中山大学计算机学院本科生实验报告

（2021 学年第 1 学期）

课程名称：Data structures and algorithms　　任课教师：张子臻

年级	20 级	专业（方向）	软件工程
学号	20337270	姓名	钟海财
电话	13397996670	Email	2940599563@qq.com
开始日期	2021/9/9	完成日期	2021/9/11

1. 实验题目

1）Bracket Matching　2）后缀表达式计算　3）中缀表达式转后缀表达式

2.实验目的

通过使用栈的数据结构，分别实现：1.检查输入字符串中的括号是否匹配 2.后缀表达式的计算；3.将中缀表达式转化为相应的后缀表达式

3.程序设计

题目 1）Bracket Matching

设计思路：

1.左括号入栈，右括号与栈顶相匹配的出栈，不匹配或栈为空匹配失败。

2.最后看栈里有没有元素，有同样匹配失败。

题目 1）代码

```
//Bracket Matching
#include<iostream>
using namespace std ;

class Stack{
private:
    char s[100] ;
    int num;
    bool match ; //match 为括号匹配的结果
public:
    Stack(){
        num = 0 ;
        match = true ;//默认构造函数对 match 进行初始化为 true
    }

    void pop(const char t){
        if(t=='(' || t=='[' || t=='{'){
            s[num] = t;
            num++;
        }//t 是括号就入栈
        else if(t==')' || t==']' || t=='}'){
            if(num==0) match = false ;
            //如果栈为空则不匹配，
            if(num>0){ //栈不为空时进行匹配判断
                if( t==')' && s[num-1]=='(' ){
                --num; }
                else if( t==']' && s[num-1]=='[' ){
                --num; }
                 else if( t=='}' && s[num-1]=='{' ){
                --num; }//如果 t 与栈顶元素匹配则出栈
                else  match = false ;
                //如果不匹配则将匹配结果=false
             }
        }
    }
    void judge(const string s){ //将字符串 s 的字符依次进行 pop 操作
        for(int i = 0 ; i<s.size() ; ++i ){
            this->pop( s.at(i) );
            if( this->match == false) break;//当 match==false 时直接停止
        }
    }
    bool getmatch()const{ //返回匹配结果：
        if(num==0) return this->match;//栈中元素为空时返回 match
```

```
        else return false ;//栈中元素不为空时返回 false
    }
     void Match()const{
     if( this->getmatch() ) cout<<"Yes" ;
        else cout<<"No" ;
    }
};

int main()
 {
    Stack strs ;
    string s ;
    cin>>s ;
    strs.judge(s)  ;
    strs.Match() ;
    return 0;
 }
```

题目 2）后缀表达式计算

设计思路：

1.遇到操作数：入栈

2.遇到操作符：将栈顶的两个操作数弹出，进行相应运算，并将结果入栈

3.最后栈里有且只有一个元素，该元素为计算结果

题目 2）代码

```
//后缀表达式的计算
#include<iostream>
#include<iomanip>
using namespace std ;

class Stack{
private:

    double*data; //储存数值
    int  num;
    int Capacity ;
```

```
public:

Stack(){
    Capacity = 100 ;
    num = 0 ;
    this->data = new double[Capacity];
    }
~Stack(){
    delete [] this->data ;
    }

bool isop(char t){ //判断是否为操作符
        if( t=='+' || t=='-' || t=='*' || t=='/') return true;
        else return false;
    }

bool isnum(char t){ //判断是否为数字
        double k =t ;
        if( k>=97 && k<=122 ){
            //data[num]=k-96;++num;
            return true ;
        }
        else return false ;
    }

double set(double a ,double b,char t){//进行相应得运算
        if(t=='+') return a+b ;
        if(t=='-') return a-b ;
        if(t=='*') return a*b ;
        if(t=='/') return a/b ;
    }

void pop(char t){
        if( isnum(t) ){ //是字母就将对应得数字入栈
        double k = t ;
        data[num]=k-96;
        ++num;
        }
        if( isop(t) ){ //是操作符就将上面两个数字出栈并运算，并将运算结果入栈
            double h=set(data[num-2],data[num-1],t);
            --num;
            data[num-1]=h;
        }
    }

void calculate(const string s){ //将后缀表达式进行计算
```

```
    for(int i = 0 ; i<s.size() ; ++i ){
            this->pop( s.at(i) ) ;
        }
    }
 double get_result()const{ //得到结果
       if( num == 1 )   return data[0];
    }
};

int main(){
    Stack exp ;
    string s ;
    cin>>s ;
    exp.calculate(s);
    cout<<fixed<<setprecision(2)<<exp.get_result() ;//保留 2 位小数
    return 0 ;
}
```

题目 3）中缀表达式转后缀表达式

设计思路：

1. 操作数直接输出

2.操作符入栈，优先级高的可以压住优先级低的，优先级低的入栈会将优先级高的和同级的依次弹出再入栈

3.左括号 "(" 入栈直至 右括号")" 读入才出栈（但不输出），

并在右括号读入时左括号之上的操作符依次出栈

4 如果我们读到了输入的末尾，则将栈中所有元素依次弹出。.

题目 3）代码

```
#include<iostream>
using namespace std ;

class Stack{
private:
```

```
    char*data; //储存操作符
    int  num;
    int Capacity ;
public:
    Stack(){
     Capacity = 100 ;
    num = 0 ;
    this->data = new char[Capacity];
    }
    ~Stack(){
    delete [] this->data ;
    }

    int isop(const char t){
    //判断是否为操作符且操作符的优先级 3 */  > 2 +- > 1 (  > 0）
        if( t=='*' || t=='/') return 3 ;
        else if( t=='+' || t=='-' ) return 2;
        else if( t=='(') return 1 ;
        else if( t==')') return 0 ;
        else return -1;
    }

    void pop(char t){
        if( isop(t) != -1 ){ //不是数字时
            if(num==0){ data[num] = t ; ++num; } //栈为空直接入栈
            else{
                if(isop(t)>isop(data[num-1])){ //优先级高的可以压住低的
                    data[num] = t ; ++num;
                }
                else if(isop(t)==1){ //左括号 '(' 直接入栈
                    data[num] = t ; ++num;
                }
                else if(isop(t)==0){ //右括号读入时
                    while(isop(data[num-1])!=1 && num>0 ){
                        cout<<data[num-1];
                        --num;}
                         --num;//不要输出'('
                }
                else{ //优先级低的读入时先将高的依次出栈，再入栈
                    while(isop(t) <= isop(data[num-1]) && num>0){
                         cout<<data[num-1];
                        --num; }
                    data[num] = t ; ++num;
                }
            }
        }
```

```
        else cout<<t; //如果是数字直接输出
    }
    void change(string ss){
        for(int i = 0 ; i< ss.size() ; ++i ){
         this->pop( ss.at(i) );
        }
        for(int i = 0 ; i < num ; ++i ){ //将栈中剩余操作符输出
            cout<<data[num-1-i];
        }
    }
};

int main(){
    Stack exp ;
    string s="";
    cin>>s ;
    exp.change(s);
    return 0 ;
}
```

4.程序运行与测试

题目 1 测试：

序号	1	2	3	4	5
测试输入	123]	abc(	12(a+b)	[a+b-a)]	1{abc+d[a]}()
测试输出	No	No	Yes	No	Yes
是否通过	是	是	是	是	是

题目 2、3 测试：

序号	测试输入	对应的中缀表达式	测试输出	是否通过
1	abc*+d-	a+b*c-d	3.00	是
2	ab/cd/-g-	(a/b-c/d)-g	-7.25	是

3	ag+k-d*zk/+	(a+g-k)*d+z/k	-9.64	是
4	jkbc/l-*+er*-	j+k*(b/c-l)-e*r	-204.67	是
5	dbc-b*+ac/-	d+(b-c)*b-a/c	1.67	是

本次实验测试过程中题目 2 和题目 3 的测试是同时进行的，随机写出一个中缀表达式，先用题目 3 中的代码得到对应的后缀表达式，再用题目 2 中的代码计算出该后缀表达式的结果，再计算出原中缀表达式的结果，如果两者相等，便说明两题中的代码都正确。或者也可以分别进行测试。

5.实验总结与心得

本次实验的三个题目：括号匹配，后缀表达式的计算，中缀表达式转后缀表达式都是通过对数据分析，选择出最适合处理该类问题的数据结构是栈这种线性数据结构：

如题目 1 中括号匹配，左右对应括号的匹配涉及到左括号的入栈及遇到相应的右括号时的出栈。

题目 2 中的后缀表达式计算，根据对后缀表达式相应的二叉树的后序遍历可将问题转化为：1）遇到操作数：入栈；遇到操作符：2）将栈顶的两个元素弹出，进行运算，将结果入栈；3）最后栈里有且只有一个元素，为计算结果。

题目 3 中的中缀表达式转后缀表达式，也是通过二叉树的中序遍历及后序遍历的分析，将问题转化为对操作符的出入栈处理，从而实现转化。

所以在解决类似问题时，我们可以首先分析要处理的数据的特点，再根据其选择合适的数据结构作为解决方法，可以更为便捷的解决问题。

附录、提交文件清单