Linear table application Experiment Report

Class: 网络181 Student ID 1:18401190107 Name 1: 王云浩 Experiment Date: 2019-09-28

One. Experimental purpose

1、Use the basic operations to implement the specific operations for the linear table;
2、Master the application of file operations;
3、Improve the understanding of the data structure of linked storage structure, and gradually cultivate the programming ability to solve practical problems.

Two. Experimental environment

sublime with c++11

Three. Experimental content

> Design an automatic calculator, which is described as follows:

> （1）The expressions that need to be calculated are stored in a TXT file;

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

> （3）Expressions include operands, addition, subtraction, multiplication, division, and parentheses; for example:（34-72.3）*54.7-82.4

> （4）The “automatic calculator” calculates each expression in the text file according to the input file name, and stores each result to the file named [original filename]_new.txt. You’d better overwrite the file when saving the records. Each row in the file should be written in the following format: For example: the original file is A1.txt The calculated output file is: A1_out.txt The result in A1_out.txt text is in the form of：（34-72.3）*54.7-82.4 = -2177.41 If the calculation result is decimal, it should hold to 4 decimal places.

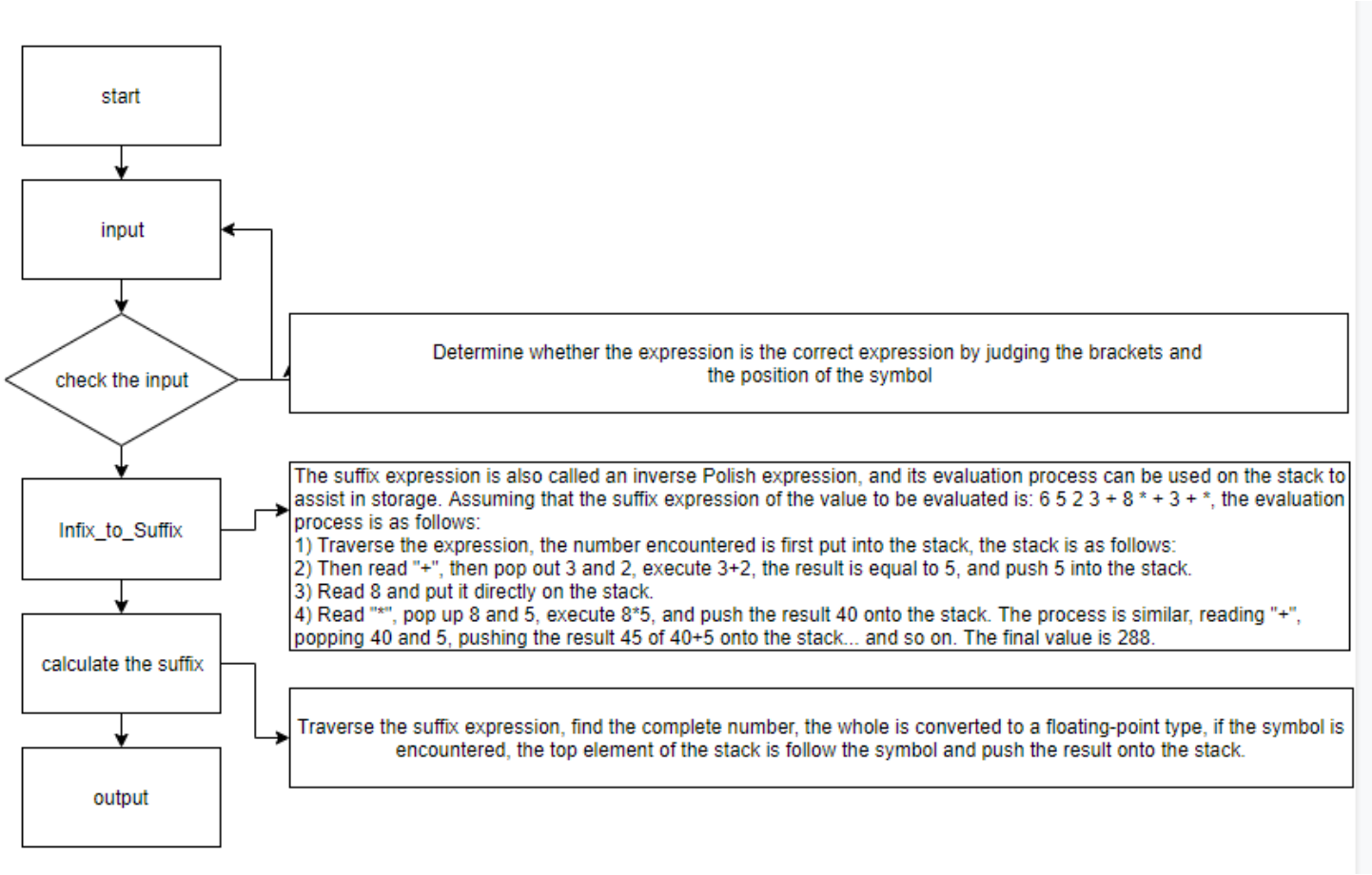
> （5）When the calculation is completed, a statistical file should be generated. Its content is as follows： Execution time: xxxx-xx-xx hh:mm:ss The total number of expressions is: XXX The number of correct expression is: XXX The number of incorrect expressions is：XXX The rule of naming the

> statistical file: [original filename] _log.txt, and the file is written in additional write mode. For example: If the original filename is A1.txt file, then the statistical document should be: A1_log.txt

Four. Important data structures

```
double String_to_double(string s) //Convert a string to a float
string Infix_to_Suffix(string s)//Infix expression to suffix expression
bool judge_infix(string str)//Determine whether the infix expression is correct
struct stack{
int a[maxn], l = 0;//Use array to store data, l is the top of the stack
void push(int x){
a[++l] = x;//Push data into the stack
}
int top(){
return a[l];//return the top of stack
}
void pop(){
l--;//Move the top of the stack pointer down to push the data out of the stack
}
int empty(){
return l \>= 0 ? 1 : 0;//Determine whether the stack is empty by whether the top pointer of the
}
};
```

Five. Implementation analysis



Step1 ： input from the file :

```
Read data from A.txt, store it in the format of each line string in A.txt, one read to EOF
```

Step2:check the input

```
Determine whether the expression is the correct expression by judging the brackets and the posit
```

```
bool judge_infix(string str) {
    int temp = 0;
    if (str[0] == '/' || str[0] == '*')
        return false;
    if (str[str.length() - 1] < '0' && str[str.length() - 1] > '9')
        return false;
    for (int i = 0; i < str.length(); i++) {
        if (str[i] == '(') {
            if (i == 0 && (str[i + 1] == '*' || str[i + 1] == '/'))
                return false;
            else if (str[i - 1] >= '0' && str[i - 1] <= '9')
                return false;
            temp++;
        } else if (str[i] == ')') {
            if (i == 0)
                return false;
            else if (str[i - 1] == '+' || str[i - 1] == '*' || str[i - 1] == '-' || str[i - 1] =
                return false;
            else if (str[i + 1] >= '0' && str[i + 1] <= '9')
                return false;
            temp--;
        }
    }
    if (temp == 0)
        return true;
    return false;
}
```

Step 3 : infix to shufix:

1. any infix expression consists of operands, operators, brackets (large, medium, small), these three parts.
2. Start scanning from the left side of the infix expression. If an operand is encountered, it will be output directly (not pushed into the stack).
3. If you encounter the left parenthesis, push it on the stack.
4. If the closing parenthesis is encountered, the expression of the infix in the parentheses has been scanned. At this time, the operator at the top of the stack needs to be popped up and output until the left parenthesis is encountered [the left parenthesis pops up but does not output].
5. If you encounter an operator:

> a, if the priority of the operator is greater than the priority of the top-of-stack operator, push it
>
> b. If the priority of the operator is less than the priority of the top-of-stack operator, the top-of-stack operator is popped and output, and then compared with the new top-of-stack operator. If it is greater, it is pushed onto the stack. If it is less than, Continue to pop the top-of-stack operator and output... (always recursive until the operator is greater than the top-of-stack cloud operator).

In the last step, if you scan to the end of the infix expression [that is, the end of the scan], if there are still existing operators in the stack, they will pop up and output.

```
string Infix_to_Suffix(string s) {
    string ans;
    stack<char>st;
    for (int i = 0; i < s.size(); i++) {
        if (s[i] == ' ') continue;
        else if (isdigit(s[i])) {
            string str;
            while (isdigit(s[i]) || s[i] == '.') {
                str = str + s[i];
                i++;
            }
            ans = ans + str + ' ';
            i--;
        } else {
            if (s[i] == '(') st.push(s[i]);
            else if (s[i] == ')') {
                while (!st.empty()) {
                    char o = st.top();
                    st.pop();
                    if (o == '(') break;
                    ans = ans + o + ' ';
                }
            } else {
                while (!st.empty()) {
                    char o = st.top();
                    st.pop();
                    if (o == '(') {
                        break;
                    }
                    if (hash1[s[i]] < hash1[o]) {
                        ans = ans + o + ' ';
                    }
                }
                st.push(s[i]);
            }
        }
    }
    while (!st.empty()) {
        char o = st.top();
        st.pop();
        ans = ans + o + ' ';
    }
    return ans;
}
```

Step 4 : calculate:

Traverse the suffix expression, find the complete number, the whole is converted to a floating-point type, if the symbol is encountered, the top element of the stack is follow the symbol and push the result onto the stack.

```
        for (int i = 0; i < s.size(); i++) {
            cout<<i<<" "<<s[i]<<endl;
            if (s[i] == ' ') continue;
            if (isdigit(s[i])) {
                string stt;
                while (isdigit(s[i]) || s[i] == '.') {
                    // cout<<stt<<endl;
                    stt = stt + s[i];
                    i++;
                }
                st.push(String_to_double(stt));
            } else {
                double temp;
                double Secondnum = st.top();
                st.pop();
                double Firstnum = st.top();
                st.pop();
                // cout << Firstnum << " " << Secondnum << endl;
                if (s[i] == '+') temp=Firstnum + Secondnum;
                if (s[i] == '-') temp=Firstnum - Secondnum;
                if (s[i] == '*') temp=Firstnum * Secondnum;
                if (s[i] == '/') temp=Firstnum / Secondnum;

                st.push(temp);

            }
            ans=st.top();

        }
        cout << str << " = " << ans << endl << endl;
```

Step 5: output

```
When the calculation is completed, a statistical file should be generated. Its content is as fol
 Execution time: xxxx-xx-xx hh:mm:ss
 The total number of expressions is: XXX
 The number of correct expression is: XXX
 The number of incorrect expressions is: XXX
 The rule of naming the statistical file: [original filename] _log.txt
 and the file is written in additional write mode.
```

```
    fout << str << " = " << ans << endl << endl;

    logout << "Execution time:" << totaltime << "s" << endl;
    // fout << "" << AllEle << endl;
    logout << "The total number of expressions is:" << AllEle << endl;
    logout << "The number of correct expression is: " << AllEle - WrongEle << endl;
    logout << "The number of incorrect expressions is: " << WrongEle << endl;
    fin.close();
    fout.close();
    logout.close();
```

Six. Debugging problem analysis

> When calculate （34-72.3）*54.7-82.4 = -2177.41,when i8 program break, and now the answer also not right , and then I calculate it by hand ,find that I have not push the every step answer into the stack so when i8 it have not enough element to calculate.

Seven. Summary

> In this experiment we learned the establishment of the tree and the use of the stack, and have a deeper understanding of the data structure.

Eight. **Crew Division**

Group division		
Member name	**Work done**	**Completion situation**
王云浩	design code debug	well