C++ - Einführung in die Programmiersprache

Standard Templates

C++ - Standardbibliothek

- Standardisierte Sammlung von häufig vorkommenden Funktionen und Klassen.
- 1998 erste Standardbibliothek bestehend aus den IO-Streams, Strings und der Standard Template Library.
- Erweiterung zum C++ 11-Standard: Reguläre Ausdrücke, Smart Pointer, Hashtabellen, Zufallszahlen und Zeit.

Bücher zur Standardbibliothek

- Rainer Grimm: C++-Standardbibliothek - kurz & gut. O'Reilly. 1. Auflage.

Informationen im Web

- http://www.cplusplus.com/reference/
- http://en.cppreference.com/w/

Bereiche der Standardbibliothek

<table>
<tr><td>Speicherverwaltung</td><td rowspan="3">Container, Iteratoren, Funktoren und Algorithmen der Standard Template Library</td></tr>
<tr><td>Ein- und Ausgabe-Streams</td></tr>
<tr><td>Lokale Einstellungen</td></tr>
<tr><td>Strings</td><td>Exception Handling</td></tr>
<tr><td>Hilfsklassen</td><td>Numerische Berechnungen</td></tr>
<tr><td></td><td>C-Header-Dateien</td></tr>
</table>

Standard Template Library (STL)

- Klassen und Methoden zur Datenverwaltung.
- Vordefinierte Templates für Vektoren, Warteschlangen etc.

Bücher zur Standardbibliothek

- Ivor Horten: Using the C++ Standard Template Libraries. Apress.

Informationen im Web

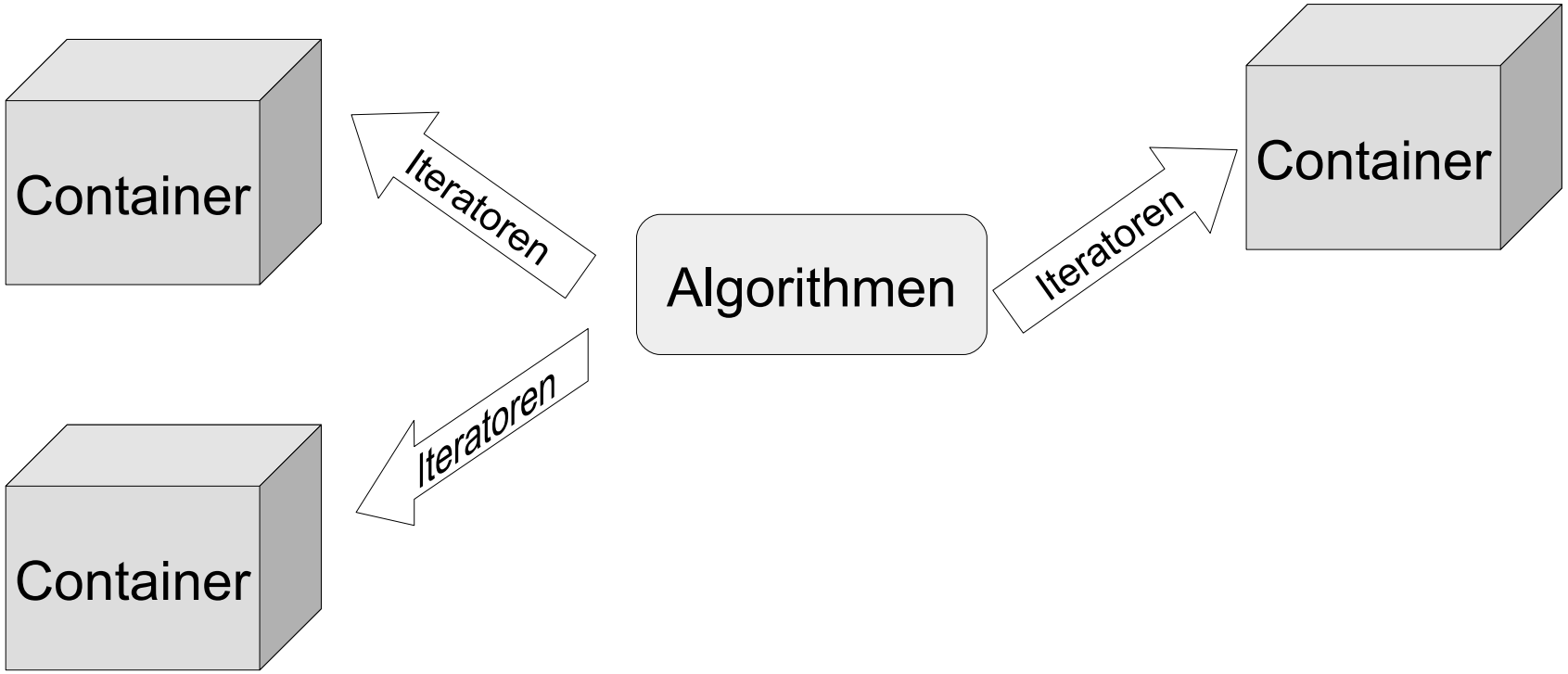
- http://www.cplusplus.com/reference/stl/
- https://www.sgi.com/tech/stl/

Komponenten

- Container zur Verwaltung von Objekten.
- Iteratoren für den Zugriff auf Elemente in einem Container.
- Algorithmen verarbeiten die Elemente in einem Container.

Leibniz Universität Hannover

Zusammenarbeit



Container

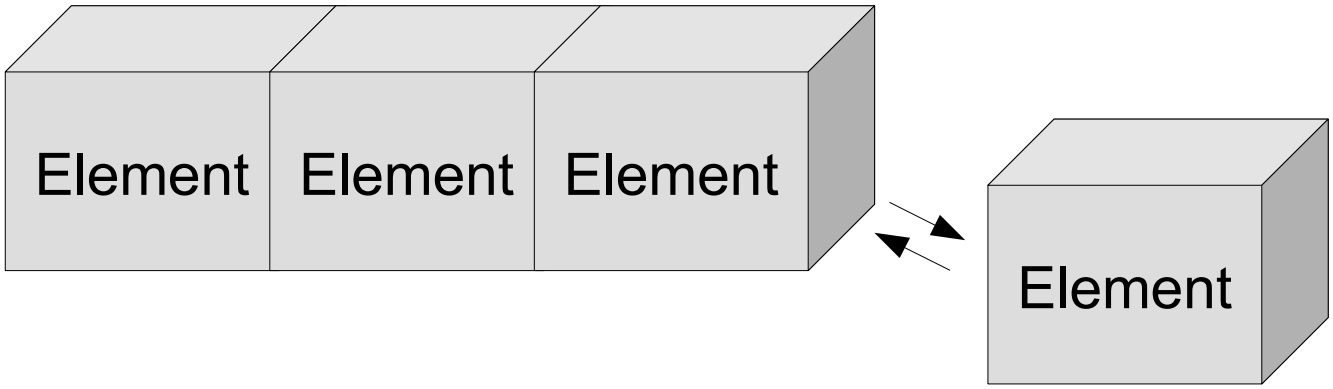
- Container nehmen andere Objekte auf.
- Jeder der Container ist als Template realisiert und kann jeden beliebigen Datentyp aufnehmen.
- Die Elemente eines Containers können aus einem einfachen Datentyp wie zum Beispiel Integer, Strings oder Klassen bestehen.
- Die Container werden in sequentielle und assoziative Container unterteilt.

Sequentielle Container

- Unsortierte, lineare Ablage im Speicher.
- Das Anfügen oder Entfernen von Elementen am Anfang oder Ende wird sehr schnell ausgeführt.
- Eine Suche in einem unsortierten Container ist sehr langsam.

<vector>

- Implementierung eines Arrays.
- Ein wahlfreier Zugriff auf die Elemente mit Hilfe eines Indizes ist möglich.
- Die Anzahl seiner Elemente ist dynamisch veränderbar.
- Elemente können nur am Ende hinzugefügt werden



Leibniz Universität Hannover

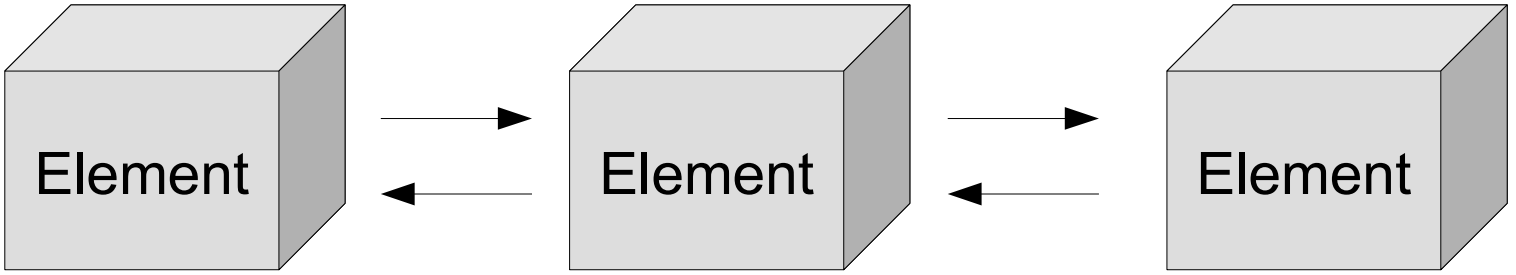
Beispiel

Beispiele/cppGen_001_Vector...

```
#include <iostream>
#include <vector>

using namespace std;

int main(){
   const int MaxAnzahl = 10;
   int count;
   vector <int> dynamicArray;

   for (count = 0; count < MaxAnzahl; count++){
     dynamicArray.push_back(count);
     cout << dynamicArray.at(count) << "; ";
     dynamicArray.at(count) = dynamicArray.at(count) * 3;
     cout << dynamicArray.at(count) << "; ";
   }

   return 0;
}
```

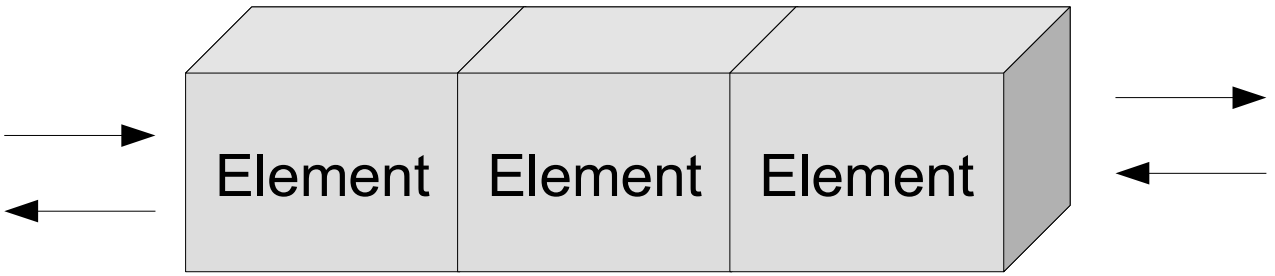
<list>

- Doppelt verkettete Liste.
- Jedes Element hat einen Zeiger auf seinen Nachfolger und Vorgänger.
- Die Liste wird mit Hilfe von Iteratoren durchlaufen.



<deque>

- Warteschlange.
- Elemente können am Anfang und Ende eingefügt werden.
- Ein Zugriff über ein Index ist möglich.



Leibniz Universität Hannover

Beispiel

Beispiele/cppGen_001_Deque...

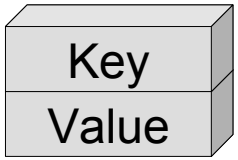
```
#include <deque>
#include <iostream>

using namespace std;

int main(){
    deque<string> formen(5);
    int count;

    formen.push_front("Rechteck");
    formen.push_back("Dreieck");
    formen.push_front("Kreis");
    formen.pop_back();

    for(count = 0; count < formen.size(); ++count){
        cout << formen.at(count) << endl;
    }

}
```

Assoziative Container

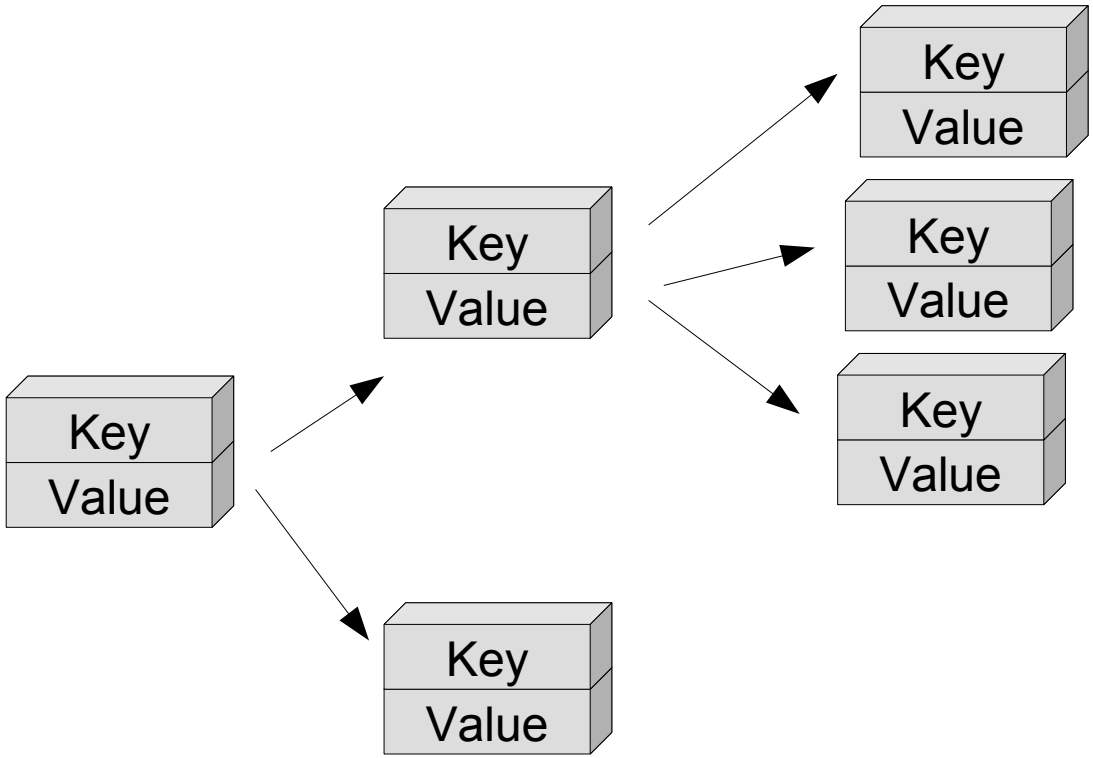
- Sortierte Ablage in Binärbäumen.
- Binärbäume können sehr schnell durchsucht werden.
- Ein Anfügen oder Entfernen von Elementen in einem Baum erfolgt sehr langsam.

<map>

- Elemente, die aus einem Schlüssel und einen dazugehörigen Wert bestehen.
- Jeder Schlüssel ist einmal vorhanden.
- Die Elemente werden nach dem Schlüssel sortiert.
- Es können Tabellen, die auf Listen basieren oder binäre Bäume abgebildet werden.



Grafische Darstellung



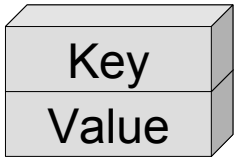
Beispiel

Beispiele/cppGen_001_Map...

```
#include <set>
#include <iostream>
#include <string>

using namespace std;

int main(){
   const int Anzahl = 3;
   const char* Obst[Anzahl] = {"Banane", "Apfel", "Erdbeere"};
   string slogan("Verkauf von Obst und Gemüse");
   set<string> setGemuese;
   set<char> setSlogan;

   set<string> setObst(Obst, Obst + Anzahl);
   setGemuese.insert("Weisskohl");
   setGemuese.insert("Kartoffeln");
   setSlogan.insert(slogan.begin(), slogan.end());

}
```

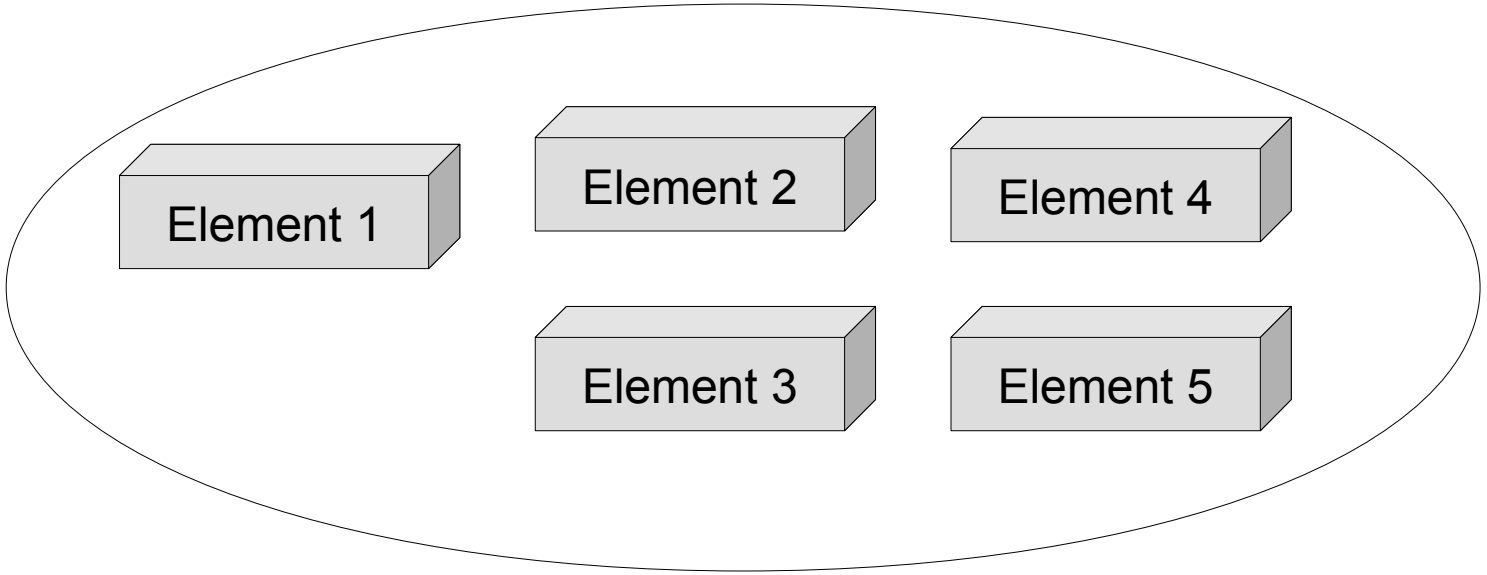
<multimap>

- Elemente, die aus einem Schlüssel und einen dazugehörigen Wert bestehen.
- Die Schlüssel können mehrmals vorkommen.
- Die Elemente werden nach dem Schlüssel sortiert.



<set>

- Sortierte Menge von Objekten.



Leibniz Universität Hannover

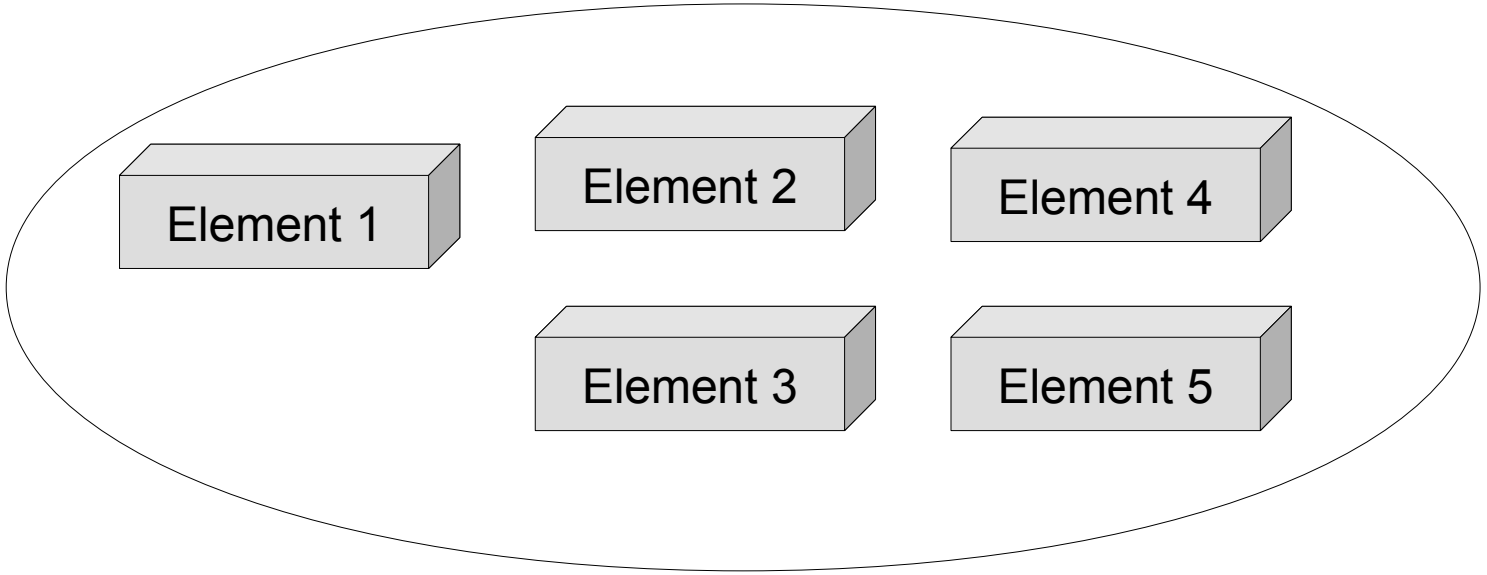
Beispiel

```
#include <map>
#include <iostream>
#include <string>

using namespace std;

int main(){
   map<string, string>adressbook;

   adressbook["aue"] = "aue@rrzn.uni-hannover.de";
   pair<string, string>paar1("firmaXYZ","firmaXYZ@myFirma.de");
   adressbook.insert(paar1);
   typedef pair<string, string>element;
   element paar2("institutXYZ","institutXYZ@myInstitut.de");
   adressbook.insert(paar2);

   cout << "firma XYZ hat folgende E-Mail-Adresse ";
   cout << adressbook["firmaXYZ"];
};
```

Beispiele/cppGen_001_Map...

<multiset>

- Sortierte Menge von Objekten.
- Die Elemente können mehrmals vorkommen.



Leibniz Universität Hannover

Beispiel

Beispiele/cppGen_001_Set...

```
#include <map>
#include <iostream>
#include <string>

using namespace std;

int main(){
   map<string, string>adressbook;

   adressbook["aue"] = "aue@rrzn.uni-hannover.de";
   pair<string, string>paar1("firmaXYZ","firmaXYZ@myFirma.de");
   adressbook.insert(paar1);
    typedef pair<string, string>element;
   element paar2("institutXYZ","institutXYZ@myInstitut.de");
   adressbook.insert(paar2);

   cout << "firma XYZ hat folgende E-Mail-Adresse ";
   cout << adressbook["firmaXYZ"];
};
```

Einbindung eines Containers

```
#include <vector>
```

- Für jeden Container muss die entsprechende Header-Datei mit Hilfe von #include eingebunden werden.

Deklaration eines Containers

```
vector<int> intZahl;
vector<float> floatZahl(2);
vector<char> charZeichen(5);
```

Beispiele/cppGen_001_VectorErzeugen...

- Deklaration eines Containers von dem Datentyp ....
- Welche Vorlage wird genutzt?
- Für welche Datentyp wird die Vorlage genutzt?
- Wie heißt der Container?
- Mit welchen Werten wird der Container initialisiert?

Container vom Datentyp ...

vector	<	int	>	dynamicArray	;
Container	<	Datentyp	>	variablenname	;

- Welche Vorlage wird genutzt?. In diesem Beispiel wird die Vorlage vector für die Erstellung eines Containers genutzt.
- In spitzen Klammern wird der Datentyp für die Elemente des Containers angegeben. In diesem Beispiel werden Elemente vom Datentyp Integer in dem Container gespeichert.
- Jeder Container hat einen Namen. Der Name ist frei wählbar.

Nutzung des Standard-Konstruktors

```
vector<int> intZahl;
```

- Falls dem Namen des Containers keine runden Klammern folgen, wird der Container mit dem Standardkonstruktor erzeugt.
- Der Container wird entsprechend der Standardwerte erstellt.

Nutzung von allgemeinen Konstruktoren

```
vector<float> floatZahl(2);
vector<char> charZeichen(5);
```

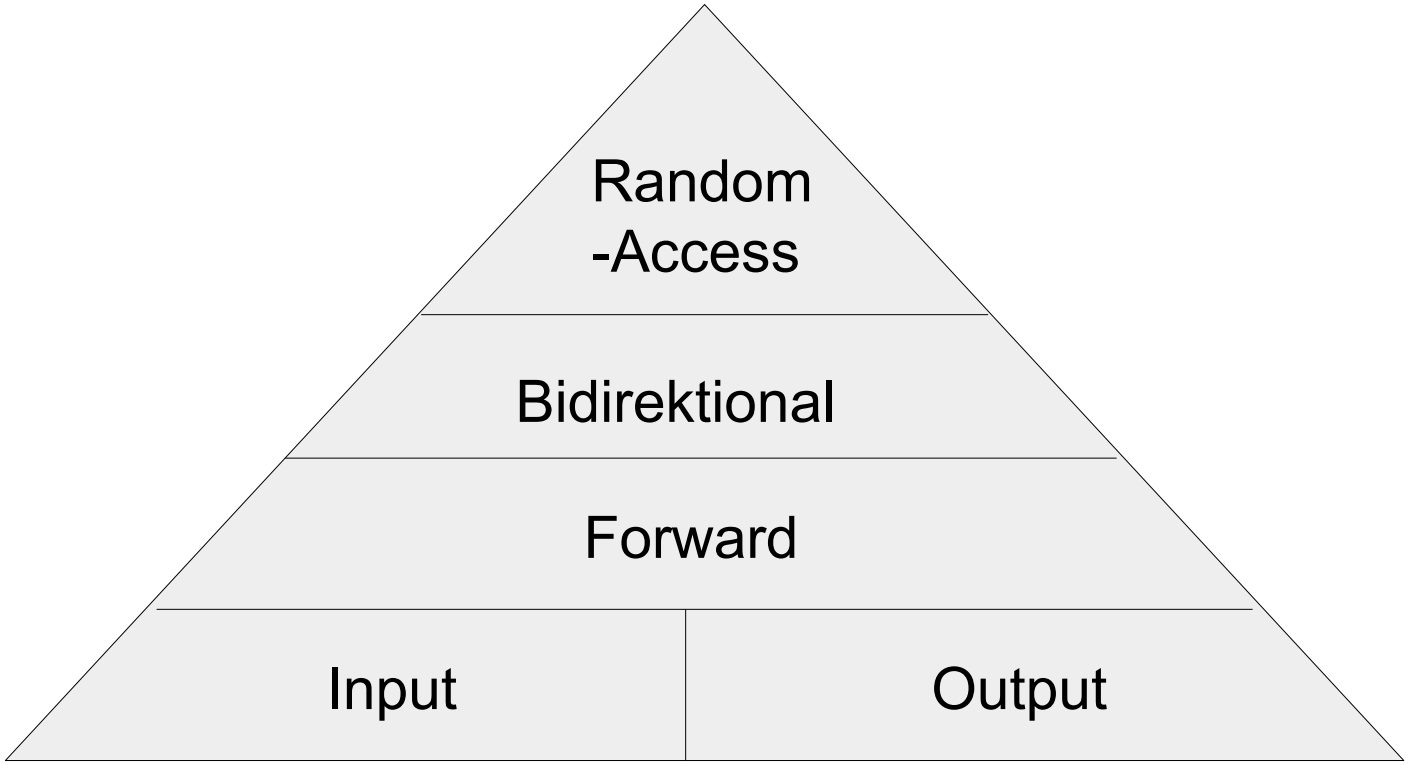
- Dem Namen des Containers folgen runden Klammern.
- In den runden Klammern wird entsprechend der Vorlage die Parameter, getrennt durch Kommata, übergeben.
- Entsprechend der Anzahl und / oder des Datentyps der Parameter wird ein passender Konstruktor ausgewählt.
- Der Container wird mit Hilfe der übergebenen Parameter erzeugt.

Iteratoren

- Zeiger auf Elemente in einem beliebigen Container.
- Zugriff auf die Elemente in einem Container.
- Positionszeiger innerhalb eines Containers.

Leibniz Universität Hannover

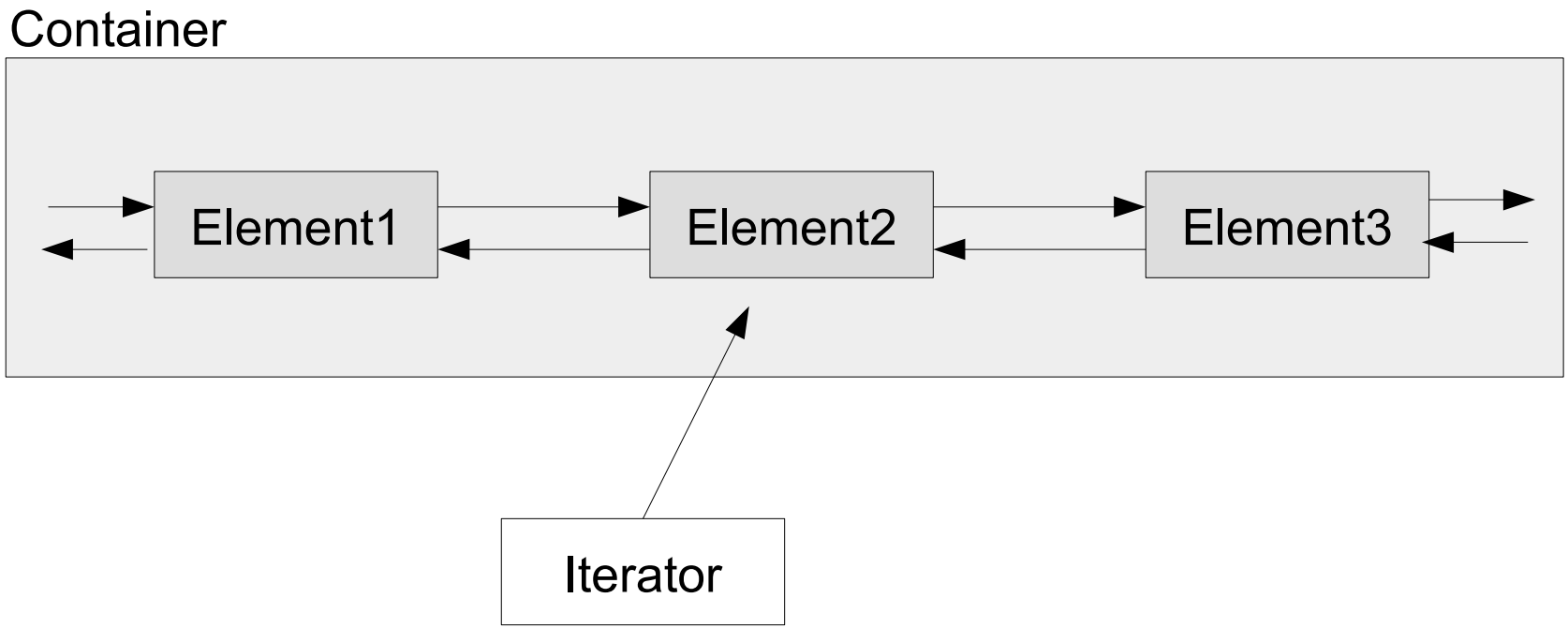
Hierarchie der Iteratoren



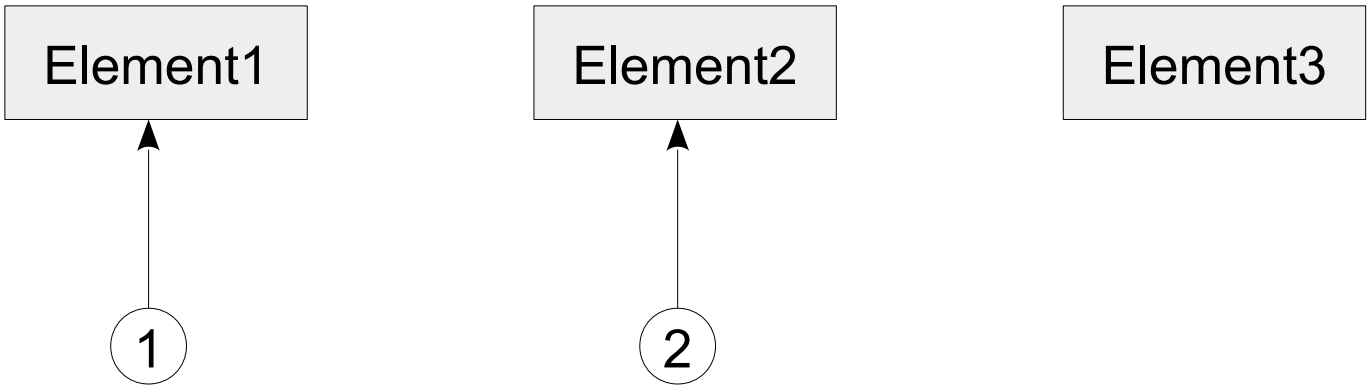
Erläuterung

- Iteratoren sind hierarchisch geordnet.
- Um so weiter man nach unten in der Pyramide geht, um so spezieller werden die Iteratoren.
- Um so weiter man nach oben in der Pyramide geht, um so allgemeiner werden die Iteratoren.

Beispiel

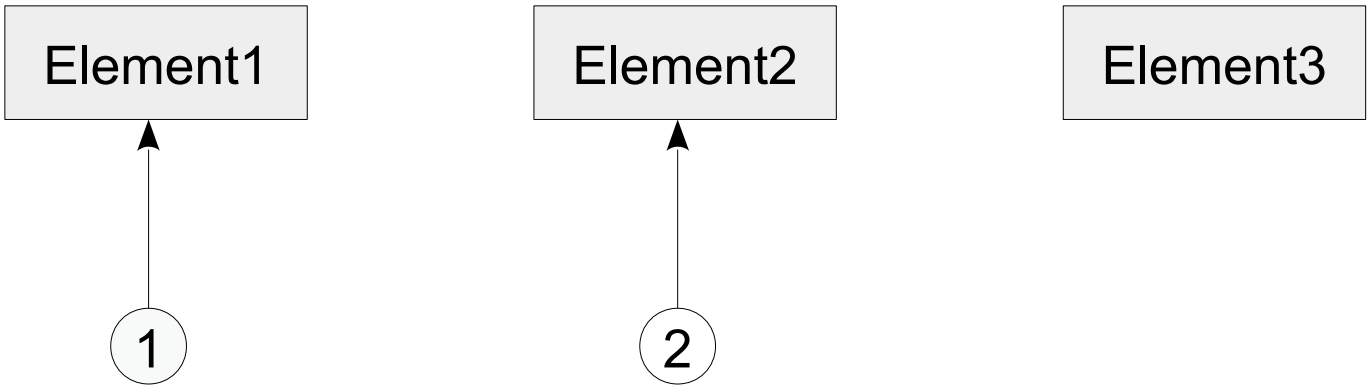


Input-Iteratoren



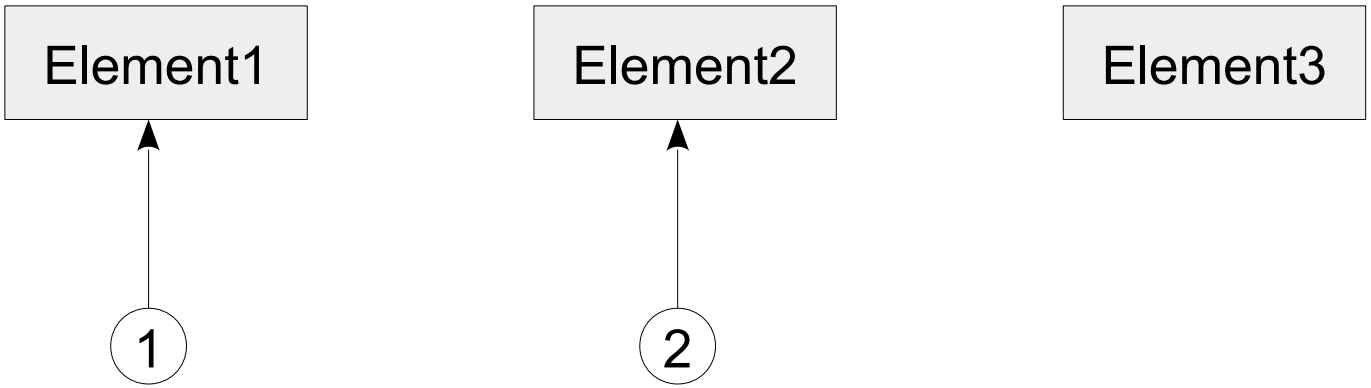
- Lesezugriff auf Elemente in einem Container.
- Der Iterator kann sich nur vorwärts bewegen. Der Iterator greift immer auf das nächste Element zu.
- Ein einmal gelesenes Element kann nicht wieder gelesen werden.

Output-Iteratoren



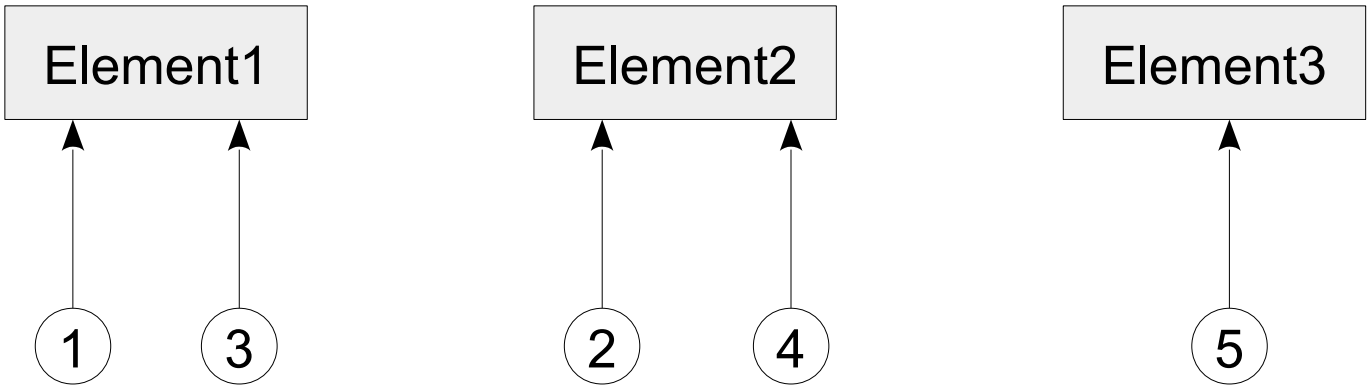
- Schreibzugriff auf Elemente in einem Container.
- Der Iterator kann sich nur vorwärts bewegen. Der Iterator greift immer auf das nächste Element zu.
- Sequentieller Zugriff.

Forward-Iteratoren



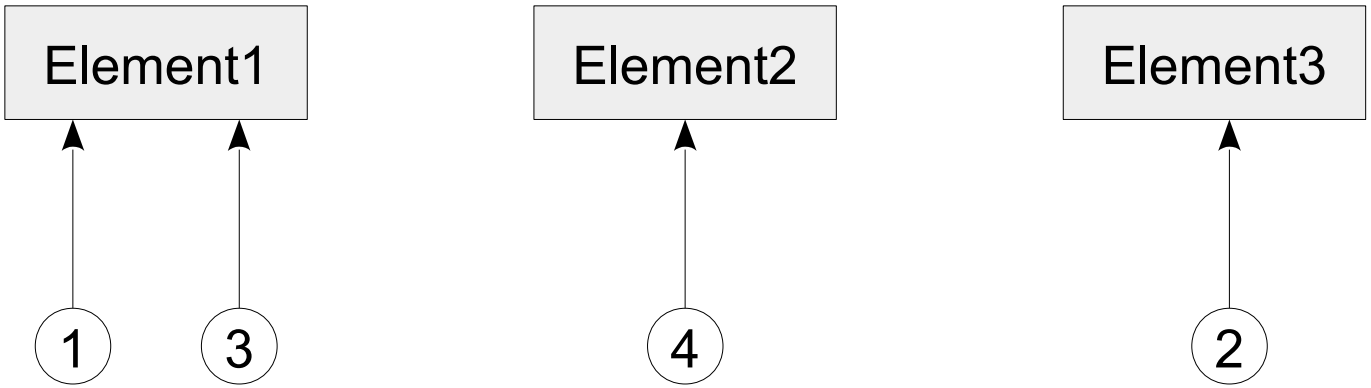
- Mischung aus Input- und Output-Operator.
- Sequentieller Schreib- und Lesezugriff auf die Elemente in einem Container.

Bidirectional-Iteratoren



- Lese- und Schreibzugriff auf Elemente in einem Container.
- Zugriff auf das vorherige und nächste Element.

Random-Access-Iteratoren



- Lese- und Schreibzugriff auf Elemente in einem Container.
- Wahlfreier Zugriff auf die Elemente.

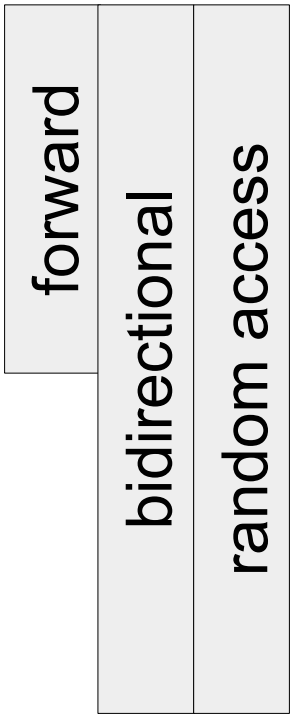
Leibniz Universität Hannover

Operatoren für Input- und Output-Iteratoren

	input	output	forward	bidirectional	random access
type itA(objB)					
++itA					
itA++					
itA == itB					
itA != itB					
*itA					
itA->methode					
*itA = element					
*itA++ = element					

Operatoren für Forward- und Bidirectional-Iteratoren

type itA
type itA()
{itB=itA; *itA++; *itB}
--itA
itA--
*itA--



Leibniz Universität Hannover

Operatoren für Random Access-Iteratoren

itA + itB
itA - itB
itA < itB
itA > itB
itA <= itB
itA >= itB
itA += itB
itA -= itB
itA[index]

random access

Leibniz Universität Hannover

Iteratoren von Containern

Beispiele/cppGen_001_VectorIterator...

```
#include <vector>
#include <iostream>

using namespace std;

int main(){
    vector<int> feld(5);
    vector<int>::iterator ptrVektor;
    int count;

    count = 1;

    for(ptrVektor=feld.begin(); ptrVektor < feld.end(); ptrVektor++)
    {
        cout << "Vektor-Element " << count << ": " << *ptrVektor << '\n';
        count++;
    }
}
```

Deklaration des Iterators

iterator	ptrVektor	;
iterator	name	;

- ptrVektor ist vom Typ iterator.
- Jeder Container besitzt einen Iterator, um auf die darin enthaltenen Elemente zuzugreifen

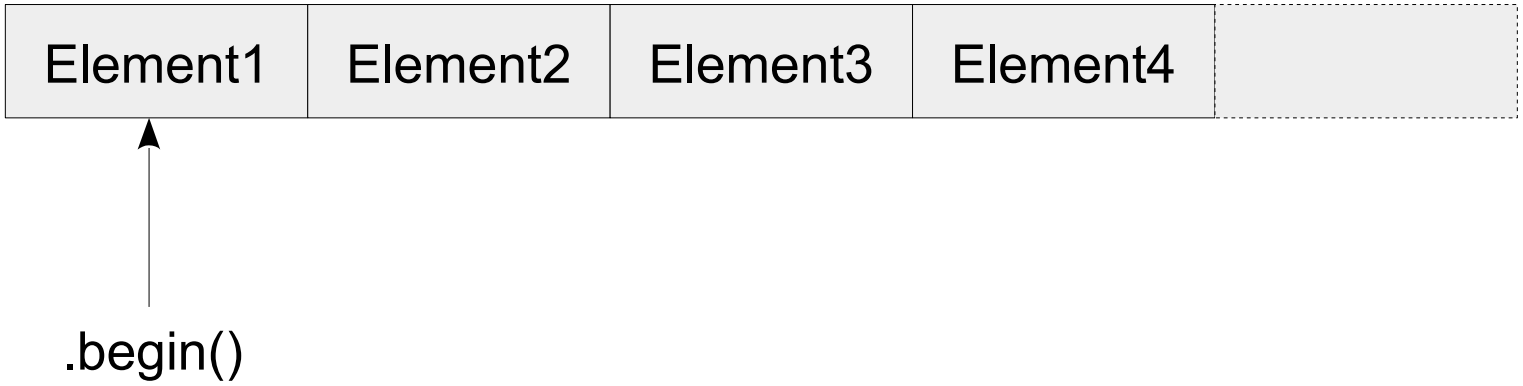
Verweis auf ...

vector	<	int	>	::	iterator	ptrVektor	;
Container	<	Datentyp	>	::	iterator	name	;

- Links vom Bereichsoperator (::) wird der Container beschrieben, auf deren Elemente der Iterator verweisen soll.
- Der Iterator verweist in diesem Beispiel auf ein Array, in dem Ganzzahlen gespeichert werden.
- Der Iterator und die Elemente, auf die verwiesen wird, sollten den gleichen Datentyp haben.

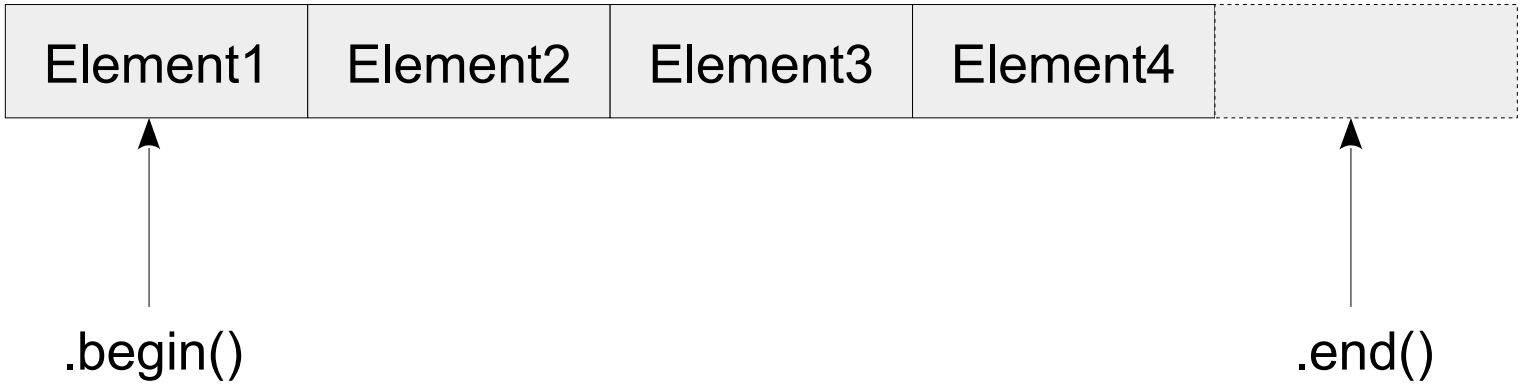
Verweis auf das erste Element

```
vector<int> feld(5);
vector<int>::iterator ptrVektor;

ptrVektor = feld.begin()
```



Verweis auf das „letzte“ Element

```
vector<int> feld(5);
vector<int>::iterator ptrVektor;

ptrVektor = feld.end()
```



Verweis auf ein beliebiges Element

```
vector<int> feld(5);
vector<int>::iterator ptrVektor;

ptrVektor = feld.begin() + 2;
```

- Falls ein Random Access-Iterator genutzt wird, kann der Iterator mit Hilfe der arithmetischen Operatoren verschoben werden.

Zuweisung eines Wertes

```
for(count=0; count < anzahlElement; count++)
{
        feld[count] = count;
}
```

- Mit Hilfe des Index wird ein Element in dem Container vector identifiziert.
- Dem Element wird ein Wert entsprechend des Datentyps des Containers zugewiesen.

Hinzufügung von Elementen

```
vector<int> feld(5);
vector<int>::iterator ptrVektor;

ptrVektor = feld.begin() + 2;
feld.insert(ptrVektor, 3);
```

Beispiele/cppGen_001_VectorInsert...

- Mit Hilfe der Methode .insert() wird ein Element in den Container vector an einer bestimmten Position eingefügt.
- Alle nachfolgenden Elemente werden entsprechend verschoben.

Parameter der Methode

```
vector<int> feld(5);
vector<int>::iterator ptrVektor;

ptrVektor = feld.begin() + 2;
feld.insert(ptrVektor, 3);
```

- An welcher Position wird das Element eingefügt? Der Iterator verweist auf die entsprechende Position. Das Element wird an der dritten Stelle eingefügt.
- Welcher Wert wird an der Position gespeichert. In diesem Beispiel wird der Wert 3 gespeichert.

Lesen von allen Elementen

```
vector<int> feld(5);
vector<int>::iterator ptrVektor;

for(ptrVektor=feld.begin(); ptrVektor < feld.end(); ptrVektor++)
{
        cout << "Vektor-Element: " << *ptrVektor << '\n';
}
```

Beispiele/cppGen_001_VectorInsert...

- Mit Hilfe einer for-Schleife kann der Container vollständig durchlaufen werden.
- Der Iterator wird inkrementiert. Der Iterator greift auf das nächste Element zu.

Andere Möglichkeit seit C++11

```
vector<int> feld(5);
vector<int>::iterator ptrVektor;

for(const auto& element : feld)
{
        cout << "Vektor-Element " << element << '\n';
}
```

Beispiele/cppGen_001_VectorInsert...

Aufbau der for each -Schleife

for(const auto& element : feld)	Schleifenkopf

{
cout << "Element " << element << '\n';
}
	Schleifenrumpf

Schleifenkopf

for	(	const auto& element	:	feld	)
for	(	datentyp element	:	arrayName	)

- Der Schleifenkopf beginnt mit dem Schlüsselwort for.
- Dem Schlüsselwort folgt die Anweisung „Für jedes Element in dem Array“ in runden Klammern.
- Das Element links bezieht sich auf das Array rechts vom Doppelpunkt.
- Das Element und das Array sollten den gleichen Datentyp besitzen.

Hinweise

- Das Array wird vollständig vom ersten bis zum letzten Element durchlaufen.
- Die Array-Elemente können nicht verändert werden. Die Werte des Array-Elements werden an die Variable „call by value“ automatisiert übergeben.

Konstante Referenz

const	<	int	>	::	iterator	ptrVektor	;
Container	<	Datentyp	>	::	iterator	name	;

- Links vom Bereichsoperator (::) wird der Container beschrieben, auf deren Elemente der Iterator verweisen soll.
- Der Iterator verweist in diesem Beispiel auf ein Array, in dem Ganzzahlen gespeichert werden.
- Der Iterator und die Elemente, auf die verwiesen wird, sollten den gleichen Datentyp haben.

Konstante Referenz

const	auto&	element
const	auto&	refName

- Mit Hilfe des Schlüsselwortes const wird eine Konstante deklariert. Der Wert, der referenziert wird, kann nicht verändert werden.
- Mit Hilfe des kaufmännischen Und wird eine Referenz von einem Datentyp … gekennzeichnet.
- Der Name der Referenz ist frei wählbar.

Datentyp auto

const	auto&	element
const	auto&	refName

- Mit Einführung von C++11 hat sich die Bedeutung des Schlüsselwortes auto verändert.
- Der Datentyp der Variablen, der Referenz etc. wird in Abhängigkeit der Initialisierung automatisch vom Compiler festgelegt.

Leibniz Universität Hannover

Iteratoren für Arrays

Beispiele/cppGen_001_IteratorArray...

```
#include <iostream>
#include <iterator>
#include <algorithm>
using namespace std;

int main(){
    char myArray[] = {'a', 'b', 'c', 'd'};
    int index = 0;

    transform(begin(myArray), end(myArray), begin(myArray), [](char c) {return toupper(c);});

    auto beginIter = begin(myArray);
    auto endIter = end(myArray);

    for(auto iter = beginIter; iter != endIter; iter++){
        cout << "Element " << index << ": " << *iter << '\n';
        index++;
    }
};
```

Einbindung von Iteratoren

```
#include <iterator>
```

- Iteratoren sind in der Header-Datei <iterator> implementiert.

Algorithmen

- Verarbeitung von Elementen mit Hilfe von Iteratoren in einem Container
- Vordefinierte Funktionen für einen Container.

Leibniz Universität Hannover

… im Web

- http://en.cppreference.com/w/cpp/algorithm
- http://www.cplusplus.com/reference/algorithm/

Einbindung von Algorithmen

```
#include <algorithm>
```

- Algorithmen sind in der Header-Datei <algorithm> implementiert.

Beispiel

```
transform(begin(myArray),
          end(myArray),
          begin(myArray),
          [](char c) {return toupper(c);}
         );
```

Beispiele/cppGen_001_IteratorArray...

- Führt eine Funktion in einem Bereich von Elementen aus.

Welche Elemente werden bearbeitet?

```
transform(begin(myArray),
          end(myArray),
          begin(myArray),
          [](char c) {return toupper(c);}
         );
```

- Alle Elemente zwischen dem ersten und zweiten Parameter werden bearbeitet.
- Der erste Parameter kennzeichnet das erste Element, welches bearbeitet werden soll.
- Das vor dem zweiten Parameter liegende Element beschreibt das letzte zu bearbeitende Element.

Wo startet die Bearbeitung?

```
transform(begin(myArray),
          end(myArray),
          begin(myArray),
          [](char c) {return toupper(c);}
         );
```

- Der dritte Parameter gibt an, mit welchen Element innerhalb des Bereichs begonnen werden soll.
- In diesem Beispiel wird mit dem ersten Element begonnen.

Wie werden die Elemente bearbeitet?

```
transform(begin(myArray),
          end(myArray),
          begin(myArray),
          [](char c) {return toupper(c);}
         );
```

- Der vierte Parameter beschreibt, wie die Elemente bearbeitet werden sollen.
- In diesem Beispiel wird der Funktion ein Array von dem Datentyp `char` übergeben. Das Element wird in einen Großbuchstaben umgewandelt und zurückgegeben.