C++

4. HÉT – RULE OF FIVE

A JEGYZETET KÉSZÍTETTE:

CSAPÓ ÁDÁM BALÁZS ÉS ŐSZ OLIVÉR

A C++ mögött máig aktív szabványosító testület áll – minden 3-4 évben új szabvány jelenik meg.

Szerencsére a 'gyártók' (akik a fordítókat készítik) a szabványt komolyan is veszik.

AZ ELSŐ NÉHÁNY HÉTBEN A C++11 ÉS AZÓTA MEGJELENT ÚJÍTÁSOKRÓL ESIK SZÓ

Ismétlés: Rule of Three

Ha egy osztály saját maga kezeli a dinamikus memóriáját, akkor a helyes működéshez meg kell valósítani ezt a 3 dolgot:

- Destruktor: memória felszabadítása
- Másoló konstruktor: dinamikus memória másolása
- Értékadás operátor: dinamikus memória másolása

A szabály nem csak dinamikus memóriára, más erőforrásokra is alkalmazható.

Rule of Zero: A standard library számos kész eszközt, adatstruktúrát nyújt dinamikus adattárolásra, ha nincs valami speciális igényünk, akkor használjuk ezeket, és ne írjunk saját memóriakezelést.

Move konstruktor

Másoló konstruktorhoz hasonló, csak jobbérték referenciát (&&) vár paraméterként.

Akkor hívódik meg, ha a lemásolandó objektumról tudni lehet, hogy a másolás után megszűnik.

Ezt kihasználva másolás helyett mozgatást végezhetünk. Az erőforrásokat nem lemásoljuk, hanem átvesszük a paraméter objektumtól.

A mozgatott objektumnak ettől még meg fog hívódni a destruktora, ezért ennek megfelelő állapotban kell hagyni.

```
BigInt(BigInt&& other) noexcept
        : length(other.length)
        , digits(other.digits)
{
    other.digits = nullptr;
    other.length = 0;
}
```

Move assignment operátor

Hasonlóan, az értékadás operátornak is lehet move változatot definiálni.

```
BigInt& operator=(BigInt&& other) noexcept
{
    length = other.length;
    digits = other.digits;
    other.digits = nullptr;
    other.length = 0;
    return *this;
}
```

Rule of Five: A destruktor, copy konstruktor, copy assignment, move konstruktor és move assignment közül, ha az egyiket definiáljuk, akkor a többit is definiálni kell.

Noexcept specifier

A noexcept kulcsszóval jelezhetjük a fordítónak, hogy egy függvény nem fog kivételt dobni.

Ez bizonyos függvényeknél lehetővé tesz további fordítói kódoptimalizálást.

A standard library tárolói csak akkor használják ki, hogy a típusunk move konstruktorral (és move assignment operátorral) rendelkezik, ha az noexcept-nek van jelölve.

A noexcept után zárójelben megadható egy konstans logikai kifejezés, ami ki- vagy bekapcsolja a hatását. Ez hasznos lehet pl. ha template-et készítünk és a template argumentum bizonyos tulajdonságaitól függ, hogy dobhatunk-e kivételt. Ebben segít a noexcept operátor, aminek megadva egy kifejezést, az visszaadja, hogy az nem dobhat-e kivételt.

```
void will_throw() noexcept(false) { throw 666; }
void may_throw() { throw 42; } // default is
noexcept(false)
void wont_throw() noexcept { } // noexcept is
the same as noexcept(true)

template <class T>
void it_depends() noexcept(noexcept(T())) { }

int main() {
    cout << noexcept(will_throw()) << endl; // 0
    cout << noexcept(may_throw()) << endl;  // 0
    cout << noexcept(wont_throw()) << endl; // 1
}
```

Konverziós konstruktor

Ha egy konstruktornak egyetlen (az adott osztálytól különböző) paramétere van, azt konverziós konstruktornak nevezzük, és felhasználható implicit típuskonverzióra.

Ez hasznos lehet, mert így egy X objektum helyett egy Y objektumot is használhatunk egy kifejezésben, ha van X(Y) konstruktor, anélkül, hogy azt kézzel meg kellene hívnunk.

Viszont a nem várt konverziók hibához is vezethetnek, ezért biztonságosabb ha letiltjuk az implicit konverziót. Ehhez írjuk be az explicit kulcsszót a konverziós konstruktor elé.

Konverziós operátor

A konverziós konstruktorral definiálhatunk más típusokból a saját típusunkra való konverziókat.

Viszont ha a saját típusunkat akarjuk más típusra konvertálni, és a másik típushoz nem tudunk konverziós konstruktort definiálni, akkor más megoldás kell. Erre szolgál a konverziós operátor.

A metódus neve az operator szó után a másik típus neve, amivé konvertálunk. Visszatérési érték nincs, az megegyezik a névben szereplő típussal, és paraméter sincs.

Itt is vigyázni kell a váratlan konverziókkal. A példában szereplő, ártalmatlannak tűnő bool-konverzió okozhat pár meglepetést.

```
class BigInt {
    int length;
    char* digits;
public:
    operator bool() const {
        return length > 0;
    }
    // ...
};
```