Program neve: hazi.c

Program lényege:

Kiszámolja egy autó átlagfogyasztását a felhasználó által megtett út és a fogyasztott üzemanyag mennyiségéből.

Lehetőség van (/lesz) a gépjármű adatainak tárolására:

- Autó neve (becenév is lehetséges)
- Üzemanyag fajtája (benzin, diesel)
- Tankolás helye
- Rendszám

A fentebb említett adatokat első alkalommal, vagy új jármű hozzáadása esetén a felhasználónak kell megadnia. A felhasználónak minden esetben megkell adnia a megtett km-t. Lekérdezés esetén ezeket az adatokat egy már korábban mentett file-ból olvassa be a program.

A program kiszámolja, egy előre letöltött adatbázisból, mely a benzinkutak egységárait (és neveit) tartalmazza, hogy melyik üzemanyag forgalmazó cégnél mennyibe került a felhasználó által megadott km-en fogyott benzin vagy gázolaj ára. Ezeket az adatokat, (hol tankol, mennyiért, mennyit) eltárolja egy txt fileban, ami az autó egyéb (rendszám, név, stb.) adatai mellett lesznek tárolva egy nagy adatszerkezetként. A benzinkutas adatbázis segítségével lehetőség lesz kutankénti csoportosításra, akár üzemanyag fajták szerint is.

Továbbá a program kilistázza a szenderd outputra a beadott autó adatait és a fentebb kiszámolt fogyasztást és, hogy melyik kúton mennyibe kerül ugyan ennyi benzin (gázolaj). A program számontartja, hogy melyik benzinkúton tankltunk a legtöbbet, mindegy, hogy melyik autó adatait íratni. A benzinkutas adatbázis egy txt vagy excel adathalmaz és a programmal együtt érkezik egy mappában, ennek adatait a felhasználó az exe file futása közben nem tudja (nem kéne) módosítani. Tankolási log vezetésére is lehetőség lesz egy külön file-ban, ahol előző tankolással kapcsolatos információk lesznek elmentve.

Az adatok típúsai:

- Autónév = string;
- Üzemanyag típus = char (pl.: benzin esetén „b”)
- Tankolás helye = string;
- Rendszám = string;
- Km = int;
- Benzin ár = int;
- Átlagfogyasztás = valós;

A program 3 fájlt használ melyekben a következő adatok vannak eltárolva a következő sorrendben:

auto_data.txt

AAA111 ➔ rendszám, string

teszt ➔ Auto neve, string

b ➔ Üzemanyag típus (b: benzin, d: diesel), char

…

tank_log.txt

34000 ➔ Tankolás előtt megtett kilométeróra állás, int

Mol ➔ Benzinkút neve, string

b ➔ Üzemanyag típus (b: benzin, d: diesel), char

40 ➔ Tankolt üzemanyag mennyisége

kut.txt

OMV ➔ Benzinkut neve, string

480 ➔ Aktuális benzinkút 95-ös benzinár, int

520 ➔ Aktuális benzinkút 100-as benzinár, int

482 ➔ Aktuális benzinkút diesel ár, int

A programban ezeket a fájlokat listák kezelik, mindegy egyes fájlhoz külön-külön.

Az auto_data.txt fájlhoz a következő struktúra tartozik:

```
typedef struct autok
{
   char rendszam[10];
   char uzemanyag;
   char nev[20];

   struct autok *next;
}autok, *autok_ptr;
```

A tank_log.txt fájlhoz a következő struktúra tartozik:

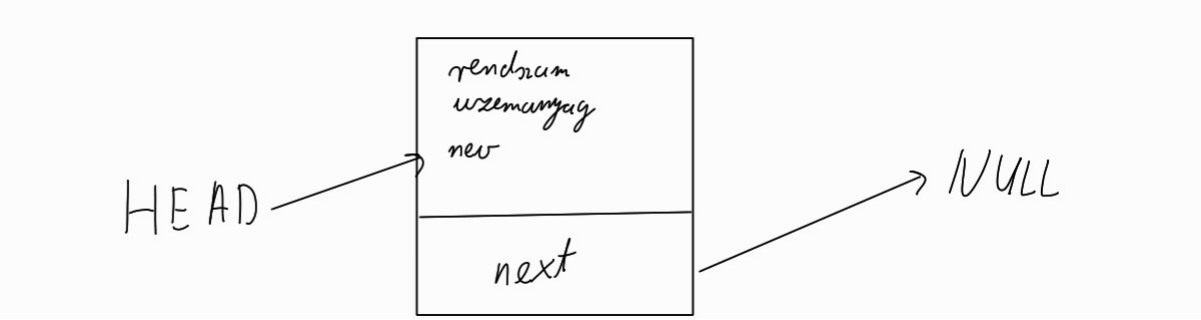
```
typedef struct uzemA
{
   char benzinkut[20];
   char uzemanyag;
   int uzema_meny;
   int km;
   struct uzemA *next;
}uzemA_List, uzemA_ptr;
```

A kut.txt fájlhoz a következő struktúra tartozik:

```
typedef struct kutak
{
    char benzinkut[20];
    int b95;
    int b100;
    int Diesel;

    struct kutak *next;
}kutak, *kutak_ptr;
```

Mindegyik struktúrát láncolt listaként használom a következő egyszerűsített példa szerint:



Algoritmus leírás:

Indításkor a main meghívja a progValasztas() függvényt amely köszönti a felhasználót, majd egy int változót kér be. A megfelelő szám beírása után a progValasztas() függvény vissza adja a számot a progVal változónak.

A progValazstas() függvény vizualizása:

progValasztas() függvény int visszatérésü függvény ami void típust kap pararméterként

Kiírja a lehetőségeket a következő féle módon:

```
Adja meg mit szeretne tenni

Auto hozzadas eseten irja be az 1-es szamot
Atlag fogyasztas szamlalohoz 2-es
Tankolas mentese, 3-es
Legtobbszor tankolt hely megtekintese 4-es
```

Ezekután beolvassa a felhasználó által adott számot.

Megfelőszám beírása után a függvény visszaadja az értéket a main függvénynek.

A program() függvény leírása:

A program függvény felelős a további függvények meghívásáért. Innen indul ki minden. Maga a függvény, amennyire fontos feladatot lát el, annyira egyszerű. A következő módon működik:

A függvény bekér egy inttet és egy, az auto_data.txt-hez tartozó struktúrájó, autok* head-et

switch case segítségével eldönti a program, hogy a négy beprogramozott program közül melyiket hívja meg

4-nél nagyobb vagy 1-nél kisebb szám esetén a függvény újra meghívja a progValaszatas függvényt, hogy a felhasználú új számot tudjon megadni

A függvény miután sikeresen meghívta a megfelelő függvényeket, visszatérési értékként egy autok* struktúrájú elemet ad vissza a main függvénynek. Ez a lépés egyfajta kontroll ként működik, ezzel biztosítva hogy a meghívott függvény működött és lefutott. Ha esetleg NULL pointert adna vissza, akkor az többek között azt jelenti, hogy a program egy olyan fájlt akart megnyitni ami nem létezik.

A program() által visszaadott autok* struktúrát a main-ben meghívott deleteL() függvény törli és szabadítja fel az általa lefoglalt memóriát.

A különböző struktúrákat különböző függvények törlik. Mindegyik ugyan azon elvre épül, csak a struktúra típusa különbözik. Mindegyik void visszatérésű:

deleteL(autok* head), deleteLog(uzemA_List* head), deleteK(kutak* head).

deleteL() függvény megkapja az autok* stuktúrájú láncolt lista első elemének címét

Megvizsgálja, hogy a következő cím az a NULL pointer e?

Ha igen akkor az aktuális címet felszabadítja és ráállítja a NULL pointerre.

Ha nem akkor pedig egy while cikluson belül egy segéd p pointerrel végigjárja a listát és az elejétől a végéig minden elemet felszabadít.

deleteL() függvény részletesebb leírása:

A program fájlokból való olvasásáért és fájlokba való írásáért összesen 5 függvény felelős, 3 csak olvasásra, 1 csak írásra, 1 írásra és olvasásra. Ez azért van így, mert a kut.txt fájl az egyetlen olyan, amit nem tud a felhasználó módosítani.

Az olvasó programok egy láncolt listát készítenek a beolvasott adatokból. Az olvasás minden esetben hasonló módon történik. A read_L() függvény példaként:

read_L() függvény egy uzemA_List* head-et kap paraméterként és uzemA_List stuktúrájú láncolt listát ad vissza.

A függvény megnyitja a tank_log.txt fájlt olvasásra, majd megvizsgálja, hogy létezik e a fájl. Ha igen akkor folytatja, ha nem akkor NULL pointert ad vissza.

A függvény egy while ciklussal addig olvas a fájlból ameddig tud.

A ciklus belsejében létrehoz egy ideiglenes változót aminek segítségével létrehozza a listát. Beolvassa az adatokat a következő sorrendben:

1, Fájlban mentett kilóméter
2, Lementett benzinkút neve
3, Uzemanyag típúsa
4, Uzemanyag mennyisége

Miután egy láncelemnyi adatot beolvasott utánna a paraméter listán kapott pointerre ráálítja az ideiglenes pointerre, majd ezt a folyamatot ismétli.

Miután a ciklus véget ért a függvény bezárja a fájlt és visszaadja a láncolt lista első elemének címét.

A többi listát a program szükség szerint hozza létre, a fentebb részletezettekhez hasonlóan.

Fájlokba írás a következőképpen történik: (példa save_Log)

A save_Log() függvény paraméterként egy uzemA_List struktúrájú láncolt listának az első címét kapja meg paraméterként, visszatérési értéke void típúsu.

Első lépésben megnyittja a tank_log.txt fájlt append módon. Ez alatt azt kell érteni, hogy ami eddig belevolt írva a fáljba az nem változik viszont új adatot ettől függetlenül lehet hozzá írni. Minden új adat a fájl végén található.

A függvény megvizsgálja, hogy létezik e a fájl. Ha nem létezik akkor visszatér oda ahonnan meghívták.

Ha a fájl létezik akkor a függvény megvizsgálja, hogy a paraméter listán kapott pointer az NULL pointer-e? Ha igen akkor kiírja standard outputra, hogy nincsen mit kimenteni, hiszen NULL pointerre mutat a lista első eleme. Ezek után visszatér oda ahonnan meghívták

Ha létezik a fájl és a lista sem üres akkor egy while ciklussal végig megy a listán egészen addig, amíg NULL pointerrel nem találkozik. A végigjárás közben kiírja program a fájlba az adatokat a következő sorrendben:

1, Fájlban mentett kilóméter
2, Lementett benzinkút neve
3, Uzemanyag típúsa
4, Uzemanyag mennyisége

Miután végzett bezárja a fájlt.

A prog1, prog2, prog3, prog4 függvények mind a fentebb említett függvények segítségével működnek. A prog1 felelős az autók hozzáadásához az auto_data.txt fájlba. A prog2 számolja ki az átlagfogyasztást, vagy tank_log.txt segítségével, vagy pedig a felhasználó által beírt adatok segítségével. Ehhez egy atlagF() segéd függvényt hív meg.

Az atlagF() függvény paraméterlistán kap egy tankolás előtti kilométeróra állást, egy tankolás utánni kilométeróra állást illetve, hogy mennyit tankolt összesen.

A függvény visszatérési értéke az átlag fogyasztás, ami egy valós szám.

A következő képlet alapján számol:

A tankolt üzemanyag mennyiséget elosztja a két km-óra állás különbségével, majd ezt az egészet beszorozza 100-al.

C- beli megvalósítása:

```
atlag = benzin / (km2 - km1);
atlag *= 100;
```

Ezek után a függvény visszatér a kiszámolt értékkel

A prog3 felelős a tankolás mentéséért. A prog4 összegzi, hogy melyik volt az a hely ahol a legtöbbször voltunk, és azt hogy a második legtöbb helyen mennyit fizettünk. Azért a második legtöbb, mert a kedvenc helyén elköltött pénzre inkább nem kíváncsi senki.

Tesztelés dokumentálása:

Kezdetben a prog1 függvényt teszteltem. Egy láncolt lista elemmel. A függvény sikeresen létrehozta a láncolt listát, de a fájlba írással voltak gondok kezdetben. Mint később kiderült rossz adatokat írt ki a program ami a rossz láncolásnak volt betudható. Miután egy elemre működött teszteltem 2 elemre. 2 elemre működött, ahogy 3 és 4 elemre. Következésképpen a függvény megbízhatóan működik. Miután sikeresen működik a lista létrehozása és fájlba való kiírása, a listatörlő függvényt teszteltem. Kezdetben úgy tűnt, hogy a törlő függvényeim is jól működnek, de miután leellenőriztem, hogy van e memória szivárgás, az infoc.bme.hu oldalról letöltött debugmalloc.h könyvtár segítségével, kiderült, hogy mégsem megfelelő a működésük. A problémát egy lefelejtett kapcsoszárójel okozta, aminek pótlása után a törlő függvények megfelelően működtek.

Következőnek az átlagfogyasztásért felelős függvényeket teszteltem. A prog2, ami többek között eldönti, hogy fájlból, vagy standard inputról vett adatokkal szeretnénk számolni, kezdetben nem működött. A problémát a rossz időben való memória foglalás okozta. Ezek után a prog2 függvény ezen része megfelelően működött, de a másik fele, ami az atlagF() függvényt hívja meg azzal voltak gondok továbbra is. Egyik ilyen gond az volt, hogy rossz sorrendben kapta meg az atlagF() függvény az adatokat. Mindezek kiküszöbölése után a programot teszteltem különböző nagyságú adatokkal, ami megfelelő működést eredményezett.

A tankolás mentésért felelős prog3 függvény elsőre hiba nélkül lefutott. De a mentésért felelős függvények, amiket a prog3 függvény hív meg, azok kezdetben rossz sorrendben írták ki az adatokat a fájlba. Miután ezt kijavítottam, minden, ami a prog3-hoz tartozik megfelelően működött.

A prog4 függvény felelős a tank_log.txt összegzéséért. A függvény már az első tesztelésnél is szintaktika és szemantikai hibáktól mentes volt, de számolási hibát tartalmazott, aminek a rossz sorrendbeli összeadás és szorzás volt az oka. (Mert nem mindegy, hogy előbb összeadok és utána szorzok vagy fordítva). Ezen problémák kijavítása után a program megfelelően működött.

Miután meggyőzöttem arról, hogy tényleg működik, úgy, ahogy én azt megterveztem, megkértem egy barátomat, hogy tesztelje le ő is. Az ő tesztelésének köszönhetően, pár probléma, ami eddig elkerülte a figyelmemet, napvilágot látott. Például, mi van akkor, ha a felhasználó nem megfelelő adatot ad meg. Nos ekkor az történt, hogy a program összeomlott. Ezt kiküszöbölve ismét megfelelően működött.