Programozói dokumentáció

NHF - Könyvtár

A projekt feladata egy Könyvtár adatbázis létrehozása és kezelése. Felépítése a következő:

Struktúra

Egy könyvet a képen látható struktúrában tárolok el. Egy könyvről lehet tudni: szerzője, címe, kiadási éve, témája, hogy kinél van (ha ennek értéke “szabad”, akkor nincs senkinél, nem lehet üres). Valamint a struktúra még tartalmaz egy Konyv* kovetkezo pointert is, ez a program alapját szolgálja. A fájlból beolvasott könyveket a beolvasó függvény dinamikusan foglalt láncolt listába építi, és ennek segítségével kezeli. A +1-ek a lezáró ’\0’ miatt vannak jelezve.

```
typedef struct Konyv{
    char szerzo[30+1];
    char cim[50+1];
    int kiadasiev;
    char tema[20+1];
    char kinelvan[30+1];
    struct Konyv* kovetkezo;
}Konyv;
```

Fajlkezeles.c

Ebben a modulban 3 függvény található, amik csak a fájlból beolvasással, fájlkezeléssel, fájlba írással foglalkoznak.

1. int fajlban_sorokat_megszamol(FILE* fp)

A függvény feladata a paraméterként kapott fájl (FILE*-rel veszi át) sorait megszámolni (megnyitnia már nem kell) és visszatérni vele (egész szám). Működése: létrehoz egy int c-t (char c helyett), majd fgetc() -cel addig megy, amíg EOF-t nem olvas, és addig, ha ‘\n’-t olvasott be, hozzáad egyet az int sorok_szama –hoz. A végén rewind(fp)-vel visszaállítja a fájlpointert a fájl elejére, hogy a többi fájlkezelő függvénynél a helyes pozícióban legyen a FILE* fp. Segédfüggvény, amely a listaba_beolvas() függvénynél segít a listába való beolvasásnál.

2. Konyv* listaba_beolvas()

Ez a függvény látja el a függvényből láncolt listába beolvasás feladatát. Miután elvégezte visszatér a lista elejére mutató pointerrel. Első lépésként megnyitja a Konyvtar.txt-t, olvasás módra. Perror-ral kezeli, ha nem sikerült a fájl megnyitása. Létrehoz két Konyv* pointert, az első és utolsó elemre, ezeket eleinte NULL-ra állítva. Ezután meghívja a sor számoló függvényt, hogy tudja hányszor kell lefutnia magának a foglalásnak és beolvasásnak. A ciklus (amely annyiszor fut le, ahány sor van a fájlban, mivel egy sor = egy könyv) minden lefutásakor lefoglal helyet egy új Konyv* elemnek, majd fscanf(fp, "%[^\t]\t%[^\t]\t%d\t%[^\t]\t%[^\n]\n", …) módszerrel olvas be a fájlból egy elemet. A listába a következő módon helyezi el: (még cikluson belül vagyunk) az aktuális elem kovetkezo pointere NULL lesz, majd megvizsgálja, hogy van-e már legutobbi_elem, ha nincs (tehát ez az első), akkor ez az aktuális könyv lesz a legelső elem, ha már van, akkor a legutobbi_elem->kovetkezo -vé teszi ezt az elemet. Majd végül beállítja, hogy az újonnan beolvasott elem lesz a legutóbbi elem is egyben. Ezután bezárja a fájlt és visszatér a láncolt lista első elemével. Ez a függvény minden elinduláskor magától lefut. Ha lefutott rendesen, a szabványos kimeneten tudatja a felhasználóval a program elindulásakor.

3. void listat_lement_es_felszabadit(Konyv* eleje)

Paraméterként kapja a láncolt lista legelejére mutató pointert, hogy egyrészt le tudja menteni fájlba, másrészt meghívhassa a felszabadító függvényt. Ez a függvény a program minden (helyes) bezárásakor lefut. Hibakezelés: ha az eleje pointer NULL (tehát üres a lista), figyelmezteti a felhasználót, hogy nem mentett le egy könyvet sem. Megnyitja a Konyvtar.txt-t “w”, azaz írás módra (perror-ral kezeli, ha meghiúsult a fájlmegnyitás), majd fprintf(fp, "%s\t%s\t%d\t%s\t%s\n", …) módszerrel írja bele a fájlba a könyveket. Egyszerű láncolt listán végig menő for ciklussal határozom meg hányszor kell fprintf-elni. Amikor kész van a for ciklus a fájlba írással, a függvény meghívja a felszabaditó függvényt (lancoltlistakezeles.c -ben található) és bezárja a fájlt. Majd a szabványos kimeneten tájékoztatja a felhasználót, hogy helyesen lefutott.

Fajlkezeles.h

fajlkezeles.h

```
1   #ifndef FAJLKEZELES_H_INCLUDED
2   #define FAJLKEZELES_H_INCLUDED
3
4
5   int fajlban_sorokat_megszamol(FILE* fp);
6   Konyv* listaba_beolvas();
7   void listat_lement_es_felszabadit(Konyv* eleje);
8
9   #endif // FAJLKEZELES_H_INCLUDED
10
```

Lancoltlistakezeles.c

Ebben a modulban 6 függvény található, melyek mind láncolt listát kezelnek, például törlés, felszabadítás, módosítás hozzáadás stb.

1. Konyv* konyv_beolvas()

A függvény azt csinálja, hogy lefoglal egy új Konyv* változót (malloc), majd a felhasználótól a szabványos bemenetről beolvassa (mindet enterig) az adatait. Csak olyan könyvet vehet fel az adatbázis, amelyiknek minden adatát ismerjük, nem lehet szabadon hagyni egyik jellemzőt sem. Ha nincsen senkinél a könyv jelenleg (senki sem kölcsönözte ki jelenleg), akkor “szabad”-ot kell beírni a kinél van mezőben. Ha kész a beolvasással, visszatér az elemre mutató pointerrel. Fflush(stdin) -re azért van szükség, hogy “kiürítse” a bemeneti fájlt, a benne maradt enterek miatt, hiszen ezek zavarnák az újabb enterig való beolvasást. Ezt a függvényt a többi láncolt lista kezelő függvény hívja meg (már amelyiknél szükséges).

2. void felszabadit(Konyv* eleje)

Egyszerű, előadáson is bemutatott függvény. Ciklussal megy végig a paraméterként kapott első pointerrel a láncolt listán. Egy segédváltozót használt (Konyv* temp = eleje), a ciklus feltétele: while(temp), tehát ameddig érvényes elemünk van. A cikluson belül eltárolja egy változóban az aktuális elem következőjét, felszabadítja az aktuálist, majd az aktuálist átírja az ő maga következőjére. A lefutása végén tájékoztatja a felhasználót a feladata helyes végrehajtásáról. A függvényt a listába lementő függvény hívja meg, a program minden egyes (szabályos) bezárásakor.

3. void listat_kiir(Konyv* eleje)

Ugyanúgy, nem egy bonyolult működésű függvény, a felszabadítóhoz hasonló módon végigmegy a láncolt listán, paraméterként megkapta ennek a legelső elemét. A ciklus futása közben kiírja a szabványos kimenetre az aktuális elem minden adatát, valamint sorszámát. A függvényt a főmenüből a felhasználó tudja meghívni.

4. void hozza_ad(Konyv* eleje)

Ennek feladata a láncolt lista végére egy új könyv fűzése. Paraméterként megkapja a láncolt lista legelső elemét. Konyv_beolvas függvény segítségével először beolvassa az új könyv adatait. Beállítja az új elem *kovetkezo pointerét NULL-ra (hiszen ez lesz a legvégén a listának), ezután megvizsgálja, hogy NULL-e az eleje pointer (avagy üres-e a lista), ha igen, akkor akkor ez az új elem lesz az eleje pointer. Ha nem, akkor ciklus segítségével megkeresi a legutóbbi elemet, majd annak a *kovetkezo-jét átirányítja az imént beolvasott új elemre. Ezután kiírja a szabványos kimenetre, hogy sikeres volt a hozzáadás. A függvényt a főmenüből a felhasználó tudja meghívni.

5. void torles(Konyv* eleje, char* mit)

A függvény feladata egy elem törlése a láncolt listából. Paraméterként kapja a lista elejére mutató pointerre mutató pointert, és a törlendő könyv címét. A függvény azért vesz át Konyv** eleje-t, mivel hogyha az első elemet akarom kitörölni, valahogy át kell irányítani az addigi eleje pointert az újra. Első lépésként létrehoz két változót, egy lemaradó pointert és egy temp pointert, előbbit NULL-ra, utóbbit az elejére állítja. Ezután ciklussal végigmegy a listán (fentebb is bemutatott módon) és megáll a keresett elemnél (strcmp-vel vizsgálom meg a kapott keresett címet és az aktuális könyv címét). A lemaradó pointer a törlendő elem megelőzője. Habár itt beleírtam egy temp == NULL ellenõrzõt is, a valóságban ez nem történhet meg, mivel eleve a kereső függvény hívja meg a törlést, így mindenképpen érvényes adatot kéne törölni. Majd megvizsgálom a lemaradó elemet, ha ez NULL pointer, vagyis, ha az első elemet törlöm, átirányítom az új elejét a törlendő következőjére, és felszabadítom a törlendő elemet. Hogyha pedig a lista közepéből törlök, a lemaradó következőjét egyenlővé teszem a törlendő következőjével, majd felszabadítom a törlendő elemet. Lefutás végén a függvény közli a helyes lefutást a felhasználóval. Ezt a függvényt minden keresés függvény meghívhatja a lefutása végén, felajánlja a felhasználónak a törlés lehetőségét.

6. void modositas(Konyv* mit)

Ez a függvény látja el egy lista elem módosításának feladatát. Első lépésként konyv_beolvas() függvénnyel beolvassa a paraméterként kapott elem új adatait. Második lépésként strcpy-vel átmásolja az adatokat a már meglévő könyvbe. Utolsó lépése pedig, hogy felszabadítsa a segédváltozót, amelybe csak beolvasta az új adatokat. Végül megmondja a felhasználónak, hogy sikeres volt a lefutása, megtörtént a módosítás. Ezt a függvényt a kereső függvények hívhatják meg a lefutásuk után, felajánlják a felhasználónak a módosítás lehetőségét.

Lancoltlistakezeles.h

lancoltlistakezeles.h

```h
#ifndef LANCOLTLISTAKEZELES_H_INCLUDED
#define LANCOLTLISTAKEZELES_H_INCLUDED

Konyv* konyv_beolvas();
void felszabadit(Konyv* eleje);
void listat_kiir(Konyv* eleje);
void hozza_ad(Konyv* eleje);
void torles(Konyv* eleje, char* mit);
void modositas(Konyv* mit);

#endif // LANCOLTLISTAKEZELES_H_INCLUDED

```

Keresesek.c

Mint ahogy a neve is sugallja, ebben a modulban találhatóak a keresések lebonyolításával foglalkozó függvények. Minden ebben található függvény közvetlenül a főmenüből, a felhasználó által hívható, kivéve a legelső, a lehetosegek_kiir() függvény, amely csak a főmenü megjelenítését segíti (de ez a függvény is tulajdonképpen segít a felhasználónak, hogy megtalálja melyik opciót keresi, ezért illik ide a kereső függvényekhez).

Az összes kereső függvény ugyanazon módszert alkalmazza, ezért összefoglalom itt, hogy hogyan is működnek. Tehát hogyha azt írom: “a függvény …", az összes keresesek.c modulon belül található függvényre gondolok (kivéve a lehetőségek_kiír() függvényre).

Kereső függvények működése: Egy bool változóban tárolja el a függvény, hogy volt-e találat vagy sem. Először beolvassa a felhasználótól, hogy melyik nevet/címet/kiadásiévet/témát/olvasót keresi, hogyha sztringet olvas be (nem a kiadásiévet), akkor annak mallocol kellő mennyiségű helyet (amit a keresés végén fel is szabadít). Ezután végigmegy a listán, amíg strstr-rel nem talál egyezést a listában található könyvek és a beolvasott kulcsszó között. Ha megvan, a találtat átrakja true-ra, és kiírja a keresett adatait a képernyőre. A találatokat egymás után számozva írja ki. Minden találat után a függvény felajánlja a felhasználónak, hogy módosítsa, vagy törölje a kiírt könyvet. Ha a találtak közül valamelyiket sikeresen törölte/módosította a felhasználó, a kereső függvény folytatja munkáját és továbbmegy a találatok kiírásával. Ha a bool talalt false maradt a ciklus lefutása után, akkor nem volt találat és a függvény közli ezt a felhasználóval.

1. **void lehetosegek_kiir();**

Az egész projekt legrövidebb függvénye, feladata összesen annyi, hogy a főmenü opcióit kiírja, nem tér vissza semmivel és nem is kap pataméterként semmit.

2. **void kereses_szerzo(Konyv* eleje);**
3. **void kereses_cim(Konyv* eleje);**
4. **void kereses_kiadasiev(Konyv* eleje);**
5. **void kereses_tema(Konyv* eleje);**
6. **void olvaso_konyvei(Konyv* eleje);**

Keresesek.h

keresesek.h

```c
#ifndef KERESESEK_H_INCLUDED
#define KERESESEK_H_INCLUDED

void lehetosegek_kiir();
void kereses_szerzo(Konyv* eleje);
void kereses_cim(Konyv* eleje);
void kereses_kiadasiev(Konyv* eleje);
void kereses_tema(Konyv* eleje);
void olvaso_konyvei(Konyv* eleje);

#endif // KERESESEK_H_INCLUDED

```

Main.c

A main-ben a főmenü található. A main hívja meg elinduláskor a listába beolvasó függvényt, bezáráskor pedig a listába lementő és felszabadító függvényt. Meghívja a lehetosegek_kiir() függvényt egy do while-lal, majd beolvassa a felhasználó választását (számokkal tud választani a felhasználó). Switch case hívja meg ezután a választott függvényeket.