# Spracherweiterungen in C++11/14 (Auszug)

Es sollen einige häufig gebrauchte neue Sprachelemente von C++11/14 vorgestellt werden.

### Automatische Typableitung mit `auto`

Das Schlüsselwort `auto` in C++11/14 kann in Variablendefinitionen an Stelle eines Typnamens benutzt werden und zeigt an, dass der Datentyp der Variable aus ihrer Initialisierung abgeleitet werden soll. Die Ableitung des Typnamens entspricht der Ableitung von Templateparametern in Funktionstemplates aus den eingesetzten Argumenten.

Die sehr seltene Verwendung von `auto` als Speicherklassenangabe wie in C und C++98 ist nicht mehr erlaubt.

*Bsp.:*

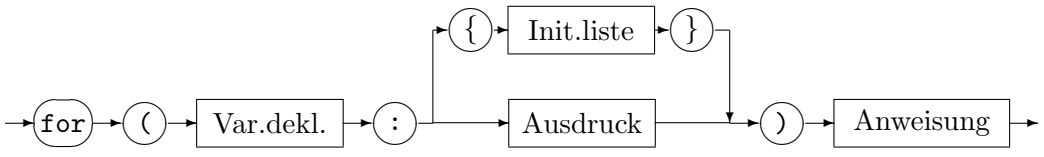
```
int main()
{
  auto i=1;            // i: int
  const auto k=5;      // k: const inst
  auto x=1.0;          // x: double
  auto& j=i;           // j: Referenz auf int
  auto& l=k;           // l: Referenz auf const int
  auto& m=2;           // unzulaessig (m Referenz, nicht konstante Referenz)
  const auto& n=2;     // n: Referenz auf const int
  auto p=2,y=2.0;      // unzulaessig (kein einheitlicher Typname ableitbar)
}
```

`auto` kann auch in Variablendefinitionen in den Köpfen von Wiederholungs- und Bedingungsanweisungen verwendet werden, am häufigsten kommt es in Initialisierungsteil der üblichen und der noch zu behandelnden bereichsbasierten `for`-Anweisung zum Einsatz.

*Bsp.:*

```
double Norm(const vector<double>& x)
{
  double s=0;
  for (auto cpos=x.begin(); cpos!=x.end(); ++cpos) s += (*cpos)*(*cpos);
  return sqrt(s);
}
```

### Bereichsbasierte for-Anweisung




Diese Anweisung ist im wesentlichen äquivalent zu

```
auto&& range = Ausdruck;  // spaeter!
for (auto pos=begin(range),ende=end(range); pos!=ende; ++pos) {
  Var.dekl. = *pos;
  Anweisung
}
```

wobei im Fall der Behälterklassen die Umsetzungen `begin(`$a$`)`$\rightarrow$ $a$`.begin()` und `end(`$a$`)`$\rightarrow$ $a$`.end()` neu in C++11/14 eingeführt wurden. Im Fall eines C-Vektors $a$ mit $N$ Komponenten, ist `begin(`$a$`)` als $a$ und `end(`$a$`)` als $a + N$ festgelegt.

*Bsp.: C-Vektoren ändern und ausgeben*

```
#include <iostream>

using namespace std;

int main()
{
  double a[] = {1.0,2.0,3.0,4.0,5.0};

  for (auto x : a)
    cout << x << " ";
  cout << endl;

  for (auto& y : a)
    y *= 2;

  for (auto z : a)
    cout << z << " ";

  cout << endl;
  return 0;
}
```

*Ausgabe:*

```
1 2 3 4 5
2 4 6 8 10
```

Auch Einlesen in Behälter ist mit der bereichsbasierten for-Anweisung möglich, der Behälter muss mit der richtigen Größe bereits angelegt sein.

*Bsp.: Einlesen auf einen STL-Vektor*

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

using namespace std;

int main()
{
  int n;
  cout << "n: "; cin >> n;
  vector<int> a(n);

  cout << "a: ";
  for (auto& x: a)
    cin >> x;

  for (auto x: a)
    cout << x << " ";

  cout << endl;
  return 0;
}
```

Enthält ein Behälter weitere Behälter, so sind auch Doppelschleifen möglich.

*Bsp.: Matrix als* `vector<vector<double>>` *(nicht empfehlenswert!)*

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

using namespace std;

int main()
{
  int m,n;
  cout << "m n: "; cin >> m >> n;
  vector<vector<double>> a(m,vector<double>(n));

  for (auto& v: a) {
    cout << "Matrixzeile: ";
    for (auto& x: v) cin >> x;
  }

  cout << endl;

  for (auto v: a) {
    for (auto x: v) cout << x << " ";
    cout << endl;
  }

  return 0;
}
```

Bei Maps ist wie bisher zu berücksichtigen, dass Index/Werte-Paare gespeichert werden.

*Bsp.: Punktetabelle*

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

using namespace std;

int main()
{
  map<string,int> a;
  string s; int n, gesamt=0;

  while(cin >> s >> n) { a[s] += n; gesamt += n; }

  for (auto x: a) {
    cout << left  << setw(10  << x.first
         << right << setw(8)  << x.second << endl;
  }

  cout << endl << "gesamt:   " << right << setw(8) << gesamt << endl;
  return 0;
}
```

**Initialisierungslisten**

Bei Initialisierungen von Klassen und Behältern können jetzt wie bei C-Vektoren und C-Records (struct) durch Komma getrennte und in Mengenklammern eingeschlossene Listen von Werten angegeben werden. Bei Klassen handelt es sich im wesentlichen um zusätzliche Schreibweisen für Konstruktorargumente. Bei Behältern sind zusätzliche Konstruktoren für das neugeschaffene Klassentemplate `initializer_list<T>` erforderlich, die für die STL-Behälter mit variabler Länge vordefiniert sind. Sie haben bei Initialisierungen Vorrang vor den anderen Konstruktoren.

*Bsp.: Initialisierungen für komplexe Zahlen*

```
complex<double> z1(3.0,4.0);   // bisher moeglich
complex<double> z2={3.0,4.0};  // neu
complex<double> z3{3.0,4.0};   // neu (vermeidet gefaehrliche Typumw.)
```

*Bsp.: Initialisierung von STL-Vektoren*

```
vector<double> a1={0.5,1.0};  // neu
vector<double> a2{0.5,1.0};   // neu

vector<int> b1={0.5,1.0};     // neu (gefaehrl. Typumw. double->int)
vector<int> b2{0.5,1.0};      // verboten  (gefaehrl. Typumw. double->int)
```

Verschachtelte Initialisierungslisten sind ebenfalls möglich.

*Bsp.: Komplexer STL-Vektor*

```
vector<complex<double>> u = {{0.0,1.0},{0.0,-1.0}};

complex<double> i = {0.0,1.0};
vector<complex<double>> v = {i,-i};

vector<complex<double>> wa = {1.0,-1.0};    // Laenge 2
vector<complex<double>> wb = {{1.0,-1.0}};  // Laenge 1
}
```

Bei einer Map `map`$<I,T>$ erfolgt die Initialisierung über Wert/Index-Paare. Letztere können jeweils durch $\{i,t\}$ initialisiert werden, weil das Klassen-Template `pair` entsprechende Konstruktoren enthält.

*Bsp.: Initialisierung einer Map*

```
map<string,int> a = {{"Moser",2},{"Meyer",8},{"Mueller",3}};
```

Das Klassentemplate `initializer_list<T>` stellt Iteratoren (vom Typ `const T *`) zum Durchwandern der Liste bereit und definiert auch die Funktionen `begin` und `end`. Damit ist die Verwendung in der bereichsbasierten for-Anweisung möglich.

*Bsp.:*

```
#include <initializer_list>
          :
for (auto i : {2,3,5,7,11,13,17}) cout << i << " ";
```

Neben vielen Komponentenfunktionen für Behälter akzeptieren auch manche STL-Algorithmen Initialisierungslisten als Argumente.

*Bsp.: STL-Funktion max aus* `<algorithm>`

```
double a,b,c;
cout << "max(a,b,c)=" << max({a,b,c}) << endl;
```