L01: csütörtök 10-12 rendelkezésre álló idő:15 **perc**

Programozás alapja 2. 3. beszámoló dolgozat. 2018.04.19. Kurz/**Terem**: L01/	Elért pontszám:
Név: **Neptun:**	

Labor e.d. (beugró): első feladatból min 2 pont.

1. Jelölje (pl. karikázza be), hogy az állítás igaz (I) vagy hamis (H) a C++ nyelvre! 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. (4p)

Az implicit értékadó operator nem hívja meg az ősosztály értékadó operátorát, ezért örökléskor mindig kell értékadó operátort deklarálni.	I	**H**
Statikus tagfüggvénynek nincs this pointere.	**I**	H
A destruktor mindig meghívja a tartalmazott objektumok destuktorait.	**I**	H
A konstruktor előbb hajtja végre a programozott törzset, és csak ezután hívja az ősosztályok konstruktorát..	I	**H**

2. A *Szamlalo* osztály egy szabadon futó számlálót modellez. A *clock* tagfüggvény minden meghívásakor a számláló eggyel növekszik. Az osztály felhasználásával, annak módosítása nélkül hozzon létre egy *Nszaml* osztályt, ami a *Szamlalo* osztályhoz hasonlóan működik, de a konstruktorában megadott érték (*N*) elérésénél elölről kezdi a számlálást! Pl. *N=3* esetén *cnt 0, 1, 2, 0, 1, 2, 0, 1, 2, 0* ..... értékeket vesz fel. (6p)

```
class Szamlalo {
    int cnt;
public:
    Szamlalo() :cnt(0)     { }
    int value() const      { return cnt; }
    void value(int v)      { cnt = v; }
    virtual void clock()   { cnt++; }
    virtual ~Szamlalo()    { }
};
```

A *Nszaml* osztályt úgy valósítsa meg, hogy az kompatibilis legyen a *Szamlalo* osztállyal! Valósítson meg minden olyan osztályt és függvényt, ami az alábbi kódrészlet működéséhez kell! Az insertert se felejtse el! A komment segít megérteni az elvárt működést és az elvárt kimenetet.

```
std::cout << Szamlalo() << std::endl;// Kiír: 0
Nszaml s1(10);                       // Létrejött egy tízes számláló
std::cout << s1 << std::endl;        // A számláló lekérdezése. Kiír: 0
s1.clock();                          // Növeljük a számlálót
std::cout << s1 << std::endl;        // Számláló növekedett. Kiír: 1
s1.value(9);                         // Kilenc értékűre állítjuk a számlálót
s1.clock();                          // A számláló „átfordul”
std::cout << s1 << std::endl;        // Kiír: 0
```

```
std::ostream& operator<<(std::ostream& os, const Szamlalo& sz) {
    os << sz.value();
    return os;
}
class Nszaml : public Szamlalo {
    int limit;
public:
    Nszaml (int limit) :limit(limit) {}
    void clock() {
        Szamlalo::clock();
        if (value() >= limit)
            value(0);
    }
};
```

L02: csütörtök 15-17 rendelkezésre álló idő:15 **perc**

Programozás alapja 2. 3. beszámoló dolgozat. 2018.04.19. Kurz/**Terem**: L02/	Elért pontszám:
Név: **Neptun:**	

Labor e.d. (beugró): első feladatból min 2 pont.

1. Jelölje (pl. karikázza be), hogy az állítás igaz (I) vagy hamis (H) a C++ nyelvre! 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. (4p)

Az implicit másoló konstruktor nem hívja meg az alaposztály másoló konstruktorát, ezért örökléskor mindig kell másoló konstruktort deklarálni.	I	**H**
Statikus tagfüggvénynek objektum megadása nélkül is hívhatók.	**I**	H
A destruktor mindig meghívja az ősosztály destruktorát.	**I**	H
Az egyparaméteres konstruktor konverziós operátorként is képes viselkedni.	**I**	H

2. Az *Ember* osztály felhasználásával, annak módosítása nélkül hozzon létre egy *KovetkezoEmber* osztályt, melynek segítségével váróteremben kialakuló sort lehet modellezni! Amikor egy ember belép egy nem üres váróterembe, akkor Ő lesz a *KovetkezoEmber*, aki a sor végére állva megjegyzi, hogy ki vár előtte. A *KovetkezoEmber* osztály a konstruktorban kapja meg az előtte álló objektumot, ami vagy Ember vagy *KovetkezoEmber* típusú. (6p)

```
class Ember {
    String nev;
public:
    Ember(String nev) :nev(nev)    { }
    virtual void print(std::ostream& os) const { os << nev; }
    virtual ~Ember()   { }
};
```

A *KovetkezoEmber* osztályt úgy valósítsa meg, hogy az kompatibilis legyen az *Ember* osztállyal! Valósítson meg minden olyan osztályt és függvényt, ami az alábbi kódrészlet működéséhez kell! A komment segít megérteni az elvárt működést és az elvárt kimenetet. Az insertert se felejtse el! (Tételezze fel, hogy a *String* osztály létezik, és az elvárt módon működik!)

```
{
    using std::cout;
    using std::endl;
    Ember v("Vektor");                          // v létrejön
    cout << v << endl;                          // Kiír: Vektor
    KovetkezoEmber s("a Soros", v);             // Létrejon s, aki előtt v áll
    KovetkezoEmber naki("Ki", s);               // Létrejön naki, aki előtt s áll
    cout << naki;                               // Kiír: Ki a Soros Vektor
}
std::ostream& operator<<(std::ostream& os, const Ember& e) {
    e.print(os);
    return os;
}

class KovetkezoEmber : public Ember {
    const Ember& kov;
public:
    KovetkezoEmber(String nev, const Ember& e) : Ember(nev), kov(e) {}
    void print(std::ostream& os) const {
        Ember::print(os);
        os << " ";
        kov.print(os);
    }
};
```

L03: péntek 10-12 rendelkezésre álló idő:15 **perc**

Programozás alapja 2. 3. ellenőrző dolgozat. 2018.034.20. Kurz/**Terem**: L03/	Elért pontszám:
Név: **Neptun:**	

Labor e.d. (beugró): első feladatból min 2 pont.

1. Jelölje (pl. karikázza be), hogy az állítás igaz (I) vagy hamis (H) a C++ nyelvre! 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. (4p)

Az implicit másoló konstruktor mindig meghívja az alaposztály másoló konstruktorát, ezért örökléskor sosem kell másoló konstruktort deklarálni.	I	**H**
Statikus tagfüggvény nem lehet inline.	I	**H**
A destruktor sosem hívja meg az ősosztály destruktorát, ezért explicit módon kell hívni a származtatottból.	I	**H**
Többszörös öröklés esetén mindig virtuális öröklést kell alkalmazni.	I	**H**

2. A *Hello* osztály *greetings(String s)* metódusa *„Hello ”+s* formában ad vissza minden paraméterként kapott sztringet. Az osztály felhasználásával, annak módosítása nélkül hozzon létre egy *Hello2* osztályt, melynek *greetings(String s)* metódusa a konstruktorában megadott sztring mögé fűzi a paraméterként kapott sztringet! (6p)

```
struct Hello {
    virtual String greetings(String s) const { return "Hello " + s; }
    virtual ~Hello()                          { }
};
```

A *Hello2* osztályt úgy valósítsa meg, hogy az kompatibilis legyen a *Hello* osztállyal! Valósítson meg minden olyan osztályt és függvényt, ami az alábbi kódrészlet működéséhez kell! Az insertert se felejtse el! A komment segít megérteni az elvárt működést és az elvárt kimenetet. (Tételezze fel, hogy a *String* osztály létezik, és az elvárt módon működik!)

```
    using std::cout;
    using std::endl;
    Hello h1;
    cout << h1.greetings("World") << endl;      // Kiír: Hello World
    Hello2 h2("Bye");                           // Létrejött egy új objektum
    cout << h2.greetings("Alex") << endl;       // Kiír: Bye Alex
    cout << h1 << "inserter" << endl;           // Kiír: Hello inserter
    cout << h2 << "bye" << endl;                // Kiír: Bye bye
```

```
std::ostream& operator<<(std::ostream& os, const Hello& sz) {
    os << sz.greetings("");
    return os;
}

class Hello2 : public Hello {
    String prefix;
public:
    Hello2(String s) :prefix(s) {}
    String greetings(String s) const { return prefix + " " + s; }
};
```

L04: péntek 12-14 rendelkezésre álló idő:15 **perc**

Programozás alapja 2. 3. ellenőrző dolgozat. 2018.04.20. Kurz/**Terem**: L04/	Elért pontszám:
Név: **Neptun:**	

Labor e.d. (beugró): első feladatból min 2 pont.

1. Jelölje (pl. karikázza be), hogy az állítás igaz (I) vagy hamis (H) a C++ nyelvre! 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. (4p)

Az implicit másoló konstruktor mindig meghívja alaposztály másoló konstruktorát.	**I**	H
Virtuális alaposztály konstruktora az öröklési lánc elején hívódik.	I	**H**
A destruktor nem lehet virtuális.	I	**H**
A tartalmazott objektum konstruktora a konstruktor programozott törzsének lefutása után fut le.	I	**H**

2. A *Szumma* osztály felhasználásával, annak módosítása nélkül hozzon létre egy *Atlag* osztályt, ami valós mennyiségek átlagszámításához használható! Az osztály tartsa nyilván, hogy hányszor hívták az *operator+=* metódusát és a *getVal* metódus pedig képezze az átlagot! (6p)

```
class Szumma {
    double sum;
public:
    Szumma(double d = 0) :sum(d)     { }
    virtual double getVal() const    { return sum; }
    virtual void operator+=(double d)  { sum += d; }
};
```

Az *Atlag* osztályt úgy valósítsa meg, hogy az kompatibilis legyen a *Szumma* osztállyal! Valósítson meg minden olyan osztályt és függvényt, ami az alábbi kódrészlet működéséhez kell! Amennyiben nullával kellene osztani, úgy dobjon kivételt. A komment segít megérteni az elvárt működést és az elvárt kimenetet.

```
    using std::cout;
    using std::endl;
    Szumma s1;                          // létrejön az s1 0 kezdőértékkel
    s1 += 3;                            // hozzáadunk 3-at
    cout << s1.getVal() << endl;        // Kiír: 3
    Atlag a1;                           // létrejön a1 sum=0, n=0 kezdőértékkel
    a1 += 1;                            // növeljük az összeget és a darabszámot
    a1 += 10;                           // növeljük az összeget és a darabszámot
    cout << a1.getVal() << endl;        // Kiír: 5.5
```

```
class Atlag : public Szumma {
    int n;
public:
    Atlag() : n(0) {}
    void operator+=(double d) {
        n++;
        Szumma::operator+=(d);
    }
    double getVal() {
    if (n == 0) throw "Osztas nullaval"
    return Szumma::getVal() / n;
    }
};
```