

13. LABOR

GYAKORLÓ ÓRA

Laborvezetőnek: a labor témája

Ezen a laboron mindenkivel a **sablonok, operátorok (konverziós operátor is)** témakörét gyakoroljuk, illetve az STL-t alap szinten felhasználjuk. STL-ben több segítséget is kaphatnak, mert csak a jobb osztályzatokért kérjük számon azok kimerítő ismeretét.

Hallgatónak: általános információk

Ezen a laboron mindenkivel a **sablonok, operátorok (konverziós operátor is)** témakörét gyakoroljuk, illetve az STL-t alap szinten felhasználjuk.

1. Feladat: „Adatpárok és Adattároló” rendszer

Leírás

Készíts egy olyan rendszert, amelyben különböző típusú adatpárokat (például számokat, szövegeket, stb.) tudsz tárolni és kezelni egy általános adattárolóban. Az `AdatPar<T1, T2>` osztály egy kétértékű adatpárt reprezentál, míg az `AdatTarolo<P>` osztály ezekből az adatpárokból tartalmaz egy tömböt, és képes új adatpárokat hozzáadni, valamint az összes tárolt adatpárt szimbolikusan "feldolgozni". A rendszerhez tartozik egy `feldolgoz` függvény, amely egy `AdatTarolo<P>&` paramétert vár, és végrehajt egy szimbolikus műveletet a benne tárolt adatpárokon (például kiírja, hogy "feldolgozás alatt" a két értéket).

A következő sablon- és tagneveket használd:

- `AdatPar<T1, T2>`: kétféle típusú értéket tároló osztály, tagváltozói: `elso`, `masodik` (privát), getterek: `getElso()`, `getMasodik()`
- `AdatTarolo<P>`: tetszőleges típusú adatpárokat tartalmazó osztály, tagváltozója: `adatok` (`std::vector<P>`, privát), getter: `getAdatok()`
- `feldolgoz`: külső függvény, amely egy `AdatTarolo<P>&` paramétert kap

1.1 Részfeladat

Hozd létre az `AdatPar<T1, T2>` osztálysablon, amely két különböző típusú értéket tárol privát tagként, és getter függvényekkel biztosítja azok elérését.

Laborvezetőnek: megoldás

Az `AdatPar<T1, T2>` sablonosztály két privát adattagot (`elso`, `masodik`) tartalmaz. Ezekhez csak getter függvényeken (`getElso()`, `getMasodik()`) keresztül lehet hozzáférni.

```
template<typename T1, typename T2>
class AdatPar {
private:
    T1 elso;
    T2 masodik;
public:
    AdatPar(const T1& e, const T2& m) : elso(e), masodik(m) {}

    T1 getElso() const {
        return elso;
    }
    T2 getMasodik() const {
        return masodik;
    }
};
```

1.2 Részfeladat

Készítsd el az AdatTarolo<P> sablonosztályt, amely egy privát std::vector<P> típusú tagváltozóban adatpárokat tárol, és getterrel teszi elérhetővé azokat.

Laborvezetőnek: megoldás

Az AdatTarolo<P> sablonosztály privát adattagként tartalmaz egy std::vector<P> típusú adatok nevű vektort. Ehhez csak egy publikus getter (getAdatok()) biztosít hozzáférést, amely konstans referenciát ad vissza. Ez a kapszulázás megakadályozza, hogy a tárolt adatokat közvetlenül módosítsák kívülről, és csak a tároló által biztosított műveleteken keresztül lehessen módosítani vagy lekérdezni az adatokat.

```
#include <vector>

template<typename P>
class AdatTarolo {
private:
    std::vector<P> adatok;
public:
    void hozzaad(const P& p) { //következő részfeladat lesz
        adatok.push_back(p);
    }

    const std::vector<P>& getAdatok() const {
        return adatok;
    }
};
```

1.3 Részfeladat

Egészítsd ki az AdatTarolo<P> osztályt egy hozzaad tagfüggvénnyel, amely egy új adatpárt ad hozzá a tárolóhoz.

Laborvezetőnek: megoldás

A hozzaad függvény lehetővé teszi, hogy a felhasználó új adatpárokat adjon a tárolóhoz. Ha ezt nem készítenénk el, csak a konstruktorban lehetne adatokat megadni, ami nagyon kötötté tenné az osztály használatát. A függvény paramétere konstans referenciaként van megadva, így nem másoljuk feleslegesen az

adatpárt, csak ha ténylegesen beillesztjük a vektorba. A getterrel biztosítjuk, hogy az adatok csak olvashatók legyenek kívülről, módosítani csak a tároló metódusain keresztül lehet.

(A fenti kódrészlet már tartalmazza a hozzáad függvényt.)

1.4 Részfeladat

Készíts egy `feldolgoz` nevű függvényt, amely egy `AdatTarolo<P>&` paramétert vár, és szimbolikusan "feldolgozza" az összes benne tárolt adatpárt (például írja ki, hogy "feldolgozás alatt: <első>, <masodik>").

Laborvezetőnek: megoldás

A `feldolgoz` függvény sablonként valósul meg, hogy bármilyen típusú adatpárokat tartalmazó tárolót tudjon kezelni. Referenciaként vesszük át a paramétert, hogy ne másoljuk feleslegesen a tárolót. A függvény minden elemre meghívja a gettereket, és kiírja a két értéket egy szimbolikus szöveggel. Ha nem gettereket használnánk, megsértenénk az adatok kapszulázását. Az adatokhoz a tároló getterén keresztül férünk hozzá, így a kapszulázás ezen a szinten is érvényesül.

```
#include <iostream>

template<typename P>
void feldolgoz(const AdatTarolo<P>& tarolo) {
    const std::vector<P>& lista = tarolo.getAdatok();
    for (std::size_t i = 0; i < lista.size(); ++i) {
        std::cout << "feldolgozás alatt: "
                  << lista[i].getElso()
                  << ", "
                  << lista[i].getMasodik()
                  << std::endl;
    }
}
```

1.5 Részfeladat

Demonstráld a rendszer működését a `main` függvényben: hozz létre egy `AdatTarolo<AdatPar<int, std::string>>` példányt, adj hozzá néhány adatpárt, majd hívd meg a `feldolgoz` függvényt.

Laborvezetőnek: megoldás

A `main` függvényben példányosítjuk az adattárolót, feltöltjük néhány adattal, majd kiíratjuk azokat a szimbolikus feldolgozó függvénnyel. Így jól látszik, hogy a sablonosztályok segítségével bármilyen típusú párokat és tárolókat könnyedén kezelhetünk, miközben az adatok kapszulázása is biztosított.

```
#include <string>

int main() {
    AdatTarolo<AdatPar<int, std::string>> tarolo;
    tarolo.hozzaad(AdatPar<int, std::string>(1, "alma"));
    tarolo.hozzaad(AdatPar<int, std::string>(2, "korte"));
    tarolo.hozzaad(AdatPar<int, std::string>(3, "szilva"));

    feldolgoz(tarolo);
    return 0;
}
```

2. Feladat: Könyvtári könyvek kölcsönzése

Készítsen olyan rendszert, amely egy könyvtár könyveinek példányszámát és kölcsönzési statisztikáit nyilvántartja. A Book osztály egy könyvet reprezentál, amelynek van címe (title, amely létrehozás után nem változik), összes példányszáma (copies), és a jelenleg kölcsönzött példányok száma (borrowed). Egy könyv példányszáma nem lehet kevesebb 1-nél. A kölcsönzések számát az adott könyv objektumon egyesével lehet növelni, a prefix inkremens operátor akár halmozott használatával:

```
//A "C++ Mesterfokon" című könyv példányszámát 5-re állítottuk a létrehozáskor  
++(++cppMaster); //2 példányt kölcsönöztek ki
```

a.)

Tervezze és írja meg a könyvet reprezentáló osztályt! Mutassa be a könyv létrehozását! Egészítse ki az osztályt úgy, hogy a fenti működésmodnak megfeleljen!

b.)

A könyv objektum tud double-ként viselkedni, ekkor a kölcsönzött példányok arányát adja vissza (borrowed/copies):

```
cout << "Kölcsönzési arány: " << (double)cppMaster << endl; //0.4 az érték, ha 2 példányt  
kölcsönöztek ki
```

c.)

Két könyv példányainak kölcsönzéseit össze lehet vonni, amikor például egy könyv új kiadása miatt a régi példányokat átvezetik az új könyv statisztikájába. Ezt a következő módon kell megvalósítani:

```
//A "C++ Alapok" című könyv példányszáma 3, nincs kölcsönzött példány  
++cppBasics; //1 példányt kölcsönöztek ki  
cout << "Kölcsönzési arány: " << (double)cppBasics << endl; //0.333...  
//Átvezetjük a kölcsönzéseket a mesterfokra:  
cppMaster += cppBasics;  
cout << "Kölcsönzési arány (mesterfokon): " << (double)cppMaster << endl; //0.6, 3 példány  
kölcsönözve 5-ből  
cout << "Kölcsönzési arány (alapok): " << (double)cppBasics << endl; //0, nincs már kölcsönzött  
példány
```

Írja meg azokat a kódrészleteket, amelyek a fenti működéshez szükségesek! (Egyéb kódokkal, pl. könyv visszaadása stb. nem kell foglalkozni.)

2.1 Részfeladat

Készítse el a Book osztályt, amely privát adattagként tárolja a könyv címét, példányszámát és a kölcsönzött példányok számát! Gondoskodjon róla, hogy a példányszám 1-nél kisebb ne lehessen, és a cím, példányszám létrehozás után ne változzon!

Laborvezetőnek: megoldás

A Book osztályban a cím (title), példányszám (copies) és a kölcsönzött példányok száma (borrowed) privát adattagok. A példányszámot a konstruktorban ellenőrizzük, és csak akkor engedjük létrehozni az objektumot, ha legalább 1 példány van. A cím és példányszám csak olvasható, a kölcsönzött példányszámot a későbbiekben lehet módosítani. Ha nem védenénk le a példányszámot, hibás, értelmetlen könyvobjektumok is létrejöhetnének.

```
#include <iostream>
```

```
#include <stdexcept>
#include <string>

class Book {
private:
    std::string title;
    int copies;
    int borrowed;
public:
    Book(const std::string& t, int c) : title(t), copies(c), borrowed(0) {
        if (copies < 1) {
            throw std::invalid_argument("A példányszám nem lehet kevesebb, mint 1!");
        }
    }
    std::string getTitle() const {
        return title;
    }
    int getCopies() const {
        return copies;
    }
    int getBorrowed() const {
        return borrowed;
    }
};
```

2.2 Részfeladat

Valósítsa meg a prefix inkremens operátort (++), amely egy példány kölcsönzését szimulálja! Gondoskodjon róla, hogy ne lehessen több példányt kölcsönözni, mint amennyi elérhető!

Laborvezetőnek: megoldás

A prefix inkremens operátor (operator++()) növeli a kölcsönzött példányok számát, de csak akkor, ha még van elérhető példány. Ha nem védenénk le ezt, akkor a kölcsönzött példányok száma túlléphetné a példányszámot, ami hibás állapothoz vezetne. Az operátor referenciát ad vissza, hogy többszörösen egymás után is használható legyen (pl. ++(++obj)).

```
Book& operator++() {
    if (borrowed < copies) {
        ++borrowed;
    }
    return *this;
}
```

A teljes osztály most így néz ki:

```
class Book {
private:
    std::string title;
    int copies;
    int borrowed;
public:
    Book(const std::string& t, int c) : title(t), copies(c), borrowed(0) {
        if (copies < 1) {
            throw std::invalid_argument("A példányszám nem lehet kevesebb, mint 1!");
        }
    }
    std::string getTitle() const { return title; }
    int getCopies() const { return copies; }
    int getBorrowed() const { return borrowed; }

    Book& operator++() {
        if (borrowed < copies) {
            ++borrowed;
        }
    }
};
```

```
    }  
    return *this;  
}  
};
```

2.3 Részfeladat

Valósítsa meg, hogy a könyv objektum `double` típusra konvertálható legyen, ekkor a kölcsönzött példányok arányát adja vissza (`borrowed/copies`)!

Laborvezetőnek: megoldás

A `double` típusra konverziót egy `operator double()` tagfüggvénnyel valósítjuk meg. Ez visszaadja a kölcsönzött példányok arányát. Ha nem lenne ilyen operátor, a könyv objektumot nem lehetne egyszerűen arányként használni, mindenhol külön getterrel kellene számolni.

```
operator double() const {  
    return static_cast<double>(borrowed) / static_cast<double>(copies);  
}
```

A teljes osztály most így néz ki:

```
class Book {  
private:  
    std::string title;  
    int copies;  
    int borrowed;  
public:  
    Book(const std::string& t, int c) : title(t), copies(c), borrowed(0) {  
        if (copies < 1) {  
            throw std::invalid_argument("A példányszám nem lehet kevesebb, mint 1!");  
        }  
    }  
    std::string getTitle() const { return title; }  
    int getCopies() const { return copies; }  
    int getBorrowed() const { return borrowed; }  
  
    Book& operator++() {  
        if (borrowed < copies) {  
            ++borrowed;  
        }  
        return *this;  
    }  
  
    operator double() const {  
        return static_cast<double>(borrowed) / static_cast<double>(copies);  
    }  
};
```

2.4 Részfeladat

Valósítsa meg a += operátort, amely egy másik könyv kölcsönzött példányait átvonja az aktuális könyvre! Az átvezetett könyv kölcsönzött példányainak száma nullázódjon, de ne lehessen több példányt kölcsönözni, mint amennyi az aktuális könyvnél elérhető!

Laborvezetőnek: megoldás

A += operátor átveszi a másik könyv kölcsönzött példányait, de csak annyit, amennyi még elfér az aktuális könyv példányszámában. A másik könyv kölcsönzött példányainak száma a művelet után 0 lesz. Ha nem így járnánk el, az egyik könyv kölcsönzött példányai duplikálódhatnának, vagy hibás állapot keletkezhetne.

```
Book& operator+=(Book& other) {
    int szabad = copies - borrowed;
    int atviheto = (other.borrowed < szabad) ? other.borrowed : szabad;
    borrowed += atviheto;
    other.borrowed -= atviheto;
    // Biztonság kedvéért, ha other.borrowed negatív lenne (bár nem lehetne)
    if (other.borrowed < 0) other.borrowed = 0;
    return *this;
}
```

A teljes osztály most így néz ki:

```
class Book {
private:
    std::string title;
    int copies;
    int borrowed;
public:
    Book(const std::string& t, int c) : title(t), copies(c), borrowed(0) {
        if (copies < 1) {
            throw std::invalid_argument("A példányszám nem lehet kevesebb, mint 1!");
        }
    }
    std::string getTitle() const { return title; }
    int getCopies() const { return copies; }
    int getBorrowed() const { return borrowed; }

    Book& operator++() {
        if (borrowed < copies) {
            ++borrowed;
        }
        return *this;
    }

    operator double() const {
        return static_cast<double>(borrowed) / static_cast<double>(copies);
    }

    Book& operator+=(Book& other) {
        int szabad = copies - borrowed;
        int atviheto = (other.borrowed < szabad) ? other.borrowed : szabad;
        borrowed += atviheto;
        other.borrowed -= atviheto;
        if (other.borrowed < 0) other.borrowed = 0;
        return *this;
    }
};
```

2.5 Részfeladat

Mutassa be a működést egy main függvényben a következő lépések szerint:

- Hozzon létre két könyvet, példányszámokkal!

- Végezzen el kölcsönzéseket a prefix inkremens operátorral!
- Írja ki a kölcsönzési arányokat!
- Vezesse át az egyik könyv kölcsönzéseit a másikra a += operátorral!
- Ismét írja ki a kölcsönzési arányokat!

Laborvezetőnek: megoldás

A main függvényben példányosítjuk a könyveket, végrehajtjuk a kölcsönzéseket, majd átvezetjük az egyik könyv kölcsönzéseit a másikra. Az operátorok és a konverzió segítségével a kód olvasható és természetes marad. Ha nem használnánk operátorokat, minden lépéshez külön metódusokat kellene hívni, ami kevésbé lenne átlátható.

```
int main() {
    Book cppMaster("C++ Mesterfokon", 5);
    ++(++cppMaster); // 2 példányt kölcsönöztek ki
    std::cout << "Kölcsönzési arány (mesterfokon): " << (double)cppMaster << std::endl; // 0.4

    Book cppBasics("C++ Alapok", 3);
    ++cppBasics; // 1 példányt kölcsönöztek ki
    std::cout << "Kölcsönzési arány (alapok): " << (double)cppBasics << std::endl; // 0.333...

    cppMaster += cppBasics;
    std::cout << "Kölcsönzési arány (mesterfokon): " << (double)cppMaster << std::endl; // 0.6
    std::cout << "Kölcsönzési arány (alapok): " << (double)cppBasics << std::endl; // 0

    return 0;
}
```