Computation Tree

content: Structure; Pointer; Memory Management
Yangye Chen, 2024.10

Background

Tree is a very important data structure, a binary tree in its simplest form is composed of a root node and two children. This assignment will start with one of the simplest trees and look at accessing the root and child nodes via pointers. Also, deep/shallow copy is a very important point to know. I hope that you will be able to understand the tree as a data structure through this problem and become proficient in using pointers and deep/shallow copy.

Description

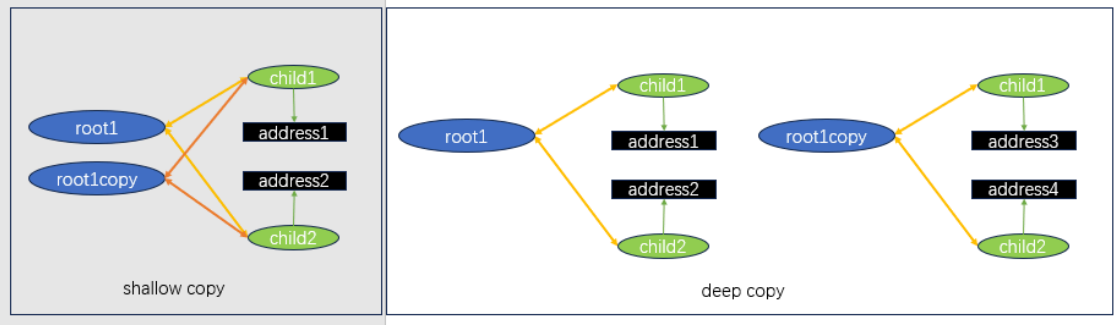
In this question, you need to complete a simple **Computation Tree**, where the **root node** holds the **char** data type's operation symbols (including '+', '-', '*', '/'), and a pointer to the left/right **child node**; the **child node** holds data of type **float** and a pointer to the **root node**.

The following operation should be supported by **Computation Tree**:

1. `setLeft`: Deletes an existing left node and updates the value and pointer.
2. `setRight`: Deletes an existing right node and updates the value and pointer.
3. `compute`: Calculate the result of this computational tree.
4. `print`: Print the formula according to the fixed format.
5. `deepCopy`: Deep copy of both left and right child nodes.
6. `shallowCopy`: Share the original left and right child nodes.

Hint

- The following picture will roughly illustrate the difference between deep copy and shallow copy:



Test Cases

- Case 1-8 : Basic operation of **Computation Tree**
- Case 9-10 : memory management(`deepCopy` and `shallowCopy` are only checked in this cases)

Template

```
//PREPRND BEGIN
#include <iostream>
#include <stdexcept>

using namespace std;
struct ChildNode;
struct OperationNode;

struct ChildNode {
    float value;
    OperationNode* parent;

    ChildNode(float val) : value(val), parent(nullptr) {}
};

struct OperationNode {
    char op;
    ChildNode* left;
    ChildNode* right;

    OperationNode(char operation)
        : op(operation), left(nullptr), right(nullptr) {}

    ~OperationNode();

    void setLeft(ChildNode* leftChild);

    void setRight(ChildNode* rightChild);

    float compute() const;

    void print() const;

    OperationNode* deepCopy() const;

    OperationNode* shallowCopy() const;
};

OperationNode::~OperationNode() {
    if (left && left->parent == this) delete left;
    if (right && right->parent == this) delete right;
}
//PREPRND END

// TEMPLATE BEGIN
```

```cpp
// Deletes an existing left node and updates the value and pointer
void OperationNode::setLeft(ChildNode* leftChild) {
    //TODO
}

// Deletes an existing right node and updates the value and pointer
void OperationNode::setRight(ChildNode* rightChild) {
    //TODO
}

// Calculate the result of this computational tree
float OperationNode::compute() const {
    //TODO
}

// Print the formula according to the fixed format: Parent(*): 10.5 * 2 = 21
void OperationNode::print() const {
    //TODO
}

//Deep copy of both left and right child nodes
OperationNode* OperationNode::deepCopy() const {
    OperationNode* copy = new OperationNode(op);
    //TODO
}

//Share the original left and right child nodes
OperationNode* OperationNode::shallowCopy() const {
    OperationNode* copy = new OperationNode(op);
    //TODO
}
//TEMPLATE END

//APPEND BEGIN
void testCalculationTree() {
    OperationNode root('*');

    ChildNode* leftChild = new ChildNode(5.5f);
    ChildNode* rightChild = new ChildNode(3.2f);
    root.setLeft(leftChild);
    root.setRight(rightChild);

    root.print();  // Parent(*): 5.5 * 3.2 = 17.76

    OperationNode* deepCopy = root.deepCopy();
    root.setLeft(new ChildNode(10.5f));
    root.setRight(new ChildNode(2.0f));

    cout << "Deep Copy Result: ";
    deepCopy->print();  // Parent(*): 5.5 * 3.2 = 17.76
    cout << "Modified Original Result: ";
    root.print();       // Parent(*): 10.5 * 2 = 21

    OperationNode* shallowCopy = root.shallowCopy();
    root.left->value = 1.0f;
```

```
    root.right->value = 2.0f;
    cout << "Shallow Copy Result: ";
    shallowCopy->print(); // Parent(*): 1 * 2 = 2

    delete deepCopy;
    delete shallowCopy;
}
int main() {
    testCalculationTree();
    return 0;
}
//APPEND END
```