Programmierung 1

Logik & Bedingungen

Dr. Monika Schak

Woche 2

29. Oktober 2025

Wiederholung

- **Variablen** dienen dazu, Daten zu verwalten
  - Variablen haben einen **Bezeichner**, eine **Speicheradresse** und einen **Datentyp**
  - Variablenwerte können durch **Zuweisungen** verändert werden
- Aufbau von **Zuweisungen**: `<variable> = <ausdruck>;`
  - Wert des Ausdrucks wird mittels **aktuellem Variablenzustand** ermittelt
  - Variablen auf der linken Seite der Zuweisung wird der ermittelte Wert zugeordnet
  - Die Zuweisung (mit Zuweisungsoperator =) ist selbst ein **Ausdruck**
  - Ein mit einem Semikolon abgeschlossener Ausdruck ist eine **Anweisung**
- Ein **Programm** ist eine geordnete Folge von Anweisungen
  - Es besteht damit aus einer **Anweisungssequenz**
  - Anweisungsblöcke sind durch **geschweifte Klammern** zusammengefasst

Ausdrücke

- Konstanten: Werte, die keine Variablen sind, z.B. 4711
- Zwei Ausdrücke mit einem Operator: `+`, `-`, `*`, `/`
  Vorsicht bei Division (`/`): Sind beide Werte ganzzahlig (`int`), handelt es sich um eine ganzzahlige Division (Nachkommazahl wird nicht berücksichtigt), wird das Ergebnis einer Integer-Variablen zugewiesen, ist es auch eine ganzzahlige Division.
  Modulo-Operation (`%`): Gibt den Rest bei ganzzahliger Division.
- Geklammerter Ausdruck, z.B. $2 * (17 + 4)$

```
7 + 3                    9 / 4
12 - 4                   23 / 7
2 * 8                    13 % 2
2 + 3 * (3 + 4) - 1      10 % 4
13 / 2                   20 % 7
```

printf-Funktion

```
int j = 13, zahl = 9;
int j = j + zahl;
printf("j hat den Wert %d\n", j);
```

- Funktion `printf()` schreibt Argument(e) auf die Konsole. Das erste Argument ist immer eine Zeichenkette, optional sind weitere Argumente durch Komma getrennt möglich. Die weiteren Argumente sind Ausdrücke, deren Wert man ausgeben möchte.
- Erlaubt eine formatierte Ausgabe: Platzhalter `%d` steht für eine ganze Zahl (`int`) als Dezimalzahl. Für jeden weiteren Ausdruck ist je ein Platzhalter nötig.

Eingabe

```
int age;
printf("Bitte Alter eingeben: ");
scanf("%d", &age);
printf("Du wurdest %d geboren.\n", 2022-age);
```

- Die Funktion `scanf()` liest von der Standardeingabe (Tastatur)
  - `scanf()` ist ähnlich aufgebaut wie `printf()`
  - Formatzeichen `%d` spezifiziert, dass eine ganze Zahl eingelesen werden soll
  - Zeichen `&` ist ein Adressoperator
- In C muss beim Einlesen von Werten durch die Speicheradresse angegeben werden (mittels Adressoperator), wo der Wert im Speicher abgelegt werden soll

Probleme bei Eingabefunktionen

- Eingabefunktionen wie `scanf()` haben generell Sicherheitsproblem
  - Leider auf jeder Plattform, ist aber erst für Produktivcode relevant
  - `scanf()` prüft nicht auf Pufferüberlauf, was unbehandelt Hackern Tür und Tor öffnet, ist aber leichter zu nutzen als direkt die komplette Problembehandlung umzusetzen
- Viele IDEs sehen solche Funktionen als veraltet (deprecated) oder unsicher an
  - Sie warnen vor der Nutzung dieser Funktion.
  - Alternativ gibt es seit C11-Standard eine sichere Variante: `scanf_s()`

Boolesche Variablen

- Datentyp: `bool`
  Mit C99-Standard als `_Bool` eingeführt
  Bindet man `stdbool.h` ein, kann man den Typ `bool` verwenden
- Variablen vom Typ `bool` können einen von zwei Werten annehmen:
  - `false` (in C wird 0 als false ausgewertet)
  - `true` (in C wird alles ungleich 0 als true ausgewertet)
- Beispiel: `bool human = true;`

Boolesche Ausdrücke

Boolesche Operatoren berechnen aus einem oder zwei booleschen Ausgangswerten (vom Typ `bool`) einen neuen booleschen Wert

- Logisches UND `&&` (nur *true*, wenn beide Argumente **true**)
- Logisches ODER `||` (nur **false**, wenn beide Argumente *false*)
- Logisches Nicht: `!` (ergibt den jeweils anderen Wahrheitswert)

a	b	a && b
false	*false*	*false*
false	*true*	*false*
true	*false*	*false*
true	*true*	*true*

a	b	a || b
false	*false*	*false*
false	*true*	*true*
true	*false*	*true*
true	*true*	*true*

a	!a
false	*true*
true	*false*

Übung

- Ordnen Sie die Begriffe dem folgenden Code zu: Datentyp, Variable, Wert, Deklaration, Zuweisung

```
bool a = true, b = false;
int x = 18, y = (x/4) % 3, z;
x = x * 2;
a = a && true;
b = (a && b) || false;
//printf("%d\n", x);
printf("%d\n", y);
```

- Was sind die Werte der Variablen a, b, x, y und z nach Ausführung des obigen Codes?
- Was wird am Ende ausgegeben?

Vergleiche

- Das Resultat von Vergleichen (z.B. zwischen Zahlen) ist ein boolescher Wert
  - == (Gleichheit), != (Ungleichheit): Vergleich von Zahlen, Zeichen und booleschen Werten auf Gleichheit bzw. Ungleichheit
  - > (Größer), >= (Größer gleich): Vergleich von Zahlen, welche größer ist
  - < (Kleiner), <= (Kleiner gleich): Vergleich von Zahlen, welche kleiner ist
- In der Bedingung von Fallunterscheidungen (sog. `if`-Anweisungen) steht ein boolescher Ausdruck, der einen Wahrheitswert ergibt
  - Fallunterscheidung dienen der Modellierung alternativer Programmabläufe

Bedingte Anweisungen

- Die Schlüsselwörter für bedingte Anweisungen sind `if` und `else`
- Erst wird der Bedingungsausdruck ausgewertet: `n > 0`
  $\rightarrow$ muss einen Wahrheitswert ergeben
- Falls das Ergebnis `true` ist, weiter mit erstem Block, sonst mit zweitem (else)
- Bei einer einseitigen Fallunterscheidung gibt es nur den ersten Block
- Geschweifte Klammern definieren Blöcke (sollten immer ordentlich eingerückt werden)
- In Blöcken können wieder Anweisungen oder Fallunterscheidungen stehen, es gibt aber auch leere Blöcke `{ }`

```
int n = 3;

if (n < 0) {
  printf("Die Zahl ist negativ.\n");
}
else {
  printf("Die Zahl ist positiv.\n");
}
```

Ternärer Operator

- Beispiel: Bestimmung des Maximus (= größter Wert)

```
if (a > b) {
  x = a;
} else {
  x = b;
}
```

- Geht auch übersichtlicher als Einzeiler:

```
x = (a > b) ? a : b;
```

- Hier wird der größte Wert mit Hilfe des sog. **Ternären Operators** bestimmt (über die Bedingung a > b) und das Ergebnis wird x zugewiesen

Ternärer Operator

- Bedingungsoperator `?:` ist einziger ternärer Operator, d.h. Operator mit 3 Operanden, (z.B. + ist binärer Operator, ! ist unärer Operator)
- Syntax: `<Bedingungsausdruck> ? <Anweisung1> : <Anweisung2>`
  - Falls Bedingung wahr: Anweisung1 wird ausgewertet, sonst Anweisung 2
  - Es wird immer nur genau ein Asudruck ausgewertet
  - Kurzform der if/else-Anweisung (v.a. bei Zuweisungen)
- Gilt für jeden Datentyp, kann zugewiesen werden wie andere Ausdrücke auch, oder alleine stehen
- Beispiel:

```
printf("Zahl %d ist %sgerade\n", zahl, (zahl % 2 == 0) ? "" : "un");
```

Datentyp “char”

- Für einzelne Zeichen (Characters) gibt es den Datentyp `char`
- Dient zur Speicherung einzelner Buchstaben, Ziffern u. Sonderzeichen (inkl. Steuerzeichen = Escape-Sequenzen wie `\n`) aus ASCII-Zeichensatz
  ASCII-Tabelle:
  `https://www.c-howto.de/tutorial/anhang/ascii-tabelle/`
- Die Größe eines `char`s beträgt genau 1 Byte: Wertebereich von -128 bis +127 (bzw. von 0 bis 255 = Entspricht dann Eintrag in der ASCII-Tabelle)
- Angabe eines Zeichenwertes in einfachen Anführungszeichen
- Für Ein- und Ausgaben gibt es das Formatzeichen %c
- Beispiel:

```
char c0 = 'p', c1 = 'r', c2 = 'o', c3 = 'g', nl = '\n';
printf("%c%c%c%c%c", c0, c1, c2, c3, nl);
```

ASCII-Tabelle

Scan-code	ASCII hex	ASCII dez	Zeichen
	00	0	NUL ^@
	01	1	SOH ^A
	02	2	STX ^B
	03	3	ETX ^C
	04	4	EOT ^D
	05	5	ENQ ^E
	06	6	ACK ^F
	07	7	BEL ^G
0E	08	8	BS ^H
0F	09	9	TAB ^I
	0A	10	LF ^J
	0B	11	VT ^K
	0C	12	FF ^L
1C	0D	13	CR ^M
	0E	14	SO ^N
	0F	15	SI ^O
	10	16	DLE ^P
	11	17	DC1 ^Q
	12	18	DC2 ^R
	13	19	DC3 ^S
	14	20	DC4 ^T
	15	21	NAK ^U
	16	22	SYN ^V
	17	23	ETB ^W
	18	24	CAN ^X
	19	25	EM ^Y
	1A	26	SUB ^Z
01	1B	27	Esc ^[
	1C	28	FS ^\
	1D	29	GS ^]
	1E	30	RS ^^
	1F	31	US ^_

Scan-code	ASCII hex	ASCII dez	Zch.
	20	32	SP
02	21	33	!
03	22	34	"
29	23	35	#
05	24	36	$
06	25	37	%
07	26	38	&
0D	27	39	'
09	28	40	(
0A	29	41	)
1B	2A	42	*
1B	2B	43	+
33	2C	44	,
35	2D	45	-
34	2E	46	.
08	2F	47	/
0B	30	48	0
02	31	49	1
03	32	50	2
04	33	51	3
05	34	52	4
06	35	53	5
07	36	54	6
08	37	55	7
09	38	56	8
0A	39	57	9
34	3A	58	:
33	3B	59	;
2B	3C	60	<
0B	3D	61	=
2B	3E	62	>
0C	3F	63	?

Scan-code	ASCII hex	ASCII dez	Zch.
	40	64	@
1E	41	65	A
30	42	66	B
2E	43	67	C
20	44	68	D
12	45	69	E
21	46	70	F
22	47	71	G
23	48	72	H
17	49	73	I
24	4A	74	J
25	4B	75	K
26	4C	76	L
32	4D	77	M
31	4E	78	N
18	4F	79	O
19	50	80	P
10	51	81	Q
13	52	82	R
1F	53	83	S
14	54	84	T
16	55	85	U
2F	56	86	V
11	57	87	W
2D	58	88	X
2C	59	89	Y
15	5A	90	Z
	5B	91	[
	5C	92	\
	5D	93	]
29	5E	94	^
35	5F	95	_

Scan-code	ASCII hex	ASCII dez	Zch.
0D	60	96	`
1E	61	97	a
30	62	98	b
2E	63	99	c
20	64	100	d
12	65	101	e
21	66	102	f
22	67	103	g
23	68	104	h
17	69	105	i
24	6A	106	j
25	6B	107	k
26	6C	108	l
32	6D	109	m
31	6E	110	n
18	6F	111	o
19	70	112	p
10	71	113	q
13	72	114	r
1F	73	115	s
14	74	116	t
16	75	117	u
2F	76	118	v
11	77	119	w
2D	78	120	x
2C	79	121	y
15	7A	122	z
	7B	123	{
	7C	124	|
	7D	125	}
	7E	126	~
53	7F	127	DEL

Komplexere if-Anweisungen

Beispiel: Ein einfacher Taschenrechner

```
char operator;
int wert1, wert2, erg;

printf("Term angeben (z.B. 1 + 1)!\n");
scanf("%d %c %d", &wert1, &operator, &wert2);

// TODO: Implementierung der Berechnung

if (operator == ' ') {
  printf("Fehler, falsche Eingabe\n");
} else {
  printf("%d %c %d = %d\n", wert1, operator, wert2, erg);
}
```

Mehrfachauswahl

```
switch (operator) {
  case '+': erg = wert1 + wert2; break;
  case '-': erg = wert1 - wert2; break;
  case '*': erg = wert1 * wert2; break;
  case '/': erg = wert1 / wert2; break;
  case '%': erg = wert1 % wert2; break;
  default: operator = ' ';
}
```

- Bedingungsauswahl im *Schalter* (`switch`) liefert Wert (vom Typ `int` oder `char`)
- Für jeden möglichen Wert ist ein Fall (`case`) vorgesehen
- Nicht berücksichtigte Fälle werden unter `default` behandelt
- Die Anweisung `break` bedeutet: wird wird abgebrochen und Ausführung nach dem `switch` fortgeführt
- Ohne `break` wird je nachfolgendes `case` weiter abgearbeitet

Zusammenfassung

- Zwei Typen von Fallunterscheidungen
  - ```
    if (Bedingung_erfuellt) { ... }
    else if (andere_Bedingung) { ... }
    else { ... ]
    ```
  - ```
    switch (Ausdruck) { // Typ char oder int
      case Const_Term_1: ... break;
      case Const_Term_2: ... break;
      case Const_Term_3: ... break;
      ...
      default: ...
    ```
- Bedingungsoperator (Ternärer Operator):
  - `<Bedingung> ? <Anweisung1> : <Anweisung2>`

Logische Aussagen

- Boolesche Operatoren: UND, ODER, NICHT
  - Je nach Fachrichtung gibt es verschiedene Schreibweisen:

		C	Mathe	DigiTech
Konjunktion	UND	`a && b`	$a \wedge b$	a * b
Disjunktion	ODER	`a || b`	$a \vee b$	a + b
Negation	NICHT	`!a`	$\neg a$	$\overline{a}$

- In C und in der Digitaltechnik wird **0** als *false* und **1** als *true* interpretiert
- Auswertungsreihenfolge:
  - UND bindet stärker als ODER
  - NICHT bindet stärker als UND sowie ODER
  - Vergleichsoperatoren binden schwächer als NICHT, aber stärker als UND/ODER
  - Im Zweifel, oder bei anderer Reihenfolge: Klammern setzen!

Regeln

Kommutativgesetze	$a \wedge b = b \wedge a$	$a \vee b = b \vee a$
Assoziativgesetze	$(a \wedge b) \wedge c = a \wedge (b \wedge c)$	$(a \vee b) \vee c = a \vee (b \vee c)$
Idempotenzgesetze	$a \wedge a = a$	$a \vee a = a$
Distributivgesetze	$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$	$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$
Neutralitätsgesetze	$a \wedge 1 = a$	$a \vee 0 = a$
Extremalgesetze	$a \wedge 0 = 0$	$a \vee 1 = 1$
Doppelnegationsgesetz	$\neg(\neg a) = a$	
De Morganschesetze	$\neg(a \wedge b) = \neg a \vee \neg b$	$\neg(a \vee b) = \neg a \wedge \neg b$
Komplementärgesetze	$a \wedge \neg a = 0$	$a \vee \neg a = 1$
Dualitätsgesetze	$\neg 0 = 1$	$\neg 1 = 0$
Absorptionsgesetze	$a \vee (a \wedge b) = a$	$a \wedge (a \vee b) = a$

Übung

- Gegeben ist eine main-Funktion, die drei Integer-Variablen `a`, `b`, `c` einliest, sowie eine Character-Variable `ok` (gültige Werte für `ok` sind `'y'` für yes und `'n'` für no)
- Das Programm soll *Condition true* ausgeben, wenn `a < b < c`. Wenn `ok` den Wert `'y'` hat, dann muss für *Condition true* nur die Bedingung `b < c` gelten. Andernfalls soll *Condition false* ausgegeben werden.

```
if ((_______________) || (___________________) {
  printf("Condition true\n");
} else {
  printf("Condition false\n");
}
```

Übung

- Gegeben ist eine main-Funktion, die ein `int` namens `age` und ein `char` namens `hasTicket` besitzt. Gültige Werte für `hasTicket` sind `'y'` und `'n'` (wie vorhin). Bei anderen Werten soll nur eine Fehlermeldung ausgegeben werden.
- Das Programm soll ausgeben, ob die Person ins Kino darf oder nicht.
- Eine Person darf ins Kino, wenn sie mindestens 16 Jahre alt ist und **und** ein Ticket hat. Außerdem bekommt sie freien Eintritt, wenn sie älter ist als 65 - dann wird kein Ticket benötigt.
- **Wie muss die Fallunterscheidung aussehen?**