计算机程序设计_数据结构

线性结构

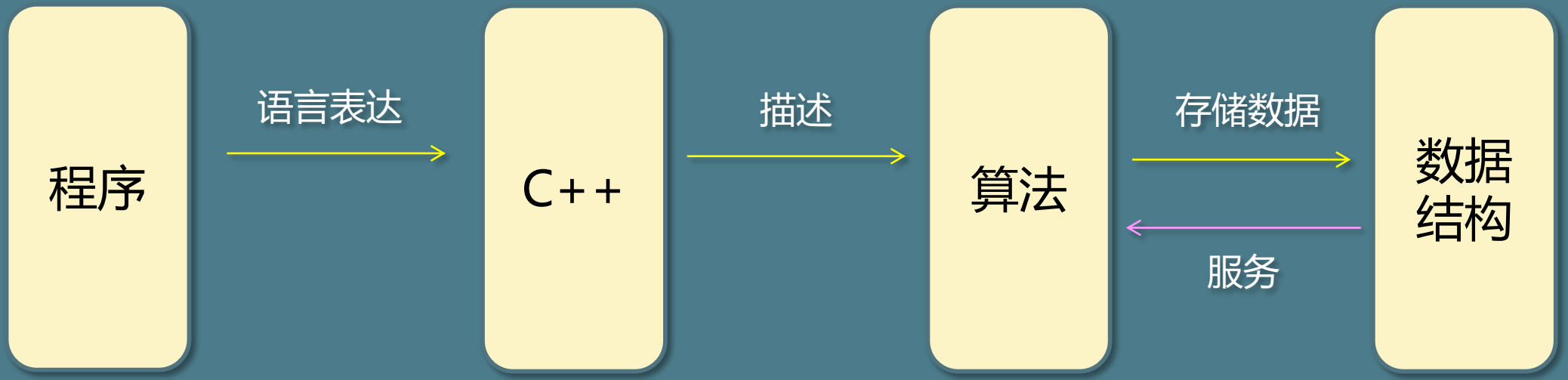
Which Challenges Are the Most Popular?

Percentage of HackerRank Tests Taken By Type

Rank	Domain	Percent of Tests
1	Algorithms	39.5%
2	Java	9.3%
3	Data Structures	9.1%
4	C++	6.6%
5	Tutorials	6.5%
6	Mathematics	6.1%
7	Python	5.3%
8	SQL	4.6%
9	Shell	3.1%
10	Artificial Intelligence	2.9%
11	Functional Programming	2.5%
12	Databases	1.5%
13	Ruby	1.0%
14	Distributed Systems	1.0%
15	Security	0.9%
	Total	100.0%

什么是数据结构?

数据结构(Data Structure),用于描述计算机中数据的存储、组织形式。合理的数据结构可以给程序带来更高的存储和运行效率。



什么是数据结构?

数据结构(Data Structure),用于描述计算机中数据的存储、组织形式。合理的数据结构可以给程序带来更高的存储和运行效率。

常用的数据结构有哪些?

1.线性结构　栈、队列、链表

2.树型结构

3.图型结构

1.队列

队列

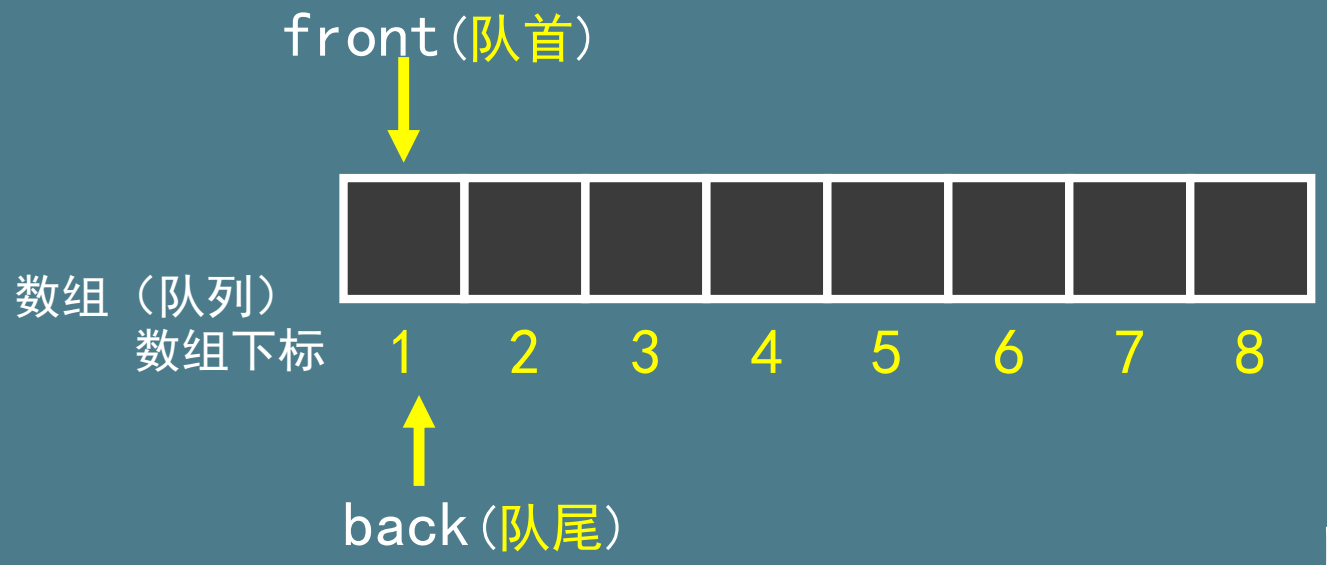
队列（queue）是另一种特殊的线性表，它的特点是删除操作在一头进行，插入操作在另一头进行。

允许删除的一端称为队首（front），允许插入的一端称为队尾(back)。

不含任何元素的队称为空队。

队列的修改是按先进先出(First In First Out)的原则进行

用数组模拟队列的“先进先出”



当front==back
时表示队列为空

1. 插入数据： C
2. 插入数据： F
3. 插入数据： X
4. 删除数据
4. 删除数据
5. 插入数据： M

队列的代码实现


```
#define maxn 101
char Queue[maxn];
int front,back;
```

```
//入队
void insert(char x)
{
  if(back>maxn)cout<<"full";
  else
  {
        Queue[back]=x;
        back++;
   }
}
```

```
//出队
void del()
{
if(back==front)cout<<"empty";
  else
  {
        cout<<Queue[front];
        front++;
    }
}
```

queue

queue 队列

queue（队列），插入只可以在尾部进行，删除、检索和修改只允许从头部进行。按照先进先出的原则。

```
#include <queue>
```

常用函数:

push(e) - 将元素e加入队列尾部
pop() - 将队首元素删除
front() - 返回队首元素的值
back() - 返回队尾元素的值
size()–队列中元素的个数
empty()–判断队列是否为空

queue 队列

```
#include <iostream>
#include <queue>
using namespace std;

int main ()
{
    queue<int> que;                          //申明一个int类型的queue变量que
    int sum =0;
    for (int i=1;i<=10;i++) que.push(i);     //将数字1到10入队
    cout<<que.size();                        //输出队中元素个数10
    while (que.empty()==false)               //只要队不为空
    {
        sum += que.front();                  //将队首元素的值累加
        que.pop();                           //队首元素出队
    }
    cout << sum << endl;
}
```

【例1】舞会 nkoi3629

在新年舞会上，n名男士(编号1到n)和m名女士(编号1到m)，各自排成一队。跳舞开始时，依次从男队和女队的队头上各出一人配成舞伴。规定每个舞曲只能有一对跳舞者。一曲结束后，跳舞的一对舞者各自回到自己队伍的末尾。

舞会总共有k个舞曲，问每曲参与跳舞的男士和女士编号是多少？

输入：一行两队的人数和舞曲数量n,m,k

输出：k行，每行两个整数，表示对应舞曲的男士和女士的编号

输入样例：

3 4 6

输出样例：

1 1
2 2
3 3
1 4
2 1
3 2

【舞会 参考程序】

```
#include<queue>
using namespace std;
queue<int>boy;
queue<int>girl;
main()
{
      int m,n,k,i,b,g;
      cin>>n>>m>>k;
      for (i=1;i<=n;i++) boy.push(i);
      for (i=1;i<=m;i++) girl.push(i);
      for (i=1;i<=k;i++)
      {
            b=boy.front();
            g=girl.front();
            boy.pop();
            girl.pop();
            cout<<b<<" "<<g<<endl;
            boy.push(b);
            girl.push(g);
}
}
```

例2：纸牌游戏 nkoi1917

桌上有一叠纸牌，共n张牌。从位于顶端的纸牌开始从上往下依次编号为1到n。现在反复进行以下操作：把位于顶端的牌扔到，然后把新的位于顶端的牌放到整叠牌的底部。直到只剩下一张牌。输入n(<=100)，输出每次扔掉的牌的编号以及最后剩下的牌的编号。

样例输入：7

样例输出：1 3 5 7 4 2 6

```
#include <cstdio>
#include <queue>
using namespace std;
queue<int>q;                                    //申明一个存储整数的队列"q"
int main()
{
      int n,i,x;
      scanf("%d",&n);
      for (i=1;i<=n;i++)q.push(i);              //将n个数字依次加入队列
      while (q.size())                          //当队列不为空q.size()!=0
      {
            printf("%d ",q.front());                  //输出队首元素
            q.pop();                                  //将队首元素删除
            x=q.front();                              //取出队首元素
            q.push(x);                            //将队首元素加入队尾
            q.pop();                                  //删除队首元素
      }
}
```

```
//手工队列版本的代码
int q[201],front,back,i,n;
int main()
{
    cin>>n;
    for(i=1;i<=n;i++)q[i]=i;          //队列赋初值
    front=1;                          //队首指针指向位于顶端的牌的位置
    back=n+1;                         //对尾指针指向下一个空位
    while(front<back)                 //如果队列不为空
    {
        printf("%d ",q[front]);       //输出队首，即扔到最顶端的牌
        front++                       //队首指针指向新的位于顶端的牌
        q[back]=q[front];               //将位于最顶端的牌移到队尾
        back++;                         //对尾指针指向下一个可用空位
        front++;                       //队首指针指向新的位于顶端的牌
    }
   return 0;
}
```

例2：约瑟夫问题 nkoi1700

设有n个人围坐在一个圆桌周围，现从第s个人开始报数，数到第m的人出列，然后从出列的下一个人重新开始报数，数到第m的人又出列，……，如此重复直到所有的人全部出列为止。对于任意给定的n,s和m，求出按出列次序得到的n个人员的顺序表。

```
int main()
{
	int n,s,m;
	cin>>n>>s>>m;
	queue<int> q;
	for(int i=1;i<=n;i++)q.push(i);
	for(int i=1;i<s;i++)
	{
		q.push(q.front());
		q.pop();
	}
	while(q.size())
	{
		for(int i=1;i<m;i++)
		{
			q.push(q.front());
			q.pop();
		}
		cout<<q.front()<<endl;
		q.pop();
	}
}
```

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。

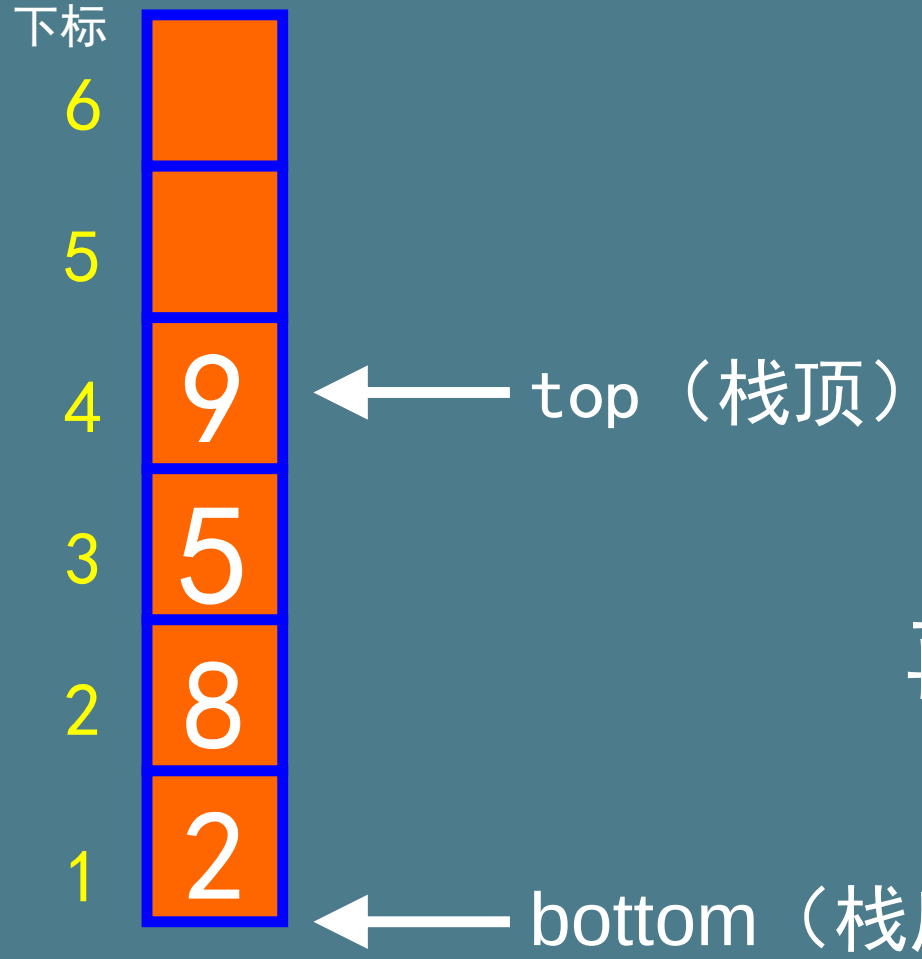
栈的实例1：汉诺塔

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--;
  }
}
```

逆波兰表达式(后缀表达式)

在通常的表达式中，二元运算符总是置于与之相关的两个运算对象之间，这种表示法也称为**中缀表示**。

波兰逻辑学家J.Lukasiewicz于1929年提出了另一种表示表达式的方法，按此方法，每一运算符都置于其运算对象之后，故称为**后缀表示**。

正常的表达式		逆波兰表达式
a+b	--->	a b +
a+(b-c)	--->	a b c - +
a+(b-c)*d	--->	a b c - d * +
a+d*(b-c)	--->	a d b c - * +

将下列式子转换为后缀表达式：

1. (a + b) * (b / c - d)
2. a / (b - c) + (d * e % f + g) * (h + k)

答案：

1. a b + b c / d - *
2. a b c - / d e f % * g + h k + * +

计算逆波兰表达式的值

我们使用一个栈S，我们将会看到该栈中最后存放的是最终的表达式的值。我们从左至右的遍历逆波兰表达式，然后按照下面的规则进行操作栈S.

（1）如果遇到的是数字，那么直接将数字压入到S中；

（2）如果遇到的是单目运算符，那么取S栈顶的一个元素进行单目运算之后，将结果再次压入到栈S中；

（3）如果遇到的是双目运算符，那么取S栈顶的两个元素进行，首先出栈的在左，后出栈的在右进行双目运算符的计算，将结果再次压入到S中；

按照上面的三个规则，遍历完整个逆波兰表达式，那么最后S中的值就是该式计算的结果了。，所以我们可以看出来使用逆波兰表达式进行求值是很简单的，只有两种操作要么是直接压栈，要么是运算之后将结果压栈。

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。

#include <stack>
常用函数：
push(v) - 将元素v从栈顶压入栈
pop() - 删除栈顶元素
top() - 返回栈顶元素的值

size()-栈中元素的个数
empty()-判断栈是否为空

stack 栈

```
#include <iostream>
#include <stack>
using namespace std;

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

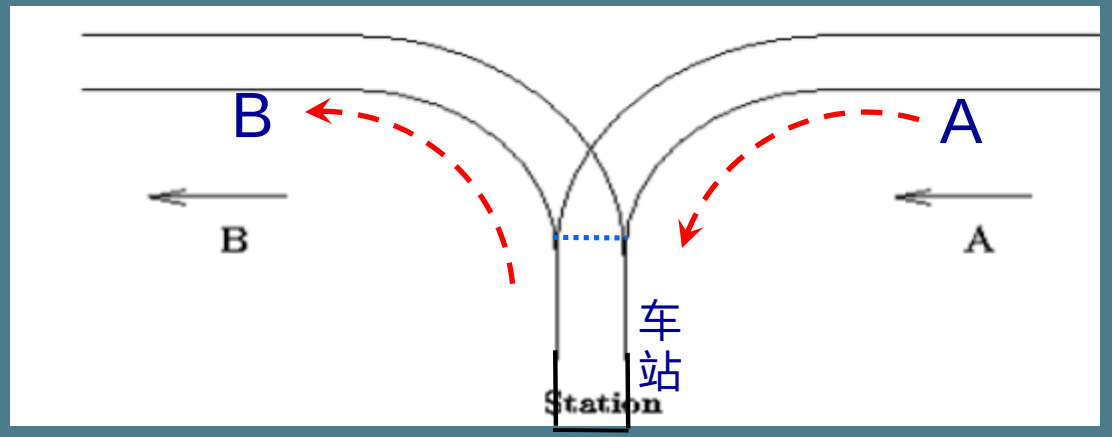
样例输入2：

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

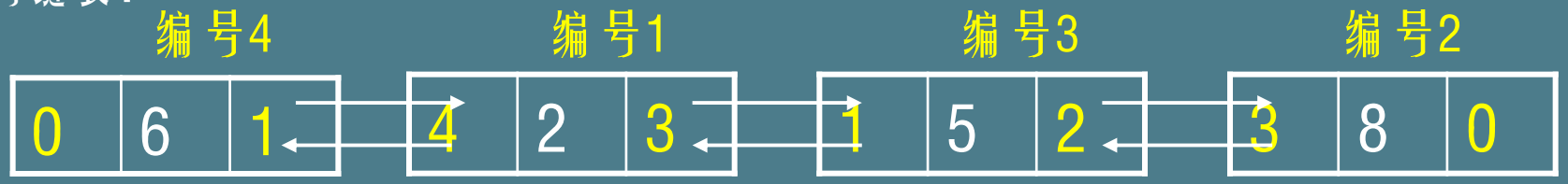
链表（list），由多个结点连接而成的链状结构。每个结点包含两部分，数值和存放结点间的相互关系

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。

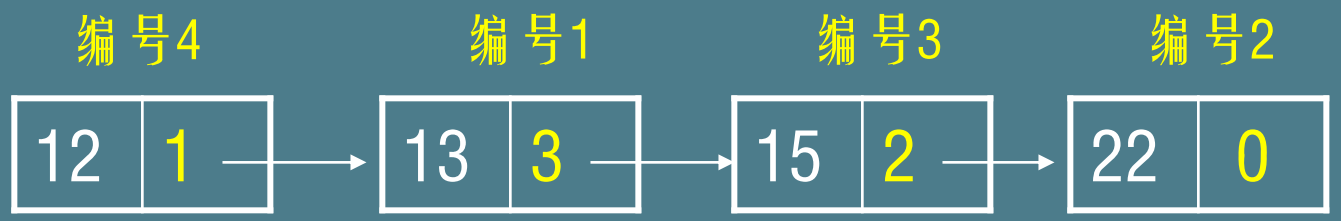
左：无 右：甲	左：丙 右：戊	左：甲 右：乙	左：戊 右：丁	左：乙 右：...

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：



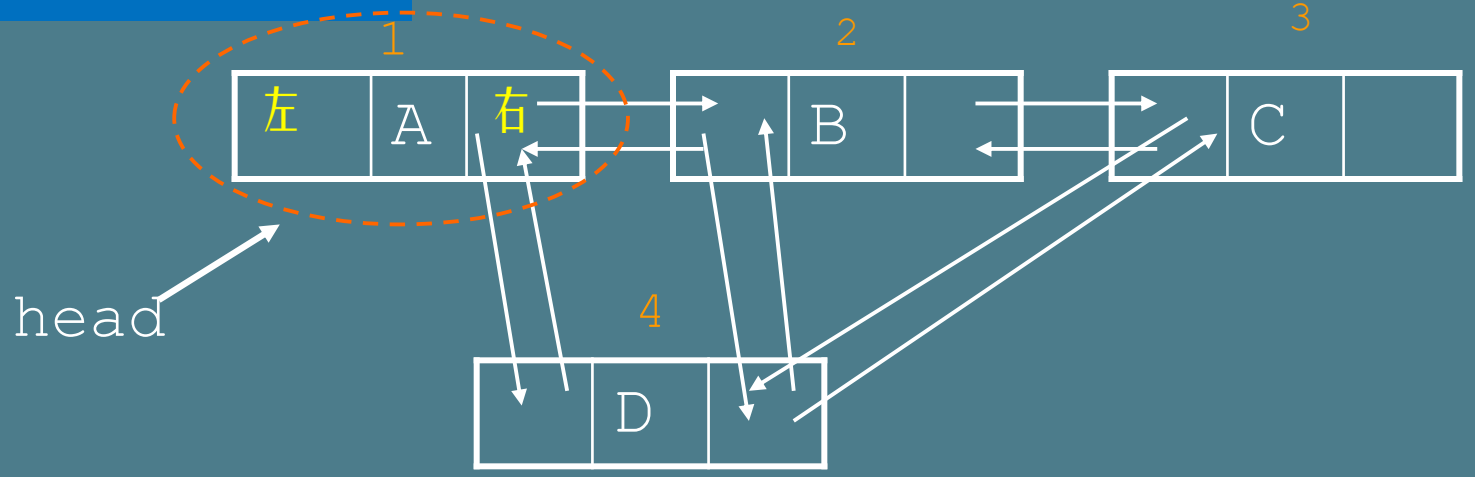
单向链表：



```
struct node
{
    char name;
    int left,right;
};
```

```
node list[100];
```

```
list[1].right=2;list[2].right=3;list[3].right=0;
list[1].left=0;list[2].left=1;
list[3].left=2;
```



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];
```

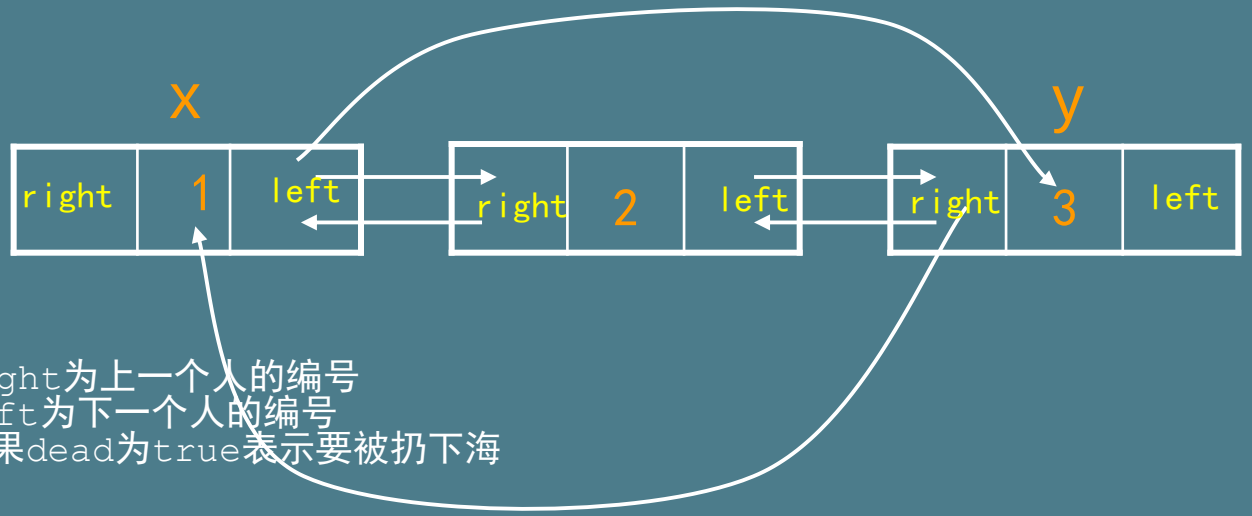
```
//删除编号为x的结点
void del(int x)
{
  int p,q;
  if(head==x)head=list[x].left;
  p=list[x].left;
  q=list[x].right;
  list[p].right=list[x].right;
  list[q].left=list[x].left;
  list[x].left=0;
  list[x].right=0;
}
```

```
//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helang的结点的编号
void searchList(string s)
{
  int p;
  p=head;
  while((p!=0) &&
(list[p].name!="helang"))p=list[p].right;
  if(list[p].name=="helang") 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<<" ";

}
```

课后练习：
队列：3629，1917，1095
栈：1914，1085
链表：1700
高一习题,1914，1085，1095，3767思维 3891