Data Structures

栈和队列
Stacks & Queues

2022年9月19日

学而不厌 诲人不倦

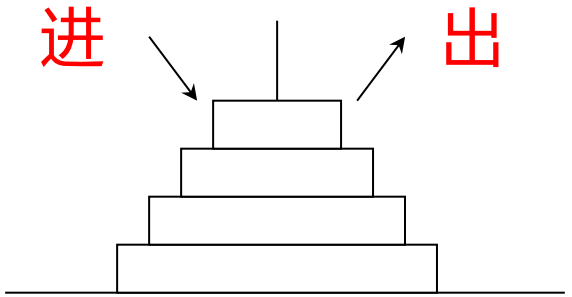
Chapter 3 栈和队列

- 3.1 引言
- 3.2 栈
- 3.3 队列
- 3.4 扩展与提高
- 3.5 应用举例

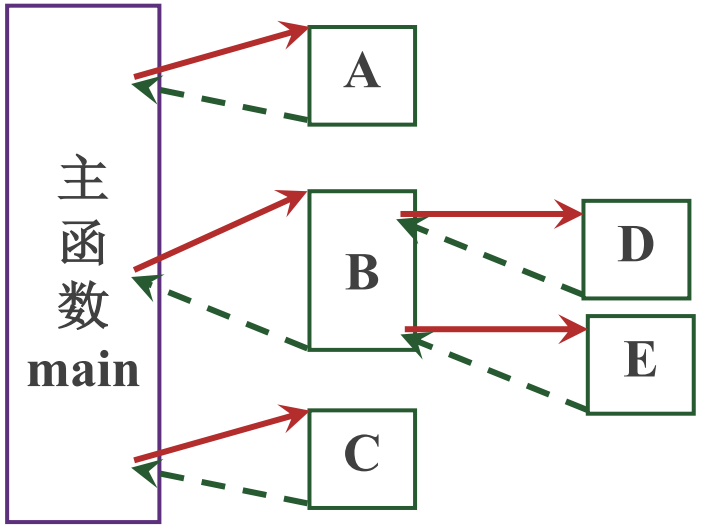
3.1 引言

3.1 引言

- 栈和队列是两种重要的线性结构
- 栈和队列是操作受限的线性表



汉诺塔



函数嵌套调用

$$(23)_{10} = (10111)_2$$

2	23	1
2	11	1
2	5	1
2	2	0
2	1	1
	0	

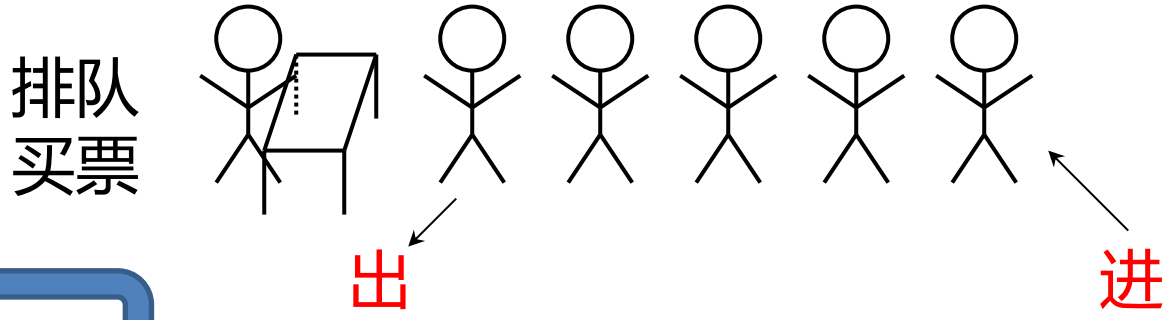
进制转换问题

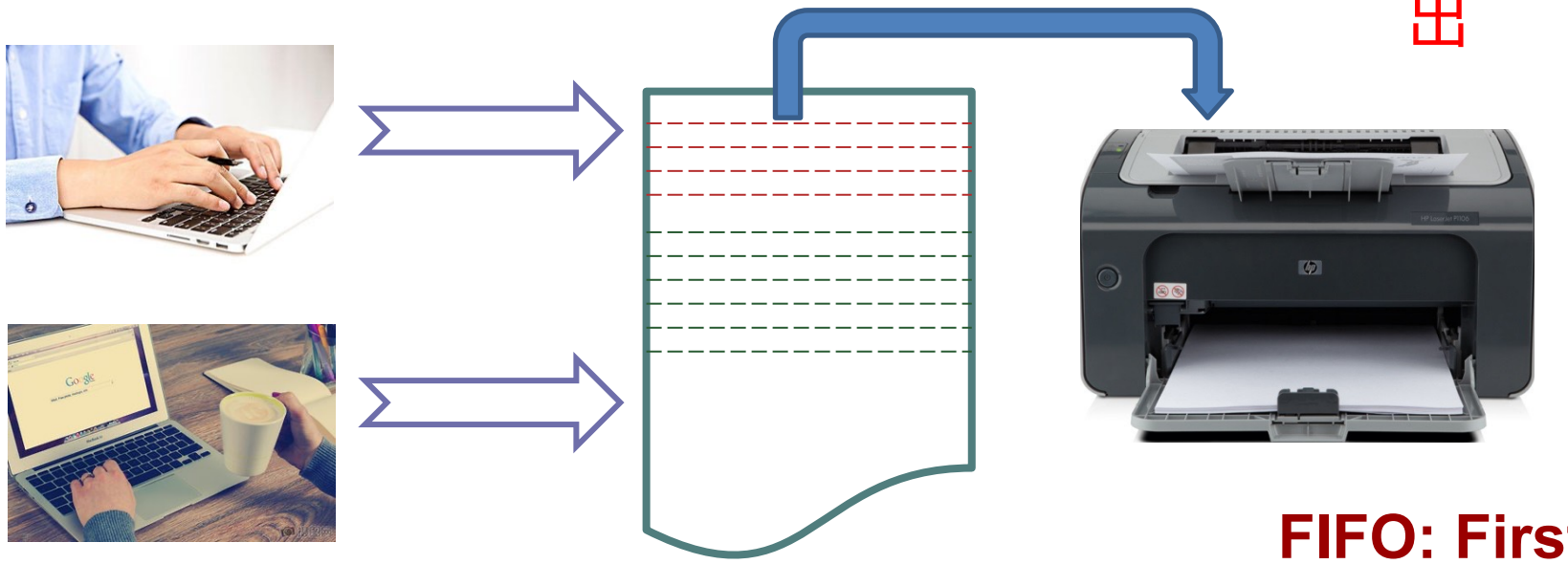
LIFO: Last In First Out

在实际问题的处理过程中，有些数据具有后到先处理的特点

3.1 引言

- 栈和队列是两种重要的线性结构
- 栈和队列是操作受限的线性表





FIFO: First In First Out

在实际问题的处理过程中，有些数据具有先到先处理的特点

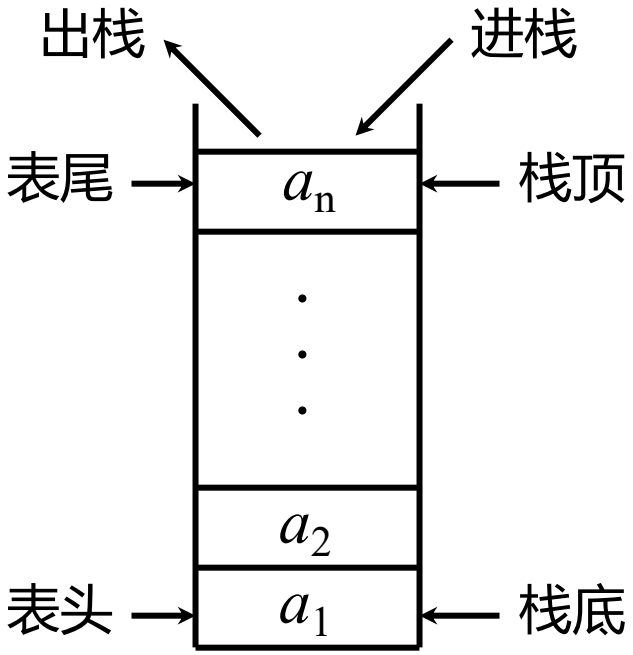
3.2 栈

3-2-1　栈的逻辑结构

1. 栈的定义

栈是限定仅在表尾进行插入或删除操作的线性表。

通常，表头端称为栈底，表尾允许插入和删除的一端称为栈顶（top）。



a_1为栈底元素

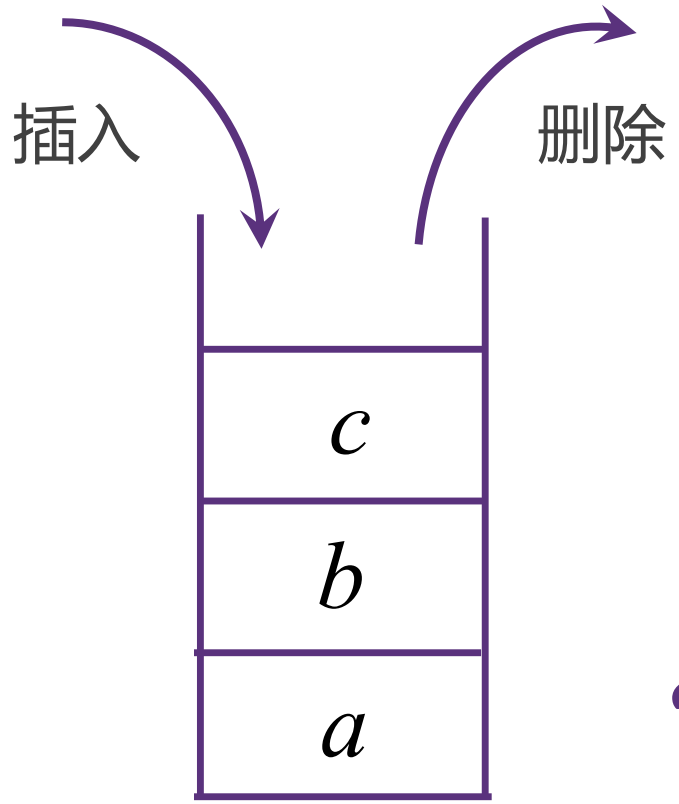
a_n为栈顶元素

按a_1、a_2…、a_n顺序进栈

按a_n…、a_2、a_1顺序出栈

栈的操作特性：后进先出（Last In First Out，LIFO）

2. 栈的操作特性



插入：入栈、进栈、压栈 Push

删除：出栈、弹栈 Pop

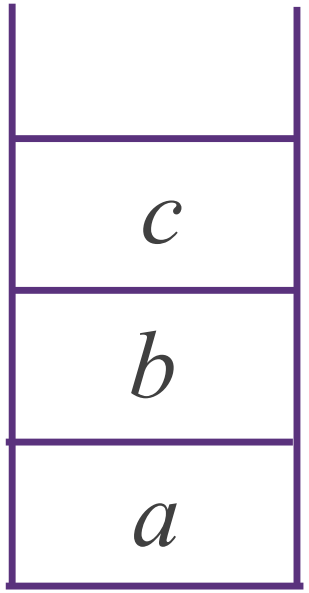
空栈：不含任何数据元素的栈

条件判断

此时执行出栈操作，哪个元素可以出栈呢？

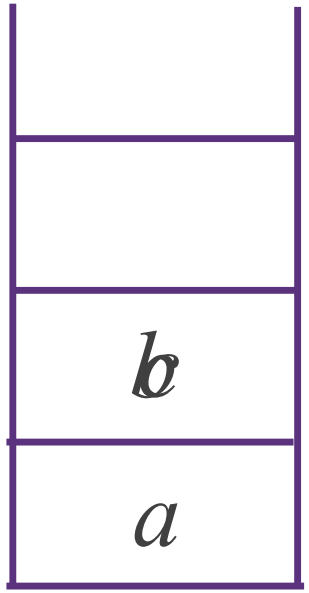
2. 栈的操作特性

例：有三个元素按a、b、c的次序依次进栈，且每个元素只允许进一次栈，则可能的出栈序列有多少种？



情况一

出栈：c b a



情况二

出栈：b c a

能否得到如下出栈序列？

出栈：c a b

例：一个栈的入栈序列为1，2，3，4，5，则如下不可能的输出序列是（ ）。

A. 54321　　B. 45321　　C. 43512　　D. 12345

3. 栈的抽象数据类型定义

ADT Stack

DataModel

栈中元素具有相同类型及后进先出特性，相邻元素具有前驱和后继关系

Operation

InitStack：栈的初始化

DestroyStack ：栈的销毁

Push：入栈

Pop：出栈

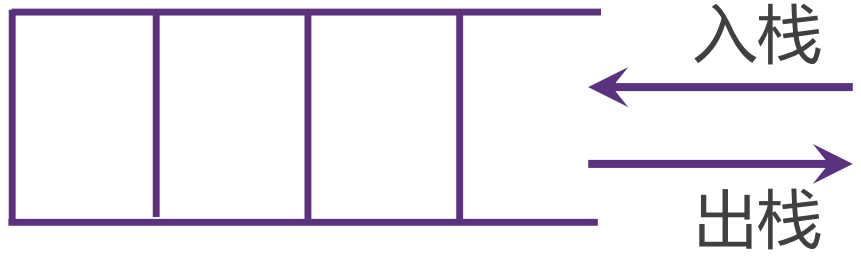
GetTop：取栈顶元素

Empty：判空

endADT

相对于其他数据结构，栈的基本操作是确定的

3. 栈的抽象数据类型定义



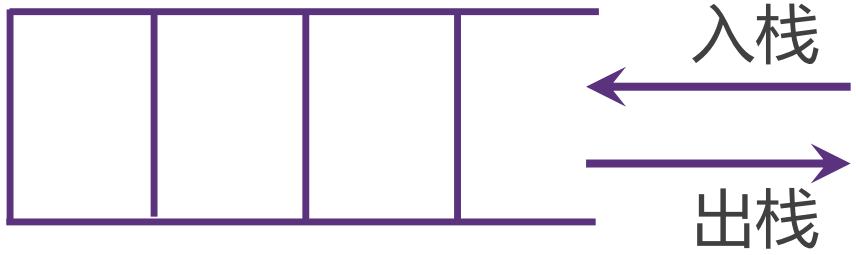
InitStack
输入：无
功能：栈的初始化，初始化一个空栈
输出：无

DestroyStack
输入：无
功能：销毁栈，释放栈所占用的存储空间
输出：无

Push
输入：元素值 x
功能：在栈顶插入一个元素 x
输出：如果插入成功，栈顶增加了一个元素，否则返回失败信息

Push操作需要指明插入位置吗？

3. 栈的抽象数据类型定义

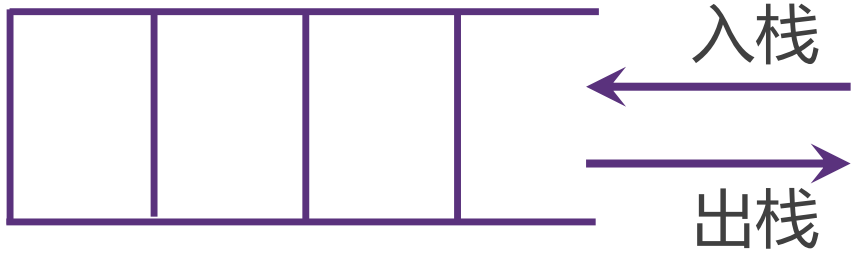


Pop
输入：无
功能：删除栈顶元素
输出：如果删除成功，返回被删元素值；否则返回失败信息

GetTop
输入：无
功能：读取当前的栈顶元素
输出：若栈不空，返回当前的栈顶元素值；否则返回失败信息

Empty
输入：无
功能：判断栈是否为空
输出：如果栈为空，返回 1；否则，返回 0

3. 栈的抽象数据类型定义



Pop
输入：无
功能：删除栈顶元素
输出：如果删除成功，返回被删元素值；否则返回失败信息

GetTop
输入：无
功能：读取当前的栈顶元素
输出：若栈不空，返回当前的栈顶元素值；否则返回失败信息

Empty
输入：无
功能：判断栈是否为空
输出：如果栈为空，返回 1；否则，返回 0

3.2 栈

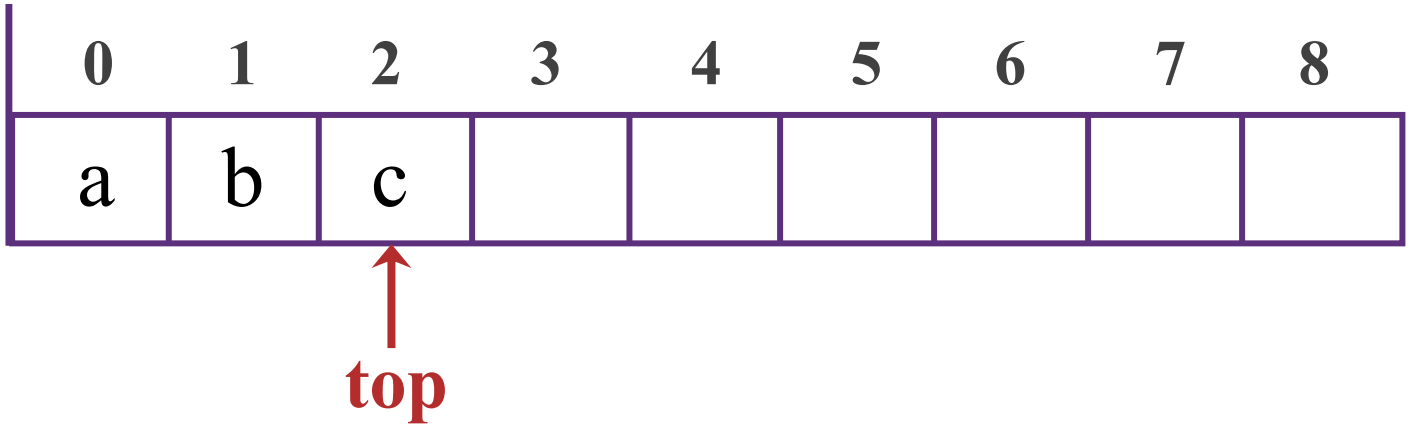
3-2-2　栈的顺序存储结构及其实现

2022年9月23日 第1节

1. 顺序栈的存储结构定义

顺序栈：栈的顺序存储结构

什么是顺序存储？



如何改造数组实现栈的顺序存储呢？

如何表示栈底：用数组的一端作为栈底

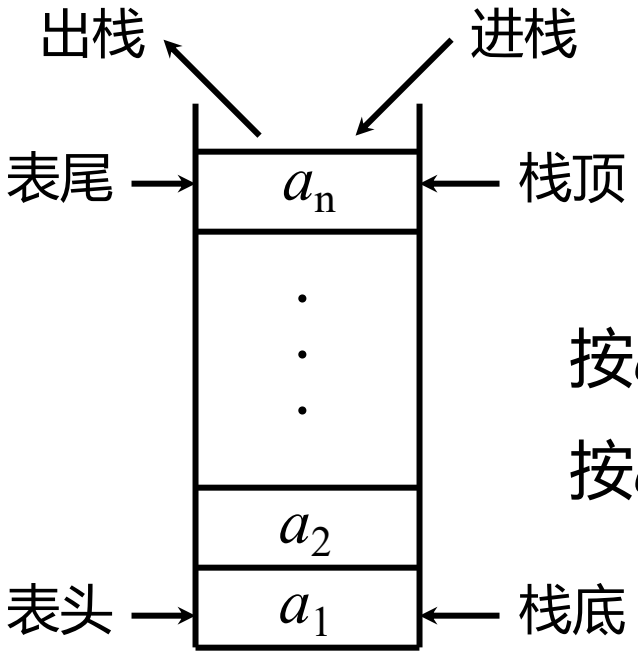
如何表示栈顶：设变量top存储栈顶元素所在的下标

1. 栈的定义

栈的操作特性：后进先出（Last In First Out，LIFO）

栈是限定仅在表尾进行插入或删除操作的线性表。

表头端称为栈底，表尾允许插入和删除的一端称为栈顶（top）。



a_1为栈底元素

a_n为栈顶元素

按a_1、a_2…、a_n顺序进栈

按a_n…、a_2、a_1顺序出栈

```
const int StackSize = 10;
template <typename DataType>
class SeqStack
{
public:
        SeqStack();
        ~SeqStack();
        void Push(DataType x);
        DataType Pop();
        DataType GetTop();
        int Empty();
private:
        DataType data[StackSize];
        int  top;
};
```

2. 顺序栈的类定义

栈的抽象数据类型定义？

InitStack：栈的初始化

DestroyStack ：栈的销毁

Push：入栈

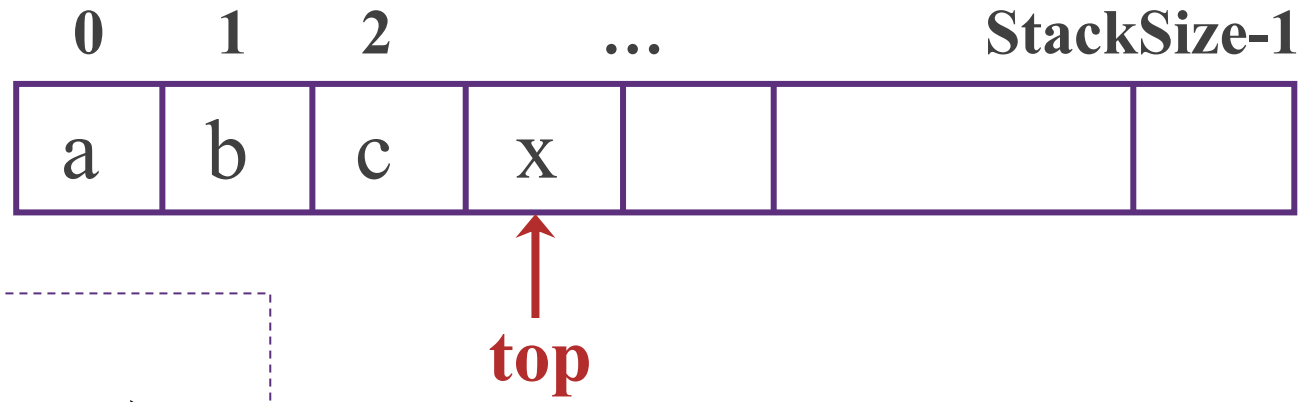
Pop：出栈

GetTop：取栈顶元素

Empty：判空

```
const int StackSize = 10;
template <typename DataType>
class SeqStack
{
public:
        SeqStack();
        ~SeqStack();
        void Push(DataType x);
        DataType Pop();
        DataType GetTop();
        int Empty();
private:
        DataType data[StackSize];
        int  top;
};
```

3. 顺序栈的实现——入栈



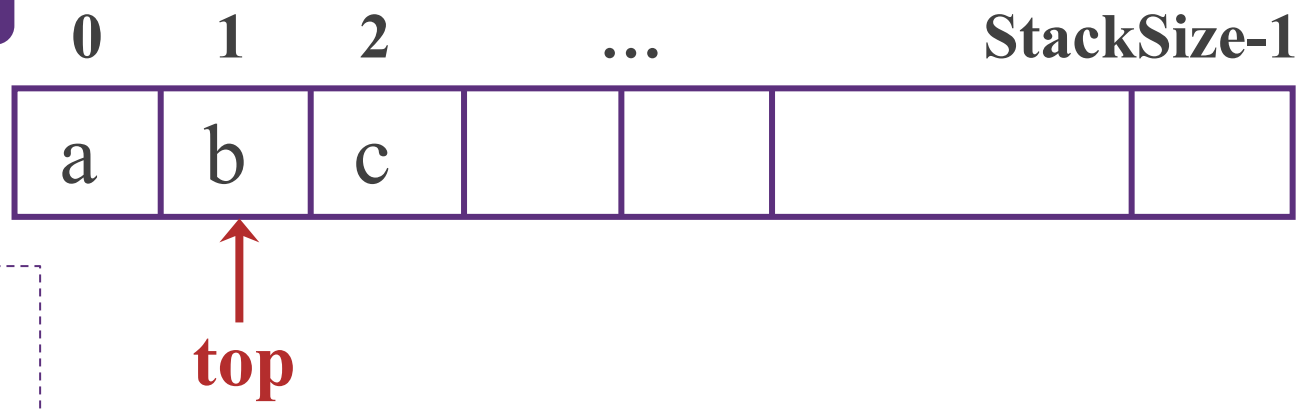

```
template <typename DataType>
void SeqStack<DataType> :: Push(DataType x)
{
    if (top == StackSize - 1) throw "上溢";
    data[++top] = x;
}
```

栈满：top=StackSize−1

什么情况下无法入栈？

时间复杂度？

4. 顺序栈的实现——出栈




```
template <typename DataType>
DataType SeqStack<DataType> :: Pop( )
{
    DataType x;
    if (top == -1) throw "下溢";
    x = data[top--];
    return x;
}
```

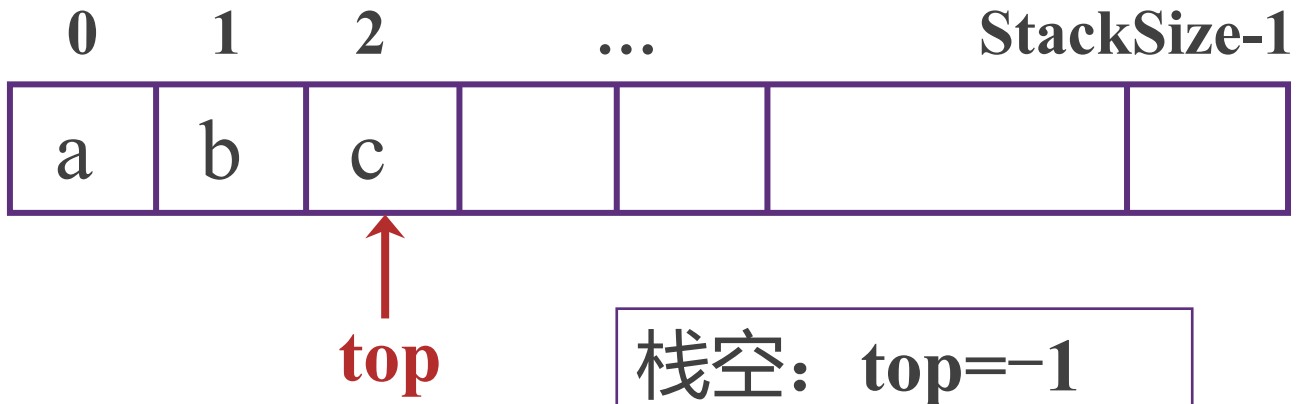
栈空：top=−1

什么情况下无法出栈？

取栈顶元素的实现？

5. 顺序栈的实现——判空

判空的函数原型是什么？



栈空：top=−1

Empty

输入：无

功能：判断栈是否为空

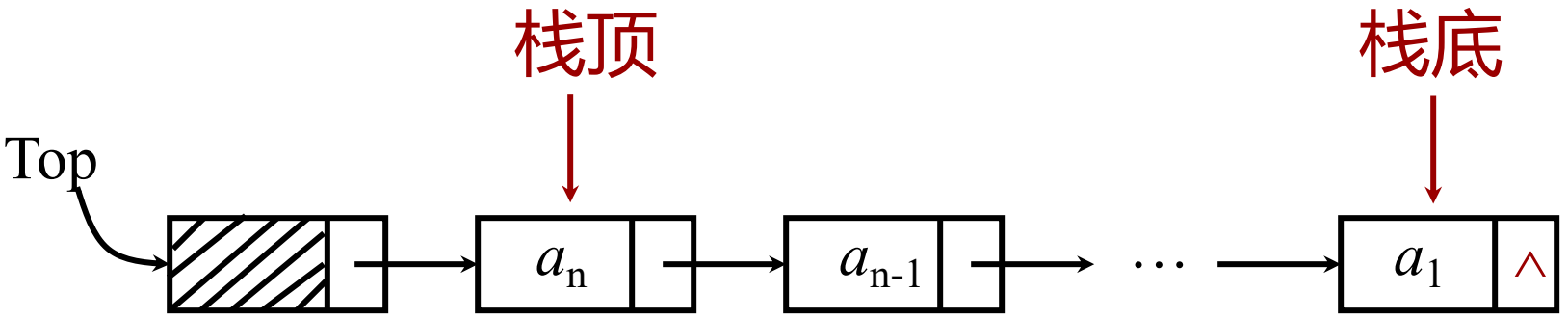
输出：如果栈为空，返回 1；否则，返回 0

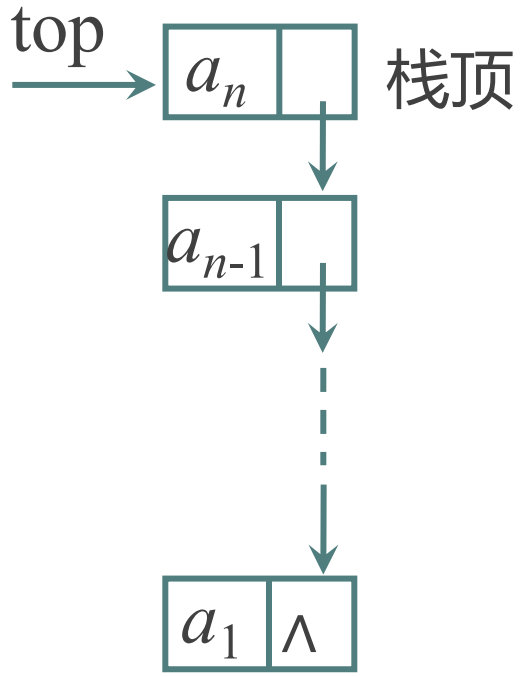
```
template <typename DataType>
int  SeqStack<DataType> :: Empty( )
{
    if (top == -1) return 1
    else return 0;
}
```

3.2 栈

3-2-3　栈的链接存储结构及其实现

1. 链栈的存储结构定义



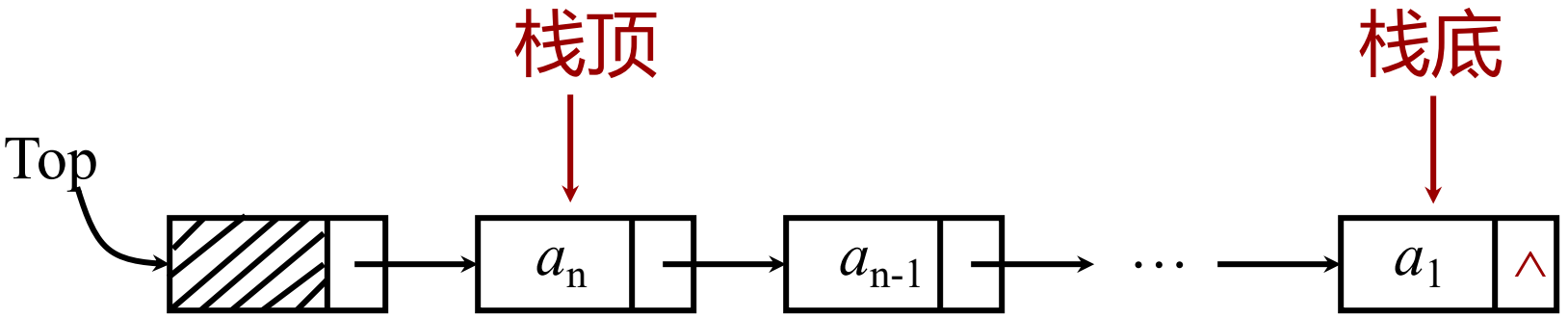


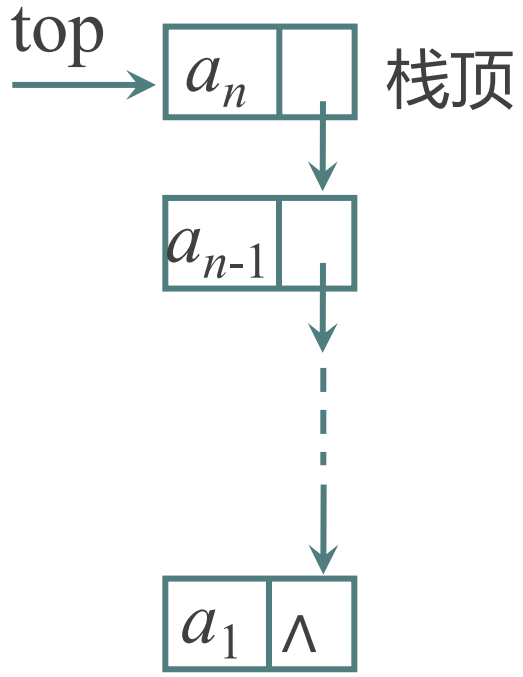

```
template <typename DataType>
struct Node
{
        DataType data;
        Node<DataType>*next;
};
```

P：链栈需要加头结点吗？

P：单链表为什么加头结点？

1. 链栈的存储结构定义






```
template <typename DataType>
struct Node
{
        DataType data;
        Node<DataType>*next;
};
```

单链表→栈

```
template <typename DataType>
class LinkStack
{
public:
        LinkStack();
        ~LinkStack();
        void Push(DataType x);
        DataType Pop();
        DataType GetTop();
        int Empty();
private:
        Node<DataType> * top;
};
```

1. 链栈的存储结构定义

ADT 定义

InitStack：栈的初始化

DestroyStack ：栈的销毁

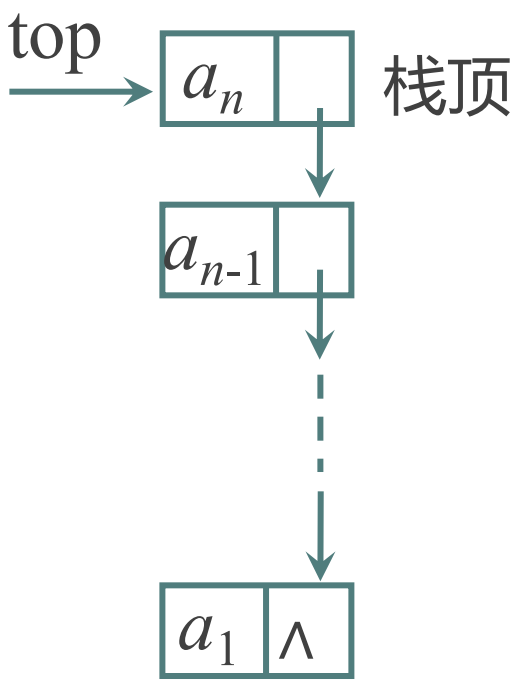
Push：入栈

Pop：出栈

GetTop：取栈顶元素

Empty：判空

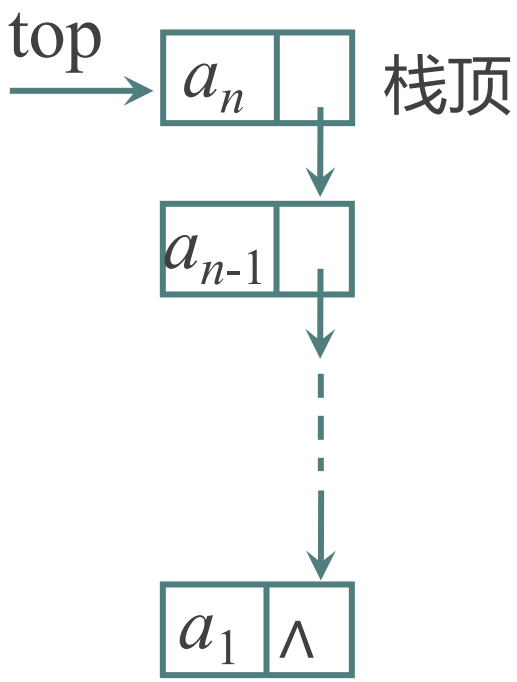
```
template <typename DataType>
class LinkStack
{
public:
        LinkStack();
        ~LinkStack();
        void Push(DataType x);
        DataType Pop();
        DataType GetTop();
        int Empty();
private:
        Node<DataType> * top;
};
```



2. 链栈的初始化、判空、销毁

```
template <typename DataType>
LinkStack<DataType>::LinkStack()
{
        top = nullptr;
}
```

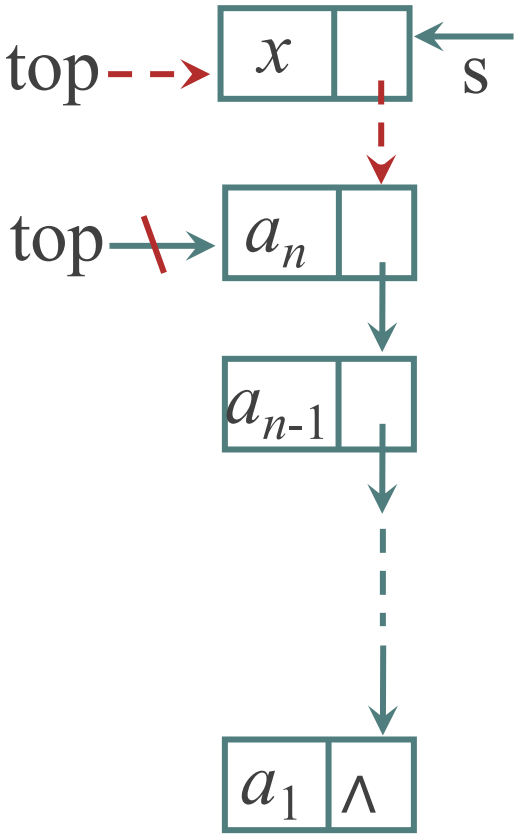
```
template <typename DataType>
LinkStack<DataType>::~LinkStack()
{
        cout<<"程序退出，析构函数被调用!"<<endl;
        while (!Empty())
        {
                cout<<"出栈元素: "<<Pop()<<endl;
        }
        cout<<"程序退出链栈已清空!"<<endl;
}
```




```
template <typename DataType>
int LinkStack<DataType>::Empty()
{
        if(top == nullptr) return 1;
        return 0;
}
```

3. 链栈的实现---入栈

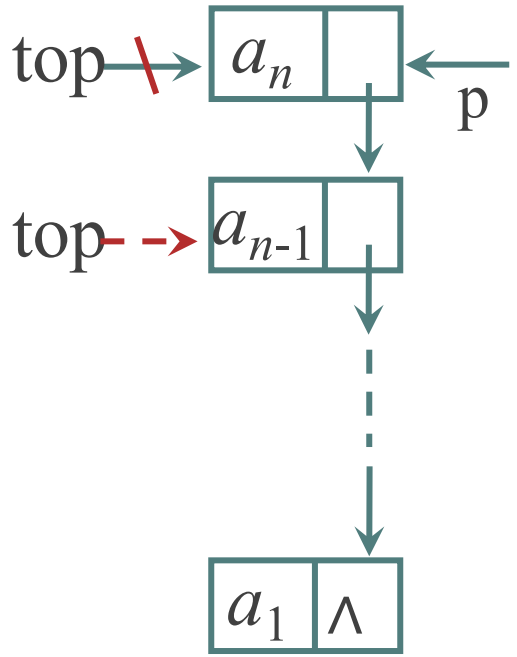
```
template <typename DataType>
void LinkStack<DataType> :: Push(DataType x)
{
    Node<DataType> *s = nullptr;
    s = new Node<DataType>;
    s->data = x;                          //申请结点s数据域为x
    s->next = top;
    top = s;                              //将结点s插在栈顶
}
```



链栈的入栈操作为什么不用判断是否栈满？

4. 链栈的实现---出栈

```
template <typename DataType>
DataType LinkStack<DataType> :: Pop( )
{
    Node<DataType> *p = nullptr;
    DataType x;
    if (top == nullptr) throw "下溢";
    x = top->data; p = top;                    //暂存栈顶元素
    top = top->next;                           //将栈顶结点摘链
    delete p;
    return x;
}
```



空栈　top = nullptr

什么情况下无法删除？

取栈顶元素的实现？

5. 链栈的使用

```
int main()
{
	LinkStack<int> S;
	cout<<"将15和10压栈"<<endl;
	S.Push(15);
	cout<<"进栈元素："<<S.GetTop()<<endl;
	S.Push(10);
	cout<<"进栈元素："<<S.GetTop()<<endl;

	cout<<"出栈元素："<<S.Pop()<<endl;
	cout<<S.Empty()<<endl;
	cout<<"出栈元素："<<S.Pop()<<endl;
	cout<<S.Empty()<<endl;
	return 0;
}
```

小结

顺序栈和链栈的比较

时间性能：O(1)

空间性能：

顺序栈：必须确定固定长度，所以有存储元素个数的限制和浪费空间的问题。

链栈：没有栈满的问题，只有当内存没有可用空间时才会出现栈满，但是每个元素都需要一个指针域，产生结构性开销。

当使用栈的过程中元素个数变化较大时，采用链栈，反之采用顺序栈。

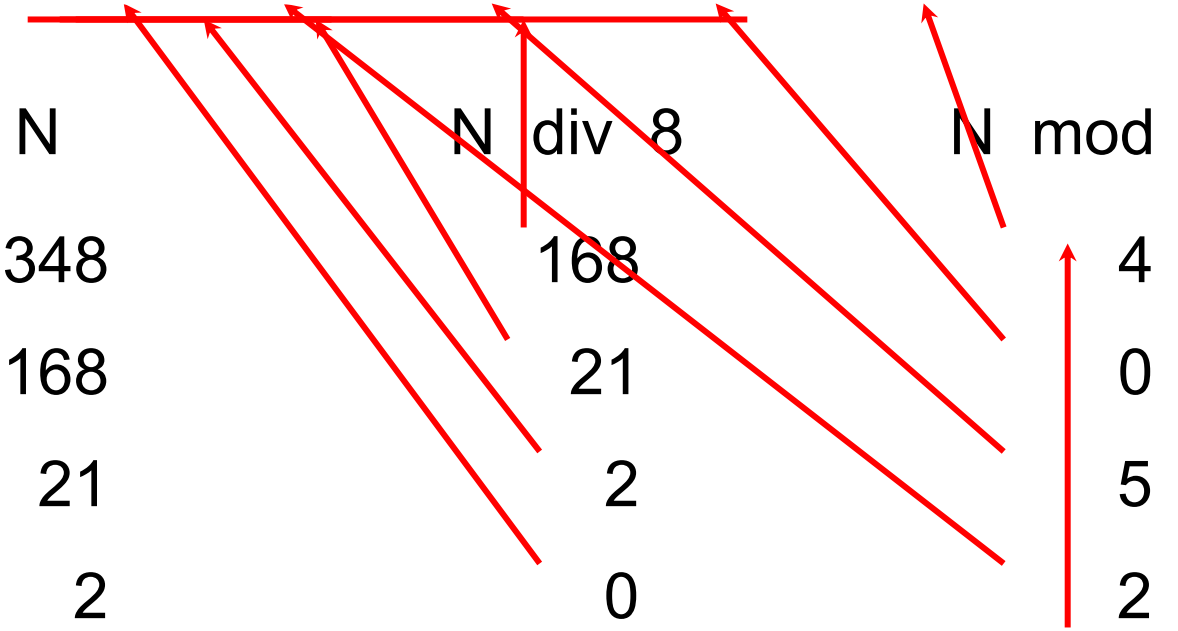
作业　算法：数制转换

1：设计算法，把十进制整数转换为二至九进制之间的任一进制输出。

例，$(1348)_{10}=(2504)_8$　　$(1348)_{10}$ = (((0×8+2)×8+5)×8+0)×8+4

一般性转换：　　N=(N div d)×d+N mod d

8



$(1348)_{10} = (2504)_8$

算术表达式 4 + 2 * 3 - 10 / 5 求值

4 + 2 * 3 - 10 / 5

= 4 + 6 - 10 / 5

= 10 - 10 / 5

= 10 - 2

= 8

思想: 使用栈存放操作符

栈顶操作符与当前输入操作符比较

优先级低，新操作符进栈

优先级高，栈顶操作符出栈，计算结果

算法： 表达式求值

为了方便计算机实现，增加界符# ， #优先级最低。

4 + 2 * 3 - 10 / 5

操作符和界符统称为运算符

其他为操作数(运算数)

任意两个相继的运算符θ_1 和θ_2之间存在优先关系:

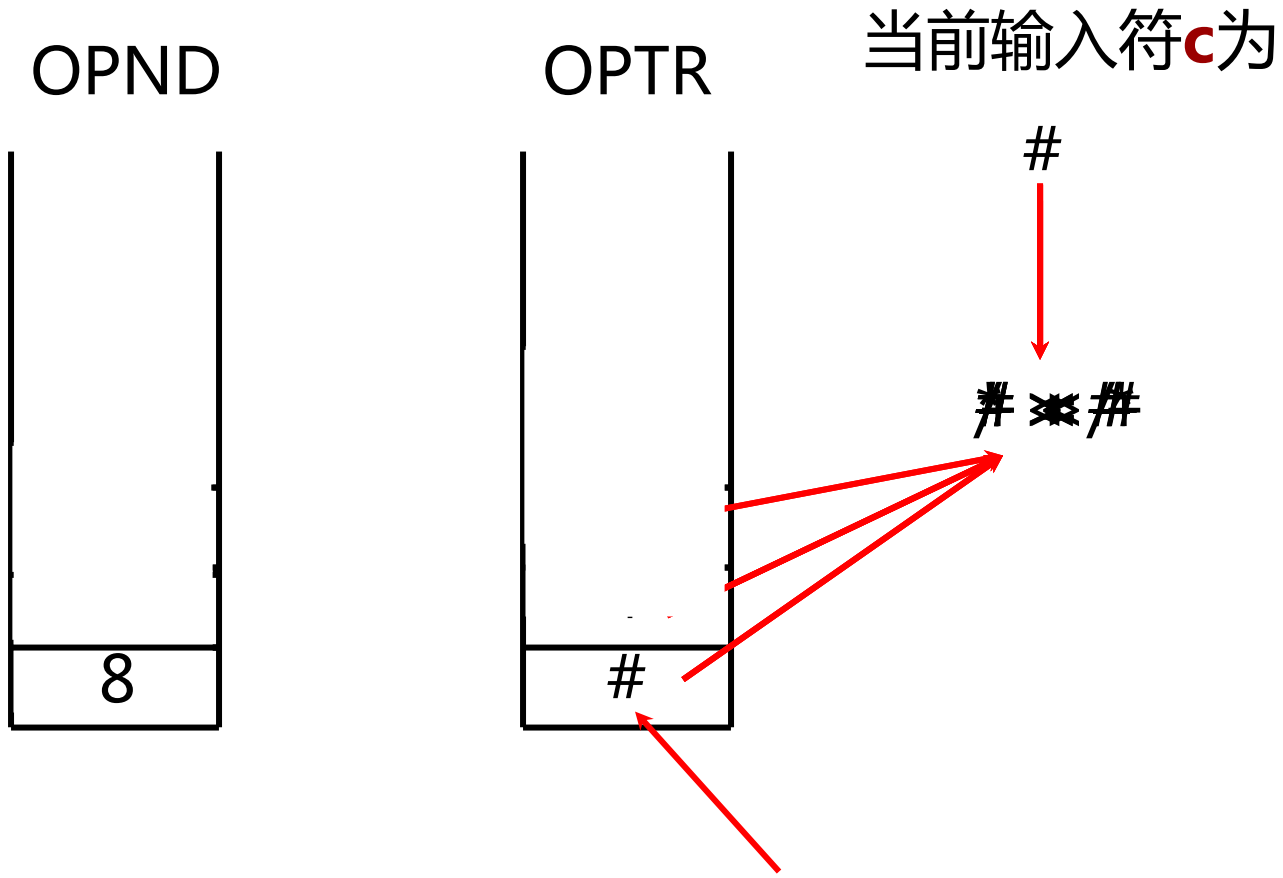
$\theta_1 < \theta_2$ θ_1的优先权低于θ_2

$\theta_1 = \theta_2$ θ_1的优先权等于θ_2

$\theta_1 > \theta_2$ θ_1的优先权高于θ_2

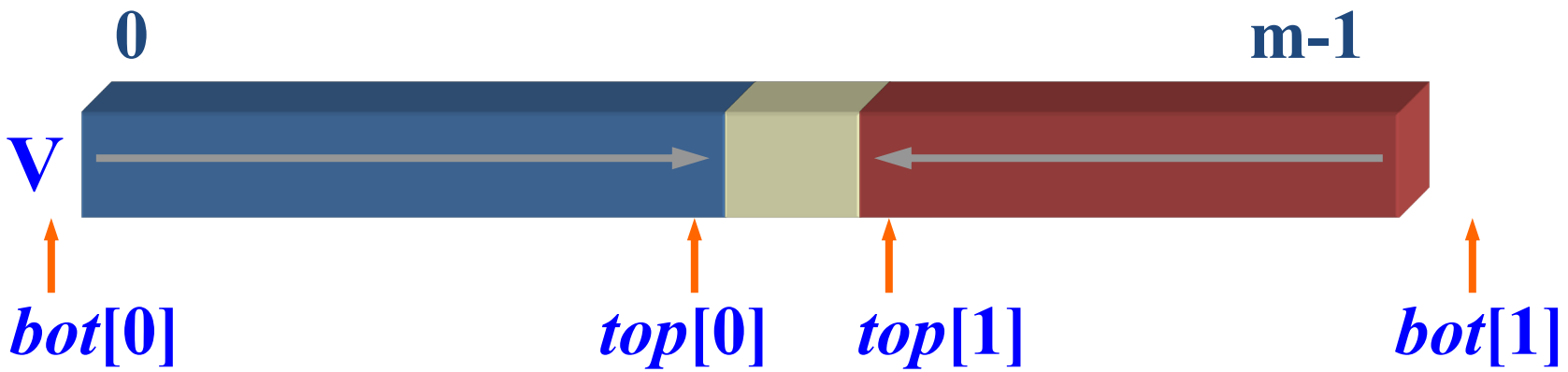
扩展　算法： 表达式求值

4 + 2 * 3 - 10 / 5 #



扩展

双栈共享一个栈空间



栈空：

top[0] = -1

top[1] = m

优点：互相调剂，灵活性强，减少溢出机会

试编写判断栈空、栈满、进栈和出栈四个算法的函数

栈空：top[i] == bot[i] i表示栈的编号

栈满：top[0]+1==top[1] 或top[1]-1==top[0]

```
int Dblpop(DblStack &s,int i,SElemType &x) ;
void Dblpush(DblStack &s,SElemType x,int i) ;
```

Thank You !

Q & A