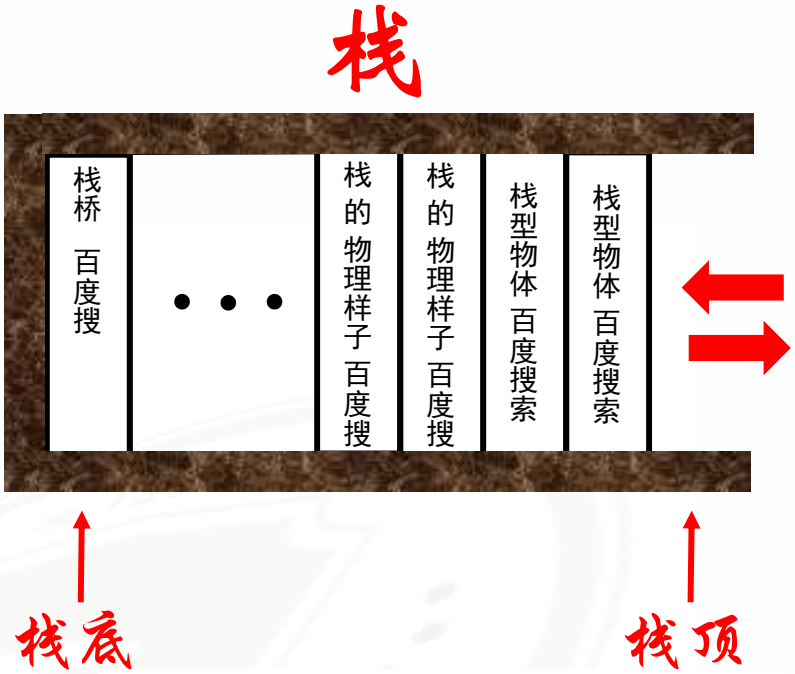
北京交通大學
BEIJING JIAOTONG UNIVERSITY

第三章 栈和队列

本节内容

- 3.1.1 栈的概念
- 3.1.2 栈的基本操作
- 3.1.3 顺序存储结构及操作
- 3.1.4 链式存储结构及操作
- 3.1.5 栈的应用

北京交通大学
BEIJING JIAOTONG UNIVERSITY



LIFO

Last In First Out

3.1 栈(Stack)

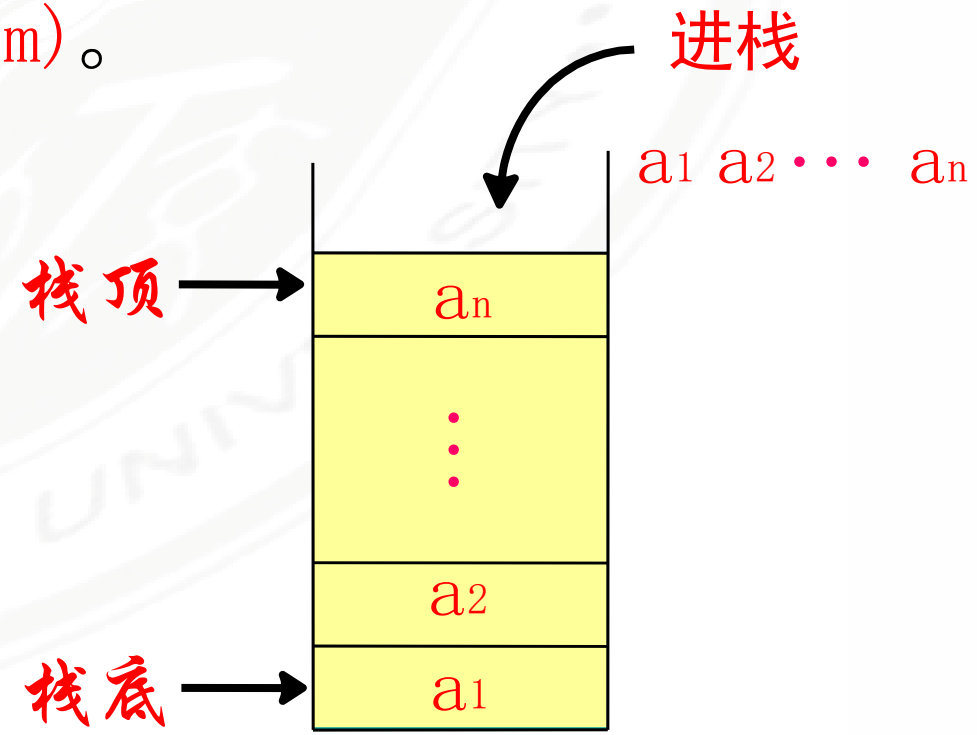
- 栈是限定在表尾进行插入和删除操作的线性表。

$$S=(a_1,a_2,\cdots a_n)$$

- 允许进行插入和删除操作的一端称为栈顶(top)，另一端称为栈底(bottom)。

a_1 --- 栈底元素

a_n --- 栈顶元素



3.1 栈(Stack)

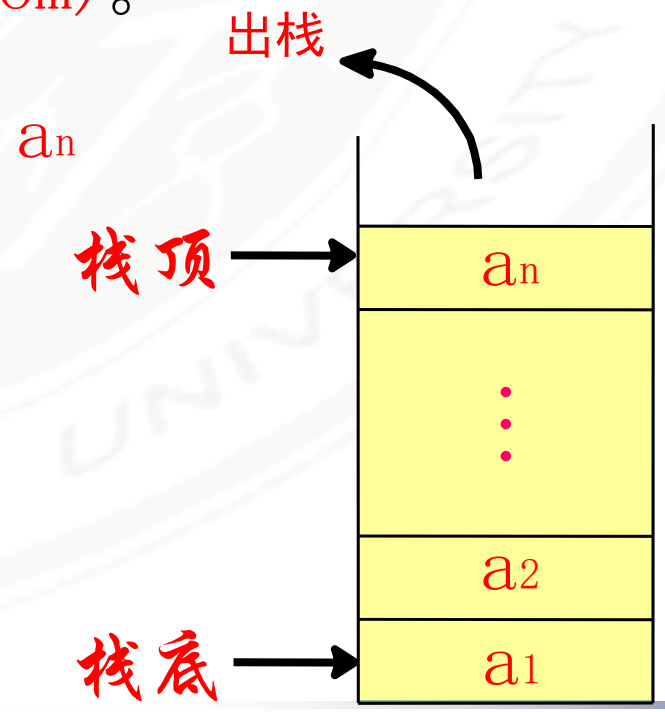
- 栈是限定在表尾进行插入和删除操作的线性表。

$$S = (a_1, a_2, \cdots a_n)$$

- 允许进行插入和删除操作的一端成为栈顶(top)，另一端称为栈底(bottom)。

a_1 --- 栈底元素

a_n --- 栈顶元素



3.1 栈(Stack)

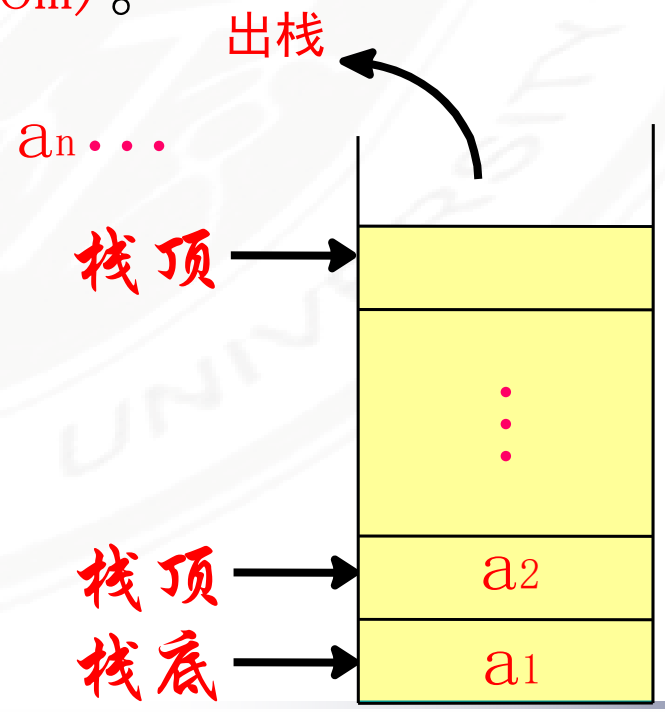
- 栈是限定在表尾进行插入和删除操作的线性表。

$$S = (a_1, a_2, \cdots a_n)$$

- 允许进行插入和删除操作的一端成为栈顶(top)，另一端称为栈底(bottom)。

a_1 --- 栈底元素

a_n --- 栈顶元素



3.1 栈(Stack)

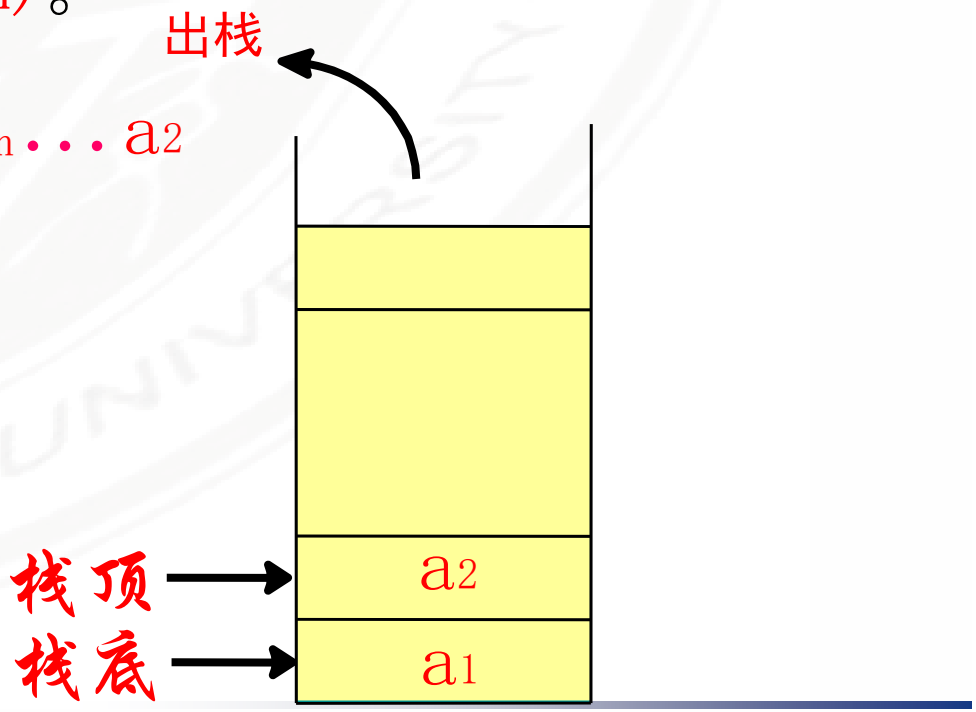
- **栈**是限定在表尾进行插入和删除操作的线性表。

$$S = (a_1, a_2, \cdots a_n)$$

- 允许进行插入和删除操作的一端成为栈顶(top)，另一端称为**栈底**(bottom)。

a_1 ---- 栈底元素

a_n ---- 栈顶元素



3.1 栈(Stack)

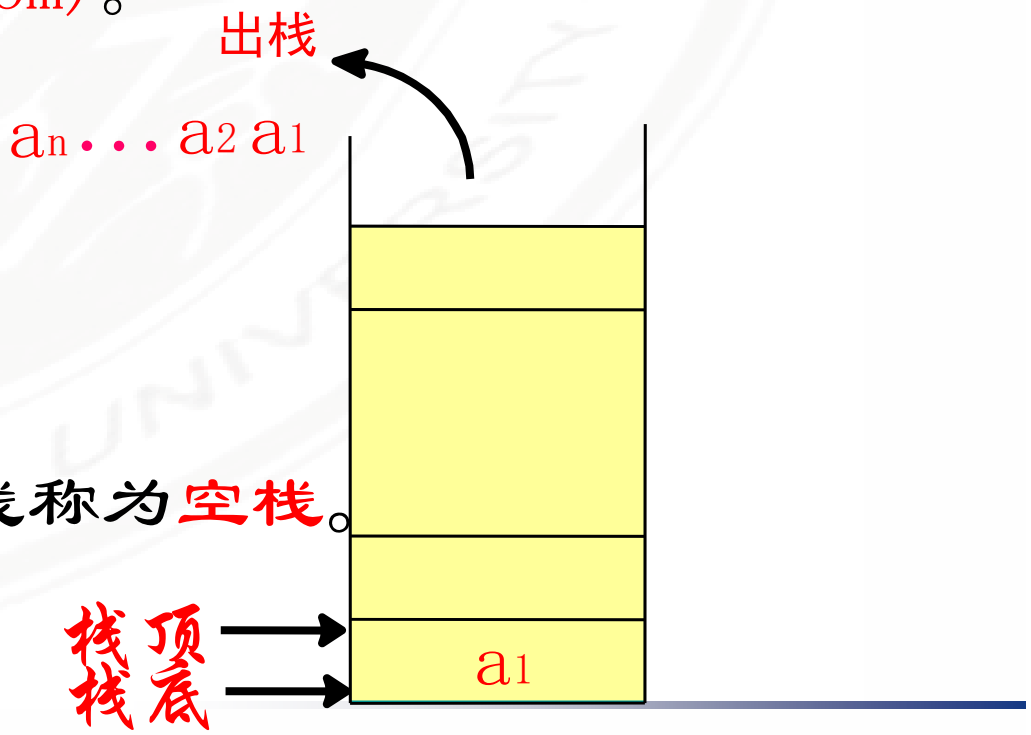
- **栈**是限定在表尾进行插入和删除操作的线性表。

$$S = (a_1, a_2, \cdots a_n)$$

- 允许进行插入和删除操作的一端成为栈顶(top)，另一端称为**栈底**(bottom)。

a_1 --- 栈底元素

a_n --- 栈顶元素



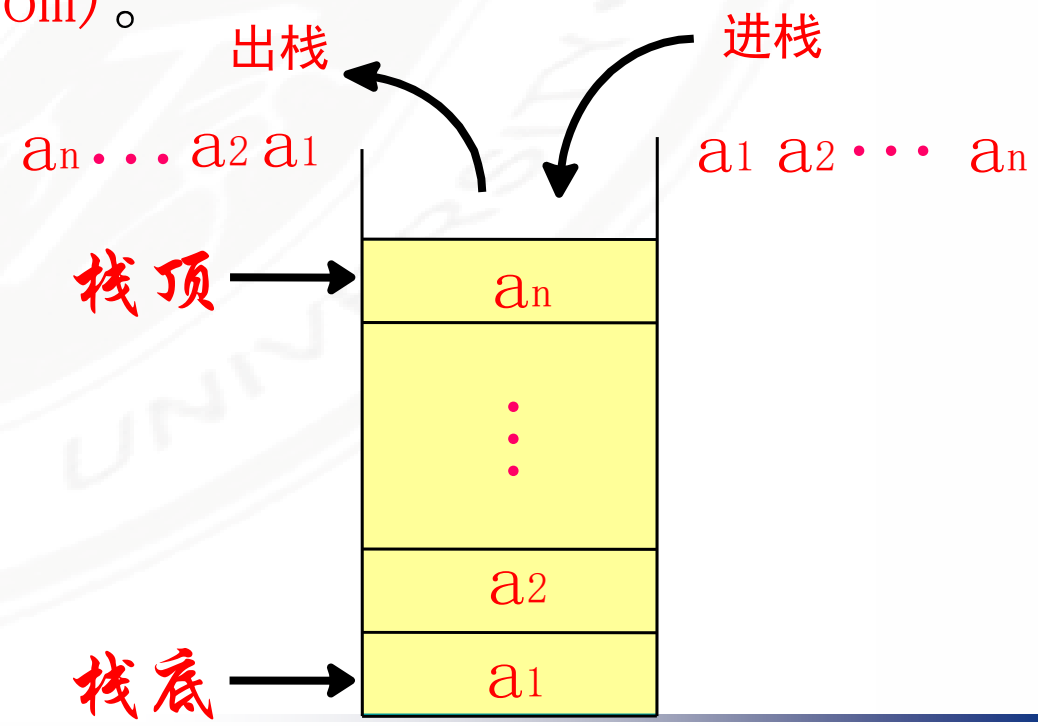
- 没有任何数据元素的栈称为**空栈**。

3.1 栈(Stack)

- 栈是限定在表尾进行插入和删除操作的线性表。

$$S = (a_1, a_2, \cdots a_n)$$

- 允许进行插入和删除操作的一端成为栈顶(top)，另一端称为栈底(bottom)。



a_1 --- 栈底元素

a_n --- 栈顶元素

LIFO

Last In First Out

3.1 栈(Stack)

理解栈的概念需要注意

首先它是一个线性表，也就是说，栈元素具有线性关系，即前驱后继关系。只不过它是一种特殊的线性表而已。定义中说是在线性表的表尾进行插入和删除操作，这里表尾是指栈顶，而不是栈底。

它的特殊之处就在于限制了这个线性表的插入和删除位置，它始终只在栈顶进行。这也就使得：栈底是固定的，最先进栈的只能在栈底。

思考题

将1, 2, 3顺序入栈，求可能的出栈序列？

1(+) → 2(+) → 3(+) → 3(-) → 2(-) → 1(-)	3,2,1
1(+) → 1(-) → 2(+) → 2(-) → 3(+) → 3(-)	1,2,3
1(+) → 1(-) → 2(+) → 3(+) → 3(-) → 2(-)	1,3,2
1(+) → 2(+) → 2(-) → 1(-) → 3(+) → 3(-)	2,1,3
1(+) → 2(+) → 2(-) → 3(+) → 3(-) → 1(-)	2,3,1

本节内容

- 3.1.1 栈的概念
- 3.1.2 栈的基本操作
- 3.1.3 顺序存储结构及操作
- 3.1.4 链式存储结构及操作
- 3.1.5 栈的应用

基本操作：

(1) InitStack(&S) //构造空栈

(2) DestroyStack(&S) //销毁栈

(3) ClearStack(&S) //清空栈

(4) StackEmpty(S) //栈判空. 空--TRUE,

(5) StackLength(S) //求长度

(6) Push(&S,e) //进栈

(7) GetTop(S,&e) //取栈顶元素,

(8) Pop(&S,&e) //出栈

(9) StackTraverse(S,visit()) //遍历

本节内容

- 3.1.1 栈的概念
- 3.1.2 栈的基本操作
- 3.1.3 顺序存储结构及操作
- 3.1.4 链式存储结构及操作
- 3.1.5 栈的应用

顺序栈的存储结构定义

我们定义一个 top 变量来指示栈顶元素在数组中的位置，这 top 就如同中学物理学过的游标卡尺的游标，如图 4-4-1，它可以来回移动，意味着栈顶的 top 可以变大变小，但无论如何游标不能超出尺的长度。同理，若存储栈的长度为 StackSize，则栈顶位置 top 必须小于 StackSize。

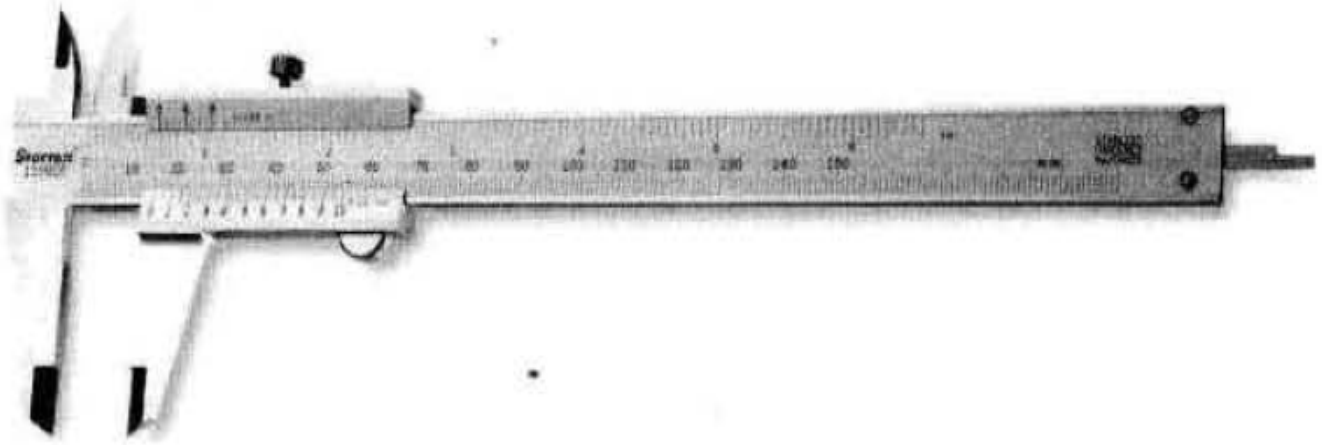


图 4-4-1

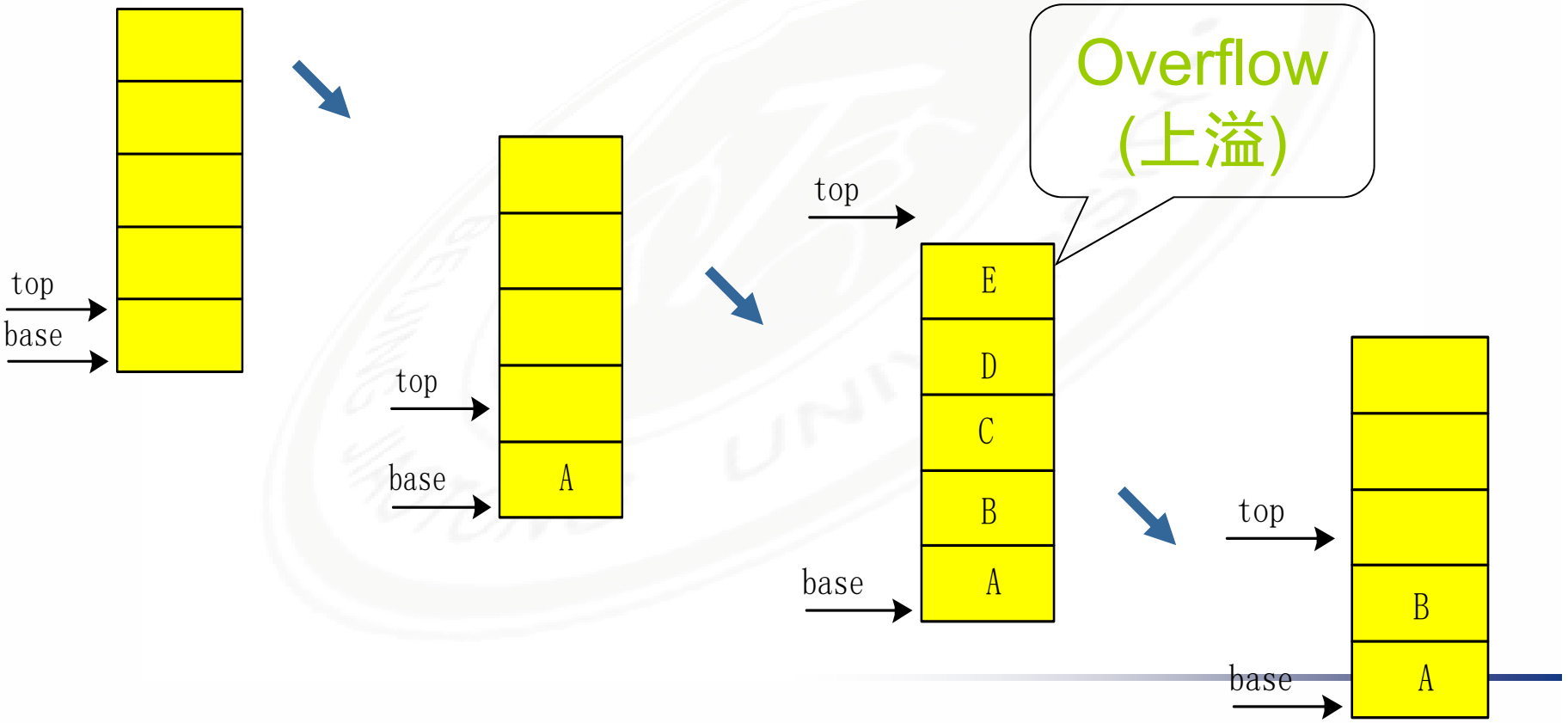
顺序栈的存储结构定义

```
#define STACK_SIZE   100    //顺序栈存储空间的分配量
SElemType   SqStack[STACK_SIZE]; //存放栈元素的一维数组
int top;   //栈顶位置
```

```
#define STACK_SIZE   100    //顺序栈存储空间的分配量
typedef struct{
   SElemType  base[STACK_SIZE];  //存放栈元素的一维数组
   int top;  //栈顶位置
}SqStack;
```

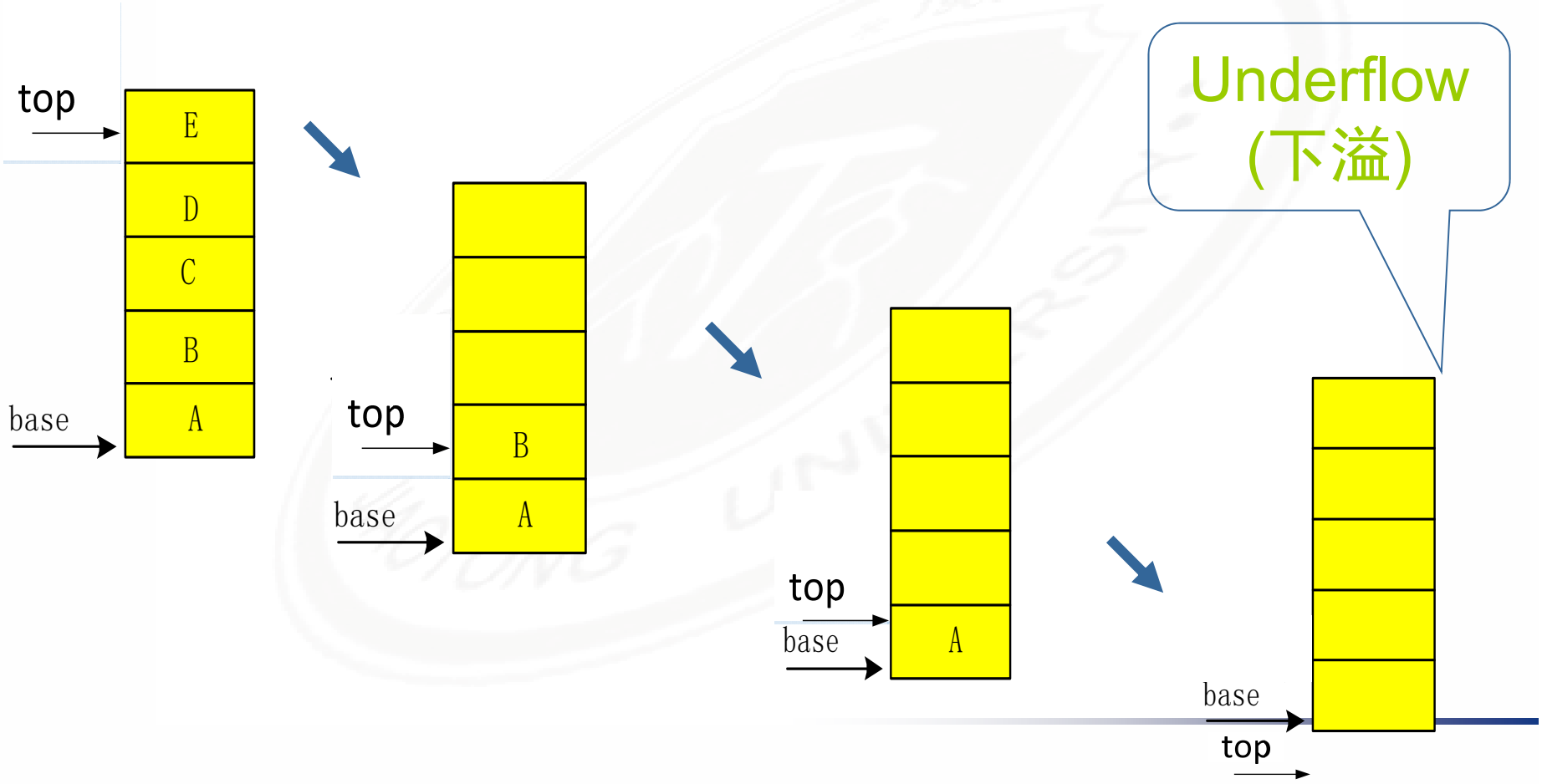
顺序栈存储结构的定义

- 设有一个顺序栈，StackSize是5，栈底位置是SElemType base[？]，空栈时栈顶位置是SElemType base[？]，满栈时栈顶位置是SElemType base[？]



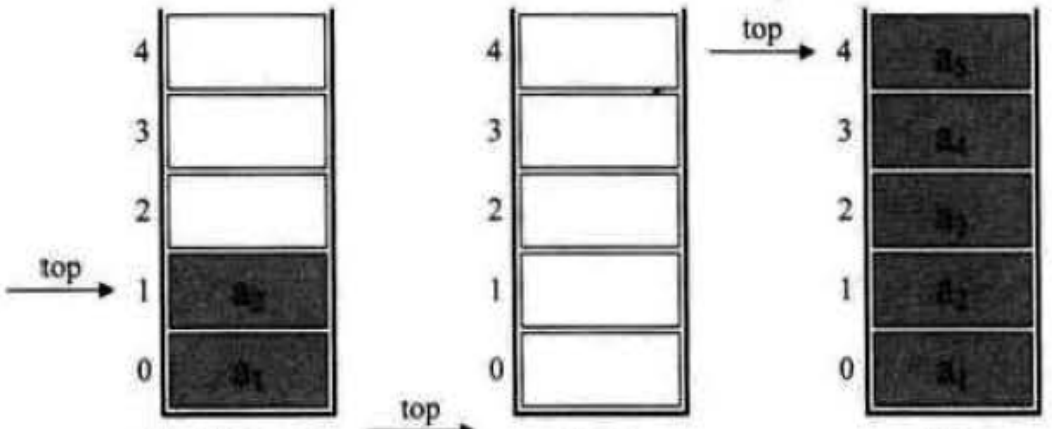
顺序栈存储结构的定义

- 设有一个顺序栈，StackSize是5，栈底位置是SElemType base[？]，空栈时占地位置是SElemType base[？]，满栈时栈顶位置是SElemType base[？]



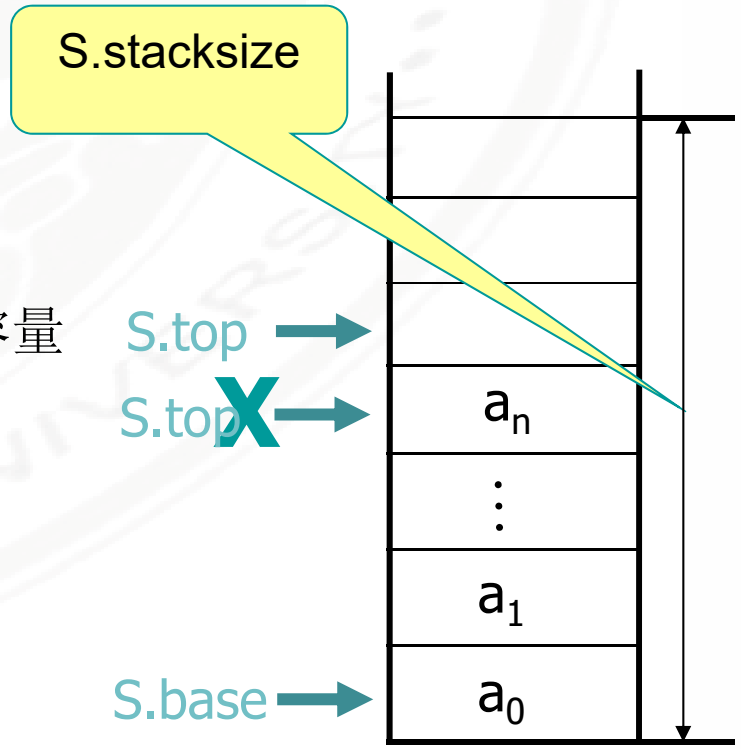
顺序栈存储结构的定义

- 在下图情况下，top分别等于几？
  - 严：栈顶指针指向栈顶元素的下一个位置
    - S.top=S.base（栈底指针）
  - 李：栈顶指针指向栈顶元素
    - 没有栈底指针，只靠S.top=-1来定义，

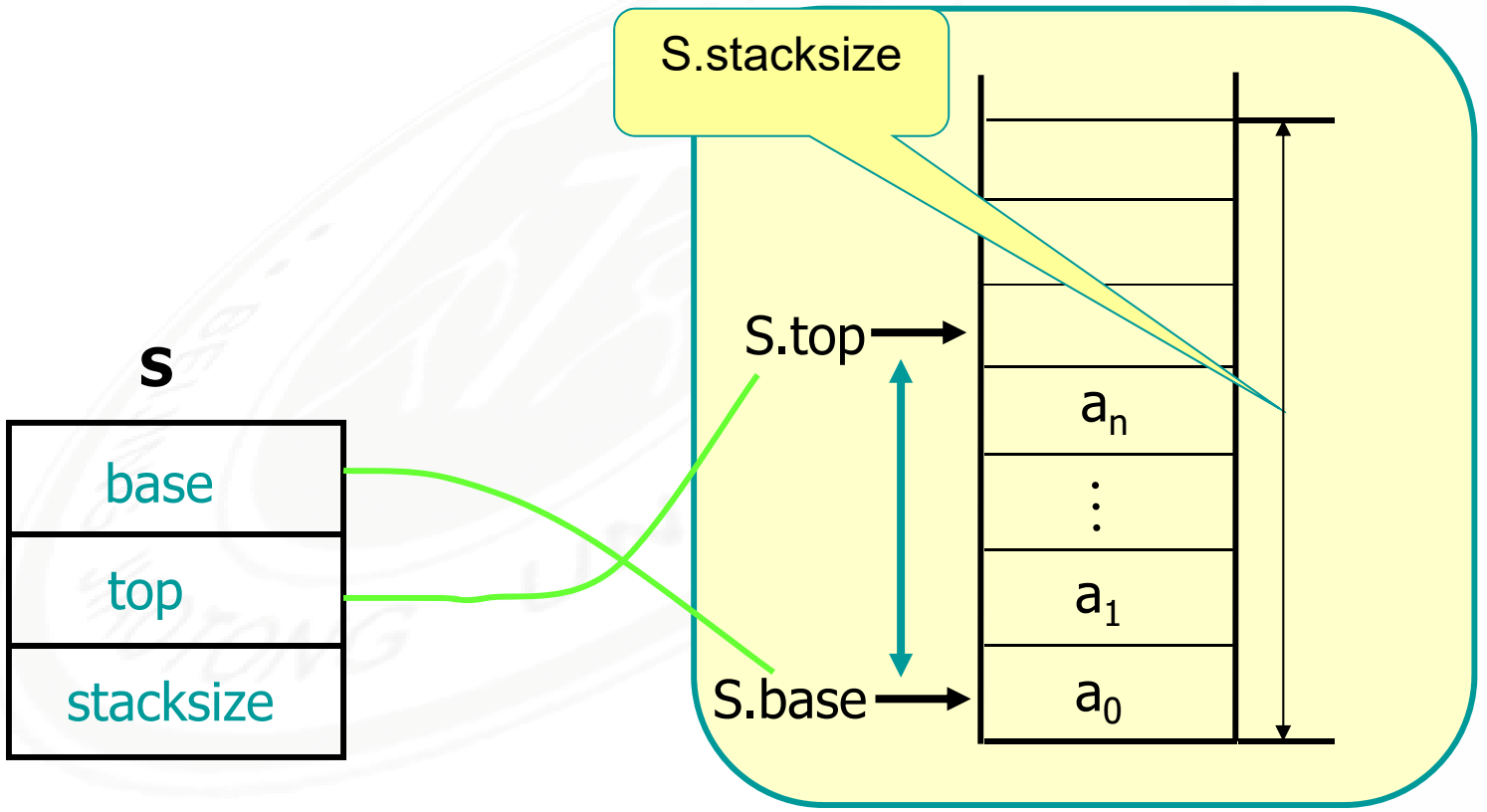


顺序栈的存储结构定义

```
#define STACK_INIT_SIZE   100        //存储空间的初始分配量
#define STACKINCREMENT  10           //存储空间的分配增量
typedef struct{
   SElemType  *base;     //栈底指针
   SElemType  *top;      //栈指针
   int        stacksize;     //当前分配的存储容量
                             //以元素为单位
}SqStack;

 SqStack  S;
```

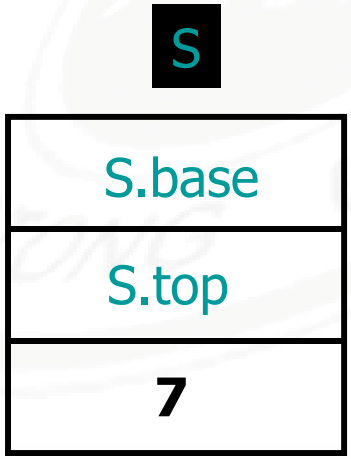


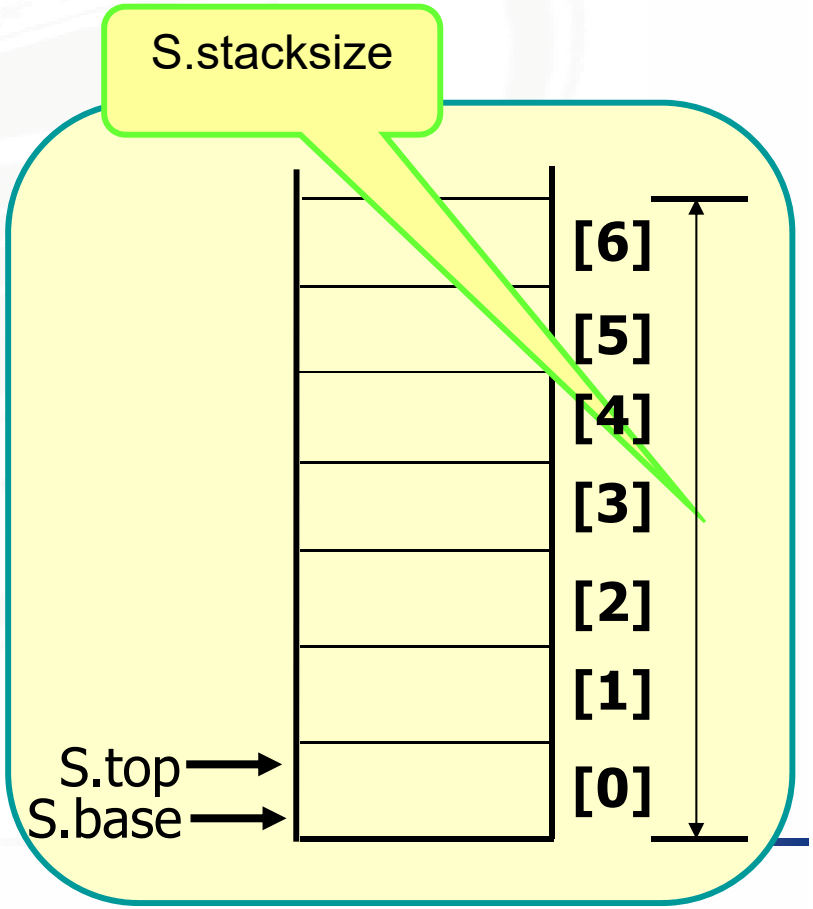
```
typedef struct{
   SElemType  *base;    //栈底指针
   SElemType  *top;     //栈指针
   int        stacksize;//当前分配的存储容量
                        //以元素为单位
}SqStack;

 SqStack  S;
```



(1) 顺序栈初始化

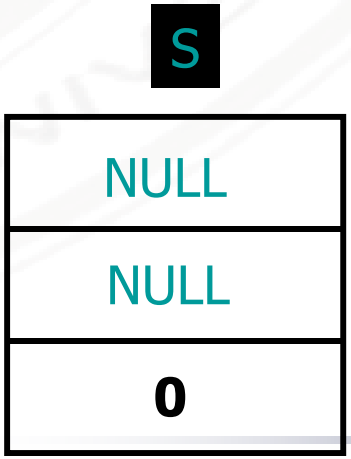
```
Status InitStack(SqStack &S){
    S.base=(SElemType *)malloc(STACK_INIT_SIZE*sizeof(SElemType));
    if(!S.base) exit(OVERFLOW);
    S.top=S.base;
    S.stacksize=STACK_INIT_SIZE;
    return OK;
}//InitStack
```





(2) 销毁顺序栈

```
Status DestroyStack (SqStack &S){
    free(S.base);
    S.base=NULL;
    S.top=NULL;
    S.stacksize=0;
    return OK:
}// DestroyStack
```



(3) 清空顺序栈

是不是要全部pop出来?

```
Status ClearStack (SqStack &S){
   S.top=S.base;
   return OK:
}// ClearStack
```

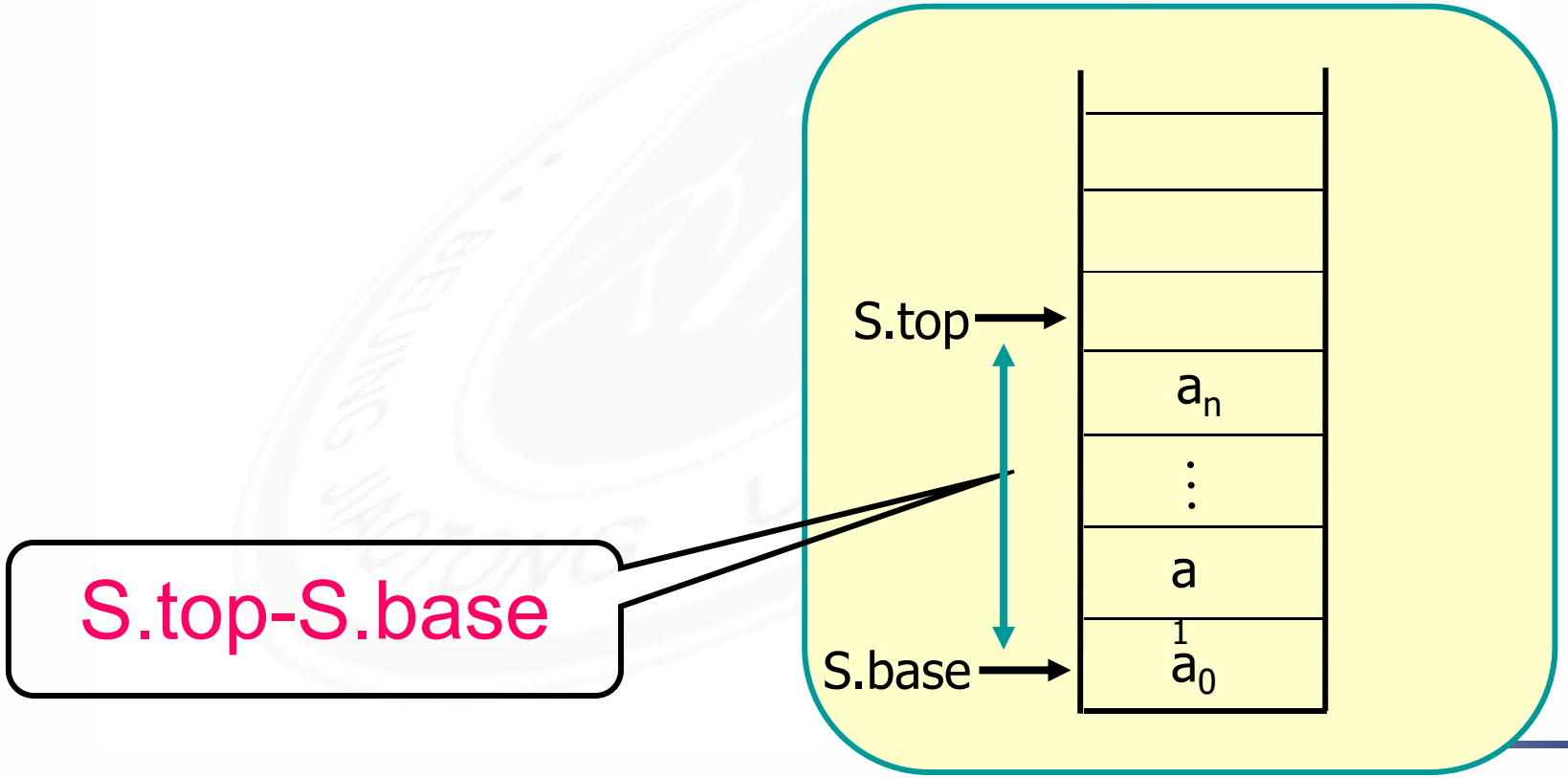
(4)顺序栈判空

栈的判空条件是什么？

```
Status StackEmpty (SqStack S){
    return (S.top==S.base);   //空栈条件
 }// StackEmpty
```

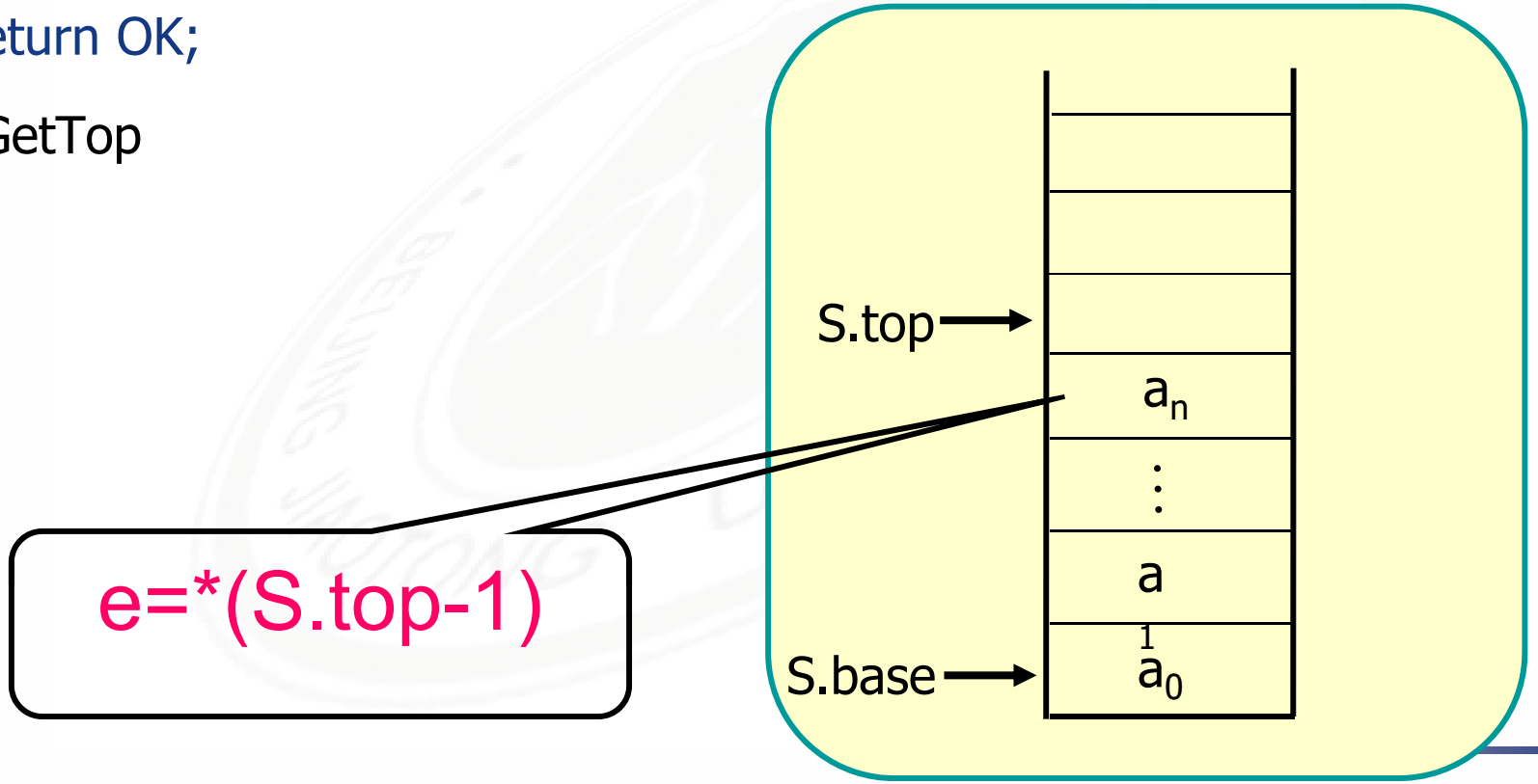
(5) 求顺序栈长度

```
int  StackLength (SqStack S){
   return (S.top-S.base);
}// StackLength
```



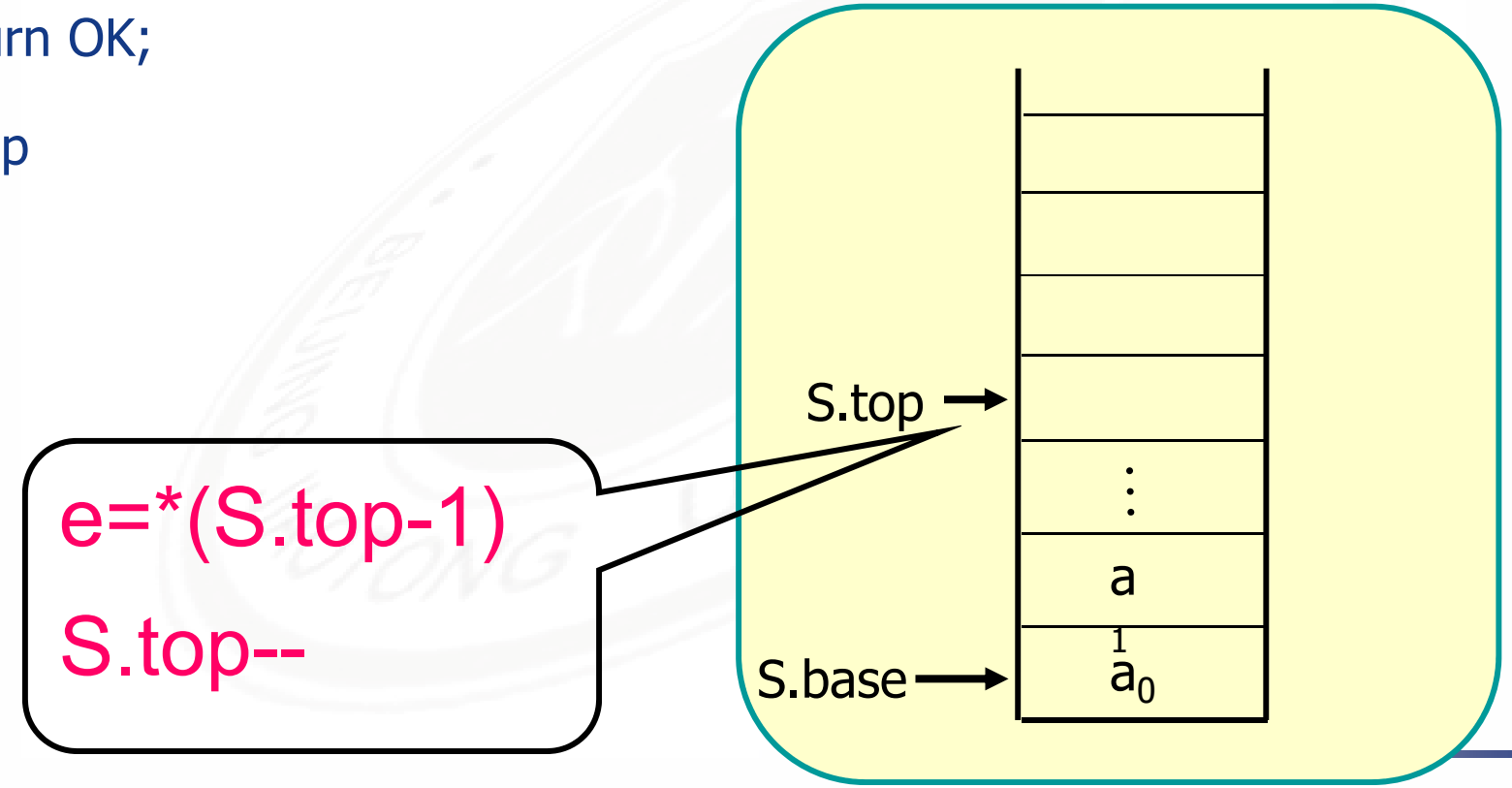
(6) 取顺序栈顶元素

```
Status GetTop(SqStack S, SElemType &e){
  if(S.top==S.base) return ERROR;
  e=*(S.top-1);
  return OK;
}//GetTop
```



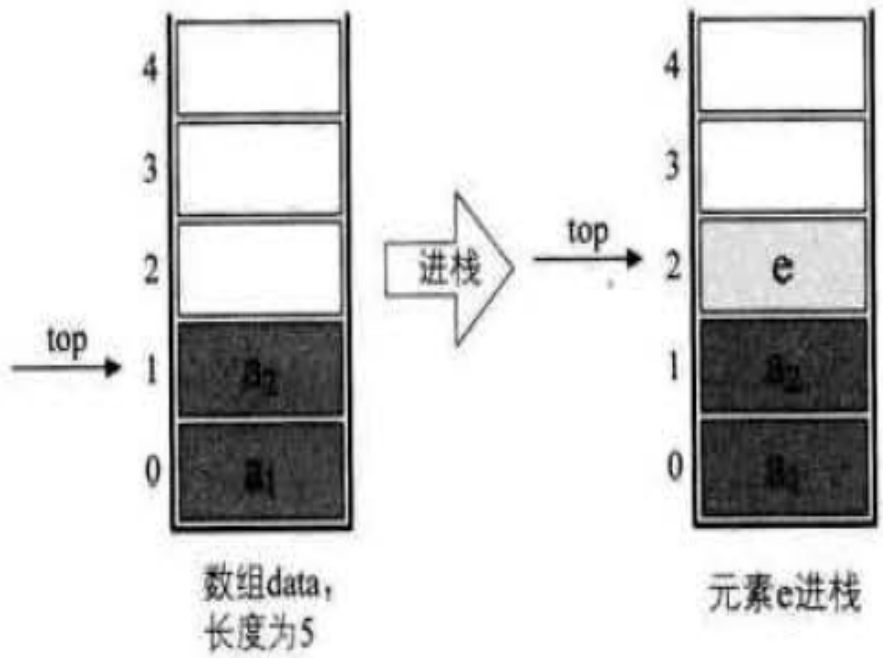
(7)顺序栈出栈

```
Status Pop(SqStack &S,SElemType &e){
  if(S.top==S.base) return ERROR;
  e=*--S.top;
  return OK;
}//Pop
```



顺序栈存储结构的定义

- 对栈的插入操作就是进栈操作。



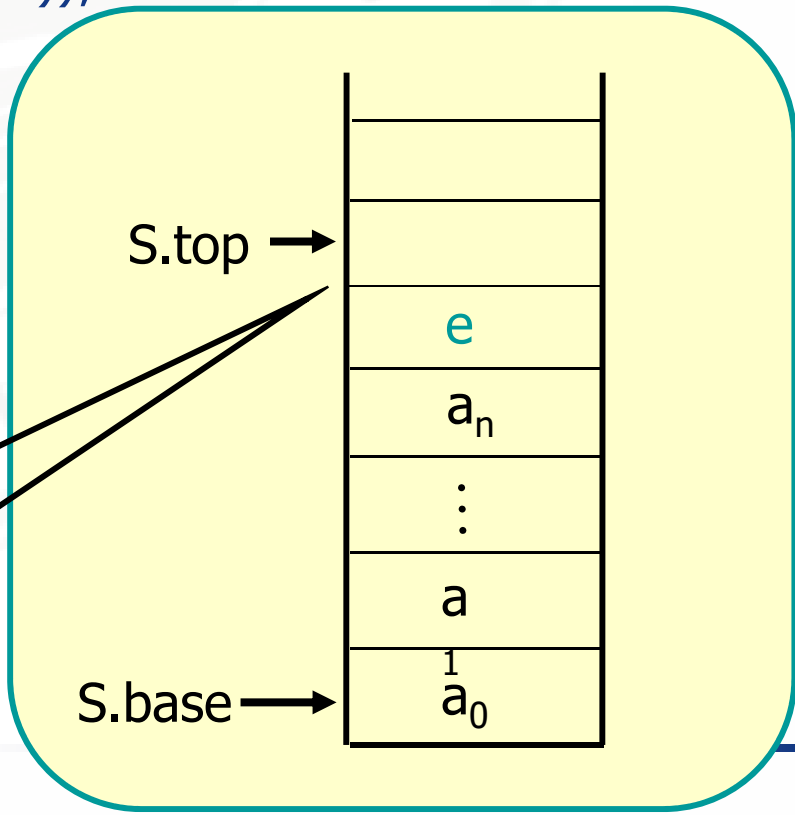
- 请写出进栈操作的算法。

(8) 顺序栈入栈

```
Status Push(SqStack &S,SElemType e){
    if(S.top-S.base>=S.stacksize){      //栈满,追加存储空间
        S.base=(SElemType *)realloc(S.base,(S.stacksize+
        STACKINCREMENT)*sizeof(SElemType));
          if(!S.base) return ERROR;
        S.top=S.base+S.stacksize;
        S.stacksize+=STACKINCREMENT;
    }
    *S.top++=e;
    return OK;
}//Push
```

*S.top=e

S.top++



(9) 顺序栈的遍历

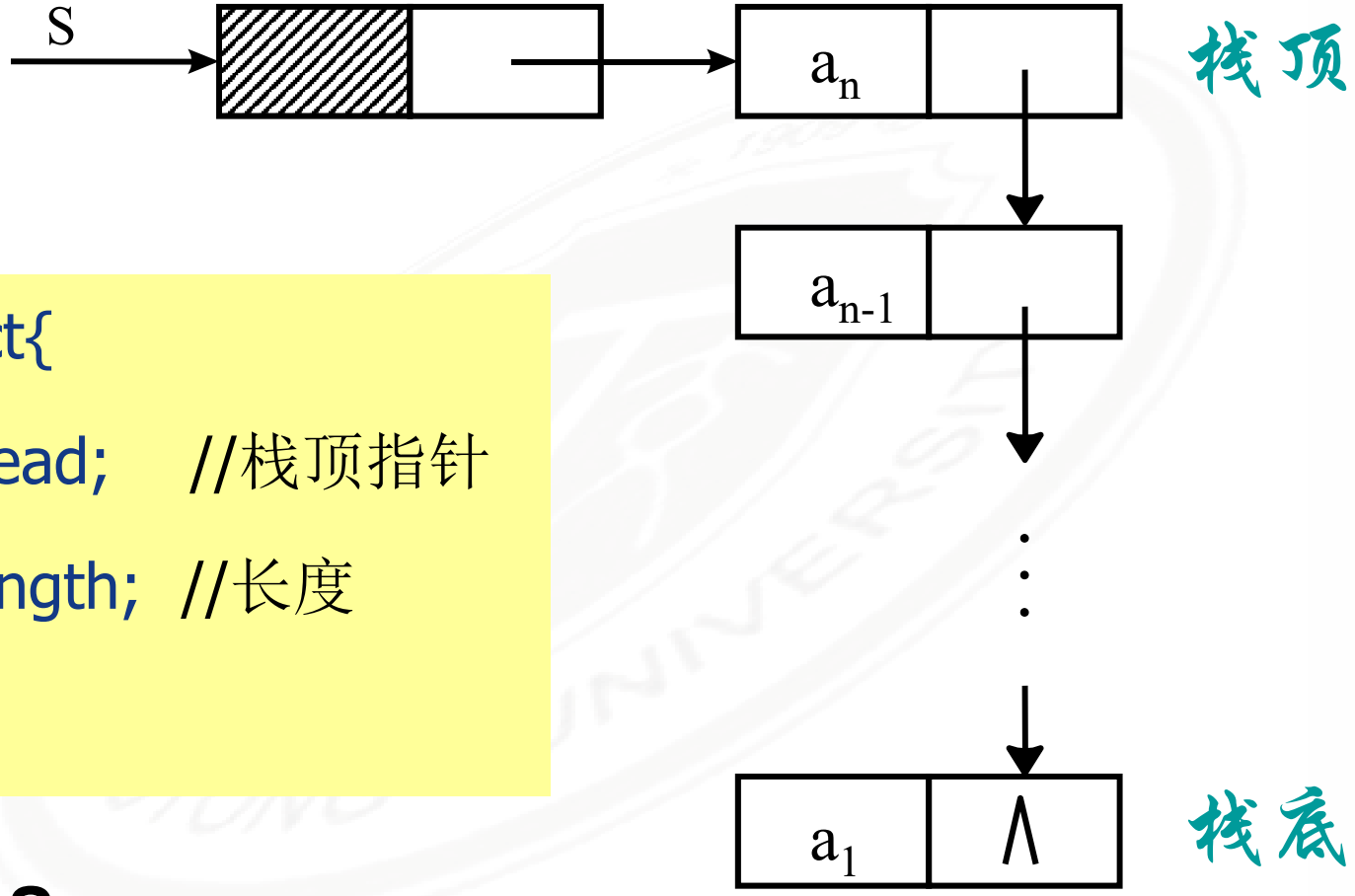
```
Status StackTraverse (SqStack S, Status (*visit)(ElemType)) {
    for(p=S.base; p<S.top; p++)
        if(!visit(*p)) return ERROR;
    return OK;
}// StackTraverse

Status visit(ElemType e){
   printf("The value of the element is %d\n", (int) e);
   return OK;
}
```

本节内容

- 3.1.1 栈的概念
- 3.1.2 栈的基本操作
- 3.1.3 顺序存储结构及操作
- 3.1.4 链式存储结构及操作
- 3.1.5 栈的应用

栈的链式存储结构及操作

- 栈的链式存储结构简称栈链。

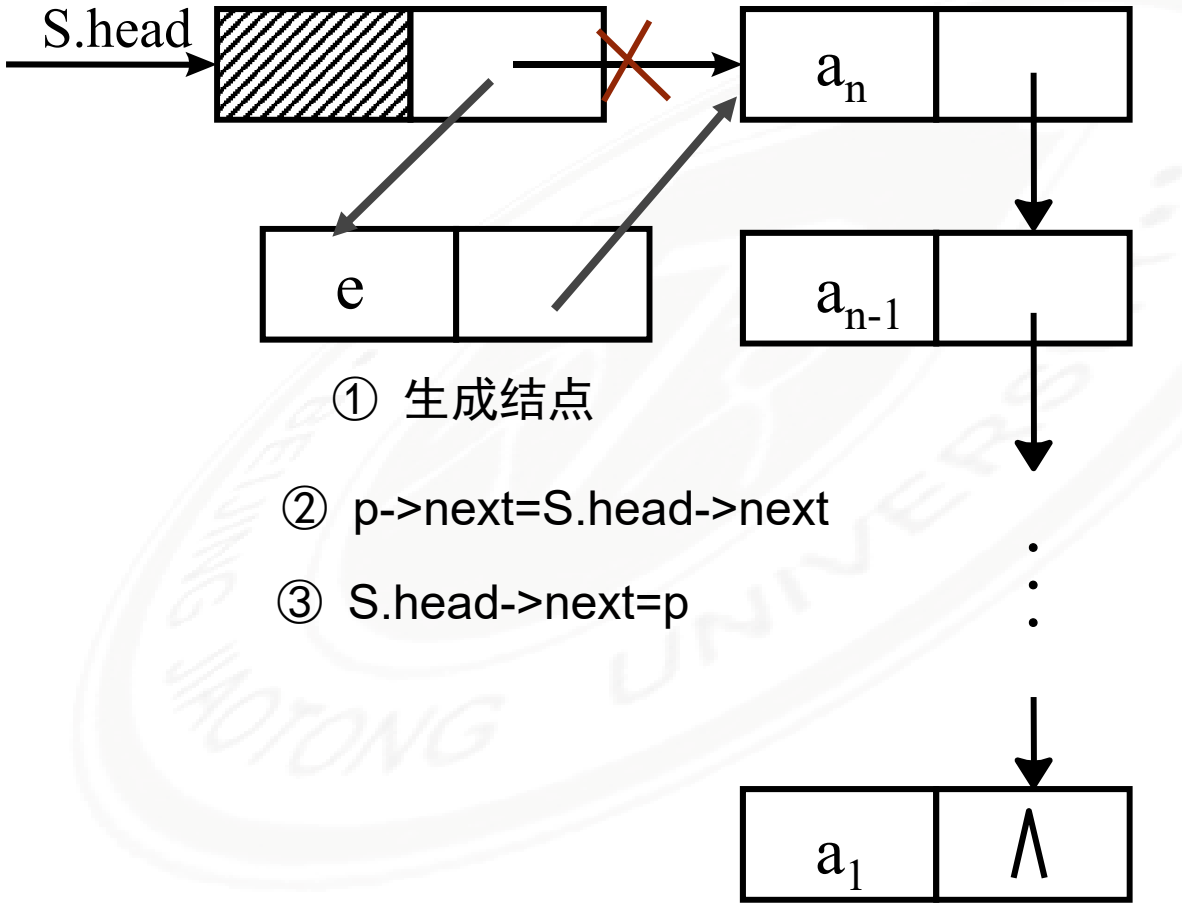



```
typedef struct{
   LinkList  Head;    //栈顶指针
   int       length;  //长度
}LinkStack;
```

LinkStack S;

课堂练习

- 请设计链栈的push操作和pop操作。



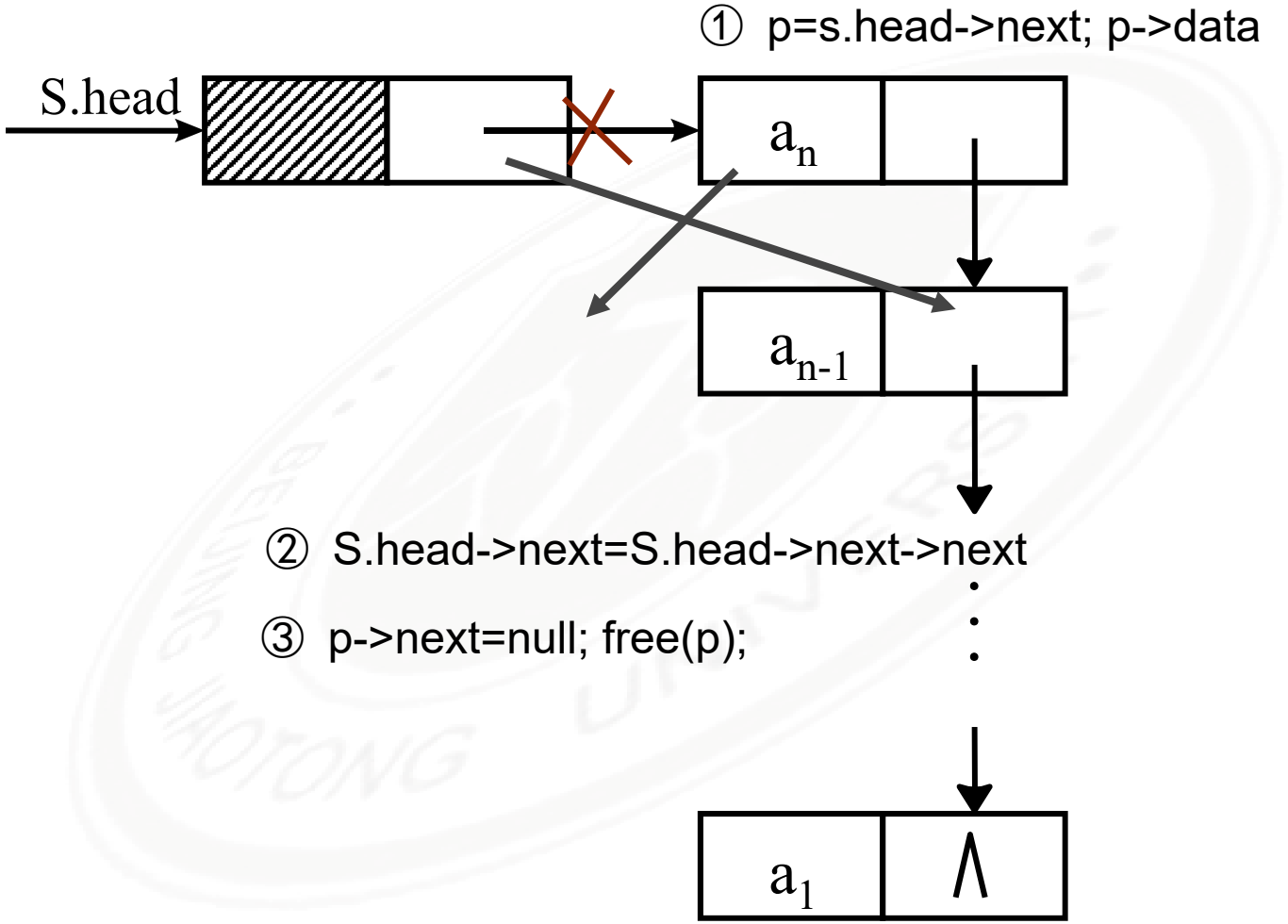
课堂练习

- 请设计链栈的push操作和pop操作。

```
Status Push (LinkStack &S,SElemType e) {
    if(Linklist *p=(Linklist) malloc(sizeof(StackNode))==NULL)  return ERROR;
     p->data=e;
     p->next=s.head->next;
     s.head->next=p;
     s.length++;
     return OK;
}// Push
```

课堂练习

- 请设计链栈的push操作和pop操作。



课堂练习

- 请设计链栈的push操作和pop操作。

```
Status Pop (LinkStack &S,SElemType &e) {
    Linklist p;
     if(p=S.head->next==NULL)    return ERROR;
     e=p->data;
     s.head->next=p->next;
     p->next=NULL;
     free(p);
     s.length--;
     return OK;
}// Push
```

本节内容

- 3.1.1 栈的概念
- 3.1.2 栈的基本操作
- 3.1.3 顺序存储结构及操作
- 3.1.4 链式存储结构及操作
- 3.1.5 栈的应用

3.1.5 栈的应用举例

- 数制转换
- 括号匹配检验

例1 数制转换

$$y = y_0 + y_1 \cdot d + y_2 \cdot d^2 + \ldots + y_m \cdot d^m$$

$$N = (N \text{ div } d) \times d + N \bmod d$$

		余数	对应的八进制数位
8	1348	··· 4	y_0
8	168	··· 0	y_1
8	21	··· 5	y_2
8	2	··· 2	y_3
	0		

北京交通大学
BEIJING JIAOTONG UNIVERSITY

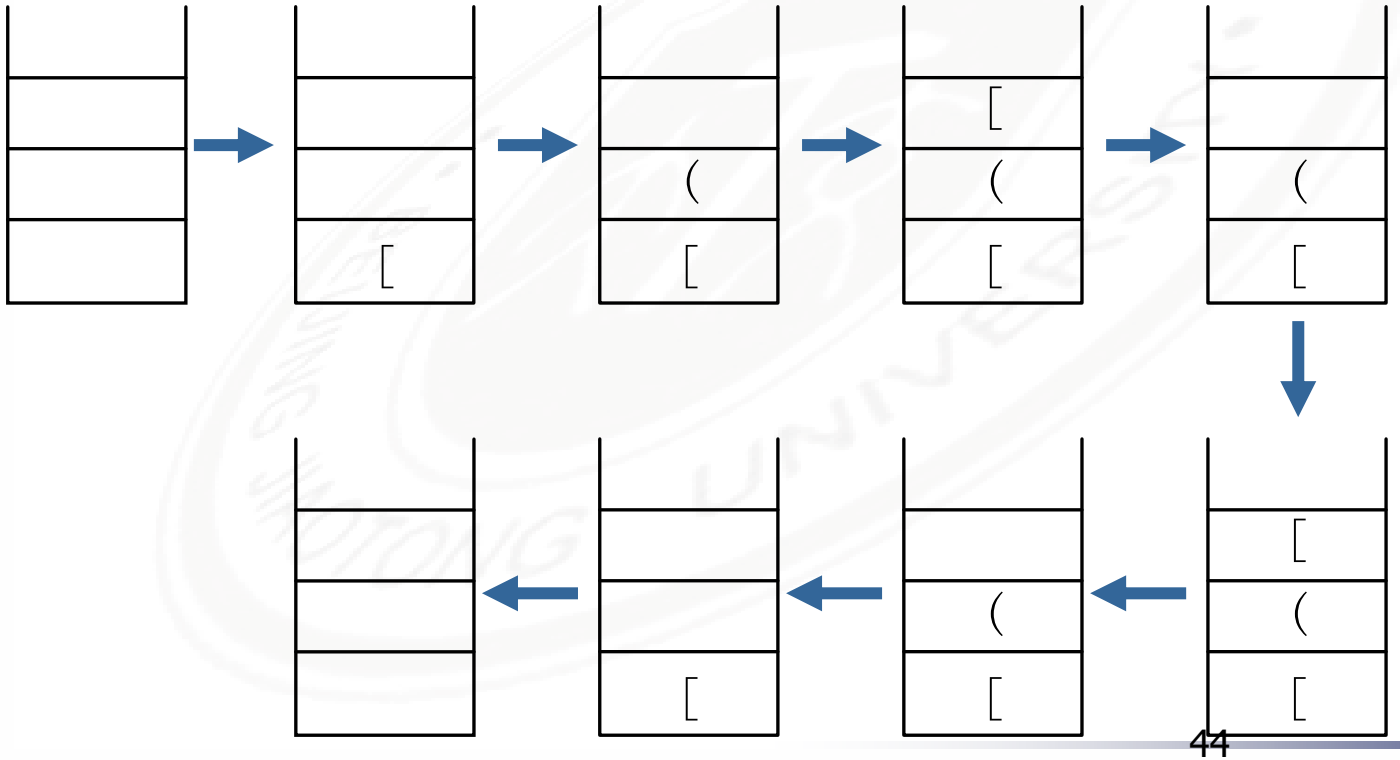
例1 数制转换

```
void conversion(){
  //对于输入的任意一个非负十进制数,打印输出与其等值的八进制数
  InitStack(S);
  scanf("%d",N);
  while(N){
    Push(S,N%8);
    N=N/8;
  }
  while(!StackEmpty(S)){
    Pop(S,e);
    printf("%d",e);
  }
}//conversion
```

例2 括号匹配检验

“期待的急迫程度”

[	(	[	]	[	]	)	]
1	2	3	4	5	6	7	8



实验4 回文判断

- 假设称正读和反读都相同的字符序列为“回文”，例如‘abba’和‘abcba’是回文，‘abcde’和‘ababab’则不是。
- 试写一个算法判别读入的一个以‘@’结束符的字符序列是否是“回文”

```
Status JudgeHuiwen ( )
{
    InitQueue(Q);
    InitStack(S);
    while((c=getchar())!='@'){
        Push(S, c);
        EnQueue(Q,c)
    };
    while(!StackEmpty(S)){
        Pop(S, a);
        DeQueue(Q, b);
        if(a!=b) return FALSE;
    }
    return TRUE;
}
```