```
#include<iostream>
#define NMax 10000
using namespace std;
typedef short Huge[NMax+3];// StarsAndBars1 3741
int n, k;
void AtribValue(Huge H,short X)
{
    H[0] = 0;
    while (X)  { H[++H[0]] = X % 10;  X /= 10;  }
}
void Mult(Huge H,short X)
{
    short i,T=0;
    for (i=1;i<=H[0];i++)
    {
        H[i]=H[i]*X+T;
        T=H[i]/10;
        H[i]=H[i]%10;
    }
    while (T) { H[++H[0]]=T%10;  T/=10;  }
}
void Divide(Huge A,short X)
{
    short i,R=0;
    for (i=A[0];i;i--)
    {
        R=10*R+A[i];
        A[i]=(R)/X;
        R%=X;
    }
    while (!A[A[0]] && A[0]>1) A[0]--;
}
void Afisez(Huge H)
{
    for(short i=H[0];i>0;--i)     cout<<H[i];
    cout<<'\n';
}

void BinComb(unsigned int n,unsigned int k)
{
    Huge res;
    AtribValue(res,1);
    if(k>n-k)   k=n-k;
    for(unsigned int i=0;i<k;++i) {  Mult(res,n-i);  Divide(res,i+1); }
    Afisez(res);
}
int main()
{
    int n;   cin>>n>>k;  n=n+k-1;   k--;
    BinComb(n,k);
    return 0;
}
```

```cpp
#include<iostream>// StarsAndBars2 #3742
#define NMax 10000
using namespace std;
typedef short Huge[NMax+3];
int n, k;
void AtribValue(Huge H,short X)
{
    H[0] = 0;
    while (X) { H[++H[0]] = X % 10; X /= 10; }
}
void Mult(Huge H,short X)
{
    short i,T=0;
    for (i=1;i<=H[0];i++)
    {
        H[i]=H[i]*X+T;
        T=H[i]/10;
        H[i]=H[i]%10;
    }
    while (T) { H[++H[0]]=T%10;  T/=10; }
}
void Divide(Huge A,short X)
{
    short i,R=0;
    for (i=A[0];i;i--)
    {
        A[i]=(R=10*R+A[i])/X;
        R%=X;
    }
    while (!A[A[0]] && A[0]>1) A[0]--;
}
void Afisez(Huge H)
{
    for(short i=H[0];i>0;--i)    cout<<H[i];
    cout<<'\n';
}
void BinComb(unsigned int n,unsigned int k)
{
    Huge res;
    AtribValue(res,1);
    if(k>n-k)    k=n-k;
    for(unsigned int i=0;i<k;++i) {  Mult(res,n-i);  Divide(res,i+1); }
    Afisez(res);
}
int main()
{    int n; cin>>n>>k;  n--; k--; BinComb(n,k);
    return 0;
}
```

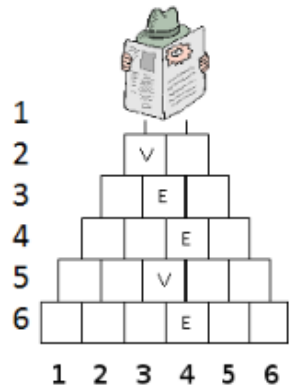
```cpp
#include <iostream> //Cladire #392
#include <fstream>
using namespace std;
long long a[1001][1001];
ifstream in("cladire.in");
ofstream out("cladire.out");
int main()
{
    int n,i,j,m;
    in>>n>>m;
    for(i=1; i<=n; i++)     a[i][1]=1;
    for(i=1; i<=m; i++)     a[1][i]=1;
    for(i=2;i<=n;i++)
        for(j=2;j<=m;j++)
            a[i][j]=(a[i-1][j]+a[i][j-1])%9901;
    out<<a[n][m];
    return 0;
```

Spion1

#1110

Spionul 008 vrea să găsească o locație secretă în junglă, având asupra lui un dispozitiv de localizare. Inițial spionul se află la intrarea în junglă pe nivelul `1` și cu fiecare pas, el avansează de la nivelul `i` la nivelul `i+1`, ajungând la locația secretă, aflată pe ultimul nivel, în poziția `u` față de marginea stângă a nivelului curent. Pentru a ajunge în locația secretă, el poate să se deplaseze cu o poziție spre Sud-Est (codificat cu caracterul `E`) sau spre Sud-Vest (codificat cu caracterul `V`), trecând de pe nivelul `i` pe nivelul `i+1` cu viteză constantă. Numărul de poziții de pe un nivel crește cu unu față de nivelul anterior, conform imaginii alăturate. Numim traseu o succesiune formată din caracterele `E` sau `V`, corespunzătoare deplasării spionului de pe nivelul `1` la locația secretă. Pentru exemplul din figura alăturată succesiunea de caractere `VEEVE` reprezintă un traseu ce corespunde locației secrete din poziția `4` a nivelului `6`.

Cunoscând succesiunea de caractere corespunzătoare unui traseu, determinați:
a) poziția locației secrete de pe ultimul nivel;
b) numărul de trasee distincte pe care le poate urma spionul plecând din poziția inițială pentru a ajunge în locația secretă corespunzătoare traseului dat. Două trasee se consideră distincte dacă diferă prin cel puțin o poziție.



```cpp
#include <iostream>
#include <cstdio>
#define M 100003
using namespace std;
int q;
char c;  long long ne = 0, nv = 0;
long long i, p, pe, pv;

long long fast_pow(long long a, long long b)
{
  long long r = 1;
  while(b > 1)
 {
    if(b & 1) r = r * a % M;
    a = a * a % M;
    b >>= 1;
  }
  return (r * a) % M;
}
int main() {
  FILE *fin = fopen("spion1.in", "r");
  FILE *fout = fopen("spion1.out", "w");
  fscanf(fin, "%d ", &q);
  while(fscanf(fin, "%c", &c) != EOF)
    if(c == 'V')  nv++;
    else if(c == 'E')  ne++;
  if(q == 1)    fprintf(fout, "%lld", ne + 1);
  else
  {
    p = pe = pv = 1; ///  (ne + nv)! / (ne! * nv!)
    for(i = 1; i <= ne + nv; i++)
    {
      p = p* i % M;
      if(i == ne)   pe = p;
      if(i == nv)  pv = p;
    }
    long long x =(( pb% M * fast_pow(pe, M - 2) )% M * fast_pow(pv, M - 2)) )% M;
    fprintf(fout, "%lld", x);
  }
  return 0;
}
```

//subimp3 #3249 Se citeste un număr natural n. Calculati si afisati câte din submultimile multimii {1, 2, ..., n} sunt formate dintr-un număr impar de elemente.

```
#include <fstream>
using namespace std;
string  nume = "date";
ifstream fin(nume + ".in");
ofstream fout(nume + ".out");
typedef unsigned long long ull;
typedef long long ll;
ull MOD = 9001;
ull expo(ull b, ull e)
{
    ull p= 1;
    while (e)
    {
        if (e % 2)
            p= (p * b) % MOD;
        b = (b % MOD * b % MOD) % MOD;
        e = e >> 1ull;
    }
    return p % MOD;
}
int main()
{
    ull n;
    cin >> n;
    cout << expo(2, n - 1);


}
```

PCR #3025 Se dă n un număr natural. Cifrele lui n se permută pentru a forma un număr natural, de aceeași lungime cu n, și care să fie palindrom. Aflați câte asemenea numere se pot obține.

```
#include <iostream>
using namespace std;
short f[10];
char s[50];
bool Palindrom(int x)
{
    int cnt(0);
    for (int i(0); i<10; i++)
        if (f[i] % 2)
            cnt++;
    return (cnt==x);
}
long long Aranjamente(long long n, long long k)
{
    long long p(1);
    for (int i=k+1; i<=n; i++)  p*=i;
    return p;
}
long long Fact(long long n)
{
    for (long long i(n-1); i>1; i--)    n*=i;
    return n;
}
int main ()
{
    cin.getline(s,50);
    long long i, cifre1(0), cifre2(1), n;
    for (i=0; s[i]; i++)  f[s[i]-48]++;
    if (i==1)   {  cout << 1; return 0; }
    if (Palindrom(i%2) == 0) { cout << 0; return 0;}
    for (int i(0); i<10; i++)
    {
        if (f[i] % 2)  f[i]--;
        if (f[i])
        {
            f[i] /= 2;
            cifre1 += f[i];
            cifre2 *= Fact(f[i]);
        }
    }
    n = Fact(cifre1) - Fact(cifre1-1) * f[0];
    cout << n / cifre2;
}
```