500.AA Második Megoldó Kulcsár 500.A

Programozás alapjai 2. (inf.) 2. pZH 2019.05.21. lab. hiányzás: +	Ex: +
ABCDEF **IB.028/1.**	p: e:

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) **Jelölje** (pl. karikázza be), hogy az állítás igaz (I), vagy hamis (H) a C++ nyelvre! Minden bejelölt válasz 0.5 pont, ha helyes, -0.5 pont, ha hibás! Az esetleges negatív eredmény is összeadódik a többi részfeladatra kapott ponttal. (4p)

Az implicit értékadó operátor nem hívja meg az ősosztály értékadó operátorát.	I	**H**
Statikus tagfüggvénynek nincs this pointere.	**I**	H
A konstruktor előbb hajtja végre a programozott törzset, és csak ezután hívja az ősosztályok konstruktorát.	I	**H**
Az iterátor egy általánosított pointer.	**I**	H
Template paraméter nem lehet másik template típus	I	**H**
A sorozattárolók end() tagfüggvénye az utolsó elemre mutató pointert ad.	I	**H**
A sorozattárolók rend() tagfüggvénye az első elemet megelőző (nem létező) elemre mutató iterátort ad.	**I**	H
Minden szabványos sorozattárolónak van insert() tagfüggvénye.	**I**	H

b) Rövid kódrészletben hozzon létre egy `std::vector` típusú tárolót (*tarolo*) könyvcímek (*std::string*) tárolására. A szabványos bemenetről olvasson be könyvcímeket a tárolóba! A könyvcímek soronként érkeznek. A tetszőleges hosszúságú címekben szóközök is lehetnek. Az adatok végét fájl vége jelzi. A sorok beolvasásához célszerű használni az `std::istream& std::getline (std::istream&, std::string&);` függvényt (3p)

```
std::vector<std::string> tarolo;
std::string cim;
while (std::getline(std::cin, cim))
    tarolo.push_back(cim);
```

c) Definiáljon funktort az `std::for_each` algoritmushoz, amellyel a **b)** részfeladatban feltöltött tárolóból kiírathatóak a szabványos kimenetre azok a könyvcímek, melyek hossza rövidebb a funktor konstruktorában megadott értéknél! Ezt felhasználva írja ki a tárolóból a 30 karakternél rövidebb címeket! (3p)

```
struct rovidCim {
    size_t len;
    rovidCim(size_t len) :len(len) {}
    void operator()(const std::string& s) {
        if (s.size() < len)
            std::cout << s << endl;
};

std::for_each(tarolo.begin(), tarolo.end(), rovidCim(30));
```

500.AA Második Megoldó Kulcsár 500.A

2. Feladat **Σ 10 pont**

a) Készítsen egy LIFO (*Stack*) adapter sablont, ami a sablonparaméterként átadott tárolóból verem működésű tárolót hoz létre. Alapértelmezett tárolóként válasszon megfelelő szabványos sorozattárolót! A verem rendelkezzen a következő műveletekkel:

- *konstruktor* – csak paraméter nélküli konstruktora legyen, ami létrehoz egy üres vermet
- *empty()* – igaz, ha a verem üres
- *size()* – visszaadja, hogy hány elem van a veremben
- *top()* – verem legfelső elemének elérése
- *push()* – elem betétele a verembe
- *pop()* – verem legfelső elemének eldobása

Tételezze fel, hogy a sablonparaméterként átadott tároló megvalósítja a következő műveleteket: *back(), push_back(), pop_back(), empty(), size()*. Ezek funkciója, paraméterezése és működése megegyezik az STL tárolóknál megismert azonos nevű metódusokéval. Az adapter ne tároljon felesleges adatot! A tárló metódusai közvetlenül ne legyenek elérhetők! 5p

b) Hozzon létre két db *Stack* típusú objektumot (*obj1, obj2*) *long* típusú adatok tárolására! A két *Stack* objektum eltérő szabványos tárolót használjon! Töltse fel az egyik objektumot a standard inputról fájl végéig érkező adatokkal. 2p

c) Készítsen olyan generikus függvényt (*kiir*), amellyel egy tetszőleges *Stack* típusú objektum adatai kiírathatók a szabványos kimenetre! Az adatokat vesszővel válassza el egymástól! Feltételezheti, hogy a tárolt típusnak van inserter operátora. Ügyeljen arra, hogy a verem tartalma megmaradjon a kiírás után is! 3p

// vagy örököl, vagy delegál (ha mindkettő, akkor 0 pont)

Egy lehetséges megoldás delegációval:

```
Template<typename T, typename S = std::vector<T> >
class Stack {
    S s;
public:
    size_t size() const {
        return s.size();
    }
    bool empty() const {
        return s.empty();
    }
    void push(const T& d) {
        s.push_back(d);
    }
    void pop() {
        s.pop_back();
    }
    T& top() {
        return s.back();
    }
};
```

Egy lehetséges megoldás örökléssel:

```
template<typename T, typename S = std::vector<T> >
struct Stack : private S {
    size_t size() const {
        return S::size();
    }
    bool empty() const {
        return S::empty();
    }
    void push(const T& d) {
        S::push_back(d);
    }
    void pop() {
        S::pop_back();
    }
    T& top() {
        return S::back();
    }
};
```

b)

```
    Stack<long> obj1;
    Stack<long, std::deque<long> > obj2;
    long l;
    while (std::cin >> l)
        obj1.push(l);
```

b)

```
template<class T>
void kiir(T st) {
   while (!st.empty()) {
       std::cout << st.top();
       st.pop();
       if (!st.empty()) std::cout << ",";
   }
   std::cout << std::endl;
}
```

IMSC FELADAT:

d) Egészítse ki a *Stack* sablont olyan konstruktorral, ami a paraméterként kapott tárolóból feltölti a vermet. A paraméter típusa megegyezik a *Stack* tárolójának típusával. (Elegendő leírnia az új kódrészleteket.) 3p

e) Készítsen olyan **generikus** függvényt, ami a predikátuma által meghatározott elemeket eltávolítja a veremből. Ügyeljen az elemek sorrendjének megtartására! 5p

f) Rövid kódrészlettel mutassa be a d) és e) részfeladatokban elkészített sablonok használatát! 2p

d)

Delegációval:

```
    Stack() {}
    Stack(const S& t) : s(t){ }
```

Örökléssel:

```
 Stack() {}
    Stack(const S& t) {
       for (typename S::const_iterator i = t.cbegin(); i != t.cend(); ++i)
           push(*i);
    }
```

```
template<typename T, typename P>
void kihagy(T& st, P pred) {
    T tmp;
    while (!st.empty()) {
        if (!pred(st.top()))
            tmp.push(st.top());
        st.pop();
    }
    while (!tmp.empty()) {
        st.push(tmp.top());
        tmp.pop();
    }
}
```

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

```
class Gomb {
  CallBack& cb;
public:
  Gomb(CallBack& cb) :cb(cb) {}
  void Megnyom() { cb.call(); }
  virtual ~Gomb() {}
};
```

a) Definiáljon megfelelő absztrakt *CallBack* osztályt! (2p)

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

b) 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)

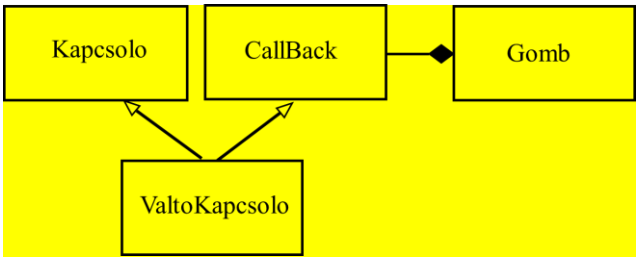
// Ha felvesz olyan attribútumokat, melyeket valamelyik ős tartalmaz, akkor nem érti az öröklés lényegét. 0p a részfeladatra

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

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

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

d) Rajzoljon UML osztálydiagramot, amin csak a fenti négy osztály neve szerepel! (1p)



500.AA Második Megoldó Kulcsár 500.A

4. Feladat **Σ 10 pont**

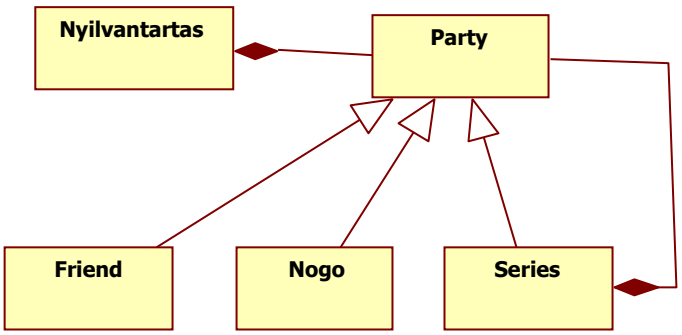
A Pulitikatudományi Intézet (PuT_In) nyilvántartást (*Nyilvantartas*) készít a különböző partikról (*Party*). Minden partinak van neve (*name*, string), és lekérhető, hogy mennyi pénzt költött piára (*getAmount()*). Több fajta partink lehet, és ezek száma később nőni fog. A *Friend* típusú parti például mindig 0-t ad vissza, ha a piapénzről kérdezzük, de van kedvenc focicsapata (*team*, string). A *Nogo* típusú parti egy külön attribútumban tartja nyilván, hogy melyik évben fogadta először a fő gyíkembert (*soros*, int). Vannak sorozatpartik, amelyek több másik partiból állnak (*Series*), ezeknél a *getAmount()* mindig a bennük levő partik *getAmount()*-jának összege. Egy Series-be a *join* metódussal lehet új partit felvenni. Minden parti ki tudja írni a megadott ostreamre az összes adatát (*report*). A *Series* típusúak rekurzívan a bennük levő partikét is.
A nyilvántartásba fel tudunk venni új partit (*add*), és ki is tudjuk listázni a meglevőket (*list*), aminek a során az egyes partik egyedi kiíratása fut le. Ha egy nyilvántartás megsemmisül, a tárolt adatok is elvesznek. Ugyanez igaz a *Series* típusú partikra is.

Tervezzen objektum-orientált megoldást a fenti leírás alapján a dőlt betűs megnevezések felhasználásával! Az attribútumok kivétel nélkül legyenek privátok és konstruktorban állíthatók. A megoldásban használja ki az STL adta lehetőségeket!
Az alábbi kódrészlet mutatja az egyes metódusok és konstruktorok paraméterezését és használatát.

```
Nyilvantartas ny3per1;
ny3per1.add(new Nogo(”Lofarok”, 1989));
ny3per1.add(new Friend(”Strasse”, ”FPÖ Vienna”));
Series* spangli = new Series("Spangli");
spangli->join(new Nogo(”420”, 2008));
spangli->join(new Nogo(”BobMarley”, 1981));
ny3per1.add(spangli);
ny3per1.list(std::cout);
```

Egy lehetséges megoldás:

Rajzoljon UML osztálydiagramot, amin ne szerepeljenek az attribútumok és a metódusok sem. (1p)



Definiálja az alábbi osztályokat, de a metódusoknak csak a deklarációját adja meg! (1.5p)

```
class Nyilvantartas {
    vector<Party*> parties;
public:
    void add(Party*);
    void list(ostream&) const;
    ~Nyilvantartas();
};
```

```
class Party {
    string name;
    double amount;
public:
    Party(string);
    virtual double getAmount() const;
    virtual void report(ostream&) const;
    virtual ~Party();
};
```

500.AA Második Megoldó Kulcsár 500.A

```
class Series : public Party {
    vector<Party*> parties;
public:
    Series(string n = "Noname");
    double getAmount() const;
    void report(ostream&) const;
    void join(Party*);
    ~Series();
};
```

```
class Nogo : public Party {
    int soros;
public:
    Nogo(string, int);
    double getAmount() const;
    void report(ostream&) const;
};
```

Implementálja külső definícióval a *Series* összes metódusát és konstruktorát, ami a példakód futtatásához szükséges! Ahol lehet, használjon iterátort!

```
Series::Series(string n) : Party(n) {}

void Series::join(Party *o) {
    parties.push_back(o);
}

void Series::report(ostream& os) const {
    Party::report(os);
    os << endl;
    for (vector<Parties>::iterator i = parties.begin();
         i < parties.end(); i++) {
        (*i)->report(os);
        os << endl;
    }
}

Series::~Series() { //1p ha iterátor
    for (vector<Parties>::iterator i = parties.begin();
         i < parties.end(); i++) {
        delete (*i);
    }
}
```