Telefonkönyv **– Programozói dokumentáció**

Ez a **C** nyelven megvalósított program gyakorlatilag felfogható egy kezdetleges adatbáziskezelő programként is. Az adatokat kizárólag szöveges formátumban kezeli, egy erre a célra kialakított struktúrában. Ennek a felépítése ahogy következő ábrákon is látszik, megegyezik egy megszokott relációs adatbáziséval:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Vezeteknev	Keresztnev	Mobil szam	Otthoni szam	Munkahelyi szam	Lakcim	Lakcim_varos	Lakcim_irsz	Mhely_cim	Mhely_varos	Mhely_irsz	Ceg	Osztaly	Beosztas	Email	Szuletesnap	
2	Kovacs	Janos	067012345678	nincs	1231231	Tulipan ut 11	Pomaz	111	nincs	nincs	nincs	nincs	nincs	nincs	nincs	19780424	
3	Elekes	Tamas	06309871625	061341341	nincs	Rozsa krt 5	Vac	2123	nincs	nincs	nincs	nincs	nincs	nincs	nincs	nincs	
4	Neves	Karoly	062145367126	nincs	1341134	Narcisz utca 1	Szigliget	3124	Mezo utca 89	Pecs	1234	Azur	nincs	konyvelo	nincs	nincs	
5	Nagy	Imre	nincs	123413	nincs	Pipacs ter 123	Siofok	1234	nincs	nincs	nincs	nincs	nincs	nincs	nincs	nincs	

```
typedef struct Nev
{
    char fname[50];
    char lname[50];
}Nev;

typedef struct Cim
{
    char addr[150];
    char city[50];
    char zip[30];
}Cim;

typedef struct Rekord
{
    Nev name;
    char ctel[40];
    char htel[40];
    char wtel[40];
    Cim address;
    Cim waddr;
    char ceg[150];
    char osztaly[150];
    char beosztas[100];
    char email[80];
    char bday[9];
    struct Rekord* kov;
}Rekord;
```

Ebből kifolyólag szépen ábrázolható táblázatos formában, amit a program megírása során ki is használtam, és a szokványos **.txt** formátum helyett **.csv** formátumot használtam az adatbázisok elmentésekor és megnyitásakor. Emellett lehetőség van **.vcf** kiterjesztésű, úgynevezett „virtuális névjegykártya” fájlok kezelésére is. Ezeknek a specifikációjáról a későbbiekben bővebben írok. Fontos kiemelnem, hogy mindkét típusú fájl kezelését olyan függvények végzik, amik fix formátumot várnak el a megnyitott fájloktól. Az adatszerkezetre visszatérve, a program dinamikusan kezeli az adatok részére lefoglalt memóriát, láncolt listákat használ. Ezek a listák úgy épülnek fel, hogy az új rekordok mindig a lista elejét képviselő elem elé kerülnek. A fő lista mellett megjelennek helyenként kisebb részeredményeket tároló változók és struktúrák és egy globális bool változót is használ a program. Ez ***aktiv_ab*** néven van definiálva, és a célja, hogy a különböző adatbáziskezelő függvények minél előbb kezelhessék azt az esetet, ha adatbázis létrehozása vagy importálása előtt lennének meghívva. Az adatbázis kezelésének megvalósítása bővebben kifejtésre kerül a felhasználói felület ismertetése után.

A program hagyományos konzolos felületen kommunikál a felhasználóval. A menüszerkezetet és a kért műveletek eredményét a szabványos kimenetére írja ki. Itt közli továbbá az esetleges hibaüzeneteket. A program kezeli a fájlnyitáskor, memóriafoglaláskor, és adatbevitelkor felmerülő hibákat. A bevitt karaktereket helyességét ASCII érték alapján ellenőrzi, mivel a kezelt fájlok sajátosságaiból adódóan erre a karakterkészletre szorítottam a beviteli lehetőségeket. A fájlok megnyitását ugyan ellenőrzi a program, de a beolvasott adatok feldolgozása eléggé primitív, fix formátumos ***scanf*** és ***sscanf*** függvények nyerik ki az adatokat a beolvasott sorokból. Helytelen adatformátum esetén is lefutnak az értelmező függvények, de ilyen esetben nincs értelme a beolvasott adatoknak. Ez főleg a vCard fájlok kezelésénél lesz fontos. A program a vCard szabvány 2.1-es verziójú fájljai közül tud néhányat beolvasni. Ugyan ezek a fájlok szabványosan épülnek fel, de mégsem találtam két egyforma szerkezeti specifikációt. Ezért a Windows 10-be épített Névjegyzék alkalmazás által generált fájl szerkezetét vettem alapul a funkciók megírásánál. Az így generált fájlokat leteszteltem több platformon is, és a fájlokból való importálással mindenhol sikerrel jártam. Az alábbi ábrákon a fenti táblázatban is szereplő 2 rekord vCard verziója látható, ezeket a program generálta. Az adatbázis exportálásakor ezek egymás után kerülnek a kimeneti VCF fájlba, amit a felhasználó nevez el.

```
BEGIN:VCARD
VERSION:2.1
N:Janos;Kovacs
TEL;CELL;VOICE:067012345678
TEL;WORK;VOICE:1231231
ADR;HOME:;;Tulipan ut 11;Pomaz;;111
BDAY:19780424
END:VCARD
```

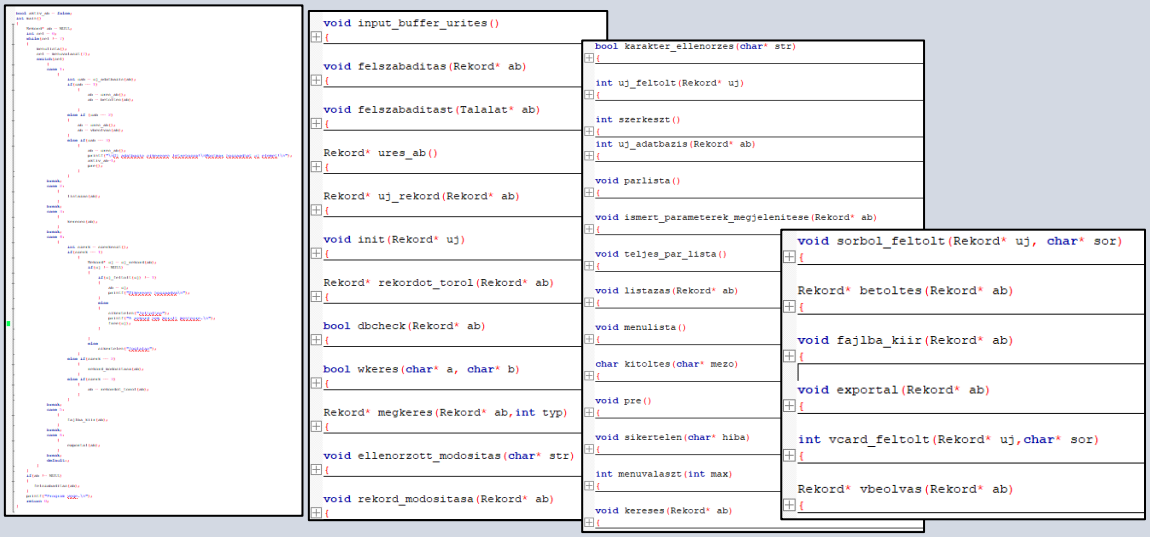
```
BEGIN:VCARD
VERSION:2.1
N:Tamas;Elekes
TEL;CELL;VOICE:06309871625
TEL;HOME;VOICE:1341341
ADR;HOME:;;Rozsa krt 5;Vac;;2123
END:VCARD
```

```
****************************************
[Fomenu: Telefonkonyv]

[1] Uj adatbazis letrehozasa
[2] Adatbazis listazasa
[3] Kereses az adatbazisban
[4] Adatbazis szerkesztese
[5] Adatbazis mentese
[6] Exportalas vCard fajlba
[7] Kilepes

Menupont kivalasztasa a sorszam begepelesevel lehetseges.

Valasztott menupont szama:
```

```
Valasztott menupont szama:
1
****************************************
[Almenu: Uj adatbazis]

[1] Beolvasas fajlbol
[2] Ures adatbazis letrehozasa
[3] Vissza a fomenube

Menupont kivalasztasa a sorszam begepelesevel lehetseges.

Valasztott menupont szama:
```

A forráskód négy darab C fájlra van felosztva, és a hozzájuk tartozó 3 header fájl is a beadott fájlok között van. A main.c fájlban egy olyan ciklus fut, ami akkor áll le, ha a menüből a kilépésnek megfelelő pontot választjuk ki. Ameddig ez nem történik meg, a ciklusmag minden egyes futása meghívja a menüt megjelenítő és a választást jóváhagyó függvény. Az utóbbinak az a célja, hogy addig kér a felhasználótól választ, amíg egy lehetséges értéket nem kap. Ha a felhasználó kiválasztott egy érvényes értéket, a program egy **switch** elágazás segítségével végrehajtja az adott menüponttal asszociált függvényeket. Ez több esetben egy azonos felépítésű almenübe vezet, ahol a válasz feldolgozása ugyanígy zajlik. Amennyiben a program külső inputot vár, ezt minden esetben jelzi a user felé. Összességében úgy gondolom sikerült egészen átlátható felületet adni a programnak. Egy adott eredmény kiírása után a program addig nem lép tovább, amíg az inputról nem kap megerősítést, de ez a funkció is kezdetleges, a bemeneti buffer tartalmától függően. Előfordulhat, hogy csak a második gombnyomásra megy tovább a program. Ennek a kiküszöbölésére először az fflush(stdin) függvényt használtam, de amikor megtudtam, hogy ennek a fordítása nincs szabványosítva, egyedi megoldásokkal kezdtem kísérletezni.

Az elkészítés során felhasználtam a „**debugmalloc**” függvénykönyvtárat, ami egy rendkívül hasznos eszköznek bizonyult. Az adatszerkezet feltöltése a fájlból olvasás mellett lehetségeses billentyűzetről is. Ezt a célt szolgálja az **uj_feltolt** függvény, ami a program leghosszabb függvénye lett. Ez annak köszönhető, hogy az adatmezőket egyesével tölti fel, és minden egyes mezőnél megkérdezi a felhasználótól, hogy szeretné-e kitölteni azt. Célszerű lett volna valamilyen rekurzivitást, függvényesítést alkalmaznom, de valamiért úgy éreztem, úgy kell hagynom ahogy a feladat elején elkészült. A hasonló, teljes rekordokat kezelő függvényeket később jóval kompaktabb formában készítettem el. Igyekeztem minél több ismétlődő kódrészletet függvénnyé komponálni, de lenne még sok helyen ilyenre lehetőség. Ahhoz képest amennyi időt a program megírásába fektettem, messze nem vagyok elégedett a végeredménnyel. Az ígért funkciók ugyan működnek, de mégis vannak olyan hibák/tökéletlenségek, amik miatt még nem szívesen adom le értékelésre a programot. A továbbiakban az egyes funkciók és az azokat megvalósító folyamat leírását szeretném ismertetni.



main.c **adatkezeles.c** **felulet.c** **fajlkezeles.c**

Ezek az képek a függvények felosztását és a forrásfájlok arányait szemléltetik.

A főprogram rövid, csupán százpár soros, fentebb kifejtettem a szerkezetét. A leadott program kb. 1100 sorból áll.

Ez a kép a specifikációból származik. Az itt megjelölt funkciókat fogom levezetni. Tanulság számomra továbbá, hogy ha a jövőben előre kell elkészíteni specifikációt, jobban át kell majd gondolnom, hogy mit és milyen pontossággal írok majd bele.

[Figyelem!]

A következő leírás strukturális felépítésű, feltételezi, hogy az Új adatbázis bekezdéstől elkezdve, folyamatosan kerül elolvasásra, így egy függvény többnyire csak egy helyen van kifejtve.

Új adatbázis

A **menuvalaszt()** függvény 1-es értéket ad eredményül, ezért a program meghívja az uj_**adatbazis()** nevű függvényt. Ez ellenőrzi, hogy van-e **aktiv_ab**, és van-e benne bármilyen adat. Ha van, akkor megkérdezi a felhasználót, hogy szeretné-e menteni az adott adatbázist. Ha 1-esre értékelődik ki a válasz akkor meghívja a **fajlba_kiir()** függvényt. Felszabadítja a lefoglalt memóriaterületet és 0-ra állítja az adatbázis státuszt.
Ezután megjeleníti az Új adatbázis almenüt, amit szintén a **menuvalaszt()** függvény kezel. A választott menüpont számával tér vissza. A legtöbb folyamat végére be van iktatva a **pre()** nevű függvény, aminek az a célja, hogy addig ne jelenjen meg a menü, ameddig a felhasználó nem olvasta es a kimenetet.

Beolvasás CSV fájlból

Amennyiben a visszatérési érték 1 volt, meghívja az **ures_ab()** függvényt, ami az **aktiv_ab** státuszát 1-be állítja, és lefoglal egy rekordnyi helyet a memóriában, és a **kov** pointerét NULL-ra inicializálja. Ennek a rekordnak a címével tér vissza a main-be, ahol ez az ab változóban a lista címeként eltárolásra kerül. Ezután meghívja a **betoltes()** függvényt, ami bekér egy fájlnevet, hozzáfűzi a **.csv** kiterjesztésjelzőt, és megpróbálja megnyitni azt. Amennyiben nem jár sikerrel, erről a felhasználót tájékoztatja, majd alaphelyzetbe állítja az **aktiv_ab** státuszt. Ha viszont megtalálta a fájlt, elkezd belőle sorokat olvasni az **fgets()** függvény segítségével. Mivel egy sorban egy rekordot tárolunk, annyiszor fut le a ciklus ahány sort az **fgets** sikeresen beolvas. Minden egyes futásnál memóriát foglal, és az előző rekord elé fűzi az elemet. A **sorbol_feltolt()** függvény paraméterként megkapja az új elem címét és a beolvasott sort. Ez egy formázott **sscanf()** függvényt tartalmaz, ami a megfelelő helyre másolja a beolvasott stringeket. Ha az utolsó sort is feldolgozta, visszaadja az utolsó elem címét, így az lesz a lista eleje.

Beolvasás VCF fájlból

Ha 2-vel tér vissza a függvény, az **ures_ab()** függvény ugyanúgy meghívásra kerül, utána viszont a **vbeolvas()** nevű funkció aktiválódik, ami az előzőhöz hasonlóan fájlnevet kér, ha tudja megnyitja, és ellenőrzi a fájl első két sorát. Ha ezek eltérnek az általa várt szövegtól, hibaüzenettel szakítja meg a betöltést, és visszatér a főmenübe. Egyébként az előzőhöz hasonlóan, soronként olvassa a fájlt, de nem foglal minden sorhoz memóriát, mivel itt egy sorban csak egy mező adata van. Ennek a vezérlését a sorértelmező **vcard_feltolt()** függvény értéke végzi. Ez 0-val tér vissza ha rekord vége, azaz „**END:VCARD**” sort olvasott be. Ha nem ezt a sort kapta, akkor ellenőrzi a sor első néhány karakterét, és ha talál olyan kifejezést amire van feladata specifikálva, akkor elvégzi azt, tehát bemásolja a formázott stringet a megfelelő mezőbe,és 1-el tér vissza. Ha a fájl végére ért a függvény, az előzőhöz hasonlóan visszadja az utoljára felfűzött rekord címét.

Üres adatbázis létrehozása

Ha 3 a visszatérési érték, akkor is az **ures_ab()** fut le, viszont nem kerül semmilyen adat az új rekordba.

Listázás

Ha a Főmenü 2. pontját választjuk, akkor ezzel meghívjuk a **listazas()** függvényt. Ez a függvény roppant egyszerű: először a **dbcheck()** funkciót hívja segítségül. Ez eldönti hogy van-e egyáltalán bármilyen adat amivel dolgozni tudna. Ezt a segédfüggvényt az összes következő menüpont is használni fogja. Ha nem talált adatokat, akkor ezt jelzi, és visszatér a főmenübe. Ha van mit kiírnia, akkor addig lépked a listában mindig kiírva az aktuális rekordban tárolt nevet, ameddig a végére nem ér.

Keresés

A Keresés menüponthoz a 3-as vezérlőszám tartozik, és a **kereses()** függvény. Ez a **dbcheck()**-el ellenőrzi az **aktiv_ab** státuszt, és ha ez nem talál hibát, akkor a parlista() megjeleníti a keresőparaméterek almenüjét, amit szokásos módon a **menuvalaszt()** értékel ki. Ebben az esetben nem találunk elágazást utána, hanem a **megkeres()** nevű függvény kapja meg a választás értékét. Ez alapján dönti el, hogy melyik mezővel kell majd összehasonlítania a kapott stringet. Egy db nevű egész változóban fogja számolni a találatok számát. Itt találkozunk először az **input_buffer_urites()** nevű segédfüggvénnyel is, ami azt hivatott meg valósítani, hogy a beolvasott sorok ne tartalmazzanak előző műveletekből hátramaradt szemetet. Itt használom a **Talalat** nevű struktúrát is, ami egy ideiglenes láncolt lista, ami a **wkeres()** függvény által megfelelőnek ítélt Rekordok címét, és egy hozzájuk rendelt sorszámot tartalmaz. A fenti menüből átvett **typ** számnak megfelelő mező értékét az **nb** karaktertömbbe másolja. Ez lesz az az érték, amivel a **wkeres()** összehasonlítja a soron következő mezőket. Ez a függvény képes egyszerre pontos egyezést kimutatni és 1 csillagos wildcard kifejezést is matchelni az átadott stringgel. Ezt úgy valósítottam meg, hogy megnézi hogy milyen hosszúak az adott stringek, megnézi, hogy talál-e benne valahol *-ot. Ha talál, akkor elmenti a helyét (**ch**), ha nem akkor **strcmp()** függvénnyel hasonlítja össze a bemeneteket. Ha talált csillagot, 3 irányba mehet tovább: a csillag a kezdő vagy végkarakter, esetleg valahol középen található. Ennek megfelelően az ismert karaktereket a szó elején valamint végén összehasonlítja, és ha nem talál eltérést, megegyezőnek ítéli a két stringet. A listát bejáró **megkeres()** függvény ekkor hozzáfűzi a **Talalat** listahoz. Ha elfogytak a rekordok, a db szám értéke alapján kilistázza a sorszámozott találatokat. A **menuvalaszt()** segítségével végül visszaadja annak a Rekordnak a címét amelyiket választottuk, és felszabadítja az ideiglenes listát a **felszabaditast()** függvény segítségével, ami speciálisan ehhez az adattípushoz készült.

Szerkesztés

A 4-es almenü a **szerkeszt()** függvény által kerül kirajzolásra. A **menuvalaszt()** függvény értéke alapján választ utat a program.

Új bejegyzés

Ha az 1-es gombot nyomtuk meg, az **uj_rekord()** függvényt hívjuk. Ez lefoglal egy rekordnyi helyet, a lista elejére fűzi, és visszaadja a lista új címét. A program addig nem írja át a lista címét, amíg nem győződött meg róla, hogy sikeres volt a memóriafoglalás. Ha nem ez lenne a helyzet, akkor hibát jelez és visszatér a főmenübe. Viszont ha sikerült, akkor meghívja rá az **uj_feltolt()** függvényt. Ez először meghívja az új rekordra az **init()** függvényt, ami annak minden mezőjének „nincs" érétket ad. Ezután kér egy nevet a user-től, és ellenőrzi a karakterek helyességét. Erre szolgál a **karakter_ellenorzes()** függvény, ami eldönti hogy a kapott string csak érvényes **ASCII** karaktereket tartalmaz-e. Amennyiben érvénytelen karaktert talál, törli a rekordot, és visszatér a főmenübe. Ez a szigorú szabály csak a név mezőre igaz, mivel ez vCard-ban kötelező mező. Ezután minden egyes mezőre egyesével meghívja a **kitoltes()** függvényt, ami a felhasználó döntését regisztrálja arra vonatkozóan, hogy szeretné-e kitölteni az adott mezőt. És ennek megfelelően beolvas, továbblép, vagy ha éppen nem szeretnénk több mezőt kitölteni, akkor sikeres értékkel tér vissza, így a lista címe ténylegesen az új elem címe lesz.

Módosítás

2-es alpont, meghívja a **rekord_modositasa()** funkciót, a **dbcheck()-**el ellenőrzi az adatbázis létezését, majd a megkeres() függvény segítségével kiválasztja a kívánt rekordot, ha talál megfelelőt. A **teljes_par_lista()** megjeleníti az összes elérhető paramétert, amik közül a **menuvalaszt()** függvény segítségével választhatunk egyet, vagy visszaléphetünk. Az eredmény egy **switch**-el választja ki a megfelelő mezőbe való írást. Az **ellonorzott_modositas()** függvény a **karakter_ellenorzes()** függvénnyel ellenőrzi a bemenet helyességét. A függvény üres sor bevitele esetén „**nincs**"-re állítja a mező értékét. Ha helyes a bemenet, akkor bemásolja a megfelelő helyre, egyébként meghagyja az eredeti értéket. A paraméterválasztó menü addig újra meg fog jelenni, ameddig nem választottuk ki azt, hogy nem kívánunk további változtatásokat végezni.

Törlés

A 3. alpont kiválasztása a **rekordot_torol()** függvényt használja. A **kereses()-**hez hasonlóan kezdődik, **dbcheck(),** aztán **megekeres().** Ellenőrzi, hogy sikeres volt-e a kiválasztás, és megkérdezi, hogy biztosan szeretnénk-e törölni a rekordot. A **menuvalaszt()** kiértékeli a döntést, és visszatér a főmenübe ha kell. Egyébként ellenőrzi hogy az első elemet kell-e törölni, ha igen, akkor a kov paraméterét adja vissza, a rekordot pedig felszabadítja. Ha nem az elsőt kell törölni, megekeresi azt a rekordot aminek a **kov** paramétere a törlendő rekordra mutat, ezt a pointert pedig átállítja a törlendő rekord **kov** pointerére, majd törli a rekordot.

Mentés

Az 5-ös menüpont meghívja a már említett **fajlba_kiir()** függvényt. Ez a **dbcheck()** funkcióval ellenőrzi az adatbázis státuszát. Amennyiben van értelme a mentésnek, a függvény fájlnevet kér be. A **.csv** kiterjesztést itt is szövegkezelő függvény fűzi a címhez. A fájlnyitás ellenőrzését a szokásos módon kezeli. A megnyitott fájlba először egy fejlécsort ír, hogy táblázatkezelővel megnyitva könnyű legyen eligazodni a mezők között. Ezt beolvasáskor átugorja. A többi sorba az első rekord mezőit írja, vesszővel elválasztva. Az utolsó mező után új sorba lép, és megy a következő rekordra amíg a végére nem ér. A fájl bezárása után státuszüzenetet közöl.

Exportálás

A 6-os menüpont alatt a **VCF**, a másik fájlkimenetet találjuk. Ennek a függvényét **exportal()-**nak neveztem el. Az indulás szinte megegyezik a CSV fájlokat író függvénnyel. Itt sem maradhat el a **dbcheck()** ellenőrzés, és a fájlnevet is azonos módon kéri be. A fájlnyitás ellenőrzése után az első rekordtól kezdve elkezdi soronként kiírni a mezőket, de előtte **strcmp()** függvénnyel ellenőrzi, hogy kaptak-e értéket az adott mezők, mivel ha nem, akkor nem kell őket kiírni. A listabejárás a megszokott módon történik. A sikerességről itt is tájékoztatást kapunk. A **sikertelen()** függvény jelenleg csak egy egyszerű függvény, ami a paraméterül kapott stringet írja ki.