.AA Megoldó Alfréd AA .

Programozás alapjai 2. (inf.) pót zárthelyi 2013.05.23. gyak. hiányzás:	kZH: ZH:0
ABCDEF **MEG/200.**	Hftest: (p)

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! *Ponthatárok: 8, 11, 14, 17*

F.	Max.	Elért
1	3	
2	4	
3	4	
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! Abba a sorba írjon, ahol a kiírás keletkezik! Figyeljen a változók élettartamára!

```
#include <iostream>
std::ostream& cout = std::cout;
inline void nl()                    { cout << std::endl; }

struct Adat {
  int i;
  Adat(int i = -1) :i(i)            { cout << i << "k"; }
  Adat(const Adat& a) :i(a.i)       { cout << i << "c"; }
  Adat operator+(int n)             { Adat tmp(i + n); cout << '+'; return tmp; }
  ~Adat()                           { cout << "d"; }
};

Adat operator+(int n, Adat& a)      { Adat tmp(a + n); return tmp; }

struct Uj {
  Adat a;
  Uj(int a = -3, const char *m = "cde"):a(a)      { cout << m << "K"; }
  Uj(const Uj& u)                                 { cout << "C" ; }
  ~Uj()                                           { cout << "D"; }
};

int main() {
  Adat *p = new Adat(200);          nl();
  *p = *p + 3;                      nl();
  *p = 2 + *p;                      nl();
  Uj u3(4, "ABCDEF");               nl();
  Uj uj16 = u3;                     nl();
  delete p;                         nl();
  return 0;
}
```

2	0	0	k											
2	0	3	k	+	d									
2	0	5	k	+	d									
4	k	A	B	C	D	E	F	K						
-	1	k	C											
d														
D	d	D	d											

A fenti megoldásban nem tüntettük fel a kioptimalizálható másolókonstruktor hívásokat, de ha valaki azt is feltüntette természetesen elfogadtuk.

2. Feladat **Σ 4 pont**

2.a) Készítsen sablont adatpár tárolására (*Par*)! A pár mindkét adata azonos típusú. Az adattagok legyenek közvetlenül elérhetők. Az osztálynak az automatikusan keletkező tagfüggvényein kívül ne legyen más tagfüggvénye. Oldja meg, hogy az osztályból példányosított objektumok kiírhatók legyenek egy std::ostream típusú objektumra a szokásos << operátorral. (1p)

```
template <class T>
struct Par {
    T tag1, tag2;
};

template <class T>
ostream& operator<<(ostream &os, Par<T> p) {
    return os << p.tag1 << p.tag2;
}
```

2.b) Készítsen egy olyan függvénysablont (*kombi*), ami előállítja egy sorozat másodosztályú kombinációit, azaz az elemeiből álló összes lehetséges párt *(Par)*! A függvény első két paramétere egy-egy iterátor, amely a sorozat elejét és végét (jobbról nyílt intervallum kezdete és vége) jelöli. A sorozat elemeiről feltételezheti, hogy mind különbözőek. A harmadik paraméter egy olyan std::vector, amelybe az eredményt kell betennie: n(n-1)/2 darab párt *(Par),* ahol n a bemeneti sorozat hosszát jelenti. A megoldáshoz használhatja az STL elemkészletét! Amennyiben helyesen oldotta meg mindkét részfeladatot, az alábbi kódrészlet a következőt írja a standard outputra: **AB,AC,AD,BC,BD,CD,**

```
using namespce std;
char betuk[] = "ABCD";
vector<Par<char> > parok;
kombi(betuk, betuk+4, parok);
copy(parok.begin(), parok.end(), ostream_iterator<Par<char> >(cout, ","));
```

```
template <class I, class T>
void kombi(I first, I last, vector<Par<T> >& vec) {
    while (first != last) {
        Par<T> p;
        p.tag1 = *first++;
        for (I i = first; i != last; i++) {
            p.tag2 = *i;
            vec.push_back(p);
        }
    }
}
```

3. Feladat Σ 4 pont

Egy generikus halmaz (*Set*) tárolót kell készíteni, ami tetszőleges számú adatot képes tárolni. A tároló rendelkezzen a következő műveletekkel:

- *isElement()* – annak eldöntése, hogy egy adott elem benne van-e a halmazban;
- *insert()* – elem hozzáadása a halmazhoz ;
- *size()* – visszaadja, hogy hány elem van a halmazban;
- *empty()* – igaz, ha a halmaz üres;
- *clear()* – törli az összes elemet

Az osztályból példányosított objektum:

- legyen átadható érték szerint függvényparaméterként;
- kezelje helyesen a többszörös értékadást (*s1=s2=s3*);

Deklarálja a *Set* osztályt! **Valósítsa** meg az osztály konstruktorát, másoló konstruktort, valamint a clear() tagfüggvényt

Hozza létre a {10.0, 10.3, 10.8} halmazt!

Specializálja ennek a halmaznak (double) az *isElement* tagfüggvényét úgy, hogy két elemet akkor tekintsen azonosnak, ha különbségük abszolútértéke kisebb, mint 1e-8!

```
template <class T>
class Set {
   T *t;
   size_t siz;
public:
   Set() :t(NULL), siz(0) { }
   Set(const Set& rhs);
   Set& operator=(const Set& rhs);
   bool isElement(const T& val) const;
   void insert(const T& val);
   size_t size() const;
   bool empty() const;
   void clear();
   ~Set();
};

template <class T>
Set<T>::Set(const Set& rhs) :t(NULL) {
    siz = rhs.siz;
    t = new T[siz];
    for (size_t i = 0; i < siz; i++)
        t[i] = rhs.t[i];
}

template <class T>
void Set<T>::clear() {
    siz = 0;
    delete [] t;
    t = NULL;
}
```

Létrehozás:

```
Set<double> s;    s.insert(10.0);    s.insert(10.3);    s.insert(10.8);
```

Specializáció:

```
template <>
bool Set<double>::isElement(const double& val) const {
    for (size_t i = 0; i != siz; i++)
        if (abs(t[i]-val) < 1e-10) return true;
    return false;
}
```

}

Más csoportok feladatai:

```
template <class T>
Set<T>& Set<T>::operator=(const Set& rhs) {
    if (this != &rhs) {
        delete [] t;
        siz = rhs.siz;
        t = new T[siz];
        for (size_t i = 0; i < siz; i++)
          t[i] = rhs.t[i];
    }
    return *this;
}
```

```
template <class T>
void Set<T>::insert(const T& val) {
    if (isElement(val)) return;
    T* tuj = new T[siz+1];
    for (size_t i = 0; i < siz; i++)
        tuj[i] = t[i];
    tuj[siz++] = val;
    delete [] t;
    t = tuj;
}

template <class T>
size_t Set<T>::size() const { return siz; }
```

4. Feladat Σ 4 pont

Klónozhatónak nevezünk egy objektumot, ha létezik a *clone()* tagfüggvénye, ami dinamikusan létrehoz egy eredetivel megegyező típusú és tartalmú objektumot, és visszaadja annak címét. Az *std::vector* osztály felhasználásával **készítsen** egy olyan generikus osztályt (*ClonableCollection*), amelyet klónozható heterogén kollekcióként használhatunk! Az így klónozott objektum egy különálló heterogén kollekció. A tárolt típusról tételezze fel, hogy az klónozható alaposztályra mutató pointer!
Hozzon létre egy klónozható osztályt, és segítségével mutassa be a *ClonableCollection* osztály klónozását!

```
using namespce std;
template<class T>
class ClonableCollection: public vector<T> {
public:
    ClonableCollection* clone() {
        ClonableCollection* myClone = new ClonableCollection;
        for (size_t i=0; i< vector<T>::size();i++)
            myClone->push_back((*this)[i]->clone());
        return myClone;
    }
    ~ClonableCollection() {
        for (size_t i=0; i< vector<T>::size();i++)
            delete (*this)[i];
    }
};

struct A {
    virtual A* clone() {return new A;}
    Virtual ~A() {}
};

ClonableCollection<A*> cc, *ccp;
ccp=cc.clone();
```

5. Feladat Σ 5 pont

Hackerképző iskolát modellezünk, ahol a diákok (Student) különféle képességeket (Skill) tanulhatnak (Student::learn). A képesség például jelszótörés (PwdCracking), ami adott időt vesz igénybe (int), féregtelepítés (WormInstall), stb. Ha egy diák meghal, a tanult képességei elvesznek. Bármely képességet lehet alkalmazni (Skill::apply) egy számítógépen (Computer). A számítógépnek van védelme (int). Ha egy számítógépen egy képességet alkalmaznak, csökken a védelme. Pl. a jelszótörés eggyel csökkenti a védelmet, a féregtelepítés mindig egy véletlen számmal. Ha a védelem 1 alá csökken, a gép feltörtnek minősül, ami lekérdezhető (Computer::isBroken). A diák megkísérelheti egy iterátorpárral megadott számítógép-sorozat feltörését (Student::crack).

Feladatok:

- **STL** eszközök felhasználásával tervezzen olyan OO modellt, ami könnyen bővíthető további képességekkel, hiszen az iskolában folyamatosan újabb képességeket tanítanak!
- Rajzolja fel a modell osztálydiagramját! Ne tüntesse fel a tagfüggvényeket az UML ábrán! Használja a megadott dőlt betűs neveket!
- Deklarálja és implementálja a *Computer, Student, Skill és PwdCracking* osztályokat!
- Írjon egy olyan kódrészletet, amiben dinamikusan létrehoz 80 számítógépből álló tömböt, két diákot valamilyen képességgel, és megkíséreli mindkét diák a számítógépek feltörését, majd kiírja, hogy hány számítógép feltörése sikerült.

```
class Computer {
   int vedelem;
public:
   Computer(int v0):vedelem(v0) {}
   void addVedelem(int delta) { vedelem += delta;}
   bool isBroken() const { return vedelem < 1; }
};

class Skill {
public:
   virtual void apply(Computer& c)=0;
   virtual ~Skill() {}
};

class PwdCracking : public Skill {
public:
   void apply(Computer& c) { c.addVedelem(-1); }
};

class Student {
   vector<Skill*> skills;
public:
   void addSkill(Skill*s) {
       skills.push_back(s);
   }
   template<class I>
   void crack(I start, I end) {
       while (start != end) {
           for (size_t i = 0; i < skills.size(); i++)
               skills[i]->apply(*start);
           start++;
       }
   }
   ~Student() {
       for (size_t i = 0; i < skills.size(); i++)
           delete skills[i];
   }
};

vector<Computer> sz(80,Computer(2));
Student jozsi, gezu;
jozsi.addSkill(new PwdCracking);
gezu.addSkill(new PwdCracking);
gezu.crack(sz.begin(), sz.end());
jozsi.crack(sz.begin(), sz.begin()+5);
int cnt = 0;
for (size_t i = 0; i < 80; i++)
    if (sz[i].isBroken()) cnt++;
cout << cnt << " db gepet tortunk fel" << endl;
```