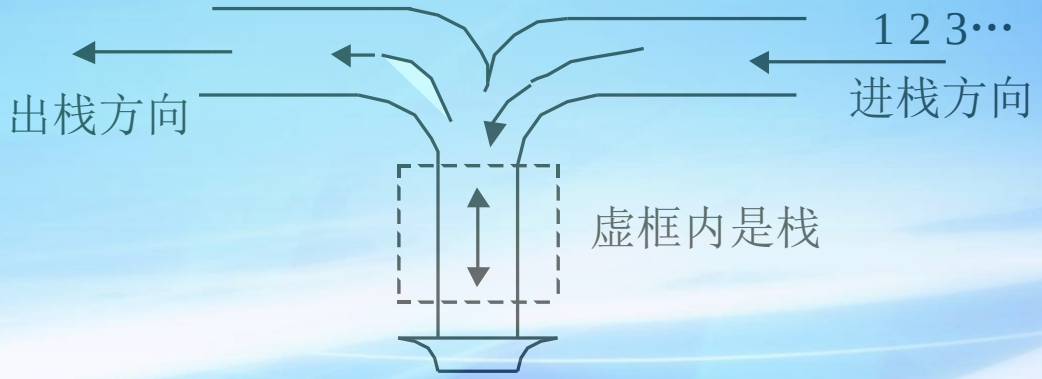
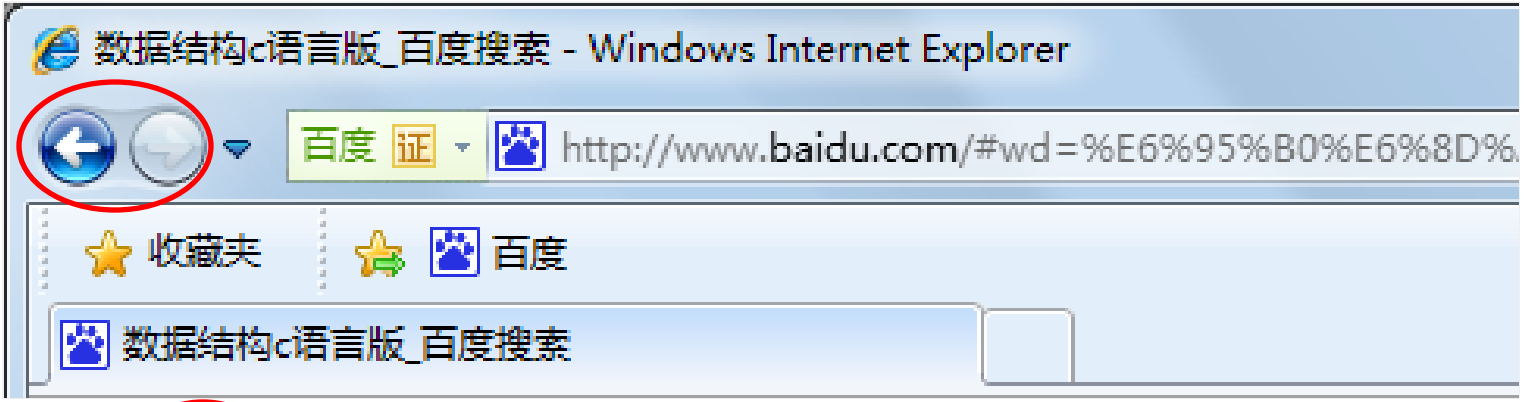
栈

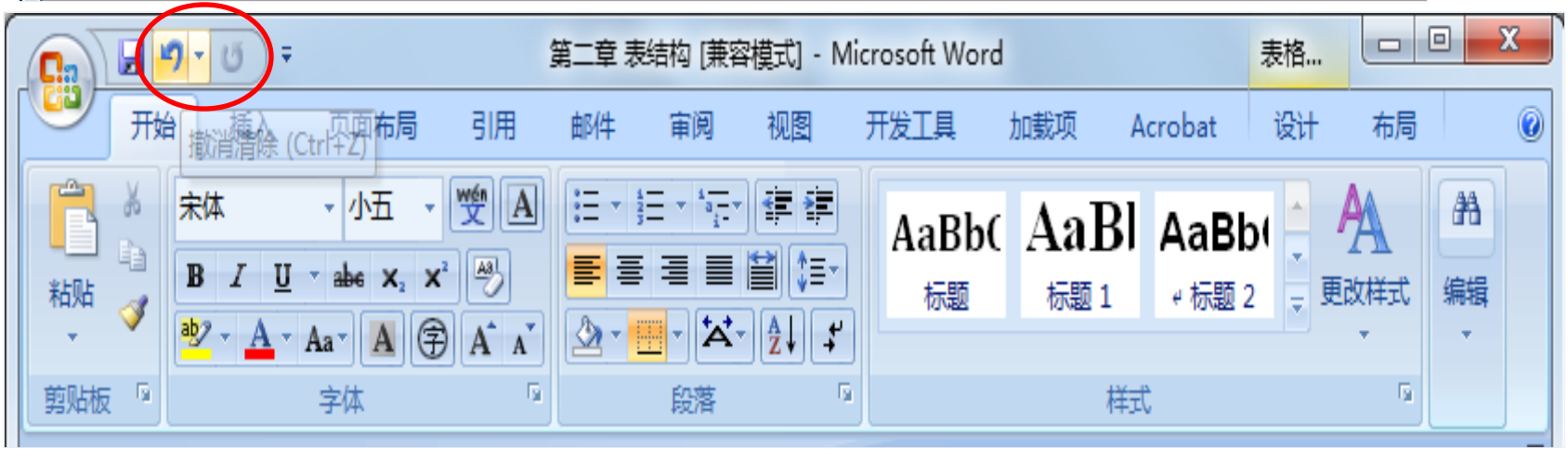
《数据结构》

网络工程教研中心　陈卫卫



应用举例


数据结构c语言版_百度搜索 - Windows Internet Explorer
百度
http://www.baidu.com/#wd=%E6%95%B0%E6%8D%
收藏夹
百度
数据结构c语言版_百度搜索

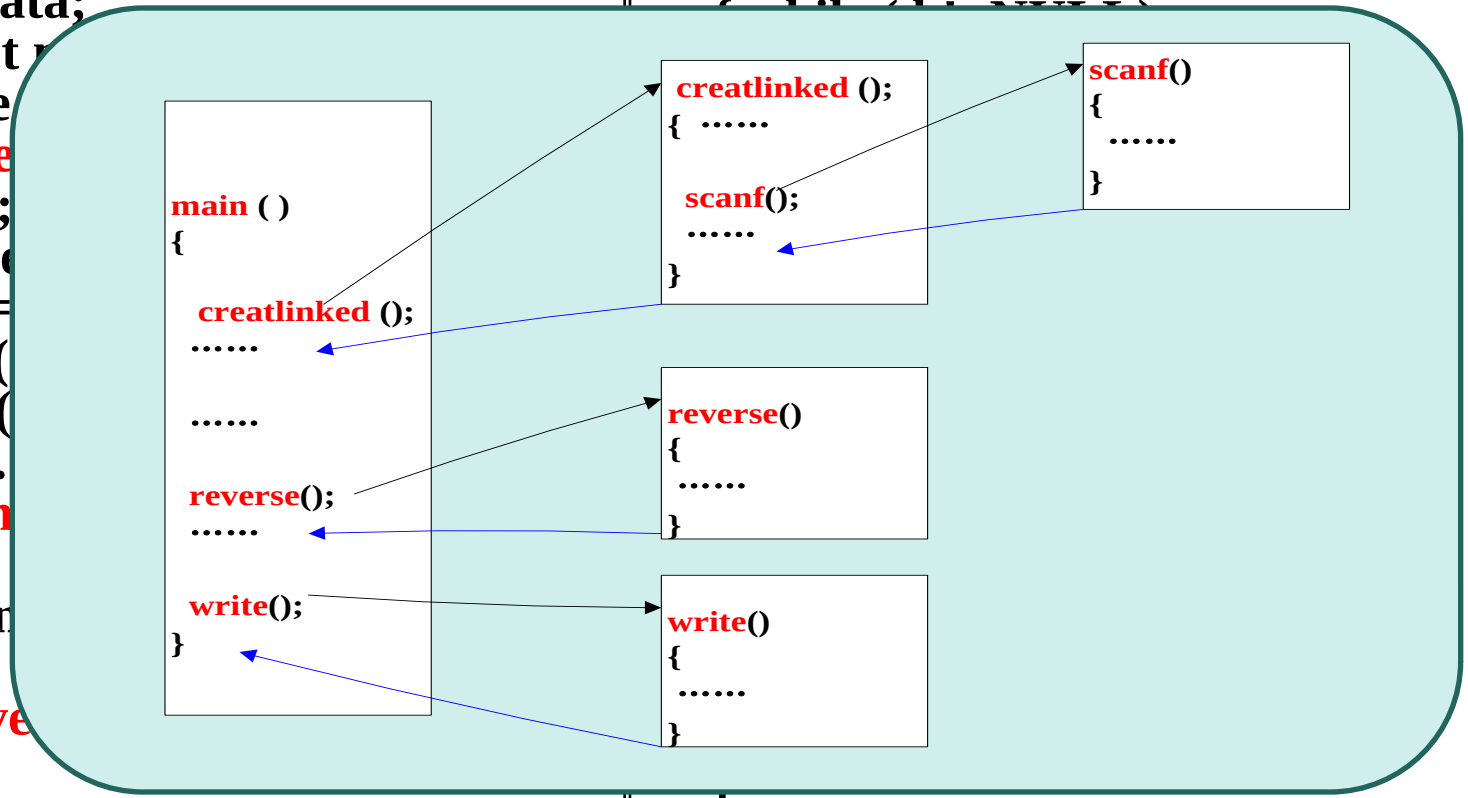

第二章 表结构 [兼容模式] - Microsoft Word
表格...
开始
插入
页面布局
引用
邮件
审阅
视图
开发工具
加载项
Acrobat
设计
布局
撤消清除 (Ctrl+Z)
宋体
小五
粘贴
剪贴板
字体
段落
样式
更改样式
编辑
AaBbC
标题
AaBl
标题 1
AaBbC
标题 2

```
#include  <stdio.h>
typedef   struct node
 { int data;
   struct node * next;
 }snode ,*ptr;
ptr  creatlinked( )
{ int  x;
  ptr  head, p;
  head=NULL;
  scanf("%d", &x);
  while(x!=0)
  {  ……;
     scanf("%d", &x);
  }
  return(head);
}
ptr  reverse(ptr h)
{  ……
  return  g;  }
```

```
void  write (ptr h)
{ while ( h!=NULL)
  {  printf("%5d",h->data);
     h=h->next;
   }
    printf("\n");
}
void main ( )
{ ptr head;
   head= creatlinked ();
   write(head);
   head=reverse(head);
   write(head);
}
```

```
#include <stdio.h>
typedef  struct node
 { int data;
   struct n
 }snode
ptr  cre
{ int  x;
  ptr  he
  head=
  scanf(
  while(
  { ……
    scan
  }
  return
}
ptr  reve
{ ……
  return  g;  }
```

```
void  write (ptr h)
```



栈的基本操作

学习目标和要求

1.能够准确描述栈的特点；

2.能够写出顺序栈的入栈和出栈算法；

3.能够写出链栈的入栈和出栈算法；

1. 栈的概念

❖ （1）栈的术语和图示

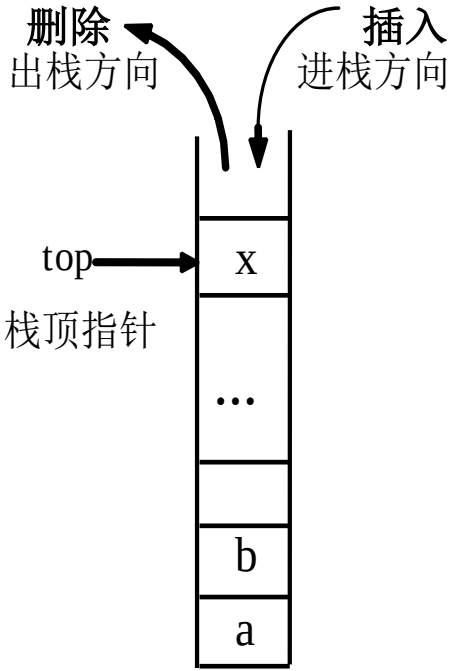
只允许在同一个端点处进行插入或删除的表结构称为**栈**（stack）

栈顶（top）

栈底（bottom）

进栈（push）

出栈（pop）

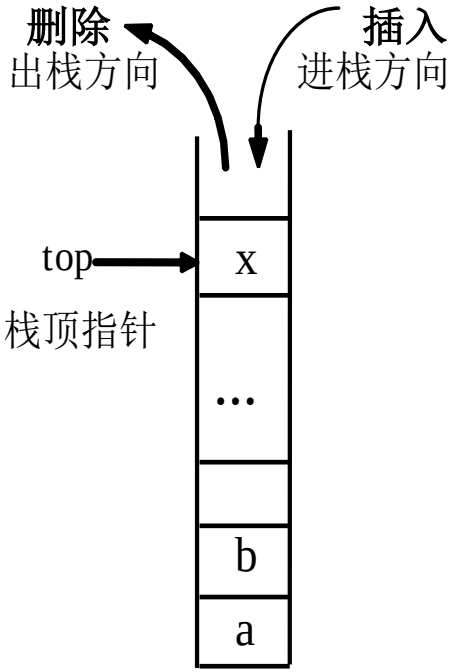


2. 栈的特点

栈结构，像开口向上的圆桶
结点像编了号的木块

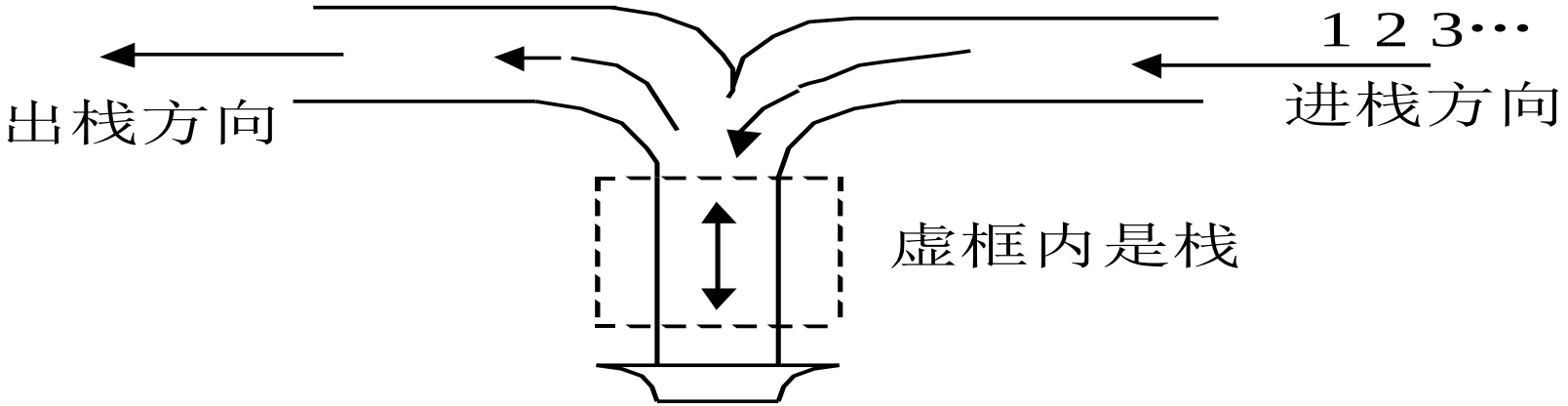
先放的在下边
后放的在上边
后进先出表

LIFO表

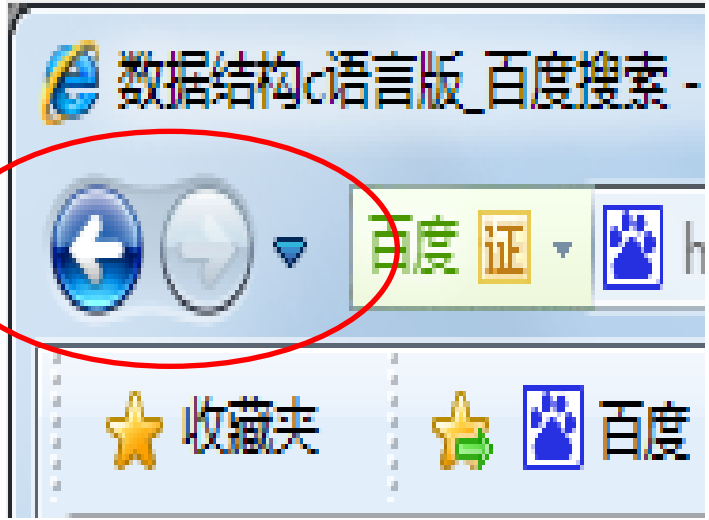


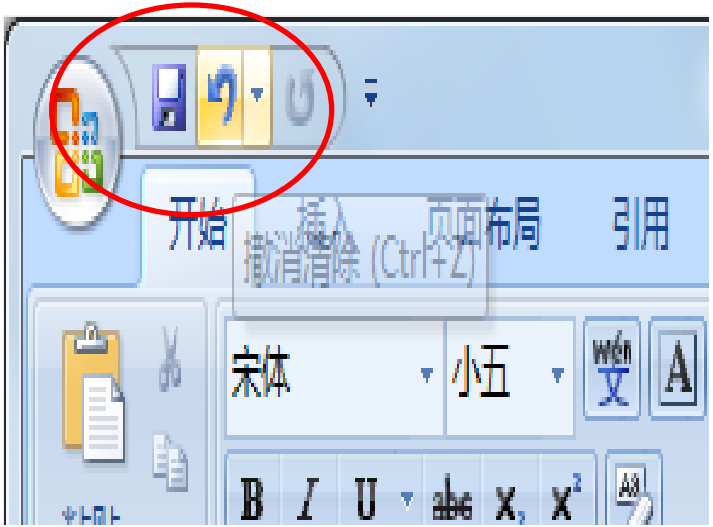
3. 栈的实例

火车道岔——典型的栈结构



3. 栈的实例





4.顺序栈的定义

```
#define  EMPTY  -1  //空栈的栈顶指针
const  int  m=1000;  //预定的栈空间大小
int s[m];  //定义顺序栈
```

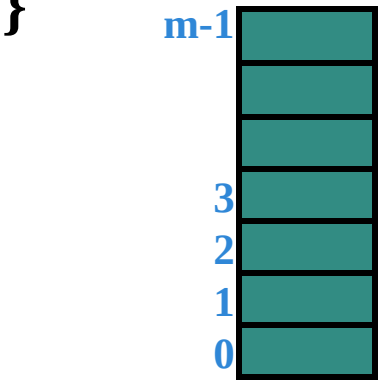
程序执行期间，将**不定期的进栈、出栈**

开始时，栈空，指针top的值等于EMPTY

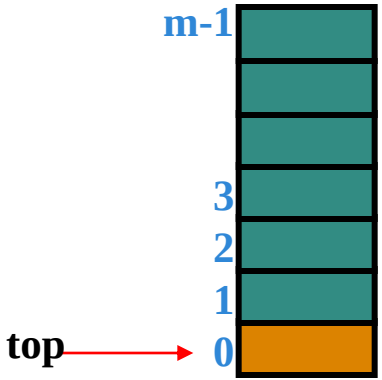
进出栈算法**需能够配合**

5. 顺序栈的进栈算法

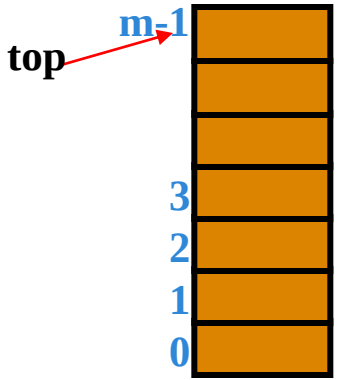
```
int push(int s[ ],int &top, int x)  //进栈函数
{   if(top==m-1)return 0; //表示栈满,不能进栈
    s[++top]=x;
    return 1; //表示进栈成功
}
```



栈空状态



进栈状态

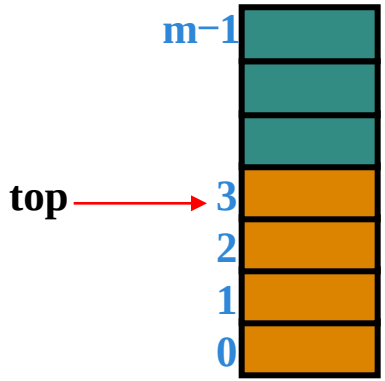


栈满状态

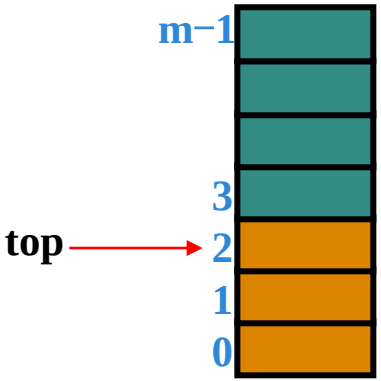
6. 顺序栈的出栈算法

```
int pop(int s[ ],int &top, int &x)  //退栈函数
{   if(top==EMPTY )return 0; //表示栈空,不能出栈
    x=s[top--];
    return 1; //表示出栈成功
}
```

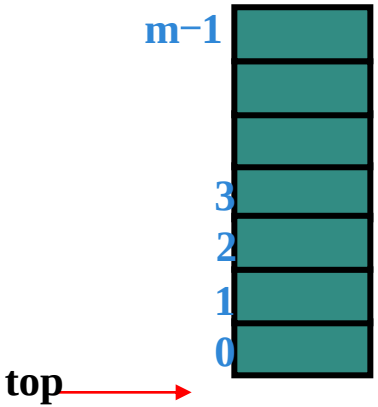
时间复杂性：T(n)=O(1)



当前状态



出栈状态

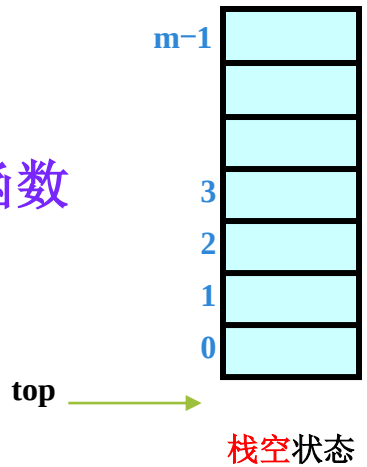


栈空状态

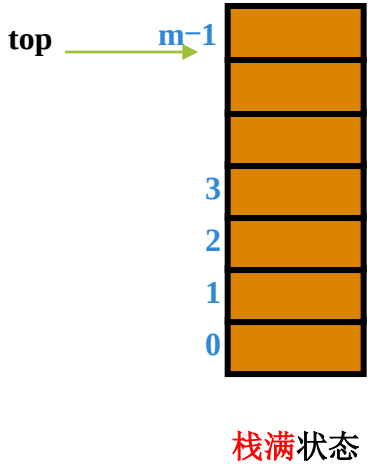
注意

进退栈算法要能够配合

```
int push(int s[ ],int &top, int x)  //进栈函数
{   if(top==m−1)  return FAIL;
    s[++top]=x;
    return SUCC;
}
int pop(int s[ ],int &top, int &x)  //退栈函数
{   if(top==EMPTY )  return FAIL;
    x=s[top- -];
    return SUCC;
}
```



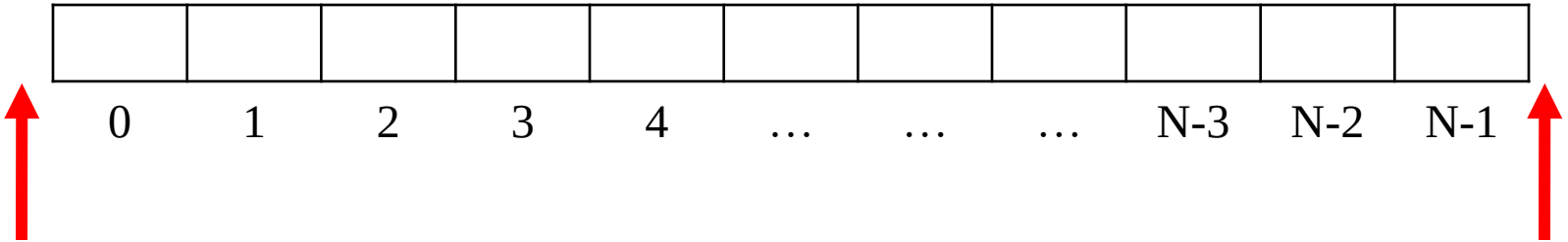
栈空状态



栈满状态

7. 两个堆栈利用空间

```
typedef struct stack
{
        int a[N];
        int top1,top2;
}stack;
stack s;
```

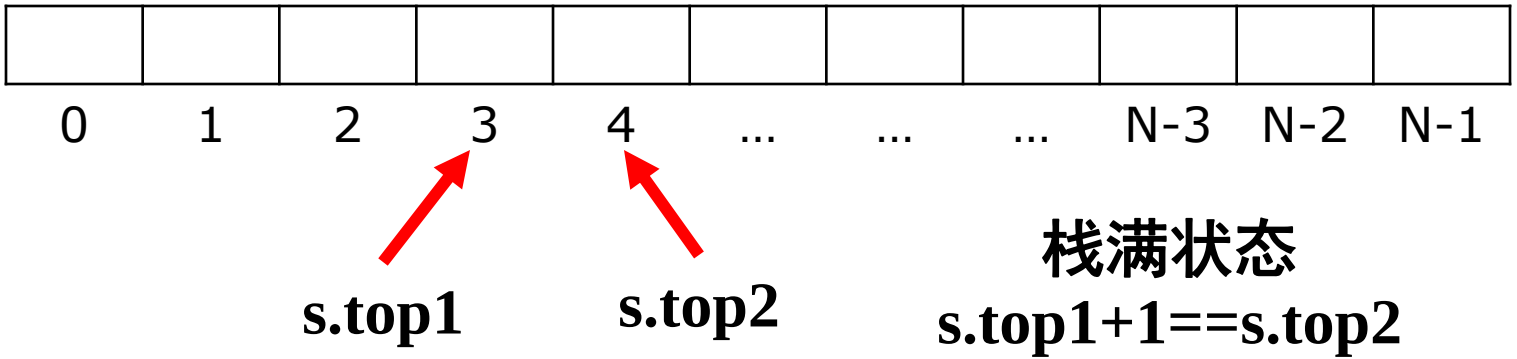


栈空状态s.top1=-1

栈空状态s.top2=N

7. 两个堆栈利用空间

```
typedef struct stack
{
        int a[N];
        int top1,top2;
}stack;
stack s;
```



8. 链栈的进栈算法

```
int  push(ptr &top,int x)
{    ptr p;
1.   p=new snode;                    //申请结点
2.   if(p==NULL)return  0;           //申请失败
3.   p->data=x;                      //x进栈
4.   p->next=top;                    //表头插入法
5.   top=p;                          //修改表头指针
6.    return  1;                     //进栈成功
}
```

9. 链栈的出栈算法

```
int  pop(ptr &top,int &x)
{   ptr  p;
    if(top==NULL)return  0; //退栈不成功
    x=top->data;
    p=top;                          //表头删除法
    top=top->next;
    free(p);
    return  1;                      //退栈成功
}
```

学习目标和要求

1.能够准确描述栈在程序中断中的作用

2.掌握简单算术表达式求值的方法

1.栈的应用—程序中断

（1）程序中断 （interrupt）

中断：一个程序执行期间，被其他程序所打断

断点：被打断的地方

现场：被打断的程序当前执行情况

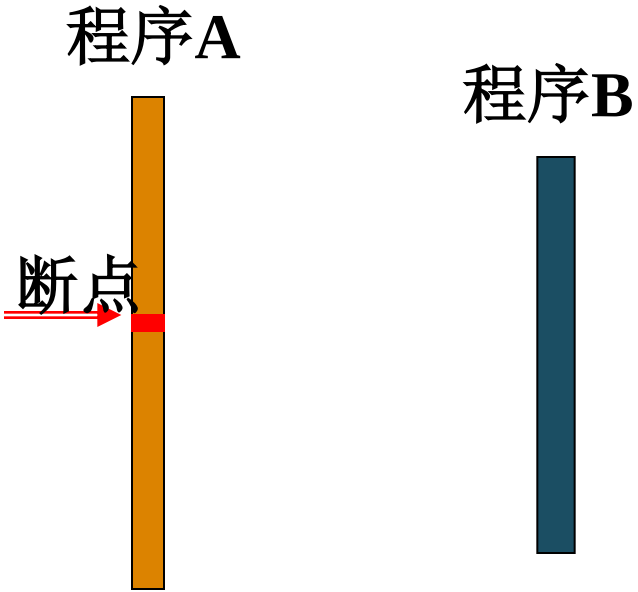
保护现场和恢复现场

嵌套中断的现场需要层层保护

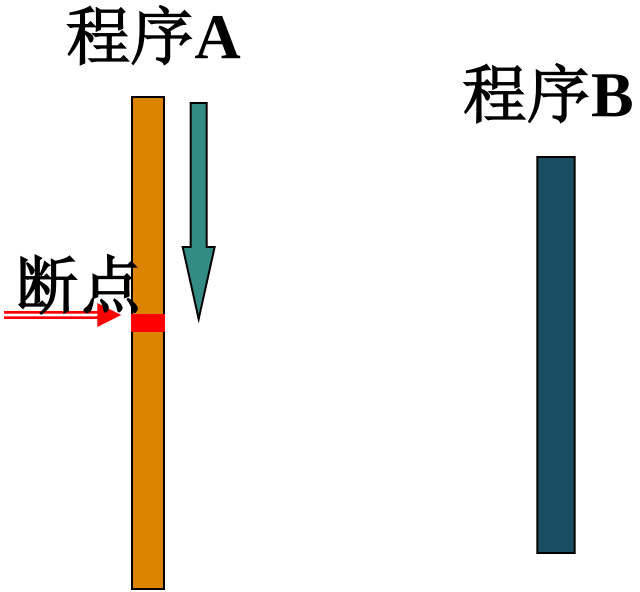
必须“后进先出”——栈

一个现场——栈架（stack frame）

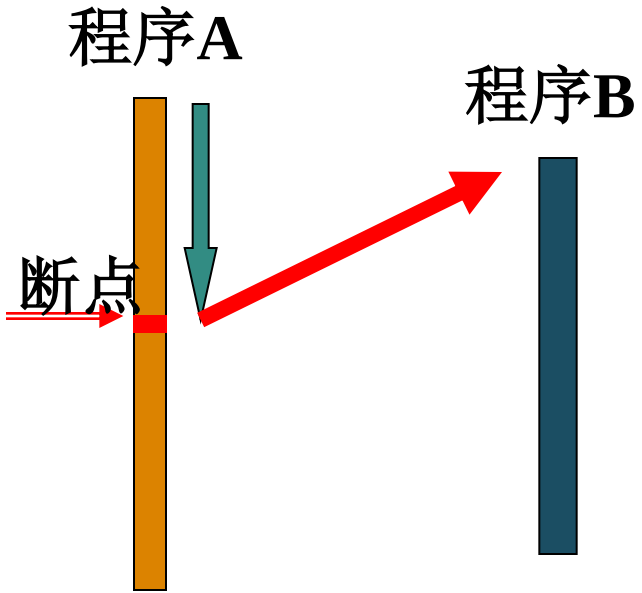
1. 栈的应用—程序中断



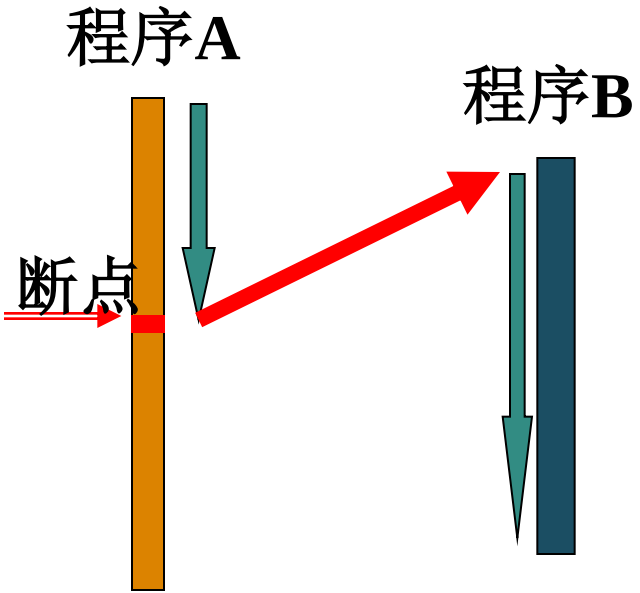
1. 栈的应用—程序中断



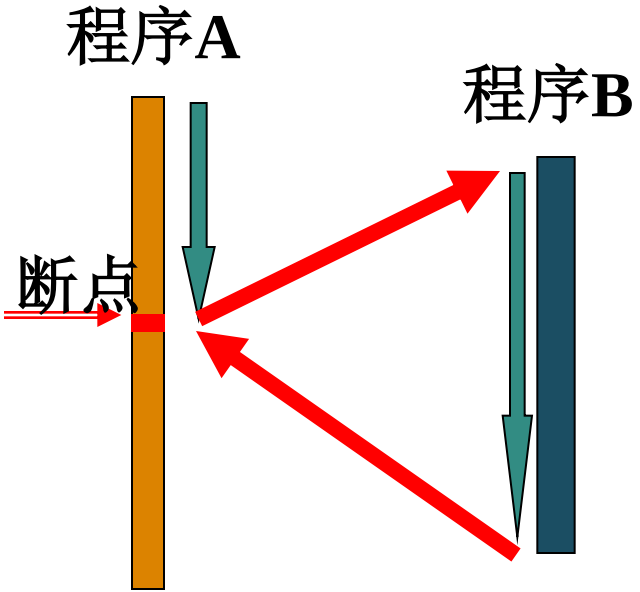
1. 栈的应用—程序中断



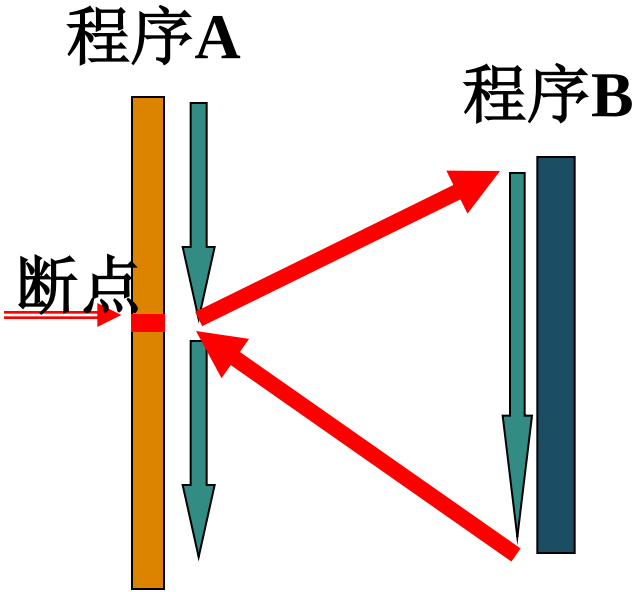
1. 栈的应用—程序中断



1. 栈的应用—程序中断



1. 栈的应用—程序中断



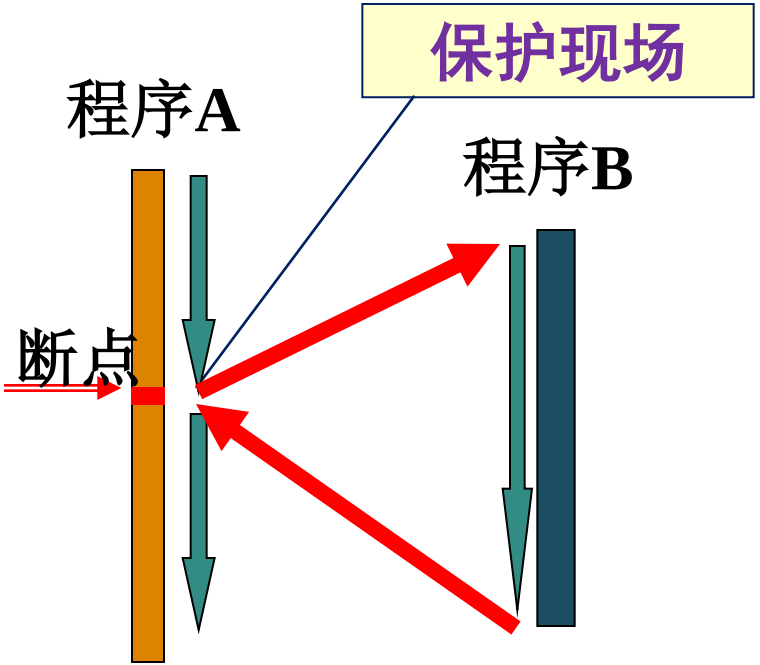
断点处的执行状态：现场

先保护现场，再执行B

B执行完后，要恢复现场

才能继续执行A的剩余部分

1. 栈的应用—程序中断

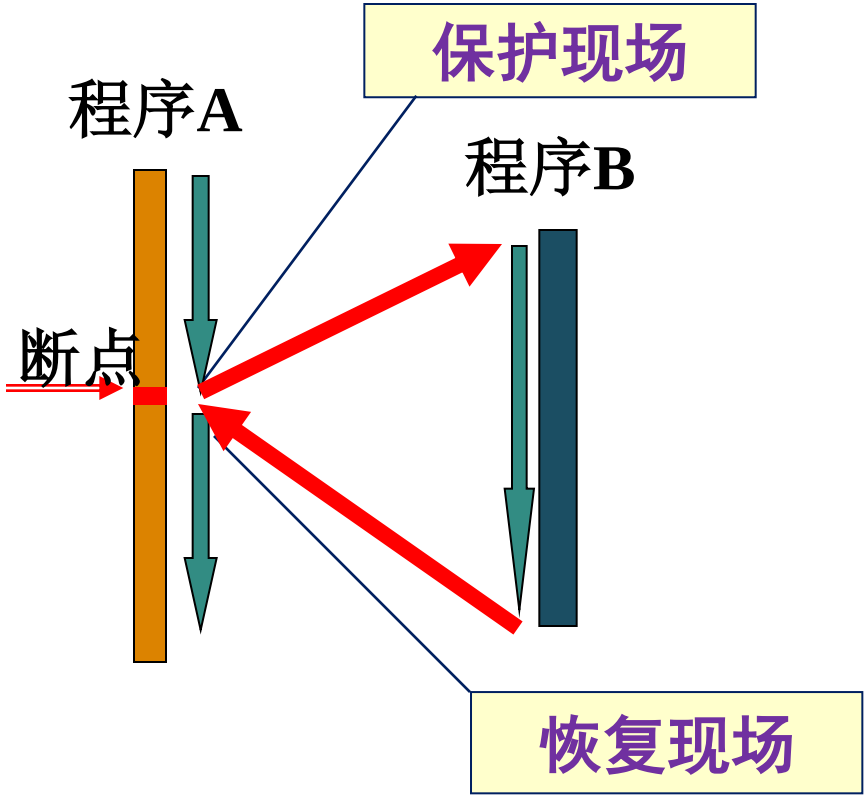


现场内容保存方式：

保护现场时

将现场内容（各寄存器当前值）保存在栈中

1. 栈的应用—程序中断



现场内容保存方式：

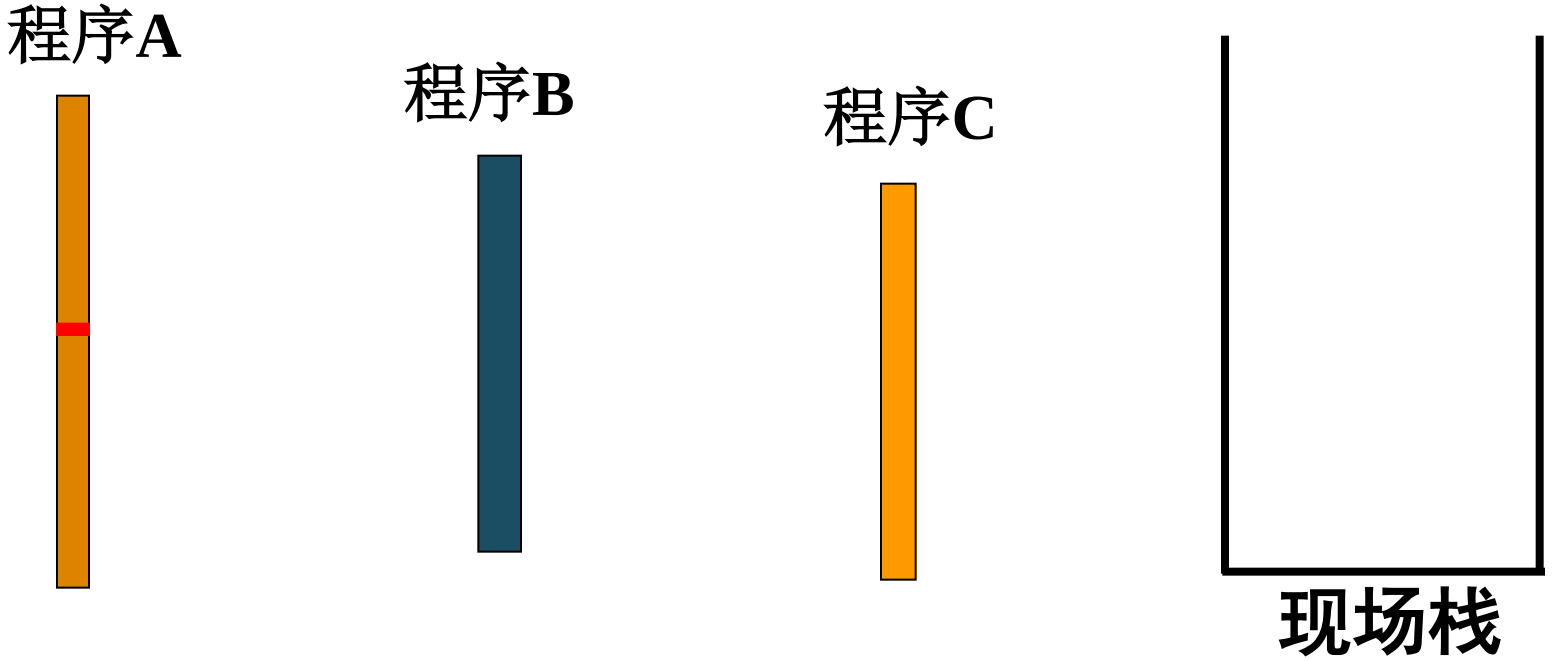
保护现场时

将现场内容（各寄存器当前值）保存在栈中

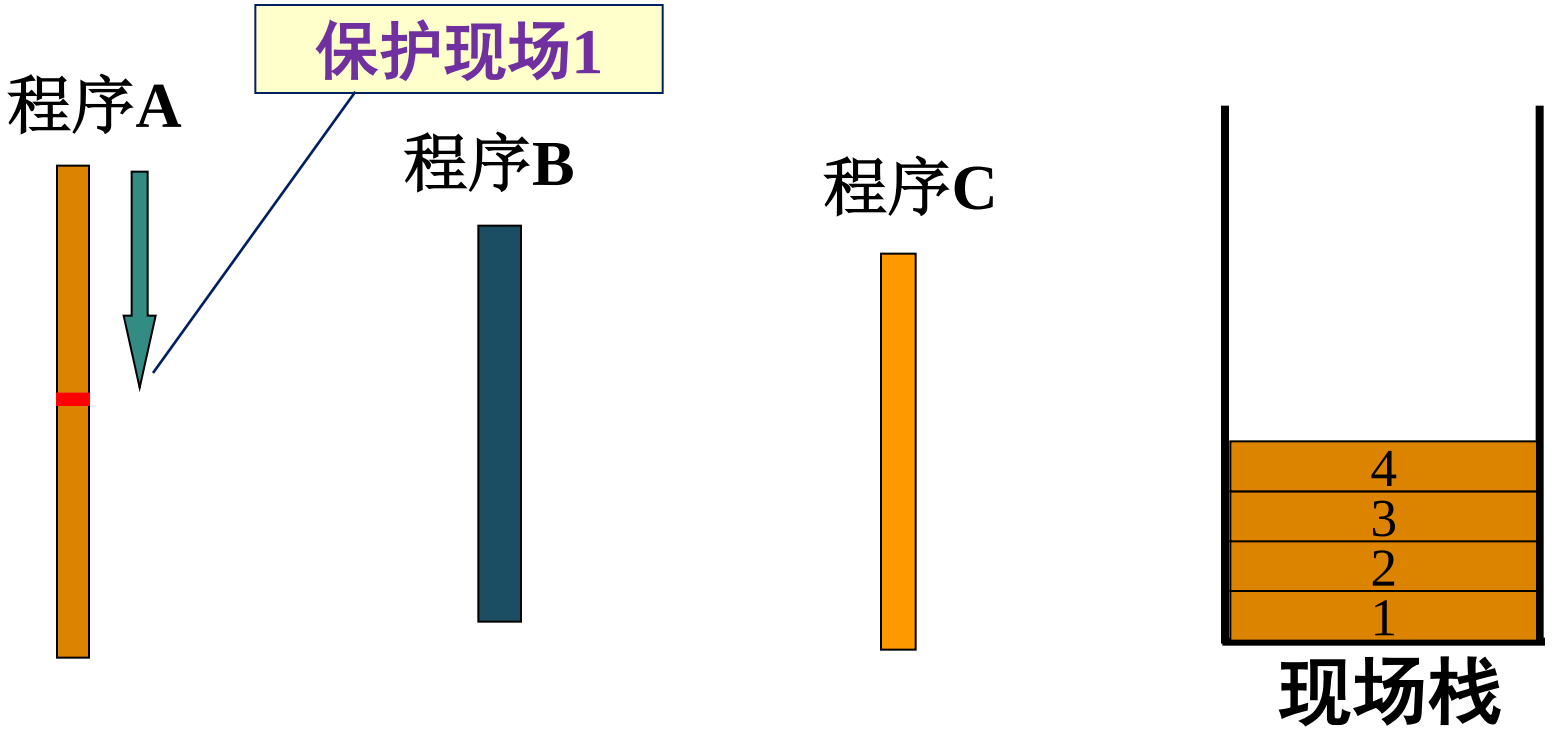
恢复现场时

将栈中内容还原到原来的寄存器中

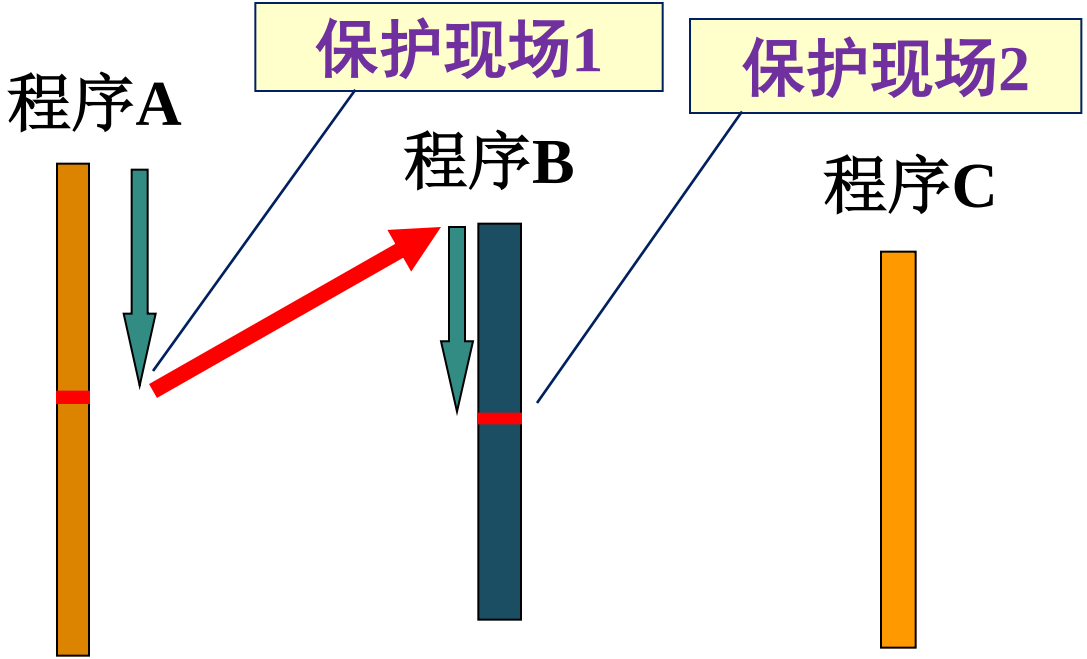
1. 栈的应用—程序中断

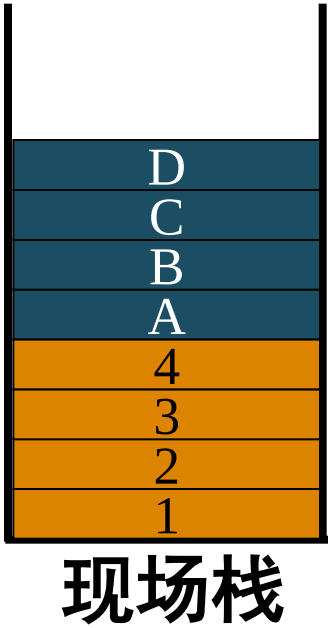


1. 栈的应用—程序中断

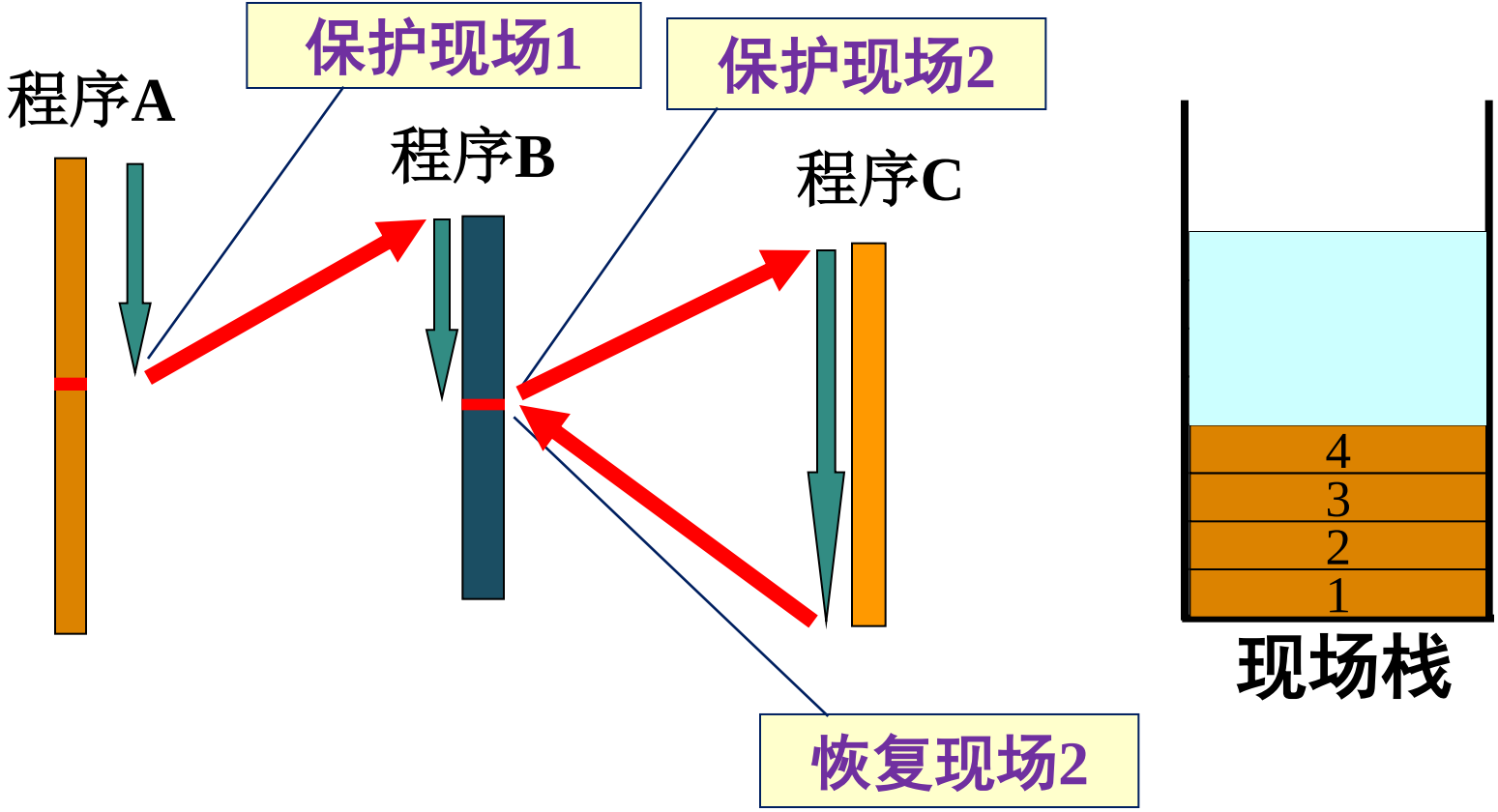


1. 栈的应用—程序中断

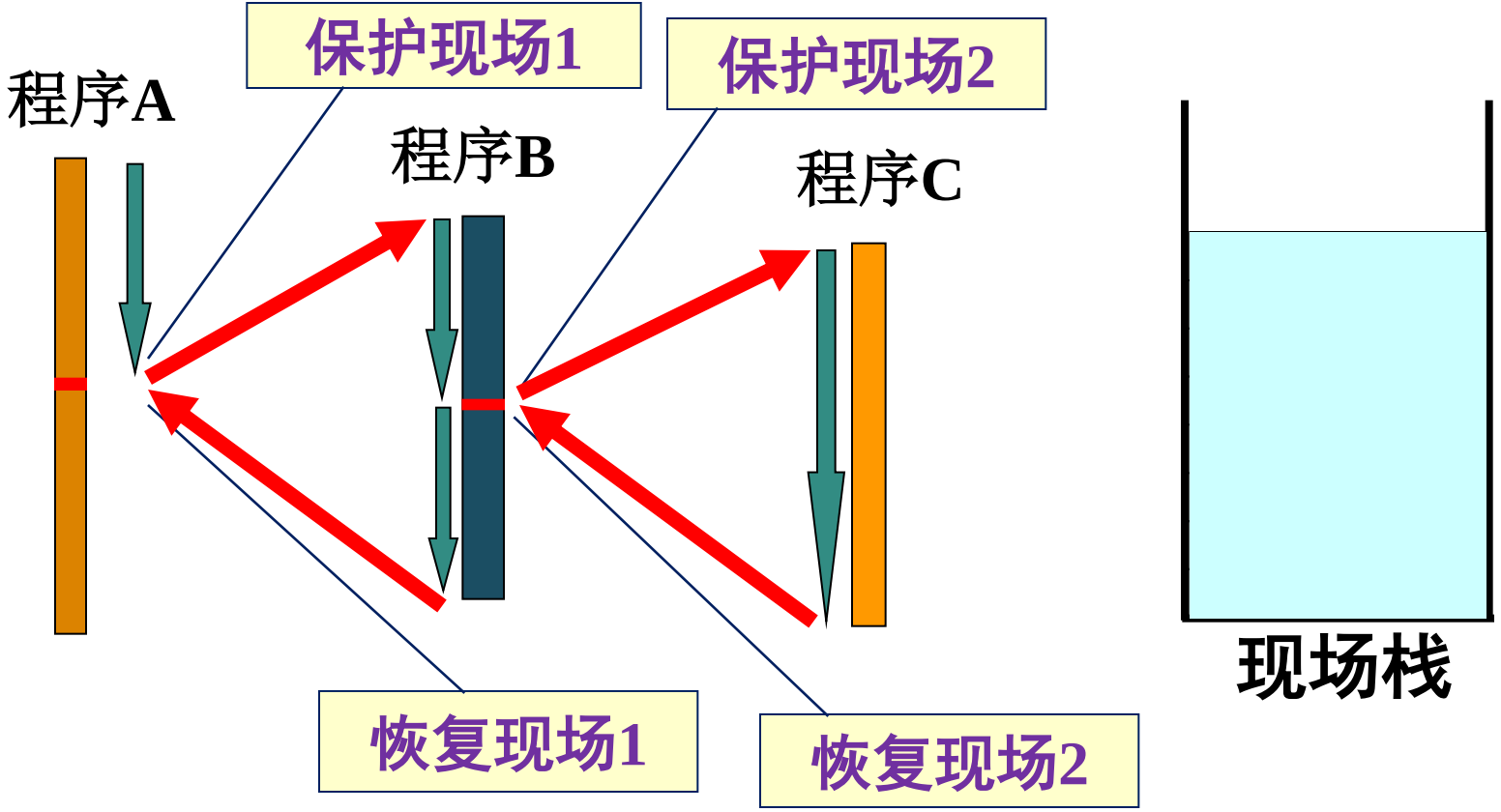




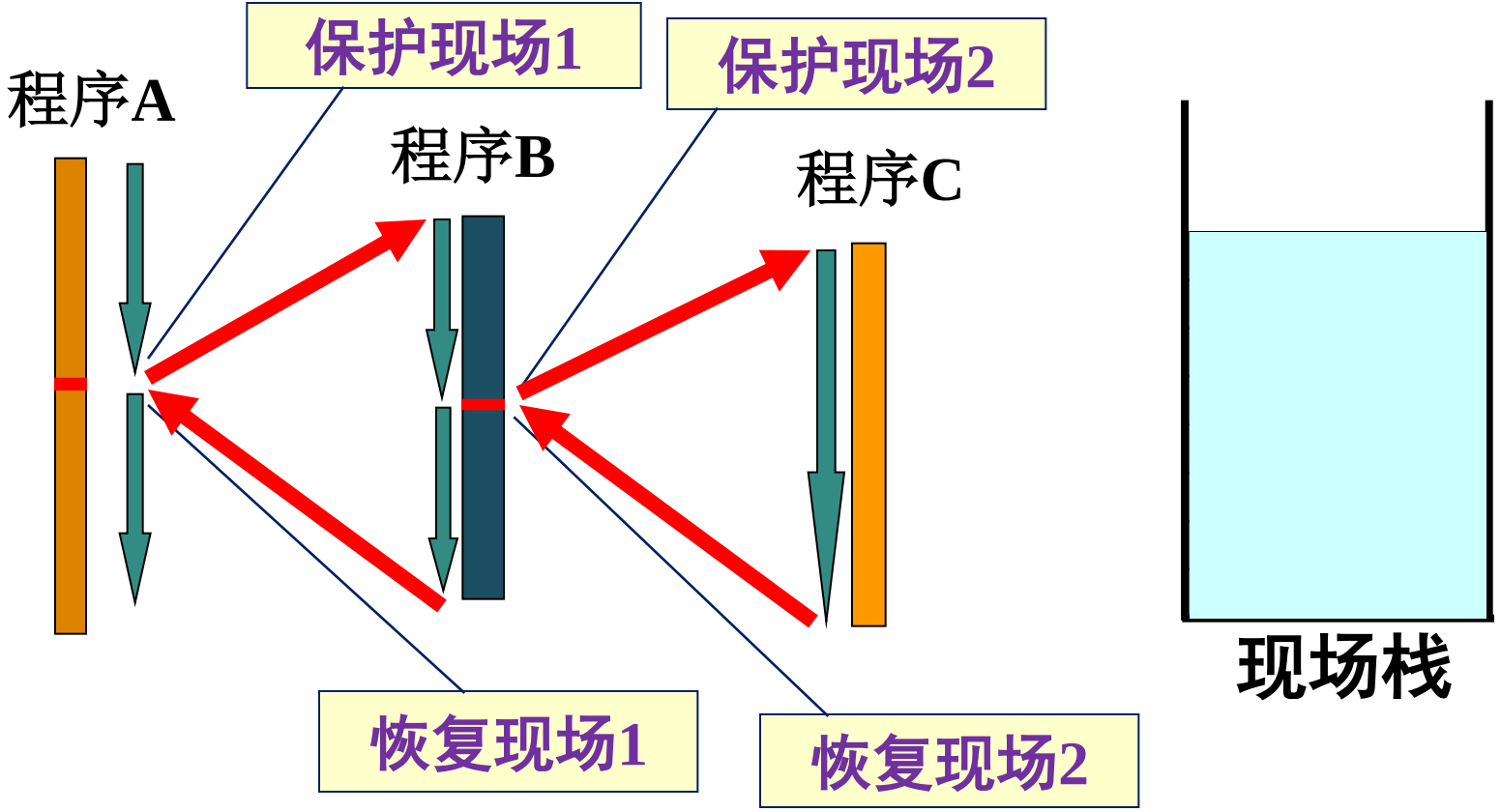
1. 栈的应用—程序中断



1. 栈的应用—程序中断



1. 栈的应用—程序中断



2. 表达式求值

简单算术表达式计算

算术表达式的组成：操作符、操作数、运算规则（算符优先级）

例：求a+b*c^d-e/f#

计算步骤:

（1）t1=c^d

（2）t2=b*t1

（3）t3=a+t2

（4）t4=e/f

（5）t5=t3-t4

2. 表达式求值

例：求a+b*c^d-e/f#

算符优先函数表

^	*	/	+	-	#
3	2	2	1	1	0

数越大，优先级越高

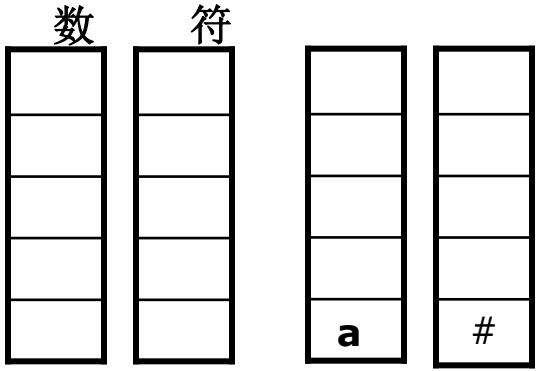
计算步骤:

（1）t1=c^d
（2）t2=b*t1
（3）t3=a+t2
（4）t4=e/f
（5）t5=t3-t4

- 实现时：两个栈（操作符栈、操作数栈），一张算符优先级表。

1. 表达式求值

输入：a+b*c^d-e/f #



规则：

当前输入若为数，则数进栈；

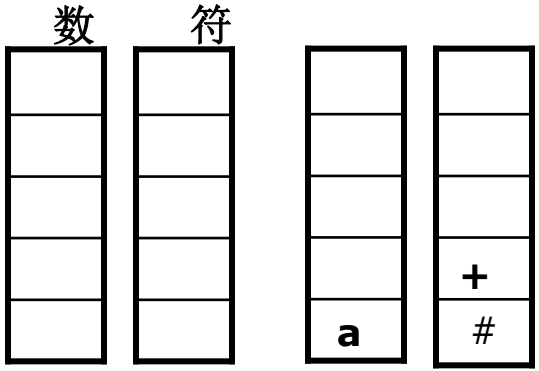
若为符，则需与符栈顶符比较，

优先级高于顶符则进栈，

否则做栈顶符运算，符、数出栈。

1. 表达式求值

输入：a+b*c^d-e/f #



规则：

当前输入若为数，则数进栈；

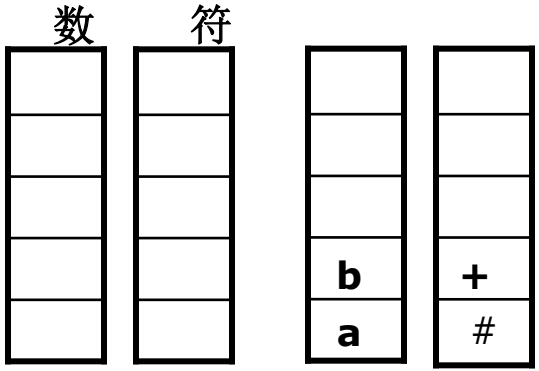
若为符，则需与符栈顶符比较，

优先级高于顶符则进栈，

否则做栈顶符运算，符、数出栈。

1. 表达式求值

输入：a+b*c^d-e/f #



规则：

当前输入若为数，则数进栈；

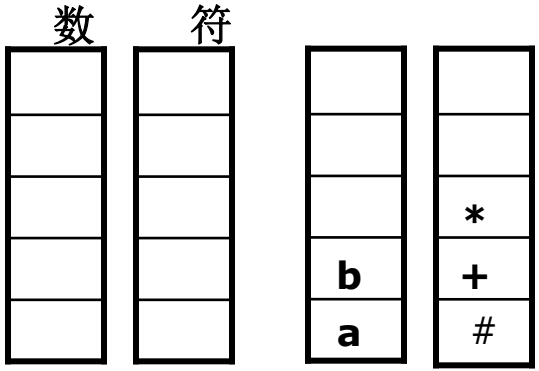
若为符，则需与符栈顶符比较，

优先级高于顶符则进栈，

否则做栈顶符运算，符、数出栈。

1. 表达式求值

输入：a+b*c^d-e/f #



规则：

当前输入若为数，则数进栈；

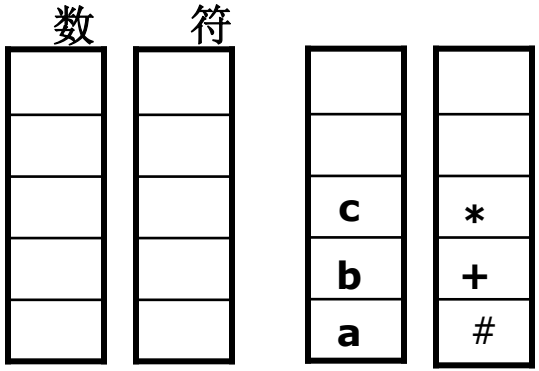
若为符，则需与符栈顶符比较，

优先级高于顶符则进栈，

否则做栈顶符运算，符、数出栈。

1. 表达式求值

输入：a+b*c^d-e/f #



规则：

当前输入若为数，则数进栈；

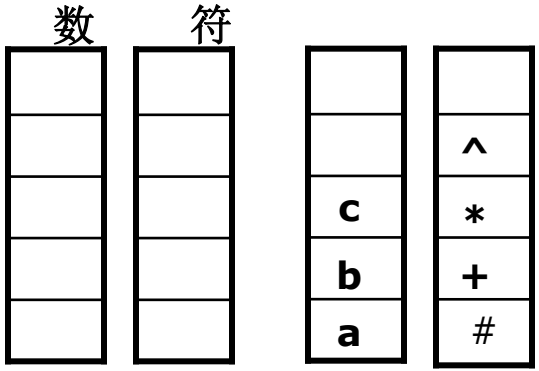
若为符，则需与符栈顶符比较，

优先级高于顶符则进栈，

否则做栈顶符运算，符、数出栈。

1. 表达式求值

输入：a+b*c^d-e/f #



规则：

当前输入若为数，则数进栈；

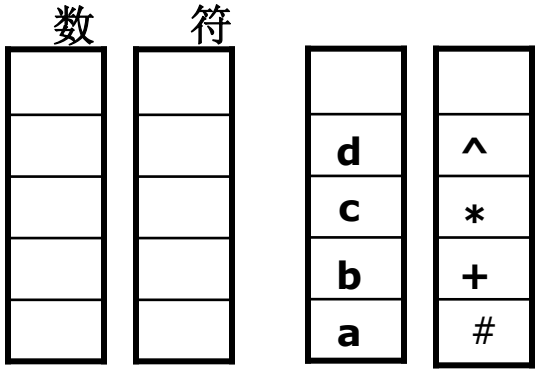
若为符，则需与符栈顶符比较，

优先级高于顶符则进栈，

否则做栈顶符运算，符、数出栈。

1. 表达式求值

输入：a+b*c^d-e/f #



规则：

当前输入若为数，则数进栈；

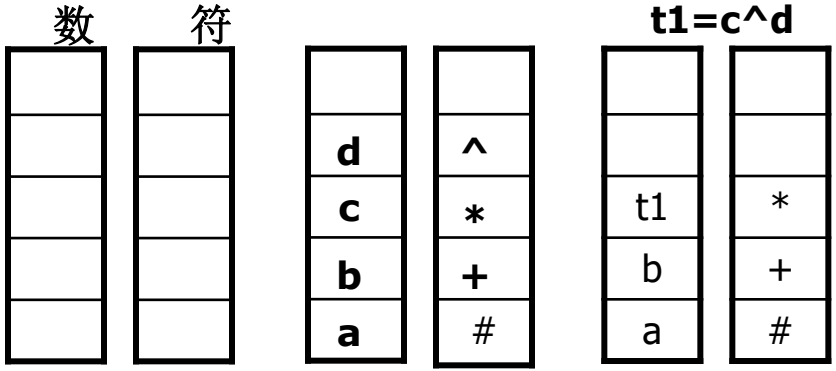
若为符，则需与符栈顶符比较，

优先级高于顶符则进栈，

否则做栈顶符运算，符、数出栈。

1. 表达式求值

输入：a+b*c^d-e/f #



规则：

当前输入若为数，则数进栈；

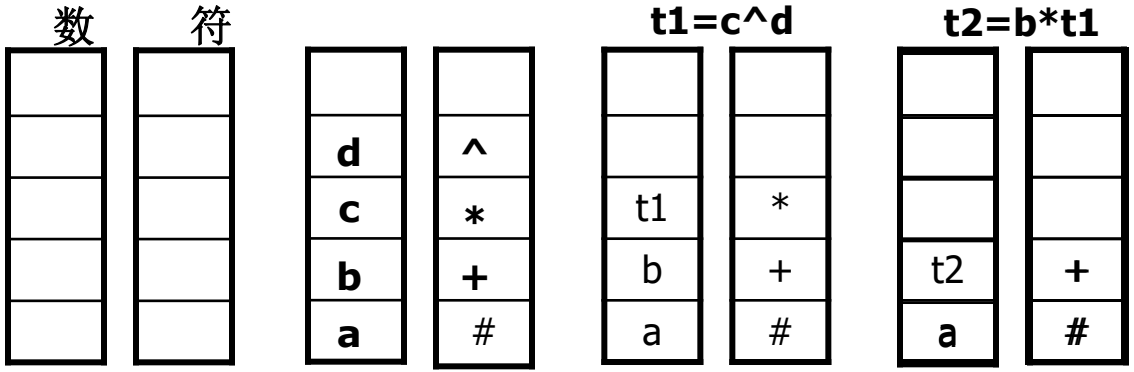
若为符，则需与符栈顶符比较，

优先级高于顶符则进栈，

否则做栈顶符运算，符、数出栈。

1. 表达式求值

输入：a+b*c^d-e/f #



规则：

当前输入若为数，则数进栈；

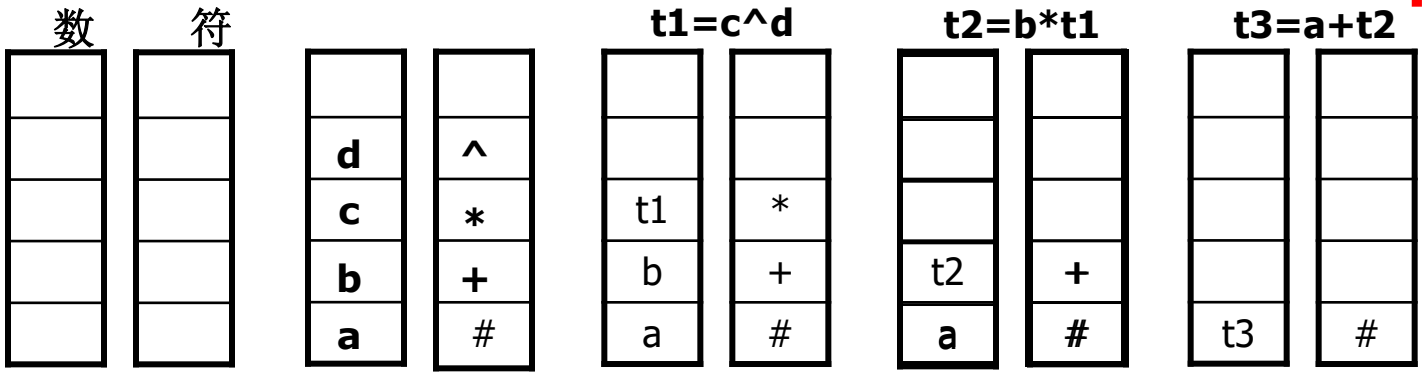
若为符，则需与符栈顶符比较，

优先级高于顶符则进栈，

否则做栈顶符运算，符、数出栈。

1. 表达式求值

输入：a+b*c^d-e/f #



规则：

当前输入若为数，则数进栈；

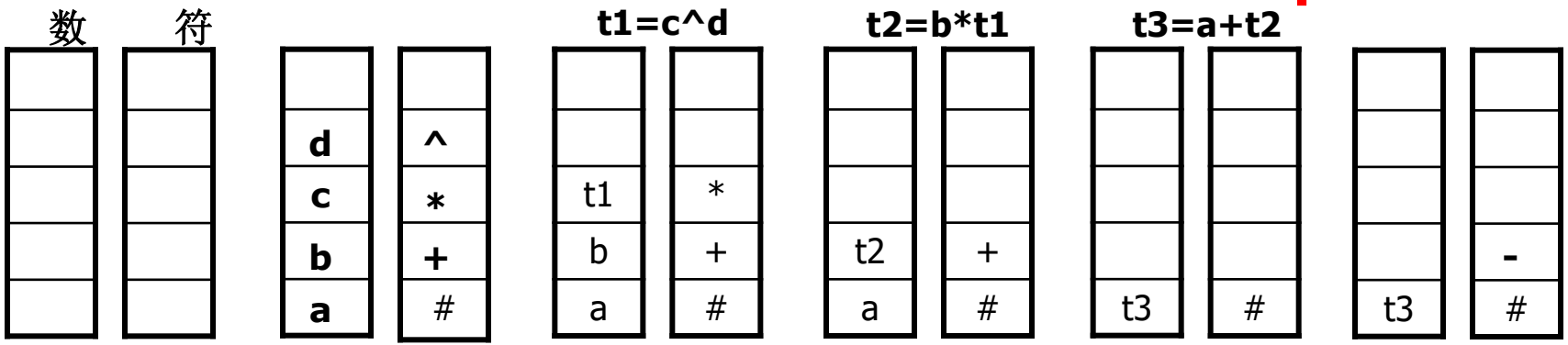
若为符，则需与符栈顶符比较，

优先级高于顶符则进栈，

否则做栈顶符运算，符、数出栈。

1. 表达式求值

输入：a+b*c^d-e/f #



规则：

当前输入若为数，则数进栈；

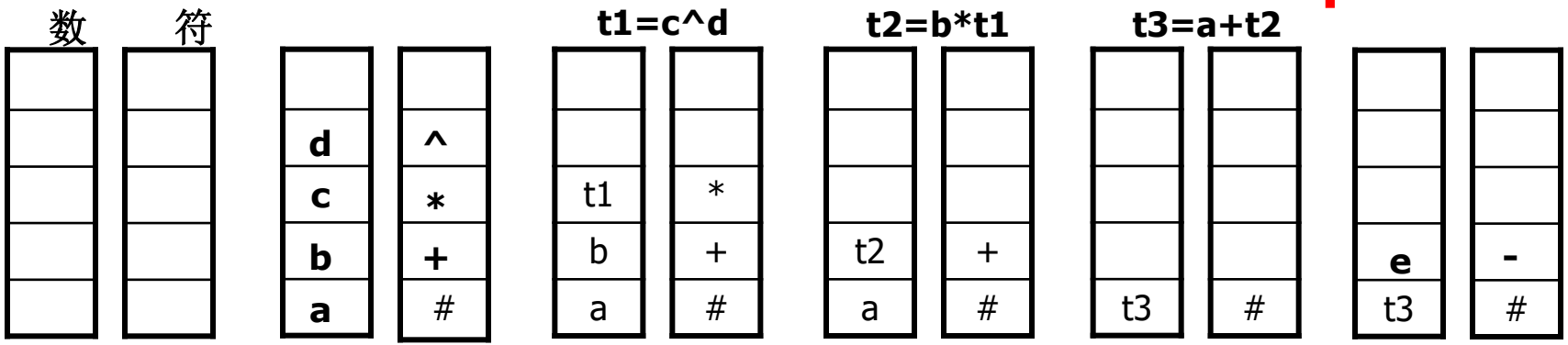
若为符，则需与符栈顶符比较，

优先级高于顶符则进栈，

否则做栈顶符运算，符、数出栈。

1. 表达式求值

输入：a+b*c^d-e/f #



规则：

当前输入若为数，则数进栈；

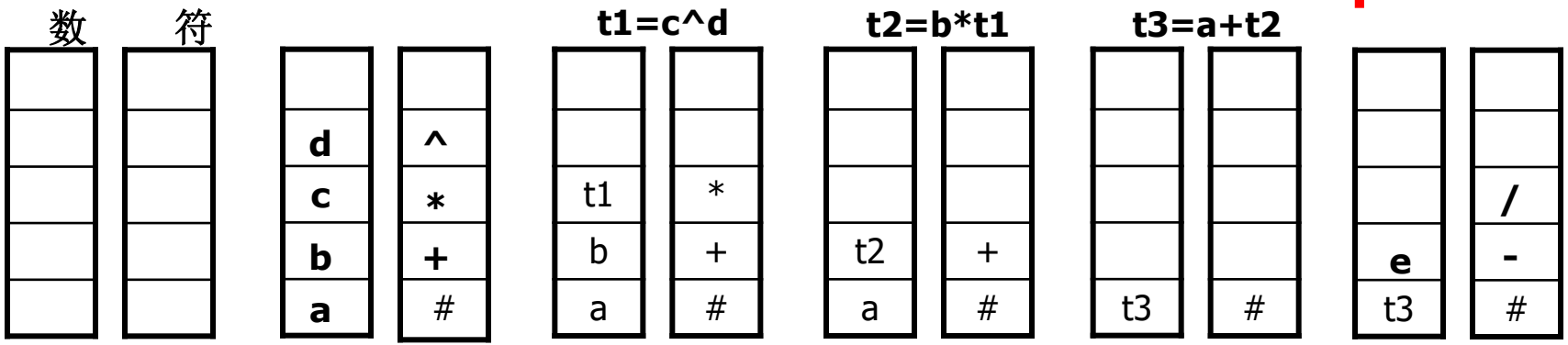
若为符，则需与符栈顶符比较，

优先级高于顶符则进栈，

否则做栈顶符运算，符、数出栈。

1. 表达式求值

输入：a+b*c^d-e/f #



规则：

当前输入若为数，则数进栈；

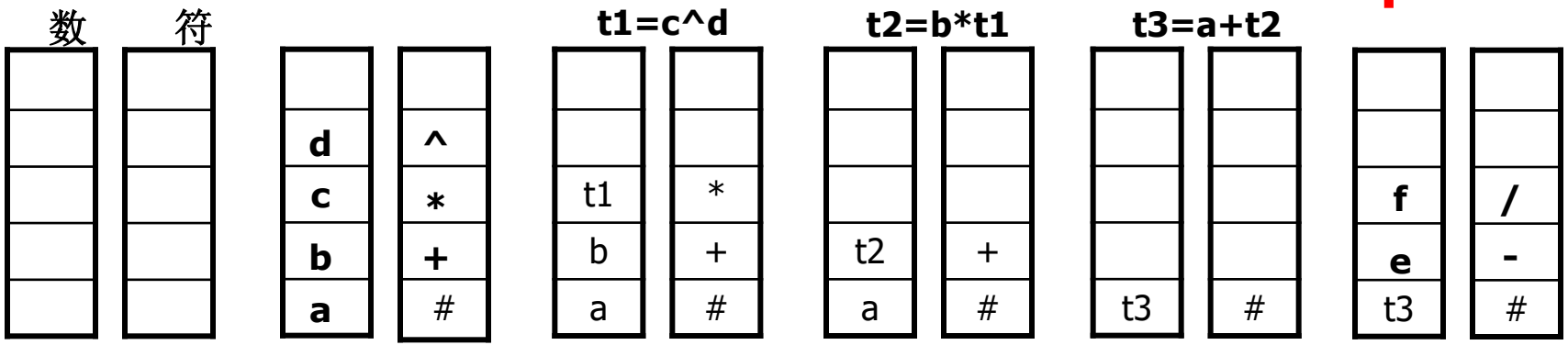
若为符，则需与符栈顶符比较，

优先级高于顶符则进栈，

否则做栈顶符运算，符、数出栈。

1. 表达式求值

输入：a+b*c^d-e/f #



规则：

当前输入若为数，则数进栈；

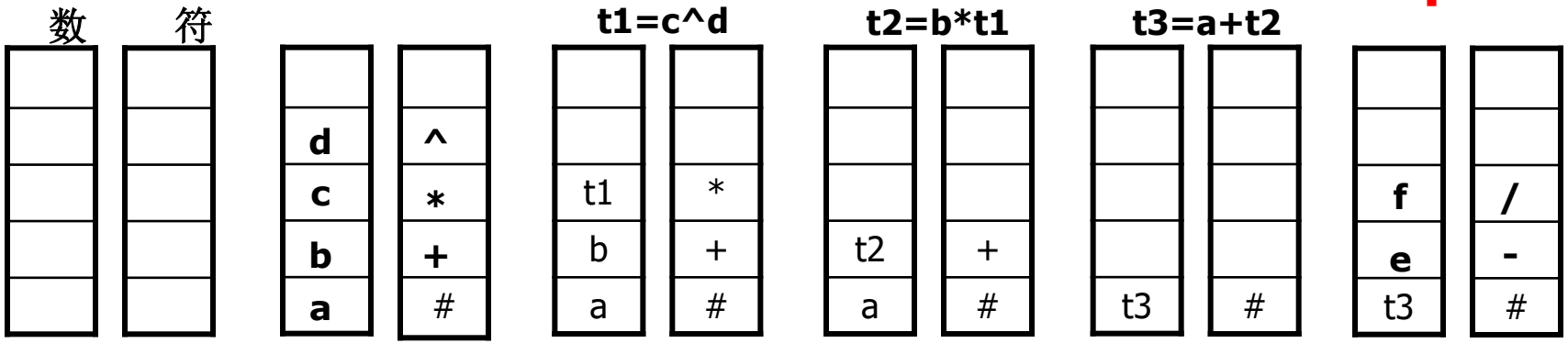
若为符，则需与符栈顶符比较，

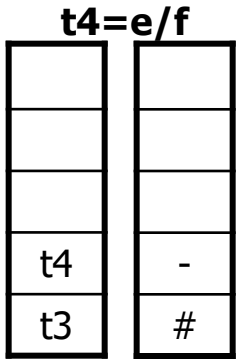
优先级高于顶符则进栈，

否则做栈顶符运算，符、数出栈。

1. 表达式求值

输入：a+b*c^d-e/f #





规则：

当前输入若为数，则数进栈；

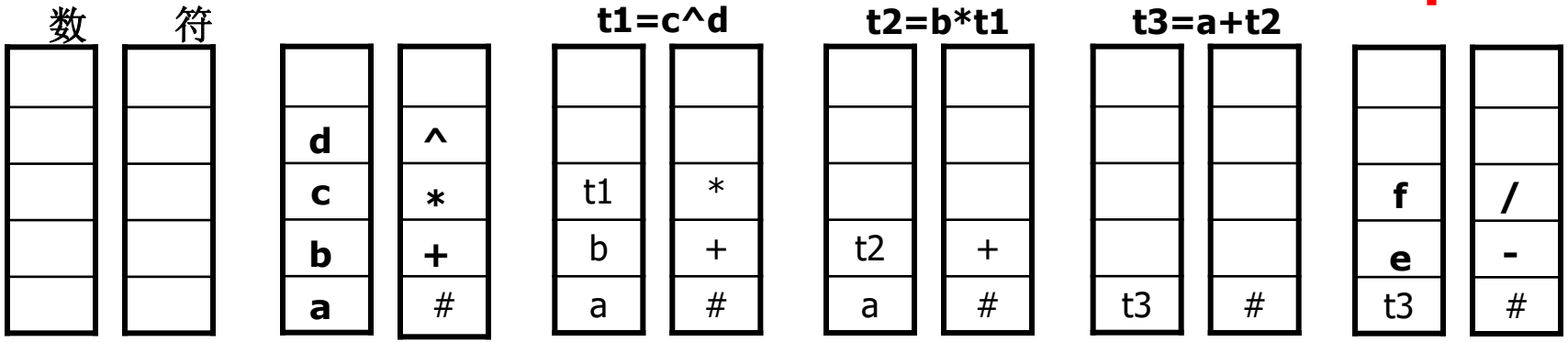
若为符，则需与符栈顶符比较，

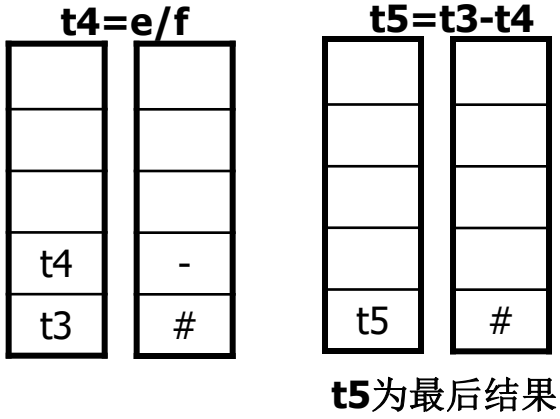
优先级高于顶符则进栈，

否则做栈顶符运算，符、数出栈。

1. 表达式求值

输入：a+b*c^d-e/f #





t5为最后结果

规则：

当前输入若为数，则数进栈；

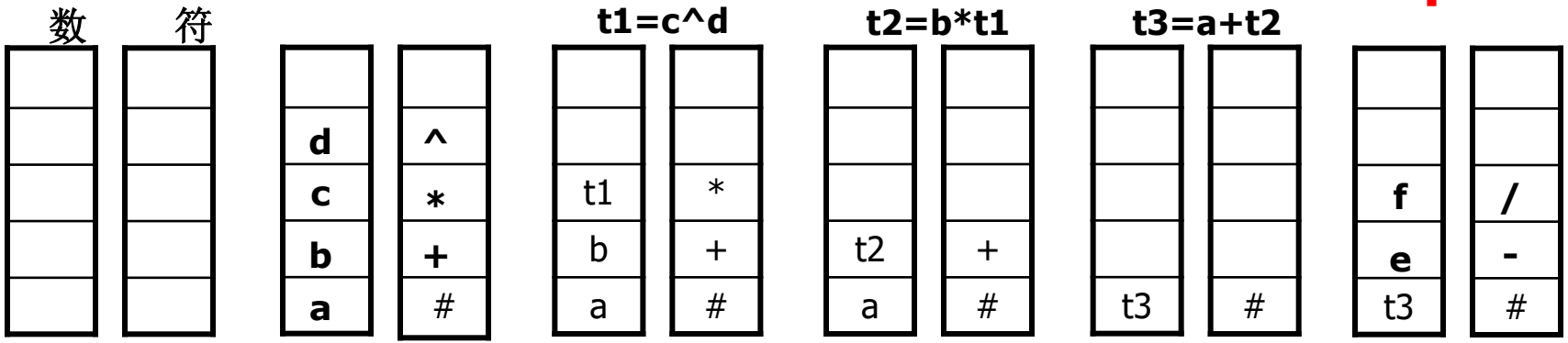
若为符，则需与符栈顶符比较，

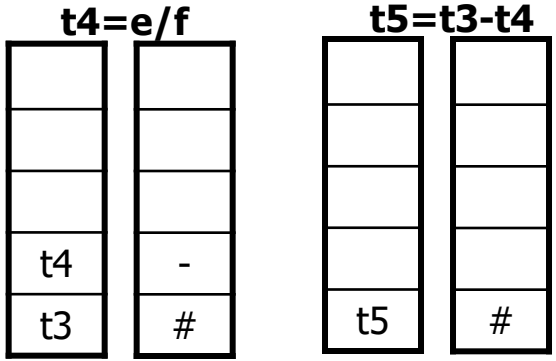
优先级高于顶符则进栈，

否则做栈顶符运算，符、数出栈。

1. 表达式求值

输入：a+b*c^d-e/f #





t5为最后结果

计算步骤：

（1）t1=c^d

（2）t2=b*t1

（3）t3=a+t2

（4）t4=e/f

（5）t5=t3-t4

2. 表达式求值算法

△:当前运算符
⊕:栈顶运算符

步骤1）扫描表达式的当前元素

步骤2）如果当前元素是操作数，使其进s栈，转步骤1

步骤3）如果当前元素是运算符△，则反复执行步骤4~8

步骤4）若f栈空，转步骤9终止循环；否则，进行下一步

步骤5）使△与f的栈顶运算符⊕比较优先级

步骤6）若△的优先级高于⊕的优先级，转步骤9；否则
（△的优先级不高于⊕的优先级）进行下一步

步骤7）从s栈中依次退出两个操作数x2和x1，从f栈退出一个运算符⊕，进行一次运算x3=x1⊕x2，使x3进入栈s

步骤8）转步骤4

步骤9）若△是结束符“#”，则算法结束，此时s栈中只有一个元素，即是计算的结果；否则，使△进f栈，转步骤1

2. 表达式求值算法

△ \ ⊕	+	-	*	/	(	)	=
+	>	>	<	<	<	>	>
-	>	>	<	<	<	>	>
*	>	>	>	>	<	>	>
/	>	>	>	>	<	>	>
(	<	<	<	<	<	=	e
)	>	>	>	>	e	>	>
=	<	<	<	<	<	e	=

当前运算符

栈顶运算符

小结

- 栈是最常用的表结构
- 从根本上认识栈的基本原理和操作方法，对于认识程序的嵌套和递归调用，在程序设计中自觉地使用栈具有重要的意义
- 在应用方面，借助中缀表达式求值算法的原理，进一步理解栈在算法设计中的应用，是非常必要的。