Copy vs. Move constructors

Robbe Claessens
Ceder Dens
Lander Geeraert
Mitchel Pyl
Ben Van Muylder

Copy Constructor

```
class A {
  A(const A & o);
};
```

- Kopie maken van een object

- Klassen kunnen meerdere copy constructoren hebben

- Compiler maakt impliciet copy constructor aan als:
  - Er geen move constructor gedeclareerd is
  - Er geen move assignment operator gedeclareerd is

Copy Constructor

```
class A {
  A(const A & o);
};
```

- Een copy constructor kan triviaal zijn
- Na een copy, zijn er 2 verschillende objecten die exact hetzelfde bevatten

- Expliciet te genereren via `A(const A & o) = default;`
- Expliciet te verwijderen via `A(const A & o) = delete;`

Move Constructor

```
class A {
  A(A && o);
  A(const A && o);
};
```

- Overhandigen van resources tussen objecten
- Enkel bruikbaar met rvalues
- Expliciet aan te roepen met `std::move()`
- Sneller dan copy constructor, omdat deze niet alles moet kopiëren maar gewoon kan overnemen

Move Constructor

```
class A {
  A(A && o);
  A(const A && o);
};
```

- Na move van een object is het oorspronkelijke object in een onbepaalde staat en wordt deze beter niet hergebruikt
- Impliciet aanwezig tenzij een van de volgende expliciet aanwezig is:
  - destructor
  - copy constructor
  - copy assignment operator
  - move assignment operator

Move Constructor

```
class A {
  A(A && o);
  A(const A && o);
};
```

- Expliciet te genereren via `A(A && o) = default;`
- Expliciet te verwijderen via `A(A && o) = delete;`
- `std::move_if_noexcept()` staat toe om te kiezen tussen move en copy indien er geen exceptions gethrowd mogen worden, neemt move indien deze als `noexcept` gemarkeerd is, anders copy

Voorbeelden

```
class X {
public:
  X() { cout << "'Constructor'" << endl; }
  X(const X & o) { cout << "'Copy'" << endl; }
  X(X && o) { cout << "'Move'" << endl; }
};

X a;

X b(a);
X c = a;

X d(std::move(b));
X e = std::move(c);
```

Output:

```
'Constructor'
'Copy'
'Copy'
'Move'
'Move'
```

Voorbeelden

```
class X { // Klasse X met move maar zonder copy constructor
public:
  X() { cout << "Constructor" << endl; }
  X(X && o) { cout << "Move" << endl; }
};

X f(X & x) { return x; } // Fout, copy is niet gedefiniëerd (in return)
X g(X & x) { return std::move(x); } // OK, move is gedefiniëerd
X h(X x) { return std::move(x); } // OK

X a;
X y = std::move(a);
X b(g(y)); // OK

X c(h(a)); // Fout, copy constructor van X niet gedefiniëerd
X d(h(std::move(a))); // Wel OK, geeft rvalue aan g
```

Voorbeelden

```
class Y {
public:
  // Copy en move constructors impliciet gedefiniëerd
  Y(string s) : m_s(s) {}
  void print() { cout << "'" << m_s << "'" << endl; }
private:
  string m_s;
};

Y a("Foo"), b("Bar");

Y x = a;
Y y = std::move(b);

a.print(); b.print(); x.print(); y.print();
```

Output:

```
'Foo'
''
'Foo'
'Bar'
```

Triviale Copy/Move Constructor

- Maakt een bytewise kopie van de representatie in het geheugen
- Kan manueel via `std::memmove()`
- Alle data types van C zijn triviaal kopieerbaar

- Een copy/move constructor is triviaal als:
  - Impliciet gedefinieerd of default gedefinieerd
  - De klasse geen virtuele methodes heeft
  - De klasse geen virtuele basis klasse heeft
  - De copy/move constructor voor elke basis klasse triviaal is
  - De copy/move constructor voor elke niet-statische member triviaal is

Copy elision

- Optimaliseert copy of move constructors bij pass-by-value objecten

- Elimineert onnodig kopiëren van objecten
  - Zelfs wanneer copy/move waarneembare neveneffecten heeft

- Situaties waarin elision voorkomt
  - Return statements
  - Throw exceptions
  - Catch clausules

Copy elision

- Behandeld de source & target als 2 verschillende manieren om te verwijzen naar hetzelfde object

- Destruction gebeurt wanneer source & target oorspronkelijk verwijderd zouden zijn

- Meerdere elisions mogen na elkaar komen om meerdere kopieën te verwijderen

Voorbeelden

```
int n = 0;
struct C {
  C(int) {}
  // De copy-constructor heeft zichtbaar effect
  C(const C &) { ++n; }
};
int main() {
  C c1(42);          // Direct-initialisatie, calls C::C(42)
  C c2(C(42));       // Copy-constructor, calls C::C( C::C(42) )
  cout << n << endl; // Print 0 als copy elision heeft opgetreden
  C c3(c2);          // Geen naamloze temporary
  cout << n << endl; // Print 1 want bij c3(c2) kan geen
                     // copy-elision optreden
}
```

Output:

```
0
1
```

Voorbeelden

```
struct C {
  C() {}
  // De copy-constructor heeft zichtbaar effect
  C(const C &) { cout << ”Hello World!” << endl; }
};
void f() { C c; throw c; }
int main() {
    try {
        f();
    } catch(C c) {}
}
```

verwachte output:

```
...
```

output:

```
Hello World!
Hello World!
```