Tartalomjegyzék

Fejlesztői dokumentáció 2
Feladat leírása 2
Rendszerterv 2
A meres.txt állomány szerkezete 2
Képernyőterv 3
Változótábla 4
Felhasznált struktúratípus 6
Felhasznált osztálydefiníció 6
A felhasznált függvények 6
Felhasználói dokumentáció 11
Hardver és szoftver követelmények 11
Hardverigény 11
Szoftverigény 12
Program kezelése 12
A program telepítése 12
A program indítása 12
A program bemenete 12
A program eredménye 12
A program leállítása 12
Hibalehetőségek 12
A program egy szabályos lefutása utáni képernyőkép 13
Tesztelés 13
Továbbfejlesztési lehetőségek 15
Irodalomjegyzék 15
Forráskód 16

Fejlesztői dokumentáció

Feladat leírása

Két várost (Egyik, Másik) összekötő út 1000 m hosszú részének felújításán dolgoznak. Ennek a szakasznak a forgalmát figyelték egy nap néhány óráján keresztül, aminek eredményét egy szöveges állományban rögzítették. Az említett szakaszon előzési tilalom van érvényben ezért, ha az útszakaszon egyik jármű utoléri a másikat, akkor változatlan sorrendben, ugyanabban az időpillanatban hagyják el a szakasz, mint ahogy a lassabb jármű tenné. A program célja a meres.txt szöveges fájl forgalmi adatainak beolvasása, a megadott kérdésekre adott válaszok képernyőre valamint fájlba történő kiíratása.

A kérdések alapján az alábbi feladatokat valósítja meg a program:

1. Beolvassa a meres.txt állományban talált adatokat.
2. A felhasználótól bekért m érték alapján kiírja a képernyőre, hogy az m-edikként belépő jármű melyik város felé haladt.
3. Kiírja a képernyőre, hogy a Másik város irányába tartó utolsó két jármű hány másodperc különbséggel érte el az útszakasz kezdetét.
4. Meghatározza óránként és irányonként, hogy hány jármű érte el a szakaszt és soronként kiírja egy-egy óra adatait a képernyőre. Az első érték az órát, a második érték az Egyik, a harmadik a Másik város felől érkező járművek számát jelenti. A kiírásban csak azok az órák jelennek meg, amelyekben volt forgalom valamely irányban.
5. A belépéskor mért értékek alapján meghatározza a 15 leggyorsabb járművet. Kiírja a képernyőre ezek belépési idejét, a várost (Egyik, illetve Másik), amely felől érkezett, és m/s egységben kifejezett sebességét két tizedes pontossággal, sebességük szerinti növekvő sorrendben. Soronként egy jármű adatai vannak megjelenítve szóközzel tagolva. (A feladat megoldásakor figyelni kellett arra, hogy a következő feladatban az adatok eredeti sorrendjét még fel kell használni.)
6. Kiírja az egyik.txt állományba azokat az időpontokat, amikor az Egyik város felé tartók elhagyták a kérdéses útszakaszt úgy, hogy ha egy jármű utolér egy másikat, akkor a kilépésük időpontja a lassabb kilépési ideje lesz. A fájl minden sorába egy-egy időpont kerül óra perc másodperc formában pontosan egy szóközzel elválasztva.

Rendszerterv

A meres.txt állomány szerkezete

Első sorában a megfigyelési időszakban áthaladó járművek száma (legfeljebb 2000) látható, továbbiakban pedig soronként egy áthaladó jármű adatai olvashatók időrendben. Egy sorban az

első három szám azt az időpontot jelöli (óra, perc, másodperc), amikor a jármű belép a vizsgált útszakaszra. A következő szám jelöli, hogy a jármű az érintett távolságot hány másodperc alatt tenné meg (legfeljebb 300) – a belépéskor mért sebességgel –, ha haladását semmi nem akadályozná. Ezt egy betű követi, amely jelzi, hogy a jármű melyik város irányából érkezett. Ennek megfelelően a betű A (Egyik) vagy F (Másik) lehet. Az egyes adatokat pontosan egy szóköz választja el egymástól

A meres.txt állomány egy részlete:

1105

7 21 1 60 F

7 21 58 69 F

7 22 4 117 F

7 22 39 155 A

7 23 11 99 A

…

A 3. sor (példában aláhúzva) megmutatja, hogy a 7 óra 21 perc 58 másodperckor a Másik (F) város felől érkező jármű 69 másodperc alatt tenné meg ezt az 1 km hosszú távolságot. Ez a jármű – ha más járművek nem akadályozzák – 7 óra 23 perc 7 másodperckor lép ki az útszakaszról, tehát akkor már nem tartózkodik ott.

Képernyőterv

A feladat megfogalmazása nem ír elő semmilyen grafikus megjelenítést, ezért csak a parancssoros ablakban jeleníti meg a feladatokat egyszerű szövegként és billentyűzetről olvassa be a felhasználótól bekért adatot.

A képernyőkép:

1. feladat

Az állományban tárolt adatok beolvasása megtörtént.

2. feladat Adja meg a jármű sorszámát (1-1105)! ➔ Billentyűzetről várja a sorszám értékét

A megadott sorszámú jármű az Egyik / Másik város felé haladt.

3. feladat

A Másik város irányába tartó utolsó két jármű időkülönbsége másodpercben: xyz

4. feladat

Óránkénti és irányonkénti statisztika:

x óra y z

y óra z x

…

z óra x y

5. feladat

A 15 leggyorsabb jármű adata:

x y z A xy.xy m/s

x y z F xy.xy m/s

...

x y z A xy.xy m/s

6. feladat

A Fájlba írás megtörtént.

Változótábla

név	**típus**	**hatókör**	**leírás**
fajlnev	string	main()	beolvasandó fájl neve
F	Felujitas osztály	main()	a konstruktor meghívására szolgáló objektum
jszam	int	Felujitas osztály	áthaladó járművek száma
jadat[]	forgalom struktúra	Felujitas osztály	áthaladó járművek adatait tároló struktúratömb
be	ifstream osztály	Felujitas(string fnev)	a meres.txt fájl tartalmának beolvasására szolgáló objektum
i	int	Felujitas(string fnev)	ciklusváltozó a struktúratömb elemeinek indexeléséhez
m	int	Kiir()	a jármű sorszáma
utolso	int	Kulonbseg()	Az A városból érkező utolsó jármű belépési ideje másodpercben
utolsoe	int	Kulonbseg()	Az A városból érkező utolsó előtti jármű belépési ideje másodpercben
db	int	Kulonbseg()	számláló az utolsó két jármű számlálásához
i	int	Kulonbseg()	ciklusváltozó a struktúratömb elemeinek indexeléséhez

dbAbol[]	int	OrankentiForgalom()	Az A városból érkező járművek óránkénti belépési idejének száma
dbFbol[]	int	OrankentiForgalom()	Az F városból érkező járművek óránkénti belépési idejének száma
i	int	OrankentiForgalom()	ciklusváltozó a struktúratömb és a dbAbol[], dbBbol[] tömbök elemeinek indexeléséhez
jadat2[]	forgalom struktúra	Leggyorsabb15()	áthaladó járművek adatainak másolatát tároló struktúratömb
seged	forgalom struktúratípus	Leggyorsabb15()	a struktúratömb elemeinek cseréjéhez szükséges segédváltozó
i	int	Leggyorsabb15()	ciklusváltozó, amely tárolja meddig rendezett a segéd (jadat2[]) struktúratömb
j	int	Leggyorsabb15()	ciklusváltozó a segéd (jadat2[]) struktúratömb elemeinek indexeléséhez
ki	ofstream osztály	KilepesiIdokAba()	az egyik.txt fájlba való írásra szolgáló objektum
elozoido	int	KilepesiIdokAba()	Az F városból érkező aktuálisan vizsgált jármű előtti jármű kilépési ideje másodpercben
aktualisido	int	KilepesiIdokAba()	Az F városból érkező aktuálisan vizsgált jármű kilépési ideje másodpercben
i	int	KilepesiIdokAba()	ciklusváltozó a jadat[] struktúratömb elemeinek indexeléséhez

Felhasznált struktúratípus

A meres.txt állomány szerkezete alapján logikus volt valamilyen struktúra típus deklarálása. Forgalom néven létrehoztam egy struktúratípust 5 struktúraelemmel:

- ora, perc, mperc: int típusúak, a belépési időt tárolják óra, perc, másodperc bontásban.
- seb: int típusú, az áthaladási idő másodpercben való tárolására szolgál.
- varosbol: char típusú, tárolja, hogy a jármű melyik városból érkezett.

A meres.txt állományban található adatokat egy ilyen típusú struktúratömbbe olvastam be. Azért neveztem el az áthaladási időt seb-nek, mert rögzített útszakasz esetén, mint most, minél kisebb ez az érték, annál nagyobb az áthaladó jármű sebessége. Ezt az 5. feladatban ki fogom használni.

Felhasznált osztálydefiníció

Adattagok

Felujitas néven létrehoztam egy osztályt 3 privát adattaggal:

- jszam: int típusú, az áthaladó járművek számát tárolja, amit a meres.txt első sorából olvasok be.
- jadat: forgalom struktúratípusra mutató pointer, ami a dinamikus tömbhöz kell.

Tagfüggvények

Minden részfeladathoz létrehoztam egy publikus tagfüggvényt, amelyeknek definícióját az osztályon kívül adtam meg:

- Felujitas(string fnev);
- void Kiir();
- int Kulonbseg();
- void OrankentiForgalom();
- void Leggyorsabb15();
- void KilepesiIdokAba();
- ~Felujitas()

A felhasznált függvények

int main()

Ez a függvény írja ki a képernyőre az egyes részfeladatok sorszámát és ez a függvény hívja meg az egyes feladatokat megvalósító függvényeket is. Itt van megadva a feldolgozni kívánt forgalomi adatokat tartalmazó meres.txt fájl.

Felujitas(string fnev)

Ez a konstruktor, ami inicializálja a Felujitas osztály adattagjait. Lefoglalja a jadat[] struktúratömb számára a memóriaterületet. Beolvassa a meres.txt fájl első sorából a járművek számát a jszam változóba, majd a második sortól kezdődően beolvassa soronként az állomány

tartalmát a jadat[] struktúratömbbe. A végén az állományt lezárja. Figyeli a fájlművelet sikerességét, hiba esetén értesít és kilép.

void Kiir()

A felhasználótól bekéri a jármű sorszámát és kiírja a képernyőre, hogy sorszámadik jármű melyik város felé haladt. A megadott sorszámot az m változóban tároltam el, ami a jadat[] tömb indexe egyben. Arra kellett figyelni, hogy a tömbök indexelése 0-tól indul, ezért m-1 lesz a kiírandó elem indexe. Ha a struktúratömb m-1-edik varosbol adattagja megegyezik A-val, akkor kiírja, hogy " A megadott sorszámú jármű a Másik város felé haladt." ellenkező esetben, hogy "A megadott sorszámú jármű az Egyik város felé haladt.". Az algoritmust kiegészítettem ellenőrzött adatbekéréssel.

Be: jadat[1..2000], jszam

Algoritmus:

Ciklus

Ki: ’Adja meg a jármű sorszámát 1 és jszam között!’

Be: m //m>0 és m<jszam, egész

amíg m<1 vagy m>jszam

Ciklus vége.

Ha (jadat[m-1].varosbol = ’A’) akkor Ki: ’ A megadott sorszámú jármű a Másik város felé haladt.’

Különben Ki: ’A megadott sorszámú jármű az Egyik város felé haladt.’

Elágazás vége.

Algoritmus vége.

int Kulonbseg()

Kiírja a képernyőre, hogy a Másik város irányába tartó utolsó két jármű hány másodperc különbséggel érte el az útszakasz kezdetét. Mivel ez egy egész szám, így a függvény visszatérési értéke int.

Az utolsó jármű belépési idejének tárolására az utolso, az utolsó előtti jármű belépési idejének tárolására az utolsoe változó szolgál. Létrehoztam egy számlálót db néven, amivel az A városból érkező járműveket számoltam és az értékét 2-re állítottam, mert az utolsó 2 járművet kell figyelembe venni. Mivel az utolsó két jármű belépési idejének meghatározása volt a feladat, ezért a ciklust az utolsó (jszam-1) tömbindexről indítottam. A ciklus addig fut, amíg az index és a db számláló is nagyobb, mint 0. Ha A várost talált és, ha a db változó értéke 2, akkor az utolso változó értéke egyenlő lesz a belépési idő másodpercben megadott idejével, majd csökkentem a db változó és i index értékét 1-gyel. Ha újra A várost talált és a db nem egyenlő

2-vel, akkor az utolsoe változó értéke egyenlő lesz a belépési idő másodpercben megadott idejével és újra csökkentem a db és i változók értékét 1-gyel. Ha a db változó értéke eléri a 0-át, akkor a ciklus befejeződik és visszaadja az utolsó illetve utolsó előtti jármű belépési idejének különbségét. Feltételeztem, hogy van legalább két A városból érkező jármű.

Be: jadat[1..2000], jszam

Algoritmus:

db := 2

i := jszam - 1

Ciklus amíg i > 0 és db > 0

Ha (jadat[i].varosbol = 'A') akkor

Ha (db = 2) akkor utolso = jadat[i].ora * 3600 + jadat[i].perc * 60 + jadat[i].mperc

Különben utolsoe = jadat[i].ora * 3600 + jadat[i].perc * 60 + jadat[i].mperc

Elágazás vége.

db := db – 1

Elágazás vége.

i := i – 1

Ciklus vége.

Ki: utolso - utolsoe

Algoritmus vége.

void OrankentiForgalom()

Meghatározza óránként és irányonként, hogy hány jármű érte el a szakaszt és soronként kiírja egy-egy óra adatait a képernyőre, de csak azokét, amelyekben volt forgalom valamely irányban. Ehhez indexelt tömböket és a kiválasztás tételével kombinált megszámlálás tételét használtam. Létrehoztam két 24 elemű tömböt az A és F városból érkező járművek belépési idejének óra részének számolására. A tömbök elemeinek 0 kezdőértéket adtam. Az indexek jelölik az órát (0-23), az indexhez tartozó értékek pedig az adott városból érkező járművek óránkénti számát. A jadat[] struktúratömbön for ciklussal végig megyek, és ha A várost találok, akkor a jadat[] tömb ora adattagjának értékét, mint indexet felhasználva megnövelem az A városhoz tartozó dbAbol[] tömb megfelelő indexű elemének értékét 1-gyel. B városra ugyanígy. A végén pedig kiíratom a dbAbol[] és dbFbol[] azon indexű elemeket, amelyeknek értéke nagyobb nullánál, az indexekkel együtt.

Be: jadat[1..2000], jszam

Algoritmus:

dbAbol[1..24] := 0 : dbFbol[1..24] := 0

Ciklus i := 1-től jszam-ig

Ha (jadat[i].varosbol = 'A') akkor dbAbol[jadat[i].ora] := dbAbol[jadat[i].ora] + 1

Különben

dbFbol[jadat[i].ora] := dbFbol[jadat[i].ora] +1

Elágazás vége.

Ciklus vége.

Ciklus i := 1-től 24-ig

Ha (dbAbol[i] <> 0 vagy dbFbol[i] <> 0) akkor Ki: i, dbAbol[i], dbFbol[i]

Elágazás vége.

Ciklus vége.

Algoritmus vége.

void Leggyorsabb15()

A belépéskor mért értékek alapján meghatározza a 15 leggyorsabb járművet. Kiírja a képernyőre ezek belépési idejét, a várost, amely felől érkezett, és m/s egységben kifejezett sebességét két tizedes pontossággal, sebességük szerinti növekvő sorrendben. A feladat megoldásakor figyelni kellett arra, hogy a következő feladatban az adatok eredeti sorrendjét még fel kell használni. A program első változatában a konstruktorban hoztam létre a dinamikus jadat2[] segéd struktúratömböt, amihez a Felujitas osztályban deklaráltam egy forgalom struktúratípusra mutató pointert. A meres.txt fájl adatainak a jadat[] tömbbe való beolvasásával egy időben a jadat[] tömb elemeit átmásoltam a jadat2[] tömbbe. Így futási időt takaríthattam meg, mivel egy ciklussal fel lett töltve a két tömb. Itt a függvényben már ezzel nem kellett foglalkozni. Azonban a jadat2[] segédtömb egész addig memóriaterületet foglat, amíg azt a destruktor fel nem szabadította. Ekkor gondolkodtam el azon, hogy gyengébb, kevesebb memóriával rendelkező számítógép esetén kevésbé zavaró, ha két ciklus miatt lassabban fut le a program, mintha az elfogyó szabad memória miatt hibával leállna. Ezért úgy módosítottam a programot, hogy ebben a függvényben dinamikusan foglaltam le a helyet a jadat2[] tömbnek. Így megoldható volt, hogy csak az aktuális járművek számával megegyező méretű tömbnek foglaljak helyet a memóriában, a megadott 2000 helyett. Ezzel memóriát takaríthattam meg. Ezután a jadat2[] tömbbe bemásoltam a jadat[] tömb tartalmát, majd buborékos rendezés módszerével rendeztem a seb adattag értéke alapján növekvően. Ez sebesség szerinti csökkenő rendezést jelent, amit egy for ciklussal járok be a 15. (14. indexű) elemtől az elsőig (0. indexű). Kiírattam a járművek belépési idejét (óra, perc, másodperc), a várost, amely felől érkeznek és

a sebességet két tizedes pontossággal. A végén a delete operátorral felszabadítottam a lefoglalt memóriablokkot.

Be: jadat2[1..jszam] seb szerint rendezetlen, jszam

Algoritmus:

```
    Ciklus 1 := 1-től jszam-ig
        jadat2[i] := jadat[i]
    Ciklus vége.
    Ciklus i  := 2 től jszam-ig
        Ciklus j := jszam-1-től i - ig  –1–esével
                Ha( jadat2[j-1].seb > jadat2[j].seb) akkor
                                seged := jadat2[j-1]
                                jadat2[j-1] := jadat2[j]
                                jadat2[j] := seged
                Elágazás vége.
        Ciklus vége.
    Ciklus vége.
    Ciklus i := 15-től 1-ig -1-essével
          Ki: jadat2[i].ora, jadat2[i].perc, jadat2[i].mperc, jadat2[i].varosbol,
              1000 / jadat2[i].seb ’m/s’
    Ciklus vége.
```

Algoritmus vége.

void KilepesiIdokAba()

Kiírja az egyik.txt állományba az Egyik (F) város felé tartók kilépési idejét úgy, hogy ha egy jármű utolér egy másikat, akkor a kilépésük időpontja a lassabb kilépési ideje lesz. A fájl minden sorába egy-egy időpont kerül óra, perc, másodperc formában pontosan egy szóközzel elválasztva. Ha létezik, akkor felülírja, ha nem létezik, akkor létrehozza az egyik.txt fájlt. Figyeli a fájlművelet sikerességét, hiba esetén értesít és kilép. A maximum kiválasztás tételét felhasználva megyek végig a jadat[] tömb elemein. Ha F várost talál, akkor megvizsgálja, hogy az aktuális belépési idő és az áthaladási idő összegének másodpercben mért értéke nagyobb, mint az előző idő. Akkor ennek értékét eltárolom az aktualisido változóban, ezt kiíratom, majd az elozoido változónak értékül adom az aktualisido változó értékét, ugyanis tárolni kell az előző időt, hogy össze lehessen hasonlítani, hogy utolért-e egy jármű egy másikat. Ha az aktuális kilépési idő (belépési idő és az áthaladási idő összegének másodpercben mért értéke) nem nagyobb az előző időnél, akkor az előző időt íratom ki. A végén az állományt lezárom.

```
Be: jadat[1..2000] belépési idő szerint rendezett, jszam
Algoritmus:
        elozoido := 0
        Ciklus i = 1-től jszam-ig
            Ha (jadat[i].varosbol = 'F') akkor
                Ha (jadat[i].ora * 3600 + jadat[i].perc * 60 + jadat[i].mperc +
                    jadat[i].seb > elozoido)
                    akkor aktualisido = jadat[i].ora * 3600 + jadat[i].perc * 60 +
                                        jadat[i].mperc + jadat[i].seb
                        Ki(fájlba): aktualisido / 3600, (aktualisido % 3600) / 60,
                                    aktualisido % 60
                        elozoido = aktualisido
                    Különben
                        Ki(fájlba): elozoido / 3600, (elozoido % 3600) / 60,
                                    elozoido % 60
                Elágazás vége.
            Elágazás vége.
        Ciklus vége.
Algoritmus vége.
```

Felhasználói dokumentáció

Hardver és szoftver követelmények

Hardverigény

Bármilyen Windows 10 operációs rendszer futtatására alkalmas IBM kompatibilis személyi számítógép (PC).

Minimális hardverigény:

- 1 GHz-es vagy gyorsabb processzor,
- 1 GB memória 32 bites vagy 2 GB memória 64 bites rendszerhez,
- 1 MB merevlemez-terület,
- 800x600 pixel felbontású kijelző.

Ajánlott hardverigény:

- 1 GHz-es vagy gyorsabb, legalább 2 magos, kompatibilis 64 bites processzor,
- 4 GB memória,
- 1 MB merevlemez-terület,

- nagy felbontású (720p) kijelző.

Szoftverigény

Windows 10 (32/64 bites) 1809-es vagy újabb verzió: Home, Professional, Education, Enterprise változat.

Program kezelése

A program telepítése

A program nem igényel telepítést. A Felujitas mappát a C:\ meghajtóra kell másolni.

A program indítása

A Felujitas mappában található a Felujitas.exe futtatható állomány. A Fájlkezelőben meg kell nyitni a Felujitas mappát és az egér bal gombjával a Felujitas.exe fájlra való dupla kattintással indítható a program.

A program bemenete

A program indítás után beolvassa a Felujitas mappában található meres.txt fájl tartalmát, majd bekéri billentyűzetről egy jármű sorszámát és automatikusan lefut. Lényeges, hogy a forgalmi adatokat tartalmazó fájlt a Felujitas mappába kell másolni meres.txt néven. Fontos, hogy a meres.txt fájl tartalma a következőképpen nézzen ki:

1105

7 21 1 60 F

7 21 58 69 F

…

Első sorában az áthaladó járművek száma (legfeljebb 2000) látható. A továbbiakban pedig soronként egy áthaladó jármű adatai olvashatók időrendben. Egy sorban az első három egész szám a belépési időpontot jelöli óra, perc, másodperc bontásban. A következő egész szám jelöli az áthaladási időt másodpercben. Ezt egy betű követi, amely jelzi, hogy a jármű melyik város irányából érkezett. Ennek megfelelően a betű A (Egyik) vagy F (Másik) lehet. Az egyes adatokat pontosan egy szóköz választja el egymástól.

A program eredménye

A program a parancssoros ablakban megjeleníti a feladatokat és eredményeket, majd a kilépési időket az egyik.txt fájlba írja.

A program leállítása

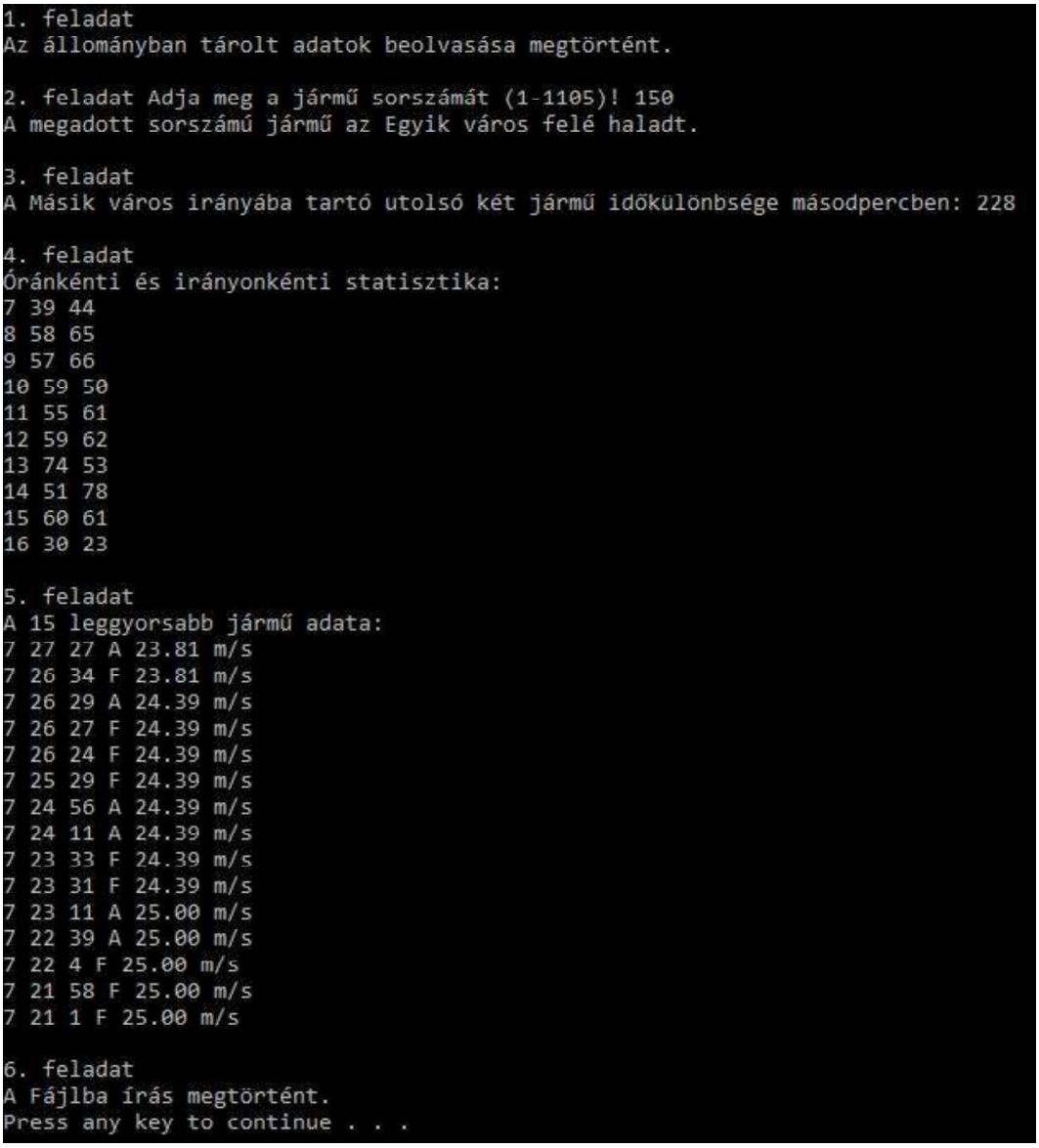
Egy tetszőleges billentyűzet lenyomására vagy a parancssori ablak bezárás gombjára az egér bal gombjával kattintva a program leáll.

Hibalehetőségek

- A meres.txt fájl nem a Felujitas mappába lett másolva vagy hiányzik.

- A forgalmi adatokat tartalmazó fájl nem meres.txt néven lett elnevezve.
- Több mint 2000 sort tartalmaz a meres.txt állomány.
- A billentyűzetről bekért jármű sorszámának a 2. feladatban megjelenített tartományon belül kell lennie. A tartományba az alsó és felső határ is beletartozik. Ha a felhasználó tartományon kívüli sorszámot ír be, akkor újra bekéri az adatot, egész addig, amíg megfelelőt meg nem ad.

A program egy szabályos lefutása utáni képernyőkép



Tesztelés

A fejlesztés során folyamatosan teszteltem a feladatokat megvalósító tagfüggvényeket. A programot a különböző függvények egyenkénti megírásával készítettem. Elsőnek megadtam a szükséges fejlécállományokat és beállítottam a standard névtere. Megadtam a struktúrát és az

osztály definícióját az adattagokkal és a konstruktorral, valamint a destruktorral együtt. A main() függvényben példányosítottam az osztály objektumot és kiírattam az inicializált struktúratömböt. Majd megírtam a Kiir() függvényt és futtatam a programot és így tovább.

A program elkészítése után az alábbi speciális eseteket vizsgáltam:

- Hiányzó, elgépelt vagy más elnevezésű állomány esetén „Hibás fájl megnyitás” hibaüzenetet ad az elvárásoknak megfelelően.
- Egynél kisebb vagy 1105-nél (max 2000) nagyobb sorszám megadása esetén az ellenőrzött adatbevitel megvalósítása miatt újra bekéri a sorszámot egész addig, amíg tartományon belüli értéket nem adunk meg.
- Ha még nem létezett az egyik.txt fájl, akkor létrehozza, ha már létezett, akkor felülírja a régebbit az elvárásoknak megfelelően.
- Ha nem tudja létrehozni az egyik.txt fájlt, akkor „Hibás fájl megnyitás” hibaüzenetet ad az elvárásoknak megfelelően.
- Ha 2000-nél több sort tartalmaz a fájl, akkor az elvárásoknak megfelelően hibával leáll.
- Ha az első sor nem tartalmazza a járművek számát, akkor a második sor 1. elemét olvassa be a jszam változóba. Hibával leáll a program.
- Ha kevesebb sort tartalmaz a bemeneti állomány, mint az 1. sorban megadott járművek száma, akkor hibaüzenet nélkül leáll a program.
- Ha több sort tartalmaz a bemeneti állomány, mint az 1. sorban megadott járművek száma, akkor az elvártnak megfelelően hiba nélkül lefut a program.
- Ha az áthaladási idő helyére, aminek egész számnak kellene lennie betű került, akkor hibaüzenet nélkül leáll a program.
- Ha a várost jelölő betű helyére egyjegyű szám vagy A-tól és F-től különböző karakter kerül, akkor hiba nélkül lefut a program. A 4. feladatban elkészült irányonkénti statisztika eredményét módosítja, attól függően, hogy melyik várost jelölő betű lett elírva. Ha negatív előjelű egyjegyű vagy többjegyű szám kerül a várost jelölő betű helyére, akkor hibaüzenet nélkül leáll a program. Ha a várost jelölő betű a legnagyobb sebességű járműveknél lett elírva A-tól és F-től különböző karakterre, akkor a program az elvártnak megfelelően hiba nélkül lefut. Az 5. feladatban kiírja a hibás város betűjelét.
- Ha az elválasztó szóköz a várost jelölő betű kivételével bárhol kimarad, akkor az elvártnak megfelelően hibajelzés nélkül leáll a program. A betű előtti szóközhiány nem okoz futási hibát.

- A szóköztöbblet az elvártaknak megfelelően nem okoz hibát, a program lefut.
- Ha nem egész számot adunk meg az idő, a perc, a másodperc és az áthaladási idő értékének, akkor az elvártnak megfelelően a program hibajelzés nélkül leáll.
- Ha a 2. feladatban sorszámnak a járművek számánál nagyobb vagy negatív értéket adunk meg, akkor véletlenszerűen vagy hibaüzenet nélküli hibával áll le a program, vagy hibátlannak tűnő módon fut le. Ez az elvártaknak megfelelő működést jelent, mivel a C++ engedi a túlindexelést.
- Ha a meres.txt alapján nem haladt legalább két jármű A-ból az F város felé, akkor a 3. feladat megvalósítása miatt az elvártnak megfelelően hibával leáll a program.
- 0-nál kisebb és 23-nál nagyobb óra értékek esetén a program az elvártnak megfelelően hiba nélkül fut le, mert a C++ engedi a túlindexelést. A 4. feladat algoritmusa csak 0-23 tartományba eső indexű elemek értékét írja ki.

Továbbfejlesztési lehetőségek

- A felhasználótól kérné be a program a beolvasandó illetve a kiírandó állomány nevét.
- A feladat szövege nem tért ki rá, de a 2. feladat készítése során már megvalósítottam az ellenőrzött adatbekérést, mint továbbfejlesztési lehetőséget. A felhasználótól nem várható el, hogy ismerje a vizsgált útszakaszon áthaladó járművek számát, ezért a járművek maximális számát is megjelenítettem a képernyőn.
- A 3. feladathoz kapcsolódóan vizsgálni lehetne, hogy haladt-e legalább 2 jármű a Másik város irányába.
- Alkalmassá lehetne tenni a programot, hogy ne csak 15 rögzített, hanem tetszőleges számú leggyorsabb jármű adatát írja ki.

Irodalomjegyzék

[1] Programfejlesztés, Bevezetés az objektum-orientált programozásba GAMF, Kecskemét

[2] Programozás II. gyakorlati jegyzet 8. gyakorlat
https://okt.sed.hu/prog2/gyakorlat/gyak8/

[3] Dinamikus memóriakezelés
http://moodle.autolab.uni-pannon.hu/Mecha_tananyag/c++programozas/ch01.html#ch-I.6.3

[4] Lővei Péter: Felhasználói és fejlesztői dokumentáció
http://loveipeter.web.elte.hu/progalap/beadando/pdf/bead.pdf

[5] SZIRÉN integrált könyvtárkezelő rendszer Felhasználói kézikönyv 9.526-es verzió: 2020. július

http://sziren.com/index_sziren.htm - letöltések - Felhasználói kézikönyv 2020

[6] A Windows 10 rendszerkövetelményei

https://support.microsoft.com/hu-hu/windows/a-windows-10-rendszerk%C3%B6vetelm%C3%A9nyei-6d4e9a79-66bf-7950-467c-795cf0386715

[7] A Windows 11 rendszerkövetelményei

https://www.microsoft.com/hu-hu/windows/windows-11-specifications

Forráskód

```
#include <iostream>
#include <fstream>
#include <iomanip>

using namespace std;

struct forgalom
{
	int ora, perc, mperc, seb;
	char varosbol;
};

class Felujitas
{
	private: int jszam;
			forgalom* jadat; //a dinamikus tömbhöz kell
	public: Felujitas(string fnev);
		void Kiir();
		int Kulonbseg();
		void OrankentiForgalom();
		void Leggyorsabb15();
		void KilepesiIdokAba();

	~Felujitas()
	{
		delete[]jadat;
	}
};

int main()
{
	setlocale(LC_ALL, "hun");
	string fajlnev = "meres.txt";
	cout << "1. feladat" << endl;
	Felujitas F(fajlnev);
	cout << endl << "2. feladat";
	F.Kiir();
	cout << endl << "3. feladat"<<endl;
```

```
	cout << "A Másik város irányába tartó utolsó két jármű időkülönbsége másodpercben:
" << F.Kulonbseg() << endl;
	cout << endl << "4. feladat" << endl;
	cout << "Óránkénti és irányonkénti statisztika:" << endl;
	F.OrankentiForgalom();
	cout<< endl << "5. feladat" << endl;
	cout << "A 15 leggyorsabb jármű adata: " << endl;
	F.Leggyorsabb15();
	cout << endl << "6. feladat" << endl;
	F.KilepesiIdokAba();

	system("pause");
	return 0;
}
Felujitas::Felujitas(string fnev)
{
	jadat= new forgalom[2000];
	ifstream be(fnev);
	if (be.fail())
	{
		cout << "Hibás fájl megnyitás";
		system("pause");
		exit(1);
	}
	be >> jszam;
	//cout << "Járművek száma: " << jszam<<endl;	//Megnézzük beolvasta-e a
járművek számát.
	for (int i = 0; i < jszam; i++) {
		be >> jadat[i].ora >> jadat[i].perc >> jadat[i].mperc >> jadat[i].seb >>
jadat[i].varosbol;
		//cout << jadat[i].ora << " " << jadat[i].perc << " " << jadat[i].mperc << " " <<
jadat[i].seb << " " << jadat[i].varosbol<<endl;	//Megnézzük tényleg feltöltöttük-e a
tömböt
	}
	be.close();
	cout << "Az állományban tárolt adatok beolvasása megtörtént." << endl;
}
void Felujitas::Kiir()
{
	int m;
	{
		cout << " Adja meg a jármű sorszámát (1-" << jszam << ")! ";
		cin >> m;
	} while (m<1 || m>jszam);
	if (jadat[m - 1].varosbol == 'A')
		cout << "A megadott sorszámú jármű a Másik város felé haladt." << endl;
	else
		cout << "A megadott sorszámú jármű az Egyik város felé haladt." << endl;
}
int Felujitas::Kulonbseg()
```

```
{
	int utolso, utolsoe, db = 2, i = jszam - 1;
	while (i > 0 && db > 0)
	{
		if (jadat[i].varosbol == 'A')
		{
			if (db == 2)
				utolso = jadat[i].ora * 3600 + jadat[i].perc * 60 + jadat[i].mperc;
			else
				utolsoe = jadat[i].ora * 3600 + jadat[i].perc * 60 +
jadat[i].mperc;
			db--;
		}
		i--;
	}
	return utolso - utolsoe;
}
void Felujitas::OrankentiForgalom()
{
	int dbAbol[24] = { 0 }, dbFbol[24] = { 0 }, i;
	for (i = 0; i < jszam; i++)
	{
		if (jadat[i].varosbol == 'A')
			dbAbol[jadat[i].ora]++;
		else
			dbFbol[jadat[i].ora]++;
	}
	for (i = 0; i < 24; i++)
		if (dbAbol[i] != 0 || dbFbol[i] != 0)
			cout << i << " " << dbAbol[i] << " " << dbFbol[i] << endl;
}
void Felujitas::Leggyorsabb15()
{
	forgalom* jadat2=new forgalom[jszam];
	for (int i = 0; i < jszam; i++) {
		jadat2[i] = jadat[i];
	}
	//Buborékos rendezés
	forgalom seged;
	for (int i = 1; i < jszam; i++)
	{
		for (int j = jszam-1; j >= i; j--)
		{
			if (jadat2[j].seb < jadat2[j - 1].seb)
			{
				seged = jadat2[j - 1];
				jadat2[j - 1] = jadat2[j];
				jadat2[j] = seged;
			}
		}
```

```
	}
	/*cout << "Rendezett tömb:" << endl;		//Megnézzük tényleg rendezett-e a tömb
	for (int i = 0; i < jszam; i++) {
			cout << jadat2[i].ora << " " << jadat2[i].perc << " " << jadat2[i].mperc
<< " " << jadat2[i].seb << " " << jadat2[i].varosbol << endl;
	}*/
	for (int i = 14; i >= 0; i--)
	{
		cout << jadat2[i].ora << " ";
		cout << jadat2[i].perc << " ";
		cout << jadat2[i].mperc << " ";
		cout << jadat2[i].varosbol << " ";
		cout.setf(ios::fixed);
		cout << setprecision(2) << (float)1000 / jadat2[i].seb << " m/s" << endl;
	}
	delete[] jadat2;
}
void Felujitas::KilepesiIdokAba()
{
	ofstream ki("egyik.txt");
	if (ki.fail())
	{
		cout << "Hibás fájl megnyitás";
		system("pause");
		exit(1);
	}
	int elozoido = 0, aktualisido;
	for (int i = 0; i < jszam-1; i++) {
		if (jadat[i].varosbol == 'F') {
			if (jadat[i].ora * 3600 + jadat[i].perc * 60 + jadat[i].mperc + jadat[i].seb
> elozoido) {
				aktualisido = jadat[i].ora * 3600 + jadat[i].perc * 60 +
jadat[i].mperc + jadat[i].seb;
				ki << aktualisido / 3600 << " " << (aktualisido % 3600) / 60 <<
" " << aktualisido % 60 << endl;
				elozoido = aktualisido;
			}
			else
				ki << elozoido / 3600 << " " << (elozoido % 3600) / 60 << " "
<< elozoido % 60 << endl;
		}
	}
	ki.close();
	cout << "A Fájlba írás megtörtént." << endl;
}
```