Zeiger mit impliziter Speicherfreigabe (`unique_ptr`, `<memory>`)

Objekte der Klasse `unique_ptr<T>`, für T entweder ein beliebiger Typ oder ein Array-Typ der Form T'`[]` (in diesem Fall ist das relevante template spezialisiert), enthalten intern einen Zeiger. Der Destruktor der Klasse `unique_ptr<T>` ruft den `delete`-Operator auf den enthaltenen Zeiger auf. Der Konstruktor der Klasse `unique_ptr<T>` akzeptiert einen Parameter vom Typ Zeiger auf T bzw. T falls T ein Array-Typ, also der Form T'`[]`, ist. Der Konstruktor ist als `explicit` markiert, nimmt also nicht Teil an der impliziten Typkonvertierung.

Beispiel. Das Beispielprogramm weist mit dem `new`-Operator Speicher für einen Vektor von Werten vom Typ `int` der Länge 1 000 auf dem heap zu. Der erzeugte Zeiger wird an den Konstruktor von `unique_ptr<int[]>` übergeben. Das für C-Arrays spezialisierte Template von `unique_ptr` erlaubt den Zugriff auf Array-Element mit dem subscript-Operator. Sobald die Variable `v` ihren syntaktischen Gültigkeitsbereich verlässt (am Ende der `main`-Funktion) wird automatisch der Destrkutor auf das in `v` enthaltene Objekt aufgerufen. Bei Ausführung des Destruktors wird der `delete`-Operator auf den enthaltenen Zeiger angewandt um den mit `new` zugewiesenen Speicher wieder freizugeben. Das Programm hat kein Speicherleck.

unique_ptr_demo.cpp

```cpp
#include <memory>

using namespace std;

int main() {
  unique_ptr<int[]> v{new int[1000]};

  for (int i = 0; i < 1000; i++)
    v[i] = i;
}
```

Assertions (`<cassert>`)

Für einen Ausdruck b vom Typ `bool` bricht das Präprozessor-Makro `assert((`b`))` die Ausführung des Programmes sofort mit einer Fehlermeldung ab, wenn b zu `false` auswertet. Wird an den compiler der Kommandozeilenparameter `-DNDEBUG` bei Übersetzung des Programms übergeben, werden alle Vorkommnisse von `assert(...)` durch i.W. Leerzeichen ersetzt, sodass sie im übersetzten Programm keinerlei Auswirkung haben.

Beispiel. Wir übersetzen ein einfaches Testprogramm mit einer assertion zu Beginn einmal ohne und einmal mit dem Kommandozeilenparameter `-DNDEBUG`. Bei Übersetzung mit `-DNDEBUG` hat die assertion keinerlei Auswirkung bei Ausführung des Programms.

assert.cpp

```cpp
#include <iostream>
#include <cassert>

using namespace std;
```

```cpp
int main() {
  assert((1 == 2));
  cout << "output" << endl;
}
```

```
g++ assert.cpp; ./a.out
```

```
assert: _copy/assert.cpp:7: int main(): Assertion `(1 == 2)' failed.
```

```
g++ -DNDEBUG assert.cpp; ./a.out
```

```
output
```