Datenstrukturen und Container

Als Datenstruktur bezeichnet man eine bestimmte Art und Weise, wie zusammenhängende Daten gespeichert werden.

Bis jetzt haben wir nur Stacks verwendet. In der *Standard Template Library* (*STL*), die zur C++-Standardbibliothek gehört, gibt es viele andere Möglichkeiten, Objekte im Speicher zu organisieren. Man nennt diese vorgefertigten Datenstrukturen Container.

Die Klasse Vector

> Vorläufige Definition: Eine **Klasse** ist ein Datentyp, der seine eigenen Funktionen mitbringt. Diese Funktionen nennt man **Methoden**. Variablen, die eine Klasse als Datentyp haben, heißen **Objekte**.

Ein Vector ist ein Speicher mit dynamischer Größe. Die einzelnen Elemente sind nummeriert und könnendirekt angesprungen werden. Das folgende Beispiel zeigt den grundlegenden Umgang mit einem Vector.

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
          vector<int> v; // einen leeren Vector für int-Werte anlegen

          int i = 42;

          v.push_back(1); // den Wert 1 am Ende hinzufügen
          v.push_back(13);
          v.push_back(i); // eine Kopie von i am Ende hinzufügen

          // den Inhalt mit einer herkömmlichen for-Schleife ausgeben
          for (size_t i=0; i<v.size(); i++)
          {
                    cout << v.at(i) << endl;                    // Ausgabe 1 13 42
          }

          v.pop_back(); // das letzte Element entfernen

          for (size_t i=0; i<v.size(); i++)
                    cout << v.at(i) << endl;                    // Ausgabe 1 13
}
```

`size_t` ist ein spezieller Datentyp für Größenangaben. `int` würde hier zu einer Warnung des Compilers führen. Die Methode `size()` liefert die Zahl der Elemente.

Die for-each-Schleife

Die bereichsbasierte Schleife, die in vielen anderen Sprachen for-each-Schleife heißt, durchläuft alle Werte eines Vectors, ohne dass Sie einen Zähler dafür anlegen müssen.

Syntax:

```
for ( <Typ> <Variable> : <Vector>)
{
        ...
}
```

- *Typ* gibt an, welcher Datentyp im Vector gespeichert ist. Schreiben Sie am besten `auto`, damit der Compiler den Typ ermitteln.
- Über die *Variable* kann man nacheinander auf die Objekte zugreifen.
- *Vector* ist der Name des Vectors, der durchlaufen werden soll.

Beispiel:

```
vector<char> v;

v.push_back('A');
v.push_back('B');
v.push_back('C');

for (auto zeichen : v)              // oder for (char zeichen : v)
{
        cout << zeichen << endl; // Gibt A B C aus.
}
```

`zeichen` ist jeweils eine Kopie eines Container-Elements. Wenn Sie die Daten im Container bearbeiten möchten, müssen Sie stattdessen eine Referenz verwenden:

```
vector<char> v;

v.push_back('A');
v.push_back('B');
v.push_back('C');

for (auto& zeichen : v)
{
        zeichen++;
}

for (auto i : v)
{
        cout << i << endl; // Gibt B C D aus.
}
```

Weiterführende Informationen

Dieses Skript enthält nur die nötigsten Informationen über die Klasse Stack. Alles weitere finden Sie unter anderem auf www.cplusplus.com und www.cppreference.com. Lassen Sie sich nicht davon abschrecken, dass wir noch nicht alles verstehen, was dort steht!

Wenn Sie zum Beispiel auf cplusplus.com nach `vector` suchen, erhalten Sie zunächst eine grundlegende Beschreibung der Klasse (hier gekürzt) und die benötigte Headerdatei (rechts oben).

class template

std::vector

<vector>

```
template < class T, class Alloc = allocator<T> > class vector; // generic template
```

Vector

Vectors are sequence containers representing arrays that can change in size.

Just like arrays, vectors use contiguous storage locations for their elements, which means that their elements can also be accessed using offsets on regular pointers to its elements, and just as efficiently as in arrays. But unlike arrays, their size can change dynamically, with their storage being handled automatically by the container.

Weiter unten stehen die Member-Funktionen, also die Methoden.

fx Member functions

(constructor)	Construct vector (public member function)
(destructor)	Vector destructor (public member function)
operator=	Assign content (public member function)

Iterators:

begin	Return iterator to beginning (public member function)
end	Return iterator to end (public member function)
rbegin	Return reverse iterator to reverse beginning (public member function)
rend	Return reverse iterator to reverse end (public member function)
cbegin C++11	Return const_iterator to beginning (public member function)
cend C++11	Return const_iterator to end (public member function)
crbegin C++11	Return const_reverse_iterator to reverse beginning (public member function)
crend C++11	Return const_reverse_iterator to reverse end (public member function)

Capacity:

size	Return size (public member function)
max_size	Return maximum size (public member function)
resize	Change size (public member function)
capacity	Return size of allocated storage capacity (public member function)
empty	Test whether vector is empty (public member function)
reserve	Request a change in capacity (public member function)
shrink_to_fit C++11	Shrink to fit (public member function)

Element access:

operator[]	Access element (public member function)
at	Access element (public member function)
front	Access first element (public member function)
back	Access last element (public member function)
data C++11	Access data (public member function)

Modifiers:

assign	Assign vector content (public member function)
push_back	Add element at the end (public member function)
pop_back	Delete last element (public member function)
insert	Insert elements (public member function)

Ein Mausklick auf eine Methode führt Sie zu einer genaueren Beschreibung mit Prototyp und Beispiel.

Auf dem Vorlagenlaufwerk liegt eine Offline-Version von cppreference.com.