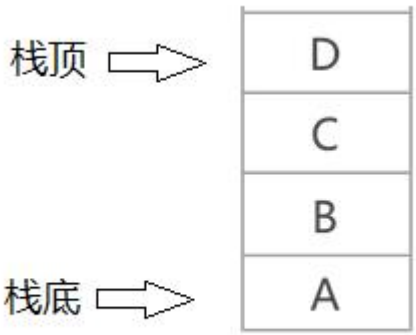
栈（FILO）

概论

基本概念

只能从线性表的一端进行插入和删除元素操作的数据结构即为栈。



应用场景

根据概念我们就可以得出，凡是只在数据结构一端进行插入删除的，并且不需要支持随机存取的场景就可以采用栈结构。

下面列举一些栈的使用场景：

- C / C++ 函数调用是借助 **栈帧** 结构实现的，栈帧即栈结构（有兴趣的读者可以看看这个具体是怎么实现的）
- 编译过程中的语法检查（同上）
- ……

分类

按照代码实现，可以分为两类：**顺序栈** 和 **链式栈**。前者是基于顺序表实现的，后者是基于链表实现的。

方法

> 以下方法用链式栈实现，可以实现动态增长

定义节点

每个节点都有两个域：**数值域** 和 **指针域**。

```
typedef int DataType;
typedef struct StackNode{
    DataType _elem;
    struct StackNode* _next;
} Stack;
```

压栈

这里采用链式栈的头插操作，因为时间复杂度为O(1)，如果采用尾插，时间复杂度为O(n)。

步骤：

- 检测参数的合法性
- 定义一个指针指向新分配的节点空间
- 填充新节点的数值域
- 让新节点的指针域指向旧的栈顶指针
- 让新的栈顶指针指向新节点

```
void Push(Stack **stack, DataType elem){
    assert(stack);
    Stack *tmp = (Stack*)malloc(sizeof(Stack));
    tmp->_elem = elem;
    tmp->_next = *stack;
    *stack = tmp;
}
```

注意：第一个参数是一个二级指针，原因是需要在函数内部更改外部指针的指向。指针也是一个变量，如果需要在函数内部修改外部变量的值，参数就要传递外部变量的地址，一级指针的地址就是二级指针。

出栈

这里采用链式栈的头删操作，因为时间复杂度为O(1)，如果采用尾删，时间复杂度为O(n)。

步骤：

- 检测参数的合法性
- 如果栈头指针为NULL，则说明栈为空，函数直接返回
- 否则定义一个指针指向当前的栈头
- 让当前的栈顶指针指向当前栈的倒数第二个元素
- 释放刚才取下的栈顶节点空间（即刚才定义新定义的指针指向的空间）

```
void Pop(Stack **stack){
    assert(stack);
    if(NULL == *stack){
        return;
    }
    Stack *tmp = *stack;
    *stack = (*stack)->_next;
    free(tmp);
}
```

获取栈顶

```
Stack* Top(Stack* stack){
    return stack; //如果栈为空，则返回NULL
}
```