22336216-陶宇卓-Project4-实验报告

程序功能简要说明

一个表达式和一棵二叉树之间，存在着自然的对应关系。

假设算术表达式Expression内可以含有变量(a~z)、常量(0~9)和二元运算符(+,–,*,/,^(乘幂))。可以实现以下操作：

(1)　ReadExpr(E)——以字符序列的形式输入语法正确的前缀表达式并构成表达式E；

(2)　WritrExpr(E)——用带括弧的中缀表示式输出表达式E；

(3)　Assign(V, c)——实现对变量V的赋值(V = c)，变量的初值为0；

(4)　Value(E)——对算术表达式E求值；

(5)　CompoundExpr(P, E1, E2)——构成一个新的复合表达式(E1)P(E2)。

除此之外，还可以实现：

(1)　增加求偏导数运算Diff(E, V)——求表达式E对变量V的导数；

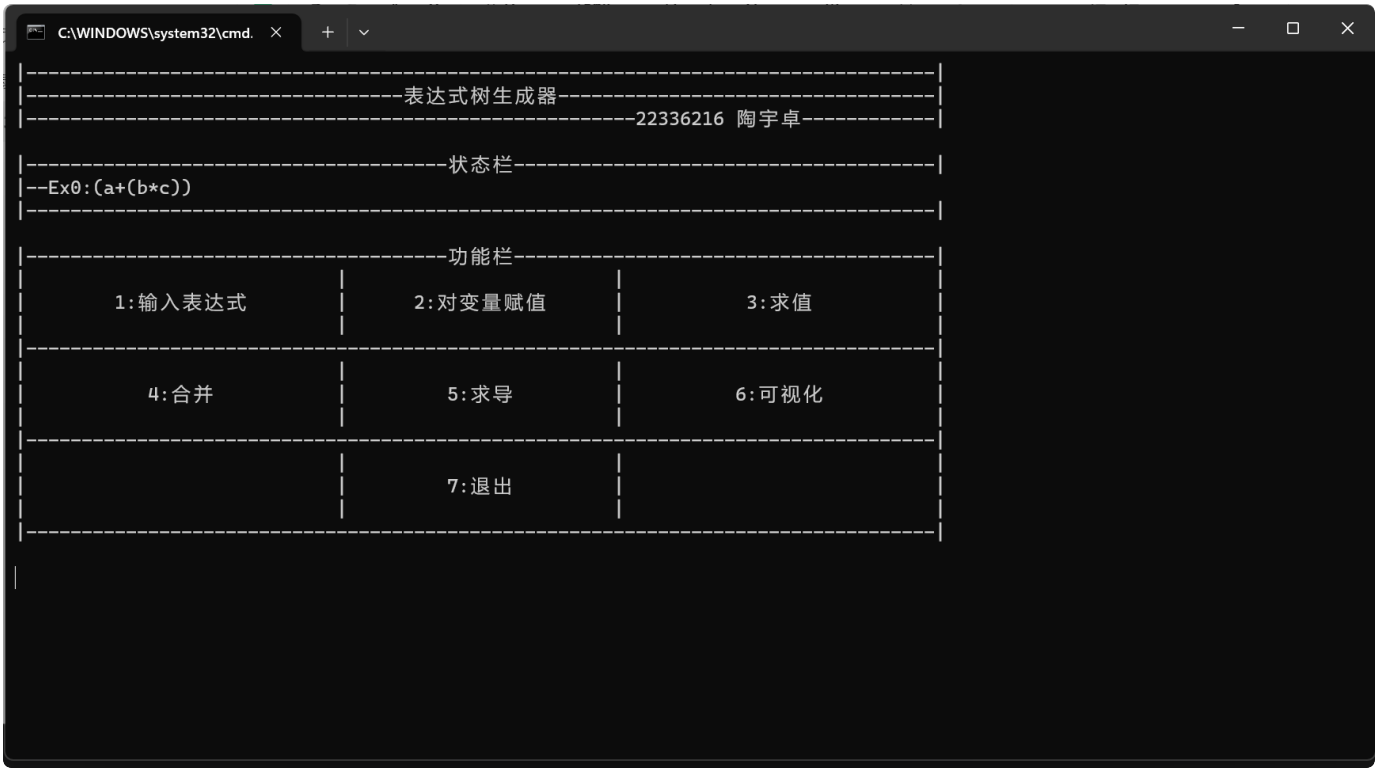
(2)　在表达式中添加三角函数等初等函数的操作；

(3)　增加常数合并操作MergeConst(E)——合并表达式E中所有常数运算。例如，对表达式E = (2+3–a)*(b+3*4)进行合并常数的操作后，求得E = (5–a)*(b+12)；

注：以上扩展功能未实现完全，可能会有bug

程序运行截图，包括计算功能演示、部分实际运行结果展示、命令行或交互式界面效果等

输入表达式：


C:\WINDOWS\system32\cmd.
表达式树生成器
22336216 陶宇卓
状态栏
--Ex0:(a+(b*c))
功能栏
1:输入表达式
2:对变量赋值
3:求值
4:合并
5:求导
6:可视化
7:退出

赋值:


C:\WINDOWS\system32\cmd.
表达式树生成器
22336216 陶宇卓
状态栏
--Ex0:(2.000000+(3.000000*4.000000))
功能栏
1:输入表达式
2:对变量赋值
3:求值
4:合并
5:求导
6:可视化
7:退出

求值:


C:\WINDOWS\system32\cmd.
|--|
|----------------------------------表达式树生成器----------------------------------|
|---22336216 陶宇卓------------|
|--------------------------------------状态栏---------------------------------------|
|--Ex0:(2.000000+(3.000000*4.000000))
|--|
|--------------------------------------功能栏---------------------------------------|
| 1:输入表达式 | 2:对变量赋值 | 3:求值 |
| 4:合并 | 5:求导 | 6:可视化 |
| 7:退出 |
|--|
3
|---------------------------请输入表达式编号（0开始）----------------------------|
0
结果为： 14
请按任意键继续. . .

合并表达式：


C:\WINDOWS\system32\cmd.
|--|
|----------------------------------表达式树生成器----------------------------------|
|---22336216 陶宇卓------------|
|--------------------------------------状态栏---------------------------------------|
|--Ex0:(2.000000+(3.000000*4.000000))
|--Ex1:(9-1)
|--Ex2:((a+(b*c))+(9-1))
|--|
|--------------------------------------功能栏---------------------------------------|
| 1:输入表达式 | 2:对变量赋值 | 3:求值 |
| 4:合并 | 5:求导 | 6:可视化 |
| 7:退出 |
|--|

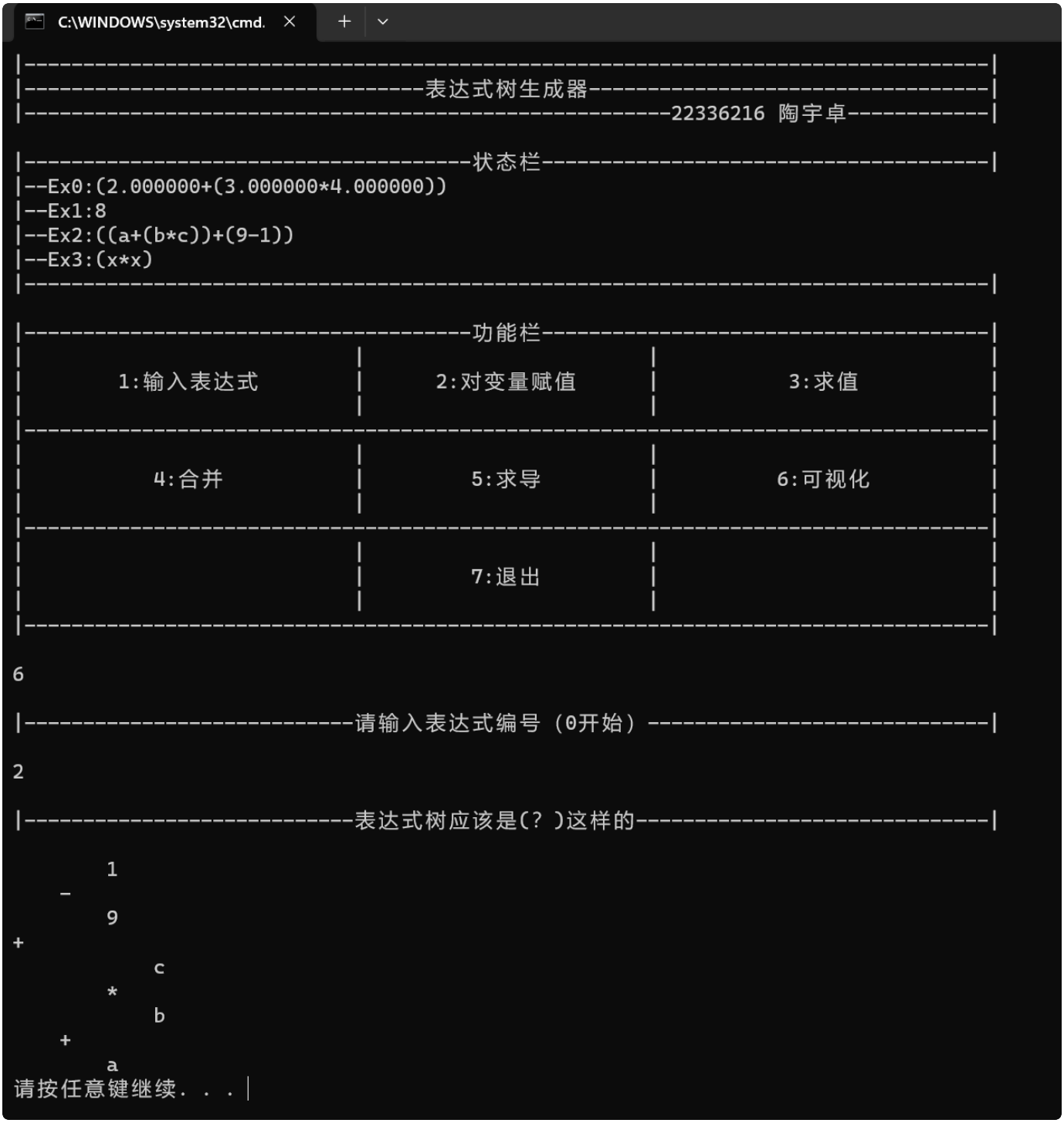
合并常数：


C:\WINDOWS\system32\cmd.
|----------表达式树生成器----------|
|----------22336216 陶宇卓----------|
|----------状态栏----------|
|--Ex0:(2.000000+(3.000000*4.000000))
|--Ex1:8
|--Ex2:((a+(b*c))+(9-1))
|----------功能栏----------|
1:输入表达式
2:对变量赋值
3:求值
4:合并
5:求导
6:可视化
7:退出

求导：

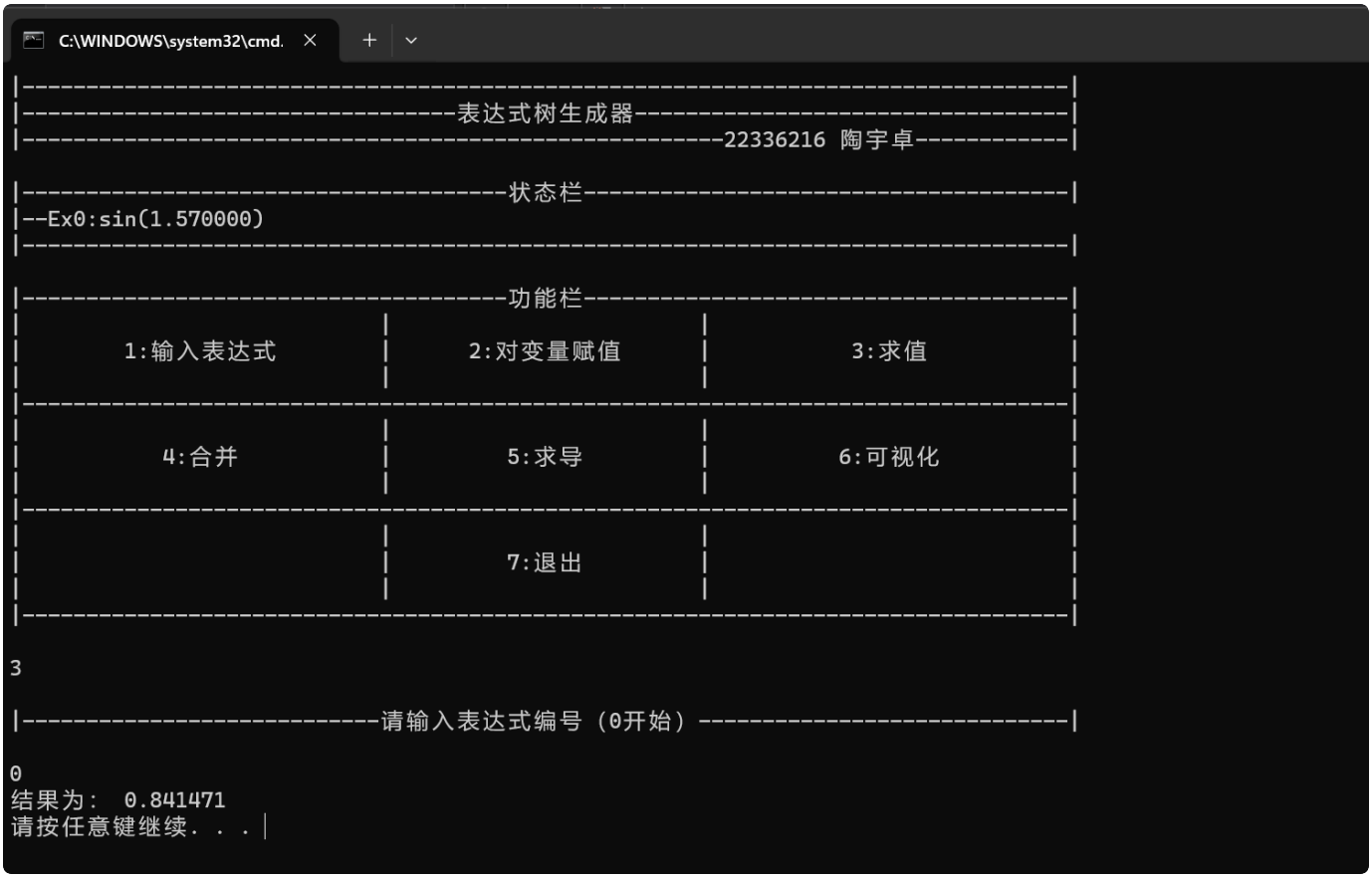

C:\WINDOWS\system32\cmd.
|--Ex1:8
|--Ex2:((a+(b*c))+(9-1))
|--Ex3:(x*x)
|功能栏
1:输入表达式
2:对变量赋值
3:求值
4:合并
5:求导
6:可视化
7:退出
5
|请输入表达式编号（0开始）
3
|请输入变量名
x
|求偏导结果为
((1*x)+(x*1))
请按任意键继续. . .

可视化：


C:\WINDOWS\system32\cmd.
|----------表达式树生成器----------|
|----------22336216 陶宇卓----------|
|----------状态栏----------|
|--Ex0:(2.000000+(3.000000*4.000000))
|--Ex1:8
|--Ex2:((a+(b*c))+(9-1))
|--Ex3:(x*x)
|----------功能栏----------|
1:输入表达式
2:对变量赋值
3:求值
4:合并
5:求导
6:可视化
7:退出
6
|----------请输入表达式编号（0开始）----------|
2
|----------表达式树应该是(?)这样的----------|
1
-
9
+
c
*
b
+
a
请按任意键继续. . .

（旋转90°）

三角函数:



部分关键代码及其说明

表达式树节点 C++

```C++
struct TreeNode {
    string word;
    TreeNode* left;
    TreeNode* right;
    TreeNode() : word(""), left(nullptr), right(nullptr) {}
    TreeNode(string x) : word(x), left(nullptr), right(nullptr) {}
    TreeNode(string x, TreeNode* left, TreeNode* right) : word(x), left(left), right(right) {}
};
```

储存多个表达式的容器 C++

```C++
unordered_map<int, TreeNode*> roots;
unordered_map<int, TreeNode*> copyroots;
```

读取表达式并创建表达式树 C++

```cpp
TreeNode* ReadExpr(string prestr) {
    prestr.erase(std::remove_if(prestr.begin(), prestr.end(), [](char c) { return std::isspace(static_cast<unsigned char>(c)); }), prestr.end());
    if (!isValidString(prestr)) return nullptr;
    stack<TreeNode*> postfix;
    int i = 0;
    for (i = prestr.size() - 1; i >= 0; i--) {
        if (isalnum(prestr[i])) {
            TreeNode* newnode = new TreeNode(string(1,prestr[i]));
            postfix.push(newnode);
        }
            if (prestr[i] == '+' || prestr[i] == '-' || prestr[i] == '*' || prestr[i] == '/' || prestr[i] == '^') {
                if (postfix.size() < 2) break;
                TreeNode* leftn = postfix.top(); postfix.pop();
                TreeNode* rightn = postfix.top(); postfix.pop();
                TreeNode* newnode = new TreeNode(string(1, prestr[i]), leftn, rightn);
                postfix.push(newnode);
            }
            if (prestr[i] == 'n' && i - 2 >= 0 && prestr[i - 1] == 'i' && prestr[i - 2] == 's') {
                postfix.pop();
                if (postfix.size() < 1) break;
                TreeNode* leftn = postfix.top(); postfix.pop();
                TreeNode* newnode = new TreeNode("sin", leftn, nullptr);
                postfix.push(newnode);
                i = i - 2;
            }
            if (prestr[i] == 's' && i - 2 >= 0 && prestr[i - 1] == 'o' && prestr[i - 2] == 'c') {
                postfix.pop();
                if (postfix.size() < 1) break;
                TreeNode* leftn = postfix.top(); postfix.pop();
                TreeNode* newnode = new TreeNode("cos", leftn, nullptr);
                postfix.push(newnode);
                i = i - 2;
            }
            if (prestr[i] == 'n' && i - 2 >= 0 && prestr[i - 1] == 'a' && prestr[i - 2] == 't') {
                postfix.pop();
                if (postfix.size() < 1) break;
                TreeNode* leftn = postfix.top(); postfix.pop();
                TreeNode* newnode = new TreeNode("tan", leftn, nullptr);
                postfix.push(newnode);
```

```
                i = i - 2;
            }

        }
        if (i == -1) return postfix.top();
        else return nullptr;

    }
```

打印表达式树 C++

```cpp
void printTree(TreeNode* root, int indent = 0) {
    if (root == nullptr) {
        return;
    }

    // 输出右子树，右子树在上方，所以缩进增加
    printTree(root->right, indent + 4);

    // 输出当前节点
    for (int i = 0; i < indent; ++i) {
        std::cout << " ";
    }
    std::cout << root->word << std::endl;

    // 输出左子树
    printTree(root->left, indent + 4);
}

```

转化为带括号的中缀表达式 C++

```cpp
string WriteExpr(TreeNode* root) {
    if (!root) return "";
    if (root->word == "+" || root->word == "-" || root->word == "*" || root->word == "/" || root->word == "^" ) {
        return "(" + WriteExpr(root->left) + root->word + WriteExpr(root->right) + ")";
    }
    if (root->word == "sin" || root->word == "cos" || root->word == "tan") {
        return root->word + "(" + WriteExpr(root->left) + ")";
    }
    return root->word;
}

```

赋值 C++

```cpp
void Assign(TreeNode* root, string v, double c) {
    if (root == nullptr) {
        return;
    }

    // 如果当前节点是变量节点，并且与给定的变量v相等，则替换为c
    if (root->word == v) {
        root->word = to_string(c); // 将整数转换为字符
    }

    // 递归处理左子树和右子树
    Assign(root->left, v, c);
    Assign(root->right, v, c);
}

```

求值 C++

```cpp
double Value(TreeNode* root) {//后序遍历求值
    if (root == nullptr) {
        return 0.0;
    }
    if (root->word.size() == 1 && isalpha(atoi(root->word.c_str()))) {
        return 0.0;
    }
    // 如果节点是操作符
    if (root->word == "+" || root->word == "-" || root->word == "*" || root->word == "/" || root->word == "^" || root->word == "sin" || root->word == "cos" || root->word == "tan") {
        double leftValue = Value(root->left);
        double rightValue = Value(root->right);

        // 根据操作符进行相应的运算
        if (root->word == "+")
            return leftValue + rightValue;
        if (root->word == "-")
            return leftValue - rightValue;
        if (root->word == "*")
            return leftValue * rightValue;
        if (root->word == "/") {
            // 避免除以0的情况
            if (rightValue != 0) {
                return leftValue / rightValue;
            }
            else {
                cout << "ERROR!" << endl;
                return 0; // 返回一个默认值
            }
        }
        if (root->word == "^") {
            if (leftValue == 0 && rightValue == 0) {
                return 1;
            }
            return pow(leftValue, rightValue);
        }
        if (root->word == "sin")
            return sin(leftValue);
        if (root->word == "cos")
            return cos(leftValue);
        if (root->word == "tan")
            return tan(leftValue);
    }
    else {
```

```
            // 如果节点是数字，直接返回其值
            return (double)stoi(root->word); // 假设表达式树中的数字都是单个数字字符
        }
        return 0;
    }

```

合并表达式 C++

```
TreeNode* Compound(string p, TreeNode* root1, TreeNode* root2) {
    TreeNode* compoundRoot = new TreeNode(p, root1, root2);
    return compoundRoot;
}

```

求导（有bug）

C++

```cpp
TreeNode* Derivative(TreeNode* root, const string& variable) {
    if (root == nullptr) {
        return nullptr;
    }

    // 如果节点是变量节点，且变量名匹配，导数为 1
    if (root->word.size() == 1 && root->word == variable)
        return new TreeNode("1");



    // 如果节点是操作符节点
    if (root->word == "+") {
        TreeNode* leftDerivative = Derivative(root->left, variable);
        TreeNode* rightDerivative = Derivative(root->right, variable);
        return new TreeNode("+", leftDerivative, rightDerivative);
    }
    else if (root->word == "-") {
        TreeNode* leftDerivative = Derivative(root->left, variable);
        TreeNode* rightDerivative = Derivative(root->right, variable);
        return new TreeNode("-", leftDerivative, rightDerivative);
    }
    else if (root->word == "*") {
        TreeNode* left = new TreeNode("*", Derivative(root->left, variable), root->right);
        TreeNode* right = new TreeNode("*", root->left, Derivative(root->right, variable));
        return new TreeNode("+", left, right);
    }
    else if (root->word == "/") {//答辩
        TreeNode* numerator = new TreeNode("-", new TreeNode("*", Derivative(root->left, variable), root->right),
            new TreeNode("*", root->left, Derivative(root->right, variable)));
        TreeNode* denominator = new TreeNode("^", root->right, new TreeNode("2"));
        return new TreeNode("/", numerator, denominator);
    }
    else if (root->word == "^") {
        if (root->left->word == variable) {
            TreeNode* exponent = new TreeNode("-", root->right, new TreeNode("1"));
            TreeNode* base = new TreeNode("^", root->left, root->right);
            TreeNode* product = new TreeNode("*", exponent, base);
            return product;
```

```
        }
        else {
            TreeNode* base = new TreeNode("^", root->left, root->right);
            TreeNode* lnBase = new TreeNode("ln", root->left, nullptr);
            TreeNode* exponent = new TreeNode("*", root->right, lnBase);
            TreeNode* powerRule = new TreeNode("*", base, exponent);
            return powerRule;
        }
    }
    // 其他操作符的处理可以扩展 但是不会（
    if (root->word != variable)
        return root;

    return nullptr;
}
```

合并常数 C++

```cpp
void Merge(TreeNode* root) {
    if (root == nullptr) {
        return ;
    }
    cout << isNumeric(root->word) << endl;
    if (isNumeric(root->word)) return ;
    Merge(root->left);
    Merge(root->right);

    // 如果当前节点的左右子树都是常数，则将当前节点的值替换为它们的计算结果
    if (root->left && root->right && isNumeric(root->left->word) && isNumeric(root->right->word)) {
        int leftValue = stoi(root->left->word);
        cout << leftValue << endl;
        int rightValue = stoi(root->right->word);
        cout << rightValue << endl;
        if (root->word == "+") {
            root->word = to_string(leftValue + rightValue);
        }
        else if (root->word == "-") {
            root->word = to_string(leftValue - rightValue);
        }
        else if (root->word == "*") {
            root->word = to_string(leftValue * rightValue);
        }
        else if (root->word == "/") {
            if (rightValue == 0) return;
            root->word = to_string(leftValue / rightValue);
        }
        else if (root->word == "^") {
            root->word = to_string(pow(leftValue,rightValue));
        }

        // 清空左右子树，将当前节点设为叶子节点
        if (root->left) delete root->left;
        if (root->right)delete root->right;
        root->left = nullptr;
        root->right = nullptr;
    }

    return ;
}
```

程序运行方式简要说明

定义一个变量func，当输入为1时，你可以输入表达式：

```C++
        if (func == 1) {
            string prefix;
            cout << endl;
            cout << "|------------------------------请输入前缀表达式：------
--------------------------| " << endl;
            cout << endl;
            getline(cin, prefix);
            TreeNode* root = ReadExpr(prefix);
            if (!root) goto cls;
            roots[cnt] = root;
            copyroots[cnt] = Copy(root);
            cnt++;
            cls:
            system("cls");
            Cls();
            }

```

当输入为2时，你可以对特定表达式内的所有变量赋值：

```C++
else if (func == 2) {
            int number; double val = 0; string v = "";
            cout << endl;
            cout << "|----------------------------请输入表达式编号（0开始）----------------------------| " << endl;
            cout << endl;
            cin >> number;
            copyroots[number] = Copy(roots[number]);
            cout << endl;
            cout << "|-------------------------------请输入变量名以及值-------------------------------| " << endl;
            cout << endl;
            while (1) {
                cin >> v;
                if (v == "exit") break;
                cin >> val;
                Assign(copyroots[number], v, val);
            }
            //copyroots[number] = Copy(roots[number].first);
            //Assign(copyroots[number], v, val);
            system("cls");
            Cls();
            }
```

当输入为3时，你可以对特定表达式求值，默认变量值为0：

```C++
else if (func == 3) {
            int number;
            cout << endl;
            cout << "|----------------------------请输入表达式编号（0开始）----------------------------| " << endl;
            cout << endl;
            cin >> number;
            if (copyroots.count(number) == 0) {
                cout << "结果为:  " << Value(roots[number]) << endl;
            }
            else cout << "结果为:  " << Value(copyroots[number]) << endl;
            system("pause");
            system("cls");
            Cls();
        }
```

当输入为4时，你可以选择合并表达式或者是合并常数：

```C++
else if (func == 4) {
            int funct;
            cout << endl;
            cout << "|----------------------------合并常数(1)or表达式(2)? ----------------------------| " << endl;
            cout << endl;
            cin >> funct;
            if (funct == 1) {
                int number;
                cout << endl;
                cout << "|----------------------------请输入表达式编号（0开始）----------------------------| " << endl;
                cout << endl;
                cin >> number;
                Merge(roots[number]);
                copyroots[number] = Copy(roots[number]);
                system("cls");
                Cls();
            }
            else if (funct == 2) {
                int number1, number2;
                cout << endl;
                cout << "|----------------------------请输入表达式编号（0开始）----------------------------| " << endl;
```

当输入为5时，你可以对特定表达式中的特定变量求导：

```C++
else if (func == 5) {
            int number;
            cout << endl;
            cout << "|----------------------------请输入表达式编号（0开始）----------------------------| " << endl;
            cout << endl;
            cin >> number;
            cout << endl;
            cout << "|---------------------------------请输入变量名----------------------------| " << endl;
            cout << endl;
            string v;
            cin >> v;
            TreeNode* derivativeTree = Derivative(roots[number], v);
            cout << endl;
            cout << "|---------------------------------求偏导结果为----------------------------| " << endl;
            cout << endl;
            cout << WriteExpr(derivativeTree) << endl;
            system("pause");
            system("cls");
            Cls();
        }
```

当输入6时，你可以选择可视化你选择的表达式：

```C++
else if (func == 6) {
            int number;
            cout << endl;
            cout << "|----------------------------请输入表达式编号（0开始）----------------------------| " << endl;
            cout << endl;
            cin >> number;
            cout << endl;
            cout << "|----------------------------表达式树应该是(？)这样的----------------------------| " << endl;
            cout << endl;
            printTree(copyroots[number]);
            system("pause");
            system("cls");
            Cls();

        }
```

当输入7时，退出程序：

```C++
else if(func == 7) {
            cout << endl;
            cout << "|--------------------------------谢谢使用！再见！-------------------------------|" << endl;
            return 0;
        }
```