500.AA Első Megoldó Kulcsár 500.A

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

f.	max.	elért	jav.
1.	10		
2.	10		
3.	10		
4.	10		
Σ	**40**		
IM.	10		

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**.*

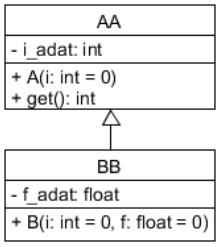
Az első feladatban minimum 5 pontot el kell érnie ahhoz, hogy a többi feladatot értékeljük.

1. feladat: Beugró **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. (2p)

Referencia típusú tagváltozót konstruktor törzsében lehet inicializálni.	I	**H**
B osztály kompatibilis A-val, ha B mindenütt használható, ahol A használható.	**I**	H
A destruktor mindig meghívja a tartalmazott objektumok destruktorait.	**I**	H
A konstruktor előbb hajtja végre a programozott törzset, és csak ezután hívja a tartalmazott objektumok konstruktorait.	I	**H**

b) Valósítsa meg C++ nyelven az alábbi osztálydiagramon szereplő osztályokat! A tagváltozók kezdeti értékét a konstruktorparaméterek adják. . A diagramon külön nem szerepel a destruktor, de arról se feledkezzen el! (2.5p)




```
class AA {
    int i_adat;
public:
    AA(int i = 0) :i_adat(i) {}
    int get() { return i_adat; }
    virtual ~AA() {}
};
class BB : public AA {
};
    int f_adat;
public:
    BB(int i = 0, float f = 0) : AA(i), f_adat(f) {}
};
```

c) Készítsen olyan *inserter* operátort, ami a **b)** feladatban készített ***AA*** osztály adatát (*i_adat*) kiírja egy *std::ostream* típusú adatfolyamra! Az operátor legyen fűzhető! (Ügyeljen az adattag láthatóságára!) Egy **kódrészlettel mutassa** be az operátor használatát! Megjegyzésben adja meg azt is, hogy mit ír ki a kódrészlet az adatfolyamra (2p)

```
std::ostream& operator<<(std::ostream& os, const AA& p) {
  return os << p.get();
}
std::cout << A(1); // 1
```

d) A **c)** részfeladatban elkészített inserter alkalmazható-e egy BB típusú objektumra? Mi történik, ha megpróbáljuk? (1p)

Igen, kiírja az AA adattagját!

e) Írjon programrészletet, melyben a dinamikus memóriaterületen létrehoz egy olyan területet, ami alkalmas arra, hogy segítségével elérjünk maximum 100 db b) feladat osztályaiból példányosított objektumot (AA és BB típusú objektumokat vegyesen)! Ezután **olvasson** be a szabványos bemenetről egy egész számot (*n*), majd a dinamikus memóriaterületen **hozzon létre** *n* darab *AA*, és 100-*n* darab *BB* típusú objektumot. Végül szabadítson fel minden memóriaterületet! Feltételezheti, hogy 0 <= *n* <= 100! (2.5p)

```
AA **p = new AA*[100];
int i = 0, n; std::cin >> n;
while (i < n) p[i++] = new AA;
while (i < 100)  p[i++] = new BB;
for (i = 0; i < 100; i++) delete p[i];
delete[] p;
```

2. Feladat **Σ 10 pont**

Tételezze fel, hogy egy olyan csapatban dolgozik, melynek egy olyan osztályt kell létrehoznia, amely karaktersorozatok dinamikus tárolására alkalmas. 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. Indexelési hiba esetén dobjon *std::out_of_range_error* kivételt!

```
class String {
  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 (size_t siz = 0);            // siz hosszú üres stringet hoz létre
  String (const char *s);             // Létrehoz stringet s-ből. Feltételezhető, hogy s soha sem NULL
  String (const String&);             // Másoló konstruktor.
  String& operator=(const String&);   // értékadó (string = string)
  String operator+(char);             // új stringet hoz létre, melyben a karaktert string végéhez fűzi
  char& operator[](size_t);           // indexoperátor. Hibát dob, ha az index rossz
  char operator[](size_t) const;      // indexoperátor. Hibát dob, ha az index rossz
  const char*c_str() const;           // C stílusú stringet ad vissza
  size_t size() const;                // Visszaadja a string tényleges hosszát;
  void erase();                       // törli a stringet
  bool is_null();                     // igaz értéket ad, ha a string üres ("")
  ~String ();                         // megszünteti az objektumot
};
```

A tagfüggvények elkészítését felosztották egymás között. Önre **a másoló konstruktor, az értékadó operator, a destruktor, az indexoperátorok és a size** 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.

Egy lehetséges megoldás:

```
String::String(const String& rhs) {
  len = rhs.len;
  str = new char[len+1];
  strcpy(str, rhs.str);
}

String& String::operator=(const String& rhs) {
  if (this != &rhs) {
    delete[] str;
    len = rhs.len;
    str = new char[len+1];
    strcpy(str, rhs.str);
  }
  return *this;
}

char& String::operator[](size_t i) {
  if (i >= len) throw std::out_of_range("String: index hiba");
  return str[i];
}

char String::operator[](size_t i) const {
  if (i >= len) throw std::out_of_range("String: index hiba");
  return str[i];
}

size_t String::size() const { return len; }

String::~String() { delete[] str; }
```

500.AA Első Megoldó Kulcsár 500.A

Pontozás:

A másoló és az op = **3-3 pont**
Többi függvény: **1-1 pont**.
Durva pointerkezelési hiba **-1pont**
delete[] helyett delete: **-0.5pont**
memory leak: -1p

IMSC feladat:

b) Valósítsa meg az oparator+(char) metódust is!

c) Tételezze fel, hogy elkészült a *String* osztály és helyesen működik! Annak módosítása nélkül **készítsen** *inserter* és *exractor* operátort, melyek alkalmasak egy *String* szabványos adatfolyamra történő kiírására, illetve adatfolyamról történő beolvasására! Beolvasáskor a bevezető *whitespace* karaktereket dobja el és a beolvasást az első *whitespace* karakternél hagyja abba! Az operátorok legyenek fűzhetőek. A beolvasáshoz segítségül megadjuk az *isteram* néhány tagfüggvényét (nem biztos, hogy mindet kell használnia):

`istream& get (char& c);`	- beolvas egy karaktert az adatfolyamról.
`putback(char c);`	- visszateszi a stream bufferbe az onnan beolvasott karaktert, így újból beolvasható
`int peek();`	- az adatfolyam következő karakterét adja anélkül, hogy azt kivenné az adatfolyamból file vége esetén EOF értéket ad.

d) Tételezze fel, hogy elkészült az *inserter*, és jól működik! Mit ír ki az alábbi kódrészlet a szabványos kimenetre?

```
std::cout << String('A') + 'C';
```

a) (1.5P)

```
String String::operator+(char ch) { // const fv kelene, de nem const lett
    String tmp(len+1);
    strcpy(tmp.str, str);
    tmp.str[len] = ch;
    tmp.str[len+1] = '\0';
    return tmp;
}
```

c)

```
std::ostream& operator<<(std::ostream&os, const String& s) {
    return os << s.c_str();
}

std::istream& operator>>(std::istream&is, String& s) {
    char ch;
    String tmp;
    s.erase();
    while (is.get(ch) && iswspace(ch));
    if (!is) return is;
    do {
      tmp = tmp + ch;
    } while (is.get(ch) && !iswspace(ch));
    s = tmp;
    return is ;
}
```

d) A size_t típusú konstruktor fut le. Nem 1db karaktert tesz el. Szemetet, vagy semmit sem ír ki.

3. Feladat Σ 10 pont

a) Készítsen generikus függvényt (***Kiir***), ami kiírja a szabványos kimenetre egy paraméterként kapott indexelhető objektum azon elemeit, amelyek megfelelnek a szintén paraméterként kapott predikátummal megadott feltételnek! A függvény paraméterként vegye át az elemek számát, és egy szeparátort (const char*) is! A kiíráskor az egyes elemeket a szeparátorral válassza el egymástól (az utolsó elem után is lehet szeparátor)! A kiírást soremeléssel zárja!

b) Készítsen olyan függvényt, ami a fenti függvénysablon predikátumaként alkalmazható és képes eldönteni, hogy egy egész szám pozitív-e!

c) Írjon kódrészletet, melyben beolvas maximum 500 valós számot egy tömbbe, majd az **a)** részfeladatban elkészített függvénysablon és a **b)** részfeladatban elkészített predikátum segítségével kiírja a beolvasott számok közül a pozitív elemeket a szabványos kimenetre!

d) Rövid kódrészlettel **mutassa meg**, hogy hogyan lehetne használni függvénysablont a 2. feladatban elkészített *String* osztályból létrehozott objektumokra! Készítsen ehhez a részfeladathoz olyan predikátumot, ami a számjegyeket szűri! Tegye fel, hogy van olyan *extractor*, amivel *String* típusú objektumba lehet beolvasni szavakat! Olvasson be file végéig szavakat és írja ki azokat úgy, hogy a sámjegyeket ne írja ki.

```
template <typename T, typename P>
void Kiir(const T& t, int n, P pred, const char *sep ="") {
    for (int i = 0; i < n; i++)
        if (pred(t[i]))
            std::cout << t[i] << sep;
    std::cout << std::endl;
}

bool ispos(double d) { return d > 0; }

int tomb[140];
int db = 0;
while (db < 140 && std::cin >> tomb[db])
    db++;

Kiir(tomb, db, ispos, ",");

bool notanumber(char ch) { return !isdigit(ch); }

String str;
while (std::cin >> str)
    Kiir(strr, str.size(), notanumber);
```

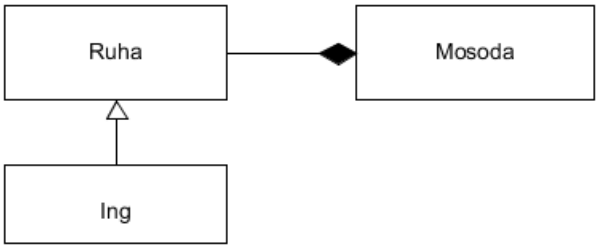
4. Feladat Σ 10 pont

Egy mosoda (*Mosoda*) működését szeretnénk modellezni. Az érkező koszos ruhákat (*Ruha*) a mosoda azonnal kimossa és eltárolja. A ruháknak van tulajdonosuk (*string*). Mosáskor (*mos*) a ruhák koszos állapotból tiszta állapotba kerülnek. Az általános ruhafajtán kívül van a színes ing (*Ing*), ami minden alkalommal 3%-ot veszít a színéből (*double*). Ezen kívül a jövőben további speciális ruhafajtákat tervezünk bevezetni (pl. kétrészes ruha). Ha a mosoda megsemmisül (pl. leég), az összes még ki nem adott ruha is megsemmisül. A rendszerben a következő műveleteket kell megvalósítani:

- új ruha beadása a mosodába (*bead*);
- adott tulajdonos tiszta ruhájának kiadása a *Mosoda*-ból (*kiad*); ha nincs ilyen ruha, dobjon egy standard kivételt;
- mosoda megsemmisülése;
- ruha adatainak kiírása (*kiir*); ing esetén a szín erősségét is írja ki;
- ruha tisztítása (*mos*).

Feladatok:

- **Tervezzen** olyan OO modellt, ami könnyen bővíthető további ruhafajtákkal. Feltételezheti, hogy a mosodában egyszerre maximum 100 ruha lehet. Rajzolja fel a modell osztálydiagramját! Ne tüntesse fel a tagfüggvényeket és adattagokat az UML ábrán! Használja a dőlt betűs neveket!



- **Adja meg** a *Ruha* osztály összes virtuális metódusának deklarációját!

```
virtual void Ruha::mos();
virtual void Ruha::kiir();
virtual Ruha::~Ruha();
```

- **Deklarálja** a *Mosoda* osztályt. A tagfüggvényeknek is csak a deklarációját adja meg.

```
class Mosoda {
    Ruha* ruhak[100];
    int db;
public:
    Mosoda()
    ~Mosoda()
    void bead(Ruha*);
    Ruha* kiad(std::string);
};
```

- **Adja meg** az *Ing* osztály *kiir* metódusának külső definícióját (valósítsa meg), feltételezve, hogy a Ruha osztálynak minden attribútuma privát!

```
void Ing::kiir() {
   Ruha::kiir();
   std::cout << szin << std::endl;
}
```

- **Írjon** egy olyan kódrészletet, ami különböző ruhákat dinamikusan létrehoz, bead a mosodába, és kiadat egy kimosott ruhát, amelynek az adatait kiírja! A kivett ruhát ezután szüntesse meg!

```
Mosoda mosoMasa;
mosoMasa.bead(new Ruha("Jozsi"));
mosoMasa.bead(new Ing("Feri", 23.2));
try {
   Ruha *tisztaRuha = mosoMasa.kiad("Feri");
   tisztaRuha->kiir();
   delete tisztaRuha;
} catch (std::exception& e) {
   std::cout << e.ahat();
}
```