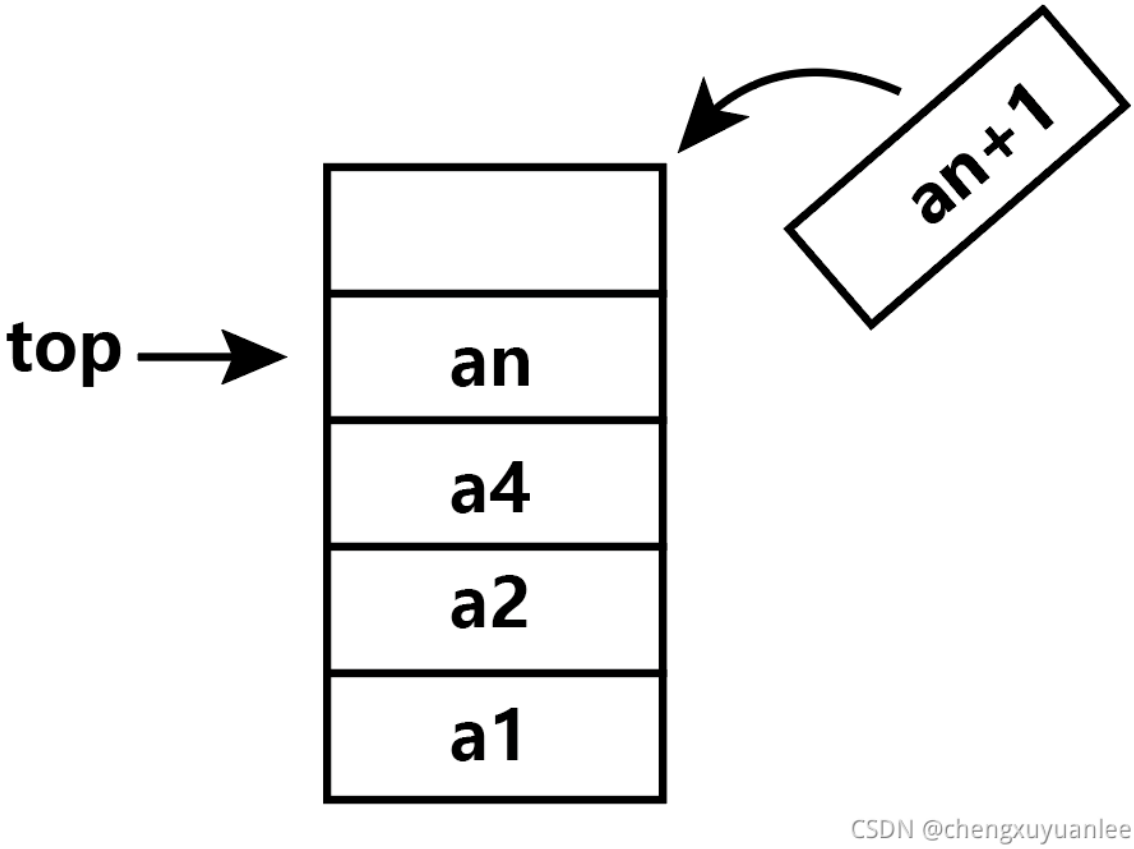
栈是一种特殊的线性表，只能由栈顶进入栈顶删除，即**后进先出。**(FIFO)类似手电筒放入电池，先放入的只能最后取出。

栈可用顺序表和链表两种方式表示即**顺序栈**和链**栈，**本文暂不实现两端栈的情况。



本文采用两种方式分别建立栈

与顺序表的**静态数组**和**动态数组**原理相同，动态数组后期可**扩容**

1、将top看做**栈的指针,**实际是数组**下标**，创建静态数组，初始化**top=-1，**同时也是栈空条件

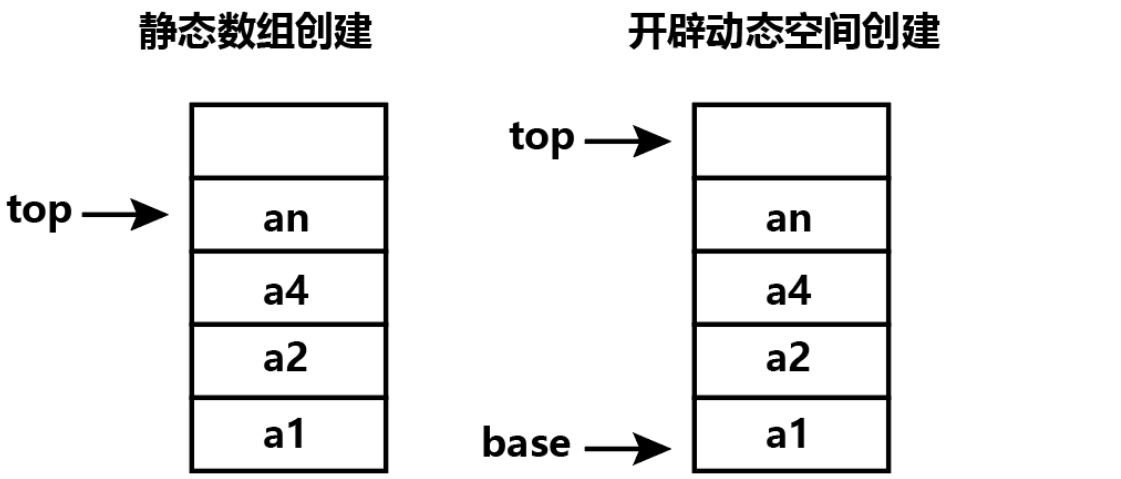
top类似顺序表中length

```
#define MAXSIZE 10            //定义最大元素个数
typedef struct stack {
    int data[MAXSIZE];        //用数组存放栈中的元素
    int top;                 //栈顶指针
}Stack;
```

2、以栈顶与栈底双指针操作，动态开辟空间，

后面为方便操作将top指针指向栈顶的后继位置

```
#define MAXSIZE 10            //定义最大元素个数
typedef struct stack{
    SElemType *top;         //栈顶指针
    SElemType *base;        //栈底指针
    int stacksize;          //栈的最大空间
}stack;
```



CSDN @chengxuyuanlee

初始化栈

静态数组方式只需要将top=-1。不过多赘述

动态开辟空间方式需先开辟一块空间后将top指针与base指针相等即可。

```
void InitStack(Stack S) {
    S.base = (Stack*)malloc(MAXSIZE*sizeof(int));//开辟空间为 MAXSIZE
    if (!S.base) {                    //确定开辟空间成功
        prinff("开辟空间失败");
    }
    else {
        S.base == S.top;
    }
}
```

入栈操作（即增加）

静态数组方式：① 判断栈空间是否满，注意下标 ② 栈顶指针+1 ③给数组赋值

注意：因为初始化top从-1开始，因此先++

```
Stack InsertStack(Stack S) {
    if (S.top==MAXSIZE-1) { //判断栈未满，下标为MAXSIZE-1
        printf("栈空间已满");
    }else{
        S.top++;
        S.data[0] = 1;    //加入数值
    }
    return S;
}
```

动态空间方式：① 判断栈空间是否满，，注意判断条件 ② 赋值 ③栈顶指针+1

注意：由图2可知，要保证S.top始终指向空位置，并且S.top-S.base 值小于MAXSIZE

```
Stack InsertStack(Stack S,int e) {
    if (S.top - S.base == MAXSIZE) { //判断栈空间是否满
        printf("栈空间满");
    }
    {
        *S.top = e;            //栈顶值为e
        S.top++;              //栈顶指针+1  等价于*S.top++=e;
        return   S;
    }
```

出栈操作（即删除）

静态数组方式：① 判断栈空间是否空 ，注意下标 ② 栈顶指针+1 ③给数组赋值

```
Stack DeletStack(Stack S) {
    if(S.top==0){           //判断栈是否为空
        printf("栈已经空了");
    }
    else {
        int e = 0;          //用e存下要删除的元素
        e = S.data[S.top];
        S.top--;           //栈顶指针向下移动
    }
}
```

动态空间方式：① 判断栈空间是否空，注意判断条件 ② 栈顶指针移动 ③ 拷贝要删除元素给e

```
Stack DeletStack(Stack S) {
    if(S.top==S.base){    //判断栈为空
        printf("栈为空");
    }
    else {
        int e = 0;
        S.top--;
        e = *S.top;    //等价于 e==*S.top--; 先赋值再--
    }
}
```

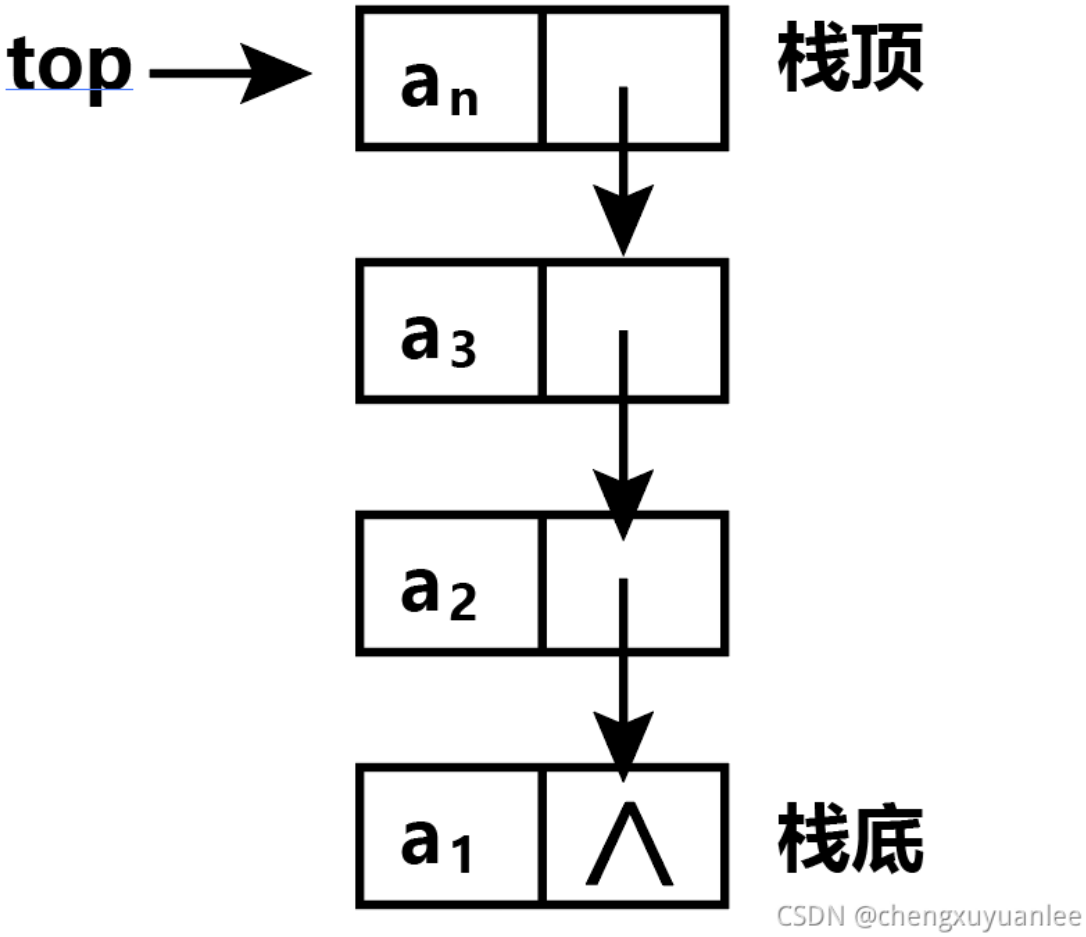
栈改/查

栈只能查/改栈顶元素，因此直接将栈顶元素返回或修改即可。

链栈

链栈即栈的存储结构以链表形式展现，与顺序栈不同地方在于空间大小可调整，指针top指向栈顶元素即可

链栈不需要考虑栈满，链栈动态存储基本来说不会满，链栈空条件为 top->next=NULL;



链栈创建代码实现

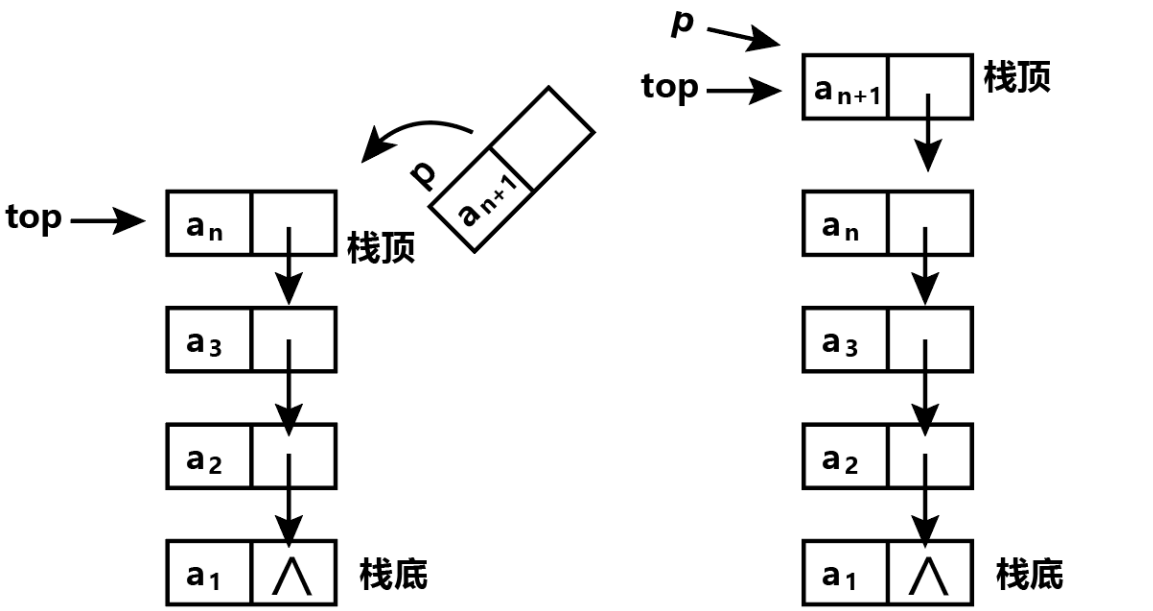
```
typedef struct stack {
    int data;                   //数据域
    struct stack *next;         //指针域
}Stack;
```

链栈代码类似链表，方便记忆，具体代码可参照插入图解，**开辟空间后需要加上判断**这里方便看代码因此没有加

```
void CreatStack(Stack* S) {
    Stack* top= (Stack*)malloc(sizeof(Stack));  //动态开辟空间,并将top指向该空间首地址
    top->next = NULL;                           //指向头结点
    for (int i = 0;i < 5;i++) {
    Stack* p= (Stack*)malloc(sizeof(Stack));
    p->data= i;                         //给数据域赋值
    p->next = top;
    top = p;
    printf("%d ",p->data);              //连接节点，并将top移动
    }
}
```

链栈入栈

如下图 步骤为： ①先开辟空间p ②将p的next域指向top ③top指向栈顶

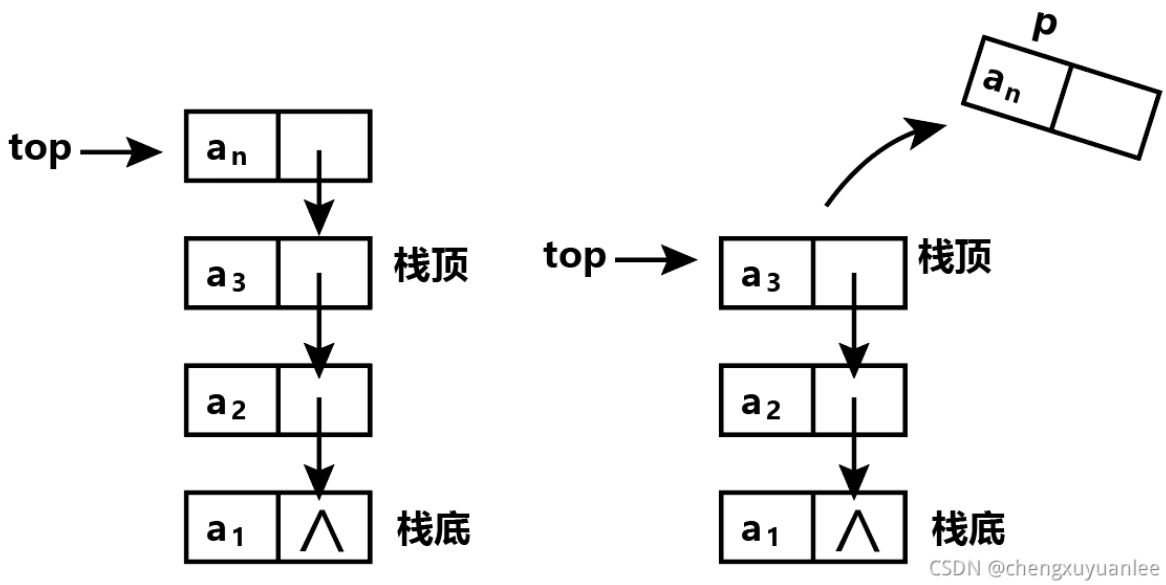


链栈入栈代码实现

```
void InsertStack(Stack* S) {
    Stack* top = S;
    Stack* p = (Stack*)malloc(sizeof(Stack));
    if(p){
        p->data= 1;             //数据域赋值
        p->next = top;          //新结点指向top
        top = p;               //top指向栈顶
    }
}
```

链栈出栈实现

如下图 步骤为： ①先用变量存下top数值 ② p指针指向an ③ top指针移动 ④ free（p）



链栈出栈代码实现

```
Stack* DeleteStack(Stack* S) {
    Stack* top = S;
    if(top){
        int x = top->data;      //数据域赋值
        Stack* p = top;         //p为删除的指针，top指向下一个数据域
        top = top->next        //top移动
        free(p);               //释放空间
    }
    return S;
}
```