

# Programmiertechnik (C++)

Prof. Dr. Stefan Enderle

NTA Isny

# 4. STL

# Einführung

- STL ist eine Bibliothek, die zum C++ Standard gehört.
- Vorteile:
  - Lösungen für Standard-Probleme
  - flexibel
  - standardisiert (somit portabel)
  - gut getestet
  - effizient (mit Angaben für Algorithmen)

# Inhalt

- Die STL beinhaltet:
  - **Container** zur Speicherung von Objekten gleichen Types  
z.B. Vektoren, Listen, Bäume, Assoziative Arrays
  - **Iteratoren**, Verallgemeinerungen von Pointern zum einheitlichen Zugriff auf Container-elemente
  - **Algorithmen**  
z.B. Suchen, Sortieren
  - **Utilities** für kleine Aufgaben



# Container

- Container-Klassen sind meist austauschbar, da sich große Teile der Schnittstellen überschneiden.
- Die Effizienz von Operation unterscheidet sich jedoch:
  - Einfügen in einen `vector` ist teuer, in eine `list` billig
  - In einer `list` kann man jedoch nur zum Nachbarelement springen, in einem `vector` überall hin.

# Container

- Beispiele:

- Vektor von Integers:

```
vector<int> v(10);
```

- Liste von Integers:

```
list<int> v(10);
```

- Vektor von Strings:

```
vector<string> v(10);
```

# Container

- Operationen:

- `size()` Anzahl gespeicherter Objekte
- `empty()` Container leer?
- `begin()` Iterator auf erstes Element
- `end()` Iterator auf Ende
- `push_front(x)` `x` am Anfang einfügen
- `push_back(x)` `x` am Ende einfügen
- `pop_front(x)` Erstes Element entfernen
- `pop_back(x)` Letztes Element entfernen

# Map

- Der STL Container „Map“ ist ein assoziatives Array, das einen beliebigen Datentyp auf einen anderen beliebigen Datentyp abbilden („mappen“) kann.

- Include-File:

```
#include <map>
```

# Map - Deklaration

- Deklaration eines Map-Objektes:

```
map<Typ1, Typ2> meineMap;
```

- Beispiele:

```
map<int, string> meineMap;
```

```
map<string, double> meineMap;
```

```
map<myClass1, myClass2> meineMap;
```

# Map - Einspeichern

- Einspeichern:

Um in eine Map Paare von Objekten einzuspeichern bietet sich der Map-Operator[] an.

- Beispiel:

```
map<string, string> stringMap;  
stringMap["Maier"]    = "Uwe";  
stringMap["Enderle"] = "Stefan";  
stringMap["Maier"]    = "Bernd"; // Uwe weg!
```

# Map – Operator[ ]

- Vorsicht: Der operator[ ] besitzt eine eigentümliche Semantik:
  - Suche den angegebenen Key
  - Falls gefunden, liefere die Referenz auf den Value zurück
  - Falls nicht gefunden, erzeuge ein neues Key-Value-Objekt und liefere die Referenz hierauf zurück.

- Problem:

```
string vorname = stringMap["Maier"];
```

Falls es bisher keinen Maier gab, so gibt es ihn jetzt!!

# Map – find()

- Zum Suchen eines Eintrags sollte besser die Methode find() verwendet werden.
- Die Semantik von find() geht folgendermaßen:
  - Suche den angegebenen Key
  - Falls gefunden, liefere einen Iterator auf das Key-Value-Paar
  - Falls nicht gefunden, liefere map::end() zurück

- Beispiel:

```
map<string, string>::iterator i;  
i = stringMap.find("Maier");  
if (i==stringMap.end()) cout << „not found“;  
else cout << „Name:“ << i->second
```

# Map – Iterator

- Außer zum Suchen wird der `map::iterator` z.B. zur sequentiellen Ausgabe aller Elemente benötigt.
- Dies geschieht analog zu den anderen STL-Containern (`vector`, `list`, etc.), außer dass der Iterator auf ein Key-Value-Pair zeigt.
- Beispiel: Ausgeben aller Map-Einträge:

```
map<string,string> stringMap;

... // map füllen

map<string,string>::iterator i;
for (i = stringMap.begin(); i!=stringMap.end(); i++)
    cout << i->first << " " << i->second << endl;
```

# Key-Value-Pairs

- Innerhalb einer Map werden Key-Value-Pairs vom STL Typ `pair` verwendet.
- Die Klasse `pair` ist in der Header-Datei `<utility>` definiert.
- Ein `pair` besitzt die beiden öffentlichen (public) Elemente `first` und `second`.
- Beispiel: Anlegen eines Pairs:

```
pair<string,double> p;  
p = make_pair(„Produkt“, 75.99);  
  
cout << p.first << " " << p.second << endl;
```

# Map Implementation

- Der C++ Standard schreibt keine spezielle Implementierung von Map vor.
- Die meisten Implementierungen besitzen jedoch einen balancierten Binärbaum, d.h. Zugriffe werden mit  $O(\log n)$  realisiert.
- Somit sind Maps (auch) für sehr große Datenmengen effizient einsetzbar.
- Aufgrund dieser Implementierung existieren jedoch eine Reihe von Anforderungen an die Klassen, die in der Map gespeichert werden sollen.

# Map - Anforderungen

- Anforderungen an die Objekte, die in der Map gespeichert werden sollen:
  - Ein Zuweisungsoperator (assignment operator) muss existieren
  - Dieser muss „unabhängig“ sein:  
a=b; und dann Änderung von b darf a nicht beeinflussen
  - Der Key-Typ muss eine strikte Ordnung besitzen:
    - operator< muss existieren
    - a<a muss false sein
    - Gleichheit muss nur durch < bestimmt werden können, also  $a==b \iff !(a<b) \ \&\& \ !(b<a)$
    - Wenn  $a<b$  und  $b<c$ , dann muss  $a<c$  gelten
  - Der Value-Typ muss einen Default-Konstruktor besitzen