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

| Programozás alapja 2. 2. beszámoló dolgozat. 2019.03.28. Kurz/**Terem**: L01/ | Elért pontszám: |
|---|---|
| **Név:** **Neptun:** | |

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

Egész számokból álló tömb tárolására alkalmas osztályt (**Tomb**) kell készíteni. A tömb méretét a konstruktorban lehet megadni. Követelmény, hogy az osztályból létrehozott objektumok legyenek indexelhetők, át lehessen adni azokat értékparaméterként, és működjön helyesen a többszörös értékadás is. Az osztály deklarációját elkezdtük, de nem fejeztük be:

```
class Tomb {
    size_t siz;         // tömb mérete
    int *p;             // pointer az adatokra
public:
    Tomb(size_t s = 0) :siz(s)     { p = new int[s]; }
    size_t size() const            { return siz; }
    ~Tomb()                        { delete[] p; }
    int& operator[](size_t);
    int operator[](size_t) const;
    Tomb(const Tomb&);
    Tomb& operator=(const Tomb&);


};
```

1. Amennyiben szükséges, egészítse ki az osztály deklarációját további tagfüggvényekkel úgy, hogy az osztály a fenti követelményeknek eleget tegyen! Ügyeljen arra, hogy konstans környezetben is indexelhető legyen! A tagfüggvényeket ne valósítsa meg! Feleslegesen deklarált tagfüggvényért pontot vonunk le! (4p)
2. Sorolja fel a **Tomb** osztály azon tagfüggvényeit, melyek legalább egyszer meghívódnak az alábbi kódrészlet végrehajtása során! Ha olyan függvényt is megemlít, ami nem hívódik meg, azért pontlevonás jár! (2p)

```
{ Tomb t0, t3(3); t3[0] = 8; t0 = t3; /* ITT */ }
```

**konstruktor, destruktor, nem konst operator[], operator=**

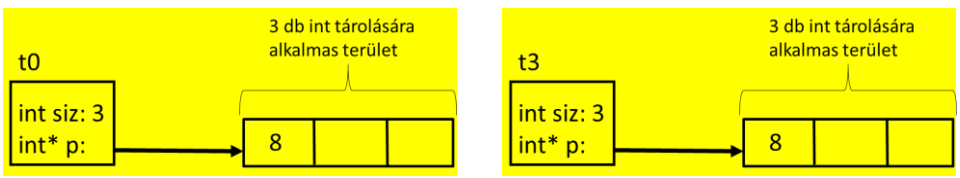
3. Kívül definiált tagfüggvényként valósítsa meg a fenti példában meghívódó indexelést lehetővé tevő tagfüggvényt! Hibás indexelés esetén az Ön Neptun azonosítóját tartalmazó kivételt dobjon! (3p)

**Egy lehetséges megoldás:**

```
int& Tomb::operator[](size_t i) {
  if (i >= siz) throw "ABC123";
  return p[i];
}
```

4. Tételezze fel, hogy a minden szükséges tagfüggvény helyesem működik! Vázlatosan rajzolja le a fenti példában keletkező **t0** és **t3** objektumok azon memóriaképét, amilyen állapotot az ITT kommenttel jelzett helyen felvesznek (adja meg/rajzolja le az adattagokat, és azok értékét)! (1p)

**Egy lehetséges megoldás:**



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

| Programozás alapja 2. 2. beszámoló dolgozat. 2019.03.28. Kurz/**Terem**: L02/ | Elért pontszám: |
|---|---|
| **Név:** **Neptun:** | |

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

Valós számokból álló vektor tárolására alkalmas osztályt (**Vektor**) kell készíteni. A vektor méretét a konstruktorban lehet megadni. Követelmény, hogy az osztályból létrehozott objektumok legyenek indexelhetők, át lehessen adni azokat értékparaméterként, és működjön helyesen a többszörös értékadás is. Az osztály deklarációját elkezdtük, de nem fejeztük be:

```
class Vektor {
    const size_t siz;  // vektor mérete
    double *p;         // pointer az adatokra
public:
    Vektor(size_t s = 0) :siz(s)  { p = new double[s]; }
    size_t size() const           { return siz; }
    ~Vektor()                     { delete[] p; }
    double& operator[](size_t);
    double operator[](size_t) const;
    Vektor(const Vektor&);
    Vektor& operator=(const Vektor&);



};
```

1. Amennyiben szükséges, egészítse ki az osztály deklarációját további tagfüggvényekkel úgy, hogy az osztály a fenti követelményeknek eleget tegyen! Ügyeljen arra, hogy konstans környezetben is indexelhető legyen! A tagfüggvényeket ne valósítsa meg! Feleslegesen deklarált tagfüggvényért pontot vonunk le! (4p)
2. Sorolja fel a **Vektor** osztály azon tagfüggvényeit, melyek legalább egyszer meghívódnak az alábbi kódrészlet végrehajtása során! Ha olyan függvényt is megemlít, ami nem hívódik meg, azért pontlevonás jár! (2p)

```
{ Vektor v1(3); v1[1] = 3; const Vektor v2 = v1; /* ITT */ }
```

**konstruktor, másoló konstruktor, destruktor, nem konst operator[]**

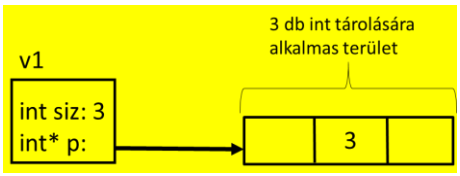
3. Kívül definiált tagfüggvényként valósítsa meg a fenti példában meghívódó indexelést lehetővé tevő tagfüggvényt! Hibás indexelés esetén az Ön Neptun azonosítóját tartalmazó kivételt dobjon! (3p)

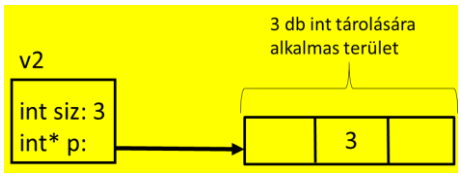
**Egy lehetséges megoldás:**

```
double& Vektor::operator[](size_t i) {
  if (i >= siz) throw "ABC123";
  return p[i];
}
```

4. Tételezze fel, hogy a minden szükséges tagfüggvény helyesem működik! Vázlatosan rajzolja le a fenti példában keletkező **v1** és **v2** objektumok azon memóriaképét, amilyen állapotot az ITT kommenttel jelzett helyen felvesznek (adja meg/rajzolja le az adattagokat, és azok értékét)! (1p)

**Egy lehetséges megoldás:**



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

| Programozás alapja 2. 2. ellenőrző dolgozat. 2019.03.29. Kurz/**Terem**: L03/ | Elért pontszám: |
|---|---|
| **Név:** **Neptun:** | |

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

10 db egész érték tárolására alkalmas osztályt (Array) kell készíteni. Az osztály konstruktora töltse fel a tárolót a paraméterként kapott értékkel! Ha nincs paraméter, akkor 0-val! További követelmény, hogy az osztályból létrehozott objektumok legyenek indexelhetők, át lehessen adni azokat értékparaméterként, és működjön helyesen a többszörös értékadás is. Az osztály deklarációját elkezdtük, de nem fejeztük be:

```
class Array {
    static const size_t siz = 10; // a fix méret így könnyebben változtatható
    int adat[siz];                // itt vannak az adatok
public:
    int& operator[](size_t);
    size_t size() const           { return siz; }
    Array(int val = 0);
    int operator[](size_t) const;



};
```

1. Amennyiben szükséges, egészítse ki az osztály deklarációját további tagfüggvényekkel úgy, hogy az osztály a fenti követelményeknek eleget tegyen! Ügyeljen arra, hogy konstans környezetben is indexelhető legyen! A tagfüggvényeket ne valósítsa meg! Feleslegesen deklarált tagfüggvényért pontot vonunk le! (Figyeljen! Az osztály nem tartalmaz dinamikus adatot!) (2p)
2. Készítsen inserter operátort, amivel egy **Array** típusú objektum minden eleme kiíratható egy **std::ostream** típusú objektumra! Az értékeket szóköz válassza el egymástól! Az operator használatára a 3. feladatban lát példát! (3p)

```
Egy lehetséges megoldás:

std::ostream& operator<<(std::ostream& os, const Array& a) {
  for (size_t i = 0; i < a.size(); i++)
    os << a[i] << ' ';
    return os;
}
```

3. Sorolja fel az **Array** osztály azon **tagfüggvényeit**, melyek legalább egyszer meghívódnak az alábbi kódrészlet végrehajtása során! Ha olyan függvényt is megemlít, ami nem hívódik meg, azért pontlevonás jár! (3p)

```
{ Array a1(3); a1[1] = 3; const Array a2 = a1; std::cout << a2 << std::endl; }

A válasz részben függ a 2. feladat konkrét megoldásától:
konstruktor, másoló konstruktor, destruktor, mindkét operator[], size
```

4. Tételezze fel, hogy hibás indexelés esetén const char* típusú hiba keletkezik! Rövid kódrészlettel demonstrálja a hibás indexelést és mutassa be a kivétel elkapását! (2p)

```
Egy lehetséges megoldás:

{ Array a0;
  try {
    std::cout << a0[10];
  } catch (const char *p) {
    cout << "Megvagy:" << p << std::endl;
  }
}
```

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

| Programozás alapja 2. 2. ellenőrző dolgozat. 2019.03.29. Kurz/**Terem**: L04/ | Elért pontszám: |
|---|---|
| **Név:** **Neptun:** | |

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

3 db valós értéket tartalmazó vektor tárolására alkalmas osztályt (Vektor3D) kell készíteni. Az osztály konstruktora töltse fel a tárolót a paraméterként kapott értékkel! Ha nincs paraméter, akkor 0-val! További követelmény, hogy az osztályból létrehozott objektumok legyenek indexelhetők, át lehessen adni azokat értékparaméterként, és működjön helyesen a többszörös értékadás is. Az osztály deklarációját elkezdtük, de nem fejeztük be:

```
class Vektor3D {
    static const size_t siz = 3;  // a fix méret így könnyebben változtatható
    double adat[siz];             // itt vannak az adatok
public:
    double operator[](size_t) const;
    size_t size() const           { return siz; }
    Vektor3D(double val = 0);
    double& operator[](size_t);



};
```

1. Amennyiben szükséges, egészítse ki az osztály deklarációját további tagfüggvényekkel úgy, hogy az osztály a fenti követelményeknek eleget tegyen! Ügyeljen arra, hogy konstans környezetben is indexelhető legyen! A tagfüggvényeket ne valósítsa meg! Feleslegesen deklarált tagfüggvényért pontot vonunk le! (Figyeljen! Az osztály nem tartalmaz dinamikus adatot!) (2p)
2. Készítsen inserter operátort, amivel egy **Vektor3D** típusú objektum minden eleme kiíratható egy **std::ostream** típusú objektumra! Az értékeket szóköz válassza el egymástól! Az operator használatára a 3. feladatban lát példát! (3p)

**Egy lehetséges megoldás:**

```
std::ostream& operator<<(std::ostream& os, const Vektor3D& v) {
  for (size_t i = 0; i < v.size(); i++)
    os << v[i] << ' ';
    return os;
}
```

3. Sorolja fel a **Vektro3D** osztály azon **tagfüggvényeit**, melyek legalább egyszer meghívódnak alábbi kódrészlet végrehajtása során! Ha olyan függvényt is megemlít, ami nem hívódik meg, azért pontlevonás jár (3p)

```
{ Vektor3D v0, v3(3.14); v3[0] = 8; v0 = v3; std::cout << v0 << std::endl;}
```

**A válasz részben függ a 2. feladat konkrét megoldásától:**

**konstruktor, destruktor, operator=, mindkét operator[], size**

4. Tételezze fel, hogy hibás indexelés esetén const char* típusú hiba keletkezik! Rövid kódrészlettel demonstrálja a hibás indexelést és mutassa be a kivétel elkapását! (2p)

**Egy lehetséges megoldás:**

```
{ Vektor3D v;
  try {
    std::cout << v[10];
  } catch (const char *p) {
    cout << "Megvagy:" << p << std::endl;
  }
}
```