rendelkezésre álló idő:15 **perc**

Programozás alapjai 2. 5. beszámoló dolgozat. 2017.04.21. Kurz/**Terem**: Gy4/	Elért pontszám:
Név: **Neptun:**	

1. Készítsen **absztrakt** alaposztályt (*Log*) naplóbejegyzések heterogén kollekcióban történő tárolására! Az osztálynak ne legyen adattagja, csak egy *void print(std::ostream&)* metódusa, ami képes a paraméterként megadott adatfolyamra kiírni a konkrét naplóbejegyzés adatait! Az osztály felhasználásával hozzon létre egy konkrét naplóbejegyzést (*LastLog*), melynek print metódusa kiírja a konstruktorában kapott szöveget. A szöveg tárolására használja az *std::string* osztályt! (4p)

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

class LastLog: public Log {
   std::string txt;
public:
   LastLog(const char *txt) :txt(txt) {}
   void print(std::ostream& os) { os << txt; }
};
```

2. Tételezze fel, hogy rendelkezésére áll egy generikus tároló (*Store*<*T*>), ami tetszőleges számú generikus adatot tud tárolni. Adatot az *add()* tagfüggvénnyel lehet a tárolóba tenni. Tudjuk, hogy a tárolónak van iterátora (*Store*<*T*>*::iterator*), amivel az bejárható. Van iterátor értékű *begin()* és *end()* tagfüggvénye is. Az osztálynak csak paraméter nélkül hívható konstruktora van. A belső működéséről, további tagfüggvényeiről azonban semmit nem tudunk. Az osztály felhasználásával hozzon lére egy LogStore osztályt, ami képes az első feladatban megfogalmazott naplóbejegyzések heterogén kollekcióban történő tárolására, és a konkrét naplóadatok kiírására (*printAll()*)! A tároló megszűnése szüntesse meg a tárolt elemeket is! A *LogStore* osztály is legyen bejárható iterátorral, azaz legyen iterátora és *begin*/*end* tagfüggvénye is. Az alábbi kódrészlet a használatra mutat példát: (6p)

```
LogStore st;
st.add(new LastLog("user1"));
st.add(new LastLog("user2"));
st.add(new MyLog(100));      // a MyLog osztályt nem kell megvalósítania!
st.printAll(std::cout);      // minden tárolt naplóbejegyzés adatát kiírjuk
```

Célszerű, de nem kötelező olyan adapter megoldást választani, melyben *Store*<*T*> sablonból példányosodott osztály publikus tagfüggvényei elérhetőek a *LogStore* osztály interfészén is.

```
struct LogStore: Store<Log*> {
    void printAll(std::ostream& os) {
        iterator it = begin();
        while (it != end()) {
            (*it++)->print(os);
        }
    }
    ~LogStore() {
        iterator it = begin();
        while (it != end()) {
            delete (*it++);
        }
    }
};
```

BBBBBBBBBBBBBBBBBBBBBBBBBBBBB rendelkezésre álló idő:15 **perc**

Programozás alapjai 2. 5. beszámoló dolgozat. 2017.04.21. Kurz/**Terem**: Gy4/	Elért pontszám:
Név: **Neptun:**	

1. Készítsen **absztrakt** alaposztályt (*Naplo*) naplóbejegyzések heterogén kollekcióban történő tárolására! Az osztálynak ne legyen adattagja, csak egy *void kiir(std::ostream&)* metódusa, ami képes a paraméterként megadott adatfolyamra kiírni a konkrét naplóbejegyzés adatait! Az osztály felhasználásával hozzon létre egy konkrét naplóbejegyzést (*HibaNaplo*), melynek kiir metódusa kiírja a konstruktorában kapott szöveget. A szöveg tárolására használja az *std::string* osztályt! (4p)

```
struct Naplo {
    virtual void kiir(std::ostream&) = 0;
    virtual ~Naplo() {}
};

class HibaNaplo: public Naplo {
    std::string txt;
public:
    HibaNaplo(const char *txt) :txt(txt) {}
    void kiir(std::ostream& os) { os << txt; }
};
```

2. Tételezze fel, hogy rendelkezésére áll egy generikus tároló (*Tarolo<T>*), ami tetszőleges számú generikus adatot tud tárolni. Adatot a *felvesz()* tagfüggvénnyel lehet a tárolóba tenni. Tudjuk, hogy a tárolónak van iterátora (*Tarolo<T>::iterator*), amivel az bejárható. Van iterátor értékű *begin()* és *end()* tagfüggvénye is. Az osztálynak csak paraméter nélkül hívható konstruktora van. A belső működéséről, további tagfüggvényeiről azonban semmit nem tudunk. Az osztály felhasználásával hozzon lére egy *NaploTar* osztályt, ami képes az első feladatban megfogalmazott naplóbejegyzések heterogén kollekcióban történő tárolására, és a konkrét naplóadatok kiírására (*mindentKiir()*)! A tároló megszűnése szüntesse meg a tárolt elemeket is! A *NaploTar* osztály is legyen bejárható iterátorral, azaz legyen iterátora és *begin*/*end* tagfüggvénye is. Az alábbi kódrészlet a használatra mutat példát: (6p)

```
NaploTar tar;
tar.felvesz(new HibaNaplo("Baj Van"));
tar.felvesz(new HibaNaplo("Nagy Baj Van"));
tar.felvesz(new MyLog(100));      // a MyLog osztályt nem kell megvalósítania!
tar.mindentKiir(std::cout);      // minden tárolt naplóbejegyzés adatát kiírja
```

Célszerű, de nem kötelező olyan adapter megoldást választani, melyben *Tarolo<T>* sablonból példányosodott osztály publikus tagfüggvényei elérhetőek a *NaploTar* osztály interfészén is.

```
// Tartalmazott objektummal kicsit hosszabb:
class NaploTar {
    Tarolo<Naplo*> t;
public:
    typedef Tarolo<Naplo*>::iterator iterator;
    iterator begin() { return t.begin(); }
    iterator end() { return t.end(); }
    void felvesz(Naplo* naplo) { t.felvesz(naplo);}
    void mindentKiir(std::ostream& os) {
        iterator it = t.begin();
        while (it != t.end()) {
            (*it++)->kiir(os);
        }
    }
    ~NaploTar() {
        iterator it = t.begin();
        while (it != t.end()) {
            delete (*it++);
        }
    }
};
```