454.AA Megoldó Bendegúz AA 454.

Programozás alapjai 2. (inf.) zárthelyi	2014.05.15. gyak. hiányzás: 0	kZHpont:
	ABCDEF **QA.F15/13.**	Hftest: 0 (p)

F.	Max.	Elért
1	3	
2	3	
3	4	
4	4	
5	6	
Σ	**20**	

Minden beadandó megoldását a feladatlapra, a feladat után írja! Készíthet piszkozatot, de csak a feladatlapra írt megoldásokat 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. kalkulátor, számítógép, notebook, mobiltelefon, Kindle) 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!*

A feladatra koncentráljon! Jó munkát! *Minimálisan elérendő pontszám: 8*

1. Feladat **6*0.5=3 pont**

Mit ír ki a szabványos kimenetre az alábbi program? Válaszához használja a négyzetrácsos területet! Abba a sorba írjon, ahol a kiírás keletkezik! Figyeljen a változók élettartamára!

```
#include <iostream>
inline std::ostream& Pr() { return std::cout << std::endl; }
template <typename T> std::ostream& Pr(T v) { return std::cout << v; }

struct Adat5 {
  int i;
  Adat5 (int i = -1) :i(i)          { Pr(i) << "k"; }
  Adat5 (const Adat5& a) :i(a.i)    { Pr(i) << "c"; }
  Adat5& operator+=(int n)          { i += n; Pr(i) << "+="; return *this; }
  virtual ~Adat5 ()                 { Pr("d"); }
};
struct Uj : public Adat5{
  Uj(int a = -5, const char *m = "cde"): Adat5 (a)  { Pr(m) << "K"; }
  Uj(const Uj& u) :Adat5 (u)                        { Pr("C"); }
  ~Uj()                                             { Pr("D"); }
};

int main() {
  Adat5 a(13);                      Pr();
  a += 3;                           Pr();
  Uj *p = new Uj(4, "ABCDEF");      Pr();
  Uj uj5 = *p;                      Pr();
  Adat5 *pa = p;                    Pr();
  delete p;                         Pr();
  return 0;
}
```

1	3	k												
1	6	+	=											
4	k	A	B	C	D	E	F	K						
4	c	C												
D	d													
D	d	d												

2. Feladat **Σ 3 pont**

2.a) Készítsen olyan függvénysablont (*osszegez*), ami képes hozzáadni egy sorozattároló elemeit egy adott kezdőértékhez! A sablon használatára mutat példát az alábbi kódrészlet.

```
int t[] = { 1, 2, 3, 4, 5, 1 };
std::vector<double> v(t, t+6);
std::cout << osszegez(t, t+6, 0) << std::endl;  // kezdőérték: 0
std::cout << osszegez(v.begin(), v.end(), 2.1) << std::endl;  // kezdőérték: 2.1
```

A fenti kódrészlet 16 és 18.1 értéket ír ki a szabványos kimenetre.

2.b) Határozza meg, hogy megoldása milyen követelményeket (műveleteket) támaszt a függvényparaméterként átadott adattípusokkal szemben?

2.c) Készítsen egy újabb változatot a függvénysablonból, ami paraméterként egy predikátumot is átvesz. Ezzel döntse el, hogy az adott elemmel el kell-e végeznie a műveletet! Mutassa be az újabb változat használatát a fenti adatokon úgy, hogy az csak a páratlan elemekkel végezzen műveletet!

Egy lehetséges megoldás:

2a)

```
template<class I, class T>
T osszegez(I first, I last, T v) {
  while (first != last)
    v = v + *first++;
  return v;
}
```

2b)Követelmények:

I-vel szemben: másoló, destruktor, post inkremens, deref, op!=

T-vel szemben: másoló, destruktor, op.T+T, op.T=T,

2c)

```
template<class I, class T, class F>
T osszegez(I first, I last, T v, F pred) {
  while (first != last) {
    if (pred(*first)) v = v + *first;
      first++;
  }
  return v;
}
bool odd()(int v) { return v % 2 == 1; }
std::cout << osszegez(t, t+6, 0, odd) << std::endl;
```

---

3. Feladat Σ 4 pont

Karaktersorozatok dinamikus tárolására alkalmas string osztályt kell létrehozni String_5 néven. Meg kell valósítani az alábbi műveleteket:

operator[]	Egy karakter direkt elérése (indexelés). hibás indexelés esetén `std::out_of_range` kivételt dob.
operator+	két string összeadása (összefűzés)
operator+	string + karakter összeadása (karaktert a string végére fűzi)
operator+=	string + karakter összeadása (összefűzés), az **eredmény a bal oldali op.**
operator+=	két string összeadása (összefűzés), az **eredmény a bal oldali op.**
length	string hosszát adja
(const char*)	C stílusú stringre konvertál
find	karaktersorozat első előfordulásának indexét adja vissza
erase	törli a stringet
is_null	igaz értéket ad, ha a string üres (**""**)

Az osztály megvalósításán többen dolgoznak egyszerre, akikkel megállapodtak az osztály belső adatszerkezetében, és a tagfüggvények funkcióiban. Úgy döntöttek, hogy a karaktersorozatot lezáró **nullával együtt** tárolják, de a tárolt hosszba (`len`) a nullát nem számolják bele. Abban is megállapodtak, hogy ha lehet, használják a C stringkezelő függvényeit `<cstring>`. Azt is megbeszélték, hogy az üres stringet (**""**) is tárolják. A megállapodást az alábbi kommentezett deklaráció rögzíti, amin **nem lehet változtatni**. Követelmény, hogy az osztály legyen átadható érték szerint függvényparaméterként, és működjön helyesen a többszörös értékadás is.

```
class String_5 {
protected:
  char *str;      // Pointer a dinamikus adatterületre. Ezen a területen tároljuk a karaktereket.
  size_t len;     // A string tényleges hossza, amibe a lezáró nulla nem számít bele.
public:
  String_5(const char *s = "");       // Létrehoz stringet s-ből. Feltételezhető, hogy s soha sem null
  String_5(const String_5&);          // Másoló konstruktor.
  String_5& operator=(const String_5&);// értékadó (string = string)
  String_5& operator+=(char);         // értékadó (string += chat)
  String_5& operator+=( const String_5&);// értékadó (string += string)
  char& operator[](size_t);           // indexoperátor. Hibát dob, ha az index rossz
  String_5 operator+(char) const;     // string + char
  String_5 operator+(const String_5&) const;  // string + string
  operator const char*() const;       // C stílusú stringre konvertál
  size_t length() const;              // Visszaadja a string tényleges hosszát;
  size_t find(const char) const;      // egy karakter első előfordulását keresi, ha nem talál, -1-et ad vissza
  void erase();                       // törli a stringet
  bool is_null();                     // igaz értéket ad, ha a string üres ("")
  ~String_5();                        // megszünteti az objektumot
};
```

A tagfüggvények elkészítését felosztották egymás között. Önre **a másoló konstruktor, az indexoperátor és a length** tagfüggvény megírása jutott. **Valósítsa** meg a feladatul kapott tagfüggvényeket! Vegye figyelembe, hogy a mások által írt függvények belső megoldásait nem ismeri, azaz nem használhat ki olyan működést, ami a fenti kódrészletből, vagy annak megjegyzéseiből nem olvasható ki.

Az „A” csoport feladatának egy lehetséges megoldása:

```
String_5::String_5(const String_5& rhs) {
  len = rhs.len;
  str = new char[len+1];
  strcpy(str, rhs.str);
}
char& String_5::operator[](size_t i) {
  if (i >= len) throw std::out_of_range("String_5: index hiba");
  return str[i];
}

size_t String_5::length() const { return len; }
```

A „B” csoport feladatának egy lehetséges megoldása:

```
String_5& String_5::operator=(const String_5& rhs) {
  if (this != &rhs) {
    delete[] str;
    len = rhs.len;
    str = new char[len+1];
    strcpy(str, rhs.str);
  }
  return *this;
}
void String_5::erase() {
  if (len != 0) {
    len = 0;
    delete[] str;
    str = new char[1]; // üres string kell, hogy a többi fv. jól működjön
    str[0] = 0;
  }
}
size_t String_5::find(const char rhs) const {
  const char *p = strchr(str, rhs);
  if (p == NULL) return -1;
  return p-str;
}
```

A „C” csoport feladatának egy lehetséges megoldása:

```
String_5& String_5::operator+=(const String_5& rhs) {
  len += rhs.len;
  char *p = new char[len+1];
  strcpy(p, str);
  strcat(p, rhs.str);
  delete[] str;
  str = p;
  return *this;
}

size_t String_5::length() const { return len; }

String_5::operator const char*() const {
  return str;
}
```

A „D” csoport feladatának egy lehetséges megoldása:

```
String_5& String_5::operator+=(const char rhs) {
  len += 1;
  char *p = new char[len+1];
  strcpy(p, str);
  p[len-1] = rhs;
  p[len] = 0;
  delete[] str;
  str = p;
  return *this;
}

String_5::~String_5() { delete[] str; }

size_t String_5::length() const { return len; }
```

---

4. Feladat Σ 4 pont

Tételezze fel, hogy rendelkezésére áll egy generikus tároló osztály (*Tarolo*), ami pontosan olyan publikus interfésszel rendelkezik, mint az *std::list!*

- A *Tarolo* osztály felhasználásával **hozzon** létre egy olyan új generikus tárolót (*UjTarolo*), ami pontosan úgy viselkedik, mint a *Tarolo* osztály, és van olyan tagfüggvénye (*kiir*), ami képes kiírni a tároló taralmát a paraméterként kapott *std::ostream* objektumra! A kiírás sorrendjét és szerkezetét (soronként, egy sorba, ...) Ön határozhatja meg. A kiírás után a tároló tartalma ugyanaz maradjon, mint a kiírás előtt volt!
- Ügyeljen arra, hogy a *Tarolo* minden tagfüggvénye (a konstruktorok is) elérhető legyen!
- Határozza meg, hogy megoldása milyen új követelményeket támaszt a generikus adattal szemben!
- Az elkészített *UjTarolo* sablon felhasználásával hozzon létre egy *double* elemeket tartalmazó tárolót! Olvasson be a szabványos bemenetről fájl végéig értékeket ebbe a tárolóba, majd törölje ki az átlagnál kisebb értékeket, végül írja ki a tároló tartalmát a szabványos kimenetre!

A feladat megoldásához használhat STL algoritmust és/vagy tárolót.

Egy lehetséges megoldás:

```
template <class T>
class UjTarolo : public Tarolo<T> {
public:
  UjTarolo(size_t n = 0, const T& v = T()) : Tarolo<T>(n, v) {}
  template <class I>
  UjTarolo(I first, I last) : Tarolo<T>(first, last) {}
  void kiir(std::ostream& os) {
    ostream_iterator<T> out(os, ",");
    copy(this->begin(), this->end(), out);
  }
};
```

```
int main() {
  UjTarolo<double> t;
  double x;

  while (cin >> x) t.push_back(x);
  x=0;
  x = accumulate(t.begin(), t.end(), x);
  x = x / t.size();
  t.erase(remove_if(t.begin(), t.end(), bind2nd(less<double>(), x)), t.end());
  t.kiir(cout);
  return 0;
}
```

Alternatív megoldás a törlésre:

```
typedef UjTarolo<double>::iterator iter;
for (iter it = t.begin(); it != t.end(); ) {
  if ((*it) < x)
    it = t.erase(it); // Törléskor az iterátorok érvénytelenné válnak!
                      // Az erase visszaad egy érvényeset a köv. elemre.
  else
    ++it;
 }
```

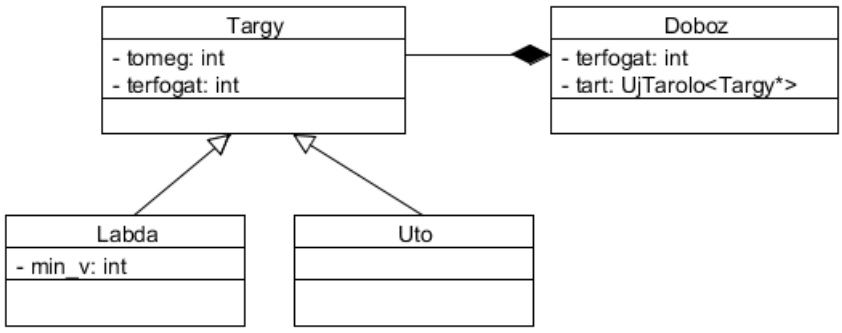
5. Feladat Σ 6 pont

Tárgyakat (*Targy*) tárolunk dobozokban (*Doboz*). A tárgyaknak lekérdezhető a tömege (*getTomeg*) [g, egész] és a térfogata (*getTerfogat*) [cm^3, egész]. Egyelőre kétféle tárggyal foglalkozunk: labdákkal (*Labda*) és ütőkkel (*Uto*) de ez később bővülni fog. Ha a *Doboz* megsemmisül (pl. leég), az összes benne tárolt tárgy is megsemmisül. A rendszerben a következő műveleteket kell megvalósítani:

- Labda létrehozása, adatainak lekérdezése. A labda tömege állandó, de a tömeg lekérdezésekor a térfogat csökken 1 cm^3-rel, ha még nem ért el egy minimális térfogatot. Konstruktorában legyen megadható a labda tömege, minimális és pillanatnyi térfogata.
- Ütő létrehozása. Az ütő tömege és térfogata állandó. Konstruktorában legyen megadható az ütő tömege és térfogata!
- Új tárgy dobozba helyezése (*berak*). Ez csak akkor sikerül, ha a dobozban levő tárgyak térfogatának összege az új tárggyal együtt is kisebb lesz, mint a doboz térfogata. Ellenkező esetben a függvény dobjon *std::overflow_error* hibát.
- Dobozban levő tárgyak össztömegének lekérdezése (*getOsszTomeg*).
- Doboz megsemmisítése.

Feladatok:

- A 4. Feladatban megvalósított *UjTarolo* osztály **felhasználásával tervezzen** olyan OO modellt, ami könnyen bővíthető további tárgyakkal. Rajzolja fel a modell osztálydiagramját! Ne tüntesse fel a tagfüggvényeket az UML ábrán! Használja a dőlt betűs neveket!
- **Deklarálja** és **implementálja** a *Doboz*, *Targy*, *Labda*, *Uto* osztályokat és tagfüggvényeit! A *Doboz* osztály ne legyen másolható, és az értékadás művelet használata is okozzon fordítási hibát! Használja a dőlt betűs neveket!
- **Írjon** egy olyan kódrészletet, ami különböző tárgyakat dinamikusan létrehoz és beteszi azt a dobozba!



```
class Targy {
  int tomeg, terfogat;
public:
  Targy(int m, int v):tomeg(m), terfogat(v) {}
  void setTonmeg(int m) { tomeg = m ; }
  void setTerfogat(int v) { terfogat = v ; }
  virtual int getTomeg() { return tomeg; }
  virtual int getTerfogat() { return terfogat; }
  virtual ~Targy() {}
};
class Doboz {
  int terfogat;
  UjTarolo<Targy*> tart;
  typedef typename UjTarolo<Targy*>::const_iterator iter;
  Doboz(const Doboz&);
  Doboz& operator=(const Doboz&);
public:
  Doboz(int v) :terfogat(v) {}
  void berak(Targy *t);
  int getOsszTomeg() const ;
  ~Doboz();
};

class Labda: public Targy {
  int min_v;
public:
  Labda(int m, int v, int v0):Targy(m,v),min_v(v0) {}
  int getTomeg() {
    int v = Targy::getTerfogat();
    if (v > min_v) Targy::setTerfogat(v-1);
    return Targy::getTomeg();
  }
};

class Uto: public Targy {
public:
   Uto(int m, int v) : Targy(m, v) {}
}

void Doboz::berak(Targy *t) {
  int sum = 0;
  for(iter it=tart.begin(); it!=tart.end(); ++it)
    sum += (*it)->getTerfogat();
  if (sum >= terfogat) throw std::overflow_error("nem fer be");
  else tart.push_back(t);
}

int Doboz::getOsszTomeg() const {
  int sum = 0;
  for(iter it=tart.begin(); it!=tart.end(); it++)
    sum += (*it)->getTomeg();
  return sum;
}

Doboz::~Doboz() {
  for(iter it=tart.begin(); it!=tart.end(); it++)
    delete (*it);
}

  Doboz d(10);
  d.berak(new Labda(1,2,1));
  d.berak(new(Uto(2,4));
```