



目录

CONTENTS

01

【 顺序栈的存储结构 】

02

【 构造一个空栈 】

03

【 取栈顶元素 】

04

【 入栈 】

05

【 出栈】

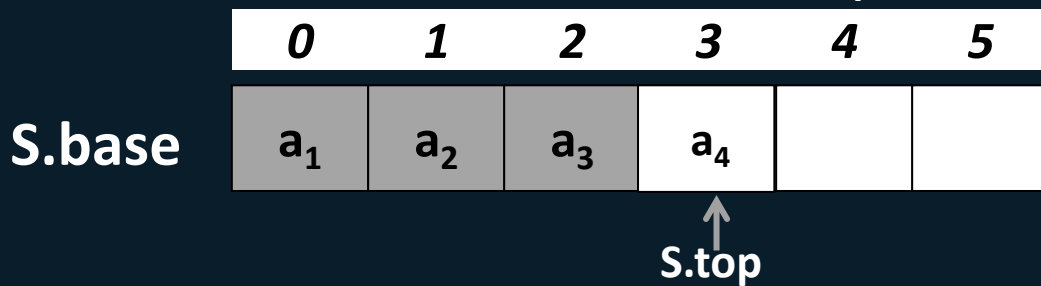
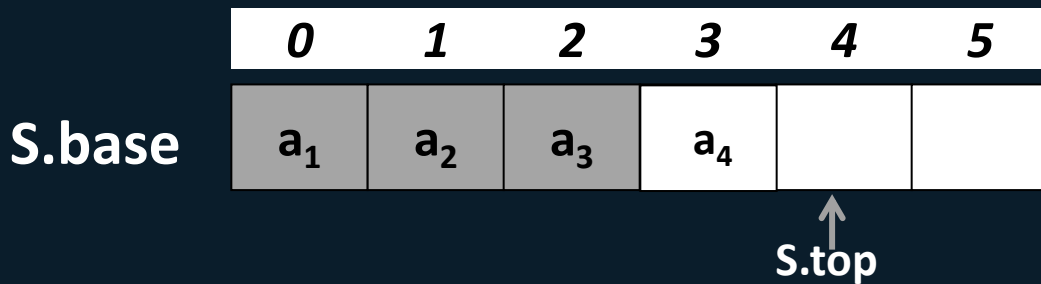
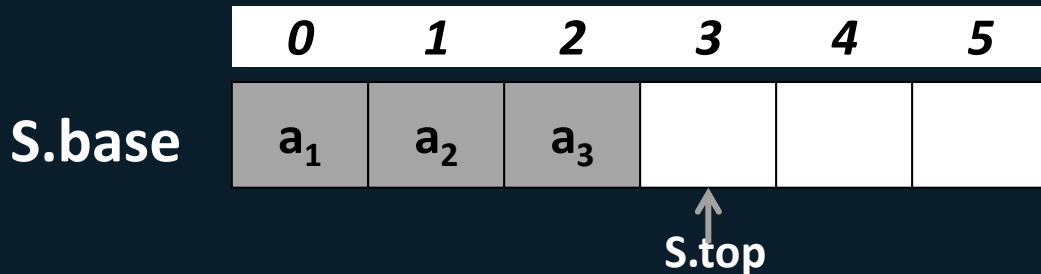


01

顺序栈的存储结构

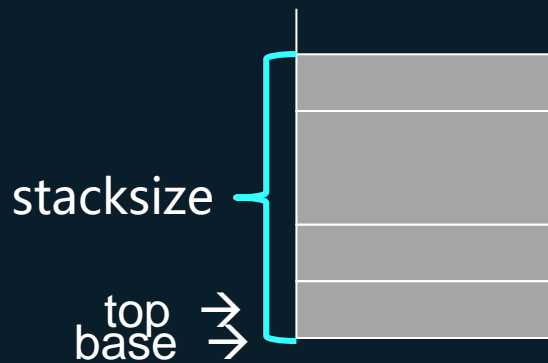
顺序栈的存储结构

利用一组地址连续的存储单元依次自栈底到栈顶存放栈的数据元素，同时附设一个整形变量top指示栈顶元素在顺序栈中的位置，数据入栈或出栈时使整形变量top分别加1或减1。



//栈的顺序存储表示

```
typedef struct {  
    SElemType *base;    //栈底指针  
    SElemType *top;     //栈顶指针  
    int stacksize;     //当前已分配的存储空间  
} SqStack;
```



栈顶指针和栈顶元素的关系



图1

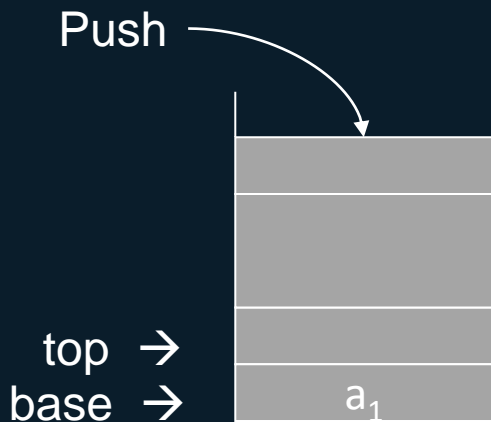


图2

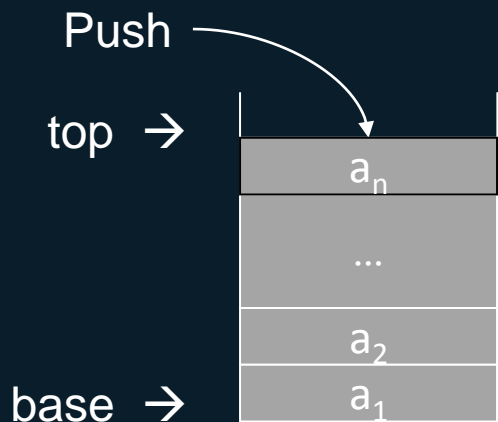


图3

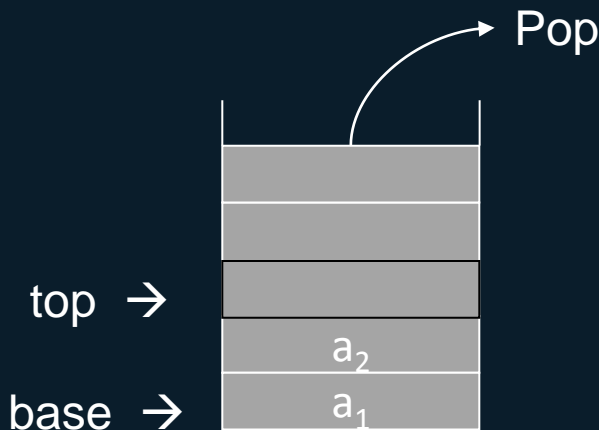


图4

非空顺序
栈中，栈
顶指针始
终在栈顶
元素的下
一个位置



02

构造一个空栈

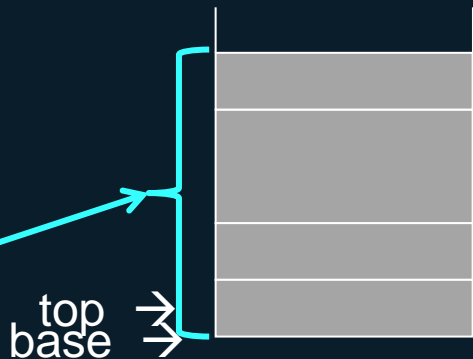
构造一个空栈

1. 问题分析

- 操作名称：InitStack (&S)
- 操作结果：构造一个空栈S

算法思想：

- 1) 申请初始分配量 (`STACK_INIT_SIZE`) 个数据元素的存储空间，
- 2) 栈顶和栈底同时指向空间开始地址，当前已分配存储空间 (`stacksize`) 即初始化大小。



构造一个空栈

2. 算法描述

//构造一个空栈s

Status InitStack (SqStack &S)

{ //申请初始化大小的存储空间

S.base = (SElemType *) malloc(STACK_INIT_SIZE)* sizeof (SElemType));

if (!S.base) exit(OVERFLOW); //存储分配失败

//设置栈顶指针的位置及栈的当前存储空间

S.top = S.base;

S.stacksize = STACK_INIT_SIZE;

return OK;

} // InitStack



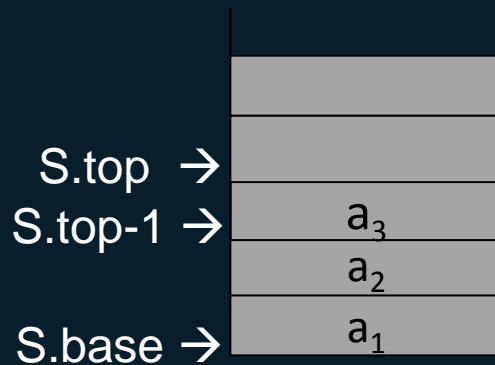
03

取栈顶元素

取栈顶元素

1. 问题分析

- 操作名称：GetTop (S, &e)
- 初始条件：栈S已存在且非空
- 操作结果：用e返回S的栈顶元素



算法思想：

- 1) 首先判断栈是否为空，若栈为空，则返回ERROR；
- 2) 若栈不空则取栈顶元素，返回OK。

以上内容仅为本文档的试下载部分，为可阅读页数的一半内容。如要下载或阅读全文，请访问：<https://d.book118.com/43714020065006051>