MŰEGYETEM 1782

A PROGRAMOZÁS ALAPJAI 2.

HÁZI FELADAT DOKUMENTÁCIÓ

LIFT

KÉSZÍTETTE: GÁSPÁR BÁLINT, ANNBWP
gasparbalint13@gmail.com

KÉSZÍTÉS FÉLÉVE: 2018/19/2

Tartalomjegyzék

Felhasználói dokumentáció

A program feladata egy lift működésének szimulálása. A program bemenetén txt- fájlban várja a „kéréseket”. A kérés egy olyan nemnegatív számokból álló számpárost jelent, aminek az első tagja a kiindulási, második tagja az érkezési emeletet jelöli. Az emeleteknek van egy maximális lehetséges értéke (static const unsigned MAX_LEVEL), a földszintnél lejjebb a lift nem mehet. A bemeneten a kérések tagjait szóközzel elválasztva, az egyes kéréseket külön sorban várja a program. Amennyiben a kérések nem ebben a formátumban szerepelnek a txt-ben, vagy az emeletek nem a megfelelő tartományon belülre esnek, a program a felhasználót a megfelelő hibaüzenettel és hibakóddal tájékoztatja erről.

A fájl beolvasásakor a kérések egy listába kerülnek. (A listában lévő sorrendjük megegyezik a szöveges fájlban szereplő sorrenddel.) Miután a lift átveszi ezt a listát, rögtön elkezdi teljesíteni őket. A lift kezdetben a földszintről indul. A lift először megnézi, hogy mely kéréseket tudja teljesíteni az első kéréssel egy időben, tehát melyek azok a kérések, amelyeknek a kiindulási és célemelete az első listaelem kiindulási és célemelete közé esik, továbbá iránya (starttól a célhoz) megegyezik az első listaelemével. A lift ezeket a kéréseket együtt teljesíti, majd eltávolítja a listából. Ezek után a folyamat kezdődik elölről mindaddig, amíg a listában van még teljesítetlen kérés.

A programnak két kimenete van: ír egy szöveget, ami megörökíti a lift mozgását („lift_log.txt”), illetve egy ehhez hasonló kimeneti szöveget a standard outputon is megjelenít. A szövegben szerepel, hogy a lift hol nyitja illetve csukja az ajtaját, valamint becsukás után kiírja, hogy a lift melyik emelethez indul el.

Az egyes hibaüzeneteket standard outputon jeleníti meg (fájlba nem írja).

Az alkalmazás nem használ menüt, a felhasználónak csak a forrásfájlon keresztül van lehetősége a programmal interakcióba lépni.

Osztályok statikus leírása

Request

Felelőssége

Példányai tárolják az egyes kéréseket, függvényeire a listákba rendezéseknél van szükség.

Attribútumok

static const unsigned MAX_LEVEL=18: Statikus konstans, amely az emeletek maximumát jelöli. Statikus, mert minden kérésre általánosan igaz, és konstans, mert a program lefutása során nem változik az épület emeleteinek száma.

Privát

- unsigned start: a kérés kiindulási emelete
- unsigned destination: a kérés célemelete

Metódusok

Publikus

- Request(unsigned=0, unsigned=1); : A Request osztály konstruktora. Alapértelmezett értékek használatával helyettesítve van a default konstruktor. Az alapértelmezett értékeknek különbözniük kell, ugyanis a program ellenőrzi a bejövő kéréseket, és az is hibás kérésnek számít, melynek kiindulási és célemelete megegyezik (ERROR 3)
- Request(const Request&); : Másoló konstruktor, amely inicializációs listán teszi egyenlővé az új példány tagváltozóit a paraméterként kapott elem tagváltozóival.
- bool operator==(const Request&) const; : A többi osztály függvényeihez szükséges operátor.
- bool operator!=(const Request&) const; : A többi osztály függvényeihez szükséges operátor.

- bool isMovingUp() const; : Értéke 'true', ha a kérés célemelete nagyobb, mint a kiindulási emelete; ellenkező esetben 'false'.
- bool isInbetween(const Request&); : Azt vizsgálja, hogy paraméterként kapott kérés teljesíthető-e azzal egy időben, amire meghívtuk.
- unsigned getStart() const; unsigned getDestination() const; - a privát tagváltozók eléréséhez szükséges getterek.
- ~Request(); - Default konstruktor (nem történik a Request osztályon belül dinamikus memóriafoglalás, ezért elég a default)

fifo

Felelőssége

Feladata a bemeneti fájl feldolgozása, a kérések tárolása. Függvényei a lista gyors és letisztult használatát teszik lehetővé.

Attribútumok

Privát

- unsigned elements: A lista elemeinek száma.
- Request* pData: A lista első elemére mutató pointer.

Metódusok

Publikus

- fifo(); : Konstruktor, mely az elemszámot 0-ba, a pData-t nullptr-be állítja.
- void loadFile(const char[] = "lista_be.txt"); : A fájl beolvasásáért felelős függvény. Paraméterként megadható a bemeneti fájl neve, de ez nem kötelező, ugyanis van alapértelmezett értéke. A függvény meghívásakor először standard outputon jelzi a felhasználó számára, hogy megkezdődik a fájl feldolgozása. A függvény fontos része a hibakezelés. Lehetséges (és lekezelt) hiba a fájl sikertelen megnyitása (pl. nem létezik a megadott nevű szöveges fájl) (ERROR 5), illetve a fájl hibás formátuma (számokon kívül más karakterek – ERROR 6). A kérések felé támasztott egyéb feltételeket a Request konstruktor vizsgálja (ERROR 1-3). A függvény soronként olvassa be a kéréseket, ezeket a fifo osztály push függvényének segítségével rakja a listába.
- bool isEmpty(); : Hibakezeléseknél illetve más függvények esetszétválasztásánál használt függvény, értéke 'true', ha a lista üres, különben 'false'.
- bool isNew(const Request&); : Megvizsgálja, hogy a paraméterként kapott kérés szerepel-e már a listában. Értéke 'true' ha a kérés új, különben 'false'.
- unsigned getElements(); Request getPdata(unsigned); : A privát tagváltozók eléréséhez szükséges getterek.
- bool push(const Request&); : A paraméterként kapott Requestet a lista végére fűzi. Visszatérési értéke 'true' ha az új listaelemet sikeresen beillesztette, 'false', ha a paraméterként kapott elem már szerepel a listában. Hibakóddal lép ki, ha nem sikerült a helyfoglalás (ERROR 7).
- void print(); : Eredetileg lett volna egy olyan funkció, amellyel megszakítható a lift működése, ekkor a vod print() függvény elmentette volna fájlba, illetve standard outputra is kiírta volna a megmaradó listaelemeket. Végül azért nem került törlésre a kódból, mert debuggolásnál nagyon hasznos.
- void takeOut(const Request&); : Megkeresi és kiveszi a listából a paraméterként kapott Requestet. Hibakezelésnél csak azt kell megvizsgálni, hogy a lista üres-e, mert a takeOut függvény csak olyan esetben van meghívva, amikor biztosan a listában van a megadott Request.
- Request& front(); : Visszaadja a lista első elemét, de nem törli.
- void pop(); : Törli a lista első elemét. Üres lista esetén 4.1- es hibát ír.
- ~fifo(); : A fifo destruktora. Nem volt megfelelő a default destruktor, mivel dinamikus foglalás történt.

Lift

Felelőssége

A feladat érdemi részét valósítja meg a másik két osztály összehangolt felhasználásával.

Attribútumok

Privát

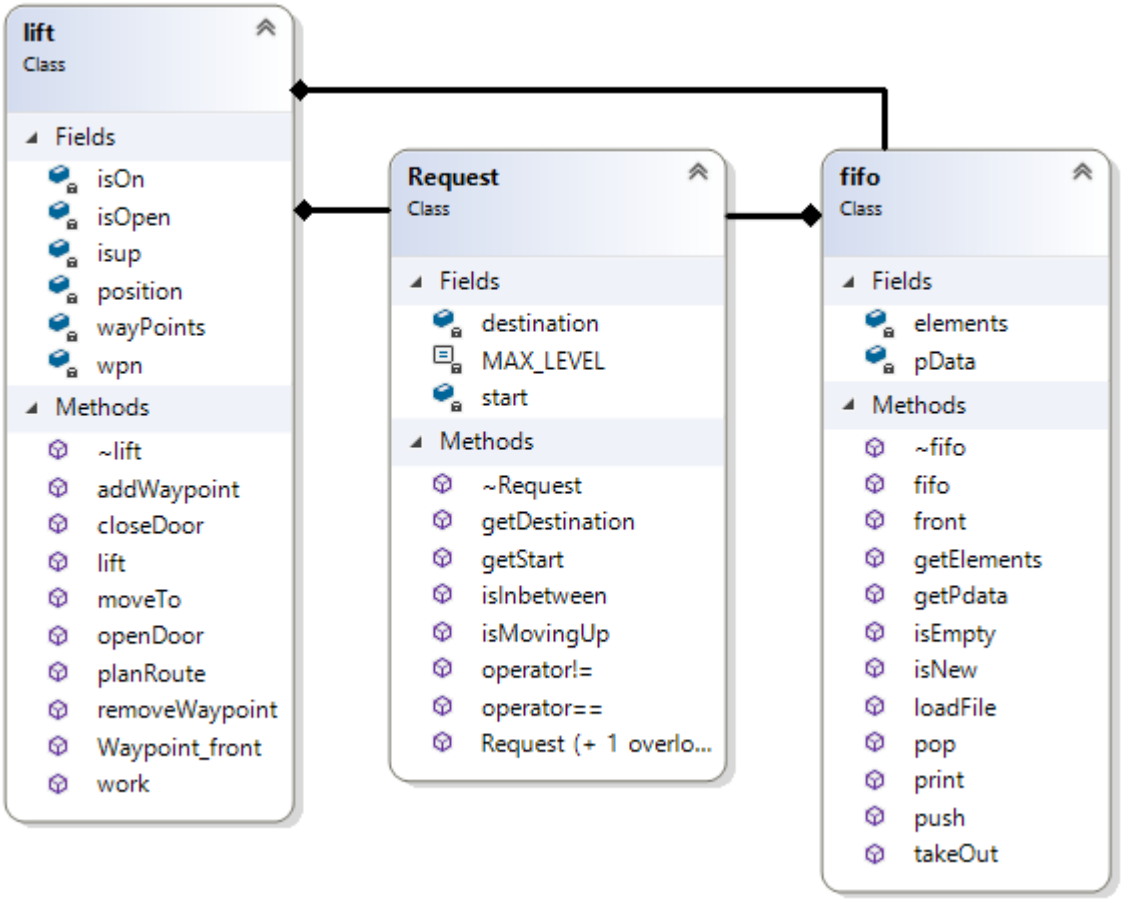
- bool isOpen; : A liftajtó állapota – 'true' ha az ajtó ki van nyitva, 'false' ha az ajtó be van csukva.
- bool isup; : A Request osztály isMovingUp () függvényének gyorsabb használata érdekében lett bevezetve. Megmondja, hogy a liftnek fel- vagy lefelé kell mozognia a kérés teljesítéséhez.
- unsigned position; : A lift aktuális pozícióját tárolja. Az egyszerűség kedvéért csak a fontosabb időpillanatokban frissül, tehát mozgás közben nem, csak amikor a lift megáll.
- unsigned wpn; : (WayPointsNumber) a wayPoints tömb elemszámát tárolja.
- unsigned* wayPoints; : Sorba rendezve tárolja a azoknak az emeleteknek a számát, amelyeken a liftnek meg kell állnia. Ha a lift eléri a tömb soron következő elemét, az törlődik a listából, majd a lift elindul a következő felé. Ezt addig csinálja, amíg a tömb ki nem ürül, majd ezután a planRoute() függvénnyel újra feltölti a tömböt, amennyiben a fifo nem üres.

Metódusok

Publikus

- lift(); : Konstruktor, mely minden tagváltozót 0-ba (illetve nullptr-be) állít.
- void openDoor(std::ofstream&); : Az ajtót nyitja, azaz 'true' –ba állítja az isOpen tagváltozót. Erről tájékoztatja a felhasználót.
- void closeDoor(std::ofstream&); : Az openDoor-hoz hasonló függvény, amely az ajtót csukja, és ezt tudatja a felhasználóval.
- void addWaypoint(const Request&); : A kapott Request két tagváltozóját megvizsgálja, hogy szerepelnek-e már a wayPoints tömbben. Amelyik szint nem szerepel a tömbben, azt hozzáadja, majd növeli a wpn-t.
- bool removeWaypoint(); : Miután a lift elérte a tömb következő emeletét, ez a függvény eltávolítja azt a tömbből. Visszatérési értéke 'false' ha a tömb üres, 'true' ha sikeresen eltávolította a kívánt elemet.
- unsigned& Waypoint_front(); : Visszaadja a tömb első elemét.
- void moveTo(unsigned, std::ofstream&); : A liftet mozgatja, azaz a pozícióját a paraméterként kapott emeletre állítja. Erről tájékoztatja a felhasználót a standard outputon illetve a paraméterként kapott fájlon keresztül is.
- bool planRoute(fifo&); : Megtervezi a lift következő útját a fifo soron következő elemének felhasználásával. Először a fifo következő elemét adja át az addWaypoint függvénynek, majd végigmegy a tömbön, és az isInbetween függvény használatával megvizsgálja az összes elemet. Amennyiben van olyan listaelem, amely az első elemmel egy időben teljesíthető, azt szintén átadja az addWaypoint függvénynek. A fifo első elemét a pop, a többi felhasznált elemet a takeOut függvénnyel távolítja el a fifo-ból.
- void work(fifo&); : Azért, hogy a main függvény a lehető legletisztultabb legyen, a függvények összehangolását a work függvény végzi el. Ha üres listát kap, hibaüzenettel jelzi. A lista ellenőrzése után megnyit egy szöveges fájlt, amibe írni fog. A planRoute függvény visszatérési értékét és a wpn-t feltételként felhasználva egymásba ágyazott while - ciklusokkal addig mozgatja a liftet, ameddig a fifo ki nem ürül.
- ~lift(); : Dinamikus memóriafoglalás történt, ezért a destruktorban felszabadítás történik.

UML osztálydiagramm



Összegzés

Mit sikerült és mit nem sikerült megvalósítani a specifikációból?

Sajnos a program debuggolása egy komolyabb hiba miatt elhúzódott, így a lift be- és kikapcsolásához megírt függvényeket nem tudtam beépíteni a kódba, ezért inkább töröltem. A specifikáció többi részét sikerült megvalósítani.

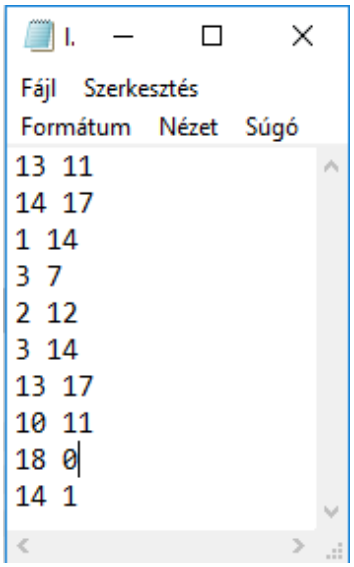
Mit tanultál a megvalósítás során?

A feladat jól begyakoroltatta a lista kezeléséhez szükséges különböző függvények megvalósítását, illetve használatát. Bár öröklés nincs benne, a többi objektumorientált alapelv sok helyen látszódik a programban, a szemléletmódot segített elsajátítani.

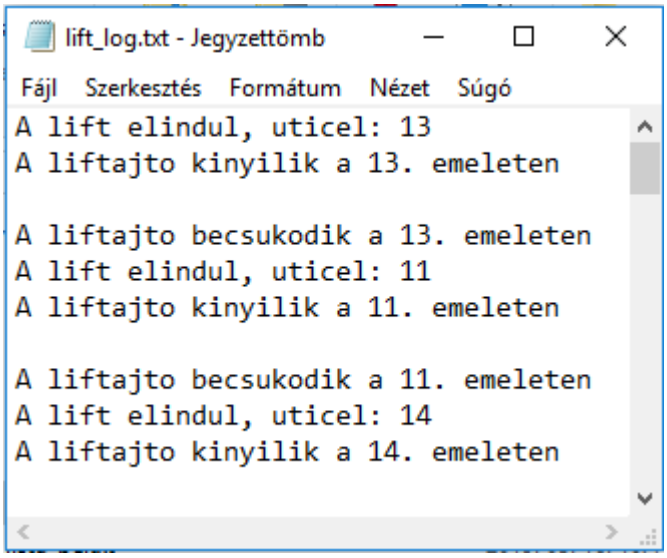
Továbbfejlesztési lehetőségek

A kitörölt kikapcsol és bekapcsol függvény hasznos lenne, különösen akkor, ha lehetne időzíteni is a lift mozgását és valós időben interakcióba lépni a lifttel. Például időközben megadni új kéréseket, vagy kikapcsolni a liftet. Másik érdekes fejlesztési lehetőség lehet több lift összehangolása.

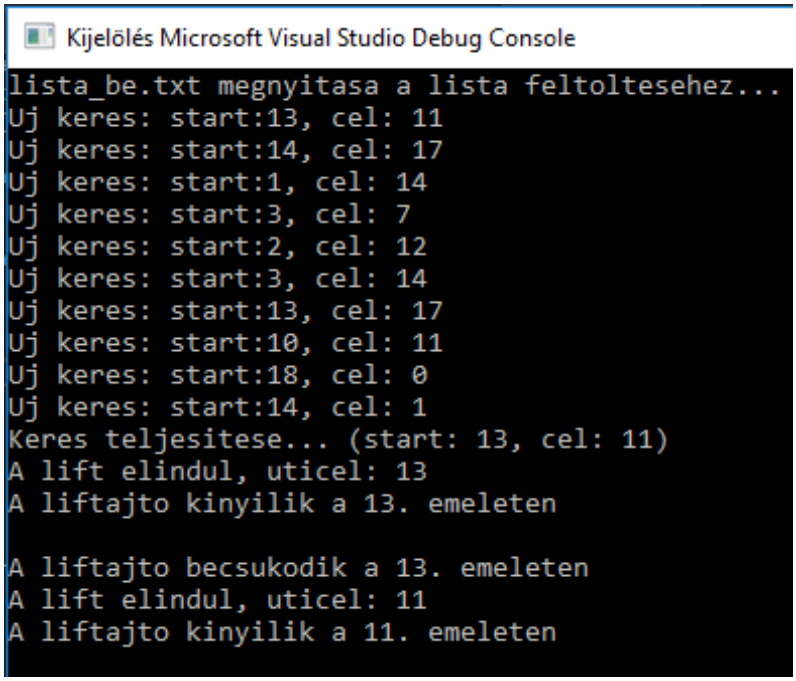
Képernyőképek a futó alkalmazásról



1. ábra: Példa hibátlan bemenetre



2. ábra: Kimeneti fájl (részlet)



3. ábra: Kimenet (exception nélkül, részlet)