C语言中常见的头文件 <http://blog.csdn.net/weiyong1999/article/details/8062246>

几种数据类型以及位数

8位：Byte（字节型） 16位：short（短整型）、char（字符型） 32位：int（整型）、float（单精度型/浮点型） 64位：long（长整型）、double（双精度型） 最后一个：boolean(布尔类型

错误处理 errno.h 2 字符处理 ctype.h 3 地区化 local.h 4 数学函数 math.h 5 信号处理 signal.h 6 输入输出 stdio.h 7 实用工具程序 stdlib.h 8 字符串处理 string.h

浮点数 8.0 和5.0别看做实数 专业一点就是浮点数 注意运算符 / 是个多面手 既可以做整数除法也可以针对 浮点

变量的定义范围要小 可以避免一些其他问题 （经验）

最后的输出时候 如果是数字的话 要注意第一位是否可能为0 ，输出的结构可以优化为 %0aa 等

在程序完成后要进行相应的检验 一个是数据是否会溢出使用中间变量的方法 检验结果

当循环的结构体不是确定的时候 进行的是while循环

Do {循环体} while {条件} 循环体至少执行一次 执行完循环体 进行相应的条件判断

10-6次方的表示 1e-6

间隔相加 奇数相加 偶数想减 for(int i=0;;i++) if(i%2==0) else

Factorial 累乘器 也就是阶乘计算 一定要初始化为1

对结果进行输出的时候 要是取后几位的话一般是 % 进行取余输出

程序要考虑运行的快慢 先累加计算 在进行结果改变 也可以进行每部改变 部分相加

算法 的输入输出框架

Freopen (“input.txt”,”r”,stdin);

Freopen (“output.txt”,”w”,stdout) 但是需要在头文件加上 # define LOCAL 才可以进行重定向 # ifdenfLOCAL #endif

数组的理解 a[n++]= x 首先复制a[n]=x 然后执行 n= n+1

数组是不能进行复制操作的 比如 in a[maxn] ,b[maxn] 不能直接使用 b=a; 如果想将 a的数组k个元素复制到b 中 可以采用 memcopy( b, a, sizeof(int) \* k); 使用此函数需要头文件string.h, 要是去全部复制的话，直接使用memcopy (b, a,sizeof)

Memset( a, 0 , sizeof(a)) 是用来数组清零

字符数组的使用 字符串就相当于字符数组

Scanf (“%s”,s) 会读取一个不含空格 tab个回车符的字符串 存在字符串数组s 中。

Scanf(“%s”s[i])表示方、读取第i个字符但是 scanf(“%s”,s)遇到空白会停下来

当读取含有tab或者空格的字符串时，可以采用以下的方法

1 fgetc(fin) 读取一个打开的文件 fin 读取一个字符 返回一个int 值 文件读取结束后 fgetc 会返回一个 EOF

2 fgets ( buf , maxn ,fin) 读取完整的一行 其中buf的声明为char buf[ maxn] ， 这个函数的读取不超过 maxn -1 字符 在末尾添加**结束符”\0”**  ，因此不会出现越界的情况 一旦读到回车符 \n 读取就会停止 所以说只能读取一行 边读边存在buf中 maxn是一个数组 fin是需要的读取的文件

Getchar 是一个读取用户键盘输入的字符 如果想要进行输入和输出具有一定的对应性和关联性， 可以用选择语句 if 或者switch 来进行但是太麻烦了 考虑用 常量数组

Putchar() 是一个想终端输出一个字符

回文词 就是反转以后和原串相同

镜像串 是指左右镜像以后和用之前的一样

C++随机数的产生 <http://blog.csdn.net/candyliuxj/article/details/4396666>

rand() 随机数发生器 int rand(void) 所在头文件 stdlib.h

rand()的内部实现是用线性同余法做的，它不是真的随机数，因其周期特别长，故在一的范围里可看成是随机的。

rand()返回一随机数值的范围在0至RAND\_MAX 间。RAND\_MAX的范围最少是在32767之间(int)。用

unsigned int 双字节是65535，四字节是4294967295的整数范围。0~RAND\_MAX每个数字被选中的机率是相同的。

用户未设定随机数种子时，系统默认的随机数种子为1。

rand()产生的是伪随机数字，每次执行时是相同的

Srand 初始化随机数发生器 void srand (unsigned int seed) 用来设定产生随机数时的随机数种子，参数seed必须是个整数 ,通常可以用time(0)的返回值或NULL来做seed

如果每次的seed设置相同 rand ()产生的随机数就会相同

产生一定范围随机数的通用表示公式

要取得[a,b)的随机整数，使用(rand() % (b-a))+ a （结果值含a不含b）。

要取得[a,b]的随机整数，使用(rand() % (b-a+1))+ a （结果值含a和b）。

要取得(a,b]的随机整数，使用(rand() % (b-a))+ a + 1 （结果值不含a含b）。

（总的来说，通用公式：a + rand() % n ；其中的a是起始值，n是整数的范围）

要取得a到b之间的随机整数，另一种表示：a + (int)b \* rand() / (RAND\_MAX + 1)。

要取得0～1之间的浮点数，可以使用rand() / double(RAND\_MAX)。

在自定义的函数或者是结构体中 {函数体或者是结构体}

执行函数时如果没有return会返回一个不确定的值，碰到return会直接退出这个函数

Struct Point { double x, y ;};

Double dist (struct point a , sturct point b)

{ return hypot ( a.x –b.x , a.y – b.y);

}

对于栈的认识 <http://blog.sina.com.cn/s/blog_6975d67c01013jm7.html>

在往箱子里面放衣物的时候，放在最上面的衣物总是我们最后放上去的；而当我们从箱子里取出衣物的时候，总是最先拿出上面的。这就是现实生活中的栈。

准确的讲，栈就是一种可以实现“先进后出(或者叫后进先出)”的存储结构。

学过数据结构的人都知道：栈可以用两种方式来实现，一种方法是用数组实现栈，这种栈成为静态栈；另外一种方法是用链表实现栈，这种栈叫做动态栈

C 语言中 调用栈 就是来处理调用 函数之间的关系

形参和实参进行传值 或者是交换的时候 要进行地址的传递

Eg：

# include <stdio.h>

Void swap ( int \*a , int \* b);

{

Int t = \* a; \* a = \* b; \*b =\*c ;

Int main(){

Int a =3 ,b =4;

Swap ( & a , & b);

Printf(“ %d %d \n” ,a,b);

Return 0;

}

}

}

数组作为参数进行传值 sizeof( a)

Int a[] 等价于int \*a 因此 a只是相当于一个指针

**数学概念与方法**

欧几里德算法又称辗转相除法，是指用于计算两个正整数a，b的最大公约数

gcd(a,b) = gcd(b,a mod b) (不妨设a>b 且r=a mod b ,r不为0)

**RSA算法**基于一个十分简单的数论事实：将两个大质数相乘十分容易，但是想要对其乘积进行因式分解却极其困难，因此可以将乘积公开作为加密密钥 （用来进行加密）

最简单的gcd算法：

1http://www.cppblog.com/Images/OutliningIndicators/None.gifint gcd(int x, int y)  
2http://www.cppblog.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
3http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif     if(y == 0) return x;      
4http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif     if(x < y)      return gcd(y,x);      
5http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif     else        return gcd(y, x%y);   
6http://www.cppblog.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}

ACM中常用的gcd算法：

1http://www.cppblog.com/Images/OutliningIndicators/ExpandedBlockStart.gifint gcd(int a, int b){ return a == 0 ? b : gcd(b % a, a); }

经过优化的gcd算法(分成奇偶两种情况)：

 1http://www.cppblog.com/Images/OutliningIndicators/None.gifint gcd(int x,int y )  
 2http://www.cppblog.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
 3http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif    if(x < y) return gcd(y,x);  // x>y  
 4http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif    if( y == 0) return x;  // if y=0, x is GCD   
 5http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif    else  
 6http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif    {  
 7http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif         if( !(x%2) )  
 8http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif         {                   
 9http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif           if( !(y%2) )  //x,y both even  
10http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif               return 2\*gcd(x >> 1, y >> 1);      
11http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif           else      // x is even, y is odd  
12http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif               return gcd(x >> 1, y );    
13http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif

       }  
14http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif         else   
15http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif         {  
16http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif           if( !(y%2) )  // x is  odd,  y is even  
17http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif               return gcd(x, y >> 1);  
18http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif           else       // x, y both odd  
19http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif               return gcd(y,x-y);   
20http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif         }  
21http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif    }  
22http://www.cppblog.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
23http://www.cppblog.com/Images/OutliningIndicators/None.gif

筛选法求素数 <http://blog.csdn.net/qq_32680617/article/details/50629574>

Eraosthenes 筛法 双层循环进行删选，2p ，3p，4p等逐个排除）

筛选法 vis[i]表示已经被删除

memset(vis ,0 , sizeof(vis));

for (int i=2; i<=n ; i++)

for (int j=i\*2; j<=n; j+=i) vis[j]=1;

改进

int m= sqrt(n+0.5);

memset (vis ,0,sizeof(vis));

for (int i=2;i<=m ;i++) if(!vis[i])

for (int j=i\*i;j<=n;j+=i) vis[j]=1;

**同余和模算术**

mod即使取余计算

通常取模运算也叫取余运算，它们返回结果都是余数.rem和mod唯一的区别在于:

当x和y的正负号一样的时候，两个函数结果是等同的；当x和y的符号不同时，rem函数结果的符号和x的一样，而mod和y一样。

注意：

给定一个正整数p，任意一个整数n，一定存在等式

n = kp + r 其中k、r是整数，且 0 ≤ r < p，称呼k为n除以p的商，r为n除以p的余数。

对于正整数p和整数a,b，定义如下运算：

取模运算：a mod p 表示a除以p的余数。

模p加法：(a + b) mod p ，其结果是a+b算术和除以p的余数，也就是说，(a+b) = kp +r，则 (a+b) mod p = r

Int mul\_mod ( int a,int b, int n){

a % = n; b %= n;

retrun (int )(longlong )a \* b % n);

}

最大数 1234=((1\*10+2)\*10+3)\*10+4 每步取余

Scanf (“%s%d”,n, &m);

Int len = strlen(n);

Int ans = 0;

For (int i=0; I < len; i++)

Ans = ( int )(( long long) ans\* 10 + n[i] –‘0’ ) % m);

Printf( “%d\n”, ans);

幂取模 输入正整数 a ,n 和m

当n较小时

Int pow\_mod ( int a, int n, int m){

Int ans =1;

For ( int i=0 ;i<n ; i++)

Ans = ( int )( long long) ans \* n % m);

}

分治法 处理n较大时

Int pow\_mod (int a,int n, int m) {

If ( n==0) return 1;

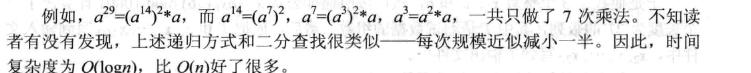
Int x = pow\_mod(a, n/2,m);

Long long ans = (long long ) x \* x %m;

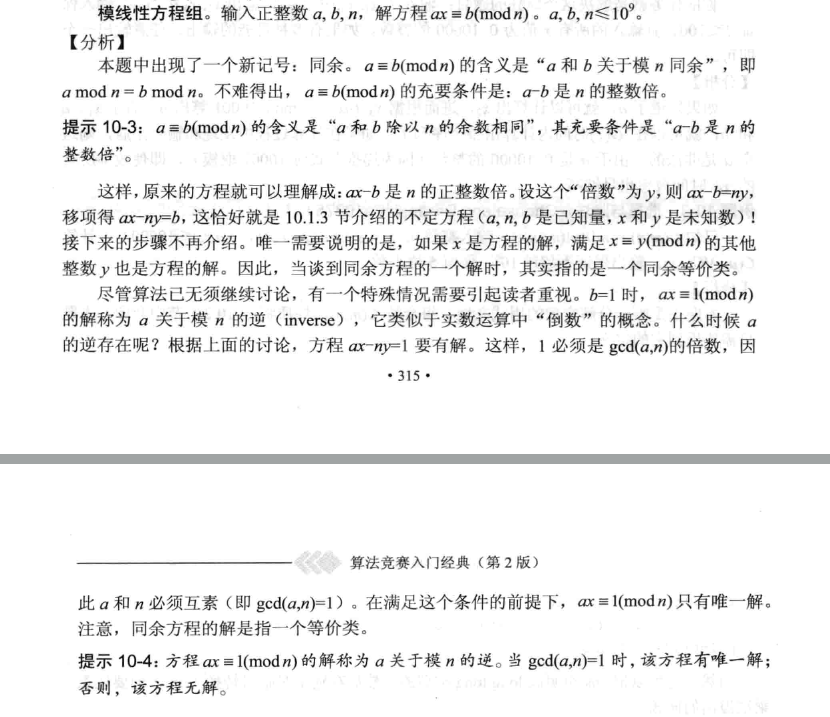
If( n%2 ==1) ans = ans \*a % m;

Return (int) ans ;

}



横线性方程组



暴力求解

乘积 的表示 ans long long

枚举的条件 可以根据题意进行缩小

用栈来进行运算中数据的存

vector是一个动态的序列容器，相当于一个size可变的数组。vector能更快的索引元素(就像数组一样)，而且能相对高效的在尾部插入和删除元素。如果不是在尾部插入和删除元素，效率就没有这些容器高。

当需要使用vector的时候，需要包含头文件：#include <vector>

常见用法(以int类型为例)

1.声明及初始化

vector<int> a; //声明一个int型向量a

vector<int> a(10); //声明一个初始大小为10的向量

vector<int> a(10, 1); //声明一个初始大小为10且初始值都为1的向量

vector<int> b(a); //声明并用向量a初始化向量b

vector<int> b(a.begin(), a.begin()+3); //将a向量中从第0个到第2个(共3个)作为向量b的初始值

int n[] = {1, 2, 3, 4, 5};

vector<int> a(n, n+5); //将数组n的前5个元素作为向量a的初值

vector<int> a(&n[1], &n[4]); //将n[1] - n[4]范围内的元素作为向量a的初值

2.添加元素

vector<int> a;

a.push\_back(1); //在尾部加入一个数据

a.push\_back(2);

a.pop\_back(); //删除最后一个数据

a.insert(a.begin(), 0); //在a.begin()之前加入0

a.erase(a.begin()); //将a.begin()的元素删除

a.erase(a.begin() + 1, a.end()); //将第二个元素以后的元素均删除

判断vector是否为空

vector<int> a;

if(a.empty()){

a.push\_back(1);

}

4.遍历访问vector

vector<int> a;

//像数组一样以下标访问

for(int i = 0; i < a.size(); i++){

cout << a[i];

}

//以迭代器访问

vector<int>::iterator it;

for(it=a.begin(); it!=a.end(); it++){

cout<<\*it<<" " ;

}

排序必须包含algorithm头文件

#include <algorithm>

vector<int> a;

sort(a.begin(), a.end());

6.以vector实现二维数组

vector<vector<int>> a(10, vector<int>(5));

枚举排列

生成1~n的 排列

递归函数时需要以下参数

1已经确定好的前缀序列 2 需要进行排列的元素集合，以便一次选出第一个元素

类似的伪代码

Void print\_permutation ( 序列a， 集合s)

{

If

（ s为空） 输出序列a；

Else 按照从小到大的顺序一次考虑s 的每一个元素

{

Print\_permutation ( 在a 的末尾加v 后得到 新序列， s-{v})

}

}

数组来存序列 a ， c语言中函数接受数组参数时无法得知数组的个数 所以xuyao确定一下位置

Void print\_permutation ( int n,int \*A , int cur){

If (cur ==n) { //递归边界

For (int i=0; i< n; i++) printf(“ %d”,A[i]);

Printf(“\n”);

}

Else for (int i=1; i<=n ; i++){

Int ok =1;

For ( int j=0 ; j< cur ; j++)

If( A[j] == i) ok =0;

If(ok) {

A[cur] =I;

Print\_permutation (n , A, cur+1);

}

}

}

二叉树的性质

性质1：二叉树第i层上的结点数目最多为 2{i-1} (i≥1)。

性质2：深度为k的二叉树至多有2{k}-1个结点(k≥1)。

性质3：包含n个结点的二叉树的高度至少为log2 (n+1)。

性质4：在任意一棵二叉树中，若终端结点的个数为n0，度为2的结点数为n2，则n0=n2+1

While( scanf(“%d%d”, &D , & I)==2) {

Int k =1;

For( int i=0; i<D-1 ; i++)

If ( I %2) { k = K\*2; I = ( I +1) /2 ;}

Else { k = k\* 2+1; I / = 2;}

Printf(“ %d\n” , k);

}