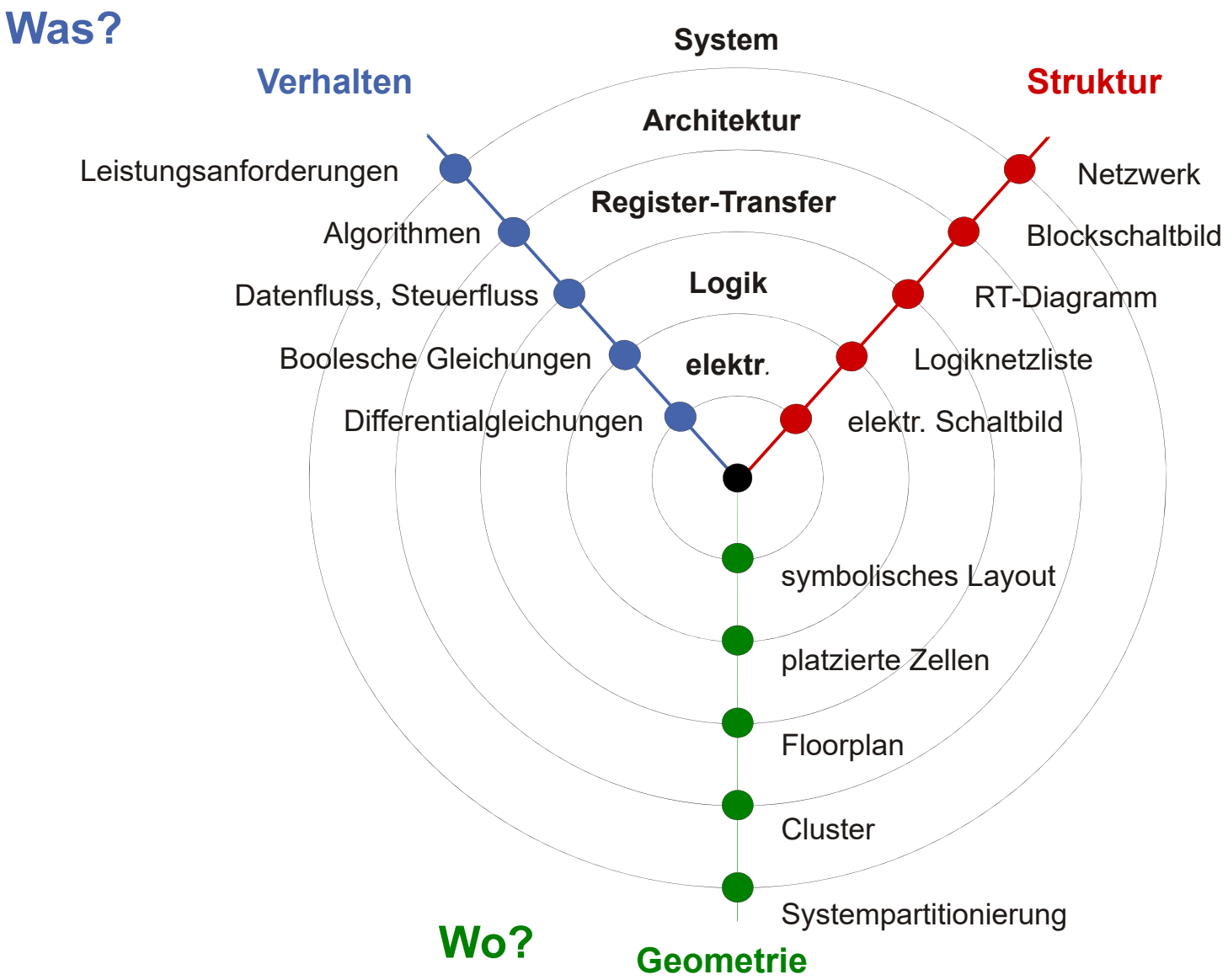
Hardwarearchitektur: Entwurfsprinzipien

- Legt die Struktur und den Aufbau eines Rechensystems aus einzelnen Teilelementen fest und beschreibt die Schnittstellen dieser Elemente und deren Zusammenwirken zur Realisierung des Gesamtsystems
- **Problem: Hohe Komplexität**
  - Mehrere Hundert Millionen Transistoren auf heutigen ICs
- **Lösung: Hierarchischer Entwurfsstil**
  - Ausgehend von einer Darstellungsebene hohen Abstraktionsgrades wird das zu entwerfende Systems über mehrere Abstraktionsebenen immer weiter verfeinert

Y-Diagramm [D. D. Gajski] (1)



Y-Diagramm [D. D. Gajski] (2)

- Entwickelt von Daniel D. Gajski in 1983
- Beschreibung der unterschiedlichen Sichtweisen beim Hardwareentwurf
  - **Entwurfsebenen:** Abstraktionsgrad eines Entwurfs
    - Kreise im Diagramm
    - Äußere Kreise stellen Verallgemeinerungen dar
    - Innere Kreise sind Verfeinerungen
  - **Sichten:** Darstellung eines Entwurfs auf einer Ebene
    - 3 Sichten: Achsen des Diagramms
      - funktionell (was)
      - strukturell (wie)
      - physikalisch (wo)

Kapitel 2
Die Programmiersprache C

- Vom Quellcode zum ausführbaren Programm
- Die Entwicklungsgeschichte von C
- Grundlagen: Datentypen, Operatoren, Ausdrücke
- Kontrollstrukturen
- Funktionen und Programmstruktur
- Zeiger und Vektoren

Die Programmiersprache C

Begriffe

- **Maschinensprache:** Repräsentation von Anweisungen, die für einen Mikroprozessor unmittelbar verständlich sind, z.B.

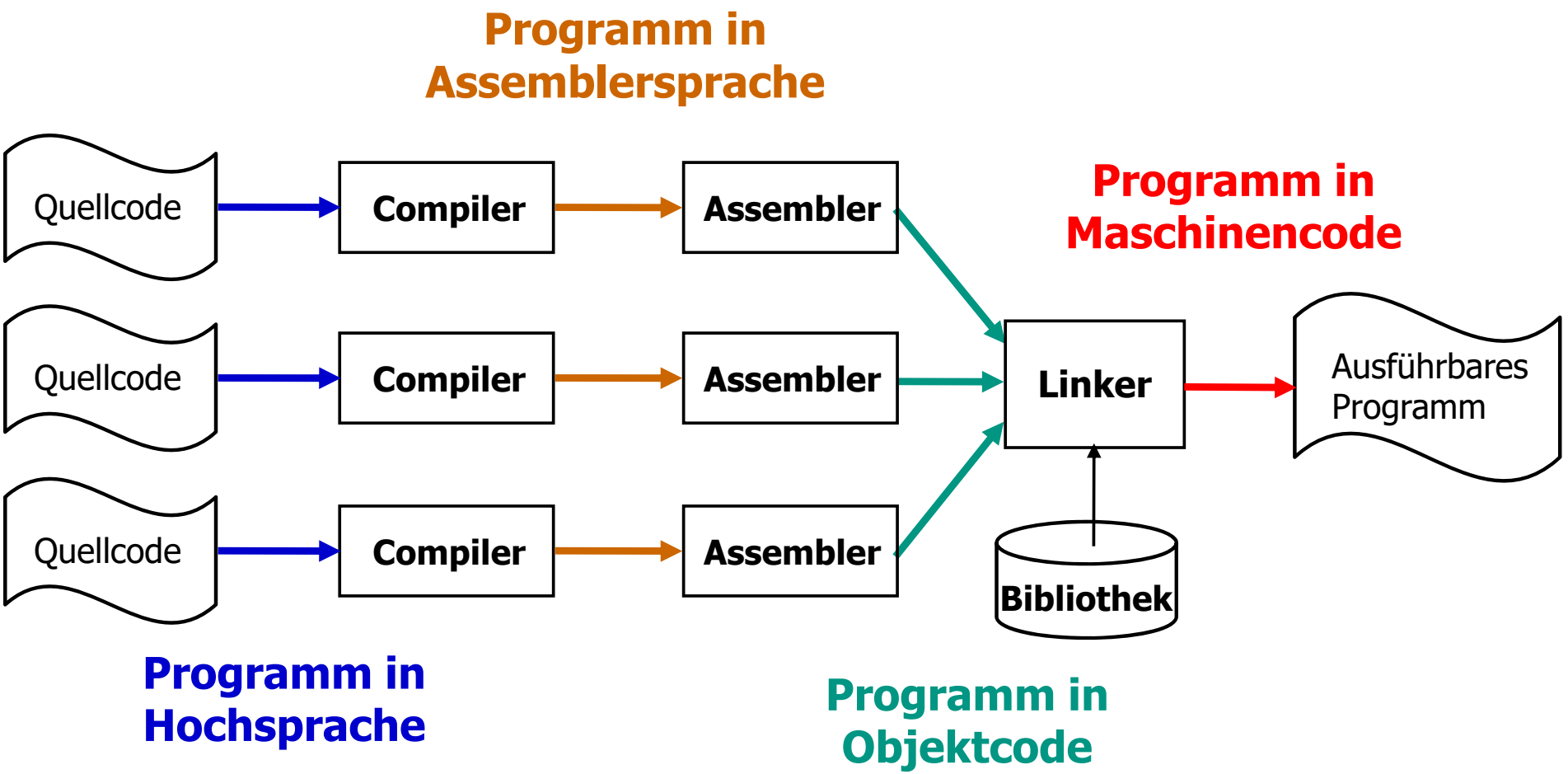
```
00000000110000100011000000100001
```

- **Assemblersprache:** Symbolische Repräsentation der Maschinensprache, die für den Menschen verständlich und anschaulich ist, z.B.

```
add $s2, $s1, $s0 # $s2:= $s1 + $s0
```

- Symbolischer Befehl ≡ Maschinen-Befehl

2.1 Vom Quellcode zum ausführbaren Programm



Vom Quellcode zum ausführbaren Programm

- **Assembler:**
  Programm, das einen Quellcode in Assemblersprache in eindeutiger Weise in Maschinensprache übersetzt

- **Objektcode:**
  Repräsentation eines Maschinenprogramms, in dem noch ungelöste Referenzen auf externe Unterprogramme oder Speicherbereiche enthalten sind

- Zusätzlich können im Objektcode Informationen enthalten sein, die die Fehlersuche mit einem **Debugger** ermöglichen

- **Linker**: Programm, das die ungelösten Referenzen mehrere Objektcode-Module auflöst und sie zu einem ausführbaren Programm verbindet

2.2 Entwicklungsgeschichte von C

- **1969:** Ken Thomson (Bell Laboratories) erstellte erste Version von UNIX in Assembler
- **1970:** Ken Thomson entwickelte auf einer PDP 7 (Minirechner von Digital Equipment Corporation (DEC)) die Sprache B als Weiterentwicklung der Sprache BCPL
  - B ist eine typlose Sprache, sie kennt nur Maschinenworte
- **1974:** Weiterentwicklung von B zu C durch Dennis M. Ritchie; erste Implementation auf einer PDP 11
- **Heute:** C ist eigenständige, betriebssystemunabhängige Programmiersprache; sie ist auf praktisch allen Rechnerplattformen vom PC bis hin zum Supercomputer und unter allen wichtigen Betriebssystemen verfügbar

Entwicklungsgeschichte von C

- Im Laufe der Zeit entstanden "C-Dialekte", welche die dringende Notwendigkeit einer Standardisierung zeigten
- 1988 hat das ANSI-Komitee X3J11 diesen Sprachstandard für die Programmiersprache C veröffentlicht, der kurz ANSI C genannt wird
  - Neuere C-Compiler sollten dem ANSI-Standard entsprechen
- **The C programming language** von Brian W. Kernighan und Dennis M. Ritchie, 9. Auflage, ANSI-Standard
- Die Entwicklung von ist C eng mit der UNIX-Entwicklung verbunden

Die Programmiersprache C

- C nimmt eine Zwischenstellung zwischen Assembler und Hochsprache ein

- Sie vereint zwei an sich widersprüchliche Eigenschaften:
  - gute Anpassung an die Rechnerarchitektur (Hardware)
  - hohe Portabilität → einfachere und vor allem schnellere Anpassungen an eine andere Hardware

- Einfachere Programmierung → kürze Entwicklungszeiten für die Software

- C-Compiler erzeugen sehr effizienten Code sowohl bzgl. Laufzeit als auch bzgl. Programmgröße

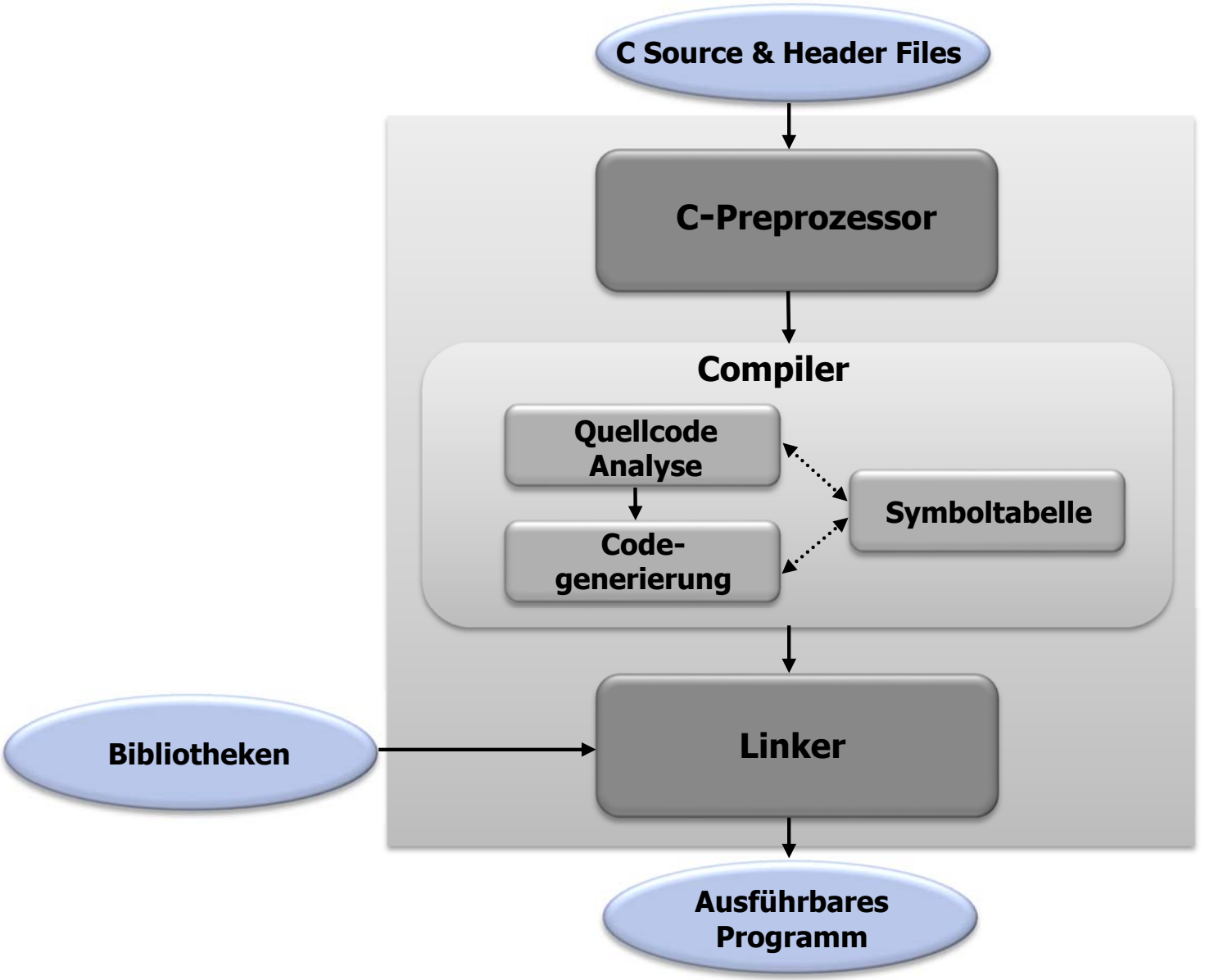
Die Programmiersprache C

- 4 grundlegende Datentypen: **char**, **int**, **float** und **double**
- Es gibt einfache Kontrollstrukturen: Entscheidungen, Schleifen, Zusammenfassungen von Anweisungen (Blöcke) und Unterprogramme
- Ein-/Ausgabe ist nicht Teil der Sprache C sondern wird über Bibliotheksfunktionen erledigt
- Keine Operationen auf zusammengesetzten Datenstrukturen als Ganzes (z.B. Zeichenketten)
- Parameter werden als Werte an Funktionen übergeben, daher kann eine Funktion diese nicht ändern

Die Programmiersprache C

- Möglichkeit von Zeigern als Parametern, deren Zielobjekt geändert werden kann
  - Zeiger (Pointer) stellen sehr mächtige und vielseitige Anwendungsmöglichkeiten bereit
  - Sie werden durch eine automatische Adressarithmetik unterstützt
- Für Datenwandlungen zwischen den einzelnen Typen gibt es kaum Beschränkungen
- Weiterhin gibt es z.B. mittels der Speicherklasse "register" Bezüge zur Hardware
- ...

Die Programmiersprache C



Die Programmiersprache C

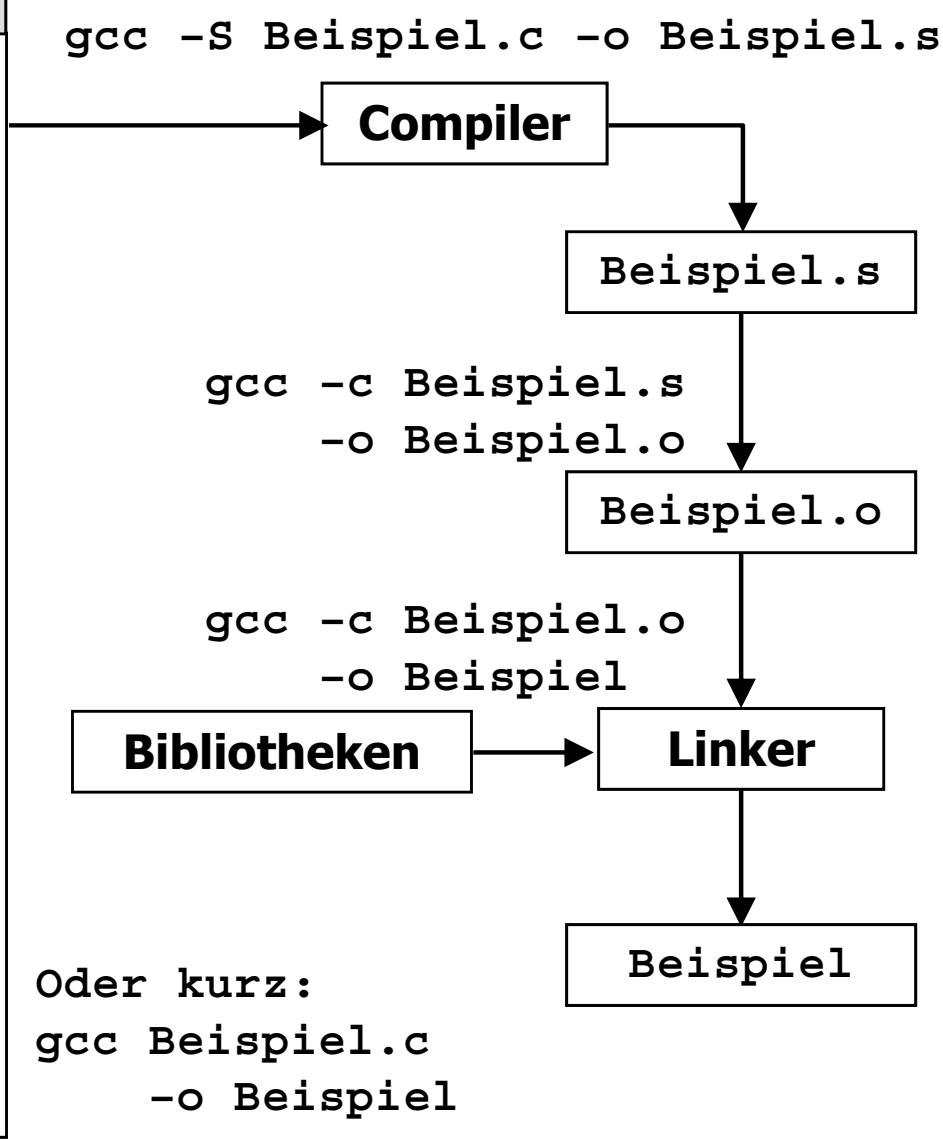
Beispiel.c

```
/* Das ist ein Blockkommentar.
   Kann sich über mehrere Zeilen
   erstrecken und muss am Ende
   geschlossen werden.
*/

// Das ist ein Zeilenkommentar.
// Gilt nur bis zum Ende der Zeile.
// Ist eigentlich kein gültiges C
// (sondern C++), wird aber von
// vielen Compilern toleriert.


#include <stdio.h> // Standard I/O

main()
{
  printf("Viel Spass mit TI!\n");
}
```



2.3 Grundlagen: Datentypen (1)

- **4 Grunddatentypen**
  - **char:** einzelnes Zeichen (Charakter); meist 1 Byte
  - **int:** Integerzahl; meist 2 oder 4 Byte
    - Einstellbare Genauigkeiten:
      - short int
      - long int
      - long long int
      - Auch mit signed/unsigned kombinierbar
  - **float:** Gleitkommazahl; meist 4 Byte
  - **double:** Gleitkommazahl in doppelter Genauigkeit; meist 8 Byte

Grundlagen: Datentypen

- **4 Grunddatentypen**
  - Wertebereiche dieser Grunddatentypen ist rechnerabhängig !
    - In <limits.h> und <float.h> stehen die Größen, die größte darstellbare Zahl, …
    - Funktion „sizeof“: Größe eines Datentyps in Byte
    - In C gilt immer:
      - sizeof(char) ≤ sizeof(short int) ≤ sizeof(int) ≤ sizeof(long int) ≤ sizeof(long long int)
  - Beispiele:
    - unsigned char buchstabe = ´T´; /* eigentlich ein Zahlendatentyp (0-255), der hier mit dem ASCII Wert von ´T´ initialisiert wird */
    - int x = 4, y, z = 42;
    - const double pi = 3.1415;

Grundlagen: Operatoren

Arithmetische Operatoren

Operator Symbol	Operation	Beispiel
*	Multiplikation	x * y
/	Division	x / y
%	Modulo	x % y
+	Addition	x + y
-	Subtraktion	x - y

Grundlagen: Operatoren

- **Bit-Operatoren:**

Operator Symbol	Operation	Beispiel
~	Bitweise NOT	~x
<<	links Schieben	x << y
>>	rechts Schieben	x >> y
&	Bitweise AND	x & y
^	Bitweise XOR	x ^ y
|	Bitweise OR	x | y

Grundlagen: Operatoren

- **Vergleichsoperatoren:**

Operator Symbol	Operation	Beispiel
>	größer als	x > y
>=	größer gleich	x >= y
<	kleiner als	x < y
<=	kleiner gleich	x <= y
==	gleich	x == y
!=	ungleich	x != y

Grundlagen: Operatoren

- **Spezial-Operatoren:**

Operator Symbol	Operation	Beispiel
++	inkrement (postfix)	x++
--	dekrement (postfix)	x--
++	inkrement (präfix)	++x
--	dekrement (präfix)	--x
+=	add and assign	x += y
-=	subtract and assign	x -= y
*=	multiply and assign	x *= y
/=	divide and assign	x /= y
%=	modulus and assign	x %= y
&=	and and assign	x &= y
|=	or and assign	x |= y
^=	xor and assign	x ^= y
<<=	left-shift and assign	x <<= y
>>=	right-shift and assign	x >>= y

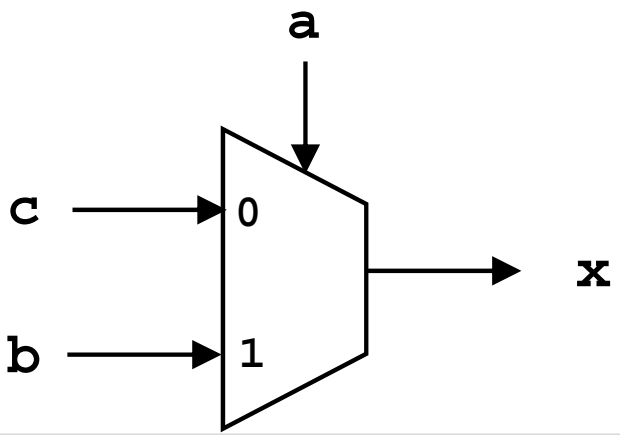
Grundlagen: Operatoren

- **Spezial-Operatoren:**
  - **Operator-paar „?:“** `expr1 ? expr2 : expr3`
  - Beispiel 1:

```
z = (a > b) ? a : b;          /* z = max(a,b) */
```

  - Beispiel 2:

```
x = a ? b : c                 /* a true dann x = b  */
                              /* sonst x = c  */
```



Bedingter Ausdruck entspricht der logischen Funktion eines Multiplexers

Karlsruher Institut für Technologie

Grundlagen: Operatoren

- **Operatoren nach Priorität:**

Priorität	Symbol	Assoziativität	Bedeutung
15	()	L-R	Funktionsaufruf
	[]		Indizierung
	->	R-L	Elementzugriff
	.		Elementzugriff
14	+(Vorzeichen)	R-L	Vorzeichen
	-(Vorzeichen)		Vorzeichen
	!		
	~		Bitkomplement
	++(Präfix)		Präfix-Inkrement
	--(Präfix)		Präfix-Dekrement
	(Postfix)++		Postfix-Inkrement
	(Postfix) --		Postfix-Dekrement
	&		Adresse
	*		Zeigerdereferenzierung
	(Typ)		Typumwandlung
	sizeof		Größe

Grundlagen: Operatoren

- **Operatoren nach Priorität:**

Priorität	Symbol	Assoziativität	Bedeutung
13	*	L-R	Multiplikation
	/		Division
	%		Modulo
12	+		Addition
	-		Subtraktion
11	<<	L-R	Links-Shift
	>>		Rechts-Shift
10	<	L-R	Kleiner
	<=		Kleiner gleich
	>	L-R	größer
	>=		größer gleich
9	==	L-R	gleich
	!=		ungleich

Grundlagen: Operatoren

- **Operatoren nach Priorität:**

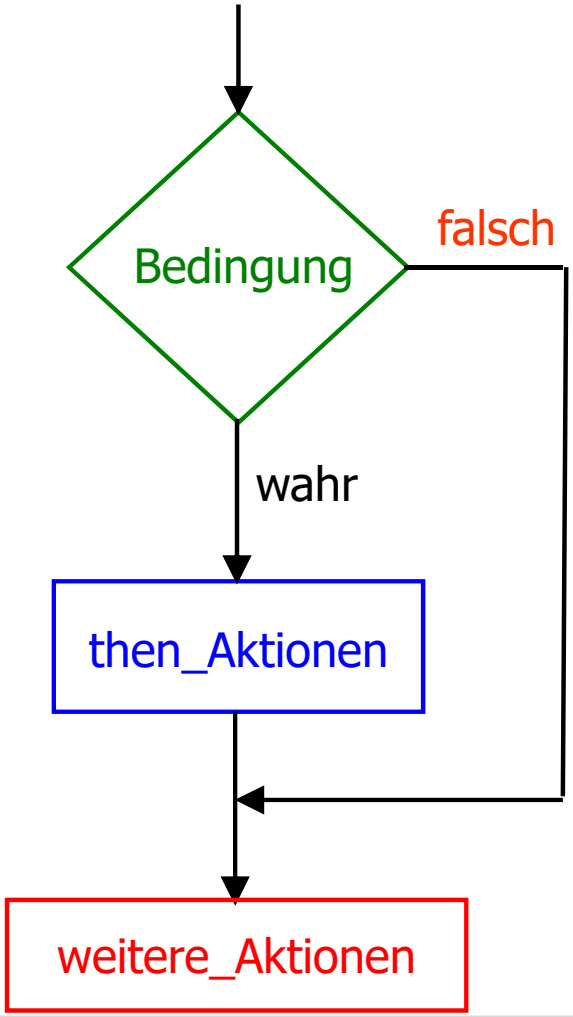
Priorität	Symbol	Assoziativität	Bedeutung
8	&	L-R	Bitweise UND
7	^	L-R	Bitweise exklusives ODER
6	|	L-R	Bitweise ODER
5	&&	L-R	logisches UND
4	||	L-R	Bitweise ODER
3	:?	L-R	Bedingung
2	=	L-R	Zuweisung
	*=, /=, %=, +=, -=, &=, ^=, |=, <<=, >>=		Zusammengesetzte Zuweisung
1	,		Komma-Operator

2.4 Grundlagen: Kontrollstrukturen

Kontrollstrukturen definieren die Reihenfolge von Berechnungen

- **`if` Anweisung:**

```
if (Bedingung)
{
        then_Aktionen;
}
weitere_Aktionen;
```



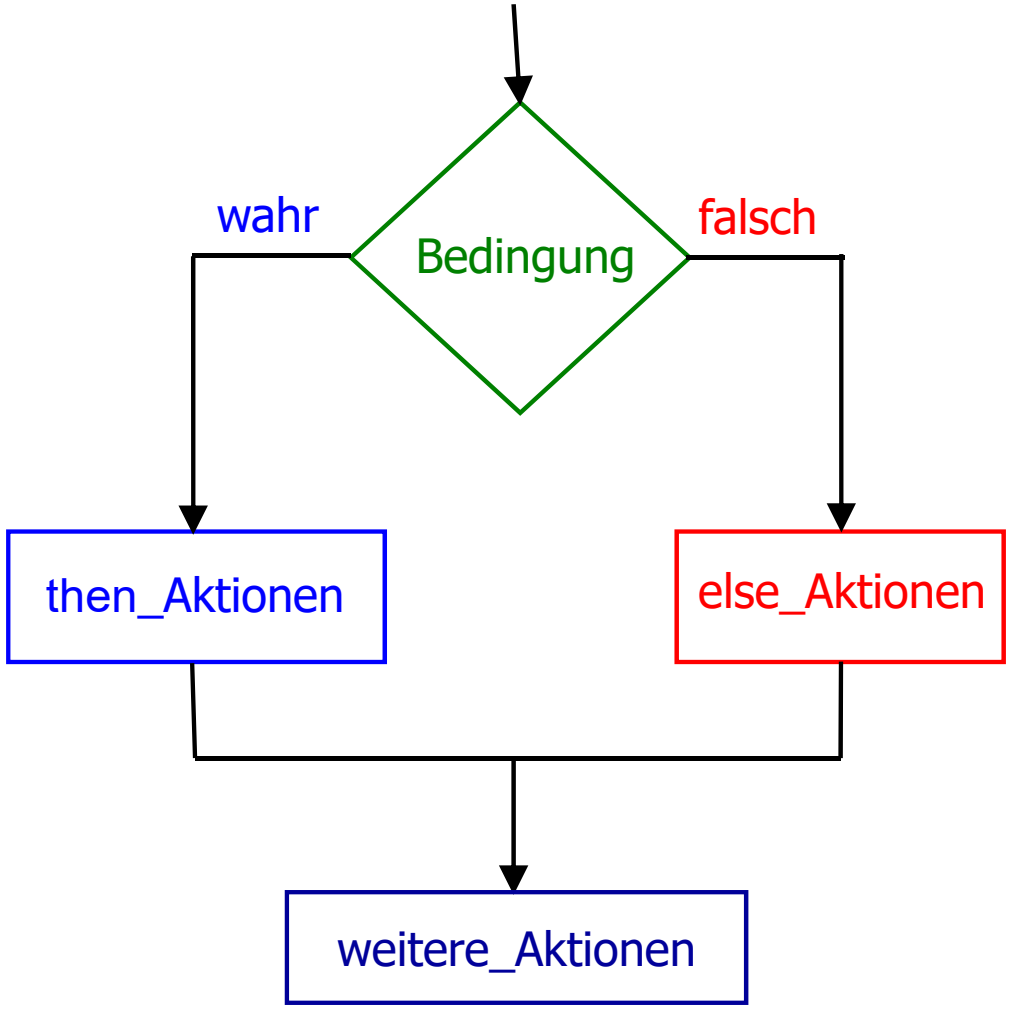
Grundlagen: Kontrollstrukturen

Kontrollstrukturen definieren die Reihenfolge von Berechnungen

- **`if-else` Anweisung:**

```
if (Bedingung)
{
    then_Aktionen;
}
else
{
    else_Aktionen;
}

weitere_Aktionen;
```



Grundlagen: Kontrollstrukturen

Kontrollstrukturen definieren die Reihenfolge von Berechnungen

- `switch` **Anweisung:**

```
switch(ausdruck)
{
   case a: a Anweisungen    /* falls ausdruck == a */
   break;

   case b: b Anweisungen    /* falls ausdruck == b */
   break;

   case c: c Anweisungen    /* falls ausdruck == c */
   break;

   defaut: default Anweisungen    /* sonst */
   break;

}
```