# Tree Comprehensive Experiment Report

Class:计科201 Student ID 1: 20401010119 Name 1: 周宸 Experiment Date:2021/12/7

Student ID 2：20401010110 Name 2：徐彤

Student ID 3：20401010126 Name 3：张雅康

**One. Experimental purpose**

1、Do linear list operations by using basic operations of tree or binary tree.

2、Handle file operations.

3、Deep the understanding of tree and binary tree, gradually develop the programming ability to solve practical problems.

**Two. Experimental environment**

Computers equipped with Visual C6.0/CFree.

The experiment lasted for 4 hours.

**Three. Experimental content**

1. design a "automatic calculator" as follows：

（1）The expression that needs to be calculated is stored in the text file in the TXT text；

（2）Each line in the text is an expression；

（3）Expressions include operands, operators such as addition, subtraction, multiplication and division, and parentheses；

For example： （34-72.3）\*54.7-82.4

（4）"Automatic calculator" calculates each expression in the text file according to the input file name, and writes every expression of the result to the original file name in the \_out.txt, you should use the method of covering and when saving the records. The format of each row is：

expression = result。

For example：the original file is: A1.txt

The file for output is： A1\_out.txt

The format of the text in A1\_out.txt is：

（34-72.3）\*54.7-82.4 = -2177.41

For all the calculated results, you'd keep 4 digits after the decimal point if it is decimal.

（5）Generate a statistical document after the calculation, its content is：

Execution time：xxxx-xx-xx hh:mm:ss

The total number of expressions is：XXX

The number of correct expressions is：XXX

The number of error expressions is：XXX

Naming rules for filenames：original file name :\_log.txt，Write files with append write method。

For example：A1.txt corresponding to the statistical file：A1\_log.txt

**Special remind**：★The calculation process requires transform the infix expression to the postfix expression and then transform the postfix expression to expression tree. Finally get the result by calculating the expressions.

(If you are getting into trouble in calculating decimal, you can only consider integer calculation.)

**Four. Important data structures**

void InitStack()//初始化栈

int IsEmpty() //判断栈是否为空

int Push(T X) //压数进栈

T Pop() //出栈

T Peek() //返回第一个数据

int GetTop() //栈顶元素

T ReturnData(int Top)//返回数组元素

int TotalExp = 0; //表达式总数

int ErrorExp = 0;//错误表达式

struct Node {

double Operand;

char Operator;

};

struct TreeNode {

double Operand;

char Operator;

struct TreeNode \*Lchild; //左孩子

struct TreeNode \*Rchild; //右孩子

};

**Five. Realization idea analysis**

**//a founction to calculate the number int an expression;**

**double integer = 0; //initialize the interger part;**

**double decimal = 0; //initialize the decimal part;**

**int k = 0; //a variable stores how many digits behind the decimal point;**

**while (!IsOperator(Expression[i])) //add i < Expression.size() when no "="**

**{**

**while (Expression[i] != '.' && !IsOperator(Expression[i]))//calculate the interger part;**

**{ //add i < Expression.size() when no "="**

**integer \*= 10;**

**integer += (Expression[i] - '0');**

**i++;**

**}**

**while (!IsOperator(Expression[i])) { // add i < Expression.size() when no "="**

**if (Expression[i] == '.')**

**i++;**

**else //calculate the decimal part;**

**{**

**decimal \*= 10;**

**decimal += (Expression[i] - '0');**

**i++;**

**k++;**

**}**

**}**

**}**

**i--;**

**while (k--) //get the decimal;**

**decimal /= 10;**

**return integer + decimal;**

**//a fonction to change infix to postfix;**

**Node node;**

**Stack<char> OperatorStack; //creat a stack with type of char to store operators;**

**Stack<Node> PostStack; //creat a stack with type of Node to store postfix expressions;**

**OperatorStack.InitStack();**

**PostStack.InitStack();**

**while (Expression[i] != '=') { //i < Expression.size() when no "="**

**node.Operator = '#';**

**if (Expression[i] == '+' || Expression[i] == '-')**

**{**

**//Process all + - \* / in the top of the Operrator satck;**

**while (OperatorStack.Peek() == '+' || OperatorStack.Peek() == '+' ||**

**OperatorStack.Peek() == '\*' || OperatorStack.Peek() == '/') {**

**node.Operator = OperatorStack.Pop();**

**PostStack.Push(node);**

**}**

**OperatorStack.Push(Expression[i]);//Push the + or - operator into the Operator stack;**

**}**

**else if (Expression[i] == '\*' || Expression[i] == '/')**

**{**

**//Process all \* / in the top of the Operrator satck;**

**while (OperatorStack.Peek() == '\*' || OperatorStack.Peek() == '/') {**

**node.Operator = OperatorStack.Pop();**

**PostStack.Push(node);**

**}**

**OperatorStack.Push(Expression[i]);//Push the \* or / operator into the Operator stack;**

**}**

**else if (Expression[i] == '(')**

**OperatorStack.Push(Expression[i]);//Push ( to Operator stack;**

**else if (Expression[i] == ')')**

**{**

**while (OperatorStack.Peek() != '(') {//Process all the operators int the stack until seeing ( ;**

**node.Operator = OperatorStack.Pop();**

**PostStack.Push(node);**

**}**

**OperatorStack.Pop();//Pop the ( symbol from the stack;**

**}**

**else**

**{**

**node.Operand = Num(Expression, i); //calculate the number from the expressions;**

**PostStack.Push(node);//Push an operand to the stack ;**

**}**

**i++;**

**}**

**while (!OperatorStack.IsEmpty()) { //Process all the remaining operators int the stack;**

**node.Operator = OperatorStack.Pop();**

**PostStack.Push(node);**

**}**

**return PostStack;**

**//using the post stack to creat a tree;**

**Stack<TreeNode\*> TreeStack;**

**TreeStack.InitStack();**

**TreeNode \*node, \*root;**

**int bottom = PostStack.GetTop(); //get the total number of the stack;**

**int top = 0;**

**while (top <= bottom)**

**{**

**if (PostStack.ReturnData(top).Operator == '#')**

**{**

**node = (TreeNode\*)malloc(sizeof(TreeNode));**

**node->Operand = PostStack.ReturnData(top).Operand;**

**node->Lchild = node->Rchild = NULL;**

**TreeStack.Push(node);**

**}**

**else**

**{**

**root = (TreeNode\*)malloc(sizeof(TreeNode));**

**root->Operator = PostStack.ReturnData(top).Operator;**

**root->Rchild = TreeStack.Pop();**

**root->Lchild = TreeStack.Pop();**

**TreeStack.Push(root);**

**}**

**top++;**

**}**

**return TreeStack;**

**//a function to calculate the operater tree;**

**if (T->Lchild == NULL && T->Rchild == NULL)**

**return T->Operand;**

**switch (T->Operator)**

**{**

**case '+':return Calculate(T->Lchild) + Calculate(T->Rchild);**

**case '-':return Calculate(T->Lchild) - Calculate(T->Rchild);**

**case '\*':return Calculate(T->Lchild) \* Calculate(T->Rchild);**

**case '/':return Calculate(T->Lchild) / Calculate(T->Rchild);**

**}**

**Six. Program debugging problem analysis**

一．How do suffix expressions generate expression trees？  
Traverse the suffix expression string array to determine whether it is an operator or an operand. If it is an operand, construct constantexpress, and then press the stack. If it is an operator, get out of the stack twice, construct a binaryexpression, and then press it on the stack;  
二．How to convert infix expression to suffix expression  
When an operand is encountered, the operand is put into the result set; When an operator is encountered, the condition for judging whether to press the stack is  
  
1. If the current stack is empty, press the stack. 1. If the current stack is empty, press the stack  
  
2. If the priority of the operator at the top of the stack is lower than that of the current operator, the conditions for pressing the stack out of the stack  
  
3. If the priority of the element at the top of the stack is higher than or equal to the priority of the current operator, get out of the stack until the priority of the element at the top of the stack is lower than the current operator, and then put the current operator on the stack

**Seven. Experimental summary**

Learned the storage of branch nodes and leaf nodes.  
Application and conversion of prefix expression, infix expression and suffix expression

**Eight. Crew Division**

|  |  |  |
| --- | --- | --- |
| **Group division** | | |
| **Member name** | **Work done** | **Completion situation** |
| **周宸** | **代码和报告** | **100%** |
| **徐彤** | **代码和报告** | **100%** |
| **张雅康** | **代码和报告** | **100%** |