**Stancz Levente**

**RI8WCW**

Programozói dokumentáció

**A feladat**

A feladat egy telefonkönyvet mimikáló programot alkotni mely képes vCard formátumban exportálni és importálni, valamint tud a \* wildcard operátor segítségével keresni a kontaktok között. A kontaktokat struktúrában tároltam, ugyanis ez jóval megkönyítí a kódolást és a strukt adatszerkezet ilyen célra lett kitalálva. A dinamikus memória kezelést egy láncolt listával oldottam meg. Tehát van egy láncolt listám melyben egy adott elem tartlalmazza az előbb említett Kontakt struktúrát, valamint a következő elemre mutató pointer-t. A feladatban érdekes volt számomra a vCard formátumának megismerése és tanulmányozása, valamint a \* wildcard operátorral való keresés is kihívást okozott.

**A program felépítése modulonként**

**Main.c azaz az összekötő**

A main nevű modulban indul el a program és tulajdonképpen ez az összekötő a két másik modul között (megjelenítés.c és adatfeldolgozás.c). A main-ben hivódnak meg a másik modulok függvényei. A main felépítése switch szerkezetek segítségével van megoldva. Van egy darab változó (“valasztas”) ezt a változót írja felül a felhasználó amikor valamilyen menüpontot megad a konzolban. A konzolban csak a megadott menüpontok sorszámai adhatóak meg, addig kérdezi újra a felhasználótól amíg nem ad meg egy megfelelő menüpontot. Ezután egy újabb esetszétválasztással dől el, hogy a program mit jelenít meg a konzolon. A mainben kerülnek létrehozásra az átmeneti változók is (például kontakt szerkesztésnél egy Kontakt struktúra), hogy ezt egyszerűen lehessen a megjelenítésért felelős modul és az adatok feldolgozásával kapcsolatos modul között küldeni.

Minden “almenü” egy külön függvény, az átláthatóság miatt, így nem a “main” nevű függvény nagyon hosszú, hanem sok kisebb függvényből áll össze, melyek saját maguk feladatának elvégzéséért felelősek.

**Megjelenítés.c azaz a megjelenítésért felelős modul**

A megjelenítés.c fájlban található függvények mind csak megjelenítésért fontosak. Célszerű volt ezeket a függvényeket külön modulba összegyűjteni ugyanis így sokkal átláthatóbb a kód. A függvények többsége első szemléletre is elmondja, hogy mit csinál.

**Fontosabb függvények:**

**Char HibaUzenet(char \*uzenet);**

A HibaUzenet függvény a nevéből adódoan is érthető, ez a függvény a paraméterként kapott üzenetet jeleníti meg a felhasználó számára, persze ez egy hibaüzenet.

**Char UjKontaktMenu(Kontakt \*\*ujkontakt);**

Az UjKontaktMenu felelős a mainben dinamikusan lefoglalt kontakt (strukt) adatainak megadásáért. Sorban megkérdezi a felhasználót, hogy melyik adatnak milyen értéket szeretne adni. A beolvasást követően megkérdezi, hogy megfelelőek-e az adatok amiket megadott a felhasználó.

**Bool CsakSzam(const char \*str);**

A CsakSzam függvény egy segédfüggvény mely ellenőrzi, hogy a paraméterként megadott karaktertömb csak számokból áll-e.

**Char SzerkesztesMenu(ListaElem \*\*szerkKontakt);**

Hasonlósan az UjKontaktMenu-höz ez a függvény is sorban bekérdezi az adatokat a felhasználótól. Abban különbözik, hogy itt a paraméterben megadott ListaElem (láncolt lista elem) egy már létező kontaktot tartalmaz. Az új adatok beolvasása közben megjeleníti a régieket is. A végén megkérdezi a felhasználótól, hogy megfelelőek-e az adatok és ha igen akkor felülirja a paraméterben kapott kontaktot.

**Adatfeldolgozas.c azaz az adatok kezelését végző modul**

Az adatfeldolgozás.c fájlban megtalálható minden olyan függvény amely az adatok kezelésével kapcsolatos, legyen az adat módosítás, fájlba mentés, vagy fájlból beolvasás.

**Fontosabb függvények:**

**ListaElem \*FajlbolBeolvas();**

Ez a függvény beolvassa a program adatbázisából (“db.txt”) az adatokat. Ezt úgy teszi, hogy az adatok beolvasása közben a láncolt listát is létrehozza és annak első elemét adja vissza. Amennyiben nem sikerül megnyitni a fájlt vagy üres a fájl az első elem NULL értékű és ezt adja vissza. Sikeres beolvasás esetén a Megfordít() rekurzív függvény segítségével az elemeket a listában megfordítja, ez azért fontos, hogy a kontaktok id-je mindig megfelelő sorrendben legyen, valamint egy rendezett listát kapjunk amiben sokkal gyorsabban tudunk keresni, mint egy rendezetlenben.

**Void ListaFelszabadit(ListaElem \*eleje);**

Ez a függvény a nevéhez hűen, azt csinálja, hogy végigmegy a listán melyet paraméterben kap és felszabadítja a lefoglalt területeket (Kontakt és ListaElem).

**Int UtolsoIdKereses();**

Ez a függvény végigmegy a fájlon és a legutolsó sor id-jét adja vissza integer tipusként. Az előzőekben a listamegfordit függvény miatt mindig a legnagyobb id lesz a legutolsó helyen.

**Bool KontaktMenteseFajlba(Kontakt \*ujkontakt);**

Ez a függvény a paraméterként kapott kontaktot menti a fájlba. A fájl megnyítása ‘a’ – append mely ha a fájl még nem létezik létrehoz egyet. Ha létezik a fájl akkor csak a végére, utolsó sor után írja a kontakt adatait. A függvény sikerességét egy bool változó jelzi, ez a függvény visszatérési értéke is.

**Bool FajlUjraIras(ListaElem \*listaElso);**

Ez a függvény a teljes adatbázist felülírja a paraméterként kapott listával. Ez a függvény a törlés miatt fontos, ugyanis ha a láncolt listánk közepéről törlünk akkor a fájlban is a közepéről kéne, ami nem lehetséges, ezért az egészet felülírjuk az új (töröltelem nélküli) listával.

**ListaElem \*DB\_Kereses(ListaElem \*elso, const char mezo, const char \*ertek, bool specKer);**

A függvény paraméterei:

elso = A láncolt lista amelyben keresni szerenténk.

mezo = annak a menüpontnak a száma amely alapján keresni szerenténk.

ertek = amit szeretnénk keresni.

specKer = eldönti, hogy speciális keresést hajtunk-e vége.

A függvényben alapvető működése, hogy a megkapott listából kitöröl minden olyan elemet amely nem felel meg a keresési feltételnek (ertek) és az maradék elemeket pedig visszaadja egy láncoltlista-ként. Speciális keresés esetén a WildcardEllenorzes segédfüggvény használatával ellenőzizzük a keresési értékeket.

A WildcardEllenorzes függvény egy rekurziv függvény amely, ha több \*\*-t adunk meg azt is kezelni tudja. A csillag előfordulásakor pedig két irányba ágazik a függvény, az első eset, hogy a csillagot megtartjuk és csak a találati értéken megyünk végig, a másik pedig, hogy a csillagon tovébblépünk és a keresési érték sztringen megyünk végig. Ennek a két esetnek a vagy kapcsolata adja meg az eredményt.

**bool VCardBeolvasas(const char \*utvonal,Kontakt \*\*kontakt);**

Ez a függvény 2.1-es verziójú vCard-ok beolvasásra képes. Amennyiben valamilyen mező nem található meg a fájlban egy alapértelmezett “X” értéket kap. A függvény soronként beolvassa a fájl tartalmát és a vCard formatum specifikációk alapján kulcs:érték párosokra bontja azokat. Ezeknek a kulcs:érték párosok vizsgálásával dönthetőek el, mely adatok vannak megadva a fájlban.

**bool KontaktExportalas(const ListaElem \*expKontakt, const char \*utvonal);**

Ez a függvény a 2.1-es vCard formátumnak megfelelően exportálja a megadott elérési útvonalba a megadott kontaktot.

**global.h azaz a globális változókért felelős modul**

A global.h modulban található meg a struktura (LáncoltLista és Kontakt) amely az egész programban használva van, valamint egy felsorolással létre vannak hozva az adott mezők maximális méreteinek váltózojai is.

Strukturák, enum

Kontakt

Két strukturát használok a programban, az első az a Kontakt nevű struktúra, ennek felépítése:

Int id; //egyedi megkülönböztető

Char nev[MAX\_NEV];//kontakt neve

Char telszam[MAX\_TSZAM];//kontakt telefonszáma

Char email[MAX\_EMAIL];//kontakt email címe

Char nem;//kontakt neme

Char sznap[MAX\_SZNAP];//kontakt születési dátuma

Char cim[MAX\_CIM];//kontakt címe

ListaElem

Ez a struktúra felel a láncolt listáért. Felépítése:

Kontakt \*kontakt; //mutató a kontakt struktúrára

Struct ListaElem \*kov; //mutató a következő Listaelemre

Enum

Ebben tárolom a mezők maximális méretét. Felépítése:

MAX\_NEV = 71,

MAX\_TSZAM = 16,

MAX\_EMAIL = 321,

MAX\_SZNAP = 11,

MAX\_CIM = 66,

Megjegyzés:

-vCard exportálás esetén némely telefon nem a szabványos 2.1-es vCard formátum szerint exportálja az adatokat, ezeket a program nem tudja kezelni.