



```

#include<iostream>
using namespace std;
void Hanoi(int n, char src, char mid, char dest);

int main(){
    int n;
    cin >> n;
    Hanoi(n, 'A', 'B', 'C');
    system("pause");
    return 0;
}

void Hanoi(int n, char src, char mid, char dest){
    if (n == 1){
        cout << src << "->" << dest<<endl;
        return;
    }
    Hanoi(n - 1, src, dest, mid);//先将n-1个 src移动到mid
    cout << src << "->" << dest << endl;
    Hanoi(n - 1, mid, src, dest);//再将n-1个从mid 移动带dest
    return;
}

```

## N皇后问题

说明：N皇后 N行N列 N个皇后不能在同一行 同一列 同意对角线

```

#include<iostream>
#include<vector>
#include<math.h>
using namespace std;
int N;

void N_help(vector<vector<int>> &res, vector<int>&t,int k){
    int i;
    if (k == N){
        res.push_back(t);
        return;
    }
    for (i = 0; i < N; ++i){//逐个尝试
        int j;
        for (j = 0; j < k; ++j){//和已经摆好的K-1个皇后比较
            if (t[j] == i || abs(t[j] - i) == abs(k - j))break;
        }
        if (j == k){
            t[k] = i;
            N_help(res,t, k + 1);
        }
    }
}

int main(){
    vector<vector<int>> res;

    cin >> N;
    vector<int>t(N, 0);
}

```

```

N_help(res, t, 0);
for (int i = 0; i < res.size(); ++i){
    for (int j = 0; j < N; ++j){

        cout << res[i][j]+1 << " ";

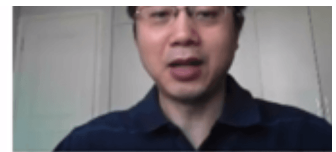
    }
    cout << endl;
}
system("pause");
return 0;
}

```

## 逆波兰表达式

### 定义

逆波兰表达式的定义：



- 1) 一个数是一个逆波兰表达式，值为该数
- 2) "运算符 逆波兰表达式 逆波兰表达式" 是逆波兰表达式，值为两个逆波兰表达式的值运算的结果

### code C++

```

#include<iostream>
#include<stdio>
#include<stdlib>
using namespace std;
double exp(){
    char s[20];
    cin >> s;
    switch (s[0]){
        case '+':return exp() + exp();
        case '-':return exp() - exp();
        case '*':return exp() * exp();
        case '/':return exp() / exp();
        default:return atof(s);
        break;
    }
}
int main(){
    printf("%lf", exp());
    system("pause");
    return 0;
}

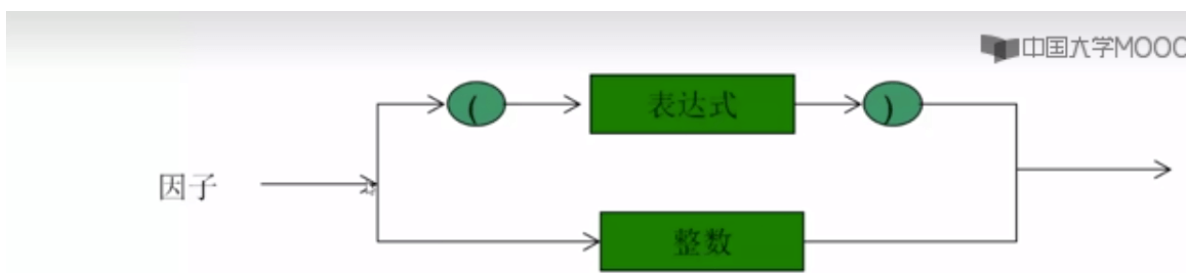
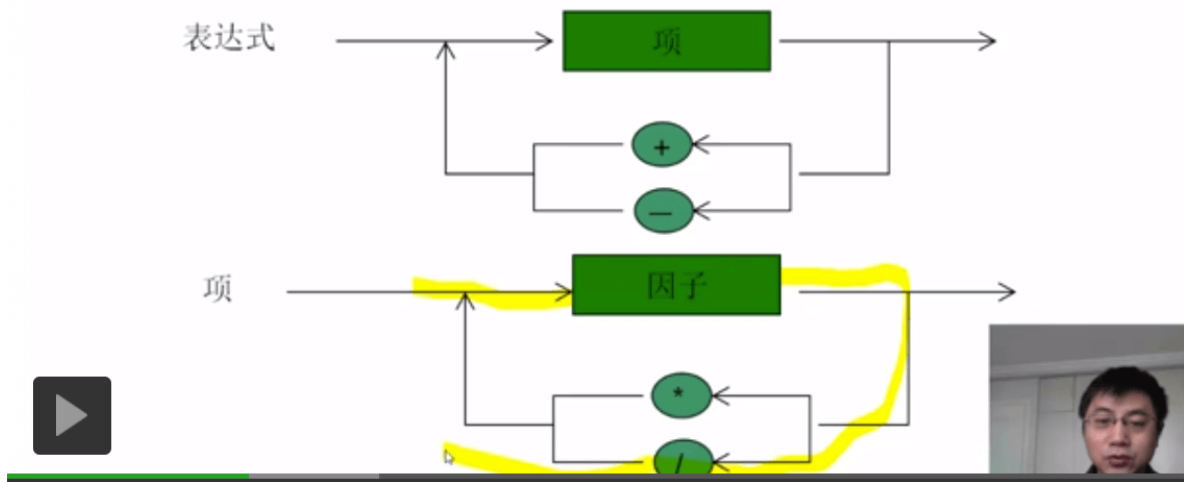
```

## 表达式求值

加减乘除括号

### 逻辑图

表达式是个递归的定义：



因此对于表达式可以进行递归分析处理

## c++code

```
#include<iostream>
#include<cstring>
#include<cstdlib>
using namespace std;
int factor_value();//因子
int term_value();//项
int expression_value();//表达式

int main(){
    cout << expression_value() << endl;
    system("pause");
    return 0;
}

int factor_value(){
    char op = cin.peek();
    int result = 0;
    if (op == '('){
        cin.get();
        result = expression_value();
    }
}
```

```

        cin.get();
    }
    else while (isdigit(op)){
        result = 10 * result + op - '0';
        cin.get();
        op = cin.peek();

    }
    return result;

}

int term_value(){
    int result = factor_value();
    while (1){
        char op = cin.peek();
        if (op == '*' || op == '/'){
            cin.get();
            int value = factor_value();
            if (op == '*')result *= value;
            else result /= value;
        }
        break;
    }
    return result;
}

int expression_value(){
    int result = term_value();
    while (1){
        char op = cin.peek();
        if (op == '+' || op == '-'){
            cin.get();
            int value = term_value();
            if (op == '+')result += value;
            else result -= value;
        }
        break;
    }
    return result;
}

```

## 爬楼梯问题

### 题目

# 用递归将问题分解为规模更小的子问题进行求解

中国大学MOOC

## 例题: 爬楼梯

树老师爬楼梯，他可以每次走1级或者2级，输入楼梯的级数，求不同的走法数

例如：楼梯一共有3级，他可以每次都走一级，或者第一次走一级，第二次走两级，也可以第一次走两级，第二次走一级，一共3种方法。

## 输入

输入包含若干行，每行包含一个正整数N，代表楼梯级数， $1 \leq N \leq 30$  输出不同的走法数 每一行输入对应一行

## 题解

### 爬楼梯

n级台阶的走法 =

先走一级后，n-1级台阶的走法 +  
先走两级后，n-2级台阶的走法

$$f(n) = f(n-1) + f(n-2)$$

边界条件：

$n < 0$	0	$n = 0$	1	$n = 1$	1
$n = 0$	1	$n = 1$	1	$n = 2$	2

边界条件的意思就是：递归的终止条件 递归一定要有终止条件 不能一直递归下去

## c++ code

```
#include<iostream>
using namespace std;
int louti(int n);
int main(){
    int N;
    while (cin >> N){
        cout << louti(N) << endl;
    }
    system("pause");
    return 0;
}
int louti(int n){
    if (n == 1)return 1;
    if (n == 2)return 2;
    return louti(n - 1) + louti(n - 2);
}
```



# 放苹果

## 题目

### 例题：放苹果

中国大学MOOC

把M个同样的苹果放在N个同样的盘子里，允许有的盘子空着不放，问共有多少种不同的分法？5，1，1和1，5，1 是同一种分法。

#### 输入

第一行是测试数据的数目t (0 <= t <= 20)。以下每行均包含二个整数M和N，以空格分开。1<=M, N<=10。

#### 输出

对输入的每组数据M和N，用一行输出相应的K。

#### 样例输入

1

7 3

#### 样例输出

3

## 解释

### 例题：放苹果

设i个苹果放在k个盘子里放法总数是  $f(i, k)$ ，则：

$k > i$  时，  $f(i, k) = f(i, i)$

$k \leq i$  时，总放法 = 有盘子为空的放法 + 没盘子为空的放法

$f(i, k) = f(i, k-1) + f(i-k, k)$

边界条件？



## c++code

```
#include<iostream>
using namespace std;
int f(int m, int n){
    if (n > m)return f(m, m);
    if (m == 0)return 1;
    if (n == 0)return 0;
    return f(m, n - 1) + f(m - n, n);
    //有盘子为空和没有盘子为空
    //没有盘子为空代表，先每个盘子放一个。剩下m-n在放
}
void main(){
    int t, m, n; //m是苹果数目，n是盘子数目
    cin >> t;
    while (t--){
        cin >> m >> n;
        cout << f(m, n) << endl;
    }
}
```

```
}  
system("pause");  
}
```

## 算24

### 题目

#### 例题：算24

中国大学MOOC

给出4个小于10个正整数，你可以使用加减乘除4种运算以及括号把这4个数连接起来得到一个表达式。现在的问题是，是否存在一种方式使得得到的表达式的结果等于24。

这里加减乘除以及括号的运算结果和运算的优先级跟我们平常的定义一致（这里的除法定义是实数除法）。

比如，对于5, 5, 5, 1，我们知道 $5 * (5 - 1 / 5) = 24$ ，因此可以得到24。又比如，对于1, 1, 4, 2，我们怎么都不能得到24。

### 解释

#### 例题：算24

中国大学MOOC

n个数算24，必有两个数要先算。这两个数算的结果，和剩余n-2个数，就构成了n-1个数求24的问题

枚举先算的两个数，以及这两个数的运算方式。

边界条件：一个数算24

注意：浮点数比较是否相等，不能用 `==`

### c++ code

```
#include<iostream>  
#include<cmath>  
using namespace std;  
#define EPS 1e-7  
double a[5];  
bool isZero(double x){  
    return fabs(x) < EPS;  
}  
bool count24(double a[], int n){  
    if (n == 1) return isZero(a[0] - 24);  
    double b[5];  
    for (int i = 0; i < n - 1; ++i){  
        for (int j = i + 1; j < n; ++j){  
            int m = 0; //还剩下m个数 m=n-2
```



```

        for (int t = 0; t < n; ++t){
            if (t != i && t != j)
                b[m++] = a[t]; //把其余的数放b里面
        }
        b[m] = a[i] + a[j];
        if (count24(b, m + 1)) return true;
        b[m] = a[i] - a[j];
        if (count24(b, m + 1)) return true;
        b[m] = a[j] - a[i];
        if (count24(b, m + 1)) return true;
        b[m] = a[i] * a[j];
        if (count24(b, m + 1)) return true;
        if (!isZero(a[i])){
            b[m] = a[j] / a[i];
            if (count24(b, m + 1)) return true;
        }
        if (!isZero(a[j])){
            b[m] = a[i] / a[j];
            if (count24(b, m + 1)) return true;
        }
    }
}

return false;
}

void main(){
    int t = 4;
    for (int i = 0; i < t; ++i){
        cin >> a[i];
    }
    cout << count24(a, 4);
    system("pause");
}

```