Das erste C-Programm

- **Gliederung**

  Sprache C

  Ablaufsteuerung

  Datentypen

  Unterprogramme

  Übersetzung

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

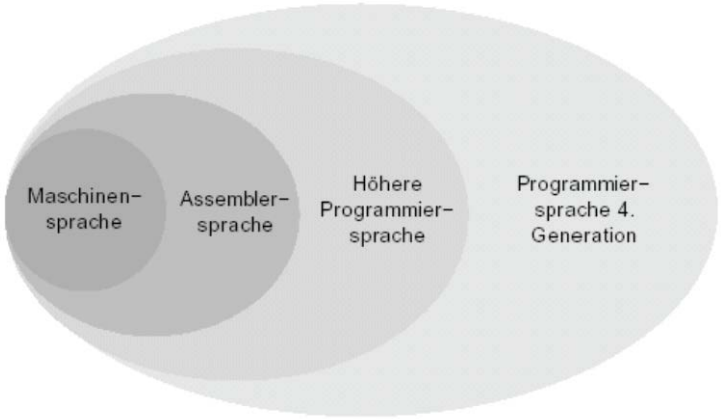
- **K&R-C** von den Erfindern Kernighan und Ritchie
- **ANSI-C** standardisierte Normen **C89** C95 **C99** **C11** C18
- **C++** objektorientierte Erweiterungen (Bjarne Stroustrup)
- Dialekte sind abwärtskompatibel

Eigenschaften der Sprache C

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

  Konstrukte höherer Sprachen:
  Ablaufsteuerung,
  Datentypenkonzept (nicht streng),
  Unterprogrammtechnik

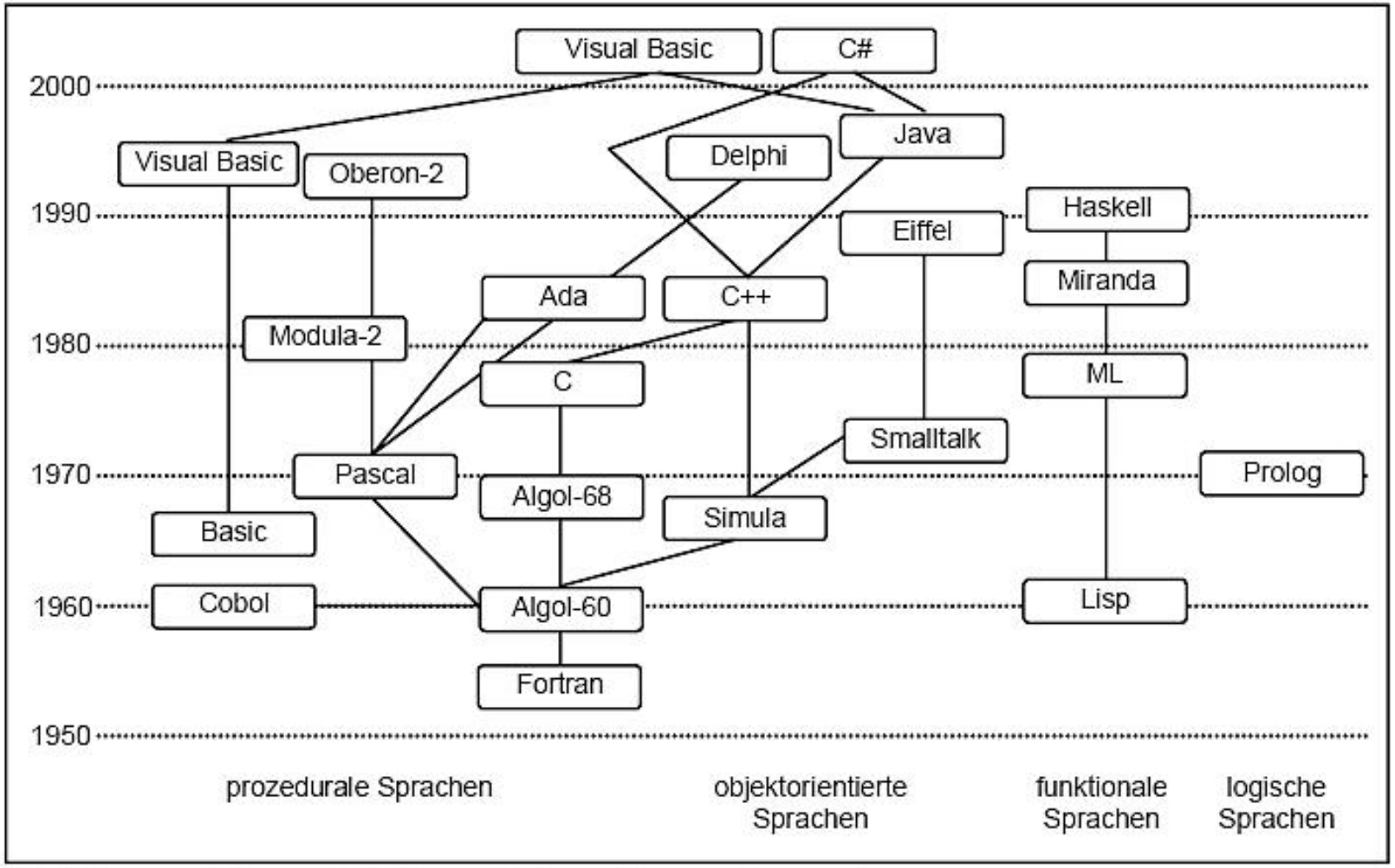
  Maschinennahe Sprachkonstrukte:
  Zeiger,
  Adressarithmetik,
  direkte Bit-Manipulation



- 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)
- Portabilität

Warum C?

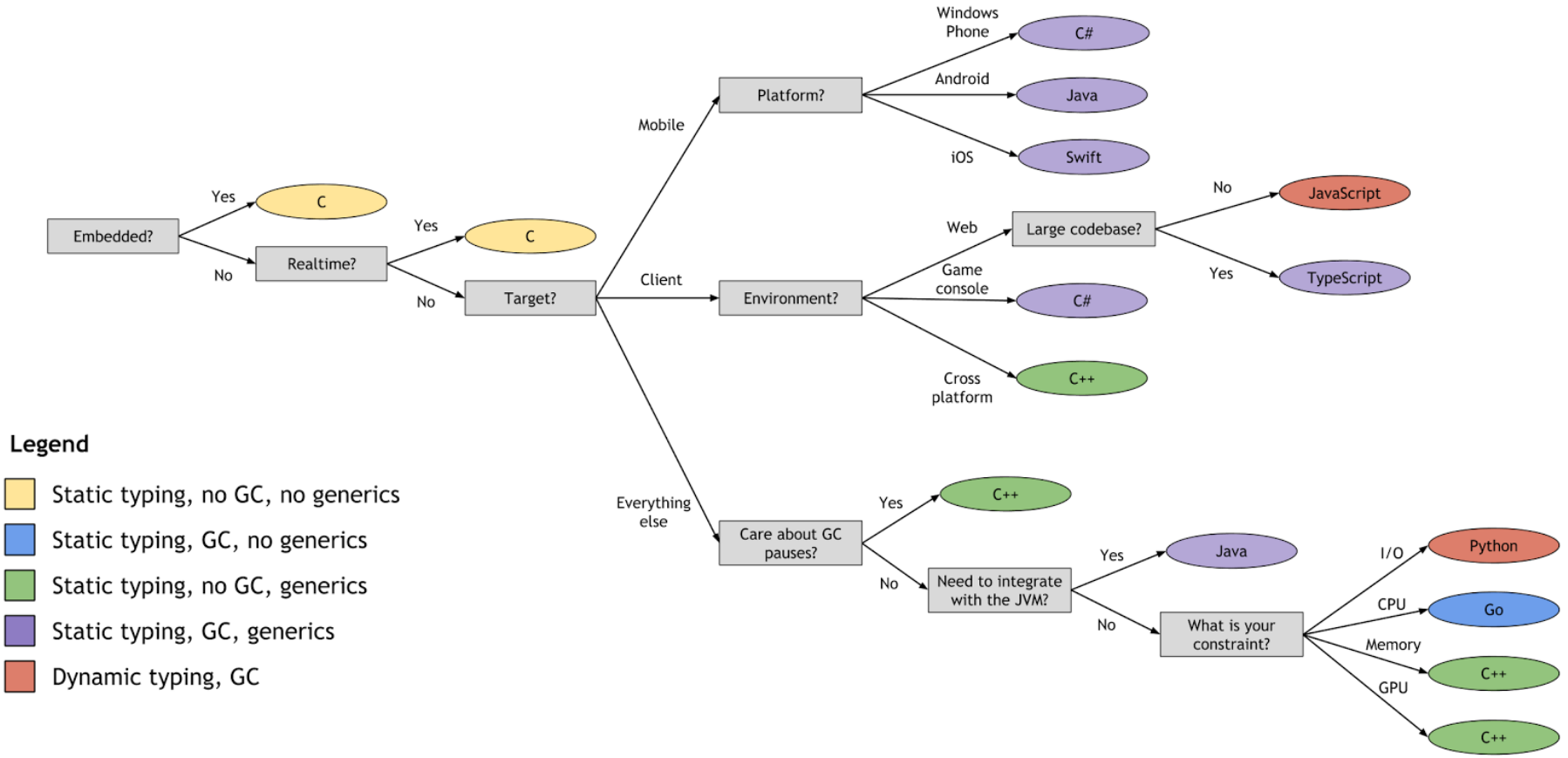
- mächtig und flexibel
- bedient Bedarf nach Leistung, Maschinennähe, Ressourcenknappheit
- Grundkonzepte und Syntax sind Basis vieler Sprachen
- hoher Verbreitungsgrad → viele Bibliotheken
- Aktuelle Popularität: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html



Entwicklung und Verwandtschaft verschiedener Programmiersprachen

Which programming language should I use for my project?

by onebigfluke.com



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;
}
```

Ablaufsteuerung

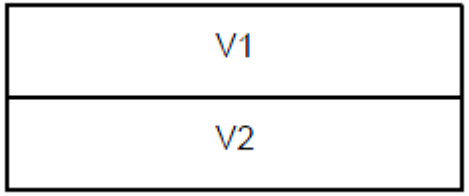
Struktogramme

- 1973 von Nassi und Shneiderman (→ Nassi-Shneiderman-Diagramme)
- DIN 66261
- Strukturierte Programmierung: Sequenz, Iteration, Selektion
- Grundsymbol Rechteck ≙ 1 Verarbeitungsschritt

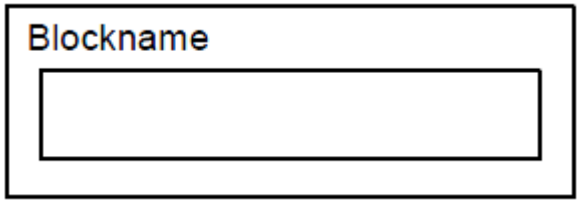
- entspricht einer Anweisung (oder Gruppe von Anweisungen)

Sequenz

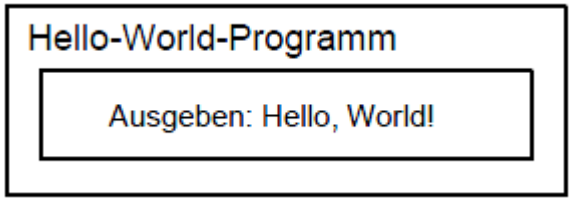
- 2 Verarbeitungsschritte V1 und V2:



- Block: Hauptprogramm, Unterprogramm oder zusammenhängende Verarbeitungsschritte

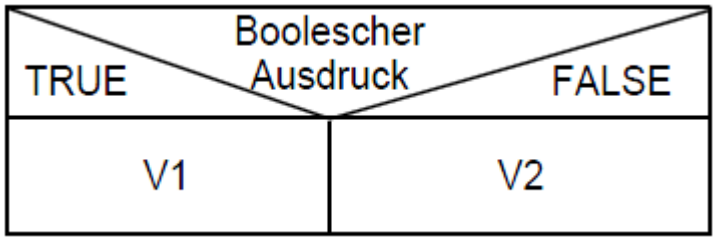


- Beispiel:



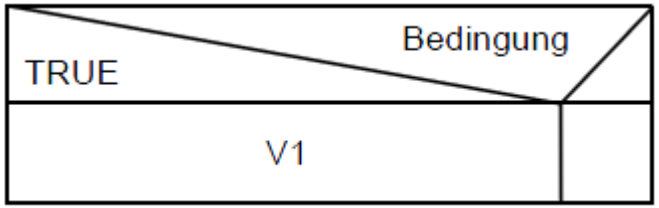
Selektion: `if, else, switch`

- einfache Alternative

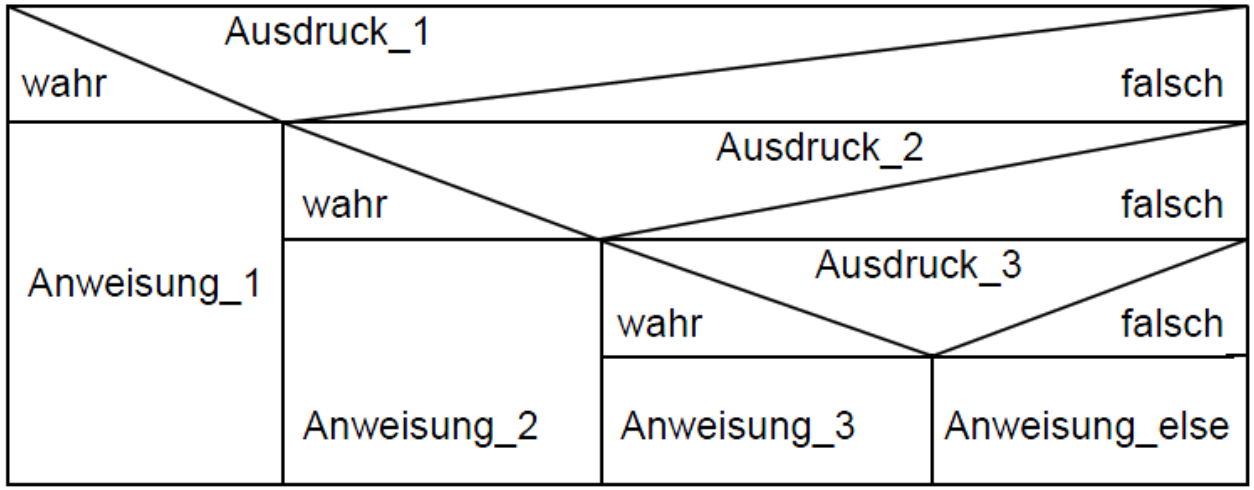



```
if (a > b) V1
else V2
```

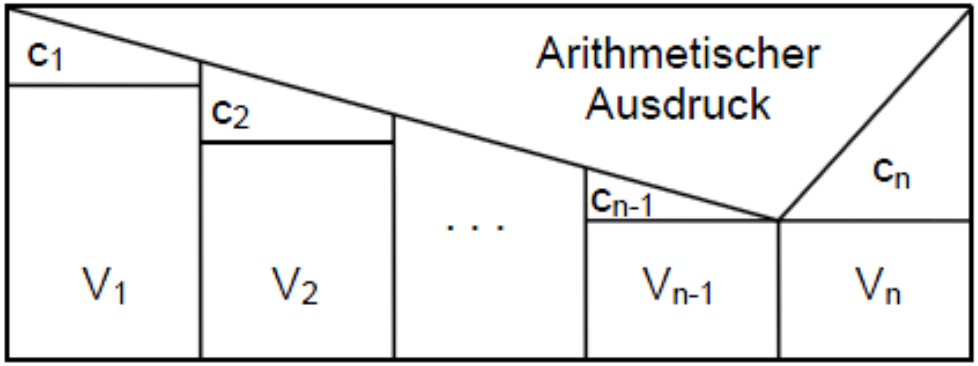
- jeder Zweig kann einen Verarbeitungsschritt bzw. Block enthalten
- bedingte Verarbeitung




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



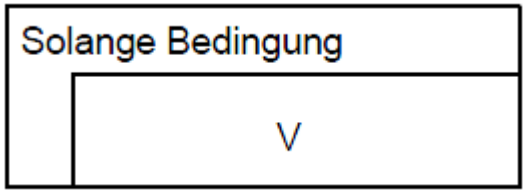

```
if (Ausdruck_1)
   Anweisung_1
else if (Ausdruck_2)
   Anweisung_2
else if (Ausdruck_3)
   Anweisung_3
      .
      .
      .
else if (Ausdruck_n)
   Anweisung_n
else
   Anweisung_else
```




```
switch (Ausdruck)
{
   case k1:
      Anweisungen_1
      break;                          /* ist optional        */
   case k2:
      Anweisungen_2
      break;                          /* ist optional        */
     .
     .
     .
   case kn:
      Anweisungen_n
      break;                          /* ist optional        */
   default:                           /* ist                 */
      Anweisungen_default             /* optional            */
}
```

Iteration `while, do while`

- Wiederholung mit vorheriger Prüfung (abweisende Schleife):




```
while (Bedingung) V
```

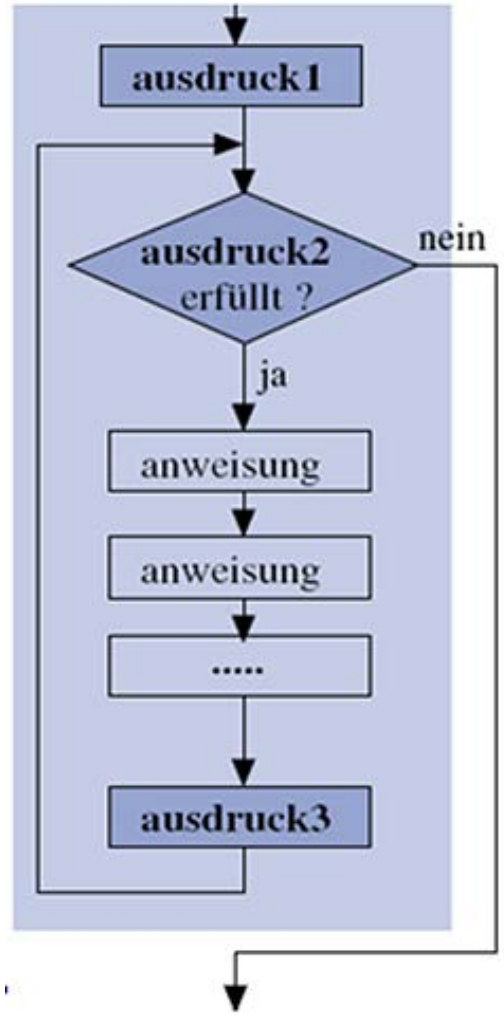
- Wiederholung mit nachfolgender Prüfung (annehmende Schleife):


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

- Endlos-Schleifen!

Iteration `for`

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



Iteration `for`

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

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

Iteration for

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

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

Iteration for

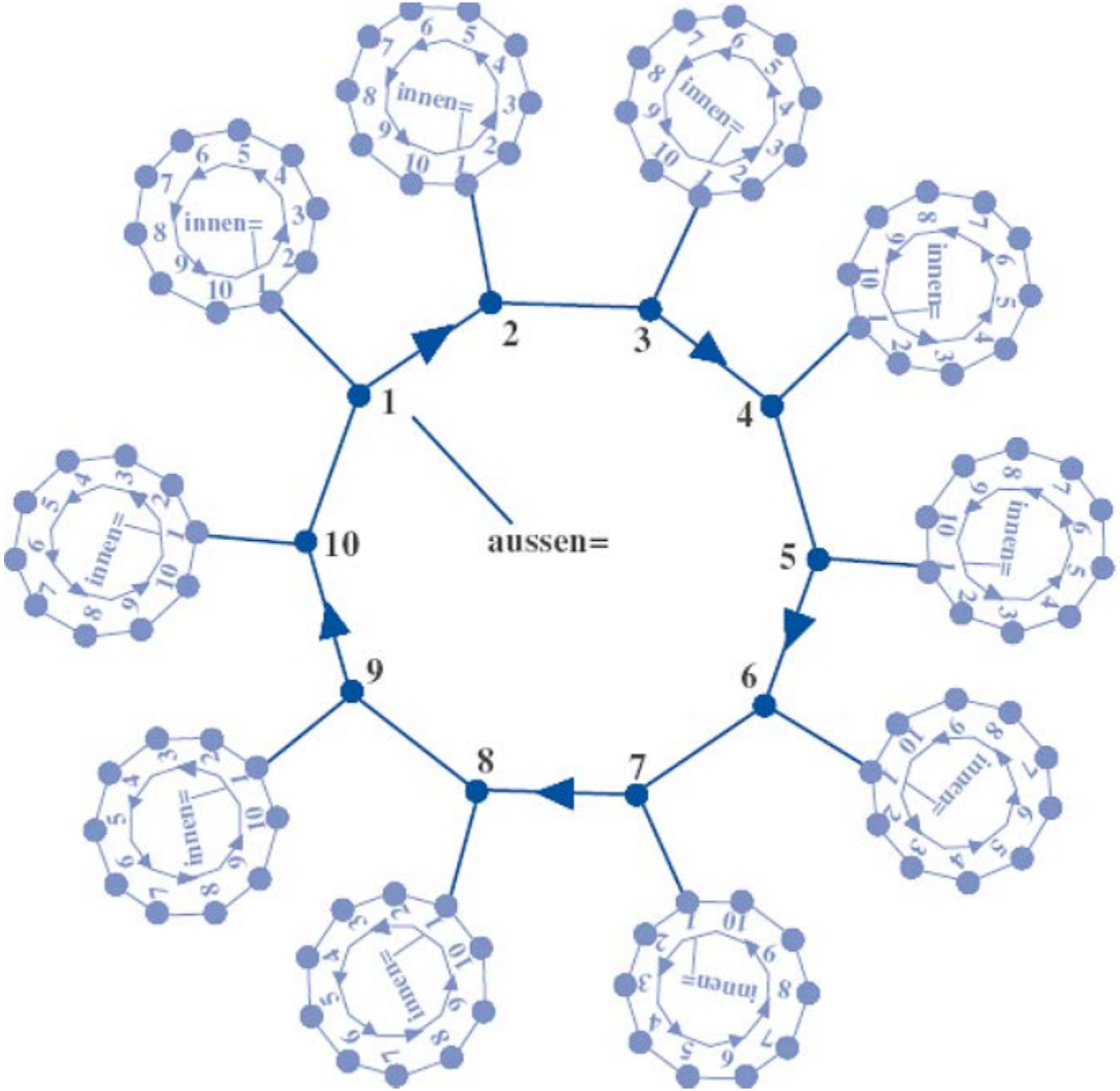
```
for ( i = 0 ; i <= 10 ; i = i + 1 )
  printf ("i = %d \n", i);
```

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

Beispiel: Iteration

Entwerfen Sie ein 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
```


1
2
3
4
5
6
7
8
9
10
aussen=
innen=

Iteration `break / continue`

continue	Abbruch **eines** Schleifendurchlaufs
break	Verlassen der **gesamten** Schleifenanweisung

Beispiel: Was gibt das folgende Programm aus? (Das %-Zeichen ist der Modulo-Operator in C, liefert den 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, j = 0;

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

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

Euklidscher Algorithmus

- Anwendung:

$$\frac{x_{\text{ungekürzt}}}{y_{\text{ungekürzt}}} = \quad \Rightarrow \quad = \frac{x_{\text{gekürzt}}}{y_{\text{gekürzt}}}$$

Euklidscher Algorithmus

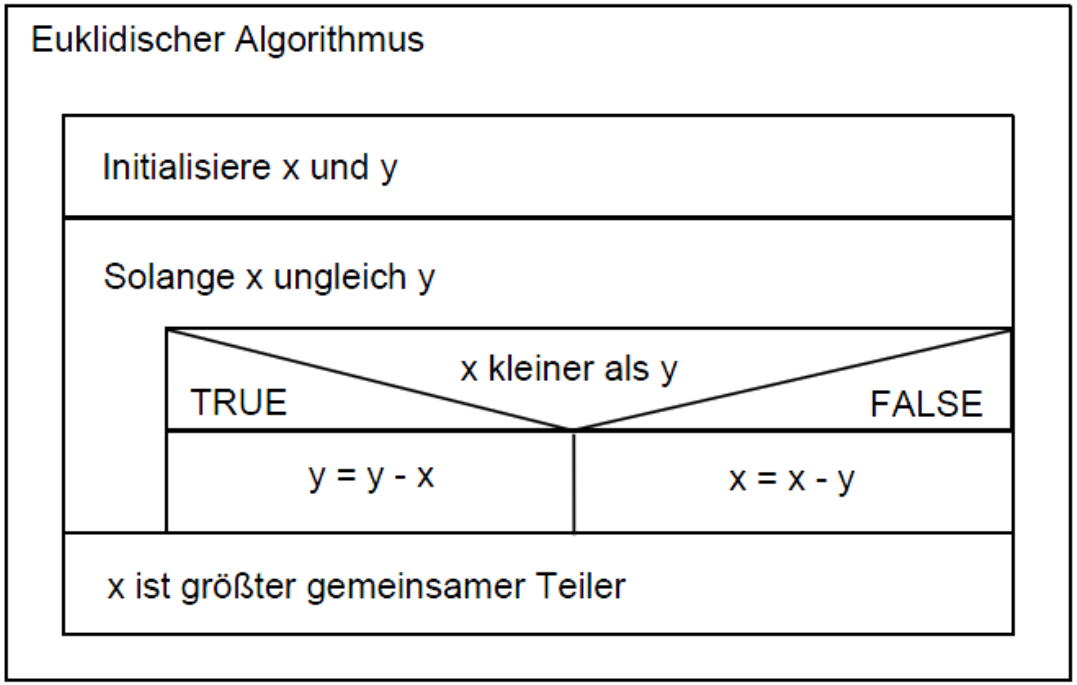
- Anwendung:

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

- 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 in Pseudocode

```
Euklidischer Algorithmus
{
   Initialisiere x und y
   while (x ungleich y)
   {
      if (x kleiner als y)
         y = y - x
      else
         x = x - y
   }
   x ist der größte gemeinsame Teiler
}
```

Euklidscher Algorithmus

- **Trace-Tabelle** zeigt Funktion der Variablen
- Zuweisungssymbol ist das Gleichheitszeichen (C-Syntax)

Verarbeitungsschritt	Werte von x	Werte von y

Initialisierung
`x = 24, y = 9`
`x = x - y`
	24	9

Ergebnis: ggT = 3		

Verarbeitungsschritt	Werte von x	Werte von 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	Werte von 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 als C-Programm

Aufgabe Geldautomat

Erstellen Sie für den in Pseudocode gegebenen Algorithmus ein Struktogramm:

```
begin Geldautomat
  while Geldautomat bereit
  do
    Karte einführen;
    if Karte lesbar
    then
      Fehlversuche := 0;
      while Fehlversuche < 3
      do
        Einlesen (PIN);
        if (PIN = PIN der Karte)then
          Eingabe Geldbetrag;
          Geld auswerfen;
        else
              Fehlversuche := Fehlversuche + 1;
        end if
      end do
    end if
    Geldkarte auswerfen;
  end do
end Geldautomat
```

Aufgabe Quadratzahlen

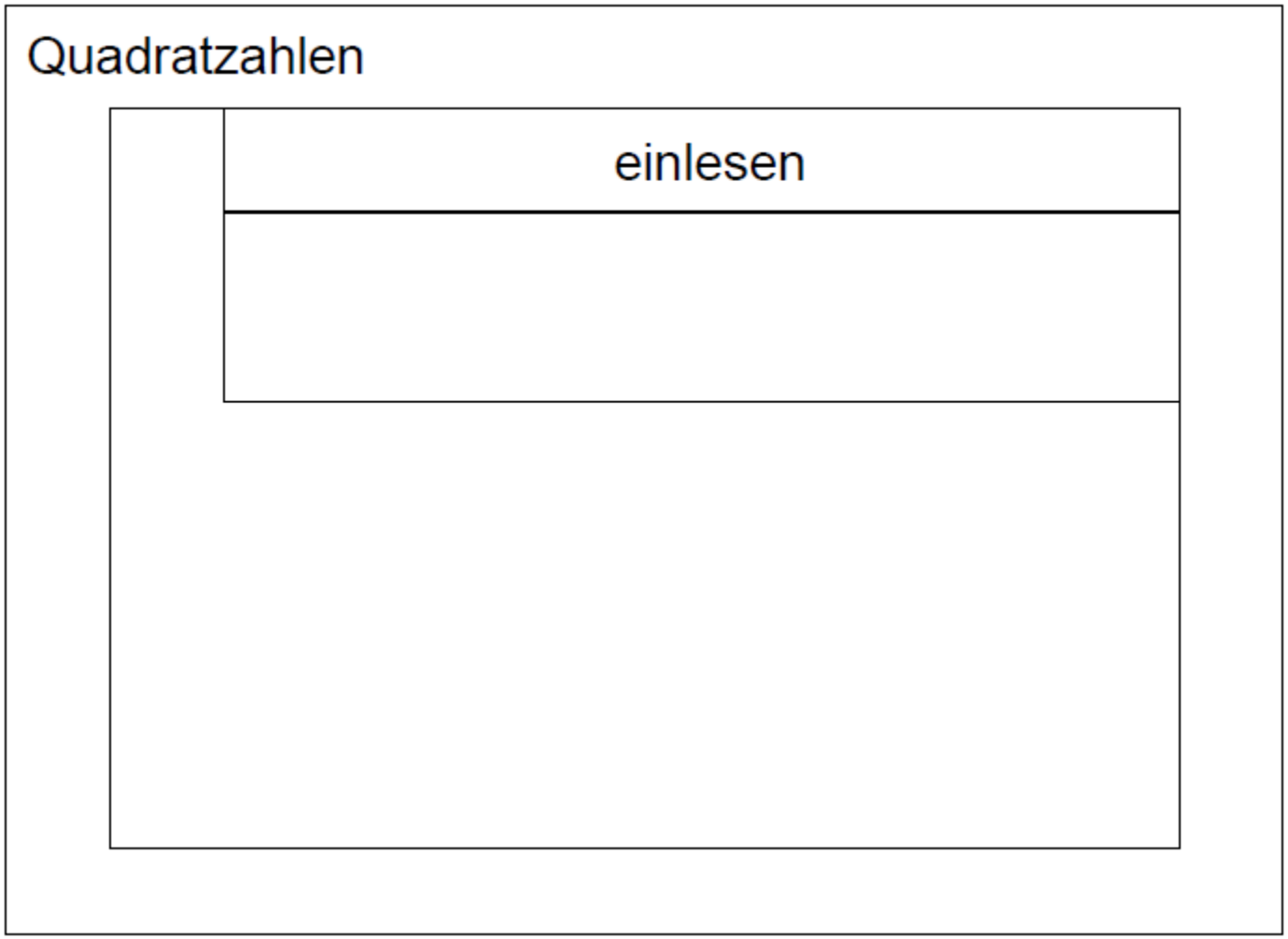
Vervollständigen Sie das unten angegebene Struktogramm für ein Programm, welches in einer (äußeren) Schleife ganze Zahlen in eine Variable n einliest. Die Reaktion des Programms soll davon abhängen, ob der in die Variable eingelesene Wert positiv, negativ oder gleich Null ist. Treffen Sie die folgende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

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

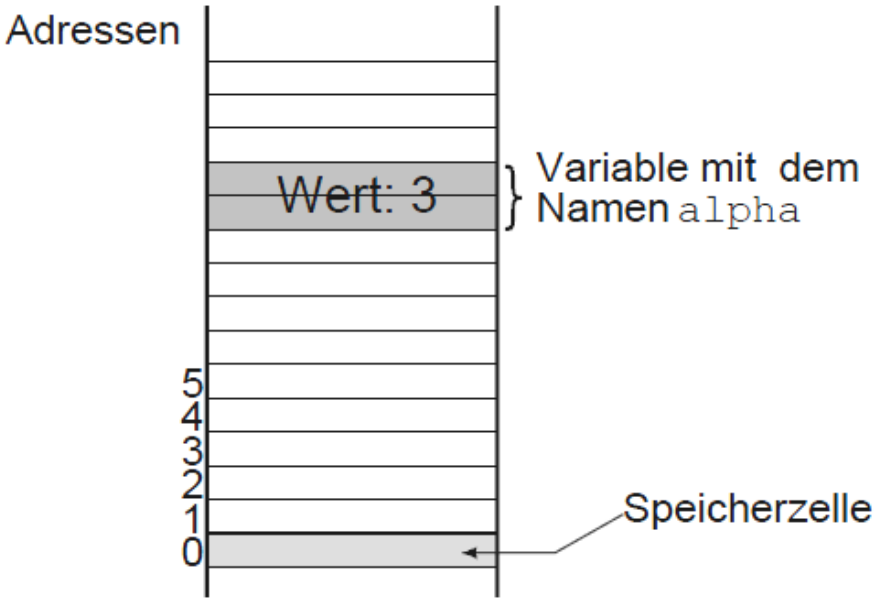
Datentypen

- Datentyp = "Bauplan" für eine Variable
- legt fest:
  - Bedeutung
  - zulässige Operationen
  - Repräsentation im Speicher

- einfache Datentypen (atomar)
- Standarddatentypen (einer Sprache)
- selbst definierte (`struct, enum`)

Variablen

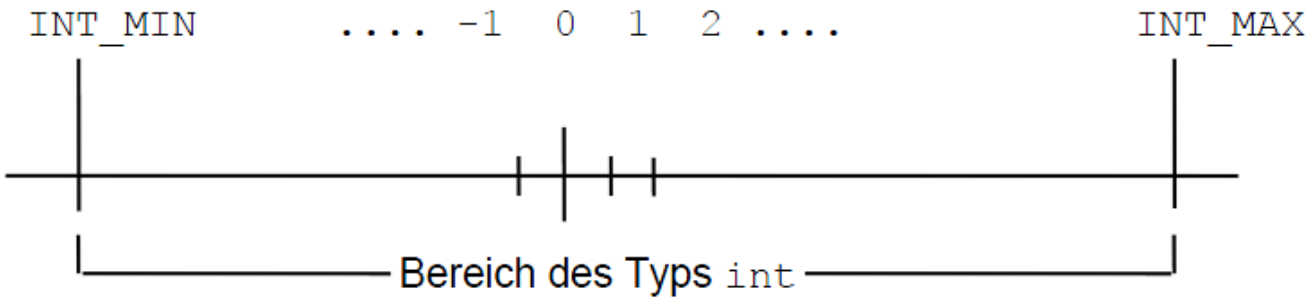
- Variable = benannte Speicherstelle
- Variablenname ermöglicht Zugriff auf Speicherstelle
- vgl. Konstanten
- Initialisierung
- rechnerinterne Darstellung:



- 4 Kennzeichen
  - Variablenname
  - Datentyp
  - Wert
  - Adresse

Datentyp `int`

- ganze Zahlen (Integer)
- endlicher Zahlenbereich



- Zahlenüberlauf – Was tun?
  - immer den größten Wertebereich nehmen?
  - → Gleitpunkt-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`

- rationalen und reellen Zahlen
- im Rechner: Wertebereich endlich → Genauigkeit begrenzt
- floating point numbers
- Exponentialzahlen in der Form

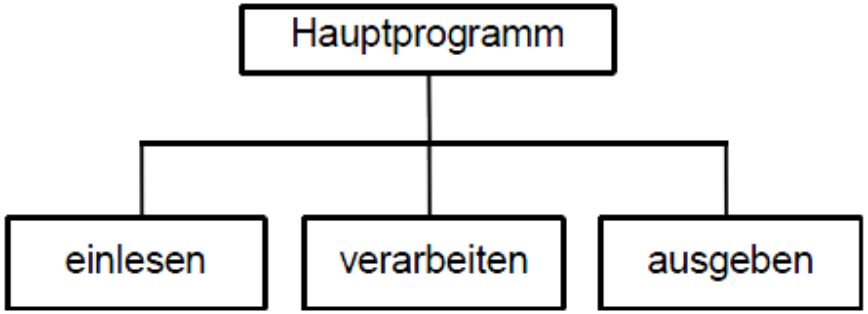
$$\texttt{Mantisse} * \texttt{Basis}^{\texttt{Exponent}}$$

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

Unterprogramme

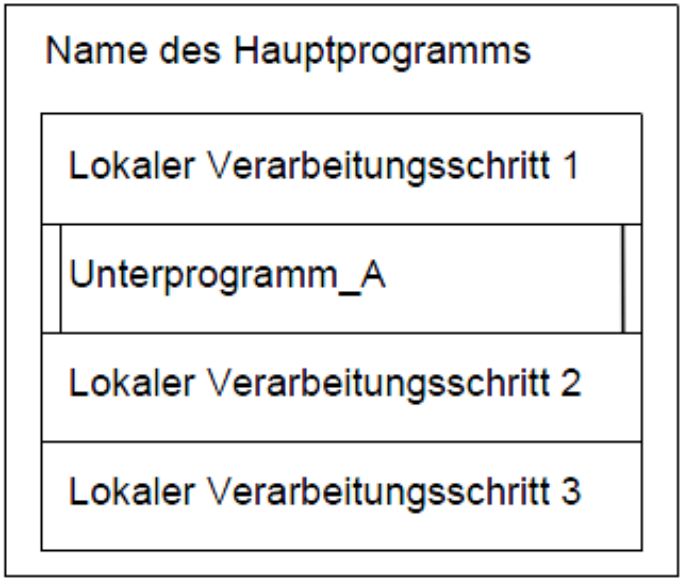
Unterprogramme

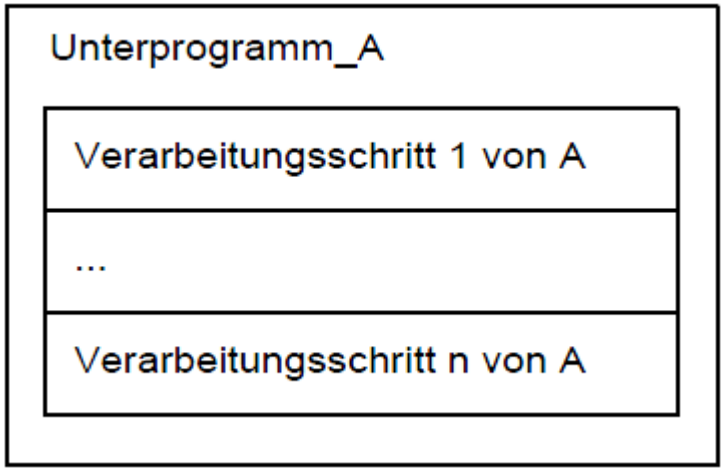
- Mittel zur Strukturierung eines Programms
- Modularität statt "riesengroßes" Programm
- Wiederverwendbarkeit
- Hauptprogramm sollte primär Unterprogramme aufrufen
- Prozedur vs. Funktion
- Bibliotheken (Libraries) = Sammlung von Unterprogrammen
- Beispiel Aufrufhierarchie:



Unterprogramme in Struktogrammen

- jedes Unterprogramm ein Diagramm
  → Programm = Menge von Struktogrammen

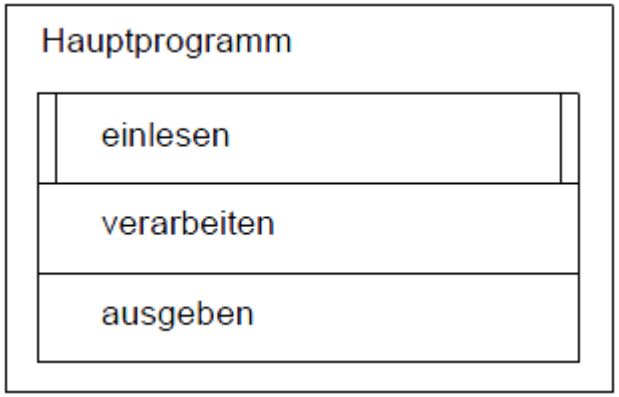




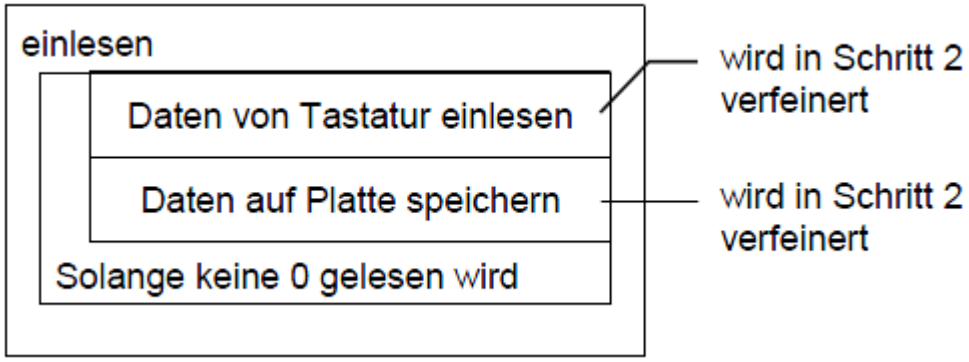
- Wann führt man ein Unterprogramm ein?

Schrittweise Verfeinerung (top-down)

- Grobstruktur eines Hauptprogramms:

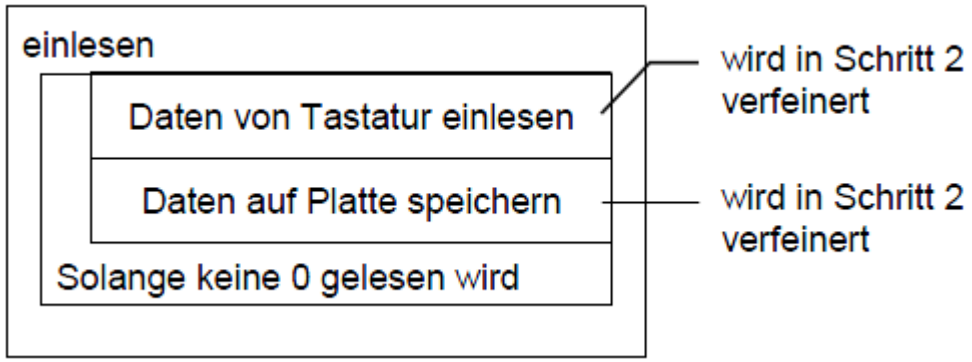


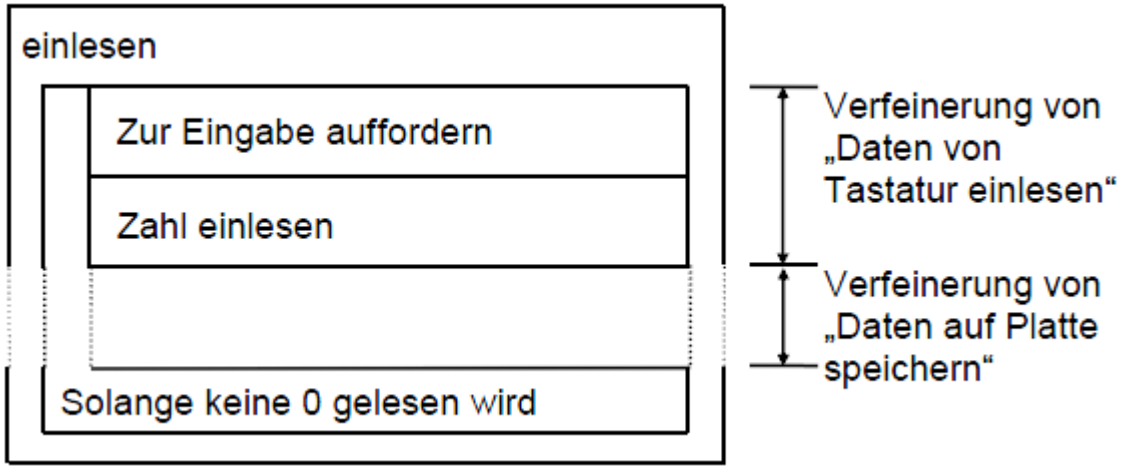
- Verfeinerung Schritt 1:



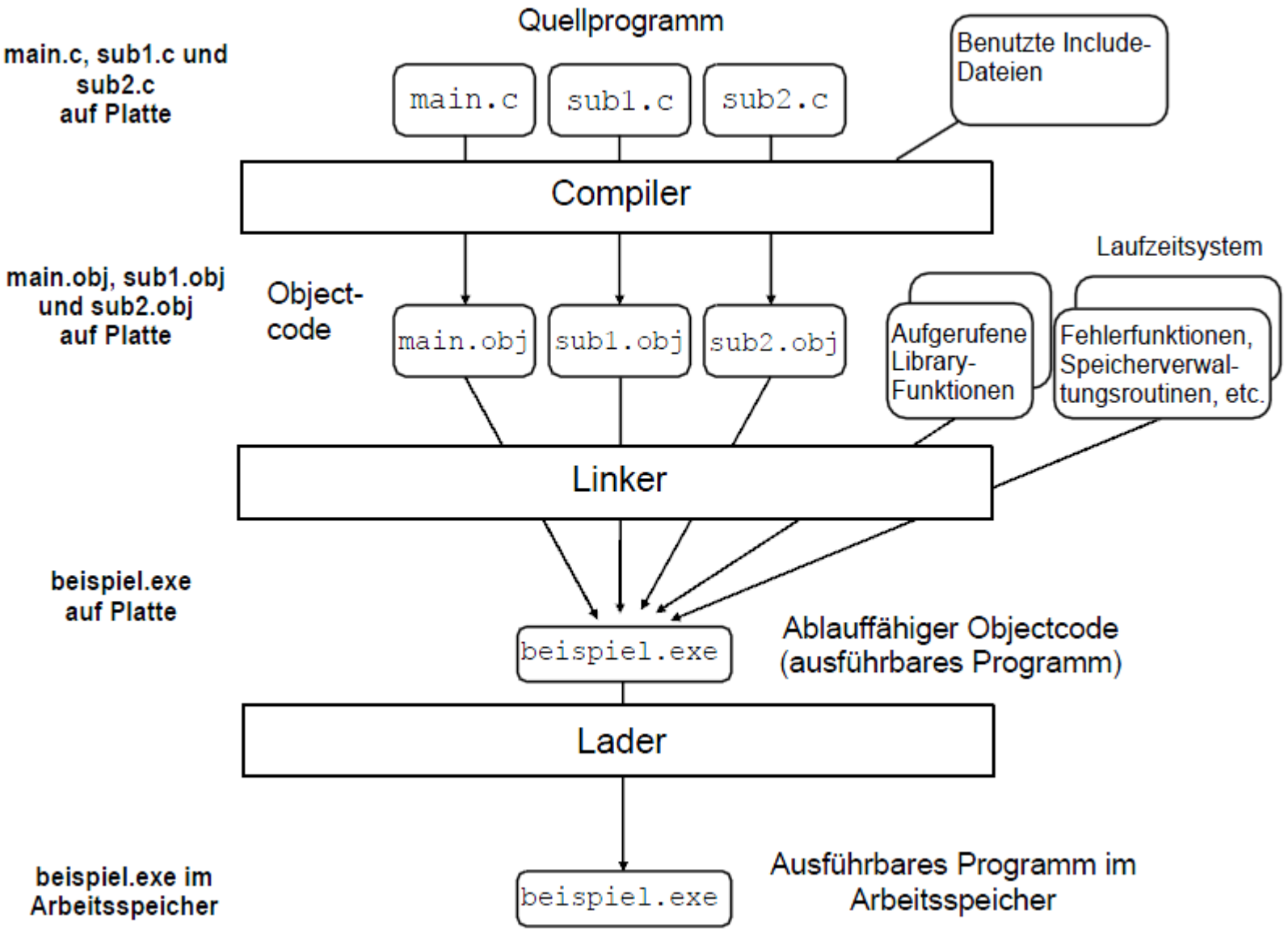
Schrittweise Verfeinerung (top-down)

- Verfeinerung Schritt 1 zu Schritt 2:

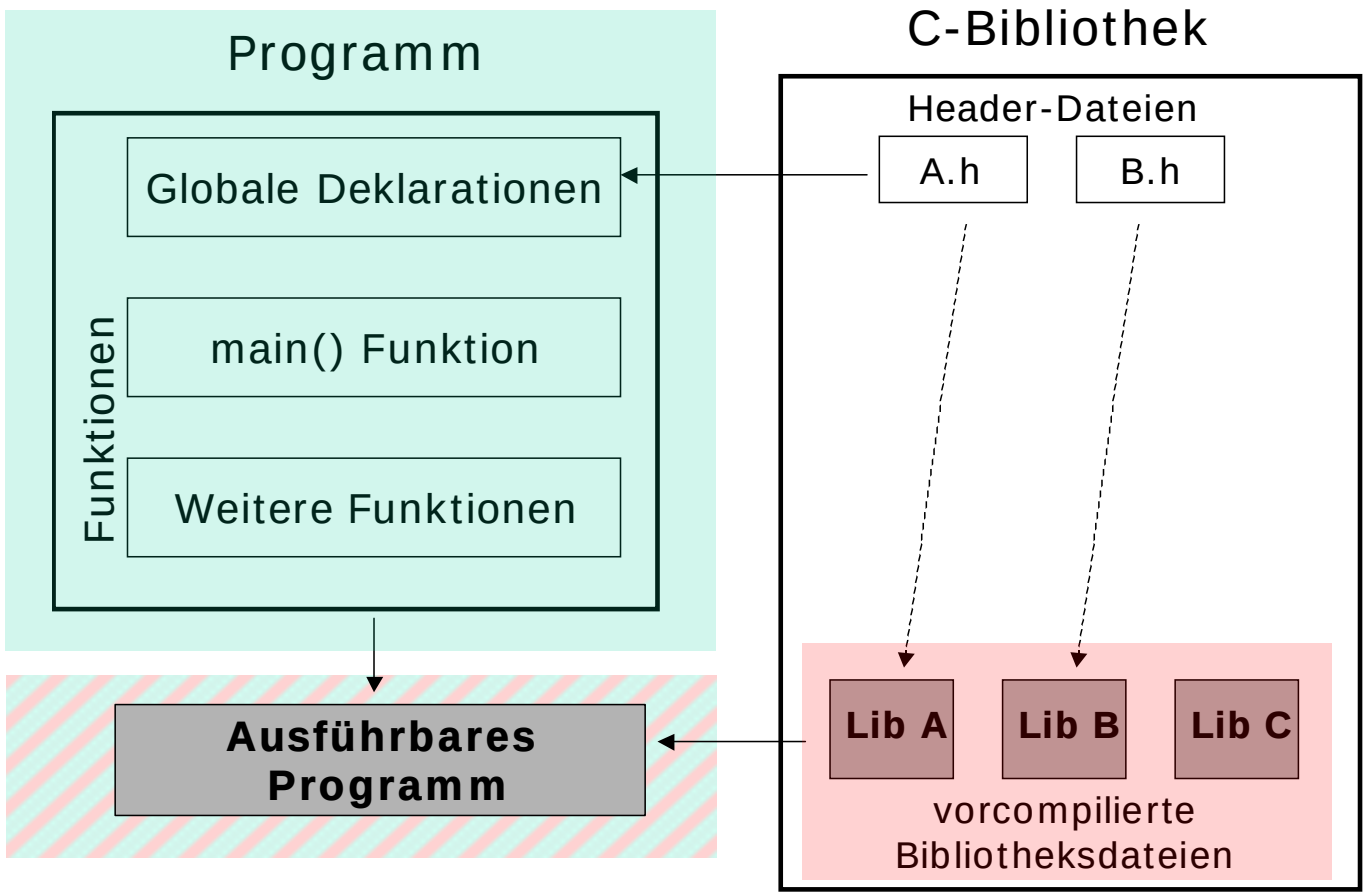




Übersetzung


Quellprogramm
main.c, sub1.c und sub2.c auf Platte
main.c
sub1.c
sub2.c
Benutzte Include-Dateien
Compiler
Laufzeitsystem
main.obj, sub1.obj und sub2.obj auf Platte
Object-code
main.obj
sub1.obj
sub2.obj
Aufgerufene Library-Funktionen
Fehlerfunktionen, Speicherverwal-tungsroutinen, etc.
Linker
beispiel.exe auf Platte
beispiel.exe
Ablauffähiger Objectcode (ausführbares Programm)
Lader
beispiel.exe im Arbeitsspeicher
beispiel.exe
Ausführbares Programm im Arbeitsspeicher

Einbinden von Bibliotheken



Dateinamenskonventionen

Die Dateinamen haben üblicherweise folgende Erweiterungen:

(Linux)

- **.c** für den Quellencode
- **.h** für Include-Dateien
- **.o** für den Objektcode
- **a.out** bzw. keine Endung für das lauffähige Programm

(MS-Windows)

- **.c** für den Quellcode (.cpp für C++ Quellcode)
- **.h** für Include-Dateien
- **.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
- erschwerte Fehlersuche
- hohe Ausführungsgeschwindigkeit

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

- Programmtext wird schrittweise zur Laufzeit übersetzt
- einfach zu realisieren
- leichte Fehlersuche

Compreter - Java, .NET Sprachen

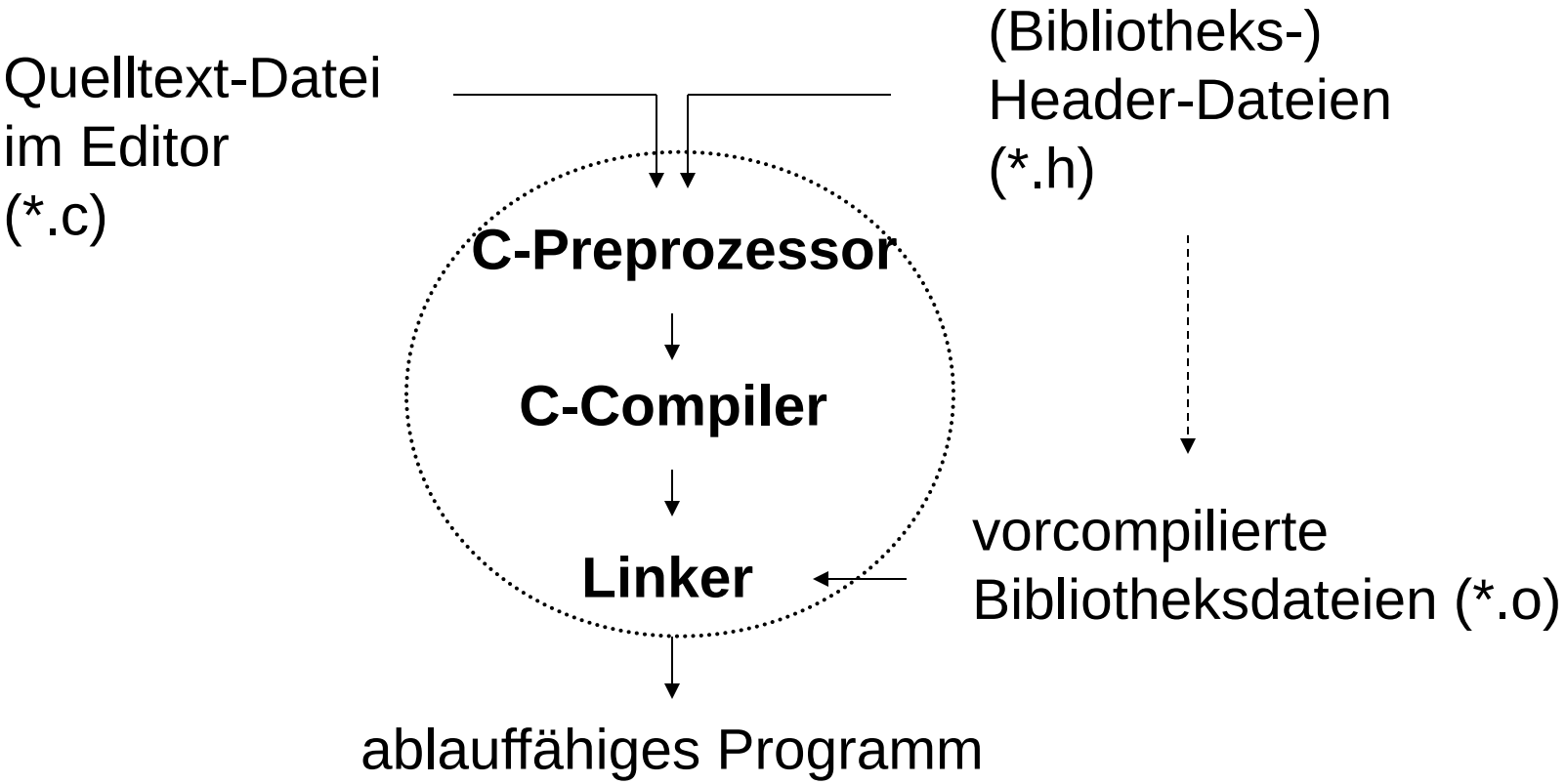
- Übersetzung vor Ausführung in Bytecode
- Interpretation zur Laufzeit durch VM

Compiler: C → Assembler/Maschinencode

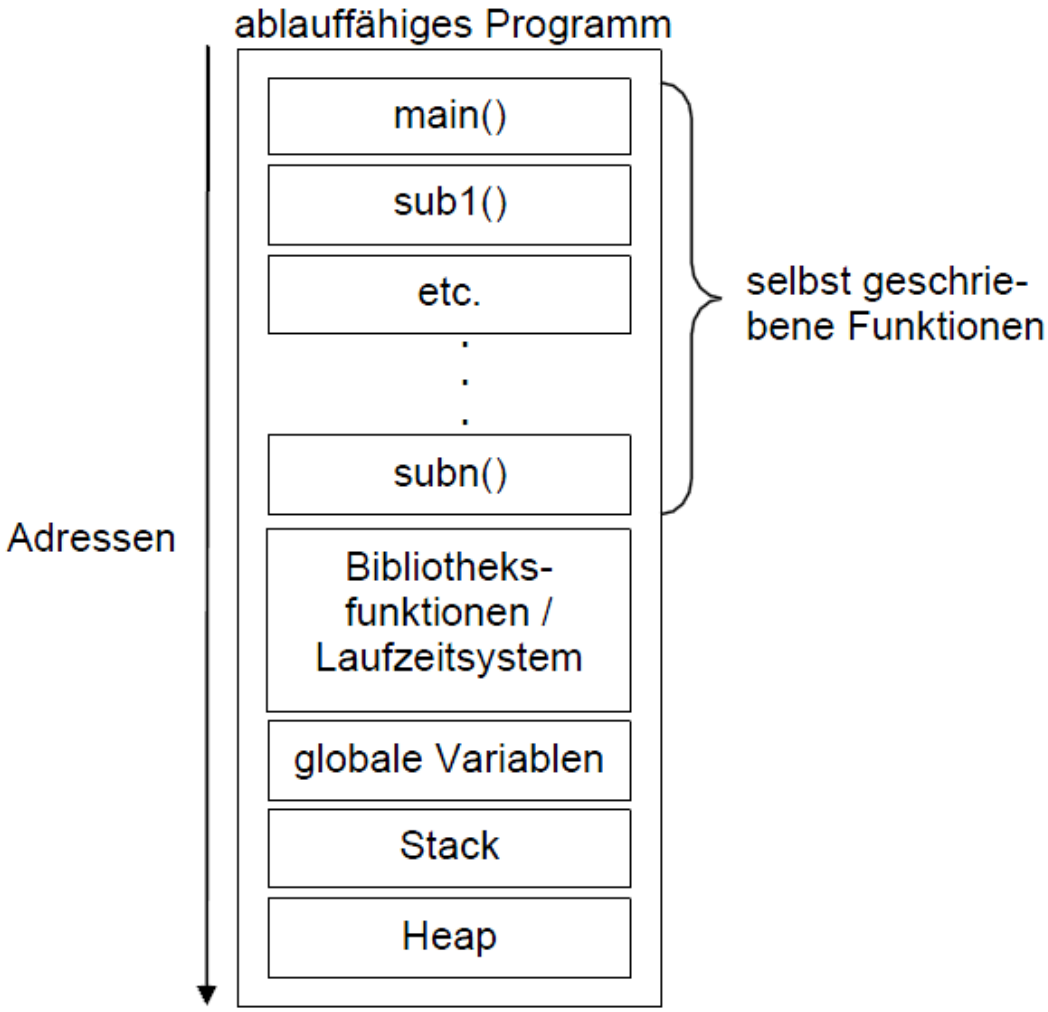
Die Darstellung des Hochsprachen-zusammen mit dem Assembler- bzw. Maschinencode und den Ladeadressen

```
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
```

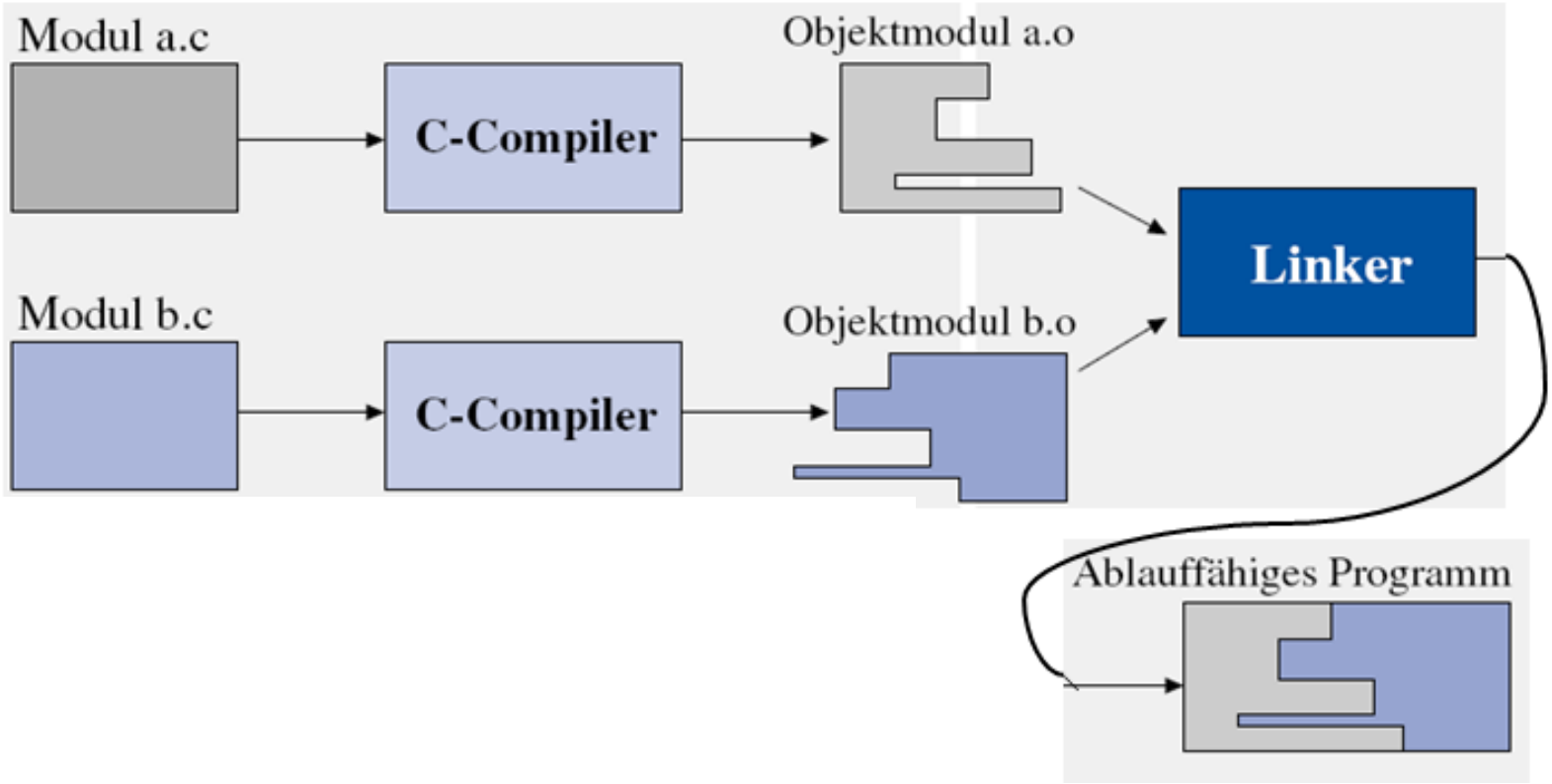
Preprozessor - Compiler - Linker



Linker


ablauffähiges Programm
main()
sub1()
etc.
subn()
selbst geschrie-
bene Funktionen
Adressen
Bibliotheks-
funktionen /
Laufzeitsystem
globale Variablen
Stack
Heap

Linken mehrerer Module



Linken von Bibliotheksroutinen