string s1_81, s2_81;
    stringstream ss_81(s_81);
    ss_81 >> s1_81 >> s2_81;
    FTree_81* p_81 = findPerson_81(s2_81);
    if (s1_81.length() > 5) {
        if (s1_81[0] == 'd') {
            cout << printDescentdants_81(p_81) << "\n";
        }
        else {
            printGeneration_81(p_81, 0);
            cout << maxs_81 << "\n";
        }
    }
}
return 0;
}

```

Test:

Input

```

Peter Newman
Michael Thomas
John David
Paul Mark
Stephan Mark
Pierre Thomas
Mark Newman
Bill David
David Newman
Thomas Mark
***
descendants Newman
descendants Mark
descendants David
generation Mark
***

```

Correct output

```
10
5
2
2
```

User output

```
10
5
2
2
```

BST - Chèn và truyền tải đặt hàng trước

Cho một BST được khởi tạo bởi NULL. Thực hiện một chuỗi các thao tác trên BST bao gồm:

- chèn k: chèn khóa k vào BST (không chèn nếu khóa k tồn tại)

Đầu vào

- Mỗi dòng chứa lệnh có dạng: "insert k"
- Đầu vào được kết thúc bằng một dòng chứa #

đầu ra

- Viết chuỗi khóa của các nút được duyệt qua theo thứ tự trước (cách nhau bằng ký tự SPACE)

Source code

```

//Mai Minh Hoàng
//20215381
#include <bits/stdc++.h>
using namespace std;

// Định nghĩa lớp Node_81 đại diện cho một nút trong cây BST
class Node_81
{
public:
    int root_81;    // Giá trị của nút
    Node_81 *leftNode_81; // Con trái
    Node_81 *rightNode_81; // Con phải

    // Constructor để khởi tạo một nút với giá trị root_81
    Node_81(int root_81) {
        this->root_81 = root_81;
        this->leftNode_81 = NULL;
        this->rightNode_81 = NULL;
    }
};

// Hàm in ra giá trị của các nút trong cây theo thứ tự trước (pre-order)
void printt(Node_81* node_81) {
    cout << node_81->root_81; // In giá trị của nút
    cout << " ";
    if (node_81->leftNode_81 != NULL) {
        // Hàm in ra giá trị của các nút trong cây theo thứ tự trước (pre-order)
        void printt(Node_81* node_81) {
            cout << node_81->root_81; // In giá trị của nút
            cout << " ";
            if (node_81->leftNode_81 != NULL) {
                printt(node_81->leftNode_81); // Gọi đệ quy để in nút con bên trái
            }
            if (node_81->rightNode_81 != NULL) {
                printt(node_81->rightNode_81); // Gọi đệ quy để in nút con bên phải
            }
        }
    }
}

```

```
// Hàm để chèn một nút node_81 vào cây BST với gốc là Root_81
void insert(Node_81* node_81, Node_81* Root_81) {
    if (Root_81 == NULL) {
        Root_81 = node_81; // Nếu gốc là NULL, thì node_81 trở thành gốc
        return;
    }
    if (node_81->root_81 == Root_81->root_81) {
        return; // Không chèn nếu giá trị đã tồn tại
    }
    if (node_81->root_81 < Root_81->root_81) {
        if (Root_81->leftNode_81) {
            insert(node_81, Root_81->leftNode_81); // Gọi đệ quy để chèn vào cây con trái
        }
        else {
            Root_81->leftNode_81 = node_81; // Chèn nút vào cây con trái
        }
    }
    else {
        if (Root_81->rightNode_81) {
            insert(node_81, Root_81->rightNode_81); // Gọi đệ quy để chèn vào cây con phải
        }
        else {
            Root_81->rightNode_81 = node_81; // Chèn nút vào cây con phải
        }
    }
    return;
}
```

```

int main()
{
    vector<Node_81*> vtr_81;
    string n_81;
    getline(cin, n_81);
    Node_81* rootNode_81 = new Node_81(stoi(n_81.substr(7))); // Khởi tạo gốc của cây
    vtr_81.insert(vtr_81.begin(), rootNode_81);
    while (getline(cin, n_81)) {
        if (n_81 == "#") {
            break;
        }
        Node_81* node_81 = new Node_81(stoi(n_81.substr(7))); // Tạo một nút mới với giá trị từ đầu vào
        insert(node_81, rootNode_81); // Chèn nút mới vào cây BST
    }
    printt(rootNode_81); // In ra giá trị của cây BST theo thứ tự trước

    return 0;
}

```

Test:

Input

```

insert 20
insert 10
insert 26
insert 7
insert 15
insert 23
insert 30
insert 3
insert 8
#

```

Correct output

```
20 10 7 3 8 15 26 23 30
```

User output

```
20 10 7 3 8 15 26 23 30
```

```

insert 7360
insert 5315
insert 4437
insert 2042
insert 3980
insert 5138
insert 1747
insert 2936
insert 3376
insert 9718
insert 4151
insert 4119
#

```

Correct output

```
4860 2814 268 1132 696 270 569 768 929 1910
```

User output

```
4860 2814 268 1132 696 270 569 768 929 1910
```

Vấn đề: Chèn vào giữa chuỗi

Cho dãy số nguyên a_1, a_2, \dots, a_n . Phần tử ở giữa được xác định là phần tử ở chỉ số $n/2$ nếu n chẵn và $n/2 + 1$ nếu ngược lại.

Thực hiện một chuỗi các hành động có dạng:

- THÊM v : thêm giá trị v vào ngay sau phần tử ở giữa dãy a_1, a_2, \dots, a_n .
- IN: in chuỗi ra stdout, các phần tử cách nhau bằng ký tự SPACE

Đầu vào

- Dòng 1: chứa số nguyên dương n ($1 \leq n \leq 100000$)
- Dòng 2: chứa n số nguyên dương a_1, a_2, \dots, a_n ($1 \leq a_i \leq 1000000$)

- Các dòng tiếp theo (số dòng có thể lên tới 100000), mỗi dòng chứa một hành động có định dạng trên

đầu ra

Viết (vào mỗi dòng) kết quả của thao tác IN tương ứng

Source code

```
//Mai Minh Hoàng
//20215381
#include <iostream>
#include <vector>
#include <string>
#include <sstream>
using namespace std;

int main() {
    int n_81; // Khai báo biến n_81 để lưu số lượng phần tử trong dãy
    cin >> n_81;
    vector<int> sequence_81; // Khai báo vector sequence_81 để lưu dãy số

    // Đọc n_81 số nguyên từ input và lưu vào vector sequence_81
    for (int i_81 = 0; i_81 < n_81; i_81++) {
        int ai_81;
        cin >> ai_81;
        sequence_81.push_back(ai_81);
    }

    cin.ignore(); // Đọc và bỏ qua dấu xuống dòng sau số nguyên

    string action_81;
    while (getline(cin, action_81)) {
        if (action_81 == "#") {
```

```

string action_81;
while (getline(cin, action_81)) {
    if (action_81 == "#") {
        break; // Thoát khỏi vòng lặp nếu gặp ký tự #
    }
    else if (action_81 == "PRINT") {
        // In ra dãy số trong vector sequence_81
        for (int i_81 = 0; i_81 < sequence_81.size(); i_81++) {
            cout << sequence_81[i_81];
            if (i_81 < sequence_81.size() - 1) {
                cout << " "; // In khoảng trắng sau mỗi số (trừ số cuối cùng)
            }
        }
        cout << endl; // In dấu xuống dòng
    }
}

```

```

    else {
        istringstream iss_81(action_81); // phân tích lệnh
        string cmd_81; // tên lệnh
        iss_81 >> cmd_81;
        if (cmd_81 == "ADD") {
            int v_81;
            iss_81 >> v_81; // tham số của lệnh
            int middle_81 = sequence_81.size() / 2;

            // Chèn giá trị v_81 vào vị trí giữa của dãy số
            if (sequence_81.size() % 2 == 0) {
                sequence_81.insert(sequence_81.begin() + middle_81, v_81);
            }
            else {
                sequence_81.insert(sequence_81.begin() + middle_81 + 1, v_81);
            }
        }
    }
}

return 0;
}

```

Test:

Input

```
2
8 4
ADD 6
ADD 5
ADD 4
PRINT
ADD 2
ADD 4
PRINT
#
```

Correct output

```
8 6 4 5 4
8 6 4 4 2 5 4
```

User output

```
8 6 4 5 4
8 6 4 4 2 5 4
```

```
2
1 4
ADD 5
ADD 6
ADD 10
ADD 6
ADD 2
ADD 7
ADD 7
ADD 4
ADD 8
ADD 4
PRINT
#
```

Correct output

```
1 5 10 2 7 8 4 4 7 6 6 4
```

User output

```
1 5 10 2 7 8 4 4 7 6 6 4
```

Vấn đề: Thao tác danh sách liên kết

Viết chương trình thực hiện công việc sau:

Xây dựng danh sách liên kết với các từ khóa được cung cấp ban đầu là chuỗi a_1 , Một₂, ..., Một_N, sau đó thực hiện các thao tác trên danh sách bao gồm: thêm 1 phần tử vào đầu, vào cuối danh sách, hoặc vào trước, vào sau 1 phần tử nào trong danh sách, hoặc loại bỏ 1 phần tử nào trong đó danh sách

Đầu vào

- Dòng 1: ghi số nguyên dương n ($1 \leq n \leq 1000$)
- Dòng 2: write các số nguyên dương a_1 , Một₂, ..., Một_N.
- Các dòng tiếp theo được dùng để thao tác (kết thúc bởi ký hiệu #) với các loại sau:
 - addlast k: thêm phần tử có khóa bằng k vào danh sách cuối cùng (nếu k tồn tại)
 - addfirst k: thêm phần tử có khóa bằng k vào đầu danh sách (nếu k tồn tại)
 - addafter uv: thêm phần tử có khóa bằng u vào sau phần tử có khóa bằng v trên danh sách (nếu v tồn tại trên danh sách và u không tồn tại)
 - add before uv: add phần tử có khóa bằng u vào trước phần tử có khóa bằng v trên danh sách (nếu v tồn tại trên danh sách và u tồn tại)
 - loại bỏ k: loại bỏ phần tử có khóa khỏi danh sách
 - đảo ngược: đảo ngược thứ tự các phần tử của danh sách (không thể cung cấp các phần tử mới, chỉ thay đổi liên kết kết nối)

đầu ra

- Ghi lại khóa chuỗi của danh sách sau 1 chuỗi thao tác lệnh đã chọn

Source code

```
//Mai Minh Hoàng
//20215381
#include <bits/stdc++.h>

using namespace std;
int n_81;          // Số lượng phần tử trong danh sách
vector<int> listt_81; // Danh sách các phần tử

int main() {
    string t_81;
    getline(cin, t_81);
    stringstream ss_81(t_81);
    ss_81 >> n_81;
    string s_81;
    for (int i_81 = 0; i_81 < n_81; i_81++) { //nhập danh sách ban đầu n phần tử
        int ai_81;
        cin >> ai_81;
        listt_81.push_back(ai_81);
    }
    while (getline(cin, s_81)) {
        if (s_81 == "#") { // Nếu gặp dấu #_81, kết thúc vòng lặp
            break;
        }
    }
}
```

```

while (getline(cin, s_81)) {
    if (s_81 == "#") { // Nếu gặp dấu #_81, kết thúc vòng lặp
        break;
    }
    if (s_81.length() >= 5) {
        if (s_81[0] == 'a') { // Xử lý lệnh 'add'
            if (s_81[3] == 'l') { // add last
                stringstream ss_81(s_81);
                string act_81;
                int k_81;
                ss_81 >> act_81 >> k_81;
                auto iter_81 = find(listt_81.begin(), listt_81.end(), k_81);
                if (iter_81 == listt_81.end()) {
                    listt_81.push_back(k_81);
                };
            }
            else if (s_81[3] == 'f') { // add first
                stringstream ss_81(s_81);
                string act_81;
                int k_81;
                ss_81 >> act_81 >> k_81;
                auto iter_81 = find(listt_81.begin(), listt_81.end(), k_81);
                if (iter_81 == listt_81.end()) {
                    listt_81.insert(listt_81.begin(), k_81);
                };
            }
            else if (s_81[3] == 'a') { // add after
                stringstream ss_81(s_81);

```

```

    if (iter_81 == listt_81.end()) {
        listt_81.insert(listt_81.begin(), k_81);
    };
}

else if (s_81[3] == 'a') { //add after
    stringstream ss_81(s_81);
    string act_81;
    int u_81, v_81;
    ss_81 >> act_81 >> u_81 >> v_81;
    auto iter_81 = find(listt_81.begin(), listt_81.end(), u_81);
    auto iter1_81 = find(listt_81.begin(), listt_81.end(), v_81);
    if (iter_81 == listt_81.end() && iter1_81 != listt_81.end()) {
        for (int i_81 = 0; i_81 < listt_81.size(); i_81++) {
            if (listt_81[i_81] == v_81) {
                listt_81.insert(listt_81.begin() + (i_81 + 1), u_81);
            }
        }
    }
    else {
        continue;
    }
}

else { //add before
    stringstream ss_81(s_81);
    string act_81;
    int u_81, v_81;

```

```

else {//add before
    stringstream ss_81(s_81);
    string act_81;
    int u_81, v_81;
    ss_81 >> act_81 >> u_81 >> v_81;
    auto iter_81 = find(listt_81.begin(), listt_81.end(), u_81);
    auto iter1_81 = find(listt_81.begin(), listt_81.end(), v_81);
    if (iter_81 == listt_81.end() && iter1_81 != listt_81.end()) {
        listt_81.insert(iter1_81, u_81);
    }
    else {
        continue;
    }
}
}
else if (s_81[2] == 'm') {// Xử lý lệnh 'remove'
    stringstream ss_81(s_81);
    string act_81;
    int k_81;
    ss_81 >> act_81 >> k_81;
    auto iter_81 = find(listt_81.begin(), listt_81.end(), k_81);
    if (iter_81 != listt_81.end()) {
        for (int i_81 = 0; i_81 < listt_81.size(); i_81++) {
            if (listt_81[i_81] == k_81) {
                listt_81.erase(listt_81.begin() + i_81);
            }
        }
    }
}
}

```



```

ss_81 >> act_81 >> k_81;
auto iter_81 = find(listt_81.begin(), listt_81.end(), k_81);
if (iter_81 != listt_81.end()) {
    for (int i_81 = 0; i_81 < listt_81.size(); i_81++) {
        if (listt_81[i_81] == k_81) {
            listt_81.erase(listt_81.begin() + i_81);
        }
    }
}
else {
    reverse(listt_81.begin(), listt_81.end()); // Xử lý lệnh 'reverse'
}
}
for (int i_81 = 0; i_81 < listt_81.size(); i_81++) {
    cout << listt_81[i_81] << " ";
}
return 0;
}

```

Test:

Input

```

5
5 4 3 2 1
addlast 3
addlast 10
addfirst 1
addafter 10 4
remove 1
#

```

Correct output

```
5 4 3 2 10
```

User output

```
5 4 3 2 10
```