Határidőnapló-Programozói dokumentáció

A cél egy olyan program készítése, amely dinamikus memóriakezelést használva képes határidőnaplóba rekordokat tárolni, módosítani, létrehozni, keresni, fájlból beolvasni, fájlba írni. A CodeBlocks fejlesztői környezetben készült a program. A GNU GCC Compiler-rel fordul le a program.

Adatszerkezetek és fejlécek

A program alapvetően rekordokat tárol, illetve azokkal dolgozik. Egy rekord a dátumból, időből, helyből, esemény nevéből, illetve az eseményhez tartozó megjegyzésből áll. Ahhoz, hogy a későbbi kilistaz() függvény működőképes legyen, a dátumot fel kell bontani külön hónapra és napra, az időt órára és percre. Ezek a rekordok egy egyszerű lácolt listában tárolhatóak el. Egy db rekord összetartozó elemeit struktúrában tárolhatjuk:

```
typedef struct Rekordok{
   int honap, nap, ora, perc;
   char hely[200];
   char esemeny[200];
   char megjegyzes[200];
   struct Rekordok *kov;
}Rekordok;
```

Itt a honap, nap, ora, perc egy-egy integer, a hely, esemeny és a megjegyzes pedig egy-egy meghatározott méretű karaktertömb. A *kov mutat a láncolt lista következő elemére.
A struktúrát a **letrehoz_keres_felszabadit.h** fejlécben inicializáltam. Ebben a fejlécben deklaráltam még a *létrehoz*, a *keres*, a *felszabadit*, a *hanyadiknap* és a *hetelsonap* függvényeket is:

*Rekordok *letrehoz(Rekordok *naplo, int honap, int nap, int ora, int perc, char *hely, char *esemeny, char *megjegyzes);*
*void keres(Rekordok *naplo);*
*void felszabadit(Rekordok *naplo);*
int hanyadiknap(int honap, int nap);
int hetelsonap(int het);
A modul forrásfájlja a **letrehoz_keres_felszabadit.c.**

A **modosit.h** fejlécben deklaráltam a modosit függvényt:
*Rekordok *modosit(Rekordok *naplo);*
A modul forrásfájlja a **modosit.c**.

A **kilistaz.h** fejlécben deklaráltam a *kilistaz* függvényt:
*void kilistaz(Rekordok *naplo);*
A modul forrásfájlja a **kilistaz.c**.

Az **econio.h** fejlécre szükségem volt az *econio_clrscr()* függvény használata miatt, ami az **econio.c** forrásfájlban található meg.

Függvények

int main(void)
A **main.c** –ben található.
A main() – függvényben létrehoztam egy naplo nevű üres listát, majd definiáltam egy 200 elemű karaktertömböt. Printf segítségével kiírattam egy kérdést, amely a felhasználótól megkérdezte, hogy kívánja-e beolvastatni a proba.txt nevű próbafájlt a programba. A válaszát scanf segítségével kértem be és töltöttem bele a korábban definiált karaktertömbbe. Ezután if függvényt használtam. A feltétel az, hogy a felhasználó által korábban megadott válasz „igen” legyen. Ezt egy sztringkezelő függvénnyel, az strcmp-vel hajtottam végre. Ha a feltétel teljesül, akkor a program megpróbálja fopen segítségével beolvasni, valamint a FILE *fp pointernek átadni a próbafájlt. If függvény segítségével ellenőrizzük, hogy a fájlt sikerült-e beolvasni vagy sem. Ha nem sikerült (tehát *fp==NULL, ez itt az if igazága), akkor a program kiírja, hogy nem sikerült megnyitni és továbblép a *main()*-ben. Ha sikerült átadni a fájlt az *fp pointernek, akkor először definiálunk egy 1000 elemű karaktertömböt (sor változót), amely majd egy sor adatait fogja eltárolni. Ezután *while()*-ciklus segítségével a naplo-t, mint láncolt listát feltöltjük a próbafájl rekordjaival. A *while()* feltételének azt adjuk meg, hogy *fgets()*-szel beolvasva a próbafájlból az egyes sorokat nincs vége a fájlnak. Ezután *sscanf* függvény segítségével beolvassuk az egyes sorok rekordjait a megfelelő formátumból, majd a *letrehoz()* függvény segítségével létre is hozzuk a fájl rekordjaiból álló láncolt listát. *Fclose* segítségével lezárjuk a megnyitott próbafájlt.
A *main()* függvényben van a főmenü menükezelése is, amit do-while függvény segítségével valósítottam meg. Printf segítségével kiírattam a főmenü 5 pontját :

"1. Uj rekord letrehozasa\n"
"2. Rekord (esemeny) modositasa\n"
"3. Esemenyek kilistazasa\n"
"4. Esemeny keresese\n"
"5. Kilepes es mentes”

A felhasználótól scanf segítségével bekértem az általa választott menüpont számát, majd switch-case szerkezettel megadtam mind az 5 esetet. Az 1-es esetben (tehát az 1-es menüpont választása esetén) először az *econio_clrscr()* függvény segítségével „megtisztítjuk a képernyőt”, majd *scanf* segítségével bekérjük a struktúrához tartozó adatokat:

```
   printf("Datum:")
while(scanf("%d.%d.",&honap, &nap)!=2){
   printf("Nem ho.nap. formaban adta meg.\n");
   scanf("%*[^\n]%*c");
   printf("Datum: \n");
}
printf("Ido:");

while(scanf("%d:%d", &ora, &perc)!=2){
   printf("Nem ora.perc. formaban adta meg.\n");
   scanf("%*[^\n]%*c");
   printf("Ido: \n");
}
printf("Hely:");
scanf("%s", hely);
printf("Esemeny:");
scanf("%s", esemeny);
printf("Megjegyzes:");
scanf("%s", megjegyzes);
```

While-ciklus segítségével az integer beolvasásánál a program „hibakezelést” végez, tehát ha a felhasználó nem megfelelő formátumban adta meg az adatokat, akkor a program nem áll le, hanem továbbfut, és szól a felhasználónak, hogy nem jó formában adta meg az adatot. Ezt a hibakezelést a program további, integert beolvasó részében is használtam.
A beolvasás után meghívom a *letrehoz()* függvényt a korábban bekért adatok segítségével, és a letrehoz() visszatérési értéke megváltoztatja a naplo változót. (A láncolt lista elejére a *letrehoz()* függvény segítségével beszúrunk egy elemet.) Ismét „megtisztítjuk a képernyőt”.

A 2-es esetben a *modosit()* függvény visszatérési értékével változtatom meg a naplo változót. A 3-as esetben meghívom a *kilistaz()*, a 4-es esetben a keres(), az 5-ös esetben a felszabadit() függvényt. Minden *case* végén *break* segítségével kilép a program a *switch-case*-ből és „kezdődik elölről” a *do-while*. Hiba (*default*) esetén is kilép a program a *switch-case*-ből és értesíti a felhasználót, hogy nem megfelelő számot adott meg. A *do-while* addig fut, amíg a program az 5-ös esetbe nem lép.

*Rekordok *letrehoz(Rekordok *naplo, int honap, int nap, int ora, int perc, char *hely, char *esemeny, char *megjegyzes)*

A **letrehoz_keres_felszabadit.c** –ben található. Ezzel a függvénnyel tudjuk létrehozni a láncolt lista egy-egy elemét, tehát egy-egy rekordot. Paraméterei: láncolt lista egy eleme, valamint a Rekordok struktúrában is szereplő adatok. A függvény úgy működik, hogy a láncolt lista elejére szúrja be az új elemet, majd a visszatérési értéke is az új elem címe lesz. A függvénytörzsben dinamikusan foglalunk le területet az új elemnek, majd az új elem integerre mutató részeit feltöltjük a függvény megfelelő paramétereivel, az új elem karaktertömbjeire mutató részeibe pedig *strncpy()* segítségével bemásoljuk a függvények paraméterei közül a megfelelőket. Végül az uj elem „következő” pointerét beállítjuk a naplo értékére.

*void keres(Rekordok *naplo)*

A **letrehoz_keres_felszabadit.c** –ben található. Ezzel a függvénnyel meg tudunk keresni bármilyen eseményt, amely a láncolt listában szerepel. A függvény visszatérési értéke void, hiszen nincs szükségünk visszatérési értékre, csak egy printf segítségével kiíratjuk a megtalált eseményt és a hozzá tartozó adatokat. A függvénynek átadott paraméter a láncolt lista egy elemére mutat. A függvénytörzsben létrehozunk egy 200 elemű karaktertömböt, amelybe *scanf* segítségével beolvassuk a keresett eseményt. Definiálunk egy Rekordok *mozgo elemet, amelyet beállítunk a naplo értékére. *While*-ciklus segítségével végigmegyünk a listán. A *while* feltétele az, hogy *mozgo nem nullpointer, illetve, hogy a mozgo eseményre mutató értéke nem egyenlő a felhasználó által megadott eseménnyel. Ez utóbbit strcmp segítségével vizsgáljuk. A cikluson belül a mozgo felveszi a mozgo következő pointere elemének értékét. Ha nem találta meg az eseményt a program (tehát kilépett a *while*-ból, mert a mozgo nullpointer), akkor azt kiírja és leáll. Ha pedig megtalálta, akkor kiíratja a megfelelő rekordot.

*void felszabadit(Rekordok *naplo)*

A **letrehoz_keres_felszabadit.c** –ben található. Ennek a függvénynek az a feladata, hogy fájlba elmentse a láncolt lista elemeit, és ezután fel is szabadítsa a listát. Visszatérési értéke nincs a függvénynek, paraméterként a láncolt lista egy elemét veszi át. Definiálunk egy 200 elemből álló karaktertömböt, majd *scanf* segítségével bekérünk a felhasználótól egy fájlnevet, amelybe fogja a függvény elmenteni a láncolt lista elemeit. Ha nem sikerül valami miatt a

fájlt megnyitni, akkor ezt jelzi a program a felhasználónak egy üzenettel. Ha sikerült, akkor definiáljuk a lista egy mozgo elemét, majd *for* ciklus segítségével végigmegyünk a láncolt listán a mozgo=mozgo->kov lépésekkel, és *fprintf*-fel beleírjuk a fájlba az egyes rekordokhoz (lista eleméhez) tartozó megfelelő, tabulátorral elválasztott adatokat. Ezután bezárjuk a fájlt, ami így elmentődött. Most következik a lista felszabadítása, amihez először definiáljuk a lista egy elemét (Rekordok *iter), majd *while* ciklus segítségével végigmegyünk a listán és felszabadítjuk az összes elemét.

int hanyadiknap(int honap, int nap)

A **letrehoz_keres_felszabadit.c**-ben található. A függvény paraméterei két integer, a hónap és a nap, amelyek alapján a meghatározza, hogy az adott hónap adott napja az év hanyadik napja.

int hetelsonap(int het);

A **letrehoz_keres_felszabadit.c**-ben található. A függvény paramétere egy integer: a hét. A függvény meghatározza, hogy a hét első napja az hanyadik napja az évnek.

*Rekordok *modosit(Rekordok *naplo)*

A **modosit.c**-ben található. A *modosit()* függvény összetettebb, mint a többi, hiszen ezen belül is van menükezelés, illetve a láncolt listának a felhasználó által megadott rekordjának bármely részét képes módosítani, illetve akár egy teljes rekordot is törölni abból. Először is definiálunk módosított hónap, nap, óra, perc integereket, 200 elemű módosított hely, esemény neve, megjegyzés karaktertömböket, egy integer n-t, ami majd a választott menü sorszámát jelzi, illetve egy 200 elemű karaktertömböt, amelybe majd a felhasználó által megadott eseményt töltjük be. Itt, ugyanúgy, mint a *main()*-ben *do-while*-t és *switch-case*-t használunk a menükezeléshez. *Printf*-fel kiíratjuk a választható menüpontokat:

"1. Datum modositasa\n"
"2. Ido modositasa\n"
"3. Hely modositasa\n"
"4. Esemeny nevenek modositasa\n"
"5. Megjegyzes modositasa\n"
"6. Rekord (esemeny) torlese\n"

Ezután *scanf*-fel bekérjük a felhasználótól a választott menüpont sorszámát, azt betöltjük az n-be, majd *if()* függvénnyel megvizsgáljuk. Ha n>6, akkor a program kiírja, hogy nincs ilyen menüpont és kilép a függvényből, vissza a *main()* függvénybe, ahol meg lett hívva. Ha az *if* igazága nem teljesült, akkor *econio_clrscr()*-ral megtisztítjuk a képernyőt, majd *scanf*-fel bekérjük annak az eseménynek a nevét, amelyet a felhasználó módosítani szeretne.
Definiáljuk a listának két elemét. Az egyik nullpointer lesz (lemarado), a másik pedig egyenlő lesz a naplo értékével(mozgo). Ezután egy *while* ciklus segítségével végigmegyünk a listán és megkeressük *strcmp* segítségével a lista azon elemét, amelynek az eseményre mutató pointere megegyezik a felhasználó által megadott eseménnyel. Ezt úgy valósítjuk meg, mint a *keres()* függvényben, csak itt a lemarado=mozgo; mozgo=mozgo->kov .
If függvény segítségével megvizsgáljuk a mozgo értékét. Ha nullpointer, akkor nincs olyan esemény, amit a felhasználó megadott, ezt jelzi is a program üzenetben, valamint kilép a függvényből *break* segítségével és visszalép a *main()*-be. Az *else* ágban lesz a *switch-case*. Az 1-es esetben a program bekéri a felhasználótól az új dátumot (hónap, nap), a 2-es esetben az új időt (óra, perc) , a 3-as esetben az új helyet, a 4-es esetben az új nevét az eseménynek, az 5-

ös esetben pedig az új megjegyzést. Az 1,2 esetekben a mozgo megfelelő adatokra mutató pointerének értéke a megfelelő bekért adat értékével lesz egyenlő. A 3,4,5 esetekben ugyanez *strncpy*-val van megoldva. A 6-os esetben a megtalált eseményhez tartozó összes többi adatot is(tehát magát a rekordot) kell kitörölni, azaz felszabadítani a láncolt listából. Ezt két esetre kell bontani. Ezt *if*-fel és *else*-zel oldjuk meg. Ha lemarad==NULL, akkor a lista elejéről kell törölni, tehát definiálni kell egy új listaelemet, amely majd az új első eleme lesz a listának, és egyenlő a mozgo következőre mutató pointerével. Free()-vel fel kell szabadítani a mozgot, majd a naplo értékét egyenlővé kell tenni az új első elem értékével. Az *else* ágban lesz a másik lehetőség, amikor a lemarado következőre mutató pointerének értéke egyenlő a mozgo következő elemére mutató pointerével. Minden eset után a program kilép a *switch-case*-ből és folytatódik a *do-while* (tehát újra megjelenik a menü). A *do-while*-ból akkor lép ki a program, ha n<7 feltétel nem teljesül. Ekkor a program visszalép a main()-be.

*void kilistaz(Rekordok *naplo)*

A **kilistaz.c**-ben található. Ez a függvény is összetettebb, ebben is van menükezelés. Visszatérési típusa void, hiszen nincs visszatérési értéke, csak *printf*-fel kiírja a kilistázott eseményeket. Paraméterként veszi át a láncolt lista egy elemét. A függvényen belül van még két apró függvény. Az első a int *hetelsonap(int het),* amely visszaadja a paraméterként megadott hét első napját az évben. A másik a int *hanyadiknap(int honap, int nap),* amely paraméterként kap két számot: a hónapot és a napot, és ezek alapján kiszámolja , hogy az az év hanyadik napja, ez a visszatérési értéke is. Definiáljuk a lista egy elemét (mozgo), majd *do-while* függvényt használunk itt is. Printf-fel kiíratjuk a menüt:

"1. Nap szerinti listazas\n"
"2. Het szerinti listazas\n"
"3. Honap szerinti listazas\n"

A felhasználótól ezután *scanf*-fel bekérjük a választott menü számát, amit már korábban definiált int n-be töltünk be. Ezután itt is *switch-case*-t használunk. Az 1-es esetben *scanf*-fel bekérjük a felhasználótól, hogy hanyadik hónap, hanyadik napja alapján kívánja kilistázni az eseményeket. Definiálunk egy bool vanilyenesemeny logikai változót, majd *for*-ciklus és a mozgo segítségével végigmegyünk a láncoltlistán. A cikluson belül van még egy *if* függvény is, amelynek igazága akkor teljesül, ha a mozgo hónapra mutató pointerének értéke és a napra mutató pointerének értéke is megegyezik a felhasználó által megadottakkal. Ekkor vanilyenesemeny=true, és printf-fel kiíratjuk a feltételnek megfelelő elemeit a listának. A *for* ciklus után is van egy if-függvény, amely akkor teljesül, ha vanilyenesemeny=false. Ilyenkor a program kiírja, hogy „Nincs ilyen esemény.” A 2-es esetben szinte ugyanaz történik, mint az 1-esben, csak itt a hetet kell megadni. Illetve a *for* cikluson belül még van egy *for* ciklus, amely kezdőértékének veszi a *hetelsonap(het)*-et, és végigmegy a hét napjain. A *for*-ciklusokon belüli *if*-ben itt az a feltétel, hogy *ahanyadiknap()* füvvgénybe belerakva a mozgonak a hónapra mutató pointere és a napra mutató pointere , a függvény visszatérési értéke megegyezzen a hét valamelyik napjával. Ha ez a feltétel teljesül, akkor a program kiírja a lista megfelelő elemeit. Ha bool vanilyenesemeny=false, akkor itt is azt írja ki, hogy „Nincs ilyen esemeny.” A 3-as eset teljesen ugyanaz, mint az 1-es, csak itt egyedül a hónapot kell megadni, illetve vizsgálni. Mindhárom eset végén kilép a *switch-case*-ből és addig folytatja a *do-while*-t, amíg n<4 (tehát a felhasználó által menüválasztásként beírt szám <4). Ilyenkor egyébként is a default lép életbe, ami során jelzi a program a felhasználónak egy üzenetben,

hogy nem létezik olyan menüpont, amit beírt. Amikor n>=4, akkor kilép a függvényből, és visszalép a *main()*-be.