485. AA Megoldó Alfréd AA 485.

Programozás alapjai 2. (inf.) zárthelyi	2011.05.12. gyak. hiányzás: 0	kZHpont: 0
MEG123	**OTT/1.**	Hftest: **0**

Minden beadandó megoldását a feladatlapra, a feladat után írja! *A megoldások során feltételezheti, hogy minden szükséges input adat az előírt formátumban rendelkezésre áll.*
A feladatok megoldásához csak a letölthető C és C++ összefoglaló használható. Számítógép, notebook, menedzser kalkulátor, organiser, mobiltelefon nem használható.

*A feladatokat **figyelmesen olvassa** el, megoldásukhoz **ne használjon fel STL** tárolót, kivéve, ha a feladat ezt külön engedi! **Ne írjon felesleges** függvényeket ill. kódot, a feladatra koncentráljon! Jó munkát!* *Értékelés: 0-8:1, 9-11:2, 12-14:3, 15-17:4, 18-20:5*

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

1. Feladat 6*0.5=3 pont

Mit ír ki a szabványos kimenetre az alábbi program? Válaszához használja a négyzetrácsos területet!

```
#include <iostream>
using namespace std;
class Alap {
  int alap;
public:
  Alap(int i) :alap(i)                          { cout << alap << 'k';}
  Alap(const Alap& a) :alap(a.alap)             { cout << alap << 'c';}
  operator bool() { return alap == 0; }
  ~Alap()                                       { cout << 'd'; }
};

class Masik {
  Alap a;
  const char *m;
public:
  Masik(int a, const char *m) :a(a), m(m)  { cout << m << 'M'; }
  void operator=(const Masik& e) { a = e.a; cout << '=' << e.m; }
  ~Masik()                                      { cout << 'D'; }
};

int main() {
  Masik m1(1, "MEG123");  cout << endl;
  Masik m2 = m1;          cout << endl;
  m1 = m2;                cout << endl;
  Alap a1(0);             cout << endl;
  if (a1)        cout << "g12" << endl;
  else            cout << "me" << endl;
  return(0);
}
```

1	k	M	E	G	1	2	3	M						
1	c													
=	M	E	G	1	2	3								
0	k													
g	1	2												
*														
d	D	d	D	d										

*** Az utolsó sort itt is elfogadtuk, mivel az if egyik ágán az endl nem íródik ki.**

2. Feladat Σ 3 pont

Írjon függvénysablont (***my_replace_if***), ami az STL *replace_if* sablonjához hasonlóan a függvény 4. paramétereként megadott értékre állítja egy tároló azon elemeit, melyek a 3. paraméterben átadott predikátumnak eleget tesznek! A függvény első két paramétere két előrehaladó iterátor, ami kijelöli a jobbról nyílt intervallum kezdetét és végét. A függvény *void* visszatérési értékű. Amennyiben helyesen oldja meg a feladatot, akkor az alábbi kódrészlet a következőt írja ki: **1, 2, 3, 8, 8, 8,**

```
int v[] = { 1, 2, 3, 17, 18, 6 };
my_replace_if(v, v+6, bind2nd(greater<int>(), 5), 8);
for (int *p = v; p != v+6; ++p)
   cout << *p << ", ";
```

Készítsen egy olyan függvényobjektum sablont (**greater_than**), amit a fenti példában így lehetne alkalmazni:

```
my_replace_if(v, v+6, greater_than<int>(5), 8);
```

Egy lehetséges megoldás:

```
template <class Iter, class Pred, class T>
void my_replace_if(Iter first, Iter last, Pred pred, T v) {
    for (; first != last; first++) {
        if (pred(*first))
            *first = v;
    }
}
template <class T>
class greater_than {
    T ref;
public:
    greater_than(T ref) :ref(ref) {}
    bool operator()(T a) {
        return a > ref;
    }
};
```

---

3. Feladat Σ 5 pont

Csapatának egy Verem osztályt kell létrehoznia egész számok tárolására. A csapattagokkal megállapodtak az osztály publikus interfészében, valamint abban, hogy a verembe tehető adatok számát nem korlátozzák, a tárolás dinamikus. A megállapodást az alábbi kommentezett deklaráció rögzíti. A deklarációt **a pontozott részen** kiegészítheti, de a **publikus** interfészt nem változtathatja meg! Követelmény, hogy az osztály legyen átadható érték szerint függvényparaméterként, és működjön helyesen a többszörös értékadás is.

```
class Verem private deque<int> {
public:
  Verem(int n = 0, int val = 0); // Létrehoz egy vermet n db val értékkel (minden eleme val)
  void push(int);                // Verembe tesz egy egész értéket.
  int  pop() throw(range_error); // Kiveszi verembe utoljára betett értéket;
                                 // üres verem esetén range_error() kivételt dob.
  bool empty() const;            // Igaz, ha üres a verem.
  void dup() throw(range_error); // A verem tetején levő elemet megduplázza (újból beteszi);
                                 // üres verem esetén range_error() kivételt dob
  void swap()throw(range_error); // A verem tetején levő két elemet megcseréli;
                                 // üres verem esetén range_error() kivételt dob.
  int size() const;              // Visszaadja a veremben levő elemek számát.
  ~Verem();
   ........................
   ........................
   ........................
};
```

Egészítse ki a fenti deklarációt úgy, hogy az az elvárásoknak megfelelően működjön! A megoldáshoz felhasználhatja az STL bármelyik tárolóját ill. algoritmusát! (2p) Ügyeljen arra, ha származtatással oldja meg a feladatot, akkor is csak a jelenlegi interfész legyen publikusan elérhető!

A tagfüggvények közül **valósítsa meg** a következőket: *destruktor*, *konstruktorok*, *pop*, *empty*!

Célszerűen egy STL tárolóból származtatunk private vagy protected örökléssel:

```
Verem::~Verem() {}

Verem::Verem(int n, int val) {
  for (int i = 0; i < n; i++)
    push(val);
}

int Verem::pop() throw(range_error) {
  if (empty()) throw range_error("Verem::pop - empty");
  int tmp = back();
  pop_back();
  return tmp;
}

inline bool Verem::empty() const {
  return deque<int>::empty();
}
```

Tartalmazott STL, vagy saját tárolóval is megoldható. Ez utóbbival többet kellett írni.

További tagfüggvények (ebben a csoportban nem kellett megírni):

```
inline void Verem::push(int val) {
    push_back(val);
}

void Verem::dup() throw(range_error) {
  if (empty()) throw range_error("Verem::dup - empty");
  push(back());
}

void Verem::swap() throw(range_error) {
  if (size() < 2) throw range_error("Verem::swap - empty");
  int tmp1 = pop();
  int tmp2 = pop();
  push(tmp1);
  push(tmp2);
}

inline int Verem::size() const {
  return deque<int>::size();
}
```

4. Feladat Σ 4 pont

Tételezze fel, hogy a **3. feladat** *Verem* osztálya elkészült és hibátlanul működik! Ezen osztály **felhasználásával deklaráljon** egy *Szamologep* osztályt elemi aritmetikai műveletek (*osszead, kivon, szoroz, oszt*) végzésére. A műveletvégző tagfüggvények a verem két legfelső elemét kiveszik, elvégzik a műveletet, majd az eredményt a verem tetejére teszik! Legyen az osztálynak egy *kiir* tagfüggvénye is, ami kiírja a legfelső elemet a standard kimenetre, de kiírt érték a veremben is maradjon meg! Egy *istream* típusú objektumból legyen lehetőség egész számokat beolvasni a szokásos >> operátorral! **Valósítsa** meg az osztály *osszead* és *kiir* tagfüggvényét, valamint a >> *operátort*! Működjön helyesen a következő kódrészlet:

```
Szamologep gep;
std::cin >> gep >> gep;      // input: 4 2
gep.osszead();
gep.kiir();                  // output: 6
```

Tartalmazott objektum alkalmazása a célszerű:

```
class Szamologep {
  Verem v;
public:
  void osszead();
  void kivon();
  void szoroz();
  void oszt();
  void kiir();
  friend istream& operator>>(istream&, Szamologep&);
};

void Szamologep::osszead() {
  int op1 = v.pop();
  int op2 = v.pop();
  v.push(op2+op1);
}

void Szamologep::kiir() {
  v.dup();
  cout << v.pop();
}

istream& operator>>(istream& is, Szamologep& szg) {
  int i;
  is >> i;
  szg.v.push(i);
  return is;
}
```

Privát örökléssel is megoldható.

További tagfüggvények (eben a csoportban nem kellett megírni):

```
void Szamologep::kivon() {
  int op1 = v.pop();
  int op2 = v.pop();
  v.push(op2-op1);
}

void Szamologep::szoroz() {
  int op1 = v.pop();
  int op2 = v.pop();
  v.push(op2*op1);
}

void Szamologep::oszt() {
  int op1 = v.pop();
  int op2 = v.pop();
  if (op1 == 0) then throw range_error("Szamologep::oszt - nullával oszt");
  v.push(op2/op1);
}
```

A hibakezelés hiányát nem vettük itt hibának.

5. Feladat Σ 5 pont

Egy számítógéppel felügyelt kaszinó működését szeretnénk modellezni. A kaszinóban a játék**asztalok** (*Asztal*) nyilvántartják a pillanatnyi tétet, és az asztalnál ülő **játékos**okat, valamint azt, hogy az adott játék hanyadik körében járunk. Egy asztalnál maximum 10 játékos ülhet, de lehet üres asztal is. Bármelyik játékos bármelyik asztalhoz leülhet és ott a leülés sorrendjében játszhat. A játék során az asztalba épített számítógép **kör**önként, sorban felszólítja a játékosokat, hogy **lép**jenek. Egy játékos a játék fajtájától függően többféleképpen léphet: passzolhat, tétet emelhet, stb. Többféle játékos lehet, amelyek eltérő stratégiát alkalmaznak: például kezdő (*Kezdo*) játékos, aki minden páratlan körben passzol, a többiben emeli a tétet, vagy egy minden körben passzoló robot (*Robot*) játékos. Minden játékos nyilvántartja, hogy éppen melyik asztalnál ül. A rendszerben az alábbi műveleteket kell megvalósítani:

Asztal:

1. Új játékos csatlakozása az asztalhoz (*ujJatekos*). Ilyenkor a játékos is megjegyzi, hogy melyik asztalnál ül.
2. Játék kezdése az asztalnál (*ujJatek*). Ennek hatására a tét és a körszámláló nullázódik.
3. Kör végrehajtása (*kor*), melyben minden asztalnál ülő játékos egyet lép.
4. Kör sorszámának lekérdezése (*getKor*).
5. Tét emelése (*emel*)

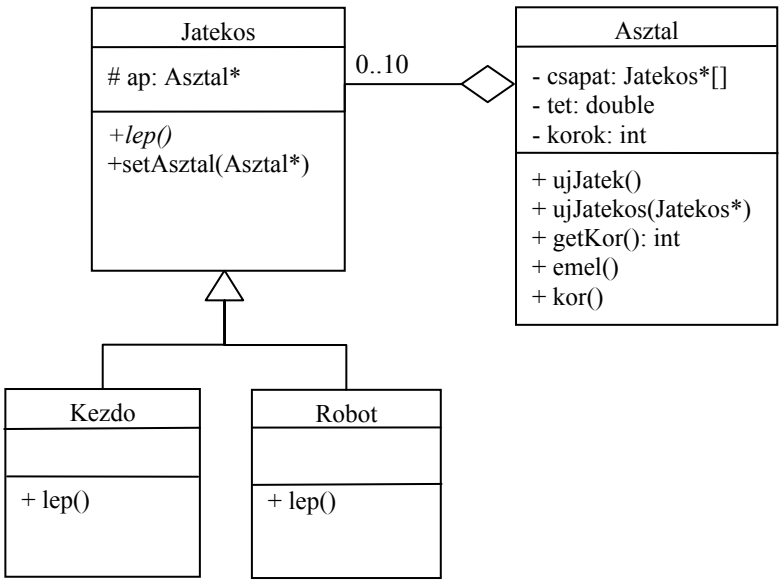
Játékos:

6. Lép (*lep*). A játékos ilyenkor megkérdezheti az asztalt, hogy hanyadik körnél tartanak, vagy éppen szólhat az asztalnak, hogy emeli a tétet.

Feladatok:

- **Tervezzen** meg OO modellt a problémának megfelelően, amelyben van *Asztal*, *Robot*, valamint *Kezdo* játékos, és könnyen bővíthető további játékosokkal. Rajzolja fel a modell osztálydiagramját! Használja a dőlt betűs neveket! Használhat STL tárolókat, algoritmusokat is!
- **Implementálja** az osztályokat és tagfüggvényeit!
- **Írjon** egy egyszerű programrészletet, ami egy asztalhoz leültet két különböző játékost, és „lejátszik” egy kört!

Egy lehetséges megoldás:



```
class Asztal;

class Jatekos {
protected:
    Asztal *ap;
public:
    void setAsztal(Asztal *a) { ap  = a; }
    virtual void lep() {};
    virtual ~Jatekos() {}
};

class Asztal {
    vector<Jatekos*> csapat;
    double tet;
    int korok;
public:
    Asztal() :tet(0) {}
    void ujJatek() { korok = 0; tet = 0; }
    void ujJatekos(Jatekos *uj);
    void kor();
    int getKor() { return korok;}
    void emel(double v) { tet += v;}
};

void Asztal::ujJatekos(Jatekos *uj) {
    csapat.push_back(uj);
    uj->setAsztal(this);
}

void Asztal::kor() {
    korok++;
    for (unsigned int i = 0; i < csapat.size(); i++)
        csapat[i]->lep();
}

class Kezdo :public Jatekos {
public:
    void lep() {
        if (ap->getKor() % 2 == 0)
            ap->emel(100);
    }
};

class Robot :public Jatekos { };

    Asztal a;
    Robot r;
    Kezdo k;
    a.ujJatekos(&r);
    a.ujJatekos(&k);
    r.lep();
```