

## S. 42.

Kővágó Zoltán  
9. o. Budapest  
Szent István Gimnázium

# 1. Fordító

A programot az alábbi fordítókkal fordítottam le:

- g++ 4.3.3, Gentoo Linux x86\_64 (AMD64)
- g++ 4.2.4, Gentoo Linux x86\_64 (AMD64)
- g++ 4.1.2, Gentoo Linux x86\_64 (AMD64)
- g++ 4.3.2, Cross compiler Windowsra, Gentoo Linux x86\_64-en, wine-al tesztelve

Ha a gcc jól van feltelepítve, egy terminálban elég ennyit beírni a program lefordításához:

```
$ cd cpp_fajl/konyvtara  
$ g++ -o s42 s42.cpp
```

Ha nem történ semmi probléma a fordítás során, az alábbi paranccsal lehet futtatni: (bár valószínűleg inkább a gazdaprogramnak fogjuk megadni a futtatható fájlt)

```
$ ./s42
```

Természetesen a terminál használatán kívül bármelyik IDE használható, mely a gcc-t használja a fájlok fordításához.

# 2. A programról

Először a lényegről szeretnék néhány szót ejteni. Abból indultam ki, hogy az alakzatokat úgy kell lerakni, hogy ezáltal egy olyan négyzet se legyen, ahol nincs semmilyen alakzat, de fölötte van valami, azaz ne maradjunk lyukak. Másrészt nem árt, ha úgy helyezzük el az alakzatokat, hogy a lehető leglejjebb, s így egyenletes növekedést biztosítva a toronynak. Miután kész voltam a programmal, és egy saját készítésű gazdaprogramban teszteltem, akkor tudatosult, hogy a lehető legjobb helyre rakjuk nem túl jó megoldás, mivel kis pályákon ( $W < 10$ ) olyan hatást tapasztaltam, hogy 2 egymás melletti oszlop igen magasra nő, a két szélén pedig egy olyan oszlop, ami sokkal alacsonyabb. Így aztán kompromisszumot kellett kötni, hogy az elem ne kerüljön jóval magasabbra, akkor se, ha ez azt jelenti, hogy több ilyen lyuk keletkezik.

Úgy döntöttem hogy egy általánosított helyett több, egy-egy alakzatra speciálisan megírt függvényt alkalmazok. Azért döntöttem így, mert véleményem

szerint egy olyan általánosított eljárás, amely az összes alakzatot lekezezi nagyon bonyolult lett volna, vagy rengeteg fölösleges számítási műveletet végzett volna el.

Mivel ha egy üres hely fölött már van valami, akkor az lyuk lesz, oda már semmilyen alakzatot se lehet betenni, így azt nem is fontos tudni, hogy ott van-e vagy sem; így én az oszlopoknak csak a magasságát tárolom el, ezzel memóriát megtakarítva; sőt szerintem még az algoritmus is egyszerűbb, mintha a komplett játékkeret eltárolnánk.

Az 'o', 'j', 'l', 's' és 'z' hasonló eljárást alkalmaznak. Megnézik, hogy a különböző pozíciókban milyen magasra kerülnének, s mennyi négyzet maradna használhatatlan, s ez alapján döntenek. Az 'o'-nál nem kell forgatással foglalkozni, mivel akárhogy forgatjuk, önmagát kapjuk, 'j'-nél és 'l'-nél 2 féle forgatást kell megnézni, mivel ha  $180^\circ$ -al forgatjuk el őket, ugyan azt kapjuk, mintha nem forgattuk volna el. Ezen kívül mivel egymás tükörképei, egy eljárás kezeli le mindkettőt, 's' és 'z' hasonló helyzetben van, azzal a kivétellel hogy itt 4 féle forgatást kell megnézni.

Az 'i' eljárása viszont teljesen más, először azt nézi meg, hogy fekvő állapotban el lehet-e helyezni tökéletesen. Ha nem, álló helyzetben fogja elhelyezni. Mivel így egyetlen négyzet sem marad használhatatlan, így állónál azt nézi mennyire fog kilógni a környezetéből. Ezek után ha van fekvő, és az nem kerülne jóval feljebb, mint az álló, akkor fekvő lesz, különben álló.