C++11: Move szemantika

Pataki Norbert

ELTE Informatikai Kar,
Programozási Nyelvek és
Fordítóprogramok Tanszék

2014. november 21.

Másolás

- Költség
- Pointerek, referenciák
- Pimpl, opaque pointer
- Copy-on-write (COW)
  - `std::vector` vs. `std::string`
- Expression templates: felesleges temporálisok
- Return value optimization: compiler szinten

C++11: Move semantics

- Nyelvi eszköz a felesleges másolások csökkentésére
- Számos esetben elkerülhető a másolás
- Nem mindig van szükség másolandó adat megőrzésére
- Mozgatás: „Könnyebb”, olcsóbb művelet lehet, mint a másolás
- `std::auto_ptr`

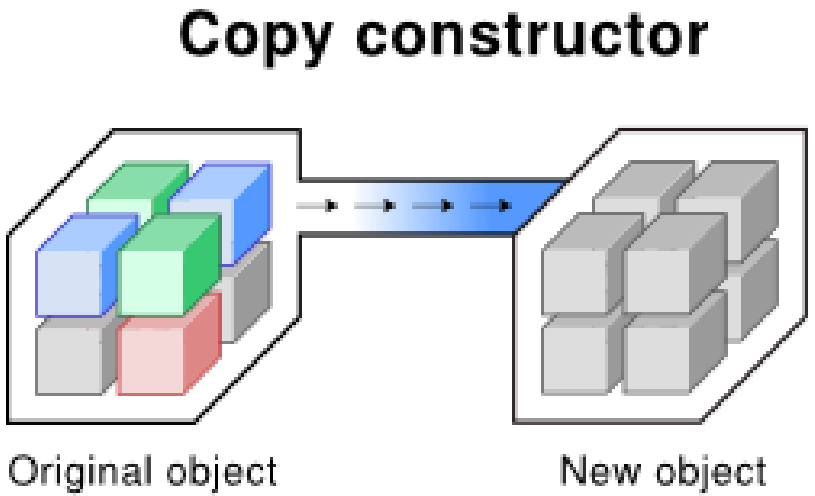
Jobbérték referencia

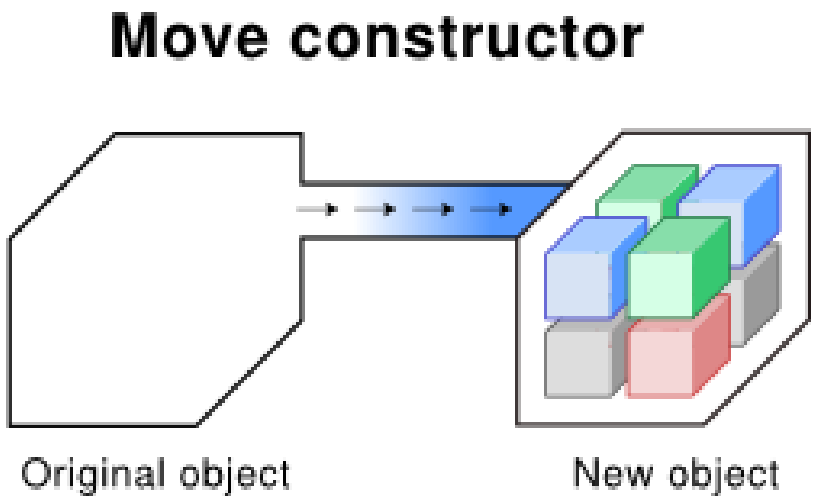
Temporálisokhoz köt:

```
int x = 4;
int&& r = x + 7;

std::string&& get_msg()
{
  std::string ret;
  // ...
  return std::move( ret );
}
```

Move constructor





Mátrix osztály

```
class Matrix
{

  int rows, columns;
  T* m;
public:

  Matrix(int r, int c ): rows( r ), columns( c )
  {
    m = new T[ rows * columns ];
  }
  // ...
};
```

Move műveletek

```
class Matrix
{
  // ...
public:

  Matrix( Matrix&& rhs )
  {
    rows = rhs.rows;
    columns = rhs.columns;
    m = rhs.m;
    rhs.m = nullptr;
  }

  Matrix& operator=( Matrix&& rhs );
};
```

Move műveletek

```
Matrix& Matrix::operator=( Matrix&& rhs )
{
  if ( this != &rhs )
  {
    delete [] m;
    rows = rhs.rows;
    columns = rhs.columns;
    m = rhs.m;
    rhs.m = nullptr;
  }
  return *this;
}
```

Használat

```
#include <utility>

// ...

Matrix m( 3, 6 );
// ...
Matrix s = std::move( m );
// ...
m = std::move( s );
```

Implicit move

- Ha nincs user-defined copy ctor
- Ha nincs user-defined értékadó operátor
- Ha nincs user-defined move
- Ha nincs user-defined destruktor
- Tagonkénti `std::move`