rendelkezésre álló idő:18 **perc**

Programozás alapja 2. 1. ellenőrző dolgozat. 2016.03.07. Kurz/**Terem**: G3/	Elért pontszám:
Név: **Neptun:**	

Az első két feladatnál minden bejelölt válasz 1 pont, ha helyes, -1 pont, ha hibás! Az esetleges negatív eredmény is összeadódik a többi feladatra kapott pontokkal. A teljes dolgozat eredménye azonban nem lehet negatív.

1. Jelölje, hogy mely kijelentés(ek) igaz(ak) a C++ nyelvre! (2p)

- ☒ A referencia egy alternatív név.
- ☐ **`inline`** függvényben nem lehet **`for`** utasítás.
- ☐ **`realloc`** helyett **`renew`** utasítást kell használni.
- ☐ Minden program a **`using namespace std;`** utasítással kell, hogy kezdődjön.
- ☐ A konstruktor mindig **`void`** típusú, amit nem kell kiírni.
- ☒ Konstans tagfüggvény nem változtathatja meg az objektum állapotát.

2. Jelölje, hogy mely kijelentés(ek) igaz(ak) az alábbi C++ kódrészletre! (2p)

```
{ struct S {}; delete[] new S[3]; S s1;}
```

- ☒ **`S`** egy osztály.
- ☐ **`S`** egy objektum.
- ☐ **`S`**-nek nincs konstruktora.
- ☐ **`S`** konstruktora 3-szor hívódik meg a kódrészletben.
- ☒ **`S`** konstruktora 4-szer hívódik meg a kódrészletben.
- ☐ **`S`** destruktora 3-szor hívódik meg a kódrészletben.

3. **Tervezzen** egy olyan osztályt (Tav), ami távolságok tárolására alkalmas! A távolság (double) értéke mellett egy szorzót (double) is tárolni kell az osztályban, amit a műveletekben kell figyelembe venni. Az osztálynak (6p)

- legyen paraméter nélkül hívható konstruktora, ami a távolságot 0-ra, a szorzót 1-re állítja;
- legyen 1 paraméterrel hívható konstruktora, ami a távolságot a paraméterként megadott értékre, szorzót pedig 1-re álltja;
- legyen 2 paraméterrel hívható konstruktora, ami mind a távolságot, mind a szorzót beállítja;
- legyen olyan lekérdező függvénye, amivel a távolság felszorzott értéke kiolvasható;
- legyen olyan lekérdező függvénye, amivel a szorzó kiolvasható;
- legyen olyan operátora (+), amivel két távolságot össze lehet adni. Az eredmény egy olyan távolság objektum legyen, amiben a szorzó 1, és a távolság pedig a két távolság (felszorzott) összege.

Valósítsa meg az osztályt C++ nyelven úgy, hogy az osztálynak legyen legalább 1 nem inline tagfüggvénye! Ne lehessen az adattagokhoz kívülről közvetlenül hozzáférni. Az összeadás operator dobjon **const double** hibát, ha a távolság kisebb, mint 1!
Működjön helyesen az alábbi kódrészlet:

```
Tav t0, t1m(1), t1km(1, 1000);                    // 0m, 1m, 1km
std::cout << t1km.getTav() << std::endl;          // Kiír: 1000
std::cout << (t1m + t1km).getTav() << std::endl;  // Kiír: 1001
```

Egy lehetséges megoldás:

```
class Tav {
    double tavolsag;
    double szorzo;
public:
    Tav(double tavolsag=0, double szorzo=1) :tavolsag(tavolsag), szorzo(szorzo) {}
    double getTav() const { return tavolsag * szorzo; }
    double getSzorzo() const { return szorzo; }
    Tav operator+(const Tav&) const;
};

Tav Tav::operator+(const Tav& rhs) const {
    double tmp = getTav() + rhs.getTav();
    if ( tmp < 0 ) throw 3.14;
    return Tav(tmp, 1);
}
```

BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB rendelkezésre álló idő:18 **perc**

Programozás alapja 2. 1. ellenőrző dolgozat. 2016.03.07. Kurz/**Terem**: G3/	Elért pontszám:
Név: **Neptun:**	

Az első két feladatnál minden bejelölt válasz 1 pont, ha helyes, -1 pont, ha hibás! Az esetleges negatív eredmény is összeadódik a többi feladatra kapott pontokkal. A teljes dolgozat eredménye azonban nem lehet negatív.

1. Jelölje, hogy mely kijelentés(ek) igaz(ak) a C++ nyelvre! (2p)

- ☐ Egy osztálynak több destruktora is lehet.
- ☐ A **new** képes kivételt generálni.
- ☐ **inline** függvényben nem lehet ciklus.
- ☐ A **new** által allokált hely felszabadítható **free**-vel is.
- ☐ Minden osztálynak van konstruktora.
- ☐ A **private** kulcsszó osztályon belül csak egyszer szerepelhet.

2. Jelölje, hogy mely kijelentés(ek) igaz(ak) az alábbi C++ kódrészletre! (2p)

```
{ class C {}; delete[] new C[3]; C c1;}
```

- ☐ **c1** egy objektum.
- ☐ **C** deklarációja hibás, mert nincs adattagja.
- ☐ **C**-nek nincs destruktora.
- ☐ **C** konstruktora 3-szor hívódik meg a kódrészletben.
- ☐ **C** konstruktora 4-szer hívódik meg a kódrészletben.
- ☐ A kódrészletben memóriaszivárgás lép fel.

3. **Tervezzen** egy olyan osztályt (Ar), ami árak eltérő pénznemben történő tárolására alkalmas! Az ár (double) értéke mellett egy árfolyamot (double) is tárolni kell az osztályban, amit a műveletekben figyelembe kell venni. Az osztálynak (6p)

- legyen paraméter nélkül hívható konstruktora, ami az árat 0-ra, az árfolyamot 1-re állítja;
- legyen 1 paraméterrel hívható konstruktora, az árat a paraméterként megadott értékre. az árfolyamot pedig 1-re álltja;
- legyen 2 paraméterrel hívható konstruktora, ami mind az árat, mind az árfolyamot beállítja;
- legyen olyan lekérdező függvénye, amivel az ár Forintban kiolvasható (ár * árfolyam);
- legyen olyan lekérdező függvénye, amivel az árfolyam kiolvasható;
- legyen olyan operátora (-), amivel két árat ki lehet vonni. Az eredmény egy olyan ár objektum legyen, amiben a szorzó 1 és az ár pedig a két ár árfolyamhelyes különbsége.

Valósítsa meg az osztályt C++ nyelven úgy, hogy az osztálynak legyen legalább 1 nem inline tagfüggvénye! Ne lehessen az adattagokhoz kívülről közvetlenül hozzáférni. A kivonást elvégző operátor kivételként dobja el a kivonás eredményét (double), ha az negatív!

Működjön helyesen az alábbi kódrészlet:

```
Ar a0, a1Ft(1), a1Eur (1, 310);                    // 0Ft,.1Ft, 1EUR
std::cout << a1Eur.getAr() << std::endl;           // Kiír: 310
std::cout << (a1Eur - a1Ft).getAr() << std::endl;  // Kiír: 309
```

Egy lehetséges megoldás:

```
class Ar {
    double ar;
    double arfolyam;
public:
    Ar(double ar = 0, double arfolyam = 1) :ar(ar), arfolyam(arfolyam) {}
    double getAr() const { return ar * arfolyam; }
    double getArfolyam() const { return arfolyam; }
    Ar operator-(const Ar&) const;
};

Ar Ar::operator-(const Ar& rhs) const {
    double tmp = getAr() - rhs.getAr();
    if ( tmp < 0 ) throw tmp;
    return Ar(tmp, 1);
}
```