rendelkezésre álló idő:15 **perc**

Programozás alapjai 2. 5. beszámoló dolgozat. 2017.04.20. Kurz/**Terem**: Gy1/	Elért pontszám:
Név: **Neptun:**	

1. Készítsen **absztrakt** alaposztályt (*Event*) események heterogén kollekcióban történő tárolására! Az osztálynak ne legyen adattagja, csak egy *void print(std::ostream&)* metódusa, ami kiírja konkrét esemény adatait! Az osztály felhasználásával hozzon létre egy konkrét eseményt (*Msg*), ami kiírja a konstruktorában kapott szöveget. A szöveg tárolására használja az *std::string* osztályt! (4p)

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

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

2. Az *std::vector* tároló felhasználásával hozzon létre egy olyan tárolót (*EventList*), ami képes az előző feladatban megfogalmazott heterogén kollekciót tárolni! Legyen az *EventList* osztálynak *add()*, és *list()* metódusa, amivel új eseményt tehetünk a tárolóba, illetve ki tudjuk listázni a tárolóban levő események adatait. Az *EventList* osztály adatait közvetlenül ne lehessen elérni! A felsoroltakon és az implicit módon is keletkező tagfüggvényeken kívül más publikus függvénye ne legyen! Amennyiben egy *EventList* típusú objektum értékparaméterként átadásra kerülne, akkor dobjon *std::invalid_argument* hibát! A tároló megszűnése szüntesse meg a tárolt elemeket is! Az alábbi kódrészlet a használatra mutat példát: (6p)

```
EventList ev;
ev.add(new Msg("Tuz van"))
ev.add(new Msg("Nagy tuz van"));
ev.add(new Etc(13));                // az Etc osztályt nem kell megvalósítania!
ev.list(std::cout);
```

std::vector fontosabb tagjai: push_back(), pop_back(), front(), back(), size(), capacity(), reserve(), operator[], at(), insert(), erase(), iterator

```
class EventList {
  std::vector<Event*> t;
public:
  EventList() {}
  EventList(const EventList&) { throw std::invalid_argument("EventList"); }
  void add(Event *e) { t.push_back(e); }
  void list(std::ostream& os) {
    for (size_t i = 0; i < t.size(); i++)
       t[i]->print(os);
  }
  ~EventList() {
    for (size_t i = 0; i < t.size(); i++)
       delete t[i];
   }
};
```

BBBBBBBBBBBBBBBBBBBBBBBBBBBBB rendelkezésre álló idő:15 **perc**

Programozás alapjai 2. 5. beszámoló dolgozat. 2017.04.20. Kurz/**Terem**: Gy1/	Elért pontszám:
Név: **Neptun:**	

1. Készítsen **absztrakt** alaposztályt (*Esemeny*) események heterogén kollekcióban történő tárolására! Az osztálynak ne legyen adattagja, csak egy *void kiir(std::ostream&)* metódusa, ami kiírja konkrét esemény adatait! Az osztály felhasználásával hozzon létre egy konkrét eseményt (*Uzenet*), ami kiírja a konstruktorában kapott szöveget. A szöveg tárolására használja az *std::string* osztályt! (4p)

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

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

2. Az *std::vector* tároló felhasználásával hozzon létre egy olyan tárolót (*EsemenyLista*), ami képes az előző feladatban megfogalmazott heterogén kollekciót tárolni! Legyen az *EsemenyLista* osztálynak *betesz()*, és *kilistaz()* metódusa, amivel új eseményt tehetünk a tárolóba, illetve ki tudjuk listázni a tárolóban levő események adatait. Az *EsemenyLista* osztály adatait közvetlenül ne lehessen elérni! A felsoroltakon és az implicit módon is keletkező tagfüggvényeken kívül más publikus függvénye ne legyen! Amennyiben egy *EsemenyLista* típusú objektumból értékadással másolat készülne, akkor dobjon *std::invalid_argument* hibát! A tároló megszűnése szüntesse meg a tárolt elemeket is! Az alábbi kódrészlet a használatra mutat példát:

```
EsemenyLista lista;
lista.betesz(new Uzenet("Tuz van"));
lista.betesz(new Uzenet("Nagyon nagy tuz van"));
lista.betesz(new Egyeb(13));    // az Egyeb osztályt nem kell megvalósítania!
lista.kilistaz(std::cout);
```

std::vector fontosabb tagjai: push_back(), pop_back(), front(), back(), size(), capacity(), reserve(), operator[], at(), insert(), erase(), iterator

```
class EsemenyLista {
  std::vector<Esemeny*> t;
public:
  EsemenyLista& operator=(const EsemenyLista&) {
                          throw std::invalid_argument("EsemenyLista"); }
  void betesz(Esemeny *e) { t.push_back(e); }
  void kilistaz(std::ostream& os) {
     for (size_t i = 0; i < t.size(); i++)
        t[i]->kiir(os);
  }
  ~EsemenyLista() {
     for (size_t i = 0; i < t.size(); i++)
        delete t[i];
   }
};
```