Programozói dokumentáció az Autószerviz programhoz

Tóth Kornél, Programozás alapjai 1., 2021.11.25.

Program célja, felépítése

Egy autószerviz nyilvántartási programjának készítése a célunk, amelynek segítségével el tudunk tárolni autókat, javításokat, illetve ügyfeleket úgy, hogy azokhoz lehessen hozzáadni, törölni belőlük, keresni bennük, vagy kiírni őket.

A programban *láncolt listákkal* dolgozunk, melyekbe az adatokat szöveges dokumentumokból töltjük be. Három* darab .txt fájl fog segítségképpen rendelkezésünkre állni. Egyikben az ügyfeleket, másikban az autókat, harmadikba na javítások adatait tároljuk. Ezeket a fájlokat minden indítás elején soronként betöltjük, a benne lévő különböző adatokat beolvassuk a három láncolt listába, majd bezáráskor ezekbe mentjük a módosításainkat. Az adatok közötti összeköttetéseket függvényeken belül oldjuk meg. A láncolt listákba rendezetten szúrjuk be az elemeket. A programot konzolon keresztül kezelhetjük, egy szöveg alapú menüben választhat a használó a funkciók közül.

Azért rendezett láncolt listákat választottuk, mert sokszor változhatnak az adatok. Egy lista építése egyszerű, könnyen bővíthető, illetve törölhető belőle listaelem.

**A specifikációban mondtuk, hogy lehetséges saját névvel létrehozni fájlokat, de az átláthatóság érdekében ezt a funkciót kihagytuk a programból, így csak 3 fájlunk lesz.*

Alapadatok felépítése

Az ügyfeleket, autókat és a javításokat is összetett típusként (*struktúra*) kezeljük. Mivel láncolt listákkal dolgozik a program, a struktúrákban van egy, a következő listaelemre mutató *pointer*.

Dátum: nap, hónap, év (könnyebb dátumkezelés érdekében definiáltuk)

Ügyfelek: Azonosító (egyedi), Név, Telefonszám

Autók: Rendszám (egyedi), Típus (márka), Műszaki vizsga érvényessége (Dátum típus), Tulajdonos ügyfélazonosítója

Javítások: Javítás dátuma (Dátum típus), Javítás típusa, Javított autó rendszáma, Javítás ára, Javított autó tulajdonosa

Adatok beolvasása

Az adatok a három főfájlból érkeznek, melyekben minden sorban egy új rekord található. A fájlokat soronként olvassuk be és a sorokat lebontjuk a pontosvesszők közötti részekre és annak segítségével hozzuk létre az új listaelemet és szúrjuk be a listánkba. Így keletkezik egy láncolt lista, amelynek a módosításával, ha végeztünk, a mentése soronként, fprintf függvénnyel történik. (Fájlkezelő függvények pl. a sortOlvas, ugyfelBe, UgyfelMentes)

Adatok beszúrása

Az ügyfeleket és autókat az ügyfélazonosító szerint rendezzük növekvő sorrendbe már az adatok beszúrásakor. A javítások dátum szerint csökkenő sorrendben (első hely a legkésőbbi, utolsó hely a legkorábbi) kerülnek a láncolt listába.

A felhasználó a beszúrás függvény meghívásánál megadja az adatokat és a függvény az azonosító vagy a dátum vizsgálatával szúrja be azokat a listába.

**Az ügyfélbeszúrás működik mindenféle feltétel nélkül. Autót csak akkor szúrhatunk be, ha létezik olyan azonosító az ügyfelek közt, amit használója megadott. Csak olyan autóhoz tartozó javítás szúrható be, amelynél a megadott autó és a megadott ügyfélazonosító is a megfelelő listában van.*

Ezekre azért van szükség, mert így hozunk létre összeköttetést a különböző struktúrák között ezzel téve a nyilvántartást átláthatóbbá, tehát nem lesz olyan javítás, amelyhez nem tartozik autó vagy tulajdonos, illetve olyan autó sem lesz a listákban, amelynek a tulajdonosa nincs az ügyféllistában. Csak akkor maradhat lehet a programban olyan javítás, amelyhez nem tartozik autó vagy ügyfél, ha azokat utólag törölték.

Adatok törlése

Autókat, illetve ügyfeleket törölhetünk. Az adatok törlése a lista átláncolásával történik, ha törlendő adat alapból nincs a listában a törlés folyamata nem fut le, jelez a program. Ha ügyfelet törlünk, a fent említett feltételek alapján az ügyfél autói is törlődnek. Autó törlésénél javítás nem törlődik, egyedül itt sérül a fenti elv.

Adatok keresése

A keresőfüggvényeink az egyik legfontosabbak. Ezek a programban több helyen előfordulnak, mint segédfüggvény, paraméterként listát kapnak, a visszatérési értékük vagy a talált ügyfél/autó vagy NULL pointer. Például, ha autóra szeretnénk rendszám alapján keresni, akkor először megkeressük és ha NULL-t ad vissza a keresőfüggvény, jelezzük, ha nem akkor kiírjuk a keresett autót (összes kereső függvény hasonlóan működik a programban).

Egy kivétel van, amikor egy azonosító alapján keressük meg az összes autóját egy ügyfélnek, akkor kihasználva a lista rendezettségét, megkeressük az első passzoló elemet, és addig megyünk amíg más azonosítót nem találunk.

A dátumokat a seged_fuggvenyek modulban talalható datumHasonlit fuggveny segítségével hasonlítjuk össze, melynek a visszatérési értéke

A „lejarok” lehetőség kiírja nekünk az aznapi dátumhoz számítva 6 hónapon belül lejáró autókat. Ezt a time.h modul segítségével érjük el.

A program adattípus, illetve használat szerint van szétbontva modulokra. A struktúrákat is külön modulban deklaráljuk. Van egy autókat kezelő modul, egy ügyfeleket kezelő modul és egy javításokat kezelő modul. A fájlok beolvasásához és mentéséhez szükséges függvények külön vannak, úgy, mint ahogy a menüket megjelenítő függvények is.

Menü felépítése

A menü sok egymásba ágyazott switch() meghívva, ami addig fut amíg a 0-s billentyűt írjuk be, akkor visszatér 0-val a program. A menü függvények visszatérési értéke a felhasználó „válasza” ezekre épülnek a switch-ek. A system(„cls”) -t többször meghívjuk függvényen belül, mert esztétikusabb lesz a program.

Függvények

A függvényeink közül, azok amelyeket a főprogramban használunk, egy már megírt, több paramétereket kérő függvényre épül, annyi a különbség, hogy a felhasználótól kéri paraméterbe az adatokat, és kevesebb (alapból adott) paramétere van. Vegyük például az egyik legbonyolultabb függvényünket: inputJavitasBeszur. Ennek három paramétere van, mindegyik egy lista elejére mutató pointerre mutató pointer (kettős indirekció). A fent említett feltételek* miatt szükséges mindhárom listát megadni mivel, nem tudhatjuk, hogy a használó hogyan akarja megadni az adatokat.