南開大學

《数据结构》

实验报告

实验（ 二 ）

年　　级: 2023 级

专　　业: 计算机科学与技术

姓　　名: 袁田

学　　号: 2314022

一．实验内容

1.实现以下内容：实现一个栈，包括出栈、入栈、获取栈顶元素、判断是否为空和统计栈中元素个数 5 个基本方法。
2.使用内容 1 中的栈进行表达式计算，表达式是一个可能包括“+”，“-”，“*”，“/”，“(”，“)”，数字以及未知数的字符串（未知数定义为 A~Z，a~z）。
2.1 判断表达式是否合法。
2.2 表达式求值：计算表达式的运算结果，如果包含未知数，返回一个包含未知数的简化表达式。
2.3 表达式优化：减少不必要的计算步骤，例如将 x+0 简化为 x，将 y*0 简化为 0

二．设计思想

T1.设计一个栈，利用链表的形式构建
T2.1 判断表达式的合法性：

a. 通过遍历输入的表达式，检验表达式中是否均为合法字符，包括数字，大小写字母，左右括号，和规定的+-*/四个运算符
b. 检验括号的匹配性：设计一个栈遍历字符串，将左括号直接插入，右括号插入前确定栈顶是否为左括号，若不是直接返回非法；若确为左括号直接删除栈顶元素，遍历结束后判断栈是否为空，若不为空，则为非法
c. 检验字符前内容的合法性：
   遍历字符串，确定类型后对于数字，字母，左括号，右括号，运算符均有进行判定的标准，并且注意在字符串结尾不得为运算符

T2.2.表达式第一次化简，构建函数 Simplify
T2.3 表达式第二次化简，将其中与 0 和 1 进行加减乘除的特殊情况进行考虑

三．程序效果

程序的运行效果，输入及输出的相关要求和具体执行结果如下所示：

Part1. 设计一个栈，利用链表的形式构建。
Part2.
案例一：
输入： ((3+4*(2-1))/5+x+0*y)
输出：
True
1.4+x+0*y
1.4+x
先是判断表达式的合法性，然后进行两次化简。
案例二：
输入： 3+4*(2-1)+
输出： False

四．核心代码

T1.

构建 node 结构体和 LinkStack 类(并实现 Add,Delete,GetTop,IsEmpty)

```
template<class T>
struct node
{
    public:
    T data;
    node* next;
    node()
    {
        next=NULL;
    }
    node(T val)
    {
        data=val;
        next=NULL;
    }
};
```

Node 结构体

```
class LinkStack
{
private:
    node<T>* head;
    int size;
public:
    LinkStack()
    {
        head=new node<T>();
        size=0;
    }
    ~LinkStack()
    {
        if(head==NULL) return;
        node<T>* tmp=head->next;
        while(tmp!=NULL)
        {
            delete head;
            head=tmp;
            tmp=tmp->next;
        }
        delete head;
    }
    void Add(T val)
    {
        node<T>* Insertnode=new node<T>(val);
        node<T>*tmp=head->next;
        head->next=Insertnode;
        Insertnode->next=tmp;
        size++;
    }
    void Delete()
    {
        if(IsEmpty()) return;
        node<T>* Deletenode=head->next;
        head->next=Deletenode->next;
        size--;
        delete Deletenode;
    }
    T GetTop()
    {
        node<T>* tmp=head->next;
        T var=tmp->data;
        return var;
    }
    bool IsEmpty()
    {
        return size==0;
    }
```

Link Stack 类

T2.Part1 判断表达式的合法性，若合法返回 True

```
//判断s中出现的字符是否为合法字符(同样起到了判断是否有错误运算符如 @ 的情况)
//只能为：运算符(+ - * /);操作数(数字和未知数(此处为字母));左右括号
for(int i=0;i<s.size();i++)
{
    if(s[i]=='('||s[i]==')') continue;
    if( IsNumber(s[i]) ) continue;
    if( IsAlpha(s[i]) ) continue;
    if(IsOperator(s[i])) continue;
    cout<<"不为合法字符:"<<s[i]<<endl;
    return false;
}
```

判断字符的合法性，避免有￥@等非法字符的出现

```
//括号不匹配
LinkStack<char> t1;
for(int i=0;i<s.size();i++)
{
    if(s[i]=='(') t1.Add(s[i]);
    if(s[i]==')')
    {
        if(t1.GetTop()!='(')
        {
            //cout<<"括号不匹配：有右括号无左括号"<<endl;
            return false;
        }
        t1.Delete();
    }
}
if(!t1.IsEmpty())
{
    //cout<<"括号不匹配：有左括号无右括号"<<endl;
    return false;
}
```

判断括号的匹配性

```
//运算符错误或语法错误
for(int i=1;i<=s.size();i++)
{
    if(i==s.size()-1)
    {
        if(IsOperator(s[i]))
        {
            //cout<<"结尾字符为运算符"<<endl;
            return false;
        }
    }
    if( IsNumber(s[i]) )
    {
        if(!( IsNumber(s[i-1]) ))
        {
            if(s[i-1]=='(') continue;//若操作数前为(，为合法的，继续进行遍历
            if( IsOperator(s[i-1]) ) continue;//若操作数前为运算符也可
            //cout<<"数字前面错误:"<<s[i]<<endl;
            return false;//如果某个操作数前为)时，由于右括号后必须要有运算符才能后接操作数
        }
    }
    if( IsOperator(s[i]) )
    {
        if(IsOperator(s[i-1]))
        {
            //cout<<"操作符前操作符: "<<s[i]<<endl;
            return false;//如果运算符前一个也为运算符，为非法操作
        }
        if(s[i-1]=='(')
        {
            //cout<<"操作符前为左括号"<<endl;
            return false;
        }
        continue;  //其余情况为合法操作
    }
    if( IsAlpha(s[i]) )
    {
        if(IsOperator(s[i-1])) continue;//未知数前一个为操作数，是可行的
        if(s[i-1]=='(') continue;
        //cout<<"未知数前错误: "<<s[i]<<endl;
        return false;
    }
    if(s[i]=='(')
    {
        if(IsOperator(s[i-1])) continue;
        if(s[i-1]=='(') continue;
        //cout<<"(前错误"<<endl;
        return false;
    }
    if(s[i]==')')
    {
        if(IsAlpha(s[i-1])) continue;
        if(IsNumber(s[i-1])) continue;
        if(s[i-1]==')') continue;
        //cout<<")前错误"<<endl;
        return false;
```

Part2.简化表达式（若为数值则计算得到值）

```
char c=s[i];
if(IsNumber(c))
{
    string num;
    while(i < s.size()&&(IsNumber(s[i])))
    {
        num = num + s[i++];
    }
    i--;
    number.Add(num);
}
else if(IsAlpha(c))
{
    string tmp(1,c);
    number.Add(tmp);
}
else if(c=='(')
{
    opr.Add(c);
}
```

```
else if(c==')')
{
    while(!opr.IsEmpty()&&opr.GetTop()!='(')  //运算符栈中不为空且未遍历到左括号
    {
        string r=number.GetTop();number.Delete();
        string l=number.GetTop();number.Delete();
        char oper=opr.GetTop();opr.Delete();
        if(string_IsAlpha(r)||string_IsAlpha(l))
        {
            string tmp=l+oper+r;
            number.Add(tmp);
        }
        else
        {
            string tmp=GetNewNumber(l,oper,r);
            number.Add(tmp);
        }
    }
    opr.Delete();//由于(未进入到 while循环中，在最后应该删除
}
```

```
            else if(IsOperator(c))
            {
                if(opr.IsEmpty()) opr.Add(c);
                else if(pre(opr.GetTop()) < pre(c)) opr.Add(c);
                else
                {
                    while(pre(opr.GetTop()) >= pre(c)) //t的优先级比c高
                    {
                        char oper=opr.GetTop();
                        opr.Delete();
                        string r=number.GetTop();number.Delete();
                        string l=number.GetTop();number.Delete();
                        if(string_IsAlpha(r)||string_IsAlpha(l))
                        {
                            string tmp=l+oper+r;
                            number.Add(tmp);
                        }
                        else
                        {
                            string tmp=GetNewNumber(l,oper,r);
                            number.Add(tmp);
                        }
                    }
                    opr.Add(c);
                }
            }
```

```
}
while(!opr.IsEmpty())
{
    char oper=opr.GetTop();
    opr.Delete();
    string r=number.GetTop();number.Delete();
    string l=number.GetTop();number.Delete();
    if(string_IsAlpha(r)||string_IsAlpha(l))
    {
        string tmp=l+oper+r;
        number.Add(tmp);
    }
    else
    {
        string tmp=GetNewNumber(l,oper,r);
        number.Add(tmp);
    }
}
return number.GetTop();
```

Part3.类似操作，增加判断

```
if(string_IsAlpha(r)&&string_IsAlpha(l))  //两个字符进行运算
{
    string tmp=l+oper+r;
    number.Add(tmp);
}
else if( (string_IsAlpha(r)) && l!=str0 && l!=str1 )
{
    string tmp=l+oper+r;
    number.Add(tmp);
}
else if( (string_IsAlpha(l)) && r!=str0 && r!=str1 )
{
    string tmp=l+oper+r;
    number.Add(tmp);
}
else if(string_IsAlpha(r)&&l==str0)//为：0 oper r
{
    if(oper=='+') number.Add(r);
    else if(oper=='/')
    {
        string tmp(1,'0');
        number.Add(tmp);
    }
    else if(oper=='*')
    {
        string tmp(1,'0');
        number.Add(tmp);
    }
```

```
    else if(string_IsAlpha(l)&&r==str0)//为：l oper 0
    {
        if(oper=='+') number.Add(l);
        else if(oper=='-') number.Add(l);
        else if(oper=='/') cout<<"不合法操作，不可以进行除以0操作";
        else if(oper=='*')
        {
            string tmp(1,0);
            number.Add(tmp);
        }
    }
    else if(string_IsAlpha(r)&&l==str1)//为：1 oper r
    {
        if(oper=='*')
        {
            number.Add(r);
        }
        else
        {
            string tmp=l+oper+r;
            number.Add(tmp);
        }
    }
```

```
    else if(string_IsAlpha(l)&&r==str1)//为：l oper 1
    {
        if(oper=='*')
        {
            number.Add(l);
        }
        else
        {
            string tmp=l+oper+r;
            number.Add(tmp);
        }
    }
    else  //两个数字进行运算
    {
        string tmp=GetNewNumber(l,oper,r);
        number.Add(tmp);
    }
}
opr.Delete();//由于(未进入到 while循环中，在最后应该删除
```

五．总结

通过本次实验，我对于栈的各项功能有了更加深入的理解：

利用栈实现了表达式的计算，让我理解了栈先进后出的性质