

Programmieren in C: Elemente von C

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 1

Elemente von C

- In diesem Kapitel wollen wir zeigen,
 - ➡ **welche Schlüsselwörter ein C- Compiler verwendet**
 - ➡ **welche Zeichen, Namen, Datentypen und einfachen Operatoren ein C- Programm verwenden darf.**

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 2

Elemente von C: Zeichenvorrat (1)

- C verwendet die folgenden ➡ **druckbaren Zeichen**:
 - Die 26 Großbuchstaben des **englischen Alphabets**:
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
 - Die 26 Kleinbuchstaben des **englischen Alphabets**:
a b c d e f g h i j k l m n o p q r s t u v w x y z
 - Die 10 Ziffern:
0 1 2 3 4 5 6 7 8 9
 - 30 Sonderzeichen (incl. des Leerzeichens (*blank*)):
^ ! " % & / { ([] = } ? \ + * ~ # ' < > | , ; . : - _

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 3

Zeichenvorrat (2)

- C verwendet die folgenden ➡ **nichtdruckbaren Zeichen**:
 - Die 2 Tabulatoren:
 - * Horizontaler Tabulator (*horizontal tab (ASCII: HT)*)
 - * Vertikaler Tabulator (*vertical tab (ASCII: VT)*)
 - Die 2 Vorschubbefehle:
 - * Seitenvorschub (*form feed (ASCII: FF)*)
 - * Zeilenvorschub (*line feed (ASCII: LF)*)
 - Diese Zeichen und das Leerzeichen (*blank*) heißen auch
➡ **weiße Leerzeichen (white spaces)**.
Sie werden auf dem Bildschirm als Zwischenräume dargestellt.

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 4

Zeichenvorrat (3)

- C verwendet folgende ➡ **Schlüsselwörter**:

auto	break	case	char	const	continue
default	do	double	else	enum	extern
float	for	goto	if	int	long
register	return	short	signed	sizeof	static
struct	switch	typedef	union	void	unsigned
volatile	while				

- C unterscheidet **Groß- und Kleinschreibung!**
Die **Schlüsselwörter** müssen **klein geschrieben** werden!
- Die Schlüsselwörter sind reserviert. Sie dürfen **nicht für eigene Namen** verwendet werden.

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 5

Elemente von C: Namen (1)

- ➡ **Namen** können in C **Variable** und **Konstanten** bezeichnen.

Für Namen gelten folgende Regeln:

- Ein Name besteht **nur aus Buchstaben, Ziffern und dem Unterstrich _**.
Andere Sonderzeichen (^ & ...) sind nicht erlaubt.
- Ein Name darf **nicht mit einer Ziffer anfangen**.
- Namen dürfen **nicht mit Schlüsselwörtern** (wie z.B. **int**, **main**, **void**, **return** ...) **übereinstimmen**.
- Die zulässige Länge von Namen hängt vom Compiler ab.

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 6

Elemente von C: Namen (2)

- ➔ **Namen** in C können **Variable** und **Konstanten** bezeichnen.
Man unterscheidet **lokale** und **globale Namen**.
Werden Namen **innerhalb desselben Programms** verwendet, so nennt man sie ➔ **lokal**.
Werden Namen an **andere Programme übergeben**, so nennt man sie ➔ **global**.
- ➔ **Globale Namen** sind verschieden, wenn sie sich in **mindestens einem der ersten 6 Zeichen** unterscheiden.
Groß- und Kleinschreibung **ist dabei ohne Bedeutung**:
Beispiel:
`Variable = variable = VARIABLE ≠ vAria1`

19.11.2004



Namen (3)

- Namen innerhalb desselben Programms** nennt man ➔ **lokale Namen**.
- ➔ Nach **ANSI C** sind **lokale Namen** verschieden, wenn sie sich in **mindestens 1 der ersten 31 Zeichen** unterscheiden (signifikante Länge).
Groß- und Kleinschreibung **ist von Bedeutung**:
Beispiel:
`Variable ≠ variable ≠ VARIABLE ≠ vAriable`
`Donaudampfschiffahrtsgesellschaftskapitaen =`
`DonaudampfschiffahrtsgesellschaftskapitAen =`
`Donaudampfschiffahrtsgesellschaftskapit`
- In **MSVC++ 6.0** sind die **ersten 247 Zeichen** eines Namens **signifikant**.

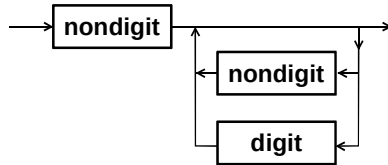
19.11.2004



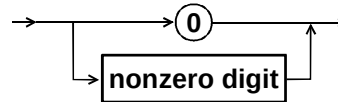
Namen (4)

- Welche Namen **erlaubt** sind, lässt sich übersichtlich als **Blockschaltbild** darstellen

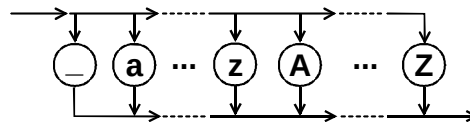
1. **identifier** (Name):



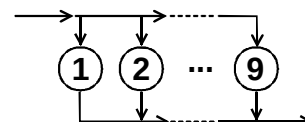
3. **digit** (Ziffer):



2. **nondigit** (keine Ziffer):



4. **nonzero digit** (keine 0-Ziffer):



Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 9

19.11.2004



Namen in C (1)

Welche der angegebenen lokalen Namen
sind in C gültig und warum?

FHH_EA1	gültig
Fhh/EA1	ungültig: / ist falsch
1.Semester	2* ungültig: beginnt mit Zahl, Punkt ist falsch
_1Semester	gültig: _ als Anfangszeichen erlaubt
Semester 1	ungültig: Name enthält Leerzeichen
SeMeSteR_1	gültig: Groß/Kleinschreibung zulässig
Autobus	gültig
auto	ungültig: Schlüsselwort
Auto	gültig: Groß-Kleinschreibung wird unterschieden
Groß_S	ungültig: ß

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 10

19.11.2004



Namen in C (2)

Welche der dargestellten Namen sind als **globale Namen** bzw. als **lokale Namen** (ANSI C-Standard) gleich?

1. FHH_EA1
2. Fhh_EA2
3. Fachhochschulstudentenbibliotheksausweiskontrolle
4. Fachhochschulstudentenbibliotheksausweis
5. Fachhochschulstudentenbibliothek1
6. FachHochschulStudentenBibliothek
7. Fachhochschulstudenten

globale Namen: gleich sind: 1. & 2., sowie 3., 4., 5., 6. & 7.

lokale Namen: gleich sind: 3. & 4.

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 11

Elemente von C: Datentypen (1)

- C verwendet folgende ➡ **Datentypen:**

Elementare Datentypen:

char Byte (*character*),
kann ein Zeichen aus dem Zeichensatz enthalten

int ganze Zahl (*integer*),
natürliches Format des verwendeten Prozessors

float einfach genaue Gleitkommazahl (*floating point*)

double doppelt genaue Gleitkommazahl (*double precision*)

Varianten:

short kurz

long lang

signed vorzeichenbehaftet (2-Komplementzahl)

unsigned vorzeichenlos (positive Zahl)

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 12

Datentypen (2)

- Es gelten die folgenden Vereinbarungen:
- ➔ Nach **short** oder **long** darf ein **int** eingefügt werden:
`short = short int` `long = long int`
- ➔ **short**, **long** und **char** vereinbaren vorzeichenbehaftete Zahlen:
`short = signed short` `short = signed short int`
`long = signed long` `long = signed long int`
- ➔ Positive Zahlen müssen stets als solche vereinbart werden: `unsigned short` `unsigned long`
- ➔ Die Gleitkommazahlen **float**, **double** und **long double** sind stets vorzeichenbehaftet.
- ➔ Die Länge **int**-Datentypen hängt vom Prozessor ab:
short- und **long**-Zahlen sind in der Regel 16 bzw. 32 bit breit.

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
 Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
 K4_C-Elemente.ppt, Seite 13

Datentypen in MSVC++ 6.0

- ➔ MSVC bietet zusätzlich das Format **hyper** (64-Bit-Integer) an. Es wird auch als `__int64` bezeichnet.
- ➔ Bevor man den Datentyp **hyper** verwendet muss man die Datei **rpc.h** mit `#include <rpc.h>` einbinden.
- ➔ Die genauen Definitionen der signifikanten Stellen und der Maximal- und Minimalwerte der Datentypen befinden sich in den Dateien **limits.h** und **float.h**

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
 Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
 K4_C-Elemente.ppt, Seite 14

Genauigkeit der Datentypen in MSVC++ 6.0

Datentyp	Wortlänge / [Bit]	Zahlbereich	Dezimalstellen
unsigned char	8	0 .. 255	$0 \dots 2^8 - 1$
signed char	8	-128 .. 127	$-2^7 \dots 2^7 - 1$
unsigned int	32	0 .. 4 294 967 295	$0 \dots 2^{32} - 1$
signed int	32	-2 147 483 648 .. 2 147 483 647	$-2^{31} \dots 2^{31} - 1$
unsigned short	16	0 .. 65 535	$0 \dots 2^{16} - 1$
signed short	16	-32 768 .. 32 767	$-2^{15} \dots 2^{15} - 1$
unsigned long	32	0 .. 4 294 967 295	$0 \dots 2^{32} - 1$
signed long	32	-2 147 483 648 .. 2 147 483 647	$-2^{31} \dots 2^{31} - 1$
unsigned hyper ¹	64	0 .. 18446744073709551616	$0 \dots 2^{64} - 1$
signed hyper ¹	64	-9223372036854775808 .. +9223372036854775807	$-2^{63} \dots 2^{63} - 1$
float	32	ca. $1.18 \cdot 10^{-38} \dots 3.4 \cdot 10^{38}$	6
double	64	ca. $2.23 \cdot 10^{-308} \dots 1.8 \cdot 10^{308}$	15
long double	64	ca. $2.23 \cdot 10^{-308} \dots 1.8 \cdot 10^{308}$	15

¹ Für das Format **hyper** muss in MSVC++ 6.0 `#include <rpc.h>` eingebunden sein!

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 15

Elemente von C: arithmetische Operatoren (1)

- C verwendet u.a. die folgenden **elementaren Zuweisungen** und **arithmetischen Operatoren**:

➡ **Zuweisung: =** **Zuweisungsrichtung beachten!**

Beispiel:

`b = a;` /* Nach der Operation: **b = a** */
b ← a

➡ **Addition: +** **Summe der Operanden**

Beispiel:

`s = e + f;` /* Nach der Operation: **s ← Summe aus e und f** */

➡ **Subtraktion: -** **Differenz der Operanden**

Beispiel:

`d = y - x;` /* Nach der Operation: **d ← Differenz aus y und x** */

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 16

arithmetische Operatoren (2)

- C verwendet u.a. die folgenden **elementaren arithmetischen Operatoren**:

➔ **Multiplikation: *** **Produkt der Operanden**

Beispiel:

$p = g * h;$ /* Nach der Operation: $p \leftarrow$ Produkt aus g und h */

➔ **Division: /** **Quotient der Operanden**

Beispiel:

$q = z / n;$ /* Nach der Operation: $q \leftarrow$ Quotient aus z und n */

➔ **Divisionsrest: %** **Rest der Division (modulo)**

Beispiel:

$r = w \% t;$ /* Nach der Operation: $r \leftarrow$ Rest der Division w / t */



arithmetische Operatoren (3)

- Es gelten die folgenden Regeln:

➔ **Vorrangregel: Punktrechnung geht vor Strichrechnung,**
wenn die Reihenfolge der Operationen nicht durch die
runde Klammern () verändert wird

Beispiel:

$y = (a + b) * c;$ /* Die Summe aus a und b wird mit c multipliziert */

➔ **Typregel 1: Die Operanden der Addition (+),**
Subtraktion (-), Multiplikation (*) und Division (/) dürfen
beliebige ganze Zahlen oder Gleitkommazahlen sein.
Auch Konstanten sind zulässig.

Beispiel:

$y = (a + 3.4) / 7;$ /* Variable a , Gleitkommakonstante 3.4 ,
ganzzahlige Konstante 7 */



arithmetische Operatoren (4)

- Es gelten die folgenden Regeln:

➔ **Typregel 2:** Die Operanden des **Modulo-Operators (%)** **müssen ganzzahlige Konstanten** oder **Variablen** sein.

Beispiel:

```
y = (a + 123) % 4; /* ganzzahlige Variable a,  
                  ganzzahlige Konstanten: 123, 4 */
```

➔ **Gültigkeit:** Wenn ein Zwischenergebnis oder das Endergebnis den **gültigen Wertebereich** verlässt, **ist das Ergebnis undefiniert**.

Diese Fehler werden bei vielen C-Entwicklungsumgebungen nicht oder nicht immer erkannt!

19.11.2004



Hinweise zu den C-Programmbeispielen in der Vorlesung

- Die C-Programmbeispiele in der Vorlesung **erfüllen nicht immer die Vorgaben für die C-Programme im Praktikum**.
- Um eine übersichtliche Darstellung auf 1 Folie zu erreichen, mussten gelegentlich notwendige Bildschirmausdrucke etc. weggelassen werden.
- In den Aufgaben zur Einarbeitung ins Praktikum finden Sie Beispiele, die den Vorgaben genügen.

19.11.2004



Zweites C-Programm: DM → € umrechnen

- Wir lernen die **Anweisungen** `float` und `scanf`, und die **Operatoren** `=` und `/` kennen

```

/* DM -> Euro Umrechnung */
/* Datei: dm_euro.c */
/* U. Vogelsang, HAW, FB E&I 1.10.99 */
#include <stdio.h>
int main (void)
{
    float DM, Euro, Umrechnungsfaktor_DM_Euro; /* Vereinbarung von
                                                    Variablen */
    printf("Umrechnungskurs eingeben 1 Euro = ?? DM: ");
    scanf("%f", &Umrechnungsfaktor_DM_Euro); /*Einlesen:
                                                    Umrechnungskurs ... */

    printf("Betrag in DM eingeben: ");
    scanf("%f", &DM); /* Einlesen: Betrag in DM */
    Euro = DM / Umrechnungsfaktor_DM_Euro; /* Euro-Betrag berechnen */
    printf("Betrag in Euro: %f \n", Euro);
    return 0;
}

```

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 21

19.11.2004

2. Programm: DM → € umrechnen (1)

```

#include <stdio.h>
int main (void)
{
    float DM, Euro, Umrechnungsfaktor_DM_Euro; /* Vereinbarung ... */
    printf("Umrechnungskurs eingeben 1 Euro = ?? DM: ");
    scanf("%f", &Umrechnungsfaktor_DM_Euro); /*Einlesen:Umr ... */
    printf("Betrag in DM eingeben: ");
    scanf("%f", &DM); /* Einlesen: Betrag in DM */
    Euro = DM / Umrechnungsfaktor_DM_Euro; /* Euro-Betrag berechnen */
    printf("Betrag in Euro: %f \n", Euro);
    return 0;
}

```

- Die Anweisung `float DM, ...;` vereinbart die drei **32-Bit Gleitkommavariablen** `DM`, `Euro`, `Umrechnungsfaktor_DM_Euro` im Zahlbereich $\pm 3.4 \cdot 10^{-38}$... $\pm 3.4 \cdot 10^{38}$.

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 22

19.11.2004

2. Programm: DM → € umrechnen (2)

```
#include <stdio.h>
int main (void)
{
    float DM, Euro, Umrechnungsfaktor_DM_Euro; /* Vereinbarung ..*/
    printf("Umrechnungskurs eingeben 1 Euro = ?? DM : ");
    scanf("%f", &Umrechnungsfaktor_DM_Euro); /*Einlesen: Umr ... */
    printf("Betrag in DM eingeben: ");
    scanf("%f", &DM); /* Einlesen: Betrag in DM */
    Euro = DM / Umrechnungsfaktor_DM_Euro; /* Euro-Betrag berechnen */
    printf("Betrag in Euro: %f \n", Euro);
    return 0;
}
```

- Die Anweisung `scanf("%f", &Umr...);` liest eine Gleitkommazahl ein und weist sie der 32-Bit Gleitkomma-variablen `Umrechnungsfaktor_DM_Euro` zu.
- Bei jedem Einlesen muss dem Variablennamen der Adressoperator `&` vorangestellt werden.

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 23

19.11.2004

2. Programm: DM → € umrechnen (3)

```
#include <stdio.h>
int main (void)
{
    float DM, Euro, Umrechnungsfaktor_DM_Euro; /* Vereinbarung ..*/
    printf("Umrechnungskurs eingeben 1 Euro = ?? DM: ");
    scanf("%f", &Umrechnungsfaktor_DM_Euro); /*Einlesen: Umr ... */
    printf("Betrag in DM eingeben: ");
    scanf("%f", &DM); /* Einlesen: Betrag in DM */
    Euro = DM / Umrechnungsfaktor_DM_Euro; /* Euro-Betrag berechnen */
    printf("Betrag in Euro: %f \n", Euro);
    return 0;
}
```

- Der Operator `/` berechnet den Quotienten der Variablen `DM` und `Umrechnungsfaktor_DM_Euro`
- Der Operator `=` weist der Variablen `Euro` den Quotienten zu.

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 24

19.11.2004

Fehlerbeseitigung (*debugging*) in der Software

- Die Fehler in C-Programmen werden in der Regel in **3 nacheinander folgenden Schritten beseitigt**:
 1. **Syntaxfehler** mit Hilfe der **Fehlermeldungen**, die nach der **Kompilierung** angezeigt werden
 2. **Programmcodefehler** mit Hilfe der **schrittweisen Ausführung des Programms** (*single step mode*)
 3. **Algorithmus- oder mathematische Fehler** während der schrittweisen Ausführung des Programms und mit dem **Nachrechnen bzw. optischen Verifizieren der Ergebnisse** in jedem Programmschritt.

19.11.2004



Typische Fehlerarten

- Typische **Syntaxfehler** sind:
 - ➡ Vergessene Klammern
 - ➡ zu viele oder zu wenige Semikolons
 - ➡ fehlende Präprozessoranweisungen (#) ...
- Typische **Programmcodefehler** sind:
 - ➡ Fehlerhafte Operanden
 - ➡ Fehler am Anfang / Ende einer Schleife
 - ➡ Zeigerfehler ...
- Typische **Algorithmus-/mathematische Fehler** sind:
 - ➡ ungeeignete mathematische Formel programmiert
 - ➡ Algorithmus liefert Fehler für große / kleine Fehler oder konvergiert nicht
 - ➡ Sonderfälle wurden nicht berücksichtigt ...

19.11.2004



Programmcodes- und Algorithmusfehler

- Häufig lassen sich die Fehlerarten

➡ **Programmcodesfehler**

➡ **Algorithmus-/mathematische Fehler**

nicht exakt gegeneinander abgrenzen.

- Typische **Syntaxfehler und deren Beseitigung** haben wir exemplarisch in vorherigen Kapitel behandelt.

- Im Folgenden stellen wir am **2. Programmbeispiel** exemplarisch die **Beseitigung von Programmcodes- und Algorithmusfehlern** vor.

Beispiel: Im 2. Programmbeispiel wurde der **€-Umrechnungsfaktor falsch interpretiert: Operator * statt /**.

Euro = DM * Umrechnungsfaktor_DM_Euro;

Dieser Fehler lässt sich **nur durch Nachrechnen** beseitigen!

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 27

19.11.2004

Beispiel: Programmcodes- / Algorithmusfehler

- Gibt man den Umrechnungsfaktor so ein, **wie er bei der Eingabeaufforderung vorgegeben** ist, so ist die Umrechnung falsch!

```

/* DM -> Euro Umrechnung Erläuterung */
/* Datei: DM_Euro_falsch.c Dateiname */
/* U. Vogelsang, HAW, FB ESI 13.9.2003 Verfasser, etc. */
#include <stdio.h> /* Bibliothek fuer die Ausgabe */
/* Deklaration des Hauptprogramms */
int main (void) /* Hauptprogrammanfang */
{
    float DM, Euro, Umrechnungsfaktor_DM_Euro; /* Vereinbarung von Variablen */
    printf("Umrechnungsfaktor eingeben: 1 Euro = ?? DM: "); /* Ausgabe: Umrech... */
    scanf("%f", &Umrechnungsfaktor_DM_Euro); /* Eingabe: Umrechnungsfaktor... */
    printf("Betrag in DM eingeben: "); /* Ausgabe: Betrag... */
    scanf("%f", &DM); /* Eingabe: Betrag in DM */
    Euro = DM * Umrechnungsfaktor_DM_Euro; /* Euro-Betrag berechnen */
    printf("Betrag in Euro: %f\n", Euro); /* Ausgabe: Betrag... */
    return 0; /* Abschluss des Hauptprogramms */
}
  
```

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 28

19.11.2004

- Der Fehler: `Euro = DM * Umrechnungsfaktor_DM_Euro ;`
kann bei der Kompilierung **nicht erkannt werden**.

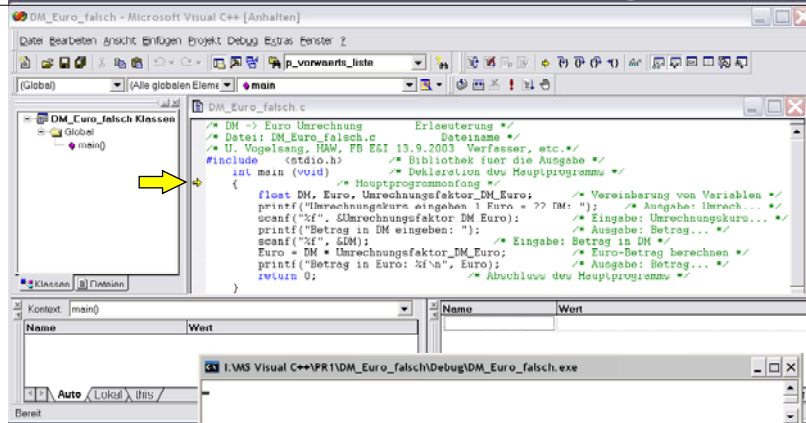


- Zur Fehlererkennung kann man das **Programm schrittweise ausführen** und Zwischenergebnisse nachrechnen.
- Die **schrittweise Programmausführung** kann man auch mit der Tastenkombination **Strg(ctrl)+F10** starten.



Beispiel: Programmcode- / Algorithmusfehler (3)

- Mit der Tastenkombination **Strg(ctlr)+F10** wird die Programmausführung am **Cursor im Textfenster** angehalten. Die markierte Programmzeile ist **noch nicht ausgeführt**: **MSVC** markiert sie mit einem **gelben Pfeil**.

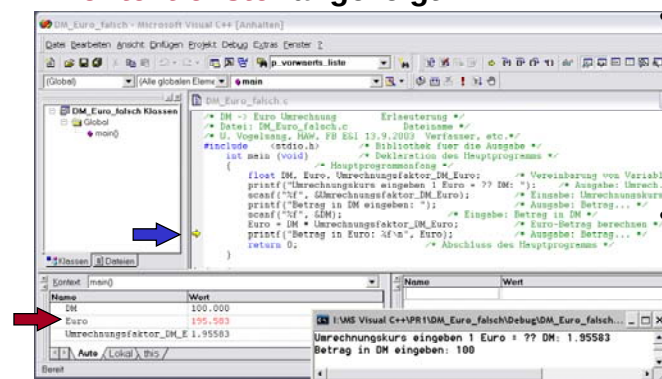


Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd

Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 31

Beispiel: Programmcode- / Algorithmusfehler (4)

- Das **Kontext-Fenster** kann ggf. mit **Alt+4** oder **Ansicht → Debug-Fenster → Variablen** geöffnet werden.
- Wenn man mit **F10** oder mit **F11** das Programm ausführt, werden **bei jedem Schritt die Werte der Variablen** im **Kontextfenster** angezeigt.



- Im **letzten Schritt** **aktualisierte Werte** erscheinen **rot**.
- Das **Nachrechnen** zeigt, dass der **Umrechnungsfaktor falsch** ist.

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd

Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 32

Programmdebug: Haltepunkte (*breakpoints*) (1)

- Ein Anfängerfehler ist eine **falsche Formatangabe beim Einlesen von Daten** mit **scanf (..)**. Die Daten im Computer sind dann falsch und alle berechneten Ergebnisse ebenfalls.

➔ **Der C-Compiler kann diesen Fehler nicht erkennen.**

Deshalb muss man im Debugmodus überprüfen, ob die Daten richtig eingelesen wurden.

- In diesem Fall und bei umfangreichen Programmen ist der Debug im Einzelschrittbetrieb (*single step*) zu langwierig.
- Man arbeitet besser mit **Haltepunkten**.

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 33

Programmdebug: Haltepunkte (*breakpoints*) (2)

- Der folgende Programmausschnitt enthält in **scanf (..)** eine falsche Formatangabe: **%d** statt **%f**.
- Diesen Fehler kann ein **C-Compiler nicht erkennen**.

/ ... (vorhergehende Programmzeilen) */*

```
int main (void) /* Deklaration des Hauptprogramms */
{              /* Hauptprogrammanfang */
    float DM, Euro, Umrechnungsfaktor_DM_Euro; /* Variable */
    printf("Umrechnungskurs eingeben 1 Euro = ?? DM: ");
    /* Ausgabe: Umrechnungskurs */
```

➔ **!!! falsche Formatangabe: %d statt %f in der nächsten Zeile: */**

```
scanf ( "%d", &Umrechnungsfaktor_DM_Euro);
/* Eingabe: Umrechnungskurs ... */
```

/ ... (weitere Programmzeilen) */*

19.11.2004

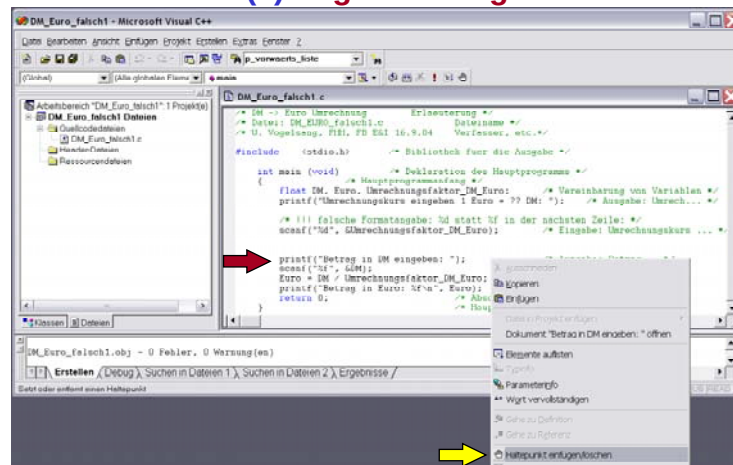
Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 34

Programmdebug: Haltepunkte (*breakpoints*) (3)

- Das Programm enthält in `scanf (..) %d` statt `%f`.
- Man fügt im Debugmodus einen Haltepunkt an der auf `scanf (..)` folgende Programmzeile ein.



Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd

Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 35

Programmdebug: Haltepunkte (*breakpoints*) (4)

- Das vorherige Programm enthält in `scanf (..)` eine falsche Formatangabe: `%d` statt `%f`.
- Nach dem Compilieren (**Strg+F7**) fügt man im **Debugmodus** einen **Haltepunkt** an der auf `scanf (..)` folgenden Programmzeile ein:
 - ➔ Im Quellcodefenster Cursor auf die Zeile setzen, an der das Programm angehalten werden soll.
 - ➔ Rechte Maustaste betätigen.
 - ➔ **Haltepunkt einfügen/löschen** auswählen.
 - ➔ Programm mit **F5** starten.
 - ➔ Nach der Eingabe des Umrechnungskurses hält das Programm **vor der Ausführung der Haltepunktzeile** an.

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd

Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 36

Programmdebug: Haltepunkte (*breakpoints*) (5)

- Der Wert von `Umrechnungsfaktor_DM_Euro` ist wegen des **Formatfehlers falsch eingelesen**.

The screenshot shows the Microsoft Visual C++ IDE with a C program named `DM_Euro_falsch1.c`. The code includes `<stdio.h>` and defines a `main` function. It declares `float DM, Euro, Umrechnungsfaktor_DM_Euro;` and prints a prompt for the exchange rate. A red arrow points to a comment indicating a format error: `/* !!! falsche Formatangabe: %d statt %f in der nächsten Zeile: */`. The code uses `scanf("%d", &Umrechnungsfaktor_DM_Euro);` to read the exchange rate. The debug window shows the variable `Umrechnungsfaktor_DM` with a value of `1.40130e-045`, which is incorrect. The console output shows the prompt `Umrechnungskurs eingeben 1 Euro = ?? DM: 1.95583` and the user input `1.95583`.

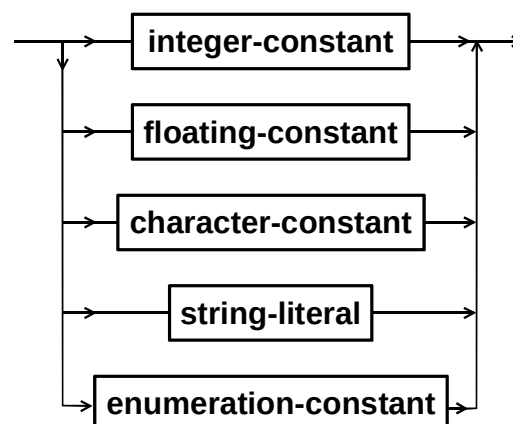
Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd

Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 37

Elemente von C: Konstanten

- Welche Konstanten **erlaubt** sind, entnehmen wir dem **Blockschaltbild**.

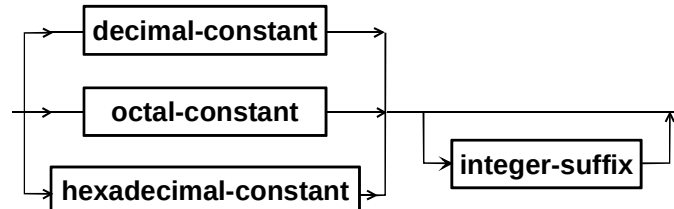
1. **constant** (Konstante):



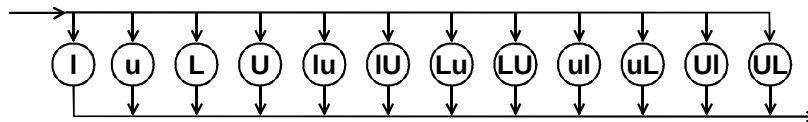
Integer-Konstanten (1)

- Welche Integer-Konstanten **erlaubt** sind, lässt sich aus den folgenden ➡ **Blockschaltbildern** entnehmen

2.1 Integer-constant (ganzzahlige Konstante):



2.2 Integer-suffix (Ganzzahl-Nachsilbe):



Integer-Konstanten (2)

- Für Integer-Konstanten (ganzzahlig) gelten die folgenden Vereinbarungen:

- ➡ Eine **Dezimalkonstante** (Basis 10) beginnt **nie mit einer 0**.
Sie enthält die Ziffern von **0 .. 9**.
- ➡ Eine **Oktalkonstante** (Basis 8) beginnt **immer mit einer 0**.
Sie enthält die Ziffern von **0 .. 7**.
- ➡ Eine **Hexadezimalkonstante** (Basis 16) beginnt **immer mit dem Präfix 0X oder 0x**.
Sie enthält die Ziffern von **0 .. 9** und **a, b, c, d, e, f**.

Integer-Konstanten (3)

- Die **Ganzzahl-Nachsilbe** (*integer-suffix*) beeinflusst die **Speicherung von Konstanten**:
 - ➔ **..L** oder **..l** bewirkt, dass die Konstante als ➔ **long** (32-Bit Zahl) gespeichert wird.
 - ➔ **..U** oder **..u** veranlasst, dass die Konstante als ➔ **unsigned** (positive Zahl) gespeichert wird.
 - ➔ Man darf die Nachsilben **L** und **U** in beliebiger Reihenfolge und in Groß- und Kleinschrift verwenden.

Beispiele:

Dezimalkonstanten: **1**, **-987**, **987u**, **1234567UL**, **-352l**

Oktalkonstanten: **0**, **0345**, **0774u**, **077777LU**

Hexadezimalkonstanten: **0x1357**, **0x0AFFEu**, **0xcaffeeL**

19.11.2004



Integer-Konstanten (4)

- Ist **keine** Ganzzahl-Nachsilbe (*integer-suffix*) angegeben, so wird der **Datentyp von links jeweils dem 1. passenden Typ aus den folgenden Listen** zugeordnet:
 - dezimale Konstante: ➔ **int**, **long**, **unsigned long**
 - hexadezimale oder oktale Konstante: ➔ **int**, **unsigned int**, **long**, **unsigned long**
- In **printf** lautet die Typangabe für eine ➔ **unsigned int-Variable u**

Beispiel:

```
#include <stdio.h> /* Bibliothek fuer die Ausgabe */
int main (void)
{
    unsigned int  unsigned_int = 65535 ;
    printf (" unsigned_int = %u", unsigned_int ) ;
    return 0 ;
}
```

19.11.2004





Zahlenformate in C

Welche Zahl gibt das folgende Programm für **x** aus?

Begründung?

Wie können Sie ein korrektes Ergebnis erhalten?

```
#include <stdio.h>          /* Bibliothek fuer die Ausgabe */
int main (void)
{
    short a = 32767, b = 1, x ;
    x = a + b ;
    printf ( "a + b = %hd \n", x ) ;
    return 0 ;
}
```



Zahlenformate in C

Ausgabe:

a + b = -32768

Begründung: **Zahlbereichsüberschreitung**

$32767_{10} = 0111\ 1111\ 1111\ 1111_2 + 1_2 = 1000\ 0000\ 0000\ 0000_2$
 $= -32768_{10}$ 💣

Korrekte Ausgabe

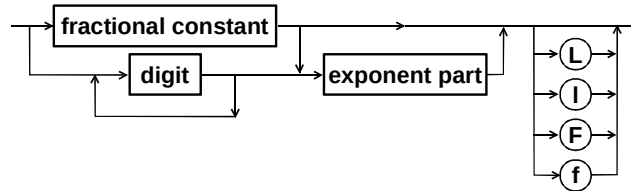
```
#include <stdio.h>          /* Bibliothek fuer die Ausgabe */
int main (void)
{
    ➡ unsigned short a = 32767, b = 1, x ;
    x = a + b ;
    printf ( "a + b = %hu \n", x ) ;
    return 0 ;
}
```

Ausgabe: **a + b = 32768**

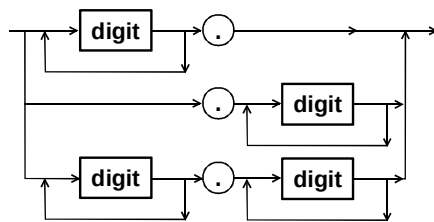


Gleitkomma-Konstanten (1)

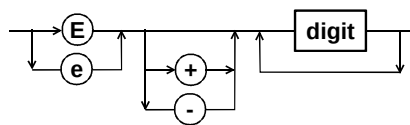
1. **floating constant** (Gleitkomma-Konstante):



2. **fractional constant** (Mantisse):



3. **exponent part** (Exponent):



Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 45

19.11.2004

Gleitkomma-Konstanten (2)

- Für Gleitkomma-Konstanten gelten die folgenden Vereinbarungen:

➔ **Gleitkommakonstanten** werden als **double** (64 Bit) gespeichert, wenn **nichts anderes angegeben ist**.
Nachgestelltes **f** oder **F** führen zum Datentyp **float**.
Das Suffix **l** oder **L** zum Format **long double**.
Die Vorzeichen **+** und **-** sind zulässig, werden aber formal erst auf einer höheren Ebene definiert.

Beispiele für Gleitkommakonstanten:

1.0, **324.**, **.987**, **1.234567L**, **352.E12l**, **1.234567L**, **.234567e-22f**

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd

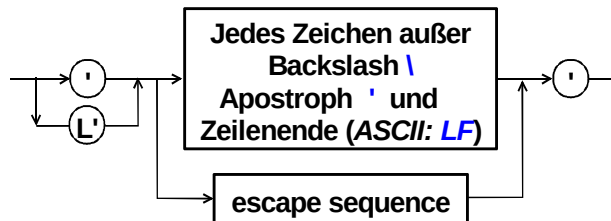


Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 46

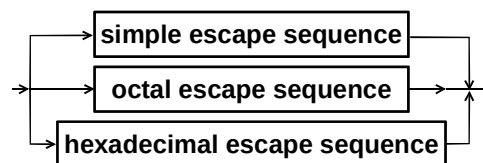
19.11.2004

Zeichenkonstanten (1)

1. **character constant** (Zeichenkonstante):



2. **Escape sequence** (Escape-Sequenz):



Zeichenkonstanten (2)

- Für Zeichenkonstanten gelten die folgenden Vereinbarungen:

➔ Eine einfache **Escape-Sequenz** wird für die folgenden 11 Zeichen verwendet:

\a	Alert (Piepton)
\b	Backspace (←-Taste)
\f	Form feed (Zeilenvorschub)
\n	New line (neue Zeile)
\r	Return (Wagenrücklauf)
\t	Horizontal Tab (horizontaler Tabulator)
\v	Vertical Tab (vertikaler Tabulator)
\\	Backslash (Schrägstrich rückwärts)
\'	Apostrophe (Zeichenkonstanten-Begrenzer)
\"	Quotation mark (Zeichenketten-Begrenzer)
\?	Question mark („Trigrafen“-Anfangszeichen)

Zeichenkonstanten (3)

- Für Zeichenkonstanten gelten die folgenden weiteren Vereinbarungen:

➔ Mit einer **oktalen** oder **hexadezimalen Escape-Sequenz** `\xXX` bzw. `\000` kann der *ASCII*-Code eines Zeichens mit genau 4 Zeichen eingegeben werden.

Beispiele:

Hexadezimale Escape-Sequenzen:

`\x0d` (*ASCII: Return*)

`\x1B` (*ASCII: Escape*)

Oktale Escape-Sequenz:

`\0` (*ASCII: NUL*)

➔ Das **Präfix L** leitet Zeichen eines erweiterten Zeichensatzes ein, der z.B. auch japanische Zeichen enthalten kann.

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 49

ASCII-Zeichen im Konsolmodus von MSVC 6.0 (1)

☛ Microsoft verwendet für *Windows NT/2000/XP* einen **anderen Zeichensatz** als für *DOS*.

- Dadurch **unterscheiden sich** u.a. die **Zahldarstellungen der deutschen Sonderzeichen** *ä, ö, ü, ß*.
- Deshalb sollte man im **Konsolmodus** von **MSVC 6.0** z.B. **nicht 'ü'** (*ASCII-Zeichendarstellung*) sondern statt dessen die Dezimal- oder **Hexadezimaldarstellung** (**0x81**) verwenden!

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 50

ASCII-Zeichen im Konsolmodus von MSVC 6.0 (2)

- Folgendes Beispiel erläutert den Sachverhalt:

```

/* HAW VGL, 27.11.2003 */
/* dateiname: TEST_ue.c */
#include <stdio.h>
int main ( void )
{
    unsigned char c0='a', c1='A', c2='ä', c3='ö', c4='ü', c5='ß' ;
    unsigned char d0='a', d1='A', d2=0x84, d3=0x94, d4=0x81, d5=0xe1 ;
    printf (" \n\n c0 ... c5: \n ") ;
    printf (" a: %c  A: %c  ae: %c  oe: %c  ue: %c  sz: %c\n",
            c0, c1, c2, c3, c4, c5) ;
    printf (" \n\n d0 ... d5: \n ") ;
    printf (" a: %c  A: %c  ae: %c  oe: %c  ue: %c  sz: %c\n\n",
            d0, d1, d2, d3, d4, d5) ;
    return 0 ;
}

```

19.11.2004



ASCII-Zeichen im Konsolmodus von MSVC 6.0 (3)

- Das Programmbeispiel erzeugt im **Konsolmodus** den folgenden Ausdruck:

```

c0 ... c5:
 a: a  A: A  ae: ä  oe: ö  ue: ü  sz: ß

d0 ... d5:
 a: a  A: A  ae: ä  oe: ö  ue: ü  sz: ß

Press any key to continue

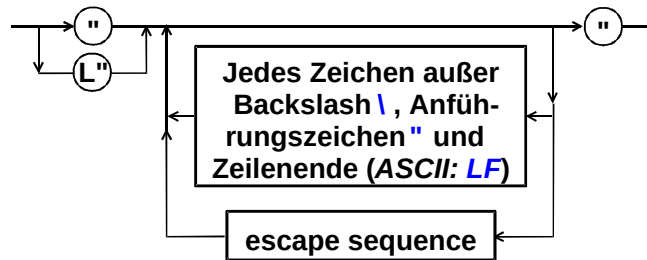
```

19.11.2004



Stringkonstanten (1)

1. **string constant** (konstante Zeichenkette):



19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 53

Stringkonstanten (2)

- Für **Stringkonstanten** (konstante Zeichenketten) gelten die folgenden Vereinbarungen:
 - ➔ **Stringkonstanten** werden im Speicher fortlaufend abgelegt. Jedes ASCII-Zeichen benötigt 1 Byte. Das ASCII-Zeichen **NUL** = `\0` wird hinzugefügt und dient als Abschlusszeichen.
 - ➔ Alle bei **Zeichenkonstanten** beschriebenen **Escape-Sequenzen** sind auch bei Stringkonstanten erlaubt.
 - ➔ Das **Präfix L** leitet Zeichen eines erweiterten Zeichensatzes ein, der z.B. auch japanische Zeichen enthalten kann.

Beispiele für Stringkonstanten:

`"Ich bin eine Stringkonstante."` /* benötigt 30 Bytes incl. der `\0` */

`"Ein Backslash \ und Anführungszeichen \" im String."`

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 54

Aufzählungskonstanten (1)

- **Aufzählungskonstanten (*enumeration constants*)** erleichtern die Programmierung.

- Für sie gelten die folgenden Vereinbarungen:

➔ **Aufzählungskonstanten** weisen **Namen ganzzahligen Konstanten** zu.

Beispiel für Aufzählungskonstanten:

```
enum Tage { Montag, Dienstag, Mittwoch, Donnerstag, Freitag,
           Samstag, Sonntag } ;
```

/* Hier hat Montag den Wert 0, Dienstag den Wert 1, usw. ...
Sonntag (Wert 6) */

➔ Die **Werte der Konstanten** müssen **nicht eindeutig** sein.

Beispiel:

```
enum Semester { E1A, IE1 = 3, D7 = E1A + 3, I5 } ;
```

/* Hier hat E1A den Wert 0, IE1 den Wert 3, D7 den Wert 3 ,
I5 hat den Wert 4 */

19.11.2004



Typumwandlungen (1)

- Bei der Auswertung eines Ausdrucks im Prozessor, beispielsweise $x + 0.3$ muss der Compiler **beide Operanden** in den
➔ **gleichen Datentyp umwandeln**.

- **ANSI-C** schreibt zur Typangleichung die folgenden
➔ **2 Schritte** vor:

➔ **1. Schritt: Format** mindestens **int**

Datentyp	Wird umgewandelt in	Verfahren
unsigned char	int	MS-Byte = 0
signed char	int	Vorzeichen erweitern
short	int	Nicht verändern
unsigned short	unsigned int	Nicht verändern
enum	int	Nicht verändern

19.11.2004



Typumwandlungen (2)

- Bei der Auswertung eines Ausdrucks, z.B. $x + 0.3$ müssen **beide Operanden** den **gleichen Typ aufweisen**.
 - Nach dem 1. Schritt haben beide Operanden **mindestens das Integer-Format**.
- Im **2. Schritt** schreibt ANSI-C zur **Typangleichung** das folgende Vorgehen vor:

➔ 2. Schritt: Umwandlung in den Typ der größeren Genauigkeit

In einem Ausdruck wird jeweils der Datentyp mit der niedrigeren Priorität in den beteiligten Datentyp der höheren Priorität umgewandelt:

Datentyp	Priorität
long double	maximal
double	
float	
unsigned long	
long	
unsigned int	
int	minimal

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 57

Typumwandlungen (3)

- Beispiel für eine Typumwandlung in C:

```
...
signed char a = -1;
float x = 0.5, y;
y = x + a;    /* Hier ist eine 2-stufige Typumwandlung von a nötig! */
```

• 1. Schritt: Umwandlung char ➔ int

`signed char a = 0xff (8 Bit)` ➔ `signed int : 0xffff (16 Bit integer)`

• 2. Schritt: Umwandlung int ➔ float

`signed int = 0xffff (16 Bit)` ➔ `float = -1.0 (32 Bit Gleitkommzahl)`

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 58

Erzwungene Typumwandlung (*casting*) (1)

- Den Datentyp eines Ausdrucks kann man mit dem
➡ **cast Operator** (Datentyp) ... erzwingen
- Casting kann beispielsweise angewendet werden, wenn
➡ die Auswertung eines Ausdrucks zu einem
Zahlbereichsüberlauf führen würde.
- ➡ dem C-Compiler das **Zugriffsformat auf Register**
eines Peripheriebausteins nicht bekannt ist.

Beispiel für **Casting**:

```
...
char a = 120, b = 10, y;
y = (short) a + b - 100;    /* Mit casting: kein Datenüberlauf ! */
```

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 59

19.11.2004



Casting

Verwenden Sie Casting, um bei dem folgenden Programm zur Berechnung eines ganzzahligen Mittelwertes zu vereinfachen und einen Zahlbereichsüberlauf sicher zu verhindern!

```
#include <stdio.h>
int main ( void )
{
    short a = 19251, b = 13618, x, sum ;
    sum = ( a + b ) ;
    x = sum / 2 ;
    printf ( " a = %hd   b = %hd\n ganzzahliger Mittelwert von a "
            "und b: %hd\n\n", a, b, x ) ;
    return 0 ;
}
```

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 60

19.11.2004

✓ Casting

- **Fehler:** Die Auswertung von $a + b$ kann zu einem Zahlbereichsüberlauf führen, der **nicht in jedem Fall** auftritt.

sicherer: ➡ **Casting**

```
#include <stdio.h>           /* Bibliothek für die Ausgabe */
int main (void)
{
    short a = 19251, b = 13618, x ; /* a, b und x definieren */
    x = ((long) a + b) / 2 ; /* Mittelwert von a und b */
    printf ( " a = %hd    b = %hd\n ganzzahliger Mittelwert"
            " von a und b: %hd\n\n", a, b, x ) ;
    return 0 ;
}
```



Elemente von C: logische Variablen

- C kann **jeden Ausdruck** als logische Variable interpretieren.
 - Dabei bedeutet ein **Eingangswert**
 - ➡ = 0: falsch (*false*)
 - ≠ 0: wahr (*true*)
- Das **Ergebnis** einer **logischen Operation** ist entweder
 - ➡ = 0: falsch (*false*) oder
 - = 1: wahr (*true*)



Elemente von C: logische Operationen (1)

- Die *Boole'sche Algebra* definiert **logische Operationen** :

➔ **logische Negation** (Inversion) : **!a** (**NICHT a**)

➔ **logische ODER-Verknüpfung** : **a || b** (**a ODER b**)

➔ **logische UND-Verknüpfung** : **a && b** (**a UND b**)

Das **Ergebnis der logischen Operationen** ist entweder
= **0 falsch (false)** oder = **1 wahr (true)**

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 63

Logische Operationen (2)

- C benutzt für folgende logische Operationen **die Symbole**:

➔ **Negation** (Inversion) **!a** (**NICHT a**)

➔ **ODER-Verknüpfung** **a || b** (**a ODER b**)

➔ **UND-Verknüpfung** **a && b** (**a UND b**)

Logische Variable	Logische Variable	Negation NICHT a	Negation NICHT b	Disjunktion a ODER b	Konjunktion a UND b
a	b	!a	!b	a b	a && b
0 (false)	0 (false)	1 (true)	1 (true)	0 (false)	0 (false)
0 (false)	1 (true)	1 (true)	0 (false)	1 (true)	0 (false)
1 (true)	0 (false)	0 (false)	1 (true)	1 (true)	0 (false)
1 (true)	1 (true)	0 (false)	0 (false)	1 (true)	1 (true)

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 64

Logische Operatoren

- C verwendet folgende **logische Operatoren**.
- Das Ergebnis einer Vergleichsoperation ist entweder
 = 0 falsch (*false*) oder
 ➡ = 1 wahr (*true*)

➡ logisches NOT: !

Beispiel:

if !(a < b) ... /* NICHT(a < b)? ➡ 0 (falsch) oder 1 (wahr) */

➡ logisches UND: &&

Beispiel:

if ((a == b) && (c != d)) ... /* (a == b) UND (c != d)?
 ➡ 0 (falsch) oder 1 (wahr) */

➡ logisches ODER: ||

Beispiel:

if ((x > y) || (x <= v)) ... /* (x > y) ODER (x <= v)?
 ➡ 0 (falsch) oder 1 (wahr) */



Elemente von C: Vergleichsoperatoren (1)

- C verwendet folgende **logische Operatoren**.
- Das Ergebnis einer Vergleichsoperation ist entweder
 ➡ = 0 falsch (*false*) oder
 = 1 wahr (*true*)

➡ gleich: ==

Beispiel:

if (a == b) ... /* a == b? ➡ 0 (falsch) oder 1 (wahr) */

➡ ungleich: !=

Beispiel:

if (a != b) ... /* a != b? ➡ 0 (falsch) oder 1 (wahr) */

➡ größer: >

Beispiel:

if (a > b) ... /* a > b? ➡ 0 (falsch) oder 1 (wahr) */



Vergleichsoperatoren (2)

- C verwendet folgende **logische Operatoren**.
- Das Ergebnis einer Vergleichsoperation ist entweder

➡ = 0 falsch (*false*) oder
 = 1 wahr (*true*)

➡ **kleiner:** <

Beispiel:

if (a < b) ... /* a < b? ➡ 0 (falsch) oder 1 (wahr) */

➡ **größer gleich:** >=

Beispiel:

if (a >= b) ... /* a größer gleich b? ➡ 0 (falsch) oder 1 (wahr) */

➡ **kleiner gleich:** <=

Beispiel:

if (a <= b) ... /* a kleiner gleich b? ➡ 0 (falsch) oder 1 (wahr) */



Bitoperationen (1)

- C verwendet folgende **Operatoren für Bitmanipulationen**.
- Sie dürfen nur auf **ganzzahlige Variablen** oder **ganzzahlige Konstanten** angewendet werden.

➡ **bitweise Negation NOT (1-Komplement):** ~

Beispiel:

a = ~b; /* b = 0110 1110 ➡ a = 1001 0001 */

➡ **linksschieben:** << (Nullen werden nachgezogen)

Beispiel:

L = X << 2; /* X = 0101 1001 ➡ L = 0110 0100 */



Bitoperationen (2)

- C verwendet folgende **Operatoren für Bitmanipulationen**.
 - Sie dürfen nur auf **ganzzahlige Variablen** oder **ganzzahlige Konstanten** angewendet werden.

➔ **Rechtsschieben *unsigned* : >>**
(0en werden nachgezogen)

Beispiel:

```
unsigned char ru, u = 0xCB ;
ru = u >> 3 ;    /* u = 1100 1011  ➔ ru = 0001 1001 */
```

➔ **Rechtsschieben *signed* : >>**
(MSB wird kopiert!)

Beispiel:

```
signed char rs, s = 0xCB ;
rs = s >> 3 ;    /* s = 1100 1011  ➔ rs = 1111 1001 */
```

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 69

Bitoperationen (3)

- C verwendet folgende **Operatoren für Bitmanipulationen**.
 - Sie dürfen nur auf **ganzzahlige Variablen** oder **ganzzahlige Konstanten** angewendet werden.

➔ **bitweise ODER : |**

Beispiel:

```
O = x | y; /* x = 1100 1011  y = 0001 1001  ➔ O = 1101 1011 */
```

➔ **bitweise UND : &**

Beispiel:

```
U = c & D; /* c = 1100 1011  D = 0001 1001  ➔ U = 0000 1001 */
```

➔ **bitweise ANTIVALENZ : ^**

Beispiel:

```
a = E ^ H; /* E = 1100 1011  H = 0001 1001  ➔ a = 1101 0010 */
```

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 70

Elemente von C: 2-Zeichen-Operatoren

- C verwendet **Operatoren**, die aus **2 oder 3 Zeichen** bestehen.
- Sie machen Programme übersichtlicher, aber auch für den Anfänger unverständlicher.
 - Die **Mehrzeichen-Operatoren** bestehen aus **ASCII-Zeichen**, die **ohne Leerzeichen unmittelbar aufeinander folgen müssen**.

Beispiele:

```

==    &&    ||    >>    <<    ++    --    +=
-=    |=    &=    *=    /=    %=    ^=
<<=   >>=
  
```

19.11.2004



Elemente von C: erweiterte Zuweisungen (1)

- C verwendet folgende erweiterte **arithmetische Zuweisungsoperationen**.
- Sie machen Programme übersichtlicher aber für den Anfänger unverständlicher.

➡ Zuweisung : **+=**

Beispiel:

```
a += b;    /* entspricht: a = a + b; */
```

➡ Zuweisung : **-=**

Beispiel:

```
a -= b;    /* entspricht: a = a - b; */
```

➡ Zuweisung : ***=**

Beispiel:

```
a *= b;    /* entspricht: a = a * b; */
```

19.11.2004



Erweiterte Zuweisungen (2)

- C verwendet folgende erweiterte **Zuweisungsoperationen**.
 - Sie machen Programme übersichtlicher aber für den Anfänger unverständlicher.

➡ Zuweisung : **/=**

Beispiel:

`a /= b;` /* entspricht: `a = a / b;` */

➡ Zuweisung : **<<=**

Beispiel:

`c <<= d;` /* entspricht: `c = c << d;` */

➡ Zuweisung : **>>=**

Beispiel:

`v >>= w;` /* entspricht: `v = v >> w;` */

19.11.2004



Erweiterte Zuweisungen (3)

- C verwendet außerdem noch die folgenden **erweiterten Zuweisungsoperationen**

➡ Zuweisung : **|=**

Beispiel:

`z |= T;` /* entspricht: `z = z | T;` */

➡ Zuweisung : **&=**

Beispiel:

`c1 &= d4;` /* entspricht: `c1 = c1 & d4;` */

➡ Zuweisung : **^=**

Beispiel:

`v ^= w;` /* entspricht: `v = v ^ w;` */

19.11.2004



Erweiterte Zuweisungen (4)

- C verwendet außerdem noch die folgenden **erweiterten Zuweisungsoperationen**

➔ Zuweisung : %=

Beispiel:

`R %= d1;` /* entspricht: `R = R % d1;` */

➔ Zuweisung : ? :

Beispiel:

`z = (T1 <= t2) ? T1 : t2;` /* entspricht:
 if (T1 <= t2) z = T1;
 else z = t2; */

19.11.2004



Erweiterte Zuweisungen (5)

- **erweiterte Zuweisungsoperationen in C**

➔ Zuweisung : ++

Beispiele:

`++i` /* `i = i + 1`: In MSVC wird **i** zunächst um 1 erhöht.
 Erst danach werden alle anderen
 Operationen des Ausdrucks ausgeführt! */

`i++` /* `i = i + 1`: In MSVC wird **i** sofort verwendet
 und erst nachdem alle anderen
 Operationen des Ausdrucks ausgeführt
 sind um 1 erhöht! */

➔ Zuweisung : --

Beispiele:

`--j` /* `j = j - 1`: In MSVC wird **j** zunächst um 1
 erniedrigt. Erst danach werden alle anderen
 Operationen des Ausdrucks ausgeführt! */

`j--` /* `j = j - 1`: In MSVC wird **j** sofort verwendet und
 erst nachdem alle anderen
 Operationen des Ausdrucks ausgeführt
 sind um 1 erniedrigt! */

19.11.2004



Erweiterte Zuweisungen (6)

- Bei den zuvor dargestellten **erweiterten Zuweisungsoperationen** ist folgendes zu beachten:

- Die Zuweisungen haben die Form
LValue = RValue;

1. gleicher Datentyp, unterschiedliche Wortlänge

1.1 Länge (LValue) >= Länge (RValue):

RValue wird vorzeichenrichtig erweitert,
Der Zahlenwert von **RValue** wird nicht verändert.

1.2 Länge (LValue) < Länge (RValue):

Der **MS-Teil (RValue)** wird abgeschnitten,
Der Zahlenwert von **RValue** wird verändert 💣*

19.11.2004



Erweiterte Zuweisungen (7)

- Bei den zuvor dargestellten **erweiterten Zuweisungsoperationen** ist folgendes zu beachten:

- Die Zuweisungen haben die Form
LValue = RValue;

2. unterschiedlicher Datentyp

2.1 **LValue**: Gleitkommazahl **RValue**: ganze Zahl

Als Nachkommastellen werden Nullen angenommen.
Im Rahmen der verfügbaren Stellen wird der Wert korrekt zugewiesen.

2.2 **LValue**: ganze Zahl **RValue**: Gleitkommazahl

Die **Nachkommastellen** werden abgeschnitten und
die Vorkommastellen werden **nur bei ausreichender Stellenanzahl** korrekt übernommen

Der Zahlenwert von **RValue** wird ggf. verändert 💣*

19.11.2004



Vorrang & Bearbeitungsrichtung der Operatoren (1)

- Bei den zuvor dargestellten Operatoren sind die

➡ **Priorität** und ➡ **Bearbeitungsrichtung** zu beachten:

Prioritätsgruppe	Operator	Bearbeitungsrichtung
1	() [] -> .	→
2	! ~ ++ -- + - (typ) * & sizeof	←
3	* / % (binär)	→
4	+ - (binär)	→
5	<< >>	→
6	< <= >= >	→
7	== !=	→
8	&	→
9	^	→
10		→
11	&&	→
12		→
13	? :	←
14	= += -= /= usw.	←
15	,	→

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 79

19.11.2004

Vorrang & Bearbeitungsrichtung der Operatoren (2)

↓
Prioritätsgruppe

↓
Bearbeitungsrichtung

Prioritätsgruppe	Operator	Bearbeitungsrichtung
1	() [] -> .	→
2	! ~ ++ -- + - (typ) * & sizeof	←
3	* / % (binär)	→
4	+ - (binär)	→

- Die Operatoren sind nach ➡ **Prioritätsgruppen** geordnet.
Die Priorität von Operatoren der **Gruppe n** ist jeweils **untereinander gleich** und **höher als die der Gruppe n+1, n=1 ... 15**
- Bei gleicher Prioritätsgruppe wird jeweils die angegebenen
➡ **Bearbeitungsrichtung** verwendet.
➡ **Unär** haben die Operatoren **+, - und ***
höhere Priorität als ➡ **binär**.

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 80

19.11.2004

Unäre und binäre Operatoren

- C unterscheidet:

➔ unäre Operatoren:

Ein unärer Operator bezieht sich **nur** auf die **folgende** Variable oder Konstante.

Beispiel: unärer - Operator: `char a = -7 ;`

➔ binäre Operatoren

Ein binärer Operator verknüpft die **vorhergehende** Variable oder Konstante mit der **folgenden Variablen** oder Konstanten.

Beispiel: binärer - Operator: `c = a - b ;`

Beispiel zu Priorität und Bearbeitungsrichtung:

`x = !~a % b++ / --c ; /* entspricht: x = (!(~(-a))) % ((b++) / (--c)); */`



Elemente in C: wichtige mathematische Funktionen

- Die wichtigsten Funktionen der **Bibliothek** ➔ `<math.h>`

Funktion	Erläuterung
<code>double sin (double x)</code>	Sinus, x im Bogenmaß
<code>double cos (double x)</code>	Kosinus, x im Bogenmaß
<code>double tan (double x)</code>	Tangens, x im Bogenmaß
<code>double asin (double x)</code>	Arcus Sinus, $-1 \leq x \leq +1$, $-\pi/2 \leq \text{asin} \leq +\pi/2$
<code>double acos (double x)</code>	Arcus Kosinus, $-1 \leq x \leq +1$, $0 \leq \text{acos} \leq +\pi$
<code>double atan (double x)</code>	Arcus Tangens, $-\pi/2 \leq \text{atan} \leq +\pi/2$
<code>double atan2 (double y, double x)</code>	Arcus Tangens, $\text{atan}(y/x)$ $-\pi \leq \text{atan2} \leq +\pi$
<code>double sinh (double x)</code>	Sinus Hyperbolicus
<code>double cosh (double x)</code>	Kosinus Hyperbolicus
<code>double tanh (double x)</code>	Tangens Hyperbolicus
<code>double exp (double x)</code>	e^x
<code>double log (double x)</code>	natürlicher Logarithmus $\ln(x)$, $x > 0$
<code>double log10 (double x)</code>	dekadischer Logarithmus $\log(x)$, $x > 0$
<code>double pow (double x, double y)</code>	x^y
<code>double sqrt (double x)</code>	$\sqrt{x}$
<code>double ceil (double x)</code>	aufrunden
<code>double floor (double x)</code>	abrunden
<code>double fabs (double x)</code>	$ x $



Eigenschaften der mathematische Funktionen

- Die Argumente und Ergebnisse der Funktionen von `<math.h>` sind als **double** deklariert.
- Um Fehler zu vermeiden, muss in sehr seltenen Fällen der **cast-Operator** verwendet werden.

Beispiel:

```
#include <stdio.h>
#include <math.h>          /* für die Anwendung der mathematischen
                           /* Funktionen erforderlich */

/* ... = weitere Befehle */
int main(void)
{ /* ... */
    double x , y = 69.96 ;
    float f, g = 96.96 ;
    short i = 2 ;
    x = sin (y) ;           /* kein casting erforderlich */
    f = pow ( (double) g, (double) i ) ; /* casting möglich, ... */
    /* ... */ }            /* ... kein casting nötig! */
```

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 83

19.11.2004



Konstanten in C

Welche der folgenden Konstanten sind gültig?

Wenn eine Konstante gültig ist, geben Sie den genauen Typ der Konstante an.

Falls eine Konstante ungültig ist, geben Sie an, warum.

0.5	double	0515	int (oktal)
27,822	falsch: ,	0x54.32	falsch: hex & .
9.3e12	double	'\'	char: \
9.3e-12f	float	'/n'	falsch: '/n'
123456L	long	"Pommes mit Majo"	char (string)
0.8E 8	falsch: [blank]	"1.3e-12"	char (string)
0XCFAL	long (hex)	11.11.1995	falsch: ein . zuviel
0XYu	falsch: Y		

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 84

19.11.2004

**Ausdrücke in C (1)**

Berechnen Sie die Werte der folgenden Ausdrücke.
Verwenden Sie dabei jeweils die folgenden
vorgegebenen Werte der Variablen:

int i = 8, j = 5; float x = 0.005, y = -0.01; char c = 'c', d = 'd';

(3 * i - 2 * j) % (2 * d - c)	14 % 101	⇒ 14
2 * ((i / 5) + (4 * (j - 3)) % (i + j - 2))	2 * (1 + (8) % (11))	⇒ 18
-(i + j)		⇒ -13
++i		⇒ 9 (vorher erhöht!)
i--		⇒ 7 (nachher erniedrigt!)
++x		⇒ 1.005
i <= j		⇒ 0 (false)
x >= 0.f		⇒ 1 (true)
(2 * x + y) == 0		⇒ 1 (true)
(i > 0) && (j < 5)		⇒ 0 (false)
(x > y) && (i > 0) && (j <= 5)		⇒ 1 (true)
!(c == '9')		⇒ 0 (false)
5 * (i + j) > 'c'		⇒ 0 (false)

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 85

**Ausdrücke in C (2)**

Berechnen Sie die Werte der folgenden Ausdrücke.
Verwenden Sie dabei jeweils die folgenden
vorgegebenen Werte der Variablen:

int i = 8, j = 5, k; float x = .005, y = -0.01, z; char c = 'c', d = 'd';

k = (i + j)	k = 13	
z = (x + y)	z = -0.005	
i = j	i = 5	
k = (x + y)	k = 0	
i += 2	i = 10	
x *= 2	x = .01	
i /= j	i = 1	
i += (j - 2)	i = 11	⚡ (Reihenfolge!)
k = (j == 5) ? i : j	k = 8	
z = (y >= 0) ? y : 0.	z = 0.	
i -= (j > 0) ? j : 0	i = 3	⚡ (Reihenfolge!)

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.-Ing. U. Vogelsang, Prof. Dr. U. Sauvagerd



Programmieren in ANSI C:
K4_C-Elemente.ppt, Seite 86