L01: csütörtök 10-12 rendelkezésre álló idő:15 **perc**

Programozás alapjai 2. 3. beszámoló dolgozat. 2019.05.02. Kurz/**Terem**: L01/	Elért pontszám:
Név: **Neptun:**	

Labor e.d. (beugró): első feladatból min 2 pont.

1. Készítsen 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 kimenetre 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);
```

Egy lehetséges megoldás:

```
template <typename Iter, typename Func>
int szamlal_ha(Iter first, Iter last, Func func) {
    int cnt = 0;
    while (first != last) {
        if (func(*first++)) cnt++;
    }
    return cnt;
}
```

2. A *Serializable* osztály felhasználásával, annak módosítása nélkül hozzon létre egy olyan **perzisztens viselkedésű osztályt** (*PInteger*), ami egy egész érték perzisztens tárolására alkalmas és mindenütt használható ahol az **int** típus használható! Az adatfolyam feldolgozásakor nem kell hibát kezelnie, de ügyeljen arra, hogy helyesen olvassa vissza az egymás után kiírt adatokat is! (6p)

```
struct Serializable {
    virtual void write(std::ostream&) const = 0;
    virtual void read(std::istream&) = 0;
    virtual ~Serializable() {}
};
```

Egy lehetséges megoldás:

```
struct PInteger : public Serializable {
    int data;
    PInteger(int i = 0) : data(i) {}
    operator int&() { return data; }
    operator int() const { return data; }
    void write(std::ostream& os) const { os << data << " "; }
    void read(std::istream& is) { is >> data; }
};
```

L02: csütörtök 15-17 rendelkezésre álló idő:15 **perc**

Programozás alapjai 2. 3. beszámoló dolgozat. 2019.05.02. Kurz/**Terem**: L02/	Elért pontszám:
Név: **Neptun:**	

Labor e.d. (beugró): első feladatból min 2 pont.

1. Készítsen C++ **függvénysablont** (*lecserel_ha*), ami egy adott értékre cseréli 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 4. paraméter pedig egy érték, amire a megfelelő elemeket cserélni kell. Ha jól oldja meg a feladatot, akkor az alábbi kódrészletben a *lecserel_ha* függvény az *adatok* tömb minden negatív elemét 0-ra cseréli.

```
bool negativ(int a) { return a < 0; }
...
int adatok[] = { 1, -4, 9, -16, -25, 8, 7, -33, -6, -2 };        // a sorozat
lecserel_ha(adatok, adatok+10, negativ, 0);
```

Egy lehetséges megoldás:

```
template <typename Iter, typename Func, typename T>
void lecserel_ha(Iter first, Iter last, Func func, const T& val) {
    while (first != last) {
        if (func(*first)) *first = val;
        first++;
    }
}
```

2. A Serializable osztály felhasználásával, annak módosítása nélkül hozzon létre egy olyan **perzisztens viselkedésű osztályt** (PDouble), ami egy valós érték perzisztens tárolására alkalmas és mindenütt használható ahol a **`double`** típus használható! Az adatfolyam feldolgozásakor nem kell hibát kezelnie, de ügyeljen arra, hogy helyesen olvassa vissza az egymás után kiírt adatokat is! (6p)

```
struct Serializable {
    virtual void write(std::ostream&) const = 0;
    virtual void read(std::istream&) = 0;
    virtual ~Serializable() {}
};
```

Egy lehetséges megoldás:

```
struct PDouble : public Serializable {
    double data;
    PDouble(double i = 0) : data(i) {}
    operator double&() { return data; }
    operator double() const { return data; }
    void write(std::ostream& os) const { os << data << std::endl; }
    void read(std::istream& is) { is >> data; }
};
```

L03: péntek 10-12 rendelkezésre álló idő:15 **perc**

Programozás alapjai 2. 3. ellenőrző dolgozat. 2019.05.03. Kurz/**Terem**: L03/	Elért pontszám:
Név: **Neptun:**	

Labor e.d. (beugró): első feladatból min 2 pont.

1. Készítsen **generikus algoritmust** (`forEach()`), ami egy iterátorokkal megadott adatsorozat minden elemére meghívja a paraméterként kapott funktort (függvényobjektumot)! A függvény első két paramétere két iterátor, ami kijelöli az adatsorozat kezdetét és végét (jobbról nyílt intervallum). A függvény 3. paramétere egy egyparaméteres függvényobjektum. A függvény visszatérési értéke maga a meghívott funktor legyen! 4p

Egy lehetséges megoldás:

```
template <typename Iter, typename Func>
Func forEach(Iter first, Iter last, Func func) {
    while (first != last) {
        func(*first++);
    }
    return func;
}
```

2. Készítsen **generikus osztályt** (`osszegzo`), ami egy generikus adatot tárol! Az adat kezdőértékét konstruktorában lehessen megadni! Legyen az osztálynak egyoperandusú függvényhívás operátora, ami a letárolt adathoz **hozzáadja** a függvényhívás operátor operandusát. Az így kapott összeg legyen lekérdezhető (*get*)! Az osztályt többnyire funktorként fogjuk használni. Működjön helyesen a következő kódrészlet! 4p

```
osszegzo<int> sum(10);          // Kezdőérték: 10
sum(1);                         // hozzáad 1-et
sum(10);                        // hozzáad 10-et
std::cout << sum.get();         // Kiír: 21...
```

Egy lehetséges megoldás:

```
template <typename T>
struct osszegzo {
    T adat;
    osszegzo(T val = T()) :adat(val) {}
    T get() const { return adat; }
    void operator()(T val) { adat+= val; }
};
```

3. Hozzon létre **valós** számokból egy tömböt, töltse fel értékekkel, és mutassa be, hogy hogyan lehet az 1. és 2. feladatban elkészített sablonok felhasználásával összeadni a tömbben levő értékeket! Az összeget a standard kimentre írja ki! 2p

Egy lehetséges megoldás:

```
double szamok[] = { 1.1, 10.01, 100.001 };
std::cout << forEach(szamok, szamok+3, osszegzo<double>(0)).get() << std::endl;
```

Labor e.d. (beugró): első feladatból min 2 pont.

1. Készítsen **generikus algoritmust** (`forAll()`), ami egy iterátorokkal megadott adatsorozat minden elemére meghívja a paraméterként kapott funktort (függvényobjektumot)! A függvény első két paramétere két iterátor, ami kijelöli az adatsorozat kezdetét és végét (jobbról nyílt intervallum). A függvény 3. paramétere egy egyparaméteres függvényobjektum. A függvény visszatérési értéke maga a meghívott funktor legyen! 4p

Egy lehetséges megoldás:

```
template <typename Iter, typename Func>
Func forAll(Iter first, Iter last, Func func) {
    while (first != last) {
        func(*first++);
    }
    return func;
}
```

2. Készítsen **generikus osztályt** (`szorzo`), ami egy generikus adatot tárol! Az adat kezdőértékét konstruktorában lehessen megadni! Legyen az osztálynak egyoperandusú függvényhívás operátora, ami a letárolt adatot **megszorozza** a függvényhívás operátor operandusával. Az így kapott szorzat legyen lekérdezhető (*get*). Az osztályt tipikusan funktorként fogjuk használni. Működjön helyesen a következő kódrészlet! 4p

```
szorzo<int> mul(10);            // Kezdőérték: 10
mul(10);                        // megszorozza 10-zel
mul(10);                        // megszorozza 10-zel
std::cout << mul.get();         // Kiír: 1000...
```

Egy lehetséges megoldás:

```
template <typename T>
struct szorzo {
    T adat;
    szorzo(T val = T()) :adat(val) {}
    T get() const { return adat; }
    void operator()(T val) { adat*= val; }
};
```

3. Hozzon létre **valós** számokból egy tömböt, töltse fel értékekkel, és mutassa be, hogy hogyan lehet az 1. és 2. feladatban elkészített sablonok felhasználásával képezni a tömbben levő értékek szorzatát! A szorzatot a standard kimentre írja ki! 2p

Egy lehetséges megoldás:

```
double szamok2[] = { 2.22, 0.5, 100. };
std::cout << forEach(szamok2, szamok2+3, szorzo<double>(1)).get() << std::endl;
```