# Programozói dokumentáció

*(Hajdú Patrik Zsolt – RP329D)*

## 1. Projekt Felépítése

A projekt forráskódjai a következő fájlokban találhatóak:

- debugmalloc.h: Segéd fájl a dinamikus memóriakezelés hibáinak detektálására.
- strukt_beolvas.h: Az adatszerkezetek és típusdefiníciók deklarációit tartalmazza.
- seged.h: A programhoz általánosan sok helyen szükséges függvények eléréséhez.
- menu_0.h: A [0.] menüpont használatához.
- menu_1.h: A [1.] menüpont használatához.
- menu_2.h: A [2.] menüpont használatához.
- menu_3.h: A [3.] menüpont használatához.
- menu_4.h: A [4.] menüpont használatához.
- menu_5.h: A [5.] menüpont használatához.
- menu_6.h: A [6.] menüpont használatához.
- main.c: A fő forráskód, amely a programot megvalósítja.

## 2. Fordítás és Környezet

A program C nyelven íródott és szükség van a következő könyvtárakra:

- stdio.h: Be- és kimeneti műveletekhez.
- stdlib.h: Dinamikus memóriakezeléshez.
- string.h: Karakterláncműveletekhez.
- stdbool.h: Logikai típus használatához.
- ctype.h: Karaktervizsgálati műveletekhez.
- time.h: Időkezeléshez.

## 3. Főmenü Dokumentációja

### 4.1 main.c tartalma

#### Adatok beolvasása

Ellenőrzi, hogy a megadott fájl (auto.txt) elérhető-e. Ha nem, hibát ír ki és kilép. A file_beolvas() függvény segítségével beolvassa az autók adatait a fájlból egy dinamikus tömbbe.

#### Főmenü

- Egy végtelen ciklusban fut, ahol a felhasználó menüpontokat választhat.
- A menüpontokat karakterként olvassa be, ellenőrzi, hogy egy szám-e, majd számként tárolja.
- Switch-case szerkezet kezeli a választott menüpontot.

#### Case 0 - Autók rangsorolása

- Kiírja a rangsorolásra vonatkozó almenüt.
- Az elodontendo_kerdes() függvény segítségével kiválasztja, hogy forgalmi vagy olajcsere szerint szeretnénk rangsorolni.
- Rangsorolja az autókat a választott kritérium alapján és kiírja az eredményt.

## Case 1 - Név alapján kiírás

- Bekéri a felhasználótól a keresett névét.
- Meghívja a ki_listazo() függvényt, amely kiírja az összes olyan autót, amelynek a tulajdonosának a neve megegyezik a beírt névvel.

## Case 2 - Rendszám alapján a javítások kiírása

- Bekéri a felhasználótól a keresett rendszámot.
- Ellenőrzi a rendszám helyességét a rendszam_csekkolo() függvénnyel.
- Ha a rendszám helyes, akkor meghívja a ki_listazo() függvényt, amely kiírja az adott rendszámú autó adatait.

## Case 3 - Új autó rögzítése

- Bekéri a felhasználótól az összes adatott, ami szükséges egy teljes értékű az Autó struktúrának megfelelő autó felvételéhez.
- Ellenőrzi az összes adat helyességét.
- Ha az össze adat meglett adva helyesen, akkor ki írja a ki_listazo() függvénnyel, ugyan azokat az adatokat amelyeket fel vesz az „adatok_tomb”-be.

## Case 4 - Javítás hozzáadása egy autóhoz

- Kezdésnek bekéri a felhasználótól a rendszámot.
- Bekéri a felhasználótól az összes adatott, ami szükséges egy teljes értékű az Javítás struktúrának megfelelő javítás felvételéhez.
- Majd hozzá rendeli a bekért rendszámhoz tartozó autóhoz a javítását.
- Ellenőrzi az összes adat helyességét.
- Ha az össze adat meglett adva helyesen, akkor ki írja a ki_listazo() függvénnyel, ugyan azokat az adatokat amelyeket fel vesz az „adatok_tomb”-be.

## Case 5 - Meglevő autó törlése

- Bekéri a felhasználótól egy autó rendszámát, amelynek az összes adatát törölni szeretné a felhasználó.
- Ellenőrzi a rendszám helyességét a rendszam_csekkolo() függvénnyel.
- Ha a rendszám helyes, akkor kiírja az adott rendszámú autó adatait sikeresen töröltük az „adatok_tomb” -ből.

## Case 6 - Fájlba kiírás

- Ki írja az „auto_adatok.txt” fájlba az „adatok_tomb” -ben aktuálisan bent lévő autókat.
- Ugyan ezt a konzolra is ki írja!

## Case 7 - Kilépés

Kiírja a kilépési üzenetet és befejezi a programot.

## Memóriakezelés

A program végén felszabadítja a dinamikus memóriát, amit az autók és azok javításainak a tárolására használt.

## Hibakezelés

- Az autók adatainak beolvasásakor ellenőrzi, hogy a fájl (auto.txt) elérhető-e. Ha a fájl nem található vagy nem olvasható, a program kiír egy hibaüzenetet és kilép.
- Ha a felhasználó olyan karaktert ad meg, amely nem 0, 1, 2, 3, 4, 5, 6, 7 a program kiír egy hibaüzenetet, tájékoztatva a helytelen bemenetről, majd visszatér a főmenübe.

# 4. Adatszerkezetek Dokumentációja

## 4.2 strukt_beolvas.c / strukt_beolvas.h tartalma

### Autó struktúra

Az Autó struktúra az autók adatainak tárolására szolgál. Az alábbi mezőket tartalmazza:

- auto_tulaja: Tulajdonos neve és telefon száma.
- rendszam: Az autó rendszámát tartalmazó, csak 7 vagy 8 karakter hosszúságú karaktertömb.
- auto_tipusa: Az autó típusát tartalmazó, maximum 40 karakter hosszúságú karaktertömb.
- elozo_muszaki_idopontja: Az előző műszaki vizsga időpontja.
- elozo_olaj_csere_km: Az előző olajcsere kilométerszáma.
- elozo_olaj_csere_idopontja: Az előző olajcsere időpontja.
- alvaz_szam: Az autó alvázszámát tartalmazó, csak 22 karakter hosszúságú karaktertömb.
- km_ora_allas: Az autó kilométeróra állása.
- javitasok_szama: Az eddigi javítások száma.
- eddigi_javitasok: Javításokat tartalmazó dinamikus tömb.

### Javítás struktúra

A Javítás struktúra egy javítás részleteit tárolja:

- mikor: A javítás időpontja.
- javitas_tipusa: A javítás típusát tartalmazó, maximum 40 karakter hosszúságú karaktertömb.
- anyag_koltseg: A javításhoz felhasznált anyagok költsége.
- javitas_osszege: A javítás összköltsége.

### Menü Opció struktúra

A Menü Opció struktúra egy olyan adatszerkezet, amely egy menü opció karakterét és leírását tárolja:

- karakter: Az opcióhoz tartozó karakter, amelyet a felhasználó választhat.
- leiras: Az opcióhoz tartozó szöveges leírás.

### Dátum struktúra

A Dátum struktúra egy olyan adatszerkezet, amely egy dátumot tárol:

- ev: Az év értéke.
- honap: A hónap értéke.
- nap: A nap értéke.

### Tulajdonos struktúra

A Tulajdonos struktúra az autó tulajdonosának adatait tárolja:

- nev: A tulajdonos nevét tartalmazó, maximum 40 karakter hosszúságú karaktertömb.
- telefon_szam: A tulajdonos telefonszámát tartalmazó, csak 11 karakter hosszúságú karaktertömb.

### File_beolvas()

Ez a függvény kap egy sztring bemenetet, ami a file neve, majd egy tömböt, amibe bele tölti az „auto.txt” -ből a beolvasott adatokat. Végül pedig egy int változóban eltárolja hány darab a struktúrának megfelelő autó adatait olvasta be.

### Használat:

```
void file_beolvas(const char *filenev,  Auto **autok, int *elemek_szama);
```

Példa:

```
file_beolvas(fajlnev, &adatok_tomb, &elemek_szama);
```

## 5. Függvények Dokumentációja

### 4.3 seged.c / seged.h tartalma:

#### nem_talalt_adat() - függvény

Ez a függvény felelős azért, hogy értesítse a felhasználót, amikor a keresett adatot nem találja meg az "adatbázisban" (tömbben). A függvény egy színes, de könnyen olvasható és felismerhető üzenetet jelenít meg.

#### helytelen_adat() - függvény

Ez a függvény felelős azért, hogy értesítse a felhasználót, amikor a megadott adat hibásan lett megadva. A függvény egy színes, de könnyen olvasható és felismerhető üzenetet jelenít meg.

#### konzol_torles() – függvény

Ez a függvény felelős a konzol kiürítéséért, és platformfüggetlen módon működik mind Windows, mind Linux rendszereken. A system(„cls”) parancsot használja Windows rendszerekhez, és a system(„clear”) parancsot Linux rendszerekhez.

#### tovabb() – függvény

Ez a függvény kap egy sztring bemenetet, amit ki ír. Majd vár egy billentyűleütésre, ami kiüríti a bemeneti puffert és a konzolt. Ezáltal a felhasználó kényelmesen folytathatja a program használatát.

Használat:

```
void tovabb(char *szoveg_ki_ir)
```

Példa:

```
tovabb("\033[0;32mNyomj le egy billentyut a folytatashoz!");
```

#### karakter_csere() – függvény

Ez a függvény egy karakterláncot kap bemenetként, és minden „mit” karaktert kicserél az „mire” karakterre. Ez hasznos lehet, ha a felhasználó olyan adatot ad meg, amely nem tartalmazhat egy adott karaktert a struktúrában.

Használat:

```
void szokoz_torlo(char *keresett, const char *mit, char mire)
```

Példa:

```
szokoz_torlo(nev, " ", '_');
```

#### puffer_urites() – függvény

Ez a függvény egy üzenetet nyomtat ki, majd vár egy billentyűlenyomásra. Miután a felhasználó lenyom egy billentyűt, a getchar (beépített) függvény segítségével kiüríti a bemeneti puffert, és végül a konzol_torles() függvényt hívja meg.

## eldontendo_kerdes() – függvény

Ez a függvény egy kérdést és két lehetséges választ kínál fel a felhasználónak. A felhasználó választásának karakterét visszatéríti.

### Használat:

```
int eldontendo_kerdes(char *kerdes, MenuOpcio *menu, int menu_meret)
```

### Példa:

```
MenuOpcio menu_of [ ] = {
   {’o’, "Olajcsere"},
   {'f', "Forgalmi"},
};

int menu_of_meret = sizeof(menu_of) / sizeof(menu_of[0]);
int valasztott_opcio_of = eldontendo_kerdes("Valassz egy opciot, ami alapjan rangsorolni szeretnel:",
menu_of, menu_of_meret);
```

## char *rendszam_beker() – függvény

Ez a függvény addig kér be egy rendszámot míg helyesen nem lesz megadva.

### Használat:

```
char *rendszam_beker(Auto *adatok_tomb, int elemek_szama, const char* cim, const char*
mit_ker_be)
```

### Kimenet:

A függvény visszatérési értéke egy rendszám.

### Példa:

```
char *rendszam = rendszam_beker(adatok_tomb, elemek_szama, "\033[1;0mMelyik rendszamu
autohoz szeretnel uj javitast felvenni: ", "");
```

## void datum_beker() – függvény

Ez a függvény addig kér be egy dátumot míg helyesen nem lesz megadva.

### Használat:

```
int eldontendo_kerdes(char *kerdes, MenuOpcio *menu, int menu_meret)
```

### Példa:

```
datum_beker(uj_javitas, "\t\033[1;0m>>> Javitasok <<<\n", "\tJavitas idopontja (ev/honap/nap): ",
"mikor");
```

## void beker_szamot() – függvény

Ez a függvény addig kér be egy számot míg helyesen nem lesz megadva.

### Használat:

```
void beker_szamot(void *hova, const char* cim, const char* mit_ker_be, const char*
milyen_tipusba)
```

Példa:

```
beker_szamot(uj_javitas, "\t\033[1;0m>>> Javitasok <<<\n", "\tAnyag koltseg: ", "anyag_koltseg");
```

### void beker_szoveg() – függvény

Ez a függvény addig kér be egy maximum 41 karakter hosszú sztringet míg helyesen nem lesz megadva.

Használat:

```
void beker_szoveg(void *hova, const char* cim, const char* mit_ker_be, const char*
milyen_tipusba)
```

Példa:

```
beker_szoveg(uj_javitas, "\t\033[1;0m>>> Javitasok <<<\n", "\tJavitas tipusa: ", "javitas_tipusa");
```

---

## 4.4 menu_0.c / menu_0.h tartalma

Ebben a szakaszban a 0. menüpontot megvalósító függvények dokumentációját találod. Ezek a függvények a felhasználói menü 0. pontjához kapcsolódnak, és az autók rangsorolását végzik el különböző szempontok alapján.

### datum_osszehasonlito() - függvény

Ez a függvény összehasonlítja két dátumot és visszatér az összehasonlítás eredményével. A visszatérési érték lehet -1, 0 vagy 1, attól függően, hogy az első dátum előbbi, azonos vagy későbbi, mint a második dátum.

Használat:

```
int datum_osszehasonlito(struct Datum datum_1, struct Datum datum_2);
```

Példa:

```
struct Datum d1 = {2023, 11, 15};
struct Datum d2 = {2023, 11, 20};

int eredmeny = datum_osszehasonlito(d1, d2);
```

### auto_asc_or_desc() – függvény

Ez a függvény az autók összehasonlítására szolgál. Az összehasonlítás az előző olajcsere vagy a műszaki vizsga időpontja alapján történik.

Használat:

```
int auto_asc_or_desc(const void *auto_1, const void *auto_2);
```

Példa:

```
qsort(adatok_tomb, elemek_szama, sizeof(struct Auto), (int (*)(const void *, const void
*))auto_asc_or_desc);
```

### kiir_auto() – függvény

Ez a függvény egy autó adatait írja ki a konzolra a „szures_modja” alapján.

Használat:

```
void kiir_auto(const struct Auto *auto_adat, int szures_modja)
```

Példa:

```
for (int i = 0; i < ideiglenes_index; i++) {
    kiir_auto(&ideiglenes_tomb[i], szures_modja);
}
```

## idopont_ellenorzo() – függvény

Ez a függvény ellenőrzi, hogy az aktuális időpontban és az előző vizsga időpontja között eltelt-e a megadott időtartam. A vizsgálat típusát a global_szures_modja változó határozza meg.

Használat:

```
int idopont_ellenorzo(struct Datum idopont);
```

Példa:

```
struct Datum idopont = {2022, 10, 15};
int eredmeny = idopont_ellenorzo(idopont);
```

## rendez_auto_tomb() – függvény

Ez a függvény rendez egy autótömböt az előző olajcsere vagy műszaki vizsga időpontja alapján, és kiírja azokat az autókat, amelyeknek lejárt a vizsgájuk vagy elértek az olajcsere kilométertartamukat.

Használat:

```
void rendez_auto_tomb(struct Auto *adatok_tomb, int elemek_szama, int szures_modja, int
rendezes_modja);
```

Példa:

```
MenuOpcio menu_of[] = {
    {'o', "Olajcsere"},
    {'f', "Forgalmi"},
};
int menu_of_meret = sizeof(menu_of) / sizeof(menu_of[0]);
int valasztott_opcio_of = eldontendo_kerdes("Valassz egy opciot, ami alapjan rangsorolni szeretnel:",
menu_of, menu_of_meret);

MenuOpcio menu_nc[] = {
    {'n', "Novekvo"},
    {'c', "Csokkeno"}
};
int menu_nc_meret = sizeof(menu_nc) / sizeof(menu_nc[0]);
int valasztott_opcio_nc = eldontendo_kerdes("Valassz egy opciot, ami alapjan rangsorolni
szeretnel:", menu_nc, menu_nc_meret);

rendez_auto_tomb(adatok_tomb, elemek_szama, valasztott_opcio_of, valasztott_opcio_nc);
```

Hibakezelés:

Amennyiben az autók tömbje üres, vagy nincs olyan autó, amelynek lejárt volna a vizsgája vagy elérte az olajcsere kilométertartamát a szűrési feltételek alapján, a függvény ezt jelzi a felhasználónak egy megfelelő

üzenettel. A hibaüzenet tájékoztatja a felhasználót arról, hogy nincs találat a keresésre, és javaslatot tesz arra, hogy módosítsa a szűrési feltételeket.

---

## 4.5 menu_1.c / menu_1.h tartalma

Ebben a szakaszban a 1. menüpontot megvalósító függvények dokumentációját találod. Ezek a függvények a felhasználói menü 1. pontjához kapcsolódnak, és az autók keresett tulajdonságait listázzák ki.

### ki_listazo() - függvény

Ez a függvény ki listázza egy keresett név (mi_alapjan = „n”) vagy rendszámhoz (mi_alapjan = „r”) tartozó autó információját. Ha megtalálja a keresett nevet vagy rendszámot az autók között, akkor kiírja az autó tulajdonosának adatait és ha (mit_ki_ir = „1”) az autó részletes információit, ha (mit_ki_ir = „2”) az autó javításainak részletes információit. Végül, ha (mit_ki_ir = „0”) mind kettőt ki irja!

### Használat:

```
bool ki_listazo(FILE *file, Auto *adatok_tomb, int elemek_szama, const char *keresett, const int mit_ki_ir, const char mi_alapjan)
```

### Kimenet:

A függvény visszatérési értéke egy logikai érték (true vagy false), amely jelzi, hogy talált-e egyezést vagy sem.

### Példa:

```
if (!ki_listazo(stdout, adatok_tomb, elemek_szama, nev, 0, 'n')) {
    nem_talalt_adat();
}
```

### Hibakezelés:

Amennyiben a keresett név vagy rendszám alapján nem található autó az adatok között, a függvény false értékkel tér vissza. Ebben az esetben a felhasználó értesítést kap arról, hogy nincs találat a keresésre, és javaslatot kap arra, hogy ellenőrizze a keresési feltételeket.

A hibaüzenet segítséget nyújt a felhasználónak abban, hogy módosítsa a keresést, vagy ellenőrizze az adatokat.

Ha a tulajdonosa megegyezik a keresett névvel, vagy a rendszáma a függvény kiírja a keresett személy, rendszámhoz tartozó információkat.

---

## 4.6 menu_2.c / menu_2.h tartalma

Ebben a szakaszban a 2. menüpontot megvalósító függvény dokumentációját találod. Ezek a függvények a felhasználói menü 2. pontjához kapcsolódnak, és a rendszám validációjáért felelős.

### rendszam_csekkolo() – függvény

Ez a függvény ellenőrzi egy adott rendszám helyességét a magyar rendszámtáblák szabályai szerint. Ellenőrzi a hosszt, a karakterek helyességét, valamint a tiltott karakterpárokat a rendszám első és második karaktereiben.

### Használat:

```
bool rendszam_csekkolo(char *rendszam, int rendszam_hossza);
```

### Kimenet:

A függvény visszatérési értéke egy logikai érték (true vagy false), amely jelzi, hogy a rendszám helyes vagy sem.

Példa:

```
char rendszam1[] = "ABC-123";
bool eredmeny1 = rendszam_csekkolo(rendszam1, strlen(rendszam1));

char rendszam2[] = "XYZ-W23";
bool eredmeny2 = rendszam_csekkolo(rendszam2, strlen(rendszam2));
```

Hibakezelés:

Ez a függvény ellenőrzi egy adott rendszám helyességét a magyar rendszámtáblák szabályai szerint. Ellenőrzi a hosszt, a karakterek helyességét, valamint a tiltott karakterpárokat a rendszám első és második karaktereiben.

A függvény több kritériumot is ellenőriz, beleértve a karakterek típusát, a tiltott karakterpárokat, a kötőjelet és a számjegyeket.

Ha a rendszám megfelel az összes szabálynak, a függvény true-t ad vissza, egyébként false-t, és közben egy hibaüzenetet is kiír.

---

## 4.7 menu_3.c / menu_3.h tartalma

Ebben a szakaszban a 3. menüpontot megvalósító függvény dokumentációját találod. Ezek a függvények a felhasználói menü 3. pontjához kapcsolódnak, és egy új autó rögzítéséért felelős.

### char *uj_auto_felvevo() – függvény

Egy olyan függvény, amelynek lényege, hogy a konzolon egy értelműen leírja, hogy milyen adatott vár a felhasználótól. Ami után a felhasználó megadta az adatott, megkezdődik a hiba kezelés! Ha rosszul adja meg a felhasználó meg mondja mit hiba okát és el és újra bekéri. Ez mind addig megy míg a felhasználó nem add meg minden autóhoz tartozó változóba helyes adatott.

Használat:

```
char *uj_auto_felvevo(Auto *adatok_tomb, int elemek_szama, Auto *uj_auto);
```

Kimenet:

A függvény visszatérési értéke char* ami rendszám lesz, ami azért szükséges, hogy ki tudjuk írni a felvett autónak az adatait a képernyőre, hogy a felhasználó láthassa.

Példa:

```
struct Auto *uj_auto = (struct Auto *)malloc(sizeof(struct Auto));

if (uj_auto == NULL) {
   perror("Nem sikerult memoriat allokalni!");
   exit(EXIT_FAILURE);
}

char *temp_3 = uj_auto_felvevo(adatok_tomb, elemek_szama, uj_auto);
```

Hiba kezelés:

**Példa 1:**

Milyen hibák léphetnek fel egy **telefonszám** megadásánál:

```
tovabb("\033[0;31mHibas bemenet! Telefonszam hossza nem megfelelő!");
tovabb("\033[0;31mHibas bemenet! A telefonszam nem tartalmazhat ékezetes karaktert vagy betűt.
Add meg újra!");
tovabb("\033[0;31mA telefonszam mar szerepel az adatbázisban. Adj meg egy másikat!");
```

Az Adatszerkezetek Dokumentációja-ból kiderül egy telefonszámot mindig csak akkor fogadható el helyesnek, ha 11 számból áll!

**Példa 2:**

Milyen hibák léphetnek fel egy **rendszám** megadásánál:

```
tovabb("\033[0;31mHibas bemenet! A rendszam túl rövid/hosszú lett. Add meg újra!");
tovabb("\033[0;31mA rendszam mar szerepel az adatbázisban. Adj meg egy másikat!");
tovabb("\033[0;31mHibas bemenet! A rendszam nem felel meg a magyar hatósági előírásoknak. Add meg ujra!");
```

Itt szintén az Adatszerkezetek Dokumentációja-ból kiderül, hogy egy rendszám csak akkor fogadható el helyesnek, ha 7 vagy 8 hosszúságú!

### Auto *auto_tomb_bovites () – függvény

Dinamikusan növeli az Autó típusú struktúrákból álló tömb méretét. A függvény először megnöveli a „elemek_szama” értékét egyel, majd a realloc (beépített) függvény segítségével növeli a dinamikusan lefoglalt terület méretét.

#### Használat:

```
Auto *auto_tomb_bovites (Auto *adatok_tomb, int *elemek_szama)
```

#### Kimenet:

A függvény visszatérési értéke az „adatok_tomb” új címére mutatói pointer.

#### Példa:

```
adatok_tomb = auto_tomb_bovites(adatok_tomb, &elemek_szama);
adatok_tomb[elemek_szama - 1] = *uj_auto;
free(uj_auto);
```

---

## 4.8 menu_4.c / menu_4.h tartalma

Ebben a szakaszban a 4. menüpontot megvalósító függvény dokumentációját találod. Ezek a függvények a felhasználói menü 4. pontjához kapcsolódnak, és egy új javítás rögzítéséért felelős.

### char *uj_auto_felvevo() – függvény

Egy olyan függvény, amelynek lényege, hogy a konzolon egy értelműen leírja, hogy milyen adatott vár a felhasználótól. Ami után a felhasználó megadta az adatott, megkezdődik a hiba kezelés! Ha rosszul adja meg a felhasználó meg mondja mit hiba okát és el és újra bekéri. Ez mind addig megy míg a felhasználó nem add meg minden javításhoz tartozó változóba helyes adatott. Majd a függvény egy adott rendszámhoz tartozó autó adatainak megfelelő javítások listájához hozzáad egy új javítást. A függvény először végigmegy az autók tömbjén, és megkeresi azt az autót, amelynek a rendszáma megegyezik a megadott rendszam-mal. Ha megtalálja az autót, akkor a következő lépéseket hajtja végre:

- Növeli az autó javításainak számát („javitasok_szama”) eggyel.
- Használja a realloc (beépített) függvényt az „eddigi_javitasok” tömb méretének növelésére az új javítás számának megfelelően. Ez biztosítja, hogy elegendő hely legyen az új javítás tárolásához.
- Ellenőrzi, hogy a realloc sikeres volt-e. Ha nem, hibaüzenetet ír ki, és kilép a programból.
- Másolja az új javítás adatait a megfelelő helyre az „eddigi_javitasok” tömbben a memcpy (beépített) függvény segítségével.

### Használat:

```
char *uj_javitas_felvevo (Auto *adatok_tomb, int elemek_szama, Javitas *uj_javitas)
```

### Kimenet:

A függvény visszatérési értéke char* ami rendszám lesz, ami azért szükséges, hogy ki tudjuk írni a felvett javítási adatokat a képernyőre, hogy a felhasználó láthassa.

### Példa:

```
struct Javitas *uj_javitas = (struct Javitas *)malloc(sizeof(struct Javitas));

if (uj_javitas == NULL) {
    perror("Nem sikerult memoriat allokalni!");
    exit(EXIT_FAILURE);
}

char *temp_4 = uj_javitas_felvevo(adatok_tomb, elemek_szama, uj_javitas);
```

### Hiba kezelés:

**Példa 1:**

Milyen hibák léphetnek fel a **javítás időpontjának** megadásánál:

```
tovabb("\033[0;31mHibas bemenet! Kérem, adja meg helyes formátumba az értékeket!
év/hónap/nap");
```

Az Adatszerkezetek Dokumentációja-ból kiderül egy javítás időpont Dátum típusú, amely 3 darab int-ből adódik össze. Ennek helyes beolvasásáért a scanf("%d/%d/%d",..); felel.

**Példa 2:**

Milyen hibák léphetnek fel egy **javítás összege** megadásánál:

```
perror("\033[0;31mHiba történt az adatok beolvasása során.");
tovabb("\033[0;31mHibas bemenet! A javítás összege csak számot tartalmazhat. Add meg újra!");
```

Azért használunk perror()-t, a printf() helyet mert fgets-el van az adat beolvasva.

---

## 4.9 menu_5.c / menu_5.h tartalma

Ebben a szakaszban a 5. menüpontot megvalósító függvény dokumentációját találod. Ezek a függvények a felhasználói menü 5. pontjához kapcsolódnak, és egy autó az összes információjával együtt való törléséért felelős.

### auto_torlo () – függvény

Egy olyan függvény, amelynek lényege, bekéri annak az autónak a rendszámát, amelynek az összes adatát törölni szeretnénk. Majd kitörli az „adatok_tomb” -ből a keresett rendszám alapján minden információt róla. A kitörlőt elemnyi helyen pedig a felesleges memória szivárgást kiküszöböli. Úgy, hogy az „adatok_tomb” -ből a törlőt autó adatainak méretével kisebb memória területet foglal.

### Használat:

```
void auto_torlo(Auto **adatok_tomb, int *elemek_szama)
```

### Példa:

```
auto_torlo(&adatok_tomb, &elemek_szama);
```

## 4.10 menu_6.c / menu_6.h tartalma

Ebben a szakaszban a 6. menüpontot megvalósító függvény dokumentációját találod. Ezek a függvények a felhasználói menü 6. pontjához kapcsolódnak, és egy új autó rögzítéséért felelősek.

### void ki_ir_fileba () – függvény

Egy olyan függvény, amelynek lényege, hogy az „adatok_tomb” teljes tartalmát ki írja az „auto_adatok.txt” -be.

Használat:

```
void ki_ir_fileba(Auto *adatok_tomb, int elemek_szama);
```

Példa:

```
ki_ir_fileba(adatok_tomb, elemek_szama);
printf("Az \"auto_adatok.txt\"-be lett ki irva az adatok_tomb jelenlegi adatai!\n");
```

## 6. A program vége ([7] menüpont)

A programból való kilépéséért felel. Ezt az opciót választva bezáródik a (do while /switch) úgy nevezet „menü” ciklus, amivel vége a programnak és felszabadítja az „adatok_tomb” -öt.