AAAAAAAAAAAAAAAAAAA rendelkezésre álló idő: 15 **perc**

Programozás alapja 2. 4. ellenőrző dolgozat. 2015.04.22. Kurz/**Terem**: G2/	Elért pontszám:
Név: **Neptun:**	

1. Írjon C++ függvénysablont (*szamlal_ha*), ami összeszámolja egy iterátorokkal megadott adatsorozat azon elemeit, amelyekre a paraméterként megadott predikátum igaz értéket ad! 4p
   A függvény első két paramétere két iterátor, ami kijelöli az adatsorozat kezdetét és végét (jobbról nyílt intervallum). A függvény 3. paramétere a predikátum, ami egy egyparaméteres függvény vagy függvényobjektum. A függvény visszatérési értéke egy egész szám. Ha jól oldja meg a feladatot, akkor az alábbi kódrészlet a szabványos kimentre 4-et ír ki.

```
bool negativ(int a) { return a < 0; }
...
int input[] = { 1, -4, 9, -16, -25, 8, 7, 33, 76, -2 };        // a sorozat
std::cout << szamlal_ha(input, input+10, negativ);
```

2. Hozzon létre az *std::vector* sablon felhasználásával egy olyan vektort, amiben a fenti példa *input* tömbjének adatai vannak! A 1. feladatban elkészített sablon és az *std::vector* sablon iterátoros interfészének felhasználásával írja ki a szabványos kimenetre, hogy hány páros elem van a vektorban! 3p
3. Készítsen olyan függvényobjektum sablont, ami a *szamlal_ha* sablon predikátumaként felhasználható egy generikus adatsorozat elemszámának meghatározására! Specializálja a sablont úgy, hogy ha a generikus adat *double*, akkor a *szamlal_ha* csak a nem negatív elemeket számolja össze. 2p
4. Rövid kódrészlettel mutassa be a specializált sablon használatát! 1p

Egy lehetséges megoldás:

```
template <class Iter, class P>
int szamlal_ha(Iter first, Iter last, P pred) {
    int sz = 0;
    while (first != last)
        if (pred(*first++)) sz++;
    return sz;
}

std::vector<int> vec(input, input+10);
bool paros(int a) { return (a&1) == 0; }
std::cout << szamlal_ha(vec.begin(), vec.end(), paros);

template <class T>
struct Mindig {
    bool operator()(T a) { return true; }
};
template<>
bool Mindig<double>::operator()(double a) { return a >= 0; }

std::vector<double> vd;
szamlal_ha(vd.begin(), vd.end(), Mindig<double>());
```

BBBBBBBBBBBBBBBBBBBBBB rendelkezésre álló idő:15 **perc**

Programozás alapja 2. 4. ellenőrző dolgozat. 2015.04.22. Kurz/**Terem**: G2/	Elért pontszám:
Név: **Neptun:**	

1. Írjon C++ függvénysablont (*osszeszamol_ha*), ami összeszámolja egy iterátorokkal megadott adatsorozat azon elemeit, amelyekre a paraméterként megadott predikátum igaz értéket ad! 4p
   A függvény első két paramétere két iterátor, ami kijelöli az adatsorozat kezdetét és végét (jobbról nyílt intervallum). A függvény 3. paramétere a predikátum, ami egy egyparaméteres függvény vagy függvényobjektum. A függvény visszatérési értéke egy egész szám. Ha jól oldja meg a feladatot, akkor az alábbi kódrészlet a szabványos kimentre 8-at ír ki.

```
bool szamjegy(int ch) { return ch >= '0' && ch <= '9'; }

char *azon = "nzh_20150422";                              // a sorozat
std::cout << osszeszamol_ha(azon, azon+12, szamjegy);
```

2. Hozzon létre az *std::vector* sablon felhasználásával egy olyan vektort, amiben a fenti példa azon tömbjének adatai vannak! A 1. feladatban elkészített sablon és az *std::vector* sablon iterátoros interfészének felhasználásával írja ki a szabványos kimenetre, hogy hány db kettes számjegy van a vektorban! 3p
3. Készítsen olyan függvényobjektum sablont, ami az *osszeszamol_ha* sablon predikátumaként felhasználható egy generikus adatsorozat elemszámának meghatározására! Specializálja a sablont úgy, hogy ha a generikus adat *int*, akkor az *osszeszamol_ha* csak a páratlan elemeket számolja össze. 2p
4. Rövid kódrészlettel mutassa be a specializált sablon használatát! 1p

Egy lehetséges megoldás:

```
template <class Iter, class P>
int osszeszamol_ha(Iter first, Iter last, P pred) {
    int sz = 0;
    while (first != last)
        if (pred(*first++)) sz++;
    return sz;
}

std::vector<char> vec(azon, azon+12);
bool kettes(char ch) { return ch == '2'; }
std::cout << osszeszamol_ha(vec.begin(), vec.end(), kettes);

template <class T>
struct Mindig {
    bool operator()(T a) { return true; }
};
template<>
bool Mindig<int>::operator()(int a) { return a&1; }

std::vector<int> vi;
osszeszamol_ha(vi.begin(), vi.end(), Mindig<int>());
```