

Programozás alapjai 2. (inf.) 2. zárthelyi 2015.05.07. gyak./lab. hiányzás: 3/2

IB.028/1.

Minden beadandó megoldását a feladatlagra, a feladat után írja! Készíthet piszkozatot, de csak a feladatlagra írt megoldásokat értékeljük! A megoldások során feltételezheti, hogy minden szükséges input adat az előírt formátumban rendelkezésre áll. A feladatok megoldásához csak a letölthető C, C++ és STL összefoglaló használható. Elektronikus eszköz (pl. tablet, notebook, mobiltelefon) nem használható. A feladatokat **figyelmesen olvassa el**, megoldásukhoz **használhat STL elemeket**, kivéve, ha a feladat ezt külön tiltja!
Ne írjon felesleges függvényeket ill. kódot!
 Az első feladatrészben (beugró) minimum 5 pontot el kell érnie ahhoz, hogy a többit értékeljük.

fel.	max.	elért	jav.
1.	10		
2.	10		
3.	10		
4.	10		
Σ	40		

1. feladat: Beugró. A feladatok megoldásához **használjon STL elemeket!**

Σ 10 pont

a) Írjon C++ programrészletet, amely fájl végéig valós számokat olvas be, egy `std::set` típusú tárolóba! (2p)

b) Írjon olyan függvénysablont, ami a paraméterként kapott tároló minden elemét soronként kiírja a standard kimenetre! Tételezze fel, hogy a tárolóban tárolt generikus adatokhoz létezik megfelelő inserter (`<<`) operátor, és a tárolónak van iterátora. (2p)

c) Egészítse ki az alábbi felsorolást és a tevékenységek sorrendjének megfelelően számozza azokat be! (2p)

A konstruktor az alábbi tevékenységeket hatja végre:

- hívja a tartalmazott objektumok konstruktorait
- hívja a közvetlen nem virtuális alapsztályok konstruktorait,
- az öröklési lánc végén hívja a virtuális alapsztályok konstruktorait
- beállítja a virtuális alapsztály mutatóit
- beállítja a virtuális függvények mutatóit
- végrehajtja a programozott törzset

d) Készítsen generikus függvényobjektumot, ami pontosan úgy használható, mint az `std::less` sablon, de mindig igaz értékkel tér vissza! (2p)

e) Tételezze fel, hogy `list1` és `list2` `std::list` típusú tárolók, melyek nagyság szerint növekvő sorrendbe rendezett valós értékeket tartalmaznak. Egyesítse a két listát úgy, hogy `list1` tartalmazza `list2` elemeit is és maradjon rendezett! (2p)

2. Feladat. A feladat megoldásához **használjon STL elemeket!****10 pont**

Tételezze fel hogy rendelkezésére áll az `std::vector` osztályhoz hasonló tulajdonságokkal rendelkező generikus sorozattároló.

Készítsen olyan generikus adapter osztályt (*MyVector*), ami ehhez a következő publikus hozzáférést biztosítja: 6p

legyen *paraméter nélküli konstruktor*, ami üres tárolót hoz létre;

legyen *iterátoros konstruktor*, ami a tárolót feltölti az iteterátorokkal megadott sorozattal;

legyen *másoló konstruktor*, *értékadó operátora* és *destruktor*;

legyen *írható random access iterátora*, *begin* és *end* tagfüggvénye (az `std::vector` tárolónak is van ilyen iterátora);

legyen *shift* tagfüggvénye, ami minden elemet N hellyel balra tol; N a tagfüggvény paramétere legyen; a belépő elemek a generikus adat paraméter nélkül hívott konstruktorával létrehozott objektumok legyenek; balra tolás azt jelenti, hogy a

sorozat első helyére kerül a második elem, a második helyére a harmadik elem, a harmadik helyére a negyedik elem, stb;

A felsorolt függvények és az implicit tagfüggvények kivételével más hozzáférést az adatper ne tegyen lehetővé!

Mutassa be az adapter használatát egy rövid kódrészlettel, melyben létrehoz egy 6 egész értéket tartalmazó C tömböt, majd az elkészített adapter segítségével létrehoz egy tárolót, amit feltölt a C tömb elemeivel. Ezután írja ki tároló elemeit a standard kimenetre, tolja el az elemeket a tárolóban, majd írja ki újra az elemeket! Végül STL algoritmust felhasználva számolja meg hány darab 0 értékű elem van a tárolóban!

4p

3. Feladat.

10 pont

Adottak a következő deklarációk:

```
struct Serializable {  
    virtual void read(std::istream&) = 0;  
    virtual void write(std::ostream&) = 0;  
};
```

```
class Pont {  
protected:  
    double x, y;  
public:  
    Pont(double x = 0, double y = 0)  
        :x(x), y(y) {}  
};
```

A fenti osztályok felhasználásával, azok módosítása nélkül **hozza létre** a *Pont* osztály perzisztens változatát! A perzisztens változatnak legyen *getter* tagfüggvénye is! Az adatfolyamra úgy írja ki az adatot, hogy visszaolvasáskor ellenőrizni tudja, hogy a várt adatot kapja-e (ha pontot vár, akkor pontot kap-e). Hiba esetén, dobjon *std::out_of_range* kivételt! 5p

Mutassa be egy rövid kódrészlet segítségével a perzisztens működést: Hozzon létre pár példányt a perzisztens osztályból, írja ki azokat egy fájlba vagy stringstream-re, majd olvassa vissza! Mutassa be hibakezelést is! 5p

4. Feladat. A feladat megoldásához **használhat STL elemeket!**

10 pont

A zárthelyik javítását szeretnénk programmal segíteni. A probléma analízise során több objektumot azonosítottunk: zárthelyit, feladatot, és hibákat. A zárthelyikhez (*ZH*) rendelt egyedi attribútum a hallgató kódja (*std:string*). Minden *ZH* több feladatból (*Feladat*) áll. Minden feladatnak van azonosítója (*std::string*), elérhető és elért pontszáma (*int*) és egy hibatárolója, amiben a javítás során megtalált hibákat tároljuk, hogy később kiírathassuk a hibaszövegeket. Minden hibának (*Hiba*) van hibaszövege (*std::string*), ami a hiba létrehozásakor adható meg, és később kiíratható *std::ostream* típusú adatfolyamra.

A javítás megkezdésekor az elért pontszám az elérhető maximális értéket veszi fel. A javítási folyamat során felfedezett hibákat a feladat belső hibatárolójához adjuk. A hiba tárolóba helyezésekor azonnal módosul a pontszám. A különféle hibák különböző módon változtatják a pontszámot. Vannak halálfejes (*Halalfejes*) hibák, melyek a pontszámot nullázzák és vannak kisebb hibák (*Kishiba*) melyek egy adott pontszámot vonnak le. Az elért pontszám azonban nem lehet kisebb, mint nulla. Listázáskor bizonyos halálfejes hibák a hiba szövege után még azt is kiírják, hogy „JAJ”. Ilyen pl. a *HeterogenTaroloNemPointerTarol* hiba. Később további hibákat is definiálunk.

Tervezzén objektummodell a feladat megvalósításához! Segítségül az alábbi kódrészlettel bemutatjuk a javítást:

```
Feladat f("4.", 10); // 4. feladat javításának kezdete max. pont 10
f.add(new Kishiba("Zarojel hiba", 1)); // ez egy -1 pontos hiba
f.add(new Kishiba("ciklus hamarabb áll le", 2)); // ez egy -2 pontos hiba
f.add(new Halalfejes("delete nem pointerre")); // ez egy halálfejes
f.list(std::cout); // kiírjuk az elért/max pontszámot és a hibákat
```

Gondosan tervezze meg a *Feladat* és a *Hiba* kapcsolatát. Válasszon megfelelő módszert a hibák tárolására, listázására és a pontszám módosítására! Ügyeljen arra, hogy új hiba bevezetésével ne kelljen módosítani a meglévő osztályokat! **Rajzolja le** az osztályok kapcsolatát! Az adattagokat, tagfüggvényeket nem kell feltüntetnie, csupán az osztályok nevét írja a dobozokba, és jelölje a kapcsolatokat!

Deklarálja C++ nyelven a *Feladat*, *Hiba*, *Halalfejes* és *Kishiba* osztályokat! A többi osztályt csak az osztálydiagramon kell feltüntetnie. **Valósítsa** meg az *Hiba*, *Halalfejes* és *Kishiba* osztályok összes tagfüggvényét, valamint a *Feladat* osztály azon tagfüggvényeit, amelyek az elért pontok és a hibák listázását, valamint a hiba tárolóba helyezését végzik!

