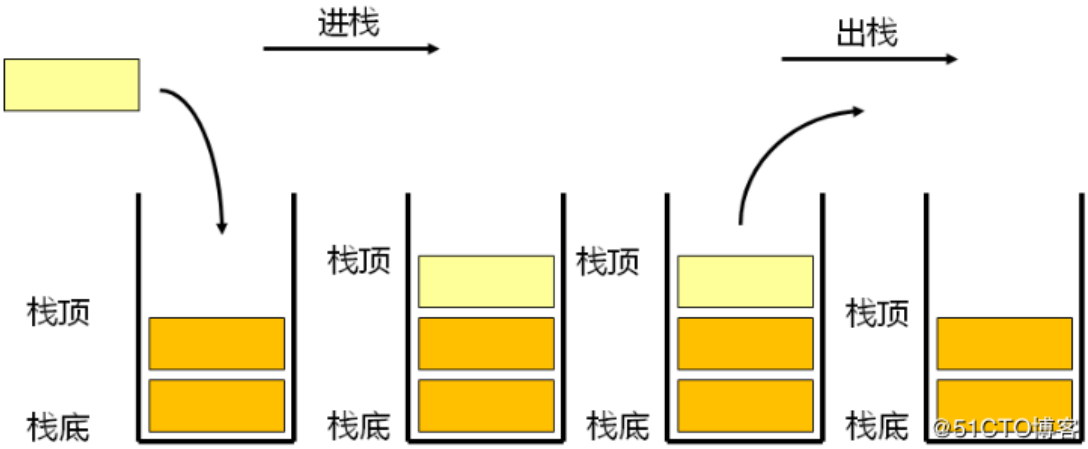
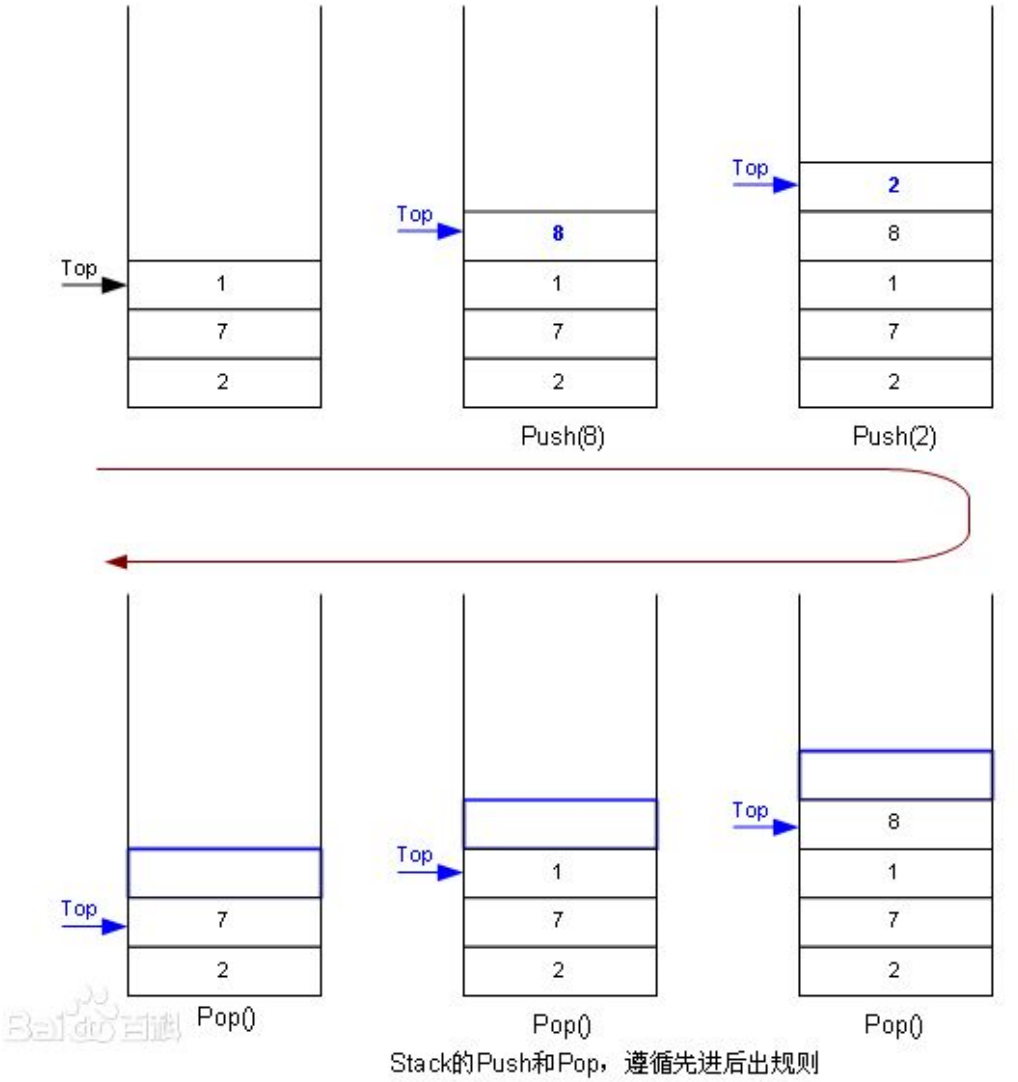
1.栈的表示和实现

> 1.1栈的概念及结构（弹夹）
>
> 栈：一种特殊的线性表，其只允许在固定的一端进行插入和删除元素操作。进行数据插入和删除操作的一端称为栈顶，另一端称为栈底。栈中的数据元素遵守后进先出LIFO（Last In First Out）
> 的原则。
> 压栈：栈的插入操作叫做进栈/压栈/入栈，入数据在栈顶。
> 出栈：栈的删除操作叫做出栈。出数据也在栈顶。

– 后进先出（Last In First Out）





Stack的Push和Pop，遵循先进后出规则

相关概念选择题

3.一个栈的初始状态为空。现将元素1、2、3、4、5、A、B、C、D、E依次入栈，然后再依次出栈，则

元素出栈的顺序是（　　）。

A 12345ABCDE

B EDCBA54321

C ABCDE12345

D 54321EDCBA

4.若进栈序列为 1,2,3,4 ，进栈过程中可以出栈，则下列不可能的一个出栈序列是（）

A 1,4,3,2

B 2,3,4,1

C 3,1,4,2

D 3,4,2,1

1.2栈的实现

栈的实现一般可以使用**数组或者链表实现**，相对而言数组的结构实现更优一些。因为数组在尾上插入数据的代价比较小。

数组实现栈时：

1、头作栈底，尾作栈顶 尾插入栈尾删出栈，时间复杂度为O(1)，适合

2、头做栈顶，尾作栈底 头插入栈头删出栈，时间复杂度为O(N)，不适合

单链表实现栈：

1、头作栈底，尾作栈顶 尾插入栈尾删出栈，时间复杂度为O(N)，不适合

2、头做栈顶，尾作栈底 头插入栈头删出栈，时间复杂度为O(1)，适合

双向链表实现栈：

1、头作栈底，尾作栈顶 尾插入栈尾删出栈，时间复杂度为O(1)，适合

2、头做栈顶，尾作栈底 头插入栈头删出栈，时间复杂度为O(1)，适合

```c
// 下面是定长的静态栈的结构，实际中一般不实用，所以我们主要实现下面的支持动态增长的栈
typedef int STDataType;
#define N 10
typedef struct Stack
{
    STDataType _a[N];
    int _top; // 栈顶
}Stack;
// 支持动态增长的栈
typedef int STDataType;
typedef struct Stack
{
    STDataType* _a;
    int _top; // 栈顶
    int _capacity;  // 容量
}Stack;
// 初始化栈
void StackInit(Stack* ps);
// 入栈
void StackPush(Stack* ps, STDataType data);
// 出栈
void StackPop(Stack* ps);
// 获取栈顶元素
STDataType StackTop(Stack* ps);
```

```
// 获取栈中有效元素个数
int StackSize(Stack* ps);
// 检测栈是否为空，如果为空返回非零结果，如果不为空返回0
int StackEmpty(Stack* ps);
// 销毁栈
void StackDestroy(Stack* ps);
```

初始化

初始化时，top给的是0，意味着top指向的是栈顶数据的下一个（先放再++）

初始化时，top给的是-1，意味着top指向的是栈顶数据（先++再放）

```
void StackInit(ST* ps)//初始化
{
    assert(ps);
    ps->a = NULL;
    ps->top = 0;//也可以给-1
    ps->capacity = 0;
}
```

释放空间

```
void STackDestory(ST* ps)//释放空间
{
    assert(ps);
    free(ps->a);
    ps->a = NULL;
    ps->capacity = ps->top = 0;
}
```

入栈

```
void StackPush(ST* ps,STDataType x)//入栈
{
    assert(ps);
    if (ps->top == ps->capacity)//栈满
    {
```

```
        int newcapacity = ps->capacity == 0 ? 4 : ps->capacity * 2;
        STDataType* tmp = realloc(ps->a, sizeof(STDataType)*newcapacity);//a为空时，相当于malloc
        if (tmp == NULL)
        {
            printf("Realloc fail!\n");
            exit(-1);
        }
        ps->a = tmp;
        ps->capacity = newcapacity;
    }
    ps->a[ps->top] = x;
    ps->top++;
}
```

出栈

```
void StackPop(ST* ps)//出栈
{
    assert(ps);
    //assert(ps->top > 0);//判空，top不能小于0
    assert(!StackEmpty(ps));//判空
    ps->top--;

}
```

返回栈顶的值

```
STDataType StackTop(ST* ps)//返回栈顶的值
{
    assert(ps);
    //assert(ps->top > 0);//top不能小于0，删空了，就不能返回top了
    assert(!StackEmpty(ps));//判空
    return ps->a[ps->top - 1];//top指向的是栈顶元素的下一个，因此-1
}
```

计算栈大小

```c
int StackSize(ST* ps)//计算栈大小
{
    assert(ps);
    return ps->top;
}
```

判空

```c
bool StackEmpty(ST* ps)//判空
{
    assert(ps);
    return ps->top == 0;//等于0就是true，不等于0就是false
}
```

OJ题

1. 括号匹配问题。

给定一个只包括“（”，“）”，“{”，“}”，“[”，“]”的字符串 s，判断字符串是否有效。

有效字符串需满足:

1.左括号必须用相同类型的右括号闭合

2.左括号必须以正确的顺序闭合。

1.左括号，入栈

2.右括号，出栈匹配跟右括号匹配

```c
//1. 括号匹配问题。
bool isValid(char* s)
{
    ST st;
    StackInit(&st);
    while (*s)//'\0'结束
    {
        if ((*s == '(')
        ||( *s == '{')
        || (*s == '['))
        {
```

```
            StackPush(&st, *s);//左括号入栈
            ++s;
        }
        else
        {
            //没有左括号栈空，只有一个右括号,不匹配
            if (StackEmpty(&st))
            {
                STackDestory(&st);//防止内存泄漏
                return false;
            }
            STDataType top = StackTop(&st);
            StackPop(&st);//右括号出栈
            if ((*s == '}' && top != '{')
            || (*s == ']' && top != '[')
            || (*s == ')' && top != '('))
            {
                STackDestory(&st);//防止内存泄漏
                return false;//不匹配
            }
            else
            {
                ++s;//匹配，继续匹配
            }
        }

    }
    //如果栈不是空，说明栈中还有左括号未出
    //没有匹配，返回时false
    bool ret = StackEmpty(&st);//没有匹配，但已经栈空
    STackDestory(&st);
    return true;
}
```