数据结构课程设计

项目说明文档

表达式转换

软件工程

张靖凯

2151396

目录

项目内容

算数表达式有前缀表示法，中缀表示法和后缀表示法等形式。日常使用的算术表达式是采用中缀表示法，即二元运算符位于两个运算数中间。请设计程序将中缀表达式转换成为后缀表达式。

项目要求

1 输入说明：输入在一行中给出以空格分隔不同对象的中缀表达式，可包含+, -, *, /, -, *, /以及左右括号，表达式不超过 20 个字符（不包括空格）。

2 输出说明：在一行中输出转换后的后缀表达式，要求不同对象（运算数，运算符号）之间以空格分隔，但是结尾不得有多余空格。

3 测试用例：

序号	输入	输出	说明
1	2 + 3 * (7 – 4) + 8 / 4	2 3 7 4 - * + 8 4 / +	正常测试 6 种运算符
2	((2 + 3) * 4 – (8 + 2)) / 5	2 3 + 4 * 8 2 + - 5 /	嵌套括号
3	1314 + 25.5 * 12	1314 25.5 12 * +	运算数超过 1 位整数且有非整数出现
4	-2 * (+3)	-2 3 *	运算数有正或负号
5	123	123	只有 1 个数字

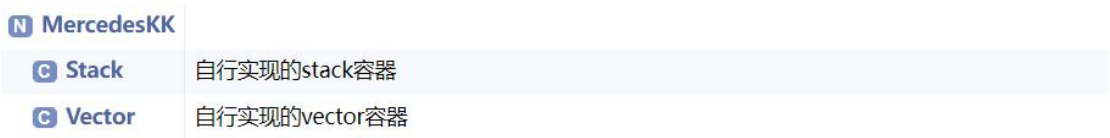
项目设计

数据结构设计

使用类模板实现栈容器，采用顺序结构，默认初始栈大小为 10，扩容方式仿照 STL 的 vector，每次超过最大容量的时候分配内存翻倍。当栈为空时栈顶指针指向-1。

类设计说明

命名空间 MercedesKK 中实现了 Stack 和 Vector，仿照 STL。



Stack 类

支持栈的 pop，push，top，size，isEmpty 基础功能，每当超过栈的容量时会调用__ExpanCapacity()函数扩容。

Public Types

using **size_type** = size_t
using **value_type** = T

Public Member Functions

Stack (**size_type** n=10)
Constructors. More...

~Stack ()

Operations

void **push** (const **value_type** &)
void **pop** ()
value_type & **top** () const
bool **isEmpty** () const
size_type **size** () const

Private Member Functions

void **__ExpanCapacity** ()
扩充数组函数 More...

Private Attributes

size_type **_capacity**
容量 More...

int **_top**
栈顶 More...

value_type * **_ptrStack**
一维数组指针 More...

Vector 类

设计了配有迭代器的 vector 容器。

由于 vector 在双向移动的基础上，支持前后位置的比较、随机存取、直接移动 n 个距离，故迭代器类型为 random-access iterator，为了方便，直接 using iterator = T*来实现，因为指针有如上的性质。

实现了默认构造、拷贝构造、拷贝赋值。重载了[]运算符，操作类函数模拟 STL 实现。

Detailed Description

template<typename T>
class MercedesKK::Vector< T >

自行实现的vector容器

实现了元素数量超出_size时容量翻倍的操作 简化，迭代器在Vector类内直接实现

Public Types

using **value_type** = T
using **iterator** = T *

Public Member Functions

iterator	**begin** ()
iterator	**end** ()

Constructors

	Vector ()
	~Vector ()
	Vector (const **Vector** &vec)
Vector &	**operator=** (const **Vector** &vec)

重载运算符

value_type &	**operator[]** (size_t index)
bool	**operator==** (const **Vector** &vec) const

Operations

void	**push_back** (**value_type** val)
void	**pop_back** ()
size_t	**Size** () const
size_t	**capacity** () const
bool	**empty** ()
void	**clear** ()
value_type	**front** () const
value_type	**back** () const
void	**insert** (**iterator** it, **value_type** val)
void	**erase** (**iterator** it)

Private Attributes

value_type *	**_data** 动态分配实现的数组 More...
size_t	**_size** 已有元素数量 More...
size_t	**_capacity** 容器容量 More...

解题思路

1）如果遇到操作数，我们就直接将其输出。

2）如果遇到操作符，则我们将其放入到栈中，遇到左括号时我们也将其放入栈中。

3）如果遇到一个右括号，则将栈元素弹出，将弹出的操作符输出直到遇到左括号为止。注意，左括号只弹出并不输出。

4）如果遇到任何其他的操作符，如“+”，“*”，“/”等，从栈中弹出元素直到遇到发现更低优先级的元素(或者栈为空)为止。弹出完这些元素后，才将遇到的操作符压入到栈中。有一点需要注意，只有在遇到"）“的情况下我们才弹出"(", 其他情况我们都不会弹出"("。

5）如果我们读到了输入的末尾，则将栈中所有元素依次弹出。

关键代码

运算符的优先级定义如下：

```
char priority[255];    ///< 存运算符优先级
```

仿照 map 思想，用 char 数组里 ASCII 码对应数字，即 char->int

```
    // 用数组来存
    priority['*'] = priority['/'] = 3;
    priority['+'] = priority['-'] = 2;
    priority['('] = 1;
    priority['#'] = 0;
```

正负号及小数点处理如下：

1. 带正负号且前一个字符为运算符（i=0 时直接带正负号的也是数字）
2. 当前字符为数字

```
if (((str[i] == '-' || str[i] == '+') && (i == 0 ||
string("+-*/(").find(str[i - 1]) != string::npos)) ||
isdigit(str[i])) ///< 右括号不需要，因为一定是减号
        {
            // 把操作数加入到后缀式中
            // 如果是正号就不用加入，负号或者数字本身都要加入
            tempStr = str[i] != '+' ? str.substr(i, 1) : "";
            while (i + 1 < str.size() && operators.find(str[i + 1])
== string::npos) ///< 小数点也算进去了已经
                tempStr += str[++i];
            ans.push_back(tempStr);
        }
```

遇到任何其他的操作符，如“+”，“*”，“/”等，从栈中弹出元素直到遇到发现更低优先级的元素(或者栈为空)为止：

```
while (priority[str[i]] <= priority[ops.top()])
{
    ans.push_back(string(1, ops.top()));
    ops.pop();
}
```

设计亮点

代码注释规范

采用了 doxygen 注释规范，对类、函数等有简要说明，命名采用驼峰命名法，类内的各个声明规范，方便本人回顾之前写过的代码和 code reviewer 查看，使 API 规范，帮助这个开发流程高效、规范地进行。

```
    /// @name Operations
    /// @{
    void push(const value_type&);
    void pop();

    value_type& top() const;
    bool isEmpty() const;
    size_type size() const;
    /// @}
    /// end of Operations

rivate:
    size_type _capacity;          ///< 容量
    int _top;                     ///< 栈顶
    value_type* _ptrStack;        ///< 一维数组指针

    void __ExpanCapacity();       ///< 扩充数组函数
```

代码测试

```
2+3*(7-4)+8/4
2 3 7 4 - * + 8 4 / +
```

```
((2+3)*4-(8+2))/5
2 3 + 4 * 8 2 + - 5 /
```

```
1314+25.5*12
1314 25.5 12 * +
```

```
-2*(+3)
-2 3 *
```

```
123
123
```

Linux 下测试（Ubuntu）

kk@LAPTOP-UJDPHKT8: ~/da

```
67
68 int main()
69 {
70     // 用数组来存
71     priority['*'] = priority['/'] = 3;
72     priority['+'] = priority['-'] = 2;
73     priority['('] = 1;
74     priority['#'] = 0;
75
76     string str;
77     getline(cin, str); // 读取空格,c++函数
78
79     trans(str);
80
81     for (int i = 0; i < ans.Size(); i++)
82         cout << (i ? " " : "") << ans[i];
83
84     cin.clear();
85     cin.ignore(1024, '\n');
86
87     cout << endl << "Enter to Exit";
88     cin.get();
89
90     return 0;
91 }
~
~
h4.cpp
```

```
kk@LAPTOP-UJDPHKT8:~$ ls
dataStruct
kk@LAPTOP-UJDPHKT8:~$ cd dataStruct/
kk@LAPTOP-UJDPHKT8:~/dataStruct$ ls
homework1  homework2  homework3  homework4  homework5  homework6  homework7  test
kk@LAPTOP-UJDPHKT8:~/dataStruct$ cd homework4
kk@LAPTOP-UJDPHKT8:~/dataStruct/homework4$ ls
Stack.hpp  Vector.hpp  h4.cpp  run
kk@LAPTOP-UJDPHKT8:~/dataStruct/homework4$ ./run
2+3*(7-4)+8/4
2 3 7 4 - * + 8 4 / +

Enter to Exit
kk@LAPTOP-UJDPHKT8:~/dataStruct/homework4$
```