Programozás alapjai 1

Házi feladat adatszerkezetek

Papp Dominik Edvárd EAT3D9

Éttermek a közelemben

A program célja, hogy bárhol, bármikor, a helyzeti adataink és elvárásaink megadásával megtudjuk, hogy légvonalban egy adott távolságon belül mely éttermek felelnek meg a mi elvárásainknak.

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(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, és ez stdin-ként van kezelve. Példa a user adataira:

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

47.507350 19.026352(koordináták)

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)

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 fésűs listát fogok használni, melyben az első fájl adataiból képzett elemekből fog indulni a második fájl adataiból képzett láncolt lista, amely már csak előre láncolt. A lista mindenféleképpen dinamikusan foglalt elemekből kell hogy álljon. A listában az etterem elemek egymást minősítés szerinti csökkenő sorrendben követik, a gyorsabb működés érdekében.

Az első fájl adataiból képzett struktúra:

```
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[3];/*$ $$ $$$*/
    char terasz[4];/*igen/nem*/
    struct etterem *next;/*következő étterem*/
    struct etterem *prev;/*előző étterem*/
    struct asztalok *head;/*asztalok listájára mutató pointer*/
}etterem;
```

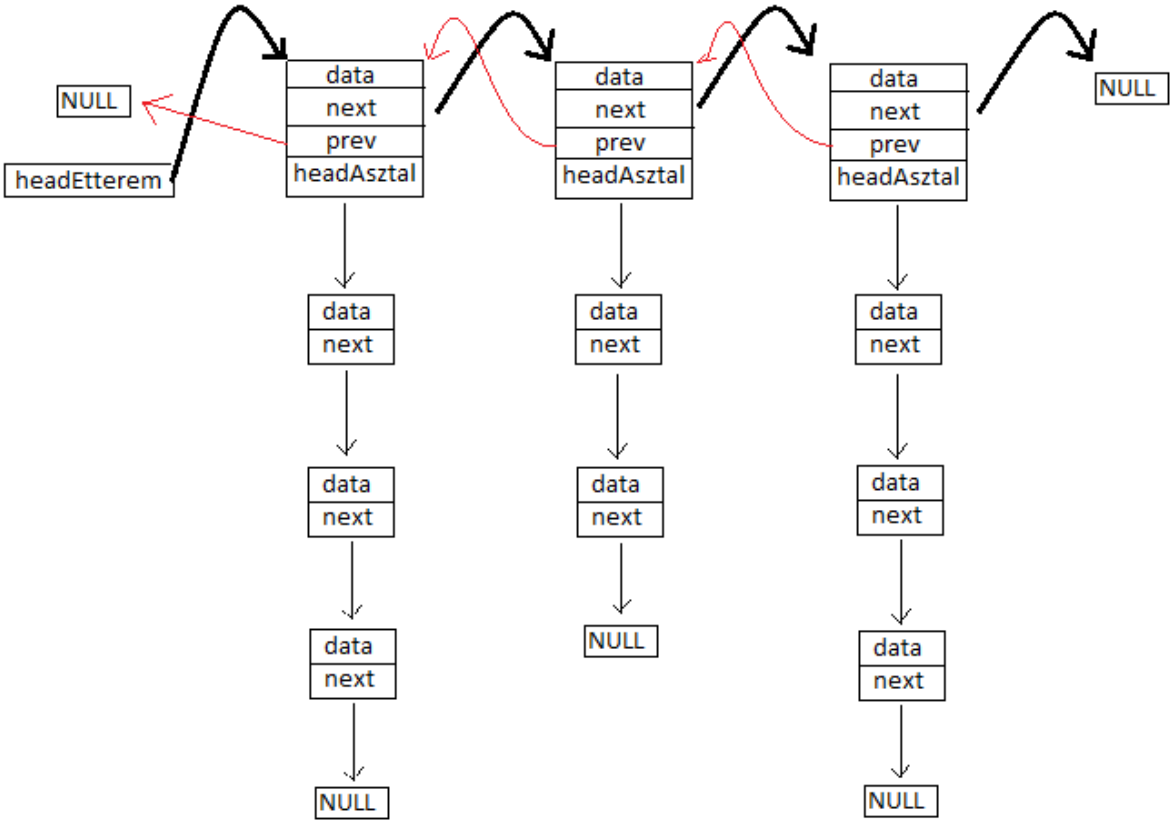
A második fájl adataiból képzett struktúra:

```
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:

```
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[3];/*$ $$ $$$*/
    char terasz[4];/*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 átláthatóságát segédfüggvényekkel biztosítom.

A create_etterem() egy étterem struktúra elemet hoz létre és dinamikusan foglal neki memóriát. A create_asztal() ugyanezt csinálja, csak asztal struktúra elemet hoz létre.

Minden create függvény az összes sztringet kinullázza.

Az adatokat egy beolvas_etterem() függvény fogja beolvasni az ettermek.txt-ből, és az asztal elemekre mutató pointert a NULL-ra irányítja. Az asztalok.txt-ből pedig a beolvas_asztal() fog beolvasni. Ezek a függvények meg fogják hívni a create függvényeket, hogy legyen hova írni a beolvasott adatokat, majd az insert függvényekkel be is illesztik a listába azokat.

Az insert_etterem() függvény a létrehozott etterem elemet beilleszti a listába, úgy hogy a minősítés szerinti csökkenő sorrend teljesüljön. Az insert_asztal() csak egymás után fűzi az elemeket. A következő asztalra mutató pointert pedig a NULL-ra irányítja. Ahhoz, hogy a

fésűs lista létrejöjjön, az asztal listát az etterem elemhez kell fűzni. Ezt a lace_asztal_to_etterem() fogja megcsinálni, mely megkeresi az asztal lista első elemének nevéhez tartozó etterem elemet, és hozzáfűzi.

Ezeknek a függvényeknek a segítségével létrejön egy fésűs lista, amellyel a program dolgozni fog, hogy a fő célját megvalósítsa. Ehhez szükséges a user elvárásainak struktúrában való tárolása.

A create_user() függvény dinamikusan foglal memóriát, létrehoz egy user elemet és a user.konyha(1,2,3) sztringeket kinullázza('\0'). Ez az összehasonlításokat egyszerűsíti.

A beolvas_user() meghívja a create_user() függvényt és feltölti a beolvasott adatokkal.

A pop_etterem() függvény felszabadít egy lista elemet úgy, hogy az asztal elemet is felszabadítja, (természetesen az utóbbi előbb valósul meg, mint az előbbi) és ha az első elemet szabadítja fel, akkor a head pointert is módosítja.

A search_for_bad() függvény megkeresi az első olyan elemet, amely már nem felel meg a user minősítési elvárásainak és a pop_etterem() függvény segítségével felszabadítja az összes azt követő elemet, így kihasználva a sorrendet, és növelve a program hatékonyságát.

A compare() függvény a lista első elemétől indulva összehasonlítja a user elvárásaival(miután a serach_for_bad függvény elvégezte dolgát) az elemet, és ha nem felel meg teljesen a user elvárásainak, akkor a pop_etterem() függvény segítségével eltávolítja a listából.

Miután az összes fent említett függvény rendeltetés szerűen elvégezte feladatát, a lista csak olyan elemekből áll, amelyek megfelelnek a user elvárásainak.

A programnak szüksége van egy kiíró függvényre is, mely egy etterem elemet fog a stdout-ra írni. A print_etterem() fogja ezt megvalósítani, mely kiírja az étterem nevét, a konyha nemzetiségét, az étterem címét, hogy hány fős az ő elvárásának felülről közelített utolsó még megfelelő asztal, valamint, hogy légvonalban hány méterre van a usertől az étterem. Miután az elemet kiírta, a pop_etterem() függvénnyel eltávolítja a listából és meghívja önmagát újra egészen addig, amíg az elemek el nem fogynak. Ekkorra már semmilyen további tevékenység nem végzendő el, így a main függvény leállítja a programot.

A távolság megállapítására egy tavolsag() függvényt kell implementálni, mely kiszámítja az étterem és a user közötti távolságot.

A main függvény

A main függvénynek csak a segédfüggvények hívásában, valamint a pointerek kezelésében van feladata. Nem kell létrehoznia pointereken kívül semmit. A program lefolyása a következő: Létrejön a láncolt lista az ettermek.txt és az asztalok.txt fájlok beolvasása után, majd létrejön a user.txt-ből a struktúra elem. Ezek után a lista elemek addig fogynak, amíg csak a megfelelőek nem maradnak, így a print_etterem() első meghívása megtörténhet, majd a program közvetlenül le is állhat.