Programozás II. 1. ZH

SZTE Szoftverfejlesztés Tanszék

2022. ősz

Technikai ismertető

- A programot C++ nyelven kell megírni.
- A megoldást a *Bíró* fogja kiértékelni.
  - A Feladat beadása felületen a Feltöltés gomb megnyomása után ki kell várni, amíg lefut a kiértékelés. **Kiértékelés közben nem szabad az oldalt frissíteni vagy a Feltöltés gombot újból megnyomni** különben feltöltési lehetőség veszik el!
- Feltöltés után a *Bíró* a programot `g++` fordítóval és a
  `-std=c++1y -static -O2 -DTEST_BIRO=1`
  paraméterezéssel fordítja és különböző tesztesetekre futtatja.
- A program működése akkor helyes, ha a tesztesetek futása nem tart tovább 5 másodpercnél és hiba nélkül (0 hibakóddal) fejeződik be, valamint a program működése a feladatkiírásnak megfelelő.
- A *Bíró* által a `riport.txt`-ben visszaadott lehetséges hibakódok:
  - Futási hiba 6: Memória- vagy időkorlát túllépés.
  - Futási hiba 8: Lebegőpontos hiba, például nullával való osztás.
  - Futási hiba 11: Memória-hozzáférési probléma, pl. tömb-túlindexelés, null pointer használat.
- A riport.txt és a fordítási log fájlok megtekinthetőek az alábbi módon:
  1. Az Eredmények megtekintése felületen a vizsgálandó próba új lapon való megnyitása
  2. A kapott url formátuma:
     `https://biro.inf.u-szeged.hu/Hallg/IBL302g-1/1/hXXXXXX/4/riport.txt`
  3. Az url-ből visszatörölve a 4-esig (`riport.txt` törlése) megkaphatók a 4-es próbálkozás adatai
- A programot 20 alkalommal lehet benyújtani, a megadott határidőig.
- A programban szerepelhet `main` függvény, amely a pontszámításkor nem lesz figyelembe véve. Azonban ha fordítási hibát okozó kód van benne az egész feladatsor 0 pontos lesz.

Általános követelmények, tudnivalók

- Csak a leírásban szereplő osztályokat, metódusokat és adattagokat kell megvalósítani, egyéb dolgokért nem jár plusz pont.
- Minden metódus, amelyik nem változtatja meg az objektumot, legyen konstans! Ha a paramétert nem változtatja a metódus, akkor a paraméter legyen konstans!
- string összehasonlításoknál az egyezés a pontos egyezést jelenti, azaz ha kis-nagy betűben térnek el, akkor már nem tekinthetők egyenlőnek (pl. a "piros" != "Piros")
- A leírásokban bemutat példákban a string-ek köré rakott idézőjelek nem részei az elvárt kimenetnek, azok csak a string határait jelölik. Például ha az szerepel, hogy a példa bemenetre az elvárt kimenet az, hogy `"3 alma"`, akkor az elvárt kimenet idézőjelek nélkül az `3 alma`, de a szóköz szükséges!
  - A tesztesetekben nem lesz ékezetes szöveg kiíratása.
- Az elvárt kimeneteknek karakterről karakterre olyan formátumúnak kell lennie, ami a feladatban le van írva (szóközöket és sortöréseket is beleértve).

Lokális tesztelés

A minta.zip tartalmaz egy kiindulási feladat.cpp-t, amit a megoldással kiegészítve lokális tesztelésre használhattok. A fordítás az előbbiekben leírt módon történjen. A fájl felépítése a következő.

```
#include <iostream>
#include <string>
#include <cassert>

using namespace std;

//////////////////////////
//Ide dolgozz!!
//////////////////////////

//=== Teszteles bekapcsolasa kikommentezessel
//#define TEST_alma
//=== Teszteles bekapcsolas vege

#if !defined TEST_BIRO
/*
Keszits egy fuggvenyt, ami visszaadja az alma sztringet!
*/
void test_alma(){
  #ifdef TEST_alma && !defined TEST_BIRO
   string s = alma();
   assert(s == "alma");
  #endif
}
int main(){
   test_alma();
```

```
}
#endif
```

Ha megoldottad az alma feladatot, úgy tudod tesztelni, ha kitörlöd a kommentjelet a `#define TEST_alma` sor elől. Ekkor **újrafordítás** után le fog futni a `test_alma()` függvény tartalma is. Ha a visszaadott sztring nem az elvárt, az assert() függvény ezt jelezni fogja. A **define-ok módosítása nem javasolt**, fordítási hibát idézhet elő a biro-n való teszteléskor! **A tesztelőkód nem végez teljes körű tesztelést!** Saját felelősségre bővíthető. A sikeres megoldás után a feladat.cpp tartalma (mely biro-ra is feltölthető):

```
#include <iostream>
#include <string>
#include <cassert>

using namespace std;

string alma(){
  return "alma";
}

//=== Teszteles bekapcsolasa kikommentezessel
#define TEST_alma
//=== Teszteles bekapcsolas vege

/*
Keszits egy fuggvenyt, ami visszaadja az alma sztringet!
*/
void test_alma(){
  #ifdef TEST_alma && !defined TEST_BIRO
  string s = alma();
  assert(s == "alma");
  #endif
}

int main(){
  test_alma();
}
```

1. Feladat: Hiba (6 pont)

Készítsd el az **Hiba** kivétel osztályt, ami publikusan öröklődik az std::exception osztályból. **(1+1 pont)**

A konstruktor egy előjel nélküli számot vár paraméterben, ami alapján beállítja a hibaüzenetet.

- Ha 50-nél nagyobb a paraméter értéke, akkor `"Sulyos hiba tortent"` legyen a hibaüzenet (idézőjelek nélkül),
- különben ha 25-nél nagyobb, akkor `"Komoly hiba tortent"`,
- minden egyéb esetben pedig `"Kis hiba tortent"` a beállítandó szöveg. **(0+2 pont)**

Definiáld felül a what metódust, mely visszaadja a megalkotott üzenetet. **(0+2 pont)**

2. Feladat: Osztály keresése (7 pont)

Valósítsd meg a mintában található `osztalytKeres` függvényt.

A függvény paraméterben egy `map`-et kap, ami egy általános iskola osztályainak a létszámait tárolja (az osztály neve a kulcs, pl.: "9a", a létszáma az érték), valamint egy sztringet, a keresendő osztály nevét.

A függvény első körben járja be a `map`-et **for ciklussal**, és ha valamely osztály **létszáma nagyobb mint 20**, akkor dobjon egy `Hiba` kivételt, amit az osztály létszámával inicializál.

Ezután keressünk a `find_if` fügvénnyel a `map`-ben olyan osztályt, aminek a **neve a második paramétrrel egyezik meg ÉS a létszáma több mint 10**. Ha van ilyen, akkor térjünk vissza igazzal, különben hamissal.

Megjegyzés: A keresésre csak akkor kapsz pontot, ha a find_if metódust használod, és helyesen hívod meg.

3. Feladat: Átlagtalanítás (6 pont)

Valósítsd meg a mintában található `atlagtalanit` függvényt. A függvény első paraméterben egy `vectort` kap referenciaként, amely `int` értékeket tárol.

Számítsd ki a vektor elemeinek az **átlagát** tetszőleges módon. (Feltételezhetjük, hogy egész szám lesz.)

Ezután **távolítsd el** a vektorból azokat az elemeket az `erase` és `remove_if` függvények segítségével, amelyeknek az imént kiszámított **átlaggal való osztási maradéka nulla**.

A metódusnak a paraméterben kapott vektort kell módosítani, így nem kell visszatérni semmivel.

Megjegyzés: A törlésre csak akkor kapsz pontot, ha a remove_if függvényt használod, és helyesen hívod meg.

4. Feladat: Rénszarvas (6 pont)

Egészítsd ki a mintában található `Renszarvas` struktúrát a `<` operátorral, amellyel majd a **sebesség szerint növekvő** sorba lehet rendezni a rénszarvasokat.

Valósítsd meg a mintában található `szantKeszit` függvényt. A függvény a készítendő rénszarvasok számát várja, illetve az elsőnek a sebességét.

Inicializálj egy vektort a **paraméterben kapott mérettel**.

Töltsd fel a vektort rénszarvasokkal a `generate` függvény segítségével. Az **első rénszarvas sebességét a parméterben megkaptuk**. A többi rénszarvas sebességét pedig mindig az **előtte lévőnek a felére állítsd be**. Feltételezhetjük, hogy végig egész számokkal fogunk dolgozni.

Megjegyzés: A vektor feltöltésére csak akkor kapsz pontot, ha a generate függvényt használod.

5. Feladat: Egyedieket rendez (9 pont)

Valósítsd meg a mintában található `egyedieketRendez` metódust!

A függvény egy `int` tömböt kap paraméterben, illetve a tömbnek a hosszát.
Recap: a mintában található `int* tomb` itt ugyan azt jelenti mint az `int tomb[]`, tehát ugyan úgy be tudjuk járni indexeléssel, mint a tömböket.

A függvény feladata, hogy a kapott tömbből az egyedi elemeket csökkenő sorbarendben visszaadja egy vektorban.

Például:
Input: `[2, 5, 5, 0, 2, 3] elemeket tartalmzó tömb és 6`
Output: `[5, 3, 2, 0] elemeket ebben a sorrendben tartalmazó vektor`

Megjegyzés: A függvény megvalósítása során most bármit használhatsz, azonban az órán tanult adatszerkezetekkel és algoritmusokkal jóval egyszerűbb a megoldás.

Hint: Érdemes lehet először az egyedieket kiszűrni, aztán rendezni.
Hint2: Több adatszerkezetet is használhatsz a függvényben.