503.DD #HIÁNYZIK Megoldó Alfréd DD 503.

Programozás alapjai 2. (inf.) zárthelyi 2013.05.13. gyak. hiányzás: 0	kZHpont: #HIÁNYZIK
ABCDEF **MEG/1.**	Hftest: 0 (p)

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! *Értékelés: 0-8:1, 9-11:2, 12-14:3, 15-17:4, 18-20:5*

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 = -2, const char *m = "cde"):a(a)      { cout << m << "K"; }
  Uj(const Uj& u)                                 { cout << "C" ; }
  ~Uj()                                           { cout << "D"; }
};

int main() {
  Adat *p = new Adat(1);            nl();
  *p = *p + 3;                      nl();
  *p = 2 + *p;                      nl();
  Uj u2(4, "ABCDEF");               nl();
  Uj uj16 = u2;                     nl();
  delete p;                         nl();
  return 0;
}
```

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

2. Feladat **Σ 4 pont**

2.a) **Írjon** függvénysablont (***zip***), ami két sorozatból egy új sorozatot képez! A sablon az új sorozat elemeit az eredeti sorozat megegyező pozíciójú elemeiből az 5. paramétereként megadott kétparaméteres függvénnyel/függvényobjektummal képezze! A *zip* függvény első három paramétere három bemeneti iterátor, amelyek bemeneti adatokat jelölik ki: Az első kettő az első sorozat elejét és végét (jobbról nyílt intervallum kezdete és vége), a harmadik pedig a másik sorozat elejét. A negyedik paraméter egy kimeneti iterátor, ami az eredmény elejére mutat.

Amennyiben helyesen oldja meg a feladatot, akkor az alábbi kódrészlet a következőt írja ki: **6, 5, 4, 4, 5, 6,**

```
int in1f[] = { 1, 2, 3, 4, 5, 6};    // egyik bemeneti sorozat
int in2f[] = { 6, 5, 4, 3, 2, 1};    // másik sorozat
int outf[6];                         // eredmény helye

zip(in1f, in1f+6, in2f, outf, max<int>);
copy(outf, outf+6, ostream_iterator<int>(cout, ", "));
```

2.b) **Készítsen** egy olyan függvényobjektumot, ami alkalmazható a fenti példában a páronkénti összegek előállítására! (1p)

2.c) **Mutassa be,** hogy az elkészített függvénysablonnal és függvényobjektummal hogyan tudná összeadni két egészeket tartalmazó *std::vector* elemeit! (1p)

```
template <class Ii1, class Ii2, class Oi1, class F>
void zip(Ii1 b1, Ii1 e1, Ii2 b2, Oi1 oi, F f) {
    while (b1 != e1) {
        *oi++ = f(*b1, *b2);
        b1++;
        b2++;
    }
}

struct Osszead {
    int operator()(int i, int j) {
        return (i + j);
    }
};

vector<int> v1, v2, v3,;
zip(v1.begin(), v1.end(), v2.begin(), v3.begi(), Osszead());
```

3. Feladat Σ 4 pont

Készítsen olyan korlátos méretű generikus sort (*Queue*), ami képes visszaadni a legnagyobb és a legkisebb elemet is a tárolóból! Az elemek összehasonlítását a **<** operátorral végezze! A tároló maximális méretét az osztály egyparaméteres konstruktora vegye át, melynek alapértelmezett értéke 160 legyen! Az osztály rendelkezzen a következő műveletekkel:

- *max_e()* – a sor *legnagyobb* elemének referenciáját adja; dobjon *std::out_of_range* kivételt, ha üres a sor;
- *min_e()* – a sor *legkisebb* elemének referenciáját adja; dobjon *std::out_of_range* kivételt, ha üres a sor;
- *top()* – a sor *legelső* elemének referenciáját adja; dobjon *std::out_of_range* kivételt, ha üres a sor;
- *push()* – elemet tesz a sor végére; amennyiben a tárolt elemek száma meghaladná a tároló méretét, úgy dobjon *std::out_of_range* kivételt;
- *pop()* – eldobja a sor *legelső* elemét; dobjon *std::out_of_range* kivételt, ha üres a sor;
- *size()* – a tárolóban levő elemek számát adja;
- *maxsize()* – a tároló maximális méretét adja.

Az osztályból példányosított objektum:

- legyen átadható érték szerint függvényparaméterként;
- kezelje helyesen a többszörös értékadást (*q1=q2=q3*);
- a *max_e, min_e,* és *top* tagfüggvényeknek nem kell konstans objektumra működni!

Deklarálja a *Queue* osztályt! **Valósítsa** meg az osztály egyparaméteres konstruktorát, a másoló konstruktort, valamint a *min_e()*, és a *push()* metódusokat!

Egy rövid kódrészlettel **mutassa be** az osztály használatát: deklaráljon egy sort valós adatokkal és tegyen bele egy elemet!

```
template<class T>
class Queue {
    T *tar;
    size_t maxsiz;
    size_t siz;
public:
    Queue(size_t maxsiz = 160): maxsiz(maxsiz), siz(0) {
        tar = new T[maxsiz];
    }
    Queue(const Queue&);
    Queue& operator=(const Queue&);
    T& max_e();
    T& min_e();
    T& top();
    void push(const T&);
    void pop();
    size_t size() const;
    size_t maxsize() const;
    ~Queue();
};
```

503.DD #HIÁNYZIK Megoldó Alfréd DD 503.

```
template<class T>
Queue<T>::Queue(const Queue& q) {
    maxsiz = q.maxsiz;
    siz = q.siz;
    tar = new T[maxsiz];
    for (size_t i = 0; i < siz; ++i) tar[i] = q.tar[i];
}
T& min_e () throw (out_of_range) {
    if (size() == 0) throw out_of_range("Queue::min_e: empty!");
    return *min_element(tar, tar+siz);
}
template<class T>
void Sor<T>::push(const T& e) {
    if (siz >= maxsize) throw std::out_of_range("Queue::push: full!");
    tar[siz++] = e;
}

Sor<double> s(100); s.push(3.56);
```

```
// másik csoportok feladatai:
template<class T>
Queue<T>& Queue<T>::operator=(const Queue& q) {
    if (&p != this) {
        delete[] tar;
        maxsiz = q.maxsiz;
        siz = q.siz;
        tar = new T[maxsiz];
        for (size_t i = 0; i < siz; ++i) tar[i] = q.tar[i];
    }
    return *this;
}
T& max_e () throw (out_of_range) {
    if (size() == 0) throw out_of_range("Queue::max_e: empty!");
    return *max_element(tar, tar+siz);
}
template<class T>
T& Queue <T>::top {
    if (siz == 0) throw std::out_of_range("Queue::top: empty!");
    return tar[0];
}
```

4. Feladat Σ 4 pont

Az *std::list* tároló felhasználásával **hozzon** létre egy olyan perzisztens tárolót (*PListString*), amely *std::string* típusú adatokat (a stringben nincsenek szóközök) tárol és úgy viselkedik, mint egy *std::list* tároló, azaz az *std::list* minden tagfüggvénye (konstruktorok is) és típusa elérhető! **Deklarálja** és **valósítsa** meg a *PListString* osztályt! **Mutassa** be az elkészült tároló használatát: egy kódrészletben hozzon létre kettő darab eltérő méretű *PListString* típusú objektumot, írja ki (perzisztálja) azokat egy fájlba, majd töltse vissza úgy, hogy a két objektum tartalma felcserélődjön!

```
using namespace std;
struct PListString: public list<string> {
	PListString(size_t n, const string& val = string()) : list<string>(n, val) {}
	template <class Iterator>
	PListString(Iterator first, Iterator last) : list<string>(first, last) {}
	void write(ostream& os) {
		os << size() << endl;
		for (iterator it = begin(); it != end(); ++it)
			os << *it << endl;
	}
	void read(istream& is) {
		size_t siz;
		is >> siz;
		erase(begin(),end());
		while (siz--) {
			string str;
			is >> str;
			push_back(str);
		}
	}
};

PListString p1(1, "haho"), p2(2, "ketto");
ofstream ofs("zh.dat");
p1.write(ofs); p2.write(ofs);
ifstream ifs("zh.dat");
p2.read(ifs); p1.read(ifs);
```

5. Feladat Σ 5 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:

- **STL** eszközök **felhasználásával tervezzen** olyan OO modellt, ami könnyen bővíthető további ruhafajtákkal. Rajzolja fel a modell osztálydiagramját! Ne tüntesse fel a tagfüggvényeket az UML ábrán! Használja a dőlt betűs neveket!
- **Deklarálja** és **implementálja** a *Mosoda*, *Ruha* és *Ing* osztályokat!
- **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!

```
class Ruha {
    enum {piszkos, tiszta} allapot;
    const string tulaj;
public:
    Ruha(const string & a) : allapot(piszkos), tulaj(a) {}
    const string& tulajdonos() const { return tulaj; }
    virtual void tisztit() { allapot = tiszta; }
    virtual void kiir() const {
        cout << "#" << tulaj << ", "
                << (allapot == piszkos ? "piszkos" : "tiszta");
    }
    virtual ~Ruha() {}
};

class Garbo : public Ruha {
    double szinero;
public:
    Garbo(const string & a, double sz = 1.0) : Ruha(a), szinero(sz) {}
    void tisztit() { Ruha::tisztit(); szinero *= .97; }
    void kiir() const { Ruha::kiir(); cout << " szinero: " << szinero; }
};

class Mosoda {
    list<Ruha*> t;
public:
    void bead(Ruha *r) {
        r->tisztit();
        t.push_back(r);
    }
    Ruha * kiad(const string & tulaj) {
        for (list<Ruha*>::iterator i = t.begin(); i != t.end(); i++)
            if ((*i)->tulajdonos() == tulaj) {
                Ruha *r = *i;
                t.erase(i);
                return r;
            }
        throw runtime_error("elvesztettuk, sajnaljuk");
    }
    ~Mosoda() {
        for (list<Ruha*>::iterator i = t.begin(); i != t.end(); i++)
            delete *i;
    }
};

Mosoda m1;
m1.bead(new Ruha("t1"));
m1.bead(new Garbo("g1"));
Ruha *r = m1.kiad("g1");
r->kiir();
delete r;
```