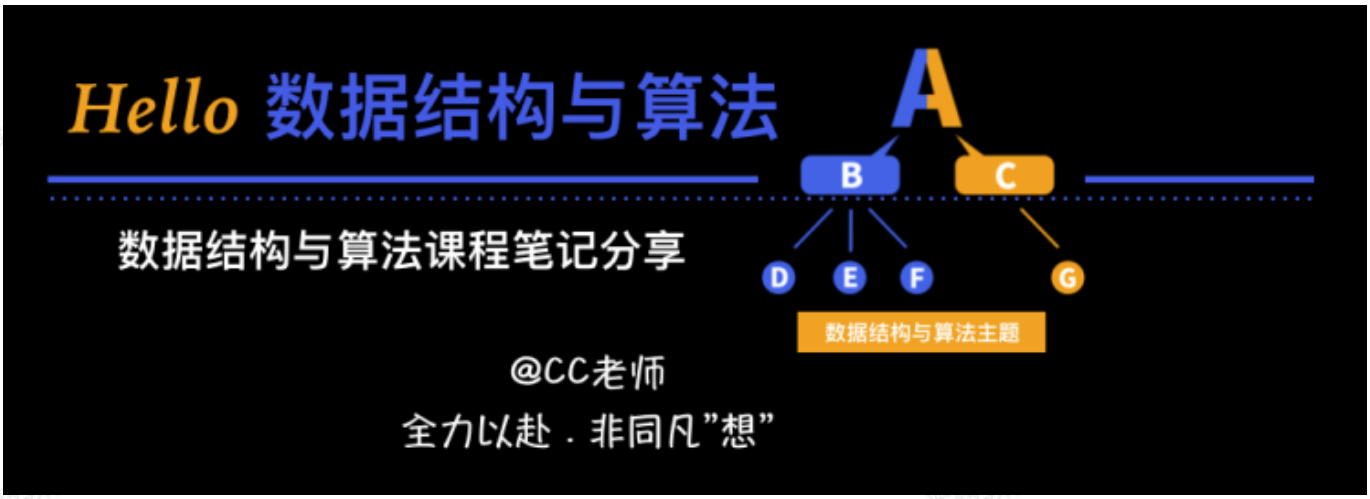
005-数据结构与算法 [栈与队列主题 1]



一. 课程安排:

- **课程日期: 2020 年 4 月 10 日 第 5 次课程 (共9次课程)**
- **授课老师: CC 老师**
- **研发老师: CC 老师**
- **课程时长: 2小时**
- **课程主题: 如何设计一个栈结构(顺序存储/链式存储角度下)**
- **课程时间安排:**
  - 上课: 20:00 – 21:00
  - 休息: 21:00 – 21:10
  - 上课: 21:10 – 22:00

- **课程作业:**
  - **博客,如何基于顺序存储/链式存储的不同角度下设计一个栈结构**
  - 实现循环队列的顺序存储实现(初始化/清空/判断是否空/队列长度/获取队头/出队/入队/遍历)
  - 实现下面算法题:
    - **括号匹配检验: (字节出现过的算法面试题/LeetCode)**
      假设表达式中允许包含两种括号:圆括号与方括号,其嵌套顺序随意,即([]()) 或者[([][])]都是正确的.而这[(]或者(())]或者([()) 都是不正确的格式. 检验括号是否匹配的方法可用"期待的急迫程度"这个概念来描述.例如,考虑以下括号的判断: [([ ] [ ])]
    - **每日气温: (LeetCode-中等)**
      题目: 根据每日气温列表，请重新生成一个列表，对应位置的输入是你需要再等待多久温度才会升高超过该日的天数。如果之后都不会升高，请在该位置0来代替。例如，给定一个列表 temperatures = [73, 74, 75, 71, 69, 72, 76, 73]，你的输出应该是 [1, 1, 4, 2, 1,

1, 0, 0]。提示：气温 列表长度的范围是 [1, 30000]。每个气温的值的均为华氏度，都是在 [30, 100] 范围内的整数

- **爬楼梯问题:(LeetCode-中等)**
  假设你正在爬楼梯。需要 n 阶你才能到达楼顶。每次你可以爬 1 或 2 个台阶。你有多少种不同的方法可以爬到楼顶呢？注意：给定 n 是一个正整数å
- **去除重复字母(LeetCode-困难)**
  给你一个仅包含小写字母的字符串，请你去除字符串中重复的字母，使得每个字母只出现一次。需保证返回结果的字典序最小（要求不能打乱其他字符的相对位置）
- **字符串编码(LeetCode-中等)**
  给定一个经过编码的字符串，返回它解码后的字符串。
  编码规则为: k[encoded_string]，表示其中方括号内部的 encoded_string 正好重复 k 次。注意 k 保证为正整数。你可以认为输入字符串总是有效的；输入字符串中没有额外的空格，且输入的方括号总是符合格式要求的。此外，你可以认为原始数据不包含数字，所有的数字只表示重复的次数 k ，例如不会出现像 3a 或 2[4] 的输入。
  例如:
  s = "3[a]2[bc]", 返回 "aaabcbc".
  s = "3[a2[c]]", 返回 "accaccacc".
  s = "2[abc]3[cd]ef", 返回 "abcabccdcdcdef".

- **课程内容安排:**
  - **限定性数据结构-栈与队列的特点**
  - **顺序栈实现(栈结构的顺序存储方式实现)**
  - **链式栈实现(栈结构的链式存储方式实现)**
  - **栈与递归**
  - **采用递归算法解决问题条件**

二. 课程内容笔记

4.1 采用递归算法解决的问题

什么是递归? 若在一个函数,过程或数据结构定义的内部又直接(或间接)出现定义本身的应用; 则称为他们是递归的. 或者是递归定义.
在下面3种情况下,我们会使用到递归来解决问题;

1. 定义是递归的.

比如很多数学定义本身就是递归定义; 例如大家非常熟悉的阶乘;

```
阶乘Fact(n)
(1) 若n = 0,则返回1;
(2) 若n > 1,则返回 n*Fact(n-1);

二阶斐波拉契数列Fib(n)
(1) 若n = 1或者n = 2, 则返回1;
(2) 若n > 2,则Fib(n-1) + Fib(n-2);
```

```
long Fact(Long n){
    if(n=0) return -1;
    else return n * Fact(n-1);
}
```

```
long Fib(Long n){

    if(n == 1 || n == 2) return 1;
    else return Fib(n-1)+Fib(n-2);
}
```

对于类似这种复杂问题,若能够分解成几个简单且解法相同或类似的子问题,来求解,便称为递归求解.

例如,在求解4!时先求解3!,然后再进一步分解进行求解, 这种求解方式叫做"分治法".

采取"分治法"进行递归求解的问题需满足以下三个条件:

- 能将一个问题转换变成一个小问题,而新问题和原问题解法相同或类同. 不同的仅仅是处理的对象, 并且这些处理更小且变化有规律的.
- 可以通过上述转换而使得问题简化
- 必须有一个明确的递归出口, 或称为递归边界.

```
void p(参数表){
    if(递归结束条件成立) 可直接求解; //递归终止条件
    else p(较小的参数);              //递归步骤
```

```
4 }
```

2. 数据结构是递归的.

其数据结构本身具有递归的特性.
例如,对于链表,其结点LNode的定义由数据域data 和指针域next 组成,而指针域next是一种指向LNode类型的指针,即LNode的定义中又用到了其自身. 所以链表是一种递归的数据结构;

```
1 void TraverseList(LinkList p){
2 
3     //递归终止
4     if(p == NULL) return;
5     else{
6         //输出当前结点的数据域
7         printf("%d",p->data);
8         //p指向后继结点继续递归
9         TraverseList(p->next);
10    }
11 }
```

总结, 在递归算法中, 如果当递归结束条件成立,只执行return 操作时,分治法求解递归问题算法一般形式可以简化为:

```
1 void p(参数表){
2     if(递归结束条件不成立)
3         p(较小参数);
4 }
```

```
1 void TraverseList(LinkList p){
2 
3     if(p){
4        printf("%d",p->data);
5        TraverseList(p->next);
6     }
7 
8 }
```

3.问题的解法是递归的

有一类问题,虽然问题本身并没有明显的递归结构,但是采样递归求解比迭代求解更简单, 如Hanoi塔问题,八皇后问题,迷宫问题.

4.2 递归过程与递归工作栈

一个递归函数,在函数的执行过程中,需要多次进行自我调用. 那么思考一下,一共递归函数是如何执行的?

在了解递归函数是如何执行之前,**先来了解一下任何的2个函数之间调用是如何进行的;**

在高级语言的程序中,调用函数和被调用的函数之间的链接与信息交换都是通过栈来进行的.

通常,当在一个函数的运行期间调用另一个函数时, 在运行被调用函数之前, 系统需要先完成3件事情:

1. 将所有的实参,返回地址等信息调用传递被调用函数保存;
2. 为被调用函数的局部变量分配存储空间
3. 将控制转移到被调函数入口;

而从被调用函数返回调用函数之前,系统同样需要完成3件事:

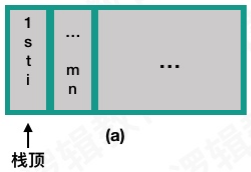
1. 保存被调用函数的计算结果;
2. 释放被调用函数的数据区
3. 依照被调用函数保存的返回地址将控制移动到调用函数.

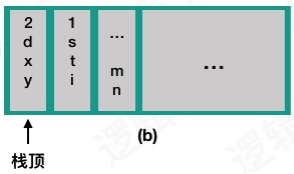
当多个函数构成嵌套调用时, 按照"先调用后返回"的原则, 上述函数之间的信息传递和控制转移必须通过"栈"来实现. 即系统将整个程序运行时的所需要的数据空间都安排在一个栈中, 每当调用一个函数时,就在它的栈顶分配一个存储区. 每当这个函数退出时,就释放它的存储区.则当前运行时的函数的数据区必在栈顶.

```
int second(int d){
    int x,y;
    //...

}

int first(int s,int t){
```

```
    int i;
    //...
    second(i)
    //2.入栈
    //...
}

void main(){

    int m,n;
    first(m,n);
    //1.入栈
    //...

}
```

在主函数main中调用函数first, 而在函数first 又嵌套调用了second 函数. 则,当我们执行当前在执行的firt函数时,则栈空间里保存了这些信息; 而当我们执行second则图(b)保存了这些信息.





一个递归函数的运行过程类似多个函数嵌套调用; 只是调用函数和被调用函数是同一个函数. 因此, 和每次调用相关的一个重要概念是递归函数运行的"层次". 假设调用该递归函数的主函数为第0层, 则从主函数调用递归函数进入第1层, 从第i层递归调用本函数为进入下一层.即第i+1层. 反正退出第i层递归应返回上一层, 即第i-1层.

为了保证递归函数正确执行, 系统需要设立一个"递归工作栈"作为整个递归函数运行期间使用的数据存储区. 每一层递归所需信息构成一个工作记录, 其中包括所有的实参,所有的局部变量以及上一层的返回地址. 每进入一层递归, 就产生一个新的工作记录压入栈顶. 每退出一个递归,就从栈顶弹出一个工作记录, 则当前执行层的工作记录必须是递归工作栈栈顶的工作记录, 称为"活动记录".