Variablendeklarationen

- in **C**: am Anfang jedes Blockes

- in **C++**: **vor** der eigentlichen Verwendung, Gültigkeitsbereich ist der aktuelle Block (**{ ... }**)

- Beispiel:
```
for(int i=0; i<256; i++){ ... }
for(auto k=0; k<=127; k++){ ... }
```

- Beispiel:
```
void main() int j=8; if(j) int i=2; else int k=3;
```

- Beispiel:
```
if(int i=0){ }; // Fehler
while(int i=0) { } //Fehler
```

- wie in **C** werden in **C++** nur **globale** Variablen und lokale **static**-Variablen mit **0** initialisiert, alle anderen Variablen besitzen **undefinierte Werte**, falls bei der Vereinbarung nicht sofort eine **Initialisierung** erfolgt (empfehlenswert), z.B. `int i = 0;`

- **Objekte** werden seitens der **Konstruktoren** initialisiert

- neu in C++ ist der Typ **bool**, z.B.
```
bool x = true; x = (3 < 1);
cout<<boolalpha<<x<<endl;
```

Elementare Datentypen in C++ (C++11)

Name	Abkürzung	Präfix (Literal)	Suffix (Literal)	Größe
bool				**true, false**
char		**u8**		**implementierungsabhängig**
wchar_t		**L**		**implementierungsabhängig**
char16_t		**u**		**mindestens 2 Byte**
char32_t		**U**		**mindestens 4 Byte**
Raw String		**R**		
short int	**short**			**≥ c h a r**
unsigned short int	**unsigned short**			**≥ c h a r**
int				**≥ s h o r t i n t**
unsigned int				**≥ s h o r t i n t**
long int	**long**		**l** oder **L**	**≥ i n t**
unsigned long int	**unsigned long**		**ul** oder **UL**	**≥ i n t**
long long int	**long long**		**ll** oder **LL**	**≥ l o n g i n t**
unsigned long long int	**unsigned long long**		**ull** oder **ULL**	**≥ l o n g i n t**
float			**f** oder **F**	**implementierungsabhängig**
double				**≥ f l o a t**
long double			**l** oder **L**	**≥ d o u b l e**

Beispiele:

```
unsigned long long w = 1234ULL;
float pi_float = 3.14f;
double pi = 3.14;
long double pi_longdouble = 3.13L;
char a = 'A';
char16_t b = u'B';
char32_t c = U'C';
wchar_t d = L'D';
```

In einem C-String-Literal müssen Zeichen wie " oder \ mit Escape-Sequenzen kodiert werden:

```
cout<<"Pfad: \"C:\\Temp\"" << endl;
```

In einem **Raw-String** (C++11) werden alle Zeichen direkt übernommen:

```
cout<<R"(Pfad: "C:\Temp")" << endl;
```

Eine Codierung ist nicht notwendig. Sollten die Begrenzungszeichen (oder) selbst innerhalb eines **Raw-Strings** benötigt werden, dann können diese durch einen maximal 16 Zeichen langen anderen Begrenzungsstring ersetzt werden, z.B.:

```
cout<<R"LIMESPfad: "C:\Temp"LIMES" << endl;
```

Als **Zeigerliteral** gibt es ab C++11 **nullptr**:

```
char *s = nullptr;      // ab C++ 11 zusätzlich
char *t = 0;            // OK
char *u = (char *)0;    // OK
char *v = NULL;         // OK, #define NULL 0 vom Compiler gesetzt
char *w = (void *)0;    // Compile Error, da Zeigertypen ungleich
```

Schlüsselwort auto ab C++11

Datentyp des return-Wertes auch hinter Funktionskopf definierbar:

```
auto max(int x, int y) -> int { return x>y ? x : y; }

int main() { // ...
             cout<<"max(3,5) = "<<max(3,5)<<endl;
             // ...
}
```

Damit ist der **return-Typ** in Abhängigkeit von den **Typen der Funktionsparameter** definierbar.
(wird in **Funktions-Templates** genutzt)

Variablendeklarationen

Datentyp von Variablen oder Objekten ist auch **automatisch** anhand des initialen Wertes bestimmbar und wird mit **auto** ausgedrückt:

```
auto a = 0;          // int
auto b = 5.0;        // double
auto c = 3.14f;      // float
auto d = max(5,7); // int
int x = 5, y(3);
auto z = x/y;        // int
auto b;              // keine initialer Wert, Compile-Error
decltype(x/y) b;     // int, da x/y vom Typ int
```

Mit **auto** definierte Objekte bzw. Variablen müssen bei ihrer Definition **initialisiert** werden.

decltype ermittelt den **Datentyp** anhand eines Ausdruckes in den Klammern, ohne der Variablen etwas zuzuweisen.

typename weist den Compiler an, dass ein unbekannter **identifier** ein **Typ** ist:

```
template<typename Typ1, typename Typ2>
auto quotient(Typ1 a, Typ2 b) -> decltype(a/b) {
     return a/b;
}

// Aufruf von quotient in main:
cout<<"5/3       = "<<quotient(5, 3)<<endl;       // 1
cout<<"5.0/3.0   = "<<quotient(5.0, 3.0)<<endl;   // 1.66667
```