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

Programozás alapja 2. 2. beszámoló dolgozat. 2018.03.22. Kurz/**Terem**: L01/	Elért pontszám:
Név: **Neptun:**	

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

Tételezze fel, hogy a **Komplex** osztály létezik, tökéletesen működik, és mindenben megfelel a feladat elvárásainak. Az alábbi osztályt komplex számok tárolására készítettük:

```
class Szamok {
    unsigned int siz;
    Komplex *p;
public:
    Szamok(unsigned s = 0) :siz(s)          { p = new Komplex[siz]; }
    unsigned int size()const                { return siz; }
    Komplex& operator[](unsigned i)         { return p[i]; }
    ~Szamok() { delete[] p; }
    Szamok& operator=(const Szamok& t) {
      if (this != &t) {
        delete[] p;
        siz = t.siz;
        p = new Komplex[siz];
        for (unsigned i = 0; i < siz; i++)
            p[i] = t.p[i];
      }
      return *this;
    }
};
```

1. Mi a hiba az alábbi kódrészletben? Adja meg a hiba jellegét is (fordítási vagy futási)! (1p)

```
{ Szamok t1; Szamok const t2; std::cout << t2.size();}
```

A size tagfüggvény nem konstans, így az konstans objektumra nem alkalmazható. Ez fordítási hibát okoz.

Módosítsa az osztályt, hogy ne legyen hiba! Használja az **üres helyeket** az osztály deklarációjában! (1p)

2. Magyarázza meg, hogy az alábbi függvény futtatásakor miért keletkezik memóriakezelési hiba! (1p)

```
Komplex osszeg(const  Szamok&       t) {
    Komplex sum = 0;
    for (unsigned i = 0; i < t.size(); i++) sum += t[i];
    return sum;
}
```

Értékparaméter másoló konstruktorral másolódik be a verembe. Az alapértelmezett másoló konstruktor azonban nem jó, mivel az osztály dinamikus memóriaterületet foglal és tart nyilván.

Módosítsa a fenti **osszeg** nevű függvény **paraméterét**, hogy ne jelentkezzen hiba és a függvény ugyanúgy legyen használható! (1p)

A paramétert értékparaméter helyett (const) referenciával kell átadni. A lényeg a referencián van, így nem hívódik a másoló konstruktor. .

3. Magyarázza meg, hogy az alábbi kódrészlet futtatásakor miért keletkezik memóriakezelési hiba! (1p)

```
{ Szamok t1(3); t1 = t1 = Szamok(5); }
```

Az alapértelmezett operator= nem jó, mivel az osztály dinamikus memóriaterületet foglal és tart nyilván.

Módosítsa az osztályt, hogy a fenti kódrészlet futtatása ne okozzon hibát, és az általános elvárásoknak megfelelően működjön! Használja az **üres helyeket** az osztály deklarációjában! (5p)

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

Programozás alapja 2. 2. beszámoló dolgozat. 2018.03.22. Kurz/**Terem**: L02/	Elért pontszám:
Név: **Neptun:**	

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

Tételezze fel, hogy a String osztály létezik, tökéletesen működik, és mindenben megfelel a feladat elvárásainak. Az alábbi osztályt sztringek tárolására készítettük:

```
class StringTomb {
    unsigned int siz;
    String *p;
public:
    StringTomb(unsigned s = 0) :siz(s)      { p = new String[siz]; }
    unsigned int size()                     { return siz; }
    String&      operator[](unsigned i)     { return p[i]; }
    ~StringTomb() { delete[] p; }
    StringTomb& operator=(const StringTomb& t) {
      if (this != &t) {
        delete[] p;
        siz = t.siz;
        p = new String[siz];
        for (unsigned i = 0; i < siz; i++) p[i] = t.p[i];
      }
      return *this;
    }

};
```

1. Mi a hiba az alábbi kódrészletben? Adja meg a hiba jellegét is (fordítási vagy futási)! (1p)

```
{ StringTomb t1[10]; StringTomb t2(3); t2[1] = String("Hi"); }
```

Az operator[] nem referencia értékű, így bal oldalon használva hibát okoz.
Elfogadtuk a fordítási hibát is, de valójában egy ideiglenes objektum keletkezik, aminek értéket adhatunk, de az hamarosan megszűnik, tehát szemantikailag értelmetlen.

Módosítsa az osztályt, hogy ne legyen hiba! Használja az **üres helyeket** az osztály deklarációjában! (1p)

2. Magyarázza meg, hogy az alábbi függvény futtatásakor miért keletkezik memóriakezelési hiba! (1p)

```
int darab(const   StringTomb&      t1, StringTomb& t2) {
    return t1.size() + t2.size();
}
```

Értékparaméter másoló konstruktorral másolódik be a verembe. Az alapértelmezett másoló konstruktor azonban nem jó, mivel az osztály dinamikus memóriaterületet foglal és tart nyilván.

Módosítsa a fenti **darab** nevű függvény **paramétereit**, hogy ne jelentkezzen hiba és a függvény ugyanúgy legyen használható! (1p)

A paramétereket értékparaméter helyett (const) referenciával kell átadni. A lényeg a referencián van, így nem hívódik a másoló konstruktor. .

3. Magyarázza meg, hogy az alábbi kódrészlet futtatásakor miért lép fel memóriakezelési hiba! (1p)

```
{ StringTomb t1(3), t2(3); t2 = t1 = t1; }
```

Az alapértelmezett értékadó operátor nem jó, mivel az osztály dinamikus memóriaterületet foglal és tart nyilván.

Módosítsa az osztályt, hogy a fenti kódrészlet futtatása ne okozzon hibát, és az általános elvárásoknak megfelelően működjön! Használja az **üres helyeket** az osztály deklarációjában! (5p)

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

Programozás alapja 2. 2. ellenőrző dolgozat. 2018.03.23. Kurz/**Terem**: L03/	Elért pontszám:
Név: **Neptun:**	

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

Az alábbi osztályt tetszőleges hosszúságú karaktersorozat tárolására készítettük

```
class Duma {
    char *p;
public:
    Duma(const char *s = "")    { p = new char[strlen(s)+1]; strcpy(p, s); }
    const char* c_str() const   { return p; }
    unsigned int size() const   { strlen(p); }
    ~Duma() { delete[] p; }
    Duma(const Duma& d) {
        p = new char[strlen(d.p)+1];
        strcpy(p, d.p);
    }
};
```

1. Magyarázza meg, hogy az alábbi kódrészlet futtatásakor miért lép fel memóriakezelési hiba! (1p)

```
{ Duma s("Hello"); std::cout << s.c_str() << std::endl; }
```

Az implicit deklarált (default) destruktor nem jó, mivel az osztály dinamikus memóriaterületet foglal és tart nyilván.

Módosítsa az osztályt, hogy a fenti kódrészlet futtatása ne okozzon hibát! Használja az **üres helyeket** az osztály deklarációjában! (1p)

Mit ír a szabványos kimenetre a fenti kódrészlet? Hello (1p)

2. Milyen további problémára kell számítani, ha készítünk, és használunk egy **Duma** visszatérési értékű függvényt (pl. **Duma fv();**)? Mi a probléma forrása? (1p)

Az objektum értékű függvény a visszatéréskor meghívja az objektum másoló konstruktorát. Az implicit deklarált másoló konstruktor azonban nem jó, mivel az osztály dinamikus memóriaterületet foglal és tart nyilván.

3. Módosítsa az osztályt, hogy ne lépjen fel az előző feladatban jelentkező probléma, és az osztály az elvárásoknak megfelelően működjön! Használja az **üres helyeket** az osztály deklarációjában! (3p)
4. Készítsen olyan **inserter** operátort, amivel egy **Duma** osztályból származó objektum kiírható egy **std::ostream** típusú objektumra. Működjön az elvárásoknak megfelelően az alábbi kódrészlet! (3p)

```
{ Duma s1("C++"); std::cerr << s1 << std::endl; }
```

```
std::ostream& operator<<(std::ostream& os, const Duma& s) {
    return os << s.c_str();
}
```

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

Programozás alapja 2. 2. ellenőrző dolgozat. 2018.03.23. Kurz/**Terem**: L04/	Elért pontszám:
Név: **Neptun:**	

Az alábbi osztályt tetszőleges hosszúságú karaktersorozat tárolására készítettük:

```
class CharSeq {
    char *seq;
public:
    CharSeq() { seq = new char[1]; seq[0] = '\0';}
    CharSeq& operator+=(char ch) {
        int len = strlen(seq);
        char *tmp = new char[len+2];
        strcpy(tmp, seq); tmp[len] = ch; tmp[len+1] = '\0';
        delete[] seq;
        seq = tmp;
        return *this;
    }
    const char* c_str() const { return seq; }
    ~CharSeq() { delete[] seq; }
    CharSeq(const CharSeq& d) {
        seq = new char[strlen(d.seq)+1];
        strcpy(seq, d.seq);
    }


};
```

1. Magyarázza meg, hogy az alábbi kódrészlet futtatásakor miért lép fel memóriakezelési hiba! (1p)

```
{ CharSeq s; s += 'A'; s += 'B'; s += 'C';
  std::cout << s.c_str() << std::endl; }
```

Az implicit deklarált (default) destruktor nem jó, mivel az osztály dinamikus memóriaterületet foglal és tart nyilván.

Módosítsa az osztályt, hogy a fenti kódrészlet futtatása ne okozzon hibát! Használja az **üres helyeket** az osztály deklarációjában! (1p)

Mit ír a szabványos kimenetre a fenti kódrészlet? ABC (1p)

2. Milyen további problémára kell számítani, ha készítünk, és használunk egy **CharSeq** visszatérési értékű függvényt (pl. **CharSeq fv();**)? (1p)

Az objektum értékű függvény a visszatéréskor meghívja az objektum másoló konstruktorát. Az implicit deklarált másoló konstruktor azonban nem jó, mivel az osztály dinamikus memóriaterületet foglal és tart nyilván.

3. Módosítsa az osztályt, hogy ne lépjen fel az előző feladatban jelentkező probléma, és az osztály az elvárásoknak megfelelően működjön! Használja az **üres helyeket** az osztály deklarációjában! (3p)
4. Készítsen olyan **inserter** operátort, amivel egy **CharSeq** osztályból származó objektum kiírható egy **std::ostream** típusú objektumra. Működjön az elvárásoknak megfelelően az alábbi kódrészlet! (3p)

```
{ CharSeq d1; d1+= 'U'; std::cerr << d1 << std::endl; }
```

```
std::ostream& operator<<(std::ostream& os, const CharSeq& s) {
    return os << s.c_str();
}
```