

Effizienzbetrachtungen

Verschiebeoperatoren und Verschiebekonstrukturen

Motivation von speziellen Verschiebeoperatoren

```
typedef Guest V_BaseType;
```

```
class Vektor{  
public:  
Vektor() = delete;  
Vektor(int dim);  
    Vektor(const Vektor& v2);  
    ~Vektor();  
    Vektor& operator=(const Vektor& v2);  
  
void Print() const;  
    void Init(V_BaseType w);  
    friend Vektor operator+(const Vektor& v1, const Vektor& v2);  
    static int GetTotalCounter() { return totalCounter; }  
    static int GetTotalCounterAssign() { return totalCounterAssign; }  
  
private:  
    static int totalCounter;  
    static int totalCounterAssign;  
  
    V_BaseType* daten;  
    int dimension;  
};
```

Motivation von speziellen Verschiebeoperatoren

```
void testPlusHelp() {  
    Vektor v1(400);  
    const V_BaseType wert1(11);  
    v1.Init(wert1);  
  
    Vektor v2(400);  
    const V_BaseType wert2(22);  
    v2.Init(wert2);  
  
    Vektor result = v1 + v2;  
    result = v1 + v1 + v1;  
}
```

Ausgangssituation:

Microsoft Visual Studio-Debugging-Konsole

```
Insgesamt hatten wir 8 Vektoren  
Insgesamt hatten wir 1 Vektor-Zuweisungen  
Insgesamt hatten wir 6004 Guests  
Insgesamt hatten wir 3600 Guest-Zuweisungen  
Ging alles gut
```

Wunsch:

Microsoft Visual Studio-Debugging-Konsole

```
Insgesamt hatten wir 5 Vektoren  
Insgesamt hatten wir 0 Vektor-Zuweisungen  
Insgesamt hatten wir 4404 Guests  
Insgesamt hatten wir 2000 Guest-Zuweisungen  
Ging alles gut
```

Effizient oder nicht effizient?

```
void testPlusHelp() {  
    Vektor v1(400);  
    const int wert1(11);  
    v1.Init(wert1);  
  
    Vektor v2(400);  
    const int wert2(22);  
    v2.Init(wert2);  
  
    Vektor result = v1 + v2;  
}
```

```
Vektor operator+(const Vektor& v1,  
                 const Vektor& v2){  
    Vektor erg(v1.dimension);  
    for (int i = 0; i < v1.dimension; ++i){  
        erg.daten[i] =  
            v1.daten[i] + v2.daten[i];  
    }  
    return erg;  
}
```

Wie viel Vektoren werden erzeugt?

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

Effizient oder nicht effizient?

```
void testPlusHelp() {  
    Vektor v1(400);  
    const V_BaseType wert1(11);  
    v1.Init(wert1);  
  
    Vektor v2(400);  
    const V_BaseType wert2(22);  
    v2.Init(wert2);  
  
    Vektor result = v1 + v2;  
}
```

```
Vektor operator+(const Vektor& v1,  
                 const Vektor& v2){  
    Vektor erg(v1.dimension);  
    for (int i = 0; i < v1.dimension; ++i){  
        erg.daten[i] =  
            v1.daten[i] + v2.daten[i];  
    }  
    return erg;  
}
```

v1, v2 --> 2
1mal im operator+
1mal beim return im operator +
Vektor result = v1 + v2; wird hier
„wegoptimiert“ werden.

Wie viel Vektoren werden
erzeugt?

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

Effizient oder nicht effizient?

```
void testPlusHelp() {  
    Vektor v1(400);  
    const V_BaseType wert1(11);  
    v1.Init(wert1);  
  
    Vektor v2(400);  
    const V_BaseType wert2(22);  
    v2.Init(wert2);  
  
    Vektor result(1)  
    result = v1 + v1 + v1;  
}
```

```
Vektor operator+(const Vektor& v1,  
                const Vektor& v2){  
    Vektor erg(v1.dimension);  
    for (int i = 0; i < v1.dimension; ++i){  
        erg.daten[i] =  
            v1.daten[i] + v2.daten[i];  
    }  
    return erg;  
}
```

Wie viel Vektoren werden erzeugt?

1. 6 + 1 Zuweisungsoperator-Aufruf
2. 7 + 1 Zuweisungsoperator-Aufruf
3. 8 + 1 Zuweisungsoperator-Aufrufe
4. 9 + 1 Zuweisungsoperator-Aufruf
5. 10 + 2 Zuweisungsoperator-Aufrufe

Effizient oder nicht effizient?

```
void testPlusHelp() {  
    Vektor v1(400);  
    const V_BaseType wert1(11);  
    v1.Init(wert1);  
  
    Vektor v2(400);  
    const V_BaseType wert2(22);  
    v2.Init(wert2);  
  
    Vektor result(1)  
    result = v1 + v1 + v1;  
}
```

Wie viel Vektoren werden erzeugt?

1. 6 + 1 Zuweisungsoperator
2. 7 + 1 Zuweisungsoperator
3. 8 + 1 Zuweisungsoperator
4. 9 + 1 Zuweisungsoperator
5. 10 + 2 Zuweisungsoperatoren

```
Vektor operator+(const Vektor& v1,  
                 const Vektor& v2){  
    Vektor erg(v1.dimension);  
    for (int i = 0; i < v1.dimension; ++i){  
        erg.daten[i] =  
            v1.daten[i] + v2.daten[i];  
    }  
    return erg;  
}
```

Result, v1, v2;

- 3mal für Vektor
- ```
result = v1 + v1 + v1;
```
- 2 mal in operator+
  - 2 mal in return in operator+
  - 1 mal Zuweisung

## Effizient oder nicht effizient?

```
void testPlusHelp() {
 Vektor v1(400);
 const V_BaseType wert1(11);
 v1.Init(wert1);

 Vektor v2(400);
 const V_BaseType wert2(22);
 v2.Init(wert2);

 Vektor result = v1 + v2;
}
```

```
Vektor operator+(const Vektor& v1,
 const Vektor& v2){
 Vektor erg(v1.dimension);
 for (int i = 0; i < v1.dimension; ++i){
 erg.daten[i] =
 v1.daten[i] + v2.daten[i];
 }
 return erg;
}
```

v1+v2 ist ein R-Wert. Dieser Vektor ist ein temporäres Objekt, Diesen Hilfsvektor können wir anders zuweisen.



## Kopierkonstruktor für R-Werte

Vektor **result** = va + vb;

```
Vektor::Vektor(const Vektor& v2) {
 dimension = v2.dimension;
 daten = new int[dimension];
 for (int i = 0; i < dimension; ++i) {
 daten[i] = v2.daten[i];
 }
}
```

```
Vektor::Vektor(Vektor && v2)
{
 dimension = v2.dimension;
 daten = v2.daten;
 v2.daten = nullptr;
}
```

v2/(bzw. va+vb) wird hinterher gar nicht mehr, gebraucht, d.h. es wird dafür der Destruktor aufgerufen.

Verbesserung: v2 als „Steinbruch“ zum „Ausschlachten“ verwenden.

## Zuweisungsoperator für R-Werte

```
result = va + vb;
```

```
Vektor&
Vektor::operator=(const Vektor& v2){
if (this != &v2) {
 delete[] daten;
 dimension = v2.dimension;
 daten = new int[dimension];
 for (int i = 0; i<dimension; ++i) {
 daten[i] = v2.daten[i];
 }
}
return *this;
}
```

```
Vektor&
Vektor::operator=(Vektor && v2){
 dimension = v2.dimension;
 daten = v2.daten;
 v2.daten = nullptr;
 v2.dimension = 0;
 return *this;
}
```

## Effizient oder nicht effizient?

```
void testPlusHelp() {
 Vektor v1(400);
 const int wert1(11);
 v1.Init(wert1);

 Vektor v2(400);
 const int wert2(22);
 v2.Init(wert2);

 Vektor result = v1 + v2;
 result = v1 + v1 + v1;
}
```

```
Vektor operator+(const Vektor& v1,
 const Vektor& v2){
 Vektor erg(v1.dimension);
 for (int i = 0; i < v1.dimension; ++i){
 erg.daten[i] =
 v1.daten[i] + v2.daten[i];
 }
 return erg;
}
```

Wie viel teure Operatoren verbleiben?

1. 4 Copy + 1 Zuweisungsoperator
2. 5 Copy + 0 Zuweisungsoperator
3. 4 Copy + 2 Zuweisungsoperator
4. 6 Copy + 0 Zuweisungsoperator
5. 6 Copy + 1 Zuweisungsoperator

## Effizient oder nicht effizient?

```
void testPlusHelp() {
 Vektor v1(400);
 const int wert1(11);
 v1.Init(wert1);
 Vektor v2(400);
 const int wert2(22);
 v2.Init(wert2);
 Vektor result = v1 + v2;
 result = v1 + v1 + v1;
}
```

```
Vektor operator+(const Vektor& v1,
 const Vektor& v2){
 Vektor erg(v1.dimension);
 for (int i = 0; i < v1.dimension; ++i){
 erg.daten[i] =
 v1.daten[i] + v2.daten[i];
 }
 return erg;
}
```

v1 erzeugen  
v2 erzeugen  
erg in operator+ erzeugen  
result erzeugen  
erg in operator+ erzeugen

Insgesamt also 5 *teure*  
Kopierkonstruktoren  
Plus 3 mal *billigen* Verschiebe-  
kopierkonstruktor (mit „&&“)  
Plus 1 mal *billigen* Verschiebe-  
Zuweisungsoperator (mit „&&“)

## Eine Klasse statt eingebauten Typ int oder double

```
class Vektor{
public:
...
 void Init(Guest w);
private:
 Guest* daten;
};
```

```
void testPlusHelp() {
 Vektor v1(400);
 const Guest wert1(11);
 v1.Init(wert1);
 Vektor v2(400);
 const Guest wert2(22);
 v2.Init(wert2);
 Vektor result = v1 + v2;
 result = v1 + v1 + v1;
}
```

```
Vektor& Vektor::operator=(const Vektor& v2){
if (this != &v2) {
 delete[] daten;
 dimension = v2.dimension;
 daten = new Guest[dimension];
 for (int i = 0; i<dimension; ++i) {
 daten[i] = v2.daten[i];
 }
 return *this; }
}
```

Insgesamt 6004 Guests  
Insgesamt 3600 Guest-Zuw.

Mit Verschiebe-Operatoren:  
Insgesamt 4404 Guests  
Insgesamt 2000 Guest-Zuw.

## Aufgabe

```
typedef int V_BaseType;

class Vektor{
public:
 Vektor(int dim);
 Vektor(const Vektor& v2);
 ~Vektor();
 Vektor& operator=(const Vektor& v2);

 void Init(V_BaseType wert);
 void Plus(const Vektor& v2, Vektor& erg);
 void Print() const;
 int GetDim() const {return dimension;}
 V_BaseType GetDatum(int index) const {return daten[index];}
 void Resize(int dim);

private:
 V_BaseType* daten;
 int dimension;
};
```

Implementieren Sie für Ihre Vektor-Klasse ebenfalls Verschiebe-Konstruktor und Verschiebe-Zuweisungs-Operator

Überlegen Sie sich, wie Sie nachweisen könnten, dass wirklich die Verschiebe-Operatoren und nicht die „normalen“ Operatoren aufgerufen wurden.

## Aufgabe

Implementieren Sie für Ihre Matrix-Klasse ebenfalls Verschiebe-Konstruktor und Verschiebe-Zuweisungs-Operator

Überlegen Sie sich, wie Sie nachweisen könnten, dass wirklich die Verschiebe-Operatoren und nicht die „normalen“ Operatoren aufgerufen wurden.

```
typedef double BasisTyp;
class Matrix {
public:
 Matrix(int z, int sp);
 Matrix(const Matrix& m2);
 ~Matrix();

 void Resize(int z, int sp);
 void Setze(int z, int sp, BasisTyp wert);
 BasisTyp Lese(int z, int sp) const;
 int GetSpAnz() const { return spanz; }
 int GetZeAnz() const { return zanz; }

 Matrix& operator=(const Matrix& m);

private:
 int GetPos(int z, int sp) const;
 BasisTyp* arr;
 int zanz;
 int spanz;
};
```

## Eine Implementierung von move

Vektor v, v1, v2;

```
void MeineFunk(const Vektor& v);
```

```
void MeineFunk(Vektor&& v);
```

```
template <typename T> T&& move(T& t) {return t;}
```

```
MeineFunk(move(v2));
```