

# HF Pontozás útmutató

A dokumentáció, a tervek, a skeleton változat és a program inkrementálisan fejlődik, így is kérjük beadni. A leírt pontozástól a konkrét feladatot figyelembe véve, el lehet térni.

## NHF Specifikáció (5p)

A feladat alapfunkciói értelmesen le vannak írva. Tudjuk, hogy mit várhatunk el a programtól és tudjuk, hogy a program mit és milyen formátumban vár el a felhasználtól. Az elvárt formátumok (pl. kezelt fájlknál, bemeneti nyelvnél) le vannak írva. Az esetleges korlátok is világosak. A korlátokhoz kapcsolódóan nagyvonalakban a hibajelzések megjelennek. A leadott dokumentáció formailag is megfelelő: nem kell tarka/barka dokumentáció, de az egyes szakaszok logikusan legyenek tagolva, legyen cím, szerző neve... A specifikációnak nem kell tartalmazni UML diagramot, megvalósítandó algoritmus leírását, csak blackbox-ként tekint a programra. (Kivétel ez alól, ha valamilyen c++ programkönyvtár készül)

Ha specifikáció nem kerül leadásra a NHF1-nél, akkor maximum 2,5 pont adható.

## NHF Terv (5p)

OO elvek mentén történő tervezés. Osztályok helyes megválasztása, azok kapcsolatának helyes leírása. Láthatóságok, felelősségek szétválasztása. Heterogén kollekció alkalmazása, ha lehet. Ha lehet, ne legyen mindent tudó Isten-objektum. Osztályok attribútumainak helyes megválasztása, jelölése. Működés algoritmikus bemutatása (ahol pl. 2 ciklusnál komplexebb logika kell). A felhasználói inputkezelés vagy a fájlok kezelésének leválasztása az üzleti modelltől. Használja a paramétereket, ahol szükséges, nem globális változókba dolgozik. Formai követelmény: lásd specifikáció, továbbá UML diagram, rövid leírás az osztályokról és azok felelősségéről (Nem kell a végleges verzióknak így kinéznie). A memóriakezeléshez tartozó tervezői döntések leírása, ha STL, akkor miért map, miért list... Tipikus hibák: öröklés és tartalmazás keverése, kapcsolat vs. attribútum, vagy egyszerre jelenik meg mindkettő a diagramon, nagyon sok get/set is probléma lehet (arra utal, hogy csak C struct-ként használja az osztályt, saját funkcionalitása nincs). Ahol lehet és érdemes a kód duplikáció miatt, használjon generikus megoldást. Ahol érdemes, használja ki az operátorok megadásának lehetőségét.

Ha a terv nem kerül leadásra a NHF2-nél, akkor maximum 2,5 pont adható.

## NHF Skeleton (5p)

Szerepel-e minden fontosabb osztály, sablon. Látszanak-e a kapcsolatok. Tagfüggvények paraméterezése. Tényleg csak skeleton-e. Nem várunk teljes programot, de nem is fogadjuk el skeletonnak. Helyes hierarchiát épített fel? Jól definiálta az interfészeket? A beadásnak tartalmazni kell egy kezdetleges programot vagy tesztkörnyezetet, ahol bemutatja az osztályok kapcsolatát. Van dokumentáció is.

Ha nem kerül leadásra a NHF3-nál, akkor 0 pont jár.

## NHF4 Végleges (össz. 25p)

A végleges leadásnál is elvárás a megfelelő specifikáció és programozói dokumentáció elkészítése. Nem kell csillogó grafika, nem azt pontozzuk!

### OO elvek (10p)

OO elvek betartása és hatékonyság. Ami kell, az statikus, konstans, referencia, stb. Tagváltozók láthatósága (elsősorban private), getterek/setterek, a setterek validálnak. Nincs indoklás nélküli friend. A belső működés el van rejtve, pl. listánál a listaelemeket nem publikáljuk kívülre. A tanult idiómák használata: pl. iterátorok, ahol lehet. Tárolóknál is elvárás, hogy legyen másolható, értékül adható. Nincs osztálytípusra if, switch, cast ..., nincs a példány típusára vonatkozó lekérdezés, pl. isX, canFly... Kerüljük az enum/int használatát a különböző viselkedések szétválasztásakor. Kerüljük a körkörös függéseket.

Hatékonyság: ha csak lehet elsősorban referenciát vagy pointert vesz át. Nincs hatalmas adat a stack-ben (lokális tömb). Bevezet sablonokat, ahol értelme van. Konverziókat jól használja. Az állapotot nem változtató tagfüggvények konstansok. Ahol lehet konstans paraméterek vannak. A felhasználói inputkezelés vagy a fájlok kezelésének leválasztása az üzleti modelltől. Használja a paramétereket (közvetlen cout-ra írás helyett ostream paraméter), ahol szükséges. Nem globális változókba dolgozik.

Az osztályok felépítése öröklés és tartalmazás szempontjából is átgondolt és helyes. Sok C-s, procedurális megoldással itt csak pár pontot lehet elérni (ha totál C-s a megoldás, akkor az eleve pótbeadás, bár ennek már a tervezéskor ki kellett buknia). Jól használja a virtuális tagfüggvényeket? Megfelelő szinteken vezeti-e be a tagváltozókat? Szerep szerinti osztálynevek, változónevek, metódusnevek. A felelősségek megfelelő elosztása: ne a Game osztály "ettesse" meg a kígyóval a gyümölcsöt.

### Hibakezelés (3p)

Hibakezelés megléte: Kivételek helyes használata. Védekezik-e a hibás felhasználói és egyéb inputok (pl. beolvasott fájl formátumhibái stb.) ellen, és azokat jól kezeli-e. Ne kapja el a kivételt a komponensen belül, ha nem ott kell kezelni. Ez tipikus hiba szokott lenni. Saját kivétel osztálya az exception-ből származik? Megfelelő leszármazottat használ? (Pl. logic\_error és nem csak ő- ő exception-t dob) Ne legyen a main-ben egy mindent elkapó catch.

### Clean Code (3p)

Mennyire szép a kód? Jól vannak-e a komponensek tagolva? Header és CPP fájlok szeparálva, cpp-ben a definíciók? Megfelelő helyen történik az inicializáció? Vannak-e kommentek, melyek leírják a függvények bementét/kimenetét lehetőleg doxygen formátumban. A metódusok törzse max 4-6 sor, ahol több, egymásba ágyazott ciklus van, azt vágjuk szét, és privát metódusba szervezzük ki a belső kódrészt (funkcionális dekompozíció). Kódduplikáció kerülése. Következetesen indentált kód.

## Tesztek (5p)

Használ valamilyen tesztelő eszközt. A tesztek megmozgatják-e az összes komponenst (ebben segít a Jporta covarage tesztje). Kihasználta-e a Jporta adottságait, környezetét (feltölthető teszt fájlok, gtest\_lite, stb)? A tesztek az érdemi funkcionalitást tesztelik, nemcsak a gettereket. Ha nem tud tesztet írni, mert mondjuk minden függvény az inputra ír, nincs paraméter, visszatérési érték, megfelelő publikus interfész, akkor rossz a modell. Figyel a lehetséges belső elágazások lefedésére is. A lehetséges hibakezelést is teszteli.

## Dokumentációk (4p)

Meghallgatta a korábbi javaslatokat, bővebben: Terv és Specifikáció. Igényes-e, teljes-e, inkrementális-e? Tartalmazza-e a visszalépéseket, önrevíziókat, ha volt ilyen? Itt vehető figyelembe a feladat komplexitása is. Ha használ valamilyen 3. fél által készített könyvtárat, akkor azt behivatkozta.

## Tipikus hibák

A programot be kell mutatni és meg kell védeni, plagizált feladat nem elfogadható. Min. 16 pontot kell elérni.

### Halálfejes hibák (nem elfogadható)

- A program nem fordítható és nem futtatható.
- A vállalt funkciók többnyire nincsenek megvalósítva.
- nincs dokumentáció
- Az elvárható funkciók (pl. telefonkönyvnél egy rekord felvétele) nincsenek megvalósítva.
- Memóriakezelési hiba: leak, túlcímzés...
- Nem OO jellegű megoldás.
- Elvárás, hogy a program tetszőlegesen sok adatot képes legyen kezelni

### További hibák

pontlevonások, arányosan a hiba gyakoriságával vagy súlyosságával kapcsolatban több is adható saját hatáskörben

- a bemeneti nyelvnek megfelelő inputot nem tudja kezelni a program (-4p)
- a felhasználói utasításoknak megfelelő használat kivételt eredményez (-4p)
- öröklés és tartalmazás keverése (-10p)
- publikus attribútumok használata (-4p)
- const-ok használatának mellőzése (-2/4p)
- indokolatlan nevek (-3p)
- globális változó használata (-4p)
- makró használata template helyett (-4p)
- a felhasználói input/output bele van kötve az üzleti modellbe (-5p)
- a bekérés karakterenként történik (-4p)
- Isten osztály (-8p)
- paraméterek használatának mellőzése (-4p)
- másolat referencia/pointer helyett (-4p)

- typecheck (cast, switch, isX, canFly) (-5p)
- heterogén tárolók használatának mellőzése (-4p)
- Ha STL konténereket használ, tudja a főbb metódusokat, nem valósítja meg azokat megint (-4p)
- a program kezelése nem logikus (-4p)
- fejlesztői dokumentáció != kód (-4p)
- printscreen kód a doksiban (-4p)
- tömbök használatának mellőzése (-3p)
- ne a fájlokat használja memóriaként: minden iterációban beolvassa a rekordokat a fájlból, épít egy listát, hozzáad egy elemet, aztán megint kimenti (-10p)
- nem a feladatnak megfelelő adatszerkezetet használ (-5p)
- beégetett magic konstansok és literálok, pl. paraméterezés helyett (-4p)
- a C-s FILE és fájlkezelő függvények használata (-5p)
- az öröklés célja nem az adat, hanem a viselkedés újrahasznosítása (-4p)
- létrehoz() és lista\_free() függvények, nem használja a konstruktort/destruktort (-5p)
- a new által adott pointer != NULL vizsgálata (arra utal, hogy malloc-al összekeveri, a new badalloc kivételt dob, ha nem tudja lefoglalni, -4p)

## Pótlás

A pótlási héten a normál bemutatón szerzett pontoknál max. 25%-al magasabb eredmény érhető el. Pl. bemutatón valaki 50%-ot ért el, akkor nem kaphat 62,5%-nál nagyobbat a pótlásin bemutatott javításokkal. Ha nem volt értékelhető, elfogadható megoldás NHF4-nél, akkor maximum 25 pontot szerezhet a pótláson.