

Tree Comprehensive Experiment

Class:网络182

Student ID:18401190125

Name:徐炜涛

Experiment Date:2019年11月30日

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

A computer with Visual Studio.

This experiment has 4 class hours in all.

Three. Experiment content

(Choose one of the following two:)

1、 The directory structure of the specified directory file system is given, you should write into the file dir_structure.txt by the indentation method, and calculate the storage area of the directory. (The indentation method is shown in P93)

2、 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

Important data structures:

```
typedef struct Node {           //结构体数组：用来存储读取文件串以后的数据和字符
    double num;                 //采用分离的方法，数据为数据、符号为符号
    char op;
    bool Judge;                 //是数字就为true
}Node;

typedef struct B_Tree           //构建树的时候使用，其实可以使用一个结构体
{
    char op;
    double value;
    bool Judge;
    B_Tree*R;
    B_Tree*L;
}B_Tree;
```

The main function

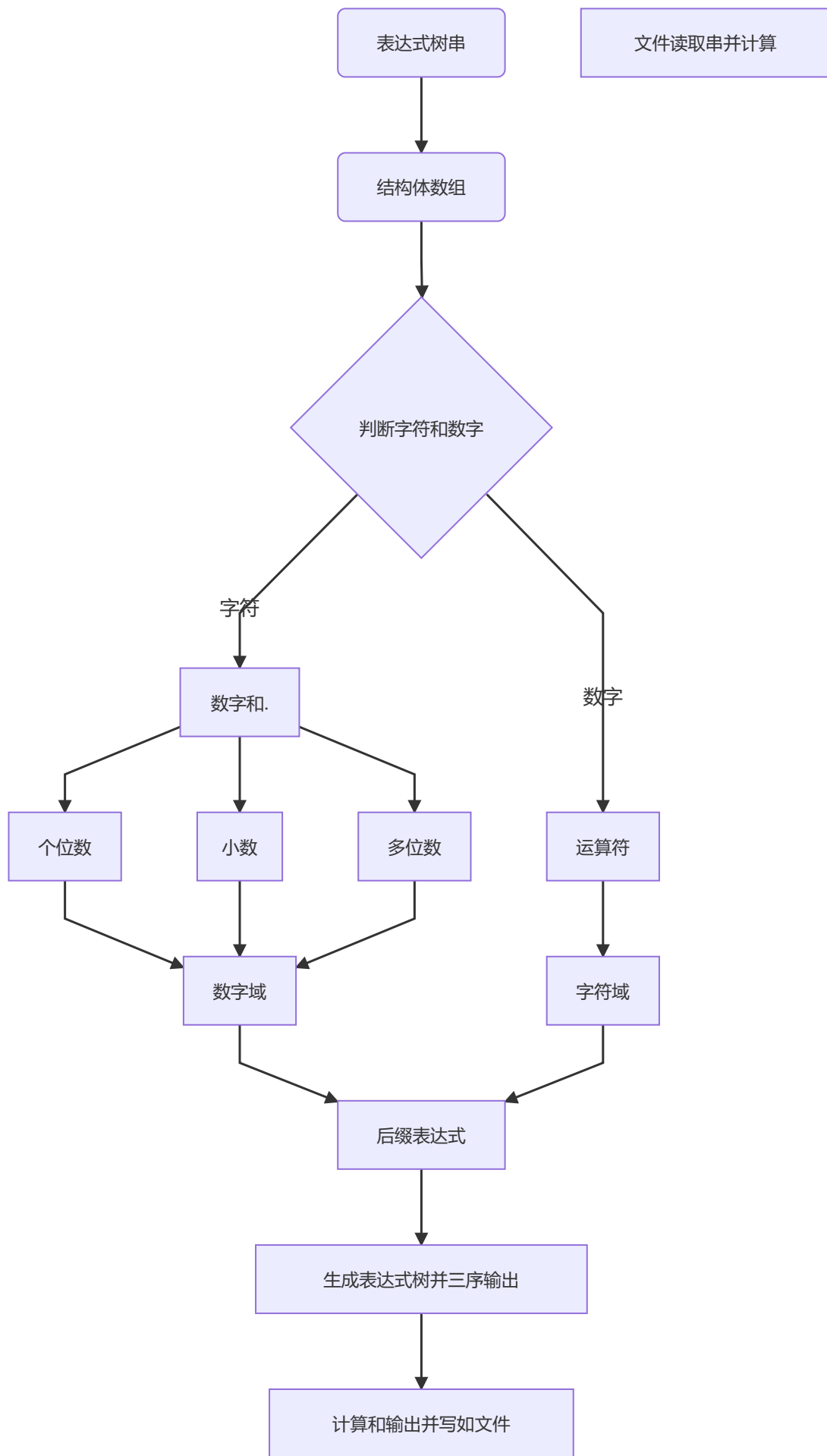
```
int main()
{
    The_Big_Function();
    return 0;
}
```

The Global function

```
extern stack<Node> s;           //为了后缀表达式的使用，使用了结构体数组其实就不用转后缀表达式
extern stack<Node> S_ans;       //为了计算最后结果的使用栈，其实可以直接使用第一个栈，因为第一次已经使用完毕了
extern queue<Node> q;           //队列，一开始用串去处理的时候使用的是 数组替代了队列，此时需要多一个全局变量k去计数，但是一样可以操作
extern queue<Node> q2;
extern stack<B_Tree*> T_Tree;    //指针树栈
extern struct B_Tree* root;     //根节点
```

Five.Implementation analysis

Program Execution Structure Diagram



The main operation:

(1):Infix to suffix

```
void Postfix_Expression(string S) {
    Node T; //定义结构体变量存放串和数字
    for (int i = 0; i < S.length();i++) { //中缀转后缀不解释了，网上一堆教程
        短就对了，本来就没那么麻烦，如果不是结构体大概也就35行
        if (S[i] == '(') {
            T.Judge = false;
            T.op = S[i];
            s.push(T);
        }
        else if (S[i] == ')') {
            while (!s.empty() && s.top().op != '(') {
                q.push(s.top());
                s.pop();
            }
            s.pop();
        }
        else if (S[i] >= '0' && S[i] <= '9') {
            i = deal_number(S,i);
        }
        else {
            T.Judge = false;
            while (!s.empty() && judge_Symbol(s.top().op) >= judge_Symbol(S[i]))
            { //优先级函数
                q.push(s.top());
                s.pop();
            }
            T.op = S[i];
            s.push(T);
        }
    }
    while (!s.empty()) {
        q.push(s.top());
        s.pop();
    }
    q2 = q;
}
```

(2):Priority judgment function

//在写这个函数的时候其实也发现了一种比较好的写法，使用map去对应可以让函数更短
//强调以下STL库很好用，多看看会有很多解决办法

```
int judge_Symbol(char Symbol)
{
    if (Symbol == '+' || Symbol == '-')
        return 1;
    else if (Symbol == '*' || Symbol == '/')
        return 2;
    else if (Symbol == '(')
        return -1;
    else
        return 0;
}
```

(3):Create tree

```
void Creat_Tree()
{
    while (!q.empty())
    {
        B_Tree * B = (B_Tree*)malloc(sizeof(B_Tree));
        if (q.front().Judge == true)
        {
            B->Judge = true;
            B->value = q.front().num;
        }
        else
        {
            B->Judge = false;
            B->op = q.front().op;
        }
        B->R = NULL;
        B->L = NULL;

        if ((B->Judge!=true)&&(B->op == '+') || (B->op == '-') || (B->op ==
        '*') || (B->op == '/'))
        {
            B->R = T_Tree.top();
            T_Tree.pop();
            B->L = T_Tree.top();
            T_Tree.pop();
        }
        T_Tree.push(B);
        q.pop();
    }
    root = T_Tree.top();
}
```

(4):Calculation answer

```
double Calculate()
{
    double num_1, num_2;
    Node T, Cur;
    while (!q2.empty())
    {
        Cur = q2.front();
        q2.pop();
        if (Cur.Judge == true)
            S_ans.push(Cur);
        else
        {
            num_2 = S_ans.top().num;
            S_ans.pop();
            num_1 = S_ans.top().num;
            S_ans.pop();
            T.Judge = true;
            if (Cur.op == '+')
                T.num = num_2 + num_1;
            else if (Cur.op == '-')
                T.num = num_1 - num_2;
```

```

        else if (Cur.op == '*')
            T.num = num_1 * num_2;
        else if (Cur.op == '/')
            T.num = num_1 / num_2;
        else if (T.op != '+' || T.op != '-' || T.op != '*' || T.op != '/')
        {
            return -999999999;
        }
        S_ans.push(T);
    }
}
return S_ans.top().num;
}

```

(5):Expression tree output

```

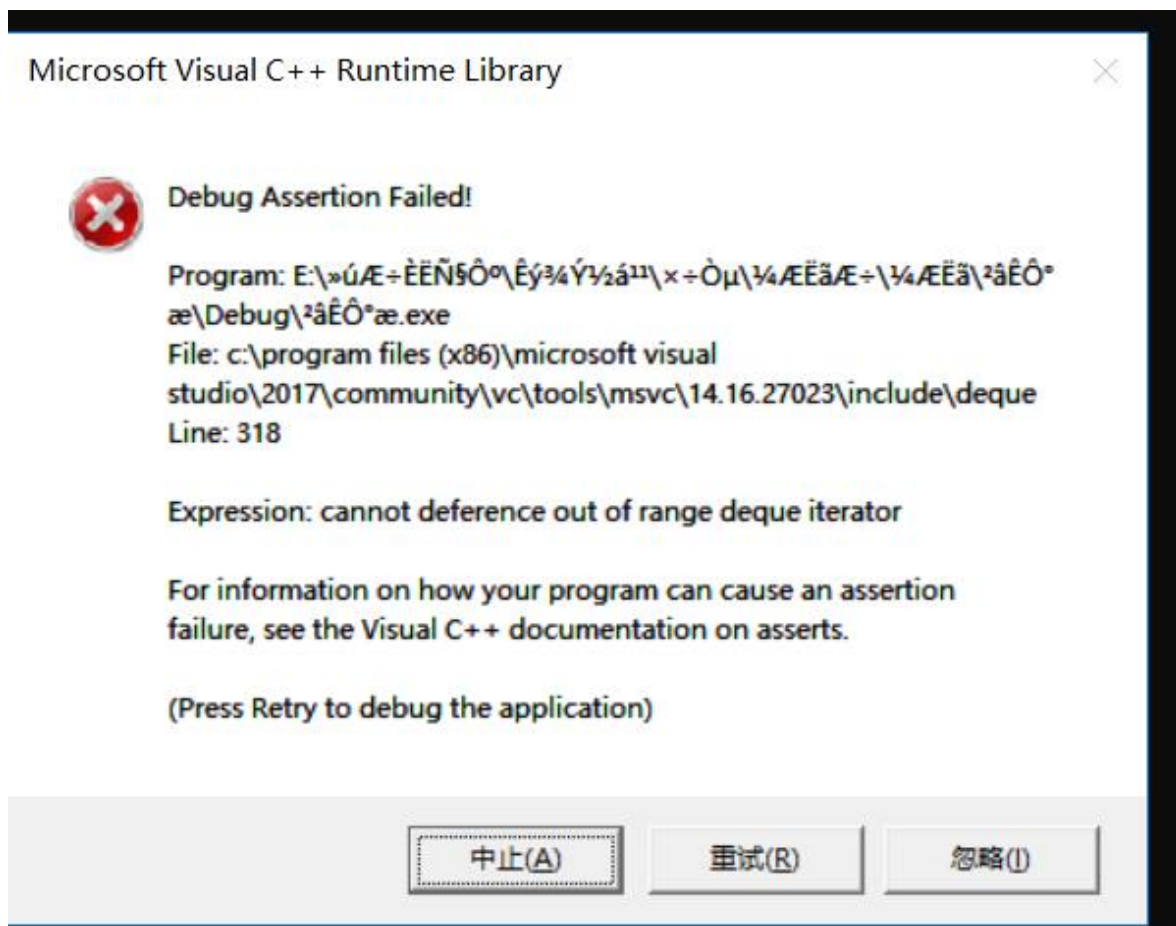
void All_Cout(B_Tree*root)
{
    cout << "先序遍历" << endl;
    Pre_Order_Cout_Tree(root);
    cout << endl;
    cout << "中序遍历" << endl;
    In_Order_Cout_Tree(root);
    cout << endl;
    cout << "后序遍历" << endl;
    Post_Order_Cout_Tree(root);
    cout << endl;
}

```

Six.Debugging problem analysis

(1)Problem:

When the expression tree was built, there was no stack pointer, which led to memory leakage. It was not remembered until later that the stack pointer needed to be built. In the next generation tree, there was a problem that the head node in the queue could not judge the symbol. Obviously, the same data as node B had been leaked in spring, but later it was found that the error of not judging Q was empty




Solvation:

Add the judgment that Q is null or directly use B as the judgment condition

(2)Problem:

In the process of decimal conversion, the symbol cannot enter the cycle. In the process of single step debugging, no error is found for a long time, and no error is found in the program, so it is stuck

 Microsoft Visual Studio 调试控制台


```
* + 1 + 1 5 3
E:\机器人学院\数据结构\作业\计算器\作业\
若要在调试停止时自动关闭控制台，请启用“工
按任意键关闭此窗口...
```

Solvation:

Careful single step debugging, from the first time you change the debugging calculation, you will find that in the middle of the place to prevent calculation errors, an auto increase causes errors, and after the end of the cycle, an auto decrease solves the problem

(3)Problem:

At first, priority function was written with map to simplify a lot, but it needs a global variable. Later, priority function was used, but the structure of map was still used

 Microsoft Visual Studio 调试控制台

```
1 1 * + + 5 3
E:\机器人学院\数据结构\作业\
```

Solvation:

Add a priority and solve it perfectly. In fact, many of these problems are caused by changing methods

Seven.Summary

At the beginning, the processing string was selected, but after the string was converted to a suffix expression, a calculation error was found during the calculation. The reason was that the processing of multiple characters was forgotten. So the code transformation idea was deleted, and the structure array was selected. After the structure array was created, the processing was more intuitive. The topic required to be converted to a suffix expression, but after the structure array was used, it was There is no need to convert it into suffix expression. In order to complete the task, I wrote suffix expression tree, but at the beginning, I did not use structure pointer stack to cause errors, and then I dealt with decimals and numbers greater than 10. In non unit number processing, decimals are troublesome. It needs to be taken into account that data like 4.0 does not make mistakes. The conclusion is, trouble one All in all, turning a direction is just another way to choose trouble. For example, at the beginning of processing strings, in order to save trouble, choose the structure, but then deal with the trouble of decimal point. Careful, debugging is very important.

Eight.Crew Division

Membername	Membername	Completionsituation
徐炜涛	Programming, Implementation, Document Writing and Summary	100%