```cpp
#include<iostream>
#include<algorithm>
#include<string>
using namespace std;
```

简单模拟

简单模拟照着做就是了，但时刻需要注意细节。比如数学题要注意循环边界，以及运算过程中有没有可能出现除零的情况。方阵图相关的模拟要注意不要越界。

P2670 扫雷

扫雷游戏是一款十分经典的单机小游戏。在n*n行m*m列的雷区中有一些格子含有地雷（称之为地雷格），其他格子不含地雷（称之为非地雷格）。玩家翻开一个非地雷格时，该格将会出现一个数字——提示周围格子中有多少个是地雷格。游戏的目标是在不翻出任何地雷格的条件下，找出所有的非地雷格。

现在给出nn行mm列的雷区中的地雷分布，要求计算出每个非地雷格周围的地雷格数。

需要用到的变量

```cpp
namespace a{
    int row, col;
    char graph[101][101];
    int dire[][2] = {{-1, 0},{-1, 1},{-1, -1},{0, 1},{0, -1},{1, 0},{1, 1},{1, -1}};
    // 相对于当前位置的偏移量
    //一般处理有关方阵图的时候，都会用得到这个数组（个人习惯）
}
```

输入部分

```
cin>> a::row >> a::col;

for(int i = 0; i < a::row; i++){
    for(int j = 0; j < a::col; j++)
        cin>>a::graph[i][j];
}
```

```
 3 3
 *??
 ???
 ?*?
```

一些很方便的函数

```
inline bool is_valid(int x, int y){
    return x >= 0 && x < a::row && y >= 0 && y < a::col;
}
//判断位置是否合法
//直接写在if里也可以，但写成inline函数看着舒服一点
```

```
int count(int x, int y){
    int res = 0;
    for(int i = 0; i < 8; i++){
        int a = x + a::dire[i][0], b = y + a::dire[i][1];
        //令当前位置加上偏移量
        if(is_valid(a, b)){
            if(a::graph[a][b] == '*')res++;
        }
    }
    return res;
}
```

最终过程

```
for(int i = 0; i < a::row; i++){
    for(int j = 0; j < a::col; j++){
        if(a::graph[i][j] == '?')a::graph[i][j] =
count(i, j) + '0';
        cout<<a::graph[i][j];
    }
    cout<<endl;
}
```

```
*10
221
1*1
```

查找&排序

P1093奖学金

每个学生都有3门课的成绩:语文、数学、英语。先按总分从高到低排序，如果两个同学总分相同，再按语文成绩从高到低排序，如果两个同学总分和语文成绩都相同，那么规定学号小的同学 排在前面，这样，每个学生的排序是唯一确定的。

```
namespace s{
    int n;
    struct stu{
        int ser;
        int Chinese;
        int Math;
        int English;
        int sum;

        //定义‘小于’的比较方法，用于排序
        bool operator<(stu& oth){
            if(sum == oth.sum){
                if(Chinese == oth.Chinese){
                    return ser > oth.ser;
```

```
                } else return Chinese < oth.Chinese;
            } else return sum < oth.sum;
        }
    }all[300+10];
}
```

```
cin>>s::n;
for(int i = 0,a,b,c; i < s::n; i++){
    cin>> a >> b >> c;
    s::all[i].ser = i + 1;
    s::all[i].Chinese = a;
    s::all[i].Math = b;
    s::all[i].English = c;
    s::all[i].sum = a + b + c;
}
```

```
 6
 90 67 80
 87 66 91
 78 89 91
 88 99 77
 67 89 64
 78 89 98
```

```
sort(s::all, s::all + s::n);
//STL排序，也可以自己手写排序
for(int i = 0; i < 5; i++){
    cout<< s::all[s::n - 1 - i].ser << ' ' << s::all[s::n - 1 - i].sum <<endl;
}
```

```
6 265
4 264
3 258
2 244
1 237
```

日期相关的问题

例： 给定两个年月日，输出其之间相差的天数。

比如，给定两个日期：2020-04-21，2020-05-01

输出其所差的天数：10。

```
namespace b{
    int month[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    //每月的天数，其中二月为平年的天数

    struct date {
    int year;
    int mon;
    int day;
    bool is_leap() {
        return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
    }

    date(int y, int m, int d) : year(y), mon(m), day(d) {}

    void operator+=(int d) {
        int a, b;
        int md = (mon == 2) ? (month[mon] + is_leap()) : month[mon];
```

```
        //如果是2月，则判断是否是闰年，如果是闰年就加上一天
        a = (day - 1 + d) / md;
        day = (day - 1 + d) % md + 1;
        b = (mon - 1 + a) / 12;
        mon = (mon - 1 + a) % 12 + 1;
        year = year + b;
    }

    bool operator!=(date& oth) {
        return !((year == oth.year && mon == oth.mon && day == oth.day));
    }
};
}
```

```
b::date d1(2020, 4, 21), d2(2020, 5, 1);
int res = 0;
while (d1 != d2) {
    res++;
    d1 += 1;
}
cout << res << endl;
```

```
11
```

图形输出

1. 根据图形的规律来进行输出

例：画个漏斗吧！n代表漏斗最上层*的个数。

```
例如，当n = 11时：
***********
 *********
  *******
   *****
    ***
     *
    ***
   *****
  *******
 *********
***********
```

通过观察可以得到如下特征：

1. 图形上下对称，因此想办法得到上半部分就行了
2. 每层开头的空格数依次递增一个，总字符数一次递减一个

因此根据这两个特征，我们只需要计算出每层的空格数和总字符数，然后填充到数组里面就可以了。

```
namespace c {
    char row[51][101];
    int n;
}
```

```
cin >> c::n;
```

```
 15
```

```
for(int i = 0; i < c::n/2; i++){
    for(int j = 0; j < c::n - i; j++){
        if(j < i)c::row[i][j] = ' ';
        else c::row[i][j] = '*';
    }
}
```

```
for(int i = 0; i < c::n/2; i++)cout<<c::row[i]
<<endl;
cout<<setw(c::n / 2 + 1)<< '*' <<endl;
for(int i = c::n/2 - 1; i >= 0; i--)cout<<c::row[i]
<<endl;
//反着输出一遍
```

```
***************
 *************
  ***********
   *********
    *******
     *****
      ***
       *
      ***
     *****
    *******
   *********
  ***********
 *************
***************
```

2. 定义字符数组，通过图形规律来填充数组，最终输出

P5461赦免战俘

现在有一个$2^n \times 2^n$的全部为1的方阵，我们需要将这个方阵分为四个小方阵，然后将左上角的那个方阵全部变为0，其余的三个再分为四个方阵，将左上角变为0.....

请输出经过这样的变换之后，最终的图形。

解： 需要用到递归的思想。对于一个方阵，我们先将其左上角全部变为0，然后再依次用同样的方法处理其余的三个部分。

```
namespace d{
    char graph[1145][1419];
    //初始的数组要开的足够大，要不然有可能会越界
    int n;
}
```

```
void set_zero(int x, int y, int length) {
    if (!(x || y))
        for (int i = 0; i < length; i++)
fill(d::graph[i], d::graph[i] + length, '1');
    //初始化

    if (length == 1) return;
    //递归边界

    for (int i = x; i < length / 2 + x ; i++)
        for (int j = y; j < length / 2 + y ; j++)
d::graph[i][j] = '0';
    //左上角置零

    int l = length / 2;
    set_zero(x, y + l, l);
    set_zero(x + l, y, l);
    set_zero(x + l, y + l, l);
    //递归处理其余部分
}
```

```
cin >> d::n;

int l = pow(2, d::n);
set_zero(0, 0, l);
for (int i = 0; i < l; i++) {
    for (int j = 0; j < l; j++) cout << d::graph[i]
[j] << ' ';
    cout << endl;
}
```

```
3
```

```
0 0 0 0 0 0 0 1
0 0 0 0 0 0 1 1
0 0 0 0 0 1 0 1
0 0 0 0 1 1 1 1
0 0 0 1 0 0 0 1
0 0 1 1 0 0 1 1
0 1 0 1 0 1 0 1
1 1 1 1 1 1 1 1
```

当然也可以试试递推。我们很容易就能写出当n = 1时的情形，通过这一点来推出下一个图形的样子：下一个图形的左上角全部为0，而其余部分则复制自其之前一个图形的样子。

进制转换

n进制转十进制

```cpp
#define ll long long
ll to_dec(ll num,int pos = 2){
    ll res = 0;
    ll p = 1;
    while(num){
        res += (num % 10) * p;
        num /= 10;
        p *= pos;
    }
    return res;
}
```

```cpp
ll to_dec_s(string num, int pos = 2){
    ll res = 0;
    ll p = 1;
    for(int i = num.length() - 1; i >= 0; i--){
        res += (num[i] - '0') * p;
        p *= pos;
    }
    return res;
}
```

```cpp
cout<< to_dec(114514, 6) <<endl;
cout<< to_dec_s("114514", 6) <<endl;
```

```
10126
10126
```

十进制转n进制

```cpp
ll to_oth(ll num, int pos){
    ll res = 0;
    ll p = 1;
    while(num){
        res += (num % pos) * p;
        num /= pos;
        p *= 10;
    }
    return res;
}
```

字符串形式的转换函数留做习题。

```cpp
cout << to_oth(114, 2) <<endl;
```

```
1110010
```

为啥一定要用十进制

十进制也不是啥天选之子，只是因为如果将一种进制转为十进制再转为其他进制的话，转换过程中的加法就可以让编译器帮咱们实现。如果你契而不舍地手写了其他进制的加法，那自然就可以直接进行进制转换，原理是一样的。

字符串处理

常见的问题包括查找字串位置，字串替换，统计字串数量，字符串格式化等等。
根本在于理解清楚题目要求，且时刻注意字符串处理的常见问题（如缓冲区残留的换行符等等）。

P1308统计单词数

给定一个单词和一行字符串，统计该单词在字符串中第一次出现的位置和单词总共出现的次数，如果没有找到则返回-1。不区分大小写。

需要干的事情:

1. 由于不区分大小写，得有个办法把一个单词全都变为小写
2. 辨别两个单词是否完全相同

```
namespace r {
char voc[100];
char str[1000010];
int res = 0,pos = -1;
}
```

```
void tolower(char* str, unsigned size) {
    for (int i = 0; i < size; i++) {
        if (str[i] >= 'A' && str[i] <= 'Z') str[i] += 32;
    }
}
```

```
bool is_match(char* v, char* s, unsigned size) {
    tolower(s, size);
    for (int i = 0; i < size; i++)
        if (v[i] != s[i]) return 0;
    return 1;
}
```

输入部分

```
cin.getline(r::voc, 100, '\n');
cin.getline(r::str, 1000010, '\n');
unsigned lv = strlen(r::voc), ls = strlen(r::str);
tolower(r::voc, lv);
```

```
 To
 to be or not to be is a question
```

```
unsigned t = 0;
for (int i = 0; i <= ls; i++) {
        if (i == ls || r::str[i] == ' ') {
            //短路特性
            if (t == lv && is_match(r::voc,
&r::str[i - t], lv)) {
                r::res++;
                if (r::pos < 0) r::pos = i - t;
            }
            t = 0;
            while (r::str[i + 1] == ' ') i++;//跳过
空格
        } else
            t++;
    }
    if (!r::res)
        cout << -1 << endl;
    else
        cout << r::res << ' ' << r::pos << endl;
```

```
2 0
```

更普遍的操作可以使用有限自动机，这部分不在此进行展开。