

# Software-Entwicklung mit C++ Programmieren ST1

**Wintersemester 2011/2012**

Dr. Klaus Mück  
cogisys Gesellschaft für kognitive  
Informationssysteme mbH

Prof. Dr. Thorsten Leize  
Hochschule Karlsruhe – Technik und  
Wirtschaft

# Software-Entwicklung mit C++ Programmieren ST1

- Übung:  
montags 09:50-11:20, 11:30-13:00  
Raum LI 142
- Vorlesung:  
montags 14:00-15:30  
Raum F003

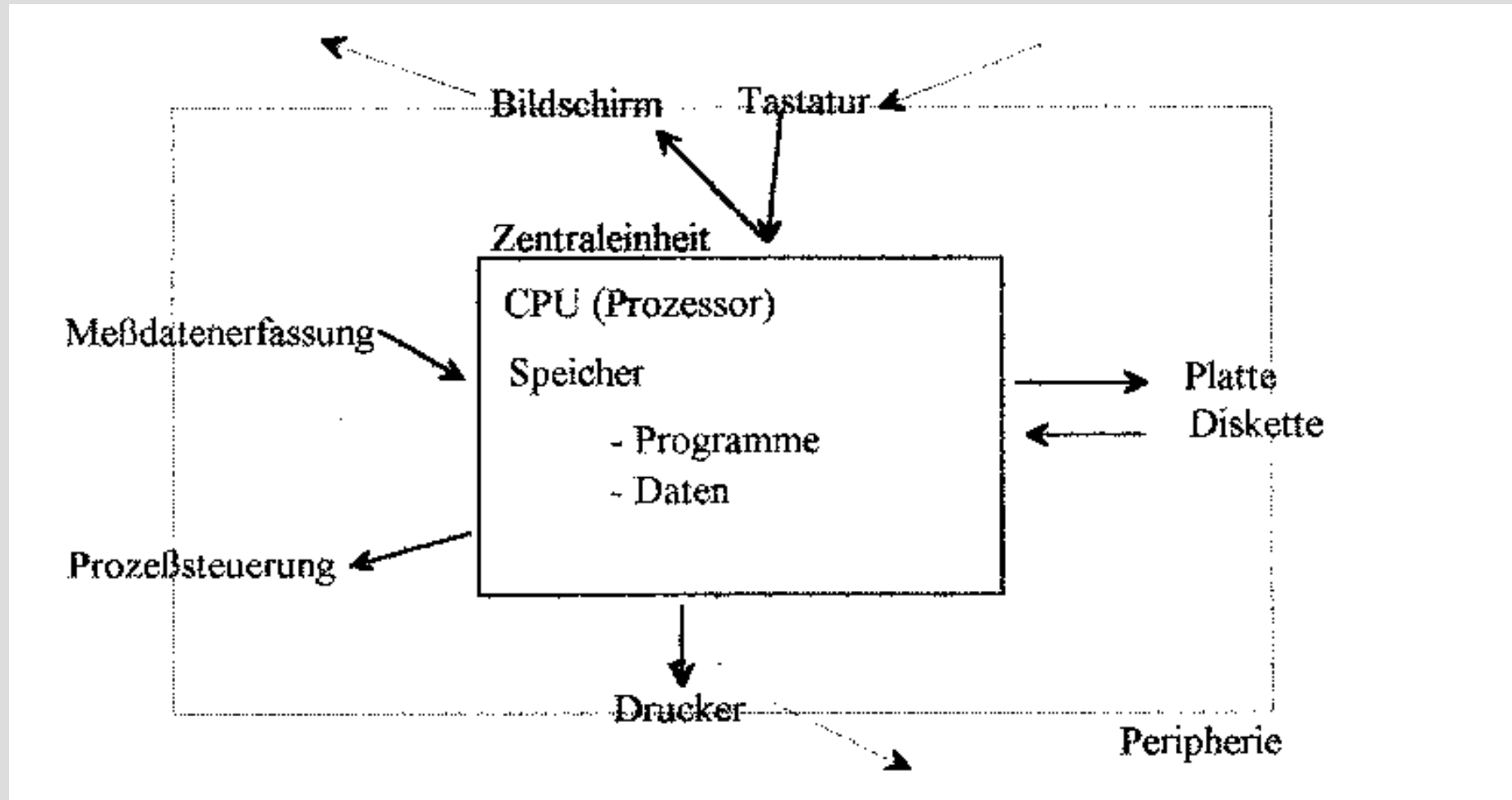
# Software-Entwicklung mit C++ Programmieren ST1

- Wo gibt es welche Infos?
- Hardware, Rechner, Programm
- Algorithmus/Programmieren
- Programmiersprachen, Übersetzer, Präprozessor
- Entwicklungsumgebung
- Sprung ins kalte Wasser! Das erste Programm.

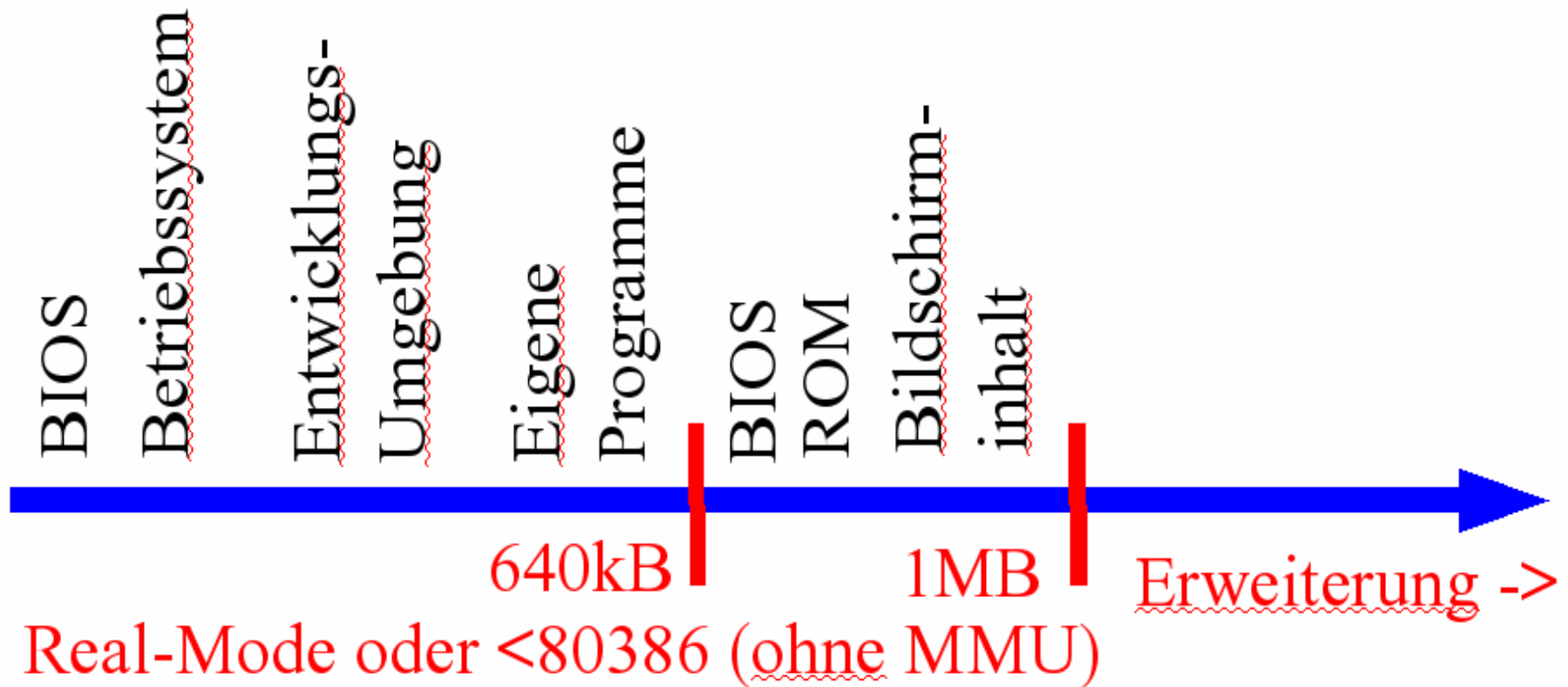
# Software-Entwicklung mit C++ Programmieren ST1

- Wo gibt es welche Infos?
  - google.de
  - wikipedia.de
  - Gaaaanz viele Tutorien im Internet zu C/C++
  - Hilfesysteme von Entwicklungsumgebungen

# Aufbau eines Rechners

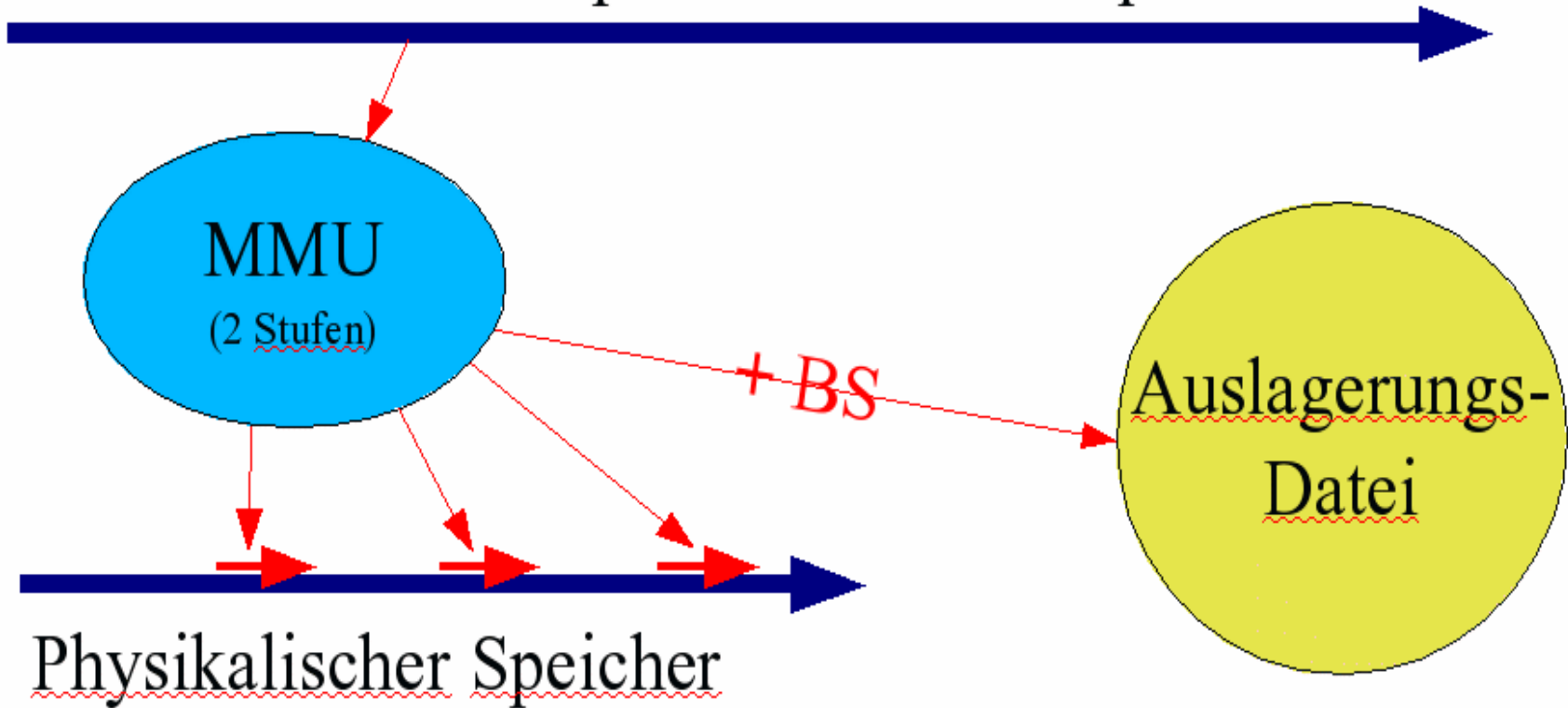


# Programme im Hauptspeicher

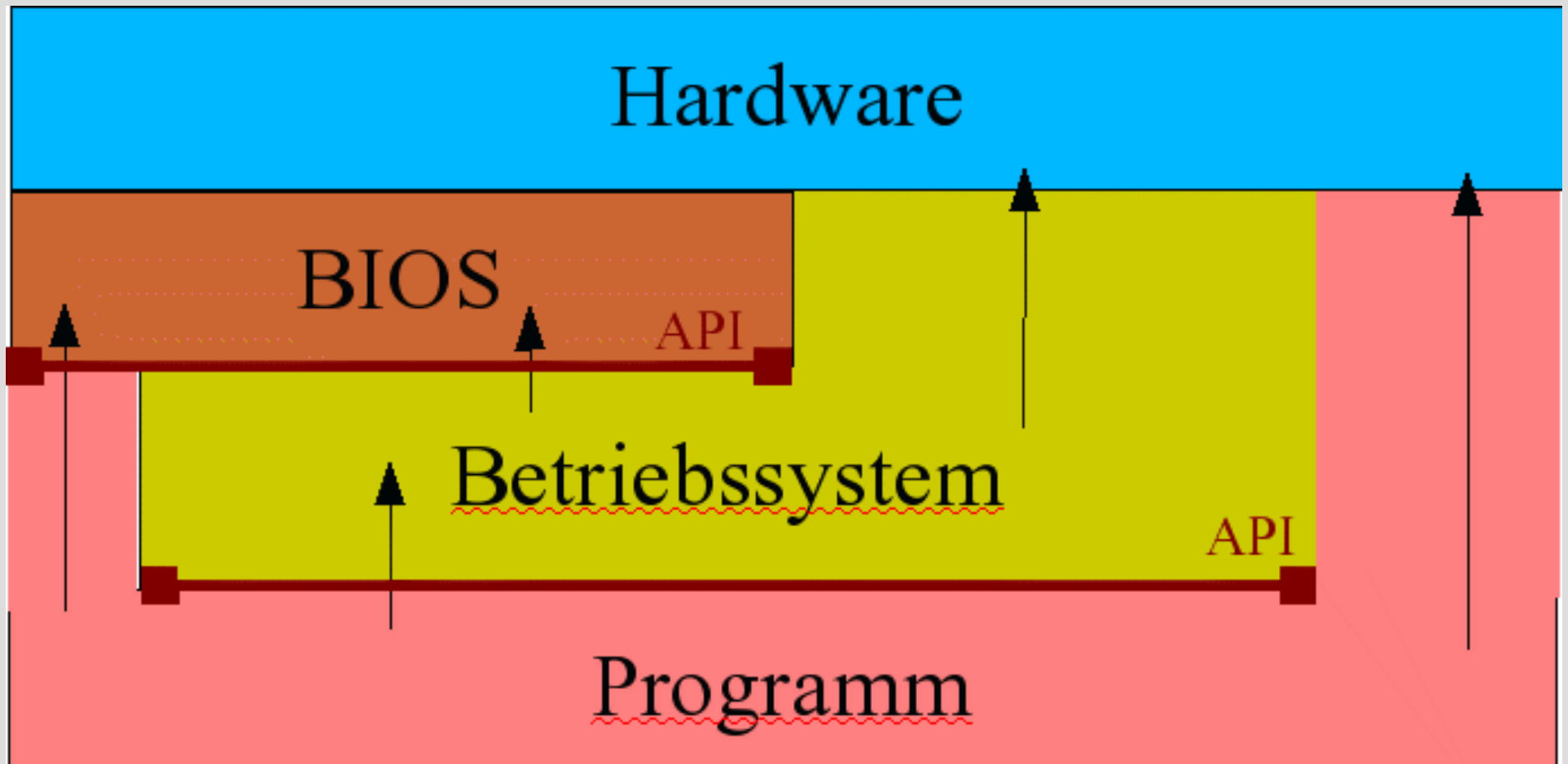


# Programme im Hauptspeicher

Pro Prozess ein kompletter virtueller Speicherraum.



# Programme im Hauptspeicher





# Prozessor

- Jeder Rechner enthält einen **Prozessor**, der **Befehle** ausführt. Der **Kern** eines Prozessors interpretiert dabei die Befehle.
- Die Gesamtheit aller Befehle, die ein Prozessor versteht, nennt man **Befehlssatz**.
- Verschiedene Prozessortypen haben verschiedene Befehlssätze: **Maschinsprache**.
- Programme in einer Maschinsprache laufen nur auf diesem Prozessortyp.
- Ausführbare Programme sind aber auch vom Betriebssystem abhängig.

# Algorithmus, Programm

- Ein **Algorithmus** ist ein allgemeines Verfahren zur systematischen und schrittweisen Lösung einer Aufgabenstellung.  
Beispiel: Kochrezept, Spielregel, Arbeitsanweisung
- Ein **Programm** ist die Formulierung eines Algorithmus in einer Programmiersprache.
- Der **Quellcode** (Quelltext) eines Programms ist eine meist textuelle Beschreibung eines Programms in einer Programmiersprache.

# Algorithmus: Addition

1. Taschenrechner einschalten.
2. Erste Zahl eintippen.
3. + Taste drücken.
4. Zweite Zahl eintippen.
5. = Taste drücken.
6. Ergebnis ablesen.
7. Wenn nochmal, dann 2.
8. Taschenrechner

# Algorithmus: Kochrezept



Ziel: Menü mit drei Gängen für 4 Personen

## Grundlegende Spezifikation:

1. Vorspeise: Badische Flädlesuppe
  2. Hauptgericht: Überbackene Schinkenröllchen mit Spargel
  3. Dessert: Vanilleeis mit heißen Himbeeren
- Getränke: Sekt, Bier, Wein, Wasser etc.

## Verfeinerung der Eigenschaften:

- (selbst gemachte/vorgefertigte) Fleischbrühe mit/ohne Fett
- welche Art von Pfannkuchen (Flädlen), welche Mengen
- viel/wenig Spargel, welche Gewürze, welcher Käse
- frische/eingefrorene Himbeeren, welches Vanilleeis
- welcher Wein, welches Bier etc.

# Algorithmus: Kochrezept

- Erreichen von Teilzielen  
Suppe gekocht, Pfannkuchen gebacken, Spargel gekocht
- Einhaltung der Reihenfolge  
Vorspeise vor Hauptgericht, Hauptgericht vor Dessert  
Auflauf erst in Backofen, wenn Auflaufform gefüllt ist ...
- Einhaltung zeitlicher Rahmenbedingungen  
Spargel 12 min. kochen lassen, Auflauf 25 min. in Backofen ...
- Parallelisierung von Aufgaben  
Suppe kochen und gleichzeitig Pfannkuchen backen,  
Schinkenröllchen richten und Spargel abkochen ...  
Eine Person richtet Vorspeise, andere das Hauptgericht
- Informationsaustausch der beteiligten Personen/Verarbeitungseinheiten  
“reichst Du mir den Käse, machst Du den Auflauf weiter, wie viele Pfannkuchen sind es schon? ...”



Kochkurs von Platten

„... und dann nehmen Sie ein frisches Ei, ein weiteres Ei ... ein weiteres Ei ... ein weiteres Ei ... ein weiteres Ei ... ein weiteres Ei ...“

# Algorithmus/Daten/Strukturen

Essenz dieses Beispiels:

- ein Algorithmus muss nicht notwendigerweise durch eine Maschine ausgeführt werden,
- mit einem Algorithmus können Organisationsabläufe beschrieben werden,
- mit einem Algorithmus kann die Bearbeitung von Strukturen beschrieben werden,
- Strukturen beschreiben Beziehungen zwischen Daten, oder fassen Daten zusammen
- Daten lassen sich durch Strukturen organisieren,
- die Erkennung von Strukturen muss geübt werden, d.h. die passende Organisation von Daten muss gefunden werden.

# Programmiersprachen

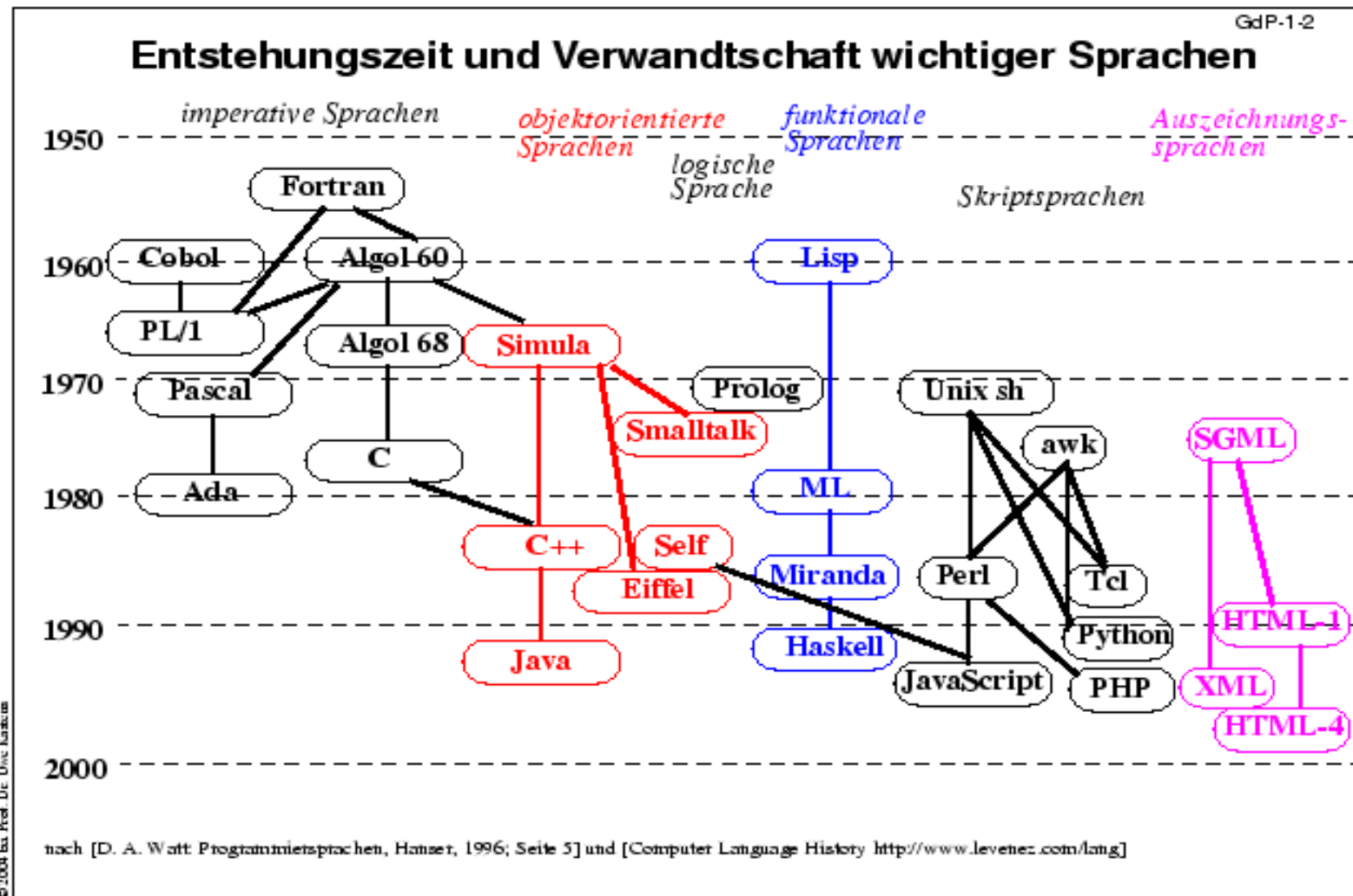
## Maschinennahe Sprachen

- Maschinensprache: Binär und vom Prozessor abhängig
- Assembler: Abkürzung für Maschinenbefehle

## Höhere Programmiersprachen

- Prozedurale/Imperative Programmiersprachen
- Logische Programmiersprachen
- Funktionale Programmiersprachen
- Objektorientierte Programmiersprachen
- Skriptsprachen
- Auszeichnungssprachen

# Programmiersprachen

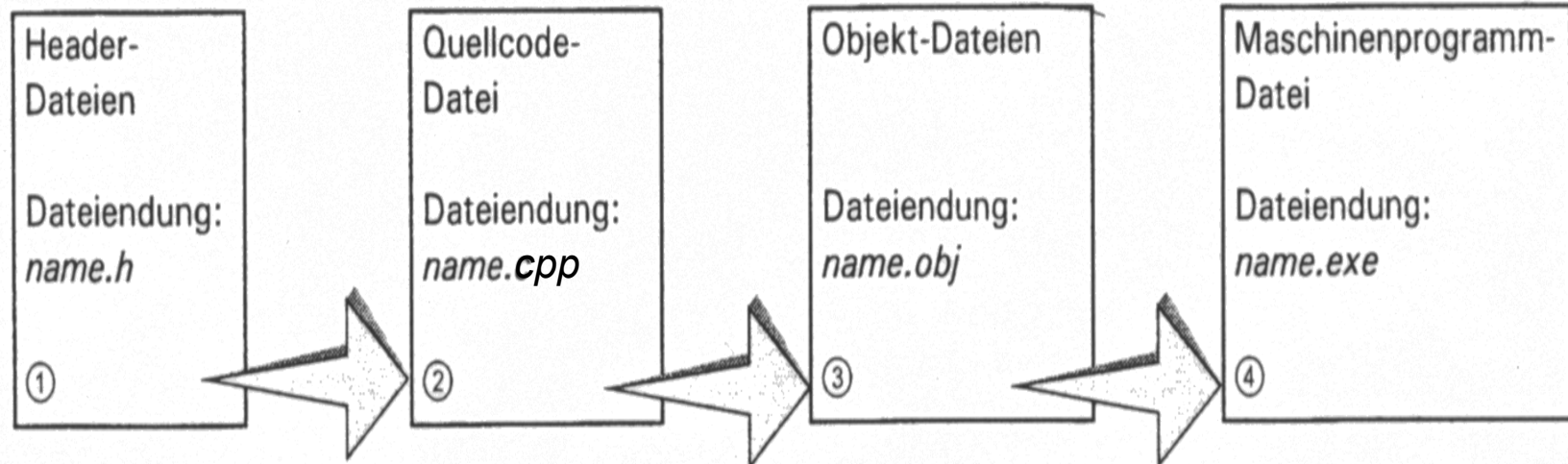




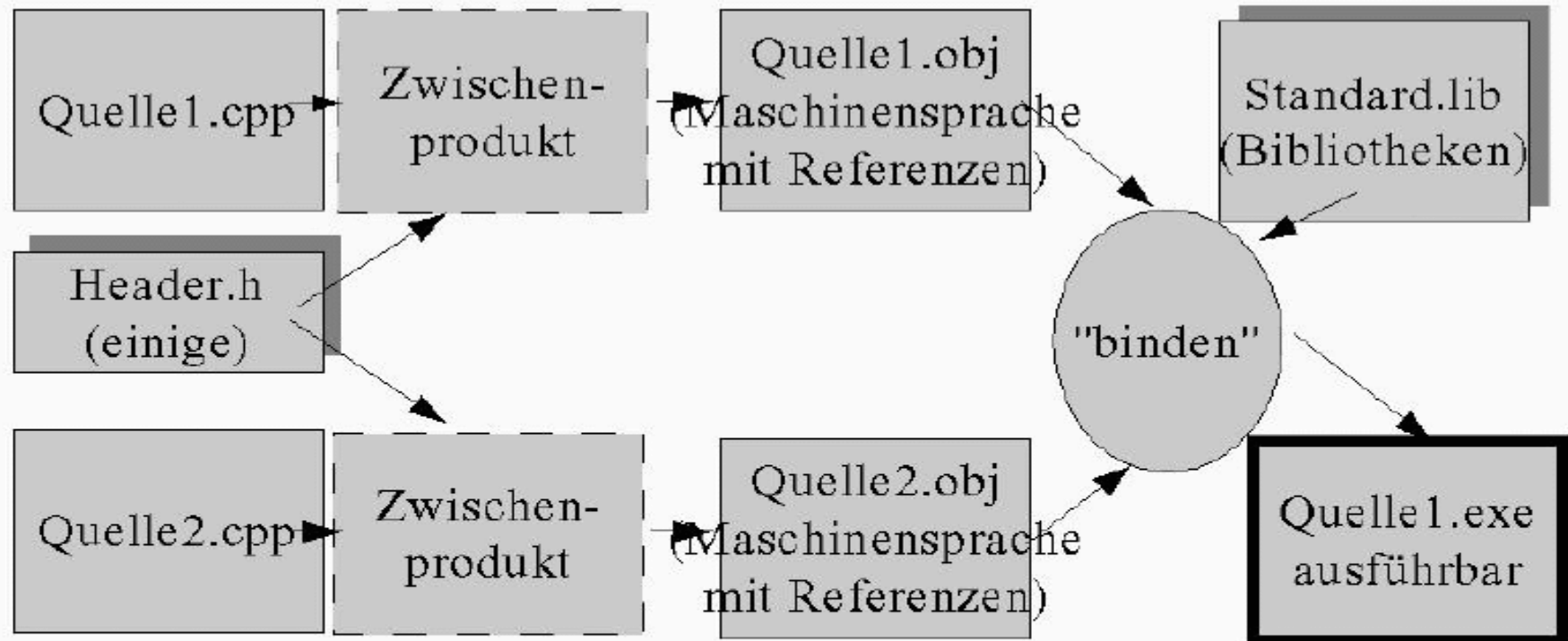
# Vorgehensweise klassisch

- 1) Genaue Formulierung der Aufgabenstellung
- 2) Abstraktion
- 3) Entwurf eines Algorithmus
- 4) Ist der entworfene Algorithmus korrekt? (zurück zu 3)?
- 5) Kodierung (Übersetzung in eine Programmiersprache)
- 6) Testen des Programms. (eventuell zurück zu 5 oder zu 3)
- 7) Effizienz?
- 8) Dokumentation

# Vom Quelltext zum Maschinenprogramm



# Vom Quelltext zum Maschinenprogramm



Quelltexte  
(beliebig viele)  
Präprozessor  
(fügt  
zusammen)

Compiler  
(übersetzt)

Linker  
(realisiert  
Referenzen)

# C(++)-Programm

- Zeichenvorrat, aus dem der Quelltext bestehen darf:  
Alle Buchstaben ohne Umlaute, Sonderzeichen wie `_.?[]()`
- C/C++ ist “case sensitive”, d.h. Groß- und Kleinbuchstaben werden unterschieden.
- “ProgrammStruktur” ist ein anderes Wort als “programmstruktur”.
- Alle Objekte haben Namen, sog. **Bezeichner**. Bezeichner bestehen aus den Zeichen: A..Z, a..z, 0..9, `_` und beginnen mit einem Buchstaben oder Unterstrich, nicht aber mit einer Ziffer!

# C(++)-Programm

- Struktur des Quelltextes als eine Folge von Deklarationen (Vereinbarungen)/Definitionen:
  - Variablen
  - Konstanten
  - Funktionen
  - Datentypen
- Eine Funktion existiert immer: main-Funktion. Hier beginnt und endet die Ausführung des Programms.
- C-Anweisungen enden mit einem Semikolon: ;
- Die Verteilung der Anweisung auf die Textzeilen ist ohne Bedeutung (Ausnahme: Präprozessorbefehle).

# Hallo Welt!

```
// Das erste C++ - Programm
// Autor: Hacker, One
// (C) by the author

#include <iostream> // ein Praeprocessorbefehl

// stellt die C++ Ein-/Ausgabe zur Verfügung

void main()          // eine Funktion ohne Argumente.
{                   // hier beginnt das Hauptprogramm

    std::cout << "Hello world" << std::endl;

}                   // hier endet das Hauptprogramm.
```

# Präprozessor

- Der Präprozessor wird beim Kompiliervorgang vor dem Compiler, also vor dem eigentlichen Übersetzungsvorgang, gestartet.
- Der Präprozessor ersetzt nur Text oder lässt Text weg. Es erfolgt keine Übersetzung in Maschinencode. Das Resultat ist ein Text.
- Alles, was der Präprozessor entfernt, ist dem Compiler nicht zugänglich.
- Die Entscheidungen des Präprozessors erfolgen zur Übersetzungszeit und nicht zur Laufzeit!

# Präprozessor

Befehl	Bedeutung
<code>#include &lt;datei.h&gt;</code>	Fügt die angegebene Datei an dieser Stelle komplett in den Quelltext ein. In den Standardverzeichnissen für Headerdateien. Oder .hpp, .hxx. In modernen Versionen kann man die Endung bei C++ weglassen.
<code>#include "datei.h"</code>	wie bei <>, aber Suche zuerst im momentanen Verzeichnis.
<code>#define name</code>	Macht diesen Namen dem Präprozessor bekannt. Löscht überall im Quelltext (ab dieser Stelle) dieses Wort "name".
<code>#define name rest</code>	wie oben, jedoch wird "name" nicht gelöscht, sondern durch "rest" (auch mehrere Wörter) ersetzt.



# Präprozessor

Befehl	Bedeutung
<code>#if Bedingung ...Teil ja #endif</code>	Trifft die Bedingung zu, bleibt „Teil ja“ als Text erhalten. Im anderen Fall wird der Text entfernt.
<code>#if Bedingung ... Teil ja #else ... Teil nein #endif</code>	Trifft die Bedingung zu, bleibt „Teil ja“ als Text erhalten und „Teil nein“ wird entfernt. Trifft die Bedingung <i>nicht</i> zu, bleibt „Teil nein“ als Text erhalten und „Teil ja“ wird entfernt.
<code>#if Bedingung1 ... Teil ja Bedingung1 #elseif Bedingung2 ... Teil ja Bedingung2 #else ... Teil nein #endif</code>	Trifft die Bedingung1 zu bleibt nur „Teil ja Bedingung1“ erhalten. Trifft Bedingung2 zu bleibt nur „Teil ja Bedingung2“ erhalten. Nur falls alle Bedingungen nicht zutreffen bleibt „Teil nein“ erhalten.

# Präprozessor

Befehl	Bedeutung
<code>#if defined name</code> ... Teil ja <code>#endif</code>	Ist „name“ definiert, bleibt „Teil ja“ als Text erhalten. Im anderen Fall wird der Text entfernt.
<code>#if defined name</code> ... Teil ja <code>#else</code> ... Teil nein <code>#endif</code>	Ist „name“ definiert, bleibt „Teil ja“ als Text erhalten und „Teil nein“ wird entfernt. Ist „name“ <i>nicht</i> definiert, bleibt „Teil nein“ als Text erhalten und „Teil ja“ wird entfernt.
<code>#ifdef name</code> ... Teil ja <code>#endif</code>	Übliche Kurzform
<code>#ifndef name</code> ... Teil ja <code>#endif</code>	Übliche Kurzform für die verneinte Form.

# Software-Entwicklung mit C++ Programmieren ST1

- Anmerkungen zum Programmieren
  - Bahnhof! ... Das ist ganz normal.
  - Es wird auch hier nur mit Wasser gekocht und es ist alles logisch nachvollziehbar.
  - Üben, üben, üben ... üben.
  - Ausprobieren, nicht nur anschauen.

# Software-Entwicklung mit C++ Programmieren ST1

- Entwicklungsumgebungen
  - Linux:  
kdevelop, anjuta
  - Linux/Windows:  
eclipse
  - Windows:  
Bloodshed, Microsoft-  
Entwicklungsumgebung

# Software-Entwicklung mit C++ Programmieren ST1

- Sprung ins kalte Wasser! Das erste Programm.
  - Schreiben Sie ein Hauptprogramm, das auf den Bildschirm “Hello world!” ausgibt.

# Datentypen

- Datentypen ist die Zusammenfassung von Objektmengen mit den darauf definierten Operationen.
- Mit Datentypen werden Speicherinhalte eindeutig interpretiert.
- Datentypen können mit Zahlenmengen aus der Mathematik assoziiert werden.
- Datentypen legen Gültigkeitsbereiche fest.
- Festlegung auf Datentypen ermöglicht die Prüfung auf Typgleichheit während des Übersetzungsvorgangs: Typsicherheit.

# Einfache/Elementare Datentypen

- Einfache Datenstrukturen:  
char, int, long, float, double

Typisch für eine 32 Bit-Plattform

char: -128 .. 127

int: -2 147 483 648 .. 2 147 483 647

long: -2 147 483 648 .. 2 147 483 647

float: 1.5E-45 .. 3.4E38

double: 5.0E-324 .. 1.7E308

# Variablendeklaration

- **Formal:**  
[Speicherspezifizierer][Modifizierer] Datentyp Variablenname;
- **Speicherspezifizierer:**  
auto – nur gültig innerhalb eines Anweisungsblocks  
static – Variable wird nur einmal angelegt  
register – Variable wird im Prozessorregister gehalten
- **Modifizierer:**  
signed – vorzeichenbehaftet  
unsigned – vorzeichenlos  
short – Einschränkung des Wertebereichs  
long – Erweiterung des Wertebereichs



# Variablendeklaration/-definition

- einfache Variablendeklaration:  
int iAnzahl;  
float fMittelwert;
- Zuweisung:  
iAnzahl = 25;  
fMittelwert = 0.5;
- Variablendefinition:  
int iAnzahl = 26;  
float fMittelwert = 0.7;

# Nutzung von Variablen

```
int zahl1, zahl2;           // Deklaration zweier ganzer Zahlen
```

```
zahl1 = 10;                 // Definition in Form einer Zuweisung  
zahl2 = -17;
```

```
zahl2 = zahl1 + zahl2;     // Definition durch Zuweisung einer Summe
```

```
int zahl = 12;            // Deklaration und Definition!
```

Wertebereich von **int** in C/C++ ist von der verwendeten Plattform abhängig!

$-32768 \leq \mathbf{int} \leq 32767$       65536 Zahlen:  $2^{16}$  entspricht 2 Byte

$0 \leq \mathbf{unsigned\ int} \leq 65535$       65536 Zahlen:  $2^{16}$  entspricht 2 Byte

# Nutzung von Variablen

Prozessor mit 32 Bit:

$$-2^{31} \leq \text{int} < 2^{31}$$

$$0 \leq \text{unsigned int} \leq 2^{32}-1$$

4 294 267 296 Zahlen:  $2^{32}$  entspricht 4 Byte

3            00000000 00000011

2            00000000 00000010

1            00000000 00000001

0            00000000 00000000

-1           11111111 11111111

-2           11111111 11111110

-3           11111111 11111101

...

-32768      10000000 00000000

Zweierkomplement

-3 + 3 =    11111111 11111101

             00000000 00000011

             00000000 00000000 = 0

# Nutzung von Variablen

Folgen des begrenzten Wertebereiches bei 16 Bit-Integer:

```
int zahl1 = 32760;
```

```
int zahl2 = zahl1 + 8;
```

zahl2 == 32768?

```
unsigned int zahl1 = 65530;
```

```
unsigned int zahl2 = zahl1 + 8;
```

zahl2 == 65538?

# Nutzung von Variablen

Folgen des begrenzten Wertebereiches bei 16 Bit-Integer:

```
int zahl2 = 32760 + 8;  
zahl2 = 32768?
```

Leider nein, denn:

```
32768 = 10000000 00000000  
      U -32768
```

---

```
unsigned int zahl2 = 65530 + 8;
```

```
zahl2 == 65538?
```

Nein, denn  $65538 > 65536$

```
65530+8 = 11111111 11111010  
        + 00000000 00001000  
        = 1 00000000 00000010  
        U 2
```

# Nutzung von Variablen

Weitere Ganzzahldefinitionen:

```
int zahl1 = 0431;           // Oktalzahl: 281 in dezimal  
int zahl2 = 0x10ab;        // Hexadezimalzahl: 4267 dezimal  
  
int zahl3 = 097;  
int zahl4 = 0xdeadbeaf;  
int zahl5 = 0x1243km0;
```

# Nutzung von Variablen

Und was ist mit PI?

```
PI = 3.1415926536;    // reelle Zahl
```

Definition einer Fließkommazahl:

```
double PI;           // Deklaration einer reellen Zahl
```

```
PI = 3.141;           // Definition einer reellen Zahl durch Zuweisung
```

```
double zahl1 = 2.43;    // Deklaration und Definition
```

```
double zahl2 = zahl1 + 0.01;
```

```
double zahl3 = 2.54e12; // Mantisse e Exponent
```

```
    u 2.54 * 10^12
```

# Nutzung von Variablen

Begrenzte Darstellung

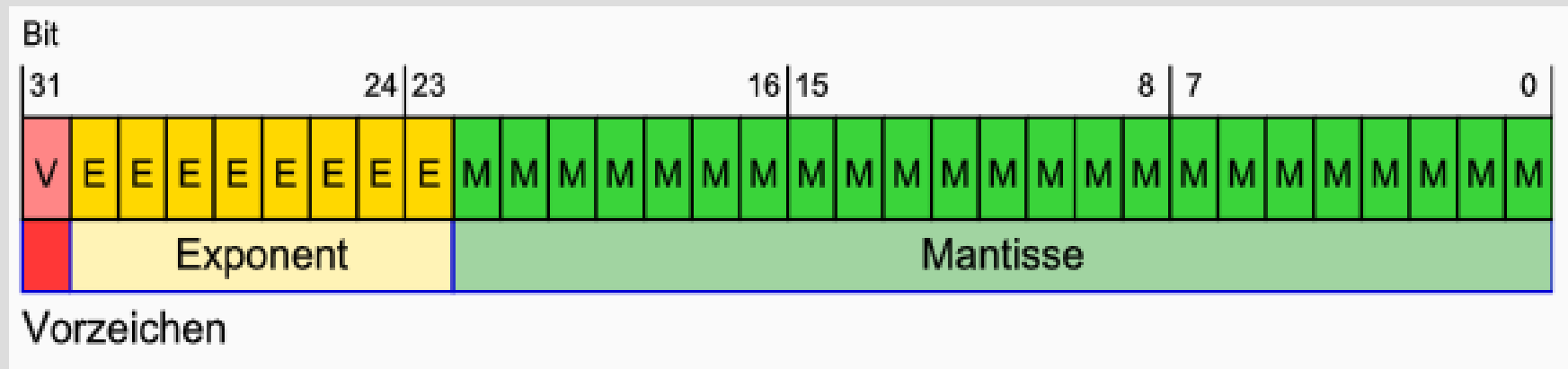
```
double zahl1 = 12000.03;
```

Interne Darstellung ist normalisiert:

zahl1-Mantisse = 0.1200003 --> 1200003

zahl1-Exponent = 5

Dieses Konzept im Binärsystem:





# Nutzung von Variablen

Probleme der begrenzten Darstellung

```
double zahl1 = 12000.03;  
double zahl2 = zahl1 + 0.0000000000000001;
```

zahl2?

Normalisierte Darstellung:

0.1200003	5
0.1	-11
0.120000300000000000	5
0.000000000000000001	5
-----	
0.120000300000000001	5
0.1200003000	5

# Character

## Einzelne Zeichen

```
char zeichen;           // Deklaration einer Variable vom Typ char
```

```
zeichen = 'A';          // Definition eines Zeichens mit Buchstaben 'A'
```

```
char zeichen2 = 'j'; // Deklaration und Definition
```

## ASCII-Zeichen

(American Standard Code for Information Interchange)

'A'	65	0x41	'0'	48	0x30
'B'	66	0x42	'1'	49	0x31
'C'	67	0x43	'2'	50	0x32
...			...		
			'9'	57	0x39

# Arithmetische Operationen

## Addition

+      `summe = summand1 + summand2;`

$$24 = 8 + 16;$$

## Subtraktion

-      `differenz = minuend - subtrahent;`

$$23 = 30 - 7;$$

## Multiplikation

\*      `produkt = faktor1 * faktor2;`

$$66 = 6 * 11;$$

## Division

/      `quotient = dividend / divisor;`

$$12 = 25 / 2;$$

$$12.5 = 25.0 / 2.0;$$

## Division mit Rest

%      `rest = dividend % divisor;`

$$4 = 31 \% 27;$$

# Bitoperationen

## Und-Verknüpfung $a \& b$

a	1 1 0 1
b	1 0 1 1
$a \& b$	1 0 0 1

## Oder-Verknüpfung $a | b$

a	1 1 0 1
b	1 0 1 1
$a   b$	1 1 1 1

## Exklusiv-Oder-Verknüpfung $a \wedge b$

a	1 1 0 1
b	1 0 1 1
$a \wedge b$	0 1 1 0

## Negation $\sim a$

a	1 1 0 1
$\sim a$	0 0 1 0

# Bitoperationen

## Linksschieben

```
a          0 0 1 1 0 1 0 1
a << 2     1 1 0 1 0 1 0 0
```

Linksschieben entspricht  
Multiplikation mit 2.

Warum?

Frage: Ist  $b1 == a$ ? Ist  $b2 == a$ ? Ist  $b1 == b2$ ?

```
int a = 00110101;
int b1 = (a >> 2) << 2;
```

## Rechtsschieben

```
a          0 0 1 1 0 1 0 1
a >> 2     0 0 0 0 1 1 0 1
```

Rechtsschieben entspricht  
Division durch 2.

Warum?

```
int a = 00110101;
int b2 = (a << 2) >> 2;
```

# Zuweisungsoperatoren

<code>x ~= x;</code>	<code>x = ~x;</code>	<code>x += y;</code>	<code>x = x + y;</code>
<code>x &amp;= 3;</code>	<code>x = x &amp; 3;</code>	<code>x -= y;</code>	<code>x = x - y;</code>
<code>x  = 7;</code>	<code>x = x   7;</code>	<code>x *= y;</code>	<code>x = x * y;</code>
<code>x ^= 9;</code>	<code>x = x ^ 9;</code>	<code>x /= y;</code>	<code>x = x / y;</code>
<code>x &lt;&lt;= 2;</code>	<code>x = x &lt;&lt; 2;</code>	<code>x %= y;</code>	<code>x = x % y;</code>
<code>x &gt;&gt;= 2;</code>	<code>x = x &gt;&gt; 2;</code>		

<code>x++, ++x</code>	<code>x += 1; x = x + 1;</code>
<code>x--, --x</code>	<code>x -= 1; x = x - 1;</code>

```
int x = 5;
int a = x++ * 2;           // a == 10, x == 6
int b = ++x * 2;          // b == 12, x == 6
```

# Umwandlung eines Datentyps

```
int a = 4;  
int b = 6;
```

```
double c = a/b;    /* c = 4 / 6 = 0 */
```

```
double d = ((double)a)/((double)b);    /* c = 4.0/6.0 = 0.6666666 */
```

```
double x = 6.66;
```

```
int y = (int) x;    /* y = 6, expliziter Cast */
```

```
int z = x;    /* z = 6, impliziter Cast */
```

**neue Form:**

```
double x = 0.66;
```

```
int y = static_cast<int>(x);
```

# Variablensichtbarkeit

- lokale Variablen:  
Variablen sind nur innerhalb eines Anweisungsblocks sichtbar
- globale Variablen:  
Variablen sind im gesamten Programm sichtbar

```
int giGlobal;

void main()
{
    int iLokal1 = 2;
    giGlobal = 5;
    {
        int iLokal1 = 4;
        int iLokal3 = 3;
        giGlobal = 7;
    }
    // iLokal1 hat den Wert 2
    // iLokal3 ist hier nicht sichtbar
    // giGlobal hat den Wert 7
}
```

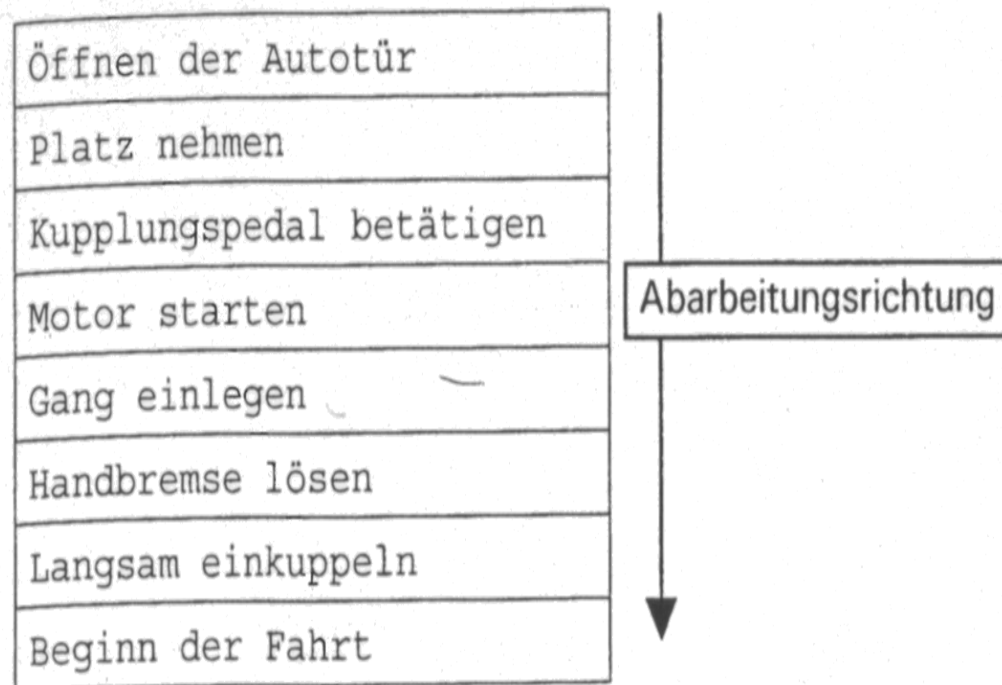


# Konstanten

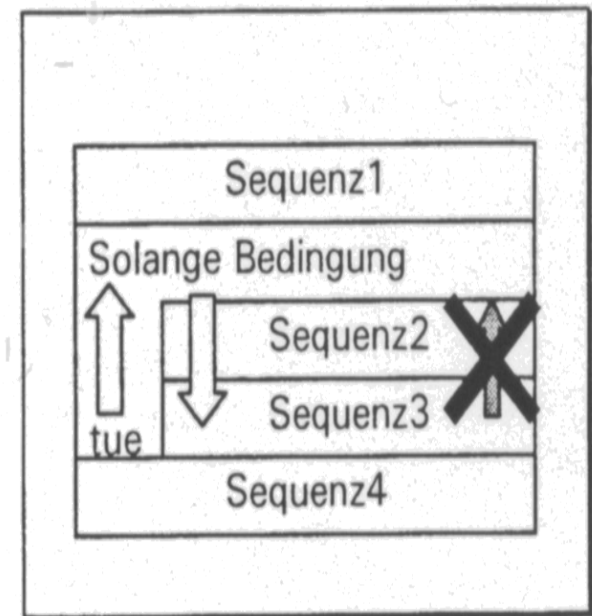
```
const double PI = 3.1415926536;  
const double MWST = 0.16;      // [ % ]  
const double ERDBESCHLEUNIGUNG = 9.81; // [m/s^2]  
const int VERSION = 6;  
const int BUILD = 1244;  
const char ZEICHEN = 'J';  
  
#include <iostream>  
using namespace std;  
  
int main()  
{  
    const double PI = 3.1415926536;  
    const double RADIUS = 14.0;  
    cout << "Der Kreisumfang betraegt:" << 2 * PI * RADIUS << endl;  
    cout << "Die Kreisflaeche betraegt: " << PI * RADIUS * RADIUS << endl;  
    return 0;  
}
```

# Programmablauf

Struktogramm: Starten eines Autos



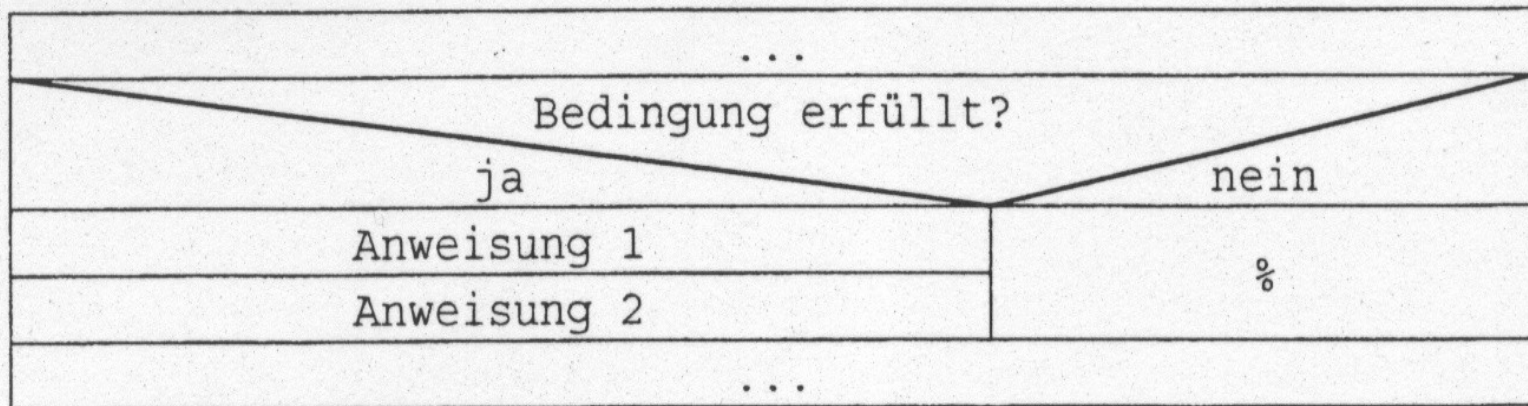
Richtungsregel



Nach den Regeln für das Lesen von Struktogrammen ist in dem Algorithmus für den Vorgang des Startens eines Autos eine Richtungskehr (z. B. Wiederholung von Sequenzen) nicht möglich.

# Einfache Selektion

Struktogramm einer einfachen Selektion



```
if( Bedingung == TRUE)
{
    Anweisung1;
    Anweisung2;
}
```

```
if( Bedingung == TRUE)
    Anweisung;
```

# Einfache Selektion

```
#include <iostream>
using namespace std;

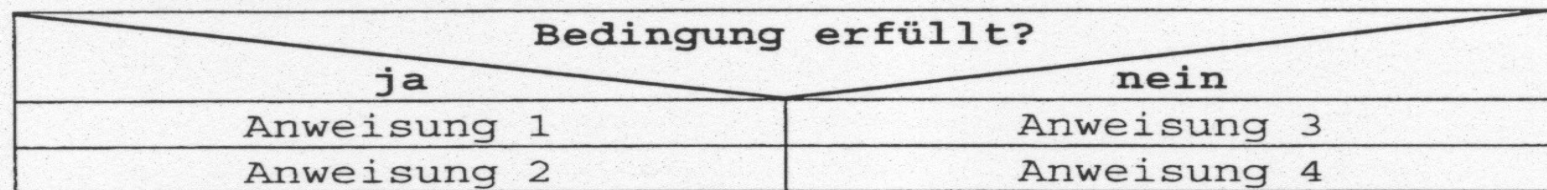
int main()
{
    int iZahl;
    cout << "Geben Sie eine Zahl ein: ";
    cin >> iZahl;

    if(iZahl == 1)
    {
        cout << "Zahl ist 1!" << endl;
    }

    return 0;
}
```

# Zweifache Selektion

Struktogramm einer zweifachen Selektion



```
if( Bedingung == TRUE)
{
    Anweisung1;
    Anweisung2;
}
else
{
    Anweisung3;
    Anweisung4;
}
```

```
if( Bedingung == TRUE)
    Anweisung1;
else
    Anweisung2;
Anweisung3;
```

# Verschachtelte if-Anweisung

```
if( Bedingung1 == TRUE)
{
    Anweisung1;
    if( Bedingung2 == TRUE)
    {
        Anweisung2;
    }
    Anweisung3;
}
else
{
    Anweisung4;
    Anweisung5;
}
Anweisung5;
```

```
int iZahl1 = 1;
int iZahl2 = 2;

if(iZahl1 ==1)
    if(iZahl2 == 3)
        Anweisung1;
    else
        Anweisung2;
    Anweisung3;
/*****/
if(iZahl1 == 1)
    if(iZahl2 == 3)
        Anweisung1;
    else
        Anweisung2;
    Anweisung3;
```

# Bedingungsoperator, Boolsche Operatoren

Ergebnis = Ausdruck ? wert1 : wert2;

Berechnung des Maximums zweier Zahlen:

```
if( a > b)                maximum = (a > b) ? a : b;
    maximum = a;
else
    maximum = b;
```

Verknüpfung mehrerer Bedingungen durch UND (&&) bzw. ODER (||)

```
if( a <= b && b <= c) ...    if( a > b || b > c)
```

TRUE, falls b in [a..c]

FALSE, falls b in [a..c]

# Beispiel für if-Anweisung

```
unsigned int uiDayOfWeek;

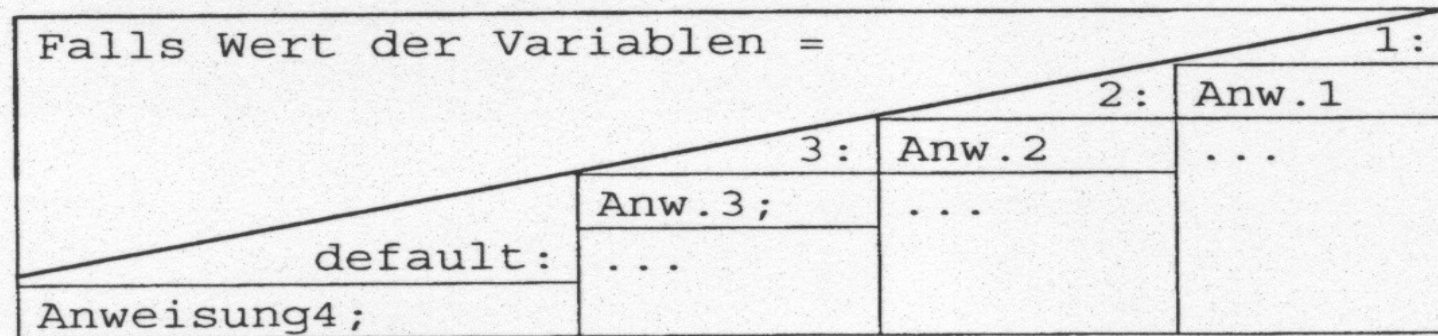
cout << "Tag der Woche [1..7]: ";
cin >> uiDayOfWeek;

if(uiDayOfWeek == 1 || uiDayOfWeek == 3 || uiDayOfWeek == 5)
{
    cout << "Es ist Montag, Mittwoch oder Freitag." << endl;
}
else
if(uiDayOfWeek == 2 || uiDayOfWeek == 4)
{
    cout << "Es ist Dienstag oder Donnerstag." << endl;
}
else
if(uiDayOfWeek == 6 || uiDayOfWeek == 7)
{
    cout << "Es ist Wochenende." << endl;
}
else
{
    cout << "Eine Woche hat nur 7 Tage ..." << endl;
}
```



# Mehrfache Selektion

## Struktogramm der mehrfachen Auswahl



**switch**(Wert)

{

**case** 1: Anweisung1; **break**;

**case** 2: Anweisung2; **break**;

**case** 3: Anweisung3; **break**;

**default**: Anweisung4;

}

# Mehrfache Selektion

```
unsigned int uiDayOfWeek;
```

```
cout << "Tag der Woche [1..7]: " ;  
cin >> uiDayOfWeek;
```

```
switch(uiDayOfWeek)
```

```
{
```

```
    case 1:
```

```
    case 3:
```

```
    case 5: cout << "Mo, Mi oder Fr" << endl;
```

```
        break;
```

```
    case 2:
```

```
    case 4: cout << "Di oder Do" << endl;
```

```
        break;
```

```
    case 6:
```

```
    case 7: cout << "Wochenende!" << endl;
```

```
        break;
```

```
    default: cout << "Eine Woche hat nur 7 Tage!" << endl;
```

```
}
```

# Funktionen

- begrenztes Vokabular (meist wenige Wörter, siehe C/C++)
- wohldefinierte Grammatik

Funktion ::= Typ FunktionsName '(' Parameterliste ')' FunktionsRumpf

Typ ::= „int“ | „double“ | „char“ | „void“ ...

FunktionsName ::= Bezeichner

Bezeichner ::= AlphabetZeichen {Zeichen}

ParameterListe ::= „“ | ParameterDefinition {',' ParameterDefinition}

ParameterDefinition ::= Typ Bezeichner

FunktionsRumpf ::= '{' {Anweisung ';' } '}'

EBNF – Extended Backus-Naur-Form

Datentyp Funktionsname(Parameter1, Parameter2, ...);

Var = Funktionsname(Parameter1, Parameter2, ...);

# Funktionen

Sinus-Funktion:  $\sin(\text{winkel})$

Definition:  $\sin(\alpha) = a/c$ ; // a Gegenkathete, c Hypothenuse

Näherungsformel für einen Einheitskreis:

$$\sin(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!} = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots = x - \frac{x^3}{6} + \frac{x^5}{120} - \dots$$

double sin(double x)

```
{
    double summe = 0.0;
    for(int i=0; i<unendlich; i++)
    {
        summe = summe + ...
    }
    return summe;
}
```

# Funktionen/Unterprogramme

Aufnahme des Nettopreises von Artikel 1  
Berechnung des Bruttopreises  
Ausgabe des Bruttopreises auf den Bildschirm  
Aufnahme des Nettopreises von Artikel 2  
Berechnung des Bruttopreises  
Ausgabe des Bruttopreises auf den Bildschirm  
Aufnahme des Nettopreises von Artikel 3  
Berechnung des Bruttopreises  
Ausgabe des Bruttopreises auf den Bildschirm

# Funktionen/Unterprogramme

```
const double cdMwSt = 1.19;
```

```
void main()
```

```
{
```

```
    double dArtikel1 = 12.90;
```

```
    double dArtikelBrutto1 = dArtikel1 * cdMwSt;
```

```
    cout << "Artikel 1 netto: " << dArtikel1  
        << ", brutto: " << dArtikelBrutto1 << endl;
```

```
    double dArtikel2 = 1.90;
```

```
    double dArtikelBrutto2 = dArtikel2 * cdMwSt;
```

```
    cout << "Artikel 2 netto: " << dArtikel2  
        << ", brutto: " << dArtikelBrutto2 << endl;
```

```
}
```

# Funktionen/Unterprogramme

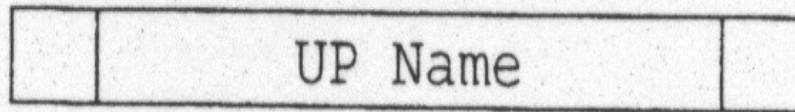
```
const double cdMwSt = 1.19;
```

```
void bearbeiteArtikel(int iInNr, double dInWert)
{
    double dArtikelBrutto = dInWert * cdMwSt;
    cout << "Artikel " << iInNr << " netto: " << dInWert
        << ", brutto: " << dArtikelBrutto << endl;
}
```

```
void main()
{
    bearbeiteArtikel(1, 12.90);
    bearbeiteArtikel(2, 1.90);
}
```

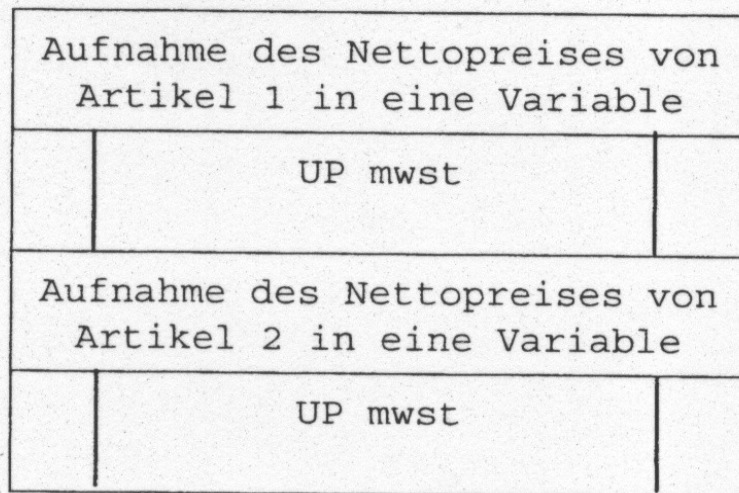
# Funktionen/Unterprogramme

## Struktogramm-Symbol

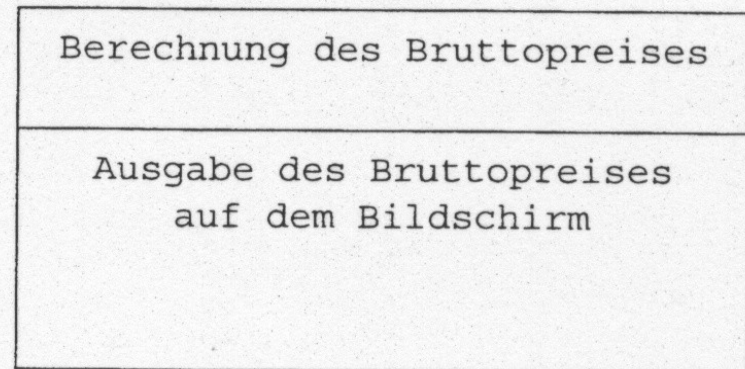


## Struktogramm des Hauptprogramms und des Unterprogramms

### Hauptprogramm

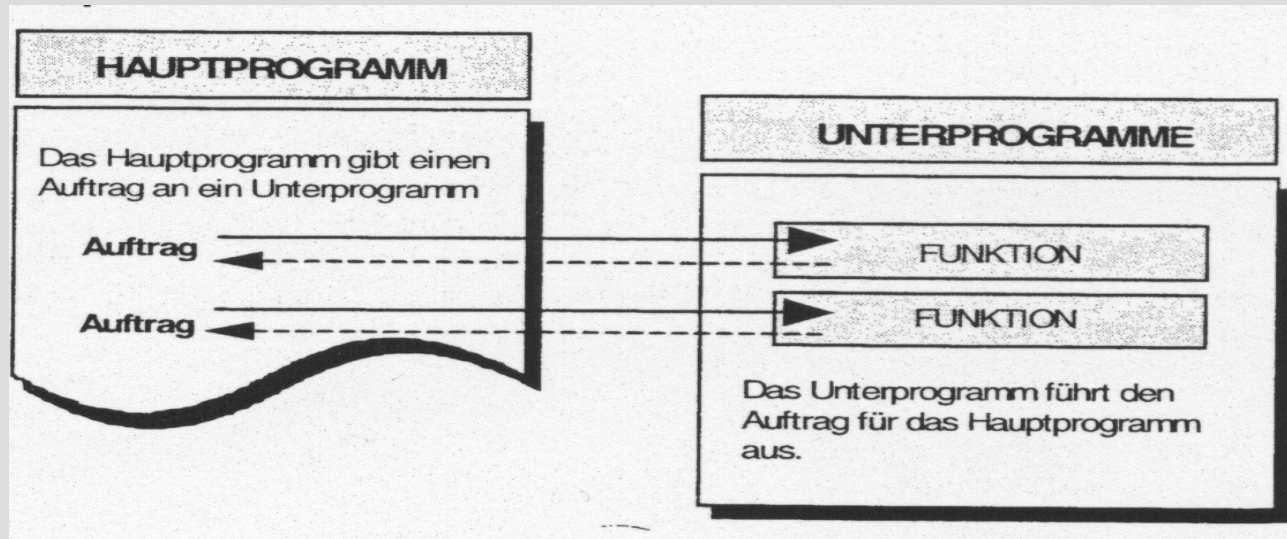


### Unterprogramm





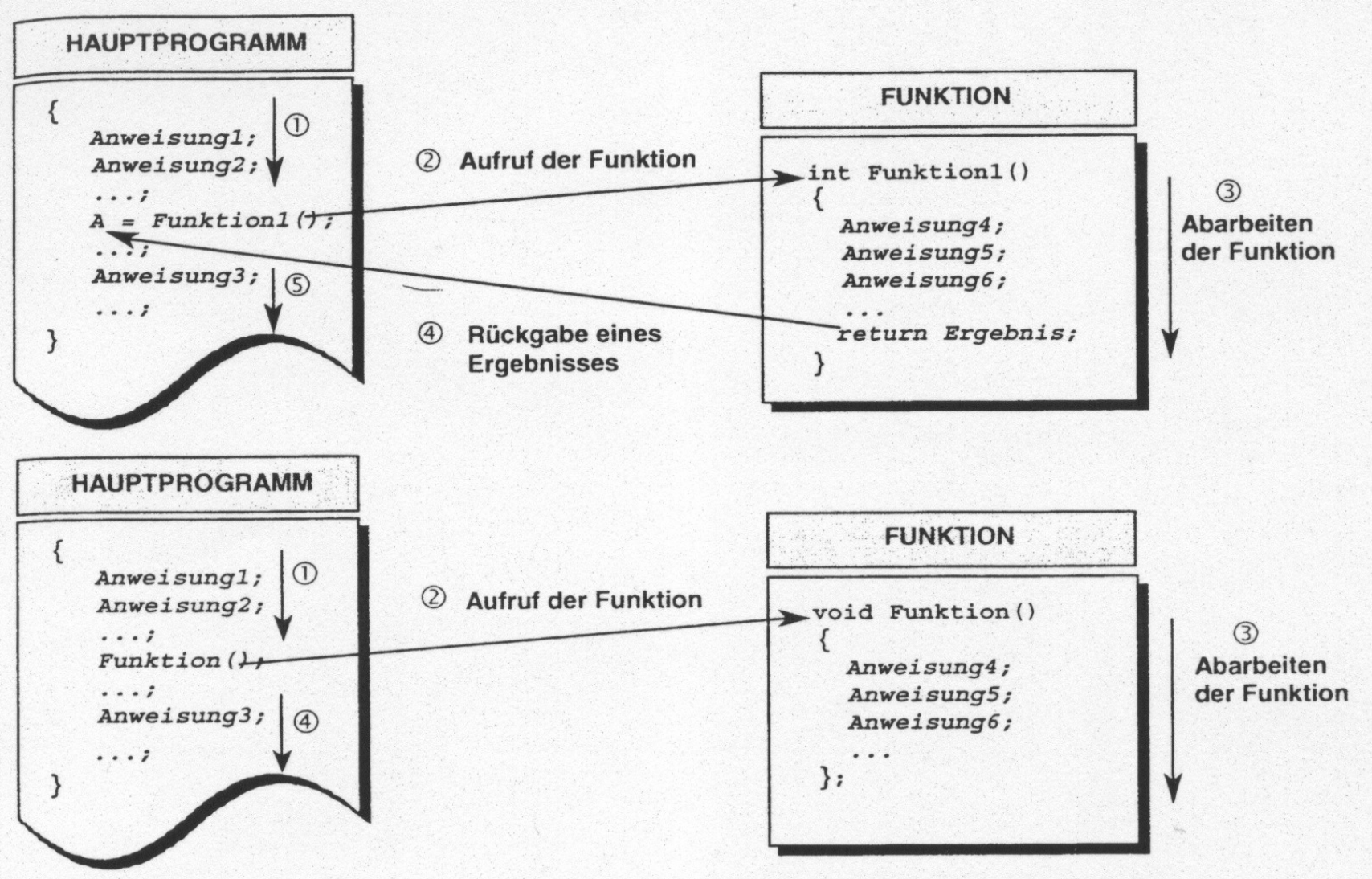
# Funktionen/Unterprogramme



```
double sin(double x) { ... }
```

```
int main()  
{  
    double dWert = sin(2.718);  
    return 0;  
}
```

# Funktionen/Unterprogramme



# Programmabschnitte

Präprozessoranweisungen

```
#include <stdio.h>
#define MOUNT_EVEREST 8848
```

Konstantendeklarationen

```
const int K = 17;
```

Variablendeklarationen

```
int preis, anzahl;
```

Funktionsdefinition

```
Funktion(Parameter)
```

Anweisungsteil

```
{
    Anweisung1;
    Anweisung2;
}
```

Hauptfunktion

```
int main()
```

Anweisungsteil

```
{
    Anweisung1;
    Anweisung2;
    return 0;
}
```

# Funktionen/Unterprogramme call by value

```
const double cdMwSt = 1.19;
```

```
void bearbeiteArtikel(int iInNr, double dInWert)
{
    double dArtikelBrutto = dInWert * cdMwSt;
    cout << "Artikel " << iInNr << " netto: " << dInWert
         << ", brutto: " << dArtikelBrutto << endl;
}
```

```
void main()
{
    bearbeiteArtikel(1, 12.90);
    bearbeiteArtikel(2, 1.90);
}
```

# Funktionen/Unterprogramme call by reference

```
const double cdMwSt = 1.19;
```

```
void berechneBrutto(double dInWert, double& rdOutWert)  
{  
    rdOutWert = dInWert * cdMwSt;  
}
```

```
void main()  
{  
    double dBrutto;  
  
    berechneBrutto(12.90, dBrutto);  
    cout << dBrutto << endl;  
    berechneBrutto(25.90, dBrutto);  
    cout << dBrutto << endl;  
}
```

# Funktionen/Unterprogramme call by reference (pointer)

```
const double cdMwSt = 1.19;
```

```
void berechneBrutto(double dInWert, double* pdOutWert)  
{  
    *pdOutWert = dInWert * cdMwSt;  
}
```

```
void main()  
{  
    double dBrutto;  
  
    berechneBrutto(12.90, &dBrutto);  
    cout << dBrutto << endl;  
    berechneBrutto(25.90, &dBrutto);  
    cout << dBrutto << endl;  
}
```

# Referenzen Zeiger

**&Variable** → ergibt **Speicheradresse der Variablen** und kann in einem Zeiger gespeichert werden

**int\*** Zeiger → **Zeigerdeklaration** auf eine Speicheradresse, deren Inhalt als **int** interpretiert wird.

**\*Zeiger** → **Dereferenzierung** eines Zeigers: Zugriff auf den Speicher**inhalt**.

Beispiel:

```
int zahl = 261;           // Variable zahl vom Datentyp int  
int* zeigerAufZahl = &zahl; // zeigerAufZahl wird die Speicheradresse  
                           // der Variablen zahl zugewiesen
```

```
cout << "Wert des Speicherinhaltes auf den der Zeiger zeigt: "  
      << *zeigerAufZahl << endl;
```

# Referenzen Zeiger

## Mögliche Organisation im Hauptspeicher

2 Byte **int** Variable „zahl“ im Speicher z.B. bei Adresse **0x1200cda0**:

**0x1200cda0: 0x05**

**0x1200cda1: 0x01**

Zusammengesetzt: **0x0105** =  $256+5 = 261$

Variable „zeigerAufZahl“ im Speicher, z.B. bei Adresse **0x1200cda2**

**0x1200cda2: 0xa0**

**0x1200cda3: 0xcd**

**0x1200cda4: 0x00**

**0x1200cda5: 0x12**

Zusammengesetzt: **0x1200cda0** → Adresse von zahl



# Referenzen Zeiger

```
int zahl = 254;
```

```
int* zeigerAufZahl = &zahl; // Ann.: zahl liegt auf 0x1200cda0
```

```
zahl++; // Welchen Wert ergibt *zeigerAufZahl?
```

```
zeigerAufZahl++; // Welchen Wert ergibt *zeigerAufZahl?
```

```
0x1200cda2: 0xa2
```

```
0x1200cda3: 0xcd
```

```
0x1200cda4: 0x00
```

```
0x1200cda5: 0x12
```

```
Zusammengesetzt: 0x1200cda2 → Adresse von zahl?
```

# Prototyping

Ein **Funktionsprototyp** hat keinen Funktionsrumpf.  
Entkopplung von **Funktionsdeklaration** und **Funktionsdefinition**.

```
int add(int iInSummand1, int iInSummand2); // Funktionsdeklaration
```

```
int main()
{
    int iSumme = add(3,4);
    return 0;
}
```

```
int add(int iInSummand1, int iInSummand2) // Funktionsdefinition
{
    ...
}
```

# Prototyping

## **Datei math.h:**

```
int add(int iInSummand1, int iInSummand2);
```

## **Datei math.cpp:**

```
#include "math.h"
```

```
int add(int iInSummand1, int iInSummand2)
{
    return iInSummand1 + iInSummand2;
}
```

## **Datei main.cpp:**

```
#include "math.h"
```

```
int main()
{
    int iSumme = add(3,4);
    return 0;
}
```

# Prototyping

**Datei math.h:**

```
#ifndef MATH_H  
#define MATH_H
```

```
#define fabs(x) ((x) < 0 ? (-x) : x)
```

```
int add(int iInSummand1, int iInSummand2);
```

```
#endif
```

# Iterationen - Schleifen

Anweisung1;  
Anweisung2;

Anweisung1;  
Anweisung2;

Anweisung1;  
Anweisung2;

Anweisung1;  
Anweisung2;

Anweisung1;  
Anweisung2

Anweisung1;  
Anweisung2

Anweisung1;  
Anweisung2;

## Wiederholungsanweisung

1. *Festgelegte Anzahl* der Wiederholungen
2. *Bedingte* Wiederholung
  - a. Kopfgetestete Wiederholung
  - b. Endegetestete Wiederholung

## Schema:

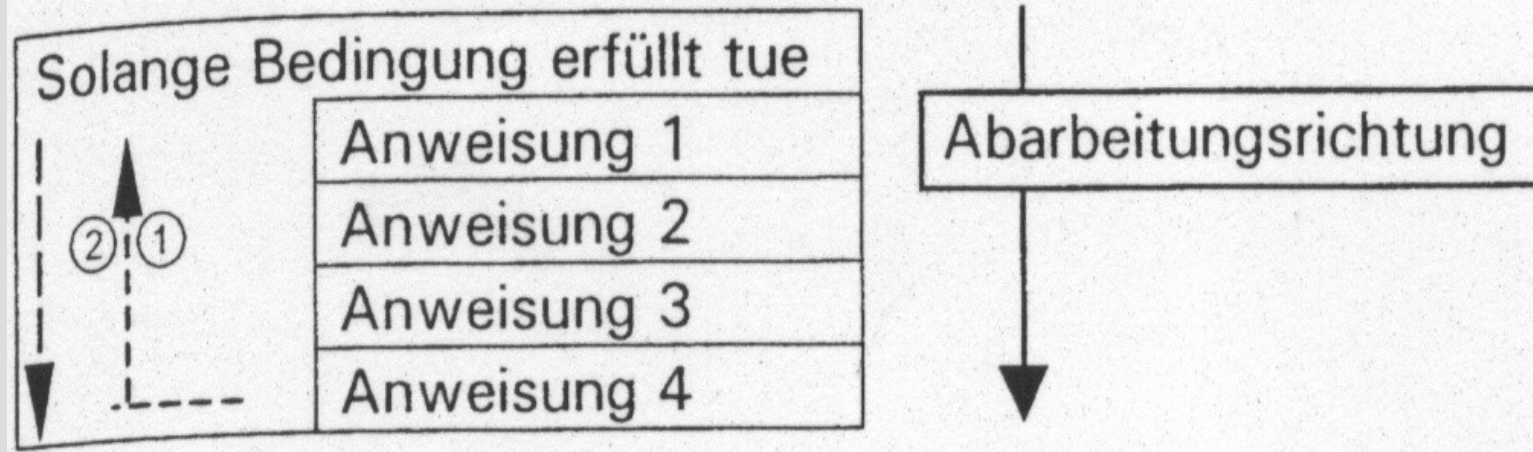
### Schleifensteuerung

Anweisung1;  
Anweisung2;

# Iterationen - Schleifen

## Kopfgetestete Schleife

### Struktogramm der kopfgetesteten Schleife



```
while(Bedingung erfüllt)
{
    Anweisung1;
    Anweisung2;
    Anweisung3;
    Anweisung4;
}
```

# Iterationen - Schleifen

## Kopfgetestete Schleife

```
#include <iostream>
using namespace std;

int main()
{
    int i = 0;
    int z = 65;                /* ASCII-Wert für 'A' */
    while(z != 69)            /* ASCII-Wert für 'E' */
    {
        i++;
        cout << „Anzahl der Schleifendurchläufe: “ << i << endl;
        cout << „Ende mit E. Taste: “;
        z = cin.get();
    }
    return 0;
}
```

# Iterationen - Schleifen

## Kopfgetestete Schleife

```
#include <iostream>
using namespace std;

int readInput(int& riInNr, char* acInText)
{
    cout << acInText << riInNr++ << endl;
    cout << „Ende mit E. Taste: “;
    int z = cin.get();
    return z;
}

int main()
{
    int i = 0;
    int z = 65;           /* ASCII-Wert für 'A' */
    while(z != 69)       /* ASCII-Wert für 'E' */
    {
        z = readInput(i, „Anzahl der Schleifendurchlaeufe: ");
    }
    return 0;
}
```



# Iterationen - Schleifen

## Kopfgetestete Schleife

```
#include <iostream>
using namespace std;

int readInput(int& riInNr, char* acInText)
{
    cout << acInText << riInNr++ << endl;
    cout << „Ende mit E. Taste: “;
    int z = cin.get();
    return z;
}

int main()
{
    int i = 0;
    while(readInput(i, „Anzahl der Schleifendurchlaeufe: ") != 69)
    {
    }
    return 0;
}
```

# Iterationen - Schleifen

## Kopfgetestete Schleife

```
#include <iostream>
using namespace std;

#define fabs(x) ((x) < 0? -(x) : (x))

double sqrt(double dInValue)
{
    double dAalt = dInValue;
    double dAneu = dInValue/2.0;
    while(fabs(dAalt - dAneu) > 0.0000001)
    {
        double dTmp = dAneu;
        dAneu = (dInValue/dAalt + dAalt)/2.0;
        dAalt = dTmp;
    }
    return dAneu;
}

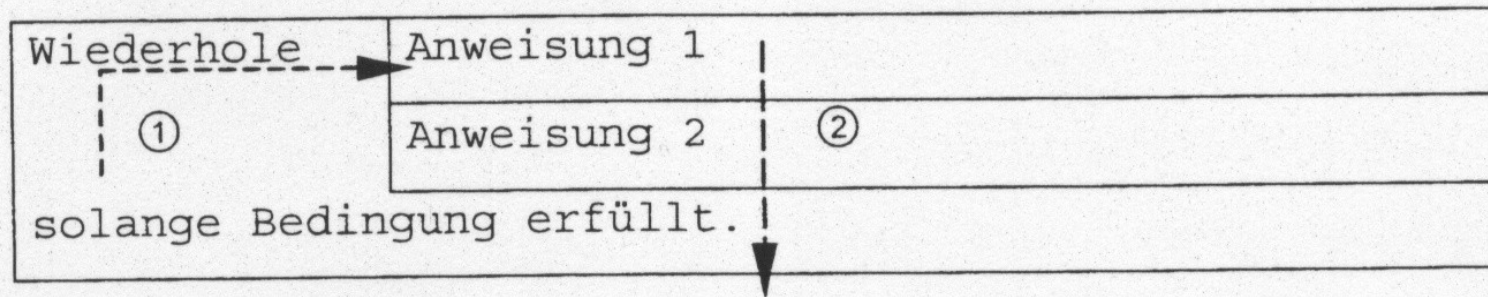
int main()
{
    double dZahl = 144.0;
    double dWurzel = sqrt(dZahl);
    cout << "Quadratwurzel von " << dZahl << " ist " << dWurzel << endl;
    return 0;
}
```

$$A_{neu} = \frac{\left( \frac{N}{A_{alt}} + A_{alt} \right)}{2}$$

# Iterationen - Schleifen

## Endegetestete Schleife

Struktogramm der endegetesteten Schleife



```
do
{
    Anweisung1;
    Anweisung2;
}
while(Bedingung erfüllt)
```

# Iterationen - Schleifen

## Endegetestete Schleife

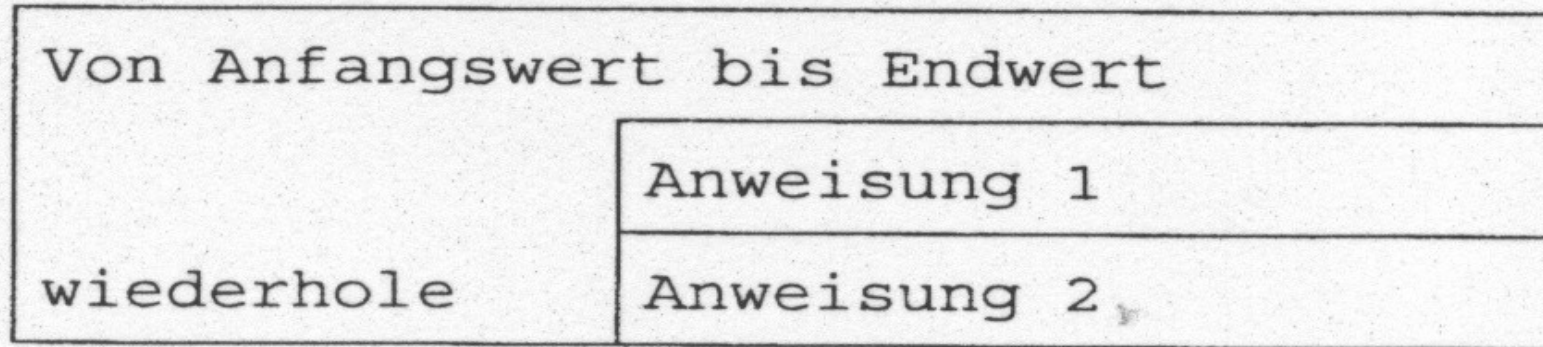
```
int wahlBox()
{
    int iEingabe;
    cout << "Willkommen in der Wahlkabine" << endl;
    do
    {
        cout << "Stimme fuer Bundeshorst: 1" << endl;
        cout << "Stimme fuer Schwanengesine: 2" << endl;
        cin >> iEingabe;
    } while(iEingabe < 1 || iEingabe > 2);
    return iEingabe;
}

void main()
{
    int z = 0;
    int iBundeshorst = 0;
    int iSchwanengesine = 0;
    do
    {
        int iStimme = wahlBox();
        switch(iStimme)
        {
            case 1: iBundeshorst++; break;
            case 2: iSchwanengesine++; break;
        }
        cout << "Wahlende (0)" << endl;
        cin >> z;
    } while(z != 0);
    cout << "Stimmen fuer Bundeshorst: " << iBundeshorst << endl;
    cout << "Stimmen fuer Schwanengesine: " << iSchwanengesine << endl;
}
```

# Iterationen - Schleifen

## Zählschleife

### Struktogramm der Zählschleife



```
for(Ausdruck1; Bedingung; Ausdruck2)
{
    Anweisung1;
    Anweisung2;
}
```

# Iterationen - Schleifen

## Zählschleife

```
#include <iostream>
```

```
int main()
{
    for(int z=0; z<=10; ++z)
    {
        std::cout << "z: " << z << std::endl;
    }
    return 0;
}
```

```
-----
#include <iostream>
```

```
int main()
{
    for(int z=10; z>=0; --z)
    {
        std::cout << "z: " << z << std::endl;
    }
    return 0;
}
```

# Iterationen - Schleifen

## Zählschleife

```
/* Summe der Zahlen von 1-100*/  
  
#include <iostream>  
  
int main()  
{  
    int iSumme = 0;  
    for(int z=1; z<=100; z++)  
    {  
        iSumme += z;  
    }  
    std::cout << "Summe von 1-100: " << iSumme << std::endl;  
    return 0;  
}
```

# Erwartungshorizont

## Stand heute

- Verständnis von Zahlensystemen, dezimal  $\leftrightarrow$  binär  $\leftrightarrow$  hexadezimal
- Zusammenhang von Datentyp **int** zu Binärdarstellung
- Begrenzte Darstellung von Zahlenräume, **int, long int, float, double**
- Unterschied zwischen **signed** und **unsigned** Datentypen
- Umwandlung (type cast) von Ganzzahl- und Fließkommatypen
- Deklaration und Definition von Variablen, Konstantendefinition
- Definition von Makros, Funktionsweise des Präcompilers
- Anwendung mathematischer Operatoren auf Variablen
- Umwandlung einer math. Formel in C-Syntax
- Unterschied lokale und globale Variablen, Sichtbarkeitsregeln
- Anwendung cout, cin
- Selektion mit **if, if-else, switch**, Kombination und Verschachtelung
- Definition von Funktionen mit und ohne Rückgabewert sowie mit Parameterliste
- Aufruf einer Funktion mit Variablen mittels call by value, call by reference
- Unterprogrammtechnik
- Verständnis von Referenzen und Zeigern sowie deren Zusammenhang
- **while**-Schleife, **do-while**-Schleife, **for**-Schleife



# Typvereinbarungen

- **einfache Datentypen**: char, int, float, double ...
- **eigene/zusammengesetzte Typdefinitionen**

```
typedef Datentyp NeuerTypName;
```

## Beispiel:

```
typedef float real;  
typedef int temperature;
```

## Aufzählung:

```
typedef enum  
{  
    Jan, Feb, Mrz, Apr, Mai, Jun, Jul, Aug, Sep, Okt, Sep, Nov, Dez  
} tMonat;
```

```
typedef enum {rot, gelb, gruen} tAmpel;
```

## Beispiel:

```
tAmpel AmpelVariable = rot;
```

# Typvereinbarungen

```
typedef enum {binary=2, octal=8, decimal=10, hex=16} tNumberSystem;

void printSystem(tNumberSystem eInSystem)
{
    switch(eInSystem)
    {
        case binary: cout << "Binaersystem: " << binary;
                    break;
        case octal:  cout << "Oktalsystem: " << octal;
                    break;
        case decimal: cout << "Dezimalsystem: " << decimal;
                    break;
        case hex:    cout << "Hexadezimalsystem: " << hex;
                    break;
        default:    cout << "Nicht unterstuetztes System!";
    }
    cout << endl;
}
```

# Array / Feld

Zusammenfassung von Variablen mit demselben Datentyp  
Karteikasten/Ordner  
Zugriff durch Indizierung

Beispiel:

```
int z0 = 31;
int z1 = 28;
int z2 = 31;
int z3 = 30;
int z4 = 31;
int z5 = 30;
int z6 = 31;
int z7 = 31;
int z8 = 30;
int z9 = 31;
int z10 = 30;
int z11 = 31;

int aiMonatstage[12];
-----
int aiMonatstage[] = {31,28,31,30,31,30,31,31,30,31,30,31};
cout << "Anzahl Tage im Monat Juli: " << aiMonatstage[6];

int iAnzahlTageJanuar = aiMonatstage[0];

for(int i=0; i < 12; i++)
{
    cout << "Monat " << i+1 << " hat " << aiMonatstage[i]
        << endl;
}
```

# Array / Feld

```
int aiMonatstage[12];

void initMonatstage(int iInJahr)
{
    for(int iMonat=0; iMonat<7; iMonat++)
    {
        aiMonatstage[iMonat] = 30 + (iMonat+1)%2;
    }
    for(int iMonat=8; iMonat<12; iMonat++)
    {
        aiMonatstage[iMonat] = 30 + iMonat%2;
    }
    aiMonatstage[1] = (iInJahr % 4 == 0) &&
        (iInJahr % 100 != 0 || iInJahr % 400 == 0)? 29 : 28;
}
```

# Dynamisches Array / Feld

Ein **dynamisches Array** wird zur **Laufzeit angelegt**, d.h. es wird im Speicherplatz bereitgestellt, der für die angegebene Anzahl an Arrayelementen hinreichend groß ist.

Der Programmentwickler muss sich selbst um die **explizite Freigabe** des reservierten Speichers kümmern.

## **Vorteil:**

Die Größe des Arrays kann so gewählt werden, wie es zur Laufzeit benötigt wird.

## **Nachteil:**

Es muss genauestens darauf geachtet werden, dass der Speicher wieder freigegeben wird.

```
int* aiMonatstage = new int[12]; // 12 Arrayelemente vom Typ int
```

```
.....
```

```
delete [] aiMonatstage;
```

# Strukturen

Strukturen fassen Daten mit unterschiedlichen Datentypen zusammen

z.B. Adresse:

```
struct tAdresse
{
    char          macVorname[20];
    char          macName[20];
    char          macStrasse[30];
    unsigned int  muiHausnummer;
    unsigned int  muiPostleitzahl;
    char *        macOrt;
};

tAdresse sMeineAdresse;
strcpy(sMeineAdresse.macVorname, "Heinz");
strcpy(sMeineAdresse.macName, "Mustermann");
strcpy(sMeineAdresse.macStrasse, "Karlsruher Strasse");
sMeineAdresse.muiHausnummer = 4;
sMeineAdresse.muiPostleitzahl = 76275;
strcpy(sMeineAdresse.macOrt, "Ettlingen");
```

# Strukturen

```
struct tAdresse
{
    char          macVorname[20];
    char          macName[20];
    char          macStrasse[30];
    unsigned int  muiHausnummer;
    unsigned int  muiPostleitzahl;
    char *        macOrt;
};

tAdresse* asAdressen;
asAdressen = new tAdresse[100];

strcpy(asAdresse[10].macVorname, "Heinz");
strcpy(asAdresse[10].macName, "Mustermann");
strcpy(asAdresse[10].macStrasse, "Karlsruher Strasse");
asAdresse[10].muiHausnummer = 4;
asAdresse[10].muiPostleitzahl = 76275;
strcpy(asAdresse[10].macOrt, "Ettlingen");

delete [] asAdressen;
```

# Sprünge - Vorsicht!

## continue

Beendigung des aktuellen Schleifendurchgangs

Anwendung nur bei Förderung der Übersicht

```
for(int i=0; i<20; i++)
{
    if(i % 2 == 0)
        continue;

    int z = i*2;
    cout << "z = " << z << endl;
}
```

```
for(int i=0; i<20; i++)
{
    if(i % 2 == 0)
    {
    }
    else
    {
        int z = i*2;
        cout << "z = " << z << endl;
    }
}
```



# Sprünge - Vorsicht!

## break

Beendigung der Schleife

Anwendung nur bei Förderung der Übersicht

```
for(int i=1; i<20; i++)
{
    if(i % 2 == 0)
        break;

    int z = i*2;
    cout << "z = " << z << endl;
}
```

```
for(int i=1; i<20 && i % 2 != 0; i++)
{
    int z = i*2;
    cout << "z = " << z << endl;
}
```

# Sprünge - Vorsicht!

## goto (vade retro satanis!)

Sprung an eine durch eine Marke definierte Position

Anwendung nur im absoluten Notfall! Am besten nie.  
*..... und so schon gar nicht!!!*

```
int i=0;

goto SprungMarke1;
SprungMarke3:
printf("Nie goto verwenden\n");
i++;
goto SprungMarke2;
SprungMarke1:
printf("absolut nie goto verwenden");
SprungMarke2:
if(i<10)
goto SprungMarke3;
```

```
int i=0;

cout << "absolut nie goto verwenden";
for( ; i<10; i++)
{
    cout << "Nie goto verwenden" << endl;
}
```

