

Teil 2: Das erste C-Programm

■ Gliederung

Sprache C

Übersetzung

Ablaufsteuerung

Datentypen

Die Sprache C

■ Historische Entwicklung

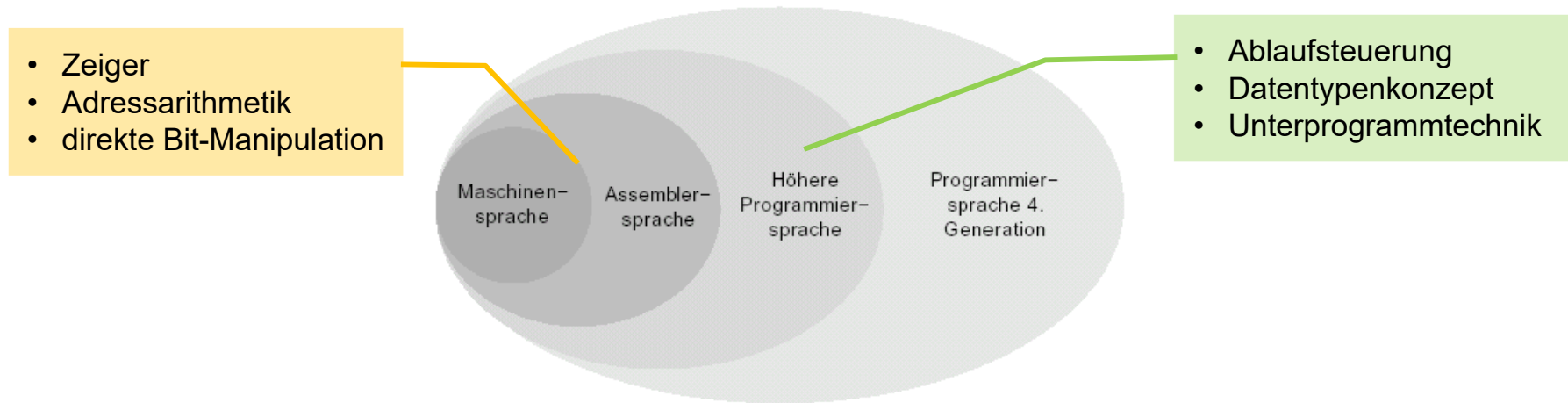
- 1970 Ken Thomson entwickelt "B" (typenlos) als Weiterentwicklung der Sprache BCPL
- 1972 Dennis M. Ritchie Weiterentwicklung von "B" zu "C", (portierbare Sprache zur Entwicklung am UNIX-OS, "Super-Assembler")
- 1978 Brian W. Kernighan und Dennis M. Ritchie **"The C programming language"**
- 1983 Arbeit an ANSI-Standard (1989 veröffentlicht) normiert neben Sprache C auch die Standard-Bibliotheken (C selbst hat z.B. keine Ein- und Ausgabe Funktionalität)

■ Dialekte

- **K&R-C** von den Erfindern Kernighan und Ritchie
- **ANSI-C** standardisierte Normen **C89** C95 **C99** **C11** C17 **C23**
- **C++** objektorientierte Erweiterungen (Bjarne Stroustrup)
- Dialekte sind abwärtskompatibel

■ Eigenschaften der Sprache C

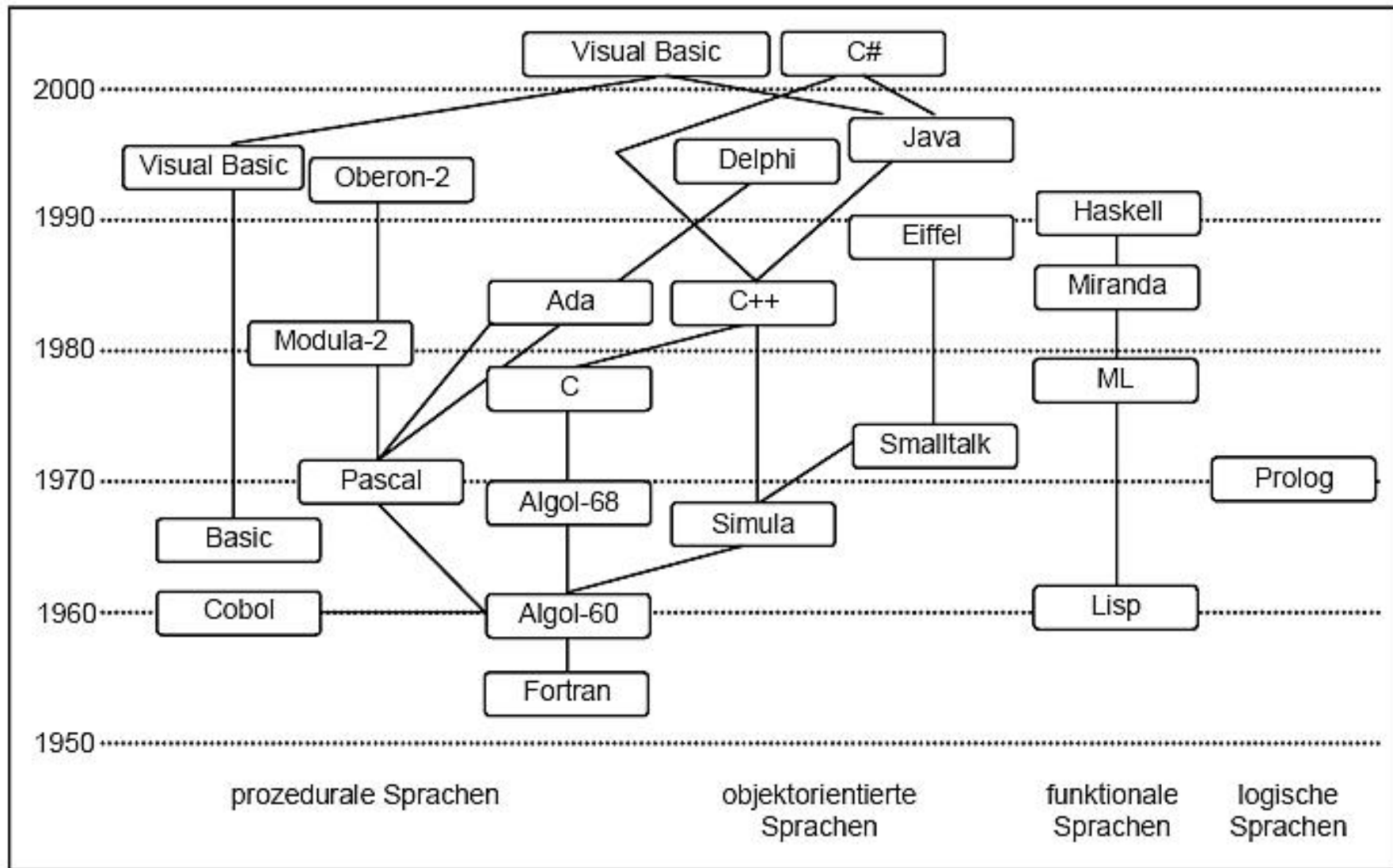
- Synthese zwischen **maschinennaher** und **höherer** Sprache:



- **kleiner Sprachumfang**, leicht erlernbar
- kompakte Quellprogramme -> Gefahr der schlechten Lesbarkeit
- kompakter und **schneller Objekt-Code**
- getrennte Kompilierbarkeit von Programmeinheiten
→ Programme aus verschiedenen Quellmodulen (Bibliothek-Nutzung)

■ Warum C?

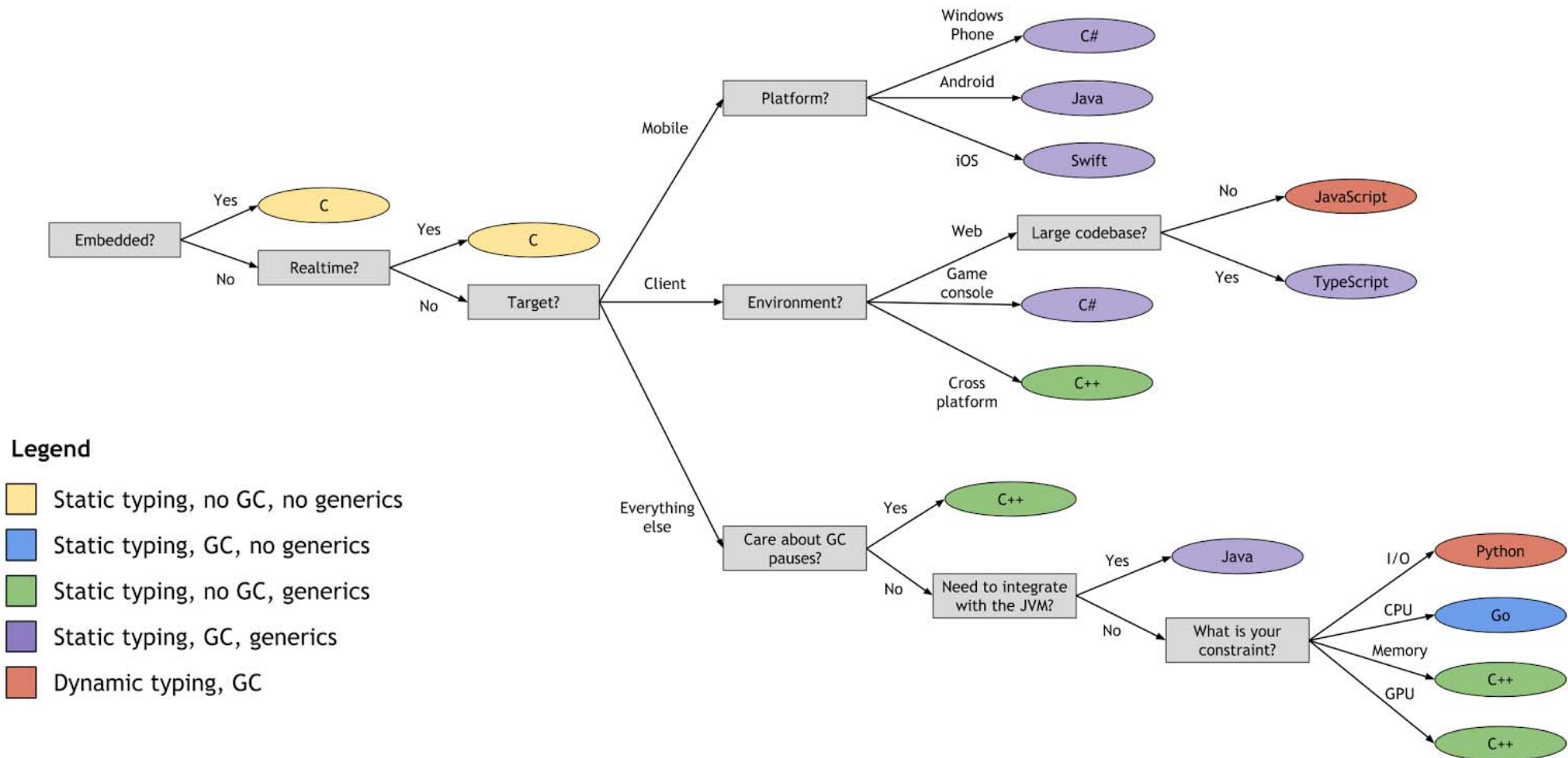
- mächtig und flexibel
- bedient Bedarf nach Leistung, Maschinennähe, Ressourcenknappheit
- Grundkonzepte und Syntax sind Basis vieler Sprachen
- Portabilität
- hoher Verbreitungsgrad → viele verfügbare Bibliotheken
- Aktuelle Popularität: <https://www.tiobe.com/tiobe-index/>



Entwicklung und Verwandtschaft verschiedener Programmiersprachen

Which programming language should I use for my project?

by onebigfluke.com

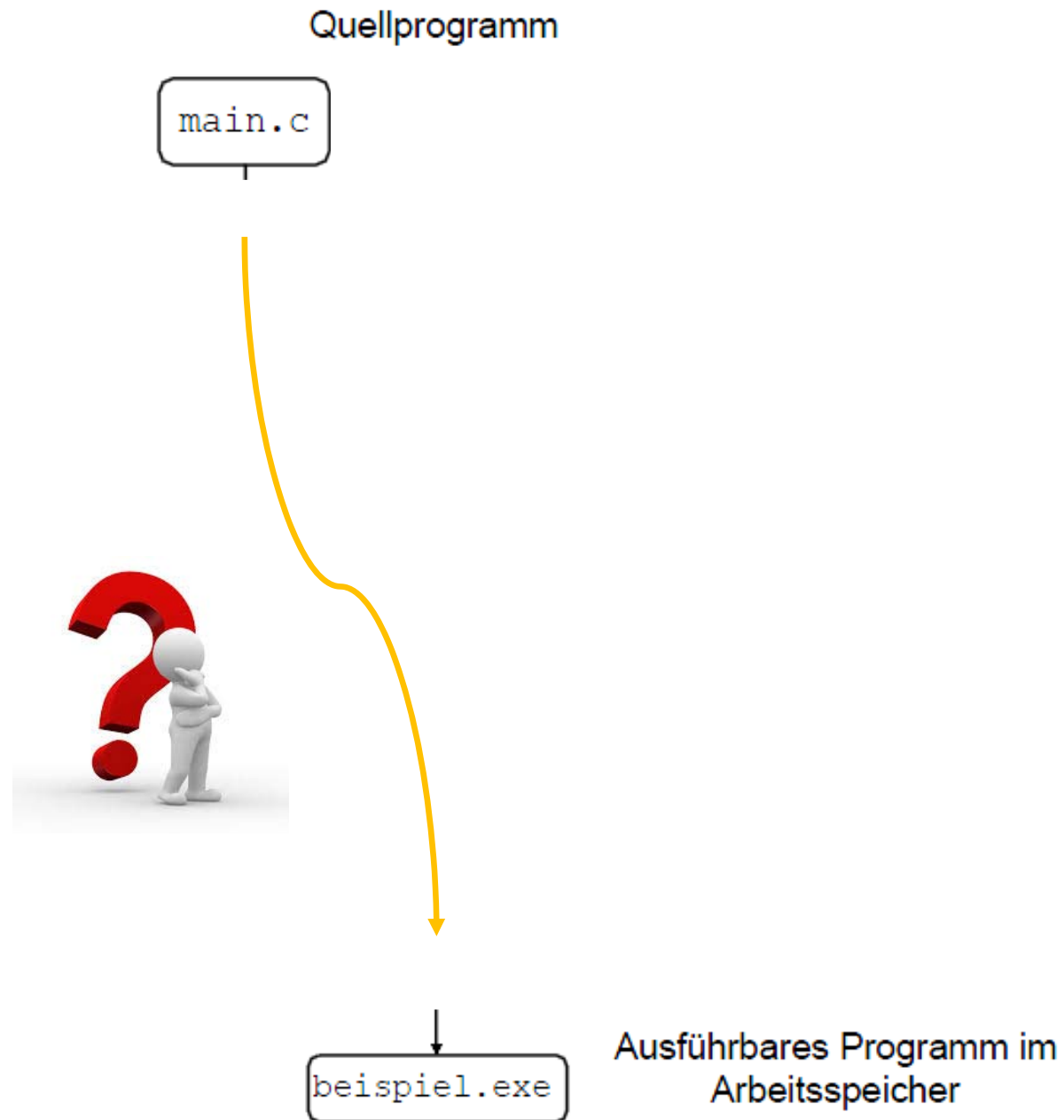


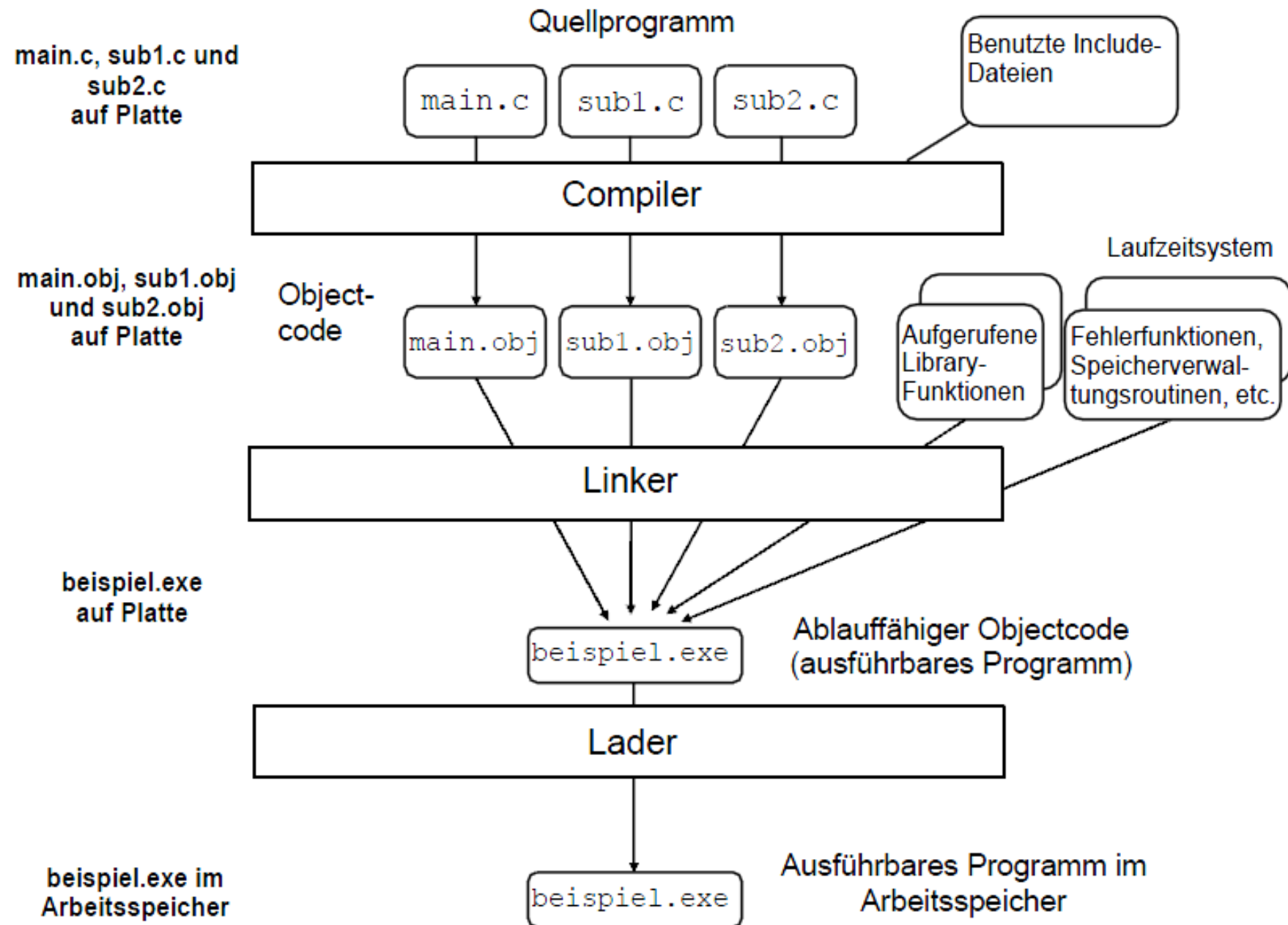
Quelle: "Mother Tongues" <https://ccrma.stanford.edu/courses/250a-fall-2005/docs/ComputerLanguagesChart.png>

■ Hello World

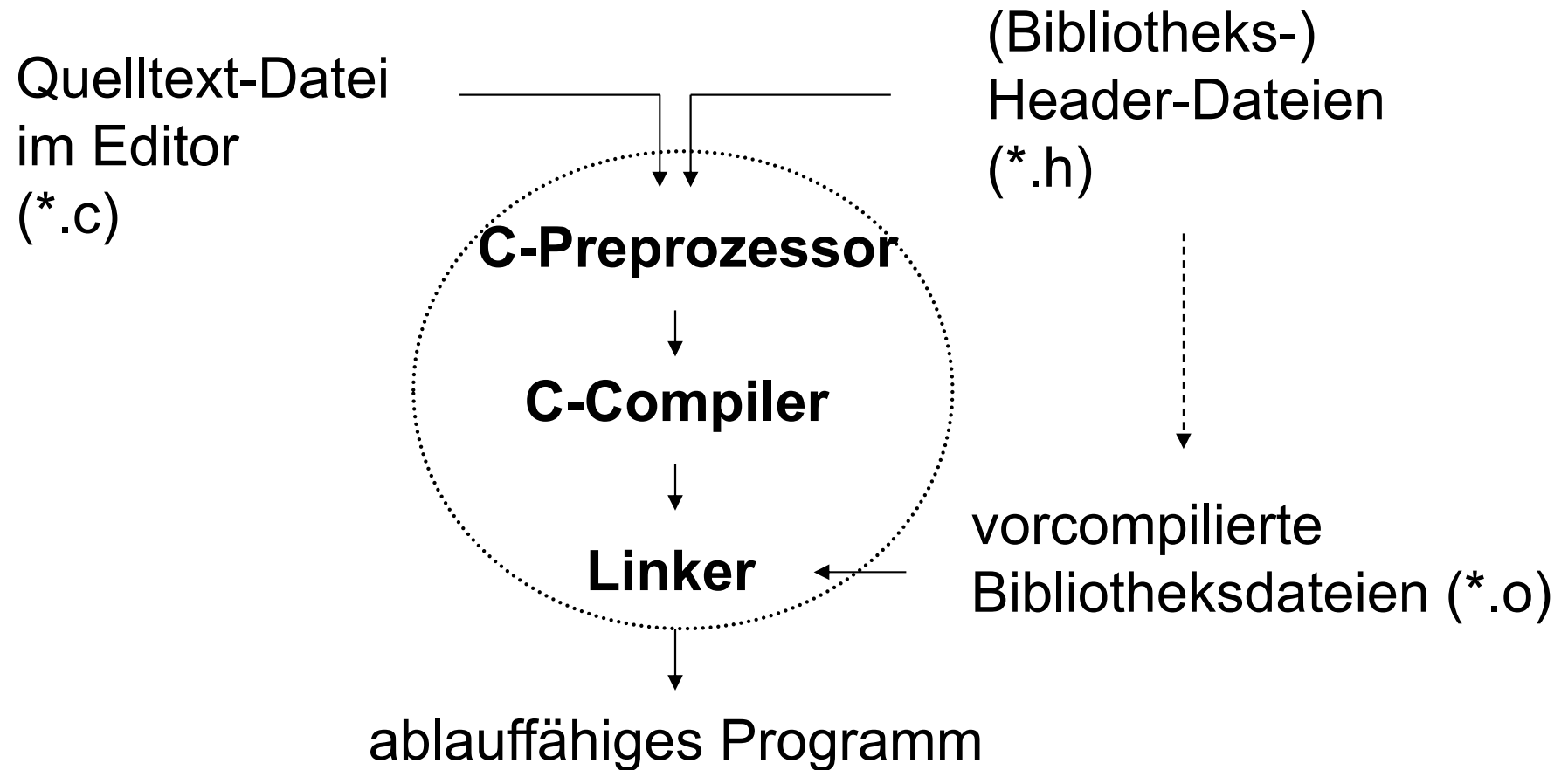
```
/* Datei: hallo.c  
   Ein simples "Hello World" Programm */  
  
#include <stdio.h>  
  
int main()  
{  
    printf("Hallo Welt!\n");  
    return 0;  
}
```

Übersetzung

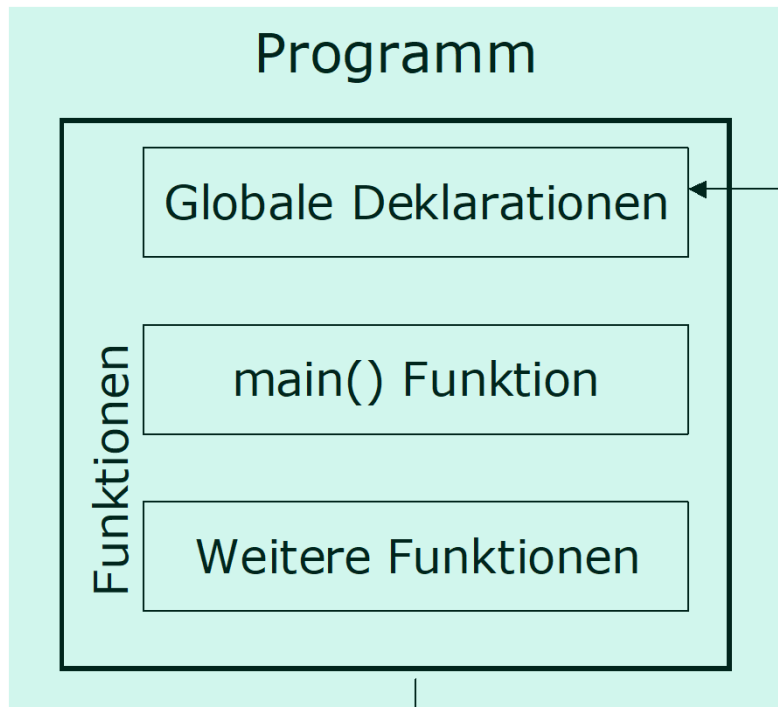




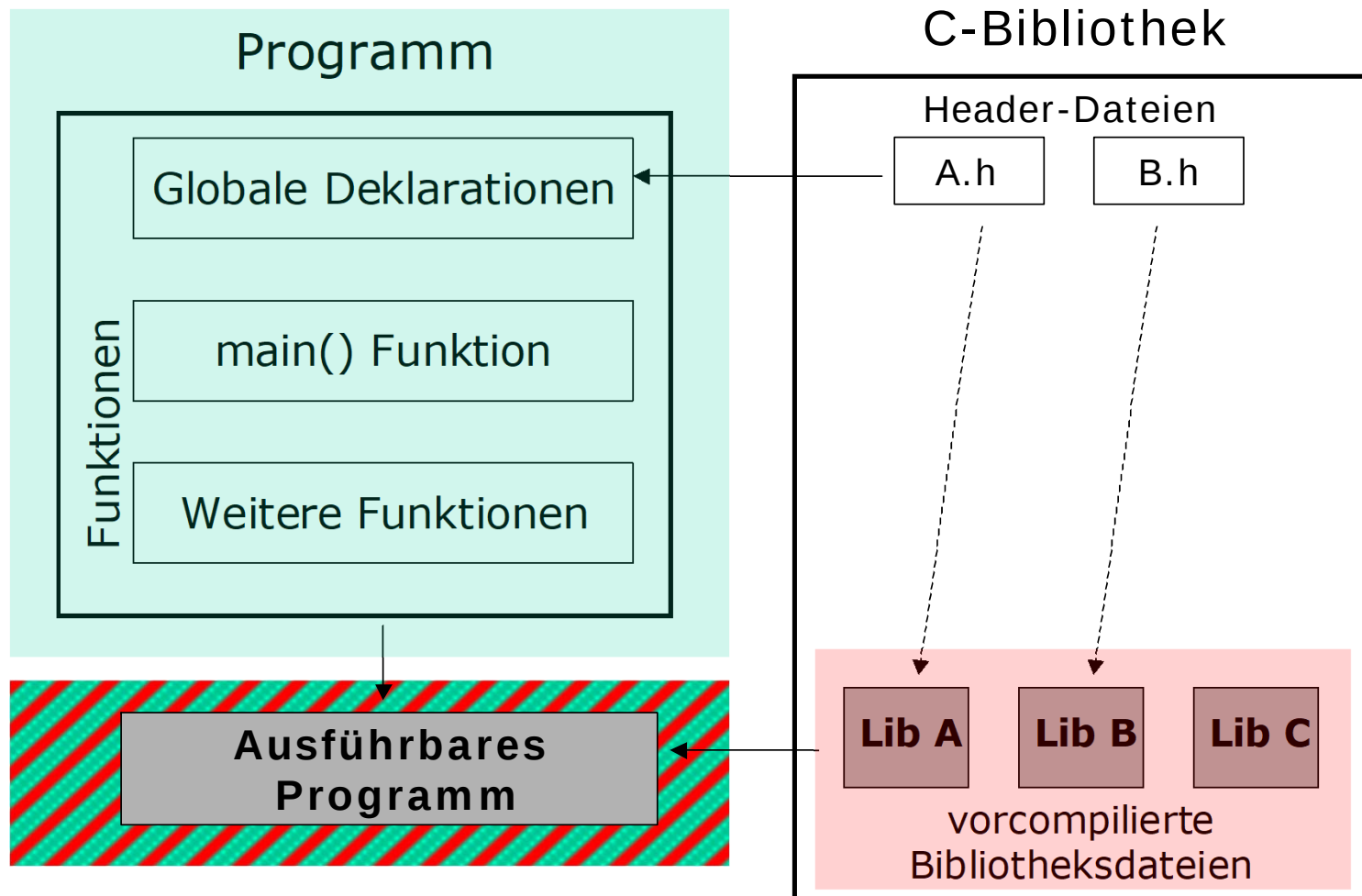
■ Präprozessor - Compiler - Linker



■ Nutzung von Bibliotheken (und Header-Dateien)



■ Nutzung von Bibliotheken (und Header-Dateien)



■ Dateinamenskonventionen

Die Dateinamen haben üblicherweise folgende Erweiterungen:

.c für den Quellcode (.cpp für C++ Quellcode)

.h für Include-Dateien

Linux:

.o für den Objektcode

a.out bzw. keine Endung für das lauffähige Programm

MS-Windows:

.obj für den Objektcode

.exe für das lauffähige Programm

■ **Programmiersysteme: Compiler und Interpreter**

Compiler (Übersetzer) - Beispiele: Pascal, C, C++

- Übersetzung vor Ausführung in einem Durchlauf
- hohe Ausführungsgeschwindigkeit
- erschwerte Fehlersuche

Interpreter (Interpretierer) - BASIC, Perl, Python, LISP und Prolog

- Programmtext wird schrittweise zur Laufzeit übersetzt
- einfach zu realisieren, aber niedrigere Ausführungsgeschwindigkeit
- leichte Fehlersuche

Compreter - Java, .NET Sprachen

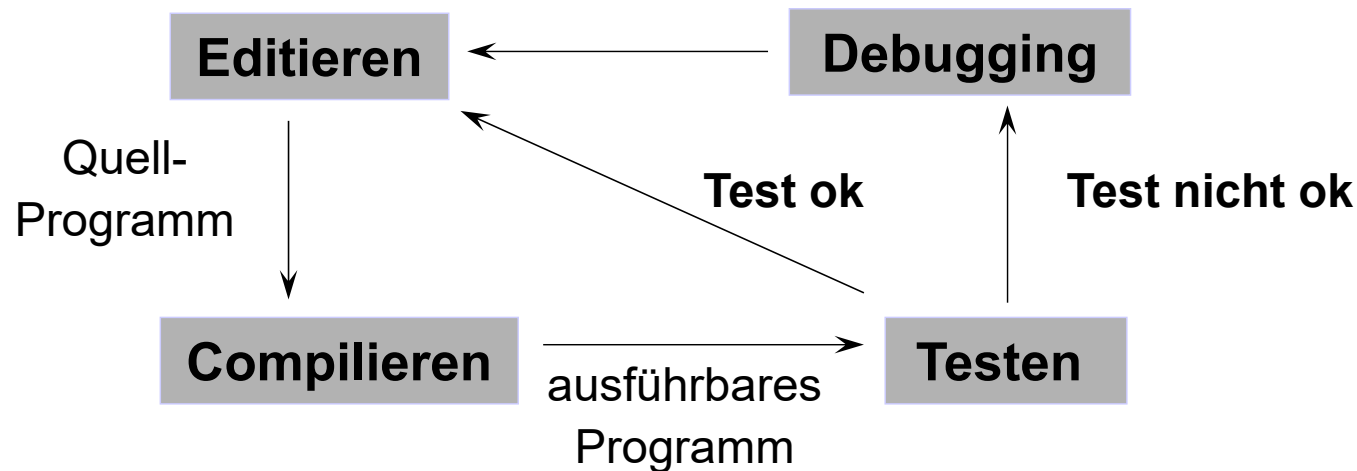
- Kombination beider Strategien
- Übersetzung vor Ausführung in Bytecode
- Interpretation des Bytecodes zur Laufzeit durch virtuelle Maschine

■ Compiler: C → Assembler/Maschinencode

			Die Darstellung des Hochsprachen-	
			zusammen mit dem Assembler- bzw.	
			Maschinencode und den Ladeadressen	
1:	#include <stdlib.h>			
2:	#include <stdio.h>			
3:				
4:	int main(void)			
5:	{			
00401010	55		push ebp	
00401011	8B EC		mov ebp,esp	
00401013	83 EC 08		sub esp,8	
6:	int v1, v2;			
7:				
8:	v1 = 1 + 2;			
00401016	C7 45 FC 03 00 00 00		mov dword ptr [v1],3	
9:	v2 = v1 + 3;			
0040101D	8B 45 FC		mov eax,dword ptr [v1]	
00401020	83 C0 03		add eax,3	
00401023	89 45 F8		mov dword ptr [v2],eax	

■ Programmierumgebung

- aufeinander abgestimmte Werkzeuge zur Programmentwicklung
 - Editor Erstellen und ändern eines Programmtextes
 - Compiler Übersetzen eines Programmtextes in ein äquivalentes Maschinenprogramm
 - Debugger Fehlersuche und -beseitigung

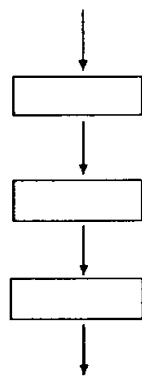


Ablaufsteuerung

■ Struktogramm

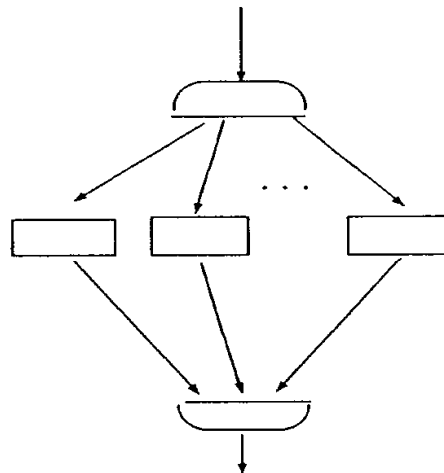
- Alternative zu Programmablaufplan (PAP)
- 1973 von Nassi und Shneiderman (→ Nassi-Shneiderman-Diagramm)
- DIN 66261
- basiert auf Prinzip der Strukturierten Programmierung:

Sequenz



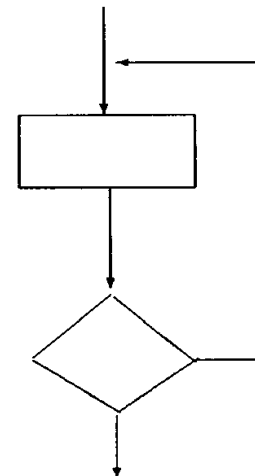
(a)

Selektion



(b)

Iteration



(c)

■ Verarbeitungsschritt

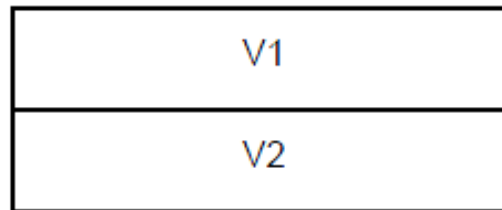
- Grundsymbol Rechteck $\triangleq$ 1 Verarbeitungsschritt



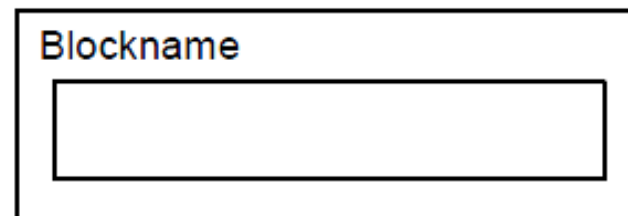
- entspricht einer Anweisung (oder Gruppe von Anweisungen)

■ Sequenz

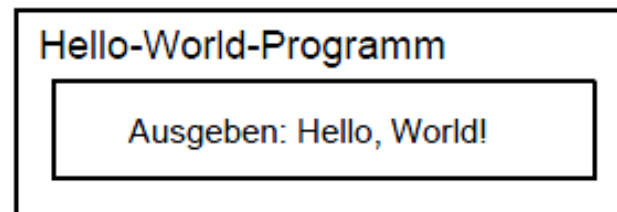
- 2 Verarbeitungsschritte V1 und V2:



- Block: Hauptprogramm, Unterprogramm oder zusammenhängende Verarbeitungsschritte

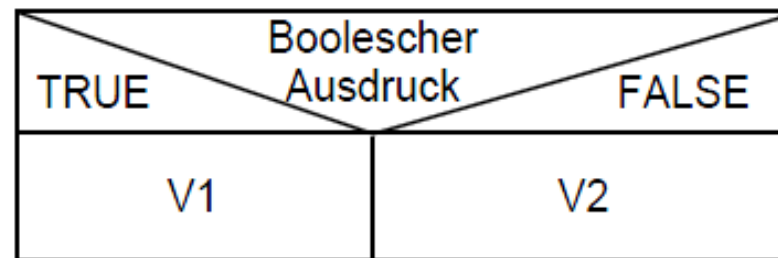


- Beispiel:



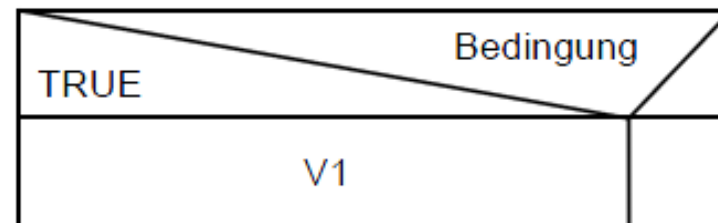
■ Selektion: **if, else, switch**

- einfache Alternative



```
if (a > b)
    V1
else
    V2
```

- jeder Zweig kann einen Verarbeitungsschritt bzw. Block enthalten
- bedingte Verarbeitung



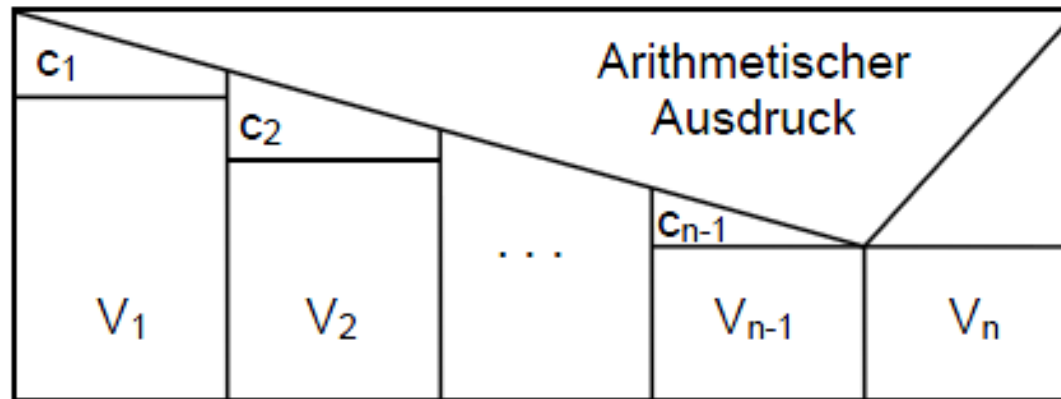
```
if (a > b)
    V1
```

		Ausdruck_1	
		wahr	falsch
Anweisung_1	Ausdruck_2		
	wahr	falsch	
	Ausdruck_3		
		wahr	falsch
	Anweisung_2	Anweisung_3	Anweisung_else

```

if (Ausdruck_1)
    Anweisung_1
else if (Ausdruck_2)
    Anweisung_2
else if (Ausdruck_3)
    Anweisung_3
    .
    .
    .
else if (Ausdruck_n)
    Anweisung_n
else
    Anweisung_else

```



```

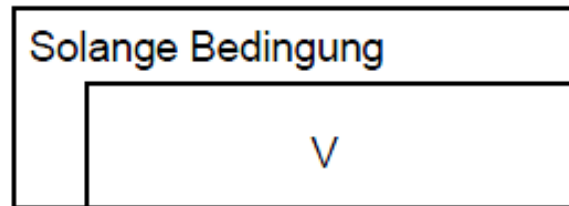
switch (Arithmetischer Ausdruck)
{
    case c1:
        V1;
        break;
    case c2:
        V2;
        break;

    ...

    case cn-1:
        Vn-1;
        break;
    default:
        Vn;
}
    
```

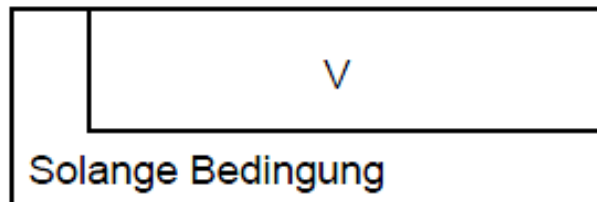
■ Iteration **while**, **do while**

- Wiederholung mit vorheriger Prüfung (abweisende Schleife):



```
while (Bedingung)  
    V;
```

- Wiederholung mit nachfolgender Prüfung (annehmende Schleife):



```
do  
    V;  
while (Bedingung);
```

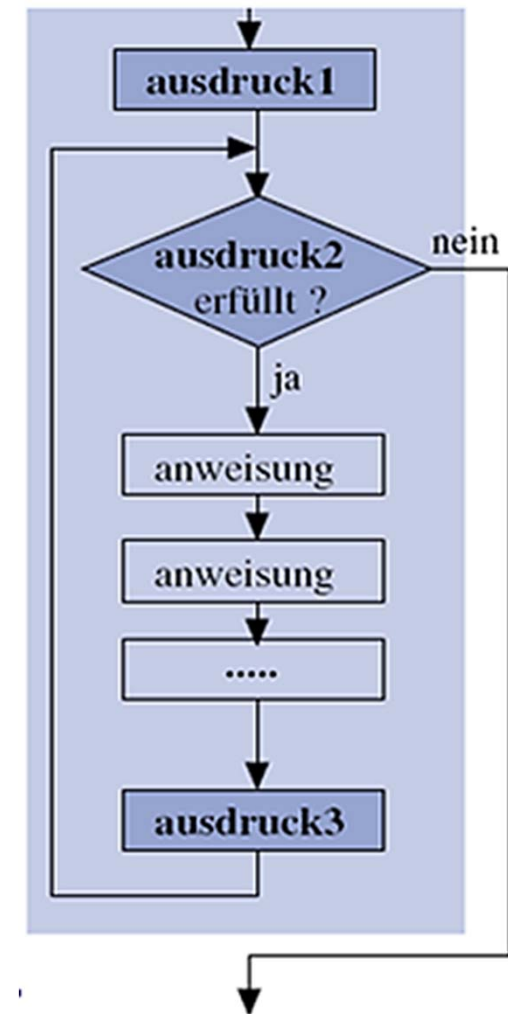
- Endlos-Schleifen!

■ Iteration for



```

for ( ausdruck1; ausdruck2; ausdruck3 )
{
    anweisung;
    anweisung;
    ...
}
    
```



■ Iteration for vs. while

```
for ( Initialisierung ; Bedingung ; Schleifenanweisung )  
{  
    anweisung;  
    anweisung;  
}
```

```
Initialisierung;  
while (Bedingung)  
{  
    anweisung;  
    anweisung;  
    Schleifenanweisung;  
}
```

■ Iteration for vs. while

```
for ( i = 0 ; i <= 10 ; i = i + 1 )
{
    printf ("i = %d \n", i);
}
```

```
_____;
while (_____)
{
    printf ("i = %d\n", i);
    ____;
}
```

■ Iteration for vs. while

```
for ( i = 0 ; i <= 10 ; i = i + 1 )  
{  
    printf ("i = %d \n", i);  
}
```

```
i = 0 ;  
while (i <= 10)  
{  
    printf ("i = %d\n", i);  
    i = i + 1;  
}
```

■ Iteration for vs. while

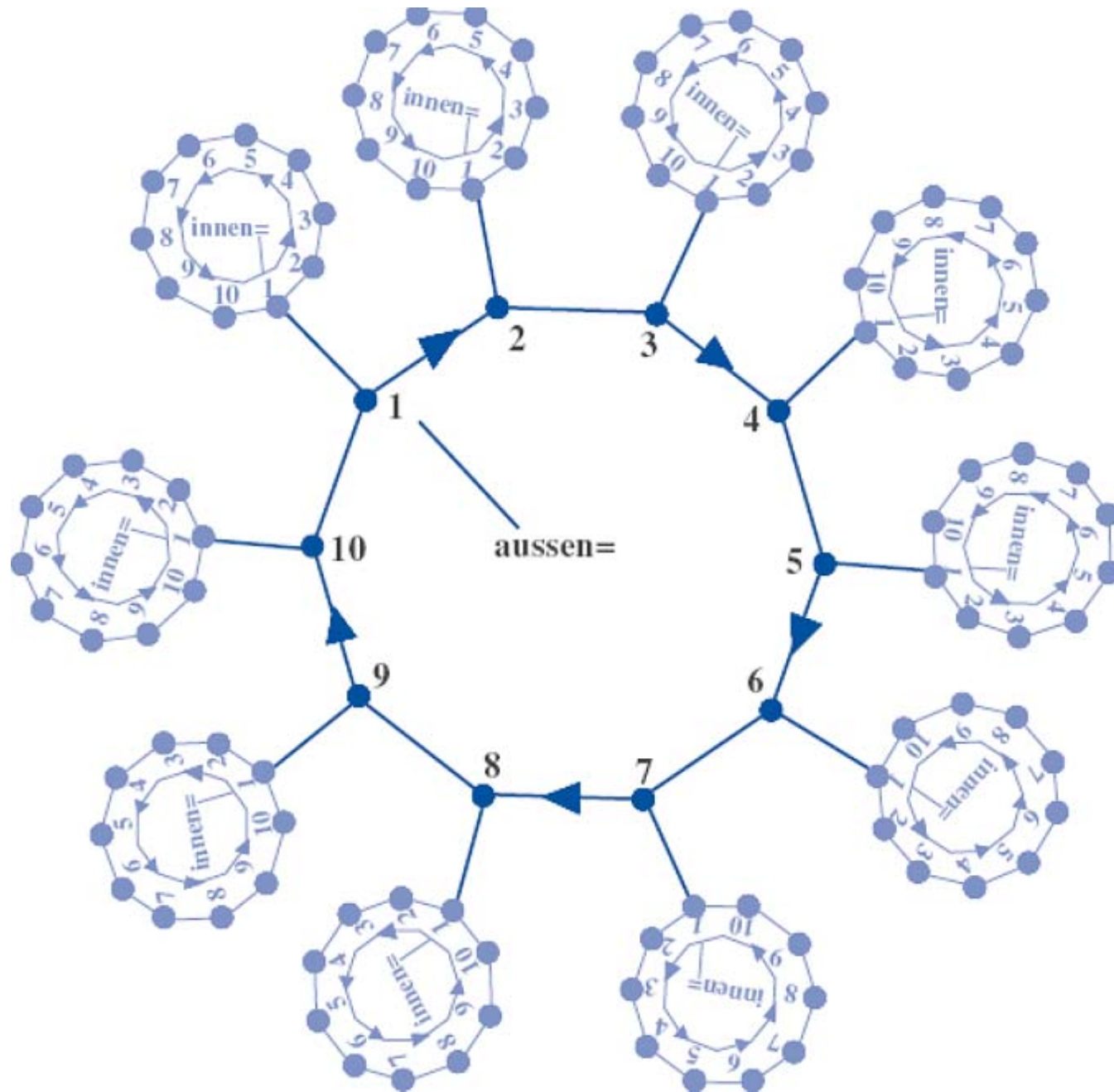
```
for ( i = 0 ; i <= 10 ; i = i + 1 )  
    printf ("i = %d \n", i);
```

```
i = 0 ;  
while (i <= 10)  
{  
    printf ("i = %d\n", i);  
    i = i + 1;  
}
```

■ Beispiel: Iteration

Entwerfen Sie ein Struktogramm und dann ein Konstrukt in C-Code, welches das Einmaleins in folgender Form ausgibt:

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100



■ Iteration: **break/continue**

break Verlassen der **gesamten** Schleifenanweisung

continue Abbruch **eines** Schleifendurchlaufs

Beispiel: Was gibt das folgende Programm aus?

Hinweis: Das "%" -Zeichen ist der Modulo-Operator (Rest der Ganzzahl-Division)

```
int main()  
{  
    int i;  
  
    for (i = 1; i <= 20; i++)  
    {  
        if (i % 2 != 0)  
            continue;  
        printf("%d ", i);  
    }  
    return 0;  
}
```

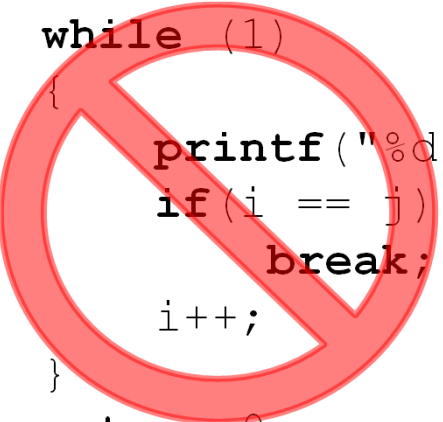
■ Iteration: break/continue

Beispiel: Was gibt das folgende Programm aus?

```
int main()
{
    int i = 0;
    int j;

    printf ("Bitte eine positive Ganzzahl eingeben: ");
    scanf ("%d", &j);

    while (1)
    {
        printf ("%d ", i);
        if (i == j)
            break;
        i++;
    }
    return 0;
}
```



■ Iteration: break/continue

Beispiel: Was gibt das folgende Programm aus?

```
int main()
{
    int i = 0;
    int j;

    printf ("Bitte eine positive Ganzzahl eingeben: ");
    scanf ("%d", &j);

    for (i = 0; i <= j; i++)
        printf("%d ", i);

    return 0;
}
```

■ Euklidischer Algorithmus

- Anwendung:

$$\frac{X_{\text{ungekürzt}}}{Y_{\text{ungekürzt}}} = \quad \longrightarrow \quad = \frac{X_{\text{gekürzt}}}{Y_{\text{gekürzt}}}$$

■ Euklidischer Algorithmus

- Anwendung:

$$\frac{x_{\text{ungekürzt}}}{y_{\text{ungekürzt}}} = \frac{x_{\text{ungekürzt}} / \text{ggT}(x_{\text{ungekürzt}}, y_{\text{ungekürzt}})}{y_{\text{ungekürzt}} / \text{ggT}(x_{\text{ungekürzt}}, y_{\text{ungekürzt}})} = \frac{x_{\text{gekürzt}}}{y_{\text{gekürzt}}}$$

- Algorithmus:

Solange x ungleich y ist, wiederhole:

Wenn x größer als y ist, dann:

ziehe y von x ab und weise das Ergebnis x zu.

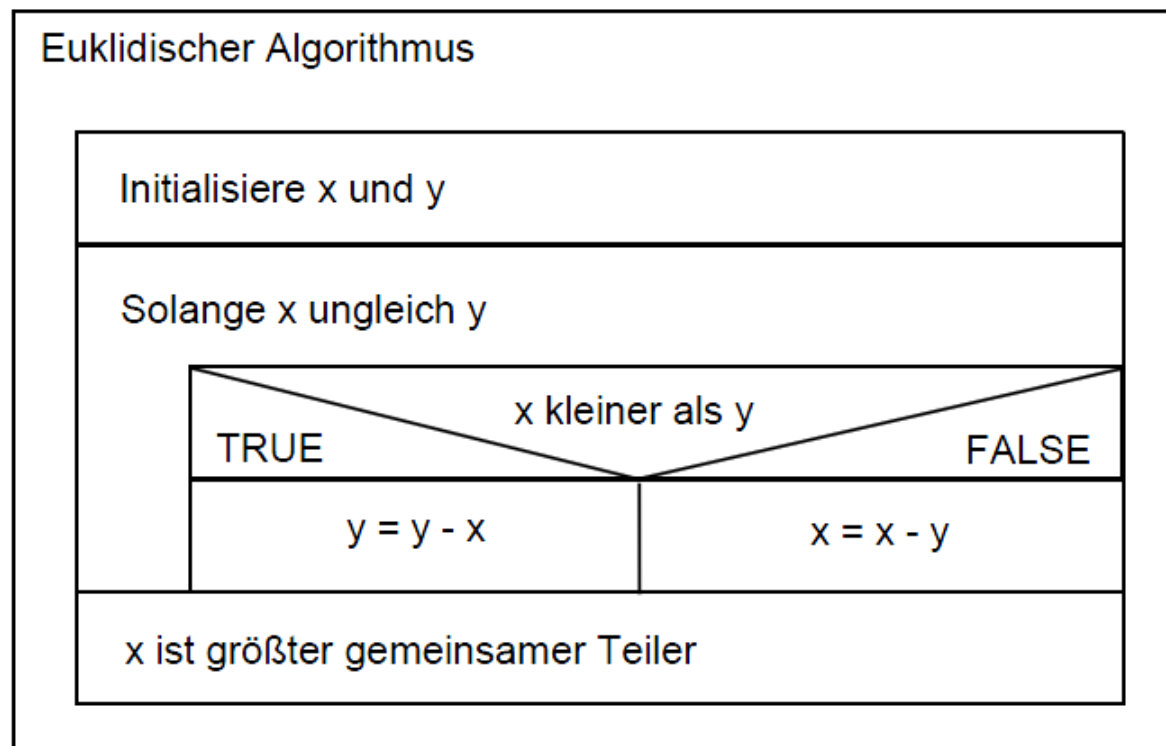
Andernfalls:

ziehe x von y ab und weise das Ergebnis y zu.

Wenn x gleich y ist, dann:

x (bzw. y) ist der gesuchte größte gemeinsame Teiler.

Solange x ungleich y ist, wiederhole:
 Wenn x größer als y ist, dann:
 ziehe y von x ab und weise das Ergebnis x zu.
 Andernfalls:
 ziehe x von y ab und weise das Ergebnis y zu.
 Wenn x gleich y ist, dann:
 x (bzw. y) ist der gesuchte größte gemeinsame Teiler.



■ Euklid Trace-Tabelle

- eine **Trace-Tabelle** zeigt die Zustände der Variablen im Ablauf
- Achtung: Das Gleichheitszeichen ist der Zuweisungsoperator in C!

Verarbeitungsschritt	Werte von	
	x	y
Initialisierung <code>x = 24, y = 9</code> <code>x = x - y</code>	24	9
Ergebnis: ggT =		

Verarbeitungsschritt	Werte von	
	x	y
Initialisierung		
$x = 24, y = 9$	24	9
$x = x - y$	15	9
$x = x - y$	6	9
$y = y - x$	6	3
$x = x - y$	3	3
Ergebnis: ggT = 3		

Verarbeitungsschritt	Werte von	
	x	y
Initialisierung		
$x = 5, y = 3$	5	3
$x = x - y$	2	3
$y = y - x$	2	1
$x = x - y$	1	1
Ergebnis: ggT = 1		

■ Euklid C-Programm

 Übung

■ Aufgabe: Quadratzahlen

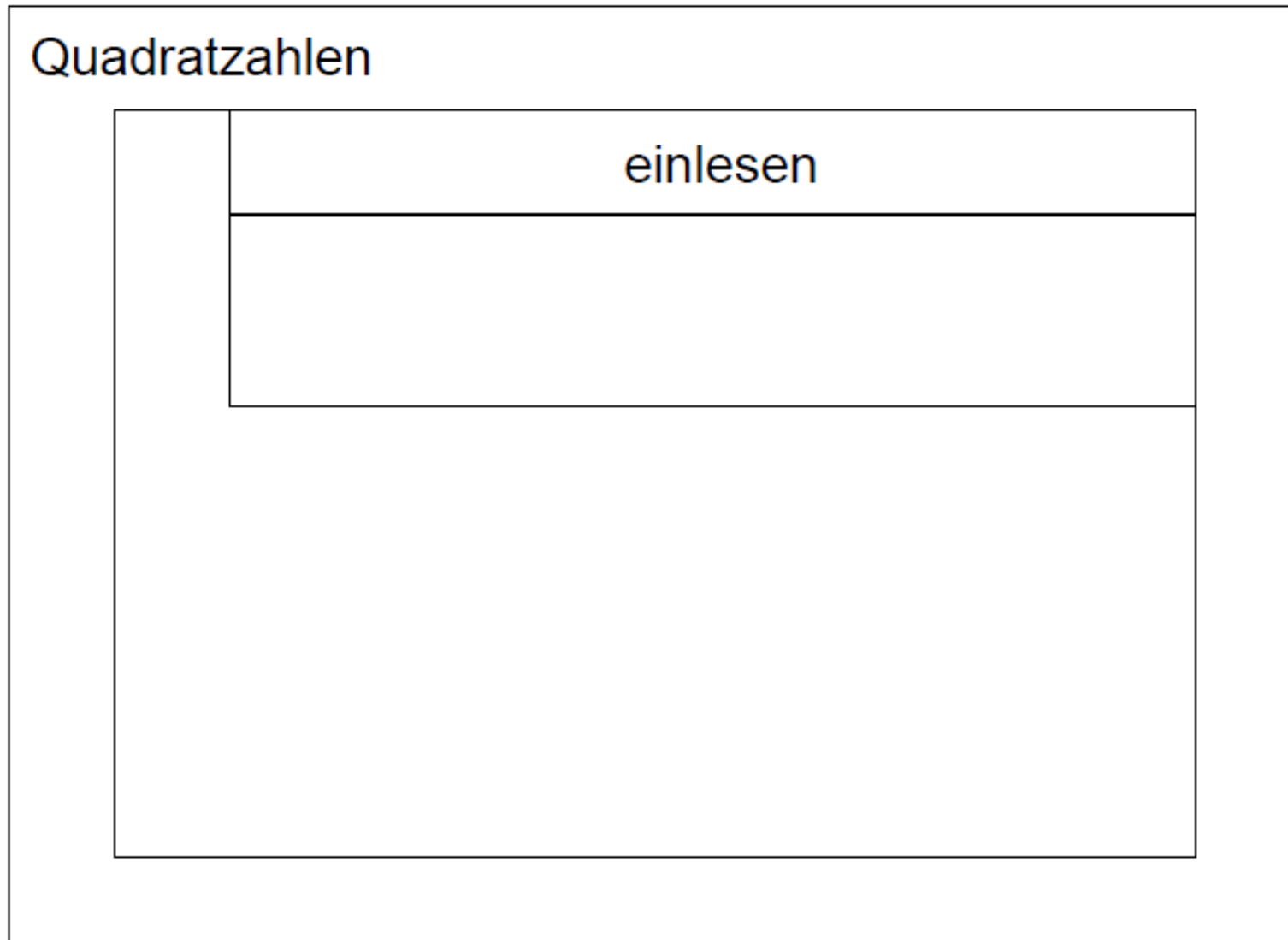
Vervollständigen Sie das auf der folgenden Folie gegebene Struktogramm für ein Programm, welches in einer (äußeren) Schleife ganze Zahlen in eine Variable n einliest. Die Reaktion des Programms soll davon abhängen, ob der in die Variable eingelesene Wert positiv, negativ oder gleich Null ist. Treffen Sie die folgende Fallunterscheidung:

- Ist die eingelesene Zahl n größer als Null, so soll in einer inneren Schleife ausgegeben werden:

Zahl	Quadratzahl
1	1
2	4
.	.
.	.
.	.
n	$n*n$

- Ist die eingelesene Zahl n kleiner als Null, so soll ausgegeben werden: Negative Zahl
- Ist die eingegebene ganze Zahl n gleich Null, so soll das Programm (die äußere Schleife) abbrechen.

■ Aufgabe: Quadratzahlen



Datentypen

■ ANSI C89: Datentypen

Typ	Wertebereich (Genauigkeit)	Größe	E / A
int	- 32 768 ... 32 767 bei 16 Bit Maschinen - 2 147 483 648 ... 2 147 483 647 32 Bit	2 Byte 4 Byte	%d
unsigned int	0 ... 65 535 bei 16 Bit Maschinen 0 ... 4 294 967 295 bei 32 Bit Maschinen	2 Byte 4 Byte	%u
short int	- 32 768 ... 32 767	2 Byte	%d
unsigned short int	0 ... 65 535	2 Byte	%u
long int	- 2 147 483 648 ... 2 147 483 647	4 Byte	%ld
unsigned long int	0... 4 294 967 295	4 Byte	%lu
char	alle Zeichen im ASCII Code	1 Byte	%c
char	-128 ... 127	1 Byte	%d
unsigned char	0 ... 255	1 Byte	%u
float	1,2 E-38 ... 3,4 E+38 (6 Stellen)	4 Byte	%f
double	2,3 E-308 ... 1,7 E+ 308 (15 Stellen)	8 Byte	%lf
long double	3.4 E-4932 ... 1.1 E+4932 (19 Stellen)	10 Byte	%lf
void	leerer Typ	0 Byte	

■ 64-Bit Architektur: C-Datenmodelle

Datenmodell	short int	int	long int	long long	Zeiger	Beispiel- Betriebssystem
LLP64	16	32	32	64	64	Microsoft Win64 (X64/IA64)
LP64	16	32	64	64	64	Unix-Systeme (z. B. Solaris) und unixoide Systeme (z. B. Linux)
ILP64	16	64	64	64	64	Cray, DEC/Alpha mit Tru64, DEC/Alpha mit Linux

■ ANSI C99: Erweiterungen

Typ	Wertebereich	Größe
_Bool	0 und 1	1 Byte
long long	−9223372036854775808 bis 9223372036854775807	8 Byte
unsigned long long	0 bis 18446744073709551615	8 Byte

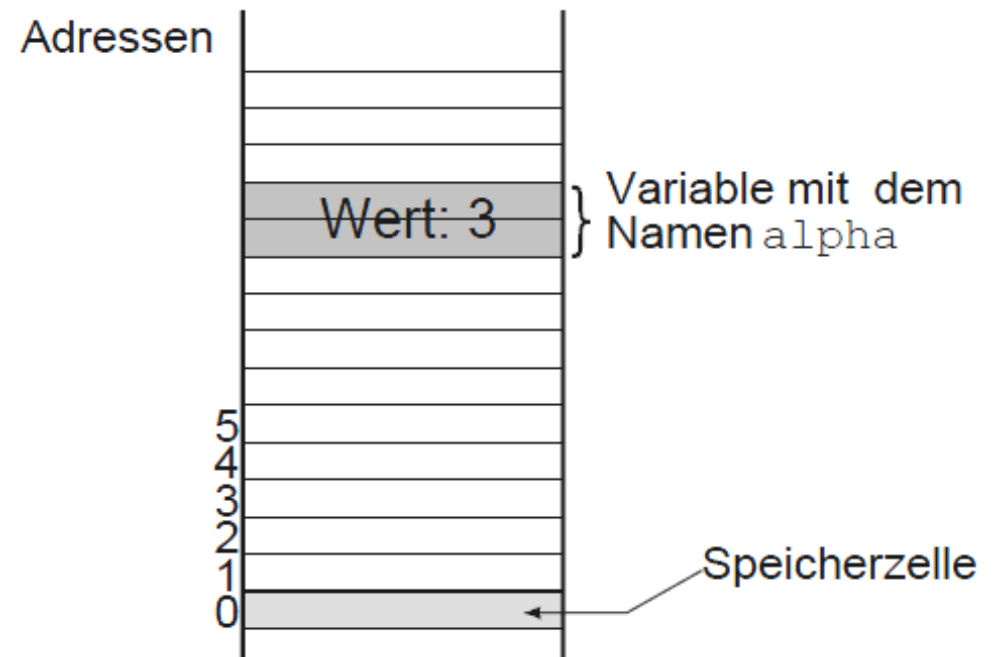
■ Datentypen

- Datentyp = "Bauplan" für eine Variable
- legt fest:
 - Bedeutung
 - zulässige Operationen
 - Repräsentation im Speicher
- einfache Datentypen (atomar)
- Standarddatentypen (einer Sprache)
- selbst definierte Typen (**struct**, **enum**)

■ Variablen

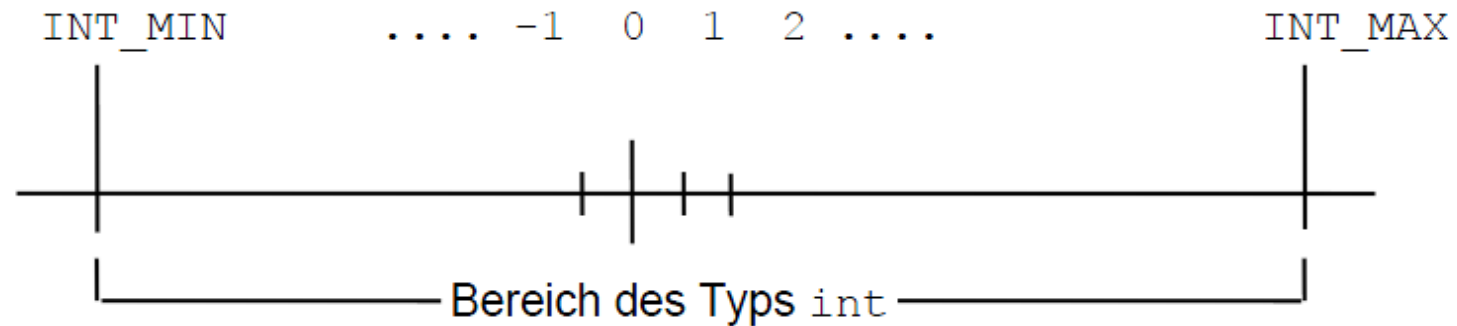
- Variable = benannte Speicherstelle
- Variablenname ermöglicht Zugriff auf Speicherstelle
- rechnerinterne Darstellung:

- 4 Kennzeichen
 - Variablenname
 - Datentyp
 - Wert
 - Adresse



■ Datentyp `int`

- ganze Zahlen (Integer)
- endlicher Zahlenbereich



- Bereichsüberlauf – Was tun?
 Immer den größten Wertebereich nehmen?
 oder Fließkomma-Datentypen?

■ Operationen auf einfachen Typen (Beispiel `int`)

Operator	Operanden	Ergebnis
Vorzeichenoperationen +, - (unär)	int	→ int
binäre arithmetische Operationen +, -, *, /, %	(int, int)	→ int
Vergleichsoperationen ==, <, <=, >, >=, !=	(int, int)	→ int (Wahrheitswert)
Wertzuweisungsoperator =	(int, int)	→ int

■ Datentypen float und double

- Fließkomma-Datentyp (*floating point numbers*)
- für rationale und reellen Zahlen
- im Rechner ist Speicher endlich → begrenzte Genauigkeit
- Speicherung als Exponentialzahlen in der Form

Mantisse * Basis^{Exponent}

- Mantisse und Exponent ganzzahlig
- keine exakte Darstellung von nicht-rationalen Zahlen (z.B. $\sqrt{2}$)
- Rundungsfehler!