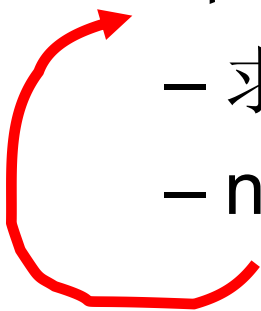


# 引例

- 输入一个数n
- 输出它的二进制表示
- 样例输入：
- 13
- 样例输出
- 1101

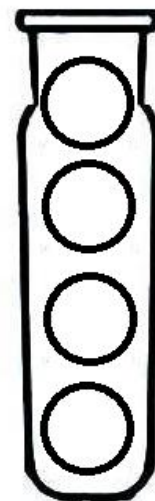
# 数学求解模拟

# 算法描述

- 输入 $n$
  - 当 $n \neq 0$ 
    - 求 $n$ 除以2的余数，并把这个余数放瓶子里
    - $n$ 变成 $n/2$
  - 将这个瓶子的数倒出来，一一输出
- 

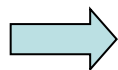
# 算法描述

- 输入 $n$
- 当 $n \neq 0$ 
  - 求 $n$ 除以2的余数，并把这个余数放瓶子里
  - $n$ 变成 $n/2$
- 将这个瓶子的数倒出来，一一输出



# 变成代码

- 输入 $n$
- 当 $n \neq 0$ 
  - 求 $n$ 除以2的余数，  
并把这个余数放瓶子里
  - $n$ 变成 $n/2$
- 将这个瓶子的数  
倒出来，一一输出



```
cin >> n;
while (n)
{
    s.push(n%2);
    n = n/2;
}

while (!s.empty())
{
    cout << s.top();
    s.pop();
}
```

# 参考代码

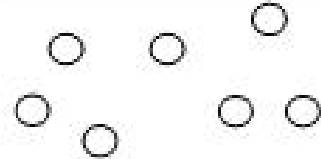

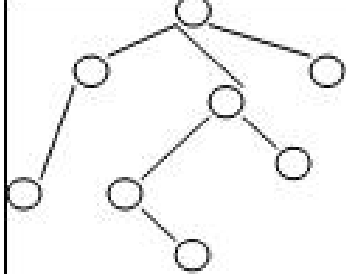
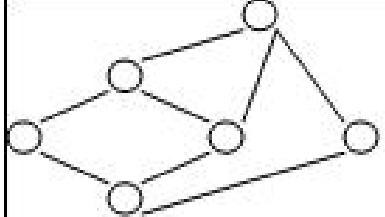
- `#include<iostream>`
- `#include<stack>`
- `using namespace std;`
- `stack<int>s;`
- `int n;`

```
int main()
{
    cin>>n;
    while (n)
    {
        s.push(n%2);
        n=n/2;
    }
    while(!s.empty())
    {
        cout<<s.top();
        s.pop();
    }
    return 0;
}
```

# 栈

一种常用的线性结构

# 四类基本数据(逻辑)结构

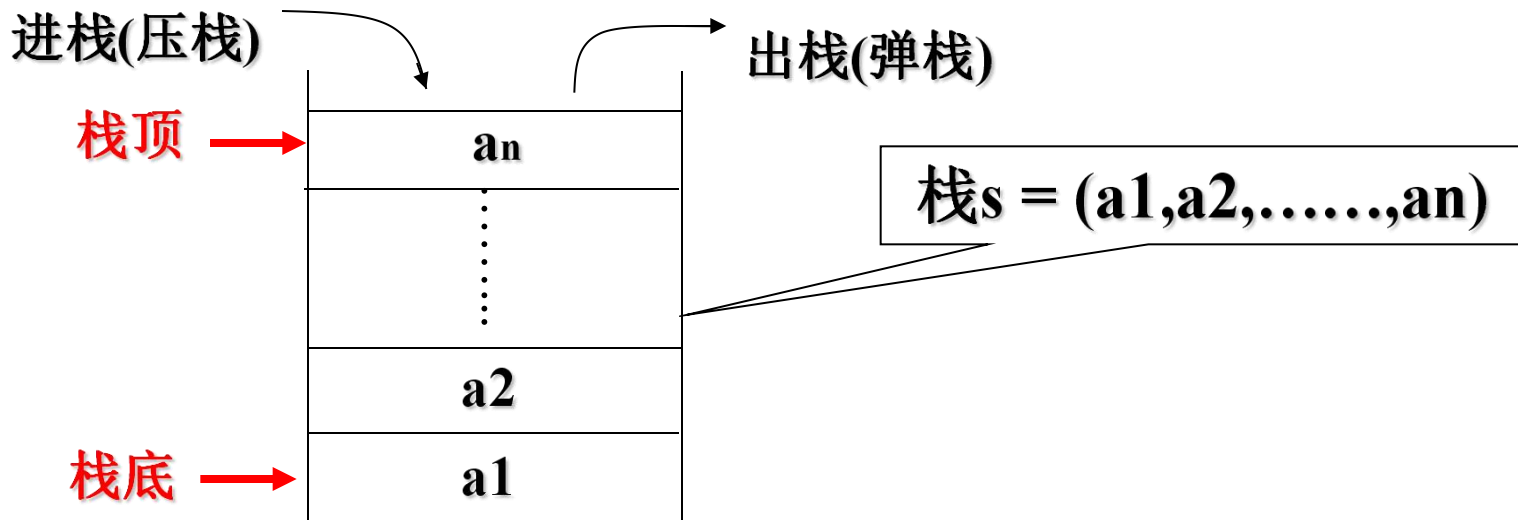
	数据间的关系特征	图示	备注
• 集合	数据元素“属于”或“不属于”集合。二者必居其一。		一般用其他数据结构代替
• 线性结构	数据元素存在一个对一个的关系		操作简单，编程复杂度低。
• 树形结构	数据元素存在一个对多个的关系		平均操作效率很高。广泛应用于查找。
• 图结构	数据元素存在多个对多个的关系		结构复杂、应用广泛，是常见的数学模型。有许多经典算法。



# 一、什么是栈

- 栈：限定仅在一端进行插入或删除操作的线性表

- $a_1$  为栈底元素
- $a_n$  为栈顶元素
- 不含元素的栈称空栈



## 二、栈的特点

- 根据栈的定义可知，最先放入栈中元素在栈底，最后放入的元素在栈顶，而删除元素刚好相反，最后放入的元素最先删除，最先放入的元素最后删除。
- 特点：后进先出
- 也就是说，栈是一种后进先出的线性表，简称为 **LIFO (Last In First Out)** 表。

# 三、栈的操作

- 头文件: `#include<stack>`
- 定义栈: `stack<int>s;`
- 压栈: `s.push();`
- 出栈: `s.pop();`
- 栈顶元素: `s.top();`
- 栈空: `s.empty()`

# 初赛题目集锦

1、有一个栈，进栈序列为A、B、C，试给出所有可能的出栈序列。

**A进 B进 C进 C出 B出 A出      CBA**

**A进 A出 B进 B出 C进 C出      ABC**

**A进 A出 B进 C进 C出 B出      ACB**

**A进 B进 B出 A出 C进 C出      BAC**

**A进 B进 B出 C进 C出 A出      BCA**

**不可能产生输出序列 CAB**

# 初赛题目集锦

- 2、某栈的入栈序列是a、b、c、d，则下列序列中不可能是它的输出序列的是（ ）
- A. a,c,b,d
- B. b,c,d,a
- C. c,d,b,a
- D. d,c,a,b

# 初赛题目集锦

- 3、一个栈的输入序列为1, 2, 3, ..., n, 若输出序列的第一个元素是n, 则输出的第i个元素是 ( )
- A. 不确定
- B.  $n-i+1$
- C. i
- D.  $n-i$

# 初赛题目集锦

- 4、今有一空栈S，对下列待进栈的数据元素序列a,b,c,d,e,f依次进行进栈，进栈，出栈，进栈，进栈，出栈的操作，则此操作完成后，栈S的栈顶元素为（ ）。
- A. f
- B. c
- C. a
- D. b

# 栈的应用——例1：括号匹配

- 输入一行括号，以#结束，判断是否匹配。
- 样例输入1：
- ((([]))<>)}#
- 样例输出1：
- YES
- 样例输入2：
- (([)]<>)#
- 样例输出2：
- NO



# 手工模拟过程

# 算法描述

- 假设是匹配的，`ok=TRUE`
- 当读入字符字符不是#
  - 如果是左括号，入栈
  - 如果是右括号，但是栈空，返回`ok=FALSE`
  - 如果是右括号，和栈顶括号对比
    - 如果匹配，栈顶括号出栈
    - 如果不匹配，返回`ok=FALSE`
- 如果处理完了，栈非空，返回`ok=FALSE`
- 根据`ok`的值输出YES或NO

# 变成代码？

- 假设是匹配的，`ok=TRUE`
- 当读入字符字符不是#
  - 如果是左括号，入栈
  - 如果是右括号，但是栈空，返回`ok=FALSE`
  - 如果是右括号，和栈顶括号对比
    - 如果匹配，栈顶括号出栈
    - 如果不匹配，返回`ok=FALSE`
- 如果处理完了，栈非空，返回`ok=FALSE`
- 根据`ok`的值输出YES或NO

# 参考代码

# 在线评测

- Codevs
- Luogu
- 题目搜索“括号”

# 栈的应用——例2：表达式求值

- 输入一个表达式，仅含有数字， $+$ ， $*$ ，以 $\#$ 结束。
- 输出表达式的值
- 样例输入
- $1+2*3*4+2\#$
- 样例输出
- $27$

# 手工模拟实现过程

- $1+2*3*4+2\#$

# 数据结构

- 用两个栈
- 一个存储操作数——数栈
- 一个存储操作符——符栈



# 算法描述

变成代码

# 数据结构演示系统

# 课后拓展

- 输入一个表达式，仅含有数字， $+$ ， $-$ ， $*$ ， $/$ ， $($ ， $)$ ，以 $\#$ 结束。
- 输出表达式的值，结果保留2位小数。
- 样例输入
- $1+3*(1+(9-5)/8)\#$
- 样例输出
- 5.50

# 在线评测

- NOIP2013PJT2 表达式求值
- <http://codevs.cn/problem/3292/>
- <https://www.luogu.org/problem/show?pid=1981>

# 总结

- 栈
  - 性质、操作
  - 初赛常考点
- 经典应用
  - 十进制转换成其他进制
  - 括号匹配
  - 表达式求值