STL-Container

Die **STL(= Standard Template Library)** hat sich seit C++98 als zentraler Bestandteil von C++ etabliert. Mit jeder C++ Evolution wird ihr Inhalt ausgebaut und optimiert. Sie definiert die sogenannten Container-Template-Klassen, die beliebige Datentypen in nahezu beliebigen Speicherformen aufnehmen können, sowie viele wichtige Algorithmen für diese Container. Der Typ der gespeicherten Objekte steht dabei in Spitzklammern, es handelt sich demnach um Templates (genaueres dazu kommt später). Wir verwenden in dieser Lehrveranstaltung nur die folgenden 2 Containertypen:

Das `std::array`-Template

Nach dem Inkludieren der Headerdatei <`array`> steht dem C++-Programmierer das Klassentemplate `std::array<T, Dim>` zur Verfügung , welches ähnlich wie ein C-Array von `T` mit der Dimension `Dim` verwendet werden kann. Der Grundtyp `T` und die konstante Dimension `Dim` **muss in den Spitzklammern** angegeben werden oder kann seit C++17 aus einer Initialisierung abgeleitet werden

```
std::array<double, 100> x;     // erzeugt array mit 100 double-Werten
std::array<double, 3> x{ 1., 2., 3.};
                               // erzeugt x mit den 3 Werten 1.0, 2.0, 3.0
std::array x{ 1., 2., 3.};     // seit C++17 erlaubt, äquivalent zu oben


x[1] = 2.0;                    // setzt 2. Element auf 2.0 (gleich wie bei C-Array)
std::cout << "x hat " << std::size(x) << " Elemente\n";
```

Dieser Container wird in C++ anstelle eines C-Arrays eingesetzt und ersetzt dieses vollwertig. Er zerfällt als Funktionsargument nicht zu einem Pointer (wie ein C-Array), sondern sein Typ und seine Dimension bleibt erhalten. Es kann im Gegensatz zum `std::vector` nicht dynamisch wachsen und sollte auch nicht zu groß definiert werden, da die Daten am **Stack** gespeichert werden, der Compiler-abhängig nicht allzu groß ist.

Das `std::vector`-Template

Nach dem Inkludieren der Headerdatei <`vector`> steht dem C++-Programmierer das Klassentemplate `std::vector<T>` zur Verfügung , welches ähnlich wie ein T-Array verwendet werden kann. Der Grundtyp `T` **muss in Spitzklammern** angegeben werden oder kann seit C++17 anhand einer Initialisierung automatisch erkannt werden:

```
std::vector<int> x;                 // erzeugt leeres x
std::vector<int> x(100);            // erzeugt x mit 100 Werten 0
std::vector<int> x(100, 1);         // erzeugt x mit 100 Werten 1
std::vector x(100, 1);              // seit C++17, erzeugt x mit 100 Werten 1
```

Achtung auf die geschwungenen Klammern in folgenden Initialisierungen:

```
std::vector<int> x{100, 1};   // erzeugt x mit Werten 100 und 1
std::vector x{100, 1};        // seit C++17: erzeugt x mit Werten 100 und 1

std::vector<double> x{1., 2., 3.};// erzeugt x mit den 3 Werten 1., 2., 3.
std::vector x{1., 2., 3.};          // seit C++17, wie oben

x[1] = 2.0;           // setzt Element mit Index 1 auf  2.0 (gleich wie bei Array)
std::cout << "x hat " << std::size(x) << " Elemente\n";
x.push_back(20.);     // hänge hinten 20.0 an x an
```

Auch dieser Container kann statt eines C-Arrays verwendet werden. Allerdings kann der Container nachträglich vergrößert werden. Außerdem speichert er seine Daten am **Heap**, der „beliebig“ (bis der Computerspeicher ausgeht) groß werden kann.

Die STL verfügt noch über weitere Containertypen, die in eigenen Includedateien definiert sind und wird laufend um weitere sinnvolle Typen ergänzt, z.B.:

für Bäume: `#include <set>` (oder `#include <unordered_set>`)
für Hash-Tabellen: `#include <map>` (entspricht dem Python Dictionary)