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();
}
```

Segítsd a szorgalmas manókat az ajándékok kezelésével!

1. Feladat: AjandekHiba (5 pont)

Készítsd el az **AjandekHiba** kivétel osztályt. **(0+1 pont)**

A kivétel példányosításakor az ajándék nevét (std::string) és a tömegét (unsigned) lehessen megadni! Ezek segítségével alkosd meg és tárold el a következő üzenetet:

`"<nev> nevu ajandek <tomeg> grammal meghaladja a maximum tomeget!"`

(0+2 pont)

Ahol a <nev> helyére a példányosításkor nevet kell beírni, <tomeg> helyére pedig az ugyan ekkor megadott tömeget.

HINT: kell egy std::string adattag az üzenet letárolásához!

Definiáld felül a what metódust, mely visszaadja a megalkotott üzenetet.**(1+1 pont)**

HINT: c_str()

2. Feladat: ajandekKatalogus (7 pont)

Valósítsd meg a mintában található ajandekKatalogus függvényt! Az első két paraméter 2 tömb, az első az adott ajándék nevét, míg a második annak súlyát tárolja. A harmadik paraméter a tömbök hosszát jelöli, az utolsó pedig a maximális tömeget, amit tárolni tudnak a manók.

A tömbök által közölt információ alapján készíts egy `std::map` típusú objektumot, amelynek kulcsa szöveg legyen, a kulcshoz tartozó érték, pedig előjeltelen egész szám. Az így készített map-et töltsd fel, úgy, hogy a kulcsai az első tömb elemei legyenek, az ezekhez rendelt értékek, pedig az azonos indexen található egész számok a második tömbből!

Ha a feltöltés során bármelyik egész szám meghaladná a maximumális tömeg értéket dobj egy `AjandekHiba` kivételt. Paraméterként a limitet meghaladó ajándék nevét, és tömegét add meg!

A feltöltés után az `std::find_if` algoritmus segítségével keresd meg az első 7-el oszható tömegű ajándékot. Ha van ilyen, akkor írd ki a nevét a standard kimenetre, és a metódus térjen vissza igazzal. A kiiratást kövesse sortörés! Egyébként ne írj ki semmit, és hamissal térj vissza.

Megjegyzés: A keresésre csak akkor kapsz pontot, ha a find_if metódust helyesen meghívod!

3. Feladat: mazsolatValogat (6 pont)

Valósítsd meg a mintában található mazsolatValogat függvényt! A metódus első paramétere egy Nasik-ból álló tömb, a második pedig ennek a tömbnek a hossza.

A tömb elemeit egyesével szúrd be egy `std::vector`-ba! Ügyelj arra, hogy ugyan abban a sorrendben legyenek, mint a tömbben.

Ezután az `std::remove_if` algoritmus segítségével vedd ki belőle azokat a Nasikat, amelyek neve: "Mazsola".

Fontos: A mazsolákat kötelező hozzáadnod a vektohoz, különben nem kapsz pontot!

A mazsolák eltávolítása után az `std::count_if` algoritmus segítségével számold meg hány olyan Nasi maradt a vektorban, amelynek tipusa: "Edes"! A függvény visszatérési értéke az így kiszámított érték legyen!

4. Feladat: ontoformak (6 pont)

Valósítsd meg a mintában található ontoformak metódust! A metódus első paramétere egy Ontoforma objektumokból álló tömb, a második pedig ennek hossza.

Az Ontoformakat egyesevel helyezd egy `std::set`-be, a formák a méterük szerint legyenek növekvő sorrendbe rendezve! (Egy forma mérete a magasságának és szélességének szorzata.)

Miután elékszítetted a halmazt, írasd ki az elemeinek nevét a standard kimenetre az `std::for_each` algoritmus segítségével. Minden kiírt nevet kövessen sortörés.

5. Feladat: legjobbGyerekek (9 pont)

Valósítsd meg a mintában található legjobbGyerekek metódust! A metódus első paramétere egy Gyerek objektumokból álló std::vector.

A metódus írja ki a 3 legjobb gyerek nevét, jóság szerinti csökkenő sorrendben. Egy gyerek jóságát a `josag` paraméter határozza meg. Minden nevet kövessen sortörés! A visszatérési értéke, pedig legyen az vektorban található gyerekek átlag jóságánál kevésbé jó gyerekek száma!

Megjegyzés: Nincs megkötés a használt eszközökre, de érdemes lehet STL adatszerkezeteket/algoritmusokat használnod!