1.AA Megoldó Aladár 1.A

Programozás alapjai 2. (inf.) 1. ZH 2019.04.05. lab. hiányzás: 0+0	E/L1-R4A
ABC123 Q-I/1.	kZH: 17 E:15

Minden beadandó megoldását a feladatlapra, a feladat után írja! Készíthet piszkozatot, de csak a feladatlapra írt megoldásokat értékeljük! Jelölje a táblázatban, ha oldott meg IMSC feladatot. Ezt csak az alapfeladatok 75%-os teljesítése mellett értékeljük.

*A megoldások során feltételezheti, hogy minden szükséges input adat az előírt formátumban rendelkezésre áll. A feladatok megoldásához csak a letölthető C, C++ és STL összefoglaló használható. Elektronikus eszköz (pl. tablet, notebook, mobiltelefon) nem használható. A feladatokat **figyelmesen olvassa** el, megoldásukhoz **ne használjon fel STL** tárolót, kivéve, ha a feladat ezt külön engedi/kéri! **Ne írjon felesleges** függvényeket ill. kódot!*

Az első feladatban minimum 5 pontot el kell érnie ahhoz, hogy a többi feladatot értékeljük.

f.	max.	elért	jav.
1.	10		
2.	10		
3.	10		
4.	10		
Σ	**40**		
IM.	10		
Van megoldott IMSC:			

1. feladat: **Σ 10 pont**

a) Az alábbi kódrészletben az **adatp** változó valós típusú elemeket tartalmazó tömbre mutat. A **novel()** függvény feladata az allokált tömb méretének megnövelése, és az adatok átmásolása a megnövelt területre. **Készítse** el a **novel** függvényt úgy, hogy az az alábbi kódrészletben használható legyen! **Figyelje** meg a függvény használatát, és válasszon ahhoz megfelelő paramétereket! Ügyeljen, hogy ne lépjen fel memóriaszivárgás! (3p)

```
{ const int meretMost = 20, meretNovel = 11;
  double *adatp = new double[meretMost];
                                  // itt használjuk a tömbünket pl. adatp[meretMost-1] = 17;
  adatp = novel(adatp, meretMost, meretNovel);// megnöveljük az allokált területet
                                  // elvárjuk a korábbi adatp[0]...adatp[meretMost-1] értékek meglétét.
  delete[] adatp;
}
```

Egy lehetséges megoldás:

```
double* novel(double* p, size_t n, size_t d) {
  double *tmp = new double[n+d];
  for (size_t i = 0; i < n; i++)
    tmp[i] = p[i];
  delete[] p;
  return tmp;
}
```

b) Definiáljon osztályt (**Kiirathato**), ami később heterogén kollekció alaposztályaként lesz használható. Az osztálynak legyen egy publikus metódusa (**print**), amelyet a leszármazottaknak **kötelező** felüldefiniálni. A metódus **void** visszatérésű, és egy **std::ostream&** paramétere van. (2p)

Egy lehetséges megoldás:

```
struct Kiirathato {
  virtual void print(std::ostream&) const = 0;
  virtual ~Kiirathato() {};
};
```

c) A b) feladat alaposztályából származtasson egy tetszőleges osztályt, melynek van valamilyen privát adattagja! Az adattag kezdőértéke legyen megadható a konstruktor paramétereként! Valósítsa meg az osztály úgy, hogy a **print** tagfüggvény írja ki az adattagot a paraméterként kapott stream-re! (3p)

Egy lehetséges megoldás:

```
class Valami : public Kiirathato {
  int adat;
public:
  Valami(int a) :adat(a) {}
  void print(std::ostream& os) const { os << adat; }
};
```

d) Jelölje (pl. karikázza be), hogy az állítás igaz (I), vagy hamis (H) a C++ nyelvre! Minden bejelölt válasz 0.5 pont, ha helyes, -0.5p pont, ha hibás! Az esetleges negatív eredmény is összeadódik a többi részfeladatra kapott pontokkal. (2p)

Referencia típusú, nem statikus tagváltozó csak inicializáló listán inicializálható.	**I**	H
Absztrakt osztálynak nem kell, hogy legyen virtuális tagfüggvénye.	I	**H**
Az implicit deklarált másoló konstruktor nem csinál semmit az adattagokkal.	I	**H**
A destruktor mindig meghívja a tartalmazott objektumok destruktorait.	**I**	H

2. Feladat **Σ 10 pont**

Készítsen maximum **n** valós adat tárolására alkalmas vektor osztályt (*Vektor*)! Az **n** értékét a konstruktorban adjuk meg, melynek alapértelmezett értéke 10. A létrehozás és megszüntetés mellett meg kell valósítani az alábbi műveleteket:

operator[]	Egy adat direkt elérése (indexelés). Nincs indexhatár ellenőrzés.
size()	Megadja a vektor méretét (n)

További követelmény, hogy az osztály **legyen átadható érték szerint** függvényparaméterként, és **működjön** helyesen **a többszörös értékadás** is. Feltételezhető, hogy az osztályból <u>további osztályokat származtatunk</u>. Valósítsa meg az osztályt C++ nyelven úgy, hogy az alábbi kódrészlet az elvárásoknak megfelelően működjön, és ne legyen memóriaszivárgás!

```
Vektor v1(3), v2(110);        // létrehoz egy 3 és egy 110 elemű tárolót
Vektor *vp = new Vektor[20];
vp[3][0] = 5;
std::cout << vp[3].size();    // kiírja: 10
v1[1] = 117.45;
std::cout << v1[1];           // kiírja : 117.45
const Vektor v0 = v1;
std::cout << v0.size();       // kiírja: 3
std::cout << v0[1];           // kiírja: 117.45
Vektor v = vp[3];
delete [] vp;
std::cout << v[0];            // kiírja: 5
```

IMSC feladat:

A 2. feladatban elkészített tároló <u>felhasználásával, annak módosítása nélkül</u> készítsen olyan *Verem* osztályt, ami valós számok verem szervezésű tárolására alkalmas, és rendelkezik a következő metódusokkal:

void push(double) – adat verembe helyezése, amennyiben a verem megtelik, kivételt dob
void pop() – verem legfelső elemének eldobása, amennyiben a verem üres, kivételt dob
double& top() – verem legfelső elemének elérése, amennyiben a verem üres, kivételt dob
bool isEmpty() – igaz értéket ad, ha a verem üres.

Legyen logikai értékre konvertáló operátora is, ami igaz értéket ad, ha a verem nem üres!

a) A verem maximális méretét a konstruktorban lehet megadni. További követelmény, hogy az osztály **legyen átadható érték szerint** függvényparaméterként, és **működjön** helyesen **a többszörös értékadás** is. Valósítsa meg az osztályt C++ nyelven úgy, hogy az alábbi kódrészlet az elvárásoknak megfelelően működjön, és ne legyen memóriaszivárgás:

```
Verem st(10);
st.push(1.1);
st.push(2.2);
st.push(3.3);
while (st) {
    std::cout << st.top() << std::endl;
    st.pop();
}
```

Egy lehetséges megoldás:

```
class Verem : private Vektor {
    size_t vsiz;
 public:
    Verem(size_t n) : Vektor(n), vsiz(0) {}
    Verem(const Verem& v) :Vektor(v), vsiz(v.vsiz) {}
    bool isEmpty() const { return vsiz == 0; }
    double& top() {
        if (isEmpty()) throw "Empty";
        return (*this)[vsiz-1];
    }
    void pop() {
        if (isEmpty()) throw "Empty";
        vsiz--;
    }
    void push(double d) {
        if (vsiz >= size()) throw "Empty";
        (*this)[vsiz++] = d;
    }
    operator bool() const { return !isEmpty(); }
};
```

Egy lehetséges megoldás:

```
class Vektor {
    size_t siz;
    double* pData;
 public:
    Vektor(size_t n = 10) :siz(n), pData(new double[siz]) {}
    Vektor(const Vektor& v) :pData(NULL) { *this = v;}
    size_t size() const { return siz; }
    Vektor& operator=(const Vektor& v) {
        if (this != &v) {
            delete[] pData;
            siz = v.siz;
            pData = new double[siz];
            for (size_t i = 0; i < siz; i++)
                pData[i] = v.pData[i];
        }
        return *this;
    }
    double operator[](size_t i) const { return pData[i]; }
    double& operator[](size_t i) { return pData[i]; }
    virtual ~Vektor() { delete[] pData; }
};
```

---

3. Feladat Σ 10 pont

a) Készítsen generikus algoritmust (*Print*), ami kiírja a szabványos kimenetre egy paraméterként kapott indexelhető objektum azon elemeit, amelyek megfelelnek a szintén paraméterként kapott predikátummal megadott feltételnek! A kiíráskor az egyes elemeket vesszővel válassza el egymástól (az utolsó elem után is lehet vessző). A kiírást soremeléssel zárja.

b) Készítsen olyan függvényt, ami a fenti algoritmus predikátumaként alkalmazható és képes eldönteni, hogy egy valós szám nagyobb-e, mint 0!

c) Írjon kódrészletet, melyben beolvas maximum 20 db valós számot a 2. feladatban elkészített tárolóba, majd az a) feladatrészben elkészített függvénysablon és a b) feladatrészben elkészített predikátum segítségével kiírja a beolvasott számok közül a pozitív elemeket a szabványos kimenetre!

d) Írjon kódrészletet, melyben beolvas maximum 15 egész számot egy tömbbe, majd az a) feladatrészben elkészített függvénysablon segítségével kiírja a beolvasott számok közül a páratlan elemeket a szabványos kimenetre! Készítse el a szükséges predikátumot!

Egy lehetséges megoldás:

a)

```
template <typename T, typename P>
void Print(const T& t, int n, P pred) {
    for (int i = 0; i < n; i++)
        if (pred(t[i]))
            std::cout << t[i] << ",";
    std::cout << std::endl;
}
```

---

b)

```
bool ispos(double d) { return d > 0; }
```

---

c)

```
Vektor v(20);
int db = 0;
while (db < 20 && std::cin >> v[db])
    db++;
Print(v, db, ispos);
```

---

d)

```
bool isodd(int i) { return (i & 1) == 1; }
int it[15];
int db = 0;
while (db < 15 && std::cin >> it[db])
    db++;
Print(it, db, isodd);
```

4. Feladat Σ 10 pont

Feladatunk, hogy nyilvántartsunk számlákat (*Szamla*). Egy számlának van számlaszáma (*szam*, String, ”1234567”), és egyenlege (egyenleg, double, 0). Ezek az attribútumok konstruktorban állíthatók, a zárójelben megadott névvel, típussal és kezdőértékkel rendelkeznek és kívülről közvetlenül nem elérhetők. A számlaszám lekérdezhető (*getSzam*), az egyenleg állítható is (*getEgyenleg*, *setEgyenleg*). Az utóbbi dobjon kivételt, ha az értéke csökken!

A rendszerben az offshore számlákat is nyilván kell tartani (*Offshore*). Az Offshore is Szamla (teljes mértékben kompatibilis vele, de a számát és az egyenlegét a konstruktorban meg kell adni, nincs default kezdőértékük), ezen kívül a tulajdonosok számát is tárolja (*tulaj*, short, 1). Ehhez az attribútumhoz egyetlen lekérdező/módosító függvény tartozik (*mennyi)*. Ha az offshore számlának lekérdezzük az egyenlegét, akkor az egy tulajdonosra eső egyenleg értékét adja vissza.

A rendszert később újabb osztályokkal bővítjük, ezért úgy tervezze meg a Szamla osztályt, hogy ez ne vezessen akaratlanul is memóriaszivárgáshoz!

a)Rajzoljon a feladathoz osztálydiagramot (metódusok és attribútumok nélkül)!

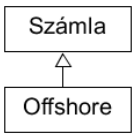
b)Definiálja a két osztályt C++ nyelven, a szövegben megjelenő metódusokat inline definiálja! Minden attribútum legyen privát láthatóságú! Az alábbi kódrészlet fusson hibátlanul:

```
Szamla* s1 = new Szamla ("20180408-10101010");
const Szamla s2("76543210-12345678", 123000000);
Offshore* s3 = new Offshore("710352-123456", 1300000000, 3);
std::cout << s2.getSzam() << ": " << s2.getEgyenleg() << std::endl;
s1->setEgyenleg(200);
std::cout << s1-> getSzam() << ": " << s1->getEgyenleg() << std::endl;
s3->mennyi() = 6;
std::cout << s3->getEgyenleg() << ", " << s3->mennyi() << std::endl;
delete s1;
delete s3;
```

c) Egy rövid kódrészletben idézzen elő olyan esetet, amikor egy *Szamla* objektum kivételt generál! A kódrészlet mutassa be a keletkező kivétel lekezelését!

Egy lehetséges megoldás:

a)



b)

```
class Szamla {
      String szam;
      double egyenleg;
public:
      Szamla(String n = "1234567", double e = 0) : szam(n), egyenleg(e) {}
      String getSzam() const { return szam;  }
      virtual double getEgyenleg() const { return egyenleg;  }
      void setEgyenleg(double d) {
            if (d < egyenleg) throw "Jajajajaj!!!");
            egyenleg = d;
      }
      virtual ~Szamla() {}
};
class Offshore : public Szamla {
      short tulaj;
public:
      Offshore(String n, double e, short t) : Szamla(n, e), tulaj(t) {}
      double getEgyenleg() const { return Szamla::getEgyenleg() / tulaj; }
      short& mennyi() { return tulaj; }
};
```

c)

```
    try {
        Szamla sz("12345678", 123);
        sz.setEgyenleg(100);
    } catch (const char*) {
        cout << "megvagy!\n";
    }
```