栈

栈常见问题

匹配与消除

普通括号匹配

```
def par_checker(symbol_string):
    s = [] # Stack()
    balanced = True
    index = 0
    while index < len(symbol_string) and balanced:
        symbol = symbol_string[index]
        if symbol in "([{":
            s.append(symbol) # push(symbol)
        else:
            top = s.pop()
            if not matches(top, symbol):
                balanced = False
        index += 1
        #if balanced and s.is_empty():
        if balanced and not s:
            return True
        else:
            return False

def matches(open, close):
    opens = "([{"
    closes = ")]}"
    return opens.index(open) == closes.index(close)

print(par_checker('{{}}[]]'))
```

带有标记括号匹配

```
lines = []
while True:
    try:
        lines.append(input())
    except EOFError:
        break

ans = []
for s in lines:
    stack = []
    Mark = []
    for i in range(len(s)):
        if s[i] == '(':
            stack.append(i)
            Mark += ' '
        elif s[i] == ')':
            if len(stack) == 0:
                Mark += '?'
            else:
                Mark += ' '
                stack.pop()
        else:
            Mark += ' '

    while(len(stack)):
        Mark[stack[-1]] = '$'
        stack.pop()

    print(s)
    print(''.join(map(str, Mark)))
```

十进制转八

```python
decimal = int(input())  # 读取十进制数

# 创建一个空栈
stack = []

# 特殊情况：如果输入的数为0，直接输出0
if decimal == 0:
    print(0)
else:
    # 不断除以8，并将余数压入栈中
    while decimal > 0:
        remainder = decimal % 8
        stack.append(remainder)
        decimal = decimal // 8

    # 依次出栈，构成八进制数的各个位
    octal = ""
    while stack:
        octal += str(stack.pop())

    print(octal)
```

表达式问题

波兰表达式

```
lis = []
n = input().split()

for i in n[::-1]:
    if i not in ['+','-','*','/']:
        lis.append(i)

    if i in ['+','-','*','/']:
        a = float(lis.pop())
        b = float(lis.pop())
        if i == '+':
            lis.append(a + b)
        elif i == '-':
            lis.append(a - b)
        elif i == '*':
            lis.append(a * b)
        elif i == '/':
            lis.append(a / b)

for i1 in lis:
    print(f"{i1:.6f}")
```

中序转后序调度场（含浮点数接收）

```
def infix_to_postfix(expression):
    precedence = {'+':1, '-':1, '*':2, '/':2}
    stack = []
    postfix = []
    number = ''

    for char in expression:
        if char.isnumeric() or char == '.':
            number += char
        else:
            if number:
                num = float(number)
                postfix.append(int(num) if num.is_integer() else num)
                number = ''
            if char in '+-*/':
                while stack and stack[-1] in '+-*/' and precedence[char] <= precedence[stack[-1]]:
                    postfix.append(stack.pop())
                stack.append(char)
            elif char == '(':
                stack.append(char)
            elif char == ')':
                while stack and stack[-1] != '(':
                    postfix.append(stack.pop())
                stack.pop()

    if number:
        num = float(number)
        postfix.append(int(num) if num.is_integer() else num)

    while stack:
        postfix.append(stack.pop())

    return ' '.join(str(x) for x in postfix)

n = int(input())
for _ in range(n):
    expression = input()
    print(infix_to_postfix(expression))
```

前序用栈从后往前遍历，后序从前往后