PROGRAM DOKUMENTÁCIÓ

Programozás 2 (levelező tagozat) – Beadandó feladat 16.

Kivonat

Egy lövészet verseny eredményeit tároló fájl beimportálása, helyezések és statisztikák kiszámolása és a rangsor külső fájlba exportálása C++ programozási nyelv segítségével

Ádám Várhegyi-Miłoś
Y1JE9I@hallgato.uni-neumann.hu

Tartalomjegyzék

1. Fejlesztői dokumentáció

1.1. A program célja

A program feladata egy bemeneti állomány beolvasása a program könyvtárából, abból egy lövészeti verseny adatainak kiolvasása, kiértékelése, és az eredmények kiírása egy kimeneti állományba. Ezeken felül egy felhasználó által megadott versenyző adatait is részletesen ki kell elemeznie a programnak, és ennek eredményeit a képernyőre írnia.

Az idézett program és ezen dokumentáció a Neumann János Egyetem GAMF Műszaki és Informatikai Karának Programozás 2 tantárgy (2021-es levelező kurzus) teljesítése céljából íródott, az ott leírt általános követelményeket és a szerző számára kiszabott feladatot igyekszik kielégítő módon megoldani.

1.2. Előzetes rendszerterv

A bemeneti állománynak adott formátumúnak kell lennie a program működéséhez: az első sorban a versenyen részt vevők száma szerepeljen, majd minden további sor egy-egy versenyző lövéssorozatát jelenítse meg + és – karakterek segítségével, melyek közül előbbi a célt talált lövéseket, utóbbi pedig a célt tévesztetteket jelöli. A fájl neve „verseny.txt” legyen!

A megfelelő működés és a felhasználó által érthetetlen hibaüzenetek és elakadások elkerülése érdekében a program ellenőrizze a fenti feltételek teljesülését. A program generáljon hibaüzenetet és lépjen ki, amennyiben:

- az állomány megnyitása sikertelen,
- az állomány első sorában nem szám szerepel,[1]
- amennyiben az első sorban szám szerepel, de az nem egyezik meg az alatta található sorok számával, vagy a sorok száma nem 2 és 100 közötti.

A programnak ezen felül úgy kell működnie, hogy amennyiben üres sorok szerepelnek a forrásfájlban a rekordok között, az ne okozzon problémát a fájl olvasásakor, és csak a nem üres sorokat vegye figyelembe a sorok megszámlálásakor éppúgy, mint a tartalom beolvasásakor.[2]

Amennyiben az előzetes ellenőrzések sikeresek voltak, a programnak fel kell dolgoznia az állományban szereplő sorokat, melyekben kizárólag + és – karakterek szerepelhetnek. Ezt

[1] [2]
[2] [4], [5]

a programnak szintén érdemes ellenőriznie, és a felhasználó hibaüzenettel jeleznie, hogy érvénytelen karakter került az állományba, aminek feldolgozását így nem lehet minden kétséget kizárólag hibátlanul folytatni.

A + és – karakterek a fájlban a lövések eredményességét hivatottak jelölni: a pluszjel a célt talált lövést, míg a mínuszjel a célt tévesztett lövést jelenti. A programnak ezekből a jelekből meg kell tudnia állapítania, hogy a versenyzők:

- hány lövést adtak le,
- hány pontot szereztek.

A pontszerzést a következők szerint kell számolnia a programnak: a legelső lövéskor egy találat 20 pontot ér. Minden egyes tévesztéskor a következőleg megszerezhető pontok száma eggyel csökken, de nulla pont alá nem mehetünk. Fontos megjegyezni, hogy a versenyzők különböző számú lövéseket adnak le; nem adott számú alkalommal lőnek, hanem egy időkereten belül kell annyit lőniük, amennyit tudnak.

A beolvasott adatok alapján a program szóközökkel elválasztva kiírja azon versenyzők sorszámát, akiknek volt legalább kettő egymást követő találatuk, majd megadja azon versenyző sorszámát, aki a legtöbb lövést adta le (azok eredményességétől függetlenül).

Az eddigieken felül a programnak egyéb információkat is ki kell számolnia egy versenyző lövéseiről. Ezen versenyző rajtszámát a felhasználó a program futása során bekérésre adja meg, ekkor a program kiírja, hogy:

- a versenyzőnek hányadik lövései találtak célt,
- összesen hány lövése talált célt,
- hány találatból állt a leghosszabb hibátlan sorozata,
- a versenyző pontszáma (amit már korábban kiszámolt a program).

Az előzetesen megfogalmazott igényeken túlmenően a program fejlesztője feltételezi, hogy a felhasználó esetlegesen további versenyzők adatait is szeretné lekérdezni, mely esetben a program újbóli és újbóli lefuttatása túlontúl körülményes volna. Ezért a programba beépítésre kerül egy egyszerű menürendszer, ahol az első versenyző részletes adatainak kiértékelése után a felhasználó meghatározhatja, hogy további versenyzők adatait is kívánja-e lekérdezni, vagy inkább a program további lépéseivel folytatná.

Utolsó lépésként a programnak el kell készítenie a verseny rangsorját az elért pontszámok alapján, melyet egy megadott nevű szöveges állományba kell exportálnia. Fontos adalék, hogy rangsoroláskor kizárólag a pontszám veendő figyelembe; azonos pontszám esetén holtversenyben kapják a versenyzők a helyezést, majd utánuk kimarad egy helyezés (tehát

két első helyezett esetén a második legnagyobb pontszámot elért versenyző a harmadik helyezést kapja, míg hármas holtverseny esetén a negyedik helyezés járna neki).

1.3. Fejlesztői környezet, programnyelv

A program a Microsoft Visual Studio 2019 felhasználói környezetben kerül megírásra, C++ nyelven. Egyéb felhasználói környezetet, illetve nyelvet nem használ a fejlesztő.

1.4. Részletes rendszerterv

1.4.1. Képernyőtervek

Az 1.2 fejezetben leírtak alapján a bemeneti állománynak (melynek a „verseny.txt” nevet kell viselnie) az **Hiba! A hivatkozási forrás nem található.**1. ábrán látható példa szerint kell kinéznie.

```
3
+---++-
-+-+-+++
--+-++++--
```

1. ábra: példa a beolvasandó fájlra

Eszerint a példa szerint a versenyen három résztvevő volt, lövéssorozataikat a plusz- és mínuszjelek jelzik. Külön indítóképernyővel a program nem rendelkezik, indításkor azonnal megnyitja, és beolvassa a bemeneti állományt.

Amennyiben a fájl nem hibás, a program elsőként kiírja a képernyőre a beolvasott adatokat, és a hozzájuk tartozó pontszámokat. Ezt azért teszi, hogy a felhasználónak legyen egy egységes képe a versenyzők eredményéről, illetve, hogy amikor egy-egy versenyzőre lekérdezi a részletesebb információkat, tudjon mi alapján választani a listából, és visszaellenőrizhesse a lekérdezés eredményeinek helyességét. A kiírandó lista az olvashatóság kedvéért táblázatszerű formába lesz rendezve, ahogy azt a 2. ábra mutatja.

```
RAJTSZÁM    LÖVÉSEK         PONTSZÁM
1           +---++-         54
2           -+-+-+++        88
3           --+-++++--      86
```

2. ábra: a beolvasott adatok kiírása képernyőre

A legalább két egymást követő találatot jegyző, a legtöbb lövést leadó versenyzők sorszámai, majd a felhasználó által kiválasztott versenyző részletes adatai mind hasonló formában, tömören egy-egy sorban kerülnek kiírásra, előttük külön sorban a feladat sorszámával:

```
2. feladat:
Az egymast kovetoen tobbszor talalo versenyzok: 1 2 3
3. feladat:
A legtobb lovest leado versenyzo rajtszama: 3
Adjon meg egy rajtszamot! [ide írhatja a felhasználó a rajtszámot]
```

Amikor a program kiírta a felhasználó által kiválasztott versenyző részletes adatait, a felhasználónak lehetősége van egy újabb versenyző adatait lekérni, vagy tovább haladni a programmal. Ezt egy kezdetleges menürendszerrel oldja meg a program a 3. ábrán látható módon.

```
********* VALASSZON EGY OPCIOT *********
1) Ujabb versenyzo adatainak lekerese
2) Kilepes a funkciobol es a program folytatasa
```

3. ábra: opcióválasztó menü

Az 1-es opció kiválasztásakor ismétlődik az előző lépés: a felhasználó újabb rajtszámot adhat meg, melyhez a program kiírja a vonatkozó adatokat, majd ismét előjön a fenti menürendszer. A 2-es opció kiválasztásával a program utolsó részéhez érkezünk.

Az utolsó szekcióban, a program kiírja a képernyőre a teljes, immáron pontszámok alapján rangsorolt listát (lásd 4. ábra), és ezt a háttérben egy „rangsor.txt” állományba is kiexportálja (ekkor már a fejlécek nélkül).

```
HELYEZES        RAJTSZAM        PONTSZAM
1               2               88
2               3               86
3               1               54
```

4. ábra: a rangsor képernyőre írása

1.4.2. A forráskódban felhasznált változók

A programban előforduló funkciók számossága miatt a program nem egy forrásfájlban kerül megírásra. Mivel osztályt is felhasznál, úgy célszerű, hogy annak definíciója külön fejlécállományban, annak metódusai pedig egy elkülönült forrásfájlban legyenek.[3] A main függvény (ami a programot indítja, információt ír a képernyőre, illetve adatot kér be a felhasználótól) és a kapcsolódó egyéb függvények, szintén különálló forrásállományban találhatóak. Ez a struktúra biztosítja, hogy az egymástól elkülönült funkciójú kódrészletek fizikailag is el legyenek választva, megkönnyítve ezzel a kód olvasását, javítását és az esetleges további fejlesztéseket.

1.4.2.a) A main függvény és kapcsolódó funkciói

A program main függvénye a `loves.cpp` forrásfájlban található. Itt kizárólag a program indítása, a lépések leírásának képernyőre írása, illetve adatbekérés történik, illetve két

[3] Ennek a megszervezéséhez a [14] forrás 6. fejezetének „Where to Put Class Declarations and Method Definitions” című szekcióját használta fel a szerző (https://learning.oreilly.com/library/view/sams-teach-yourself/0672327112/ch06.html)

függvény is itt van: a korábban említett menürendszert kezelő, illetve egy másik, ami egy szövegről megállapítja, hogy kizárólag számjegyekből áll-e. Az adatokat egy osztály dolgozza fel; mivel ezek külön fájlban találhatóak, később részletezzük őket.

Ennek megfelelően a `main` függvényben csak a program indításához és a menürendszer kezeléséhez elengedhetetlenül szükséges változók vannak. A bemeneti és kimeneti fájlok nevét egy-egy karaktertömb típusú változó tárolja, ezek az `infajlnev` és `outfajlnev`. Értéküket a kódon kívülről nem lehet megváltoztatni. A fájlt feldolgozó `Loveszet` osztályból a main függvényben egy `L` nevű objektum indít egy példányt. Időt is mérünk a main függvényben: `clock_t` típusú `kezd` és `veg` nevű változók tárolják a program futásának kezdetét és végét, melyek alapján a program kilépés előtt kiszámolja a program futási idejét.

Szintén a main függvény kezeli a 3. ábrán látható egyszerű menürendszert. A két opciót egy `Opciok` nevű enumerátor tartalmazza, a kapcsolódó `int` típusú `menukiiras` funkció végeredményét is ezen opciók egyikének felelteti meg a main függvény: ez a `valasztas` nevű egész szám változó. Mivel a menükiírást legalább egyszer futtatjuk, de bármennyiszer újra meghívhatjuk, egy `while` ciklusba van ágyazva; ehhez kapcsolódnak a `bool` típusú `fkilep` flag, és az int típusú bekeres.[4]

1. táblázat: a loves.cpp forrásfájl változói

Függvény	Változó neve	Változó típusa	Leírás
main	**infajlnev[]**	char	A bemeneti fájl nevét tárolja (kódba ágyazva).
main	**Outfajlnev[]**	char	A kimeneti fájl nevét tárolja (kódba ágyazva).
main	**L**	Loveszet	A Lövészet osztály egy példánya. Az ott kiszámolt adatokat az L publikus változói és metódusai írják a képernyőre. L-be írni a main függvény nem tud.
main	**kezd**	clock_t	A program futásának kezdeti ideje.
main	**veg**	clock_t	A program futásának vége.
main	**bekeres**	int	A felhasználó által megadott versenyző rajtszáma, input. A program ezt a rajtszámot adja meg a `Loveszet` ciklus azon metódusaiba argumentumként, amelyek egy-egy versenyző részletes adatainak lekérést szolgálják.
main	**sbekeres**	string	A felhasználó által megadott versenyző rajtszáma, input. Először ebbe a string változóba kerül az input. A kód megvizsgálja,

[4] Ennek a megtervezéséhez a [14] forrás 7. fejezetének „Controlling Flow with Switch Statements” című szekcióját (https://learning.oreilly.com/library/view/sams-teach-yourself/0672327112/ch07.html), továbbá ugyanezen forrás „Week 1. In Review” című fejezetét (https://learning.oreilly.com/library/view/sams-teach-yourself/0672327112/pt02.html) használta fel a szerző/fejlesztő.

			szám-e az érték, és ha igen, az átkerül a `bekeres` egész szám változóba.
main	**Opciok**	`enum`	`lekerdez` = 1, `kilep` = 2 A menürendszer két opciója. A `menukiiras` int függvény kimenete a lenti `valasztas` változóban tárolódik el, ezt feleltei meg egy `switch` elágazás valamelyik opciónak (a `default az` érvénytelen bevitelnek felel meg, pl. 3-at adtak meg).
main	**valasztas**	`int`	A `menukiiras` funkció végeredménye kerül ebben a változóban eltárolásra. A main függvény ezt felelteti meg az `Opciok` enumerátor valamelyik elemével (vagy a `default` paraméterrel, ha a felhasználó érvénytelen opciót, pl. 3-at adott meg).
main	**fkilep**	`bool`	Szabályozza, hogy hányszor fussanak le a versenyzők részletes eredményeit kiíró osztálymetódusok, illetve maga a menürendszer. Értéke igaz, ha a felhasználó a menürendszerben a 2-es (kilép) opciót választja, vagy érvénytelen opciószámot ad meg. Ekkor a program kilép a menürendszerből, és folytatódik a main függvény.
menukiiras	**opcio**	`int`	A kiválasztott menü opciója, ezt a változót adja vissza végül a függvény.
menukiiras	**cimhossz**	`int`	A menürendszer címének hossza.
menukiiras	**opcmaxhossz**	`Int`	A két opció leírása közül a hosszabbik szöveg hossza. A címhossz és ez a változó egyaránt felezve lesz, és ennyi számú szóközt ír a funkció a cím elé a középre zárás szimulálása érdekében.
menukiiras	**cim**	`string`	A menürendszer címe.
menukiiras	**opcstr_lekerdez**	`string`	A lekérdezési opció leírása.
menukiiras	**opcstr_kilep**	`string`	A kilépési opció leírása.
menukiiras	**szokozok**	`string`	A cím középre zárásához meghatározott számú szóközre van szükség. Ez az *n* darab szóköz ebben a változóban tárolódik el, és a cím elé kerül képernyőre íráskor.

1.4.2.b) A Lövészet osztály változói

A program központi eleme (a fájlkezelő változók, funkciók, tehát a bemeneti állományt feldolgozó rész) egy `Loveszet` nevű osztályban kerül megvalósításra. Az osztály kizárólag a fájlnevet kéri be argumentumként, az összes eddig felsorolt számítás az osztályban elhelyezett metódusokkal történik.

Az osztályban kizárólag azon változók, illetve függvények publikusak, melyekhez valamilyen módon hozzá kell férnie a main függvénynek, mint például a létszám, pontszám,

vagy a többi azon függvény, melyeknél különböző versenyzőkhöz kérhet le a felhasználó egyesével részletes adatokat.

Az versenyzők adatai (lövések, lövések száma, pontszám, helyezés, rajtszám) egy struktúra tömbben tárolódnak. Az osztály közvetlenül olyan adatokat tárol, melyek az egész versenyre érvényesek (pl. létszám). A versenyzők kerülhettek volna egy külön osztályba, és a `Loveszet` osztály mindegyik versenyzőre meghívhatott volna egy új példányt, de a programnak ebben a verziójában erre nem került sor, az egyszerűség kedvéért egy struktúra tömbben menti el a program a versenyzők adatait. Teszi ezt azért is, mert külön függvényekre nincsen szükségünk a versenyzők esetében, kizárólag adatot teszünk a tömbbe, melyeket később kiolvasunk. Ez persze nem teljesen így van, hiszen a `Loveszet` osztálynak vannak metódusai, melyek egyszerre egyetlen, argumentumként megadott versenyzőre számolnak ki adatokat (pl. leghosszabb pozitív sorozat), de ezt ugyanígy meg lehet oldani a `Loveszet` osztályból is, ráadásul a felhasználón múlik, hogy hány versenyző részletes statisztikáit szeretné lekérni, így felesleges lehet egy alosztályban ezen értékeknek memóriát lefoglalni, ha végül nagy valószínűséggel nem használjuk fel őket.

Fontos információ továbbá, hogy nincsen a `Loveszet` osztálynak olyan metódusa, amely külső forrásból változtatná meg bármely privát változójának értékét. Ez azért van így, mert a verseny már megtörtént, az ott felvett adatok fixek, utólag nem változtatjuk meg őket; a program csupán a már meglévő adatokat dolgozza fel.

2. táblázat: a Loveszet osztály fejlécállományának változói és metódusai

Elérés	Változó neve	Változó típusa	Leírás
publikus	**Versenyzo**	`struct`	A versenyzők adatait táruló struktúra. `string slovesek`, `char clovesek[40]`: A lövéssorozat +/– karaktereit tárolják. `Int pontszam`: a versenyző pontszáma. `Int l`: a lövések száma.[5] `Int helyezes`: a versenyző helyezése. `Int rajtszam`: a versenyző rajtszáma.
publikus	**Loveszet**		A `Loveszet` osztály konstruktorja. Beolvassa a fájlt, megvizsgálja az érvényességét és meghív más függvényeket a pontszámok kiszámolásához és a rangsor elkészítéséhez
publikus	**~Loveszet**		A `Loveszet` osztály destruktorja. Törli a `tmb` és a `rangsor` struktúratömböket.
publikus	**getLetszam**	`int`	A verseny résztvevőinek számát adja vissza a main függvény számára. A main függvényben a felhasználótól bekérünk egy számot a részletes lekérdezéshez, ehhez kell tudnia a main függvénynek, hogy

[5] [7]

			érvényes-e a felhasználó által megadott érték.
publikus	**getPontszam**	`int`	A felhasználó által bekért rajtszámhoz képest eggyel kisebb indexű versenyző pontszámát adja vissza a main függvény számára.
publikus	**kiir**	`void`	A beolvasott állomány tartalmát írja ki a képernyőre.
publikus	**adatexport**	`void`	A pontszámok alapján rangsorolt `Versenyzo` struktúratömböt tartalmát írja ki a képernyőre, és exportálja egy fájlba.
publikus	**min_ket_ talalat_ kivalogat**	`string`	Kiválogatás tételével megvizsgálja a `Versenyzo` tömb elemeit, és visszaadja azon versenyzők rajtszámait, akiknek volt legalább két egymást követő találatuk.
publikus	**min_ket_ talalat_ stringfinddal**	`string`	`String::find`[6] funkcióval megvizsgálja, mely versenyzők rekordjában szerepel a „++” szöveg, és ezen rajtszámokat visszaadja. Megegyezik a `min_ket_talalat_kivalogat` metódussal, csak más megközelítéssel dolgozik.
publikus	**talalatok_ sorszamai**	`string`	Egybefűzött szövegként visszaadja egy adott rajtszámú versenyző találatainak sorszámait (hányadik lövéseik találtak).
publikus	**legtobb_loves**	`int`	Visszaadja a legtöbb lövést leadó versenyző sorszámát
publikus	**loertek**	`int`	Adott versenyzőre visszaadja a lövéssorozat alapján kiszámolt pontszámot.
publikus	**ossztalalt**	`int`	Adott versenyző összes találatainak száma.
publikus	**leghosszabb_ sorozat**	`int`	Adott versenyzőre visszaadja, hány találatból áll a leghosszabb pozitív sorozata.
privát	***tmb**	`Versenyzo`	Struktúra tömb, melyben a versenyzők adatait tároljuk.
privát	***rangsor**	`Versenyzo`	Pontszámok alapján rangsorolt `tmb` struktúratömböt tárolja.
privát	**v**	`int`	A versenyzők létszáma. `For` ciklusokban releváns.
privát	**LOVES_JELEK**	`const string`	+– jeleket tárolja a lövéseket leíró szövegek érvényességének megvizsgálásához.
privát	**PLUSZ**	`const char`	A + jel karakterkódját tárolja.
privát	**MINUSZ**	`const char`	A – jel karakterkódját tárolja.
privát	**ervenyes_szoveg**	`bool`	A beolvasott állomány adott sorára megmondja, hogy kizárólag érvényes, engedélyezett karaktereket tartalmaz-e.
privát	**rangsorol**	`void`	Pontszámok alapján sorba rendezi a `tmb` struktúra tömböt, majd helyezéseket rendel a versenyzőkhöz (nem mindig egyezik a sorrenddel, mivel lehet holtverseny).

[6] [1], [3]

A `Loveszet` osztály definíciója külön fájlban található a metódusaitól, erre azok viszonylag nagy száma miatt van szükség. A definíció a `Loveszet.h` fejlécállományban, míg a metódusok a `Loveszet.cpp` állományban találhatóak. A 3. táblázat mutatja be az osztály metódusainak változóit.

3. táblázat: A Loveszet.cpp metódusainak változói

Függvény	Változó neve	Változó típusa	Leírás
Loveszet	***filenev**	char	Argumentum. A beolvasandó bemeneti fájl nevét tárolja.
Loveszet	**be**	ifstream	A beolvasandó fájl objektuma.
Loveszet	**sorok_szama**	unsigned int	A fájlban lévő sorok száma. Nem lehet negatív szám.
Loveszet	**fejlec_letszam**	unsigned int	A beolvasandó fájl első sorában szereplő létszám. Nem lehet negatív szám.
Loveszet	**sor**	string	A beolvasott állomány egy-egy sorát tárolja el ideiglenesen a sorok megszámlálása céljából.
Loveszet	**nemerv_index**	int	Az érvénytelen sorok számát menti el ide a program.
Loveszet	**i**	int	A sorok tartalmának beolvasásához és a tömbök elemszámához használt index.
ervenyes_szoveg	**&szoveg**	const string	Argumentum. Adott sorban szereplő lövések karakterei.
ervenyes_szoveg	**erv_karakterek**	string	Argumentum. Elfogadható, a kiértékelés szempontjából érvényes karakterekből álló string.
ervenyes_szoveg	**rosszchar**	bool	Értéke igazra változik, ha érvénytelen karaktert talál a függvény a lövések rekordjában.
ervenyes_szoveg	**talalat**	int	A lövés sorozatok karaktereit egyesével megkeresi a program az érvényes plusz és mínusz karakterek alkotta stringben, ennek a keresésnek az eredménye kerül a `talalat` változóba. Ha az nulla, a karakter (és így a sor) érvénytelen.
ervenyes_szoveg	***szoveg**	char	Argumentum. Amennyiben a lövések karaktersorozatként kerülnek beolvasásra, ez a mutató típusú változó tárolja a sorozat nulladik elemének memóriacímét.
ervenyes_szoveg	**h**	int	Argumentum. Amennyiben a lövések karaktersorozatként kerülnek beolvasásra, ez a változó tartalmazza a sorozat hosszát.
ervenyes_szoveg	**i**	int	A lövés sorozatok karakterein végig haladó index.
min_ket_talalat_kivalogat	**s**	string	A kiválogatás tételével azonosított, sorozatban legalább két találattal rendelkező

			versenyzők sorszámai kerülnek szóközzel elválasztva ebbe a stringbe.
min_ket_talalat_kivalogat	**j**	`int`	Adott versenyző lövéseinek indexe.
min_ket_talalat_stringfinddal	**s**	`string`	A lövés sorozatokat string változóként beolvasva a kód mindegyik versenyzőnél eldönti, van-e legalább két találata. Ezen versenyzők sorszámai kerülnek szóközzel elválasztva ebbe a stringbe.
legtobb_loves	**maxlovo**	`int`	A legtöbb lövést leadó versenyző indexe.
loertek	***sor**	`char`	Argumentum. A lövés sorozatot alkotó karaktertömb nulladik elemének memóriacíme.
loertek	**hossz**	`int`	Argumentum. A lövés sorozatot alkotó karaktertömb hossza.
loertek	**aktpont**	`int`	Az aktuálisan megszerezhető pontot tárolja egy cikluson belül; értéke minden egyes célt tévesztett lövésnél csökken.
loertek	**ertek**	`int`	Adott versenyző által megszerzett pontok számát tárolja egy cikluson belül; értéke minden egyes találatnál növekszik.
	rajtszam	`int`	Argumentum. Több metódusban is előfordul. A felhasználó által megadott lekérdezendő rajtszámot adja meg.
	r	`int`	Több metódusban is előfordul. A felhasználó által megadott rajtszám argumentumként kerül be, r változónak eggyel kisebb értéket ad a program a nullával kezdődő indexálás miatt.
	***c**	`char`	Dinamikusan létrehozott tömb 0. elemének memóriahelye, több metódus is használja. Adott tulajdonság kiválogatásakor (pl. van-e legalább kettes sorozat, talált-e a lövés) töltődik fel elemekkel.
	db	`int`	Több metódus is használja. A `*c` dinamikusan létrehozott karaktertömb elemszámlálásához használt.
ossztalalt	**talalt**	`int`	Adott versenyző összes találatainak száma.
talalatok_sorszamai	**s**	`string`	A funkció visszaadja, adott versenyzőnek hányadik lövései találtak célt. Ezek a számok ebben a string változóban, szóközzel elválasztva tárolódnak.
leghosszabb_sorozat	**talalt**	`int`	Adott versenyző lövésein végighaladva ebben a változóban tárolja el a kód az első lövés vagy a legutóbbi tévesztés óta bevitt találatok számát.
leghosszabb_sorozat	**max**	`int`	Adott versenyző leghosszabb pozitív sorozatának elemszámát tárolja.
rangsorol	***a**	`Versenyzo`	Argumentum.

			Az eredeti versenyző struktúra pontos mására megkreált `rangsor` tömb 0. elemének memóriahelyére mutató változó. A metódus ezt a tömböt rendezi sorba.
rangsorol	**seged**	`Versenyzo`	Segédváltozó a sorba rendezéshez, két elem megcserélésekor használatos.
rangsorol	**szum**	`int`	Sorba rendezés után a metódus helyezéseket rendel a versenyzőkhöz. A holtverseny miatt ez nem egyértelmű. Ez a változó számolja meg, hogy az aktuális versenyzőnél összesen hány versenyző szerzett több pontot.
adatexport	**ki**	`ofstream`	Az exportálandó állomány objektuma.

1.4.3. Alkalmazott algoritmusok

A `Loveszet.cpp` forrásállomány metódusai közül némelyik elemi algoritmusokat is felhasznál, bizonyos kiegészítésekkel. Ezekből tekint át a fejezet néhányat.[7]

Az `ervenyes_szoveg` metódusnak két túlterhelése van; ebből az egyik (amelyik a lövések alkotta karaktersorozatra úgy mutat, hogy a nulladik elemre memóriahelyét és a sor hosszát[8] kéri be argumentumként) klasszikus eldöntés tételén alapuló algoritmussal határozza meg, van-e érvénytelen karakter egy adott versenyző lövéseit leíró karaktersorozatban. Az alaphelyzet tehát az, hogy csak plusz és mínusz karakterek vannak a sorban, a tulajdonság, amit keresünk, az bármilyen ettől eltérő karakter megléte. Eltérő karakter megtalálása esetén a `bool` függvény kimenete `false` lesz.

```
Bemenet: tmb[1…N], P tulajdonság (+ karakter), M tulajdonság (- karakter)
Kimenet: Eredmény: VAN érvénytelen karakter vagy nincs
Algoritmus:
   i:=1
   Ciklus amíg i <= N és (tmb[i] = P vagy tmb[i] = M)
      i:=i+1
   Ciklus vége
   Ha (i < N) akkor Eredmény:=”van olyan elem, amely nem P és nem M”
Algoritmus vége
```

Szintén az eldöntés tételét használja fel a `main` függvény forrásfájljában lévő `szamstring` függvény. Mivel egyéb elemeket is felhasznál, amelyek az algoritmuson kívül további magyarázatra szorulhatnak, ennek a függvénynek a bemutatása az 1.4.4 fejezetben olvasható.

A `min_ket_talalat_kivalogat` és a `talalatok_sorszamai` metódusok mindketten a kiválogatás tételét használják fel. Ezek közül az utóbbi szimplán kiválogatja a + karaktereket;

[7] Az összes idézett elemi algoritmus tartalma, illetve formátuma a [15] és [16] forrásszámú előadások anyaga alapján készült.

[8] [8], [9]

ennél valamivel bonyolultabb és tágabb értelmezésű az előbbi, amely azon versenyzők sorszámát válogatja ki, akiknek a lövései között található két egymás követő + karakter. A kiválogatás végén a karaktereket szóközzel elválasztva egy string változóba tesszük, a metódusok már ezt adják vissza a main függvénynek, tekintettel arra, hogy mivel osztállyal dolgozunk, olyan eredményt kiküldeni az osztályon kívülre, amit még át kellene dolgozni, nem lenne helyes. A legjobb a teljes működést az osztály metódusán belül megvalósítani. Alant a `min_ket_talalat_kivalogat` algoritmusát mutatjuk be:

```
Bemenet: tmb[1…N], E: tmb[i] elemszáma
Kimenet: c[]
Algoritmus:
   db:=0
   Ciklus 1-től N-ig
      j:=0
      Ciklus amíg (j <= tmb[i]E-1) és ((tmb[i]lövés[j] <> +) vagy
(tmb[i]lövés[j+1] <> +))
         j:=j+1
      Ciklus vége
      Ha (j <= tmb[i]E-1) akkor
         c[db]:=i
         db:=db+1
   Ciklus vége
Algoritmus vége
```

A `legtobb_loves` metódus a maximumkiválasztás tételét alkalmazza. A metódus a legtöbb lövést (függetlenül annak eredményességétől) leadó versenyző rajtszámát adja vissza, ami megegyezik a `Versenyzo` struktúra tömb indexének eggyel inkrementált értékével (az eggyel való inkrementálás a nulla kezdetű indexelés miatt szükséges).

```
Bemenet: tmb[1…N]
Kimenet: MAX:=a legtöbb lövést leadó versenyző rajtszáma
Algoritmus:
   SORSZAM:=1
   Ciklus i:=2-től N-ig
      Ha (tmb[i] > tmb[SORSZAM]) akkor SORSZAM:=i
   Ciklus vége
Algoritmus vége
```

Az `ossztalalt` metódus számolja össze egy versenyző összes lövéseinek számát; ehhez teljes mértékben elegendő a megszámlálás tételének módosítatlan alkalmazása.

```
Bemenet: tmb[1…N], P tulajdonság (plusz jel)
Kimenet: DB (a találatok darabszáma)
Algoritmus:
   DB:=0
   Ciklus i:=1-től N-ig
      Ha (tmb[i] P tulajdonságú) akkor DB:=DB+1
   Ciklus vége
Algoritmus vége
```

A `leghosszabb_sorozat` metódus az eddigiekkel eltérően nem az egyik alap algoritmust használja módosítás nélkül. A kiszámolandó érték itt egy adott versenyző által a leghosszabb pozitív sorozat. Ehhez az kell, hogy minden lövésen számba vegyen a függvény, és megszámolja, sorozatban hányadik pluszjelnél tart; ha mínuszjelet talál, akkor lenullázza a számlálót. Ezen elágazás előtt azonban (mielőtt növeljük, vagy nullázzuk a számlálót) azt is meg kell vizsgálnia a függvénynek, hogy az eddigi pluszjelek száma magasabb-e, mint a korábbi maximum; ha igen, akkor a maximumszámlálót is inkrementálja eggyel az algoritmus. Ebben a tekintetben az algoritmus felhasznál elemeket a maximum kiválasztás tételéből éppúgy, mint a megszámlálás tételéből. Ezeknek mintegy hibrid változatából kapjuk meg a leghosszabb pozitív sorozatot:

```
Bemenet: V[1…N], P tulajdonság //adott rajtszámú versenyző lövései ill.
pluszjel a lövések között
Kimenet: MAX
Algoritmus:
   TALALT:=0, MAX:=0
   Ciklus i:=1-től N-ig
      Ha (TALALT > MAX) akkor MAX = TALALT
      Ha (P tulajdonság) akkor
         TALALT:=TALALT+1
      különben
         TALALT:=0
      Elágazás vége
   Ciklus vége
Algoritmus vége
```

A legutolsó alkalmazott algoritmusok az osztály `rangsorol` nevű metódusában szerepelnek. Ez az eredeti `Versenyzo` struktúra tömbnek már egy másolatát rendezi sorba; ez azért történik így, mivel a programban szeretnénk, ha esetleges későbbi felhasználáskor, bővítéskor megmaradna az eredeti tömb is, az eredeti sorrendben. A másolat tömböt az elért pontszámok alapján buborékos rendezéssel rendezi csökkenő sorrendbe a program. A csökkenő sorrend miatt a belső, elemcserélő ciklus elágazásában a relációs jelet meg kellett fordítani.

```
Bemenet: ÚJ[1…N] //az eredeti tömb másolata rendezetlenül
Kimenet: ÚJ[1…N] //az eredeti tömb másolata rendezve
Algoritmus:
   Ciklus i:=2-től N-ig
      Ciklus j:=N-től i-ig -1-esével
         Ha (ÚJ[j-1] < ÚJ[j]) akkor      SEGED:=ÚJ[j-1]
                                         ÚJ[j-1]:=ÚJ[j]
                                         ÚJ[j]:=SEGED
         Elágazás vége
      Ciklus vége
   Ciklus vége
Algoritmus vége
```

Ezzel a sorba rendezés még nem teljes, mivel meg kell állapítsuk a versenyzők helyezését, ami nem egyezik meg a sorba rendezett tömb indexszámával (vagy annak eggyel inkrementált értékével), mivel holtverseny is kialakulhat. Minden versenyző helyezése megegyezik a nála több pontot szerzett versenyzők számának eggyel növelt értékével. Tehát ha két első helyezett van, a második legtöbb pontszámot szerzett versenyző a harmadik lesz. Hármas holtverseny esetén két helyezés „esik ki”, ekkor a második legtöbb pontszámot szerzett versenyző a negyedik helyet szerzi meg. Ezeknek a kívánalmaknak megfelelően működik a versenyzőkhöz helyezést rendelő algoritmus:

```
Bemenet: Pontszám[1…N] //struktúra töm pontszám változója
Kimenet: Helyezés      //a struktúra tömb egyik változója
Algoritmus:
   Ciklus i:=1-től N-ig
      HELY:=0
      Ciklus j:=1-től i-ig
         Ha (Pontszám[j] > Pontszám[i]) HELY:=HELY+1
      Ciklus vége
      HELY:=HELY+1
      Helyezés[i]:=HELY
   Ciklus vége
Algoritmus vége
```

1.4.4. A program függvényeinek működése

Ez a fejezet az eddig leírt algoritmusokon túl néhány másik függvény működését írja le. Ezek nem feltétlenül jól ismert algoritmikus tételek felhasználásával épültek fel, de működésük megértése fontos lehet a kód funkcionalitásainak megismeréséhez.

A `Loveszet` osztály konstruktorában a kód dinamikus módon megkreálja a `Versenyzo` struktúra tömböt; ehhez először meg kell állapítani, pontosan hány versenyző vett részt a versenyen. Ez megegyezik a bemeneti állományban található sorok számával (mínusz egy, mivel az első sorban szintén a résztvevők száma található). A sorok számának megállapításához meg kell nyitnunk a bemeneti állományt, a `getline` függvénnyel beolvasnunk a nem üres sorokat (arra az esetre, ha üres sorok lennének a fájlban)[9], majd egyet levonunk az eredményből.[10] Végül a kód visszaugrik a fájl elejére a soron következő adatbeolvasás végett.[11]

```
while (!be.eof()) { //a be egy ifstream objektum
   getline(be, sor);
   if (sor != „”) {
      sorok_szama++;
   }
}
be.clear();
be.seekg(0);
```

[9] [10]
[10] [4], [6]
[11] [5]

Később a kód megvizsgálja, hogy az első sorban szereplő szám numerikus-e, és megegyezik-e az alatta lévő sorok számával.[12]

Az adatok beolvasásakor és a már dinamikusan létrehozott struktúra tömbbe mentésekor egy elöl tesztelős `while` ciklus figyeli, elérte-e a kód az end of file flaget, nem haladta-e meg a beolvasott sorok száma a versenyzők számát és be tudja-e olvasni a karaktereket a struktúra tömbbe (tehát nem üres-e a sor):

```
while (!be.eof() && i < v && be >> tmb[i].slovesek) {…}
```

A lövéseket jelző karaktereket elsőként tehát stringbe olvassa a kód, mivel így sokkal egyszerűbb megállapítani a lövések számát, aminek segítségével egyrészt megállapíthatjuk, nem haladja-e meg a felírt lövések száma a maximálisként megszabott 40-et (ha igen, csak az első 40 karaktert veszi figyelembe a kód a `substr`[13] funkció segítségével), másrészt pedig egyből el is menthetjük ezt az értéket a struktúra tömbben a leadott lövések számát tárolni hivatott `l` változóban.[14]

```
tmb[i].l = tmb[i].slovesek.length();
```

Ezután karakterekként is elmentjük a lövéseket tartalmazó stringet[15], mivel azokon könnyebben lehet algoritmikus műveleteket végrehajtani, illetve a program előzetes kívánalmainak is ez a formátum felel meg inkább. Ez egy `for` ciklus segítségével kerül megvalósításra: a string karakterei egyenként rendelődnek a karaktertömb elemeihez.

Az adatok beolvasása után a függvény a már korábban leírt módon megvizsgálja, érvényes-e az adatrekord, tehát kizárólag plusz és mínusz karakterek szerepelnek-e benne. Amennyiben nem, az adatok beolvasása tovább folyik, annak érdekében, hogy minden érvénytelen sorról tájékoztathassa a kód a felhasználót. Ha nem így lenne, akkor – ha például 5 hibás karaktert tartalmazó sor lenne a fájlban – a felhasználó ezeket csak öt alkalommal való futtatás után találná meg mind. Ha bármely sor érvénytelen karaktert tartalmazott, adatbeolvasás után a program ezek mindegyikéről tájékoztatást ad a felhasználó számára, szünetelteti a futását, hogy a felhasználó feljegyezhesse a hibás sorokat, majd kilép.

A pontszámot minden versenyző számára egyesével kiszámoló `loertek` függvény egy már, az igényfelmérés során kapott algoritmuson alapul, így azt szinte változtatás nélkül közöljük:

```
Függvény loertek(*sor:karaktersorozat 0. elemének memóriahelye, hossz:
egész szám):egész szám
   aktpont:=20
```

[12] [13] [17]
[13] [11]
[14] [7]
[15] [12]

```
   ertek:=0
   Ciklus i:=1-től hossz(sor)-ig
      Ha aktpont>0 és sor[i]=”-” akkor
         aktpont:=aktpont-1
      Különben
         ertek:=ertek+aktpont
      Elágazás vége
   Ciklus vége
   loertek:=ertek
Függvény vége
```

A `main` függvényben található az a függvény, amelyik kiírja azt a menüt a képernyőre, ahol a felhasználó kiválaszthatja, szeretne-e további versenyzőktől részletes statisztikákat lekérni, vagy inkább kilép ebből a funkcióból, és program további részeinek futtatásával folytatná. Az opciók egy két tagú enumerátorban vannak meghatározva: `lekerdez` és `kilep`. A main függvényben egy elöl tesztelős `while` ciklusban fut a részletes statisztikák lekérdezése. A legelső futáskor a `lekerdez` opció az alapértelmezett, így egyszer mindenképpen lefut. A ciklus lehetett volna hátul tesztelős is, de mivel az alapértelmezett értéket a cikluson kívül már hozzárendeltük a változóhoz, ami valamelyik enumerátor tagot tárolja, a ciklus működésén az sem változtatott volna.

A cikluson belül egy függvény kiírja a menürendszert a képernyőre, és egész szám formátumú inputot kér a felhasználótól. A `main` függvény ciklusában lévő feltétel ezt veti össze egy `switch` elágazás segítségével az enumerátorokkal. Amennyiben a `lekerdez` opciónak megfelelő 1 került bevitelre, újabb lekérdezés következik, minden más esetben a függvény kilép a ciklusból.

```
while (!fkilep) {
   //… (részletes adatok képernyőre írása)
   valasztas = menukiiras(); //a menukiiras függvény int eredménye
   switch (valasztas) {
      case lekerdez: break;
      case kilep: fkilep = true; break;
      default: cout << „Ervenytelen parameter…”; fkilep = true; break;
   }
}
```

Maga a menürendszer képernyőre írása egy külön, `menukiiras` nevű függvényben történik. A címet, lekérdezést és a kilépést leíró szövegek a `cim`, `opcstr_lekerdez` és `opcstr_kilep` string konstansokban vannak elmentve. Annak érdekében, hogy a cím a két opció felett középen helyezkedjen el, szóköz karaktereket ír elé a függvény. Ezt az említett string konstansok hosszából számolja ki: a hosszabbik opció szövegét felezi a függvény, és kivonja belőle a cím hosszának a felét. Az `opcmaxhossz` (`int`) változóban a hosszabbik szöveg hossza szerepel.

```
for (int i = 0; i < (opcmaxhossz/2 – cimhossz/2); i++) szokozok += „ „;
```

Végül szintén a `main` függvény moduljában szerepel egy másik függvény, amely a menürendszer kiírásakor a felhasználótól bekért adatról állapítja meg, hogy tisztán számjegyeket tartalmaz-e az input. A függvény egy bool értéket ad vissza, és a következőképpen működik: a felhasználó beírt egy értéket a képernyőre, melyet egy string változóban mentettünk el, a függvény argumentumra pedig ennek a string változónak az értéke. Magában a függvényben létrehozunk egy iterátort, amely induláskor a string első karaktere. Ezek után egyesével az összes karaktert megvizsgáljuk az iterátor segítségével egy `while` cikluson belül. Ez a `while` ciklus addig tart, amíg a string végére nem érünk vagy nem numerikus karaktert észlel a kód. Ha vizsgált string nem üres, és az iterátor a szó végére ért, az input numerikus.

```
string::const_iterator it = s.begin();
while (it != s.end() && isdigit(*it)) ++it;
return !s.empty() && it == s.end();
```

1.5. Tesztelés

1.5.1. Teszteléshez használt hardver és szoftver

A program az alábbi specifikációkkal rendelkező számítógépen került tesztelésre, Visual Studio 2019 16.8.4 alatt:

- CPU: Intel Core i7 6700 HQ
- 2x 8,0 GB DDR4 RAM (Kingston 9905630-007.A00G)
- SSD merevlemez (Kingston SM2280S3120G)
- operációs rendszer: Microsoft Windows 10 Home 10.0.19041

1.5.2. Tesztelt helyzetek

4. táblázat: a program tesztelése, eredményei és a hibák megoldásai

Tesztelt szcenárió	Eredmény	Megoldás
Átnevezett / nem létező fájl.	A program a megfelelő hibaüzenettel kilép.	N/A
Nem numerikus karakter beírása a bemeneti állomány első sorába.	A program a megfelelő hibaüzenettel kilép.	N/A
Az állomány első sorába nem a résztvevők számának megfelelő szám beírása.	A program a megfelelő hibaüzenettel kilép.	N/A
Túl kevés (egy) résztvevő a fájlban.	A program a megfelelő hibaüzenettel kilép.	N/A
Több mint 100 résztvevő a fájlban.	A program a megfele hibaüzenettel kilép.	N/A
Üres fájl.	A program nem lépett ki, üres stringet próbál sorszámnak megfeleltetni.	Új hibakezelő, ami kilépteti a programot, ha a sorok száma 0.

Üres sorok beszúrása a fájl elejére és véletlenszerűen kiválasztott sorok közé.	Nem befolyásolja a program futását, a program ignorálja az üres sorokat.	N/A
Többszörös üres sorok beszúrása véletlenszerűen kiválasztott sorok közé, illetve a legutolsó kitöltött sor után.	Nem befolyásolja a program futását, a program ignorálja az üres sorokat.	N/A
Érvénytelen karakterek beszúrása a plusz- és mínuszjelek közé. A legutolsó sor alá egy érvénytelen karaktert írtunk.	A program azzal a hibaüzenettel lép ki, hogy a fejlécben szereplő szám nem egyezik a sorok számával, bár a kívánt hibaüzenet az érvénytelen karakterekre vonatkozott volna.	A program nem került javításra, mivel az utolsó sor alá beírt érvénytelen karakter miatt valóban több sorunk lett, mint ahány résztvevő van, a hibaüzenet technikailag nem téved.
Érvénytelen karakterek (betűk számok, speciális karakterek) beszúrása a plusz- és mínuszjelek közé. A sorok száma megegyezik a résztvevők számával.	A program a megfelelő hibaüzenettel kilép.	N/A
Szóköz beírása a plusz- és mínuszjelek közé.	A program nem reagált a helyzetre, a szóközig olvasta az adott sort.	Sorszámláláskor egy `find` funkció megnézi, van-e szóköz a sorban. Ha igen, a program hibaüzenettel kilép. Nem az `ervenyes_szoveg` függvénnyel nézzük meg, mivel, ha a kód odáig jut, eleve nem olvassa be a szóközöket.
100 karakternél hosszabb lövéssorozat beírása a beolvasandó fájlba.	A program nem kezelte a 100 lövéses limitet, majd hibára futott.	A program elsőként stringbe olvassa az adatokat, ha meghaladja a hossza a 40-et, csak az első 40-et jegyzi meg, és csak ezután menti el a karaktertömbben is az értékeket.
4 karakternél rövidebb lövéssorozat beírása a beolvasandó fájlba.	A program nem kezelte a 4 lövéses minimumot.	A konstruktor adatbeolvasásra vonatkozó része kiegészítésre került: 4 lövés alatt a program minden karaktert mínuszjelre cserél, és 0 pontot ad a versenyzőnek.
Az opcióválasztó menünél érvénytelen opciószám megadása.	A program a megfelelő hibaüzenetet írja ki, majd folytatja a futást.	N/A
Az opcióválasztó menünél nem numerikus karakter megadása.	A program nullának vette a beírt értéket, majd végtelen ciklusba került, ignorálva a közbenső cin utasítást.	A rajtszámbekérésnél string változóba tesszük a beírt értéket. Egy újonnan írt funkció megvizsgálja a string karaktereit, és megnézi, azok számjegyek-e. Ha nem numerikus karakter is van a string változóban, a program új értéket kér be.[16]

[16] [2] [13] [17]

1.6. Továbbfejlesztési lehetőségek

Továbbfejlesztési lehetőségekként felmerülhet a részletesebb statisztikák készítése, és ezen adatok adatbázisba vagy táblázatkezelő programba esetleg más fájlformátumba (pl. csv) történő exportálása. Ideális esetben egy weboldal vagy akár egy mobil applikáció megléte esetén ezek adatbázisaiba küldhetné az adatot a program, ahol aztán a résztvevők és az érdeklődők online elérhetnék a legfrissebb eredményeket és statisztikákat.

A kód felépítését illetően a későbbiekben lehetőség nyílhat egy – a modern felhasználói igényeket sokkal inkább kielégítő - letisztultabb kezelői felület kidolgozására.

Ezen kívül a fejlesztendő terület lehet a `Versenyzo` struktúra tömb átalakítása a `Loveszet` osztály alosztályává.

2. Felhasználói dokumentáció

2.1. Futási környezet, követelmények

Kizárólag a `.cpp` és a `.h` kiterjesztésű állományok (loves.cpp, `Loveszet.cpp`, `Loveszet.h`) birtokában a programhoz ezen állományok importálására és futtatására alkalmas szoftver ajánlott, pl. Microsoft Visual Studio.

Amennyiben a program `.exe` kiterjesztésű állomány formájában is rendelkezésre áll, akkor annak futtatásához Microsoft Visual Studio megléte nem szükséges, csupán Microsoft Windows operációs rendszer megléte elegendő.

A program futtatásához 2 MB memóriára ill. Microsoft Windows operációs rendszerre van szükség. A program futásához igen csekély erőforrás szükséges, egyéb minimumkövetelményt így nem szabunk ki.

5. táblázat: a program szoftver és hardver követelményei

Komponens	**Követelmény**
Operációs rendszer	Microsoft Windows
Memória	2 MB
Fordító	Visual C++ fordító (interpreter)
Ha az exe fájl nem áll rendelkezésre	Microsoft Visual Studio, vagy más, `cpp` és `h` kiterjesztésű fájlok futtatására alkalmas program.

2.2. A program használata

2.2.1. A program telepítése és indítása

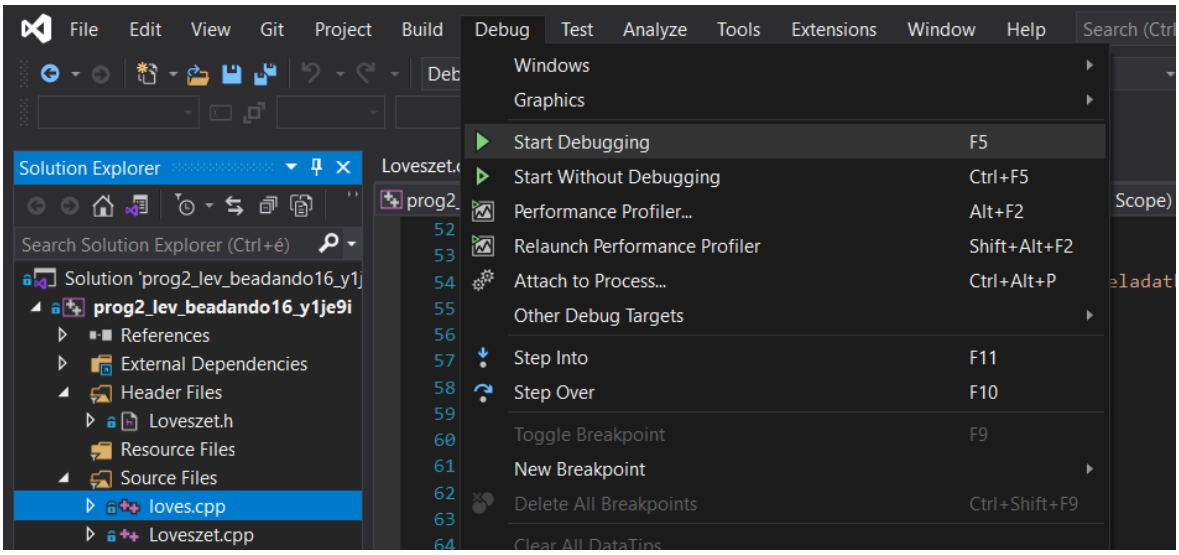
Amennyiben nem áll rendelkezésre az `exe` parancsfájl, hozzon létre egy új C++ projektet Microsoft Visual Studio környezetben, és importálja be a következő fájlokat:

- `loves.cpp` (forrásállomány)
- `Loveszet.h` (fejlécállomány)
- `Loveszet.cpp` (forrásállomány)

`Exe` fájl megléte esetén nincsen szükség külön telepítésre, a programot az `exe` állománnyal lehet majd indítani.

Indítás előtt győződjön meg róla, hogy a program könyvtárában megtalálható a `verseny.txt` állomány, amelyet a program be fog olvasni!

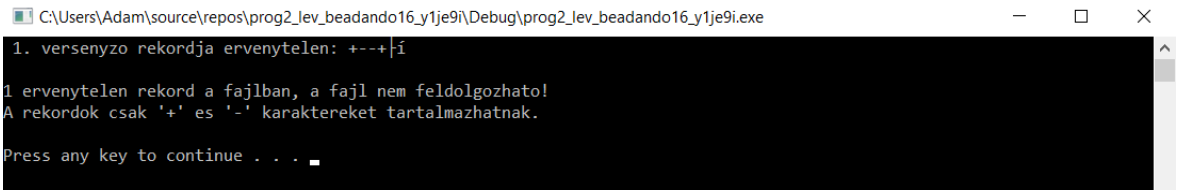
A program indításához futtassa az `exe` állományt, vagy Visual Studioban indítsa el a programot az F5 billentyűvel!



5. ábra: A program indítása Visual Studio programban („Start Debugging”)

2.2.2. A program működése

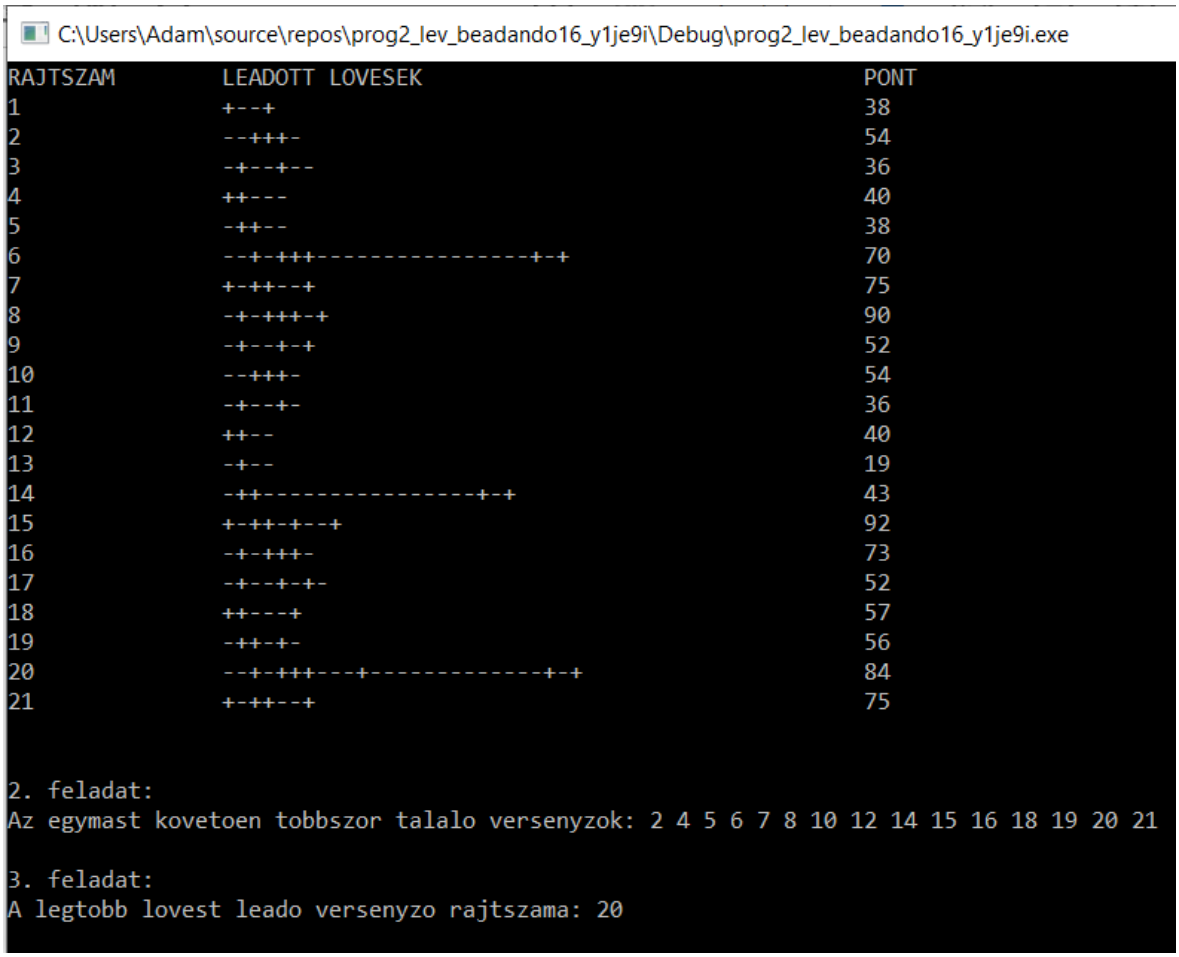
A program első lépésként beolvassa a `verseny.txt` nevű állományt. Amennyiben az állomány nem elérhető, vagy a tartalma esetleg annak formátuma nem teszi lehetővé az adatok feldolgozását, hibaüzenet értesíti a felhasználót a hiba mibenlétéről. Olvassa el a hibaüzenetet, majd nyomjon meg egy billentyűt! A program futása ekkor leáll. Vizsgálja meg a hibaüzenet által jelzett részeket, ha lehet, javítsa ki, majd futtassa ismét a programot!



6. ábra: hibaüzenet a verseny.txt beolvasásakor

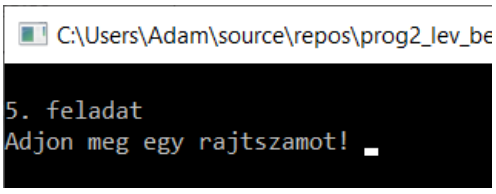
Többféle hiba is felmerülhet, ezekről részletesen a 2.3 fejezet ad tájékoztatást.

Sikeres állománybeolvasás esetén a program kiírja a képernyőre a beolvasott adatokat. Táblázatszerű elrendezésben megjeleníti a versenyzők rajtszámait, a leadott lövéseket (a + jel találatot, míg a – jel tévesztést jelez), és az elért pontszámot. Külön szekcióban kapunk információt azon versenyzők rajtszámáról, akiknek volt legalább két egymást követő találatuk; a program ezt a „2. feladat” címkénél írja ki. A „3. feladat” című résznél a program megadja a legtöbb lövést leadott versenyző rajtszámát.

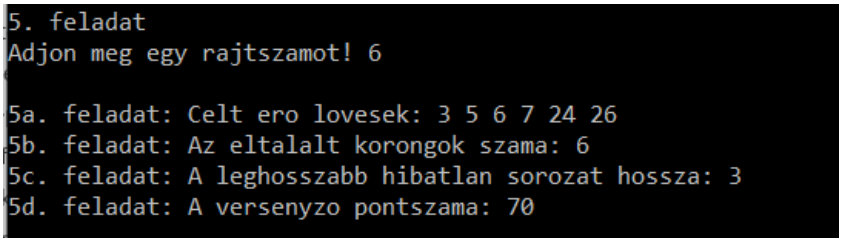


7. ábra: beolvasott adatok, pontszámok

A következő részben meg kell adnia egy versenyző rajtszámát. Ehhez a versenyzőhöz a program további adatokat számol ki, és jelenít meg.



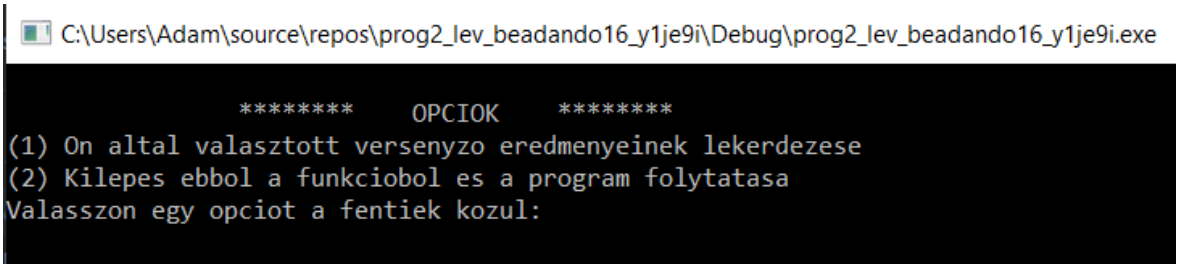
8. ábra: versenyző rajtszámának bekérése



9. ábra: versenyző részletes statisztikáinak kiírása

A rajtszámnak egész számnak kell lennie, és értelemszerűen nem lehet nagyobb, mint a versenyen részt vevők száma. Érvénytelen rajtszám esetén (nem numerikus karakter vagy 1-nél kisebb vagy 100-nál nagyobb szám) a program helyesbítést kér.

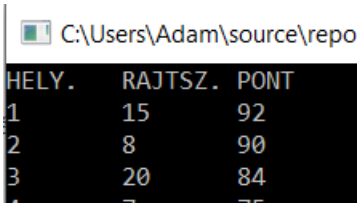
Az első bekért versenyző részletes statisztikáinak kiírása után a program egy menürendszert ír ki a képernyőre. Két lehetőség van: az 1-es opció megadásával az előző lépés ismétlődik, tehát egy újabb versenyző részletes statisztikáit lehet lekérni. A 2-es opció megadásával kilépünk a menürendszerből, és a program folytatja a további futást.



10. ábra: menü további adatlekéréshez vagy folytatáshoz

Ha nem kívánunk további részletes statisztikákat lekérni, és kiléptetjük a programot ebből a menürendszerből, a program futása a pontszámok és helyezések kiszámolásával, majd ezeknek az adatoknak a `sorrend.txt` nevű állományba írásával folytatódik és fejeződik be.

A pontszám szerinti rangsort a program táblázatszerű elrendezésben a képernyőre is kiírja a felhasználó számára.



11. ábra: rangsor kiírása

A program ekkor a végére ér. A képernyőre kiírt adatok továbbra is ott maradnak a felhasználó számára. Egy billentyű lenyomásával bezáródik a parancssor, és a program kilép.

2.3. Hibaüzenetek

Az alábbi táblázat a program által generált hibaüzeneteket listázza és írja le.

6. táblázat: hibaüzenetek és kezelésük

Hibaüzenet	Teendő
`Hiba a fajl megnyitasa soran!`	A `verseny.txt` fájl nem található a program könyvtárában, nem ezen a néven van lementve. Ha a fájl neve és helye is megfelel, a megnyitás során valamilyen más hiba lépett fel.

Szokoz eszlelve az egyik sorban. A szokozok hibas adatbeolvasast eredmenyeznek.	A versenyzők lövéseit leíró plusz- és mínuszjelek között szóközt észlelt a program, amely hibás adatfeldolgozást eredményezhetnek. Nyissa meg a verseny.txt állományt, és manuálisan törölje ki a szóköz karaktereket az állományból!
Ures fajl! Nem feldolgozhato.	A beolvasott állomány üres. A fájl első sorában a résztvevők számának kell szerepelnie, majd a további sorokban a versenyzők lövéseit szimbolizáló + és – jeleknek. Ellenőrizze, helyes fájlt töltött-e be! Töltse be a helyes fájlt, vagy javítsa a meglévőt!
Ervenytelen fajl! A fajl elso soraban szamnak (a versenyzok szamanak) kell szerepelnie.	A fájl első sorában a résztvevők számának kell szerepelnie. Ellenőrizze, helyes fájlt töltött-e be! Töltse be a helyes fájlt, vagy javítsa a meglévőt!
Hibas fajl! A fajl elso soraban levo letszam nem egyezik a fajlban tarolt adatok szamaval.	A fájl első sorában a résztvevők számának kell szerepelnie. Alatta a pontosan annyi + és – jelekből álló sornak kell szerepelnie, ahány versenyző van (mindegyik sorban külön versenyző lövései szerepelnek). Az első sorban lévő számnak és az alatta lévő sorok számának tehát egyeznie kell. Ellenőrizze, helyes fájlt töltött-e be! Töltse be a helyes fájlt, vagy javítsa a meglévőt!
Legalabb 2 es legfeljebb 100 versenyzo lehet a fajlban. A fajl nem feldolgozhato.	Egy versenyen legalább 2, maximum 100 versenyző lehetett. Ellenőrizze, helyes fájlt töltött-e be! Töltse be a helyes fájlt, vagy javítsa a meglévőt!
***n* ervenytelen rekord a fajlban, a fajl nem feldolgozhato!**	A versenyzők lövései csak + és – jeleket tartalmazhatnak. Ha bármilyen más karakter került be ezek közé, a fájl nem feldolgozható. Ellenőrizze, helyes fájlt töltött-e be! Töltse be a helyes fájlt, vagy törölje ki a meglévőben az érvénytelen karaktereket!
Hibas parameter! Kilepes a feladatbol...	Olyan opciót adott meg a menüben, amely nem volt a választható lehetőségek között. A program úgy halad tovább, mintha a „kilép” opciót választotta volna.

Ábrajegyzék

Táblázatok jegyzéke

Forrásjegyzék

[1] C++ Reference, „string::find,” [Online]. Available: http://www.cplusplus.com/reference/string/string/find/. [Hozzáférés dátuma: 2 április 2021].

[2] Stack Overflow, „How can I convert a std::string to int?,” [Online]. Available: https://stackoverflow.com/questions/7663709/how-can-i-convert-a-stdstring-to-int/7664227#7664227. [Hozzáférés dátuma: 2 április 2021].

[3] Techie Delight, „Iterate over characters of a string in C++,” [Online]. Available: https://www.techiedelight.com/iterate-over-characters-string-cpp/. [Hozzáférés dátuma: 2 április 2021].

[4] Stack Overflow, „How to detect empty lines while reading from istream object in C++?,” [Online]. Available: https://stackoverflow.com/questions/9235296/how-to-detect-empty-lines-while-reading-from-istream-object-in-c. [Hozzáférés dátuma: 2 április 2021].

[5] Stack Overflow, „Returning to beginning of file after getline,” [Online]. Available: https://stackoverflow.com/questions/5343173/returning-to-beginning-of-file-after-getline. [Hozzáférés dátuma: 2 április 2021].

[6] C++ Forum, „Counting lines in a txt file,” [Online]. Available: http://www.cplusplus.com/forum/beginner/151963/. [Hozzáférés dátuma: 2 április 2021].

[7] C++ Reference, „string::length,” [Online]. Available: http://www.cplusplus.com/reference/string/string/length/. [Hozzáférés dátuma: 2 április 2021].

[8] Stack Overflow, „How do I find the length of an array?,” [Online]. Available: https://stackoverflow.com/a/4108340/9018796. [Hozzáférés dátuma: 2 április 2021].

[9] C++ Reference, „array::size,” [Online]. Available: https://www.cplusplus.com/reference/array/array/size/. [Hozzáférés dátuma: 2 április 2021].

[10] C++ Reference, „istream::read,” [Online]. Available: http://www.cplusplus.com/reference/istream/istream/read/. [Hozzáférés dátuma: 2 április 2021].

[11] C++ Reference, „string::substr,” [Online]. Available: http://www.cplusplus.com/reference/string/string/substr/. [Hozzáférés dátuma: 15 április 2021].

[12] GeeksforGeeks, „Convert string to char array in C++,” [Online]. Available: https://www.geeksforgeeks.org/convert-string-char-array-cpp/. [Hozzáférés dátuma: 2 április 2021].

[13] C++ Reference, „isdigit,” [Online]. Available: http://www.cplusplus.com/reference/cctype/isdigit/. [Hozzáférés dátuma: 16 április 2021].

[14] B. L. Jones, Sams Teach Yourself C++ in 21 Days, ötödik ed., Sams, 2004.

[15] A. Pásztor, Szerző, *Algoritmusok és adatstruktúrák kurzus előadás anyaga.* [Performance]. Neumann János Egyetem, GAMF Műszaki és Informatikai Kar, 2021.

[16] A. Pásztor, Szerző, *Programozás I-II kurzusok előadás és gyakorlat anyaga.* [Performance]. Neumann János Egyetem, GAMF Műszaki és Informatikai Kar, 2020-2021.

[17] Stack Overflow, „Determine if a string is a number,” [Online]. Available: https://stackoverflow.com/questions/4654636/how-to-determine-if-a-string-is-a-number-with-c/4654718#4654718. [Hozzáférés dátuma: 16 április 2021].