积累点滴智慧，筑梦人生高峰

第07讲 堆栈

信息学院（智能应用研究院）
欧 新 宇

建议课时：2

第3章 栈和队列

考核要点

- **考核大纲**

  栈和队列的基本概念、栈和队列的顺序存储和链式存储、 栈和队列的应用

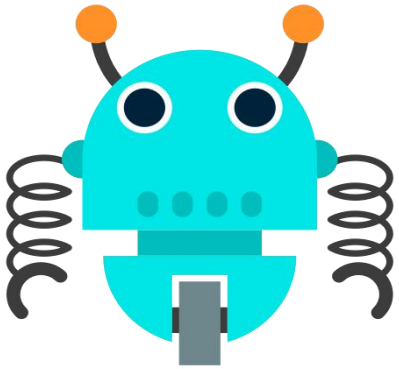
- **复习要点**

  - 掌握栈的基本操作，特别是出入站的过程、出栈序列的合法性
  - 熟练掌握栈的顺序、链式存储的基本操作实现算法，特别是栈满和栈空的条件
  - 熟练掌握标准队列、双端队列、循环队列和链队列的概念、基本操作的实现算法，特别注意队满和队空的条件
  - 掌握栈和队列的特点，并能在相应的应用问题中正确选用
  - 重点掌握算法的思想！能够使用类C或类C++语言实现

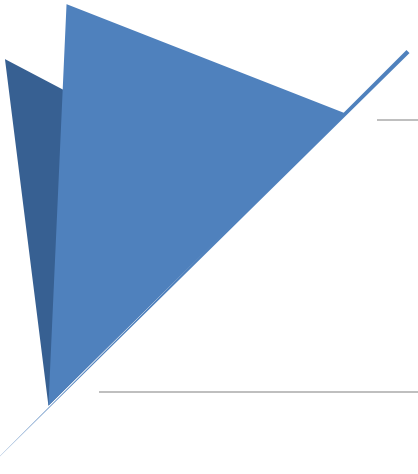
第08讲 堆栈

本讲作业

- **必做题**
  - ✓ 【课堂互动7.1】栈的基本概念
  - ✓ 【课堂互动7.2】栈的顺序存储实现
  - ✓ 【课堂互动7.3】栈的链式存储实现
  - ✓ 【课后作业07】堆栈
- **选做题**
  - ✓ 【课前自测07】堆栈
  - ✓ 【扩展练习07】堆栈



- 堆栈的基本概念
- 堆栈的顺序存储实现
- 堆栈的链式存储实现



堆栈的基本概念

/ 堆栈的定义

/ 堆栈的特点：FILO+LIFO

/ 堆栈的存储结构

/ 元素出栈序列的合法性

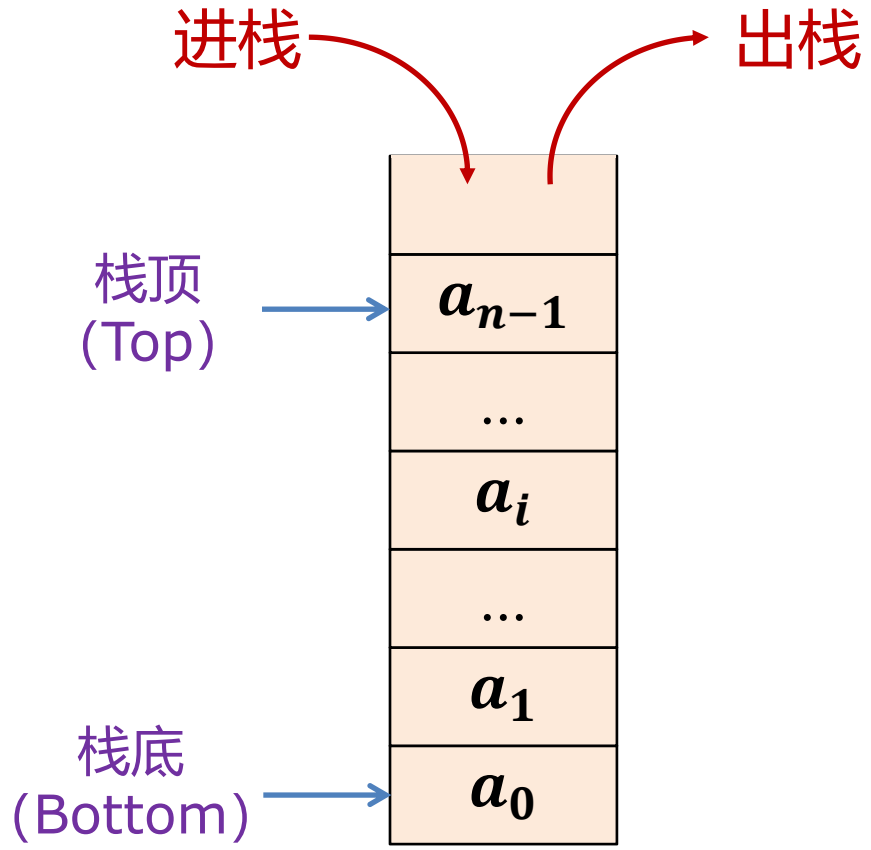
堆栈的基本概念

什么是堆栈?

栈（Stack）又称**堆栈**，是只允许在一端进行插入和删除操作的受限制的线性表。

$$S = (a_0, a_1, \ldots, a_i, \ldots, a_{n-1})$$

后进先出LIFO、先进后出FILO
(Last In First Out)
(First In Last Out)

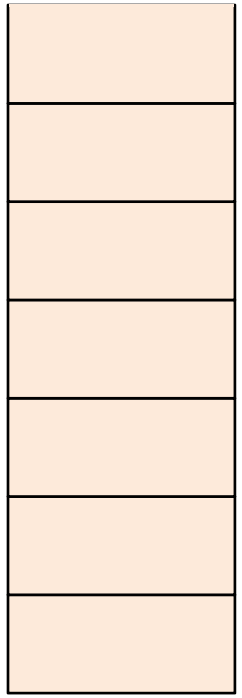


堆栈的基本概念

栈的特点：先进后出(FILO)、后进先出（LIFO）

C B A

进栈



出栈

堆栈的基本概念

栈的抽象数据类型描述

ADT Stack{

数据对象：一个包含0个或多个元素的有穷线性表。$D = \{a_i | a_i \in ElemSet, i \in \mathbb{N}^+, i \leq n\}$

数据关系：$R = \{< a_i, a_{i+1} > | a_i, a_{i+1} \in D, i \leq n\}$

基本操作：

Stack InitStack(&S)：初始化一个空堆栈。

bool StackEmpty(S)：判断堆栈S是否为空，返回True|False。

bool StackFull(S)：判断堆栈S是否已满，返回True|False。

bool Push(&S, x)：进栈，若栈未满，则将元素x压入堆栈，x变为新栈顶。

注意区别

ElementType PoP (&S, &x)：出栈，若栈非空，则弹出栈顶元素，并返回x。

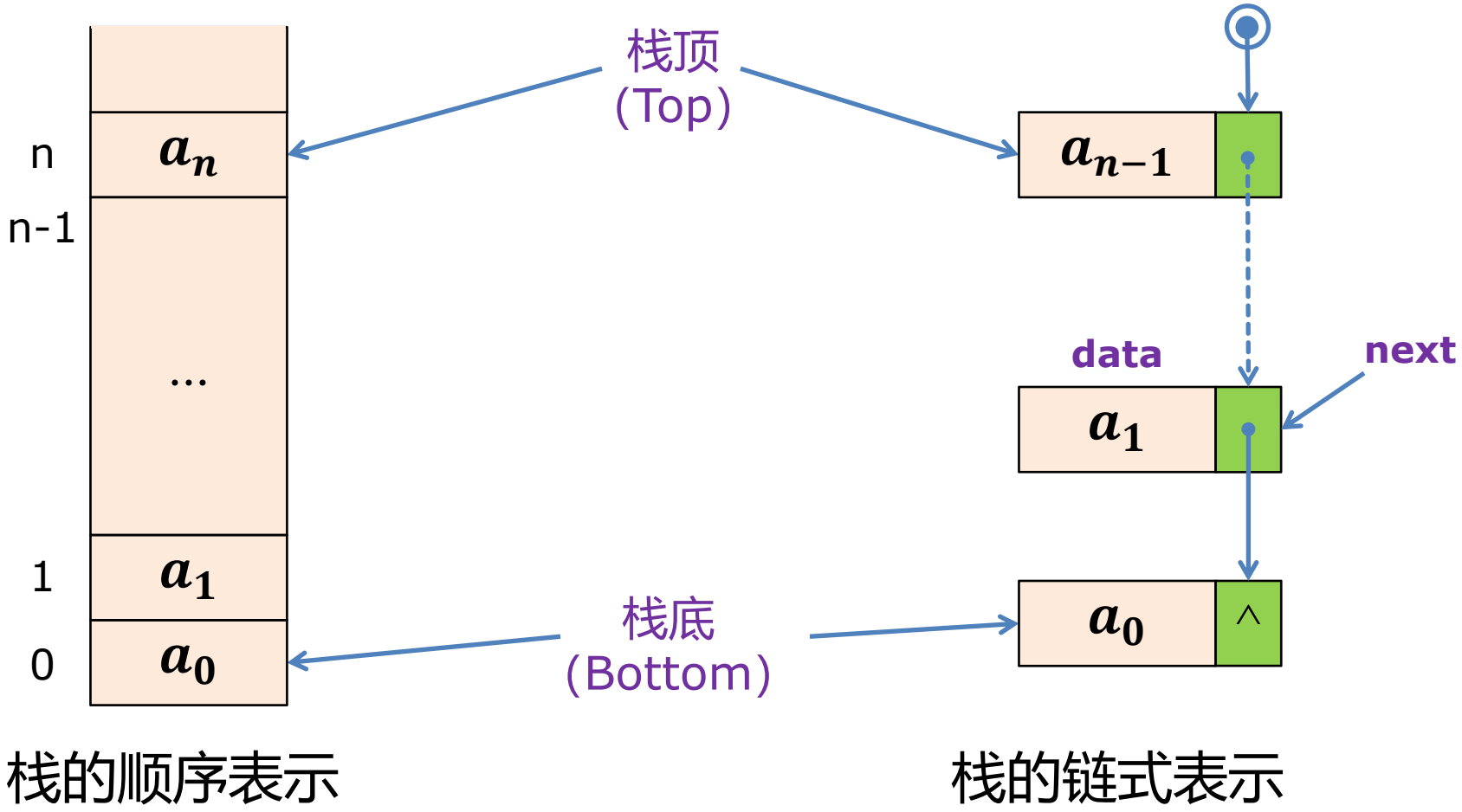
ElementType GetTop (S, &x)：读栈顶元素，若栈非空，则用x返回栈顶元素。

Stack DestoryStack (&S)：销毁栈，并释放栈占用的空间。

} ADT Stack

堆栈的基本概念

栈的顺序表示和链式表示

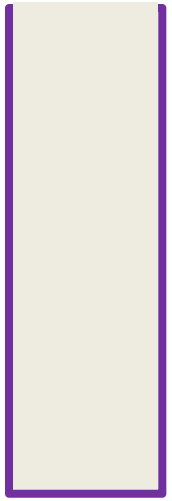


堆栈的基本概念

元素出栈序列的合法性

例1：如果三个字符按ABC的顺序压入栈S，则其出栈顺序是什么?

Push(S, A)
Push(S, B)
Push(S, C)
Pop(S)
Pop(S)
Pop(S)



是否有其他可能呢?

Push(S, A) -> PoP(S)
=> A
-> Push(S, B) -> Push(S, C)
-> PoP(S) ->PoP(S)
=> ACB

Push(S, A) -> Push(S, B)
-> PoP(S) -> Push(S, C)
-> PoP(S) ->PoP(S)
=> BCA

对于包含n个入栈元素的栈，其出栈可能性为$\frac{1}{n+1}C_{2n}^{n}$种

堆栈的基本概念

元素出栈序列的合法性

例2：若元素的进栈序列为ABCDE，运用栈操作，能否得到出栈序列BCAED和DBACE？为什么？

1. 可以得到BCAED，

其顺序为A进，B进，B出，C进，C出，A出，D进，E进，E出，D出。

2. 无法得到DBACE。

D是第一个出来的，说明ABC是按顺序进站。

所以，出栈就只能是CBA，ccc

因此题目给出的BAC是错误的。

解题技巧：

1. 从输出开始，放入堆栈，并依次对比
2. 每个输出执行前，确保前面的元素都入栈
3. 按照FILO的原则，检查是否能完成所有元素的输出

堆栈的基本概念

元素出栈序列的合法性

例2：若元素的进栈序列为ABCDE，运用栈操作，能否得到出栈序列BCAED和DBACE？为什么？

1. 可以得到BCAED，

其顺序为A进，B进，B出，C进，C出，A出，D进，E进，E出，D出。

2. 无法得到DBACE。

D是第一个出来的，说明ABC是按顺序进站。

所以，出栈就只能是CBA，

因此题目给出的BAC是错误的。

单选题 1分

设置

若让元素1，2，3，4，5依次进栈，则出栈次序不可能出现在（ ）种情况。

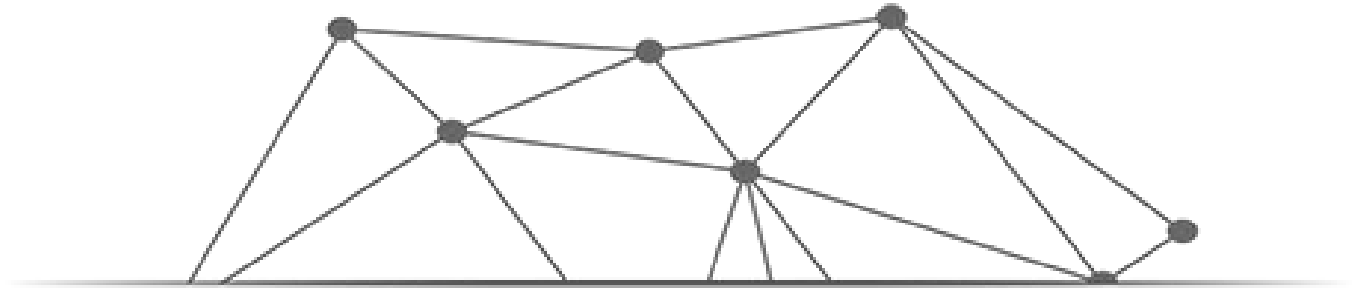
A 5，4，3，2，1

B 2，1，5，4，3

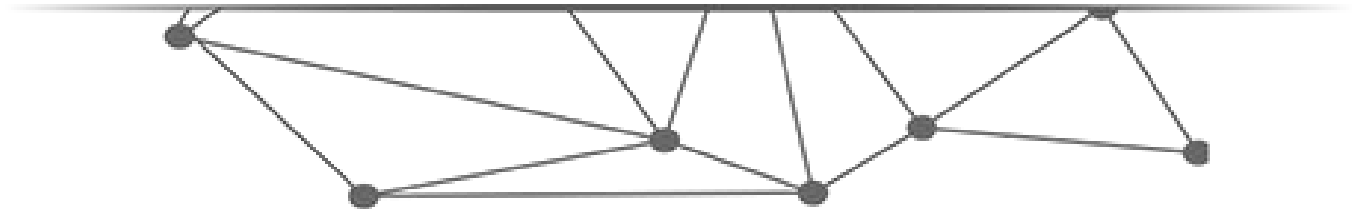
C 4，3，1，2，5

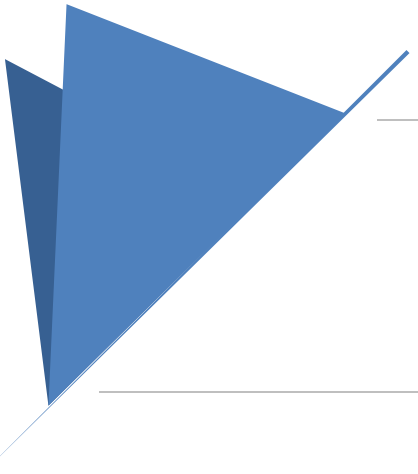
D 2，3，5，4，1

提交



课堂互动 7.1





堆栈的顺序存储实现

/ 顺序栈和顺序表的关系

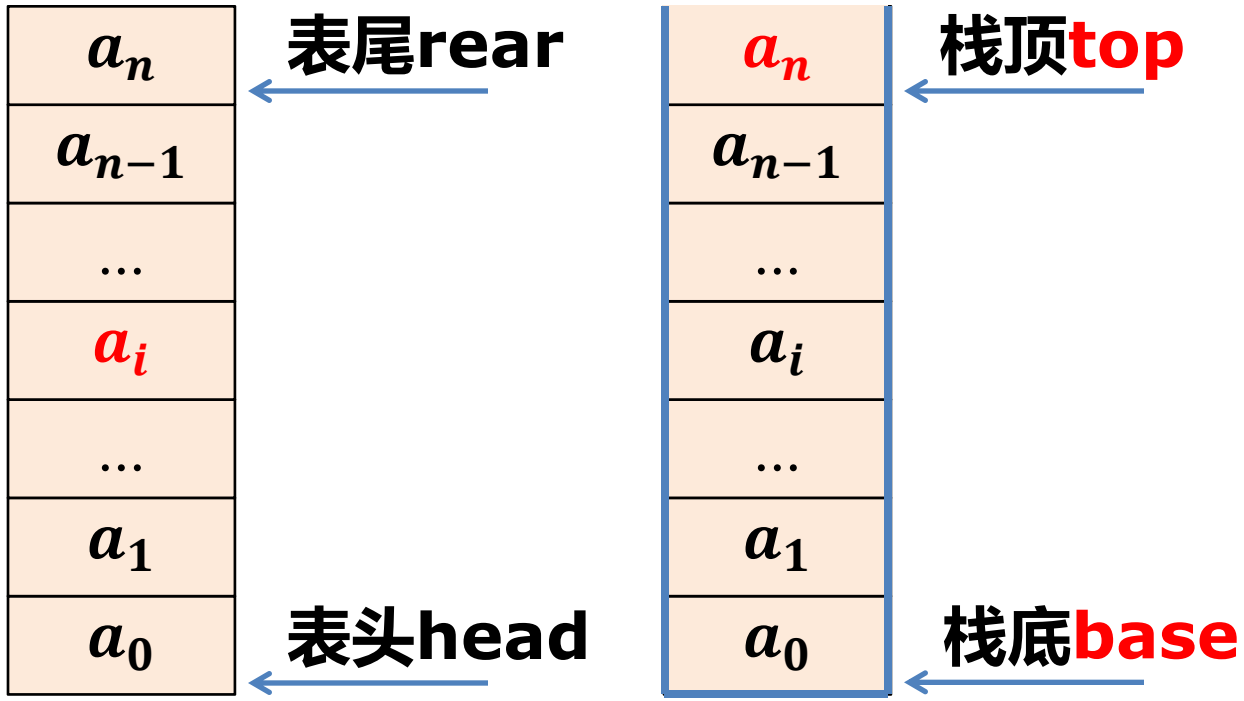
/ 顺序栈的存储结构

/ 顺序栈的基本操作

/ 共享栈

栈的顺序存储实现

顺序栈和顺序表



写入：$\mathrm{L(i)} = a_i$
读取：$x = \mathrm{L(i)}$

写入：$\mathrm{Push(S}, a_n\mathrm{)}$
读取：$x = \mathrm{Pop(S)}$

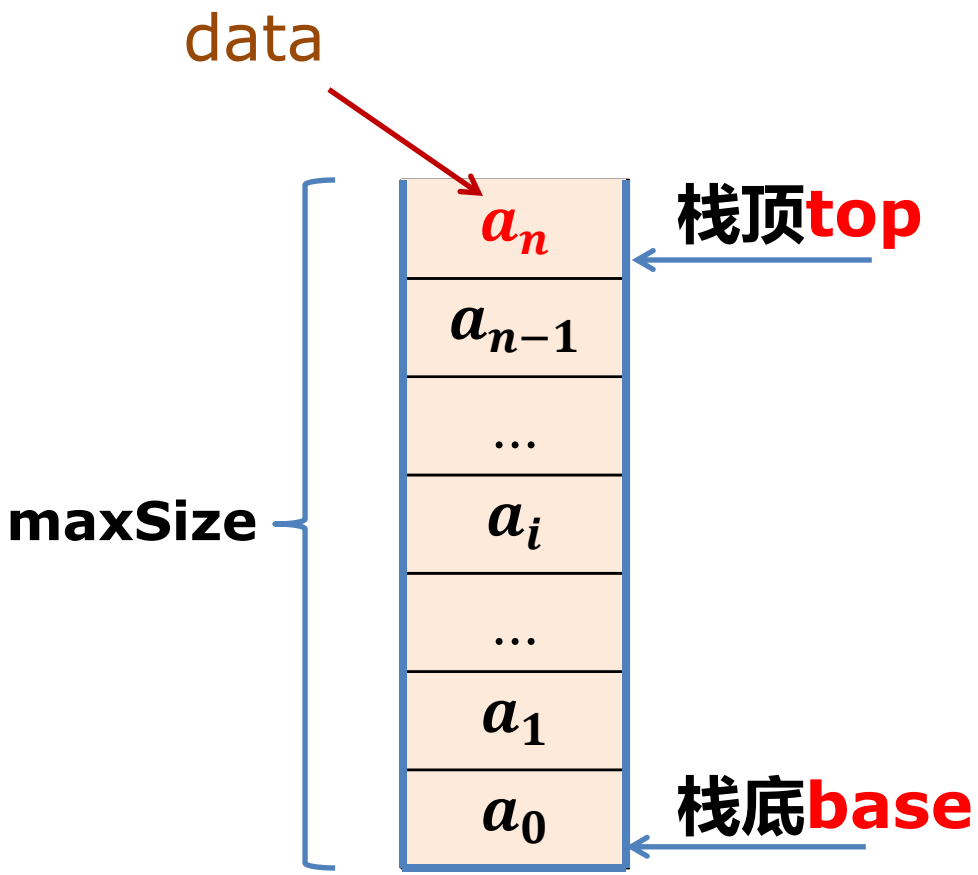
- base：栈底指针，始终指向栈底
- top：栈顶指针，始终指向栈顶的元素

- **空栈：**
  - ✓ **top == base** base指针存在
  - ✓ **top == -1** base指针不存在

- **栈满的处理方法：**
  - ✓ 报错，返回系统
  - ✓ 分配更大空间，并将当前栈数据移动到新栈

栈的顺序存储实现

顺序栈的存储结构



顺序栈的存储结构（有Base指针）

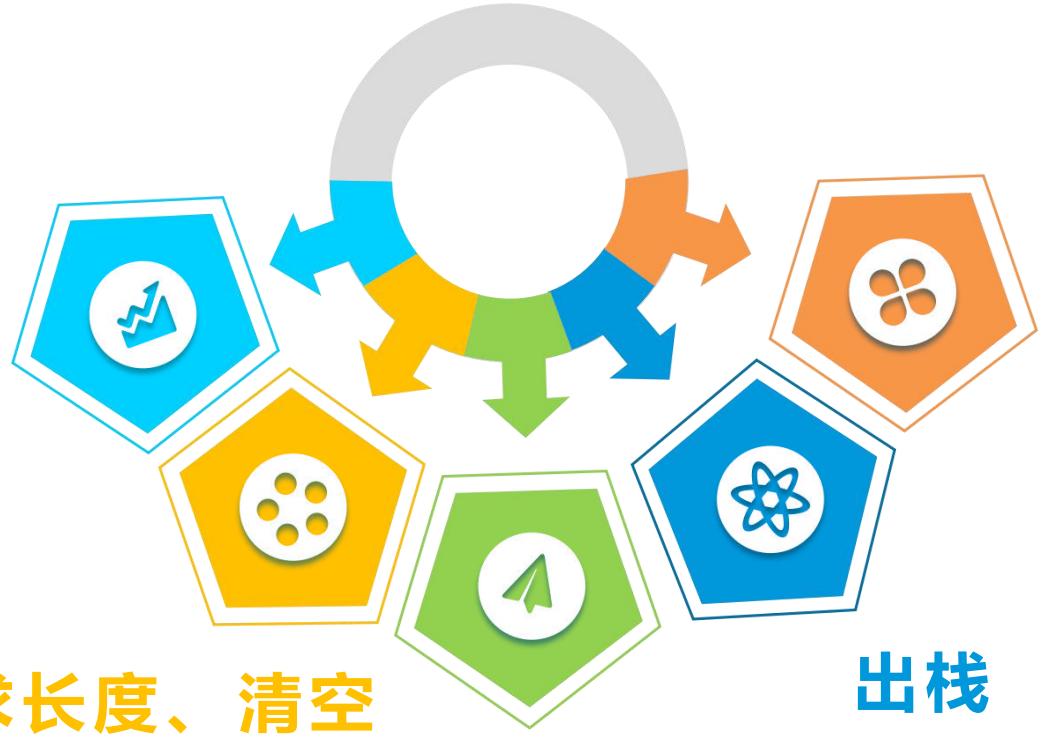
```
typedef MaxSize 50;                  // 栈最大容量
typedef struct SqStack {             // 定义顺序栈
    SElementType *base;              // 栈底指针
    SElementType *top;               // 栈顶指针
    SElementType Data[MaxSize];      // 栈结点空间
}
```

顺序栈的存储结构（没有Base指针）

```
typedef MaxSize 50;                  // 栈最大容量
typedef struct SqStack {             // 定义顺序栈
    SElementType *top;               // 栈顶指针
    SElementType Data[MaxSize];      // 栈结点空间
}
```

栈的顺序存储实现

顺序栈的基本操作



初始化

取值

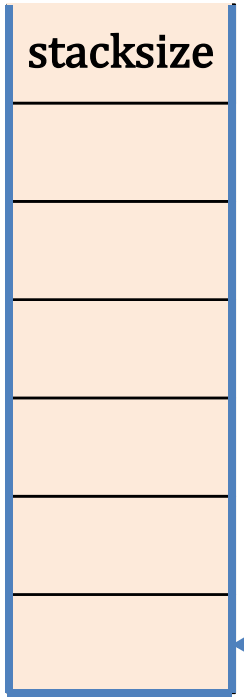
判空、销毁、求长度、清空

出栈

入栈

顺序栈的基本操作

顺序栈的初始化



1. 构造一个空栈
2. 分配空间并检查是否分配成功
3. 初始化栈顶指针
4. 设置栈的大小

1. base=top

2. top=-1

注意带base的栈top位于栈底，而不带base的栈top位于栈底-1

带base的顺序栈的初始化表

```
void InitStack (SqStack &S, MaxSize) {
   S.base = new SELemType[MaxSize];
   if (!S.base) exit (Overflow);
   S.top = S.base;
   S.stacksize = MAXSIZE;
} // InitStack
```

不带base的顺序栈的初始化表

```
void InitStack (SqStack &S, MaxSize) {
    Stack S = new SELemType[MaxSize];
    S.top = -1;
} // InitStack
```

顺序栈的基本操作

顺序栈的其他基本操作

求顺序栈的长度

```
int StackLength (SqStack S) {
    return S.top - S.base;  // 带base
    return S.top + 1        // 不带base
}
```

顺序栈判空

```
bool StackEmpty (SqStack *S) {
  if (S.top == S.base)    return true; // 带base
  if (S.top == -1)    return true;      // 不带base
  else return false;
}
```

清空顺序栈

```
Status ClearStack (SqStack S) {
   S.top = S.base;        // 带base
   S.top = -1;            // 不带base
   return OK;
}
```

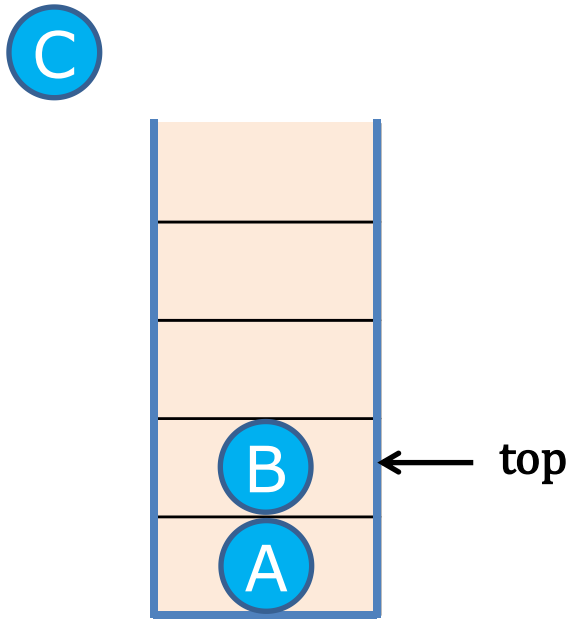
顺序栈判满

```
bool StackFull (SqStack *S) {
  if (S.top-S.base == S.MaxSize)    return true;
  if (S.top == S.MaxSize-1)    return true;
  else return false;
}
```

顺序栈的基本操作

顺序栈的入栈

1. 判断栈是否已满，若满则返回Error
2. 将元素压入栈顶
3. 栈顶指针加 1



入栈

```
Status Push (SqStack &S, SElemType e) {
    if (S.top - S.base == S.stacksize)      // 带base
    if (S.top == S.maxSize-1)               // 不带base
        printf("堆栈满，无法插入");
        return Error;
    *S.top++ = e;              // 先入栈，指针再+1(带base)
    S.data[++S.top] = e        // 指针先+1，再入栈(不带base)
    return OK;
}
```

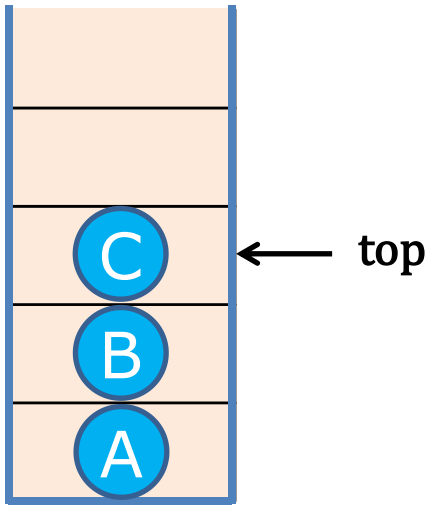
顺序栈的基本操作

顺序栈的出栈

1. 判断栈是否已空，若空则返回Error
2. 获取栈顶元素e
3. 栈顶指针减 1

出栈

```
Status Pop (SqStack &S, SElemType e) {
    if (S.top == S.base)        // 带base
    if (S.top == -1)            // 不带base
        printf("堆栈空");
        return Error;
    e = *--S.top;               // 指针先减1，再出栈(带base)
    e = S.data[S.top--]         // 先出栈，指针再减1(不带base)
    return OK;
}
```



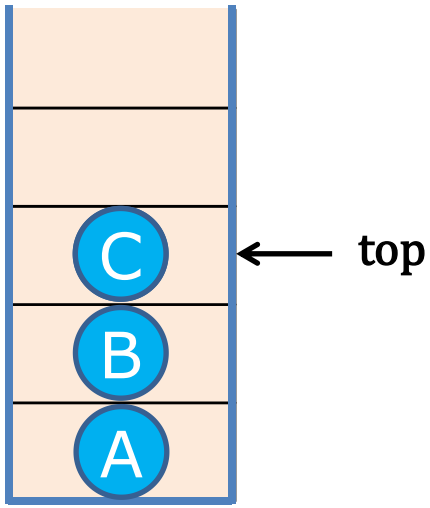
顺序栈的基本操作

顺序栈的取栈顶元素

1. 判断栈是否已空，若空则返回Error
2. 读取栈顶元素，并赋值给e，指针保持不变

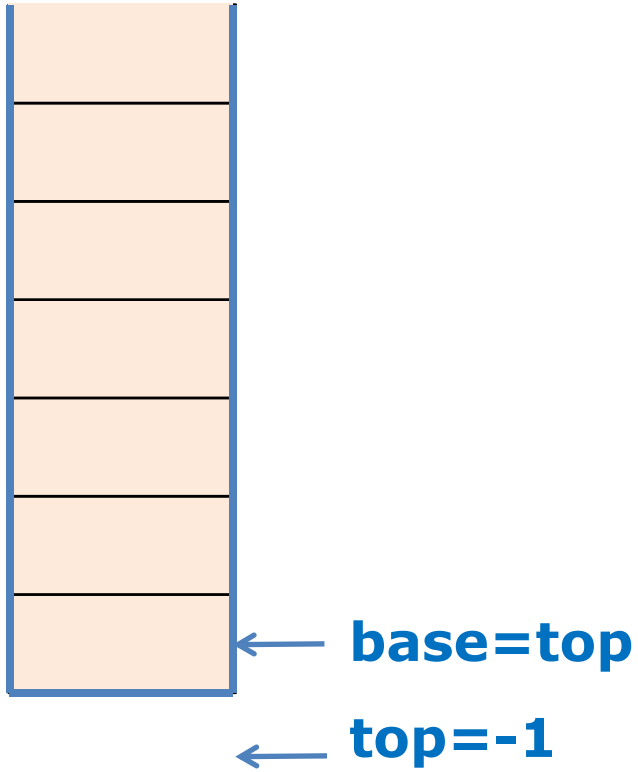
取栈顶元素

```
Status PGetTop (SqStack &S, SElemType e) {
    if (S.top == S.base)              // 带base
    if (S.top == -1)                  // 不带base
        printf("堆栈空");
        return Error;
    e = S.data[S.top-1];              // 读取栈顶元素(带base)
    e = *S.top                        // 读取栈顶元素(不带base)
    return OK;
}
```



顺序栈的基本操作

关于top的表示问题



- 在**栈**中，我们使用top来表示栈顶元素。在**不使用栈底指针base**的时候，栈顶指针通常位于**栈底-1**，即top=-1，此时，进栈操作为S.data[++S.top] = x，出栈操作为 x = S.data[S.top--]。（pDDp）
- 而当我们**使用栈底指针base**时，S.top=0，此时，进栈操作变为S.data[S.top++] = x，出栈操作为 x = S.data[--S.top]。（DppD）
- 相应的栈空、栈满操作也会发生变化。在具体使用时应灵活应变。

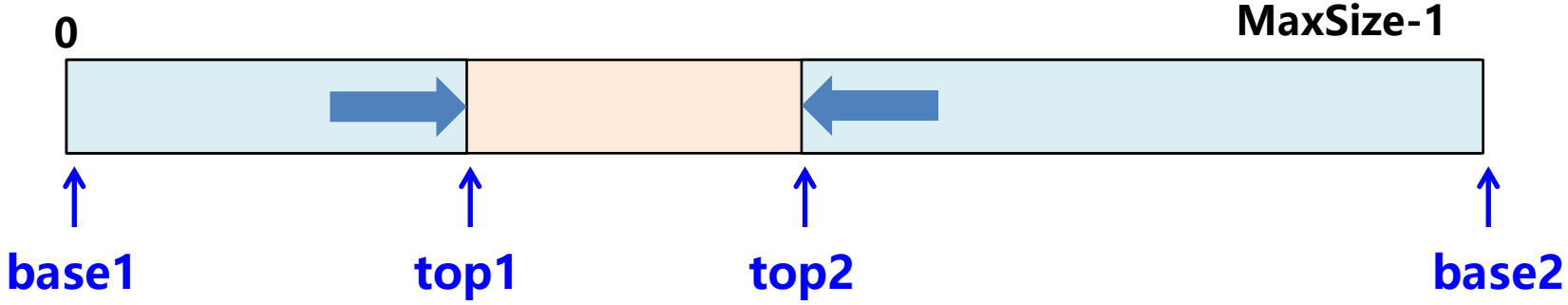
顺序栈的基本操作

例3：请使用一个数组实现两个堆栈，要求最大程度利用数组空间，使数组只要有空间入栈操作就能够成功。

a_n

$S1, S2$

问题分析：将这两个栈分别从数组的两头开始向中间生长。当两个栈的栈顶指针相遇时，表示两个栈都满了；而没有相遇时两个栈都可以根据需求向内进行生长。



共享栈的定义

```
# define MaxSize
typedef struct DStack {
        ElementType Data[MaxSize];
        int top1;
        int top2;
}
S.top1 = -1;
S.top2 = MaxSize;
```

top1 = -1时，1号栈为空；
top2 = MaxSize时，2号栈为空；
top2-top1 = 1时，栈满

顺序栈的基本操作

共享栈的入栈和出栈

共享栈的入栈

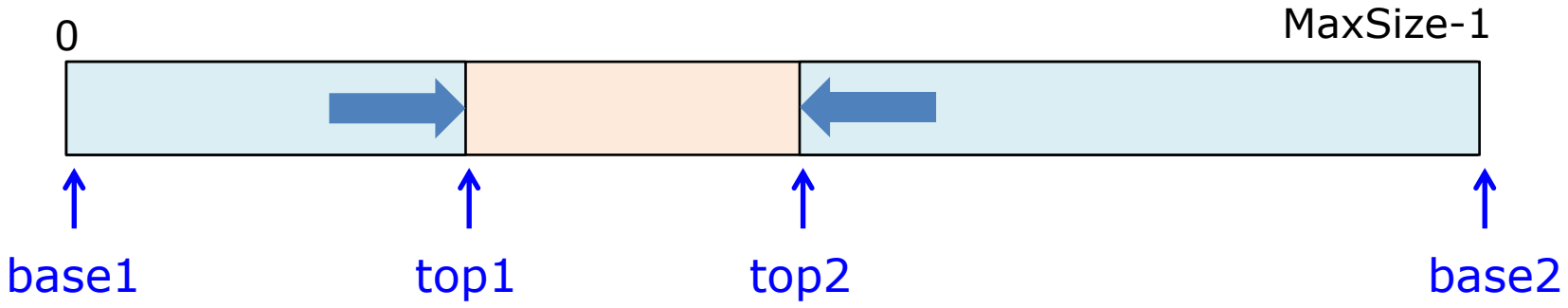
```
void Push (struct DStack &S, ElementType item, int Tag) {
    if (S.top2 - S.top1 == 1) {                    // 判断堆栈是否已满，只有在未满的时候才能进行入栈
        printf("堆栈满");    return;
    }
    if (Tag == 1)    S.data[++S.top1] = item;      // Tag标志用于标识启用栈1或栈2，对于栈1，入栈时top先加1
    else S.data[--S.top2] = item;                  // 对于栈2，入栈时top先减1
}
```

共享栈的出栈

```
void Pop (struct DStack &S, int Tag) {
    if (Tag == 1)                                  // 先选择待处理的栈：1|2
        if (S.top1 = -1) { printf("堆栈1空");    return; }    // 判断堆栈1是否为空，非空则无法出栈
        else return S.Data[S.top1--];              // 出站后，需要将栈1的top指针减1，指向栈1新的栈顶
    else {
        if (S.top2 = MaxSize) { printf("堆栈2空");    return; }  // 判断堆栈2是否为空，非空则无法出栈
        else return S.Data[S.top2++];              // 出站后，需要将栈2的top指针加1，指向栈2新的栈顶
}
```

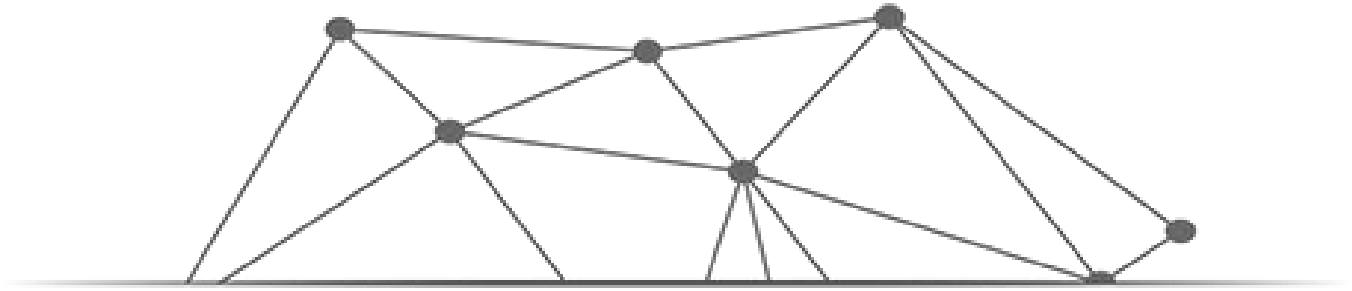
顺序栈的基本操作

共享栈的优点

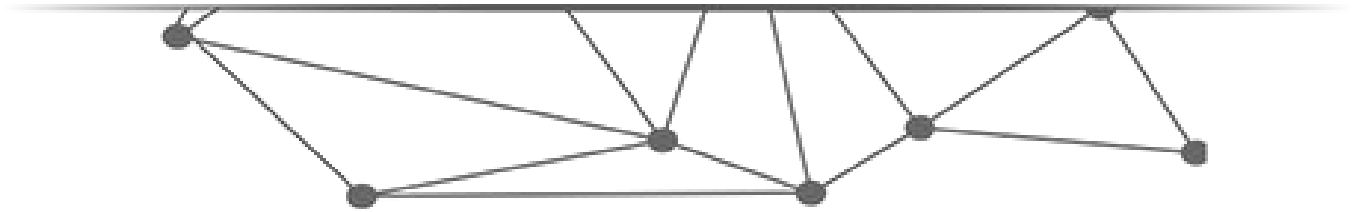


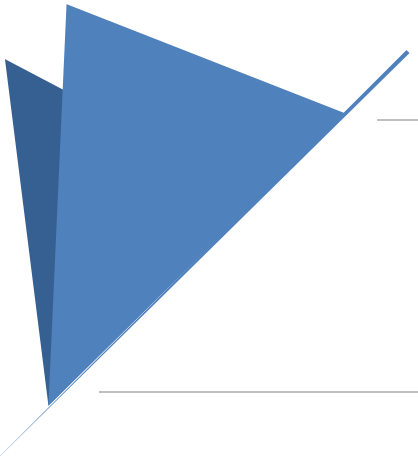
可以互相调剂，更好利用存储空间，灵活性强

只有在整个存储空间被占满时，才会发生上溢现象



课堂互动 7.2





堆栈的链式存储实现

/ 栈的存储结构

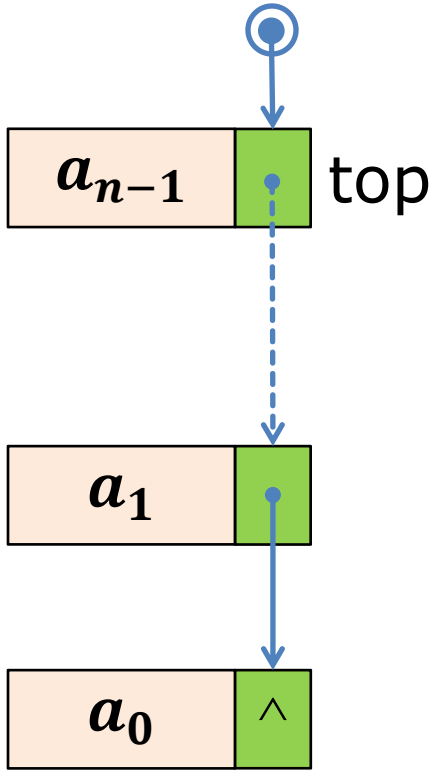
/ 栈的基本操作（初始化、判空）

/ 栈的入栈和出栈

栈的链式存储实现

链栈的存储结构

链栈（LinkStack）是栈的链式存储结构实现，它是一种受限的单链表，称为链栈、链式栈。在链栈中，结点的插入（入栈）和删除（出栈）操作只能在栈顶进行。链栈便于多个栈共享存储空间，有效地提高了存储效率，且不存在栈满上溢的问题。



链栈的存储结构（没有Base指针）

```
typedef struct StackNode {
    ElemType data;              // 数据元素
    struct StackNode *next;     // 下一个结点指针
} StackNode, *LinkStack
LinkStack S;
```

栈的链式存储实现

链栈的基本操作

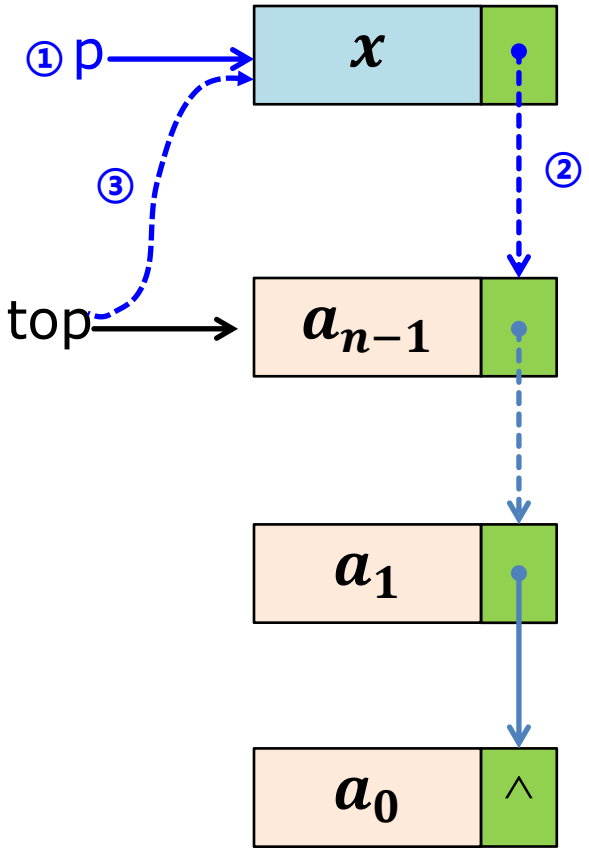
构造一个空链栈，栈顶指针置空

```
void InitStack () {
    LinkStack S;                    // 构造一个空栈
    S = (LinkStack)malloc (sizeof (struck SNode));
    S ->next = Null;                // 栈顶指针置空
    return S;
}
```

判断栈是否为空

```
bool StackEmpty (LinkStack S) {
    if (S ->next == NULL)
        return true;
    else
        return false;
}
```

栈的链式存储实现



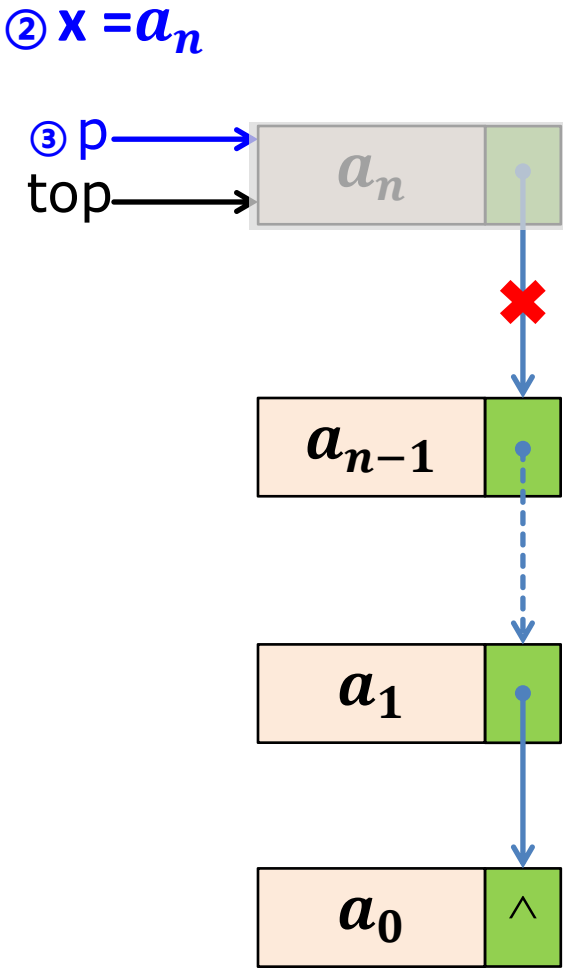
链栈的入栈

1. 构造一个值为x的新结点，并用指针p指向
2. 将新结点p的next指针域指向栈顶元素，插入新元素
3. 将原来的栈顶指针指向新结点

将值为x的结点压入链栈

```
Status Push (LinkStack &S, SELemType x) {
    p = new StackNode;          // 1.1 构造一个新结点p
    p->data = x;                // 1.2 将新结点数据域置为x
    p->next = top;              // 2. 新结点s指向栈顶
    top = p;                    // 3. 原来的栈顶指向新结点
    return OK;
}
```

栈的链式存储实现



链栈的出栈

1. 判断栈是否为空，若为空则返回Error
2. 将栈顶元素赋值给x
3. 临时保存栈顶元素，以便释放
4. 将栈顶指针top指向下一个元素
5. 释放原栈顶元素的空间

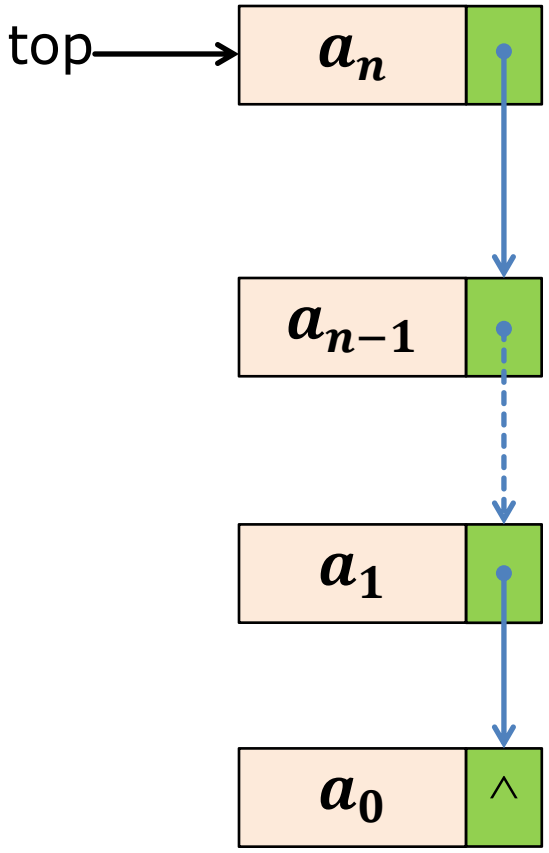
将链栈顶部的元素进行出栈操作

```
Status Pop (LinkStack &S, SELemType x) {
    if (top == NULL)
        return Error;
    x = top->data;
    p = top;
    top = top->next;
    delete top;
    return OK;
}
```

栈的链式存储实现

在链栈中取栈顶元素

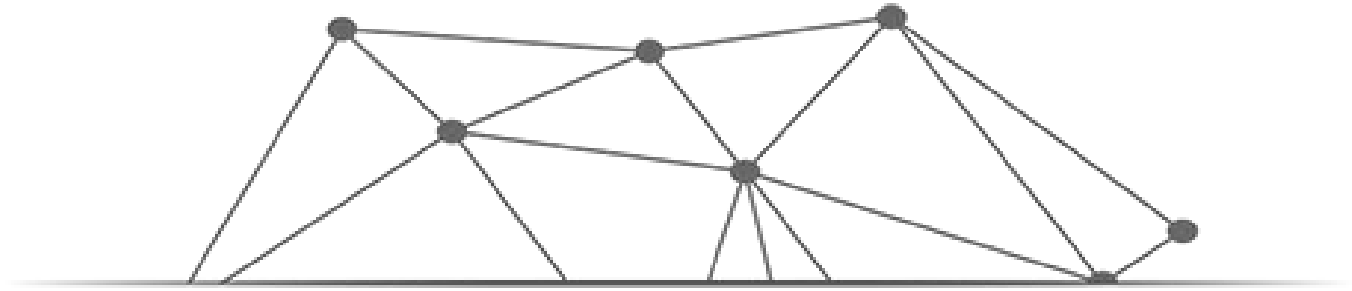
② x = a_n



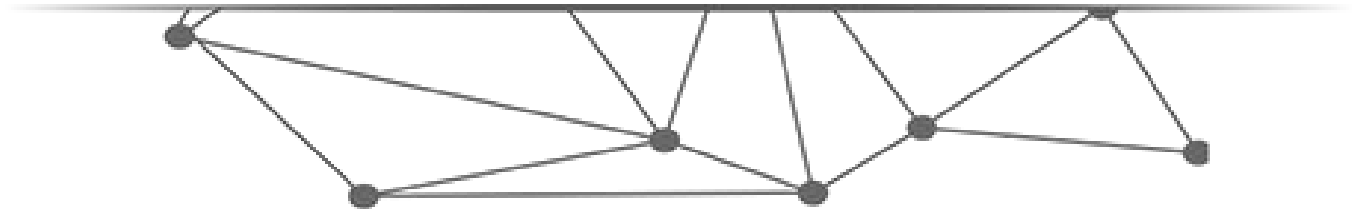
1. 判断栈是否为空，若为空则返回Error
2. 返回栈顶元素

获取堆栈顶部的元素，并赋值给x

```
SElemType Pop (LinkStack &S, SELemType x) {
    if (top == NULL)
        return Error;
    else
        x =  top->data;
    return OK;
}
```



课堂互动 7.3



第07讲 堆栈

小　结

- 栈是限定仅在表尾（栈顶）进行插入或删除的线性表
- 栈与线性表相同，数据元素间具有1对1的关系
- 具有后进先出、先进后出的特点
- 栈有两种存储表示，分别是顺序栈（顺序结构）和链栈（链式结构）
- 栈最主要的操作是进站和出栈，在进行进站和出栈时注意判满和判空
- 在顺序栈中，共享栈可以互相调剂两个栈的空间，灵活性强；只有在整个存储空间被占满时，才会发生上溢现象
- 链栈使用指针连接栈结点，可以同时实现多个栈共享存储空间，且不存在栈满上溢的问题。

欧老师的联系方式

读万卷书 行万里路 只为最好的修炼

QQ：14777591 (宇宙骑士)

Email：ouxinyu@alumni.hust.edu.cn

Website: http://ouxinyu.cn

Tel：18687840023

地址：安宁校区 诚远楼201

南院 智能应用研究院A306-2