

实验三实验报告

2112514 辛浩然

实验内容

使用数组实现栈的出栈、入栈、获取栈顶元素、判断空栈四个方法。并使用该栈进行表达式运算。表达式包含“+”、“-”、“*”、“/”、“(”、“)”和数字的字符串，输出该表达式运算结果。

实验代码

```
#include <iostream>
using namespace std;
class NoMem
{
public:
    NoMem() {}
};
class OutOfBounds
{
public:
    OutOfBounds() {}
};
template <class T>
class Stack
{
    T* stack;
    int top;
    int maxsize;

public:
    Stack()
    {
        maxsize = 99;
        top = -1;
        stack = new T[100];
    }
    Stack(int max)
    {
        maxsize = max - 1;
        top = -1;
        stack = new T[max];
    }
    bool isEmpty() { return top == -1; }
    bool isFull() { return top == maxsize; }
    void Push(T x)
    {
        if (isFull())
        {
            throw NoMem();
        }
        top++;
        stack[top] = x;
    }
    T Pop()
```

```

{
    if (isEmpty())
    {
        throw OutOfBounds();
    }
    T x = stack[top];
    top--;
    return x;
}
T Top()
{
    if (isEmpty())
        throw OutOfBounds();
    return stack[top];
}
};

int isPrior(Stack<char> s, char x) //优先级是否高于栈顶元素
{
    if (s.isEmpty())
        return 1;
    int t = s.Top();
    if (t == '+' || t == '-')
    {
        if (x == '*' || x == '/' || x == '(')
            return 1;
        else
            return 0;
    }
    else if (t == '*' || t == '/')
    {
        if (x == '(')
            return 1; // '(' 高于左边任何运算符
        else
            return 0;
    }
    else if (t == '(')
    {
        if (x == ')')
        {
            return -1;
        }
        //括号匹配消除括号，单列
        return 1; // '(' 低于右边任何运算符
    }
    else
        return 0;
    // ')' 高于右边任何运算符
}

double caculate(char x, double m, double n)
{
    if (x == '+')
        return m + n;
    else if (x == '-')
        return n - m;
    else if (x == '*')
        return m * n;
    else
    {

```

```

        if (m == 0)
            throw n;
        return n / m;
    }
}
int judgeChar(char x)
{
    if (x >= '0' && x <= '9')
        return 0;
    else if (x == '+' || x == '*' || x == '/' || x == '-' || x == '(' || x == ')')
        return 1;
    else if (x == '.')
        return 2;
    else
        return 3;
}
int main()
{
    try
    {
        Stack<double> Operand; //操作数栈
        Stack<char> Operator; //操作符栈
        char x;
        char PopOperator;
        double Operand1, Operand2, res;
        bool isNegative = 0;
        char tmp;
        cin >> x;
        while (judgeChar(x) != 3)
            //输入其他字符结束输入
        {
            if (!judgeChar(x)) //数字
            {
                double i = 10, num = 0;
                while (!judgeChar(x)) //处理多位数
                {
                    num = num * i + x - '0';
                    cin >> x;
                }
                if (x == '.') //处理小数
                {
                    cin >> x;
                    double s = 0.1;
                    while (!judgeChar(x))
                    {
                        num += s * (x - '0');
                        s *= 0.1;
                        cin >> x;
                    }
                }
                Operand.Push(num);
                if (isNegative) //如果是负数，读完操作数后，后面再赋值加一个`)`，把读到的后面的运算
符先存到tmp中
                {
                    tmp = x;
                    x = ')';
                }
            }

```

```

if (judgeChar(x) == 1) //输入运算符
{
    if (x == '-' && Operator.isEmpty() && Operand.isEmpty())
        //如输入的表达式形如-3*4，如果读到的第一个字符是`-`，这个`-`一定是负号
    {
        Operand.Push(0);
        Operator.Push('(');
        Operator.Push('-');
        cin >> x;
        isNegative = 1;
    }
    else
    {
        int priority = isPrior(Operator, x);
        if (priority == 1) //比栈顶元素优先级高，压入栈
        {
            Operator.Push(x);
            cin >> x;
            if (x == '-')
            { //负号：紧跟在运算符（除`)`外）后的`-`符号
                Operand.Push(0);
                Operator.Push('(');
                Operator.Push('-');
                cin >> x;
                isNegative = 1;
            }
        }
        else if (!priority) //比栈顶元素优先级低：取出栈顶运算符和两个操作数，计算结果
            进数字栈，再与新的栈顶比较...
        {
            PopOperator = Operator.Pop();
            Operand1 = Operand.Pop(), Operand2 = Operand.Pop();
            res = caculate(PopOperator, Operand1, Operand2);
            Operand.Push(res);
            priority = isPrior(Operator, x);
        }
        else //如果输入的是右括号，如果左右括号相遇，左括号出来，右括号就不用进了
        {
            Operator.Pop();
            if (isNegative) //如果这个右括号是因为负数自己赋值的
            {
                x = tmp; //将存的tmp赋回x，继续读入表达式
                isNegative = 0;
            }
            else
            {
                cin >> x;
                if (x == '-') //特殊情况：右括号后的`-`是减号
                {
                    Operator.Push(x);
                    cin >> x;
                }
            }
        }
    }
}
}

while (!Operator.isEmpty()) //对两个栈剩余元素处理

```

```

    {
        PopOperator = Operator.Pop();
        Operand1 = Operand.Pop(), Operand2 = Operand.Pop();
        res = caculate(PopOperator, Operand1, Operand2);
        Operand.Push(res);
    }
    cout << Operand.Top(); //最终运算符栈空，操作数栈仅剩最终结果
}
catch (NoMem)
{
    cerr << "Stack is full" << endl;
}
catch (OutOfBounds)
{
    cerr << "Stack is empty" << endl;
}
catch (double)
{
    cerr << "Error of dividing zero" << endl;
}
}
}

```

实验测试

测试用例一

输入：

$((1+2)*3-5)/2 \times$

输出：

```

Microsoft Visual Studio 调试控制台
((1+2)*3-5)/2 x
2
C:\Users\DELL\source\repos\ConsoleApplication2\x64\Debug\ConsoleApplication2.exe (进程 15936)已退出，代码为 0。
按任意键关闭此窗口。 . . .

```

测试用例二

输入：

$20/(15*(1.5*(1.3+0.7)))$

输出：

```

Microsoft Visual Studio 调试控制台
20/(15*(1.5*(1.3+0.7))) x
0.444444
C:\Users\DELL\source\repos\ConsoleApplication2\x64\Debug\ConsoleApplication2.exe (进程 11216)已退出，代码为 0。
按任意键关闭此窗口。 . . .

```

测试用例三

输入：

$102+1.512*0.332-(6.612-(1.7*(1.2+101.234)))/2$

输出：

```

Microsoft Visual Studio 调试控制台
102+1.512*0.332-(6.612-(1.7*(1.2+101.234)))/2 x
182.959
C:\Users\DELL\source\repos\ConsoleApplication2\x64\Debug\ConsoleApplication2.exe (进程 14564)已退出，代码为 0。
按任意键关闭此窗口。 . . .

```

测试用例四

输入:

$-1 + -3 * -9.3 / 3 - (3.29 - -10.14 + -9.13)$

输出:



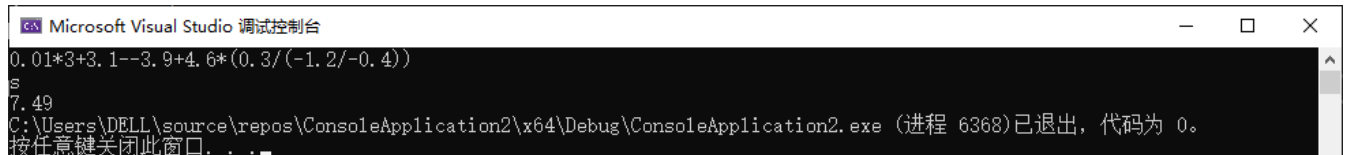
```
Microsoft Visual Studio 调试控制台
-1+-3*-9.3/3-(3.29--10.14+-9.13)
x
4
7.49
C:\Users\DELL\source\repos\ConsoleApplication2\x64\Debug\ConsoleApplication2.exe (进程 10512)已退出, 代码为 0。
按任意键关闭此窗口。 . . .
```

测试用例五

输入:

$0.01 * 3 + 3.1 - -3.9 + 4.6 * (0.3 / (-1.2 / -0.4))$

输出:



```
Microsoft Visual Studio 调试控制台
0.01*3+3.1--3.9+4.6*(0.3/(-1.2/-0.4))
S
7.49
C:\Users\DELL\source\repos\ConsoleApplication2\x64\Debug\ConsoleApplication2.exe (进程 6368)已退出, 代码为 0。
按任意键关闭此窗口。 . . .
```

测试用例六

输入:

$3 + -6 / (3 - 3)$

输出:



```
Microsoft Visual Studio 调试控制台
3+-6/(3-3)
S
Error of dividing zero
C:\Users\DELL\source\repos\ConsoleApplication2\x64\Debug\ConsoleApplication2.exe (进程 9680)已退出, 代码为 0。
按任意键关闭此窗口。 . . .
```

算法分析

中缀表达式求值

- 建立操作数栈和运算符栈，初始化为空；
- 从左至右读入表达式
- 如果读入的字符介于0-9之间，说明读的是操作数：
 - 将字符转换成数字，将得到的操作数赋值给变量 num，压入操作数栈。
 - 考虑到操作数是多位数的情况：在读入的字符介于0-9之间的条件下，写 while 循环，循环内修改 num 值为 $num * 10 + x$ ，继续读表达式，如果继续输入的字符介于0-9之间，继续循环；其他情况，跳出循环。
 - 考虑到操作数是小数的情况：在跳出多位数循环后，如果继续输入的字符是 .，说明该操作数是小数，后续读入的是小数位，后面读入的数字分别乘 0.1、0.01、0.001... 加到 num 中。
 - 最后将操作完的 num 压入操作数栈。
- 如果读入的字符是 + 或 - 或 * 或 / 或 (或)，说明读的是运算符：
 - 如果运算符栈空，将运算符压入运算符栈；
 - 如果读入的右括号，栈顶为左括号，压出左括号，继续读表达式。
 - 其他情况，将读到的运算符与运算符栈顶运算符进行优先级比较：
 - 栈顶的加减运算符优先级高于读到的加减运算符；
 - 栈顶的乘除运算符优先级高于读到的乘除、加减运算符；
 - 对于括号而言：
 - 栈顶的左括号优先级低于读入的所有运算符；
 - 栈顶的右括号优先级高于读入的所有运算符；

- 如果栈顶运算符优先级低于读入的运算符，直接将运算符压入栈；
- 如果栈顶运算符优先级高于读入的运算符，取出栈顶运算符和两个操作数，计算结果进操作数栈，再将读入的运算符与新运算符栈顶运算符比较...
- 特殊情况：操作数是负数
 - 如果操作数含负数，那么所读入的 - 就具有负号和减号两层含义，需要进行区分：
 - 跟在数字和右括号后面的一定是减号；
 - 跟在除右括号外其他运算符后面的，或是当两个栈都空时读到的 - （即表达式开头的 - ），一定是负号。
 - 对于负数的处理，如 -3，将其转换成 (0-3)。那么如果读入负号，
 - 将 (压入运算符栈；
 - 将 - 压入运算符栈；
 - 将 0 压入操作数栈；
 - 将继续读到的数字(考虑多位数和小数)转换后压入操作数栈；
 - 赋值读入)，计算括号内的结果压入操作数栈；
 - 继续读表达式。
- 表达式读完后，对两个栈中剩余元素进行处理，
 - 运算符栈压出一个元素，操作数栈压出两个元素，计算后将结果压入操作数栈；
 - 将栈中剩余运算符依次进行上述操作，并直到运算符栈空为止。
- 最终操作数栈的栈顶即为最终计算结果。

复杂度

时间复杂度为 $O(n)$ ，空间复杂度为 $O(1)$

心得总结

- 栈的数组表述以及中缀表达式求值的方法，在课堂讲过后，实现起来还是比较轻松的。
 - 在涉及到 (和) 时，思维有些混乱，在回看ppt后，理解了对于括号的相关操作。
 - 读完表达式中对栈剩余元素处理。一开始错误地认为只会剩余一个运算符，但随后在测试中发现了问题。如表达式 $3+6*6$ ，最终剩余 + 和 * 运算符，于是将程序修改为操作至运算符栈空。
- 在最初的时候，只考虑了一位数的计算。在经助教提醒后，意识到思维的不全面应考虑到小数、多位数、负数的存在。
 - 小数和多位数实现的较为顺利。
 - 在考虑负数时，遇到了一些问题，考虑的不全面导致出现了一些bug。如在一开始并没有意识到表达式最开始的 - 是负号的情况等。于是条理地梳理了下表达式中 - 是负号的情形对应地修改程序。
- 学习异常抛出操作，优化程序。
- 通过这次实验，对栈后进先出的性质理解更为深刻，对栈的应用也更加熟练。