Kopierkonstruktor

?

```
struct Cell {
  int val;

  Cell(int v): val(v) {};

  Cell(const Cell& other):
      val(other.val - 1) {};
};

std::ostream& operator<<(
    std::ostream& out, Cell c0)
{
  return out << c0.val;
}

int main() {
  Cell c1(5);
  Cell c2(c1);

  std::cout << c1 << " " << c2;
}
```

Was gibt das Programm aus?

1. 5 4
2. 4 5
3. 4 3
4. 3 4
5. 3 2
6. 2 3

Kopierkonstruktor

!

```
struct Cell {
  int val;

  Cell(int v): val(v) {};

  Cell(const Cell& other):
      val(other.val - 1) {};
};

std::ostream& operator<<(
    std::ostream& out, Cell c0)
{
  return out << c0.val;
}

int main() {
  Cell c1(5);
  Cell c2(c1);

  std::cout << c1 << " " << c2;
}
```

Was gibt das Programm aus?

1 5 4

2 4 5

3 4 3 ●

4 3 4

5 3 2

6 2 3

Kopierkonstruktor

!

```
struct Cell {
  int val;

  Cell(int v): val(v) {};

  Cell(const Cell& other):
      val(other.val - 1) {};
};

std::ostream& operator<<(
    std::ostream& out, Cell c0)
{
  return out << c0.val;
}

int main() {
  Cell c1(5);
  Cell c2(c1);

  std::cout << c1.val << " " << c2.val; // 5 4 (as expected)
}
```

Kopierkonstruktor !

```
struct Cell {
  int val;

  Cell(int v): val(v) {};

  Cell(const Cell& other):
      val(other.val - 1) {};
};

std::ostream& operator<<(
    std::ostream& out, Cell c0)
{
  return out << c0.val;
}

int main() {
  Cell c1(5);
  Cell c2(c1);

  std::cout << c1 << " " << c2;
}
```

Kopierkonstruktor

```
struct Cell {
  int val;

  Cell(int v): val(v) {};

  Cell(const Cell& other):
      val(other.val - 1) {};
};

std::ostream& operator<<(
    std::ostream& out, Cell c0)
{
  return out << c0.val;
}

int main() {
  Cell c1(5);
  Cell c2(c1);

  std::cout << c1 << " " << c2;
}
```

Call By Value!

Kopierkonstruktor

```
struct Cell {
  int val;

  Cell(int v): val(v) {};

  Cell(const Cell& other):
      val(other.val - 1) {};
};

std::ostream& operator<<(
    std::ostream& out, Cell c0)
{
  return out << c0.val;
}

int main() {
  Cell c1(5);
  Cell c2(c1);

  std::cout << c1 << " " << c2;
}
```

Call By Value!

→ `c1` (`c1.val == 5`) muss kopiert werden

Kopierkonstruktor

!

```
struct Cell {
  int val;

  Cell(int v): val(v) {};

  Cell(const Cell& other):
      val(other.val - 1) {};
};

std::ostream& operator<<(
    std::ostream& out, Cell c0)
{
  return out << c0.val;
}

int main() {
  Cell c1(5);
  Cell c2(c1);

  std::cout << c1 << " " << c2;
}
```

Call By Value!

→ `c1` (`c1.val == 5`) muss kopiert werden

→ Temporäre `Cell` namens `c0`

Kopierkonstruktor

!

```
struct Cell {
  int val;

  Cell(int v): val(v) {};

  Cell(const Cell& other):
      val(other.val - 1) {};
};

std::ostream& operator<<(
    std::ostream& out, Cell c0)
{
  return out << c0.val;
}

int main() {
  Cell c1(5);
  Cell c2(c1);

  std::cout << c1 << " " << c2;
}
```

Call By Value!

→ `c1` (`c1.val == 5`) muss kopiert werden

→ Temporäre `Cell` namens `c0`

→ Kopierkonstruktor wird aufgerufen: `Cell c0(c1)`

→ `c0.val == c1.val - 1 == 4`

Kopierkonstruktor

!

```
struct Cell {
  int val;

  Cell(int v): val(v) {};

  Cell(const Cell& other):
      val(other.val - 1) {};
};

std::ostream& operator<<(
    std::ostream& out, Cell c0)
{
  return out << c0.val;
}

int main() {
  Cell c1(5);
  Cell c2(c1);

  std::cout << c1 << " " << c2; // 4 3
}
```

Call By Value!

→ `c1` (`c1.val == 5`) muss kopiert werden

→ Temporäre `Cell` namens `c0`

→ Kopierkonstruktor wird aufgerufen: `Cell c0(c1)`

→ `c0.val == c1.val - 1 == 4`

Analog für die Ausgabe von `c2`