读书不觉已春深

一寸光阴一寸金

第三章 栈与队列

主要内容

1. 栈
2. 队列
3. 栈与递归

操作受限的线性表

栈：后进先出

队列：先进先出

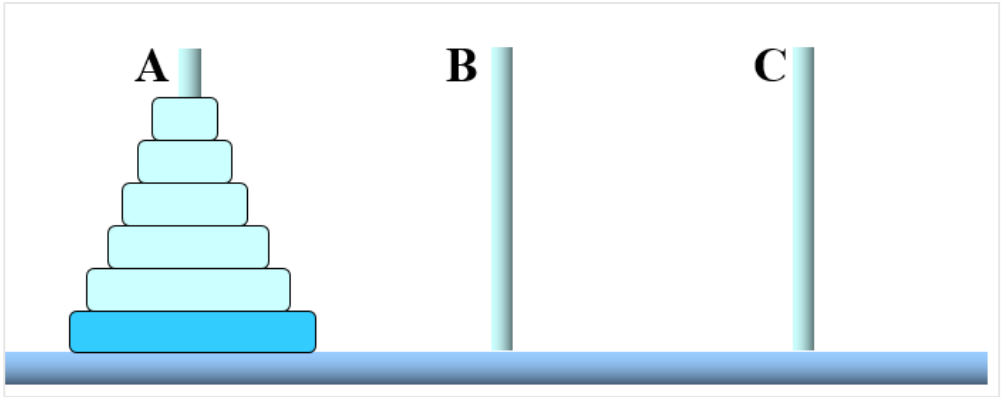
括号匹配

表达式求值

迷宫求解

排队问题

运动会赛事安排



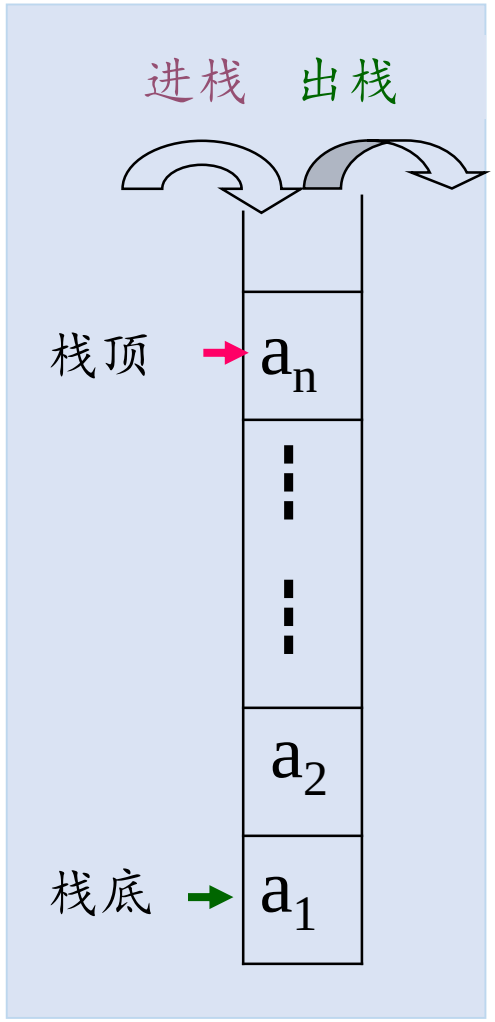
3.1　栈的基本概念和类型定义

栈(STACK)：一种限定性的数据结构，限定只能在表的**一端**进行**插入和删除**的线性表.

栈顶(TOP)：允许插入和删除的一端.

栈底(BOTTOM)：不允许插入和删除的一端

栈是后进先出表（LIFO）.



思考

如果进栈顺序为1，2，3，4，不限制出栈的时间，则下面哪一种出栈顺序不可能.

(1) 1, 2, 3, 4

(2) 2, 1, 4, 3

(3) 3, 1, 2, 4

(4) 4, 3, 2, 1

栈的抽象数据类型定义

ADT Stack {

数据对象：

D＝{ ai | ai ∈D, i=1,2,...,n, n≥0 }

数据关系：

R1＝{ <ai-1, ai >| ai-1, ai∈D, i=2,...,n }

约定an 端为栈顶，a1 端为栈底。

基本操作：

InitStack (*S)：设置一个空栈s。

Push(*S ,e)：入栈，在栈s中插入一个新的栈顶元素。

Pop(*S,*e) ：出栈函数，删除栈顶元素。

GetTop (S,*e) ：读取栈顶元素；若栈为空，返回空值。

StackEmpty (S) ： 判断s是否是空栈。

如果栈空返回1，非空返回0。

ClearEmpty(*S):将栈清空

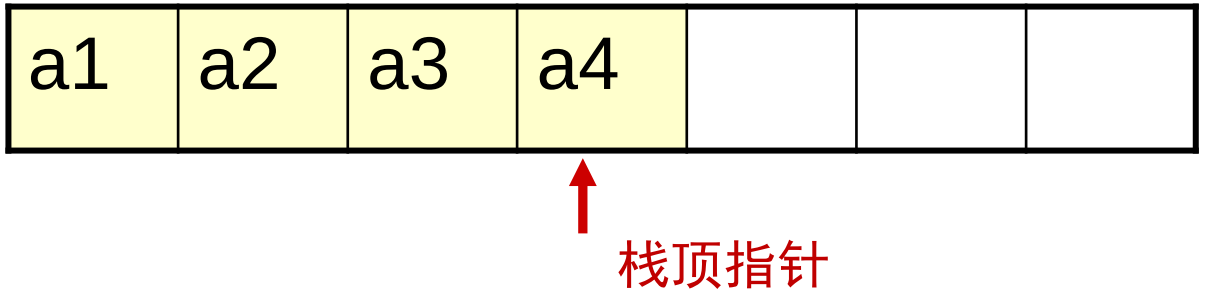
} ADT Stack

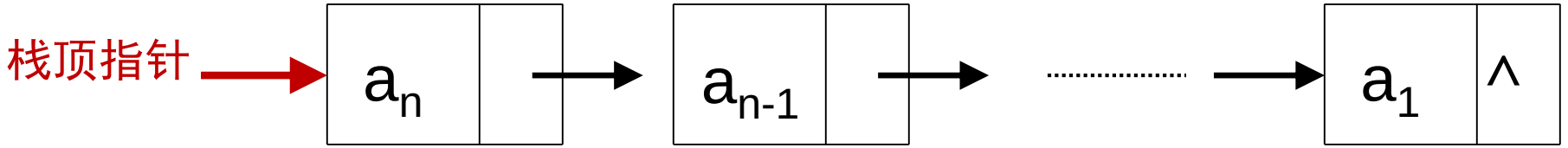
3.2 栈的实现

两种实现方法

顺序栈：顺序存储

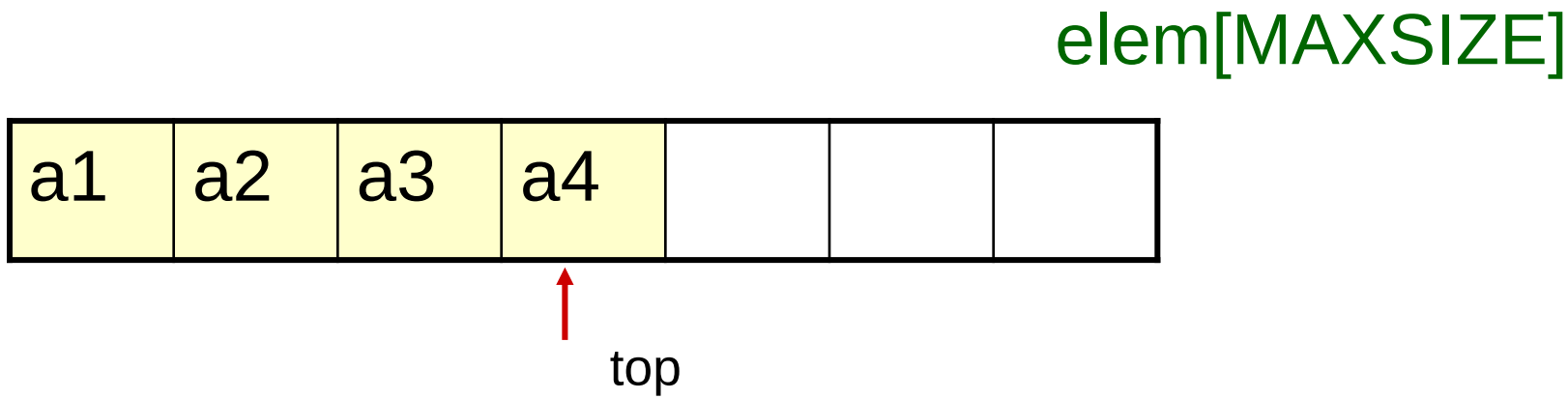
链栈：链式存储





3.2.1 顺序栈

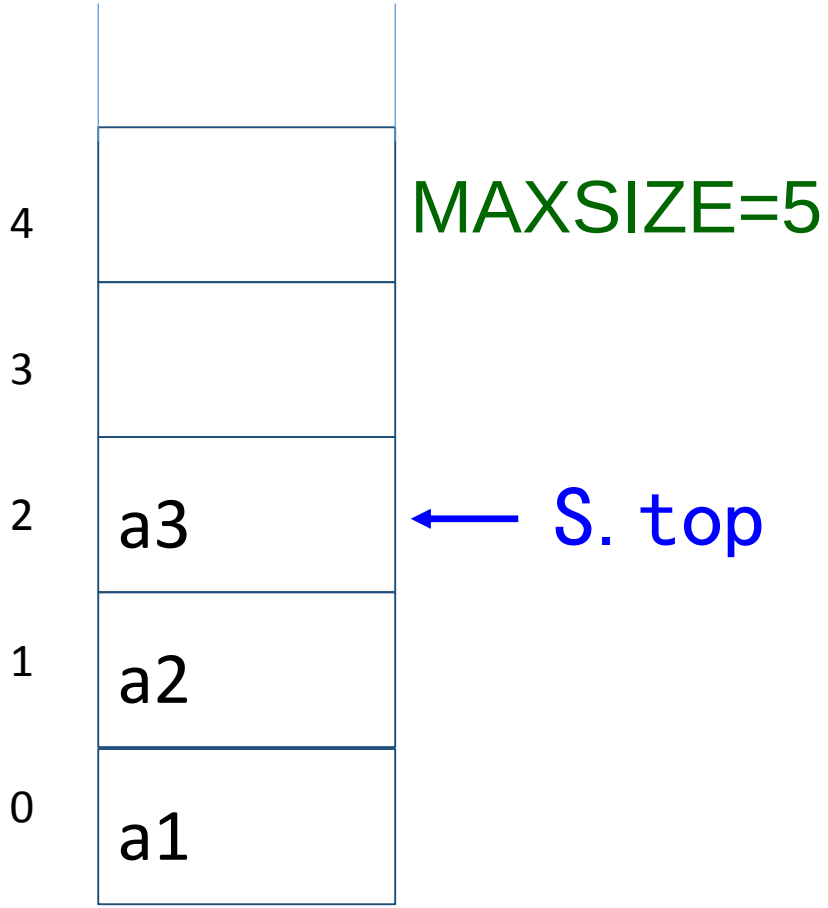
存储特点：利用一组地址连续的存储单元依次存放自栈底到栈顶的数据元素。c语言中可用数组来实现顺序栈. 设置一个指示操作一端的变量top，称之为栈顶指针.



顺序栈的 C 语言描述

```
# define  MAXSIZE   …
typedef  struct {
   ElemType   elem[MAXSIZE] ; // 数组空间
   int   top ;
 } SqStack ;
SqStack   S;
```

S.top的值　可以规定：指向栈顶元素



(1) S.top == -1： 空栈；

(2) S.top == MAXSIZE-1为栈满

(3) S.top>=0：S. elem[0]为栈底元素，S.elem[S.top]为栈顶元素；

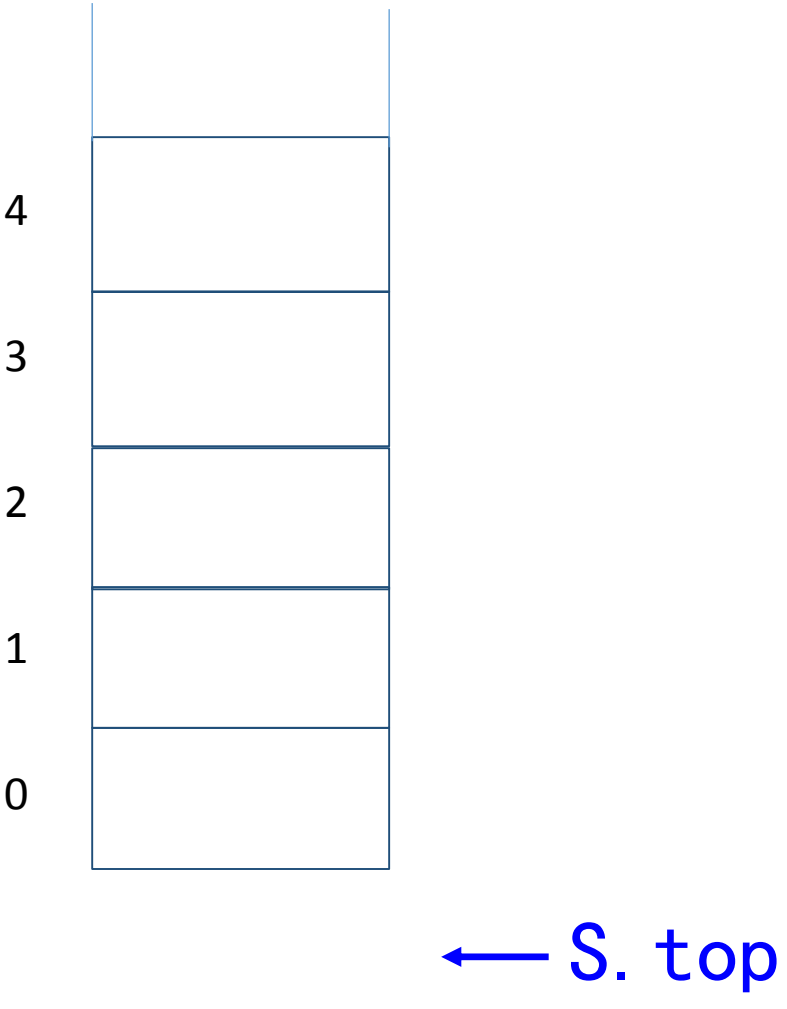
思考：如果规定S. top指向栈顶元素的后一个位置，那图示状态，s. top的值应为多少？空栈和栈满的条件是什么？

顺序栈基本操作的实现

(1) 初始化建栈

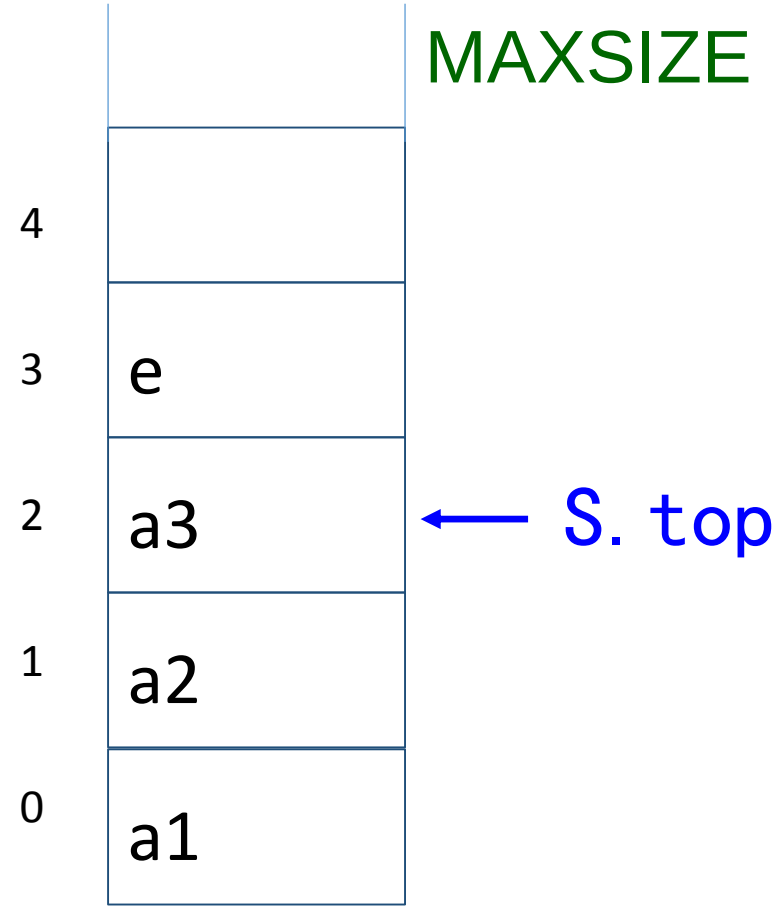
```
status InitStack (SqStack *S)
{
  if(S == NULL)
      return 0;
  S->top = -1;
  return 1;
}
```

思考：假如主函数调用该函数，定义了变量SqStack s;调用形式应该是什么？



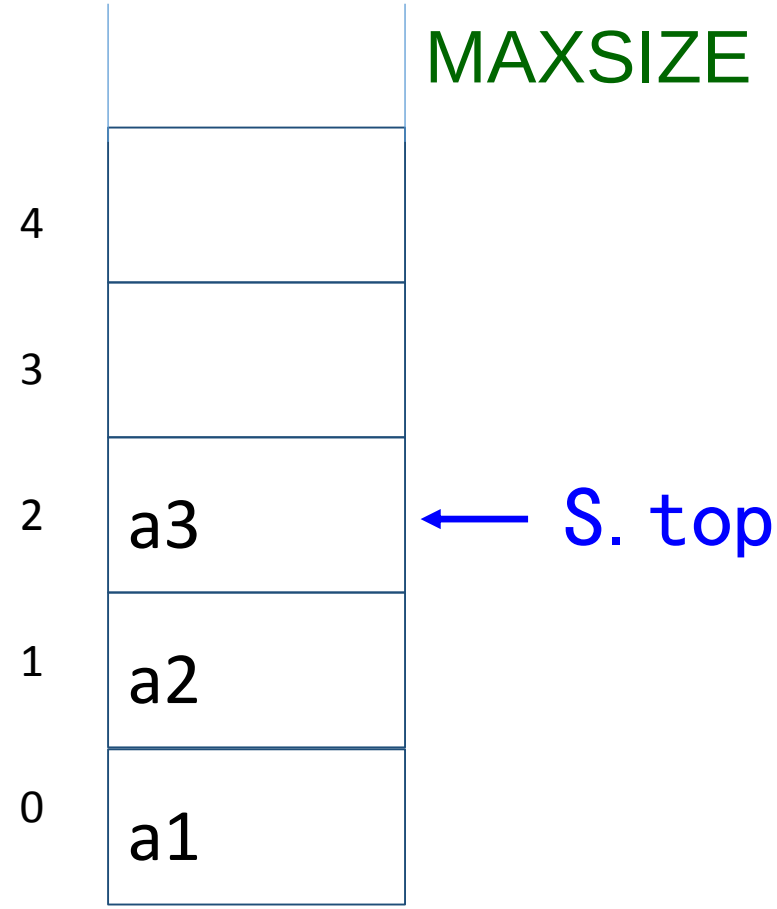
(2) 进栈

```
int  Push(SqStack *S，ElemType  e)
{
   if(S==NULL) return 0;
  if(S->top == MAXSIZE-1）
        return   0；//栈满
  else  {  S->top++;
           S->elem[S->top]= e;
           return  1；}
}
```



(3) 出栈

```
status Pop(SqStack  *S, ElemType *e)
 {
      if(S==NULL) return 0;
      if (S->top ==  -1) return 0;
      *e=  S->elem[S->top];
       S->top--
      return 1;
}
```



练习：读栈顶元素、判断栈空、判断栈满、清空栈如何实现？

```
int ClearStack (SqStack * S)
 {
    return S->top = -1;
}
```

```
int StackEmpty (SqStack  S)
 {
   return S.top == -1;
 }
```

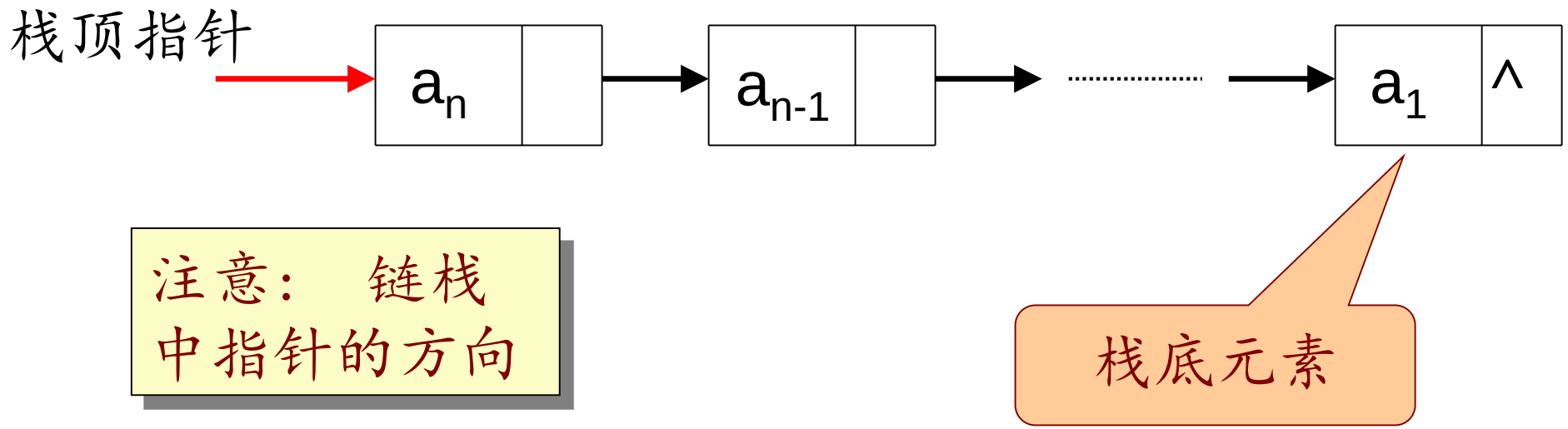
```
status GetTop(SqStack  S, ElemType *e)
 {
     if (S.top ==  -1) return 0；
     *e=  S.elem[S.top]；
    return 1;  }
```

```
int StackFull(SqStack  S)
 {
   return S.top == MAXSIZE-1;
}
```

3.2.2 链栈

链栈：即不带头结点的单链表，限定只能在表头插入和删除

存储特点是什么呢？ 同链表



链栈数据结构定义

```
typedef  struct  StackNode{
    ElemType  data;
    struct stacknode *next;
}StackNode,*LinkStack;

StackNode S;
LinkStack PS;
```

操作：

初始化

判断是否为空栈

读栈顶元素

入栈

出栈

清空链栈

(1) 初始化空栈

方法1：

```
 LinkStack InitStack( )
   {
     LinkStack LS;
     LS=NULL;
      return LS;
}
```

方法2：

```
void InitStack(LinkStack *LS)
 {
     *LS=NULL;
}
```

(2)判栈是否为空

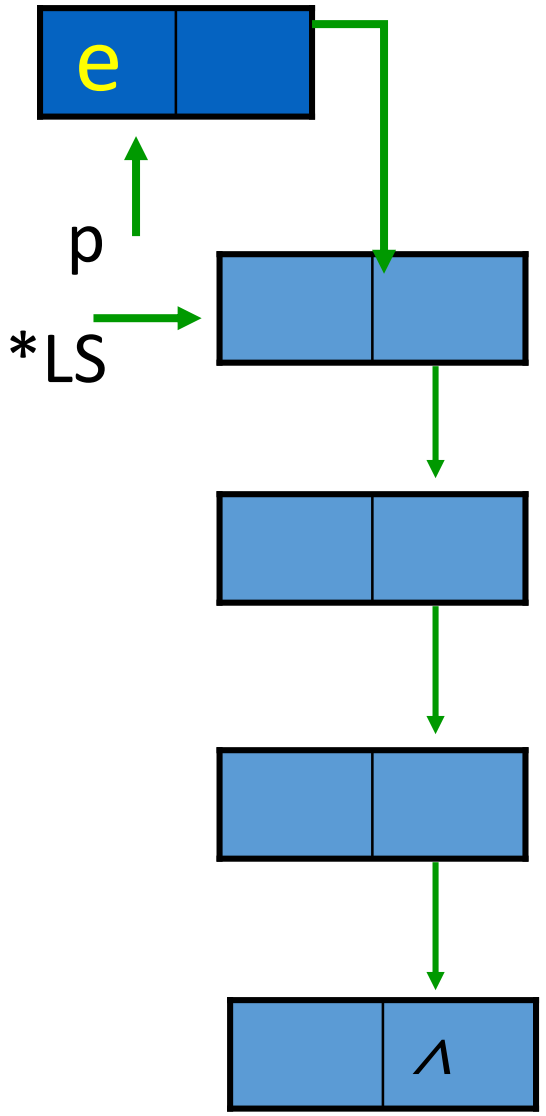
```
int stackempty(LinkStack LS)
 {
     return LS==null;
  }
```

(3) 读栈顶元素

```
status GetTop(LinkStack LS, ElemType *e)
{
    if(LS == NULL)  return 0;
    *e=LS->data;
   return 1;
}
```

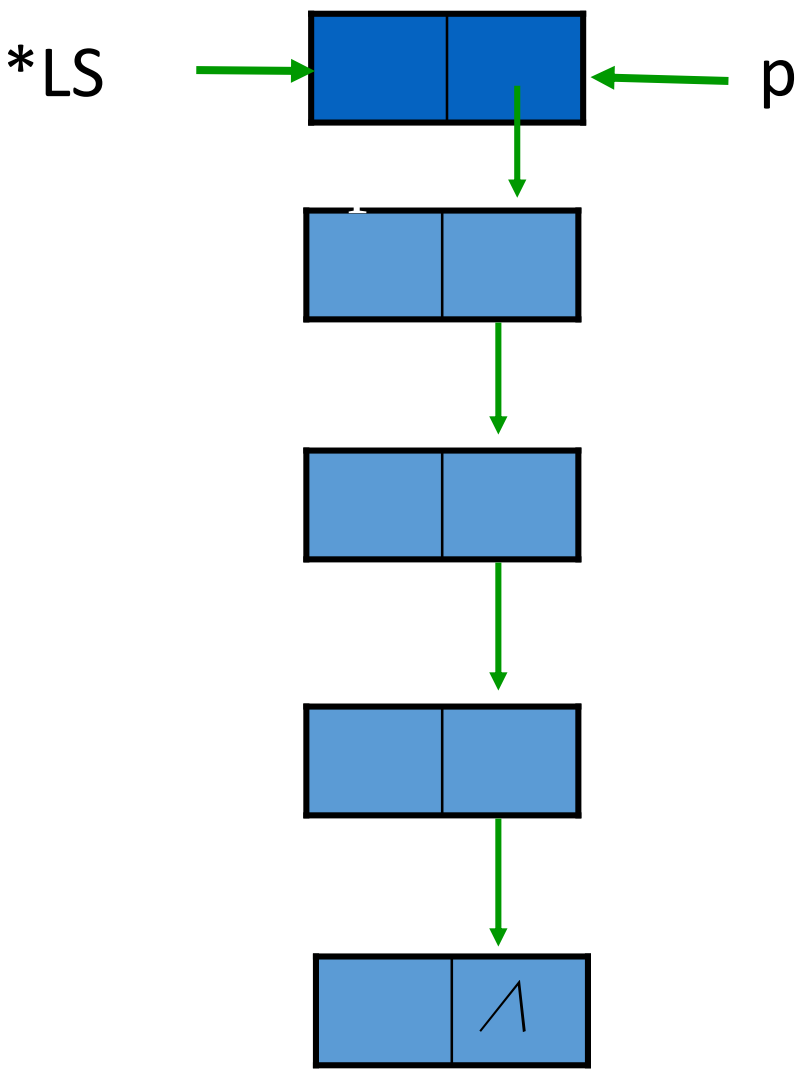
(4) 进栈

```
void Push(LinkStack *LS,ElemType e)
{
    LinkStack p;
    p=(LinkStack)malloc(sizeof(StackNode));
    p->data=e;  p->next=*LS;  *LS=p;
}
```



(5) 退栈（出栈）

```
status Pop(LinkStack *LS,ElemType *e){
  LinkStack p;
  if(*LS == NULL)  return 0;
   *e = (*LS)->data;
     p=*LS; *LS=(*LS)->next;
     free(p);
      return 1;
}
```



(6) 清除链栈为空

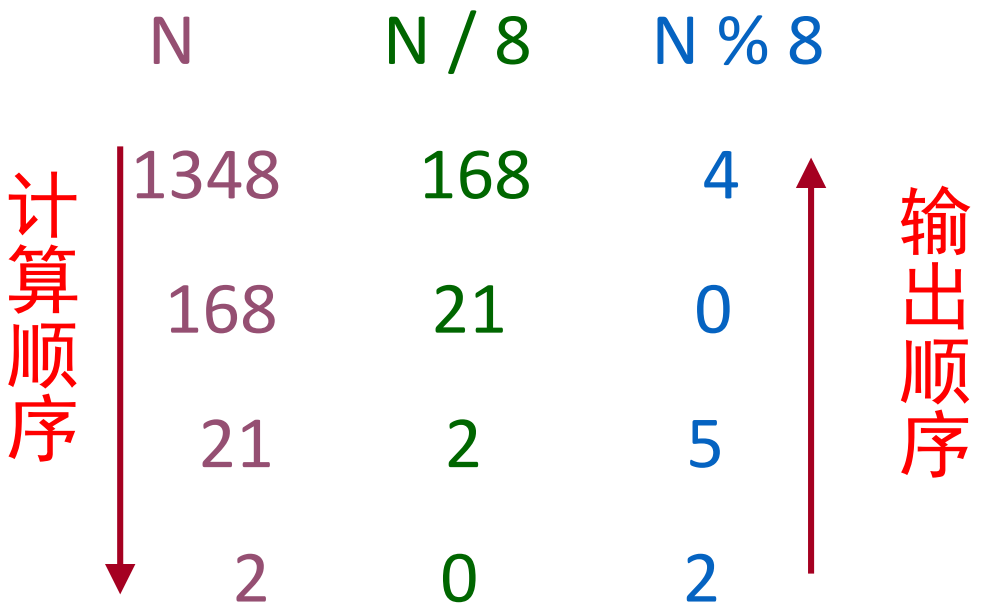
```
void ClearStack(LinkStack *LS)
{ StackNode *p;
  while(*LS )
  {   p = *LS;
      (*LS) = (*LS)->next;
       free(p);
  }
  *LS = NULL;}
```

3.3 栈的应用举例

- 数制转换
- 括号匹配的检验
- 迷宫求解
- 表达式求值

3.3.1数制转换

如：$(1348)_{10} = (2504)_8$，运算过程如下：



思考：转换之后的数保存在一个什么类型的变量里呢?

```
void conversion (int N, int  d, Stack *S) {
/* 将整数N转换成d进制*/
    while (N) {
        Push(S，N % d)；
        N = N/d；
     }
} // conversion
```

3.3.2 括号匹配的检验

假设有两个包含括号的表达式：

1+ a［3］*（k+2）

（1+x）*(3+a[5]/2+a[7]

表达式正确吗？

［（）］或（）［］

均为正确的格式。

［（］）或（［（）］或(])

均为不正确的格式。

检验括号是否匹配的方法可用“期待的急迫程度”来描述。

1+ a［3］*（k+2）

（1+x）*(3+a[5]/2+a[7]

检验括号是否匹配的方法可用“期待的急迫程度”来描述。

分析当前字符可能出现的情况

到来的“不是括号”；考虑下一个字符

到来的是左括号，暂时记下来；

到来的是右括弧，进行比较：

{不是所“期待”的；不匹配

是期待的，匹配；}

循环判断每一个字符

直到字符串结束

思考：用什么记录左括号

算法过程

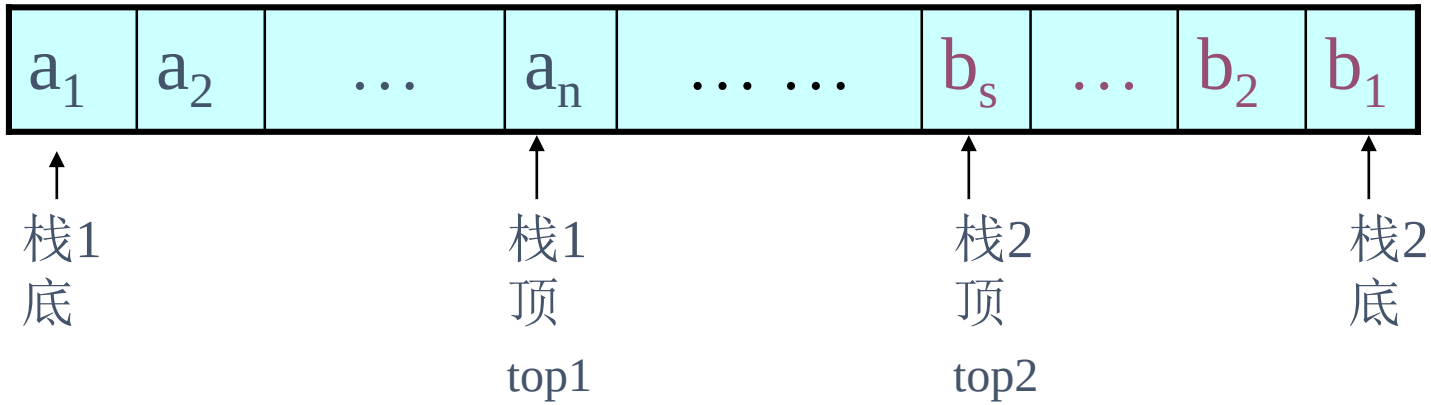
1. 当前字符不是括号，不做处理；
2. 当前字符是左括号，则进栈；
3. 当前字符是右括号：若栈空，则表明该“右括号”多余，不匹配，结束；否则和栈顶元素比较，若不匹配，结束；若和栈顶匹配，则栈顶元素出栈；
4. 当前字符是结束符时，若栈空，则匹配正确；否则表明“左括号”多余，不匹配，结束。
5. 读取下一字符，重复以上步骤

请同学们根据算法思路，自己完成代码编写以及将保存左括号的栈的变化状态画出来。

自学内容：双栈结构

为同时出现的两个栈共同开辟一段连续的存储空间，两个栈的栈底设于存储空间的两端。

优点：两栈互补余缺充分利用存储空间。



双栈的存储结构

```
typedef  struct {
   ElemType  *elem；  //待分配存储空间
   int  top1；  //栈顶指针
   int  top2;  //栈顶指针
   } DStack；
```

双栈操作

1）栈初始化 InitDStack(DStack *ds)

2）入栈　　PushDStack(DStack *ds,ElemType　elem,int iFlag)

3）出栈　　OutDStack(DStack *ds,ElemType *elem,int iFlag)

4）判栈空　IsEmpty(Dstack ds,int iflag)

5）判栈满　IsFull(DStack ds)

6）销毁栈　DestroyDStack(Dstack *ds)

作业

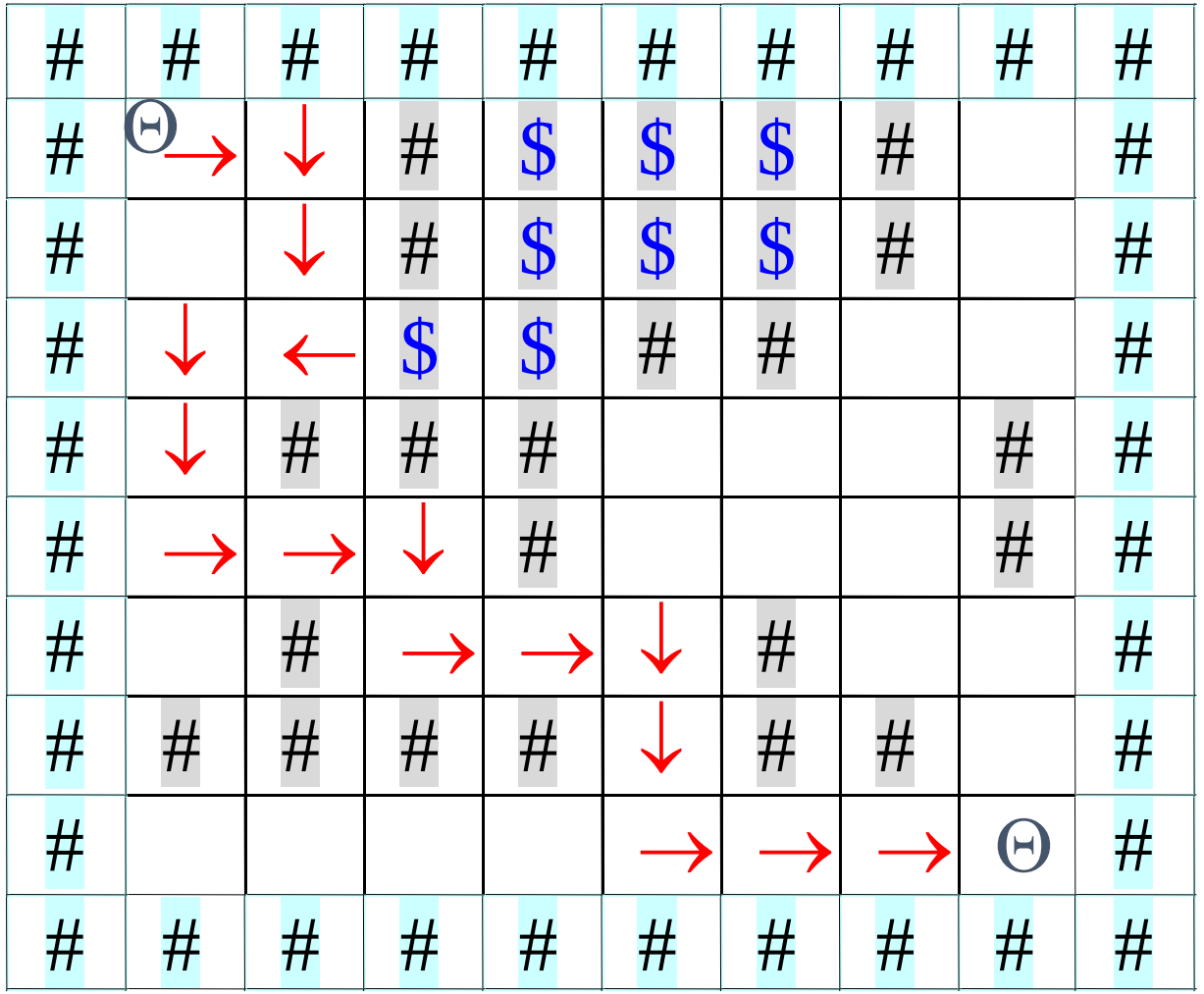
1　括号匹配

2　双栈存储奇数偶数

3　数制转换

3.3.3 迷宫求解

通常用“回溯试探方法”求解

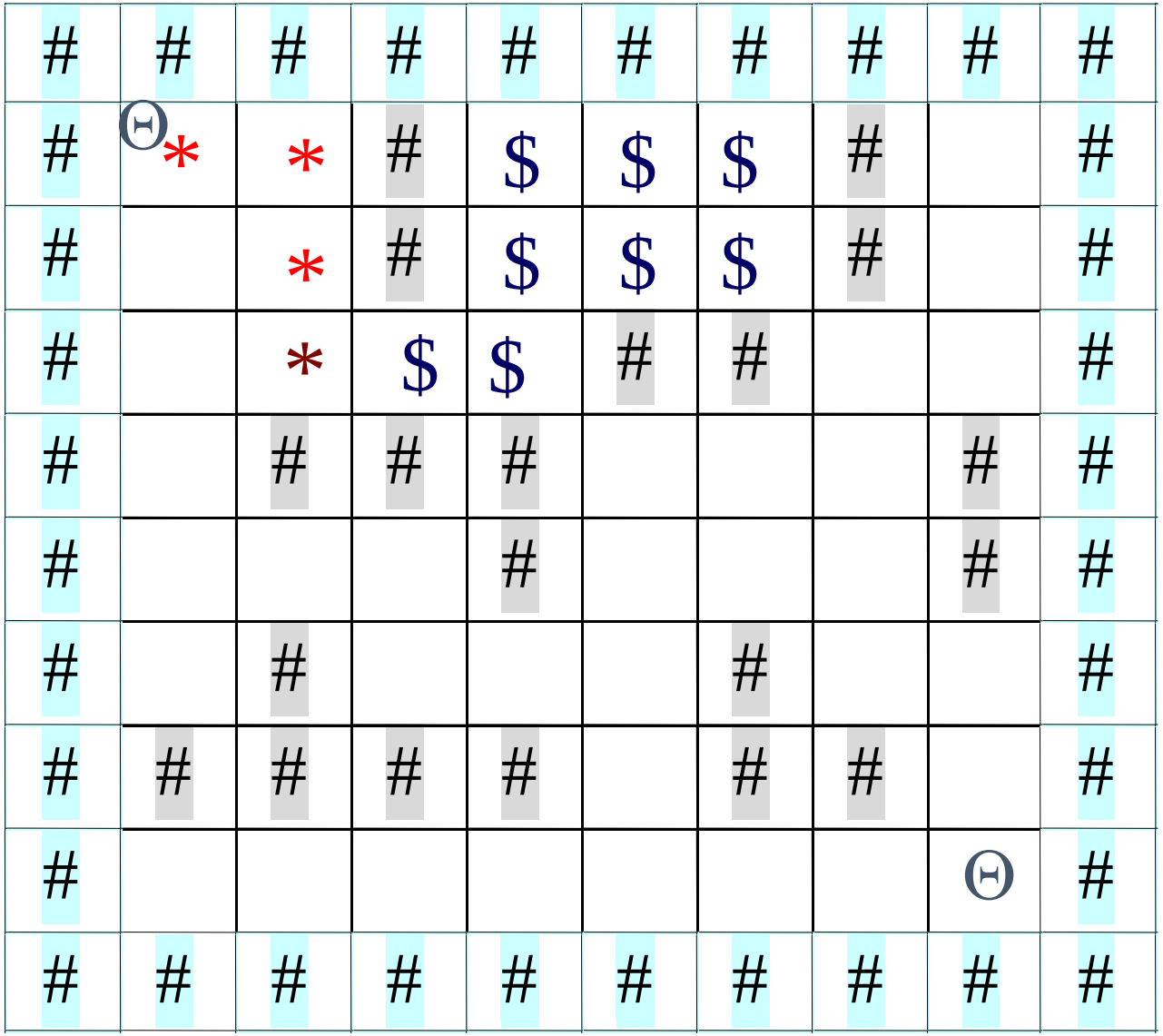


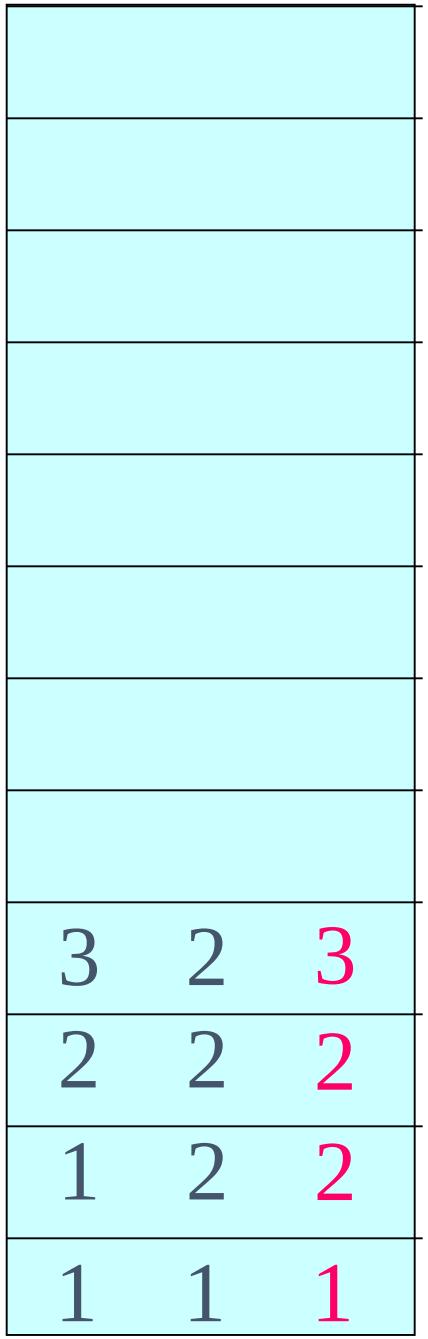
求迷宫路径算法的基本思想

若当前位置“可通”，则纳入路径，继续(向东)前进；

若当前位置“不可通”，则后退，换方向继续探索；

若四周“均无通路”，则将当前位置从路径中删除出去。


#
Θ
*
$


3 2 3
2 2 2
1 2 2
1 1 1

求迷宫中一条从入口到出口的路径的算法

设定当前位置的初值为入口位置；

do {

若当前位置可通，

则{将当前位置插入栈顶；

若该位置是出口位置，则算法结束；

否则切换当前位置的东邻方块为新的当前位置；}

否则 {……}

} while（栈不空）；

若栈不空且栈顶位置尚有其他方向未被探索，则设定新的当前位置为：沿顺时针方向旋转找到的栈顶位置的下一相邻块；

若栈不空但栈顶位置的四周均不可通，

则｛

删去栈顶位置；//回溯

若栈不空，则重新测试新的栈顶位置，直至找到一个可通的相邻块或出栈至栈空；

｝

若栈空，则表明迷宫没有通路。