【第三周】递归与栈：解决表达式求值

20. 有效的括号

https://leetcode-cn.com/problems/valid-parentheses/

```
class Stack {
public:
    Stack (int n = 100) {
        top = 0;
        data = new int[n];
    }
    void push(int val) {data[top++] = val;}
    void pop() {top--;}
    int getTop() {return data[top - 1];}
    bool empty() {return top == 0;}
    int size() {return top;}
private:
    int top, *data;
};

class Solution {
public:
    bool matched(char x, char y) {
        if (x == '(' && y == ')') return true;
        if (x == '[' && y == ']') return true;
        if (x == '{' && y == '}') return true;
        return false;
    }

    bool isValid(string s) {
        Stack sta(10005);
        for (int i = 0; i < s.size(); i++) {
            if (s[i] == '(' || s[i] == '[' || s[i] == '{') {
                sta.push(s[i]);
            }
            else {
                if (sta.empty() || !matched(sta.getTop(), s[i])) {
                    return false;
                }
                sta.pop();
            }
        }
        return sta.empty();
    }
};
```

表达式求值【递归法】

```cpp
#include <bits/stdc++.h>
using namespace std;

// s : string
// l, r : s[l] ~ s[r]
//
//
// define : 是求出字符串s中的 s[l] 到 s[r] 这一段子串的表达式的值
int calculate(char *s, int l, int r) {
    // find min priority
    int delta = 0;
    int op = -1, minPrio = 100000;
    for (int i = l; i <= r; i++) {
        int prio = 100000 + 1;
        if (s[i] == '+' || s[i] == '-') prio = 1 + delta;
        else if (s[i] == '*' || s[i] == '/') prio = 2 + delta;
        else if (s[i] == '(') delta += 100;
        else if (s[i] == ')') delta -= 100;
        if (minPrio >= prio) { minPrio = prio; op = i; }
    }
    if (op == -1) {
        // no opreator
        int num = 0;
        for (int i = l; i <= r; i++) {
            if (s[i] >= '0' && s[i] <= '9') {
                num = num * 10 + (s[i] - '0');
            }
        }
        return num;
    }
    // 我现在要求 s中的s[l] ~ s[op-1] 的这一段子串的表达式的值
    int a = calculate(s, l, op - 1);
    int b = calculate(s, op + 1, r);
    //
    if (s[op] == '+') return a + b;
    if (s[op] == '-') return a - b;
    if (s[op] == '*') return a * b;
    if (s[op] == '/') return a / b;
    return 0;
}


int main() {
    char s[10005];
    while (~scanf("%s", s)) {
        int ans = calculate(s, 0, strlen(s) - 1);
        printf("the result of %s is %d\n", s, ans);

    }

  return 0;
}
```

232. 用栈实现队列

https://leetcode-cn.com/problems/implement-queue-using-stacks/

```
class MyQueue {
public:
    MyQueue() {}
    void push(int x) {s2.push(x);}
    void move() {
        while (!s2.empty()) {
            int val = s2.top();
            s2.pop();
            s1.push(val);
        }
    }
    int pop() {
        if (s1.empty()) {move();}
        int ret = s1.top();
        s1.pop();
        return ret;
    }
    int peek() { // front
        if (s1.empty()) {move();}
        return s1.top();
    }
    bool empty() {return s1.empty() && s2.empty();}
private:
    stack<int> s1, s2;
};
```

682. 棒球比赛

https://leetcode-cn.com/problems/baseball-game/

```
class Solution {
public:

    int toInt(string s) {
        int num = 0;
        for (int i = 0; i < s.size(); i++) {
            if (i == 0 && s[i] == '-') continue;
            num = num * 10 + (s[i] - '0');
        }
        return s[0] == '-' ? -num : num;
    }

    int calPoints(vector<string>& ops) {
        stack<int> s;
        for (int i = 0; i < ops.size(); i++) {
            if (ops[i][0] == '+') {
                int x = s.top(); s.pop();
                int y = s.top();
                s.push(x);
                s.push(x + y);
            }
            else if (ops[i][0] == 'D') s.push(2 * s.top());
            else if (ops[i][0] == 'C') s.pop();
            else s.push(toInt(ops[i]));
            // s.push(atoi(ops[i].c_str()));
        }
        int ans = 0;
        while (!s.empty()) {
            ans += s.top();
            s.pop();
        }
        return ans;
    }
};
```

844. 比较含退格的字符串

https://leetcode-cn.com/problems/backspace-string-compare/3

```
class Solution {
public:
    stack<int> removeBackSpace(string &str) {
        stack<int> s;
        for (int i = 0; i < str.size(); i++) {
            if (str[i] == '#') {
                if (!s.empty()) s.pop();
            }
            else s.push(str[i]);
        }
        return s;
    }

    bool backspaceCompare(string s, string t) {
        stack<int> s1 = removeBackSpace(s);
        stack<int> s2 = removeBackSpace(t);
        while (!s1.empty() && !s2.empty()) {
            int a = s1.top(); s1.pop();
            int b = s2.top(); s2.pop();
            if (a != b) return false;
        }
        return s1.empty() && s2.empty();
    }
};

```

946. 验证栈序列

https://leetcode-cn.com/problems/validate-stack-sequences/

```
class Solution {
public:
    bool validateStackSequences(vector<int>& pushed, vector<int>& popped) {
        // i->push j->pop
        int i = 0, j = 0;
        stack<int> s;
        while (j < popped.size()) {
            while (s.empty() || s.top() != popped[j]) {
                if (i >= pushed.size()) break;
                s.push(pushed[i]);
                i++;
            }
            if (s.top() != popped[j]) return false;
            s.pop();
            j++;
        }
        return true;
    }
};
```

1249. 移除无效的括号

https://leetcode-cn.com/problems/minimum-remove-to-make-valid-parentheses/

```
class Solution {
public:
    string minRemoveToMakeValid(string s) {
        int needRemove[s.size()];
        memset(needRemove, 0, sizeof(needRemove));
        stack<int> sta;
        for (int i = 0; i < s.size(); i++) {
            if (s[i] == '(') {
                sta.push(i);
            }
            else if (s[i] == ')') {
                if (sta.empty()) needRemove[i] = 1;
                else sta.pop();
            }
        }
        while (!sta.empty()) {
            needRemove[sta.top()] = 1;
            sta.pop();
        }
        string ans = "";
        for (int i = 0; i < s.size(); i++) {
            if (!needRemove[i]) ans += s[i];
        }
        return ans;
    }
};
```

1021. 删除最外层的括号

https://leetcode-cn.com/problems/remove-outermost-parentheses/

```
class Solution {
public:
    string removeOuterParentheses(string s) {
        int cnt = 0, start = 0;
        string ans = "";
        for (int i = 0; i < s.size(); i++) {
            if (s[i] == '(') cnt++;
            else cnt--;
            if (cnt == 0) {
                // i - start + 1 - 2
                ans += s.substr(start + 1, i - start - 1);
```

```cpp
                start = i + 1;
            }
        }
        return ans;
    }
};
```

1124. 表现良好的最长时间段

https://leetcode-cn.com/problems/longest-well-performing-interval/

```cpp
// 显式使用前缀和数组
class Solution {
public:
    int longestWPI(vector<int>& hours) {
        vector<int> pre;
        pre.push_back(0);
        for (int i = 0; i < hours.size(); i++) {
            int t = hours[i] > 8 ? 1 : -1;
            pre.push_back(pre.back() + t);
        }
        map<int, int> prePos;
        int ans = 0;
        for (int i = 0; i < pre.size(); i++) {
            // 最早出现的 pre[i] 的位置是 i
            if (!prePos.count(pre[i])) prePos[pre[i]] = i;
            if (prePos.count(pre[i]  - 1)) {
                // [j ~ i]
                ans = max(ans, i - prePos[pre[i] - 1]);
            }
            // [0 ~ i]
            if (pre[i] > 0) ans = max(ans, i);
        }

        return ans;
    }
};
```

```cpp
// 隐式使用前缀和数组
class Solution {
public:
    int longestWPI(vector<int>& hours) {
        map<int, int> prePos;
        int ans = 0, pre = 0;
        prePos[0] = 0;
        for (int i = 0; i < hours.size(); i++) {
            // 最早出现的 pre[i] 的位置是 i
            pre += hours[i] > 8 ? 1 : -1;
            if (!prePos.count(pre)) prePos[pre] = i;
            if (prePos.count(pre - 1)) {
                // [j ~ i]
                ans = max(ans, i - prePos[pre - 1]);
            }
            // [0 ~ i]
            if (pre > 0) ans = max(ans, i + 1);
        }

        return ans;
    }
};
```

227. 基本计算器 II

https://leetcode-cn.com/problems/basic-calculator-ii/

```cpp
class Solution {
public:
    int calculate(string s) {
        s += '#';
        stack<int> sta;
        int num = 0, lastOp = '+';
        for (int i = 0; i < s.size(); i++) {
            if (s[i] == ' ') continue;
            if (s[i] >= '0' && s[i] <= '9') {
                num = num * 10 + (s[i] - '0');
            }
            else {
                if (lastOp == '+') sta.push(num);
                else if (lastOp == '-') sta.push(-num);
                else if (lastOp == '*') sta.top() *= num;
                else if (lastOp == '/') sta.top() /= num;
                num = 0;
                lastOp = s[i];
            }
        }
        int ans = 0;
        while (!sta.empty()) {
            ans += sta.top();
            sta.pop();
        }
        return ans;
    }
};
```

636. 函数的独占时间

https://leetcode-cn.com/problems/exclusive-time-of-functions/

```cpp
class Solution {
public:
    vector<int> exclusiveTime(int n, vector<string>& logs) {
        // ans[i] 代表 函数i 的总执行时间
        vector<int> ans(n, 0);
        stack<int> s;
        int lastTimeStamp = 0;
        for (int i = 0; i < logs.size(); i++) {
            int id = 0, timeStamp = 0;
            int l = 0, r = logs[i].size() - 1;
            while (logs[i][l] != ':') {
                id = id * 10 + (logs[i][l] - '0');
                l++;
            }
            int base = 1;
            while (logs[i][r] != ':') {
                timeStamp += base * (logs[i][r] - '0');
                base *= 10;
                r--;
            }
            if (r - l == 6) { // start
                if (!s.empty()) ans[s.top()] += timeStamp - lastTimeStamp;
                lastTimeStamp = timeStamp;
                s.push(id);
            }
            else {
                ans[s.top()] += timeStamp - lastTimeStamp + 1;
                lastTimeStamp = timeStamp + 1;
                s.pop();
            }
        }
        return ans;
    }
};
```