1 Vector

Um die Move-Semantik, und somit effizientes Verschieben, von eigenen Klassen zu erlauben, müssen entsprechende Konstruktoren und Operatoren implementiert werden. In dieser Aufgabe sollen Sie eine primitive Alternative zum `std::vector` implementieren, welche nur die Methode `PushBack` zur Verfügung stellt. Zudem soll der Container nur mit der Klasse `Item` als Value-Elemente funktionieren und statisches Memory zur internen Speicherung verwenden. Der Header dieser Implementation sollte etwa folgendermassen aussehen:

vector.h

```
class Vector {
  static constexpr size_t kMaxMemory = 4;

 public:
  Vector();
  Vector(const Vector& other);

  void PushBack(const Item& item);

 private:
  Item memory_[kMaxMemory];
  size_t size_;
};
```

1.1 Aufgabe

a) Implementieren Sie die Klassen `Item` und `Vector` mit jeweils Copy-Konstruktor und erzeugen Sie darin jeweils ein Log auf die Konsole. Folgendes Hauptprogramm soll dann kompilieren und laufen gelassen werden können.

main.cpp

```
#include <utility>
#include <iostream>

#include "item.h"
#include "vector.h"

int main() {
  Vector vector_a;

  Item item_a;
  std::cout << "normal push" << std::endl;
  vector_a.PushBack(item_a);
```

```
13    std::cout << "move push" << std::endl;
14    vector_a.PushBack(std::move(item_a));
15    std::cout << "inplace push" << std::endl;
16    vector_a.PushBack(Item());
17
18    std::cout << "normal assign" << std::endl;
19    Vector vector_b = vector_a;
20
21    std::cout << "move assign" << std::endl;
22    Vector vector_c = std::move(vector_a);
23  }
```

b) Ergänzen Sie nun die beiden Klassen mit der Implementierung der Move-Semantik. Achten Sie dabei darauf, dass im Move des `Vector`s auch dessen verwaltete `Item`s verschoben werden sollen. Korrigieren Sie etwaige andere Unterschiede, bis Sie folgenden Output erhalten:

```
1  normal push
2    Item: copy-assign
3  move push
4    Item: move-assign
5  inplace push
6    Item: move-assign
7  normal assign
8    Vector: copy-ctor
9    Item: copy-assign
10   Item: copy-assign
11   Item: copy-assign
12 move assign
13   Vector: move-ctor
14   Item: move-assign
15   Item: move-assign
16   Item: move-assign
```

1.2 Lösung

Folgend die Lösungen. Beachten Sie, dass die Move-Semantik immer optional ist und bei nicht Vorhandensein auf die normalen Copy-Operationen zurück fällt.

item.h

```
1  #pragma once
2
3  class Item {
4   public:
5    Item();
6
```

```
7    Item(const Item& other);
8    Item& operator=(const Item& other);
9
10   Item(Item&& other);
11   Item& operator=(Item&& other);
12 };
```

item.cpp

```
1  #include "item.h"
2
3  #include <iostream>
4
5  Item::Item() {
6  }
7
8  Item::Item(const Item& other) {
9    std::cout << "  Item: copy-ctor" << std::endl;
10 }
11
12 Item& Item::operator=(const Item& other) {
13   std::cout << "  Item: copy-assign" << std::endl;
14   return *this;
15 }
16
17 Item::Item(Item&& other) {
18   std::cout << "  Item: move-ctor" << std::endl;
19 }
20
21 Item& Item::operator=(Item&& other) {
22   std::cout << "  Item: move-assign" << std::endl;
23   return *this;
24 }
```

vector.h

```
1  #pragma once
2
3  #include <cstddef>
4
5  #include "item.h"
6
7  class Vector {
8    static constexpr size_t kMaxMemory = 4;
9
10  public:
11   Vector();
12
13   Vector(const Vector& other);
```

```
14    void PushBack(const Item& item);
15
16    Vector(Vector&& other);
17    void PushBack(Item&& item);
18
19   private:
20    Item memory_[kMaxMemory];
21    size_t size_;
22  };
```

vector.cpp

```
1   #include "vector.h"
2
3   #include <cassert>
4   #include <iostream>
5
6   Vector::Vector() : size_(0) {
7   }
8
9   Vector::Vector(const Vector& other) : size_(other.size_) {
10    std::cout << "  Vector: copy-ctor" << std::endl;
11    std::copy(other.memory_, other.memory_ + other.size_, memory_);
12  }
13
14  void Vector::PushBack(const Item& item) {
15    assert(size_ < kMaxMemory);
16    memory_[size_++] = item;
17  }
18
19  Vector::Vector(Vector&& other) : size_(other.size_) {
20    std::cout << "  Vector: move-ctor" << std::endl;
21    for (size_t i = 0; i < other.size_; ++i) {
22      memory_[i] = std::move(other.memory_[i]);
23    }
24  }
25
26  void Vector::PushBack(Item&& item) {
27    assert(size_ < kMaxMemory);
28    memory_[size_++] = std::move(item);
29  }
```