**HCMIU IOI Manual**

Mathematics

Contents

[1. Basic 2](#_Toc449477574)

[1.1 Library 2](#_Toc449477575)

[ Commonly Used Libraries: 2](#_Toc449477576)

[1.2 Input / Output 2](#_Toc449477577)

[ Input 2](#_Toc449477578)

[ Input a line 2](#_Toc449477579)

[ Output with format 2](#_Toc449477580)

[2. Mathematics 3](#_Toc449477581)

[2.1 Division 3](#_Toc449477582)

[ Divisibility by 2: 3](#_Toc449477583)

[ Divisibility by 3: 3](#_Toc449477584)

[ Divisibility by 4: 3](#_Toc449477585)

[ Divisibility by 5: 4](#_Toc449477586)

[ Divisibility by 6: 4](#_Toc449477587)

[ Divisibility by 8: 4](#_Toc449477588)

[ Divisibility by 11: 4](#_Toc449477589)

[ Greatest Common Divisor: 5](#_Toc449477590)

[ Least Common Multiplier: 5](#_Toc449477591)

[ Greatest Common Divisor of 3 numbers: 5](#_Toc449477592)

[ Extended GCD: 5](#_Toc449477593)

[2.2 Modulo 6](#_Toc449477594)

[ Big mod 6](#_Toc449477595)

[2.3 Exponential 6](#_Toc449477596)

[ Fast Exponential: 6](#_Toc449477597)

[2.4 Fibonacci Number 6](#_Toc449477598)

[ First 300 Fibonacci numbers: 6](#_Toc449477599)

[ Find nth Fibonacci: f(n) = f(n-1) + f(n-2) 6](#_Toc449477600)

[2.5 Prime Number 7](#_Toc449477601)

[ Primes less than 1000: 7](#_Toc449477602)

[ Check prime number: 7](#_Toc449477603)

[ Sieve of Eratosthenes: 7](#_Toc449477604)

[ Euler's totient function 8](#_Toc449477605)

[2.6 Catalan Number 8](#_Toc449477606)

[ Một vài số Catalan: 8](#_Toc449477607)

[ Recursive method: 8](#_Toc449477608)

[ Dynamic programming method: 9](#_Toc449477609)

[ Applications of Catalan number: 9](#_Toc449477610)

# Basic

## Library

### Commonly Used Libraries:

#include <iostream> // standard input / output

#include <iomanip> // formatted output using cout

#include <fstream> // for printf

#include <sstream> // convert number to string

#include <string.h>

#include <vector> //

#include <math.h>

#include <algorithm> // stl library

## Input / Output

|  |  |
| --- | --- |
| Input int main() {  int A, B, X;  std::cin >> A >> B;  X = A + B;  std::cout << "X = " << X << std::endl;  return 0;  } | Input a line int main() {  std::string str;  std::getline(std::cin, str);  return 0;  } |

### Output with format

int main() {

double A;

std::cout << setw(4) << "A = " << A << std::endl;

return 0;

}

# Mathematics

## Division

### Divisibility by 2:

**Rule:** any even number can be divided by 2. Even numbers are multiples of 2. A number is even if ends in 0, 2, 4, 6, or 8.

int main() {

std::string numStr;

std::cin >> numStr;

divisibility3(numStr);

}

bool divisibility2(std::string numStr) { // (\*) Khi làm bài toán liên quan đến digit nên dùng string

int c = numStr.at(i) - '0';

if (c == 0 || c == 2 || c == 4 || c == 6 || c == 8) {

return true;

}

return false;

}

### Divisibility by 3:

**Rule:** A number is divisible by 3 if the sum of its digits is divisible by 3(Dùng để kiểm tra **1 số rất lớn** có chia hết cho 3?)

Example: 100,002,000 = 1 +0 +0 +0 +0 +2 +0 + 0 +0 = 3 so this very large number passes this divisibility test.

int main() {

std::string numStr;

std::cin >> numStr;

divisibility3(numStr);

}

bool divisibility3(std::string numStr) { // (\*) Khi làm bài toán liên quan đến digit nên dùng string

int c;

long long sum = 0;

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

c = numStr.at(i) - '0';

sum = sum + c;

}

if (sum % 3 == 0) {

return true;

}

return false;

}

### Divisibility by 4:

**Rule:** A number is divisible by 4 if the number's last two digits are divisible by 4.

Example: 100,002,088 = 88. Yep, this satisfies rule because 88 is divisible by 4!

bool divisibility4(std::string numStr) { //

int c = (numStr.at(numStr.size() - 1) - '0') \* 10 + (numStr.at(numStr.size() - 2) - '0');

if (c % 4 == 0) {

return true;

}

return false;

}

### Divisibility by 5:

**Rule:** A number is divisible by 5 if the its last digit is a 0 or 5.

bool divisibility5(std::string numStr) { //

int c = numStr.at(numStr.size() - 1) - '0';

if (c == 0 || c == 5) {

return true;

}

return false;

}

### Divisibility by 6:

**Rule:** A number is divisible by 6 if it is even and if the sum of its digits is divisible by 3.

bool divisibility6(std::string numStr) {

return divisibility2(numStr) & divisibility3(numStr);

}

### Divisibility by 8:

**Rule:** A number passes the test for 8 if the last three digits form a number is divisible 8.

bool divisibility8(std::string numStr) { //

int c = (numStr.at(numStr.size() - 1) - '0') \* 100 + (numStr.at(numStr.size() - 2) - '0') \* 10 + (numStr.at(numStr.size() - 3) - '0');

if (c % 8 == 0) {

return true;

}

return false;

}

### Divisibility by 11:

**Rule:** A number passes the test for 11 if the difference of the sums of alternating digits is divisible by 11.

Example:

* 946 → (9 + 6) - 4 = 11 which is, of course, evenly divided by 11
* 119,777,658 → (1 + 9 + 7 + 6 + 8) - (1 + 7 + 7 +5) = 31 - 20 = 11

bool divisibility4(std::string numStr) {

int c = 0;

int d = 0;

for (int i = 0; i < numStr.size() - 2; i += 2) {

c += (numStr.at(i) - '0');

d += (numStr.at(i + 1) - '0');

}

if ((c - d) % 11 == 0) {

return true;

}

return false;

}

|  |  |
| --- | --- |
| Greatest Common Divisor: long long gcd(long long x, long long y) {  if (!x || ! y) return x > y ? x : y;  for (int i; x % y > 0; x = y, y = i) {  i = x % y;  }  return y;  } | Least Common Multiplier: long long lcm(long long x, long long y) {  return (x \* y) / gcd(x, y);  } Greatest Common Divisor of 3 numbers: long long gcd3(long long x, long long y, long long z) {  return gcd(gcd(x, y), z);  } |

### Extended GCD:

Returns d = gcd(a,b), and give one pair x, y such that ax + by = d

long long extendedGCD(long long a, long long b, long long &x, long long &y) {

long long quotient, temp;

long long tempX = 0, tempY = 1;

y = 0; x = 1;

while (b > 0) {

quotient = a / b;

temp = b;

b = a % b;

a = temp;

temp = tempX;

tempX = x - quotient\*tempX;

x = temp;

temp = tempY;

tempY = y - quotient\*tempY;

y = temp;

}

long long gcd = a;

return gcd;

}

## Modulo

### Big mod

Example:

* Find the value of (3 ^ 3) % 5 🡪 It’s easy! 3 ^ 3 = 27 and 27 Mod 5 = 2
* Find the value of (5946 ^ 968725) % 5 🡪 It’s not so easy. Here comes the Big Mod Algorithm:

(a \* b \* c) % m = ((a % m) \* (b % m) \* (c % m)) % m.

long long int bigmod(long long a, int p, int m) {

if (p == 0) {

return 1;

}

if (p % 2 > 0) {

return ((a % m) \* (bigmod(a, p - 1, m))) % m;

}

else {

long long c = bigmod(a, p / 2, m);

return ((c % m) \* (c % m)) % m;

}

}

## Exponential

### Fast Exponential:

Example: find the value of 3100. If we use the traditional method, we will have to 99 multiplications which is slow and inefficient.

long long fastexp(long a, long n) {

if (n == 0) return 1;

else if (n % 2 == 0) return (fastexp(a, n / 2)) \* (fastexp(a, n / 2));

else return a \* (fastexp(a, n - 1));

}

## Fibonacci Number

### First 300 Fibonacci numbers:

// 0 1 1 2 3 5 8 13

// 21 34 55 89 144 233 377 610

// 987 1597 2584 4181 6765 10946 17711 28657

// 46368 75025 121393 196418 317811 514229 832040 1346269

// 2178309 3524578 5702887 9227465 14930352 24157817 39088169 63245986

// 102334155 165580141 267914296 433494437 701408733 1134903170 1836311903 2971215073

//4807526976 7778742049 12586269025

### Find nth Fibonacci: f(n) = f(n-1) + f(n-2)

long long fibonacci(int n) {

long long a = 1, b = 1, c = 0;

for (int i = 3; i <= n; ++i) {

c = a + b;

a = b;

b = c;

}

return a;

}

## Prime Number

### Primes less than 1000:

*// 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53*

*// 59 61 67 71 73 79 83 89 97 101 103 107 109 113 127 131*

*// 137 139 149 151 157 163 167 173 179 181 191 193 197 199 211 223*

*// 227 229 233 239 241 251 257 263 269 271 277 281 283 293 307 311*

*// 313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409*

*// 419 421 431 433 439 443 449 457 461 463 467 479 487 491 499 503*

*// 509 521 523 541 547 557 563 569 571 577 587 593 599 601 607 613*

*// 617 619 631 641 643 647 653 659 661 673 677 683 691 701 709 719*

*// 733 739 743 751 757 761 769 773 787 797 809 811 821 823 827 829*

*// 839 853 857 859 863 877 881 883 887 907 911 919 929 937 941 947*

*// 953 967 971 977 983 991 997*

### Check prime number:

The fastest prime tester currently available.

bool checkPrime(long long x) {

// Check for special cases

if (x <= 1) return false;

if (x <= 3) return true;

if (!(x % 2) || !(x % 3)) return false;

for (long long i = 5; i \* i <= x; i += 6) {

if (!(x % i) || !(x % (i + 2))) return false;

}

return true;

}

### Sieve of Eratosthenes:

The best prime tester and prime generator, but need very big memory. Nên dùng để tạo prime look up table.

std::bitset<10000010> bs; // Tạo 1 chuỗi binary với length = 10000010 (dùng để check prime)

std::vector<long long> primes; // vector of prime number (dùng để access nhanh danh sách prime)

int main() {

long long upper\_bound;

SieveOfEratosthenes(ceil(sqrt(upper\_bound)));

int i;

if (bs[i] == 0) {

std::cout << i << " is not prime" << std::endl;

} else {

std::cout << i << " is prime" << std::endl;

}

}

void SieveOfEratosthenes(long long upper\_bound) {

bs.set(); // set to: 111111111111...111

bs[0] = bs[1] = 0;

for (long long i = 2; i <= upper\_bound + 1; i++) {

if (bs[i]) {

for (long long j = i \* i; j <= upper\_bound + 1; j += i) {

bs[j] = 0;

}

primes.push\_back((long long)i);

}

}

}

### Euler's totient function

Find the number of positive integers less than or equal to n that are relatively prime to n

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| long eulerTotient(long n) {  long r = n;  for (long i = 2; i \* i <= n; ++i)  if (n % i == 0) {  while (n % i == 0) {  n /= i;  }  r -= r / i;  }  if (n > 1) {  r -= r / n;  }  return r;  } | |  |  |  | | --- | --- | --- | | n | φ(n) | numbers coprime to n | | 1 | 1 | 1 | | 2 | 1 | 1 | | 3 | 2 | 1, 2 | | 4 | 2 | 1,3 | | 5 | 4 | 1,2,3,4 | | 6 | 2 | 1,5 | | 7 | 6 | 1,2,3,4,5,6 | | 8 | 4 | 1,3,5,7 | | 9 | 6 | 1,2,4,5,7,8 | | 10 | 4 | 1,3,7,9 | | 11 | 10 | 1,2,3,4,5,6,7,8,9,10 | | 12 | 4 | 1,5,7,11 | | 13 | 12 | 1,2,3,4,5,6,7,8,9,10,11,12 | | 14 | 6 | 1,3,5,9,11,13 | | 15 | 8 | 1,2,4,7,8,11,13,14 | |

## Catalan Number

Trong toán tổ hợp, số Catalan là dãy các số tự nhiên xuất hiện nhiều trong các bài toán đếm, thường bao gồm những đối tượng đệ quy.

### Một vài số Catalan:

*//* [*1*](http://en.wikipedia.org/wiki/1_(number)) *1* [*2*](http://en.wikipedia.org/wiki/2_(number))[*5*](http://en.wikipedia.org/wiki/5_(number))[*14*](http://en.wikipedia.org/wiki/14_(number))[*42*](http://en.wikipedia.org/wiki/42_(number))[*132*](http://en.wikipedia.org/wiki/132_(number)) *429*

*// 1430 4862 16796 58786 208012 742900 2674440 9694845*

*// 35357670 129644790 477638700 1767263190 6564120420 24466267020 91482563640 343059613650*

*// 1289904147324 4861946401452*

### Recursive method:

// A recursive function to find nth catalan number

unsigned long int catalan(unsigned int n) {

// Base case

if (n <= 1) {

return 1;

}

// catalan(n) is sum of catalan(i)\*catalan(n-i-1)

unsigned long int res = 0;

for (int i = 0; i < n; i++) {

res += catalan(i)\*catalan(n - i - 1);

}

return res;

}

### Dynamic programming method:

// A dynamic programming based function to find nth Catalan number

unsigned long int catalanDP(unsigned int n) {

// Table to store results of subproblems

unsigned long int catalan[n + 1];

// Initialize first two values in table

catalan[0] = catalan[1] = 1;

// Fill entries in catalan[] using recursive formula

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

catalan[i] = 0;

for (int j = 0; j < i; j++) {

catalan[i] += catalan[j] \* catalan[i - j - 1];

}

}

// Return last entry

return catalan[n];

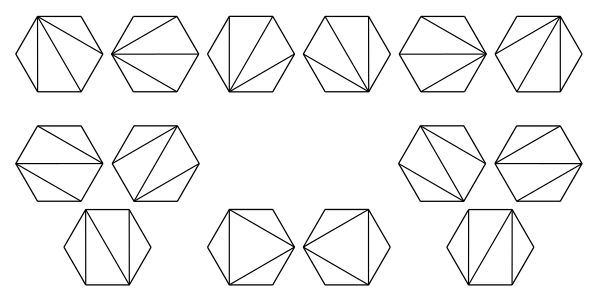
}

### Applications of Catalan number:

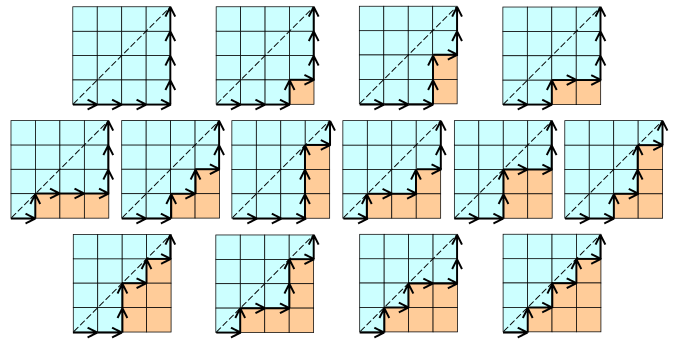
1. Number of Dyck words với độ dài . Một từ Dyck là một chuỗi ký tự gồm ký tự và ký tự , trong đó không có đoạn đầu dãy nào có nhiều ký tự hơn . Ví dụ các dyck word độ dài 6: , , , , .
2. Number of possible Binary Search Trees with keys.
3. Number of full binary trees (a rooted binary tree is full if every vertex has either two children or no children) with leaves.



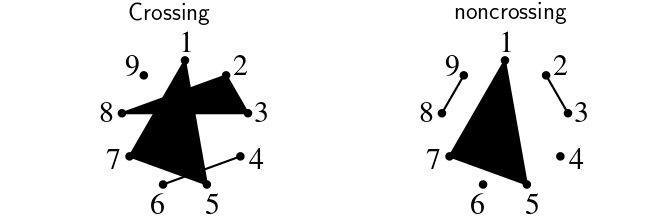
1. Number of different Unlabeled Binary Trees can be there with nodes.
2. Number of expressions containing n pairs of parentheses which are correctly matched. For , possible expressions are , , , , .
3. Số cách khác nhau để chia một đa giác lồi có cạnh thành các tam giác bằng cách nối các đỉnh của đa giác lại mà không cắt nhau.



1. Số đường đi đơn điệu theo các cầu trên một lưới có ô vuông, mà không đi lên khỏi đường chéo. Một đường đi đơn điệu là một đường đi bắt đầu từ góc dưới trên, kết thúc ở góc trên phải, chỉ đi theo hướng qua phải hoặc đi lên.



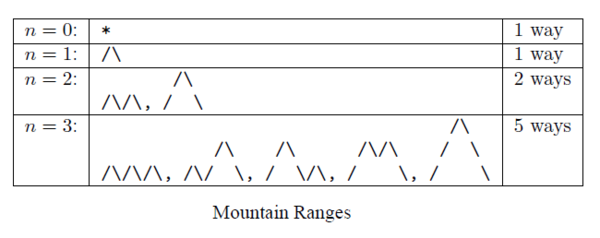
1. Number of ways to insert n pairs of parentheses in a word of letters. Ex:
   * For there are 2 ways: or .
   * For there are 5 ways, , , , , .
2. Number of noncrossing partitions of the set in which every block is of size 2. A partition is noncrossing if and only if in its planar diagram, the blocks are disjoint (i.e. don’t cross). For example, below two are crossing and non-crossing partitions of . The partition is crossing and partition is non-crossing.



1. Number of ways to tile a stairstep shape of height with rectangles. The following figure illustrates the case :



1. Số cách để vẽ hình núi với nét lên và nút xuống (không vượt xuống đường biên dưới).



1. Số hoán vị của tránh pattern (hoặc bất kỳ pattern có độ dài 3); có nghĩa là số hoán vị mà không có bất kí 3 kí tự liên tiếp tăng dần.
   * For , these permutations are , , , and .
   * For , they are , , , , , , , , , , , , and