# 2. Datentypen und Deklarationen

## Programm = Datenstrukturen+Kontrollstruktur

- Programme verarbeiten Daten.
- Daten werden in C durch Datenstrukturen aus verschiedenen Datentypen beschrieben.

Es gibt (wie in allen Programmiersprachen) elementare Datentypen, z. B. zur Beschreibung von Zahlen oder einzelnen Zeichen, und es gibt komplexe Datentypen, z. B. zur Beschreibung von verketteten Listen oder von Datensätzen, die aus verschiedenen Komponenten bestehen.

Datentypen grenzen zulässige Wertebereiche für die mit ihnen deklarierten Datenstrukturen ab.

# Deklaration von Daten

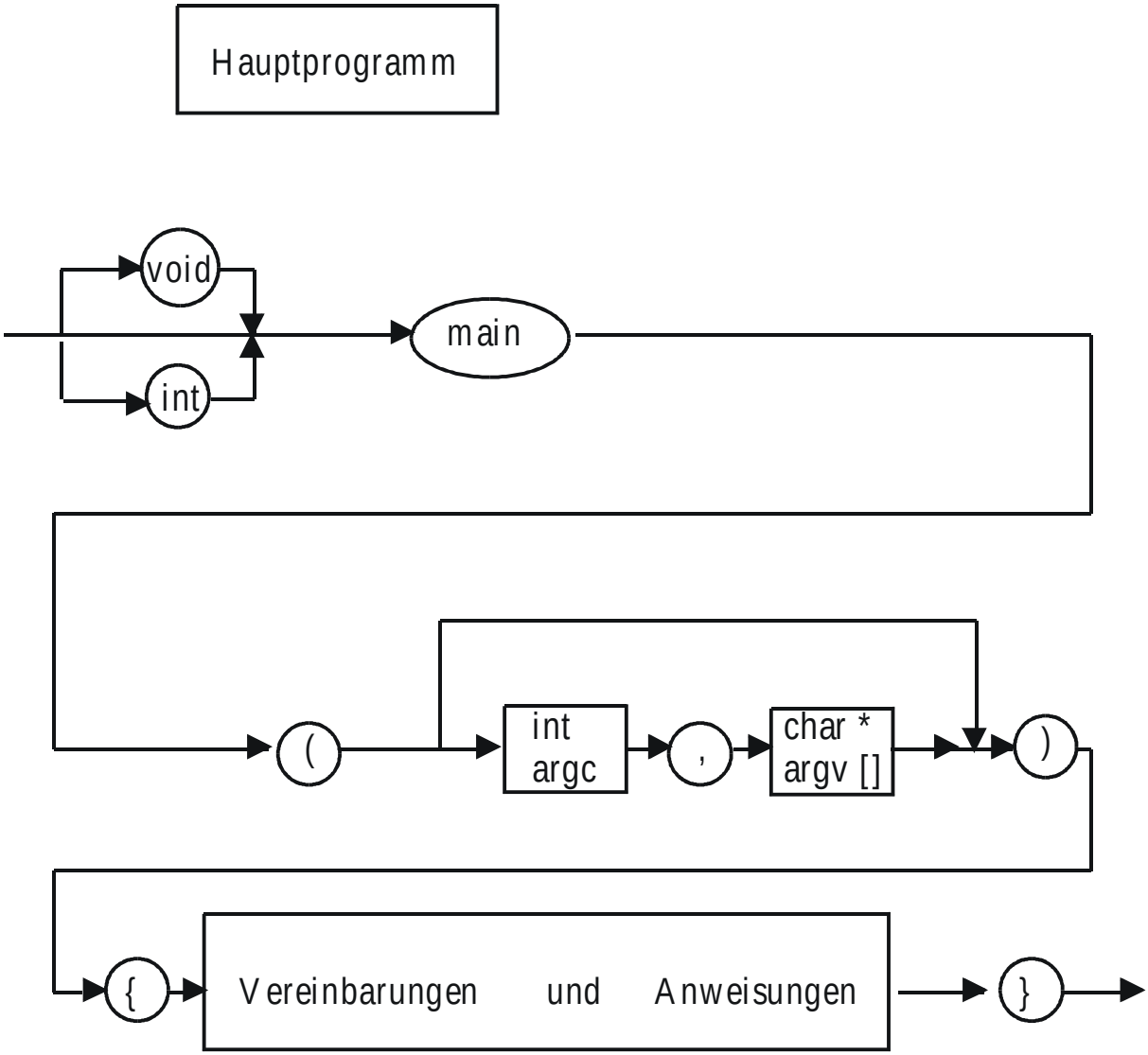
In C müssen alle Daten vor ihrer Verwendung vereinbart (deklariert) werden (anders als z. B. in Fortran oder Basic).

Ein C-Programm setzt sich aus Vereinbarungen und Anweisungen zusammen.

Die Vereinbarungen beschreiben die Datenstrukturen, auf denen ein Programm arbeitet.

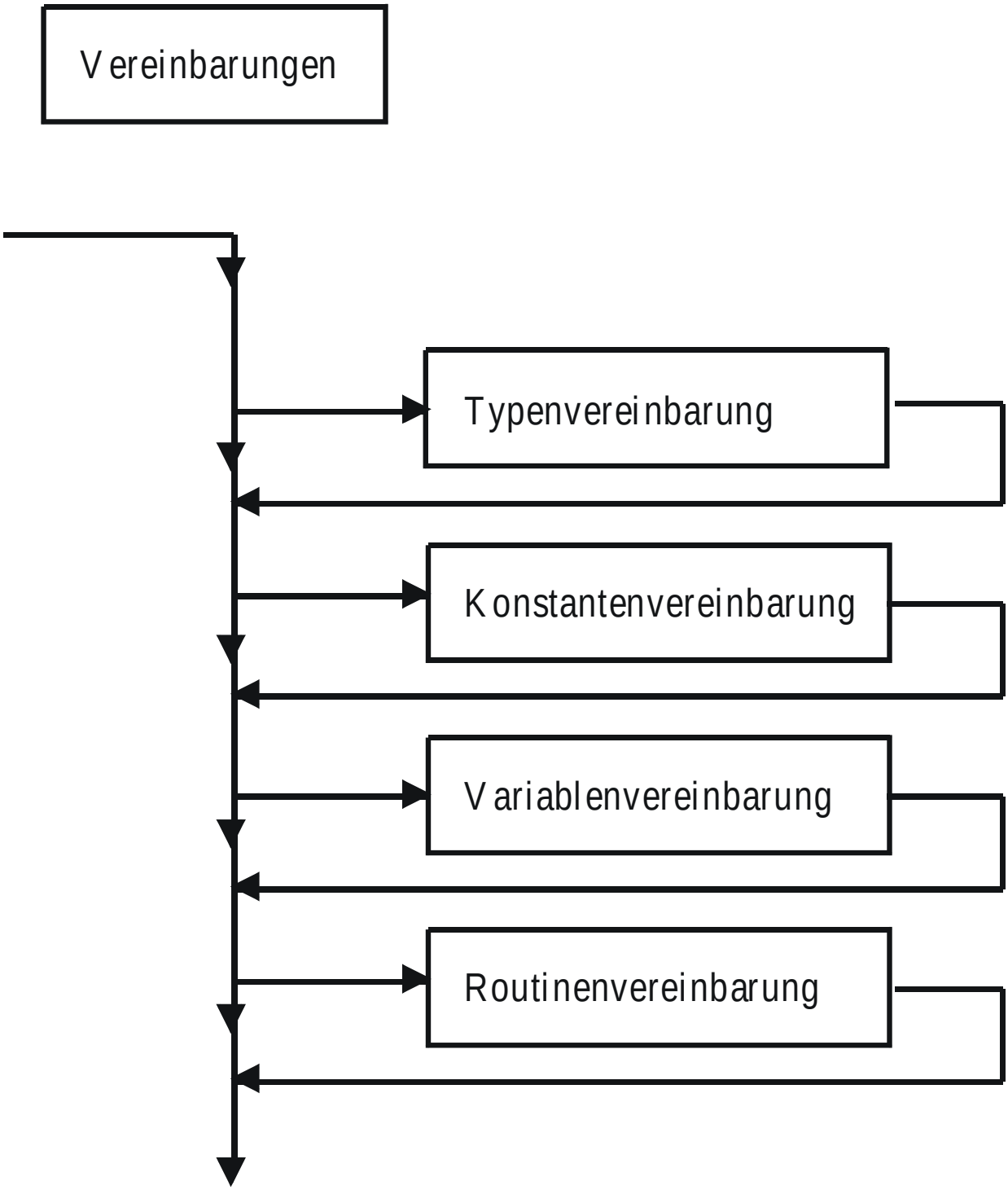
Die Anweisungen definieren die Kontrollstruktur des Programms.

# Aufbau eines C-Programms




argc und argv stellen die Verbindung des Programms zu seiner Umwelt her. Sie sind zwar im ANSI-Standard nicht beschrieben, haben sich aber als defacto-Standard etabliert. Sie werden später erklärt.

# Vereinbarungen in C-Programmen (1)



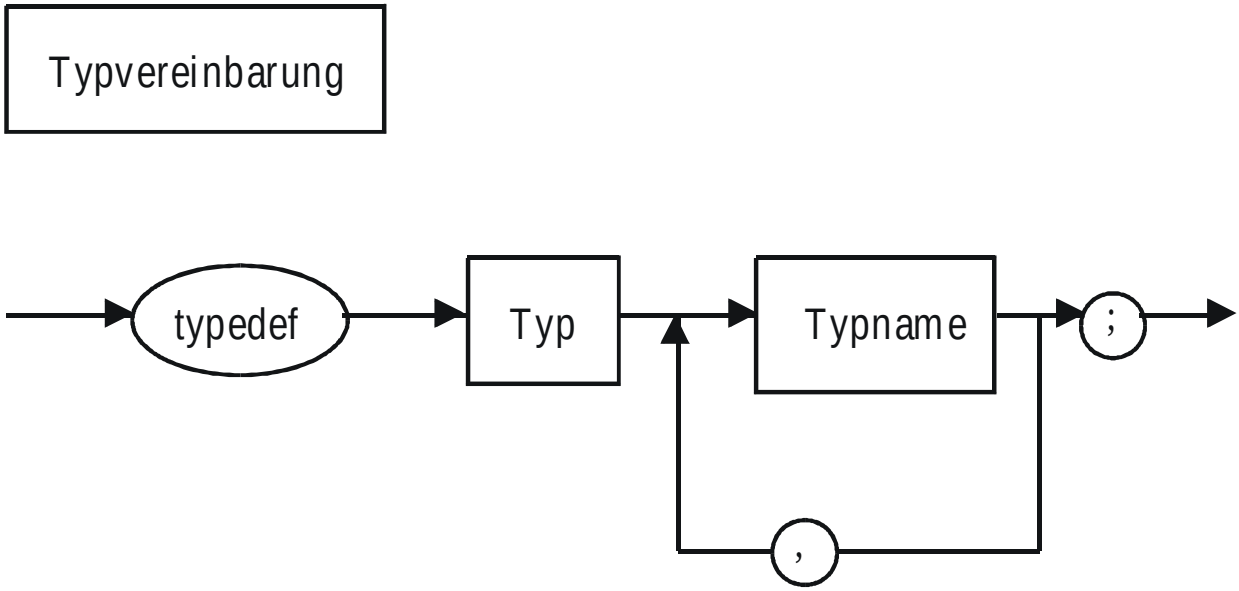
# Vereinbarungen in C-Programmen (2)

Die im Syntaxdiagramm angegebene Reihenfolge der Vereinbarungen ist nicht zwingend vorgeschrieben. Es ist jedoch sinnvoll, sich an diese Reihenfolge zu halten.

In C gibt es zwei Arten von Vereinbarungen:

- Vereinbarungen, die Speicherplatz reservieren (**Definitionen**), z. B. Variablenvereinbarungen, Routinendefinitionen
- Vereinbarungen, die eine Variable oder einen Datentyp oder eine Routine beschreiben (**Deklarationen**), z. B. Typenvereinbarungen, Prototypen von Routinen

# Typvereinbarung

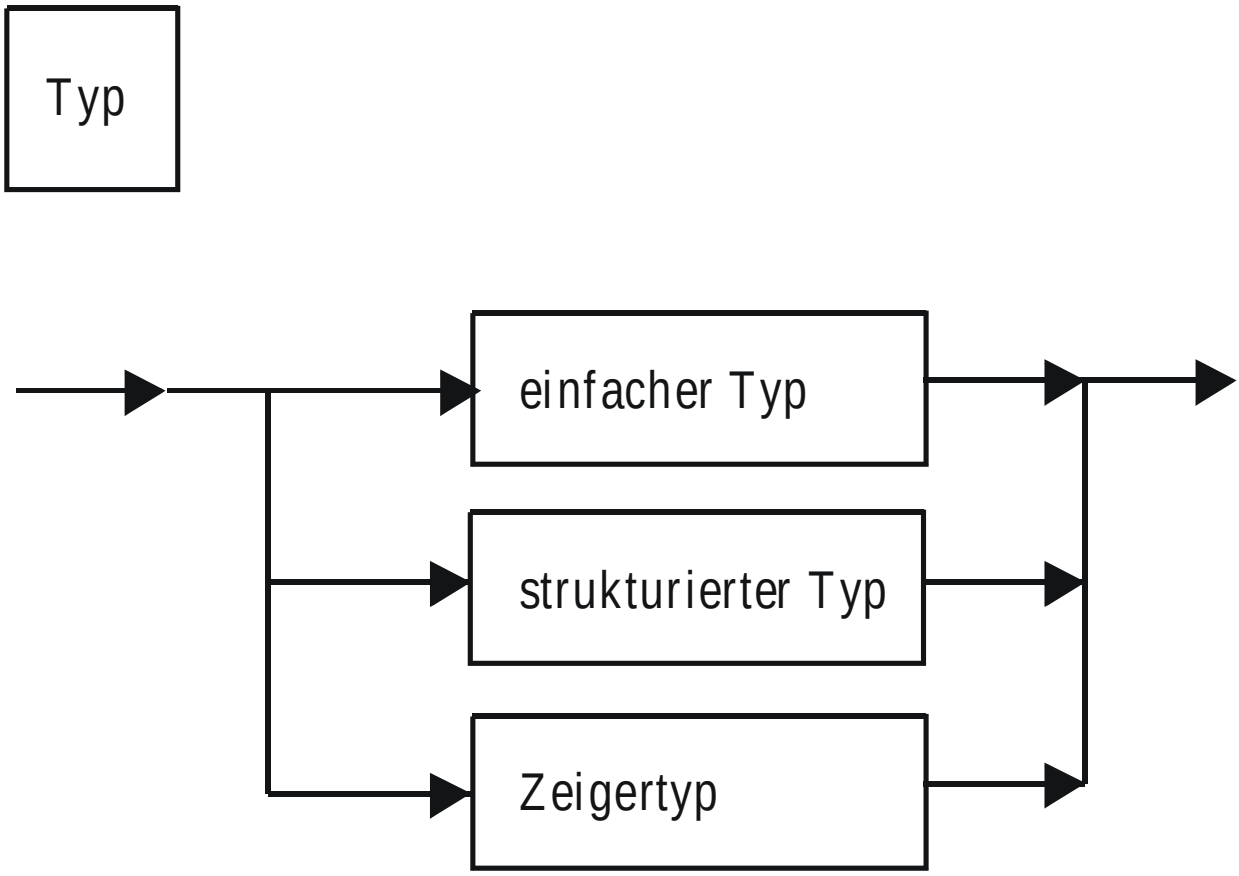



typedef dient zur Definition **eigener** Datentypen als Ableitung aus bestehenden Datentypen.

**Beispiel**

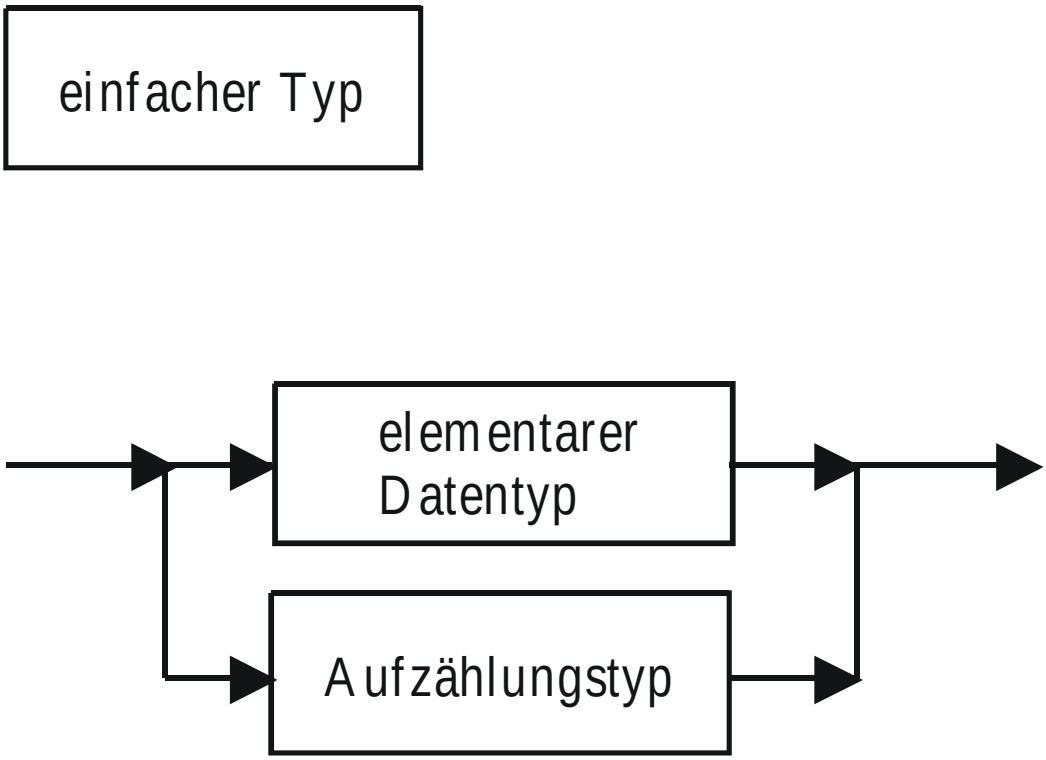
```
typedef int alter;
alter lebensalter, firmenbestandszeit;
```

# Vordefinierte Datentypen in C

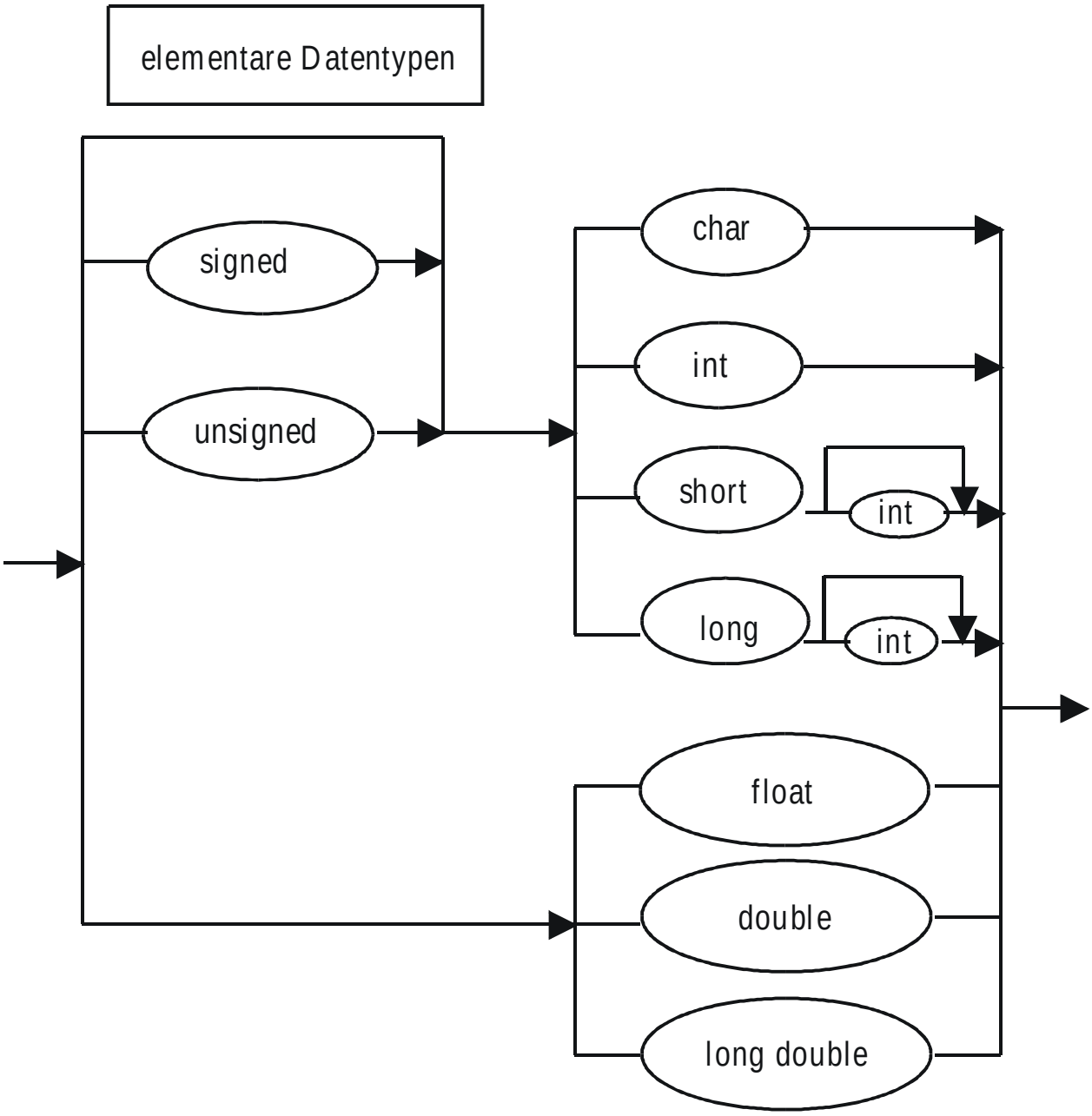



**Hinweis**: strukturierte Typen und Zeigertypen werden später erklärt.

# Einfache Datentypen



# Elementare Datentypen




Elementare Datentypen sind die Grundbausteine aller Datentypen.

# Beschreibung der elementaren Datentypen (1)

## 1.) integer

unsigned int = natürliche Zahlen inklusive 0, als Binärzahlen abgespeichert.

(signed) int = Ganzzahlen, in der Regel in Zweierkomplementdarstellung abgespeichert.

**Beispiele:**

| Bits | unsigned | signed |
|---|---|---|
| 0 0 0 | 0 | 0 |
| 0 0 1 | 1 | 1 |
| 0 1 0 | 2 | 2 |
| 0 1 1 | 3 | 3 |
| 1 0 0 | 4 | - 4 |
| 1 0 1 | 5 | - 3 |
| 1 1 0 | 6 | - 2 |
| 1 1 1 | 7 | - 1 |

N = Anzahl Bits

Wertebereich unsigned: 0 bis $2^{N - 1}$

Wertebereich signed: $-2^{N-1}$ bis $2^{N-1} - 1$

# Beschreibung der elementaren Datentypen (2)

## 2.) char / short / long int

codieren verschiedene Längen von ganzen Zahlen.

Es gilt:

$$\text{len(char)} \leq \text{len(short)} \leq \text{len(int)} \leq \text{len(long)}$$

In der Regel belegt char 8 Bits (1Byte), short 16 Bits (2 Bytes) und int 32 Bits (4 Bytes). Dies ist jedoch abhängig von der Hardware-Architektur des Zielrechners und vom Compiler.

### Beispiele für ganze Zahlen in C

| | |
|---|---|
| 456 | int |
| 2000000 | long |
| 456ul | unsigned long |
| O23 | int in oktaler Schreibweise |
| 'a' | char |
| 0XAB4 | int in hexadezimaler Schreibweise |
| 0xffffu | unsigned int in hexadezimaler Schreibweise |

# Beschreibung der elementaren Datentypen (3)

## Zeichen (char)

Für den Datentyp char gilt, dass der Wertebereich aus diskreten Einzelwerten besteht, die einer Ordnungsrelation unterliegen. Zeichen werden in einem Byte gespeichert und auf ganze Zahlen abgebildet (z. B. ASCII-Code).

Im ASCII-Code entspricht '0' der Zahl 48, 'A' der Zahl 65 und 'a' der Zahl 97.

Zeichenkonstanten wie 'a', 'b', '0', '1' werden vom Compiler in Ganzzahlwerte (int) umgewandelt.

Für Steuerzeichen gibt es in C eine Ersatzdarstellung.

Beispiele: \n steht für Zeilenvorschub
\g steht für Glocke
\t steht für Tabulator.

Der backslash (\) wird dabei als Escape-Zeichen bezeichnet, welches "die normale Interpretationsebene verlässt".

# Beschreibung der elementaren Datentypen (4)

## Merke

Die Ordnung der Zeichen ist in C implementierungsabhängig!!

Der Standard fordert lediglich, dass die Ziffern 0 - 9 aufsteigend und lückenlos angeordnet sind sowie dass die Großbuchstaben und die Kleinbuchstaben in alphabetischer Reihenfolge angeordnet sind. Eine Ordnung zwischen der Menge der Großbuchstaben und der Menge der Kleinbuchstaben ist beispielsweise nicht vorgeschrieben.

# Beschreibung der elementaren Datentypen (5)

## 3.) float/double

Diese werden zur Darstellung von reellen Zahlen (Gleitkommazahlen) verwendet.

Der im Prinzip kontinuierliche Zahlenbereich der reellen Zahlen kann im Computer nicht vollständig dargestellt werden, da hierzu unendlich viele Codewörter benötigt würden. Man kann nur rationale Zahlen mit einer begrenzten Anzahl von Stellen darstellen. Dazu werden reelle Zahlen ins Dualsystem übertragen und in Gleitpunktdarstellung mit Vorzeichen, Mantisse und Exponent gespeichert.

**Beispiel**

$5.75 = 4 + 0 + 1 + 0.5 + 0.25$

$= 1*2^2 + 0*2^1 + 1*2^0 + 1*2^{-1} + 1*2^{-2}$

$= 101.11$

$= 0.10111 * 2^3$

Wir codieren:

Vorzeichen: +  Mantisse: 10111  Exponent: 3

C kennt die Typen float, double und long double für Gleitkommazahlen.

# Beschreibung der elementaren Datentypen (6)

Folgende Grenzwerte müssen von einem Compiler mindestens eingehalten werden:

| Typ | kleinste Zahl | größte Zahl | Genauigkeit |
|---|---|---|---|
| float | 1E-37 | 1E+37 | 1E-5 |
| double | 1E-37 | 1E+37 | 1E-9 |

# Beschreibung der elementaren Datentypen (7)

## 4.) Wahrheitswerte

Jede ganze Zahl x kann in C einen Wahrheitswert darstellen!

Dabei gilt:

**`x = 0`** wird als **`false`** interpretiert.

**`x ≠ 0`** wird als **`true`** interpretiert.

Falls ein Wahrheitswert in einer Variablen gespeichert werden soll, gilt:

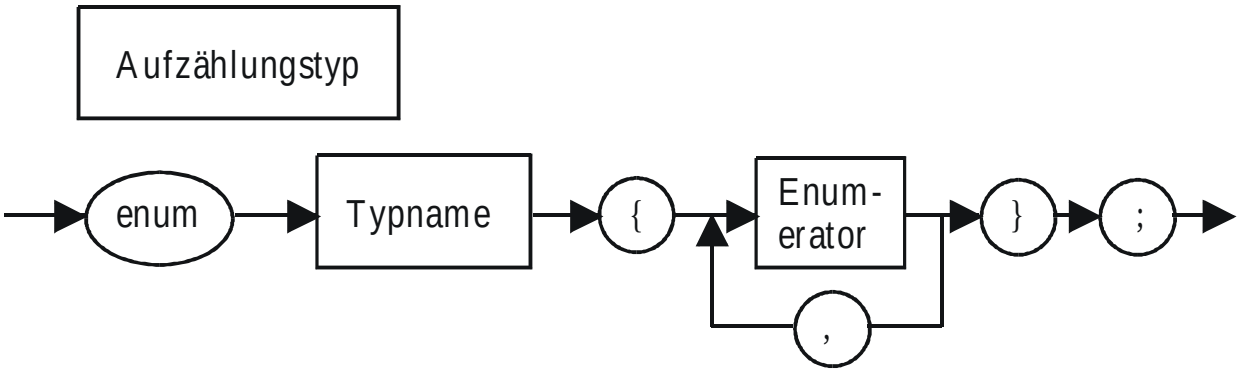
**`false`** setzt x auf 0

**`true`** setzt x auf 1

# Aufzählungstyp (enumeration type) (1)

Eine **Aufzählung** ist eine Folge von Zeichenkonstanten, denen vom Compiler je ein Integer-Wert zugeordnet wird.

**Syntaxdiagramm**:




**Beispiel**: enum Wahrheitswert {falsch, wahr};

Wahrheitswert stellt nun einen Typ dar, der die Werte falsch und wahr annehmen kann. Die Elemente der Liste werden impliziert nummeriert, beginnend mit 0. Im Beispiel hat also das Element falsch den Wert 0, wahr den Wert 1. Durch die Reihenfolge der Aufzählung wird eine Ordnung definiert, also **`falsch < wahr`**. Man kann die Zuordnung von Integerwerten jedoch auch selbst steuern.

# Aufzählungstyp (enumeration type) (2)

**Beispiele**

enum fruit {apple = 7, pear, orange = 3, lemon, peach}

Nun gilt: apple = 7, pear = 8, orange = 3, lemon = 4, peach = 5.

enum controls {TAB = '\t', NEWLINE = '\n', RETURN = '\r'}

**Merke**

Aufzählungen werden syntaktisch wie Konstanten behandelt.

Mehrfache Verwendung desselben Integerwertes ist möglich, aber die Enumeratoren müssen eindeutig sein.

# Konstantenvereinbarung

In C gibt es zwei Möglichkeiten, Konstanten zu vereinbaren:

- const-Vereinbarung

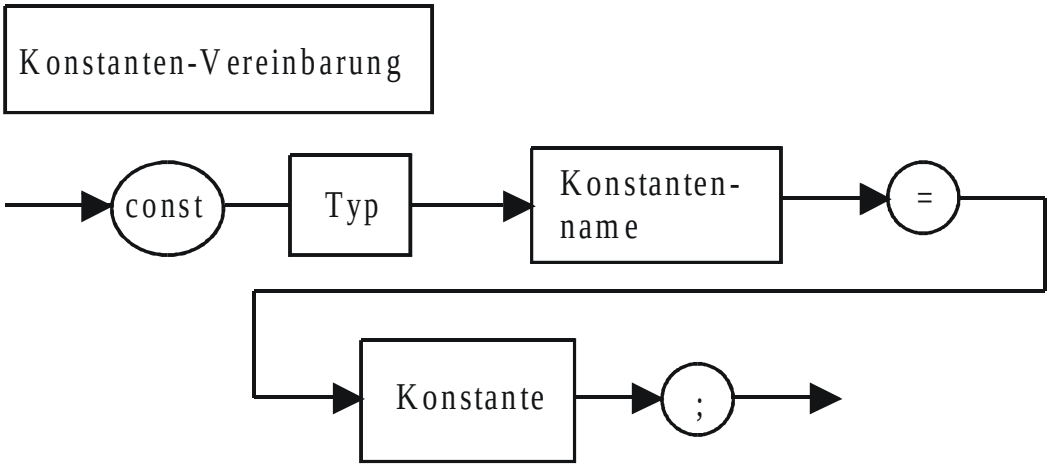
  Die const-Vereinbarung definiert eine **Variable**, die nur gelesen und nicht zugewiesen werden darf. Die Verwendung zur Definition der Länge einer anderen Datenstruktur (z. B. Vektorlänge) ist deshalb nicht erlaubt.

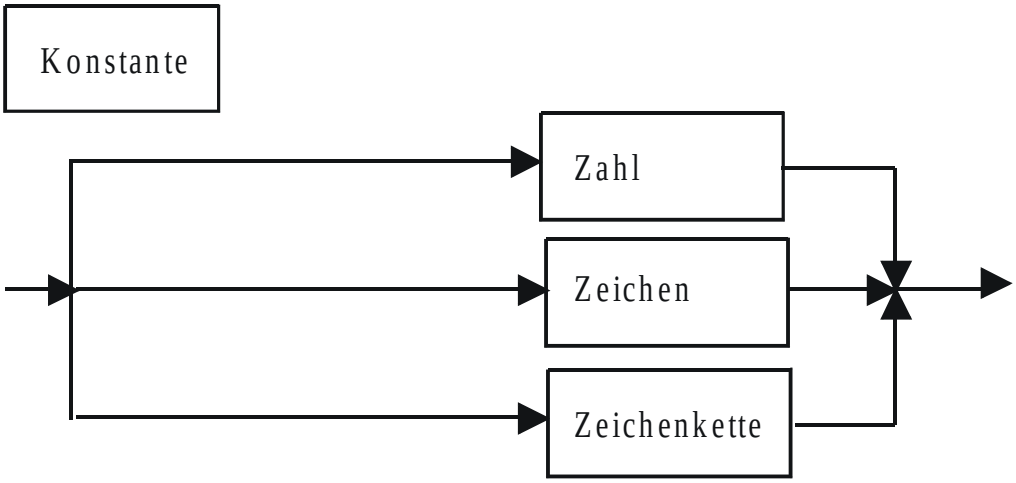
- #define-Klausel
  **`#define suchtext ersatztext`**

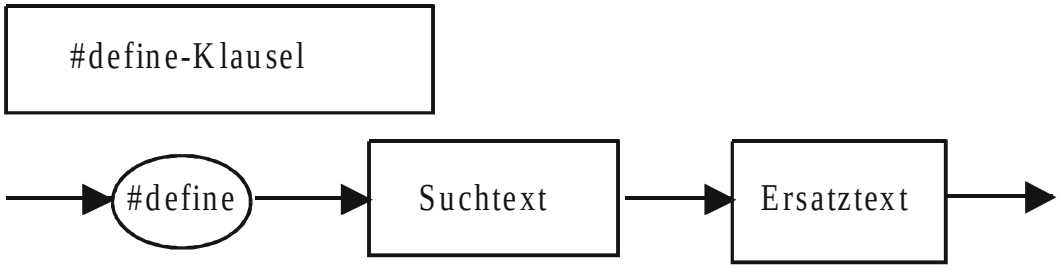
  Die #define-Klausel definiert so genannte **Compiler-Konstanten.** Beim Übersetzen wird vor dem eigentlichen Übersetzungsschritt im gesamten Programmcode der **`suchtext`** durch den **`ersatztext`** ersetzt. Wenn man also **`suchtext`** als Konstantenname und **`ersatztext`** als Konstante ansieht, hat man eine Compiler-Konstante.

  In der #define-Klausel können als Konstanten auch Ausdrücke eingesetzt werden.

# Syntax in C


Konstanten-Vereinbarung
const
Typ
Konstanten-name
=
Konstante
;


Konstante
Zahl
Zeichen
Zeichenkette


#define-Klausel
#define
Suchtext
Ersatztext

# Beispiele für Konstantenvereinbarungen

1) Zahlen

```
const       float pi = 3.14159;
const       int ganze_Zahl = 79;
const       int hexa_Zahl = 0x13;
```

2) Zeichen

```
const       char anfang = "a";
```

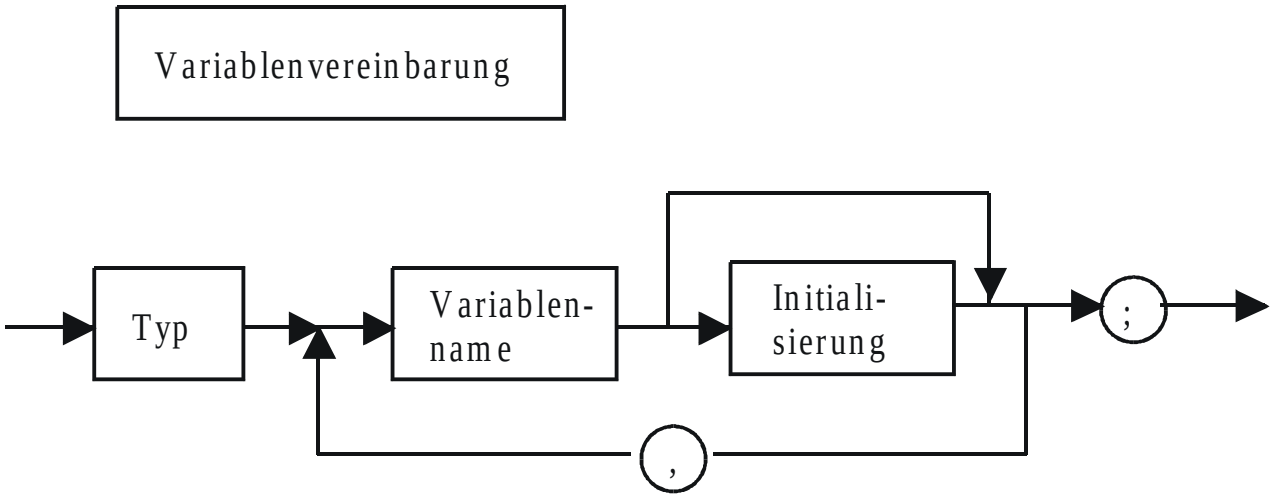
3) Zeichenkette

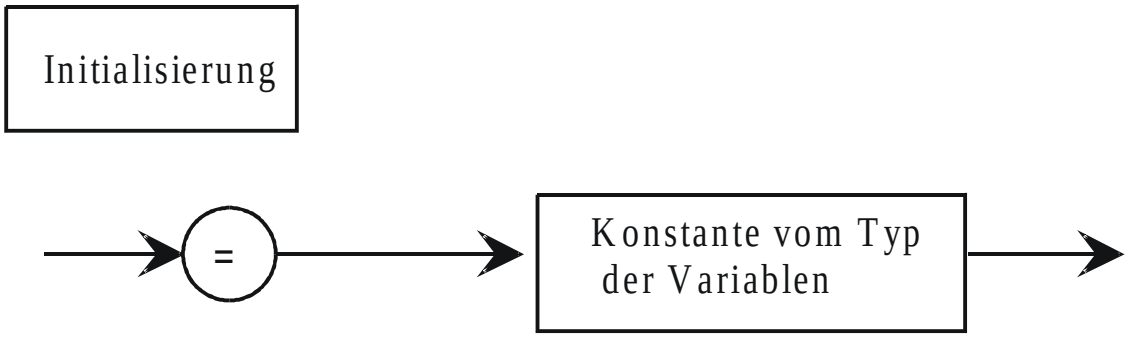
```
const       char* alphabet = "abc";
```

4) #define-Konstanten

```
#define WAHR 1
#define FALSCH 0
```

# Variablenvereinbarung




## Beispiele

```
int    anz_kinder;
float  groesse;
char   initial_vorname, initial_nachname;
int    a = 5;
```