Задача 1(Tree-1)

Вася пишет новую структуру данных — дерево. В узлах и листьях дерева хранятся строковые ключи. Каждый путь от корня до какого-нибудь узла можно записать, перечисляя последовательные ключи узлов. Типичный пример — иерархия папок в файловой системе. Вася уже выбрал способ хранения дерева:

```
#include <map>
#include <string>
#include <vector>
struct Node {
std::map<std::string, Node> children;
};
class Tree {
private:
Node root;
public:
bool Has(const std::vector<std::string>& node) const;
void Insert(const std::vector<std::string>& node);
void Delete(const std::vector<std::string>& node);
};
// Ваш код будет вставлен сюда
#include "your_code"
```

Не будем обсуждать, насколько это эффективно.

Ваша задача — написать реализации функций Has, Insert и Delete для этого класса. В примере с папками в файловой системе функция Has должна проверить, существует ли такая папка, функция Insert должна создать папку (возможно, с промежуточными родительскими папками), а Delete — удалить папку со всеми вложенными подпапками, если такая папка существует.

Можно считать, что вектор, передаваемый на вход этих функций, всегда непустой.

Задача 1(Tree-2)

Коля пишет класс «Дерево». Узел дерева может хранить целое число, а также знает о своём родителе и о своих потомках. У узла есть функция AddChild для добавления потомка с заданным числовым значением, а также функция Print для красивой печати поддерева начиная с этого узла.

Вот что получилось у Коли:

```
#include <iostream>
#include <vector>
class TreeNode {
private:
int value;
TreeNode* root = nullptr;
std::vector<TreeNode*> children;
public:
TreeNode(int val): value(val) {
}
TreeNode(const TreeNode&) = delete;
TreeNode& operator=(const TreeNode&) = delete;
TreeNode* AddChild(int child_value) {
auto node = new TreeNode(child_value);
node->root = this;
children.push_back(node);
return node;
}
void Print(int depth = 0) const {
for (int i = 0; i < depth; ++i) {
std::cout << " ";
}
std::cout << "- " << value << "\n";
for (const auto& child : children) {
child->Print(depth + 1);
}
}
};
```

Использоваться этот класс будет примерно так:

```
#include "tree.h"
int main() {
TreeNode root(1);
auto left_son = root.AddChild(10);
```

```
    auto middle_son = root.AddChild(20);
    auto right_son = root.AddChild(30);

    left_son->AddChild(100);
    left_son->AddChild(200);

    root.Print();
}
```

Однако эта работающая на первый взгляд тестовая программа падает, если её собрать с адресным санитайзером. Исправьте код класса `TreeNode`, чтобы решить эту проблему.