山东大学 计算机科学与技术 学院

数据结构与算法 课程实验报告

学号：202000130198	姓名： 隋春雨	班级： 20.4
实验题目：栈		
实验学时：2	实验日期： 2021-11-4	

实验目的：

1、 掌握栈结构的定义与实现；

2、 掌握栈结构的使用。

软件开发环境：

CLION2020

1. 实验内容

题目描述：

创建栈类，采用数组描述；计算数学表达式的值。 输入数学表达式，输出表达式的计算结果。数学表达式由单个数字和运算符“+”、“-”、“*”、“/”、“(”、“) ”构成，例如 2+3*(4+5) - 6/4。

输入输出格式：

输入：

第一行一个整数 n(1<=n<=100)，代表表达式的个数。

接下来 n 行，每行一个表达式，保证表达式内的数字为单个整数，表达式内各运算符和数字间没有空格，且表达式的长度不超过 2000。

输出：

每行表达式输出一个浮点数，要求保留两位小数，保证输入表达式合法。

2. 数据结构与算法描述 （整体思路描述，所需要的数据结构与算法）

1. 算法：首先我们知道，对于计算机来说，中缀表达式是一个很复杂的结构，它读取到一个符号，不知道该不该进行运算，因为它后面的数据我们不知道，比如说 3*（6*（6/7*（4+2）)），进行判断及其的麻烦，所以我们采取后缀表达式（逆波兰式），中缀表达式是相对人类的思维结构来说比较简单的，对计算机而言中序表达式是非常复杂的结构。相对的，逆波兰式在计算机看来却是比较简单易懂的结构。因为计算机普遍采用的内存结构是栈式结构，它执行先进后出的顺序。首先用一个 string 来存储这个表达式，如果读取到数字，那么直接压入数字 stack。如果读取到了符号，若是(，那么直接压入，因为我们知道(在读取到的时候，它的优先级是最高的。

```
for (int i = 0; i < str.size(); i++)
{
    if (str[i] >= '0' && str[i] <= '9')//数字
    {
        num.push( theElement: double(str[i] - '0'));
    }
    else
    {
        if (str[i] == '(')
        {
            ope.push(str[i]);
        }
```

如果读取到了)，那么 stack 一直 pop，直到 pop 到了（。如果都不是，那么比较优先级。优先处理优先级高的符号，如果优先级相同（比如* /）,那么执行从左到右的顺序。值得注意的是，我们需要用一个函数来表示优先级，否则使用起来太麻烦了 。

```
int level(const char& c)
{
    switch (c)
    {
        case '+':
        case'-':
            return 0;
        case'*':
        case'/':
            return 1;
        default:
            return 0;
    }
}
```

当结束的时候，因为符号 stack 可能还有符号，那么我们直接一个一个 pop，并且计算即可

```
    while (!ope.empty())
    {
        char c = ope.top();
        ope.pop();
        double num2 = num.top();
        num.pop();
        double num1 = num.top();
        num.pop();
        num.push(caculate( &: num1,  &: num2,  &: c));
    }
```

2. 数据结构：我们在 1 中的算法中提及到了后进先出，那么我们很自然的想到了栈的这种存储结构

```cpp
template<class T>
class arrayStack
{
public:
    arrayStack(int initialCapacity = 10);
    ~arrayStack() { delete[] stack; }
    bool empty() const { return stackTop == -1; }
    int size() const
    {
        return stackTop + 1;
    }
    bool empty()
    {
        return stackTop == -1;
    }
    void checkEmpty()
    {
        if (empty())
        {
            throw "the stack is empty";
        }
    }
```

```cpp
    T& top()
    {
        checkEmpty();
        return stack[stackTop];
    }
    void pop()
    {
        checkEmpty();
        stackTop--; // destructor for T
    }
    void push(const T& theElement);
```

3. 测试结果（测试输入，测试输出）

输入：

输入

```
3
1+6/1*7+2*1*4+9/1+2*0*9+9+7/(9*5)-1*6-0*8-7-9*2+6-(0-5-2*8-7-9*5*(6-5*5*2*6-2-7-5+6*7+6*9-1*0*0+3*0+2/1-6/6+5))
0-4-1/6*(1-(6/7)-4+6+2+6*1)-1*7+2-8*2+0-(4+6-6*1+(3-8*6/4-6-5)*6/4/8+7-1*4/9*5)-0/6+1-0-2+7-2+6*4-3*6+2/8+6+1*6*2
5-3*9+5/1*5-9+1*8-6-8-4*1+5-2+9/3*2-2/5/(2-6)*2/7-9*0-2+4/6*6*7*8-8-8*6+8*9*(3+0*1/5/2*7*8+0-8*8-5+8/5*2-0)
```

输出：

```
3
1+6/1*7+2*1*4+9/1+2*0*9+9+7/(9*5)-1*6-0*8-7-9*2+6-(0-5-2*8-7-9*5*(6-5*5*2*6-2-7-5+6*7+6*9-1*0*0+3*0+2/1-6/6+5))
-9197.84
0-4-1/6*(1-(6/7)-4+6+2+6*1)-1*7+2-8*2+0-(4+6-6*1+(3-8*6/4-6-5)*6/4/8+7-1*4/9*5)-0/6+1-0-2+7-2+6*4-3*6+2/8+6+1*6*2
-3.47
5-3*9+5/1*5-9+1*8-6-8-4*1+5-2+9/3*2-2/5/(2-6)*2/7-9*0-2+4/6*6*7*8-8-8*6+8*9*(3+0*1/5/2*7*8+0-8*8-5+8/5*2-0)
-4362.57
```

提交 OJ 最后的结果：

✓Accepted

#	Result	Score	Time
1	✓Accepted	5	1 ms
2	✓Accepted	5	11 ms
3	✓Accepted	5	55 ms
4	✓Accepted	5	274 ms
5	✓Accepted	5	302 ms
6	✓Accepted	5	1 ms
7	✓Accepted	5	11 ms
8	✓Accepted	5	62 ms
9	✓Accepted	5	329 ms
10	✓Accepted	5	249 ms
11	✓Accepted	5	1 ms

4. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）

(1) 对于循环，我们一定一定要防止 RE，比如下面这组代码，我们可能最开始想的是要弹走栈顶，但是我们需要知道的是，弹走栈顶一定要保证非空，否则最后会导致数组下标越界，

```
    while (!ope.empty() && ope.top() != '('//探出并且计算
    {
        char c = ope.top();
        ope.pop();
        double num2 = num.top();
        num.pop();
        double num1 = num.top();
        num.pop();
        num.push(caculate( &: num1,  &: num2,  &: c));
    }
    ope.pop();//弹掉(
```

(2) 对于特殊情况一定要特殊讨论，比如我们计算一个括号中的表达式，应该考虑一下，栈顶是否为(，以及栈顶为(的时候，它的优先级是最高还是最低？经过我们的思考，我们得出结论，此时的优先级为最低，同时我们需要考虑先计算优先级高的，如果优先级相同，那么从左到右计算

```
while (!ope.empty() && ope.top() != '(' && level(ope.top()) >= level(str[i]))
{//找到比它优先级低的，或者最后是空
    char c = ope.top();
    ope.pop();
    double num2 = num.top();
    num.pop();
    double num1 = num.top();
    num.pop();
    num.push(caculate( &: num1,  &: num2,  &: c));
}
ope.push(str[i]);
```

(3) 对于边界条件的处理也一定要注意，这个 string 最后一定 stack 非空，那么我们需要边界处理一下，计算表达式

```
while (!ope.empty())//最后的处理
{
    char c = ope.top();
    ope.pop();
    double num2 = num.top();
    num.pop();
    double num1 = num.top();
    num.pop();
    num.push(caculate( &: num1,  &: num2,  &: c));
}
```

(4) 值得注意的是，一定一定要记得更新私有成员，因为 public 接口的函数是没有记忆性的，只有靠着

私有成员才能判断自身的状态。并且不要硬编码，如果硬编码的话，写的时间会很长，并且写代码出 bug 的几率也会更大，所以我们一定要先想好思路再开始，否则会很麻烦，比如下图

```
int level(const char& c)//用来计算等级的，也就是优先级
{
    switch (c)
    {
        case '+':
        case'-':
            return 0;
        case'*':
        case'/':
            return 1;
        default:
            return 0;
    }
}
```

如果是暴力解决也可以硬枚举出来，但是用函数可以封装的更好更优雅，如果出了 Bug 改起来更快，也更容易。所以更推荐用函数解决。

(5) 计算表达式的时候，一定要分清谁去计算谁，比如下图，也反映了我们计算的时候思路清不清晰，要先有思路，再有代码

```
double caculate(double& a, double& b, char& c)//计算表达式
{
    switch (c)
    {
        case'+':
            return a + b;
        case'-':
            return a - b;
        case'*':
            return a * b;
        case'/':
            return a / b;//一定一定要注意除法的使用，因为a/b!=b/a
        default:
            throw "Error";
    }
}
```

(6) 自己写的时候测的样例都是对的，交到 oj 平台上就 RE 了，怎么办？

解决：RE 常见情况的是数组下标越界，但是经过自己 debug 发现，实际情况是 switch case 条件没有 break 语句，才 RE，在平时，能用 switch case 尽量用 switch case 而不是 If else ，因为 switch case 执行的次数少。

5. 附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释）

```
#include<iostream>
#include <sstream>
#include<string>
#include<iomanip>
#include<stack>
using namespace std;
template<class T>
class arrayStack
{
public:
    arrayStack(int initialCapacity = 10);//初始化
    ~arrayStack() { delete[] stack; }//析构
    bool empty() const { return stackTop == -1; }
    int size() const//size 函数
    {
        return stackTop + 1;
    }
    bool empty()//判断是否为空
    {
        return stackTop == -1;
    }
    void checkEmpty()
    {
        if (empty())
        {
            throw "the stack is empty";
        }
    }
    T& top()//返回栈顶
    {
        checkEmpty();
        return stack[stackTop];
    }
    void pop()//弹出栈顶
    {
        checkEmpty();
```

```
37.         stackTop--; // destructor for T
38.     }
39.     void push(const T& theElement);
40. 
41. private:
42.     int stackTop;        // current top of stack
43.     int arrayLength;     // stack capacity
44.     T* stack;            // element array
45. 
46. };
47. 
48. template<class T>
49. arrayStack<T>::arrayStack(int initialCapacity)
50. {
51.     arrayLength = initialCapacity;
52.     stack = new T[arrayLength];
53.     stackTop = -1;//表示栈顶的位置，并且-1 也表示空
54. }
55. 
56. template<class T>
57. void arrayStack<T>::push(const T& theElement)
58. {
59.     if (stackTop == arrayLength - 1)//如果满了，那么扩容
60.     {
61.         T* temp = new T[2 * arrayLength];
62.         copy(stack, stack + arrayLength, temp);
63.         delete[]stack;
64.         stack = temp;
65.         arrayLength *= 2;//更新私有变量
66.     }
67.     stack[++stackTop] = theElement;
68. }
69. 
70. 
71. int level(const char& c)//用来计算等级的，也就是优先级
72. {
73.     switch (c)
74.     {
75.         case '+':
76.         case'-':
77.             return 0;
78.         case'*':
79.         case'/':
80.             return 1;
81.         default:
82.             return 0;
```

```
    }
}

double caculate(double& a, double& b, char& c)//计算表达式
{
    switch (c)
    {
        case'+':
            return a + b;
        case'-':
            return a - b;
        case'*':
            return a * b;
        case'/':
            return a / b;//一定一定要注意除法的使用，因为 a/b!=b/a
        default:
            throw "Error";
    }
}


int main()
{

    int n;
    cin >> n;
    for (int j = 0; j < n; j++)
    {
        string str;
        cin >> str;
        arrayStack<double>num;//创建两个 stack
        arrayStack<char>ope;//符号 stack
        for (int i = 0; i < str.size(); i++)
        {
            if (str[i] >= '0' && str[i] <= '9')//数字
            {
                num.push(double(str[i] - '0'));//数字直接 push
            }
            else
            {
                if (str[i] == '(')
                {
                    ope.push(str[i]);//(直接 push
                }
                else if (str[i] == ')')
                {
```

```
129.                 while (!ope.empty() && ope.top() != '('//探出并且计算
130.                 {
131.                     char c = ope.top();
132.                     ope.pop();
133.                     double num2 = num.top();
134.                     num.pop();
135.                     double num1 = num.top();
136.                     num.pop();
137.                     num.push(caculate(num1, num2, c));
138.                 }
139.                 ope.pop();//弹掉(
140.             }
141.             else
142.             {//+ - * /
143.                 if (ope.empty() || ope.top() == '('//如果是空，或者栈顶是(那么直接压入
144.                 {
145.                     ope.push(str[i]);
146.                 }
147.                 else
148.                 {
149.                     while (!ope.empty() && ope.top() != '(' && level(ope.top()) >= level(str[i]))
150.                     {//找到比它优先级低的，或者最后是空
151.                         char c = ope.top();
152.                         ope.pop();
153.                         double num2 = num.top();
154.                         num.pop();
155.                         double num1 = num.top();
156.                         num.pop();
157.                         num.push(caculate(num1, num2, c));
158.                     }
159.                     ope.push(str[i]);
160.                 }
161. 
162.             }
163.         }
164.     }
165.     while (!ope.empty())//最后的处理
166.     {
167.         char c = ope.top();
168.         ope.pop();
169.         double num2 = num.top();
170.         num.pop();
171.         double num1 = num.top();
172.         num.pop();
173.         num.push(caculate(num1, num2, c));
174.     }
```

```
175.         cout << fixed << setprecision(2) << num.top() << endl;//输出两位
176.
177.     }
178.
179.
180.
181.     return 0;
182. }
```