数据结构——栈和队列

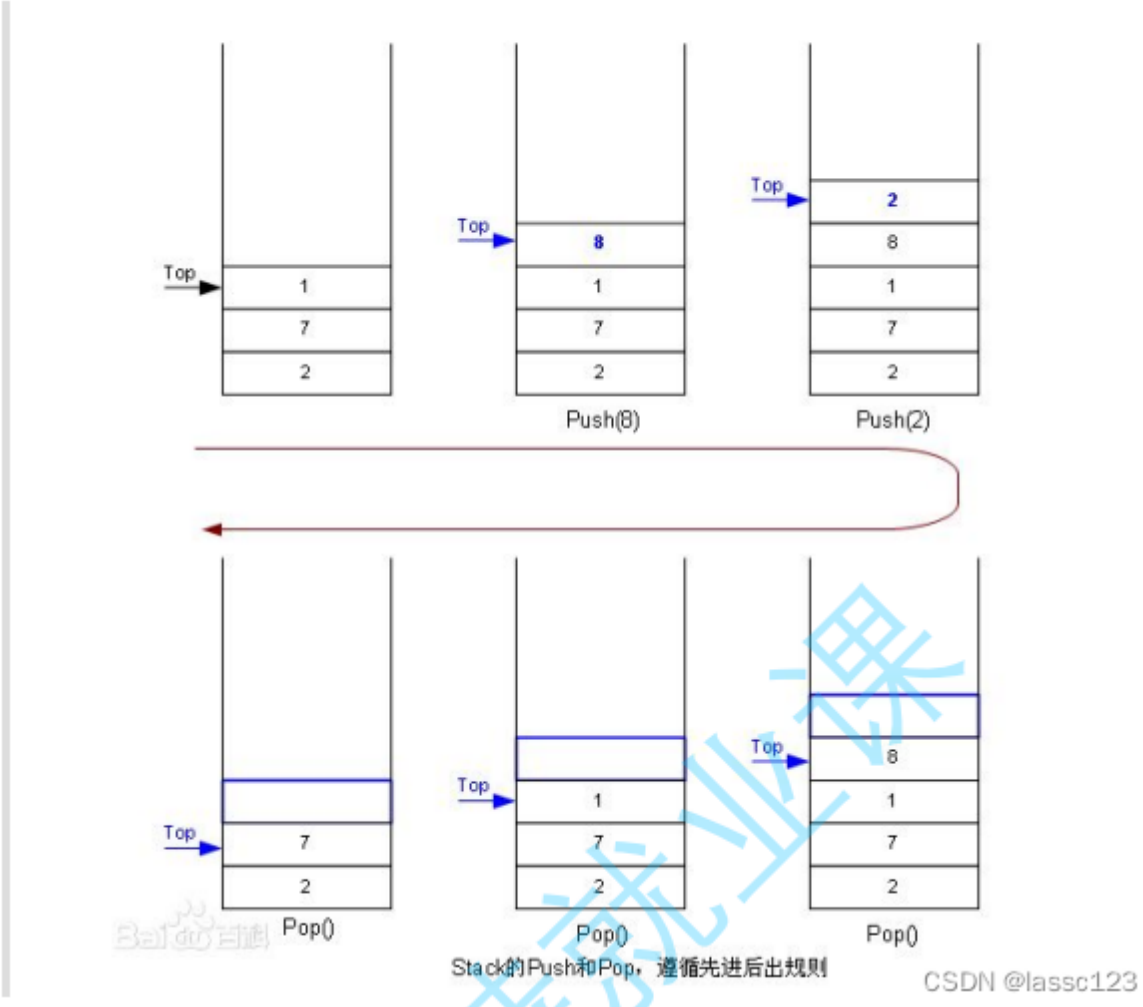
1.栈

1.1栈的概念及结构

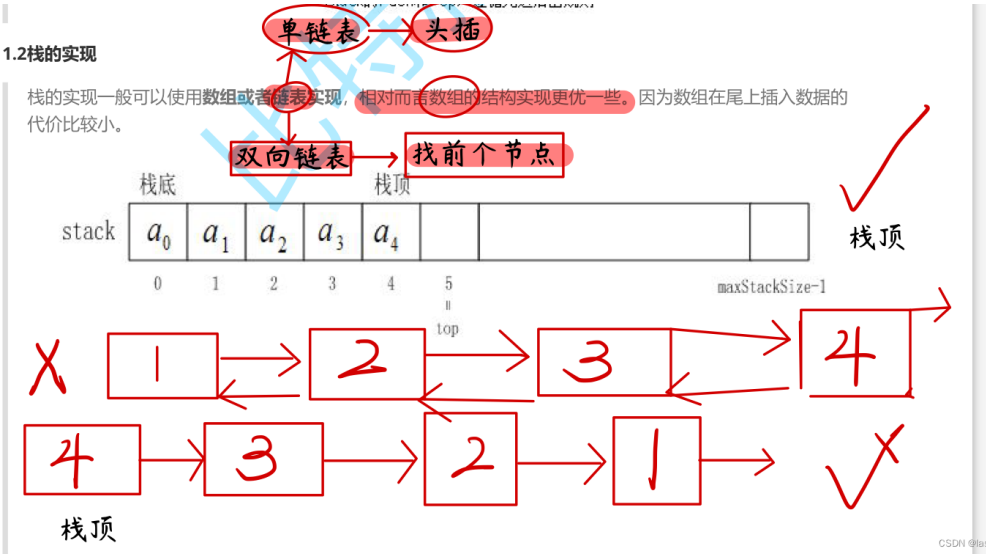
栈：一种特殊的线性表，其只允许在固定的一端进行插入和删除元素操作。**进行数据插入和删除操作的一端 称为栈顶，另一端称为栈底。**栈中的数据元素遵守后进先出LIFO（Last In First Out）的原则。

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。**（ps：栈的结构类似于羽毛球和弹匣）



1.2栈的实现



数组/单向链表（ps:使用头插法）/和双向链表都可以实现栈，所以要通过分析找到一个最优选择。

1.双向链表虽然可以实现，但是其结构复杂，效率较低，pass掉!

2.单向链表与数组其实都很不错，我一般选择数组，因为相比之下数组更加简单。

```
//Stack.h
#pragma once
#include<stdio.h>
#include<stdlib.h>
#include<assert.h>
#include<stdbool.h>
//栈结构
typedef int STDataType;
typedef struct Stack
{
    STDataType* a;
    int size;
    int top;
    int capacity;
}ST;
//栈的初始化 栈的销毁
void StackInit(ST* s);
void StackDestroy(ST* s);
//入栈 出栈
void StackPush(ST* s, STDataType x);
void StackPop(ST* s);
//栈的取顶和判空
STDataType StackTop(ST* s);
bool StackEmpty(ST* s);
```

```
//Stack.c
#define _CRT_SECURE_NO_WARNINGS 1
#include"Stack.h"
```

```c
//栈的初始化 栈的销毁
void StackInit(ST* s)
{
    s->a = NULL;
    s->top = -1;
    s->size = s->capacity = 0;
};
void StackDestroy(ST* s)
{
    s->size = s->capacity = 0;
    s->top = -1;
    free(s->a);
    s->a = NULL;
};
//入栈 出栈
void StackPush(ST* s, STDataType x)
{
    //检查容量是否满了
    if (s->capacity==s->size )
    {
        int newcapacity = s->capacity == 0 ? 4 : s->capacity * 2;
        STDataType* tmp = (STDataType*)realloc(s->a, newcapacity *
sizeof(STDataType));
        if (tmp == NULL)
        {
            perror("realloc failed");
            return;
        }
        s->a = tmp;
        s->capacity=newcapacity;
    }
    s->top++;
    s->a[s->top] = x;
    s->size++;

};
void StackPop(ST* s)
{
    assert(!StackEmpty(s));
    s->top--;
    s->size--;
};
//栈的取顶和判空
STDataType StackTop(ST* s) {
    assert(!StackEmpty(s));
    return s->a[s->top];

};

bool StackEmpty(ST* s)
{
    if (s->size == 0)
        return true;
    else
        return false;
};
```