Programozás II. Gyakorló Feladat

SZTE Szoftverfejlesztés Tanszék

2023. ősz

Ismertető

- A programot C++ nyelven kell megírni.
- **A benyújtandó fájl neve kötelezően `feladat.cpp`.**
- 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://biro2.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.
- A megvalósított függvények semmit se írjanak ki a standard outputra!

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();
}
```

VilagitoDisz

1. feladat (4 pont)

Készíts egy VilagitoDisz nevű osztályt, mely egy karácsonyfa díszítésére alkalmas! Előjel nélküli egészként tároljon el fényesség értéket, logikai értékként pedig azt tárolja el, hogy be van-e kapcsolva. Rendelkezzen egy `get_fenyesseg` metódussal, ami visszaadja a dísz fényesség értékét, ha be van kapcsolva. ha nincs bekapcsolva nullát adjon vissza. Legyen egy `is_bekapcsolva` gettere is.

Készítsd el a VilagitoDisz konstruktorát is, mely a fényességet várja és állítja be. Default konstruktorként (default paraméter érték) a 0 értéket állítsa be!

2. feladat (3 pont)

Definiáld felül a VilagitoDisz pre ++ és pre – – operátorát! A ++ operátor kapcsolja be a világítást, a – – operátor kapcsolja ki azt. Az operátorok pre verzióként működjenek, azaz a módosított értékkel az eredeti objektumot adják vissza!

KisKaracsonyfa

3. feladat (4 pont)

Készíts egy KisKaracsonyfa nevű osztályt, amely egyetlen világító dísz tárolására képes. Nevezzük ezt a díszt csúcsdísznek. A fa a csúcsdíszt nem közvetlenül tárolja, hanem pointerként (`VilagitoDisz*`), mely kezdetben (az objektum konstruálásakor) `nullptr`. Készíts az adattaghoz egy gettert `get_csucs_disz` néven. A fa tárolja el azt is szöveges formátumban, hogy milyen fajta. Pl. `luc`. Legyen ehhez is getter `get_fa_tipus` néven.

Valósítsd meg a KisKaracsonyfa konstruktorát, mely a fa típusát várja! Default paraméter értékként a "luc" értéket add meg!

4. feladat (4 pont)

Írj egy **disz_felhelyezese** metódust, mely egy VilagitoDisz-t vár paraméterben, figyelj annak pontos típusára! A metódus törölje a fán lévő díszt (ha van) és foglaljon dinamikusan egy új díszt, lemásolva a paraméterben kapottat. A metódus ne térjen vissza semmivel.

Tipp: másoló konstruktor hívás.

Mivel az osztályban dinamikusan kezelünk memóriát, gondoskodj a helyes memóriafelszabadításról is az objektum megszűnésekor!

5. feladat (4 pont)

Legyen egy **void bekapcsol()** és egy **void kikapcsol()** metódus, melyek rendre be vagy kikapcsolják a csúcsdíszt, ha van a fán!

6. feladat (2 pont)

Legyen egy **get_fenyesseg** metódus, mely a fa aktuális fényességét határozza meg! Ha nincsen csúcsdísz a fán, adjon vissza nullát, különben a dísz fényességét!

7. feladat (3 pont)

Valósítsd meg a KisKaracsonyfa másoló konstruktorát, mely az adattagokat lemásolja! Figyelj a dinamikusan foglalt adattagok helyes kezelésére!

8. fealdat (4 pont)

Valósítsd meg a értékadás (assignment) operátort! Az operátor kezelje a lehetséges hibákat! Figyelj a dinamikusan foglalt adattagok helyes kezelésére!

NagyKaracsonyfa

9. feladat (4 pont)

Valósítsd meg a NagyKaracsonyfa osztályt! Ez már több világító dísz letárolására képes. A maximálisan tárolható díszek száma a NagyKaracsonyfa konstruktorában érkezik. Tárolja a

díszeket egy dinamikusan inicializált tömbben, mely a `get_diszek` getteren keresztül lekérdezhető.

A dinamikus memória használat miatt ügyelj az objektum életének végekor a megfelelő memória felszabadításra!

10. feladat (2 pont)

Legyen egy void **disz_felhelyezese** metódus, mely a tömb első üres pozíciójára elhelyezi a paraméterben érkező VilagitoDisz-t! Ha nem fér el több dísz a fán, ne tegyen semmit!

11. feladat (4 pont)

Legyen egy **void bekapcsol()** és egy **void kikapcsol()** metódus, melyek rendre be- és kikapcsolják az éppen fán lévő díszeket (mindet)! Csak azokat, melyeket már felraktunk a fára!

12. feladat (2 pont)

Legyen egy unsigned **get_fenyesseg** metódus, mely a fa fényességét adja meg! A fa fényessége az éppen rajta lévő díszek fényességének összege.

13. feladat (4 pont)

Valósítsd meg a NagyKaracsonyfa másoló konstruktorát! Figyelj oda a dinamikus elemek helyes másolására!

14. fealdat (4 pont)

Valósítsd meg az értékadás (assignment) operátort! Figyelj a lehetséges hibákra, és a dinamikus elemek helyes másolására!