Legyen Ön Is Milliomos - Programozói dokumentáció

Módszerek áttekintő magyarázata

A program a használat egyszerűségére van kihegyezve, tehát mindig reagál a felhasználó válaszaira, és mindig egyértelműen határozza meg azt, hogy milyen bemenetet vár, amit mindig le is ellenőriz, hogy érvényes-e, és ha nem, új bemenetet vár. A kis- és nagybetűket a válaszadásnál nem különbözteti meg. Ha hibát észlel a program, azt kiírja, és az adott függvény egy olyan számmal tér vissza, ami a hiba típusára jellemző, ezek a dokumentáció végén találhatóak meg. A kérdéseket és a sorkérdéseket is egy ciklikus láncolt listában tárolja. Minden kérdésfeltevés előtt generál egy random számot 0 és a kérdések száma között, és annyit lép előre a listán, amennyi ez a szám. Ekkor megjelöli ezt a kérdést, hogy már feltette, ez biztosítja nekünk, hogy egy kérdést sosem fog kétszer feltenni. A függvények alapvetően a menübe térnek vissza, ami egy végtelen ciklusba van téve, így mindig újra meg tud jelenni. A fájlok kezelése úgy van megoldva, hogy beolvassa az egész fájlt a program, elvégzi a módosításokat, majd visszaírja az új tartalmat.

Adatszerkezetek

A legjelentősebb adatszerkezet a ciklikus láncolt lista, amiben a kérdéseket, és a sorkérdéseket tárolja a program, miután a fájlból beolvasta. A „sima” kérdésekhez és a sorkérdésekhez nem azonos struktúra tartozik, és nem is egy időben léteznek.

Egy sorkérdést tároló struktúra:

```
typedef struct Sorkerdes{
char kerdess[200];char a[100]; char b[100]; char c[100]; char d[100];
char valasz[5]; char kategoria[40]; struct Sorkerdes* kov;
}Sorkerdes;
```

Egy kérdést tároló struktúra:

```
typedef struct Kerdes{
int nehezseg; char kerdess[200]; char a[100]; char b[100]; char c[100];
char d[100]; char valasz; char kategoria[40]; bool volt_e; struct Kerdes* kov;
}Kerdes;
```

Mindkét esetben: a kerdess a kérdést tárolja, az a, b, c, és d a válaszokat, a kategoria a kategóriát, a kov értelemszerűen a láncolt lista következő elemére mutat.

Sorkérdés: itt az 5 elemű tömb első 4 elemébe a válasz kerül, az 5.-be pedig lezáró 0, ennek a kiíratásnál van szerepe.

Kérdés: valasz: egy karakter, a helyes válasz betűjele; volt_e: ez jelöli, hogy az adott kérdés fel volt-e már téve az adott játékban.

A láncolt listákat a megfelelő helyeken a program felszabadítja.

A beolvasott dicsőséglista egy struktúrából alkotott tömbben tárolódik, amíg a program dolgozik rajta:

```
typedef struct Dicsoseg{
int helyezes; char nev[50]; int nyeremeny; unsigned ido;
}Dicsoseg;
```

Itt a változók neveiből kiderül, hogy mit tárolnak.

Játék közben a játékos adatai is struktúrában tárolódnak. A program eltárolja a nevet, a megszerzett nyereményt, a felhasznált segítségeket, és a játékidőt. Ezeket az adatokat a játék végén átadja a dicsőséglistát kezelő függvénynek, ami ezek alapján eldönti, hogy felrakja-e a dicsőséglistára a játékost. Egy játékos adatait tároló struktúra:

```
typedef struct Jatekos{
char nev[50]; int nyeremeny; bool segitsegek[3]; unsigned ido;
}Jatekos;
```

A program további, kevésbé jelentős változókat is használ, melyekben a válaszokat, és ideiglenesen beolvasott karaktereket tárol.

Modulok, függvények

main.c: a menüt tartalmazza.

Függvényei:

char menu(): kiírja a lehetőségeinket, és kér egy karaktert.

int main(): üdvözli a felhasználót, majd a menu() függvénytől kapott karakter szerint cselekszik. Ha ez a karakter:

1: Új játék kezdődik
2: Kiírja a dicsőséglistát
3: Kilép

Visszatérési értéke: 0, ha minden rendben ment.

jatek.c: a játék menetét bonyolítja le.

Függvényei:

int felszabadit(Kerdes* kerdes, int kerdesszam): felszabadítja a láncolt listát. Paraméterei: a láncolt lista első elemére mutató pointer, és a láncolt lista elemeinek száma. Visszatérési értéke: 0, ha rendben ment.

int segitsegszam(bool* segitsegek): megszámolja, hogy a felhasználó hány segítséget használt fel. Paramétere: a segítségek felhasználtságát nyilvántartó, logikai értékeket tároló tömbre mutató pointer. Visszatérési értéke: az az egész szám, amennyi segítséget a játékos felhasznált.

int jatek(): a tényleges játékmenet ezen a függvényen belül történik. Először meghívja a sorkérdés-függvényt. Majd beolvassa a dicsoseg.txt-ből a kérdéseket, és egy ciklikus láncolt listába eltárolja. Kiírja, hogy hány kérdést sikerült beolvasnia. Létrehozza a **Jatekos** strukúra egy példányát, ezután bekéri a nevünket, és hogy milyen nehézségen szeretnénk játszani. Elkezdi mérni az időt. A kérdésekre 8-féleképpen válaszolhatunk. Ha a válaszunk A, B, C, vagy D, akkor a program leellenőrzi, hogy helyes-e a válaszunk, és ha igen, jön a következő kérdés. Ha nem helyes a válaszunk, vége a játéknak. Ha a válaszunk T, F, vagy K, akkor a telefon, a felezés, és a közönség-segítséget használhatjuk fel, amennyiben még rendelkezésre áll. Ha a válaszunk M, akkor megállunk, és elvisszük az addig megnyert pénzt. Ha bármi mást válaszolunk, az érvénytelen. A játék végén megáll az időszámláló, és a játékidőnket, a felhasznált segítségeink számát, a nyereményünket, és a nevünket továbbítja a dicsoseg modulnak feldolgozásra. Ha minden kérdésre jól válaszolunk, gratulációt kapunk. A függvényből való kilépéskor minden esetben meghívja a láncolt listát felszabadító függvényt.

Visszatérési értékei:

0, ha minden rendben ment.
1, ha gond volt a memóriafoglalással.
2, ha gond volt a fájlmegnyitással.
3, ha gond volt a sorkérdésekkel.
4, ha gond volt a dicsőséglistával.
5, ha gond volt a memóriafelszabadítással.

sorkerdes.c: a sorkérdést bonyolítja le.

Függvényei:

`bool jovalasz(char *valasz, char vizsgal):` megvizsgálja, hogy az általunk a sorkérdésre megadott válasz egyezik-e az adott sorkérdéshez tartozó válasszal.Paraméterei: a saját, és a helyes válaszokat tároló tömbökre mutató pointerek. Visszatérési értéke: igaz, ha a válaszunk helyes, és hamis, ha a válaszunk helytelen.

`int sorkerdes():` beolvassa fájlból a sorkérdéseket, és eltárolja egy ciklikus láncolt listában. Kiírja, hány sorkérdést sikerült beolvasnia. Létrehozza a 7 virtuális ellenfelet, és random választ és időt ad nekik, sorba rendezi őket idő szerint növekvően, és a jó választ adóakat az elejére rendezi. Generál egy random sorkérdést. Erre vár a felhasználótól egy választ, majd leellenőrzi, hogy jó-e a megadott válasz. Ha a válaszunk nem jó, akkor nem kezdhetjük el a játékot, a függvény visszatér. Ha jó a válaszunk, megvizsgálja, hogy a legkisebb idejű ellenfél jó választ adott-e, és ha igen, akkor az ideje kisebb-e mint a miénk. Ha ez igaz, akkor kiírja, hogy egy ellenfél gyorsabb volt nálunk, és nem kezdhetjük el a játékot. Ekkor kiírja a játékosokat is sorrend szerint, válasszal és idővel. Ha mi voltunk a leggyorsabb jó választ adók, akkor a is kiírja ezeket az adatokat, és gratulációt is kapunk. A függvény visszatérés előtt mindig felszabadítja a sorkérdések láncolt listáját, amennyiben azt már létrehozta.

Visszatérési értékei:

0, ha minden rendben ment.
1, ha gond volt a memóriafoglalással.
2, ha gond volt a fájlmegnyitással.
4, ha gond volt a dicsőséglistával.
6, ha a felhasználó rossz választ adott a sorkérdésre.
7, ha a felhasználó jól válaszolt a sorkérdésre, de valamelyik virtuális ellenfél gyorsabb volt.

segitseg.c: a segítségkérésekre reagál.

Függvényei:

int segitseg(Kerdes* kerdes, char milyen, char* megmaradtak): ha „milyen” a T karakter, akkor az átadott kérdésre 80%-os eséllyel a jó tippet kiírja. Ha a „milyen” F, akkor a jó választ, és a rossz válaszok közül egyet kiír a képernyőre. Ha a „milyen” K, akkor a közönség szavaz, és ennek a szavazásnak az eredményét írja ki a képernyőre. Paraméterei: a kérdésre mutató pointer, a segítségkérés típusa, és a felezés után megmaradt válaszlehetőségek kételemű karaktertömbjére mutató pointer.

Visszatérési értéke: 0, ha minden rendben ment.

dicsoseg.c: a dicsőséglistát kezeli.

Függvényei:

`int dicsoseg_ir(char* nev, int nyeremeny, unsigned ido, int segitsegek):` megkapja egy játékos paramétereit. Leellenőrzi, hogy létezik-e a dicsőséglista, és ha nem, akkor létrehozza. Ha létezik, akkor beolvassa egy Dicsoseg struktúrából alkotott tömbbe a dicsőséglistát, majd megnézi, hogy a játékost az eredményei alapján hanyadik helyre tudja berakni. Amikor felkerül a dicsőséglistára a játékos, az ő helyén lévő és az utána lévők egy helyet lejjebb csúsznak. Miután ezzel végzett, az adatokat visszaírja a fájlba.

Visszatérési értékei:

0, ha minden rendben ment.
8, ha a játékos nem került fel a dicsőséglistára.

`int dicsoseg_olvas():` megnyitja a dicsőséglista fájlját, és a tartalmát kiírja a képernyőre. Ha nem talált dicsőséglistát, azt is közli a felhasználóval.

Visszatérési értékei:

0, ha minden rendben ment.
2, ha nem találta meg a dicsőséglistát

Minden modulhoz léteznek header-fájlok is, melyekben deklarálva vannak az adott modul adatszerkezetei és függvényei. Ha az adott adatszerkezetre vagy függvényre másik modulnak is szüksége van, akkor az adott modul include-olva van a megfelelő helyen.

Az azonos jelenségekre vonatkozó visszatérési értékek:

0: minden rendben
1: memóriafoglalási hiba
2: fájl-megnyitási hiba
3: hiba a sorkérdésekkel
4: hiba a dicsőséglistával
5: hiba a memória-felszabadítással
6: rossz válasz a sorkérdésre
7: egy virtuális ellenfél gyorsabban adott jó választ a sorkérdésre
8: a játékos nem került fel a dicsőséglistára

A program UTF-8 kódolást használ.