Választó program – Programozói dokumentáció

Tartalomjegyzèk

Projektleírás

Ez a projekt egy egyszerű szavazórendszer C nyelven megvalósítva. A program különböző modulokra bontva kezeli a szavazók adatait, a fájlműveleteket és a segédfüggvényeket. A memóriakezelés hibáinak felderítéséhez egy debugmalloc modul is használatban van.

Fájlstruktúra

```
Választás
└── Választás
    ├── files          (Adattároló mappa)
    │   ├── allapot.txt
    │   ├── eredmeny.txt
    │   ├── jeloltek.txt
    │   ├── lezaras.txt
    │   ├── szavazok.txt
    │   ├── uj_jelolt.txt
    │   └── uzenetek_adminnak.txt
    │
    ├── admin_fuggvenyek.c   (adminfuggvenyek)
    ├── debugmalloc.h        (memóriakezelés)
    ├── fajlmuveletek.c      (fájlkezelés)
    ├── main.c               (Főprogram)
    ├── menu_fuggvenyek.c    (menürendszer)
    ├── segedfuggvenyek.c    (segédfüggvények)
    └── strukturak.h          (adatstruktúrák)
```

Adattároló mappák(files)

Minden fájl tartalomnál az értékek ‘-‘ -el, szóköz nélkül vannak párosítva, új értékek esetén másik sorba kezdődnek, kivéve az allapot.txt, lezaras.txt esetben lentebb részletezve.

---

allapot.txt (Rendszerállapot)

- **Leírás:** A választás státuszát jelző kapcsoló.
- **Tartalom:** Egyetlen számjegy (0 vagy 1).
- **Jelentés:**
  - 0: NYITOTT (lehet szavazni)
  - 1: ZÁRT (a szavazás lezárult)

---

eredmeny.txt (Szavazatok összesítése)

- **Leírás:** A jelöltekre leadott szavazatok aktuális számát tárolja.
- **Formátum:** Név-Párt-Szavazatszám
- **Példa:** Dobrev Klara-Demokratikus Koalicio-972

---

jeloltek.txt (Hivatalos jelöltek listája)

- **Leírás:** A választáson induló, jóváhagyott jelölteket tartalmazza.
- **Formátum:** Név-Párt
- **Példa:** Orban Viktor-Fidesz-KDNP

---

lezaras.txt (Lezárás időpontja)

- **Leírás:** A szavazás lezárásának pontos időpontját rögzíti, egyetlen datum.
- **Formátum:** ÉÉÉÉ-HH-NN ÓÓ:PP:MP
- **Példa:** 2025-11-24 17:49:33

---

szavazok.txt (Szavazók adatbázisa)

- **Leírás:** A regisztrált szavazók adatait tartalmazza egyszerűsített, soronkénti formátumban.
- **Formátum:** Név-Jelszó-Jelölt-Párt
- **Példa:** Kovacs Peter-Kovacs123-Dobrev Klara-Demokratikus Koalicio

---

uj_jelolt.txt (Jelöltnek jelentkezők várólistája)

- **Leírás:** Azok a felhasználók, akik jelöltnek jelentkeztek, de az admin még nem hagyta jóvá őket.
- **Formátum:** Név-Párt
- **Példa:** Toth Eszter-Fuggetlen

---

uzenetek_adminnak.txt (Felhasználói üzenetek)

- **Leírás:** A felhasználók által az adminisztrátornak küldött üzenetek naplója.
- **Formátum:** Feladó-Üzenet szövege
- **Példa:** Kovacs Peter-Veletlenul elirtam a jelszavam...

Struktúra felépítése(strukturak.h)

Csak azokhoz az adatokhoz (Szavazó, Jelölt, Eredmény) készült struktúra, amelyekből listát (tömböt) épít a program a memóriában, hogy később dolgozni tudjon velük (pl. keresés, sorbarendezés, módosítás).

---

1. Szavazó struktúra

A szavazásra jogosult felhasználók adatait tárolja. Egy-egy ilyen rekord felel meg a `szavazok.txt`-ben tárolt blokkoknak.

- **Definíció:** typedef struct Szavazo { ... } Szavazo;
- **Mezők:**
  - char nev[128]: A szavazó teljes neve. Ez azonosítja a felhasználót a bejelentkezésnél.
  - char jelszo[64]: A felhasználó jelszava.
  - char jelolt[128]: Annak a jelöltnek a neve, akire a szavazó a voksát leadta.
  - char part[128]: A választott jelölt pártjának neve.

2. Jelölt struktúra

A választáson induló jelöltek (vagy a jelentkezők) alapvető adatait írja le. Ezt használja a program a jeloltek.txt és az uj_jelolt.txt fájlok kezelésekor.

- **Definíció:** typedef struct Jelolt { ... } Jelolt;
- **Mezők:**
  - char nev[128]: A jelölt neve.
  - char part[128]: A jelöltet támogató párt.

3. Eredmeny struktúra

A szavazás jelenlegi állását tárolja memóriában. A eredmeny.txt fájl beolvasásakor ebbe a struktúrába töltődnek az adatok.

- **Definíció:** typedef struct Eredmeny { ... } Eredmeny;
- **Mezők:**
  - char nev[128]: A jelölt neve.
  - char part[128]: A jelölt pártja.
  - int szavazat: A jelöltre eddig leadott érvényes szavazatok száma.

4. Jelölt Lista struktúra (Láncolt lista elem)

A várólista kezelésére szolgáló egyszeresen láncolt lista eleme.

- **Definíció:**

```
typedef struct JeloltLista {
    Jelolt adat;
    struct JeloltLista *kov;
} JeloltLista;
```

- **Szerepe:** Lehetővé teszi a jelentkezők hatékony beszúrását és törlését a lista bármely pontjáról anélkül, hogy a memóriában adatokat kellene másolni (mint a tömböknél).

Egyéb adattípusok txt-ből

1. Állapot (allapot.txt)

- **Miért nem kell struktúra?** Ez az adat egyetlen egyszerű szám (`0` vagy `1`).
- **Hogyan kezeli a kód?** Egy sima `int` (egész szám) változóban tárolja (`int allapot`). Felesleges lenne ehhez egy külön struktúrát létrehozni, mivel nem több adatból áll össze, hanem csak egy elemi érték.
    - *Kód hivatkozás:* `beolvas_allapot(..., int *allapot_out).`

2. Lezárás időpontja (lezaras.txt)

- **Miért nem kell struktúra?** A program ezt szövegként (`string`) olvassa be és írja ki.
- **Hogyan kezeli a kód?** Egy egyszerű karaktertömböt (`char lez[64]`) használ a tárolására. Bár a háttérben a C nyelv `struct tm`-et használ az idő kiszámolására, a fájlkezelés szempontjából ez csak egy egyszerű szöveges adat, nem saját definiált típus.
    - *Kód hivatkozás:* `beolvas_lezaras_idopont(..., char *buf, ...).`

3. Üzenetek az adminnak (uzenetek_adminnak.txt)

- **Miért nem kell struktúra?** A program **nem tárolja el** ezeket az üzeneteket a memóriában.
- **Hogyan kezeli a kód?**
    - Amikor az admin megtekinti az üzeneteket, a program megnyitja a fájlt, soronként beolvassa, kiírja a képernyőre (`printf`), majd azonnal el is "felejti" az adatot, és jön a következő sor.
    - Mivel sosem kell egyszerre az összes üzenetet memóriában tartani (pl. rendezéshez vagy kereséshez), nincs szükség `Uzenet` struktúrára sem.
    - *Kód hivatkozás:* `beolvas_uzenetek_adminnak` függvény csak `printf`-el.

Technikai megvalósítások és algoritmusok

1. Dinamikus memóriakezelés

A program nem használ fix méretű globális tömböket az adatok tárolására.

- **Tömbök:** A szavazók, hivatalos jelöltek és eredmények tárolása `malloc` és `realloc` segítségével kezelt dinamikus tömbökben történik. A tömb mérete szükség szerint növekszik (geometrikus növekedés).
- **Láncolt lista:** A jelentkezők várólistája tisztán dinamikus láncolt lista, ahol minden elem külön foglalódik le.

2. Kettős indirekció (Pointerre mutató pointer)

A láncolt lista kezelésénél a program **kettős indirekciót** alkalmaz a lista módosítására.

- **Példa:** `int jeloltlista_torol_index(JeloltLista **eleje, int index)`
- **Magyarázat:** A függvény a lista fej-pointerének *címét* veszi át (`**eleje`). Erre azért van szükség, mert ha a lista legelső elemét kell törölni, a függvénynek meg kell változtatnia a hívó fél által birtokolt fej-pointert, hogy az az új első elemre mutasson.

A strukturak.h fejlécállományban deklarált összes függvény részletes leírása

1. Fájlkezelő függvények

Ezek a függvények végzik az adatok mozgatását a háttértár (`.txt` fájlok) és a memória (struktúrák tömbjei) között.

- **int beolvas_szavazok(const char *path, Szavazo **tomb_out, int *db_out)** Beolvassa a szavazók adatait a megadott útvonalról. Dinamikusan foglal memóriát a tömbnek (`realloc`), és visszaadja a tömbre mutató pointert, valamint a beolvasott elemek számát.
- **int mentes_szavazok(const char *path, const Szavazo *arr, int db)** A memóriában lévő szavazók listáját visszaírja a fájlba a módosított formátumban (`Név-Jelszó-Jelölt-Párt`).
- **int beolvas_jeloltek(const char *path, Jelolt **tomb_out, int *db_out)** Beolvassa a hivatalos jelöltek listáját a `jeloltek.txt`-ből egy dinamikus tömbbe.
- **int mentes_jeloltek(const char *path, const Jelolt *arr, int db)** Felülírja a jelöltek fájlját a memóriában lévő aktuális listával (pl. új jelölt felvétele után).
- **int beolvas_eredmeny(const char *path, Eredmeny **tomb_out, int *db_out)** Beolvassa a szavazatok aktuális állását az `eredmeny.txt`-ből.

- **int mentes_eredmeny(const char *path, const Eredmeny *arr, int count)** Elmenti a frissített szavazatszámokat a fájlba.
- **int beolvas_uj_jeloltek(const char *path, Jelolt **tomb_out, int *db_out)** Beolvassa a `uj_jelolt.txt` tartalmát (várólista), hogy az admin dönthessen róluk.
- **int mentes_uj_jeloltek_lista(const char *path, const Jelolt *arr, int db)** A teljes várólistát újraírja (pl. egy jelölt elfogadása vagy elutasítása után, amikor törlődik a listából).
- **int mentes_uj_jelolt(const char *path, const Jelolt *j)** Hozzáír (`append`) egyetlen új jelentkezőt a várólista fájl végéhez (a felhasználói jelentkezésnél használatos).
- **int beolvas_allapot(const char *path, int *allapot_out)** Beolvassa a rendszer állapotát (0=NYITOTT, 1=ZÁRT) a fájlból.
- **int mentes_allapot(const char *path, int allapot)** Elmenti a rendszer módosított állapotát.
- **int beolvas_uzenetek_adminnak(const char *path)** Megnyitja és soronként kiírja a konzolra a felhasználók üzeneteit.
- **int mentes_uzenet_adminnak(const char *path, const char *nev, const char *uzenet)** Egy új üzenetet fűz hozzá az üzenőfájl végéhez.

2. Menü és Felhasználói interfész

A program fő vezérlését és a felhasználóval való kommunikációt végző eljárások.

- **int mainmenu(int allapot)** Megjeleníti a főmenüt. A menüpontok listája attól függ, hogy a választás `allapot`-a nyitott vagy zárt. Visszatér a választott menüpont számával.
- **void szavazas_leadasa(int allapot)** A teljes szavazási folyamatot kezeli: név bekérése, belépés/regisztráció, jelszókezelés, jelöltválasztás és mentés.
- **void statisztika_megtekintese(void)** Kilistázza a jelölteket és a hozzájuk tartozó szavazatszámokat.
- **void szavazatom_megtekintese(void)** Jelszavas azonosítás után megmutatja a felhasználónak, hogy kire adta le a voksát.
- **void jelolt_jelentkezes(void)** Bekéri a felhasználó nevét és pártját, majd felveszi a jelentkezők várólistájára.
- **void uzenet_adminnak(void)** Lehetőséget ad a felhasználónak, hogy rövid szöveges üzenetet küldjön az üzemeltetőnek.
- **int kerdezz_sor(const char *prompt, char *buf, size_t cap)** Biztonságos szövegbeolvasó segédfüggvény. Kiír egy kérdést (`prompt`), beolvas egy sort, és levágja a sorvége jelet.

3. Adminisztrációs felület függvényei

Ezek a függvények csak a jelszóval védett admin módban érhetők el.

- **void admin_menu_kiir(void)** Kiírja az adminisztrátori almenü lehetőségeit.

- **void admin_szavazok_kezelese(void)** Listázza a regisztrált szavazókat, és lehetőséget ad törlésükre.
- **void admin_jelolt_jelentkezesek(void)** Kezeli a várólistát: az admin elfogadhatja (átkerül a hivatalos jelöltek közé) vagy elutasíthatja a jelentkezőket.
- **void admin_uzenetek_megtekintese(void)** Meghívja a beolvasó függvényt az üzenetek listázásához.
- **void admin_allapot(int *allapot)** Lehetővé teszi a választás lezárását vagy újranyitását, és frissíti a főprogram állapotváltozóját.

4. Egyéb segédfüggvények

Kisebb, technikai jellegű feladatokat ellátó függvények.

- **void sor_urites(void)** Kiüríti a bemeneti puffert (stdin), hogy a `scanf` utáni `fgets` hívások helyesen működjenek.
- **void sorveg_levagas(char *s)** Eltávolítja a szöveg végéről a `\n` vagy `\r` karaktereket.
- **int keres_szavazo_index(const Szavazo *t, int db, const char *nev)** Megkeresi egy adott nevű szavazó indexét a tömbben. Ha nincs ilyen, -1-gyel tér vissza.
- **int ellenoriz_es_biztosit_jelszo(Szavazo *s)** Komplex jelszókezelő: ha a szavazónak nincs jelszava, létrehoztat vele egyet; ha van, akkor bekéri és ellenőrzi.

5. Láncolt Lista

A `JeloltLista` kezelésére szolgáló specifikus függvények.

- `int beolvas_uj_jeloltek_list(...)`: Várólista betöltése láncolt listába.
- `int mentes_uj_jeloltek_list(...)`: Láncolt lista tartalmának fájlba írása.
- `void jeloltlista_felszabadit(...)`: Teljes lista memóriájának felszabadítása.
- `int jeloltlista_beszur_vegere(...)`: Új elem fűzése a lánc végére.
- `int jeloltlista_torol_index(...)`: Adott sorszámú elem törlése a láncból (kettős indirekcióval).
- `int jeloltlista_keres_nev(...)`: Név keresése a listában.

Main függvény(main.c)

1. static void admin_felulet(int *allapot)

Ez a függvény kezeli a védett adminisztrátori menübe való belépést és az ottani navigációt. Csak a main függvényből hívható meg.

- **Változók:**
  - int *allapot (Paraméter): Egy mutató (pointer) a main-ben lévő allapot változóra.

        - *Célja:* Lehetővé teszi, hogy az admin menüben történt változtatás (pl. szavazás lezárása) közvetlenül módosítsa a főprogram állapotát.
    - char jelszo[128]: Karaktertömb a jelszó tárolására.
        - *Célja:* A felhasználó által beírt jelszó ide kerül beolvasásra.
    - int v: Egész szám.
        - *Célja:* Tárolja, hogy az admin melyik menüpontot választotta (1-4 vagy 0).
- **Működés részletesen:**

    1. **Jelszó bekérése:** A kerdezz_sor függvénnyel kéri be a jelszót.
    2. **Hitelesítés:** Összehasonlítja a beírt jelszót a fix "adminpass" szöveggel (strcmp). Ha nem egyezik, hibaüzenetet ír és kilép a függvényből (visszatér a főmenübe).
    3. **Admin ciklus:** Ha a jelszó helyes, egy while (true) ciklusba lép.
    4. **Menü kiírása:** Meghívja az admin_menu_kiir()-t (ez csak a szöveget írja ki a képernyőre).
    5. **Bemenet kezelése:** scanf-fel beolvassa a választott számot a v változóba. Ha a beolvasás sikertelen (pl. betűt írtál szám helyett), törli a puffert (sor_urites) és újrakezdi a ciklust.
    6. **Elágazás (switch):** A v értéke alapján meghívja a megfelelő adminisztrátori függvényt (pl. admin_szavazok_kezelese()).
        - A 4-es pontnál (admin_allapot(allapot)) átadja az állapot mutatóját, hogy ott módosítható legyen.

- **Hibakezelés (Errorok):**
    - **"Hiba a jelszo beolvasasakor."**: Ha a kerdezz_sor függvény 0-val tér vissza (pl. bemeneti hiba).
    - **"Hibas jelszo."**: Ha a beírt szöveg nem egyezik az "adminpass"-szal.
    - **"1-4 kozott valassz"**: Ha a felhasználó érvénytelen számot ütött be a menüben.
    - **scanf hiba**: Ha nem számot adtál meg, a kód érzékeli (if (scanf(...) != 1)), üríti a szemetet a bemenetről, és újra kéri a választást, így nem omlik össze a program.

---

2. int main(void)

- **Változók:**
    - int allapot
        - *Célja:* Tárolja, hogy a választás éppen nyitva van-e (0) vagy le van zárva (1). Kezdetben 0.
    - int input

        - *Célja:* A felhasználó főmenüben hozott döntését (1-6 vagy 0) tárolja.
- **Működés részletesen:**

    1. **Inicializálás:** Rögtön az elején megpróbálja beolvasni a korábbi állapotot a files/allapot.txt-ből (beolvas_allapot). Így újraindítás után is emlékszik rá, ha le volt zárva a szavazás.
    2. **Főciklus (while (true)):** A program sosem áll le magától, folyamatosan visszatér a főmenübe, amíg a felhasználó ki nem lép.
    3. **Menü megjelenítése:** Meghívja a mainmenu(allapot) függvényt.
        - *Fontos:* Átadja neki az allapot-ot, mert a menünek tudnia kell, hogy "NYITOTT" vagy "ZÁRT" feliratot írjon-e ki, illetve lezárt állapotban kevesebb opciót jelenítsen meg.
    4. **Bemenet tisztítása:** sor_urites()-t hív, hogy a scanf után a bemeneti pufferben maradt "Enter" karakter (\n) ne zavarja meg a későbbi szöveges beolvasásokat (pl. név megadása).
    5. **Irányítás (switch):** Az input értéke alapján eldönti, hova tovább:
        - 1: Szavazás (szavazas_leadasa).
        - 2-4: Egyéb felhasználói funkciók (statisztika, saját adat, jelentkezés).
        - 5: Admin felület hívása (admin_felulet(&allapot)). Itt referenciát (címet) ad át (&), hogy az admin felület tartósan meg tudja változtatni az allapot változót.
        - 6: Üzenetküldés.
        - 0: Kilépés (return 0).

Hibakezelés (Errorok):

- **"Kerlek 0-6 kozt valassz."**: Ez a default ág a switch-ben. Ha a felhasználó olyan számot ír be, ami nincs a listán, ezt az üzenetet kapja, és a ciklus újraindul.

Fájl műveletek (fajlmuveletek.c)

Ez a modul felelős az adatok (szavazók, jelöltek, eredmények) fájlból való beolvasásáért és fájlba történő kiírásáért.

static void szelso_levag(char *s)

Cél: Eltávolítja a "whitespace" (szóköz, tabulátor, sortörés) karaktereket a megadott sztring elejéről és végéről (trim funkció).

- **Paraméterek:**
    - `[in/out] char *s`: A módosítandó, null-terminált karaktertömb mutatója.
- **Működés:**

1. **Eleje tisztítás:** Megkeresi az első nem-whitespace karaktert. Ha van vezető szóköz, a `memmove` segítségével a hasznos tartalmat a tömb elejére csúsztatja (ez biztonságos átlapolódó memóriaterületeknél is).
2. **Vége tisztítás:** A sztring végétől visszafelé haladva a vezérlőkaraktereket (`\n`, `\r`, `\t`,) null-terminátorra (`\0`) cseréli.

int beolvas_szavazok(const char *path, Szavazo **tomb_out, int *db_out)

Cél:

Beolvassa a szavazók teljes adatbázisát a megadott fájlból egy dinamikusan foglalt tömbbe.

- **Paraméterek:**
  - [in] const char *path: A forrásfájl elérési útja (pl. szavazok.txt).
  - [out] Szavazo **tomb_out: A lefoglalt tömb címét visszaadó pointer.
  - [out] int *db_out: A beolvasott rekordok számát visszaadó pointer.
- **Visszatérési érték:**
  - 1: Sikeres művelet.
  - 0: Hiba (fájlnyitási vagy memóriafoglalási hiba).
- **Implementációs Részletek:**
  - **Formátum:** Nev-Jelszo-Jelolt-Part
  - **Parsing:** A sorokat a - elválasztó karakter mentén darabolja strchr hívásokkal.
  - **Memóriakezelés:** Geometrikusan növekvő stratégiát használ (realloc), kezdetben 8 elem, majd duplázódik.
  - **Validáció:** Ellenőrzi, hogy mind a 4 mező sikeresen kinyerhető-e. Üres mezők esetén a sort kihagyja.
  - **Tisztítás:** Minden mezőre meghívja a szelso_levag függvényt a felesleges szóközök eltávolítására.

int beolvas_jeloltek(const char *path, Jelolt **tomb_out, int *db_out)

Cél:

Betölti a jelöltek listáját a memóriába.

- **Paraméterek:**
  - [in] const char *path: A jelöltek fájljának útvonala.
  - [out] Jelolt **tomb_out: A kimeneti tömb mutatója.
  - [out] int *db_out: A kimeneti elemszám.
- **Visszatérési érték:**
  - 1: Siker.
  - 0: Hiba.
- **Implementációs Részletek:**
  - **Formátum:** Nev-Part

- **Logika:** Az első kötőjel (-) választja el a nevet a párttól. Ez lehetővé teszi, hogy a párt nevében további kötőjelek szerepeljenek (bár a névben nem).

int beolvas_eredmeny(const char *path, Eredmeny **tomb_out, int *db_out)

Cél:

Beolvassa a választás aktuális állását (eredményeket).

- **Paraméterek:**
  - [in] const char *path: Az eredményfájl útvonala.
  - [out] Eredmeny **tomb_out: A kimeneti tömb.
  - [out] int *db_out: A kimeneti elemszám.
- **Visszatérési érték:**
  - 1: Siker.
  - 0: Hiba.
- **Implementációs Részletek:**
  - **Formátum:** Nev-Part-Szavazatszam
  - **Parsing:** A robusztusság érdekében az *első* (strchr) és az *utolsó* (strrchr) kötőjelet keresi.
    - Név: Sor eleje -> első -.
    - Párt: Első - -> utolsó -.
    - Szavazat: Utolsó - után (konverzió: atoi).

int mentes_eredmeny(const char *path, const Eredmeny *tomb, int db)

Cél:

A memóriában lévő eredménytömb tartalmával felülírja a megadott fájlt.

- **Paraméterek:**
  - [in] const char *path: A célfájl útvonala.
  - [in] const Eredmeny *tomb: A mentendő adatok tömbje.
  - [in] int db: A tömb elemeinek száma.
- **Visszatérési érték:** 1 siker esetén, 0 hiba esetén.
- **Mód:** "w" (Write) - A fájl korábbi tartalma törlődik.

int mentes_uj_jelolt(const char *path, const Jelolt *j)

Cél:

Egy új jelölt jelentkezését fűzi hozzá a várólista fájlhoz.

- **Paraméterek:**
  - [in] const char *path: A fájl útvonala.
  - [in] const Jelolt *j: Az egyetlen mentendő jelölt adatai.
- **Visszatérési érték:** 1 siker esetén, 0 hiba esetén.

- **Mód:** "a" (Append) - Az adat a fájl végére íródik, a meglévő tartalom megmarad.

int mentes_jeloltek(const char *path, const Jelolt *arr, int db)

Cél: A hivatalos jelöltek listájának mentése a fájlba (pl. új jelölt felvétele után).

- **Bemenet:**
  - `path`: A fájl elérési útja (`files/jeloltek.txt`).
  - `arr`: A jelöltek tömbje.
  - `db`: A tömb elemeinek száma.
- **Kimenet:** Frissített fájl.
- **Működés:**
  - Megnyitja a fájlt írásra (`"w"`), ezzel törli a régi tartalmat.
  - Soronként kiírja a jelölteket `Név-Párt` formátumban.

int mentes_szavazok(const char *path, const Szavazo *arr, int db)

Cél: A szavazók adatbázisának frissítése (pl. szavazat leadása vagy új regisztráció után).

- **Bemenet:** `path`, `arr` (Szavazo tömb), `db`.
- **Működés:**
  - Felülírja a `szavazok.txt` fájlt.
  - Formátum: `Név-Jelszó-Jelölt-Párt`.
  -

Láncolt Listás Megvalósítás

int beolvas_uj_jeloltek_list(const char *path, JeloltLista **eleje_out)

Cél: A jelentkezők betöltése dinamikus láncolt listába (ez váltja ki a tömbös megoldást az adminisztrációban).

- **Bemenet:** `path`.
- **Kimenet:** `**eleje_out` – A lista fej-pointerének címe (**kettős indirekció**).
- **Működés:**
  1. Megnyitja a fájlt.
  2. Soronként olvassa és parse-olja az adatokat (`strchr`, `strncpy`).
  3. Minden sorhoz meghívja a `jeloltlista_beszur_vegere` függvényt, ami új csomópontot fűz a listához.

int mentes_uj_jeloltek_list(const char *path, const JeloltLista *eleje)

Cél: A memóriában lévő láncolt lista tartalmának visszaírása fájlba.

- **Paraméterek:** `const JeloltLista *eleje` (Lista kezdőpontja).
- **Működés:**
  - Végigiterál a listán (`for (it = eleje; it; it = it->kov)`).
  - Soronként kiírja a `Név-Párt` adatokat a fájlba.

Rendszerállapot és Üzenetek

int beolvas_allapot(const char *path, int *allapot_out)

Cél: A választás státuszának (Nyitott/Zárt) betöltése.

- **Kimenet:** `*allapot_out` pointeren keresztül (0 vagy 1).
- **Működés:**
  - Ha a fájl nem létezik vagy hibás, alapértelmezett 0 (NYITOTT) értéket ad vissza.
  - `fscanf`-fel olvas be egyetlen számot.

int mentes_allapot(const char *path, int allapot)

Cél: A választás státuszának rögzítése.

- **Működés:** A fájlba írja az `allapot` változó értékét (0 vagy 1).

int beolvas_uzenetek_adminnak(const char *path)

Cél: A felhasználói üzenetek listázása a képernyőre.

- **Működés:**
  - Soronként olvassa a fájlt és `printf`-fel kiírja. Nem tárolja el a memóriában az adatokat.

int mentes_uzenet_adminnak(const char *path, const char *nev, const char *uzenet)

Cél: Új üzenet küldése.

- **Működés:**
  - `"a"` (append) módban nyitja meg a fájlt, így az új üzenet a végére kerül, a régiek megmaradnak.
  - Formátum: `Név-Üzenet`.

Idő Kezelése

int mentes_lezaras_idopont(const char *path)

Cél: Rögzíti a szavazás lezárásának pillanatát.

- **Működés:**
  - A `mai_ido_yyyy_mm_dd_hh_mm_ss` segédfüggvény (mely a `localtime`-ot használja) segítségével előállítja az időbélyeget.
  - Kiírja a `files/lezaras.txt` fájlba.

int beolvas_lezaras_idopont(const char *path, char *buf, size_t cap)

Cél: Visszaolvassa a lezárás idejét a fájlból megjelenítéshez.

- **Bemenet:** `buf` (puffer a szövegnek), `cap` (puffer mérete).
- **Működés:**
  - Beolvassa az első sort a fájlból.
  - Levágja a sorvége jelet (`sorveg_levagas`).
  - Siker esetén `1`-gyel tér vissza.

Hibakezelés

- **Fájl megnyitás:** Minden függvény ellenőrzi az fopen visszatérési értékét. Sikertelen megnyitás esetén hibaüzenetet ír a standard kimenetre (printf), és 0 hibakóddal tér vissza.
- **Memóriafoglalás:** A realloc visszatérési értékét minden esetben ellenőrzi a kód (if (!uj)). Hiba esetén felszabadítja a korábban lefoglalt területet (free), lezárja a fájlt, és biztonságosan kilép, megelőzve a memóriaszivárgást.
- **Buffer Overflow védelem:** A sztringműveleteknél (strncpy) a kód explicit módon kezeli a méretkorlátokat (sizeof dest - 1) és gondoskodik a lezáró null-bájtról.

Menu fuggvenyek(menu_fuggvenyek.c)

Ez a forrásfájl tartalmazza a program felhasználói interakcióért felelős függvényeit. Itt valósul meg a menürendszer mögötti logika: a szavazás folyamata, az adatok megjelenítése, a felhasználói bemenetek kezelése és a jogosultságok ellenőrzése (pl. jelszó).

int kerdezz_sor(const char *prompt, char *buf, size_t cap)

Cél: Biztonságos szövegbeolvasást valósít meg a standard bemenetről (`stdin`).

- **Paraméterek:**

- `[in] const char *prompt`: A felhasználónak megjelenítendő kérdés/üzenet.
- `[out] char *buf`: A beolvasott szöveg tárolására szolgáló puffer.
- `[in] size_t cap`: A puffer maximális mérete (kapacitás).

- **Működés:**

  1. Kiírja a kérdést.
  2. `fgets` segítségével beolvassa a sort (maximum `cap` méretig), védve a túlcsordulástól.
  3. A `sorveg_levagas` függvénnyel eltávolítja a sorvége jeleket (`\n`).

- **Visszatérési érték:** `1` siker esetén, `0` hiba esetén.

int keres_szavazo_index(const Szavazo *t, int db, const char *nev)

Cél: Megkeresi egy adott nevű szavazó indexét a betöltött tömbben.

- **Paraméterek:**
  - `[in] const Szavazo *t`: A szavazók tömbje.
  - `[in] int db`: A tömb elemeinek száma.
  - `[in] const char *nev`: A keresett név.
- **Működés:** Lineáris keresést végez (`strcmp`).
- **Visszatérési érték:** A találat indexe (`0..db-1`), vagy `-1`, ha nem található.

int ellenoriz_es_biztosit_jelszo(Szavazo *s)

Cél: Kezeli a felhasználói hitelesítést a szavazás során.

- **Paraméterek:**
  - `[in/out] Szavazo *s`: A vizsgált szavazó rekordjára mutató pointer.
- **Működés:**

  1. **Ha nincs jelszó:** Felszólítja a felhasználót egy új jelszó létrehozására, és azt elmenti a memóriában lévő struktúrába.
  2. **Ha van jelszó:** Bekéri a jelszót és összehasonlítja a tárolttal.

- **Visszatérési érték:** `1` sikeres azonosítás/létrehozás esetén, `0` hiba vagy rossz jelszó esetén.

---

void szavazas_leadasa(int allapot)

Cél: A teljes szavazási folyamat vezérlése: regisztráció, belépés, jelöltválasztás és szavazatmentés.

- **Paraméterek:** `allapot` (nem használt, kompatibilitási okokból).
- **Folyamat:**
    1. **Adatbetöltés:** Beolvassa a szavazókat (`beolvas_szavazok`).
    2. **Azonosítás:** Bekéri a nevet.
        - *Új szavazó:* Ha a név nem létezik, új rekordot hoz létre (`realloc`), és jelszót kér be.
        - *Meglévő szavazó:* Ellenőrzi, hogy szavazott-e már (`meg_nem_szavazott`), majd hitelesíti a jelszót (`ellenoriz_es_biztosit_jelszo`).
    3. **Szavazás:**
        - Betölti és listázza a jelölteket (`beolvas_jeloltek`).
        - Bekéri a választott jelölt nevét, és ellenőrzi létezését.
    4. **Eredmény frissítése:**
        - Betölti az eredményfájlt (`beolvas_eredmeny`).
        - Megkeresi a jelöltet és növeli a szavazatszámát.
        - Visszamenti az eredményfájlt (`mentes_eredmeny`).
    5. **Szavazó mentése:**
        - Rögzíti a szavazó rekordjában a választott jelöltet és pártot.
        - Visszamenti a frissített szavazói listát (`mentes_szavazok`).
    6. **Memóriatakarítás:** Minden dinamikusan foglalt tömböt felszabadít (`free`).

void statisztika_megtekintese(void)

Cél: Megjeleníti a szavazás jelenlegi állását.

- **Működés:**
    - Beolvassa az `eredmeny.txt` fájlt.
    - A konzolra írja a jelöltek nevét, pártját és a kapott szavazatokat.
    - Hiba esetén (pl. üres fájl) megfelelő üzenetet ad.

void szavazatom_megtekintese(void)

Cél: Lehetővé teszi a felhasználónak, hogy ellenőrizze, kire adta le a voksát.

- **Működés:**
    1. Bekéri a nevet és betölti a szavazólistát.
    2. Megkeresi a felhasználót.
    3. **Jelszóellenőrzés:**
        - Ha a felhasználónak nincs jelszava (pl. admin vitte fel, de még nem lépett be), a funkció hibaüzenettel visszairányítja a szavazás menüpontba regisztrálni.
        - Ha van jelszó, bekéri és ellenőrzi.
    4. Siker esetén kiírja a tárolt szavazatot (`Jelolt`, `Part`).

void jelolt_jelentkezes(void)

Cél: Kezeli az új jelöltek jelentkezését a várólistára.

- **Működés:**
  1. Bekéri a jelölt nevét és pártját.
  2. Betölti a *hivatalos* jelöltek listáját (`jeloltek.txt`), hogy ellenőrizze, szerepel-e már a név (duplikáció szűrése).
  3. Ha nem szerepel, létrehoz egy `Jelolt` struktúrát és hozzáírja a várólistához (`mentes_uj_jelolt`), amit később az admin bírálhat el.

void uzenet_adminnak(void)

Cél: Egyszerű üzenetküldési felület a felhasználók számára.

- **Működés:**
  1. Bekéri a feladó nevét és az üzenet szövegét (egy sorban).
  2. Validálja, hogy a mezők nem üresek-e.
  3. A `mentes_uzenet_adminnak` segítségével hozzáfűzi az üzenetet a naplófájlhoz.

Hibakezelés

A modul kiterjedt hibakezelést alkalmaz:

- **Bemenet:** Minden `kerdezz_sor` hívás visszatérési értékét ellenőrzi.
- **Fájlműveletek:** A betöltő függvények hibája esetén (`!beolvas_...`) hibaüzenetet ír, felszabadítja a már lefoglalt erőforrásokat, és visszatér a menübe, megakadályozva a program összeomlását.
- **Memóriaszivárgás ellen:** Minden hibaágon (`return` előtt) gondoskodik a dinamikusan foglalt tömbök (`szavazok`, `jeloltek`, `eredm`) felszabadításáról (`free`).

Admin fuggvenyek(admin_fuggvenyek.c)

Ez a modul valósítja meg az adminisztrátori felület üzleti logikáját. A fájl tartalmazza a szavazók kezelését (törlés), a jelöltek elbírálását (elfogadás/elutasítás láncolt listával), a rendszerüzenetek olvasását, valamint a választás globális állapotának vezérlését.

void admin_menu_kiir(void)

Cél: Kirajzolja a konzolra az adminisztrátori almenü opcióit.

- **Működés:**
  - Kiírja a választható funkciókat:
    1. Szavazók kezelése
    2. Jelölt jelentkezések
    3. Üzenetek megtekintése
    4. Választás állapota
    5. Vissza a főmenübe
- **Visszatérési érték:** Nincs.

---

void admin_szavazok_kezelese(void)

Cél: Listázza a regisztrált szavazókat, és lehetőséget biztosít a törlésükre.

- **Működés:**
  1. **Betöltés:** Meghívja a `beolvas_szavazok` függvényt, ami dinamikus tömbbe tölti az adatokat.
  2. **Ciklus:** Folyamatosan listázza a neveket, és várja a felhasználó utasítását.
  3. **Törlés (Algoritmus):**
     - Név alapján megkeresi a törlendő elem indexét (`strcmp`).
     - **Tömbös törlés:** A törlendő elem utáni összes elemet egy pozícióval balra csúsztatja (`t[i-1] = t[i]`), felülírva a törölt elemet.
     - Csökkenti a darabszámot (`db--`) és szükség esetén felszabadítja a memóriát.
  4. **Mentés:** Sikeres törlés után azonnal frissíti a `szavazok.txt` fájlt.

void admin_jelolt_jelentkezesek(void)

Cél: A `uj_jelolt.txt`-ben lévő várólista kezelése. Ez a függvény demonstrálja a **láncolt listák** és a **kettős indirekció**használatát.

- **Adatszerkezet:** `JeloltLista` (láncolt lista).
- **Működés:**
  1. **Betöltés:** A `beolvas_uj_jeloltek_list` függvénnyel felépíti a láncolt listát a memóriában.
  2. **Listázás:** Végigiterál a láncon (`it = it->kov`) és kiírja a jelentkezőket.
  3. **Elfogadás ('e'):**
     - Az admin sorszámot választ.
     - A program kimenti a kiválasztott jelölt adatait a listából.
     - Hozzáadja a jelöltet a hivatalos `jeloltek.txt` fájlhoz (tömb bővítése `realloc`-kal).
     - Hozzáadja a jelöltet az `eredmeny.txt` fájlhoz (0 szavazattal).

        - **Törlés a listából:** Meghívja a `jeloltlista_torol_index(&lista, idx)` függvényt, amely kivágja az elemet a láncból és felszabadítja a memóriáját.
        - Frissíti a `uj_jelolt.txt`-t a lista új tartalmával.
    4. **Elutasítás ('x'):**
        - Sorszám alapján törli az elemet a listából (`jeloltlista_torol_index`).
        - Frissíti a fájlt (az adat elveszik).
- **Memóriakezelés:** A funkció végén a `jeloltlista_felszabadit` hívással törli a teljes listát.

void admin_uzenetek_megtekintese(void)

Cél: Kilistázza a felhasználói üzeneteket.

- **Működés:**
    - A `beolvas_uzenetek_adminnak` segédfüggvényt hívja, amely soronként olvassa és írja ki a fájl tartalmát.

void admin_allapot(int *allapot)

Cél: Vezérli a választás globális státuszát (NYITOTT/ZÁRT).

- **Paraméterek:**
    - `[in/out] int *allapot`: Pointer a `main.c`-ben lévő állapotváltozóra.
- **Működés:**

    1. Kiírja a fájlban és a memóriában tárolt állapotot.
    2. **Állapot váltása ('a'):**
        - Megfordítja az értéket (`0` <-> `1`).
        - Mentés fájlba (`mentes_allapot`).
        - **Közvetlen memória-módosítás:** A pointeren keresztül (`*allapot`) azonnal frissíti a futó program állapotát is.

Hibakezelés

- **Fájlműveletek:** Minden betöltésnél (`if (!beolvas_...)`) ellenőrzi a sikert, hiba esetén üzenetet ír és visszatér.
- **Bemenet validáció:**
    - `scanf` visszatérési értékének ellenőrzése (`!= 1`).
    - `sor_urites()` a puffer tisztítására hibás bevitel után.
    - Indexhatárok ellenőrzése tömböknél és listáknál.
- **Memóriabiztonság:**

- A láncolt lista minden művelete (beszúrás, törlés) kezeli a `NULL` pointereket.
- A függvények kilépéskor felszabadítják a lefoglalt erőforrásokat (`free`, `jeloltlista_felszabadit`).

Segedfuggvenyek(segedfuggvenyek.c)

Ez a fájl tartalmazza az alkalmazás általános célú segédfüggvényeit. Három fő területet fed le:

1. **String kezelés:** Bemeneti pufferek tisztítása, sorvége jelek kezelése.
2. **Állapot- és Időkezelés:** A választás lezárási idejének ellenőrzése, rendszerállapot (nyitott/zárt) fájlba írása és olvasása.
3. **Láncolt Lista:** A `JeloltLista` dinamikus adatszerkezet kezeléséhez szükséges műveletek (létrehozás, beszúrás, törlés, keresés), amelyek **kettős indirekciót** alkalmaznak.

Ezek a függvények csak modulon belül érhetők el.

String segédfüggvények

void sor_urites(void)

Cél: A standard bemeneti puffer (`stdin`) kiürítése.

- **Bemenet:** A pufferben maradt feldolgozatlan karakterek.
- **Kimenet:** Üres bemeneti puffer.
- **Működés:**
  - Egy ciklussal karakterenként olvassa a bemenetet (`getchar()`), amíg újsor (`\n`) karaktert vagy fájl végét (`EOF`) nem talál.
  - **Használat:** Tipikusan `scanf` hívások után, hogy a "beragadt" Enter billentyű ne zavarja meg a későbbi beolvasásokat.

void sorveg_levagas(char *s)

Cél: A sztring végéről eltávolítja a sorvége jeleket.

- **Bemenet:** `char *s` – A módosítandó sztring pointere.
- **Kimenet:** A tisztított sztring (helyben módosítva).
- **Működés:**

- Megvizsgálja a sztring végét. Ha `\n` (Linux/Unix) vagy `\r` (Windows) karaktert talál, lecseréli őket null-terminátorra (`\0`).

Láncolt Lista (JeloltLista)

Ez a rész valósítja meg a dinamikus memóriakezelést és a **kettős indirekciót** a várólista kezeléséhez.

static JeloltLista *jeloltlista_uj_node(const Jelolt *j)

Cél: Új láncolt lista csomópont (node) létrehozása és inicializálása.
(Belső, `static` segédfüggvény).

- **Bemenet:** `const Jelolt *j` – A tárolandó adat (név, párt). Ha `NULL`, üres adatot hoz létre.
- **Kimenet:** Pointer az újonnan foglalt memóriaterületre, vagy `NULL` hiba esetén.
- **Működés:**
  - `malloc`-kal foglal memóriát a csomópontnak.
  - Beállítja a `kov` pointert `NULL`-ra.
  - Másolja az adatokat a struktúrába.

int jeloltlista_beszur_vegere(JeloltLista **eleje, const Jelolt *j)

Cél: Új elem hozzáfűzése a lánc végére.

- **Bemenet:**
  - `JeloltLista **eleje`: Pointer a lista fej-pointerére (**kettős indirekció**).
  - `const Jelolt *j`: A beszúrandó adat.
- **Kimenet:** `1` siker esetén, `0` hiba esetén.
- **Működés:**
  - Létrehoz egy új csomópontot (`jeloltlista_uj_node`).
  - **Kettős indirekció oka:** Ha a lista üres (`*eleje == NULL`), a függvénynek meg kell változtatnia a hívó fél `eleje`pointerét, hogy az az új elemre mutasson (`*eleje = uj`).
  - Ha a lista nem üres, végigiterál az utolsó elemig, és hozzáfűzi az újat (`it->kov = uj`).

int jeloltlista_torol_index(JeloltLista **eleje, int index)

Cél: Elem törlése a listából a sorszáma (indexe) alapján.

- **Bemenet:**
  - `JeloltLista **eleje`: Pointer a lista fej-pointerére (**kettős indirekció**).

    - `int index`: A törlendő elem 0-alapú indexe.
- **Kimenet:** `1` siker esetén, `0` hiba esetén (pl. érvénytelen index).
- **Működés:**

    1. **Index 0 (Fej törlése):** Ha az első elemet töröljük, a fej-pointert (`*eleje`) átállítjuk a második elemre (`curr->kov`). Itt elengedhetetlen a kettős indirekció.
    2. **Belső elem törlése:** Megkeressük a törlendő elemet (`curr`) és az előzőt (`prev`). Átkötjük a láncot: `prev->kov = curr->kov` (az előző mostantól a törölt utánira mutat).
    3. **Felszabadítás:** A kivágott elem memóriáját felszabadítjuk (`free(curr)`).

int jeloltlista_keres_nev(const JeloltLista *eleje, const char *nev)

Cél: Megkeresi, hányadik indexen található egy adott nevű jelölt.

- **Bemenet:**
    - `const JeloltLista *eleje`: A lista kezdőpontja.
    - `const char *nev`: A keresett név.
- **Kimenet:** Az elem indexe, vagy `-1` ha nem található.
- **Működés:**
    - Lineáris kereséssel végigiterál a listán (`strcmp` segítségével hasonlít).

void jeloltlista_felszabadit(JeloltLista *eleje)

Cél: A teljes lista memóriájának felszabadítása.

- **Bemenet:** A lista fej-pointere.
- **Működés:**
    - Végigiterál a listán, és minden csomópontra meghívja a `free`-t, megelőzve a memóriaszivárgást.

Időkezelés

static int mai_ido_yyyy_mm_dd_hh_mm_ss(char *buf, size_t cap)

Cél: Az aktuális rendszeridő formázott sztringgé alakítása. (Belső, `static` függvény).

- **Bemenet:** Puffer (`buf`) és mérete (`cap`).
- **Kimenet:** `1` siker, `0` hiba. A pufferben: `"YYYY-MM-DD HH:MM:SS"`.
- **Működés:**
    - Lekéri az időt (`time`, `localtime`).
    - `snprintf` segítségével formázza a kimenetet.

static int fix_idopont_elmult(void)

Cél: Ellenőrzi, hogy a rendszerben rögzített fix határidő (2026.04.05 12:00) elmúlt-e már. (Belső, `static` függvény).

- **Bemenet:** Rendszeridő (`time(NULL)`).
- **Kimenet:** `1` (IGAZ) ha elmúlt, `0` (HAMIS) ha még nem.
- **Működés:**
  - Feltölt egy `struct tm` struktúrát a fix dátummal.
  - Átalakítja időbélyeggé (`mktime`).
  - Összehasonlítja a jelenlegi idővel.

int lezaro_idopont_elmult(void)

Cél: Publikus burkolófüggvény (wrapper) az időellenőrzéshez.

- **Működés:** Egyszerűen meghívja a `fix_idopont_elmult` függvényt és visszatér annak eredményével.

Rendszerkövetelmények és könyvtárak

A program **Szabványos C (Standard C99 vagy újabb)** nyelven íródott.

- **Fordítóprogram:** Bármilyen szabványos C fordító (GCC, Clang, MSVC/Visual Studio, MinGW).
- **Külső könyvtárak:** A program **nem igényel** előre telepítendő külső könyvtárakat (pl. grafikus csomagokat).
- **Belső függőség (Mellékelve):** A program használja a `debugmalloc.h` modult a memóriakezelés ellenőrzésére. Ez a fájl a forráskód mellékelve van, telepítést nem igényel, de fordításkor a forrásfájlok mellett kell lennie.

Parancssori fordítás (GCC/Clang példa)

Nyissunk egy terminált (vagy parancssort) a projekt gyökérkönyvtárában (ahol a `.c` fájlok vannak), és futtassuk az alábbi parancsot:

Windows (MinGW/GCC):

Bash
```
gcc main.c admin_fuggvenyek.c menu_fuggvenyek.c fajlmuveletek.c
segedfuggvenyek.c -o valasztas.exe -Wall -Wextra
```

Linux / macOS:

Bash

```
gcc main.c admin_fuggvenyek.c menu_fuggvenyek.c fajlmuveletek.c
segedfuggvenyek.c -o valasztas -Wall -Wextra
```

(A `-Wall -Wextra` kapcsolók ajánlottak a figyelmeztetések megjelenítéséhez, de a működéshez nem feltétlenül szükségesek.)

Fordítás IDE-ben (Code::Blocks, Visual Studio, CLion)

1. Hozzunk létre egy **új C Console Application** projektet.
2. Adjuk hozzá a projekthez az **összes** meglévő `.c` és `.h` fájlt (beleértve a `debugmalloc.h`-t és a `strukturak.h`-t is).
3. Nyomjuk meg a **Build & Run** gombot.

Futtatás és könyvtárszerkezet

A program helyes működéséhez elengedhetetlen, hogy a futtatható állomány (`valasztas.exe` vagy `./valasztas`) mellett létezzen a **`files`** nevű mappa, amelyben a szöveges adatbázisok találhatók.

Fontos: Ha a program indításkor "Nem sikerult megnyitni a ... fájlt" hibaüzenetet ad, győződjön meg róla, hogy a `files` mappa közvetlenül a futtatható fájl mellett található, és a programnak van írási/olvasási jogosultsága a mappában.