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();
}
```

Ügyelj rá, hogy minden olyan metódus konstans legyen, ami nem módosít az adattagok értékein! Ha egy metódus nem változtat a paraméterén, akkor az legyen konstans!

1. feladat: Cica (11 pont)

Készítsd el a `Cica` osztályt, amely egy cicát fog reprezentálni. **(1+0 pont)**

A cica adattagjait az 1. táblázat mutatja be.

Adattag neve	Típusa	Jelentése	Getter neve	Setter neve
szin	string	az cica szine	get_szin	set_szin
aranyos	bool	aranyos-e	is_aranyos	-

1. táblázat. Cica adattagok

Az adattagok csak az osztályból legyenek elérhetőek, de készíts hozzájuk getter és setter metódusokat a fent látható táblázat szerint. **(1+1 pont)**

Készíts az osztályhoz egy default konstruktort, ami a színt "Tarka"-ra, az aranyosságot igazra állítja be. **(0+1 pont)**

Készíts az osztályhoz egy paraméteres konstruktort, amely 2 paramétert vár és inicializálja az adattagokat azok alapján. A konstruktor paramétereinek sorrendje: *szin*, *aranyos*. **(0+2 pont)**

Egy cicát lehessen szöveggé alakítani! A konverzió után a következő formájú szöveg keletkezzen:

`"A␣<szin>␣szinu␣cica,␣ami␣nagyon␣aranyos."`

A <szin> helyére a cica színe kerüljön, a nagyon jelző, pedig az "aranyos" logikai értéknek megfelelően jelenjen meg. (Maradjon ki ha hamis, legyen benne ha igaz) A szöveget ne kövesse sortörés! A kimaradó jelző esetén figyelj a szóközök számára!**(1+2 pont)**

Definiáld felül a ! operátort. Az operátor negálja a cica aranyosságát, és a módosított objektummal térjen vissza! **(1+1 pont)**

2. feladat: Gazdi (10 pont)

Készítsd el a `Gazdi` osztályt, ami egy néhány cica gazdáját reprezentálja.

A gazdi adattagjait a 2. táblázat mutatja be.

Adattag neve	Típusa	Jelentése	Getter neve	Setter
nev	string	a gazdi neve	get_nev	-
haziallatok	Cica[5]	a gazdi cicabefogadó képessége	get_haziallatok	-
haziallatszam	unsigned	a gazdi lakásában leledző cicák száma	get_haziallatszam	-

2. táblázat. Gazda adattagok

Az adattagok csak az osztályból, vagy annak leszármazottaiból legyenek elérhetőek, de készíts hozzájuk getter metódusokat! **(0+1 pont)**

Az osztálynak legyen egy 1 paraméteres konstruktora, ami a gazdi nevét inicializálja. A *haziallatszam* kezdetben nulla. **(0+1 pont)**

Definiáld felül az osztályban a $+=$ operátort. Jobb oldali operandusa egy Cica, amelyet a Gazdi befogad. Ellenőrizd, hogy van-e elég hely a tömbben. Ha nincs, akkor ne legyen hozzáadás és nem kell hibaüzenetet sem kiírni. Ha van, akkor add hozzá a tömb első szabad helyéhez. A metódus visszatérési értéke a módosított objektum referenciája legyen. **(1+3 pont)**

Egy gazdit is lehessen szöveggé konvertálni, ebben az esetben a következő szöveg jöjjön létre:

`"<nev> <haziallatszam> cicat tart, ezek kozul <mennyi> aranyos, ezek szinei: <szinek>"`

A <nev> és <haziallatszam> helyére a gazdi neve, illetve az összesen tartott cicák száma kerüljön. A <mennyi> helyére az aranyos cicák száma kerüljön, amik színei a kettőspont után legyenek felsorolva szóközzel, és vesszővel elválasztva. Figyelj, hogy az utolsó szín után ne legyen se vessző, se szóköz. Egy szín többször is előfordulhat a felsorolásban! A szöveg végén ne legyen írásjel, és ne kövesse sortörés! Példa:

szin:Voros,aranyos:true	szin:Tarka,aranyos:false	szin:Oliva,aranyos:true	szin:Voros,aranyos:true

3. táblázat. A haziallatok tömb tartalma

A 3. táblázatban ábrázolt esetben a kimeneti szöveg a következő:
"Timea 4 cicat tart, ezek kozul 3 aranyos, ezek szinei: Voros, Oliva, Voros"

(1+3 pont)