Tartalom

Felhasználói dokumentáció

Téma

Témának a Schönherz kollégium igazgatását választottam. A programban a kollégium lakóit költöztetni lehet, kirúgni, büntetőpontokat gyűjteni és egyéb igazgatási tevékenységet végezni. A QPA-nál megvan az esély, hogy történik valami a kollégium körül, amivel járhat büntetőpont, vagy költözés is az adott személynek. Lehetséges, hogy bizonyos adatai változnak ilyenkor, például, ha Schönherz QPA ideje jön el, akkor a sörmérések átlaga javulhat.

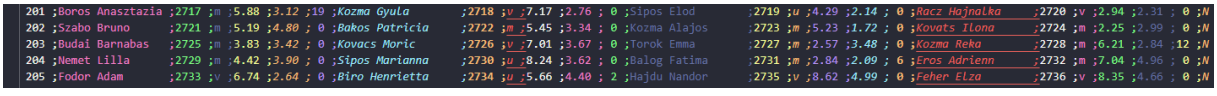
A program megkap egy adatbázist, amiben szerepelnek 1-18-ig a szintek, ahol minden szinten 16 szoba van. Ezekben a szobákban négy ember lakik. (Ha van üres hely arra a kódban külön jelölés van).

Beolvasás/Kiírás

Az adatok egy generált txt fájlból vannak beolvasva, minden sora egy szobának az adatait tartalmazza pontosvesszővel tagolva. Ez azt jelenti, hogy rendre az adatok:

- **Szobaszám (101-1816)**: 3 vagy 4 számjegyű (pl.:911, 1110) Ez teljesen megegyezik a kollégium szobaszámozásával.
- **Lakók** (4 db): Beolvasás után minden hallgató rendelkezik majd egy
  - **Név** (Max 25 karakteres teljes név): Minden név string-ben van tárolva.
  - **Kártyaszám** (Egész szám 1000 és 9999 között): Ezek egyedi azonosítók, minden ember kap egyet, általában növekvő sorrendben vannak kiosztva. A programban egészként van tárolva
  - **Szak** (egy betű): Lehet **m→ Mérnökinformatikus, v→ Villamosmérnök, u → Üzemmérnök informatikus.** A programban ez egy karakterként van tárolva
  - **Sörmérés** (Mp): Egy double változóban eltárolt, kettő tizedesjegy pontosságú változó. Egy ember sörmérését írja le.
  - **Tanulmányi átlag** (1-5): Egy double változóban eltárolt, kettő tizedesjegy pontosságú változó. A hallgató előző féléves átlagát mutatja.
  - **Büntetőpont** (0-20): Egészként van tárolva a programban. Egy bizonyos szám felett kirúgást eredményez. (Valószínűleg beolvasáskor és büntetőpont osztásakor nézi csak meg a program, hogy van-e olyan, akinek összegyűlt.)
- **Gólyaszoba (N vagy I)**: Ez jelzi, hogy az adott szobában gólyák laknak-e: **Nem → N**, **Igen → I** jelölést kapta. A programban egy **bool adat-ként van tárolva.**

Példa a fájlban tárolt adatokra:



Kimenet/Eredmény

A program parancssorban működik, itt lehet különböző parancsokat megadni, például elköltöztetni egy lakót, vagy megnézni, hogy melyik szinten milyen lakók vannak.

A program tudja módosítani a beolvasott adatbázist a különböző műveletekkel. A program pontosan abban a formátumban írja ki az adatokat, ahogyan beolvassa. Minden bezárással elmenti a változtatásokat.

Program használata

Mivel az adatbázist tönkre tudja tenni, ha rossz adatokat ad meg egy felhasználó, ezért mindig figyelmezteti a program a felhasználót, ha nem megfelelő adatot adott meg (például, ha megpróbál megkeresni egy szobát és szobaszámnak „asd”-ot ír). Ezek a hibalehetőségek kezelve vannak.

Érdemes az adatbázist Excel-ben megnyitni, ott strukturáltabb átláthatóbb és támogatja is a megnyitását, viszont fontos, hogy nem mentsük el ott a fájlt, mert az elronthatja a struktúráját.

File megnyitása

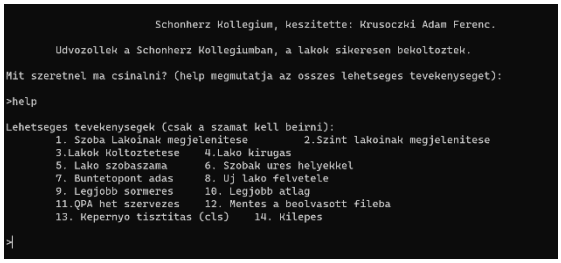
Az adatfile megnyitására kettő lehetőség van:

- Command line argument-ként megadjuk a fájl nevét. Itt először viszont fontos megadni, a „debug” parancsszót elé (ez a debugmalloc dump-jait egy log fájlba vezeti).
- Ha nem kapja meg az előző módon, akkor lehetőségünk van megadni a program elindításakor.

Fontos, hogy az adatfájl a programmal egy mappában legyen! Ha a program, jelezte, hogy a lakók sikeresen beköltöztek, akkor az adatok beolvasása és feldolgozása sikeresen megtörtént.

Menü kezelése

Ahhoz, hogy tudjunk tevékenységeket futtatni, meg kell adni a menüpont számát. Ha szeretnénk, hogy a program kiírja nekünk a menüpontokat, írjuk be a parancssorba, hogy „help”. Az itt felsorolt tevékenységeket lehet végezni a programban. Erre példaként szolgál az jobbra lévő kép.

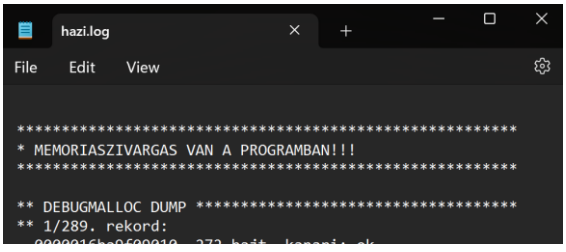


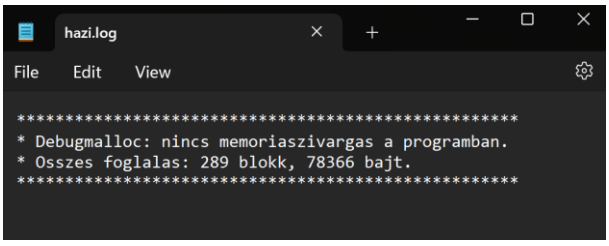
A szám megadása után a program, ha szükség van rá, felveszi az adatokat, és elvégzi a kért parancsot.

Mivel a program több header fájlból áll, ezért a fordítást és a megnyitást megkönnyíti, ha egy „.bat” kiterjesztésű fájlba az alábbi szkriptet bemásolja a felhasználó (az adat.csv helyett a saját fájlját megadva):

```
@echo off
gcc main.c ./headerek/adatkezeles.c ./headerek/tevekenysegek.c -Wall -Werror -
o futas
.\futas.exe debug_log adat.csv
echo A kilepeshez nyomj meg egy gombot...
pause >nul
```

és ezt lefuttatva a program elindul és létre is hoz egy hazi.log fájlt amiben a debugmalloc dump-ját megtaláljuk. Ezt az alábbi képek illusztrálják.





Parancsok

A parancsokat a menüpontból tudjuk kiválasztani, ebben a részben azokat fogom ismertetni.

1. Szoba lakóinak megjelenítése

Ezzel a paranccsal meg tudjuk nézni, hogy az általunk választott szobában kik laknak, a szakjukat, büntetőpontjukat, és a kártyaszámukat is.

2. Szint megjelenítése

Ugyanabban a megjelenítésben meg tudjuk nézni, kik laknak egy általunk megadott szinten.

3. Lakók költöztetése

Ezt a parancsot választva kettő kártyaszámot megadva költöztethetünk lakókat. Ha egy üres helyre szeretnénk költöztetni, akkor is az üres helyhez tartozó kártyaszámot kell megadnunk, hiszen ezt a szobával adják át a lakónak.

4. Lakó kirúgása

Ha egy lakót el szeretnénk távolítani az adatbázisból, akkor ezzel a paranccsal a kártyaszámát megadva ki tudjuk rúgni a kollégiumból.

5. Lakó szobaszáma

Ezzel egy általunk megadott kártyaszámú lakos szobáját kaphatjuk meg, és azt, hogy hányadik sorszámú lakos abban a szobában.

6. Szobák üres helyekkel

Ez a parancs megadja nekünk azokat a szobákat, amelyek legalább egy üres hellyel rendelkeznek.

7. Büntetőpont adása

Egy általunk választott kártyaszámú embernek tetszőleges számú pozitív egész büntetőpontot adhatunk (20-tól kirúgják automatikusan a lakót).

8. Új lakó felvétele

Itt felvehetünk az adatbázisba egy új lakót. Meg kell adnunk az adatait, abban a formátumban, ahogy a program kéri. Ezután ad a program neki egy kártyaszámot (az előző lakóét).

9. Legjobb sörmérés

Ezzel a függvénnyel megkapjuk a kollégium legjobb sörmérőjét. Ezen javíthatunk, ha QPA-kat futtatunk le

10. Legjobb átlag

Ezzel a kollégium legjobb tanulmányi átlagával rendelkező lakót kapjuk meg.

11. QPA Hét

Ezzel egy QPA-hetet futtathatunk le, amivel véletlenszerű események járnak, mint az ablakon kizuhanó hűtő, sörmérések, tanulmányi átlagok romlása, vagy akár mikró robbantás is.

12. Manuális mentés

Ezzel menthetjük a beolvasott fájlba a módosításokat, ha szeretnénk. A bezáráskor automatikusan ment a program, de hosszabb időeltöltés után érdemesebb menteni manuálisan is.

13. Képernyő tisztítás (CLS)

Ez a parancs hasonlóan működik, mint a PowerShell-ben a „cls” függvény. Letisztítja a korábbi szövegeket a képernyőről.

14. Kilépés

Így tudunk megfelelően kilépni, más esetben nem ment automatikusan a program, és ha az ablakot zárjuk be, akkor nem biztosított, hogy a program megfelelően lefut és felszabadít minden lefoglalt memóriát.

Adatfájlok

Az adatok mind generálva vannak véletlenszerű nevekkel és adatokkal, viszont mind az első oldalon illusztrált struktúra alapján. A programhoz csatolva van több ilyen fájl is, mind más-más lakókkal, párban előre kirúgva embereket.

Amíg a szükséges elválasztás és a felépítés megvan addig a program gond nélkül fel tudja használni a fájlt (feltéve, hogy nem teljesen üres a fájl), akárhányszor is lett a **program által módosítva**. Viszont, ha saját kézzel módosítjuk (nem megtartva a struktúrát), akkor nem garantált, hogy jól fog működni.

Fejlesztői dokumentáció

Adatstruktúrák

A program két fő adatstruktúrája a lako_struct és a szoba_struct. A szoba_struct egy láncolt listát alkot és minden eleme egy beolvasott szobának felel meg. Ezen belül minden szobának van 4 darab lakója, amik 4 elemű tömbökként vannak a megfelelő szobákhoz rendelve. A programnak ez az egyetlen dinamikusan foglalt adatstruktúrája.

```
typedef struct lako_struct // * E
{
    char nev [NEVMAX]; // * Lako
    int kartyaszam; // *Schonherz
    szak szak; // * Enummal a szo
    double meres; // * Sormerese
    double atlag; // * Tanulmanyi
    int buntipont; // * A bunteto
} lako_struct;

typedef struct szoba_struct  // *
{
    lako_struct lakok[4]; // * Ez
    int szobaszam;
    bool golyaszoba; // * Ez megm
    struct szoba_struct * kov; //
} szoba_struct;
```

Ezen kívül három felsorolt típust tartalmaz a program. Az első az összes függvény visszatérési értékével rendelkező **errorok** nevű típus. Ez segít a hibák kezelésében *(például létezik sikeres, vagy nem_letezo_szoba visszatérési érték is)*.

A második felsorolt típus a **szak**. Ez a 3 szakot tartalmazza és az üres emberek szakját („-”). Ezeknek a száma a beolvasott szak karakterének ASCII kódja *(például a mérnökinformatikus → m → 77)*.

A harmadik felsorolt típus a **felhasznalo.** Mivel a programban a menüpontokat számokkal lehet kiválasztani, ezért minden lehetséges tevékenység kap a felsorolt típusban egy számot. Így a felhasználó kiválasztott menüpontját egy **switch**-ben fel lehet dolgozni.

A programban egy globális változó van, az **ures_lako.** Ez egy üres/kirúgott lakó lako_struct-ja. Ezt illeszti be a program, ha valakit például kirúgnak.

Főbb algoritmusok leírása

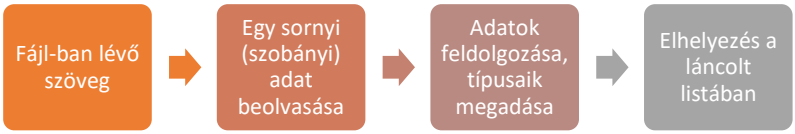
A program során legtöbb függvénynél megkapott „szoba_struct szoba_parameter” paraméter a fő láncolt lista fejét jelenti **mindenhol**. Ezért erre a legtöbb helyen nem térek ki, mindenhol egységesen ezt jelenti. **Visszatérési érték (a random_generator-on kívül) csak az errorok felsorolt típusból lehet.**

Adatkezelés Header File

Ebben a header fájlban vannak azok a függvények, amik az adatok kezeléséért, tárolásáért felelnek és a fő adattípusok definiálása is itt történik. A legfontosabb függvények:

Beolvasás (beolvas)

Az adatok beolvasásáért és eltárolásáért a **beolvas** függvény felel, ami megkapja a file nevét, és az adatstruktúra dupla pointerjét. ***A paraméterlistán visszatér a kész adatszerkezetekkel.*** Itt egy ciklusban beolvas egy sornyi adatot, ami egy szobának felel meg, és ezeknek elhelyezi a láncolt lista egyik elemében. Visszatérési értékei lehetnek a: ***sikeres, nem_letezo_file, nem_sikeres_memoriaf*** (ha nem sikerül memóriát foglalni)**.** A ciklus rövid diagrammja:



Kiírás (kiiras)

Az adatok kiírását a **kiiras** függvény végzi. Itt végigfut a láncolt listán, és ugyanabban a formátumban kiírja a szöveges fájlba az adatokat, egy cikluson keresztül. Ha menteni szeretnénk az adatokat akkor ez a függvény lesz meghívva. Megkapja a file nevét és a láncolt lista első elemének pointerjét. Nincs visszatérési értéke.

Felszabadítás (kuka)

A felszabadítást a „kuka" függvény végzi. A láncolt lista pointerjét megkapja és ezen végig futva felszabadítja egyes elemeit. Nincs visszatérési értéke.

Hibakezelés (error_kezeles)

Az error_kezeles függvény lényege, hogy a visszatérési értékekhez kapcsoljon megfelelő hibaüzenetet a felhasználónak. Majdnem mindegyik függvényben elő van hívva, ahol lehetséges, hogy hiba történne.

Tevékenységek Header File

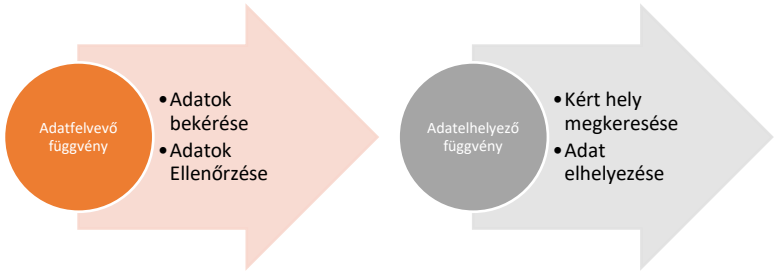
Ebben a header fájlban vannak azok a függvények, amik a beolvasott adattal foglalkoznak. A legtöbb függvény kártyaszámmal működik, ez az elsődleges kulcs. Itt több függvény is szerepel, de a főbb függvények:

Lakó megkeresése (hol_lakik)

Ezt a függvényt hívja segítségül a legtöbb függvény. Egy kártyaszámot kér és visszaadja a megfelelő szoba pointerjét, és a lakó sorszámát a szobában a **paraméterlistán keresztül**. Ez a függvény van használva akkor is, amikor egy adott lakóról szeretnénk megtudni, hogy létezik-e egyáltalán.

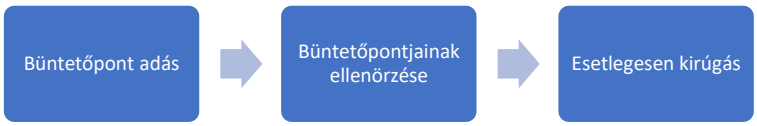
Új lakó felvétele (ujlako_adatfelvetel, ujlako_listaba_felvetel)

Ez kettő függvényből áll, az egyik felveszi a lakó adatait a felhasználótól, és ezeket átadja annak a függvénynek, ami ezeket elhelyezi a megfelelő szobában. Itt nagyon sok hibalehetőség van (nem megfelelő adatok megadása) ezért itt az adatok ellenőrzésé sok munkát igényel.



Büntetőpont kezelés és kirúgás (buntetopont_ellenorzes, buntetopont_adas, kirugas)

A büntetőpontok kezelése kettő függvényből áll. Az egyik büntetőpontot tud kiosztani, a másik ellenőrzi, hogy egy lakónak összegyűlt-e annyi büntetőpont, hogy kirúgják. Ha összegyűltek a büntetőpontok, a kirúgás függvényét felhasználva eltávolítja a lakót a kollégiumból. A kirúgás függvény egy adott kártyaszámú lakót távolít el a kollégiumból. Ezt használhatja egy függvény is, de akár a felhasználó is rúghat ki lakókat.



Költöztetés (koltoztetes)

A költöztetés függvény kettő lakó költöztetését, vagy egy lakó üres helyre költözését biztosítja. Ilyenkor a két szoba tömbjéből felcseréli a lakókat, a lakó megkeresése (hol_lakik) függvény segítségével. Ezt manuálisan tudja végezni a felhasználó.

Szoba/Szintkiírás (lako_print, szintprint)

A lako_print függvény kiírja egy szoba lakóit. Itt a „szinthez” paraméter jelzi, hogy a felhasználó, vagy a szintprint függvény hívta meg a jelenlegi függvényt. Ha szinthez lett meghívva, akkor nem írja ki a szobaszámot és az ahhoz tartozó szöveget.

A szintprint függvény csak egy adott szinthez tartozó szobákat felsorolva kiírja, más szövegezéssel.

QPA (qpa)

A QPA függvény együtt működik a random_generator függvénnyel. Itt generál egy random eseményt a qpa_esemenyek felsorolt típusból. Itt ezután generál véletlenszerűen lakókat, akikkel a random esemény történni fog. Itt azért, hogy a random esemény ne tudjon egy üres lakót generálni, több generálás is történhet, amíg egy megfelelő lakót nem találunk. Ha a hűtős esemény történik, akkor megvan (50%) az esély, hogy még egy hűtő ki fog esni.

Legjobb sörmérés/átlag (legjobb)

A „legjobb” függvény egyszerre tud legjobb sörmérést és átlagot kiírni. Itt a sörmérés paraméter, ha igaz akkor sörmérést keres, ha hamis, akkor tanulmányi átlagot fog keresni.

QPA felsorolt típus

Ez a felsorolt típus utólag került bele. Ez három eseményt tartalmaz, ezek közül lesz a függvényhívás esetén generálva egy.

Szabad Hely(szabad_hely)

Ez a függvény megnézi, hogy van-e szabad hely egy megkapott szobában. Visszatérésként adja meg, hogy talált-e szabad helyet, az errorok-ból.

Összes szabad hely

Az összes szabad helyet keresi meg és írja ki automatikusan. Ezt meghívja az új lakó felvételkor a „ujlako_adatfelvetel” függvény is.

Main Fájl

A main fájl foglalkozik a felhasználói interakciókkal. Három függvény van (és persze a main függvény):

Főmenü (fomenu)

Ez tartalmazza a menüt és a fő ciklust. Ez a függvény a **help** is parancsot szolgáltatja, ami kilistázza a lehetséges parancsokat. A benne lévő ciklus alatt fut a program és fogad el parancsokat. Ha megkap egy parancsot, azt továbbítja a függvény keresőnek

Függvény kereső (fuggveny_kereso)

Itt a megkapott parancsot egy switch-ben megkeresi, bekéri az adatokat (például szobaszám) és előhívja a megfelelő függvényt ehhez. Ennek a segítségére szolgál a **felhasznalo** felsorolt adattípus. A legtöbb adat ellenőrzése itt történik, ha nem igényel sok adatot a parancs.

Argumentum kezelés/Fájlnév olvasás (argumentum_kezeles, filenev_olvasas)

Mivel a fájlnév elsődleges megadási módja a command line argument-ekből van ezért itt van feldolgozva. Ha megadjuk a debug_log parancsot elsőnek, akkor egy log fájlba menteni fogja a

debugmalloc dump-jait. **Másodiknak a fájlnevet** kell megadni. Ha nem itt adjuk meg akkor a „filenev_olvasas” függvény bekéri a felhasználótól.

Kommentek

A kommentek színeket kapnak egy VSC extension-nel, aminek a neve Better Comments by)Aaron Bond)

```
// ! README !
// A kommentelesek egy kulso VsCode extensionnel szineket kapnak es atlathatobbak lesznek (Better Comments, Aaron Bond)
// ! Ezek a kommentek (pirosak) Fontos infot tartalmaznak, figyelmeztetesre vagy esetleges hibara utalnak.
// TODO Ezek a kommentek (narancs) teendoket jelolnek a kodban, altalaban befejezetlen fuggvenynel, vagy modositando dolognal szerepelnek.
// * Ezek a kommentek (zoldek) Adatok, strukturak, es lepesek jelolesere szolgalnak.
// ? Ezek a kommentek (kek) altalaban fuggvenyek / nagyobb lepesek vazlatos leirasa. Egy fuggvenynel vagy egy hosszabb ciklusnal talalhatoak altalaban.
```

Tesztdokumentáció

Tesztadatok

A programhoz csatolva van 10 darab teljesen új tesztadat és 3 darab olyan tesztadat, ami már módosítgatva van. Minden ilyen struktúrájú fájllal működik, eddig a txt és csv kiterjesztésű fájlokkal lett tesztelve. A lehetséges hibákra tesztelve lett, a felhasználói hibázásra és több fájlra is. A program tesztelve lett Windows 10/11 operációs rendszeren is, az egyetlen gond az lehet, hogy az színkódok nem működnek egyes gépeken. A tesztadatok mind véletlenszerűen vannak generálva, ezért teljesen különböznek az adatok.

Memóriaszivárgás

A debugmalloc dump-jaiban látható, hogy **nincs memóriaszivárgás** semmilyen esetben sem. A program minden bezárás előtt felszabadít minden memóriát. Ha nem a „Kilépés” paranccsal lép ki a felhasználó, akkor nem tud a program teljesen lefutni, és nem tudja felszabadítani és automatikusan menteni a változtatásokat, ezért az nem ajánlott.