Programozói dokumentáció

Működés

A `main()` funkció a program összes lefutásakor betölti az adatokat, valamint a végén elmenti és felszabadítja azokat. A futtatáskor a felhasználó által megadott argumentumok alapján dől el, hogy milyen funkciók kerülnek majd meghívásra.

Ha a program először fut le, ignorálja a megadott extra argumentumokat és megkéri a felhasználót, hogy állítsa be az alap paramétereket (legalább egy asztal és egy menü elem).

Bármely lefutáskor ha nincs megadva argumentum, akkor a program menü üzemmódban fut addig, amíg a felhasználó le nem állítja (ez a funkció is elérhető a programban).

A program a CLion fejlesztőkörnyezetben lett megírva, fordításához szükséges egy C complier, javasolt a CLion-ba épített fordító.

A program építőelemei

A különböző funkciók paraméterei a kódban részletesen is el vannak magyarázva. A programban a fájl- és funkció/struktúra nevek angolul szerepelnek, de mindenhez tartozik részletes magyar nyelvű magyarázat.

A program `main()` funkciója végzi el a parancsok kezelését, illetve első használat esetén az alap adatok felvételét is.

Input

A felhasználóval való kommunikációt segítő funkciók.

Fájl: `io.c` / `io.h`

```
void printWelcome()
```

Üdvözli a felhasználót a program első lefutásakor, vagy a menü üzemmódban.

```
MenuItem requestMenuItem()
```

Létrehoz egy új `MenuItem` objektumot, majd beolvassa szükséges adatokat hozzá a konzolról (az étel neve és ára). A létrehozott objektumot visszaadja, így az később hozzáfűzhető egy láncolt listához.

`Table requestTable()`

Létrehoz egy új `Table` objektumot, majd beolvassa a szükséges adatokat hozzá a konzolról (asztal helye és kapacitása). A létrehozott objektumot visszaadja, így az később hozzáfűzhető egy láncolt listához.

`char *readString(char end)`

Dinamikus `string`-be olvas a konzolról, a megadott karakterig.

`char *readStringLine(char end)`

A `\n` karakterig olvas be a konzolról, dinamikus `string`-be.

`bool verifyArgc(int argc, int requested)`

Ellenőrzi, hogy a megfelelő számú argumentumot kapta-e meg a program a meghívott parancshoz.

`char *equalSpace(unsigned long strlen, unsigned long linelen)`

Készít egy szóközökből álló stringet, ami használható formázott kiírásokhoz, például ugyan akkora hely kihagyásához minden sorban.

`size_t istrlen(int num)`

Kiszámolja, hogy egy `integer` milyen hosszú string `%d` formátumban.

Generikus láncolt lista

Fájl: `linkedlist.c` / `linkedlist.h`

A programban külön modul foglalkozik a láncolt listák kezelésével. A generikus láncolt lista implementáció lehetővé teszi az asztalok, a menün szereplő ételek és a rendelések moduláris kezelését a C `void*` típusának használatával.

`ListItem` adatstruktúra

A struktúra leírja, hogy hogyan néz ki egy adott elem a láncolt listában. A *data* `void` típusú pointer tárolja a memóriacímet az adott lista elem értékéhez, míg a *next* `ListElem` típusú pointer a következő elemre mutat a listában. Amennyiben a következő elem nem létezik, az értéke `NULL`.

```
typedef struct ListItem {
    void *data;
    struct ListItem *next;
} ListItem;
```

`ListItem *push(ListItem *list, void *data)`

Hozzáad a láncolt lista végéhez egy új elemet, majd visszaadja a lista első elemére (*head*) mutató pointert. Ez azért fontos, mert ha a lista eddig üres volt, akkor ez az első elem maga a hozzáadott elem lesz, így a *head* változik.

> **Fontos:** Az új ListItem típusú pointert fel kell szabadítani, ha már nincs rá szükség. Ez a programozó felelőssége. A `*data` pointer csak dinamikusan lefoglalt memóriára mutathat, mert a `freeList()` funkció később felszabadítja.

`ListItem *removeItem(ListItem *list, int index)`

Kiszed a láncolt listából egy elemet, majd visszaadja a lista első elemére (*head*) mutató pointert. Hasonlóan a `push()` funkcióhoz, a *head* változhat, ha az első elemet távolítjuk el, ezért fontos a visszatérési érték.

> **Megjegyzés:** A funkció automatikusan felszabadítja az eltávolított elem memóriáját.

`void freeList(ListItem *list)`

Amennyiben már nincsen szükség a lista egyik elemére sem, ez a funkció végigmegy rajta és felszabadítja az összes elemet, valamint az értékeit is.

`ListItem *getItemByIndex(ListItem *list, int index)`

Visszaad egy elemet egy generikus láncolt listából a megadott index alapján. Ha nem találja az elemet, `NULL` pointert ad vissza.

`size_t getListLength(ListItem *list)`

Megszámolja, hogy milyen hosszú egy adott láncolt lista (vagyis hány elem van összeláncolva benne).

Példa a generikus lista használatára

A program fontos része a generikus láncolt lista kezelő modul, mert három féle adattípust is ezzel tárolunk, így indokolt használatának példásítása.

Vegyünk egy egyszerű láncolt listát, ami `int` típusú értékeket tárol. Ezt a következő módon hozhatjuk létre:

```
// egy üres lista inicializálása
// ez a változó tartalmazza a lista "head"-jét
// (vagyis a lista első elemére mutató pointert)
ListItem *listanev = NULL;

// hozzuk létre az első elem értékét
// ez BÁRMILYEN típusú lehet
int elso = 34;

// hozzunk létre egy új lista elemet
listanev = (ListItem*) malloc(sizeof (ListItem));

// végül adjuk meg az értékét a korábban
// megadott értékre mutató pointerrel
listanev->data = &elso;

// ne felejtsük el felszabadítani a listát később
freeList(listanev);
```

A listához egyszerűbben is hozzáfűzhetünk új elemeket (akár a létrehozásnál/első elemnél is):

```
// hozzuk létre a második elem értékét
int masodik = 87;

// a lista végéhez fűzzük
listanev = push(listanev, &masodik);
```

```
// ne felejtsük el felszabadítani a listát később
freeList(listanev);
```

A listából természetesen törölhetünk is elemet:

```
// a masodik elem törlése
listanev = removeItem(listanev, 1);

// a lista maradékát így is fontos felszabadítani
freeList(listanev);
```

Menü

Az étterem menüjét kezelő funkciók.

Fájl: `menu.c` / `menu.h`

`MenuItem` adatstruktúra

Leírja, hogy hogyan épül fel egy menün szereplő elem. Tartalmazza az integer típusú árat (Forintban), valamint a nevét, ami egy dinamikus `string` első karakterére mutató `char` pointer. Használható láncolt lista építésére a generikus láncolt lista modullal, ha egy ilyen adatstruktúrára mutató pointert adunk meg a lista értékére.

```
typedef struct MenuItem {
    char *name;
    int price;
} MenuItem;
```

`ListItem *loadMenu()`

A funkció betölti a már elmentett menü elemeket a `menu.txt` fájlból. Végigmegy a fájl összes során, dinamikus `string`-be olvassa azokat, majd azt feldolgozva beolvassa az étel árát és a nevét (szintén dinamikusan).

A funkció a generikus láncolt lista modulra épül, így a menüt is egy ilyen listába olvassa.

> **Fontos:** A visszaadott elemeket és a dinamikusan beolvasott étel neveket a funkció meghívójának saját felelőssége felszabadítani.

> Az utóbbira külön kell figyelni, ha eltávolítanak egy elemet a menü listából, hogy ne legyen memóriaszivárgás.

`void saveMenu(ListItem *list)`

Elmenti az adott menü láncolt lista elemeit a `menu.txt` fájlba. A lista összes elemét egy-egy sorba rakja, ahol az étel nevét és árát egy tabulátor választja el.

`void freeMenu(ListItem *list)`

Felszabadítja a menü láncolt lista elemeit, valamint a dinamikusan beolvasott étel nevét is.

`void printMenu(ListItem *list)`

Kiírja a menü lista elemeit, formázva. Ez a funkció használható a menü megjelenítésére a felhasználónak.

`ListItem *newMenuItem(char *name, int price, ListItem *list)`

Felvesz egy új elemet a menüre (hozzáadja a listához). A funkció használható a felhasználótól kapott input-tal.

`ListItem *removeMenuItem(int index, ListItem *list)`

Töröl a menüről egy elemet. A felhasználótól kapott input-tal használható.

Asztal

Az asztalokat kezelő funkciók.

Fájl: `table.c` / `table.h`

`Table` adatstruktúra

Az adatstruktúra leírja az asztalokat tároló változók felépítését. Tárolja az asztal helyét az étteremben (x és y koordinátával), hogy hány ember fér el körülötte, valamint hogy foglalt-e. Használható láncolt lista építésére a generikus láncolt lista modullal, ha egy ilyen adatstruktúrára mutató pointert adunk meg a lista értékére.

```
typedef struct Table {
    int x, y;
```

```
    int capacity;
    bool occupied;
} Table;
```

`ListItem *loadTables()`

Betölti az asztalokat a `tables.txt` fájlból. Végigmegy a fájl összes során és beolvassa a struktúra összes adatát.

> **Fontos:** A visszaadott elemeket a funkció hívójának felelőssége felszabadítani.

`void saveTables(ListItem *list)`

Elmenti az asztalokat a `tables.txt` fájlba. Minden elemet a saját sorába menti, az értékeket tabulátorral választja el.

`ListItem *newTable(int x, int y, int capacity, ListItem *tableList)`

Új asztalt vesz fel a felhasználó által megadott adatok alapján, mely ez után megnyitható lesz.

`ListItem *openTable(int index, ListItem *tableList)`

Megnyit egy létező asztalt vendégek számára. Ez foglaltként jelöli meg az asztalt, melyhez ez után lehet csak rendeléseket leadni.

`ListItem *setTableOccupied(int index, ListItem *tableList, bool occupied)`

Belső funkció, ami megkeresi az index-szel megadott asztalt, majd beállítja az állapotát foglaltra vagy szabadra, attól függően, hogy milyen értéket kap (`occupied` változó).

`void printTableMap(ListItem *tableList)`

Megrajzolja a konzolon a felhasználó számára az étterem térképét. Jelzi, hogy melyik asztal foglalt, illetve melyik szabad.

Rendelés

A rendeléseket kezelő funkciók.

Fájl: `order.c` / `order.h`

`Order` adatstruktúra

Az adatstruktúra leírja a rendeléseket tároló változók felépítését. Az asztal és az adott menü elem indexét tárolja.
Használható láncolt lista építésére a generikus láncolt lista modullal, ha egy ilyen adatstruktúrára mutató pointert adunk meg a lista értékére.

```
typedef struct Order {
    int table;
    int food;
} Order;
```

`ListItem *loadOrders()`

Betölti az asztalokat az `orders.txt` fájlból. Végigmegy a fájl összes során és beolvassa az összes indexet.

> **Fontos:** A visszaadott elemeket a funkció hívójának felelőssége felszabadítani.

`void saveOrders(ListItem *list)`

Elmenti a rendeléseket az `orders.txt` fájlba. Minden elemet a saját sorába menti, a két érték között tabulátor található.

`ListItem *orderNewItem(int menuItemIndex, int tableIndex, ListItem *menuList, ListItem *tableList, ListItem *orderList)`

Felvesz egy új rendelést a rendelések láncolt listájába. Megkeresi a megadott étel- és asztal indexét, majd a láncolt listába helyezi ezeket új elemként. Az asztalhoz való számla generálásánál törlődik.

`BillResult` adatstruktúra

Egy egyszerű struktúra, mely tárolja az asztal- és rendelések láncolt listák fejére mutató pointereket.

```
typedef struct BillResult {
    ListItem *tableList;
    ListItem *orderList;
} BillResult;
```

`BillResult issueBill(int tableIndex, ListItem *menuList, ListItem *tableList, ListItem *orderList)`

A funkció számlát generál a megadott asztalhoz. Kiírja a felhasználónak az asztalhoz rendelt összes ételt, illetve az összegzett árat is. Ezt követően törli az asztalhoz tartozó rendeléseket a rendelések láncolt listából és átállítja a megadott asztal állapotát “nem foglalt”-ra.

GUI

A menü üzemmódot kezelő funkciók.

Fájl: `gui.c` / `gui.h`

`int showAndChooseMenu()`

Megjeleníti a felhasználó számára a grafikus menüt, majd beolvassa a kiválasztott menüpont számát. Ha az nem egy létező szám, rekurzív módon újra megkéri a felhasználót egy menüpont választására.

`int getTableIndex()`

Elkér a felhasználótól egy asztal számot, majd az asztal indexét (szám - 1) adja vissza.

`int getMenuItemIndex()`

Elkéri a felhasználótól egy a menün szereplő étel számát, majd az elem indexét (szám - 1) adja vissza.

`GuiHandleResult handleMenu(int selected, ListItem *menuList, ListItem *tableList, ListItem *orderList)`

Ez a funkció a megadott menüpont alapján bekéri az ahhoz szükséges adatokat a felhasználótól, majd ugyan azokat a funkciókat hívja meg, amiket a normál üzemmódban hívnak meg a parancsok.

Debugmalloc

Fájl: `debugmalloc.h`

Debugoláshoz használt külső modul, ami segít feltárni az esetleges memóriaszivárgásokat. A hozzá tartozó dokumentációt a *header* fájl tartalmazza.