

```

// _____ NÚMEROS GRANDES _____

#include <cmath> //log10,pow,floor
#include <cstring> //memset
#include <string> //string class
#include <sstream> //stringstream class
#include <algorithm> //max and min
using namespace std;

const int NUM=25; //number of elementos in array

typedef long long int base_t;
const base_t BASE=pow(10,floor(log10(2)*4*sizeof(base_t)));
const int zeros=log10(BASE);

struct longint{
    longint(long long int i=0) :used(0),sign((i>=0)?1:-1){
        memset(inner,0,NUM*sizeof(base_t));
        i*=sign;
        for(;i>=BASE;i=i/BASE)
            inner[used++]=i%BASE;
        inner[used++]=i;
    }

    longint operator+(longint b) const{
        if(sign!=b.sign) return b.sign*=-1,((sign<0)?*this-b: b-*this);
        b.used=min(max(b.used,used)+1 , NUM); base_t carry=0;
        for(int i=0;i<b.used;i++){
            carry=(b.inner[i]+=inner[i]+carry)/BASE;
            b.inner[i]%=BASE;
        }
        while(b.used && !b.inner[b.used-1]) --b.used;
        return b;
    }

    longint operator-(longint b) const{
        longint const *m=this,*M=&b;
        if(used>b.used || used==b.used && inner[used-1]>b.inner[used-1])
            M=this,m=&b;
        if(b.sign!=sign) return b.sign*=-1,((*M)+(*m));
        longint res(*M); base_t carry=0;
        for(int i=0;i<=min(m->used,NUM-1);i++){
            carry=(res.inner[i]-(m->inner[i]+carry))<0?res.inner[i]+=BASE,1:0;
        }
        while(res.used && !res.inner[res.used-1]) --res.used;
        return res;
    }

    longint operator*(const longint& b) const{
        longint res;
        longint const *m,*M= (used>b.used)? (m=&b,this) : (m=this,&b);
        for(int i=0;i<m->used; i++){
            for(base_t j=0,carry=0 ; j<=(min(M->used,NUM-i-1)) ; j++){
                carry=(res.inner[j+i]=(M->inner[j]*(m->inner[i])+carry)/BASE;
                res.inner[j+i]%=BASE;
            }
            res.used=min(M->used+m->used,NUM);
            while(res.used && !res.inner[res.used-1]) --res.used;
            res.sign=sign*b.sign;
        }
        return res;
    }

    string str() const{
        stringstream ss;
        ss << sign*inner[used-1];
        for(int i=used-2;i>=0;i--){
            int cfr= inner[i] ? floor(log10(inner[i]))+1 : 1;
            for(int j=cfr;j<zeros;j++) ss << 0;
            ss << inner[i];
        }
        return ss.str();
    }
}

char sign; //sign bit

```

```

    int used; //number of base_t in used
    base_t inner[NUM]; //base array
};

longint operator+(long long int i, const longint& l){ return l+i; }
longint operator*(long long int i, const longint& l){ return l*i; }
longint operator-(long long int i, const longint& l){ return longint(i)-l;}

// _____ DPLETARIO _____
#include <iostream>
#include <vector>
#include <stack>
#include <utility>
using namespace std;
typedef pair<int,int> ii;
typedef vector<int> vi;

//calcula la primera LIS que vale. Tuneese como corresponda.
void dplis(){
    int n=8;
    int sec[]={-7,10,9,2,3,8,8,1};
    int tam[8]; //tamaño de la LIS que termina en i
    int prev[8]; //contiene el predecesor de i en la LIS que termina en i
    vector<int> lis;
    stack<int> st;
    int bestEnd=0;

    tam[0]=1;
    prev[0]=-1; //casos base;

    for(int i=1; i<n; i++){
        tam[i]=1;
        prev[i]=-1;
        for(int j=0; j<i; j++) if(sec[j]<sec[i] && tam[j]+1>tam[i]){
            tam[i]=tam[j]+1;
            prev[i]=j;
        }
    }

    for(int k=0; k<n; k++) if(tam[k]>tam[bestEnd]) bestEnd=k;

    //deshacer el camino de la LIS empezando en bestEnd.
    while(bestEnd!=-1){
        st.push(bestEnd);
        bestEnd=prev[bestEnd];
    }
    while(!st.empty()){
        cout<<st.top()<<" ";
        lis.push_back(st.top());
        st.pop();
    } //enhorabuena, en lis tienes tu LIS
}

//calcula la primera LCS que vale. (como subsecuencia)
void lcd() {
    const int n=9, m=10;
    int a[]={1,4,5,3,7,9,4,7,0};
    int b[]={7,5,3,7,9,1,4,2,0,5};
    int tabla[100][100]; //tamaño de la maxima LCS entre las subcadenas que empiezan en i y
j
    ii next[100][100]; //principio de la LCS en ambas subcadenas
    int i,j;

    memset(tabla,0, sizeof tabla);

    for(i=0; i<n; i++) tabla[i][m-1]=(a[i]==b[m-1])?1:0;
    for(j=0; j<m; j++) tabla[n-1][j]=(a[n-1]==b[j])?1:0;

    for(i=n-1; i>=0; i--) for(j=m-1; j>=0; j--) if(a[i]==b[j]){
        tabla[i][j]=tabla[i+1][j+1]+1;
        next[i][j]=make_pair(i+1,j+1);
    }else if(tabla[i][j+1]>tabla[i+1][j]){

```

```

        tabla[i][j]=tabla[i][j+1];
        next[i][j]=next[i][j+1];
    }else{
        tabla[i][j]=tabla[i+1][j];
        next[i][j]=next[i+1][j];
    }
    cout<<tabla[0][0]<<endl;

    for(i=0; i<n; i++) {for(j=0; j<m; j++)
    {
        cout<<'('<<next[i][j].first<<','<<next[i][j].second<<") ";
    }cout<<endl;}

    ii indice=(a[0]==b[0])?make_pair(0,0):next[0][0];

    while(indice.first || indice.second)
    {
        cout<<indice.first<<" "<<indice.second<<endl;
        indice=next[indice.first][indice.second];
    }
}

//calcula la primera LCSS que vale. (subcadena, no subsecuencia)
//en realidad, se resuelve en  $O((n+m)\log(n+m))$  usando un suffix array)
void dplcss()
{
    const int n=9, m=10;
    int a[]={1,4,5,3,7,9,4,7,0};
    int b[]={7,5,3,7,9,1,4,2,0,5};
    int tabla[100][100]; //tamaño de la coincidencia entre a[i] y b[j]
    int i,j;
    int bestA=0, bestB=0;

    for(i=0; i<n; i++) tabla[i][m-1]=(a[i]==b[m-1])?1:0;
    for(j=0; j<m; j++) tabla[n-1][j]=(a[n-1]==b[j])?1:0;

    for(i=n-1; i>=0; i--) for (j=m-1; j>=0; j--)
        tabla[i][j]=(a[i]==b[j])?tabla[i+1][j+1]+1:0;

    for(i=0; i<n; i++) for(j=0; j<m; j++) if(tabla[bestA][bestB]<tabla[i][j]){
        bestA=i; bestB=j;
    }
    cout<<bestA<<" "<<bestB<<" "<<tabla[bestA][bestB];
    //tu LCS empieza en bestA,bestB y tiene tamaño tabla[bestA][bestB];
}

// _____ MATESLETARIO _____
#include <bitset> // compact STL for Sieve, more efficient than vector<bool>!
#include <cmath>
#include <cstdio>
#include <map>
#include <vector>
using namespace std;

typedef long long ll;
typedef vector<int> vi;
typedef pair<int,int> ii;
typedef map<int, int> mii;

ll _sieve_size;
bitset<10000010> bs; //  $10^7$  should be enough for most cases
vi primes; // compact list of primes in form of vector<int>

// llamar a este metodo nada mas empezar.
void sieve(ll upperbound) { // create list of primes in [0..upperbound]
    _sieve_size = upperbound + 1; // add 1 to include upperbound
    bs.set(); // set all bits to 1
    bs[0] = bs[1] = 0; // except index 0 and 1
    for (ll i = 2; i <= _sieve_size; i++) if (bs[i]) {
        // cross out multiples of i starting from i * i!
        for (ll j = i * i; j <= _sieve_size; j += i) bs[j] = 0;
        primes.push_back((int)i); // also add this vector containing list of primes
    }
}

```

```

} } // call this method in main method

bool isPrime(ll N) { // a good enough deterministic prime tester
    if (N <= _sieve_size) return bs[N]; // O(1) for small primes
    for (int i = 0; i < (int)primes.size() && i*i<=N; i++)
        if (N % primes[i] == 0) return false;
    return true; // it takes longer time if N is a large prime!
} // note: only work for N <= (last prime in vi "primes")^2

vi primeFactors(ll N) { // remember: vi is vector of integers, ll is long long
    vi factors; // vi `primes' (generated by sieve) is optional
    ll PF_idx = 0, PF = primes[PF_idx]; // using PF = 2, 3, 4, ..., is also ok
    while (N != 1 && (PF * PF <= N)) { // stop at sqrt(N), but N can get smaller
        while (N % PF == 0) { N /= PF; factors.push_back(PF); } // remove this PF
        PF = primes[++PF_idx]; // only consider primes!
    }
    if (N != 1) factors.push_back(N); // special case if N is actually a prime
    return factors; // if pf exceeds 32-bit integer, you have to change vi
}

ll EulerPhi(ll N) {
    ll PF_idx = 0, PF = primes[PF_idx], ans = N; // start from ans = N
    while (N != 1 && (PF * PF <= N)) {
        if (N % PF == 0) ans -= ans / PF; // only count unique factor
        while (N % PF == 0) N /= PF;
        PF = primes[++PF_idx];
    }
    if (N != 1) ans -= ans / N; // last factor
    return ans;
}

//Algoritmo de Floyd de busqueda de ciclos.
//pongase la funcion f de la que calcular ciclos aqui
int f(int x) { return 2; }

ii floydCycleFinding(int x0) { // function "int f(int x)" must be defined earlier
    // 1st part: finding v, hare's speed is 2x tortoise's
    int tortoise = f(x0), hare = f(f(x0)); // f(x0) is the element/node next to x0
    while (tortoise != hare) { tortoise = f(tortoise); hare = f(f(hare)); }
    // 2nd part: finding mu, hare and tortoise move at the same speed
    int mu = 0; hare = x0;
    while (tortoise != hare) { tortoise = f(tortoise); hare = f(hare); mu++; }
    // 3rd part: finding lambda, hare moves, tortoise stays
    int lambda = 1; hare = f(tortoise);
    while (tortoise != hare) { hare = f(hare); lambda++; }
    return ii(mu, lambda);
}

//Algoritmo de Euclides extendido
//ax+by=g=gcd(a,b)
void eea(int a, int b,
        int& gcd, int& x, int& y) {
    x=0, y=1;
    int u=1, v=0, m, n, q, r;
    gcd = b;
    while (a!=0) {
        q=gcd/a; r=gcd%a;
        m=x-u*q; n=y-v*q;
        gcd=a; a=r; x=u; y=v; u=m; v=n;
    }
}

//ecuacion diofantica
//con d=mcd(A,B)
x=x0 + L*(B/d)
y=y0 - L*(A/d)

```

```

// _____ MISCLETARIO _____
#include <vector>
using namespace std;

typedef pair<int, int> ii;
typedef vector<ii> vii;
typedef vector<int> vi;

//mascaras de bits
#define isOn(S, j) (S & (1 << j))
#define setBit(S, j) (S |= (1 << j))
#define clearBit(S, j) (S &= ~(1 << j))
#define toggleBit(S, j) (S ^= (1 << j))
#define lowBit(S) (S & (-S))
#define setAll(S, n) (S = (1 << n) - 1)

//Arbol de segmentos. Permite buscar el minimo elemento de un array LEERLO
// Segment Tree Library: The segment tree is stored like a heap array
void st_build(vi &st, const vi &A, int vertex, int L, int R) {
    if (L == R) // as L == R, either one is fine
        st[vertex] = L; // store the index
    else { // recursively compute the values in the left and right subtrees
        int nL = 2 * vertex, nR = 2 * vertex + 1;
        st_build(st, A, nL, L, (L + R) / 2);
        st_build(st, A, nR, (L + R) / 2 + 1, R);
        int lContent = st[nL], rContent = st[nR];
        int lValue = A[lContent], rValue = A[rContent];
        st[vertex] = (lValue <= rValue) ? lContent : rContent;
    } }

void st_create(vi &st, const vi &A) { // if original array size is N,
    // the required segment tree array length is 2*2^(floor(log2(N)) + 1);
    int len = (int)(2*pow(2.0, floor((log((double)A.size())/log(2.0)) + 1)));
    st.assign(len, 0); // create vector of size 'len' and fill it with zeroes
    st_build(st, A, 1, 0, (int)A.size() - 1); // recursive build
}

int st_rmq(vi &st, const vi &A, int vertex, int L, int R, int i, int j) {
    if (i > R || j < L) return -1; // current segment outside query range
    if (L >= i && R <= j) return st[vertex]; // inside query range

    // compute the min position in the left and right part of the interval
    int p1 = st_rmq(st, A, 2 * vertex, L, (L+R) / 2, i, j);
    int p2 = st_rmq(st, A, 2 * vertex + 1, (L+R) / 2 + 1, R, i, j);

    // return the position where the overall minimum is
    if (p1 == -1) return p2; // if we try to access segment outside query
    if (p2 == -1) return p1; // same as above
    return (A[p1] <= A[p2]) ? p1 : p2; }

int st_rmq(vi &st, const vi& A, int i, int j) { // function overloading
    return st_rmq(st, A, 1, 0, (int)A.size() - 1, i, j); }

int st_update_point(vi &st, vi &A, int node, int b, int e, int idx, int new_value) {
    // this update code is still preliminary, i == j
    // must be able to update range in the future!
    int i = idx, j = idx;

    // if the current interval does not intersect
    // the update interval, return this st node value!
    if (i > e || j < b)
        return st[node];

    // if the current interval is included in the update range,
    // update that st[node]
    if (b == i && e == j) {
        A[i] = new_value; // update the underlying array
        return st[node] = b; // this index
    }

    // compute the minimum position in the
    // left and right part of the interval
    int p1, p2;
    p1 = st_update_point(st, A, 2 * node, b, e, (b + e) / 2, idx, new_value);

```

```

    p2 = st_update_point(st, A, 2 * node + 1, (b + e) / 2 + 1, e, idx, new_value);

    // return the position where the overall minimum is
    return st[node] = (A[p1] <= A[p2]) ? p1 : p2;
}

int st_update_point(vi &st, vi &A, int idx, int new_value) {
    return st_update_point(st, A, 1, 0, (int)A.size() - 1, idx, new_value); }

//Arbol de Fenwick. Permite resolver el problema de la suma en un rango
//con actualizaciones. (RSQ(a,b)=suma de los a[i], i entre a y b)
// initialization: n + 1 zeroes, ignoring index 0, just using index [1..n]
void ft_create(vi &ft, int n) { ft.assign(n + 1, 0); }

int ft_rsq(const vi &ft, int b) { // returns RSQ(1, b)
    int sum = 0; for (; b; b -= LSONe(b)) sum += ft[b];
    return sum; }

int ft_rsq(const vi &ft, int a, int b) { // returns RSQ(a, b)
    return ft_rsq(ft, b) - (a == 1 ? 0 : ft_rsq(ft, a - 1)); }

// adjusts value of the k-th element by v (v can be +ve/inc or -ve/dec)
void ft_adjust(vi &ft, int k, int v) { // note: n = ft.size() - 1
    for (; k < (int)ft.size(); k += LSONe(k)) ft[k] += v; }

```

//Backtracking

Funcion Backtracking (Etapai) devuelve: boolean

Inicio

```

    Éxito = falso;
    IniciarOpciones(i, GrupoOpciones o);
    Repetir
        SeleccionarnuevaOpcion(o, Opcion n);
        Si (Aceptable(n)) entonces
            AnotarOpcion(i, n);
            SiSolucionCompleta(i) entonces
                Éxito = verdadero;
            Sino
                Éxito = Backtracking(i+1);
                Si Éxito = false entonces
                    cancelamosAnotacion(i, n);
            finsi;
        Finsi;
    Finsi;
    Hasta (Éxito = verdadero) o (NoQuedanOpciones(o));
    Retorna Éxito;

```

Fin;

```

// _____ STRINGLETARIO _____
#include <cstdio>
#include <cstring>
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;

#define MAXN 100010

//Knuth-Morris-Pratt: sirve para buscar la cadena P
//en la cadena T. Ojo, hay que preprocesar P usando
//kmpPreprocess(). MAXN<=100010
char T[MAXN], P[MAXN]; // T = text, P = pattern
int b[MAXN], n, m; // b = back table, n = length of T, m = length of P

void kmpPreprocess() { // call this before calling kmpSearch()
    int i = 0, j = -1; b[0] = -1; // starting values
    while (i < m) { // pre-process the pattern string P
        while (j >= 0 && P[i] != P[j]) j = b[j]; // if different, reset j using b
        i++; j++; // if same, advance both pointers
    }
}

```

```

        b[i] = j; // observe i = 8, 9, 10, 11, 12 with j = 0, 1, 2, 3, 4
    } } // in the example of P = "SEVENTY SEVEN" above

void kmpSearch() { // this is similar as kmpPreprocess(), but on string T
    int i = 0, j = 0; // starting values
    while (i < n) { // search through string T
        while (j >= 0 && T[i] != P[j]) j = b[j]; // if different, reset j using b
        i++; j++; // if same, advance both pointers
        if (j == m) { // a match found when j == m
            printf("P is found at index %d in T\n", i - j);
            j = b[j]; // prepare j for the next possible match
        } } }

/*ejemplo de uso
int main() {
    n = (int)strlen(gets(T));
    m = (int)strlen(gets(P));

    kmpPreprocess();
    kmpSearch();

    return 0;
}*/

//Array de sufijos. r contiene la posicion del iesimo sufijo ordenadamente de s
//pongo un ejemplo de uso mas abajo
//Ojo: MAXN<1000005
//Importante: añadir un '$' al final de las cadenas para romper desempates en el
//count sort.
int n,t; //n es el tamaño de la cadena
int p[MAXN],r[MAXN],h[MAXN];
//p es el inverso del suffix array, no usa indices del suffix array ordenado
//h el el tamaño del lcp entre el i-esimo y el i+1-esimo elemento de suffix array ordenado
string s;
void fix_index(int *b, int *e) {
    int pkm1, pk, np, i, d, m;
    pkm1 = p[*b + t];
    m = e - b; d = 0;
    np = b - r;
    for(i = 0; i < m; i++) {
        if (((pk = p[*b+t]) != pkm1) && !(np <= pkm1 && pk < np+m)) {
            pkm1 = pk;
            d = i;
        }
        p[(b++)] = np + d;
    }
}

bool comp(int i, int j) {
    return p[i + t] < p[j + t];
}

void suff_arr() {
    int i, j, bc[256];
    t = 1;
    for(i = 0; i < 256; i++) bc[i] = 0; //alfabeto
    for(i = 0; i < n; i++) ++bc[int(s[i])]; //counting sort inicial del alfabeto
    for(i = 1; i < 256; i++) bc[i] += bc[i - 1];
    for(i = 0; i < n; i++) r[--bc[int(s[i])]] = i;
    for(i = n - 1; i >= 0; i--) p[i] = bc[int(s[i])];
    for(t = 1; t < n; t *= 2) {
        for(i = 0, j = 1; i < n; i = j++) {
            while(j < n && p[r[j]] == p[r[i]]) ++j;
            if (j - i > 1) {
                sort(r + i, r + j, comp);
                fix_index(r + i, r + j);
            }
        }
    }
}

void lcp() {
    int tam = 0, i, j;
    for(i = 0; i < n; i++) if (p[i] > 0) {
        j = r[p[i] - 1];

```

```

        while(s[i + tam] == s[j + tam]) ++tam;
        h[p[i] - 1] = tam;
        if (tam > 0) --tam;
    }
    h[n - 1] = 0;
}
/*ejemplo de uso
int main(){
    s="margarita$";
    n=s.size();
    suff_arr();
    lcp();
    for(int i=0;i<n;i++)cout<<r[i]<<" ";cout<<endl;
    for(int i=0;i<n;i++)cout<<h[i]<<" ";cout<<endl;
    return 0;
}*/

// _____ IOMANIP _____

// FLAGS DE FORMATO
- boolalpha/noboolalpha: muestra su valor verdadero/falso
    cout << boolalpha << b; := true
- showbase/noshowbase: muestra el prefijo de la base (0x_hex_, 0_oct_, nada_dec_).
    cout << hex << showbase << 14 << endl; := 0x14
- showpoint/noshowpoint: muestra el punto de _double/float_. Se puede combinar con
    setprecision.
    cout.precision(5); cout << showpoint << 30.0 << 10000.0 << 3.1416; := 30.000 10000.
3.1416
- showpos/noshowpos: muestra el signo + para numeros positivos
    cout << showpos << 1 << 0 << -1; := +1 +0 -1
- skipws: elimina todos los espacios en blanco de la entrada
    istrstream iss(" 123");
    iss >> skipws >> a >> b >> c;
    cout << a << b << c; := 123
- uppercase/nouppercase: convierte el texto a mayusculas.
    cout << showbase << hex;
    cout << uppercase << 77; := 0X4D
    cout << nouppercase << 77; := 0x4d

- fixed: escribe _numeros flotantes_ en notacion de punto fijo
- scientific: escribe _numeros flotantes_ en notación científica
    cout.setprecision(5);
    cout <<          << 3.1415926532 << 2006.0 << 1.0e-10;
        3.1416      2006          1e-010
    cout << fixed    << 3.1415926532 << 2006.0 << 1.0e-10;
        3.14159      2006.00000      0.00000
    cout << scientific << 3.1415926532 << 2006.0 << 1.0e-10;
        3.14159e+000      2.00600e+003      1.00000e-10

- internal: espacia el texto separando el signo del valor
- left: muestra el texto alineado a la izquierda con el ancho indicado
- right: muestra el texto a la derecha con el ancho indicado

    cout.width(6); cout << internal << -77 << endl; := - 77
    cout.width(6); cout << left << -77 << endl; := -77
    cout.width(6); cout << right << -77 << endl; := -77

// MANIPULADORES PARAMETRIZADOS
- setfill(char c): rellena el ancho indicado con el caracter pasado por parametro
    cout << setfill('x') << setw(10);
    cout << 77 << endl; := xxxxxxxx77
- setprecision: imprime el número según la precisión indicada. Si es fixed rellena con 0
    cout << setprecision(5) << 3.14159 << endl; := 3.1416
    cout << setprecision(9) << 3.14159 << endl; := 3.14159
    cout << fixed;
    cout << setprecision(5) << 3.14159 << endl; := 3.14159
    cout << setprecision(9) << 3.14159 << endl; := 3.1415900000
- setw(int n): setea el ancho de linea indicado
    cout << setw(10);
    cout << 77 << endl;

```


SCANF-PRINTF

```
#include <stdio.h>
```

A *format specifier* for `fscanf` follows this prototype:

```
%[*][width][length]specifier
```

% specifier:

i, u – Integer, d – decimal integer, x – hexadecimal integer, f – floating point number, c – char, s – string of chars.

[<i>characters</i>]	Scanset	Any number of the characters specified between the brackets. A dash (–) that is not the first character may produce non-portable behavior in some library implementations.
[^ <i>characters</i>]	Negated scanset	Any number of characters none of them specified as <i>characters</i> between the brackets.
n	Count	No input is consumed. The number of characters written so far is stored in the pointed location.
%	%	A % followed by another % matches a single %.
*	An optional starting asterisk indicates that the data is to be read from the stream but ignored	
width	Specifies the maximum number of characters to be read in the current reading operation (optional).	

specifiers						
length	d i	u o x	f e g a	c s [] [^]	p	n
(none)	int*	unsigned int*	float*	char*	void**	int*
hh	signed char*	unsigned char*				signed char*
h	short int*	unsigned short int*				short int*
l	long int*	unsigned long int*	double*	wchar t*		long int*
ll	long long int*	unsigned long long int*				long long int*
j	intmax t*	uintmax t*				intmax t*
z	size t*	size t*				size t*
t	ptrdiff t*	ptrdiff t*				ptrdiff t*
L			long double*			

A *format specifier* for `printf` follows this prototype:

```
%[flags][width][.precision][length]specifier
```

flags	description
–	Left-justify within the given field width; Right justification is the default (see <i>width</i> sub-specifier).
+	Forces to precede the result with a plus or minus sign (+ or –) even for positive numbers. By default, only negative numbers are preceded with a – sign.
(space)	If no sign is going to be written, a blank space is inserted before the value.
#	Used with o, x or X specifiers the value is preceeded with 0, 0x or 0X respectively for values different than zero. Used with a, A, e, E, f, F, g or G it forces the written output to contain a decimal point even if no more digits follow. By default, if no digits follow, no decimal point is written.
0	Left-pads the number with zeroes (0) instead of spaces when padding is specified (see <i>width</i> sub-specifier).

width	description
(number)	Minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded with blank spaces. The value is not truncated even if the result is larger .

*	The <i>width</i> is not specified in the <i>format</i> string, but as an additional integer value argument preceding the argument that has to be formatted.
---	-------------------------------------------------------------------------------------------------------------------------------------------------------------

.precision	description
.number	For integer specifiers (d, i, o, u, x, X): <i>precision</i> specifies the minimum number of digits to be written. If the value to be written is shorter than this number, the result is padded with leading zeros. The value is not truncated even if the result is longer. A <i>precision</i> of 0 means that no character is written for the value 0. For a, A, e, E, f and F specifiers: this is the number of digits to be printed after the decimal point. For g and G specifiers: This is the maximum number of significant digits to be printed. For s: this is the maximum number of characters to be printed. By default all characters are printed until the ending null character is encountered. If the period is specified without an explicit value for <i>precision</i> , 0 is assumed.
.*	The <i>precision</i> is not specified in the <i>format</i> string, but as an additional integer value argument preceding the argument that has to be formatted.

```
int main ()
{
    char str [80];
    int i;

    printf ("Enter your family name: ");
    scanf ("%s",str);
    printf ("Enter your age: ");
    scanf ("%d",&i);
    printf ("Mr. %s , %d years old.\n",str,i);
    printf ("Enter a hexadecimal number: ");
    scanf ("%x",&i);
    printf ("You have entered %#x (%d).\n",i,i);

    return 0;
}
```

```
//output
Enter your family name: Soulie
Enter your age: 29
Mr. Soulie , 29 years old.
Enter a hexadecimal number: ff
You have entered 0xff (255).
```

// ejemplo printf

```
printf ("Characters: %c %c \n", 'a', 65);
printf ("Decimals: %d %ld\n", 1977, 650000L);
printf ("Preceding with blanks: %10d \n", 1977);
printf ("Preceding with zeros: %010d \n", 1977);
printf ("Some different radixes: %d %x %o %#x %#o \n", 100, 100, 100, 100, 100);
printf ("floats: %4.2f %+.0e %E \n", 3.1416, 3.1416, 3.1416);
printf ("Width trick: %*d \n", 5, 10);
printf ("%s \n", "A string");
```

//Output:

```
Characters: a A
Decimals: 1977 650000
Preceding with blanks:          1977
Preceding with zeros: 0000001977
Some different radixes: 100 64 144 0x64 0144
floats: 3.14 +3e+000 3.141600E+000
Width trick:    10
A string
```