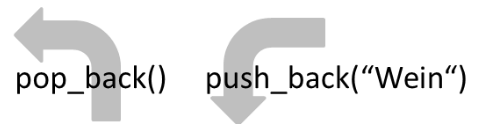
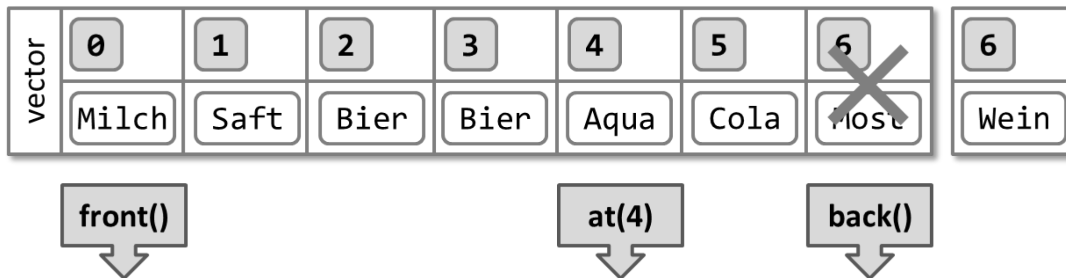
Ein Vektor ist ein dynamisches Array mit automatisch Speicherverwaltung.
Auf die Elemente in einem Vektor kann genauso effizient wie auf die Elemente in einem Array zugegriffen werden, mit dem Vorteil, dass sich die Größe von Vektoren dynamisch ändern kann.

	Deklarierung (ggf. auch mit Initialisierung)
①	`#include <vector>`
❶	`//vector <Typ> bezeichner;`
❷	`//vector <Typ> bezeichner{val₁, val₂, ...};`
②	`//vector <string> vs;`
③	`vector <string> vs{"Milch","Saft","Bier",\ "Bier","Aqua","Cola","Most"};`

Hinweise:

- Mit ❶ und ❷ werden nur einige von verschieden Deklarationsmöglichkeiten dargestellt. Als Typ-Angabe sind auch eigene Klassendeklarationen und somit das Speichern von Objekten möglich.
- Durch ③ wird der bildlich dargestellte **vector** über Strings erzeugt (deklariert und initialisiert).





In dieser Bibliothek sind auch Methoden zum Zugriff und zur Manipulation definiert:

Syntax	Beschreibung	
`at(n)`	Repräsentiert das n.te Element.	`vs.at(4) -> "Aqua"`
`size()`	Liefert die Anzahl aller Elemente.	`vs.size() -> 7`
`pop_back()`	Entfernt das letzte Element.	`vs.pop_back();`
`push_back(val)`	Fügt die Daten aus **val** ans Ende an.	`vs.push_back("Wein");`
`front()`	Liefert eine Referenz auf das erste Element.	`vs.front() -> "Milch"`
`back()`	Liefert eine Referenz auf das letzte Element.	`vs.back() -> (jetzt) "Wein"`
`= (Operator)`	Erstellt eine Kopie von vector<Typ>.	`vs_copy = vs`
`empty()`	Prüft, ob vector<Typ> leer ist.	`if(vs.empty()){ ... }`
`clear()`	Entfernt alle Elemente aus vector<Typ>.	`vs.clear()`

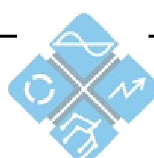
Eine Iteration beschreibt in der Informatik den schrittweisen, wiederholten (sequenziellen) Zugriff auf Elemente in einer Datenstruktur. Neben der zählergesteuerten Wiederholung gibt es auch ein sogenanntes foreach-Konstrukt.

In beiden Varianten ist kein Hinzufügen oder Entfernen von Elementen zulässig!

zählergesteuert	foreach
`for ( int i=0 ; i < vector.size(); i++){...}`	`for ( Typ variable : vector ) { ... }`

`for ( int i=0 ; i < vs.size(); i++) {`
`cout << i << vs.at(i) << endl;`
`}`
	`for ( string x : vs ) {`
	`cout << x << endl;`
	`}`

Copyright by A.Rimbakowsky© (www.rimbakowsky.de)

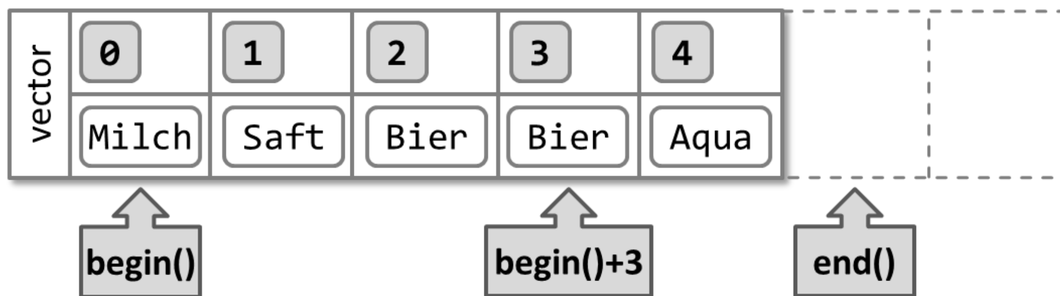


Zugriff mittels Iterator:

Mit einem Iterator (verallgemeinerter Zeiger) kann auf eine Datenstruktur in einem Datenverbund zugegriffen werden, ohne dies über einen Index der Datenstruktur zu tun.
Konzeptionell ist es wichtig zu wissen, dass Iteratoren Positionen und keine Elemente sind.
Sie werden zum Einfügen und Löschen von Elementen verwendet.

	Definition und Initialisierung
❶	`//vector <Typ> :: iterator Bezeichner ;`
①	`vector <string> vs ;`
②	`vector <string> :: iterator it;`
③	`it = vs.begin();`

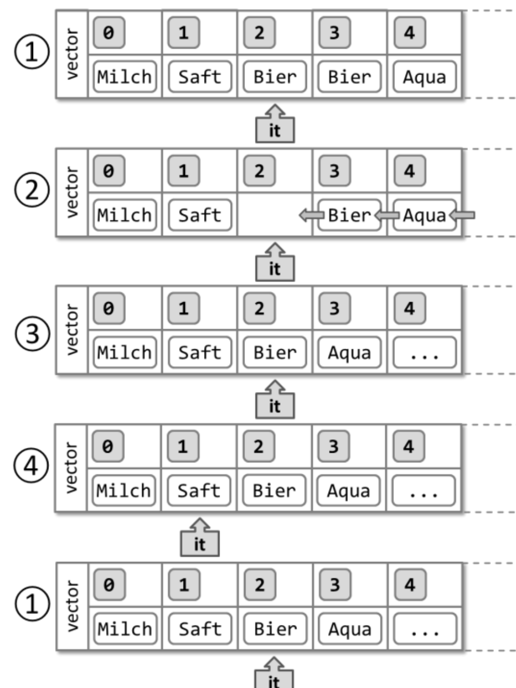
Hinweise:
- Mit ② wird ein Iterator-Objekt erzeugt, welches mit ③ auf den Anfang von `vector vs` gesetzt wird.
- Mit Iteratoren kann man (wie mit Zeigern) rechnen. Beispielsweise `it=vs.begin()+3;` positioniert den Iterator auf das 3. Element



Syntax	Beschreibung
begin()	Liefert einen Iterator vom ersten Element.
end()	Liefert einen Iterator hinter dem letzten Element.
erase(i)	Entfernt das Element an Position **i.**
insert(i, val)	Fügt ein neues Element mit dem Wert **val** vor dem Element an Position i ein.

Beispiel : Löschen aller Elemente "Bier" aus vector vs

```
vector <string> :: iterator it ;

// Iterator wird  vom Vector-Anfang bis Ende bewegt
for(it = vs.begin(); it != vs.end(); it++) {

    // String-Vergleich findet "Bier" an aktueller Iterator Position
    if ( ! (*it).compare("Bier") ) {

        // akt. Element wird entfernt (Elemente „rücken nach")
        vs.erase( it ) ;

        // Iterator wird erniedrigt
        it-- ;
    }

// nächster Schleifendurchlauf -> Iterator wird um eins erhöht
}
```



Copyright by A.Rimbakowsky© (www.rimbakowsky.de)