rendelkezésre álló idő:18 **perc**

| Programozás alapja 2. 1. ellenőrző dolgozat. 2016.03.11. Kurz/**Terem**: G5/ | 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 **`while`** feltételében is lehet változót deklarálni.
   ☐ A **`delete`** null pointer esetén kivételt dob.
   ☐ Van külön logikai típus.
   ☐ Nincs makró, helyette **`inline`** van.
   ☐ Mindig meg kell írni az értékadó operátort.
   ☐ Minden programot **`using namespace std;`** utasítással kell kezdeni

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

```
{ struct S {}; S* s1 = new S; S s2(1); }
```

   ☐ **`s1`** egy osztály.
   ☐ **`s1`** egy objektum.
   ☐ **`*s1`** egy objektum.
   ☐ **`s2`** deklarációja hibás, mert nincs 1 paraméteres konstruktora.
   ☐ **`S`** nem osztály, hanem struktúra.
   ☐ **`S`**-nek nincs címképző operátora.

3. **Tervezzen** egy osztályt (Uveg), ami fél literes sörösüveg modellezésére alkalmas! (6p)

   Az osztályban tárolni akarjuk, hogy *mennyi* sör van benne és azt, hogy *kinyitották*-e már. Az osztályból mindig lezárt, tele üvegek jönnek létre. Üveget felnyitni az **`open`** tagfüggvényével lehet. Felnyitott üvegből a **`>>`** operatorral lehet kiönteni, mégpedig a jobboldali operandusnak megfelelő mennyiséget (liter, double). Ha többet akarunk kiönteni, mint amennyi benne van, akkor az üveg kiürül. Bontatlan üvegből nem lehet kiönteni, azaz az operátor nem változtatja a benne levő mennyiséget.
   **Valósítsa** meg az osztályt C++ nyelven úgy, hogy az osztálynak legyen legalább 1 nem inline tagfüggvénye! Ne lehessen az adattagokhoz kívülről közvetlenül hozzáférni. Adatai legyenek lekérdezhetőek. Készítsen egy inserter operatort is, amivel egy **`ostream`** típusú objektumra kiírható az üvegben levő mennyiség, és az hogy felnyitották-e!
   **Működjön** helyesen az alábbi kódrészlet:

```
Uveg sorom;
std::cout << sorom << std::endl;          // Kiír: Bontatlan, 0.5l
sorom.open();
sorom >> 0.3;
std::cout << sorom << std::endl;          // Kiír: Bontott, 0.2l
sorom >> sorom.getMennyiseg()             // kiöntjük a maradékot
```

**Egy lehetséges megoldás:**

```
Uveg {
    double mennyiseg;
    bool bontott;
public:
    Uveg() :mennyiseg(0.5), bontott(false) {}
    void open() { bontott = true; }
    double getMennyiseg() const { return mennyiseg; }
    bool getBontott() const { return bontott; }
    Uveg& operator>>(double);
};
Uveg& Uveg::operator>>(double v) {
    if (bontott) {
        mennyiseg -= v;
        if (mennyiseg < 0) mennyiseg = 0;
    }
    return *this;
}
ostream& operator<<(ostream &os, const Uveg& u) {
    os << (u.getBontott() ? "Bontott, " : "Bontatlan, ") << u.getMennyiseg() << 'l';
    return os;
}
```

BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB rendelkezésre álló idő:18 **perc**

| Programozás alapja 2. 1. ellenőrző dolgozat. 2016.03.11. Kurz/**Terem**: G5/ | 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 **realloc** helyett **`catch`** uatsítást kell használni.
   ☐ Az inicializálás nem azonos az értékadással.
   ☐ A **`new`** által allokált hely felszabadítható **`free`**-vel is.
   ☐ A destruktor mindig **`void*`** paramétert kap, amit nem kell jelölni.
   ☐ Default paraméter csak a paraméter lista végén lehet.

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

```
{ class C {}; C c1; new C();}
```

   ☐ **`C`** egy objektum.
   ☐ **`c1`** deklarációja hibás, mert C-nek nincs megfelelő konstruktora.
   ☐ **`C`**-nek nincs másoló konstruktora.
   ☐ **`C`** konstruktora 2-szer hívódik meg a kódrészletben.
   ☐ **`C`** konstruktora 6-szor hívódik meg a kódrészletben.
   ☐ A kódrészletben memóriaszivárgás lép fel.

3. **Tervezzen** egy osztályt (Zsák), ami 20 literes szemeteszsákok modellezésére alkalmas! (6p)

   Az osztályban tárolni akarjuk, hogy *mennyi* szemét van benne és azt, hogy *nyitott, vagy zárt*. Az osztályból mindig nyitott üres zsákok jönnek létre. Egy zsákot lezárni a **`close`** tagfüggvényével lehet. Felnyitott zsákba a **`<<`** operátorral lehet szemetet tölteni, mégpedig a jobboldali operandusnak megfelelő mennyiséget (liter, double). Ha többet akarunk beletölteni, mint amennyi belefér, vagy le van zárva akkor **const char*** kivétel keletkezik.

   **Valósítsa** meg az osztályt C++ nyelven úgy, hogy az osztálynak legyen legalább 1 nem inline tagfüggvénye! Ne lehessen az adattagokhoz kívülről közvetlenül hozzáférni. Adatai legyenek lekérdezhetőek. Készítsen egy inserter operatort is, amivel egy **`ostream`** típusú objektumra kiírható a zsákban levő mennyiség, és az hogy zárt, vagy nyitott.

   **Működjön** helyesen az alábbi kódrészlet:

```
Zsak zsakom;
std::cout << zsakom << std::endl;          // Kiír: Nyitott, 0l
zsakom << 6;
std::cout << zsakom << std::endl;          // Kiír: Nyitott, 6l
zsakom << 20 - zsakom.getMennyiseg();      // Teletömjük
zsakom.close();
std::cout << zsakom << std::endl;          // Kiír: Zart, 20l
```

**Egy lehetséges megoldás:**

```
class Zsak {
    double mennyiseg;
    bool zart;
public:
    Zsak() :mennyiseg(0), zart(false) {}
    void close() { zart = true; }
    double getMennyiseg() const { return mennyiseg; }
    bool getZart() const { return zart; }
    Zsak& operator<<(double);
};
Zsak& Zsak::operator<<(double v) {
    if (zart || mennyiseg + v > 20) throw "Tele a zsakom";
    mennyiseg += v;
    return *this;
}
ostream& operator<<(ostream &os, const Zsak& u) {
    os << (u.getZart() ? "Zart, " : "Nyitott, ")<< u.getMennyiseg() << 'l';
    return os;
}
```