Döntő **2021. február 19.**

Okos otthon szimuláció

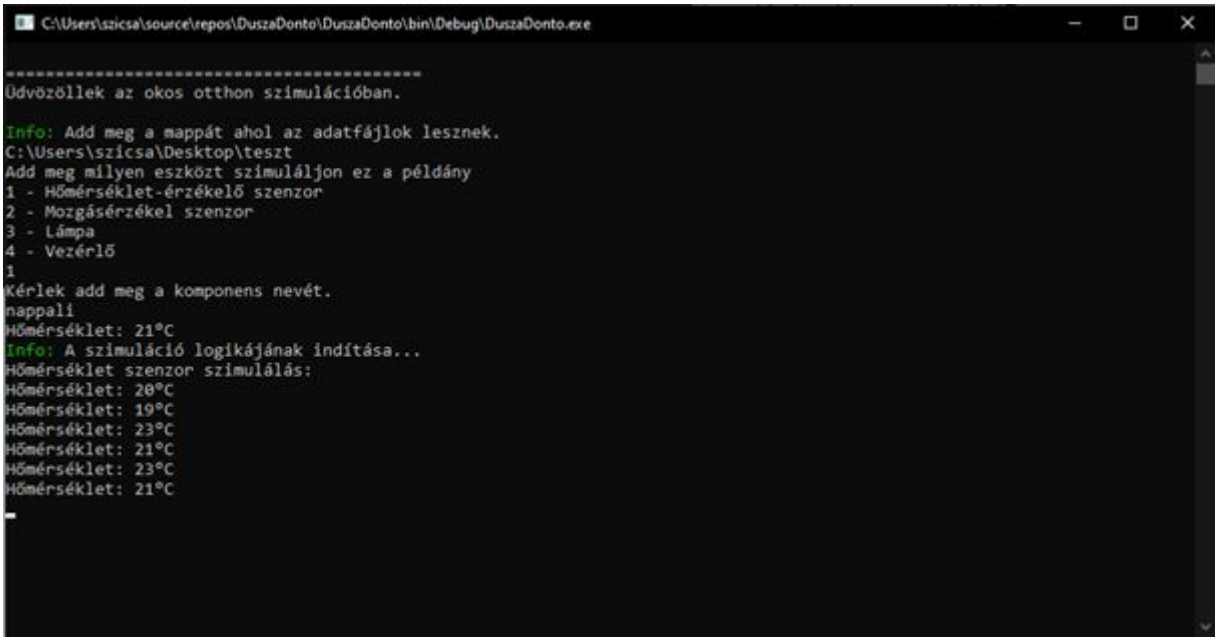
Manapság egyre elérhetőbbek az okos otthon megoldások. Ezeknek a rendszereknek az a lényege, hogy különböző céleszközöket helyezhetsz el az otthonodban (pl.: hőmérséklet-érzékelő, mozgásérzékelő, lámpa, izzó stb.), melyeket távolról irányíthatsz egy vezérlő alkalmazás segítségével. A feladatotok egy ilyen rendszer szimulációjának a leprogramozása.

A rendszer megvalósítása

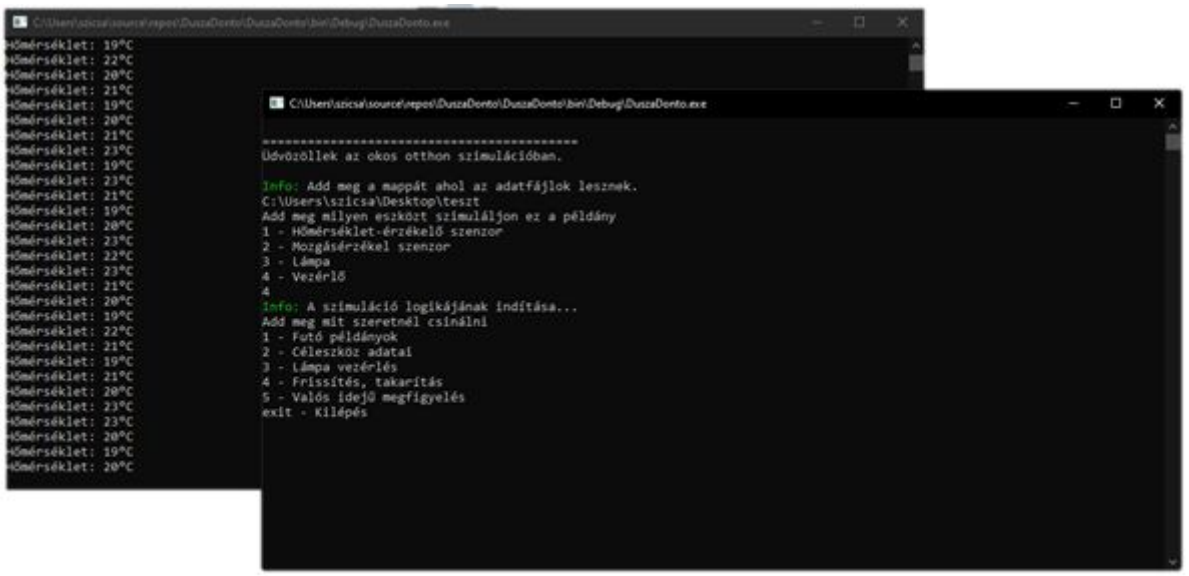
Jelen környezetben természetesen nem kell valós eszközökkel dolgoznotok, sem pedig a hálózati kommunikációt megvalósítanotok. Az okos otthon elemei mind-mind egy alkalmazás példányai, melyek fájlokon keresztül kommunikálhatnak egymással. A példányoknak természetesen kijelölt feladataik vannak. Mivel a rendszert a lehető legdinamikusabban kell megoldani, az Okos otthon szimuláció alkalmazásunknak 2 fontos módja kell hogy legyen:

1. Céleszközt szimuláló program
    a. A konkrét feladata (hőmérsékletet mér, lámpát kapcsol stb.) indítás után beállítható
2. Vezérlő alkalmazás, mely a céleszközökkel tud kommunikálni

A rendszert úgy kelthetjük életre, ha a fentebb leírt alkalmazásból több példányt indítunk el, vagyis egyetlen futtatható állományt futtatunk többször egyidőben. Például elindítunk 4-et úgy, hogy az egyik hőmérsékletet mérjen, a másik egy lámpa, a harmadik egy mozgásérzékelő, a negyedik pedig egy vezérlő legyen. Az alábbi képen éppen egy hőmérséklet példány fut.



A következő képen pedig az látható, hogy két példány fut párhuzamosan különböző módokban:



A verseny során tehát egyetlen programot kell megírnotok, amit a felhasználó párhuzamosan többször is elindíthat, de a bemenettől függően különböző „módokon tud viselkedni”.

A kommunikáció, adatfolyam

Valós környezetben a kommunikáció az alkalmazások között sokféleképpen történhet (például közvetlen hívásokon keresztül, mint amikor ellátogatott egy weboldalra egy URL segítségével). A mi megoldásunk egy adatbázison keresztüli kommunikációra hasonlít a leginkább. Mivel nem áll módunkban most tényleges adatbázist használni, fájlokat fogunk kijelölni erre a célra, melyek megadott helyen, megadott névvel és pontos adatstruktúrával rendelkeznek. A következő fájlokra lesz szükség:

1. Egy központi fájl, mely a futó alkalmazásokat és feladatukat tartalmazza (hőmérséklet-érzékelő, mozgásérzékelő, lámpa.)
2. Minden egyes céleszköznek egy saját fájl, mely tartalmazza a rá vonatkozó adatokat. Amennyiben ő egy érzékelő, neki írnia kell bele a generált adatait, amiből a központi alkalmazás olvas, de ha ő egy lámpa, akkor a központi alkalmazás írja bele az utasításait, amit a céleszköz-alkalmazásnak ki kell olvasnia.

A programok között tehát direktben semmiféle kapcsolat vagy közvetlen vezérlés nincsen. Nem kell az egyik folyamatnak a másikat elindítania, leállítani, adatot küldenie. Minden kapcsolat közvetett, konkrétan fájlokon keresztüli.

Mint a fentebb leírtakból az látható, a feladat egyik legfontosabb megoldandó problémája, hogy egy fájlt egyszerre több folyamat is olvashat (a központi fájlt például pontosan annyi, ahány alkalmazás fut), ami természetesen szigorúan tilos, sőt lehetetlen, mert az operációs rendszer nem enged egy nyitott fájlt egy másik folyamat által újra megnyitni. Előtte le kell zárni.

Párhuzamos fájlkezelés

A probléma megoldásának kulcsa, hogy „zároltnak” kell jelölnünk egy fájlt mindaddig, ameddig valaki dolgozik vele (angolul *lock*-olás). Ez a gyakorlatban annyit jelent, hogy ha egy folyamat éppen ír pl. az *eszkozok.txt* fájlba, akkor először létrehozza az *eszkozok.txt.lock* nevezetű üres fájlt.

A fájl puszta megléte jelöli a többi folyamat számára, hogy valaki éppen dolgozik vele. A fájlok írása és olvasása tehát 4 lépcsős kell hogy legyen:

1. Ellenőrzi, hogy a fájl lockolt-e (létezik-e a *fajlNév.lock* fájl).
2. Ha nem létezik, akkor létrehozza. Ha létezik, akkor vár a sorára, amíg el nem tűnik a fájl.
3. Ír, olvas.
4. Törli a *fajlNév.lock*-ot.

Az egész feladat során, minden egyes fájlkezelési műveletnek ezt az egységes logikát kell követnie.

A program általános működése

A programot tehát úgy kell megírni, hogy induláskor ki lehessen választani, mi lesz annak a konkrét példánynak a feladata. A következő lehetséges opciókat kell megvalósítani:

1. A program egy hőmérséklet-érzékelő berendezés szimulációja
2. A program egy mozgásérzékelő berendezés szimulációja
3. A program egy lámpa
4. A program egy vezérlő alkalmazás

Ezen felül induláskor ki lehet választani egy elérési utat, ahova a szükséges fájlok mentve lesznek (természetesen ellenőrizni kell, hogy a megadott útvonal létezik-e).

Céleszközt szimuláló mód

Az első 3 opció tartozik a *céleszközt szimuláló* kategóriába. Az ilyen példányoknak lehessen megadni egy megnevezést, mely arra utal, hogy a házban hol helyezkedik el az eszköz. A megnevezés természetesen csak egy, a felhasználó által megadott szöveg, amit nem tudunk ellenőrizni, hiszen túl sok minden elképzelhető (elméletileg tehát pl. *alma* is megadható), de olyan adatokat várunk ide, mint például: hálószoba, nappali, garázs stb.

Ezekben a módokban az alkalmazás létrehozza a neki megfelelő fájlt *elnevezes.mód.azonosit.txt* formátumban, ahol az elnevezés a korábban leírt megadott szöveg, a mód rendre *homerseklet, mozgas, lampa* értékek valamelyike, az azonosító pedig egy egész szám, mely az aktuálisan létező legnagyobb azonosítóhoz képest a rákövetkező szám. Egy komponens létezését elsősorban ennek a fájlnak a megléte definiálja. Amennyiben ez eltűnik, állítsa le magát (később látni fogjuk, hogy egy vezérlő példány kitörölheti), kivéve ha lámpáról van szó, hiszen az csak vár az utasításokra.

A program továbbá létrehoz egy bejegyzést az *eszkozok.txt* fájlban is, mely az aktuálisan futó alkalmazások listáját tartalmazza. A formátuma a következő:

```
AZONOSÍTÓ;DÁTUM;ELNEVEZES;MÓD
```

Egy lehetséges fájl tehát így nézhet ki:

```
1;2021.02.04 10.25.50;hálószoba;homerseklet
2;2021.02.04 10.26.50;nappali;mozgas
3;2021.02.04 10.27.50;nappali;lampa
```

Hőmérséklet-érzékelő

Ebben a módban az alkalmazás 10 másodpercenként generál adatot. Mivel szimulációról van szó, a hőmérséklet adatok véletlenszámok. Egy ház adott területén a hőmérséklet viszonylag kicsi intervallumban mozog, ám hogy milyen érték körül, az változhat. Induláskor tehát véletlenszerűen ki kell választani 19 és 24 között egy véletlen egész számot (jelölje *n*). A későbbiekben az adatok a [n-2,n+2]-ba eső véletlen egész számok percenként. Az első „mért adat", vagyis az első bejegyzett hőmérséklet az *n* hőmérséklet legyen.

Sajnos a valóság nem annyira szép, mint a matematika. Ezek az eszközök gyakran hamis értéket mérnek valamilyen környezeti hatás miatt. A szimulációban az esetek 1%-ban az eszköz egy [-100,100]-ba eső véletlen egész számot ad vissza (melyet a későbbiekben a vezérlő alkalmazás megszűr majd). Ez meglehetősen ritkán következik be. A tesztelhetőség miatt az alkalmazás produkáljon *ENTER* lenyomás esetén 2 hibás adatot. Figyeljünk rá, hogy biztosan hibás legyen mind a kettő hőmérséklet, azaz nem elegendő a [-100,100] intervallumba esnie, de a „jónak" gondolt számok egyike sem lehet. A generált adatot a képernyőre, valamint fájlba is kell írni

Az adatfájl (pl.: haloszoba.homerseklet.1.txt) struktúrája a következő legyen

```
Dátum;hőmérséklet
```

A dátum ÉÉÉÉ.HH.NN ÓÓ.PP.MM formátumban legyen. Egy lehetséges fájlt tehát így nézhet ki:

```
2021.02.04 10.25.50;21
2021.02.04 10.26.00;20
2021.02.04 10.26.10;19
2021.02.04 10.26.20;22
```

Mozgás-érzékelő

Ebben a módban az alkalmazás 3 másodpercenként generál adatot. A mozgás valójában egy igaz-hamis érték (igaz, ha van mozgás). Véletlenre tenni a mozgást életszerűtlen lenne, így az *igaz* érték valószínűségét definiáljuk: az esetek 10%-ban van mozgás. Bár ennél az érzékelőnél is lehetnek hibák, azok szimulálása nem annyira egyszerű feladat, így ettől eltekintünk. Az adatok a képernyőre, valamint fájlba is kell írni.

A fájl struktúrája kövesse a fentit, így egy fájl így nézhet ki például:

```
2021.02.04 10.25.50;igaz
2021.02.04 10.25.53;hamis
2021.02.04 10.25.56;hamis
2021.02.04 10.25.59;hamis
```

Lámpa

Egy lámpa alkalmazás nem generál adatot, viszont lehet vezérelni. Három utasítást tud befogadni:

1. BE
2. KI
3. KOVET;<*azonosító*>

A lámpa fájlja tehát egyetlen sor, mely tartalmazza az utasítást (amit egy vezérlő példányból lehet kiadni). Miután érzékeli az új utasítást, a programnak fel kell tölteni ezt a sort egy időbélyeggel, hogy jelezze, mikor történt a feldolgozás.

A fel nem dolgozott fájl egyetlen sora tehát úgy nézhet ki például, hogy:

```
BE
```

Feldolgozás után pedig:

```
2021.02.04 10.25.50;BE
```

A „BE” és „KI” utasítást értelemszerűen azt jelenti, hogy a lámpát fel vagy le kell kapcsolni. Amennyiben ugyanolyan utasítást kap, mint amilyen állapotban van éppen (például BE, amikor már be van kapcsolva), ne csináljon semmit.

A „KOVET;<*azonosító*>” utasítás esetében a lámpa képes egy megjelölt mozgásérzékelőt követni (az <*azonosító*> tehát egy mozgásérzékelő azonosítóját jelöli). A követés azt jelenti, hogy a lámpa felkapcsol, amennyiben van mozgás, majd lekapcsol, amennyiben nincs mozgás. Ez folyamatosan így legyen, amíg a lámpa újabb utasítást nem kap, vagy a követett érzékelő le nem áll.

Mivel nem tudunk konkrét (valós) lámpát felkapcsolni, ezért elegendő az alkalmazásban jelölni valahogy a bekapcsolt, illetve a kikapcsolt állapotot. A konkrét megvalósítás a csapatokra van bízva, hisz ez nyelv- és környezetspecifikus is. Elfogadható egy kiírás, egy karakteres kirajzolás, de konkrét kép megjelenése is.

Vezérlő mód

A vezérlő példányok nem írják magukat az *eszkozok.txt* fájlba (azaz több vezérlő is futhat). Fő célja a futó alkalmazások monitorozása és vezérlése (szigorúan csak fájlokon keresztül), a generált adatok tárolása és kezelése. A felhasználó számára a következő lehetőségeket kell biztosítania:

1. Képes kiírni az aktuálisan futó példányok adatait (mikor indították el, mi az elnevezése és milyen típusú).
2. Minden olyan futó céleszköz által generált adatokat le lehet kérni, amelyek képesek rá (tehát egy lámpáét nem lehet).
3. Ha fut lámpa, akkor azoknak felkínálja az utasításokat, amiket ki is lehet adni.
4. Lehet frissíteni az eszközök listáját. Ez esetben leállítani is képes példányt (persze nem közvetlenül, hanem csak a fájlját törli ki).
5. Van opció az összes (a lámpát is) céleszköz adatainak valósidejű megfigyelésére, mely gombnyomásra ér véget.

Fontos megjegyezni, hogy ha több példány is fut akár egy kategóriából, akkor is ki lehet jelölni pontosan, melyik példánytól lehet olvasni/melyiket lehet vezérelni. Ha például több lámpa van, akkor mindegyiket egyesével külön lehessen vezérelni.

Törlés és mentés

A 4-es opció esetén a vezérlő alkalmazás takarítást is végez. Ez azt jelenti, hogy ha egy példány hibásan megy, vagy leállt, akkor kiveszi a rendszerből. A leállást onnan tudjuk, hogy nem érkezett adat, amikor kellett volna, vagy nem hajtotta végre az utasítást, amikor kellett volna. Mivel a lámpa esetében nem érkezik folyamatosan adat (még a folyamatos követésnél sem frissül a fájl),

ezért ott csak úgy tudunk ellenőrizni, ha küldünk egy parancsot, s megfigyeljük azt végrehajtotta-e a lámpa, vagyis frissült ez az időbélyeg. Hibás működésről pedig a hőmérséklet-érzékelésnél beszélhetünk. Ha kettő egymást követő esetben hibás hőmérséklet-adat jött, akkor törölni kell.

A törlés azt jelenti, hogy a bejegyzés eltűnik az *eszkozok.txt* fájlból, illetve a saját adatfájlja is törlődik. A példányok erre reagálnak, azaz ha nem találják a saját fájljukat, akkor leállítják magukat.

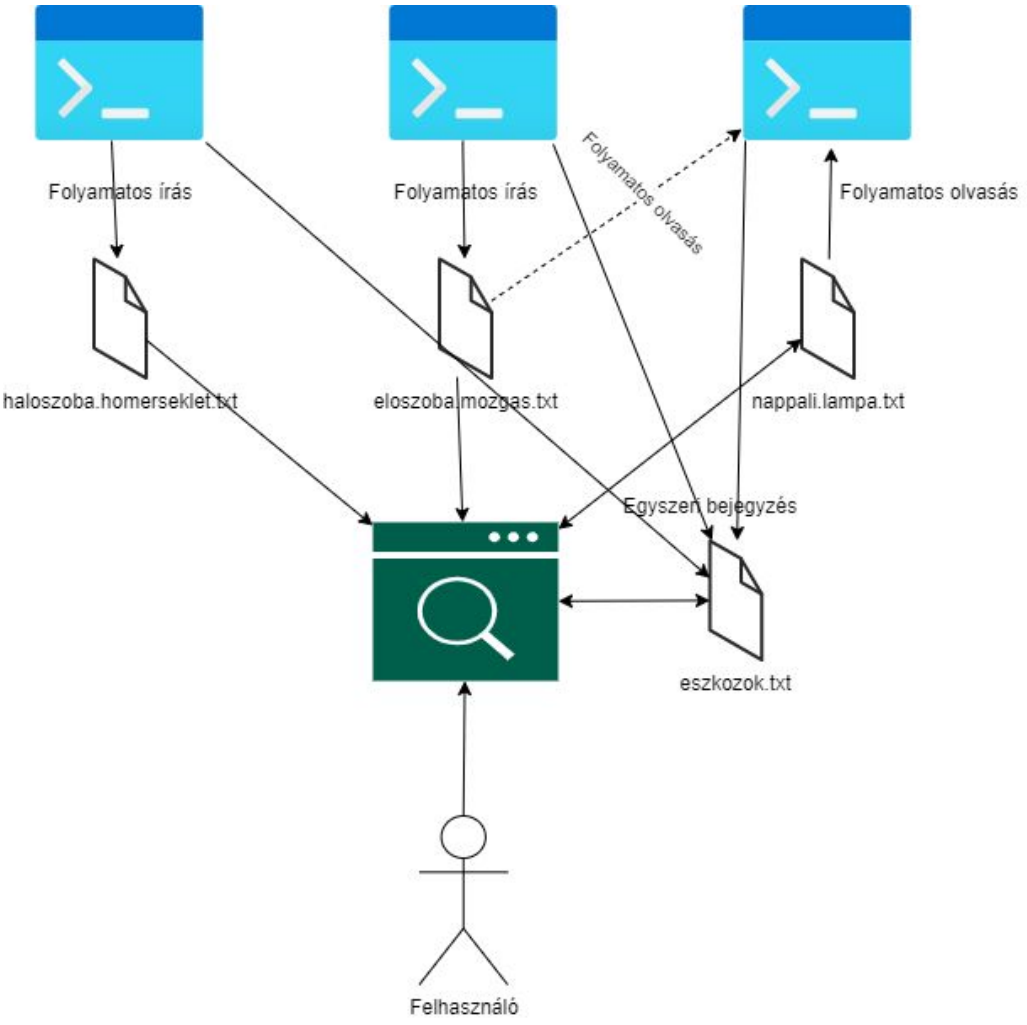
Összefoglalás

A feladat tehát **egyetlen program megírása**, mely különböző módokon tud futni (tehát nem külön-külön futtatható alkalmazás az okos otthon komponenseinek). A program két bemenetet vár el:

1. A mód
2. A mappa helye, ahol dolgozni fog

A megírt programot a felhasználó többször is elindíthatja párhuzamosan. A rendszer úgy fog életre kelni, ha a felhasználó minél több példányt indít el az alkalmazásunkból.

A teljes architektúra tehát a következőképpen néz ki:



A feladat során számtalanszor felmerülhet, hogy 1-1 adat esetleg nem felel meg a valóságnak, értelmetlen vagy használhatatlan. Például a felhasználó megad egy azonosítót a lámpának (hogy azt kövesse), de olyan azonosító nem létezik, vagy nem egy mozgásérzékelő példányé. Minden ilyen esetben ellenőrizni kell a bemeneti adatot.

Beadandó

A program **forráskódja** (a programozási környezettől függően a forráskód több fájl is lehet, esetleg mappák is tartozhatnak hozzá) és a **lefordított fájl** (okosotthon.exe), ha a programozási környezet a fordítást támogatja.

A döntő menetéről

A döntő során nem szükséges dokumentációt készíteni a programhoz (kommentezni továbbra is kell), azonban szükség van prezentáció készítésére, melyet a következő napon kell bemutatni. A prezentációt nem kell leadni a verseny ideje alatt.

A prezentálás

Mivel a prezentáció is online formában lesz, ezért vegyétek figyelembe a következő irányelveket:

1. A csapatok függetlenül attól, hogy egy helyen vannak-e, egyetlen képernyőt kell, hogy megosszanak.
2. A megosztó fél végezze a navigálást (a futó programban, kódban, prezentáció diáiban stb.)!
3. A csapat minden tagja szólaljon fel legalább egyszer!
4. A prezentáció elején röviden mutatkozzatok be!
5. 15 perc jut maximum egy csapatra.

Kódolási alapelvek

A forráskód minőségét is értékeljük.

Irányelvek, szempontok:

- Egységes kódolási szabályok az azonosítókra:
  - a változók egységes elnevezése (kis- és nagybetűk vagy más speciális karakterek használata)
  - az osztályok egységes elnevezése (objektum-orientált programnyelv esetén)
  - a függvények és eljárások tartalomra utaló elnevezése
  - a programkód egységes strukturáltsága, tagoltsága
- A kód minősége (könnyen – emberek számára – érthető illetve karbantartható kód):
  - áttekinthető, lehetőség szerint rövid eljárások, függvények, fájlok
  - beszédes, tömör elnevezésű azonosítók
  - objektum-orientált nyelveknél globális változók mellőzése
- Törekedjetek arra, hogy a kódotok minél hatékonyabb legyen!
- Kommentezés:
  - A kommentezés elsődleges célja, hogy a programban a miért? kérdésre adjon választ. (A mit? kérdésre az azonosítók megfelelő elnevezése és a megfelelően strukturált kód, a hogyan? kérdésre pedig az áttekinthető forráskód ad választ.)

- A túlzásba vitt kommentezés csökkenti az áttekinthetőséget, a túl kevés komment nehezíti a megértést.
- Elvárás a változók, osztályok, függvények és eljárások szerepének rövid, értelemszerű kommentezése.

Elérhető pontszám: 150 pont (Ebből a helyes kommentezés/kódminőség – 15 pont; szóbeli bemutatás – 30 pont)

Jó munkát kíván a versenybizottság!