1000.AA Megoldo Alfréd1000.A

Programozás alapjai 2. (inf.) 2. PZH 2018.05.22. hiányzás:0+3 L4-R4P	ZH: 27,5+26
ABCDEF **IB.028/100.**	Sum:0 e:19

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! **Ne írjon felesleges** függvényeket ill. kódot! Súlyos hibákért (pl. privát változó külső elérése, memóriakezelési hiba, stb.) pontot vonunk le.*

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: Beugró. Használhat STL tárolót és algoritmust! **Σ 10 pont**

a) Adott az alábbi kódrészlet:

```
int a[] = { 9, 4, 1, 8, 8, 4, 8, 3, 3, 3};
int *p = std::unique(a, a+10);
std::ostream_iterator<int> out(std::cout, ",");
std::copy(a, p, out);

std::list<int> tarolo(a, p);

tarolo.sort();

..std::copy(tarolo.begin(), tarolo.end(), out);
```

Milyen értéket vesz fel p? p == a + 7 (1p)

Mit ír ki a kódrészlet a szabványos kimenetre? 9,4,1,8,4,8,3, (1p)

Ciklusszervező utasítás használata nélkül a pontozott vonalra írva egészítse ki a kódrészletet:

1. a *unique* eredményeként kapott sorozat elemeit tegye be egy *std::list* tárolóba! (1p)
2. rendezze a listát, majd írja ki a rendezett sorozatot a szabványos kimenetre! (1p)

b) Adott az alábbi kódrészlet:

```
struct Valami {
  std::set<std::string> attr;
} t1;
```

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

`t1.attr.insert("haho");` utasítás helyes.	**I**	H
`Valami t2 = t1;` utasítás helyes, de a `t1` adattagját nem másolta le az implicit másoló konstruktor.	I	**H**
`t1 = t1 = t2;` utasítás hibás, mert a valami osztálynak nincs az értékadó operátora.	I	**H**
`t2.insert("jajaj");` utasítás helyes, mert az *std::set*-nek van insert metódusa.	I	**H**

c) Az alábbi kódrészletben az **adatp** valós elemeket tartalmazó tömbre mutat. A **novel()** függvény feladata az allokált tömb méretének megnövelése a paraméterként kapott értékkel úgy, hogy a tömbben levő adatok a megnövelt tömbben elérhetőek legyenek. **Készítse** el a **novel** függvényt! (2p)

```
double *adatp = new double[N];
    // itt használjuk a tömbünket pl. adatp[N-1] = 8;
adatp = novel(adatp, 100);
    // itt elvárjuk, hogy az adatp[0]...adatp[N-1] értékek változatlanul elérhetőek legyenek.
```

Egy lehetséges megoldás:

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

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 ponttal. (2p)

Az implicit értékadó operátor lemásolja adattagokat.	**I**	H
Sablonnak nem lehet másik sablon a paramétere.	I	**H**
A `try catch` blokkban létrejött objektumokat a blokk után kell megszüntetni.	I	**H**
A *const_iterator* értéke megváltoztatható, csak az adat nem változtatható meg, amire az iterátor hivatkozik.	**I**	H

2. Feladat **Σ 10 pont**

Készítsen adapter sablont (*ModuloTomb*), ami minden olyan *szabványos* sorozattárolóra alkalmazható, aminek van at() és size() tagfüggvénye! A *ModuloTomb* elemei **modulo N** aritmetika szerint érhetők el az *at()* tagfüggvény használatával, ahol **N** a tömb pillanatnyi mérete (*size()*). Azaz az *at()* tagfüggvény paraméterét maradékosan el kell osztani a tömb méretével. Úgy alakítsa ki a sablont, hogy alapértelmezésként a sorozattároló az *std::vector* legyen! A sorozattároló minden tagfüggvénye legyen elérhető, kivéve az *operator[]*! Ügyeljen a sorozattárolókra jellemző konstruktorok megvalósítására is! (5p)

Példa a használatra:

```
ModuloTomb<int> t3(3);          // 3 elemű int tömb, nullával feltöltve
t3.at(0) = 1;                   // első eleme
std::cout << t3.at(3);          // ez is az első (3%3 = 0) , ezért 1-et ír ki!
t3.at(2) = 3;                   // utolsó eleme
std::cout << t3.at(5);          // ez is az utolsó, ezért 3-at ír ki!
```

b) Készítsen generikus algoritmust (monoton) annak eldöntésére, hogy egy iterátorokkal adott jobbról nyílt intervallum elemei szigorúan monoton növekvő sorozatot alkotnak-e! Az algoritmust úgy valósítsa meg hogy csak az ún. Input Iterator-okra jellemző műveleteket használja (konstruktor, másoló, értékadó, ++, ==, != ++, *, ->). (3p)

c) Rövid kódrészletben hozzon létre az elkészített adapter és az *std::deque* felhasználásával egy double elemeket tartalmazó tömböt! Olvasson be a standard inputról valós számokat file végéig, majd a monoton sablon segítségével döntse el, hogy a beolvasott sorozat szigorúan monoton sorozatot alkot-e! (2p)

IMSC FELADAT: Specializálja a *ModuloTomb* osztályt, pointerek tárolására! Tételezze fel, hogy a sorozattárolónak van iterártora, új pointert pedig csak a *push_back()* tagfüggvénnyel teszünk be a tárlóba! Ezzel a „tárolóra bízzuk” az objektumot. Maradjon a tároló továbbra is átadható értékként és értékadás bal oldalán is szerepelhessen! Tételezze fel, hogy a tárolt pointerek olyan objektumokra mutatnak, melyeknek van *clone()* tagfüggvénye! Ügyeljen a hibakezelésre! 10 p

Egy lehetséges megoldás:

a)

```
template <typename T, class C = std::vector<T> >
class ModuloTomb : public C {        // csak öröklés jöhet szóba, mivel mindennek (majdnem) látszani kell
    T& operator[](size_t ix);
    T operator[](size_t ix) const;
 public:
    ModuloTomb(size_t n = 0, const T& value = T()) : C(n, value) {}
    template<typename Iter>
    ModuloTomb(Iter first, Iter last) : C(first, last) {}
    T& at(int ix) {
                return C::at(ix%C::size()); }
    const T& at(int ix) const {
                return C::at(ix%C::size()); }
};
```

b)

```
template <typename Iter>
bool monoton(Iter first, Iter last) {
    if (first != last) {             // üres halmaz kivédése
        Iter next = first;
        ++next;
        while (next != last) {
            if (*next <= *first)     // szebb lenne a tagadott nagyobb reláció: !(*next > *first)
                return false;
            ++first;
            ++next;
        }
    }
    return true;
}
```

c)

```
    ModuloTomb<double, std::deque<double> > t;
    double d;
    while (std::cin >> d) {
        t.push_back(d);
    }
    std::cout << monoton(t.begin(), t.end());
```

IMSC:

```
template <typename T, class C>
class ModuloTomb<T*, C > : public C {
    T*& operator[](size_t ix);
    T* operator[](size_t ix) const;
 public:
    ModuloTomb() : C() {}

    ModuloTomb(const ModuloTomb& t) {
        *this = t;
    }

    ModuloTomb& operator=(const ModuloTomb& t) {
        if (this != &t) {
            clear();
            for (size_t i = 0; i < t.size(); i++)
                push_back(t.at(i)->clone());
        }
        return *this;
    }

    void push_back(T* p) {
        try {
            C::push_back(p);
        } catch (...) {
            delete p;
        }
    }

    void clear() {
        for (size_t i = 0; i < C::size(); i++)
            delete at(i);
        C::clear();
    }

    T*& at(int ix) {
                return C::at(ix%C::size()); }

    T* const & at(int ix) const {
                return C::at(ix%C::size()); }

    ~ModuloTomb() { clear(); }
};
```

3. Feladat **Σ 10 pont**

Áramkörmodellünket grafikus felhasználói felülettel akarjuk ellátni. A grafikus felületen megjelenő nyomógomb megnyomásakor egy kapcsolót szeretnénk be- ill. újabb megnyomásra kikapcsolni. A megoldáshoz ún. *callback* technikát választottunk. A grafikus felület osztályait és az áramkörmodell osztályait sem módosíthatjuk. Ezek egyszerűsített deklarációi a következők:

```
class Gomb {
  CallBack& cb;
public:
  Gomb(CallBAck& cb) :cb(cb) {}
  void megnyom() { cb(); }
  virtual ~Gomb() {}
};
```

```
class Kapcsolo {
public:
  void bekapcsol();
  void kikapcsol();
  virtual ~Kapcsolo();
};
```

a) A *callback* megoldáson kívül milyen más megoldásról tanult? (1p)

signal/slot mechanizmus

b) Definiálja a *CallBack* osztályt! (2p)

```
struct CallBack {
    virtual void operator()() = 0;
};
```

c) A *Kapcsolo* osztály felhasználásával, annak módosítása nélkül hozzon létre egy *ValtoKapcsolo* osztályt, ami „összeköthető” a nyomógombbal, melynek megnyomására be- ill. kikapcsol a kapcsoló. A kapcsoló kezdeti állapota kikapcsolt legyen! (5p)

Egy lehetséges megoldás:

```
class ValtoKapcsolo : public Kapcsolo, public CallBack {
  bool all;
public:
  ValtoKapcsolo() : all(false) {}
  void operator()() {
      all = !all;
      if (all) bekapcsol();
      else kikapcsol();
  }
};
```

d) Egy rövid kódrészletben Mutassa be, hogy hogyan lehet működtetni a kapcsolót! (2p)

```
ValtoKapcsolo k;
Gomb g(k);
g.Megnyom();
g.Megnyom();
```

4. Feladat **Σ 10 pont**

Burkus király egységesíteni akarja a közalkalmazotti nyilvántartásokat. Egységes rendszerben (*Gephaz*) kell tárolni a közalkalmazottakat (*Kozalkalmazott*), akik az első verzióban katonák (*Katona*) és az orvosok (*Orvos*) lehetnek, de később további szakmák dolgozóit (bírók, tanárok, stb) is kezelni kell tudni. A modell adjon támogatást arra, hogy vannak orvosok, akik egyben katonák is (*Szanitec*).

A közalkalmazottaknak kiíratható (*nyomtat*) az azonosítója (*azon*, string), a katonáknak a rendfokozata (*rang*, string), az orvosoknak pedig a diploma megszerzésének éve (*ev*, int). A *Szanitec*-nek azonosítója, rangja és diplomaszerzési éve is van, és kezelhető katonaként és orvosként is. Kiíratáskor a *Szanitec* minden adata kiíródik, nem baj, ha többször is. A kiíratást minden tárolt típusnál a függvényparaméterként megadott ostream-re lehet kérni.

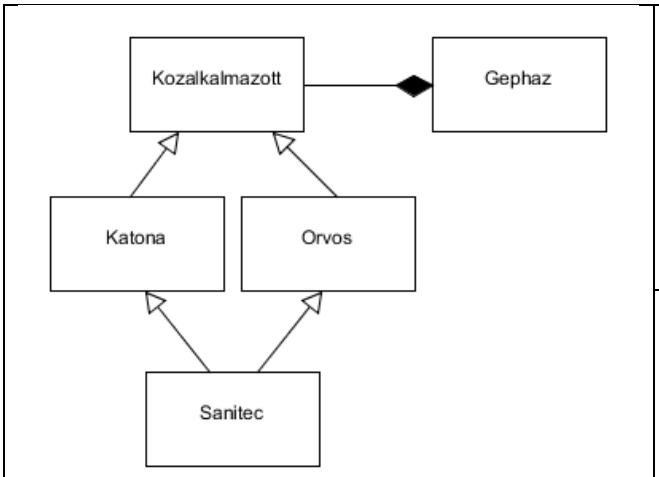
A *Gephaz* rendszerbe fel lehessen venni új közalkalmazottat (*alkalmaz*), illetve paraméterben megadott ostream-re ki lehessen írni a meglevő közalkalmazottak minden tárolt adatát (*leker*). Ha a *Gephaz* rendszer megsemmisül, a benne tárolt adatok is elvesznek.

Tervezzen objektum-orientált megoldást a fenti leírás alapján a dőlt betűs megnevezések felhasználásával! Az attribútumok kivétel nélkül legyenek privátok és konstruktorban állíthatók. Használja ki az STL adta lehetőségeket! **Rajzoljon** UML osztálydiagramot, amin csak az osztályok neve szerepel!

Definiálja a **Gephaz**, **Kozalkalmazott**, **Orvos**, és **Szanitec** osztályokat! A konstruktorokat *inline* adja meg, a többi metódust csak deklarálja. Másoló konstruktort nem kell készíteni. Használjon std::*vector* tárolót!

Valósítsa meg az *Gephaz* osztály **alkalmaz** és **leker** metódusát, és az ezek végrehajtása közben meghívott további metódusokat minden definiált osztályhoz!

1000.AA Megoldo Alfréd1000.A




```
class Gephaz{
  std::vector<Kozalkalmazott*> data;
public:
  void alkalmaz(Kozalkalmazott*);
  void leker(ostream&) const;
  ~Gephaz();
};
```

```
class Kozalkalmazott {
  string azon;
public:
  Kozalkalmazott(string n) : azon(n) {}
  virtual void nyomtat(std::ostream&) const;
  virtual ~Kozalkalmazott() {}
};
```

```
using std::string;
using std::ostream;
```

```
class Orvos : virtual public Kozalkalmazott {
  int ev;
public:
  Orvos(string n, int d) : Kozalkalmazott(n), ev(d) {}
  void nyomtat(ostream&) const;
};
```

```
class Sanitec : public Katona, public Orvos {
public:
  Sanitec(string n, string r, int d) : Kozalkalmazott(n), Katona(n,r), Orvos(n,d) {}
  void nyomtat(ostream&) const;
};
```

```
void Gephaz::alkalmaz(Kozalkalmazott* s) {
  data.push_back(s);
}
```

```
void Gephaz::leker(ostream& os) const {
  for (size_t i = 0; i < data.size(); i++) {
    data[i]->nyomtat(os);
    os << std::endl;
  }
}
```

```
void Kozalkalmazott::nyomtat(ostream& os) const {
      os << azon;
}
void Orvos::nyomtat(ostream& os) const {
      Kozalkalmazott::nyomtat(os);
      os << " " << ev;
}
void Sanitec::nyomtat(ostream& os) const {
      Katona::nyomtat(os);
      os << " ";
      Orvos::nyomtat(os);
}
```