教学内容---第三章

DS/SS

1. 绪论
2. 线性表
3. 栈、队列和串
4. 数组
5. 广义表
6. 树和二叉树
7. 图
8. 动态存储管理
9. 查找
10. 内部排序
11. 外部排序
12. 文件

3.1 栈的逻辑结构

DS/SS

栈、队列和串是特殊的线性表

- 栈和队列是操作受限的线性表
- 栈和队列的“操作受限”指操作位置受限
- 串的特殊性在于线性表中数据元素只能是字符
- 串一般以子串为操作单位
- 栈、队列和串具有一般线性表共性的特点
- 特殊的线性表反而应用面更宽

3.1 栈的逻辑结构（续）

DS/SS

栈的基本概念和术语

- 栈（Stack）：限定仅在表尾进行插入或删除操作的线性表。
- 栈顶（top）：插入或删除的表尾端。
- 栈底（bottom）：表头端。
- 空栈：空表。

栈的操作特点：后进先出（Last In First Out---LIFO）

3.1 栈的逻辑结构（续）

栈的抽象数据类型定义

ADT Stack {

数据对象：D={a_i | a_i属于ElemSet, i = 1,2, … ,n, n>=0}

数据关系：R1={< a_{i-1}, a_i >| a_{i-1}, a_i属于D, i =2,3, …, n}// a_1 为栈底， a_n 栈顶

基本操作：

InitStack(&S)

DestroyStack (&S)

StackEmpty(S)

ClearStack (&S)

StackLength(S)

GetTop(S,&e)

初始条件：S存在,非空；

操作结果：用e返回S的栈顶元素

Push(&S,e) ---------- 入栈（插入）

初始条件：S存在

操作结果：插入元素e为新的栈顶元素，长度加1

Pop(&S, &e) ---------- 出栈（删除）

初始条件：S存在；非空

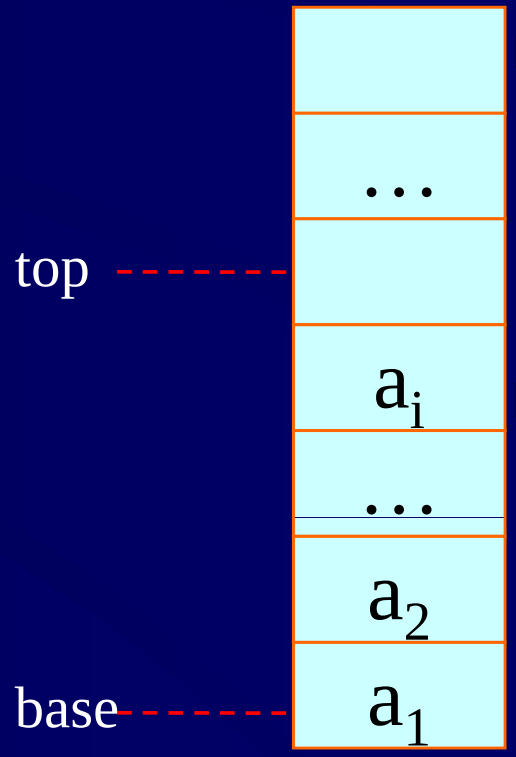
操作结果：删除S的栈顶元素，e返回值，长度减1

}ADT Stack

3.2 栈的存储结构

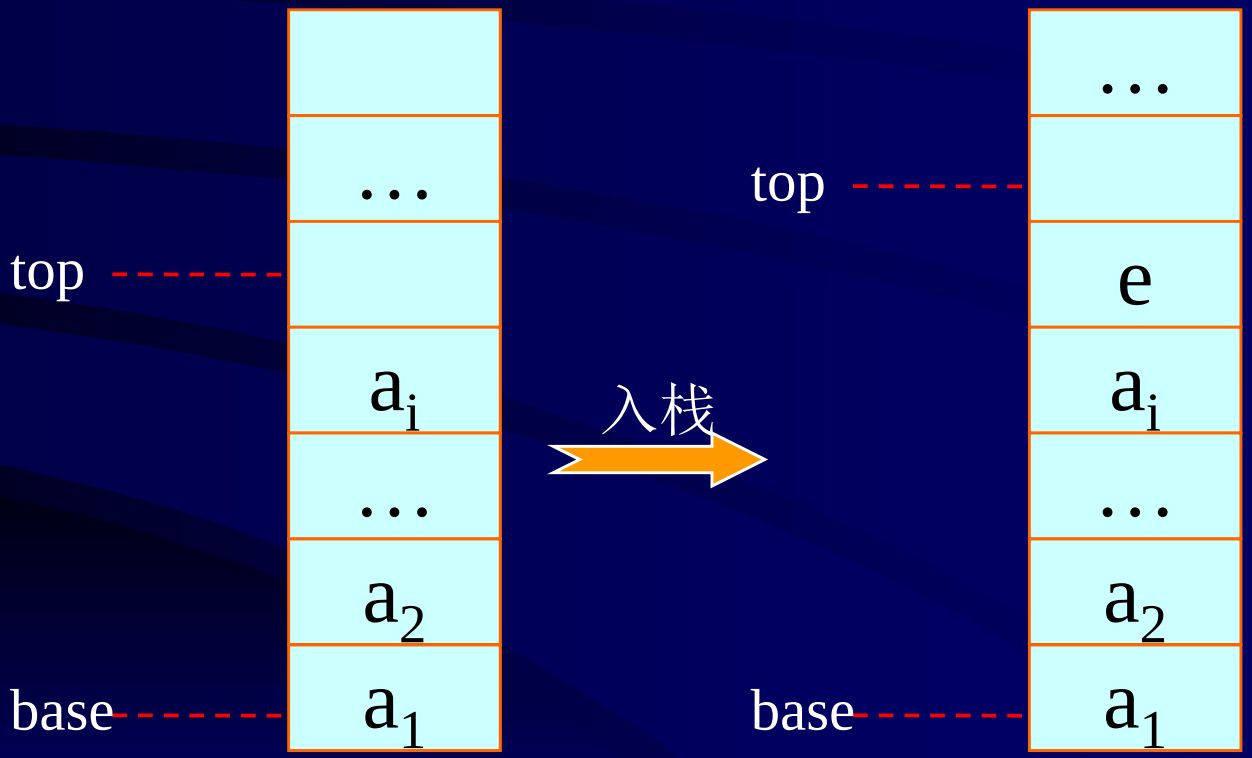
栈的顺序存储

```
//----------动态分配-----------
#define STACK_INIT_SIZE    100   //空间初始分配量
#define STACKINCREMENT 10    //空间分配增量
typedef  struct {
          sElemType          *base
          sElemType          *top
          int          stacksize          //当前分配的存储容量
} SqStack
```



3.2 栈的存储结构（续）

顺序栈上的入栈



3.2 栈的存储结构（续）

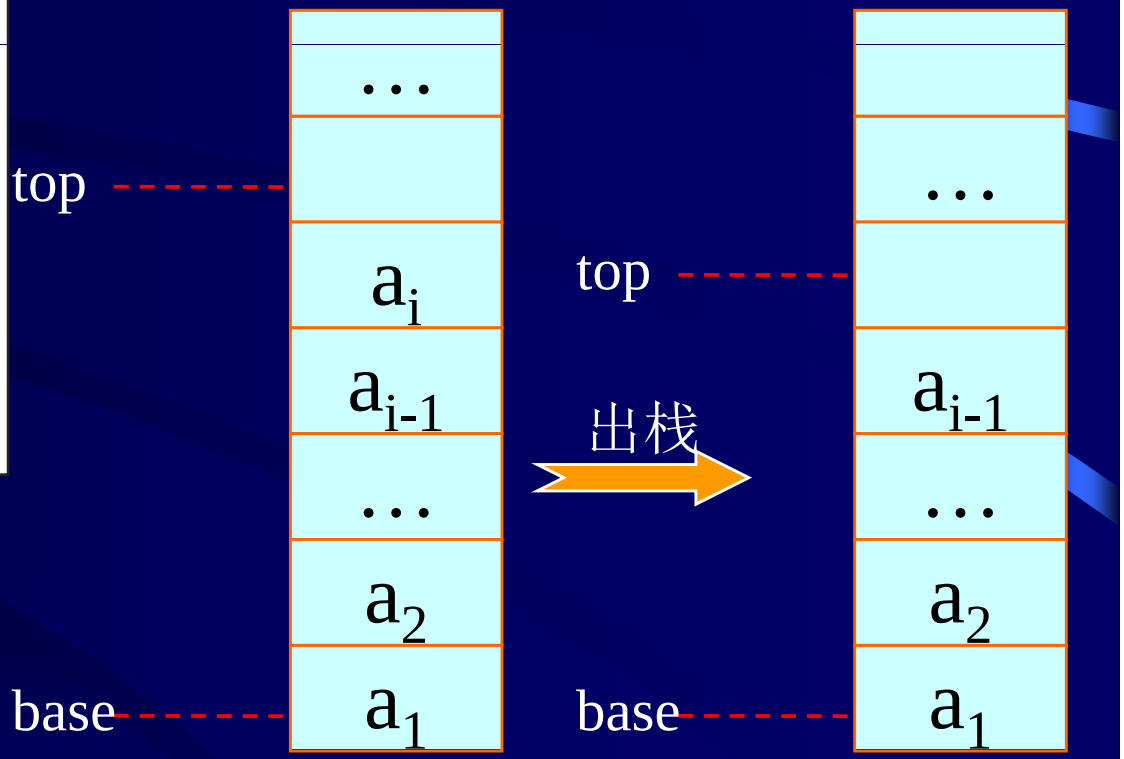
顺序栈上的入栈

```
Status Push(SqStack &S,SElemType e)
    if(S.top – S.base>= S.stacksize) {
        S.base = (SElemType *) realloc ( S.base,(S.stacksize
                +STACKINCERMENT) * sizeof (SElemType));
        if (!S.base) exit (OVERFLOW);  // 存储分配失败
        S.top = S.base + S.stacksize;
        S.stacksize += STACKINCERMENT;}
    *S.top++ = e;      return  OK;
} // Push
```

3.2 栈的存储结构（续）

顺序栈上的出栈

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

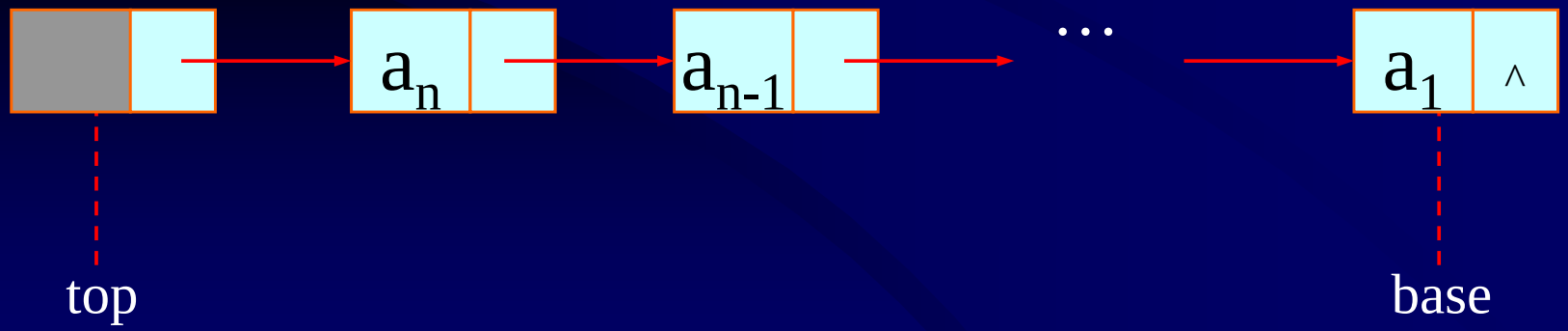


3.2 栈的存储结构（续）

栈的链式存储---链栈

```
typedef  struct {
         ElemType          data
         struct Lnode      *next
} Lnode,* StackPtr
```

```
typedef  struct {
         StackPtr          top
         StackPtr          base
} LinkStack
```



3.2 栈的存储结构（续）

DS/SS

链栈上的入栈与出栈

- 链栈上的入栈与出栈与单链表上元素插入删除操作类似
- 插入删除位置都在栈顶处，不花费查找时间
- 栈空的判断
- 与顺序栈相比，时间效率高

3.3 栈的应用

栈的应用领域

- 操作以“后进先出”为特
- 数值转换、括号匹配检验、行编辑程序、迷宫求解、表达式求值等
- 栈在程序调用中的作用
- 栈与递归

数值转换中主要利用栈实现按照从高位到低位输出转换后的数值。

括号匹配检验中利用栈来检验括号间是否以合法的嵌套层次成对出现。

迷宫求解中，利用栈保存从入口到当前位置的路径，当一趟“试探”失败后可以回退到当前位置（本质是在树中寻找合理路径）。

行编辑程序中利用栈回退刚刚输入到缓冲区中的字符或者字符串。

3.3 栈的应用（续）

DS/SS

函数调用中栈的应用

调用函数和被调用函数之间的链接和信息交换通过栈来进行

调用之前的三项工作：

*将实参、返回地址等信息传递给被调用函数保存；

*为被调用函数的局部变量分配存储区；

*将控制转移到被调函数入口。

调用之后的三项工作：

*保存被调用函数的执计算结果；

*释放被调函数的数据区；

*依照返回地址将控制转移到调用函数。

3.3 栈的应用（续）

DS/SS

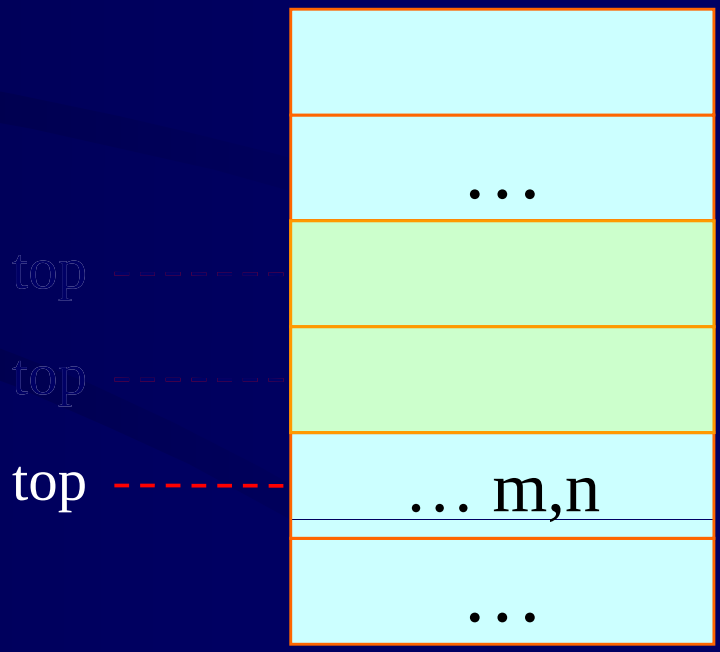
函数调用中栈的应用

当多个函数构成嵌套调用时，函数之间的信息传递和控制转移必须通过“栈”来实现，因为存在“后调用、先返回”的规则。

具体做法：系统对整个程序运行时所需的数据空间用栈进行管理。当存在函数调用时，就为它在栈顶分配一个存储区，放置着当前调用过程中的控制转移信息和数据信息；当函数运行完成，就释放这段存储区。当前正在运行的函数的数据区在栈顶。

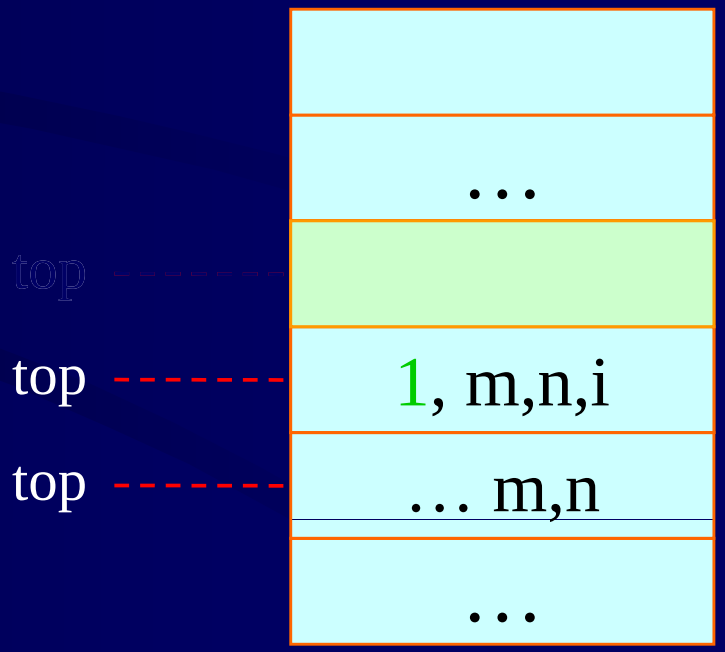
3.3 栈的应用（续）

```
int     first(int s, int t);
int     second(int d);
int     main(){
        int m, n;
        …
        first(m,n);
        1: …  }
int     first(int s, int t){
        int i;
        …
        second(i);
        2: …  }
int     second(int d){  int x,y;  …  }
```



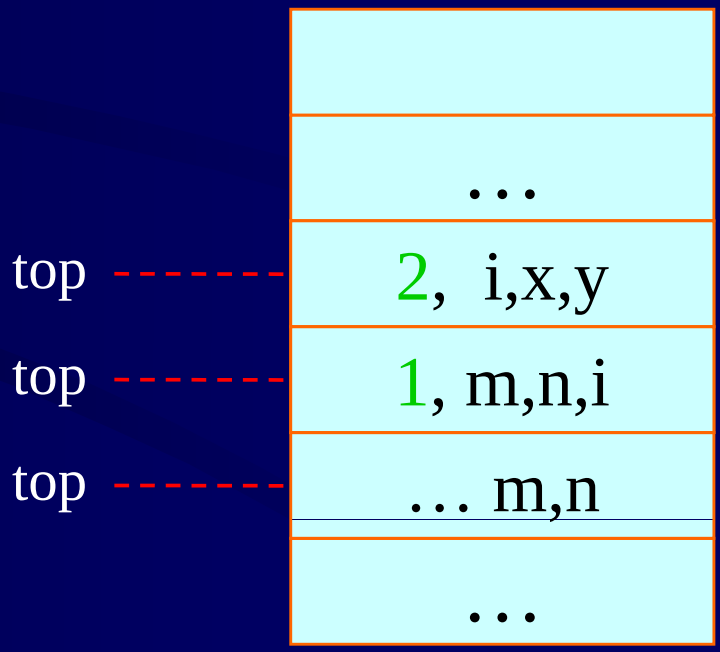
3.3 栈的应用（续）

```
int     first(int s, int t);
int     second(int d);
int     main(){
        int m, n;
        …
        first(m,n);
        1: … }
int     first(int s, int t){
        int i;
        …
        second(i);
        2: … }
int     second(int d){  int x,y;  … }
```



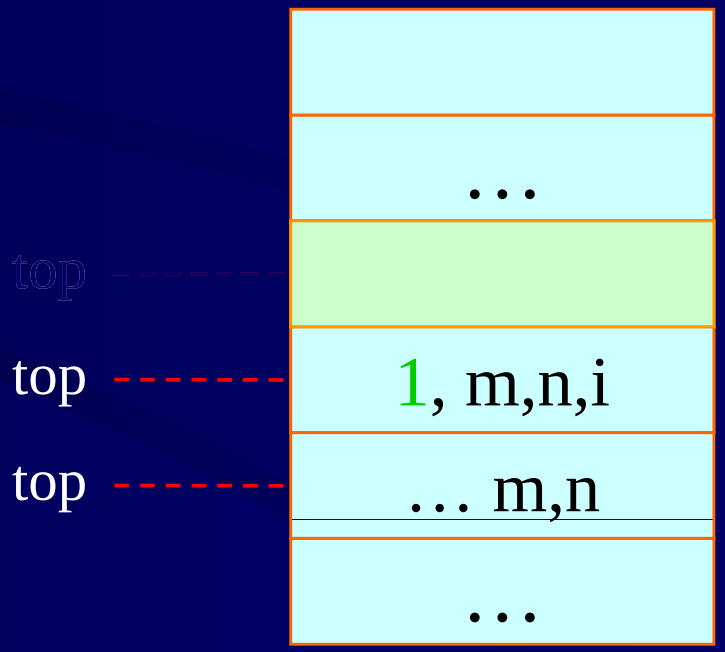
3.3 栈的应用（续）

```
int     first(int s, int t);
int     second(int d);
int     main(){
        int m, n;
        …
        first(m,n);
        1: …  }
int     first(int s, int t){
        int i;
        …
        second(i);
        2: …  }
int     second(int d){  int x,y;  …  }
```



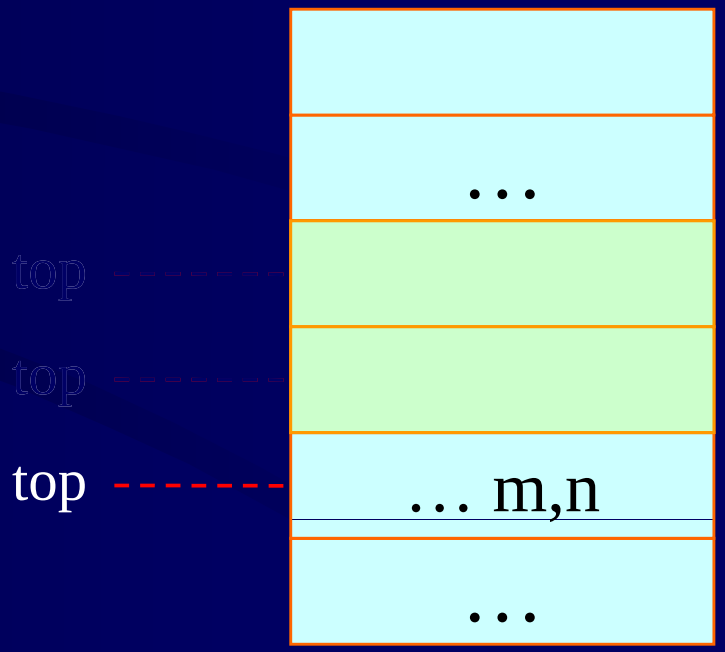
3.3 栈的应用（续）

```
int     first(int s, int t);
int     second(int d);
int     main(){
        int m, n;
        …
        first(m,n);
        1: …  }
int     first(int s, int t){
        int i;
        …
        second(i);
        2: …  }
int     second(int d){  int x,y;  …  }
```



3.3 栈的应用（续）

```
int     first(int s, int t);
int     second(int d);
int     main(){
        int m, n;
        …
        first(m,n);
        1: …  }
int     first(int s, int t){
        int i;
        …
        second(i);
        2: …  }
int     second(int d){  int x,y;  …  }
```



3.3 栈的应用（续）

DS/SS

递归中栈的应用

类似于函数嵌套调用，调用函数与被调函数是同一个函数。调用关系用“层次”来表达。设调用递归函数的主函数为第0层，则第一次调用进入第1层，依次，第i层递归调用本函数进入第i+1层。返回时，按照“后调用、先返回”的原则。

具体做法：系统设立一个“递归工作栈”作为递归函数运行期间使用的数据存储区；每一层递归所需要的控制转移信息和数据信息构成一个“工作记录”，包括：实参、局部变量以及上一层的返回地址。当前正在运行层的工作记录在栈顶，称为“活动记录”，指示活动记录的栈顶指针为“当前环境指针”。