项目说明文档

数据结构课程设计

——表达式计算

作 者 姓 名： 杨宇琨

学 号： 2252843

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

# 1 分析

## 1.1 背景分析

表达式求值是计算机科学中的一个经典问题，涉及到计算机科学、数据结构和算法等多个领域。在实际应用中，表达式求值常常涉及到构建表达式的抽象语法树（AST），并基于该树进行遍历和求值操作。本项目要求实现对表达式的解析、构建二叉树，并对二叉树进行遍历，输出相应的逆波兰式、中序表达式和波兰表达式。这一系列的操作将有助于加深对表达式求值过程的理解，以及对二叉树的操作和遍历方法的熟悉。

## 1.2 功能分析

1. 表达式解析：程序能够接收用户输入的表达式，包括括号，并进行解析和处理，确保表达式符合语法规则。

2. 构建二叉树：根据解析得到的表达式构建对应的表达式二叉树，包括操作符和操作数等节点。

3. 遍历二叉树：实现二叉树的前序遍历、中序遍历和后序遍历操作，获取遍历结果。

4. 输出逆波兰式：对构建的表达式二叉树进行后序遍历，输出对应的逆波兰式。

5. 输出中序表达式：对构建的表达式二叉树进行中序遍历，输出对应的中序表达式。

6. 输出波兰表达式：对构建的表达式二叉树进行前序遍历，输出对应的波兰表达式。

# 2 设计

## 2.1 数据结构设计

1. 表达式二叉树：采用节点结构表示表达式二叉树，左右子节点指针连接子树。

2. 栈数据结构：使用数组实现的栈，包括数据数组、容量、栈顶指针等成员。

## 2.2 类结构设计

1. TreeNode类：表示表达式二叉树的节点，包括数据、操作符、左右子节点指针。

2. MyStack类：表示栈数据结构，用于辅助表达式树的构建和遍历。

//TreeNode

class TreeNode {

public:

int data;

char op;

TreeNode\* left;

TreeNode\* right;

TreeNode(int data ,char val) : data(data), op(val),left(nullptr), right(nullptr) {}

};

//MYSTACK

class MyStack {

private:

TreeNode\*\* data;//An array in which all the elements are a pointer pointing to the treenode

int capacity;

int top;//Top index

public:

MyStack(int cap); //Initialize the the stack(mallocate a space)

~MyStack();//Destory the memory

void push(TreeNode\* node);//Push a tree node in the stack

TreeNode\* pop();//Pop out a tree node

TreeNode\* peek();//Get the top of the stack

bool isEmpty();//Check whether the stack is empty

};

## 2.3 成员与操作设计

1. TreeNode类成员：

- data：整型数据，表示操作数。

- op：字符型操作符。

- left、right：指向左右子节点的指针。

2. MyStack类成员：

- data：指向TreeNode指针的数组。

- capacity：栈的容量。

- top：栈顶指针。

3. MyStack类操作：

- push：将节点指针压入栈。

- pop：弹出栈顶节点指针。

- peek：获取栈顶节点指针。

- isEmpty：判断栈是否为空。

## 2.4 系统设计

1. 表达式树构建：通过createExpressionTree函数实现表达式二叉树的构建，逐个解析表达式字符，创建对应的节点，并利用栈辅助构建树结构。

2. 表达式遍历：通过printorder、printInorder、printPostorder函数实现对表达式树的前序、中序、后序遍历，输出逆波兰式、中序表达式和波兰表达式。

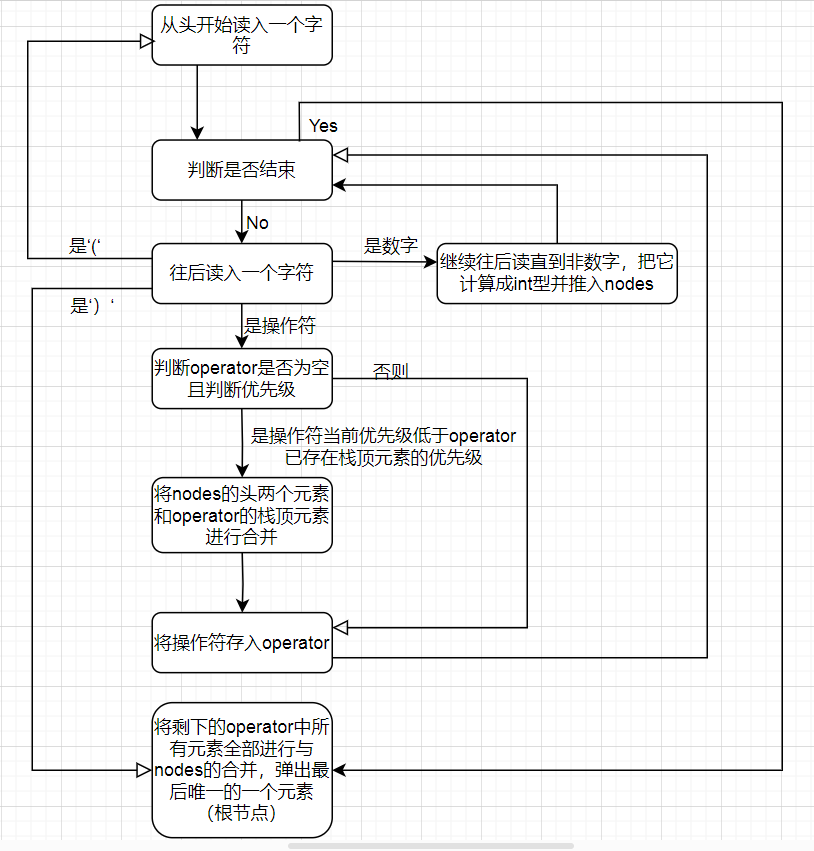
3. 表达式求值：通过evaluateExpression函数实现对表达式树的求值，支持四则运算，并做了除零判断。

4. 用户交互：在main函数中实现用户输入表达式、调用函数对表达式进行处理和输出结果，完成了与用户的交互。

# 3 实现

## 3.1创造表达树功能的实现

### 3.1.1 1创造表达树功能流程图



### 3.1.2创造表达树功能核心代码

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:createExpressionTree

Function:To create the expression tree

Input Parameters:an expression an index

Return Value:void

Documentation：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

TreeNode\* createExpressionTree(const char\* expression, int& pos)

{

MyStack nodes(100); // Max capacity of 100, adjust as needed

MyStack operators(100); // Stack to hold operators

//Read until end

while (expression[pos] != '\0') {

char token = expression[pos];//The char we are checking

if (isdigit(token)) {

int operand = 0;//Calculate its value

while (isdigit(expression[pos])) {

operand = operand \* 10 + (expression[pos] - '0');

++pos;

}//Caculate the number

TreeNode\* operandNode = new TreeNode(operand, '\0');//Create a treenode to store it

nodes.push(operandNode);//Push it into the nodes

}

else if (isOperator(token)) {

//If the operatorsstack is not empty and the operators we are checking is prior

while (!operators.isEmpty() && getPrecedence(operators.peek()->op) >= getPrecedence(token)) {

TreeNode\* operatorNode = operators.pop();//Get the operators top element

TreeNode\* rightNode = nodes.pop();//Get the top two elements in nodesstack

TreeNode\* leftNode = nodes.pop();

operatorNode->left = leftNode;//Connect these two elements(operatorsNode is the root)

operatorNode->right = rightNode;

nodes.push(operatorNode);//Push the operatorsNode into the node stack

}

//If not ,just put the operate into the stack directly without any processing

TreeNode\* operatorNode = new TreeNode(0, token);

operators.push(operatorNode);

++pos;//Index++

}

else if (token == '(') {

++pos;

TreeNode\* subexpression = createExpressionTree(expression, pos); //Create a subexpression and make it a node

++pos; // Skip ')'

nodes.push(subexpression);

}

else if (token == ')')

break;//If thers is a ')',then process

else

++pos;//Read a blank or something unimportant

}

//Process until the operatorsstack is empty

while (!operators.isEmpty()) {

TreeNode\* operatorNode = operators.pop();//Get the operators top element

TreeNode\* rightNode = nodes.pop();//Get the top two elements in nodesstack

TreeNode\* leftNode = nodes.pop();

operatorNode->left = leftNode;//Connect these two elements(operatorsNode is the root)

operatorNode->right = rightNode;

nodes.push(operatorNode);//Push the operatorsNode into the node stack

}

return nodes.pop();//There will be only a root treenode in this stack

}

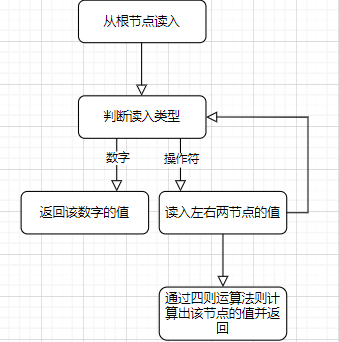
### 

### 3.1.3 创造表达树功能截屏示例



## 3.2 计算功能的实现

### 3.2.1 计算功能流程图



### 3.2.2 计算功能核心代码

int evaluateExpression(TreeNode\* root) {

if (!root) //If there is no root ,return

return 0;

if (root->op == '\0') //If read a number,return it

return root->data;

int leftValue = evaluateExpression(root->left);

int rightValue = evaluateExpression(root->right);

//If read an operator

switch (root->op) {

//Caculate

case '+':

return leftValue + rightValue;

case '-':

return leftValue - rightValue;

case '\*':

return leftValue \* rightValue;

case '/':

if (rightValue != 0) {

return leftValue / rightValue;

}

else {

std::cerr << "零不能作为除数！" << std::endl;

exit(1);

}

}

return 0; //In case of unexpected operator

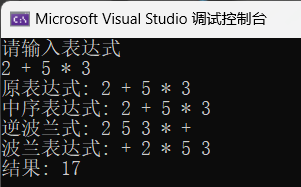
}nt;

}

}

### 

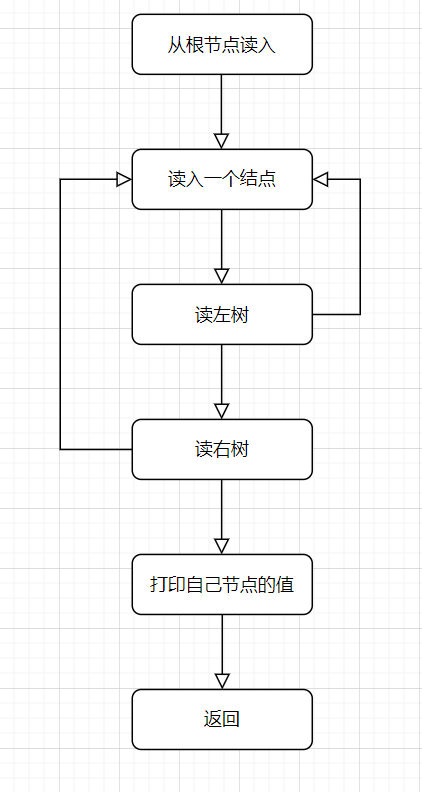
### 3.2.3 删除功能截屏示例



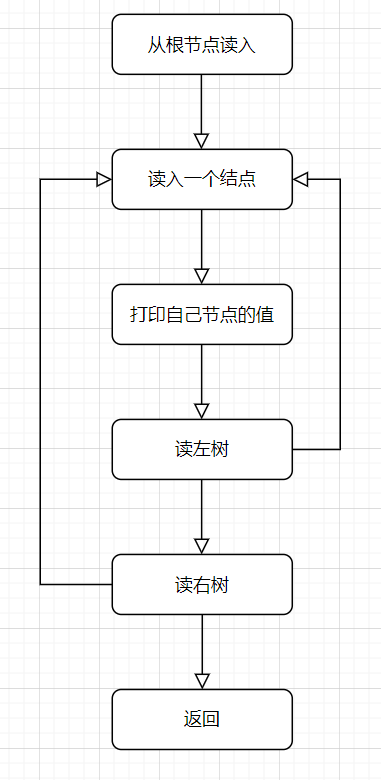
## 3.3 正逆波兰表达式显示功能的实现

### 3.3.1 正逆波兰表达式显示功能流程图

逆波兰表达式



波兰表达式



### 3.3.2 正逆波兰表达式显示功能核心代码

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:printorder

Function:To print the tree in a order

Input Parameters:the root of the tree

Return Value:void

Documentation：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void printorder(TreeNode\* root) {

if (root) {

if (root->op != '\0')

std::cout << root->op << ' ';

else

std::cout << root->data << ' ';

printorder(root->left);

printorder(root->right);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:printPostorder

Function:To print the tree in a post order

Input Parameters:the root of the tree

Return Value:void

Documentation：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void printPostorder(TreeNode\* root) {

if (root) {

printPostorder(root->left);

printPostorder(root->right);

if (root->op != '\0')

std::cout << root->op << ' ';

else

std::cout << root->data << ' ';

}

}

### 3.3.3 正逆波兰表达式显示功能截屏示例

