Programozói Dokumentáció – Receptkönyv

A projekt felépítése és környezet

Fejlesztési környezet

A program szabványos **C99** nyelven készült. A fejlesztés során a **CLion** integrált fejlesztőkörnyezetet és a **MinGW/GCC** fordítót használtam.

Modulok szerkezete

A projekt moduláris felépítésű, a különböző felelősségi körök külön `.c` és `.h` fájlokba lettek szervezve. A körkörös hivatkozások elkerülése érdekében a típusdefiníciókat egy közös header fájl tartalmazza.

- **`main.c`**: A belépési pont. Inicializálja a véletlenszám-generátort (`srand`), betölti az adatbázisokat, meghívja a menüvezérlőt, majd kilépéskor gondoskodik a mentésről és a teljes memóriafelszabadításról.
- **`adatszerkezet.h`**: Csak a `struct` definíciókat és `typedef`-eket tartalmazza. Ezt minden modul include-olja, így mindenki ismeri az adattípusokat.
- **`menu.c / .h`**: A felhasználói felület vezérlése. Kezeli a főmenüt, a bemenetek validálását, és hívja a megfelelő logikai modulokat. Pointerek címeivel (**) dolgozik, hogy módosítani tudja a `main`-ben lévő listafejeket.
- **`receptek.c / .h`**: Tartalmazza a receptek létrehozását, módosítását, törlését, valamint a speciális keresési algoritmusokat.
- **`osszetevok.c / .h`**: A globális összetevő-lista kezelése (CRUD műveletek).
- **`fajlkezelo.c / .h`**: Az adatok tárolásáért felel. Szöveges fájlokból olvas és ír.

Adatszerkezetek dokumentációja

A program alapvető adatszerkezete az **egyszeresen láncolt lista**.

Miért láncolt lista?

A receptkönyv jellege miatt nem tudjuk előre megbecsülni, hány receptet vagy hány összetevőt fog rögzíteni a felhasználó.

- A **statikus tömb** pazarló lenne (ha túl nagyot foglalunk) vagy korlátozó (ha betelik).
- A **dinamikus tömb** átméretezése (`realloc`) költséges lehet beszúráskor/törléskor.
- Az **egyszeresen láncolt lista** dinamikusan nő, és a beszúrás/törlés műveletek pointer-átkötéssel hatékonyan megoldhatók.

A "Lista a listában" szerkezet

A program egy beágyazott láncolt lista modellt használ.

1. **Fő lista (`ReceptCsomopont`):** Ez tárolja a recepteket. Minden elem tartalmaz egy `Recept` adatot és egy kovetkezo pointert.
2. **Belső lista (`ReceptHozzavalok`):** Minden egyes recept tartalmaz egy pointert (`eleje`), ami egy másik láncolt lista feje. Ez a belső lista tárolja az adott ételhez tartozó hozzávalókat.

Definíciók (`adatszerkezet.h`):

```
typedef struct ReceptHozzavalok {
    char *osszetevo_nev;       // Dinamikusan foglalt string
    int mennyiseg;
    char *mertekegyseg;        // Dinamikusan foglalt string
    struct ReceptHozzavalok *kovetkezo;
} ReceptHozzavalok;

typedef struct Recept {
    char *etel_nev;            // Dinamikusan foglalt string
    char *elkeszites;          // Dinamikusan foglalt string
    ReceptHozzavalok *eleje;   // A BELSŐ lista feje
} Recept;

typedef struct ReceptCsomopont {
    struct Recept adat;
    struct ReceptCsomopont *kovetkezo; // A KÜLSŐ lista láncolása
} ReceptCsomopont;
```

Fájlkezelés és Formátum

A fájlba írás és olvasás szöveges formátumban történik. Mivel a receptek változó hosszúságúak (tetszőleges számú hozzávaló), a fájlban egy elválasztó karakter (#) jelzi egy recept végét.

A `receptek.txt` szerkezete:

```
Recept Neve
Elkészítés módja (egy sorban)
Hozzávaló1 Neve
Mennyiség
Mértékegység
Hozzávaló2 Neve...
#
Következő Recept Neve...
```

A beolvasó algoritmus (`receptek_beolvasasa_fajlbol`) soronként olvas (`fgets`), és állapotgépszerűen működik: amíg nem talál # jelet vagy fájl végét, addig az olvasott sorokat új hozzávalóként fűzi az aktuális recepthez.

4. Fontosabb Függvények dokumentációja

Modul: `receptek.c`

- **`recept_keres_tobb_osszetevo`**
  - *Feladat:* Kezeli a több összetevős keresés teljes folyamatát (bekérés, szűrés, listázás, választás).
  - *Paraméterek:* A receptek és az összetevők listájának feje.
  - *Visszatérés:* `ReceptCsomopont*` (a kiválasztott recept pointere) vagy `NULL`.
  - *Megjegyzés:* Dinamikus segédtömböket használ (`malloc`/`realloc`), amiket a végén felszabadít.
- **`recept_modositasa`**
  - *Feladat:* Lehetővé teszi egy meglévő recept adatainak (név, leírás) és hozzávalóinak szerkesztését.
  - *Működés:* Nem új receptet hoz létre, hanem a meglévő struktúrában cseréli a pointereket (régi `free`, új `malloc`).
- **`recept_torlese`**
  - *Feladat:* Kifűz egy elemet a láncolt listából.
  - *Paraméterek:* A törlendő elem pointere és a lista feje.

- *Működés:* Kezeli a speciális esetet, ha az első elemet kell törölni. Ha nem az elsőt, "lemaradó" pointerrel keresi meg az előző elemet a lánc átkötéséhez. Gondoskodik a belső lista (hozzávalók) felszabadításáról is (`recept_torlese_segedfuggveny`).

- **`recept_keres_egy_osszetevo`**
  - **Feladat:** Lehetővé teszi a receptek szűrését egyetlen megadott alapanyag alapján. Kilistázza az összes olyan receptet, amely tartalmazza a keresett összetevőt, majd a felhasználó választása alapján visszaadja a recept pointerét.
  - **Paraméterek:**
    - `ReceptCsomopont *receptek_eleje`: A receptek láncolt listájának kezdete (ebben keresünk).
    - `Osszetevok *osszetevok_eleje`: A globális összetevő lista (a lista megjelenítéséhez a felhasználónak).
  - **Visszatérés:** `ReceptCsomopont*` (a kiválasztott receptre mutató pointer) vagy `NULL` (ha nem volt találat vagy a felhasználó kilépett).
  - **Működés:** Kétmenetes algoritmus.
    i. **Listázás:** Végigmegy a recepteken, és minden receptnél belső ciklussal ellenőrzi a hozzávalókat. Ha egyezést talál (`strcmp`), kiírja a receptet és `break`-kel ugrik a következőre (elkerülve a többszöri kiírást).
    ii. **Kiválasztás:** A felhasználó által megadott sorszám alapján újra végigmegy a listán ugyanazzal a feltétellel, hogy megszerezze a memóriacímet.

- **`recept_kereses_random`**
  - **Feladat:** Véletlenszerűen kiválaszt és megjelenít néhány receptet a listából, majd lehetőséget ad a felhasználónak, hogy válasszon közülük.
  - **Paraméterek:**
    - `ReceptCsomopont *receptek_eleje`: A receptek listája.
  - **Visszatérés:** `ReceptCsomopont*` (a kiválasztott recept pointere).
  - **Működés:**
    iii. Megszámolja a lista hosszát.
    iv. Véletlenszám-generátorral (`rand()`) meghatározza, hány receptet ajánljon fel.
    v. Egy ciklusban véletlen indexeket generál és odaléptet egy pointert.
  b. **Deduplikáció:** Egy dinamikus segédtömbben (`kisorsolt_receptek`) tárolja a már kiválasztott pointereket, és ellenőrzi, hogy ne sorsolja ki ugyanazt a receptet kétszer.
  c. Végül a segédtömb alapján adja vissza a felhasználó által választott elemet.

- **`recept_kereses_nev_alapjan`**

- **Feladat:** Szabadszavas keresést valósít meg a receptek neveiben.
- **Paraméterek:**
  - `ReceptCsomopont *receptek_eleje`: A keresési tér.
- **Visszatérés:** `ReceptCsomopont*` (találat esetén) vagy `NULL`.
- **Működés:** A szabványos `strstr` függvényt használja, így részleges egyezéseket is talál (pl. "leves" keresőszó megtalálja a "Húsleves"-t is). Ez is kétmenetes algoritmus: először megszámolja és listázza a találatokat, majd a választott sorszám alapján visszakeresi a pointert. Gondoskodik a dinamikusan beolvasott keresőszó felszabadításáról is.

- **`interakcio_uj_recept`**
  - **Feladat:** Magas szintű vezérlőfüggvény, amely koordinálja egy teljesen új recept létrehozását, beleértve a névbekérést, validációt és a hozzávalók feltöltését.
  - **Paraméterek:**
    - `ReceptCsomopont *receptek_eleje`: A meglévő receptlista (az új elem befűzéséhez és duplikáció ellenőrzéséhez).
    - `Osszetevok *osszetevok_eleje`: Az összetevők listája (a hozzávalók kiválasztásához).
  - **Visszatérés:** A receptlista (esetlegesen megváltozott) eleje pointer.
  - **Működés:**
    i. **Validáció:** Ciklusban kéri be a nevet, biztosítva, hogy ne legyen üres vagy csak szám.
    ii. **Ellenőrzés:** Megvizsgálja (`van_e_ilyen_recept`), hogy létezik-e már ilyen nevű étel. Ha igen, hibaüzenettel visszatér.
    iii. **Létrehozás:** Meghívja a `recept_letrehozasa` függvényt (memóriafoglalás, láncolás).
    iv. **Feltöltés:** Átadja a vezérlést a `recept_hozzavalok_beolvasasa` függvénynek a részletek kitöltéséhez.**`recept_hozzavalok_beolvasasa`**

3. **Feladat:** Ez a függvény felelős egy éppen készülő recept hozzávalóinak összeállításáért. Egy ciklusban fut, amíg a felhasználó be nem fejezi a rögzítést, lehetőséget adva meglévő alapanyag választására vagy teljesen új felvételére.
4. **Paraméterek:**
   a. `ReceptCsomopont *aktualis_recept`: Az a recept (csomópont), amelyhez éppen hozzávalókat adunk.
   b. `Osszetevok *elerheto_osszetevok`: A globális összetevő lista (a választék megjelenítéséhez).
   c. `ReceptHozzavalok *hozzavalok_lista`: A recept belső hozzávaló-listájának aktuális feje (ehhez fűzzük az újakat).
5. **Visszatérés:** `ReceptCsomopont*` (Az aktuális recept pointerével tér vissza, hogy a hívó oldalon frissíteni lehessen a láncolást).

6. **Működés:**
    a. Listázza az összes elérhető összetevőt a globális raktárból.
    b. Menüt kínál fel:
        i. **1. Meglévőből választás:** Sorszám alapján bekéri a választott alapanyagot. Ellenőrzi (`hozzavalo_szerepel_e`), hogy ez az összetevő szerepel-e már az adott receptben (duplikáció elkerülése). Ha nem, lefoglalja a memóriát az új elemnek, és átmásolja az adatokat (név).
        ii. **2. Új felvétele:** Bekéri az új nevet (validációval: nem lehet üres vagy szám). Ha még nem létezik a globális listában, oda is felveszi (`interakcio_uj_osszetevo`), majd a recepthez is hozzáadja.
        iii. **3. Kilépés:** Befejezi a hozzávalók felvételét.
    c. Minden új elemnél bekéri a mennyiséget és a mértékegységet is.

Modul: `osszetevok.c`

- **`osszetevo_torlese`**
    - *Feladat:* Töröl egy alapanyagot a globális listából.
    - *Megjegyzés:* Csak akkor hívódik meg, ha az ellenőrző függvény (`osszetevo_hasznalatban_e`) megerősítette, hogy az összetevő nincs használatban egyetlen receptben sem.
- **`interakcio_uj_osszetevo`**
    - **Feladat:** Ez a függvény végzi egy új alapanyag felvételét a globális összetevő-adatbázisba. Gondoskodik a duplikációk elkerüléséről (már létező elem nem vehető fel újra).
    - **Paraméterek:**
        - `Osszetevok *osszetevok_eleje`: A jelenlegi összetevő lista feje (ebben keresünk, és ehhez fűzzük az újat).
        - `char *beolvasott_osszetevo_nev`: Az új összetevő neve, amelyet a hívó függvény (pl. a menü) már beolvasott és validált.
    - **Visszatérés:** `Osszetevok*` (A lista elejére mutató pointer. Ez azért fontos, mert ha a lista üres volt, vagy az új elem az elejére kerül, a listafej megváltozik).
    - **Működés:**
        - Meghívja az `osszetevo_letezik_e` segédfüggvényt, hogy ellenőrizze, szerepel-e már ez a név a listában.
        - **Ha nem létezik:**
            - Létrehoz egy új listaelem-csomópontot (`osszetevo_letrehozasa`).
            - Belemásolja a nevet és beállítja a pointereket (`osszetevo_feltoltes`).

        - **Ha létezik:**
            - Hibaüzenetet ír ki a konzolra ("Már létezik ilyen összetevő"), és nem módosít a listán.
        - Visszatér az (esetlegesen frissült) listafejjel.

Modul: `menu.c`

- **`menu`**
    - *Feladat:* A főmenü végtelen ciklusa.
    - *Paraméterek:* `Osszetevok **osszetevok_ptre`, `ReceptCsomopont **receptek_ptr`.
    - *Fontos:* Dupla pointereket vár, hogy a meghívott függvények (pl. törlés, új felvétel) által módosított listafejeket vissza tudja adni a `main` függvénynek.

Modul: `fajlkezelo.c`

- **`fajl_sor_beolvasas`**
    - **Feladat:** Egyetlen sor beolvasása szöveges fájlból tetszőleges hosszúságú sor esetén. Ez a függvény hidalja át a C nyelv `scanf` függvényének korlátait a dinamikus memóriakezelés segítségével.
    - **Paraméterek:**
        - `FILE *fajlmutato`: A megnyitott fájlra mutató pointer, ahonnan olvasni kell.
    - **Visszatérés:** `char*` (a dinamikusan lefoglalt sztring kezdőcíme) vagy `NULL` (ha fájl vége vagy hiba történt).
    - **Működés:** Karakterenként (`fgetc`) olvassa a fájlt egy ciklusban. Minden beolvasott karakter után dinamikusan növeli a memóriaterületet (`realloc`), amíg sorvége jelet (`\n`) vagy fájl végét (`EOF`) nem talál. A sor végére automatikusan elhelyezi a lezáró nullát (`\0`).
- **`osszetevok_beolvasasa_fajlbol`**
    - **Feladat:** A program indításakor betölti a globális alapanyag-listát a szöveges adatbázisból (`osszetevok.txt`).
    - **Paraméterek:**
        - `char *fajlnev`: A betöltendő fájl neve (stringként).
    - **Visszatérés:** `Osszetevok*` (A felépített láncolt lista feje).
    - **Működés:**
        - Megnyitja a fájlt olvasásra.
        - Soronként olvassa a fájlt (`fgets`).
        - Minden sorhoz létrehoz egy új `Osszetevok` csomópontot (`malloc`).

        - Lefoglalja a memóriát a névnek, és átmásolja a fájlból olvasott szöveget (a sorvégi entert levágva).
        - Az új elemet a lista elejére fűzi be (verem/LIFO elv), mivel ez a leggyorsabb művelet.
        - Bezárja a fájlt és visszaadja a listafejet.
- **`receptek_beolvasasa_fajlbol`**
    - **Feladat:** Betölti a recepteket és azok hozzávalóit a strukturált szöveges fájlból (`receptek.txt`), felépítve a "lista a listában" memóriaszerkezetet.
    - **Paraméterek:**
        - `char *fajlnev`: A receptek fájl neve.
    - **Visszatérés:** `ReceptCsomopont*` (A kész receptlista feje).
    - **Működés:** Kettős ciklust alkalmaz a fájl speciális szerkezete miatt.
        - **Külső ciklus:** A fájl végéig fut. Beolvassa a recept nevét és elkészítési módját a `fajl_sor_beolvasas` segítségével. Létrehoz egy új recept csomópontot.
        - **Belső ciklus:** A hozzávalókat olvassa be sorban.
            - Figyeli a # elválasztó karaktert: ha ezt olvassa be névként, az a recept végét jelzi -> kilép a belső ciklusból.
            - Ha valódi hozzávaló, akkor beolvassa a mennyiséget (amit szövegből számmá konvertál `atoi`-val) és a mértékegységet.
            - Az új hozzávalót befűzi az aktuális recept belső listájába (`adat.eleje`).
        - A teljesen beolvasott receptet hozzáfűzi a fő receptlistához.