Snowboardkölcsönző dokumentáció

Készítette: Sághy Dániel

1. A feladat rövid ismertetése:

A feladat címe: Snowboard kölcsönző

* **Mi a program célja?**

Egy tetszőleges snowboard kölcsönző nyilvántartásának kezelése.

* **Kiknek készült?**

Elsősorban a kölcsönző üzemeltetőjének, hogy nagyban megkönnyítsük és átláthatóbbá tegyük a kölcsönzéseket.

Ezen kívül a bérlőknek, akinek szintén hasznukra válik a gördülékenyebb kölcsönzés.

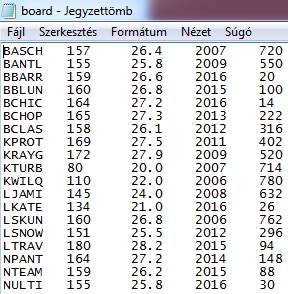
1. A program ismertetése:

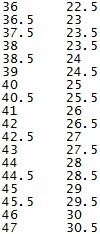
A programunk megadja, hogy egy adott emberhez milyen snowboardot tudunk ajánlani. Ehhez figyelembe veszi, a bérlő paramétereit és lehetőség szerint többféle deszkát is felajánl. Ezek után a bérlő maga választhat, a kalkulált ár és a snowboardok ismeretében.

A programunktól megkérdezhetjük, hogy egy adott szezon után melyek azok a snowboardok melyeket már le kell selejtezni, mely felszereléseket használták a legtöbbet. Szezon alatt mindig az aktuális konkrét évet értjük.

A program 3 szöveges fájlt használ, ebből kettőt a bezáráskor az aktuális frissített adatokkal újra ír, a harmadikból csak olvas.

Ezek a fájlok pedig a következők:



* + - BOARD.TXT
    - BERLOK.TXT
    - LABMERETEK.TXT

A dokumentum első oszlopa a lábméretek standard európai méretben, mellettük pedig a centiméterben mért megfelelőjük, hogy a program tudjon számolni velük.

A programunk egy parancssori menüvel indul, melyben 10 db funkció érhető el. A felhasználó ezek közül tud választani. Ezek pedig a következők:

1. Összes bérlés listázása:

Itt kiírja a program a kölcsönzőben jelenleg nyilvántartott snowboardokat. Valamint azokhoz, amik épp bérelve vannak, kiírja a bérlőiket.

1. A bérlés ára:

Ha egy új bérlő érkezik a kölcsönzőbe ezt a funkciót kell használunk. Itt felvesszük a nyilvántartásba, hogy mikor melyik deszkát szeretné bérelni.

1. Bérlés vége, a bérlő visszahozta a bérelt terméket:

A funkció nevéből adódik, mikor használjuk. Ilyenkor a bérlő kikerül a nyilvántartásból, azért hogy például egy sok évig működő kölcsönzőnek ne halmozódjon fel egy kezelhetetlen nagy adatbázisa.

1. A legtöbbet bérelt deszka:

A program kiírja a nyilvántartásban szereplő legtöbbet bérelt deszkát.

1. Szezon végén leselejtezendő deszkák:

A baleset veszély elkerülése érdekében minden szezon végén a 10 évnél idősebb deszkákat le kell selejtezni. Ezek azonosítóját kapjuk meg.

1. Snowboard hozzáadása:

Új snowboard hozzáadása a nyilvántartáshoz.

1. Snowboard törlése:

Snowboard eltávolítása a nyilvántartásból.

1. Bérlők listázása:

Jelenlegi bérlők listázása.

1. Boardok listázása:

Jelenlegi snowboardok listázása.

1. Kilépés.
2. A program felépítése:

Az korábban felsorolt funkciók működését a következőképpen oldottam meg:

1. Definiáltam 5-féle adattípust:

typedef struct datum{

int ev;

int honap;

int nap;

}datum;

typedef struct berles{

datum kezdete;

datum vege;

}berles;

typedef struct nev{

char vezeteknev[20];

char keresztnev[20];

}nev;

A vezeték és a keresztnév is maximum 19 karakter + ’\0’.

typedef struct berlo{

nev neve;

int testtomeg;

int testmagassag;

double labmeret;

char szint[15];

berles berles\_idotartama;

char azonosito[6];

struct berlo \*kovetkezo\_berlo;

struct berlo \*kov\_berles;

}berlo;

A \*kovetkezo\_berlo a láncolt listában a következő bérlőre mutató pointer. A \*kov\_berles pedig a fésűs listában egy adott snowboardot bérlők listájában mutat a következő elemre. Az azonosító minden bérlőnél, annak a snowboardnak az azonosítója, amit bérel (5 karakter + ’\0’).

typedef struct snowboard{

char azonosito[6];

int hossz;

double szelesseg;

int ev;

int napok;

struct snowboard \*kov;

struct berlo \*berlo;

}snowboard;

A \*kov pointer a snowboardok láncolt listájában a következő boardra mutató pointer. A \*berlo pedig arra a bérlőre mutat, aki az adott boardot bérli. Ha ez nem csak egy elem, akkor ez lesz ennek a listának a head pointere és benne az elemek már mutatnak a következőre.

1. Snowboardokat kezelő függvények:
   1. snowboard\* board\_beolvas();

Ez a függvény a program furásának leg elején a lényeges, amikor beolvassa az adatokat a BOARD.TXT-ből és felépíti belőlük a boardok láncolt listáját. A végén pedig vissza adja a lista első elemére mutató pointert.

|  |
| --- |
| Head board |

|  |
| --- |
| Azonosító |
| Hossz |
| Szélesség |
| Év |
| Napok |
| \*kov |
| \*berlo |

|  |
| --- |
| Azonosító |
| Hossz |
| Szélesség |
| Év |
| Napok |
| \*kov |
| \*berlo |

|  |
| --- |
| Azonosító |
| Hossz |
| Szélesség |
| Év |
| Napok |
| \*NULL |
| \* berlo |

* 1. snowboard board\_beker();

A standard inputrol kér be adatokat melyből létrehoz egy snowboard típusú elemet és ezt adja vissza.

* 1. void board\_hozzaad

(snowboard \*head\_board, snowboard uj\_board);

A bekért új elemet hozzáfűzi a már meglévő láncolt listához. Ehhez át kell adnunk a lista kezdő elemére mutató pointert, valamint a hozzáfűzni kívánt snowboard típusú elemet. Végigfut a láncolt lista elemein, majd az utolsó elem következő elemre mutató NULL pointerét átállítja, hogy az új elemre mutasson.

* 1. snowboard\* board\_torol

(snowboard \*head\_board, char \*azonosito)

Eltávolítja a láncolt listából a törölni kívánt elemet. Ehhez át kell adnunk a lista head pointerét, valamint a törölni kívánt elem azonosítóját. Ezután addig megyünk a listában, amíg a következő elem azonosítója nem egyezik meg a törölni kívánt board azonosítójával. Megjegyezzük ezt az elemet. Majd annak az elemnek, amin épp állunk a következő elemre mutató pointere, a törölni kívánt elem következő elemre mutató pointer lesz. Végül pedig felszabadítjuk a törölt elem által foglalt memóriát.

* 1. void board\_kiir(snowboard \*board);

Átadunk a függvénynek egy board típusú elemet, melyet kiír a standard outputra. A függvénynek nincs visszatérési értéke.

* 1. void board\_listaz(snowboard \*head\_board);

Végigmegy a snowboardokból épített láncolt listán és mindegyiket kiírja a standard outputra, visszatérési értéke nincs.

1. Bérlőket kezelő függvények:
   1. berlo\* berlo\_beolvas();

Ez a függvény a program furásának leg elején a lényeges, amikor beolvassa az adatokat a BERLOK.TXT-ből és felépíti belőlük a bérlők láncolt listáját. A végén pedig vissza adja a lista első elemére mutató pointert.

Head bérlő

|  |
| --- |
| Név |
| Testtömeg |
| Testmagasság |
| Lábméret |
| Szint |
| Bérlés időtartama |
| Azonosító |
| \*kovetkezo\_berlo |
| \*kov\_berles |

|  |
| --- |
| Név |
| Testtömeg |
| Testmagasság |
| Lábméret |
| Szint |
| Bérlés időtartama |
| Azonosító |
| \*kovetkezo\_berlo |
| \*kov\_berles |

|  |
| --- |
| Név |
| Testtömeg |
| Testmagasság |
| Lábméret |
| Szint |
| Bérlés időtartama |
| Azonosító |
| \*NULL |
| \* kov\_berles |

* 1. berlo\* berlo\_beker();

A standard inputrol kér be adatokat melyből létrehoz egy snowboard típusú elemet és ezt adja vissza.

* 1. void berlo\_hozzaad

(berlo \*head\_berlo, berlo \*uj\_berlo);

A bekért új elemet hozzáfűzi a már meglévő láncolt listához. Ehhez át kell adnunk a lista kezdő elemére mutató pointert, valamint a hozzáfűzni kívánt berlo típusú elemet. Végigfut a láncolt lista elemein, majd az utolsó elem következő elemre mutató NULL pointerét átállítja, hogy az új elemre mutasson.

* 1. berlo\* berlo\_torol

(snowboard \*head\_board, berlo \*head\_berlo, nev neve);

Egy törölni kívánt bérlő típusú elem lehetséges, hogy nem csak láncolt listát alkot a többi bérlővel, hanem a fésűslistának is eleme, amit a boardokkal alkotnak. Ehhez át kell adnunk a snowboardok és a bérlők listájának head pointerét, valamint a törölni kívánt bérlő nevét. Ezután addig megyünk a bérlők listájában, amíg a következő bérlő neve nem egyezik meg a törölni kívánt bérlő nevével. Megjegyezzük ezt az elemet. Majd annak az elemnek, amin épp állunk a következő elemre mutató pointere, a törölni kívánt elem következő elemre mutató pointer lesz. Ezután még a fésűs listából is el kell távolítanunk (lásd: 4.3 fesus\_lista\_kivesz függvény). Végül pedig felszabadítjuk a törölt elem által foglalt memóriát.

* 1. void berlo\_kiir(berlo \*berlo1);

Átadunk a függvénynek egy berlo típusú elemet, melyet kiír a standard outputra. A függvénynek nincs visszatérési értéke.

* 1. void berlok\_listaz(berlo \*head\_berlo);

Végigmegy a bérlőkből épített láncolt listán és mindegyik elemét kiírja a standard outputra. Visszatérési értéke nincs.

1. Fésűs lista:
   1. int fesus\_lista\_befuzes

(snowboard \*head\_board, berlo \*berlo1);

A függvénynek át kell adnunk a boardok láncolt listájának head pointerét, valamint egy bérlőt, akit be szeretnénk fűzni a listába. Addig megyünk a boardok listájában, amíg nem találunk olyan snowboardot, aminek az azonosítója megegyezik a függvénynek átadott bérlő azonosítójával. Ha megtaláltuk, akkor a board az őt bérlőkre mutató pointerét átállítjuk, hogy az átadott bérlőre mutasson a bérlő kov\_berles pointere pedig az lesz, amire eddig a board berlo pointere mutatott.

* 1. void fesus\_lista\_inditas

(snowboard \*head\_board, berlo \*head\_berlo);

Átadjuk a függvénynek a boardok és a bérlők láncolt listáinak head pointereit. Végigszaladunk a bérlők listáján és befűzzük őket a fésűs listába.

* 1. void fesus\_lista\_kivesz

(snowboard \*head\_board, berlo \*berlo1);

A függvénynek át kell adnunk a boardok listájának head pointerét és azt a bérlőt, akit törölni szeretnénk. Egy for ciklussal elindulunk a boardok láncolt listáján. Ha a törölni kívánt bérlő azonosítója megegyezik a board azonosítójával, elindulunk „a fésű fogain” és keressük azt a bérlőt, akit törölni szeretnénk. Majd amikor megtaláltuk kivesszük a listából. Ha ez volt az utolsó elem az adott „fésű fogban”, tehát a board bérlőre mutató pointere NULL pointer, akkor kilépünk a ciklusból. Ha azonban ez nem NULL, megyünk tovább, amíg meg nem találjuk. Aztán ott is elvégezzük a szükséges lépéseket az eltávolításhoz.

* 1. void fesus\_lista\_listazas(snowboard \*head\_board);

Elindulunk a boardok listáján, ehhez csupán a head\_boardra van szükségünk. Minden boardot kiírunk és ha a board bérlőre mutató pointere nem NULL, tehát bérlik, akkor kiírjuk a board után, hogy ki és mikor tart rá igényt.

1. A feladatok megoldásához szükséges függvények:
   1. int berles\_idotartama(berles berles1);

A függvény egy berles típusú változót kap paraméterként, ami két dátum típusú változót tartalmaz. Ezekből az adatokból kiszámolja a két időpont eltelt időt és ezt az int típusú értéket adja vissza. A bérlés árának kiszámolásakor lesz rá szükség.

* 1. int napi\_egysegar(snowboard board1);

A paraméterként kapott snowboard típusú változó tartalmazza az adott board vásárlási évét és az addig bérelt napokat is. A program ezek ismeretében kiszámolja, hogy mennyibe kerülne egy napi bérlése az adott deszkának és ezt az értéket adja vissza.

* 1. datum datum\_beolvas();

A standard inputról olvas be adatokat (év hónap nap) és ezeket egy dátum típusú változóban tárolja el, ami a függvény visszatérési értéke lesz.

* 1. double labmeret\_atvalto(double meret);

A LABMERETEK.TXT fájlból olvas a függvény, egyesével nézi végig az elemeit, amíg nem talál egyezést a kapott változóval. Ekkor vissza adja a txt szerinti eredmény, hogy mely értéknek mi felel meg.

* 1. void leselejtezendo(snowboard \*head\_board);

Paraméterként a snowboardok láncolt listájának első elemére mutató pointert kapja. Bekér egy évet a standard inputról és ezt hasonlítja össze, hogy melyek a 10 vagy annál régebbi deszkák, amiket le kell selejtezni.

* 1. snowboard\* legtobbet\_berelt\_deszka

(snowboard \*head\_board);

Végig kell járnunk a snowboardok láncolt listáját, ehhez szükségünk van a lista head pointerére. Definiálunk egy maximumot, ez tárolja azt, hogy mennyi volt eddig a legtöbb bérelt nap egy adott deszkánál. Kezdetben nullára állítjuk feltételezve, hogy ennél biztosan nagyobb lesz. Majd a függvény vissza adja arra a snowboardra mutató pointert, amelyik a legtöbbet volt bérelve a nyilvántartás szerint.

* 1. int board\_szabad(snowboard \*board, berlo \*berlo2);

A függvény azt vizsgálja, hogy a bérlő által adott időpontban a számára megfeleő boardok közül melyek azok, amik szabadok is ebben az időpontban, tehát, hogy nem bérelte ki más, a bérlő által kívánt időponttal átfedésben. Ehhez szükség van a vizsgált boardra mutató pointerre, valamint a bérlőre. A berlo típus magában hordozza az adott személy által bérelni kívánt időtartamot. Az adott snowboardnál ahhoz, hogy megtudjuk szabad-e. A board hoz kapcsolódó bérlők listáját kell vizsgálnunk. A függvény visszatérési értéke 1 vagy 0, attól függően szabd vagy nem szabad a board a kívánt intervallumon.

* 1. void berles\_ara

(snowboard \*head\_board, berlo \*head\_berlo);

Ebben a függvényben először bekérjük egy bérlő adatait a standard inputról, majd hozzáfűzzük a bérlők láncolt listájához. Ezután megvizsgáljuk, mely boardok lennének megfelelőek a bérlő számára. Ezeket kiírjuk a standard outputra, így már az ár és a deszkák ismeretében választhat egyet. Ezután ehhez a deszkához a fésűs listához is hozzá fűzzük az új bérlőt.

1. Lezáró függvények:
   1. void boardok\_fajlba\_irasa(snowboard \*head\_board);

Annak érdekében, hogy a program lefutása során történő változtatások, új boardok felvétele, törlése ne vesszen el és a program ne mindig ugyan onnan induljon (ugyanis az nem lenne túlságosan életszerű) ezzel a függvénnyel kiírom a snowboardok láncolt listáját a BOARDOK.TXT-be. Ezzel mentve a változtatások, valamint biztosítva a program jó működését a későbbiekben is.

* 1. void berlok\_fajlba\_irasa(berlo \*head\_berlo);

Az előző függvénnyel szint teljesen azonos, csak itt a bérlők láncolt listáját írom ki a BERLOK.TXT-be. Így a program újbóli indításakor a fésűslista betöltésnál a már korábban felvett bérlők is ott lesznek, amíg nem hívják meg azt a funkciót, hogy lejárt a bérlés és vissza hozták a deszkát.

Az előbbieken és a main-en kívül még egy függvényem van: void menu(); melyben a parancssori menüt valósítom meg, valamint a korábban írt függvényeket hívom meg a feladatok megoldásához.

A snowboard kölcsönző nevű programmal egy egyszerű, könnyen kezelhető nyilvántartás létrehozás volt a cél, mely a kölcsönző tulajdonosának és a bérlőknek is a javára válik.