C++ tanácsok

Input/output

Beolvasáshoz és kiíráshoz az input/output stream-ek (*cin*, *cout*) használata nagyon kényelmes, de alapértelmezett beállításokkal kicsit lassú. Érdemes kikapcsolni a C stílusú beolvasás-kiírással (printf, scanf) való szinkronizációt a main függvény elején az `ios::sync_with_stdio(false);` paranccsal.
A feladatok nagy része nem interaktív feladat, ezeknél javasoljuk a *cin* és *cout* közötti kapcsolat megszüntetését is a `cin.tie(0);` paranccsal. Ezt interaktív feladat esetében ne használjátok!
Ha sok sort kell kiírni, akkor a sortörések kiírására a `cout << endl;` parancs használata lassíthatja a programot, mert mindig flush-olja a kimenetet. Javasoljuk az `endl` helyett a `'\n'` karakter használatát a sortörés kiírására (hagyományos, nem interaktív feladatoknál). Interaktív feladatoknál hasznos lehet az `endl` használata.
Ha magatoknak akartok debug üzeneteket kiírni, akkor arra használjátok a *cout* helyett a *cerr* kimeneti stream-et. Az oda írt adatokat az értékelő rendszer figyelmen kívül hagyja, viszont a program futási idejét növelhetik.

Nagy számok

C++ -ban az egész szám típusoknak van alsó és felső határa. Az `int` típus esetén ez általában -2 147 483 648 … 2 147 483 647, és ha egy számolás közben ezt túllépjük, akkor nem jelez hibát a program, csak hibás eredményt kapunk (túlcsordul). Ha ennél nagyobb számokkal kell számolni, akkor használjátok a `long long` típust, amelynek tartománya legalább $-9{,}22 \cdot 10^{18}$ … $9{,}22 \cdot 10^{18}$. Ha ennél is nagyobb számok lehetnek, akkor lehet őket például egy string-ben tárolni.

Példa program

Álljon itt egy példa program, ami beolvas két számot, majd kiírja ezek összegét.

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    long long a, b;
    cin >> a >> b;
    cout << a + b << '\n';
    return 0;
}
```

Python tanácsok

Input/output

Beolvasáshoz és kiíráshoz használjátok a `sys` modulból az `stdin.readline()` és `stdout.write()` függvényeket. Ezek némileg gyorsabbak, mint az `input()` és a `print()`.
Az `stdin.readline()` egy sort ad vissza string-ként, a sorvége karakterrel együtt.
Amennyiben a sor sok, szóközzel elválasztott számot tartalmaz, ezeket a `map(int, stdin.readline().split())` paranccsal kaphatjátok meg. Ez egy map object-et ad vissza, ami iterable, tehát például egy for ciklussal végig lehet menni rajta (egyszer), vagy listává lehet alakítani a list(...) függvénnyel.
Ha egy sor csak két számot tartalmaz, akkor a fentinél gyorsabb a következő két parancs használata: `ab = stdin.readline().split()` majd `a,b = int(ab[0]), int(ab[1])`
As `stdout.write()`-nak egy paramétere van, aminek string-nek kell lennie, és ezt sorvége karakter nélkül írja a kimenetre. Tehát ha például két külön sorba szeretnétek írni az `a` és `b` számokat, akkor ezt az `stdout.write(str(a) + '\n' + str(b) + '\n')` paranccsal tudjátok megtenni.

Main függvény

Definiálj egy `main()` függvényt, amibe a program törzsét írod, majd ezt egyedüli parancsként hívd meg. Ez a trükk tapasztalataink szerint gyorsít a programon.

Példa program

Álljon itt egy példa program, ami beolvas n számot, majd kiírja ezek összegét és szorzatát.

```
from sys import stdin, stdout

def main():
 n = int(stdin.readline())
 szamok = list(map(int, stdin.readline().split()))
 osszeg = sum(szamok)
 szorzat = 1
 for x in szamok:
   szorzat *= x
 stdout.write(str(osszeg) + '\n' + str(szorzat) + '\n')

main()
```