

# Programozói Dokumentáció - Autószerviz

## 1. A Programról

A program egy autószerviz nyilvántartását segíti. Kezeli az ügyfeleket, az autóikat és az autókon elvégzett javításokat.

## 2. Adatkezelés és tarolás.

Az adatokat láncolt listában és szöveges fájlokban taroljuk. Ezeket dinamikus memóriával kezeljük, futásidőben. A rendszer három fő részre bontható, melyek külön szöveges állományokban találhatók:

- **Ügyfelek** (ugyfelek.txt): Az autótulajdonosok személyes adatai.
- **Autók** (autok.txt): A gépjárművek adatai, melyek a tulajdonos nevével kapcsolódnak az ügyfelekhez.
- **Javítások** (javitasok.txt): A szervizelés bejegyzései, melyek rendszám alapján kapcsolódnak az autóhoz.

A program soronként olvassa be az adatokat (`fgets`), majd a sorvégi entereket levágja (`strcspn`), és a pontosvesszővel elválasztott mezőket dolgozza fel (`sscanf`).

Ezeket a data mappában taroljuk.

Mivel az adatok mennyisége változó vagy nem ismert, ezért a program dinamikus memória kezelést használ. Ehhez segítségül minden .c fájlban include-olva van a `debugmalloc.h` segéd file ami memória szivárgás esetén kijelzi a hibát a program futtatása után.

## 3. Adatszerkezetek

A program láncolt listát használ. A feladathoz ez volt a legmegfelelőbb adat struktúra, de e mellett meg kényelmesen használható is. A struktúra definíciók a `structs.h` fájlban találhatóak. A struktúrák idegen kulcs szerű adatokkal dolgoznak. Az ügyfél struktúrában a név egyedi kell hogy legyen, ez lesz az autó struktúrában a Tulajdonos név, ami összekapcsolja a két struktúrát. Hasonlóan az autó struktúra rendszám értéké egy idegenkulcsa a javítás struktúrának, így annak is feltételé hogy egyedi legyen.

3.1 Ügyfél struktúra (Ugyfel): A tulajdonosok adatait tárolja. A nev mező egyedi azonosítóként is szolgál a rendszerben.

```
typedef struct Ugyfel {  
    char nev[64];  
    char email[64];  
};
```

```

char telSz[64];

struct Ugyfel *kov; // Pointer a következő elemre

} Ugyfel;

```

**3.2 Autó struktúra (Auto): Az autók adatait tárolja. A tulajNev mező kapcsolja az autót az Ügyféllel.**

```

typedef struct Auto {

    char rendSz[16]; // Egyedi azonosító (kulcs)

    char model[64];

    char vizsgaErv[16]; // Dátum string (ÉÉÉÉ-HH-NN)

    char tulajNev[64]; // Kapcsolat a tulajdonossal

    struct Auto *kov;

} Auto;

```

**3.3 Javítás struktúra (Javitas): Az elvégzett javításokat rögzíti. A rendSz mező kapcsolja össze a javítást az autóval.**

```

typedef struct Javitas {

    char rendSz[16];

    char tipus[64];

    char datum[16];

    int ar;

    struct Javitas *kov;

} Javitas;

```

#### **4. Modulok és függvények**

A program moduláris felépítésű, az egyes funkciók külön forrásfájlokban (.c) és hozzáartozó fejlécfájlokban (.h) kerültek.

##### **4.1. Főprogram (main.c)**

Itt van megvalósítva a felhasználói menü, ami meghívja a különböző modulokban levő függvényeket. A menü pontjait a felhasználó 1-9 és számmal tudja kiválasztani, más bemenet esetén a program menüje újból megjelenik. Ez a modul felelős továbbá a program indulásakor az adatok betöltéséért a fájlok ból, valamint a kilépéskor a változások mentéséért és a dinamikusan foglalt memória felszabadításáért. Segédfüggvények definiálásához használható a functions.c és functions.h, ami hasznos teszteléshez.

## 4.2. Ügyfél modul (ugyfel.c)

Ez a modul tartalmazza az ügyfelek adatainak kezeléséhez szükséges függvényeket. Itt valósul meg az *ugyfelHozzaad* függvény, amely új ügyfelet szűr be a láncolt lista elejére, valamint az *ugyfelKeres*, amely név alapján keres a listában. A modul felelős az *ugyfelek.txt* fájl beolvasásáért (*betoltUgyfelek*) és a módosított lista fájlba történő visszaírásáért (*mentUgyfelek*). és persze a *felszabaditUgyfel* függvény is, ami a program bezárásakor felszabadítót a dinamikus memóriát.

*Ugyfel\* betoltUgyfelek(const char\* filename)* Láncolt listaként beolvassa az ügyfeleket a szöveges állományból. minden sorból egy új Ugyfel elemet hoz létre.

*void felszabaditUgyfelek(Ugyfel \*lista)* Felszabadítja az ügyfelek láncolt listáját a memóriából.

*Ugyfel\* ugyfelHozzaad(Ugyfel \*ugyfelek, const Ugyfel \*ujUgyfel)* Új ügyfelet ad hozzá a listához. Az új elem a lista elejére kerül beszúrásra.

*Ugyfel\* ugyfelKeres(Ugyfel \*ugyfelek, const char \*nev)* Keres egy ügyfelet a neve alapján a listában. Ha megtalálja, visszaadja az elem címét, ellenkező esetben NULL pointerrel tér vissza.

*void mentUgyfelek(const char \*filename, Ugyfel \*lista)* Kimenti az ügyfelek listáját a megadott fájlba CSV formátumban (név;email;telefonszám).

## 4.3. Autó modul (auto.c)

Itt találhatóak a gépjárművek nyilvántartását végző függvények. A modul elemei az *autoTorles*, amely egy adott rendszámú autót távolít el a láncolt listából a láncolás helyreállításával, valamint a *lejaroVizsgak*, amely dátum-összehasonlítást végezve listázza a lejárt műszaki vizsgával rendelkező járműveket. Emellett ez a modul kezeli az autókhöz tartozó szerviztörténet lekérdezését is az *autoSzervizTortenet* függvénykel. Rendszam szerinti (*autoKeres*) függvény is itt található. Itt is szerepelnek a fájl kezelő és memória kezelő függvények (*betoltAutok*, *felszabaditAutok*, *autoHozzaad*, *mentAutok*)

*Auto\* betoltAutok(const char\* filename)* Soronként beolvassa az autók adatait a fájlból, és létrehozza a láncolt listát. A sorvégi entereket és a mezőket elválasztó karaktereket kezeli.

*void felszabaditAutok(Auto \*lista)* Végigiterál a listán, és felszabadítja az autókhoz foglalt memóriaterületeket.

*Auto\* autoHozzaad(Auto \*autok, const Auto \*ujAuto)* Létrehoz egy új autót a memóriában a paraméterben kapott adatokkal, és beszűrja a lista elejére. Visszatér a módosított lista fejével.

*Auto\* autoKeres(Auto \*autok, const char \*rendSz)* Lineáris keresést végez a listában rendszám alapján. Ha megtalálja az autót, visszatér a rá mutató pointerrel, egyébként NULL-t ad vissza.

*int autoTorles(Auto \*\*autok, const char \*rendSz)* Eltávolít egy autót a listából a megadott rendszám alapján. Kezeli a láncolt lista újrafűzését (akár a lista elejéről, akár közepéről történik a törlés). Visszatérési értéke jelzi a törlés sikereségét.

*void autoSzervizTortenet(Auto \*autok, Javitas \*javitasok, const char \*rendSz)* Kikeresi az adott rendszámú autót, majd végignézi a javítások listáját, és kiírja a képernyőre az adott autóhoz tartozó összes szervizbejegyzést.

*void lejaroVizsgak(Auto \*autok, const char \*maiDatum)* Összehasonlítja az autók műszaki vizsgájának érvényességét a megadott dátummal, és kilistázza azokat, amelyeknek lejárt a vizsgája.

*void mentAutok(const char \*filename, Auto \*lista)* A program bezárásakor elmenti az autók aktuális listáját a háttértárolóra.

#### 4.4. Javítás modul (javitas.c)

Ez a forrásfájl kezeli a szervizmunkálatok rögzítését. A modul speciális függvénye a *javitasTorlesRendszamSzerint*, amely nemcsak egy elemet, hanem az összes adott rendszámhoz tartozó javítást törli a listából. Ez biztosítja, hogy autó törlésekor ne maradjanak autóhoz nem tartozó javítások a rendszerben. A modul szintén tartalmazza a fájlkezelő (*betoltJavitasok*, *mentJavitasok*) és memóriakezelő (*felszabaditJavitasok*) függvényeket.

*Javitas\* betoltJavitasok(const char\* filename)* Beolvassa a javításokat a megadott nevű szöveges fájlból, és felépít belőlük egy láncolt listát. Hiba esetén NULL pointerrel tér vissza.

*void felszabaditJavitasok(Javitas \*lista)* Felszabadítja a javításokat tartalmazó láncolt lista összes elemét a dinamikus memóriából.

*Javitas\* javitasHozzaad(Javitas \*javitasok, const Javitas \*ujJavitas)* Új javítási bejegyzést ad hozzá a meglévő lista elejéhez. Dinamikusan foglal memóriát az új elemnek, és visszatér a lista új fejével.

*void mentJavitasok(const char \*filename, Javitas \*lista)* Elmenti a javítások láncolt listájának tartalmát a megadott fájlba, pontosvesszővel tagolt (CSV) formátumban.

*void javitasTorlesRendszamSzerint(Javitas \*\*javitasok, const char \*rendSz)* Kötődik az autó törléséhez, mivel ott kérdezi meg a program, hogy akarjuk-e torolni az autóhoz tartozó javításokat. Törli a listából az összes olyan javítási bejegyzést, amely a paraméterben kapott rendszámhoz tartozik. Módosíthatja a lista fejét, ezért a listamutató címét veszi át.

## 5. Függvények használta

### 5.1 Ügyfél hozzáadása (1. menüpont)

A funkció kiválasztásakor a program először a nevet olvassa be. Mielőtt bármilyen további adatot bekérne, a program egyből ellenőrzi az adatbázist a keresőfüggvény segítségével (*ugyfelKeres*). Ez azért fontos, hogy ne legyen két azonos nevű ügyfél, így elkerüljük a duplikációt. Ha a név egyedi, akkor bekéri az e-mailt és telefonszámot is. Végül egy megerősítő kérdés után a memóriába tölti az új struktúrát, és meghívja a listaépítő függvényt (*ugyfelHozzaad*), ami beszúrja az új elemet a lista elejére.

### 5.2 Keresés ügyfél neve szerint (4. menüpont)

Ez a menüpont logikailag összekapcsolja a két láncolt listát. A név beolvasása után először megkeresi a konkrét ügyfelet az ügyfelek listájában. Ha megtalálta, kiírja az adatait, de nem áll meg: elindít egy ciklust az autók listáján is. Végigmegy az összes járművön, és ahol a tulajdonos neve megegyezik a talált ügyféllel, azt is kiírja. Ez a „kézi” összekapcsolás teszi lehetővé, hogy lássuk az ügyfél összes autóját anélkül, hogy közvetlen pointert tárolnánk.

### 5.3 Autó és kapcsolódó adatok törlése (6. menüpont)

A törlési folyamat a főprogramban több lepésekben valósul meg, amely biztosítja az adatbázis konzisztenciáját. A felhasználó által megadott rendszám alapján a program először megkísérli az autó törlését az autoTorles függvénytel. Ez a függvény a láncolt listából való törlést valósít meg: két mutató segítségével (p az aktuális elemre, prev az előzőre mutat) keresi meg a törlendő elemet. Amennyiben megtalálja, a láncolást úgy módosítja, hogy az előző elem mutatóját (prev->kov) átirányítja a törlendő elem utáni

elemre, majd felszabadítja a memóriát. Ha a felhasználó akarja torolni az autó javításait, egy nagyon hasonlóan működő javitasTorlesRendszamSzerint függvény kerül meghívásra.

#### 5.4 Autó szerviztörténet (7. menüpont)

Itt a főprogram csak előkészíti az adatokat. A rendszám beolvasása után átadja a vezérlést a listázó modulnak (autoSzervizTortenet), paraméterként átadva minden az autók, minden a javítások listájának kezdőcímét. A függvény működése két lépésre oszlik. Először egy kereséssel ellenőrzi az autók listájában, hogy a megadott rendszám egyáltalán létezik-e. Ha nem találja, a függvény kilép. Ha az autó megvan, a program kiírja az adatait, majd elindít egy iterációt a javítások listáján. Végigmegy az összes elemen, és összehasonlítja (strcmp) a bejegyzések rendszámát a keresett rendszámmal. Ahol egyezést talál, azt listázza, és egy jelzőváltozót (volt) állít be. Ha a ciklus végéig nem volt találat, a program külön üzenetben jelzi, hogy az autóhoz még nem volt javítva.

#### 5.5 Lejáró vizsgájú autók listázása (8. menüpont)

A felhasználó megadja a dátumot, amit a program dinamikusan foglal le a beírt karakterek hossza alapján. Ezt a dátum-sztringet adja át a lejaroVizsgak függvénynek, ami elvégzi az összehasonlítást. A dátumok összehasonlításához a program a sscanf függénnyel egész számokká (év, hónap, nap) alakítja a szövegesen tárolt „ÉÉÉÉ-HH-NN” formátumú dátumokat. Ezt követően egy vizsgálatot végez: először az éveket, egyezés esetén a hónapokat, végül pedig a napokat hasonlítja össze. Amennyiben a tárolt vizsgaidőpont ezen logika alapján korábbinak bizonyul a megadott dátumnál, az autó kiírásra kerül. A végén a főprogram felelőssége, hogy ezt a segédterületet felszabadítsa (free), így nem lesz memóriaszivárgás.

#### 5.6 Kilépés és adatmentés (9. menüpont)

Kilépés és adatmentés (9. menüpont) Ez a funkció nemcsak a program futásának megszakítását jelenti, hanem az adatok tartós tárolását is. A kilépési szándék jelzésekor a program először egy megerősítő kérdést tesz fel (I/N). Igen válasz esetén a vezérlés sorban meghívja a mentUgyfelek, ment Autók és mentJavitasok függvényeket, lementik a változásokat a data mappában levő fájlokba, felülírva a régi adatokat. Csak a sikeres mentés után történik az összes adattal foglalt memória felszabadítása. Nem válasz esetén a folyamat megszakad, és a program visszatér a főmenühez.

### 6. A program környezete

A program C nyelven íródott, fordításához GCC vagy más C fordító szükséges. A fejlesztés és tesztelés Visual Studio Code környezetben, Windows operációs rendszeren történt, de a kód más operációs rendszereken is fordítható.