实验二：将算术表达式转换成语法树形式

算法描述

实验一可以将算术表达式转换为后缀表达式，后缀表达式可以很容易的转换为二叉树，再用前序遍历的方式打印。打印前创建vector存放树高大小个string。

伪代码

```
BtreeNode{elem(operator,number),leftLeaf,rightLeaf}

function post2Tree()
input:postfix expression postExpr, there is a space between terms
output: BtreeNode root
    init a stack st of BtreeNode
    turn postExpr to vector vec(split by space)
    for each term t in vec
        if t is not operator
            st.push(BtreeNode(t))
        else
            lhs<-st.pop()
            rhs<-st.pop()
            st.push(BtreeNode(t,lhs,rhs))

function traverse()
input:BtreeNode root,level l
init a vector vec of tree height strings
    if(root==null)return
    vec[l].concatenate(root.elem)
    traverse(root.leftLeaf)
    for each string in vec
        string.concatenate('\t')
    traverse(root.rightLeaf)
```

C++代码

postEval.h

```cpp
#include <stack>
#include <string>
#include <map>
#include <cctype>
#include <exception>
#include <iostream>
using namespace std;
const string operStr="+-*/()#";
char opOrder[][8] = {"><<<>>", ">><<><>", ">>>>><>", ">>>>><>", "<<<<=<=","<<<<=<=", "======="};

map<char,int> op2id={{'+',0},{'-',1},{'*',2},{'/',3},{'#',4},{'(',5},{')',6}};

```

```cpp
string splitNum(string s,int &i,char pre){
    string intStr;
    for (i=0; i < s.size(); i++)
    {
        if(operStr.find(s[i])==string::npos)
            intStr.push_back(s[i]);
        else if(i==0&&s[0]=='-'&&operStr.find(s[1])==string::npos&&operStr.find(pre)!=string::npos){
            intStr.push_back('-');
        }else
            break;
    }
    if(intStr.size()>0){
        return intStr;
    }
    else return "";
}
char lt(char lhs,char rhs){
    try
    {
        return opOrder[op2id[lhs]][op2id[rhs]];
    }
    catch(const std::exception& e)
    {
        std::cerr << e.what() << '\n';
        return 0;
    }
}

string convert(string &s){
    s+="#";
    string res;
    stack<string>numStack;
    stack<char>opStack;
    int strSize=s.size();
    int index=0;
    string tmpInt;
    int subIndex;
    char op=0;
    while(index<strSize){
        subIndex=0;
        if(isspace(s[index])){
            index++;
            continue;
        }
        else if ((tmpInt=splitNum(s.substr(index),subIndex,op))!=""){
            numStack.push(tmpInt);
            op=tmpInt[0];
            index+=subIndex;
        }
        else if(!opStack.empty()){
            op=s[index];
            char cmpRes=lt(opStack.top(),op);
            if(cmpRes=='<'){
                opStack.push(op);
            }
            else if(cmpRes=='='){
                opStack.pop();
            }
            else{
                do
                {
                    string lhs=numStack.top();
                    numStack.pop();
                    string rhs=numStack.top();
                    numStack.pop();
                    res=rhs+' '+lhs+' '+opStack.top();
                    opStack.pop();
                    numStack.push(res);
                } while (!opStack.empty()&&lt(opStack.top(),op)=='>');
                if(!opStack.empty()&&opStack.top()=='('&&op==')')opStack.pop();
                if(op!=')')
                    opStack.push(op);
            }
            index++;
        }
        else if(operStr.find(s[index])!=string::npos){
            op=s[index];
            opStack.push(s[index]);
            index++;
        }
        else
            throw invalid_argument("bad input!");
    }

    return numStack.top();
}

```

main.cpp

```
#include "postEval.h"
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

void replace_all_distinct(string &str, const string &old_val, const string &new_val)
{
    for (size_t pos = 0; pos != string::npos; pos += new_val.length())
    {
        if ((pos = str.find(old_val, pos)) != string::npos)
            str.replace(pos, old_val.length(), new_val);
        else
            break;
    }
}

class ExprTree
{
private:
    struct TreeNode
    {
```

```cpp
        string oper;
        TreeNode *left = nullptr;
        TreeNode *right = nullptr;
    };

    TreeNode *root = nullptr;
    bool isOper(string term)
    {
        return term.size() == 1 && operStr.find(term[0]) != string::npos;
    }
    vector<string> expr2Vec(string expr)
    {
        auto posExpr = convert(expr);
        vector<string> ret;
        int start = 0, end = 0;
        auto e_size = posExpr.size();
        for (int i = 0; i < e_size; i++)
        {
            if (!isspace(posExpr[i]))
                end++;
            else
            {
                ret.push_back(posExpr.substr(start, end - start));
                end = start = end + 1;
            }
        }
        ret.push_back(posExpr.substr(start, e_size - start));
        return ret;
    }
    int max_level(TreeNode *node)
    {
        if (!node)
            return 0;
        return std::max(max_level(node->left), max_level(node->right)) + 1;
    }

public:
    int height;
    ExprTree(string expr)
    {
        auto vec = expr2Vec(expr);
        stack<TreeNode *> st;
        for (int i = 0; i < vec.size(); i++)
        {
            if (!isOper(vec[i]))
            {
                st.push(new TreeNode{vec[i]});
            }
            else
            {
                auto rhs = st.top();
                st.pop();
                auto lhs = st.top();
                st.pop();
                st.push(new TreeNode{vec[i], lhs, rhs});
            }
        }
        root = st.top();
        height = max_level(root);
    }
    friend ostream &operator<<(ostream &os, ExprTree exprTree);
    void traverse(TreeNode *node, vector<string> &treeVec, int level = 0)
    {
        if (!node)
            return;
        treeVec[level] += node->oper;
        traverse(node->left, treeVec, level + 1);
        for (int i = 0; i < height; i++)
            treeVec[i] += '\t';
        traverse(node->right, treeVec, level + 1);
    }
    vector<string> to_vector()
    {
        vector<string> treeVec(height);
        traverse(root, treeVec);
        return treeVec;
    }
    ~ExprTree() = default;
};
ostream &operator<<(ostream &os, ExprTree exprTree)
{
    auto vec = exprTree.to_vector();
    string::size_type pos(0);
    for (auto &line : vec)
    {
        replace_all_distinct(line, "\t\t", "\t");
        os << line << endl;
    }
    return os;
}

int main(int argc, const char **argv)
{
    string s;
    getline(cin, s);
    auto t = ExprTree(s);
    cout << t;
    return 0;
}
```

这次修改了上次没有处理负数的问题，在分离数字的过程中，如果发现负号要单独检查，负号的下一个如果是数字，就要检查上一个识别到的对象是什么，如果上一个是符号，说明负号代表正负，要继续分离数字，否则是减号，认为没找到数字。