Tartalomjegyzék

0.1. Mi az a standard be- és kimenet?

A standard bemenet (`stdin`) és kimenet (`stdout`) nem más, mint két olyan kommunikációs csővezeték (`pipe`) megnevezése, melyek a program indulásakor, a programozó részéről minden további erőfeszítés nélkül, automatikusan létrejönnek és rendelkezésre állnak. Egy „csővezeték” adatok egyirányú továbbítására szolgál két program között: két vége van, az egyik végén csak írni lehet bele, a másikon pedig a beleírt adatot (ugyanabban a sorrendben) csak kiolvasni.[1]

Ennek megfelelően az `stdin` az a csővezeték, melynek olvasható végéből a program a bemenetet olvasssa, az `stdout`, melynek írható végébe a program a kimenetet írja. Ha interaktív parancsértelmezőből minden további beállítás nélkül elindítjuk a programot, akkor az `stdin` írható végére – a parancsértelmezőn keresztül – általában a billentyűzet, az `stdout` olvasható végére pedig a képernyő csatlakozik.

Fontos hangsúlyozni, hogy *„a parancsértelmezőn keresztül”*, és *„minden további beállítás nélkül”*. Ha ugyanis pl. a `test.exe`-t a következőképpen indítjuk el:

```
C:\> test.exe < be.txt > ki.txt
```

akkor a parancsértelmező a `test.exe` standard bemenetére nem a billentyűzetet, hanem a `be.txt` fájlt csatlakoztatja (egyszeri végigolvasásra), a program kimenete pedig a `ki.txt` fájlba kerül – mindebből a `test.exe` semmit sem vesz észre. Ezzel a módszerrel automatizáljuk a két pontversenyben a megoldások tesztelését.

0.2. A két mintafeladat leírása

A továbbiakban két mintafeladaton keresztül mutatjuk be, hogyan lehet a standard adatfolyamokat, illetve a hozzájuk tartozó formázott beolvasó és kiíró rutinokat hatékonyan és egyszerűen használni. A feladatokat abban a stílusban fogalmazzuk meg, ahogy a feladatkiírásban szerepelni szoktak:

[1] Ezen kívül még van egy meghatározott méretű puffer: ha az olvasás lassabb, mint az írás, akkor (a bizonyos méretig) a beleírt adatok átmenetileg a pufferben kerülnek tárolásra, ha az olvasás a gyorsabb, akkor az várakozni kényszerül, amíg van mit olvasni.

1. **Kiválasztás**

   Írjunk programot, amely egy N hosszú tömbnek kiírja K darab, pozíciójával megadott elemét.

   A standard bemenet első sorában két, szóközzel elválasztott szám, az eredeti tömb N hossza, majd a kiírandó elemek $0 \leq K \leq N$ száma található. A bemenet második sorában egy-egy szóközzel elválasztva az eredeti $x_1, x_2, \ldots, x_N$ (egész, 64 biten ábrázolható) tömbelemek, míg harmadik sorában rendre a kiírandó $i_1, i_2, \ldots, i_K$ indexek szerepelnek.

   A standard kimenet egyetlen sorába egyetlen éppen K darab, szöközzel elválasztott szám kerüljön, a fenti indexek által kijelölt tömbelemek.

2. **Tükrözés**

   Írjunk programot, amely N darab karakterlánc tükrözöttjét írja ki.

   A standard bemenet első sorában egyetlen szám, a sztringek N száma található. Az ezt követő N darab sor mindegyike egy-egy sztringet tartalmaz (figyelem: a sztringben lehet szóköz, tabulátor, ..., csak újsor nem).

   A standard kimenet N darab sorába kerüljenek a tükrözött sztringek, a bemenet sorrendjében.

0.3. A standard adatfolyamok kezelése Pascalban

Pascal nyelvben a standard adatfolyamok kezeléséhez mindössze négy függvényre lesz szükségünk: `read/readln` a beolvasáshoz, illetve `write/writeln` a kiíráshoz. Mielőtt továbbhaladnánk, a két legfontosabb Pascal-specifikus tudnivaló:

Figyelem! A fenti négy függvény csak abban az esetben használja a standard adatfolyamokat, amíg a `CRT` modult nem használjuk. Ha a programban szerepel, hogy „`uses crt;`”, abban az esetben a `read` (a parancsértelmezőt megkerülve!) a billentyűzetről olvas, míg a `write` a képernyőre ír, emellett olyan kényelmi szolgáltatások is elérhetőek lesznek mint a `clrscr`. A `CRT` modul használata esetén viszont a be-/kimenet nem irányítható fájlba, ezáltal lehetetlenné válik a tesztelés. Emiatt soha nem használjuk a `CRT` modult, illetve az általa nyújtott `clrscr` és egyéb függvényeket!

Figyelem! A fájlokról, és az azt kezelő függvényekről (`assign`, `rewrite`, `close`) szintén felejtkezzünk el, ha a feladat a standard bemenetet említi, akkor a verseny során, szinte kivétel nélkül, soha nem kell fájlokat használni, és ne is használjunk!

0.3.1. „Kiválasztás” Pascalban

A `read` eljáráshívás az argumentumként átadott (pl.: `read(a,b,c);`) változó(k)ba olvas be értéket, viselkedése a változók típusától függ. Ha számokról van szó, nagyon egyszerű dolgunk van: a `read` az összes whitespace karaktert (szóköz, tabulátor és újsor itt egységesen viselkedik) megeszi a következő számig, majd azt tízes számrendszerben értelmezve beolvassa az adott változóba.

A `readln` ennél annyivan több, hogy a szám után *mindent* megeszik (akár további számokat!) a következő újsor karakterig, azt is beleértve. Fontos kiemelni, hogy az „új sor” operációs rendszertől függően más karakterekkel lehet jelölve (`'\n'`, `"\r\n"` vagy `"\n\r"`), a beépített függvények ezzel is megbírkóznak, és minden platformon jól fognak működni. (Ellentétben azzal, ha magunk kezdjük el a bemenetet karakterenként bogarászni!)

A `write` szintén több argumentumot fogad, egymás után kiírja formázva (tehát pl. egy számot tízes számrendszerben) a változók értékeit – a `writeln` pedig az után még egy új sort is kezd.

Ennek megfelelően a forráskód a feladatra a következő. Vegyük észre a `longint` (32 bites előjeles egész), és `int64` (64 bites előjeles egész) típusokat!

```
program stdio;

var i,n,k : longint;
    elems : array[1..1000000] of int64;
    indices : array[1..1000000] of longint;

begin
    readln(n,k);
    for i:=1 to n do read(elems[i]);
    for i:=1 to k do read(indices[i]);

    for i:=1 to k do
    begin
        write(elems[indices[i]],' ');
    end;
    writeln;
end.
```

0.3.2. „Tükrözés” Pascalban

Pascalban a sztringek tárolására szolgáló `string` típus hátránya, hogy legfeljebb 255 karakterből állhat. Helyette használjuk inkább az `AnsiString` típust, mellyel ugyanúgy tudunk dolgozni, mindössze a hosszának lekérdezése történik picit másképp: ez a `length(s)` hívással kapható meg.

A `read` eljárás szempontjából némileg másként viselkedik egy sztring, mint az egész szám. Itt már nem úgyanúgy viselkednek a különböző whitespace karakterek. A szóköz és tabulátor meg fog jelenni a beolvasott sztringben, az újsor nem, viszont ez fogja megszabni, mennyi olvasódik be a bemenetről az átadott sztring változóba. Egészen pontosan az a része a bemenetnek a változóba, amit az aktuális olvasási pozíciótól kezdődik, és egészen a következő újsorig tart (az újsor már nem).

Fontos megemlíteni, hogy ha pl. a bemenet első sorában lévő egész számot egy „`read(n);`” hívással olvastuk be, akkor ez után a bemenetben az olvasási pozíciónk épp az ezt követő újsor előtt lesz. Ha ekkor sztringet olvasunk, akkor egy üressztringet kapunk eredményül. Ha megint olvasunk, ismét üressztring lesz az eredmény – egészen addig, amíg az újsort egy `readln` hívással le nem vesszük a bemenetről.

```
program stdio;

function rev(s : AnsiString) : AnsiString;
var i,k : longint;
    c : char;
begin
    k := length(s);
    for i:=1 to k div 2 do
    begin
        c := s[i];
        s[i] := s[k-i+1];
        s[k-i+1] := c;
    end;
    rev := s;
end;

var i,n : longint;
    str : AnsiString;
begin
    readln(n);
    for i:=1 to n do
    begin
        readln(str);
        writeln(rev(str));
    end;
end.
```

0.4. A standard adatfolyamok kezelése C-ben

0.4.1. „Kiválasztás” C-ben

A 64 bites előjeles típus a `long long`, melyet az `"%lld"` formátumsztringgel lehet beolvasni, illetve kiírni.

```
1  #include <cstdio>
2  using namespace std;
3
4  int n, k;
5  long long elems[1000000];
6  int indices[1000000];
7
8  int main()
9  {
10     scanf("%d%d",&n,&k);
11     for(int i=0; i<n; ++i) scanf("%lld",&elems[i]);
12     for(int i=0; i<k; ++i) scanf("%d",&indices[i]);
13
14     for(int i=0; i<k; ++i)
15     {
16         printf("%lld␣",elems[indices[i]-1]);
17     }
18     printf("\n");
19     return 0;
20 }
```

0.4.2. „Tükrözés” C-ben

Megoldható C-ben is, de sztringek kezeléséhez célszerűbb a C++ eszköztárát bevetni.

A standard adatfolyamok kezelése C++-ban

0.4.3. „Kiválasztás” C++-ban

```
1  #include <iostream>
2  using namespace std;
3
4  int n, k;
5  long long elems[1000000];
6  int indices[1000000];
7
8  int main()
9  {
10     cin >> n >> k;
11     for(int i=0; i<n; ++i) cin >> elems[i];
12     for(int i=0; i<k; ++i) cin >> indices[i];
13
14     for(int i=0; i<k; ++i)
15     {
16         cout << elems[indices[i]-1] << "␣";
17     }
18     cout << endl;
19     return 0;
20 }
```

0.4.4. „Tükrözés” C++-ban

A Pascallal ellentétben C++ nyelven a sztringek beolvasása is a következő whitespace karakterig történik, ami szóköz, tabulátor, újsor bármelyike lehet. Ha egy teljes sort akarunk beolvasni,

használhatjuk a `getline` függvényt, itt viszont vigyázzunk, hogy a szokásos `>>` operátoros beolvasás az újsor jelet nem veszi le a bemenetről, így először egy üressztringet fogunk olvasni.

```
1  #include <iostream>      // cin/cout-hoz
2  #include <string>        // std::string-hez
3  using namespace std;
4
5  //#define SPLIT_AT_WHITESPACE
6
7  string rev(string s)
8  {
9      for(int i=0; i<s.size()/2; ++i)
10     {
11         swap(s[i], s[s.size()-i-1]);
12     }
13     return s;
14 }
15
16 int main()
17 {
18     int n;
19     string s;
20
21     cin >> n;
22
23 #ifdef SPLIT_AT_WHITESPACE
24     for(int i=0; i<n; ++i)
25     {
26         cin >> s;
27         cout << rev(s) << endl;
28     }
29 #else
30     getline(cin,s); // elso sor sorvege kiolvasasa
31     for(int i=0; i<n; ++i)
32     {
33         getline(cin,s);
34         cout << rev(s) << endl;
35     }
36 #endif
37 }
```

0.5. Mit ne csináljunk?

0.5.1. Ne ellenőrizzük a bemenet helyességét!

Soha ne ellenőrizzük, hogy a bemenet helyes-e, azaz a bemeneti specifikációnak megfelelő-e! Ez egyrészt felesleges, másrészt rengeteg hibalehetőséget teremt. A feladatkiírást egyfajta szerződésként célszerű felfogni: e szerződés szerint a programnak azt kell vállalnia, hogy amennyiben a bemeneti specifikációnak megfelelő bemenetet kap, arra a kimeneti specifikációban leírt formátumú (és tartalmú) kimenettel válaszol. Ha a felhasználó nem tartja be a „szerződésben vállalt kötelezettségeit”, akkor szerződésszegést követ el, és így cserébe a program is mentesül minden szerződésbeli kötelezettsége alól – akármit csinálhat.

Meg kell jegyezni, hogy ipari felhasználású programokban ez a megközelítés nyilvánvalóan nem működhet. Ott viszont a program specifikációjának explicit része, hogy a bemenet helyességét ellenőrizni kell. Ez egy rettentő unalmas munka, ami elvonja az időt az érdemi feladatmegoldástól. Ezért a verseny keretein belül szándékosan csak olyan teszteseteken értékeljük a program futását, amelyek a feladatkiírásnak eleget tesznek.

Hasonlóan nem kell ellenőrizni pl. azt sem, hogy sikerült-e a dinamikus memóriafoglalás.

0.5.2. Ne írjunk ki olyat, ami nem szerepel a kimeneti specifikációban!

A tesztelés során a program bemenetét nem kézzel fogjuk beírni, hanem egy fájl lesz a standard bemenetre irányítva. Ennek fényében semmi értelme olyan jellegű szövegeket kiírni, hogy „Írja be N értékét”, stb, hiszen úgysem olvassa senki, a kiírásoktól függetlenül a bemenet a feladat szövege által előírt formátumban lesz. Ha ezt nem támogatja a program, akkor a kiírások sem segítenek, ha pedig igen, akkor feleslegesek. Sőt! Nemhogy nem segítenek, hanem mivel ezek a kimeneti fájlban meg fognak jelenni, megnehezítik annak összevetését a várt eredménnyel, így ne tegyük!

Hasonlóan, mielőtt beadjuk a programot, legalább kommentezzük ki a hibakereséshez használt extra kiírásokat!

0.5.3. Ne várjunk a futás végén új sort, ne használjuk a pause rendszerhívást!

A megoldások tesztelését igyekszünk minél inkább automatizálni. A tesztelő keretrendszer azt várja, hogy miután a program a tesztesethez tartozó teljes bemenetet beolvasta, elkezd dolgozni, és némi számolgatás után kiírja a kimenetet, és kilép. Ha az előírt bemeneten túl még további új sorokra, karakter leütésére vár, vagy meghívta a `pause` rendszerhívást, akkor az automatizált tesztelés elakad.