中山大学计算机学院本科生实验报告

（2025学年第1学期）

课程名称：数据结构与算法实验　　任课教师：张子臻

年级	2024级	专业（方向）	计算机科学与技术（人工智能）
学号	242325157	姓名	梁玮麟
电话	18620062182	Email	3371676041@qq.com
开始日期	2025.9.17	结束日期	2025.9.18

第一题

1、实验题目

z2-后缀表达式计算

题目描述

- 来自USA的Mr.Allison是某班的英语老师，学期开始之前他要求自己班的同学阅读一本英文小说，并写一份50000字的读书报告。该班的同学很苦恼，并想方设法让Mr.Allison放弃读书笔记这个念头，于是该班的大牛PYY想到一个借口：看那么多份读书笔记会花费很多时间的！把这个理由告诉Mr.Allison之后，Mr.Allison也觉得挺有道理，但一共要阅读多少文字呢？于是PYY就给出一条后缀表达式，并告诉Mr.Allison说，这条表达式的结果就是您要阅读的文字。Mr.Allison的数学不咋地，于是就找你来帮他计算这条后缀表达式的值。

输入描述

- 一个长度不超过100的字符串，代表一条后缀表达式。表达式中只含有+、-、、/ 四种运算符和26个小写英文字母，不含其它字符。每一个英文字母代表一个正整数： a = 1,b = 2,c = 3...y = 25,z = 26。

输出描述

- 每一个输入样例，单独一行输出结果：后缀表达式的值，一个正实数 S，保留两位小数。

样例输入

```
ab+c*
int**py++
```

样例输出

```
256
```

2、实验目的

- 使用栈实现后缀表达式的计算。

3、算法设计

设计思路如下：

1. 初始化一个char类型的栈，用于存储字符
2. 用一个string存储输入，然后遍历这个string:
    1. 如果是数字，直接压栈；
    2. 如果是符号，则取出栈顶的两个元素，并从栈顶移除。把两个数用这个符号计算后，重新压栈。
3. 最后计算完毕后，栈内只有一个数，就是计算结果。

细节注意:

- stack应该存储double或者float类型的数据，因为除法后可能有小数点。
- 因为输入无空格，所以可以直接使用cin>>string。但是如果有空格，则需要使用getline(cin,string)。
- 遍历字符串，将数字压栈时，注意此时是char类型，需要-96（'a'=97，a=1）转化成对应的数字后计算。
- 注意操作数顺序！除法时栈顶的第二个元素当作被除数，减法时栈顶的第二个元素当作被减数。

具体实现：

```
#include<iostream>
#include<stack>
#include<string>
#include<iomanip>
using namespace std;
int main(){
    string str;
    cin>>str;//初始化一个string类型的变量，存储用户输入
    stack<double> s;
    //初始化一个存储double的栈
    //之所以是double，是为了进行除法运算的时候不出错
    for (int i=0;i<str.length();i++){//表达式有数字和符号的区分
    //如果是数字，直接压栈；如果是符号，取出栈顶的两个元素计算
        if(str[i]>=97&&str[i]<=122){
            s.push(str[i]-96);
            //数字直接压栈。但是因为这里的数字是char类型，而且是小写字母，需要-96，转化
成大于等于1的整数再压栈。
        }
        else if(str[i]=='+'){//处理加法
            double a = s.top();
            s.pop();//取出第一个数，并从栈中移除
```

```
            double b = s.top();
            s.pop();//取出第二个数，并从栈中移除
            s.push(a+b);//将两个数相加后重新压栈
        }
        else if(str[i]=='*'){
            double a = s.top();
            s.pop();//同上
            double b = s.top();
            s.pop();//同上
            s.push(a*b);//相乘后重新压栈
        }
        else if(str[i]=='-'){
            double a = s.top();
            s.pop();
            double b = s.top();
            s.pop();
            s.push(b-a);//注意操作数顺序
        }
        else if(str[i]=='/'){
            double a = s.top();
            s.pop();
            double b = s.top();
            s.pop();
            s.push(b/a);//注意操作数顺序
        }
    }
    cout<<fixed<<setprecision(2)<<s.top()<<endl;
    //栈内只有一个元素，直接将它输出
    return 0;
}
```

4、程序运行与测试

运行结果：

(仅挑选个别复杂测试样例作为展示)

测试样例一

- 标准输入：

```
abcdefghijklmnopqrstuvwxyz++++++++++----------*****
```

- 实际输出：

```
abc-d*e+f/+
```

- 期望输出：

```
abc-d*e+f/+
```

测试样例二

- 标准输入：

```
zyxwvutsrqponmlkjihgfedcba/////////////////////////
```

- 实际输出：

```
6.45
```

- 期望输出：

```
6.45
```

5、实验总结与心得

- 这一题并不复杂，理清楚思路之后实现起来也不难。就是需要注意细节不要漏掉，否则调试起来十分麻烦。特别是操作数的顺序不要弄反，我第一次写的时候就弄反了，明明程序能运行，但是错的莫名奇妙的感觉，debug了很久才发现这个错误。

第二题

1、实验题目

z2-中缀表达式转后缀表达

题目描述

- 将中缀表达式（infix expression）转换为后缀表达式（postfix expression）。假设中缀表达式中的操作数均以单个英文字母表示，且其中只包含左括号'('，右括号')'和双目算术操作符+，-，*，/。

输入描述

- 表示中缀表达式的一个字符串（其中只包含操作数和操作符和左右括号，不包含任何其他字符），长度不超过100个字符。

输出描述

- 对应后缀表达式字符串（其中只包含操作数和操作符，不包含任何其他字符）

样例输入

```
A+B*C-D-E/F
a+(b-c)*d+e/f
```

样例输出

```
ABC*+D-EF/-
abc-d*+ef/+
```

2、实验目的

- 使用栈实现将中缀表达式转化成后缀表达式的功能。

3、算法设计

设计思路如下：

1. 初始化一个char类型的栈，用于存储字符
2. 初始化一个string类型的result，用来存储输出
3. 用一个string存储输入，然后遍历这个string：
    1. 如果是字母，直接输出；
    2. 如果是符号：
        - 如果是+或者-：
            - while循环取出栈内元素并放入result，直到可以压栈（栈顶是（或者空栈）
            - 压栈
        - 如果是*或者/：
            - while循环取出栈内元素并放入result，直到可以压栈（栈顶不是*、/或者空栈）
            - 压栈
        - 如果是（：
            - 直接压栈
        - 如果是）：
            - while循环取出栈内元素并放入result，直到可以压栈（取出（之后即可，但是这里的左括号不会进入输出）
4. 将栈内仅存的符号按照FILO的顺序输出
5. 输出

细节注意：

- 循环的时候，记得用str[i]判断数字与符号，不要直接用i。

- 因为str[i]是char类型，不能用string类型比对，应该用char类型。

具体实现：

```
//z2-中缀表达式转后缀表达
#include<iostream>
#include<string>
#include<stack>
using namespace std;
int main(){
    string str;
    stack<char> s;
    cin>>str;
    string result;
    for (int i=0; i<str.length();i++){
        if (str[i]>='A'&&str[i]<='Z' ||str[i]>='a'&&str[i]<='z' ){//直接输出
            result +=str[i];
        }
        else if(str[i]=='+'||str[i]=='-'){//+ -
            while((!s.empty())&&s.top()!='('){//非空且最上面不是左括号，这个时候需要把里面的符号压出来
                result+=s.top();
                s.pop();
            }
            s.push(str[i]);
        }
        else if(str[i]=='*'||str[i]=='/'){//*  /
            while((!s.empty())&&(s.top()=='*'||s.top()=='/')){//非空且最上面是*或者/，这个时候需要把里面的符号压出来
                result+=s.top();
                s.pop();
            }
            s.push(str[i]);
        }
        else if (str[i]=='('){
            s.push(str[i]);
        }
        else if (str[i]==')'){
            while(true){
                if (s.top()=='('){
                    s.pop();
                    break;
                }
                else{
                    result+=s.top();
                    s.pop();
                }
            }
        }
    }
    while(!s.empty()){
        result+=s.top();
```

```
            s.pop();
        }
        cout<<result<<endl;
        return 0;
    }
```

4、程序运行与测试

运行结果：

(仅挑选个别复杂测试样例作为展示)

测试样例一

- 标准输入：

```
a+((b-c)*d+e)/f
```

- 实际输出：

```
abc-d*e+f/+
```

- 期望输出：

```
abc-d*e+f/+
```

测试样例二

- 标准输入：

```
a+(b-c)*d+e/f
```

- 实际输出：

```
abc-d*+ef/+
```

- 期望输出：

```
abc-d*+ef/+
```

5、实验总结与心得

- 这一题相对上一题来说难一点，但是因为在课上已经讲过思路，把转化过程内化之后其实也还好了，就只是中间的取出栈内元素的循环起止条件稍显复杂。
- 这一题应该可以对循环过程进行优化，提取出共有的出栈的过程，减少代码的编写工作。时间复杂度大概是O(n)。

第三题

1、实验题目

z2-Bracket Matching

题目描述

- 检查输入字符串中的括号是否匹配。括号包括：{, }, (,), [,].

输入描述

- 一个长度不超过100的字符串。

输出描述

- “Yes" or "No"，代表是否匹配。

样例输入

```
a
2-[(1+2)*2]
(a+b])
```

样例输出

```
Yes
Yes
No
```

2、实验目的

- 使用栈实现表达式中前后括号匹配功能。

3、算法设计

设计思路如下：

1. 初始化一个char类型的栈，用于存储字符
2. 用一个string存储输入，然后遍历这个string：
    1. 如果是左括号，直接压栈；
    2. 如果是右括号：
        - 首先判断此时是否空栈。如果是空栈，直接输出"No"，并且return 0.
        - 如果是）：
            - 如果栈顶是（，把（压出栈；
            - 如果栈顶不是（，输出"No"，并且return 0;
        - 如果是】：
            - 如果栈顶是【，把【压出栈；
            - 如果栈顶不是【，输出"No"，并且return 0;
        - 如果是}：
            - 如果栈顶是{，把{压出栈；
            - 如果栈顶不是{，输出"No"，并且return 0;
3. 如果此时栈内还有符号，说明匹配不成功，输出"No"，并且return 0；
4. 否则说明前面所有符号匹配成功，输出"Yes"，return 0.

细节注意：

- 道理大家都懂，只是有可能忘记右括号更多或者左括号更多这两种情况。
- 每次碰到右括号，都要判断是否空栈。
- 所有右括号都遍历完，都要看是否还有左括号未匹配。

具体实现：

```
//z2-Bracket Matching
#include <iostream>
#include <stack>
using namespace std;
int main() {
    string line;
    cin >> line;
    stack<char> s;
    for (int i = 0; i < line.length(); i++) {
        if (line[i] == '(' || line[i] == '{' || line[i] == '[') {//左括号直接入栈
            s.push(line[i]);
        } else if (line[i] == ')' || line[i] == '}' || line[i] == ']') {
            //右括号需要分三种情况

            //记得判断是否空栈
            if(s.empty()){
                cout<<"No"<<endl;
                return 0;
            }
            else if (line[i] == ')') {
```

```
                if ('(' == s.top()) {
                    s.pop();
                } else {
                    cout << "No" << endl;
                    return 0;
                }
            }
            else if (line[i] == ']') {
                if ('[' == s.top()) {
                    s.pop();
                } else {
                    cout << "No" << endl;
                    return 0;
                }
            }
            else{
                if ('{' == s.top()) {
                    s.pop();
                } else {
                    cout << "No" << endl;
                    return 0;
                }
            }
        }
    }
    //记得判断是否空栈
    if (s.empty()) cout << "Yes" << endl;
    else cout<<"No"<<endl;
    return 0;
}
```

4、程序运行与测试

运行结果：

(仅挑选个别复杂测试样例作为展示)

测试样例一

- 标准输入：

```
2-[(1+2)*2]
```

- 实际输出：

```
Yes
```

- 期望输出：

```
Yes
```

测试样例二

- 标准输入：

```
{
```

- 实际输出：

```
No
```

- 期望输出：

```
No
```

5、实验总结与心得

- 这一题相通过栈这种数据结构很容易实现。注意好判断空栈情况就可以了。

总结

- 虽然题目都不太难，但是重点还是应当放在为什么这种情况用栈来写特别方便。栈的最大特点就是FILO，在往后碰到题目的时候要根据算法实现方式选择合适的容器。