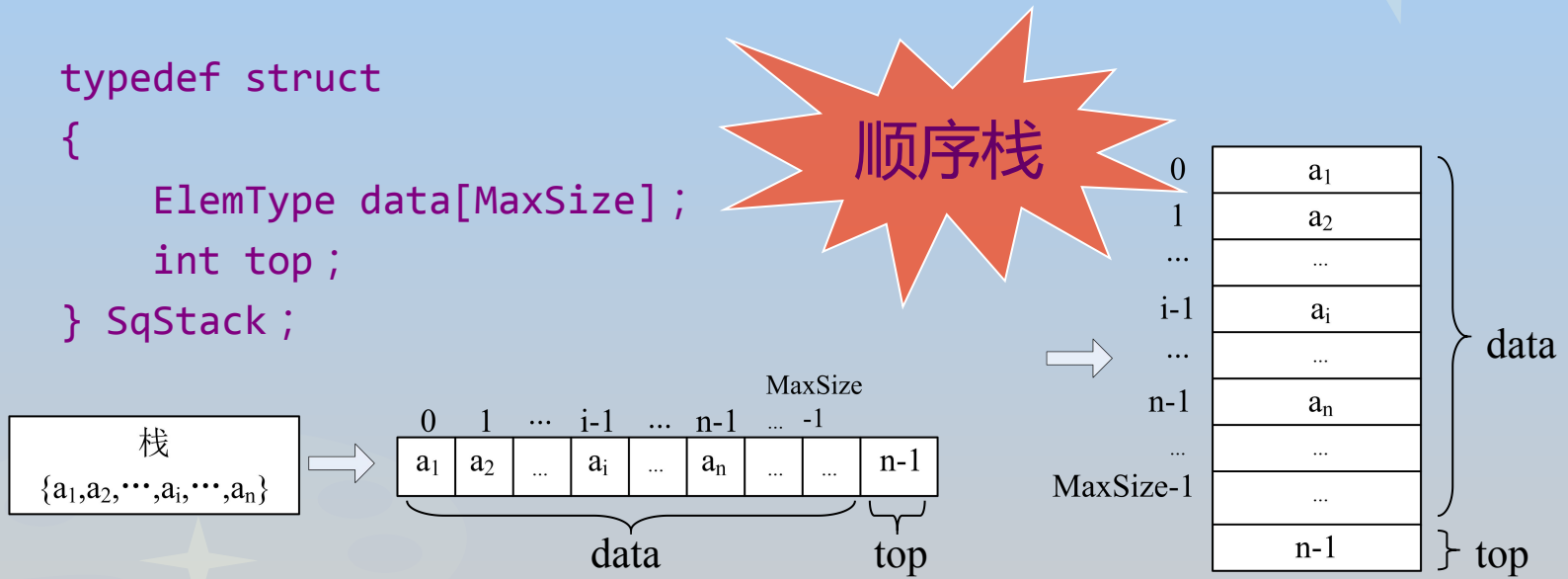
本节主题：

栈的顺序存储结构及其基本运算实现

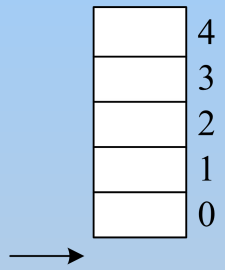
栈的顺序存储结构

- 描述栈
  - 数据元素：元素具有同一类型（ElemType），最多MaxSize。
  - 当前栈顶：记录栈顶的下标（栈顶指针）

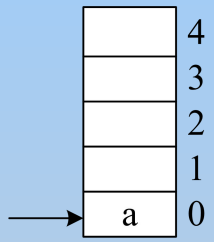
```
typedef struct
{
    ElemType data[MaxSize];
    int top;
} SqStack;
```



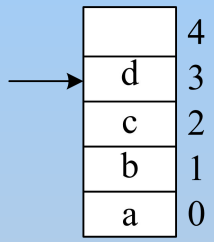
顺序栈4要素：以MaxSize=5的栈为例



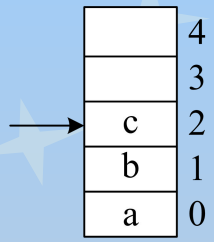
(1) 空栈



(2) 元素a入栈



(3) 元素b,c,d入栈



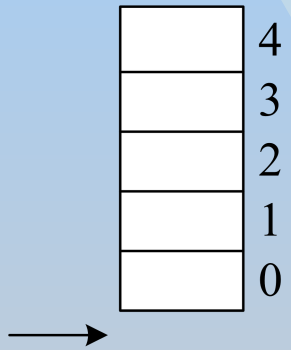
(4) 元素d出栈

- 栈空条件： top=-1
- 栈满条件： top=MaxSize-1
- 进栈e操作：top++; 将e放在top处
- 退栈操作： 从top处取出元素e; top--;

初始化栈initStack(&s)

- 建立一个新的空栈s,实际上是将栈顶指针指向-1即可

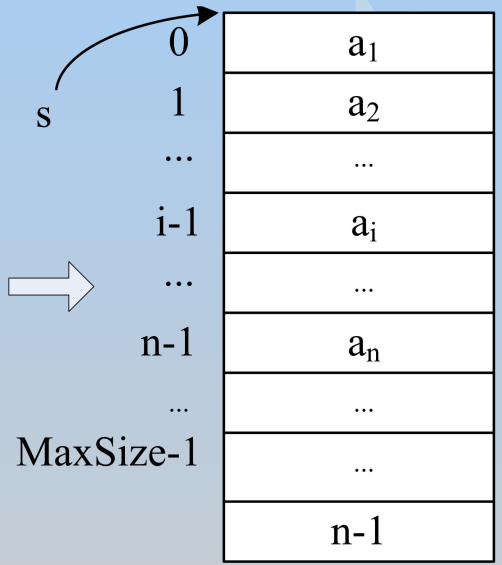
```
void InitStack(SqStack *&s)
{
    s=(SqStack *)malloc(sizeof(SqStack));
    s->top=-1;
}
```



销毁栈ClearStack(&s)

- 释放栈s占用的存储空间

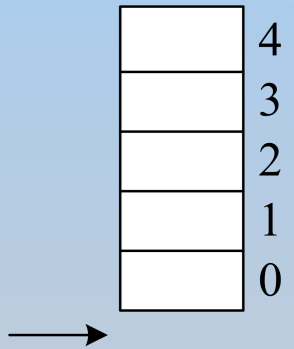
```
void DestroyStack(SqStack *&s)
{
    free(s);
}
```



判断栈是否为空StackEmpty(s)

- 要点：栈S为空的条件是s->top==-1

```
bool StackEmpty(SqStack *s)
{
    return(s->top==-1);
}
```

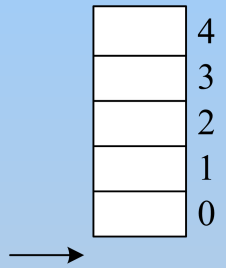


进栈Push(&s,e)

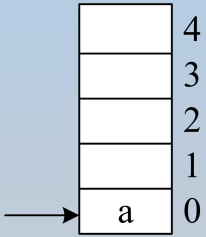
- 条件：在栈不满时，可以进栈
- 操作
  - 将栈指针增1
  - 在该位置上插入元素e

```
bool Push(SqStack *&s,ElemType e)
{
    if (s->top==MaxSize-1)
        return false;
    s->top++;
    s->data[s->top]=e;
    return true;
}
```

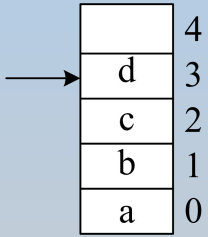
栈上溢出



(1) 空栈



(2) 元素a入栈



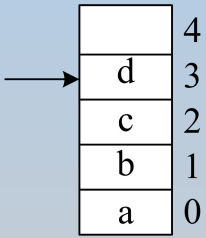
(3) 元素b,c,d入栈

出栈Pop(&s,&e)

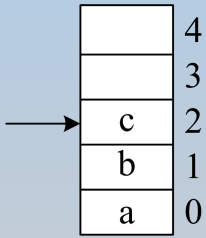
- 条件：栈不为空时
- 操作
  - 将栈顶元素赋给e
  - 将栈指针减1

```
bool Pop(SqStack *&s,ElemType &e)
{
    if (s->top==-1)
        return false;
    e=s->data[s->top];
    s->top--;
    return true;
}
```

栈下溢出



(1) 元素a,b,c,d在栈中



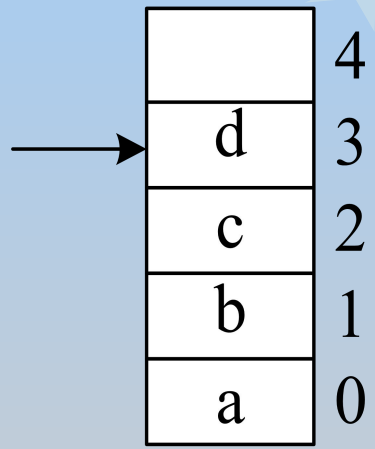
(2) 元素d出栈

取栈顶元素GetTop(s,e)

- 条件：栈不为空时
- 操作：将栈顶元素赋给e

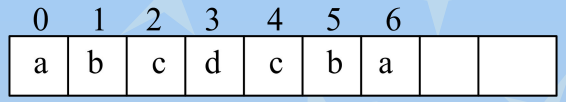
```
bool GetTop(SqStack *s,ElemType &e)
{
    if (s->top==-1)
      return false;
    e=s->data[s->top];
    return true;
}
```

栈下溢出

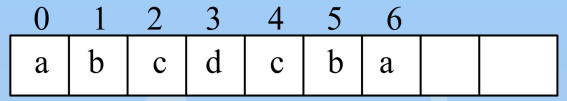


用栈的思维解决问题

- 问题
  - 编写一个算法利用顺序栈判断一个字符串str是否是对称串
  - 对称串：从左向右读和从右向左读的序列相同
- 解法1
  - 顺序表存储
  - 从两边“夹逼”，直到得出结论
- 解法2
  - 顺序栈存储
  - 先将str所有元素进栈，出栈的将是逆序的字符串
  - 从头开始扫描str，并出栈元素，将两者进行比较
  - 实质：从头开始扫描str是从左向右读，出栈序列是从右向左读，两者相等说明该串是对称串。



用栈判断对称串算法




```
bool symmetry(ElemType str[])
{
    int i;
    ElemType e;
    SqStack *st;
    InitStack(st);
    //将串所有元素进栈
    //出栈的字符与从左向右读取的字符串比较
    DestroyStack(st);
    return true;
}
```

```
for (i=0; str[i]!='\0'; i++)
    Push(st,str[i]);
```

```
for (i=0; str[i]!='\0'; i++)
{
    Pop(st,e);
    if (str[i]!=e)
    {
        DestroyStack(st);
        return false;
    }
}
```