**Programozás alapjai 1**

**Házi feladat adatszerkezetek**

**Papp Dominik Edvárd EAT3D9**

**Forrásfájlok** (a zárójelek csak kommentek, nem kerülnek bele a fájlokba)

Az éttermek adatait éttermenként egy sorban tárolja az etterem.txt fájl. Egy soron beül az elemek a ’;’ karakterrel vannak elválasztva. Egy étteremre egy példa:

*123421(Étterem ID, unsigned int)*

*Laposföld étterem (neve: max. 50 karakter hosszú, szóközöket tartalmazhat)*

*Arany János utca 42/a (címe: max. 50 karakter hosszú, szóközöket tartalmazhat)*

*Olasz (a konyha nemzetisége: max. 30 karakter)*

*47.4983 19.0408(Gps koordinátái: két valós érték. Előbb az északi-szélességi majd a keleti-hosszúsági fokok)*

*4.6(minősítése: egy valós érték)*

*$$(árkategória max. 3 karakter ($$$))*

*Igen (terasz elérhetősége: igen/nem)*

*„123421*;*Laposföld étterem*;*Arany János utca 42/a;Olasz*;*47.4983;19.0408*;*4.6*;*$$*;*Igen(\n ha nem EOF)”*

*A fenti sor kerül az etterem.txt fájlba.*

Az éttermek asztalainak elérhetőségeit egy külön fájl tárolja, az asztalok.txt fájl. A két fájl közötti kapcsolatot az étterem ID-ja teremti meg. Minden sorban szerepel az étterem ID-ja, majd, hogy hány fős asztalból, hány darab elérhető van. A nem elérhető asztalokat is feltünteti (feltesszük, hogy az elérhető asztalokból legalább egy a teraszon van, ha van terasz). A tagok itt is ’;’ karakterrel vannak elválasztva, de soronként vannak elválasztva a különböző asztalok. Egy étterem asztalaira egy példa.:

*123421;2;6*

*123421;3;2*

*123421;4;3*

*123421;6;1*

*123421;8;0*

A harmadik fájl a user elvárásait tartalmazza, amivel a program dolgozni fog. Ennek a fájlnak a neve user.txt. Példa a user adataira:

*1024(a kör sugara, amelyen belül keresi az éttermet) (valós szám méter egységben)*

*47.507350 19.026352(Gps koordinátái)*

*Olasz Kínai Japán (A konyha nemzetisége. (max. 3 sorolható fel, több mint egy megadása azt jelenti, hogy a usernek mindegy, milyen az étterem konyhája, amíg azok közül az egyik)*

*4.1(Az étterem minősítése (alsó határ, legalább ennyi legyen))*

*$$(Árkategória, felső határ, legfeljebb ilyen drága legyen)*

*Igen (Teraszrész elérhetősége (teraszon akar ülni vagy nem))*

*6(A leülni kívánó személyek száma)*

*„1024; 47.507350;19.026352; Olasz Kínai Japán; 4.1;$$;Igen;6”*

**Adatszerkezet**

Az adatok tárolására egy két irányba láncolt fésűslista van alkalmazva, melyben az első fájl adataiból képzett elemekből fog indulni a második fájl adataiból képzett láncolt lista, amely már csak előre láncolt. A lista dinamikusan foglalt elemekből áll. A listában az etterem elemek egymást minősítés szerinti csökkenő sorrendben követik, a gyorsabb működés érdekében.

Az első fájl adataiból képzett struktúra:(nagyhazi.c 5. sor)

typedef struct etterem{

    unsigned int id;

    char nev[50];

    char cim[50];

    char konyha[30];

    double eszaki;/\* északi keleti koordináták\*/

    double keleti;

    double minosites;/\*5.0-1.0\*/

    char arkat[4];/\*$ $$ $$$\*/

    char terasz[4];/\*igen/nem\*/

    struct etterem \*next;/\*következő étterem\*/

    struct etterem \*prev;/\*előző étterem\*/

    struct asztalok \*head;/\*asztalok listájára mutató pointer\*/

}etterem;

A második fájl adataiból képzett struktúra:(nagyhazi.c 20. sor)

typedef struct asztal{

    unsigned int id;/\*ez megegyezik az etterem.id -val\*/

    int ferohely;/\*asztal ülőhelyeinek száma\*/

    int szabad;/\*szabad asztalok száma\*/

    struct asztalok \*next;/\*az étterem asztalainak következő elemére mutat\*/

}asztal;

A harmadik fájlból képzett struktúra:(nagyhazi.c 27. sor)

typedef struct user{

    int sugar;

    double eszaki;/\* északi keleti koordináták\*/

    double keleti;

    char konyha1[20];

    char konyha2[20];/\*csak akkor kerül bele adat, ha több mint egy van megadva. Egyébként csupa '\0'\*/

    char konyha3[20];

    double minosites;

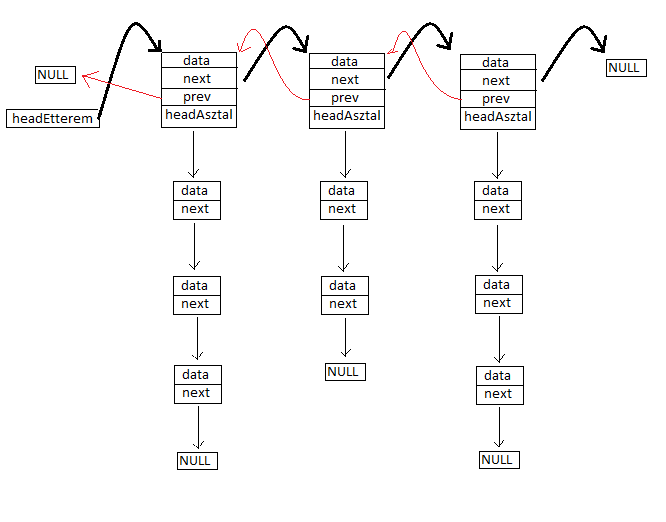
    char arkat[4];/\*$ $$ $$$ felsőhatár\*/

    char terasz[5];/\*igen/nem\*/

    int fo;/\*legalább ennyi fős asztal kell\*/

}user;

A listák viszonyai szemléltetve: (a data minden olyan struktúra elemet foglal magába, amely nem egy pointer)

****

**Adatok beolvasása és függvények**

A program átláthatósága segédfüggvényekkel van biztosítva.

etterem\* create\_etterem(void)

A create\_etterem() dinamikusan foglal „kinullázott” memóriát egy etterem struktúra elemnek.

Paraméterek: Nincs

Visszatérési érték: etterem\*

nagyhazi.c 40. sor

asztal\* create\_asztal(void)

A create\_asztal() dinamikusan foglal „kinullázott” memóriát egy asztal struktúra elemnek.

Paraméterek: Nincs

Visszatérési érték: asztal\*

nagyhazi.c 44.sor

user\* create\_user(void)

A create\_user() dinamikusan foglal „kinullázott” memóriát egy user struktúra elemnek.

Paraméterek: Nincs

Visszatérési érték: user\*

nagyhazi.c 48.sor

etterem\* insert\_etterem(etterem\* insert, etterem\* head)

Az insert\_etterem() beszúr a lácolt listába egy etterem elemet.

0 elem esetén a head a beszúrni kívánt elem lesz. Az elemet úgy szúrja be, hogy az éttermek a head-next irányába minősítés szerinti csökkenő sorrendben legyenek.

Paraméterek: etterem\* insert A beszúrni kívánt elem.

etterem\* head A láncolt lista kezdőcíme.

Visszatérési érték: etterem\*

nagyhazi.c 52. sor

void insert\_asztal(asztal\* insert, etterem\* head)

Az insert\_asztal() hozzáláncolja a megfelelő étteremhez tartozó asztalt az ahhoz tartozó lista végéhez.

Paraméterek: asztal\* insert A beláncolandó asztal elem.

etterem\* head A láncolt lista kezdőcíme.

Visszatérési érték: Nincs

nagyhazi.c 90. sor

etterem\* beolvas\_etterem(etterem\* head)

A beolvas\_etterem() egy etterem elemet feltölt adatokkal.

A beolvas\_etterem() fő funkciója a fent említett, azonban a program egyik algoritmusa is. Hogy legyen hova beolvasni létrehoz egy etterem elemet a create\_etterem() függvénnyel. Ezt az elemet tölti fel adatokkal majd az insert\_etterem függvénnyel beilleszti a listába. A beolvas\_etterem() az ettermek.txt összes adatsorát feldolgozza és megépíti a fésűs lista gerincét. Kevés étteremhez tartozó adat esetén hibaüzenetet ír ki a standard outputra. Ilyenkor a program helyes működése nem garantált. A függvény kezeli az ettermek.txt fájl kinyitását és becsukását.

Paraméterek: etterem\* head A láncolt lista kezdőcíme.

Visszatérési érték: etterem\*

nagyhazi.c 103. sor

void beolvas\_asztal(etterem\* head)

A beolvas\_asztal() egy asztal elemet feltölt adatokkal.

Ennek a függvénynek a lefolyása hasonló a beolvas\_etterem()-éhez. Létrehoz a create\_asztal() függvénnyel egy elemet, ahova be tud olvasni, majd beilleszti a megfelelő helyre az insert\_asztal() függvénnyel. A beolvas\_asztal() függvény az asztal.txt összes adatsorát feldolgozza és nem megfelelő mennyiségű adat esetén hibaüzenetet ír ki a standard outputra. Ilyenkor a program megfelelő működése nem garantált. A függvény kezeli az asztalok.txt fájl kinyitását és becsukását.

Paraméterek: etterem\* head A láncolt lista kezdőcíme.

Visszatérési érték: Nincs

nagyhazi.c 119. sor

user\* beolvas\_user(void)

A beolvas\_user() egy user elemet feltölt adatokkal.

A függvény a create\_user() függvénnyel hozza létre a user elemet, ahova be fog olvasni. Nem megfelelő mennyiségű adat esetén hibaüzenetet ír ki a standard outputra. Ilyenkor a program megfelelő működése nem garantált. A függvény kezeli az user.txt fájl kinyitását és becsukását.

Paraméterek: Nincs

Visszatérési értek: user\*

nagyhazi.c 134. sor

etterem\* pop\_etterem(etterem\* pop, etterem\* head)

A pop\_etterem() felszabadít egy etterem elemet és az ahhoz tartozó teljes asztal listát.

A függvény külön kezel 4 esetet. A felszabadítani kívánt elem az egyetlen a listában, az első elem, az utolsó elem és amikor az elem tetszőleges helyen van a listában. Az első két esetben módosítja a lista kezdőcímére mutató pointert. A függvény újra láncolás után szabadít fel így a sorrend továbbra is kihasználható. Felszabadítás előtt megvizsgálja, hogy tartozik-e asztal az étteremhez. Amennyiben nem, hibaüzenetet ír ki a standart outputra.

Paraméterek: etterem\* pop A felszabadítani kívánt elem.

etterem\* head A láncolt lista kezdőcíme.

Visszatérési érték: etterem\* Ha a felszabadítani kívánt elem az egyetlen a fésűs listában akkor a visszatérési értéke NULL. Ez kihasználható, ha a teljes listát kell felszabadítani.

nagyhazi.c 147. sor

etterem\* search\_for\_bad(user\* user, etterem\* head)

A search\_for\_bad() feladata, hogy javítsa a program futási idejét a fésűs lista minősítés béli csökkenő sorrendjét kihasználva. A függvény megkeresi az első olyan elemet, amely már nem felel meg a user minősítés béli elvárásainak és attól az elemtől kezdve felszabadítja a lista összes további elemét a pop\_etterem() többszöri meghívásával.

Paraméterek: user\* user A user adataiból álló struktúra.

etterem\* head A láncolt lista kezdőcíme.

Visszatérési érték: etterem\* Ha nincsen megfelelő étterem akkor NULL pointerrel tér vissza. Ez kihasználható a függvény meghívásakor.

nagyhazi.c 172. sor

double tavolsag(user\* user, etterem\* a)

A tavolsag() kiszámolja a user és egy étterem közötti távolságot.

Mivel földrajzi koordináták állnak rendelkezésre ezért a függvény a Haversine formulát alkalmazza két pont közötti távolság kiszámítására egy gömbön. A függvény csak közelítő értéket tud számolni mivel a Föld nem szabályos gömb alakú. A függvény 6371km-es gömb (Föld) sugárral számol.

Paraméterek: user\* user A user adataiból álló struktúra.

etterem\* a Annak az étteremnek az adatait tartalmazó elem, amelynek a usertől mért távolságát kell kiszámolni.

Visszatérési érték: double Az étterem és a user közötti távolság méterben.

nagyhazi.c 185. sor

etterem\* compare(user\* user, etterem\* head)

A compare() a user minőség béli elvárásán kívül az összessel összehasonlítja a fésűs listában található összes étteremet. Amennyiben az étterem nem felel meg az elvárásoknak a pop\_etterem() függvénnyel felszabadítja azt. Ez a függvény is a program egyik algoritmusa. Tovább ritkítja az elvárásoknak megfelelő listát. A függvény lefutása után már csak olyan elemek maradnak a listában, amelyek megfelelnek az elvárásoknak.

Paraméterek: user\* user A user adataiból álló struktúra.

etterem\* head A láncolt lista kezdőcíme.

Visszatérési érték: etterem\* Amennyiben nincsen megfelelő étterem akkor NULL pointerrel tér vissza. Ez kihasználható a függvény meghívásakor.

nagyhazi.c 194. sor

void print\_etterem(user\* user, etterem\* head)

A print\_etterem() kiírja egy étterem adatait a standard outputra.

A függvény rekurzív és egy algoritmus is. A függvény feldolgozza listában található első elemet és kiírja az étterem: nevét, címét, az első olyan asztal méretét, ahova le tud ülni a társaság és a usertől mért távolságot(tavolsag() függvénnyel). A kiírás után az elemet felszabadítja a pop\_etterem()-el és meghívja önmagát újra. A print\_etterem() kihasználja a pop\_etterem() azon tulajdonságát, hogy ha a lista üres, akkor a visszatérési érték NULL pointer. Ez a print\_etterem escape feltétele. A függvényből való távozás után a fésűs lista teljesen üres.

Paraméterek: user\* user A user adataiból álló struktúra.

etterem\* head A láncolt lista kezdőcíme.

Visszatérési érték: Nincs

nagyhazi.c 227. sor

**A main függvény**

A main függvény feladata a segédfüggvények kezelése. Létrehozza a fésűs listát beolvas\_etterem() és a beolvas\_asztal() függvények meghívásával (ebben a sorrendben!). Létrehozza a user struktúra elemet is majd a search\_for\_bad() függvénnyel elkezdi csökkenteni a nem megfelelő éttermek elemeinek a számát. Ezután a compare() függvénnyel tovább csökkenti ezen elemek számát, majd így már csak a megfelelő elemek vannak a listában. Ilyenkor minden etterem elemet kiirat a standard outputra a print\_etterem() függvénnyel, ami felszabadítja az elemeket a kiírás után. Ezután a main függvény felszabadítja a user elemet és leáll a program. Amennyiben nincsen megfelelő étterem hibaüzenetet ír ki a standard outputra.

nagyhazi.c

**Algoritmusok**

A programnak 3 fő algoritmusa van melyeket a beolvas\_etterem(), compare() és print\_etterem() függvények valósítanak. Részletes leírás ezen algoritmusok működéséréről a függvények leírásainál található.