NHF - Közösségi hálók

Neptun	GLJD1D
Név	Nemes Lili

Specifikáció

A program célja

A program célja közösségi hálókkal kapcsolatos szociológiai elméletek látványos modellezése. Az első szimuláció az információk terjedését, a második a közösség különböző

tagjainak a közösség egészéről alkotott képét, a harmadik pedig egy tulajdonság közösségen belüli terjedését mutatja be ugyanazon, a felhasználó által megadott kapcsolatrendszeren keresztül.

A program működése

Egy rövid, az egymásra épülő kísérletek tudományos jelentőségéről illetve a program használatáról szóló bevezető után a felhasználó feladata megadni egy közösségi hálót. Ez két fő részből áll:

1. A közösség tagjainak létrehozása, esetleg törlése (pl. e+ 1→ 1-es azonosítójú ember létrehozása vagy e- 2 → 2-es azonosítójú ember törlése)
2. A tagok közti kapcsolatok létrehozása, esetleg törlése. (k+ 1 2 → 1-es és 2-es azonosítójú emberek közt kapcsolat létrehozása, vagy k- 1 2 → 1-es és 2-es azonosítójú emberek közti kapcsolat törlése)

A kapcsolat két ember között kétirányú. A ismeri B-t-ből következik, hogy B is ismeri A-t. Ha a felhasználó nem ad meg egyetlen embert sem a szimulációk nem futtathatóak.

Mikor a felhasználó végez a kapcsolatok létrehozásával mentődnek ezek két .txt fájlba. Az első fájlban az emberek azonosítói találhatóak meg, a másodikban az emberek soronként jelennek meg, melettük feltüntetve a kapcsolataik. Például:

1

2

25

...

Az első .txt fájl tartalma

---

1 2 4 25

A második .txt fájl tartalma, az 1-es azonosítójú ember, aki kapcsolatban áll a 2-es, 4-es, 25-ös azonosítójú emberekkel.

Emellett létezik olyan lehetőség is, hogy a felhasználó már egy korábban .txt fájlba mentett kapcsolatrendszert tölt be, azon változtathat is. A felhasználó a továbbiakban egy menüből választhat, a menüpont sorszámának beírásával. Ennek a menünek 5 fő pontja van:

1. Első Kísérlet - információáramlás
2. Második Kísérlet - feltételezések

3. Harmadik Kísérlet - viselkedési tendenciák terjedése
4. Új közösség létrehozása
5. Kilépés

Kísérletek megjelenítése

A kísérletekkor a képernyőn láthatóak lesznek táblázatszerűen az emberek azonosítói, illetve azok azonosítói felsorolásszerűen akikkel kapcsolatban vannak. Az adott kísérlet futtatása szempontjából fontos elemek új oszlopokként jelennek meg a táblázatban.

Első menüpont

Először egy rövid leírást láthatunk a kísérletről. Ezután a felhasználónak meg kell adnia, hogy a közösség mely tagja az információ forrása (az adott ember azonosítójának beírásával, pl. 1, 2, stb.). A szimuláció ezután lépésenként bemutatja, hogyan terjed az információ a közösség tagjai között.

> Azon embereket kell vizsgálni, akik birtokában vannak az információnak. Minden lépésben az ő összes olyan ismerősük aki nem értesült még a hírről értesül. A következő lépésben az ő ismerőseiket kell vizsgálni és így tovább. Előfordulhat, hogy lesz(nek) olyan(ok) aki(k) nem rendelkezik(nek) megfelelő összeköttetésekkel és nem jut majd el hozzá(juk) az információ.

A szimuláció lefutása után megjelenik egy felirat mely jelzi, hogy a kísérlet lefutott. Ekkor a felhasználó két lehetőség közül választhat: vagy újrajátsza a kísérletet (1), tetszőleges embert megadva az információ forrásaként, vagy kilép a szimulációból (0) és a menühöz kerül vissza.

Második menüpont

Először egy rövid leírást láthatunk a kísérletről. Ezután a felhasználónak emberenként meg kell adnia, hogy az adott vizsgált tulajdonság (pl. önkénteskedik-e?) igaz-e az adott emberre (minden, a képernyőn az azonosítójával és kapcsolataival megjelenített ember mellé sorban egy 0-t (hamis) vagy 1-et (igaz) kell írni). Ezután a szimuláció emberenként feltünteti, hogy ismerőseiknek hány százalékára igaz ez a tulajdonság.

> A szimuláció azon a feltevésen alapul, hogy az emberek hajlamosak az egész közösségre általánosítani ismerőseik tulajdonságait. Pl. ha egy ember ismerőseinek 50%-a önkénteskedik akkor hajlamos feltételezni, hogy az egész közösség fele is így tesz, holott ez egyáltalán nem biztos, hogy így van.

Így ahhoz, hogy a szimulációt könnyebb legyen értelmezni, a program megadja azt is, hogy a közösségnek valójában hány százalékára igaz ez a tulajdonság. Ezután a felhasználó két lehetőség közül választhat: vagy újrajátsza a kísérletet (0), újonnan megadott tulajdonságokkal, vagy kilép a menübe (1).

Harmadik menüpont

Először egy rövid leírást láthatunk a kísérletről.

> Mivel a kísérlet célja az, hogy megmutassa, hogy egy adott viselkedési minta miképp terjed a közösségen belül, a százalék amit a felhasználó megad arra vonatkozik, hogy egy ember ha ismerőseinek x%-a rendelkezik ezzel a tulajdonsággal átveszi tőlük.

Majd, amennyiben előtte elvégeztük az előző kísérletet a közösség tagjai megöröklik az ott megadott tulajdonságaikat, a felhasználónak csupán egy százalékot kell megadnia ami fölött átveszik ismerőseik tulajdonságait. Amennyiben a felhasználó az előző kísérletet nem végezte el most kell megadnia tagonként, hogy a tulajdonság igaz vagy hamis (az előzőhez hasonló módon). Ezután a szimuláció lépésenként bemutatja, hogyan veszik át a közösség tagjai egymástól ezt a tulajdonságot, hogyan terjed az.

> A szimuláció ugyanúgy működik majd, mint az első menüpontban lévő, hiszen ez egy feltételhez kötött több forrásos információáramlás. A feltétel kimerül abban, hogy minden szomszédnál ellenőrizni kell, az (igaz ismerősök/hamis ismerősök*100)≥ megadott százalék-e.

Ezután a felhasználó két lehetőség közül választhat: vagy újrajátsza a kísérletet (0), újonnan megadott tulajdonságokkal, vagy kilép a menübe (1).

Negyedik menüpont

Törli a jelenleg megadott közösséget, újat kell megadnia a felhasználónak.

Ötödik menüpont

Kilépés, leállítja a programot, felszabadítja a lefoglalt memóriaterületeket.

Programozói dokumentáció

A program három, közösségi hálókkal foglalkozó szociológiai kísérletet modellez. Ezen kívül képes a console ablakra megfelelően beírt adatokat fileba menteni, onnan később beolvasni, a programon belül új adatokat felvenni a régiek helyére, majd az összes dinamikusan lefoglalt terület felszabadításával a futást abbahagyni.

main.c

A main.c-ben találhatóak a felhasználó számára elhelyezett, segítő/magyarázó szövegeken felül a beolvasás kiválasztott típusát meghívó függvényrészek, illetve a két főbb menüt hívó függvényrészek (ezekről majd később). A főmenüből történő visszatérés után a main hívja meg a delete_all függvényt is, melynek segítségével a dinamikusan foglalt területek felszabadulnak.

datamgmt.c + datamgmt.h

> Ez a modul tartalmazza a program alapját adó adatszerkezeteket, illetve az egyszerűbb, az egész programban többször használt függvényeket - ezek legtöbbször az adatszerkezetek kezelését könnyítik meg.

```
typedef struct List1{
    char *person;
    List2 *connections;
    bool information;
    bool characteristic;
    int percentage;
    struct List1 *next;
}List1;
```

Ez a lista tárolja majd a program futásához szükséges összes adatot a kísérletben résztvevő emberenként. Azért választottam ezt, mivel a felhasználón múlik azemberek, kapcsolatok száma, így dinamikusan foglalt területen kell elhelyezni, dinamikus tömbként pedig nem lenne praktikus kezelni azadatokat, már csak a pazarló újrafoglalások miatt sem, bináris fa pedig a kapcsolatok jellegéből adódóan nem jöhetne létre. A lista első komponense az ember egyedi azonosítója, dinamikusan foglalt sztringre mutató pointer (hiszen a felhasználón múlik az azonosító hossza, szám/betű/szó...) , második komponense egy List2 lista első elemére mutató pointer, mely az adott személy kapcsolatait tárolja. Harmadik és negyedik komponense két boolean melyek a kísérletek elvégzésekor bizonyulnak majd hasznosnak, csak úgy, mint az int típusú százalékot tároló komponens. (A *százalékot tartalmazó típus azért int és nem double vagy float, mivel a programnak nem kell tizedesjegy pontossággal számolnia az adattal. Az int-tel való számolás százalékszámításkor egészrész számításként is értelmezhető, így nem történhet meg, hogy felfele kerekítés miatt hibás eredményeket ad a program - pl. 50 százaléktól számít, 49.8-at 49-ként értelmezi a számítások sorrendjének köszönhetően.*). Az utolsó komponens a List1 lista következő elemére mutató pointer.

```
typedef struct List2{
    char *connection;
    struct List2 *next;
}List2;
```

A List2 lista tárolja azok azonosítóit, akikkel az adott List2-t tartalmazó List1 elemnek megfelelő személy kapcsolatban áll. A választás oka ugyanaz, mint az előbb. Első komponense egy dinamikusan foglalt sztringre mutató pointer (a kapcsolatban lévő ember azonosítója), második a List2 lista következő elemére mutató pointer.

```
char *define_digits(const char *source, int start);
```

A függvény képes egy paraméterként megadott sztringből a szintén paraméterként megadott int elemétől a következő spaceig vagy lezáró nulláig kivágni egy sztringet, majd visszatérni ennek értékével. A program főként beolvasáskor az azonosítók meghatározására használja.

```
bool compare (const List1 *search, const char *ID);
```

A függvény összehasonlítja egy paraméterként megadott List1 elem azonosítóját egy szintén paraméterként megadott sztringgel. Bool típusú, visszatérési értéke true ha a kettő azonos, false ha nem.

```
bool compare2 (const List2 *search, const char *ID);
```

A függvény pontosan ugyanúgy működik ugyanazzal a céllal, mint az előző, csak List1 helyett List2 elemek azonosítóját vizsgálja.

```
List1 *search(List1 *where, const char *what);
```

A függvény kikeresi a paraméterként kapott List1 listából azt az elemét amelynek azonosítója megegyezik a szintén paraméterként kapott sztringgel. Visszatérési értéke az erre az elemre mutató pointer.

```
List2 *new_connection_list2(List2 *where, char *what);
```

A függvény egy, paraméterként megadott List2 listához hozzáfűzi a szintén paraméterként megadott sztringet. Külön kezeli azt az esetet ha a lista üres. Visszatérési értéke a List2 lista elejére mutató pointer.

```
List2 *delete(List2 *wherefrom, char *what);
```

A függvény egy, paraméterként megadott List2 listából törli a szintén paraméterként megadott sztringet. Külön kezeli azokat az eseteket, ha a sztring nincs a lsitában killetve ha a lista eredetileg csak egy elemű (az viszont a sztring). Visszatérési értéke a List2 lista elejére mutató pointer.

readin.c + readin.h

> Ez a modul a console-ról való beolvasást segítő függvényeket tartalmazza.

```
char *readin_one_row(char c);
```

Ez a függvény beolvas egy sort a consoleról. Paraméterként megkapja a beolvasás első karakterét, visszatérési értéke egy '\0'- val lezárt, dinamikusan foglalt sztring.

```
List1 *add_new_person(List1 *where, char *who);
```

Ez a függvény egy paraméterként megadott List1 listához fűz hozzá egy új elemet, melynek azonosítója a szintén paraméterként megadott sztring. Az új listaelem minden komponensét inicializálja. Külön kezeli azt az esetet, mikor a lista üres. Visszatérési értéke a lista első elemére mutató pointer.

```
List1 *remove_person(List1 *where, char *who);
```

A függvény egy paraméterként megadott List1 listából törli azt az elemet, melynek azonosítója szintén paraméterként megadott sztring. Külön kezeli azt az esetet ha a lista üres, illetve ha az első a törlendő elem. Visszatérési értéke a lista első elemére mutató pointer.

```
List1 *add_new_connection(List1 *where, char *one, char *two);
```

Ez a függvény egy paraméterként megadott List1 lista két, szintén paraméterként megadott azonosítójú elemének List2 kapcsolatlistájához fűzi hozzá a másik azonosítóját, ha az még nincs benne. Visszatérési értéke a lista elejére mutató pointer.

```
List1 *remove_connection (List1 *where, char *one, char *two);
```

Ez a függvény egy paraméterként megadott List1 lista két, szintén paraméterként megadott azonosítójú elemének List2 kapcsolatlistájából törli a másik azonosítóját. Visszatérési értéke a lista elejére mutató pointer.

```
List1 *readin(List1 *Data);
```

Ez a függvény felelős a consoleablakból történő beolvasásért, az ő feladat az esetekre szétválasztás illetve a segédfüggvények meghívása. Paraméterként megkapja a List1 listát amit fel kell töltenie adatokkal, visszatérési értéke a módosított lista első elemére mutató pointer.

filemgmt.c + filemgmt.h

> Ez a modul tartalmazza a fájlkezeléssel kapcsolatos függvényeket.

```
void file_out(List1 *Data);
```

Két, a specifikációban megadott formátumú .txt fileba írja ki a paraméterként megadott List1 lista adatait.

```
char *readin_one_row_file(FILE *fp, char c);
```

Beolvas egy sort egy fájlból és egy dinamikus sztringbe teszi (visszatérési értéke is ez). Praméterként megkapja a filet amiből be kell olvasson, illetve a beolvasott sor első karakterér.

```
List1 *add_connection_file(List1 *where, char *source);
```

Egy paraméterként megadott List1 listának két elemének List2 kapcsolat listáihoz fűz hozzá, miután a szintén paraméterként kapott sztringből kivágta azok azonosítóit. Visszatérési értéke a lista elejére mutató pointer.

```
List1 *freadin(List1 *Data);
```

Ez a függvény felel a fileból való beolvasásért, paraméterként kapja meg a filet amibe olvasnia kell, viisszatérési értéke a módosított List1 lista első elemére mutató pointer.

menu.c + menu.h

> Ebbe a modulba tartoznak a menük létrehozásáért felelős függvények.

```
void read_menu(List1 *Data);
```

Ez a függvény felelős a további beolvasást kiválasztó menü meghívásáért. Mivel meghívja egyes esetekben a beolvasás függvényt ezért paraméterként át kell neki adni a beolvasás függvény által módodított List1 listát mely az adatokat tárolja. Visszatérési értéke nincs.

```
List1 *menu_universal(List1 *Data, int menuPoint);
```

A függvény hívja meg a menüt, mely minden kísérlet lefutása előtt/után megjelenik, segítségével a felhasználó kiléphet a főmenübe/újrajátszhatja a kísérletet. Paraméterként át kell neki adni a listát melyet a kísérletek függvényei is megkapnak, illetve egy inetet melynek segítségével a függvény eldönti, hogy melyik függvényt kell meghívnia. Visszatérési értéke a paraméterként megadott a kísérletekhez szükséges adatokat tartalmazó List1 lista.

```
List1* menu(List1 *Data);
```

Ez a főmenüt meghívó menü. Paraméterként megkapja az alanyokról az összes adatot tartalmazó listát, ezt adja majd tovább az előbb említett menu_universal függvénynek, amely majd továbbadja a kísérletek függvényeinek. Visszatérési értéke az ugyanennek a listának az első elemére mutató függvény. '4' esetén meghívja a közösséget törlő függvényt és újat vesz fel, '5' esetén a függvényből visszakerül a mainbe.

information.c + information.h

> Ebbe a modulba tartoznak az első kísérlettel kapcsolatos függvények. Egy-két függvényt a spreading.c -ben is felhasznál a program.

```
typedef struct ListofChars
{
    char *IDs;
    struct ListofChars *next;
}ListofChars;
```

Ez a Lista sztringeket tartalmaz, az alanyok azonosítóinak tárolására használja a program. Minden listaelem egy dinamikusan foglalt sztringet '\0'-val lezárva illetve a lista következő

elemére mutató pointert tartalmaz. Azért listát használtam mivel dinamikus tömb esetén a realloc nagyon pazarló lett volna a bináris fát pedig az adatstruktúra nem indokolja.

```
List1 *bool_initializer (List1 *Data);
```

A függvény egy paraméterként átvett List1 lista minden elemén végigmegy és az information komponensüket hamissá változtatja. Visszatérési értéke a megváltoztatott lista első elemére mutató pointer.

```
void print_out(List1 *Data);
```

A függvény egy paraméterként átvett List1 lista minden elemének soronként kiírja azonosítóját, kapcsolatainak azonosítóit és 1-est ha information true, 0-ást ha false. Nincs visszatérési értéke.

```
bool check(List1 *Data);
```

Ez a függvény ellenőrzi, hogy egy paraméterként átvett List1 listának van-e olyan eleme, amihez tartozó information komponens hamis, de kapcsolatai alapján még lehetne igaz. Ha van ilyen akkor a boolean típusú függvény visszatérési értéke false, ellenkező esetben true.

```
List1 *changer(List1 *Data, ListofChars *IDs, bool *n_c);
```

Ez a függvény megváltozatatja a paraméterként átvett List1 lista néhány elemének information komponensét. Végigmegy a listán és ha olyan elemet talál, ahol az information komponens hamis, de az elemhez tartozó List2 listának és a paraméterként megadott ListofChars listának van metszete akkor az informationt igazra változtatja. Ha a függvény legalább egy ilyen cserét végrehajt a címével átvett bool pointer értéke hamisra változik. A függvény visszatérési értéke a módosított List1 lista első elemére mutató pointer.

```
ListofChars *add_char(ListofChars *where, char *what);
```

A függvény a paraméterként kapott lista végére fűzi a szintén paraméterként kapott sztringet. Visszatérési értéke a lista első elemére mutató pointer.

```
ListofChars *adder(ListofChars *not_init, List1 *Data);
```

A függvény kikeresi a paraméterként kapott List1 listából azokat az elemeket melyek information komponense true, de még nincsenek benne a szintén paraméterként kapott ListofChars listában és hozzáadja őket. Visszatérési értéke a módosított lista első elemére mutató pointer.

```
void free_LoC(ListofChars *what);
```

Ez a függvény felszabadítja egy paraméterként megadott ListofChars lista minden elemét.

```
List1 *information (List1 *Data);
```

Ez a függvény felel az első kísérlet meghívásáért. Paraméterként kapja azt a List1 listát ami a kísérlet alanyainak adatait tartalmazza, visszatérési értéke ugyanennek a listának az első elemére mutató pointer.

misconceptions.c + misconceptions.h

> Ebben a modulban találhatóak a második kísérlet végrehajtásához szükséges függvények.

```
List1 *give_characteristics(List1 *Data);
```

Ez a függvény paraméterként kap egy List1 listát. Segítségével a felhasználó listaelemenként megadhatja, hogy a characteristic komponens igaz vagy hamis legyen (0 vagy 1 consolera írásával). Visszatérési értéke a módosított lista első elemére mutató pointer.

```
List1 *calculator(List1 *Data);
```

Ez a függvény végigmegy a paraméterként megadott List1 lista minden elemén és azok kapcsolatai alapján kiszámolja, hogy ismerőseik hány százalékára igaz a characteristic komponens. Ez az egész számként megadott érték a percentage komponensbe kerül. Visszatérési értéke a módosított lista első elemére mutató pointer.

```
void write_out(List1 *Data);
```

A függvény egy paraméterként átvett List1 lista minden elemének soronként kiírja azonosítóját, kapcsolatainak azonosítóit, percentage komponensét és 1-est ha characteristic true, 0-ást ha false. Nincs visszatérési értéke.

```
int average_of_sum(List1 *Data);
```

A függvény kiszámolja, hogy egy paraméterként átvett List1 lista elemeinek hány százalékának true a characteristic komponense. Ez az int a függvény visszatérési értéke.

```
List1 *misconceptions(List1 *Data);
```

Ez a függvény felel a második kísérlet meghívásáért. Paraméterként kapja azt a List1 listát ami a kísérlet alanyainak adatait tartalmazza, visszatérési értéke ugyanennek a listának az első elemére mutató pointer.

spreading.c + spreading.h

> Ebbe a modulba tartoznak a harmadik kísérlettel kapcsolatos függvények (kivéve párat amik az information.c-ben is megjelentek már).

```
ListofChars *who_are_1(List1 *Data, ListofChars *where_to);
```

Ez a függvény kiválasztja egy paraméterként kapott List1 lista azon elemeit, melyek characteristic komponense 1 és bemásolja óket a szintén paraméterként kapott ListofChars listába. Visszatérési értéke ennek a listának az első elemére mutató pointer.

```
void print_out_3(List1 *Data);
```

A függvény egy paraméterként átvett List1 lista minden elemének soronként kiírja azonosítóját, kapcsolatainak azonosítóit, percentage komponensét és 1-est ha characteristic true, 0-ást ha false. Nincs visszatérési értéke.

```
List1 *transformer(List1 *Data, ListofChars *are_true, bool *n_c, int percentage);
```

Ez a függvény megváltozatatja a paraméterként átvett List1 lista néhány elemének characteristic komponensét. Végigmegy a listán és ha olyan elemet talál, ahol a characteristic komponens hamis, de az elemhez tartozó percentage nagyobb, mint a paraméterként átvett int akkor igazra változtatja. Ha a függvény legalább egy ilyen cserét végrehajt a címével átvett bool pointer értéke hamisra változik. A függvény visszatérési értéke a módosított List1 lista első elemére mutató pointer.

```
ListofChars *adder_3(ListofChars *not_init, List1 *Data);
```

A függvény kikeresi a paraméterként kapott List1 listából azokat az elemeket melyek characteristic komponense true, de még nincsenek benne a szintén paraméterként kapott ListofChars listában és hozzáadja őket. Visszatérési értéke a módosított lista első elemére mutató pointer.

```
bool check_3(List1 *Data);
```

Ez a függvény ellenőrzi, hogy egy paraméterként átvett List1 listának van-e olyan eleme, amihez tartozó characteristic komponens hamis, de kapcsolatai alapján még lehetne igaz. Ha van ilyen akkor a boolean típusú függvény visszatérési értéke false, ellenkező esetben true.

```
bool all_is_false(List1 *Data);
```

Ez a függvény ellenőrzi, hogy a paraméterként megadott List1 lista minden elemének characteristic komponense hamis-e. Ha igen, a visszatérési érték igaz, ha nem, hamis.

```
List1 *make_it_false(List1 *Data);
```

Ez a függvény a paraméterként megadott lista minden elemének characteristicét false-ra változtatja. Visszatérési értéke a lista első tagjára mutató pointer.

```
List1 *spreading(List1 *Data);
```

Ez a függvény felelős a harmadik kísérlet végrehajtásáért. Paraméterként egy list1 listát kap, benne a szükséges adatokkal. Visszatérési értéke a lista elejére mutató pointer.

delete_society.c + delete_society.h

> Ebbe a modulba tartoznak a dinamikusan foglalt adatszerkezetek felszabadítását segítő függvények.

```
void clear_list1(List1 *lista);
```

Ez a függvény felszabadítja a paraméterként megadott List1 listát.

```
void clear_list2(List2 *lista);
```

Ez a függvény felszabadítja a paraméterként megadott List2 listát.

```
void delete_all(List1 *Data);
```

Ez a függvény felszabadítja az egész, paraméterként megadott List1 listát és annak minden dinamikusan fogalalt részét (List2 lista, kétféle dinamikus sztringek). Visszatérési értéke nincs.

Felhasználói dokumentáció

💡 A szociológia egyik alapja, hogy a társadalmakban bekövetkező eseményeket, változásokat, kialakult normákat ne csak az egyének alapján, hanem a köztük lévő kapcsolati hálókon keresztül is vizsgáljuk. A program három különböző szociológiai kísérletet modellez és ezek segítségével mutatja meg, milyen fontosak a közösségi hálók a társadalmakban.

A program, főként az inspirációként használt források miatt angol nyelvű. Elindítása után a felhasználó feladata megadni a közösséget, amellyel a továbbiakban dolgozni szeretne.

```
F - read in from previously made files.
C - read in from console window.
```

A felhasználónak a console ablakba kell írnia egy C betűt és egy entert ha a beolvasást a programablakból - továbbiakban a console-ról - folytatná, egy F-et és egy entert ha fileból olvasna be. FIGYELEM! fileból csak akkor lehet beolvasni, ha a felhasználó már legalább egyszer futtatta a programot, hiszen előbb ki kell írni egy közösséget csak aztán lehet beolvasni (első futtatás előtt a beolvasandó fileok nem léteznek még).

⚠️ Általánosan igaz a programra, hogy console-ra való íráskor egy, összetartozó sor után mindig entert kell nyomni. Erre a továbbiakban nem fog a dokumentáció figyelmeztetni és a program sem teszi ezt minden esetben.

Beolvasás Consoleról

Az emberek és kapcsolatok megadásának 4 lehetősége van, ezek mind külön sorokba kell kerüljenek (ENTER!).

Az IDk a kísérletben résztvevő alanyok egyedi azonosítói - ennek megfelelően nem szabad két alanynak ugyanazt az IDt adni.

⚠️ Az ID bármi lehet, név, szám, stb., **két dolog nem:** szókapcsolat melyben szóköz szerepel, illetve bármi ami nagy X-et tartalmaz.

→ a kísérletek akkor a legkönnyebben áttekinthetőek, ha minél kevesebb, és/vagy egyenlő mennyiségű karakterből állnak az azonosítók.

e+ ID

//ember hozzáadása a kísérlethez

e- ID

//ember kitörlése a kísérletből

⚠️ Nem szabad olyan embert törölni aki még nincs létrehozva.

k+ ID1 ID2

//kapcsolat létrehozása két ember között

⚠️ Csak már létrehozott emberek közt lehet létrehozni kapcsolatot.

⚠️ Minden kapcsolatot egyszer lehet létrehozni.

k- ID1 ID2

//kapcsolat törlése

⚠️ Nem szabad olyan kapcsolatot törölni ami még nincs létrehozva.

Amennyiben a felhasználó végzett az adatok beírásával, a következő sorba tett **X**-el léphet ki a függvényből.

⚠️ Legalább két embert és egy kapcsolatot létre kell hozni, hiszen ezalatt a szükséges minimum alatt a kísérleteknek nincs semmi értelmük. **A program nem determinisztikus a nem az előírásoknak megfelelően megadott bemenetekre!**

A console-os megadás után, ha a megadás helyes, a program automatikusan létrehozza a megadott adatokból a beolvasáshoz szükséges fileokat, az *IDs.txt*-t és a *connections.txt*. A felhasználónak ezekkel a következő fájlból beolvasásig nincs tennivalója.

Beolvasás fileból

A kért helyre írt F után a felhasználónak nincs több tennivalója a megfelelő függvény meghívódik és a megfelelő fájlokból beolvassa az adatokat.

Menük

Az egyes számú menü a beolvasáshoz kapcsolódik. Itt a felhasználó mindig dönthet, hogy szeretne-e még embereket vagy kapcsolatokat hozzáadni a már létrehozott közösséghez (**1**) vagy haladna tovább a kísérletekkel (**0**).

Az utóbbi esetben a felhasználó a főmenübe kerül. Itt 5 menüpont közül választhat:

1. Első Kísérlet - információáramlás
2. Második Kísérlet - feltételezések
3. Harmadik Kísérlet - viselkedési tendenciák terjedése
4. Új közösség létrehozása
5. Kilépés

Egy menüpont kiválasztásához a felhasználónak be kell írnia a kiválasztott cselekmény előtti számot. A főmenübe a kísérletek után vissza lehet térni a 0-ás beírásával, 1-es esetén pedig mindig újra lehet játszani az adott kísérletet. A főmenü pontjait alább részletezem:

- *A főmenü pontjain nem muszáj sorban haladni* -

Menüpont 1

> A szimuláció a közösségen belüli információáramlást mutatja be.

A felhasználónak meg kell adnia egy létező azonosítót, hogy ki legyen az információ forrása. Ezután lefut a kísérlet - lépésenként mutatja be, hogyan terjed az információ - kiírja consolera a táblázatot, melynek minden sorában egy embernek az azonosítója, kapcsolatai és egy 1-es(értesült) vagy 0-ás (nem értesült) szerepel. A kísérlet megáll magától. Innen a már említett menü segítségével választhat a felhasználó, hogy újrajátsza a kísérletet vagy visszalép a főmenübe.

Menüpont 2

> A szimuláció azon a feltevésen alapul, hogy az emberek hajlamosak az egész közösségre általánosítani ismerőseik tulajdonságait. Pl. ha egy ember ismerőseinek 50%-a önkénteskedik akkor hajlamos feltételezni, hogy az egész közösség fele is így tesz, holott ez egyáltalán nem biztos, hogy így van.

A felhasználónak alanyonként meg kell adnia, hogy igaz-e, hogy önkénteskedik (**1** ha igaz, **0** ha hamis). Ezután nincs más tennivalója, a console-on meg fog jelenni egy táblázat, melynek egy sorában egy ember azonosítója, kapcsolatai, az előbb megadott jellemzője, illetve egy arány, hogy ismerőseinek hány százalékára igaz ez a tulajdonság fog szerepelni. Innen a már említett menü segítségével választhat a felhasználó, hogy újrajátsza a kísérletet vagy visszalép a főmenübe.

Menüpont 3

> A szimuláció lényege, hogy megmutatja hogyan befolyásol minket a környezetünk - minél nagyobb százaléka önkénteskedik az ismerőseinknek annál valószínűbb, hogy mi is így fogunk tenni.

→ Ha a felhasználó már végrehajtotta a második kísérletet csak egy százalékot kell megadnia, - ez jelképezi majd az emberek ingerküszöbét - akinek az ismerőseinek ennél nagyobb, vagy ennyi százaléka önkénteskedik, az szintén el fog kezdeni.

→ Ha nem, a felhasználónak alanyonként meg kell adnia, hogy igaz-e, hogy önkénteskednek (**1** ha igaz, **0** ha hamis). Majd meg kell adnia az előbb említett százalékot is.

Ezután a felhasználónak nincs más tennivalója, a console-on meg fog jelenni minden lépés után egy táblázat, melynek egy sorában egy ember azonosítója, kapcsolatai, az előbb megadott jellemzője, illetve egy arány, hogy ismerőseinek hány százalékára igaz ez a tulajdonság fog szerepelni. Innen a már említett menü segítségével választhat a felhasználó, hogy újrajátsza-e a kísérletet vagy visszalép a főmenübe.

Menüpont 4

Az előbb megadott közösség törlődik, a felhasználó pedig újat adhat meg, visszakerül a már említett beolvasás fázishoz. Innentől kezdve a program pontosan úgy működik, ahogyan az első közösséggel tette.

A program vége

A program akkor ér véget ha a felhasználó a menüben az 5-ös, kilépés pontot választja. Innentől kezdve a felhasználónak nincs semmilyen tennivalója.