2.栈

栈

栈(stack)是一种特殊的线性结构，它只能在一端进行插入和删除操作。

能插入和删除的一端栈顶（top），另一端称为栈底 (bottom)。

不含任何元素的栈称为空栈。

只允许在栈顶进行插入和删除，所以栈的操作是按“后进先出”（Last In First Out）的原则进行的。

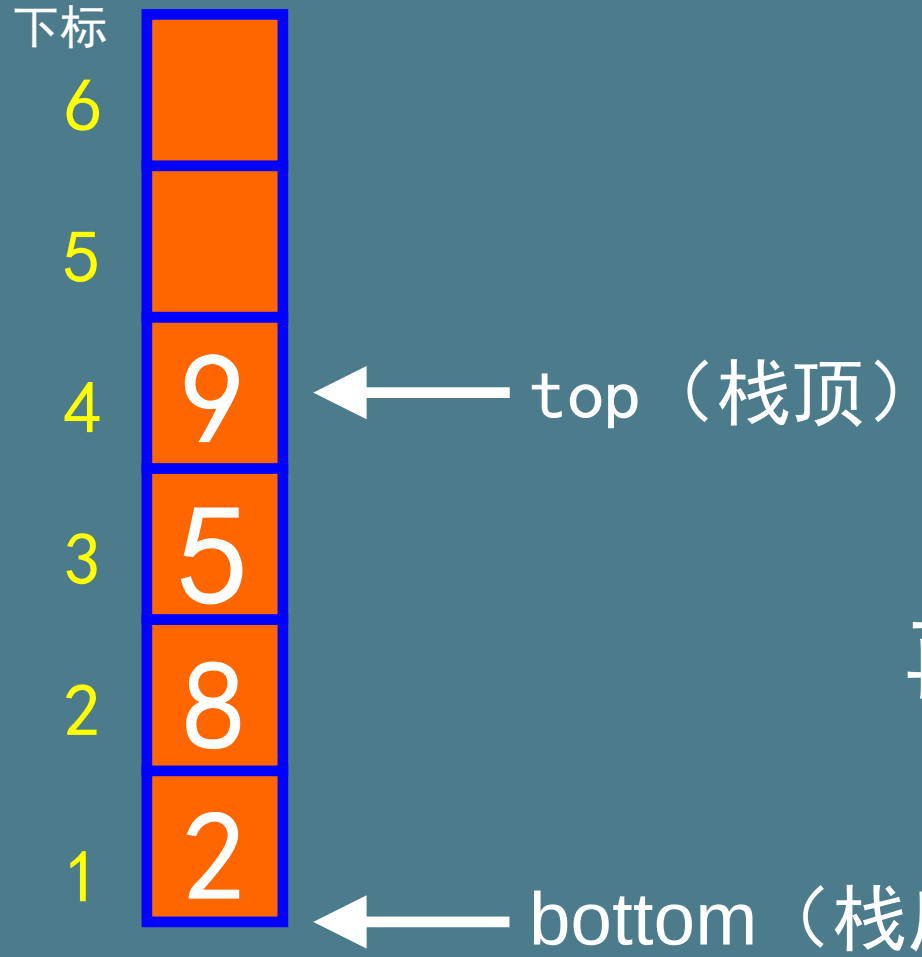
栈的实例1：汉诺塔

栈的实例2：弹夹

用数组模拟栈的“后进先出”

数组（栈）

栈为空的条件是：top==bottom



加入一个数　7

取出栈顶元素

再取出栈顶元素

top始终指向栈顶

```
#define maxn 100
int Stack[maxn+1];
int top;
```

```
//插入（压栈）
void push(int x)
{
  if(top==maxn) cout<<"full";
  else
  {
        top++;
        Stack[top]=x;
   }
}
```

```
//删除（弹栈）
void pop()
{
  if(top==0) cout<<"empty";
  else
  {
        cout<<Stack[top];
        top--;
  }
}
```

考题（初赛）

若S是一个大小为4的栈，若元素1，2，3，4，5，6，7按顺序依次进栈，则这7个元素的出栈顺序可能为（　　）

A.1,2,3,4,5,6,7

B.1,4,3,5,7,2,6

C.1,3,2,4,7,5,6

D.2,3,7,6,5,4,1

试题（初赛）

- **设栈S和队列Q**初始状态为空，元素1 ，2 ，3 ，4 ，5 ， 6依次通过栈S，一个元素出栈后即进入队列Q，若出队的顺序为2 ，4 ，3 ，6 ，5 ，1 ，则栈S的容量至少应该为（ ）.

  A)2 B)3 C)4 D)5

stack

stack 栈

stack(栈)。是项的有限序列，并满足序列中被删除、检索和修改的项只能是最近插入序列的项。即按照后进先出的原则。

stack<数据类型> 栈名； 声明栈，例如：

```
stack<int>s;
```

常用函数：

push(e) - 将元素e从栈顶压入栈，例如：`s.push(5);`

pop() - 删除栈顶元素，例如：`s.pop();`

top() - 返回栈顶元素的值，例如：`cout<<s.top();`

size()-栈中元素的个数，例如：`cout<<s.size();`

empty()-判断栈是否为空，例如：`s.empty();`

stack 栈

```
int main ()
{
    stack<int> sk;                  //申明一个int类型的stack变量sk
    int sum =0;
    for (int i=1;i<=10;i++) sk.push(i);   //将数字1到10入栈
    cout<<sk.size();                      //输出栈中元素个数10
    while (!sk.empty())                   //只要栈不为空
    {
        sum += sk.top();                  //将栈顶元素的值累加
        sk.pop();                         //栈顶元素出栈
    }
  cout << sum << endl;
}
```

栈的实例3：括号匹配问题

我们做小学数学题时，数学式子里往往会加很多括号，有时候这些括号可能出错。比如下面式子：

```
32 x [ 56 + ( 72 - 65 ] x 25 )          括号不匹配
32 x [ 56 + ( 72 - 65 x 25 ) ]          括号匹配
```

现在我们只关心括号是否匹配，所以，我们可以把式子中的括号单独提出来

```
[ ( ] )
[ ( ) ]
```

对于给出的由‘{’ ‘}’ ‘[’ ‘]’ ‘(’ ‘)’ 构成的长度不超过100000的括号序列，请你快速回答里面的括号是否匹配

输入格式：

一行，一个括号序列

输出格式：

一行，一个单词"yes"或者"no"

样例输入1：

```
([{()[]}[()](())])
```

样例输出1：

```
yes
```

样例输入2：

```
([{([)]}[()](())])
```

样例输出2：

```
no
```

栈的实例3：括号匹配问题

解法：从左往右扫描每一个括号，若遇到一个“右”括号，那么它一定与左边离它最近的那一个还未配对的“左”括号配对。若配对成功，则将这对括号删除，否则说明配对出错。

```
int main()
{
        stack<char>A;
        string s;
        cin>>s;
        int n=s.length();
        for(int i=0;i<n;i++)
        {
                if(s[i]=='[' || s[i]=='(' || s[i]=='{')A.push(s[i]);
                else
                {
                   if(A.size()==0){ cout<<"no";  return 0; }
                  if( (s[i]==']' && A.top()=='[') || (s[i]==')'  && A.top()=='(') || (s[i]=='}' && A.top()=='{') )A.pop();
                   else  { cout<<"no";  return 0; }
                }
        }
        if(A.size()==0)cout<<"yes";
        else cout<<"no";
        return 0;
}
```

栈的实例4：火车调度　nkoi1914

某城市有一个火车站，如下图 所示，现有 n（n < =10000）节火车车厢，顺序编号为 1,2,3,...,n，按编号连续依次从 A 方向的铁轨驶入车站，从 B 方向铁轨驶出。一旦车厢进入车站就不能再回到 A 方向的铁轨上；在车站的门口有工人可以将车厢拖出车站，工人一次只能拖一节车厢，并且只能将车厢拖入B方向的铁轨。一旦车厢出了车站就不能再回到车站。车站一开始为空，最多能停放 10000 节车厢。

为了方便装货，调度员需要将车厢从新排列，问能否将车厢编号排列成A1,A2,.....,An。也就是使车厢从B方向驶出的编号是A1,A2,.....,An。如果能输出"yes"，否则输出"no"。

输入格式：

第一行，一个整数n

第二行，n个用空格间隔的整数，表示出站时车厢编号要排列成的顺序A1,A2,.....,An

输出格式：

一行，一个单词"yes"或者"no"

样例输入1：

```
5
3 2 5 4 1
```

样例输出1：

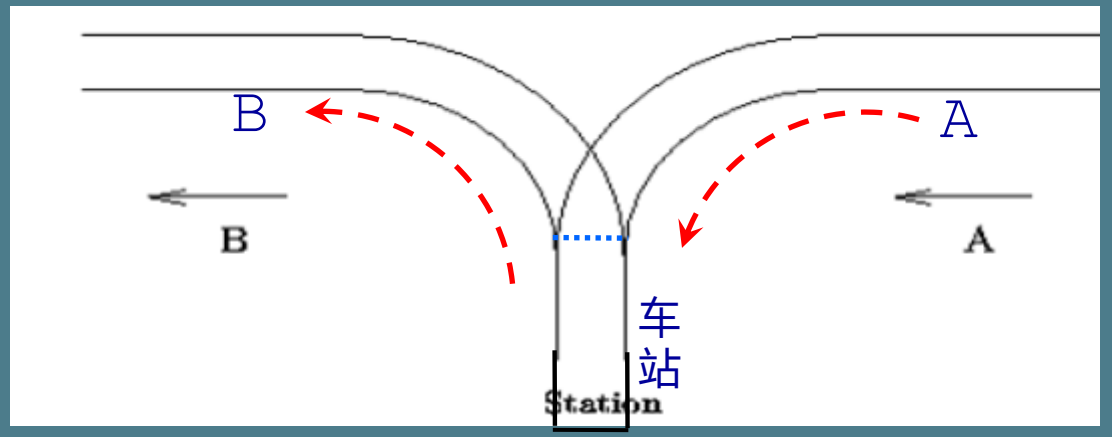
```
yes
```

样例输入2：

```
5
3 1 5 4 2
```

样例输出2：

```
no
```



```
//将题目要求的出站序列读入a数组，然后通过栈从左到右依次去匹配a数组
#include <iostream>
#include <cstdio>
#include <stack>
using namespace std;
int a[10001],n,i,j,top=0;
stack<int>s;
int main()
{
	scanf("%d",&n);
	for(i=1;i<=n;i++)scanf("%d",&a[i]);
	i=j=1;
	while(i<=n)                  //依次讨论火车进站的编号1到n
	{                            //将第i辆车入站(栈)，把编号存入栈顶
		s.push(i);
		//while讨论如果栈顶的编号与当前要匹配的a的编号相同，则表示可以出站
		while(s.size( )>0 && s.top()==a[j] ){ s.pop(); j++; }
		i++;                     //讨论下一辆车
	}                            //如果栈不为空，表示有编号无法匹配
	if(s.size()==0)printf("yes\n"); else printf("no\n");
}
```

```
//手工栈版本的代码
//将题目要求的出站序列读入a数组，然后通过栈从左到右依次去匹配a数组
#include <iostream>
#include <cstdio>
using namespace std;
int s[10001],a[10001],n,i,j,top=0;   //s数组用于模拟栈
int main()
{
      scanf("%d",&n);
      for(i=1;i<=n;i++)scanf("%d",&a[i]);
      i=j=1;
      while(i<=n)   //依次讨论火车进站的编号1到n
      {                        //将第i辆车入站(栈)，把编号存入栈顶
            top++;
            s[top]=i;   //while讨论如果栈顶的编号与当前要匹配的a的编号相同，则表示可以出站
            while((s[top]==a[j])&&(top>0)){top--;j++;}
            i++;          //讨论下一辆车
      }                       //如果栈不为空，表示有编号无法匹配
      if(top==0)printf("yes\n"); else printf("no\n");
      return 0;
}
```

课后练习：网页浏览 NKOI1085

网页浏览器都包含有前进和后退功能，以此来快速打开之前你访问过的网页。你的任务就是实现这个功能。实现此功能的常用方法是通过forword和back两个栈来保存前进和后退时要打开的网页。下列命令是你需要实现的：

BACK：把当前显示的网页压入到forword栈中。然后把back栈栈顶的网页弹出作为当前显示的网页。如果back栈为空，那么这条命令就不执行。

FORWARD：把当前显示的网页压入到back栈中。然后把forward栈栈顶的网页弹出作为当前显示的网页。如果forward栈为空，那么这条命令就不执行。

VISIT ：把当前显示的网页压入到back栈中。然后浏览器显示用户新输入的网址对应的网页。清空forward栈

QUIT：关闭浏览器

假设只要浏览器一打开就会自动打开网页http://www.acm.org/

输入格式：

包含若干条命令，这些命令由BACK,FORWARD,VISIT和QUIT构成。每条命令占一行，长度不超过70。命令的总条数不超过100。一旦出现QUIT表示结束输入命令。

输出格式：

对于每个命令，如果它不是QUIT，那么输出命令执行后浏览器显示的网页。如果该命令无法执行，则输出 "Ignored"，除QUIT命令外，每一个输入命令对应一行输出结果。

样例输入

```
VISIT http://acm.ashland.edu/
VISIT http://acm.baylor.edu/acmicpc/
BACK
BACK
BACK
FORWARD
VISIT http://www.ibm.com/
BACK
BACK
FORWARD
FORWARD
FORWARD
QUIT
```

样例输出

```
http://acm.ashland.edu/
http://acm.baylor.edu/acmicpc/
http://acm.ashland.edu/
http://www.acm.org/
Ignored
http://acm.ashland.edu/
http://www.ibm.com/
http://acm.ashland.edu/
http://www.acm.org/
http://acm.ashland.edu/
http://www.ibm.com/
Ignored
```

何老板倾情翻译

3.链表

链表

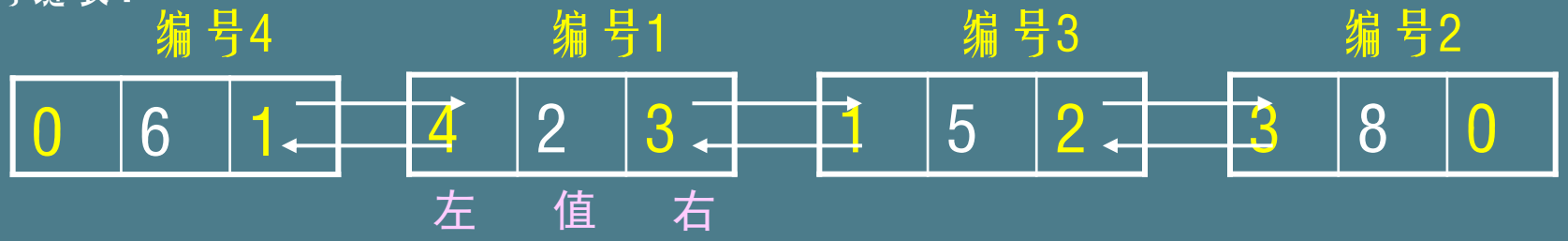
链表（List），由多个结点连接而成的链状结构。每个结点包含两部分，数值和存放结点间的相互关系

右图中，每个人看做一个节点，数值就是每个人自己的名字。同时记录下左边是谁，右边是谁，这就是节点间的关系。

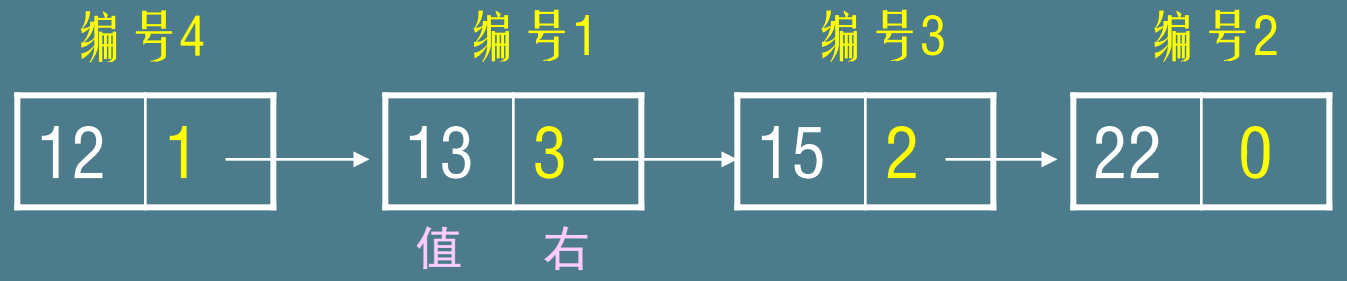
左：无 右：甲	左：丙 右：戊	左：甲 右：乙	左：戊 右：丁	左：乙 右：...

链表（List），由多个结点连接而成的链状结构。每个结点包含两部分，数值和存放结点间的相互关系。

双向链表：



单向链表：

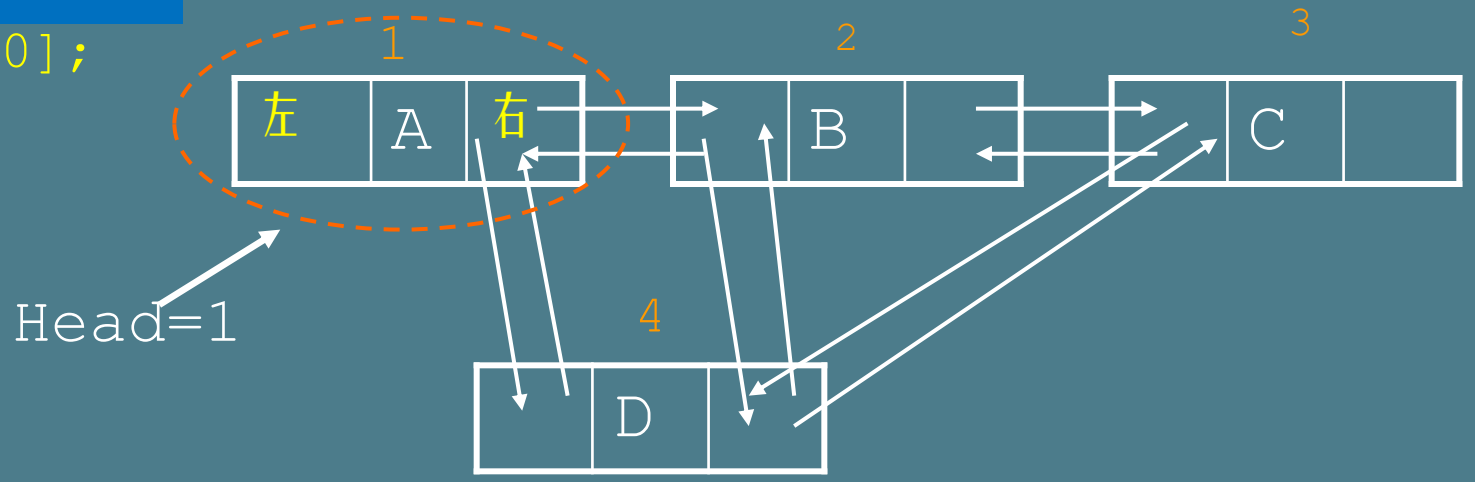


```
struct node
{
    char name;
    int left,right;
};
node List[100];
```

```
List[1].right=2;
List[1].left=0;
List[1].name='A';

List[2].right=3;
List[2].left=1;
List[2].name='B';

List[3].right=0;
List[3].left=2;
List[3].name='C';
```



在1和2间插入4号点D

1. 让4的左手牵2的左手牵的对象
   `List[4].left=List[2].left;`
2. 让4的右手牵1的右手牵的对象
   `List[4].right=List[1].right`
3. 让1的右手放开2，然后牵4
   `List[1].right=4;`
4. 让2的左手放开1，然后牵4
   `List[2].left=4;`

删除 B

1. 让4的右手牵2的右手牵的对象
   `List[4].right=List[2].right;`
2. 让3的左手牵2的左手牵的对象
   `List[3].left=List[2].left;`
3. 让2的右手放开
   `List[2].right=0;`
4. 让2的左手放开
   `List[2].left=0;`

链表的 代码实现

```
struct node
{
        string name;
        int left,right;
};
node List[100];
```

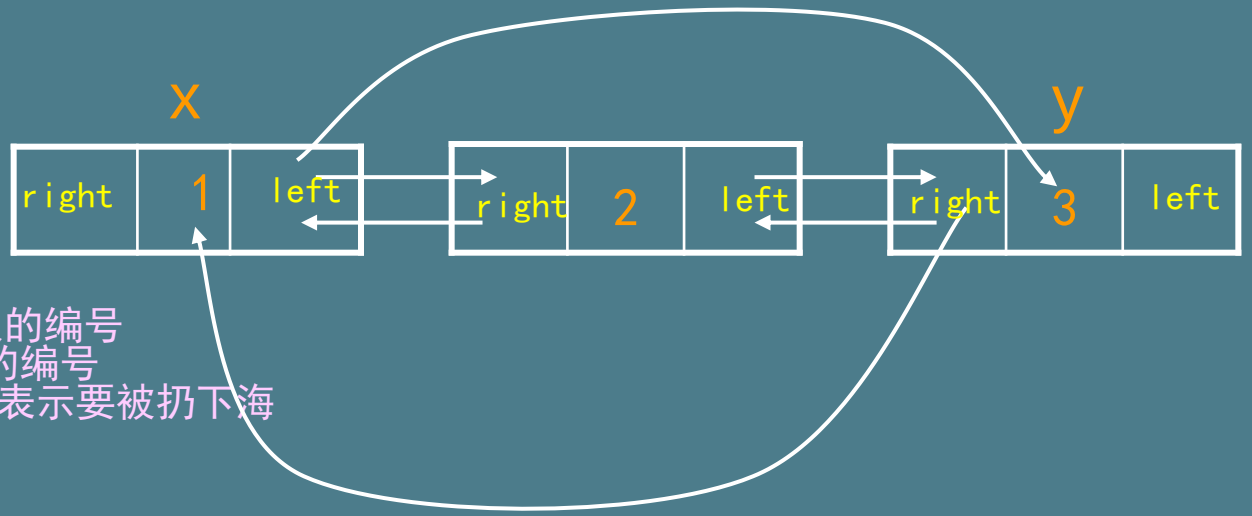
```
//删除编号为x的结点
void del(int x)
{
  int p,q;
  if(head==x)head=List[x].left;
  p=List[x].left;
  q=List[x].right;
  List[p].right=List[x].right;
  List[q].left=List[x].left;
  List[x].left=0;
  List[x].right=0;
}
```

```
//把编号x的元素插到p号点之后
void insert(int p,int x)
{
    List[x].right=List[p].right;
    List[x].left=p;
    List[List[p].right].left=x;
    List[p].right=x;
}
```

```
//查找链表中的名为tom的结点的编号
void searchList(string s)
{
  int p;
  p=head;
  while((p!=0) && (List[p].name!= “tom"))p=List[p].right;
  if(List[p].name== “tom") cout<<p;
  else cout<<"not found";
}
```

17世纪的法国数学家加斯帕在《数目的游戏问题》中讲了这样一个故事：25个基督教徒和25个非教徒在深海上遇险，必须将一半的人投入海中，其余的人才能幸免于难，于是想了一个办法：50个人围坐成一圆圈，从第一个人开始顺时针依次报数，每数到第九个人就将他扔入大海，如此循环进行直到仅余25个人为止。问怎样排座位，才能使每次投入大海的都是非教徒。

要删除2号节点




```
struct people
{                          //right为上一个人的编号
       int left,right;//left为下一个人的编号
       bool dead;      //如果dead为true表示要被扔下海
};
people ren[51];
int i,n,p,x,y;
int main()
{
    for(i=1;i<=50;i++)
    {
        ren[i].left=i+1;
        ren[i].right=i-1;
        ren[i].dead=false;
    }
    ren[1].right=50;    ren[50].left=1;
    n=50; p=50;
    while(n>25)
    {
         for(i=1;i<=9;i++) p=ren[p].left;
         x=ren[p].right;   y=ren[p].left;
         ren[x].left=y;
         ren[y].right=x;
         ren[p].dead=true;
         n--;
    }
    for(i=1;i<=50;i++)
      if(ren[i].dead==false)cout<<i<<" ";

}
```