Hochschule **Rosenheim**
University of Applied Sciences

Prozedurale Programmierung

Ein-/Ausgabe
Ausdrücke

Technische Hochschule Rosenheim

WS 2018/19

Prof. Dr. F.J. Schmitt

Überblick

- Welche wichtigen Funktionen enthält die Standard-Bibliothek zur Ein- und Ausgabe?
- Welche Aspekte müssen bei der Ein- und Ausgabe von unterschiedlichen Datentypen berücksichtigt werden?
- Wie werden in C Ausdrücke behandelt?

PRINTF()/SCANF()

Ausgabefunktionen (1)

- Standardbibliothek unterstützt nur Textausgaben, keine Grafik

- Wichtigster Befehl: `printf`
  - Erwartet mindestens ein Argument, eine Zeichenkette
  - Unterstützt verschiedene Escape-Sequenzen (= Steuerzeichen):
    - \n Neue Zeile (newline)
    - \t Tabulator
    - \\ Backslash
    - \“ Anführungszeichen
    - \a Klingelton (beep)

Ausgabefunktionen (2)

- Wichtigster Befehl: `printf`
  - Ausgabe Zeichenkette (`printf("Ausgabetext");`)
  - Ausgabe von Variableninhalten (oder Werten) über Platzhalter
    - Eingeleitet mit %
    - Stößt `printf` bei der Ausgabe auf einen Platzhalter, so wird der nächstfolgende Parameter anstelle des Platzhalters ausgegeben

Ausgabe von ganzen Zahlen

- Platzhalter für ganze Zahl: `%d` oder `%i`
- Platzhalter für Daten vom Typ `long`: `%ld`

```
printf ("Eine ganze Zahl: %ld\n", wert);
```

```
printf (" oktal:     %lo\n", wert);
printf (" dezimal:      %ld\n", wert);
printf (" hexadezimal: %lx\n", wert);
```

```
long   zahl1 = 1;
long   zahl2 = 3;
long   summe;
summe = zahl1 + zahl2;
printf ("Summe von %ld und %ld  ist %ld\n ",
         zahl1, zahl2, summe);
```

Ausgabe einer Gleitpunktzahl

- Platzhalter für eine Gleitpunktzahl: `%f`, `%e` oder `%g`

```
double wert = 123.456;
printf ("Punktnotation:                    %f\n", wert);
printf ("Wissenschaftliche Notation:       %e\n", wert);
printf ("Variable Notation:                %g\n", wert);
```

```
Ausgabe lautet:
Punktnotation:                  123.456000
Wissenschaftliche Notation:     1.234560e+02
Variable Notation:              123.456
```

- Zusätzliche Angabe für Ausgabegenauigkeit möglich

```
printf ("Punktnotation:    %.2f\n", wert); //123.46
printf ("Punktnotation:    %10.2f\n", wert);
//10 ist Ausgabebreite - rechtsbündige Darstellung  123.46
```

Weitere Ausgaben

- Ausgabe von Adressen (Zeiger, *engl. pointer)*

```
printf (" Die Adresse  %p wird ausgegeben.\n", ptr);
```

- Ausgabe eines Zeichens

```
printf (" Das Zeichen  %c wird ausgegeben.\n", z);
```

- Ausgabe einer Zeichenkette

```
printf (" Der Name lautet  %s. \n", name);
```

Fazit Ausgabe

- Bei falscher Anwendung können schwere Programmfehler auftreten
- Überprüfen Sie stets, ob für die jeweilige Variable der richtige Platzhalter ausgewählt wurde!

```
long wert;
// ...
printf (" Der Wert ist: %g\n", wert);
```

Eingabefunktionen (1)

- Eingabefunktionen der Standardbibliothek sind ähnlich den Ausgabefunktionen zu verwenden

- Wichtiger Befehl: `scanf`
  - Liest Werte von der Tastatur und schreibt diese in Variablen
  - auch Verwendung von Platzhaltern
    (Typ des Werts, der gelesen werden soll)
  - Rückgabe:
    - Anzahl der erfolgreich gelesenen Werte
      - kann 0 sein, wenn Eingabe nicht zum Format passt
      - Bsp: %d erwartet ganze Zahl, es wird aber ein Buchstabe eingegeben
    - die Konstante `EOF` (end of file), falls Fehler aufgetreten oder Eingabe beendet wurde

Eingabefunktionen (2)

- Eingabe einer ganzen Zahl (`%d` oder `%i` oder `%ld`)

```
scanf ("%ld", &longwert);
```

Adressoperator vor Variablennamen!

- Eingabe einer Gleitpunktzahl (`%f` oder `%lf`)

```
scanf ("%f",   &floatWert);
scanf ("%lf",  &doubleWert);
```

- Eingabe eines Zeichens (`%c`)

- Eingabe einer Zeichenkette (`%s`)

Anwendungsaspekte (1)

- Benutzer müssen zur Eingabe aufgefordert werden
  - Separate `printf` Anweisung ist notwendig

```
printf ("Geben Sie eine ganze Zahl ein:");
scanf  ("%ld", &zahl);
```

- Lesen einer Variablen erfolgt in drei Schritten:
  (1) Es wird solange Text(!) von der Tastatur gelesen bis Benutzer die Eingabetaste drückt
  (2) Konvertierung der Eingabe in den gewünschten Datentyp
  (3) Gelesene und konvertierte Daten werden in den Speicher geschrieben

Anwendungsaspekte (2)

```
long zahl;
//...
printf ("Geben Sie eine ganze Zahl ein:");
scanf  ("%lf",  &zahl);
```

- Ziffern werden von der Tastatur gelesen und auf Grund des Platzhalters in `double` konvertiert
- Beim Abspeichern in die Variable `zahl` kann es zu Programmabsturz kommen (Hinausschreiben über Speicherbereich - 64 Bit, aber nur 32 Bit reserviert)

➢ Achte auf Übereinstimmung von Platzhalter und Datentyp der Variablen sowie auf Adressoperator!

Aufgabe

- Schreiben Sie ein C-Programm, das
  - zwei Gleitpunktzahlen einliest,
  - die Inhalte der eingelesenen Variablen vertauscht
  - und diese dann ausgibt.

AUSDRÜCKE

Ausdrücke

- Zuweisungen und Bedingungen werden in C in sog. Ausdrücken (*engl. Expressions)* geschrieben
- Wichtig für C-Programmierer:
  - Wie werden Ausdrücke behandelt und ausgewertet?

- es existiert eine Vielzahl von Operatoren, die in Ausdrücken verwendet werden können
- Beachte:
  - sinnvoller Einsatz muss angestrebt werden
  - schwer durchschaubare Konstrukte sind zu vermeiden
  - komplexe Ausdrücke sollten angemessen dokumentiert werden

Auswertung von Ausdrücken

- C geht bei der Bearbeitung eines Ausdrucks nach einer genau definierten Reihenfolge vor
  - Operator mit der höchsten Priorität wird ausgeführt (siehe Tabelle)
  - dann Ausführung des Operators mit der nächst niedrigeren Priorität
  - haben zwei Operatoren dieselbe Priorität, so entscheidet die Assoziativität

- Zusammenfassung von Ausdrücken: Runde Klammern (…)
  - Priorität eines Operators unklar ⇒ Klammern einfügen
  - Beeinflussung der Abarbeitungsreihenfolge

Assoziativität

- Beschreibung der Bindung zwischen gleichwertigen Operatoren
- alle binären Operatoren (außer Zuweisung) sind links bindend
  - haben genau zwei Operanden, bspw. + oder *

```
1 - 2 + 3    ⇒    (1 - 2) + 3
```

- alle unären Operatoren sind rechts bindend
  - haben genau einen Operanden, bspw. ! oder ~

Operatoren

- Alle Operatoren haben ***immer*** einen Rückgabewert
- Auswahl wichtiger Operatoren:
  - Arithmetische Operatoren
  - Zuweisung
  - Logische Operatoren

Arithmetische Operatoren

- 3 + 2 * (8 – 4)
- Auswertungsdiagramm:

```
3  +  2  *  ( 8  –  4 )
3  +  2  *       4
3  +        8
     11
```

Zuweisung (1)

- Auf der linken Seite steht ein Lvalue (Links-Wert)
  - ist besondere Art eines Ausdrucks
  - steht für eine Speicherzelle
- Ergebnis des Ausdrucks auf der rechten Seite wird zugewiesen

```
a = b = c = d = 0;
```

- Auswertungsdiagramm:

```
a = b = c = d = 0;
a = b = c = 0;
a = b = 0;
a = 0;
```

Zuweisung (2)

- So nicht (auch wenn C das erlaubt)!

```
x = 2 * (y = ( z = 4) + 1);
```

- Besser:

```
z = 4;
y = z + 1;
x = 2 * y;
```

Zuweisung (3)

- Inkrement
  - Erhöhung des Werts einer Variablen um eins

```
wert = wert + 1;    ⇒   wert++;
```

- Dekrement
  - Verkleinern des Werts einer Variablen um eins

```
wert = wert - 1;    ⇒   wert--;
```

- beide Operatoren ++ und -- können auch vor dem Operanden geschrieben werden

```
wert = 10;
++wert;
```

Zuweisung (4)

- Präinkrement

Anweisungen	Wert von a	Wert von b
a = 3;	3	?
b = ++a;	4	4

- Postinkrement

Anweisungen	Wert von a	Wert von b
a = 3;	3	?
b = a++;	4	3

Zuweisung (5)

- Abkürzungen

```
x1 = x1 * 10;      ⇒  x1 *= 10;
x2 = x2 + 10;      ⇒  x2 += 10;
```

- Aber: evtl. Fehlerquelle

```
x = x * 2 + y;     ⇒ x *= 2 + y;
```

```
x  = x * (2 + y);
```

Tipp: Nur in einfachen Ausdrücken verwenden!

Logische Operatoren (1)

- Verwendung: Berechnung von Wahrheitswerten
- Eigener Datentyp für Wahrheitswerte erst ab C99 (wird von MSVC nicht unterstützt)
  ⇒ Benutzung des Datentyps `int`
  - 0 bedeutet „falsch“
  - jede Zahl ungleich 0 bedeutet „wahr“
  - ist das Ergebnis einer Auswertung von Vergleichsoperatoren oder logischen Operatoren „wahr“, so erhält man die 1

Logische Operatoren (2)

- Vergleichsoperatoren in C

Operator	Erklärung
<	Kleiner
<=	Kleiner gleich
>=	Größer gleich
>	Größer
==	Gleich
!=	Ungleich

Logische Operatoren (3)

- Operator &&
  - Logische UND-Verknüpfung

Operand a	Operand b	a && b
0	0	0
0	1	0
1	0	0
1	1	1

Bsp: Liegt Zahl zwischen 10 und 20?

```
(zahl >= 10) && (zahl <= 20)
```

Wird nicht ausgewertet, falls erster Ausdruck bereits 0

Logische Operatoren (4)

- Operator ||
  - Logische ODER-Verknüpfung

Operand a	Operand b	a || b
0	0	0
0	1	1
1	0	1
1	1	1

Bsp: Ist Zahl kleiner 10 oder größer 20?

```
(zahl < 10) || (zahl > 20)
```

Wird nicht ausgewertet, falls erster Ausdruck bereits 1

Logische Operatoren (5)

- Operator !
  - Negieren von Wahrheitswerten
  - Wert „wahr“ wird in „falsch“ gewandelt und Wert „falsch“ wird in „wahr“ gewandelt

```
!(x1 && x2 && … && xn)      // ist äquivalent zu
!x1 || !x2 || … || !xn

!(x1 || x2 || … || xn)      // ist äquivalent zu
!x1 && !x2 && … !xn

!( x1  &&  x2   ||  x3)     // ist äquivalent zu
!((x1  &&  x2)  ||  x3)     // ist äquivalent zu
((!x1  || !x2)  && !x3)
```

Priorität der Operatoren

Priorität	Assoziativität	Operatoren
1 (höchste)	Links-nach-rechts	Funktionsaufruf (), Indexzugriff [], Elementzugriffe -> .
2	Rechts-nach-Links	Vorzeichen + -, logisches/bitweises NOT ! ~, prä-/post- Inkrement/Dekrement ++ --, Adresse &, Zeigerdereferenzierung *, Tyumwandlung (typ), sizeof
3	Links-nach-rechts	Mutliplikation, Division, Modulo * / %
4	Links-nach-rechts	Addition, Subtraktion + -
5	Links-nach-rechts	Links-/Rechtsshift << >>
6	Links-nach-rechts	kleiner/größer (gleich) < <= > >=
7	Links-nach-rechts	Gleich, ungleich == !=
8	Links-nach-rechts	Bitweises AND &
9	Links-nach-rechts	Bitweises XOR ^
10	Links-nach-rechts	Bitweises OR |
11	Links-nach-rechts	Logisches AND &&
12	Links-nach-rechts	Logisches OR ||
13	Rechts-nach-Links	Bedingung ?
14	Rechts-nach-Links	(zusammengesetzte) Zuweisung = *= /= %= += -= &= ^= |= <<= >>=
15 (niedrigste)	Links-nach-rechts	Kommaoperator ,

Häufige Fehler

- Verwechslung von = (Zuweisung) und
  == (Abfrage auf Gleichheit)

- Prüfung auf Gleichheit bei Gleitpunktzahlen
  - Exakte Abfrage auf Gleichheit nicht möglich
  - Bestimmung, ob das Ergebnis hinreichend genau ist
  - Bsp: Bereiche, wobei `eps` die Genauigkeit und `wert` den „genauen“ Wert angibt

```
((wert - eps) < d ) && (d < (wert + eps))
```

Zusammenfassung

- Ein-/Ausgabefunktionen printf(), scanf()
- Auswertung von Ausdrücken
  - arithmetische
  - logische
  - Zuweisungen