228. AA Default Konstruktor AA 228.

Programozás alapjai 2. (inf.) zárthelyi	2012.05.17. gyak. hiányzás:	kZHpont:
ABCDEF	**Q-II/107.**	nZh: Hftest:

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

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, 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! *Elégséges: 8 ponttól*

1. Feladat

Σ 3 pont

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

```
#include <iostream>
using namespace std;
struct Alap {
  Alap(int i = 9)                    { cout << i << "k"; }
  Alap(const Alap&)                  { cout << "c" << "z";; }
  virtual void f(int i)              { cout << "f"; }
  void operator=(const Alap& a)      { cout << "=" << "z"; }
  virtual ~Alap()                    { cout << "d"; }
};
struct Uj :public Alap {
  Uj(int a = 228, char *m = "z")     { cout << m << "K"; }
  Uj(const Uj& u) :Alap(u)           { cout << "C" << "z"; }
  void f(int i)                      { cout << "F" << i; }
  ~Uj()                              { cout << "D"; }
};
void f(Uj& u) {
  u.f(9);
}

int main() {
  Uj u19(107, "cde");          cout << endl;
  Alap *pa = new Uj;           cout << endl;
  Uj u2 = u19;                 cout << endl;
  u19 = u2;                    cout << endl;
  pa->f(107);                  cout << endl;
  f(u19);                      cout << endl;
  delete pa;                   cout << endl;
}
```

9	k	c	d	e	K				
9	k	z	K						
c	z	C	z						
=	z								
F	1	0	7						
F	9								
D	d								
D	d	D	d						

2. Feladat

Σ 4 pont

Írjon függvénysablont (*kivalaszt*), ami egy tárolóból összehasonlítással képes kiválasztani egy adott tulajdonságnak (pl. maximum) megfelelő elemet! Ha több, az adott feltételnek megfelelő elemet talál, akkor az első megfelelőt válassza! A függvény első két paramétere két iterátor, ami kijelöli a jobbról nyílt intervallum kezdetét és végét. A harmadik paraméter pedig egy olyan kétparaméterű függvény vagy függvényobjektum, ami a kiválasztáshoz két elemet összehasonlít. A függvény visszatérési értéke a kiválasztott elemre mutató iterátor. Amennyiben helyesen oldja meg a feladatot, akkor az alábbi kódrészlet a következőt írja ki: **37**

```
int v9[] = { 2, -2, 3, 37, -21, 6, 0};
cout << *kivalaszt(v9, v9+6, greater<int>()) << endl;
```

Készítsen egy olyan **generikus függvényobjektumot**, ami alkalmazható lenne a fenti példában a legnagyobb abszolút értékű szám kiválasztására! **Adja meg,** hogy az elkészített függvényobjektum milyen tulajdonságokat (műveleteket) tételez fel a generikus adatról!

Egy lehetséges megoldás:

```
template<typename Iter, class Pred>
Iter kivalaszt(Iter first, Iter last, Pred pred) {
    Iter tmp = first++;
    while (first != last) {
        if (pred(*first, *tmp)) tmp = first;
        ++first;
    }
    return tmp;
}

template<class T>
struct AbsMax {
    bool operator()(T i, T j) {
        return abs(i) > abs(j);
    }
};
```

A függvényobjektum által feltételezett tulajdonságok:
Van másoló konstruktora és destruktora, (érték szerint veszi át a paramétert).
Működik rá az *abs()* függvény.

---

3. Feladat Σ 4 pont

Deklaráljon egy verem (*Stack*) szervezésű generikus tárolót, ami tetszőleges számú adatot képes tárolni. A tároló működését úgy alakítsa ki, hogy amennyiben a tároló dinamikus memóriaterületét növelni kell, úgy ne csak 1 adatnak megfelelő mérettel növeljen, hanem a sablon sablonparaméteréként megadott konstansnak megfelelő mérettel. Ezen sablonparaméter alapértelmezett értéke 100 legyen! A tároló rendelkezzen a következő műveletekkel:

- konstruktorok/destruktor
  - – paraméter nélküli konstruktor üres tárolót hoz létre;
  - – kétparaméteres konstruktor a paraméterként kapott 2 iterátornak megfelelő elemekkel tölti fel a tárolót;
- *top()* – a verem legfelső elemének referenciáját adja; dobjon *std::out_of_range* kivételt, ha a verem üres;
- *push()* – elemet tesz a verembe;
- *pop()* – eldobja a verem legfelső elemét; dobjon *std::out_of_range* kivételt, ha üres a verem;
- *swap()* – a verem legfelső 2 elemét felcseréli; dobjon *std::out_of_range* kivételt, ha a veremben 2-nél kevesebb adat van;
- *size()* – a tárolóban levő elemek számát adja;
- *empty()* – igaz értéket ad, ha a tároló üres.

Az osztályból példányosított objektum legyen átadható érték szerint függvényparaméterként, kezelje helyesen a többszörös értékadást (*s1=s2=s2*)! A *top* tagfüggvény konstans objektumra is működjön!

Valósítsa meg az osztály konstruktorait, valamint a *push(), swap() és empty()* metódusokat!

Egy lehetséges megoldás:

```
template<class T, size_t inc = 100>
class Stack {
    T *tar;
    size_t msiz;
    size_t siz;
public:
    Stack();
    Stack(const Stack&);
    template <typename Iter>                    // template paraméter az iterátor típusa
    Stack(Iter first, Iter last);
    Stack& operator=(const Stack&);
    T& top();
    const T& top() const;
    void push(const T&);
    void pop();
    void swap();
    int size() const { return siz; }
    bool empty() const { return siz == 0; }
    ~Stack() { delete[] tar; }
};
```

```
// lehetne az osztály dekl.val együtt, de a megodás menetének így jobban megfelel
template<class T, size_t inc>
Stack<T, inc>::Stack(): msiz(0), siz(0) { // konstruktor
    tar = new T[msiz];
}
template<class T, size_t inc>
Stack<T, inc>::Stack(const Stack& s2) {    // másoló konstruktor
    tar = NULL;
    *this = s2;
}
template<class T, size_t inc>
template <typename Iter>
Stack<T, inc>::Stack(Iter first, Iter last) :msiz(0), siz(0) { // konstruktor 2 iterátorral
    tar = new T[msiz];
    while(first != last)
        push(*first++);
}
template<class T, size_t inc>
void Stack<T, inc>::push(const T& e) {
    if (siz == msiz) {
        T *tmp = new T[msiz+inc];
        for (size_t i = 0; i < siz; i++)
            tmp[i] = tar[i];
        delete[] tar;
        tar = tmp;
        msiz += inc;
    }
    tar[siz++] = e;
}
template<class T, size_t inc>
void Stack<T, inc>::swap() {
    if (siz = < 2) throw std::out_of_range("Sor::swap: siz<2!");
    T tmp = tar[siz-1];
    tar[siz-1] = tar[siz-2];
    tar[siz-2] = tmp;
}

// További csoportok megoldásai (pop, top)
template<class T, size_t inc>
T& Stack<T, inc>::top() {
    if (siz == 0) throw std::out_of_range("Stack::top: empty!");
    return tar[siz-1];
}
template<class T, size_t inc>
const T& Stack<T, inc>::top() const {
    if (siz == 0) throw std::out_of_range("Stack::top: empty!");
    return tar[siz-1];
}
template<class T, size_t inc>
void Stack<T, inc>::pop() {
    if (siz == 0) throw std::out_of_range("Stack::pop: empty!");
    siz--;
}
```

228. AA Default Konstruktor AA 228.

4. Feladat Σ 4 pont

Tételezze fel, hogy a **3. feladat** *Stack* osztálya elkészült, és hibátlanul működik! Ezen osztály **felhasználásával deklaráljon** *Calculator* osztályt, ami aritmetikai műveleteket tud végezni double típusú adatokkal! A *Calculator* osztály rendelkezzen a következő műveletekkel:

- *store()* – adatot ír a *Calculator* veremmemóriájába;
- *print()* – kiveszi a veremből a legfelső elemet és kiírja paraméterként kapott *std:ostream*-re;
- *add()* – összeadja a verem két legfelső elemét és az eredményt a verem tetejére teszi;
- *sub()* – kivonja a verem legfelső elemét az alatta levő elemből és az eredményt a verem tetejére teszi;

Amennyiben a kétparaméteres műveletek esetében kevesebb, mint 2 elem van a veremben, dobjon *std::out_of_range* kivételt! Működjön az elvárásoknak megfelelően az alábbi kódrészlet:

```
Calculator cal9;
std::cin >> cal9;  // adatot olvas a Calculator veremmemóriájába
std::cout << cal9; // kiveszi a Calculator veremmemóriájának legfelső elemét és kiírja
```

Ügyeljen arra, hogy a *Stack* osztály metódusai ne legyenek publikusan elérhetők! **Valósítsa** meg a Calculator osztály print() műveletét, valamint a beolvasást a >> operatorral!

```
class Calculator {
    Stack<double> st;
public:
    void store(double x);
    void print(ostream& os);
    void add();
    void sub();
};
void Calculator::store(double x) {
    st.push(x);
}
void Calculator::print(ostream& os) {
    os << st.top();
    st.pop();
}
```

```
// További csoportok megoldásai (add, sub)
void Calculator::add() {
    double d = st.top();
    st.pop();
    st.top() += d;
}
void Calculator::sub() {
    double d = st.top();
    st.pop();
    st.top() -= d;
}
```

```
ostream& operator<<(ostream& os, Calculator& c) {
    c.print(os);
    return os;
}

// További csoportok megoldása (opeartor>>())
istream& operator>>(istream& is, Calculator& c) {
    double d;
    is >> d;
    c.store(d);
    return is;
}
```

5. Feladat Σ 5 pont

Egy többszereplős kalandjátékot szeretnénk modellezni. A játékosok avatarokat (*Avatar*) irányítanak. Egy avatarnak van ereje (*strength, double*), neve (*name, string*). Az avatarok különböző dolgokat (*Item*) tudnak felvenni és tárolni, például gyémántot (*Diamond*), láncfűrészt (*Chainsaw*). Később szeretnénk a modellt több fajta dologgal is bővíteni. Minden dolognak van neve (*name, string*), értéke (*value, double*), és különböző hatással lehetnek más avatarokra. A gyémánt a másik avatár erejét megduplázza, a láncfűrész a másik avatart megöli, stb. Az avatarokat a játékmester (*Master*) ismeri és lépteti, gondoskodik róla, hogy az avatarok találkozzanak, illetve dolgokat vegyenek fel. Egy avatarnál maximum 5 dolog lehet, ha több lenne nála, akkor a legrégebbi dolgot eldobja, az újat megtartja. A tervezőknek speciális avatarjuk (*Immortal*) van, amelyik nem tud meghalni, és tetszőleges számú dolgot birtokolhat. A rendszerben az alábbi műveleteket (tagfüggvényeket) kell megvalósítani:

Avatar	1.	setter/getter metódus az erő módosításához
	2.	meghalás: életerő lenullázása (*die*)
	3.	dolog felvétele (*pick*)
	4.	találkozás egy másik avatarral (*meet*); ekkor az avatár az összes tárgyával hatást gyakorol (*affect*) a másikra
Item	5.	hatás egy avatarra (*affect*)
Master	6.	új avatar felvétele (newAvatar)
	7.	halott avatarok elhamvasztása (deleteAvatar)

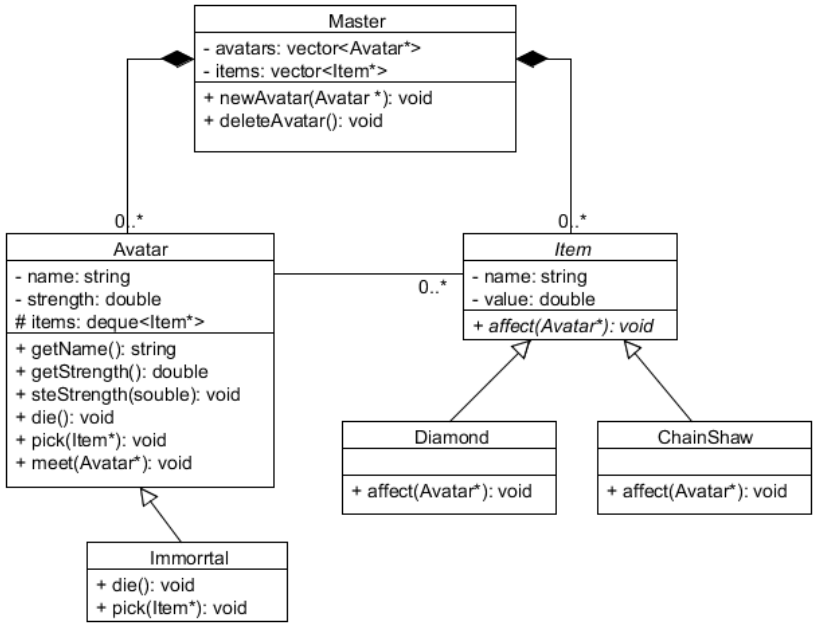
228. AA Default Konstruktor AA 228.

Feladatok:

- **Tervezzen** egy OO modellt a problémának megfelelően, amelyben van *Avatar, Immortal, Item, Diamond, Chainsaw, Master* és könnyen bővíthető újabb típusú avatarokkal és dolgokkal, amikben dinamikus memóriakezelés is lehet. **Rajzolja fel** a modell osztálydiagramját! **Használja** a dőlt betűs neveket! Jelölje a láthatóságokat is!
- **Deklarálja** az objektumokat C++ nyelven!
- **Implementálja** (valósítsa meg) az *Avatar* és *Item* objektumok die(), pick(), meet(), és affect () tagfügvényét!

A megoldáshoz **Használhat** STL tárolókat, algoritmusokat is!

Osztálydiagram:



A Master és az Item között nincs feltétlenül kompozíció (tartalmazás) típusú kapcsolat. Jó modell lehet az is, ha az Avatar és az Item között építünk ki tartalmazás típusú kapcsolatot.

Osztályok deklarációja (include nem kell):

```
class Master {
private:
    vector<Avatar*> avatars;
    vector<Item*> items;
public:
    void newAvatar(Avatar* avatar);
    void deleteAvatar();
};

class Avatar {
private:
    string name;
    double strength;
protected:
    deque<Item*> items;
public:
    double getStrength() const;
    string getName() const;
    void setStrength(double value);
    virtual void die();
    virtual void pick(Item* item);
    virtual void meet(Avatar* other);
    virtual ~Avatar();
};
```

```
class Immortal :public Avatar {
public:
    void die();
    void pick(Item* item);
};

class Item {
private:
    string name;
    double value;
public:
    virtual ~Item();
    virtual void affect(Avatar* avatar) = 0;
};

struct Diamond : Item {
    void affect(Avatar* avatar);
};

struct Chainsaw : Item {
    void affect(Avatar* avatar);
};
```

Implementációk:

```
void Avatar::die() {
    setStrength(0);
}
void Avatar::pick(Item* item) {
    if (items.size() >= 5)
        items.pop_front();
    items.push_back(item);
}
void Avatar::meet(Avatar* other) {
    for (deque<Item*>::iterator it = items.begin(); it != items.end(); ++it)
        (*it)->affect(other);
}
```

Az alábbiakat nem kellett megvalósítani, de a további megértéshez segíthetnek:

```
void Immortal::die() {} // nem fog meghalni

void Immortal::pick(Item* item) { items.push_back(item); }

void Master::newAvatar(Avatar *av) { avatars.push_back(av); }

void Master::deleteAvatar() {   // halottak hamvasztása
    for(vector<Avatar*>::iterator it = avatars.begin(); it != avatars.end(); ) {
        if ((*it)->getStrength() == 0) {
            delete *it;
            it = avatars.erase(it);
        } else it++;
    }
}

struct Chainsaw : Item {
    void affect(Avatar* av) { av->die(); }
};
```