O'REILLY®

3. Auflage
Aktuell zu C++17

C++

kurz & gut

O'REILLYS TASCHENBIBLIOTHEK

Kyle Loudon
& Rainer Grimm

Es gibt keine einheitlichen Stilkonventionen für Bezeichner. Viele Programmierer setzen aber Namen, die mit Kleinbuchstaben beginnen, für lokale Variablen, Instanzvariablen und Funktionen ein. Namen, die mit Großbuchstaben beginnen, werden dementsprechend für Typen, Namensräume und globale Variablen verwendet. Namen, die der Präprozessor verarbeitet, und Konstanten werden ganz in Großbuchstaben geschrieben.

Reservierte Schlüsselwörter

C++ definiert eine Reihe von Schlüsselwörtern und alternativen Token. Dies sind Zeichenfolgen mit besonderer Bedeutung in der Sprache. Diese Wörter sind reserviert und können nicht als Bezeichner verwendet werden. Die reservierten Schlüsselwörter in C++ lauten:

```
alignas     alignof     and
and_eq      asm         auto
bitand      bitor       bool
break       case        catch
char        char16_t    char32_6
class       compl       const
constexpr   const_cast  continue
decltype    default     delete
do          double      dynamic_cast
else        enum        explicit
export      extern      false
final       float       for
friend      goto        if
inline      int         long
mutable     namespace   new
noexcept    not         not_eq
nullptr     operator    or
```

or_eq	override	private
protected	public	register
reinterpret_cast	return	short
signed	sizeof	static
static_assert	static_cast	struct
switch	template	this
thread_local	throw	true
try	typedef	typeid
typename	union	unsigned
using	virtual	void
volatile	wchar_t	while
xor	xor_eq	

Literale

Literale sind lexikalische Elemente, die explizite Werte in einem Programm repräsentieren. C++ definiert viele verschiedene Typen von Literalen, die alle unter ihrem jeweiligen Typ im Abschnitt »Fundamentale Typen« auf Seite 45 beschrieben werden. Darüber hinaus kennt C++ Funktionsliterale, die auch als Lambda-Funktionen bezeichnet werden.

Benutzerdefinierte Literale

Benutzerdefinierte Literale sind selbst definierte Literale. Diese werden in C++ für ganze Zahlen, Fließkommazahlen, C-Strings und auch Zeichen unterstützt. Für sie gilt die folgende Syntax: `Built-in-Literal + _ + Suffix`. Das Suffix stellt in der Regel eine Einheit dar:

```
101000101_b
63_s
10345.5_dm
123.45_km
100_m
131094_cm
33_cent
"Hallo"_i18n
```

Benutzerdefinierte Literale erlauben es, Werte direkt mit ihrer Einheit zu verknüpfen. Damit lassen sich Operanden in C++ implementieren, die bei Operationen ihre Einheit berücksichtigen: `Dist myDis= 10345.5_dm + 123.45_km – 100_m + 131094_cm`. Die C++-Laufzeitumgebung bildet die benutzerdefinierten Literale auf die entsprechenden Literaloperatoren ab (siehe Abschnitt »Operatoren überladen« auf Seite 146). Diese müssen vom Programmierer implementiert werden. So definiert in dem folgenden Beispiel der Namensraum `Unit` alle Literaloperatoren, die für das richtige Interpretieren des arithmetischen Ausdrucks `Dist myDis= 10345.5_dm + 123.45_km – 100_m + 131094_cm` benötigt werden. In Kombination mit der Klasse `Dist`, die einen Konstruktor, einen Plus- und einen Minusoperator anbietet, lässt sich der ganze Ausdruck evaluieren. `Dist` verwendet als Einheit cm:

```
class Dist{
  public:
    explicit Dist(double i):cm(i){}

    friend Dist operator +(const Dist& a, const Dist& b){
      return Dist(a.cm + b.cm);
    }
    friend Dist operator -(const Dist& a, const Dist& b){
      return Dist(a.cm - b.cm);
    }
  private:
    double cm;
  };

namespace Unit{
  Dist operator "" _km(long double d){
    return Dist(100000 * d);
  }
  Dist operator "" _m(long double m){
    return Dist(100 * m);
  }
  Dist operator "" _dm(long double d){
    return Dist(10 * d);
  }
  Dist operator "" _cm(long double c){
    return Dist(c);
  }
```

```
}

...

Dist myDis= 10345.5_dm + 123.45_km
              - 100_m + 131094_cm;  // 12589549_cm
```

C++ unterstützt benutzerdefinierte Literale für Fließkommazahlen und natürliche Zahlen in zwei Formen: in der sogenannten *raw*- und in der *cooked*-Form. Das erste Beispiel stellte die *cooked*-Form vor, das anschließende zweite Beispiel hingegen die *raw*-Form. Während bei der *cooked*-Form der Literaloperator das benutzerdefinierte Literal ohne Unterstrich und Suffix annimmt, erhält er bei der *raw*-Form das Literal als `const char*`-Typ. Im Fall des C-Strings erhält er es sogar als Paar (`const char*`, `size_t`). So führt das Literal `"10345.5"_dm` zu einem Aufruf des Literaloperators mit den Argumenten (`"10345.5"`,7). Für Zeichen und C-Strings existiert nur die *raw*-Form für benutzerdefinierte Literale. Im Zweifelsfall besitzt die *cooked*-Form die höhere Präzedenz. Unter der Voraussetzung, dass Instanzen der Klasse `Dist` durch einen `const char*` und eine natürliche Zahl initialisiert werden können, besitzen die entsprechenden Literaloperatoren in der *raw*-Form die folgende Implementierung:

```
namespace Unit{
  Dist operator "" _km(const char* k, std::size_t s){
    return Dist(k, 100000);
  }
  Dist operator "" _m(const char* m, std::size_t s){
    return Dist(m, 100);
  }
  Dist operator "" _dm(const char* d, std::size_t s){
    return Dist(d, 10);
  }
  Dist operator "" _cm(const char* c, std::size_t s){
    return Dist(c);
  }
}

...

Dist myDis= "10345.5"_dm + "123.45"_km
              - "100"_m + "131094"_cm;  // "12589549"_cm
```

Im letzten Beispiel ist schön zu sehen, dass nicht nur das Argument des Literaloperators ein C-String ist, sondern auch das benutzerdefinierte Literal.

Da die Suffixe der benutzerdefinierten Literale kurz sind und gern für Einheiten verwendet werden, sind Namenskollisionen unvermeidlich. Daher sollten sie in Namensräumen definiert und anschließend importiert werden.

Neue Built-in-Literale

C++14 enthält neue Built-in-Literale für Binärzahlen, C++-Strings, komplexe Zahlen und Zeiteinheiten.

Tabelle 2-1: Built-in-Literale

Typ	Präfix/Suffix	Beispiel
binäre Zahl	`0b`	`0b10`
`std::string`	`s`	`"Hello"s`
`complex<double>`	`i`	`5i`
`complex<long double>`	`il`	`5il`
`complex<float>`	`if`	`5if`
`std::chrono::hours`	`h`	`5h`
`std::chrono::minutes`	`min`	`5min`
`std::chrono::seconds`	`s`	`5s`
`std::chrono::milliseconds`	`ms`	`5ms`
`std::chrono::microseconds`	`us`	`5us`
`std::chrono::nanoseconds`	`ns`	`5ns`

Binärzahlen werden durch das Präfix `0b`, alle weiteren Literale durch ein Suffix definiert. Die Built-in-Literale besitzen im Gegensatz zu den benutzerdefinierten Literalen keine Unterstriche (_). Wird das C-String-Literal `"Hello"` um den Buchstaben `s` erweitert, entsteht das C++-String-Literal `"Hello"s`.