Programozói dokumentáció

A projekt felépítése

A program több különálló modulra van bontva, melyek össze dolgoznak. A szükséges header fájlok és könyvtárak importálása minden esetben a megfelelő működés figyelembevételével történt, ezt a későbbiekben nem részletezem. Így pl. minden .c fájlban megtalálható a debugmalloc.h hívása, mellyel ellenőrizhető az esetleges memóriaszivárgás. A következő fájlok találhatók a projektben:

- main.c
- bejegyzes.h
- debugmalloc.h
- kontaktok.nevjegy
- lancoltlista.c és lancoltlista.h
- fajlkezeles.c és fajlkezeles.h
- kereses.c és kereses.h
- menu.c és menu.h

Mint az látható, minden 'valami.c'-hez tartozik egy azonos nevű header fájl is, hiszen a függvény deklarációk a fejléc fájlban történnek, a tanultak szerint. Ezekre később csak a nevükkel hivatkozok, de mindkettő tartalmáról nyilatkozok.

Egyes fájlok tartalma

main

Az ékezeteket megfelelően kezelő karakterkódolás beállításához a windows-1250 karakterkészletet használtam. A megfelelő függvényhívások megtalálhatók a main()-en belül, ezekről részletesebben később írok.

bejegyzes.h

A bejegyzés fejléc fájl egy struktúrát tartalmaz, melynek tartalmára a későbbiekben a program összes többi eleme épít. A struktúra tartalma a következő:

```
typedef struct bejegyzes{
    char nev[50];
    char telefonszam[50];
    char lakcim[50];
    char egyeb[50];
}bejegyzes;
```

Minden adat karaktertömbként tárolódik, véleményem szerint 49 (+1 a lezáró nullának) karakternek elégnek kell lennie még a cím megadásához is, leszámítva néhány speciális esetet, mellyel külön nem foglalkozom. A változó nevek magukért beszélnek.

debugmalloc.h

Egy hasznos függvénykönyvtár, melynek használatát nem csak engedélyezték, hanem erősen ajánlották a nagy házi memóriaszivárgásainak teszteléséhez.

kontaktok.nevjegy

Ebben a fájlban tárolódnak a felhasználó által létrehozott/módosított kontaktok adatai, egy névjegy 4 egymást követő sorban tárolódik. Ezeket a program összetartozó egységként kezeli.

lancoltlista

A lácolt listába történő beszúráshoz, elemek törléséhez szükséges függvényeket tartalmaz. Ezen kívül itt található a feladat további részében felhasznált listaelem típusú struktúra deklarációja is.

A struktúra magyarázata:

```
typedef struct listaelem{
    bejegyzes elem;
    struct listaelem *kovetkezo;
} listaelem;
```

Egy bejegyzés struktúra típusú elem és egy ’következő’ pointer található benne. Ezeknek a későbbiekben fontos szerepe lesz, a fájlon végzett műveleteknél.

A beszúró függvény magyarázata:

```
listaelem *hatra_beszur(listaelem* meglevoListaElsoEleme, bejegyzes ujelem)
```

2 paramétert vár, a meglévő lista első elemének mutatóját, illetve egy bejegyzés példányt – ami tartalmazza a 4 bejegyzés típusú adatot. A fajlkezeles.c-ben lesz használva.

Törlő függvények magyarázata:

```
listaelem *mindent_torol(listaelem *eleje)
listaelem *torol(listaelem *megLevoListaElsoEleme, listaelem *toroltelem)
```

A mindent töröl függvény a main.c-ben található main() függvényben kerül elő, paraméterként a keresett névjegy első bejegyzését kapja, onnét tudja honnan kell törölnie a megadott kontaktot. A második törlő függvény a meglévő lista első elemének mutatóját és egy listaelem típusú törölt elem paramétert vár. Itt a név csalóka, hiszen az elem még nincs kitörölve, az az elem kerül oda, amit éppen törölni fogunk.

fajlkezeles

A ’kontaktok.nevjegy’ fájl írásához és olvasásához szükséges függvénypárost tartalmazza, az stdio.h-ban levő FILE* pointer használatával. A használt függvények:

```
listaelem* beolvasas()
void kiiras(listaelem *lista)
```

A fájl beolvasását végző függvény nem vár paramétert, csak teszi a dolgát. Az fopen() és fclose() függvények segítségével történik a fájlok megnyitása, majd lezárása. Az esetleges hibák jelzését a perror() végzi. A kiíráshoz használt függvény egy listaelem típusú pointert vár.

kereses

A keresést egy felsorolt típussal (enum) végzem, melyhez értékként 1-től 4-ig rendelek számokat, ezzel gyorsítva a programot és érthetőbbé téve a kódot (laborvezetőm javaslatára). 2 függvény szerepel a fájlban, melyek a következők:

```
bool hasonlo(char *str, char* minta);
listaelem *keres(listaelem *eleje, char* k, enum keresesOpciok opcio);
```

Az első függvény 2 karakterláncot hasonlít össze, majd igaz vagy hamis értéket ad vissza. Ezt a közvetlenül utána levő függvényben fel is használom, egy adott névjegy kereséséhez. A keresésért felelős függvény már komolyabb, 3 paramétert vár. Ismernie kell, hogy a listán belül hol van az az elem, amit éppen keresünk, utána összehasonlítja a felhasználó által megadott értéket a lista elemeivel, ha egyezés van, akkor a talált elemet adja vissza, egyébként semmit. A kereső opciók a header fájlban definiáltak szerint vár egy számot, amit a menu.c-ben egyébként felhasználóbarát módon ki is ír a program.

menu

Ez egy alaposan átgondolt része a programnak, hosszas tervezés áll mögötte. Egy switch()-case szerkezetre épül fel, a program indulása után a főmenü fogad minket, ahonnan almenükbe, majd onnét további almenükbe navigálhatunk. Nem kívánom részletezni, ugyanis egyrészt leírtam a felhasználói dokumentációban, másrészt a használata nagyon egyszerű és triviális. Esetleges hibák jelzését is remekül végzi.

```
listaelem *fomenu(listaelem* lista);
listaelem *ujnevjegy(listaelem* lista);
listaelem *szerkesztes(listaelem* lista);
listaelem *torles(listaelem* lista);
void kereses(listaelem* lista);
```